Αλγόριθμοι Ταξινόμησης Μέρος 1

Μανόλης Κουμπαράκης

Το Πρόβλημα της Ταξινόμησης

- Το πρόβλημα της **ταξινόμησης (sorting)** μιας ακολουθίας στοιχείων με κλειδιά ενός γνωστού τύπου (π.χ., τους ακέραιους ή τις συμβολοσειρές) είναι ένα από τα πιο ενδιαφέροντα προβλήματα στην Πληροφορική.
- Έχει μελετηθεί διεξοδικά και πολλές σχετικές δομές δεδομένων και αλγόριθμοι έχουν προταθεί.
- Θα παρουσιάσουμε μερικούς από αυτούς τους αλγόριθμους σ' αυτό το μέρος του μαθήματος.
- Θα ασχοληθούμε κυρίως με **ταξινόμηση πινάκων με ακέραια κλειδιά.** Τα στοιχεία του πίνακα θα πρέπει να ταξινομηθούν σε **αύξουσα σειρά**.
- Οι μέθοδοι που θα παρουσιάσουμε μπορούν να επεκταθούν σε άλλες δομές δεδομένων (π.χ., συνδεδεμένες λίστες) ή τύπους κλειδιών (π.χ., συμβολοσειρές).

Στοιχειώδεις Μέθοδοι Ταξινόμησης

- Θα μελετήσουμε πρώτα διάφορους **στοιχειώδεις** αλγόριθμους ταξινόμησης.
- Οι αλγόριθμοι αυτοί είναι προτιμότεροι των πιο πολύπλοκων αλγορίθμων ταξινόμησης που θα παρουσιάσουμε στη συνέχεια αν έχουμε μικρούς πίνακες ή πίνακες με ειδική δομή.

Ταξινόμηση με Επιλογή

- Ο αλγόριθμος της **ταξινόμησης με επιλογή (selection sort)** λειτουργεί ως εξής.
- Πρώτα βρίσκει το μικρότερο στοιχείο του πίνακα και το αντιμεταθέτει με το στοιχείο που βρίσκεται στην πρώτη θέση του πίνακα.
- Έπειτα βρίσκει το δεύτερο μικρότερο στοιχείο και το αντιμεταθέτει με το στοιχείο που βρίσκεται στην δεύτερη θέση, και συνεχίζει με αυτό τον τρόπο μέχρι να ταξινομηθεί ολόκληρος ο πίνακας.

Παράδειγμα

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	S	O	R	T	I	N	G	E	X	**A**	M	P	L	E
A	A	O	R	T	I	N	G	**E**	X	S	M	P	L	E
A	A	E	R	T	I	N	G	O	X	S	M	P	L	**E**
A	A	E	E	T	I	N	**G**	O	X	S	M	P	L	R
A	A	E	E	G	**I**	N	T	O	X	S	M	P	L	R
A	A	E	E	G	I	N	T	O	X	S	M	P	**L**	R
A	A	E	E	G	I	L	T	O	X	S	**M**	P	N	R
A	A	E	E	G	I	L	M	O	X	S	T	P	**N**	R
A	A	E	E	G	I	L	M	N	X	S	T	P	**O**	R
A	A	E	E	G	I	L	M	N	O	S	T	**P**	X	R
A	A	E	E	G	I	L	M	N	O	P	T	S	X	**R**
A	A	E	E	G	I	L	M	N	O	P	R	**S**	X	T
A	A	E	E	G	I	L	M	N	O	P	R	S	X	**T**
A	A	E	E	G	I	L	M	N	O	P	R	S	T	**X**
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Υλοποίηση σε C

- Θα δώσουμε κώδικα στη γλώσσα προγραμματισμού C που υλοποιεί τους διάφορους αλγόριθμους ταξινόμησης που παρουσιάζουμε.
- Οι συναρτήσεις που θα δώσουμε θα μπορούν να αντικαταστήσουν τη γενική συνάρτηση `sort` που χρησιμοποιούμε παρακάτω.
- Στο αρχείο διεπαφής δίνονται όλοι οι ορισμοί τύπων και όλες οι μακροεντολές που θα χρησιμοποιήσουμε.

Υλοποίηση σε C: Αρχείο Διεπαφής `sorting.h`

```
#ifndef SORTING_INCLUDED
#define SORTING_INCLUDED

/* Macros to be used in sorting algorithms */
#define key(A) (A)
#define less(A, B) (key(A) < key(B))
#define exch(A, B) { Item t = A; A = B; B = t; }
#define compexch(A, B) if (less(B, A)) exch(A, B)

/* Item is the type of elements in the arrays to be sorted */
typedef int Item;

/* Prototypes of sorting functions */
void sort(Item a[], int l, int r);

#endif
```

Υλοποίηση σε C: Το αρχείο `main.c`

```
#include <stdio.h>
#include <stdlib.h>

/* Driver program to be used for demonstrating the functionality
of sorting algorithms */

int main(int argc, char *argv[])
{
    int i, N = atoi(argv[1]), sw = atoi(argv[2]);
    int *a = malloc(N*sizeof(int));

    if (sw)
      for (i = 0; i < N; i++)
        a[i] = 1000*(1.0*rand()/RAND_MAX);
    else
      while (scanf("%d", &a[N]) == 1) N++;

    /* call the sorting algorithm */
    sort(a, 0, N-1);

    for (i = 0; i < N; i++) printf("%3d ", a[i]);
    printf("\n");

    return 0;
}
```

Υλοποίηση σε C: Το αρχείο `sort.c`

```
void sort(Item a[], int l, int r)
{
  /* details of the sorting algorithm */
}
```

Σχόλια για το `main.c`

- Η συνάρτηση `int rand(void)` επιστρέφει ένα ψευδοτυχαίο ακέραιο στο διάστημα `0` έως `RAND_MAX` που είναι τουλάχιστον `32767`.
- Η κλήση του εκτελέσιμου προγράμματος πρέπει να δίνει τιμές στο `N` και το `sw`. Με `sw==1`, το πρόγραμμα γεμίζει τον πίνακα `a` που θα ταξινομηθεί με `N` ψευδοτυχαίους αριθμούς. Με `N=0` και `sw=0`, το πρόγραμμα αναμένει από το χρήστη να εισάγει όσους ακεραίους θέλει στον πίνακα `a` (και να τερματίσει με `EOF`).

Ταξινόμηση με Επιλογή στη C

```
void selection(Item a[], int l, int r)
{
    int i, j;
    for (i = l; i < r; i++)
      { int min = i;
        for (j = i+1; j <= r; j++)
            if (less(a[j], a[min])) min = j;
        exch(a[i], a[min]);
      }
}
```

Ιδιότητες

- Στον εσωτερικό βρόχο του αλγόριθμου ταξινόμησης με επιλογή γίνεται απλώς μια σύγκριση του τρέχοντος στοιχείου με το μικρότερο στοιχείο που έχει βρεθεί μέχρι εκείνη τη στιγμή.
- Η εργασία μετακίνησης των στοιχείων βρίσκεται έξω από τον εσωτερικό βρόχο: κάθε αντιμετάθεση τοποθετεί ένα στοιχείο στην τελική του θέση, οπότε το **πλήθος των αντιμεταθέσεων** είναι $n-1$ όπου n είναι το πλήθος των στοιχείων του πίνακα (δεν χρειάζεται αντιμετάθεση για το τελευταίο στοιχείο).
- Ο χρόνος εκτέλεσης του αλγόριθμου ταξινόμησης με εισαγωγή εξαρτάται επίσης από το **πλήθος των συγκρίσεων**.
- Για κάθε i από 1 μέχρι $n-1$, γίνεται μια αντιμετάθεση και $n-i$ συγκρίσεις. Επομένως έχουμε

$$(n-1)+(n-2)+\cdots+2+1=\frac{n(n-1)}{2}$$

  συγκρίσεις.
- Επομένως η χρονική πολυπλοκότητα του αλγόριθμου ταξινόμησης με εισαγωγή είναι $\boldsymbol{O(n^2)}$ στη χειρότερη περίπτωση.
- Η πολυπλοκότητα είναι ίδια στη μέση περίπτωση.

Ιδιότητες

- Ένα μειονέκτημα της ταξινόμησης με επιλογή είναι ότι ο χρόνος εκτέλεσης εξαρτάται ελάχιστα από την έκταση της υπάρχουσας ταξινόμησης σε ένα πίνακα.
- Έτσι μπορεί να απαιτείται ο ίδιος σχεδόν χρόνος εκτέλεσης για ένα πίνακα που είναι ήδη ταξινομημένος, για ένα πίνακα του οποίου όλα τα κλειδιά είναι ίσα και για ένα πίνακα που έχει τυχαία ταξινόμηση.

Ιδιότητες

- Η ταξινόμηση με επιλογή είναι η προτιμώμενη μέθοδος για την ταξινόμηση πινάκων με πολύ μεγάλα στοιχεία (υλοποιημένα ως δομές `struct`) και μικρά κλειδιά.
- Για τέτοιες εφαρμογές, το κόστος της μετακίνησης δεδομένων είναι μεγαλύτερο από το κόστος εκτέλεσης των συγκρίσεων.
- Κανένας αλγόριθμος δεν μπορεί να ταξινομήσει ένα πίνακα με σημαντικά λιγότερες μετακινήσεις δεδομένων από τον αλγόριθμο ταξινόμησης με επιλογή.

Ταξινόμηση με Εισαγωγή

- Η μέθοδος της **ταξινόμησης με εισαγωγή (insertion sort)** λειτουργεί ως εξής.
- Διατρέχουμε τον πίνακα ξεκινώντας από την αρχή του.
- Εισάγουμε κάθε στοιχείο που επισκεπτόμαστε στη θέση που πρέπει να είναι ανάμεσα στα ήδη ταξινομημένα στοιχεία.
- Δημιουργούμε χώρο για να εισάγουμε το νέο στοιχείο μετακινώντας τα μεγαλύτερα στοιχεία μια θέση δεξιά, και στη συνέχεια εισάγουμε το νέο στοιχείο στην κενή θέση.

Παράδειγμα

A S O R T I N G E X A M P L E
A **S** O R T I N G E X A M P L E
A **O** S R T I N G E X A M P L E
A O **R** S T I N G E X A M P L E
A O R S **T** I N G E X A M P L E
A **I** O R S T N G E X A M P L E
A I **N** O R S T G E X A M P L E
A **G** I N O R S T E X A M P L E
A **E** G I N O R S T X A M P L E
A E G I N O R S T **X** A M P L E
A A E G I **M** N O R S T X P L E
A A E G I M N O **P** R S T X L E
A A E G I **L** M N O P R S T X E
A A E **E** G I L M N O P R S T X
A A E E G I L M N O P R S T X

Υλοποίηση σε C

```
void simpleinsertion(Item a[], int l, int r)
{
    int i, j;

    for (i = l+1; i <= r; i++)
      for (j = i; j > l; j--)
        compexch(a[j-1], a[j]);
}
```

Ερώτηση

- Μπορείτε να σκεφτείτε βελτιώσεις στον κώδικα της προηγούμενης συνάρτησης `simpleinsertion`;

Μια Καλύτερη Υλοποίηση

```
void insertion(Item a[], int l, int r)
{
    int i;

    for (i = l+1; i <= r; i++) compexch(a[l], a[i]);

    for (i = l+2; i <= r; i++)
      { int j = i; Item v = a[i];
        while (less(v, a[j-1]))
          { a[j] = a[j-1]; j--; }
        a[j] = v;
      }
}
```

Παρατηρήσεις για τη Συνάρτηση `insertion`

- Η συνάρτηση `insertion` αποτελεί **σημαντική βελτίωση** της συνάρτησης `simpleinsertion` επειδή:
  - Τοποθετεί αρχικά το μικρότερο στοιχείο του πίνακα στην πρώτη θέση έτσι ώστε αυτό το στοιχείο να μπορεί να παίξει το ρόλο φρουρού (sentinel). Έτσι δεν χρειαζόμαστε τον έλεγχο `j > l` της `simpleinsertion`.
  - Στον εσωτερικό βρόχο εκτελεί μια ανάθεση τιμής αντί για αντιμετάθεση. Με αυτές τις αναθέσεις, τα μεγαλύτερα στοιχεία μετακινούνται προς τα δεξιά.
  - Τερματίζει τον εσωτερικό βρόχο μόλις βρεθεί η σωστή θέση για το στοιχείο που εισάγεται.

Ιδιότητες

- Αντίθετα από την ταξινόμηση με επιλογή, ο χρόνος εκτέλεσης της ταξινόμησης με εισαγωγή **εξαρτάται κυρίως από την αρχική σειρά των κλειδιών** των δεδομένων εισόδου.
- Για παράδειγμα, αν ο πίνακας είναι μεγάλος και τα κλειδιά ήδη ταξινομημένα (ή σχεδόν ταξινομημένα), η ταξινόμηση με εισαγωγή είναι γρήγορη ενώ η ταξινόμηση με επιλογή είναι αργή.

Ιδιότητες

- Μπορούμε να αποδείξουμε ότι η ταξινόμηση με εισαγωγή έχει χρονική πολυπλοκότητα $O(n^2)$ στη χειρότερη και στη μέση περίπτωση.

Ταξινόμηση Φυσαλίδας

- Η **ταξινόμηση φυσαλίδας (bubblesort)** είναι ίσως η μέθοδος ταξινόμησης που οι περισσότεροι προγραμματιστές μαθαίνουν πρώτη.
- Κατά την ταξινόμηση φυσαλίδας, συνεχίζουμε να σαρώνουμε τον πίνακα, αντιμεταθέτοντας τα γειτονικά στοιχεία που βρίσκονται σε λάθος σειρά, μέχρι ο πίνακας τελικά να ταξινομηθεί πλήρως.
- Η ταξινόμηση φυσαλίδας **υλοποιείται εύκολα** όπως και η ταξινόμηση με επιλογή ή εισαγωγή.

Παράδειγμα

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	**A**	S	O	R	T	I	N	G	E	X	**E**	M	P	L
A	A	**E**	S	O	R	T	I	N	G	E	X	**L**	M	P
A	A	E	**E**	S	O	R	T	I	N	G	**L**	X	M	P
A	A	E	E	**G**	S	O	R	T	I	N	L	**M**	X	P
A	A	E	E	G	**I**	S	O	R	T	**L**	N	M	**P**	X
A	A	E	E	G	I	**L**	S	O	R	T	**M**	N	P	X
A	A	E	E	G	I	L	**M**	S	O	R	T	N	P	X
A	A	E	E	G	I	L	M	**N**	S	O	R	T	P	X
A	A	E	E	G	I	L	M	N	**O**	S	**P**	R	T	X
A	A	E	E	G	I	L	M	N	O	**P**	S	R	T	X
A	A	E	E	G	I	L	M	N	O	P	**R**	S	T	X
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Υλοποίηση σε C

```
void bubble(Item a[], int l, int r)
{
    int i, j;

    for (i = l; i < r; i++)
      for (j = r; j > i; j--)
        compexch(a[j-1], a[j]);
}
```

Παρατηρήσεις στη Συνάρτηση `bubble`

- Για κάθε `i` μεταξύ `l` και `r-1`, ο εσωτερικός βρόχος τοποθετεί το μικρότερο από τα στοιχεία `a[i], …, a[r]` στο `a[i]`, σαρώνοντας τα στοιχεία από δεξιά προς τα αριστερά και συγκρίνοντας και αντιμεταθέτοντας διαδοχικά στοιχεία.
- Σε όλες τις συγκρίσεις, το μικρότερο στοιχείο μετακινείται προς τα πάνω σαν φυσαλίδα.
- Όπως και στην ταξινόμηση με επιλογή, καθώς ο αριθμοδείκτης `i` κινείται από αριστερά προς τα δεξιά, τα στοιχεία αριστερά τους βρίσκονται στην τελική τους θέση στον πίνακα.

Ιδιότητες

- Ο αλγόριθμος της ταξινόμησης φυσαλίδας μπορεί να βελτιωθεί ελέγχοντας για την περίπτωση που δεν γίνεται καμία αντιμετάθεση σε κάποια από τις διελεύσεις που σημαίνει ότι ο πίνακας είναι ήδη ταξινομημένος και μπορούμε να βγούμε από τον εξωτερικό βρόχο.

Ιδιότητες

- Κατά την i-οστή διέλευση του εσωτερικού βρόχου, η ταξινόμηση φυσαλίδας κάνει $n - i$ συγκρίσεις.
- Στη χειρότερη περίπτωση που ο πίνακας είναι ταξινομημένος με αντίστροφη σειρά, κατά την i-οστή διέλευση του εσωτερικού βρόχου, η ταξινόμηση φυσαλίδας κάνει $n - i$ αντιμεταθέσεις.
- Οπότε στη χειρότερη περίπτωση έχουμε $O(n^2)$ συγκρίσεις και αντιμεταθέσεις, άρα η χρονική πολυπλοκότητα της ταξινόμησης φυσαλίδας είναι $\boldsymbol{O(n^2)}$ στη χειρότερη περίπτωση.
- Μπορούμε να αποδείξουμε ότι η πολυπλοκότητα είναι ίδια στη μέση περίπτωση.

Ο Αλγόριθμος Ταξινόμησης `shellsort`

- Η κεντρική ιδέα του αλγόριθμου shellsort είναι να αναδιατάξουμε τον δοσμένο πίνακα ώστε να αποκτήσει την παρακάτω ιδιότητα.
- Όταν παίρνουμε κάθε h-οστό στοιχείο του πίνακα ξεκινώντας οπουδήποτε, να έχουμε ένα ταξινομημένο υποπίνακα. Ένας τέτοιος πίνακας λέγεται ***h*-ταξινομημένος**.
- Δηλαδή, ένας h-ταξινομημένος πίνακας αποτελείται από ανεξάρτητους υποπίνακες που είναι ταξινομημένοι.
- Με την h-ταξινόμηση για μεγάλες τιμές του h, μπορούμε να μετακινήσουμε τα στοιχεία του πίνακα σε μεγάλες αποστάσεις και να διευκολύνουμε την h-ταξινόμηση για μικρές τιμές του h.
- Χρησιμοποιώντας τη διαδικασία της h-ταξινόμησης για οποιαδήποτε ακολουθία τιμών του h που καταλήγει στο 1, έχουμε ένα ταξινομημένο πίνακα.
- Κάθε h-ταξινόμηση γίνεται χρησιμοποιώντας τον αλγόριθμο **ταξινόμησης με εισαγωγή**.

Υλοποίηση σε C

```
void shellsort(Item a[], int l, int r)
{
    int i, j, h;

    for (h = 1; h <= (r-l)/9; h = 3*h+1) ;
    for ( ; h > 0; h /= 3)
      for (i = l+h; i <= r; i++)
        { int j = i; Item v = a[i];
          while (j >= l+h && less(v, a[j-h]))
            { a[j] = a[j-h]; j -= h; }
          a[j] = v;
        }
}
```

Παρατηρήσεις στη Συνάρτηση `shellsort`

- Η υλοποίηση που παρουσιάσαμε χρησιμοποιεί την εξής ακολουθία τιμών του h: 1, 4, 13, 40, 121, …
- Η ακολουθία αυτή προτάθηκε από τον Knuth το 1969, είναι εύκολο να υπολογιστεί και οδηγεί σε μια σχετικά αποδοτική ταξινόμηση για μεγάλους πίνακες.
- Γενικά, θέλουμε να χρησιμοποιούμε ακολουθίες τιμών που μειώνονται με γεωμετρικό τρόπο με συνέπεια το πλήθος των τιμών του h που θα χρησιμοποιήσουμε να είναι λογαριθμικό στο μέγεθος του πίνακα.

Ιδιότητες

- Η μαθηματική ανάλυση της μεθόδου ταξινόμησης `shellsort` είναι αρκετά δύσκολη. Μια ενδιαφέρουσα ιδιότητα είναι η παρακάτω.
- Η ταξινόμηση `shellsort` εκτελεί λιγότερες από $\boldsymbol{O}(\boldsymbol{n}^{\frac{\boldsymbol{3}}{\boldsymbol{2}}})$ συγκρίσεις για την ακολουθία τιμών του h που δώσαμε.

Μελέτη

- Robert Sedgewick. *Αλγόριθμοι σε C*. 3η Αμερικανική Έκδοση. Εκδόσεις Κλειδάριθμος.
  - Κεφ. 6