laborator 3

>> Structuri liniare II

PROBLEME

1. **(2p)** Să se creeze o listă ordonată folosind algoritmul de sortare prin inserţie. (Implementaţi o funcţie `insertionSort`, care primeşte ca parametru de intrare capul unei liste simplu înlănţuite şi rearanjează nodurile din listă astfel încât acestea sunt ordonate crescător după câmpul `info`.)

2. **(1p)** Implementaţi următoarele funcţii pentru o listă simplu înlănţuită în care se reţin numere întregi:

   a. `countPositiveNumbers` care primeşte ca parametru de intrare capul unei liste simplu înlănţuite şi returnează numărul de noduri din listă al căror câmp `info` conţine un număr pozitiv.

   b. `computeArithmeticMeanOfNegativeNumbers` care primeşte ca parametru de intrare capul unei liste simplu înlănţuite şi returnează media artimetică a elementelor negative. (Prin element negativ se înţelege câmpul `info` care conţine un număr negativ.)

3. **(2p)** Implementaţi o funcţie `insertArithmeticMean`, care inserează între fiecare pereche de două noduri consecutive din listă un nod al cărui câmp `info` conţine media lor aritmetică. (Între un nod `x` şi nodul `x->next` ce îl urmează, se inserează un nod `nou`, pentru care avem `nou->info = (x->info + (x->next)->info)/2;`.)

4. **(1p)** Implementaţi o funcţie `deleteAt`, care primeşte ca parametru un număr întreg `k` şi capul unei liste simplu înlănţuite şi şterge al `k`-lea nod din listă.

5. **(1p)** Să se implementeze cu ajutorul unei liste liniare, simplu înlănţuite şi alocate dinamic, un polinom de grad n. Fiecare nod se va considera că reţine gradul fiecărui monom, precum şi coeficientul său. Structura poate fi definită astfel :

```
struct pol {
  int gr, coef;
  pol *next;
};
```

   Scrieţi un program care creează un polinom implementat prin liste şi calculează şi afişează coeficienţii polinomului obţinut prin înmulţirea cu un scalar `a`, citit de la tastatură.

6. **(4p)** Implementaţi o funcţie `reverse` care primeşte ca parametru capul unei liste simplu înlănţuite L_1 şi inversează ordinea elementelor din listă

   a. Varianta I (fără a modifica lista L_1): se creează o nouă listă L_2 care reţine rezultatul inversării, iar funcţia `reverse_1` returnează capul listei L_2.

   b. Varianta II (prin modificarea legăturilor din lista L_1): nu se creează o listă nouă şi se operează doar asupra listei L_1.

7. **(3p)** Implementaţi o funcţie `combine` care primeşte ca parametru capul unei liste simplu înlănţuite L_1 sortată crescător şi capul unei liste simplu înlănţuite L_2 sortată crescător şi returnează capul unei liste simplu înlănţuite L_3, care conţine elementele din L_1 şi L_2 în ordine crescătoare. Nu se va folosi memorie suplimentară (se alocă spaţiu în memorie doar pentru a reţine capul listei L_3).

8. **(3p)** Implementaţi o funcţie `split` care primeşte ca parametru trei liste simplu înlănţuite L_3, L_1, L_2, unde $L_1 = \emptyset$ $L_2 = \emptyset$. Funcţia distribuie elementele din L_3 în listele L_1 şi L_2 astfel: L_1 va conţine elementele de pe poziţiile impare din L_3 şi L_2 va conţine elementele de pe poziţiile pare din L_3. Nu se va folosi memorie suplimentară.

9. **(4p)** Spunem că o matrice `X` de dimensiuni `n`×`m` (`n` linii şi `m` coloane) este **rară**, dacă există „foarte multe" elemente egale cu 0. Pentru a economisi memoria, putem reprezenta o astfel de matrice prin liste simplu înlănţuite. Pentru fiecare linie `i` nenulă, va fi creată o listă care conţine doar elemente nenule, în care fiecare nod are trei câmpuri:

   - `column`: indicele coloanei (`j`)
   - `info`: valoarea elementului nenul (x_{ij})
   - `next`: legătura către următorul element nenul de pe linie

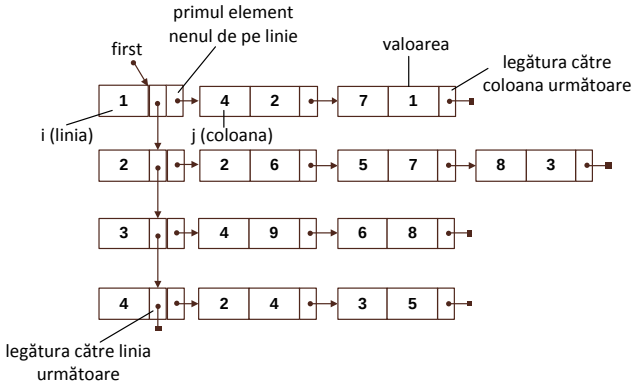
   Pentru că trebuie să se reţină şi care este primul element nenul dintr-o linie, se foloseşte şi o listă suplimentară în care fiecare nod conţine următoarele informaţtii:

   - `line`: indicele `i` al unei linii nenule
   - `next`: legătura către următoarea linie nenulă
   - `first`: legătura către primul element nenul din lista corespunzătoare liniei `i`.

   Spre exemplu, matricea 4×8

$$\begin{pmatrix} 0 & 0 & 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 6 & 0 & 0 & 7 & 0 & 0 & 3 \\ 0 & 0 & 0 & 9 & 0 & 8 & 0 & 0 \\ 0 & 4 & 5 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

   va fi reprezentată ca:



   Folosind această reprezentare pentru matrice rare, să se scrie un program care citeşte două matrice, le reprezintă ca mai sus şi face suma lor, reprezentată tot ca matrice rară.

10. **(10ps)** Fie `a` un vector de `n` componente întregi, neordonate. Spunem că un element `x` este majoritar în `a`, dacă apare de cel puţin $\left\lfloor \frac{n}{2} \right\rfloor + 1$ ori în `a`. Descrieţi şi implementaţi un algoritm ce rulează în timp $O(n)$ care să decidă dacă există un element majoritar, şi dacă da, să îl afişeze.

■ TERMEN DE PREDARE: **Săptămâna 5 (29 octombrie-2 noiembrie) inclusiv.**

■ DETALII: **Studenţii pot obţine un maxim de 21 puncte. Problemea 1 este obligatorie. Problemele 2–9 sunt suplimentare. Problema 10 este facultativă, iar termenul de predare pentru ea este săptămâna 4 (22-26 octombrie). Un singur student poate rezolva problema facultativă.**