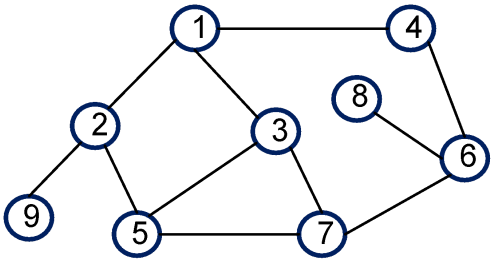
Bibliografie

1. J. Kleinberg, É. Tardos, **Algorithm Design**, Addison-Wesley 2005 http://www.cs.princeton.edu/~wayne/kleinberg-tardos/
2. T.H. Cormen, C.E. Leiserson, R.L. Rivest – Introduction to algorithms, MIT Press, Cambridge, 1990/2001
3. G.A. Giumale – Introducere în analiza algoritmilor, Polirom, 2004
4. H. Georgescu – Tehnici de programare, Editura Universității din București, 2005
5. J.A. Bondy, U.S.R Murty – Graph theory with applications, The Macmillan Press 1976, Springer 2008
6. http://www.cplusplus.com (http://www.cplusplus.com/reference/stl/)

Modalităţi de reprezentare a grafurilor

- matrice de adiacenţă
- liste de adiacenţă
- listă de muchii

- **Exemplu – graf neorientat**



- matricea de adiacenţă

```
0 1 1 1 0 0 0 0 0
1 0 0 0 1 0 0 0 1
1 0 0 0 1 0 1 0 0
1 0 0 0 0 1 0 0 0
0 1 1 0 0 0 1 0 0
0 0 0 1 0 0 1 1 0
0 0 1 0 1 1 0 0 0
0 0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0 0
```

- liste de adiacenţă

```
1:  2, 3, 4
2:  1, 9, 5
3:  1, 5, 7
4:  1, 6
5:  2, 3, 7
6:  4, 8, 7
7:  3, 6, 5
8:  6
9:  2
```

- listă de muchii

```
1 2
1 3
1 4
2 5
2 9
3 5
3 7
5 7
6 7
6 8
4 6
```

- **Exerciţii:**

1. Propuneți modalități de reprezentare şi pentru grafuri orientate şi pentru multigrafuri neorientate/orientate (care admit muchii/arce multiple şi bucle)
2. Precizați, în fiecare caz, pentru fiecare modalitate de reprezentare cum se pot determina numărul de muchii ale grafului, gradul fiecărui vârf (gradul interior şi exterior al fiecărui vârf în cazul orientat).

Parcurgerea grafurilor

Scop: Dat un graf şi un vârf de start `s`, să se determine toate vârfurile accesibile din `s` (printr-un lanț/drum, după cum graful este neorientat/orientat).

Ideea: pornim din vârful de start `s`; dacă ştim că un vârf `i` este accesibil din `s` şi există muchie (arc în caz orientat) de la `i` la `j`, atunci şi `j` este accesibil din `s`.

▪ Parcurgea pe lățime (BF - Breadth First)

Parcurgerea pe lățime BF urmăreşte vizitarea vârfurilor **în ordinea crescătoare a distanțelor** lor față de `s` (distanța este dată de numărul de muchii).

Se porneşte din vârful de start `s`, se vizitează vecinii acestuia (vârfurile adiacente cu el), apoi vecinii nevizitați anterior ai acestora, procesul repetându-se până nu mai există vârf cu vecini nevizitați.

Algoritm: Algoritmul va folosi o coadă C în care sunt introduse vârfurile care sunt vizitate şi care urmează să fie explorate (în sensul că vor fi cercetați vecinii lor); operațiile de introducere în coadă şi eliminare din coadă sunt simbolizate prin $\Rightarrow$, respectiv $\Leftarrow$. Eventuala existență a ciclurilor conduce la necesitatea de a marca vârfurile vizitate.

```
for i=1,n
   vizitat(i) ← false
C ← ∅;

{se viziteaza varful de start si se introduce in coada}
s ⇒ C; vizitat(s) ← true

while C ≠ ∅
    {se extrage un varf din coada pentru a fi explorat: se marcheaza ca
vizitate varfurile vecine ale acestuia nevizitate anterior si se
introduc in coada}
     i ⇐ C; prelucreaza(i);
     for toți j vecini ai lui i
         if not vizitat(j) then j ⇒ C; vizitat(j)←true
```

Procedura de prelucrare a unui vârf poate consta, de exemplu, din afişarea informației din vârful respectiv.

Muchiile folosite de algoritm pentru a descoperi noi vârfuri accesibile din `s` formează un **arbore de rădăcină `s`.** Pentru a reține acest arbore putem folosi legături de tip tată (predecesor). Astfel, vom mai avea un vector `tata` care se inițializează cu 0. Atunci când vârful `j` este descoperit ca vecin nevizitat al lui `i` şi introdus în coadă vom atribui lui `tata(j)` valoarea `i` (`j` este fiu al lui `i` în arbore).

Pentru orice vârf `i` accesibil din `s` **lanţul/drumul determinat prin parcurgerea BF de la `s` la `i` (din arbore) este minim (ca număr de muchii).** Putem reconstitui un astfel de drum folosind legătura `tata`, mergând înapoi din `i` spre `s`.

- **Parcurgerea în adâncime (DF – Depth First)**

Pentru parcurgea în adâncime a componentei conexe a lui `s` se porneşte din acest vârf şi se trece mereu la primul dintre vecinii vârfului curent nevizitați anterior, dacă un astfel de vecin există. Dacă toți vecinii vârfului curent au fost deja vizitați se merge înapoi pe drumul de la `s` la vârful curent până se ajunge la un vârf care mai are vecini nevizitați; se trece la primul dintre aceştia şi se reia procedeul.

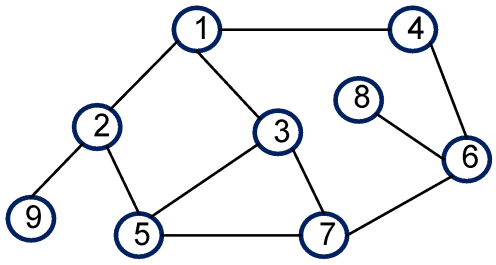
Procedura recursivă pentru parcurgerea în adâncime (DF) este

```
procedure DF(i)
   prelucreaza(i);
   vizitat(i) ←true
   for toți j vecini ai lui i
      if not vizitat(j) then DF(j)
```

Vectorul `vizitat` se inițializează cu `false`. Pentru determinarea componentei conexe a lui s procedura se apelează DF(s). Dacă dorim determinarea tuturor componentelor conexe se apelează procedura pentru fiecare vârf rămas nevizitat.

```
for i=1,n
   if not vizitat(i)then DF(i)
```

- **Exemplu**



BF: 1, 2, 3, 4, 5, 9, 7, 6, 8

DF: 1, 2, 5, 3, 7, 6, 4, 8, 9

✍ Probleme

Pentru toate problemele din cadrul laboratorului datele se vor citi din fişierul *graf.in*.
Dacă nu se precizează în enunţ, un graf este dat prin următoarele informaţii: numărul de vârfuri n, numărul de muchii m şi lista muchiilor (o muchie fiind dată prin extremităţile sale).

graf.in
9 11 1 2 1 3 1 4 2 5 2 9 3 5 3 7 5 7 6 7 6 8 4 6

A. Memorarea unui graf

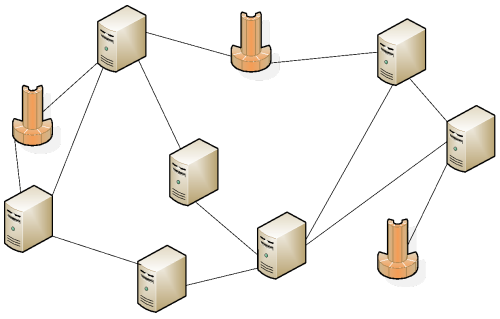
1. Scrieţi un subprogram pentru construirea în memorie a matricei de adiacenţă a unui graf neorientat citit din fişierul *graf.in* cu structura precizată mai sus şi un subprogram pentru afişarea matricei de adiacenţă

2. Scrieţi un subprogram pentru construirea în memorie a listelor de adiacenţă pentru un graf neorientat citit din fişierul *graf.in* cu structura precizată mai sus şi un subprogram pentru afişarea listelor de adiacenţă

3. Scrieţi subprograme similare pentru un graf orientat.

4. Implementaţi algoritmi de trecere de la o modalitate de reprezentare la alta.

B. Determinarea de drumuri minime din parcurgerea BF



1. Se dă o reţea neorientată cu n noduri şi o listă de noduri reprezentând puncte de control pentru reţea. Se citeşte un nod de la tastatură. Să se determine cel mai apropiat punct de control de acesta şi un lanţ minim până la acesta **O(n+m)**

Exemplu: Dacă în graful considerat ca exemplu la parcurgeri punctele de control sunt 8 şi 9, şi nodul de start este 1, cel mai apropiat punct de control de 1 este 9, aflat la distanţa 2. Un lanţ minim va fi 1, 2, 9

graf.in – cel de mai sus	graf.out
	1 2 9

2. Se dă o matrice n*m (n,m <= 1000), cu p <= 100 puncte marcate cu 1 (restul valorilor din matrice vor fi 0). Distanța dintre 2 puncte ale matricei se măsoară în locații străbătute mergând pe orizontală și pe verticală între cele 2 puncte (distanța Manhattan). Se dă o mulțime *M* de q puncte din matrice (q <= 1000000). Să se calculeze cât mai eficient pentru fiecare dintre cele q puncte date, care este cea mai apropiată locație marcată cu 1 din matrice. (**Licență iunie 2015**)

Datele de intrare:

Pe prima linie a fișierului „**graf.in**" se afla valorile *n* și *m* separate printr-un spațiu.

Următoarele *n* linii reprezintă matricea cu valori 1 și 0. În final, pe câte o linie a fișierului, se află câte o pereche de numere, reprezentând coordonatele punctelor din *M*.

Date de ieșire:

Fișierul "**graf.out**" va conține q = |M| linii; pe fiecare linie *i* va fi scrisă distanța de la al *i*-lea punct din M la cel mai apropiat element marcat cu 1 în matrice, precum și coordonatele acelui element cu valoarea 1.

Exemplu

graf.in	**graf.out**
5 4 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 1 2 1 4 4 1 5 3	2 [1, 3] 1 [1, 3] 1 [1, 3] 2 [3, 2] 2 [4, 4]

3. http://www.infoarena.ro/problema/rj

4. http://www.infoarena.ro/problema/graf

C. Grafuri aciclice

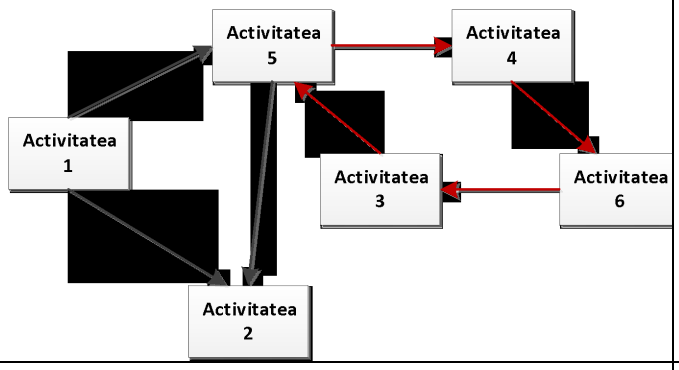
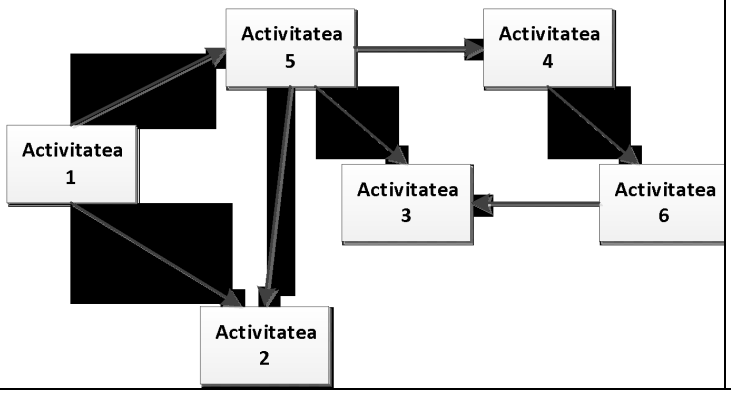
1. Dat un graf neorientat (nu neapărat conex), să se verifice dacă graful conține un ciclu elementar (nu este aciclic). În caz afirmativ **să se afișeze un astfel de ciclu**.

graf.in	**graf.out**
7 8 1 3 2 4 3 4 3 5 3 6 5 6 6 7 3 7	3 5 6 3 (nu neaparat in aceasta ordine; solutia nu este unica, un alt ciclu este de exemplu 3 6 7 3)

2. În cadrul unui proiect trebuie realizate n activităţi, numerotate 1,…,n. Activităţile nu se pot desfăşura în orice ordine, ci sunt activităţi care nu pot începe decât după terminarea altora. Date m perechi de activităţi (a, b) cu semnificaţia că activitatea trebuie să se desfăşoare înainte de activitatea b, să se testeze dacă proiectul este realizabil, adică nu există dependenţe circulare între activităţile sale. În cazul în care proiectul nu se poate realiza să se afişeze o listă de activităţi între care există dependenţe circulare.

Datele de intrare: Pe prima linie a fişierului „**`graf.in`**” se afla valorile *n* şi *m* separate printr-un spaţiu. Pe fiecare dintre următoarele m linii sunt două numere a şi b cu semnificaţia din enunţ

Date de ieşire: Fişierul ”**`graf.out`**” va conţine o listă de activităţi între care există dependenţe circulare, separate prin spaţiu, dacă astfel de activităţi există sau mesajul REALIZABIL altfel.

`graf.in`	**`graf.out`**
6 7 1 2 1 5 5 2 5 4 3 5 4 6 6 3 	5 4 6 3 (nu neaparat in aceasta ordine)
graf.in	graf.out
6 7 1 2 1 5 5 2 5 4 5 3 4 6 6 3 	REALIZABIL