Structuri de date

Laboratorul 8

Mihai Nan
mihai.nan.cti@gmail.com
Grupa 314CC

Facultatea de Automatică şi Calculatoare
Universitatea Politehnica din Bucureşti
Anul universitar 2017 - 2018

1 Grafuri neorientate

1.1 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.

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

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 $n \in N^*$.

1.3 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**.

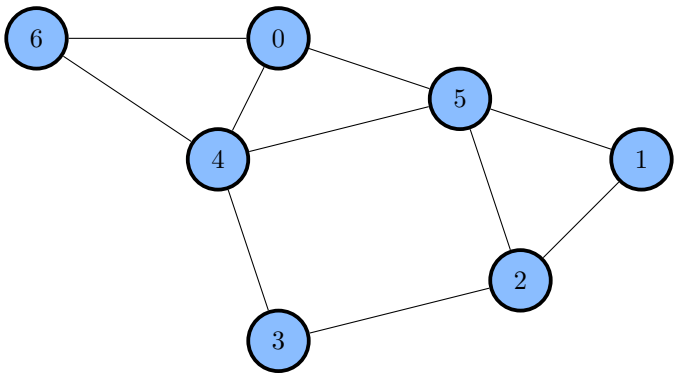
1.3.1 Matrice de adiacenta

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.

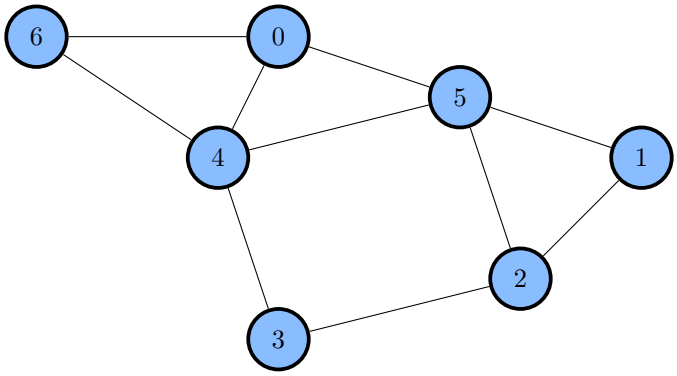


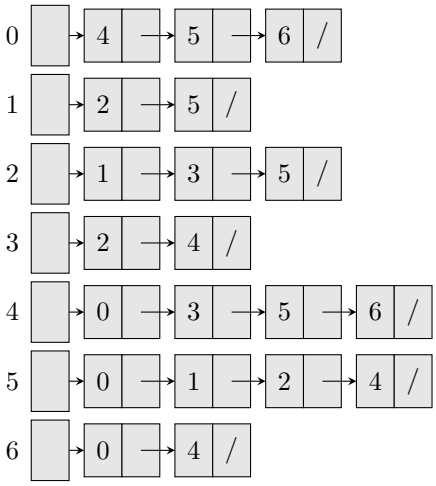
	0	1	2	3	4	5	6
0	0	0	0	0	1	1	1
1	0	0	1	0	0	1	0
2	0	1	0	1	0	1	0
3	0	0	1	0	1	0	0
4	1	0	0	1	1	0	1
5	1	1	1	0	0	0	0
6	1	0	0	0	1	0	0

1.3.2 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.

1.4 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)**

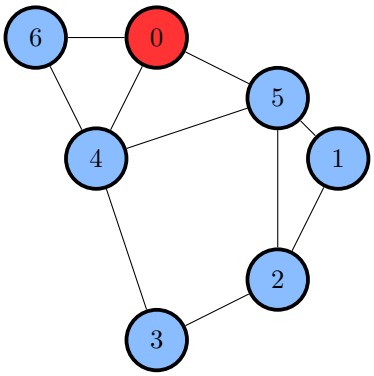
Prin parcurgerea unui graf neorientat se intelege examinarea in mod sistematic a nodurilor sale, plecand dintr-un varf dat i, astfel incat fiecare nod accesibil din i, pe muchii adiacente doua cate doua, sa fie atins o singura data.

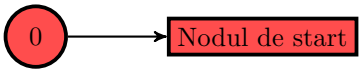
Graful este structura neliniara de organizare a datelor, iar rolul traversarii sale poate fi si determinarea unei aranjari linieare a nodurilor in vederea trecerii de la unul la altul.

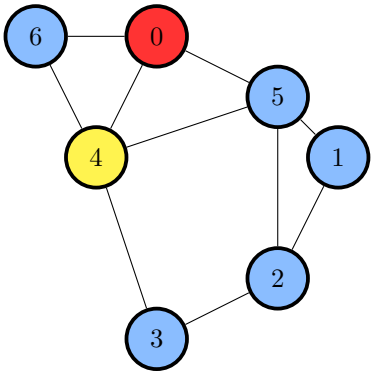
1.4.1 Parcurgerea in latime

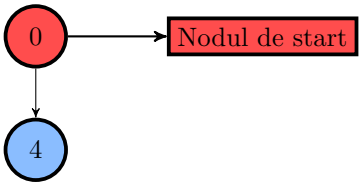
Parcurgerea in latime este unul dintre cei mai simplii si, poate, folositori algoritmi de cautare in grafuri. Se obtin drumurile dintr-un nod sursa catre orice nod din graf astfel:

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

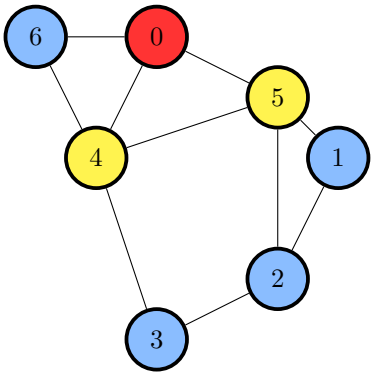


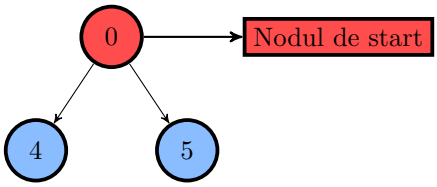



6
0
5
4
1
2
3

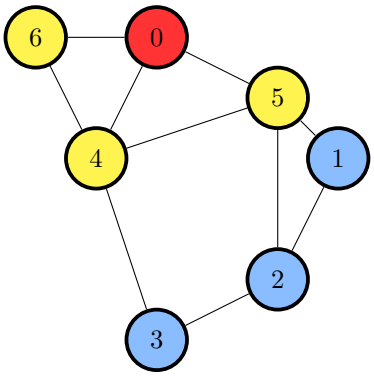

0
Nodul de start
4

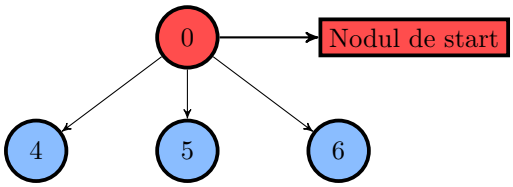
0 **4**


6
0
5
4
1
2
3

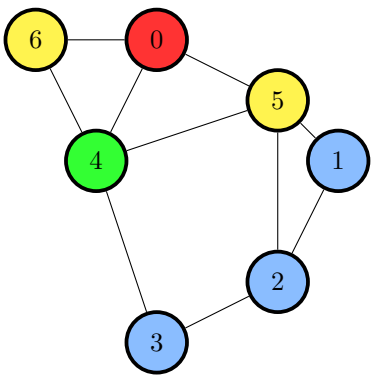

0
Nodul de start
4
5

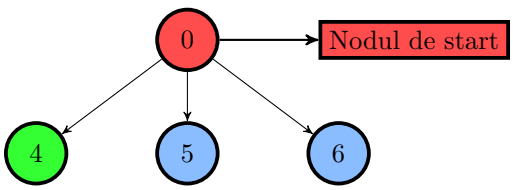
0 **4** **5**


6
0
5
4
1
2
3

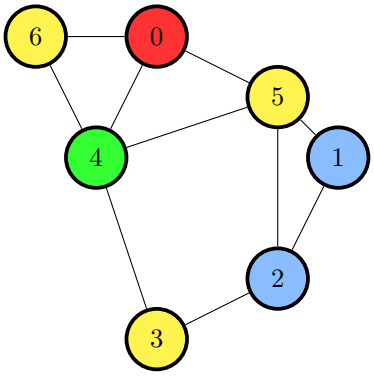

0
Nodul de start
4
5
6

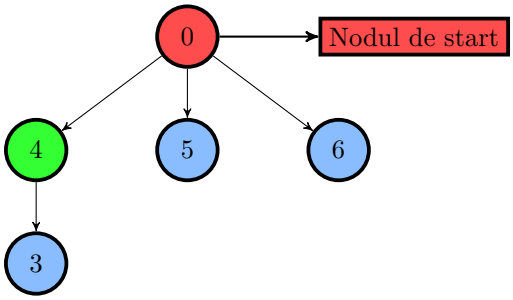
0 **4** **5** **6**


6
0
5
4
1
2
3

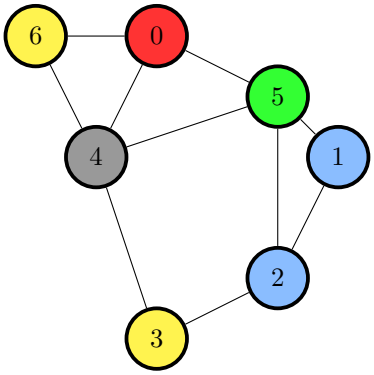

0
Nodul de start
4
5
6

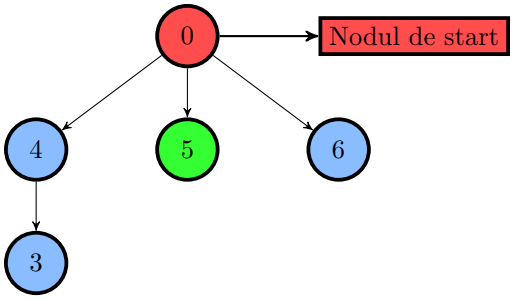
0	4	5	6



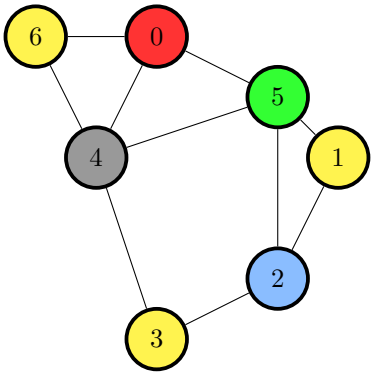


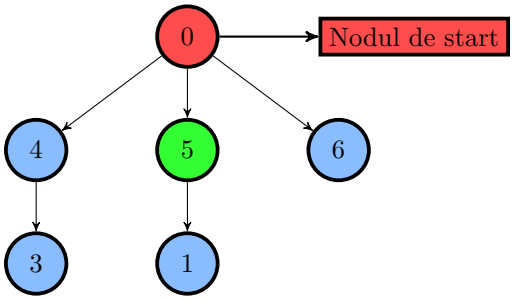
0	4	5	6	3



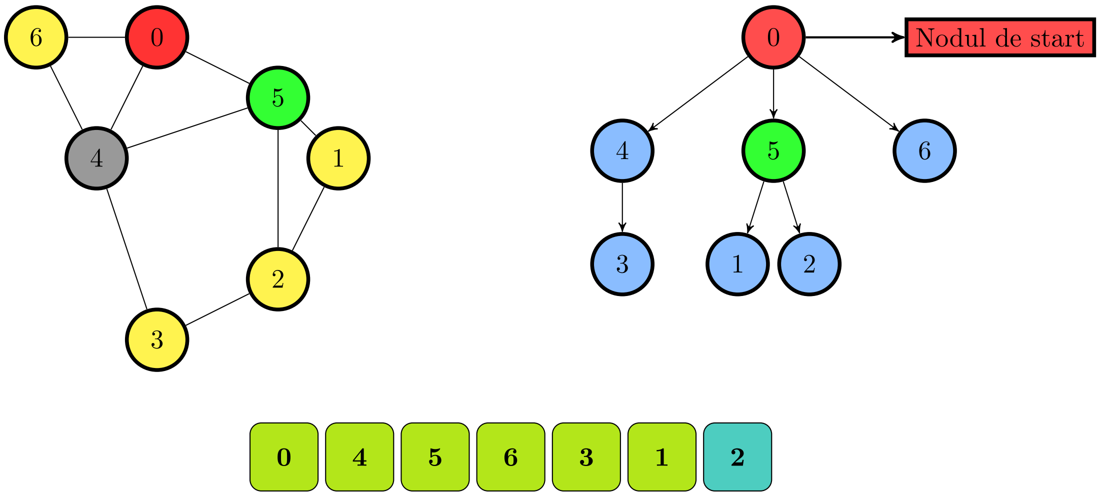


0	4	5	6	3

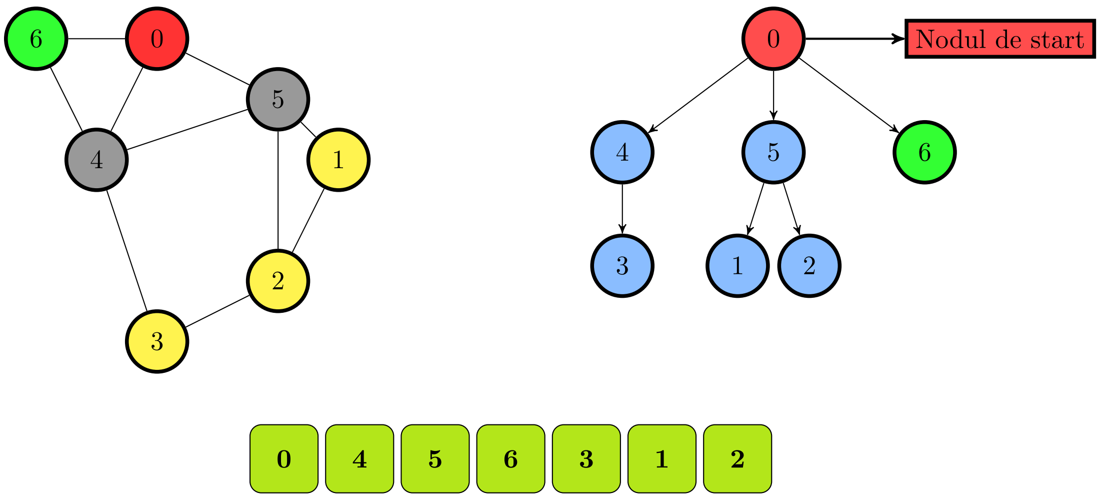




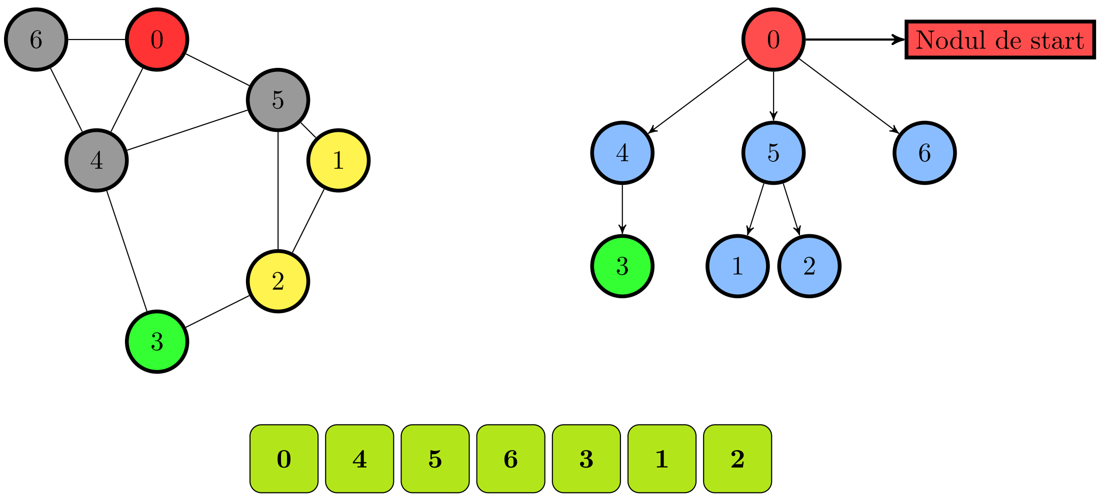
0	4	5	6	3	1



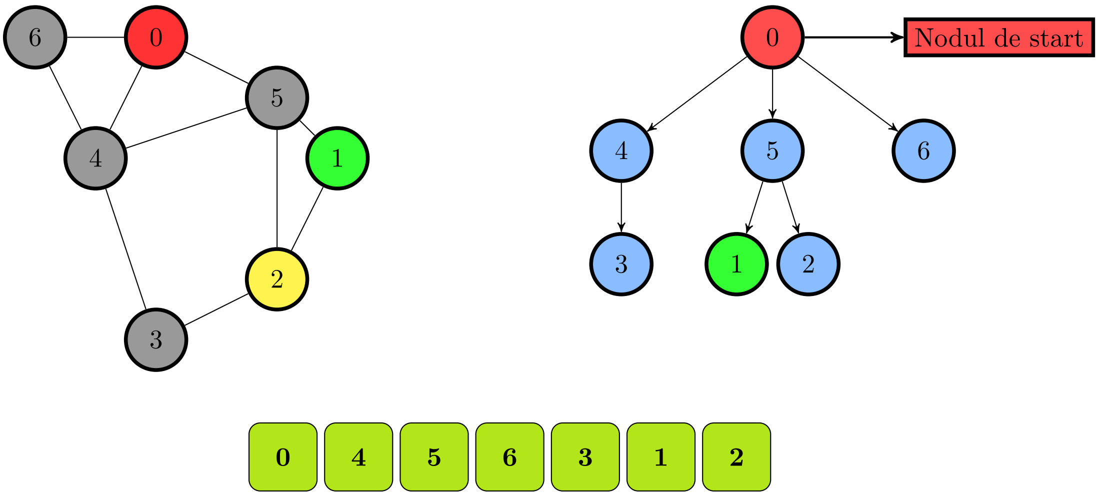
Nodul de start
0
4
5
6
3
1
2

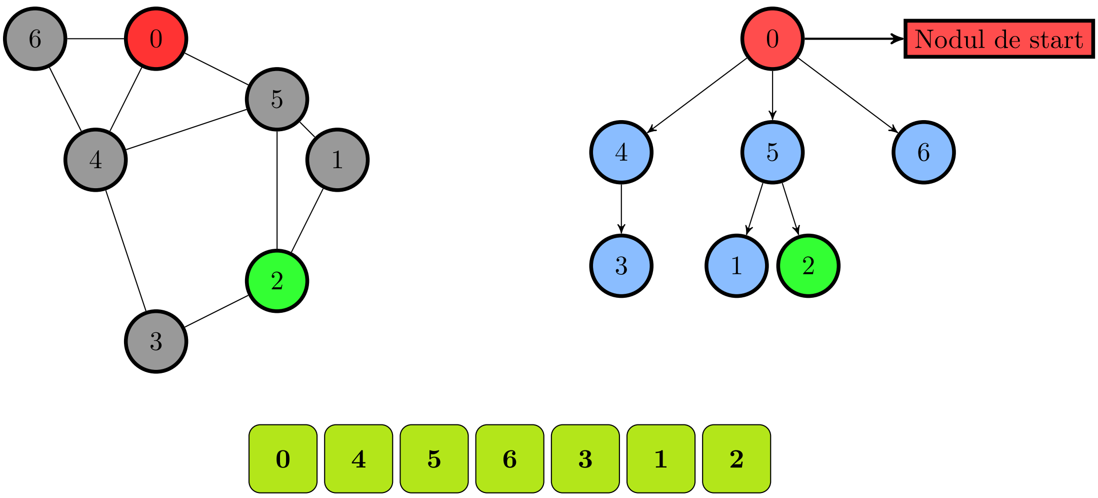


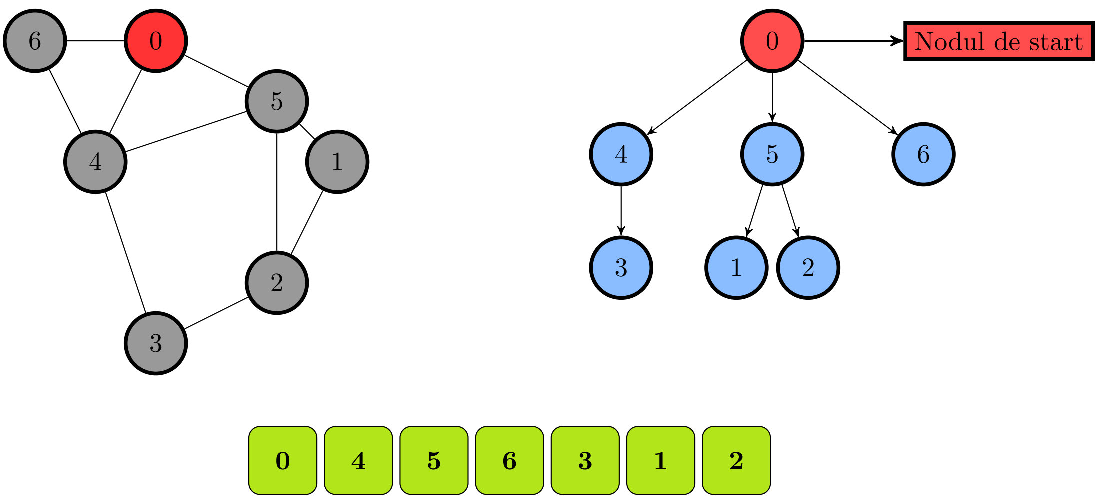
Nodul de start
0
4
5
6
3
1
2



Nodul de start
0
4
5
6
3
1
2







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.

Algorithm 1 BreadthFirstSearch

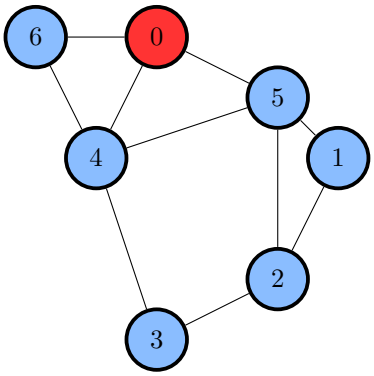
```
1:  procedure BFS(G = {V, E}, start)
2:      for each vertex u ∈ V\{start} do
3:          color[u] ← WHITE
4:          dist[u] ← ∞
5:          parent[u] ← NIL
6:      color[start] ← GRAY
7:      dist[start] ← 0
8:      Q ← ∅
9:      Q ← enqueue(Q, start)
10:     while Q ≠ ∅ do
11:         u ← dequeue(Q)
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
```

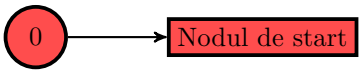
1.4.2 Parcurgerea in adancime

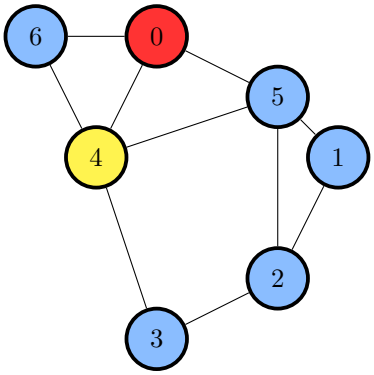
Spre deosebire de algoritmul prezentat anterior, in cazul parcurgerii in adancime sunt explorate in felul urmator muchiile: daca ajunge in nodul $\boldsymbol{v}$, vor fi explorate acele muchii care sunt conectate la $\boldsymbol{v}$, dar care inca nu au fost descoperite. Atunci cand toate muchiile lui $\boldsymbol{v}$ au fost explorate, cautarea se retrage pentru a explora muchiile ce pleaca din nodurile adiacente lui $\boldsymbol{v}$. Daca raman noduri nedescoperite, atunci unul din ele poate fi stabilit ca sursa si cautarea se repeta din acea sursa.

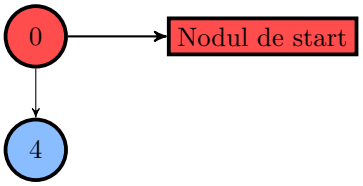
Algorithm 2 Depth First Search

```
1:  procedure DFS(G = {V, E}, start)
2:      S ← ∅
3:      S ← push(S, v)
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)
```

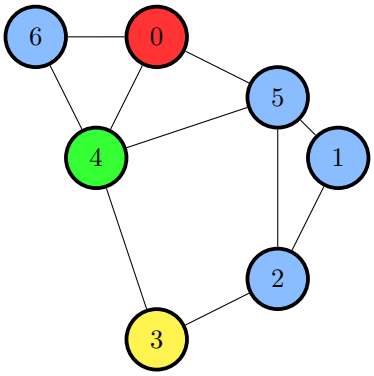


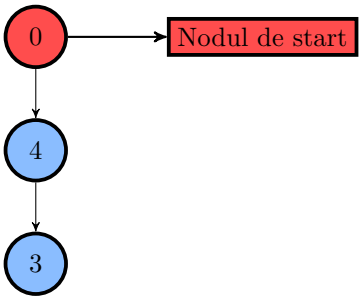



6
0
5
4
1
2
3

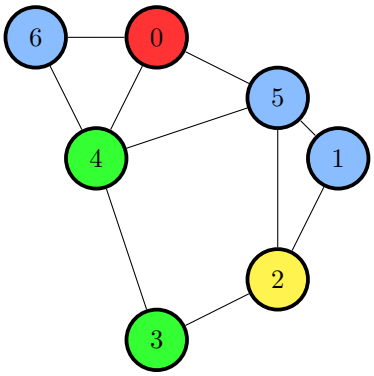

0
Nodul de start
4

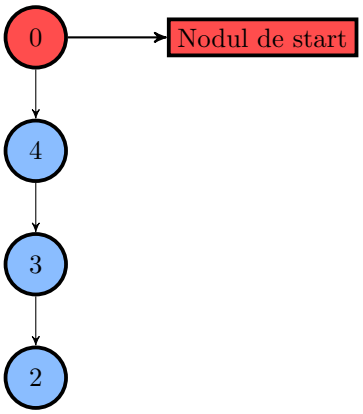

0
4


6
0
5
4
1
2
3

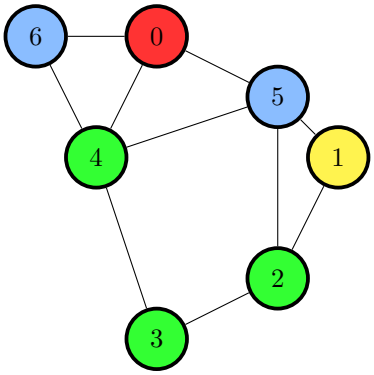

0
Nodul de start
4
3

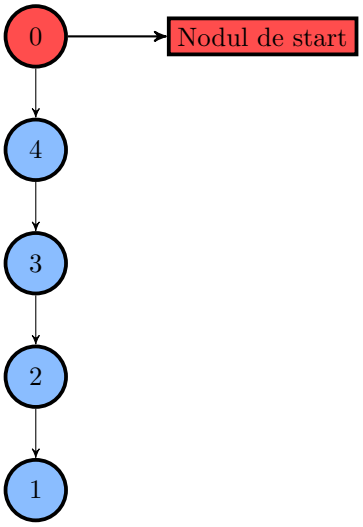

0
4
3


6
0
5
4
1
2
3

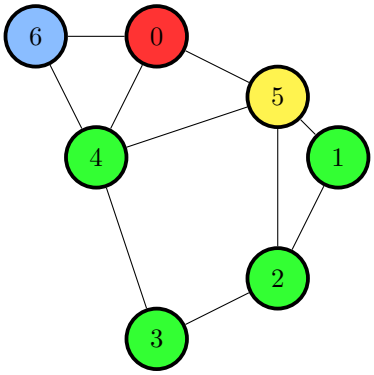

0
Nodul de start
4
3
2

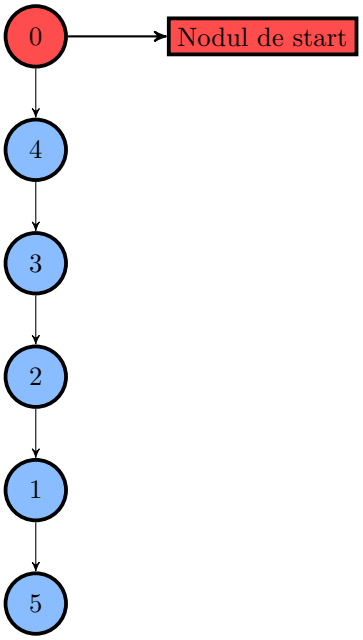

0
4
3
2

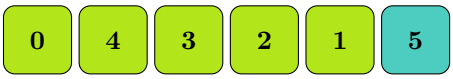

6
0
5
4
1
2
3

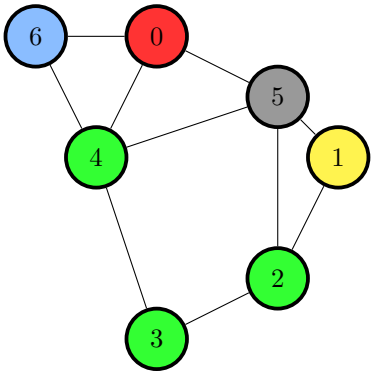

0
Nodul de start
4
3
2
1

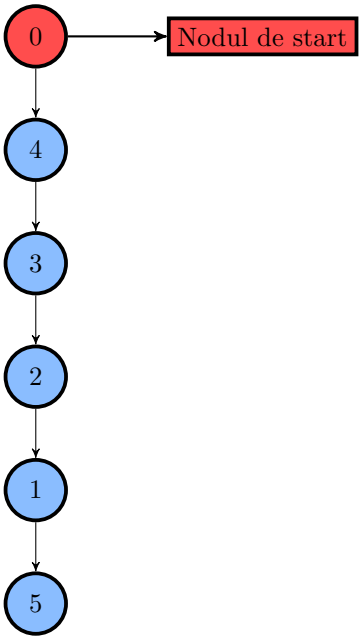

0
4
3
2
1


6
0
5
4
1
2
3


0
Nodul de start
4
3
2
1
5

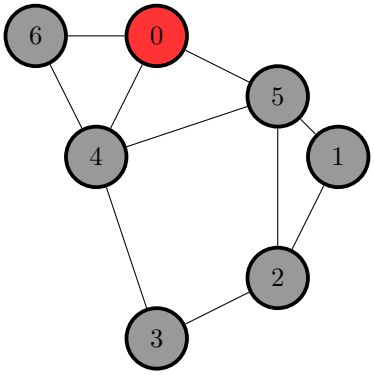

0
4
3
2
1
5

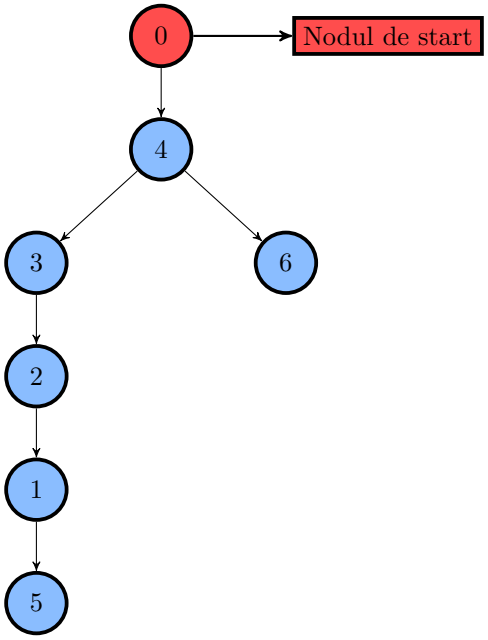




In acest moment, ne vom intoarce in nodul 1 si vedem ca nu mai avem nimic de vizitat, apoi in 2 si descoperim acelasi lucru si tot asa pana ajungem in nodul 4. Cand ajungem in nodul 4, descoperim nodul 6 nevizitat si il afisam si adaugam in stiva. Dupa acest pas, o sa avem in stiva doar nodul 6 pe care il scoatem, la ultimul pas al parcurgerii, si descoperim ca nici acesta nu mai are niciun vecin nevizitat, ceea ce inseamna ca algoritmul de parcurgere s-a terminat.

In figura de mai jos, este prezentat rezultatul final al algoritmului de parcurgere in adancime.





2 Probleme de laborator

Observatie

In arhiva laboratorului, gasiti un schelet de cod de la care puteti porni implementarea functiilor propuse, avand posibilitatea de a le testa functionalitatea.

2.1 Probleme standard

Problema 1 - 2 puncte

Sa se defineasca o structura de date de tip graf, uzitand reprezentarea cu liste de adiacenta. Implementati functiile descrise mai jos, pentru definirea operatiilor elementare cu acest tip de structura de date, si testati functionalitatea lor, folosind scheletul din arhiva laboratorului.

Cod sursa C

```
//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 intern al nodului v
int getInDegree(Graph g, int v);
//Afiseaza graful
void printGraph(Graph g);
```

Problema 2 - 2 puncte

Sa se defineasca o functie iterativa, care foloseste ca structura auxiliara de date o stiva, pentru parcurgerea in adancime a unui graf. Pentru testarea functionalitatii, apelati aceasta functie pentru fiecare nod din graf.

Problema 3 - 2 puncte

Sa se defineasca o functie recursiva pentru parcurgerea in adancime a unui graf. Comparati rezultatele cu cele ale functiei iterative.

Problema 4 - 3 puncte

Implementati o functie pentru parcurgerea in latime a unui graf neorientat pornind din nodul 1. Pentru fiecare nod **u** din graf se va determina distanta minima de la nodul 1 la **u**, ca numar de muchii. Distanta o sa aiba valoarea -1 pentru noduri inaccesibile din 1.

Problema 5 - 1 punct

Scrieti o functie care determina numarul de componente conexe dintr-un graf neorientat.

Definitie: Fie $G = (V, E)$ un graf neorientat, unde V are n elemente (noduri) si E are m elemente (muchii). Definim $G_1 = (V_1, E_1)$ ca fiind o componenta conexa daca:

- pentru orice pereche x, y de noduri din V_1 exista un lant de la x la y (implicit si de la y la x, deoarece graful este neorientat);
- nu exista alt subgraf al lui G, $G_2 = (V_2, E_2)$ care sa indeplineasca prima conditie si care sa-l contina pe G_1.

2.2 Probleme bonus

Problema 6 - 1 punct

Implementati o functie care verifica daca un graf neorientat, primit ca parametru, este conex.

Problema 7 - 3 puncte

Se da un graf neorientat $G = (V, E)$. Un graf se numeste graf biconex daca nu are puncte de articulatie. Un nod se numeste punct de articulatie daca subgraful obtinut prin eliminarea nodului si a muchiilor incidente cu acesta nu mai este conex. O componentă biconexa a unui graf este un subgraf biconex maximal cu aceasta proprietate.

Dandu-se un graf neorientat $G = (V, E)$ se cere sa se determine componentele sale biconexe.

3 Interviu

Observatie

Aceasta sectiune este una optionala si incearca sa va familiarizeze cu o serie de intrebari ce pot fi adresate in cadrul unui interviu tehnic. De asemenea, aceasta sectiune poate fi utila si in pregatirea pentru examenul final de la aceasta disciplina.

Intrebari interviu

1. Definiti termenul de graf bipartit. Cum putem determina daca un graf este sau nu bipartit? Ce structuri de date considerati ca am putea folosi in implementare?
2. Definiti notiunea de sortare topologica. Ce tip de parcurgere uzitam in implementarea sortarii topologice?
3. Ce tip de parcurgere se poate folosi pentru determinarea drumului minim, ca numar de noduri din care este compus, intre doua noduri?
4. Ce tip de reprezentare este mai recomandata in cazul unui graf dens? Dar a unui graf rar? Motivati alegerile!
5. Cum se numeste un graf in care toate nodurile au acelasi grad? La ce ar putea fi util un astfel de graf?

Feedback

Pentru imbunatatirea constanta a acestui laborator, va rog sa completati formularul de feedback disponibil aici.

De asemenea, va rog sa semnalati orice greseala / neclaritate depistata in laborator pentru a o corecta.

Va multumesc anticipat!