Δομές Δεδομένων

Εισαγωγή

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Βιβλιογραφία

- Kurt Mehlhorn and Peter Sanders: Algorithms and Data Structures, The Basic Toolbox, Springer, 2008.
- Robert Sedgewick, Αλγόριθμοι σε C, Μέρη 1-4 (Θεμελιώδεις Έννοιες, Δομές Δεδομένων, Ταξινόμηση, Αναζήτηση), Τρίτη Αμερικάνικη Έκδοση, Εκδόσεις Κλειδάριθμος
- Γεώργιος Φρ. Γεωργακόπουλος, Δομές Δεδομένων: Έννοιες, Τεχνικές, Αλγόριθμοι, Πανεπιστημιακές Εκδόσεις Κρήτης, Ηράκλειο 2002.
- Cormen, Leiserson, Rivest and Stein. “Introduction to Algorithms”, MIT Press, Third Edition, 2009.

Δομές Δεδομένων

Στην επιστήμη υπολογιστών μία **δομή δεδομένων** είναι ένας συγκεκριμένος τρόπος αποθήκευσης και οργάνωσης δεδομένων στον υπολογιστή με σκοπό αυτά τα δεδομένα να μπορούν να χρησιμοποιηθούν αποδοτικά.

Τύποι Δεδομένων

Ένας **τύπος δεδομένων** αποτελείται από:

- ένα σύνολο τιμών
- ένα σύνολο από πράξεις που μπορούν να εκτελεστούν σε αυτές τις τιμές

Τύποι Δεδομένων

Ένας **τύπος δεδομένων** αποτελείται από:

- ένα σύνολο τιμών
- ένα σύνολο από πράξεις που μπορούν να εκτελεστούν σε αυτές τις τιμές

Παράδειγμα **boolean**

- 2 τιμές: true, false
- πράξεις: AND, OR, NOT, κ.τ.λ

Τύποι Δεδομένων

Ένας **τύπος δεδομένων** αποτελείται από:

- ένα σύνολο τιμών
- ένα σύνολο από πράξεις που μπορούν να εκτελεστούν σε αυτές τις τιμές

Παράδειγμα **ακέραιος**

- τιμές $\mathbb{Z}$
- πράξεις: πρόσθεση, αφαίρεση, κ.τ.λ

Στοιχειώδης Δομές Δεδομένων

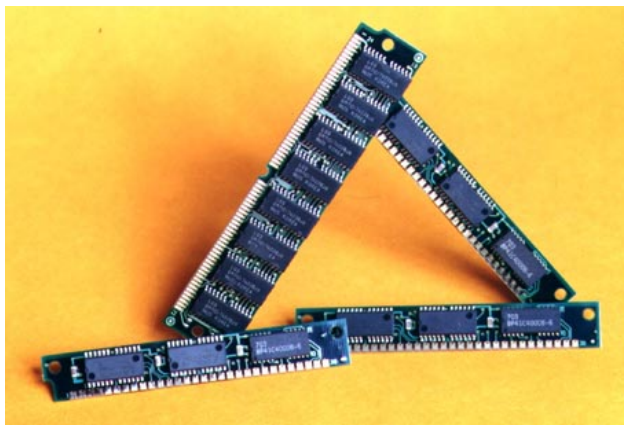
Θα μιλήσουμε για τις παρακάτω στοιχειώδης δομές:

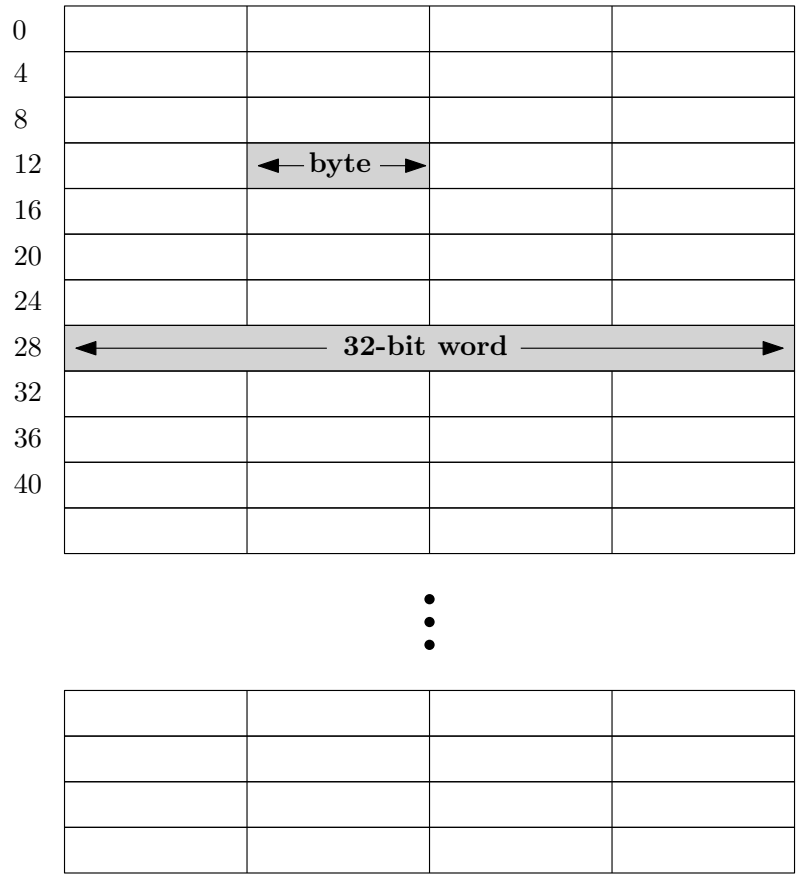
- Πίνακες
- Λίστες

Οι δομές αυτές είναι βασικοί λίθοι για άλλες πιο πολύπλοκες δομές δεδομένων.

Πίνακας (array)

Η πιο θεμελιώδης δομή δεδομένων





Πίνακας (array)

Η πιο θεμελιώδης δομή δεδομένων

Ένας πίνακας αποτελεί μία σταθερή συλλογή δεδομένων ίδιου τύπου τα οποία αποθηκεύονται ακολουθιακά σε κάποιο σημείο της μνήμης.

Πίνακας (array)

Η πιο θεμελιώδης δομή δεδομένων

Ένας πίνακας αποτελεί μία σταθερή συλλογή δεδομένων ίδιου τύπου τα οποία αποθηκεύονται ακολουθιακά σε κάποιο σημείο της μνήμης.

Χαρακτηριστικά

- Σταθερό μέγεθος
- Ένας τύπος δεδομένων
- Συνεχόμενη αποθήκευση στην μνήμη
- Τυχαία πρόσβαση (RAM - random access memory)

Πίνακας (array)

Η πιο θεμελιώδης δομή δεδομένων

Μία δήλωση σε γλώσσα C όπως η παρακάτω:

```
int array[100];
```

δεσμεύει ένα κομμάτι μνήμης 100 ακεραίων.

Η δήλωση περιέχει όλη την απαραίτητη πληροφορία ώστε ο μεταγλωττιστής να μπορεί να υλοποιήσει εντολές όπως

```
array[23] = 5;
```

με την χρήση τυχαίας πρόσβασης.

Πίνακας (array)

Η πιο θεμελιώδης δομή δεδομένων

- Στην C το όνομα ενός πίνακα είνα συνώνυμο με την διεύθυνση μνήμης του πρώτου στοιχείου του πίνακα. Έτσι και αλλιώς ο μεταγλωττιστής αποφασίζει που να αποθηκεύσει τον πίνακα.
- Από τον τύπο του πίνακα (`int`) ξέρει πως κάθε στοιχείο του πίνακα αποτελείται από 4 bytes (σε 32-bit αρχιτεκτονική).
- Αφού τα στοιχεία είναι συνεχόμενα στην μνήμη, το στοιχείο 23 είναι στην διεύθυνση μνήμης

$$\mathit{array} + 23 \times 4$$

Πίνακας

Ποια είναι τα πλεονεκτήματα και μειονεκτήματα των πινάκων

Για να απαντήσουμε αυτό το ερώτημα θα κάνουμε μία μικρή παρένθεση.

Ανάλυση Αλγορίθμων

- διαδικασία κατανόησης της απόδοσης και συμπεριφοράς
- σημαντική στη σχεδίαση και υλοποίηση

Διαφορετικοί τρόποι ανάλυσης

- Εμπειρική ανάλυση (empirical analysis)
  Προσπάθεια κατανόησης συμπεριφοράς μέσω μίας υλοποίησης

- Μαθηματική ανάλυση
  Προσπάθεια κατανόησης συμπεριφοράς μέσω των μαθηματικών

Εμπειρική ανάλυση

- Χρειαζόμαστε μία σωστή και ολοκληρωμένη υλοποίηση
- Πρέπει να αποφασίσουμε τι δεδομένα θα χρησιμοποιήσουμε
  - πραγματικά
  - τυχαία
- Χρειαζόμαστε ένα μέτρο σύγκρισης

Ασυμπτωτική Ανάλυση

Μετράμε την απόδοση ενός αλγορίθμου ή δομής δεδομένων ως συνάρτηση του μεγέθους της εισόδου $n \mapsto f(n)$.

Συνήθως μετράμε ως μονάδα κόστους μία σημαντικού κόστους εντολή του υπολογιστή, π.χ:

- μαθηματικές πράξεις
- συγκρίσεις
- load-store από την μνήμη

Παράδειγμα

Έστω ένας πίνακας με n ακεραίους. Πόσο χρόνο χρειάζεται ο παρακάτω αλγόριθμος για να βρει το μέγιστο.

```
1  int findmax( int *array, int n )
2  {
3      int max = array[0];
4      for( int i = 1; i < n; ++i )
5      {
6          if ( array[i] > max )
7              max = array[i];
8      }
9      return max;
10 }
```

Ασυμπτωτική Ανάλυση

Γενική Ιδέα

- δεν μας ενδιαφέρουν οι σταθερές
- εάν ένας αλγόριθμος κάνει 2 προσθέσεις και μία σύγκριση για n στοιχεία, εμείς δεν μετράμε $3n$ κόστος αλλά n
- προσπαθούμε να κάνουμε την ανάλυση ανεξάρτητη από την ταχύτητα του υπολογιστή που τρέχει ο αλγόριθμος

Ασυμπτωτική Ανάλυση

Γενική Ιδέα

- δεν μας ενδιαφέρουν οι σταθερές
- εάν ένας αλγόριθμος κάνει 2 προσθέσεις και μία σύγκριση για n στοιχεία, εμείς δεν μετράμε $3n$ κόστος αλλά n
- προσπαθούμε να κάνουμε την ανάλυση ανεξάρτητη από την ταχύτητα του υπολογιστή που τρέχει ο αλγόριθμος

Προσπαθούμε να καταλάβουμε πως συμπεριφέρεται ένας αλγόριθμος όταν $n \to \infty$.

Εάν ένας αλγόριθμος έχει κόστος $4n^4 + 2n^2 + 30$ εμείς αγνοούμε τους μικρούς όρους και τις σταθερές και λέμε πως ο αλγόριθμος συμπεριφέρεται $\approx n^4$ για πολύ μεγάλα n.

- Έστω ένας αλγόριθμος Α που για είσοδο μεγέθους n τρέχει σε χρόνο $3n^2 + 2n + 5$

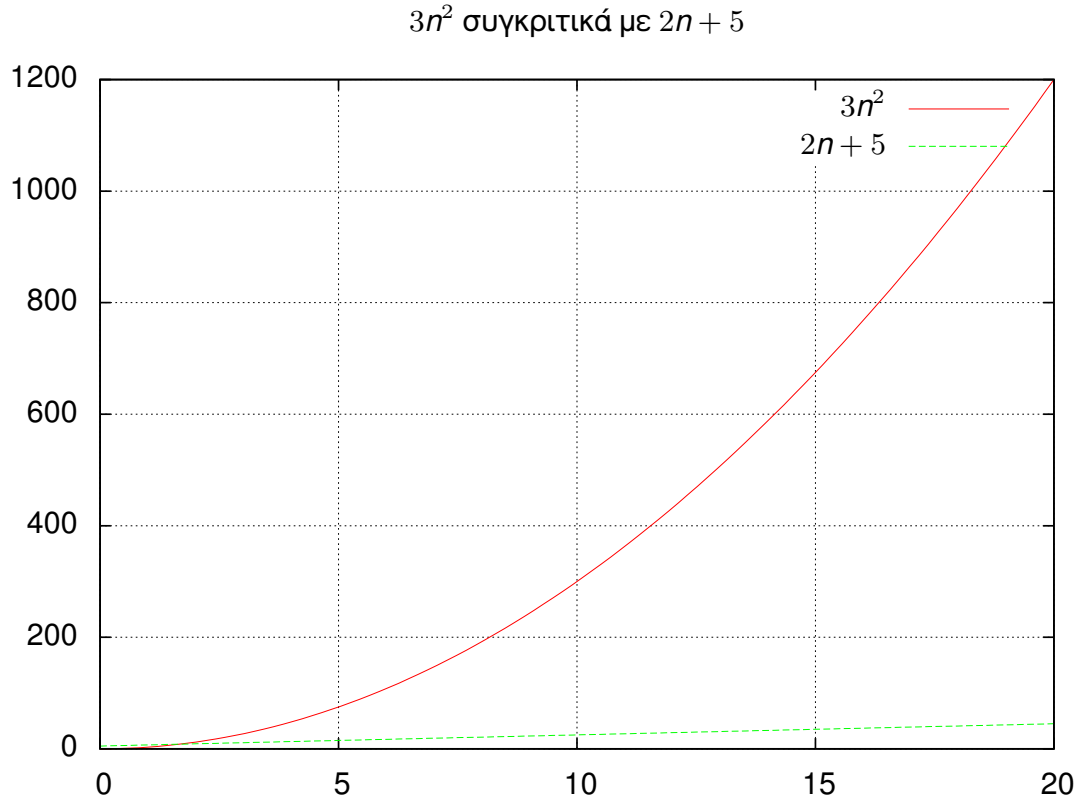
- Έστω ένας αλγόριθμος Α που για είσοδο μεγέθους n τρέχει σε χρόνο $3n^2 + 2n + 5$
- Ισχυριζόμαστε πως για αρκετά μεγάλες εισόδους οι χαμηλές δυνάμεις δεν έχουν μεγάλη σημασία

- Έστω ένας αλγόριθμος Α που για είσοδο μεγέθους n τρέχει σε χρόνο $3n^2 + 2n + 5$
- Ισχυριζόμαστε πως για αρκετά μεγάλες εισόδους οι χαμηλές δυνάμεις δεν έχουν μεγάλη σημασία

n	$3n^2$	$2n + 5$
2	12	9
10	300	25
1000	3000000	2005
100000	$3 \cdot 10^{10}$	200005

Ασυμπτωτική Ανάλυση

Παράδειγμα



Ο τετραγωνικός όρος αυξάνει πολύ πιο γρήγορα από τον γραμμικό όρο.

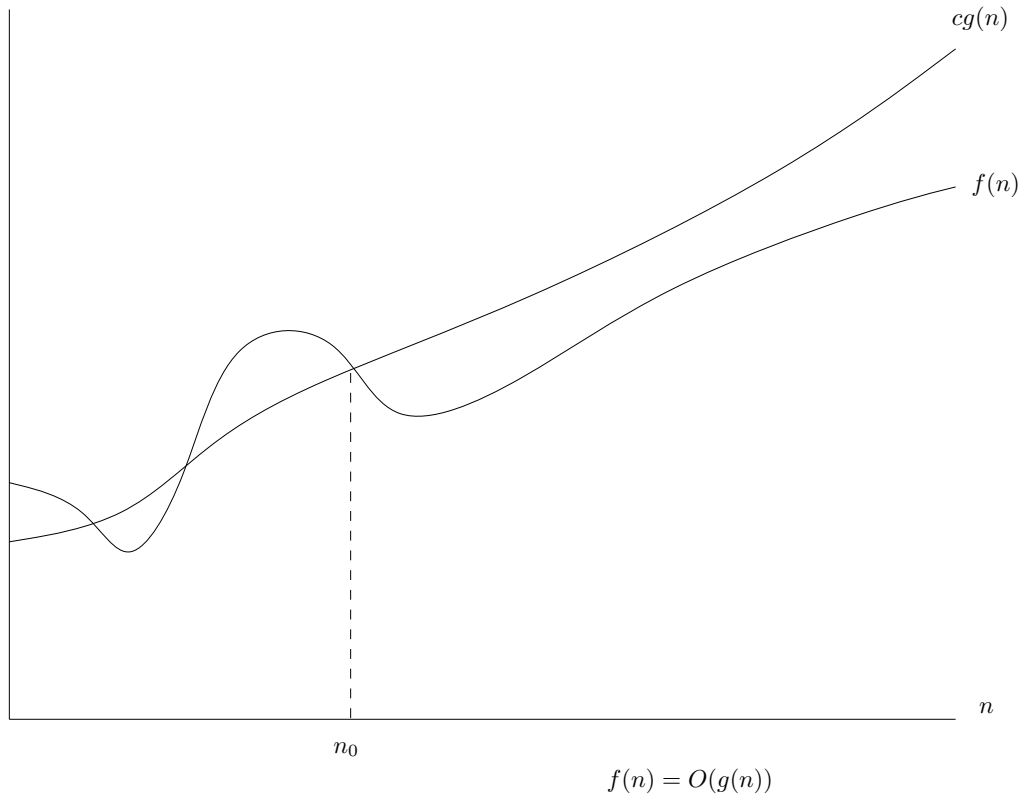
Αγνοώντας τους μικρούς όρους χάνουμε πολύ λίγο για μεγάλες εισόδους

Συμβολισμός μεγάλου όμικρον
big-Oh notation

$$\mathcal{O}(g(n)) = \left\{ f(n) : \begin{array}{l} \text{υπάρχουν θετικές σταθερές } c \text{ και } n_0 \text{ ώστε} \\ 0 \leq f(n) \leq c \cdot g(n) \text{ για όλα τα } n \geq n_0 \end{array} \right\}$$

Συμβολισμός μεγάλου όμικρον
big-Oh notation

$$\mathcal{O}(g(n)) = \left\{ f(n) : \begin{array}{l} \text{υπάρχουν θετικές σταθερές } c \text{ και } n_0 \text{ ώστε} \\ 0 \leq f(n) \leq c \cdot g(n) \text{ για όλα τα } n \geq n_0 \end{array} \right\}$$



Θεώρημα

$2n^3 + 100n^2 + n \in \mathcal{O}(n^3)$

Απόδειξη

Αρκεί να βρούμε θετική σταθερά c και $n_0 \geq 0$ ώστε να ισχύει:

$$2n^3 + 100n^2 + n \leq cn^3$$

Θεώρημα

$2n^3 + 100n^2 + n \in \mathcal{O}(n^3)$

Απόδειξη

Αρκεί να βρούμε θετική σταθερά c και $n_0 \geq 0$ ώστε να ισχύει:

$$2n^3 + 100n^2 + n \leq cn^3$$

Διαιρώντας με $n > 1$ έχουμε πως αρκεί

$$2n^2 + 100n + 1 \leq cn^2.$$

Θεώρημα

$2n^3 + 100n^2 + n \in \mathcal{O}(n^3)$

Απόδειξη

Αρκεί να βρούμε θετική σταθερά c και $n_0 \geq 0$ ώστε να ισχύει:

$$2n^3 + 100n^2 + n \leq cn^3$$

Διαιρώντας με $n > 1$ έχουμε πως αρκεί

$$2n^2 + 100n + 1 \leq cn^2.$$

Για $n \geq 1$ έχουμε $2n^2 + 100n + 1 \leq 2n^2 + 100n^2 + n^2 \leq 103n^2$. Άρα μπορούμε να θέσουμε $c = 103$ και $n_0 = 1 > 0$. □

Θεώρημα

$n \in \mathcal{O}(2^n)$

Απόδειξη

Πρέπει να βρούμε $c > 0, n_0 > 0$ ώστε να ισχύει

$$n \leq c \cdot 2^n.$$

Θεώρημα

$n \in \mathcal{O}(2^n)$

Απόδειξη

Πρέπει να βρούμε $c > 0, n_0 > 0$ ώστε να ισχύει

$$n \leq c \cdot 2^n.$$

Παρατηρώντας τον τρόπο που μεγαλώνουν οι συναρτήσεις βλέπουμε πως για $c = 1$ και $n_0 = 1$ η ανισότητα ισχύει. Πρέπει όμως να αποδείξουμε πως παραμένει αλήθεια για κάθε $n \geq n_0 = 1$.

Θεώρημα

$n \in \mathcal{O}(2^n)$

Απόδειξη

Πρέπει να βρούμε $c > 0, n_0 > 0$ ώστε να ισχύει

$$n \leq c \cdot 2^n.$$

Παρατηρώντας τον τρόπο που μεγαλώνουν οι συναρτήσεις βλέπουμε πως για $c = 1$ και $n_0 = 1$ η ανισότητα ισχύει. Πρέπει όμως να αποδείξουμε πως παραμένει αλήθεια για κάθε $n \geq n_0 = 1$. Έστω $f(n) = 2^n - n$. Η συνάρτηση είναι συνεχής και παραγωγίσιμη. Η πρώτη παράγωγος είναι $f'(n) = ln(2) \cdot 2^n - 1$ και είναι θετική για κάθε $n \geq 1$.

Θεώρημα

$n \in \mathcal{O}(2^n)$

Απόδειξη

Πρέπει να βρούμε $c > 0, n_0 > 0$ ώστε να ισχύει

$$n \leq c \cdot 2^n.$$

Παρατηρώντας τον τρόπο που μεγαλώνουν οι συναρτήσεις βλέπουμε πως για $c = 1$ και $n_0 = 1$ η ανισότητα ισχύει. Πρέπει όμως να αποδείξουμε πως παραμένει αλήθεια για κάθε $n \geq n_0 = 1$. Έστω $f(n) = 2^n - n$. Η συνάρτηση είναι συνεχής και παραγωγίσιμη. Η πρώτη παράγωγος είναι $f'(n) = ln(2) \cdot 2^n - 1$ και είναι θετική για κάθε $n \geq 1$.
Άρα η συνάρτηση $f(n)$ είναι γνησίως αύξουσα για $n \geq 1$ που σημαίνει πως παραμένει θετική για κάθε $n \geq n_0 = 1$. □

Θεώρημα

$1000000 \in \mathcal{O}(1)$

Απόδειξη

Πρέπει να δείξουμε πως υπάρχουν $c > 0, n_0 > 0$ ώστε

$$1000000 \leq c \cdot 1$$

για κάθε $n \geq n_0$.
Η σχέση ισχύει για $c = 1000001$ και $n_0 = 1$. □

Θεώρημα

Για $a > 0$ ισχύει πως $f(n) = an^2 + bn \in \mathcal{O}(n^2)$.

Απόδειξη

Πρέπει να βρούμε θετική σταθερά c και $n_0 \geq 0$ ώστε

$$an^2 + bn \leq cn^2 \text{ για κάθε } n \geq n_0$$

ή ισοδύναμα

$$an + b \leq cn.$$

Επειδή $an + b \leq |a|n + |b| \leq |a|n + |b|n$ για $n \geq 1$ μπορούμε να διαλέξουμε $c = |a| + |b|$ και n_0 οποιαδήποτε θετική τιμή μεγαλύτερη ή ίση του ένα π.χ $n_0 = 1$.

□

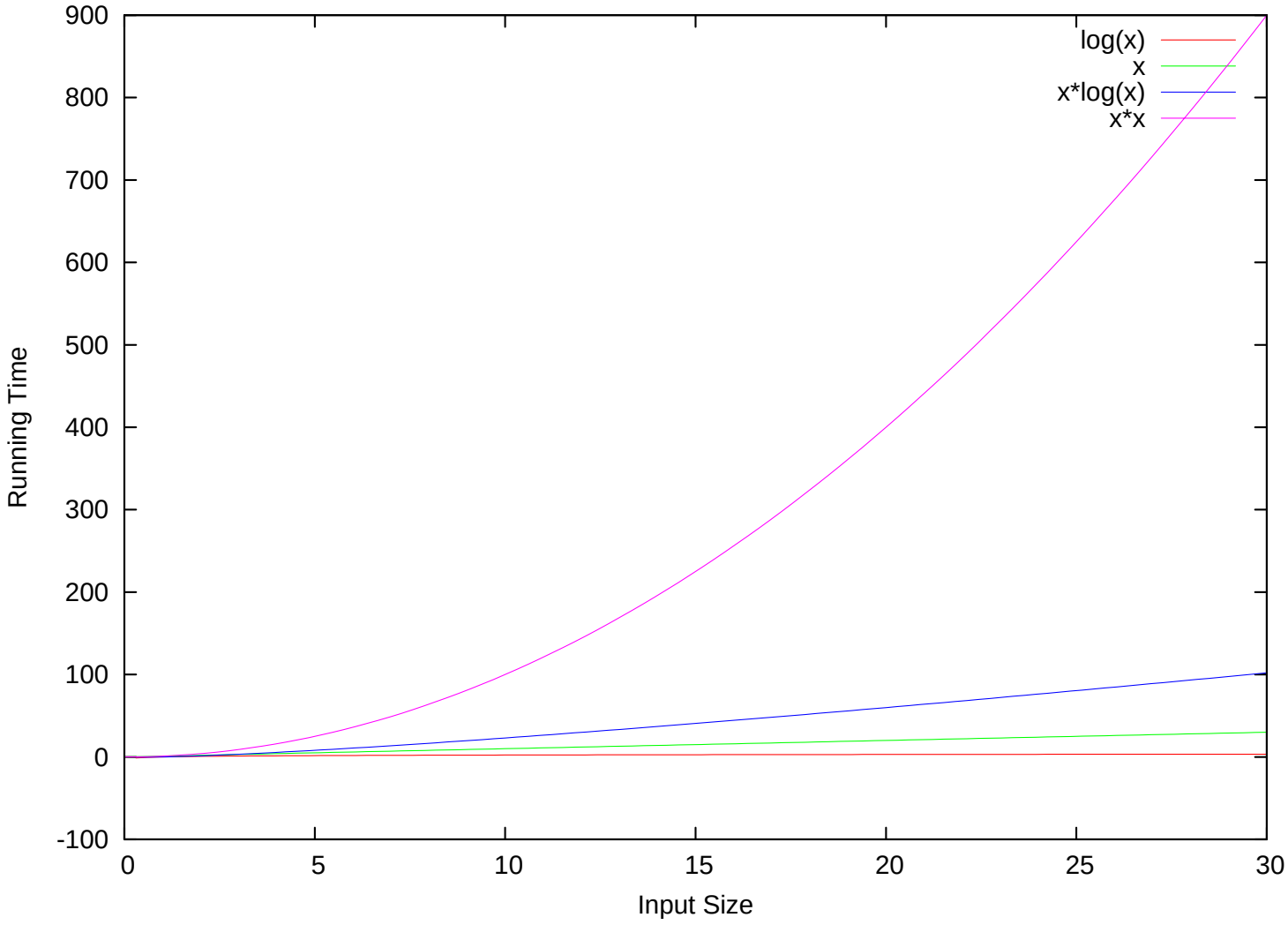
Συμβολισμός μεγάλου όμικρον
big-Oh notation

Συναντάμε συχνά τους εξής χρόνους:

- σταθερός $\mathcal{O}(1)$
- λογαριθμικός $\mathcal{O}(\log n)$
- γραμμικός $\mathcal{O}(n)$
- $\mathcal{O}(n \log n)$
- τετραγωνικός $\mathcal{O}(n^2)$
- πολυωνυμικός $\mathcal{O}(n^k)$
- εκθετικός $\mathcal{O}(2^n)$

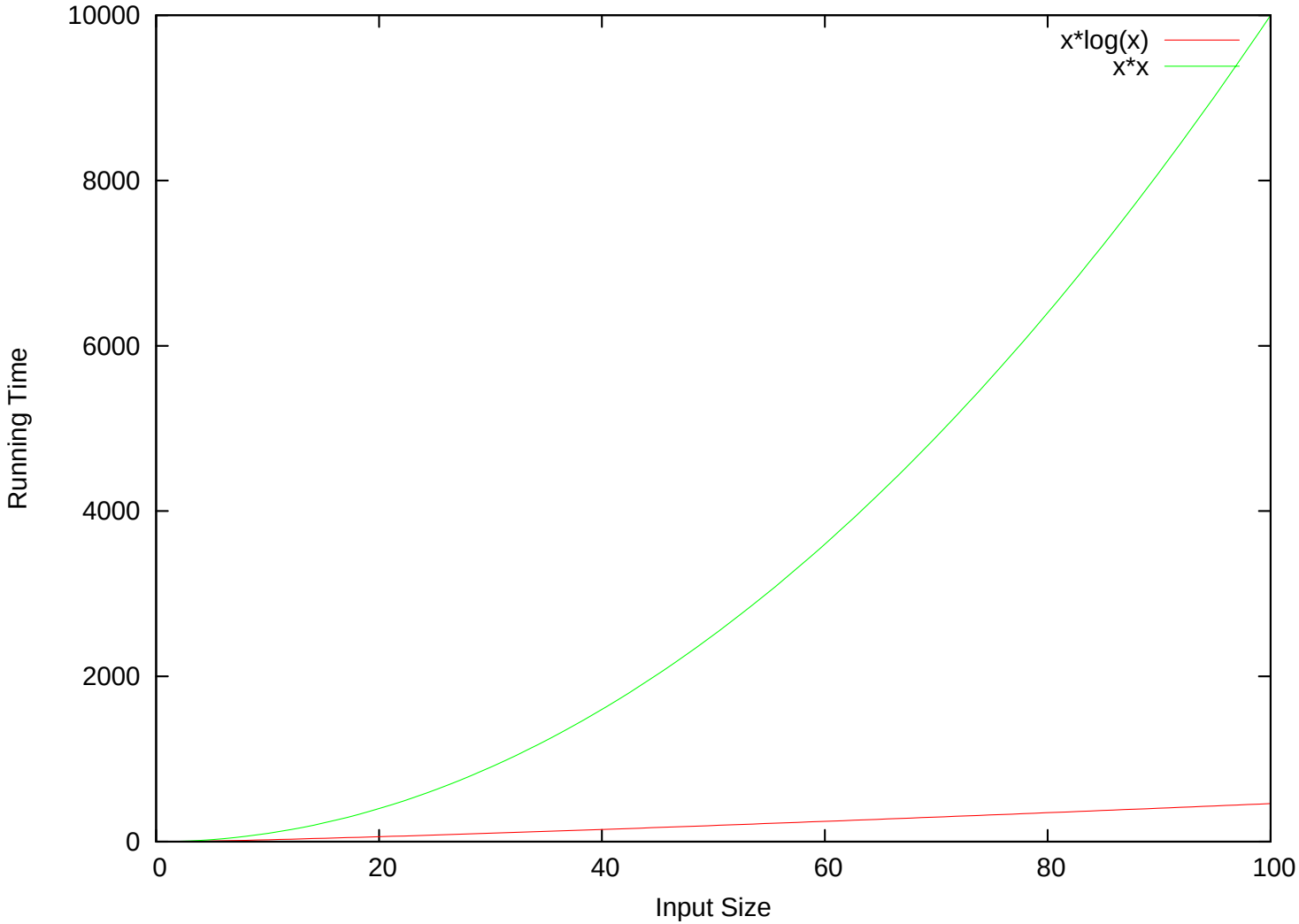
Συμβολισμός μεγάλου όμικρον

big-Oh notation



Συμβολισμός μεγάλου όμικρον

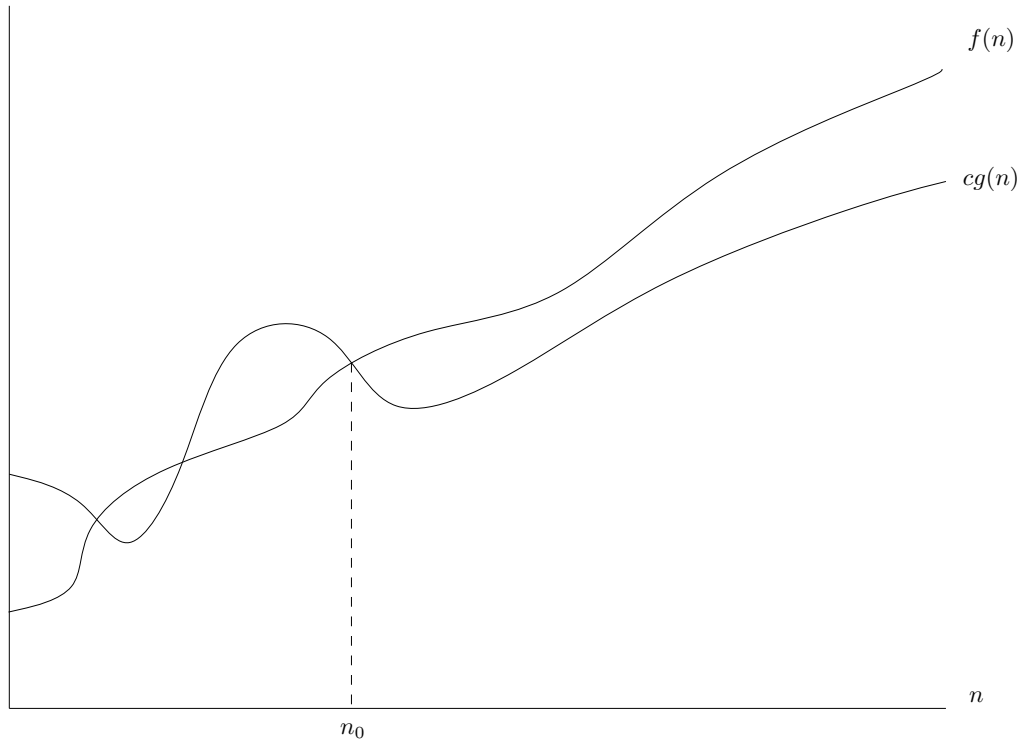
big-Oh notation



Συμβολισμός Ομέγα

Ω-notation

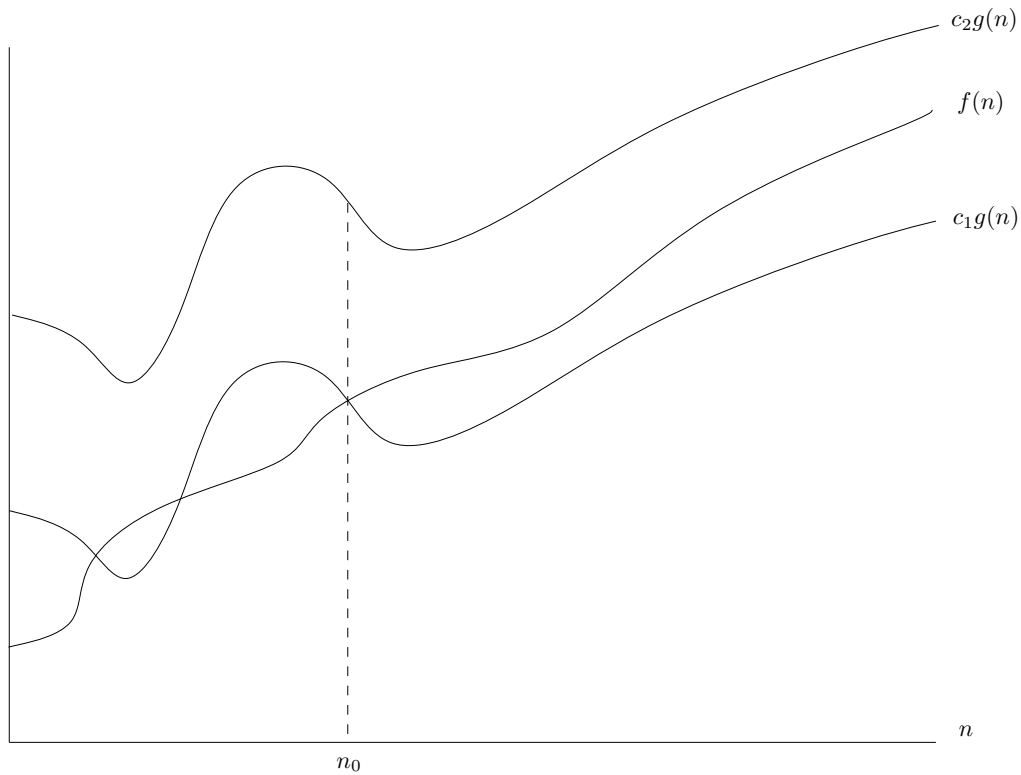
$$\Omega(g(n)) = \left\{ f(n) : \begin{array}{l} \text{υπάρχουν θετικές σταθερές } c \text{ και } n_0 \text{ ώστε} \\ 0 \leq c \cdot g(n) \leq f(n) \text{ για όλα τα } n \geq n_0 \end{array} \right\}$$



Συμβολισμός Θήτα

Θ-notation

$$\Theta(g(n)) = \left\{ f(n) : f(n) = \mathcal{O}(g(n)) \text{ και } f(n) = \Omega(g(n)) \right\}$$



$f(n) = \Theta(g(n))$

- Εμπειρική ανάλυση (empirical analysis)
  Προσπάθεια κατανόησης συμπεριφοράς μέσω μίας υλοποίησης

- Μαθηματική ανάλυση
  Προσπάθεια κατανόησης συμπεριφοράς μέσω των μαθηματικών

Καμία μέθοδος δεν είναι πανάκεια

Πλεονεκτήματα

- προσπέλαση στοιχείου σε σταθερό χρόνο, $\mathcal{O}(1)$
- τυχαία προσπέλαση

Μειονεκτήματα

- σταθερό μέγεθος
- δεν μπορούμε να προσθέσουμε στοιχεία στη μέση ενός πίνακα
- αναδιάταξη στοιχείων μόνο με αντιγραφή

Διδιάστατοι Πίνακες

- πίνακες με 2 αριθμοδείκτες, πχ *array*$[3][2]$
- αντιστοιχούν σε μήτρες (matrices)
- υλοποιούνται με μονοδιάστατους πίνακες, πχ για 5×5 μεγέθους πίνακα έχουμε *array*$[i][j] =$ *array*$[5 * i + j]$
- άρα κάθε στοιχείο χρειάζεται $\mathcal{O}(1)$ χρόνο για να προσπελαστεί

Διδιάστατοι Πίνακες

Πολλαπλασιασμός Πινάκων

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mp} \end{pmatrix}$$

```
for( i = 0; i < n; i++ )
   for( j = 0; j < p; j++ )
      for( k = 0, c[i][j] = 0.0; k < m; k++ )
         c[i][j] += a[i][k] * b[k][j];
```

Διδιάστατοι Πίνακες

Πολλαπλασιασμός Πινάκων

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mp} \end{pmatrix}$$

```
for( i = 0; i < n; i++ )
   for( j = 0; j < p; j++ )
      for( k = 0, c[i][j] = 0.0; k < m; k++ )
         c[i][j] += a[i][k] * b[k][j];
```

Η πολυπλοκότητα είναι $\mathcal{O}(npm)$

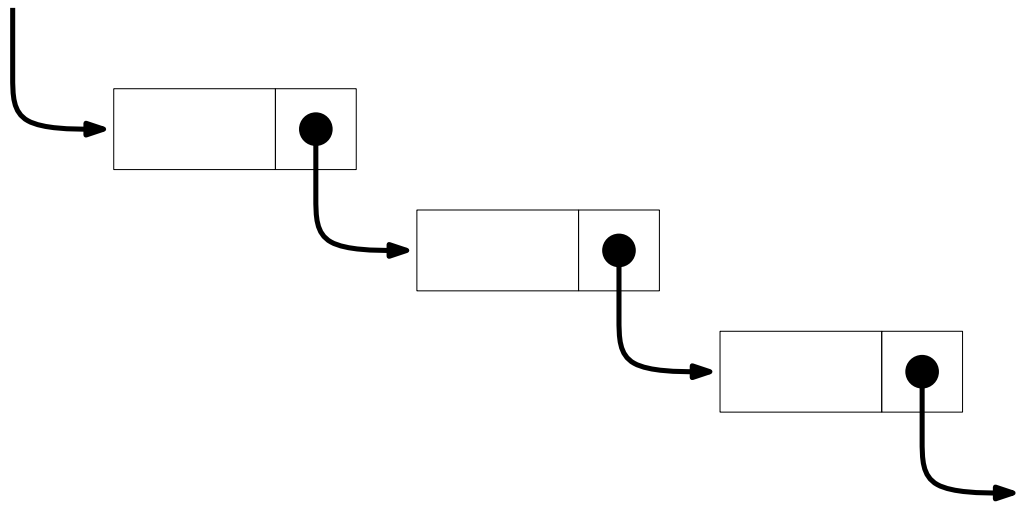
Συνδεδεμένες Λίστες
Linked Lists

Στόχος

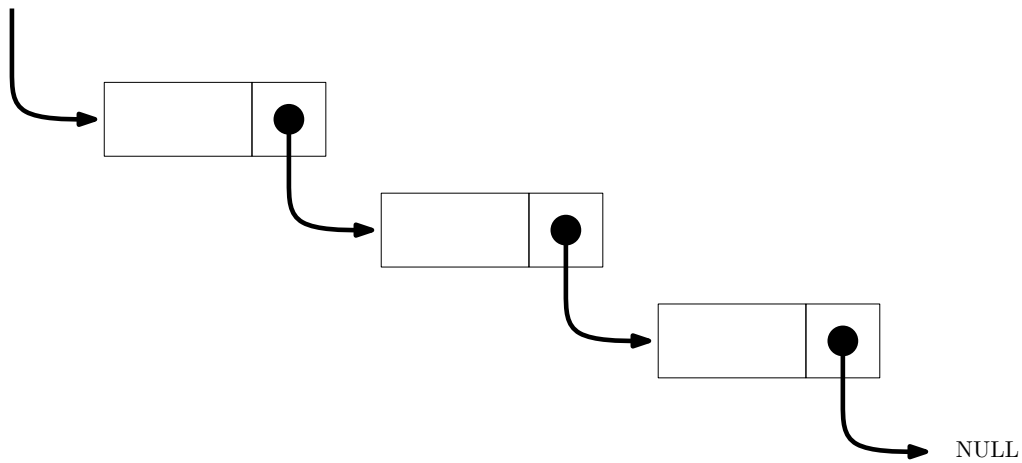
- πρόσβαση σε μία συλλογή στοιχείων με τη σειρά (ακολουθιακά)
- αποδοτική αναδιάταξη στοιχείων, διαγραφή, προσθήκη, κ.τ.λ

Ορισμός

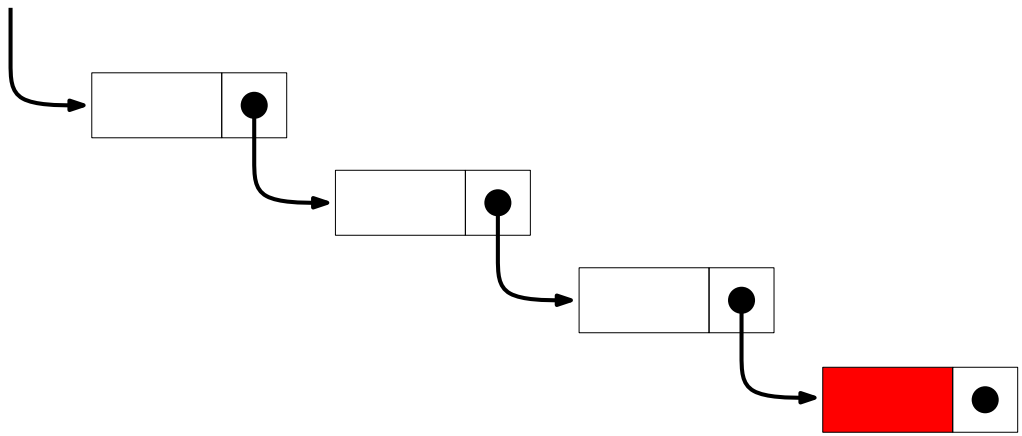
Μία **συνδεδεμένη λίστα** είναι ένα σύνολο στοιχείων όπου κάθε στοιχείο αποτελεί τμήμα ενός **κόμβου** (node) ο οποίος περιέχει επίσης ένα **σύνδεσμο** (link) προς κάποιον κόμβο.



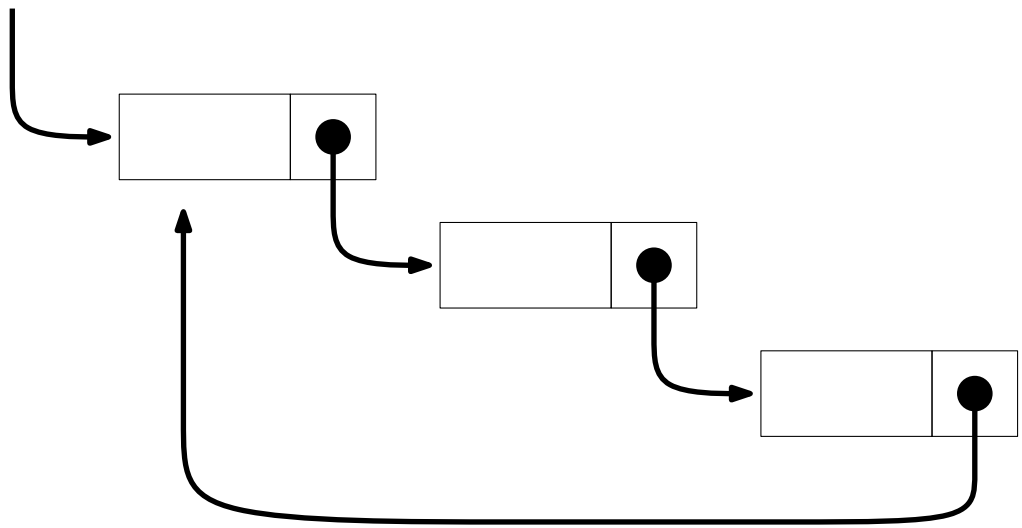
Με **μηδενικό σύνδεσμο** (null link)



Με **ψευδο-κόμβο** (dummy node)

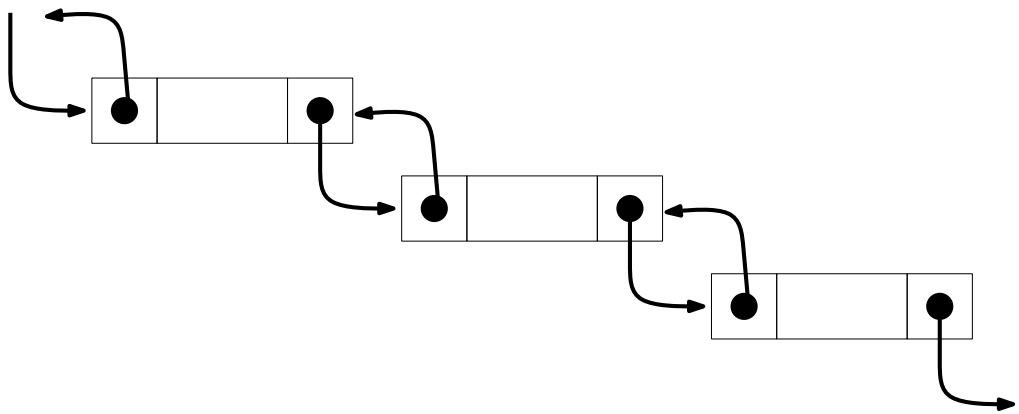


Χωρίς τερματισμό, **κυκλική** λίστα (circular)



Διπλά Συνδεδεμένες Λίστες

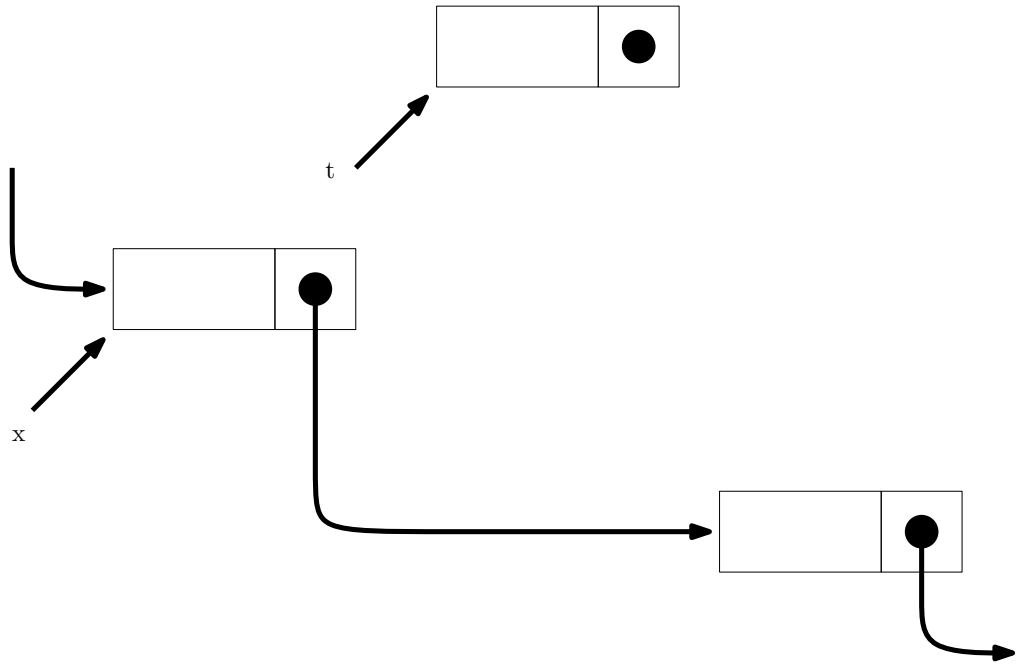
Περιέχουν και σύνδεσμο προς το προηγούμενο στοιχείο της λίστας



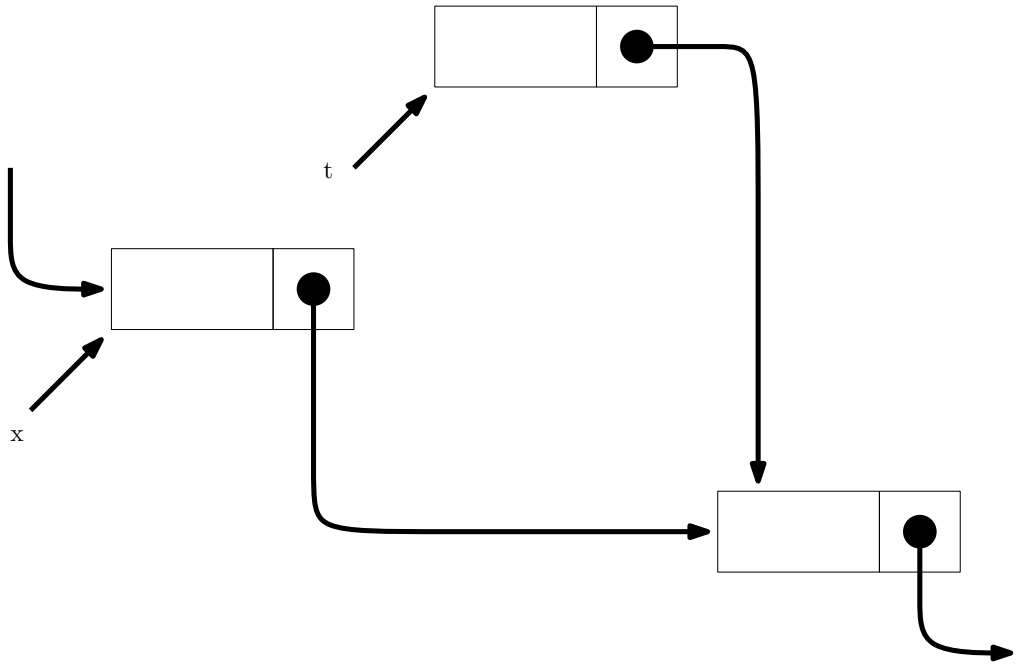
Χαρακτηριστικά Λιστών

- προσθήκη, διαγραφή, μετακίνηση σε $\mathcal{O}(1)$ χρόνο
- αναζήτηση k-στού στοιχείου σε $\mathcal{O}(k)$ χρόνο
- καταναλώνουν περισσότερο χώρο από πίνακες, αφού κρατάμε και τους συνδέσμους

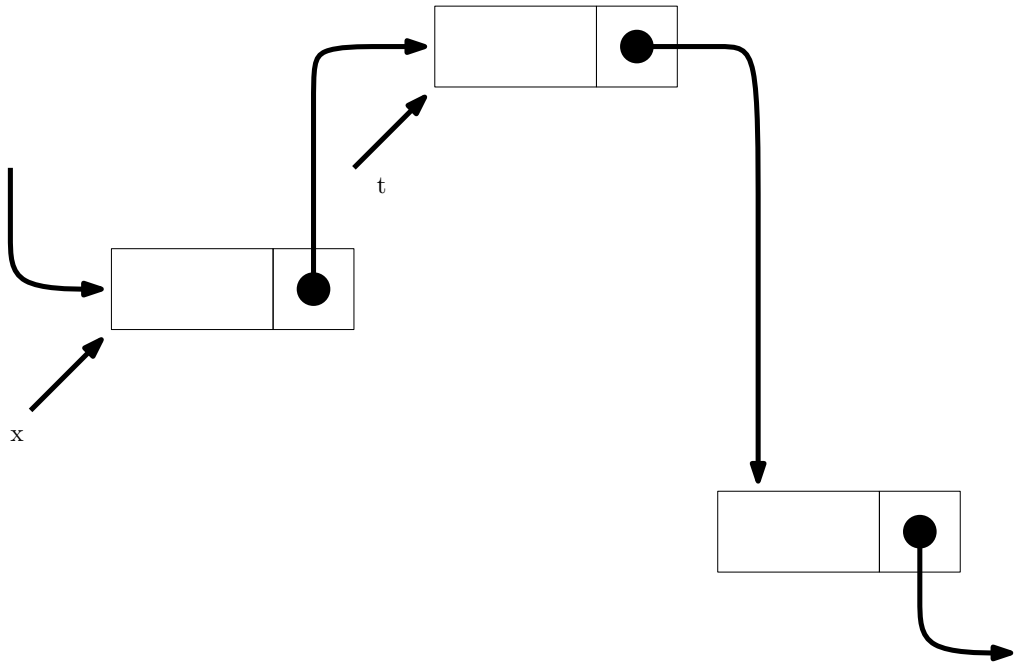
Προσθήκη σε Λίστα



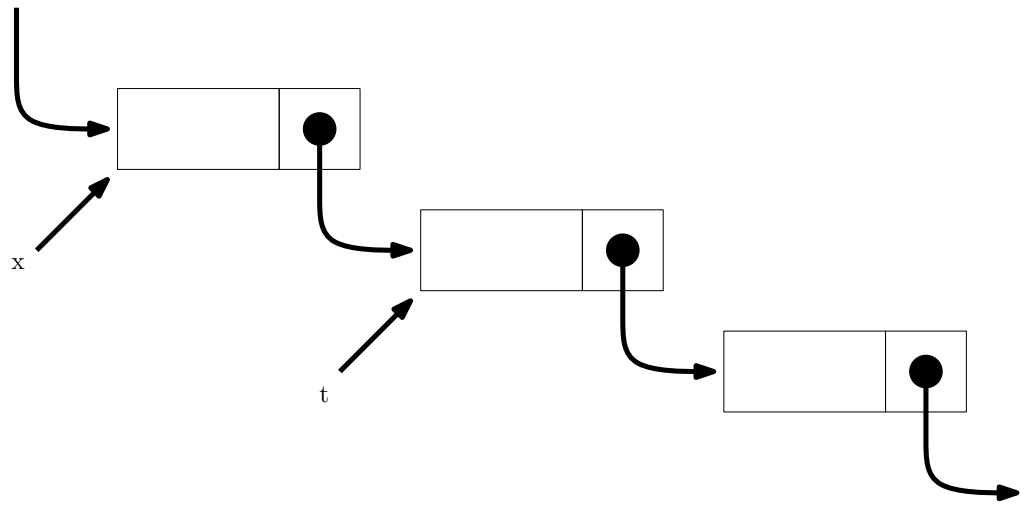
Προσθήκη σε Λίστα



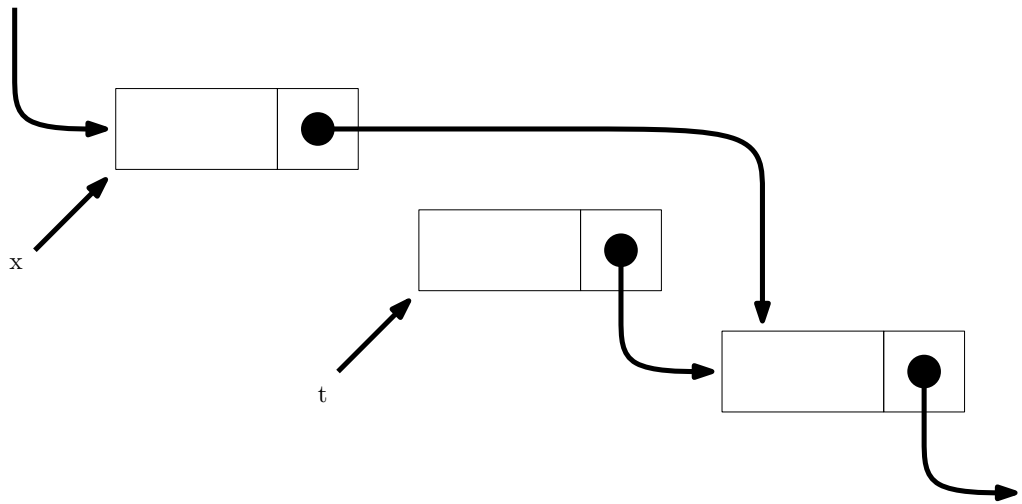
Προσθήκη σε Λίστα



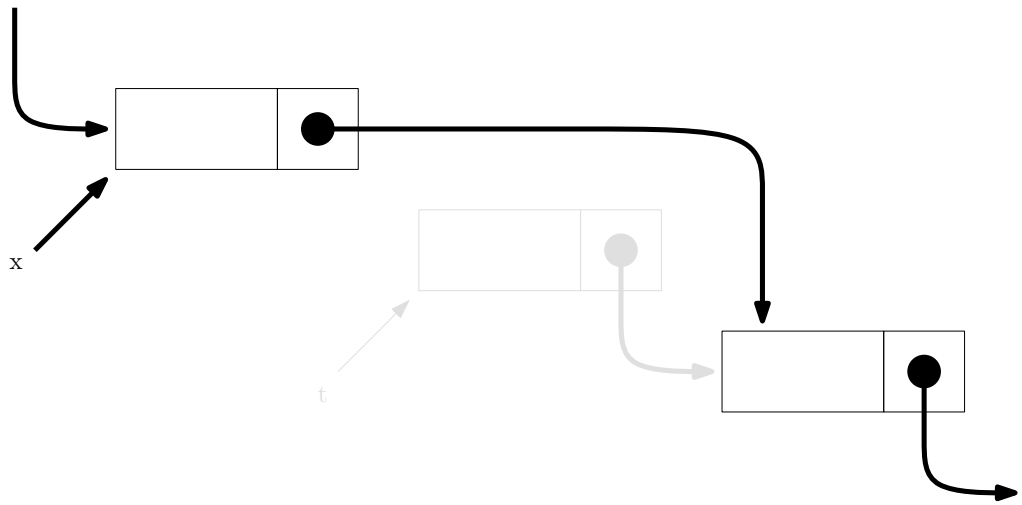
Διαγραφή από Λίστα



Διαγραφή από Λίστα



Διαγραφή από Λίστα



Αφαίρεση (abstraction)

Ανάπτυξη αφηρημένων μοντέλων που διαχωρίζουν την λειτουργία από την υλοποίηση.

Για να αναπτύξουμε ένα πολύπλοκο σύστημα (όπως τα σημερινά υπολογιστικά συστήματα) βασιζόμαστε σε **στρώσεις αφαίρεσης** (layers of abstraction).

Στρώσεις αφαίρεσης

Παράδειγμα

- bits από φυσικές ιδιότητες υλικών
- αφηρημένο μοντέλο μηχανής από τις δυναμικές ιδιότητες των τιμών ενός συνόλου από bits
- αφηρημένο μοντέλο γλώσσας προγραμματισμού
- αφηρημένη έννοια αλγορίθμου

Αφηρημένος Τύπος Δεδομένων (ΑΤΔ)
Abstract Data Type

Ορισμός

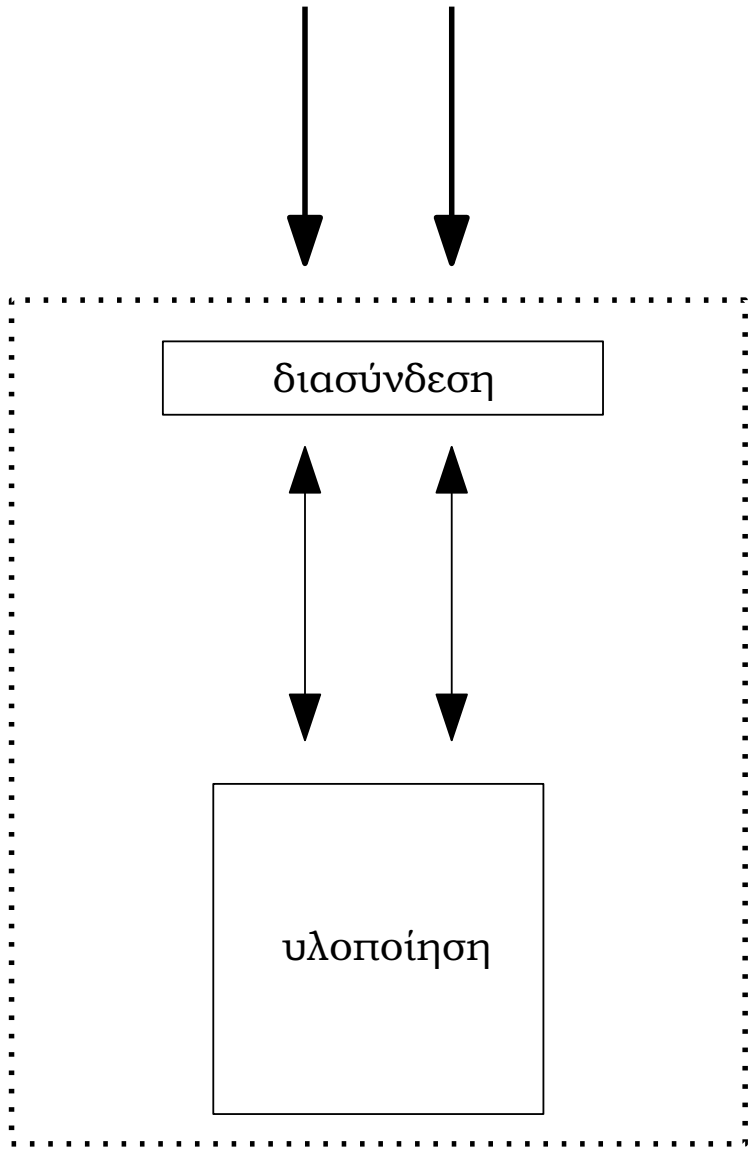
Ένας **αφηρημένος τύπος δεδομένων** είναι ένας τύπος δεδομένων (ένα σύνολο τιμών και μία συλλογή πράξεων ή λειτουργιών που εφαρμόζονται σε αυτές τις τιμές) ο οποίος μπορεί να προσπελαστεί μόνο μέσω κάποιας **διασύνδεσης** (interface).

Ονομάζουμε **πελάτη** (client) ένα πρόγραμμα που χρησιμοποιεί κάποιον αφηρημένο τύπο δεδομένων και **υλοποίηση** (implementation) ένα πρόγραμμα στο οποίο ορίζεται ο τύπος δεδομένων.

Αφηρημένος Τύπος Δεδομένων (ΑΤΔ)

Abstract Data Type

- Χρήση μόνο μέσα από την **διασύνδεση**
- Μπορούμε να αλλάξουμε ελεύθερα την **υλοποίηση**
- Θα δούμε αργότερα παραδείγματα



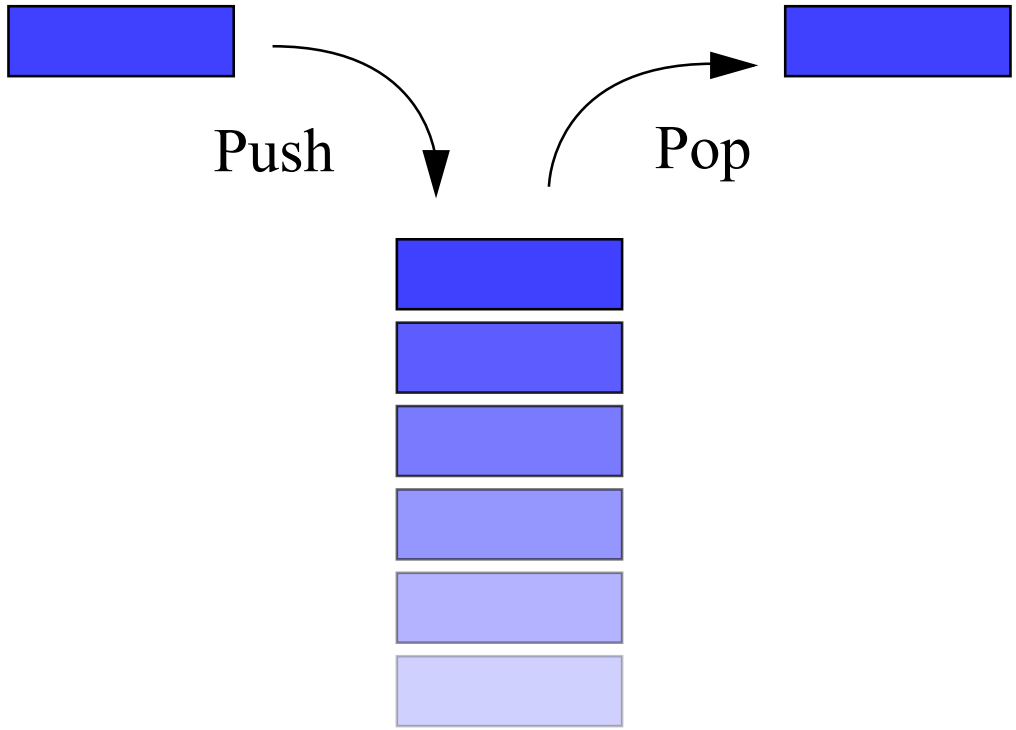
Στοίβα
ως ΑΤΔ

Μία στοίβα ώθησης προς τα κάτω (pushdown stack) είναι ένας ΑΤΔ που περιλαμβάνει τις εξής λειτουργίες:

- Εισαγωγή (ώθηση) ενός νέου στοιχείου, PUSH(S,x).
- Διαγραφή (απώθηση, pop) του τελευταίου στοιχείου που προστέθηκε στη στοίβα, POP(S).
- Έλεγχος κενής στοίβας, EMPTY(S).
- Επιστροφή του μεγέθους της στοίβας, SIZE(S).

Στοίβα

Η στοίβα ακολουθεί την αρχή **τελευταίο μέσα, πρώτο έξω** ή αλλιώς LIFO (last-in, first-out)



Αφού ορίζουμε μία στοίβα ως ΑΤΔ δεν μας ενδιαφέρει η ακριβής υλοποίηση της στοίβας. Μπορούμε για παράδειγμα να υλοποιήσουμε μια στοίβα με λίστα αλλά και με πίνακα.

Παράδειγμα Χρήσης Στοίβας

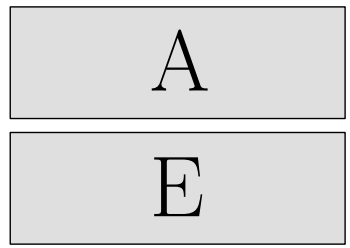
- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()

Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()

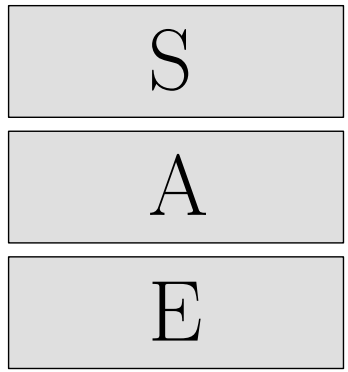
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



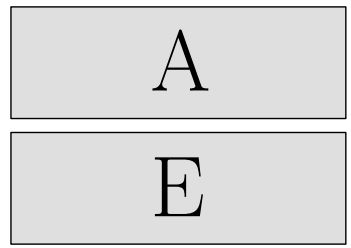
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



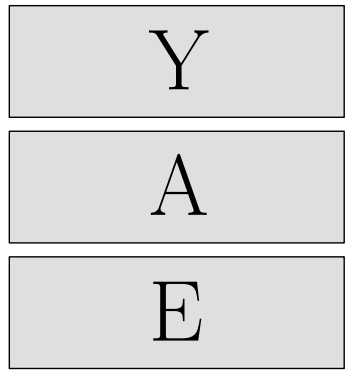
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



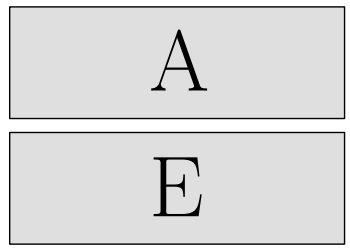
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



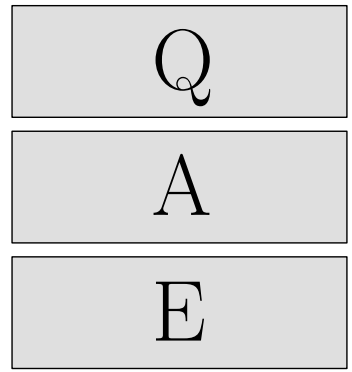
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



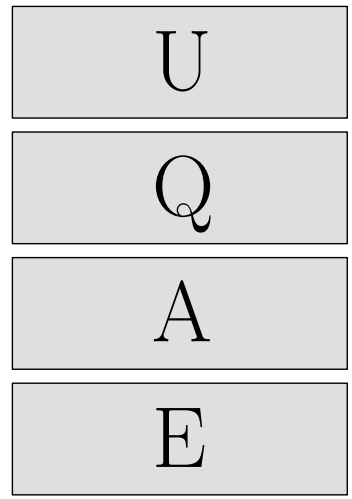
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



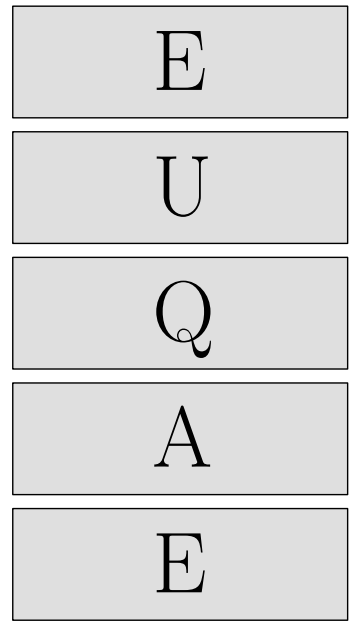
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



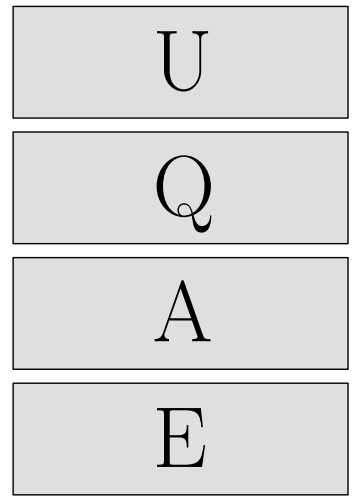
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



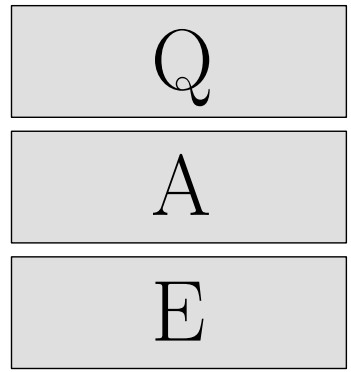
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



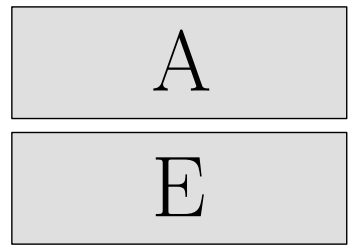
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



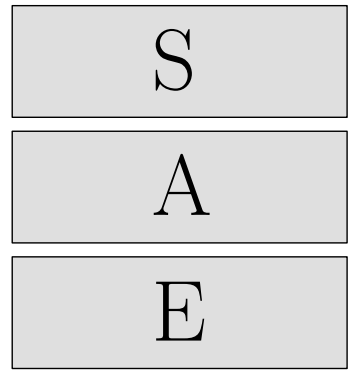
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



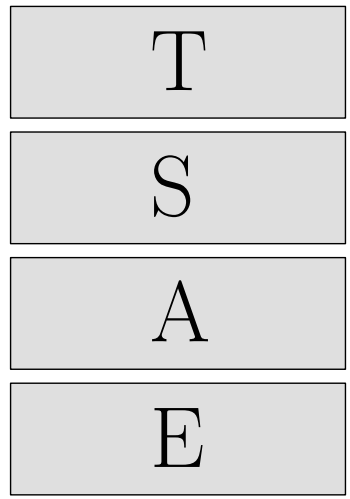
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



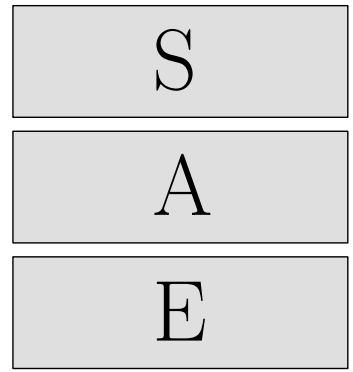
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



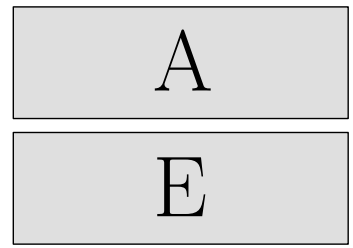
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()

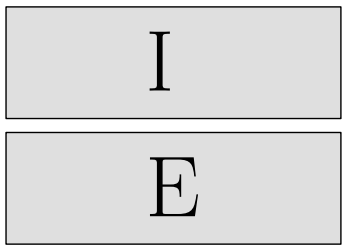


Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()

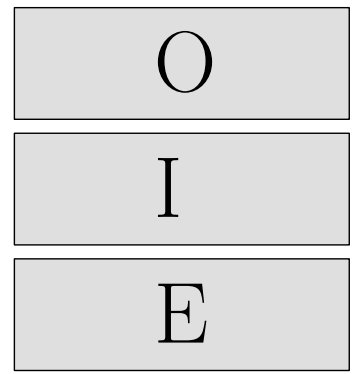
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



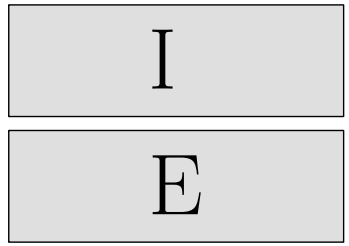
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



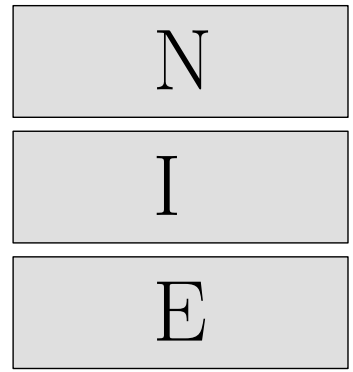
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



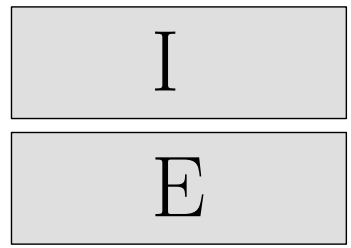
Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()



Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()

Παράδειγμα Χρήσης Στοίβας

- EAS*Y*QUE***ST***IO*N***
- τα γράμματα σημαίνουν *push*() και τα αστεράκια *pop*()

Εφαρμογή Στοίβας

Υπολογισμών εκφράσεων (infix notation)

Μια σημαντική εφαρμογή της στοίβας είναι ο υπολογισμός αριθμητικών παραστάσεων.

π.χ για να υπολογίσουμε την παράσταση

$$5 \times (((9 + 8) \times (4 \times 6)) + 7)$$

πρέπει να έχουμε ένα μηχανισμό αποθήκευσης ενδιάμεσων αποτελεσμάτων.

Μία στοίβα είναι ένας ιδανικός μηχανισμός.

Εφαρμογή Στοίβας

Υπολογισμών Εκφράσεων (infix notation)

Για να υπολογίσουμε ενθεματικές (infix) παραστάσεις, θα κάνουμε δύο βήματα:

1. μετατροπή από ενθεματική σε μεταθεματική (postfix) αναπαράσταση
2. υπολογισμός την μεταθεματικής παράστασης

Και τα δύο βήματα θα γίνουν με την χρήση στοίβας.

Εφαρμογή Στοίβας

Μεταθεματική Αναπαράσταση (postfix notation)

Η παράσταση

$$5 \times (((9 + 8) \times (4 \times 6)) + 7)$$

γράφεται ως εξής σε postfix

$$5 \;\; 9 \;\; 8 + 4 \;\; 6 \times \times \; 7 + \times$$

Για να επιστρέψουμε σε ενθεματική μορφή αντικαθιστούμε όλες τις εμφανίσεις των όρων $a \;\; b \times$ και $a \;\; b +$ με $(a \times b)$ και $(a + b)$.

Εφαρμογή Στοίβας

Υπολογισμός Μεταθεματικής Παράστασης (postfix notation)

Για να υπολογίσουμε την τιμή μιας μεταθεματικής παράστασης καθώς διαβάζουμε σύμβολα κάνουμε τα εξής:

- εαν το σύμβολο που διαβάσαμε είναι αριθμός, τον αποθηκεύσουμε μέσα στην στοίβα
- εαν το σύμβολο είναι πράξη, βγάζουμε δύο στοιχεία από την στοίβα, εκτελούμε την πράξη μεταξύ τους και βάζουμε το αποτέλεσμα στην στοίβα

Εφαρμογή Στοίβας

Μετατροπή Πλήρους Ενθεματικής σε Μεταθεματική Παράσταση

Για να μετατρέψουμε μια πλήρης ενθεματική παράσταση σε μεταθεματική κάνουμε την εξής διαδικασία:

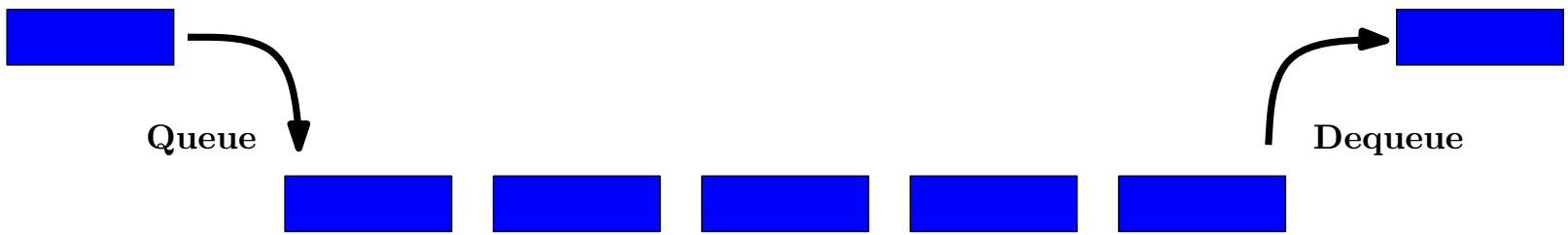
- ωθούμε τους τελεστές σε μια στοίβα και τους όρους (αριθμούς) στην έξοδο
- κάθε φορά που βλέπουμε μια δεξιά παρένθεση $)$, αποθούμε ένα τελεστή από την στοίβα και τον στέλνουμε στην έξοδο

Ουρές FIFO

πρώτο μέσα, πρώτο έξω (first-in, first-out)

Παρόμοια δομή με την στοίβα, με αντίθετο κανόνα διαγραφής.

Αφαιρείται το παλιότερο στοιχείο



Λειτουργεί ακριβώς όπως η ουρά στο ταμείο μιας τράπεζας.