Facultatea de Matematica si Informatica, Universitatea din Bucuresti

Recursivitate

1. *Tipuri de subprograme recursive*
2. Verificarea corectitudinii subprogramelor recursive
3. Complexitatea timp
4. Criterii de alegere; avantaje si dezavantaje
5. Greseli tipice in elaborarea subprogramelor recursive
6. Culegere de probleme

1. Tipuri de subprograme recursive

2. Verificarea corectitudinii subprogramelor recursive

3. Complexitatea timp

4. Criterii de alegere; avantaje si dezavantaje

5. Greseli tipice in elaborarea subprogramelor recursive

6. Culegere de probleme

---

(1) bazate pe funcţii (primitiv) recursive unare, (funcţia se apelează pe ea însăşi în mod direct, ca în cazul calculării factorialului);

(2) bazate pe funcţii (primitiv) recursive de mai multe argumente (ca în cazul calculării cmmdc pentru două numere naturale);

acestea sunt cunoscute sub numele de recursivitate liniară directă:

```
int cmmdc1 (int x, int y)
{ if (x==y) return x;
    else
     if (x>y) return cmmdc1(x-y, y);
       else  return cmmdc1(x, y-x);
}
int cmmdc2 (int x, int y)
{ if  (0==y) return x;
     else
       return cmmdc2(y, x%y);
}
```

---

(3) bazate pe funcţii care se apelează una pe alta (aşa numita recursivitate indirectă),

```
int a (int n)
{
    if  (0==n) return 1;
    return a(n-1)+b(n-1);
}
int b (int n)
{
    if  (0==n) return 1;
    return a(n-1)*b(n-1);
}
```

1. ***Tipuri de subprograme recursive***
2. **Verificarea corectitudinii subprogramelor recursive**
3. **Criterii de alegere**
4. **Avantaje si dezavantaje**
5. **Greseli tipice in elaborarea subprogramelor recursive**
6. **Culegere de probleme**

(4) bazate pe funcţii care au nevoie de mai multe valori anterioare pentru a calcula valoarea curentă (aşa numita recursivitatea neliniară sau în cascadă, ca în cazul determinării unui element al şirului lui Fibonacci după formula:

```
int fibonacci (int n)
{
  if (n<=1) return 1;
  return fibonacci(n-1) + fibonacci(n-2);
}
```

- se face intr-un mod similar verificării corectitudinii subprogramelor nerecursive.
- este mult simplificată de forma subprogramului (care permite utilizarea comodă a metodei inducţiei matematice complete):
  - se verifică mai întâi dacă toate cazurile particulare (de terminare a apelului recursiv) funcţionează corect;
  - se trece apoi la o verificare formală, prin inducţie, a funcţiei recursive corespunzătoare, pentru restul cazurilor.

1. Tipuri de subprograme recursive

2. Verificarea corectitudinii subprogramelor recursive

3. Complexitatea timp

4. Criterii de alegere; avantaje si dezavantaje

5. Greseli tipice in elaborarea subprogramelor recursive

6. Culegere de probleme

Exemplificare: calculul factorialului unui număr

```
int factorial (int n)
{
 if (0==n) return 1;
 return n*factorial(n-1);
}
```

Demonstrarea corectitudinii: doi pasi:

- pentru n = 0, valoarea 1 returnată de program este corectă;
- dacă n>1 atunci, presupunând corectă valoarea returnată pentru (n-1), prin înmulţirea acesteia cu n se obţine valoarea corectă a factorialului numărului natural n, valoare returnată de subprogram.

  În ambele situaţii este satisfăcută condiţia de oprire.

1. Tipuri de subprograme recursive

2. Verificarea corectitudinii subprogramelor recursive

3. Complexitatea timp

4. Criterii de alegere; avantaje si dezavantaje

5. Greseli tipice in elaborarea subprogramelor recursive

6. Culegere de probleme

Exemplul 1

Fie f : N $\rightarrow$ N, $f(n) = 5n^3 + 2n^2 + 22n + 6$;

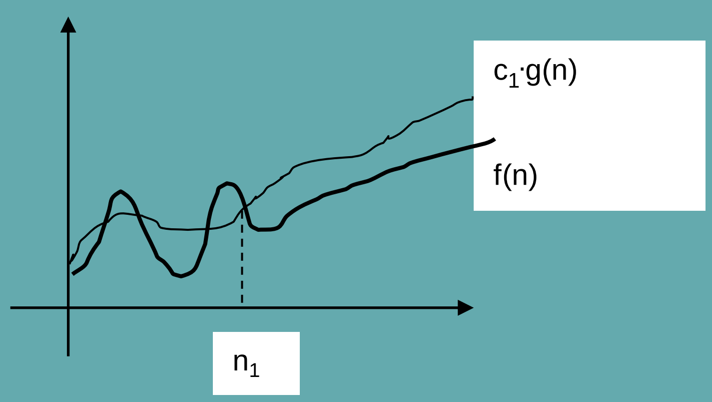
spunem că f tinde asimptotic către n^3 şi notăm acest lucru cu $\boldsymbol{O}(n^3)$

Definiţie 3

Fie f, g : N $\rightarrow$ R_+

(i) f(n) = O(g(n)) şi citim "f(n) este de ordin cel mult g(n)" sau "f(n) este O mare de g(n)" $\Leftrightarrow$

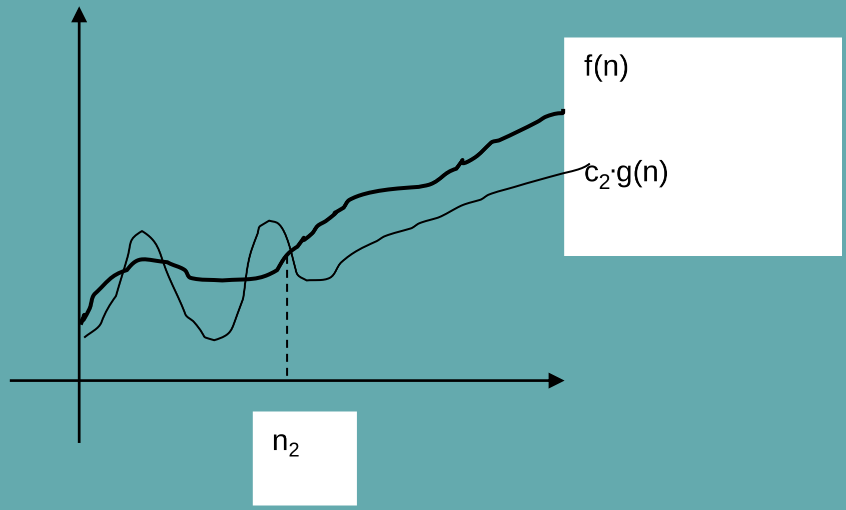
($\exists$) constantele $c_1 > 0$ şi $n_1 \in N$ astfel încât $f(n) \leq c_1.g(n)$, ($\forall$) $n \geq n_1$.



1. Tipuri de subprograme recursive
2. Verificarea corectitudinii subprogramelor recursive
3. Complexitatea timp
4. Criterii de alegere; avantaje si dezavantaje
5. Greseli tipice in elaborarea subprogramelor recursive
6. Culegere de probleme

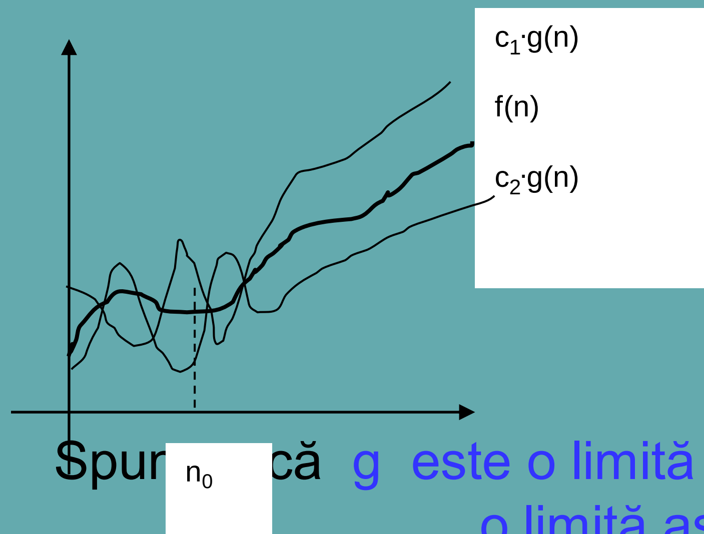
(ii) $f(n) = \Omega(g(n))$ şi citim "f(n) este de ordin cel puţin g(n)" sau "f(n) este omega mare de g(n)" $\Leftrightarrow$

$(\exists)$ constantele $c_2 > 0$ şi $n_2 \in N$ astfel încât $f(n) \geq c_2 . g(n)$, $(\forall)\ n \geq n_2$.



(iii) $f(n) = \Theta(g(n))$ şi citim "f(n) este de ordin g(n)" sau "f(n) este theta de g(n)" $\Leftrightarrow$

$f(n) = O(g(n))$ şi $f(n) = \Omega(g(n))$.



Spun că g este o limită asimptotică superioară,
o limită asimptotică inferioară, respectiv
o limită asimptotică pentru f.

Exemplul 2

Revenim la notaţia ***O*** şi la funcţia polinomială

$$f(n) = 5n^3 + 2n^2 + 22n + 6 \Rightarrow$$

$f(n) = \boldsymbol{O}(n^3)$, de exemplu, pentru $c_1 = 6$ şi $n_1 = 10$;

$f(n) = \boldsymbol{O}(n^4)$, de exemplu, pentru $c_1 = 1$ şi $n_1 = 6$ sau

pentru $c_1 = 36$ şi $n_1 = 1$;

$f(n) \neq \boldsymbol{O}(n^2)$,

presupunem prin absurd că există $c_1 > 0$ şi $n_1 \in N$ astfel încât

$5n^3 + 2n^2 + 22n + 6 \leq c_1 . n^2, (\forall)\ n \geq n_1 \Leftrightarrow$

$5n^3 + (2- c_1) . n^2 + 22n + 6 \leq 0$ etc.

1. **Tipuri de subprograme recursive**
2. **Verificarea corectitudinii subprogramelor recursive**
3. **Complexitatea timp**
4. **Criterii de alegere; avantaje si dezavantaje**
5. **Greseli tipice in elaborarea subprogramelor recursive**
6. **Culegere de probleme**

Exemplul 3

Fie $f_1 : \mathbf{N} \rightarrow \mathbf{N}$, $f_1(n) = 3n.\log_2 n + 5n.\log_2(\log_2 n) + 2 \Rightarrow$

$f_1(n) = \boldsymbol{O}(n.\log n)$ pentru că log n domină log(log n).

Analog, $f_2(n) = \boldsymbol{O}(n^2) + \boldsymbol{O}(n) \rightarrow f_2(n) = \boldsymbol{O}(n^2)$

pentru că $\boldsymbol{O}(n^2)$ domină $\boldsymbol{O}(n)$.

Observaţia 1

A) Specificarea bazei logaritmilor nu este necesară ea intervenind cu cel mult un coeficient constant, conform formulei: $\log_a x = \frac{\log_b x}{\log_b a}$

B) Analog, nici specificarea bazei exponenţialei nu este necesară pentru că:

$$\forall x > 0: \; x = 2^{\log_2 x} \;\rightarrow\; n^c = 2^{c \cdot \log_2 n} \;\Rightarrow$$

$2^{\boldsymbol{O}(\log n)}$ este o limită superioară pentru n^c, unde c este o constantă oarecare. Evident, şi $2^{\boldsymbol{O}(n)}$ este o limită superioară pentru n^c.

1. Tipuri de subprograme recursive
2. Verificarea corectitudinii subprogramelor recursive
3. Complexitatea timp
4. Criterii de alegere; avantaje si dezavantaje
5. Greseli tipice in elaborarea subprogramelor recursive
6. Culegere de probleme

Observaţia 2

Limitele asimptotice de tipul n^c se numesc limite polinomiale.

Limitele asimptotice de tipul $2^{n^{\delta}}$ se numesc limite exponenţiale.

Limitele asimptotice de tipul k.n se numesc limite lineare.

Limitele asimptotice de tipul $\sqrt{n}$ se numesc limite sublineare.

Observaţia 3

Pe lângă notaţiile **O** şi **Ω** mai există şi notaţiile **o** şi **ω** , obţinute din Definiţia 2 prin înlocuirea inegalităţii ≤ cu inegalitatea strictă < , sau

$$f(n) = o(g(n)) \Leftrightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Exemplul 4

$\sqrt{n} = o(n)$

$n = \boldsymbol{o}(n.\log\log n)$

$n.\log\log n = \boldsymbol{o}(n.\log n)$

$n.\log n = \boldsymbol{o}(n^2)$

$n^2 = \boldsymbol{o}(n^3)$

Propozitia 1

(i) Notaţiile O, Ω, Θ, o, ω sunt tranzitive:

$$f(n) = O(g(n)) \ \& \ g(n) = O(h(n)) \rightarrow f(n) = O(h(n))$$

etc.;

(ii) Notaţiile O, Ω, Θ, sunt reflexive, dar nu şi o, ω

$$f(n) = O(f(n)) \text{ dar } f(n) \neq o(f(n))$$ etc.;

(iii) Notaţia Θ este simetrică dar nu şi celelalte notaţii:

$$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n)).$$

Propozitia 2

Notaţiile *O*, Ω etc. pot fi manipulate algebric, dar cu precautie (de ex. egalitatea din formula (5) are loc intr-un singur sens: de la stanga la dreapta):

1. $c.O(f(n)) = O(f(n));$
2. $O(f(n)) + O(f(m)) = O(f(n));$
3. $O(O(f(n))) = O(f(n));$
4. $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n));$
5. $O(f(n) \cdot g(n)) = f(n) \cdot O(g(n)).$

1. Tipuri de subprograme recursive
2. Verificarea corectitudinii subprogramelor recursive
3. Complexitatea timp
4. Criterii de alegere; avantaje si dezavantaje
5. Greseli tipice in elaborarea subprogramelor recursive
6. Culegere de probleme

Pentru a calcula complexitatea timp a unui algoritm (nr de pasi executati) vom proceda astfel:

- pentru fiecare etapa calculam:
  - nr de pasi necesari executarii ei,
  - de cate ori se executa etapa respectiva,
  - inmultim cele 2 valori;
- adunam valorile obtinute pentru etapele algoritmului si aplicam regulile calculului asimptotic.

	Nr de pasi	*Nr de executii*	*Total*
Etapa nr. 1	n^2	k	$k.n^2$
Etapa nr. 2	n^k	n	n^{k+1}
……………			

1. Tipuri de subprograme recursive
2. Verificarea corectitudinii subprogramelor recursive
3. Complexitatea timp
4. Criterii de alegere; avantaje si dezavantaje
5. Greseli tipice in elaborarea subprogramelor recursive
6. Culegere de probleme

Alegerea variantei recursive / iterative pentru scrierea unui program presupune:

- cunoasterea fiecarei metode in parte si a particularitatilor sale;
- cunoasterea tehnicii de transformare a recursivitatii in iteratie;
- studiu profund al structurilor de date optime reprezentarii datelor problemei;
- stapanirea tuturor facilitatilor oferite de limbajul de programare in care va fi implementat algoritmul.

1. Tipuri de subprograme recursive
2. Verificarea corectitudinii subprogramelor recursive
3. Complexitatea timp
4. Criterii de alegere; avantaje si dezavantaje
5. Greseli tipice in elaborarea subprogramelor recursive
6. Culegere de probleme

In alegerea intre metoda recursiva si iterativa in elaborarea unui program trebuie sa se tina seama de :

- eficienta oferita programului de catre fiecare dintre variante,
- relatia dintre timpului de rulare si spatiului de memorie necesar
- nevoia de compactizare a programului.

1. **Tipuri de subprograme recursive**
2. **Verificarea corectitudinii subprogramelor recursive**
3. **Complexitatea timp**
4. **Criterii de alegere; avantaje si dezavantaje**
5. **Greseli tipice in elaborarea subprogramelor recursive**
6. **Culegere de probleme**

- Exista o legatura stransa intre recursivitate si structurile de date de tip stiva, arbore, etc folosite in limbajele Borland Pascal si C++ pentru reprezentarea functiilor / procedurilor recursive (insasi definitia stivelor, arborilor, listelor realizandu-se recursiv).
- Iterativitatea minimizeaza complexitatea unor algoritmi
- Deseori, variantele iterative necesita folosirea explicita a structurii de tip stiva, generand astfel un cod extrem de laborios. In aceste situatii se considera solutia recursiva mai eleganta, datorita simplitatii sale.
- Recursivitatea poate fi inlocuita prin iteratie atunci cand recursivitatea este prea adanca sau cand limbajul de programare nu permite implementarea de apeluri recursive.
- Din punctul de vedere al memoriei solicitate, o varianta recursiva necesita un spatiu de stiva suplimentar pentru fiecare apel fata de varianta iterativa.
- Dimensiunea stivei trebuie aleasa astfel incat sa poata permite memorarea elementelor pentru toate iteratiile.

1. **Tipuri de subprograme recursive**
2. **Verificarea corectitudinii subprogramelor recursive**
3. **Complexitatea timp**
4. **Criterii de alegere; avantaje si dezavantaje**
5. **Greseli tipice in elaborarea subprogramelor recursive**
6. **Culegere de probleme**

- Exemple de probleme pentru care solutia recursiva este mai putin intuitiva decat cea iterativa:
- Sa se calculeze suma cifrelor unui numar natural *n*.

```
// suma cifrelor unui numar - varianta iterativa
#include<iostream.h>
 #include<conio.h>
 int suma(int n)
  { int s=0;
    while(n!=0)
     { s=s+n%10;
       n=n/10; }
     return s;  }
  void main()
   { int n;
     cout<<" n = "; cin>>n;
     int n1=n;
     cout<<" Suma cifrelor numarului "<<n1<<" = "<< suma(n); }
```

```
// suma cifrelor unui numar - varianta recursiva
 #include<iostream.h>
 #include<conio.h>
 int suma(int n)
  {  if(n==0) return 0;
     else return n%10 + suma(n/10);  }
  void main()
   { int n;
     clrscr();
     cout<<" n = "; cin>>n;
     int n1=n;
     cout<<" Suma cifrelor numarului "<<n1<<" = "<< suma(n); }
```

1. **Tipuri de subprograme recursive**
2. **Verificarea corectitudinii subprogramelor recursive**
3. **Complexitatea timp**
4. **Criterii de alegere; avantaje si dezavantaje**
5. **Greseli tipice in elaborarea subprogramelor recursive**
6. **Culegere de probleme**

- Sa se verifice egalitatea a doua siruri de caractere introduse de la tastatura .

```
// egalitatea a 2 stringuri - varianta iterativa
var a,b:string;
    egal:boolean;
    i:byte;
begin
writeln('introduceti sirurile :');
readln(a); readln(b);
egal:=true;
if length(a)<>length(b) then egal:=false
   else
        for i:=1 to length(a) do
              if a[i]<>b[i] then egal:=false;
if egal then writeln('sunt egale')
           else writeln('nu sunt egale')
end.
```

```
// egalitatea a 2 stringuri - varianta recursiva
var a,b:string;
function egal(a,b:string):boolean;
begin
if length(a)<>length(b) then egal:=false
   else
       if a[1]<>b[1]   then egal:=false
          else
              if length(a)=1  then egal:=true
                 else
egal:=egal(copy(a,2,length(a)-1),
                 copy(b,2,length(b)-1))
end;
begin
writeln('introduceti sirurile :');
readln(a);  readln(b);
if egal(a,b) then writeln('sunt egale')
                 else writeln('nu sunt egale')
end.
```

1. Tipuri de subprograme recursive
2. Verificarea corectitudinii subprogramelor recursive
3. Complexitatea timp
4. Criterii de alegere; avantaje si dezavantaje
5. Greseli tipice in elaborarea subprogramelor recursive
6. Culegere de probleme

- ***Motivul:*** Dificultatea descoperirii formulei de recurenta nu de iteratie
- Să se scrie un program care calculează suma

$$S_n = 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots + \frac{1}{2^n}$$

- Indicatie: In algoritmul recursiv se vor folosi 2 funcţii definite astfel:

$$2^n = \begin{cases} 1 & , n = 0 \\ 2 \cdot 2^{n-1} & , n \geq 1 \end{cases} \qquad S_n = \begin{cases} 1 & , n = 0 \\ \dfrac{1}{2^n} + S_{n-1} & , n \geq 1 \end{cases}$$

1. Tipuri de subprograme recursive
2. Verificarea corectitudinii subprogramelor recursive
3. Complexitatea timp
4. Criterii de alegere; avantaje si dezavantaje
5. Greseli tipice in elaborarea subprogramelor recursive
6. Culegere de probleme

// calculul sumei - varianta iterativa

```
#include<iostream.h>
#include<conio.h>
unsigned n;
void citire()
{cout<<"n=";cin>>n;}
float suma(unsigned n)
{unsigned i;
 long p=1;
 float s=1;
 for(i=1;i<=n;i++)
          {p*=2;
           s+=(float)1/p;}
 return s;}
void main()
{clrscr();
 citire();
 cout<<"suma pentru n="<<n<<" este="<<suma(n);
 getch();}
```

// calculul sumei - varianta recursiva

```
#include<iostream.h>
#include<conio.h>
unsigned n;
void citire()
{cout<<"n=";cin>>n;}
long putere2(unsigned n)
{if(n==0) return 1;
         else return 2*putere2(n-1);}
float suma(unsigned n)
{if(n==0) return 1;
         else
  return (float)1/putere2(n)+suma(n-1);}
void main()
{clrscr();
 citire();
 cout<<"suma pentru n="<<n<<" este="<<suma(n);
 getch();}
```

1. Tipuri de subprograme recursive
2. Verificarea corectitudinii subprogramelor recursive
3. Complexitatea timp
4. Criterii de alegere; avantaje si dezavantaje
5. Greseli tipice in elaborarea subprogramelor recursive
6. Culegere de probleme

Greseli tipice in realizarea subprogramelor recursive:

1. Declararea globala a unor variabile care controleaza adresa de revenire (cazul cand apelurile se fac din interiorul unei structuri repetitive).
2. Declararea ca parametri transmisi prin valoare sau ca variabile locale a unor date structurate (de exemplu de tip tablou) micsoreaza semnificativ adancimea acceptata a recursivitatii, deoarece memorarea lor necesita un spatiu mare de memorie.

---

4. Absenta unei conditii de oprire.
5. Exprimarea unei conditii de oprire in care nu intervin nici variabile locale si nici parametrii transmisi prin valoare sau prin referinta ai subprogramului.
6. Marirea dimensiunii problemei prin transmiterea in cadrul autoapelurilor a unor parametri actuali care nu tind sa se apropie de valorile impuse prin conditia de oprire.
7. Neutilizarea directivei forward (limbajul Borland Pascal) in cazul declararii unor subprograme indirect recursive

1. **Tipuri de subprograme recursive**
2. **Verificarea corectitudinii subprogramelor recursive**
3. **Complexitatea timp**
4. **Criterii de alegere; avantaje si dezavantaje**
5. **Greseli tipice in elaborarea subprogramelor recursive**
6. ***Culegere de probleme re rezolvate si propuse***

Probleme propuse:

1. Calculul recursiv al mediei aritmetice într-un vector.
2. Calculul recursiv al maximului şi al minimului dintr-un vector.
3. Calculul recurent / inductiv al funcţiei:

$$f_n(x)=\begin{cases}\dfrac{1}{1+x}, n=0\\ f_{n-1}(x)+\dfrac{2^n}{1+x^{2^n}}, n>0\end{cases}$$

4. Calculul recurent / inductiv al funcţiei:

$$f_{n,k}=\begin{cases}\dfrac{1}{k(k+1)}, n=1\\ f_{n-1,k}+\dfrac{1}{(k+n-1)(k+n)}, n>1\end{cases}$$ unde $k\geq 1$ este un număr natural.

1. **Tipuri de subprograme recursive**
2. **Verificarea corectitudinii subprogramelor recursive**
3. **Complexitatea timp**
4. **Criterii de alegere; avantaje si dezavantaje**
5. **Greseli tipice in elaborarea subprogramelor recursive**
6. ***Culegere de probleme re rezolvate si propuse***

---

5. Valoarea funcţiei Ackerman în variantă recursivă şi nerecursivă pentru perechea (*m*, *n*), unde funcţia este definită astfel:

$$Ack(m,n)=\begin{cases} n+1, m=0 \\ Ack(m-1,1), n=0 \\ Ack(m-1, Ack(m,n-1)), \text{altfel} \end{cases}$$

6. Să se realizeze parcurgerea arborilor binari (preordine, postordine, inordine), recursiv şi iterativ.
7. Se consideră şirul a_1, a_2, …, a_n în progresie aritmetică (i.e. $\forall n \geq 2$: $a_n=(a_{n-1}+a_{n+1})/2$) . Să se calculeze recursiv suma dată prin formula de recurenţă:

$$S_n:\begin{cases} S_1=\dfrac{1}{\sqrt{a_1}+\sqrt{a_2}}, n=1 \\ S_n=S_{n-1}+\dfrac{1}{\sqrt{a_n}+\sqrt{a_{n+1}}}, n\geq 2 \end{cases}$$

---

8. Se consideră şirul $a_1, a_2, \ldots, a_n$ in progresie geometrica (i.e. $\forall n \geq 2$: $a_n = \sqrt{a_{n-1} \cdot a_{n+1}}$). Să se calculeze recursiv suma dată prin formula de recurenţă:

$$S_n : \begin{cases} S_1 = \dfrac{\sqrt{a_1}}{\sqrt{a_2} - \sqrt{a_1}}, n = 1 \\ S_n = S_{n-1} + \dfrac{\sqrt{a_n}}{\sqrt{a_{n+1}} - \sqrt{a_n}}, n \geq 2 \end{cases}$$

9. Să se calculeze recursiv suma: $\dfrac{1}{a_1 a_2 \ldots a_k} + \dfrac{1}{a_2 a_3 \ldots a_{k+1}} + \ldots + \dfrac{1}{a_n a_{n+1} \ldots a_{n+k-1}}$

unde şirul $(a_n)_{n\geq 1}$ este reprezentat printr-o progresie aritmetică.

10. Să se calculeze recursiv suma

$a_1 a_2 \ldots a_k + a_2 a_3 \ldots a_{k+1} + \ldots + a_{n+1} a_{n+2} + \ldots + a_{n+k}$ unde şirul $(a_n)_{n\geq 1}$ este reprezentat printr-o progresie aritmetică.

1.Tipuri de subprograme recursive
2. Verificarea corectitudinii subprogramelor recursive
3. Complexitatea timp
4. Criterii de alegere; avantaje si dezavantaje
5. Greseli tipice in elaborarea subprogramelor recursive
6. Culegere de probleme re rezolvate si propuse

---

11. Se consideră matricea: $A=\begin{pmatrix}1 & 1 & a\\ 0 & 1 & 1\\ 0 & 0 & 1\end{pmatrix}$. Calculaţi recursiv $A^{n,}$ n≥2.

12. Idem pentru matricele: .

$A=\begin{pmatrix}0 & e^{x} & e^{-x}\\ e^{-x} & 0 & 0\\ 0 & 0 & 0\end{pmatrix}$, $A=\begin{pmatrix}0 & e^{x}+1 & e^{-x}+1\\ e^{-x} & 0 & 1\\ e^{x} & 0 & 0\end{pmatrix}$, $A=\begin{pmatrix}-a & a & a\\ a & -a & a\\ a & a & -a\end{pmatrix}$.

13. Fiind dată matricea $A=\begin{pmatrix}x & 0 & x\\ 0 & y & 0\\ x & 0 & x\end{pmatrix}$, să se calculeze $\sum_{k=1}^{n} A^k$.

14. Să se calculeze în variantă recursivă şi iterativă primele n (n≥5 dat) elemente din şirul lui Fibonacci.

15. Problema turnurilor din Hanoi.

16. Să se calculeze recursiv a^n, n≥2.

1.Tipuri de subprograme recursive
2. Verificarea corectitudinii subprogramelor recursive
3. Complexitatea timp
4. Criterii de alegere; avantaje si dezavantaje
5. Greseli tipice in elaborarea subprogramelor recursive
6. Culegere de probleme re rezolvate si propuse

---

19. Să se calculeze recursiv C_n^k, $n \geq 2$, $0 \leq k \leq n$.
20. Să se scrie funcţia recursivă pentru calculul sumei cifrelor unui număr natural.
21. Să se scrie funcţia recursivă care citeşte un număr oarecare de caractere şi le afişează in ordinea inversă citirii. Atenţie! nu se lucrează cu şiruri, nu se cunoaşte numărul de caractere, iar sfârşitul şirului este dat de citirea caracterului ‘0’.
22. Se cere calculul recursiv al sumei a *n* numere naturale citite.
23. Să se realizeze programele recursive pentru problema aflării celui mai mare divizor comun pentru calculul simplu, dar şi pentru calculul folosind algoritmul lui Euclid.

1. Tipuri de subprograme recursive
2. Verificarea corectitudinii subprogramelor recursive
3. Complexitatea timp
4. Criterii de alegere; avantaje si dezavantaje
5. Greseli tipice in elaborarea subprogramelor recursive
6. Culegere de probleme re rezolvate si propuse

---

24. Să se caute rădăcina unei funcţii crescatoare, cunoscându-se următorul rezultat: Fie f o funcţie crescătoare. Dacă f(a) < 0 şi f(b) > 0, aunci f are rădăcină în intervalul [a, b].
25. Să se scrie programul *C/C*++ pentru realizarea căutării binare (recursiv şi iterativ) (se caută cheia v intr-un tablou sortat şi se returnează indicele).
26. Drum minim. Între n oraşe există o reţea de drumuri care permite ca dintr-un oraş să se ajungă în oricare dintre celelalte. Între două oraşe, există cel mult un drum direct, de lungime cunoscută, iar timpul necesar parcurgerii unui drum este proporţional cu lungimea sa. Să se scrie programul recursiv pentru determinarea traseului pe care se poate merge între două oraşe date, într-un timp minim.