Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων &
Μηχανικων Υπολογιστων

Αλγόριθμοι και Πολυπλοκότητα

Συμπληρωματικές Σημειώσεις

Δημήτρης Φωτάκης

Λέκτορας

Οκτώβριος 2010

Περιεχόμενα

1 Εισαγωγικές ΄Εννοιες

Η σύγχρονη υπολογιστική τεχνολογία βασίζεται όλο και περισσότερο στο σχεδιασμό και την εφαρμογή αποδοτικών αλγορίθμων. Η συνολική λειτουργία κάθε υπολογιστικού συστήματος διαιρείται σε στοιχειώδεις λειτουργίες. Κάθε προσπάθεια του χρήστη να αλληλεπιδράσει με το σύστημα επιφέρει την εκτέλεση πολλών στοιχειωδών λειτουργιών, των οποίων τα αποτελέσματα συνθέτουν την επιθυμητή απόκριση. Η αποδοτική υλοποίηση κάθε στοιχειώδους λειτουργίας καθορίζει την απόδοση του συστήματος.

΄Ενα σχετικά μικρό υποσύνολο στοιχειωδών υπολογιστικών λειτουργιών καταναλώνει τη συντριπτική πλειοψηφία του χρόνου επεξεργασίας κάθε σύγχρονου υπολογιστικού συστήματος. Αυτό σημαίνει ότι η συνεχής επιτάχυνση του υλικού των υπολογιστών δεν είναι ο μόνος, και ίσως ούτε ο πλέον αποδοτικός, τρόπος για να κάνουμε τα υπολογιστικά συστήματα γρηγορότερα. Η έστω και μικρή επιτάχυνση της εκτέλεσης μερικών στοιχειωδών υπολογιστικών λειτουργιών θα γινόταν αντιληπτή από τους χρήστες σαν σημαντική επιτάχυνση του συστήματος. Για παράδειγμα, ας προσπαθήσουμε να αναλογιστούμε τη δραματική επίδραση που θα είχε στην απόδοση ενός υπολογιστικού συστήματος ένας γρηγορότερος αλγόριθμος πολλαπλασιασμού αριθμού ή πινάκων, ταχύτεροι αλγόριθμοι για την ταξινόμηση, την αναζήτηση, και γενικότερα την οργάνωση των στοιχείων στις Βάσεις Δεδομένων, ή ένα γρηγορότερο σύστημα αρχείων. Αυτή με απλά λόγια είναι η τεχνολογική διάσταση της Θεωρίας Αλγορίθμων.

Η *Θεωρία Αλγορίθμων* κωδικοποιεί τις σημαντικότερες στοιχειώδεις υπολογιστικές λειτουργίες σε *υπολογιστικά προβλήματα*, δηλαδή σε πρότυπα μαθηματικά μοντέλα που συγκεντρώνουν όλα τα ουσιώδη χαρακτηριστικά του πραγματικού προβλήματος. Η Θεωρία Αλγορίθμων μας εφοδιάζει με αποδοτικούς αλγόριθμους για την επίλυση των σημαντικότερων υπολογιστικών προβλημάτων και με ένα σύνολο αρχών και μεθόδων που μπορούν να εφαρμοστούν σε κάθε πρόβλημα.

Αυτές οι σημειώσεις εστιάζουν στις αλγοριθμικές ιδέες που χρησιμοποιούνται για την αποδοτική επίλυση προβλημάτων *συνδυαστικής βελτιστοποίησης*. Τα προβλήματα συνδυαστικής βελτιστοποίησης αποτελούν μια από τις σημαντικότερες κατηγορίες προβλημάτων που μελετά η Θεωρία Αλγορίθμων και γενικότερα η Επιστήμη των Υπολογιστών και έχουν πολλές και εξαιρετικά σημαντικές πρακτικές εφαρμογές.

Από την άλλη πλευρά, παρά τις σημαντικές προόδους της Θεωρίας Αλγορίθμων και τις επίμονες και μακροχρόνιες προσπάθειες πλήθους ικανών επιστημόνων, υπάρχουν πολλά σημαντικά προβλήματα για τα οποία δεν έχουμε κατορθώσει να σχεδιάσουμε αποδοτικούς αλγόριθμους. Η *Θεωρία Υπολογιστικής Πολυπλοκότητας* αναπτύχθηκε για να μελετήσει τους λόγους για τους οποίους κάποια υπολογιστικά προβλήματα είναι δύσκολο ή αδύνατο να λυθούν από έναν Υπολογιστή.

Η Θεωρία Υπολογιστικής Πολυπλοκότητας μελετά διαφορετικά υπολογιστικά μοντέλα και την επίδραση που έχουν στη δυνατότητα των Υπολογιστών να επιλύσουν ένα πρόβλημα αποδοτικά. Για κάθε υπολογιστικό μοντέλο, η Θεωρία Υπολογιστικής Πολυπλοκότητας εξετάζει αν ένα υπολογιστικό πρόβλημα μπορεί να επιλυθεί ή όχι. Αν το πρόβλημα είναι επιλύσιμο, η Θεωρία Πολυπλοκότητας καθορίζει την ελάχιστη ποσότητα υπολογιστικών πόρων που απαιτούνται για την επίλυση του προβλήματος στο συγκεκριμένο μοντέλο. Με βάση τις απαντήσεις στα παραπάνω ερωτήματα, τα υπολογιστικά προβλήματα εντάσσονται σε *κλάσεις πολυπλοκότητας*. Κάθε κλάση πολυπλοκότητας αποτελείται από προβλήματα που εμφανίζουν παρόμοια συμπεριφορά ως προς την επιλυσιμότητα τους σε συγκεκριμένο υπολογιστικό μοντέλο.

Η Θεωρία Υπολογιστικής Πολυπλοκότητας μας εφοδιάζει με έναν ικανό αριθμό κλάσεων πολυπλοκότητας στις οποίες έχουν ενταχθεί τα σημαντικότερα υπολογιστικά προβλήματα. Επιπλέον, μας εφοδιάζει με τεχνικές και μεθόδους για την ένταξη νέων υπολογιστικών προβλημάτων στις υπάρχουσες κλάσεις πολυπλοκότητας. Έχοντας εντάξει ένα πρόβλημα σε μία κλάση πολυπλοκότητας, γνωρίζουμε κατά πόσον υπάρχει αποδοτικός αλγόριθμος για την επίλυση αυτού του προβλήματος. Μία από τις σημαντικότερες εφαρμογές της Υπολογιστικής Πολυπλοκότητας είναι ότι μας επιτρέπει να αποφύγουμε τη μάταια αναζήτηση αποδοτικών αλγόριθμων για προβλήματα για τα οποία εικάζεται ή έχει αποδειχθεί ότι τέτοιοι αλγόριθμοι δεν υπάρχουν.

Σε αυτές τις σημειώσεις γίνεται μια πολύ σύντομη εισαγωγή στη Θεωρία Υπολογιστικής Πολυπλοκότητας. Το ενδιαφέρον εστιάζεται στη διάκριση των προβλημάτων συνδυαστικής βελτιστοποίησης σε αυτά που επιλύονται σε πολυωνυμικό χρόνο και θεωρούνται *πρακτικώς επιλύσιμα* (tractable), και σε αυτά που είναι πλήρη για την κλάση **NP**. Για αυτή την κατηγορία προβλημάτων εικάζεται ότι υπάρχουν μόνο αλγόριθμοι εκθετικού χρόνου και θεωρούνται *πρακτικώς δισεπίλυτα* (intractable).

1.1 Αλγόριθμος, Υπολογιστικό Πρόβλημα, και Στιγμιότυπο

Αλγόριθμος. Με απλά λόγια, οι αλγόριθμοι είναι οι "συνταγές" που ακολουθούν οι υπολογιστές για να λύνουν προβλήματα. Πιο συγκεκριμένα, αλγόριθμος είναι μια διαδικασία ή ένα σύνολο κανόνων με σκοπό το μηχανιστικό υπολογισμό. Η εκτέλεση ενός αλγορίθμου δεν πρέπει να απαιτεί τη χρήση της διαίσθησης ή της δημιουργικότητας[1]. Ένας αλγόριθμος αποτελεί το "εργαλείο" για την επίλυση ενός *υπολογιστικού προβλήματος*. Ο ορισμός του υπολογιστικού προβλήματος καθορίζει την σχέση μεταξύ των *δεδομένων εισόδου* (input) και των *δεδομένων εξόδου* (output) και ο αλγόριθμος πραγματοποιεί τον επιθυμητό μετασχηματισμό. Με βάση τα παραπάνω, θα λέμε ότι *αλγόριθμος* είναι κάθε καλώς ορισμένη, πεπερασμένη υπολογιστική διαδικασία για την επίλυση ενός υπολογιστικού προβλήματος.

Περίπου στο 1930, διάφοροι μαθηματικοί ορισμοί της έννοιας μιας "πεπερασμένης υπολογιστικής διαδικασίας" εδόθησαν από τους Church, Godel, Herbrand, Kleene, Post, Turing, και άλλους. Οι ορισμοί αυτοί είχαν ως βάση μεθόδους που επιφανειακά εμφανίζονταν διαφορετικές μεταξύ τους (π.χ. τυπικά συστήματα επεξεργασίας συμβόλων, ιδανικά μοντέλα υπολογιστικών μηχανών). Όμως, όλοι αυτοί οι ορισμοί καταλήγουν να είναι ισοδύναμοι, με την έννοια ότι όλες οι "μορφές" αλγορίθμων υπολογίζουν την ίδια κατηγορία συναρτήσεων, συγκεκριμένα την κλάση

[1]Πάντως τα όρια μεταξύ αλγορίθμων και ανθρώπινων σχεδίων προς εκτέλεση γίνονται όλο και περισσότερο δυσδιάκριτα, βλέπε για παράδειγμα τυχαιότητα στους αλγορίθμους, fuzzy λογική, κλπ.

των των μερικώς αναδρομικών συναρτήσεων (partial recursive functions). Μέχρι τώρα, και παρά τις προόδους της τεχνολογίας, η εμπειρίας μας καταδεικνύει ότι η αυτή η κλάση των συναρτήσεων είναι ακριβώς ότι μπορεί να υπολογισθεί συστηματικά (από υπολογιστές, προγράμματα, ή πεπερασμένες διαδικασίες).

Τυπικά παραδείγματα αλγορίθμων είναι οι μέθοδοι πρόσθεσης, αφαίρεσης, πολλαπλασιασμού και διαίρεσης αριθμών που μαθαίνουμε στο σχολείο. Ένα άλλο παράδειγμα είναι ο αλγόριθμος του Ευκλείδη για τον υπολογισμό του Μέγιστου Κοινού Διαιρέτη δύο ακεραίων αριθμών.

Αναπαράσταση Αλγορίθμων. Εκτός από τη φυσική γλώσσα, για την αναπαράσταση των αλγορίθμων θα χρησιμοποιήσουμε μια μορφή δομημένου ψευδοκώδικα που θυμίζει τις γλώσσες C και Pascal.

Όλες οι εντολές έχουν την ίδια σημασία με τις αντίστοιχες εντολές στις γλώσσες C και Pascal, όμως ο ψευδοκώδικας δεν έχει την αυστηρότητα που απαιτούν οι συνηθισμένες γλώσσες προγραμματισμού (π.χ. απουσία δηλώσεων μεταβλητών, χρήση ρουτινών που δεν έχουν οριστεί, περιστασιακή χρήση φυσικής γλώσσας αντί για εντολές, κλπ.). Κάθε σύνολο εντολών με κοινή στοίχιση θεωρείται σαν ομάδα (block) εντολών που (θα έπρεπε να) βρίσκεται ανάμεσα σε αγκύλες $\{\ldots\}$ στη C (ή σε begin ... end στην Pascal).

Ο ψευδοκώδικας χρησιμοποιεί τη συνάρτηση σαν βασική δομή προγράμματος και υποστηρίζει αναδρομή. Χρησιμοποιούμε τους πίνακες (arrays), τις δομές (structures ή εγγραφές - records στην Pascal) και περιστασιακά τα σύνολα (sets) για την οργάνωση των δεδομένων. Θεωρούμε επίσης δεδομένες τις βασικές δομές δεδομένων (π.χ. διασυνδεδεμένες λίστες, πίνακες, ουρές προτεραιότητας, δέντρα αναζήτησης) και βασικούς αλγόριθμους (π.χ. αλγόριθμοι ταξινόμησης και αναζήτησης, αλγόριθμοι διαχείρισης ξένων συνόλων) που διδάχθηκαν στο προηγούμενο εξάμηνο.

Υπολογιστικό Πρόβλημα. Ένα *υπολογιστικό πρόβλημα* (computational problem) ορίζει έναν επιθυμητό μετασχηματισμό ενός συνόλου δεδομένων εισόδου (input) σε ένα σύνολο δεδομένων εξόδου (output). Ο ορισμός του υπολογιστικού προβλήματος ορίζει τόσο τη μορφή όσο και τους περιορισμούς που πρέπει να ικανοποιούν τα δεδομένα εισόδου και τα δεδομένα εξόδου.

Για παράδειγμα, στο πρόβλημα του πολλαπλασιασμού δύο αριθμών, τα δεδομένα εισόδου είναι δύο αριθμοί (x, y), και τα δεδομένα εξόδου είναι το γινόμενό τους $x \times y$. Στο πρόβλημα της ταξινόμησης n ακεραίων αριθμών σε αύξουσα σειρά, τα δεδομένα εισόδου είναι μία ακολουθία $X = (x_1, x_2, \ldots, x_n)$ από n ακέραιους αριθμούς. Τα δεδομένα εξόδου είναι μια μετάθεση αυτών των αριθμών $X' = (x'_1, x'_2, \ldots, x'_n)$ ώστε κανένας αριθμός να μην είναι μεγαλύτερος από τον επόμενό του, δηλαδή $x'_1 \leq x'_2 \leq \cdots \leq x'_n$.

Στιγμιότυπο και Λύση. Κάθε σύνολο δεδομένων εισόδου που συμφωνεί με τους περιορισμούς που θέτει ο ορισμός ενός υπολογιστικού προβλήματος αποτελεί ένα *στιγμιότυπο* (instance) του προβλήματος. Για παράδειγμα, οι αριθμοί $(15, 32)$ αποτελούν ένα στιγμιότυπο για το πρόβλημα του πολλαπλασιασμού δύο αριθμών. Η ακολουθία $(8, 1, 7, 5, 3, 6, 2, 4)$ αποτελεί ένα στιγμιότυπο για το πρόβλημα της ταξινόμησης ακεραίων αριθμών.

Κάθε στιγμιότυπο ενός προβλήματος έχει καμία, μία ή περισσότερες *λύσεις* (solutions). Η λύση ενός στιγμιότυπου για ένα πρόβλημα είναι κάθε σύνολο δεδομένων εξόδου που σε συνδυασμό με το στιγμιότυπο ικανοποιούν τους περιορισμούς που θέτει ο ορισμός του προβλήματος. Για παράδειγμα το 480 αποτελεί τη λύση του στιγμιότυπου $(15, 32)$ για το πρόβλημα του πολλαπλα-

σιασμού. Η ακολουθία $(1, 2, 3, 4, 5, 6, 7, 8)$ αποτελεί τη λύση του στιγμιότυπου $(8, 1, 7, 5, 3, 6, 2, 4)$ για το πρόβλημα της ταξινόμησης.

Ορθότητα Αλγορίθμων. Τα προβλήματα που θα μελετήσουμε έχουν άπειρο αριθμό στιγμιοτύπων. Για να θεωρηθεί ένας αλγόριθμος *σωστός* (ή *ορθός* - correct), πρέπει να υπολογίζει σωστά τις λύσεις για όλα τα στιγμιότυπα του προβλήματος που λύνει. Για να τεκμηριώσουμε ότι την ορθότητα ενός αλγόριθμου για κάποιο πρόβλημα πρέπει να αποδείξουμε (με τυπική μαθηματική απόδειξη) ότι πράγματι ο αλγόριθμος υπολογίζει μία λύση για κάθε στιγμιότυπο του προβλήματος. Για να τεκμηριώσουμε τη μη-ορθότητα ενός αλγόριθμου για κάποιο πρόβλημα, αρκεί να περιγράψουμε ένα στιγμιότυπο του προβλήματος για το οποίο ο αλγόριθμος δεν υπολογίζει λύση.

1.2 Προβλήματα Συνδυαστικής Βελτιστοποίησης

Ένα παράδειγμα υπολογιστικού προβλήματος στο οποίο θα αναφερθούμε αρκετές φορές είναι το *Πρόβλημα του Σακιδίου* (Knapsack Problem). Στο Πρόβλημα του Σακιδίου, δίνεται ένα σακίδιο μεγέθους $B \geq 0$ και n αντικείμενα, καθένα από τα οποία έχει μέγεθος $s_i \geq 0$ και αξία $p_i \geq 0$, $i = 1, 2, \ldots, n$. Το ζητούμενο είναι να υπολογιστεί μία συλλογή αντικειμένων που χωρά στο σακίδιο και έχει *μέγιστη* αξία. Το Πρόβλημα του Σακιδίου μοντελοποιεί πληθώρα πρακτικών προβλημάτων και έχει πολλές και σημαντικές πρακτικές εφαρμογές.

Υπάρχουν δύο βασικές εκδοχές για το Πρόβλημα του Σακιδίου: η *ακέραια* (ή $0-1$) εκδοχή και η *κλασματική* εκδοχή. Στην κλασματική εκδοχή του προβλήματος, τα αντικείμενα μπορούν να διαιρεθούν και να συμπεριληφθούν στο σακίδιο τμηματικά. Αν το κλάσμα του αντικειμένου i που έχει συμπεριληφθεί στο σακίδιο είναι f_i, $f_i \in [0, 1]$, αυτό καταλαμβάνει χώρο $f_i s_i$ και προσθέτει αξία $f_i p_i$. Στο Κλασματικό Πρόβλημα του Σακιδίου (Fractional Knapsack Problem), πρέπει να προσδιοριστεί το κλάσμα κάθε αντικειμένου που θα συμπεριληφθεί στο σακίδιο ώστε να μην παραβιάζεται το μέγεθος του σακιδίου και να μεγιστοποιείται η συνολική αξία. Τυπικά, ζητείται η μεγιστοποίηση του $\sum_{i=1}^{n} f_i p_i$, ώστε $\sum_{i=1}^{n} f_i s_i \leq B$ και $f_i \in [0, 1]$ για κάθε $i = 1, \ldots, n$.

Στο Ακέραιο Πρόβλημα του Σακιδίου (ή απλά Πρόβλημα του Σακιδίου), τα αντικείμενα θεωρούνται αδιαίρετα. Έτσι κάθε αντικείμενο είτε συμπεριλαμβάνεται ολόκληρο στο σακίδιο είτε δεν συμπεριλαμβάνεται καθόλου. Τυπικά, κάθε f_i μπορεί να είναι είτε 1 είτε 0 (ενώ στην κλασματική εκδοχή, κάθε f_i μπορούσε να πάρει οποιαδήποτε τιμή μεταξύ 0 και 1). Το ζητούμενο είναι να προσδιοριστεί ένα μέγιστης αξίας υποσύνολο αντικειμένων που χωρά στο σακίδιο. Τυπικά, ζητείται η μεγιστοποίηση του $\sum_{i=1}^{n} f_i p_i$, ώστε $\sum_{i=1}^{n} f_i s_i \leq B$ και $f_i \in \{0, 1\}$ για κάθε $i = 1, \ldots, n$.

Στο Πρόβλημα του Σακιδίου, τα δεδομένα εισόδου είναι το μέγεθος του σακιδίου B και τα διατεταγμένα ζεύγη $(s_1, p_1), (s_2, p_2), \ldots, (s_n, p_n)$ που προσδιορίζουν το μέγεθος και την αξία κάθε αντικειμένο. Κάθε σύνολο δεδομένων εισόδου στη μορφή $(B, (s_1, p_1), (s_2, p_2), \ldots, (s_n, p_n))$ ορίζει ένα στιγμιότυπο. Τα δεδομένα εξόδου είναι n αριθμοί f_i που ορίζουν ποια αντικείμενα συμπεριλαμβάνονται στο σακίδιο. Για την κλασματική εκδοχή πρέπει που $f_i \in [0, 1]$, και για την ακέραια εκδοχή πρέπει $f_i \in \{0, 1\}$.

Τα δεδομένα εξόδου αποτελούν μία *αποδεκτή λύση* (feasible solution) για το Πρόβλημα του Σακιδίου όταν τα αντικείμενα που έχουν συμπεριληφθεί χωρούν στο σακίδιο. Μια αποδεκτή λύση είναι *βέλτιστη λύση* (optimal solution), ή απλά *λύση*, όταν η αξία των αντικειμένων που έχουν συμπεριληφθεί στο σακίδιο δεν είναι μικρότερη από την αξία των αντικειμένων κάποιας άλλης

αποδεκτής λύσης.

Στην κλασματική εκδοχή, κάθε στιγμιότυπο έχει άπειρες αποδεκτές λύσεις. Στην ακέραια εκδοχή, ο αριθμός των εφικτών λύσεων είναι πεπερασμένος. Συγκεκριμένα, κάθε στιγμιότυπο έχει τουλάχιστον μία (το κενό σύνολο) και το πολύ 2^n αποδεκτές λύσεις (όλα τα διαφορετικά υποσύνολα n αντικειμένων). Και στις δύο εκδοχές, κάθε στιγμιότυπο έχει μία ή περισσότερες βέλτιστες λύσεις. Όμως στην κλασματική εκδοχή, οι βέλτιστες λύσεις μπορεί να είναι άπειρες, ενώ στην ακέραια εκδοχή ο αριθμός των βέλτιστων λύσεων είναι πεπερασμένος (αφού πεπερασμένος είναι ο αριθμός των εφικτών λύσεων).

Παράδειγμα 1.1. Το $(30, ((20, 5), (10, 4), (15, 10), (40, 10)))$ αποτελεί ένα στιγμιότυπο για το Πρόβλημα του Σακιδίου όπου το σακίδιο έχει μέγεθος 30 και υπάρχουν τέσσερα αντικείμενα με μεγέθη 20, 10, 15, και 40, και αξίες 5, 4, 10, και 10, αντίστοιχα. Οι τετράδες $(1/2, 1, 1/3, 1/8), (1, 1, 0, 0)$, και $(0, 1, 0, 1/2)$ είναι κάποιες από τις (άπειρες) αποδεκτές λύσεις για το συγκεκριμένο στιγμιότυπο και την κλασματική εκδοχή. Το συγκεκριμένο στιγμιότυπο έχει επίσης άπειρες βέλτιστες λύσεις για την κλασματική εκδοχή. Κάθε τετράδα της μορφής $(1/4 - x, 1, 1, x/2)$, όπου $x \in [0, 1/4]$, αποτελεί μία βέλτιστη λύση συμπεριλαμβάνοντας αντικείμενα συνολικής αξίας 15.25. Για την ακέραια εκδοχή, το συγκεκριμένο στιγμιότυπο έχει μόλις έξι αποδεκτές (ποιές;) και μία βέλτιστη λύση, την $(0, 1, 1, 0)$ με συνολική αξία 14.

Από την άλλη πλευρά, το στιγμιότυπο $(20, ((5, 2), (7, 10), (4, 9)))$ έχει μόνο μία βέλτιστη λύση κοινή και για τις δύο εκδοχές, την τετράδα $(1, 1, 1)$. Ακόμη το $(17, ((6, 6), (11, 10), (9, 8), (8, 8)))$ έχει δύο βέλτιστες λύσεις για την ακέραιη εκδοχή, τις (1, 1, 0, 0) και (0, 0, 1, 1) με αξία 16, και μόνο μία βέλτιστη λύση για την κλασματική εκδοχή, την $(1, 3/11, 0, 1)$ με αξία $16\frac{8}{11}$. Παρατηρούμε επίσης ότι μια βέλτιστη λύση για την κλασματική εκδοχή έχει αξία μεγαλύτερη ή ίση της αξίας της βέλτιστης λύσης για την ακέραιη εκδοχή. Ο λόγος είναι ότι η βέλτιστη λύση για την ακέραιη εκδοχή αποτελεί μία εφικτή λύση για την κλασματική εκδοχή, ενώ το αντίστροφο δεν ισχύει. □

Το Πρόβλημα του Σακιδίου αποτελεί ένα τυπικό παράδειγμα *προβλήματος βελτιστοποίησης* (optimization problem). Ένα πρόβλημα συνδυαστικής βελτιστοποίησης έχει πολλές *αποδεκτές λύσεις*, σε καθεμία από τις οποίες αντιστοιχεί μία *αντικειμενική τιμή* (objective value). Το ζητούμενο είναι ο υπολογισμός μιας *βέλτιστης λύσης*, δηλαδή μιας αποδεκτής λύσης με τη *βέλτιστη* (μέγιστη ή ελάχιστη) αντικειμενική τιμή. Συχνά δεν ζητείται μια βέλτιστη λύση, αλλά απλώς η βέλτιστη αντικειμενική τιμή.

Ορισμός 1.1 (Πρόβλημα Βελτιστοποίησης). Ένα *πρόβλημα βελτιστοποίησης* Π χαρακτηρίζεται από τα ακόλουθα:

- Ένα σύνολο στιγμιοτύπων Σ_Π.
- Για κάθε στιγμιότυπο $\sigma \in \Sigma_\Pi$, ένα σύνολο αποδεκτών λύσεων $\Lambda_\Pi(\sigma)$.
- Για κάθε στιγμιότυπο εισόδου $\sigma \in \Sigma_\Pi$, μια αντικειμενική συνάρτηση $f_\sigma : \Lambda_\Pi(\sigma) \mapsto \mathbb{R}$ που αντιστοιχεί μια αντικειμενική τιμή σε κάθε εφικτή λύση.

Δεδομένου στιγμιότυπου $\sigma \in \Sigma_\Pi$, ζητείται να υπολογισθεί αποδεκτή λύση $\lambda^*_\sigma \in \Lambda_\Pi(\sigma)$:

$$\forall\, \lambda \in \Lambda_\Pi(\sigma),\ \ f_\sigma(\lambda^*_\sigma) \geq f_\sigma(\lambda) \quad \text{όταν πρόκειται για πρόβλημα } \textit{μεγιστοποίησης}$$

ή $\forall\, \lambda \in \Lambda_\Pi(\sigma),\ \ f_\sigma(\lambda^*_\sigma) \leq f_\sigma(\lambda)$ όταν πρόκειται για πρόβλημα *ελαχιστοποίησης*

Η λύση λ^*_σ ονομάζεται *βέλτιστη λύση* και η τιμή της $f_\sigma(\lambda^*_\sigma)$ *βέλτιστη αντικειμενική τιμή*.

Μία ιδιαίτερα σημαντική κατηγορία προβλημάτων βελτιστοποίησης είναι τα *προβλήματα συνδυαστικής βελτιστοποίησης* (combinatorial optimization problems). Διαισθητικά, ένα πρόβλημα συνδυαστικής βελτιστοποίησης χαρακτηρίζεται από την ιδιότητα ότι υπάρχει ένα πεπερασμένο σύνολο αποδεκτών λύσεων που συμπεριλαμβάνει μια βέλτιστη λύση αν και μόνο αν μια τέτοια λύση υπάρχει[2]. Κάθε πρόβλημα βελτιστοποίησης με πεπερασμένο αριθμό αποδεκτών λύσεων είναι πρόβλημα συνδυαστικής βελτιστοποίησης.

Παράδειγμα 1.2. Στην περίπτωση του Προβλήματος του Σακιδίου, η αντικειμενική τιμή μιας αποδεκτής λύσης είναι η συνολική αξία των αντικειμένων που συμπεριλαμβάνονται στο σακίδιο. Το ζητούμενο είναι να υπολογισθεί η αποδεκτή λύση με τη μέγιστη αντικειμενική τιμή. Πρόκειται δηλαδή για πρόβλημα μεγιστοποίησης. Η ακέραια εκδοχή του Προβήματος του Σακιδίου είναι ένα πρόβλημα συνδυαστικής βελτιστοποίησης, αφού ένα στιγμιότυπο με n αντικείμενα έχει το πολύ 2^n αποδεκτές λύσεις. □

Ένα τυπικό παράδειγμα προβλήματος ελαχιστοποίησης είναι το Πρόβλημα του Συντομότερου Μονοπατιού (Shortest Path Problem): Δίνεται ένα γράφημα με βάρη / μήκη στις ακμές και δύο διακεκριμένες κορυφές s και t, και ζητείται το μονοπάτι ελάχιστου συνολικού μήκους από την κορυφή s στην κορυφή t.

Ένα άλλο παράδειγμα προβλήματος συνδυαστικής βελτιστοποίησης είναι το *Πρόβλημα του Περιοδεύοντος Πωλητή* (Travelling Salesman Problem − TSP): Δίνονται n σημεία και για κάθε ζευγάρι σημείων $i, j \in \{1, \ldots, n\}$, οι αποστάσεις d_{ij} και d_{ji} για τη μετάβαση από το i στο j και από το j στο i αντίστοιχα (πάντα θα υποθέτουμε ότι οι αποστάσεις είναι μη-αρνητικές και ότι η απόσταση κάθε σημείου από τον εαυτό του είναι 0). Το ζητούμενο είναι μία *περιοδεία* (tour), δηλαδή ένας κύκλος που διέρχεται από όλα τα σημεία ακριβώς μία φορά, ελάχιστου συνολικού μήκους.

Το Πρόβλημα του Περιοδεύοντος Πωλητή είναι ένα πρόβλημα ελαχιστοποίησης. Όταν οι αποστάσεις είναι συμμετρικές, δηλαδή για κάθε ζευγάρι σημείων $i, j \in \{1, \ldots, n\}$ είναι $d_{ij} = d_{ji}$, έχουμε το Συμμετρικό Πρόβλημα του Περιοδεύοντος Πωλητή. Όταν επιπλέον οι αποστάσεις ικανοποιούν την τριγωνική ανιστότητα τότε τα σημεία και οι αποστάσεις τους συγκροτούν ένα *μετρικό χώρο*[3] (metric space) και έχουμε το Μετρικό Πρόβλημα του Περιοδεύοντος Πωλητή, που είναι από τις πιο σημαντικές περιπτώσεις στην πράξη.

[2]Είναι ενδεχόμενο να υπάρχουν άπειρες αποδεκτές λύσεις και αυτές που θα συμπεριληφθούν στο αναφερόμενο πεπερασμένο σύνολο να χρειάζεται να επιλεγούν κατάλληλα.

[3]Ένα σύνολο σημείων N με μια μη-αρνητική συνάρτηση απόστασης $d : N \times N \mapsto \mathbb{R}_+$ συγκροτούν ένα μετρικό χώρο όταν η συνάρτηση απόστασης ικανοποιεί τις παρακάτω ιδιότητες: (α) οι αποστάσεις είναι συμμετρικές, δηλαδή $\forall i, j \in N, d(i, j) = d(j, i)$, (β) η απόσταση κάθε σημείο από τον εαυτό του είναι 0, δηλαδή $\forall i \in N,\ d(i, i) = 0$, και (γ) οι αποστάσεις ικανοποιούν την τριγωνική ανισότητα, δηλαδή για κάθε τριάδα σημείων $i, j, k \in N$, $d(i, j) \leq d(i, k) + d(k, j)$.

1.3 Ανάλυση Αλγορίθμων

Για να συγκρίνουμε διαφορετικούς αλγόριθμους που λύνουν το ίδιο πρόβλημα απαιτείται να προσδιορίσουμε την ποσότητα των υπολογιστικών πόρων που απαιτούνται για την εκτέλεση του αλγόριθμου (π.χ. υπολογιστικός χρόνος, θέσεις μνήμης, αριθμός επεξεργαστών, επικοινωνία μέσω δικτύου, κλπ.). Η διαδικασία της μαθηματικής εκτίμησης των υπολογιστικών πόρων που απαιτεί η εκτέλεση ενός αλγόριθμου ονομάζεται *ανάλυση του αλγόριθμου*.

Για να αναλύσουμε έναν αλγόριθμο, πρέπει να θεωρήσουμε ένα μοντέλο του υπολογιστή στον οποίο θα εκτελεστεί ο αλγόριθμος. Για τους σκοπούς του μαθήματος, θεωρούμε ότι κάθε αλγόριθμος θα υλοποιηθεί σε μία Υπολογιστική Μηχανή Άμεσης Προσπέλασης Μνήμης (Random Access Machine, RAM) με έναν επεξεργαστή. Στη Μηχανή Άμεσης Προσπέλασης Μνήμης, οι εντολές εκτελούνται ακολουθιακά και κάθε στοιχειώδες υπολογιστικό βήμα έχει μοναδιαίο κόστος (για όλα τα είδη υπολογιστικών πόρων). Σαν στοιχειώδη υπολογιστικά βήματα θεωρούνται η προσπέλαση μιας θέσης μνήμης για ανάγνωση ή εγγραφή, οι βασικές εντολές (π.χ. ανάθεση τιμής σε μεταβλητή, συγκρίσεις αριθμών ή χαρακτήρων, εντολές ελέγχου ροής, κλπ.), και οι βασικές αριθμητικές και λογικές πράξεις (π.χ. πρόσθεση, πολλαπλασιασμός, λογικό και, λογικό ή, κλπ.) εφόσον αφορούν αντικείμενα που μπορούν να αποθηκευτούν σε μικρό αριθμό θέσεων μνήμης. Το μοντέλο της Μηχανή Άμεσης Προσπέλασης Μνήμης είναι πολύ κοντά στα σύγχρονα μονοεπεξεργαστικά υπολογιστικά συστήματα.

Η πιο συνηθισμένη κατηγορία ανάλυσης ενός αλγορίθμου αφορά το χρόνο εκτέλεσης. Ο χρόνος εκτέλεσης ενός αλγορίθμου προσδιορίζεται από τον αριθμό των στοιχειωδών βημάτων που εκτελούνται. Το είδος των στοιχειωδών βημάτων τα οποία προσμετρώνται ποικίλει ανάλογα με το πρόβλημα (π.χ. για τα προβλήματα ταξινόμησης και αναζήτησης, ο χρόνος εκτέλεσης προσδιορίζεται από τις συγκρίσεις μεταξύ στοιχείων της συλλογής, για τα προβλήματα αριθμητικών πράξεων, ο χρόνος εκτέλεσης προσδιορίζεται από τις αριθμητικές πράξεις μεταξύ μονοψήφιων αριθμών, κλπ.) Για τα προβλήματα συνδυαστικής βελτιστοποίησης, ο χρόνος εκτέλεσης προσδιορίζεται από τον αριθμό των απλών αριθμητικών και λογικών πράξεων, των συγκρίσεων, και των εντολών εκχώρησης που εκτελούνται κατά τη διάρκεια του αλγόριθμου. Υποθέτουμε ότι η εκτέλεση αυτών των εντολών αντιστοιχεί σε ένα στοιχειώδες υπολογιστικό βήμα μίας Μηχανής Άμεσης Προσπέλασης Μνήμης.

Είναι εύλογο ότι η ποσότητα των υπολογιστικών πόρων εξαρτάται άμεσα από το μέγεθος του στιγμιότυπου που πρέπει να επιλυθεί. Για παράδειγμα, ο υπολογιστικός χρόνος για την επίλυση ενός στιγμιότυπου του Προβλήματος του Σακιδίου με δέκα αντικείμενα δεν μπορεί να είναι συγκρίσιμος με τον υπολογιστικό χρόνο για την επίλυση ενός στιγμιότυπου με εκατό, χίλια ή ένα εκατομμύριο αντικείμενα.

Τα αποτελέσματα της ανάλυσης ενός αλγόριθμου εκφράζονται σαν συνάρτηση του μεγέθους του στιγμιότυπου που επιλύεται. Το *μεγέθος ενός στιγμιότυπου* είναι ο αριθμός των δυαδικών ψηφίων (bits) που απαιτούνται για να αναπαρασταθεί το στιγμιότυπο στη μνήμη του υπολογιστή. Για να διευκολυνθούμε, συνήθως υποθέτουμε ότι κάθε "βασική συνιστώσα" του στιγμιότυπου χρειάζεται περίπου τον ίδιο αριθμό δυαδικών ψηφίων για να αποθηκευθεί στη μνήμη του υπολογιστή και χρησιμοποιούμε τον αριθμό των "βασικών συνιστωσών" σαν μέτρο του μεγέθους του. Για παράδειγμα, στο Πρόβλημα του Σακιδίου, βασικές συνιστώσες είναι τα αντικείμενα. Έτσι το μέγεθος ενός στιγμιότυπου μπορεί να θεωρηθεί ανάλογο του αριθμού των αντικειμένων του.

Αντίστοιχα, οι βασικές συνιστώσες στο Πρόβλημα του Περιοδεύοντος Πωλητή είναι τα σημεία και στο Πρόβλημα της Ταξινόμησης είναι οι αριθμοί που θα ταξινομηθούν.

Επισήμανση. Η παραπάνω υπόθεση σε σχέση με το μέγεθος ενός στιγμιότυπου είναι βάσιμη μόνο όταν η κάθε "βασική συνιστώσα" χρειάζεται περίπου τον ίδιο αριθμό δυαδικών ψηφίων για να αποθηκευθεί στη μνήμη του υπολογιστή. Για παράδειγμα, είναι εύλογο να δεχθούμε ότι τα αντικείμενα $(10, 5)$ και $(100, 30)$ συνεισφέρουν το ίδιο στο μέγεθος ενός στιγμιότυπου του Προβλήματος του Σακιδίου. Η ίδια παραδοχή δεν μπορεί να γίνει φυσικά για τα αντικείμενα $(1, 1)$ και $(2^{2004}, 2^{2005})$. □

1.3.1 Αρχή του Αναλλοίωτου και Ασυμπτωτική Εκτίμηση

Έχοντας καταλήξει σε ένα υπολογιστικό μοντέλο για την ανάλυση των αλγορίθμων και σε ένα μέτρο του μεγέθους των στιγμιότυπων ενός προβλήματος, ανακύπτει το ερώτημα ποια είναι η υλοποίηση του αλγόριθμου για την οποία θα γίνει η ανάλυση.

Η *αρχή του αναλλοίωτου* (principle of invariance) απαντάει ότι ουσιαστικά αυτό είναι αδιάφορο. Η αρχή του αναλλοίωτου λέει ότι δύο διαφορετικές υλοποιήσεις του ίδιου αλγόριθμου δεν διαφέρουν σε αποτελεσματικότητα παρά μόνο κατά μια σταθερή αναλογία (π.χ. αυτή εξαρτάται από την τεχνολογία του υλικού και του λογισμικού του υπολογιστικού συστήματος στο οποίο θα εκτελεστεί ο αλγόριθμος). Συνεπώς, οι πολλαπλασιαστικές σταθερές δεν παίζουν ιδιαίτερα σημαντικό ρόλο στη *θεωρητική εκτίμηση* της αποδοτικότητας των αλγορίθμων και μπορούν να αγνοηθούν (δεν πρέπει να αγνοούνται όμως στην πράξη).

Ορισμός 1.2 (Αρχή του Αναλλοίωτου). Ένας αλγόριθμος απαιτεί χρόνο της τάξεως $T(n)$ για να εκτελεσθεί, όπου n είναι το μέγεθος του στιγμιότυπου προς επίλυση, εάν υπάρχει θετική σταθερά c και υλοποίηση του αλγορίθμου ικανή να λύσει κάθε στιγμιότυπο μεγέθους n σε χρόνο όχι περισσότερο από $cT(n)$ βήματα.

Με χρήση της αρχής του αναλλοίωτου, κάθε άλλη υλοποίηση του ίδιου αλγορίθμου θα έχει τον ίδιο χρόνο εκτέλεσης (με ενδεχόμενη αλλαγή της σταθεράς c). Αυτή η ιδιαίτερα σημαντική έννοια είναι γνωστή ως *ασυμπτωτική εκτίμηση*. Η ασυμπτωτική εκτίμηση καταδεικνύει ότι η συμπεριφορά του αλγόριθμου για μεγάλα στιγμιότυπα είναι αυτή που χαρακτηρίζει την αποδοτικότητά του.

Με απλά λόγια, το σημαντικό στη (θεωρητική) ανάλυση ενός αλγόριθμου είναι ο προσδιορισμός της *τάξης μεγέθους* του χρόνου εκτέλεσης σαν συνάρτηση του μεγέθους του στιγμιότυπου[4]. Όταν το μέγεθος του στιγμιότυπου n γίνει αρκετά μεγάλο, οι τιμές της συνάρτησης που καθορίζει την τάξη μεγέθους είναι σημαντικά μεγαλύτερες από οποιουσδήποτε άλλους όρους. Για παράδειγμα, όταν το n γίνει αρκετά μεγάλο, το 2^n είναι σημαντικά μεγαλύτερο από το $1000n^2$ και το n^2 είναι σημαντικά μεγαλύτερο από το $100\, n \log n$. Στη θεωρία λοιπόν, ένας αλγόριθμος με χρόνο εκτέλεσης μικρότερης τάξης μεγέθους είναι προτιμότερος από έναν αλγόριθμο με χρόνο εκτέλεσης μεγαλύτερης τάξης μεγέθους. Στην πράξη, ο πρώτος αλγόριθμος είναι γρηγορότερος από τον δεύτερο μόνο όταν τα στιγμιότυπα που επιλύονται είναι αρκετά μεγάλα.

Μερικές τάξεις μεγέθους είναι ιδιαίτερα συνηθισμένες. Όταν το $T(n)$ είναι ανεξάρτητο του n λέμε ότι ο χρόνος εκτέλεσης είναι *σταθερός* (σε αυτή την περίπτωση μπορούμε να θεωρήσουμε

[4]Στο εξής το μέγεθος του στιγμιότυπου εισόδου θα συμβολίζεται με n εκτός αν ρητά αναφέρεται κάτι διαφορετικό. Επίσης, θα συμβολίζουμε με $\log n$ το λογάριθμο με βάση 2 και με $\ln n$ το φυσικό λογάριθμο του n.

ότι $T(n) = 1$ με βάση την αρχή του αναλλοίωτου). Όταν $T(n) = n$, n^2, ή n^3 λέμε ότι ο χρόνος εκτέλεσης είναι *γραμμικός*, *τετραγωνικός*, και *κυβικός* αντίστοιχα. Όταν $T(n) = n^k$ για κάποια συγκεκριμένη σταθερά k, λέμε ότι ο χρόνος εκτέλεσης είναι *πολυωνυμικός*, ενώ όταν $T(n) = d^n$ για κάποια συγκεκριμένη σταθερά $d > 1$, λέμε ότι ο χρόνος εκτέλεσης είναι *εκθετικός*.

Επισήμανση. Στη συζήτηση σχετικά με την ασυμπτωτική εκτίμηση, η έννοια του χρόνου εκτέλεσης μπορεί να αντικατασταθεί από οποιονδήποτε υπολογιστικό πόρο (π.χ. μνήμη, αριθμό επεξεργαστών, μηνύματα δικτύου, κλπ.) ως προς την οποία υπολογίζουμε την αποδοτικότητα του αλγόριθμου. □

1.3.2 Ανάλυση Χειρότερης Περίπτωσης

Η ασυμπτωτική εκτίμηση χρησιμοποιείται για την εξαγωγή συμπερασμάτων σχετικά με το χρόνο εκτέλεσης των αλγορίθμων, και γενικότερα σχετικά με τις απαιτήσεις τους σε διάφορα είδη υπολογιστικών πόρων. Στην ιδανική περίπτωση, θα θέλαμε ο χρόνος εκτέλεσης ενός αλγόριθμου να δίνεται από μία συνάρτηση του n, δηλαδή του μεγέθους του στιγμιότυπου εισόδου.

Είναι όμως πολύ συνηθισμένο, ένας αλγόριθμος να παρουσιάζει εντελώς διαφορετικό χρόνο εκτέλεσης για διαφορετικά στιγμιότυπα του ίδιου μεγέθους. Υπάρχουν δηλαδή σημαντικές αποκλίσεις στο χρόνο εκτέλεσης, οι οποίες εξαρτώνται εκτός από το μέγεθος, και από τη δομή στιγμιότυπου εισόδου.

Για παράδειγμα, ας θεωρήσουμε τον αλγόριθμο γραμμικής αναζήτησης σε έναν πίνακα με n στοιχεία. Ο χρόνος εκτέλεσης του αλγορίθμου εξαρτάται από τη θέση του στοιχείου που αναζητούμε. Αν το στοιχείο βρίσκεται στις πρώτες θέσεις, ο χρόνος εκτέλεσης του αλγόριθμου είναι σταθερός, ενώ αν το στοιχείο βρίσκεται στις τελευταίες θέσεις ή δεν υπάρχει στον πίνακα, ο χρόνος εκτέλεσης είναι γραμμικός. Με άλλα λόγια, ο αλγόριθμος γραμμικής αναζήτησης έχει σταθερό χρόνο εκτέλεσης *καλύτερης περίπτωσης* και γραμμικό χρόνο εκτέλεσης *χειρότερης περίπτωσης*.

Τίθεται λοιπόν το ερώτημα ποια από τις δύο περιπτώσεις μπορεί να θεωρηθεί αντιπροσωπευτική για τη συμπεριφορά του αλγόριθμου (στη θεωρία και στην πράξη). Η απλούστερη και ασφαλέστερη επιλογή είναι να θεωρήσουμε μόνο τα στιγμιότυπα που μεγιστοποιούν το χρόνο εκτέλεσης του αλγόριθμου, δηλαδή να εφαρμόσουμε τη λεγόμενη *ανάλυση χειρότερης περίπτωσης* (worst-case analysis).

Από θεωρητικής άποψης, η ανάλυση χειρότερης περίπτωσης είναι ιδιαίτερα επιθυμητή επειδή παρέχει ένα άνω φράγμα στο χρόνο εκτέλεσης που ισχύει για κάθε στιγμιότυπο εισόδου. Όμως και στην πράξη, οι περισσότεροι αλγόριθμοι έχουν χρόνους εκτέλεσης που δεν διαφοροποιούνται σημαντικά από τα αποτελέσματα της ανάλυσης χειρότερης περίπτωσης (π.χ. βλ. αλγόριθμους ταξινόμησης merge-sort και heap-sort ή αλγόριθμο δυαδικής αναζήτησης). Για κάποιους αλγόριθμους που τα στιγμιότυπα εισόδου χειρότερης περίπτωσης είναι η εξαίρεση και όχι ο κανόνας (π.χ. πιθανοτική εκδοχή της quicksort, αναζήτηση με παρεμβολή), εφαρμόζουμε ανάλυση *μέσης περίπτωσης* (avegage-case analysis) για να έχουμε σαφή και ασφαλή εικόνα για το χρόνο εκτέλεσης του αλγόριθμου στην πράξη.

1.3.3 Ασυμπτωτικός Συμβολισμός

Η ασυμπτωτική εκτίμηση αγνοεί τις σταθερές και εξετάζει μόνο την τάξη μεγέθους του χρόνου εκτέλεσης ενός αλγόριθμου. Ο *ασυμπτωτικός συμβολισμός* χρησιμοποιείται για να εκφραστούν

τα αποτελέσματα της ασυμπτωτικής εκτίμησης.

Για διευκόλυνση, ορίζουμε τις διάφορες μορφές ασυμπτωτικού συμβολισμού για συναρτήσεις με πεδίο ορισμού το σύνολο των φυσικών αριθμών $\mathbb{N} = \{0, 1, 2, \ldots\}$. Ο λόγος είναι ότι το μέγεθος ενός στιγμιότυπου είναι φυσικός αριθμός. Οι ορισμοί μπορούν εύκολα να επεκταθούν σε πραγματικές συναρτήσεις. Σε όλους τους ορισμούς υποθέτουμε ότι οι συναρτήσεις είναι θετικές (δηλ. παίρνουν τιμές στο $\mathbb{R}_+^*$).

Συμβολισμός Θ. Το Θ χρησιμοποιείται για τον *ακριβή προσδιορισμό* της τάξης μεγέθους μίας συνάρτησης. Συγκεκριμένα, για κάθε συνάρτηση $g(n)$, η κλάση συναρτήσεων $\Theta(g(n))$ αποτελείται από όλες τις συναρτήσεις με την ίδια τάξη μεγέθους. Τυπικά, δεδομένης μιας συνάρτησης $g(n)$, συμβολίζουμε με $\Theta(g(n))$ το σύνολο συναρτήσεων

$$\Theta(g(n)) = \{f(n) : \mathbb{N} \mapsto \mathbb{R}_+ \mid (\exists c_1, c_2 \in \mathbb{R}_+^*)\,(\exists n_0 \in \mathbb{N})\,(\forall n \geq n_0)\; c_1\, g(n) \leq f(n) \leq c_2\, g(n)\}$$

Με άλλα λόγια, μια συνάρτηση $f(n)$ ανήκει στην κλάση συναρτήσεων $\Theta(g(n))$ αν υπάρχουν θετικές σταθερές c_1, c_2 τέτοιες ώστε το $f(n)$ να είναι μεταξύ $c_1\, g(n)$ και $c_2\, g(n)$ για όλες τις τιμές του n που ξεπερνούν ένα κατώφλι n_0 (διαισθητικά, για όλες τις *μεγάλες* τιμές του n).

Ο συμβολισμός $\Theta(g(n))$ δηλώνει κλάση συναρτήσεων. Μολαταύτα, γράφουμε $f(n) = \Theta(g(n))$ (και όχι $f(n) \in \Theta(g(n))$) για να δηλώσουμε ότι η $f(n)$ ανήκει στην κλάση συναρτήσεων $\Theta(g(n))$. Το ίδιο συμβαίνει και με όλες τις μορφές ασυμπτωτικού συμβολισμού.

Εξ' ορισμού ο ασυμπτωτικός συμβολισμός αγνοεί τις σταθερές. Επιπλέον, ο ασυμπτωτικός συμβολισμός προσδιορίζει την τάξη μεγέθους των συναρτήσεων και αγνοεί όρους μικρότερης τάξης μεγέθους. Καθώς το n τείνει στο άπειρο, οι όροι μικρότερης τάξης μεγέθους καθίστανται αμελητέοι σε σχέση με τον κυρίαρχο όρο.

Για παράδειγμα, για κάθε πολυώνυμο δευτέρου βαθμού $f(n) = an^2 + bn + c$, $a > 0$, ισχύει ότι $f(n) = \Theta(n^2)$. Πράγματι, καθώς το n τείνει στο άπειρο, ο κυρίαρχος όρος n^2 καθορίζει τη συμπεριφορά της συνάρτησης $f(n)$. Γενικότερα, κάθε πολυώνυμο βαθμού d ανήκει στην κλάση $\Theta(n^d)$. Ομοίως, είναι $106n^2 + 104n^3 + 10^{-2}n^3 \log n = \Theta(n^3 \log n)$, αφού το $n^3 \log n$ είναι ο κυρίαρχος όρος. Ακόμη, κάθε σταθερή συνάρτηση ανήκει στην κλάση $\Theta(1)$, αφού το σημαντικό είναι ότι η συνάρτηση έχει την ίδια τιμή για κάθε n και όχι ποια ακριβώς είναι αυτή η τιμή.

Συμβολισμός O. Δεδομένης μιας συνάρτησης $g(n)$, συμβολίζουμε με $\mathrm{O}(g(n))$ το σύνολο των συναρτήσεων

$$\mathrm{O}(g(n)) = \{f(n) : \mathbb{N} \mapsto \mathbb{R}_+ \mid (\exists c \in \mathbb{R}_+^*)\,(\exists n_0 \in \mathbb{N})\,(\forall n \geq n_0)\; f(n) \leq c\, g(n)\}$$

Με άλλα λόγια, μια συνάρτηση $f(n)$ ανήκει στην κλάση συναρτήσεων $\mathrm{O}(g(n))$ αν υπάρχει σταθερά c τέτοια ώστε το $f(n)$ να μην ξεπερνά το $c\, g(n)$ για όλες τις μεγάλες τιμές του n.

Ο συμβολισμός O δίνει ένα ένα ασυμπωτικό άνω φράγμα στην τάξη μεγέθους μιας συνάρτησης. Όπως και για το συμβολισμό Θ, γράφουμε $f(n) = \mathrm{O}(g(n))$ αν και το $\mathrm{O}(g(n))$ ορίζει κλάση συναρτήσεων.

Συγκρίνοντας τους ορισμούς των συμβολισμών Θ και O, καταλήγουμε στο συμπέρασμα ότι κάθε συνάρτηση $f(n)$ που ανήκει στο $\Theta(g(n))$, ανήκει και στο $\mathrm{O}(g(n))$. Δηλαδή, $\Theta(g(n)) \subseteq \mathrm{O}(g(n))$.

Συμβολισμός Ω. Δεδομένης μιας συνάρτησης $g(n)$, συμβολίζουμε με $\Omega(g(n))$ το σύνολο των συναρτήσεων

$$\Omega(g(n)) = \{f(n) : \mathbb{N} \mapsto \mathbb{R}_+ \mid (\exists c \in \mathbb{R}_+^*)\,(\exists n_0 \in \mathbb{N})\,(\forall n \geq n_0)\; c\,g(n) \leq f(n)\}$$

Με άλλα λόγια, μια συνάρτηση $f(n)$ ανήκει στην κλάση συναρτήσεων $\mathrm{O}(g(n))$ αν υπάρχει σταθερά c τέτοια ώστε το $f(n)$ να μην υπολείπεται του $c\,g(n)$ για όλες τις μεγάλες τιμές του n.

Ο συμβολισμός Ω δίνει ένα ένα ασυμπτωτικό κάτω φράγμα στην τάξη μεγέθους μιας συνάρτησης. Συγκρίνοντας τους ορισμούς των συμβολισμών Θ, O, και Ω, καταλήγουμε στο συμπέρασμα ότι οι συναρτήσεις $f(n)$ και $g(n)$ είναι $f(n) = \Theta(g(n))$ αν και μόνο αν $f(n) = \mathrm{O}(g(n))$ και $f(n) = \Omega(g(n))$. Για παράδειγμα, όταν $f(n) = \Theta(n^2)$, ισχύει επίσης ότι $f(n) = \mathrm{O}(n^2)$ και $f(n) = \Omega(n^2)$. Αντίστροφα, όταν $g(n) = \mathrm{O}(n^3)$ και $g(n) = \Omega(n^3)$, ισχύει ότι $g(n) = \Theta(n^3)$.

Συμβολισμός o. Το ασυμπτωτικό άνω φράγμα του συμβολισμού O δεν είναι πάντα ακριβές (tight), δηλ. δεν είναι πάντα το καλύτερο δυνατό. Για παράδειγμα, το φράγμα $2n^2 = \mathrm{O}(n^2)$ είναι ακριβές, ενώ το φράγμα $2n^2 = \mathrm{O}(n^3)$ δεν είναι. Ο συμβολισμός o δηλώνει ένα άνω φράγμα το οποίο δεν είναι ακριβές.

Δεδομένης μιας συνάρτησης $g(n)$, συμβολίζουμε με $\mathrm{o}(g(n))$ το σύνολο των συναρτήσεων

$$\mathrm{o}(g(n)) = \{f(n) : \mathbb{N} \mapsto \mathbb{R}_+ \mid (\forall c \in \mathbb{R}_+^*)\,(\exists n_0 \in \mathbb{N})\,(\forall n \geq n_0)\; f(n) < c\,g(n)\}\}$$

Ο ορισμός του o απαιτεί για *κάθε* θετική σταθερά c, το $f(n)$ να είναι μικρότερο του $c\,g(n)$ για όλες τις μεγάλες τιμές του n. Μια ισοδύναμη μαθηματική διατύπωση είναι ότι $f(n) = \mathrm{o}(g(n))$ αν $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. Επομένως, $2n^2 = \mathrm{o}(n^3)$, αλλά $2n^2 \neq \mathrm{o}(n^2)$.

Συμβολισμός ω. Παρόμοια με το o, ο συμβολισμός ω χρησιμοποιείται για ένα κάτω φράγμα που δεν είναι ακριβές. Δεδομένης μιας συνάρτησης $g(n)$, συμβολίζουμε με $\omega(g(n))$ το σύνολο των συναρτήσεων

$$\omega(g(n)) = \{f(n) : \mathbb{N} \mapsto \mathbb{R}_+ \mid (\forall c \in \mathbb{R}_+^*)\,(\exists n_0 \in \mathbb{N})\,(\forall n \geq n_0)\; c\,g(n) < f(n)\}$$

Ο ορισμός του ω απαιτεί για *κάθε* θετική σταθερά c, το $f(n)$ να είναι μεγαλύτερο του $c\,g(n)$ για όλες τις μεγάλες τιμές του n. Μια ισοδύναμη μαθηματική διατύπωση είναι ότι $f(n) = \omega(g(n))$ αν $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$. Επομένως, $2n^2 = \omega(n)$, αλλά $2n^2 \neq \omega(n^2)$.

Άσκηση 1.1. Έστω $f(n)$ και $g(n)$ θετικές συναρτήσεις με πεδίο ορισμού τους φυσικούς αριθμούς. Για κάθε μια από τις παρακάτω προτάσεις είτε να αποδείξετε ότι είναι αληθής είτε να εξηγήσετε γιατί είναι ψευδής:

1. $10f(n) + 10^{10} = \mathrm{O}(f(n))$.
2. $f(n) + g(n) = \Theta(\min\{f(n), g(n)\})$.
3. $f(n) + g(n) = \Omega(\min\{f(n), g(n)\})$.
4. $f(n) + g(n) = \mathrm{O}(\max\{f(n), g(n)\})$.
5. Αν $f(n) = \Theta(g(n))$, τότε $2^{f(n)} = \Theta(2^{g(n)})$.

6. $f(n) = \Omega(f(n/2))$.

Λύση. Οι 1, 3, 4, και 6 είναι αληθείς και αποδεικνύονται με τους ορισμούς. Οι 2 και 5 είναι ψευδείς και μπορούν να βρεθούν αντιπαραδείγματα. □

Άσκηση 1.2. Να τοποθετηθούν οι παρακάτω συναρτήσεις σε αύξουσα σειρά τάξης μεγέθους:

2^{5n}	$\log^4 n$	$(\log n)^{100} \log\log n$	$n \log\log n$	$n^{0.1} \log\log n$
2^n	$n^{0.6}$	$2^n + n^{2^{100}}$	$n^{1/\log n}$	$\log(n!)$
$n^{\log n}$	$\log\log n$	$2^{\log^3 n}$	$\frac{n}{\log_n 2} + n$	$(\log n)^{\log n}$

Λύση. Είναι

$$n^{1/\log n} = \Theta(1),\ \log\log n,\ \log^4 n,\ (\log n)^{100}\log\log n,\ n^{0.1}\log\log n,$$

$$n^{0.6},\ n\log\log n,\ \log(n!) = \Theta(n\log n),\ \tfrac{n}{\log_n 2} + n = \Theta(n\log n),$$

$$(\log n)^{\log n} = \Theta(n^{\log\log n}),\ n^{\log n},\ 2^{\log^3 n} = \Theta(n^{\log^2 n}),\ 2^n,\ 2^n + n^{2^{100}} = \Theta(2^n),\ 2^{5n}$$

Άσκηση 1.3. Στον πίνακα που ακολουθεί, σημειώστε σε ποιες περιπτώσεις η συνάρτηση $f(n)$ ανήκει στις κλάσεις $\Theta(g(n)), \mathrm{O}(g(n)), \mathrm{o}(g(n)), \Omega(g(n))$, και $\omega(g(n))$ και σε ποιες όχι.

$f(n)$	$g(n)$	$\Theta(g(n))$	$\mathrm{O}(g(n))$	$\mathrm{o}(g(n))$	$\Omega(g(n))$	$\omega(g(n))$
2^{n+5}	$2^n + 2^5 + n^{100}$					
$n^4 - n^3$	$16^{\log n}$					
5^{4n}	10^{2n}					
$n^{1/\log\log n}$	$n^{0.001}$					
$n!$	n^n					
$n^{\log^{20} n}$	2^n					

Λύση.

$f(n)$	$g(n)$	$\Theta(g(n))$	$\mathrm{O}(g(n))$	$\mathrm{o}(g(n))$	$\Omega(g(n))$	$\omega(g(n))$
2^{n+5}	$2^n + 2^5 + n^{100}$	ναι	ναι	όχι	ναι	όχι
$n^4 - n^3$	$16^{\log n}$	ναι	ναι	όχι	ναι	όχι
5^{4n}	10^{2n}	όχι	όχι	όχι	ναι	ναι
$n^{1/\log\log n}$	$n^{0.001}$	όχι	ναι	ναι	όχι	όχι
$n!$	n^n	όχι	ναι	ναι	όχι	όχι
$n^{\log^{20} n}$	2^n	όχι	ναι	ναι	όχι	όχι

1.4 Βιβλιογραφικές Αναφορές

Τα [11], [7], [3], [37], [30], και [1] αποτελούν εξαιρετικά βιβλία σε θέματα Δομών Δεδομένων, Θεωρίας Αλγορίθμων, και Υπολογιστικής Πολυπλοκότητας. Ειδικότερα, το [11] αποτελεί σημείο αναφοράς στην περιοχή της Θεωρίας Αλγορίθμων, διδάσκεται σε όλα τα μεγάλα Πανεπιστήμια της Αμερικής και της Ευρώπης, και καλύπτει με εξαιρετικό τρόπο όλα τα βασικά και αρκετά προχωρημένα ζητήματα. Το [1] ήταν το πρώτο βιβλίο εισαγωγής στη Θεωρία Αλγορίθμων. Οι τρεις τόμοι του K. Mehlhorn [30] αποτελούσαν για πολλά χρόνια σημείο αναφοράς στην περιοχή των Δομών Δεδομένων. Εξαιρετικά βιβλία σε θέματα Αλγορίθμων και Δομών Δεδομένων είναι τα [3] και [37] (βλ. και σχετική ιστοσελίδα που συντηρείται από τον R. Sedgewick). Το [7] αποτελεί ένα εξαιρετικό βιβλίο που καλύπτει όλο το φάσμα της Θεωρίας Αλγορίθμων εστιάζοντας περισσότερο στις μεθόδους και τις τεχνικές. Τα [11] και [7] έχουν πολλά εξαιρετικά παραδείγματα και ασκήσεις.

Το [32] εστιάζει στην αποδοτική επίλυση προβλημάτων συνδυαστικής βελτιστοποίησης με σημείο εκκίνησης και αναφοράς το Γραμμικό Προγραμματισμό. Τα [41] και [17] εστιάζουν σε αλγόριθμους γραφημάτων. Τα τρία αυτά βιβλία προχωρούν σε βάθος και απαιτούν πολύ καλό μαθηματικό και αλγοριθμικό υπόβαθρο από τον αναγνώστη.

Τα [20] και [33] αποτελούν εξαιρετικά βιβλία σε θέματα Υπολογιστικής Πολυπλοκότητας. Ειδικότερα το [20] αποτελεί σημείο αναφοράς για θέματα **NP**-πληρότητας και δυσεπίλυτων υπολογιστικών προβλημάτων γενικότερα. Το [33] αποτελεί ίσως την πληρέστερη και πιο σύγχρονη εισαγωγή στη Θεωρία Υπολογιστικής Πολυπλοκότητας.

Τα [31] και [42] περιγράφουν τις σύγχρονες εξελίξεις στην περιοχή της Θεωρίας Αλγορίθμων. Το πρώτο αναφέρεται αποκλειστικά σε Πιθανοτικούς Αλγόριθμους και το δεύτερο σε θέματα Αλγόριθμων Προσέγγισης για δυσεπίλυτα υπολογιστικά προβλήματα.

2 Διαίρει-και-Βασίλευε

Η μέθοδος του διαίρει-και-βασίλευε (divide-and-conquer) είναι μία ιδιαίτερα απλή και πολύ αποτελεσματική μέθοδος σχεδιασμού αλγορίθμων που βρίσκει εφαρμογή σε πληθώρα προβλημάτων από διαφορετικά πεδία (π.χ. ταξινόμηση, πολλαπλασιασμός πινάκων, μετασχηματισμός Fourier).

Η μέθοδος διαίρει-και-βασίλευε είναι αναδρομική από τη φύση της. Η εφαρμογή του διαίρει-και-βασίλευε για την επίλυση ενός υπολογιστικού προβλήματος συνίσταται σε τρία βασικά βήματα:

1. *Διαίρεση* του στιγμιότυπου εισόδου σε δύο ή περισσότερα υπο-στιγμιότυπα (subinstances).

2. *Επίλυση* των υπο-στιγμιοτύπων με αναδρομική εφαρμογή του ίδιου αλγόριθμου.

3. *Σύνθεση* της λύσης του αρχικού στιγμιότυπου από τις λύσεις των υπο-στιγμιοτύπων.

Η μέθοδος διαίρει-και-βασίλευε εφαρμόζεται πάντα από την κορυφή προς τη βάση (top-down). Διαιρεί συνεχώς μεγάλα στιγμιότυπα σε μικρότερα μέχρι να φτάσει σε στοιχειώδη στιγμιότυπα. Στην πράξη, όταν το μέγεθος των υπο-στιγμιοτύπων γίνει αρκετά μικρό, τερματίζουμε την αναδρομή και επιλύουμε τα υπο-στιγμιότυπα με κάποιον μη-αναδρομικό αλγόριθμο.

Τυπικά παραδείγματα εφαρμογής της μεθόδου διαίρει-και-βασίλευε είναι ο αλγόριθμος ταξινόμησης με συγχώνευση (merge-sort), ο αλγόριθμος ταχείας ταξινόμησης (quicksort), και ο αλγόριθμος επιλογής σε γραμμικό χρόνο που βασίζεται στην ίδια διαδικασία διαίρεσης με την ταχεία ταξινόμηση. Αυτοί οι αλγόριθμοι παρουσιάστηκαν αναλυτικά στο μάθημα των Δομών Δεδομένων.

Οι προϋποθέσεις για επιτυχημένη εφαρμογή της μεθόδου διαίρει-και-βασίλευε είναι:

1. Η διαίρεση να είναι σημαντικά ευκολότερη από την επίλυση του αρχικού στιγμιότυπου. Για παράδειγμα, στην ταξινόμηση με συγχώνευση, η διαίρεση είναι τετριμένη, ο αρχικός πίνακας απλώς διαιρείται σε δύο ισομεγέθεις υποπίνακες. Στην ταχεία ταξινόμηση, η διαίρεση γίνεται εύκολα εξετάζοντας κάθε στοιχείο του πίνακα μία φορά.

2. Η διαίρεση να παράγει υπο-στιγμιότυπα που είναι σημαντικά μικρότερα και εξ' αιτίας αυτού, σημαντικά ευκολότερο να επιλυθούν σε σχέση με το αρχικό στιγμιότυπο. Για παράδειγμα, στους αλγόριθμους ταξινόμησης με συγχώνευση και ταχείας ταξινόμησης, η ταξινόμηση δύο συνόλων $n/2$ στοιχείων είναι σημαντικά ευκολότερη από την ταξινόμηση ενός συνόλου n στοιχείων.

3. Η διαίρεση να μην οδηγεί σε ίδια (ή επικαλυπτόμενα) στιγμιότυπα. Επειδή κάθε υποστιγμιότυπο λύνεται από μια ανεξάρτητη αναδρομική κλήση, η επανάληψη υπο-στιγμιοτύπων

οδηγεί σε σπατάλη υπολογιστικών πόρων. Για παράδειγμα, στους αλγόριθμους ταξινόμησης με συγχώνευση και ταχείας ταξινόμησης, από τη διαίρεση προκύπτουν ξένοι μεταξύ τους υποπίνακες που ταξινομούνται ανεξάρτητα.

4. Η σύνθεση της λύσης για το αρχικό στιγμιότυπο από τις λύσεις των υπο-στιγμιοτύπων να είναι σημαντικά ευκολότερη από την επίλυση του αρχικού στιγμιότυπου. Στην ταχεία ταξινόμηση για παράδειγμα, η διαδικασία της διαίρεσης καθιστά τη διαδικασία της σύνθεσης τετριμμένη. Στην ταξινόμηση με συγχώνευση, δύο ταξινομημένοι υποπίνακες μπορούν να συγχωνευθούν εύκολα σε έναν ταξινομημένο πίνακα σε γραμμικό χρόνο.

Εκτός από την απλότητα στη σύλληψη και την εφαρμογή, ένα σημαντικό πλεονέκτημα της μεθόδου διαίρει-και-βασίλευε είναι ότι οδηγεί σε (αναδρομικούς) αλγόριθμους που είναι εύκολο να αναλυθούν. Η ανάλυση ενός αλγόριθμου διαίρει-και-βασίλευε συνίσταται στη διατύπωση και την επίλυση της αναδρομικής εξίσωσης που διέπει τη λειτουργία του αλγορίθμου. Η αναδρομική εξίσωση προκύπτει συνήθως εύκολα από την περιγραφή του αλγόριθμου. Η επίλυση της αναδρομικής εξίσωσης είναι ακόμη ευκολότερη αφού είναι γνωστά πολλά ισχυρά μαθηματικά εργαλεία για αυτό το σκοπό, ενώ μπορεί να γίνει και με τη χρήση μαθηματικών πακέτων λογισμικού (Mapple, Mathematica, κλπ).

Στη συνέχεια θα παρουσιάσουμε αλγόριθμους διαίρει-και-βασίλευε για τον πολλαπλασιασμό πολυψήφιων αριθμών, για τον πολλαπλασιασμό πινάκων, για τον υπολογισμό της n-οστής δύναμης ενός αριθμού, καθώς και τον αλγόριθμο Γρήγορου Μετασχηματισμού Fourier (Fast Fourier Transform).

Άσκηση 2.1. Δίνεται ο παρακάτω αλγόριθμος για τον υπολογισμό του n-οστού όρου της ακολουθίας Fibonacci. Ποιός είναι ο χρόνος εκτέλεσης του αλγορίθμου; Γιατί ο αλγόριθμος αυτός δεν είναι αποδοτικός; Να διατυπώσετε έναν αλγόριθμο που να υπολογίζει το n-οστό όρο της ακολουθίας Fibonacci σε γραμμικό χρόνο.

```
FibonacciRec(n)
    if n ≤ 1 then return(n);
    return(FibonacciRec(n − 1) + FibonacciRec(n − 2));
```

Λύση. Ο χρόνος εκτέλεσης $T(n)$ για τον υπολογισμός του n-οστού όρου δίνεται από την αναδρομική σχέση $T(n) = \Theta(1) + T(n-1) + T(n-2)$ με αρχική συνθήκη $T(1) = \Theta(1)$. Η αναδρομική σχέση που περιγράφει το $T(n)$ έχει την ίδια ασυμπτωτική συμπεριφορά με την ακολουθία Fibonacci και η λύση της είναι $T(n) = \Theta(\varphi^n)$, όπου το φ είναι η μεγαλύτερη λύση της εξίσωσης $x^2 = x + 1$. Το φ είναι γνωστό σαν *χρυσή τομή* (golden ratio) και έχει τιμή $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$. Ο χρόνος εκτέλεσης του αλγόριθμου είναι λοιπόν *εκθετικός*! Αναπτύσσοντας την αναδρομή, καταλαβαίνουμε την αιτία: Κάθε φορά που κάποιος όρος χρειάζεται για τον υπολογισμό κάποιων άλλων, αυτός υπολογίζεται εξαρχής από μία ανεξάρτητη αναδρομική κλήση. Αυτό έχει σαν αποτέλεσμα έναν εκθετικό αριθμό από αναδρομικές κλήσεις (περίπου φ^n) για τον υπολογισμό μόλις n όρων της ακολουθίας.

Υπάρχει ένας απλός επαναληπτικός αλγόριθμος με χρόνο εκτέλεσης $\Theta(n)$.

```
Fibonacci(n)
    f_c ← 1; f_p ← 0;
    for i ← 1 to n do
        f_c ← f_c + f_p;
        f_p ← f_c − f_p;
    return(f_p);
```

Ο αλγόριθμος αυτός διατηρεί την αμετάβλητη συνθήκη (invariant) ότι μετά την ολοκλήρωση του for-loop για την τιμή i, $i = 1, \ldots, n$, η μεταβλητή f_c είναι ίση με τον $i+1$-οστό όρο της ακολουθίας Fibonacci, και η μεταβλητή f_p είναι ίση με τον i-οστό όρο. Αυτή η αμετάβλητη συνθήκη είναι εύκολο να αποδειχθεί επαγωγικά. □

2.1 Πολλαπλασιασμός Πολυψήφιων Αριθμών

Στις περισσότερες περιπτώσεις οι αριθμητικές πράξεις θεωρούνται στοιχειώδεις υπολογιστικές λειτουργίες που μπορούν να εκτελεστούν σε σταθερό χρόνο. Όταν όμως πρόκειται για πολυψήφιους αριθμούς, οι αριθμητικές πράξεις δεν μπορούν πλέον να θεωρηθούν στοιχειώδεις λειτουργίες και διαφορετικοί αλγόριθμοι μπορεί να έχουν σημαντικά διαφορετικό χρόνο απόκρισης. Σε αυτές τις περιπτώσεις, μπορούμε να προσδιορίσουμε το χρόνο εκτέλεσης θεωρώντας τις πράξεις μονοψήφιων αριθμών σαν στοιχειώδεις λειτουργίες. Προκειμένου για αριθμητικές πράξεις πολυψήφιων αριθμών, το μέγεθος της εισόδου είναι ίσο με το άθροισμα των ψηφίων των αριθμών, αφού η αύξηση των ψηφίων είναι το γεγονός που αυξάνει τη δυσκολία του προβλήματος.

Έστω ότι θέλουμε να υπολογίσουμε το άθροισμα δύο αριθμών x και y, που αποτελούνται από n δυαδικά ψηφία. Ο κλασικός αλγόριθμος της πρόσθεσης με κρατούμενο υπολογίζει το άθροισμα χρησιμοποιώντας $\Theta(n)$ στοιχειώδεις προσθέσεις μεταξύ δυαδικών ψηφίων.

Όσον αφορά στον πολλαπλασιασμό δύο n-ψηφίων αριθμών x και y, ο αλγόριθμος που διδάσκεται στο δημοτικό σχολείο απαιτεί $\Theta(n^2)$ στοιχειώδεις πράξεις (n προσθέσεις αριθμών που αποτελούνται από n ή περισσότερα δυαδικά ψηφία). Επίσης, ο αλγόριθμος πολλαπλασιασμού a la russe απαιτεί $\Theta(n^2)$ στοιχειώδεις πράξεις (αυτός ο αλγόριθμος περιγράφεται στις εισαγωγικές έννοιες για το μάθημα των Δομών Δεδομένων). Συγκεκριμένα, για κάθε δυαδικό ψηφίο του πολλαπλασιαστή που είναι 1, απαιτείται μία πρόσθεση μεταξύ αριθμών με $\Theta(n)$ δυαδικά ψηφία. Ανακύπτει λοιπόν το ερώτημα αν υπάρχει αλγόριθμος πολλαπλασιασμού με ασυμπτωτικά καλύτερο χρόνο εκτέλεσης χειρότερης περίπτωσης.

Θα σχεδιάσουμε έναν γρηγορότερο αλγόριθμο πολλαπλασιασμού εφαρμόζοντας τη μέθοδο διαίρει-και-βασίλευε. Θεωρούμε δυαδικούς αριθμούς και υποθέτουμε ότι ο αριθμός n των δυαδικών ψηφίων είναι άρτιος. Έστω λοιπόν $x = x_n x_{n-1} \cdots x_2 x_1$ και $y = y_n y_{n-1} \cdots y_2 y_1$, όπου $x_i, y_i \in \{0, 1\}$ είναι τα δυαδικά ψηφία των αριθμών x και y αντίστοιχα.

Στη φάση της *διαίρεσης*, γράφουμε τους αριθμούς x και y ξεχωρίζοντας τα πιο σημαντικά και τα λιγότερο σημαντικά δυαδικά τους ψηφία. Ο αριθμός x μπορεί να γραφεί σαν $x = 2^{n/2} x_h + x_\ell$, όπου ο x_h αποτελείται από τα $n/2$ πιο σημαντικά δυαδικά ψηφία και ο x_ℓ αποτελείται από τα $n/2$ λιγότερο σημαντικά δυαδικά ψηφία. Ομοίως, $y = 2^{n/2} y_h + y_\ell$. Το γινόμενο $x \times y$ μπορεί να υπολογισθεί ως εξής:

$$x \times y = 2^n x_h y_h + 2^{n/2}(x_h y_\ell + x_\ell y_h) + x_\ell y_\ell = 2^n z_h + 2^{n/2} z_m + z_\ell \tag{2.1}$$

όπου $z_h = x_h y_h$, $z_m = x_h y_\ell + x_\ell y_h$, και $z_\ell = x_\ell y_\ell$.

Ο υπολογισμός του γινομένου $x \times y$ απαιτεί τον υπολογισμό τεσσάρων γινομένων ($x_h y_h$, $x_h y_\ell$, $x_\ell y_h$, και $x_\ell y_\ell$) που προκύπτουν από πολλαπλασιασμούς $\frac{n}{2}$-ψηφίων αριθμών. Στη φάση της *επίλυσης*, υπολογίζουμε αυτά τα γινόμενα πραγματοποιώντας τις αντίστοιχες αναδρομικές κλήσεις του ίδιου αλγόριθμου.

Στη φάση της *σύνθεσης*, χρησιμοποιούμε την (2.1) για να υπολογίσουμε το τελικό γινόμενο $x \times y$ από τα επιμέρους γινόμενα που υπολογίσαμε αναδρομικά. Οι δύο ολισθήσεις προς τα αριστερά και οι τρεις προσθέσεις που απαιτούνται για την εφαρμογή της (2.1) αφορούν αριθμούς με όχι περισσότερα από $2n$ δυαδικά ψηφία και μπορούν εύκολα να εκτελεσθούν σε χρόνο $\Theta(n)$.

Έχουμε ολοκληρώσει λοιπόν το σχεδιασμό ενός διαίρει-και-βασίλευε αλγόριθμου για τον πολλαπλασιασμό πολυψηφίων αριθμών. Απομένει η φάση της ανάλυσης, η οποία συνίσταται στη διατύπωση και την επίλυση της αναδρομικής εξίσωσης που διέπει τη λειτουργία του αλγόριθμου. Έστω $T_1(n)$ ο χρόνος για τον υπολογισμό του γινομένου δύο n-ψηφίων αριθμών με βάση τον παραπάνω αλγόριθμο. Από τη συζήτηση που προηγήθηκε προκύπτει ότι

$$T_1(n) = 4\,T_1(n/2) + \Theta(n)$$

με αρχική συνθήκη $T_1(1) = \Theta(1)$. Είναι εύκολο να δει κανείς ότι η λύση αυτής της εξίσωσης είναι $T_1(n) = \Theta(n^2)$. Επομένως, η *προφανής* εφαρμογή της μεθόδου διαίρει-και-βασίλευε δεν πλεονεκτεί έναντι του κλασικού αλγόριθμου πολλαπλασιασμού που γνωρίζουμε από το σχολείο.

Μπορούμε όμως να αποκτήσουμε σημαντικό πλεονέκτημα αν είμαστε πιο προσεκτικοί στον υπολογισμό του z_m. Συγκεκριμένα, μπορούμε να υπολογίσουμε το z_m με χρήση των γινομένων $x_h y_h$ και $x_\ell y_\ell$, και έναν μόνο πολλαπλασιασμό μεταξύ δύο $(\frac{n}{2}+1)$-ψηφίων αριθμών:

$$z_m = (x_h + x_\ell)(y_h + y_\ell) - x_h y_h - x_\ell y_\ell = x_h y_\ell + x_\ell y_h$$

Με βάση τον παραπάνω υπολογισμό του z_m, η εφαρμογή της (2.1) απαιτεί 3 μόνο πολλαπλασιασμούς μεταξύ $\frac{n}{2}$-ψηφίων αριθμών, 6 προσθέσεις μεταξύ αριθμών που δεν έχουν περισσότερα από $2n$ ψηφία, και δύο ολισθήσεις προς τα αριστερά. Όπως και προηγούμενα, οι προσθέσεις και οι ολισθήσεις μπορούν να εκτελεσθούν σε χρόνο $\Theta(n)$.

Έστω $T(n)$ ο χρόνος για τον υπολογισμό του γινομένου δύο n-ψήφιων αριθμών από τον τροποποιημένο αλγόριθμο. Η αναδρομική εξίσωση που περιγράφει το $T(n)$ είναι:

$$T(n) = 3\,T(n/2) + \Theta(n)$$

με αρχική συνθήκη $T(1) = \Theta(1)$. Η λύση αυτής της εξίσωσης είναι $T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.59})$. Επομένως, ο διαίρει-και-βασίλευε αλγόριθμος που προκύπτει από την πιο προσεκτική υλοποίηση της φάσης της σύνθεσης είναι σημαντικά ταχύτερος από τον κλασικό αλγόριθμο και τον αλγόριθμο a la russe.

2.2 Πολλαπλασιασμός Πινάκων

Έστω A, B δύο πίνακες $n \times n$ των οποίων θέλουμε να υπολογίσουμε το γινόμενο $C = A \times B$. Μια πρώτη προσέγγιση είναι να εφαρμόσουμε τον ορισμό:

$$\forall i = 1, \ldots, n,\ j = 1, \ldots, n \quad C[i,j] = \sum_{k=1}^{n} A[i,k]B[k,j] \tag{2.2}$$

Με βάση τον ορισμό, κάθε στοιχείο του πίνακα C μπορεί να υπολογιστεί σε γραμμικό χρόνο $\Theta(n)$. Αφού ο C έχει n^2 στοιχεία, ο χρόνος εκτέλεσης του αλγορίθμου είναι $\Theta(n^3)$.

Στα τέλη της δεκαετίας του 1960, ο V. Strassen [39] παρουσίασε έναν διαίρει-και-βασίλευε αλγόριθμο πολλαπλασιασμού πινάκων με χρόνο εκτέλεσης $\mathrm{O}(n^{\log 7}) = \mathrm{O}(n^{2.81})$. Η βασική ιδέα του αλγόριθμου του Strassen είναι παρόμοια με αυτή του αλγόριθμου πολλαπλασιασμού πολυψήφιων αριθμών που παρουσιάστηκε στην προηγούμενη ενότητα.

Για απλότητα, υποθέτουμε ότι το n είναι άρτιος. Τότε οι πίνακες A, B και C μπορούν να διαιρεθούν σε τέσσερις υποπίνακες μεγέθους $\frac{n}{2} \times \frac{n}{2}$ ως εξής:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

όπου

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

Με αυτόν τον τρόπο, ο υπολογισμός του γινομένου δύο πινάκων $n \times n$ ανάγεται στον υπολογισμό των γινομένων οκτώ ζευγαριών πινάκων $\frac{n}{2} \times \frac{n}{2}$ και σε τέσσερις προσθέσεις πινάκων $\frac{n}{2} \times \frac{n}{2}$. Το άθροισμα δύο πινάκων $n \times n$ μπορεί να υπολογισθεί σε χρόνο $\Theta(n^2)$. Επομένως, ο χρόνος εκτέλεσης $T_1(n)$ του παραπάνω αλγορίθμου πολλαπλασιασμού πινάκων δίνεται από την αναδρομική εξίσωση $T_1(n) = 8\,T_1(n/2) + \Theta(n^2)$ με αρχική συνθήκη $T_1(1) = \Theta(1)$. Η λύση αυτής της εξίσωσης είναι $T_1(n) = \Theta(n^3)$. Όπως και στον πολλαπλασιασμό αριθμών, η προφανής εφαρμογή της μεθόδου διαίρει-και-βασίλευε δεν οδηγεί σε καμία βελτίωση.

Μπορούμε όμως να βελτιώσουμε το χρόνο εκτέλεσης υπολογίζοντας τους υποπίνακες του C με διαφορετικό τρόπο. Αρχικά υπολογίζουμε τους παρακάτω πίνακες $\frac{n}{2} \times \frac{n}{2}$:

$$\begin{aligned} D_1 &= (A_{21} + A_{22} - A_{11})(B_{22} - B_{12} + B_{11}) \\ D_2 &= A_{11}B_{11} \\ D_3 &= A_{12}B_{21} \\ D_4 &= (A_{11} - A_{21})(B_{22} - B_{12}) \\ D_5 &= (A_{21} + A_{22})(B_{12} - B_{11}) \\ D_6 &= (A_{12} - A_{21} + A_{11} - A_{22})B_{22} \\ D_7 &= A_{22}(B_{11} + B_{22} - B_{12} - B_{21}) \end{aligned}$$

Στη συνέχεια υπολογίζουμε τους τέσσερις υποπίνακες του γινομένου C από τους τύπους:

$$\begin{aligned} C_{11} &= D_2 + D_3 \\ C_{12} &= D_1 + D_2 + D_5 + D_6 \\ C_{21} &= D_1 + D_2 + D_4 - D_7 \\ C_{22} &= D_1 + D_2 + D_4 + D_5 \end{aligned}$$

Η παραπάνω διαδικασία υπολογισμού είναι γνωστή σαν αλγόριθμος του Strassen και απαιτεί την εκτέλεση μόνο επτά πολλαπλασιασμών πινάκων $\frac{n}{2} \times \frac{n}{2}$ και ενός σταθερού αριθμού προσθέσεων μεταξύ πινάκων $\frac{n}{2} \times \frac{n}{2}$. Το άθροισμα δύο πίνακων $\frac{n}{2} \times \frac{n}{2}$ μπορεί να υπολογισθεί σε χρόνο $\Theta(n^2)$. Έτσι, ο χρόνος εκτέλεσης του αλγόριθμου του Strassen, έστω $T(n)$, δίνεται από την αναδρομική εξίσωση $T(n) = 7\,T(n/2) + \Theta(n^2)$ με αρχική συνθήκη $T(1) = \Theta(1)$. Η λύση αυτής της εξίσωσης είναι $T(n) = \Theta(n^{\log 7}) = \Theta(n^{2.81})$.

Η μεγάλη πολλαπλασιαστική σταθερά στο χρόνο εκτέλεση του αλγορίθμου του Strassen καθιστά τον αλγόριθμο πρακτικά εφαρμόσιμο μόνο για μεγάλες τιμές του n. Σήμερα είναι γνωστός αλγόριθμος πολλαπλασιασμού πινάκων με χρόνο εκτέλεσης $O(n^{2.376})$ [10]. Επίσης οι R. Raz και A. Shpilka [36] απέδειξαν πρόσφατα ότι το $\Omega(n^2 \log n)$ είναι κάτω φράγμα στο χρόνο εκτέλεσης των αλγορίθμων πολλαπλασιασμού πινάκων.

2.3 Υπολογισμός Δύναμης

Παράδειγμα Χρησιμότητας. Ας ξεκινήσουμε περιγράφοντας μια απλή κρυπτογραφική εφαρμογή. Έστω ότι η Αλίκη και ο Βασίλης επικοινωνούν για πρώτη φορά και χρειάζεται να συμφωνήσουν σε ένα κρυπτογραφικό κλειδί προκειμένου να προστατεύσουν τη μετέπειτα επικοινωνία τους. Ο καθορισμός του κρυπτογραφικού κλειδιού πρέπει να γίνει τηλεφωνικά. Η Αλίκη και ο Βασίλης υποψιάζονται ότι η Εύα παρακολουθεί την τηλεφωνική συνομιλία τους (η Εύα μπορεί να ακούσει αλλά όχι να μεταβάλει τα λεγόμενά τους). Προφανώς, το πρωτόκολλο καθορισμού του κρυπτογραφικού κλειδιού πρέπει να εξασφαλίζει ότι το κλειδί θα γίνει γνωστό μόνο στην Αλίκη και στο Βασίλη (ακόμη και αν η Εύα παρακολουθεί τη συνομιλία τους). Θα παρουσιάσουμε την πρώτη λύση σε αυτό το πρόβλημα που δόθηκε από τους Diffie και Hellman [13] στα μέσα της δεκαετίας του 70.

Αρχικά η Αλίκη και ο Βασίλης συμφωνούν σε έναν πολυψήφιο πρώτο αριθμό p (σε πραγματικές εφαρμογές οι αριθμοί που χρησιμοποιούνται έχουν μήκος μερικές εκατοντάδες ψηφία) και σε έναν πολυψήφιο ακέραιο αριθμό q μικρότερο του p. Οι αριθμοί p και q δεν μπορούν να θεωρηθούν μυστικοί, αφού η Εύα μπορεί να τους πληροφορηθεί παρακολουθώντας τη συνομιλία της Αλίκης και του Βασίλη.

Στη συνέχεια, η Αλίκη και ο Βασίλης διαλέγουν τυχαία και ανεξάρτητα από έναν πολυψήφιο ακέραιο αριθμό μικρότερο του p. Έστω a και b αντίστοιχα αυτοί οι αριθμοί. Η Αλίκη υπολογίζει τον αριθμό $q_a = q^a \bmod p$ και τον κοινοποιεί στον Βασίλη[1]. Ομοίως, ο Βασίλης υπολογίζει τον αριθμό $q_b = q^b \bmod p$ και τον κοινοποιεί στην Αλίκη. Όπως και τα p και q, οι αριθμοί q_a και q_b δεν μπορούν να θεωρηθούν μυστικοί από την Εύα.

[1] Η χρήση αριθμητικής υπολοίπου ως προς p (modulo-p arithmetic) λαμβάνει υπόψη μόνο τα $\log p$ λιγότερο σημαντικά δυαδικά ψηφία του αποτελέσματος κάθε πράξης. Η αριθμητική υπολοίπου δεν επιτρέπει τη δημιουργία αριθμών μεγαλύτερων από το p, που θα ήταν πολύ δύσκολο να διαχειριστούν. Έχει όμως τις περισσότερες από τις ιδιότητες των συνηθισμένων αριθμητικών πράξεων, ενώ έχει και μερικές ακόμη επιθυμητές ιδιότητες που αξιοποιούνται στην κρυπτογραφία. Σε αυτό το παράδειγμα χρησιμοποιούμε μόνο τις παρακάτω ιδιότητες: $(q^a \bmod p)^b \bmod p = q^{ab} \bmod p$ και ότι δεδομένων των p, q, και q_a, είναι εξαιρετικά δύσκολο να υπολογισθεί ένας αριθμός a' τέτοιος ώστε $q_a = q^{a'} \bmod p$. Το πρόβλημα του υπολογισμού του a', που ουσιαστικά είναι ο λογάριθμος με βάση q του q_a στην αριθμητική υπολοίπου ως προς p, είναι γνωστό σαν πρόβλημα του διακριτού λογάριθμου (discrete logarithm problem) και είναι ένα από τα δύο (θεωρούμενα) δύσκολα προβλήματα στα οποία βασίζεται η σύγχρονη κρυπτογραφία. Το δεύτερο πρόβλημα είναι αυτό του υπολογισμού των πρώτων παραγόντων (factoring) ενός μεγάλου ακεραίου αριθμού.

Γνωρίζοντας τους αριθμούς p, a και q_b, η Αλίκη υπολογίζει το

$$K_a = q_b^a \bmod p = (q^b \bmod p)^a \bmod p = q^{ab} \bmod p$$

Ομοίως, γνωρίζοντας τους αριθμούς p, b, και q_a, ο Βασίλης υπολογίζει το

$$K_b = q_a^b \bmod p = (q^a \bmod p)^b = q^{ab} \bmod p$$

Παρατηρούμε ότι $K_a = K_b$. Αυτός ο αριθμός, ο οποίος έχει υπολογισθεί κατ' ιδίαν και από τα δύο μέρη και δεν χρειάζεται να μεταδωθεί μέσω της τηλεφωνικής συνομιλίας, μπορεί να χρησιμοποιηθεί σαν κοινό μυστικό κλειδί από την Αλίκη και το Βασίλη.

Παρακολουθώντας την τηλεφωνική συνομιλία, η Εύα μπορεί να γνωρίζει τους αριθμούς p, q, q_a, και q_b. Για να υπολογίσει όμως τα $K_a = K_b$ χρειάζεται να γνωρίζει και ένα εκ των a ή b (που επίσης δεν έχουν μεταδωθεί τηλεφωνικά). Αυτό μπορεί να γίνει μόνο αν η Εύα λύσει το πρόβλημα του διακριτού λογαρίθμου, πράγμα που θεωρείται εξαιρετικά δύσκολο.

Υπολογισμός Δύναμης. Η εφαρμογή αυτού του πρωτοκόλλου προϋποθέτει τον γρήγορο υπολογισμό της n-οστής δύναμης ενός ακεραίου αριθμού x στην αριθμητική υπολοίπου ως προς p. Χρειαζόμαστε λοιπόν έναν αποδοτικό αλγόριθμο για τον υπολογισμό του $x^n \bmod p$ όπου x, n, και p είναι πολυψήφιοι ακέραιοι αριθμοί. Ο αλγόριθμος αυτός είναι απαραίτητος για τον υπολογισμό των $q_a = q^a \bmod p$ και $q_b = q^b \bmod p$.

Η προφανής προσέγγιση του υπολογισμού όλων των δυνάμεων $x^i \bmod p$, $i = 2, \ldots, n$, πολλαπλασιάζοντας το $x^{i-1} \bmod p$ με το x δεν μπορεί να εφαρμοστεί αφού το n είναι ένας τεράστιος αριθμός[2]. Η μέθοδος διαίρει-και-βασίλευε δίνει έναν απλό αναδρομικό αλγόριθμο για την επίλυση αυτού του προβλήματος.

Αν το n είναι άρτιος, η n-οστή δύναμη του x προκύπτει ως το τετράγωνο της $\frac{n}{2}$-οστή δύναμη του x. Η $\frac{n}{2}$-οστή δύναμης του x υπολογίζεται αναδρομικά με τον ίδιο αλγόριθμο. Αν το n είναι περιττός, η n-οστή δύναμη του x προκύπτει πολλαπλασιάζοντας το x με το τετράγωνο της $\frac{n-1}{2}$-οστής δύναμη του x. Η τελευταία υπολογίζεται αναδρομικά από τον ίδιο αλγόριθμο. Μια αναδρομική υλοποίηση του αλγόριθμου δίνεται στη συνέχεια.

```
ExponRec(x, n, p)
    if n = 1 then return(x mod p);
    t ← ExponRec(x, ⌊n/2⌋, p);
    t ← t^2 mod p;
    if n is odd then return(t × x mod p);
    else return(t);
```

Έστω $T(n)$ ο χρόνος εκτέλεσης του παραπάνω αλγόριθμου για τον υπολογισμό της n-οστής δύναμης ενός αριθμού. Ο αλγόριθμος εκτελεί μία αναδρομική κλήση για τον υπολογισμό της $\frac{n}{2}$-οστής δύναμης και το πολύ δύο πολλαπλασιασμούς αριθμών που δεν ξεπερνούν το p (οπότε έχουν

[2]Αν το n έχει μέγεθος 512 δυαδικά ψηφία, η παραπάνω διαδικασία θα πρέπει να εκτελέσει τον εξωπραγματικά μεγάλο αριθμό των 2^{512} πολλαπλασιασμών!

το πολύ $\log p$ ψηφία). Η αναδρομική κλήση απαιτεί χρόνο $T(n/2)$. Ο χρόνος για τον πολλαπλασιασμό δύο αριθμών μικρότερων του p είναι $O(\log^2 p)$. Το $T(n)$ δίνεται από την αναδρομική σχέση:

$$T(n) = T(n/2) + O(\log^2 p)$$

με αρχική συνθήκη $T(1) = O(1)$. Η λύση αυτής της αναδρομικής εξίσωσης είναι $T(n) = O(\log n \log^2 p)$ αφού σε κάθε βήμα ο αλγόριθμος χρειάζεται χρόνο $O(\log^2 p)$ και το n υποδιπλασιάζεται. Για να γίνει αισθητή η βελτίωση, αν το n έχει 512 δυαδικά ψηφία, ο αλγόριθμος αυτός εκτελεί το πολύ $2^{10} = 1024$ πολλαπλασιασμούς.

Επισήμανση. Ο αλγόριθμος ExponRec πραγματοποιεί μία μόνο αναδρομική κλήση για στιγμιότυπο με μέγεθος ίσο με το μισό του αρχικού. Δηλαδή από τη φάση της διαίρεσης, προκύπτει ένα και όχι δύο ή περισσότερα υπο-στιγμιότυπα. Επομένως μπορούμε να θεωρήσουμε ότι έχουμε *εξειδίκευση* του αρχικού στιγμιότυπου σε υπο-στιγμιότυπο μικρότερου μεγέθους και όχι *διαίρεση* σε μικρότερα υπο-στιγμιότυπα.

Άσκηση 2.2. Να διατυπώσετε την επαναληπτική εκδοχή του αλγόριθμου ExponRec.

Λύση. Η επαναληπτική εκδοχή ουσιαστικά υπολογίζει το $x^n \bmod p$ από την δυαδική αναπαράσταση του n. Παρουσιάζει πολλές ομοιότητες με τον αλγόριθμο πολλαπλασιασμού a la russe (εκεί χρησιμοποιείται η δυαδική αναπαράσταση του πολλαπλασιαστή).

```
ExponIter(x, n, p)
    t ← x; r ← 1;
    while n ≥ 1 do
        if n is odd then r ← r × t mod p;
        t ← t^2 mod p;
        n ← ⌊n/2⌋;
    return(r);
```

Έστω $\ell = \lfloor \log n \rfloor + 1$ το μήκος της δυαδικής αναπαράστασης και έστω $n = n_\ell n_{\ell-1} \ldots n_2 n_1$ η δυαδική αναπαράσταση του n. Ο αλγόριθμος διατηρεί την αμετάβλητη συνθήκη ότι στην αρχή της i-οστής επανάληψης του while-loop, $i = 1, \ldots, \ell$, είναι $t = x^{2^{i-1}} \bmod p$ (μπορεί να αποδειχθεί εύκολα με μαθηματική επαγωγή). Όταν το n_i (δηλαδή το i-οστό bit του n) είναι 1, η τρέχουσα τιμή της μεταβλητής r πολλαπλασιάζεται με $t = x^{2^{i-1}} \bmod p$. Επομένως, στο τέλος του αλγόριθμου η τιμή της μεταβλητής r είναι ίση με x υψωμένο στη δύναμη $\sum_{i=1}^{\ell} n_i 2^{i-1} = n$ (σε αριθμητική υπολοίπου ως προς p). Δηλαδή αποδείξαμε ότι η μεταβλητή r έχει την τιμή $x^n \bmod p$ στο τέλος του αλγορίθμου. □

Άσκηση 2.3. Δίνεται ο 2×2 πίνακας $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$. Συμβολίζουμε με $F_n = [f_n, f_{n-1}]$ το διάνυσμα με στοιχεία το n-στό και τον $(n-1)$-οστό όρο της ακολουθίας Fibonacci. Παρατηρείστε ότι για κάθε ακέραιο $n \geq 1$, το γινόμενο $A \times F_n = [f_n + f_{n-1}, f_n]$ δίνει το διάνυσμα F_{n+1} με τους επόμενους όρους της ακολουθίας. Χρησιμοποιώντας τον πίνακα A και την προηγούμενη παρατήρηση, να δώσετε έναν αλγόριθμο με χρόνο εκτέλεσης $O(\log n)$ που να υπολογίζει το n-οστό όρο της ακολουθίας Fibonacci.

Λύση. Έστω $F_1 = [1, 0]$ το διάνυσμα με στοιχεία τους δύο πρώτους όρους της ακολουθίας. Είναι εύκολο να αποδειχθεί με μαθηματική επαγωγή ότι $F_n = A^{n-1} \times F_1$ για κάθε ακέραιο $n \geq 1$. Ο πίνακας A^{n-1} (δηλαδή η $n-1$ δύναμη του πίνακα A) μπορεί να υπολογισθεί σε χρόνο $O(\log n)$ παρόμοια με τη n-οστή δύναμη ενός αριθμού. Η διατύπωση του ψευδοκώδικα και η ανάλυση αφήνονται στον αναγνώστη. Παρατηρείστε επίσης ότι κάθε δύναμη του πίνακα A έχει τη μορφή $[a+b, a; a, b]$. Αυτή η παρατήρηση μπορεί να οδηγήσει σε απλούστερο και γρηγορότερο αλγόριθμο στην πράξη (δεν επηρεάζει όμως την ασυμπτωτική συμπεριφορά του αλγορίθμου). □

2.4 Πράξεις με Πολυώνυμα

Σε αυτή την ενότητα θα περιγράψουμε αλγόριθμους για την εκτέλεση πράξεων πολυωνύμων, όπως πρόσθεση, πολλαπλασιασμός, και υπολογισμός τιμών.

Έστω $A(x) = \sum_{i=0}^{n-1} a_i x^i$ και $B(x) = \sum_{i=0}^{n-1} b_i x^i$ πολυώνυμα βαθμού $n-1$. Το άθροισμα των πολυωνύμων $A(x)$ και $B(x)$ είναι το πολυώνυμο $A(x) + B(x) = \sum_{i=0}^{n-1}(a_i + b_i)x^i$ βαθμού επίσης $n-1$. Το γινόμενο των πολυωνύμων $A(x)$ και $B(x)$ είναι το πολυώνυμο $A(x)B(x) = \sum_{i=0}^{2(n-1)} c_i x^i$ βαθμού $2(n-1)$ με συντελεστές $c_i = \sum_{j=0}^{i} a_j b_{i-j}$, $i = 0, \ldots, 2(n-1)$, όπου θεωρείται ότι $a_i, b_i = 0$ για κάθε δείκτη $i \geq n$. Η τιμή ενός πολυωνύμου στο σημείο x_0 προκύπτει αντικαθιστώντας όπου x την τιμή x_0 και κάνοντας τις πράξεις.

Ο απλούστερος τρόπος αναπαράστασης ενός πολυωνύμου $A(x)$ είναι η *αναπαράσταση με συντελεστές* (coefficient representation) όπου το πολυώνυμο αναπαρίσταται από το διάνυσμα των συντελεστών του $A = [a_0, a_1, \ldots, a_{n-1}]$. Χρησιμοποιώντας την αναπαράσταση με συντελεστές, το άθροισμα δύο πολυωνύμων μπορεί να υπολογισθεί σε χρόνο $\Theta(n)$ και το γινόμενό τους σε χρόνο $\Theta(n^2)$ με βάση τους ορισμούς. Επίσης, ο υπολογισμός της τιμής ενός πολυωνύμου στο σημείο x_0 μπορεί να γίνει με τη μέθοδο Horner σε χρόνο $\Theta(n)$. Η μέθοδος υπολογισμού Horner, γνωστή και ως μέθοδος φωλιάς, υπολογίζει την τιμή ενός πολυωνύμου από τον παρακάτω τύπο:

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-2} + x_0 a_{n-1}) \cdots))$$

Άσκηση 2.4. Να διατυπώσετε ψευδοκώδικα για τον υπολογισμό της τιμής ενός πολυωνύμου στη θέση x_0 με τη μέθοδο Horner. Αποδείξτε ότι ο χρόνος εκτέλεσης της μεθόδου είναι $\Theta(n)$.

Λύση. Ο ψευδικώδικας είναι:

```
Horner(A[a_0, ..., a_{n-1}], x_0)
    r ← A[n − 1]; i ← n − 2;
    while i ≥ 0 do
        r ← A[i] + r × x_0;
        i ← i − 1;
    return(r);
```

Το while-loop εκτελείται ακριβώς n φορές και κάθε εκτέλεσή του χρειάζεται σταθερό χρόνο (περιλαμβάνει έναν πολλαπλασιασμό και δύο προσθέσεις). Επομένως, ο χρόνος εκτέλεσης της διαδικασίας Horner είναι $\Theta(n)$. □

Άσκηση 2.5. Να διατυπώσετε έναν αλγόριθμο διαίρει-και-βασίλευε για τον πολλαπλασιασμό δύο πολυωνύμων βαθμού $n-1$. Ο χρόνος εκτέλεσης του αλγορίθμου πρέπει να είναι $O(n^{\log 3})$. Υποθέστε ότι το n είναι δύναμη του 2 και ότι τα πολυώνυμα αναπαρίστανται με το διάνυσμα των συντελεστών.

Λύση. Θα εφαρμόσουμε τη μέθοδο διαίρει-και-βασίλευε παρόμοια με την περίπτωση του πολλαπλασιασμού πολυψήφιων αριθμών.

Έστω $A(x) = \sum_{i=0}^{n-1} a_i x^i$ και $B(x) = \sum_{i=0}^{n-1} b_i x^i$ δύο πολυώνυμα βαθμού $n-1$, και $C(x) = A(x)B(x)$ το γινόμενό τους. Τα πολυώνυμα $A(x)$ και $B(x)$ μπορούν να γραφούν σαν $A(x) = A_\ell(x) + x^{n/2}A_h(x)$, όπου $A_\ell(x) = \sum_{i=0}^{n/2-1} a_i x^i$ και $A_h(x) = \sum_{i=0}^{n/2-1} a_{n/2+i} x^i$, και $B(x) = B_\ell(x) + x^{n/2}B_h(x)$, όπου $B_\ell(x) = \sum_{i=0}^{n/2-1} b_i x^i$ και $B_h(x) = \sum_{i=0}^{n/2-1} b_{n/2+i} x^i$.

Παρόμοια με την περίπτωση των αριθμών, το γινόμενο $C(x) = A(x)B(x)$ μπορεί να υπολογιστεί αναδρομικά με βάση τη σχέση:

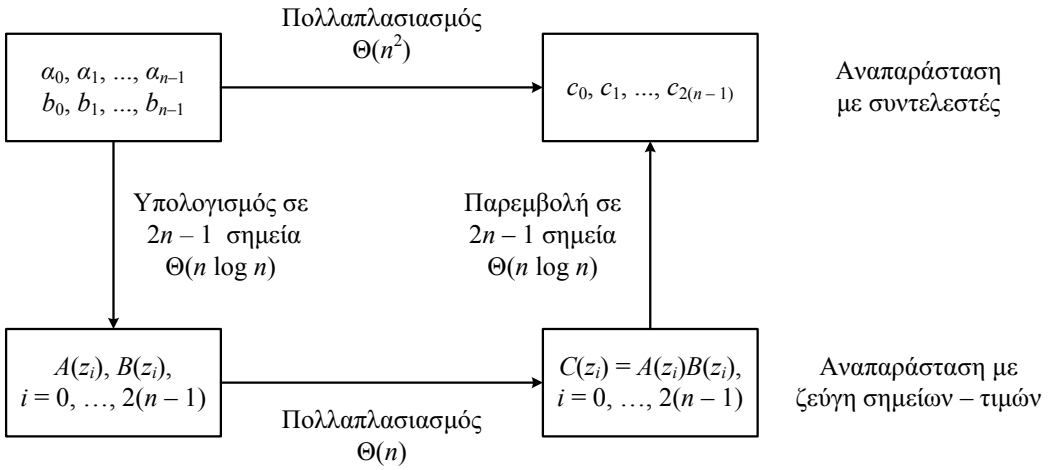
$$\begin{aligned} C(x) &= (A_\ell(x) + x^{n/2}A_h(x))(B_\ell(x) + x^{n/2}B_h(x)) \\ &= A_\ell(x)B_\ell(X) + x^{n/2}(A_\ell(x)B_h(x) + A_h(x) + B_\ell(x)) + x^n A_h(x)B_h(x) \\ &= A_\ell(x)B_\ell(x) + x^{n/2}[(A_\ell(x) + A_h(x))(B_\ell(x) + B_h(x)) - A_\ell(x)B_\ell(x) - A_h(x)B_h(x)] \\ &\quad + x^n A_h(x)B_h(x) \end{aligned}$$

Με βάση την παραπάνω σχέση, ο υπολογισμός του $C(x)$ απαιτεί τρεις πολλαπλασιασμούς πολυωνύμων βαθμού $n/2-1$ και έξι προσθέσεις πολυωνύμων βαθμού το πολύ $2(n-1)$. Έστω $T(n-1)$ ο χρόνος για τον υπολογισμό του γινομένου δύο πολυωνύμων βαθμού $n-1$. Αφού το άθροισμα πολυωνυμών βαθμού το πολύ $2n$ μπορεί να υπολογισθεί σε χρόνο $\Theta(n)$, η αναδρομική σχέση που δίνει το χρόνο εκτέλεσης του αλγόριθμου είναι $T(n-1) = 3\,T(n/2-1) + \Theta(n)$ με αρχική συνθήκη $T(0) = \Theta(1)$. Η λύση αυτής είναι $T(n) = \Theta(n^{\log 3})$ όπως απαιτείται. □

Ένας άλλος τρόπος αναπαράστασης πολυωνύμων είναι με *ζεύγη σημείων-τιμών* (point-value representation). Συγκεκριμένα, ένα πολυώνυμο βαθμού $n-1$ αναπαρίσταται με n διατεταγμένα ζεύγη $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$, όπου για κάθε $i = 0, \ldots, n-1$, το y_i συμβολίζει την τιμή του πολυωνύμου στο σημείο x_i και όλα τα x_i είναι διαφορετικά μεταξύ τους.

Αυτή η αναπαράσταση στηρίζεται στο γεγονός ότι υπάρχει ένα μοναδικό πολυώνυμο βαθμού $n-1$ που αντιστοιχεί σε ένα σύνολο n διαφορετικών ζευγαριών σημείων-τιμών (y_i, x_i). Από την άλλη πλευρά, ένα πολυώνυμο έχει άπειρες διαφορετικές αναπαραστάσεις με ζεύγη σημείων-τιμών, αφού κάθε συνδυασμός n διαφορετικών σημείων x_i δίνει μια διαφορετική αναπαράσταση.

Το σημαντικό πλεονέκτημα της αναπαράστασης με ζεύγη σημείων-τιμών είναι ότι επιτρέπει την πρόσθεση και τον πολλαπλασιασμό δύο πολυωνύμων σε γραμμικό χρόνο. Έστω δύο πολυώνυμα βαθμού $n-1$ που αναπαρίστανται από τις τιμές τους στα ίδια σημεία. Για την πρόσθεση χρειάζονται οι τιμές των πολυωνυμών στα ίδια n σημεία. Έστω λοιπόν ένα πολυώνυμο $A(x)$ βαθμού $n-1$ που αναπαρίσταται σαν $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ και ένα πολυώνυμο $B(x)$ βαθμού $n-1$ που αναπαρίσταται σαν $(x_0, y'_0), \ldots, (x_{n-1}, y'_{n-1})$. Το άθροισμα $A(x) + B(x)$ έχει αναπαράσταση $(x_0, y_0 + y'_0), \ldots, (x_{n-1}, y_{n-1} + y'_{n-1})$ που μπορεί να υπολογιστεί εύκολα σε χρόνο $\Theta(n)$.



Σχήμα 2.1: Αλγόριθμος πολλαπλασιασμού πολυωνύμων σε χρόνο $\Theta(n \log n)$.

Για τον πολλαπλασιασμό χρειάζονται οι τιμές των πολυωνύμων στα ίδια $2n - 1$ σημεία. Έστω ότι το $A(x)$ αναπαρίσταται σαν $(x_0, y_0), \ldots, (x_{2(n-1)}, y_{2(n-1)})$ και το $B(x)$ αναπαρίσταται σαν $(x_0, y'_0), \ldots (x_{2(n-1)}, y'_{2(n-1)})$. Το γινόμενο $A(x)B(x)$ έχει αναπαράσταση $(x_0, y_0 y'_0), \ldots, (x_{2(n-1)}, y_{2(n-1)} y'_{2(n-1)})$ που μπορεί να υπολογιστεί εύκολα σε χρόνο $\Theta(n)$.

Το βασικό μειονέκτημα της αναπαράστασης με ζεύγη σημείων-τιμών είναι ότι ο υπολογισμός της τιμής του πολυωνύμου σε ένα σημείο που δεν συμπεριλαμβάνεται στην αναπαράσταση μπορεί να γίνει μόνο αφού υπολογίσουμε τους συντελεστές του πολυωνύμου.

Κάθε αναπαράσταση έχει τα δικά της πλεονεκτήματα και μειονεκτήματα, τα οποία θα μπορούσαν να συνδυαστούν αν υπήρχε ένας αποδοτικός αλγόριθμος για τη μετατροπή της μίας αναπαράστασης στην άλλη.

Δεδομένης της αναπαράστασης ενός πολυωνύμου με το διάνυσμα των συντελεστών, ο υπολογισμός της αναπαράστασης με ζεύγη σημείων-τιμών είναι γνωστός και σαν *πρόβλημα υπολογισμού τιμών σε n σημεία* (n-point evaluation problem). Ο υπολογισμός τιμών σε n σημεία μπορεί να γίνει με τη μέθοδο Horner σε χρόνο $\Theta(n^2)$. Αν όμως τα σημεία x_i επιλεγούν κατάλληλα, μπορεί να εφαρμοστεί η μέθοδος διαίρει-και-βασίλευε και ο υπολογισμός να γίνει σε χρόνο $\Theta(n \log n)$.

Ο υπολογισμός του διανύσματος των συντελεστών από την αναπαράσταση με ζεύγη σημείων-τιμών ονομάζεται και *πρόβλημα της παρεμβολής σε n σημεία* (n-point interpolation problem). Το πρόβλημα της παρεμβολής σε (κατάλληλα επιλεγμένα) n σημεία μπορεί να λυθεί σε χρόνο $\Theta(n \log n)$.

Έχοντας αλγορίθμους χρόνου $\Theta(n \log n)$ για τα προβλήματα υπολογισμού τιμών και παρεμβολής, μπορούμε να υπολογίσουμε το γινόμενο δύο πολυωνύμων που αναπαρίστανται με συντελεστές σε χρόνο $\Theta(n \log n)$ ακολουθώντας τα βήματα του Σχήματος 2.1: Αρχικά υπολογίζουμε τις τιμές των πολυωνύμων σε $2n - 1$ σημεία, στη συνέχεια εφαρμόζουμε τον αλγόριθμο πολλαπλασιασμού γραμμικού χρόνου για πολυώνυμα που αναπαρίστανται με ζεύγη σημείων-τιμών, και τέλος εφαρμόζουμε παρεμβολή σε $2n - 1$ σημεία για να υπολογίσουμε τους συντελεστές του γινομένου.

2.4.1 Υπολογισμός Τιμών με τη Μέθοδο Διαίρει-και-Βασίλευε

Έστω $A(x)$ ένα πολυώνυμο βαθμού $n-1$ για το οποίο θέλουμε να λύσουμε το πρόβλημα του υπολογισμού n τιμών. Για απλότητα υποθέτουμε ότι το n είναι δύναμη του 2.

Το πολυώνυμο $A(x)$ μπορεί να διαιρεθεί σε δύο πολυώνυμα βαθμού $\frac{n}{2}-1$: το $A_0(x)$ αποτελούμενο από τους συντελεστές για τις άρτιες δυνάμεις του x, και το $A_1(x)$ αποτελούμενο από τους συντελεστές για τις περιττές δυνάμεις του x. Έτσι το $A(x)$ γράφεται σαν $A(x) = A_0(x^2) + x\,A_1(x^2)$, όπου

$$A_0(x) = a_0 + a_2x + a_4x^2 + \ldots + a_{n-2}x^{n/2-1} \quad \text{και} \quad A_1(x) = a_1 + a_3x + a_5x^2 + \ldots + a_{n-1}x^{n/2-1}$$

Επιλέγουμε n σημεία $x_0, x_1, x_2, \ldots, x_{n-3}, x_{n-2}, x_{n-1}$ στα οποία θα υπολογίσουμε τις τιμές του $A(x)$ ώστε για κάθε $i = 0, \ldots, \frac{n}{2}-1$, να ισχύει $x_i^2 = x_{n/2+i}^2$ [3]. Οι τιμές του $A(x)$ θα υπολογιστούν από τη σχέση $A(x) = A_0(x^2) + xA_1(x^2)$. Η επιλογή των σημείων με αυτό τον τρόπο επιτρέπει τον υπολογισμό n τιμών του $A(x)$ από $n/2$ τιμές για το πολυώνυμο $A_0(x)$ και $n/2$ τιμές για το πολυώνυμο $A_1(x)$.

Συγκεκριμένα, θα υπολογίσουμε τις τιμές των πολυωνύμων $A_0(x)$ και $A_1(x)$ μόνο για τα τετράγωνα των σημείων με δείκτη μικρότερο του $n/2$, δηλ. τα σημεία $x_0^2, x_1^2, \ldots, x_{n/2-1}^2$. Αυτό μπορεί να γίνει με αναδρομική επίκληση του ίδιου αλγόριθμου επειδή τα πολυώνυμα $A_0(x)$ και $A_1(x)$ είναι βαθμού $n/2-1$ και υπολογίζουμε $n/2$ τιμές για το καθένα. Στη συνέχεια, για κάθε $i = 0, \ldots, n/2-1$, έχουμε:

$$A(x_i) = A_0(x_i^2) + x_iA_1(x_i^2) \tag{2.3}$$

και

$$A(x_{n/2+i}) = A_0(x_{n/2+i}^2) + x_{n/2+i}A_1(x_{n/2+i}^2) = A_0(x_i^2) + x_{n/2+i}A_1(x_i^2) \tag{2.4}$$

Στην τελευταία ισότητα έχουμε χρησιμοποιήσει την ιδιότητα των σημείων που έχουν επιλεγεί για τον υπολογισμό των τιμών.

Έστω $T(n)$ ο χρόνος για τον υπολογισμό n τιμών ενός πολυωνύμου $A(x)$ βαθμού $n-1$. Η μέθοδος διαίρει-και-βασίλευε υπολογίζει τις τιμές του $A(x)$ από $n/2$ τιμές δύο πολυωνύμων $A_0(x)$ και $A_1(x)$ βαθμού $n/2-1$. Αυτές οι τιμές μπορούν να υπολογιστούν αναδρομικά σε συνολικό χρόνο $2\,T(n/2)$ (δηλ. χρόνος $T(n/2)$ για τον υπολογισμό $n/2$ τιμών κάθε πολυωνύμου βαθμού $n/2-1$). Ο συνδυασμός τους για τον υπολογισμό των n τιμών του πολυωνύμου $A(x)$ μπορεί να γίνει σε χρόνο $\Theta(n)$ σύμφωνα με τις σχέσεις (2.3) και (2.4). Επομένως, η αναδρομική σχέση που περιγράφει το χρόνο εκτέλεσης του αλγόριθμου είναι $T(n) = 2\,T(n/2) + \Theta(n)$ με αρχική συνθήκη $T(1) = \Theta(1)$. Η λύση αυτής της εξίσωσης είναι $T(n) = \Theta(n \log n)$.

Συμπερασματικά, η αναδρομική εφαρμογή του διαίρει-και-βασίλευε αλγόριθμου που παρουσιάστηκε επιτρέπει τον υπολογισμό n τιμών (σε κατάλληλα επιλεγμένα σημεία) ενός πολυωνύμου βαθμού $n-1$ σε χρόνο $\Theta(n \log n)$. Για να εφαρμοστεί όμως αναδρομικά ο συγκεκριμένος αλγόριθμος πρέπει τα σημεία που επιλέγονται να έχουν τις εξής ιδιότητες: ανά δύο να έχουν κοινά τετράγωνα (για το 1ο επίπεδο αναδρομής), ανά τέσσερα να έχουν κοινές τέταρτες δυνάμεις (για το 2ο επίπεδο αναδρομής), ..., και γενικά ανά 2^j να έχουν κοινές 2^j-οστές δυνάμεις (για το επίπεδο j της αναδρομής), $j = 1, \ldots, \lfloor \log n \rfloor - 1$.

[3] Για παράδειγμα, αυτό ισχύει αν επιλέξουμε τα σημεία $1, 2, \ldots, \frac{n}{2}, -1, -2, \ldots, -\frac{n}{2}$.

Αυτό εξασφαλίζεται αν μπορεί να οριστεί η τετραγωνική ρίζα κάθε αριθμού (θετικού ή αρνητικού). Συγκεκριμένα, θεωρούμε σαν σημεία υπολογισμού το 1 στο τελευταίο (κατώτερο) επίπεδο της αναδρομής, τις τετραγωνικές του ρίζες -1 και 1 στο προτελευταίο επίπεδο, τις τετραγωνικές ρίζες των τετραγωνικών ριζών στο προ-προτελευταίο επίπεδο, κοκ. Καταφεύγοντας λοιπόν στους μιγαδικούς αριθμούς, εξασφαλίζεται ότι οι μιγαδικές ρίζες της μονάδας μπορούν να χρησιμοποιηθούν για την εφαρμογή του αλγόριθμου (βλ. Σχήμα 2.2).

Παράδειγμα 2.1. Έστω ότι θέλουμε να υπολογίσουμε 8 τιμές για το πολυώνυμο $A(x) = \sum_{i=0}^{7} a_i x^i$ βαθμού 7. Διαλέγουμε τα σημεία $p_0, \ldots, p_7$ έτσι ώστε: $p_0^2 = p_4^2, p_1^2 = p_5^2, p_2^2 = p_6^2, p_3^2 = p_7^2$ και $p_0^4 = p_2^4, p_1^4 = p_3^4$ (στη συνέχεια θα δείξουμε ότι οι μιγαδικές ρίζες της μονάδας όγδοης τάξης έχουν αυτές ακριβώς τις ιδιότητες).

Για να γίνει το παράδειγμα ευκολότερα κατανοητό παραλείπουμε τη φάση της διαίρεσης / ανάπτυξης της αναδρομής και αναφερόμαστε μόνο στη φάση του υπολογισμού και της συνθεσης των επιμέρους λύσεων. Οι οκτώ κλήσεις στο τρίτο επίπεδο / βάση της αναδρομής απλώς επιστρέφουν τους αντίστοιχους συντελεστές. Οι τέσσερις κλήσεις στο δεύτερο επίπεδο της αναδρομής υπολογίζουν τις παρακάτω τιμές:

$$\begin{array}{rclcrcl}
A_{00}(p_0^4) &=& a_0 + a_4 p_0^4 & & A_{00}(p_1^4) &=& a_0 + a_4 p_1^4 \\
A_{01}(p_0^4) &=& a_2 + a_6 p_0^4 & & A_{01}(p_1^4) &=& a_2 + a_6 p_1^4 \\
A_{10}(p_0^4) &=& a_1 + a_5 p_0^4 & & A_{10}(p_1^4) &=& a_1 + a_5 p_1^4 \\
A_{11}(p_0^4) &=& a_3 + a_7 p_0^4 & & A_{11}(p_1^4) &=& a_3 + a_7 p_1^4
\end{array}$$

Με βάση αυτές τιμές και την ιδιότητα $p_0^4 = p_2^4$ και $p_1^4 = p_3^4$, οι δύο κλήσεις στο πρώτο επίπεδο της αναδρομής θα υπολογίσουν τις τιμές:

$$\begin{aligned}
A_0(p_0^2) &= A_{00}(p_0^4) + p_0^2 A_{01}(p_0^4) = (a_0 + a_4 p_0^4) + p_0^2(a_2 + a_6 p_0^4) \\
A_0(p_1^2) &= A_{00}(p_1^4) + p_1^2 A_{01}(p_1^4) = (a_0 + a_4 p_1^4) + p_1^2(a_2 + a_6 p_1^4) \\
A_0(p_2^2) &= A_{00}(p_0^4) + p_2^2 A_{01}(p_0^4) = (a_0 + a_4 p_2^4) + p_2^2(a_2 + a_6 p_2^4) \\
A_0(p_3^2) &= A_{00}(p_1^4) + p_3^2 A_{01}(p_1^4) = (a_0 + a_4 p_3^4) + p_3^2(a_2 + a_6 p_3^4) \\
A_1(p_0^2) &= A_{10}(p_0^4) + p_0^2 A_{11}(p_0^4) = (a_1 + a_5 p_0^4) + p_0^2(a_3 + a_7 p_0^4) \\
A_1(p_1^2) &= A_{10}(p_1^4) + p_1^2 A_{11}(p_1^4) = (a_1 + a_5 p_1^4) + p_1^2(a_3 + a_7 p_1^4) \\
A_1(p_2^2) &= A_{10}(p_0^4) + p_2^2 A_{11}(p_0^4) = (a_1 + a_5 p_2^4) + p_2^2(a_3 + a_7 p_2^4) \\
A_1(p_3^2) &= A_{10}(p_1^4) + p_3^2 A_{11}(p_1^4) = (a_1 + a_5 p_3^4) + p_3^2(a_3 + a_7 p_3^4)
\end{aligned}$$

Με βάση αυτές τιμές και την ιδιότητα $p_0^2 = p_4^2$, $p_1^2 = p_5^2$, $p_2^2 = p_6^2$, και $p_3^2 = p_7^2$, η αρχική κλήση

(μηδενικό επίπεδο της αναδρομής) θα υπολογίσει τις τιμές του $A(x)$ που χρειαζόμαστε:

$$\begin{aligned}
A(p_0) &= A_0(p_0^2) + p_0 A_1(p_0^2) \\
A(p_4) &= A_0(p_0^2) + p_4 A_1(p_0^2) = A_0(p_4^2) + p_4 A_1(p_4^2) \\
A(p_1) &= A_0(p_1^2) + p_1 A_1(p_1^2) \\
A(p_5) &= A_0(p_1^2) + p_5 A_1(p_1^2) = A_0(p_5^2) + p_5 A_1(p_5^2) \\
A(p_2) &= A_0(p_2^2) + p_2 A_1(p_2^2) \\
A(p_6) &= A_0(p_2^2) + p_6 A_1(p_2^2) = A_0(p_6^2) + p_6 A_1(p_6^2) \\
A(p_3) &= A_0(p_3^2) + p_3 A_1(p_3^2) \\
A(p_7) &= A_0(p_3^2) + p_7 A_1(p_3^2) = A_0(p_7^2) + p_7 A_1(p_7^2)
\end{aligned}$$

Παρατηρούμε ότι για κάθε επίπεδο της αναδρομής, ο αλγόριθμος υπολογίζει συνολικά 8 τιμές (n τιμές στη γενική περίπτωση). Τα επίπεδα της αναδρομής είναι 3 ($\log n$ στη γενική περίπτωση) επειδή ο βαθμός των πολυωνύμων υποδιπλασιάζεται σε κάθε επίπεδο της αναδρομής (7 στο μηδενικό επίπεδο / αρχική κλήση, 3 στο πρώτο, 1 στο δεύτερο, και 0 στο τρίτο επίπεδο / βάση της αναδρομής). Από αυτή την παρατήρηση και από το γεγονός ότι κάθε τιμή υπολογίζεται με έναν πολλαπλασιασμό και μία πρόσθεση, προκύπτει ότι ο χρόνος εκτέλεσης του αλγόριθμου $\Theta(n)$ για κάθε επίπεδο επί $\Theta(\log n)$ επίπεδα $= \Theta(n \log n)$. □

2.4.2 Μιγαδικές Ρίζες της Μονάδας

Οι n-οστές μιγαδικές ρίζες της μονάδας είναι το σύνολο των μιγαδικών αριθμών που αποτελούν τις λύσεις της εξίσωσης $x^n = 1$. Με άλλα λόγια, κάθε μιγαδικός αριθμός ω τέτοιος ώστε $\omega^n = 1$ αποτελεί μία n-οστή ρίζα της μονάδας. Υπάρχουν ακριβώς n μιγαδικές n-οστές ρίζες της μονάδας, οι αριθμοί $e^{2\pi ik/n}$, $k = 0, 1, \ldots, n-1$, όπου $e = 2.71\ldots$ η βάση των φυσικών λογαρίθμων, $\pi = 3.14\ldots$, και $i^2 = -1$. Από τον εκθετικό ορισμό των μιγαδικών αριθμών

$$e^{iu} = \cos(u) + i\ \sin(u)$$

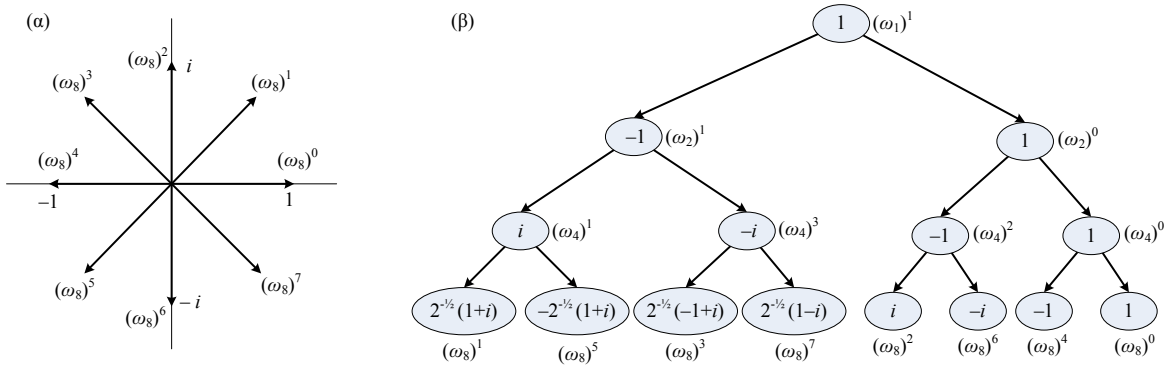
προκύπτει ότι οι n-οστές μιγαδικές ρίζες της μονάδας είναι οι αριθμοί $\cos(2\pi k/n) + i\ \sin(2\pi k/n)$, $k = 0, 1, \ldots, n-1$.

Οι μιγαδικές ρίζες της μονάδας κατανέμονται ομοιόμορφα σε έναν κύκλο με μοναδιαία ακτίνα και κέντρο την αρχή των αξόνων του μιγαδικού επιπέδου (Σχήμα 2.2). Η τιμή $\omega_n = e^{2\pi i/n}$ ονομάζεται *κύρια* n-οστή ρίζα της μονάδας, και όλες οι υπόλοιπες n-οστές ρίζες μπορούν να γραφούν σαν δυνάμεις της ω_n. Έτσι το σύνολο των n-οστών ριζών της μονάδας γράφεται σαν $\{\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}\}$.

Το σύνολο των n-οστών ριζών της μονάδας είναι κλειστό ως προς τον πολλαπλασιασμό, δηλαδή το γινόμενο δύο n-οστών ριζών της μονάδας είναι επίσης n-οστή ρίζα της μονάδας. Συγκεκριμένα,

$$\omega_n^j \omega_n^k = \omega_n^{j+k} = \omega_n^{j+k \bmod n}$$

Η βασική ιδιότητα των n-οστών ριζών της μονάδας που χρησιμοποιούμε είναι ότι αν το n είναι δύναμη του 2, το σύνολο των τετραγώνων των n-οστών ριζών της μονάδας είναι το σύνολο



Σχήμα 2.2: (α) Οι ρίζες της μονάδας για $n = 8$ και η κατανομή τους στο μιγαδικό επίπεδο. (β) Οι ρίζες της μονάδας για $n = 1, 2, 4,$ και 8. Η κύρια ρίζα απεικονίζεται πάντα αριστερά. Οι ρίζες-παιδιά ω_n^k και $\omega_n^{n/2+k}$ κάθε κόμβου $\omega_{n/2}^k$ είναι οι δύο τετραγωνικές του ρίζες. Για κάθε k, $0 \leq k \leq n/2$, έχουμε (i) $\omega_n^k = -\omega_n^{n/2+k}$ (οι ρίζες-παιδιά του $\omega_{n/2}^k$ είναι αντίθετοι αριθμοί), και (ii) $(\omega_n^k)^2 = (\omega_n^{n/2+k})^2 = \omega_{n/2}^k$ (το τετράγωνο τους είναι ίσο με τη ρίζα-πατέρα). Επομένως, οι n-οστές ρίζες της μονάδας ικανοποιούν την ιδιότητα που απαιτείται για τον αναδρομικό αλγόριθμο της Ενότητας 2.4.1.

των $\frac{n}{2}$-οστών ριζών της μονάδας. Συγκεκριμένα, ισχύει ότι

$$\{(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2\} = \{\omega_{n/2}^0, \ldots, \omega_{n/2}^{n/2-1}\}$$

Για να αποδείξουμε τον παραπάνω ισχυρισμό, παρατηρούμε ότι για κάθε ακέραιο $k = 0, \ldots, n/2 - 1$, αφενός

$$(\omega_n^k)^2 = (e^{2\pi ik/n})^2 = e^{2\pi ik/(n/2)} = \omega_{n/2}^k$$

και αφετέρου

$$(\omega_n^{n/2+k})^2 = \omega_n^{2k+n} = \omega_n^n \omega_n^{2k} = \omega_n^{2k} = e^{2\pi i 2k/n} = e^{2\pi ik/(n/2)} = \omega_{n/2}^k$$

Δηλαδή για κάθε ακέραιο $k = 0, \ldots, n/2 - 1$,

$$(\omega_n^k)^2 = (\omega_n^{n/2+k})^2 = \omega_{n/2}^k$$

Αυτή η ιδιότητα των n-οστών ριζών της μονάδας εγγυάται ότι αν το n είναι δύναμη του 2, μπορούμε να επιλέξουμε ένα σύνολο (μιγαδικών) αριθμών ώστε ανά 2^j να έχουν κοινές 2^j-οστές δυνάμεις, για κάθε $j = 1, \ldots \log n$. Έτσι μπορούμε να υπολογίζουμε τις τιμές ενός πολυωνύμου βαθμού $n-1$ στις n-οστές ρίζες της μονάδας σε χρόνο $\Theta(n \log n)$ χρησιμοποιώντας τον αλγόριθμο διαίρει-και-βασίλευε που παρουσιάστηκε στην Ενότητα 2.4.1.

2.5 Διακριτός Μετασχηματισμός Fourier και FFT

Ο Διακριτός Μετασχηματισμός Fourier (Discrete Fourier Transform - DFT) ενός διανύσματος $A = [a_0, \ldots, a_{n-1}]$ με n στοιχεία είναι το διάνυσμα $Y = [y_0, \ldots, y_{n-1}]$ όπου $y_k = \sum_{j=0}^{n-1} a_j \omega_n^{kj}$

για κάθε $k = 0, \ldots, n-1$. Με απλά λόγια ο Διακριτός Μετασχηματισμός Fourier του διανύσματος A αποτελείται από τις τιμές του πολυωνύμου $A(x) = \sum_{j=0}^{n-1} a_j x^j$ στις μιγαδικές n-οστές ρίζες της μονάδας. Συγκεκριμένα, το k-οστό στοιχείο του Διακριτού Μετασχηματισμού Fourier είναι η τιμή του πολυωνύμου στην k στη σειρά n-οστή ρίζα της μονάδας, δηλαδή στο σημείο $e^{2\pi ik/n}$. Τυπικά, $y_k = A(e^{2\pi ik/n}) = A(\omega_n^k)$, για κάθε $k = 0, \ldots, n-1$.

Ο Διακριτός Μετασχηματισμός Fourier έχει πληθώρα σημαντικών εφαρμογών. Στα πλαίσια του μαθήματος δεν θα ασχοληθούμε με τις εφαρμογές του Διακριτού Μετασχηματισμού Fourier αλλά μόνο με τον υπολογισμό του με τη μέθοδο διαίρει-και-βασίλευε. Σε όλόκληρη την ενότητα υποθέτουμε ότι το n είναι δύναμη του 2.

2.5.1 Γρήγορος Μετασχηματισμός Fourier

Για τον υπολογισμό του Διακριτού Μετασχηματισμού Fourier ενός διανύσματος A με n στοιχεία, το διάνυσμα θεωρείται ότι περιέχει τους συντελεστές του πολυώνυμου $A(x) = \sum_{j=0}^{n-1} a_j x^j$ με βαθμό $n-1$. Ο μετασχηματισμός αποτελείται από τις τιμές του πολυωνύμου $A(x)$ στις n-οστές ρίζες της μονάδας και μπορεί να υπολογισθεί με τον διαίρει-και-βασίλευε αλγόριθμο της Ενότητας 2.4.1.

Στην πραγματικότητα, αυτός ο αλγόριθμος ονομάζεται *Γρήγορος Μετασχηματισμός Fourier* (Fast Fourier Transform - FFT) και διατυπώθηκε από τους Cooley και Tukey, 1965, για τον υπολογισμό του Διακριτού Μετασχηματισμού Fourier. Ακολουθεί μια αναδρομική υλοποίηση του αλγόριθμου σε ψευδοκώδικα[4]:

```
FFT(A[a_0, ..., a_{n-1}])   /* Το n είναι δύναμη του 2 */
    if n = 1 then return(a_0);
    ω_n ← e^{2πi/n}; ω ← 1;
    A_0 ← [a_0, a_2, ..., a_{n-2}]; A_1 ← [a_1, a_3, ..., a_{n-1}];   /* Διαίρεση σε υπο-στιγμιότυπα */
    Y_0 ← FFT(A_0); Y_1 ← FFT(A_1);   /* Αναδρομική επίλυση υπο-στιγμιοτύπων */
    for k ← 0 to n/2 − 1 do   /* Σύνθεση λύσης από λύσεις υπο-στιγμιοτύπων */
        y_k ← Y_0[k] + ωY_1[k]; y_{n/2+k} ← Y_0[k] − ωY_1[k]; ω ← ω × ω_n;
    return([y_0, ..., y_{n-1}]);
```

Στην Ενότητα 2.4.1 έχουμε εξηγήσει τη βασική ιδέα πίσω από τον αλγόριθμο FFT και έχουμε αποδείξει ότι έχει χρόνο εκτέλεσης $\Theta(n \log n)$.

---

[4] Στην παρακάτω υλοποίηση, η μεταβλητή ω_n είναι ίση με την κύρια n-οστή ρίζα της μονάδας. Στην k-οστή επανάληψη του for-loop, $k = 0, \ldots, n/2-1$, η μεταβλητή ω είναι ίση με την k στη σειρά n-οστή ρίζα της μονάδας ω_n^k. Στην k-οστή επανάληψη του for-loop, χρησιμοποιούνται οι ιδιότητες:
(α) Η $n/2+k$ στη σειρά n-οστή ρίζα της μονάδας $\omega_n^{n/2+k}$ είναι ίση με $-\omega_n^k$ (δηλαδή τον αντίθετο της k στη σειρά ρίζας). Πράγματι,

$$\omega_n^{n/2+k} = e^{2\pi i(n/2+k)/n} = e^{2\pi ik/n} e^{\pi i} = \omega_n^k(\cos(\pi) + i\,\sin(\pi)) = -\omega_n^k$$

όπου χρησιμοποιήσαμε $\cos(\pi) = -1$ και $\sin(\pi) = 0$. Έτσι χρησιμοποιούμε το $-\omega$ για να υπολογίσουμε την τιμή στο σημείο $\omega_n^{n/2+k}$.
(β) Είναι

$$\omega_n^k \times \omega_n = e^{2\pi ik/n} e^{2\pi i/n} = e^{2\pi i(k+1)/n} = \omega_n^{k+1}$$

Συνεπώς, η μεταβλητή ω έχει την τιμή που πρέπει στην επόμενη επανάληψη.

Άσκηση 2.6. Χρησιμοποιώντας τον αλγόριθμο FFT, να υπολογίσετε το Διακριτό Μετασχηματισμό Fourier των διανυσμάτων $A = [1,0,0,1,0,0,0,1]$, $B = [1,4,2,-3,4,5,-2,1]$, και $C = [1,0,0,0,0,0,0,0,1,0,0,-1,0,0,1,1]$.

Λύση. Εφαρμόζουμε τη μεθοδολογία του Παραδείγματος 2.1 για τα διανύσματα A, B, και C και τις μιγαδικές ρίζες της μονάδας αντίστοιχης τάξης.

Για τα A και B, θα χρησιμοποιήσουμε τις ρίζες όγδοης τάξης. Δηλαδή θα είναι $p_0 = 1, p_1 = \frac{\sqrt{2}}{2}(1+i), p_2 = i, p_3 = \frac{\sqrt{2}}{2}(-1+i), p_4 = -1, p_5 = -\frac{\sqrt{2}}{2}(1+i), p_6 = -i$, και $p_7 = \frac{\sqrt{2}}{2}(1-i)$. Ακολουθώντας το Παράδειγμα 2.1, βρίσκουμε τα αποτελέσματα των αναδρομικών κλήσεων του αλγόριθμου FFT. Τελικά, ο Διακριτός Μετασχηματισμός Fourier του διανύσματος A είναι $[3, 1, 1-2i, 1, -1, 1, 1+2i, 1]$, και του B είναι $[12, -0.879+0.464i, 5+11i, -5.121-7.536i, -2, -5.121+7.536i, 5-11i, -0.879-0.464i]$.

Με παρόμοια μεθοδολογία και χρησιμοποιώντας τις ρίζες της μονάδας 16ης τάξης, βρίσκουμε ότι ο Διακριτός Μετασχηματισμός Fourier του διανύσματος C είναι $[3, 2.014-0.166i, 3.414-2.414i, -1.248-2.014i, 1, -0.166-0.599i, 0.586-0.414i, -0.599+1.248i, 3, -0.599-1.248i, 0.586+0.414i, -0.166+0.599i, 1, -1.248+2.014i, 3.414+2.414i, 2.014+0.166i]$. □

2.5.2 Παρεμβολή και Αντίστροφος Μετασχηματισμός Fourier

Έχοντας λύσει το πρόβλημα υπολογισμού τιμών στις n-οστές ρίζες της μονάδας σε χρόνο $\Theta(n \log n)$, απομένει να λύσουμε το πρόβλημα της παρεμβολής στις n-οστές ρίζες της μονάδας σε χρόνο $\Theta(n \log n)$ ώστε να ολοκληρώσουμε τον αλγόριθμο πολλαπλασιασμού πολυωνύμων που περιγράφεται στο Σχήμα 2.1.

Για να υπολογίσουμε το διάνυσμα των συντελεστών $A = [a_0, a_1, \ldots, a_{n-1}]$ ενός πολυωνύμου $A(x)$ από το διάνυσμα των τιμών του στις $Y = [y_0, y_1, \ldots, y_{n-1}]$ n-οστές ρίζες της μονάδας, πρέπει να λύσουμε το σύστημα εξισώσεων

$$y_k = \sum_{j=0}^{n-1} a_j \omega_n^{kj} \quad k = 0, \ldots, n-1$$

ως προς τους αγνώστους $a_0, a_1, \ldots, a_{n-1}$. Το πρόβλημα της παρεμβολής συνίσταται λοιπόν στην επίλυση του συστήματος $Y = V \times A$ με n εξισώσεις και n αγνώστους, όπου A είναι το διάνυσμα των αγνώστων συντελεστών του πολυωνύμου, Y είναι το διάνυσμα των τιμών του πολυωνύμου στις n-οστές ρίζες της μονάδας, και V είναι ο πίνακας $n \times n$ με στοιχεία $V[k,j] = \omega_n^{kj}$, $k, j \in \{0, 1, \ldots, n-1\}$.

Το διάνυσμα A προκύπτει από το γινόμενο $V^{-1} \times Y$, όπου V^{-1} είναι ο αντίστροφος του πίνακα V. Η μορφή του αντίστροφου πίνακα V^{-1} προκύπτει από την ακόλουθη πρόταση.

Πρόταση 2.1. *Έστω V ο πίνακας $n \times n$ με στοιχεία $V[k,j] = \omega_n^{kj}$, $k, j \in \{0, 1, \ldots, n-1\}$. Τα στοιχεία του αντίστροφου πίνακα V^{-1} είναι $V^{-1}[k,j] = \omega_n^{-kj}/n$, $k, j \in \{0, 1, \ldots, n-1\}$.*

Απόδειξη. Θα αποδείξουμε ότι το γινόμενο $V \times V^{-1}$ είναι ίσο με το μοναδιαίο πίνακα. Για κάθε $k, j \in \{0, 1, \ldots, n-1\}$, είναι

$$(V \times V^{-1})[k,j] = \sum_{\ell=0}^{n-1} \omega_n^{k\ell} \omega_n^{-\ell j}/n = \frac{1}{n} \sum_{\ell=0}^{n-1} \omega_n^{\ell(k-j)} = \frac{1}{n} \sum_{\ell=0}^{n-1} (\omega_n^{(k-j)})^{\ell} \tag{2.5}$$

Για τα διαγώνια στοιχεία (δηλ. όταν $k = j$) ισχύει ότι $\omega_n^{k-k} = \omega_n^0 = 1$ και $(V \times V^{-1})[k,k] = 1$ για κάθε $k = 0, 1, \ldots, n-1$. Για τα μη-διαγώνια στοιχεία είναι $k \neq j$ και $-(n-1) \leq k-j \leq n-1$. Επομένως, η ποσότητα $k-j$ δεν είναι πολλαπλάσιο του n και $\omega_n^{k-j} \neq 1$. Το τελευταίο άθροισμα στην (2.5) αποτελείται από τους n όρους γεωμετρικής προόδου με πρώτο όρο $(\omega_n^{k-j})^0 = 1$ και λόγο ω_n^{k-j}. Εφαρμόζοντας τον τύπο για το άθροισμα των n πρώτων όρων γεωμετρικής προόδου, έχουμε:

$$\sum_{\ell=0}^{n-1} (\omega_n^{(k-j)})^\ell = \frac{(\omega_n^{k-j})^n - 1}{\omega_n^{k-j} - 1} = \frac{(\omega_n^n)^{k-j} - 1}{\omega_n^{k-j} - 1} = \frac{1^{k-j} - 1}{\omega_n^{k-j} - 1} = 0$$

Δηλαδή αποδείξαμε ότι όλα τα διαγώνια στοιχεία του πίνακα $V \times V^{-1}$ είναι ίσα με 1 και όλα τα μη-διαγώνια στοιχεία είναι ίσα με 0. Με άλλα λόγια, ο $V \times V^{-1}$ είναι ο μοναδιαίος πίνακας και ο V^{-1} είναι ο αντίστροφος του V.

Επομένως, οι άγνωστοι συντελεστές του πολυωνύμου $A(x)$ δίνονται από τον τύπο $a_k = (1/n) \sum_{j=0}^{n-1} y_j \omega_n^{-kj}$, $k = 0, \ldots, n-1$. Δηλαδή, οι συντελεστές του $A(x)$ προκύπτουν από τον υπολογισμό των τιμών του πολυώνυμου με συντελεστές $[y_0/n, \ldots, y_{n-1}/n]$ στους αντίστροφους των n-οστών ριζών της μονάδας $\{\omega_n^0, \omega_n^{-1}, \ldots, \omega_n^{-(n-1)}\}$.

Αυτό το σύνολο ταυτίζεται με το σύνολο των n-οστών ριζών της μονάδας, αφού $\omega_n^{-k} = \omega_n^n \, \omega_n^{-k} = \omega_n^{n-k}$ για κάθε $k \in \{0, 1, \ldots, n-1\}$. Επομένως, το πρόβλημα της παρεμβολής στους αντίστροφους των n-οστών ριζών της μονάδας μπορεί να λυθεί σε χρόνο $\Theta(n \log n)$ από τον αλγόριθμο FFT αν αντικαταστήσουμε την κύρια n-οστή ρίζα της μονάδας ω_n με τον αντίστροφό της ω_n^{-1}. Ο τροποποιημένος αλγόριθμος καλείται με είσοδο το διάνυσμα $Y/n = [y_0/n, \ldots, y_{n-1}/n]$ και επιστρέφει το διάνυσμα των συντελεστών του πολυωνύμου βαθμού $n-1$ που έχει τιμές $y_0, \ldots, y_{n-1}$ στις n-οστές ρίζες της μονάδας. Ο τροποποιημένος αλγόριθμος είναι γνωστός σαν Αντίστροφος Μετασχηματισμός Fourier (Inverse Fourier Transform).

Χρησιμοποιώντας λοιπόν την ορθή και την αντίστροφη μορφή του FFT, μπορούμε να μετατρέψουμε την αναπαράσταση με συντελεστές δύο πολυωνύμων βαθμού $n-1$ σε αναπαράσταση με ζεύγη σημείων-τιμών σε χρόνο $\Theta(n \log n)$, να υπολογίσουμε το γινόμενο των δύο πολυωνύμων σε χρόνο $\Theta(n)$, και να ανακτήσουμε τους συντελεστές του γινομένου σε χρόνο $\Theta(n \log n)$.

Άσκηση 2.7. Να εφαρμόσετε τον Αντίστροφο Μετασχηματισμό Fourier στα αποτελέσματα του Διακριτού Μετασχηματισμού Fourier για τα διανύσματα της Άσκησης 2.6 και να επαληθεύσετε ότι προκύπτουν τα αρχικά διανύσματα.

Άσκηση 2.8. Έστω $A(x) = 3 - x$ και $B(x) = 2 + 3x$ δύο πολυώνυμα πρώτου βαθμού. Περιγράψτε αναλυτικά τον υπολογισμό του γινομένου $C(x) = A(x)B(x)$ ακολουθώντας τον αλγόριθμο του Σχήματος 2.1 και χρησιμοποιώντας τον FFT για τη μετατροπές των αναπαραστάσεων.

Λύση. Το γινόμενο $C(x)$ των πολυωνύμων $A(x)$ και $B(x)$ είναι βαθμού 2. Χρειαζόμαστε λοιπόν τουλάχιστον 3 ζεύγη σημείων-τιμών για να προσδιορίσουμε το $C(x)$. Επειδή ο αλγόριθμος FFT εφαρμόζεται μόνο για δυνάμεις του 2, θα υπολογίσουμε τις τιμές των πολυωνύμων σε 4 σημεία ($n = 4$).

Τα διανύσματα συντελεστών είναι $A = [3, -1, 0, 0]$ και $B = [2, 3, 0, 0]$. Οι μιγαδικές ρίζες της μονάδας για $n = 4$ είναι $\{1, i, -1, -i\}$ με κύρια ρίζα το $\omega_4 = i$. Οι τετραγωνικές ρίζες της μονάδας είναι $\{1, -1\}$ με κύρια ρίζα την $\omega_2 = -1$.

Για το πολυώνυμο $A(x)$, ο FFT καλείται με παράμετρο $[3,-1,0,0]$, θέτει $A_0 = [3,0]$, $A_1 = [-1,0]$, $\omega_4 = i$, και κάνει τις αναδρομικές κλήσεις και FFT($[-1,0]$).

Η αναδρομική κλήση FFT($[3,0]$) θέτει $\omega_2 = -1$, και καλεί αναδρομικά FFT($[3]$) και FFT($[0]$). Αυτές οι κλήσεις επιστρέφουν $Y_0 = [3]$ και $Y_1 = [0]$, οπότε ο αλγόριθμος υπολογίζει τα $y_0 = 3 + 1 \times 0 = 3$ και $y_1 = 3 - 1 \times 0 = 3$. Επομένως, η κλήση FFT($[3,0]$) επιστρέφει $[3,3]$.

Ομοίως, η κλήση FFT($[-1,0]$) υπολογίζει τα $y_0 = -1+1\times 0 = -1$ και $y_1 = -1-1\times 0 = -1$, και επιστρέφει $[-1,-1]$.

Επιστρέφουμε στην βασική κλήση του αλγορίθμου FFT($[3,-1,0,0]$). Για $k = 0$, ο αλγόριθμος υπολογίζει $y_0 = 3 + 1 \times (-1) = 2$, και $y_2 = 3 + (-1) \times (-1) = 4$. Για $k = 1$, ο αλγόριθμος υπολογίζει $y_1 = 3 + i \times (-1) = 3 - i$, και $y_3 = 3 + (-i) \times (-1) = 3 + i$. Επομένως, ο αλγόριθμος επιστρέφει το $Y_A = [2, 3-i, 4, 3+i]$. Είναι εύκολο να επιβεβαιώσουμε ότι το $Y_A = [A(1), A(i), A(-1), A(-i)]$, δηλαδή αποτελείται από τις τιμές του $A(x)$ στις ρίζες της μονάδας τέταρτης τάξης $\{1, i, -1, -i\}$.

Με παρόμοιο τρόπο η κλήση FFT($[2,3,0,0]$) για το πολυώνυμο $B(x)$ επιστρέφει το $Y_B = [5, 2+3i, -1, 2-3i]$.

Οι τιμές του γινομένου $C(x)$ στις μιγαδικές ρίζες της μονάδας είναι $Y_C = [10, 9+7i, -4, 9-7i]$. Ο αντίστροφος FFT καλείται με παράμετρο το διάνυσμα $Y_C/4 = [5/2, \frac{9+7i}{4}, -1, \frac{9-7i}{4}]$ και πραγματοποιεί τις αναδρομικές κλήσεις FFT($[5/2,-1]$) και FFT($[\frac{9+7i}{4}, \frac{9-7i}{4}]$). Η αναδρομική κλήση FFT($[5/2,-1]$) επιστρέφει $Y_0 = [3/2, 7/2]$. Η κλήση FFT($[\frac{9+7i}{4}, \frac{9-7i}{4}]$) επιστρέφει $Y_1 = [9/2, 7i/2]$.

Για $k = 0$, η βασική κλήση του αλγορίθμου υπολογίζει $y_0 = 3/2 + 1^{-1} \times 9/2 = 6$ και $y_2 = 3/2 + (-1)^{-1} \times 9/2 = -3$. Για $k = 1$, ο αλγόριθμος υπολογίζει $c_1 = 7/2 + i^{-1} \times 7i/2 = 7$, και $c_3 = 7/2 + (-i)^{-1}7i/2 = 0$. Έτσι παίρνουμε το διάνυσμα των συντελεστών του γινομένου $C = [6, 7, -3, 0]$ που αντιστοιχεί στο πολυώνυμο $C(x) = 6 + 7x - 3x^2$. □

2.6 Βιβλιογραφικές Αναφορές

Η μέθοδος διαίρει-και-βασίλευε καλύπτεται πολύ καλά στα [7] και [11]. Ο αλγόριθμος πολλαπλασιασμού αριθμών είναι των Karatsuba και Ofman [25]. Ο αλγόριθμος πολλαπλασιασμού πινάκων είναι του Strassen [39]. Ο πιο γρήγορος (ασυμπτωτικά) γνωστός αλγόριθμος πολλαπλασιασμού πινάκων είναι των Coppersmith και Winograd [10]. Το παράδειγμα από την κρυπτογραφία που δείχνει την αναγκαιότητα της ύψωσης μεγάλων αριθμών σε μεγάλες δύναμεις είναι των Diffie και Hellman [13]. Ο αλγόριθμος FFT είναι των Cooley και Tukey [9].

3 Δυναμικός Προγραμματισμός

Η μέθοδος του *δυναμικού προγραμματισμού* (dynamic programming), όπως και η μέθοδος διαίρει-και-βασίλευε, επιλύει ένα στιγμιότυπο συνδυάζοντας τις λύσεις κατάλληλα επιλεγμένων υπο-στιγμιοτύπων. Για την επίλυση ενός προβλήματος, η μέθοδος διαίρει-και-βασίλευε διαιρεί το στιγμιότυπο εισόδου σε υπο-στιγμιότυπα, επιλύει τα υπο-στιγμιότυπα *ανεξάρτητα* (συνήθως με αναδρομική επίκληση του ίδιου αλγόριθμου), και υπολογίζει τη λύση του αρχικού στιγμιότυπου συνθέτοντας τις λύσεις των υπο-στιγμιοτύπων. Σε πολλά προβλήματα, η φάση της διαίρεσης οδηγεί σε ίδια ή επικαλυπτόμενα υπο-στιγμιότυπα. Η ανεξάρτητη επίλυσή τους σημαίνει ότι κατά για τον υπολογισμό της λύσης του αρχικού στιγμιότυπου θα πρέπει να επιλύσουμε πολλές φορές το ίδιο υπο-στιγμιότυπο από την αρχή. Αυτό είναι σπατάλη υπολογιστικού χρόνου και οδηγεί σε μη-αποδοτικούς αλγόριθμους (βλ. Άσκηση 2.1).

Από την άλλη πλευρά, μπορούμε να εκμεταλλευτούμε την επανάληψη των υπο-στιγμιοτύπων προς όφελός μας λύνοντας κάθε υπο-στιγμιότυπο μία μόνο φορά και αποθηκεύοντας το αποτέλεσμα για μελλοντική χρήση. Αυτή είναι η βασική ιδέα πίσω από τη μέθοδο του δυναμικού προγραμματισμού: Υπολογίζουμε κάθε λύση μία φορά και την αποθηκεύουμε σε έναν πίνακα που συμπληρώνεται καθώς επιλύουμε όλο και μεγαλύτερα υπο-στιγμιότυπα. Ο δυναμικός προγραμματισμός εφαρμόζεται από τη βάση προς την κορυφή (bottom-up). Ξεκινά με την επίλυση των στοιχειωδών στιγμιοτύπων και συνεχίζει με όλο και μεγαλύτερα στιγμιότυπα, των οποίων τις λύσεις συνθέτει από τις λύσεις που έχει ήδη υπολογίσει. Η μέθοδος τερματίζει όταν υπολογίσει τη λύση του αρχικού στιγμιότυπου. Ο όρος ''προγραμματισμός'' περιγράφει τη διαδικασία υπολογισμού / συμπλήρωσης των στοιχείων ενός πίνακα και όχι τη δημιουργία ενός προγράμματος για εκτέλεση σε υπολογιστή.

Τόσο η μέθοδος διαίρει-και-βασίλευε όσο και η μέθοδος του δυναμικού προγραμματισμού εφαρμόζονται σε προβλήματα που μπορούν να επιλυθούν *αναδρομικά* (recursively) με ''σύνθεση'' της λύσης ενός στιγμιότυπου από τις λύσεις (κατάλληλα επιλεγμένων) υπο-στιγμιοτύπων. Όμως οι δύο μέθοδοι ακολουθούν διαφορετική στρατηγική. Η μέθοδος διαίρει-και-βασίλευε εφαρμόζεται από την κορυφή προς τη βάση (top-down), ''επιτίθεται'' δηλαδή απευθείας στο αρχικό στιγμιότυπο επιχειρώντας *κατάλληλη* διαίρεση. Το διαίρει-και-βασίλευε έχει καλά αποτελέσματα όταν κάθε στιγμιότυπο μπορεί να διαιρεθεί σε *ανεξάρτητα* υπο-στιγμιότυπα (βλ. αλγόριθμους ταξινόμησης merge-sort και quicksort και αλγόριθμο FFT).

Ο δυναμικός προγραμματισμός εφαρμόζεται από τη βάση προς την κορυφή (bottom-up), επιχειρεί δηλαδή να ''κτίσει'' τη λύση του αρχικού στιγμιότυπου από τις λύσεις των υπο-στιγμιοτύπων. Ο δυναμικός προγραμματισμός εφαρμόζεται όταν τα υπο-στιγμιότυπα *δεν είναι ανεξάρτητα*, αλλά επαναλαμβάνονται ή είναι επικαλυπτόμενα. Αυτό το χαρακτηριστικό ονομάζεται *ιδιότητα των*

επικαλυπτόμενων υπο-στιγμιοτύπων (overlapping sub-instances). Οι αλγόριθμοι δυναμικού προγραμματισμού επιλύουν μία φορά κάθε υπο-στιγμιότυπο και αποθηκεύουν τις λύσεις σε έναν πίνακα για μελλοντική χρήση. Αυτό το χαρακτηριστικό κάνει τους αλγορίθμους δυναμικού προγραμματισμού ιδιαίτερα απαιτητικούς σε αποθηκευτικό χώρο.

Παράδειγμα 3.1 (Διωνυμικοί Συντελεστές - Τρίγωνο του Pascal). Ας ξεκινήσουμε με το παράδειγμα των δυωνυμικών συντελεστών που θυμίζει την Άσκηση 2.1. Οι διωνυμικοί συντελεστές δίνουν τους διαφορετικούς συνδυασμούς k από n διαφορετικά αντικείμενα:

$$C(n,k) = \binom{n}{k} = \frac{n!}{k!(n-k)!} \quad n \in \mathbb{N},\ 0 \leq k \leq n$$

Οι διωνυμικοί συντελεστές είναι οι συντελεστές των μονονύμων x^k στο ανάπτυγμα του διωνύμου $(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$ (από αυτό προκύπτει και το όνομά τους).

Το $C(n,k)$ μπορεί να υπολογιστεί από την αναδρομική σχέση

$$C(n,k) = \begin{cases} C(n-1,k-1) + C(n-1,k) & \text{αν } 0 < k < n \\ 1 & \text{διαφορετικά} \end{cases} \tag{3.1}$$

Ο αντίστοιχος αλγόριθμος σε ψευδοκώδικα είναι:

```
BinomialDC(n, k)
    if k = 0 or k = n then return(1);
    else return(BinomialDC(n − 1, k − 1) + BinomialDC(n − 1, k));
```

Ο χρόνος εκτέλεσης χρειάζεται $\Omega(C(n,k))$ προσθέσεις (το $C(n,k)$ είναι $\Omega((n/e)^k)$ και $\mathrm{O}(n^k)$ - δηλαδή εκθετικό στο k) αφού το αποτέλεσμα προκύπτει από το άθροισμα $C(n,k)$ μονάδων. Ένας πρόχειρος υπολογισμός αποδεικνύει ότι ο αλγόριθμος αυτός είναι πολύ αργός ακόμη και για σχετικά μικρές τιμές των n και k (π.χ. $C(30,15) = 155117520$).

Το λεγόμενο *τρίγωνο του Pascal* αποτελεί μια αποδοτική μέθοδο για τον υπολογισμό των διωνυμικών συντελεστών. Η ιδέα είναι να διατηρούμε έναν πίνακα C με στοιχεία $C[n,k] = C(n,k)$, $0 \leq k \leq n$. Οι γραμμές του πίνακα συμπληρώνονται η μία μετά την άλλη χρησιμοποιώντας την αναδρομική εξίσωση (3.1). Ο αντίστοιχος αλγόριθμος σε ψευδοκώδικα είναι:

```
Binomial(n, k)
    for i ← 0 to n do
        C[i, 0] ← 1; C[i, i] ← 1;
        for j ← 1 to i − 1 do
            C[i, j] ← C[i − 1, j − 1] + C[i − 1, j];
    return(C[n, k]);
```

Ο χρόνος εκτέλεσης είναι $\Theta(nk)$ και απαιτούνται $\Theta(nk)$ θέσεις μνήμης (μπορείτε να τροποποιήσετε τον αλγόριθμο ώστε να χρησιμοποιεί μόνο $\Theta(k)$ θέσεις μνήμης;). Ουσιαστικά πρόκειται για μια απλή εφαρμογή της μεθόδου του δυναμικού προγραμματισμού. □

Η μέθοδος του δυναμικού προγραμματισμού εφαρμόζεται συνήθως σε προβλήματα συνδυαστικής βελτιστοποίησης που έχουν την ιδιότητα των *βέλτιστων επιμέρους λύσεων* (optimal substructures ή principle of optimality): Κάθε τμήμα μιας βέλτιστης λύσης για ένα στιγμιότυπο αποτελεί βέλτιστη λύση για το αντίστοιχο υπο-στιγμιότυπο. Πολλά σημαντικά προβλήματα έχουν αυτή την ιδιότητα (και πολλά άλλα δεν την έχουν).

Παράδειγμα 3.2 (Συντομότερα και Μακρύτερα Μονοπάτια). Από τη Θεωρία Γραφημάτων γνωρίζουμε ότι κάθε τμήμα ενός συντομότερου μονοπατιού είναι ένα συντομότερο μονοπάτι μεταξύ των άκρων του. Με άλλα λόγια, το Πρόβλημα του Συντομότερου Μονοπατιού έχει την ιδιότητα των βέλτιστων επιμέρους λύσεων. Διαισθητικά, αυτό συμβαίνει γιατί το ελάχιστο κόστος μετάβασης από μια κορυφή v σε μία κορυφή w είναι ανεξάρτητο από το ελάχιστο κόστος μετάβασης από τη w σε μια άλλη κορυφή u. Έτσι αν ενώσουμε ένα συντομότερο μονοπάτι από τη v στη w και ένα συντομότερο μονοπάτι από τη w στη v, θα πάρουμε το συντομότερο μονοπάτι από τη v στη u που διέρχεται από τη w. Παίρνοντας το συντομότερο τέτοιο μονοπάτι ως προς όλες τις ενδιάμεσες κορυφές w, θα βρούμε το συντομότερο μονοπάτι μεταξύ v και u.

Αυτό δεν ισχύει για το Πρόβλημα του Μακρύτερου Μονοπατιού. Στο Πρόβλημα του Μακρύτερου Μονοπατιού δίνονται ένα γράφημα και δύο διακεκριμένες κορυφές s και t, και ζητείται το μακρύτερο *απλό* μονοπάτι από το s στο t. Αν θεωρήσουμε ένα μακρύτερο μονοπάτι μεταξύ δύο κορυφών, τότε τα τμήματά του δεν είναι κατ' ανάγκην μακρύτερα μονοπάτια μεταξύ των άκρων τους (διατυπώστε συγκεκριμένα παραδείγματα). Διαισθητικά, αυτό συμβαίνει γιατί αν ενώσουμε ένα μακρύτερο μονοπάτι από τη v στη w και ένα μακρύτερο μονοπάτι από τη w στη v, πιθανόν να πάρουμε ένα μονοπάτι που δεν είναι απλό. □

Αντιστρέφοντας τη διατύπωση, όταν ένα πρόβλημα έχει την ιδιότητα των βέλτιστων επιμέρους λύσεων, η βέλτιστη λύση ενός στιγμιότυπου μπορεί να προκύψει από τη σύνθεση των βέλτιστων λύσεων *κατάλληλα επιλεγμένων* υπο-στιγμιότυπων. Σε αρκετά προβλήματα όμως δεν γνωρίζουμε εκ των προτέρων ποια είναι αυτά τα "κατάλληλα επιλεγμένα" υπο-στιγμιότυπα (π.χ. δεν μπορούμε να γνωρίζουμε ότι το συντομότερο μονοπάτι μεταξύ των κορυφών u και v διέρχεται από μια συγκεκριμένη κορυφή w). Αυτό καθιστά δυσχερή την εφαρμογή μιας top-down προσέγγισης, όπως αυτή της μεθόδου διαίρει-και-βασίλευε, που χρειάζεται να γνωρίζει ποια υπο-στιγμιότυπα πρέπει να λύσει για να είναι αποδοτική. Αντίθετα, επιβάλει την εφαρμογή μιας bottom-up προσέγισης, όπως ο δυναμικός προγραμματισμός, που πρώτα λύνει όλα τα διαφορετικά υπο-στιγμιότυπα και μετά αποφασίζει ποια θα συνδυαστούν για να δώσουν τη βέλτιστη λύση.

Ο σχεδιασμός ενός αλγόριθμου δυναμικού προγραμματισμού για συγκεκριμένο πρόβλημα συνδυαστικής βελτιστοποίησης ακολουθεί συνήθως τα παρακάτω βήματα:

1. Εξετάζουμε αν το πρόβλημα έχει την ιδιότητα των βέλτιστων επιμέρους λύσεων και πως οι επιμέρους λύσεις συνδυάζονται για να προκύψει η βέλτιστη λύση του αρχικού στιγμιότυπου.

2. Διατυπώνουμε την αναδρομική εξίσωση που περιγράφει την τιμή της βέλτιστης λύσης.

3. Με βάση την αναδρομική εξίσωση, υπολογίζουμε τη βέλτιστη αντικειμενική τιμή από τη βάση προς την κορυφή (εφαρμόζουμε bottom-up στρατηγική).

4. Γνωρίζοντας ποια ακριβώς υπο-στιγμιότυπα πρέπει να συνδυάσουμε, υπολογίζουμε μία βέλτιστη λύση. Το βήμα αυτό μπορεί να παραληφθεί αν ενδιαφερόμαστε μόνο για τη βέλτιστη αντικειμενική τιμή και όχι για την ίδια τη λύση.

3.1 Πολλαπλασιασμός Ακολουθίας Πινάκων

Θέλουμε να υπολογίσουμε το γινόμενο μιας ακολουθίας πινάκων χρησιμοποιώντας τον απλό αλγόριθμο που λειτουργεί με βάση τον ορισμό της πράξης (βλ. Εξίσωση 2.2 στο προηγούμενο κεφάλαιο). Για τον υπολογισμό του γινομένου ενός πίνακα $p \times q$ με έναν πίνακα $q \times r$, ο αλγόριθμος αυτός εκτελεί ακριβώς $p\,q\,r$ πολλαπλασιασμούς μεταξύ στοιχείων των πινάκων.

Έστω ότι δίνεται μια ακολουθία πινάκων $A_1, A_2, \ldots, A_n$, όπου ο πίνακας A_i είναι $d_{i-1} \times d_i$, $i = 1, 2, \ldots, n$, και θέλουμε να υπολογίσουμε το γινόμενο $A = A_1 A_2 \cdots A_n$. Ο πολλαπλασιασμός πινάκων είναι προσεταιριστική πράξη[1] και το γινόμενο μπορεί να υπολογιστεί με πολλούς διαφορετικούς τρόπους. Για παράδειγμα, μπορούμε να υπολογίσουμε:

$$\begin{aligned} A &= (\cdots((A_1 A_2)A_3)\cdots A_n) \quad \text{ή} \\ A &= (A_1(A_2(A_3 \cdots (A_{n-1}A_n)\cdots))) \quad \text{ή ακόμη} \\ A &= ((A_1A_2)(A_3A_4)\cdots(A_{n-1}A_n)) \quad \text{κοκ.} \end{aligned}$$

Πρέπει λοιπόν να χρησιμοποιήσουμε παρενθέσεις για να καθορίσουμε τη σειρά με την οποία οι πίνακες θα πολλαπλασιαστούν. Το ζήτημα είναι ότι η σειρά με την οποία θα πολλαπλασιάσουμε τους πίνακες έχει σημαντική επίδραση στον χρόνο εκτέλεσης του αλγόριθμου!

Παράδειγμα 3.3. Θεωρούμε το γινόμενο $A_1A_2A_3A_4$, όπου ο A_1 είναι 13×5, ο A_2 είναι 5×89, ο A_3 είναι 89×3, και ο A_4 είναι 3×34. Υπάρχουν πέντε διαφορετικοί τρόποι να τοποθετήσουμε παρενθέσεις / υπολογίσουμε το γινόμενο. Κάθε διαφορετική σειρά υπολογισμού απαιτεί την εκτέλεση διαφορετικού αριθμού πολλαπλασιασμών μεταξύ στοιχείων των πινάκων.

Σειρά Υπολογισμού	**Αριθμός Πολλαπλασιασμών**
$(((A_1A_2)A_3)A_4)$	$13 \times 5 \times 89 + 13 \times 89 \times 3 + 13 \times 3 \times 34 = 10582$
$((A_1A_2)(A_3A_4))$	54201
$((A_1(A_2A_3))A_4)$	2856
$(A_1((A_2A_3)A_4))$	4055
$(A_1(A_2(A_3A_4)))$	26418

Είναι εύλογο να υποθέσουμε ότι ο αριθμός των πολλαπλασιασμών καθορίζει σε σημαντικό βαθμό το χρόνο υπολογισμού του γινομένου. Βλέπουμε λοιπόν ότι η σειρά εκτέλεσης των πολλαπλασιασμών μεταξύ των πινάκων καθορίζει σε σημαντικό βαθμό το χρόνο υπολογισμού. □

Το *Πρόβλημα του Πολλαπλασιασμού μιας Ακολουθίας Πινάκων* (Matrix Chain Multiplication) είναι δεδομένης μιας ακολουθίας n πινάκων $(A_1, A_2, \ldots, A_n)$, όπου ο πίνακας A_i είναι $d_{i-1} \times d_i$,

[1] Μια πράξη $\otimes$ είναι προσεταιριστική αν για κάθε τριάδα στοιχείων x, y, z ισχύει ότι $x \otimes y \otimes z = x \otimes (y \otimes z) = (x \otimes y) \otimes z$. Πράγματι, για τον πολλαπλασιασμό πινάκων ισχύει ότι $A_1A_2A_3 = (A_1A_2)A_3 = A_1(A_2A_3)$.

$i = 1, 2, \ldots, n$, να υπολογιστεί το γινόμενο $A_1 A_2 \cdots A_n$ εκτελώντας τον ελάχιστο αριθμό πολλαπλασιασμών μεταξύ στοιχείων των πινάκων. Πρόκειται δηλαδή για ένα πρόβλημα συνδυαστικής βελτιστοποίησης. Κάθε διαφορετική σειρά υπολογισμού του γινομένου αποτελεί μια αποδεκτή λύση. Το ζητούμενο είναι η *βέλτιστη* σειρά υπολογισμού (ισοδύναμα, ο βέλτιστος τρόπος να τοποθετηθούν παρενθέσεις).

Ένας τρόπος να λύσουμε το πρόβλημα είναι η *εξαντλητική αναζήτηση* (exhaustive search): Υπολογίζουμε τον αριθμό των πολλαπλασιασμών (μεταξύ στοιχείων πινάκων) για κάθε διαφορετική σειρά υπολογισμού, και να επιλέγουμε αυτή με την ελάχιστη τιμή. Έστω $P(n)$ οι διαφορετικές σειρές υπολογισμού για n πίνακες. Μπορούμε να διαιρέσουμε (τοποθετώντας παρενθέσεις) την ακολουθία μεταξύ των πινάκων i και $i+1$, για κάποιο $i = 1, 2, \ldots, n-1$, και να τοποθετήσουμε ανεξάρτητα παρενθέσεις στις δύο επιμέρους ακολουθίες (μήκους i και $n-i$ αντίστοιχα). Επομένως, το $P(n)$ δίνεται από την αναδρομική σχέση[2]

$$P(n) = \sum_{i=1}^{n-1} P(i)P(n-i)$$

με αρχική συνθήκη $P(1) = 1$. Μπορεί να αποδειχθεί ότι $P(n) = T(n-1)$, όπου $T(n)$ είναι ο n-οστός αριθμός Catalan:

$$T(n) = \frac{1}{n+1}\binom{2n}{n} = \Omega\left(\frac{4^n}{n^{3/2}}\right)$$

Επομένως, ο αριθμός των διαφορετικών περιπτώσεων που πρέπει να ελεγχθούν είναι τεράστιος (π.χ. το $P(n)$ είναι της τάξης του 10^{15} για $n = 30$) και ο χρόνος για την εφαρμογή της εξαντλητικής αναζήτησης μπορεί να είναι πολύ μεγαλύτερος από την επιτάχυνση που θα έχουμε λόγω του βέλτιστου υπολογισμού του γινομένου.

Μια εναλλακτική προσέγγιση είναι η εφαρμογή της μεθόδου του δυναμικού προγραμματισμού. Αρχικά εξετάζουμε αν το πρόβλημα έχει την ιδιότητα των βέλτιστων επιμέρους λύσεων. Ας συμβολίσουμε με $A_{i..j}$ το επιμέρους γινόμενο $A_i \cdots A_j$. Το ζητούμενο είναι ο υπολογισμός του γινομένου $A_{1..n}$. Κάθε (βέλτιστη) λύση διαιρεί την ακολουθία μεταξύ των πινάκων i και $i+1$, για κάποιο $1 \le i < n$, υπολογίζει ανεξάρτητα τα επιμέρους γινόμενα $A_{1..i}$ και $A_{i+1..n}$, και μετά υπολογίζει το γινόμενο $A_{1..n} = A_{1..i}\, A_{i+1..n}$. Ο αριθμός των πολλαπλασιασμών στον υπολογισμό του $A_{1..n}$ είναι ίσος με $d_0 \times d_i \times d_n$ συν τον αριθμό των πολλαπλασιασμών για τον υπολογισμό των $A_{1..i}$ και $A_{i+1..n}$. Αν λοιπόν η σειρά υπολογισμού του $A_{1..n}$ είναι βέλτιστη, και η σειρά υπολογισμού των $A_{1..i}$ και $A_{i+1..n}$ πρέπει είναι βέλτιστη (αν υπήρχε καλύτερη σειρά υπολογισμού για κάποιο από τα $A_{1..i}$ και $A_{i+1..n}$, θα τη χρησιμοποιούσαμε για να βελτιώσουμε τον υπολογισμό του $A_{1..n}$.

Ας επιστρέψουμε στο Παράδειγμα 3.3. Η βέλτιστη λύση διαιρεί την ακολουθία στη θέση 3 και υπολογίζει το γινόμενο $A_{1..3}\, A_4$ εκτελώντας 1326 πολλαπλασιασμούς. Υπάρχουν δύο τρόποι να υπολογισθεί το $A_{1..3}$. Ο ένας είναι ως $A_{1..2}\, A_3$ με 9256 πολλαπλασιασμούς και ο άλλος ως $A_1\, A_{2..3}$ με 1530 πολλαπλασιασμού. Η βέλτιστη λύση χρησιμοποιεί τον βέλτιστο (δεύτερο τρόπο) υπολογισμού του $A_{1..3}$ και υπολογίζει το συνολικό γινόμενο ως $(A_1(A_2A_3))A_4$ με 2856 πολλαπλασιασμούς.

[2]Παρατηρούμε ότι κάθε διαφορετική σειρά υπολογισμού αντιστοιχεί σε ένα δυαδικό δέντρο με n φύλλα.

Με βάση την προηγούμενη παρατήρηση, μπορούμε να διατυπώσουμε την αναδρομική εξίσωση που περιγράφει τη βέλτιστη λύση. Έστω $m[i,j]$ ο *ελάχιστος* αριθμός πολλαπλασιασμών για τον υπολογισμό του γινομένου $A_{i..j}$. Για κάθε $1 \leq i \leq n$, είναι $m[i,i] = 0$ αφού $A_{i..i} = A_i$ είναι ένας μεμονωμένος πίνακας. Για να υπολογίσουμε το $A_{i..j}$ με βέλτιστο τρόπο, υπολογίζουμε με βέλτιστο τρόπο τα $A_{i..k}$ και $A_{k+1..j}$ για κάποιο k, $i \leq k < j$ και εκτελούμε τον πολλαπλασιασμό $A_{i..k}\, A_{k+1..j}$. Ο συνολικός αριθμός των πολλαπλασιασμών είναι $m[i,k] + m[k+1,j] + d_{i-1}\, d_k\, d_j$. Επειδή δεν γνωρίζουμε το καλύτερο σημείο για να διαιρέσουμε την ακολουθία $A_i, \ldots, A_j$, υπολογίζουμε τον συνολικό αριθμό πολλαπλασιασμών για όλες τις δυνατές θέσεις $k = i, i+1, \ldots, j-1$ και διαλέγουμε την καλύτερη. Επομένως, ο *ελάχιστος* αριθμός πολλαπλασιασμών για τον υπολογισμό του γινομένου $A_{i..j}$ δίνεται από την αναδρομική σχέση:

$$m[i,j] = \begin{cases} \min_{i \leq k < j}\{m[i,k] + m[k+1,j] + d_{i-1}\, d_k\, d_j\} & \text{αν } i < j \\ 0 & \text{αν } i = j \end{cases} \tag{3.2}$$

Το ζητούμενο είναι ο υπολογισμός του $m[1,n]$, δηλαδή του ελάχιστου αριθμού πολλαπλασιασμών για τον υπολογισμό του γινομένου $A_{1..n}$.

Ο υπολογισμός του $m[1,n]$ απαιτεί τον υπολογισμό των $m[i,j]$ για όλες τις τιμές των i και j, $1 \leq i < j \leq n$. Δηλαδή πρέπει να υπολογίσουμε τις βέλτιστες τιμές για $\frac{n(n-1)}{2}$ υπο-στιγμιότυπα. Η βέλτιστη τιμή $m[i,j]$ για κάθε υπο-στιγμιότυπο $A_{i..j}$ μπορεί να υπολογισθεί σε γραμμικό χρόνο αν είναι γνωστές οι βέλτιστες τιμές $m[i,k]$ και $m[k+1,j]$ για όλα τα υπο-στιγμιότυπα $A_{i..k}$ και $A_{k+1..j}$, $i \leq k < j$.

Εφαρμόζουμε τη μέθοδο του δυναμικού προγραμματισμού για τον υπολογισμό του $m[1,n]$. Εργαζόμαστε από τη βάση προς την κορυφή. Συγκεκριμένα, υπολογίζουμε τις βέλτιστες τιμές $m[i,j]$ για όλα τα στιγμιότυπα $A_{i..j}$ με $j - i \leq p - 1$ και τις χρησιμοποιούμε για να υπολογίσουμε τις βέλτιστες τιμές για τα στιγμιότυπα $A_{i..j}$ όπου $j - i = p$, $p = 2, \ldots, n-1$. Με άλλα λόγια, ξεκινάμε με τις τιμές $m[i,j]$ όπου τα i και j είναι διαδοχικοί αριθμοί, συνεχίζουμε με i και j που διαφέρουν κατά 2, 3, κοκ. Από τη σχέση (3.2) προκύπτει ότι για κάθε $1 \leq i < j \leq n$, οι τιμές $m[i,k]$ και $m[k+1,j]$, $1 \leq k < j$, έχουν υπολογιστεί πριν από την τιμή $m[i,j]$. Αυτή η μέθοδος υλοποιείται από τον παρακάτω ψευδοκώδικα:

```
MatrixChainMultiplication(d[0, 1, ..., n])  /* Το διάνυσμα d περιέχει τα μεγέθη των πινάκων */
    for i ← 1 to n do
        m[i, i] ← 0;
    for p ← 2 to n do
        for i ← 1 to n − p + 1 do
            j ← i + p − 1; m[i, j] ← ∞;
            for k ← i to j − 1 do
                q ← m[i, k] + m[k + 1, j] + d[i − 1]d[k]d[j];
                if q < m[i, j] then m[i, j] ← q;
    return(m[1, n]);
```

Σε κάθε επανάληψη του βασικού for-loop που σχετίζεται με τη μεταβλητή p, η MatrixChain-Multiplication υπολογίζει τις τιμές $m[i,j]$ για όλα τα i και j, $1 \leq i < j \leq n$, που διαφέρουν κατά $p-1$ (δηλ. $j - i = p - 1$). Το δεύτερο for-loop που σχετίζεται με τη μεταβλητή i εκτελείται

$\frac{n(n-1)}{2}$ φορές. Ο χρόνος εκτέλεσης κάθε επανάληψης είναι $O(n)$. Επομένως, ο συνολικός χρόνος εκτέλεσης του αλγορίθμου είναι $O(n^3)$. Επίσης, ο αλγόριθμος χρησιμοποιεί $\Theta(n^2)$ θέσεις μνήμης για την αποθήκευση των στοιχείων $m[i,j]$, $1 \leq i < j \leq n$.

Άσκηση 3.1. Εξηγείστε πως λειτουργεί η MatrixChainMultiplication με είσοδο το διάνυσμα $d = [30, 35, 15, 5, 10, 20, 25]$ (έχουμε $n = 6$ πίνακες).

Λύση. Υπολογίστε τα $m[i,j]$ για κάθε $1 \leq i < j \leq n$. Είναι $m[1,n] = 15125$ και βέλτιστη σειρά υπολογισμού $((A_1(A_2A_3))((A_4A_5)A_6)$. Η λύση δίνεται στο [11], Σχήμα 15.3. □

Άσκηση 3.2. Διατυπώστε έναν αναδρομικό (top-down) αλγόριθμο που υπολογίζει το $m[1,n]$ με βάση την αναδρομική σχέση (3.2). Ποιος είναι ο χρόνος εκτέλεσης του αλγορίθμου; Παρουσιάστε αναλυτικά τη δομή των αναδρομικών κλήσεων του αλγόριθμου για 4 πίνακες. Με βάση αυτό το παράδειγμα εξηγείστε γιατί ο αναδρομικός αλγόριθμος έχει μεγαλύτερο χρόνο εκτέλεσης από τον αλγόριθμο δυναμικού προγραμματισμού.

Λύση. Ο αναδρομικός αλγόριθμος υλοποιείται από τον παρακάτω ψευδοκώδικα:

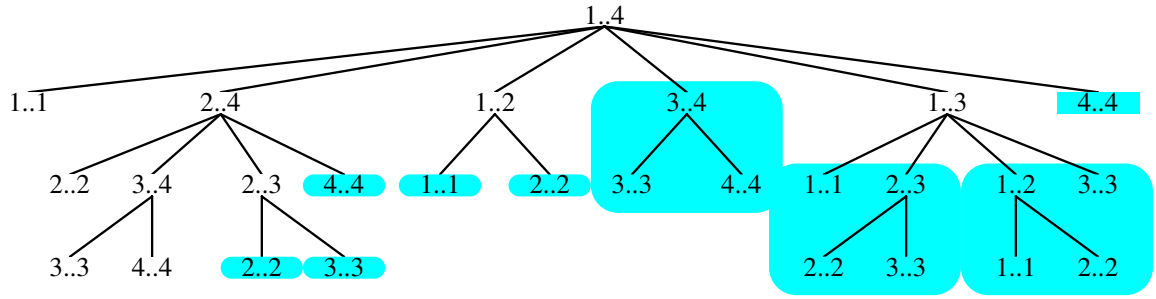
```
RecMatrixChain(d[i − 1, ..., j])
    if i = j then return(0);
    m ← ∞;    /* Το m θα πάρει την τιμή m[i, j] */
    for k ← i to j − 1 do
        q ←RecMatrixChain(d[i − 1, ..., k]) +
            RecMatrixChain(d[k, ..., j]) + d[i − 1]d[k]d[j];
        if q < m then m ← q;
    return(m);
```

Η αρχική κλήση πρέπει να είναι RecMatrixChain($d[0, \ldots, n]$). Παρατηρούμε ότι για τον υπολογισμό του $m[1,n]$ γίνονται δύο αναδρομικές κλήσεις σε κάθε εκτέλεση του for-loop (δηλαδή δύο κλήσεις για κάθε $k = 1, \ldots, n-1$ και ένα πολλαπλασιασμός. Η πρώτη κλήση αφορά ένα στιγμιότυπο με k πίνακες και υπολογίζει το $m[1,k]$ και η δεύτερη κλήση αφορά ένα στιγμιότυπο με $n-k$ πίνακες και υπολογίζει το $m[k+1,n]$. Συνεπώς, ένα κάτω φράγμα στο χρόνο εκτέλεσης $T(n)$ του αναδρομικού αλγόριθμου για n πίνακες προκύπτει από την αναδρομική σχέση:

$$T(n) \geq 1 + \sum_{k=1}^{n-1}[T(k) + T(n-k) + 1] = n + 2\sum_{k=1}^{n-1} T(k)$$

με αρχική συνθήκη $T(1) = \Omega(1)$. Με μαθηματική επαγωγή, μπορούμε να αποδείξουμε ότι $T(n) = \Omega(2^n)$, το οποίο είναι δραματικά μεγαλύτερο από $\Theta(n^3)$ που είναι ο χρόνος εκτέλεσης του αλγόριθμου δυναμικού προγραμματισμού.

Ο λόγος είναι ότι το Πρόβλημα του Πολλαπλασιασμού Ακολουθίας Πινάκων έχει την ιδιότητα των επικαλυπτόμενων υπο-στιγμιοτύπων. Έτσι ο αναδρομικός αλγόριθμος λύνει πολλές φορές το ίδιο υπο-στιγμιότυπο σε αντίθεση με τον αλγόριθμο δυναμικού προγραμματισμού κάθε υπο-στιγμιότυπο ακριβώς μία φορά (βλ. Σχήμα 3.1). □



Σχήμα 3.1: Το δέντρο των αναδρομικών κλήσεων του αλγόριθμου RecMatrixChain για 4 πίνακες. Τα υπο-στιγμιότυπα με γρίζο χρώμα στα δεξιά έχουν ήδη λυθεί από προηγούμενες κλήσεις. Σε αυτές τις περιπτώσεις ο αναδρομικός αλγόριθμος ξαναλύνει το υπο-στιγμιότυπο, ενώ ο αλγόριθμος δυναμικού προγραμματισμού απλώς ανακαλεί τη βέλτιστη αντικειμενική τιμή. Το παράδειγμα είναι από το [11], Σχήμα 15.5.

Άσκηση 3.3. Η MatrixChainMultiplication επιστρέφει μόνο τη βέλτιστη αντικειμενική τιμή $m[1, n]$. Να τροποποιήσετε τον αλγόριθμο ώστε υπολογίζει μια βέλτιστη λύση. Να διατυπώσετε σε ψευδοκώδικα έναν αλγόριθμο που θα υπολογίζει το γινόμενο $A_{1..n}$ χρησιμοποιώντας ακριβώς $m[1, n]$ πολλαπλασιασμούς μεταξύ στοιχείων των πινάκων.

Λύση. Για να είναι δυνατή η ανακατασκευή της βέλτιστης λύσης πρέπει να αποθηκεύουμε την τιμή του k (σημείο διαίρεσης της υπακολουθίας $A_i \dots A_j$) για την οποία προκύπτει η βέλτιστη τιμή $m[i, j]$, για κάθε $1 \leq i < j \leq n$. Αυτό μπορεί να γίνει χρησιμοποιώντας έναν πίνακα $s[i, j]$. Με αυτό τον τρόπο, αφού υπολογίσουμε τη βέλτιστη τιμή $m[1, n]$, μπορούμε να ανακαλέσουμε τις επιμέρους βέλτιστες λύσεις που συνδυάστηκαν για να σχηματιστεί η βέλτιστη λύση για το αρχικό στιγμιότυπο. Ο υπολογισμός του γινομένου μπορεί να γίνει αναδρομικά διαιρώντας την υπακολουθία $A_i \dots A_j$ στο σημείο $s[i, j]$ και υπολογίζοντας το επιμέρους γινόμενο $A_{i..j} = A_{i..s[i,j]} A_{s[i,j]+1..j}$. □

3.1.1 Αναδρομή με Μνήμη

Στην Άσκηση 3.2 διατυπώσαμε έναν top-down αλγόριθμο για το Πρόβλημα του Πολλαπλασιασμού Ακολουθίας Πινάκων με χρόνο εκθετικό εκτέλεσης, τη στιγμή που ο αντίστοιχος αλγόριθμος δυναμικού προγραμματισμού είχε μόλις κυβικό χρόνο εκτέλεσης. Μπορούμε όμως να συνδυάσουμε την αποδοτικότητα του δυναμικού προγραμματισμού με την απλότητα στο σχεδιασμό και την υλοποίηση που προσφέρει η αναδρομική (top-down) προσέγγιση χρησιμοποιώντας την τεχνική της *αναδρομής με μνήμη*.

Όπως στο δυναμικό προγραμματισμό, αποθηκεύουμε τις βέλτιστες λύσεις των υποστιγμιοτύπων σε έναν πίνακα. Όμως η συμπλήρωση αυτού του πίνακα μιμείται την πορεία του αναδρομικού αλγόριθμου. Αρχικά, τα στοιχεία του πίνακα περιέχουν μία ειδική τιμή που δηλώνει ότι τα αντίστοιχα υπο-στιγμιότυπα δεν έχουν επιλυθεί. Κάθε αναδρομική κλήση αρχικά εξετάζει την αντίστοιχη θέση του πίνακα. Αν το υπο-στιγμιότυπο έχει ήδη επιλυθεί, η βέλτιστη αντικειμενική τιμή ανακαλείται από τον πίνακα και η κλήση ολοκληρώνεται. Διαφορετικά, το

υπο-στιγμιότυπο επιλύεται και η βέλτιστη αντικειμενική τιμή αποθηκεύεται στον πίνακα για μελλοντική χρήση. Με απλά λόγια, κάθε φορά που εμφανίζεται το ίδιο υπο-στιγμιότυπο, αντί να το επιλύσουμε από την αρχή όπως θα έκανε ο αναδρομικός αλγόριθμος, ανατρέχουμε στον πίνακα των επιμέρους βέλτιστων λύσεων, σύμφωνα με τις επιταγές του δυναμικού προγραμματισμού.

Οι παρακάτω συναρτήσεις εφαρμόζουν την τεχνική της αναδρομής με μνήμη στο Πρόβλημα του Πολλαπλασιασμού Ακολουθίας Πινάκων.

```
RecMemMatrixChain(d[0, ..., n])
    for i ← 1 to n do
        for j ← 1 to n do
            m[i, j] ← ∞;      /* Αρχικοποίηση πίνακα */
    return(RecMemChain(d[0, ..., n]));

RecMemChain(d[i − 1, ..., j]);
    if m[i, j] < ∞ then return(m[i, j]);
    if i = j then m[i, j] = 0;
    else
        for k ← i to j − 1 do
            q ←RecMemChain(d[i − 1, ..., k]) +
                RecMemChain(d[k, ..., j]) + d[i − 1]d[k]d[j];
            if q < m[i, j] then m[i, j] ← q;
    return(m[i, j]);
```

Ο χρόνος εκτέλεσης της RecMemMatrixChain είναι $\Theta(n^3)$ επειδή κάθε θέση του πίνακα $m[i, j]$, $1 \leq i < j \leq n$, αρχικοποιείται μόνο μία φορά. Υποθέτωντας επαγωγικά ότι οι θέσεις $m[i, k]$ και $m[k + 1, j]$ έχουν ήδη αρχικοποηθεί, το περιεχόμενο της θέσης $m[i, j]$ υπολογίζεται σε γραμμικό χρόνο.

3.2 Το Πρόβλημα του Περιοδεύοντος Πωλητή

Ένα τυπικό παράδειγμα προβλήματος συνδυαστικής βελτιστοποίησης στο οποίο εφαρμόζεται η μέθοδος του δυναμικού προγραμματισμού είναι το Πρόβλημα του Περιοδεύοντος Πωλητή (Travelling Salesman Problem, TSP). Το Πρόβλημα του Περιοδεύοντος Πωλητή είναι από τα σημαντικότερα και ευρύτατα μελετημένα προβλήματα συνδυαστικής βελτιστοποίησης με πολλές παραλλαγές και πληθώρα εξαιρετικά σημαντικών πρακτικών εφαρμογών.

Ας ανακαλέσουμε πρώτα τον ορισμό του προβλήματος που παρουσιάστηκε στην Ενότητα 1.2. Ένα στιγμιότυπο για το Πρόβλημα του Περιοδεύοντος Πωλητή αποτελείται από ένα σύνολο $N = \{1, 2, \ldots, n\}$ σημείων και μια συνάρτηση απόστασης $d : N \times N \mapsto \mathbb{R}_+$ που ορίζει τις αποστάσεις d_{ij} και d_{ji} για τη μετάβαση από το σημείο i στο σημείο j και από το j στο i αντίστοιχα. Το ζητούμενο είναι μια *περιοδεία* (δηλ. ένας κύκλος που διέρχεται από κάθε σημείο ακριβώς μία φορά) ελαχίστου μήκους.

Χωρίς βλάβη της γενικότητας, θεωρούμε ότι οι αποστάσεις είναι θετικές και ότι η απόσταση κάθε σημείου από τον εαυτό του είναι μηδενική, δηλ. $d_{ii} = 0$, $\forall i \in N$. Σε αυτή την ενότητα θα θεωρήσουμε τη γενική μορφή του προβλήματος όπου οι αποστάσεις δεν είναι κατ' ανάγκη

συμμετρικές (δηλ. μπορεί να είναι $d_{ij} \neq d_{ji}$) και δεν επαληθεύουν κατ' ανάγκη την τριγωνική ανισότητα. Κάθε περιοδεία αποτελεί μια αποδεκτή λύση για το Πρόβλημα του Περιοδεύοντος Πωλητή με αντικειμενική τιμή το συνολικό μήκος της. Η βέλτιση λύση, που είναι και το ζητούμενο, είναι η περιοδεία με το μικρότερο μήκος (ή απλούστερα η *συντομότερη περιοδεία*).

Αφού μια περιοδεία διέρχεται από κάθε σημείο ακριβώς μία φορά, μπορούμε να θεωρήσουμε ότι κάθε περιοδεία ξεκινά από το σημείο 1. Επειδή δεν υπάρχουν άλλοι περιορισμοί, κάθε μετάθεση (permutation) των σημείων που διατηρεί το σημείο 1 στην πρώτη θέση συνιστά μια περιοδεία. Με άλλα λόγια, κάθε 1-1 και επί συνάρτηση (αντιστοιχία) $\pi : N \mapsto N$ με $\pi(1) = 1$ αναπαριστά μια περιοδεία η οποία επισκέπτεται το σημείο $\pi(i)$ στη σειρά i. Το συνολικό μήκος της περιοδείας π είναι:

$$L(\pi) = d_{\pi(n)1} + \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)}$$

Υπάρχουν λοιπόν $(n-1)!$ διαφορετικές περιοδείες, όσες και οι διαφορετικές μεταθέσεις των σημείων $N \setminus \{1\}$. Το γεγονός αυτό καθιστά απαγορευτικό τον υπολογισμό της συντομότερης περιοδείας με εξαντλητική αναζήτηση.

Το Πρόβλημα του Περιοδεύοντος Πωλητή είναι δύσκολο για την κλάση υπολογιστικής πολυπλοκότητας **NP** (**NP**-hard, βλέπε αντίστοιχο κεφάλαιο). Αποτελεί κοινή επιστημονική πεποίθηση ότι για κανένα πρόβλημα σε αυτή την κατηγορία δεν υπάρχει αλγόριθμος που να υπολογίζει τη βέλτιστη λύση και να έχει χρόνο εκτέλεσης χειρότερης μικρότερο από εκθετικό (sub-exponential). Σε αυτή την ενότητα, θα διατυπώσουμε έναν αλγόριθμο δυναμικού προγραμματισμού για το Πρόβλημα του Περιοδεύοντος Πωλητή. Ο αλγόριθμος, αν και έχει εκθετικό χρόνο εκτέλεσης, είναι σημαντικά ταχύτερος από την εξαντλητική αναζήτηση.

Αρχικά εξετάζουμε αν το πρόβλημα έχει την ιδιότητα των βέλτιστων επιμέρους λύσεων. Έστω μια βέλτιστη περιοδεία που ξεκινά από το σημείο 1 και συνεχίζει στο σημείο j. Το μήκος της περιοδείας είναι d_{1j} συν το μήκος του τμήματος / μονοπατιού της από το σημείο j στο 1. Επειδή πρόκειται για περιοδεία, το τμήμα της από το j στο 1 διέρχεται από όλα τα υπόλοιπα σημεία (δηλ. όλα τα σημεία του συνόλου $N \setminus \{1, j\}$). Επειδή η περιοδεία είναι βέλτιστη, το τμήμα της από το j στο 1 είναι το συντομότερο μονοπάτι που συνδέει το j με το 1 διερχόμενο από όλα τα σημεία του $\setminus \{1, j\}$. Με απλά λόγια, κάθε βέλτιστη περιοδεία ακολουθεί μόνο συντομότερα / βέλτιστα μονοπάτια (που διέρχονται από όλα τα υπόλοιπα σημεία) για να μεταβεί από τα ενδιάμεσα σημεία στο αρχικό. Παρατηρούμε λοιπόν ότι το Πρόβλημα του Περιοδεύοντος Πωλητή έχει την ιδιότητα των βέλτιστων επιμέρους λύσεων.

Με βάση αυτή την παρατήρηση, προσπαθούμε να διατυπώσουμε μια αναδρομική σχέση που θα περιγράφει τη βέλτιστη λύση. Έστω $S \subseteq N \setminus \{1\}$ ένα υποσύνολο σημείων, και έστω $i \in N \setminus S$ ένα σημείο διαφορετικό από το αρχικό σημείο 1 εφόσον $N \setminus S \neq \{1\}$. Συμβολίζουμε με $L(i, S)$ το μήκος του συντομότερου μονοπατιού που συνδέει το σημείο i με το σημείο 1 διερχόμενο από όλα τα σημεία του S. Προφανώς, αν $S = \emptyset$ (δεν έχει μείνει κανένα σημείο ακάλυπτο), είναι $L(i, \emptyset) = d_{i1}$ για κάθε $i \in N \setminus \{1\}$. Αν $S \neq \emptyset$, τότε το μήκος του συντομότερου μονοπατιού που ξεκινάει από το i, καταλήγει στο 1, και διέρχεται από όλα τα σημεία του S δίνεται από την αναδρομική σχέση:

$$L(i, S) = \min_{j \in S} \{d_{ij} + L(j, S \setminus \{j\})\} \tag{3.3}$$

Όταν $S = N \setminus \{1\}$, θεωρούμε σαν σημείο εκκίνησης το σημείο 1 επειδή είναι το μοναδικό σημείο στο σύνολο $N \setminus S$. Επομένως, το μονοπάτι ξεκινάει και καταλήγει στο σημείο 1 διερχόμενο από όλα τα υπόλοιπα σημεία και το $L(1, N \setminus \{1\})$ είναι το μήκος της συντομότερης περιοδείας.

Έχοντας διατυπώσει την αναδρομική σχέση 3.3 για το μήκος της συντομότερης περιοδείας, θα προχωρήσουμε στον υπολογισμό του με τη μέθοδο του δυναμικού προγραμματισμού. Ο υπολογισμός των ενδιάμεσων μηκών γίνεται από τα μικρότερα στα μεγαλύτερά σύνολα. Συγκεκριμένα, για να υπολογίσουμε το $L(i, S)$ για κάποιο σύνολο S με $k + 1$ σημεία, πρέπει να έχουμε ήδη υπολογίσει όλες τις τιμές $L(j, S \setminus \{j\})$. Υπολογίζουμε λοιπόν τις τιμές $L(i, S)$ ξεκινώντας από $S = \emptyset$, και συνεχίζοντας με σύνολα μεγέθους $1, 2, \ldots, n-1$.

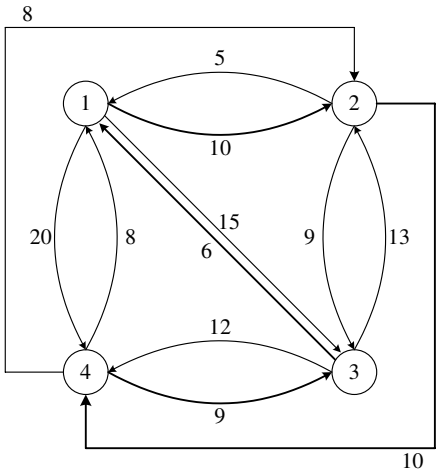
Μια υλοποίηση αυτής της στρατηγικής δίνεται από τον παρακάτω ψευδοκώδικα. Χρησιμοποιούμε τον βοηθητικό πίνακα J για την αποθήκευση της βέλτιστης περιοδείας. Το $J(i, S)$ δίνει το επόμενο σημείο j του μονοπατιού από το i στο 1 που ακολουθεί η βέλτιστη περιοδεία.

```
TSP(d[1...n][1...n])
    for i ← 2 to n do
        L(i, ∅) ← d[i, 1];
    for k ← 1 to n − 2 do
        for all S ⊂ N \ {1}, |S| = k do
            for all i ∈ (N \ {1}) \ S do
                q ← ∞;
                for all j ∈ S do
                    if d[i, j] + L(j, S \ {j}) < q then
                        q ← d[i, j] + L(j, S \ {j});   t ← j;
                    L(i, S) ← q;   J(i, S) ← t;
    q ← ∞;
    for j ← 2 to n do
        if d[1, j] + L(j, N \ {1, j}) < q then
            q ← d[1, j] + L(j, N \ {1, j});    t ← j;
    L(1, N \ {1}) ← q;   J(1, N \ {1}) ← t;
    return(L(1, N \ {1}), J);
```

Η αρχικοποίηση των στοιχείων $L(i, \emptyset)$ και ο υπολογισμός του στοιχείου $L(1, N \setminus \{1\})$ γίνονται σε χρόνο $\Theta(n-1)$. Για $k = 1, \ldots, n-2$, ο αριθμός των διαφορετικών γνήσιων υποσυνόλων $S \subset N \setminus \{1\}$ με μέγεθος k είναι $\binom{n-2}{k}$. Για κάθε τέτοιο σύνολο θεωρούμε $n-k-1$ αρχικά σημεία $i \in (N \setminus \{1\}) \setminus S$. Για κάθε τέτοιο συνδυασμό (i, S), ο υπολογισμός της τιμής $L(i, S)$ γίνεται σε χρόνο $\Theta(k)$ από την αναδρομική σχέση 3.3. Συνεπώς, ο χρόνος εκτέλεσης του αλγόριθμου δυναμικού προγραμματισμού για το Πρόβλημα του Περιοδεύοντος Πωλητή είναι:

$$\Theta\left(\sum_{k=1}^{n-2} \binom{n-2}{k} (n-k-1)k\right) = \Theta(n^2 2^n)$$

Όσον αφορά στις απαιτήσεις σε μνήμη, ο πίνακας L έχει $\Theta(n2^n)$ στοιχεία. Επομένως, ο αλγόριθμος δυναμικού προγραμματισμού χρειάζεται $\Theta(n2^n)$ θέσεις μνήμης για να υπολογίσει τη βέλτιστη περιοδεία.



Σχήμα 3.2: Στιγμιότυπο του Προβλήματος του Περιοδεύοντος Πωλητή με 4 σημεία. Το παράδειγμα είναι από το [7], Παράδειγμα 5.6.1.

Ο χρόνος εκτέλεσης και ο χώρος αποθήκευσης αυξάνονται λοιπόν εκθετικά με τον αριθμό των σημείων. Επομένως, αυτός ο αλγόριθμος μπορεί να χρησιμοποιηθεί στην πράξη μόνο για στιγμιότυπα με μικρό αριθμό σημείων (π.χ. $n \leq 20$). Από την άλλη πλευρά, ο αλγόριθμος δυναμικού προγραμματισμού είναι δραματικά ταχύτερος από την εξαντλητική αναζήτηση. Ενδεικτικά αναφέρουμε ότι για 20 σημεία, η ποσότητα $n^2 2^n$ είναι περίπου ίση με 4.2×10^8, ενώ ο αριθμός των διαφορετικών περιοδειών που πρέπει να ελεγχθούν από τον αλγόριθμο εξαντλητικής αναζήτησης είναι 1.2×10^{17}.

Άσκηση 3.4. Έστω ένα σύνολο τεσσάρων σημείων $N = \{1, 2, 3, 4\}$ των οποίων οι αποστάσεις απεικονίζονται στο Σχήμα 3.2. Περιγράψτε τη λειτουργία του αλγόριθμου δυναμικού προγραμματισμού TSP για αυτό το στιγμιότυπο εισόδου.

Λύση. Οι τιμές των πινάκων L και J φαίνονται στον παρακάτω πίνακα. Τα στοιχεία του πίνακα συμπληρώνονται από αριστερά προς τα δεξιά και από πάνω προς τα κάτω. Έχουν μείνει κενές οι θέσεις του πίνακα που δεν αντιστοιχούν σε έγκυρους συνδυασμούς παραμέτρων.

$i \vert S$	$\emptyset$	$\{2\}$	$\{3\}$	$\{4\}$	$\{3,4\}$	$\{2,4\}$	$\{2,3\}$	$\{2,3,4\}$
1								**35 (2)**
2	5		15 (3)	18 (4)	**25 (4)**			
3	**6**	18 (2)		20 (4)		25 (4)		
4	8	13 (2)	**15 (3)**				23 (2)	

Η βέλτιστη περιοδεία έχει μήκος 35. Ακολουθώντας τα περιεχόμενα που πίνακα J, διαπιστώνουμε ότι η βέλτιστη περιοδεία είναι $1, 2, 4, 3$. □

3.3 Το Πρόβλημα του Σακιδίου

Ένα άλλο πρόβλημα συνδυαστικής βελτιστοποίησης στο οποίο εφαρμόζεται με επιτυχία η μέθοδος του δυναμικού προγραμματισμού είναι το Ακέραιο Πρόβλημα του Σακιδίου (ή απλά Πρόβλημα

του Σακιδίου − Knapsack Problem). Ας ανακαλέσουμε τον ορισμό του προβλήματος που παρουσιάστηκε στην Ενότητα 1.2. Δίνονται ένα σακίδιο μεγέθους $B > 0$ και n αντικείμενα μεγέθους $s_i > 0$ και αξίας $p_i > 0$, $i = 1, \ldots, n$. Το ζητούμενο είναι η συλλογή αντικειμένων με τη μεγαλύτερη αξία που χωράει στο σακίδιο. Τυπικά, ζητείται η μεγιστοποίηση του $\sum_{i=1}^{n} f_i\, p_i$ υπό τους περιορισμούς $\sum_{i=1}^{n} f_i\, s_i \leq B$ και $f_i \in \{0, 1\}$ για κάθε $i = 1, \ldots, n$.

Όπως και το Πρόβλημα του Περιοδεύοντος Πωλητή, το Πρόβλημα του Σακιδίου είναι δύσκολο για την κλάση **NP**. Είναι λοιπόν κοινή επιστημονική πεποίθηση ότι κάθε αλγόριθμος για το Πρόβλημα του Σακιδίου έχει χρόνο εκτέλεσης χειρότερης περίπτωσης εκθετικό στο μέγεθος του στιγμιότυπου εισόδου. Ο αλγόριθμος εξαντλητικής αναζήτησης δοκιμάζει όλα τα δυνατά υποσύνολα αντικειμένων και έχει χρόνο εκτέλεσης $\Theta(n2^n)$. Σε αυτή την ενότητα, θα διατυπώσουμε έναν πολύ αποδοτικότερο αλγόριθμο δυναμικού προγραμματισμού.

Αρχικά εξετάζουμε αν το Πρόβλημα του Σακιδίου έχει την ιδιότητα των βέλτιστων επιμέρους λύσεων. Έστω $A^* \subseteq \{1, \ldots, n\}$ η βέλτιστη συλλογή αντικειμένων. Παρατηρούμε ότι αν απομακρύνουμε οποιοδήποτε αντικείμενο $j \in A^*$ από τη βέλτιστη συλλογή, η συλλογή $A^* \setminus \{j\}$ αποτελεί μια βέλτιστη λύση για σακίδιο μεγέθους $B - s_j$ και αντικείμενα $\{1, \ldots, n\} \setminus \{j\}$. Αν μάλιστα j είναι το αντικείμενο της βέλτιστης συλλογής με το μεγαλύτερο δείκτη, το σύνολο των αντικειμένων μπορεί να περιοριστεί σε $\{1, \ldots, j-1\}$.

Με βάση αυτή την παρατήρηση, θα διατυπώσουμε τη μαθηματική σχέση που περιγράφει τη βέλτιστη λύση. Για κάθε $i = 0, 1, \ldots, n$ και $b = 0, \ldots, B$, έστω $P(i, b)$ η μέγιστη αξία αντικειμένων από το σύνολο $\{1, \ldots, i\}$ που χωρούν σε σακίδιο μεγέθους b. Η αξία της βέλτιστης συλλογής είναι $P(n, B)$ αφού επιλέγουμε αντικείμενα από το σύνολο $\{1, \ldots, n\}$ που πρέπει να χωρούν σε σακίδιο μεγέθους B.

Είναι $P(0, b) = 0$ για κάθε $b = 0, \ldots, B$ (το κενό σύνολο έχει μηδενική αξία). Επίσης όταν $b \leq 0$, έχουμε $P(i, b) = 0$ για κάθε $i = 1, \ldots, n$ (κανένα αντικείμενο δεν μπορεί να χωρέσει σε σακίδιο με μηδενικό ή αρνητικό μέγεθος).

Έχοντας υπολογίσει το $P(i-1, b)$, εξετάζουμε τις δυνατότητες για το αντικείμενο i για κάθε διαφορετική τιμή του b, $b = 0, \ldots, B$. Η πρώτη δυνατότητα είναι το i να μείνει εκτός σακιδίου οπότε η αξία των αντικειμένων στο σακίδιο παραμένει $P(i-1, b)$. Η δεύτερη δυνατότητα είναι το i να συμπεριληφθεί στο σακίδιο, οπότε ο χώρος που διατίθεται για το σύνολο $\{1, \ldots, i-1\}$ γίνεται $b - s_i$. Η συνολική αξία των αντικειμένων γίνεται $P(i-1, b-s_i) + p_i$. Επιλέγουμε τη δυνατότητα που μεγιστοποιεί την τιμή του $P(i, b)$. Επομένως, οι τιμές του $P(i, b)$ δίνονται από την αναδρομική εξίσωση:

$$P(i,b) = \begin{cases} \max\{P(i-1,b), P(i-1,b-s_i)+p_i\} & \text{για κάθε } i = 1,\ldots,n \text{ και } b = 1,\ldots,B \\ 0 & \text{αν } b \leq 0 \\ 0 & \text{αν } i = 0 \end{cases} \tag{3.4}$$

Ο υπολογισμός της βέλτιστης αντικειμενικής τιμής μπορεί να γίνει με δυναμικό προγραμματισμό. Υπολογίζονται πρώτα οι τιμές $P(i, b)$ που αντιστοιχούν σε μικρότερες τιμές του i. Για κάθε συγκεκριμένη τιμή του i, υπολογίζονται πρώτα τα στοιχεία που αντιστοιχούν σε μικρότερες τιμές του b. Κάθε στοιχείο της γραμμής i μπορεί να υπολογιστεί σε σταθερό χρόνο αν είναι διαθέσιμα όλα τα στοιχεία της γραμμής $i-1$. Συνολικά υπολογίζονται $(n+1)(B+1)$ τιμές. Ο χρόνος εκτέλεσης του αλγόριθμου δυναμικού προγραμματισμού είναι $\Theta(nB)$. Οι θέσεις μνήμης που

απαιτούνται για τον υπολογισμό της βέλτιστης αντικειμενικής τιμής είναι $\Theta(B)$ επειδή ο υπολογισμός των στοιχείων της γραμμής i απαιτεί μόνο τα στοιχεία της γραμμής $i-1$. Έτσι ο αλγόριθμος χρειάζεται να αποθηκεύει μόνο τις τιμές της τρέχουσας γραμμής i και της προηγούμενης γραμμής $i-1$.

Για λόγους απλότητας, παραθέτουμε μια υλοποίηση του αλγόριθμου που χρησιμοποιεί $\Theta(nB)$ θέσεις μνήμης. Αφήνεται σαν άσκηση να τροποποιηθεί ο παρακάτω ψευδοκώδικας ώστε να χρησιμοποιεί $\Theta(B)$ θέσεις μνήμης.

```
Knapsack(B, (s_1, p_1), ..., (s_n, p_n))
    for i ← 0 to n do P[i, 0] ← 0;
    for b ← 1 to B do P[0, b] ← 0;
    for i ← 1 to n do
        for b ← 1 to B do
            if b − s_i ≥ 0 then
                t ← P[i − 1, b − s_i] + p_i;
            else t ← 0;
            if P[i − 1, b] ≥ t then
                P[i, b] ← P[i − 1, b];
            else P[i, b] ← t;
    return(P[n, B]);
```

Άσκηση 3.5. Να τροποιήσετε τον παραπάνω ψευδοκώδικα ώστε να εκτυπώνει τα στοιχεία της βέλτιστης συλλογής. Ποιές είναι οι απαιτήσεις σε μνήμη του τροποιημένου αλγόριθμου;

Λύση. Για τον υπολογισμό της βέλτιστης λύσης, πρέπει να ανατρέξουμε στον τρόπο υπολογισμού του $P(n, B)$ και να δούμε ποια αντικείμενα πρέπει να συμπεριληφθούν στη βέλτιστη συλλογή.

Διατρέχουμε τον πίνακα από τις γραμμές με μεγαλύτερο δείκτη προς τις γραμμές με μικρότερο δείκτη. Ψάχνουμε τον μεγαλύτερο δείκτη j για τον οποίο $P(j, B) > P(j-1, B)$. Το j είναι το τελευταίο στοιχείο που έχει συμπεριληφθεί στη βέλτιστη συλλογή. Μειώνουμε το μέγεθος του σακιδίου σε $B - s_j$ (δεσμεύοντας χώρο για το στοιχείο j) και συνεχίζουμε ψάχνοντας τον μεγαλύτερο δείκτη j' για τον οποίο $P[j', B - s_j] > P[j'-1, B - s_j]$, κοκ. Ο παραπάνω ψευδοκώδικας πρέπει να συμπληρωθεί ακριβώς πριν την εντολή return ως εξής:

```
b ← B;
for i ← n down to 1 do
    if P[i, b] > P[i − 1, b] then
        print(i);   b ← b − s_i;
```

Βλέπουμε ότι για να υπολογίσουμε τη βέλτιστη συλλογή, ο αλγόριθμος δυναμικού προγραμματισμού χρειάζεται να αποθηκεύσει ολόκληρο τον πίνακα $P[0 \ldots n, 0 \ldots B]$. Συνεπώς, ενώ για τον υπολογισμό της βέλτιστης αντικειμενικής τιμές χρειάζονται $\Theta(B)$ θέσεις μνήμης, για τον υπολογισμό της βέλτιστης λύσης χρειάζονται $\Theta(nB)$ θέσεις μνήμης. □

Άσκηση 3.6. Εξηγείστε τη λειτουργία του αλγόριθμου δυναμικού προγραμματισμού για το στιγμιότυπο $(10, ((3,5),(2,7),(4,4),(6,8),(5,4)))$ του Προβλήματος του Σακιδίου.

Λύση. Ακολουθούν οι τιμές της συνάρτησης P για το συγκεκριμένο στιγμιότυπο.

$i \vert b$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	5	**5**	5	5	5	5	5	5
2	0	0	7	7	7	12	**12**	12	12	12	12
3	0	0	7	7	7	12	12	12	12	16	**16**
4	0	0	7	7	7	12	12	12	15	16	16
5	0	0	7	7	7	12	12	12	15	16	16

Η αξία της βέλτιστης συλλογής είναι $P(4,10) = 16$. Το συγκεκριμένο στιγμιότυπο έχε δύο βέλτιστες λύσεις, τις $\{1,2,3\}$ και $\{1,2,5\}$. Η τροποποίηση της Άσκησης 3.5, θα υπολογίσει την πρώτη. □

Άσκηση 3.7. Στο *Πρόβλημα της Διαμέρισης* (Partition Problem) δίνεται ένα σύνολο φυσικών αριθμών $X = \{x_1, x_2, \ldots, x_n\}$ και πρέπει να αποφασίσουμε αν μπορεί να χωριστεί σε δύο υποσύνολα X_1 και X_2 με το ίδιο άθροισμα στοιχείων. Πρέπει δηλαδή να αποφασίσουμε αν η παρακάτω λογική πρόταση αληθεύει για το σύνολο X:

$$\exists X_1, X_2 \subseteq X : (X_1 \cap X_2 = \emptyset) \wedge (X_1 \cup X_2 = X) \wedge \left(\sum_{i \in X_1} x_i = \sum_{i \in X_2} x_i \right) \tag{3.5}$$

Να διατυπώσετε και να αναλύσετε έναν αλγόριθμο δυναμικού προγραμματισμού για το Πρόβλημα της Διαμέρισης. Ποιός είναι ο χρόνος για να λυθεί το πρόβλημα εξετάζοντας όλα τα πιθανά υποσύνολα του X;

Λύση. Χωρίς βλάβη της γενικότητας, θεωρούμε ότι το άθροισμα των στοιχείων του X είναι ζυγός αριθμός. Με άλλα λόγια, υπάρχει φυσικός αριθμός S τέτοιος ώστε $2\,S = \sum_{i \in X} x_i$. Ισοδύναμα, πρέπει να αποφασίσουμε αν υπάρχει $X' \subseteq X$ με άθροισμα στοιχείων ακριβώς S. Παρατηρούμε επίσης ότι το Πρόβλημα της Διαμέρισης δεν είναι πρόβλημα βελτιστοποίησης (βλ. επίσης Άσκηση 3.8). Μολαταύτα, το Πρόβλημα της Διαμέρισης έχει μια ιδιότητα ταυτόσημε με αυτή των βέλτιστων επιμέρους λύσεων και μπορεί να λυθεί με δυναμικό προγραμματισμό.

Έστω X' ένα υποσύνολο του X με άθροισμα στοιχείων S (θεωρούμε το X' σαν τη "λύση" του προβλήματος). Έστω $j \in X'$ το στοιχείο του X' με το μεγαλύτερο δείκτη. Τότε το σύνολο $X' \setminus \{j\}$ αποτελεί μια "λύση" στο ερώτημα αν "υπάρχει ένα υποσύνολο του $\{x_1, \ldots, x_{j-1}\}$ με άθροισμα στοιχείων ακριβώς $S - x_j$;". Αυτή η ιδιότητα του Προβλήματος της Διαμέρισης είναι ταυτόσημη με την ιδιότητα των βέλτιστων επιμέρους λύσεων.

Ορίζουμε τη λογική συνάρτηση $p : [n] \times [S] \mapsto \{0,1\}$ ως εξής[3]: $p(i,j) = 1$ αν υπάρχει υποσύνολο του $X_i = \{x_1, \ldots, x_i\}$ με άθροισμα στοιχείων j, και $p[i,j] = 0$ αν δεν υπάρχει τέτοιο υποσύνολο, $0 \leq i \leq n$, $0 \leq j \leq S$.

[3] Έστω k φυσικός αριθμός. Συμβολίζουμε με $[k]$ το σύνολο των φυσικών αριθμών που δεν ξεπερνούν το k, δηλαδή το σύνολο $\{0, \ldots, k\}$.

Για κάθε $j > 0$, είναι $p[0, j] = 0$ γιατί το άθροισμα των στοιχείων του κενού συνόλου είναι 0. Για κάθε $i \in [n]$, είναι $p[i, 0] = 1$ για τον ίδιο λόγο, και $p[i, j] = 0$ αν $j < 0$, επειδή δεν υπάρχει υποσύνολο με αρνητικό άθροισμα στοιχείων.

Για κάθε $i = 1, \ldots, n$ και $j = 1, \ldots, S$, είναι $p[i, j] = 1$ όταν είτε $p[i-1, j] = 1$ (δηλ. υπάρχει ήδη υποσύνολο του $\{x_1, \ldots, x_{i-1}\}$ με άθροισμα στοιχείων j) είτε $p[i-1, j-x_i] = 1$ (δηλ. υπάρχει υποσύνολο του $\{x_1, \ldots, x_{i-1}\}$ με άθροισμα στοιχείων $j - x_i$ που γίνεται j με την προσθήκη του x_i). Με άλλα λόγια, $p[i, j] = p[i-1, j] \vee p[i-1, j-x_i]$. Καταλήγουμε λοιπόν στην παρακάτω αναδρομική σχέση που περιγράφει τη λύση στο Πρόβλημα της Διαμέρισης:

$$p(i,j) = \begin{cases} p(i-1,j) \vee p(i-1,j-x_i) & \text{για κάθε } i = 1, \ldots, n \text{ και } j = 1, \ldots, S \\ 1 & \text{αν } j = 0 \\ 0 & \text{αν } j < 0 \\ 0 & \text{αν } i = 0 \text{ και } j > 0 \end{cases} \tag{3.6}$$

Η απάντηση στο Πρόβλημα της Διαμέρισης δίνεται από την τιμή $p(n, S)$ και μπορεί να υπολογισθεί με τη μέθοδο του δυναμικού προγραμματισμού. Ο ψευδοκώδικας είναι παρόμοιος με αυτόν για το Πρόβλημα του Σακιδίου και η διατύπωσή του αφήνεται σαν άσκηση στον αναγνώστη.

Ο αλγόριθμος δυναμικού προγραμματισμού έχει χρόνο εκτέλεσης $\Theta(nS)$ και χρησιμοποιεί $\Theta(S)$ θέσεις μνήμης. Ο αριθμός των διαφορετικών υποσυνόλων του X είναι 2^n. Επομένως, ο χρόνος εκτέλεσης για τον αλγόριθμο εξαντλητικής αναζήτησης είναι $\Omega(n2^n)$, δραματικά μεγαλύτερος από το χρόνο εκτέλεσης του αλγόριθμου δυναμικού προγραμματισμού. □

Άσκηση 3.8. Το Πρόβλημα της Διαμέρισης ανήκει στην κατηγορία των *προβλημάτων απόφασης*. Τυπικά, ένα πρόβλημα απόφασης διατυπώνεται με τη μορφή μιας λογικής πρότασης για την οποία πρέπει να αποφασίσουμε αν είναι ικανοποιήσιμη. Η "λύση" ενός προβλήματος απόφασης είναι *Ναι* αν η πρόταση είναι ικανοποιήσιμη και *Όχι* διαφορετικά, δηλαδή έχει μήκος μόλις ενός δυαδικού ψηφίου. Στη συντριπτική πλειοψηφία των περιπτώσεων, ένα πρόβλημα απόφασης μπορεί να θεωρηθεί σαν *εξειδίκευση* ενός (ή και περισσοτέρων) προβλημάτων βελτιστοποίησης: Δίνεται μία *τιμή κατωφλίου* B και η ερώτηση είναι "υπάρχει λύση με κόστος το πολύ B;" όταν πρόκειται για πρόβλημα ελαχιστοποίησης , και "υπάρχει λύση με κέρδος τουλάχιστον B;" για πρόβλημα μεγιστοποίησης. Η λύση για το πρόβλημα απόφασης είναι "Ναι" αν και μόνο αν η βέλτιστη αντικειμενική τιμή είναι B ή καλύτερη. Επομένως, αν γνωρίζουμε τη βέλτιστη αντικειμενική τιμή, λύνουμε εύκολα το αντίστοιχο πρόβλημα απόφασης. Ισχύει όμως και το αντίστροφο: Αν μπορούμε να απαντήσουμε την ερώτηση, τότε υπολογίζουμε τη βέλτιστη αντικειμενική σε πολυωνυμικό χρόνο χρησιμοποιώντας δυαδική αναζήτηση (βλ. αντίστοιχη ενότητα στο κεφάλαιο για υπολογιστική πολυπλοκότητα).

Διατυπώστε ένα πρόβλημα βελτιστοποίησης του οποίου μια εξειδίκευση αποτελεί το Πρόβλημα της Διαμέρισης.

Λύση. Να διαιρέσετε το σύνολο $X = \{x_1, \ldots, x_n\}$ σε δύο σύνολα X_1 και X_2 ώστε να ελαχιστοποιηθεί το μεγαλύτερο από τα δύο αθροίσματα:

$$\max\left\{ \sum_{i \in X_1} x_i, \sum_{i \in X_2} x_i \right\}$$

Έστω $B = \frac{1}{2}\sum_{i \in X} x_i$ τιμή κατωφλίου. Η βέλτιστη αντικειμενική τιμή είναι το πολύ B αν και μόνο αν το αντίστοιχο στιγμιότυπο του Προβλήματος της Διαμέρισης έχει λύση "Ναι". □

3.3.1 Αλγόριθμοι Ψευδο-Πολυωνυμικού Χρόνου

Αναφέραμε ότι το Πρόβλημα του Σακιδίου είναι δύσκολο για την κλάση **NP** και αποτελεί κοινή επιστημονική πεποίθηση ότι για τέτοιου είδους δεν υπάρχουν αλγόριθμοι με χρόνο εκτέλεσης χειρότερης περίπτωσης που είναι *πολυωνυμικός* στο μέγεθος του στιγμιότυπου εισόδου. Από την άλλη πλευρά, ο αλγόριθμος δυναμικού προγραμματισμού έχει χρόνο εκτέλεσης που είναι πολυωνυμικός στον αριθμό των αντικειμένων n και στο μέγεθος του σακιδίου B.

Η παραπάνω αντίφαση είναι μόνο φαινομενική. Το μέγεθος του στιγμιότυπου εισόδου είναι ίσο με τον αριθμό των δυαδικών ψηφίων που χρειάζονται για να αποθηκευθεί το στιγμιότυπο στη μνήμη του υπολογιστή. Η δυαδική κωδικοποίηση του B χρησιμοποιεί $\log B$ και όχι B δυαδικά ψηφία. Έτσι ο παράγοντας B στο χρόνο εκτέλεσης του αλγόριθμου μεγαλώνει εκθετικά γρήγορα ως προς την ποσότητα $\log B$ που αποτελεί μέτρο για το μέγεθος του στιγμιότυπου εισόδου. Συνεπώς, ο χρόνος εκτέλεσης του αλγόριθμου δυναμικού προγραμματισμού δεν είναι (και δεν πρέπει) να χαρακτηρισθεί πολυωνυμικός στο μέγεθος του στιγμιότυπου εισόδου.

Το Ακέραιο Πρόβλημα του Σακιδίου αποτελεί ένα τυπικό παράδειγμα *αριθμητικού προβλήματος* (number problem). Ένα πρόβλημα χαρακτηρίζεται αριθμητικό όταν το μέγεθος των αριθμών που εμπλέκονται σε ένα στιγμιότυπο του προβλήματος μπορεί να είναι μεγαλύτερο από οποιοδήποτε πολυώνυμο του πλήθους των "βασικών συνιστωσών" του. Στο Πρόβλημα του Σακιδίου οι "βασικές συνιστώσες" είναι τα αντικείμενα και το πλήθος τους είναι n. Όμως το μέγεθος ή η αξία κάποιων αντικειμένων μπορεί να ξεπερνά κατά πολύ το 2^n. Σε αυτή την περίπτωση, είναι το μέγεθος των αριθμών και όχι το πλήθος των αντικειμένων που καθορίζει το μέγεθος του στιγμιότυπου εισόδου.

Το μέγεθος του στιγμιότυπου εισόδου ενός αριθμητικού προβλήματος καθορίζεται από το πλήθος των "βασικών συνιστωσών" του και από το πλήθος των δυαδικών ψηφίων που απαιτούνται για την αποθήκευση των αριθμών που εμφανίζονται στο στιγμιότυπο. Το πλήθος των δυαδικών ψηφίων για την αποθήκευση ενός αριθμού είναι λογαριθμικό στο μέγεθος του αριθμού. Ένας αλγόριθμος έχει *πολυωνυμικό* χρόνο εκτέλεσης όταν ολοκληρώνεται σε χρόνο που δεν ξεπερνά ένα πολυώνυμο του πλήθους των "βασικών συνιστωσών" και του *λογάριθμου* του μεγαλύτερου αριθμού που εμφανίζεται στο στιγμιότυπο.

Για αρκετά αριθμητικά προβλήματα υπάρχουν αλγόριθμοι με χρόνο εκτέλεσης πολυωνυμικό στο πλήθος των "βασικών συνιστωσών" (που συνήθως συμβολίζεται με n) και στο μέγεθος των αριθμών που εμφανίζονται στο στιγμιότυπο. Τέτοιοι αλγόριθμοι χαρακτηρίζονται σαν *αλγόριθμοι ψευδο-πολυωνυμικού χρόνου* (pseudo-polynomial-time algorithms). Ένα παράδειγμα αλγόριθμου ψευδο-πολυωνυμικού χρόνου είναι ο αλγόριθμος δυναμικού προγραμματισμού για το Πρόβλημα του Σακιδίου.

Τυπικά, έστω x η δυαδική κωδικοποίηση κάποιου στιγμιότυπου ενός αριθμητικού προβλήματος, n το πλήθος των "βασικών συνιστωσών" του x, και $\max(x)$ η τιμή του μεγαλύτερου αριθμού που εμφανίζεται στο στιγμιότυπο. Ένας αλγόριθμος έχει ψευδο-πολυωνυμικό χρόνο εκτέλεσης αν ολοκληρώνεται σε χρόνο $\Theta((n \max(x))^k)$, για κάποια σταθερά $k \geq 1$. Από την άλλη πλευρά, ένας αλγόριθμος έχει πολυωνυμικό χρόνο εκτέλεσης αν ολοκληρώνεται σε χρόνο $\mathrm{O}((n \log \max(x))^k)$,

για κάποια σταθερά $k \geq 1$.

Κάθε ψευδο-πολυωνυμική ποσότητα, που έχει τάξη μεγέθους $\Theta((n \max(x))^k)$, μεγαλώνει εκθετικά γρήγορα σε σχέση με το μήκος της δυαδικής κωδικοποίησης (δηλαδή το μέγεθος) του στιγμιότυπου εισόδου, που έχει τάξη μεγέθους $\Theta(n \log \max(x))$. Επομένως, ένας αλγόριθμος ψευδο-πολυωνυμικού χρόνου δεν έχει πολυωνυμικό αλλά *εκθετικό* χρόνο εκτέλεσης!

Τυπικά παραδείγματα αλγόριθμων ψευδο-πολυωνυμικού χρόνου είναι οι αλγόριθμοι δυναμικού προγραμματισμού για το Πρόβλημα του Σακιδίου και το Πρόβλημα της Διαμέρισης (βλ. Άσκηση 3.7).

3.4 Βιβλιογραφικές Αναφορές

Η μέθοδος του δυναμικού προγραμματισμού και οι εφαρμογές της στην επίλυση προβλημάτων συνδυαστικής βελτιστοποίησης παρουσιάζεται πολύ καλά στα [11], [32], και [5]. Ο Bellman θεωρείται ότι ξεκίνησε τη συστηματική μελέτη της μεθόδου του δυναμικού προγραμματισμού. Ο αλγόριθμος δυναμικού προγραμματισμού για το Πρόβλημα του Περιοδεύοντος Πωλητή είναι των Held και Karp [22].

4 Άπληστοι Αλγόριθμοι

Οι περισσότεροι αλγόριθμοι για προβλήματα βελτιστοποίησης λειτουργούν σε βήματα κάνοντας κάποιες επιλογές για τη μορφή της βέλτιστης λύσης σε κάθε βήμα. Στο δυναμικό προγραμματισμό, υπολογίζουμε τη βέλτιστη λύση όλων των υπο-στιγμιοτύπων και επιλέγουμε τις επιμέρους λύσεις που πρέπει να συνδυαστούν για να σχηματίσουν τη βέλτιστη λύση. Έτσι λύνουμε πολύ περισσότερα υπο-στιγμιότυπα από αυτά που πραγματικά χρειάζονται για να υπολογίσουμε τη βέλτιστη λύση και αυξάνουμε σημαντικά το χρόνο εκτέλεσης του αλγόριθμου.

Μια διαφορετική στρατηγική είναι να επιλέγουμε αυτό που δείχνει καλύτερο με βάση την τρέχουσα κατάσταση και να λύνουμε (εφαρμόζοντας την ίδια στρατηγική) το υπο-στιγμιότυπο που προκύπτει από αυτή την επιλογή. Αυτή είναι η βασική στρατηγική κάθε *άπληστου αλγόριθμου* (greedy algorithm). Με άλλα λόγια, οι άπληστοι αλγόριθμοι κάνουν *τοπικά* βέλτιστες επιλογές ελπίζοντας ότι αυτές θα οδηγήσουν σε μία *συνολικά* βέλτιστη λύση.

Οι άπληστοι αλγόριθμοι εφαρμόζονται σε προβλήματα βελτιστοποίησης. Η βασική δομή ενός άπληστου αλγόριθμου είναι ιδιαίτερα απλή. Ο αλγόριθμος λειτουργεί σε βήματα και κάνει μια επιλογή σε σχέση με τη μορφή της λύσης σε κάθε βήμα. Ο άπληστος αλγόριθμος:

1. ταξινομεί τις "βασικές συνιστώσες" του στιγμιότυπου εισόδου ως προς κάποιο κριτήριο που εξαρτάται από το πρόβλημα,

2. επιλέγει αμετάκλητα αν θα συμπεριλάβει την καλύτερη (ως προς το συγκεκριμένο κριτήριο) "βασική συνιστώσα" στη λύση, και

3. εφαρμόζει τον εαυτό του στο υπο-στιγμιότυπο που προκύπτει με βάση την παραπάνω επιλογή.

Με βάση την παραπάνω περιγραφή, ένας άπληστος αλγόριθμος χαρακτηρίζεται από το κριτήριο ταξινόμησης των "βασικών συνιστώσων" και από το κριτήριο επιλογής της καλύτερης "βασικής συνιστώσας" στη λύση.

Ένας άπληστος αλγόριθμος ονομάζεται *προσαρμοστικός* (adaptive) αν μπορεί να διαφοροποιήσει την ταξινόμηση των "βασικών συνιστωσών" από βήμα σε βήμα, και *μη-προσαρμοστικός* αν ακολουθεί την αρχική ταξινόμηση των "βασικών συνιστωστών" σε όλα τα βήματα.

Η επιλογή για κάθε "βασική συνιστώσα" είναι *αμετάκλητη* (irrevocable), δηλαδή δεν μπορεί να αλλάξει στο μέλλον. Η καλύτερη "βασική συνιστώσα" εξαιρείται από το υπο-στιγμιότυπο που θα εξεταστεί στο επόμενο βήμα. Αν έχει επιλεγεί στη λύση, τότε οι υπόλοιπες παράμετροι του υπο-στιγμιότυπου αναπροσαρμόζονται με βάση αυτή την επιλογή. Διαφορετικά, η καλύτερη "βασική συνιστώσα" αγνοείται. Όλες οι υπόλοιπες "βασικές συνιστώσες" (εκτός από την καλύτερη) συμπεριλαμβάνονται στο υπο-στιγμιότυπο για να εξετασθούν στα επόμενα βήματα.

Σε κάθε βήμα, ο άπληστος αλγόριθμος εφαρμόζει την ίδια στρατηγική στο υπο-στιγμιότυπο που προκύπτει από τις επιλογές των προηγούμενων βημάτων. Όμως η εφαρμογή του άπληστου αλγόριθμου στο υπο-στιγμιότυπο δεν είναι επί της ουσίας αναδρομική, αφού σε κάθε βήμα έχουμε περιορισμό / εξειδίκευση και όχι διαίρεση του στιγμιότυπου. Έτσι οι άπληστοι αλγόριθμοι διατυπώνονται επαναληπτικά και δεν θεωρούνται αναδρομικοί αλγόριθμοι. Το γεγονός ότι εφαρμόζουν την ίδια στρατηγική σε όλο και μικρότερα στιγμιότυπα βρίσκει εφαρμογή στην απόδειξη ορθότητας, η οποία γίνεται συνήθως με μαθηματική επαγωγή.

Οι άπληστοι αλγόριθμοι είναι συνήθως απλοί στη σύλληψη, εύκολοι στο σχεδιασμό, και ιδιαίτερα αποδοτικοί από πλευράς απαιτήσεων σε υπολογιστικούς πόρους (π.χ. χρόνος εκτέλεσης, αριθμός θέσεων μνήμης). Όμως για πολλά προβλήματα, η άπληστη στρατηγική δεν οδηγεί στη βέλτιστη λύση. Η επιτυχία ενός άπληστου αλγόριθμου βασίζεται στις ιδιότητες της *άπληστης επιλογής* (greedy-choice) και των βέλτιστων επιμέρους λύσεων. Η ιδιότητα της άπληστης επιλογής εξασφαλίζει ότι οι τοπικά βέλτιστες επιλογές μπορούν να οδηγήσουν σε μια συνολικά βέλτιστη λύση. Η ιδιότητα των βέλτιστων επιμέρους λύσεων πρέπει να ισχύει για τα υπο-στιγμιότυπα που προκύπτουν από τις επιλογές που γίνονται σε κάθε βήμα. Η ανάλυση κάθε άπληστου αλγόριθμου περιλαμβάνει τη λεγόμενη *απόδειξη ορθότητας*, όπου αποδεικνύεται ότι ο αλγόριθμος υπολογίζει πράγματι μια βέλτιστη λύση.

4.1 Επιλογή Ανταγωνιστικών Δραστηριοτήτων

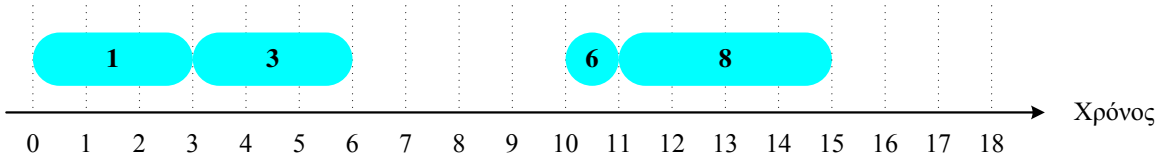
Στην απλούστερή του μορφή, ένα πρόβλημα *Επιλογής Ανταγωνιστικών Δραστηριοτήτων* (activity selection problem) αποτελείται από ένα σύνολο δραστηριοτήτων $A = \{1, 2, \ldots, n\}$ που απαιτούν τη χρήση ενός κοινόχρηστου πόρου για να εκτελεστούν (π.χ. επεξεργαστή προκειμένου για υπολογιστικές διεργασίες, αίθουσας διδασκαλίας για μαθήματα, κλπ.). Κάθε δραστηριότητα $i \in A$ απαιτεί αποκλειστική χρήση του πόρου από την έναρξή της, τη χρονική στιγμή s_i, μέχρι την ολοκλήρωσή της, τη χρονική στιγμή f_i, $f_i > s_i \geq 0$. Το ζητούμενο είναι να υπολογισθεί ο μέγιστος αριθμός δραστηριοτήτων που μπορεί να δρομολογηθεί.

Από τους περιορισμούς του προβλήματος, το πολύ μία δραστηριότητα μπορεί να βρίσκεται σε εξέλιξη κάθε χρονική στιγμή. Θεωρούμε ότι η εκτέλεση της δραστηριότητας i δεσμεύει τον κοινόχρηστο πόρο για το διάστημα $[s_i, f_i)$. Δύο δραστηριότητες i και j είναι *συμβατές* όταν η εκτέλεση της μία δεν αποκλείει την εκτέλεση της άλλης, δηλαδή όταν $[s_i, f_i) \cap [s_j, f_j) = \emptyset$. Η βέλτιστη λύση αντιστοιχεί σε ένα σύνολο συμβατών δραστηριοτήτων με μέγιστο πληθικό αριθμό.

Οι "βασικές συνιστώσες" για το πρόβλημα επιλογής ανταγωνιστικών δραστηριοτήτων είναι οι δραστηριότητες. Με βάση τη γενική περιγραφή, για τη διατύπωση ενός άπληστου αλγόριθμου πρέπει να προσδιορίσουμε ένα κριτήριο ταξινόμησης και ένα κριτήριο επιλογής για τις δραστηριότητες.

Υπάρχουν διάφορα κριτήρια ταξινόμησης των δραστηριοτήτων. Η βασική ιδέα είναι ότι η "καλύτερη" δραστηριότητα επιτρέπει τη μέγιστη εκμετάλευση του κοινόχρηστου πόρου από τις υπόλοιπες. Θα μπορούσαμε λοιπόν να ταξινομήσουμε τις δραστηριότητες ώς προς τη διάρκεια (δηλ. σε αύξουσα σειρά των $f_i - s_i$), ως προς το χρόνο έναρξης (δηλ. σε αύξουσα σειρά των s_i), ως προς το χρόνο ολοκλήρωσης (δηλ. σε αύξουσα σειρά των f_i), ή ακόμη και ως προς το βαθμό επικάλυψης με τις υπόλοιπες δραστηριότητες.

Θα αναλύσουμε τον άπληστο αλγόριθμο που ταξινομεί τις δραστηρότητες σε αύξουσα σειρά



Σχήμα 4.1: Μια βέλτιστη λύση για το παράδειγμα του προβλήματος επιλογής ανταγωνιστικών δραστηριοτήτων.

ως προς το χρόνο ολοκλήρωσης. Χωρίς βλάβη της γενικότητας, υποθέτουμε ότι η αρίθμηση των δραστηριοτήτων είναι τέτοια ώστε $f_1 \leq f_2 \leq \cdots \leq f_n$ (μπορεί να επιτευχθεί σε χρόνο $O(n \log n)$ εκτελώντας έναν αλγόριθμο ταξινόμησης). Κριτήριο επιλογής είναι η συμβατότητα της καλύτερης δραστηριότητας στο τρέχον υπο-στιγμιότυπο με τις δραστηριότητες που έχουν επιλεγεί σε προηγούμενα βήματα. Ακολουθεί η διατύπωση αυτού του αλγόριθμου σε ψευδοκώδικα:

```
greedySelection((s_1, f_1), ..., (s_n, f_n))   /* Υποθέτουμε ότι f_1 ≤ f_2 ≤ ··· ≤ f_n */
    C ← {1}; j ← 1;
    for i ← 2 to n do
        if s_i ≥ f_j then
            C ← C ∪ {i}; j ← i;
    return(C);
```

Οι δραστηριότητες ελέγχονται σε αύξουσα σειρά χρόνου ολοκλήρωσης. Οι επιλεγμένες δραστηριότητες τοποθετούνται στο σύνολο C και η μεταβλητή j διατηρεί το δείκτη της τελευταίας δραστηριότητας που έχει επιλεγεί. Αν η δραστηριότητα i επιλεγεί, ο κοινόχρηστος πόρος δεσμεύεται μέχρι τη χρονική στιγμή f_i που η i ολοκληρώνεται. Μπορούμε να θεωρήσουμε ότι ο αλγόριθμος συνεχίζει εφαρμόζοντας τον εαυτό του στο υπο-στιγμιότυπο $A_i = \{k \in A : s_k \geq f_i\}$ από το οποίο έχουν εξαιρεθεί οι δραστηριότητες που δεν είναι συμβατές με την i.

Πρόκειται για έναν μη-προσαρμοστικό άπληστο αλγόριθμο, αφού οι δραστηριότητες ταξινομούνται μια και μόνη φορά στην αρχή του αλγορίθμου. Ο χρόνος εκτέλεσης του αλγόριθμου είναι $\Theta(n)$ αν υποθέσουμε ότι οι δραστηριότητες δίνονται ταξινομημένες σε αύξουσα σειρά χρόνου ολοκλήρωσης.

Για παράδειγμα, ας θεωρήσουμε τη λειτουργία του αλγόριθμου greedySelection για το παρακάτω στιγμιότυπο

i	1	2	3	4	5	6	7	8	9	10
s_i	0	2	3	5	4	10	9	11	11	14
f_i	3	5	6	7	9	11	12	15	16	17

Ο αλγόριθμος επιλέγει τη δραστηριότητα 1, οπότε αποκλείεται η δραστηριότητα 2. Στη συνέχεια, επιλέγεται η δραστηριότητα 3, αποκλείονται οι δραστηριότητες 4 και 5, επιλέγεται η 6, αποκλείεται η 7, επιλέγεται η 8, και αποκλείονται οι 9 και 10. Το αποτέλεσμα είναι η επιλογή τεσσάρων αμοιβαία συμβατών δραστηριοτήτων που δρομολογούνται όπως φαίνεται στο Σχήμα 4.1.

Το σύνολο $\{1, 3, 6, 8\}$ αποτελεί μία βέλτιστη λύση για αυτό το στιγμιότυπο, αφού δεν υπάρχει κανένα σύνολο πέντε ή περισσότερων συμβατών δραστηριοτήτων. Μία διαφορετική βέλτιστη λύση είναι το σύνολο $\{2, 4, 7, 10\}$. Παρατηρούμε ότι η δραστηριότητα 2 μπορεί να αντικατασταθεί από την 1 χωρίς να επηρεαστούν οι υπόλοιπες δραστηριότητες της δεύτερης λύσης. Αυτό συμβαίνει γιατί $f_1 \leq f_2$. Στη συνέχεια, η δραστηριότητα 4 μπορεί να αντικατασταθεί από την 3, η 7 από την 6, και η 10 από την 8 (με αυτή τη σειρά).

Άσκηση 4.1. Θεωρείστε τους άπληστους αλγόριθμους που ταξινομούν τις δραστηριότητες σε αύξουσα σειρά ως προς τη διάρκεια, ως προς το χρόνο έναρξης, και ως προς το βαθμό επικάλυψης με τις υπόλοιπες δραστηριότητες, και χρησιμοποιούν σαν κριτήριο επιλογής τη συμβατότητα με τις ήδη επιλεγμένες δραστηριότητες. Να δώσετε αντιπαραδείγματα όπου αυτοί οι αλγόριθμοι αποτυχάνουν να υπολογίσουν τη βέλτιστη λύση.

Λύση. Έστω ο αλγόριθμος που ταξινομεί τις δραστηριότητες σε αύξουσα σειρά ως προς τη διάρκεια. Στο στιγμιότυπο $((1, 5), (4, 6), (5, 10))$ ο αλγόριθμος επιλέγει μόνο τη δραστηριότητα 2, ενώ η βέλτιστη λύση επιλέγει τις δραστηριότητες 1 και 3.

Έστω ο αλγόριθμος που ταξινομεί τις δραστηριότητες σε αύξουσα σειρά ως προς το χρόνο έναρξης. Στο στιγμιότυπο $((1, 10), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 10))$, ο αλγόριθμος επιλέγει μόνο τη δραστηριότητα 1, ενώ η βέλτιστη λύση επιλέγει όλες τις υπόλοιπες δραστηριότητες.

Έστω ο αλγόριθμος που ταξινομεί τις δραστηριότητες σε αύξουσα σειρά ως προς το βαθμό επικάλυψης. Στο στιγμιότυπο $((1, 3), (1, 3), (2, 4), (2, 4), (3, 5), (4, 6), (5, 7), (6, 8), (6, 8), (7, 9), (7, 9))$, ο αλγόριθμος επιλέγει αμετάκλητα τη δραστηριότητα 6. Στη συνέχεια, ο αλγόριθμος μπορεί να επιλέξει το πολύ άλλες δύο δραστηριότητες, π.χ. τις 1 και 8. Αντίθετα, η βέλτιστη λύση επιλέγει τέσσερις δραστηριότητες, π.χ. τις 1, 5, 7, και 10. □

4.1.1 Απόδειξη Ορθότητας

Στην Άσκηση 4.1 είδαμε τρία παραδείγματα άπληστων αλγορίθμων που αποτυγχάνουν να υπολογίσουν τη βέλτιστη λύση σε κάποιες περιπτώσεις. Ο μόνος τρόπος να είμαστε σίγουροι ότι ένας άπληστος αλγόριθμος υπολογίζει τη βέλτιστη λύση για κάθε στιγμιότυπο εισόδου είναι να διατυπώσουμε μια μαθηματική απόδειξη. Τέτοιου είδους αποδείξεις ονομάζονται *αποδείξεις ορθότητας* και αποτελούν απαραίτητο συστατικό της ανάλυσης κάθε άπληστου αλγόριθμου.

Θεώρημα 4.1. *Ο αλγόριθμος greedySelection υπολογίζει μία βέλτιστη λύση για το Πρόβλημα της Επιλογής Ανταγωνιστικών Δραστηριοτήτων.*

Απόδειξη. Έστω A ένα σύνολο δραστηριοτήτων ταξινομημένων σε αύξουσα σειρά χρόνων ολοκλήρωσης. Ο αλγόριθμος greedySelection επιλέγει τη δραστηριότητα 1 (αυτή δηλαδή με το μικρότερο χρόνο ολοκλήρωσης) και εφαρμόζει τον εαυτό του στο υπο-στιγμιότυπο $A_1 = \{i \in A : s_i \geq f_1\}$. Έστω $C(A)$ ($C(A_1)$) ο αριθμός των δραστηριοτήτων που επιλέγει ο αλγόριθμος από το σύνολο A (από το A_1 αντίστοιχα). Ισχύει ότι $C(A) = 1 + C(A_1)$.

Θα αποδείξουμε το θεώρημα με μαθηματική επαγωγή. Αν έχουμε μόνο μία δραστηριότητα, ο αλγόριθμος την επιλέγει και αυτή είναι η βέλτιστη λύση. Επαγωγικά υποθέτουμε ότι ο αλγόριθμος υπολογίζει τη βέλτιστη λύση για κάθε σύνολο από $n - 1$ ή λιγότερες δραστηριότητες. Θα αποδείξουμε τον ισχυρισμό για κάθε σύνολο A με n δραστηριότητες.

Έστω $C^*(A)$ ($C^*(A_1)$) η βέλτιστη αντικειμενική τιμή (μέγιστος αριθμός συμβατών δραστηριοτήτων) για το σύνολο A (για το A_1 αντίστοιχα). Από την επαγωγική υπόθεση, ο αλγόριθμος υπολογίζει τη βέλτιστη λύση για το A_1. Έτσι $C(A_1) = C^*(A_1)$ και $C(A) = 1 + C^*(A_1)$. Θα δείξουμε ότι $C^*(A) \leq 1 + C^*(A_1) = C(A)$ που σημαίνει ότι ο αλγόριθμος υπολογίζει τη βέλτιστη λύση για το σύνολο A.

Έστω μια βέλτιστη λύση C^* για το σύνολο A, και έστω ℓ η δραστηριότητα του C^* που ολοκληρώνεται πρώτη. Ισχύει ότι $f_\ell \geq f_1$ επειδή υποθέσαμε ότι οι δραστηριότητες είναι ταξινομημένες σε αύξουσα σειρά χρόνου ολοκλήρωσης[1]. Επομένως κάθε δραστηριότητα $i \in C^* \setminus \{\ell\}$ έχει χρόνο έναρξης $s_i \geq f_\ell \geq f_1$ και το $C^* \setminus \{\ell\}$ αποτελεί αποδεκτή λύση για το υπο-στιγμιότυπο A_1. Ο αριθμός των δραστηριοτήτων στο $C^* \setminus \{\ell\}$ είναι $C^*(A) - 1$ εξ' ορισμού. Επειδή η βέλτιστη λύση επιλέγει τις περισσότερες δραστηριότητες από κάθε άλλη αποδεκτή λύση,

$$C^*(A) - 1 \leq C^*(A_1) \Rightarrow C^*(A) \leq 1 + C^*(A_1) = C(A)$$

όπως απαιτείται. □

Επισήμανση. Ουσιαστικά αποδείξαμε ότι η βέλτιστη αντικειμενική τιμή περιγράφεται από τη σχέση $C^*(A) = 1 + C^*(A_1)$. Επομένως, το πρόβλημα έχει την ιδιότητα της άπληστης επιλογής, αφού η επιλογή της δραστηριότητας που ολοκληρώνεται πρώτη οδηγεί σε μια συνολικά βέλτιστη λύση, και την ιδιότητα των βέλτιστων επιμέρους λύσεων, αφού η βέλτιστη λύση αποτελείται από την πρώτη δραστηριότητα και τη βέλτιστη λύση του υπο-στιγμιότυπου A_1. Η απόδειξη του Θεωρήματος 4.1 συνίσταται στην επαγωγική εφαρμογή αυτών των δύο ιδιοτήτων. □

Η επαγωγική απόδειξη του Θεωρήματος 4.1 είναι μια τυπική απόδειξη ορθότητας άπληστου αλγόριθμου. Η πρώτη παρατήρηση είναι ότι κάθε βέλτιστη λύση πρέπει να περιέχει μια βέλτιστη λύση για το υπο-στιγμιότυπο στο οποίο ο άπληστος αλγόριθμος εφαρμόζει τον εαυτό του. Δηλαδή το πρόβλημα έχει την ιδιότητα των βέλτιστων επιμέρους λύσεων. Η δεύτερη παρατήρηση είναι ότι μπορούμε να αντικαταστήσουμε μια από τις επιλογές της βέλτιστης λύσης με την άπληστη επιλογή του αλγόριθμου. Δηλαδή το πρόβλημα έχει την ιδιότητα της άπληστης επιλογής, αφού υπάρχει μια βέλτιστη λύση που συμφωνεί με την άπληστη επιλογή. Η απόδειξη ορθότητας συνίσταται στην επαγωγική εφαρμογή των δύο ιδιοτήτων.

Άσκηση 4.2. Να διατυπώσετε έναν αλγόριθμο δυναμικού προγραμματισμού για το πρόβλημα επιλογής ανταγωνιστικών δραστηριοτήτων. Μπορείτε να υποθέσετε ότι οι δραστηριότητες είναι διατεταγμένες σε αύξουσα σειρά χρόνου ολοκλήρωσης.

Λύση. Έστω $m[i, T]$ ο μέγιστος αριθμός δραστηριοτήτων που μπορούν να επιλεγούν από το σύνολο $A_i = \{1, \ldots, i\}$ και να ολοκληρωθούν μέχρι τη στιγμή T. Το $m[i, T]$ ορίζεται από την παρακάτω αναδρομική εξίσωση:

[1] Αυτό σημαίνει ότι η δραστηριότητα ℓ μπορεί να αντικατασταθεί από την 1 χωρίς να επηρεαστούν οι υπόλοιπες δραστηριότητες στο C^*. Με άλλα λόγια, υπάρχει μια βέλτιστη λύση που περιέχει τη δραστηριότητα που ολοκληρώνεται πρώτη και το πρόβλημα έχει την ιδιότητα της άπληστης επιλογής.

$$m[i,T] = \begin{cases} m[i-1,T] & \text{αν } i > 0 \text{ και } 0 \leq T < f_i \\ \max\{m[i-1,T], m[i-1,s_i]+1\} & \text{αν } i > 0 \text{ και } T \geq f_i \\ 0 & \text{αν } i = 0 \text{ ή } T < 0 \end{cases}$$

Η βέλτιστη αντικειμενική τιμή δίνεται από το στοιχείο $m[n, f_n]$, όπου f_n χρόνος ολοκλήρωσης της τελευταίας δραστηριότητας. Η βέλτιστη αντικειμενική τιμή καθώς και η βέλτιστη λύση μπορεί να υπολογισθεί με τη μέθοδο του δυναμικού προγραμματισμού σε χρόνο $\Theta(n\, f_n)$ και χρησιμοποιώντας $\Theta(n\, f_n)$ θέσεις μνήμης.

Ένας διαφορετικός αλγόριθμος που αν και ο ορισμός του ακολουθεί το παράδειγμα του δυναμικού προγραμματισμού, λειτουργεί όπως ο άπληστος αλγόριθμος, προκύπτει ως εξής: Έστω $m[i] = (x_i, y_i)$, όπου x_i ο μέγιστος αριθμός δραστηριοτήτων που μπορεί να επιλεγεί από το σύνολο $A_i = \{1, \ldots, i\}$ και y_i ο χρόνος ολοκλήρωσης των επιλεγμένων διαδικασιών. Το $m[i]$ ορίζεται από την παρακάτω αναδρομική εξίσωση:

$$m[i] = \begin{cases} (x_{i-1}, y_{i-1}) & \text{αν } i > 0 \text{ και } s_i < y_{i-1} \\ (x_{i-1}+1, f_i) & \text{αν } i > 0 \text{ και } s_i \geq y_{i-1} \\ (0,0) & \text{αν } i = 0 \end{cases}$$

Θεωρώντας τις δραστηριότητες ταξινομημένες σε αύξουσα σειρά χρόνου ολοκλήρωσης, το $m[n]$ μπορεί να υπολογιστεί σε γραμμικό χρόνο με χρήση σταθερού αριθμού θέσεων μνήμης. □

Άσκηση 4.3. Έστω n δραστηριότητες, όπου η δραστηριότητα i, $i = 1, \ldots, n$, έχει χρόνο έναρξης s_i και χρόνο ολοκλήρωσης f_i, $f_i > s_i \geq 0$. Το ζητούμενο είναι να δρομολογήσουμε όλες τις δραστηριότητες χρησιμοποιώντας τον ελάχιστο αριθμό από κοινόχρηστους πόρους (π.χ. αίθουσες διδασκαλίας προκειμένου για μαθήματα). Όπως και στο πρόβλημα της επιλογής ανταγωνιστικών δραστηριοτήτων, οι δραστηριότητες που δρομολογούνται στον ίδιο πόρο πρέπει να είναι συμβατές. Να διατυπώσετε και να αναλύσετε έναν άπληστο αλγόριθμο που υπολογίζει μια βέλτιστη λύση.

Λύση. Το πρόβλημα αυτό είναι πιο γνωστό σαν Πρόβλημα *Χρωματισμού Διαστημάτων* (interval coloring problem). Οι δραστηριότητες αντιστοιχούν σε χρονικά διαστήματα και κάθε χρώμα σε διαφορετικό κοινόχρηστο πόρο. Το ζητούμενο είναι ο χρωματισμός των διαστημάτων με τον ελάχιστο αριθμό χρωμάτων, ώστε κάθε ζευγάρι επικαλυπτόμενων διαστημάτων να έχει διαφορετικά χρώματα.

Έστω d ο μέγιστος αριθμός διαστημάτων (δραστηριοτήτων) που επικαλύπτονται (αντίστοιχα, η εκτέλεσή τους συμπίπτει για κάποια χρονική στιγμή). Προφανώς, χρειαζόμαστε τουλάχιστον d χρώματα. Ένας άπληστος αλγόριθμος είναι να ταξινομήσουμε τα διαστήματα σε αύξουσα σειρά χρόνου έναρξης. Σε κάθε διάστημα αναθέτουμε το μικρότερο χρώμα που είναι διάθεσιμο κατά τη χρονική στιγμή έναρξης. Ένα χρώμα που έχει ανατεθεί σε ένα διάστημα γίνεται πάλι διαθέσιμο τη χρονική στιγμή που το διάστημα ολοκληρώνεται. Παρατηρούμε ότι σε κάθε χρονική στιγμή k, ο αλγόριθμος χρησιμοποιεί ακριβώς τόσα χρώματα όσα και τα διαστήματα που περιέχουν το k. Επομένως, ο αλγόριθμος χρησιμοποιεί ακριβώς d χρώματα.

```
greedyIntervalColoring((s_1, f_1), ..., (s_n, f_n))
    F ← max_{1≤i≤n}{f_i}; c ← 1;
    for k ← 0 to F do
        for all j ∈ {i : (1 ≤ i ≤ n) ∧ (f_i = k)} do
            c ← c − 1;   /* Το διάστημα j ολοκληρώνεται και το χρώμα c γίνεται διαθέσιμο */
        for all j ∈ {i : (1 ≤ i ≤ n) ∧ (s_i = k)} do
            C[j] ← c; c ← c + 1;   /* Το διάστημα j ξεκινάει και παίρνει το χρώμα c */
    return(C);
```

Αυτή η υλοποίηση έχει χρόνο εκτέλεσης $\Theta(n\,F)$. Ο χρόνος εκτέλεσης μπορεί να βελτιωθεί αν επικεντρώσουμε την προσοχή μας μόνο στα σημεία που είτε ξεκινάει είτε ολοκληρώνεται κάποιο διάστημα. Υπάρχουν το πολύ $2n$ διαφορετικά σημεία έναρξης και ολοκλήρωσης. Κάθε τέτοιο σημείο μπορεί να υπολογιστεί σε σταθερό αν τα s_i και f_i είναι ταξινομημένα σε αύξουσα σειρά. Ο τροποποιημένος αλγόριθμος χρειάζεται $\Theta(n \log n)$ χρόνο για την ταξινόμηση και $\Theta(n)$ χρόνο για την ανάθεση των χρωμάτων. □

4.2 Δρομολόγηση Εργασιών

Έστω ένας εξυπηρετητής (π.χ. υπολογιστής, εκτυπωτής, ταμείο, μέσο μεταφοράς, κλπ.) και n πελάτες / εργασίες με ατομικούς χρόνους εξυπηρέτησης $t_1, t_2, \ldots, t_n$. Θέλουμε να δρομολογήσουμε τους πελάτες στον εξυπηρετητή ώστε να ελαχιστοποιήσουμε το μέσο χρόνο εξυπηρέτησης. Το πρόβλημα αυτό είναι γνωστό σαν πρόβλημα Δρομολόγησης για την Ελαχιστοποίηση του Μέσου Χρόνου Εξυπηρέτησης και αποτελεί ένα αντιπροσωπευτικό δείγμα μιας μεγάλης κατηγορίας προβλημάτων βελτιστοποίησης που είναι γνωστά σαν *προβλήματα δρομολόγησης* (scheduling problems).

Για το συγκεκριμένο πρόβλημα, μια δρομολόγηση είναι απλώς μια μετάθεση του συνόλου των πελατών. Μια δρομολόγηση μπορεί να αναπαρασταθεί σαν μια μετάθεση π του συνόλου $\{1, \ldots, n\}$ των πελατών, όπου ο πελάτης i δρομολογείται στη σειρά $\pi(i)$. Κάθε τέτοια μετάθεση/δρομολόγηση αποτελεί μια αποδεκτή λύση. Ο χρόνος εξυπηρέτησης του πελάτη i σε μια συγκεκριμένη δρομολόγηση π, ας τον συμβολίσουμε με $c_i(\pi)$, είναι ίσος με το άθροισμα των ατομικών χρόνων εξυπηρέτησης όλων των πελατών με σειρά μικρότερη ή ίση του $\pi(i)$. Τυπικά,

$$c_i(\pi) = \sum_{j:\pi(j)\leq\pi(i)} t_j$$

Ο μέσος χρόνος εξυπηρέτησης των πελατών για τη δρομολόγηση π, ας τον συμβολίσουμε με $A(\pi)$, είναι

$$A(\pi) = \frac{1}{n}\sum_{i=1}^{n} c_i(\pi)$$

Ο συνολικός χρόνος εξυπηρέτησης των πελατών για τη δρομολόγηση π, ας τον συμβολίσουμε με $T(\pi)$, είναι $T(\pi) = n\,A(\pi)$. Η βέλτιστη δρομολόγηση είναι αυτή που ελαχιστοποιεί το μέσο χρόνο εξυπηρέτησης των πελατών. Επειδή το n είναι καθορισμένο για κάθε στιγμιότυπο, η βέλτιστη δρομολόγηση ελαχιστοποιεί ισοδύναμα το συνολικό χρόνο εξυπηρέτησης των πελατών.

Παράδειγμα 4.1. Έστω 4 πελάτες με ατομικούς χρόνους εξυπηρέτησης 8, 7, 2, και 5. Αν τους δρομολογήσουμε με αυτή τη σειρά, οι χρόνοι εξυπηρέτησης είναι 8, 15, 17, και 22, και ο συνολικός χρόνος εξυπηρέτησης είναι 62. Αν τους δρομολογήσουμε με τη σειρά 2, 5, 7, 8, οι χρόνοι εξυπηρέτησης είναι 2, 7, 14, και 22, και ο συνολικός χρόνος εξυπηρέτησης είναι 45. Αυτή είναι και η βέλτιστη λύση.

Υπάρχει ένας πολύ απλός άπληστος αλγόριθμος για αυτό το πρόβλημα δρομολόγησης. Ο αλγόριθμος ταξινομεί τους πελάτες σε αύξουσα σειρά ατομικών χρόνων εξηπηρέτησης και τους δρομολογεί με αυτή ακριβώς τη σειρά. Ο αλγόριθμος είναι γνωστός και σαν κανόνας Ελάχιστου Ατομικού Χρόνου Εξυπηρέτησης (ΕΑΧΕ).

Χωρίς βλάβη της γενικότητας, υποθέτουμε στο εξής ότι η αρίθμηση των πελατών είναι τέτοια ώστε $t_1 \leq t_2 \leq \cdots \leq t_n$ (μπορεί να επιτευχθεί σε χρόνο $O(n \log n)$ εκτελώντας έναν αλγόριθμο ταξινόμησης). Ο συνολικός χρόνος εξυπηρέτησης είναι:

$$\begin{aligned} T &= t_1 + (t_1 + t_2) + (t_1 + t_2 + t_3) + \cdots + (t_1 + \cdots + t_n) \\ &= n\,t_1 + (n-1)t_2 + (n-2)t_3 + \cdots + t_n \\ &= \sum_{i=1}^{n} (n - i + 1)t_i \end{aligned}$$

Ο παρακάτω ψευδοκώδικας υπολογίζει τον μέσο χρόνο εξυπηρέτησης με βάση τον κανόνα ΕΑΧΕ.

```
greedyScheduling(t_1, t_2, ..., t_n)   /* Υποθέτουμε ότι t_1 ≤ t_2 ≤ ··· ≤ t_n */
    c ← 0; T ← 0;
    for i ← 1 to n do
        c = c + t_i; T ← T + c;
    return(T/n);
```

Θεώρημα 4.2. *Ο κανόνας Ελάχιστου Ατομικού Χρόνου Εξυπηρέτησης ελαχιστοποιεί το μέσο χρόνο εξυπηρέτησης.*

Απόδειξη. Ισοδύναμα, θα αποδείξουμε ότι ο κανόνας ΕΑΧΕ ελαχιστοποιεί το συνολικό χρόνο εξυπηρέτησης. Έστω ένα σύνολο n πελατών ταξινομημένων σε αύξουσα σειρά ατομικών χρόνων εξυπηρέτησης. Για να καταλήξουμε σε άτοπο, υποθέτουμε ότι η δρομολόγηση του κανόνα ΕΑΧΕ δεν είναι βέλτιστη.

Έστω λοιπόν π^* η βέλτιστη δρομολόγηση που συμφωνεί με τη δρομολόγηση του κανόνα ΕΑΧΕ στη σειρά δρομολόγησης όσο το δυνατόν περισσότερων πελατών (σε σχέση με τις υπόλοιπες βέλτιστες δρομολογήσεις). Έστω ακόμη λ^* η αντίστροφη μετάθεση της π^*. Δηλαδή, $\lambda^*(j)$ είναι ο πελάτης που δρομολογείται στη σειρά j από την π^*. Ο ελάχιστος συνολικός χρόνος εξυπηρέτησης είναι $T(\pi^*) = \sum_{j=1}^{n} (n - j + 1)t_{\lambda^*(j)}$.

Αφού η βέλτιστη δρομολόγηση π^* είναι διαφορετική από τη δρομολόγηση του κανόνα ΕΑΧΕ, έστω k ο πρώτος πελάτης που δρομολογείται σε σειρά μεγαλύτερη από k, και έστω ℓ ο πελάτης που δρομολογείται στη σειρά k από την π^*. Χρησιμοποιώντας τον παραπάνω συμβολισμό, $\pi^*(k) > k$. Επειδή οι πελάτες μέχρι και τον $k-1$ δρομολογούνται στην ίδια σειρά από τις δύο δρομολογήσεις, ισχύει ότι $k < \ell$ και $t_k \leq t_\ell$.

Η συνεισφορά των t_k και t_ℓ στον συνολικό χρόνο εξυπηρέτησης $T(\pi^*)$ είναι $(n-\pi^*(k)+1)t_k$ και $(n-k+1)t_\ell$ αντίστοιχα. Αν ανταλλάξουμε αμοιβαία τη σειρά δρομολόγησης των k και ℓ, οπότε ο k θα δρομολογείται στη σειρά k και ο ℓ στη σειρά $\pi^*(k)$, η συνεισφορά των t_k και t_ℓ στον νέο συνολικό χρόνο εξυπηρέτησης θα είναι $(n-k+1)t_k$ και $(n-\pi^*(k)+1)t_\ell$ αντίστοιχα. Η μεταβολή στο συνολικό χρόνο εξυπηρέτησης που προκύπτει από την ανταλλαγή των k και ℓ είναι:

$$\begin{aligned}&[(n-k+1)t_k+(n-\pi^*(k)+1)t_\ell]-[(n-\pi^*(k)+1)t_k+(n-k+1)t_\ell] &= \\ &=(n-k+1)(t_k-t_\ell)-(n-\pi^*(k)+1)(t_k-t_\ell)=(\pi^*(k)-k)(t_k-t_\ell) &\leq 0\end{aligned}$$

επειδή $\pi^*(k)>k$ και $t_k\leq t_\ell$. Επομένως, η δρομολόγηση που προκύπτει από την π^* με την ανταλλαγή των k και ℓ έχει επίσης τον ελάχιστο συνολικό χρόνο εξυπηρέτησης (είναι δηλαδή βέλτιστη) και συμφωνεί με την άπληστη δρομολόγηση του κανόνα ΕΑΧΕ τουλάχιστον σε έναν ακόμη πελάτη. Αυτό αποτελεί αντίφαση στο κριτήριο επιλογής της π^*. □

Άσκηση 4.4. Αποδείξτε ότι το πρόβλημα της Δρομολόγησης για την Ελαχιστοποίηση του Μέσου Χρόνου Εξυπηρέτησης έχει τις ιδιότητες της βέλτιστης επιλογής και των βέλτιστων επιμέρους λύσεων.

Λύση. Η απόδειξη του Θεωρήματος 4.2 ουσιαστικά δείχνει ότι υπάρχει μια βέλτιστη δρομολόγηση που συμφωνεί με τη δρομολόγηση του κανόνα ΕΑΧΕ στις σειρές των k πρώτων πελατών, για κάθε $k=1,2,\ldots,n$. Δηλαδή, το πρόβλημα έχει την ιδιότητα της βέλτιστης επιλογής. Η ιδιότητα των βέλτιστων επιμέρους λύσεων προκύπτει από την εξίσωση υπολογισμού του συνολικού χρόνου εξυπηρέτησης. Ο συνολικός χρόνος εξυπηρέτησης για n πελάτες είναι $n\,t_1$ συν το συνολικό χρόνο εξυπηρέτησης για τους υπόλοιπους $n-1$ πελάτες. Αν λοιπόν αγνοήσουμε τον πρώτο πελάτη μιας βέλτιστης δρομολόγησης, προκύπτει μια βέλτιστη δρομολόγηση για τους υπόλοιπους $n-1$ πελάτες. □

Άσκηση 4.5. Θεωρούμε το πρόβλημα της κάλυψης όλων των σημείων ενός συνόλου ρητών αριθμών με τον ελάχιστο αριθμό μοναδιαίων διαστημάτων. Συγκεκριμένα, δίνεται ένα σύνολο n ρητών αριθμών $X=\{x_1,\ldots,x_n\}$ και πρέπει να υπολογιστεί η μικρότερη συλλογή κλειστών διαστημάτων μοναδιαίου μήκους που περιέχει όλα τα στοιχεία του X. Να διατυπώσετε και να αναλύσετε έναν άπληστο αλγόριθμο για αυτό το πρόβλημα.

Λύση. Θεωρούμε ότι τα στοιχεία του X είναι ταξινομημένα σε αύξουσα σειρά (κριτήριο ταξινόμησης), $x_1\leq\cdots\leq x_n$. Ο αλγόριθμος προσθέτει στη λύση ένα μοναδιαίο διάστημα με αρχή το μικρότερο στοιχείο που δεν έχει καλυφθεί, αφαιρεί από το τρέχον στιγμιότυπο τα στοιχεία που καλύφθηκαν από το νέο διάστημα, και συνεχίζει εφαρμόζοντας τον εαυτό του στο υποστιγμιότυπο που προκύπτει (σαν άσκηση, να διατυπώσετε τον αλγόριθμο σε ψευδοκώδικα). Ο χρόνος εκτέλεσης του αλγόριθμου είναι $\Theta(n\log n)$ για την ταξινόμηση των στοιχείων και $\Theta(n)$ για τον υπολογισμό των διαστημάτων (κάθε στοιχείο εξετάζεται μία μόνο φορά). Για να αποδείξουμε την ορθότητα του άπληστου αλγόριθμου, θα δείξουμε ότι το πρόβλημα έχει τις ιδιότητες της άπληστης επιλογής και των βέλτιστων επιμέρους λύσεων.

Έστω A^* μία βέλτιστη συλλογή διαστημάτων που δεν περιέχει το πρώτο διάστημα της άπληστης λύσης, δηλ. το διάστημα $[x_1,x_1+1]$. Από τους περιορισμούς του προβλήματος, η A^* περιέχει

κάποιο άλλο διάστημα J που καλύπτει το στοιχείο x_1. Αφού το x_1 είναι το μικρότερο στοιχείο του X, το διάστημα J μπορεί να αντικατασταθεί από το $[x_1, x_1+1]$. Η λύση $(A^* \setminus \{J\}) \cup \{[x_1, x_1+1]\}$ που προκύπτει από την αντικατάσταση καλύπτει όλα τα στοιχεία του X, άρα είναι αποδεκτή λύση, και έχει τον ίδιο αριθμό διαστημάτων με την A^*, άρα είναι βέλτιστη. Επομένως, υπάρχει μία βέλτιστη λύση που συμφωνεί με την πρώτη επιλογή του άπληστου αλγόριθμου (ιδιότητα άπληστης επιλογής).

Έστω $X_1 = X \setminus [x_1, x_1+1]$ το σύνολο των σημείων του X που δεν καλύπτονται από το πρώτο διάστημα της άπληστης λύσης, και έστω A^* μια βέλτιστη συλλογή διαστημάτων που περιέχει το διάστημα $[x_1, x_1+1]$. Η λύση $A_1^* = A^* \setminus \{[x_1, x_1+1]\}$ αποτελεί μια βέλτιστη λύση για το υποστιγμιότυπο X_1 (διαφορετικά θα μπορούσαμε να βελτιώσουμε την A^*). Επομένως, το πρόβλημα έχει την ιδιότητα των βέλτιστων επιμέρους λύσεων.

Η ορθότητα του άπληστου αλγόριθμου αποδεικνύεται με επαγωγική εφαρμογή των ιδιοτήτων της άπληστης επιλογής και των βέλτιστων επιμέρους λύσεων. □

Άσκηση 4.6. Έχουμε στη διάθεσή μας κέρματα αξίας 1, 5, και 20 λεπτών για να δώσουμε ρέστα χρησιμοποιώντας το μικρότερο αριθμό κερμάτων. Διατυπώστε και αναλύστε έναν άπληστο αλγόριθμο για αυτό το πρόβλημα. Δώστε ένα παράδειγμα συνόλου κερμάτων για το οποίο ο άπληστος αλγόριθμος δεν υπολογίζει πάντα τη βέλτιστη λύση.

Λύση. Ο άπληστος αλγόριθμος χρησιμοποιεί όσο περισσότερα 20λεπτα μπορεί, στη συνέχεια όσα περισσότερα 5λεπτα μπορεί, και τέλος συμπληρώνει το ποσόν με 1λεπτα.

Έστω x το συνολικό ποσόν (εκφρασμένο σε λεπτά) που πρέπει να δώσουμε ρέστα, και έστω c_{20}, c_5, και c_1 ο αριθμός των κερμάτων 20, 5, και 1 λεπτού αντίστοιχα που χρησιμοποιεί ο άπληστος αλγόριθμος για να συμπληρώσει το x. Ο αλγόριθμος θέτει:

$$c_{20} \leftarrow \lfloor x/20 \rfloor \text{ και } x_5 \leftarrow x - 20\,c_{20} \quad , \quad c_5 \leftarrow \lfloor x_5/5 \rfloor \text{ και } x_1 \leftarrow x_5 - 5\,c_5 \quad , \quad c_1 \leftarrow x_1$$

Διαπιστώνουμε ότι $20\,c_{20} + 5\,c_5 + c_1 = x$, δηλαδή ότι ο αλγόριθμος υπολογίζει μια αποδεκτή λύση. Για να αποδείξουμε ότι ο αλγόριθμος χρησιμοποιεί τον ελάχιστο αριθμό κερμάτων, θα δείξουμε ότι το πρόβλημα έχει τις ιδιότητες της άπληστης επιλογής και των βέλτιστων επιμέρους λύσεων.

Για την ιδιότητα της άπληστης επιλογής, παρατηρώ ότι η βέλτιστη λύση δεν μπορεί να χρησιμοποιεί περισσότερα από c_{20} κέρματα των 20 λεπτών. Επιπλέον, αν χρησιμοποιούσε λιγότερα, θα μπορούσαμε να αντικαταστήσουμε τέσσερα ή περισσότερα κέρματα με ένα 20λεπτο μειώνοντας τον αριθμό των κερμάτων. Άρα η βέλτιστη λύση χρησιμοποιεί ακριβώς όσα κέρματα των 20λεπτών χρησιμοποιεί και ο άπληστος αλγόριθμος. Επίσης το πρόβλημα έχει την ιδιότητα των βέλτιστων επιμέρους λύσεων. Πράγματι, αν αγνοήσουμε τα 20λεπτα μιας βέλτιστης λύσης για το ποσό x, θα πάρουμε μια βέλτιστη λύση αποτελούμενη από 5λεπτα και 1λεπτα για το ποσό $x - 20\,c_{20}$. Εφαρμόζοντας επαγωγικά τις ιδιότητες της άπληστης επιλογής και των βέλτιστων επιμέρους λύσεων αποδεικνύουμε ότι ο αλγόριθμος υπολογίζει μια βέλτιστη λύση.

Το πρόβλημα δεν έχει την ιδιότητα της άπληστης επιλογής για όλα τα σύνολα κερμάτων. Για παράδειγμα, αν έχουμε κέρματα αξίας 20, 12, και 1 λεπτών και θέλουμε να δώσουμε ρέστα 24 λεπτά, η βέλτιστη λύση είναι 2×12, ενώ ο άπληστος αλγόριθμος δίνει $1 \times 20 + 4 \times 1$. □

4.3 Κλασματικό Πρόβλημα του Σακιδίου

΄Ενα άλλο παράδειγμα προβλήματος βελτιστοποίησης που λύνεται αποδοτικά από έναν άπληστο αλγόριθμο είναι το Κλασματικό Πρόβλημα του Σακιδίου (βλ. Ενότητα 1.2). Στο Κλασματικό Πρόβλημα του Σακιδίου, δίνεται ένα σακίδιο μεγέθους $B > 0$ και n αντικείμενα, με το αντικείμενο i να έχει μέγεθος $s_i > 0$ και αξίας $p_i > 0$, $i = 1, \ldots, n$. ΄Ενα αντικείμενο i μπορεί να συμπεριληφθεί στο σακίδιο σε οποιοδήποτε ποσοστό $f_i \in [0, 1]$ καταλαμβάνοντας χώρο $f_i\, s_i$ και προσθέτοντας αξία $f_i\, p_i$. Το ζητούμενο είναι να υπολογίσουμε το ποσοστό στο οποίο πρέπει να συμπεριληφθεί κάθε αντικείμενο στο σακίδιο ώστε να μην παραβιάζεται το μέγεθος του σακιδίου και να μεγιστοποιηθεί η συνολική αξία. Με άλλα λόγια, το ζητούμενο είναι να αναθέσουμε τιμή στις μεταβλητές $f_i \in [0, 1]$ για κάθε αντικείμενο i, υπό τον περιορισμό $\sum_{i=1}^{n} f_i\, s_i \leq B$, ώστε να μεγιστοποιηθεί το $\sum_{i=1}^{n} f_i\, p_i$.

Στην Ενότητα 3.3, είδαμε ότι η ακέραια εκδοχή του Προβλήματος του Σακιδίου όπου $f_i \in \{0, 1\}$, αποτελεί **NP**-δύσκολο πρόβλημα. ΄Ετσι αποτελεί κοινή πεποίθηση ότι δεν υπάρχει αλγόριθμος πολυωνυμικού χρόνου για την επίλυσή του. Αντίθετα, η κλασματική εκδοχή του προβλήματος επιλύεται σε χρόνο $\Theta(n \log n)$ από έναν άπληστο αλγόριθμο.

΄Οπως και η ακέραια εκδοχή (βλ. Ενότητα 3.3), το Κλασματικό Πρόβλημα του Σακιδίου έχει την ιδιότητα των βέλτιστων επιμέρους λύσεων. Συγκεκριμένα, έστω μια βέλτιστη λύση $F^* = (f_1^*, \ldots, f_n^*)$. Αν αγνοήσουμε ένα οποιοδήποτε αντικείμενο i, η λύση $F_{-i}^* = (f_1^*, \ldots, f_{i-1}^*, f_{i+1}^*, \ldots, f_n^*)$ αποτελεί μια βέλτιστη λύση για σακίδιο μεγέθους $B - f_i\, s_i$ και τα $n - 1$ αντικείμενα που απομένουν[2].

Θεωρούμε σαν κριτήριο για την ταξινόμηση των αντικειμένων το λόγο αξίας προς μέγεθος (με απλά λόγια, πόσο κέρδος δίνει κάθε αντικείμενο ανά μονάδα μεγέθους). Για κάθε αντικείμενο i, θέτουμε $r_i = p_i/s_i$. Χωρίς βλάβη της γενικότητας, θεωρούμε ότι τα αντικείμενα είναι ταξινομημένα σε φθίνουσα σειρά του λόγου αξίας προς μέγεθος, δηλαδή είναι $r_1 \geq r_2 \geq \cdots \geq r_n$ (αυτό μπορεί να επιτευχθεί εφαρμόζοντας έναν αλγόριθμο ταξινόμησης σε $\Theta(n \log n)$ χρόνο). Ως κριτήριο επιλογής, ο άπληστος αλγόριθμος συμπεριλαμβάνει όσο μεγαλύτερο ποσοστό του αντικειμένου i χωράει στο σακίδιο. Ο αλγόριθμος μειώνει αντίστοιχα το διαθέσιμο μέγεθος του σακιδίου και συνεχίζει (εφαρμόζοντας τον εαυτό του) με τα υπόλοιπα αντικείμενα.

```
greedyKnapsack(B, (s_1, p_1), ..., (s_n, p_n))
    for i ← 1 to n do
        r_i ← p_i/s_i; F[i] ← 0.
    Ταξινόμηση σε φθίνουσα σειρά των r_i, δηλ. r_1 ≥ r_2 ≥ ··· ≥ r_n.
    for i ← 1 to n do
        if B ≤ 0 then F[i] ← 0;
        else if B ≥ s_i then
                 F[i] ← 1; B ← B − s_i;
             else
                 F[i] ← B/s_i; B ← B − F[i] s_i (= 0);
    return(F);
```

[2]Αν κάτι τέτοιο δεν ίσχυε και υπήρχε καλύτερη λύση για το συγκεκριμένο υπο-στιγμιότυπο, θα τη χρησιμοποιούσαμε για να βελτιώσουμε την αξία της F^*. Αυτό είναι άτοπο αφού υποθέσαμε ότι η F^* είναι μία βέλτιστη λύση.

Ο αλγόριθμος χρειάζεται $\Theta(n \log n)$ χρόνο για την ταξινόμηση των αντικειμένων και $\Theta(n)$ χρόνο για την επιλογή των αντικειμένων που θα συμπεριληφθούν στο σακίδιο. Ο συνολικός χρόνος εκτέλεσης του αλγορίθμου είναι $\Theta(n \log n)$.

Ουσιαστικά ο άπληστος αλγόριθμος υπολογίζει το μικρότερο δείκτη k για τον οποίο $\sum_{i=1}^{k} s_i > B$. Όλα τα αντικείμενα με μικρότερο δείκτη $i = 1, \ldots, k-1$ (και μεγαλύτερο λόγο αξίας προς μέγεθος) συμπεριλαμβάνονται ολόκληρα στο σακίδιο (δηλ. $f_i = 1$), όλα τα αντικείμενα με μεγαλύτερο δείκτη $i = k+1, \ldots, n$ (και μικρότερο λόγο αξίας προς μέγεθος) δεν συμπεριλαμβάνονται στο σακίδιο (δηλ. $f_i = 0$), ενώ το αντικείμενο με δείκτη k συμπεριλαμβάνεται σε ποσοστό

$$f_k = \left(B - \sum_{i=1}^{k-1} s_i \right) / s_k$$

ώστε να μην μείνει ελεύθερος χώρος στο σακίδιο.

Για να αποδείξουμε ότι ο άπληστος αλγόριθμος υπολογίζει μία βέλτιστη λύση, πρέπει να δείξουμε ότι το Κλασματικό Πρόβλημα του Σακιδίου έχει την ιδιότητα της άπληστης επιλογής.

Έστω ότι ο άπληστος αλγόριθμος συμπεριλαμβάνει στο σακίδιο το αντικείμενο 1 σε ποσοστό f_1, $0 < f_1 \leq 1$. Επειδή ο άπληστος αλγόριθμος υπολογίζει πάντα μια αποδεκτή λύση, $B \geq f_1 \, s_1$. Αφού ο άπληστος αλγόριθμος συμπεριλαμβάνει στο σακίδιο (που αρχικά είναι άδειο) όσο μεγαλύτερο ποσοστό από το αντικείμενο 1 μπορεί, δεν υπάρχει αποδεκτή λύση που να συμπεριλαμβάνει στο σακίδιο μεγαλύτερο ποσοστό από το αντικείμενο 1. Έστω μια βέλτιστη λύση F^* που περιλαμβάνει στο σακίδιο μικρότερο ποσοστό $f_1^* < f_1$ από το αντικείμενο 1. Σε αυτή τη λύση, μπορούμε να αντικαταστήσουμε $(f_1 - f_1^*) \, s_1$ μονάδες κάποιων άλλων αντικειμένων (ή ελεύθερου χώρου) με $(f_1 - f_1^*) \, s_1$ μονάδες του αντικειμένου 1. Αυτό είναι πάντα εφικτό και οδηγεί σε αποδεκτή λύση γιατί $B \geq f_1 \, s_1$. Επιπλέον, η λύση που προκύπτει συγκεντρώνει συνολική αξία τουλάχιστον όση η F^* επειδή το αντικείμενο 1 έχει το μεγαλύτερο λόγο αξίας προς μέγεθος. Δηλαδή, υπάρχει μια βέλτιστη λύση που συμφωνεί με την άπληστη λύση στο ποσοστό από το αντικείμενο 1 που συμπεριλαμβάνεται στο σακίδιο. Επομένως, η κλασματική εκδοχή του Προβλήματος του Σακιδίου έχει την ιδιότητα της άπληστης επιλογής.

Θεώρημα 4.3. *Ο αλγόριθμος greedyKnapsack υπολογίζει μια βέλτιστη λύση για το Κλασματικό Πρόβλημα του Σακιδίου.*

Απόδειξη. Εφαρμόζουμε επαγωγικά τις ιδιότητες της άπληστης επιλογής και των βέλτιστων επιμέρους λύσεων. Ο άπληστος αλγόριθμος δεν αφήνει ελεύθερο χώρο στο σακίδιο. Επομένως για κάθε $B \geq 0$, ο αλγόριθμος υπολογίζει τη βέλτιστη λύση όταν έχουμε μόνο ένα αντικείμενο. Υποθέτουμε επαγωγικά ότι για κάθε $B \geq 0$, ο αλγόριθμος υπολογίζει τη βέλτιστη λύση για $n-1$ αντικείμενα.

Έστω ένα στιγμιότυπο με n αντικείμενα και μέγεθος σακιδίου B. Έστω $F = (f_1, \ldots, f_n)$ η άπληστη λύση που υπολογίζει ο αλγόριθμος, και έστω $F^* = (f_1^*, \ldots, f_n^*)$ μια βέλτιστη λύση με $f_1^* = f_1$. Η ιδιότητα της άπληστης επιλογής εγγυάται την ύπαρξη μιας τέτοιας βέλτιστης λύσης.

Από την επαγωγική υπόθεση, η λύση $F_{-1} = (f_2, \ldots, f_n)$ είναι μια βέλτιστη λύση για σακίδιο μεγέθους $B - f_1 \, s_1$ και τα αντικείμενα $2, \ldots, n$. Από την ιδιότητα των βέλτιστων επιμέρους λύσεων, η λύση $F_{-1}^* = (f_2^*, \ldots, f_n^*)$ είναι μια βέλτιστη λύση για σακίδιο μεγέθους $B - f_1^* \, s_1 = B - f_1 \, s_1$

και τα αντικείμενα $2, \ldots, n$. Δηλαδή, οι F_{-1} και F^*_{-1} αποτελούν βέλτιστες λύσεις για το ίδιο υπο-στιγμιότυπο. Θα έχουν λοιπόν την συνολική ίδια αξία:

$$\sum_{i=2}^{n} f_i \, p_i = \sum_{i=2}^{n} f_i^* \, p_i$$

Χρησιμοποιώντας την παραπάνω ισότητα και το γεγονός ότι $f_1 = f_1^*$, καταλήγουμε στην ισότητα:

$$f_1 \, p_1 + \sum_{i=2}^{n} f_i \, p_i = f_1^* \, p_1 + \sum_{i=2}^{n} f_i^* \, p_i$$

Δηλαδή και οι λύσεις F και F^* για το αρχικό στιγμιότυπο έχουν την ίδια συνολική αξία. Επομένως, η άπληστη λύση F είναι μία βέλτιστη λύση. □

Άσκηση 4.7. Δώστε ένα παράδειγμα στιγμιότυπου για το Ακέραιο Πρόβλημα του Σακιδίου όπου ο άπληστος αλγόριθμος δεν υπολογίζει τη βέλτιστη λύση. Χρησιμοποιώντας το παράδειγμα, εξηγείστε την αποτυχία του άπληστου αλγόριθμου.

Λύση. Έστω σακίδιο μεγέθους B, όπου B αρκετά μεγαλύτερο από το 1 (π.χ. $B = 100$). Έστω επίσης δύο αντικείμενα $(1, 1+\varepsilon)$ και (B, B). Οι λόγοι αξίας προς μέγεθος είναι $r_1 = 1 + \varepsilon$ και $r_2 = 1$. Για κάθε $\varepsilon > 0$, μόνο το πρώτο αντικείμενο επιλέγεται στο σακίδιο από τον άπληστο αλγόριθμο. Η συνολική αξία της άπληστης λύσης είναι $1+\varepsilon$. Η βέλτιστη λύση επιλέγει το δεύτερο αντικείμενο και συγκεντρώνει συνολική B. Η απόκλιση της αξίας της βέλτιστης λύσης μπορεί να γίνει οσοδήποτε μεγάλη αυξάνοντας το μέγεθος του σακιδίου.

Ο άπληστος αλγόριθμος αποτυγχάνει γιατί η ακέραιη παραλλαγή του προβλήματος δεν έχει την ιδιότητα της άπληστης επιλογής. Πράγματι, στο συγκεκριμένο παράδειγμα δεν υπάρχει βέλτιστη λύση που να συμφωνεί με την άπληστη επιλογή του πρώτου αντικειμένου. Στην ακέραιη εκδοχή, υπάρχουν επιλογές αντικειμένων που δεν επιτρέπουν στο σακίδιο να γεμίσει πλήρως. Έτσι ο λόγος της αξίας προς το μέγεθος κάθε αντικειμένου δεν είναι πλέον ασφαλές κριτήριο, αφού πρέπει να λάβουμε υπόψη και το χώρο που ενδεχομένως να μείνει ελεύθερος αν συμπεριλάβουμε κάποιο συγκεκριμένο αντικείμενο στο σακίδιο. □

4.3.1 Απληστία και Δυναμικός Προγραμματισμός

Είδαμε λοιπόν ότι το Ακέραιο Πρόβλημα του Σακιδίου δεν έχει την ιδιότητα της άπληστης επιλογής και ο άπληστος αλγόριθμος δεν εγγυάται τον υπολογισμό μιας βέλτιστης λύσης. Για την ακέραια εκδοχή, η απόφαση αν ένα αντικείμενο i θα συμπεριληφθεί στο σακίδιο πρέπει να βασιστεί στη σύγκριση των βέλτιστων αξιών των δύο υπο-στιγμιοτύπων που προκύπτουν από τη συμπερίληψη ή όχι του i στο σακίδιο. Αυτή η θεώρηση οδηγεί στον αλγόριθμο δυναμικού προγραμματισμού της Ενότητας 3.3.

Γενικότερα, οι επιλογές ενός αλγόριθμου δυναμικού προγραμματισμού βασίζονται στις τιμές των βέλτιστων λύσεων κατάλληλα επιλεγμένων υπο-στιγμιοτύπων. Ένας αλγόριθμος δυναμικού προγραμματισμού πρώτα επιλύει όλα τα διαφορετικά υπο-στιγμιότυπα που έχουν μια συγκεκριμένη μορφή, και στη συνέχεια συνθέτει τη βέλτιστη λύση του αρχικού στιγμιότυπου από τις βέλτιστες λύσεις των "καλύτερων" υπο-στιγμιοτύπων.

Οι άπληστοι αλγόριθμοι είναι πιο αποδοτικοί από τους αλγόριθμους δυναμικού προγραμματισμού επειδή εκμεταλλεύονται την ιδιότητα της άπληστης επιλογής και επιλύουν ένα μικρό αριθμό υπο-στιγμιοτύπων. Ένα πρόβλημα έχει την ιδιότητα της άπληστης επιλογής όταν μια ακολουθία τοπικά βέλτιστων επιλογών οδηγεί σε μία συνολικά βέλτιστη λύση. Ένας άπληστος αλγόριθμος επιλέγει αυτό που φαίνεται καλύτερο με βάση την τρέχουσα κατάσταση και κάποιο συγκεκριμένο κριτήριο και επιλύει μόνο το υπο-στιγμιότυπο που προκύπτει από αυτή την επιλογή. Η κάθε επιλογή ενός άπληστου αλγορίθμου εξαρτάται μόνο από τις προηγούμενες επιλογές του και όχι από τις (βέλτιστες) λύσεις των υπο-στιγμιοτύπων που προκύπτουν. Σε αντίθεση λοιπόν με το δυναμικό προγραμματισμό που ακολουθεί bottom-up στρατηγική, ένας άπληστος αλγόριθμος ακολουθεί την top-down στρατηγική της εξειδίκευσης: Σε κάθε βήμα κάνει μια τοπικά βέλτιστη επιλογή. Με βάση αυτή την επιλογή, το πρόβλημα περιορίζεται σε ένα υπο-στιγμιότυπο, το οποίο επιλύεται αναδρομικά.

Από την άλλη πλευρά, η επιτυχία τόσο του δυναμικού προγραμματισμού όσο και της μεθόδου της απληστίας εξαρτάται από την ιδιότητα των βέλτιστων επιμέρους λύσεων. Η ιδιότητα αυτή επιτρέπει την αναδρομική / επαγωγική επίλυση ενός προβλήματος επειδή εγγυάται ότι η βέλτιστη λύση του αρχικού στιγμιότυπου μπορεί να συνθεθεί από τις βέλτιστες λύσεις κατάλληλα επιλεγμένων υπο-στιγμιοτύπων.

4.4 Βιβλιογραφικές Αναφορές

Η μέθοδος της απληστίας και οι εφαρμογές της στην επίλυση προβλημάτων συνδυαστικής βελτιστοποίησης παρουσιάζεται πολύ καλά στα [11], [32], και [7]. Η περιγραφή της βασικής δομής της λειτουργίας των άπληστων αλγόριθμων οφείλεται στους Borodin, Nielsen, και Rackoff [6].

5 Αλγόριθμοι Γραφημάτων

Διαισθητικά, γράφημα είναι οτιδήποτε μπορεί να αναπαρασταθεί με σημεία (κορυφές) και γραμμές (ακμές) μεταξύ των σημείων. Τα γραφήματα επιτρέπουν την κομψή μοντελοποίηση πολλών και σημαντικών υπολογιστικών προβλημάτων, γι' αυτό οι αλγόριθμοι γραφημάτων αποτελούν μια πολύ σημαντική κατηγορία αλγορίθμων στην Επιστήμη των Υπολογιστών. Αξίζει να αναφερθούμε σε μερικά παραδείγματα, όπου η μοντελοποίηση των προβλημάτων γίνεται με τη βοήθεια γραφημάτων.

Ένα δίκτυο υπολογιστών μπορεί να αναπαρασταθεί από ένα γράφημα, όπου οι κορυφές αναπαριστούν τις ενεργές συσκευές (υπολογιστές, δρομολογητές, κλπ.) και οι ακμές τις δικτυακές συνδέσεις. Το θεμελιώδες πρόβλημα της αξιοπιστίας ενός δικτύου ανάγεται στον έλεγχο της συνεκτικότητας ενός τέτοιου γραφήματος όταν κάποιες ακμές τεθούν εκτός λειτουργίας.

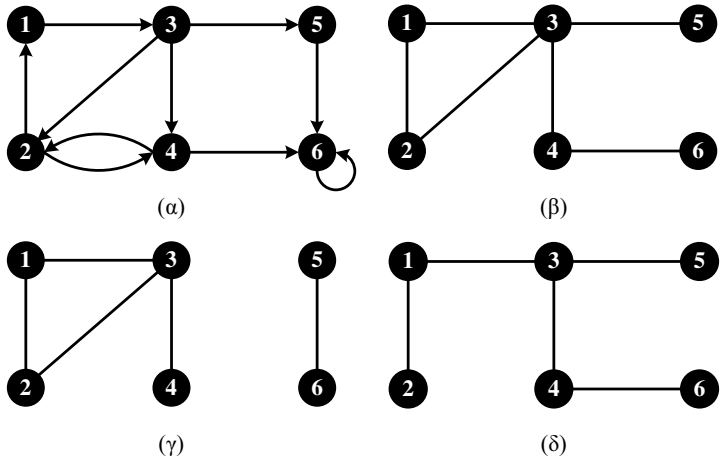
Σε προβλήματα δρομολόγησης εργασιών που χρησιμοποιούν κοινόχρηστους πόρους, οι κορυφές αναπαριστούν τις διαδικασίες και οι ακμές την απαίτηση δύο διεργασιών για τον ίδιο πόρο. Το θεμελιώδες πρόβλημα του χρωματισμού γραφημάτων χωρίζει τις διεργασίες σε ομάδες χωρίς ανταγωνιστικές απαιτήσεις, δηλαδή σε σύνολα κορυφών χωρίς ακμές μεταξύ κορυφών στο ίδιο σύνολο. Σήμερα το πρόβλημα του χρωματισμού γραφημάτων βρίσκει πολλές εφαρμογές σε ασύρματα δίκτυα, όπου οι χρήστες ανταγωνίζονται για τη χρήση ενός περιορισμένου αριθμού καναλιών επικοινωνίας.

Σε προβλήματα σχεδιασμού ολοκληρωμένων κυκλωμάτων, οι κορυφές αναπαριστούν μικρότερα κυκλώματα που συνδέονται μεταξύ τους με ακμές. Ένα θεμελιώδες πρόβλημα είναι η διάταξη (layout) των κορυφών σε μία συγκεκριμένη περιοχή (π.χ. ευθεία γραμμή, τετραγωνικές κυψέλες - grid, επίπεδο) ώστε να ελαχιστοποιηθούν οι διασταυρώσεις των ακμών. Ο αριθμός των διασταυρώσεων των ακμών καθορίζει τον αριθμό των επιπέδων που χρειάζονται για να τυπωθεί το κύκλωμα.

Σε προβλήματα υπολογισμού διαδρομών, οι κορυφές αντιστοιχούν σε περιοχές και τα μήκη των ακμών δηλώνουν τις αντίστοιχες αποστάσεις. Έχουμε ήδη αναφερθεί στο Πρόβλημα του Περιοδεύοντος Πωλητή, που αφορά στον υπολογισμό μιας περιοδείας ελάχιστου συνολικού μήκους. Άλλα θεμελιώδη προβλήματα είναι να υπολογισθεί το Συντομότερο Μονοπάτι μεταξύ δύο κορυφών, και να υπολογισθεί ένα Ελάχιστο Συνδετικό Δέντρο (Minimum Spanning Tree), δηλαδή ένα σύνολο ακμών που συνδέει όλες τις κορυφές και έχει ελάχιστο συνολικό μήκος.

5.1 Βασικοί Ορισμοί και Αναπαράσταση Γραφημάτων

Ένα γράφημα $G \equiv (V, E)$, ή απλούστερα $G(V, E)$, ορίζεται από το σύνολο των κορυφών V και το σύνολο των ακμών E. Τυπικά, ένα *γράφημα* (graph) G είναι ένα διατεταγμένο ζεύγος



Σχήμα 5.1: Παραδείγματα κατευθυνόμενων και μη-κατευθυνόμενων γραφημάτων.

$G(V, E)$, όπου $V = \{v_1, \ldots v_n\}$ είναι το σύνολο των κορυφών και $E = \{e_1, \ldots, e_m\}$ είναι το σύνολο των ακμών. Στα *μη-κατευθυνόμενα* γραφήματα (undirected graphs), κάθε ακμή είναι ένα διμελές σύνολο κορυφών, $e = \{v_1, v_2\}$, όχι απαραίτητα διαφορετικών μεταξύ τους. Στα *κατευθυνόμενα* γραφήματα (directed graphs), κάθε ακμή είναι ένα διατεταγμένο ζεύγος κορυφών, $e = (v_1, v_2)$. Μερικές φορές, χρησιμοποιούμε καταχρηστικά το συμβολισμό (v_1, v_2) και για ακμές μη-κατευθυνόμενων γραφημάτων. Τα μεγέθη που χαρακτηρίζουν ένα γράφημα $G(V, E)$ είναι ο αριθμός των κορυφών του, συνήθως συμβολίζεται με n ή $|V|$, και ο αριθμός των ακμών του, συνήθως συμβολίζεται με m ή $|E|$.

Η (μη-κατευθυνόμενη) ακμή $e = \{v_1, v_2\}$ συνδέει τις κορυφές v_1 και v_2, οι οποίες ονομάζονται και άκρα της ακμής e. Η κατευθυνόμενη ακμή $e = (v_1, v_2)$ συνδέει την κορυφή v_1 με την v_2. Η v_1 ονομάζεται ουρά (ή αρχή) της ακμής e και η v_2 ονομάζεται κεφαλή (ή τέλος) της e. Δύο κορυφές που συνδέονται με ακμή ονομάζονται *γειτονικές*. Συμβολίζουμε με $\Gamma(u)$ το σύνολο των γειτονικών κορυφών μιας δεδομένης κορυφής $u \in V$. Τυπικά, $\Gamma(u) = \{v \in V : (u, v) \in E\}$.

Μία ακμή που τα δύο άκρα της ταυτίζονται (ή η αρχή της ταυτίζεται με το τέλος της αν είναι κατευθυνόμενη) ονομάζεται ανακύκλωση (loop). Δύο ακμές με κοινά άκρα (ή κοινή αρχή και τέλος αν είναι κατευθυνόμενες) ονομάζονται *παράλληλες*. Ένα γράφημα ονομάζεται *απλό* όταν δεν έχει παράλληλες ακμές και ανακυκλώσεις. Θα θεωρούμε πάντα απλά γραφήματα (εκτός αν σαφώς δηλώνεται κάτι διαφορετικό). Θα αναφερθούμε μόνο σε *πεπερασμένα* γραφήματα που ορίζονται σε πεπερασμένα σύνολα κορυφών.

Παράδειγμα 5.1. Το γράφημα (α) του Σχήματος 5.1 είναι κατευθυνόμενο, έχει μια ανακύκλωση (ακμή $(6, 6)$), αλλά όχι παράλληλες ακμές (οι ακμές $(2, 4)$ και $(4, 2)$ δεν είναι παράλληλες αφού είναι αντίρροπες). Τα γράφηματα (β), (γ), και (δ) του Σχήματος 5.1 είναι απλά μη-κατευθυνόμενα γραφήματα.

Στις ακμές ενός γραφήματος $G(V, E)$ μπορούμε να αντιστοιχίσουμε βάρη μέσω μίας συνάρτησης $w : E \mapsto \mathbb{R}$. Τότε το γράφημα ονομάζεται *βεβαρυμένο* ή γράφημα με *βάρη στις ακμές* (weighted graph) και συμβολίζεται $G(V, E, w)$. Σε μερικές περιπτώσεις το βάρος μιας ακμής θεωρείται σαν το μήκος ή το κόστος της.

Σε ένα μη-κατευθυνόμενο γράφημα, ο *βαθμός* (degree) μιας κορυφής v, που συμβολίζεται με $\deg(v)$, είναι ο αριθμός των ακμών που εφάπτονται στη v. Σε ένα κατευθυνόμενο γράφημα, διακρίνουμε τον *εισερχόμενο βαθμό* (in-degree) της v, που συμβολίζεται με $\deg_{\text{in}}(v)$ και είναι ο αριθμός των ακμών που καταλήγουν στη v, και τον *εξερχόμενο βαθμό* (out-degree) της v, που συμβολίζεται με $\deg_{\text{out}}(v)$ και είναι ο αριθμός των ακμών που ξεκινούν από τη v. Για παράδειγμα, στο γράφημα (β) του Σχήματος 5.1, η κορυφή 2 έχει βαθμό 2, η κορυφή 3 έχει βαθμό 4, και η κορυφή 5 έχει βαθμό 1. Στο γράφημα (α), ο εξερχόμενος βαθμός της κορυφής 3 είναι 3 και ο εισερχόμενος βαθμός της ίδιας κορυφής είναι 1.

Ο *ελάχιστος βαθμός* $\delta(G)$ ενός γραφήματος $G(V, E)$ είναι ο μικρότερος βαθμός κάποιας κορυφής του, $\delta(G) \equiv \min_{v \in V}\{\deg(v)\}$. Ο *μέγιστος βαθμός* $\Delta(G)$ ενός γραφήματος $G(V, E)$ είναι ο μεγαλύτερος βαθμός κάποιας κορυφής του, $\Delta(G) \equiv \max_{v \in V}\{\deg(v)\}$.

Άσκηση 5.1. Να αποδείξετε ότι σε κάθε μη-κατευθυνόμενο γράφημα $G(V, E)$, $\sum_{v \in V} \deg(v) = 2\,|E|$, και σε κάθε κατευθυνόμενο γράφημα $G(V, E)$, $\sum_{v \in V} \deg_{\text{in}}(v) = \sum_{v \in V} \deg_{\text{out}}(v) = |E|$.

Λύση. Για τα μη-κατευθυνόμενα γραφήματα, κάθε ακμή συνεισφέρει 1 στο βαθμό των δύο άκρων της. Για τα κατευθυνόμενα γραφήματα, κάθε ακμή συνεισφέρει 1 στον εισερχόμενο βαθμό του τέλους της και 1 στο εξερχόμενο βαθμό της αρχής της. □

Μια ακολουθία κορυφών $(v_0, v_1, \ldots, v_k)$, όπου η ακμή $(v_{i-1}, v_i) \in E$ για κάθε $i = 1, \ldots, k$, αποτελεί μια *διαδρομή* (walk) μήκους k. Διαισθητικά, διαδρομή είναι μια ακολουθία κορυφών που συνδέονται με ''συνεχόμενες'' ακμές. Το μήκος της διαδρομής είναι ίσο με τον αριθμό των ακμών της. Μία διαδρομή ονομάζεται *μονοκονδυλιά* (trail) όταν όλες οι ακμές της είναι διαφορετικές και ονομάζεται *μονοπάτι* (path) όταν όλες οι κορυφές της είναι διαφορετικές. Μερικές φορές, χρησιμοποιείται και ο όρος απλό μονοπάτι (simple path) για μια διαδρομή με διαφορετικές κορυφές (και άρα ακμές). Θα λέμε ότι μια κορυφή u είναι *προσπελάσιμη* από μια κορυφή v αν υπάρχει μονοπάτι από τη v στη u.

Όταν η αρχική και η τελική κορυφή μιας διαδρομής συμπίπτουν, έχουμε μία *κλειστή διαδρομή*. Μια κλειστή διαδρομή ονομάζεται *κύκλωμα* (circuit) όταν όλες οι ακμές της είναι διαφορετικές, και ονομάζεται *κύκλος* (cycle) όταν όλες οι κορυφές της είναι διαφορετικές. Με άλλα λόγια, το κύκλωμα είναι μία κλειστή μονοκονδυλιά, και ο κύκλος είναι ένα κλειστό μονοπάτι. Μερικές φορές, χρησιμοποιείται και ο όρος απλός κύκλος (simple cycle) για μια κλειστή διαδρομή με διαφορετικές κορυφές.

Η *απόσταση* $d(u, v)$ μεταξύ δύο κορυφών u, v είναι το μήκος του συντομότερου μονοπατιού μεταξύ τους. Η *διάμετρος* $D(G)$ ενός γραφήματος $G(V, E)$ είναι η μέγιστη απόσταση μεταξύ δύο κορυφών στο G, $D(G) \equiv \max_{u,v \in V}\{d(u, v)\}$.

Άσκηση 5.2. Να αποδείξετε ότι σε κάθε γράφημα, υπάρχει διαδρομή από μια κορυφή u σε μια κορυφή w αν και μόνο αν υπάρχει μονοπάτι από τη u στη w.

Λύση. Η μία κατεύθυνση είναι προφανής, γιατί κάθε μονοπάτι είναι διαδρομή εξ' ορισμού. Για την αντίστροφη κατεύθυνση, παρατηρούμε ότι αν μια διαδρομή μεταξύ u και w δεν αντιστοιχεί σε μονοπάτι, τότε πρέπει να περιέχει κορυφές που επαναλαμβάνονται. Όμως, το τμήμα της διαδρομής ανάμεσα σε δύο διαφορετικές εμφανίσεις της ίδιας κορυφής είναι ένα κύκλωμα. Αφαιρώντας όλα τα κυκλώματα, καταλήγω σε ένα μονοπάτι από τη u στη w. Ο παραπάνω συλλογισμός

οδηγεί ακόμη στο συμπέρασμα ότι κάθε συντομότερη διαδρομή από τη u στη w είναι μονοπάτι, αφού η αφαίρεση ενός κυκλώματος μικραίνει το μήκος της διαδρομής. Αυτός είναι ο λόγος που αναφερόμαστε πάντα σε συντομότερα μονοπάτια και εξαιρετικά σπάνια σε συντομότερες διαδρομές.

Με παρόμοιο τρόπο μπορείτε να αποδείξετε ότι ένα γράφημα περιέχει μία κλειστή διαδρομή (ή ένα κύκλωμα) αν και μόνο αν περιέχει έναν κύκλο. □

Ένα γράφημα $G'(V', E')$ αποτελεί *υπογράφημα* (subgraph) του $G(V, E)$ όταν $V' \subseteq V$ και $E' \subseteq E$. Το υπογράφημα $G'(V', E')$ ονομάζεται συνδετικό ή επικαλύπτον (spanning) όταν $V' = V$ (δηλαδή, περιέχει / καλύπτει όλες τις κορυφές του αρχικού γραφήματος), και *επαγόμενο* (induced) με σύνολο κορυφών V' όταν $E' = \{(u, v) \in E : u, v \in V'\}$ (δηλαδή περιέχει όλες τις ακμές του αρχικού γραφήματος μεταξύ των κορυφών του V').

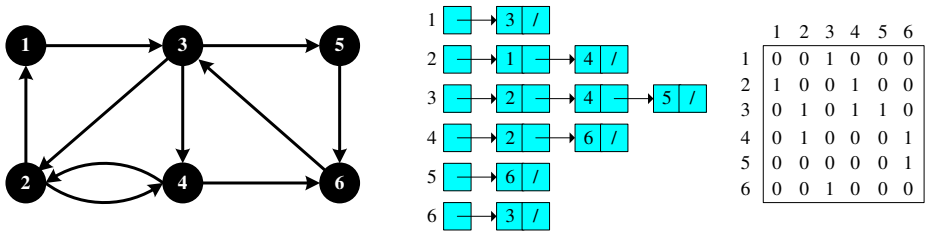
Ένα μη-κατευθυνόμενο γράφημα είναι *συνεκτικό* (connected) όταν υπάρχει μονοπάτι ανάμεσα σε κάθε ζευγάρι κορυφών. Δηλαδή, σε ένα συνεκτικό γράφημα μπορούμε να "μεταβούμε" από οποιαδήποτε κορυφή σε οποιαδήποτε άλλη ακολουθώντας τις ακμές του γραφήματος.

Όταν ένα γράφημα $G(V, E)$ δεν είναι συνεκτικό, διαμερίζεται σε *συνεκτικές συνιστώσες* (connected components). Μια συνεκτική συνιστώσα είναι ένα μεγιστικό συνεκτικό υπογράφημα. Οι συνεκτικές συνιστώσες ενός γραφήματος είναι ξένες μεταξύ τους και η ένωσή τους δίνει το γράφημα. Για παράδειγμα, τα γραφήματα (β) και (δ) του Σχήματος 5.1 είναι συνεκτικά, ενώ το γράφημα (γ) δεν είναι συνεκτικό και έχει δύο συνεκτικές συνιστώσες, τα επαγόμενα υπογραφήματα με σύνολα κορυφών $\{1, 2, 3, 4\}$ και $\{5, 6\}$. Οι συνεκτικές συνιστώσες ενός γραφήματος αντιστοιχούν στις κλάσεις ισοδυναμίας στις οποίες διαμερίζεται το σύνολο των κορυφών από τη σχέση ισοδυναμίας "υπάρχει μονοπάτι μεταξύ των κορυφών u και v".

Ένα κατευθυνόμενο γράφημα είναι *συνεκτικό* όταν για κάθε ζευγάρι κορυφών του $u, v \in V$, υπάρχει μονοπάτι (που σέβεται τις κατευθύνσεις των ακμών) είτε από τη u στη v είτε από τη v στη u. Ένα κατευθυνόμενο γράφημα είναι *ισχυρά συνεκτικό* (strongly connected) όταν για κάθε ζευγάρι κορυφών του $u, v \in V$, υπάρχουν μονοπάτια (που σέβονται τις κατευθύνσεις των ακμών) και από τη u στη v και από τη v στη u.

Όταν ένα κατευθυνόμενο γράφημα $G(V, E)$ δεν είναι ισχυρά συνεκτικό, διαμερίζεται σε *ισχυρά συνεκτικές συνιστώσες* (strongly connected components). Μια ισχυρά συνεκτική συνιστώσα είναι ένα μεγιστικό ισχυρά συνεκτικό υπογράφημα. Για παράδειγμα, το γράφημα (α) του Σχήματος 5.1 είναι συνεκτικό αλλά όχι ισχυρά συνεκτικό και αποτελείται από τρεις ισχυρά συνεκτικές συνιστώσες, τα επαγόμενα υπογραφήματα με σύνολα κορυφών $\{1, 2, 3, 4\}$ και $\{5\}$ και $\{6\}$. Οι ισχυρά συνεκτικές συνιστώσες ενός γραφήματος αντιστοιχούν στις κλάσεις ισοδυναμίας στις οποίες διαμερίζεται το σύνολο των κορυφών από τη σχέση ισοδυναμίας "υπάρχει μονοπάτι από την κορυφή u στη v και από τη v στην u".

Ανάλογα με τις ιδιότητές τους, μπορούμε να διακρίνουμε διάφορα είδη γραφημάτων. Μία από τις σημαντικότερες κατηγορίες γραφημάτων είναι τα δέντρα. Ένα ακυκλικό (acyclic) γράφημα (δηλ. ένα γράφημα που δεν περιέχει κύκλους) ονομάζεται *δάσος* (forest). Ένα ακυκλικό και συνεκτικό γράφημα ονομάζεται *δέντρο* (tree). Για παράδειγμα, το γράφημα (δ) του Σχήματος 5.1 είναι δέντρο. Οι κορυφές ενός δέντρου με βαθμό 1 ονομάζονται φύλλα. Οι σημαντικότερες ιδιότητες των δέντρων συνοψίζονται στο Θεώρημα 5.1. Η απόδειξη του θεωρήματος υπάρχει στα περισσότερα βιβλία Θεωρίας Γραφημάτων (π.χ. [21, 24, 12]).



Σχήμα 5.2: Παράδειγμα αναπαράστασης κατευθυνόμενου γραφήματος.

Θεώρημα 5.1. *Τα παρακάτω είναι ισοδύναμα για κάθε απλό μη-κατευθυνόμενο γράφημα* $G(V, E)$*:*

1. *Το γράφημα* G *είναι δέντρο.*
2. *Κάθε ζευγάρι κορυφών του* G *ενώνεται με μοναδικό μονοπάτι.*
3. *Το* G *είναι* ελαχιστικά συνεκτικό*, δηλ. αν αφαιρεθεί μια οποιαδήποτε ακμή, το γράφημα παύει να είναι συνεκτικό.*
4. *Το* G *είναι συνεκτικό και* $|E| = |V| - 1$*.*
5. *Το* G *είναι ακυκλικό και* $|E| = |V| - 1$*.*
6. *Το* G *είναι* μεγιστικά ακυκλικό*, δηλ. αν προστεθεί μια οποιαδήποτε ακμή, το γράφημα αποκτά κύκλο.*

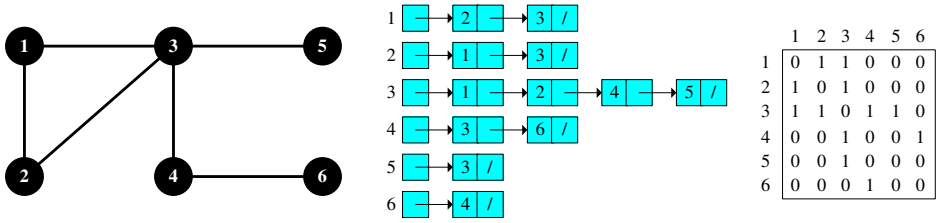
Άσκηση 5.3. Να αποδείξετε το Θεώρημα 5.1.

5.1.1 Αναπαράσταση Γραφημάτων

Η επεξεργασία ενός γραφήματος απαιτεί την αναπαράσταση του γραφήματος στη μνήμη του υπολογιστή μέσω κατάλληλης δομής δεδομένων. Οι πιο συνηθισμένοι τρόποι αναπαράστασης η *λίστα γειτνίασης* (adjacency list) και ο *πίνακας γειτνίασης* (adjacency matrix). Στο εξής θεωρούμε απλό γράφημα $G(V, E)$ με n κορυφές και m ακμές. Επίσης, θεωρούμε ότι υφίσταται μια αρίθμηση των κορυφών του γραφήματος με τους αριθμούς $1, 2, \ldots, n$.

Η λίστα γειτνίασης του γραφήματος $G(V, E)$ αποτελείται από τη λίστα των γειτονικών κορυφών για κάθε κορυφή $v \in V$. Οι δείκτες στα πρώτα στοιχεία αυτών των λιστών αποθηκεύονται σε έναν πίνακα L με n θέσεις. Συγκεκριμένα, ο δείκτης στο πρώτο στοιχείο της λίστας των γειτονικών κορυφών της $v \in V$ βρίσκεται στη θέση $L[v]$. Το μέγεθος της λίστας των γειτόνων της v είναι ίσο με το βαθμό της. Επομένως, η αναπαράσταση του G με λίστα γειτνίασης απαιτεί $\Theta(n + m)$ θέσεις μνήμης.

Ο πίνακας γειτνίασης του γραφήματος $G(V, E)$, συμβολίζεται με A, έχει $n \times n$ στοιχεία. Το στοιχείο $A[i, j]$ είναι 1 αν η ακμή $(v_i, v_j) \in E$ και 0 διαφορετικά. Αν το γράφημα έχει βάρη στις ακμές, το στοιχείο $A[i, j]$ είναι ίσο με το βάρος $w(v_i, v_j)$ της αντίστοιχης ακμής και έχει μια ειδική τιμή αν η ακμή (v_i, v_j) δεν υπάρχει στο γράφημα. Η αναπαράσταση του γραφήματος G με πίνακα γειτνίασης απαιτεί $\Theta(n^2)$ θέσεις μνήμης.



Σχήμα 5.3: Παράδειγμα αναπαράστασης μη-κατευθυνόμενου γραφήματος.

Ένα παράδειγμα αναπαράστασης κατευθυνόμενο γραφήματος με λίστα γειτνίασης και πίνακα γειτνίασης δίνεται στο Σχήμα 5.2, ενώ στο Σχήμα 5.3 δίνεται ένα παράδειγμα αναπαράστασης μη-κατευθυνόμενου γραφήματος.

Όταν το γράφημα είναι *αραιό* (sparse), δηλαδή έχει αριθμό ακμών $o(n^2)$, η αναπαράσταση με λίστα γειτνίασης απαιτεί ασυμπτωτικά λιγότερες θέσεις μνήμης από την αναπαράσταση με πίνακα γειτνίασης. Αντίθετα, όταν το γράφημα είναι *πυκνό* (dense), δηλαδή έχει αριθμό ακμών $\Theta(n^2)$, οι απαιτήσεις των δύο αναπαραστάσεων σε θέσεις μνήμης ταυτίζονται ως προς την ασυμπτωτική τους συμπεριφορά.

Η αναπαράσταση με πίνακα γειτνίασης πλεονεκτεί γιατί επιτρέπει τον έλεγχο για ύπαρξη ακμής μεταξύ δύο συγκεκριμένων κορυφών σε σταθερό χρόνο. Η αναπαράσταση με λίστα γειτνίασης πλεονεκτεί γιατί επιτρέπει την απαρίθμηση των γειτόνων μιας συγκεκριμένης κορυφής v σε χρόνο $\Theta(\deg(v))$ (αντί για $\Theta(n)$ που είναι ο αντίστοιχος χρόνος για τον πίνακα γειτνίασης).

Στο εξής, θα θεωρούμε ότι το γράφημα εισόδου αναπαρίσταται με λίστα γειτνίασης, εκτός αν ρητά αναφέρεται κάτι διαφορετικό.

Άσκηση 5.4. Να διατυπώσετε αλγόριθμους γραμμικού χρόνου που μετατρέπουν την αναπαράσταση ενός γραφήματος από λίστα γειτνίασης σε πίνακα γειτνίασης και αντίστροφα.

Άσκηση 5.5. Το τετράγωνο ενός γραφήματος $G(V, E)$, συμβολίζεται με $G^2(V, E^2)$, περιέχει την ακμή (u, w) αν και μόνο αν οι κορυφές u και w είναι γειτονικές ή έχουν κοινό γείτονα στο G (ισοδύναμα, αν και μόνο αν οι κορυφές u και w βρίσκονται βρίσκονται σε απόσταση το πολύ 2 στο G). Να διατυπώσετε αποδοτικούς αλγορίθμους για τον υπολογισμό του G^2 όταν το G αναπαρίσταται με λίστα γειτνίασης και με πίνακα γειτνίασης.

Λύση. Όταν το G αναπαρίσταται με λίστα γειτνίασης, ένας αποδοτικός αλγόριθμος είναι ο ακόλουθος: Για κάθε κορυφή u, πρόσθεσε στη λίστα των γειτόνων της u τις λίστες όλων των $v \in V : (v, u) \in E$ (δηλ. τις λίστες όλων των γειτόνων της u). Ειδική μέριμνα χρειάζεται για να μην εμφανίζεται η ίδια κορυφή στη νέα λίστα πολλές φορές. Ο αλγόριθμος αυτός μπορεί να υλοποιηθεί σε χρόνο $O(nm)$.

Στο τετράγωνο A^2 του πίνακα γειτνίασης, το $A^2[v_i, v_j]$ ισούται με τον αριθμό των διαδρομών μήκους 2 μεταξύ των κορυφών v_i και v_j. Η ακμή (v_i, v_j) θα υπάρχει στο G^2 αν και μόνο αν $A[v_i, v_j] + A^2[v_i, v_j] > 0$. Αυτή η παρατήρηση οδηγεί στον ακόλουθο αλγόριθμο: Υπολογίζουμε το τετράγωνο $A^2 = A \times A$ του πίνακα γειτνίασης του G. Έστω ο πίνακας γειτνίασης του G^2. Θέτουμε $B[v_i, v_i] = 0$ για κάθε $v_i \in V$. Για κάθε $v_i, v_j \in V$, θέτουμε $B[v_i, v_j] = 1$ αν

$A[v_i, v_j] + A^2[v_i, v_j] > 0$ και $B[v_i, v_j] = 0$ διαφορετικά. Ο αλγόριθμος μπορεί να υλοποιηθεί σε χρόνο $O(n^{\log_2 7})$ χρησιμοποιώντας τον αλγόριθμο πολλαπλασιασμού πινάκων του Strassen. □

Άσκηση 5.6. Το *ανάστροφο γράφημα* (transpose graph) ενός κατευθυνόμενου γραφήματος $G(V, E)$ είναι το γράφημα $G^T(V, E^T)$ όπου $E^T = \{(v, u) \in V \times V : (u, v) \in E\}$. Δηλαδή, το ανάστροφο γράφημα G^T προκύπτει από το G αντιστρέφοντας τη φορά των ακμών του. Να διατυπώσετε αλγόριθμους γραμμικού χρόνου για τον υπολογισμό του ανάστροφου γραφήματος G^T όταν το G αναπαρίσταται με λίστα γειτνίασης και με πίνακα γειτνίασης.

Λύση. Όταν το G αναπαρίσταται με λίστα γειτνίασης, για κάθε κορυφή $u \in V$ και κάθε κορυφή $v \in L[u]$, προσθέτουμε τη u στη λίστα $L^T[v]$ της κορυφής v για το ανάστροφο γράφημα. Ο ακόλουθος ψευδοκώδικας υπολογίζει τη λίστα γειτνίασης του G^T από τη λίστα γειτνίασης του G. Ο αλγόριθμος έχει χρόνο εκτέλεσης $\Theta(n + m)$, δηλαδή γραμμικό στο μέγεθος της λίστας γειτνίασης.

```
for all v ∈ V do
    L^T[v] ← ∅;
for all u ∈ V do
    for all v ∈ L[u] do
        L^T[v] ← L^T[v] ∪ {u};
```

Όταν το G αναπαρίσταται με πίνακα γειτνίασης, αρκεί να υπολογίσουμε τον ανάστροφο πίνακα A^T του A (ο ανάστροφος πίνακας A^T ενός πίνακα A έχει σαν γραμμές τις στήλες του πίνακα A).

```
for all v ∈ V do
    for all u ∈ V do
        A^T[u, v] ← A[v, u];
```

Ο χρόνος εκτέλεσης είναι $\Theta(n^2)$, δηλαδή γραμμικός στο μέγεθος του A. □

5.2 Αναζήτηση Κατά Πλάτος

Θα ξεκινήσουμε με δύο απλές μεθόδους για την εξερεύνηση ενός γραφήματος, την Αναζήτηση Κατά Πλάτος και την Αναζήτηση Κατά Βάθος.

Δεδομένου ενός γραφήματος $G(V, E)$ και μια διακεκριμένης κορυφής $s \in V$, η Αναζήτηση Κατά Πλάτος (Breadth-First Search) εξερευνά τις κορυφές του γραφήματος σε αύξουσα σειρά απόστασης από την κορυφή s.

Η Αναζήτηση Κατά Πλάτος (ΑΚΠ) λειτουργεί σε φάσεις. Στη φάση k, η ΑΚΠ εξερευνά τις κορυφές που βρίσκονται σε απόσταση k από την s. Συγκεκριμένα, στην πρώτη φάση, η ΑΚΠ εξερευνά τους γειτόνους της s (κορυφές σε απόσταση 1 από την s). Στη δεύτερη φάση, η ΑΚΠ εξερευνά τις κορυφές που δεν εξερεύνησε στην πρώτη φάση και είναι γείτονες κάποιου γείτονα της s (κορυφές σε απόσταση 2 από την s). Γενικότερα, στη φάση k, η ΑΚΠ εξερευνά τις κορυφές που δεν έχουν εξερευνηθεί σε προηγούμενες φάσεις και είναι γείτονες κορυφής που εξερευνήθηκε στη φάση $k - 1$. Ο χαρακτηρισμός "κατά πλάτος" προκύπτει από το γεγονός ότι η

ΑΚΠ ολοκληρώνει την εξερεύνηση των κορυφών σε απόσταση k από την s πριν επεκταθεί στην εξερεύνηση των κορυφών σε απόσταση $k+1$.

Από την περιγραφή της ΑΚΠ, διακρίνουμε τρία είδη κορυφών: Τις *ανεξερεύνητες* κορυφές, τις οποίες δεν έχουμε επισκεφθεί ακόμη, τις *υπο-εξέταση* κορυφές, τις οποίες επισκεφθήκαμε αλλά απομένει η εξερεύνηση των γειτόνων τους, και τις *εξερευνημένες* κορυφές, τις οποίες έχουμε επισκεφθεί και έχουμε ολοκληρώσει την εξερεύνηση των γειτόνων τους. Αρχικά όλες οι κορυφές είναι ανεξερεύνητες. Όταν η ΑΚΠ επισκέπτεται μία κορυφή για πρώτη φορά, η κορυφή γίνεται υπο-εξέταση. Η υπο-εξέταση κορυφή θεωρείται εξερευνημένη στην επόμενη φάση, όταν έχουμε επισκεφθεί όλους τους γειτόνους της.

Για να εξασφαλιστεί η κατά πλάτος εξερεύνηση των κορυφών, η σειρά με την οποία οι υπο-εξέταση κορυφές γίνονται εξερευνημένες πρέπει να είναι ίδια με τη σειρά που οι ανεξερεύνητες κορυφές γίνονται υπο-εξέταση. Γι' αυτό το σκοπό, ο αλγόριθμος διατηρεί μια First-In-First-Out (FIFO) ουρά από υπο-εξέταση κορυφές. Αρχικά, η μόνη υπο-εξέταση κορυφή (και άρα η μόνη κορυφή στην ουρά) είναι η s. Η ΑΚΠ εξάγει την πρώτη κορυφή της ουράς, εξερευνά τις γειτονικές κορυφές (με τη σειρά που παρατίθενται στη λίστα γειτνίασης), και θεωρεί την κορυφή εξερευνημένη. Σαν αποτέλεσμα αυτής της διαδικασίας, κάποιες κορυφές γίνονται υπο-εξέταση και προστίθενται στο τέλος της ουράς. Η ΑΚΠ διατηρεί επίσης δύο βοηθητικούς πίνακες, τους m και p. Για κάθε κορυφή v, το $m[v]$ περιέχει την τρέχουσα κατάσταση της v σε σχέση με την ΑΚΠ (δηλ. αν η v είναι ανεξερεύνητη, υπο-εξέταση, ή εξερευνημένη) και το $p[v]$ την κορυφή από την οποία η ΑΚΠ επισκέφθηκε τη v για πρώτη φορά (είναι $p[v] = \text{NIL}$ ενόσω η v παραμένει ανεξερεύνητη). Για να διευκολύνουμε την περιγραφή του αλγόριθμου, χαρακτηρίζουμε τις ανεξερεύνητες κορυφές με το A, τις υπο-εξέταση κορυφές με το Y, και τις εξερευνημένες κορυφές με το E.

```
BFS(G(V, E), s)
    addToQueue(s); m[s] ← Y; p[s] ← NIL;
    for all v ∈ V \ {s} do
        m[v] ← A; p[v] ← NIL;
    while not emptyQueue() do
        u ← extractFromQueue(); m[u] ← E;
        for all v ∈ L[u] do
            if m[v] = A then
                addToQueue(v); m[v] ← Y; p[v] ← u;
```

΄Ενα παράδειγμα εφαρμογής της ΑΚΠ φαίνεται στο Σχήμα 5.4. Κάθε κορυφή αλλάζει χαρακτηρισμό δύο φορές (ή και λιγότερες αν το γράφημα δεν είναι συνεκτικό): από A σε Y όταν προστίθεται στο τέλος της ουράς, και από Y σε E όταν εξάγεται από την αρχή της ουράς. Η εισαγωγή στο τέλος και η εξαγωγή από την αρχή μιας FIFO ουράς υλοποιείται εύκολα σε σταθερό χρόνο. Ο συνολικός χρόνος για αυτές τις λειτουργίες είναι $O(n)$. Η λίστα των γειτόνων μιας κορυφής εξετάζεται όταν η κορυφή αλλάζει χαρακτηρισμό από Y σε E. Το άθροισμα των γειτόνων όλων των κορυφών είναι $\Theta(m)$ (βλ. Άσκηση 5.1) και ο αλγόριθμος αφιερώνει σταθερό χρόνο για κάθε γειτονική κορυφή που δεν χαρακτηρίζεται A. Ο αντίστοιχος χρόνος είναι $O(m)$. Επομένως, ο συνολικός χρόνος εκτέλεσης της ΑΚΠ είναι $O(n+m)$, δηλαδή γραμμικός στο μέγεθος της

λίστας γειτνίασης.

Παρατηρούμε ότι με εξαίρεση την αρχική κορυφή s, κάθε κορυφή v που είναι εξερευνημένη ή υπο-εξέταση έχει $p[v] \neq \text{NIL}$. Επίσης, στο τέλος του αλγόριθμου όλες οι κορυφές είναι είτε εξερευνημένες είτε ανεξερεύνητες γιατί κάθε υπο-εξέταση κορυφή εξάγεται από την ουρά πριν την ολοκλήρωση του αλγόριθμου και γίνεται εξερευνημένη. Μάλιστα, αν όλες οι υπόλοιπες κορυφές είναι προσπελάσιμες από την s (π.χ. το $G(V, E)$ είναι μη-κατευθυνόμενο συνεκτικό γράφημα), ο αλγόριθμος τερματίζει με όλες τις κορυφές εξερευνημένες.

Ας θεωρήσουμε το υπογράφημα $G_p(V_p, E_p)$ που αποτελείται από τις εξερευνημένες κορυφές και τις ακμές μέσω των οποίων τις επισκεφθήκαμε για πρώτη φορά. Δηλαδή, το $G_p(V_p, E_p)$ ορίζεται ως:

$$V_p = \{v \in V : m[v] = \text{E}\} \quad \text{και} \quad E_p = \{(p[v], v) \in E : v \in V_p \setminus \{s\}\}$$

Το γράφημα G_p είναι δέντρο γιατί η ΑΚΠ χρησιμοποιεί ένα μοναδικό μονοπάτι από την s στη v για να εξερευνήσει κάθε κορυφή $v \in V_p$. Μάλιστα το G_p είναι συνδετικό δέντρο του αρχικού γραφήματος $G(V, E)$, δηλαδή $V_p = V$, αν το G είναι μη-κατευθυνόμενο και συνεκτικό ή κατευθυνόμενο και ισχυρά συνεκτικό.

Το δέντρο G_p ονομάζεται δέντρο της ΑΚΠ. Όταν όλες οι ακμές του αρχικού γραφήματος είναι μοναδιαίου μήκους, το δέντρο της ΑΚΠ περιέχει ένα συντομότερο μονοπάτι από όλες τις υπόλοιπες κορυφές που είναι προσπελάσιμες από την s (βλ. επίσης Ενότητα 5.6.2 και Άσκηση 5.30). Επομένως, η ΑΚΠ υπολογίζει τις αποστάσεις όλων των κορυφών από την s όταν οι ακμές έχουν μοναδιαίο μήκος.

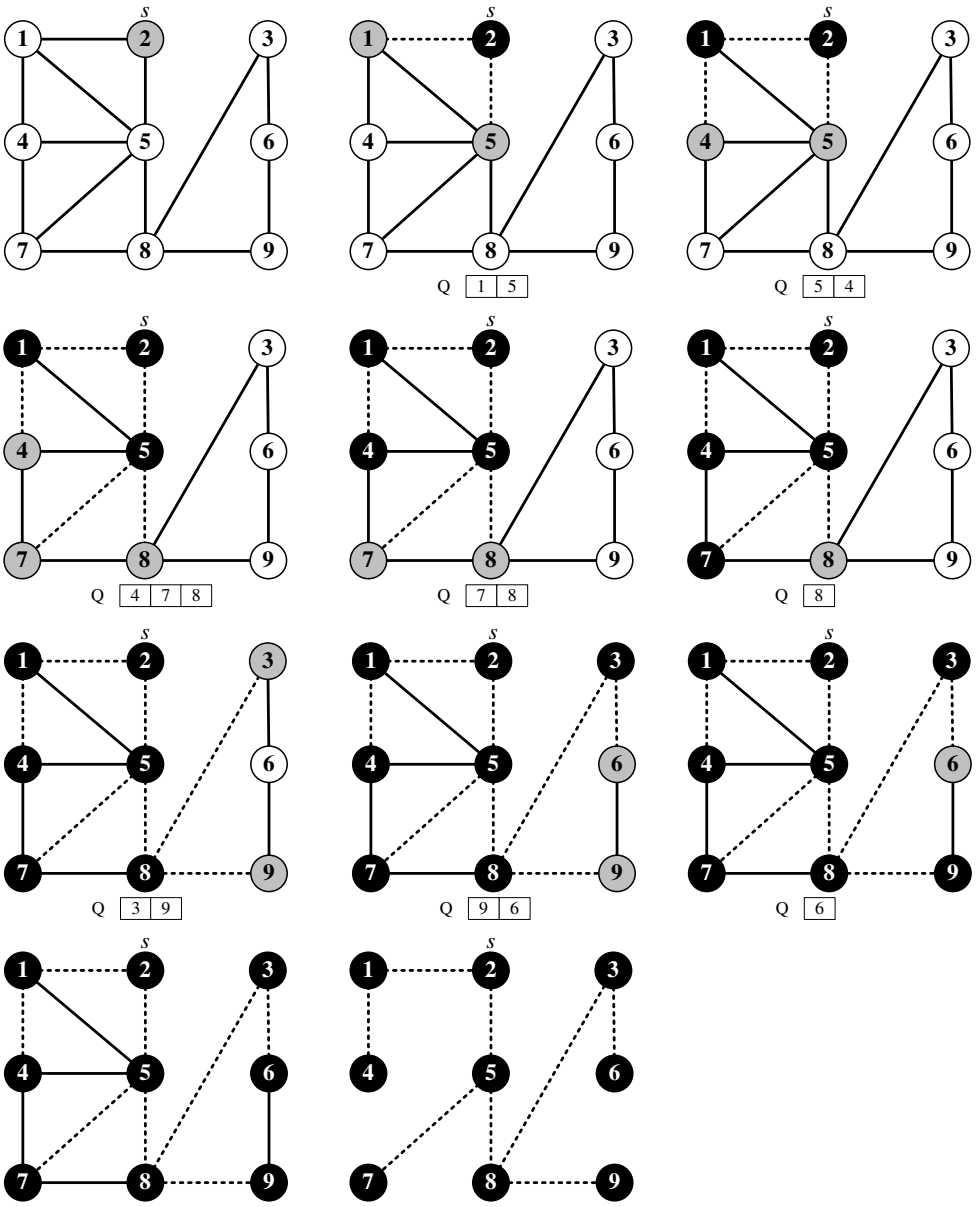
Άσκηση 5.7. Να αποδείξετε ότι ένα μη-κατευθυνόμενο γράφημα $G(V, E)$ είναι συνεκτικό αν και μόνο αν το δέντρο της ΑΚΠ είναι συνδετικό δέντρο του G (δηλ. $V_p = V$). Να τροποποιήσετε την ΑΚΠ ώστε να υπολογίζει τις συνεκτικές συνιστώσες του G.

Λύση. Αν το δέντρο της ΑΚΠ είναι συνδετικό δέντρο του G, το G είναι συνεκτικό αφού ένα υποσύνολο ακμών αρκεί για να "συνδέσει" όλες τις κορυφές. Αν το δέντρο της ΑΚΠ είναι δεν συνδετικό δέντρο του G, δηλ. $V_p \neq V$, δεν υπάρχουν ακμές μεταξύ των κορυφών του V_p και των (υπόλοιπων) κορυφών του $V \setminus V_p$. Επομένως καμία κορυφή του $V \setminus V_p$ δεν είναι προσπελάσιμη από την s. Έτσι το G δεν είναι συνεκτικό, και το υπογράφημα που επάγεται από τις κορυφές του V_p αποτελεί μια συνεκτική συνιστώσα του G.

Για να υπολογίσουμε τις συνεκτικές συνιστώσες, αρχικά θεωρούμε όλες τις κορυφές ως ανεξερεύνητες. Ενόσω υπάρχουν ανεξερεύνητες κορυφές, διαλέγουμε μια από αυτές σαν αρχική κορυφή και εφαρμόζουμε ΑΚΠ. Οι κορυφές που εξερευνώνται σε αυτή την κλήση της ΑΚΠ αποτελούν μια συνεκτική συνιστώσα. Η διαδικασία συνεχίζεται μέχρι όλες οι κορυφές να έχουν εξερευνηθεί και άρα ενταχθεί σε κάποια συνεκτική συνιστώσα. □

Άσκηση 5.8. Ποιά είναι τα αποτελέσματα της ΑΚΠ όταν το αρχικό γράφημα είναι: (α) πλήρες γράφημα, (β) δέντρο, (γ) απλός κύκλος, και (δ) αστέρας.

Άσκηση 5.9. Ένα γράφημα $G(V, E)$ ονομάζεται *διμερές* (bipartite) όταν υπάρχει διαμέριση των κορυφών του σε δύο σύνολα V_1 και V_2 ώστε κάθε ακμή του E να έχει το ένα άκρο της στο V_1 και



Σχήμα 5.4: Παράδειγμα εφαρμογής Αναζήτησης Κατά Πλάτος σε μη-κατευθυνόμενο γράφημα. Οι ανεξερεύνητες κορυφές σημειώνονται με λευκό χρώμα, οι υπο-εξέταση με γκρίζο, και οι εξερευνημένες με μαύρο. Οι ακμές που σημειώνονται με διακεκομμένες γραμμές χρησιμοποιήθηκαν για την επίσκεψη νέων κορυφών και συμπεριλαμβάνονται στο δέντρο της ΑΚΠ.

το άλλο άκρο της στο V_2. Ισοδύναμα, ένα γράφημα $G(V, E)$ είναι διμερές αν και μόνο αν δεν έχει κύκλους περιττού μήκους. Περιγράψτε έναν αποδοτικό αλγόριθμο που αποφασίζει αν ένα γράφημα είναι διμερές ή όχι.

Λύση. Χωρίς βλάβη της γενικότητας θεωρούμε ότι το γράφημα $G(V, E)$ είναι συνεκτικό και μη-κατευθυνόμενο. Η λύση που θα δώσουμε γενικεύεται εύκολα σε μη-συνεκτικά ή/και σε κατευθυνόμενα γραφήματα.

Ο αλγόριθμος συνίσταται στην εκτέλεση μιας ΑΚΠ από μια οποιαδήποτε αρχική κορυφή $s \in V$. Η ΑΚΠ υπολογίζει τις αποστάσεις όλων των κορυφών από την s. Στο V_1 βάζουμε τις κορυφές σε άρτια απόσταση από την s και στο V_2 βάζουμε τις κορυφές σε περιττή απόσταση από την s. Αν δεν υπάρχουν ακμές μεταξύ των κορυφών του V_1 και του V_2, το γράφημα είναι διμερές. Αν υπάρχει ακμή μεταξύ κάποιων κορυφών στο V_1 ή στο V_2, το γράφημα έχει κύκλο περιττού μήκους (να το αποδείξετε), οπότε δεν είναι διμερές. Ο αλγόριθμος έχει χρόνο εκτέλεσης $O(n + m)$. □

5.3 Αναζήτηση Κατά Βάθος

Η στρατηγική της Αναζήτησης Κατά Βάθος (Depth-First Search) είναι η απομάκρυνση από την αρχική κορυφή ενόσω αυτό είναι εφικτό. Όταν η Αναζήτηση Κατά Βάθος (ΑΚΒ) επισκέπτεται μια κορυφή για πρώτη φορά, εξερευνά όλες τις γειτονικές της πριν φύγει από αυτή. Η ΑΚΒ είναι φύσει αναδρομική διαδικασία: Ξεκινάει εξερευνώντας όλες τις γειτονικές κορυφές μιας κορυφής v που επιλέγεται σαν αρχική. Η σειρά με την οποία εξερευνώνται οι γειτονικές κορυφές της v καθορίζεται από τη θέση τους στη λίστα γειτνίασης. Κάθε φορά που επισκέπτεται μια νέα κορυφή u, η ΑΚΒ εφαρμόζει τον εαυτό της με αρχική κορυφή τη u. Αυτό συνεχίζεται μέχρι να εξερευνηθούν όλες οι κορυφές που είναι προσπελάσιμες από την v. Αν υπάρχουν ακόμη κορυφές που παραμένουν ανεξερεύνητες, επιλέγεται μια ανεξερεύνητη κορυφή σαν αρχική και εφαρμόζεται η ίδια διαδικασία.

Όπως και στην ΑΚΠ, μπορούμε να διακρίνουμε τρία είδη κορυφών σε κάθε σημείο του αλγόριθμου. Αρχικά όλες οι κορυφές είναι *ανεξερεύνητες*. Την πρώτη φορά που η ΑΚΒ επισκέπτεται μια ανεξερεύνητη κορυφή v, η v γίνεται *υπο-εξέταση* και η ΑΚΒ εφαρμόζεται αναδρομικά με αρχική κορυφή τη v. Η κορυφή v θεωρείται *εξερευνημένη* όταν ολοκληρωθεί η αναδρομική κλήση.

Η υλοποίηση της ΑΚΒ διατηρεί μια βοηθητική μεταβλητή, την t, και τρεις βοηθητικούς πίνακες, τους m, p, και d. Η τρέχουσα τιμή της μεταβλητής t αντιστοιχεί στον αριθμό των κορυφών που έχει επισκεφθεί η ΑΚΒ. Όταν η κορυφή v αλλάζει χαρακτηρισμό από ανεξερεύνητη σε υπο-εξέταση, η t αυξάνεται κατά 1 και το $d[v]$ τίθεται στην τρέχουσα τιμή της t. Έτσι το $d[v]$ δείχνει τη σειρά με την οποία η ΑΚΒ επισκέφθηκε την κορυφή v.

Η χρήση των m και p είναι ίδια με τη χρήση τους στην ΑΚΠ. Το $m[v]$ περιέχει την τρέχουσα κατάσταση της v σε σχέση με την ΑΚΒ και το $p[v]$ την κορυφή από την οποία η ΑΚΒ επισκέφθηκε τη v για πρώτη φορά. Όπως και στην ΑΚΠ, χαρακτηρίζουμε τις ανεξερεύνητες κορυφές με το A, τις υπο-εξέταση κορυφές με το Y, και τις εξερευνημένες κορυφές με το E.

DFS_Init($G(V, E)$) /* Αρχικοποίηση αναζήτησης */

```
    t ← 0;
    for all v ∈ V do
        m[v] ← A; p[v] ← NIL;
    for all v ∈ V do
        if m[v] = A then DFS(v);

DFS(v)
    t ← t + 1; m[v] ← Y; d[v] ← t;
    for all u ∈ L[v] do
        if m[u] = A then
            p[u] ← v; DFS(u);
    m[v] ← E;
```

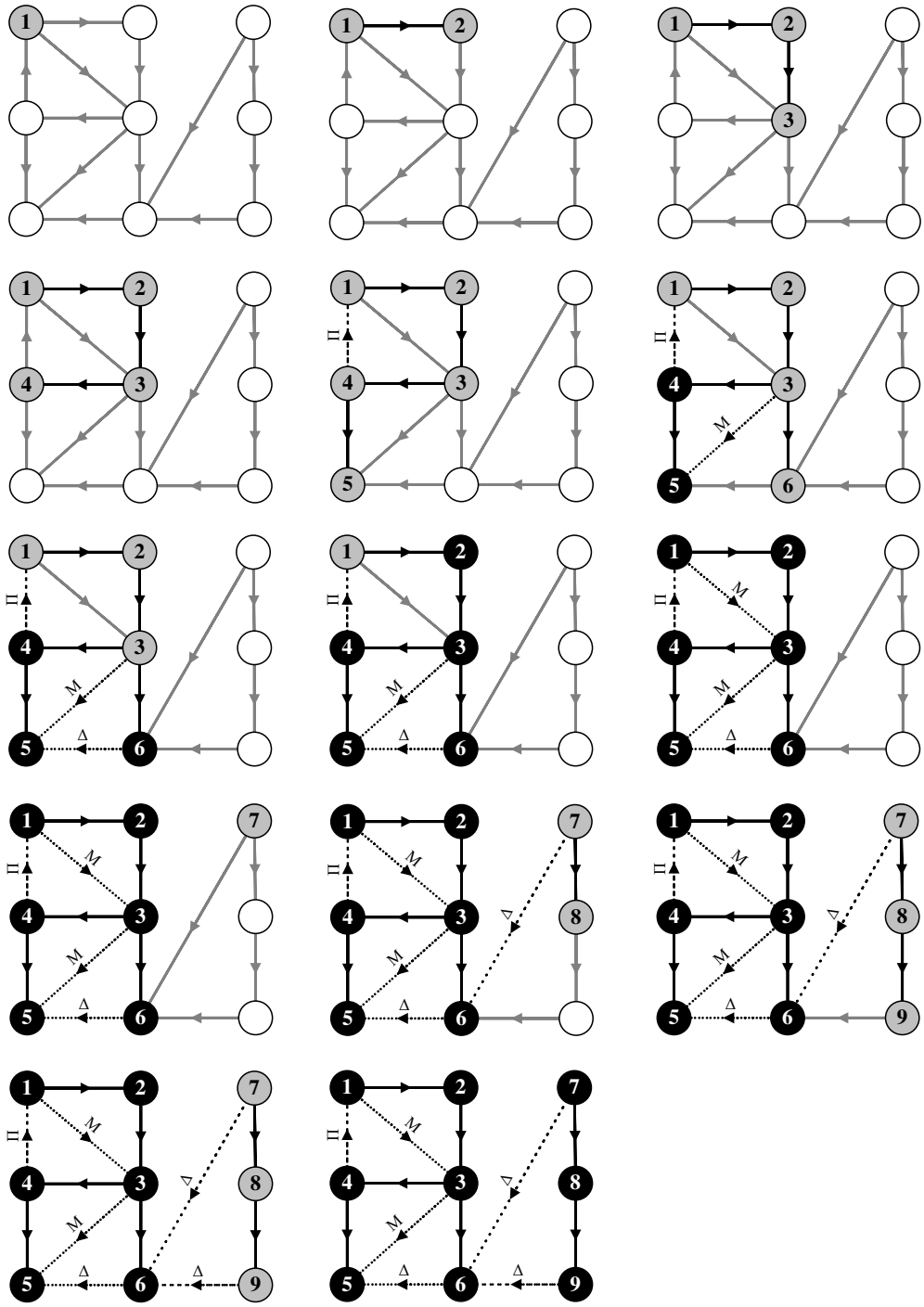
Ένα παράδειγμα εφαρμογής της ΑΚΒ φαίνεται στο Σχήμα 5.5. Η διαδικασία DFS_Init αρχικοποιεί τις μεταβλητές που χρησιμοποιούνται στον αλγόριθμο (παρατηρείστε ότι όλες οι μεταβλητές πρέπει να είναι σφαιρικής εμβέλειας) και ξεκινάει την αναδρομική διαδικασία DFS. Η διαδικασία DFS καλείται μία φορά για κάθε κορυφή, αφού η κλήση DFS(v) αλλάζει το χαρακτηρισμό της v από A σε Y και στη συνέχεια σε E. Η κλήση DFS(v) ελέγχει όλους τους γείτονες της v για κορυφές που παραμένουν ανεξερεύνητες. Επομένως, κάθε ακμή ελέγχεται μία φορά (δύο αν πρόκειται για μη-κατευθυνόμενο γράφημα). Ο συνολικός χρόνος εκτέλεσης της ΑΚΒ είναι $O(n+m)$, δηλαδή γραμμικός στο μέγεθος της λίστας γειτνίασης.

Όπως και στην ΑΚΠ, ορίζουμε το υπογράφημα $G_p(V, E_p)$ που αποτελείται από τις εξερευνημένες κορυφές και τις ακμές μέσω των οποίων τις επισκεφθήκαμε για πρώτη φορά. Σύμφωνα με τη διατύπωση της ΑΚΒ, το G_p είναι επικαλύπτον υπογράφημα του G γιατί ο αλγόριθμος τερματίζει αφού έχει εξερευνήσει όλες τις κορυφές. Το σύνολο ακμών ορίζεται ως $E_p = \{(p[v], v) \in E : v \in V \text{ και } p[v] \neq \text{NIL}\}$. Το γράφημα G_p είναι δάσος (δηλ. ακυκλικό) γιατί η πρώτη επίσκεψη σε κάθε κορυφή v γίνεται από μοναδικό μονοπάτι από την αντίστοιχη αρχική κορυφή προς τη v. Το G_p ονομάζεται δάσος της ΑΚΒ.

Το G_p είναι συνεκτικό μόνο αν όλες οι κορυφές είναι προσπελάσιμες από την πρώτη αρχική κορυφή. Ειδικότερα, αν το γράφημα εισόδου $G(V, E)$ είναι μη-κατευθυνόμενο, το G_p είναι συνεκτικό αν και μόνο αν το G είναι συνεκτικό. Σε διαφορετική περίπτωση, κάθε φορά που η DFS_Init καλεί τη DFS με διαφορετική αρχική κορυφή, ένα νέο δέντρο / συνεκτική συνιστώσα προστίθεται στο G_p. Μάλιστα, το δάσος G_p απεικονίζει την εξέλιξη των αναδρομικών κλήσεων της DFS. Συγκεκριμένα, το γεγονός ότι $p[u] = v$ σημαίνει ότι η κλήση DFS(u) έγινε κατά τον έλεγχο των γειτόνων της κορυφής v (με άλλα λόγια, η κλήση DFS(u) προέκυψε σαν αναδρομική κλήση από την DFS(v)).

Άσκηση 5.10. Ποια είναι τα αποτελέσματα της ΑΚΒ όταν το αρχικό γράφημα είναι: (α) πλήρες γράφημα, (β) δέντρο, (γ) απλός κύκλος, και (δ) αστέρας. Συγκρίνετε με τα αντίστοιχα αποτελέσματα της ΑΚΠ.

Άσκηση 5.11. Να αποδείξετε ότι αν το γράφημα εισόδου $G(V, E)$ είναι μη-κατευθυνόμενο, η ΑΚΒ υπολογίζει τις συνεκτικές συνιστώσες του G.



Σχήμα 5.5: Παράδειγμα εφαρμογής Αναζήτησης Κατά Βάθος σε κατευθυνόμενο γράφημα. Οι ανεξερεύνητες κορυφές σημειώνονται με λευκό χρώμα, οι υπο-εξέταση με γκρίζο, και οι εξερευνημένες με μαύρο. Η ετικέτα κάθε κορυφής αντιστοιχεί στη σειρά της πρώτης ΑΚΒ-επίσκεψης (δηλ. στην τιμή του $d[v]$). Οι ακμές που έχουν εξερευνηθεί από την ΑΚΒ σημειώνονται με έντονο μαύρο χρώμα. Οι ακμές του δάσους της ΑΚΒ σημειώνονται με συμπαγή βέλη και οι υπόλοιπες ακμές με διακεκομμένα βέλη.

Κατηγορίες Ακμών. Έχοντας σαν σημείο αναφοράς το δάσος της ΑΚΒ, μπορούμε να ορίσουμε τέσσερα διαφορετικά είδη ακμών:

- Οι *ακμές του δάσους* (forest ή tree edges) είναι οι ακμές στο σύνολο E_p. Μια ακμή (u, v) ανήκει στο δάσος αν χρησιμοποιήθηκε για την πρώτη επίσκεψη στην κορυφή v, δηλαδή τη στιγμή της εξερεύνησής της η v είναι ανεξερεύνητη.

- Οι *πίσω ακμές* (back edges) συνδέουν μία κορυφή u με μία κορυφή v που είναι πρόγονος της u στο δάσος της ΑΚΒ. Μία ακμή (u, v) είναι πίσω ακμή αν κατά την εξερεύνησή της η κορυφή v είναι υπό-εξέταση. Οι ανακυκλώσεις θεωρούνται πίσω ακμές.

- Οι *μπρος ακμές* (forward edges) συνδέουν μία κορυφή u με μία κορυφή v που είναι απόγονος της u στο δάσος της ΑΚΒ. Μία ακμή (u, v) είναι μπρος ακμή αν κατά την εξερεύνησή της η κορυφή v είναι εξερευνημένη και $d[u] < d[v]$ (δηλ. έχουμε επισκεφθεί τη u πριν τη v).

- Οι υπόλοιπες ακμές ονομάζονται *ακμές διασταύρωσης* (cross edges). Οι ακμές διασταύρωσης συνδέουν κορυφές στο ίδιο δέντρο που δεν έχουν σχέση προγόνου - απογόνου ή κορυφές σε διαφορετικά δέντρα του δάσους G_p. Μία ακμή (u, v) είναι ακμή διασταύρωσης αν κατά την εξερεύνησή της η κορυφή v είναι εξερευνημένη και $d[u] > d[v]$ (δηλ. έχουμε επισκεφθεί τη u μετά από τη v).

Στο Σχήμα 5.5, διακρίνουμε τα είδη των ακμών που παράγει η ΑΚΒ. Οι ακμές δάσους σημειώνονται με συμπαγή μαύρα βέλη και οι υπόλοιπες ακμές με διακεκομμένα μαύρα βέλη. Οι πίσω ακμές σημειώνονται με Π, οι μπρος ακμές με Μ, και οι ακμές διασταύρωσης με Δ. Το δάσος της ΑΚΒ έχει τελικά ζωγραφιστεί ώστε οι πίσω ακμές να κατευθύνονται προς τα πάνω, οι μπρος ακμές προς τα κάτω, και οι ακμές διασταύρωσης από δεξιά προς τα αριστερά.

Αν το γράφημα εισόδου είναι μη-κατευθυνόμενο, στην ακμή $\{u, v\}$ χαρακτηρίζεται με βάση τον πρώτο (χρονικά) χαρακτηρισμό που της αποδίδεται. Με άλλα λόγια, η ακμή $\{u, v\}$ χαρακτηρίζεται από την κατεύθυνση (u, v) ή (v, u) που εξερευνήθηκε πρώτη από την ΑΚΒ.

Άσκηση 5.12. Να αποδείξετε ότι η ΑΚΒ σε μη-κατευθυνόμενα γραφήματα παράγει μόνο ακμές δάσους και πίσω ακμές.

Λύση. Έστω $\{v, u\}$ ακμή μη-κατευθυνόμενου γραφήματος. Χωρίς βλάβη της γενικότητας, υποθέτουμε ότι $d[v] < d[u]$ (δηλαδή ότι η ΑΚΒ επισκέφθηκε τη v πριν τη u). Αφού η u είναι στη λίστα των γειτόνων της v, η κορυφή u γίνεται αρχικά υπο-εξέταση και στη συνέχεια εξερευνημένη πριν η v γίνει εξερευνημένη. Αν η ακμή $\{v, u\}$ εξερευνήθηκε πρώτα στην κατεύθυνση (v, u), θα γίνει ακμή δάσους. Αν εξερευνήθηκε πρώτα στην κατεύθυνση (u, v), θα γίνει πίσω ακμή. □

Άσκηση 5.13. Να αποδείξετε ότι ένα κατευθυνόμενο γράφημα είναι ακυκλικό αν και μόνο αν η ΑΚΒ δεν παράγει πίσω ακμές.

Λύση. Έστω κατευθυνόμενο γράφημα $G(V, E)$ και πίσω ακμή (u, v). Τη στιγμή που εξερευνάται η ακμή (u, v) τόσο η κορυφή u όσο και η κορυφή v είναι υπο-εξέταση. Επομένως, υπάρχει

μονοπάτι από τη v στη u που χρησιμοποιεί ακμές του δάσους. Το μονοπάτι συμπληρώνεται σε κύκλο από την ακμή (u, v).

Αντίστροφα, έστω ότι το γράφημα $G(V, E)$ περιέχει κύκλο C. Έστω v η πρώτη κορυφή του C που εξερευνάται από την ΑΚΒ, και έστω (u, v) η ("τελευταία") ακμή του κύκλου C που καταλήγει στην κορυφή v. Η u θα ενταχθεί στο δέντρο της ΑΚΒ με ρίζα τη v γιατί (α) υπάρχει μονοπάτι από τη v στην u (το μέρος του κύκλου $C \setminus (u, v)$), και (β) όταν επισκεπτόμαστε τη v, όλες οι κορυφές του C είναι ακόμη ανεξερεύνητες. Επομένως, η (u, v) θα γίνει πίσω ακμή. □

Άσκηση 5.14. Να αποδείξετε τον ισχυρισμό της Άσκησης 5.13 για μη-κατευθυνόμενα γραφήματα. Να διατυπώσετε αλγόριθμο με χρόνο εκτέλεσης $\Theta(n)$ (δηλαδή ανεξάρτητο του αριθμού των ακμών) που αποφασίζει αν ένα μη-κατευθυνόμενο γράφημα είναι ακυκλικό.

Λύση. Ο ισχυρισμός ότι ένα μη-κατευθυνόμενο γράφημα είναι ακυκλικό αν και μόνο αν η ΑΚΒ δεν παράγει πίσω ακμές αποδεικνύεται με τον ίδιο τρόπο. Ο αλγόριθμος εκτελεί την ΑΚΒ και τερματίζει όταν βρεθεί η πρώτη πίσω ακμή. Αν δεν υπάρχουν πίσω ακμές, έχουμε $m \leq n - 1$. Αν υπάρχουν πίσω ακμές, η πρώτη θα εξερευνηθεί ύστερα από την εξερεύνηση το πολύ $n - 1$ ακμών. Και στις δύο περιπτώσεις, ο χρόνος εκτέλεσης είναι $\mathrm{O}(n + m) = \mathrm{O}(n)$. □

5.3.1 Υπολογισμός Σημείων Κοπής

Μια κορυφή ενός μη-κατευθυνόμενου γραφήματος ονομάζεται *σημείο κοπής* (articulation point) αν η αφαίρεσή της αυξάνει τον αριθμό των συνεκτικών συνιστωσών. Ένα γράφημα ονομάζεται *δισυνεκτικό* (biconnected) αν είναι συνεκτικό και δεν έχει σημεία κοπής. Ο υπολογισμός των σημείων κοπής και η επιβεβαίωση της δισυνεκτικότητας ενός γραφήματος έχουν πολλές πρακτικές εφαρμογές. Για παράδειγμα, η δισυνεκτικότητα ενός τηλεπικοινωνιακού δικτύου εξασφαλίζει ότι το δίκτυο θα συνεχίσει να λειτουργεί ομαλά ακόμη και όταν ένας κόμβος τεθεί εκτός λειτουργίας. Από την άλλη πλευρά, αν ένας κόμβος που αντιστοιχεί σε σημείο κοπής τεθεί εκτός λειτουργίας, κάποια τμήματα του δικτύου δεν θα μπορούν να επικοινωνήσουν.

Σε αυτή την ενότητα, θα δείξουμε πως η ΑΚΒ μπορεί να χρησιμοποιηθεί για τον υπολογισμό των σημείων κοπής ενός μη-κατευθυνόμενου γραφήματος $G(V, E)$. Χωρίς βλάβη της γενικότητας, θα θεωρήσουμε ότι το γράφημα εισόδου G είναι συνεκτικό. Έστω $G_p(V, E_p)$ το συνδετικό δέντρο που παράγει η ΑΚΒ και s η κορυφή από την οποία ξεκίνησε η ΑΚΒ. Στο εξής θεωρούμε την s σαν ρίζα του G_p.

Πρόταση 5.1. *Η αρχική κορυφή / ρίζα s είναι σημείο κοπής του G αν και μόνο αν έχει τουλάχιστον δύο παιδιά στο G_p.*

Απόδειξη. Αν η κορυφή s έχει μόνο ένα παιδί στο G_p, η αφαίρεσή της δεν αυξάνει τις συνεκτικές συνιστώσες του G_p. Συνεπώς, η s δεν είναι σημείο κοπής του G_p και άρα ούτε του G.

Για το αντίστροφο, έστω ότι η s έχει τουλάχιστον δύο παιδιά στο G_p. Αφού η ΑΚΒ παράγει μόνο ακμές δάσους και πίσω ακμές σε μη-κατευθυνόμενα γραφήματα (βλ. Άσκηση 5.12), όλα τα μονοπάτια που συνδέουν κορυφές σε διαφορετικά υποδέντρα με ρίζες παιδιά της s πρέπει να διέρχονται από την s. Άρα η s είναι σημείο κοπής. □

Το παραπάνω κριτήριο προκύπτει από το γεγονός ότι η ρίζα του G_p δεν έχει προγόνους. Έτσι οι απόγονοί της δεν μπορούν να την παρακάμψουν μέσω πίσω ακμών. Το κριτήριο είναι διαφορετικό για τις υπόλοιπες κορυφές του G_p, οι οποίες έχουν προγόνους.

Πρόταση 5.2. *Έστω v μία κορυφή διαφορετική από την s. Η v είναι σημείο κοπής του G αν και μόνο αν υπάρχει στο G_p υποδέντρο με ρίζα παιδί της v του οποίου καμία κορυφή δεν έχει πίσω ακμή προς πρόγονο της v.*

Απόδειξη. Έστω $u_1, \ldots, u_k$ τα παιδιά της v στο G_p, και έστω $T_{u_1}, \ldots, T_{u_k}$ τα υποδέντρα με ρίζες τα παιδιά της v. Αν σε όλα τα υποδέντρα $T_{u_1}, \ldots, T_{u_k}$ υπάρχουν κορυφές με πίσω ακμές προς προγόνους της v, η κορυφή v δεν είναι σημείο κοπής γιατί οι πίσω ακμές εξασφαλίζουν τη συνεκτικότητα των κορυφών στα $T_{u_1}, \ldots, T_{u_k}$ με τις υπόλοιπες κορυφές στο G_p και μεταξύ τους. Για το αντίστροφο, έστω ότι η κορυφή v είναι σημείο κοπής. Η αφαίρεση της v πρέπει να "αποσυνδέει" κάποιο υποδέντρο T_{u_i} από τις υπόλοιπες κορυφές του G_p. Επομένως, πρέπει να υπάρχει T_{u_i} του οποίου καμία κορυφή δεν έχει πίσω ακμή προς πρόγονο της v. □

Είναι εύκολο να ελέγξουμε το κριτήριο της Πρότασης 5.1 για τη ρίζα του G_p. Για να ελέγξουμε το κριτήριο της Πρότασης 5.2 που αφορά τις υπόλοιπες κορυφές, ορίζουμε την παρακάτω ποσότητα $\ell(v)$ για κάθε κορυφή $v \in V$:

$$\ell(v) = \min \begin{cases} \ell(u) & \text{για κάθε παιδί } u \text{ της } v \text{ στο } G_p \\ d[w] & \text{για κάθε πρόγονο } w \text{ της } v \text{ για τον οποίο υπάρχει πίσω ακμή } (v, w) \\ d[v] & \end{cases}$$

Με απλά λόγια, η κορυφή v έχει μονοπάτι που *καταλήγει με πίσω ακμή* προς τον πρόγονό της w με σειρά επίσκεψης από την ΑΚΒ $d[w] = \ell(v)$. Το μονοπάτι αυτό χρησιμοποιεί μια πίσω ακμή από τη v ή κάποιο απόγονό της προς την w. Επομένως, οι ακμές του G_p μεταξύ w και v δεν είναι απαραίτητες για τη σύνδεση τους.

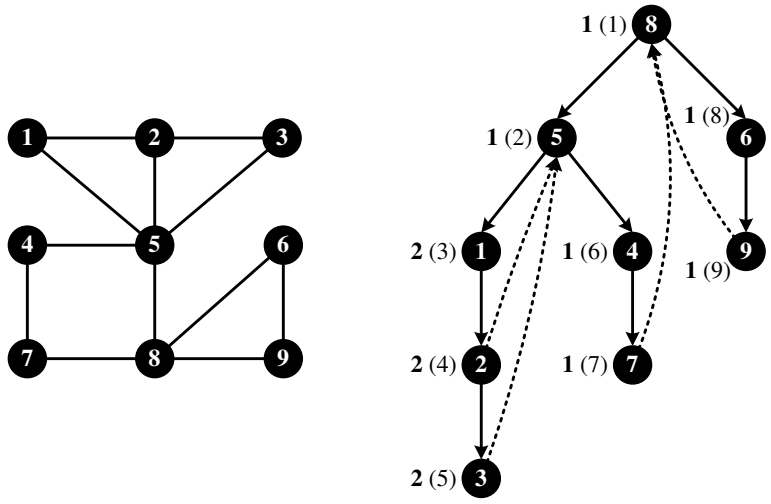
Η παρακάτω πρόταση προκύπτει από την Πρόταση 5.2 και τον ορισμό του $\ell(v)$.

Πρόταση 5.3. *Έστω κορυφή v διαφορετική από τη ρίζα του G_p. Η v είναι σημείο κοπής αν και μόνο αν κάποιο παιδί της u στο G_p έχει $\ell(u) \geq d[v]$.*

Απόδειξη. Έστω ότι υπάρχει παιδί u της v με $\ell(u_i) \geq d[v]$. Τότε όλες οι πίσω ακμές που ξεκινούν από το υποδέντρο με ρίζα τη u καταλήγουν σε κορυφές με σειρά ΑΚΒ-επίσκεψης μεγαλύτερη ή ίση από τη σειρά επίσκεψης της v. Αυτές οι κορυφές είναι απόγονοι της v (ή η ίδια η v) στο G_p. Εφαρμόζοντας την Πρόταση 5.2, συμπεραίνουμε ότι η κορυφή v είναι σημείο κοπής.

Αντίστροφα, έστω $u_1, \ldots, u_k$ τα παιδιά της v στο G_p. Αν για κάθε u_i, $i = 1, \ldots, k$, είναι $\ell(u_i) < d[v]$, τότε όλα τα υποδέντρα με ρίζες παιδιά της v έχουν πίσω ακμές προς κορυφές με σειρά ΑΚΒ-επίσκεψης μικρότερη από $d[v]$. Αυτές οι κορυφές είναι πρόγονοι της v στο G_p. Εφαρμόζοντας την Πρόταση 5.2, συμπεραίνουμε ότι η κορυφή v δεν είναι σημείο κοπής. □

Η παραπάνω πρόταση υποδεικνύει τον ακόλουθο αλγόριθμο για τον υπολογισμό των σημείων κοπής ενός μη-κατευθυνόμενου γραφήματος $G(V, E)$. Ένα παράδειγμα λειτουργίας του αλγόριθμου φαίνεται στο Σχήμα 5.6.



Σχήμα 5.6: Παράδειγμα λειτουργίας του αλγόριθμου για τον υπολογισμό των σημείων κοπής. Αριστερά απεικονίζεται το γράφημα εισόδου $G(V, E)$. Δεξιά απεικονίζεται το δέντρο της ΑΚΒ. Οι πίσω ακμές έχουν σχεδιαστεί διακεκομμένες. Αριστερά κάθε κορυφής v σημειώνεται η τιμή του $\ell(v)$ και σε παρένθεση η σειρά επίσκεψης $d[v]$. Τα σημεία κοπής είναι η ρίζα 8 επειδή έχει δύο παιδιά και η κορυφή 5 επειδή $\ell(1) = 2 \geq 2 = d[5]$.

1. Εκτελούμε ΑΚΒ στο γράφημα $G(V, E)$ και υπολογίζουμε το δέντρο $G_p(V, E_p)$ και τη σειρά επίσκεψης $d[v]$ για κάθε κορυφή $v \in V$.

2. Εξετάζουμε όλες τις κορυφές του G_p αναδρομικά αρχίζοντας από το αριστερότερο υποδέντρο, συνεχίζοντας προς τα δεξιά, και εξετάζοντας τη ρίζα τελευταία (αντίστοιχα με την μετα-διατεταγμένη (postorder) διέλευση). Για κάθε κορυφή v υπολογίζουμε το $\ell(v)$ σαν το ελάχιστο των $d[v]$, $d[w]$ για κάθε πρόγονο w της v προς τον οποίο υπάρχει πίσω ακμή (v, w), και $\ell(u)$ για κάθε παιδί u της v.

3. Τα σημεία κοπής είναι η ρίζα s του G_p εφόσον έχει δύο ή περισσότερα παιδιά στο G_p και κάθε κορυφή v με παιδί u που έχει $\ell(u) \geq d[v]$.

Τόσο ο υπολογισμός του $\ell(v)$ όσο και ο έλεγχος αν μια κορυφή v είναι σημείο κοπής μπορούν να ενσωματωθούν στη λειτουργία της ΑΚΒ. Ο υπολογισμός του $\ell(v)$ και ο έλεγχος για την ύπαρξη παιδιού u με $\ell(u) \geq d[v]$ μπορούν να πραγματοποιηθούν στην επανάληψη που εξερευνά τις ακμές που εφάπτονται στη v. Ο χρόνος για τους επιπλέον υπολογισμούς είναι σταθερός για κάθε ακμή. Συνεπώς, ο συνολικός χρόνος εκτέλεσης του αλγόριθμου είναι $\mathrm{O}(n + m) = \mathrm{O}(m)$.

Άσκηση 5.15. Μία κορυφή r ονομάζεται *ρίζα* (root) ενός κατευθυνόμενου γραφήματος $G(V, E)$ αν κάθε άλλη κορυφή v είναι προσπελάσιμη από την r. Ένα κατευθυνόμενο γράφημα μπορεί να έχει καμία, μία, ή περισσότερες ρίζες.

1. Έστω s η ρίζα του τελευταίου δέντρου που παράγεται από την ΑΚΒ στο G. Να δείξετε ότι αν το G έχει ρίζες, η κορυφή s είναι μία από αυτές.

2. Να διατυπώσετε αλγόριθμο με γραμμικό χρόνο εκτέλεσης που επιστρέφει μία ρίζα του γραφήματος εισόδου $G(V, E)$ ή NIL αν το γράφημα δεν έχει ρίζα.

3. Έστω ότι δίνεται μία ρίζα του G. Να διατυπώσετε αλγόριθμο με γραμμικό χρόνο εκτέλεσης που υπολογίζει όλες τις ρίζες του G.

Λύση. Αν η ΑΚΒ παράγει περισσότερα από ένα δέντρα, μόνο οι κορυφές του τελευταίου μπορούν να είναι ρίζες γιατί η s δεν είναι προσπελάσιμη από καμία κορυφή των προηγούμενων δέντρων. Αν κάποια κορυφή v του τελευταίου δέντρου είναι ρίζα, τότε ρίζα είναι και η s γιατί η v είναι προσπελάσιμη από την s. Μέσω της v, η s μπορεί να προσπελάσει όλες τις υπόλοιπες κορυφές.

Ο αλγόριθμος εκτελεί ΑΚΒ στο G και υπολογίζει τη ρίζα s του τελευταίου δέντρου. Στη συνέχεια, εκτελεί νέα ΑΚΒ αρχίζοντας από την s. Αν η s είναι ρίζα, η ΑΚΒ ολοκληρώνεται με όλες τις κορυφές εξερευνημένες. Διαφορετικά, το G δεν έχει ρίζες.

Έστω r η δεδομένη ρίζα του G. Μια κορυφή u είναι επίσης ρίζα αν και μόνο αν η r είναι προσπελάσιμη από τη u. Για να βρούμε τις κορυφές από τις οποίες η r είναι προσπελάσιμη, εκτελούμε DFS(r) (ή και BFS(r)) στο ανάστροφο γράφημα $G^T(V, E^T)$ (βλ. Άσκηση 5.6). Οι εξερευνημένες κορυφές στο τέλος της κλήσης αποτελούν το σύνολο των ριζών του G. □

5.3.2 Τοπολογική Διάταξη

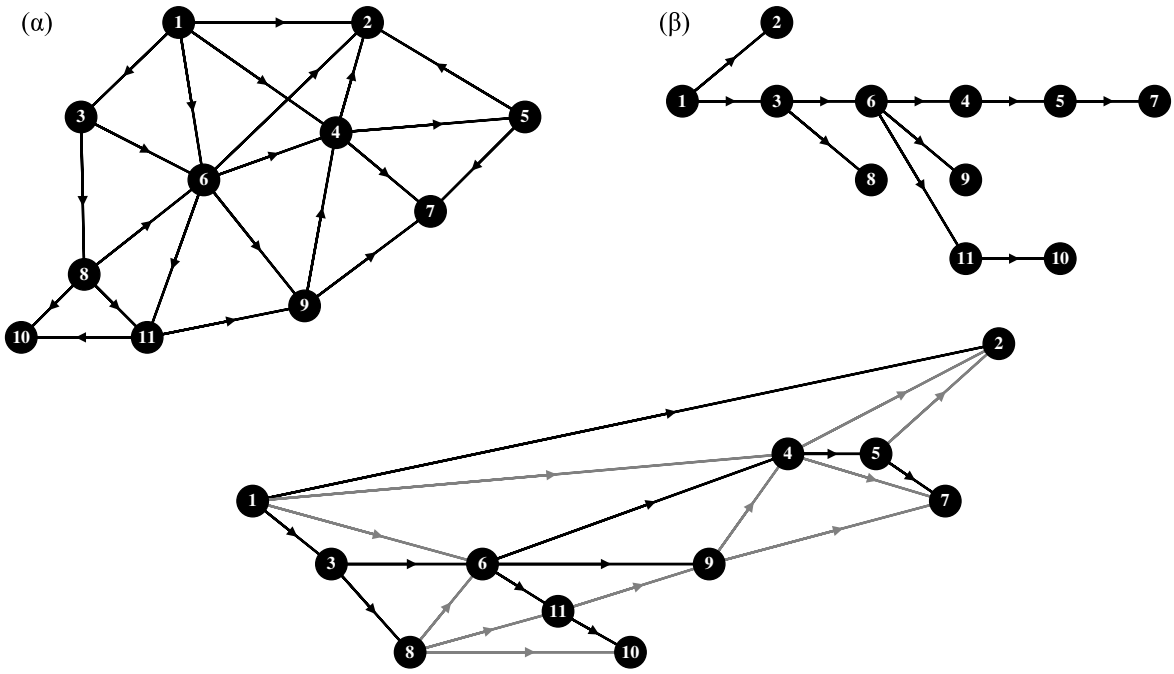
Ένα κατευθυνόμενο ακυκλικό γράφημα[1] μπορεί να χρησιμοποιηθεί για την αναπαράσταση μιας σχέσης μερικής διάταξης, όπου οι κορυφές αντιστοιχούν στα στοιχεία του συνόλου και οι ακμές στα ζεύγη που ανήκουν στη σχέση. Συγκεκριμένα, η ακμή (u, v) δηλώνει ότι το u είναι "μικρότερο ή ίσο" (ή "μεγαλύτερο ή ίσο") του v στη σχέση μερικής διάταξης. Μια πρακτική εφαρμογή είναι στον καθορισμό της σειράς υπολογισμού αλγεβρικών παραστάσεων με επεναλαμβανόμενα τμήματα (π.χ. $a\,c\,x^2 + [(a + c)(b + d) - a\,c - b\,d]\,x + b\,d$).

Μια άλλη σημαντική πρακτική εφαρμογή είναι ο προγραμματισμός των εργασιών που συνθέτουν ένα μεγάλο έργο. Σε αυτή την περίπτωση, η εξάρτηση της έναρξης κάποιων εργασιών από την ολοκλήρωση κάποιων άλλων ορίζει μια σχέση μερικής διάταξης στο σύνολο των εργασιών. Αυτή η σχέση αναπαρίσταται φυσιολογικά από ένα κατευθυνόμενο ακυκλικό γράφημα. Κάθε εργασία αντιστοιχεί σε μία κορυφή του γραφήματος. Η ύπαρξη της ακμής (u, v) δηλώνει ότι η εργασία v δεν μπορεί να αρχίσει πριν ολοκληρωθεί η εργασία u. Οι εξαρτήσεις έχουν νόημα μόνο αν δεν υπάρχει κύκλος (αν υπήρχε, θα αποτελούσε το λεγόμενο "φαύλο κύκλο").

Η ΑΚΒ μπορεί να χρησιμοποιηθεί για να ελέγξουμε αν ένα γράφημα είναι ακυκλικό (Άσκηση 5.13). Στον προγραμματισμό εργασιών όμως δεν αρκεί να επιβεβαιώσουμε ότι οι εξαρτήσεις έχουν οριστεί σωστά. Χρειάζεται να υπολογίσουμε μια σειρά έναρξης των εργασιών που να είναι συμβατή με τις εξαρτήσεις που έχουμε ορίσει. Αυτή η λειτουργία είναι γνωστή σαν τοπολογική διάταξη (topological sort).

Μια γραμμική τοποθέτηση των κορυφών ενός κατευθυνόμενου ακυκλικού γραφήματος $G(V, E)$ ονομάζεται *τοπολογική διάταξη* αν για κάθε ακμή $(u, v) \in E$, η κορυφή u προηγείται της v. Με απλά λόγια, οι κορυφές του γραφήματος τοποθετούνται σε μια ευθεία ώστε όλες οι

[1] Τα κατευθυνόμενα ακυκλικά γραφήματα είναι γνωστά σαν DAGs στη διεθνή βιβλιογραφία, από τα αρχικά των λέξεων Directed Acyclic Graphs.



(γ) Τοπολογική Διάταξη: 1, 3, 8, 6, 11, 10, 9, 4, 5, 7, 2

Σχήμα 5.7: Παράδειγμα υπολογισμού τοπολογικής διάταξης. (α) Το γράφημα εισόδου $G(V, E)$. (β) Το δέντρο της ΑΚΒ με αρχική κορυφή την 1. (γ) Η τοπολογική διάταξη. Οι κορυφές έχουν ζωγραφιστεί από τα αριστερά προς τα δεξιά σύμφωνα με την τοπολογική διάταξη. Οι ακμές του ΑΚΒ δέντρου απεικονίζονται μαύρες και οι υπόλοιπες ακμές του γραφήματος γκρίζες. Όλες οι ακμές έχουν κατεύθυνση από τα αριστερά προς τα δεξιά.

ακμές να έχουν κατεύθυνση από αριστερά προς τα δεξιά. Υπάρχει μια τοπολογική διάταξη για τις κορυφές ενός κατευθυνόμενου γραφήματος αν και μόνο αν το γράφημα είναι ακυκλικό.

Η ΑΚΒ μπορεί να τροποποιηθεί ώστε να υπολογίζει μια τοπολογική διάταξη ενός κατευθυνόμενου ακυκλικού γραφήματος. Ο τροποποιημένος αλγόριθμος διατηρεί μια Last-In-First-Out (LIFO) ουρά Q. Κάθε κορυφή v εισάγεται στην αρχή της ουράς τη στιγμή που χαρακτηρίζεται εξερευνημένη (στην τελευταία γραμμή της κλήσης DFS(v)). Η τοπολογική διάταξη προκύπτει τυπώνοντας τις κορυφές σε αντίστροφη σειρά από τη σειρά εισαγωγής τους στην ουρά Q (δηλαδή Last-In-First-Out) μετά την ολοκλήρωση της ΑΚΒ. Η εκτύπωση πρέπει να γίνει μετά την ολοκλήρωση της for-επανάληψης στην διαδικασία DFS_Init, οπότε όλες οι κορυφές είναι εξερευνημένες και η ΑΚΒ έχει ολοκληρωθεί. Ο χρόνος εκτέλεσης του αλγόριθμου παραμένει $\Theta(n + m)$, δηλαδή γραμμικός στο μέγεθος της λίστας γειτνίασης. Ένα παράδειγμα λειτουργίας του τροποποιημένου αλγόριθμου φαίνεται στο Σχήμα 5.7.

Για να δείξουμε ότι αυτή η τροποποίηση στην ΑΚΒ παράγει όντως μια τοπολογική διάταξη, αρκεί να δείξουμε ότι για κάθε ακμή (u, v), η κορυφή u χαρακτηρίζεται εξερευνημένη και εισάγεται στην ουρά μετά την κορυφή v. Αν ισχύει αυτό, η u προηγείται στην τοπολογική διάταξη της κορυφής v επειδή η ουρά είναι LIFO.

Πρόταση 5.4. *Για κάθε ακμή (u, v) ενός κατευθυνόμενου ακυκλικού γραφήματος, η κορυφή u χαρακτηρίζεται εξερευνημένη και εισάγεται στην ουρά μετά από την κορυφή v.*

Απόδειξη. Έστω μια οποιαδήποτε ακμή (u, v). Τη στιγμή που η ΑΚΒ εξερευνά την (u, v), η κορυφή u είναι υπο-εξέταση και η κορυφή v είναι είτε εξερευνημένη είτε ανεξερεύνητη. Η κορυφή v δεν είναι υπο-εξέταση γιατί η (u, v) θα ήταν πίσω ακμή και το γράφημα θα είχε κύκλο (βλ. Άσκηση 5.13).

Αν η v είναι εξερευνημένη, βρίσκεται ήδη στην ουρά. Αφού η u είναι ακόμη υπο-εξέταση, εισάγεται στην ουρά μετά τη v. Αν η v είναι ανεξερεύνητη, η ακμή (u, v) είναι ακμή δάσους και πραγματοποιείται η κλήση DFS(v). Αυτή καταλήγει χαρακτηρίζοντας τη v εξερευνημένη κορυφή και εισάγοντάς την στην ουρά. Η u παραμένει υπο-εξέταση ενόσω εξελίσσεται η κλήση DFS(v). Επομένως, η u χαρακτηρίζεται εξερευνημένη και εισάγεται στην ουρά μετά την v. □

Άσκηση 5.16. Στο γράφημα του Σχήματος 5.7, να υπολογίσετε την τοπολογική διάταξη που προκύπτει αν η ΑΚΒ ξεκινήσει από την κορυφή 3.

5.4 Ελάχιστο Συνδετικό Δέντρο

Έστω ένα συνεκτικό μη-κατευθυνόμενο γράφημα $G(V, E, w)$ με βάρη στις ακμές, όπου το βάρος κάθε ακμής δίνεται από τη συνάρτηση $w : E \mapsto \mathbb{R}^*_+$. Το βάρος ενός επικαλύπτοντος υπογραφήματος $T(V, E_T)$ του G συμβολίζεται με $w(T)$ και είναι ίσο με το συνολικό βάρος των ακμών του, δηλαδή $w(T) = \sum_{e \in E_T} w(e)$.

Θέλουμε να υπολογίσουμε το συνεκτικό επικαλύπτον υπογράφημα του G με το ελάχιστο βάρος. Αυτό το υπογράφημα θα είναι δέντρο, αφού πρέπει να είναι συνεκτικό και ακυκλικό. Η συνεκτικότητα προκύπτει εξ' ορισμού. Η απουσία κύκλων έπεται από την απαίτηση για ελάχιστο συνολικό βάρος. Αν υπήρχαν κύκλοι, θα μπορούσαμε να μειώσουμε το βάρος διατηρώντας τη συνεκτικότητα με την αφαίρεση μιας ακμής που βρίσκεται σε κύκλο.

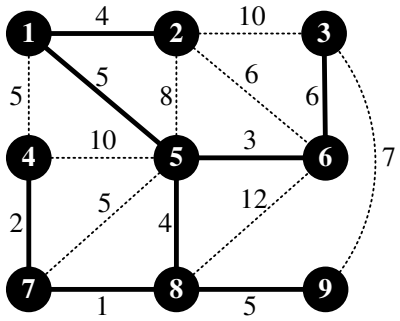
Το *Ελάχιστο Συνδετικό Δέντρο* (Minimum Spanning Tree - MST) ενός γραφήματος $G(V, E, w)$ με βάρη στις ακμές είναι το συνδετικό δέντρο[2] του G με το ελάχιστο συνολικό βάρος. Το πρόβλημα του υπολογισμού ενός τέτοιου δέντρου είναι γνωστό σαν πρόβλημα του Ελάχιστου Συνδετικού Δέντρου και αποτελεί ένα τυπικό παράδειγμα προβλήματος συνδυαστικής βελτιστοποίησης.

Το πρόβλημα του Ελάχιστου Συνδετικού Δέντρου εφαρμόζεται στο σχεδιασμό δικτύων όταν επιδιώκουμε να ελαχιστοποιήσουμε το κόστος σύνδεσης κάποιων σημείων (π.χ. σχεδιασμός ηλεκτρικού, επικοινωνιακού, ή οδικού δικτύου). Σε αυτή την ενότητα, θα περιγράψουμε δύο άπληστους αλγόριθμους για τον υπολογισμό ενός Ελάχιστου Συνδετικού Δέντρου (ΕΣΔ).

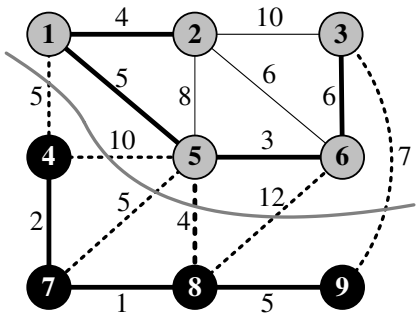
5.4.1 Άπληστος Υπολογισμός Ελάχιστου Συνδετικού Δέντρου

Για να εξηγήσουμε την βασική ιδέα των αλγορίθμων, χρειάζεται να εισάγουμε την έννοια της τομής ενός γραφήματος. Μια *τομή* (cut) $(S, V \setminus S)$ ενός γραφήματος $G(V, E)$ είναι μια διαμέριση των κορυφών του σε δύο υποσύνολα $S \subseteq V$ και $V \setminus S$ (βλ. Σχήμα 5.9). Το σύνολο των ακμών που έχουν το ένα άκρο τους στο S και το άλλο στο $V \setminus S$ ονομάζεται *σύνολο τομής* (cut set) και συμβολίζεται

[2] Ο αναγνώστης που δεν είναι εξοικειωμένος με την έννοια και τις βασικές ιδιότητες των συνδετικών δέντρων μπορεί να βρει μια σύντομη εισαγωγή στο Παράρτημα, Κεφάλαιο Β.



Σχήμα 5.8: Παράδειγμα Ελάχιστου Συνδετικού Δέντρου (ΕΣΔ). Οι ακμές του δέντρου σημειώνονται παχύτερες και συμπαγείς.



Σχήμα 5.9: Παράδειγμα τομής. Το σύνολο S αποτελείται από τις γκρίζες κορυφές, $S = \{1, 2, 3, 5, 6\}$, και το σύνολο $V \setminus S$ από τις μαύρες κορυφές, $V \setminus S = \{4, 7, 8, 9\}$. Οι παχύτερες συμπαγείς ακμές αποτελούν τα ΕΣΔ για τα δύο επαγόμενα υπογραφήματα. Οι ακμές του συνόλου $\delta(S, V \setminus S)$, δηλαδή οι ακμές που διασχίζουν την τομή, σημειώνονται διακεκομμένες. Η ακμή ελάχιστου βάρους που διασχίζει την τομή είναι η $\{5, 8\}$ με βάρος 4.

με $\delta(S, V \setminus S)$. Τυπικά, για κάθε $S \subseteq V$, $\delta(S, V \setminus S) = \{(u, v) \in E : u \in S \text{ και } v \notin S\}$. Μια ακμή e *διασχίζει* την τομή $(S, V \setminus S)$ αν έχει το ένα άκρο της στο S και το άλλο στο $V \setminus S$ (δηλαδή αν $e \in \delta(S, V \setminus S)$). Ένα σύνολο ακμών $E' \subseteq E$ διασχίζει την τομή $(S, V \setminus S)$ αν $E' \cap \delta(S, V \setminus S) \neq \emptyset$ (δηλ. υπάρχει ακμή στο E' που διασχίζει την τομή).

Το σύνολο των ακμών κάθε συνδετικού δέντρου πρέπει να διασχίζει όλες τις τομές του γραφήματος. Διαφορετικά, θα υπήρχε ένα σύνολο κορυφών που δεν συνδέεται με τις υπόλοιπες. Με βάση αυτή την παρατήρηση, το ΕΣΔ αντιστοιχεί στο σύνολο ακμών ελάχιστου συνολικού βάρους που διασχίζει όλες τις τομές του γραφήματος.

Θα δείξουμε ότι το πρόβλημα του ΕΣΔ έχει τις ιδιότητες των βέλτιστων επιμέρους λύσεων και της άπληστης επιλογής, και επομένως μπορούμε να εφαρμόσουμε τη μέθοδο της απληστίας.

Έστω $T(V, E_T)$ ένα ΕΣΔ για το γράφημα $G(V, E, w)$, και έστω e μια οποιαδήποτε ακμή στο E_T. Η αφαίρεση της e από το T δημιουργεί δύο συνεκτικές συνιστώσες. Έστω S και $V \setminus S$ τα σύνολα κορυφών των δύο συνεκτικών συνιστωσών, και έστω T_S και $T_{\overline{S}}$ τα αντίστοιχα υποδέντρα του $T \setminus e$. Το T_S είναι ένα ΕΣΔ του επαγόμενου υπογραφήματος με σύνολο κορυφών S, και το $T_{\overline{S}}$ είναι ένα ΕΣΔ του επαγόμενου υπογραφήματος με σύνολο κορυφών $\overline{S} = V \setminus S$. Διαφορετικά,

θα μπορούσαμε να μειώσουμε το βάρος του T παίρνοντας ένα ελαφρύτερο συνδετικό δέντρο για κάποιο από τα δύο επαγόμενα υπογραφήματα. Αυτό αποδεικνύει ότι το πρόβλημα του ΕΣΔ έχει την ιδιότητα των βέλτιστων επιμέρους λύσεων.

Όσον αφορά στην ιδιότητα της άπληστης επιλογής, παρατηρούμε ότι η ακμή e είναι μια ελαφρύτερη ακμή που διασχίζει την τομή $(S, V \setminus S)$. Διαφορετικά, θα μπορούσαμε να μειώσουμε το βάρος του T αντικαθιστώντας την e με μία ελαφρύτερη ακμή από το σύνολο τομής $\delta(S, V \setminus S)$ (βλ. επίσης Άσκηση B.2 και Θεώρημα 5.2).

Αυτές οι ιδιότητες επιτρέπουν τον υπολογισμό ενός ΕΣΔ από άπληστους αλγόριθμους που λειτουργούν αυξητικά. Στη γενική του μορφή, ο αλγόριθμος διατηρεί ένα δάσος Δ το οποίο αποτελεί υπογράφημα ενός ΕΣΔ[3]. Αρχικά, το δάσος είναι κενό. Σε κάθε βήμα προστίθεται στο δάσος Δ μία ακμή με την προσθήκη της οποίας το Δ παραμένει υπογράφημα ενός ΕΣΔ. Καλούμε αυτές τις ακμές *επαυξάνουσες* ή *ακμές επαύξησης* για το Δ. Ο αλγόριθμος ολοκληρώνεται όταν το Δ αποκτήσει $n-1$ ακμές, οπότε αποτελεί ένα ΕΣΔ. Ακολουθεί μια διατύπωση της παραπάνω γενικής στρατηγικής σε ψευδοκώδικα.

```
MST(G(V, E, w))
    Δ ← ∅;
    while |Δ| < n − 1 do
        Υπολόγισε μια ακμή επαύξησης e για το Δ;
        Δ ← Δ ∪ {e};
    return(Δ);
```

Μια ακμή e είναι επαυξάνουσα για το Δ όταν:

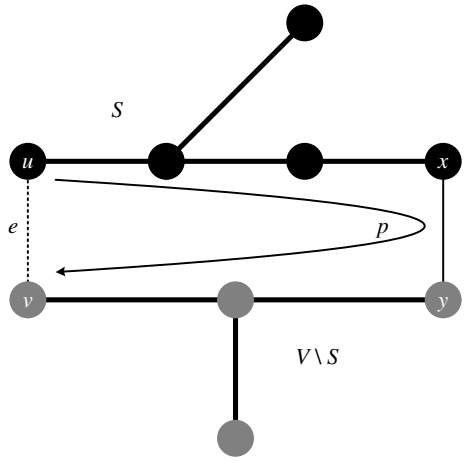
1. Αν το Δ είναι δάσος, το $\Delta \cup \{e\}$ παραμένει δάσος (δηλαδή παραμένει ακυκλικό), και
2. Αν το Δ είναι υπογράφημα ενός ΕΣΔ, το $\Delta \cup \{e\}$ παραμένει υπογράφημα ενός ΕΣΔ.

Με βάση τον ορισμό των ακμών επαύξησης του Δ, είναι φανερό ότι ο παραπάνω γενικός αλγόριθμος υπολογίζει ένα ΕΣΔ του G. Αρχικά το γράφημα χωρίς ακμές αποτελεί υπογράφημα κάθε ΕΣΔ του G. Αν σε κάποιο βήμα το Δ αποτελεί υπογράφημα ενός ΕΣΔ (επαγωγική υπόθεση), το $\Delta \cup \{e\}$ παραμένει υπογράφημα ενός ΕΣΔ γιατί η e είναι ακμή επαύξησης του Δ. Όταν το Δ αποκτήσει $|V|-1$ ακμές, είναι μεγιστικά ακυκλικό (βλ. Θεώρημα 5.1) και δεν μπορεί να επαυξηθεί περαιτέρω. Επομένως, το Δ ταυτίζεται με κάποιο ΕΣΔ του Γ.

Απομένει να δείξουμε πως υπολογίζουμε αποδοτικά μια ακμή επαύξησης του Δ σε κάθε βήμα του αλγόριθμου. Η ιδέα είναι να βρίσκουμε μια τομή την οποία δεν διασχίζει το Δ και να συμπληρώνουμε το Δ με μια ακμή ελάχιστου βάρους που διασχίζει αυτή την τομή. Οι ιδιότητες των βέλτιστων επιμέρους λύσεων και της άπληστης επιλογής οδηγούν στο συμπέρασμα ότι αυτή είναι όντως μια ακμή επαύξησης του Δ.

Θεώρημα 5.2. *Έστω $G(V, E, w)$ συνεκτικό γράφημα με βάρη στις ακμές, έστω $(S, V \setminus S)$ μια τομή του G, και έστω Δ ένα υπογράφημα ενός ΕΣΔ που δεν διασχίζει την τομή $(S, V \setminus S)$. Κάθε ακμή ελάχιστου βάρους του συνόλου τομής $\delta(S, V \setminus S)$ αποτελεί ακμή επαύξησης του Δ.*

[3]Υπενθυμίζουμε ότι όλα τα συνδετικά δέντρα ενός γραφήματος n κορυφές έχουν $n-1$ ακμές. Κάθε υπογράφημα ενός συνδετικού δέντρου είναι μη-συνεκτικό (άρα δάσος) ενόσω έχει λιγότερες από $n-1$ ακμές και γίνεται συνεκτικό (άρα δέντρο) όταν αποκτήσει $n-1$ ακμές.



Σχήμα 5.10: Για κάθε τομή $(S, V \setminus S)$ και κάθε ακμή ελάχιστου βάρους $e = \{u, v\}$ του αντίστοιχου συνόλου τομής $\delta(S, V \setminus S)$, υπάρχει ένα ΕΣΔ που περιέχει την e.

Απόδειξη. Έστω T ένα ΕΣΔ του G του οποίο το Δ είναι υπογράφημα, και έστω $e = \{u, v\}$ μια ακμή ελάχιστου βάρους του συνόλου τομής $\delta(S, V \setminus S)$. Αν το $\Delta \cup \{e\}$ παραμένει υπογράφημα του T, η ακμή e είναι όντως μια ακμή επαύξησης του Δ. Ας υποθέσουμε λοιπόν ότι η e δεν ανήκει στο T και το $\Delta \cup \{e\}$ δεν αποτελεί υπογράφημα του T. Σε αυτή την περίπτωση, θα κατασκευάσουμε ένα άλλο ΕΣΔ T' του οποίου το $\Delta \cup \{e\}$ αποτελεί υπογράφημα.

Έστω p το μονοπάτι σύνδεσης στο T των άκρων της ακμής e, δηλαδή των κορυφών u και v. Το $p + e$ αποτελεί κύκλο γιατί έχουμε υποθέσει ότι η e δεν ανήκει στο T. Αφού το ένα άκρο της e ανήκει στο S και το άλλο ανήκει στο $V \setminus S$, το p πρέπει να "διασχίζει" την τομή $(S, V \setminus S)$ (βλ. επίσης Θεώρημα B.2). Θα πρέπει λοιπόν να υπάρχει μια ακμή του p που ανήκει στο $\delta(S, V \setminus S)$. Έστω $e' = \{x, y\} \in \delta(S, V \setminus S)$ αυτή η ακμή (βλ. Σχήμα 5.10).

Το $T' = (T + e) \setminus e'$ αποτελεί ένα συνδετικό δέντρο του G επειδή η αφαίρεση της e' από το $T + e$ αφαιρεί τον κύκλο $p + e$ χωρίς να επηρεάζει τη συνεκτικότητα. Το Δ είναι υπογράφημα του $T \setminus e'$ γιατί έχουμε υποθέσει ότι το Δ είναι υπογράφημα του T και δεν διασχίζει την τομή $(S, V \setminus S)$. Επομένως, το $\Delta \cup \{e\}$ αποτελεί υπογράφημα του T'.

Αφού υποθέσαμε ότι η e είναι μια ακμή ελάχιστου βάρους του συνόλου τομής $\delta(S, V \setminus S)$ και ότι $e' \in \delta(S, V \setminus S)$, το βάρος της e είναι μικρότερο ή ίσο του βάρους της e'. Συνεπώς, το συνολικό βάρος του T' δεν ξεπερνά το συνολικό βάρος του T:

$$w(T') = w(T) + w(e) - w(e') \leq w(T)$$

όπου χρησιμοποιήσαμε το γεγονός ότι $w(e) \leq w(e')$. Επομένως, το T' αποτελεί ένα ΕΣΔ του G.

Δείξαμε λοιπόν ότι το $\Delta \cup \{e\}$ αποτελεί υπογράφημα ενός ΕΣΔ του G. Επομένως, κάθε ακμή ελάχιστου βάρους του συνόλου τομής $\delta(S, V \setminus S)$ αποτελεί μια ακμή επαύξησης του Δ. □

Άσκηση 5.17. Να σχολιάσετε την ορθότητα της ακόλουθης πρότασης: Το ΕΣΔ ενός γραφήματος $G(V, E, w)$ δεν μπορεί να περιέχει την ακμή μέγιστου βάρους του G.

Λύση. Η πρόταση δεν είναι αληθής. Για παράδειγμα, το ΕΣΔ ενός δέντρου περιλαμβάνει όλες τις ακμές του. □

Άσκηση 5.18. Έστω γράφημα $G(V, E, w)$, και έστω e μία ακμή μέγιστου βάρους που ανήκει σε κύκλο. Να αποδείξετε ότι υπάρχει ένα ΕΣΔ του γραφήματος G το οποίο είναι ΕΣΔ και για το υπογράφημα γράφημα $G'(V, E \setminus \{e\}, w)$ που προκύπτει από την αφαίρεση της e.

Λύση. Παρατηρούμε ότι κάθε ΕΣΔ του G που δεν περιέχει την e είναι ένα ΕΣΔ και για το γράφημα $G(V, E \setminus e)$. Επομένως, αρκεί να δείξουμε ότι υπάρχει ένα ΕΣΔ του G που δεν περιέχει την e.

Έστω T ένα ΕΣΔ του G το οποίο περιλαμβάνει την ακμή e. Το $T \setminus e$ ορίζει μία τομή. Έστω $(S, V \setminus S)$ τα αντίστοιχα σύνολα κορυφών. Επειδή η e ανήκει σε κύκλο, πρέπει να υπάρχει μια ακόμη ακμή e' που διασχίζει την ίδια τομή. Αφού η e είναι ακμή μέγιστου βάρους, $w(e') \leq w(e)$. Επομένως, το $T' = (T \setminus e) + e'$ είναι επίσης ένα ΕΣΔ του G και δεν περιέχει την e. □

Άσκηση 5.19. Να σχολιάσετε την ορθότητα της ακόλουθης πρότασης: Έστω δισυνεκτικό γράφημα $G(V, E)$, και έστω $T(V, E_T)$ ένα συνδετικό δέντρο του G. Το υπογράφημα $G'(V, E \setminus E_T)$ που προκύπτει από την αφαίρεση των ακμών του E_T είναι συνεκτικό.

Λύση. Η πρόταση δεν είναι αληθής. Για παράδειγμα, ένα απλός κύκλος είναι δισυνεκτικό γράφημα, αλλά αν αφαιρέσουμε τις ακμές ενός συνδετικό δέντρου, το γράφημα που προκύπτει δεν είναι συνεκτικό. □

Άσκηση 5.20. Για κάθε συνδετικό δέντρο $T(V, E_T)$ ενός γραφήματος $G(V, E, w)$ με βάρη στις ακμές, συμβολίζουμε με $W_{\max}(T)$ το βάρος της βαρύτερης ακμής του T, δηλ. $W_{\max}(T) = \max\{w(e) : e \in E_T\}$. Να αποδείξετε ότι για κάθε ΕΣΔ T και κάθε άλλο συνδετικό δέντρο T' του G, ισχύει ότι $W_{\max}(T) \leq W_{\max}(T')$. Με άλλα λόγια, το ΕΣΔ αποτελεί τη βέλτιστη λύση και στο πρόβλημα υπολογισμού ενός συνδετικού δέντρου του G με την ελαφρύτερη ακμή μέγιστου βάρους (bottleneck ή minimax πρόβλημα).

Λύση. Έστω T ένα ΕΣΔ, και έστω $e^* = \{u, v\}$ η βαρύτερη ακμή του T. Ισχύει ότι $w(e^*) = W_{\max}(T)$. Θεωρούμε την τομή $(S, V \setminus S)$ που προκύπτει από την αφαίρεση της e^* από το T. Επειδή το T είναι ένα ΕΣΔ, η e^* είναι μια ακμή ελάχιστου βάρους που διασχίζει την τομή $(S, V \setminus S)$. Αφού κάθε άλλο συνδετικό δέντρο T' επίσης διασχίζει την τομή $(S, V \setminus S)$, πρέπει να περιέχει ακμή με βάρος μεγαλύτερο ή ίσο από $w(e^*) = W_{\max}(T)$. Επομένως, για κάθε συνδετικό δέντρο T' του G, ισχύει ότι $W_{\max}(T') \geq W_{\max}(T)$. □

5.4.2 Ο Αλγόριθμος του Kruskal

Ο αλγόριθμος του Kruskal υπολογίζει ένα ΕΣΔ ακολουθώντας την άπληστη στρατηγική που περιγράφηκε στην προηγούμενη ενότητα. Ο αλγόριθμος εξετάζει τις ακμές σε αύξουσα σειρά βάρους και προσθέτει στο Δ κάθε ακμή που δεν σχηματίζει κύκλο με τις ήδη υπαρχουσες (βλ. Σχήμα 5.12).

Για την υλοποίηση του αλγόριθμου του Kruskal (βλ. Σχήμα 5.11) χρησιμοποιούμε έναν αλγόριθμο ταξινόμησης των ακμών σε αύξουσα σειρά βάρους και μια δομή δεδομένων για διαχείριση ξένων συνόλων (πρόβλημα ένωσης-εύρεσης, union-find). Οι βασικές λειτουργίες μιας δομής για το πρόβλημα ένωσης-εύρεσης είναι η makeSet(v), η οποία δημιουργεί ένα σύνολο με μοναδικό στοιχείο την κορυφή v, η findSet(v), η οποία επιστρέφει το σύνολο στο οποίο ανήκει η v, και η

```
MST-Kruskal(G(V, E, w))
    Ταξινόμησε τις ακμές σε αύξουσα σειρά βάρους, w(e_1) ≤ ··· ≤ w(e_m).
    for all v ∈ V do makeSet(v);
    Δ ← ∅; i ← 1;
    while |Δ| < n − 1 and i ≤ m do
        Έστω e_i = {u, v}.
        if findSet(u) ≠ findSet(v) then
            Δ ← Δ ∪ {e_i}; union(u, v);
        i ← i + 1;
    return(Δ);
```

Σχήμα 5.11: Μια υλοποίηση του αλγόριθμου του Kruskal σε ψευδοκώδικα. Η υλοποίηση βασίζεται σε μια δομή δεδομένων για τη διαχείριση ξένων συνόλων. Η επόμενη ακμή προστίθεται στο δέντρο αν τα άκρα της ανήκουν σε διαφορετικά σύνολα / συνεκτικές συνιστώσες. Η προσθήκη της επιφέρει τη συνένωση των αντίστοιχων συνεκτικών συνιστωσών σε μία. Μετά από την προσθήκη $n-1$ ακμών, υπάρχει μόνο μία συνεκτική συνιστώσα και ο αλγόριθμος ολοκληρώνεται.

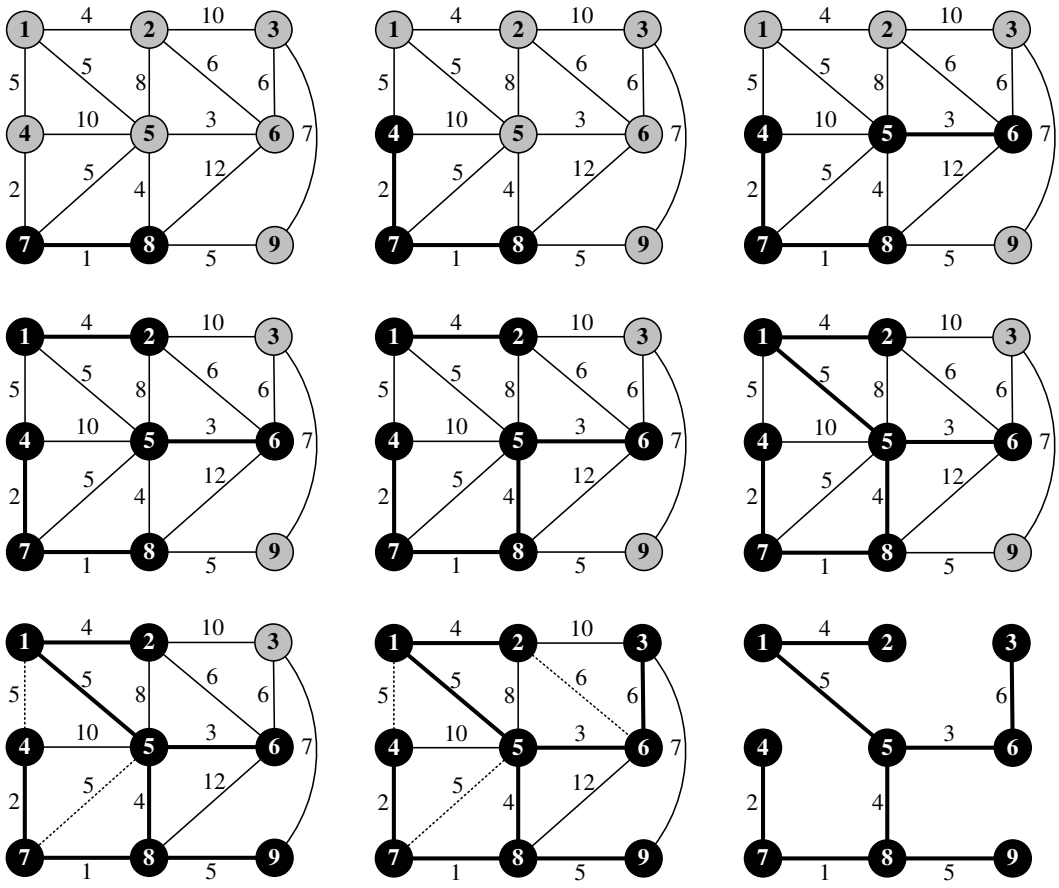
union(u, v), η οποία αντικαθιστά τα σύνολα στα οποία ανήκουν οι u και v με την ένωσή τους. Τα σύνολα κορυφών αντιστοιχούν στις συνεκτικές συνιστώσες που σχηματίζονται από τις ακμές του Δ. Αρχικά το Δ είναι κενό και κάθε κορυφή βρίσκεται απομονωμένη σε μια διαφορετική συνεκτική συνιστώσα / σύνολο. Μια ακμή δεν δημιουργεί κύκλο (άρα προστίθεται στο Δ) μόνο αν τα άκρα της βρίσκονται σε διαφορετικές συνεκτικές συνιστώσες. Η προσθήκη μιας ακμής προκαλεί την ένωση δύο συνεκτικών συνιστωσών σε μία.

Ο αλγόριθμος μπορεί να υλοποιηθεί ώστε να έχει χρόνο εκτέλεσης χειρότερης περίπτωσης $\Theta(m \log m)$. Η ταξινόμηση των ακμών σε αύξουσα σειρά βάρους μπορεί να γίνει σε χρόνο $\Theta(m \log m)$. Η αρχικοποίηση των συνεκτικών συνιστωσών χρειάζεται χρόνο $\Theta(n)$ για τις κλήσεις makeSet. Στη φάση της δημιουργίας του δέντρου, απαιτούνται $O(m)$ λειτουργίες εύρεσης (κλήσεις findSet) και $\Theta(n)$ λειτουργίες ένωσης (κλήσεις union). Αυτές οι λειτουργίες μπορούν να εκτελεσθούν σε συνολικό χρόνο $\Theta(m \log m)$.

Όσον αφορά στην ορθότητα του αλγόριθμου, παρατηρούμε ότι το $\Delta \cup \{e_i\}$ δεν έχει κύκλο αν και μόνο αν η ακμή e_i διασχίζει μια τομή την οποία δεν διασχίζει το Δ. Αφού εξετάζουμε τις ακμές σε αύξουσα σειρά βάρους, η ακμή e_i έχει το ελάχιστο βάρος από όλες τις ακμές που διασχίζουν τη συγκεκριμένη τομή. Σύμφωνα με το Θεώρημα 5.2, κάθε ακμή που προστίθεται είναι ακμή επαύξησης για το Δ και ο αλγόριθμος υπολογίζει ένα ΕΣΔ.

Άσκηση 5.21. Δίνεται ένα γράφημα $G(V, E, w)$ με βάρη στις ακμές και ένα ΕΣΔ T του G. Να διατυπώσετε έναν αποδοτικό αλγόριθμο για τον υπολογισμό ενός ΕΣΔ για το γράφημα $G(V, E, w/2)$ που προκύπτει υποδιπλασιάζοντας τα βάρη των ακμών.

Λύση. Το T αποτελεί ένα ΕΣΔ για το γράφημα G' γιατί το ΕΣΔ του G' έχει βάρος $w(T)/2$. Για να αποδείξουμε τον ισχυρισμό μας, θεωρούμε τον αλγόριθμο του Kruskal με μια συγκεκριμένη ταξινόμηση των ακμών του G σε αύξουσα σειρά βαρών. Το αποτέλεσμα είναι ένα ΕΣΔ με συνολικό βάρος $w(T)$. Ο υποδιπλασιασμός των βαρών των ακμών δεν επηρεάζει τη σειρά τους



Σχήμα 5.12: Παράδειγμα εφαρμογής του αλγόριθμου του Kruskal. Οι ακμές που έχουν συμπεριληφθεί στο δέντρο σημειώνονται παχύτερες και οι ακμές που έχουν εξεταστεί χωρίς να συμπεριληφθούν στο δέντρο σημειώνονται διακεκομμένες. Οι κορυφές που εφάπτονται με ακμή του δέντρου είναι μαύρες, ενώ οι υπόλοιπες είναι γκρίζες.

στην ταξινόμηση σε αύξουσα σειρά βάρους. Ο αλγόριθμος του Kruskal με την ίδια ταξινόμηση θα διαλέξει τις ίδιες ακμές που τώρα θα έχουν το μισό συνολικό βάρος. Ο αλγόριθμος απλά επιστρέφει το T. □

5.4.3 Ο Αλγόριθμος του Prim

Ο αλγόριθμος του Prim διατηρεί ένα δέντρο που καλύπτει ένα υποσύνολο των κορυφών. Ο αλγόριθμος ξεκινάει με μια οποιαδήποτε κορυφή και κενό σύνολο ακμών. Σε κάθε επανάληψη, ο αλγόριθμος προσθέτει στο δέντρο την ελαφρύτερη ακμή που συνδέει μια κορυφή που καλύπτεται με μια κορυφή που δεν καλύπτεται από το δέντρο. Ο αλγόριθμος τερματίζει έπειτα από $n - 1$ επαναλήψεις (βλ. Σχήμα 5.14).

Η υλοποίηση του αλγορίθμου (βλ. Σχήμα 5.13) διατηρεί το σύνολο S με τις κορυφές του δέντρου και το κόστος ένταξης $c[v]$ για κάθε κορυφή $v \notin S$. Η υλοποίηση διατηρεί επίσης την κορυφή $p[v]$ η οποία έχει ενταχθεί στο δέντρο και συνδέεται με τη v με ακμή βάρους $c[v]$ (δηλ. ισχύει ότι $\{p[v], v\} \in E$ και $w(p[v], v) = c[v]$). Σε κάθε επανάληψη, η κορυφή $v \notin S$ με το

```
MST-Prim(G(V, E, w))
    for all v ∈ V do
        c[v] ← ∞; p[v] ← NIL;
    Έστω r οποιαδήποτε κορυφή του V.
    c[r] ← 0;  S ← ∅;  Δ ← ∅;
    while |S| < n do
        Έστω v κορυφή με c[v] = min_{u∉S}{c[u]};
        if p[v] ≠ NIL then
            Δ ← Δ ∪ {v, p[v]};
        S ← S ∪ {v};
        for all u ∈ L[v] do
            if u ∉ S and w(v, u) < c[u] then
                c[u] ← w(v, u); p[u] ← v;
    return(Δ);
```

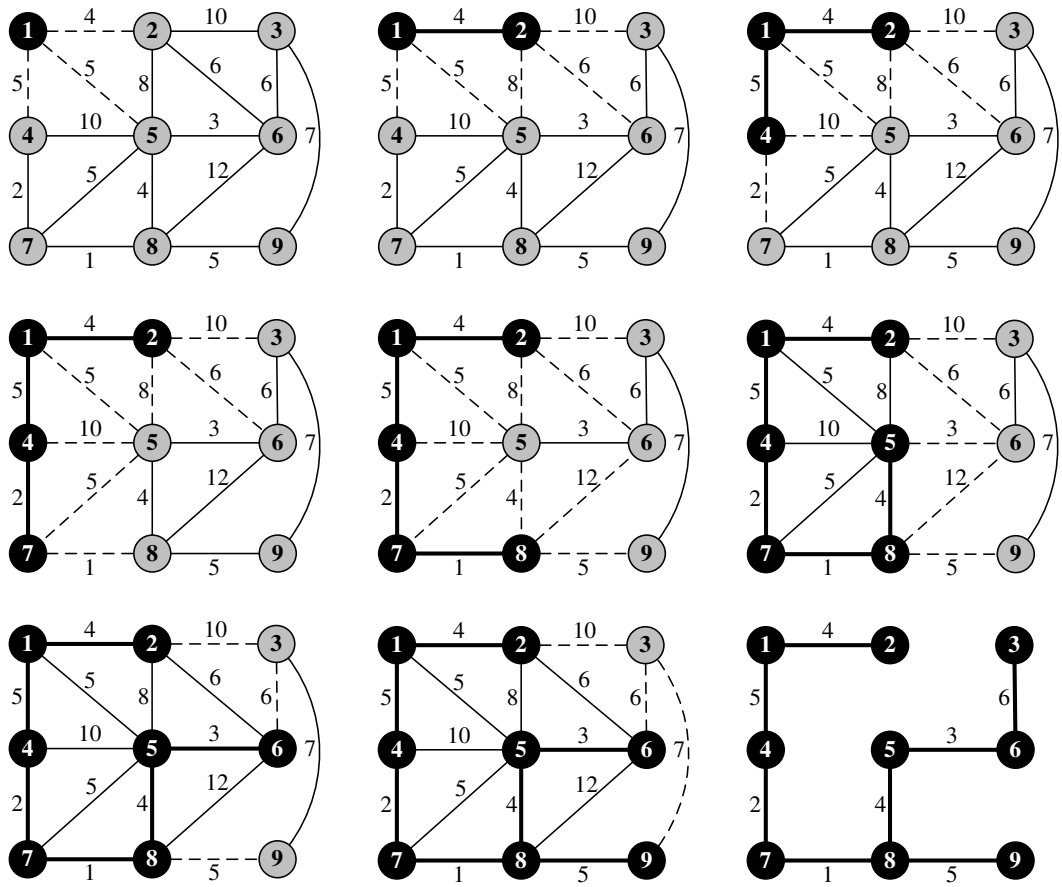
Σχήμα 5.13: Μια υλοποίηση του αλγόριθμου του Prim σε ψευδοκώδικα. Οι κορυφές που έχουν ενταχθεί στο δέντρο αποτελούν το σύνολο S. Σε κάθε επανάληψη η κορυφή $v \notin S$ με το μικρότερο $c[v]$ εντάσσεται στο δέντρο. Σαν αποτέλεσμα, το κόστος ένταξης των εκτός δέντρου γειτόνων της v αναπροσαρμόζεται.

ελάχιστο κόστος ένταξης $c[v]$ και η ακμή $\{p[v], v\}$ προστίθενται στο δέντρο,.

Για την αποδοτικότερη υλοποίηση, οι κορυφές μπορούν να οργανωθούν σε σε μια ουρά προτεραιότητας με βάση το κόστος ένταξης τους στο δέντρο. Αν η ουρά προτεραιότητας υλοποιηθεί με δυαδικό σωρό (binary heap), ο αλγόριθμος του Prim έχει χρόνο εκτέλεσης $O(m \log n)$, δηλ. ασυμπτωτικά ίδιο με το χρόνο εκτέλεσης του αλγόριθμου του Kruskal. Συγκεκριμένα, η αρχικοποίηση των πινάκων c και p και η αρχικοποίηση της ουράς χρειάζονται χρόνο $\Theta(n)$. Εκτελούνται $n-1$ εξαγωγές ελάχιστου στοιχείου, μια για κάθε κορυφή εκτός της αρχικής, σε συνολικό χρόνο $O(n \log n)$. Μετά την ένταξη μιας νέας κορυφής στο δέντρο, γίνεται ενημέρωση του κόστους ένταξης των γειτονικών της κορυφών με συνακόλουθη αναδιοργάνωση της ουράς προτεραιότητας. Έχουμε το πολύ μία ενημέρωση και αναδιοργάνωση της ουράς για κάθε ακμή του γραφήματος. Επομένως ο συνολικός χρόνος είναι $O(m \log n)$. Μια πιο αποδοτική υλοποίηση του αλγόριθμου του Prim χρησιμοποιεί σωρό Fibonacci για την ουρά προτεραιότητας και έχει χρόνο εκτέλεσης $O(m + n \log n)$. Αυτός ο χρόνος είναι ασυμπτωτικά μικρότερος από το χρόνο εκτέλεσης του αλγόριθμου του Kruskal για πυκνά γραφήματα.

Για την ορθότητα του αλγόριθμου του Prim, παρατηρούμε ότι η ακμή που προστίθεται σε κάθε επανάληψη είναι μια ακμή ελάχιστου βάρους που διασχίζει την τομή $(S, V \setminus S)$, δηλαδή την τομή που δημιουργείται από τις κορυφές εντός και εκτός δέντρου. Αφού το Δ δεν διασχίζει αυτή την τομή, η ακμή που προστίθεται είναι ακμή επαύξησης για το Δ (βλ. Θεώρημα 5.2) και ο αλγόριθμος υπολογίζει ένα ΕΣΔ.

Άσκηση 5.22. Έστω $G(V, E, w)$ ένα απλό γράφημα με διαφορετικά βάρη στις ακμές, και έστω e^* η (μοναδική) ακμή με το ελάχιστο βάρος (αφού όλα τα βάρη είναι διαφορετικά, ισχύει ότι $\forall e \in E \setminus \{e^*\}, w(e^*) < w(e)$. Να αποδείξετε ότι κάθε ΕΣΔ του G περιέχει την e^*. Τι ισχύει



Σχήμα 5.14: Παράδειγμα εφαρμογής του αλγόριθμου του Prim με αρχική κορυφή την 1. Σε κάθε βήμα, οι κορυφές που έχουν ενταχθεί στο δέντρο είναι μαύρες και οι κορυφές που δεν έχουν ουν ενταχθεί στο δέντρο γκρίζες. Οι διακεκομμένες ακμές διασχίζουν την τομή που σχηματίζεται από τις κορυφές εντός και εκτός δέντρου. Ο αλγόριθμος επιλέγει την ελαφρύτερη διακεκομμένη ακμή και την προσθέτει στο δέντρο μάζί με την αντίστοιχη κορυφή εκτός δέντρου. Οι ακμές που έχουν συμπεριληφθεί στο δέντρο σημειώνονται παχύτερες.

για τη δεύτερη ελαφρύτερη και την τρίτη ελαφρύτερη ακμή; Μπορείτε να καταλήξετε σε ένα γενικότερο συμπέρασμα;

Λύση. Η απόδειξη είναι με απαγωγή σε άτοπο. Έστω T ένα ΕΣΔ του G που δεν περιέχει την e^*. Η προσθήκη της e^* στο T δημιουργεί ακριβώς ένα κύκλο. Έστω e μια οποιαδήποτε ακμή αυτού του κύκλου. Η αφαίρεσή της "σπάει" τον κύκλο δεν επηρεάζει τη συνεκτικότητα. Συνεπώς, το $(T + e^*) - e$ αποτελεί ένα συνδετικό δέντρο του G. Όμως είναι $w(e) > w(e^*)$, επειδή η e^* είναι η μοναδική ακμή ελάχιστου βάρους, και το $(T + e^*) - e$ έχει μικρότερο συνολικό βάρος από το T. Αυτό αποτελεί αντίφαση στο γεγονός ότι το T είναι ένα ΕΣΔ.

Επειδή η προσθήκη μιας ακμής σε ένα συνδετικό δέντρο ενός απλού γραφήματος δημιουργεί κύκλο μήκους τουλάχιστον 3, με το ίδιο επιχείρημα αποδεικνύουμε ότι και η δεύτερη ελαφρύτερη ακμή ανήκει σε κάθε ΕΣΔ του G. Η τρίτη ελαφρύτερη ακμή δεν συμπεριλαμβάνεται σε κανένα ΕΣΔ του G αν σχηματίζει κύκλο μήκους 3 με την ελαφρύτερη ακμή και τη δεύτερη ελαφρύτερη

ακμή (π.χ. θεωρούμε ένα τρίγωνο με βάρη ακμών 1, 2, και 3).

Γενικότερα, αν το μήκος του συντομότερου κύκλου στο γράφημα G είναι g, τότε αποδεικνύεται ότι οι $g-1$ ελαφρύτερες ακμές συμπεριλαμβάνονται σε κάθε ΕΣΔ του G. Ένας τρόπος απόδειξης είναι αυτός που εφαρμόσαμε για την ελαφρύτερη ακμή e^*. Ένας εναλλακτικός τρόπος απόδειξης είναι να χρησιμοποιήσουμε το γεγονός ότι το ΕΣΔ ενός γραφήματος με διαφορετικά βάρη στις ακμές είναι μοναδικό (βλ. Άσκηση 5.23). Στη συνέχεια παρατηρούμε ότι ο αλγόριθμος του Kruskal θα επιλέξει όλες τις $g-1$ ελαφρύτερες ακμές αφού δεν σχηματίζουν κύκλο. □

Άσκηση 5.23. Έστω $G(V, E, w)$ ένα απλό γράφημα με διαφορετικά βάρη στις ακμές. Να αποδείξετε ότι το ΕΣΔ του G είναι μοναδικό (δηλ. αν T ένα ΕΣΔ, για κάθε άλλο συνδετικό δέντρο T', είναι $w(T') > w(T)$).

Λύση. Έστω ότι υπήρχαν δύο ΕΣΔ T και T' για το γράφημα G. Προφανώς, αυτά θα είχαν το ίδιο βάρος $w(T) = w(T')$. Αφού το T και το T' είναι διαφορετικά, υπάρχει ακμή $e' \in T' \setminus T$. Έστω $(S, V \setminus S)$ η τομή που προκύπτει από την αφαίρεση της e' από το T'. Αφού η $e' \notin T$, υπάρχει μια διαφορετικά ακμή $e \in T$ που διασχίζει την τομή $(S, V \setminus S)$. Αφού όλα τα βάρη είναι διαφορετικά, έστω ότι $w(e) < w(e')$ (αν $w(e) > w(e')$ απλώς αλλάζουμε τις ονομασίες). Τότε, το γράφημα $(T' \setminus e') + e$ είναι ένα συνδετικό δέντρο του G με συνολικό βάρος μικρότερο από το $w(T')$. Αυτό αποτελεί αντίφαση στην υπόθεση ότι το T' είναι ένα ΕΣΔ. □

Άσκηση 5.24. Δίνεται ένα γράφημα $G(V, E, w)$ με βάρη στις ακμές και ένα ΕΣΔ T του G. Έστω G' το γράφημα που προκύπτει από το G με την προσθήκη μιας νέας ακμής e' με βάρος $w(e')$. Να διατυπώσετε έναν αλγόριθμο με χρόνο εκτέλεσης $O(n)$ που υπολογίζει ένα ΕΣΔ για το G' (δεδομένου του T).

Λύση. Ο αλγόριθμος εισάγει τη νέα ακμή στο T. Αυτό δημιουργεί ένα κύκλο C γιατί η e' δεν ανήκει στο αρχικό γράφημα και άρα δεν ανήκει στο T. Ο αλγόριθμος εξετάζει όλες τις ακμές του C και αφαιρεί τη βαρύτερη. Το αποτέλεσμα είναι ένα ΕΣΔ για το G' και ο χρόνος εκτέλεσης είναι $O(n)$, αφού n είναι το μεγαλύτερο μήκος που μπορεί να έχει ο κύκλος. □

5.5 Συντομότερα Μονοπάτια

Στο πρόβλημα του υπολογισμού των *συντομότερων μονοπατιών*, θεωρούμε ένα κατευθυνόμενο γράφημα $G(V, E, w)$ με βάρη στις ακμές. Το βάρος κάθε ακμής δίνεται από τη συνάρτηση $w : E \mapsto \mathbb{R}$ και μπορεί να είναι αρνητικό ή μηδενικό. Ανάλογα με την συγκεκριμένη εφαρμογή, τα βάρη των ακμών μπορεί να αντιπροσωπεύουν απόσταση, καθυστέρηση, κόστος, κλπ. Στο εξής, θα θεωρούμε ότι η ποσότητα $w(e)$ αντιπροσωπεύει το *μήκος* της ακμής $e \in E$ και ότι το συνολικό μήκος ενός συντομότερου μονοπατιού αντιπροσωπεύει την απόσταση μεταξύ των άκρων του.

Σε ένα γράφημα $G(V, E, w)$ με βάρη / μήκη στις ακμές, το μήκος μιας (πεπερασμένης) διαδρομής $p = (v_0, v_1, \ldots, v_k)$ είναι ίσο με το άθροισμα των μηκών των ακμών που περιλαμβάνονται στη διαδρομή. Τυπικά, $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$. Η *απόσταση* $d(u, v)$ της κορυφής v από την κορυφή u είναι το μήκος της συντομότερης διαδρομής από τη u στη v. Θεωρούμε ότι η απόσταση

$d(u, v) = \infty$ αν η κορυφή v δεν είναι προσπελάσιμη από τη u. Ένα *συντομότερο μονοπάτι* (shortest path) από τη u στη v είναι ένα μονοπάτι με μήκος ίσο με την απόσταση $d(u, v)$.

Υπάρχει πεπερασμένη συντομότερη διαδρομή[4] μεταξύ δύο κορυφών αν και μόνο αν υπάρχει μονοπάτι μεταξύ τους που έχει το ίδιο (ελάχιστο) μήκος (βλ. επίσης Άσκηση 5.2). Αυτός είναι ο λόγος που αναφερόμαστε πάντα σε συντομότερα μονοπάτια και εξαιρετικά σπάνια σε συντομότερες διαδρομές. Χάριν συντομίας, μερικές φορές χρησιμοποιούμε τον όρο $s - t$ μονοπάτι για να αναφερθούμε σε ένα μονοπάτι από μια κορυφή s σε μια κορυφή t.

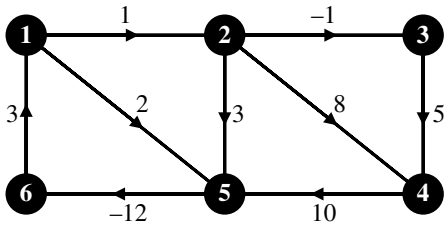
Στο Πρόβλημα των Συντομότερων Μονοπατιών (Shortest Paths Problem - SPP), δίνεται ένα γράφημα $G(V, E, w)$ με μήκη στις ακμές και ζητούνται τα συντομότερα μονοπάτια μεταξύ κάποιων κορυφών. Σε μερικές περιπτώσεις, το ζητούμενο είναι ο υπολογισμός των αντίστοιχων αποστάσεων και όχι τα ίδια τα μονοπάτια. Ανάλογα με τις κορυφές για τις οποίες πρέπει να υπολογίσουμε τα συντομότερα μονοπάτια, ορίζουμε διαφορετικές παραλλαγές του προβλήματος. Το Πρόβλημα των Συντομότερων Μονοπατιών έχει πολλές και σημαντικές πρακτικές εφαρμογές.

Στη βασική παραλλαγή του προβλήματος, δίνονται ένα γράφημα $G(V, E, w)$, μια αρχική κορυφή $s \in V$, και μια τελική κορυφή $t \in V$, και ζητείται ένα συντομότερο μονοπάτι από την s στην t. Αυτή η παραλλαγή είναι γνωστή σαν Πρόβλημα Συντομότερου Μονοπατιού Ζεύγους Κορυφών (Single-Pair Shortest Path Problem).

Στην πιο γνωστή παραλλαγή, δίνονται ένα γράφημα $G(V, E, w)$ και μία αρχική κορυφή $s \in V$, και ζητούνται τα συντομότερα μονοπάτια από την s προς όλες τις υπόλοιπες κορυφές του γραφήματος. Αυτή η παραλλαγή είναι γνωστή σαν Πρόβλημα Συντομότερων Μονοπατιών από μία Αρχική Κορυφή (Single-Source Shortest Paths Problem). Θα παρουσιάσουμε δύο διαφορετικούς αλγόριθμους για αυτή την παραλλαγή: τον αλγόριθμο δυναμικού προγραμματισμού των Bellman-Ford και τον άπληστο αλγόριθμο του Dijkstra. Αξίζει να σημειωθεί ότι οι αλγόριθμοι που υπολογίζουν τα συντομότερα μονοπάτια από μία αρχική κορυφή χρησιμοποιούνται και για τον υπολογισμό του Συντομότερου Μονοπατιού Ζεύγους Κορυφών. Αν και η παραλλαγή με ένα ζεύγος κορυφών αποτελεί ειδική περίπτωση της παραλλαγής με αρχική κορυφή, δεν είναι γνωστός κανένας γρηγορότερος αλγόριθμος για την πρώτη.

Σε μια πιο γενική παραλλαγή του προβλήματος, δίνεται ένα γράφημα $G(V, E, w)$ και ζητούνται τα συντομότερα μονοπάτια μεταξύ όλων των ζευγαριών κορυφών. Αυτή η παραλλαγή είναι γνωστή σαν Πρόβλημα Συντομότερων Μονοπατιών για όλα τα Ζεύγη Κορυφών (All-Pairs Shortest Paths Problem) και μπορεί να λυθεί εφαρμόζοντας τους προηγούμενους αλγόριθμους $|V|$ φορές, μια για κάθε διαφορετική αρχική κορυφή. Όμως υπάρχουν και γρηγορότεροι αλγόριθμοι για αυτή την παραλλαγή. Από αυτούς, θα παρουσιάσουμε τον αλγόριθμο των Floyd-Warshall που βασίζεται στη μέθοδο του δυναμικού προγραμματισμού.

[4]Σε μια διαδρομή που δεν είναι μονοπάτι, το τμήμα μεταξύ δύο διαδοχικών επαναλήψεων μιας κορυφής αποτελεί ένα κύκλωμα. Αν το συνολικό μήκος του κυκλώματος είναι μη-αρνητικό, η αφαίρεσή του δεν αυξάνει το μήκος της διαδρομής. Από την αφαίρεση όλων των διαδοχικών επαναλήψεων κάθε κορυφής προκύπτει ένα μονοπάτι με μήκος μικρότερο ή ίσο από το μήκος της αρχικής διαδρομής. Αν το συνολικό μήκος του κυκλώματος είναι αρνητικό, κάθε επανάληψή του μειώνει ακόμη περισσότερο το μήκος της διαδρομής. Τότε δεν μπορεί να οριστεί η έννοια της συντομότερης διαδρομής και της απόστασης μεταξύ των άκρων της.



Σχήμα 5.15: Παράδειγμα γραφήματος με κύκλους αρνητικού μήκους. Ο κύκλος $(1, 2, 5, 6, 1)$ έχει μήκος -5, και ο κύκλος $(1, 5, 6, 1)$ έχει μήκος -7. Έτσι δεν ορίζονται οι αποστάσεις των κορυφών από την 1 αλλά και από τις υπόλοιπες κορυφές γιατί η 1 είναι προσπελάσιμη από όλες τις κορυφές.

5.5.1 Ακμές και Κύκλοι Αρνητικού Μήκους

Αν όλα τα μήκη των ακμών του γραφήματος $G(V, E, w)$ είναι μη-αρνητικά, υπάρχει πάντα ένα συντομότερο μονοπάτι για κάθε ζεύγος κορυφών. Επομένως, για κάθε $u, v \in V$, η απόσταση $d(u, v)$ είναι καλώς ορισμένη και πεπερασμένη. Το ίδιο συμβαίνει και όταν το γράφημα έχει ακμές με αρνητικό μήκος, αλλά δεν έχει κύκλους των οποίων το συνολικό μήκος είναι αρνητικό (βλ. επίσης υποσημείωση 4). Σε αυτή την περίπτωση, η απόσταση μεταξύ κάποιων κορυφών ενδέχεται να είναι αρνητική, αλλά εξακολουθεί να είναι καλώς ορισμένη και πεπερασμένη.

Αντίθετα, αν το γράφημα έχει κύκλο αρνητικού μήκους (π.χ. το γράφημα του Σχήματος 5.15) που είναι προσπελάσιμος από μια αρχική κορυφή s, οι αποστάσεις από την s προς τις κορυφές του κύκλου δεν ορίζονται. Ο λόγος είναι ότι για κάθε διαδρομή p από την s προς μια κορυφή του κύκλου αρνητικού μήκους, μπορούμε να φτιάξουμε μια νέα διαδρομή p' που διαφέρει από την p μόνο γιατί διέρχεται περισσότερες φορές από τον κύκλο. Δηλαδή, πάντα υπάρχει διαδρομή με ακόμη μικρότερο μήκος και επομένως οι έννοιες της συντομότερης διαδρομής και της απόστασης δεν μπορούν να οριστούν. Το ίδιο συμβαίνει και με κάθε κορυφή που είναι προσπελάσιμη από την s μέσω κορυφής που ανήκει σε κύκλο αρνητικού μήκους. Αν για κάποια κορυφή u, υπάρχει μονοπάτι από την s στη u που διέρχεται από κορυφή κύκλου αρνητικού μήκους, η απόσταση $d(s, u)$ δεν ορίζεται και γράφουμε $d(s, u) = -\infty$.

5.5.2 Ιδιότητες Συντομότερων Μονοπατιών

Οι αλγόριθμοι για το Πρόβλημα των Συντομότερων Μονοπατιών εκμεταλλεύονται την ιδιότητα ότι κάθε τμήμα ενός συντομότερου μονοπατιού αποτελεί επίσης συντομότερου μονοπάτι μεταξύ των άκρων του. Με άλλα λόγια, το Πρόβλημα των Συντομότερων Μονοπατιών έχει την ιδιότητα των βέλτιστων επιμέρους λύσεων. Όπως γνωρίζουμε, αυτή η ιδιότητα επιτρέπει την εφαρμογή του δυναμικού προγραμματισμού και της μεθόδου της απληστίας.

Πρόταση 5.5. *Έστω γράφημα $G(V, E, w)$, και έστω ένα συντομότερο μονοπάτι $p = (u_0, u_1, \ldots, u_k)$ από την κορυφή u_0 στην κορυφή u_k. Για κάθε i, j, $0 \leq i < j \leq k$, το τμήμα του p από την κορυφή u_i μέχρι την κορυφή u_j αποτελεί ένα συντομότερο $u_i - u_j$ μονοπάτι.*

Απόδειξη. Έστω p_{ij} το τμήμα του p από μεταξύ των κορυφών u_i και u_j, και έστω ότι αυτό δεν είναι ένα συντομότερο μονοπάτι. Τότε υπάρχει $u_i - u_j$ μονοπάτι p'_{ij} με μήκος $w(p'_{ij}) < w(p_{ij})$.

Αντικαθιστώντας το p_{ij} με το p'_{ij} στο p, παίρνουμε ένα $u_0 - u_k$ μονοπάτι με μήκος μικρότερο από το μήκος του p. Αυτό αποτελεί αντίφαση στο γεγονός ότι το p είναι ένα συντομότερο $u_0 - u_k$ μονοπάτι. □

Μια άμεση συνέπεια της παραπάνω πρότασης αποτελεί το γεγονός ότι οι αποστάσεις των σε ένα γράφημα $G(V, E, w)$ ικανοποιούν την τριγωνική ανισότητα[5]. Συγκεκριμένα, για κάθε $u, v, w \in V$, ισχύει ότι $d(u, v) \leq d(u, w) + d(w, v)$. Δηλαδή, το μήκος του συντομότερου μονοπατιού από τη u στη v δεν υπερβαίνει το μήκος της διαδρομής που προκύπτει από την ένωση των συντομότερων $u - w$ και $w - v$ μονοπατιών. Το ακόλουθο πόρισμα αποτελεί εξειδίκευση αυτής της γενικής παρατήρησης.

Πόρισμα 5.1. *Έστω γράφημα $G(V, E, w)$, και έστω $s \in V$ μία αρχική κορυφή. Για κάθε ακμή $(v, u) \in E$, ισχύει ότι $d(s, u) \leq d(s, v) + w(v, u)$.*

Άσκηση 5.25. Έστω γράφημα $G(V, E, w)$, και έστω p ένα συντομότερο μονοπάτι από την κορυφή s στην κορυφή t. Να δείξετε ότι το p θα παραμείνει συντομότερο μονοπάτι αν τα μήκη όλων των ακμών πολλαπλασιαστούν με τον ίδιο θετικό αριθμό. Τι θα συμβεί αν στα μήκη όλων των ακμών προστεθεί ο ίδιος θετικός αριθμός;

5.5.3 Δέντρο Συντομότερων Μονοπατιών

Τα συντομότερα μονοπάτια ενός γραφήματος $G(V, E, w)$ από μία αρχική κορυφή s αναπαρίστανται από το Δέντρο Συντομότερων Μονοπατιών (Shortest Paths Tree). Πρόκειται για το ελάχιστο (ως προς τον αριθμό των ακμών) υπογράφημα του G που περιέχει ένα συντομότερο μονοπάτι προς κάθε κορυφή u που είναι προσπελάσιμη από την s. Τα αρχικά τμήματα του συντομότερου $s - u$ μονοπατιού αποτελούν επίσης συντομότερα μονοπάτια προς τις αντίστοιχες κορυφές (Πρόταση 5.5). Μπορούμε λοιπόν να έχουμε λοιπόν ένα συντομότερο μονοπάτι προς κάθε κορυφή που είναι προσπελάσιμη από την s και το υπογράφημα είναι δέντρο με ρίζα την s.

Το Δέντρο Συντομότερων Μονοπατιών (ΔΣΜ) αναπαρίσταται με τον πίνακα p με n στοιχεία. Το στοιχείο $p[u]$ δηλώνει τον πατέρα κάθε κορυφής u στο ΔΣΜ. Δηλαδή, το $p[u]$ δηλώνει την τελευταία κορυφή πριν τη u στο συντομότερο $s - u$ μονοπάτι (βλ. επίσης την αναπαράσταση του δέντρου της ΑΠΠ, Ενότητα 5.2, και του δέντρου της ΑΠΒ, Ενότητα 5.3). Δεδομένου του πίνακα p, το Δέντρο Συντομότερων Μονοπατιών είναι το υπογράφημα $G_p(V_p, E_p)$ που ορίζεται ως:

$$V_p = \{v \in V : p[v] \neq \text{NIL}\} \cup \{s\} \quad \text{και} \quad E_p = \{(p[v], v) \in E : v \in V_p \setminus \{s\}\}$$

Άσκηση 5.26. Να δώσετε παράδειγμα γραφήματος $G(V, E, w)$ για το οποίο το Δέντρο Συντομότερων Μονοπατιών με ρίζα μια κορυφή s είναι διαφορετικό από το Ελάχιστο Συνδετικό Δέντρο του G.

[5]Γενικότερα, οι αποστάσεις μεταξύ των κορυφών ενός μη-κατευθυνόμενου γραφήματος με βάρη στις ακμές ορίζουν ένα μετρικό χώρο.

5.6 Συντομότερα Μονοπάτια από μία Αρχική Κορυφή

Ο υπολογισμός των συντομότερων μονοπατιών από μια αρχική κορυφή βασίζεται στο Πόρισμα 5.1. Έστω γράφημα $G(V, E, w)$ και αρχική κορυφή s. Για κάθε κορυφή u, ο αλγόριθμος διατηρεί μια "απαισιόδοξη" εκτίμηση $\ell[u]$ της απόστασης $d(s, u)$. Αρχικά είναι $\ell[s] = 0$ και $\ell[u] = \infty$ για κάθε κορυφή u διαφορετική από την s. Έτσι λαμβάνουμε υπόψη το ενδεχόμενο η u να μην είναι προσπελάσιμη από την s. Η τιμή της εκτίμησης $\ell[u]$ αντιστοιχεί στο μήκος του συντομότερου μονοπατιού από την s στη u που έχει ανακαλύψει ο αλγόριθμος *μέχρι τη συγκεκριμένη στιγμή*.

Ο αλγόριθμος εξετάζει τις ακμές του γραφήματος και αναπροσαρμόζει τις εκτιμήσεις του με βάση το Πόρισμα 5.1. Συγκεκριμένα, αν η εκτίμηση $\ell[u] > \ell[v] + w(v, u)$, η εξέταση της ακμής (v, u) προκαλεί την αναπροσαρμογή της σε $\ell[u] = \ell[v] + w(v, u)$. Υποθέτοντας επαγωγικά ότι η εκτίμηση για τη v τεκμηριώνεται από ένα $s - v$ μονοπάτι μήκους $\ell[v]$, καταλήγουμε στο συμπέρασμα ότι η αναπροσαρμοσμένη εκτίμηση για τη u τεκμηριώνεται από ένα $s - u$ μονοπάτι μήκους $\ell[u]$. Το μονοπάτι αυτό αποτελείται από το αντίστοιχο μονοπάτι από την s στη v και την ακμή (v, u).

Η εκτίμηση $\ell[u]$ είναι "απαισιόδοξη" με την έννοια ότι είναι πάντοτε μεγαλύτερη ή ίση της απόστασης $d(s, u)$. Ο λόγος είναι ότι η τρέχουσα τιμή της εκτίμησης τεκμηριώνεται από το συντομότερο $s - u$ μονοπάτι που έχει ανακαλυφθεί από τον αλγόριθμο. Κανένα $s - u$ μονοπάτι όμως δεν έχει μήκος μικρότερο από $d(s, u)$. Αν ο αλγόριθμος ανακαλύψει ένα συντομότερο $s - u$ μονοπάτι, η εκτίμηση γίνεται $\ell[u] = d(s, u)$. Η εκτίμηση παραμένει σε αυτή την τιμή μέχρι το τέλος του αλγορίθμου, αφού δεν υπάρχει $s - u$ μονοπάτι με μήκος μικρότερο μήκος.

Λήμμα 5.1. *Έστω γράφημα $G(V, E, w)$ με αρχική κορυφή s. Για κάθε κορυφή u, η εκτίμηση $\ell[u]$ παραμένει μεγαλύτερη ή ίση της απόστασης $d(s, u)$ για κάθε ακολουθία εξέτασης ακμών και αναπροσαρμογών.*

Απόδειξη. Αρχικά είναι $\ell[u] = \infty \geq d(s, u)$ για κάθε κορυφή $u \in V \setminus \{s\}$. Επίσης, για την αρχική κορυφή είναι $\ell[s] = 0 \geq d(s, s)$ επειδή είτε $d(s, s) = 0$ αν η s δεν εμπλέκεται σε κύκλο αρνητικού μήκους, είτε $d(s, s) = -\infty$ διαφορετικά.

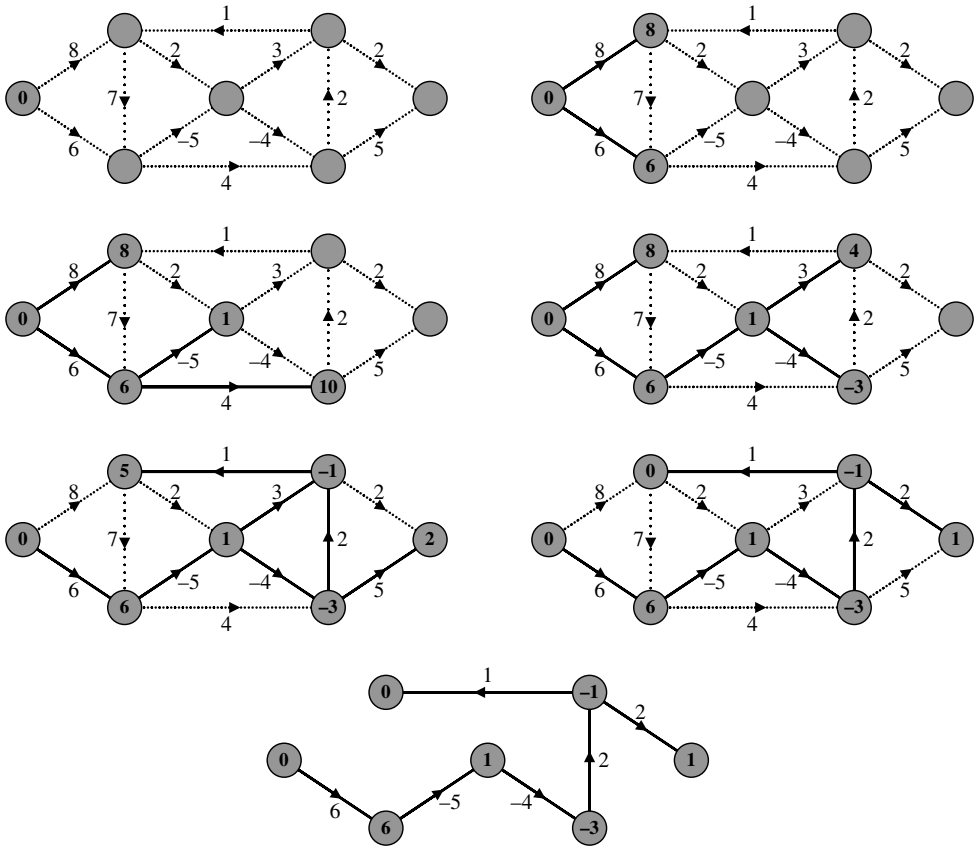
Υποθέτουμε επαγωγικά ότι η ιδιότητα ισχύει μετά την εξέταση κάποιων ακμών και τις αντίστοιχες αναπροσαρμογές. Έστω (v, u) η επόμενη ακμή που εξετάζεται. Αν η εκτίμηση $\ell[u]$ δεν αναπροσαρμόστηκε, συνεχίζει να είναι $\ell[u] \geq d(s, u)$ λόγω της επαγωγικής υπόθεσης. Αν η εκτίμηση $\ell[u]$ αναπροσαρμόστηκε, έχουμε:

$$\ell[u] = \ell[v] + w(v, u) \geq d(s, v) + w(v, u) \geq d(s, u)$$

Η πρώτη ανισότητα ισχύει γιατί $\ell[v] \geq d(s, v)$ λόγω της επαγωγικής υπόθεσης. Η δεύτερη ανισότητα προκύπτει από το Πόρισμα 5.1. □

Επισήμανση. Παρατηρούμε ότι αν η κορυφή u δεν είναι προσπελάσιμη από την αρχική κορυφή s, παραμένει $\ell[u] = \infty$ για κάθε ακολουθία εξέτασης ακμών και αναπροσαρμογών. □

Η διαφορά μεταξύ διαφορετικών αλγορίθμων έγκειται στη στρατηγική με την οποία εξετάζουν τις ακμές του γραφήματος. Κάθε τέτοια στρατηγική πρέπει να εξασφαλίζει ότι τελικά οι εκτιμήσεις για όλες τις κορυφές είναι ίσες με τις αποστάσεις τους από την αρχική κορυφή.



Σχήμα 5.16: Παράδειγμα λειτουργίας του αλγόριθμου Bellman-Ford. Στην επανάληψη i υπολογίζονται τα μήκη των συντομότερων μονοπατιών από την αρχική κορυφή που έχουν i ή λιγότερες ακμές. Οι ετικέτες των κορυφών δηλώνουν αυτά τα μήκη. Η έλλειψη ετικέτας δηλώνει ότι το αντίστοιχο μήκος δεν έχει αρχικοποιηθεί ακόμη (είναι $L(i,u) = \infty$). Οι ακμές που χρησιμοποιούνται στα αντίστοιχα μονοπάτια σημειώνονται συμπαγείς και οι υπόλοιπες διακεκομμένες.

5.6.1 Ο Αλγόριθμος των Bellman-Ford

Ο αλγόριθμος των Bellman-Ford εφαρμόζει την τεχνική του δυναμικού προγραμματισμού για την επίλυση του Προβλήματος των Συντομότερων Μονοπατιών από μια αρχική κορυφή. Ο αλγόριθμος αναπροσαρμόζει τις εκτιμήσεις για τις αποστάσεις των κορυφών από την την αρχική κορυφή θεωρώντας μονοπάτια με όλο και περισσότερες ακμές. Ο αλγόριθμος Bellman-Ford επιλύει τη γενική περίπτωση του προβλήματος, όπου κάποιες ακμές μπορούν να έχουν αρνητικά μήκη.

Θεωρούμε γράφημα $G(V, E, w)$ με n κορυφές και m ακμές και αρχική κορυφή s. Για κάθε $i = 0, \ldots, n-1$ και κάθε κορυφή $u \in V$, έστω $L(i,u)$ το μήκος του συντομότερου $s-u$ μονοπατιού με i ή λιγότερες ακμές. Κάθε μονοπάτι έχει το πολύ $n-1$ ακμές (εξ' ορισμού τα μονοπάτια δεν έχουν επαναλαμβανόμενες κορυφές). Υπό την προϋπόθεση ότι δεν υπάρχει κύκλος αρνητικού μήκους προσπελάσιμος από την s, ισχύει ότι $L(n-1,u) = d(s,u)$ για κάθε $u \in V$.

Αρχικά, είναι $L(0,s) = 0$ και $L(0,u) = \infty$ για κάθε κορυφή u διαφορετική από την αρχική. Πράγματι, αν περιοριστούμε σε τετριμμένα μονοπάτια χωρίς ακμές, η αρχική κορυφή είναι η μόνη

κορυφή που είναι προσπελάσιμη από τον εαυτό της.

Ένα $s-u$ μονοπάτι με $i+1$ ακμές προκύπτει από ένα $s-v$ μονοπάτι με i ακμές και την ακμή (v,u), για κάποια κορυφή v με $(v,u) \in E$. Το συντομότερο $s-u$ μονοπάτι με $i+1$ ή λιγότερες ακμές είτε ταυτίζεται με το συντομότερο $s-u$ μονοπάτι με i ή λιγότερες ακμές είτε αποτελείται από το συντομότερο $s-v$ μονοπάτι με i ακμές και την ακμή (v,u), για κάποια κορυφή v με $(v,u) \in E$. Το συντομότερο $s-u$ μονοπάτι με $i+1$ ή λιγότερες ακμές προκύπτει από το καλύτερο από τα παραπάνω ενδεχόμενα. Τυπικά, το $L(i+1,u)$ δίνεται από την αναδρομική εξίσωση:

$$L(i+1,u) = \min\{L(i,u), \min_{v:(v,u)\in E}\{L(i,v)+w(v,u)\}\} \tag{5.1}$$

Εφόσον το γράφημα G δεν περιέχει κύκλο αρνητικού μήκους που είναι προσπελάσιμος από την s, το μήκος της συντομότερης διαδρομής από την s προς μια κορυφή u (και η αντίστοιχη απόσταση) είναι το μήκος του συντομότερου $S-u$ μονοπατιού με $n-1$ ή λιγότερες ακμές. Άρα αν δεν υπάρχει κύκλος αρνητικού μήκους προσπελάσιμος από την s, είναι $L(i,u) = L(n-1,u) = d(s,u)$ για κάθε $i \geq n$ και κάθε $u \in V$. Αν προκύψει ότι $L(n,u) < L(n-1,u)$ για κάποια κορυφή u, συμπεραίνουμε ότι το γράφημα έχει κύκλο αρνητικού μήκους.

Οι αποστάσεις όλων των κορυφών από την s μπορούν να υπολογιστούν από την αναδρομική εξίσωση (5.1) με τη μέθοδο του δυναμικού προγραμματισμού. Για κάθε $i = 0,\ldots,n-1$ και κάθε κορυφή u, ο υπολογισμός των μηκών $L(i+1,u)$ απαιτεί μόνο τα μήκη $L(i,v)$ για όλες τις κορυφές $v \in V$. Για τον υπολογισμό των συντομότερων μονοπατιών, το $p[u]$ περιέχει την τελευταία (πριν την u) κορυφή στο συντομότερο $s-u$ μονοπάτι με i ή λιγότερες ακμές.

Αυτή είναι η βασική ιδέα του αλγόριθμου Bellman-Ford. Ένα παράδειγμα λειτουργίας του αλγόριθμου Bellman-Ford δίνεται στο Σχήμα 5.16. Ο αλγόριθμος μπορεί να υλοποιηθεί εύκολα ακολουθώντας τα βήματα που μόλις περιγράψαμε. Όμως μπορούμε να πετύχουμε πιο αποδοτική υλοποίηση συνδυάζοντας την παραπάνω ιδέα με την τεχνική της αναπροσαρμογής των εκτιμήσεων που περιγράφηκε στην Ενότητα 5.6. Στη συνέχεια περιγράφουμε τη δεύτερη, πιο αποδοτική υλοποίηση.

```
Bellman-Ford(G(V,E,w), s)
    for all u ∈ V do
        ℓ[u] ← ∞; p[u] ← NIL;
    ℓ[s] ← 0;
    for i ← 1 to n − 1 do
        for all (v,u) ∈ E do
            if ℓ[u] > ℓ[v] + w(v,u) then
                ℓ[u] ← ℓ[v] + w(v,u); p[u] ← v;
    for all (v,u) ∈ E do
        if ℓ[u] > ℓ[v] + w(v,u) then
            return(ΚΥΚΛΟΣ ΑΡΝΗΤΙΚΟΥ ΜΗΚΟΥΣ);
```

Ο αλγόριθμος εξελίσσεται σε $n-1$ φάσεις που αντιστοιχούν στις επαναλήψεις της for-ανακύκλωσης. Για κάθε κορυφή u, η εκτίμηση $\ell[u]$ είναι μικρότερη ή ίση από το $L(i,u)$ (δηλ. από το μήκος του συντομότερου $s-u$ μονοπατιού με i ή λιγότερες ακμές) μετά την ολοκλήρωση της

i-οστής φάσης. Αρχικά είναι $\ell[s] = 0$ και $\ell[u] = \infty$ για κάθε $u \in V \setminus \{s\}$. Επαγωγικά υποθέτουμε ότι μετά την ολοκλήρωση της φάσης i, $i = 0, 1, \ldots, n-1$, είναι $\ell[u] \leq L(i, u)$. Στη φάση $i+1$, ο αλγόριθμος εξετάζει όλες τις ακμές του γραφήματος και αναπροσαρμόζει τις εκτιμήσεις με βάση το Πόρισμα 5.1. Από την αναδρομική εξίσωση (5.1), προκύπτει ότι μετά την ολοκλήρωση της φάσης $i+1$, είναι $\ell[u] \leq L(i+1, u)$ για κάθε κορυφή u. Επομένως, μετά την ολοκλήρωση της φάσης $n-1$ είναι $\ell[u] \leq L(n-1, u)$. Αν δεν υπάρχει κύκλος αρνητικού μήκους προσπελάσιμος από την s, έχουμε $\ell[u] = d(s, u)$ στο τέλος της $(n-1)$-οστής φάσης. Αν υπάρχει κύκλος αρνητικού μήκους, η εκτίμηση κάποιας κορυφής θα μειωθεί αν εξετάσουμε τις ακμές για μία ακόμη φορά. Αυτός ο έλεγχος εφαρμόζεται στο τέλος του αλγόριθμου.

Κάθε ακμή του γραφήματος εξετάζεται ακριβώς n φορές. Επομένως, ο χρόνος εκτέλεσης του αλγόριθμου Bellman-Ford είναι $\Theta(nm)$. Στη συνέχεια αποδεικνύουμε την ορθότητα του αλγόριθμου.

Λήμμα 5.2. *Έστω γράφημα $G(V, E, w)$ και αρχική κορυφή s. Αν δεν υπάρχει κύκλος αρνητικού μήκους προσπελάσιμος από την s, στο τέλος του αλγόριθμου ισχύει ότι $\ell[u] = d(s, u)$ για κάθε κορυφή u. Επίσης, ο αλγόριθμος δεν επιστρέφει ένδειξη ύπαρξης κύκλου αρνητικού μήκους.*

Απόδειξη. Έστω u μια οποιαδήποτε κορυφή. Αν η u δεν είναι προσπελάσιμη από την s, τότε παραμένει $\ell[u] = \infty = d(s, u)$ καθόλη τη διάρκεια του αλγόριθμου (βλ. Λήμμα 5.1 και ακόλουθη επισήμανση). Αν η u είναι προσπελάσιμη από την s, έστω $p = (v_0, v_1, \ldots, v_{k-1}, v_k)$, $v_0 = s$, $v_k = u$, ένα συντομότερο $s - u$ μονοπάτι που περιλαμβάνει k ακμές. Θα δείξουμε ότι στο τέλος της φάσης i, $i = 0, 1, \ldots, k$, είναι $\ell[v_i] = d(s, v_i)$. Αυτό αρκεί γιατί (α) υπάρχει μονοπάτι με $n-1$ ή λιγότερες ακμές για κάθε κορυφή u προσπελάσιμη από την s, και (β) η εκτίμηση $\ell[u]$ δεν μπορεί να γίνει μικρότερη από $d(s, u)$ (Λήμμα 5.1).

Αρχικά (φάση 0) είναι $\ell[s] = 0 = d(s, s)$. Επαγωγικά υποθέτουμε ότι το ζητούμενο ισχύει μέχρι και την ολοκλήρωση της φάσης $i-1$. Μετά την εξέταση της ακμής (v_{i-1}, v_i) στη φάση i, είναι

$$d(s, v_i) \leq \ell[v_i] \leq \ell[v_{i-1}] + w(v_{i-1}, v_i) = d(s, v_{i-1}) + w(v_{i-1}, v_i) = d(s, v_i)$$

Επομένως, $\ell[v_i] = d(s, v_i)$ όπως απαιτείται. Στην παραπάνω σχέση, η πρώτη ανισότητα έπεται από το Λήμμα 5.1. Η δεύτερη ανισότητα ισχύει μετά την εξέταση της ακμής (v_{i-1}, v_i) και την ενδεχόμενη αναπροσαρμογή της εκτίμησης $\ell[v_i]$. Η επόμενη ισότητα ισχύει γιατί $\ell[v_{i-1}] = d(s, v_{i-1})$ από την επαγωγική υπόθεση. Η τελευταία ισότητα ισχύει γιατί το p είναι ένα συντομότερο $s - u$ μονοπάτι και άρα το τμήμα του μέχρι τη v_i είναι ένα συντομότερο $s - v_i$ μονοπάτι (Πρόταση 5.5).

Επομένως μετά την ολοκλήρωση της φάσης $n-1$, είναι $\ell[u] = d(s, u)$ για κάθε κορυφή u. Από το Λήμμα 5.1, καμία ακολουθία εξέτασης ακμών δεν προκαλεί αναπροσαρμογή της εκτίμησης $\ell[u]$ σε τιμή μικρότερη από $d(s, u)$. Άρα οι έλεγχοι στην τελευταία φάση αποτυγχάνουν και ο αλγόριθμος δεν επιστρέφει ένδειξη ύπαρξης κύκλου αρνητικού μήκους. □

Λήμμα 5.3. *Έστω γράφημα $G(V, E, w)$ και αρχική κορυφή s. Αν υπάρχει στο G κύκλος αρνητικού μήκους προσπελάσιμος από την s, ο αλγόριθμος επιστρέφει ένδειξη ύπαρξης κύκλου αρνητικού μήκους.*

Απόδειξη. Έστω $c = (v_0, v_1, \ldots, v_k)$, $v_0 = v_k$, ένας κύκλος αρνητικού μήκους προσπελάσιμος από την αρχική κορυφή s. Αφού όλες οι κορυφές του κύκλου είναι προσπελάσιμες από την s, οι εκτιμήσεις $\ell[v_i]$, $i = 0, 1, \ldots, k$, θα έχουν πεπερασμένες τιμές μετά την ολοκλήρωση της φάσης $n-1$. Για να καταλήξουμε σε άτοπο, υποθέτουμε ότι οι έλεγχοι στην τελευταία φάση αποτυγχάνουν και ο αλγόριθμος δεν επιστρέφει ένδειξη ύπαρξης κύκλου αρνητικού μήκους. Πρέπει λοιπόν να είναι $\ell[v_i] \leq \ell[v_{i-1}] + w(v_{i-1}, v_i)$ για κάθε $i = 1, \ldots, k$ (δηλαδή για όλες τις ακμές κατά μήκος του κύκλου). Αθροίζοντας κατά μέλη, έχουμε

$$\sum_{i=1}^{k} \ell[v_i] \leq \sum_{i=1}^{k} \ell[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i) \Rightarrow \sum_{i=1}^{k} w(v_{i-1}, v_i) \geq 0$$

Το συμπέρασμα προκύπτει επειδή $v_0 = v_k$ και τα δύο αθροίσματα των εκτιμήσεων συμπίπτουν. Το συμπέρασμα αποτελεί αντίφαση στην υπόθεση ότι ο c είναι κύκλος αρνητικού μήκους. □

Άσκηση 5.27. Να εφαρμόσετε τον αλγόριθμο Bellman-Ford στο γράφημα του Σχήματος 5.15 με αρχική κορυφή την a.

Άσκηση 5.28. Να διατυπώσετε ψευδοκώδικα που τυπώνει έναν κύκλο αρνητικού μήκους όταν ο αλγόριθμος Bellman-Ford επιστρέφει ένδειξη για ύπαρξη τέτοιου κύκλου.

Λύση. Έστω (u, v) η πρώτη ακμή με $\ell[u] > \ell[v] + w(v, u)$ στο τέλος του αλγορίθμου. Η ακμή (u, v) ανήκει σε κύκλο αρνητικού μήκους που είναι προσπελάσιμος από την s. Με μόνη πιθανή εξαίρεση την ακμή (u, v), οι ακμές του κύκλου έχουν χρησιμοποιηθεί για αναπροσαρμογή των εκτιμήσεων στις προηγούμενες φάσεις και οι κορυφές του έχουν αποθηκευθεί στον πίνακα p. Ακολουθώντας τα στοιχεία του πίνακα p, βρίσκουμε όλες τις κορυφές του c. Ο παρακάτω ψευδοκώδικας αντικαθιστά την τελευταία φάση στον αλγόριθμο Bellman-Ford.

```
for all (u, v) ∈ E do
    if ℓ[u] > ℓ[v] + w(v, u) then
        print(u, v);
        y ← v; x ← p[y]; p[y] ← NIL;
        while x ≠ NIL do
            print(x);
            y ← x; x ← p[y]; p[y] ← NIL;
return(ΚΥΚΛΟΣ ΑΡΝΗΤΙΚΟΥ ΜΗΚΟΥΣ);
```

5.6.2 Ο Αλγόριθμος του Dijkstra

Ο αλγόριθμος του Dijkstra είναι ένας άπληστος αλγόριθμος που επιλύει το Πρόβλημα των Συντομότερων Μονοπατιών από μία αρχική κορυφή. Ο αλγόριθμος του Dijkstra εφαρμόζεται μόνο όταν τα μήκη των ακμών είναι μη-αρνητικά και είναι ασυμπτωτικά ταχύτερος από τον αλγόριθμο Bellman-Ford. Όπως και στις προηγούμενες ενότητες, θεωρούμε γράφημα $G(V, E, w)$ με n κορυφές και m ακμές και αρχική κορυφή s. Το γράφημα δεν έχει ακμές αρνητικού μήκους, άρα ούτε κύκλους αρνητικού μήκους.

Ο αλγόριθμος του Dijkstra εντάσσει τις κορυφές στο ΔΣΜ σε αύξουσα σειρά απόστασης από την αρχική κορυφή. Από αυτή την άποψη μπορεί να θεωρηθεί σαν γενίκευση της Αναζήτησης Πρώτα σε Πλάτος σε γραφήματα με βάρη στις ακμές. Τη στιγμή που μια κορυφή εντάσσεται στο ΔΣΜ, έχει καθοριστεί ένα συντομότερο μονοπάτι και η απόστασή της από την αρχική κορυφή. Ο αλγόριθμος συνεχίζει εντάσσοντας στο ΔΣΜ κορυφές σε μεγαλύτερη απόσταση. Αφού δεν υπάρχουν ακμές αρνητικού μήκους, οι κορυφές σε μεγαλύτερη απόσταση δεν επηρεάζουν τις αποστάσεις των προηγούμενων κορυφών.

Η βασική τεχνική στον αλγόριθμο του Dijkstra είναι αυτή της διατήρησης και αναπροσαρμογής εκτιμήσεων. Ο αλγόριθμος διατηρεί μια εκτίμηση $\ell[u]$ για την απόσταση κάθε κορυφής u από την αρχική κορυφή s. Αρχικά είναι $\ell[s] = 0$ και $\ell[u] = \infty$ για κάθε κορυφή u διαφορετική από την αρχική. Ο αλγόριθμος διατηρεί ακόμη το σύνολο S των κορυφών που έχουν ενταχθεί στο ΔΣΜ. Για κάθε κορυφή $u \in S$, στο $p[u]$ αποθηκεύεται η τελευταία (πριν τη u) κορυφή στο συντομότερο $s - u$ μονοπάτι στο ΔΣΜ.

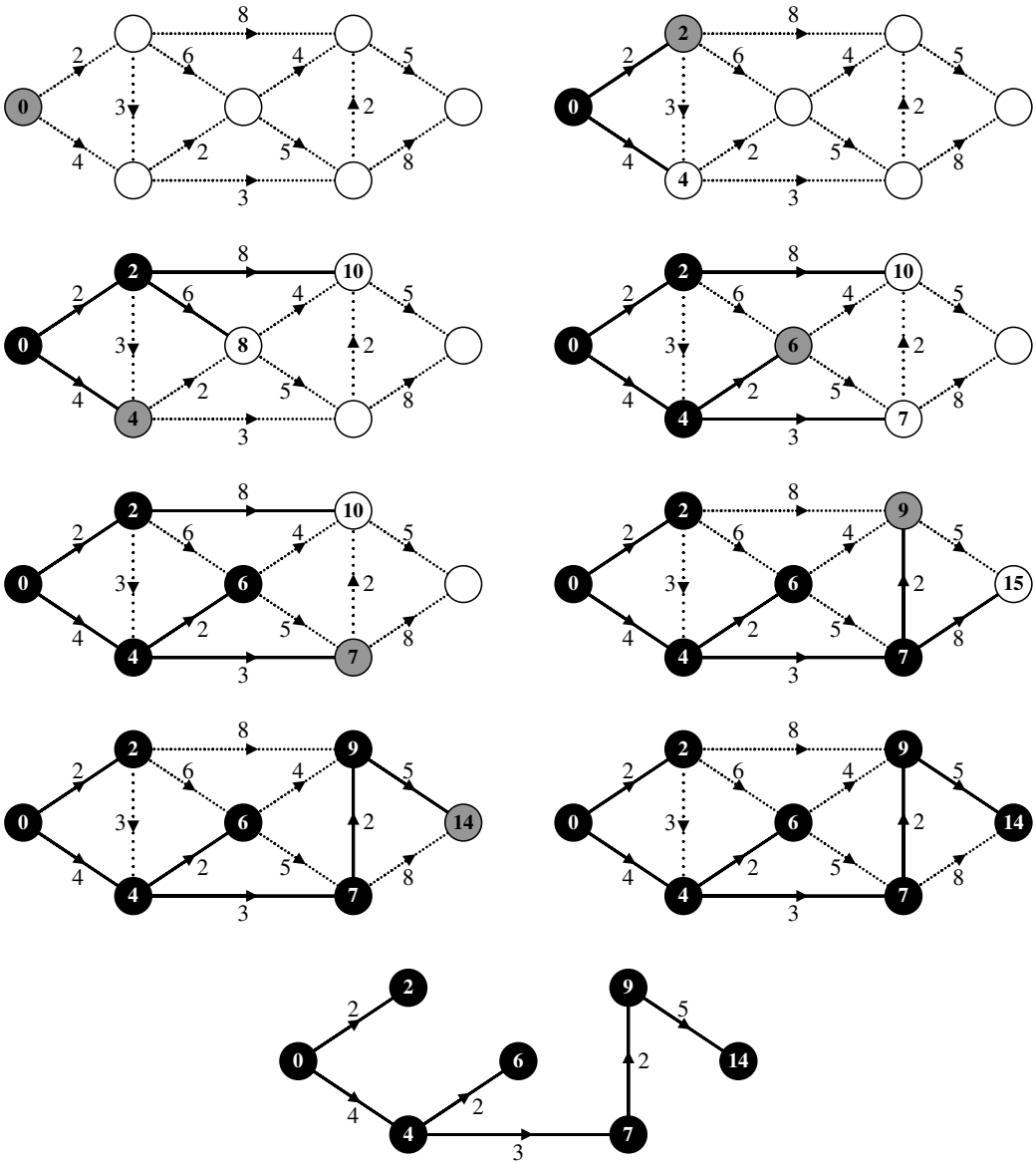
Ο αλγόριθμος λειτουργεί σε επαναλήψεις, εντάσσοντας μία νέα κορυφή στο ΔΣΜ σε κάθε επανάληψη. Στην τρέχουσα επανάληψη, ο αλγόριθμος εντάσσει στο ΔΣΜ την κορυφή $u \notin S$ (εκτός δέντρου) με την μικρότερη εκτίμηση (*άπληστη επιλογή*). Σε αυτό το σημείο, η εκτίμηση $\ell[u]$ είναι ίση με την απόστασή της u από την s (βλ. Λήμμα 5.4). Έτσι το συντομότερο μονοπάτι και η απόσταση της u παγιώνονται. Πριν προχωρήσει στην επόμενη επανάληψη, ο αλγόριθμος εξετάζει όλες τις εξερχόμενες ακμές από τη u και αναπροσαρμόζει τις εκτιμήσεις των γειτονικών κορυφών της.

Στο τέλος του αλγόριθμου, όλες οι κορυφές που είναι προσπελάσιμες από την s έχουν ενταχθεί στο ΔΣΜ και οι εκτιμήσεις τους έχουν τιμή ίση με την απόστασή τους από την s. Οι κορυφές που δεν είναι προσπελάσιμες από την s συνεχίζουν να διατηρούν άπειρες εκτιμήσεις. Ένα παράδειγμα λειτουργίας του αλγόριθμου του Dijkstra δίνεται στο Σχήμα 5.17.

```
Dijkstra(G(V, E, w), s)
    for all u ∈ V do
        ℓ[u] ← ∞; p[u] ← NIL;
    ℓ[s] ← 0; S ← ∅;
    while |S| < n do
        Έστω u κορυφή με ℓ[u] = min_{v∉S}{ℓ[v]};
        S ← S ∪ {u};
        for all v ∈ L[u] do
            if ℓ[v] > ℓ[u] + w(u, v) then
                ℓ[v] ← ℓ[u] + w(u, v); p[v] ← u;
```

Ο αλγόριθμος εκτελεί n επαναλήψεις. Σε κάθε επανάληψη, υπολογίζεται η κορυφή εκτός ΔΣΜ με την ελάχιστη εκτίμηση και εντάσσεται στο ΔΣΜ. Κάθε εξερχόμενη ακμή της εξετάζεται και η εκτίμηση της αντίστοιχης γειτονικής κορυφής αναπροσαρμόζεται. Συνολικά κάθε ακμή εξετάζεται μία φορά και συμβαίνουν $O(m)$ αναπροσαρμογές εκτιμήσεων.

Αν οι κορυφές εκτός ΔΣΜ οργανωθούν σε δυαδικό σωρό (binary heap) με βάση τις εκτιμήσεις τους, η εξαγωγή της κορυφής με την ελάχιστη εκτίμηση γίνεται σε χρόνο $O(\log n)$. Επίσης, η αναδιοργάνωση του σωρού μετά από κάθε αναπροσαρμογή εκτίμησης γίνεται σε χρόνο $O(\log n)$.



Σχήμα 5.17: Παράδειγμα λειτουργίας του αλγόριθμου του Dijkstra. Οι ετικέτες των κορυφών δηλώνουν τις εκτιμήσεις του αλγόριθμου. Οι κορυφές που έχουν ενταχθεί στο ΔΣΜ (σύνολο S) σημειώνονται μαύρες. Οι κορυφές που δεν έχουν ενταχθεί ακόμη στο ΔΣΜ (σύνολο $V \setminus S$) σημειώνονται άσπρες και η κορυφή εκτός ΔΣΜ με την ελάχιστη ετικέτα σημειώνεται γκρίζα. Οι ετικέτες των μαύρων κορυφών είναι ίσες με τις αποστάσεις τους από την αρχική κορυφή. Οι ακμές που έχουν ενταχθεί στο ΔΣΜ είναι συμπαγείς και οι υπόλοιπες ακμές διακεκομμένες.

Ο συνολικός χρόνος εκτέλεσης του αλγόριθμου είναι $O((n+m)\log n)$ στη χειρότερη περίπτωση. Ο χρόνος εκτέλεσης χειρότερης περίπτωσης βελτιώνεται σε $O(n\log n+m)$ αν οι κορυφές εκτός ΔΣΜ οργανωθούν σε σωρό Fibonacci.

Επισήμανση. Είναι αξιοσημείωση η ομοιότητα μεταξύ των αλγόριθμων του Prim και του Dijkstra. Και οι δύο αλγόριθμοι ξεκινούν από μία αρχική κορυφή και διατηρούν ένα δέντρο που συνδέει όλες τις κορυφές του S. Σε κάθε επανάληψη, η κορυφή με το ελάχιστο κόστος επιλέγεται για ένταξη στο δέντρο. Στον αλγόριθμο του Prim, το κόστος ένταξης είναι το βάρος της ελαφρύτερης ακμής προς κορυφή του δέντρου, $c[u]=\min_{v\in S}\{w(v,u)\}$. Στον αλγόριθμο του Dijkstra, το κόστος ένταξης είναι η εκτίμηση της απόστασης από την αρχική κορυφή με βάση τις αποστάσεις των κορυφών του δέντου, $\ell[u]=\min_{v\in S}\{\ell[v]+w(v,u)\}$ (θεωρούμε ότι $w(v,u)=\infty$ αν $(v,u)not\in E$). Η μόνη ουσιαστική διαφορά στην υλοποίηση των δύο αλγόριθμων είναι ο τρόπος υπολογισμού του κόστους ένταξης στο δέντρο (παρατηρείστε επίσης την ομοιότητα στη χρονική πολυπλοκότητα των δύο αλγόριθμων). □

Το παρακάτω λήμμα τεκμηριώνει την ορθότητα του αλγόριθμου του Dijkstra όταν το γράφημα δεν έχει ακμές αρνητικού μήκους. Αυτό το λήμμα αρκεί γιατί η εκτίμηση μιας κορυφής δεν μπορεί να αναπροσαρμοστεί σε τιμή μικρότερη της απόστασής της.

Λήμμα 5.4. *Έστω γράφημα $G(V,E,w)$ χωρίς ακμές αρνητικού μήκους, και έστω s αρχική κορυφή. Τη στιγμή της ένταξης κάθε κορυφής u στο ΔΣΜ, ισχύει ότι $\ell[u]=d(s,u)$.*

Απόδειξη. Θα αποδείξουμε το λήμμα με επαγωγή στο $|S|$, δηλ. στον αριθμό των κορυφών που έχουν ενταχθεί στο ΔΣΜ. Αρχικά το S έχει μόνο την αρχική κορυφή, για την οποία ισχύει $\ell[s]=d(s,s)=0$. Επαγωγικά υποθέτουμε ότι το λήμμα ισχύει για όλες τις κορυφές που έχουν ενταχθεί στο ΔΣΜ πριν από την κορυφή u. Θα αποδείξουμε ότι τη στιγμή που η κορυφή u εντάσσεται στο ΔΣΜ, ισχύει ότι $\ell[u]=d(s,u)$. Από τη λειτουργία του αλγόριθμου, τη δεδομένη στιγμή η u έχει τη μικρότερη εκτίμηση $\ell[u]$ από όλες τις κορυφές εκτός ΔΣΜ.

Για να καταλήξουμε σε άτοπο, υποθέτουμε ότι τη δεδομένη στιγμή είναι $\ell[u]>d(s,u)$ (αυτό είναι το μοναδικό ενδεχόμενο λόγω του Λήμματος 5.1). Υπάρχει λοιπόν συντομότερο $s-u$ μονοπάτι p με συνολικό μήκος μικρότερο την τρέχουσα εκτίμηση $\ell[u]$. Έστω z η τελευταία κορυφή του p πριν από τη u. Επομένως, το μήκος του p είναι $d(s,u)=d(s,z)+w(z,u)<\ell[u]$. Αφού το γράφημα δεν έχει ακμές αρνητικού μήκους (άρα $w(z,u)\geq 0$), συμπεραίνουμε ότι

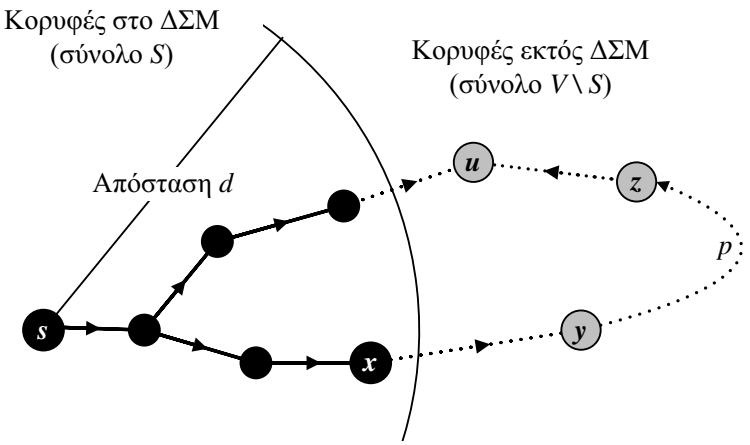
$$d(s,z)<\ell[u] \tag{5.2}$$

Η z δεν έχει ενταχθεί στο ΔΣΜ (δηλ. $z\notin S$). Διαφορετικά, θα ήταν

$$\ell[u]\leq\ell[z]+w(z,u)=d(s,z)+w(z,u)$$

επειδή ο αλγόριθμος θα είχε εξετάσει την ακμή (z,u) και αναπροσαρμόσει την εκτίμηση $\ell[u]$ κατά την προσθήκη της z στο S. Επίσης είναι $\ell[z]=d(s,z)$ από την επαγωγική υπόθεση.

Καταλήγουμε λοιπόν στο συμπέρασμα ότι το μονοπάτι p διέρχεται από κορυφές που δεν ανήκουν στο S. Έστω x η τελευταία κορυφή του S στο μονοπάτι p (η x μπορεί να ταυτίζεται με την s), και έστω y η αμέσως επόμενη κορυφή της x στο p (η y μπορεί να ταυτίζεται με τη z). Ο ορισμός των κορυφών x, y, και z απεικονίζεται στο Σχήμα 5.18. Αφού το p είναι συντομότερο $s-u$



Σχήμα 5.18: Η απόδειξη ορθότητας του αλγόριθμου του Dijkstra σε σχήμα. Οι κορυφές προστίθενται στο ΔΣΜ σε αύξουσα σειρά απόστασης από την αρχική κορυφή s. Αφού δεν υπάρχουν ακμές αρνητικού μήκους, η απόσταση της κορυφής u από την s δεν εξαρτάται από τις κορυφές y και z που βρίσκονται σε μεγαλύτερη απόσταση.

μονοπάτι, τα τμήματα του p από την s μέχρι την x, την y, και την z είναι αντίστοιχα συντομότερα $s - x$, $s - y$, και $s - z$ μονοπάτια (Πρόταση 5.5). Αφού το γράφημα δεν έχει ακμές αρνητικού μήκους, συμπεραίνουμε ότι

$$d(s, y) = d(s, x) + w(x, y) \leq d(s, z) \tag{5.3}$$

Από την παραπάνω ανάλυση, καταλήγουμε στο συμπέρασμα ότι

$$\ell[y] \leq \ell[x] + w(x, y) = d(s, x) + w(x, y) \leq d(s, z) < \ell[u]$$

Η πρώτη ανισότητα ισχύει γιατί ο αλγόριθμος έχει εξετάσει την ακμή (x, y) και αναπροσαρμόσει την εκτίμηση $\ell[y]$ κατά την προσθήκη της x στο S. Η επόμενη ισότητα προκύπτει από την επαγωγική υπόθεση. Οι τελευταίες δύο ανισότητες είναι οι (5.3) και (5.2) αντίστοιχα.

Δηλαδή υπάρχει κορυφή $y \notin S$ με εκτίμηση $\ell[y] < \ell[u]$. Αυτό αποτελεί αντίφαση στην υπόθεση ότι η u επιλέγεται για ένταξη στο ΔΣΜ. □

Άσκηση 5.29. Να σχολιάσετε την ορθότητα του ακόλουθου αλγόριθμου για το Πρόβλημα των Συντομότερων Μονοπατιών από μία αρχική κορυφή όταν το γράφημα έχει ακμές αρνητικού μήκους: Έστω $-x$, για κάποιο $x > 0$, το μικρότερο (αρνητικό) μήκος ακμής. Αυξάνουμε τα μήκη όλων των ακμών κατά x και χρησιμοποιούμε τον αλγόριθμο του Dijkstra για τον υπολογισμό των συντομότερων μονοπατιών.

Άσκηση 5.30. Να αποδείξετε ότι η Αναζήτηση Πρώτα σε Πλάτος υπολογίζει ένα Δέντρο Συντομότερων Μονοπατιών με ρίζα την αρχική κορυφή s όταν οι ακμές έχουν μοναδιαίο μήκος. Τροποποιήστε τον ψευδοκώδικα της ΑΠΠ ώστε να υπολογίζει και να επιστρέφει τις αποστάσεις όλων των κορυφών από την αρχική κορυφή.

Λύση. Η τροποποίηση του ψευδοκώδικα είναι απλή. Διατηρούμε έναν πίνακα ℓ με τις αποστάσεις όλων των κορυφών από την s. Αρχικά είναι $\ell[s] = 0$ και $\ell[v] = \infty$ για κάθε κορυφή v διαφορετική από τη ρίζα. Στο σημείο που η κορυφή v γίνεται υπό-εξέταση και θέτουμε $p[v] \leftarrow u$, θέτουμε επίσης $\ell[v] \leftarrow \ell[u] + 1$.

Για να αποδείξουμε ότι αυτή η τροποποίηση υπολογίζει τις αποστάσεις όλων των κορυφών από την s, θα δείξουμε ότι η ΑΠΠ μπορεί να θεωρηθεί σαν ειδική περίπτωση του αλγόριθμου του Dijkstra όταν όλες οι ακμές έχουν μοναδιαίο μήκος.

Χρειάζεται λοιπόν να αποδείξουμε ότι οι κορυφές $v \in V$ γίνονται πρώτα υπο-εξέταση και στη συνέχεια εξερευνημένες σε αύξουσα σειρά των εκτιμήσεων $\ell[v]$. Δηλαδή, θα αποδείξουμε ότι για κάθε κορυφή v' που γίνεται υπο-εξέταση και (στη συνέχεια εξερευνημένη) πριν από μια κορυφή v, ισχύει ότι $\ell[v'] \leq \ell[v]$.

Θα χρησιμοποιήσουμε μαθηματική επαγωγή. Ο ισχυρισμός είναι τετριμμένος για την αρχική κορυφή, αφού αυτή είναι η πρώτη υπο-εξέταση κορυφή. Θεωρούμε τη στιγμή που μια κορυφή v αλλάζει χαρακτηρισμό από ανεξερεύνητη σε υπο-εξέταση. Έστω x ο πατέρας της v στο δέντρο της ΑΠΠ. Είναι δηλαδή $p[v] = x$ και $\ell[v] = \ell[x] + 1$. Επαγωγικά υποθέτουμε όλες οι κορυφές x' που γίνονται εξερευνημένες πριν τη x έχουν $\ell[x'] \leq \ell[x]$.

Κάθε κορυφή v' που γίνεται υπο-εξέταση (και στη συνέχεια εξερευνημένη) πριν τη v συνδέεται με ακμή είτε με τη x είτε με κορυφή x' που έγινε εξερευνημένη πριν από την x. Η ακμή αυτή εξερευνάται από την ΑΠΠ πριν εξερευνηθεί η ακμή (x, v) και η κορυφή v γίνει εξερευνημένη. Εφαρμόζοντας την επαγωγική υπόθεση, προκύπτει το ζητούμενο: $\ell[v'] \leq \ell[x] + 1 = \ell[v]$.

Η ΑΠΠ αρχικοποιεί την εκτίμηση για μια κορυφή τη στιγμή που αυτή γίνεται υπο-εξέταση. Αυτή η εκτίμηση είναι τελική επειδή οι κορυφές εξερευνώνται σε αύξουσα σειρά εκτιμήσεων και οι ακμές έχουν το ίδιο μήκος (άρα η εκτίμηση για μια κορυφή δεν μπορεί να μικρύνει από επόμενη ακμή). Αφού οι εκτιμήσεις υπολογίζονται με βάση το Πόρισμα 5.1 και οι κορυφές προστίθενται στο δέντρο σε αύξουσα σειρά εκτιμήσεων, η ΑΠΠ μπορεί να θεωρηθεί σαν μια ειδική περίπτωση (ή σαν μια ταχύτερη υλοποίηση) του αλγόριθμου του Dijkstra όταν όλες οι ακμές έχουν μοναδιαίο μήκος. □

Άσκηση 5.31. Μια παραλλαγή του προβλήματος του Συντομότερου Μονοπατιού προκύπτει όταν το κόστος κάθε μονοπατιού είναι το μήκος της μακρύτερης ακμής του (δηλ. το κόστος ενός μονοπατιού p είναι $b(p) = \max_{e \in p}\{w(e)\}$). Δεδομένου ενός γραφήματος $G(V, E, w)$, μιας αρχικής κορυφής s, και μιας τελικής κορυφής t, θέλουμε να υπολογίσουμε το $s - t$ μονοπάτι ελάχιστου κόστους. Το πρόβλημα αυτό είναι γνωστό σαν *Πρόβλημα της Στενωπού* (Bottleneck Shortest Path Problem). Να τροποποιήσετε τον αλγόριθμο του Dijkstra ώστε να λύνει το Πρόβλημα της Στενωπού. Βασισμένοι στην απόδειξη ορθότητας του αλγόριθμου του Dijkstra, να αποδείξετε την ορθότητα του τροποποιημένου αλγόριθμου για το πρόβλημα της στενωπού. Συνεχίζει να είναι αναγκαία η υπόθεση ότι οι ακμές του γραφήματος δεν έχουν αρνητικά μήκη;

Άσκηση 5.32. Να διατυπώσετε αλγόριθμο με γραμμικό χρόνο εκτέλεσης χειρότερης περίπτωσης (δηλ. χρόνο εκτέλεσης $O(n + m)$) για το Πρόβλημα των Συντομότερων Μονοπατιών από μία Αρχική Κορυφή σε ένα ακυκλικό κατευθυνόμενο γράφημα με βάρη στις ακμές. Θεωρήστε ότι η αρχική κορυφή δεν έχει εισερχόμενες ακμές.

Λύση. Έστω $G(V, E, w)$ ακυκλικό κατευθυνόμενο γράφημα, και έστω s η αρχική κορυφή με εισερχόμενο βαθμό 0. Παρατηρούμε ότι αφού το γράφημα είναι ακυκλικό, δεν έχει κύκλους αρνητικού μήκους ακόμη και αν κάποιες ακμές έχουν αρνητικά μήκη.

Αρχικά υπολογίζουμε σε γραμμικό χρόνο μια τοπολογική διάταξη των κορυφών χρησιμοποιώντας Αναζήτηση Πρώτα σε Βάθος (Ενότητα 5.3.2). Αφού η s δεν έχει εισερχόμενες ακμές, μπορούμε να θεωρήσουμε ότι είναι η πρώτη κορυφή στην τοπολογική διάταξη.

Για να υπολογίσουμε τις αποστάσεις των κορυφών, διατηρούμε μια εκτίμηση $\ell[u]$ για την απόσταση κάθε κορυφής u από την s. Αρχικά είναι $\ell[s] = 0$ και $\ell[u] = \infty$ για κάθε κορυφή u διαφορετική από την αρχική. Ο αλγόριθμος προσθέτει τις κορυφές στο ΔΣΜ με τη σειρά που έχουν στην τοπολογική διάταξη. Κατά την προσθήκη μιας κορυφής στο ΔΣΜ, ελέγχονται όλες οι εξερχόμενες ακμές της και αναπροσαρμόζονται οι εκτιμήσεις των γειτόνων της. Αφήνεται σαν άσκηση να διατυπώσετε τον αλγόριθμο σε ψευδοκώδικα.

Η διαφορά αυτού του αλγόριθμου από τον αλγόριθμο του Dijkstra είναι ότι οι κορυφές προστίθενται στο ΔΣΜ με τη σειρά τους στην τοπολογική διάταξη και όχι σε αύξουσα σειρά των εκτιμήσεών τους. Αυτό κάνει τον αλγόριθμο ταχύτερο, αφού δεν χρειάζεται να οργανώνει τις κορυφές σε αύξουσα σειρά εκτιμήσεων. Ο αλγόριθμος χρειάζεται σταθερό χρόνο για την προσθήκη κάθε κορυφής στο δέντρο. Κάθε ακμή εξετάζεται μία φορά και η αντίστοιχη αναπροσαρμογή εκτίμησης γίνεται σε σταθερό χρόνο. Άρα ο αλγόριθμος έχει χρόνο εκτέλεσης χειρότερης περίπτωσης $\Theta(n + m)$ όπως απαιτείται.

Για την ορθότητα του αλγόριθμου, πρέπει να αποδείξουμε ότι τη στιγμή της προσθήκης μιας κορυφής u στο ΔΣΜ, η εκτίμηση $\ell[u]$ είναι ίση με την απόστασή της $d(s, u)$. Αυτό μπορεί να γίνει με επαγωγή στον αριθμό των κορυφών του ΔΣΜ χρησιμοποιώντας το γεγονός ότι όλες οι εισερχόμενες ακμές στην u έχουν ήδη εξεταστεί και η εκτίμησή $\ell[u]$ έχει πάρει την τελική της τιμή. Ο λόγος είναι ότι οι κορυφές προστίθενται στο ΔΣΜ με τη σειρά τους στην τοπολογική διάταξη. Έτσι όλες οι ακμές που καταλήγουν στη u ξεκινούν από κορυφές που βρίσκονται ήδη στο ΔΣΜ και έχουν εξεταστεί κατά την προσθήκη των αντίστοιχων κορυφών στο δέντρο. □

5.7 Συντομότερα Μονοπάτια για Όλα τα Ζεύγη Κορυφών

Σε αυτή την παραλλαγή του Προβλήματος των Συντομότερων Μονοπατιών, θεωρούμε ένα κατευθυνόμενο γράφημα $G(V, E, w)$ με μήκη στις ακμές, και υπολογίζουμε την απόσταση $d(u, v)$ και ένα συντομότερο $u - v$ μονοπάτι για κάθε ζευγάρι κορυφών (u, v). Θεωρούμε ότι οι ακμές του γραφήματος μπορεί να έχουν αρνητικά μήκη και ότι το γράφημα μπορεί να έχει κύκλους αρνητικού μήκους.

Ένας απλός τρόπος είναι να εφαρμόσουμε τον αλγόριθμο του Dijkstra (αν δεν υπάρχουν ακμές αρνητικού μήκους) ή τον αλγόριθμο των Bellman-Ford (αν υπάρχουν ακμές αρνητικού μήκους) με κάθε κορυφή $u \in V$ σαν αρχική. Ο αλγόριθμος για το Πρόβλημα των Συντομότερων Μονοπατιών από μία Αρχική Κορυφή εφαρμόζεται n φορές. Ο χρόνος εκτέλεσης είναι $O(nm + n^2 \log n)$ όταν δεν υπάρχουν ακμές αρνητικού μήκους, και $O(n^2 m)$ σε περίπτωση ύπαρξης ακμών αρνητικού μήκους.

Σε αυτή την ενότητα, θα παρουσιάσουμε τον αλγόριθμου των Floyd-Warshall που λύνει το Πρόβλημα των Συντομότερων Μονοπατιών για Όλα τα Ζεύγη Κορυφών σε χρόνο $O(n^3)$ ακόμη και αν το γράφημα έχει ακμές αρνητικού μήκους. Όταν το γράφημα δεν έχει ακμές αρνητικού

```
Floyd-Warshall(G(V, E, w))
    for i ← 1 to n do
        for j ← 1 to n do
            if (v_i, v_j) ∈ E then D_0[i, j] ← w(v_i, v_j);
            else D_0[i, j] ← ∞;
        D_0[i, i] ← 0;
    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                if D_{k-1}[i, j] > D_{k-1}[i, k] + D_{k-1}[k, j] then
                    D_k[i, j] ← D_{k-1}[i, k] + D_{k-1}[k, j];
                else D_k[i, j] ← D_{k-1}[i, j];
```

Σχήμα 5.19: Διατύπωση του αλγόριθμου Floyd-Warshall σε ψευδοκώδικα.

μήκους, ο αλγόριθμος των Floyd-Warshall είναι ταχύτερος από την εκτέλεση του αλγόριθμου του Dijkstra n φορές μόνο όταν το γράφημα είναι *πυκνό*, έχει δηλαδή $\Theta(n^2)$ ακμές.

5.7.1 Ο Αλγόριθμος των Floyd-Warshall

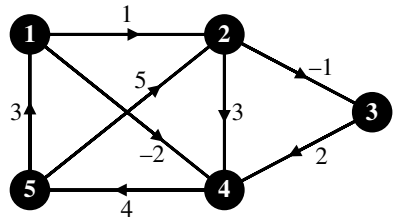
Χάριν απλότητας, υποθέτουμε αρχικά ότι το γράφημα $G(V, E, w)$ δεν περιέχει κύκλους αρνητικού μήκους (μπορεί όμως να έχει ακμές αρνητικού μήκους). Στο τέλος της ενότητας, θα δούμε πως διαπιστώνουμε την ύπαρξη κύκλων αρνητικού μήκους.

Για την περιγραφή του αλγόριθμου, θεωρούμε μια αυθαίρετα επιλεγμένη αρίθμηση των κορυφών. Έστω λοιπόν $V = \{v_1, v_2, \ldots, v_{n-1}, v_n\}$. Για κάθε $v_i, v_j \in V$, θεωρούμε ότι $w(v_i, v_j) = \infty$ αν $i \neq j$ και η ακμή (v_i, v_j) δεν υπάρχει στο γράφημα, και ότι $w(v_i, v_i) = 0$.

Η βασική ιδέα του αλγόριθμου προκύπτει από την ιδιότητα ότι ένα συντομότερο μονοπάτι αποτελείται από επιμέρους συντομότερα μονοπάτια. Συγκεκριμένα, έστω ότι θέλουμε να υπολογίσουμε την απόσταση $d(v_i, v_j)$ για κάποιες κορυφές $v_i, v_j \in V$ και γνωρίζουμε τις "ενδιάμεσες" αποστάσεις $d(v_i, v_k)$ και $d(v_k, v_j)$ για όλες τις κορυφές $v_k \in V \setminus \{v_i, v_j\}$ (βέβαια οι 'ενδιάμεσες" αποστάσεις και είναι επίσης μέρος του ζητούμενου στη συγκεκριμένη παραλλαγή). Αν το συντομότερο $v_i - v_j$ μονοπάτι δεν έχει καμία ενδιάμεση κορυφή, η απόσταση $d(v_i, v_j)$ είναι ίση με το μήκος της ακμής (v_i, v_j). Αν το συντομότερο $v_i - v_j$ μονοπάτι διέρχεται από την κορυφή v_k, το πρώτο τμήμα του αποτελεί ένα συντομότερο $v_i - v_k$ μονοπάτι και το δεύτερο τμήμα του ένα συντομότερο $v_k - v_j$ μονοπάτι (Πρόταση 5.5). Φυσικά το συντομότερο μονοπάτι προκύπτει από την καλύτερη επιλογή. Με άλλα λόγια,

$$d(v_i, v_j) = \min\{w(v_i, v_j), \min_{v_k \in V \setminus \{v_i, v_j\}} \{d(v_i, v_k) + d(v_k, v_j)\} \tag{5.4}$$

Ο αλγόριθμος Floyd-Warshall βασίζεται στην παραπάνω ιδέα και χρησιμοποιεί την τεχνική του δυναμικού προγραμματισμού για να υπολογίσει τις όλες τις αποστάσεις με bottom-up τρόπο. Για το συστηματικό bottom-up υπολογισμό των αποστάσεων, θεωρούμε το μήκος του συντομότερου $v_i - v_j$ μονοπατιού με κορυφές μόνο από το σύνολο $V_k = \{v_1, \ldots, v_k\}$, για κάθε $k = 0, 1, \ldots, n$. Με άλλα λόγια, υπολογίζουμε το μήκους του συντομότερου $v_i - v_j$ μονοπατιού με *ενδιάμεσες*



Σχήμα 5.20: Παράδειγμα γραφήματος για την εφαρμογή του αλγόριθμου Floyd-Warshall.

κορυφές μόνο από το V_k, για ολοένα και μεγαλύτερα σύνολα V_k. Αρχικά $V_0 = \emptyset$, και η μόνη επιλογή είναι η ακμή (v_i, v_j). Τελικά, $k = n$ και $V_n = V$, οπότε καταλήγουμε στο συντομότερο $v_i - v_j$ μονοπάτι.

Για κάθε $(v_i, v_j) \in V \times V$, και για κάθε $k = 0, 1, \ldots, n$, έστω $D_k[v_i, v_j]$ το μήκος του συντομότερου $v_i - v_j$ μονοπατιού με ενδιάμεσες κορυφές μόνο από το σύνολο $V_k = \{v_1, \ldots, v_k\}$. Αρχικά είναι $D_0[v_i, v_j] = w(v_i, v_j)$ γιατί $V_0 = \emptyset$. Έστω ότι για κάποια συγκεκριμένη τιμή του k, έχουμε υπολογίσει τις αποστάσεις $D_k[v_i, v_j]$ για κάθε $(v_i, v_j) \in V \times V$.

Το συντομότερο $v_i - v_j$ μονοπάτι με ενδιάμεσες κορυφές μόνο από το σύνολο $V_{k+1} = V_k \cup \{v_{k+1}\}$ διέρχεται καμία ή μία φορά από την κορυφή v_{k+1} (δεν διέρχεται περισσότερες γιατί πρόκειται για μονοπάτι). Αν δεν διέρχεται από την v_{k+1}, το συντομότερο $v_i - v_j$ μονοπάτι με ενδιάμεσες κορυφές μόνο από το V_{k+1} έχει μήκος $D_{k+1}[v_i, v_j] = D_k[v_i, v_j]$ (ταυτίζεται με το συντομότερο $v_i - v_j$ μονοπάτι με ενδιάμεσες κορυφές από το V_k). Διαφορετικά, το συντομότερο $v_i - v_j$ μονοπάτι αποτελείται από το συντομότερο $v_i - v_k$ μονοπάτι και το συντομότερο $v_k - v_j$ μονοπάτι αμφότερα με ενδιάμεσες κορυφές μόνο από το V_k. Το συνολικό μήκος αυτού του μονοπατιού είναι $D_{k+1}[v_i, v_j] = D_k[v_i, v_k] + D_k[v_k, v_j]$. Το συντομότερο μονοπάτι ακολουθεί την καλύτερη από τις παραπάνω επιλογές. Επομένως,

$$D_{k+1}[v_i, v_j] = \min\{D_k[v_i, v_j], D_k[v_i, v_k] + D_k[v_k, v_j]\}$$

Από την παραπάνω ανάλυση προκύπτει η ακόλουθη αναδρομική σχέση για τον υπολογισμό της απόστασης μεταξύ ενός ζεύγους κορυφών v_i και v_j. Τελικά είναι η απόσταση $d(v_i, v_j)$ είναι ίση με $D_n[v_i, v_j]$.

$$D_k[v_i, v_j] = \begin{cases} w(v_i, v_j) & \text{για } k = 0 \\ \min\{D_{k-1}[v_i, v_j], D_{k-1}[v_i, v_k] + D_{k-1}[v_k, v_j]\} & \text{για } k = 1, 2, \ldots, n \end{cases} \tag{5.5}$$

Η αναδρομική σχέση (5.5) αποτελεί ένα διαφορετικό τρόπο να γράψουμε τη σχέση (5.4) και να υπολογίσουμε τις αποστάσεις $D_k[v_i, v_j]$ για όλα τα ζεύγη κορυφών ταυτόχρονα. Η σημαντική παρατήρηση είναι ότι τα μήκη D_k για όλα τα ζεύγη κορυφών προκύπτουν εύκολα αν έχουμε υπολογίσει τα μήκη D_{k-1} για όλα τα ζεύγη των κορυφών. Για κάθε $k = 0, 1, \ldots, n$, μπορούμε να αποθηκεύουμε τα μήκη D_k σε έναν πίνακα $n \times n$. Ο αρχικός πίνακας D_0 περιέχει τα βάρη των ακμών του γραφήματος. Οι αποστάσεις για όλα τα ζεύγη κορυφών δίνονται από τον τελικό πίνακα D_n.

Μπορούμε λοιπόν να υπολογίσουμε τις αποστάσεις για όλα τα ζεύγη κορυφών υπολογίζοντας την ακολουθία πινάκων $D_0, D_1, \ldots, D_n$. Αυτό ακριβώς κάνει ο αλγόριθμος Floyd-Warshall. Ο

$$D_0 = \begin{pmatrix} 0 & 1 & \infty & -2 & \infty \\ \infty & 0 & -1 & 3 & \infty \\ \infty & \infty & 0 & 2 & \infty \\ \infty & \infty & \infty & 0 & 4 \\ 3 & 5 & \infty & \infty & 0 \end{pmatrix} \qquad D_1 = \begin{pmatrix} 0 & 1 & \infty & -2 & \infty \\ \infty & 0 & -1 & 3 & \infty \\ \infty & \infty & 0 & 2 & \infty \\ \infty & \infty & \infty & 0 & 4 \\ 3 & 4 & \infty & 1 & 0 \end{pmatrix}$$

$$D_2 = \begin{pmatrix} 0 & 1 & 0 & -2 & \infty \\ \infty & 0 & -1 & 3 & \infty \\ \infty & \infty & 0 & 2 & \infty \\ \infty & \infty & \infty & 0 & 4 \\ 3 & 4 & 3 & 1 & 0 \end{pmatrix} \qquad D_3 = \begin{pmatrix} 0 & 1 & 0 & -2 & \infty \\ \infty & 0 & -1 & 1 & \infty \\ \infty & \infty & 0 & 2 & \infty \\ \infty & \infty & \infty & 0 & 4 \\ 3 & 4 & 3 & 1 & 0 \end{pmatrix}$$

$$D_4 = \begin{pmatrix} 0 & 1 & 0 & -2 & 2 \\ \infty & 0 & -1 & 1 & 5 \\ \infty & \infty & 0 & 2 & 6 \\ \infty & \infty & \infty & 0 & 4 \\ 3 & 4 & 3 & 1 & 0 \end{pmatrix} \qquad D_5 = \begin{pmatrix} 0 & 1 & 0 & -2 & 2 \\ 8 & 0 & -1 & 1 & 5 \\ 9 & 10 & 0 & 2 & 6 \\ 7 & 8 & 7 & 0 & 4 \\ 3 & 4 & 3 & 1 & 0 \end{pmatrix}$$

Σχήμα 5.21: Η ακολουθία πινάκων $D_0, D_1, \ldots, D_5$ που υπολογίζει ο αλγόριθμος Floyd-Warshall για το γράφημα του Σχήματος 5.20.

ψευδοκώδικας του αλγόριθμου Floyd-Warshall δίνεται στο Σχήμα 5.19. Η ακολουθία των πινάκων $D_0, D_1, \ldots, D_n$ που υπολογίζει ο αλγόριθμος Floyd-Warshall για το γράφημα του Σχήματος 5.20 φαίνεται στο Σχήμα 5.21.

Ο χρόνος εκτέλεσης του αλγόριθμου Floyd-Warshall καθορίζεται από τα τρία for-loops που εξετάζουν όλες τις τριάδες $(k, i, j) \in \{1, \ldots, n\}^3$ και είναι $\Theta(n^3)$. Ο αλγόριθμος απαιτεί $\Theta(n^2)$ θέσεις μνήμης, αφού για κάθε $k = 1, \ldots, n$, αρκεί να κρατάμε στη μνήμη τους πίνακες D_k και D_{k-1}.

Η ύπαρξη κύκλων αρνητικού μήκους διαπιστώνεται εύκολα ελέγχοντας τα διαγώνια στοιχεία του πίνακα D_n στο τέλος του αλγόριθμου. Μια κορυφή v_i ανήκει σε κύκλο αρνητικού μήκους αν και μόνο αν είναι $D_n[v_i, v_i] < 0$ στο τέλος του αλγόριθμου Floyd-Warshall.

5.7.2 Αναπάρασταση Συντομότερων Μονοπατιών για Όλα τα Ζεύγη Κορυφών

Η επίλυση του Προβλήματος των Συντομότερων Μονοπατιών για Όλα τα Ζεύγη Κορυφών απαιτεί ακόμη τον υπολογισμό των ίδιων των συντομότερων μονοπατιών. Τα συντομότερα μονοπάτια για όλα τα ζεύγη κορυφών αναπαρίστανται με n Δέντρα Συντομότερων Μονοπατιών (ΔΣΜ), ένα για κάθε αρχική κορυφή.

Ένα ΔΣΜ με αρχική κορυφή v_i αναπαρίσταται μέσω του πίνακα p με n στοιχεία. Το στοιχείο $p[v_j]$ δηλώνει τον πατέρα κάθε κορυφής v_j στο ΔΣΜ με ρίζα την v_i. Δηλαδή, το $p[v_j]$ δηλώνει την τελευταία κορυφή πριν τη v_j στο συντομότερο $v_i - v_j$ μονοπάτι. Για να αναπαραστήσουμε n ΔΣΜ, χρησιμοποιούμε έναν πίνακα P με $n \times n$ στοιχεία. Τα στοιχεία της i-οστής γραμμής του πίνακα P αναπαριστούν το ΔΣΜ με ρίζα την κορυφή v_i, $i = 1, \ldots, n$. Συγκεκριμένα, για κάθε ζεύγος κορυφών (v_i, v_j), το στοιχείο $P[v_i, v_j]$ δηλώνει την τελευταία κορυφή πριν τη v_j στο συντομότερο $v_i - v_j$ μονοπάτι. Αν η v_j δεν είναι προσπελάσιμη από τη v_i ή οι κορυφές v_i και v_j ταυτίζονται (δηλ. $i = j$), το στοιχείο $P[v_i, v_j]$ έχει την τιμή NIL.

Μπορούμε εύκολα να υπολογίσουμε τον πίνακα P με τον αλγόριθμο Floyd-Warshall. Η ιδέα είναι να υπολογίσουμε μια ακολουθία πινάκων $P_0, P_1, \ldots, P_n$ που βρίσκεται σε αντιστοιχία με τις αποστάσεις των πινάκων $D_0, D_1, \ldots, D_n$. Συγκεκριμένα, ο πίνακας P_k περιέχει τα ΔΣΜ για όλες τις αρχικές κορυφές v_i που έχουν ενδιάμεσες κορυφές μόνο από το σύνολο $V_k = \{v_1, \ldots, v_k\}$.

Ο αρχικός πίνακας P_0 αποτελεί μια εναλλακτική μορφή του πίνακα γειτνίασης του γραφήματος. Συγκεκριμένα, το στοιχείο $P_0[v_i, v_j]$ είναι ίσο με v_i όταν η ακμή (v_i, v_j) υπάρχει στο γράφημα, και είναι ίσο με NIL όταν $i = j$ ή $(v_i, v_j) \notin E$. Τυπικά,

$$P_0[v_i, v_j] = \begin{cases} \text{NIL} & \text{αν } i = j \text{ ή } (v_i, v_j) \notin E \\ v_i & \text{διαφορετικά} \end{cases}$$

Ο υπολογισμός των στοιχείων του πίνακα P_k από τα στοιχεία του πίνακα P_{k-1} ακολουθεί τον υπολογισμό των αποστάσεων του πίνακα D_k από τις αποστάσεις του πίνακα D_{k-1}. Αν $D_{k-1}[v_i, v_j] \leq D_{k-1}[v_i, v_k] + D_{k-1}[v_k, v_j]$, το μονοπάτι που αντιστοιχεί στο $D_k[v_i, v_j]$ δεν διέρχεται από την κορυφή v_k. Σε αυτή την περίπτωση, θέτουμε $P_k[v_i, v_j] = P_{k-1}[v_i, v_j]$ αφού το συντομότερο $v_i - v_j$ μονοπάτι (άρα και ο πατέρας της v_j στο αντίστοιχο ΔΣΜ) δεν μεταβάλλεται όταν θεωρήσουμε την v_k σαν ενδιάμεση κορυφή. Αν $D_{k-1}[v_i, v_j] > D_{k-1}[v_i, v_k] + D_{k-1}[v_k, v_j]$, το μονοπάτι που αντιστοιχεί στο $D_k[v_i, v_j]$ διέρχεται από την κορυφή v_k. Θέτουμε λοιπόν $P_k[v_i, v_j] = P_{k-1}[v_k, v_j]$ επειδή το καταληκτικό τμήμα του νέου συντομότερου $v_i - v_j$ μονοπατιού είναι το συντομότερο $v_k - v_j$ μονοπάτι που περιγράφεται στο πίνακα P_{k-1}. Επομένως, ο πατέρας της v_j στο νέο συντομότερο $v_i - v_j$ μονοπάτι είναι ίδιος με τον πατέρα της v_j στο συντομότερο $v_k - v_j$ μονοπάτι που υπολογίστηκε στην προηγούμενη φάση του αλγόριθμου.

Από τα παραπάνω προκύπτει ότι τα στοιχεία του πίνακα P_k μπορούν να υπολογιστούν από τα στοιχεία του πίνακα P_{k-1} με βάση την ακόλουθη αναδρομική σχέση. Τελικά, τα συντομότερα μονοπάτια για όλα τα ζεύγη κορυφών δίνονται από τον πίνακα P_n.

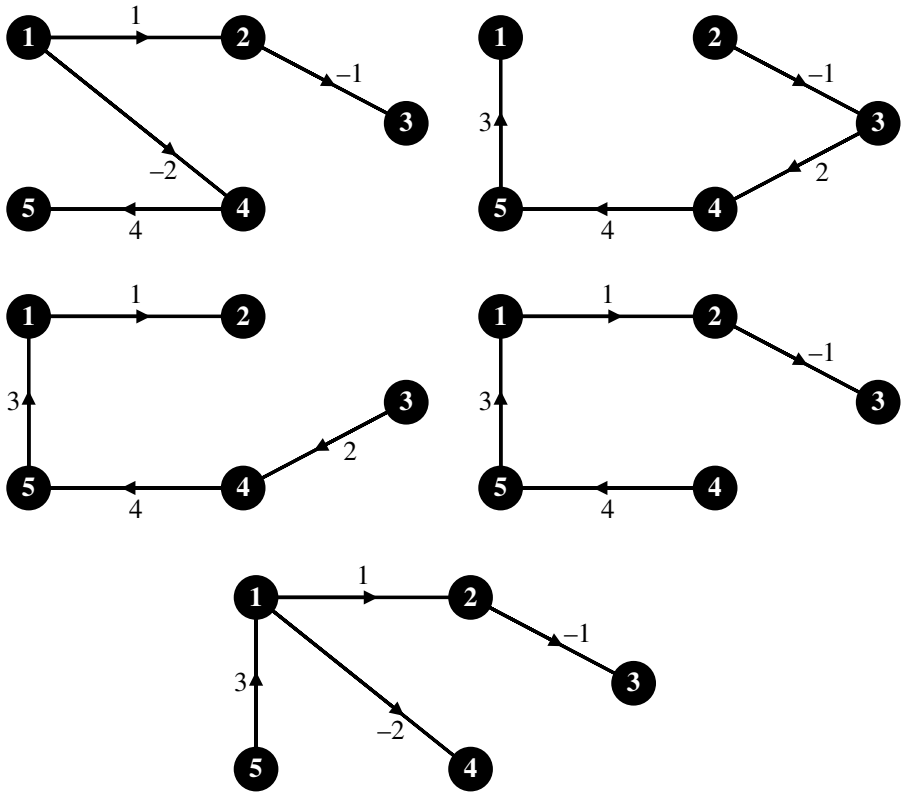
$$P_k[v_i, v_j] = \begin{cases} P_{k-1}[v_i, v_j] & \text{αν } D_{k-1}[v_i, v_j] \leq D_{k-1}[v_i, v_k] + D_{k-1}[v_k, v_j] \\ P_{k-1}[v_k, v_j] & \text{αν } D_{k-1}[v_i, v_j] > D_{k-1}[v_i, v_k] + D_{k-1}[v_k, v_j] \end{cases}$$

Η αρχικοποίηση του πίνακα P_0 και ο υπολογισμός των στοιχείων του πίνακα P_k από τα στοιχεία του P_{k-1} μπορεί εύκολα να ενσωματωθεί στην αρχική διατύπωση του αλγόριθμου Floyd-Warshall. Η χρονική πολυπλοκότητα του αλγόριθμου παραμένει $\Theta(n^3)$. Οι απαιτήσεις σε μνήμη παραμένουν $\Theta(n^2)$ γιατί ο υπολογισμός του P_k απαιτεί μόνο τα στοιχεία του P_{k-1}.

Άσκηση 5.33. Να τροποποιήσετε τον ψευδοκώδικα του αλγόριθμου Floyd-Warshall ώστε να υπολογίζει επίσης τα Συντομότερα Μονοπάτια για Όλα τα Ζεύγη Κορυφών και να τα αποθηκεύει στον πίνακα P_n. Να υπολογίσετε την ακολουθία πινάκων $P_0, P_1, \ldots, P_5$ και τα ΔΣΜ που αντιστοιχούν στον πίνακα P_5 για το γράφημα του Σχήματος 5.20.

Λύση. Οι πίνακες $P_0, P_1, \ldots, P_5$ για το γράφημα του Σχήματος 5.20 φαίνονται στο Σχήμα 5.23. Τα ΔΣΜ που αντιστοιχούν στον πίνακα P_5 φαίνονται στο Σχήμα 5.22. □

Άσκηση 5.34. Να εφαρμόσετε τον αλγόριθμο Floyd-Warshall στο γράφημα του Σχήματος 5.15.



Σχήμα 5.22: Τα Δέντρα Συντομότερων Μονοπατιών για το γράφημα του Σχήματος 5.20.

5.8 Βιβλιογραφικές Αναφορές

Τα βιβλία [21, 24, 12] αποτελούν εξαιρετική εισαγωγή στη Θεωρία Γραφημάτων. Το [21] αποτελεί ακόμη και σήμερα το κλασικό εισαγωγικό βιβλίο στη Θεωρία Γραφημάτων. Το [24] είναι μεταγενέστερο και καλύπτει επίσης αλγοριθμικά θέματα και εφαρμογές της Θεωρίας Γραφημάτων. Το [12] είναι πρόσφατο και καλύπτει όλες τις τελευταίες εξελίξεις στη Θεωρία Γραφημάτων.

Το [19] αποτελεί το πρώτο βιβλίο για αλγόριθμους γραφημάτων και δικτύων. Μια εξαιρετική εισαγωγή στους αλγόριθμους γραφημάτων υπάρχει στο [11], όπου ο αναγνώστης μπορεί να βρει πολλά παραδείγματα και ασκήσεις. Άλλα παραδείγματα βιβλίων που συζητούν εκτενώς και σε βάθος πολλούς αλγόριθμους γραφημάτων είναι τα [17, 41, 32, 2]. Το [17] παρουσιάζει σε βάθος πολλούς αλγόριθμους γραφημάτων. Το [41] αναλύει λεπτομερώς πολλά ζητήματα που σχετίζονται με το σχεδιασμό και την αποδοτική υλοποίηση αλγόριθμων γραφημάτων. Το [41] δίνει έμφαση στις δομές δεδομένων που επιτρέπουν την πλέον αποδοτική υλοποίηση των αλγορίθμων. Το [32] αποτελεί ένα κλασικό βιβλίο στην περιοχή της συνδυαστικής βελτιστοποίησης. Η παρουσίαση των αλγόριθμων στο [32] βασίζεται στο παράδειγμα του Γραμμικού Προγραμματισμού (Linear Programming). Το [32] πραγματεύεται ακόμη ζητήματα όπως η μέθοδος της απληστίας, ο δυναμικός προγραμματισμός, η πληρότητα για την κλάση **NP**, και οι αλγόριθμοι προσέγγισης. Το [2] αποτελεί ένα εξαιρετικό βιβλίο που καλύπτει όλο το φάσμα των αλγόριθμων γραφημάτων και δικτύων. Η παρουσίαση των αλγόριθμων στο [2] είναι πλήρης, κατανοητή, και σε βάθος. Εκτός από τη θεωρία και τις βασικές ιδέες πίσω από τους αλγόριθμους, το [2] παρουσιάζει πολλές εφαρμογές

$$P_0 = \begin{pmatrix} \text{NIL} & 1 & \text{NIL} & 1 & \text{NIL} \\ \text{NIL} & \text{NIL} & 2 & 2 & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 3 & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} & 4 \\ 5 & 5 & \text{NIL} & \text{NIL} & \text{NIL} \end{pmatrix} \qquad P_1 = \begin{pmatrix} \text{NIL} & 1 & \text{NIL} & 1 & \text{NIL} \\ \text{NIL} & \text{NIL} & 2 & 2 & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 3 & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} & 4 \\ 5 & 1 & \text{NIL} & 1 & \text{NIL} \end{pmatrix}$$

$$P_2 = \begin{pmatrix} \text{NIL} & 1 & 2 & 1 & \text{NIL} \\ \text{NIL} & \text{NIL} & 2 & 2 & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 3 & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} & 4 \\ 5 & 1 & 2 & 1 & \text{NIL} \end{pmatrix} \qquad P_3 = \begin{pmatrix} \text{NIL} & 1 & 2 & 1 & \text{NIL} \\ \text{NIL} & \text{NIL} & 2 & 3 & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 3 & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} & 4 \\ 5 & 1 & 2 & 1 & \text{NIL} \end{pmatrix}$$

$$P_4 = \begin{pmatrix} \text{NIL} & 1 & 2 & 1 & 4 \\ \text{NIL} & \text{NIL} & 2 & 3 & 4 \\ \text{NIL} & \text{NIL} & \text{NIL} & 3 & 4 \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} & 4 \\ 5 & 1 & 2 & 1 & \text{NIL} \end{pmatrix} \qquad P_5 = \begin{pmatrix} \text{NIL} & 1 & 2 & 1 & 4 \\ 5 & \text{NIL} & 2 & 3 & 4 \\ 5 & 1 & \text{NIL} & 3 & 4 \\ 5 & 1 & 2 & \text{NIL} & 4 \\ 5 & 1 & 2 & 1 & \text{NIL} \end{pmatrix}$$

Σχήμα 5.23: Η ακολουθία πινάκων $P_0, P_1, \ldots, P_5$ που υπολογίζει ο αλγόριθμος Floyd-Warshall για το γράφημα του Σχήματος 5.20.

και συζητά την αποδοτική υλοποίηση των αλγορίθμων.

Πολλές εφαρμογές της Αναζήτησης Πρώτα σε Πλάτος υπάρχουν στο [1]. Για εφαρμογές της Αναζήτησης Πρώτα σε Βάθος, προτείνονται στον αναγνώστη τα αντίστοιχα κεφάλαια από τα [7, 11] καθώς και το άρθρο [40].

Για το Πρόβλημα του Ελάχιστου Συνδετικού Δέντρου, ο αλγόριθμος του Kruskal παρουσιάστηκε το 1956 [27]. Ο αλγόριθμος του Prim παρουσιάστηκε το 1957 [35], αλλά είχε ήδη εφευρεθεί από τον Jarnik το 1930.

Για το Πρόβλημα των Συντομότερων Μονοπατιών, ο αλγόριθμος του Dijkstra παρουσιάστηκε το 1959 [14]. Ο αλγόριθμος των Bellman-Ford παρουσιάστηκε το 1958 και οφείλεται σε ανεξάρτητες εργασίες των R. Bellman [4] και L. Ford [19]. Ο αλγόριθμος των Floyd-Warshall παρουσιάστηκε επίσης το 1962 και οφείλεται σε ανεξάρτητες εργασίες των R. Floyd [18] και S. Warshall [43]. Μάλιστα, η εργασία του R. Warshall αφορούσε τον υπολογισμό συντομότερων μονοπατιών και συνεκτικών συνιστωσών σε γραφήματα με ακμές μοναδιαίου μήκους υπολογίζοντας τις δυνάμεις του πίνακα γειτνίασης.

6 Υπολογιστική Πολυπλοκότητα

Η Θεωρία Υπολογιστικής Πολυπλοκότητας (Computational Complexity Theory) εξετάζει τους λόγους για τους οποίους μερικά προβλήματα είναι δύσκολο ή αδύνατον να λυθούν από έναν υπολογιστή. Η *υπολογιστική πολυπλοκότητα* (ή απλά πολυπλοκότητα), που ουσιαστικά δεν υπήρχε σαν επιστημονική περιοχή πριν τριάντα χρόνια, γνώρισε ραγδαία ανάπτυξη και σήμερα αποτελεί μία από τις σημαντικότερες περιοχές της Επιστήμης των Υπολογιστών.

Από τη Θεωρία Υπολογισμού, γνωρίζουμε ότι υπάρχουν προβλήματα που δεν μπορούν να λυθούν από έναν υπολογιστή. Ένα τέτοιο παράδειγμα είναι το διάσημο πρόβλημα του τερματισμού μιας μηχανής Turing (Halting Problem).

Από την άλλη πλευρά, υπάρχει η πληθώρα προβλημάτων που μπορούν να λυθούν από τους υπολογιστές. Σε αυτή την κατηγορία εντάσσονται όλα τα προβλήματα που συναντήσαμε μέχρι τώρα. Για αυτά τα προβλήματα, η Θεωρία Υπολογιστικής Πολυπλοκότητας μελετά την ποσότητα των υπολογιστικών πόρων που απαιτούνται για την επίλυση κάθε προβλήματος σε διάφορα αντιπροσωπευτικά υπολογιστικά μοντέλα. Ένα κεντρικό ερώτημα είναι η επίδραση του υπολογιστικού μοντέλου στους πόρους που απαιτεί η επίλυση του προβλήματος.

Για να καθορίσει τους υπολογιστικούς πόρους που απαιτούνται για την επίλυση των προβλημάτων σε διάφορα υπολογιστικά μοντέλα, η θεωρία πολυπλοκότητας εντάσσει τα προβλήματα σε *κλάσεις πολυπλοκότητας* (complexity classes), δηλαδή ομάδες προβλημάτων με παρόμοια συμπεριφορά. Τα προβλήματα κάθε κλάσης *ανάγονται* σε έναν σχετικά μικρό αριθμό προβλημάτων που θεωρούνται αντιπροσωπευτικά για όλη την κλάση. Τα προβλήματα αυτά ονομάζονται *πλήρη* (complete) για την κλάση. Οι μέθοδοι αναγωγής εξασφαλίζουν ότι αν κάποια σημαντική πρόοδος συμβεί στην επίλυση ενός πλήρους προβλήματος, αυτή θα έχει εφαρμογή στην επίλυση όλων των προβλημάτων της κλάσης.

6.1 Αλγόριθμοι και Προβλήματα

Για τη Θεωρία Αλγορίθμων, ένα υπολογιστικό πρόβλημα είναι το μαθηματικό μοντέλο ενός πραγματικού προβλήματος που πρέπει να λυθεί αποδοτικά. Η συντριπτική πλειοψηφία των προβλημάτων που εξετάζονται από τη Θεωρία Αλγορίθμων είναι προβλήματα βελτιστοποίησης. Η περιγραφή του προβλήματος βρίσκεται σε αντιστοιχία με το πραγματικό πρόβλημα. Η περιγραφή των αλγόριθμων σε ψευδοκώδικα επιτρέπει την εύκολη υλοποίησή τους σε διάφορες γλώσσες προγραμματισμού και τη χρήση τους για την επίλυση συγκεκριμένων στιγμιότυπων του προβλήματος. Το γεγονός ότι αλγόριθμοι σχεδιάζονται για να χρησιμοποιηθούν στην πράξη καθιστά αναγκαίο τον ακριβή προσδιορισμό των υπολογιστικών πόρων που απαιτεί η επίλυση ενός στιγμιότυπου.

Για τη Θεωρία Πολυπλοκότητας, τα υπολογιστικά προβλήματα είναι μαθηματικές οντότητες

που καθ' αυτές χρήζουν μελέτης. Τα προβλήματα περιγράφονται σαν τυπικές γλώσσες (formal languages) και οι αλγόριθμοι σαν υπολογιστικές μηχανές ειδικού σκοπού. Ο προσδιορισμός των υπολογιστικών πόρων που απαιτείται για τη λύση ενός προβλήματος δεν χρειάζεται να είναι ακριβής, αφού εκείνο που ενδιαφέρει είναι η κατηγορία των συναρτήσεων και όχι η ακριβής τάξη μεγέθους.

Για τη θεωρία πολυπλοκότητας, ένα πρόβλημα Π είναι μία δυαδική σχέση που περιλαμβάνει όλα τα διαφορετικά ζεύγη των στιγμιότυπων του Π και των λύσεων τους. Σαν παράδειγμα, ας θεωρήσουμε το πρόβλημα του Συντομότερου Μονοπατιού Ζεύγους Κορυφών. Ένα στιγμιότυπο αποτελείται από κατευθυνόμενο γράφημα με μήκη στις ακμές $G(V, E, w)$, και δύο κορυφές $s, t \in V$. Κάθε τέτοια τριάδα αποτελεί ένα έγκυρο στιγμιότυπο του προβλήματος. Κάθε συντομότερο μονοπάτι από την s στην t αποτελεί μία λύση του προβλήματος. Σαν λύση μπορεί να θεωρηθεί επίσης και το κενό μονοπάτι που δηλώνει ότι η t δεν είναι προσπελάσιμη από την s. Επειδή το συντομότερο $s-t$ μονοπάτι δεν είναι μοναδικό, ένα στιγμιότυπο του προβλήματος μπορεί να έχει περισσότερες από μία λύσεις.

Το πρόβλημα του Συντομότερου Μονοπατιού Ζεύγους Κορυφών είναι ένα πρόβλημα βελτιστοποίησης. Δεδομένης μιας τριάδας $(G(V, E, w), s, t)$, ζητείται το συντομότερο $s-t$ μονοπάτι. Κάθε $s-t$ μονοπάτι αποτελεί αποδεκτή λύση. Βέλτιστη λύση είναι κάθε $s-t$ μονοπάτι με ελάχιστο συνολικό μήκος.

Ένας διαφορετικός τρόπος να περιγραφεί το πρόβλημα του Συντομότερου Μονοπατιού είναι να θεωρήσουμε έναν αριθμό - φράγμα B και να εξετάσουμε αν υπάρχει $s-t$ μονοπάτι t με μήκος το πολύ . Ένα στιγμιότυπο καθορίζεται από μία τετράδα $(G(V, E, w), s, t, B)$. Οι δυνατές λύσεις του προβλήματος σε αυτή τη μορφή είναι δύο: ΝΑΙ και ΟΧΙ. Κάθε πρόβλημα που έχει σύνολο λύσεων {ΝΑΙ, ΟΧΙ} ονομάζεται *πρόβλημα απόφασης* (decision problem).

Ορισμός 6.1. Ένα *πρόβλημα* Π είναι μία δυαδική σχέση στο καρτεσιανό γινόμενο του συνόλου των στιγμιότυπων και του συνόλου S των λύσεων, $\Pi \subseteq I \times S$. Ένα ζεύγος $(i, s) \in \Pi$ όταν το i είναι ένα έγκυρο στιγμιότυπο του Π και το s μία λύση του i. Ένα πρόβλημα ονομάζεται *πρόβλημα απόφασης* όταν έχει δύο μόνο λύσεις ΝΑΙ και ΟΧΙ.

Στην υπολογιστική πολυπλοκότητα, ασχολούμαστε κυρίως με προβλήματα απόφασης. Κάθε πρόβλημα βελτιστοποίησης μπορεί να γραφεί σαν πρόβλημα απόφασης με την προσθήκη ενός αριθμού - φράγματος B. Στο πρόβλημα απόφασης ζητάμε την απάντηση στην ερώτηση "Υπάρχει αποδεκτή λύση με κόστος το πολύ B;" προκειμένου για πρόβλημα ελαχιστοποίησης, και "Υπάρχει αποδεκτή λύση με κέρδος τουλάχιστον B;" προκειμένου για πρόβλημα μεγιστοποίησης. Αν ένα πρόβλημα απόφασης μπορεί να λυθεί αποδοτικά (π.χ. σε πολυωνυμικό χρόνο), τότε και το αντίστοιχο πρόβλημα βελτιστοποίησης μπορεί να λυθεί αποδοτικά (π.χ. σε πολυωνυμικό χρόνο), και αντίστροφα.

Ένας από τους σημαντικότερους λόγους για τους οποίους στη Θεωρία Πολυπλοκότητας μελετάμε προβλήματα απόφασης είναι ότι αυτά επιτρέπουν τη χρήση μαθηματικών εργαλείων από την περιοχή των *τυπικών γλωσσών* (formal languages).

Για να λυθεί κάποιο πρόβλημα, πρέπει τα στιγμιότυπα του προβλήματος να αναπαρασταθούν με τρόπο αντιληπτό από τον υπολογιστή. Για το σκοπό αυτό, χρησιμοποιούμε μία *κωδικοποίηση* (encoding) e που αντιστοιχεί κάθε στιγμιότυπο του προβλήματος σε μια μοναδική συμβολοσειρά

$e(i)$ ενός αλφάβητου Σ με τουλάχιστον δύο σύμβολα[1]. Το μήκος του $e(i)$ ονομάζεται *μέγεθος* του στιγμιότυπου i. Μια (τυπική) γλώσσα Λ που ορίζεται σε ένα αλφάβητο Σ είναι ένα υποσύνολο του Σ^*, δηλαδή του συνόλου όλων των πεπερασμένων συμβολοσειρών του Σ συμπεριλαμβανομένης και της κενής συμβολοσειράς. Όπως το ισοδύναμο ενός στιγμιότυπου i είναι η συμβολοσειρά $e(i)$ του Σ, το ισοδύναμο ενός προβλήματος απόφασης Π είναι η γλώσσα $\mathcal{L}(\Pi, e)$ του Σ^*.

Ορισμός 6.2. Δεδομένου ενός προβλήματος απόφασης Π και μιας κωδικοποίησης e για τα στιγμιότυπα του Π, η γλώσσα $\mathcal{L}(\Pi, e)$ είναι το σύνολο των συμβολοσειρών $e(i) \in \Sigma^*$, όπου i είναι ένα στιγμιότυπο του Π με απάντηση NAI.

Στο εξής, θα χρησιμοποιούμε τους όρους *πρόβλημα* και *γλώσσα* σαν ταυτόσημους.

6.1.1 Ευεπίλυτα και Δυσεπίλυτα Προβλήματα

Από τα προβλήματα που μπορούν να λυθούν από τον υπολογιστή, κάποια λύνονται με αποδοτικό τρόπο και θεωρούνται *ευεπίλυτα* (tractable), και κάποια άλλα δεν λύνονται με αποδοτικό τρόπο και θεωρούνται *δυσεπίλυτα* (intractable). Η διάκριση γίνεται ανάλογα με τους υπολογιστικούς πόρους που απαιτούνται για την επίλυση ενός προβλήματος. Όταν υπάρχει ένας αποδοτικός αλγόριθμος (δηλαδή ένας αλγόριθμος που απαιτεί εύλογη ποσότητα υπολογιστικών πόρων) για κάποιο πρόβλημα, τότε αυτό εντάσσεται στην κατηγορία των ευεπίλυτων προβλημάτων.

Όταν για κάποιο πρόβλημα δεν είναι γνωστός κανένας αποδοτικός αλγόριθμος, δεν είναι προφανές αν αυτό πρέπει να ενταχθεί στα δυσεπίλυτα προβλήματα. Ο λόγος είναι ότι μπορεί να υπάρχει αποδοτικός αλγόριθμος για αυτό πρόβλημα, αλλά να μην έχει ανακαλυφθεί ακόμα. Για παράδειγμα, ο πρώτος αλγόριθμος πολυωνυμικού χρόνου για το πρόβλημα του γραμμικού προγραμματισμού ανακαλύφθηκε μόλις στο τέλος της δεκαετίας του 1970. Ένα πρόβλημα είναι δυσεπίλυτο όταν αποδεδειγμένα δεν μπορεί να υπάρξει αποδοτικός αλγόριθμος για αυτό. Υπάρχει όμως και μία μεγάλη κατηγορία σημαντικών προβλημάτων για τα οποία υπάρχουν ισχυρές ενδείξεις, αλλά όχι μαθηματική απόδειξη, ότι είναι δυσεπίλυτα. Η συντριπτική πλειοψηφία, αν όχι το σύνολο, των επιστημόνων θεωρεί ότι και αυτά τα προβλήματα είναι δυσεπίλυτα.

Η κρατούσα αντίληψη εντάσσει στην κατηγορία των ευεπίλυτων προβλημάτων αυτά και μόνο τα προβλήματα που λύνονται σε πολυωνυμικό χρόνο. Ένα πρόβλημα λύνεται σε πολυωνυμικό χρόνο όταν υπάρχει αλγόριθμος του οποίου ο χρόνος εκτέλεσης χειρότερης περίπτωσης είναι $O(n^k)$, όπου n το μέγεθος του στιγμιότυπου εισόδου και k μία οποιαδήποτε σταθερά. Τα προβλήματα που λύνονται σε πολυωνυμικό χρόνο αποτελούν την κλάση πολυπλοκότητας $\mathbf{P}$. Η πεποίθηση ότι η κλάση $\mathbf{P}$ ταυτίζεται με την κλάση των ευεπίλυτων προβλημάτων αποδίδεται στους S. Cook και R. Karp, και μερικές φορές αναφέρεται σαν αξίωμα των Cook - Karp.

Υπάρχουν αρκετά σημαντικά επιχειρήματα που υποστηρίζουν το αξίωμα των Cook - Karp. Κατ' αρχήν, η ταχύτητα με την οποία αυξάνεται ένα πολυώνυμο μικρού βαθμού επιτρέπει την επίλυση αρκετά μεγάλων στιγμιότυπων ενός προβλήματος σε εύλογο χρονικό διάστημα. Επιπλέον, η ύπαρξη ενός αλγόριθμου πολυωνυμικού χρόνου επιτρέπει τη σημαντική αύξηση του μεγέθους των στιγμιότυπων που λύνει ο αλγόριθμος σε δεδομένο χρονικό διάστημα, σαν αποτέλεσμα της

[1] Υποθέτουμε πάντα *εύλογες* κωδικοποιήσεις που χρησιμοποιούν τα σύμβολα του Σ με τον πλέον αποδοτικό τρόπο για να κωδικοποιήσουν τα στιγμιότυπα του προβλήματος (π.χ. κάθε αριθμός x κωδικοποιείται με $\lceil \log_{|\Sigma|} x \rceil$ ψηφία, το $e(i)$ δεν περιέχει πλεονασμό ή άχρηστη πληροφορία, κλπ.).

συνεχούς αύξησης της ταχύτητας των υπολογιστών. Τα παραπάνω δεν ισχύουν για συναρτήσεις που μεγαλώνουν πολύ γρήγορα (π.χ. εκθετικές συναρτήσεις).

Από την άλλη πλευρά, ένα πρόβλημα που λύνεται σε χρόνο $\Theta(n^{100})$ δεν μπορεί να χαρακτηριστεί ευεπίλυτο, σε αντίθεση με ένα πρόβλημα που λύνεται σε χρόνο $\Theta(2^{n/100})$. Όμως πολύ λίγα πρακτικά προβλήματα απαιτούν χρόνο επίλυσης που δίνεται από ένα πολυώνυμο τόσο μεγάλου βαθμού. Η συντριπτική πλειοψηφία των προβλημάτων που ανήκουν στην κλάση $\mathbf{P}$ λύνονται σε χρόνο που φράσσεται από ένα πολυώνυμο μικρού βαθμού (π.χ. 1, 2, 3).

Επιπλέον, το σύνολο των πολυωνύμων είναι κλειστό ως προς τις πράξεις της πρόσθεσης, του πολλαπλασιασμού, και της σύνθεσης. Επομένως η κλάση των προβλημάτων που λύνονται σε πολυωνυμικό χρόνο έχει αντίστοιχες ιδιότητες κλειστότητας. Για παράδειγμα, αν συνθέσουμε δύο αλγόριθμους πολυωνυμικού χρόνου, χρησιμοποιώντας την έξοδο του ενός σαν είσοδο του άλλου, το αποτέλεσμα θα είναι ένας αλγόριθμος πολυωνυμικού χρόνου.

Άσκηση 6.1. Να ορίσετε τα προβλήματα απόφασης που αντιστοιχούν στο πρόβλημα του Πολλαπλασιασμού Ακολουθίας Πινάκων και στο πρόβλημα του Ελάχιστου Συνδετικού Δέντρου. Είναι αυτά τα προβλήματα ευεπίλυτα ή δυσεπίλυτα;

6.2 Ντετερμινιστικές Μηχανές Turing

Παρά την μάλλον άχαρη εικόνα τους, οι μηχανές Turing μπορούν να προσομοιώσουν οποιοδήποτε αλγόριθμο χωρίς σημαντική υποβάθμιση της απόδοσής του. Οι μηχανές Turing θα αποτελέσουν το βασικό υπολογιστικό μοντέλο για την σύντομη εισαγωγή μας στη Θεωρίας Πολυπλοκότητας. Συγκεκριμένα, θα θεωρήσουμε σαν υπολογιστικό μοντέλο αυτό των ντετερμινιστικών μηχανών Turing με πολλαπλές ταινίες (Multitape Deterministic Turing Machines).

Μια τέτοια μηχανή Turing έχει $k \geq 1$ ταινίες από τις οποίες μπορεί να διαβάσει και να γράψει. Κάθε *ταινία* (tape) εκτείνεται απεριόριστα προς τα δεξιά και αποτελείται από *κελιά* (cells) τα οποία αριθμούνται από τα αριστερά προς τα δεξιά. Ένα κελί είτε είναι κενό (blank) είτε περιέχει κάποιο στοιχείο ενός πεπερασμένου αλφάβητου Σ, το οποίο ονομάζεται *αλφάβητο εισόδου* ή απλά αλφάβητο. Κάθε ταινία έχει μία *κεφαλή* (tape head) η οποία κινείται κατά μήκος της ταινίας και διαβάζει ή γράφει στο κελί πάνω από το οποίο βρίσκεται. Κάθε χρονική στιγμή, η μηχανή Turing βρίσκεται σε μία *κατάσταση* (state). Το σύνολο των επιτρεπτών καταστάσεων είναι πεπερασμένο και συμβολίζεται με Q.

Οι καταστάσεις της μηχανής Turing και οι ενέργειες των κεφαλών καθορίζονται από ένα πεπερασμένο σύνολο εντολών, το οποίο ονομάζεται *συνάρτηση μετάβασης* (transition function). Δεδομένης μιας κατάστασης και των περιεχόμενων των κελιών στα οποία βρίσκονται οι κεφαλές, η συνάρτηση μετάβασης προσδιορίζει την επόμενη κατάσταση, το σύμβολο που θα γράψει κάθε κεφαλή στο κελί που βρίσκεται, και την κίνηση κάθε κεφαλής. Όσον αφορά στις κινήσεις μιας κεφαλής, αυτή μπορεί να μετακινηθεί ένα κελί δεξιά (συμβολίζεται με R), ή ένα κελί αριστερά (συμβολίζεται με L), ή να μείνει στάσιμη (συμβολίζεται με S). Ο υπολογισμός μιας μηχανής Turing ξεκινάει από μία ειδική *αρχική κατάσταση* (initial state) και τερματίζεται όταν η μηχανή φτάσει σε κάποιες ειδικές *τελικές καταστάσεις* (final state).

Ορισμός 6.3 (Ντετερμινιστική Μηχανή Turing). Έστω ακέραιος $k \geq 1$. Μία *Ντετερμινιστική Μηχανή Turing* M με k ταινίες είναι μία διατεταγμένη πεντάδα $M = (Q, \Sigma, \delta, q_0, F)$ όπου:

- Q είναι ένα πεπερασμένο σύνολο καταστάσεων. Το Q περιέχει την αρχική κατάσταση q_0 και το σύνολο F των τελικών καταστάσεων.
- Σ είναι ένα πεπερασμένο αλφάβητο που ονομάζεται αλφάβητο εισόδου. Το λεγόμενο αλφάβητο ταινίας Γ αποτελείται από τα σύμβολα του Σ και το διακεκριμένο σύμβολο $\sqcup$ που δηλώνει το κενό σύμβολο.
- $q_0 \in Q$ είναι η αρχική κατάσταση.
- $F \subseteq Q$ είναι το σύνολο των τελικών καταστάσεων.
- $\delta : (Q \setminus F) \times \Gamma^k \mapsto Q \times \Gamma^k \times \{L, R, S\}^k$ είναι η συνάρτηση μετάβασης.

Το αριστερό άκρο μιας ταινίας συμβολίζεται με το σύμβολο $\triangleright$. Μια παραδοχή είναι ότι αν η μηχανή Turing προσπαθήσει να κινήσει μια κεφαλή πέραν του $\triangleright$, η κεφαλή θα μείνει στη θέση της ακόμα και αν η συνάρτηση μετάβασης ορίζει ότι πρέπει να κινηθεί προς τα αριστερά. Θεωρούμε πάντα ότι η πρώτη ταινία είναι η *ταινία εισόδου* (input tape).

Η συνάρτηση μετάβασης δ αποτελεί το *πρόγραμμα* της μηχανής Turing. Αφού τα υπολογιστικά βήματα μιας Ντετερμινιστικής Μηχανής Turing (DTM) καθορίζονται από τη συνάρτηση μετάβασης, η επόμενη διαμόρφωση της μηχανής προσδιορίζεται από την τρέχουσα διαμόρφωσή της. Επομένως, ο υπολογισμός και τελικά η απάντηση / έξοδος μιας DTM για δεδομένη είσοδο προσδιορίζεται *μονοσήμαντα* (ή *ντετερμινιστικά*) από τη συνάρτηση μετάβασης και την είσοδο.

Μία Ντετερμινιστική Μηχανή Turing M ξεκινάει τον υπολογισμό από την αρχική κατάσταση q_0. Αρχικά όλες οι ταινίες περιέχουν μόνο κενά σύμβολα ($\sqcup$), εκτός από την ταινία εισόδου που στην αρχή της περιέχει τη *συμβολοσειρά εισόδου* $x \in \Sigma^*$. Η συμβολοσειρά x οριοθετείται από το πρώτο κενό σύμβολο στην ταινία εισόδου.

Ο υπολογισμός της M τερματίζει μόνο όταν η τρέχουσα κατάσταση γίνει μία από τις τελικές καταστάσεις του συνόλου F. Υπάρχουν τρεις τελικές καταστάσεις: το ΝΑΙ που αντιστοιχεί στην *αποδοχή* της εισόδου, το ΟΧΙ που αντιστοιχεί στην *απόρριψη*, και το ΤΕΛΟΣ (halt) που δηλώνει την ολοκλήρωση του υπολογισμού. Όταν η μηχανή Turing επεξεργάζεται ένα πρόβλημα απόφασης, είναι $F = \{\text{ΝΑΙ}, \text{ΟΧΙ}\}$. Όταν η μηχανή Turing υπολογίζει μια συνάρτηση, είναι $F = \{\text{ΤΕΛΟΣ}\}$.

Αν η τελική κατάσταση της M με είσοδο τη συμβολοσειρά x είναι ΝΑΙ, γράφουμε $M(x) = \text{ΝΑΙ}$ και λέμε ότι η μηχανή M *αποδέχεται* (accepts) το x. Αν η τελική κατάσταση είναι ΟΧΙ, γράφουμε ότι $M(x) = \text{ΟΧΙ}$ και λέμε ότι η μηχανή M *απορρίπτει* (rejects) το x. Το σύνολο των συμβολοσειρών $x \in \Sigma^*$ που γίνονται αποδεκτές από την M αποτελεί τη *γλώσσα που γίνεται αποδεκτή από την* M και συμβολίζεται με $\mathcal{L}(M)$.

Αν η τελική κατάσταση είναι ΤΕΛΟΣ, το αποτέλεσμα του υπολογισμού βρίσκεται στην αρχή της τελευταίας ταινίας, η οποία θεωρείται σαν *ταινία εξόδου* (output tape). Το αποτέλεσμα της M είναι μια συμβολοσειρά $y \in \Sigma^*$ που αρχίζει από το αριστερό άκρο της ταινίας εξόδου και οριοθετείται από το πρώτο κενό σύμβολο. Σε αυτή την περίπτωση γράφουμε $M(x) = y$.

Υπάρχει ακόμη περίπτωση η M με είσοδο τη συμβολοσειρά x να μην φτάνει ποτέ σε κάποια τελική κατάσταση. Τότε λέμε ότι η $M(x)$ δεν *τερματίζει*.

Παράδειγμα 6.1. Θα περιγράψουμε μια Ντετερμινιστική Μηχανή Turing με μία ταινία που μετατοπίζει την είσοδό της μία θέση προς τα δεξιά γράφοντας το κενό σύμβολο στο πρώτο κελί. Για

παράδειγμα, αν αρχικά η ταινία εισόδου περιέχει το 1101, στο τέλος του υπολογισμού πρέπει να περιέχει το $\sqcup 1101$.

Έστω $M = (Q, \Sigma, \delta, q_0, F)$. Για ευκολία, υποθέτουμε ότι το αλφάβητο εισόδου είναι $\Sigma = \{0, 1\}$, οπότε $\Gamma = \{0, 1, \sqcup\}$. Η αρχική κατάσταση είναι q_0, οι τελικές καταστάσεις είναι $F = \{\text{ΤΕΛΟΣ}\}$ και το σύνολο των καταστάσεων είναι $Q = \{q_0, s, s_0, s_1\} \cup F$. Η συνάρτηση μετάβασης φαίνεται στον παρακάτω πίνακα:

$p \in Q \setminus F$	$\sigma \in \Gamma$	$\delta(p, \sigma)$
q_0	$\triangleright$	$(q_0, \triangleright, R)$
q_0	0	$(q_0, 0, R)$
q_0	1	$(q_0, 1, R)$
q_0	$\sqcup$	$(s, \sqcup, L)$
s	$\triangleright$	$(\text{ΤΕΛΟΣ}, \triangleright, R)$
s	0	$(s_0, \sqcup, R)$
s	1	$(s_1, \sqcup, R)$
s	$\sqcup$	$(s, \sqcup, S)$
s_0	$\triangleright$	$(\text{ΤΕΛΟΣ}, \triangleright, R)$
s_0	0	$(q_0, 0, L)$
s_0	1	$(q_0, 0, L)$
s_0	$\sqcup$	$(q_0, 0, L)$
s_1	$\triangleright$	$(\text{ΤΕΛΟΣ}, \triangleright, R)$
s_1	0	$(q_0, 1, L)$
s_1	1	$(q_0, 1, L)$
s_1	$\sqcup$	$(q_0, 1, L)$

Ακολουθεί ένα παράδειγμα λειτουργίας για είσοδο τη συμβολοσειρά 1101 όπου πρώτα αναγράφεται η τρέχουσα κατάσταση της μηχανής και στη συνέχεια το περιεχόμενο της ταινίας. Το υπογραμμισμένο σύμβολο δηλώνει την τρέχουσα θέση της κεφαλής.

$$
\begin{array}{ccccccccccc}
(q_0, \underline{\triangleright}1101) & \to & (q_0, \triangleright\underline{1}101) & \to & (q_0, \triangleright 1\underline{1}01) & \to & (q_0, \triangleright 11\underline{0}1) & \to & (q_0, \triangleright 110\underline{1}) & \to \\
(q_0, \triangleright 1101\underline{\sqcup}) & \to & (s, \triangleright 110\underline{1}\sqcup) & \to & (s_1, \triangleright 110\sqcup\underline{\sqcup}) & \to & (q_0, \triangleright 110\underline{\sqcup}1) & \to & (s, \triangleright 11\underline{0}\sqcup 1) & \to \\
(s_0, \triangleright 11\sqcup\underline{\sqcup}1) & \to & (q_0, \triangleright 11\underline{\sqcup}01) & \to & (s, \triangleright 1\underline{1}\sqcup 01) & \to & (s_1, \triangleright 1\sqcup\underline{\sqcup}01) & \to & (q_0, \triangleright 1\underline{\sqcup}101) & \to \\
(s, \triangleright\underline{1}\sqcup 101) & \to & (s_1, \triangleright\sqcup\underline{\sqcup}101) & \to & (q_0, \triangleright\underline{\sqcup}1101) & \to & (s, \underline{\triangleright}\sqcup 1101) & \to & (\text{ΤΕΛΟΣ}, \triangleright\underline{\sqcup}1101) &
\end{array}
$$

6.2.1 Καθολικές Μηχανές Turing

Σε αντίθεση με τη Μηχανή Άμεσης Προσπέλασης Μνήμης, που είναι προγραμματιζόμενη, η μηχανή Turing μπορεί να εκτελέσει μόνο το πρόγραμμα που καθορίζεται από τη συνάρτηση μετάβασης (μοντέλο υπολογιστή καθορισμένου προγράμματος). Αυτό δεν συνιστά ουσιαστικό περιορισμό, αφού μπορούμε να περιγράψουμε μία *καθολική μηχανή Turing* (universal Turing machine), που προσομοιώνει τη λειτουργία κάθε άλλης μηχανής.

Συγκεκριμένα, μια καθολική DTM U δέχεται σαν είσοδο την κωδικοποίηση μιας οποιασδήποτε DTM M και τη συμβολοσειρά εισόδου x της M. Αν η $M(x)$ τερματίζει, η $U(M; x)$ τερματίζει στην ίδια κατάσταση και με τα ίδια περιεχόμενα στις ταινίες της όπως η $M(x)$. Αυτό το δηλώνουμε γράφοντας $U(M; x) = M(x)$. Αν η $M(x)$ δεν τερματίζει, ούτε η $U(M; x)$ τερματίζει.

6.3 Υπολογισιμότητα

Μια γλώσσα $\mathcal{L}$ είναι DTM-*ημιαποφασίσιμη* (semidecidable) αν υπάρχει μια DTM M που τερματίζει με είσοδο όλες τις συμβολοσειρές της $\mathcal{L}$ και μόνο αυτές. Δηλαδή, για κάθε $x \in \mathcal{L}$, $M(x)$ τερματίζει, και για κάθε $x \notin \mathcal{L}$, η $M(x)$ δεν τερματίζει.

Μια γλώσσα $\mathcal{L}$ ορισμένη σε ένα αλφάβητο Σ είναι DTM-*αποφασίσιμη* (decidable) αν υπάρχει μια DTM M που αποδέχεται όλες τις συμβολοσειρές της $\mathcal{L}$ και απορρίπτει όλες τις συμβολειρές της $\Sigma^* \setminus \mathcal{L}$ [2]. Δηλαδή, για κάθε $x \in \mathcal{L}$, $M(x) =$ ΝΑΙ, και για κάθε $x \notin \mathcal{L}$, $M(x) =$ ΟΧΙ.

Για να είναι μια γλώσσα *αποφασίσιμη* πρέπει να υπάρχει μηχανή Turing που για κάθε συμβολοσειρά $x \in \Sigma^*$, απαντάει σωστά στο ερώτημα "$x \in \mathcal{L}$;" (η μηχανή πρέπει να τερματίζει πάντα και να δίνει τη σωστή απάντηση). Για να είναι μια γλώσσα *ημιαποφασίσιμη* αρκεί να υπάρχει μια μηχανή Turing που αναγνωρίζει σωστά τις συμβολοσειρές της $\mathcal{L}$. Για τις συμβολοσειρές που δεν ανήκουν στην $\mathcal{L}$, αρκεί η μηχανή Turing να μην τερματίζει (οπότε δεν απαντάει καθόλου).

Μια DTM M *υπολογίζει* (computes) μια μερική συνάρτηση $f : \Sigma^* \mapsto \Sigma^*$ όταν για κάθε $x \in \Sigma^*$

- αν $f(x) = y$, είναι $M(x) = y$, ενώ
- αν το $f(x)$ δεν ορίζεται, η $M(x)$ δεν τερματίζει.

Μια μερική συνάρτηση f είναι DTM-υπολογίσιμη αν υπάρχει μια DTM που την υπολογίζει. Η έννοια της υπολογισιμότητας μιας συνάρτησης είναι αντίστοιχη με αυτή της αποφασισιμότητας μιας γλώσσας επειδή η M πρέπει να υπολογίζει σωστά το αποτέλεσμα της συνάρτησης για όλα τα σημεία που αυτή ορίζεται.

Το αξίωμα των Church - Turing πρεσβεύει ότι κάθε γλώσσα (συνάρτηση) που μπορεί να αποφασισθεί (υπολογιστεί) από αλγόριθμο σε οποιοδήποτε υπολογιστικό μοντέλο μπορεί να αποφασισθεί (υπολογιστεί) από μια Ντετερμινιστική Μηχανή Turing. Το αξίωμα έχει αποδειχθεί για όλα τα υπολογιστικά μοντέλα που έχουν κατά καιρούς προταθεί (π.χ. λ-definability του Church, μ-αναδρομικές συναρτήσεις του Godel, γενικές αναδρομικές συναρτήσεις των Herbrand - Godel, κανονικά συστήματα του Post, αλγόριθμοι του Markov). Επίσης κάθε συστηματική προσπάθεια για να αυξηθεί η κλάση των γλωσσών (συναρτήσεων) που μπορούν να αποφασιστούν (υπολογισθούν) από τις Ντετερμινιστικές Μηχανές Turing δεν είχε επιτυχία. Το αξίωμα των Church - Turing τεκμηριώνει την επιλογή των Ντετερμινιστικών Μηχανών Turing σαν βασικό υπολογιστικό μοντέλο για να αναπτύξουμε της Θεωρία Υπολογιστικής Πολυπλοκότητας.

Άσκηση 6.2. Να αποδείξετε τις ακόλουθες προτάσεις:

1. Κάθε DTM-αποφασίσιμη γλώσσα είναι DTM-ημιαποφασίσιμη.
2. Αν μια γλώσσα $\mathcal{L}$ είναι DTM-αποφασίσιμη, το συμπλήρωμά της $\overline{\mathcal{L}}$ είναι επίσης DTM-αποφασίσιμο.
3. Μια γλώσσα $\mathcal{L}$ είναι DTM-αποφασίσιμη αν και μόνο αν τόσο η $\mathcal{L}$ όσο και το συμπλήρωμά της $\overline{\mathcal{L}}$ είναι DTM-ημιαποφασίσιμες γλώσσες.

[2] Η γλώσσα $\Sigma^* \setminus \mathcal{L}$ που περιέχει όλες τις συμβολοσειρές που δεν ανήκουν στην $\mathcal{L}$ ονομάζεται *συμπλήρωμα* (complement) της $\mathcal{L}$ και συμβολίζεται με $\overline{\mathcal{L}}$.

Λύση. Το (1) ισχύει εξ' ορισμού (αν μία DTM αποφασίζει μία γλώσσα, την ημιαποφασίζει επίσης). Για το (2), ανταλλάσσουμε αμοιβαία τις τελικές καταστάσεις NAI και OXI στην DTM που αποφασίζει την $\mathcal{L}$. Το αποτέλεσμα είναι μία DTM που αποφασίζει το συμπλήρωμα $\overline{\mathcal{L}}$.

Για το (3), τα (1) και (2) αποφασίζουν τη μία κατεύθυνση. Για το αντίστροφο, έστω DTM M που ημιαποφασίζει την $\mathcal{L}$ και DTM $\overline{M}$ που ημιαποφασίζει την $\overline{\mathcal{L}}$. Μπορούμε να φτιάξουμε μια DTM M' που αποφασίζει την $\mathcal{L}$ εκτελώντας παράλληλα τις M και $\overline{M}$ (ένα βήμα της M, μετά ένα βήμα της $\overline{M}$, κοκ.). Η τελική κατάσταση της $M'(x)$ είναι NAI αν η $M(x)$ τερματίζει και OXI αν η $\overline{M}(x)$ τερματίζει. Αφού κάθε συμβολοσειρά $x \in \Sigma^*$ είτε ανήκει στην $\mathcal{L}$ είτε δεν ανήκει (και ανήκει στην $\overline{\mathcal{L}}$), ακριβώς μία από τις $M(x)$ και $\overline{M}(x)$ θα τερματίσει. □

6.3.1 Μη-Υπολογισιμότητα: Το Πρόβλημα του Τερματισμού

Υπάρχουν γλώσσες που δεν μπορούν να αποφασισθούν από τις Ντετερμινιστικές Μηχανές Turing. Ο λόγος είναι απλός: υπάρχουν πολύ περισσότερες γλώσσες από τους τρόπους να τις αποφασίσουμε (διαφορετικές μηχανές Turing). Συγκεκριμένα, υπάρχουν μη-αριθμήσιμα (uncountably) άπειρες διαφορετικές γλώσσες και μόλις αριθμήσιμα (countably) άπειρες διαφορετικές μηχανές Turing. Με βάση το αξίωμα των Church - Turing, οι μη-αποφασίσιμες γλώσσες αντιστοιχούν σε προβλήματα που δεν μπορούν να λυθούν από τους υπολογιστές (ανεξαρτήτως των διαθέσιμων υπολογιστικών πόρων).

Κεντρική θέση ανάμεσα σε αυτά τα προβλήματα κατέχει το πρόβλημα του τερματισμού μιας μηχανής Turing (Halting Problem), όπου δίνονται μία DTM M και η συμβολοσειρά εισόδου x, και πρέπει να αποφασίσουμε αν η $M(x)$ τερματίζει.

Θεώρημα 6.1. *Το πρόβλημα του Τερματισμού είναι μη-αποφασίσιμο (undecidable).*

Απόδειξη. Η απόδειξη γίνεται με απαγωγή σε άτοπο χρησιμοποιώντας την τεχνική της διαγωνιοποίησης. Ας υποθέσουμε ότι υπάρχει μία DTM H που δέχεται σαν είσοδο μια DTM M και μια συμβολοσειρά x, και αποφασίζει αν η $M(x)$ τερματίζει ή όχι. Δηλαδή είναι $H(M; x) = \text{NAI}$ αν η $M(x)$ τερματίζει, και $H(M; x) = \text{OXI}$ διαφορετικά.

Με βάση την H, ορίζουμε μία νέα DTM J που λειτουργεί ως εξής:

- Η J με είσοδο την κωδικοποίηση της M (δηλαδή η $J(M)$) προσομοιώνει τη λειτουργία της $H(M; M)$ μέχρι το σημείο που η H βρίσκεται ένα βήμα πριν τον τερματισμό. Αφού η H αποφασίζει το προβλήμα του Τερματισμού, πρέπει να τερματίζει πάντα.
- Αν $H(M; M) = \text{NAI}$, η $J(M)$ δεν τερματίζει.
- Αν $H(M; M) = \text{OXI}$, η $J(M)$ τερματίζει.

Με απλά λόγια, η $J(M)$ τερματίζει αν και μόνο αν η $M(M)$ δεν τερματίζει. Η αντίφαση προκύπτει εξετάζοντας τη λειτουργία της J με είσοδο τον εαυτό της (δηλαδή το $J(J)$). Εξ' ορισμού, η $J(J)$ τερματίζει αν και μόνο αν η $J(J)$ δεν τερματίζει. □

Είναι γνωστές αρκετές ακόμη γλώσσες που δεν είναι DTM-αποφασίσιμες. Η βασική τεχνική για να αποδείξουμε ότι μία γλώσσα δεν είναι DTM-αποφασίσιμη είναι με αναγωγή στο πρόβλημα του τερματισμού.

6.4 Χρονική Πολυπλοκότητα - Η Κλάση P

Η υπολογιστική πολυπλοκότητα ενός προβλήματος ορίζεται με βάση την υπολογιστική πολυπλοκότητα των αλγόριθμων που λύνουν το πρόβλημα. Τα διάφορα κριτήρια αποδοτικότητας ενός αλγόριθμου (π.χ. χρόνος εκτέλεσης, αριθμός θέσεων μνήμης) ορίζουν αντίστοιχα μέτρα υπολογιστικής πολυπλοκότητας.

Το βασικότερο ίσως κριτήριο για την αποδοτικότητα ενός αλγόριθμου είναι ο υπολογιστικός χρόνος. Στην περίπτωση μιας μηχανής Turing M, η μονάδα μέτρησης του υπολογιστικού χρόνου είναι το στοιχειώδες υπολογιστικό βήμα της M, το οποίο συνίσταται σε μία εφαρμογή της συνάρτησης μετάβασης. Η χρονική πολυπλοκότητα ενός προβλήματος Π είναι μια συνάρτηση που φράσσει άνω τον αριθμό των στοιχειωδών βημάτων που χρειάζεται η μηχανή Turing για να αποφασίσει το Π.

Συγκεκριμένα, έστω μία DTM M που τερματίζει για κάθε είσοδο. Ο *χρόνος εκτέλεσης* (running time) ή η *χρονική πολυπλοκότητα* (time complexity) της M είναι μία αύξουσα συνάρτηση $t : \mathbb{N} \mapsto \mathbb{N}$ που ορίζεται ως εξής: Για κάθε $n \in \mathbb{N}$, το $t(n)$ είναι ο μέγιστος αριθμός στοιχειωδών βημάτων που χρειάζεται η M για να τερματίσει με είσοδο μια συμβολοσειρά x μήκους n. Αν η συνάρτηση $t(n)$ δίνει το χρόνο εκτέλεσης της M, λέμε ότι η M έχει χρόνο εκτέλεσης $t(n)$ ή είναι μία DTM $t(n)$-χρόνου.

Η χρονική πολυπλοκότητα ενός προβλήματος απόφασης Π (μιας γλώσσας $\mathcal{L}$) είναι η χρονική πολυπλοκότητα της πιο αποδοτικής (όσον αφορά στον χρόνο εκτέλεσης) DTM που αποφασίσει το Π (την $\mathcal{L}$). Αν υπάρχει μια DTM $t(n)$-χρόνου που αποφασίζει το Π, λέμε ότι το Π αποφασίζεται (ή λύνεται) σε χρόνο $\mathrm{O}(t(n))$.

Δεδομένης μιας αύξουσας[3] συνάρτησης $t : \mathbb{N} \mapsto \mathbb{N}$, ορίζουμε την κλάση πολυπλοκότητας περιορισμένου ντετερμινιστικού υπολογιστικού χρόνου (deterministic time complexity class) $\mathbf{DTIME}[t(n)]$ που περιλαμβάνει όλες τις γλώσσες / προβλήματα με χρονική πολυπλοκότητα $\mathrm{O}(t(n))$. Συγκεκριμένα, ορίζουμε

$$\mathbf{DTIME}[t(n)] \equiv \{\mathcal{L} : \mathcal{L} \text{ είναι γλώσσα αποφασίσιμη από μία DTM } \mathrm{O}(t(n))\text{-χρόνου}\}$$

6.4.1 Ιεραρχία Κλάσεων Χρονικής Πολυπλοκότητας

Μια σημαντική παρατήρηση σχετικά με τις κλάσεις $\mathbf{DTIME}[t(n)]$ είναι περιλαμβάνουν όλο και περισσότερα προβλήματα όσο αυξάνει η τάξη μεγέθους της συνάρτησης $t(n)$. Το ακόλουθο θεώρημα, που παρατίθεται χωρίς απόδειξη, είναι γνωστό σαν *Θεώρημα της Ιεραρχίας των Κλάσεων Χρονικής Πολυπλοκότητας* (Time Hierarchy Theorem).

Θεώρημα 6.2. *Για κάθε συνάρτηση υπολογιστικής πολυπλοκότητας $t(n)$,* $\mathbf{DTIME}[t(n)] \subset \mathbf{DTIME}[t(n)\log^2 t(n)]$.

[3] Στην πραγματικότητα δεν αρκεί η $t(n)$ να είναι αύξουσα. Πρέπει να ικανοποιεί και μερικές άλλες ιδιότητες, η συζήτηση των οποίων είναι έξω από το πνεύμα αυτών των σημειώσεων. Στο εξής θα αναφερόμαστε στις συναρτήσεις που ικανοποιούν όλες αυτές τις ιδιότητες σαν *συναρτήσεις υπολογιστικής πολυπλοκότητας*. Οι πολυωνυμικές, οι εκθετικές, και οι λογαριθμικές συναρτήσεις (καθώς και κάθε άθροισμα τους, γινόμενό τους, και σύνθεσή τους) είναι συναρτήσεις υπολογιστικής πολυπλοκότητας.

Ένα ενδιαφέρον πόρισμα του Θεωρήματος 6.2 είναι η ιεραρχία των κλάσεων $\mathbf{DTIME}[n^k]$, για $k = 1, 2, 3, \ldots$. Συγκεκριμένα, ισχύει ότι

$$\mathbf{DTIME}[n] \subset \mathbf{DTIME}[n^2] \subset \mathbf{DTIME}[n^3] \subset \cdots$$

Δύο σημαντικές κλάσεις ντετερμινιστικής χρονικής πολυπλοκότητας είναι οι $\mathbf{P}$ και $\mathbf{EXP}$. Η κλάση $\mathbf{P}$ αποτελείται από όλα τα προβλήματα με πολυωνυμική χρονική πολυπλοκότητα, ενώ η κλάση $\mathbf{EXP}$ από όλα τα προβλήματα με εκθετική χρονική πολυπλοκότητα. Τυπικά, είναι

$$\mathbf{P} \equiv \bigcup_{k \geq 0} \mathbf{DTIME}[n^k] \quad \text{και} \quad \mathbf{EXP} \equiv \bigcup_{k \geq 0} \mathbf{DTIME}[2^{n^k}]$$

Αφού κάθε πολυώνυμο είναι μικρότερο (ως προς την τάξη μεγέθους) από μια εκθετική συνάρτηση, τα προβλήματα που λύνονται σε πολυωνυμικό χρόνο είναι υποσύνολο των προβλημάτων που λύνονται σε εκθετικό χρόνο. Τυπικά, $\mathbf{P} \subseteq \mathbf{EXP}$. Μάλιστα, αποτελεί άμεση συνέπεια του Θεωρήματος 6.2 ότι η κλάση $\mathbf{P}$ είναι γνήσιο υποσύνολο της κλάσης $\mathbf{EXP}$. Δηλαδή υπάρχουν προβλήματα που λύνονται σε εκθετικό χρόνο αλλά δεν μπορούν να λυθούν σε πολυωνυμικό χρόνο.

Άσκηση 6.3. Να αποδείξετε ότι $\mathbf{P} \subset \mathbf{EXP}$.

Λύση. Άμεση συνέπεια του Θεωρήματος 6.2. □

Άσκηση 6.4. Με βάση τους αλγόριθμους που ήδη γνωρίζετε, να εντάξετε σε κλάσεις χρονικής πολυπλοκότητας τα προβλήματα του Πολλαπλασιασμού Ακολουθίας Πινάκων και του Σακιδίου (ακέραιη και κλασματική εκδοχή).

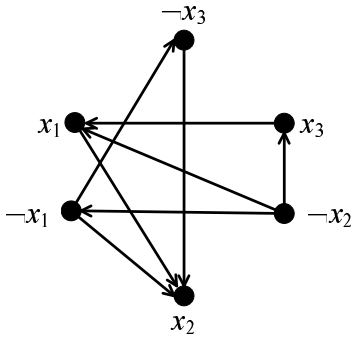
6.4.2 Η Κλάση P

Η κλάση $\mathbf{P}$ είναι μια από τις σημαντικότερες κλάσεις υπολογιστικής πολυπλοκότητας. Έχουμε ήδη αναφερθεί στο αξίωμα των Cook - Karp που ταυτίζει την κλάση $\mathbf{P}$ με την κλάση των προβλημάτων που μπορούν να λυθούν αποδοτικά από μια υπολογιστική συσκευή. Επιπλέον είναι κοινή επιστημονική πεποίθηση ότι η κλάση $\mathbf{P}$ παραμένει αμετάβλητη για οποιοδήποτε εύλογο ακολουθιακό υπολογιστικό μοντέλο.

Με εξαίρεση το Πρόβλημα του Περιοδεύοντος Πωλητή και το Ακέραιο Πρόβλημα του Σακιδίου, όλα τα προβλήματα στα οποία έχουμε αναφερθεί ανήκουν στην κλάση $\mathbf{P}$.

Ικανοποιησιμότητα Λογικών Προτάσεων

Μια λογική πρόταση (Boolean formula) αποτελείται από λογικές μεταβλητές που σχετίζονται μεταξύ τους με λογικούς τελεστές. Μια λογική πρόταση είναι σε *Συζευκτική Κανονική Μορφή* (ΣΚΜ, Conjuctive Normal Form - CNF) αν αποτελείται από συζεύξεις διαζεύξεων. Συγκεκριμένα, η πρόταση ϕ είναι σε ΣΚΜ αν αποτελείται από τη σύζευξη όρων (clauses), $\phi = c_1 \wedge c_2 \wedge \ldots \wedge c_m$, όπου κάθε όρος αποτελείται από τη διάζευξη ατόμων (literals), $c_i = \ell_{i_1} \vee \ell_{i_2} \vee \ldots \vee \ell_{i_k}$. Άτομο είναι μία λογική μεταβλητή x_i ή η άρνησή της $\neg x_i$. Για παράδειγμα, η λογική πρόταση $(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$ είναι σε ΣΚΜ. Κάθε λογική πρόταση μπορεί να γραφεί ισοδύναμα σε Συζευκτική Κανονική Μορφή.



Σχήμα 6.1: Το γράφημα $G_\phi(V, E)$ που αντιστοιχεί στην πρόταση $\phi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_2)$ [33, Σχήμα 9.1].

Μία λογική πρόταση είναι *ικανοποιήσιμη* (satisfiable) αν υπάρχει μία αποτίμηση των λογικών μεταβλητών που την επαληθεύει. Στο πρόβλημα της k-ικανοποιησιμότητας (k-SAT), δίνεται μια λογική πρόταση ϕ σε ΣΚΜ με k ή λιγότερα άτομα σε κάθε όρο. Το ζητούμενο είναι να αποφασισθεί αν η ϕ είναι ικανοποιήσιμη ή όχι.

Το πρόβλημα της k-ικανοποιησιμότητας είναι από τα κεντρικά προβλήματα στη Θεωρία Υπολογιστικής Πολυπλοκότητας επειδή η λειτουργία μιας μηχανής Turing M μπορεί να κωδικοποιηθεί σε μια λογική πρόταση ϕ ώστε η ϕ να είναι ικανοποιήσιμη αν και μόνο αν η M αποδέχεται την είσοδό της.

Θεώρημα 6.3. *Το πρόβλημα της 2-ικανοποιησιμότητας ανήκει στην κλάση* **P**.

Απόδειξη. Έστω μία λογική πρόταση ϕ σε ΣΚΜ με n λογικές μεταβλητές $x_1, \ldots, x_n$ και m όρους. Χωρίς βλάβη της γενικότητας, θεωρούμε ότι η ϕ περιέχει ακριβώς 2 άτομα σε κάθε όρο. Παρατηρούμε ότι ένας όρος της μορφής $x \vee y$ είναι ισοδύναμος με $(\neg x \to y) \wedge (\neg y \to x)$.

Για να αποφασίσουμε αν η ϕ είναι ικανοποιήσιμη, ορίζουμε ένα κατευθυνόμενο γράφημα $G_\phi(V, E)$ με μια κορυφή για κάθε άτομο της ϕ. Δηλαδή, $V = \{x_1, \ldots, x_n\} \cup \{\neg x_1, \ldots, \neg x_n\}$. Για κάθε όρο $\ell_i \vee \ell_j$ της ϕ, προσθέτουμε στο G_ϕ τις ακμές $(\neg\ell_i, \ell_j)$ και $(\neg\ell_j, \ell_i)$ θεωρώντας τις κορυφές x και $\neg\neg x$ σαν ταυτόσημες (βλ. Σχήμα 6.1). Από κατασκευή, το γράφημα G_ϕ είναι συμμετρικό όσον αφορά στις ακμές του. Δηλαδή, το G_ϕ περιέχει την ακμή (ℓ, ℓ') αν και μόνο αν περιέχει την ακμή $(\neg\ell', \neg\ell)$.

Θα δείξουμε ότι η πρόταση ϕ είναι ικανοποιήσιμη αν και μόνο αν για καμία μεταβλητή x δεν υπάρχει μονοπάτι από την κορυφή x στην κορυφή $\neg x$ και από την κορυφή $\neg x$ στην κορυφή x. Το θεώρημα έπεται επειδή (α) το γράφημα G_ϕ μπορεί να κατασκευαστεί σε χρόνο $O(n + m)$, και (β) μπορούμε να ελέγξουμε την προσπελασιμότητα κάθε κορυφής $\neg x$ από την κορυφή x σε συνολικό χρόνο $O(n(n + m))$ εκτελώντας n Αναζητήσεις Πρώτα σε Πλάτος. Αφού η περιγραφή της ϕ έχει μέγεθος $\Omega(n + m)$, ουσιαστικά διατυπώνουμε έναν αλγόριθμο πολυωνυμικού χρόνου για το πρόβλημα της 2-ικανοποιησιμότητας.

Έστω αρχικά ότι η ϕ είναι ικανοποιήσιμη και ότι υπάρχει μονοπάτι p από κάποια κορυφή x στην κορυφή $\neg x$ και από κάποια κορυφή $\neg x$ στην κορυφή x. Θεωρούμε μια αποτίμηση α που επαληθεύει την ϕ και έχει $\alpha(x) = 1$. Αφού $\alpha(x) = 1$ και $\alpha(\neg x) = 0$, το μονοπάτι p πρέπει να περιέχει ακμή (ℓ_1, ℓ_2) για την οποία $\alpha(\ell_1) = 1$ και $\alpha(\ell_2) = 0$. Όμως τότε ο αντίστοιχος όρος

$\neg\ell_1 \vee \ell_2$ δεν επαληθεύεται από την α. Αυτό αντιβαίνει στην υπόθεση ότι η ϕ είναι ικανοποιήσιμη. Με το ίδιο επιχείρημα χειριζόμαστε και την περίπτωση που $\alpha(x) = 0$.

Για το αντίστροφο, υποθέτουμε ότι δεν υπάρχει κορυφή $\neg x$ που να είναι προσπελάσιμη από την κορυφή x, και κατασκευάζουμε μια αποτίμηση α που ικανοποιεί τη ϕ. Ουσιαστικά πρέπει να αναθέσουμε λογικές τιμές στις κορυφές του γραφήματος G_ϕ ώστε να μην υπάρχει ακμή που κατευθύνεται από κορυφή με τιμή 1 σε κορυφή με τιμή 0.

Για την ανάθεση των λογικών τιμών, εργαζόμαστε ως εξής: ενόσω υπάρχουν κορυφές χωρίς λογική τιμή, επιλέγουμε μία τέτοια κορυφή ℓ και αναθέτουμε την τιμή 1 στην ℓ και σε κάθε άλλη κορυφή που είναι προσπελάσιμη από την ℓ. Επίσης, αναθέτουμε την τιμή 0 στις αρνήσεις των παραπάνω κορυφών (οι οποίες λόγω συμμετρίας αντιστοιχούν στους κόμβους από τους οποίους η $\neg\ell$ είναι προσπελάσιμη).

Σε αυτή τη διαδικασία δεν εμφανίζεται πρόβλημα συνέπειας στις τιμές των κορυφών. Κατ' αρχήν δεν υπάρχει ζευγάρι κορυφών x και $\neg x$ που να είναι αμφότερες προσπελάσιμες από την ℓ. Αν συνέβαινε κάτι τέτοιο, θα υπήρχαν μονοπάτια $\ell - x - \neg\ell$ και $\ell - \neg x - \neg\ell$ εξαιτίας της συμμετρίας του G_ϕ. Έχουμε όμως υποθέσει ότι τέτοιο μονοπάτι δεν υπάρχει. Επίσης, αν υπήρχε μονοπάτι από την ℓ σε μία κορυφή x που έχει ήδη τιμή 0 από προηγούμενο βήμα, η $\neg\ell$ θα ήταν προσπελάσιμη από τη $\neg x$ (λόγω συμμετρίας). Επομένως, η ℓ έπρεπε να είχε πάρει την τιμή 0 στο ίδιο βήμα με τη x.

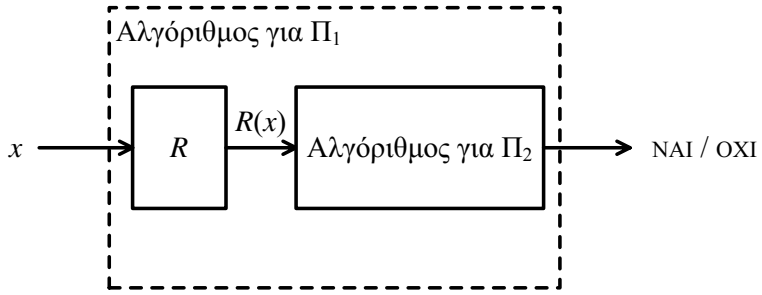
Η διαδικασία αυτή ολοκληρώνεται έχοντας αναθέσει λογικές τιμές σε όλες τις κορυφές του G_ϕ. Όπως εξηγήσαμε δεν εμφανίζονται προβλήματα συνέπειας. Από κατασκευής, δεν υπάρχει ακμή που να κατευθύνεται από κορυφή με τιμή 1 σε κορυφή με τιμή 0. Συνεπώς, η διαδικασία καταλήγει σε μια αποτίμηση των λογικών μεταβλητών που ικανοποιεί την ϕ. □

Άσκηση 6.5. Να αποδείξετε ότι η κλάση $\mathbf{P}$ είναι κλειστή ως προς την ένωση, την τομή, και το συμπλήρωμα. Δηλαδή πρέπει να δείξετε ότι αν δύο γλώσσες $\mathcal{L}_1$ και $\mathcal{L}_2$ ανήκουν $\mathbf{P}$, τότε και οι γλώσσες $\mathcal{L}_1 \cap \mathcal{L}_2$, $\mathcal{L}_1 \cup \mathcal{L}_2$, και $\overline{\mathcal{L}}_1$ ανήκουν στο $\mathbf{P}$.

Λύση. Έστω M_1 και M_2 οι DTM πολυωνυμικού χρόνου που αποφασίζουν τις $\mathcal{L}_1$ και $\mathcal{L}_2$ αντίστοιχα. Χωρίς βλάβη της γενικότητας, θεωρούμε ότι και οι δύο γλώσσες (και οι αντίστοιχες μηχανές Turing) ορίζονται στο ίδιο αλφάβητο Σ. Μια συμβολοσειρά $x \in \Sigma^*$ ανήκει στη γλώσσα $\mathcal{L}_1 \cap \mathcal{L}_2$ αν και μόνο αν $M_1(x) = \text{NAI}$ και $M_2(x) = \text{NAI}$, ανήκει στη γλώσσα $\mathcal{L}_1 \cup \mathcal{L}_2$ αν και μόνο αν είτε $M_1(x) = \text{NAI}$ είτε $M_2(x) = \text{NAI}$, και ανήκει στη γλώσσα $\overline{\mathcal{L}}_1$ αν και μόνο αν $M_1(x) = \text{OXI}$. Η προσομοίωση της λειτουργίας των $M_1(x)$ και $M_2(x)$ μπορεί να γίνει σε πολυωνυμικό χρόνο. Από τα αποτελέσματα των προσομοιώσεων μπορούμε εύκολα να συνάγουμε αν η συμβολοσειρά εισόδου ανήκει ή όχι στην γλώσσα. Επομένως, οι παραπάνω γλώσσες ανήκουν στο $\mathbf{P}$. □

6.5 Αναγωγή και Πληρότητα

Οι έννοιες της *αναγωγής* (reduction) και της *πληρότητας* (completeness) σχετίζονται με το γεγονός ότι συγκεκριμένα προβλήματα συνοψίζουν τη δυσκολία μιας ολόκληρης κλάσης πολυπλοκότητας. Διαισθητικά, ένα πρόβλημα είναι *δύσκολο* (hard) για μια κλάση πολυπλοκότητας αν είναι τουλάχιστον τόσο δύσκολο να λυθεί όσο κάθε άλλο πρόβλημα της κλάσης. Ένα πρόβλημα είναι *πλήρες*



Σχήμα 6.2: Αναγωγή του προβλήματος Π_1 στο πρόβλημα Π_2 [33, Σχήμα 8.1].

(complete) για μία κλάση πολυπλοκότητας αν είναι δύσκολο και ταυτόχρονα ανήκει στην κλάση. Με απλά λόγια, τα πλήρη προβλήματα συνοψίζουν τη δυσκολία μιας κλάσης πολυπλοκότητας.

Οι παραπάνω διαισθητικοί ορισμοί της δυσκολίας και της πληρότητας βασίζονται στη σύγκριση της υπολογιστικής δυσκολίας δύο προβλημάτων. Η έννοια της αναγωγής επιτρέπει αυτή τη σύγκριση. Ένα πρόβλημα Π_1 *ανάγεται* σε ένα πρόβλημα Π_2 αν υπάρχει ένας μετασχηματισμός R που για κάθε συμβολοσειρά x παράγει μια συμβολοσειρά $R(x)$ τέτοια ώστε $x \in \Pi_1$ αν και μόνο αν $R(x) \in \Pi_2$. Ο μετασχηματισμός R ονομάζεται *αναγωγή* του Π_1 στο Π_2. Δηλαδή η αναγωγή R απεικονίζει τα ΝΑΙ-στιγμιότυπα του Π_1 σε ΝΑΙ-στιγμιότυπα του Π_2 και τα ΟΧΙ-στιγμιότυπα του Π_1 σε ΟΧΙ-στιγμιότυπα του Π_2. Η αναγωγή R από το Π_1 στο Π_2 επιτρέπει τη χρήση ενός αλγόριθμου / μηχανής Turing που αποφασίζει το Π_2 για να αποφασίσουμε το Π_1 (βλ. Σχήμα 6.2).

Η αναγωγή ενός προβλήματος Π_1 σε ένα πρόβλημα Π_2 δεν αρκεί για να στοιχειοθετήσει ότι το Π_2 είναι τουλάχιστον τόσο δύσκολο όσο το Π_1. Πρέπει ακόμη η διαδικασία της αναγωγής (ο μετασχηματισμός R) να μπορεί να υπολογίζεται αποδοτικά.

Ορισμός 6.4 (Πολυωνυμική Αναγωγή). Ένα πρόβλημα Π_1 *ανάγεται πολυωνυμικά* σε ένα πρόβλημα Π_2 αν υπάρχει μια συνάρτηση R που υπολογίζεται σε πολυωνυμικό χρόνο και για κάθε συμβολοσειρά x, $x \in \Pi_1$ αν και μόνο αν $R(x) \in \Pi_2$. Η συνάρτηση / μετασχηματισμός R αποτελεί μια *πολυωνυμική αναγωγή* του Π_1 στο Π_2.

Η αναγωγή του Ορισμού 6.4 ονομάζεται και αναγωγή του Karp. Στη συνέχεια όταν αναφερόμαστε σε αναγωγή, θα εννούμε την πολυωνυμική αναγωγή.

Άσκηση 6.6. Να αποδείξετε ότι ότι η πολυωνυμική αναγωγή είναι μεταβατική (δηλαδή αν το Π_1 ανάγεται στο Π_2, και το Π_2 ανάγεται στο Π_3, τότε και το Π_1 ανάγεται στο Π_3).

Λύση. Η σύνθεση δύο DTM πολυωνυμικού χρόνου είναι μία DTM πολυωνυμικού χρόνου. Επομένως, η σύνθεση δύο πολυωνυμικών αναγωγών αποτελεί μια πολυωνυμική αναγωγή. □

Άσκηση 6.7. Να αποδείξετε ότι η κλάση $\mathbf{P}$ είναι κλειστή ως προς την πολυωνυμική αναγωγή (δηλαδή αν το Π_1 ανάγεται στο Π_2 και $\Pi_2 \in \mathbf{P}$, τότε και $\Pi_1 \in \mathbf{P}$).

Λύση. Αν η αναγωγή R από το Π_1 στο Π_2 υπολογίζεται από μια DTM πολυωνυμικού χρόνου και το Π_2 αποφασίζεται επίσης από μια DTM πολυωνυμικού χρόνου, η σύνθεση των δύο μηχανών

είναι μία DTM πολυωνυμικού χρόνου (βλ. Σχήμα 6.2). Το αποτέλεσμα της σύνθεσης είναι λοιπόν μία DTM πολυωνυμικού χρόνου που αποφασίζει το Π_1. □

Άσκηση 6.8. Δεδομένου ενός μη-κατευθυνόμενου γραφήματος $G(V, E)$, θέλουμε να αποφασίσουμε αν το G είναι διμερές, δηλαδή αν το σύνολο κορυφών μπορεί να διαμεριστεί σε σύνολα ανεξαρτησίας[4] V_1 και V_2. Γνωρίζουμε ότι το πρόβλημα αυτό ανήκει στο **P**, αφού μπορεί να λυθεί με Αναζήτηση Πρώτα σε Πλάτος (βλ. Άσκηση 5.9). Σε αυτή την άσκηση, ζητείται να περιγράψετε μια πολυωνυμική αναγωγή του προβλήματος αναγνώρισης ενός διμερούς γραφήματος στο πρόβλημα της 2-ικανοποιησιμότητας.

Λύση. Έστω μη-κατευθυνόμενο γράφημα $G(V, E)$. Θα περιγράψουμε έναν μετασχηματισμό / αλγόριθμο πολυωνυμικού χρόνου που υπολογίζει μία λογική πρόταση ϕ_G σε ΣΚΜ με 2 άτομα ανά όρο και θα δείξουμε ότι το G είναι διμερές αν και μόνο αν η ϕ_G είναι ικανοποιήσιμη.

Σε κάθε κορυφή $i \in V$ του αντιστοιχούμε μία λογική μεταβλητή x_i. Για κάθε ακμή $\{i, j\} \in E$, προσθέτουμε τους όρους $(x_i \vee \neg x_j) \wedge (\neg x_i \vee x_j) \equiv x_i \oplus x_j$. Η ιδέα είναι ότι αν δύο μεταβλητές συνδέονται με ακμή, οι λογικές τιμές τους πρέπει να είναι διαφορετικές (ισοδύναμα, οι αντίστοιχες κορυφές δεν μπορεί να ανήκουν στο ίδιο σύνολο ανεξαρτησίας). Αυτό περιγράφει την κατασκευή της ϕ_G που μπορεί να γίνει σε πολυωνυμικό χρόνο.

Έστω ότι το $G(V, E)$ είναι διμερές, και V_1, V_2 τα δύο σύνολα ανεξαρτησίας. Αναθέτοντας τη λογική τιμή 1 στις κορυφές του V_1 και τη λογική τιμή 0 στις κορυφές του V_2 ικανοποιούμε όλους τους όρους της ϕ_G, άρα και τη ϕ_G συνολικά.

Αντίστροφα, αν η ϕ_G είναι ικανοποιήσιμη, θεωρούμε μια αποτίμηση που ικανοποιεί τη ϕ_G και βάζουμε στο V_1 τις κορυφές που έχουν λογική τιμή 1 και στο V_2 τις κορυφές που έχουν λογική τιμή 0. Τα V_1 και V_2 είναι σύνολα ανεξαρτησίας αφού δεν μπορεί να υπάρχει ακμή μεταξύ δύο λογικών μεταβλητών με την ίδια τιμή (οι αντίστοιχοι όροι δεν θα ικανοποιούνταν). □

Αν το ζητούμενο είναι η υπολογισιμότητα σε πολυωνυμικό χρόνο, η πολυωνυμική αναγωγή επιτρέπει τη σύγκριση της υπολογιστικής δυσκολίας δύο προβλημάτων. Αν το Π_1 ανάγεται πολυωνυμικά στο Π_2 και το Π_2 υπολογίζεται σε πολυωνυμικό χρόνο, τότε και το Π_1 υπολογίζεται σε πολυωνυμικό χρόνο (βλ. Σχήμα 6.2 και Ασκήσεις 6.7 και 6.8). Από την άλλη πλευρά, αν το Π_1 δεν υπολογίζεται σε πολυωνυμικό χρόνο, τότε ούτε το Π_2 υπολογίζεται σε πολυωνυμικό χρόνο. Μπορούμε λοιπόν να πούμε ότι το Π_2 είναι τουλάχιστον τόσο δύσκολο να λυθεί όσο το Π_1 (σχηματικά $\Pi_1 \leq \Pi_2$). Πρέπει όμως να είναι σαφές ότι η σύγκριση αυτή αφορά μόνο την υπολογισιμότητα σε *πολυωνυμικό χρόνο*.

Ιδιαίτερο ενδιαφέρον παρουσιάζουν τα προβλήματα μιας κλάσης που είναι τουλάχιστον τόσο δύσκολο να λυθούν όσο κάθε άλλο πρόβλημα στην κλάση.

Ορισμός 6.5. Έστω **C** μια κλάση πολυπλοκότητας. Ένα πρόβλημα Π ονομάζεται *δύσκολο για την κλάση* **C** (ή **C**-δύσκολο, **C**-hard) αν κάθε πρόβλημα $\Pi' \in \mathbf{C}$ ανάγεται στο Π. Αν επιπλέον $\Pi \in \mathbf{C}$, το Π ονομάζεται *πλήρες για την κλάση* **C** (ή **C**-πλήρες, **C**-complete).

[4] Ένα σύνολο κορυφών $X \subseteq V$ ενός γραφήματος $G(V, E)$ ονομάζεται σύνολο ανεξαρτησίας (independent set) αν οι κορυφές του X δεν έχουν ακμές μεταξύ τους.

Δεν είναι προφανές ότι υπάρχουν πλήρη προβλήματα για κάθε κλάση. Παρόλα αυτά, μπορεί να αποδειχθεί ότι αρκετά γνωστά προβλήματα είναι πλήρη για την κλάση $\mathbf{P}$. Στο επόμενο κεφάλαιο, θα αποδείξουμε ότι μερικά οικεία προβλήματα είναι πλήρη για μια άλλη πολύ σημαντική κλάση, το $\mathbf{NP}$.

Τα πλήρη προβλήματα αποτελούν μια κεντρική έννοια και ένα πολύ σημαντικό εργαλείο για τη Θεωρία Πολυπλοκότητας. Η πολυπλοκότητα ενός υπολογιστικού προβλήματος θεωρείται ότι έχει καθοριστεί όταν αυτό αποδειχθεί πλήρες για κάποια κλάση. Τα πλήρη προβλήματα μια κλάσης $\mathbf{C}$ συνοψίζουν την υπολογιστική δυσκολία όλων των προβλημάτων της $\mathbf{C}$ και αποτελούν το σύνδεσμο της Θεωρίας Πολυπλοκότητας με τη Θεωρία Αλγορίθμων και τις εφαρμογές.

Η ύπαρξη σημαντικών πρακτικών προβλημάτων που είναι πλήρη για κάποια κλάση προσδίδει στην κλάση πρακτική αξία, που συνήθως δεν είναι ξεκάθαρη από τον ορισμό της (βλ. σημασία του $\mathbf{P}$ και του $\mathbf{NP}$). Αντίστροφα, αν μια κλάση δεν έχει σημαντικά ή έστω γνωστά πρακτικά πλήρη προβλήματα, αυτόματα η κλάση θεωρείται θεωρητικό κατασκεύασμα χωρίς ουσιαστικό περιεχόμενο.

Η έννοια της πληρότητας χρησιμοποιείται για την απόδειξη αρνητικών αποτελεσμάτων της εξής μορφής: Αν ένα πρόβλημα Π είναι πλήρες για μία κλάση $\mathbf{C}$, τότε θεωρείται απίθανο το Π να ανήκει σε κάποια πιο μικρότερη κλάση $\mathbf{C}' \subset \mathbf{C}$. Ο λόγος είναι ότι αν η $\mathbf{C}'$ είναι κλειστή ως προς την αναγωγή και το Π ανήκει στη $\mathbf{C}'$, τότε η $\mathbf{C}$ πρέπει να ταυτίζεται με το υποσύνολό της $\mathbf{C}'$, δηλαδή η $\mathbf{C}$ καταπίπτει (collapse) στην $\mathbf{C}'$.

Άσκηση 6.9. Να αποδείξετε ότι αν δύο κλάσεις $\mathbf{C}$ και $\mathbf{C}'$ είναι κλειστές ως προς την αναγωγή και υπάρχει ένα πρόβλημα Π που είναι πλήρες τόσο για τη $\mathbf{C}$ όσο και για τη $\mathbf{C}'$, τότε $\mathbf{C} = \mathbf{C}'$.

Λύση. Θεωρούμε οποιοδήποτε πρόβλημα $\Pi' \in \mathbf{C}'$. Αφού το Π είναι πλήρες για τη $\mathbf{C}'$, το Π' ανάγεται στο Π. Αφού όμως $\Pi \in \mathbf{C}$ και η $\mathbf{C}$ είναι κλειστή ως προς την αναγωγή, ισχύει ότι $\Pi' \in \mathbf{C}$. Επομένως, κάθε πρόβλημα της $\mathbf{C}'$ ανήκει και στη $\mathbf{C}$. Τυπικά, $\mathbf{C}' \subseteq \mathbf{C}$. Λόγω συμμετρίας, ισχύει επίσης ότι $\mathbf{C} \subseteq \mathbf{C}'$. Συνεπώς, $\mathbf{C} = \mathbf{C}'$. □

6.6 Βιβλιογραφικές Αναφορές

Για μία ολοκληρωμένη και σε βάθος παρουσίαση της Θεωρίας Υπολογιστικής Πολυπλοκότητας, προτείνεται το [33]. Η παρουσίαση των βασικών εννοιών της υπολογιστικής πολυπλοκότητας σε αυτό και στο επόμενο κεφάλαιο έχουν επηρεαστεί από το [33]. Επίσης πολλά παραδείγματα και ασκήσεις είναι προέρχονται από το ίδιο βιβλίο.

Όσον αφορά στις έννοιες των τυπικών γλωσσών, των μηχανών Turing, βασικών εναλλακτικών μοντέλων υπολογισμού, και θεμελιωδών κλάσεων υπολογιστικής πολυπλοκότητας, προτείνονται τα [23, 29, 38]. Το [23] αποτελεί κλασσικό βιβλίο σε θέματα αυτομάτων, τυπικών γλωσσών, μηχανών Turing, και μη-υπολογισιμότητας. Το [29] επιχειρεί μια εξαιρετική και σε βάθος παρουσίαση όλων των θεμάτων που άπτονται της Θεωρίας Υπολογισμού με ιδιαίτερη έμφαση στα διάφορα είδη αυτομάτων και μηχανών Turing καθώς και σε θέματα μη-υπολογισιμότητας. Το [29] έχει εξαιρετική συλλογή ασκήσεων και κυκλοφορεί μεταφρασμένο στα Ελληνικά. Το [38] αποτελεί ένα σύγχρονο βιβλίο που διαπραγματεύεται τόσο θέματα Θεωρίας Υπολογισμού όσο και θέματα υπολογιστικής πολυπλοκότητας.

7 Μη-Ντετερμινισμός και NP-Πληρότητα

Μέχρι αυτό το σημείο, έχουμε θεωρήσει τα ρεαλιστικά υπολογιστικά μοντέλα της Μηχανής Άμεσης Προσπέλασης Μνήμης (RAM) και της Ντετερμινιστικής Μηχανής Turing (DTM). Σε αυτό το κεφάλαιο, θα θεωρήσουμε το μη-ρεαλιστικό υπολογιστικό μοντέλο των Μη-Ντετερμινιστικών Μηχανών Turing (Non-Deterministic Turing Machines - NDTM).

Αντί του μονοσήμαντα ορισμένου (*ντετερμινιστικού*) τρόπου με τον οποίο εξελίσσεται ο υπολογισμός μίας DTM για δεδομένη είσοδο, ο υπολογισμός μιας NDTM εξελίσσεται με πολλές διαφορετικές (*όχι όμως και ανεξάρτητες*) εκδοχές. Η NDTM αποδέχεται μία είσοδο αν μια τουλάχιστον από αυτές τις υπολογιστικές εκδοχές καταλήξει σε αποδοχή.

Σε διαισθητικό επίπεδο, μπορούμε να θεωρήσουμε ότι μια NDTM αποτελείται από έναν μεγάλο αριθμό διαφορετικών (*αλλά συσχετιζόμενων*) DTM. Για κάθε είσοδο, η NDTM διαλέγει και επικαλείται την DTM με το "πλέον επιθυμητό" αποτέλεσμα. Ισοδύναμα, η NDTM προσομοιώνει όλες τις διαφορετικές DTM και επιστρέφει το "καλύτερο αποτέλεσμα". Ο χρόνος της προσομοίωσης καθορίζεται από τη χρονική πολυπλοκότητα της πιο αργής DTM (και όχι από το άθροισμα του χρόνου για την προσομοίωση όλων των DTM).

Θα δούμε ότι κάθε NDTM μπορεί να προσομοιωθεί από μία DTM με εκθετική επιβάρυνση στο χρόνο εκτέλεσης. Αν αυτή η προσομοίωση μπορεί να γίνει χωρίς σημαντική χρονική επιβάρυνση[1] ή όχι αποτελεί το πιο σημαντικό ανοικτό ερώτημα στη Θεωρητική Επιστήμη των Υπολογιστών. Μια από τις πιο γνωστές διατυπώσεις αυτού του ερωτήματος αφορά στην διαφορετικότητα των κλάσεων πολυπλοκότητας **P** και **NP**.

Από την παραπάνω αδρή περιγραφή γίνεται φανερό γιατί το μοντέλο των Μη-Ντετερμινιστικών Μηχανών Turing αποτελεί ένα εξαιρετικά ισχυρό αλλά ελάχιστα ρεαλιστικό υπολογιστικό μοντέλο. Άλλωστε ο μη-ντετερμινισμός δεν έχει στόχο να αποτελέσει ένα μαθηματικό μοντέλο για μια συγκεκριμένη, πρακτική ή ιδεατή, μορφή υπολογισμού. Ο λόγος που ο μη-ντετερμινισμός αποτελεί αντικείμενο διεξοδικής μελέτης στη Θεωρία Υπολογιστικής Πολυπλοκότητας είναι οι εφαρμογές του σε άλλες περιοχές όπως η Μαθηματική Λογική, η Τεχνητή Νοημοσύνη, και η Συνδυαστική Βελτιστοποίηση. Η κλάση **NP**, με τα πολλά και σημαντικά πλήρη και δύσκολα προβλήματα, αποτελεί μια από τις βασικές εφαρμογές του μη-ντετερμινισμού στη περιοχή της Συνδυαστικής Βελτιστοποίησης.

[1] Αν μια NDTM μπορούσε να προσομοιωθεί από μία DTM χωρίς σημαντική επιβάρυνση στο χρόνο εκτέλεσης, το υπολογιστικό μοντέλο των Μη-Ντετερμινιστικών Μηχανών Turing θα καθίστατο αυτόματα ρεαλιστικό.

7.1 Μη-Ντετερμινιστικές Μηχανές Turing

Μια Μη-Ντετερμινιστική Μηχανή Turing (Non-Deterministic Turing Machine - NDTM) N με $k \geq 1$ ταινίες είναι μια διατεταγμένη πεντάδα $N = (Q, \Sigma, \Delta, q_0, F)$. Όπως και στον Ορισμό 6.3, το Q είναι το σύνολο καταστάσεων, το $q_0 \in Q$ είναι η αρχική κατάσταση, το $F \subseteq Q$ είναι το σύνολο των τελικών καταστάσεων, και το Σ είναι το αλφάβητο εισόδου. Η ουσιαστική διαφορά είναι ότι ο υπολογισμός της N καθορίζεται από τη *σχέση μετάβασης* Δ που ορίζεται ως

$$\Delta \subseteq ((Q \setminus F) \times \Gamma^k) \times (Q \times \Gamma^k \times \{L, R, S\}^k)$$

Για κάθε διαμόρφωση / στοιχείο του $(Q \setminus F) \times \Gamma^k$, η σχέση μετάβασης Δ ορίζει τις επιτρεπτές επόμενες διαμορφώσεις (που μπορεί να είναι καμία, μία, ή και περισσότερες). Με άλλα λόγια, το επόμενο βήμα του υπολογισμού μιας NDTM δεν προσδιορίζεται μονοσήμαντα από την τρέχουσα διαμόρφωση, αλλά η NDTM επιλέγει το επόμενο βήμα της από το σύνολο των επιτρεπτών διαμορφώσεων που καθορίζει η σχέση μετάβασης Δ.

Εξ΄ ορισμού, οι DTM αποτελούν μια ειδική κατηγορία NDTM, αυτή όπου η σχέση μετάβασης Δ έχει την επιπλέον ιδιότητα να είναι *συνάρτηση*.

7.1.1 Γλώσσες Ημιαποφασίσιμες και Αποφασίσιμες από NDTM

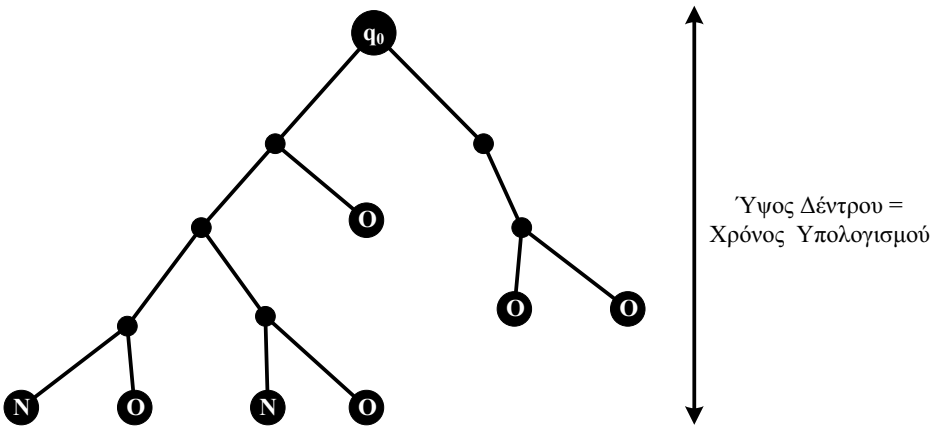
Το μη-ντετερμινιστικό υπολογιστικό μοντέλο είναι πολύ ισχυρό και συνάμα ελάχιστα ρεαλιστικό επειδή δεν υπάρχει απόλυτη εξάρτηση ανάμεσα στην είσοδο και στην εξέλιξης του υπολογισμού και τελικά στην έξοδο μιας NDTM. Στα ντετερμινιστικά υπολογιστικά μοντέλα, κάθε είσοδος καθορίζει μονοσήμαντα την εξέλιξη του υπολογισμού και τελικά την έξοδο / απάντηση της μηχανής. Αντίθετα, η είσοδος μιας NDTM επιτρέπει την επιλογή ανάμεσα σε πολλές διαφορετικές *υπολογιστικές εκδοχές* (computation paths) και εξόδους / απαντήσεις. Από όλες τις διαφορετικές απαντήσεις, η NDTM επιλέγει κάθε φορά την ''πλέον επιθυμητή''.

Μια NDTM N *ημιαποφασίζει* μία συμβολοσειρά εισόδου $x \in \Sigma^*$ αν μία τουλάχιστον από τις υπολογιστικές εκδοχές της $N(x)$ τερματίζει. Η N *απορρίπτει* τη x μόνο αν *καμία* υπολογιστική εκδοχή της $N(x)$ δεν τερματίζει.

Θεωρούμε τον υπολογισμό μιας NDTM N με συμβολοσειρά εισόδου $x \in \Sigma^*$. Μια εκδοχή του υπολογισμού της $N(x)$ καλείται *υπολογισμός αποδοχής* αν καταλήγει στην τελική κατάσταση NAI, και καλείται *υπολογισμός απόρριψης* αν καταλήγει στην τελική κατάσταση OXI. Θα λέμε ότι η *αποδέχεται* το x αν *μία τουλάχιστον* εκδοχή του υπολογισμού της $N(x)$ είναι υπολογισμός αποδοχής. Θα λέμε ότι η *απορρίπτει* το x αν *καμία* εκδοχή του υπολογισμού της $N(x)$ δεν είναι υπολογισμός αποδοχής. Σε περίπτωση αποδοχής, γράφουμε $N(x) =$ NAI, και σε περίπτωση απόρριψης, γράφουμε $N(x) =$ OXI.

Μία NDTM N *αποφασίζει* μια γλώσσα $\mathcal{L} \subseteq \Sigma^*$ αν για κάθε συμβολοσειρά εισόδου x, (α) όλες οι εκδοχές του υπολογισμού της $N(x)$ τερματίζουν, και (β) $x \in \mathcal{L}$ αν και μόνο αν $N(x) =$ NAI. Μια γλώσσα είναι NDTM-αποφασίσιμη (decidable) αν υπάρχει μια NDTM που την αποφασίζει.

Στα μη-ντετερμινιστικά υπολογιστικά μοντέλα, οι υπολογισμοί αποδοχής επιβάλουν τη θέλησή τους ακόμα και αν αποτελούν ισχνή μειοψηφία. Αυτή η ασυμμετρία υπέρ του NAI είναι πολύ συνηθισμένη στα μαθηματικά. Μια μαθηματική πρόταση είναι θεώρημα (θεωρείται δηλαδή αποδεδειγμένη) αν έχουμε βρει μια σωστή απόδειξη για αυτή. Αντίστροφα, μια πρόταση δεν ισχύει αν έχουμε βρει ένα αντι-παράδειγμα.



Σχήμα 7.1: Παράδειγμα δέντρου υπολογισμού μιας NDTM. Η χρονική πολυπλοκότητα δίνεται από το ύψος του δέντρου.

Ένας τρόπος αναπαράστασης του υπολογισμού της $N(x)$ είναι μέσω του *δέντρου υπολογισμού* (computation tree). Η ρίζα του δέντρου είναι η αρχική διαμόρφωση της $N(x)$ με κατάσταση q_0 και είσοδο x. Οι ενδιάμεσες διαμορφώσεις στις διαφορετικές υπολογιστικές εκδοχές αναπαρίστανται από τις κορυφές του δέντρου. Τα παιδιά κάθε κορυφής / ενδιάμεσης διαμόρφωσης είναι οι διαμορφώσεις που (σύμφωνα με τη σχέση μετάβασης Δ) μπορούν να προκύψουν σε ένα βήμα. Έτσι κάθε διαφορετική υπολογιστική εκδοχή της $N(x)$ αντιστοιχεί σε έναν κλάδο του δέντρου υπολογισμού (Σχήμα 7.1). Αν όλες οι υπολογιστικές εκδοχές / κλάδοι του δέντρου υπολογισμού τερματίζουν, το δέντρο είναι πεπερασμένο. Η N αποδέχεται την είσοδο x αν τουλάχιστον ένα από τα φύλλα του δέντρου βρίσκεται στην κατάσταση NAI και απορρίπτει το x αν κανένα από τα φύλλα δεν βρίσκεται σε κατάσταση NAI. Εφόσον ένας τουλάχιστον κλάδος του δέντρου αντιστοιχεί σε υπολογισμό αποδοχής, θεωρούμε ότι η N θα επιλέξει μη-ντετερμινιστικά (θα "μαντέψει") τις κατάλληλες μεταβάσεις ώστε να αποδεχθεί το x.

Η χρονική πολυπλοκότητα μιας NDTM N με είσοδο x καθορίζεται από το μήκος της μεγαλύτερης υπολογιστικής εκδοχής, ή ισοδύναμα από το ύψος του δέντρου υπολογισμού της $N(x)$. Συγκεκριμένα, έστω μια συνάρτηση υπολογιστικής πολυπλοκότητας $t : \mathbb{N} \mapsto \mathbb{N}$. Ο *χρόνος εκτέλεσης* ή η *υπολογιστική πολυπλοκότητα* μιας NDTM N είναι $t(n)$ αν για κάθε συμβολοσειρά x μήκους n, το μήκος *όλων* των κλάδων υπολογισμού της $N(x)$ είναι μικρότερο ή ίσο του $t(n)$. Λέμε ότι μια γλώσσα αποφασίζεται σε μη-ντετερμινιστικό χρόνο $t(n)$ αν υπάρχει NDTM με χρόνο εκτέλεσης $t(n)$ που την αποφασίζει. Με βάση αυτό τον ορισμό, η χρονική πολυπλοκότητα μιας NDTM μπορεί να είναι εκθετικά μικρότερη από την συνολική (ντετερμινιστική) υπολογιστική δραστηριότητα.

7.1.2 Η Κλάση NTIME

Το σύνολο των γλωσσών που αποφασίζονται σε μη-ντετερμινιστικό χρόνο $\mathrm{O}(t(n))$ αποτελούν την κλάση υπολογιστικής πολυπλοκότητας $\mathbf{NTIME}[t(n)]$. Η οικογένεια κλάσεων πολυπλοκότητας περιορισμένου μη-ντετερμινιστικού χρόνου (non-deterministic time complexity class), $\mathbf{NTIME}[\cdot]$ ορίζεται τυπικά ως:

$$\mathbf{NTIME}[t(n)] \equiv \{\mathcal{L} : \mathcal{L} \text{ είναι γλώσσα αποφασίσιμη από μία NDTM } \mathrm{O}(t(n))\text{-χρόνου}\}$$

Αφού οι DTM αποτελούν ειδική κατηγορία των NDTM, η κλάση $\mathbf{DTIME}[t(n)]$ είναι υποσύνολο της κλάσης $\mathbf{NTIME}[t(n)]$ για κάθε συνάρτηση $t(n)$. Για μερικές συναρτήσεις πολυπλοκότητας, έχει αποδειχθεί ότι η κλάση **DTIME** είναι *γνήσιο υποσύνολο* της αντίστοιχης κλάσης **NTIME** (π.χ. $\mathbf{DTIME}[n] \subset \mathbf{NTIME}[n]$ [34]). Θα ήταν εξαιρετικά σημαντικό (και κατά κοινή πεποίθηση *απολύτως απίθανο*) να αποδειχθεί ότι για κάθε συνάρτηση πολυπλοκότητας $t(n)$, η κλάση $\mathbf{NTIME}[t(n)]$ είναι υποσύνολο της κλάσης $\mathbf{DTIME}[t'(n)]$, όπου $t'(n)$ είναι ένα πολυώνυμο της $t(n)$. Προς το παρόν, ας αποδείξουμε είναι ότι αυτό ισχύει όταν $t'(n) = \mathrm{O}(c^{t(n)})$, για κάποια σταθερά $c > 1$.

Θεώρημα 7.1. *Κάθε γλώσσα που αποφασίζεται σε μη-ντετερμινιστικό χρόνο $t(n)$ μπορεί να αποφασιστεί σε ντετερμινιστικό χρόνο $\mathrm{O}(c^{t(n)})$ για κάποια σταθερά $c > 1$.*

Απόδειξη. Έστω $\mathcal{L}$ μια τέτοια γλώσσα, και έστω N μια NDTM με χρόνο εκτέλεσης $t(n)$ που αποφασίζει την $\mathcal{L}$. Έστω d ο μέγιστος αριθμός επιτρεπτών διαμορφώσεων που μπορεί να προκύψει από μια διαμόρφωση της N. Το d ονομάζεται "βαθμός μη-ντετερμινισμού" της N. Υποθέτουμε ότι $d > 1$, γιατί διαφορετικά η N θα ήταν μια DTM. Θα δείξουμε πως μια DTM M μπορεί να προσομοιώσει τη λειτουργία της N στο συγκεκριμένο χρόνο.

Η ντετερμινιστική προσομοίωση της N βασίζεται στο γεγονός ότι μια ακολουθία από μη-ντετερμινιστικές επιλογές της N μπορεί να αναπαρασταθεί από έναν d-αδικό αριθμό ίδιου μήκους. Για κάθε πιθανό ύψος t του δέντρου υπολογισμού, η M προσομοιώνει τη λειτουργία της N για όλες τις διαφορετικές ακολουθίες μη-ντετερμινιστικών επιλογών. Ας σημειωθεί ότι η M πρέπει να λειτουργεί χωρίς εκ των προτέρων γνώση του χρονικού ορίου $t(n)$. Αυτός είναι ο λόγος που η M δοκιμάζει όλα τα δυνατά χρονικά όρια σε αύξουσα σειρά.

Για την εκτέλεση της προσομοίωσης, η M διατηρεί έναν d-αδικό αριθμό $(c_1, \ldots, c_t)$ στη δεύτερη ταινία της και προσομοιώνει τη λειτουργία της N κάνοντας στο βήμα i την μη-ντετερμινιστική επιλογή c_i. Αν η αντίστοιχη υπολογιστική εκδοχή καταλήξει σε αποδοχή, η M αποδέχεται την είσοδο και τερματίζει. Διαφορετικά, η M αυξάνει τον αριθμό $(c_1, \ldots, c_t)$ κατά 1 και συνεχίζει την προσομοίωση. Όταν η M έχει θεωρήσει όλους τους d-αδικούς αριθμούς μήκους t, αυξάνει το t κατά 1 και συνεχίζει. Η M απορρίπτει την είσοδο μόνο αν για κάποιο t, όλες οι υπολογιστικές εκδοχές μήκους το πολύ t καταλήγουν στις τελικές καταστάσεις ΟΧΙ ή ΤΕΛΟΣ.

Αφού η χρονική πολυπλοκότητα της N είναι $t(n)$, η M θα έχει ολοκληρώσει την προσομοίωση όταν θα έχει θεωρήσει όλους τους d-αδικούς αριθμούς μήκους $1, 2, \ldots, t(n)$. Η ντετερμινιστική προσομοίωση για κάθε d-αδικό αριθμό μήκους t μπορεί να γίνει σε χρόνο $\mathrm{O}(d^t)$. Επομένως, η χρονική πολυπλοκότητα της M είναι $\sum_{t=1}^{t(n)} \mathrm{O}(d^t) = \mathrm{O}(d^{t(n)+1})$. □

Μια ισοδύναμη διατύπωση του Θεωρήματος 7.1 είναι ότι για κάθε συνάρτηση υπολογιστικής πολυπλοκότητας $t(n)$,

$$\mathbf{NTIME}[t(n)] \subseteq \bigcup_{c>1} \mathbf{DTIME}[c^{t(n)}]$$

Μια σημαντική συνέπεια του Θεωρήματος 7.1 είναι ότι το σύνολο των γλωσσών που αποφασίζονται από Μη-Ντετερμινιστικές Μηχανές Turing ταυτίζεται με το σύνολο των γλωσσών που γίνονται αποφασίζονται από τις Ντετερμινιστικές Μηχανές Turing. Δηλαδή η επαύξηση του υπολογιστικού μοντέλου με ένα τόσο ισχυρό χαρακτηριστικό όπως ο μη-ντετερμινισμός δεν αυξάνει

το σύνολο των γλωσσών που αποφασίζονται από τις Ντετερμινιστικές Μηχανές Turing. Από αυτή την άποψη το Θεώρημα 7.1 μπορεί να θεωρηθεί σαν ένα ακόμη τεκμήριο υπέρ του αξιώματος των Church-Turing.

Άσκηση 7.1. Να αποδείξετε ότι το Πρόβλημα του Περιοδεύοντος Πωλητή (σε μορφή προβλήματος απόφασης) ανήκει στην κλάση $\mathbf{NTIME}[n^2]$.

Λύση. Ορίζουμε μια NDTM που ''μαντεύει'' (λειτουργώντας μη-ντετερμινιστικά) και γράφει σε κάποια ταινία την συμβολοσειρά που κωδικοποιεί την περιοδεία ελάχιστου μήκους. Αυτό μπορεί να γίνει ορίζοντας τη σχέση μετάβασης ώστε να δημιουργεί όλες τις δυνατές περιοδείες. Στη συνέχεια, η μηχανή (λειτουργώντας ντετερμινιστικά) αποδέχεται την είσοδο αν και μόνο αν η συμβολοσειρά που έγραψε αποτελεί την κωδικοποίηση μιας περιοδείας μήκους το πολύ B. Αν υπάρχει τέτοια περιοδεία, η μηχανή θα τη ''μαντέψει'' και θα αποδεχθεί την είσοδο. Διαφορετικά, η μηχανή είναι υποχρεωμένη να απορρίψει την είσοδο. Αυτές οι ενέργειες μπορούν να γίνουν σε χρόνο $\mathrm{O}(n^2)$. Για τις λεπτομέρειες, βλ. [33, Παράδειγμα 2.9]. □

7.2 Η Κλάση NP

Μια από τις σημαντικότερες κλάσεις πολυπλοκότητας είναι η κλάση **NP**, δηλαδή η κλάση των προβλημάτων που μπορούν να λυθούν σε πολυωνυμικό μη-ντετερμινιστικό χρόνο. Τυπικά,

$$\mathbf{NP} = \bigcup_{k \geq 0} \mathbf{NTIME}[n^k]$$

Άσκηση 7.2. Να αποδείξετε ότι το πρόβλημα της k-Ικανοποιησιμότητας ανήκει στο **NP** για κάθε φυσικό αριθμό $k \geq 1$. Τι ισχύει για το πρόβλημα της 2-Ικανοποιησιμότητας;

Λύση. Έστω ϕ μια οποιαδήποτε λογική πρόταση σε k-ΣΚΜ. Ορίζουμε μια NDTM που με είσοδο την πρόταση ϕ, ''μαντεύει'' (λειτουργώντας μη-ντετερμινιστικά) την ''καλύτερη δυνατή'' αποτίμηση για τις μεταβλήτες της ϕ (π.χ. την αποτιμήση που ικανοποιεί τους περισσότερους όρους). Στη συνέχεια, η NDTM λειτουργεί ντετερμινιστικά και αποδέχεται τη ϕ (δηλαδή απαντάει ότι η ϕ είναι ικανοποιήσιμη) αν και μόνο αν η αποτίμηση που ''μάντεψε'' ικανοποιεί την ϕ. Αυτές οι ενέργειες μπορούν εύκολα να γίνουν σε πολυωνυμικό μη-ντετερμινιστικό χρόνο. □

Υπάρχει ένας ισοδύναμος (και πολύ πιο ενδιαφέρον) ορισμός του **NP** που βασίζεται στον τρόπο με τον οποίο λειτουργεί μια NDTM πολυωνυμικού χρόνου. Έστω μία δυαδική σχέση $R \subseteq \Sigma^* \times \Sigma^*$. Η R καλείται *πολυωνυμικά αποφασίσιμη* (polynomially decidable) αν υπάρχει μία DTM πολυωνυμικού χρόνου που για κάθε $x, y \in \Sigma^*$, αποφασίζει αν $(x, y) \in R$. Η R καλείται *πολυωνυμικά ισορροπημένη* (polynomially balanced) αν για κάθε $(x, y) \in R$, το μήκος του y είναι πολυωνυμικό στο μήκος του x (τυπικά, υπάρχει σταθερά $k \geq 1$ τέτοια ώστε $|y| \leq |x|^k$).

Θεώρημα 7.2 (Χαρακτηρισμός του NP). *Μια γλώσσα $\mathcal{L}$ ανήκει στο* **NP** *αν και μόνο αν υπάρχει μία πολυωνυμικά αποφασίσιμη και πολυωνυμικά ισορροπημένη σχέση R τέτοια ώστε*

$$\mathcal{L} = \{x \in \Sigma^* : (x, y) \in R \text{ για κάποιο } y \in \Sigma^*\}$$

Απόδειξη. Αρχικά υποθέτουμε ότι υπάρχει μία τέτοια σχέση R. Τότε η $\mathcal{L}$ αποφασίζεται από μια NDTM N που λειτουργεί ως εξής: Δεδομένης εισόδου x, η N προσπαθεί να "μαντέψει" (λειτουργώντας μη-ντετερμινιστικά) κάποια συμβολοσειρά y με μήκος $|y| \leq |x|^k$ τέτοια ώστε $(x, y) \in R$. Στη συνέχεια, η N λειτουργεί ντετερμινιστικά και αποδέχεται το x αν και μόνο αν $(x, y) \in R$. Με άλλα λόγια, αρχικά οι κλάδοι υπολογισμού της $N(x)$ δημιουργούν όλες τις διαφορετικές συμβολοσειρές y με μήκος το πολύ $|x|^k$. Η N αποδέχεται το x αν και μόνο αν κάποιος από τους κλάδους υπολογισμού έχει δημιουργήσει ένα y τέτοιο ώστε $(x, y) \in R$ (βλ. επίσης Ασκήσεις 7.1 και 7.2).

Εξ' ορισμού η N αποφασίζει τη γλώσσα $\mathcal{L}$. Επειδή οι συμβολοσειρές y έχουν μήκος το πολύ $|x|^k$ και το $(x, y) \in R$ αποφασίζεται από μία DTM πολυωνυμικού χρόνου, ο χρόνος εκτέλεσης της N είναι πολυωνυμικός στο μέγεθος της εισόδου x.

Για το αντίστροφο, έστω γλώσσα $\mathcal{L} \in \mathbf{NP}$ και μία NDTM N που αποφασίζει την $\mathcal{L}$ σε χρόνο n^k. Ορίζουμε τη σχέση R ως εξής: $(x, y) \in R$ αν και μόνο αν το y είναι η κωδικοποίηση ενός υπολογισμού αποδοχής της $N(x)$. Εξ' ορισμού,

$$\mathcal{L} = \{x \in \Sigma^* : (x, y) \in R \text{ για κάποιο } y \in \Sigma^*\}$$

Η R είναι πολυωνυμικά ισορροπημένη γιατί κάθε κλάδος υπολογισμού της $N(x)$ έχει μήκος το πολύ $|x|^k$. Η R είναι πολυωνυμικά αποφασίσιμη γιατί μπορεί να ελεγχθεί ντετερμινιστικά σε γραμμικό χρόνο αν μια συμβολοσειρά y αποτελεί την κωδικοποίηση ενός υπολογισμού αποδοχής της $N(x)$. □

Το Θεώρημα 7.2 αποτελεί τον καλύτερο ίσως τρόπο για να κατανοήσουμε την κλάση **NP**. Σύμφωνα με το Θεώρημα 7.2, κάθε γλώσσα / πρόβλημα που ανήκει στο **NP** έχει μια αξιοσημείωτη ιδιότητα: Για κάθε είσοδο x που ανήκει στην γλώσσα, υπάρχει ένα "*συνοπτικό πιστοποιητικό*" y που μπορεί να ελεγχθεί από μία DTM πολυωνυμικού χρόνου και να πιστοποιήσει ότι το x ανήκει στη γλώσσα. Αντίθετα, κανένα τέτοιο "πιστοποιητικό" y δεν υπάρχει για τα x που δεν ανήκουν στη γλώσσα. Δεν γνωρίζουμε πως να υπολογίσουμε ένα τέτοιο "συνοπτικό πιστοποιητικό" σε πολυωνυμικό (ντετερμινιστικό) χρόνο. Γνωρίζουμε όμως ότι μια συμβολοσειρά ανήκει στη γλώσσα αν και μόνο αν υπάρχει ένα "συνοπτικο πιστοποιητικό" για το γεγονός αυτό.

Για παράδειγμα, το "πιστοποιητικό" ότι μια λογική πρόταση ϕ είναι ικανοποιήσιμη (δηλαδή ανήκει στη γλώσσα που αποτελείται από τις ικανοποιήσιμες λογικές προτάσεις σε ΣΚΜ) είναι μια αποτίμηση των μεταβλητών που ικανοποιεί την ϕ. Η αποτίμηση περιγράφεται από μια δυαδική συμβολοσειρά με μήκος ίσο με τον αριθμό των μεταβλητών της ϕ. Εφόσον μια αποτίμηση που ικανοποιεί την ϕ είναι διαθέσιμη, μπορεί να ελεγθεί και να επιβεβαιώσει την ικανοποιησιμότητα της ϕ σε πολυωνυμικό χρόνο. Το αντίστοιχο "πιστοποιητικό" για το Πρόβλημα του Περιοδεύοντος Πωλητή είναι μία περιοδεία με μήκος που δεν ξεπερνά το B. Μια περιοδεία είναι μια μετάθεση των σημείων εισόδου. Το μήκος της μπορεί εύκολα να υπολογισθεί και να συγκριθεί με το B σε γραμμικό χρόνο.

Άσκηση 7.3. Να αποδείξετε ότι το **NP** είναι κλειστό ως προς την ένωση και την τομή. Δηλαδή να δείξετε ότι αν δύο γλώσσες $\mathcal{L}_1, \mathcal{L}_2 \in \mathbf{NP}$, τότε και οι γλώσσες $\mathcal{L}_1 \cup \mathcal{L}_2, \mathcal{L}_1 \cap \mathcal{L}_2 \in \mathbf{NP}$.

Άσκηση 7.4. Να αποδείξετε ότι το **NP** είναι κλειστό ως προς την πολυωνυμική αναγωγή. Δηλαδή να δείξετε ότι αν το πρόβλημα Π_1 ανάγεται πολυωνυμικά στο πρόβλημα Π_2 και το $\Pi_2 \in \mathbf{NP}$, τότε και το $\Pi_1 \in \mathbf{NP}$.

7.3 NP-Πληρότητα

Η κλάση **P** είναι υποσύνολο της κλάσης **NP** επειδή οι DTM είναι ειδική περίπτωση των NDTM. Τα περισσότερα προβλήματα βελτιστοποίησης που συναντήσαμε ανήκουν στο **NP** και λύνονται σε πολυωνυμικό ντετερμινιστικό χρόνο. Από την άλλη πλευρά, υπάρχουν προβλήματα που ανήκουν στο **NP** και δεν είναι γνωστό αν λύνονται σε πολυωνυμικό χρόνο, όπως το Πρόβλημα του Περιοδεύοντος Πωλητή και το πρόβλημα της k-Ικανοποιησιμότητας. Το σημαντικότερο ανοικτό ερώτημα στη Θεωρητική Επιστήμη των Υπολογιστών είναι αν τα προβλήματα αυτής της κατηγορίας λύνονται σε πολυωνυμικό ντετερμινιστικό χρόνο ή όχι.

Μια ισοδύναμη διατύπωση του ίδιου ερωτήματος αφορά στη διαφορετικότητα των κλάσεων **P** και **NP**. Με άλλα λόγια, το ερώτημα είναι αν είναι δυνατή η ντετερμινιστική προσομοίωση κάθε NDTM πολυωνυμικού χρόνου με πολυωνυμική χρονικής επιβάρυνση. Το ερώτημα αυτό έχει τεράστια πρακτική και θεωρητική σημασία, αλλά προς το παρόν δεν μοιάζει ώριμο να απαντηθεί. Παρόλα αυτά, αποτελεί κοινή επιστημονική πεποίθηση ότι το **P** είναι γνήσιο υποσύνολο του **NP**.

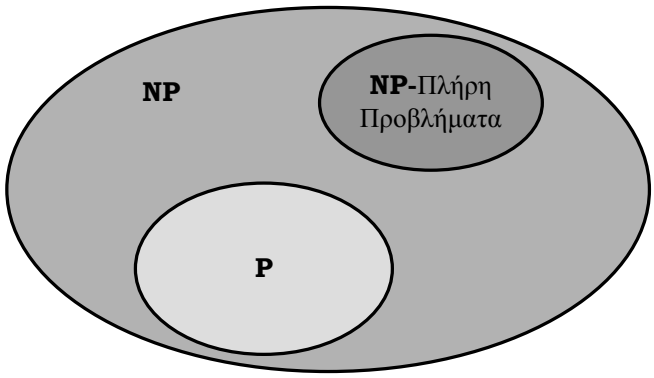
Αν όντως το **P** διαφέρει από το **NP**, πρέπει να υπάρχουν κάποια προβλήματα στο **NP** που δεν λύνονται σε πολυωνυμικό χρόνο. Γνωρίζουμε ότι τα πλήρη προβλήματα μιας κλάσης "συνοψίζουν" την υπολογιστική της δυσκολία. Αν λοιπόν υπάρχουν κάποια προβλήματα στο **NP** που δεν λύνονται σε πολυωνυμικό χρόνο, τότε ανάμεσα σε αυτά πρέπει να είναι και όλα τα **NP**-πλήρη προβλήματα (εφόσον φυσικά τέτοια προβλήματα υπάρχουν). Υπενθυμίζεται ότι ένα πρόβλημα Π είναι **NP**-πλήρες (**NP**-complete) αν ανήκει στο **NP** και κάθε άλλο πρόβλημα $\Pi' \in \mathbf{NP}$ ανάγεται πολυωνυμικά στο Π.

Θεώρημα 7.3. *Έστω Π ένα **NP**-πλήρες πρόβλημα. Το Π λύνεται σε πολυωνυμικό ντετερμινιστικό χρόνο αν και μόνο αν* $\mathbf{P} = \mathbf{NP}$.

Απόδειξη. Αν $\mathbf{P} = \mathbf{NP}$, όλα τα προβλήματα στο **NP** (και τα **NP**-πλήρη) λύνονται σε πολυωνυμικό χρόνο. Αντίστροφα, έστω ότι το Π λύνεται σε πολυωνυμικό χρόνο (οπότε $\Pi \in \mathbf{P}$). Αφού κάθε πρόβλημα $\Pi' \in \mathbf{NP}$ ανάγεται πολυωνυμικά στο Π, κάθε πρόβλημα $\Pi' \in \mathbf{NP}$ λύνεται σε πολυωνυμικό χρόνο και ανήκει στο **P** (βλ. επίσης Άσκηση 6.7). Συνεπώς, $\mathbf{P} = \mathbf{NP}$. □

Επισήμανση. Μια άμεση συνέπεια του Θεωρήματος 7.3 είναι ότι αν $\mathbf{P} \neq \mathbf{NP}$, κανένα **NP**-πλήρες πρόβλημα δεν λύνεται σε πολυωνυμικό χρόνο. Αν όπως πιστεύεται, οι κλάσεις **P** και **NP** διαφέρουν, το σύνολο των προβλημάτων που είναι πλήρη για το **NP** αποτελούν ένα υποσύνολο του **NP** ξένο με την κλάση **P**. Μάλιστα, γνωρίζουμε ότι αν $\mathbf{P} \neq \mathbf{NP}$, υπάρχουν κάποια προβλήματα, τα λεγόμενα **NP**-ενδιάμεσα (intermediate) προβλήματα, που δεν ανήκουν στο **P** και δεν είναι **NP**-πλήρη (βλ. Σχήμα 7.2). Το πιο γνωστό υποψήφιο **NP**-ενδιάμεσο πρόβλημα είναι το πρόβλημα του ισομορφισμού δύο γραφημάτων (graph isomorphism).

Το Θεώρημα 7.3 εξηγεί σε ένα βαθμό γιατί η Θεωρία της **NP**-πληρότητας έχει πολλές και σημαντικές πρακτικές εφαρμογές. Έστω ότι προσπαθούμε να σχεδιάσουμε έναν αποδοτικό αλγόριθμο για κάποιο πρόβλημα που ανήκει στο **NP** (η συντριπτική πλειοψηφία των πρακτικών



Σχήμα 7.2: Η εικόνα της κλάσης **NP** αν $\mathbf{P} \neq \mathbf{NP}$.

προβλημάτων ανήκει στο **NP**). Έχοντας ταυτίσει τους αποδοτικούς αλγόριθμους με τους αλγόριθμους πολυωνυμικού χρόνου, είναι σημαντικό να αποφασίσουμε αν το πρόβλημα λύνεται σε πολυωνυμικό χρόνο (ανήκει στο **P**) ή είναι **NP**-πλήρες. Μετά από μερικές αποτυχημένες προσπάθειες για αλγόριθμο πολυωνυμικού χρόνου, συνήθως στρέφουμε το ενδιαφέρον μας στη διατύπωση απόδειξης **NP**-πληρότητας. Η αιτία που δεν καταλήγουμε σε αλγόριθμο πολυωνυμικού χρόνου συχνά οδηγεί αυτόματα σε απόδειξη **NP**-πληρότητας. Άλλες φορές, η αιτία που δεν καταφέρνουμε να διατυπώσουμε μια απόδειξη **NP**-πληρότητας οδηγεί σε αλγόριθμο πολυωνυμικού χρόνου.

Άσκηση 7.5. Έστω προβλήματα $\Pi_1, \Pi_2 \in \mathbf{NP}$ τέτοια ώστε το Π_1 ανάγεται πολυωνυμικά στο Π_2. Να εξηγήσετε ποιες από τις παρακάτω προτάσεις είναι αληθείς, ψευδείς, ή αποτελούν ανοικτά προβλήματα.

1. Αν $\Pi_1 \in \mathbf{P}$, τότε και $\Pi_2 \in \mathbf{P}$.
2. Αν $\Pi_2 \in \mathbf{P}$, τότε και $\Pi_1 \in \mathbf{P}$.
3. Αν το Π_2 δεν είναι **NP**-πλήρες, ούτε και το Π_1 είναι.
4. Αν αμφότερα τα Π_1 και Π_2 είναι **NP**-πλήρη προβλήματα, τότε και το Π_2 ανάγεται στο Π_1.

Λύση. Το (1) αποτελεί ανοικτό πρόβλημα, αφού αληθεύει μόνο αν $\mathbf{P} = \mathbf{NP}$. Αν $\mathbf{P} \neq \mathbf{NP}$, ένα πρόβλημα $\Pi_1 \mathbf{P}$ ανάγεται σε κάποιο **NP**-πλήρες Π_2 αλλά το $\Pi_2 \notin \mathbf{P}$. Τα (2), (3), (4) είναι αληθείς προτάσεις. □

Άσκηση 7.6. Είναι το **NP** κλειστό ως προς το συμπλήρωμα; Δηλαδή αν μια γλώσσα $\mathcal{L} \in \mathbf{NP}$, είναι αλήθεια ότι το συμπλήρωμά της $\overline{\mathcal{L}}$ ανήκει επίσης στο **NP**;

Λύση. Δεν γνωρίζουμε αν **NP** είναι κλειστό ως προς το συμπλήρωμα. Αν $\mathbf{P} = \mathbf{NP}$, γνωρίζουμε ήδη ότι το **P** είναι κλειστό ως προς το συμπλήρωμα (βλ. Άσκηση 6.5). Αν $\mathbf{P} \neq \mathbf{NP}$, αποτελεί κοινή πεποίθηση ότι το **NP** δεν είναι κλειστό ως προς το συμπλήρωμα. Η απόδειξη αυτής της πεποίθησης αποτελεί σημαντικό ανοικτό πρόβλημα. Η διαφορά σε σχέση με το **P** έγκειται στην ασυμμετρία του ορισμού αποδοχής μιας NDTM. Για περισσότερες λεπτομέρειες, βλ. π.χ. [33, Κεφάλαιο 10]. □

7.3.1 NP-Πλήρη Προβλήματα

Έχοντας επιχειρηματολογήσει για τις σημαντικές εφαρμογές της **NP**-πληρότητας, απομένει να αποδείξουμε ότι όντως υπάρχουν προβλήματα που είναι **NP**-πλήρη.

Ικανοποιησιμότητα Λογικών Προτάσεων

Το πρώτο πρόβλημα που αποδείχτηκε πλήρες για το **NP** είναι αυτό της Ικανοποιησιμότητας Λογικών Προτάσεων σε Συζευκτική Κανονική Μορφή (ΣΚΜ), η απλά το πρόβλημα της Ικανοποιησιμότητας. Στο πρόβλημα της Ικανοποιησιμότητας (Satisfiability), δίνεται μια λογική πρόταση ϕ σε ΣΚΜ (χωρίς περιορισμούς στον αριθμό ατόμων σε κάθε όρο). Το ζητούμενο είναι αν η ϕ είναι ικανοποιήσιμη, δηλαδή αν υπάρχει μια αποτίμηση των λογικών μεταβλητών που επαληθεύει τη ϕ. Η Ικανοποιησιμότητα αποδείχτηκε **NP**-πλήρης στις αρχές της δεκαετίας του 70 από τον S. Cook [8] και τον L. Levin [28] σε ανεξάρτητες εργασίες. Το Θεώρημα 7.4 είναι γνωστό σαν Θεώρημα των Cook - Levin.

Θεώρημα 7.4. *Το πρόβλημα της Ικανοποιησιμότητας είναι* **NP***-πλήρες.*

Απόδειξη (βασική ιδέα). Είναι εύκολο να αποδειχθεί ότι η Ικανοποιησιμότητα ανήκει στο **NP** (βλ. επίσης Άσκηση 7.2). Επομένως, απομένει να δείξουμε ότι κάθε γλώσσα $\mathcal{L} \in \mathbf{NP}$ ανάγεται πολυωνυμικά στην Ικανοποιησιμότητα. Η απόδειξη ουσιαστικά κωδικοποιεί τη λειτουργία μιας NDTM σε μία λογική πρόταση σε ΣΚΜ.

Έστω μια οποιαδήποτε γλώσσα $\mathcal{L} \in \mathbf{NP}$, και έστω μία NDTM N με μία ταινία[2] που αποφασίζει την $\mathcal{L}$ σε χρόνο $t(n)$. Για κάθε είσοδο x, θα κατασκευάσουμε σε πολυωνυμικό χρόνο μία λογική πρόταση ϕ_x σε ΣΚΜ που είναι ικανοποιήσιμη αν και μόνο αν $x \in \mathcal{L}$.

Έστω ότι το αλφάβητο ταινίας της N είναι $\{a_1, \ldots, a_m\}$ και το σύνολο καταστάσεων είναι $\{q_0, q_1, \ldots, q_{\ell-1}\}$. Επίσης, έστω n το μήκος της εισόδου x. Αφού η N ολοκληρώνει τον υπολογισμό της σε χρόνο $t(n)$, η κεφαλή της ταινίας δεν επισκέπτεται περισσότερα από $t(n)$ κελιά. Η λογική πρόταση ϕ_x χρησιμοποιεί τρία είδη λογικών μεταβλητών:

1. ΚΕΛΙ(i, j, t) που δηλώνει ότι "το κελί i περιέχει το σύμβολο a_j τη χρονική στιγμή t", $1 \leq i \leq t(n)$, $1 \leq j \leq m$, $1 \leq t \leq t(n)$. Συνολικά $mt^2(n)$ μεταβλητές.

2. ΚΕΦΑΛΗ(i, t) που δηλώνει ότι "η κεφαλή της ταινίας βρίσκεται στο κελί i τη χρονική στιγμή t", $1 \leq i \leq t(n)$, $1 \leq t \leq t(n)$. Συνολικά $t^2(n)$ μεταβλητές.

3. ΚΑΤΑΣΤΑΣΗ(k, t) που δηλώνει ότι "η N βρίσκεται στην κατάσταση q_k τη χρονική στιγμή t", $0 \leq k \leq \ell - 1$, $1 \leq t \leq t(n)$. Συνολικά $\ell t(n)$ μεταβλητές.

Για κάθε χρονική στιγμή t, η τρέχουσα διαμόρφωση της N περιγράφεται πλήρως από ένα σύνολο $(m+1)t(n) + \ell$ λογικών μεταβλητών. Συνολικά χρησιμοποιούμε $(m+1)t^2(n) + \ell t(n)$ λογικές μεταβλητές για να περιγράψουμε $t(n)$ βήματα (μη-ντετερμινιστικού) υπολογισμού της N.

[2] Η υπόθεση ότι η N έχει μία μόνο ταινία γίνεται για λόγους απλότητας και χωρίς βλάβη της γενικότητας. Κάθε NDTM πολυωνυμικού χρόνου με k ταινίες μπορεί να προσομοιωθεί από μια NDTM πολυωνυμικού χρόνου με μία ταινία.

Η λογική πρόταση ϕ_x αποτελείται από τη σύζευξη επιμέρους λογικών προτάσεων που γράφονται σε ΣΚΜ χρησιμοποιώντας τις μεταβλητές που περιγράψαμε. Κάθε επιμέρους λογική πρόταση διατυπώνεται ώστε να είναι ικανοποιήσιμη αν και μόνο αν ισχύει η αντίστοιχη από τις παρακάτω συνθήκες:

1. Κάθε χρονική στιγμή, η κεφαλή της ταινίας βρίσκεται σε ένα και μόνο κελί.
2. Κάθε χρονική στιγμή, κάθε κελί περιέχει ένα και μόνο σύμβολο του αλφάβητου.
3. Κάθε χρονική στιγμή, η N βρίσκεται σε μία μόνο κατάσταση.
4. Αρχικά τα πρώτα n κελιά περιέχουν την είσοδο x και τα υπόλοιπα $t(n) - n$ περιέχουν το κενό σύμβολο. Επιπλέον, η N ξεκινάει στην αρχική κατάσταση q_0.
5. Κάποια χρονική στιγμή $t \leq t(n)$, η N φτάνει σε κατάσταση αποδοχής.
6. Κάθε χρονική στιγμή, μόνο το περιεχόμενο του κελιού στο οποίο βρίσκεται η κεφαλή της ταινίας μπορεί να μεταβληθεί.
7. Κάθε χρονική στιγμή, η επόμενη διαμόρφωση καθορίζεται από τη σχέση μετάβασης. Δηλαδή οι τιμές των μεταβλητών που περιγράφουν τις διαμορφώσεις της N σε δύο διαδοχικές χρονικές στιγμές πρέπει να υπακούουν στους κανόνες που θέτει η σχέση μετάβασης.

Οι παραπάνω συνθήκες συνοψίζουν τους κανόνες λειτουργίας μιας NDTM με μία ταινία. Η N αποδέχεται το x αν και μόνο αν υπάρχει μια ακολουθία διαμορφώσεων που ικανοποιεί τις παραπάνω συνθήκες. Αυτό συμβαίνει αν και μόνο αν υπάρχει μια αποτίμηση των λογικών μεταβλητών που επαληθεύει την ϕ_x. Εδώ ολοκληρώνεται η περιγραφή της βασικής ιδέας.

Για να ολοκληρωθεί η απόδειξη απαιτείται να δείξουμε ότι οι επιμέρους προτάσεις διατυπώνονται ισοδύναμα σε ΣΚΜ και ότι αυτή η μορφή μπορεί να υπολογισθεί από την περιγραφή της N και την είσοδο x σε πολυωνυμικό χρόνο. □

Έχουμε ήδη αποδείξει ότι η 2-Ικανοποιησιμότητα, όπου η λογική πρόταση έχει 2 το πολύ άτομα σε κάθε όρο, λύνεται σε πολυωνυμικό χρόνο (Θεώρημα 6.3). Στη συνέχεια, θα αποδείξουμε ότι η 3-Ικανοποιησιμότητα, όπου η λογική πρόταση έχει 3 το πολύ άτομα σε κάθε όρο, είναι **NP**-πλήρης.

Θεώρημα 7.5. *Το πρόβλημα της 3-Ικανοποιησιμότητας είναι* **NP***-πλήρες.*

Απόδειξη. Έχουμε ήδη αποδείξει ότι το πρόβλημα της 3-Ικανοποιησιμότητας ανήκει στο **NP** (βλ. Άσκηση 7.2). Θα αποδείξουμε ακόμη ότι η Ικανοποιησιμότητα ανάγεται πολυωνυμικά στην 3-Ικανοποιησιμότητα. Συγκεκριμένα, θα δείξουμε ότι μια λογική πρόταση σε ΣΚΜ μπορεί να μετατραπεί σε πολυωνυμικό χρόνο σε μία ισοδύναμη λογική πρόταση σε 3-ΣΚΜ.

Έστω ϕ μια πρόταση σε ΣΚΜ. Η μετατροπή συνίσταται στην αντικατάσταση κάθε όρου c_j με $k \geq 4$ άτομα από μια ομάδα όρων c'_j με 3 άτομα ανά όρο. Έστω όρος $c_j = \ell_{j_1} \vee \ldots \vee \ell_{j_k}$ της ϕ

με $k \geq 4$ άτομα. Ορίζουμε $k-3$ νέες μεταβλητές $z_{j_1}, \ldots, z_{j_{k-3}}$, και αντικαθιστούμε τον όρο c_j με τους όρους:

$$\begin{aligned} c'_j &= (\ell_{j_1} \vee \ell_{j_2} \vee z_{j_1}) \wedge (\neg z_{j_1} \vee \ell_{j_3} \vee z_{j_2}) \wedge (\neg z_{j_2} \vee \ell_{j_4} \vee z_{j_3}) \wedge \ldots \\ &\quad \wedge (\neg z_{j_{k-4}} \vee \ell_{j_{k-2}} \vee z_{j_{k-3}}) \wedge (\neg z_{j_{k-3}} \vee \ell_{j_{k-1}} \vee \ell_{j_k}) \end{aligned}$$

Η ϕ' προκύπτει από τη ϕ αντικαθιστώντας κάθε όρο c_j της ϕ με $k \geq 4$ άτομα με την αντίστοιχη ομάδα όρων c'_j. Η ϕ' μπορεί να υπολογιστεί σε πολυωνυμικό χρόνο.

Θα αποδείξουμε ότι η ϕ είναι ικανοποιήσιμη αν και μόνο αν η ϕ' είναι ικανοποιήσιμη (δηλ. ότι οι ϕ και ϕ' είναι ισοδύναμες). Αρκεί να δείξουμε ότι η ισοδυναμία ισχύει για κάθε ζευγάρι όρων c_j και c'_j. Έστω μια αποτίμηση που ικανοποιεί τον όρο c_j, και έστω ℓ_p το πρώτο άτομο που γίνεται αληθές από αυτή την αποτίμηση. Η ομάδα όρων c'_j ικανοποιείται αν συμπληρώσουμε την αποτίμηση με τις ακόλουθες τιμές για τις μεταβλητές z_{j_i}, $i = 1, \ldots, k-3$:

$$z_{j_i} = \begin{cases} 1 & \text{αν } i < p-1 \\ 0 & \text{αν } i \geq p-1 \end{cases}$$

Για το αντίστροφο, παρατηρούμε ότι η ομάδα όρων c'_j είναι ικανοποιήσιμη μόνο αν τουλάχιστον ένα από τα άτομα $\ell_1, \ldots, \ell_k$ είναι αληθές. Επομένως, κάθε αποτίμηση που ικανοποιεί την ομάδα όρων c'_j ικανοποιεί και τον όρο c_j. □

Η μεθοδολογία της απόδειξης του Θεωρήματος 7.5 χρησιμοποιείται συχνά στις αποδείξεις **NP**-πληρότητας. Ειδικότερα, για να ανάγουμε ένα πρόβλημα σε ένα άλλο επιχειρούμε να μετασχηματίσουμε τμήματα των στιγμιοτύπων του πρώτου προβλήματος σε τμήματα των στιγμιοτύπων του δεύτερου προβλήματος. (π.χ. μετασχηματισμός ενός όρου με πολλά άτομα σε ομάδα όρων με 3 άτομα σε κάθε όρο). Αν ένας τέτοιος μετασχηματισμός διατηρεί τις επιθυμητές ιδιότητες (π.χ. ικανοποιησιμότητα), μπορεί να εφαρμοστεί τμηματικά σε μεγαλύτερα στιγμιότυπα και να αποτελέσει τη βάση για μια απόδειξη **NP**-πληρότητας.

Άσκηση 7.7. Στο πρόβλημα της Μέγιστης 2-Ικανοποιησιμότητας (Maximum 2-Satisfiability) δίνεται ένα σύνολο όρων ϕ που καθένας αποτελείται από τη διάζευξη δύο ατόμων, και ένα φυσικός αριθμός . Το ζητούμενο είναι αν υπάρχει αποτίμηση που ικανοποιεί τουλάχιστον B όρους του ϕ. Να αποδείξετε ότι το πρόβλημα της Μέγιστης 2-Ικανοποιησιμότητας είναι **NP**-πλήρες.

Λύση. Αποδεικνύεται εύκολα ότι η Μέγιστη 2-Ικανοποιησιμότητα ανήκει στο **NP**. Για να αποδείξουμε ότι είναι **NP**-πλήρης, θα ανάγουμε το πρόβλημα της 3-Ικανοποιησιμότητας στη Μέγιστη 2-Ικανοποιησιμότητα. Η αναγωγή που θα περιγράψουμε βασίζεται στο ακόλουθο σύνολο 10 όρων που αποτελούνται από τέσσερις λογικές μεταβλητές:

$$C = \{(x), (y), (z), (w), (\neg x \vee \neg y), (\neg y \vee \neg z), (\neg x \vee \neg z), (x \vee \neg w), (y \vee \neg w), (z \vee \neg w)\}$$

Παρατηρούμε ότι η αποτίμηση που δεν ικανοποιεί τον όρο $x \vee y \vee z$ (δηλαδή δίνει στα x, y, και z τιμή 0) μπορεί να ικανοποιήσει το πολύ 6 από τους 10 όρους του C. Αντίθετα, κάθε αποτίμηση που ικανοποιεί τον όρο $x \vee y \vee z$ μπορεί να ικανοποιήσει τουλάχιστον 7 από τους 10 όρους του

C (για κατάλληλα επιλεγμένη τιμή του w). Αυτή η παρατήρηση αποτελεί και την κεντρική ιδέα της απόδειξης.

Έστω $\phi = c_1 \wedge \ldots c_m$ ένα στιγμιότυπο της 3-Ικανοποιησιμότητας. Υποθέτουμε ότι κάθε όρος της ϕ έχει ακριβώς τρία άτομα. Σε κάθε όρο $c_j = \ell_{j_1} \vee \ell_{j_2} \vee \ell_{j_3}$ αντιστοιχούμε το σύνολο όρων

$$\begin{aligned} C_j \quad = \quad & \{(\ell_{j_1}), (\ell_{j_2}), (\ell_{j_3}), (w_j), (\neg\ell_{j_1} \vee \neg\ell_{j_2}), (\neg\ell_{j_2} \vee \neg\ell_{j_3}), \\ & (\neg\ell_{j_1} \vee \neg\ell_{j_3}), (\ell_{j_1} \vee \neg w_j), (\ell_{j_2} \vee \neg w_j), (\ell_{j_3} \vee \neg w_j)\} \end{aligned}$$

όπου η μεταβλητή w_j είναι διαφορετική για κάθε σύνολο όρων C_j. Το σύνολο όρων ϕ' για το πρόβλημα της Μέγιστης 2-Ικανοποιησιμότητας αποτελείται από την ένωση των συνόλων όρων C_j, $j = 1, \ldots, m$. Η τιμή του B είναι $7m$. Το ϕ' μπορεί να υπολογιστεί από την ϕ σε πολυωνυμικό χρόνο.

Σύμφωνα με τα παραπάνω, υπάρχει αποτίμηση που ικανοποιεί ταυτόχρονα $7m$ όρους του ϕ' αν και μόνο αν αυτή η αποτίμηση ικανοποιεί ταυτόχρονα όλους τους όρους της ϕ, δηλαδή αν και μόνο αν η ϕ είναι ικανοποιήσιμη. □

Υπολογισμός Γραφοθεωρητικών Παραμέτρων

Στη συνέχεια, θα αποδείξουμε ότι ο υπολογισμός αρκετών γραφοθεωρητικών παραμέτρων όπως το μέγεθος του μέγιστου ανεξάρτητου συνόλου, το μέγεθος της ελάχιστης κλίκας, το μέγεθος του ελάχιστου συνόλου κάλυψης αντιστοιχεί σε **NP**-πλήρη προβλήματα απόφασης.

Ένα *ανεξάρτητο σύνολο* (independent set) ενός μη-κατευθυνόμενου γραφήματος είναι ένα σύνολο κορυφών χωρίς ακμές μεταξύ τους. Στο πρόβλημα του Μέγιστου Ανεξάρτητου Συνόλου (Maximum Independent Set), δίνεται ένα μη-κατευθυνόμενο γράφημα $G(V, E)$ και ένας φυσικός αριθμός B. Το ζητούμενο είναι αν το G περιέχει ανεξάρτητο σύνολο με B ή περισσότερες κορυφές.

Θεώρημα 7.6. *Το πρόβλημα του Μέγιστου Ανεξάρτητου Συνόλου είναι* **NP***-πλήρες.*

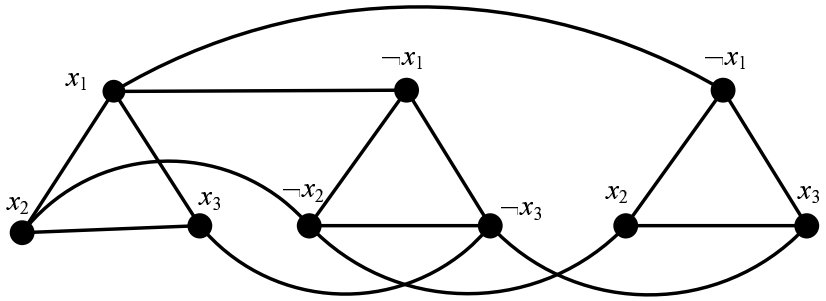
Απόδειξη. Μια NDTM πολυωνυμικού χρόνου μπορεί να "μαντέψει" ένα ανεξάρτητο σύνολο μεγέθους B (εφόσον υπάρχει) και να επιβεβαιώσει ότι όντως πρόκειται για ανεξάρτητο σύνολο σε χρόνο $O(n^2)$. Επομένως, το Μέγιστο Ανεξάρτητο Σύνολο (ΜΑΣ) ανήκει στο **NP**.

Για να αποδείξουμε ότι το ΜΑΣ είναι **NP**-πλήρες, θα ανάγουμε το πρόβλημα της 3-Ικανοποιησιμότητας στο ΜΑΣ. Έστω ϕ μια λογική πρόταση με m όρους και n μεταβλητές σε 3-ΣΚΜ. Χωρίς βλάβη της γενικότητας, θεωρούμε ότι κάθε όρος έχει ακριβώς 3 άτομα. Θα κατασκευάσουμε ένα γράφημα $G_\phi(V, E)$, $|V| = 3m$, που θα έχει ένα ανεξάρτητο σύνολο μεγέθους m αν και μόνο αν η ϕ είναι ικανοποιήσιμη.

Για την κατασκευή του γραφήματος G_ϕ, κάθε όρος της ϕ αναπαρίσταται με ένα τρίγωνο. Κάθε άτομο του όρου αντιστοιχεί σε μία κορυφή του τριγώνου. Δύο κορυφές διαφορετικών τριγώνων συνδέονται με ακμή αν και μόνο αν αντιστοιχούν σε συμπληρωματικά άτομα, δηλαδή άτομα που το ένα είναι η άρνηση του άλλου, π.χ. x_i και $\neg x_i$. Ένα παράδειγμα της αναγωγής φαίνεται στο Σχήμα 7.3

Τυπικά, έστω $\phi = c_1 \wedge \ldots \wedge c_m$, με κάθε όρο να έχει τη μορφή $c_j = \ell_{j_1} \vee \ell_{j_2} \vee \ell_{j_3}$. Το γράφημα $G_\phi(V, E)$ έχει σύνολο κορυφών

$$V = \{v_{ji} : j = 1, \ldots, m \text{ και } i = 1, 2, 3\}$$



Σχήμα 7.3: Ένα παράδειγμα της αναγωγής από την 3-Ικανοποιησιμότητα στο Μέγιστο Ανεξάρτητο Σύνολο. Η λογική πρόταση είναι $\phi = (x1 \vee x2 \vee x3) \wedge (\neg x1 \vee \neg x2 \vee \neg x3) \wedge (\neg x1 \vee x2 \vee x3)$ [33, Σχήμα 9.2].

και σύνολο ακμών

$$E = \{\{v_{ji}, v_{jk}\} : j = 1, \ldots, m \text{ και } i \neq k\} \cup \{\{v_{ji}, v_{pk}\} : j \neq p \text{ και } \ell_{j_i} = \neg \ell_{p_k}\}$$

Υπάρχει λοιπόν μια κορυφή για κάθε εμφάνιση ατόμου σε κάθε όρο. Η πρώτη ομάδα ακμών δημιουργεί τα m τρίγωνα που αντιστοιχούν στους όρους. Η δεύτερη ομάδα ακμών συνδέει όλες τις συμπληρωματικές εμφανίσεις της ίδιας μεταβλητής μεταξύ τους. Η κατασκευή του G_ϕ μπορεί να γίνει σε πολυωνυμικό χρόνο.

Κάθε τρίγωνο συνεισφέρει το πολύ μία κορυφή σε ένα ανεξάρτητο σύνολο. Άρα κανένα ανεξάρτητο σύνολο του G_ϕ δεν έχει περισσότερες από m κορυφές. Επίσης, δυο συμπληρωματικές εμφανίσεις της ίδιας μεταβλητής (π.χ. x_i και $\neg x_i$) δεν ανήκουν στο ίδιο ανεξάρτητο σύνολο γιατί συνδέονται με ακμή.

Η ιδέα είναι ότι όλα τα άτομα ενός ανεξάρτητου συνόλου μπορεί να έχουν την τιμή 1 σε μία αποτίμηση. Αν το G_ϕ έχει ανεξάρτητο σύνολο μεγέθους m, η αποτίμηση αυτή ικανοποιεί την ϕ γιατί δίνει τιμή 1 σε ένα άτομο από κάθε όρο. Η αποτίμηση αυτή είαι συνεπής γιατί δεν έχουμε συμπληρωματικές εμφανίσεις της ίδιας μεταβλητής στο ανεξάρτητο σύνολο.

Για το αντίστροφο, θεωρούμε μια αποτίμηση που ικανοποιεί την ϕ και επιλέγουμε ένα άτομο με τιμή 1 από κάθε όρο. Οι αντίστοιχες κορυφές αποτελούν ένα ανεξάρτητο σύνολο του G_ϕ γιατί ανήκουν σε διαφορετικά τρίγωνα και δεν πρόκειται για συμπληρωματικές εμφανίσεις της ίδιας μεταβλητής, αφού τα άτομα αληθεύουν ταυτόχρονα. Επειδή έχουμε ένα άτομο από κάθε όρο, το ανεξάρτητο σύνολο έχει m κορυφές. □

Στην πραγματικότητα αποδείξαμε ότι το πρόβλημα του Μέγιστου Ανεξάρτητου Συνόλου παραμένει **NP**-πλήρες ακόμη και οι κορυφές του γραφήματος εισόδου μπορούν να διαμεριστούν σε ξένα μεταξύ τους τρίγωνα. Από την άλλη πλευρά, υπάρχουν αρκετές ειδικές περιπτώσεις του Μέγιστου Ανεξάρτητου Συνόλου που είναι λύνονται σε πολυωνυμικό χρόνο. Για παράδειγμα, όταν το γράφημα εισόδου είναι διμερές, η απόδειξη του Θεωρήματος του Konig (βλ. π.χ. [12]) αποτελεί ουσιαστικά ένα πολυωνυμικό αλγόριθμο για το Μέγιστο Ανεξάρτητο Σύνολο.

Άσκηση 7.8. Να αποδείξετε ότι το πρόβλημα του Μέγιστου Ανεξάρτητου Συνόλου είναι στο **P** όταν όλες οι κορυφές του γραφήματος εισόδου έχουν βαθμό 2.

Λύση. Ένα γράφημα με όλες τις κορυφές του βαθμού 2 αποτελείται από την ένωση ξένων μεταξύ τους απλών κύκλων. Χωρίς βλάβη της γενικότητας, θεωρούμε ότι το γράφημα είναι συνεκτικό, δηλαδή πρόκειται για έναν απλό κύκλο (αν δεν είναι συνεκτικό, αντιμετωπίζουμε κάθε κύκλο ξεχωριστά). Το Μέγιστο Ανεξάρτητο Σύνολο έχει $\lfloor n/2 \rfloor$ κορυφές, όπου n ο συνολικός αριθμός των κορυφών του γραφηματος. □

Γενικότερα, για όλα τα **NP**-πλήρη προβλήματα, υπάρχουν ειδικές περιπτώσεις που λύνονται σε πολυωνυμικό χρόνο. Για παράδειγμα, τα προβλήματα της Ικανοποιησιμότητας (Θεώρημα 7.4) και της 3-Ικανοποιησιμότητας (Θεώρημα 7.5) είναι **NP**-πλήρη, αλλά το πρόβλημα της 2-Ικανοποιησιμότητας, δηλαδή η ειδική περίπτωση όπου κάθε όρος έχει το πολύ 2 άτομα, είναι στο **P** (Θεώρημα 6.3).

Έχοντας σαν σημείο εκκίνησης το πρόβλημα του Μέγιστου Ανεξάρτητου Συνόλου, μπορούμε να αποδείξουμε ότι μερικά ακόμη γραφοθεωρητικά προβλήματα είναι **NP**-πλήρη.

Άσκηση 7.9. Σε ένα μη-κατευθυνόμενο γράφημα, *κλίκα* (clique) είναι ένα σύνολο κορυφών όπου κάθε ζευγάρι συνδέεται με ακμή (με άλλα λόγια, το αντίστοιχο επαγόμενο υπογράφημα είναι πλήρες). Στο πρόβλημα της Μέγιστης Κλίκας (Maximum Clique), δίνεται ένα μη-κατευθυνόμενο γράφημα $G(V, E)$ και ένας φυσικός αριθμός B. Το ζητούμενο είναι αν το G περιέχει κλίκα με B ή περισσότερες κορυφές. Να αποδείξετε ότι το πρόβλημα της Μέγιστης Κλίκας είναι **NP**-πλήρες.

Λύση. Το πρόβλημα της Μέγιστης Κλίκας στο G είναι ισοδύναμο με το πρόβλημα του Μέγιστου Ανεξάρτητου Συνόλου στο συμπληρωματικό γράφημα $\overline{G}$. □

Άσκηση 7.10. Σε ένα μη-κατευθυνόμενο γράφημα, ένα σύνολο κορυφών C ονομάζεται *σύνολο κάλυψης* (vertex cover) όταν κάθε ακμή έχει τουλάχιστον ένα από τα άκρα της στο C. Στο πρόβλημα του Ελάχιστου Καλύμματος Κορυφών (Minimum Vertex Cover), δίνεται ένα μη-κατευθυνόμενο γράφημα $G(V, E)$ και ένας φυσικός αριθμός B. Το ζητούμενο είναι αν το G περιέχει κάλυμμα κορυφών με B ή λιγότερες κορυφές. Να αποδείξετε ότι το πρόβλημα του Ελάχιστου Καλύμματος Κορυφών είναι **NP**-πλήρες.

Λύση. Ένα σύνολο κορυφών $X \subseteq V$ είναι ανεξάρτητο σύνολο αν και μόνο αν οι υπόλοιπες κορυφές (δηλ. το σύνολο $V \setminus X$) αποτελούν ένα κάλυμμα κορυφών. Πράγματι, οι κορυφές του X δεν έχουν ακμές μεταξύ τους αν και μόνο αν κάθε ακμή έχει τουλάχιστον ένα άκρο της στο $V \setminus X$. Επομένως το $X \subseteq V$ αποτελεί ένα Μέγιστο Ανεξάρτητο Σύνολο αν και μόνο αν το $V \setminus X$ αποτελεί ένα Ελάχιστο Κάλυμμα Κορυφών. Η αναγωγή είναι λοιπόν τετριμμένη. Το G έχει κάλυμμα κορυφών με B ή λιγότερες κορυφές αν και μόνο αν έχει ανεξάρτητο σύνολο με $|V| - B$ ή περισσότερες κορυφές. □

7.4 Βιβλιογραφικές Αναφορές

Το βιβλίο των Garey και Johnson [20] είναι κλασικό και αποτελεί αναμφίβολα σημείο αναφοράς σε θέμα **NP**-πληρότητας. Το [20] διαπραγματεύεται την έννοια της **NP**-πληρότητας από θεωρητική και πρακτική σκοπιά και έχει μια εκτενέστατη λίστα από **NP**-πλήρη προβλήματα.

Η Θεωρία **NP**-πληρότητας παρουσιάζεται διεξοδικά ενταγμένη στο γενικότερο πλαίσιο της έννοιας της πληρότητας ως προς μία κλάση πολυπλοκότητας στο [33]. Το [33] έχει συντελέσει σημαντικά στην παρουσίαση της **NP**-πληρότητας σε αυτό το κεφάλαιο. Αρκετά παραδείγματα και ασκήσεις έχουν προέρχονται από το [33].

Ιστορικά, ο Jack Edmonds (π.χ. βλ. [16, 15] και αντίστοιχες αναφορές στο [33]) διατύπωσε για πρώτη φορά την εικασία ότι υπάρχουν πρακτικά προβλήματα που δεν λύνονται σε πολυωνυμικό χρόνο, και άρα το **P** είναι διαφορετικό από το **NP**. Σαν πρώτο υποψήφιο πρόβλημα, ο Edmonds πρότεινε το Πρόβλημα του Περιοδεύοντος Πωλητή. Ο Edmonds επίσης προδιέγραψε ότι πολλά από τα προβλήματα που δεν λύνονται σε πολυωνυμικό χρόνο συνδέονται στενά μεταξύ τους (έννοια του **NP**-πλήρους προβλήματος).

Το Θεώρημα των Cook-Levin και τα πρώτα **NP**-πλήρη προβλήματα αποδείχθηκαν στις εργασίες [8, 28]. Πολλά θεμελιώδη προβλήματα αποδείχθηκαν **NP**-πλήρη στην εργασία [26].

Βιβλιογραφία

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA., 1974.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[3] S. Baase. *Computer Algorithms: Introduction to Design and Analysis*. Adisson-Wesley, 2nd edition, 1988.

[4] R. Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16(1):87-90, 1958.

[5] R.E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[6] A. Borodin, M. Nielsen, and C. Rackoff. (Incremental) Priority Algorithms. In *Proc. of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA '02)*, pp. 752-761, 2002.

[7] G. Brassard and P. Bratley. *Algorithmics: Theory and Practice*. Prentice-Hall, Inc., 1998.

[8] S. Cook. The Complexity of Theorem Proving Procedures. In *Proc. of the 3th ACM Symp. on Theory of Computing (STOC '71)*, pp. 151-158, 1971.

[9] J.M. Cooley and J.W. Tuckey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):197-301, 1965.

[10] D. Coppersmith and S. Winograd. Matrix Multiplication via Arithmetic Progressions. In *Proc. of the 19th ACM Symp. on Theory of Computing (STOC '87)*, pp. 1-6, 1987.

[11] T.H Cormen, C.E Leiserson, R.L Rivest, and C. Stein. *Introduction to Algorithms (Second Edition)*. MIT Press and McGraw-Hill, 2nd edition, 2002.

[12] R. Diestel. *Graph Theory*. Springer, New York, 2nd edition, 2002.

[13] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644-654, 1976.

[14] E.W. Dijkstra. A Note on Two Problems in Connection with Graphs. *Numerische Mathematic*, 1:269-271, 1959.

[15] J. Edmonds. Optimum Branchings. *J. Res. National Bureau of Standards*, 17B(4):233-240, 1966.

[16] J. Edmonds. Systems of Distinct Representatives and Liner Algebra. *J. Res. National Bureau of Standards*, 17B(4):241-245, 1966.

[17] S. Even. *Graph Algorithms*. Computer Science Press, 1980.

[18] R.W. Floyd. Algorithm 97 (Shortest Path). *Communications of the ACM*, 5(6):345, 1962.

[19] L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[20] M.R Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of* **NP**-*completeness*. W.H. Freeman and Company, 1979.

[21] F. Harary. *Graph Theory*. Addison-Wesley, 1969.

[22] M. Held and R. Karp. A Dynamic Programming Approach to Sequencing Problems. *SIAM Journal on Applied Mathematics*, 10(1):196-210, 1962.

[23] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA., 1979.

[24] Bondy J.A and U.S.R. Murty. *Graph Theory with Applications*. North-Holland, 1976.

[25] A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata (in Russian). *Doklady Akademii Nauk SSSR*, 145:293-294, 1962.

[26] R.M. Karp. Reducibility Among Combinatorial Problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Communications*, pp. 85-103. Plenum Press, 1972.

[27] J.B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Travelling Salesman Problem. *Proceedings of the American Mathematical Society*, 7:48-50, 1956.

[28] L.A. Levin. Universal Sorting Problems (in Russian). *Problemy Peredachi Informatsii*, 9(3):265-266, 1973.

[29] H.R. Lewis and C.H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 1981.

[30] K. Mehlhorn. *Data Structures and Algorithms: Volumes I, II, and III*. Springer, 1984.

[31] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[32] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., 1982.

[33] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[34] W.J. Paul, N. Pippenger, E. Szemeredi, and W.T. Trotter. On Determinism versus Non-Determinism and Related Problems. In *Proc. of the 24th IEEE Symp. on Foundations of Computer Science (FOCS '83)*, pp. 429-438, 1983.

[35] R.C. Prim. Shortest Connection Networks and Some Generalizations. *Bell System Technical Journal*, 36:1389-1401, 1957.

[36] R. Raz and A. Shpilka. Lower Bounds for Matrix Product in Bounded Depth Circuits with Arbitrary Gates. In *Proc. of the 33rd ACM Symp. on Theory of Computing (STOC '01)*, pp. 409-418, 2001.

[37] R. Sedgewick. *Algorithms*. Addison-Wesley, 2nd edition, 1988.

[38] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997.

[39] V. Strassen. Gaussian Elimination is not Optimal. *Numerische Mathematik*, 13:354-356, 1969.

[40] R.E. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146-160, 1972.

[41] R.E. Tarjan. *Data Structures and Network Algorithms*. SIAM, 1983.

[42] V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[43] S. Warshall. A Theorem on Boolean Matrices. *Journal of the ACM*, 9(1):11-12, 1962.

Α Επίλυση Αναδρομικών Σχέσεων

Σε αυτή την ενότητα θα παρουσιάσουμε τρεις βασικές μεθόδους για την εκτίμηση της ασυμπτωτικής συμπεριφοράς αναδρομικών σχέσεων που προκύπτουν κατά την ανάλυση αναδρομικών αλγορίθμων. Συγκεκριμένα, θα παρουσιάσουμε τη μέθοδο της επανάληψης, τη μέθοδο της αντικατάστασης, και το Θεώρημα του Κυρίαρχου Όρου (Master Theorem). Για όλες τις αναδρομικές σχέσεις αυτής της ενότητας, θεωρούμε ότι $T(1) = \Theta(1)$ όπου δεν αναφέρεται αρχική συνθήκη.

Α.1 Μέθοδος της Επανάληψης

Η βασική ιδέα της μεθόδου της επανάληψης είναι να αναπτύξουμε την αναδρομική σχέση σε άθροισμα και να υπολογίσουμε τον κλειστό τύπο του. Για παράδειγμα, ας θεωρήσουμε την παρακάτω αναδρομική σχέση:

$$T(n) = \begin{cases} 2\,T(n/2) + n & \text{αν } n = 2^k \text{ για κάποιο } k \geq 2 \\ 2 & \text{αν } n = 2 \end{cases} \tag{Α.1}$$

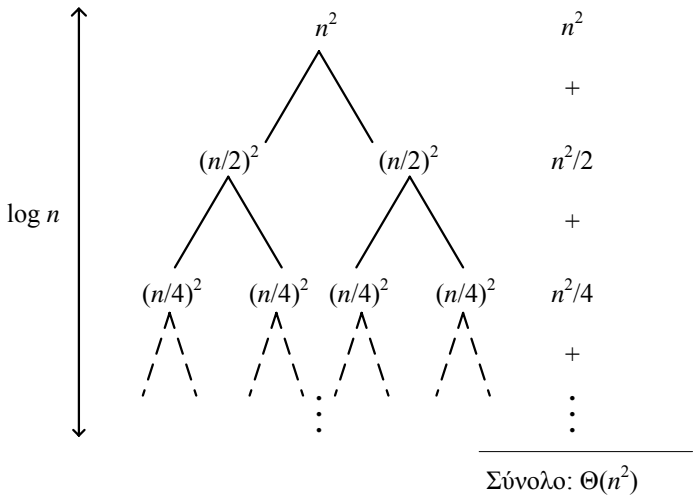
Η υπόθεση ότι το n πρέπει να είναι δύναμη 2 γίνεται για να αποφύγουμε την περίπτωση που το $n/2$ δεν είναι ακέραιος.

Αναπτύσσοντας την αναδρομική σχέση έχουμε:

$$\begin{aligned} T(n) &= n + 2\,T(n/2) \\ &= n + 2\,n/2 + 2^2\,T(n/4) \\ &= n + 2\,n/2 + 2^2 n/2^2 + 2^4\,T(n/2^3) \\ &\vdots \\ &= n + 2\,n/2 + 2^2\,n/2^2 + \cdots + 2^{i-1}\,n/2^{i-1} + 2^i\,T(n/2^i) \\ &= n\,i + 2^i\,T(n/2^i) \end{aligned}$$

Αν θέτουμε $i = \log n - 1$ στην παραπάνω ισότητα, η ανάπτυξη της αναδρομής τερματίζεται αφού $n/2^{\log n - 1} = 2$. Επομένως, $T(n) = n \log n$.

Η ιδέα της μεθόδου της επανάληψης είναι απλή, αλλά η εφαρμογή της συχνά οδηγεί σε πολύπλοκους αλγεβρικούς υπολογισμούς. Δύο είναι οι σημαντικότερες παράμετροι κατά την εφαρμογή της μεθόδου: ο αριθμός των επαναλήψεων για την πλήρη ανάπτυξη της αναδρομής (δηλ. μέχρι το σημείο που εφαμρόζεται η αρχική συνθήκη), και ο υπολογισμός του αθροίσματος των όρων που προκύπτουν από κάθε επίπεδο ανάπτυξης της αναδρομής. Η εφαρμογή της μεθόδου της επανάληψης όταν η αναδρομική σχέση ορίζεται με πάνω ή κάτω ακέραια μέρη μπορεί να οδηγήσει



Σχήμα Α.1: Το δέντρο της αναδρομής για την σχέση $T(n) = 2\,T(n/2) + n^2$.

σε πολύπλοκες αλγεβρικές εκφράσεις. Για την εκτίμηση της ασυμπτωτικής συμπεριφοράς μιας αναδρομικής σχέσης, μπορούμε να υποθέσουμε ότι το κλάσμα να δίνει πάντα ακέραιο αποτέλεσμα (π.χ. στην σχέση (Α.1) υποθέσαμε ότι το n είναι δύναμη του 2).

Μία απλή και χρήσιμη μέθοδος για την αναπαράσταση της ανάπτυξης μιας αναδρομικής σχέσης είναι το *δέντρο της αναδρομής* (recursion tree). Το δέντρο της αναδρομής επιτρέπει την καλύτερη οργάνωση των αλγεβρικών υπολογισμών κατά την ανάπτυξη της αναδρομικής σχέσης. Έστω η αναδρομική σχέση $T(n) = 2\,T(n/2) + n^2$. Το δέντρο της αναδρομής για αυτή την σχέση φαίνεται στο Σχήμα Α.1. Παρατηρούμε ότι η συνεισφορά κάθε γραμμής του δέντρου σε ύψος i (η ρίζα θεωρείται ότι βρίσκεται σε ύψος 0) είναι $n^2/2^i$. Αφού σε κάθε επίπεδο το n υποδιπλασιάζεται, το ύψος του δέντρου είναι $\log n$ (δηλαδή το δέντρο έχει $\log n + 1$ επίπεδα). Είναι λοιπόν

$$\sum_{i=0}^{\log n} n^2/2^i = n^2 \sum_{i=0}^{\log n} 2^{-i} < 2n^2 = \Theta(n^2)$$

όπου χρησιμοποιήσαμε ότι το άθροισμα των $k+1$ πρώτων όρων της γεωμετρικής προόδου με πρώτο όρο 1 και λόγο $1/2$ είναι $2 - 2^{-k}$.

Άσκηση Α.1. Να υπολογίσετε μια ακριβή ασυμπτωτική εκτίμηση για τη λύση της αναδρομικής σχέσης $T(n) = 4\,T(n/2) + n$ με αρχική συνθήκη $T(1) = 1$.

Λύση. Το δέντρο της αναδρομής έχει $\log n + 1$ επίπεδα επειδή το n υποδιπλασιάζεται σε κάθε επίπεδο. Για κάθε i, $i = 0, \ldots, \log n + 1$, η συνεισφορά του επιπέδου i είναι $2^i\,n$. Επομένως,

$$T(n) = \sum_{i=0}^{\log n} 2^i\,n = n \sum_{i=0}^{\log n} 2^i = n(2n-1) = \Theta(n^2)$$

όπου χρησιμοποιήσαμε ότι το άθροισμα των $k+1$ πρώτων όρων της γεωμετρικής προόδου με πρώτο όρο 1 και λόγο 2 είναι $2^{k+1} - 1$. □

Άσκηση Α.2. Να λύσετε την παρακάτω αναδρομική σχέση χρησιμοποιώντας τη μέθοδο της επανάληψης:

$$T(n) = \begin{cases} T(n/2) + 1 & \text{αν } n = 2^k \text{ για κάποιο } k \geq 1 \\ 1 & \text{αν } n = 1 \end{cases} \tag{Α.2}$$

Λύση. Αναπτύσσοντας την αναδρομική σχέση σε i επίπεδα, παίρνουμε $T(n) = i + T(n/2^i)$. Θέτοντας $i = \log n$, η ανάπτυξη της αναδρομής τερματίζεται αφού $n/2^{\log n} = 1$. Επομένως, $T(n) = \log n + 1$. □

Α.2 Μέθοδος της Αντικατάστασης

Η βασική ιδέα της μεθόδου της αντικατάστασης είναι να μαντέψουμε τη μορφή της λύσης και να χρησιμοποιήσουμε μαθηματική επαγωγή για να αποδείξουμε ότι η λύση είναι σωστή. Η μέθοδος είναι απλή και ισχυρή, αλλά μπορεί να εφαρμοστεί μόνο όταν μπορούμε να μαντέψουμε τη μορφή της λύσης.

Για παράδειγμα, ας θεωρήσουμε την αναδρομική σχέση $T(n) = 2\,T(n/2) + \Theta(n)$ με αρχική συνθήκη $T(1) = \Theta(1)$. Στην προηγούμενη ενότητα αποδείξαμε με τη μέθοδο της επανάληψης ότι η σχέση (Α.1), που είναι απλούστερη αλλά παρόμοια, έχει λύση $\Theta(n \log n)$. Θα χρησιμοποιήσουμε τη μέθοδο της αντικατάστασης για να αποδείξουμε ότι $T(n) = \Theta(n \log n)$.

Μπορούμε να αγνοήσουμε τις μικρές τιμές του n επειδή θέλουμε να αποδείξουμε μια ασυμπτωτική εκτίμηση για το $T(n)$. Έστω $c_1, c_2 \in \mathbb{R}^*_+$ οι σταθερές που "κρύβονται" στον προσθετικό όρο $\Theta(n)$ της αναδρομικής σχέσης[1]. Επαγωγικά υποθέτουμε ότι

$$c_1\,(n/2)\log(n/2) \leq T(n/2) \leq c_2\,(n/2)\log(n/2)$$

Πρέπει αποδείξουμε ότι

$$c_1\,n \log n \leq T(n) \leq c_2\,n \log n$$

Πράγματι, χρησιμοποιώντας την αναδρομική σχέση, παίρνουμε:

$$T(n) \geq 2\,c_1\,(n/2)\log(n/2) + c_1\,n = c_1\,n\,(\log n - 1) + c_1\,n = c_1\,n \log n$$

και

$$T(n) \leq 2\,c_2\,(n/2)\log(n/2) + c_2\,n = c_2\,n\,(\log n - 1) + c_2\,n = c_2\,n \log n$$

όπως απαιτείται. Αποδείξαμε λοιπόν ότι $T(n) = \Theta(n \log n)$.

Α.2.1 Αλλαγή Μεταβλητών

Σε κάποιες περιπτώσεις, μια αναδρομική σχέση μπορεί να απλοποιηθεί σημαντικά με αλλαγή μεταβλητών. Για παράδειγμα, ας θεωρήσουμε την σχέση $T(n) = 2\,T(\sqrt{n}) + \log n$ με αρχική συνθήκη $T(1) = \Theta(1)$. Θέτοντας $m = \log n$, παίρνουμε την σχέση $T(2^m) = 2\,T(2^{m/2}) + m$. Μπορούμε να μετονομάσουμε το $T(2^m)$ σε $S(m)$ και να οδηγηθούμε στην σχέση $S(m) = 2\,S(m/2) + m$ με αρχική συνθήκη $S(1) = \Theta(1)$. Έχουμε ήδη αποδείξει ότι η λύση αυτής της σχέσης είναι $S(m) = \Theta(m \log m)$. Πραγματοποιώντας την αντίστροφη αντικατάσταση, παίρνουμε $T(n) = \Theta(\log n \log\log n)$.

[1] Τυπικά, θεωρούμε ότι η αναδρομική σχέση είναι $T(n) = 2\,T(n/2) + f(n)$, όπου $f(n) = \Theta(n)$. Από τον ορισμό του ασυμπτωτικού συμβολισμού Θ, υπάρχουν σταθερές $c_1, c_2 \in \mathbb{R}^*_+$ τέτοιες ώστε $c_1\,n \leq f(n) \leq c_2\,n$.

Άσκηση Α.3. Να λύσετε τις παρακάτω αναδρομικές σχέσεις θεωρώντας σαν αρχική συνθήκη $T(1) = 1$.

1. $T(n) = T(n-1) + 3$.
2. $T(n) = T(n-1) + 2n$.
3. $T(n) = 2\,T(n/2) + n$.
4. $T(n) = T(n/3) + 1$.
5. $T(n) = T(n/2) + n$.
6. $T(n) = 2\,T(n-1) + 1$.

Λύση.

1. $T(n) = 3(n-1) + 1$.
2. $T(n) = 2n(n-1) + 1$.
3. $T(n) = n \log n$.
4. $T(n) = \log_3 n$.
5. $T(n) = 2n - 1$.
6. $T(n) = 2^n - 1$.

Α.3 Το Θεώρημα του Κυρίαρχου Όρου

Το Θεώρημα του Κυρίαρχου Όρου (Master Theorem) αποτελεί μια εύκολη συνταγή για την επίλυση αναδρομικών σχέσεων της μορφής $T(n) = a\,T(n/b) + f(n)$, όπου a και b είναι σταθερές και $f(n)$ είναι μία θετική συνάρτηση.

Το Θεώρημα του Κυρίαρχου Όρου διακρίνει τις ακόλουθες τρεις περιπτώσεις:

1. Αν $f(n) = \mathrm{O}(n^{\log_b a - \varepsilon})$ για κάποια σταθερά $\varepsilon > 0$, τότε $T(n) = \Theta(n^{\log_b a})$.
2. Αν $f(n) = \Theta(n^{\log_b a})$, τότε $T(n) = \Theta(n^{\log_b a} \log n)$.
3. Αν $f(n) = \Omega(n^{\log_b a + \varepsilon})$ για κάποια σταθερά $\varepsilon > 0$, και υπάρχει σταθερά $n_0 \in \mathbb{N}$ τέτοια ώστε για κάθε $n \geq n_0$, $a\,f(n/b) < f(n)$, τότε $T(n) = \Theta(f(n))$.

Κατά της εφαρμογή του Θεωρήματος του Κυρίαρχου Όρου, η μεγαλύτερη από τις συναρτήσεις $f(n)$ και $n^{\log_b a}$ (δηλ. ο κυρίαρχος όρος) καθορίζει τη λύση της σχέσης. Στην πρώτη περίπτωση, η $f(n)$ πρέπει να είναι πολυωνυμικά μικρότερη (δηλ. να υπολείπεται κατά έναν παράγοντα n^{ε}) από την $n^{\log_b a}$. Στη δεύτερη περίπτωση, οι συναρτήσεις $f(n)$ και $n^{\log_b a}$ πρέπει να έχουν την ίδια τάξη μεγέθους. Στην τρίτη περίπτωση, η $f(n)$ πρέπει να είναι πολυωνυμικά μεγαλύτερη από την $n^{\log_b a}$ και να ικανοποιεί τη συνθήκη $a\,f(n/b) < f(n)$ για μεγάλες τιμές του n. Υπάρχει ακόμη το

ενδεχόμενο η $f(n)$ να μην εντάσσεται σε κάποια από τις παραπάνω περιπτώσεις και το θεώρημα να μην εφαρμόζεται.

Σαν πρώτο παράδειγμα, θεωρούμε την σχέση $T(n) = 9\,T(n/3) + n$. Είναι $a = 9$, $b = 3$, και $n^{\log_b a} = n^{\log_3 9} = n^2$. Αφού $f(n) = n$, εφαρμόζουμε την πρώτη περίπτωση του θεωρήματος και παίρνουμε $T(n) = \Theta(n^2)$.

Σαν δεύτερο παράδειγμα, θεωρούμε την σχέση $T(n) = T(2n/3) + 1$. Είναι $a = 1$, $b = 3/2$, και $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1 = f(n)$. Επομένως εφαρμόζουμε τη δεύτερη περίπτωση του θεωρήματος και παίρνουμε $T(n) = \Theta(\log n)$.

Για την αναδρομική σχέση $T(n) = 3\,T(n/4) + n \log n$ είναι $a = 3$, $b = 4$, και $n^{\log_b a} = n^{\log_4 3} \approx n^{0.793}$. Αφού $f(n) = n \log n$ και $3\,f(n/4) < f(n)$ εφαρμόζουμε την τρίτη περίπτωση του θεωρήματος και παίρνουμε $T(n) = \Theta(n \log n)$.

Σαν ένα τελευταίο παράδειγμα, θεωρούμε την σχέση $T(n) = 2\,T(n/2) + n \log n$. Παρατηρούμε ότι δεν μπορούμε να εφαρμόσουμε το θεώρημα γιατί η ποσότητα $n^{\log_b a} = n^{\log_2 2} = n$ δεν είναι *πολυωνυμικά μικρότερη* από τη συνάρτηση $f(n) = n \log n$. Χρησιμοποιώντας το δέντρο της αναδρομής, μπορούμε να αποδείξουμε ότι $T(n) = \Theta(n \log^2 n)$. Οι λεπτομέρειες αφήνονται σαν άσκηση για τον αναγνώστη.

Άσκηση Α.4. Να υπολογίσετε ακριβείς ασυμπτωτικές εκτιμήσεις για τις αναδρομικές σχέσεις:

1. $T(n) = 6\,T(n/5) + n \log^6 n$.
2. $T(n) = 5\,T(n/6) + n$.
3. $T(n) = 3\,T(n/9) + \sqrt{n}$.
4. $T(n) = T(n-1) + n^2$.
5. $T(n) = T(n/5) + T(7n/10) + 3n$.
6. $T(n) = T(5n/9) + T(4n/9) + n$.

Λύση.

1. $T(n) = \Theta(n^{\log_5 6})$.
2. $T(n) = \Theta(n)$.
3. $T(n) = \Theta(\sqrt{n} \log n)$.
4. $T(n) = \Theta(n^3)$.
5. $T(n) = \Theta(n)$.
6. $T(n) = \Theta(n \log n)$.

Β Συνδετικά Δέντρα

Κάθε υπογράφημα που είναι δέντρο και περιλαμβάνει (καλύπτει, συνδέει) όλες τις κορυφές ενός γραφήματος ονομάζεται *συνδετικό δέντρο* (spanning tree) του γραφήματος. Άλλες ελληνικές αποδόσεις του ίδιου όρου είναι: γενετικό δέντρο, γεννητορικό δέντρο, παράγον δέντρο, διανύον δέντρο, και δέντρο-κάλυμμα δέντρο.

Θεώρημα Β.1. *Ένα γράφημα είναι συνεκτικό αν και μόνο αν έχει ένα συνδετικό δέντρο.*

Απόδειξη. Αν υπάρχει υπογράφημα που καλύπτει όλες τις κορυφές του γραφήματος και είναι δέντρο (άρα συνεκτικό), τότε το γράφημα δεν μπορεί παρά να είναι συνεκτικό (αφού η προσθήκη των ακμών που λείπουν απλώς "ενισχύει" τη συνεκτικότητα).

Αν το γράφημα είναι συνεκτικό, θεωρούμε ένα υπογράφημα T που αρχικά συμπίπτει με το γράφημα. Επομένως, το T καλύπτει όλες τις κορυφές του γραφήματος. Ενόσω το T έχει κύκλους, αφαιρούμε μία ακμή που βρίσκεται σε κύκλο. Το T παραμένει συνεκτικό αφού η αφαίρεση μιας ακμής που βρίσκεται σε κύκλο δεν επηρεάζει τη συνεκτικότητα. Αυτή η διαδικασία ολοκληρώνεται όταν το T γίνει ακυκλικό. Σε αυτή τη φάση, το T παραμένει συνεκτικό (από την κατασκευή) και συνεχίζει να καλύπτει όλες τις κορυφές του αρχικού γραφήματος (αφού ποτέ δεν αφαιρέσαμε κάποια κορυφή). Συνεπώς, το υπογράφημα που προκύπτει από αυτή τη διαδικασία είναι ένα συνδετικό δέντρο του αρχικού γραφήματος. □

Από το Θεώρημα 5.1, κάθε συνδετικό δέντρο ενός γραφήματος με n κορυφές έχει $n-1$ ακμές. Με άλλα λόγια, όλα τα συνδετικά δέντρα ενός γραφήματος έχουν τον ίδιο αριθμό ακμών.

Η απόδειξη του Θεωρήματος Β.1 δίνει έναν τρόπο να υπολογίσουμε ένα συνδετικό δέντρο ενός συνεκτικού γραφήματος. Ένας άλλος τρόπος είναι να θεωρήσουμε τις ακμές του G μία-προς-μία σε μια (οποιαδήποτε) συγκεκριμένη σειρά και να προσθέτουμε στο δέντρο κάθε νέα ακμή που δεν σχηματίζει κύκλο με τις υπάρχουσες. Άλλοι τρόποι είναι η Αναζήτηση Πρώτα σε Πλάτος και η Αναζήτηση Πρώτα σε Βάθος.

Β.1 Θεμελιώδεις Κύκλοι και Σύνολα Τομής

Έστω συνεκτικό γράφημα $G(V, E)$ με n κορυφές και m ακμές, και έστω $T(V, E_T)$ ένα συνδετικό δέντρο του G.

Παρατηρούμε ότι για κάθε συνδετικό δέντρο T, κάθε κύκλος του G πρέπει να περιέχει μια ακμή που δεν ανήκει στο T (αλλιώς το T θα περιείχε κύκλο). Από το Θεώρημα 5.1.(6), η προσθήκη κάθε ακμής που δεν ανήκει στο T (δηλαδή κάθε ακμής στο σύνολο $E \setminus E_T$) σχηματίζει ακριβώς έναν κύκλο. Κάθε τέτοιος κύκλος ονομάζεται *θεμελιώδης κύκλος* του G ως προς το συνδετικό

δέντρο T. Αφού το σύνολο $E \setminus E_T$ περιέχει $m - n + 1$ ακμές (αυτές οι ακμές λέγονται και *χορδές* (chords) του T), υπάρχουν $m - n + 1$ διαφορετικοί θεμελιώδεις κύκλοι του G ως προς κάθε συνδετικό δέντρο του.

Ένα σύνολο ακμών των οποίων η αφαίρεση κάνει το G μη-συνεκτικό ονομάζεται *σύνολο τομής* (cut set). Παρατηρούμε ότι για κάθε συνδετικό δέντρο T, κάθε σύνολο τομής του G πρέπει να περιέχει μια ακμή που ανήκει στο T (αλλιώς το T δεν θα ήταν συνεκτικό). Από το Θεώρημα 5.1.(3), η αφαίρεση κάθε ακμής του T δημιουργεί δύο συνεκτικές συνιστώσες. Έστω $e \in E_T$ μια ακμή του T, και έστω V_1 και V_2 οι κορυφές των δύο συνεκτικών συνιστωσών του $T \setminus e$ (δηλ. οι συνιστώσες που προκύπτουν από την αφαίρεση της ακμής e). Έστω $\delta(V_1, V_2)$ το σύνολο των ακμών που έχουν το ένα άκρο τους στο V_1 και το άλλο στο V_2. Προφανώς, $e \in \delta(V_1, V_2)$ και το $\delta(V_1, V_2)$ αποτελεί ένα ελαχιστικό σύνολο τομής για το γράφημα G. Κάθε σύνολο τομής που προκύπτει με αυτό τον τρόπο ονομάζεται *θεμελιώδες σύνολο τομής* του G ως προς το συνδετικό δέντρο T. Αφού το σύνολο E_T περιέχει $n - 1$ ακμές, υπάρχουν $n - 1$ διαφορετικά θεμελιώδη σύνολα τομής του G ως προς κάθε συνδετικό δέντρο του.

Σε ένα γράφημα, το σύνολο όλων των κύκλων (συνόλων τομής) αποτελεί ένα διανυσματικό χώρο. Η σημαντική παρατήρηση είναι ότι το σύνολο των θεμελιωδών κύκλων (συνόλων τομής) ως προς ένα οποιοδήποτε συνδετικό δέντρο του G αποτελεί βάση για το διανυσματικό χώρο όλων των κύκλων (αντίστοιχα συνόλων τομής). Επομένως, η διάσταση του διανυσματικού χώρου των κύκλων (συνόλων τομής) για ένα συνεκτικό γράφημα με n κορυφές και m ακμές είναι $m - n + 1$ ($n - 1$ αντίστοιχα).

Θεώρημα Β.2. *Κάθε κύκλος έχει άρτιο αριθμό κοινών ακμών με κάθε ελαχιστικό σύνολο τομής.*

Απόδειξη. Η αφαίρεση ενός ελαχιστικού συνόλου τομής χωρίζει τις κορυφές ενός γραφήματος σε δύο συνεκτικές συνιστώσες. Κάθε κύκλος χρησιμοποιεί ακμές του συνόλου τομής για να "επισκεφθεί" και να "αναχωρήσει" από μια συνεκτική συνιστώσα. Επιπλέον, ο αριθμός των "επισκέψεων" ενός κύκλου σε μια συνεκτική συνιστώσα πρέπει να είναι ίσος με τον αριθμό των "αναχωρήσεων". Έστω $\varepsilon_k = \alpha_k$ ο αριθμός των "επισκέψεων" και των "αναχωρήσεων" ενός κύκλου στη μία από τις δύο συνεκτικές συνιστώσες. Ο αριθμός των κοινών ακμών του κύκλου με το αντίστοιχο ελαχιστικό σύνολο τομής είναι $\varepsilon_k + \alpha_k = 2\varepsilon_k$, δηλαδή άρτιος. □

Άσκηση Β.1. Έστω T και T' δύο συνδετικά δέντρα ενός (συνεκτικού) γραφήματος G. Για κάθε ακμή $e \in T \setminus T'$, υπάρχει ακμή $e' \in T' \setminus T$, τέτοια ώστε το $(T' + e) \setminus e'$ είναι συνδετικό δέντρο του G.

Λύση. Αφού $e \in T \setminus T'$, το $T' + e$ περιέχει έναν απλό κύκλο που περιλαμβάνει την e. Έστω $e' \in T'$ μια ακμή αυτού του κύκλου που δεν ανήκει στο T. Μια τέτοια ακμή e' υπάρχει γιατί το T περιέχει την e αλλά δεν περιέχει τον κύκλο. Η αφαίρεση της e' αφαιρεί τον κύκλο και δεν επηρεάζει τη συνεκτικότητα. Συνεπώς, το $(T' + e) \setminus e'$ είναι συνδετικό δέντρο του G. □

Άσκηση Β.2. Έστω T και T' δύο συνδετικά δέντρα ενός (συνεκτικού) γραφήματος G. Για κάθε ακμή $e \in T \setminus T'$, υπάρχει ακμή $e' \in T' \setminus T$, τέτοια ώστε το $(T \setminus e) + e'$ είναι συνδετικό δέντρο του G.

Λύση. Αφαιρούμε την e από το T και προκύπτουν δύο συνεκτικές συνιστώσες. Έστω S και $V \setminus S$ τα σύνολα κορυφών των δύο συνιστωσών. Αφού το T' δεν περιέχει την e αλλά είναι συνεκτικό, υπάρχει μια ακμή $e' \in T'$ που συνδέει μια κορυφή του S και με μια κορυφή του $V \setminus S$ (με άλλα λόγια, η e' διασχίζει την τομή $\delta(S, V \setminus S)$). Η e' δεν ανήκει στο T γιατί η προσθήκη της θα σχημάτιζε κύκλο. Επομένως, η προσθήκη της e' στο $T \setminus e$ επαναφέρει τη συνεκτικότητα και το $(T \setminus e) + e'$ είναι συνδετικό δέντρο του G. □