Δομές Δεδομένων και Αλγόριθμοι

Χρήστος Γκόγκος

ΤΕΙ Ηπείρου

Χειμερινό Εξάμηνο 2014-2015
Παρουσίαση 22. Counting sort, bucket sort και radix sort

Ιδιότητες αλγορίθμων ταξινόμησης

- ευστάθεια (stable sort): Ένας αλγόριθμος ταξινόμησης είναι stable όταν η σχετική διάταξη στοιχείων με την ίδια τιμή κλειδιού δεν αλλάζει κατά την ταξινόμηση (π.χ. INSERTION SORT, BUBBLE SORT, MERGE SORT, QUICK SORT).
- επί τόπου (in-place, in-situ): Ένας αλγόριθμος ταξινόμησης είναι in-place όταν δεν χρησιμοποιεί επιπλέον βοηθητικό χώρο για την ταξινόμηση (π.χ. INSERTION SORT, SELECTION SORT, BUBBLE SORT).
- προσαρμοστικότητα (adaptive): Ένας αλγόριθμος ταξινόμησης είναι adaptive όταν εκμεταλλεύεται το γεγονός ότι τμήματα της ακολουθίας εισόδου είναι ήδη ταξινομημένα και ολοκληρώνει την ταξινόμηση της ακολουθίας ταχύτερα (π.χ. INSERTION SORT, SELECTION SORT, BUBBLE SORT).

Οποιοσδήποτε αλγόριθμος ταξινόμησης μπορεί να γίνει stable προσθέτοντας το δείκτη της θέσης του στην αρχή του κλειδιού.

Λεξικογραφική σειρά

- Η λεξικογραφική σειρά ονομάζεται και διάταξη λεξικού καθώς στη περίπτωση των αλφαριθμητικών πρόκειται για τη σειρά με την οποία εμφανίζονται οι λέξεις σε ένα λεξικό (π.χ. Μηχανή < Μηχανικός καθώς η < ι στην αλφαβητική σειρά των γραμμάτων).
- Η λεξικογραφική σειρά μπορεί να εφαρμοστεί εκτός από τα αλφαριθμητικά γενικότερα σε πλειάδες τιμών (tuples). Για παράδειγμα οι 2 πλειάδες (a1, a2, a3, ... an) και (b1, b2, b3, ..., bn) μπορούν να συγκριθούν λεξικογραφικά ως εξής: πρώτα θα συγκριθούν τα στοιχεία που είναι στη θέση 1 (a1 και b1) και αν αυτά είναι ίσα θα συγκριθούν τα στοιχεία στη θέση 2 (a2 και b2) κοκ.

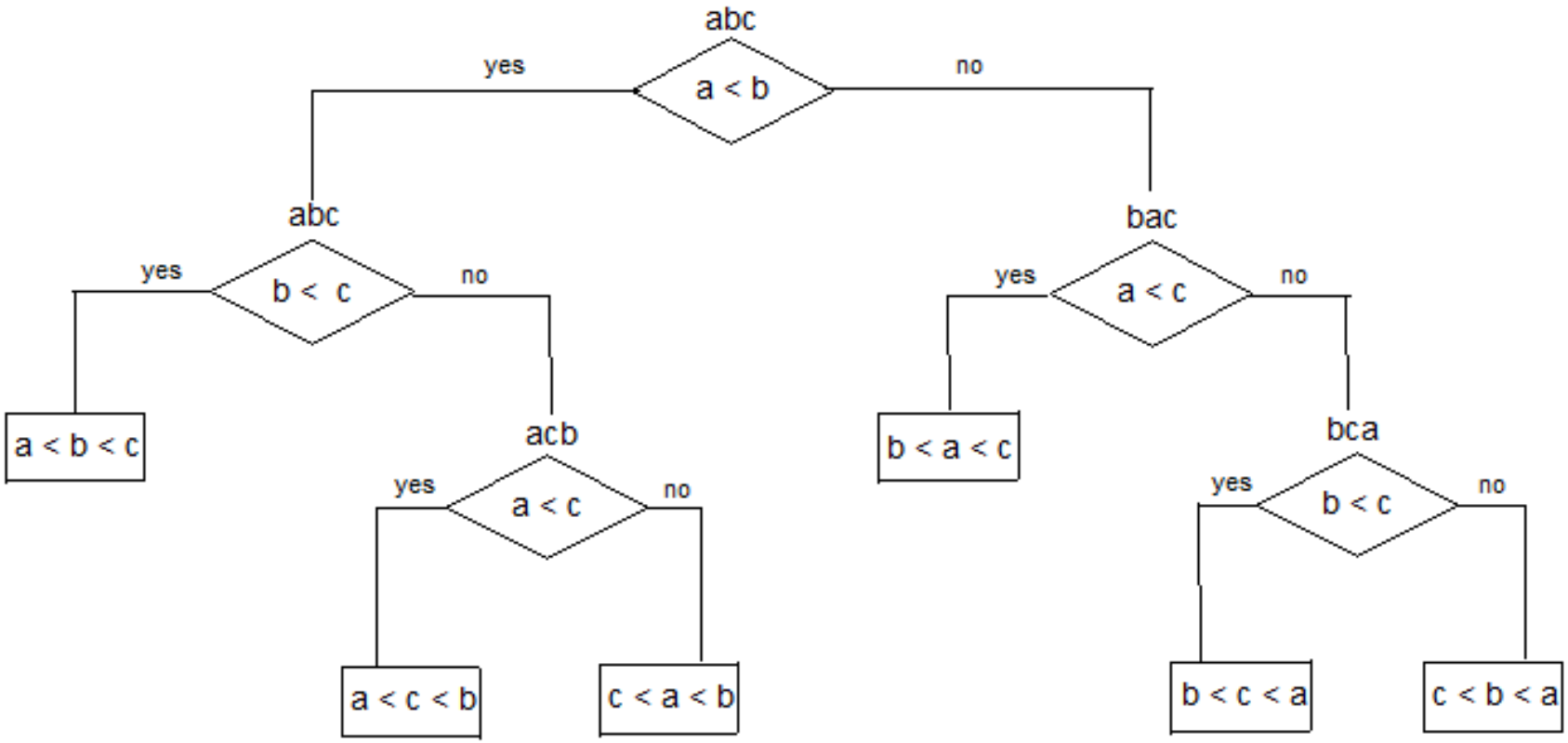
Η σύγκριση αλφαριθμητικών γίνεται αυτόματα χωρίς την παρέμβαση του προγραμματιστή καθώς οι γλώσσες προγραμματισμού είναι σε θέση να συγκρίνουν απευθείας αλφαριθμητικές τιμές. Για παράδειγμα στη C++ η συνάρτηση σύγκρισης αλφαριθμητικών είναι η: **int strcmp (const char * str1, const char * str2);**
η οποία δέχεται 2 αλφαριθμητικά ως παραμέτρους και επιστρέφει αρνητική τιμή όταν το str1 είναι λεξικογραφικά μικρότερο από το str2, μηδέν όταν είναι ίδια και θετική τιμή όταν το str2 είναι λεξικογραφικά μεγαλύτερο του str1.

Όριο απόδοσης για αλγορίθμους ταξινόμησης με σύγκριση στοιχείων

Οι αλγόριθμοι που στηρίζονται σε σύγκριση στοιχείων όπως οι QuickSort, MergeSort, HeapSort, InsertionSort κ.α. έχουν ως όριο απόδοσης $O(n \log n)$.
Ωστόσο, αν τα δεδομένα έχουν κάποια ιδιαίτερα χαρακτηριστικά τότε η ταξινόμηση μπορεί να γίνει ταχύτερα. Για παράδειγμα, η ταξινόμηση n ακεραίων τιμών που κυμαίνονται σε εύρος τιμών $[0, k)$ με k πολύ μικρότερο του n μπορεί να γίνει σε χρόνο $O(n)$ με τον αλγόριθμο Counting Sort.

Δένδρο απόφασης ως μοντέλο για την αναπαράσταση αλγορίθμων ταξινόμησης με συγκρίσεις στοιχείων

Δυαδικό δένδρο απόφασης για την ταξινόμηση 3 στοιχείων με την INSERTION SORT



A. Levitin Introduction to the Design and Analysis of Algorithms 2nd ed. (Pearson)

- Η ρίζα του δένδρου αντιστοιχεί στην αρχική κατάσταση των δεδομένων (δηλαδή μη ταξινομημένα δεδομένα) ενώ στα φύλλα του δένδρου υπάρχουν οι καταστάσεις που αντιστοιχούν στην ταξινομημένη ακολουθία.
- Κάθε κόμβος του δένδρου αναπαριστά μια κατάσταση στην οποία βρίσκονται τα δεδομένα.Για κάθε κόμβο που δεν είναι φύλλο μπορούν να ληφθούν 2 αποφάσεις που θα οδηγήσουν σε 2 νέους κόμβους.
- Ο χρόνος χειρότερης περίπτωσης ενός αλγορίθμου ταξινόμησης που αναπαρίσταται με ένα δυαδικό δένδρο απόφασης αντιστοιχεί στο ύψος του δένδρου.

Ένας αλγόριθμος ταξινόμησης δημιουργεί μια μετάθεση (αλλαγή της θέσεων) των δεδομένων εισόδου του. Αν το πλήθος των δεδομένων εισόδου είναι n, ο αριθμός των πιθανών μεταθέσεων είναι $n!$. Το δένδρο απόφασης που αντιστοιχεί στον αλγόριθμο ταξινόμησης θα πρέπει να έχει 1 φύλλο για κάθε μια από τις μεταθέσεις των n στοιχείων, άρα $n!$ φύλλα. Το ελάχιστο ύψος ενός δυαδικού δένδρου με $n!$ φύλλα είναι $\log n! \approx n \log n$ (όπως προκύπτει από τον τύπο του Stirling). Συνεπώς η απόδοση ενός αλγορίθμου ταξινόμησης που χρησιμοποιεί συγκρίσεις στοιχείων έχει ως όριο απόδοσης το $O(n \log n)$.

Counting Sort

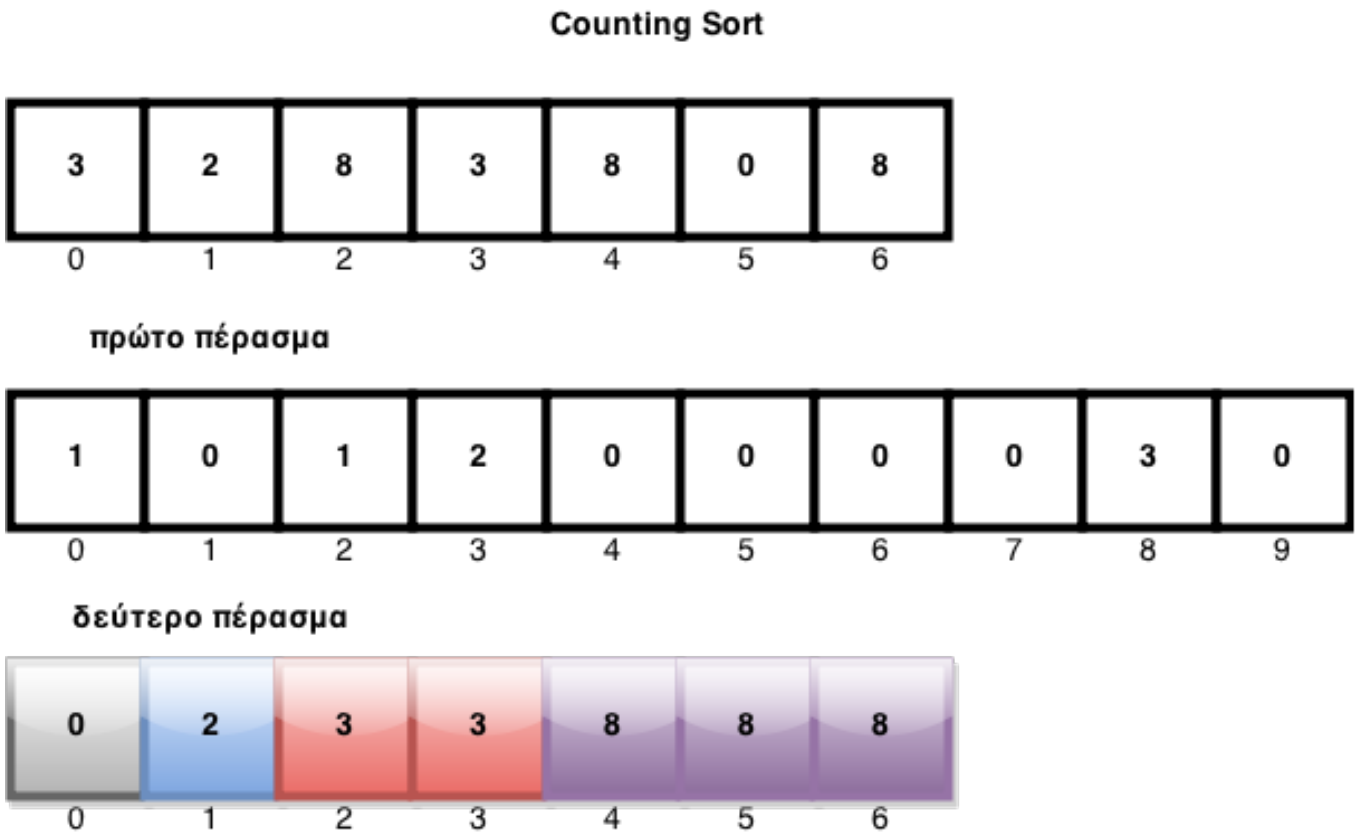
Ο αλγόριθμος Counting Sort μετρά πόσες φορές παρατηρείται η κάθε διακριτή τιμή της ακολουθίας εισόδου (δημιουργεί έναν πίνακα συχνοτήτων). Στη συνέχεια, από το μικρότερο προς το μεγαλύτερο δείκτη του πίνακα συχνοτήτων καταγράφει στον αρχικό πίνακα την τιμή του δείκτη *i* τόσες φορές όσες η αριθμητική τιμή που περιέχεται στον πίνακα συχνοτήτων στη θέση *i*.

```
void count_sort(int a[], int
     n, int k){
    int i;
    int *b=new int[k]{0};
    for(i=0;i<n;i++)
        b[a[i]]++;

    int j=0;
    for(i=0;i<k;i++)
        while (b[i]>0){
            b[i]--;
            a[j]=i;
            j++;
        }
    delete [] b;
}
```

Παράδειγμα εκτέλεσης του Counting Sort

Έστω ένα σύνολο τιμών που γνωρίζουμε ότι μπορεί να λάβει τιμές από 0 έως 9 και έστω ότι σε ένα συγκεκριμένο στιγμιότυπο εισόδου οι τιμές αυτές είναι: 3,2,8,3,8,0,8



Απόδοση του Counting Sort

Ο αλγόριθμος Counting Sort προτάθηκε από τον Harold Seward το 1954. Δεν στηρίζεται σε σύγκριση αλλά σε απαρίθμηση των τιμών. Έχει πολυπλοκότητα $O(n + k)$ όπου k είναι το εύρος των διακριτών τιμών και n είναι το μέγεθος της εισόδου.

Για να μπορεί να εφαρμοστεί αποδοτικά θα πρέπει οι τιμές εισόδου να κυμαίνονται σε ένα γνωστό (και μικρό) εύρος τιμών και ο πίνακας συχνοτήτων των τιμών να μπορεί να υπολογιστεί γρήγορα.

Bucket sort

Ο αλγόριθμος bucket sort είναι χρήσιμος όταν τα δεδομένα είναι σχετικά ομοιόμορφα κατανεμημένα σε ένα εύρος τιμών. Δημιουργεί n αριθμημένους κάδους (buckets, bins) και τοποθετεί κάθε στοιχείο σε ένα από τους κάδους ανάλογα με την τιμή του. Ταξινομεί τα στοιχεία σε κάθε κάδο ξεχωριστά χρησιμοποιώντας κάποιον αλγόριθμο ταξινόμησης (π.χ. Insertion Sort) και στη συνέχεια συνενώνει τα περιεχόμενα των κάδων έτσι ώστε να δημιουργηθεί η τελική ταξινομημένη ακολουθία.
Υπάρχει και έκδοση του bucket sort που λειτουργεί αναδρομικά δηλαδή τα στοιχεία κατανέμονται αρχικά σε κάδους και σε κάθε κάδο κατανέμονται εκ νέου τα στοιχεία του σε άλλους κάδους μέχρι το μέγεθος των κάδων να γίνει μικρό.

Για κάθε 2 κάδους A και B θα πρέπει να ισχύει ότι όλα τα στοιχεία του κάδου A είναι μικρότερα από όλα τα στοιχεία του κάδου B ή το αντίστροφο.

Μια απλή υλοποίηση του αλγορίθμου Bucket sort

```
// bn = number of buckets
// assume values in array are in [0,1)
void bucket_sort(float arr [], int n, int bn) {
    vector<float> b[bn];

    for (int i = 0; i < n; i++) {
        int bi = bn * arr[i];
        b[bi].push_back(arr[i]);
    }

    for (int i = 0; i < bn; i++)
        sort(b[i].begin(), b[i].end());

    int index = 0;
    for (int i = 0; i < bn; i++)
        for (unsigned int j = 0; j <
             b[i].size(); j++)
            arr[index++] = b[i][j];
}
...
float a[] = {0.897, 0.565, 0.656, 0.1234,
     0.665, 0.3434};
bucket_sort(a, 6, 3);
// 0.1234 0.3434 0.565 0.656 0.665 0.897
```

Προσαρμοσμένο από το `http://www.geeksforgeeks.org/bucket-sort-2/`

- 3 * 0.897 =2.691 ==> bucket 2
- 3 * 0.565 =1.695 ==> bucket 1
- 3 * 0.656 =1.968 ==> bucket 1
- 3 * 0.123 =0.370 ==> bucket 0
- 3 * 0.665 =1.995 ==> bucket 1
- 3 * 0.343 =1.030 ==> bucket 1

Απόδοση του Bucket Sort

Η πολυπλοκότητα του Bucket Sort είναι $O(mC(n/m))$ όπου n είναι το πλήθος των στοιχείων εισόδου, m είναι ο αριθμός των κάδων και $C(x)$ είναι η πολυπλοκότητα του αλγορίθμου ταξινόμησης που θα χρησιμοποιηθεί για την ταξινόμηση των στοιχείων κάθε κάδου.

Radix sort

Η ταξινόμηση radix sort είναι ταξινόμηση ψηφίο προς ψηφίο ξεκινώντας από τα λιγότερο σημαντικά ψηφία και συνεχίζοντας προς τα περισσότερο σημαντικά. Χρησιμοποιεί μια άλλη μέθοδο ταξινόμησης (η οποία θα πρέπει να είναι ευσταθής) ως υπορουτίνα.

Παράδειγμα ταξινόμησης radix sort
Έστω η ακολουθία εισόδου: 341, 167, 43, 99, 777, 51, 92, 80, 60, 101
Ταξινόμηση με το ψηφίο των μονάδων: 80, 60, 341, 51, 101, 92, 43, 167, 777, 99
Ταξινόμηση με το ψηφίο των δεκάδων: 101, 341, 43, 51, 60, 167, 777, 80, 92, 99
Ταξινόμηση με το ψηφίο των εκατοντάδων: 43, 51, 60, 80, 92, 99, 101, 167, 341, 777

Μια απλή υλοποίηση του αλγορίθμου Radix Sort

```
void count_sort_digit_by_digit(int a[],
     int n, int d) {
    int b[10] = { 0 }; int temp[n];
    for (int i = 0; i < n; i++)
        b[(a[i] / d) % 10]++;
    for (int i = 1; i < 10; i++)
        b[i] += b[i - 1];
    for (int i = n - 1; i >= 0; i--) {
        int x = (a[i] / d) % 10;
        temp[b[x] - 1] = a[i];
        b[x]--;
    }
    for (int i = 0; i < n; i++)
        a[i] = temp[i];
}
void radix_sort(int arr[], int n, int
     m) {
    int d = 1;
    for (int i = 0; i < m; i++) {
        count_sort_digit_by_digit(arr, n,
             d);
        d *= 10;
    }
}
 ...
int arr[] = { 170, 45, 75, 90, 802, 24,
     2, 66 };
radix_sort(arr, 8, 3);
// 2 24 45 66 75 90 170 802
```

Προσαρμοσμένο από το
`http://www.geeksforgeeks.org/radix-sort/`

- 170 45 75 90 802 24 2 66
- 170 90 802 2 24 45 75 66
- 802 2 24 45 66 170 75 90
- 2 24 45 66 75 90 170 802

Απόδοση του Radix Sort

- Η απόδοση του αλγορίθμου εξαρτάται από το πλήθος των ψηφίων που απαιτούνται για την αναπαράσταση των τιμών εισόδου.
- Η αλγοριθμική πολυπλοκότητα του Radix Sort είναι $O(kn)$, όπου k είναι το πλήθος των ψηφίων και n είναι το μέγεθος της εισόδου.

Κριτήρια επιλογής αλγορίθμου ταξινόμησης

- Μικρό πλήθος στοιχείων $\Rightarrow$ Ταξινόμηση με εισαγωγή (INSERTION SORT)
- Σχεδόν ταξινομημένα δεδομένα $\Rightarrow$ Ταξινόμηση με εισαγωγή (INSERTION SORT)
- Διασφάλιση καλής συμπεριφοράς για τη χειρότερη περίπτωση $\Rightarrow$ Ταξινόμηση σωρού (HEAP SORT) (HeapSort)
- Διασφάλιση καλής συμπεριφοράς κατά μέσο όρο $\Rightarrow$ Γρήγορη ταξινόμηση (QUICK SORT)
- Τα δεδομένα προέρχονται από μια πυκνή κατανομή πιθανών τιμών $\Rightarrow$ Ταξινόμηση κάδου (BUCKET SORT)
- Εύκολη και γρήγορη συγγραφή κώδικα $\Rightarrow$ Ταξινόμηση με εισαγωγή (INSERTION SORT)

Πηγή: Algorithms in a nutshell by G.T.Heineman et al.