
Algoritmi și structuri de date (I). Seminar 10: Aplicații ale tehnicii divizării. Aplicații ale interclasării.

Problema 1 Fie $a[1..m]$ și $b[1..n]$ două tablouri ordonate crescător având elemente nu neapărat distincte. Să se construiască un tablou strict crescător ce conține elementele distincte din a și b .

Rezolvare. Se aplică tehnica interclasării verificând de fiecare dată la completarea în tabloul destinație dacă elementul ce ar trebui adăugat este diferit de ultimul element din tablou.

```
interclasare(a[1..m],b[1..n])
i ← 1; j ← 1; k ← 0
if a[i] < b[j] then k ← k + 1; c[k] ← a[i]; i ← i + 1
    else if a[i] > b[j] then k ← k + 1; c[k] ← b[j]; j ← j + 1
    else k ← k + 1; c[k] ← a[i]; i ← i + 1; j ← j + 1
    endif
endif
while i ≤ m AND j ≤ n do
    if a[i] < b[j] then
        if a[i] ≠ c[k] then k ← k + 1; c[k] ← a[i]; i ← i + 1
        else i ← i + 1 endif
    else if a[i] > b[j] then
        if b[j] ≠ c[k] then k ← k + 1; c[k] ← b[j]; j ← j + 1
        else j ← j + 1 endif
        if a[i] ≠ c[k] then k ← k + 1; c[k] ← a[i]; i ← i + 1; j ← j + 1
        else i ← i + 1; j ← j + 1 endif
    endif
    endif
endwhile
while i ≤ m do
    if a[i] ≠ c[k] then k ← k + 1; c[k] ← a[i]; i ← i + 1
    else i ← i + 1 endif
endwhile
while j ≤ n do
    if b[j] ≠ c[k] then k ← k + 1; c[k] ← b[j]; j ← j + 1
    else j ← j + 1 endif
endwhile
return c[1..k]
```

Ordinul de complexitate este în acest caz $\mathcal{O}(m + n)$.

Problema 2 Se consideră două numere întregi date prin tablourile corespunzătoare descompunerilor lor în factori primi. Să se construiască tablourile similare corespunzătoare celui mai mic multiplu comun al celor două valori.

Rezolvare. Să considerăm numerele: $3415 = 3 \cdot 5^3 \cdot 7 \cdot 13$ și $966280 = 2^3 \cdot 5 \cdot 7^2 \cdot 17 \cdot 29$. Primului număr îi corespund tablourile: (3, 5, 7, 13) respectiv (1, 3, 1, 1) iar celui de al doilea număr îi corespund tablourile (2, 5, 7, 17, 29) respectiv (3, 1, 2, 1, 1).

Tablourile corespunzătoare celui mai mic multiplu comun vor fi:

- *factori primi*: tabloul care conține toți factorii primi corespunzători celor două numere și care poate fi obținut prin interclasare ținând cont că aceștia trebuie să fie distincți.
- *exponenți*: tabloul ce conține valorile maxime ale exponenților.

În cazul exemplului cele două tablouri sunt: (2, 3, 5, 7, 13, 17, 29) respectiv (3, 1, 3, 2, 1, 1, 1)

```

interclasare(fa[1..m],pa[1..m],fb[1..n],pb[1..n])
i ← 1; j ← 1; k ← 0
while i ≤ m AND j ≤ n do
  if fa[i] < fb[j] then
    k ← k + 1; fc[k] ← fa[i]; pc[k] ← pa[i]; i ← i + 1
  else if fa[i] > fb[j] then
    k ← k + 1; fc[k] ← fb[j]; pc[k] ← pb[j]; j ← j + 1
    else k ← k + 1; fc[k] ← fa[i]; pc[k] ← max(pa[i], pb[j]); i ← i + 1; j ← j + 1
  endif
endif
endwhile
while i ≤ m do
  k ← k + 1; fc[k] ← fa[i]; pc[k] ← pa[i]; i ← i + 1
endwhile
while j ≤ n do
  k ← k + 1; fc[k] ← fb[j]; pc[k] ← pb[j]; j ← j + 1
endwhile
return fc[1..k], pc[1..k]

```

Problema 3 *Problema selecției celui de al k -lea element.* Fie $a[1..n]$ un tablou, nu neapărat ordonat. Să se determine al k -lea element al tabloului, selectat în ordine crescătoare (pentru $k = 1$ se obține minimumul, pentru $k = n$ se obține maximumul etc.).

Rezolvare. O primă variantă de rezolvare o reprezintă ordonarea crescătoare a tabloului (prin metoda selecției) până ajung ordonate primele k elemente ale tabloului. Numărul de operații efectuate în acest caz este proporțional cu kn . Pentru k apropiat de $n/2$ aceasta conduce la un algoritm de complexitate pătratică. Un algoritm de complexitate mai mică se obține aplicând strategia de partiționare de la sortarea rapidă: folosind o valoare de referință (de exemplu cea aflată pe prima poziție în tablou) se partiționează tabloul în două subtablouri astfel încât elementele aflate în primul subtablou să fie toate mai mici decât valoarea de referință iar elementele din al doilea subtablou să fie toate mai mari decât valoarea de referință.

Algoritmul de partiționare poate fi cel folosit în cadrul sortării rapide:

```

partiționare(integer a[li..ls])
v ← a[li]; i ← li - 1; j ← ls + 1;
while i < j do
  repeat i ← i + 1 until a[i] ≥ v
  repeat j ← j - 1 until a[j] ≤ v
  if i < j then a[i] ↔ a[j] endif
endwhile
return j

```

Dacă poziția de partiționare este q iar $k \leq q - li + 1$ atunci problema se reduce la selecția celui de al k -lea element din subtabloul $a[li..q]$. Dacă însă $k > q - li + 1$ atunci problema se reduce la

selecția celui de al $k - (q - li + 1)$ element din subtabloul $a[q + 1..ls]$. Valoarea parametrului k este întotdeauna cuprinsă între 1 și numărul de elemente din subtabloul prelucrat (în cazul în care $li = ls$ valoarea lui k va fi 1). Algoritmul poate fi descris prin:

```

selectie( $a[li..ls]$ ,  $k$ )
if  $li = ls$  then return  $a[li]$ 
else
     $q \leftarrow$  partitionare( $a[li..ls]$ )
     $r \leftarrow q - li + 1$ 
    if  $k \leq r$  then selectie( $a[li..q]$ ,  $k$ )
    else selectie( $a[q + 1..ls]$ ,  $k - r$ )
endif
endif

```

În cazul cel mai favorabil, partiționarea este echilibrată la fiecare etapă (cele două subtablouri au aproximativ același număr de elemente). Întrucât algoritmul de partiționare are complexitate liniară putem presupune că numărul de comparații efectuate asupra elementelor tabloului satisface următoarea relație de recurență:

$$T(n) = \begin{cases} 0 & n = 1 \\ T(n/2) + n & n > 1 \end{cases}$$

Aplicând teorema master pentru estimarea ordinului de complexitate (pentru $m = 2$, $d = 1$, $k = 1$) se obține că, în cazul cel mai favorabil, algoritmul are complexitate liniară. În cazul cel mai defavorabil partiționarea ar conduce la fiecare etapă la descompunerea într-un subtablou constituit dintr-un singur element și la un subtablou constituit din celelalte elemente. Relația de recurență ar fi în acest caz:

$$T(n) = \begin{cases} 0 & n = 1 \\ T(n - 1) + n & n > 1 \end{cases}$$

ceea ce conduce la o complexitate pătratică. Similar algoritmului de sortare rapidă, și algoritmul selecției se comportă în medie similar celui mai favorabil caz având o complexitate liniară.

Problema 4 *Determinare mediană.* Mediana unui tablou cu n elemente este elementul aflat pe poziția $\lfloor (n+1)/2 \rfloor$ în cadrul tabloului ordonat crescător (obs: în cazul în care n este par se consideră uneori ca mediană media aritmetică a valorilor aflate în mijlocul tabloului). Fie $x[1..n]$ și $y[1..n]$ două tablouri ordonate crescător. Să se determine mediana tabloului $z[1..2n]$ care conține toate elementele din x și y .

Rezolvare. O primă variantă ar fi să se construiască tabloul z prin interclasare și să se returneze elementul de pe poziția n . Este însă suficient să se realizeze o interclasare parțială până când se obține elementul de poziția n . În acest scop se poate folosi interclasarea bazată pe valori santinelă mai mari decât elementele din ambele tablouri.

```

interclasare (  $x[1..n], y[1..n]$  )
 $x[n+1] \leftarrow |x[n]| + |y[n]|$ ;  $y[n+1] \leftarrow |x[n]| + |y[n]|$ ;
 $i \leftarrow 1$ ;  $j \leftarrow 1$ ;
for  $k \leftarrow 1, n$  do
    if  $x[i] < y[j]$  then  $z[k] \leftarrow x[i]$ ;  $i \leftarrow i + 1$ 
    else  $z[k] \leftarrow y[j]$ ;  $j \leftarrow j + 1$ 
    endif
endfor
return  $z[n]$ 

```

Este evident că algoritmul are complexitate liniară.

Probleme suplimentare

1. Fie $a[1..m]$ și $b[1..n]$ două tablouri ordonate strict crescător. Propuneți un algoritm de complexitate liniară pentru determinarea mulțimii elementelor comune celor două tablouri.
2. Se consideră două numere întregi date prin tablourile corespunzătoare descompunerilor lor în factori primi. Să se construiască tablourile similare corespunzătoare celui mai mare divizor comun al celor două valori.
3. Se consideră un caroiaj de dimensiuni $2^k \times 2^k$ (pentru $k = 3$ se obține o tablă de șah clasică) în care unul dintre pătrățele este marcat. Propuneți o strategie bazată pe tehnica divizării care acoperă întreaga tablă (cu excepția pătrățelului marcat) cu piese constând din 3 pătrățele aranjate în forma de L (indiferent de orientare).
4. Propuneți un algoritm de complexitate medie $\mathcal{O}(n \log n)$ pentru a verifica dacă elementele unui tablou sunt distincte sau nu.
5. Se consideră un tablou ordonat crescător $a[1..n]$. Presupunem că tabloul a a fost transformat printr-o deplasare circulară la dreapta cu k poziții. De exemplu, dacă $a = [2, 3, 6, 8, 9, 11, 14]$ iar $k = 3$ tabloul transformat este $a = [9, 11, 14, 2, 3, 6, 8]$.
 - a) În ipoteza că valoarea k este cunoscută propuneți un algoritm de complexitate $\mathcal{O}(1)$ care determină cea mai mare valoare din a .
 - b) În ipoteza că valoarea k este necunoscută propuneți un algoritm de complexitate $\mathcal{O}(\log n)$ care determină cea mai mare valoare din a .
6. Se consideră un tablou cu numere naturale distincte $a[1..n]$ ordonat crescător. Propuneți un algoritm de complexitate $\mathcal{O}(\log n)$ care verifică dacă există $i \in \{1, \dots, n\}$ cu proprietatea că $a[i] = i$.
7. Propuneți un algoritm de complexitate $\mathcal{O}(n \log n)$ pentru a determina numărul de perechi $(a[i], a[j])$ ale unui tablou $a[1..n]$ (tabloul are elemente distincte) având proprietatea că $i < j$ și $a[i] > a[j]$.

Indicație. Se folosește ideea de la sortarea prin interclasare doar că nu se modifică tabloul ci doar se numără de câte ori o valoare mai mare se află înaintea unei valori mai mici.
8. Se consideră un tablou $a[1..n]$ care conține numere întregi (atât pozitive cât și negative). Propuneți un algoritm de complexitate liniară (și care folosește spațiu suplimentar de dimensiune constantă) pentru a transforma tabloul inițial astfel încât toate valorile negative să fie înaintea celor pozitive.