* [**Evaluare**](#eval)
* [**Bibliografie**](#bibl)
* [**Modalități de reprezentare a grafurilor**](#repr)
* [**Parcurgerea grafurilor**](#parc)
* **[Probleme](#pb)**
* **Memorarea unui graf**
* **Determinarea de drumuri minime din parcurgerea BF**
* **Grafuri aciclice** **Evaluare**
* **2 puncte din nota finală**
* **teme de lucru în timpul laboratorului + teme pentru acasă**

**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 Universităţii din Bucureşti, 2005
5. J.A. Bondy, U.S.R Murty – Graph theory with applications, The Macmillan Press 1976, Springer 2008

**Modalități de reprezentare a grafurilor**

**- matrice de adiacență**

**- liste de adiacență**

**- listă de muchii**

* **Exemplu** **– graf neorientat**

|  |  |
| --- | --- |
|  | **- matricea de adiacență**  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 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 |

* **Exerciţii:**

1. Propuneţi modalităţi de reprezentare şi pentru grafuri orientate și pentru multigrafuri neorientate/orientate (care admit muchii/arce multiple şi bucle)
2. Precizaţi, 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).
3. Propuneţi algoritmi de trecere de la o modalitate de reprezentare la alta.

**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 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 lăţime (BF - Breadth First)**

Parcurgerea pe lăţime BF urmăreşte vizitarea vârfurilor **în ordinea crescătoare a distanţelor** lor faţă de s (distanţa este dată de numărul de muchii).

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

**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 cercetaţi vecinii lor); operaţiile de introducere în coadă şi eliminare din coadă sunt simbolizate prin ⇒ , respectiv ⇐. Eventuala existenţă 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 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 afişarea informaţiei din vârful respectiv.

Muchiile folosite de algoritm pentru a descoperi noi vârfuri accesibile din s formează un **arbore de rădăcină s.** Pentru a reţine acest arbore putem folosi legături de tip tată (predecesor). Astfel, vom mai avea un vector tata care se iniţializează 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 **lanțul/drumul determinat prin parcurgerea BF de la sla 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 nevizitaţi anterior, dacă un astfel de vecin există. Dacă toţi vecinii vârfului curent au fost deja vizitaţi se merge înapoi pe drumul de la s la vârful curent până se ajunge la un vârf care mai are vecini nevizitaţi; 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 toţi j vecini ai lui i

if not vizitat(j) then DF(j)

Vectorul vizitat se iniţializează 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.** 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.

**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 şi 9, și 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 conţine un ciclu elementar (nu este aciclic). În caz afirmativ **să se afişeze 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 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 printr-un 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.

|  |  |
| --- | --- |
| **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 |