Αλγόριθμοι Ταξινόμησης – Μέρος 2

Μανόλης Κουμπαράκης

Προχωρημένοι Αλγόριθμοι Ταξινόμησης

- Στη συνέχεια θα παρουσιάσουμε τρείς προχωρημένους αλγόριθμους ταξινόμησης: **treesort**, **quicksort** και **mergesort**.

`treesort`

- Στον αλγόριθμο ταξινόμησης `treesort` παίρνουμε τα στοιχεία του μη ταξινομημένου πίνακα, και τα εισάγουμε ένα προς ένα σε ένα, αρχικά κενό, **δένδρο δυαδικής αναζήτησης**.
- Μετά κάνουμε μια **ενδοδιατεταγμένη (inorder)** διάσχιση του δένδρου η οποία μας δίνει τα στοιχεία σε αύξουσα σειρά, και τα αντιγράφουμε πίσω στον πίνακα ταξινομημένα.

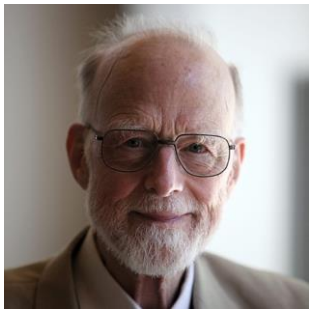
Ιδιότητες

- Μπορούμε να αποδείξουμε ότι ο αλγόριθμος ταξινόμησης `treesort` έχει πολυπλοκότητα χρόνου $\boldsymbol{O(n \log n)}$ **στη μέση περίπτωση**.
- Στη **χειρότερη περίπτωση** μπορεί να πάρει χρόνο $\boldsymbol{O(n^2)}$ όταν ο πίνακας εισόδου είναι ήδη ταξινομημένος οπότε το δένδρο δυαδικής αναζήτησης που προκύπτει είναι αλυσίδα.
- Αν χρησιμοποιήσουμε **AVL, (2,4) ή κόκκινο-μαύρο δένδρο**, η πολυπλοκότητα χειρότερης περίπτωσης θα είναι $\boldsymbol{O(n \log n)}$.

Ιδιότητες

- Ο αλγόριθμος `treesort` είναι πολύ χρήσιμος όταν τα στοιχεία που ταξινομούνται γίνονται διαθέσιμα σταδιακά.
- Επίσης είναι πολύ χρήσιμος όταν, αφού ταξινομήσουμε τα στοιχεία, θέλουμε να έχουμε τη δυνατότητα να κάνουμε εισαγωγές, διαγραφές και αναζητήσεις.
- Όλες αυτές οι πράξεις μπορούν να υλοποιηθούν σε $O(\log n)$ χρόνο στη μέση περίπτωση για απλά δένδρα δυαδικής αναζήτησης, και σε $O(\log n)$ χρόνο στη χειρότερη περίπτωση για δένδρα AVL, (2,4) ή κόκκινα-μαύρα.

Ο Αλγόριθμος quicksort



- Ο αλγόριθμος **quicksort** είναι ο αλγόριθμος ταξινόμησης που μάλλον χρησιμοποιείται περισσότερο από οποιονδήποτε άλλον.
- Ανακαλύφθηκε το 1960 από τον C.A.R. Hoare.
- Έχει μελετηθεί από πολλούς ερευνητές και υπάρχουν πολλές βελτιώσεις του, που τον κάνουν κατάλληλο για μια μεγάλη γκάμα εφαρμογών.

Ο Βασικός Αλγόριθμος

- Ο αλγόριθμος quicksort είναι μια μέθοδος ταξινόμησης του τύπου **«διαίρει και βασίλευε» (divide and conquer).**
- Λειτουργεί **διαμερίζοντας** ένα πίνακα σε δύο μέρη και **ταξινομώντας** αυτά τα μέρη, το ένα ανεξάρτητα από το άλλο.
- Η ακριβής θέση της διαμέρισης εξαρτάται από την αρχική σειρά των στοιχείων του πίνακα εισόδου.
- Μετά **ενώνει** τα δύο ταξινομημένα μέρη για να προκύψει ο ταξινομημένος πίνακας.

Η Διαμέριση

- Κατά τη διαμέριση, ο πίνακας αναδιατάσσεται έτσι ώστε να ισχύουν οι τρείς παρακάτω συνθήκες:
  - Για κάποιο `i`, το στοιχείο `a[i]` βρίσκεται στην τελική του θέση μέσα στον πίνακα.
  - Κανένα από τα στοιχεία `a[1],…,a[i-1]` δεν είναι μεγαλύτερο από το `a[i]`.
  - Κανένα από τα στοιχεία `a[i+1],…,a[r]` δεν είναι μικρότερο από το `a[i]`.

Ο Βασικός Αλγόριθμος

- Φτάνουμε στην πλήρη ταξινόμηση κάνοντας πρώτα τη διαμέριση και μετά εφαρμόζοντας αναδρομικά τη μέθοδο στους δυο υποπίνακες που προκύπτουν.
- Επειδή η διαδικασία της διαμέρισης τοποθετεί πάντα τουλάχιστον ένα στοιχείο στη θέση του, δεν είναι δύσκολο να αναπτύξουμε μια τυπική απόδειξη με επαγωγή ότι η αναδρομική μέθοδος καταλήγει σε σωστή ταξινόμηση.

Κώδικας σε C

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

    if (r <= l) return;
    i = partition(a, l, r);
    quicksort(a, l, i-1);
    quicksort(a, i+1, r);

}
```

Κώδικας σε C

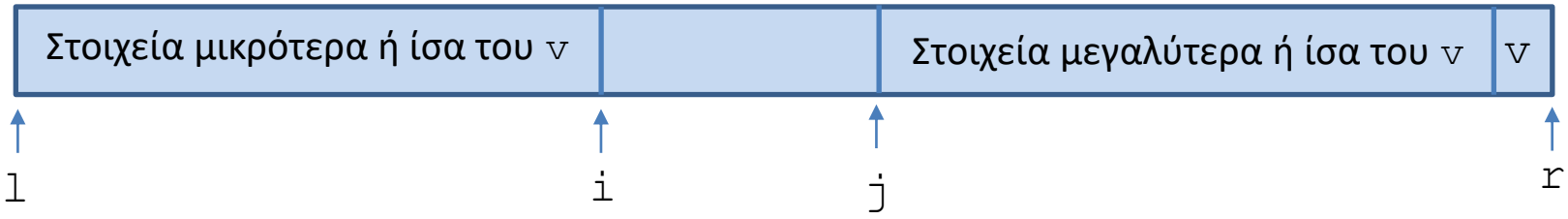
```
int partition(Item a[], int l, int r)
{
    int i = l-1, j = r; Item v = a[r];

    for (;;)
      {
        while (less(a[++i], v)) ;
        while (less(v, a[--j])) if (j == l) break;
        if (i >= j) break;
        exch(a[i], a[j]);
      }
    exch(a[i], a[r]);
    return i;
}
```

Παρατηρήσεις για τη Συνάρτηση `quicksort`

- Κάθε κλήση `quicksort(a, l, r)` ταξινομεί τον υποπίνακα `a[l:r]` του πίνακα `a`.
- Αν ο πίνακας `a` έχει ένα ή λιγότερα στοιχεία, ο αλγόριθμος δεν κάνει τίποτε.
- Αλλιώς, καλείται η συνάρτηση `partition`, η οποία τοποθετεί το στοιχείο `a[i]` στη θέση του (το `i` είναι μεταξύ `l` και `r`) και αναδιατάσσει τα υπόλοιπα στοιχεία ώστε οι αναδρομικές κλήσεις που ακολουθούν να ολοκληρώσουν την ταξινόμηση.

Παρατηρήσεις για τη Συνάρτηση `partition`

- Το **στοιχείο διαμέρισης** επιλέγεται αυθαίρετα να είναι το δεξιότερο στοιχείο του πίνακα `a[r]`.
- Στη μεταβλητή `v` τηρείται η τιμή του στοιχείου διαμέρισης `a[r]`, ενώ τα `i` και `j` είναι ο αριστερός και δεξιός δείκτης σάρωσης αντίστοιχα.
- Ο βρόχος διαμέρισης αυξάνει το `i` και μειώνει το `j`, ενώ διατηρεί αναλλοίωτη την ιδιότητα ότι κανένα στοιχείο στα αριστερά του `i` δεν είναι μεγαλύτερο από το `v` και κανένα στοιχείο στα δεξιά του `j` δεν είναι μικρότερο από το `v`.
- Ο βρόχος διαμέρισης σαρώνει τον πίνακα από το αριστερό του άκρο μέχρι να βρεθεί κάποιο στοιχείο μεγαλύτερο ή ίσο από το στοιχείο διαμέρισης. Μετά σαρώνει τον πίνακα από το δεξιό του άκρο μέχρι να βρεθεί κάποιο στοιχείο μικρότερο ή ίσο από το στοιχείο διαμέρισης. Τα δύο στοιχεία στα οποία θα σταματήσουν οι σαρώσεις είναι προφανώς εκτός θέσης στον τελικό διαμερισμένο πίνακα, οπότε τα αντιμεταθέτουμε.



Παρατηρήσεις για τη Συνάρτηση `partition`

- Όταν συναντηθούν οι δείκτες, ολοκληρώνουμε τη διαμέριση αντιμεταθέτοντας το `a[i]` με το `a[r]`, μια πράξη που έχει σαν συνέπεια να τοποθετηθεί το `v` στη θέση `a[i]` χωρίς να υπάρχουν μικρότερα στοιχεία από αυτό στα δεξιά του και μεγαλύτερα στα αριστερά του.
- Ο βρόχος διαμέρισης υλοποιείται ως ατέρμων βρόχος, με μια εντολή `break` στο σημείο που συναντώνται οι δύο δείκτες.
- Ο έλεγχος `j == l` αφορά την περίπτωση που το στοιχείο διαμέρισης είναι το μικρότερο στοιχείο του πίνακα.

Παράδειγμα `partition`

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

Παράδειγμα `quicksort`

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

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

N P O
O P
P

S T X
T **X**
T

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

Ερωτήσεις

- Πως λειτουργεί ο αλγόριθμος quicksort αν κληθεί με είσοδο ένα ταξινομημένο πίνακα;
- Πως λειτουργεί ο αλγόριθμος quicksort αν κληθεί με είσοδο ένα πίνακα με στοιχεία ταξινομημένα σε αντίθετη σειρά;

Ιδιότητες

- Αν ο αλγόριθμος quicksort κληθεί με είσοδο ένα πίνακα μεγέθους n που είναι ήδη ταξινομημένος, όλες οι διαμερίσεις εκφυλίζονται και το πρόγραμμα καλεί το εαυτό του n φορές, αφαιρώντας μόνο ένα στοιχείο σε κάθε κλήση.
- Το πλήθος των συγκρίσεων που γίνονται είναι:

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

$$= O(n^2)$$

- Συνεπώς, ο αλγόριθμος quicksort έχει πολυπλοκότητα $\boldsymbol{O(n^2)}$ **στη χειρότερη περίπτωση**.

Ιδιότητες

- Ο αλγόριθμος quicksort κάνει $O(n \log n)$ συγκρίσεις κατά μέσο όρο.
- Συνεπώς, η χρονική πολυπλοκότητα **μέσης περίπτωσης** του αλγόριθμου quicksort είναι $\boldsymbol{O(n \log n)}$.

Βελτιώσεις

- Στη βιβλιογραφία υπάρχουν διάφορες βελτιώσεις του quicksort για τα εξής θέματα:
  - Πώς να ταξινομούνται μικροί υποπίνακες
  - Πώς να επιλέγεται καλύτερα το στοιχείο διαμέρισης ώστε να διαιρεί τον πίνακα κοντά στο μέσο
  - Πώς να αντιμετωπίζονται τα διπλά κλειδιά

Ταξινόμηση με Συγχώνευση

- Θα μελετήσουμε τώρα μια τεχνική ταξινόμησης που ανήκει επίσης στην κατηγορία των τεχνικών **«διαίρει και βασίλευε»** και είναι συμπληρωματική του αλγόριθμου quicksort.
- Η **συγχώνευση (merging)** είναι μια διαδικασία που συνδυάζει δύο ταξινομημένους πίνακες σε ένα μεγαλύτερο ταξινομημένο πίνακα.
- Ο αλγόριθμος mergesort που θα παρουσιάσουμε είναι συμπληρωματικός του quicksort.
- Ο quicksort αναδιατάσσει ένα πίνακα σε δύο μέρη έτσι ώστε όταν ταξινομούνται τα δύο μέρη, να ταξινομείται ολόκληρος ο πίνακας.
- Ο mergesort μοιράζει ένα πίνακα σε δύο μέρη προς ταξινόμηση και στη συνέχεια συνδυάζει τα ταξινομημένα μέρη για να σχηματιστεί ο συνολικός ταξινομημένος πίνακας.
- Ο quicksort υλοποιείται με μια διαδικασία **επιλογής** ακολουθούμενη από δύο αναδρομικές κλήσεις.
- Ο mergesort υλοποιείται με δύο αναδρομικές κλήσεις που ακολουθούνται από μια διαδικασία **συγχώνευσης**.

Διμερής Συγχώνευση

- Αν έχουμε δύο ταξινομημένους πίνακες, μπορούμε να τους συνδυάσουμε σε ένα τρίτο ταξινομημένο πίνακα ως εξής.
- Παρακολουθούμε το μικρότερο στοιχείο κάθε πίνακα, εκτελούμε ένα βρόχο στον οποίο το μικρότερο από τα δύο στοιχεία που είναι μικρότερα στους δύο πίνακες τοποθετείται στον τρίτο πίνακα, και συνεχίζουμε με αυτό τον τρόπο μέχρι να εξαντληθούν τα στοιχεία των δύο πινάκων.
- Η διαδικασία αυτή λέγεται **διμερής συγχώνευση (two-way merging).**

Υλοποίηση σε C

- Ας υποθέσουμε ότι έχουμε δύο ξένους μεταξύ τους ταξινομημένους πίνακες `a[0], a[1], …, a[N-1]` και `b[0], b[1], …, b[M-1]` που περιέχουν ακεραίους, τους οποίους θέλουμε να συγχωνεύσουμε σε ένα τρίτο πίνακα `c[0], c[1], …, c[N+M-1]`.
- Η προφανής στρατηγική η οποία υλοποιείται εύκολα είναι να επιλέγουμε διαδοχικά για τον πίνακα `c` το μικρότερο στοιχείο που απομένει στους `a` και `b`.

Υλοποίηση σε C

```
void mergeAB(Item c[], Item a[], int N, Item b[],
int M )
{
    int i, j, k;
    for (i = 0, j = 0, k = 0; k < N+M; k++)
      {
        if (i == N) { c[k] = b[j++]; continue; }
        if (j == M) { c[k] = a[i++]; continue; }
        c[k] = (less(a[i], b[j])) ? a[i++] : b[j++];
      }
}
```

Ερώτηση

- Μπορούμε να αποφύγουμε τη χρήση του τρίτου πίνακα `c` κάνοντας **επιτόπου συγχώνευση**;
- Για παράδειγμα, μπορούμε να συγχωνεύσουμε τους πίνακες `a[l], …, a[m]` και `a[m+1], …, a[r]` μετακινώντας τα στοιχεία στις κατάλληλες θέσεις του πίνακα `a[l], …, a[r];`

Απάντηση

- Η απάντηση είναι ναι, όμως οι λύσεις που είναι γνωστές γι αυτό το πρόβλημα είναι αρκετά πολύπλοκες και δεν θα τις παρουσιάσουμε.

Αφαιρετική Επιτόπου Συγχώνευση

- Θα χρησιμοποιήσουμε την **αφαίρεση (abstraction)** της επιτόπου συγχώνευσης στον αλγόριθμος ταξινόμησης που θα μελετήσουμε ως εξής.
- Θα αντιγράψουμε αρχικά τους δύο πίνακες σε ένα βοηθητικό πίνακα `aux` και μετά θα τους συγχωνεύσουμε προσπαθώντας να αποφύγουμε μια αδυναμία του βασικού αλγόριθμου συγχώνευσης που υλοποιεί η συνάρτηση `mergeAB`.

Ερώτηση

- Στη συνάρτηση `mergeAB`, πώς μπορούμε να αποφύγουμε τους δύο ελέγχους (εντολές `if`) για τον προσδιορισμό του πότε έχουμε φτάσει στα άκρα των δύο πινάκων εισόδου;
- Η παρουσία αυτών των δύο ελέγχων, που κατά την εκτέλεση του αλγόριθμου συνήθως αποτυγχάνουν, είναι μια αδυναμία της συνάρτησης `mergeAB`.

Απάντηση

- Μπορούμε να προσθέσουμε στα άκρα των δύο πινάκων `a` και `b` δύο **τιμές φρουρούς (sentinels)** που είναι μεγαλύτερες από τις τιμές όλων των άλλων κλειδιών.
- Τότε οι έλεγχοι μπορούν να αφαιρεθούν επειδή όταν εξαντληθούν τα στοιχεία του πίνακα `a` (αντίστοιχα, `b`), η τιμή φρουρός θα φροντίσει ώστε τα επόμενα στοιχεία του πίνακα `c` να ληφθούν από τον `b` (αντίστοιχα, `a`), μέχρι να ολοκληρωθεί η συγχώνευση.
- Ωστόσο, δεν είναι πάντοτε δυνατό να χρησιμοποιούμε τιμές φρουρούς, επειδή μπορεί να μην είναι εύκολο να γνωρίζουμε το μεγαλύτερο κλειδί.

Αποδοτική Συγχώνευση Χωρίς Τιμές Φρουρούς

- Θα υλοποιήσουμε την αφαίρεση της επιτόπου συγχώνευσης χρησιμοποιώντας την ακόλουθη μέθοδο.
- Θα χρησιμοποιήσουμε ένα βοηθητικό πίνακα `aux[r]` ο οποίος θα περιέχει στην αρχή του τον ταξινομημένο πίνακα `a[l], …, a[m]` και στη συνέχεια τον πίνακα `a[m+1], …, a[r]` **ταξινομημένο σε αντίστροφη σειρά**.
- Μ’ αυτό τον τρόπο, όταν κάνουμε συγχώνευση, ο δείκτης που διατρέχει τον δεύτερο πίνακα μετακινείται από τα δεξιά προς τα αριστερά. Αυτή η διάταξη κάνει το μεγαλύτερο στοιχείο των δύο πινάκων (ανεξάρτητα από τον πίνακα που βρίσκεται) να λειτουργεί ως τιμή φρουρός για τον άλλο πίνακα.

Παράδειγμα Συγχώνευσης των Πινάκων A R S T και G I N

A R S T G I N

A R S T N I G

R S T N I G A

R S T N I A G

R S T N A G I

S T A G I N R

T A G I N R S

A G I N R S T

Διτονικές Ακολουθίες

- Μια ακολουθία κλειδιών λέγεται **διτονική (bitonic)** αν πρώτα αυξάνεται και μετά μειώνεται ή αντίστροφα.
- Η προηγούμενη μέθοδος συγχώνευσης δημιουργεί μια διτονική ακολουθία π.χ.,

  A R S T N I G.
- Η ταξινόμηση μιας διτονικής ακολουθίας είναι ισοδύναμη με συγχώνευση.

Υλοποίηση σε C

```
#define maxN 10000

Item aux[maxN];

merge(Item a[], int l, int m, int r)
{
   int i, j, k;
   for (i = m+1; i > l; i--) aux[i-1] = a[i-1];
   for (j = m; j < r; j++) aux[r+m-j] = a[j+1];
   for (k = l; k <= r; k++)
      if (less(aux[i], aux[j]))
         a[k] = aux[i++];
      else
         a[k] = aux[j--];
}
```

Παρατηρήσεις για την `merge`

- Αυτό το πρόγραμμα εκτελεί συγχώνευση χωρίς να χρησιμοποιεί τιμές φρουρούς, αντιγράφοντας το δεύτερο πίνακα στον βοηθητικό πίνακα aux (οργανώνοντας τον πίνακα `aux` σαν διτονική ακολουθία).
- Ο πρώτος βρόχος `for` μετακινεί τον πρώτο πίνακα και αφήνει το `i` να δείχνει στο `l`, έτοιμο για την εκκίνηση της συγχώνευσης.
- Ο δεύτερος βρόχος `for` μετακινεί το δεύτερο πίνακα και αφήνει το `j` να δείχνει στο `r`.
- Ο τρίτος βρόχος `for` εκτελεί τη συγχώνευση.

Αναλυτική Ταξινόμηση με Συγχώνευση

- Αν έχουμε ένα αλγόριθμο συγχώνευσης, δεν είναι δύσκολο να τον χρησιμοποιήσουμε ως βάση για μια αναδρομική διαδικασία ταξινόμησης.
- Για να ταξινομήσουμε ένα πίνακα, τον διαιρούμε στη μέση, ταξινομούμε με αναδρομή τα δύο μισά, και έπειτα τα συγχωνεύουμε.
- Αυτή η μέθοδος είναι ένα κλασσικό παράδειγμα της τεχνικής **«διαίρει και βασίλευε».**

Ο Αναλυτικός Αλγόριθμος mergesort

```
void mergesort(Item a[], int l, int r)
{
    int m = (r+l)/2;
    if (r <= l) return;

    mergesort(a, l, m);
    mergesort(a, m+1, r);
    merge(a, l, m, r);
}
```

Ιδιότητες

- Ο αλγόριθμος `mergesort` ταξινομεί ένα πίνακα μεγέθους n σε $\boldsymbol{O(n \log n)}$ χρόνο στη χειρότερη περίπτωση χρησιμοποιώντας $\boldsymbol{O(n)}$ βοηθητικό χώρο.

Βελτιώσεις

- Η βιβλιογραφία μας παρέχει βελτιώσεις του `mergesort` ώστε να ταξινομεί αποδοτικά μικρούς πίνακες, να μην χρησιμοποιεί βοηθητικό πίνακα κλπ.

Μελέτη

- Robert Sedgewick. Αλγόριθμοι σε C. 3η Αμερικανική Έκδοση. Εκδόσεις Κλειδάριθμος.
  - Κεφ. 7 και 8