ALGORITMICĂ. Seminar 1: Rezolvarea algoritmică a problemelor. Identificarea algoritmilor si a proprietăţilor lor. Descrierea algoritmilor in pseudocod.

Problema 1 (*Inmulţirea à la russe.*)(S [1]) Se consideră următoarea metodă de înmulţire (numită înmulţirea "à la russe") a două numere naturale nenule x şi y: "Se scrie x alături de y (pe aceeaşi linie). Se împarte x la 2 şi câtul impărţirii se scrie sub x (restul se ignoră deocamdată). Se înmulţeşte y cu 2 iar produsul se scrie sub y. Procedeul continuă construindu-se astfel două coloane de numere. Calculele se opresc în momentul în care pe prima coloană se obţine valoarea 1. Se adună toate valorile de pe coloana a doua care corespund unor valori impare aflate pe prima coloană."

Exemplu. Fie $x = 13$ şi $y = 25$. Succesiunea de rezultate obţinute prin aplicarea operaţiilor de mai sus este:

x	y	rest	factor multiplicare y
13	$\boxed{25}$	1	2^0
6	50	0	2^1
3	$\boxed{100}$	1	2^2
1	$\boxed{200}$	1	2^3
	325		

a) Prelucrarea descrisă prin metoda de mai sus se termină întotdeauna după un număr finit de paşi ? Ce se întâmplă în cazul în care una dintre valori este egală cu 0 ?

b) Metoda de mai sus conduce întotdeauna la produsul celor două numere (cu alte cuvinte este o metodă corectă de înmulţire) ?

c) Câte operaţii de înmulţire cu 2 sunt necesare ? Depinde acest număr de ordinea factorilor ?

Indicaţie.

a) Ca urmare a împărţirilor succesive la 2 valorile de pe prima coloană descresc până se ajunge la un cât egal cu 1 (în ipoteza că x este nenul). Prin urmare prelucrarea este finită. Dacă x este egal cu 0 metoda nu poate fi aplicată ca atare, însă dacă y este 0 ea conduce la un rezultat corect.

b) Corectitudinea prelucrării derivă din faptul că metoda realizează de fapt conversia în baza 2 a primului număr (cifrele reprezentării în baza doi sunt resturile obţinute prin împărţirile succesive la 2) iar produsul se obţine prin înmulţirea succesivă a celui de al doilea număr cu puteri ale lui 2 şi prin însumarea acelor produse care corespund unor cifre nenule în reprezentarea binară a primului număr.

c) Numărul înmulţirilor cu 2 este cu 1 mai mic decât numărul de cifre ale reprezentării binare a lui x, adică $[\log_2 x]$. Evident numărul înmulţirilor depinde de ordinea factorilor fiind mai mic dacă împărţirile la 2 se efectuează asupra celui mai mic număr.

Link: http://mathforum.org/dr.math/faq/faq.peasant.html

Problema 2 (S) Propuneţi o metodă bazată pe operaţii de adunare, scădere şi comparare pentru determinarea părţii întregi a unui număr real. Partea întreagă (numită uneori parte întreagă inferioară şi pentru un număr real x este notată cu $\lfloor x \rfloor$) a unui număr real este definită ca fiind cel mai mare număr întreg mai mic decât numărul real. De exemplu, partea întreagă a lui 2.4 este 2, iar pentru -2.4 este -3.

Indicaţie. Se va ţine cont de semnul numărului. În cazul în care este pozitiv se scade succesiv valoarea 1 până când se ajunge la o valoare mai mică strict decât 1. Numărul scăderilor efectuate indică valoarea părţii întregi. Pentru numere negative se adună succesiv 1 până se obţine o valoare mai mare sau egală cu 0. Opusul numărului de adunări reprezintă valoarea părţii întregi.

Exemplu.

[1] S = seminar, L = laborator

$x > 0$		$x < 0$	
x	contor	x	contor
3.25	0	-3.25	0
2.25	1	-2.25	-1
1.25	2	-1.25	-2
0.25	$\boxed{3}$	-0.25	-3
		0.75	$\boxed{-4}$

Problema 3 (*Problema identificării monedei mai uşoare.*) (S) Se consideră un set de n monede identice, cu excepţia uneia care are greutatea mai mică decât celelalte. Folosind o balanţă simplă să se identifice moneda cu greutatea mai mică folosind cât mai puţine comparaţii.

Indicaţie. O primă variantă este aceea prin care se selectează la întâmplare două monede şi se pun pe balanţă. Dacă una dintre ele are greutatea mai mică atunci este chiar moneda căutată. Daca ambele au aceeaşi greutate se păstrează una dintre ele pe un taler al balanţei iar pe celălalt taler se pun succesiv monedele rămase. Prelucrarea se opreşte în momentul în care se găseşte o monedă cu greutatea mai mică. Numărul de comparaţii este cel mult $n - 1$.

O altă variantă este cea în care se împarte setul de monede în două subseturi (fiecare va avea $n/2$ elemente, dacă n e par, respectiv $(n-1)/2$ daca n este impar) care se pun pe cele două talere ale balanţei. Dacă subseturile au aceeaşi greutate (ceea ce se poate întâmpla doar daca n este impar) atunci moneda pusă deoparte este cea căutată. În caz contrar se aplică acelaşi procedeu subsetului de greutate mai mică şi se continuă până se ajunge la un set de două sau trei monede. În acest moment este suficient să se mai efectueze o singură cântărire. Numărul de cântăriri este cel mult $\lfloor \log_2 n \rfloor$. Este adevărat acest rezultat şi pentru cazul în care se ştie doar că una dintre monede este diferită dar nu se ştie dacă este mai uşoară sau mai grea decât celelalte?

Problema 4 (*Problema clătitelor.*) (S) La un restaurant bucătarul a pregătit clătite (americane) pe care le-a aşezat pe un platou sub forma unei stive. Din păcate nu toate clătitele au acelaşi diametru astfel că stiva nu arată estetic. Chelnerul ia platoul şi având la dispoziţie o spatulă reuşeşte să aranjeze cu o singură mână clătitele astfel încât să fie în ordinea descrescătoare a diametrelor (cea mai mare clatită pe platou iar cea mai mică în vârful stivei). Cum a procedat? Descrieţi problema într-o manieră abstractă şi propuneţi un algoritm de rezolvare.

Indicaţie. Rearanjarea se face efectuând doar mişcări de răsturnare a unui "set" de clătite dintre cele aflate in partea de sus a stivei. Problema este identică cu cea a ordonării descrescătoare a unui şir de valori prin inversarea ordinii elementelor unor subsecvenţe de la sfârşitul şirului.

Exemplu. Pentru a ordona descrescător şirul $(5, 3, 4, 1, 6, 2)$ se aplică următoarea secvenţă de prelucrări în care subsecvenţa care se "răstoarnă" este încadrată:

5 3 4 1 $\boxed{6\ 2}$
$\boxed{5\ 3\ 4\ 1\ 2\ 6}$
6 $\boxed{2\ 1\ 4\ 3\ 5}$
6 5 3 $\boxed{4\ 1\ 2}$
6 5 $\boxed{3\ 2\ 1\ 4}$
6 5 4 $\boxed{1\ 2\ 3}$
6 5 4 3 2 1

Metoda constă în determinarea valorii maxime din secvenţă şi inversarea ordinii subsecvenţei care începe cu ea pentru a fi adusă pe ultima poziţie în şir. După aceea se răstoarnă întregul şir, astfel că valoarea maximă ajunge pe prima poziţie. Se aplică aceeaşi metodă pentru subsecvenţa care începe cu al doilea element şi se continuă până se ajunge la o subsecvenţă constituită dintr-un singur element.

Problema 5 (S) Considerăm un set de programe pentru care se cunosc estimări ale timpilor de execuţie şi care trebuie executate utilizând două calculatoare similare. Propuneţi o modalitate de distribuire a execuţiei programelor pe cele două calculatoare astfel încât încărcarea acestora să fie cât mai echilibrată.

Indicaţie. Problema poate fi reformulată astfel: se consideră o mulţime de numere pozitive, $A = \{a_1, a_2, \ldots, a_n\}$, $n > 1$ şi se cere să se determine două submulţimi disjuncte B şi C astfel încât $B \cup C = A$ şi $|\sum_{a \in B} a - \sum_{a \in C} a|$ este minimă. Cea mai simplă metodă este cea a "forţei brute" prin care se generează toate perechile de submulţimi (B, C) şi se alege cea care minimizează diferenţa specificată. Numărul de partiţii distincte (fără a ţine cont de ordinea calculatoarelor) este $2^{n-1} - 1$.Pentru n mare, numărul de partiţii ce trebuie testate devine mare (pentru $n = 10$ este 511 iar pentru $n = 100$ este de ordinul 10^{29}). E evident că în astfel de situaţii metoda forţei brute nu este eficientă, astfel că trebuie căutate metode mai puţin costisitoare (astfel de metode vor fi discutate în seminariile viitoare).

Problema 6 (S) Se consideră o mulţime cu n elemente (de exemplu $U = \{1, 2, \ldots, n\}$) şi un set S de submulţimi ale acesteia. Se pune problema determinării unei acoperiri minimale a mulţimii U folosind submulţimi din setul S (o acoperire minimală este un subset $A \subset S$ cu proprietatea că $\cup_{s \in A} s = U$ iar numărul de elemente din A este minim). De exemplu pentru $n = 5$ se poate considera setul de submulţimi $S = \{S_1, S_2, S_3, S_4\}$ cu $S_1 = \{1, 3, 5\}$, $S_2 = \{1, 4\}$, $S_3 = \{2, 4\}$, $S_4 = \{2, 5\}$. In acest caz acoperirea minimală se obţine selectând prima dată submulţimea cu cele mai multe elemente (S_1), eliminând din U elementele submulţimii alese (U devine $\{2, 4\}$) şi continuând prin selectarea unei submulţimi care conţine elemente din U (în acest exemplu ar fi S_3). Garantează strategia de a alege la fiecare etapa cea mai numeroasă mulţime care acoperă elementele rămase obţinerea unei soluţii optime? Propuneţi un contraexemplu.

Problema 7 (S+L) 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 $n = c_k 10^k + \ldots c_1 10 + c_0$ atunci numărul căutat este $m = c_0 10^k + \ldots c_{k-1} 10 + c_k = (\ldots(c_0 10 + c_1)10 + c_2 + \ldots c_{k-1})10 + c_k$. 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).

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

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

```
suma_cifre(integer n)
integer S
S ← 0
while n > 0 do
   S ← S + n MOD 10
   n ← n DIV 10
end while
return S
```

```
inversare_cifre(integer n)
integer m
m ← 0
while n > 0 do
   m ← m * 10 + n MOD 10
   n ← n DIV 10
end while
return m
```

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]
```

```
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]
```

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, ⌊(i − 1)/2⌋ do
   b[j] ↔ 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, ⌊n/2⌋ do
   if n MOD i = 0 then
      write(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 $\lfloor \sqrt{n} \rfloor$). La detectarea primului divizor se poate decide că numărul nu este prim. Aceste prelucrări sunt descrise în (sub)algoritmul `prim` din Algoritmul 3.

(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 3.

Problema 8 (S+L) 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 $\sum_{i=1}^{n}(-1)^i x^{2i}/(2i)!$.

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

Algoritmul 3 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]
```

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 ($T(i) < \epsilon$).

Problema 9 (S+L) 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
  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
```

```
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])
   write p[0]
end for
```

Atribuirea $p[1..r] \leftarrow x[1..r]$ 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 4

Algoritmul 4 Generarea elementelor şi estimarea limitei unui şi 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
   write 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 5.

Algoritmul 5 Şirul rapoartelor elementelor din şirul Fibonacci

```
sir3(integer N)
integer n, f0,f1
real x
f0 ← 1
f1 ← 1
for 1 ← 3, N do
   write 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; write x
end for
```

Probleme suplimentare.

1. Să se determine valoarea în baza 10 a unui număr natural pornind de la şirul cifrelor sale binare. Se va selecta un algoritm bazat doar pe operaţii elementare (adunare, scădere, înmulţire - ridicarea la putere nu este considerată operaţie elementară) iar numărul acestora este cât mai mic.

2. Se consideră un număr constituit din 10 cifre distincte, diferit de 9876543210. Să se determine numărul care îl succede în şirul crescător al tuturor numerelor naturale ce conţin 10 cifre distincte.

3. Se consideră un şir de valori nenule (pozitive şi negative). Să se transforme şirul astfel încât toate valorile negative să le preceadă pe cele pozitive iar numărul operaţiilor efectuate să fie cât mai mic.

4. Se consideră un şir cu $n-1$ valori distincte din mulţimea $\{1, 2, \ldots, n\}$. Să se determine, folosind un număr cât mai mic de comparaţii, valoarea care lipseşte din şir.

5. Se consideră următorul joc cu doi jucători bazat pe două stive de monede: una conţinând m monede şi cealaltă conţinând n monede. La fiecare etapă jucătorul care este la rând poate ridica o monedă dintr-una dintre stive sau câte o monedă din ambele stive. Câştigă jucătorul care ridică ultima/ultimele monede. Propuneţi un algoritm care pentru o pereche de valori (n, m) decide dacă pentru jucătorul care face prima mutare există o strategie câştigătoare.

Teme seminar/laborator

1. Să se descrie în pseudocod şi să se implementeze algoritmul înmulţirii "à la russe".

2. Să se descrie în pseudocod şi să se implementeze algoritmul metoda de sortare bazată pe răsturnarea unei subsecvenţe finale a şirului (problema clătitelor).

3. 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)

4. Algoritm care determină numărul de cifre binare egale cu 1 din reprezentarea în baza 2 a valorii naturale nenule n. Scrieţi programul aferent.

5. 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. Scrieţi programul aferent.

6. Algoritm pentru calculul sumei $\sum_{i=1}^{n}(-1)^{i+1}x^{2(i+1)}/(2(i+1))!$ şi pentru aproximarea sumei infinite corespunzătoare. Scrieţi programul aferent.

7. Algoritm pentru a afişa primele N elemente ale şirului lui Fibonacci şi care foloseşte doar două variabile de lucru pentru a reţine elementele şirului. Scrieţi programul aferent.