Δομές Δεδομένων και Αλγόριθμοι

Χρήστος Γκόγκος

ΤΕΙ Ηπείρου

Χειμερινό Εξάμηνο 2014-2015
Παρουσίαση 8. Quick Sort

Ο αλγόριθμος QuickSort

1. Προτάθηκε από τον C.A.R. (Tony) Hoare το 1961
2. Ο αλγόριθμος QuickSort επιλέγει ένα στοιχείο (το pivot) και χωρίζει με βάση αυτό το στοιχείο την ακολουθία σε 2 υποακολουθίες, την αριστερή που περιέχει τιμές μικρότερες του pivot και τη δεξιά που περιέχει τιμές μεγαλύτερες του pivot. Στη συνέχεια κάθε υποακολουθία ταξινομείται με αναδρομική κλήση του ίδιου αλγορίθμου.
3. Η επιλογή του pivot γίνεται χρησιμοποιώντας κάποια στρατηγική (πρώτο στοιχείο από αριστερά, το μεσαίο στοιχείο, τυχαία κ.α.)

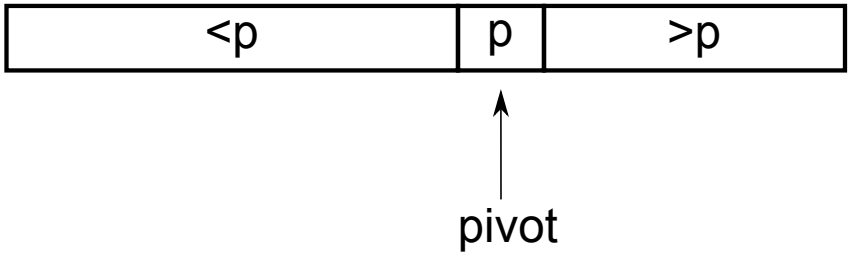
Ο αλγόριθμος QuickSort

QuickSort(A)

1 // Η ακολουθία A έχει n στοιχεία
2 **if** $n = 1$
3 return
4 **else**
5 Επιλογή στοιχείου pivot p
6 Partition(A, p) // Διαμερισμός του A σε σχέση με το p
7 QuickSort(A_l) // A_l = τμήμα του A αριστερά του p
8 QuickSort(A_r) // A_r = τμήμα του A δεξιά του p

Διαμερισμός ακολουθίας με βάση το στοιχείο pivot χρησιμοποιώντας βοηθητικό πίνακα

Ο διαμερισμός των στοιχείων θα πρέπει να γίνει σε χρόνο $O(n)$



Η διαμέριση των στοιχείων μπορεί να γίνει εύκολα αν χρησιμοποιηθεί ένας βοηθητικός πίνακας

Partition (not in place)

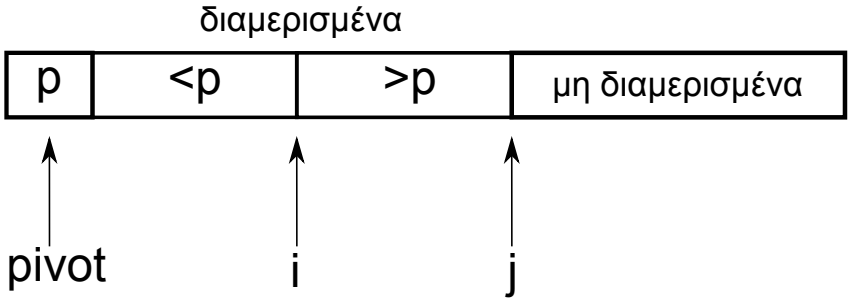
```
#include <iostream>
using namespace std;
template<class T>
void partition_not_in_place(T a[], int
     n, int p_index) {
   T temp[n] = { 0 };
   T pivot = a[p_index];
   int left = 0, right = n - 1;
   for (int i = 0; i < n; i++)
      if (a[i] < pivot) {
         temp[left] = a[i];
          left++;
      } else if (a[i] > pivot) {
         temp[right] = a[i];
          right--;
      }
   for (int i = left; i <= right; i++)
      temp[i] = pivot;
   for (int i = 0; i < n; i++)
      a[i] = temp[i];
}
int main(int argc, char **argv) {
   int n = 6;
   for (int p_index = 0; p_index < n;
        p_index++) {
      int a[n] = { 16, 5, 14, 18, 11,
           10 };
      int pivot = a[p_index];
      partition_not_in_place(a, n,
           p_index);
      printf ("Pivot=%d \t==> Array=",
           pivot);
      for (int i = 0; i < n; i++)
         printf ("%d ", a[i]);
      printf ("\n");
   }
}
```

έξοδος

```
Pivot=16 ==>Array=5 14 11 10 16 18
Pivot=5 ==>Array=5 10 11 18 14 16
Pivot=14 ==>Array=5 11 10 14 18 16
Pivot=18 ==>Array=16 5 14 11 10 18
Pivot=11 ==>Array=5 10 11 18 14 16
Pivot=10 ==>Array=5 10 11 18 14 16
```

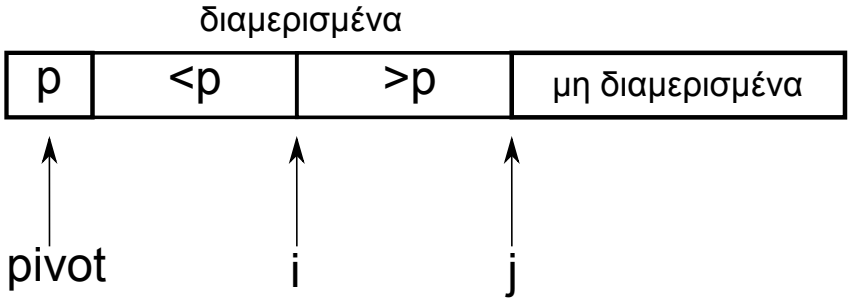
Partition (in place)

- Υποθέτουμε ότι το pivot βρίσκεται στην πρώτη θέση της ακολουθίας
- Αν το pivot δεν είναι στην πρώτη θέση της ακολουθίας τότε θα πρέπει αρχικά να γίνει αντιμετάθεσή του με το στοιχείο το οποίο βρίσκεται στην πρώτη θέση
- Καθώς λειτουργεί ο αλγόριθμος και μέχρι τη θέση στην οποία έχει διανυθεί η ακολουθία τα στοιχεία είναι διαμερισμένα.



Partition (in place)

- Υποθέτουμε ότι το pivot βρίσκεται στην πρώτη θέση της ακολουθίας
- Αν το pivot δεν είναι στην πρώτη θέση της ακολουθίας τότε θα πρέπει αρχικά να γίνει αντιμετάθεσή του με το στοιχείο το οποίο βρίσκεται στην πρώτη θέση
- Καθώς λειτουργεί ο αλγόριθμος και μέχρι τη θέση στην οποία έχει διανυθεί η ακολουθία τα στοιχεία είναι διαμερισμένα.




```
#include <iostream>
using namespace std;
template<class T> int partition (T a[], int l
    int r) {
  int p = l;
  int i = l + 1;
  for (int j = l + 1; j <= r; j++)
    if (a[j] < a[p]) {
      swap(a[j], a[i]);
      i++;
    }
  swap(a[p], a[i - 1]);
  return i - 1;
}
int main(int argc, char **argv) {
  int n = 6;
  for (int p_index = 0; p_index < n;
      p_index++) {
    int a[n] = { 16, 5, 14, 18, 11, 10 };
    int pivot = a[p_index];
    printf ("Pivot=%d \t==> Array=", pivot)
    swap(a[0], a[p_index]);
    partition (a, 0, n);
    for (int i = 0; i < n; i++) {
      printf ("%d ", a[i]);
    }
    printf ("\n");
  }
}
```

Ο πλήρης αλγόριθμος QuickSort

```
#include <iostream>
using namespace std;
template<class T> int partition (T a[], int l, int r) {
    int p = l,
    int i = l + 1;
    for (int j = l + 1; j <= r; j++)
        if (a[j] < a[p]) {
            swap(a[j], a[i]);
            i++;
        }
    swap(a[p], a[i - 1]);
    return i - 1;
}
template<class T> void quick_sort(T a[], int l, int r) {
    if (l >= r) return;
    else {
        // insert code here if you want a different strategy
        // for pivot selection (finally pivot should be swapped
        // with item at index l)
        int p = partition (a, l, r);
        quick_sort(a, l, p - 1);
        quick_sort(a, p + 1, r);
    }
}
template<class T> void quick_sort(T a[], int N) {
    quick_sort(a, 0, N - 1);
}
int main(int argc, char **argv) {
    int n = 6;
    int a[n] = { 16, 5, 14, 18, 11, 10 };
    quick_sort(a, n);
    for (int i = 0; i < n; i++) printf ("%d ", a[i]);
}
```

Η απόδοση του QuickSort

- Η μέση απόδοση του αλγορίθμου είναι $O(n \log n)$ ενώ η χειρότερη απόδοση του αλγορίθμου είναι $O(n^2)$
- Η $O(n^2)$ συμπεριφορά του QuickSort παρατηρείται όταν η διαμέριση σε κάθε αναδρομικό βήμα χωρίζει τα n στοιχεία σε ένα άδειο σύνολο και σε ένα σύνολο με όλα τα υπόλοιπα στοιχεία πλην του pivot (δλδ $n-1$ στοιχεία)
- Η επιλογή του pivot με τυχαίο τρόπο επιτρέπει στον QuickSort συχνά να είναι γρηγορότερος από ανταγωνιστικούς αλγορίθμους

Ανάλυση του QuickSort

Στην ιδανική περίπτωση η συνάρτηση διαμέρισης (partition) χωρίζει την ακολουθία τιμών στη μέση. Αν αυτό μπορούσε να συμβεί σε κάθε αναδρομή τότε ο χρόνος εκτέλεσης της QuickSort θα ήταν:
$t(n) = 2t(\frac{n}{2}) + O(n)$ όπου O(n) είναι ο χρόνος που απαιτεί η διαμέριση
Πραγματοποιώντας αντικαταστάσεις προκύπτει:
$\rightarrow 2\left[2(t(\frac{n}{4}) + O(\frac{n}{2})\right] + O(n) \rightarrow \cdots \rightarrow 2^k t(\frac{n}{2^k}) + O(k * n)$ Λόγω του ότι το ανωτέρω ανάπτυγμα τελειώνει όταν
$2^k = n \rightarrow (k = \log(n))$ και επειδή ο χρόνος επίλυσης με μέγεθος προβλήματος 1 μπορεί να θεωρηθεί σταθερός c προκύπτει ότι $t(n) = n * c + O(n \log(n))$ και καθώς το $O(n \log(n))$ είναι ασυμπτωτικά μεγαλύτερο από το cn προκύπτει ότι ο χρόνος εκτέλεσης μπορεί να γραφεί ως $O(n \log n)$

Παραλλαγές του QuickSort

- Χρήση της ταξινόμησης με εισαγωγή όταν το μέγεθος μιας διαμέρισης είναι μικρό (π.χ. 10)
- Επιλογή του pivot χρησιμοποιώντας την διάμεσο των τριών (δλδ τη διάμεσο ανάμεσα στο πρώτο, στο τελευταίο και στο μεσαίο στοιχείο της ακολουθίας)
- Κλήση της αναδρομικής συνάρτησης με την μικρότερη ακολουθία τιμών από τις 2 πρώτα (μείωση μεγέθους που απαιτείται στο stack)

IntroSort

Introspective Sort

Πρόκειται για τον αλγόριθμο που χρησιμοποιείται από την STL της C++ από το 2000. Εκτιμά το πλήθος αναδρομών που θα απαιτηθεί από την QuickSort και αν ξεπερνά ένα όριο τότε η ταξινόμηση γίνεται με την HeapSort. Αλλιώς ενεργοποιείται η QuickSort με επιλογή pivot με την μέθοδο της διαμέσου των τριών (median of three) και εφαρμόζοντας την ταξινόμηση με εισαγωγή όταν προκύπτουν μικρές ακολουθίες τιμών. Προτάθηκε από τον D. Musser το 1997