Structuri de Date și Algoritmi

Laboratorul 9: Grafuri neorientate

Mihai Nan

25 aprilie 2024

1. Introducere

Se numeste **graf** sau **graf neorientat** o structura ***G=(V,E)***, unde ***V*** este o multime nevida si finita de elemente denumite **varfurile** grafului, iar ***E*** este o submultime, posibil vida, a multimii perechilor neordonate cu componente distince din ***V***.

In cazul grafurilor neorientate, perechile de varfuri din multimea ***E*** sunt neordonate si sunt denumite **muchii**. Perechea neordonata formata din varfurile ***u*** si ***v*** se noteaza ***(u,v)***, varfurile ***u*** si ***v*** numindu-se **extremitatile** muchiei ***(u,v)***.

Daca exista un **arc** sau o **muchie** cu extremitatile ***u*** si ***v***, atunci varfurile ***u*** si ***v*** sunt **adiacente**, fiecare extremitate a muchiei fiind considerata **incidenta** cu muchia respectiva.

Fie ***G=(V,E)*** un graf. Elementul ***v*** $\in$ ***V*** se numeste **varf izolat** daca, pentru orice ***e*** $\in$ ***E***, ***v*** nu este incident cu ***e***.

Fie ***G=(V,E)*** un graf. Gradul lui ***v*** $\in$ ***V*** este numarul de muchii incidente cu ***v***.

$$grad(v) = |\{e \in E | e = (v, x) \ sau \ e = (x, v), x \in V\}| \tag{1}$$

2. Observatii

Cu ajutorul unui **graf neorientat** putem modela o **relatie simetrica** intre elementele unei multimi.

Intre oricare doua varfuri ale unui graf poate exista cel mult o muchie. Daca intre doua varfuri exista mai multe muchii, atunci structura poarta denumirea de **multigraf**. In continuare, nu vom trata aceasta structura, ci vom lucra doar cu structura simpla de **graf neorientat**.

In practica, informatiile asociate unui graf pot fi oricat de complexe, dar, pentru simplitate, vom considera ca varfurile grafului sunt etichetate cu numere naturale de la ***0*** la $n \in N^*$.

3. Metode de reprezentare

Pentru a putea prelucra un **graf neorientat** cu ajutorul unui program, trebuie mai intai sa fie reprezentat in programul respectiv. Pentru a reprezenta un graf, intr-un program, exista mai multe modalitati, folosind diverse structuri de date, precum: **matrice de adiacenta**, **liste de adiacenta**, **lista de muchii**.

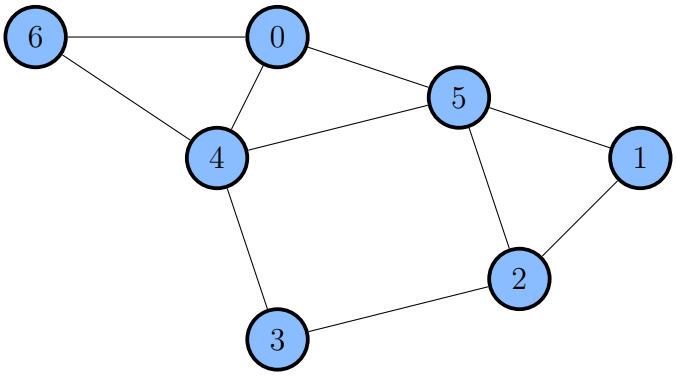
3.1 Matrice de adiacenta

Fie ***G=(V,E)*** un graf neorientat cu ***n*** varfuri si ***m*** muchii. Matricea de adiacenta, asociata grafului ***G***, este o matrice patratica de ordinul ***n***, cu elementele definite astfel:

$$a_{i,j} = \begin{cases} 1 & \text{daca } (i,j) \in E \\ 0 & \text{daca } (i,j) \notin E \end{cases} \tag{2}$$

Suma elementelor de pe linia ***i*** si respectiv suma elementor de pe coloana ***j*** au ca rezultat gradul nodului ***i***, respectiv ***j***.

Suma tuturor elementelor matricei de adiacenta este suma gradelor tuturor nodurilor, adica dublul numarului de muchii.

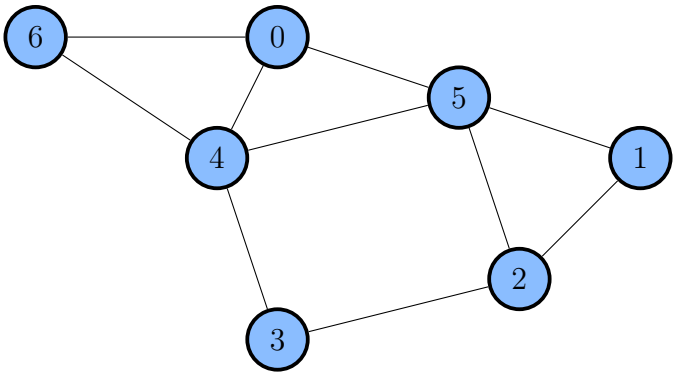


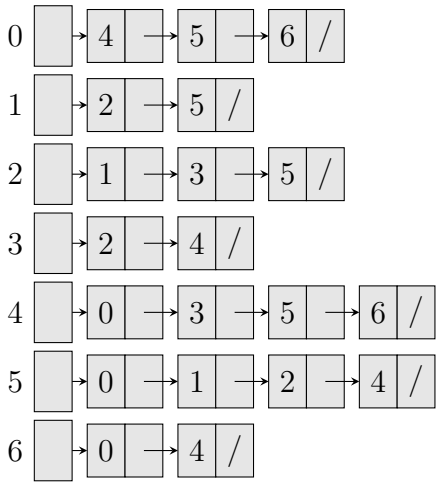
	0	1	2	3	4	5	6
0	0	0	0	0	1	1	1
1	0	0	1	0	0	1	0
2	0	1	0	1	0	1	0
3	0	0	1	0	1	0	0
4	1	0	0	1	1	0	1
5	1	1	1	0	0	0	0
6	1	0	0	0	1	0	0

3.2 Liste de adiacenta

Fie ***G=(V,E)*** un graf neorientat cu ***n*** varfuri si ***m*** muchii.Reprezentarea grafului ***G*** prin **liste de adiacenta** consta in:

- precizarea numarului de varfuri si a numarului de muchii;
- pentru fiecare varf i se precizeaza lista vecinilor sai, adica lista nodurilor adiacente cu nodul i.





Observatie In cadrul acestei reprezentari, se va folosi un vector, alocat dinamic, avand elemente de tip lista simplu inlantuita / dublu inlantuita.

4. Algoritmi de parcurgere

Parcurgerea unui graf presupune examinarea sistematica a varfurilor grafului, cu scopul prelucrarii informatiilor asociate varfurilor. Exista doua metode fundamentale de parcurgere a grafurilor:

- **Parcurgerea in latime (BFS – Breadth First Search)**
- **Parcurgerea in adancime (DFS – Depth First Search)**

Prin parcurgerea unui graf neorientat se intelege examinarea in mod sistematic a nodurilor sale, plecand dintr-un varf dat i, astfel incat fiecare nod accesibil din i, pe muchii adiacente doua cate doua, sa fie atins o singura data.

Graful este structura neliniara de organizare a datelor, iar rolul traversarii sale poate fi si determinarea unei aranjari linieare a nodurilor in vederea trecerii de la unul la altul.

4.1 Parcurgerea in latime

Parcurgerea in latime este unul dintre cei mai simplii si, poate, folositori algoritmi de cautare in grafuri. Se obtin drumurile dintr-un nod sursa catre orice nod din graf astfel:

Fiind date un graf ***G=(V,E)*** si un nod sursa $\boldsymbol{s}$, aceasta parcurgere va permite explorarea sistematica in $\boldsymbol{G}$ si descoperirea fiecarui nod, plecand din $\boldsymbol{s}$. Totodata, se poate calcula si distanta de la $\boldsymbol{s}$ la fiecare nod ce poate fi vizitat. In felul acesta, se contruieste un arbore „pe latime" cu radacina in $\boldsymbol{s}$ ce contine toate nodurile ce pot fi vizitate. Pentru orice nod $\boldsymbol{v}$, ce poate fi vizitat plecand din $\boldsymbol{s}$, drumul de la radacina la acest nod, drum refacut din arborele ,,pe latime", este cel mai scurt de la $\boldsymbol{s}$ la $\boldsymbol{v}$, in sensul ca acest drum contine cele mai putine muchii.

Algorithm 1 BreadthFirstSearch

```
1: procedure BFS(G = {V, E}, start)
2:     for each vertex u ∈ V\{start} do
3:         color[u] ← WHITE
4:         dist[u] ← ∞
5:         parent[u] ← NIL
6:     color[u] ← GRAY
7:     dist[u] ← 0
8:     Q ← ∅
9:     Q ← enqueue(Q, start)
10:    while Q ≠ ∅ do
11:        u ← dequeue(Q)
12:        for each vertex v ∈ V\{u} do
13:            if (u, v) ∈ E then
14:                if color[v] = WHITE then
15:                    color[v] ← GRAY
16:                    dist[v] ← dist[u] + 1
17:                    parent[v] ← u
18:                    Q ← enqueue(Q, v)
19:        color[u] ← BLACK
```

Observatie Parcurgerea in latime este cunoscuta si ca algoritmul lui Lee in lumea algoritmicii romanesti. Implementarea algoritmului de mai sus are la baza o metoda iterativa si foloseste, ca structura auxiliara de date, o coada. Fiecare vecin va fi introdus in coada, iar la extragerea unuia din structura vom introduce in coada toti vecinii nevizitati ai nodului curent, avand grija sa eliminam nodul curent. Algoritmul se repeta pana cand coada devine vida.

4.2 Parcurgerea in adancime

Spre deosebire de algoritmul prezentat anterior, in cazul parcurgerii in adancime sunt explorate in felul urmator muchiile: daca ajunge in nodul $\boldsymbol{v}$, vor fi explorate acele muchii care sunt conectate la $\boldsymbol{v}$, dar care inca nu au fost descoperite. Atunci cand toate muchiile lui

v au fost explorate, cautarea se retrage pentru a explora muchiile ce pleaca din nodurile adiacente lui ***v***. Daca raman noduri nedescoperite, atunci unul din ele poate fi stabilit ca sursa si cautarea se repeta din acea sursa.

Algorithm 2 Depth First Search

```
1:  procedure DFS(G = {V, E}, start)
2:      S ← ∅
3:      S ← push(S, start)
4:      while !isEmpty(S) do
5:          u ← pop(S)
6:          if viz[u] = false then
7:              viz[u] ← true
8:              for each vertex x ∈ V\{u} do
9:                  if (u, x) ∈ E then
10:                     if viz[x] = false then
11:                         S ← push(S, x)
```

5. Cerințe propuse

În arhiva laboratorului, găsiți un schelet de cod de la care puteți porni implementarea funcțiilor propuse, fiind necesară testarea manuală a funcționalității.

Cerința 1 (4p) Definiți o structură de date de tip graf, utilizând reprezentarea cu liste de adiacență. Implementați funcțiile descrise mai jos, pentru definirea operațiilor elementare cu acest tip de structura de date, și testați funcționalitatea lor, folosind scheletul din arhiva laboratorului.

```
//Aloca memorie si initializeaza campurile structurii
Graph initGraph(int nrVarfuri);
//Adauga muchia (u, v) in graf
Graph addEdge(Graph g, int u, int v);
//Sterge nodul v din graf
Graph deleteVertex(Graph g, int v);
//Returneaza gradul nodului v
int getDegree(Graph g, int v);
//Afiseaza graful
void printGraph(Graph g);
```

Cerința 2 (2p) Implementați o funcție iterativă, care folosește ca structura auxiliara de date o stivă, pentru parcurgerea în adâncime a unui graf. Pentru testarea funcționalității, apelați această funcție pentru fiecare nod din graf. Determinați astfel numărul de componente conexe ale grafului.

Cerința 3 (2p) Implementați o funcție recursivă pentru parcurgerea în adâncime a unui graf. Comparați rezultatele cu cele ale funcției iterative.

Cerința 4 (2p) Implementați o funcție pentru parcurgerea în lățime a unui graf neorientat.