

# Proiectarea Algoritmilor 2009-2010

---

## Laborator 6

# Parcurgerea Grafurilor. Sortare Topologica

---

## 1. Obiective laborator

- Intelegerea conceptului de graf
- Cunoasterea modurilor de parcurgere a grafurilor

## 2. Importanta - Aplicatii practice

Grafurile sunt necesare in numeroase aplicatii practice:

- Retele de calculatoare (ex: stabilirea unei topologii fara bucle)
- Pagini Web (ex: Google PageRank [1] )
- Retele sociale (ex: calcul radialitate [2] )
- Harti cu drumuri (ex: drum minim)
- Modelare grafica (ex: prefuse [3], graph-cut [4] )

## 3. Descrierea problemei si a rezolvarilor

Graful este o pereche de multimi  $G = (V, E)$ . Multimea  $V$  contine nodurile grafului (vertices), iar multimea  $E$  contine muchiile (edges), fiecare muchie stabilind o relatie de vecinatate intre doua noduri. O mare varietate de probleme se modeleaza folosind grafuri, iar rezolvarea presupune explorarea spatiului, adica trebuie vizitate varfurile sau muchiile. O parcurgere isi propune sa ia in discutie fiecare nod al grafului, exact o singura data, pornind de la un nod ales, numit in continuare nod sursa.

Reprezentarea in memorie a grafurilor se face, de obicei, cu liste de adiacenta sau cu matrice de adiacenta. Se pot folosi inasa si alte structuri de date, de exemplu un map de perechi  $\langle\langle\text{sursa,destinatie}\rangle,\text{cost}\rangle$ .

Pe parcursul rularii algoritmilor de parcurgere, un nod poate avea 3 culori:

- Alb = nedescoperit
- Gri = a fost descoperit si este in curs de procesare
- Negru = a fost procesat

Se poate face o analogie cu o pata neagra care se extinde pe un spatiu alb. Nodurile gri se afla pe frontiera petei negre.

Algoritmii de parcurgere pot fi caracterizati prin proprietatile completitudine si optimalitate. Un algoritm de explorare complet va descoperi o solutie, daca problema accepta solutie. Un algoritm de explorare optimal va descoperi solutia optima a problemei.

### 3.1. Parcurgerea in lățime - BFS

Parcurgerea in latime (Breadth-first Search - BFS) este un algoritm de cautare in graf, in care, atunci cand se ajunge intr-un nod oarecare,  $v$ , nevizitat, se viziteaza toate nodurile nevizitate adiacente lui  $v$ , apoi toate varfurile nevizitate adiacente varfurilor adiacente lui  $v$ , etc.

**Atentie!** BFS depinde de nodul de start. Plecand dintr-un nod se va parcurge doar componenta conexa din care acesta face parte. Pentru grafuri cu mai multe componente conexe se vor obtine mai multi arbori de acoperire.

In urma aplicarii algoritmului BFS asupra fiecarei componente conexe a grafului, se obtine un arbore de acoperire (prin eliminarea muchiilor pe care nu le folosim la parcurgere). Pentru a putea reconstitui acest arbore, se pastreaza pentru fiecare nod dat identitatea parintelui sau. In cazul in care nu exista o functie de cost asociata muchiilor, BFS va determina si drumurile minime de la radacina la oricare nod.

Pentru implementarea BFS se foloseste o coada. In momentul adaugarii in coada, un nod trebuie colorat gri (a fost descoperit si urmeaza sa fie prelucrat).

Algoritmul de explorare BFS este complet si optimal.

#### Algoritm:

```
BFS(s, G) {  
    foreach( $u \in V$ )  
         $p(u) = \text{null}$ ; // initializari  
     $\text{dist}(s, u) = \text{inf}$ ;  
     $c(u) = \text{alb}$ ;  
     $\text{dist}(s) = 0$ ; // distanta pana la sursa este 0  
     $c(s) = \text{gri}$ ; //incepem prelucrarea nodului, deci culoarea devine gri  
     $Q = ()$ ; //se foloseste o coada in care retinem nodurile de prelucrat  
     $Q = Q + s$ ; // adaugam sursa in coada  
    while ( !empty(Q) ) { // cat timp mai am noduri de prelucrat  
         $u = \text{top}(Q)$ ; // se determina nodul din varful cozii  
        foreach  $v \in \text{succs}(u)$  { // pentru toti vecinii
```

```
    if (c(v) = alb) { // nodul nu a mai fost gasit, nu e in coada
        // actualizam structura date
        dist(v) = dist(u) + 1;
        p(v) = u;
        c(v) = gri;
        Q = Q + v;
    } // close if
} // close foreach
c(u) = negru; //am terminat de prelucrat nodul curent
Q = Q - u; //nodul este eliminat din coada
} //close while
}
```

**Complexitate:**

- cu lista:  $O(|E|+|V|)$
- cu matrice:  $O(|V|^2)$

### 3.2. Parcurgerea in adancime – DFS

Parcurgerea in adancime (Depth-First Search - DFS) porneste de la un nod dat (nod de start), care este marcat ca fiind in curs de procesare. Se alege primul vecin nevizitat al acestui nod, se marcheaza si acesta ca fiind in curs de procesare, apoi si pentru acest vecin se cauta primul vecin nevizitat, si asa mai departe. In momentul in care nodul curent nu mai are vecini nevizitati, se marcheaza ca fiind deja procesat si se revine la nodul anterior. Pentru acest nod se cauta primul vecin nevizitat. Algoritmul se repeta pana cand toate nodurile grafului au fost procesate.

In urma aplicarii algoritmului DFS asupra fiecărei componente conexe a grafului, se obtine pentru fiecare dintre acestea cate un arbore de acoperire (prin eliminarea muchiilor pe care nu le folosim la parcurgere). Pentru a putea reconstitui acest arbore, pastram pentru fiecare nod dat identitatea parintelui sau.

**Pentru fiecare nod se vor retine:**

- timpul descoperirii
- timpul finalizarii
- parintele
- culoarea

Algoritmul de explorare DFS nu este nici complet (cautare pe un subarbore infinit) nici optimal (nu gaseste nodul cu adancimea minima).

Spre deosebire de BFS, pentru implementarea DFS se foloseste o stiva (abordare LIFO in loc de FIFO). Desi se poate face aceasta inlocuire in algoritmul de mai sus, de cele mai multe ori este mai intuitiva folosirea recursivitatii.

**Algoritm:**

```
DFS(G) {  
    V = noduri(G)  
    foreach (u ∈ V) {  
        // initializare structura date  
        c(u) = alb;  
        p(u)=null;  
    }  
    timp = 0; // retine distanta de la radacina pana la nodul curent  
    foreach (u ∈ V)  
        if (c(u) = alb) explorare(u); // explorez nodul  
}  
  
explorare(u) {  
    d(u) = timp++; // timpul de descoperire al nodului u  
    c(u) = gri; // nod in curs de explorare  
    foreach (v ∈ succes(u)) // incerc sa prelucrez vecinii  
        if (c(v) = alb) { // daca nu au fost prelucrati deja  
            p(v) = u;  
            explorare(v);  
        }  
    (u) = negru; // am terminat de prelucrat nodul curent  
    f(u) = timp++; // timpul de finalizare al nodului u  
}
```

**Complexitate:**

- cu lista:  $O(|E|+|V|)$
- cu matrice:  $O(|V|^2)$

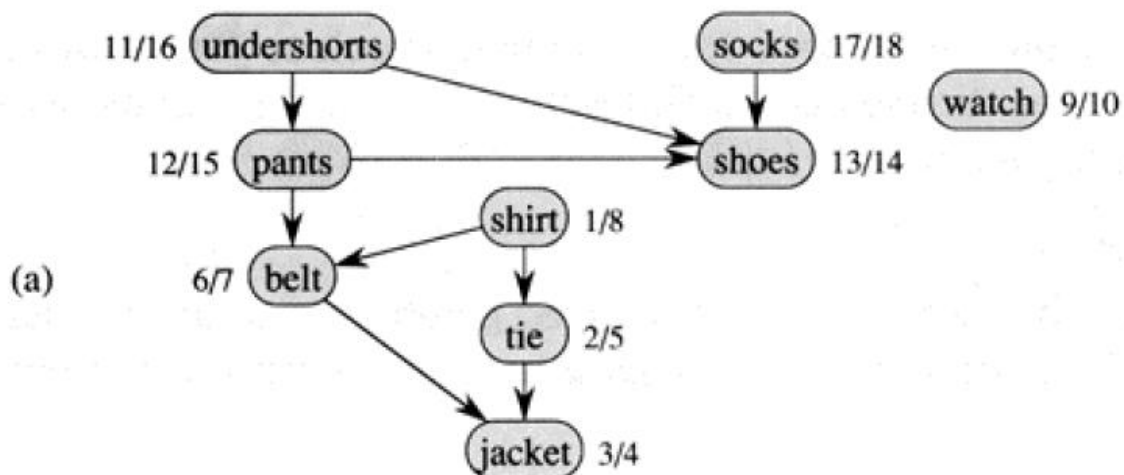
### 3.3. Sortarea Topologica

Dandu-se un graf orientat aciclic, sortarea topologica realizeaza o aranjare liniara a nodurilor in functie de muchiiile dintre ele. Orientarea muchiiilor corespunde unei relatii de ordine de la nodul sursa catre cel destinatie. Astfel, daca  $(u,v)$  este una dintre muchiiile 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).

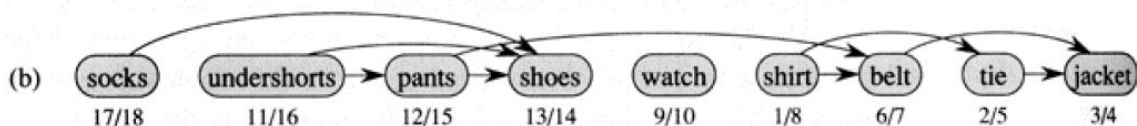
Sortarea topologica poate fi vazuta si ca plasarea nodurilor de-a lungul unei linii orizontale astfel incat toate muchiiile sa fie directionate de la stanga la dreapta.

#### Exemplu:

Profesorul Bumstead isi sorteaza topologic hainele inainte de a se imbraca.



(a) Fiecare muche  $(u,v)$  inseamna ca obiectul de imbracaminte  $u$  trebuie imbracat inaintea obiectului de imbracaminte  $v$ . Timpul de descoperire  $d(u)$  si de finalizare  $f(u)$  obtinuti in urma parcurgerii DFS sunt notati langa noduri.



(b) Acelasi graf, sortat topologic. Nodurile lui sunt aranjate de la stanga la dreapta in ordinea descrescatoare a  $f(u)$ . Observati ca toate muchiiile sunt orientate de la stanga la dreapta. Acum profesorul Bumstead se poate imbraca linistit.

#### Algoritm:

Sunt doi algoritmi cunoscuti pentru sortarea topologica.

Algoritmul bazat pe DFS:

- parcurgere DFS pentru determinarea timpilor
- sortare descrescatoare in functie de timpul de finalizare

Un alt algoritm este cel descris de Kahn:

```
TopSort(G) {  
    V = noduri(G)  
    L = vida; // lista care va contine elementele sortate  
    foreach (u ∈ V) {  
        // initializare S cu nodurile care nu au in-muchii  
        if (u nu are in-muchii)  
            S = S + u;  
    }  
    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  
}
```

**Complexitate optima:**  $O(|E|+|V|)$

## 4. Concluzii si observatii

Grafurile sunt foarte importante pentru reprezentarea si rezolvarea unei multitudini de probleme.

Cele mai uzuale moduri de reprezentare a unui graf sunt:

- liste de adiacenta
- matrice de adiacenta

Cele doua moduri uzuale de parcurgere neinformata a unui graf sunt:

- BFS – parcurgere in latime
- DFS – parcurgere in adancime

Sortarea topologica este o modalitate de aranjare a nodurilor in functie de muchiile dintre ele. In functie de nodul de start al DFS, se pot obtine sortari diferite, pastrand insa proprietatile generale ale sortarii topologice.

## 5. Referinte

- [1] <http://en.wikipedia.org/wiki/PageRank>
- [2] [http://en.wikipedia.org/wiki/Social\\_network#Social\\_network\\_analysis](http://en.wikipedia.org/wiki/Social_network#Social_network_analysis)
- [3] <http://prefuse.org/>
- [4] [http://classes.engr.oregonstate.edu/eecs/spring2008/cs419/Lectures/jun\\_graphcut.pdf](http://classes.engr.oregonstate.edu/eecs/spring2008/cs419/Lectures/jun_graphcut.pdf)
- [5] [http://en.wikipedia.org/wiki/Breadth-first\\_search](http://en.wikipedia.org/wiki/Breadth-first_search)
- [6] [http://en.wikipedia.org/wiki/Depth-first\\_search](http://en.wikipedia.org/wiki/Depth-first_search)
- [7] [http://en.wikipedia.org/wiki/Topological\\_sorting](http://en.wikipedia.org/wiki/Topological_sorting)
- [8] Introducere in Algoritmi, T. Cormen s.a., pag 403-419
- [9] <http://ww3.algorithmdesign.net/handouts/DFS.pdf>
- [10] <http://ww3.algorithmdesign.net/handouts/BFS.pdf>

**Responsabil laborator: Ciprian Coman ([ciprianmihai.coman@gmail.com](mailto:ciprianmihai.coman@gmail.com))**

# Proiectarea Algoritmilor 2009-2010

---

## *Laborator 6*

# Parcurgerea Grafurilor. Sortare Topologica – aplicații

---

- 1) Fiind dat un graf neorientat si 2 noduri, **start** si **finish**, sa se gaseasca cel mai scurt drum de la start la finish. Se vor afisa nodurile aflate pe drumul respectiv.
- 2) Realizati parcurgerea DFS a unui graf neorientat si afisati timpii de descoperire si de final pentru toate nodurile.
- 3) Folosind implementarea de la 2, realizati sortarea topologica pentru un graf **orientat**.
- 4) (BONUS) Implementi sortarea topologica folosind algoritmul lui Kahn.