Γραμμικές Λίστες

Δημήτρης Φωτάκης

Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων
Πανεπιστήμιο Αιγαίου

Αφηρημένος Τύπος Δεδομένων

- **ΑΤΔ**: **Σύνολο** (στιγμιότυπα) με **λειτουργίες** (μεθόδους) επί των στοιχείων.
- **Δομή Δεδομένων**: Υλοποίηση ενός ΑΤΔ.
  Στιγμιότυπα **αναπαρίστανται** και λειτουργίες **υλοποιούνται** με αλγορίθμους.
- **Σύνολο στιγμιότυπων**: σύνολα ή ακολουθίες **ακεραίων αριθμών**.
- **Στατική - Δυναμική ΔΔ**: Στιγμιότυπο (δεν) μεταβάλλεται.
- **Διατύπωση ΔΔ**: ορισμός αναπαράστασης και περιγραφή λειτουργιών (**θεωρία**: ιδέα και ψευδοκώδικα, **εργαστήριο**: κώδικα C++).
- **Ανάλυση ΔΔ**: προσδιορισμός απαιτήσεων σε **χώρο** αποθήκευσης (αναπαράσταση) και **χρόνο** για κάθε (βασική) λειτουργία.
- Δομές Δεδομένων: οργάνωση δεδομένων στην **κύρια μνήμη**.
  Δομές Αρχείων: οργάνωση δεδομένων σε **δευτερεύουσα μνήμη**.
- Πολύ μεγάλες ΒΔ (VLDBs), ροές δεδομένων (data streams), ιεραρχική μνήμη **ολοκληρώνουν** και δίνουν ποικιλία στην εικόνα.

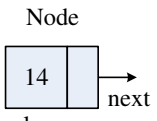
Γραμμική Λίστα σαν ΑΤΔ

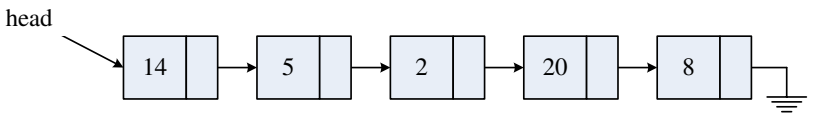
- Αναπαράσταση πεπερασμένης **διατεταγμένης ακολουθίας** αντικειμένων $(e_1, e_2, \ldots, e_n)$, n **μήκος** λίστας.
  Αντικείμενα έχουν **συγκεκριμένες θέσεις** στη λίστα (π.χ. 1ο, 2ο, 3ο, κλπ).
- **Λειτουργίες**:
  - **Αρχικοποίηση**: δημιουργία / καταστροφή.
  - **Αναζήτηση / Προσπέλαση**: αναζήτηση(x) / προσπέλαση(k-οστού).
  - **Τροποποίηση**: εισαγωγή(x, μετά k-οστό) / διαγραφή(x).
  - **Πληροφορίες**: πρώτο / τελευταίο στοιχείο, μήκος, ύπαρξη στοιχείων.

Υλοποίηση: Διασυνδεδεμένη Λίστα

- Αντικείμενο αποθηκεύεται σε **κόμβο** με **πληροφορία** (`elem`) και **δείκτη** σε επόμενο (`next`).
  Μεταβλητή `head` δείχνει **κεφαλή** (πρώτο στοιχείο).

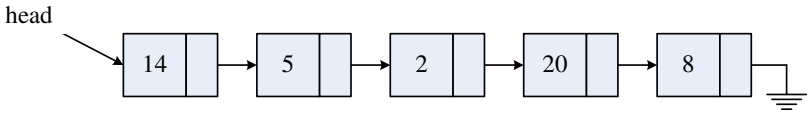
```
typedef struct _node {
    int elem;
    struct _node *next;
} Node;
Node *head = NULL;
int len = 0;
```





- **Χώρος**: $\Theta(n)$ επιπλέον θέσεις για pointers.
  Επιβάρυνση εξαρτάται από **μέγεθος πληροφορίας**.

Διασυνδεδεμένη Λίστα



Boolean isEmpty(head)
  return(head = NULL);

Node *create(k)
  $p \leftarrow$ new(Node);
  $p\rightarrow$elem $\leftarrow k$;
  return(p);

Χρόνος: $\Theta(1)$.

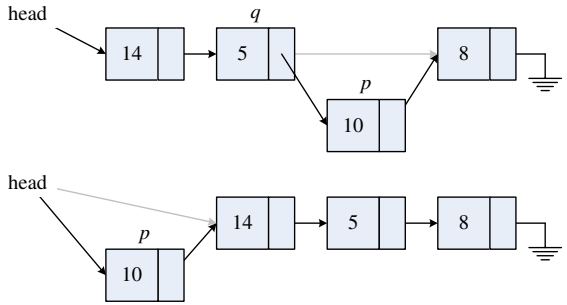
Node *Last(head)
  $p \leftarrow$ head;
  $\ell \leftarrow$ NULL;
  while $p \neq$ NULL **do**
    $\ell \leftarrow p$;
    $p \leftarrow p\rightarrow$next;
  return(ℓ);

Χρόνος: $\Theta(n)$.

Node *Search(head, x)
  $p \leftarrow$ head;
  while $p \neq$ NULL **do**
    if $p\rightarrow$elem $= x$ **then break**;
    else $p \leftarrow p\rightarrow$next;
  return(p);

Χρόνος: $O(n)$.

Εισαγωγή

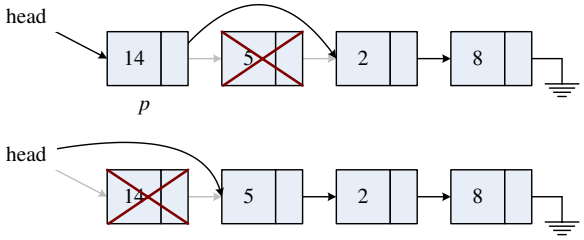


insertAfter(p, q)
  $p\rightarrow$next $\leftarrow q\rightarrow$next;
  $q\rightarrow$next $\leftarrow p$;
  len++;

insertFirst(p)
  $p\rightarrow$next $\leftarrow$ head;
  head $\leftarrow p$;
  len++;

Χρόνος $\Theta(1)$ (αν **γνωρίζουμε θέση** εισαγωγής q).
Χρόνος $O(k)$ για εύρεση k-οστού στοιχείου.

Διαγραφή

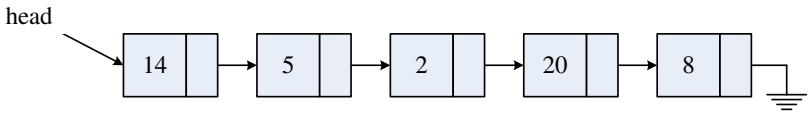


deleteAfter(p)
  if $p\rightarrow$next $\neq$ NULL **then**
    $q \leftarrow p\rightarrow$next;
    $p\rightarrow$next $\leftarrow q\rightarrow$next;
    free(q); len−−;

deleteFirst()
  if head $\neq$ NULL **then**
    $q \leftarrow$ head;
    head $\leftarrow$ head$\rightarrow$next;
    free(q); len−−;

Χρόνος $\Theta(1)$ (αν **γνωρίζουμε θέση** p).
Χρόνος $O(k)$ για διαγραφή $(k+1)$-οστού στοιχείου.

Ασκήσεις



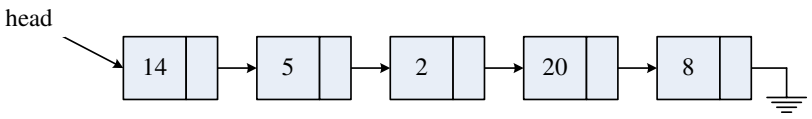
- Εισαγωγή του y **μετά το** x ή εισαγωγή του y **στην** $(k+1)$**-οστή** θέση.

insertAfterElem(head, y, x)
  $p \leftarrow$ create(y);
  $q \leftarrow$ Search(head, x);
  if $q =$ NULL **return**(−1);
  insertAfter(p, q);
  return(1);

Node *findPos(head, k)
  $q \leftarrow$ head; $i \leftarrow 1$;
  while $q \neq$ NULL **and** $i < k$ **do**
    $q \leftarrow q\rightarrow$next; i++;
  return(q);

insertAfterPos(head, y, k)
  $p \leftarrow$ create(y);
  $q \leftarrow$ findPos(head, k);
  if $q =$ NULL **return**(−1);
  insertAfter(p, q);
  return(1);

Ασκήσεις



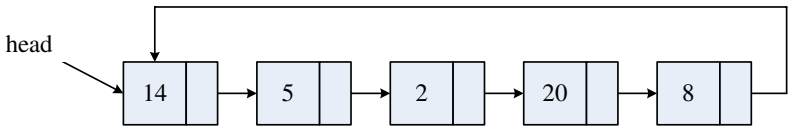
- Διαγραφή του x ή διαγραφή της $(k+1)$-οστής θέσης.
- Συνένωση δύο λιστών με κεφαλές `head1` και `head2`.

Διασυνδεδεμένη Λίστα

- **Δυναμική** δομή δεδομένων που αποθηκεύει **διατεταγμένες** ακολουθίες.
- Υποστηρίζει **προσπέλαση, εισαγωγή, και διαγραφή** σε **γραμμικό** χρόνο.
- **Πιο γρήγορη** για στοιχεία στην αρχή της λίστας.
  Χρόνος $\Theta(k)$ για στοιχείο στην k**-οστή** θέση.
- Υλοποιείται σε **γραμμικό** χώρο.
- **Εύκολη υλοποίηση**.

Κυκλική Λίστα

- Διασυνδεδεμένη λίστα όπου **ουρά δείχνει κεφαλή** αντί για NULL.

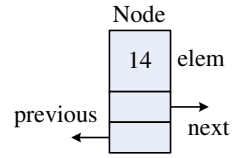


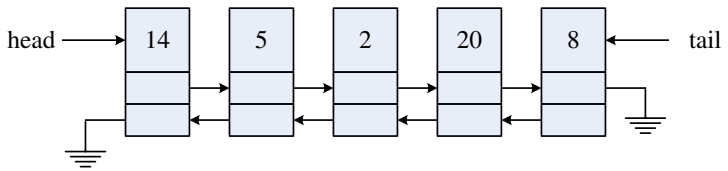
- Κεφαλή μπορεί να είναι **ειδικός κόμβος** χωρίς πληροφορία.
  Κενή λίστα όταν κεφαλή δείχνει τον εαυτό της.
- Πιο **εύκολη υλοποίηση** εισαγωγής και διαγραφής.

Διπλοσυνδεδεμένη Λίστα

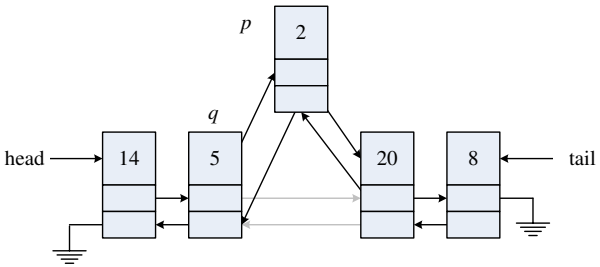
- Αντικείμενο αποθηκεύεται σε **κόμβο** με **πληροφορία** (`elem`)
  και **δείκτες** σε επόμενο και προηγούμενο (`next, previous`).
  Μεταβλητή `head` δείχνει **κεφαλή** (πρώτο στοιχείο).
  Μεταβλητή `tail` δείχνει **ουρά** (τελευταίο στοιχείο).

```
typedef struct _node {
    int elem;
    struct _node *next, *previous;
} Node;
Node head = NULL, tail = NULL;
```

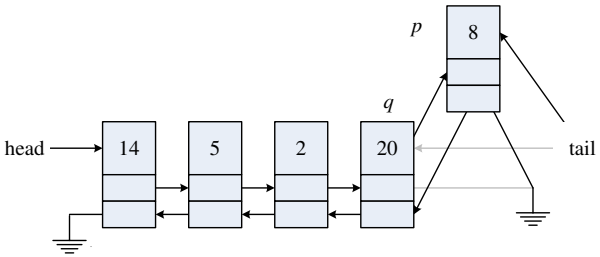




Εισαγωγή



insertAfter(p, q)
p→next ← q→next;
p→previous ← q;
q→next→previous ← p;
q→next ← p;

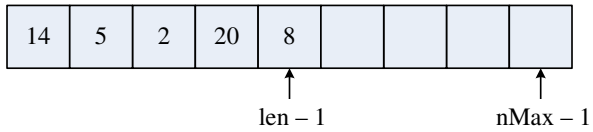


insertLast(p)
p→next ← NULL;
p→previous ← tail;
tail→next ← p;
tail ← p;

Υλοποίηση Γραμμικής Λίστας με Πίνακα

- Αντικείμενα αποθηκεύονται σε **πίνακα** με `nMax` στοιχεία.
  Μεταβλητή `len` έχει #στοιχείων στη λίστα.

```
int A[nMax], len = 0;
```



- **Ευκολότερη** υλοποίηση.
  Συγκρίσιμος χρόνος αναζήτησης: $\Theta(k)$ για στοιχείο σε k-οστή θέση.

int Search(x)
  for $i \leftarrow 0$ **to** len $-$ 1 **do**
    if $A[i] = x$ **then return**(i);
  return(-1);

Υλοποίηση Γραμμικής Λίστας με Πίνακα

- Εισαγωγή θέση k: στοιχεία $k+1, \ldots, n$ μια θέση **δεξιά**.
  Διαγραφή θέση k: στοιχεία $k+1, \ldots, n$ μια θέση **αριστερά**.
  Χρόνος;
- **Στατικός** πίνακας: όχι καλή **αξιοποίηση χώρου**
  (πρόβλεψη για **μέγιστο** αριθμό στοιχείων).
- **Δυναμικός** πίνακας: **δυσκολεύει** υλοποίηση!