Datum: 4.11.2018.

Algoritmi za sortiranje – 2 : Metode podijeli-i-osvoji (*Divide-and-Conquer*)

U ovom matrijalu će biti prikazane dvije podijeli-i-osvoji metode: MergeSort i QuickSort. Ideja ovih metoda podjeljena je u tri koraka:

(1) podijeliti početni nesortirani niz na dva podniza,

(2) sortirati podnizove, i

(3) kombinovati dva sortirana podniza u konačno rješenje.

Apstraktna strategija podijeli-i-osvoji metoda data je u slijedećoj programskoj strategiji. Ona je izražena tako da se može rekurzivno primjenjivati na podnizove niza A.

Za vizualizaciju rada algoritama se preporučuje stranica razvijena na Univerzitetu u San Francisku: https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

```
void Sort(SortNiz A, int m, int n)
/* treba sortirati podniz A[m:n] niza A u rastući poredak */
{
       if     (postoji više od jednog elementa koje treba sortirati u A[m:n], m<n)
       {
              (podjeliti A[m:n] na dva podniza A[m:i] i A[j:n])
              (sortirati A[m:i] rekurzivnim pozivom funkcije Sort(A,m,i))
              (sortirati A[j:n] rekurzivnim pozivom funkcije Sort(A,j,n))
              (sortiran originalni niz se dobja kombinacijom dva sortirana podniza)
       }
}
```

MergeSort

Apstraktni prikaz strategije algoritma MergeSort dat je u narednoj programskoj strategiji.

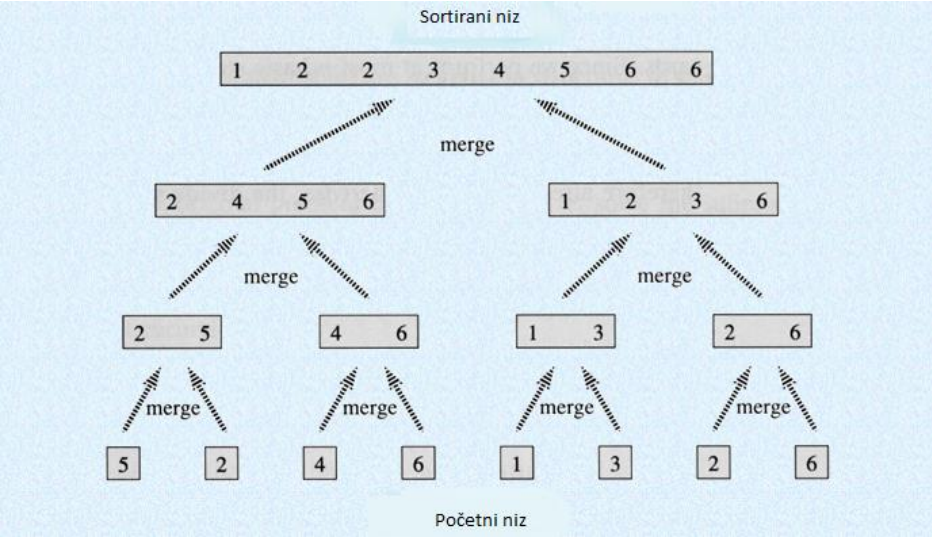
```
void MergeSort(SortingArray A, int m, int n)
{
/*  treba sortirati podniz A[m:n] niza A u rastući poredak */
       (deklarisati pomoćni brojač Srednji)
       if     (postoji više od jednog elementa koje treba sortirati u A[m:n], m<n)
       {
              (podijeliti A[m:n] na polovine, LijeviNiz i DesniNiz, Srednji=(m+n)/2)
              (sortirati LijeviNiz pozivom funkcije MergeSort(A,m,Srednji))
              (sortirati DesniNiz pozivom funkcije MergeSort(A,Srednji+1,n))
              (spojiti LijeviNiz i DesniNiz da bi se dobio rezultat)
       }
}
```

Osnova MergeSort algoritma je funkcija Merge kojom se dva sortirana niza spajaju u jedan sortirani niz. U ovoj funkciji se vrši i sortiranje. Opet ćemo navesti samo apstraktnu strategiju za funkciju Merge. Ideja je sljedeća:

```
void Merge(SortNiz *A, int Prvi, int Srednji, int Kraj)
/* Za spajanje dva sortirana niza u jedan, koristi se pomoćni niz, PomNiz */
/* Na kraju algoritma kopira se PomNiz u originalni niz A. */
/* Pošto imamo dva podniza koja treba spojiti, koristimo dva pomoćna indeksa za */
/* početak (p1 i p2) i kraj (k1 i k2) podniza i jedan pomoćni indeks, ind za PomNiz */

(inicijalno postaviti vrijednosti Prvi u p1, Srednji u k1, Srednji+1 u p2, Kraj u k2 i
p1 u ind)
       while  ((p1 nije većeod k1) i (p2 nije veće od k2))    {
/* dok su oba podniza neprazna */
              if     (A[p1] je manje od A[p1])    {
                     (u PomNiz[ind] smjestiti A[p1])
                     (uvećati p1 za 1)
                     (uvećati ind za 1)
              }else{
                     (u PomNiz[ind] smjestiti A[p2])
                     (uvećati p2 za 1)
                     (uvećati ind za 1)
              }
       }
/* ako nam je preostalo elemenata prvog podniza */
       while  (p1 nije većeod k1) {
              (u PomNiz[ind] smjestiti A[p1])
              (uvećati p1 za 1)
              (uvećati ind za 1)
       }
/* ako nam je preostalo elemenata drugog podniza */
       while  (p2 nije većeod k2) {
              (u PomNiz[ind] smjestiti A[p2])
              (uvećati p2 za 1)
              (uvećati ind za 1)
       }
}
```

Algoritam MergeSort spada u brže algoritme jer radi u vremenu $O(n\log_2 n)$, ali ima koristi pomoćni niz kojij je iste veličine kao i originalni niz. Ipak, MergeSort spada u stabilne algoritme (algoritme koji se uvijek ponašaju na isti/sličan način, što se može povezati sa razgranatošću stabla poziva rekurzivne funkcije), ali se njegova upotreba ne preporučuje za male nizove. Na slici je ilustrovana ideja.



Postoje razne implementacije koda za algoritam MergeSort u raznim programskim jezicima[1]. No, osnovna ideja i rekurzivna funkcija su srž svake imaplementacije, iako je moguće implementirati ne-rekurzivni algoritam. Poboljšanja MergeSort algoritma se odnose na:

1. Smanjivanje broja poziva provjerom uslova da je najveći element u lijevom podnizu manji od najmanjeg u desnom (u tom slučaju se podnizovi mogu direktno spojiti)
2. Izbjeći korištenje dodatnog niza (ovo poboljšanje je moguće, ali je komplikaovano).

[1] Možete pogledati https://algs4.cs.princeton.edu/lectures/22Mergesort.pdf ili http://csg.sph.umich.edu/abecasis/class/2006/615.09.pdf

Quick sort

Quick sort spada u podjeli-i-osvoji (divide et impera, divide-and-conquer) algoritme, u prosječnom slučaju radi u vremenu O(nlogn) i zbog toga se veoma često koristi u praksi.

Proces kojim se sortira niz je sljedeći: prvo, bira se element niza kao vodeći (***pivot***) element. Pivot se koristi za razdvajanje svih elemenata niza u dvije particije:

- lijeva particija sadrži elemente manje ili jednake pivotu, a
- desna particija sadrži elemente veće ili jednake pivotu.

Dalje se QuickSort rekurzivno poziva za sortiranje lijeve i desne particije. Kada se proces završi, nisu potrebna nikakva druga preuređenja niza, jer je originalni niz, koji sadrži dvije sortirane particije, lijevu i desnu, sortiran u rastućem poretku.

Analiza kompleksnosti

O-notacija za prosječan broj poređenja koja koristi QuickSort je O(n*log*n).

U najgorem slučaju, nasuprot prosječnom, vrijeme izvršavanja algoritma QuickSort za niz od *n* ključeva, može biti proporcionalno $O(n^2)$. Ovo se dešava kada je izabrani pivot ključ, stalno ili najveći ili najmanji ključ u particiji. U tom slučaju, originalni niz se procesom particije dijeli na jednu particiju koja sadrži samo jedan ključ i drugu koja sadrži (n – 1) ključeva. Particija sa (n – 1) ključeva se, dalje, dijeli na jednu particiju sa samo jednim ključem i drugu sa (n – 2) ključa, i tako dalje. Ukupan broj poređenja korišćenih u kreiranju svih tih particija je suma oblika: 2 + 3 + ... + (n – 1) + n , što je $O(n^2)$.

Primjer sortiranja

Nesortirani niz

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**8**	**2**	**15**	**4**	**7**	**1**	**3**	**6**

U prvom pozivu funkcije se sortira čitav niz, pa je lijevi=0=i, desni=7=j. Na slici je prikazan izlaz za navedeni kod.

```
C:\Users\nbijedic.FIT\Documents\Visual Studio 2010\Projects\kvik\Debug\kvik.exe
Nesortirani niz je
8       2       15      4       7       1       3       6

Rekurzivni poziv: quickSort(Niz, 0, 7) , pivot= 4
dok je Niz[i] > pivot, umanji j, j= 6
ako je i<j, zamjena Niz[0] i Niz[6], niz je:
3       2       15      4       7       1       8       6
uvecaj i i umanji j, i= 1, j= 5
dok je Niz[i] < pivot, uvecaj i, i= 2
ako je i<j, zamjena Niz[2] i Niz[5], niz je:
3       2       1       4       7       15      8       6
uvecaj i i umanji j, i= 3, j= 4
dok je Niz[i] > pivot, umanji j, j= 3
ako je i<j, zamjena Niz[3] i Niz[3], niz je:
3       2       1       4       7       15      8       6
uvecaj i i umanji j, i= 4, j= 2
ako je lijevi manji od umanjenog desnog (j)
Rekurzivni poziv: quickSort(Niz, 0, 2) , pivot= 2
ako je i<j, zamjena Niz[0] i Niz[2], niz je:
1       2       3       4       7       15      8       6
uvecaj i i umanji j, i= 1, j= 1
ako je i<j, zamjena Niz[1] i Niz[1], niz je:
1       2       3       4       7       15      8       6
uvecaj i i umanji j, i= 2, j= 0
ako je uvecani lijevi (i) manji od desnog
Rekurzivni poziv: quickSort(Niz, 4, 7) , pivot= 15
dok je Niz[i] < pivot, uvecaj i, i= 5
ako je i<j, zamjena Niz[5] i Niz[7], niz je:
1       2       3       4       7       6       8       15
uvecaj i i umanji j, i= 6, j= 6
dok je Niz[i] < pivot, uvecaj i, i= 7
ako je lijevi manji od umanjenog desnog (j)
Rekurzivni poziv: quickSort(Niz, 4, 6) , pivot= 6
dok je Niz[i] > pivot, umanji j, j= 5
ako je i<j, zamjena Niz[4] i Niz[5], niz je:
1       2       3       4       6       7       8       15
uvecaj i i umanji j, i= 5, j= 4
ako je uvecani lijevi (i) manji od desnog
Rekurzivni poziv: quickSort(Niz, 5, 6) , pivot= 7
dok je Niz[i] > pivot, umanji j, j= 5
ako je i<j, zamjena Niz[5] i Niz[5], niz je:
1       2       3       4       6       7       8       15
uvecaj i i umanji j, i= 6, j= 4
_
```