Tema 1: Rezolvarea algoritmică a problemelor. Descrierea algoritmilor în pseudocod. Verificarea corectitudinii algoritmilor. Stabilirea ordinului de complexitate.

Termen finalizare: 19.11.2021

Submiterea temelor: Se uploadează în Microsoft Teams o arhiva de tip *.zip având numele construit pe baza regulii:
NumeStudent_grupa_Tema1.zip (de exemplu `AdamEva_gr3_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_grupa_Tema1_Problema (de exemplu `AdamEva_gr3_Tema1_Pb1b.py`)

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

---

1. (15p) *Operaţii exacte cu numere raţionale.* Un număr raţional poate fi întotdeauna specificat ca o fracţie ireductibilă (raportul a două numere întregi prime între ele). Se pune problema definirii unor funcţii corespunzătoare operaţiilor aritmetice (adunare, scădere, înmulţire, împărţire) care să permite obţinerea unor rezultate exacte (atât operanzii cât şi rezultatele vor fi specificate ca perechi de valori care definesc o fracţie ireductibilă). De exemplu pentru operaţia 1/2+3/4, funcţia de adunare va prelua numărătorul şi numitorul fiecărei fracţii (adică patru numere întregi: 1,2,3,4) şi va returna numărătorul şi numitorul rezultatului (5 şi 4).

   (a) (2.5p) Descrieţi un algoritm care primeşte ca parametri numărătorul şi numitorul unei fracţii şi returnează valorile corespunzătoare fracţiei ireductibile echivalente (pentru parametrii 6 şi 9 se vor returna valorile 2 şi 3).

   (b) (2.5p) Descrieţi algoritmi corespunzători operaţiilor de adunare, scădere, înmulţire şi împărţire a fracţiilor. Fiecare algoritm va primi ca parametri de intrare numărătorul şi numitorul fiecărei fracţii şi va returna numărătorul şi numitorul rezultatului.

   (c) (5p) Descrieţi un algoritm de calcul a valorii unui polinom cu coeficienţi întregi pentru un argument număr raţional. Se consideră că polinomul este reprezentat prin tabloul coeficienţilor (un polinom $c_nX^n + c_{n-1}X^{n-1} + \ldots c_1X + c_0$ este reprezentat printr-un tablou $c[0..n]$). De exemplu pentru polinomul $2X^3 - X^2 + X + 2$ valoarea calculată în $1/2$ este $5/2$. Pentru toate operaţiile aritmetice se vor folosi funcţiile descrise la punctul (b). `Indicaţie.` Se poate folosi ca punct de pornire algoritmul de la Problema 12/Seminar 4.

   (d) (5p) Descrieţi un algoritm care determină mulţimea tuturor rădăcinilor raţionale ale unui polinom cu coeficienţi numere întregi. *Indicaţie:* rădăcinile raţionale ale unui polinom cu coeficienţi întregi au proprietatea că sunt de forma d_0/d_n unde d_0 este un divizor (pozitiv sau negativ) al termenului liber (c_0) iar d_n este un divizor al coeficientului termenului de grad maxim (c_n). Mulţimea divizorilor luaţi în considerare include pe 1 şi numărul însuşi. Nu toate numerele raţionale de forma descrisă sunt rădăcini (trebuie selectate doar cele pentru care valoarea polinomului este 0).

   Pentru toţi algoritmii descrişi mai sus se vor implementa funcţii Python corespunzătoare.

2. (15p) *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 (toate zerourile de la sfârşit şi primul 1 întâlnit) sunt lăsate nemodificate, iar toate cele care vor fi parcurse ulterior vor fi complementate).

   (a) (2.5p) 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) (2.5p) 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) (5p) 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.

3. (10p) *Aproximarea funcţiei logaritmice prin serii.* Funcţia *ln* poate fi aproximată folosind următoarele serii:

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

   (a) (2.5p) Identificaţi regula de calcul a termenului următor (T_{n+1}) din fiecare serie folosind valoarea termenului curent ($T_n = \frac{(-1)^{n+1}}{n}(x-1)^n$ respectiv $T_n = ((x-1)/x)^n/n$).

   (b) (2.5p) 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) (2.5p) Pentru unul dintre algoritmii propuşi la punctul (b) (la alegere) identificaţi un invariant şi demonstraţi corectitudinea algoritmului.

   (d) (2.5p) Descrieţi un algoritm pentru estimarea erorii de aproximare $\sum_{i=1}^{m}(f(i \cdot h) - ln(i \cdot h))^2$. Date de test: $m = 100$, $h = 5/m$.

4. (5p) Se consideră algoritmii `alg1` şi `alg2`. Pentru fiecare dintre cei doi algoritmi:

(a) (2.5p) Implementaţi fiecare algoritm în Python.

(b) (2.5p) 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 $2^{k-1} \leq n < 2^k$.

```
1: alg1(int a, b)
2: if a < b then
3:     a ↔ b
4: end if
5: c ← 0
6: d ← a
7: while d > b do
8:     c ← c + 1
9:     d ← d − b
10: end while
11: return c, d
```

```
1: alg2(int n)
2: i ← 1
3: x ← 0
4: while i <= n do
5:     i ← 2 * i
6:     x ← x + 1
7: end while
8: return x
```

5. (15p) Se consideră următoarea relaţie de recurenţă:

$$\begin{cases} x_0 & = & a \\ x_{k+1} & = & \frac{1}{3}\left(2x_k + \frac{a}{x_k^2}\right), \quad k > 0 \end{cases}$$

(a)(8p) Propuneţi un algoritm care, pentru o valoare dată a, afişează valorile x_0, x_1, ..., x_k până când este îndeplinită condiţia $|x_k - x_{k-1}| < \epsilon$. Descrieţi algoritmul în pseudocod, implementaţi-l în Python şi testaţi-l pentru $\epsilon = 0.00001$ şi $a = 8$, $a = -27$, $a = 64$.

(b)(2p) Ce returnează algoritmul pentru o valoare a?

(c) (5p) Propuneţi o proprietate invariantă cu ajutorul căreia să se demonstreze că algoritmul propus returnează valoarea specificată la punctul (b).

6. (10p) 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) (5p) Propuneţi un algoritm de complexitate $\Theta(n)$ pentru construirea tabloului m. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python.

Observaţie: pentru stabilirea ordinului de complexitate se parcurg următoarele etape: (i) se stabileşte dimensiunea problemei; (ii) se identifică operaţia dominantă; (iii) se estimează numărul de execuţii ale operaţiei dominante; (iv) se stabileşte ordinul de complexitate.

7. (10p) Se consideră un tablou, $x[1..n]$, ordonat crescător, v o valoare de acelaşi tip ca elementele tabloului şi algoritmul `alg`.

(a) (2.5p) Implementaţi algoritmul `alg` în Python şi stabiliţi ce returnează.

```
1: alg(x[1..n],v)
2: i ← n
3: while i ≥ 1  and  v < x[i] do
4:     i ← i − 1
5: end while
6: return  i + 1
```

(b) (5p) Identificaţi un invariant pentru prelucrarea repetitivă şi demonstraţi că algoritmul este corect (inclusiv finit).

(c) (2.5p) Estimaţi numărul *mediu* de comparaţii efectuate (în ipoteza în care toate clasele de date de intrare au aceeaşi probabilitate de apariţie).

8. (10p) Se consideră următoarea tehnică de compresie a unui şir de litere: "dacă o literă este precedată şi urmată de o altă literă atunci ea este transferată ca atare în textul comprimat; dacă o literă apare pe mai multe poziţii consecutive atunci în textul comprimat subsecvenţa care conţine aceeaşi litera va fi înlocuită cu o pereche formată din litera respectivă şi numărul de repetări". De exemplu, textul "urrrraa" se va înlocui cu "ur4a2".

   (a) (5p) Propuneţi câte un algoritm şi implementaţi în Python funcţia corespunzătoare pentru: (i) compresia unui şir de litere; (ii) decompresia pornind de la un şir "comprimat";

   (b) (5p) Stabiliţi ordinul de complexitate pentru algoritmii descrişi la pct. (a).

9. (10p) Implementaţi în Python un generator aleator de adrese IP de forma "a.b.c.d" unde fiecare componentă este un număr natural cuprins între 0 şi 255. *Indicaţie:* se pot folosi funcţiile din modulul (pachetul) `random` din Python. Stabiliţi ordinul de complexitate al algoritmului propus.