IPLT “Spiru Haret”

Iterativitate si Recursivitate

Referat elaborat de
Guzun Laura, clasa 11 “D”

Martie,2020

Cuprins

1. Aspecte teoretice

1.1. Iterativitate

Iterativitatea este procesul prin care rezultatul este obținut ca urmare a execuției repetate a unui set de operații, de fiecare dată cu alte valori de intrare.

Iteratia este execuția repetată a unei porțiuni de program pînă la îndeplinirea unei condiții (while,for, etc.). [8]

Numărul de iterații poate fi necunoscut sau cunoscut, dar determinabil pe parcursul execuției. Metoda de repetivitate este cunoscută sub numele de ciclu (loop) şi poate fi realizată prin utilizarea următoarelor structuri repetitive: ciclul cu test inițial, ciclul cu test final, ciclul cu număr finit de paşi. Indiferent ce fel de structură iterativă se foloseşte este necesar ca numărul de iterații să fie finit.[9]

1.2. Recursivitatea

Pe parcursul dezvoltării informaticii s-a stabilit că multe probleme de o reală importanță practică pot fi rezolvate cu ajutorul unor metode standard, denumite tehnici de programare: recursia, trierea, metoda reluării, metodele euristice ş.a.
Una din cele mai răspîndite tehnici de programare este **recursia**. [1]

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.[9]

În matematică și informatică, **recursivitatea** sau **recursia** este un mod de a defini unele funcții. Funcția este recursivă, dacă definiția ei folosește o referire la ea însăși,o situatie in care un subprogram se autoapeleaza fie direct fie prin intermediul altei functii sau proceduri.

Nu toate funcțiile matematice pot fi definite recursiv; cu alte cuvinte există și funcții nerecursive.

Recursivitatea funcționează prin definirea unuia sau a mai multe cazuri de bază, foarte simple, și apoi prin definirea unor reguli prin care cazurile mai complexe se reduc la cazuri mai simple.

Un exemplu de recursivitate este în definirea formală a numerelor naturale din cadrul teoriei mulțimilor:

- baza recursiei este faptul că 1 este număr natural;
- în plus, orice număr natural are un succesor, care este de asemenea un număr natural.

Un alt exemplu ar fi definirea conceptului de strămoș al unei persoane:

- Un părinte este strămoșul copilului. ("Baza"')
- Părinții unui strămoș sunt și ei strămoși ("Pasul de recursie"). [6]

În general, elaborarea unui program recursiv este posibilă numai atunci cînd se respectă următoarea **regulă de consistență***: soluția problemei trebuie să fi e direct calculabilă ori calculabilă cu ajutorul unor valori direct calculabile*.
Cu alte cuvinte, definirea corectă a unui algoritm recursiv presupune că în procesul derulării calculelor trebuie să existe:

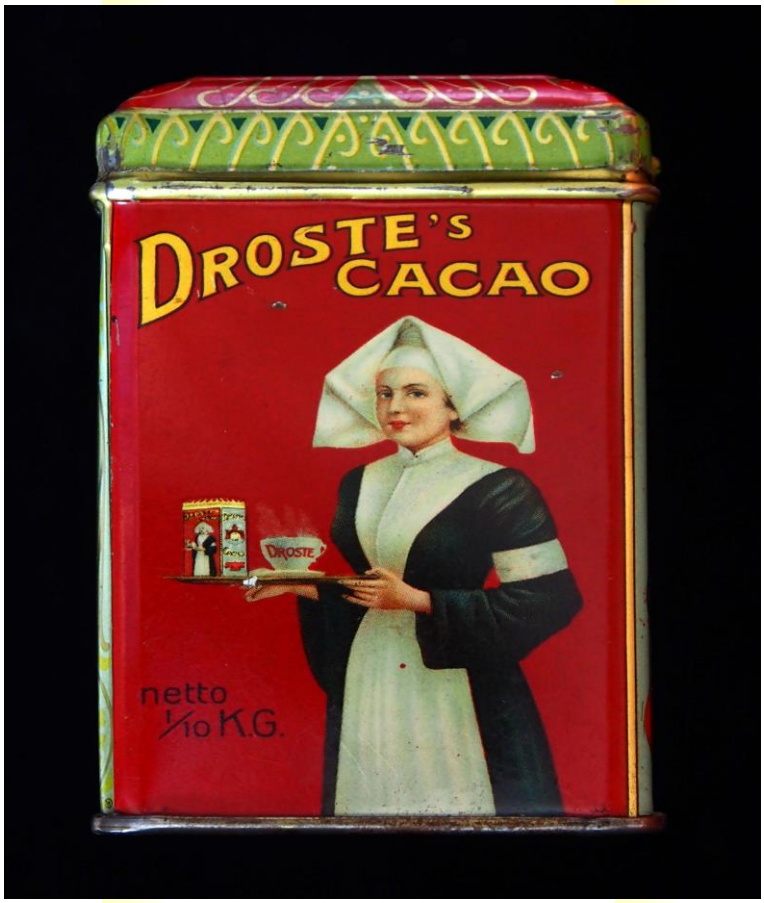
– cazuri elementare, care se rezolvă direct;
– cazuri care nu se rezolvă direct, însă procesul de calcul în mod obligatoriu progresează spre un caz elementar. [1]

Clasificare: **Recursivitatea** este de două tipuri:

- În cazul în care auto-apelul se realizează în cadrul aceleiași funcții, recursivitatea este **directă** (A->A).
- Atunci când auto-apelul se realizează prin intermediul altei funcții, recursivitatea este **indirectă** (A->B->...->A). [10]

1.2.1 Efectul Droste

Efectul Droste , cunoscut în cadrul artelor ca *mise en abyme*, reprezintă efectul prin care o imagine apare în mod **recursiv** în interiorul său, de obicei în locuri în care o cu totul altă imagine ar fi de așteptat să apară.
Efectul a fost denumit după o marcă olandeză de cacao, a cărei ilustrație de pe cutia de depozitare a fost realizată de Jan Misset în 1904. [7]



1.3 Studiul comparativ

Orice algoritm recursiv poate fi transcris intr-un algoritm iterativ si invers.
Alegerea tehnicii de programare - **iterativitatea sau recursivitatea** - tine de competenta programatorului. Evident, această alegere trebuie făcută, luand în considerare *avantajele şi dezavantajele* fiecărei metode, care variază de la caz

la caz. Pentru exemplificare, în tabelul de mai jos sunt prezentate rezultatele unui studiu comparativ al ambelor tehnici de programare în cazul prelucrării automate a textelor.[1]

Nr. criteriului	Caracteristici	Iterativitate	Recursivitate
1.	Necesarul de memorie	mic	mare
2.	Timpul de executie	acelasi	acelasi
3.	Structura programului	complicata	simpla
4.	Volumul de munca necesara scrierii program.	mare	mic
5.	Testarea si depanarea programelor	simpla	complicata

- **Abordare:** În abordarea recursivă, funcția se solicită până când condiția este îndeplinită, în timp ce în abordarea iterativă se repetă o funcție până când condiția nu reușește.
- **Utilizarea programelor de construcție**: Algoritmul recursiv utilizează o structură de ramificație, în timp ce algoritmul iterativ utilizează o construcție looping.
- **Eficiența timpului și a spațiului**: soluțiile recursive sunt adesea mai puțin eficiente din punct de vedere al timpului și spațiului, comparativ cu soluțiile iterative.
- **Test de terminare:** Iterația se termină atunci când condiția continuă a buclăului eșuează; recursiunea se termină când se recunoaște un caz de bază.
- **Invitație infinită:** Se produce o buclă infinită cu iterație dacă testul de continuare a buclării nu devine fals; se produce recurența infinită dacă etapa de recurs nu reduce problema într-o manieră care converge în cazul de bază.[5]

2. Probleme rezolvate

2.1. Prin recursie [4]

- Determinarea numărului de cifre dintr-un numar n:
  - pentru n∈{0,1,2,3,4,5,6,7,8,9} evident numărul de cifre este 1;
  - pentru n ≥ 10, se poate gândi că numărul de cifre a lui n este 1+ numarul de cifre a numărului obținut din n, din care am tăiat ultima cifră (de fiecare dată se taie cifra unităților); tăierea ultimei cifre se obține prin împărtirea întreagă la 10.

Exemplu: NrCifre(123)= 1+ NrCifre(12)= 1+(1+NrCifre(1))=1+1+1=3.

```
type Natural = 0..MaxLongInt;
function NrCifre(n:Natural) :Natural;
begin
if n in [0..9]
then NrCifre:=1
else NrCifre:=1 + NrCifre(n div 10);
end;
```

- Determinarea sumei cifrelor numărului n:
  - pentru n ∈{0,1,2,3,4,5,6,7,8,9} evident suma cifrelor este n;
  - pentru n ≥ 10, se poate gândi că suma cifrelor lui n este ultima cifră + suma cifrelor numărului obținut din n, din care am tăiat ultima cifră (de fiecare dată se taie cifra unităților); tăierea ultimei cifre se obține prin împărtirea întreagă la 10 iar ultima cifră se poate obține prin restul împărțirii întregi a lui n la 10.

Exemplu: SumaCifre(123)=3+SumaCifre(12)=3+(2+SumaCifre(1))=3+2+1=6.

```
type Natural = 0..MaxLongInt;
function SumaCifre(n:Natural):Natural;
begin
if n in [0..9]
then SumaCifre:= n
else SumaCifre:= n mod 10 + SumaCifre(n div 10);
end;
```

- Determinarea produsului cifrelor numărului natural n:
  - pentru n∈{0,1,2,3,4,5,6,7,8,9} evident produsul cifrelor este n;
  - pentru n ≥ 10, se poate gândi că produsul cifrelor lui n este ultima cifră * produsul cifrelor numărului obținut din n, din care am tăiat ultima cifră (de fiecare dată se taie cifra unităților); tăierea ultimei cifre se obține prin împărtirea întreagă la 10 iar ultima cifră se poate obține prin restul împărțirii întregi a lui n la 10.

Exemplu:
ProdusCifre(123)= 3* ProdusCifre(12)= 3*(2*ProdusCifre(1))=3*2*1=6.

```
type Natural = 0..MaxLongInt;
function ProdusCifre(n:Natural):Natural;
begin
if n in [0..9] then ProdusCifre:= n
else ProdusCifre:= n mod 10 * ProdusCifre(n div 10);
end;
```

- Determinarea oglinditului numarărului n (oglinditul lui 123 este 321);
  - pentru n∈ {0,1,2,3,4,5,6,7,8,9} evident oglinditul lui n este n;
  - pentru n ≥ 10, se poate gândi că oglinditul lui n este ultima cifră * 10NrCifre(n/10) adunată cu oglinditul lui n / 10
  -

Exemplu:
Oglinda(123)=3*102+(Oglinda(12))=300+(2*101+Oglinda(1))=300+20+1=321.

```
type Natural = 0..MaxLongInt;
function NrCifre(n:Natural):Natural;
begin
if n in[0..9] then NrCifre:=1
else NrCifre:=1+NrCifre(n div 10);
end;
function Oglinda (n:Natural): Natural;
var aux:Natural;
begin
if n in [0..9] then Oglinda:= n
else
begin
aux :=NrCifre(n div 10);
Oglinda:=trunc((n mod 10)*exp(aux*ln(10))) + Oglinda(n div 10);
end;
end;
```

- Determinarea cifrei de pe poziția poz (de la dreapta la stânga) a unui număr natural n.
  - se scad numărul de împărțiri ale lui n la 10 din variabila poz; dacă am ajuns la 0 se returnează (n mod 10) .

```
type Natural = 0..MaxLongInt;
function Pozitia (n,Poz:Natural): byte;
begin
if Poz=1
then Pozitia:=n mod 10
else
begin
Poz:=Poz-1;
Pozitia:=Pozitia(n div 10,Poz);
end;
end;
```

2.2. Prin iteratie

- Să se scrie un program care citește și afișează simbolul introdus de la tastatura și se oprește când simbolul '+' este introdus. [3]

```
Program P1;
Var c : char;
begin
repeat
writeln('Dati simbolul c; pentru a opri programul, introduceti "+".');
readln(c);
writeln(c); // Instrucțiunea repeat, va citi si afisa simboluri până când utilizatorul
va introduce "+", iar condiția de la until va deveni adevărată și se va ieși din
ciclu//
until c = '+';
end.
```

- Să se scrie un program care calculeaza media aritmetica a numerelor pozitive citite de la tastatura. [2]

```
Program P2;
var x, suma : real;
n : integer;
begin
n:=0;
suma:=0;
writeln('Dati numere pozitive:');
readln(x);
while x>0 do
begin
n:=n+1;
suma:=suma+x;
readln(x);
end; // Datorita instructiunii while , ciclul se va repeta atata timp cat conditia
x>0 devine falsa //
writeln('Ati introdus ', n, ' numere pozitive.');
if n>0 then writeln('media=', suma/n)
else writeln('media=*****'); // media nu poate fi un numar negativ//
```

```
readln;
end.
```

- Scrieți un program care calculează și afișează rezultatul operației x^n, unde x este un număr real, iar n este un numar întreg. [3]

```
Program P3;
Var x, p:real;
n, i: integer;
begin
write('x='); readln(x);
write('n='); readln(n);
p:=1; // astfel încât pentru i=1 , p=x//
for i:=1 to n do p:=p*x; //p se va înmulți cu x de n ori, obținându-se
x^n //
writeln('x^n=', p);
end.
```

- Să se scrie un program care citește de la tastatura siruri arbitrare de caractere si afiseaza la ecran numarul de spatii din aceste siruri. Derularea programului se incheie dupa introducerea sirului 'Sfarsit'. [2]

```
Program P4;
var S : string;
i, j : integer;
begin
writeln('Dati siruri de caractere');
repeat  //operatia se va repeat pana conditia de la until devine adevarata//
readln(S);
i:=0;
for j:=1 to length(S) do  //pe toata lungimea sirului se calculeaza numarul de
spatii//
if S[j]=' ' then i:=i+1;
writeln('Numarul de spatii= ', i);
until S='Sfarsit';
end.
```

- Sa se scrie un program ce calculeaza si afiseaza la ecran suma componentelor variabilei x de tip vector . Componentele vectorului se citesc de la tastatura. [2]

```
Program P5;
type Vector = array [1..5] of real;
var x : Vector;
i : integer;
s : real;
begin
writeln('Dati 5 numere');
for i:=1 to 5 do readln(x [i]);  //Citirea componentelor prin parcurgerea intregului vector//
writeln('Ati introdus');
for i:=1 to 5 do writeln(x [i]);  //Afisarea componentelor prin parcurgerea intregului vector //
s:=0;
for i:=1 to 5 do s:=s+x [i];  //Calcularea smei pe tot parcursul vectorului//
writeln('Suma=', s);
readln;
end.
```

3. Concluzii

In concluzie, atat iterativitatea, cat si recursivitatea , se bazeaza pe o structura de control: iteratia foloseste o structura de repetitie , iar recursia foloseste o structura de selectie. Un algoritm iterativ va folosi instructiuni precum *for, while si repeat* pentru a repeta aceiasi pasi, in timp ce un algoritm recursiv va folosi o functie ce se autoapeleaza pana cand conditia de oprire este indeplinita.

Un algoritm iterativ va fi mai rapid decat algoritmul recursiv din cauza cheltuielilor generale cum ar fi functiile de apel si stivele de inregisrare a datelor. De multe ori, algoritmii recursivi nu sunt eficienti deoarece necesita mai mult spatiu si timp.

Algoritmii recursivi sunt utilizati in cea mai mare parte pentru a rezolva probleme complicate atunci cand aplicarea lor este usoara si eficienta. Iterativitatea este utilizata pe scara larga, eficienta si usora. [5]

4. Bibliografie

1) Anatol Gremalschi, Manual de clasa a 11-a, editura Stiinta, 2014
2) Anatol Gremalschi, Manual de clasa a 10-a, editura Stiinta,2012
3) http://staff.cs.upt.ro/~chirila/teaching/upt/id12-sda/lectures/ID-SDA-Cap5.pdf
4) http://www.cs.ubbcluj.ro/~vcioban/InformaticaDidactica/RecursivitatealnProgramare.pdf
5) https://www.codeit-project.eu/ro/differences-between-iterative-and-recursive-algorithms/
6) https://ro.wikipedia.org/wiki/Recursivitate
7) https://ro.wikipedia.org/wiki/Efectul_Droste
8) https://www.scribd.com/document/337119802/Iterativitatea
9) https://prezi.com/hwzqekzxc509/iterativitate-sau-recursivitate/
10) http://ler.is.edu.ro/~ema/proiecte/soft/2014/recursivitate/momentul1.html