EXAMEN STRUCTURI DE DATE

Subiectul 4

1 Se consideră graful neorientat **G**={**V, E**} cu **V**={1,2,3,4} şi **E**={(1,3), (1,4), (2,3), (2,4)}.
Se cere:

0,2p a) Dacă **V**={1,2} ∪ {3,4} ce fel de graf este **G** ?

0,2p b) Reprezentaţi printr-un desen modul de memorare al grafului folosind listele de adiacenţă.

0,4p c) Ce cantitate de memorie este necesară pentru stocarea în acest fel a lui **G** ?

0,2p d) Care este matricea de adiacenţă a grefului **G**?

2 Să se arate că dintre toţi arborii **k**-ari, cei binari (**k**=2) au cel mai puţin spaţiu de memorie alocat
2p pentru legături nule.

3 Structura de date coada:

0,2p • definiţie,
0,2p • exemplu,
0,6p • declaratie pentru interfaţa (clasa **Queue**) folosind Templates,
1,0p • declaratia clasei de implementare sub formă de listă simplu înlănţuită (**QueueAsLinkedList**)

4 Sortarea topologică:

0,4p • exemplu,
1,0p • descrierea algoritmului,
0,6p • ordin de operaţii (calcul expresie)

5 Să se scrie o metodă:
1p

```
ostream&  Matrix::printSpiral(ostream& ostr) const;
```

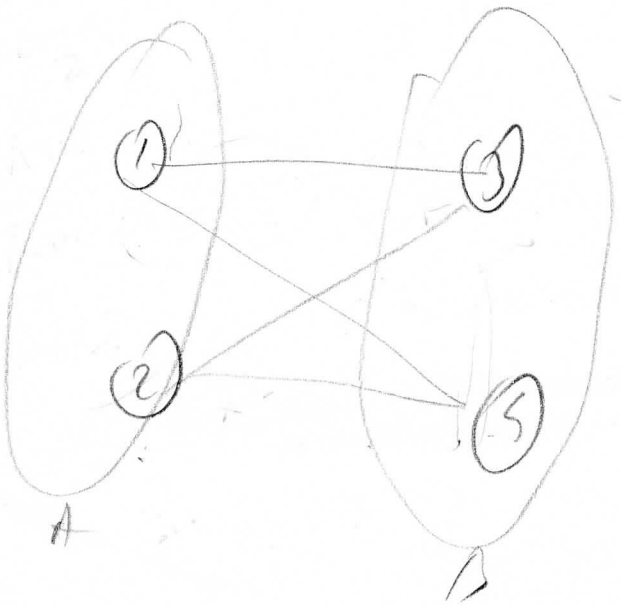
care afişează elementele unei matrici, parcurgând-o în spirală din exterior spre interior.

6 Pentru un arbore binar, scrieţi o metodă:
1p

```
int BTree::getRightEmpty()const;
```

care determină numarul total de noduri care au fiul drept vid.

TOTAL: 9p+1p oficiu=10p



Subiectul 4.

(2) se numește cei $n(k-1)+1$ din cămpuri sunt cu legături multe.

Dar $n(k-1)+1 > \frac{k-1}{k} \geq \frac{1}{2}$ cu ~~egalitate~~ dacă $k=2$

adică arborii binari

(3) Coada este o listă ordonată în care toate inserările se fac pe la un capăt numit spate și ieșirile sunt efectuate la celălalt capăt numit CAP.

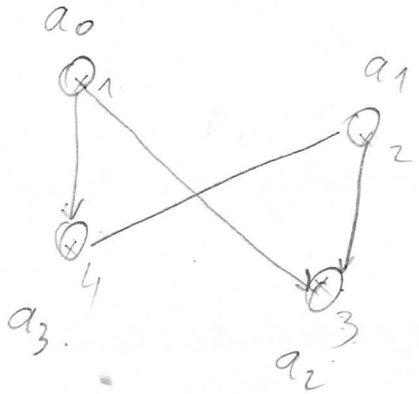
Exemplu:

- gestionarea fluxului de mașini dintr-o stație de spălătorie auto.
- utilizarea structurii de tip coadă în problema labirintului. Elementul introdus în coadă este
  - pătratul sursă
  - pătratul curent

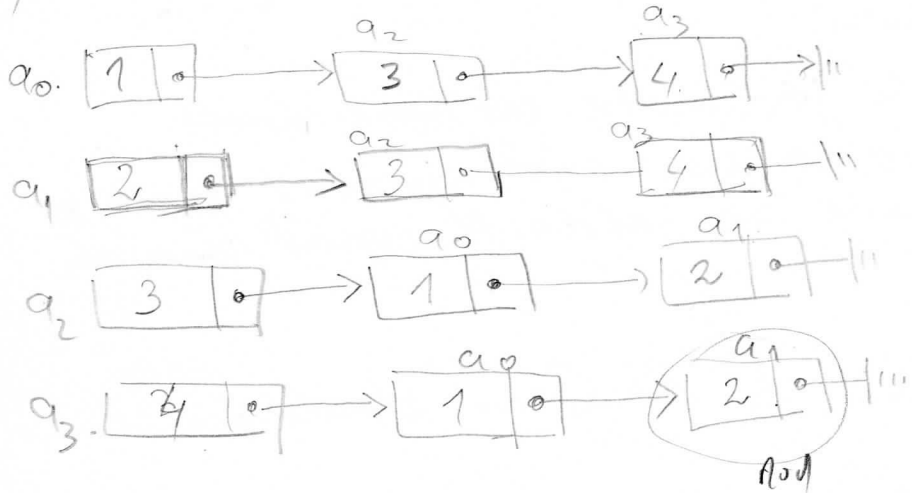
interfața este

```
#include "container.h"
#include "error.h"
template <class T>
class Queue: public virtual Container
{
public:
virtual T& getHead() const =0;
virtual void enQueue(T const&) =0;
virtual T& deQueue() =0; // coadă cu doi terminați adică
};                          o coadă în care elem. pot fi adăugate sau
                            șterse la oricare din capete.
```

Pentru coada ca o listă înlănțuită avem programul din computer lista-pa.

① 

a)



listele de adiacențe.

Nod

$n \times sizeof(nod)$

d)

$$M(4) = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Obs. De cele mai multe ori $M(n)$ este o matrice rară

⑤ Scad

⑥

④ Sortarea topologică

Deoarece există și mulțimi care nu sunt total ordonate (mț. părților, $\subseteq$) se pune problema fie ordonată parțial.

Exemplu

Fie $S = \{S_1, S_2, \dots, S_n\}$ o mț. de soldați.

între care se cunosc câteva relații de subordonare de tipul

$S_{i_{K_1}} < S_{i_{K_2}} \Leftrightarrow$ $S_{i_{K_1}}$ este subordonat lui $S_{i_{K_2}}$.

Se cere să se găsească o secvență sortată cu $<$

$S_{sort} = \{ S_{p_1}, S_{p_2}, \dots S_{p_n} \}$.

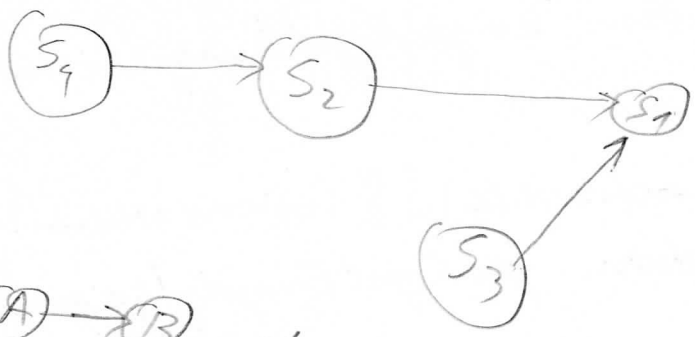
Mai concret fie $S = \{ S_1, S_2, S_3, S_4 \}$ cu relațiile următoare

$S_2 < S_1$

$S_3 < S_1$

$S_4 < S_2$

Diagrama de subordonare (graful) este.



cu

dacă și numai dacă $A < B$.

Secvențele posibile sortate din punct de vedere topologic sunt

$\delta_1 = \{ S_3, S_4, S_2, S_1 \}$

$\delta_2 = \{ S_4, S_2, S_3, S_1 \}$.

$\delta_3 = \{ S_4, S_3, S_2, S_1 \}$

Algoritmul care generează aceste sortări folosește un vector Vect care memorează numărul de arce ce intră în fiecare nod (adică nr. de predecesori). În cazul în care se găsește o valoare nulă

(nu interesă niciun arc) atunci nodul respectiv este primul. Se procedează analog cu toate nodurile în care se poate ajunge imediat dim nodul curent.

Procedeul se reia cu toate nodurile nemarcate.

⑥
```
int Btree::getRightEmpty(Btree *bt)
{
if(bt!=NULL)
  {
    if(bt->st!=NULL)&&(bt->dr==NULL)
      return 1+getRightEmpty(bt->st)
    else
      {if bt->st==NULL)&&(bt->dr==NULL)
        return getRightEmpty(bt->dr);
       else
        return getRightEmpty(bt->st)+
               getRightEmpty(bt->dr)
      }
  }
}
```