Tema 1: Rezolvarea algoritmică a problemelor. Descrierea algoritmilor în pseudocod. Verificarea corectitudinii algoritmilor. Analiza eficienţei algoritmilor.

Termen: 11.11.2018

Submiterea temelor: Se uploadează prin Classroom (cf indicaţiilor primite de la cadrul didactic coordonator de seminar) o arhiva de tip *.zip având numele construit pe baza regulii: NumeStudent_seria_subgrupa_Tema1.zip (de exemplu `AdamEva_IR_sgr3_Tema1.zip`).

Arhiva trebuie să conţină următoarele fişiere:

- Un fişier în format PDF sau DOC care să conţină soluţiile propuse (răspunsuri la întrebări, algoritmi descrişi în pseudocod, explicaţii etc.)
- Câte un fişier cu codul sursă Python corespunzător implementărilor solicitate în enunţ denumit NumeStudent_seria_subgrupa_Tema1_Problema (de exemplu `AdamEva_IR_sgr3_Tema1_Pb1b.py`)

Punctaj total: 100p (+ bonus max 10p)

---

1. *Reprezentarea numerelor întregi în complement faţă de 2* pe k biţi se caracterizează prin: (i) primul bit este folosit pentru codificarea semnului (0 pentru valori pozitive respectiv 1 pentru valori negative); (ii) ceilalţi $(k-1)$ biţi sunt utilizaţi pentru reprezentarea în baza 2 a valorii (cifrele binare corespunzătoare în cazul numerelor pozitive, respectiv valori complementate după regula specifică în cazul numerelor negative). *Exemplu:* Pentru $k = 8$ reprezentarea lui 12 este 00001100 iar reprezentarea lui -12 este 11110100 (în cazul valorilor negative, şirul cifrelor binare este parcurs începând cu cifra cea mai puţin semnificativă până la întâlnirea primei valori egale cu 1 - toate cifrele binare parcurse sunt lăsate nemodificate, iar toate cele care vor fi parcurse ulterior vor fi complementate).

   (a)(2p) Care este cel mai mare şi cel mai mic număr întreg care pot fi reprezentate (în complement faţă de 2) pe 16 poziţii binare (biţi)? Argumentaţi răspunsul.

   (b)(5p) Propuneţi un algoritm care construieşte reprezentarea în complement faţă de 2 pe 16 poziţii binare a unui număr întreg primit ca parametru. *Exemplu:* pentru valoarea 12 se obţine $[0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0]$ iar pentru -12 se obţine $[1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,0]$.

   (c)(5p) Propuneţi un algoritm care determină valoarea unui număr întreg (pozitiv sau negativ) pornind de la reprezentarea în complement faţă de 2 pe 16 biţi.

   (d)(8p) Propuneţi un algoritm care calculează suma a două numere întregi date prin reprezentările lor în complement faţă de 2 pe $k = 8$ poziţii binare. Identificaţi cazurile în care se produce *depăşire* (rezultatul nu poate fi reprezentat corect pe $k = 8$ poziţii binare). De exemplu, prin adunarea cifrelor binare aflate pe aceeaşi poziţie (începând de la cea mai puţin semnificativă poziţie) şi transferul reportului către poziţia imediat superioară (ca semnificaţie) pentru $[0,1,1,1,1,0,0,0]$ şi $[0,0,1,1,1,0,0,1]$ s-ar obţine $[1,0,1,1,0,0,0,1]$, ceea ce nu e corect însemnând că s-a produs o depăşire.

   Fiecare dintre algoritmii de mai sus va fi descris în pseudocod şi implementat în Python.

2. *Aproximarea funcţiei logaritmice prin serii.* Funcţia *ln* poate fi aproximată folosind următoarele serii:

$$f(x) = \begin{cases} \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}(x-1)^n & 0 < x \le 1 \\ \sum_{n=1}^{\infty} \frac{1}{n}\left(\frac{x-1}{x}\right)^n & x > 1 \end{cases}$$

(a)(2p) Identificaţi regula de calcul a termenului următor (T_{n+1}) din fiecare serie folosind valoarea termenului curent ($T_n = \dfrac{(-1)^{n+1}}{n}(x-1)^n$ respectiv $T_n = ((x-1)/x)^n/n$).

(b)(8p) Pentru fiecare dintre cele două serii descrieţi algoritmul (şi funcţiile Python corespunzătoare) care aproximează suma seriei prin suma finită $T_1 + T_2 + \ldots + T_k$. Numărul de termeni din sumă se stabileşte în funcţie de valoarea ultimului termen adăugat (T_k este primul termen cu proprietatea că $|T_k| < \epsilon$, ϵ fiind o constantă cu valoare mică). Date de test: $\epsilon = 10^{-5}$, $x = 0.5$, $x = 1$,$x = 5$, $x = 10$. Determinaţi în fiecare caz numărul de termeni incluşi în sumă.

(c)(5p) Pentru unul dintre algoritmii propuşi la punctul (b) (la alegere) identificaţi un invariant şi demonstraţi corectitudinea algoritmului.

(d)(5p) Descrieţi un algoritm pentru estimarea erorii de aproximare $\sum_{i=1}^{m}(f(i \cdot h) - ln(i \cdot h))^2$. Date de test: $m = 100$, $h = 5/m$. Pentru implementarea în Python pentru calculului lui $ln(i \cdot h)$ se va folosi funcţia `log` din pachetul `math`.

3. Se consideră algoritmii `alg1` şi `alg2`. Pentru fiecare dintre cei doi algoritmi:

(a)(5p) Implementaţi algoritmul în Python.

(b)(10p) Stabiliţi ce returnează fiecare dintre algoritmi atunci când este apelat pentru valori naturale nenule ale parametrilor. Identificaţi o proprietate invariantă şi demonstraţi corectitudinea fiecărui algoritm. *Indicaţie:* pentru `alg2` se poate folosi proprietatea că pentru orice număr natural nenul n există un număr natural $k > 0$ astfel încât $2^{k-1} \leq n < 2^k$.

```
alg1(int a, b)
if a < b then
    a ↔ b
end if
c ← 0
d ← a
while d > b do
    c ← c + 1
    d ← d − b
end while
return c, d
```

```
alg2(int n)
i ← 1
x ← 0
while i <= n do
    i ← 2 * i
    x ← x + 1
end while
return x
```

4. Se consideră un tablou $x[1..n]$ şi se doreşte construirea unui tablou $m[1..n]$ care conţine pe poziţia i media aritmetică a elementelor din subtabloul $x[1..i]$ ($m[i] = (x[1] + \ldots + x[i])/i$).

(a)(5p) Propuneţi un algoritm de complexitate $\Theta(n^2)$ pentru construirea tabloului m. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python.

(b)(10p) Propuneţi un algoritm de complexitate $\Theta(n)$ pentru construirea tabloului m. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python.

5. Se consideră trei tablouri de numere întregi, $a[1..n]$, $b[1..n]$, $c[1..n]$. Se pune problema verificării dacă există cel puţin un element comun în cele trei tablouri. De exemplu tablourile $a = [3, 1, 5, 10]$, $b = [4, 2, 6, 1]$, $c = [5, 3, 1, 7]$ au un element comun, pe când $a = [3, 1, 5, 10]$, $b = [4, 2, 6, 8]$, $c = [15, 6, 1, 7]$ nu au nici un element comun.

(a)(5p) Propuneţi un algoritm de complexitate $O(n^3)$ care returnează `True` dacă cele trei tablouri conţin cel puţin un element comun şi `False` în caz contrar. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python.

(b)(5p) Propuneţi un algoritm de complexitate $O(n^2)$ care returnează `True` dacă cele trei tablouri conţin cel puţin un element comun şi `False` în caz contrar. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python.

(c)(5p) Presupunând că elementele tablourilor sunt din $\{1, 2, \ldots, m\}$ propuneţi un algoritm de complexitate $O(\max(m, n))$ care returnează `True` dacă cele trei tablouri conţin cel puţin un element comun şi `False` în caz contrar. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python. *Indicaţie.* este permisă utilizarea unei zone suplimentare de memorie de dimensiune $\mathcal{O}(n)$.

(d)(5p) Presupunând că toate cele trei tablouri sunt ordonate crescător propuneţi un algoritm de complexitate $O(n)$ care returnează `True` dacă cele trei tablouri conţin cel puţin un element comun şi `False` în caz contrar. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python. *Indicaţie.* Se poate folosi ideea de la tehnica interclasării.

6. Se consideră un tablou, $x[1..n]$, ordonat crescător, v o valoare de acelaşi tip ca elementele tabloului şi algoritmul `alg`.

`alg`($x[1..n]$,v)
$i \leftarrow n$
while $i \geq 1$ and $v < x[i]$ **do**
    $i \leftarrow i - 1$
end while
return $i + 1$

(a)(5p) Implementaţi algoritmul `alg` în Python şi stabiliţi ce returnează.

(b)(5p) Estimaţi numărul *mediu* de comparaţii efectuate (în ipoteza în care toate clasele de date de intrare au aceeaşi probabilitate de apariţie).