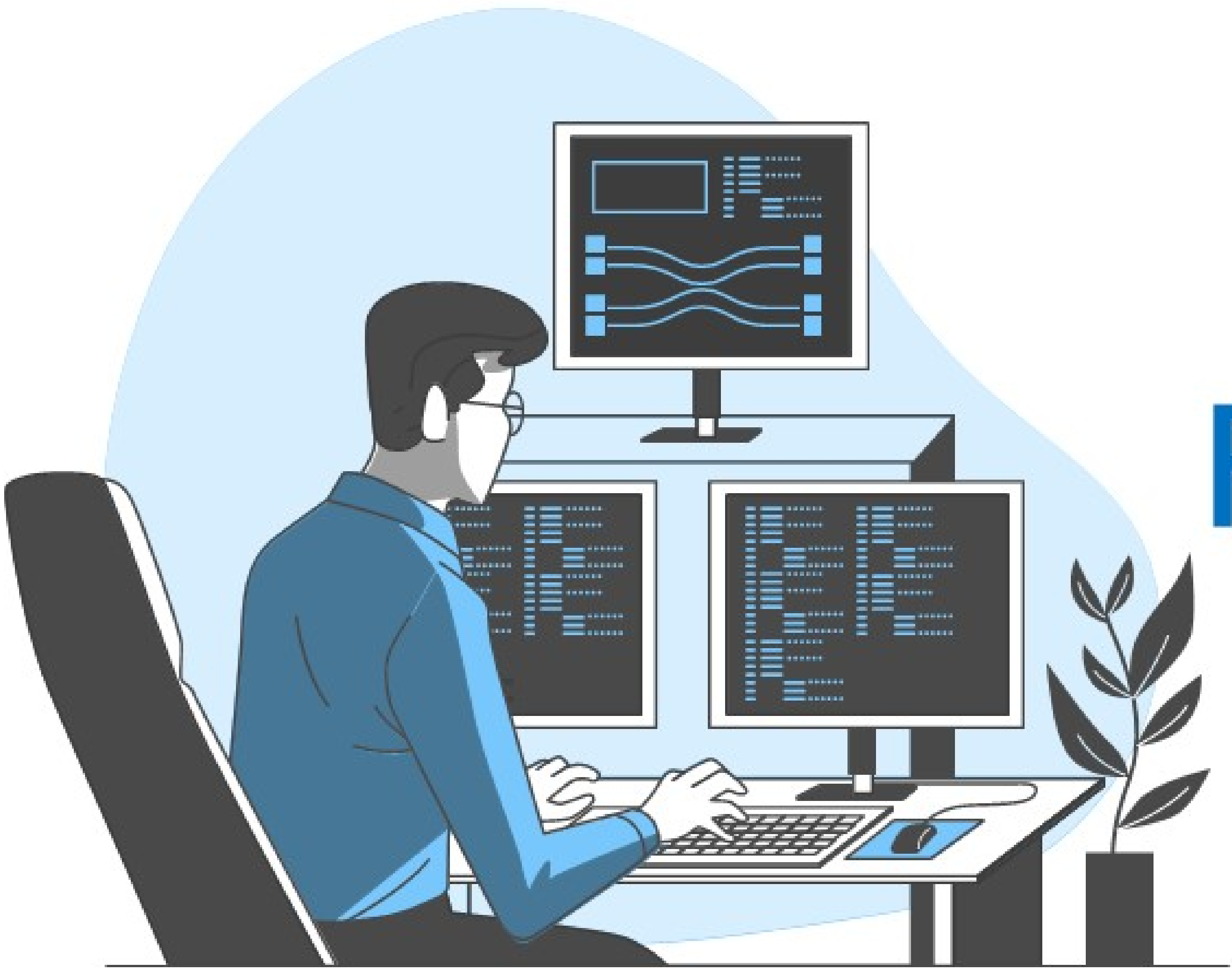


Algoritmi Fundamentali

Lector dr.
Dorin IORDACHE

Cursul nr. 7

Tehnici de programare

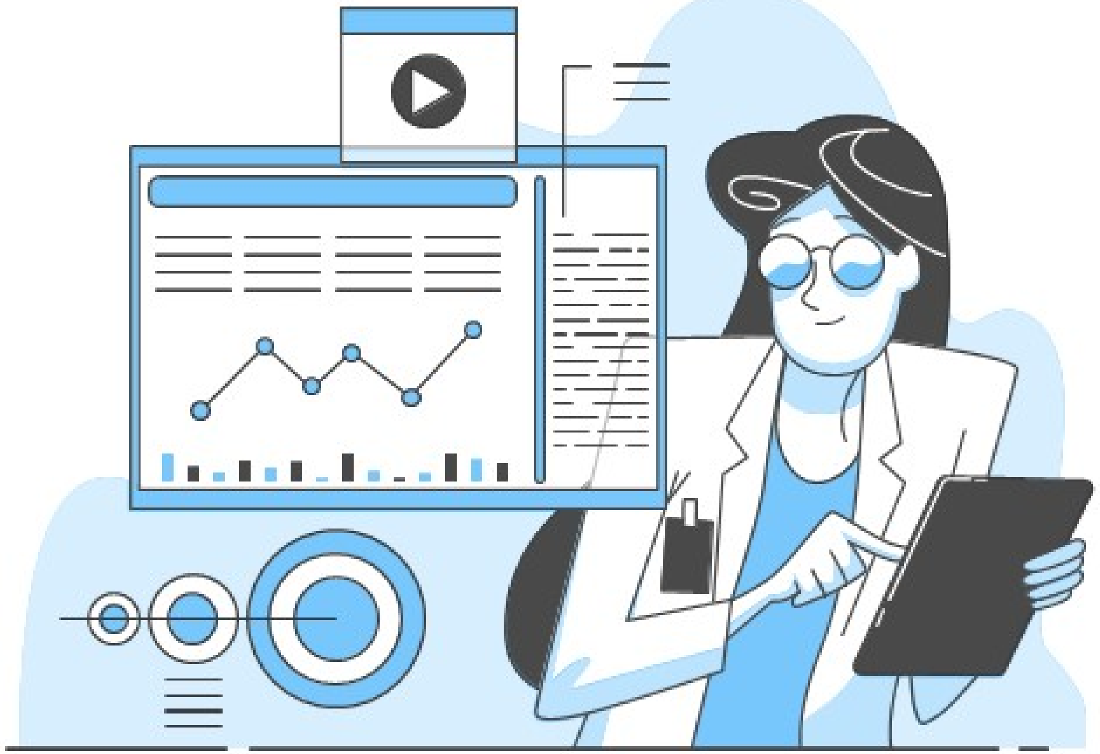
Agenda

01 Tehnica reducerii

02 Tehnica diviziunii

03 Recursivitate

04 Exemple



Tehnica de programare

… este metoda generala de rezolvare algoritmica a unei clase de probleme

… o astfel de tehnica poate fi de regula aplicata mai multor probleme provenind din diferite domenii de aplicabilitate

… furnizeaza idei de start si scheme generale de proiectare a algoritmilor destinati rezolvarii unor probleme noi

… reprezinta o colectie de instrumente utile pentru aplicatii

…

Tehnici de programare

- Tehnica fortei brute (brute force)
- Tehnica reducerii (decrease)
- Tehnica divizării (Divide et Impera, Divide and Conquer)
- Tehnica cautarii optimului local (greedy search)
- Tehnica programarii dinamice (dynamic programming)
- Tehnica cautarii cu revenire (backtracking)

…

01

Tehnica fortei brute

Forta bruta

- metode simple de rezolvare a unei probleme care se bazează pe puterea de calcul pură și pe încercarea tuturor posibilităților, mai degrabă decât a tehnicilor avansate pentru a îmbunătăți eficiența

- Exemplu:

➢ un lacăt cu 4 cifre, fiecare de la 0 la 9. Ati uitat combinația si nu vrei sa schimbi lacatul. Deoarece nu vă amintiți niciuna dintre cifre, trebuie să utilizați o metodă de forță brută pentru a deschide lacătul.
    - setați toate cifrele la 0 și încercați-le unul câte unul:
    0001, 0002, 0003 și așa mai departe.
    În cel mai rău caz, ar fi nevoie de 10^4 sau 10.000 de încercări.

Forta bruta

- Exemplu:

problema comis-voiajorului

un vânzător trebuie să viziteze 10 orașe din țară. Cum se determină ordinea în care aceste orașe ar trebui vizitate astfel încât distanța totală parcursă să fie minimă?

forța brută = calcularea distanței totale pentru fiecare rută posibilă și selectarea distantei minime.

- INEFICIENT -

Forta bruta

… este cea mai simpla (si cea mai intuitiva) cale de a rezolva problema

… algoritmii proiectati pe baza tehnicii fortei brute nu sunt intotdeauna eficienti

…

Forta bruta

Exemplu:

Calculul lui x^n, x este un numar real iar n este un numar natural

Idee: se porneste de la definitia puterii

$$x^n = x*x*\ldots*x \quad \text{(de n ori)}$$

```
Putere(x,n)
   p←1
   for i ← 1,n do
       p ← p*x
   endfor
   return p
```

Analiza eficienta

Dim. pb: n

Op. dominanta: inmultirea *

T(n)=n

Clasa de eficienta

O(n)

Forta bruta

Exemplu:

Calculul Factorial(n), n un numar natural >1

Idee: se porneste de la definitia factorialului n!=1*2*…*n

```
Factorial(n)
   f ← 1
   for i ← 1,n do
       f ← f*i
   endfor
   return f
```

Analiza eficienta

Dim. pb: n

Op. dominanta: inmultirea *

$T(n)=n$

Clasa de eficienta

$O(n)$

02

Tehnica reducerii

Diviziunea

se foloseste legatura dintre solutia unei probleme si solutia unei instante de dimensiune mai mica a aceleiasi probleme.

prin reducerea succesiva a dimensiunii problemei se ajunge la o instanta suficient de mica pentru a fi rezolvata direct

Reducerea

Exemplu.

Problema calculului puterii x^n pentru $n=2^m$, $m>=1$

Deoarece

$$x^{2^\wedge m} = \begin{cases} x*x & \text{pentru } m=1 \\ x^{2^\wedge(m-1)} * x^{2^\wedge(m-1)} & \text{pentru } m>1 \end{cases}$$

rezulta ca $x^{2^\wedge m}$ poate fi calculat dupa schema de mai jos:

$p:=x*x=x^2$
$p:=p*p=x^2*x^2=x^4$
$p:=p*p=x^4*x^4=x^8$
….

Tehnica reducerii

Exemplu.

Problema calculului puterii x^n pentru $n=2^m$, $m>=1$

```
Putere2(x,m)
  p ← x*x
  for i ← 1,m-1 do
     p ← p*p
  endfor
  return p
```

Analiza :

Correctitudine

Invariant ciclu: $p=x^{2^i}$

Eficienta

(i) dimensiune problema: m

(ii) operatie dominanta: inmultire *

$T(m) = m = \log n$

Tehnica reducerii

$$x^{2^m} = \begin{cases} x*x & \text{pentru } m=1 \\ x^{2^{(m-1)}} * x^{2^{(m-1)}} & \text{pentru } m>1 \end{cases}$$

$$x^n = \begin{cases} x*x & \text{pentru } n=2 \\ x^{n/2} * x^{n/2} & \text{pentru } n>2 \end{cases}$$

```
Putere3(x,m)
    if  m=1 then
            return x*x
    else
        p ← Putere3(x,m-1)
        return p*p
    endif
```

```
Putere4(x,n)
    if  n=2 then
            retrun x*x
    else
        p ← Putere4(x,n DIV 2)
        return p*p
    endif
```

Tehnica reducerii

```
Putere3(x,m)
   if  m=1 then
         return x*x
   else
      p ← Putere3(x,m-1)
      return p*p
   endif
```

```
Putere4(x,n)
   if  n=2 then
         retrun x*x
   else
      p ← Putere4(x,n DIV 2)
      return p*p
   endif
```

Observatii:

1. abordare descendenta (top-down): se porneste de la problema de dimensiune mare si se reduce succesiv dimensiunea pana se ajunge la o problema suficient de simpla
2. Ambii algoritmi sunt **recursivi**

Tehnica reducerii

Generalizare

```
Putere(x,n)
    if  n=1 then
            return x
    else
        if  n=2 then
            return x*x
        else
            p ← Putere(x,n div 2)
            if n mod 2=0 then
                      return p*p
             else
                      return p*p*x
             endif
         endif
  endif
```

$$x\text{^}n = \begin{cases} x*x & \text{pentru } n=2 \\ x^{n/2}*x^{n/2} & \text{pentru } n>2,\ n \text{ par} \\ x^{(n-1)/2}*x^{(n-1)/2}*x & \text{pentru } n>2,\ n \text{ impar} \end{cases}$$

03

Divide et Impera

D et I

Împarte și stăpânește (din latină divide et impera), sau Divide and conquer, în politică și sociologie câștigă și menține puterea prin spargerea unor concentrații mai mari de putere în bucăți care individual au mai puțină putere decât cel care implementează strategia.

D et I

Strategia

Divide:

- o problemă de rezolvat este împărțită într-un număr de subprobleme de aceeași formă ca și problemele originale;

Impera:

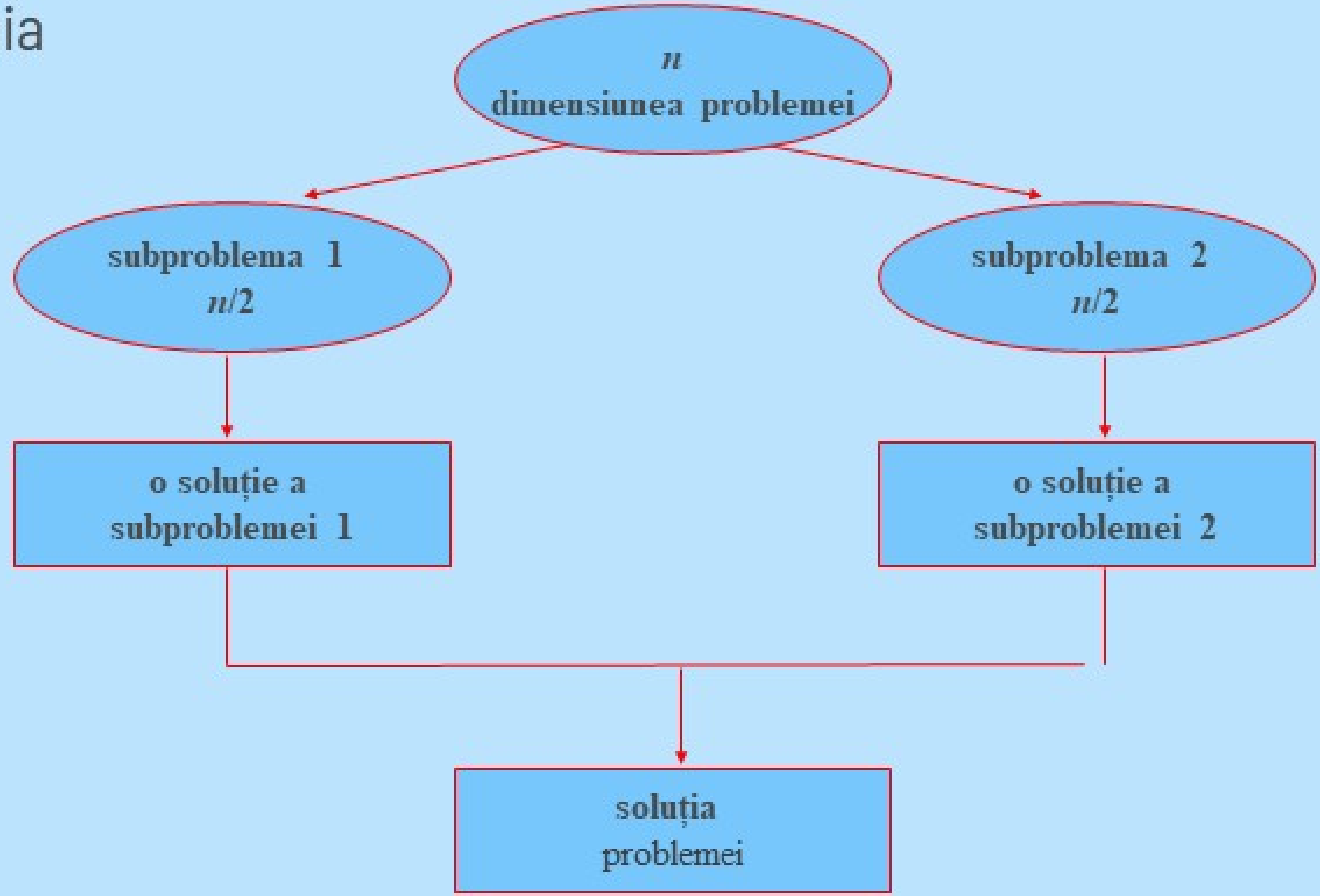
- subproblemele sunt apoi rezolvate independent, de obicei recursiv;

Combină:

- în cele din urmă, soluțiile la subprobleme sunt combinate pentru a oferi răspunsul la problema inițială.

D et I

Strategia



D et I

Scop

- Divide at Impera poate fi aplicat cu succes în calculul paralel;
- Obține algoritmi paraleli eficienți utilizând algoritmul de Divide et Impera.

D et I

Exemple

Sortare: mergesort și quicksort

Parcurgere Binary tree

Înmulțirea numerelor întregi mari

Înmulțire matrice: algoritmul Strassen's

Cea mai apropiată pereche de punte în plan

Cautare binară: (sau degenerate divide&conq.)

D et I Înmultirea numerelor întregi foarte mari

Considerăm problema de înmulțire a 2 numere întregi mari, folosind 2 vectori care conțin cifrele acestora:

A = 12345678901357986429 B = 87654321284820912836

Algoritmul:

$$
\begin{array}{r}
a_1\, a_2 \dots\ a_n \\
b_1\, b_2 \dots\ b_n \\
(d_{10})\, d_{11} d_{12} \dots d_{1n} \\
(d_{20})\, d_{21} d_{22} \dots d_{2n} \\
\dots\dots\dots\dots\dots\dots\dots \\
(d_{n0})\, d_{n1} d_{n2} \dots d_{nn}
\end{array}
$$

Complexitate: n^2

D et I Înmultirea numerelor întregi foarte mari

Exemplu:

$$A * B \text{ unde } A = 2135 \text{ și } B = 4014$$

$$A = (21 \cdot 10^2 + 35),\ B = (40 \cdot 10^2 + 14)$$

Atunci,

$$A * B = (21 \cdot 10^2 + 35) * (40 \cdot 10^2 + 14) =$$

$$= 21 * 40 \cdot 10^4 + (21 * 14 + 35 * 40) \cdot 10^2 + 35 * 14$$

În general,

dacă $A = A_1A_2$ și $B = B_1B_2$ (unde A și B au n-cifre,
și A_1, A_2, B_1, B_2 au n / 2 - cifre),

atunci

$$A * B = A_1 * B_1 \cdot 10^n + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2$$

Formula recurentă de înmulțire $M(n)$:

$$M(n) = 4M(n/2),\ \ M(1) = 1$$

Complexitate: $M(n) = n^2$

04

Recursivitate

Recursivitatea

Ce este?

Uneori, cel mai bun mod de a rezolva o problemă este rezolvarea mai întâi a unei versiuni mai mici a aceleiași probleme

Recursivitatea este o tehnică care rezolvă o problemă prin rezolvarea unei probleme reduse de același tip

Recursivitatea

Când transformați acest lucru într-un program, se utilizeaza funcții care se autoapeleaza

- funcții recursive -

```
functieF(integer x)
 integer y
  if x=0 the
    return 1
 else
    y = 2 * functieF(x-1)
   return y+1
 endif
```

Recursivitatea – aspecte

- Există multe probleme a căror soluție poate fi definită recursiv

Exemplu: *n factorial*

$$n! = \begin{cases} 1 & \text{daca } n = 0 \\ (n-1)! * n & \textit{daca } n > 0 \end{cases} \quad \textit{(soluția recursivă)}$$

$$n! = \begin{cases} 1 & \text{daca } n = 0 \\ 1*2*3*\ldots*(n-1)*n\ , & \textit{daca } n > 0 \end{cases} \quad \text{(soluția clasică)}$$

Recursivitatea – mecanism de apel

factorial(4): Stiva = [4]

4*factorial(3)

factorial(3): Stiva = [3,4]

3*factorial(2)

factorial(2): Stiva = [2,3,4]

2*factorial(1)

factorial(1): Stiva = [1,2,3,4]

```
factorial(n)
   If n<=1 then rez ← 1
           else rez ← n*fact(n-1)
   endif
return rez
```

factorial(4) 24 — Stiva = []

4*6

factorial(3) 6 — Stiva = [4]

3*2

factorial(2) 2 — Stiva = [3,4]

2*1

factorial(1) 1 — Stiva = [2,3,4]

Apel recursiv

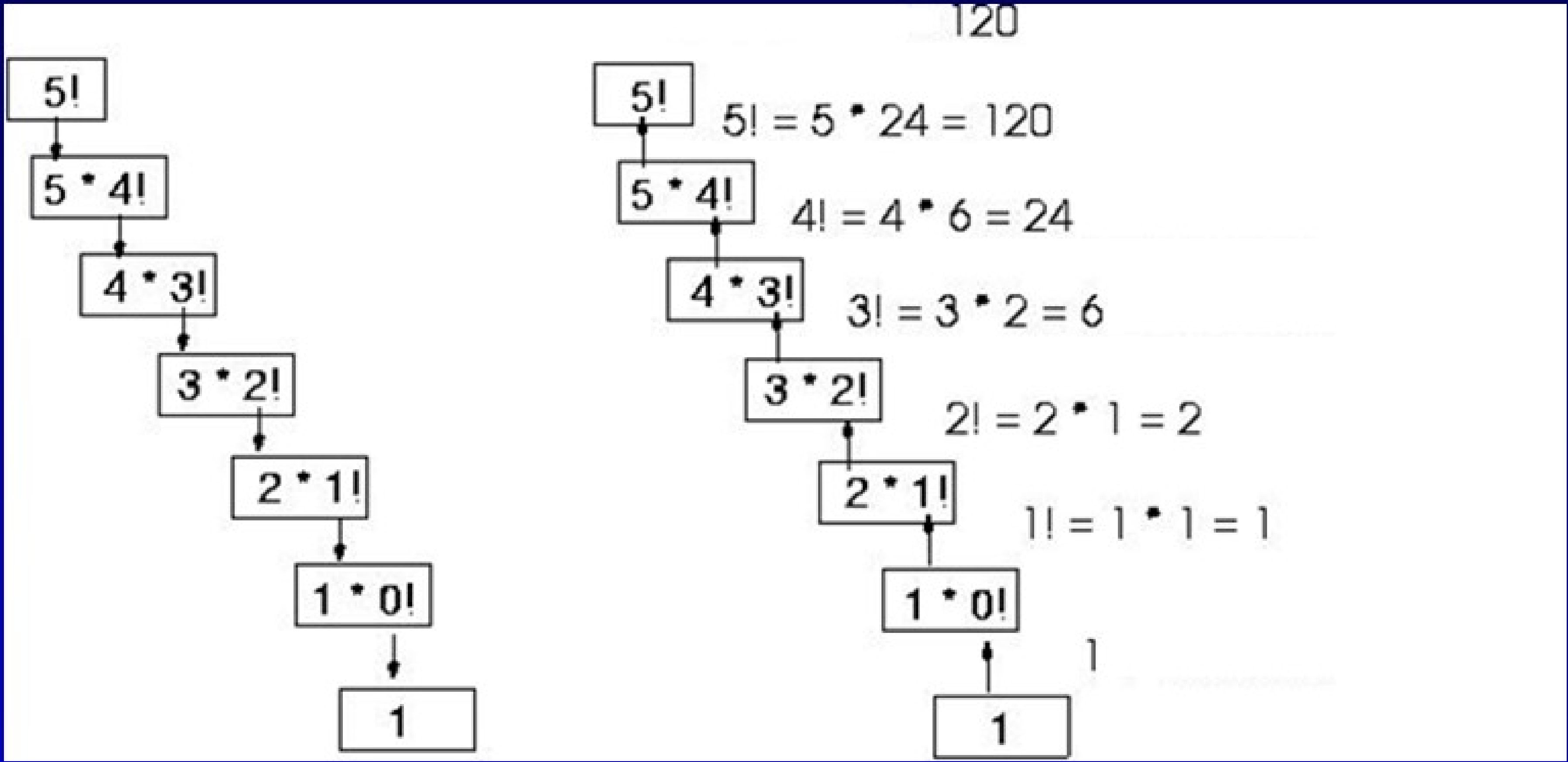
Revenire din apel

Cod program

- Implementare recursiva

```
int Factorial(int n)
{
 if (n==0)  // conditia de stop
   return 1;
 else
   return n * Factorial(n-1);
}
```

Execuție


5!
5 * 4!
4 * 3!
3 * 2!
2 * 1!
1 * 0!
1
120
5!
5! = 5 * 24 = 120
5 * 4!
4! = 4 * 6 = 24
4 * 3!
3! = 3 * 2 = 6
3 * 2!
2! = 2 * 1 = 2
2 * 1!
1! = 1 * 1 = 1
1 * 0!
1
1

Cod program

- Implementare iterativă

```
int Factorial(int n)
{
   int f = 1, i;
   if(n==0)
         return 1;
  else
        for(i = 2; i <= n; i++)
             f = f * i;
   return f;
}
```

Algoritmi recursivi - eficienta

Etapele analizei eficientei:

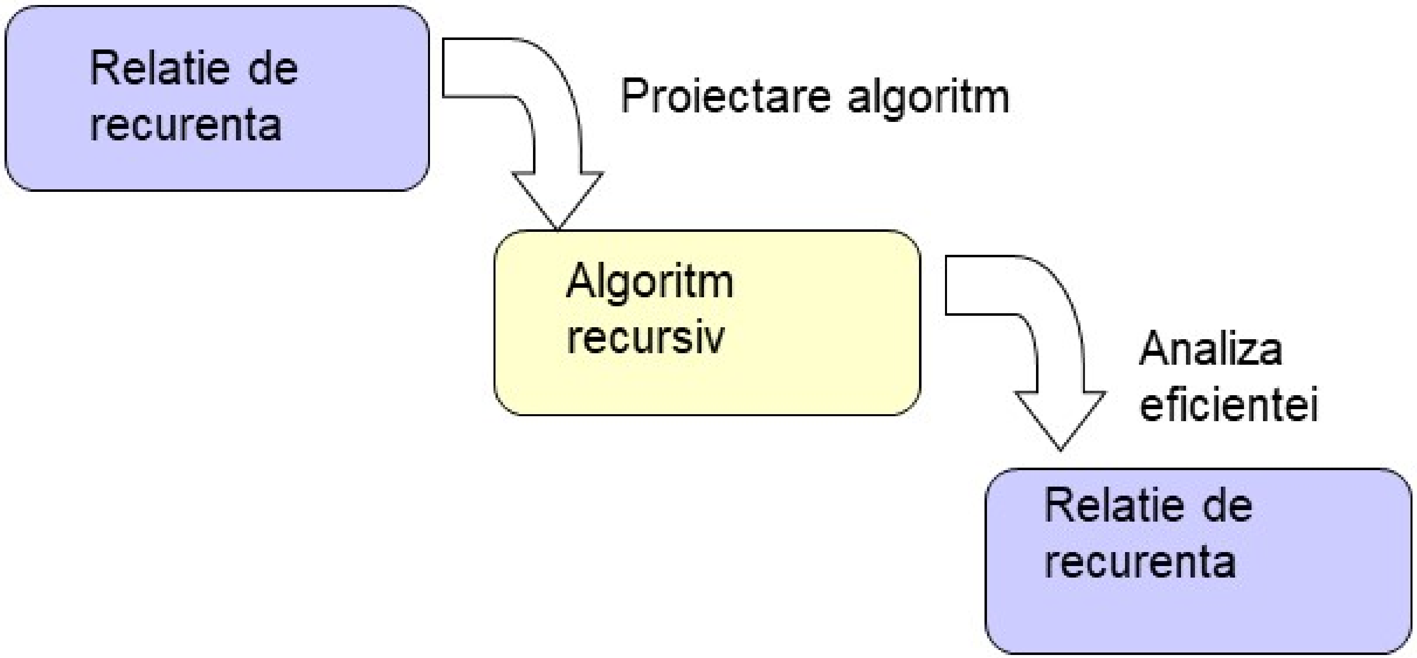
- Stabilirea dimensiunii problemei
- Alegerea operatiei dominante
- Se verifica daca timpul de executie depinde si de proprietatile datelor de intrare (in aceasta situatie se analizeaza cazul cel mai favorabil si cazul cel mai defavorabil)
- Estimarea timpului de executie

In cazul algoritmilor recursivi pentru estimarea timpului de executie se stabileste relatia de recurenta care exprima legatura dintre timpul de executie corespunzator problemei initiale si timpul de executie corespunzator problemei reduse (de dimensiune mai mica)

Estimarea timpului de executie se obtine prin rezolvarea relatiei de recurenta

Algoritmi recursivi - eficienta

Observatie:



05

Exemple

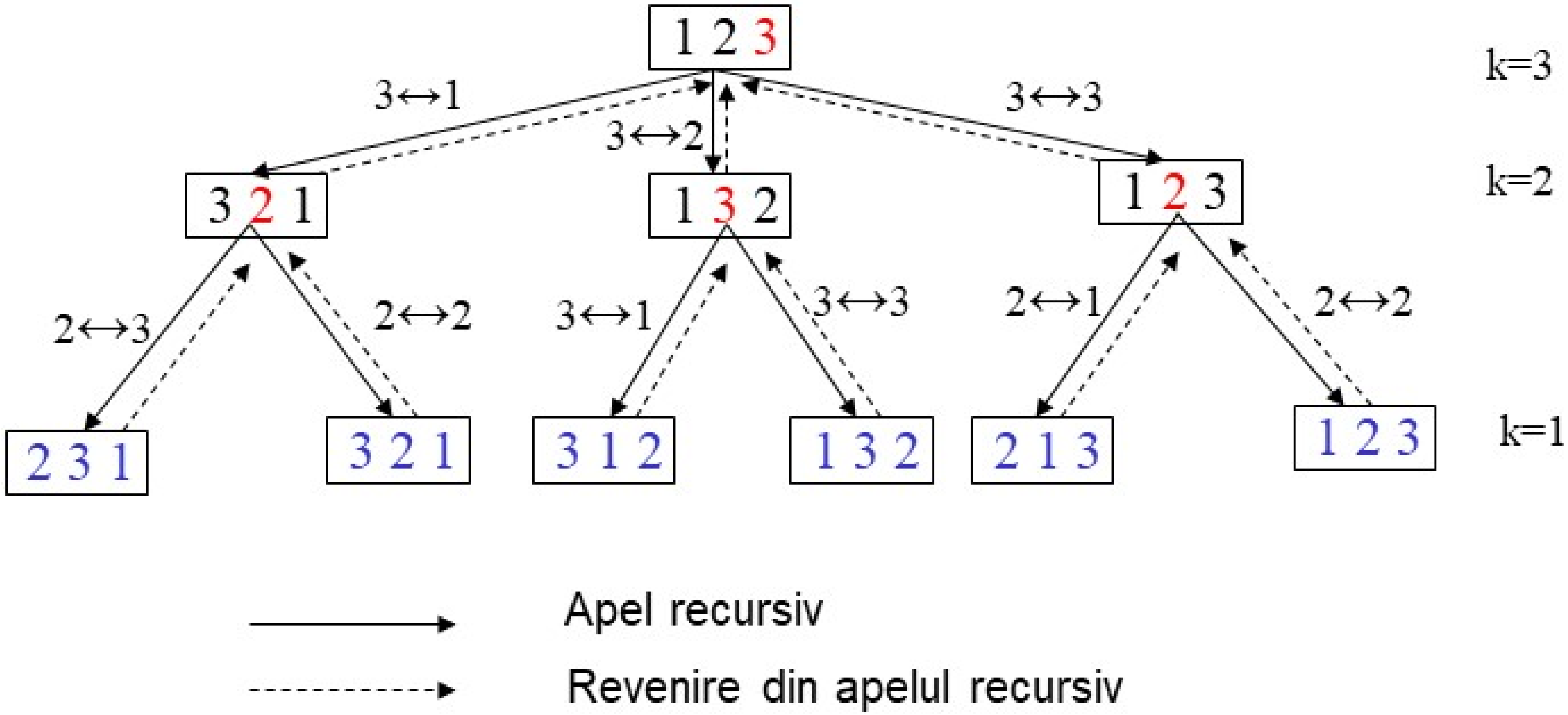
Aplicatii ale tehnicii reducerii

Exemplu 1: generarea celor n! permutari ale multimii {1,2,…,n}

Idee: cele k! permutari ale lui {1,2,…,k} pot fi obtinute din cele (k-1)! permutari ale lui {1,2,…,k-1} prin plasarea celui de al k-lea element succesiv pe prima, a doua … a k-a pozitie. Plasarea lui k pe pozitia i este realizata prin interschimbarea elementului de pe pozitia k cu cel de pe pozitia i.

Generarea permutarilor

Ilustrare pentru n=3 (abordare top-down)



Generarea permutarilor

Fie x[1..n] o variabila globala (accesibila din functie) continand initial valorile (1,2,…,n)

Algoritmul are parametrul formal k si este apelat pentru k=n.

Cazul particular este k=1, cand tabloul x contine deja o permutare completa ce poate fi prelucrata (de exemplu, afisata)

```
perm(k)
 IF k=1 THEN  WRITE x[1..n]
 ELSE
     FOR i ← 1,k DO
         x[i] ↔x[k]
         perm(k-1)
         x[i] ↔x[k]
     ENDFOR
 ENDIF
```

Apel alg: perm(n)

Analiza eficientei:

Dim pb.: k

Operatie dominanta: interschimbare

Relatie de recurenta:

$$T(k)=\begin{cases} 0 & k=1 \\ k(T(k-1)+2) & k>1 \end{cases}$$

Generarea permutarilor

$$T(k)=\begin{cases} 0 & k=1 \\ k(T(k-1)+2) & k>1 \end{cases}$$

$T(k) \quad =k(T(k-1)+2)$

$T(k-1)=(k-1)(T(k-2)+2) \quad |*k$

$T(k-2)=(k-2)(T(k-3)+2) \quad |*k*(k-1)$

…

$T(2) \quad =2(T(1)+2) \qquad |*k*(k-1)*\ldots*3$

$T(1) \quad =0 \qquad |*k*(k-1)*\ldots*3*2$

$T(k)=2(k+k(k-1)+ k(k-1)(k-2)+\ldots+k!) =2k!(1/(k-1)!+1/(k-2)!+\ldots+ \frac{1}{2}+1)$

-> 2e k! (pentru valori mari ale lui k). Pt k=n => $T(n) \in O(n!)$

Problema turnurilor din Hanoi

Istoric: problema propusa de matematicianul Eduard Lucas in 1883

Ipoteze:

- Consideram 3 vergele etichetate cu S (sursa), D (destinatie) and I (intermediar).
- Initial pe vergeaua S sunt plasate n discuri de dimensiuni diferite in ordine descrescatoare a dimensiunilor (cel mai mare disc este la baza vergelei iar cel mai mic in varf)

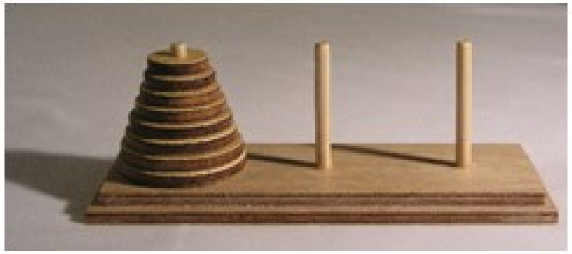
Scop:

- Sa se mute toate discurile de pe S pe D utilizand vergeaua I ca intermediara

Restrictie:

- La o etapa se poate muta un singur disc

si este interzisa plasarea unui disc mai mare peste un disc mai mic.



Problema turnurilor din Hanoi

Idee:

- Se muta (n-1) discuri de pe S pe I (utilizand D ca vergea auxiliara)
- Se muta discul ramas pe S direct pe D
- Se muta (n-1) discuri de pe I pe D (utilizand S ca vergea auxiliara)

Algoritm:

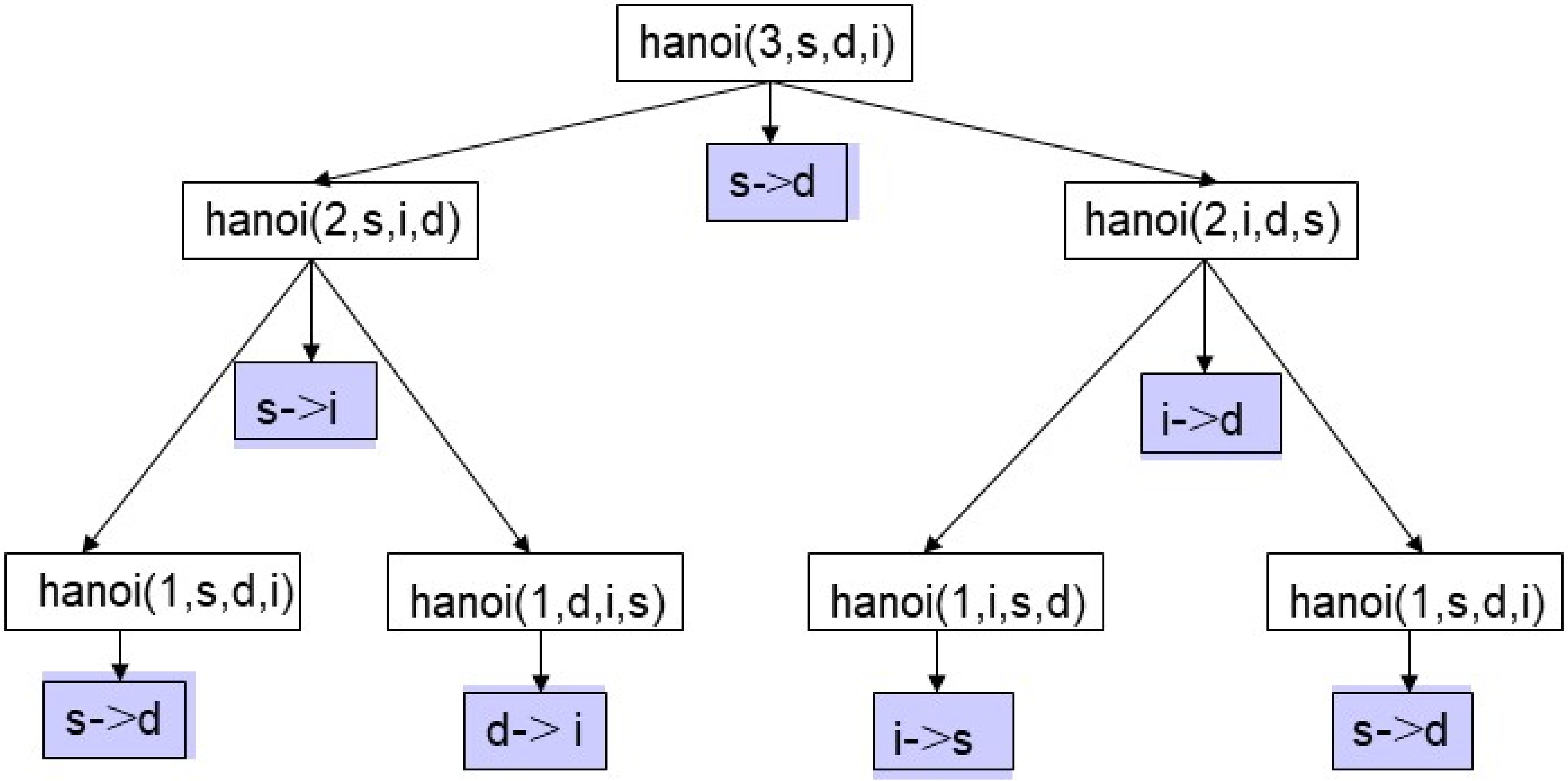
```
hanoi(n,S,D,I)
   IF n=1 THEN   "move from
   S to D"
   ELSE  hanoi(n-1,S,I,D)
         "move from S to D"
         hanoi(n-1,I,D,S)
   ENDIF
```

Semnificatia parametrilor:

- Primul parametru: numarul discurilor
- Al doilea parametru: vergea sursa
- Al treilea parametru: vergea destinatie
- Al patrulea parametru: vergea intermediara

Problema turnurilor din Hanoi

Ilustrare apeluri recursive pentru n=3.



Problema turnurilor din Hanoi

```
hanoi(n,S,D,I)
   IF n=1 THEN   "move from S to D"
   ELSE  hanoi(n-1,S,I,D)
         "move from S to D"
         hanoi(n-1,I,D,S)
   ENDIF
```

Dim pb: n

Operatie dominanta: move

Relatie de recurenta:

$$T(n)=\begin{cases}1 & n=1\\ 2T(n-1)+1 & n>1\end{cases}$$

$$\begin{aligned}
T(n) &= 2T(n-1)+1 & \\
T(n-1) &= 2T(n-2)+1 & |{*2}\\
T(n-2) &= 2T(n-3)+1 & |{*2^2}\\
&\ldots & \\
T(2) &= 2T(1)+1 & |{*2^{n-2}}\\
T(1) &= 1 & |{*2^{n-1}}
\end{aligned}$$

$T(n)=1+2+\ldots+2^{n-1} = 2^n -1$

$T(n) \in O(2^n)$

Variante ale tehnicii reducerii

- Reducere prin scaderea unei constante
  - Exemplu: n! (n!=1 if n=1
    n!=(n-1)!*n if n>1)
- Reducere prin impartirea la o constante
  - Exemplu: x^n ($x^n=x*x$ if n=2
    $x^n=x^{n/2}*x^{n/2}$ if n>2, $n=2^m$)
- Reducere prin scaderea unei valori variabile
  - Exemplu: cmmdc(a,b) (cmmdc(a,b)=a pt a=b
    cmmdc(a,b)=cmmdc(b,a-b) pt a>b
    cmmdc(a,b)=cmmdc(a,b-a) pt b>a)
- Reducere prin impartire la o valoare variabila
  - Exemplu: cmmdc(a,b)

    (cmmdc(a,b)=a pt b=0
    cmmdc(a,b)=cmmdc(b,a MOD b) pt b<>0)

Suma numerelor de la 1 la n

```
int suma(int n)
{
   if (n==0)
      return 0;
   else
     return (n + suma(n-1));
}
```

Suma valorilor unui vector a[]

```
int Suma (int n, int a[10])
{
    if(n==0)
      return 0;
    else
     return(a[n]+Suma(n-1,a));
 }
```

```
int Suma (int n, int a[10])
{
     return (n==0)? 0 : Suma(a[n]+ Suma(n-1,a));
 }
```

Suma cifrelor unui numar

```
int Suma (int n)
{
    if ( n==0 )
        return 0;
    else
        return n%10 + Suma(n/10);
}
```

```
int Suma (int n)
{
    return ( n==0 ) ? 0 : n%10 + Suma(n/10);
}
```

CMMDC - Euclid

```c
unsigned int cmmdc(unsigned int a, unsigned int b)
{
	while(a!=b)
	{
		if(a>b)
			a=a-b;
		else
			b=b-a;
	}
	return a;
}
```

CMMDC - recursiv

```c
unsigned int cmmdc(unsigned int a, unsigned int b)
{
    if(a==b)
        return a;
    else
        if(a>b)
            return cmmdc(a-b,b);
        else
            return cmmdc(a,b-a);
}
```

Mulțumesc

pentru atenție!

dorin.iordache@365.univ-ovidius.ro