Proiect individual la informatică

Tema:

Metoda desparte și stăpânește

Realizat: Gherganov Alexandr

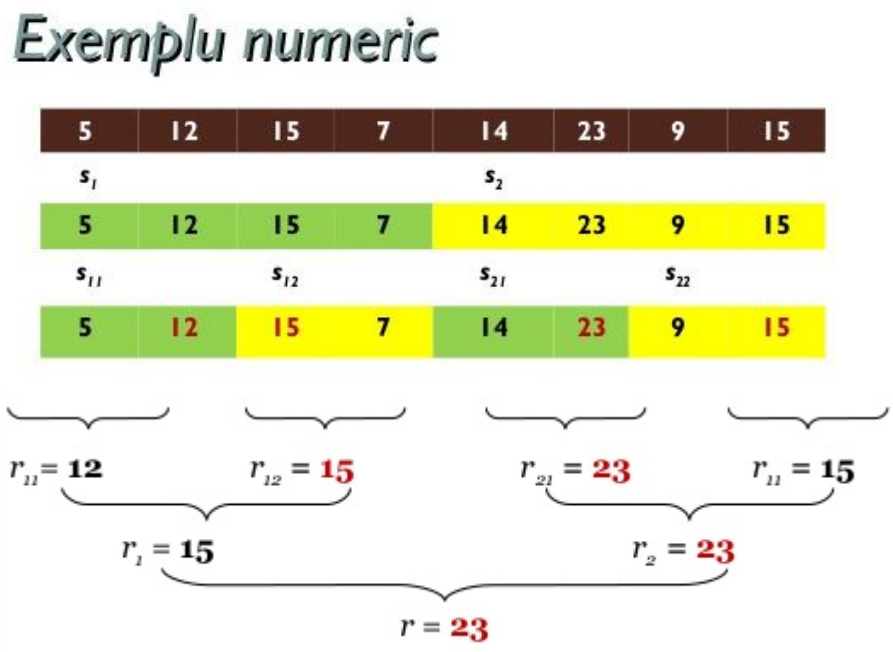
Prezentare generală:

Divide et impera se bazează pe principiul descompunerii problemei în două sau mai multe subprobleme (mai ușoare), care se rezolvă, iar soluția pentru problema inițială se obține combinând soluțiile subproblemelor. De multe ori, subproblemele sunt de același tip și pentru fiecare din ele se poate aplica aceeași tactică a descompunerii în (alte) subprobleme, până când (în urma descompunerilor repetate) se ajunge la probleme care admit rezolvare imediată.

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.

Divide et impera este o tehnică ce admite o implementare recursivă. Principiul general prin care se elaborează algoritmi recursivi este: "ce se întâmplă la un nivel, se întâmplă la orice nivel" (având grijă să asigurăm condițiile de terminare). Așadar, un algoritm prin divide et impera se elaborează astfel: la un anumit nivel avem două posibilități:

1. s-a ajuns la o problemă care admite o rezolvare imediată (condiția de terminare), caz în care se rezolvă și se revine din apel;
2. nu s-a ajuns în situația de la punctul 1, caz în care problema curentă este descompusă în (două sau mai multe) subprobleme, pentru fiecare din ele urmează un apel recursiv al funcției, după care combinarea rezultatelor are loc fie pentru fiecare subproblemă, fie la final, înaintea revenirii din apel.



În această imagine este reprezentat un șir de numere. Metoda desparte și stăpânește lucrează în următorul mod:

1. Împarte șirul în șiruri mai mici, până când rămân șiruri doar din 2 elemente.

2. Compară numerele din fiecare șir între ele, apoi între alte șiruri.

3. Obținerea rezultatului final, în baza unui număr mare de comparări.

Programul I:(căutarea elementului maxim dintr-un șir numeric)

```
Program P155;
 { Gasirea elementuli maximal prin metoda desparte şi
stăpineşte }
const nmax=100;
var A : array[1..nmax] of real;
 i, n : 1..nmax;
 x : real;
 function SolutieDirecta(i, j : integer) : boolean;
begin
 SolutieDirecta:=false;
 if (j-i<2) then SolutieDirecta:=true;
end; { SolutieDirecta }
procedure Prelucrare(i, j : integer; var x : real);
begin
 x:=A[i];
 if A[i]<A[j] then x:=A[j];
end; { Prelucrare }
procedure Combina(x1, x2 : real; var x : real);
begin
 x:=x1;
 if x1<x2 then x:=x2;
end; { Combina }
procedure DesparteSiStapineste(i, j : integer; var x : real);
var m : integer;
 x1, x2 : real;
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; { DesparteSiStapineste }
begin
 write('Daţi n='); readln(n);
 writeln('Daţi ', n, ' numere reale');
 for i:=1 to n do read(A[i]);
 writeln;
 DesparteSiStapineste(1, n, x);
 writeln('Numărul maximal x=', x);
 readln;
 readln;
end.
```

2. Căutare binară

Se citesc *n* numere întregi sortate crescător. De asemenea se citeşte un număr întreg nr. Se cere să se decidă dacă nr. se găseşte în şirul celor *n* numere citite.

Căutarea se efectuează între numerele reţinute de componentele de indice între valorile reţinute de două variabile *li* şi *ls* (iniţial *li* = 1 şi *ls* = n).

Fiind date *li* şi *ls* procedăm astfel:

- se calculează indicele componentei din mijloc, în cazul în care n este impar, sau a uneia din cele două plasate în mijloc, în cazul în care *n* este par (*k = (li+ls) div 2*);
- apar trei posibilităţi:
  - valoarea reţinută de componenta de indice calculat este egală cu nr (caz în care căutarea se termină cu succes);
  - valoarea reţinută de componenta de indice calculat este mai mică decât nr (caz în care numărul va fi căutat între componentele de indice *li = k+1* şi *ls*);
  - valoarea reţinută de componenta de indice calculat este mai mare decât nr (caz în care numărul va fi căutat între componentele de indice *li* şi *ls = k -1*).

Căutarea se termină când numărul a fost identificat sau când *li > ls* (căutare cu succes).

```
program cautare;
type vector=array[1..100] of integer;
var a:vector;
n,i,li,ls,k,nr:integer;
gasit:boolean;
begin
write('n=');readln(n);
for i:=1 to n do
begin
write('a[',i,']=');
readln(a[i]);
end;
write('nr=');readln(nr);
li:=1; ls:=n; gasit:=false;
repeat
k:=(li+ls) div 2;
if a[k]=nr
then
begin
writeln('gasit pe pozitia ',k);
gasit:=true;
```

```
end
else
if a[k]<nr
then li:=k+1
else ls:=k-1
until (li>ls) or gasit;
if li>ls then writeln('negasit');
readln
end.
```

3. Sortarea prin interclasare.

Algoritmul de **sortare prin intercasare** constituie un exemplu reprezentativ pentru folosirea metodei „*divide et impera*" în programare. Astfel, dacă avem de sortat un vector, atunci îl împărţim în două, sortăm – la fel – cele două părţi ale vectorului, apoi le interclasăm. Dacă şi vectorii rezultaţi după împărţire sunt destul de mari (mai mult decât un singur element), atunci procedăm la împărţirea şi a acestui vectori şi tot aşa.

Astfel, vom scrie o procedură SortInterclas(început, sfarsit: Integer); care va sorta vectorul *A* între poziţia *început* şi poziţia *sfarsit.* Procedura va determina poziţia din *mijloc* şi se va autoapela pentru *început* şi *mijloc*, apoi pentru *mijloc*+1 şi *sfârşit,* după care vor interclasa cele două părţi ale vectorului.

```
program SortarePrinInterclasare;
const max=10;
var a: array[1..max] of integer;i,n: 1..max;
procedure Interclaseaza(start,mijloc,finis:integer);
var b: array[1..max] of integer; i,j,k:integer;
begin
k:=start ; i:=start;j:=mijloc+1;
while (i<=mijloc) and (j<=finis) do
if a[i]<a[j] then begin b[k]:=a[i]; i:=i+1; k:=k+1 end
else begin b[k]:=a[j]; j:=j+1; k:=k+1 end;
if i<=mijloc then for j:=i to mijloc do begin b[k]:=a[j];
k:=k+1 end
else for i:=j to finis do begin b[k]:=a[i]; k:=k+1 end;
for i:=start to finis do a[i]:=b[i]
end;
procedure SortInterclas(inceput,sfarsit: Integer);
var centru:Integer;
begin
if inceput < sfarsit then
begin
centru:=(inceput+sfarsit) div 2;
SortInterclas(inceput,centru);
SortInterclas (centru+1, sfarsit );
Interclaseaza (inceput,centru,sfarsit)
end
end;
begin
Write('n='); readln(n);
for i:=1 to n do begin write('a[',i,']='); readln(a[i]) end;
SortInterclas(1,n); for i:=1 to n do write(a[i],','); readln
end.
```

4. Turnurile din Hanoi

Se spune că în Vietnamul antic, în Hanoi, erau trei turnuri, pe unul din ele fiind puse, în ordinea descrescătoare a diametrelor lor, mai multe (opt) discuri de aur. Din motive obiective, nişte călugări, care le aveau îngrijă, trebuiau să aşeze discurile pe cel de-al doilea turn, în aceeaşi ordine. Ei puteau să folosească, eventual, turnul al treilea, deoarece, altfel discurile nu ar fi avut stabilitate. Discurile puteau fi mutate unul câte unul. De asemenea, se renunţa din start la ideea de a aşeza un disc mai mare peste unul mai mic, deoarece exista riscul ca să se strice discurile, din cauza diferenţei mari de masă.

Deşi, aparent simplă, după câteva încercări, cu un prototip în faţă, se va constata că problema nu e banală. Însă este posibilă o rezolvare optimă a ei (cu numai mutări, deci 255, pentru n=8), folosind tehnica recursivă „divide et impera".

Problema este de a muta n discuri de la turnul 1 la turnul 2. Pentru a o rezolva, să vedem cum se mută, în general, *m* discuri de la un turn *p* la un turn *q*. Se mută primele *m*+1 discuri de pe *r*, *r* fiind turnul auxiliar, apoi singurul disc rămas pe *p*(discul cel mare) se mută de pe *p* pe *q,* după care cele *m*-1 discuri sunt mutate de pe *r* pe *q*. Fireşte, mutarea celor *m*-1 discuri este corectă, deoarece existenţa discurilor de diametre mai mari, la bazele celor trei turnuri nu afectează cu nimic mutările discurilor mai mici.

În cazul limită *m*=1, avem doar o mutare a discului din vârful turnului *p* spre *q.*

Programul de mai jos soluţionează (cu animaţie) problema descrisă, pentru *n*=8. Procedura de bază este *Han*, iar celelalte proceduri sunt pentru mişcarea discului curent. Există şi două proceduri cu structură inedită. Ele sunt scrise în limbaj de asamblare. Folosind întreruperea 10*h*, realizează ascunderea, respectiv reafişarea cursorului text.

```
program TurnurileDinHanoi;
uses crt;
const pauza=10; forma=#219;
varf:array[1..3] of byte=(13,22,22);
procedure HideCursor; assembler;
procedure ShowCursor; assembler;
function ColTija(tija:byte):byte;
begin ColTija:=24*tija-8 end;
procedure MutaDreapta(disc, tija1, tija2: byte);
var i,k:byte;
begin
for i:=ColTija(tija1)-disc to Pred(ColTija(tija2)-disc) do
begin
Delay(Pauza); if KeyPressed then Halt(1);
```

```
gotoxy(i,3); for k:=0 to disc do Write(' ');
gotoxy(i+1,3); for k:=0 to 2*disc do Write(forma);
end
end;
procedure MutaStanga(disc, tija1, tija2 :byte);
var i,k:byte;
begin
for i:=ColTija(tija1)-disc downto succ(ColTija(tija2)-disc) do
begin
Delay(Pauza); if KeyPressed then Halt(1);
gotoxy(i,3); for k:=0 to disc do Write(' ');
gotoxy(i-1,3); for k:=0 to 2*disc do Write(forma);
end
end;
procedure Coboara(disc, tija:byte);
var i,k:byte;
begin
for i:=3 to Pred(Varf[tija]-1) do
begin
Delay(Pauza); if KeyPressed then Halt(1);
gotoxy(ColTija(tija)-disc,i);
for k:=0 to disc do Write(' ');
gotoxy(ColTija(tija)-disc,i+1);
for k:=0 to 2*disc do Write(forma);
end;
Dec(Varf[tija])
end;
procedure Ridica(disc, tija:byte);
var i,k:byte;
begin
for i:=Varf[tija] downto 4 do
begin
Delay(Pauza); if KeyPressed then Halt(1);
gotoxy(ColTija(tija)-disc,i);
for k:=0 to disc do Write(' ');
gotoxy(ColTija(tija)-disc,i-1);
for k:=0 to 2*disc do Write(forma);
end;
Inc(Varf[tija])
end;
procedure Muta(disc, tija1, tija2:byte);
begin
Ridica(disc,tija1);
if (tija1<tija2) then
MutaDreapta(disc,tija1,tija2);
Coboara(disc, tija2)
end;
procedure Han(n, tija1, tija2, tija3:byte);
begin
if (n=1) then Muta(1,tija1,tija2)
else
begin
Han(n-1, tija1, tija3, tija2);
Muta(n, tija1,tija2);
Han(n-1, tija3, tija2,tija1)
end
end;
procedure Initializari;
var k,disc:byte;
begin
HideCursor; Clrscr;
for disc:=1 to 9 do
begin
gotoxy(ColTija(1)-disc,varf[1]+disc-1);
```

```
for k:=0 to 2*disc do
write(forma);
end
end;
begin
Initializari; gotoxy(28,1); writeln('- Turnurile din Hanoi -');
Han(8,1,2,3); ShowCursor;
readln
end.
```

5. Găsirea elementului minimal :

```
Program P155;
 { Gasirea elementuli minimal prin metoda desparte şi
stăpineşte }
const nmax=100;
var A : array[1..nmax] of real;
 i, n : 1..nmax;
 x : real;
 function SolutieDirecta(i, j : integer) : boolean;
begin
 SolutieDirecta:=false;
 if (j-i<2) then SolutieDirecta:=true;
end; { SolutieDirecta }
procedure Prelucrare(i, j : integer; var x : real);
begin
 x:=A[i];
 if A[i]>A[j] then x:=A[j];
end; { Prelucrare }
procedure Combina(x1, x2 : real; var x : real);
begin
 x:=x1;
 if x1>x2 then x:=x2;
end; { Combina }
procedure DesparteSiStapineste(i, j : integer; var x : real);
var m : integer;
 x1, x2 : real;
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; { DesparteSiStapineste }
begin
 write('Daţi n='); readln(n);
 writeln('Daţi ', n, ' numere reale');
 for i:=1 to n do read(A[i]);
 writeln;
 DesparteSiStapineste(1, n, x);
 writeln('Numărul minimal x=', x);
 readln;
 readln;
end.
```

Bibliografie:

file:///C:/Users/user/Downloads/XI_Informatica%20(in%20limba%20romana)%20(4).pdf

https://ro.wikipedia.org/wiki/Divide_et_impera_(informatic%C4%83)

https://www.slideshare.net/LuminiaMihailov/metoda-divide-et-impera-47939776

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