PROIECT INDIVIDUAL LA INFORMATICĂ
TEMA: DESPARTE ȘI STĂPÂNEȘTE

Realizat de: Mihaela Frecăuțanu, clasa a XI-a ,,C”

Verificat de: Maria Guțu

Introducere

- **Divide et impera** („dezbină și stăpânește”) este un principiu al puterii de stat, deseori aplicat de guvernele statelor multietnice, potrivit căruia cea mai bună metodă de gestionare a unui asemenea stat este provocarea dușmăniei interetnice pentru slăbirea puterii și impunerea voinței. (*wikipedia.ro*)

  În informatică, metoda desparte şi stăpîneşte este o metodă generală de elaborare a algoritmilor, care presupune:

  1) împărţirea repetată a unei probleme de dimensiuni mari în două sau mai multe subprobleme de acelaşi tip, dar de dimensiuni mai mici;

  2) rezolvarea subproblemelor în mod direct, dacă dimensiunea lor permite aceasta, sau împărţirea lor în alte subprobleme de dimensiuni şi mai mici;

  3) combinarea soluţiilor subproblemelor rezolvate pentru a obţine soluţia problemei iniţiale.

Metoda dată se poate aplica în rezolvarea unei probleme care indeplinește urmatoarele condiții :

- ✓ se poate descompune în (doua sau mai multe) subprobleme;
- ✓ aceste subprobleme sunt independente una față de alta (o subproblema nu se rezolvă pe baza alteia și nu se foloseste rezultate celeilalte);
- ✓ aceste subprobleme sunt similare cu problema inițială;
- ✓ la randul lor subproblemele se pot descompune (daca este necesar) în alte subprobleme mai simple;
- ✓ aceste subprobleme simple se pot soluționa imediat prin algoritmul simplificat.

Deoarece puține probleme îndeplinesc condițiile de mai sus, aplicarea metodei este destul de rară.

Schema generală a unui algoritm bazat pe metoda desparte şi stăpîneşte poate fi redată cu ajutorul unei proceduri recursive:

```
procedure DesparteSiStapineste(i,j: integer; var x:tip);
var m :integer;
x1, x2 :tip;
begin
if SolutieDirecta(i,j) then Prelucrare(i,j,x) else
 begin
 m:=(j-i) div 2;
 DesparteSiStapineste(i,i+m,x1);
 DesparteSiStapineste(i+m+1,j,x2);
 Combina(x1,x2,x);
 end;
 end;
```

Algoritmii ***Divide et Impera*** sunt în general rapizi, deoarece prin descompunere, de cele mai multe ori, se obțin probleme pentru care rezolvarea și combinarea soluțiilor au un grad de complexitate mai mic decât problema inițială. Algoritmii ***Divide et Impera*** se implementează, de obicei, într-un subprogram recursiv.

Exemple de probleme rezolvate

1. Problema sortării rapide(quicksort)

Un tablou V se completeaza cu n elemente numere reale. Să se ordoneze crescător folosind metoda de sortare rapidă. Soluția problemei se bazează pe următoarele etape implementate in programul principal:

- ✓ se apelează procedura "quick" cu limita inferioară li=1 și limita superioară ls=n;
- ✓ funcția "poz" realizează mutarea elementului v[i] exact pe poziția ce o va ocupa acesta în vectorul final ordonat; funcția"poz" întoarce (în k) poziția ocupată de acest element; în acest fel, vectorul V se împarte în două părți: li .k-1 si k+1.ls;
- ✓ pentru fiecare din aceste părți se reapelează procedura"quick, cu limitele modificate corespunzător ; în acest fel, primul element din fiecare parte va fi poziționat exact pe poziția finală ce o va ocupa în vectorul final ordonat (funcția"poz");
- ✓ fiecare din cele doua părți va fi, astfel, împărțită în alte două părti;
- ✓ procesul continuă pana cand limitele părților ajung să se suprapună, ceea ce indică că toate elementele vectorului au fost mutate exact pe pozițiile ce le vor ocupa în vectorul final; deci vectorul este ordonat ;
- ✓ în acest moment se produc întoarcerile din apelurile recursive și programul îți termină execuția.

```
program quicksort;
type vector= array [1..50] of real ;
var v:vector;
i,n,k:integer;
function poz(li,ls:integer):integer;
var i,j,modi,modj,m:integer;
man:real;
begin
i:=li; j:=ls;
modi:=0; modj:=-1;
while i<j do
begin
if v[i]>v[j] then
begin
 man:=v[i];
 v[i]:=v[j];
 v[j]:=man;
 m:=modi ;
 modi:=-modj;
 modj:=-m;
 end;
 i:=i+modi;
 j:=j+modj;
 end;
 poz:=i;
end;
procedure quick(li,ls:integer);
```

(1)

```
 begin
 if li<ls then begin
 k:=poz(li,ls);
 quick(li,k-1);
 quick(k+1,ls);
 end;
 end;
begin
write('cate elemente are vectorul ?=');readln(n);
for i:=1 to n do
begin
write('tastati elementul ',i,'=');
 readln(v[i]);
end;
quick(1,n);
writeln('vectorul ordonat este :');
for i:=1 to n do writeln(v[i]);
readln;
end.
```

(2)

REMARCĂ:

- dacă elementul se află în stanga, atunci se compară cu elementele din dreapta lui și se sar (j:=j1) elementele mai mari decât el ;
- dacă elementul se află în dreapta ,atunci se compară cu elemente din stânga lui ți se sar (i:=i+1) elementele mai mici decât el.

2. Problema turnurilor din Hanoi

Fie trei tije verticale notate A,B,C .Pe tija A se găsesc așezate n discuri de diametre diferite, în ordinea crescătoare a diametrelor, privind de sus în jos . Inițial, tijele B și C sunt goale. Să se afișeze toate mutările prin care discurile de pe tija A se mută pe tija B, în aceeași ordine, folosind ca tijă de manevră C și respectând următoarele reguli:

- ✓ la fiecare pas se mută un singur disc;
- ✓ un disc se poate așeza numai peste un disc cu diametrul mai mare.

Rezolvarea acestei probleme se bazează pe următoarele considerente logice:

- ✓ dacă n=1, atunci mutarea este imediată A->B(mut discul de pe A pe B);
- ✓ dacă n=2, atunci șirul mutărilor este : A->C, A->B, C->B;
- ✓ dacă n>2 procedăm astfel: mut (n-1) discuri A->C; mut un disc A->B; mut cele (n-1)discuri C->B.

```
program turnurile_hanoi;
program turnurile_hanoi;
var n:byte;
procedure hanoi(n:byte;a,b,c:char);
 begin
if n=1 then writeln(a,'a',b)
else begin
hanoi(n-1,a,c,b);
writeln(a,'a',b);
hanoi(n-1,c,b,a);
 end;
 end;
 begin
write('nr discuri pe tija A =');readln(n);
writeln('mutarile sunt urmatoarele :');
hanoi(n,'A','B','C');
readln;readln;
end.
```

REMARCĂ: Problema inițială se descompune în trei subprobleme mai simple, similare problemei inițiale: mut (n-1)discuri A->C ,mut ultimul disc pe B ,mut cele (n-1) discuri C->B.Dimensiunile acestor subprobleme sunt : n-1, 1, n-1.

Aceste subprobleme sunt independente, deoarece tijele inițiale (pe care sunt dispuse discurile), tijele finale și tijele intermediare sunt diferite. Notăm H(n,A,B,C)=șirul mutărilor a n discuri de pe A pe B, folosind C.

3. Problema maximului unui vector

Se citește un vector cu n componente, numere naturale. Se cere să se tipărească valoarea maximă.

```
program maxim;
var v:array[1..10] of integer;
n,i:integer;
function max(i,j:integer):integer;
var a,b:integer;
begin
if i=j then max:=v[i]
else begin
a:=max(i, (i+j) div 2);
b:=max((i+j) div 2+1,j);
if a>b then max:=a
else max:=b;
end;
end;
begin
write('n=');
readln(n);
for i:=1 to n do read(v[i]);
writeln(maximul este ',max(1,n));
end.
```

ALGORITM:

- ✓ dacă i=j, valoarea maximă va fi v[i];
- ✓ contrar vom împărți vectorul în doi vectori: primul vector va conține componentele de la i la (i+j) div 2, al doilea vector va conține componentele de la (I+J) div 2 +1 la j; rezolvăm problemele (aflăm maximul pentru fiecare din ele), iar soluția problemei va fi dată de valoarea maximă dintre rezultatele celor două subprobleme.

4. Problema sortării prin interclasare (mergesort)

Tabloul unidimensional V se completează cu n numere reale. Să se ordoneze crescător folosind sortarea prin interclasare.

Sortarea prin interclasare se bazează pe următoarea logică:

- ✓ vectorul V se împarte, prin înjumătățiri succesive, în vectori din ce în ce mai mici ;

- ✓ când se ating vectorii de maxim două elemente, fiecare dintre aceștia se ordonează printr-o simplă comparare a elementelor ;
- ✓ câte doi astfel de mini-vectori ordonați se interclasează succesiv până se ajunge iar la vectorul V.

```
program mergesort;
type vector=array[1..50] of real ;
var v:vector ;n,i:word;
procedure schimba(li,ls:word;var a:vector);
var man:real;
 begin
if a[li]>a[ls] then begin
man:=a[li];
a[li]:=a[ls];
a[ls]:=man;
end;
end;
procedure interclas(li,m,ls:word;var a:vector);
var b:vector;
i,k,p,j:word;
begin
i:=li; j:=m+1; k:=0;
while (i<=m)and(j<=ls) do
begin
inc(k);
if a[i] <a[j] then begin
b[k]:=a[i];
inc(i);
end;
else begin
b[k]:=a[j];
inc(j);
end;
end;
if i<=m then for p:=i to m do begin
inc(k);b[k]:=a[p];
end;
if j<=ls then for p:=j to ls do begin
inc(k) ;b[k]:=a[p];
```

(1)

```
end;
k:=0;
for p:=li to ls do begin
inc(k);
a[p]:=b[k];
end;
end;
procedure divi(li,ls:word; var a:vector);
var m:word;
begin
if (ls-li)<=1 then schimba(li,ls,a)
else begin
m:=(li+ls)div 2;
divi(li,m,a);
divi(m+1,ls,a);
interclas(li,m,ls,a);
end;
end;
begin
write('cate elemente are vectorul?=');readln(n);
for i:=1 to n do
begin
write('tastati elementul',i,'=');
readln(v[i]);
end;
divi(1,n,v);
writeln('vectorul sortat este:');
for i:=1 to n do writeln(v[i]);
end.
```

(2)

REMARCĂ:

- ✓ mecanismul general de tip Divide et Impera se găsește implementat în procedura "divi" ;
- ✓ astfel de abordare a problemei sortării unii vector conduce la economie de timp de calcul, deoarece operația de interclasare a doi vectori deja ordonați este foarte rapidă, iar ordonarea independentă celor doua jumătăți(mini- vectori) consumă în total aproximativ a doua parte din timpul care ar fi necesar ordonării vectorului luat ca întreg.

5. Problema sortării prin inserție binară

Să se ordoneze crescător un tablou unidimensional V de n numere reale, folosind sortarea prin inserție binară.

Pentru fiecare element v[i] se procedează în patru pași:

- ✓ se consideră ordonate elementele v[1],v[2],..,v[i-1]; se caută poziția k pe care urmează s-o ocupe v[i] între elementele v[1],v[2],.,v[i-1] (procedura "poz" prin căutare binară);
- ✓ se deplasează spre dreapta elementele din pozițiile k,k+1,.,n(procedura "deplasare");
- ✓ înserează elementul v[i] în poziția k (procedura"deplasare");
- ✓ se obține o succesiune de k+1 elemente ordonate crescător.

```
program sortare_binara;
type vector =array[1..50] of real ;
var n,k,i,j:integer;
v:vector;
function poz(li,ls,i:integer):integer;
 var m:integer;
begin
if li=ls then
if v[i]<v[j] then poz:=li
else poz:=i
else if ls-li=1 then if v[i]<v[ls] then if v[i]>=v[li]
then poz:=ls
else poz:=li
else poz:=i
else begin
m:=(li+ls)div 2;
if v[i]<v[m] then poz:=poz(li,m,i)
else poz :=poz(m,ls,i);
end;
end;
procedure deplasare(k,i:integer);
var man:real;
j:integer;
begin
if k<i then begin
man:=v[i];
```

(1)

```
for j:=I downto k+1 do v[j]:=v[j-1];
v[k]:=man;
end;
end;
begin
write('cate elemente are vectorul?=');readln(n);
for i:=1 to n do begin
write('tastati elementul ',i,'=');readln(v[i]);
end;
for i:=2 to n do begin
k:=poz(1,i-1,i);
deplasare(k,i);
end;
writeln('vectorul ordonat este :');
for i:=1 to n do writeln(v[i]);
readln;
end.
```

(2)

CONCLUZIE FINALĂ:

Nu toate problemele pot fi rezolvate prin utilizarea acestei tehnici. Se poate afirma că numărul celor rezolvabile prin "divide et impera" este relativ mic, tocmai datorită cerinței ca problema să admită o descompunere repetată.

BIBLIOGRAFIE:

- ✓ http://www.creeaza.com/referate/informatica/Metoda-de-programare-DIVIDE-ET449.php
- ✓ http://www.scritub.com/stiinta/informatica/METODA-DIVIDE-ET-IMPERA25186243.php
- ✓ *Anatol Gremalschi*, Informatică Manual pentru clasa a 11-a, Știința, 2014