Τμήμα Πληροφορικής

Πανεπιστήμιο Ιωαννίνων

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Παναγιώτα Φατούρου

faturu@cs.uoi.gr

Σεπτέμβριος, 2005

ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ ΕΠΕΑΕΚ

ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ
ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

ΕΛΛΑΔΑ 2008 ΚΟΙΝΩΝΙΚΟ ΠΛΑΙΣΙΟ ΣΤΗΡΙΞΗΣ
Ανάπτυξη παντού. Ανάπτυξη για όλους.

Η ΠΑΙΔΕΙΑ ΣΤΗΝ ΚΟΡΥΦΗ
Επιχειρησιακό Πρόγραμμα
Εκπαίδευσης και Αρχικής
Επαγγελματικής Κατάρτισης

Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Τ.Θ. 1186, Γραφείο Α26,
Τηλ. +30 26510 98808, Fax: +30 26510 98890, URL: http://www.cs.uoi.gr/~faturu/

ΕΝΟΤΗΤΑ 1
ΕΙΣΑΓΩΓΗ

Δεδομένα

Σύνολο από πληροφορίες που πρέπει να αποθηκευτούν σε έναν υπολογιστή.

Αλγόριθμος

Αλγόριθμος είναι ένα πεπερασμένο σύνολο βημάτων/εντολών αυστηρά καθορισμένων (και εκτελέσιμων μέσα σε πεπερασμένο χρόνο), τα οποία αν ακολουθηθούν επιλύεται κάποιο πρόβλημα.

Είσοδος: Δεδομένα που παρέχονται εξ αρχής στον αλγόριθμο.

Έξοδος: δεδομένα που αποτελούν το αποτέλεσμα του αλγορίθμου.

Ευκρίνια/Αποτελεσματικότητα: Κάθε εντολή θα πρέπει να είναι απλή και ο τρόπος εκτέλεσής της να καθορίζεται χωρίς καμία αμφιβολία.

Περατότητα: Ο αλγόριθμος θα πρέπει να τερματίζει μετά από πεπερασμένα βήματα εκτέλεσης των εντολών του.

Πρόγραμμα

Υλοποίηση ενός αλγορίθμου σε κάποια γλώσσα προγραμματισμού.

Παράδειγμα

Εύρεση ενός στοιχείου Κ σε έναν ταξινομημένο πίνακα?

Αλγόριθμοι (περιγραφή σε φυσική γλώσσα)

1. Ξεκινώντας από το πρώτο στοιχείο του πίνακα προσπέλασε κάθε στοιχείο του πίνακα και εξέτασε αν είναι το Κ, μέχρι είτε να βρεθεί το Κ, ή να εξεταστούν όλα τα στοιχεία του πίνακα.

2.

3.

4. Ξεκινώντας από το πρώτο στοιχείο του πίνακα προσπέλασε κάθε στοιχείο του πίνακα και εξέτασε αν είναι το Κ, μέχρι είτε να βρεθεί το Κ, ή να βρεθεί κάποιο στοιχείο μεγαλύτερο από το Κ ή να εξεταστούν όλα τα στοιχεία του πίνακα.

5.

6.

7. Δυαδική Αναζήτηση

- Ξεκίνησε από το μεσαίο στοιχείο του πίνακα. Αν αυτό είναι το Κ επέστρεψε. Διαφορετικά, σύγκρινε το μεσαίο στοιχείο με το Κ. Αν είναι μικρότερο, το Κ βρίσκεται στο δεξί μισό του πίνακα, αν όχι τότε βρίσκεται στο αριστερό μισό του πίνακα. Σε κάθε περίπτωση μόνο το ένα μισό του πίνακα πρέπει να εξεταστεί. Επανέλαβε την διαδικασία αυτή μέχρι είτε να βρεθεί το Κ ή το προς εξέταση μέγεθος του πίνακα να μηδενιστεί.

Παράδειγμα

Ύψωση ενός αριθμού x σε μια ακέραια δύναμη n.

Αλγόριθμος 1 (Περιγραφή με ψευδο-κώδικα)

```
double power1(double x, int n)
    int j = 0;
    double y = 1;

    while (j < n)
        y = y*x;
        j = j+1;
    return y;
```

Υλοποίηση του Αλγ. 1 στη γλώσσα C

```
double power_1(double x, int n) {
    int j;
    double y = 1;

    j = n;                  /*  j = 0 */
    while (j > 0 ) {        /*  while (j < n) { */
        y = y*x;
        j = j -1;           /* j = j +1 */
    }
    return y;
}
```

Κριτήρια Επιλογής Αλγορίθμων

- Ταχύτητα
- Απαιτούμενος χώρος μνήμης
- Ευκολία προγραμματισμού
- Γενικότητα

Μας ενδιαφέρει κυρίως η ταχύτητα και ο απαιτούμενος χώρος μνήμης. Οι δύο αυτοί παράμετροι καθορίζουν την αποδοτικότητα ενός αλγόριθμου.

Είναι η $Power_1$ ο πιο αποδοτικός αλγόριθμος για το πρόβλημα μας?

Αλγόριθμος $Power_2$: είσοδος & έξοδος ίδια με πριν

```
    double y
1.  if (n == 1) return x
2.  y = power2(x, ⌊n/2⌋)
3.  if n is even
4.          return y*y;
5.  else
6.          return x*y*y;
```

Μοντέλο Εργασίας

Ο υπολογιστής στον οποίο δουλεύουμε έχει έναν επεξεργαστή και ένα μεγάλο μπλοκ μνήμης.

Κάθε θέση μνήμης έχει μια αριθμητική διεύθυνση (διεύθυνση μνήμης).

Η μνήμη είναι τυχαίας προσπέλασης, το οποίο σημαίνει πως η πρόσβαση σε οποιαδήποτε θέση μνήμης (ανεξάρτητα από τη διεύθυνση στην οποία βρίσκεται) γίνεται με την ίδια ταχύτητα.

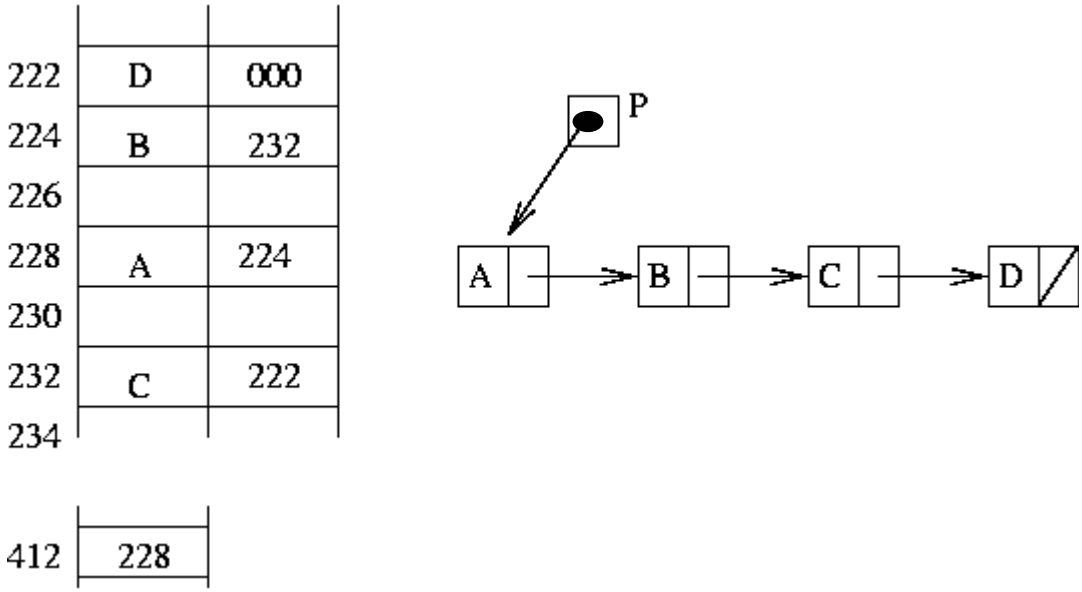
Συνεχόμενες θέσεις μνήμης μπορούν να αποθηκεύσουν τα πεδία μιας εγγραφής. Τα πεδία μιας εγγραφής μπορεί να μην είναι του ίδιου τύπου.

Συνεχόμενες θέσεις μνήμης μπορούν επίσης να αποθηκεύσουν ένα πίνακα (δηλαδή ένα σύνολο στοιχείων του ίδιου τύπου).

Μια μεταβλητή στην οποία είναι αποθηκευμένη μια διεύθυνση μνήμης ονομάζεται μεταβλητή δείκτη (ή δείκτης).

Γιατί οι δείκτες είναι πολύ χρήσιμοι;

Απλά Συνδεδεμένη Λίστα



Γιατί οι λίστες είναι πολύ χρήσιμες;

Σύγκριση με Πίνακες

Θετικά

☺ Εισαγωγή/διαγραφή νέων στοιχείων γίνεται πολύ εύκολα

☺ Ο συνολικός αριθμός στοιχείων δεν χρειάζεται να είναι γνωστός εξ αρχής

Αρνητικά

☹ Απαιτούν περισσότερη μνήμη (λόγω των δεικτών).

☹ Ανάκτηση στοιχείου για το οποίο είναι γνωστή η θέση του στη δομή (π.χ., ανάκτηση του $5^{ου}$ στοιχείου) δεν μπορεί να γίνει σε σταθερό χρόνο.

Αφηρημένος Τύπος Δεδομένων

Αποτελείται από δύο μέρη:

- Ένα ή περισσότερα πεδία (σύνολα αντικειμένων, classes of mathematical objects)
- Ένα σύνολο λειτουργιών (mathematical operations) στα στοιχεία αυτών των πεδίων.

Παράδειγμα (εύρεση στοιχείου σε πίνακα)

- Τα δεδομένα μας είναι κάποιου τύπου, έστω key, και υπάρχει μια γραμμική διάταξη ανάμεσά τους:

  $\forall$ *u,v τύπου key, είτε u* < *v, ή v* < *u, ή v* = *u.*

- Έχουμε ένα σύνολο S από στοιχεία τύπου key και θέλουμε να μπορούμε να απαντάμε το ερώτημα: u $\in$ S?

Πεδία:

- στοιχεία τύπου key (π.χ., u,v) και πεπερασμένα σύνολα αυτών (π.χ., S)

Σύνολο λειτουργιών:

- Παροχή true ή false απάντησης στην ερώτηση «u $\in$ S?”, όπου: u είναι στοιχείο τύπου key, S είναι πεπερασμένο σύνολο από στοιχεία τύπου key.

Δομή Δεδομένων

Μια δομή δεδομένων υλοποιεί έναν αφηρημένο τύπο δεδομένων.

Μια δομή δεδομένων επομένως συμπεριλαμβάνει:

- ένα σύνολο αποθηκευμένων δεδομένων τα οποία μπορούν να υποστούν επεξεργασία από ένα σύνολο λειτουργιών που σχετίζονται με τη συγκεκριμένη δομή

- μια δομή αποθήκευσης

- ένα σύνολο από ορισμούς συναρτήσεων, όπου η κάθε συνάρτηση εκτελεί μια λειτουργία στο περιεχόμενο της δομής, και

- ένα σύνολο από αλγόριθμους, έναν αλγόριθμο για κάθε συνάρτηση.

Οι βασικές λειτουργίες επί των δομών δεδομένων είναι οι ακόλουθες:

- Προσπέλαση
- Εισαγωγή
- Διαγραφή
- Αναζήτηση
- Ταξινόμηση
- Αντιγραφή
- Συγχώνευση
- Διαχωρισμός

Μαθηματικό Υπόβαθρο

- Μονοτονία Συναρτήσεων
- Ακέραια Μέρη πραγματικών αριθμών
  - $\lfloor x \rfloor, \lceil x \rceil, [x]$
- Πολυώνυμα – Πολυωνυμικές Συναρτήσεις
  $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$
- Λογάριθμοι & Εκθέτες – Ιδιότητες
- Χρήσιμοι Συμβολισμοί:
  - $\lg n = \log_2 n$
  - $\ln n = \log_e n$
  - $\lg^k n = (\lg n)^k$
  - $\lg \lg n = \lg(\lg n)$
  -
- Παραγοντικά
  - $n!$
- Άθροισμα όρων γεωμετρικής ή αριθμητικής προόδου

Μαθηματική Επαγωγή

Παράδειγμα 1 – Βασική Μέθοδος

Ορθότητα Αλγόριθμου Power$_1$

Με επαγωγή στο j.

Συμβολίζουμε με y_j την τιμή της μεταβλητής y στην αρχή της j-οστής ανακύκλωσης, j = 1, …, n+1

Αρκεί να δείξουμε πως $y_j = x^{j-1}$.

Βάση επαγωγής

j = 1. Αρχικά, $y_1 = 1 = x^0 = x^{1-1} = x^{j-1}$.

Επαγωγική Υπόθεση

Έστω ότι για κάποιο k, $1 \leq k < n+1$, $y_k = x^{k-1}$.

Επαγωγικό Βήμα

Θα δείξουμε ότι ο ισχυρισμός ισχύει για (k+1): $y_{k+1} = x^k$.

Από αλγόριθμο: $y_{k+1} = y_k * x$.

Από επαγωγική υπόθεση: $y_k = x^{k-1}$.

Άρα, $y_{k+1} = x^k$, όπως απαιτείται.

Παράδειγμα 3: Ορθότητα Προγ/τος Power_1

Με επαγωγή στο j, j = n, …, 0.

Αρκεί να δείξω ότι $y_{n-j+1}=x^{n-j}$

Βάση Επαγωγής

j = n:

$y_{n-n+1} = y_1 = 1 = x^0 = x^{n-n}$, όπως απαιτείται.

Επαγωγική Υπόθεση

Έστω ότι για κάποιο k, $n \geq k > 0$, ισχύει ότι $y_{n-k+1} = x^{n-k}$.

Επαγωγικό Βήμα

Θα δείξουμε ότι ο ισχυρισμός ισχύει για (k-1): $y_{n-k+2} = x^{n-k+1}$.

Από αλγόριθμο: $y_{n-k+2} = y_{n-k+1} * x$.

Από επαγωγική υπόθεση: $y_{n-k+1} = x^{n-k}$.

Άρα, $y_{n-k+2} = x^{n-k+1}$, όπως απαιτείται.

Παράδειγμα 3 – Ισχυρή Επαγωγή
Αιγυπτιακός Πολλαπλασιασμός

Δίνεται η συνάρτηση:

$$m(x,y) = \begin{cases} 0, \text{ αν } y = 0 \\ m(x+x, y/2), \text{ αν } y \text{ ζυγός } \& \neq 0 \\ x + m(x, y-1), \text{ διαφορετικά} \end{cases}$$

Θα δείξω ότι $m(x,y) = x*y$, $\forall$ ακέραιο x,y

Απόδειξη

Με επαγωγή στο y.

Βάση της επαγωγής

Αν $y = 0$, $m(x,y) = 0$, αλλά και $x*y = 0$, οπότε ισχύει.

Επαγωγική Υπόθεση

Φιξάρουμε μια τιμή του $y > 0$. Υποθέτουμε ότι $m(x,z) = x*z$, για κάθε z, $0 \leq z < y$.

Επαγωγικό Βήμα

Αποδεικνύουμε τον ισχυρισμό για την τιμή y: $m(x*y) = x*y$.

- y περιττός: $m(x,y) = x + m(x,y-1)$. Επαγωγική υπόθεση ($z = y-1 < y$): $m(x,y-1) = x*(y-1)$. Άρα: $m(x,y) = x + x*(y-1) = x + x*y - x = x*y$
- y άρτιος: $m(x,y) = m(x+x, y/2)$. Επαγωγική υπόθεση ($z = y/2 < y$): $m(x,y) = (x+x)*y/2 = x*y$. Άρα: $m(x,y) = x*y$.

Ανάλυση Αλγορίθμων

Εμπειρικός τρόπος μέτρησης της επίδοσης

Θετικά: απλότητα

Αρνητικά:

- Δύσκολο να προβλεφθεί η συμπεριφορά για άλλα σύνολα δεδομένων.
- Ο χρόνος επεξεργασίας εξαρτάται από το υλικό, τη γλώσσα προγραμματισμού και το μεταφραστή, αλλά και από το πόσο δεινός είναι ο προγραμματιστής.

Θεωρητικός τρόπος μέτρησης της επίδοσης

Εισάγεται μια μεταβλητή n που εκφράζει το μέγεθος του προβλήματος.

Παραδείγματα

- Στο πρόβλημα ανυψώσεως σε δύναμη το μέγεθος του προβλήματος είναι το n: η δύναμη στην οποία πρέπει να υψωθεί ο δεδομένος αριθμός.
- Σε ένα πρόβλημα ταξινόμησης ενός πίνακα, το μέγεθος του προβλήματος είναι ο αριθμός στοιχείων του πίνακα.

Απλές Αναδρομικές Σχέσεις

Έστω ότι η f(n) αναπαριστά το άθροισμα των n μικρότερων ακεραίων. Τότε, η f(n) μπορεί να οριστεί αναδρομικά ως εξής:

$f(n) = 1$, *αν* $n = 1$ και $f(n) = f(n-1) + n$, *αν* $n > 1$.

Ο υπολογισμός μιας μη-αναδρομικής φόρμας για την f(n) είναι απλός:

$f(n) = f(n-1) + n$

$= f(n-2) + (n-1) + n$

$= ....$

$= f(1) + 2 + ... + (n-1) + n$

$= 1 + 2 + ... + (n-1) + n$

$= (n+1)\, n / 2$

Παράδειγμα 2

Έστω ότι $f(0) = 1$ και $f(n) = 2f(n-1)$.

Ο υπολογισμός μιας μη-αναδρομικής φόρμας για την f(n) είναι και πάλι απλός:

$f(n) = 2\, f(n-1)$

$= 2^2\, f(n-2)$

$= ...$

$= 2^n\, f(0)$

$= 2^n$.

Αναδρομές τύπου «Διαίρει και Βασίλευε»

Δυαδική Αναζήτηση

int *BinarySearch*(**table** T[a...b], **key** K)

/ επιστρέφει τη θέση του K μέσα στον ταξινομημένο πίνακα T, αν το K υπάρχει, και -1 διαφορετικά */*

```
int middle, tmp;

if (a > b) return -1;
middle = κάτω ακέραιο μέρος του (a+b)/2;
if (K == T[middle]) return middle;
else if (K < T[middle]) {
      tmp = BinarySearch(T[a...middle-1], K);
      return tmp;
}
else {
       tmp = BinarySearch(T[middle+1...b],K);
       return tmp;
}
}
```

- Κάθε μια από τις αναδρομικές κλήσεις εξετάζει περίπου το μισό πίνακα.

- Έστω ότι *n* είναι το μέγεθος του πίνακα (*n=b-a+1*). Ας υποθέσουμε για λόγους απλότητας ότι $n = 2^k - 1$.

Αναδρομές τύπου «Διαίρει και Βασίλευε»

Τότε:

$$2^k - 1 = b - a + 1 \Rightarrow b = 2^k - 2 + a$$

Οπότε, κατά την 1η κλήση της BinarySearch():

$$middle = [(a+b)/2] = [(a+2^k - 2 + a)/2] = a + 2^{k-1} - 1$$

Η 1η αναδρομική κλήση αναφέρεται σε πίνακα μήκους:

$$middle - 1 - a + 1 = middle - a$$
$$= a + 2^{k-1} - 1 - a$$
$$= 2^{k-1} - 1.$$

Η 2η αναδρομική κλήση αναφέρεται σε πίνακα μήκους:

$$b - (middle + 1) + 1 = b - middle$$
$$= 2^k - 2 + a - a - 2^{k-1} + 1$$
$$= 2^{k-1} - 1.$$

- *Ποιο είναι το χειρότερο σενάριο που μπορεί να συμβεί κατά την εκτέλεση;*

  Ο πίνακας δεν περιέχει το κλειδί και οι αναδρομικές κλήσεις συνεχίζονται μέχρι να γίνει κλήση σε άδειο πίνακα.

Αναδρομές τύπου «Διαίρει και Βασίλευε»

Αν T(n) : μέγιστος δυνατός χρόνος εκτέλεσης σε πίνακα με n στοιχεία, και c,d: σταθερές

Τότε:

$$T(2^k-1) = c, \text{ αν } k = 0$$

και

$$T(2^k-1) = d + T(2^{k-1}-1), \text{ αν } k > 0.$$

Έχουμε:

$$\begin{aligned} T(2^k-1) &= d + T(2^{k-1}-1) \\ &= 2d + T(2^{k-2}-1) \\ &= \ldots \\ &= kd + T(0) \\ &= kd + c. \end{aligned}$$

Αφού $n = 2^k-1$, είναι $k = \log(n+1)$ και

$$T(n) = d \log(n+1) + c.$$

Ανάλυση Αλγορίθμων

Πρόβλημα Ταξινόμησης

Είσοδος (input): Μια ακολουθία από n αριθμούς $<a_1, a_2, ..., a_n>$.

Έξοδος (output): Μια «αναδιάταξη» $< a'_1, a'_2, ..., a'_n>$ της ακολουθίας εισόδου έτσι ώστε:

$$a'_1 \leq a'_2 \leq ... \leq a'_n$$

InsertionSort

```
void InsertionSort(int A[n]) {
/* ταξινόμηση των στοιχείων του A */
    int key, i, j;
    for (j = 1; j < n; j++) {
        key = A[j];
        i = j-1;
        while (i >= 0 && A[i] > key) {
            A[i+1] = A[i];
            i = i-1;
        }
        A[i+1] = key;
    }
}
```

Πως λειτουργεί ο αλγόριθμος αν A = <5,2,4,6,1,3>?

Ανάλυση του Αλγορίθμου InsertionSort

Ο χρόνος που σπαταλιέται από την InsertionSort εξαρτάται από την είσοδο:

«Η ταξινόμηση 1000 αριθμών απαιτεί περισσότερο χρόνο από την ταξινόμηση 3 αριθμών»!

Αυτό ισχύει γενικότερα:

«Ο χρόνος που απαιτεί ένας αλγόριθμος για να εκτελεστεί εξαρτάται από το μέγεθος της εισόδου»!

Χρόνος Εκτέλεσης

Ο χρόνος εκτέλεσης (running time) ενός αλγορίθμου **για μια συγκεκριμένη είσοδο** είναι ο αριθμός των βασικών λειτουργιών/εντολών (ή βημάτων) που εκτελούνται κατά την εκτέλεση του αλγορίθμου με αυτή την είσοδο.

Ανάλυση Αλγορίθμων

```
void InsertionSort(int A[n]) {                          Κόστος   Χρόνος
    int key, i, j;
    for (j = 1; j < n; j++) {          ---------->        c1        n
        key = A[j];                    ---------->        c2        n-1
        i = j-1;                       ---------->        c3        n-1
        while (i >= 0 && A[i] > key) { ---------->        c4        Σ tj
            A[i+1] = A[i];             ---------->        c5        Σ (tj – 1)
            i = i-1;                   ---------->        c6        Σ (tj – 1)
        }
        A[i+1] = key;                  ---------->        c7        n-1
    }
}
```

t_j: ο αριθμός των φορών που ο έλεγχος του while loop εκτελείται για αυτή την τιμή του j.

Ανάλυση του Αλγορίθμου InsertionSort

Συνολικός Χρόνος Εκτέλεσης

$T(n) = c_1n + c_2(n-1) + c_3(n-1) + c_4 \Sigma\ t_j + c_5 \Sigma\ (t_j-1) + c_6 \Sigma\ (t_j-1) + c_7(n-1)$

- *Ποιος είναι ο καλύτερος χρόνος εκτέλεσης που μπορεί να επιτευχθεί από την InsertionSort?*

  Ο πίνακας είναι εξ αρχής ταξινομημένος σε αύξουσα διάταξη.

  Τότε, $t_j = 1, \forall\ j = 1,2, \ldots, n-1$:

  $$T(n) = c_1n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$
  $$= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

  $\Rightarrow$ **γραμμική συνάρτηση του n!**

Ανάλυση του Αλγορίθμου InsertionSort

- *Ποιος είναι ο χειρότερος χρόνος εκτέλεσης κατά την εκτέλεση της InsertionSort?*

  Τα στοιχεία του πίνακα είναι σε φθίνουσα διάταξη. Τότε, $t_j = (j+1)$, $\forall$ j = 1, …, n-1 και:

  - $\Sigma\ t_j = \Sigma\ (j+1)$
    $= 2 + 3 + \ldots + n$
    $= (1+2+ \ldots + n) - 1$
    $= n(n+1)/2 - 1$

  - $\Sigma\ (t_j - 1) = \Sigma\ j$
    $= 1 + 2 + \ldots + (n-1)$
    $= n(n-1)/2$

  Επομένως:

  $$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n(n+1)/2 - 1) + c_5\, n(n-1)/2 + c_6\, n(n-1)/2 + c_7(n-1)$$
  $$= \ldots$$
  $$= (c_4/2 + c_5/2 + c_6/2)\, n^2 + (c_1+c_2+c_3+c_4/2 - c_5/2-c_6/2+c_7)\, n - (c_2+c_3+c_4+c_7)$$

  $\Rightarrow$ τετραγωνική συνάρτηση του n!

Χρονική Πολυπλοκότητα
Χείριστος Χρόνος Εκτέλεσης

Ο χείριστος χρόνος εκτέλεσης (ή χρονική πολυπλοκότητα) ενός αλγορίθμου ορίζεται να είναι ο μέγιστος χρόνος εκτέλεσης για κάθε είσοδο μεγέθους n (και είναι συνάρτηση του n).

- *Πως μπορούμε να βρούμε ένα πάνω όριο στη χρονική πολυπλοκότητα ενός αλγορίθμου;*

- *Πως μπορούμε να βρούμε ένα κάτω όριο στην χρονική πολυπλοκότητα ενός αλγορίθμου;*

Τάξη Χρονικής Πολυπλοκότητας

Η χρονική πολυπλοκότητα της InsertionSorn είναι an^2+bn+c, όπου $a = c_4/2 + c_5/2 + c_6/2$, $b = c_1+c_2+c_3+c_4/2-c_5/2-c_6/2+c_7$ και $c = c_2+c_3+c_4+c_7$.

Τα κόστη c_i δεν μας δίνουν χρήσιμη πληροφορία.

Πολλές φορές, κατά τη μελέτη της χρονικής πολυπλοκότητας, είναι χρήσιμο να θεωρούμε ακόμη πιο απλουστευτικές αφαιρέσεις από την an^2+bn+c.

Μας ενδιαφέρει μόνο ο ρυθμός μεταβολής της συνάρτησης της χρονικής πολυπλοκότητας:

- Από το άθροισμα an^2+bn+c μας ενδιαφέρει μόνο ο κυρίαρχος όρος an^2, αφού οι άλλοι δύο όροι είναι μη-σημαντικοί για μεγάλες τιμές του n.

- Αγνοούμε επίσης το συντελεστή a, αφού οι σταθεροί παράγοντες είναι λιγότεροι σημαντικοί για μεγάλες τιμές του n.

Η χρονική πολυπλοκότητα της InsertionSort είναι τετραγωνικής τάξης.

Τάξη Χρονικής Πολυπλοκότητας

Μερικές φορές μπορούμε να υπολογίσουμε την ακριβή πολυπλοκότητα ενός αλγορίθμου.

Όταν το n (είσοδος) είναι μεγάλο, οι σταθερές και οι χαμηλότερης τάξης όροι δεν είναι σημαντικοί (κυριαρχεί έναντι αυτών ο υψηλότερης τάξης όρος).

Η μελέτη της πολυπλοκότητας για πολύ μεγάλα n (όριο συνάρτησης πολυπλοκότητας όταν το n τείνει στο άπειρο), ονομάζεται ασυμπτωτική μελέτη.

Συνήθως, ένας αλγόριθμος που είναι ασυμπτωτικά ο πιο αποτελεσματικός, είναι η καλύτερη επιλογή για όλες εκτός από πολύ μικρές εισόδους.

Ανάλυση Αλγορίθμων

Ο χρόνος επεξεργασίας ή ο απαιτούμενος χώρος μνήμης εκτιμώνται με τη βοήθεια μιας συνάρτησης f(n) που εκφράζει τη χρονική πολυπλοκότητα ή την πολυπλοκότητα χώρου.

Μας ενδιαφέρει κυρίως η γενική συμπεριφορά αυτής της συνάρτησης, δηλαδή η τάξη της.

Συμβολισμός Ο

Έστω f(n) και g(n) δύο συναρτήσεις. Η f(n) είναι O(g(n)), f(n) = O(g(n)), αν υπάρχουν σταθερές $c \in R^+$ (c πραγματικός, c > 0) και ακέραιος $n_0 \geq 0$, έτσι ώστε για κάθε $n \geq n_0$ να ισχύει:

$$0 \leq f(n) \leq cg(n)$$

Παράδειγμα

Έστω $f(n) = an^2 + bn$, όπου a,b θετικές σταθερές.

? $f(n) = O(n^2)$.

Ψάχνουμε για c & n_0 τ.ω.

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

$0 \leq (c-a)n^2 - bn \Rightarrow$

$0 \leq n\,[(c-a)n - b] \Rightarrow$

$(c-a)n - b \geq 0$

Αν επιλέξουμε c = a+1 και οποιοδήποτε $n_0 \geq b$, η ανισότητα (c-a)n - b ≥ 0 ισχύει.

Συμβολισμός Ω

Έστω f(n) και g(n) δύο συναρτήσεις. Η f(n) είναι Ω(g(n)), f(n) = Ω(g(n)), αν υπάρχουν σταθερές $c \in R^+$ (c πραγματικός, c > 0) και ακέραιος $n_0 \geq 0$, έτσι ώστε για κάθε $n \geq n_0$ να ισχύει:

$$0 \leq cg(n) \leq f(n)$$

Παράδειγμα

Θα αποδείξω ότι f(n) = Ω(n).

$an^2 + bn \geq cn \Rightarrow$

$an^2 + (b-c)n \geq 0 \Rightarrow$

$an + b-c \geq 0.$

Αν επιλέξουμε c = b ισχύει ότι $an \geq 0$, για κάθε $n \geq 0$, άρα για c = b & $n_0 = 0$, ο ισχυρισμός αποδεικνύεται.

Συμβολισμός Θ

Έστω f(n) και g(n) δύο συναρτήσεις. Η f(n) είναι Θ(g(n)), f(n) = Θ(g(n)), αν f(n) = O(g(N)) και f(n) = Ω(g(n)).

Παράδειγμα

Αποδεικνύουμε πως f(n) = $\Omega(n^2)$, οπότε f(n) = $\Theta(n^2)$.

Ψάχνουμε για c > 0 & $n_0 \geq 0$, τ.ω.

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

$0 \geq (c-a)n^2 - bn \Rightarrow$

$0 \geq n\,[(c-a)n - b] \Rightarrow$

$(c-a)n - b \leq 0.$

Αν επιλέξουμε c = a/2 > 0 ισχύει ότι (c-a)n-b = -an/2-b ≤ 0, για κάθε n ≥ 0, άρα για c = a/2 & n_0 = 0, ο ισχυρισμός αποδεικνύεται.

Συνήθεις Κατηγορίες Χρονικής Πολυπλοκότητας

O(1): σταθερή πολυπλοκότητα

O(logn): λογαριθμική πολυπλοκότητα

O(n): γραμμική πολυπλοκότητα

O(n logn): πολυπλοκότητα O(n log n) (συνήθης πολυπλοκότητα βέλτιστων αλγόριθμων ταξινόμησης)

O(n^2): τετραγωνική πολυπλοκότητα

O(n^3): κυβική πολυπλοκότητα

O(n^k), για κάποιο προκαθορισμένο ακέραιο k: πολυωνυμική πολυπλοκότητα

O(2^n): εκθετική πολυπλοκότητα

Ιδιότητες Ο, Ω, Θ

Ανακλαστική Ιδιότητα

$f(n) = O(f(n))$

$f(n) = \Omega(f(n))$

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

Μεταβατική Ιδιότητα

$f(n) = O(g(n)) \ \& \ g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$

$f(n) = \Omega(g(n)) \ \& \ g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$

$f(n) = \Theta(g(n)) \ \& \ g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$

Συμμετρική Ιδιότητα

$f(n) = \Theta(g(n))$ αν και μόνο αν $g(n) = \Theta(f(n))$

Αντίστροφη Συμμετρική Ιδιότητα

$f(n) = O(g(n))$ αν και μόνο αν $g(n) = \Omega(f(n))$

Καλή άσκηση για το σπίτι

Αποδείξτε τις παραπάνω ιδιότητες.

ΕΝΟΤΗΤΑ 2
ΠΙΝΑΚΕΣ

Πίνακες

Δομή δεδομένων που αποθηκεύει μια ακολουθία από διαδοχικά αριθμημένα αντικείμενα.

Πιο φορμαλιστικά:

- ✓ l,u: ακέραιοι

- ➢ Το διάστημα l…u είναι το σύνολο των ακεραίων j τ.ω. $l \leq j \leq u$.

- ➢ Ένας πίνακας είναι μια συνάρτηση με πεδίο ορισμού ένα διάστημα, που λέγεται σύνολο δεικτών, και πεδίο τιμών ένα σύνολο από αντικείμενα (τα στοιχεία του πίνακα).

- ✓ X: πίνακας
- ✓ j: δείκτης στο σύνολο δεικτών του X
- ✓ X[j]: το j-οστό στοιχείο του X

Λειτουργίες σε πίνακες

- *Access(X,j):* επιστρέφει X[j]
- *Length(X):* Επιστρέφει u, τον αριθμό των στοιχείων του X
- *Assign(X,j,a):* εκτελεί τη λειτουργία X[j] = a
- *Initialize(X,a):* Αρχικοποιεί σε a κάθε στοιχείο του X
- *Iterate(X,f):* Εφαρμόζει την συνάρτηση f σε κάθε ένα στοιχείο του X, ξεκινώντας από το μικρότερο και καταλήγοντας στο μεγαλύτερο.

Πολυδιάστατοι Πίνακες

- Ένας διδιάστατος πίνακας ορίζεται ως ο μονοδιάστατος πίνακας του οποίου τα στοιχεία είναι μονοδιάστατοι πίνακες.
- Ένας d-διάστατος πίνακας ορίζεται ως ο μονοδιάστατος πίνακας του οποίου τα στοιχεία είναι (d-1)-διάστατοι πίνακες.
- Αν $<j_1, \ldots, j_d>$ είναι ένα διάνυσμα d ακεραίων που ανήκει στο σύνολο δεικτών ενός d-διάστατου πίνακα X, τότε $X[j_1,\ldots,j_d]$ είναι η τιμή του πίνακα για το συγκεκριμένο δείκτη.

Υλοποίηση Πινάκων σε Διαδοχικές Θέσεις Μνήμης

- Στη C το όνομα, π.χ. X, ενός πίνακα είναι και δείκτης στο 1ο στοιχείο του.
- Αν L είναι το μέγεθος κάθε στοιχείου του πίνακα, τότε το j-οστό στοιχείο του πίνακα βρίσκεται στη διεύθυνση X + L*(j-1).
- Κάποιες φορές το μήκος του πίνακα δεν είναι γνωστό και αντί αυτού χρησιμοποιείται ένα στοιχείο φρουρός το οποίο σηματοδοτεί το τέλος του πίνακα (αυτό π.χ. γίνεται στη C με τα strings).

Διδιάστατοι Πίνακες

Στη C οι πίνακες αποθηκεύονται κατά γραμμές:

X[0,0], X[0,1], ... , X[0, m], X[1,0], ... , X[1,m], ... , X[n,0], ... , X[n,m].

	0	**1**	**...**	**m**
0	[0,0]	[0,1]	…	[0,m]
1	[1,0]	[1,1]	…	[1,m]
...	…	…	…	…
n	[n,0]	[n,1]	…	[n,m]

Άρα το στοιχείο X[k,j] βρίσκεται στη θέση μνήμης X + L*(m*k + j). *Γιατί?*

Ειδικές Μορφές Πινάκων

Συμμετρικοί Πίνακες

Συμμετρικός λέγεται ο τετραγωνικός πίνακας με στοιχεία X[k,j] = X[j,k], για κάθε $0 \le j,k \le n$.

	0	**1**	**2**
0	5	34	31
1	34	6	87
2	31	87	45

Υπάρχουν 1 + 2 + ... + n διαφορετικά στοιχεία και άρα απαιτούνται (1+n)*n/ 2 θέσεις μνήμης για την αποθήκευση του πίνακα.

Τριγωνικοί Πίνακες

Για τα στοιχεία ενός πάνω (κάτω) τριγωνικού πίνακα ισχύει X[k,j] = 0, αν k < j (αντίστοιχα k > j), όπου $1 \le k,j \le n$.

1	0	0	0
21	43	0	0
34	7	18	0
5	86	23	43

Πως μπορούμε να υλοποιήσουμε τριδιαγώνιους πίνακες χωρίς μεγάλη σπατάλη μνήμης?

Αραιοί Πίνακες

Ένας πίνακας λέγεται αραιός, όταν ένα μεγάλο ποσοστό των στοιχείων του έχουν την τιμή 0.

Παράδειγμα

0	7	0	0	0
1	2	0	0	-3
0	0	4	0	0
12	0	0	0	0

Τρόπος Αποθήκευσης

Αποθήκευση του αντίστοιχου δυαδικού πίνακα:

0	1	0	0	0
1	1	0	0	1
0	0	1	0	0
1	0	0	0	0

ακολουθούμενο από ένα μονοδιάστατο πίνακα με τιμές:

(7 1 2 -3 4 12)