Laboratorator Algoritmi și Structuri de Date
Tema 12

Tema săptămânii 12. Ultima temă.

Grafuri

Un graf $G = (V, E)$ se defineste formal ca fiind compus dintr-o multime de varfuri V si o multime de muchii E in care fiecare muchie $e \in E$ este de forma $e = (v_1, v_2)$ cu $v_1, v_2 \in V$.

Grafurile sunt de mai multe feluri:

- (orientate, neorientate) Daca exista $e = (v_1, v_2) \in G$ inseamna ca putem sa ajungem de la v_1 la v_2, dar daca graful este orientat, nu putem sa ajungem de la v_2 la v_1. Intr-un graf orientat muchiile se numesc arce, iar $(v_1, v_2) \neq (v_2, v_1)$.

- (cu ponderi sau fara) Pentru orice muchie putem sa mentinem o pondere sau informatie asociata muchiei (de pilda costul traversarii acelei muchii, distanta dintre doua orase pe harta etc.): $e = (v_1, v_2, pondere)$

Cea mai imediata metoda de reprezentare a unui graf cu n varfuri G cu $V = \{v_1, ..., v_n\}$ in calculator este prin matricea de adiacenta.

Intr-o matrice de adiacenta vom asocia $M[i][j] = 1$ numai si numai daca $\exists(v_i, v_j) \in E$ = muchia e prezenta in graf. Daca graful e ponderat, se pune ponderea in loc de 1.

Aceasta metoda are limitari: ca sa pargurgem toti vecinii unui nod v_i parcurgem o intreaga linie in matrice in timp $O(n)$. In plus spatiul necesar stocarii grafului este $O(n^2)$. Matricea de adiacenta e folositoare daca graful este (aproape) complet = contine (mai toate) muchiile, dar in cazul acesta graful este neinteresant (exista drum direct = muchie intre oricare doua noduri).

O varianta mai buna este prin liste de adiacenta: un vector de liste care enumera vecinii fiecarui nod. Implementarea va fi de tipul:

```
int n;
struct nod {int info; nod *next;} *adiacenta[100];
//fiecare adiancenta[i] reprezinta pointerul prim
   pentru fiecare lista de vecini; daca e graf
   ponderat se pune ponderea in nod
```

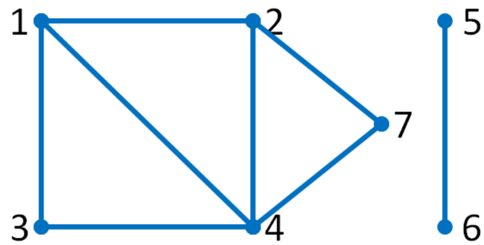

Figure 1: Un exemplu de graf

Matrice de adiacenta pentru exemplul dat:

```
   1  2  3  4  5  6  7
1  0  1  1  1  0  0  0
2  1  0  0  1  0  0  1
3  1  0  0  1  0  0  0
4  1  1  1  0  0  0  1
5  0  0  0  0  0  1  0
6  0  0  0  0  1  0  0
7  0  1  0  1  0  0  0
```

Lista de adiacenta pentru exemplul dat:

```
7 noduri
1 : 2->3->4
2 : 1->4->7
3 : 1->4
4 : 1->2->3->7
5 : 6
6 : 5
7 : 2->4
```

```
DFS(din 1): 1 2 4 3 7
BFS(din 1): 1 2 3 4 7
```

Pornind dintr-un nod initial v_i definim parcurgeri astfel:

0. se pune nodul intr-o (stiva/coada)

Cata vreme (stiva/coada) nu este vida:

1. se extrage un nod din (stiva/coada)
2. se afiseaza (sau se calculeaza ceva)
3. se marcheaza nodul ca fiind vizitat ($= 1$ intr-un vector $viz[]$)
4. se pun toti vecinii lui nevizitati in (stiva/coada)

Daca se foloseste o stiva vom avea o parcurgere DFS (Depth First Search, parcurgere in adancime), iar pentru coada vom avea BFS (Breadth First Search, parcurgere in latime).

Cerinte:

0. Sa se citeasca un graf (din fisier).

(+2p) 1. Sa se reprezinte cu matrice de adiacenta.

(+3p) 2. Sa se reprezinte cu liste de adiacenta.

Apoi, indiferent de reprezentare:

(+1p) 3. Sa se scrie o functie $grad(i)$ care calculeaza cati vecini are v_i.

(+1p) 4. Sa se scrie o functie $maxgrad()$ care afiseaza toate nodurile cu grad maxim.

(+1p) 5. Sa se scrie o functie $numarmuchii()$ care calculeaza cate muchii are graful.

(+2p) 6. Sa se parcurga graful cu DFS.

(+2p) 7. Sa se parcurga graful cu BFS.