PROIECT IDIVIDUAL

LA INFORMATICĂ

TEMA: DESPARTE ȘI STĂPÂNEȘTE

A REALIZAT: Apostol Cristina, clasa a XI-a ”C”

A VERIFICAT: Maria Guțu

Chișinău 2019

Informaţie generală:

Metoda desparte şi stăpîneşte (în latină divide et impera) 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 DIVIDE ET IMPERA 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 rara.

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;
```

Avantaje:

Programele elaborate în baza metodei desparte şi stăpîneşte sînt simple, iar timpul de execuţie este relativ mic.

Dezavantaje:

Cu regret, această metodă nu este universală, întrucît ea poate fi aplicată numai atunci cînd prelucrarea cerută admite divizarea problemei curente în subprobleme de dimensiuni mai mici. De obicei, această proprietate nu este indicată explicit în enunţul problemei şi găsirea ei, dacă există, cade în sarcina programatorului.

APLICAȚII

1.Problema turnurilor din Hanoi (Denumirea problemei provine de la o veche legendă hindusă conform căreia după mutarea celor 64 de discuri va veni sfîrșitul lumii.)

Fie trei tije verticale notate A,B,C .Pe tija A se gasesc asezate n discuri de diametre diferite ,in ordinea crescatoare a diametrelor,privind de sus in jos . Initial ,tijele B si C sunt goale .Sa se afiseze toate mutarile prin care discurile de pe tija A se muta pe tija B , in aceeasi ordine ,folosind ca tija de manevra C si resspectand urmatoarele reguli:

- la fiecare pas se muta un singur disc;
- un disc se poate aseza numai peste un disc cu diametrul mai mare .

Rezolvarea acestei probleme se bazeaza pe urmatoarele considerente logice :

- daca n=1 ,atunci mutarea este immediata AàB(mut discul de pe A pe B);
- daca n=2,atunci sirul mutarilor este : AàC,AàB,CàB;
- daca n>2 procedam astfel :
- mut (n-1)discuri AàC;
- mut un disc AàB ;
- mut cele (n-1)discuri CàB.

Observam ca problema initiala se descompune in trei subprobleme mai simple ,similare problemei initiale: mut (n-1)discuri AàC ,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 initial (pe care sunt dispuse discurile), tijele finale si tijele intermediare sunt diferite.Notam H(n,A,B,C)=sirul mutarilor a n discuri de pe A pe B, folosind C.

```
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.
```

2.Sortare rapida(quicksort)

Un tablou V se completeaza cu n elemente numere reale .Sa se ordoneze crescator folosind metoda de sortare rapida .

Solutia problemei se bazeaza pe urmatoarele etape implementate in programul principal:

- se apeleaza procedura "quick" cu limita inferioara li=1 si limita superioara ls=n;

- functia"poz" realizeaza mutarea elementului v[i] exact pe pozitia ce o va ocupa acesta in vectorul final ordonat ; functia"poz" intoarce (in k) pozitia ocupata de acest element;
- in acest fel ,vectorul V se imparte in doua parti : li .k-1 si k+1.ls;
- pentru fiecare din aceste parti se reapeleaza procedura"quick",cu limitele modificate corespunzator ;
- in acest fel ,primul element din fiecare parte va fi pozitionat exact pe pozitia finala ce o va ocupa in vectorul final ordonat (functia"poz");
- fiecare din cele doua parti va fi ,astfel ,inpartita in alte doua parti ;procesul continua pana cand limitele partilor ajung sa se suprapuna ,ceea ce indica ca toate elementele vectorului au fost mutate exact pe pozitiile ce le vor ocupa in vectorul final ;deci vectorul este ordonat ;
- in acest moment se produc intoarcerile din apelurile recursive si programul isi termina executia .

OBSERVAȚIE

- daca elementul se afla in stanga ,atunci se compara cu elementele din dreapta lui si se sar (j:=j-1) elementele mai mari decat el ;
- daca elementul se afla in dreapta ,atunci se compara cu elemente din stanga lui si se sar (i:=i+1) elementele mai mici decat el.

```
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);
  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.
```

3.Sortare prin interclasare (mergesort)

Tabloul unidimensional V se completeaza cu n numere reale. Sa se ordoneze crescator folosind sortare prin interclasare.

Sortarea prin interclasare se bazeaza pe urmatoarea logica :

- vectorul V se imparte ,prin injumatatiri succesive ,in vectori din ce in ce mai mici ;
- cand se ating vectorii de maxim doua elemente ,fiecare dintre acestia se ordoneaza printr-o simpla comparare a elementelor ;
- cate doi astfel de mini-vectori ordonati se interclaseaza succesiv pana se ajunge iar la vectorul V.

OBSERVATII

- mecanismul general de tip Divide et Impera se gaseste implementat in procedura "divi" ;
- o astfel de abordare a problemei sortarii unii vector conduce la economie de timp de calcul, deoarece operatia de interclasare a doi vectori deja ordonati este foarte rapida ,iar ordonarea independenta celor doua jumatati(mini- vectori) consuma in total aproximativ a doua parte din timpul care ar fi necesar ordonarii vectorului luat ca intreg .

```
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];
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.
```

4.Sortare prin insertie binara

Sa se ordoneze crescator un tablou unidimensional V de n numere reale ,folosind sortarea prin insertie binara .

Pentru fiecare element v[i] se procedeaza in patru pasi:

- se considera ordonate elementele v[1],v[2],..,v[i-1];
- se cauta pozitia k pe care urmeaza s-o ocupe v[i] intre elementele v[1],v[2],.,v[i-1] (procedura "poz" prin cautare binara);
- se deplaseaza spre dreapta elementele din pozitiile k,k+1,.,n(procedura "deplasare");
- insereaza elementul v[i] in pozitia k (procedura"deplasare");
- se obtine o succesiune de k+1 elemente ordonate crescator.

```
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];
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.
```

5.Maxim intr-un vector

Se citeste un vector cu n componente, numere naturale. Se cere sa se tipareasca valoarea maxima.

- daca i=j, valoarea maxima va fi v[i];
- contrar vom imparti vectorul in doi vectori: primul vector va contine componentele de la i la (i+j) div 2, al doilea vector va contine componentele de la (I+J) div 2 +1 la j; rezolvam problemele (aflam maximul pentru fiecare din ele) iar solutia problemei va fi data de valoarea maxima dintre rezultatele celor doua subprobleme.

```
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.
```

CONCLUZIE:

Algoritmii de tip Divide et Impera au buna comportare în timp ,daca se îndeplinesc urmatoarele condiții:

- dimensiunile subprogramelor (în care se imparte problema inițială) sunt aproximativ egale ("principiul balansării");
- lipsesc fazele de combinare a soluțiilor subproblemelor (cautare binara).

BIBLIOGRAFIE:

http://www.scritub.com/stiinta/informatica/METODA-DIVIDE-ET-IMPERA25186243.php

https://informaticacnet.wordpress.com/category/clasa-a-xi-a/metode-divide-et-impera/

http://www.creeaza.com/referate/informatica/Metoda-de-programare-DIVIDE-ET449.php