laborator 5

>> Structuri liniare IV – Cazuri particulare

Conţinut

- Liste dublu înlănţuite
- Operaţii specifice: traversare, căutare, inserare, ştergere
- Liste cu nod marcaj
- Liste circulare
- DEQUE (Double Ended Queue)
- Coadă cu priorităţi

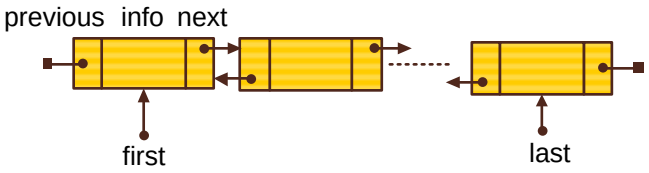
Referinţe

- **T.H. Cormen, C.E. Leiserson, R.L. Rivest.** *Introducere în algoritmi: cap 7.5 şi 11.2*, Editura Computer Libris Agora, 2000 (şi ediţiile ulterioare)
- **R. Ceterchi.** *Materiale de curs*, Anul universitar 2012-2013
- **http://laborator.wikispaces.com/**, *Tema 5*

1. Liste dublu înlănţuite

Listele dublu înlănţuite permit, spre deosebire de listele simplu înlănţuite, parcurgerea în ambele sensuri a elementelor din listă: atât parcurgerea stânga-dreapta (de la primul spre ultimul nod), cât şi parcurgerea inversă, dreapta-stânga (de la ultimul spre primul nod).

Aceasta înseamnă că vom reţine atât *capul* listei, adică primul element, ce a fost notat `first`, cât şi *coada* listei, adică ultimul element, pe care îl notăm `last`.



Observăm că nodurile unei liste dublu înlănţuite conţin un câmp adiţional, numit `previous`. Acesta este necesar pentru a păstra legătura către nodul anterior, în acelaşi mod cum folosim câmpul `next` pentru a reţine adresa nodului următor.

În C++ puteţi reţune un nod folosind structuri:

```
struct nod {
  int info;
  nod *next;
  nod *previous;
};
```

Obs. Putem afla dacă un element este ultimul din listă, verificând dacă pe câmpul său `next` reţine **`NIL`** (la fel ca la liste simplu înlănţuite. În plus, un element este primul din listă, dacă pe câmpul său `previous` reţine **`NIL`** .

Obs. Toate operaţiile efectuate pe liste simplu înlănţuite se pot efectua şi pe liste în dublă înlănţuire. În continuare sunt adăugate doar operaţiile specifice celor din urmă.

A. Operaţia de traversare

Am precizat că putem parcurge structura de listă dublu înlănţuită în ambele sensuri. Pentru parcurgerea de la primul la ultimul element (stânga–dreapta), revedeţi algoritmul de traversare pentru liste simplu înlănţuite.

Pentru parcurgerea în sens invers, dreapta–stânga, întâi se accesează ultimul nod, `last`. Apoi, din fiecare nod curent, numit `curent`, se accesează nodul anterior cu ajutorul adresei reţinute de `curent->previous`.

La pasul 1, pointerul pentru traversare (`curent`) este iniţializat, preluând adresa ultimului element din listă. Apoi, cât timp mai sunt elemente în structură (mai există adrese de parcurs), se vizitează nodul curent şi se avansează către nodul anterior (paşii 2-5). Testul de nedepăşire a structurii este realizat prin condiţia testată în ciclul **`while`** : un câmp `previous` care este **`NIL`** indică sfârşitul listei.

►► Traversează–Invers–Listă–Dublu–Înlănţuită(first, last)

```
1. curent ← last
2. while curent ≠ NIL do
3.   Vizitează(curent)
4.   curent ← curent->previous
5. endwhile
```

B. Operaţia de căutare

Următoarul algoritm caută în lista indicată de `first` şi `last` valoarea `val` folosind traversarea în sens invers. Dacă găseşte un nod, al cărui câmp `info` conţine `val`, atunci returnează în variabila `loc` adresa acestui nod. Pointerul `loc` are proprietatea că `loc->info=val`. Dacă un asemenea nod nu se găseşte, atunci `loc` va fi **`NIL`** .

►► Caută–Invers–În–Listă–Dublu–Înlănţuită(first, val, loc)

```
1.  loc ← last
2.  while (loc ≠ NIL ) and (loc->info ≠ val) do
3.    loc ← loc->previous
5.  endwhile
6.  if loc ≠ NIL then
7.    Căutarea s-a terminat cu succes.
8.  else
9.    Căutarea s-a terminat fără succes.
10. endif
```

C. Operaţia de inserare

În momentul creării unui nod nou se alocă spaţiu în mod dinamic.
Următoarea secvenţă de instrucţiuni creează un nod nou care să conţină valoarea x:

```
nod *nou = new nod;
nou->info = x;
nou->next = 0; // null (NIL)
nou->previous = 0; // null (NIL)
```

Întâi s-a realizat alocarea de memorie, apoi s-a iniţializat câmpul cheie al nodului şi la sfârşit s-a atribuit valoarea **`NIL`** câmpurilor de legătură ale nodului.

Operaţia de inserare propriu-zisă într-o listă dublu înlănţuită este operaţia care conectează nodul nou creat de celelalte noduri din listă, într-un loc anume în listă, care trebuie determinat în funcţie de natura problemei. După ce s-a determinat locul inserării, legarea noului nod la listă se face prin realocarea a **patru** pointeri (sau prin schimbul a **patru** legături).

Ca şi la listele simplu înlănţuite, următoarea secvenţă de instrucţiuni inserează un nod `nou` în capul listei:

```
nou->next = first;
first = next;
```

Determinarea locului inserării în listă se face printr-o traversare eventual incompletă a listei cu un pointer `curent`, care poate porni fie de la primul, fie de la ultimul element. Traversarea va depinde de două condiţii:

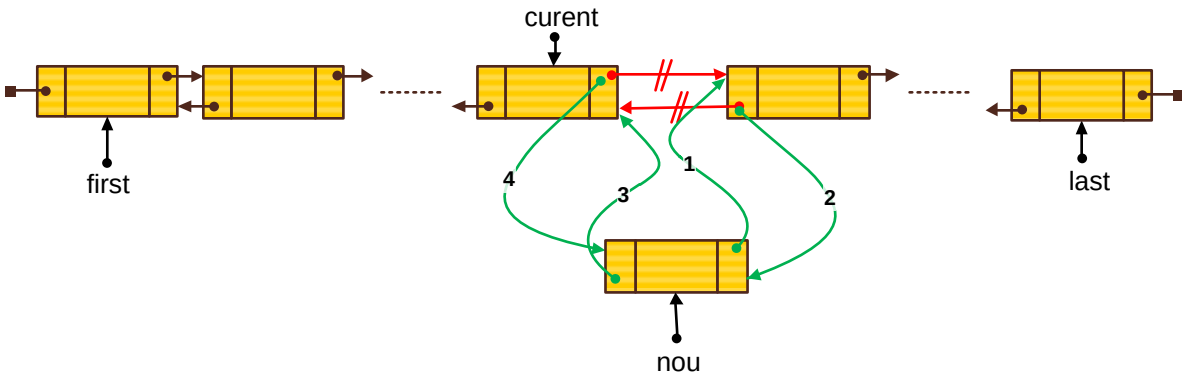
- o condiţie de nedepăşire a structurii, şi
- o condiţie referitoare la locul în care se face inserarea.

În funcţie de a doua condiţie, traversarea se poate termina într-una din următoarele situaţii:

A. pe un nod `curent` după care urmează să fie inserat nodul `nou` cu următoarea secvenţă de instrucţiuni:

```
nou->next = curent->next;                    (1)
(curent->next)->previous = nou;              (2)
nou->previous = curent;                      (3)
curent->next = nou;                          (4)
```

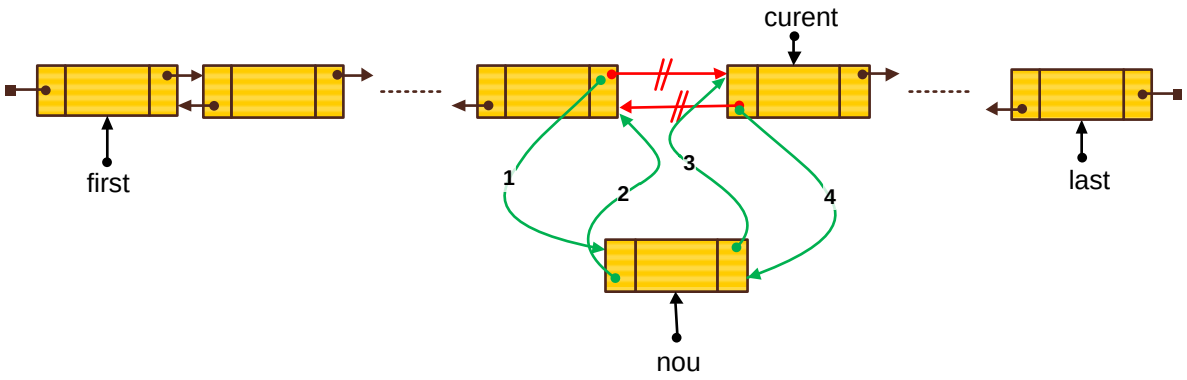
Acestea refac legăturile în ambele sensuri la stânga şi la dreapta.



B. pe un nod `curent` înaintea căruia urmează să fie inserat nodul `nou`. Observaţi că **NU** mai este nevoie pointerul adiţional `anterior` folosit în situaţia aceasta la liste simplu înlănţuite. Folosim următoarea secvenţă de instrucţiuni:

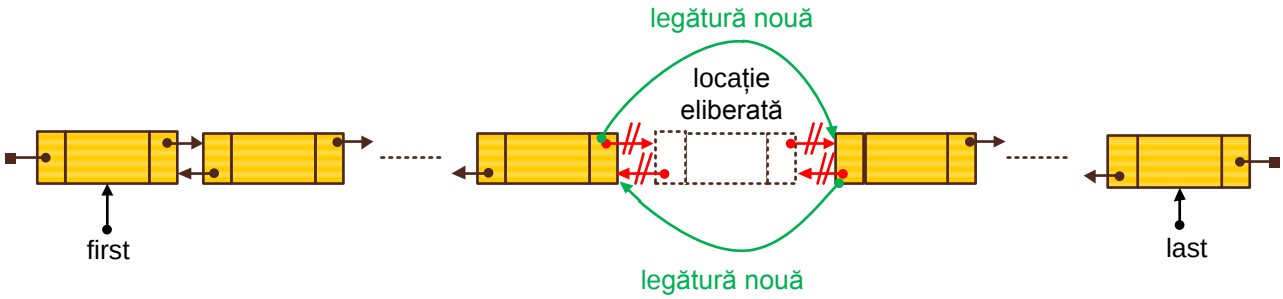
```
(curent->previous)->next = nou;              (1)
nou->previous = curent->previous;            (2)
curent->previous = nou;                      (3)
nou->next = curent;                          (4)
```

Acestea refac legăturile în ambele sensuri la stânga şi la dreapta.



D. Operaţia de ştergere/extragere

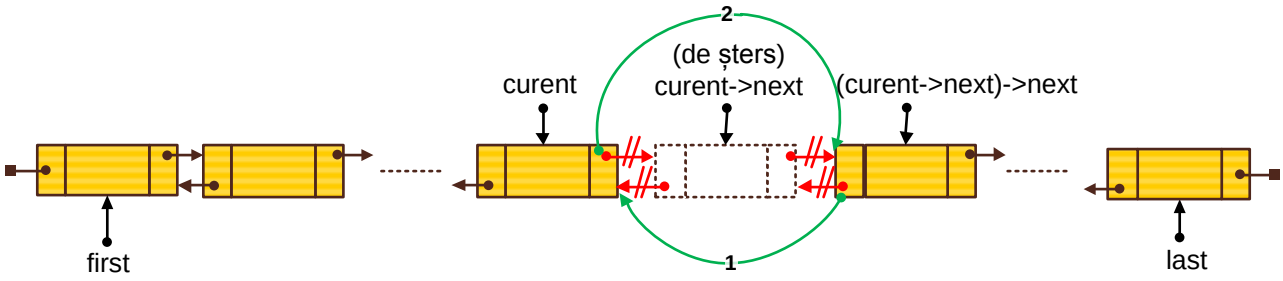
După efectuarea ştergerii sau extragerii nodului propriu-zise, operaţia de ştergere trebuie să prevadă şi refacerea structurii de listă dublu înlănţuită pe nodurile rămase. Pentru nodul extras se poate dealoca spaţiul sau efectua alte prelucrări.



Similar inserării, poziţia nodului ce trebuie şters se găseşte în urma unei traversări eventual incomplete a listei (în oricare sens), folosind un pointer `curent`. Traversarea este de asemenea guvernată de două condiţii: nedepăşirea structurii şi o condiţie specifică problemei. A doua condiţie conduce la terminarea traversării într-una din următoarele două situaţii:

A. pe un nod `curent` după care urmează nodul ce trebuie şters (se va şterge `curent->next`). Ştergerea se face prin cele două instrucţiuni:

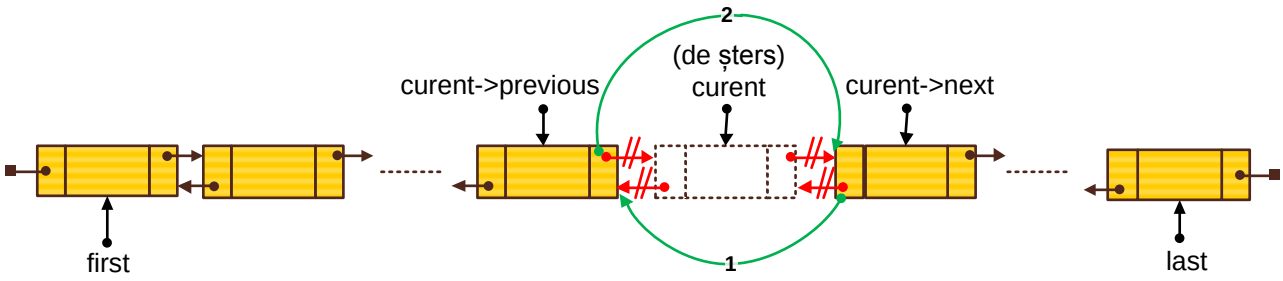
```
((curent->next)->next)->previous = curent;          (1)
curent->next = (curent->next)->next;                (2)
```



Atenţie la cazurile extreme, când pointerii referiţi sunt **`NIL`** .

B. pe un nod `curent` care este chiar nodul ce trebuie şters. Din nou, spre deosebire de listele simplu înlănţuite nu mai avem nevoie de pointerul adiţional `anterior`, iar ştergerea se realizează prin instrucţiunile:

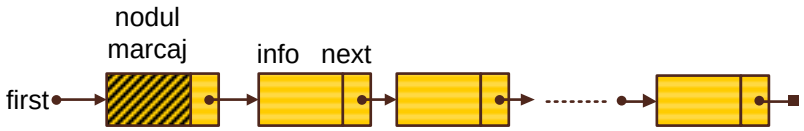
```
(curent->next)->previous = curent->previous;        (1)
(curent->previous)->next = curent->next;            (2)
```



2. Liste cu nod marcaj (santinelă)

Pentru nodul marcaj se mai foloseşte şi denumirea de santinelă *(sentinel).*

O listă cu nod marcaj are structura din diagrama următoare. Practic, primul element al listei (a cărui referinţuă este `first`, este nodul marcaj, iar elementele propriu-zise ale listei încep de la `first->next`.



Puteţi considera că nodul marcaj are câmpul său `info` setat cu o valoare nedefinită.

Dacă lista cu nod marcaj este vidă atunci ea conţine doar nodul marcaj, ca mai jos.

Nodul marcaj este introdus pentru a simplifica operaţiile de inserare şi de ştergere în cazurile extreme. Lista de mai sus reprezintă o listă simplu înlănţuită care are nodul marcaj pe primul element, prin urmare nu mai trebuie să verificăm niciodată dacă `first` este **`NIL`** , deoarece chiar şi dacă lista este vidă, ea va conţine nodul marcaj. Din aceleaşi considerente nu va trebui să mai verificăm dacă pointerul `anterior` (care se afla mereu la poziţia dinaintea pointerului de traversare `curent`) este **`NIL`** .

3. Liste circulare

În laboratorul 4, am discutat cazul particular al unei cozi circulare, în alocare statică. Introducem acum un caz mai general al *listelor circulare*, care sunt liste în care după ultimul element, urmează iarăşi primul.

Obs.: Astfel de liste (circulare) sunt utile atunci când trebuie să facem parcurgeri repetate ale listei.

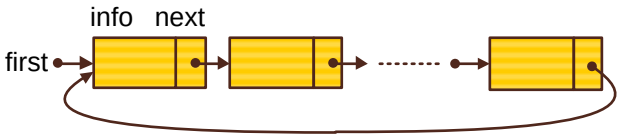
În diagrama următoare aveţi exemplul unei liste circulare (simplu înlănţuite). În cazul acestora legătura suplimentară – de la ultimul la primul element – se poate realiza având grijă ca pe câmpul `next` al ultimului nod din listă să se reţină `first`. Modificarea apare practic doar în cazul inserărilor la sfârşitul listei, care devin inserări înaintea primului element. Pentru un plus de siguranţă la iniţializarea unui nod nou, numit `nou`, (de inserat) în loc de a seta

```
nou->next = 0; //null
```

putem iniţializa astfel:

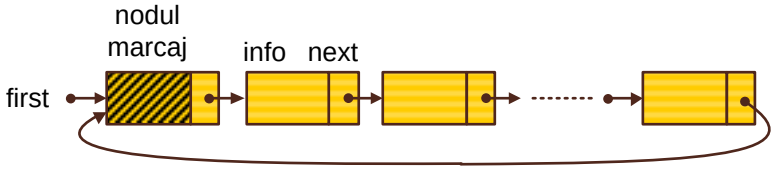
```
nou->next = first;
```

Atenţie şi la faptul că testul de nedepăşire a structurii nu se mai poate realiza prin comparaţie cu **`NIL`** , deoarece acesta nu va mai fi niciodată întâlnit şi am crea un ciclu infinit. Pentru o singură parcurgere se poate avansa cu un pointer `curent` (ce porneşte cu valoarea iniţială `first`) cât timp următorul său element nu este `first` (`curent->next != first`) într-un ciclu **`do`** –**`while`** .

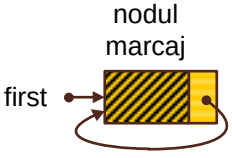


În **T.H. Cormen, C.E. Leiserson, R.L. Rivest.** *Introducere în algoritmi* se remarcă faptul că o astfel de listă poate fi privită ca un **inel de elemente**.

Structura unei **liste circulare cu nod marcaj** este dată în diagrama următoare.



Ca şi mai sus, o listă circulară cu nod marcaj care este vidă arată ca în desenul următor.



4. DEQUE (Double Ended Queue)

DEQUE (Double Ended Queue) desemnează o structură liniară în care inserările şi ştergerile se pot face la oricare din cele două capete, dar în niciun alt loc din coadă.

Pentru o astfel de structură, implementată folosind liste simplu înlănţuite (ca în laboratorul 4), operaţiile `push` şi `pop` pentru cozi simple asigură inserarea în spate şi ştergerea în faţă, deci ar trebui adăugate:

- o operaţie `push_front` pentru inserarea în faţă (adică inserarea ca la stive), şi
- o operaţie `pop_back` pentru ştergerea în spate.

Algoritmii pentru cele două noi operaţii sunt daţi în continuare.

►► PUSH_FRONT(front, rear, val)

```
1.  creează nod nou
2.  if nou ≠ NIL then
3.    nou->info ← val
4.    nou->next ← NIL
5.    front ← nou
6.    if rear = NIL then
7.      rear ← nou 8.  else
9.   overflow
10.  endif
```

Observaţi că singura modificare faţă de algoritmul de inserare pentru stive constă în liniile 6–7, care asigură că `rear` va pointa şi el către `nou`, dacă `nou` este primul element inserat.

►► POP_BACK(front, rear, x)

```
1.  if front ≠ NIL then
2.    beforerear ← front
3.    if beforerear = rear then
4.      beforerear = NIL
5.    else
6.      while beforerear->next ≠ rear do
7.        beforerear ← beforerear->next
8.      endwhile
9.    endif
10.    x ← rear->info
11.    temp ← rear
12.    rear ← beforerear
13.    distruge temp
14.    if rear = NIL then
15.      front ← NIL
16.    endif
17.  else
18.    underflow
19.  endif
```

În liniile 2–9 se obţine în variabila `beforerear` viitoarea valoare a lui `rear`. Aceasta va fi fie un pointer către penultimul element din coadă (cel anterior lui `rear`), fie **`NIL`** în situaţia în care coada conţinea un singur element. Astfel, după ştergerea ultimului element (`rear`), putem să actualizăm conform liniei 8 (noul ultim element va fi fostul penultim sau **`NIL`** – dacă structura este acum vidă). Pentru a evita parcurgerea din liniile 6–8, putem utiliza fie o **implementare cu liste circulare, fie cu liste dublu înlănţuite pentru DEQUE**.

Structura DEQUE este utilă când inserările şi ştergerile se pot face la ambele capete, dar puteţi întâlni şi situaţii în care să fie necesare cozi unde inserarea se poate realiza la un singur capăt şi extragerile la amândouă capetele, şi reciproc, extragerea să se poată realiza la un singur capăt, iar inserările la ambele.

5. Cozi cu/de priorităţi

Coada cu priorităţi este o coadă în care elementele au, pe lângă cheie şi o prioritate/o pondere (greutate/importanţă). Vom presupune că cea mai înaltă prioritate este 1, urmată de 2, şi aşa mai departe.

Aceasta înseamnă că dacă structura de coadă cu priorităţi va fi în alocare dinamică, fiecare nod va conţine pe lângă cheie (câmpul `info`) şi un câmp de tip numeric `weight` pentru prioritate.

Ordinea liniară este dată de regulile:

- elementele cu aceeaşi prioritate sunt extrase (şi procesate) în ordinea intrării;
- toate elementele cu prioritate i se află înaintea celor cu prioritate $i+1$ (şi deci vor fi extrase înaintea lor).

Extragerile se fac dintr-un singur capăt. Ca să se poată aplica regulile de mai sus la extragere, inserarea unui nou element cu prioritate i se va face la sfârşitul listei ce conţine toate elementele cu prioritate i.

PROBLEME

1. **(2p)** Să se implementeze o listă liniară dublu înlănţuită în care se vor reţine numere întregi. Scrieţi funcţii pentru:
   - **a.** Adăugarea unui element la început;
   - **b.** Adăugarea unui element la sfârşit;
   - **c.** Adăugarea unui element în interiorul listei;
   - **d.** Afişarea elementelor listei în ordinea introducerii lor;
   - **e.** Afişarea elementelor listei în ordine inversă;
   - **f.** Ştergerea unui element din listă ştiind numărul lui de ordine;
   - **g.** Ştergerea unui element din listă ştiind valoarea lui.
2. **(2p)** Considerând structura DEQUE implementată cu ajutorul unei liste liniare cu dublă înlănţuire, să se scrie procedurile de inserare şi ştergere la ambele capete ale ei; să se utilizeze aceste proceduri într-un program care afişază un meniu în vederea selectării procedurii dorite din cele patru posibile.
3. **(2p)** Să se scrie procedurile de punere şi scoatere a unui element într-o, respectiv, dintr-o coadă cu priorităţi reprezentată cu ajutorul unei liste simplu înlănţuite.
4. **(2p)** Să se scrie procedurile de inserare, respectiv, ştergere a unui nod cu o cheie dată într-o, respectiv, dintr-o listă circulară cu dublă înlănţuire şi nod marcaj.
5. **(2p)** Să se scrie procedurile de inserare şi ştergere nod într-o, respectiv, dintr-o listă circulară cu dublă înlănţuire şi nod marcaj, care implementează următoarea strategie: se inserează la dreapta nodului marcaj şi se şterge de la stânga sa; cum se poate interpreta această modalitate de modificare a listei?
6. **(10ps)** Fie w un şir de caractere. Spunem că un subşir u al lui w este o *margine (border)* dacă w începe şi se termină cu u, şi în plus u este diferit de w. **Exemple:**
   (a) pentru $w =' \texttt{aba}'$, $'\texttt{a}'$ este o margine;
   (b) pentru $w =' \texttt{ababa}'$ avem marginile $'\texttt{a}'$, $'\texttt{aba}'$.

   Spunem că cea mai lungă margine a lui w se numeşte *marginea* lui w. Să se găsească un algoritm liniar care determină marginea lui w.

■ Termen de predare: **Săptămâna 8 (19–23 noiembrie) inclusiv.**

■ Detalii: **Studenţii pot obţine un maxim de 20 puncte. Problema 1 este obligatorie. Problemele 2–5 sunt suplimentare.Problema 6 este facultativă, iar termenul de predare pentru ea este săptămâna 7 (12–16 noiembrie). Un singur student poate rezolva problema facultativă.**