- + Evaluare
- + **Bibliografie**
- + Modalități de reprezentare a grafurilor
- + Parcurgerea grafurilor
- + **Probleme**
 - Memorarea unui graf
 - Determinarea de drumuri minime din parcurgerea BF
 - Grafuri aciclice

Evaluare

- Test de laborator 1/3 din nota finală
 - obligatoriu pentru promovare examen: nota test laborator ≥ 5

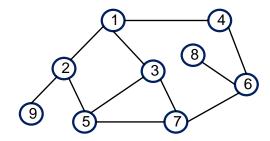
Bibliografie

- 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/reference/stl/)

Modalități de reprezentare a grafurilor

- matrice de adiacență
- liste de adiacentă
- listă de muchii

Exemplu – graf neorientat



- matricea de adiacență
 - 0
 1
 1
 1
 0
 0
 0
 0
 0
 0

 1
 0
 0
 0
 1
 0
 0
 0
 1

 1
 0
 0
 0
 1
 0
 0
 0

 1
 0
 0
 0
 1
 0
 0
 0

 0
 1
 1
 0
 0
 1
 1
 0
 0

 0
 0
 1
 0
 0
 0
 0
 0
 0

 0
 0
 0
 0
 0
 0
 0
 0
 0

 0
 1
 0
 0
 0
 0
 0
 0
 0

- liste de adiacență

| 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 |
| | |

Exerciții:

- 1. Propuneți modalități de reprezentare și pentru grafuri orientate și pentru multigrafuri neorientate/orientate (care admit muchii/arce multiple și 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 și exterior al fiecărui vârf în cazul orientat).

Parcurgerea grafurilor

Scop: Dat un graf și 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 știm că un vârf i este accesibil din s și există muchie (arc în caz orientat) de la i la j, atunci și j este accesibil din s.

Parcurgea pe lățime (BF - Breadth First)

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

Se pornește 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 și care urmează să fie explorate (în sensul că vor fi cercetați vecinii lor); operațiile de introducere în coadă și 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 ← Ø;

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

while C ≠ Ø
    {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 toţi 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 afișarea 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 și 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 lanțul/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 pornește din acest vârf și 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 aceștia și se reia procedeul.

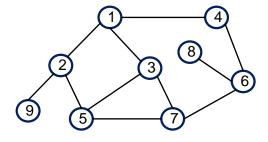
Procedura recursivă pentru parcurgerea în adâncime (DF) este

```
procedure DF(i)
  prelucreaza(i);
  vizitat(i) ←true
  for toţi 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

Des

Probleme

Pentru toate problemele din cadrul laboratorului datele se vor citi din fișierul graf.in.

Dacă nu se precizează în enunț, un graf este dat prin următoarele informații: numărul de vârfuri n, numărul de muchii m și lista muchiilor (o muchie fiind dată prin extremitățile 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 | |

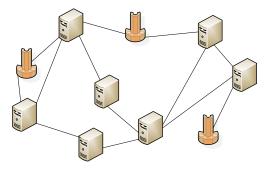
A. Memorarea unui graf

- 1. Scrieți un subprogram pentru construirea în memorie a matricei de adiacență a unui graf neorientat citit din fișierul *graf.in* cu structura precizată mai sus și un subprogram pentru afișarea matricei de adiacență
- **2.** Scrieți un subprogram pentru construirea în memorie a listelor de adiacență pentru un graf neorientat citit din fișierul *graf.in* cu structura precizată mai sus și un subprogram pentru afișarea listelor de adiacență
- 3. Scrieți subprograme similare pentru un graf orientat.
- 4. Implementați algoritmi de trecere de la o modalitate de reprezentare la alta.

B. Determinarea de drumuri minime din parcurgerea BF

1. Se dă o rețea neorientată cu n noduri și o listă de noduri reprezentând puncte de control pentru rețea. Se citește un nod de la tastatură. Să se determine cel mai apropiat punct de control de acesta și un lanț minim până la acesta O(n+m)

Exemplu: Dacă în graful considerat ca exemplu la parcurgeri punctele de control sunt 8 și 9, și nodul de start este 1, cel mai apropiat punct de control de 1 este 9, aflat la distanța 2. Un lanț 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 iesire:

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 | 2 [1, 3] |
| 0 0 1 0 | 1 [1, 3] |
| 0 0 0 0 | 1 [1, 3] |
| 0 1 0 0 | 2 [3, 2] |
| 0 0 0 1 | 2 [4, 4] |
| 0 0 0 0 | |
| 1 1 | |
| 1 2 | |
| 1 4 | |
| 4 1 | |
| 5 3 | |

- 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 | 3 5 6 3 |
| 1 3 | (nu neaparat in aceasta ordine; |
| 2 4 | solutia nu este unica, un alt |
| 3 4 | ciclu este de exemplu 3 6 7 3) |
| 3 5 | |
| 3 6 | |
| 5 6 | |
| 6 7 | |
| 3 7 | |

2. În cadrul unui proiect trebuie realizate n activități, numerotate 1,...,n. Activitățile nu se pot desfășura în orice ordine, ci sunt activități care nu pot începe decât după terminarea altora. Date m perechi de activități (a, b) cu semnificația că activitatea trebuie să se desfășoare înainte de activitatea b, să se testeze dacă proiectul este realizabil, adică nu există dependențe circulare între activitățile sale. În cazul în care proiectul nu se poate realiza să se afișeze o listă de activități între care există dependențe circulare.

Datele de intrare: Pe prima linie a fișierului "graf.in" se afla valorile *n* și *m* separate printrun spațiu. Pe fiecare dintre următoarele m linii sunt două numere a și b cu semnificația din enunț

Date de ieșire: Fișierul "graf.out" va conține o listă de activități între care există dependențe circulare, separate prin spațiu, dacă astfel de activități există sau mesajul REALIZABIL altfel.

