Laborator 6: Parcurgerea Grafurilor. Sortare Topologica

Obiective laborator

- Intelegerea conceptului de graf si a modurilor de parcurgere aferente
- Utilitatea si aplicabilitatea sortarii topologice

Importanţă – aplicaţii practice

Grafurile sunt utile pentru a modela diverse probleme si se regasesc implementati in multiple aplicatii practice:

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

Descrierea problemei și a rezolvărilor

Graful poate fi modelat drept 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 acestora presupune explorarea spatiului. 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 insa si alte structuri de date, de exemplu un map de perechi < <sursa,destinatie>,cost> .

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

- Alb = nedescoperit
- Gri = a fost descoperit si este in curs de prcesare
- 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 completitudine si optimalitate. Un algoritm de explorare complet va descoperi intotdeauna o solutie, daca problema accepta solutie. Un algoritm de explorare optimal va descoperi solutia optima a problemei din perspectiva numarului de pasi care trebuie efectuati.

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

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 fiecarei 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 (in cazul unei cautaro 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 recusivitatii.

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);
        }
    c(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)

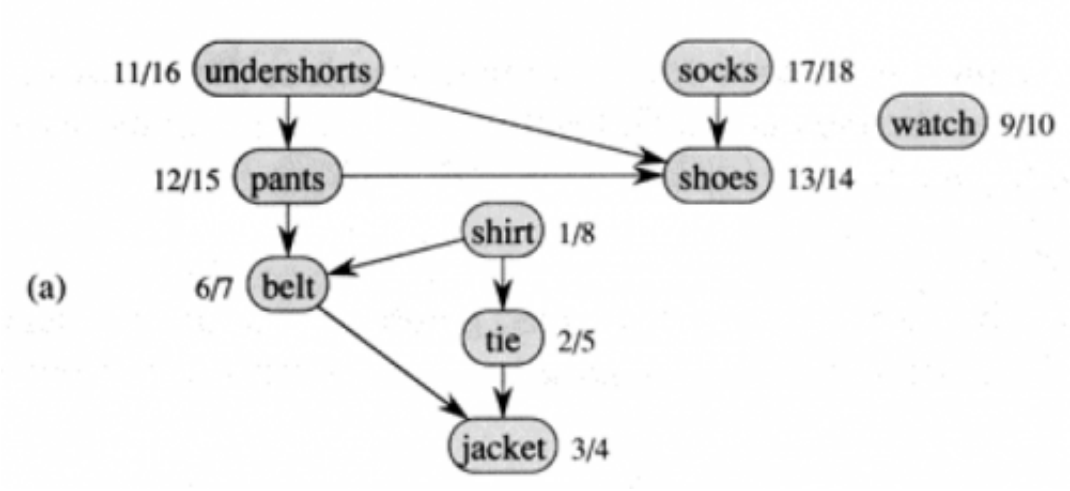
Sortarea Topologica

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).

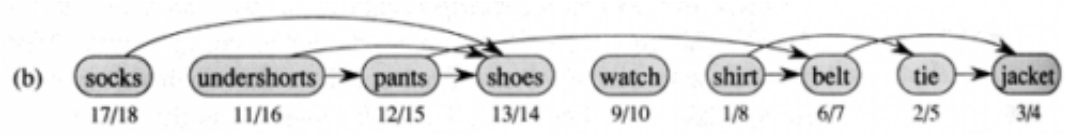
Sortarea topologica poate fi vazuta si ca plasarea nodurilor de-a lungul unei linii orizontale astfel incat toate muchiile sa fie directionate de la stanga la dreapta.

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. Timpii 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 muchiile sunt orientate de la stanga la dreapta. Acum profesorul Bumstead se poate imbraca linistit.

Algoritm:

Sunt doi algoritmi cunoscuti pentru sortarea topologina.

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
    // initializare S cu nodurile care nu au in-muchii
    foreach (u ∈ V) {
    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|)

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.

Referinte

[1] http://en.wikipedia.org/wiki/PageRank [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/ [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 [http://ww3.algorithmdesign.net/handouts/DFS.pdf]

[10] http://ww3.algorithmdesign.net/handouts/BFS.pdf [http://ww3.algorithmdesign.net/handouts/BFS.pdf]

Probleme

1. Maze (4p)

Asistentii de la PA au ascuns o comoara la capatul unui labirint(Maze). Voi va gasiti la intrarea in labirint, iar acesta nu are decat o singura iesire.

Pentru a recupera intreaga comoara va trebui sa gasiti iesirea din labirint prin 2 metode:

- DFS [2p]
- BFS [2p]

La final va trebuie sa respundeti la un quest: *Care din cele 2 metode a fost mai eficienta si de ce ?*

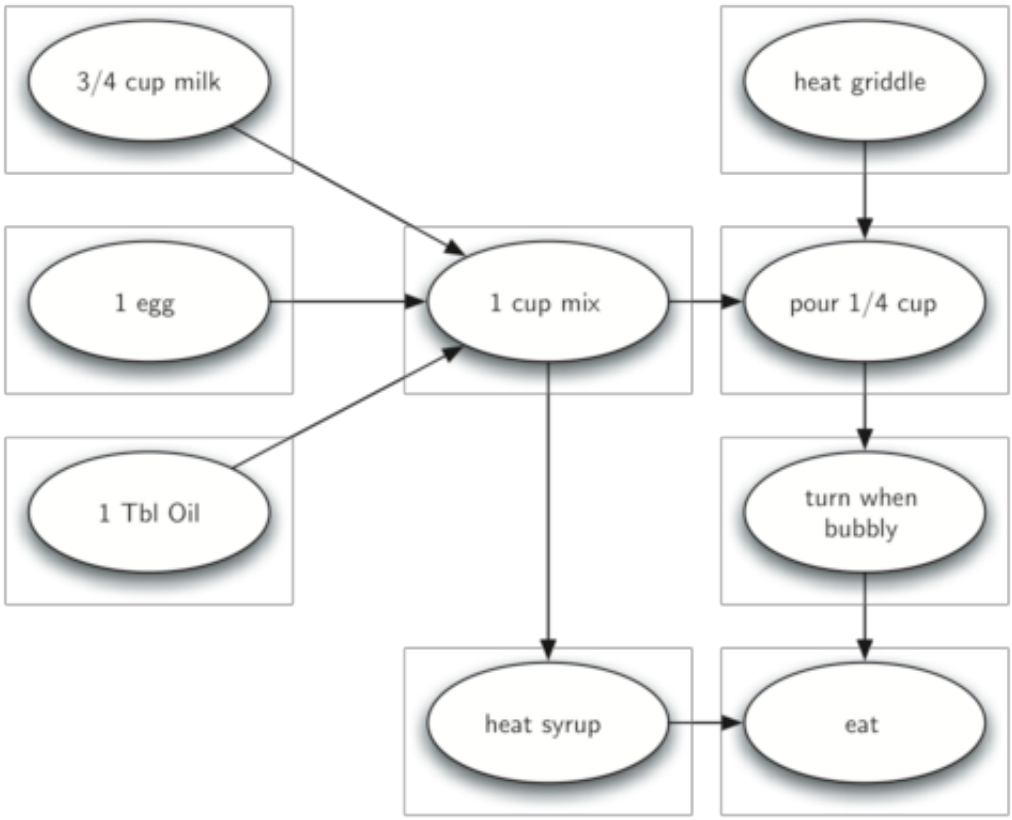
- Puteti folosii in test fix de test setand variabila random = false

2. Clatite (6p)

Acum ca aveti comoara si sunteti bogati, v-ati hotarat sa va dedicati unui hobby al vostru, gatitul.

Asa ca v-ati hotarat sa invatati sa faceti clatite.

Graful pentru reteta:



(intreaga reteta aici [http://interactivepython.org/courselib/static/pythonds/Graphs/graphdfs.html#topological-sorting])

Desi sunteti bogati dupa gasirea comorii, v-ati hotarat sa mai exersati ceea ce v-a facut bogati: algoritmii. Asa ca v-ati zis sa rezolvati aceasta reteta folosind sortarea topologica, dar nu intr-o singura metoda, ci in 2:

- Folsind DFS [3p]
- Prin metoda lui Kahn [3p]

Pentru a rezolva acest exercitiu trebuie sa scriti in fisierul dat, graful de intrare.

3. Bonus(1p)

Algoritmul de sortare topologica poate fi aplicat doar pe grafuri orientate aciclice. Dandu-se un graf, sa se specifice daca exista sau nu cicluri. [1p]

pa/laboratoare/laborator-06.txt · Last modified: 2013/04/01 11:54 by traian.rebedea