MINISTERUL EDUCAȚIEI ȘI CERCETĂRII
ROMÂNIA
UNIVERSITATEA DE MEDICINĂ, FARMACIE, ȘTIINȚE ȘI TEHNOLOGIE „GEORGE EMIL PALADE" DIN TÂRGU MUREȘ
FACULTATEA DE INGINERIE ȘI TEHNOLOGIA INFORMAȚIEI

Algoritmi fundamentali

Curs 6

Recursivitate

Dr. ing. Kiss Istvan

istvan.kiss@umfst.ro

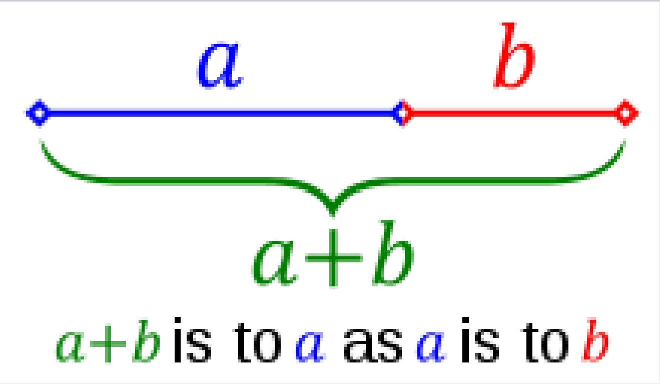
Cuprins

1. Recursivitatea
2. Exemplificare
   1. Sectiunea de aur
   2. Sirul lui Fibonacci
   3. Fractali
3. Algoritmul recursiv
   1. Functii recursive
   2. Probleme elementare…
4. Comparatie algoritm recursiv si iterativ
5. Probleme

1. Recursivitatea

- Să ne imaginăm un copac. O ramura a copacului este formată din mai multe ramuri fiecare la rândul ei fiind formată din ramuri mai mici, acestea sunt formate din ramuri şi mai mici ...
- Matematic: provine din **relatia de recurenta:**
  - $\mathbf{a_n=a_{n-1}+1,\ a_0=0;}$
- S-a aplicat prima data in informatica in anii 80.
- In sens informatic: un subalgoritm se autoapeleaza.
- **Recursivitate directa**: proprietatea functiilor de a se *autoapela*.

2. Exemplificare



1. Sectiunea de aur

$$\frac{a+b}{a} = \frac{a}{b} \stackrel{\text{def}}{=} \varphi,$$

- Raportul de aur

$$\varphi = \frac{1+\sqrt{5}}{2} = 1.6180339887\ldots.$$

- Demonstratie

$$\frac{a+b}{a} = 1 + \frac{b}{a} = 1 + \frac{1}{\varphi}.$$

$$1 + \frac{1}{\varphi} = \varphi.$$

$$\varphi + 1 = \varphi^2$$

$$\varphi^2 - \varphi - 1 = 0.$$

$$\varphi = \frac{1+\sqrt{5}}{2} = 1.6180339887\ldots.$$

$$\varphi = [1; 1, 1, 1, \ldots] = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \ddots}}}$$

2. Exemplificare

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, ...

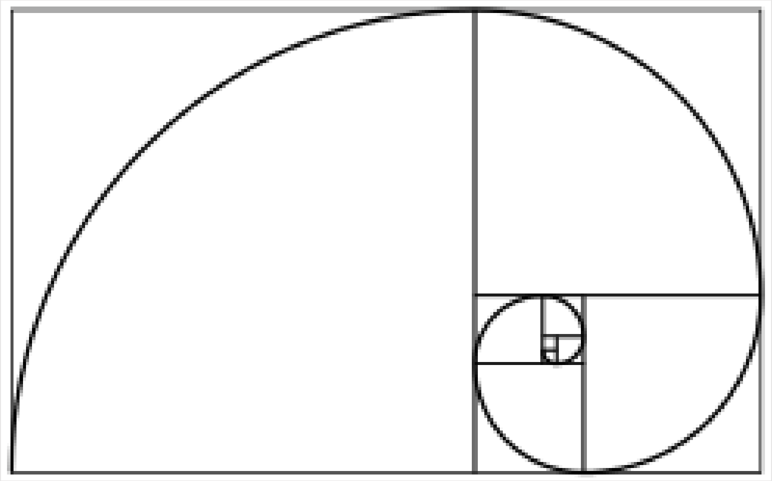
- 2. Sirul lui Fibonacci: raportul intre doi termeni consecutive tinde catre raportul de aur

$$F(n) = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}} = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}.$$

$$\lim_{n\to\infty} \frac{F(n+1)}{F(n)} = \varphi.$$

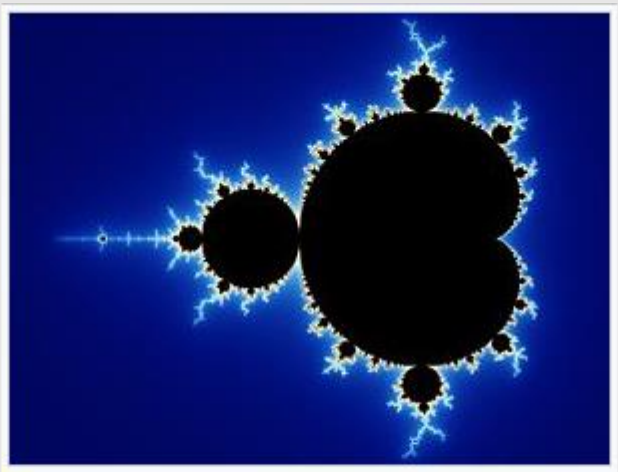
$$\varphi^{n+1} = \varphi^n + \varphi^{n-1}.$$

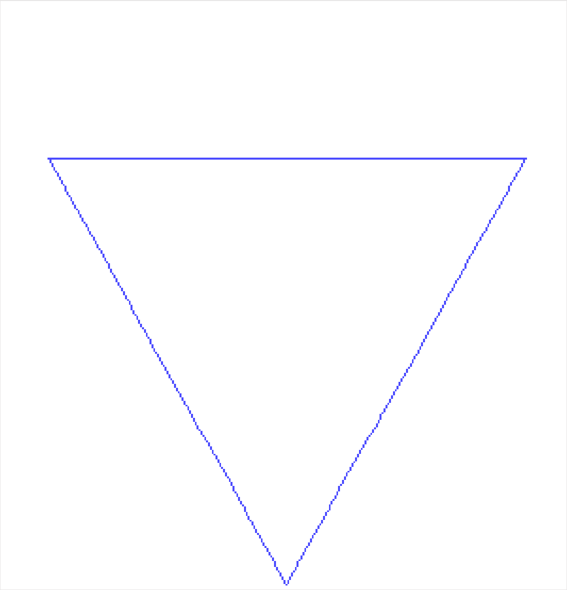
  - Spirala Fibonacci

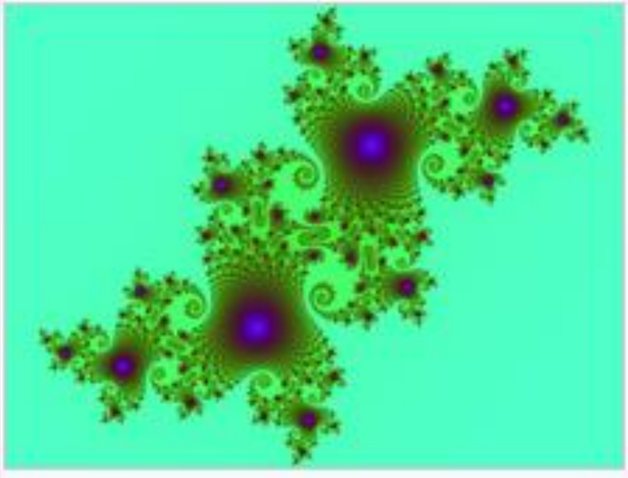


2. Exemplificare

- 3. Fractali: un **fractal** este o figură geometrică fragmentată sau frântă care poate fi divizată în părți, astfel încât fiecare dintre acestea să fie (cel puțin aproximativ) o copie miniaturală a întregului.







https://ro.wikipedia.org/wiki/Fractal

3. Algoritmul recursiv

- *Algoritmul recursive funcționează prin definirea unuia sau a mai multe cazuri de bază, foarte simple (cazuri triviale), și apoi prin definirea unor reguli prin care cazurile mai complexe se reduc la cazuri mai simple.*
- O functie este recursiva, daca se autoapeleaza.
- Orice funcţie recursivă **trebuie să conţină o condiţie de reluare a apelului recursiv**(sau de oprire).
- Fără această condiţie, funcţia teoretic se reapelează la infinit!!!

3.1. Functii recursive

- Pentru a implementa o funcţie recursivă, trebuie să:
  - **Identificăm relaţia de recurenţa (ceea ce se execută la un moment dat şi se reia la fiecare reapel)**
  - **Identificăm conditiile de oprire ale reapelului**

- **Structura unei funcţii recursive:**

Functie nume_funcţie(*lista de parameri*)
    .... {se exec. de n ori in ordine}
    daca (*condiţie de oprire*) **atunci**
        solutie triviala; {se exec. odata}
        ....
    altfel
        .... {se exec. de n-1 ori in ordine}
        nume_funcţie(*lista de parametri- pe baza rel. recurenta*);
        .... {se exec. de n-1 ori in ordine inversa}
    sf.daca
    .... {se exec. de n ori in ordine inversa}
Sf.functie

3.1. Functii recursive

- **Exista doua categorii de functii recursive**
  - **Directe, cu un singur sau mai multe apeluri recursive.**
  - **Indirecte, care se autoapeleaza prin intermediul altor funcţii.**

```
Functie b(lista de parameri)
        .................
        a(lista de parametri); //funcţia b apelează funcţia a
        ..................
Sf.functie

Functie a(lista de parameri) //funcţia a este recursivă şi apelează funcţia b
    .................
    Daca (condiţie de continuare)
        .................
        b(lista de parametri);
    Sf.daca
    ..................
}
```

3.2. Probleme elementare

- 1. Calculul permutarilor, aranjamentelor si combinarilor: (*necesita calcul factorial*)

$$P_n = n!\ ;\ A_n^m = \frac{n!}{(n-m)!}\ ;\ C_n^m = \frac{n!}{m!\cdot(n-m)!}$$

Relatia de recurenta in determinarea factorialului:

$$n! = n(n-1)!$$

Functie fact(intreg x) este:

    daca x<=1 fact:=1;

    altfel fact:=x*fact(x-1);

    sf.daca

Sf.functie

3.2. Probleme elementare

- 1. Calculul permutarilor, aranjamentelor si combinarilor: (*necesita calcul factorial*)

Algoritmul calcul este:

```
        citeste n,m
        p:=fact(n);
        A:=fact(n)/fact(n-m);
        C:=fact(n)/(fact(m)*fact(n-m));
        Afiseaza "Rezultatele sunt: ",p, A, C
Sf.calcul.
```

3.2. Probleme elementare

- 1. Calculul permutarilor, aranjamentelor si combinarilor: (*necesita calcul factorial*)

STIVA

x=1	nivelul 5
x=2	nivelul 4
x=3	nivelul 3
x=4	nivelul 2
x=5	nivelul 1

Pentru cazul concret n = 5:

- pe nivelul 1: x=5 se reapelează fact(4)
- pe nivelul 2: x=4 se reapelează fact(3)
- pe nivelul 3: x=3 se reapelează fact(2)
- pe nivelul 4: x=2 se reapelează fact(1),
- pe nivelul 5: x=1 şi în acest moment funcţia se poate calcula şi avem **fact(1)=1**.
- **fact(5)=5*fact(4)=5*4*fact(3)=5*4*3*fact(2)=5*4*3*2*fact(1)=5*4*3*2*1= 120**

3.2. Probleme elementare

- 2. Fibonacci recursiv: rel.: f(n)=f(n-1)+f(n-2).

Functie fib(intreg n) este:

  daca n<=2 **atunci** fib:=1;

  altfel fib:=fib(n-1)+fib(n-2);

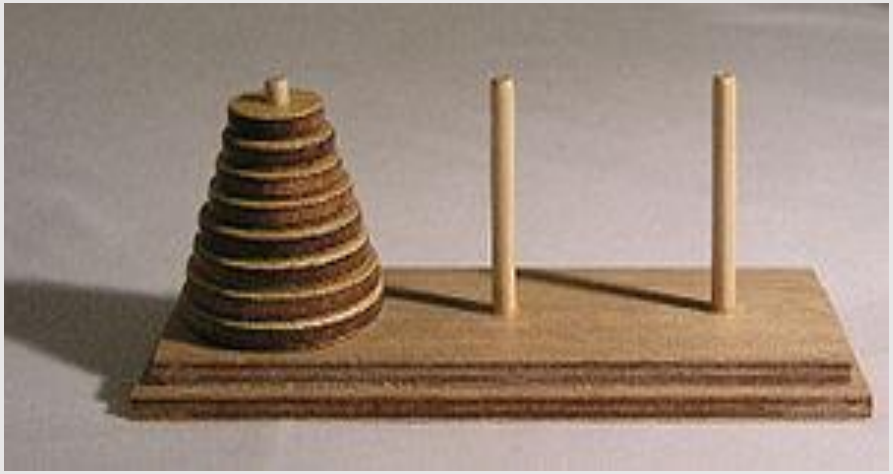
  sf.daca

Sf.fib

```
                         fib(5)
                     /              \
               fib(4)                    fib(3)
             /       \                  /      \
         fib(3)       fib(2)        fib(2)     fib(1)
        /      \      /    \        /    \
   fib(2)    fib(1) fib(1) fib(0) fib(1) fib(0)
   /     \
fib(1) fib(0)
```

3.2. Probleme elementare

- 3. Turnurile din Hanoi: propus de Eduard Lucas in 1883.
  - Format din 3 tije si un numar variabil de discuri de diferite marimi
  - Turnul format din discurile asezate pe o tija trebuie mutat pe o alta tija respectand urmatoarele reguli:
    - Doar un singur disc poate fi mutat, la un moment dat.
    - Fiecare mutare constă în luarea celui mai de sus disc de pe o tija și glisarea lui pe o altă tijă
    - Un disc mai mare nu poate fi poziționat deasupra unui disc mai mic.



https://ro.wikipedia.org/wiki/Turnul_din_Hanoi

3.2. Probleme elementare

- 3. Turnurile din Hanoi:
  - Numarul minim de miscari necesar pentru rezolvare:
    2^n-1

  - **Rezolvare**: consideram cele trei tije: cu 3 discuri…
  - S(sursa-1), I(intermediar-2) si D(destinatie-3)

1. S(1)->D(3)
2. S(1)->I(2)
3. I(3)->D(2)
4. S(1)->D(3)
5. S(2)->I(1)
6. I(2)->D(3)
7. S(1)->D(3)

3.2. Probleme elementare

- 3. Turnurile din Hanoi:

Cazul trivial este n=1, mutare 1->3

- **Hanoi(3,1,3,2)** //mutare de pe sursa 1 pe dest. 3 cu interm. 2
  1. Hanoi(2,1,2,3)
  2. Mutare 1->3
  3. Hanoi(2,2,3,1)

3.2. Probleme elementare

- 3. Turnurile din Hanoi:

```
Subalgoritmul Hanoi(n,S1,D3,I2) este:
        daca n==1 atunci
                afisare “mutate de pe” S1 “pe” D3
        altfel
                Hanoi(n-1,S1,I2,D3);
                afisare “mutate de pe” S1 “pe” D3
                Hanoi(n-1,I2,D3,S1);
        sf.daca
Sf.Hanoi
```

4. Comparatie algoritm recursiv si iterativ

- **Algoritm recursiv**: executia repetata a unui modul; daca conditia de oprire este falsa atunci modulul se reapeleaza fara ca apelurile anterioare sa fie terminate.
- **Algoritm iterativ**: executia repetata a unei portiuni de program, pana la indeplinirea unei conditii; o repetitive se lanseaza numai dupa terminarea repetitiei anterioare.

4. Comparatie algoritm recursiv si iterativ

- **Eficienta algoritmilor recursivi:**
  - In cazul implementarii intr-un limbaj de programare:
  - Induce costuri suplimentare (la fiecare apel se salveaza datele apelului in stiva).
  - Algoritmii iterativi sunt mai eficienti.
  - Orice solutie recursiva poate fi transformata intr-o solutie iterativa.

4. Comparatie algoritm recursiv si iterativ

```
Functie fib(intreg n) este:
        daca n<=2 atunci
                fib:=1;
        altfel
                fib:=fib(n-1)+fib(n-2);
        sf.daca
Sf.fibo
```

```
Functie iter_fibo(intreg n) este:
        daca n<=2 atunci iter_fibo:=1;
        sf.daca
        intreg a:=1, b:=1, suma;
        pentru i:=1;n-2 executa
                suma:=a+b;
                b:=a;
                a:=suma;
        sf.pentru
        iter_fibo:=a;
Sf.functie
```

5. Probleme

- 1. CMMDC:

```
Functie cmmdc(intreg m,n) este:
        daca m==0 atunci cmmdc:=n;
        altfel daca n==0 atunci cmmdc:=m;
        altfel daca m>n atunci
                cmmdc(n,m%n);
        altfel
                cmmdc(m,n%m);
        sf.daca
Sf.cmmdc
```