Laboratorul 7 – Proiectarea algoritmilor
Seria CD

Mihai Nan – mihai.nan.cti@gmail.com

Anul universitar 2019 – 2020

1 Recapitulare grafuri

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**.

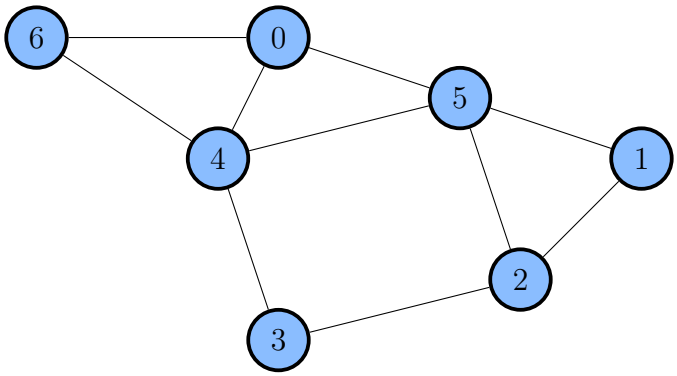
1.3.1 Matrice de adiacenta

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.

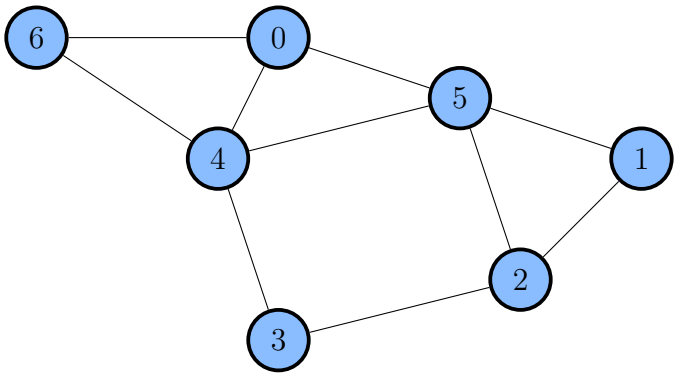


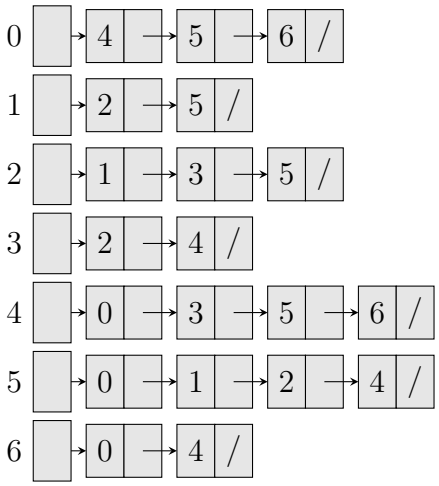
	0	1	2	3	4	5	6
0	0	0	0	0	1	1	1
1	0	0	1	0	0	1	0
2	0	1	0	1	0	1	0
3	0	0	1	0	1	0	0
4	1	0	0	1	1	0	1
5	1	1	1	0	0	0	0
6	1	0	0	0	1	0	0

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.

2 Algoritmi de parcurgere pentru grafuri

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.

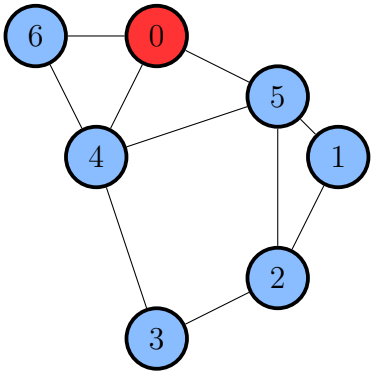
Graful este structura neliniara de organizare a datelor, iar rolul traversarii sale poate fi si determinarea unei aranjari linieare a nodurilor in vederea trecerii de la unul la altul.

2.0.1 Parcurgerea in latime

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

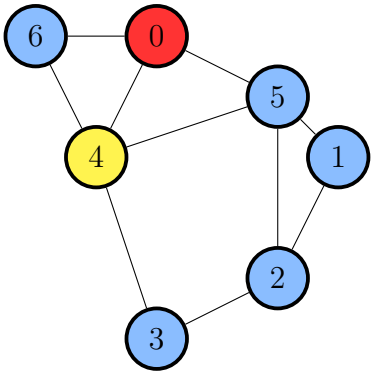
Fiind date un graf $G=(V,E)$ si un nod sursa s, aceasta parcurgere va permite explorarea sistematica in G si descoperirea fiecarui nod, plecand din s. Totodata, se poate calcula si distanta de la s la fiecare nod ce poate fi vizitat. In felul acesta, se contruieste un arbore „pe latime” cu radacina in 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.

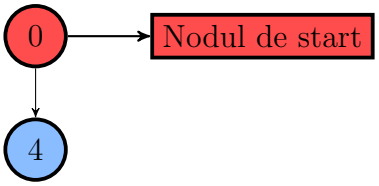
Pentru a intelege mai bine cum functioneaza acest algoritm, se ofera urmatorul exemplu de rulare al algoritmului:


6
0
5
4
1
2
3

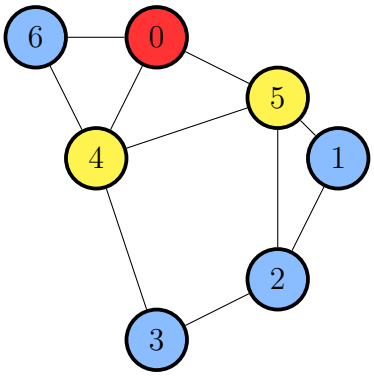

0
Nodul de start

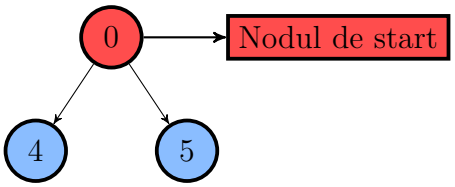
0


6
0
5
4
1
2
3

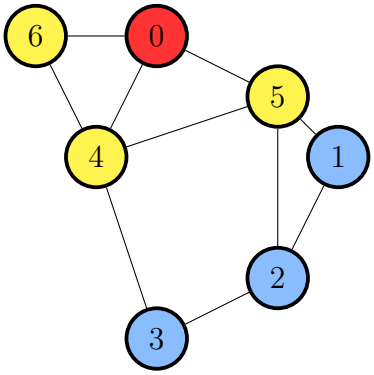

0
Nodul de start
4

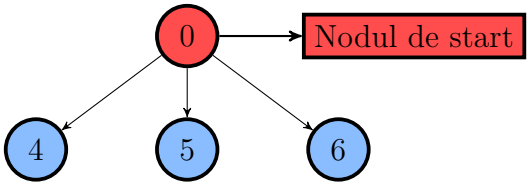
0 **4**


6
0
5
4
1
2
3

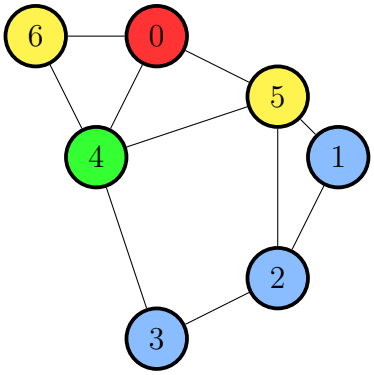

0
Nodul de start
4
5

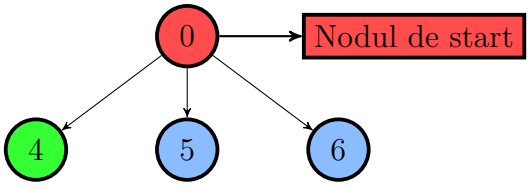
0 **4** **5**


6
0
5
4
1
2
3

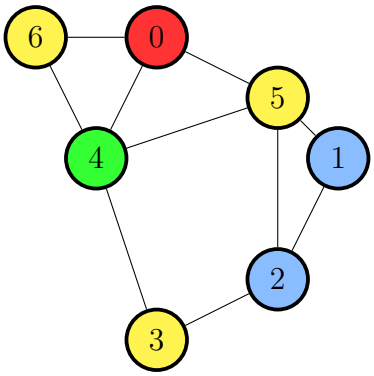

0
Nodul de start
4
5
6

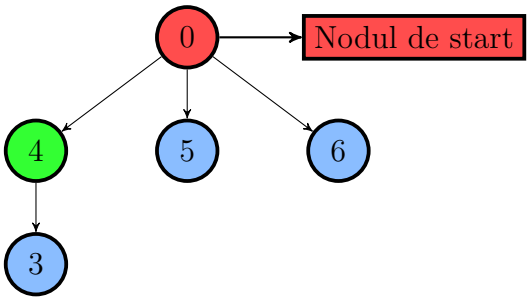

0
4
5
6


6
0
5
4
1
2
3

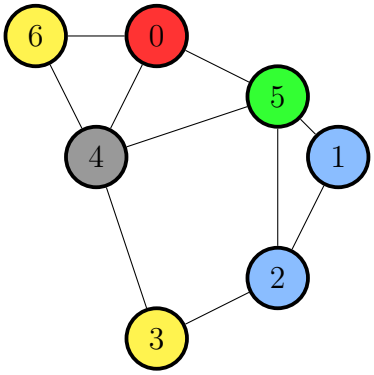

0
Nodul de start
4
5
6

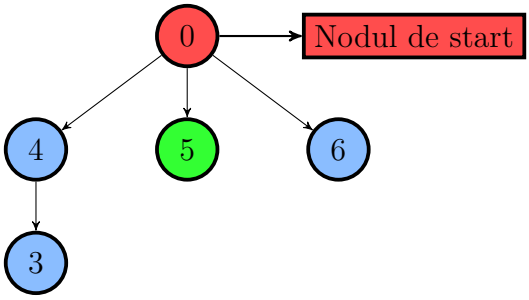

0
4
5
6


6
0
5
4
1
2
3

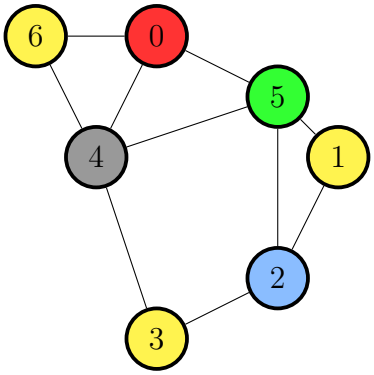

0
Nodul de start
4
5
6
3

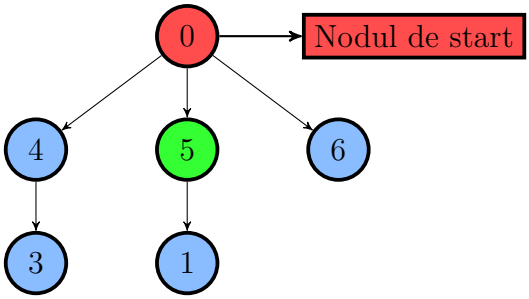

0
4
5
6
3

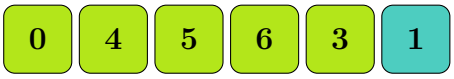

6
0
5
4
1
2
3

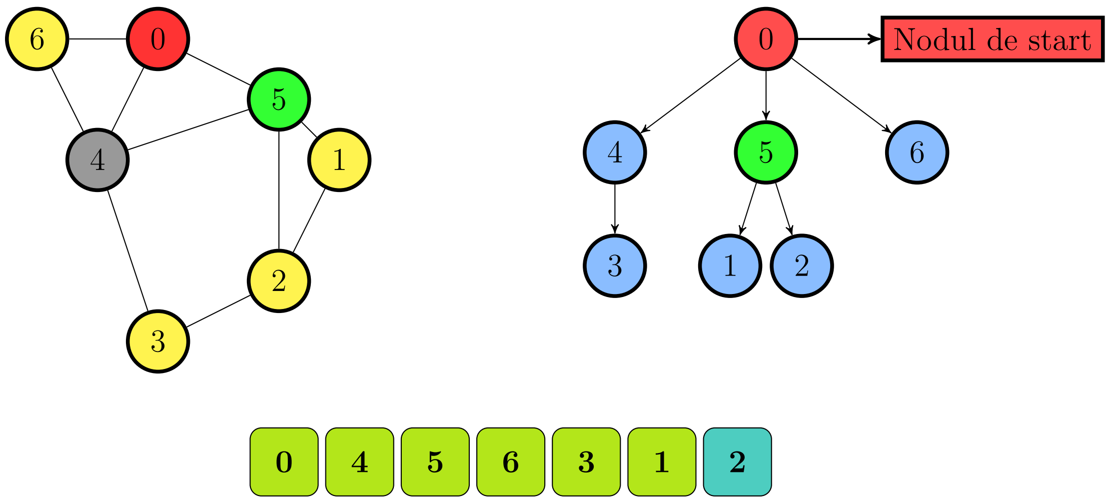

0
Nodul de start
4
5
6
3

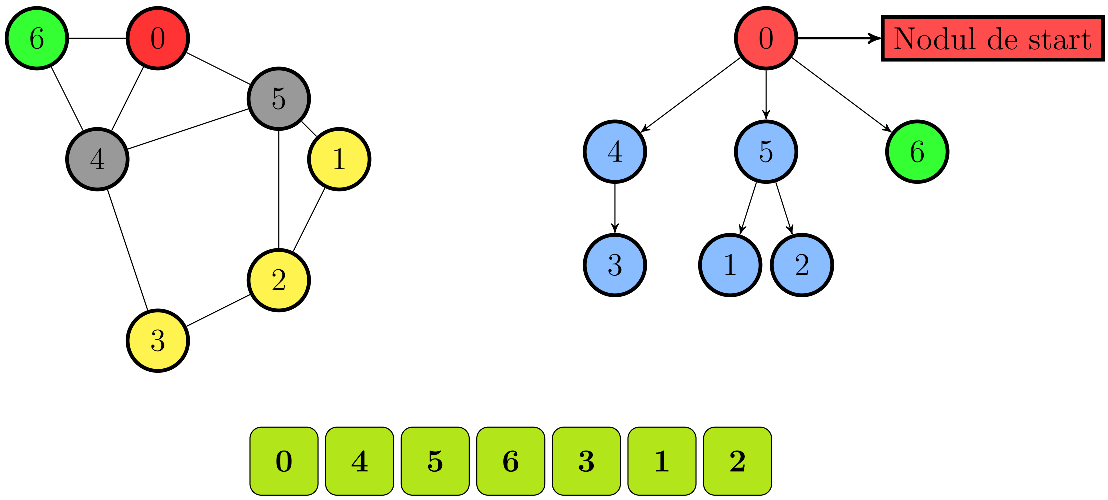

0
4
5
6
3

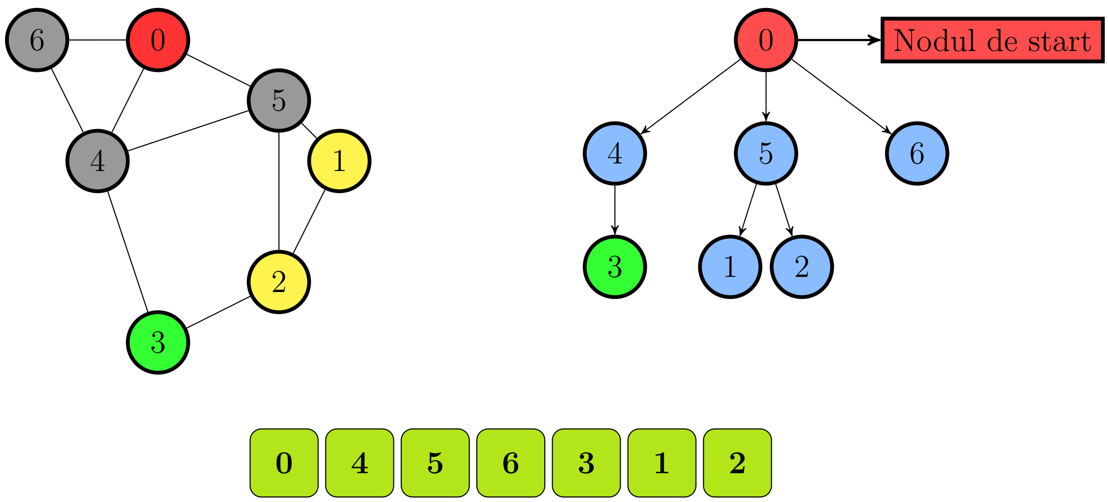

6
0
5
4
1
2
3

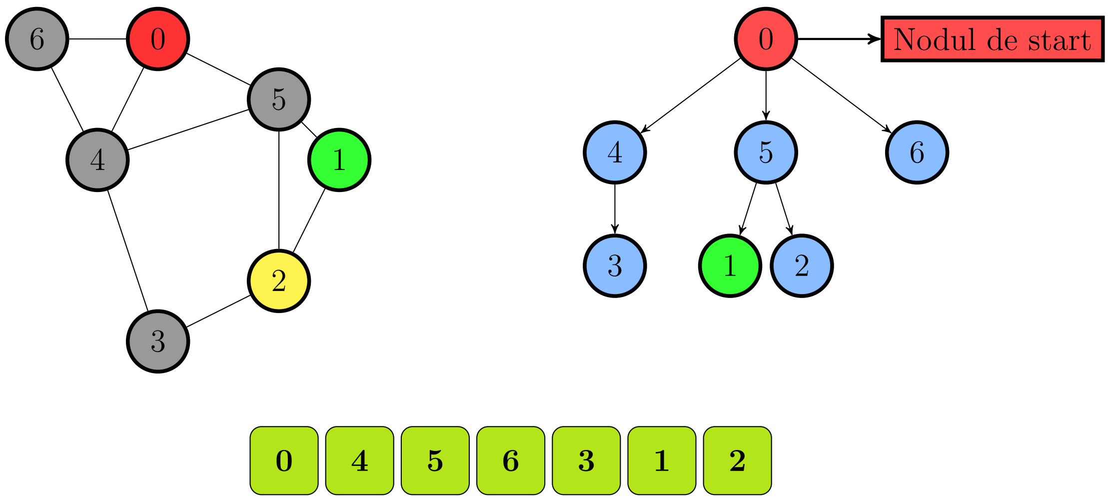

0
Nodul de start
4
5
6
3
1

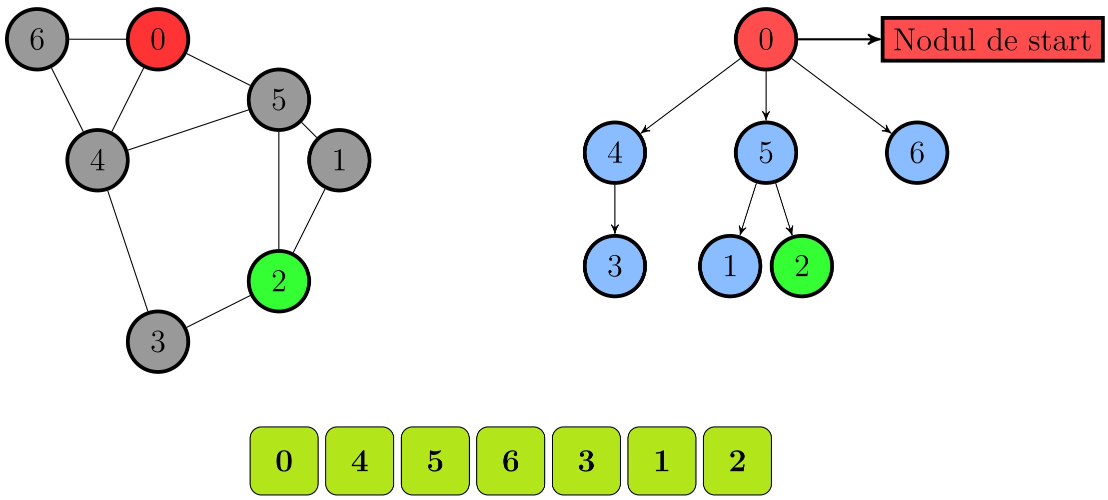

0
4
5
6
3
1

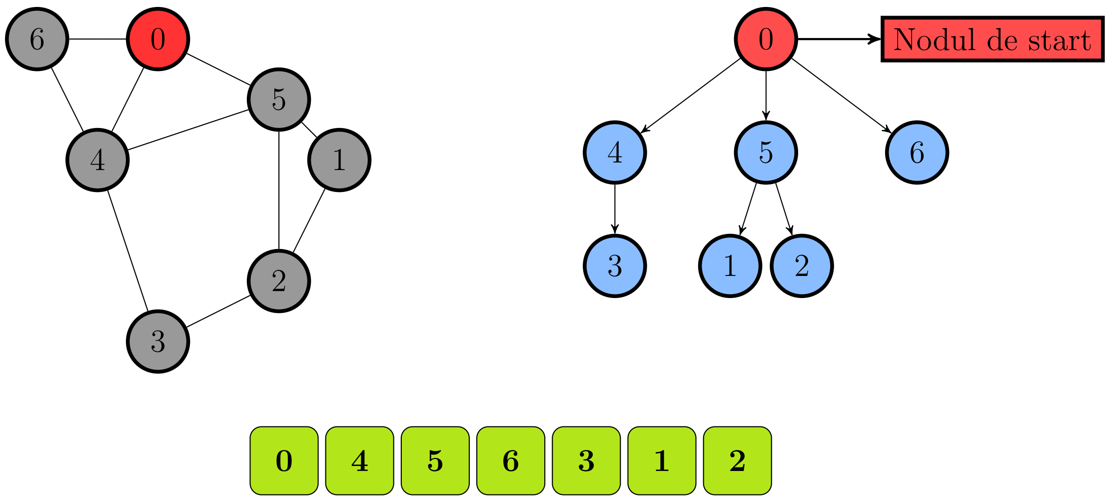

Nodul de start
0 4 5 6 3 1 2


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
```

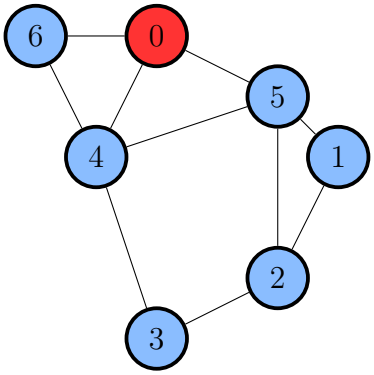
2.0.2 Parcurgerea in adancime

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

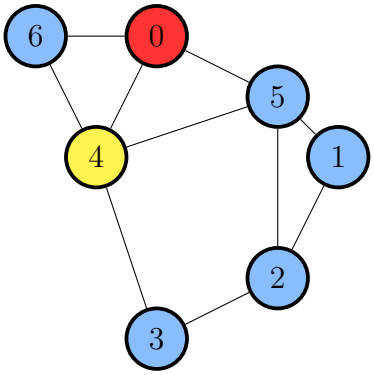
```
1: procedure DFS(G = {V, E}, start)
2:     S ← ∅
3:     S ← push(S, v)
4:     while !isEmpty(S) do
5:         u ← pop(S)
6:         if viz[u] = false then
7:             viz[u] ← true
8:             for each vertex x ∈ V\{u} do
9:                 if (u, x) ∈ E then
10:                    if viz[x]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

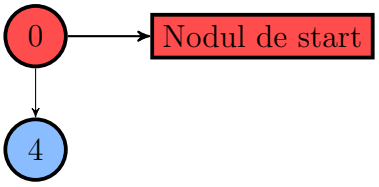
Pentru a intelege mai bine cum functioneaza acest algoritm prezentat, se ofera un exemplu de rulare pentru un graf:


6
0
5
4
1
2
3

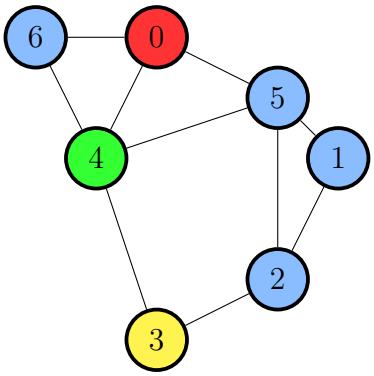

0
Nodul de start

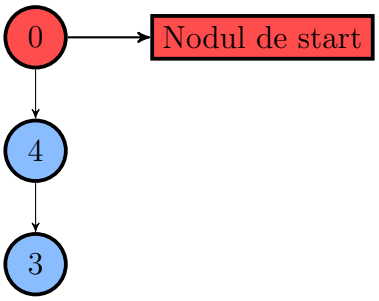

0


6
0
5
4
1
2
3

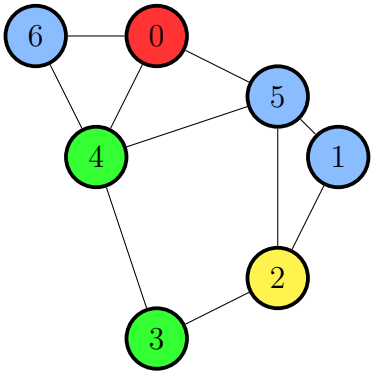

0
Nodul de start
4

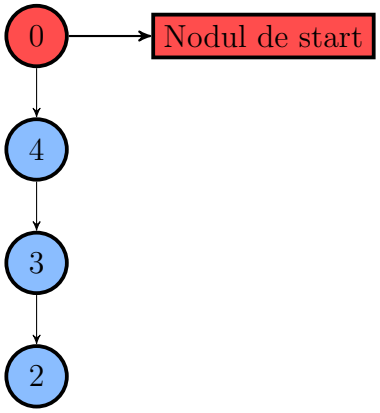

0
4

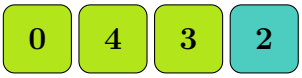

6
0
5
4
1
2
3

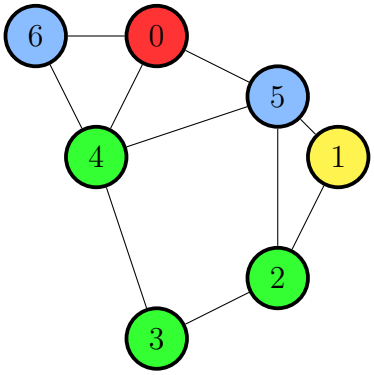

0
Nodul de start
4
3

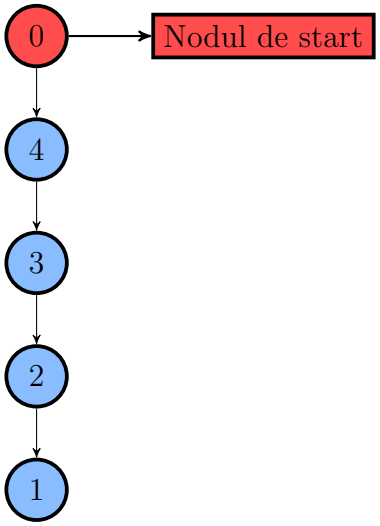

0
4
3


6
0
5
4
1
2
3

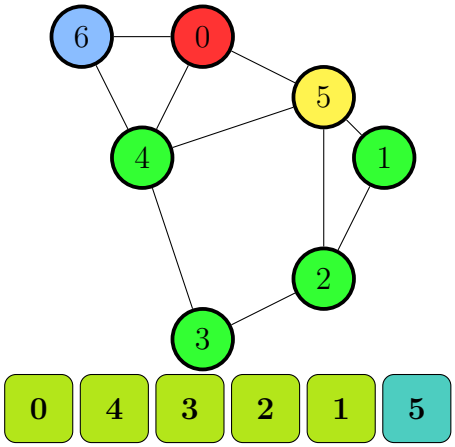

0
Nodul de start
4
3
2

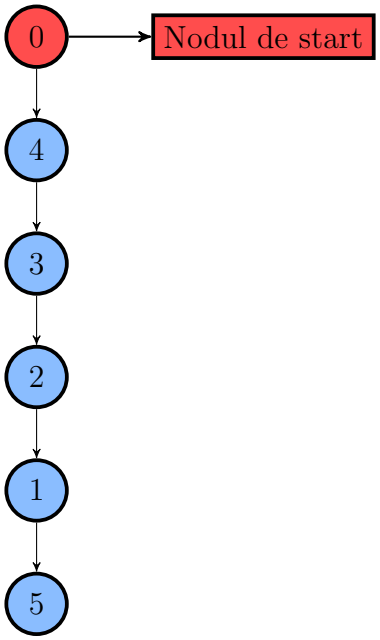

0
4
3
2

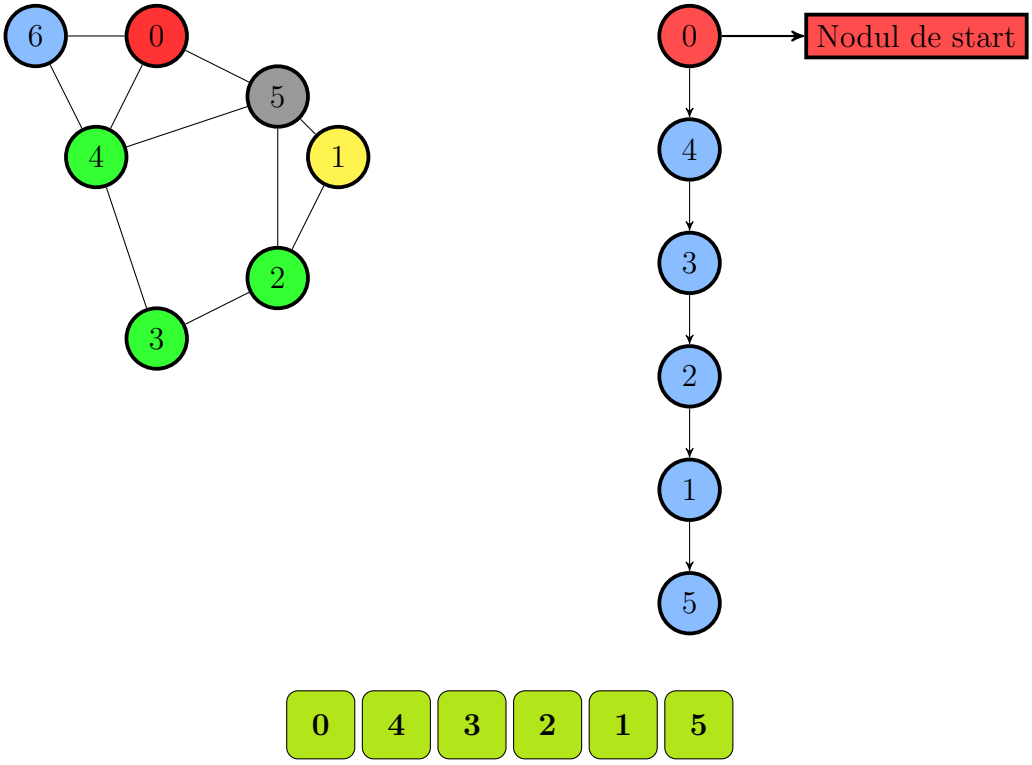

6
0
5
4
1
2
3


0
Nodul de start
4
3
2
1


0
4
3
2
1

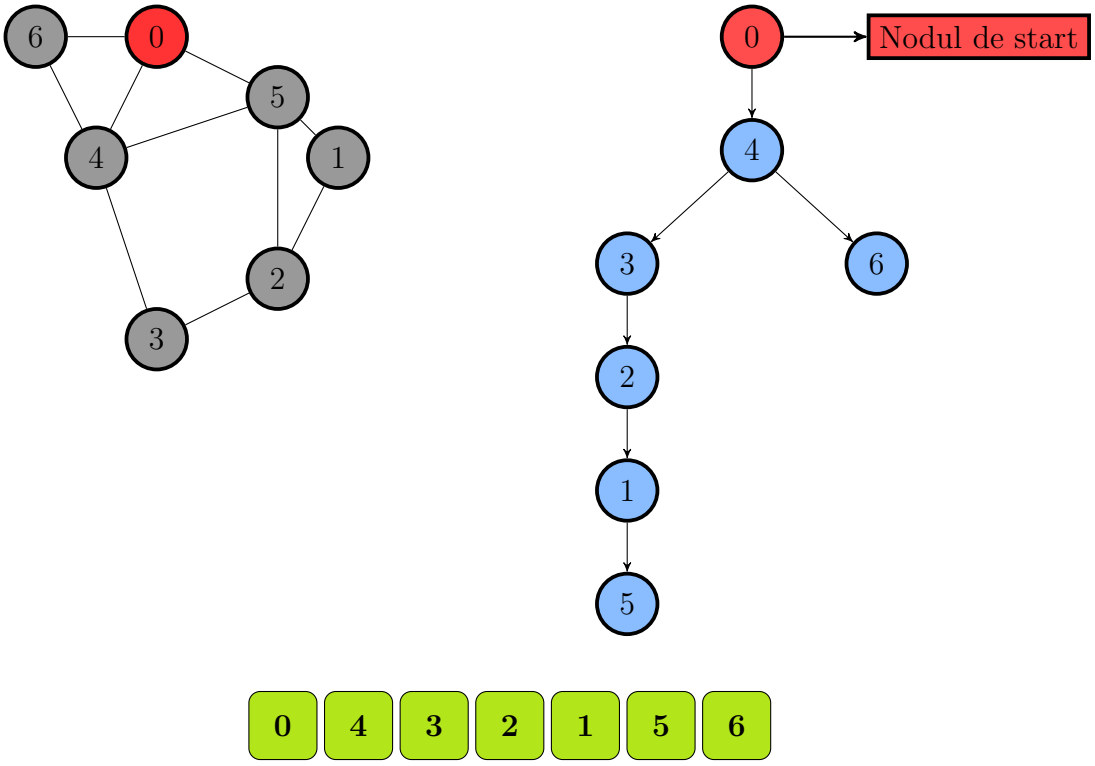

6
0
5
4
1
2
3
0
4
3
2
1
5


0
Nodul de start
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.

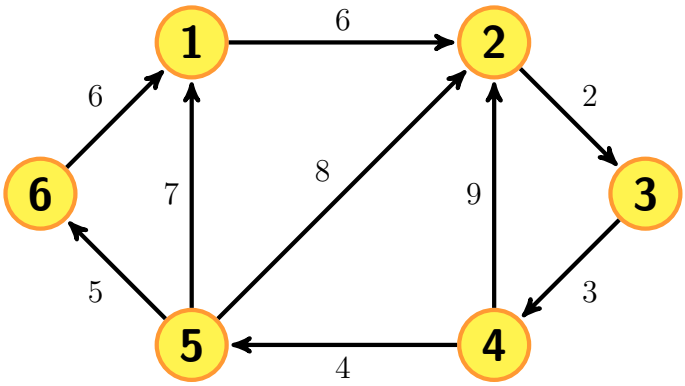
In figura de mai jos, este prezentat rezultatul final al algoritmului de parcurgere in adancime.



3 Grafuri orientate

Se numeste *graf orientat* o pereche ordonata de multimi $G = (V, E)$, unde V este o multime nevida de elemente numite *varfuri* sau *noduri*, iar E este o multime de perechi ordonate de elemente distincte din V numite *arce*.

Pentru un arc $e = (x, y) \in E$, spunem ca x si y sunt *adiacente*, x este *extremitatea initiala* a arcului e, y este *extremitatea finala*, x si y sunt *incidente* cu e, x este *predecesorul* lui y, iar y este *succesorul* lui x.



$V = \{1, 2, 3, 4, 5, 6\}$ si $E = \{(1, 2), (2, 3), (3, 4), (4, 2), (4, 5), (5, 1), (5, 2), (5, 6), (6, 1)\}$

Se numeste *gradul interior* al unui varf x, notandu-se cu $d_-(x)$, numarul arcelor de forma (y, x), $y \in V$ si $(y, x) \in E$.

Se numeste *gradul exterior* al unui varf x, notandu-se cu $d_+(x)$, numarul arcelor de forma (x, y), $y \in V$ si $(x, y) \in E$.

Definitie: Fie $G = (V, E)$ un graf orientat. Se numeste *lant* o succesiune de arce $L = [e_1, e_2, ...e_k]$ cu proprietatea ca oricare doua arce consecutive e_i, e_{i+1} au o extremitate comuna.

Observatie: Intr-un lant, orientarea arcelor nu este importanta.

Definitie: Fie $G = (V, E)$ un graf orientat. Se numeste *drum* o succesiune de varfuri $\mathbf{L} = [x_1, x_2, ...x_k]$ cu proprietatea (x_1, x_2), ... (x_{k-1}, x_k) sunt arce. Un drum in care toate varfurile sunt distincte se numeste *drum elementar*.

Definitie: Se numeste *circuit* un drum in care primul varf este identic cu ultimul, iar arcele nu se repeta. Un drum in care varfurile nu se repeta, cu exceptia primului si a ultimului se numeste *circuit elementar*.

Definitie: Se numeste *graf partial* pentru un graf orientat $G = (V, E)$ un graf $G' = (V, E')$ cu proprietatea ca $E' \in E$.

Definitie: Se numeste *subgraf* a lui G un graf $G' = (V', E')$, unde $V' \in V$ si E' contine toate arcele din E care au ambele extremitati in V'.

Definitie: Un graf orientat se numeste *complet* daca oricare doua varfuri distincte sunt adiacente.

Observatie: Spre deosebire de grafurile neorientate, unde pentru o multime de varfuri exista un singur graf complet, in cazul grafurilor orientate, pentru o multime de varfuri date putem avea mai multe grafuri complete.

Doua varfuri ***x*** si ***y*** sunt adiacente intr-un graf orientat in oricare din situatiile: exista arcul ***(x, y)***, arcul ***(y, x)*** sau arcele ***(x, y)*** si ***(y, x)***.

Sunt ***n(n-1)/2*** posibilitati de a alege doua varfuri distincte.

Pentru fiecare dintre acestea, exista 3 situatii. In concluzie, in total sunt $3^{n(n-1)/2}$ grafuri orientate complete cu ***n*** varfuri.

Definitie: Se numeste *graf turneu* un graf cu proprietatea ca intre oricare doua varfuri distincte exista exact un arc.

Proprietate: In orice graf turneu, exista un drum elementar care contine toate varfurile grafului.

3.1 Sortarea topologica

Dandu-se un graf orientat aciclic, *sortarea topologica* realizeaza o aranjare liniara a nodurilor in functie de muchiile dintre ele. Orientarea muchiilor corespunde unei relatii de ordine de la nodul sursa catre cel destinatie. Astfel, daca ***(u,v)*** este una dintre muchiile grafului, ***u*** trebuie sa apara inaintea lui ***v*** in insiruire. Daca graful ar fi *ciclic*, nu ar putea exista o astfel de insiruire (nu se poate stabili o ordine intre nodurile care alcatuiesc un ciclu).

Sortarea topologica poate fi vazuta si ca plasarea nodurilor de-a lungul unei linii orizontale astfel incat toate muchiile sa fie directionate de la stanga la dreapta.

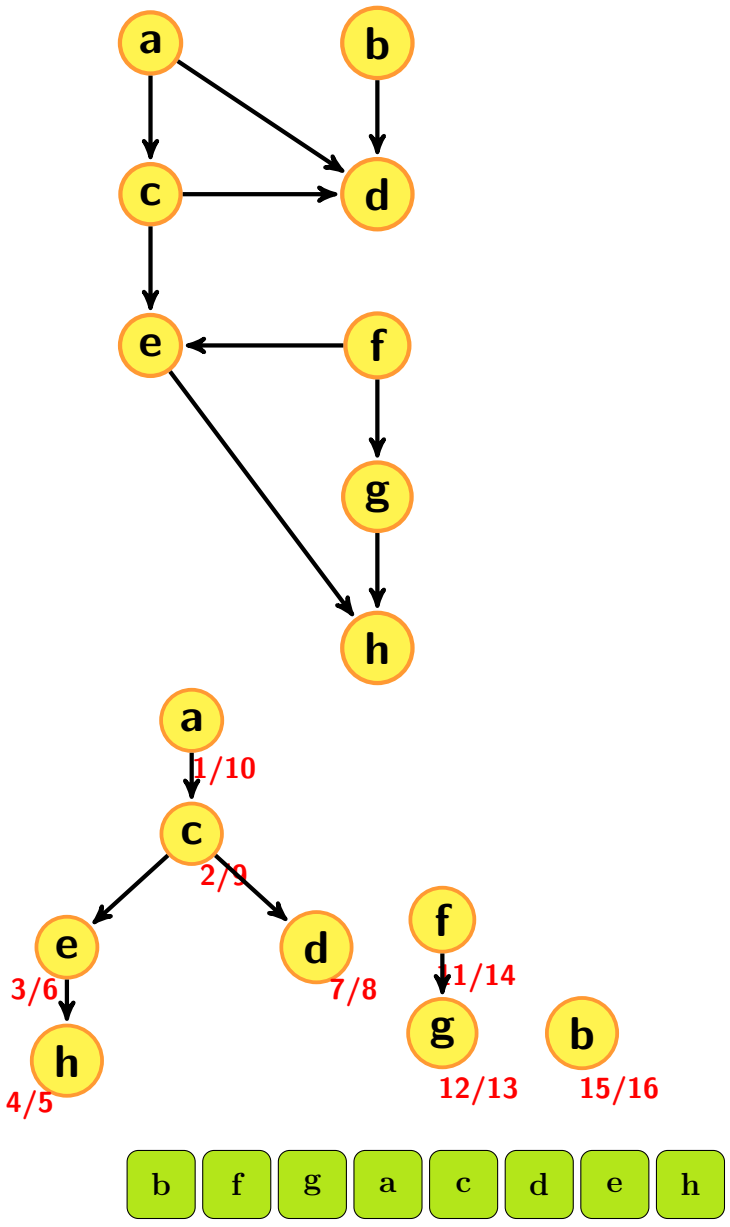
3.1.1 Algoritmul lui Kahn

Pseudocod

```
TopSort(G) {
    V = noduri(G)
    L = vida; // lista care va contine elementele sortate
    foreach (u ∈ V) {
        if (u nu are in-muchii)
          S = S + u; //noduri care nu au in-muchii
    }
    while (!empty(S)) { // cat timp mai am noduri de prelucrat
        u = random(S); // se scoate un nod din multimea S
        L = L + u;  // adaug U la lista finala
        foreach v ∈ succs(u) { // pentru toti vecinii
            sterge u-v; // sterge muchia u-v
            if (v nu are in-muchii)
              S = S + v;  // adauga v la multimea S
        } // close foreach
    } //close while
    if (G are muchii)
        print(eroare);  // graf ciclic
    else
        print(L); // ordinea topologica
}
```

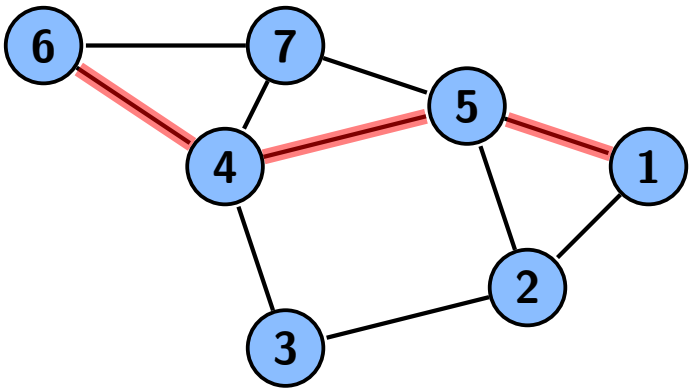
Observatie

Putem implementa sortarea topologica utilizand o parcurgere DFS a grafului pentru determinarea timpilor de vizitare si finalizare. In acest caz, sortarea topologica a grafului se rezuma la sortare descrescatoare a nodurilor in functie de timpul de finalizare.



4 Probleme propuse

1. 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.
2. Sa se defineasca o functie recursiva pentru parcurgerea in adancime a unui graf. Comparati rezultatele cu cele ale functiei iterative.
3. 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. Trebuie sa puteti reconstrui si acest drum de cost minim, specificand calea care trebuie parcursa de la nodul sursa (cel din care s-a pornit parcurgerea) pana la un nod destinatie.



4. Implementati o functie care sorteaza topologic nodurile dintr-un graf orientat (functia va verifica si daca graful este sau nu aciclic). Puteti implementa algoritmul lui Kahn sau puteti modifica algoritmul de parcurgere in adancime (DFS), retinand pentru fiecare nod un tip de descoperire si unul de finalizare.

5 Probleme suplimentare

1. In ultima ecranizare a celebrei piese shakespeariene Romeo si Julieta traiesc intr-un oras modern, comunica prin e-mail si chiar invata sa programeze. Intr-o secventa tulburatoare sunt prezentate framantarile interioare ale celor doi eroi incercand fara succes sa scrie un program care sa determine un punct optim de intalnire.

   Ei au analizat harta orasului si au reprezentat-o sub forma unei matrice cu n linii si m coloane, in matrice fiind marcate cu spatiu zonele prin care se poate trece (strazi lipsite de pericole) si cu 'X' zonele prin care nu se poate trece. De asemenea, in matrice au marcat cu 'R' locul in care se afla locuinta lui Romeo, iar cu 'J' locul in care se afla locuinta Julietei.

   Ei se pot deplasa numai prin zonele care sunt marcate cu spatiu, din pozitia curenta in oricare dintre cele 8 pozitii invecinate (pe orizontala, verticala sau diagonale). Cum lui Romeo nu ii place sa astepte si nici sa se lase asteptat, ei au hotarat ca trebuie sa aleaga un punct de intalnire in care atat Romeo, cat si Julieta, sa poata ajunge in acelasi timp, plecand de acasa. Fiindca la intalniri amandoi vin intr-un suflet, ei estimeaza timpul necesar pentru a ajunge la intalnire prin numarul de elemente din matrice care constituie drumul cel mai scurt de acasa pana la punctul de intalnire. Si cum probabil exista mai multe puncte de intalnire posibile, ei vor sa il aleaga pe cel in care timpul necesar pentru a ajunge la punctul de intalnire este minim.

   Sursa: infoarena

   Exemplu

```
5 8
XXR  XXX
 X  X  X
J X X  X
      XX
XXX XXXX
```

   Explicatie

   Traseul lui Romeo poate fi: `(1, 3), (2, 4), (3, 4), (4, 4)`. Asadar, timpul necesar lui Romeo pentru a ajunge de acasa la punctul de intalnire este 4.

   Traseul Julietei poate fi: `(3, 1), (4, 2), (4, 3), (4, 4)`. Timpul necesar Julietei pentru a ajunge de acasa la punctul de intalnire este, de asemenea, 4. In plus `(4, 4)` este punctul cel mai apropiat de ei cu aceasta proprietate.

   Rezolvare

   Pornim alternativ din pozitiile lui Romeo si a Julietei. La fiecare pas `K` generam in doua cozi separate punctele din matrice aflate la distanta `K` de pozitiile initiale pentru Romeo si Julieta.

   Comparam cele doua liste de puncte. Daca gãsim un punct comun algoritmul se incheie, daca nu, generam punctele aflate la distante `K + 1` s.a.m.d.

   In literatura de specialitate, algoritmul folosit se numeste parcugere in latime, iar in algoritmica romaneasca este conuscut ca si algoritmul lui Lee.

   Algoritmul are la baza o metoda iterativa. Fiecare vecin va fi bagat, iar la extragerea

unuia din coada vom baga in coada toti vecinii nevizitati ai nodului curent (avand grija sa eliminam nodul curent). Algoritmul se repeta pana cand coada devine vida sau gasim solutia asteptata.

2. Parcul orasului a fost neglijat mult timp, astfel ca acum toate aleile sunt distruse. Prin urmare, anul acesta Primaria si-a propus sa faca reamenajari. Parcul are forma unui patrat cu latura de N metri si este inconjurat de un gard care are exact doua porti. Proiectantii de la Primarie au realizat o harta a parcului si au trasat pe harta un caroiaj care imparte parcul in N*N zone patrate cu latura de 1 metru. Astfel harta parcului are aspectul unei matrice patratice cu N linii si N coloane. Liniile si, respectiv, coloanele sunt numerotate de la 1 la N. Elementele matricei corespund zonelor patrate de latura 1 metru. O astfel de zona poate sa contina un copac sau este libera. Edilii orasului doresc sa paveze cu un numar minim de dale patrate cu latura de 1 metru zonele libere (fara copaci) ale parcului, astfel incat sa se obtina o alee continua de la o poarta la alta.

   Sursa: infoarena

   Exemplu

```
8 6
2 7
3 3
4 6
5 4
7 3
7 5
1 1 8 8
```

   Explicatie

   O modalitate prin care se poate construi aleea cu un numar minim de dale este (cu X sunt marcati copacii, cu _ zonele libere, iar cu O dalele aleii):

```
O O O _ _ _ _ _
_ _ O O _ _ X _
_ _ X O _ _ _ _
_ _ _ O O X _ _
_ _ _ X O _ _ _
_ _ _ _ O O _ _
_ _ X _ X O O _
_ _ _ _ _ _ O O
```

   Rezolvare

   Este o problemă *clasica* a carei rezolvare se bazeaza pe algoritmul lui Lee.

   Vom utiliza o matrice A in care initial vom retine valoarea -2 pentru zonele libere, respectiv valoarea -1 pentru zonele in care se afla un copac.

   Ulterior, pe masura ce exploram matricea in scopul determinarii unei alei de lungime minima vom retine in matrice pentru pozitiile explorate un numar natural care reprezinta distanta minima de la pozitia primei porti la pozitia respectiva (exprimata in numarul de dale necesare pentru construirea aleei).

   Vom parcurge matricea incepand cu pozitia primei porti.

Pozitiile din matrice care au fost explorate le vom retine intr-o coada, in ordinea explorarii lor.

La fiecare pas vom extrage din coada o pozitie (x,y), vom explora toti cei 4 vecini liberi neexplorati ai pozitiei extrase si ii vom introduce in coada. Cand exploram un vecin, retinem in matricea A pe pozitia sa distanta minima $(1 + A[x][y])$.

Procedeul se repeta cat timp coada nu este vida si pozitia in care se afla cea de a doua poarta nu a fost explorată.

Rezultatul se va obtine in matricea A, pe pozitia celei de a doua porti.