Αναζήτηση και ταξινόμηση

Περιεχόμενα

- Αναζήτηση (searching): εύρεση ενός στοιχείου σε έναν πίνακα
- Ταξινόμηση (sorting): αναδιάταξη των στοιχείων ενός πίνακα ώστε να είναι τοποθετημένα με μια καθορισμένη σειρά
- Η αναζήτηση και η ταξινόμηση έχουν ιδιαίτερη σημασία στον προγραμματισμό
  - αποτελούν συχνά χρησιμοποιούμενες προγραμματιστικές τεχνικές
  - αποτελούν καλά παραδείγματα προβλημάτων για τα οποία υπάρχουν αλγόριθμοι με διαφορετική απόδοση

Παράδειγμα αναζήτησης

- Αναζήτηση του πρώτου φωνήεντος σε ένα αλφαριθμητικό

```
int FindFirstVowel(string word)
{
    int i;

    for (i = 0; i < StringLength(word); i++) {
        if (IsVowel(IthChar(word, i))) return (i);
    }
    return (-1);
}
```

- Για τις επισημασμένες συναρτήσεις δείτε το κεφ. 9.

Αναζήτηση σε πίνακα ακεραίων

- Η συνάρτηση αναζήτησης

```
int FindIntegerInArray(int key, int array[], int n)
{
    int i;

    for (i = 0; i < n; i++) {
        if (key == array[i]) return (i);
    }
    return (-1);
}
```

Παράλληλοι πίνακες

coinValues

0	1
1	2
2	5
3	10
4	20

coinNames

0	lepto
1	dilepto
2	pentalepto
3	dekalepto
4	eikosaletpo

- Οι πίνακες στους οποίους χρησιμοποιούνται αντίστοιχοι αριθμοδείκτες θέσης για την αποθήκευση συσχετιζόμενων τιμών ονομάζονται **παράλληλοι πίνακες**.

Παράδειγμα με χρήση παράλληλων πινάκων

- findcoin.c
- Το πρόγραμμα δέχεται ως είσοδο την αξία ενός νομίσματος και επιστρέφει το όνομά του.

Ένα δεύτερο παράδειγμα

- Ένα πρόγραμμα το οποίο
  - διαβάζει τα ονόματα δύο πόλεων
  - επιστρέφει την (απευθείας) απόσταση μεταξύ τους

- Το πρόγραμμα χρησιμοποιεί έναν δισδιάστατο πίνακα με τις αποστάσεις μεταξύ των πόλεων
- Το πρόγραμμα χρησιμοποιεί έναν μονοδιάστατο πίνακα με τα ονόματα των πόλεων

Ο κώδικας του προγράμματος

- Το πρόγραμμα υπολογισμού αποστάσεων πόλεων

Γραμμική αναζήτηση

- Αλγόριθμος γραμμικής αναζήτησης
  - Η αναζήτηση ξεκινά από την αρχή του πίνακα και διατρέχονται όλα του τα στοιχεία με τη σειρά, μέχρι να διαπιστωθεί ταύτιση ή να φτάσουμε στο τέλος του πίνακα.
- Ο αριθμός των βημάτων εκτέλεσης του αλγορίθμου (συγκρίσεων) είναι **ανάλογος** του μεγέθους του πίνακα.

Δυαδική αναζήτηση

- Θεωρούμε ότι ο πίνακας είναι ταξινομημένος

0	1
1	12
2	32
3	41
4	42
5	63
6	84
7	96
8	123
9	221
10	228
11	246

Αναζήτηση του 221

Συνάρτηση που υλοποιεί τη δυαδική αναζήτηση

```
static int FindStringInSortedArray(string key,
                                   string array[],
                                   int n)
{
    int lh, rh, mid, cmp;

    lh = 0;
    rh = n - 1;
    while (lh <= rh) {
        mid = (lh + rh) / 2;
        cmp = StringCompare(key, array[mid]);
        if (cmp == 0) return (mid);
        if (cmp < 0) {
            rh = mid - 1;
        } else {
            lh = mid + 1;
        }
    }
    return (-1);
}
```

Αποδοτικότητα του αλγορίθμου αναζήτησης

- Έστω πίνακας με N στοιχεία
- Μετά την πρώτη σύγκριση η αναζήτηση θα συνεχιστεί σε N/2 στοιχεία
- Αν k είναι ο αριθμός των βημάτων μέχρι να τελειώσει η αναζήτηση
  - $N = 2^k$
- Άρα $k = \log_2 N$

Σύγκριση γραμμικής και δυαδικής αναζήτησης

N	**$\log_2 N$**
10	3
100	7
1000	10
1.000.000	20
1.000.000.000	30

Ταξινόμηση

- Η διάταξη μιας λίστας τιμών (συνήθως σε μορφή πίνακα) σε μια καθορισμένη σειρά.
- Παράδειγμα: Ταξινόμηση ακεραίων

12	4	24	1	43	32	11	31	42
0	1	2	3	4	5	6	7	8

1	4	11	12	24	31	32	42	43
0	1	2	3	4	5	6	7	8

Μια συνάρτηση ταξινόμησης ακεραίων

```
void SortIntegerArray(int array[],int
  n);
```

- Μετά την εκτέλεση της παραπάνω συνάρτησης τα στοιχεία του πίνακα **`array`** μεγέθους **`n`** θα είναι ταξινομημένα

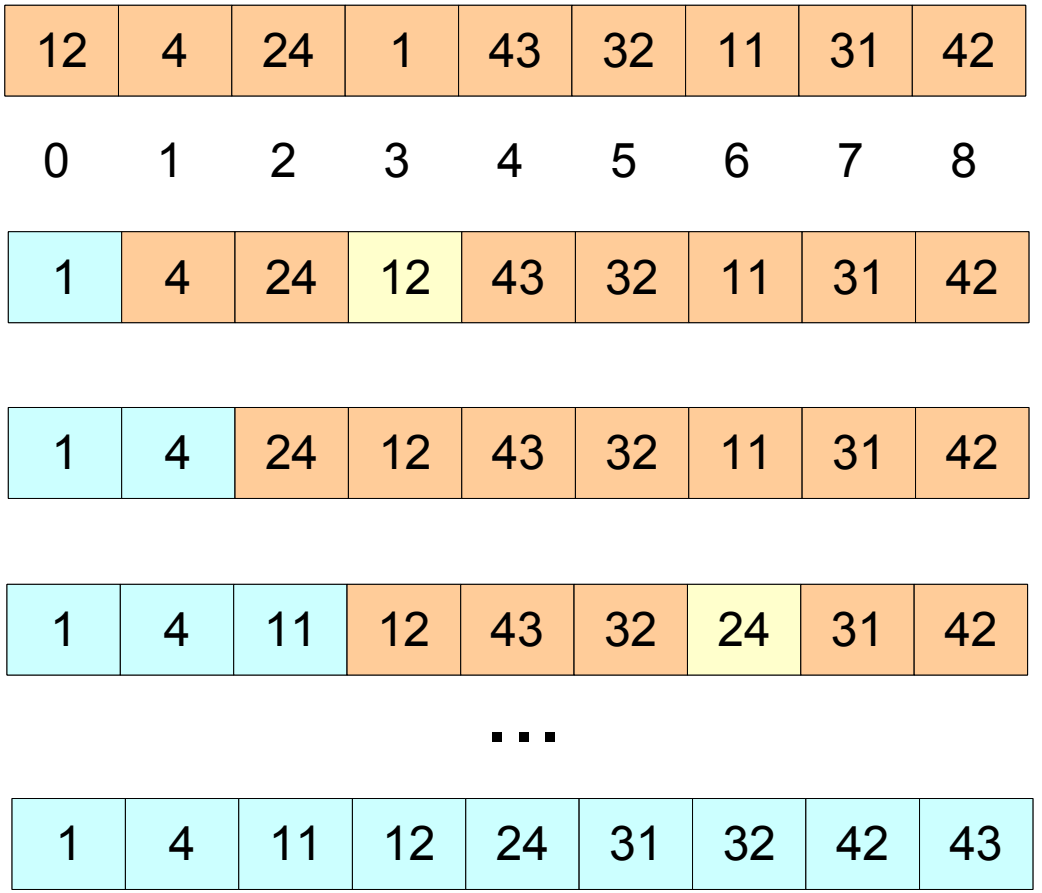
Αλγόριθμος ταξινόμησης με επιλογή

- Υπάρχει μια ποικιλία αλγορίθμων ταξινόμησης
- Θα παρουσιάσουμε έναν από τους απλούστερους: τον αλγόριθμο ταξινόμησης με επιλογή (selection sort).

Ταξινόμηση με επιλογή: περιγραφή του αλγορίθμου

1. Βρίσκουμε το μικρότερο στοιχείο του πίνακα
2. μετακινούμε το στοιχείο αυτό στην αριστερότερη θέση του πίνακα
3. επαναλαμβάνουμε τα βήματα 1 και 2 για τα **υπόλοιπα** στοιχεία του πίνακα

Ταξινόμηση με επιλογή: Παράδειγμα εκτέλεσης



Η παραπάνω διεργασία περιγράφεται από τον *ψευδοκώδικα*

for (*κάθε αριθμοδείκτη θέσης* `lh` *του πίνακα*) {

Αποθήκευσε στην `rh` *τον αριθμοδείκτη της μικρότερης τιμής μεταξύ* `lh` *και του τέλους της λίστας*

Αντιμετάθεσε τα στοιχεία με αριθμοδείκτες θέσης `lh` *και* `rh`

}

Υποπροβλήματα

- Η παραπάνω διαδικασία εισάγει δύο υποπροβλήματα:
  - Την εύρεση της θέσης του μικρότερου στοιχείου ενός πίνακα μεταξύ δυο προκαθορισμένων θέσεων

```
int FindSmallestInteger(int array[],
  int low, int high);
```

  - Την αντιμετάθεση δύο στοιχείων του πίνακα

```
void SwapIntegerElements(int array[],
  int low, int high);
```

(Γνωστή από το κεφ. 11).

Η συνάρτηση **FindSmallestInteger**

```
static int FindSmallestInteger(int array[], int low, int high)
{
    int i,spos;

    spos = low;
    for (i = low; i <= high; i++) {
        if (array[i] < array[spos]) spos = i;
    }
    return (spos);
}
```

Η συνάρτηση **SwapIntegerElements**

```
static void SwapIntegerElements(int array[],
int p1, int p2)
{
    int tmp;

    tmp = array[p1];
    array[p1] = array[p2];
    array[p2] = tmp;
}
```

Η συνάρτηση ταξινόμησης

```
void SortIntegerArray(int array[], int n)
{
    int lh, rh;

    for (lh = 0; lh < n; lh++) {
        rh = FindSmallestInteger(array, lh, n-1);
        SwapIntegerElements(array, lh, rh);
    }
}
```

Ανάλυση του αλγορίθμου ταξινόμησης με επιλογή

- Έστω ταξινόμηση πίνακα μεγέθους N
- Κατά την ταξινόμηση με επιλογή απαιτούνται
- N + (N-1) +...+ 1 βήματα
- Ισχύει

$$N+N-1+N-2+..+3+2+1=\sum_{i=0}^{N-1}(N-i)=\frac{N^2+N}{2}$$

- Αλγόριθμοι που εκτελούνται με τέτοιο αριθμό βημάτων ονομάζονται **τετραγωνικοί** και δεν είναι αποδοτικοί για μεγάλες τιμές του N