Structuri de date

Laboratorul 10

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

Facultatea de Automatica şi Calculatoare
Universitatea Politehnica din Bucureşti
Anul universitar 2016 - 2017

1 Grafuri

1.1 Algoritmul lui Dijkstra

Una din problemele teoriei grafurilor consta in determinarea unui drum de cost minim sau maxim dintre doua varfuri $x_i, x_j \in V$ ale grafului ***G = (V, E)***.

In practica, se pune adesea problema determinarii unui plan de transport printr-o retea rutiera astfel incat cheltuielile de transport sau duratele de transport sa fie minime. Este necesar sa se determne drumul cel mai scurt dintre doua noduri oarecare ale retelei rutiere. Acest tip de problema se mai intalneste si in proiectarea retelelor de calculatoare, in stabilirea traseelor mijloacelor de transport in comun etc.

Algoritmul lui Dijkstra permite cautarea lungimilor celor mai scurte drumuri de la un varf s la toate varfurile v ale unui graf ponderat ***G = (V, E, W)***, daca lungimile tuturor arcelor sunt pozitive.

Algoritmul este de tip Greedy: optimul local cautat este reprezentat de costul drumului dintre nodul sursa ***s*** si un nod ***v***. Pentru fiecare nod se retine un cost estimat ***d[v]***, initializat la inceput cu costul muchiei ***s->v***, sau cu $+\infty$, daca nu exista muchie.

Pentru a tine evidenta muchiilor care trebuie relaxate, se folosesc doua structuri: ***S*** (multimea de varfuri deja vizitate) si ***Q*** (o coada cu prioritati, in care nodurile se afla ordonate dupa distanta fata de sursa) din care este mereu extras nodul aflat la distanta minima. In ***S*** se afla initial doar sursa, iar in ***Q*** doar nodurile spre care exista muchie directa de la sursa, deci care au ***d[nod] < +***∞.

Algoritmul selecteaza, in mod repetat, nodul ***u*** care are, la momentul respectiv, costul estimat minim (fata de nodul sursa). In continuare, se incearca sa se relaxeze restul costurilor ***d[v]***. Daca ***d[v] < d[u] + w(u, v)***, ***d[v]*** ia valoarea ***d[u] + w(u, v)***.

Algoritmul se incheie cand coada ***Q*** devine vida, sau cand ***S*** contine toate nodurile.

Pentru a putea determina si muchiile din care este alcatuit drumul minim cautat, nu doar costul sau final, este necesar sa retinem un vector de parinti ***P***. Pentru nodurile care au muchie directa de la sursa, ***P[nod]*** este initializat cu sursa, pentru restul cu ***null***.

Observatie

Pentru o intelegere mai buna a algoritmului, urmariti exemplul oferit in prezentare.

Pseudocod

```
Dijkstra(sursa, dest):
    selectat(sursa) = true
    foreach nod in V // V = multimea nodurilor
        daca exista muchie[sursa, nod]
            // initializam distanta pana la nodul respectiv
            d[nod] = w[sursa, nod]
            introdu nod in Q
            // parintele nodului devine sursa
            P[nod] = sursa
        altfel
            d[nod] = +∞ // distanta infinita
            P[nod] = null // nu are parinte

    // relaxari succesive
    cat timp Q nu e vida
        u = extrage_min (Q)
        selectat(u) = true
        foreach nod in vecini[u] // (*)
            // drumul de la s la nod prin u este mai mic
            daca !selectat(nod) si d[nod] > d[u] + w[u, nod]
              // actualizeaza distanta si parinte
                d[nod] = d[u] + w[u, nod]
                P[nod] = u
                //actualizeaza pozitia in coada
                actualizeaza (Q,nod)

    // gasirea drumului efectiv
    Initializeaza Drum = {}
    nod = P[dest]
    cat timp nod != null
        insereaza nod la inceputul lui Drum
        nod = P[nod]
```

1.2 Arbore minim de acoperire

Gasirea unui arbore minim de acoperire pentru un graf are aplicatii in domenii cat se poate de variate:

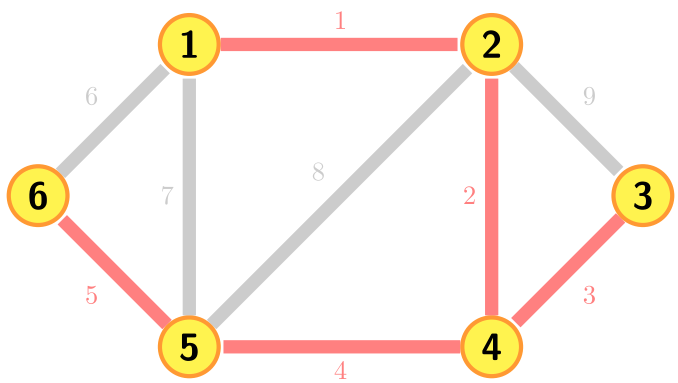
- retele (de calculatoare, telefonie, cablu TV, electricitate, drumuri): se doreste interconectarea mai multor puncte, cu un cost redus si atunci este utila cunoasterea arborelui care conecteaza toate punctele, cu cel mai mic cost posibil. ***STP (Spanning Tree Protocol)*** este un protocol de rutare care previne aparitia buclelor intr-un LAN, si se bazeaza pe crearea unui arbore de acoperire. Singurele legaturi active sunt cele care apar in acest arbore, iar astfel se evita buclele.
- Segmentarea imaginilor: impartirea unei imagini in regiuni de pixeli cu

proprietati asemanatoare. E utila mai apoi in analiza medicala a unei zone afectate de o tumoare de exemplu.

Definitie: Dandu-se un graf conex neorientat ***G =(V, E)***, se numeşte ***arbore de acoperire*** al lui ***G*** un subgraf ***G'=(V, E')*** care contine toate varfurile grafului ***G*** si o submultime minima de muchii ***E'*** $\in$ ***E*** cu proprietatea ca uneste toate varfurile si nu contine cicluri. Cum ***G'*** este conex si aciclic, el este arbore. Pentru un graf oarecare, exista mai multi arbori de acoperire.

Daca asociem o matrice de costuri, w, pentru muchiile din G, fiecare arbore de acoperire va avea asociat un cost egal cu suma costurilor muchiilor continute. Un arbore care are costul asociat mai mic sau egal cu costul oricarui alt arbore de acoperire se numeste ***arbore minim de acoperire*** (minimum spanning tree) al grafului ***G***. Un graf poate avea mai multi arbori minimi de acoperire.

Observatie: Daca toate costurile muchiilor sunt diferite, exista un singur AMA (arbore minim de acoperire).



1.2.1 Algoritmul lui Kruskal

Algoritmul a fost dezvoltat in 1956 de Joseph Kruskal. Determinarea arborelui minim de acoperire se face prin reuniuni de subarbori minimi de acoperire. Initial, se considera ca fiecare nod din graf este un arbore. Apoi, la fiecare pas se selecteaza muchia de cost minim care uneste doi subarbori disjuncti, si se realizeaza unirea celor doi subarbori. Muchia respectiva se adauga la multimea ***MuchiiAMA***, care la sfarsit va contine chiar muchiile din arborele minim de acoperire.

Observatie

Analizati exemplul din prezentare pentru a intelege modul de functionare al algoritmului.

1.2.2 Algoritmul lui Prim

Algoritmul a fost prima oara dezvoltat in 1930 de matematicianul ceh Vojtěch Jarnik, si independent in 1957 de informaticianul Robert Prim, al carui nume l-a luat. Algoritmul considera initial ca fiecare nod este un subarbore independent, ca si Kruskal. Insa spre deosebire de acesta, nu se construiesc mai multi subarbori care se unesc si in final ajung sa formeze ***AMA***, ci exista un arbore principal, iar la fiecare pas se adauga acestuia muchia cu cel mai mic cost care uneste un nod din arbore cu un nod din afara sa. Nodul radacina al arborelui principal se alege arbitrar. Cand s-au adaugat muchii care ajung in toate nodurile grafului, s-a obtinut ***AMA*** dorit. Abordarea seamana cu algoritmul Dijkstra de gasire a drumului minim intre doua noduri ale unui graf.

Pentru o implementare eficienta, urmatoarea muchie de adaugat la arbore trebuie sa fie usor de selectat. Varfurile care nu sunt in arbore trebuie sortate in functie de distanta pana la acesta (de fapt costul minim al unei muchii care leaga nodul dat de un nod din interiorul arborelui). Se poate folosi pentru aceasta o structura de ***heap***. Presupunand ca ***(u, v)*** este muchia de cost minim care uneste nodul ***u*** cu un nod ***v*** din arbore, se vor retine doua informatii:

- ***d[u] = w[u,v]*** distanta de la ***u*** la arbore;
- ***p[u] = v*** predecesorul lui ***u*** in drumul minim de la arbore la u.

La fiecare pas se va selecta nodul ***u*** cel mai apropiat de arborele principal, reunind apoi arborele principal cu subarborele corespunzator nodului selectat. Se verifica apoi daca exista noduri mai apropiate de ***u*** decat de nodurile care erau anterior in arbore, caz in care trebuie modificate distantele dar si predecesorul. Modificarea unei distante impune si refacerea structurii de ***heap***.

Observatie

De asemenea, gasiti o rulare a acestui algoritm in prezentare.

2 Bibliografie

1. Echipa ***Proiectarea Algoritmilor*** - breviar laborator

3 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.

3.1 Probleme standard

Problema 1 - 5 puncte

Folosind algoritmul lui Dijkstra, determinati distanta minima de la un nod la toate celelalte noduri dintr-un graf orientat ponderat, avand costuri pozitive.

Problema 2 - 5 puncte

Implementati o functie care determina arborele minim de acoperire al unui graf ponderat. In implementare puteti folosi algoritmul lui Kruskal sau pe cel al lui Prim.

4 Problema bonus

Problema 3 - 3 puncte

Se da un graf orientat conex cu N noduri si M muchii cu costuri. Definim un lant ca fiind un sir de noduri cu proprietatea ca intre oricare doua consecutive exista o muchie. Costul unui lant este dat de suma costurilor muchiilor care unesc nodurile ce il formeaza. Definim un ciclu ca fiind un lant cu proprietatea ca primul element al sau este egal cu ultimul.

Sa se determine daca in graful dat exista un ciclu de cost negativ. Daca nu exista, sa se determine costul minim al unui lant de la nodul 0 la fiecare dintre nodurile 1, 2, 3, ... , N-1.

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!