Structuri de date

Laboratorul 10

Mihai Nan
mihai.nan.cti@gmail.com
Grupa 312CCb

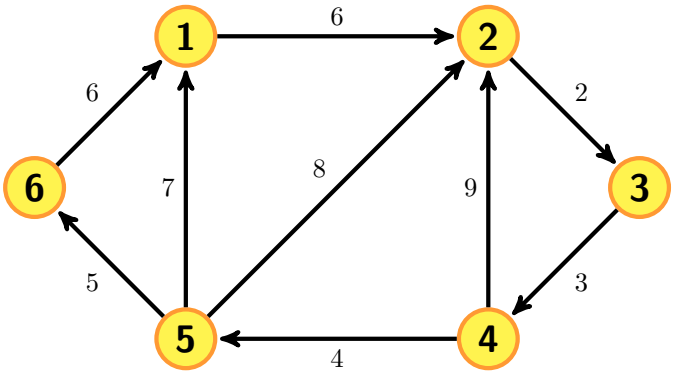
Facultatea de Automatica şi Calculatoare
Universitatea Politehnica din Bucureşti
Anul universitar 2019 - 2020

1 Grafuri orientate

1.1 Introducere

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 *graf ponderat* o structura (V, E, W), unde $G = (V, E)$ este graf si W este o functie, numita *pondere*, care asociaza fiecarei muchii a grafului un cost al parcurgerii ei.

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 $(x_1, x_2), \ldots (x_{k-1}, x_k)$ 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.

Definitie: Un graf orientat ***G = (V, E)*** se numeste *conex* daca intre oricare doua varfuri distincte exista cel putin un lant.

Definitie: Se numeste *componenta conexa* a unui graf orientat ***G = (V, E)*** un *subgraf conex si maximal* cu aceasta proprietate - oricum am mai alege un alt varf din arbore nu exista lant de la acel varf la oricare varf din subgraf.

1.2 Metode de reprezentare

Se numeste *matricea drumurilor* asociata unui graf ***G*** cu ***n*** varfuri o matrice patratica de dimensiune ***n***, in care elementele sunt ***0*** sau ***1***, cu urmatoarea semnificatie:

$$M_{i,j} = \begin{cases} 1 & \text{daca exista drum de la i la j in } \mathbf{G} \\ 0 & \text{daca nu exista drum de la i la j in } \mathbf{G} \end{cases} \quad (1)$$

Matricea drumurilor poate fi determinata plecand de la matricea de adiacenta folosind algoritmul lui ***Roy-Warshall***.

1.3 Conexitate

Definitie: Un graf orientat ***G = (V, E)*** este *tare conex* daca are un singur varf sau daca pentru oricare doua varfuri ***x*** si ***y*** din ***V***, exista un drum de la ***x*** la ***y*** si un drum de la ***y*** la ***x***.

Definitie: Se numeste *componenta tare conexa* a unui graf ***G = (V, E)*** un subgraf ***G' = (V', E')*** care este tare conex si maximal cu aceasta proprietate: daca s-ar mai adauga un varf din ***V \ V'***, subgraful nu ar mai fi tare conex.

1.4 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.

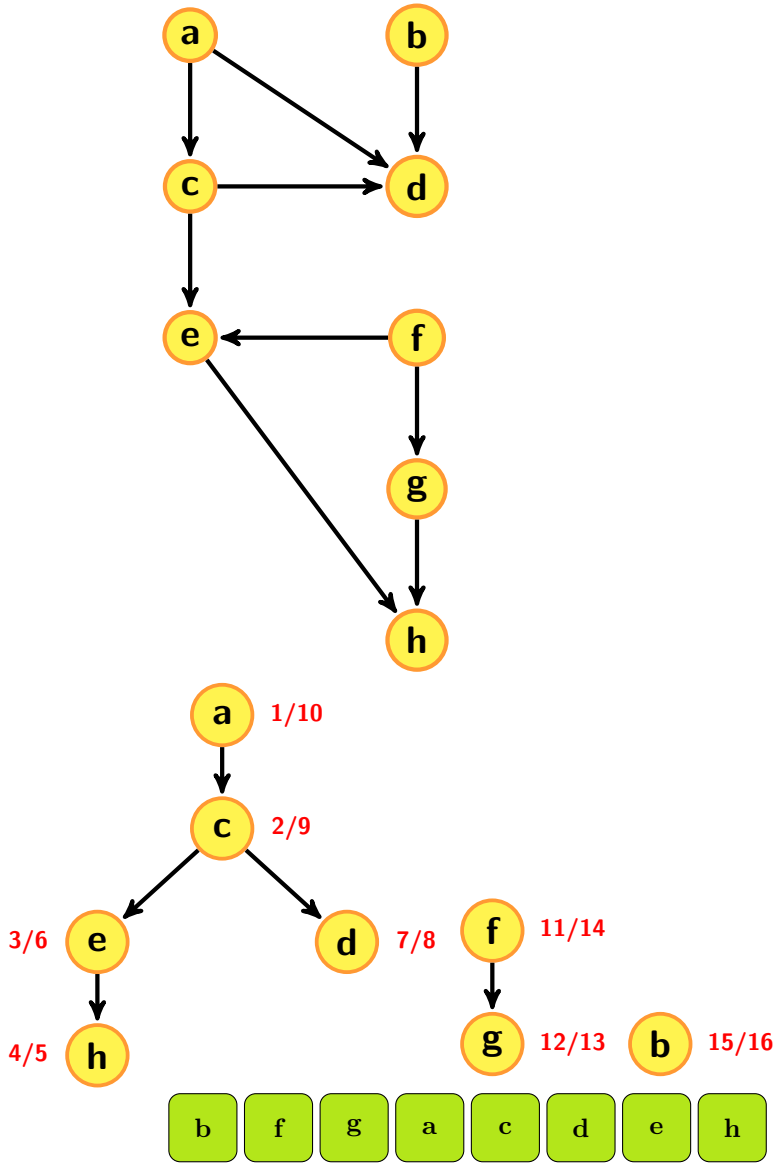
1.4.1 Algoritmul lui Kahn

Pseudocod

```
TopSort(G) {
    V = noduri(G); L = vida; // lista care va contine elementele sortate
    for each (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
        for each 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.

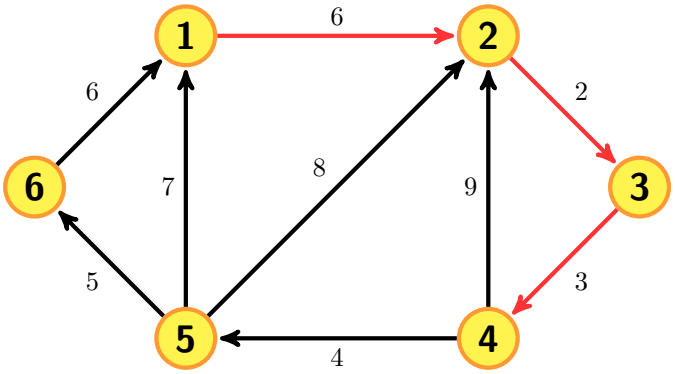


1.5 Distante minime

Fiind dat un graf orientat ***G = (V, E)***, se considera functia w: E → W, numita functie de cost, care asociaza fiecarei muchii o valoare numerica. Domeniul functiei poate fi extins, pentru a include si perechile de noduri intre care nu exista muchie directa, caz in care valoarea este ∞.

Costul unui drum format din muchiile ***p12 p23 ... p(n-1)n***, avand costurile ***w12, w23, ..., w(n-1)n***, este suma ***w = w12 + w23 + ... + w(n-1)n***.

Costul minim al drumului dintre doua noduri este minimul dintre costurile drumurilor existente intre cele doua noduri.



$w(1,4) = 6 + 2 + 3 = 11$

Observatie

Desi, in cele mai multe cazuri, costul este o functie cu valori nenegative, exista situatii in care un graf cu muchii de cost negativ are relevanta practica. O parte din algoritmi pot determina drumul corect de cost minim inclusiv pe astfel de grafuri.
Totusi, nu are sens cautarea drumului minim in cazurile in care graful contine cicluri de cost negativ - un drum minim ar avea lungimea infinita, intrucat costul sau s-ar reduce la fiecare reparcurgere a ciclului.

1.5.1 Algoritmul Floyd - Warshall

Algoritm prin care se calculeaza distantele minime intre oricare 2 noduri dintr-un graf (***drumuri optime multipunct-multipunct***).

Exemplu clasic de ***programare dinamica***.

Idee: la pasul ***k*** se calculeaza cel mai bun cost intre ***u*** si ***v***, folosind cel mai bun cost ***u..k*** si cel mai bun cost ***k..v*** calculat pana in momentul respectiv.

Observatie: Se aplica pentru grafuri ce nu contin cicluri de cost negativ.

Pseudocod

```
FloydWarshall(G):
    n = |V|
    int d[n, n]
    foreach (i, j) in (1..n,1..n)
        d[i, j] = w[i,j] // costul muchiei, sau infinit
    for k = 1 to n
        foreach (i,j) in (1..n,1..n)
            d[i, j] = min(d[i, j], d[i, k] + d[k, j])
```

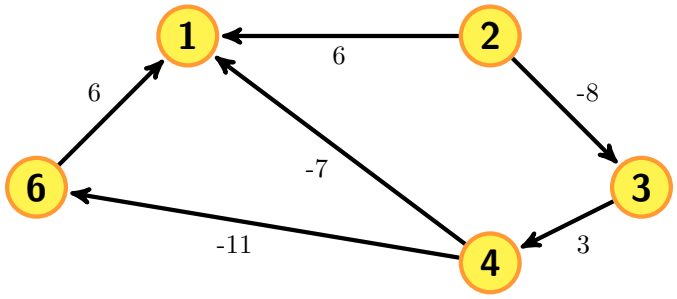
1.5.2 Algoritmul Bellman – Ford

Algoritmul Bellman Ford poate fi folosit si pentru grafuri ce contin muchii de cost negativ, dar nu poate fi folosit pentru grafuri ce contin cicluri de cost negativ (cand cautarea unui drum minim nu are sens).

Cu ajutorul sau putem afla daca un graf contine cicluri. Algoritmul foloseste acelasi mecanism de relaxare ca si Dijkstra, dar, spre deosebire de acesta, nu optimizeaza o solutie folosind un criteriu de optim local, ci parcurge fiecare muchie de un numar de ori egal cu numarul de noduri si incearca sa o relaxeze de fiecare data, pentru a imbunatati distanta pana la nodul destinatie al muchiei curente.

Motivul pentru care se face acest lucru este ca drumul minim dintre sursa si orice nod destinatie poate sa treaca prin maximum ***|V|*** noduri (adica toate nodurile grafului), respectiv ***|V|-1*** muchii; prin urmare, relaxarea tuturor muchiilor de ***|V|-1*** ori este suficienta pentru a propaga pana la toate nodurile informatia despre distanta minima de la sursa.

Daca, la sfarsitul acestor ***|E|*(|V|-1)*** relaxari, mai poate fi imbunatatita o distanta, atunci graful are un ciclu de cost negativ si problema nu are solutie.



Pseudocod

```
BellmanFord(sursa):
    // initializari
    foreach nod in V // V = multimea nodurilor
        daca muchie[sursa, nod]
            d[nod] = w[sursa, nod]
            P[nod] = sursa
        altfel
            d[nod] = +∞
            P[nod] = null
    d[sursa] = 0
    p[sursa] = null

    // relaxari succesive
    // cum in initializare se face o relaxare (daca exista drum direct de la sursa la nod
     =>
    // d[nod] = w[sursa, nod]) mai sunt necesare |V-2| relaxari
    for i = 1 to |V|-2
        foreach (u, v) in E  // E = multimea muchiilor
            daca d[v] > d[u] + w(u,v)
                d[v] = d[u] + w(u,v)
                p[v] = u;

    // daca se mai pot relaxa muchii
    foreach (u, v) in E
        daca d[v] > d[u] + w(u,v)
            fail (''exista cicluri negativ'')
```

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 - 3 puncte

Implementați o sortare topologică a vârfurilor unui graf orientat aciclic, reprezentat folosind liste de adiacență. Sortarea topologică este o operație de ordonare liniară a vârfurilor, astfel încât, dacă există un arc ***(i, j)***, atunci ***i*** apare înaintea lui ***j*** în această ordonare.

Problema 2 - 3 puncte

Implementați o funcție care primește ca parametru un graf orientat și determină pentru orice pereche de noduri ***x*** și ***y*** lungimea minimă a drumului de la nodul ***x*** la nodul ***y*** și afișează matricea drumurilor minime. Prin lungimea unui drum înțelegem suma costurilor arcelor care-l alcătuiesc.

Problema 3 - 4 puncte

Definiți o funcție care determină nodurile care compun drumul de cost minim de la ***x*** la ***y*** pentru un graf orientat cu ponderi pozitive, folosind algoritmul lui Dijkstra.

3 Problema bonus

Problema 4 - 3 puncte

Se dă un graf orientat conex cu N noduri și M muchii cu costuri. Definim un lanț ca fiind un șir de noduri cu proprietatea că între oricare două consecutive există o muchie. Costul unui lanț este dat de suma costurilor muchiilor care unesc nodurile ce îl formează. Definim un ciclu ca fiind un lanț cu proprietatea că primul element al sau este egal cu ultimul.

Să se determine dacă în graful dat există un ciclu de cost negativ. Dacă nu există, să se determine costul minim al unui lanț de la nodul 0 la fiecare dintre nodurile $1, 2, 3, \ldots, N-1$.

4 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!