Algoritmi şi 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. Descrieţi în pseudocod algoritmi pentru:

(a) Determinarea sumei tuturor cifrelor lui n. De exemplu, pentru $n = 26326$ se obţine valoarea 19.

(b) Determinarea valorii obţinute prin inversarea cifrelor numărului n. De exemplu, pentru valoarea 26326 se obţine valoarea 62362.

(c) Determinarea mulţimii tuturor cifrelor ce intervin în număr. De exemplu, pentru valoarea 26326 se obţine mulţimea $\{2, 3, 6\}$.

(d) Determinarea tuturor cifrelor binare ale lui n.

(e) Determinarea tuturor divizorilor proprii ai lui n.

(f) A verifica dacă numărul n este prim sau nu (algoritmul returnează *true* dacă numărul este prim şi *false* în caz contrar).

(g) Determinarea descompunerii în factori primi a lui n. De exemplu pentru $490 = 2^1 \cdot 5^1 \cdot 7^2$ se obţine mulţimea factorilor primi: $\{2, 5, 7\}$ şi puterile corespunzătoare: $\{1, 1, 2\}$.

Rezolvare.
(a) Cifrele numărului se extrag prin împărţiri succesive la 10. La fiecare etapă restul va reprezenta ultima cifră a valorii curente iar câtul împărţirii întregi va reprezenta următoarea valoare ce se va împărţi la 10 (vezi `suma_cifre` in Algoritmul 1).

(b) 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 împărţiri succesive la 10. Pentru a-l construi pe m este suficient să se pornească de la valoarea 0 şi pentru fiecare cifră extrasă din n să se înmulţească m cu 10 şi să se adune cifra obţinută. Algoritmul corespunzător este `inversare_cifre` (Algoritmul 1).

Algoritmul 1 Suma cifrelor şi valoarea obţinută prin inversarea ordinii cifrelor unui număr natural

`suma_cifre`(**integer** n)	`inversare_cifre`(**integer** n)
integer S	**integer** m
$S = 0$	$m = 0$
while $n > 0$ **do**	**while** $n > 0$ **do**
$S = S + n$ MOD 10	$m = m * 10 + n$ MOD 10
$n = n$ DIV 10	$n = n$ DIV 10
end while	**end while**
return S	**return** m

(c) Cifrele se determină prin împărţiri succesive la 10. Mulţimea cifrelor poate fi reprezentată fie printr-un tablou cu 10 elemente conţinând indicatori de prezenţă (elementul de pe poziţia i, $i = \overline{0, 9}$ este 1 dacă cifra i este prezentă în număr şi 0 în caz contrar) fie printr-un tablou care conţine 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.

(d) Cifrele binare ale lui n se obţin prin împărţiri succesive la 2 până se ajunge la valoarea 0. Restul primei împărţiri reprezintă cifra cea mai puţin semnificativă, iar restul ultimei împărţiri reprezintă cifra cea mai

Algoritmul 2 Determinarea mulţimii cifrelor unui număr natural

```
cifre1(integer n)
integer c[0..9], i
for i = 0, 9 do
   c[i] = 0
end for
while n > 0 do
   c[nMOD10] = 1
   n = n DIV 10
end while
return c[0..9]
```

```
cifre2(integer n)
integer c[1..10], i, j
i = 0
while n > 0 do
   r = n MOD 10; j = 1; prezent = false
   while j <= i and prezent ==false do
      if c[j] == r then
         prezent = true
      else
         j = j + 1
      end if
   end while
   if prezent ==false then
      i = i + 1; c[i] = r
   end if
   n = n DIV 10
end while
return 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 MOD 2
   i = i + 1
   n = n DIV 2
end while
return b[0..i − 1]
```

semnificativă. Presupunând că $2^{k-1} \leq n < 2^k$ sunt suficiente k poziţii 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. Secvenţa 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 secvenţa de mai sus operatorul de interschimbare $\leftrightarrow$ presupune efectuarea a trei atribuiri şi utilizarea unei variabile auxiliare (aux) pentru reţinerea temporară a valorii uneia dintre variabilele implicate în interschimbare:

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

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

`divizori`(**integer** n)
integer i
for $i = 2, \lfloor n/2 \rfloor$ **do**
  if n MOD $i == 0$ **then**
    print(i)
  end if
end for

(f) Se porneşte de la premiza că numărul este prim (iniţializând variabila ce va conţine rezultatul cu **true**) şi se verifică dacă are sau nu divizori proprii (este suficient să se parcurgă domeniul valorilor cuprinse între 2 şi $\lfloor \sqrt{n} \rfloor$ 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 reţine în două tablouri: tabloul f conţine valorile factorilor primi, iar tabloul p conţine 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 conţine 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 $(0,1)$ şi $\epsilon > 0$ o valoare reală pozitivă. Descrieţi în pseudocod un algoritm pentru:

(a) Calculul sumei finite $\sum_{i=1}^{n} (-1)^i x^{2i}/(2i)!$.

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

Indicaţie.
(a) Suma poate fi rescrisă ca $S = T(1) + \ldots + T(n)$ cu $T(i) = (-1)^i x^{2i}/(2i)!$. Pentru a reduce numărul calculelor implicate de evaluarea fiecărui termen se poate folosi relaţia $T(i) = -T(i-1) * x^2/((2i-1)2i)$ 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 ($T(i) < \epsilon$).
Observaţie. Această serie permite aproximarea funcţiei cosinus (vezi, de exemplu https://en.wikipedia.org/wiki/Taylor_series).

Algoritmul 4 Verificarea proprietăţii de număr prim şi descompunerea în factori primi

```
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
```

```
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
  end if
  d = d + 1
until n < d
return f[1..i], p[1..i]
```

Problema 3 Să se afişeze primele N elemente şi să se aproximeze (cu precizia ϵ) limitele şirurilor (cu excepţia şirului 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) Afişarea primelor N elemente ale şirului x_n este descrisă în algoritmul `elemente_sir`. În cazul unui şir 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 şi valoarea precedentă a şirului (xp). O variantă de estimare a limitei este ilustrată în algoritmul `limita_sir`.

```
elemente_sir(integer N)
integer n
for n = 1, N do
  print 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
```

```
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
```

(b) Pentru determinarea elementelor unui şir dat printr-o relaţie de recurenţă de ordin r, $x_n = f(x_{n-1}, \ldots, x_{n-r})$ pentru care se cunosc primele r valori se pot utiliza $r + 1$ variabile: r care conţin valorile anterioare din şir ($p_1, \ldots p_r$) şi una care conţine 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], . . . , p[r])
   print p[0]
end for
```

Atribuirea $p[1..r] = x[1..r]$ semnifică faptul că elementelor cu indicii cuprinşi între 1 şi 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 şi estimarea limitei (şirul converge către $\sqrt{a}$) sunt descrise în Algoritmul 5

Algoritmul 5 Generarea elementelor şi estimarea limitei unui şir dat printr-o relaţie de recurenţă simplă

```
sir_recurent2(integer N, real a)
integer n
real x
x = a
for n = 2, N do
   x = (x + a/x)/2
   print x
end for
```

```
limita_sir2(real a,eps)
real xp,xc
xc = a
repeat
   xp = xc
   xc = (xp + a/xp)/2
until |xp − xc| < eps
return xc
```

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

Algoritmul 6 Şirul rapoartelor elementelor din şirul Fibonacci

```
sir3(integer N)
integer n, f0,f1
real x
f0 = 1
f1 = 1
for 1 = 3, N do
   print f0/f1
   f2 = f1
   f1 = f0
   f0 = f1 + f2
end for
```

```
limita_sir3(real eps)
integer f0, f1
real xc, xp
f0 = 1
f1 = 1
xc = f0/f1
repeat
   xp = xc
   f2 = f1
   f1 = f0
   f0 = f1 + f2
   xc = f0/f1
until |xc − xp| ≤ eps
return xc
```

(d) Relaţiile de recurenţă de acest tip sunt folosite pentru generarea de valori (pseudo)aleatoare şi sunt utilizate pentru implementarea algoritmilor aleatori. Valorile generate sunt de relaţia dată sunt între 0 şi $c - 1$. 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
```

$x = (a * x + b)$ MOD c; **print** x
end for

Probleme suplimentare/teme

1. Transformaţi 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 şi să se implementeze algoritmul înmulţirii ”à la russe”.
3. Să se descrie în pseudocod şi să se implementeze algoritmul metoda de sortare bazată pe răsturnarea unei subsecvenţe finale a şirului (problema clătitelor).
4. Propuneţi un algoritm care aplicat pentru două şiruri cu acelaşi număr de elemente decide dacă unul dintre şiruri poate fi obţinut din celălalt printr-o singură răsturnare a unei secvenţe finale (în cazul a două stive de clătite înseamnă că una este obţinută din cealaltă prin aplicarea unei singure operaţii 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 afişa toate.
7. Algoritm pentru calculul sumei $\sum_{i=0}^{n}(-1)^i x^{2(i+1)}/(2(i+1))!$ şi pentru aproximarea sumei infinite corespunzătoare.
8. Algoritm pentru a afişa primele N elemente ale şirului lui Fibonacci şi care foloseşte doar două variabile de lucru pentru a reţine elementele şirului.