

---

**Algoritmi și Structuri de Date (I). Seminar 2:** Descrierea în pseudocod a algoritmilor

- Care sunt principalele tipuri de prelucrări? Care prelucrări sunt considerate elementare?
- Cum putem descrie un algoritm?
- Exemple de algoritmi care prelucrează date numerice

---

**Problema 1** Fie  $n$  un număr natural nenul. Descrieți în pseudocod algoritmi pentru:

- Determinarea sumei tuturor cifrelor lui  $n$ . De exemplu, pentru  $n = 26326$  se obține valoarea 19.
- Determinarea valorii obținute prin inversarea cifrelor numărului  $n$ . De exemplu, pentru valoarea 26326 se obține valoarea 62362.
- Determinarea mulțimii tuturor cifrelor ce intervin în număr. De exemplu, pentru valoarea 26326 se obține mulțimea  $\{2, 3, 6\}$ .
- Determinarea tuturor cifrelor binare ale lui  $n$ .
- Determinarea tuturor divizorilor proprii ai lui  $n$ .
- A verifica dacă numărul  $n$  este prim sau nu (algoritmul returnează *true* dacă numărul este prim și *false* în caz contrar).
- Determinarea descompunerii în factori primi a lui  $n$ . De exemplu pentru  $490 = 2^1 \cdot 5^1 \cdot 7^2$  se obține mulțimea factorilor primi:  $\{2, 5, 7\}$  și puterile corespunzătoare:  $\{1, 1, 2\}$ .

*Rezolvare.*

- Cifrele numărului se extrag prin împărțiri succesive la 10. La fiecare etapă restul va reprezenta ultima cifră a valorii curente iar câtul împărțirii întregi va reprezenta următoarea valoare ce se va împărți la 10 (vezi `suma_cifre` în Algoritmul 1).
- Dacă  $n = c_k 10^k + \dots c_1 10 + c_0$  atunci numărul căutat este  $m = c_0 10^k + \dots c_{k-1} 10 + c_k = (\dots (c_0 10 + c_1) 10 + c_2 + \dots c_{k-1}) 10 + c_k$ . Cifrele lui  $n$  se extrag, începând de la ultima, prin împărțiri succesive la 10. Pentru a-l construi pe  $m$  este suficient să se pornească de la valoarea 0 și pentru fiecare cifră extrasă din  $n$  să se înmulțească  $m$  cu 10 și să se adune cifra obținută. Algoritmul corespunzător este `inversare_cifre` (Algoritmul 1).

---

**Algoritmul 1** Suma cifrelor și valoarea obținută prin inversarea ordinii cifrelor unui număr natural

---

<code>suma_cifre(integer n)</code>	<code>inversare_cifre(integer n)</code>
<code>integer S</code>	<code>integer m</code>
<code>S = 0</code>	<code>m = 0</code>
<code>while n &gt; 0 do</code>	<code>while n &gt; 0 do</code>
<code>S = S + n MOD 10</code>	<code>m = m * 10 + n MOD 10</code>
<code>n = n DIV 10</code>	<code>n = n DIV 10</code>
<code>end while</code>	<code>end while</code>
<code>return S</code>	<code>return m</code>

---

- Cifrele se determină prin împărțiri succesive la 10. Mulțimea cifrelor poate fi reprezentată fie printr-un tablou cu 10 elemente conținând indicatori de prezență (elementul de pe poziția  $i$ ,  $i = \overline{0, 9}$  este 1 dacă cifra  $i$  este prezentă în număr și 0 în caz contrar) fie printr-un tablou care conține cifrele distincte ce apar în număr. În al doilea caz, pentru fiecare cifră extrasă se verifică dacă nu se află deja în tabloul cu cifre. Cele două variante sunt descrise în Algoritmul 2.

- Cifrele binare ale lui  $n$  se obțin prin împărțiri succesive la 2 până se ajunge la valoarea 0. Restul primei împărțiri reprezintă cifra cea mai puțin semnificativă, iar restul ultimei împărțiri reprezintă cifra cea mai

---

**Algoritmul 2** Determinarea mulțimii cifrelor unui număr natural

---

<b>cifre1(integer <math>n</math>)</b>	<b>cifre2(integer <math>n</math>)</b>
<b>integer</b> $c[0..9]$ , $i$	<b>integer</b> $c[1..10]$ , $i, j$
<b>for</b> $i = 0, 9$ <b>do</b>	$i = 0$
$c[i] = 0$	<b>while</b> $n > 0$ <b>do</b>
<b>end for</b>	$r = n \text{ MOD } 10$ ; $j = 1$ ; $prezent = \text{false}$
<b>while</b> $n > 0$ <b>do</b>	<b>while</b> $j \leq i$ <b>and</b> $prezent == \text{false}$ <b>do</b>
$c[n \text{ MOD } 10] = 1$	<b>if</b> $c[j] == r$ <b>then</b>
$n = n \text{ DIV } 10$	$prezent = \text{true}$
<b>end while</b>	<b>else</b>
<b>return</b> $c[0..9]$	$j = j + 1$
	<b>end if</b>
	<b>end while</b>
	<b>if</b> $prezent == \text{false}$ <b>then</b>
	$i = i + 1$ ; $c[i] = r$
	<b>end if</b>
	$n = n \text{ DIV } 10$
	<b>end while</b>
	<b>return</b> $c[1..i]$

---

---

**Algoritmul 3** Determinarea cifrelor binare

---

```
cifre_binare(integer  $n$ )
integer  $b[0..k-1], i$ 
 $i = 0$ 
while  $n > 0$  do
     $b[i] = n \text{ MOD } 2$ 
     $i = i + 1$ 
     $n = n \text{ DIV } 2$ 
end while
return  $b[0..i-1]$ 
```

---

semnificativă. Presupunând că  $2^{k-1} \leq n < 2^k$  sunt suficiente  $k$  poziții binare pentru reprezentare, iar algoritmul poate fi descris prin Algoritmul 3.

Pentru a avea în tabloul  $b$  cifrele binare în ordinea naturală (începând cu cifra cea mai semnificativă) este suficient să se inverseze ordinea elementelor tabloului. Secvența de prelucrări care realizează acest lucru este:

```
for  $j = 0, \lfloor (i-1)/2 \rfloor$  do
     $b[j] \leftrightarrow b[i-1-j]$ 
end for
```

În secvența de mai sus operatorul de interschimbare  $\leftrightarrow$  presupune efectuarea a trei atribuiri și utilizarea unei variabile auxiliare ( $aux$ ) pentru reținerea temporară a valorii uneia dintre variabilele implicate în interschimbare:

```
 $aux = a$ 
 $a = b$ 
 $b = aux$ 
```

(e) Pentru determinarea divizorilor proprii ai lui  $n$  (divizori diferiți de 1 și  $n$ ) este suficient să se analizeze toate valorile cuprinse între 2 și  $\lfloor n/2 \rfloor$ . Algoritmul care afișează valorile divizorilor proprii poate fi descris prin:

```
divizori(integer  $n$ )
integer  $i$ 
for  $i = 2, \lfloor n/2 \rfloor$  do
    if  $n \text{ MOD } i == 0$  then
        write( $i$ )
    end if
end for
```

(f) Se pornește de la premiza că numărul este prim (inițializând variabila ce va conține rezultatul cu **true**) și se verifică dacă are sau nu divizori proprii (este suficient să se parcurgă domeniul valorilor cuprinse între 2 și  $\lfloor \sqrt{n} \rfloor$  pentru că nu dacă nu a fost identificat deja un divizor mai mic decât  $\lfloor \sqrt{n} \rfloor$  nu poate să existe divizor mai mare). La detectarea primului divizor se poate decide că numărul nu este prim. Aceste prelucrări sunt descrise în (sub)algoritmul **prim** din Algoritmul 4.

(g) Descompunerea în factori primi a numărului  $n$  se va reține în două tablouri: tabloul  $f$  conține valorile factorilor primi, iar tabloul  $p$  conține valorile puterilor corespunzătoare. Algoritmul este similar celui de determinare a divizorilor, însă când este descoperit un divizor se contorizează de câte ori se divide  $n$  prin acea valoare. Aceasta asigură faptul că divizorii ulteriori nu-i vor conține ca factori pe divizorii mai mici. O variantă a acestei metode este descrisă în (sub)algoritmul **factori\_primi** din Algoritmul 4.

**Problema 2** Fie  $n$  un număr natural,  $x$  o valoare reală din  $(0, 1)$  și  $\epsilon > 0$  o valoare reală pozitivă. Descrieți în pseudocod un algoritm pentru:

- Calculul sumei finite  $\sum_{i=1}^n (-1)^i x^{2i} / (2i)!$ .
- Calculul aproximativ al sumei infinite  $\sum_{i=1}^{\infty} (-1)^i x^{2i} / (2i)!$  cu precizia  $\epsilon$ .

*Indicație.*

(a) Suma poate fi rescrisă ca  $S = T(1) + \dots + T(n)$  cu  $T(i) = (-1)^i x^{2i} / (2i)!$ . Pentru a reduce numărul calculelor implicate de evaluarea fiecărui termen se poate folosi relația  $T(i) = -T(i-1) * x^2 / ((2i-1)2i)$  cu  $T(1) = -x^2/2$ .

(b) Calculul aproximativ al unei sume infinite presupune însumarea unui număr finit de termeni până când ultimul termen adunat este suficient de mic ( $T(i) < \epsilon$ ).

*Observație.* Această serie permite aproximarea funcției cosinus (vezi, de exemplu [https://en.wikipedia.org/wiki/Taylor\\_series](https://en.wikipedia.org/wiki/Taylor_series)).

---

**Algoritmul 4** Verificarea proprietății de număr prim și descompunerea în factori primi

---

<pre>prim(integer n) integer i, boolean p p = true i = 2 while i ≤ ⌊√n⌋ and p == true do   if n MOD i == 0 then     p = false   else     i = i + 1   end if end while return p</pre>	<pre>factori_primi(integer n) integer f[1..k], p[1..k], i, d i = 0 d = 2 repeat   if n MOD d == 0 then     i = i + 1     f[i] = d     p[i] = 1     n = n DIV d   while n MOD d == 0 do     p[i] = p[i] + 1     n = n DIV d   end while   d = d + 1 until n &lt; d return f[1..i], p[1..i]</pre>
--	---

---

**Problema 3** Să se afișeze primele  $N$  elemente și să se aproximeze (cu precizia  $\epsilon$ ) limitele șirurilor (cu excepția șirului de la punctul (d) care nu este neapărat convergent):

- (a)  $x_n = (1 + 1/n)^n$ ;
- (b)  $x_1 = a > 0$ ,  $x_n = (x_{n-1} + a/x_{n-1})/2$ ;
- (c)  $x_n = f_{n+1}/f_n$ ,  $f_1 = f_2 = 1$ ,  $f_n = f_{n-1} + f_{n-2}$ ;
- (d)  $x_1 = s$ ,  $x_n = (ax_{n-1} + b) \text{MOD} c$ ,  $a, b, c \in N^*$ .

*Rezolvare.*

(a) Afișarea primelor  $N$  elemente ale șirului  $x_n$  este descrisă în algoritmul `elemente_sir`. În cazul unui șir convergent, limita poate fi aproximată prin elementul  $x_L$  care satisface proprietatea  $|x_L - x_{L-1}| < \epsilon$ . În acest caz trebuie cunoscută la fiecare etapă atât valoarea curentă ( $xc$ ) cât și valoarea precedentă a șirului ( $xp$ ). O variantă de estimare a limitei este ilustrată în algoritmul `limita_sir`.

<pre>elemente_sir(integer N) integer n for n = 1, N do   write putere(1 + 1/n, n) end for putere(real x, integer n) real p integer i p = 1 for i = 1, n do   p = p * x end for return p</pre>	<pre>limita_sir(real eps) integer n real xc, xp n = 1 xc = 2 repeat   xp = xc   n = n + 1   xc = putere((n + 1/n), n) until  xc - xp  ≤ eps return xc</pre>
---	---

(b) Pentru determinarea elementelor unui șir dat printr-o relație de recurență de ordin  $r$ ,  $x_n = f(x_{n-1}, \dots, x_{n-r})$  pentru care se cunosc primele  $r$  valori se pot utiliza  $r + 1$  variabile:  $r$  care conțin valorile anterioare din șir ( $p_1, \dots, p_r$ ) și una care conține valoarea curentă ( $p_0$ ). Structura generală a acestei prelucrări este descrisă în algoritmul `sir_recurent`.

```
sir_recurent(integer N, real x[1..r])
```

```

integer  $i$ 
real  $p[0..r]$ 
 $p[1..r] = x[1..r]$ 
for  $i = r + 1, N$  do
    for  $k = r, 1, -1$  do
         $p[k] = p[k - 1]$ 
    end for
     $p[0] = f(p[1], \dots, p[r])$ 
    write  $p[0]$ 
end for

```

Atribuirea  $p[1..r] = x[1..r]$  semnifică faptul că elementelor cu indicii cuprinși între 1 și  $r$  din tabloul  $p$  li se asignează valorile elementelor corespunzătoare din tabloul  $x$ . În cazul în care  $r = 1$  algoritmul devine mai simplu. Generarea elementelor cât și estimarea limitei (șirul converge către  $\sqrt{a}$ ) sunt descrise în Algoritmul 5

---

**Algoritmul 5** Generarea elementelor și estimarea limitei unui șir dat printr-o relație de recurență simplă

---

<b>sir_recurent2(integer</b> $N$ , <b>real</b> $a$ )	<b>limita_sir2(real</b> $a, eps$ )
<b>integer</b> $n$	<b>real</b> $xp, xc$
<b>real</b> $x$	$xc = a$
$x = a$	<b>repeat</b>
<b>for</b> $n = 2, N$ <b>do</b>	$xp = xc$
$x = (x + a/x)/2$	$xc = (xp + a/xp)/2$
<b>write</b> $x$	<b>until</b> $ xp - xc  < eps$
<b>end for</b>	<b>return</b> $xc$

---

(c) Șirul  $f_n$  este dat printr-o relație de recurență de ordin 2 și este cunoscut ca fiind *șirul lui Fibonacci*. Se poate demonstra că șirul  $x_n$  converge către  $\theta = (1 + \sqrt{5})/2$ . Generarea elementelor și estimarea limitei sunt descrise în Algoritmul 6.

---

**Algoritmul 6** Șirul rapoartelor elementelor din șirul Fibonacci

---

<b>sir3(integer</b> $N$ )	<b>limita_sir3(real</b> $eps$ )
<b>integer</b> $n, f0, f1$	<b>integer</b> $f0, f1$
<b>real</b> $x$	<b>real</b> $xc, xp$
$f0 = 1$	$f0 = 1$
$f1 = 1$	$f1 = 1$
<b>for</b> $1 = 3, N$ <b>do</b>	$xc = f0/f1$
<b>write</b> $f0/f1$	<b>repeat</b>
$f2 = f1$	$xp = xc$
$f1 = f0$	$f2 = f1$
$f0 = f1 + f2$	$f1 = f0$
<b>end for</b>	$f0 = f1 + f2$
	$xc = f0/f1$
	<b>until</b> $ xc - xp  \leq eps$
	<b>return</b> $xc$

---

(d) Relațiile de recurență de acest tip sunt folosite pentru generarea de valori (pseudo)aleatoare și sunt utilizate pentru implementarea algoritmilor aleatori. Valorile generate sunt de relația dată sunt între 0 și  $c - 1$ . Metoda de generare este descrisă în algoritmul **sir\_pseudoaleator**.

```

sir_pseudoaleator(integer  $N, s, a, b, c$ )
integer  $x$ 
 $x = s$ 
for  $n = 2, N$  do

```

```
 $x = (a * x + b) \text{ MOD } c;$  write  $x$   
end for
```

### Probleme suplimentare/teme

1. Transformați toate prelucrările repetitive descrise prin **repeat** în algoritmii de la exemplele de mai sus în prelucrări descrise prin **while**.
2. Să se descrie în pseudocod și să se implementeze algoritmul înmulțirii "à la russe".
3. Să se descrie în pseudocod și să se implementeze algoritmul metoda de sortare bazată pe răsturnarea unei subsecvențe finale a șirului (problema clătitorilor).
4. Propuneți un algoritm care aplicat pentru două șiruri cu același număr de elemente decide dacă unul dintre șiruri poate fi obținut din celălalt printr-o singură răsturnare a unei secvențe finale (în cazul a două stive de clătite înseamnă că una este obținută din cealaltă prin aplicarea unei singure operații de răsturnare)
5. Algoritm care determină numărul de cifre binare egale cu 1 din reprezentarea în baza 2 a valorii naturale nenule  $n$ .
6. Algoritm care să determine cifra ce apare cel mai frecvent într-un număr natural nenul  $n$ . Dacă sunt mai multe astfel de cifre se vor afișa toate.
7. Algoritm pentru calculul sumei  $\sum_{i=0}^n (-1)^i x^{2(i+1)} / (2(i+1))!$  și pentru aproximarea sumei infinite corespunzătoare.
8. Algoritm pentru a afișa primele  $N$  elemente ale șirului lui Fibonacci și care folosește doar două variabile de lucru pentru a reține elementele șirului.