Referat

Tehnici de elaborae a algoritmilor

Bordei Dan-Nicolae

Clasa a XI-a C

***Cuprins***

[Introducere 2](#_Toc36670281)

[Iterativitate sau recursivitate. 2](#_Toc36670282)

[1.1.1. Iterativitate 2](#_Toc36670283)

[1.1.2. Recursivitate 3](#_Toc36670284)

[1.1.3. Exemple de probleme recursive 3](#_Toc36670285)

[1.1.4. Exemple de programe iterative 5](#_Toc36670286)

[Concluzie 6](#_Toc36670287)

[Bibliografie 6](#_Toc36670288)

### Introducere

Un algoritm, atât în matematică cât şi informatică este o metodă sau o procedură de calcul, alcătuită din paşii elementari necesari pentru rezolvarea unei probleme sau categorii de probleme. De obicei algoritmii se implementează în mod concret prin programarea adecvată a unui calculator, sau a mai multora.

Multe probleme de o importanță practică pot fi rezolvare cu ajutorul unor metode standard denumite thenici de programare: Recursia; Trierea; Metoda reluării; Metode euristice.

# Iterativitate sau recursivitate.

În funcţie de modul de implementare, un algoritm poate fi: recursiv - face uz de sine însuşi, în mod repetat, și iterativ – repetitiv.

Orice algoritm recursiv poate fi transcris într-un algoritm iterativ și invers. Alegerea techicii de programare ­ iterativitate sau recursivitate­ ține, de asemenea, de competența programatorului. Evident, această alegere trebuie făcută luînd în considerare avantajele și neajunsurile fiecărei metode, care variază de la caz la caz.

## 1.1.1. Iterativitate

Iterativitatea este procesul prin care rezultatul este obţinut ca urmare a execuţiei repetate a unui set de operaţii, de fiecare dată cu alte valori de intrare. Numărul de iteraţii poate fi necunoscut sau cunoscut, dar determinabil pe parcursul execuţiei. Metoda de repetivitate este cunoscută sub numele de ciclu şi poate fi realizată prin utilizarea următoarelor structuri repetitive: ciclul cu test iniţial, ciclul cu test final, ciclul cu număr finit de paşi. Indiferent ce fel de structură iterativă se foloseşte este necesar ca numărul de iteraţii să fie finit.

Iteratia este execuţia repetată a unei porţiuni de program pînă la îndeplinirea unei condiţii(while, for etc.)

## 1.1.2. Recursivitate

Recursivitatea este procesul iterativ prin care valoarea unei variabile se determină pe baza uneia sau a mai multora dintre propriile ei valori anterioare. Structurile recursive reprezintă o alternativă de realizare a proceselor repetitive fără a utiliza cicluri.

În general, elaborarea unui program recursiv este posibilă numai atunci cînd se respectă următoarea regulă de consistență: soluția problemei trebuie să fie direct calculabilă ori calculabilă cu ajutorul unor valori direct calculabile. Cu alte cuvinte, defininirea corectă a unui algoritm recursiv presupune că în procesul derulării calculelor trebuie să existe: ­ cazuri elementare, care se rezolvă direct; ­ cazuri care nu se rezolvă direct, însă procesul de calcul în mod obligatoriu progresează spre un caz elementar.Orice algoritm recursiv poate fi transcris într-un algoritm iterativ și invers. Alegerea techicii de programare ­ iterativitate sau recursivitate­ține, de asemenea, de competența programatorului. Evident, această alegere trebuie făcută luînd în considerare avantajele și neajunsurile fiecărei metode, care variază de la caz la caz.

## 1.1.3. Exemple de probleme recursive

1.function Fact(n:Natural):Natural;

begin

if n=0 then Fact:=1

else Fact:=n:=Fact(n-1)

end;

2.Function Factorial(n : Integer) : LongInt;

Var

Result : LongInt;

i : Integer;

Begin

Result := n;

If (n <= 1) Then

Result := 1

Else

For i := n-1 DownTo 1 do

Result := Result \* i;

Factorial := Result;

End;

3. Function Factorial(n : Integer) : Integer;

Var

Result : Integer;

Begin

If n = 1 Then

Factorial := 1

Else

Factorial := n\*Factorial(n-1);

End;

4. function suma\_patratelor(n:integer):longint;

begin

if n=1 then suma\_patratelor:=1 else

begin

suma\_patratelor:=n\*n+suma\_patratelor(n-1);

end;

end;

5. Program Fibonacci;

Type Natural = 0.. MaxInt;

Var nr, i : Natural;

Function Fib(n: Natural):Natural;

Begin

If n=0 then Fib:=0 Else

if n=1 then Fib:=1 Else

Fib:=Fib(n-1) + Fib (n-2);

End;

Begin

Writeln (‘i=’);

Readln (i);

nr:= Fib(i);

writeln (‘Fib(’, i, ‘)=’, nr);

end.

## 1.1.4. Exemple de programe iterative

1. program fact\_iter;

var n,s,i: integer;

begin

  write('n = ');

  readln(n);

  i:=1;

  s:=1;

  while i<=n do begin

s:=s\*i;

    i:=i+1;

  end;

  writeln(n,'! = ',s);

  readln;

end.

2. var

letter:char;

begin

for letter := 'a' to 'z' do

write(letter);

writeln;

for letter := 'y' downto 'a' do

write(letter); edn;

3. var

count:integer;

begin

for count := 10 downto 1 do

writeln(count:5);

writeln('blast off!!!')

end;

4. var

count:integer;

begin

for count :=1 to 5 do

writeln('Welcome back!');

end;

5. var count,limit:integer;

Begin

count:=0;

limit:=15;

while count <= limit do

begin

count:=count+1;

write(count:3)

end;

writeln;

writeln('end of while-loop')

end;

### Concluzie

În concluze fiecare din tehnici sunt foarte utile și importante în programare. De asemenea abilitatea de a alege între tehnici poate fi necesară, de aceea , înainte de a începe un program, trebuie să ne asigurăm daca iterativitatea sau recursia va fi mai productivă, iar după cum am învățat, iterativitea este mai productiva în cazul programelor mai simple și mai ușoare, în caz contrar, unde programul necesită un număr mare de iterări cu un set de condiții specifice, atunci recursia va fi mai eficientă.

### Bibliografie

1. <https://www.codeit-project.eu/differences-between-iterative-and-recursive-algorithms/>
2. <https://www.pascal-programming.info/articles/recursion.php>
3. <https://alg24.com/en/factorial-in-pascal-iteratively>
4. Manual