Metoda Divide et Impera

Lect. Dr. Marina Cidota
Facultatea de Matematică şi Informatică
Universitatea din Bucureşti

cidota@fmi.unibuc.ro

Prezentare realizată pe baza:

1. Horia Georgescu. Tehnici de programare, editura Universităţii din Bucuresti, 2005.
2. Ioan Odăgescu, Felix Furtună. Metode şi tehnici de programare, editura Computer Libris Agora, 1998.

Metoda Divide et Impera

- constă în împărţirea repetată a unei probleme de dimensiuni mari în mai multe subprobleme *de acelaşi tip*, urmată de rezolvarea acestora şi combinarea rezultatelor obţinute pentre a determina rezultatul corespunzător problemei iniţiale.

- dacă dimensiunea unei subprobleme este suficient de mică, ea este rezolvată direct.

- caracter recursiv

Schema generală – prelucrare asupra elementelor unui vector

```
function DivImp(p,u)
   if (u–p<ε) then r ← Prel(p,u);
                 else {
                         m ← Interm(p,u);
                         r1 ← DvImp(p,m);
                         r2 ← DivImp(m+1,u);
                         r ← Combin(r1,r2);
                        }
  return r;
end;
```

Funcţia DivImp, care întoarce rezultatul prelucrării asupra unei subsecvenţe $a_p,...a_u$, va fi apelată prin DivImp(1,n).

Exemplul 1 – Sortare prin interclasare

```
procedure Inter(p,m,u)
   k1←p; k2←m+1; k3←p;
   while (k1≤m && k2≤u)
      if (a_k1<a_k2) then {b_k3←a_k1; k1←k1+1;}
                    else {b_k3←a_k2 ;k2←k2+1;}
      k3←k3+1;
   end
   if (k1>m)                // au fost epuizate elementele primei subsecvenţe
   then for i=k2,u
           b_k3←a_i; k3←k3+1;
   else                     // au fost epuizate elementele celei de a 2-a subsecvenţe
      for i=k1,m
           b_k3←a_i; k3←k3+1;
   for i=p,u
      a_i←b_i;
end
```

Exemplul 1 – Sortare prin interclasare

```
procedure SortInter(p,u)
   if (p==u)
   then
   else
        m ← ⌊(p+u)/2⌋;
        SortInter(p,m); SortInter(m+1,u);
        Inter(p,m,u);
end
```

In programul principal se face apelul SortInter(1,n)

Exemplul 2 – Sortare rapidă

Fie $(a_p,...,a_u)$ secvenţa curentă care trebuie sortată. Vom *poziţiona* pe a_p în secvenţa $(a_p,...,a_u)$, adică printr-o permutare a elementelor secvenţei:

- $x=a_p$ va trece pe o poziţie k;
- toate elementele aflate la stânga poziţiei k vor fi mai mici decât x;
- toate elementele aflate la dreapta poziţiei k vor fi mai mari decât x.

Exemplul 2 – Sortare rapidă

```
function poz(p,u)
   i←p; j←u; ii←0; jj←-1
   while (i<j)
      if (a_i<a_j)
          then
          else {a_i↔a_j; (ii,jj)←(-jj,-ii);}
      i←i+ii; j←j+jj;
   end
   poz ← i;
end
```

Exemplul 2 – Sortare rapidă

```
procedure QuickSort(p,u)
   if (p>=u)
   then
   else {k ← poz(p,u); QuickSort(p,k-1); QuickSort(k+1,u);}
end
```

Sortarea se realizează prin apelul QuickSort(1,n)

Exemplul 3 – Căutarea binară

Se consideră vectorul $a=(a_1,...,a_n)$ ordonat crescător şi o valoare x. Se cere să se determine dacă x apare printre componentele vectorului.

Vom adăuga $a_0=-\infty$, $a_{n+1}=+\infty$. Căutăm perechea (b,i) dată de:

- (true,i) dacă $a_i=x$;
- (false,i) dacă $a_{i-1}<x<a_i$.

Deoarece problema se reduce la o singură subproblemă, nu mai este necesar să folosim recursivitatea.

Exemplul 3 – Căutarea binară

```
procedure CautBin
   p ← 1; u ← n
   while (p≤u)
      i ← ⌊(p+u)/2⌋;
      case
           a_i>x : u ← i-1;
           a_i=x : write(true,i); return;
           a_i<x : p ← i+1;
   end
   write(false,p);
end
```

Exemplul 4 – Problema turnurilor din Hanoi

Se consideră 3 tije. Iniţial pe tija 1 se află n discuri cu diametrele descrescătoare privind de la bază către vârf, iar pe tijele 2 şi 3 nu se află nici un disc. Se cere să se mute aceste discuri pe tija 2, ajutându-ne şi de tija 3, respectând condiţia ca în permanenţă pe orice tijă sub orice disc să se afle baza tijei sau un disc de diametru mai mare.

Exemplul 4 – Problema turnurilor din Hanoi

Fie *H*(m;i,j) şirul de mutări prin care cele m discuri din vârful tijei i sunt mutate peste cele de pe tija j, folosind şi a treia tijă, al cărei număr este k=6-i-j. Problema constă în a determina *H*(n;1,2).

Se observă că este satisfăcută relaţia:

$H(m;i,j) = H(m-1;i,k)\ (i,j)\ \ H(m-1;k,j)$

Exemplul 4 – Problema turnurilor din Hanoi

```
procedure Hanoi(n,i,j)
   if (n==1) then write(i,j);
                 else {
                          k←6-i-j;
                          Hanoi(n-1,i,k); Hanoi(1,i,j); Hanoi(n-1,k,j);
                         }
end
```

Observaţie. Numărul de mutări este 2^n-1.

Exemplul 5 – Placa cu găuri

Se da o placă dreptunghiulară în care sunt facute n găuri şi poziţiile lor date prin coordonatele (x,y). Ştiind ca găurile sunt considerate punctiforme, să se determine placa de arie maximă care se poate decupa din placa iniţială şi care nu contine în interior nicio gaură.

Să considerăm placa iniţială având coordonatele (0,0) - colţul din stânga-jos şi (L, l) colţul din dreapta-sus.

Fie (x_i, y_i), i=1, …, n, coordonatele celor n găuri

Exemplul 5 – Placa cu găuri

```
procedure FaraGauri(a,b,c,d,k)
   float a,b,c,d;
   int k;
   for i=1,n
         if (a<x_i<c and b<y_i<d) then {k←i; return;}
         endif
   endfor
   k←0
end
```

Exemplul 5 – Placa cu găuri

procedure Placa(x_{sj},y_{sj},x_{ds},y_{ds},A,x_{01},y_{01},x_{02},y_{02})
   float x_{sj},y_{sj},x_{ds},y_{ds},A,x_{01},y_{01},x_{02},y_{02}
   call FaraGauri(x_{sj},y_{sj},x_{ds},y_{ds},k)
   if (k==0) then {$A \leftarrow (x_{ds}-x_{sj})(y_{ds}-y_{sj})$; $x_{01} \leftarrow x_{sj}$; $y_{01} \leftarrow y_{sj}$; $x_{02} \leftarrow x_{ds}$; $y_{02} \leftarrow y_{ds}$;}
            else {
                call Placa(x_{sj},y_{sj},x_{ds},y_k,A_1,x_{01}^1,y_{01}^1,x_{02}^1,y_{02}^1);
                call Placa(x_{sj},y_k,x_{ds},y_{ds},A_2,x_{01}^2,y_{01}^2,x_{02}^2,y_{02}^2);
                call Placa(x_{sj},y_{ds},x_k,y_{ds},A_3,x_{01}^3,y_{01}^3,x_{02}^3,y_{02}^3);
                call Placa(x_k,y_{sj},x_{ds},y_{ds},A_4,x_{01}^4,y_{01}^4,x_{02}^4,y_{02}^4);
                call AlegeMax(A_1,A_2,A_3,A_4,A,x_{01},y_{01},x_{02},y_{02});
                }
end

Exemplul 6 – Arbori binari

Cunoscând vectorii corespunzători parcurgerilor în preordine şi inordine ale unui arbore (care are în fiecare nod o informație număr întreg, distinct de numerele din celelalte noduri), construiți arborele inițial şi parcurgeți-l în preordine şi inordine.

Vectorii rsd[0..n-1] si srd[0..n-1] sunt cunoscuți.

```
class Nod{
 int info;
 Nod st,dr;
 Nod (int i) {info=i;st=dr=null;}
}
 void inord(Nod x){
 if (x!=null) {inord(x.st); write(x.info); inord(x.dr);}
 }
 void preord(Nod x){
 if (x!=null) {write(x.info); preord(x.st); preord(x.dr);}
 }
```

Exemplul 6 – Arbori binari

```
Nod creare(int li1,ls1,li2,ls2){
            // li1,ls1 sunt marginile inf, respectiv sup ale vectorului srd
            // li2,ls2 sunt marginile inf, respectiv sup ale vectorului rsd
 if (li2==ls2) then {Nod rad=new Nod(rsd[li2]); return rad;}
 if (li2<ls2) then {int k,p;
                          for (int i=li1;i<=ls1;i++)
                                    if srd[i]=rsd[li2] then k=i;
                          Nod rad=new Nod(rsd[li2]);
                          p=k-li1;
                          rad.st=creare(li1,k-1,li2+1,li2+p);
                          rad.dr=creare(k+1,ls1,li2+p+1,ls2);
                          return rad;
                         }
 if (li2>ls2) then return null;
}
Se apeleaza radacina=creare(0,n-1,0,n-1); inord(radacina);preord(redacina);
```

Exemplul 7 – Subiect dat la admitere in 2017

Fie n un număr natural nenul. Fie v un vector cu n poziții numerotate de la 1 la n și elemente numere naturale diferite, de la 1 la n, într-o ordine oarecare. Pentru i și j numere naturale între 1 și n, numim FLIP(n, v, i, j) operația care inversează ordinea elementelor din v situate pe pozițiile de la i la j.

a) Să se scrie în limbaj de programare o procedură (sau funcție) care implementează operația FLIP(n, v, i, j).

b) Să se scrie un program care sortează crescător vectorul v, folosind pentru schimbarea ordinii elementelor în v doar operația FLIP(n, v, 1, k), cu k de la 2 la n.

c) Considerăm că n este o putere a lui 2 ($n = 2^m$, cu m număr natural nenul) și vectorul v are proprietatea că pentru orice i de la 1 la m și orice j de la 1 la 2^{m-i}, există k de la 1 la 2^{m-i}, astfel încât pe pozițiile din v de la la $2^i(j-1)+1$ la $2^i j$ se află numerele naturale de la la $2^i(k-1)+1$ la $2^i k$, într-o ordine oarecare. Să se scrie un program care sortează crescător vectorul v, folosind pentru schimbarea ordinii elementelor în v doar operația FLIP(n, v, $2^i(j-1)+1$, $2^i j$), cu i de la 1 la m și j de la 1 la la 2^{m-i}, printr-un algoritm mai eficient decât cel implementat la punctul b), care se bazează pe proprietatea vectorului v.

Exemplul 7 – Subiect dat la admitere in 2017

```
void flip(int n, int v[], int i, int j)
{
    i=i-1;//pornim de la 0 in C
    j=j-1;//pornim de la 0 in C
    int kk, aux;
    //nu mergem cu contorul pana la capat pentru ca altfel nu facem nimic
    for(kk=i;kk<=(i+j)/2;kk++)
    {
        aux = v[kk];
        v[kk] = v[i+j-kk];
        v[i+j-kk] = aux;
    }

}
```

Exemplul 7 – Subiect dat la admitere in 2017

```
void sorteazaFlip(int n,int v[]){
    int nCopie = n,i,j;
    for(i=nCopie;i>0;i--)
    {
        //gaseste pozitia pe care se afla i = maximul din vectorul ramas
        int maxim = v[0];
        int pozMaxim = 0;
        for(j=1;j<n;j++)
            if(maxim<v[j])
            {
                maxim = v[j]; pozMaxim = j;
            }
         flip(n,v,1,pozMaxim+1);//apelez flip cu +1 o pozitie (respect cerinta problemei)
         flip(n,v,1,n);
         n = n-1;
    }
}
```

Exemplul 7 – Subiect dat la admitere in 2017

```
void sorteazaFlipOptimDivideEtImpera(int n, int v[],int i,int j)
{
    //conditia de oprire
    if (j-i==1)
    {
    // nu faci nimic daca ai doua elemente
    return;
    }

    //divide problema in 2 subprobleme
    sorteazaFlipOptimDivideEtImpera(n,v,i,(i+j)/2);
    sorteazaFlipOptimDivideEtImpera(n,v,(i+j)/2+1,j);
```

Exemplul 7 – Subiect dat la admitere in 2017

```
    //combina solutia
    // daca elementele din subvectorul din stanga sunt mai mici decat cele din dreapta
    // trebuie sa sortez subvectorul din stanga crascator si cel din dreapta crescator
    if(v[i-1]<v[j-1])
    {
        if (v[i-1] > v[(i+j)/2-1])
                flip(n,v,i,(i+j)/2);
        if(v[(i+j)/2+1-1] > v[j-1])
                flip(n,v,(i+j)/2+1,j);
    }
    else
    {
    // elementele din subvectorul din stanga sunt mai mari decat cele din dreapta
    // trebuie sa sortez subvectorul din stanga descrescator si cel din dreapta descrescator
        if (v[i-1] < v[(i+j)/2-1])
               flip(n,v,i,(i+j)/2);
        if(v[(i+j)/2+1-1] < v[j-1])
               flip(n,v,(i+j)/2+1,j);
    }
}
```

Exemplul 8 – Problema plierii

Se consideră un vector de lungime *n*. Definim plierea vectorului prin suprapunerea unei jumătăţi numită *donatoare* peste cealaltă jumătate numită *receptoare* cu precizarea că dacă numărul de elemente este impar, elementul din mijloc este eliminat.

Exemplu: Vectorul (1, 2, 3, 4, 5) se poate plia în două moduri (1, 2) sau (4, 5) elementul 3 fiind eliminat conform ipotezei. Plierea se poate aplica în continuare în mod repetat până când se ajunge la un subşir de lungime 1, numit *element final*.

1) Fiind dat $i \in \{1, 2, ..., n\}$ să se determine dacă elementul *i* poate fi *element final* ca rezultat al unor plieri succesive.

2) Să se precizeze toate *elementele finale* posibile.

3) Fiind dat un element i final, să se figureze pe ecran o succesiune de plieri prin care se ajunge la el.

Exemplul 8 – Problema plierii

```
procedure ElFinale(i,j:integer);
    var k:integer;
    begin
      if (i==j) then f[i]:=true
                    else
                    begin
                      if ((i+j)%2==0) then f[(i+j) div 2]:=false;
                      ElFinale(i,i+j-1-(i+j) div 2);
                      for k:=(i+j) div 2+1 to j do f[k]:=f[i+j-k]
                    end
    end;
```

Exemplul 8 – Problema plierii

```
procedure Mutari(i,j:integer);
    var k:integer;
    begin
      if (i==j) then m[i]:=''
                  else
                    begin
                    if ((i+j)%2==0) then m[(i+j) div 2]:='2';
                    Mutari(i,i+j-1-(i+j) div 2);
                    for k:=i to i+j-1-(i+j) div 2 do  m[k]:='1'+m[k];
                    Mutari((i+j) div 2 +1,j);
                    for k:=(i+j) div 2+1 to j do m[k]:='0'+m[k]
                    end
    end;
```

Exemplul 8 – Problema plierii

```
Mutari(1,n);
 i:=1; j:=n;
 if (!f[v])
 then writeln('Elementul ',v,' nu este element final')
 else
   for k:=1 to Length(m[v]) do
   begin
     if copy(m[v],k,1)='1'
     then begin
            j:=i+j-1-(i+j) div 2;
            write('Se pliaza dreapta peste stanga:');
            write(i,'--->',j); writeln;
          end
     else begin
            i:=(i+j) div 2 +1;
            write('Se pliaza stanga peste dreapta:');
            write(i,'--->',j); writeln;
          end
   end
```

Problemă

Se dă o tablă de dimensiuni 2^m x 2^m cu o pătrățică lipsă. Să se acopere complet tabla cu formele din partea dreaptă a figurii de mai jos.