Universitate de Stat din Moldova

Facultate de Matematica si Informatica

Tema: Tehnici de programare. Algoritmul lui Kuhn

Student: Stinca Erast

Verificat de: N. Objelean

Chisinau 2017

Universitate de Stat din Moldova

Facultate de Matematica si Informatica

Tema: Tehnici de programare. Tehnica Greedy

Student: Stinca Erast

Verificat de: N. Objelean

Chisinau 2017

**Algoritmul lui Kuhn (Ungar)**

## Introducere

Voi prezenta in continuare problema afectarii impreuna cu una din cele mai eficiente solutii pentru rezolvarea ei: Algoritmul lui Kuhn. In ultima parte a articolului, voi oferi un exemplu de implementare usor de folosit in concursurile de informatica. Se presupun cunoscute notiunile de **cuplaj maxim** si **suport minim** intr-un graf bipartit, precum si modul de obtinere a unui cuplaj maxim intr-un graf bipartit cu ajutorul drumurilor de crestere. Sa incepem cu definitia problemei discutate.

**Problema de afectare**: Exista m muncitori si n lucrari, fiecare muncitor putand efectua una sau mai multe lucrari. Notam muncitorii cu x1 , x2, ..., xm, lucrarile cu y1, y2 , ..., yn, si asociem costul vij ≤ 0 efectuarii de catre muncitorul xi a lucrarii yj, faptul ca vij = ∞ avand semnificatia ca muncitorul xi nu poate efectua lucrarea yj. Sa se gaseasca o repartizare a celor m muncitori la cele n lucrari astfel incat un muncitor sa efectueze cel mult o lucrare, iar o lucrare sa fie efectuata de cel mult un muncitor, cu conditia ca costul total de executie sa fie minim.

Definim **matricea de afectare** astfel: X = (xij), 1 ≤ i ≤ m, 1 ≤ j ≤ n, xij = 1 daca muncitorul xi efectueaza lucrarea yj, si xij = 0 in caz contrar. Cand ∑ij = min(m, n), matricea de afectare se numeste **saturanta**. Voi prezenta algoritmul elaborat in 1955 de Kuhn, numit de acesta algoritmul ungar pentru rezolvarea problemelor de afectare definite de matrice saturante, cunoscut si sub numele de metoda ungara, deoarece se bazeaza pe rezultatele a doi matematicieni maghiari: D. König si E. Egerváry. Demonstratia va presupune ca m = n; se va vedea apoi ca algoritmul se poate extinde foarte usor la matrice care nu sunt patrate.

## Descrierea algoritmului

Algoritmul utilizeaza urmatoarea proprietate:

**Propozitia 1**: Multimea solutiilor minime ale unei probleme de afectare nu se modifica daca la toate elementele unei linii sau coloane a matricei costurilor se aduna acelasi numar real λ.

De asemenea, este necesara cunoasterea urmatoarei teoreme:

**Teorema 1 (Teorema lui König)**: Fie G un graf bipartit, ν(G) cardinalul unui cuplaj maxim in acest graf, si τ(G) cardinalul unui suport minim. Atunci exista egalitatea: ν(G) = τ(G), adica numarul maxim de muchii ale unui cuplaj este egal cu numarul minim de varfuri ale unui suport.

Deoarece m = n si matricea de afectare este saturanta, orice solutie a problemei de afectare contine un singur 1 in fiecare linie si in fiecare coloana, deci, prin transformarea matricei costurilor, costul asociat oricarei solutii creste cu λ. Din acest motiv, multimea solutiilor minime ale problemei de afectare nu se modifica prin transformari de acest tip ale matricei costurilor.

Algoritmul lui Kuhn se bazeaza si pe faptul ca o solutie a problemei de afectare corespunde unui cuplaj al grafului bipartit cu partile X = {x1, x2, ..., xn} si Y = {y1, y2, ..., yn}, si se desfasoara in mai multi pasi, fiind prezentat aici intr-o varianta modificata si simplificata fata de cea originala. Ideea este urmatoarea: daca prin anumite operatii de adunare a unei constante la toate elementele unei linii sau coloane a matricei costurilor toate numerele din matrice raman pozitive si exista un cuplaj de cardinal n in graful bipartit definit de zerourile matricei, atunci solutia problemei se gaseste pe pozitiile nume din matrice.

Acest lucru se demonstreaza imediat pe baza propozitiei 1, avand in vedere faptul ca in matricea care contine numai numere nenegative nu poate exista o transversala cu costul negativ. Vom incerca, deci, sa aplicam in mod repetat transformarile descrise, pana cand va exista un cuplaj de cardinal maxim.

Vom adopta urmatoarele conventii de transpunere a limbajului de teoria grafurilor in termeni de matrice:

* Spunem despre o linie sau despre o coloana ca este **acoperita** daca la un anumit pas face parte din multimea nodurilor marcate in cadrul procesului de determinare a unui suport minim.
* Spunem despre un element al matricei ca este **incercuit** daca face parte din cuplajul gasit pana in momentul respectiv.
* Spunem despre un element al matricei ca este **taiat** daca corespunde unei muchii care va putea fi folosita pentru obtinerea unui drum de crestere, intr-un sens ce va fi clarificat ulterior.

## Algoritmul propriu-zis

Initial, toate liniile si coloanele sunt decuplate. Algoritmul se va desfasura in n pasi, fiecare pas decurgand dupa cum urmeaza:

Se descopera toate liniile si coloanele matricei si se acopera toate coloanele cuplate. Se pastreaza elementele incercuite, dar se considera ca nici un element nu este taiat. Pornind de la cuplajul obtinut la pasul anterior, se determina un suport minim de acelasi cardinal, adica o multime de linii si coloane care acopera toate zerourile. Modul de obtinere a suportului minim a fost dezvoltat de Egerváry, iar demonstratia corectitudinii sale depaseste scopul acestui articol. Cititorii interesati pot consulta Caius Iacob, Matematici clasice si moderne, vol. I, Bucuresti: Editura Tehnica, 1978 pentru o demonstratie completa. Atata timp cat putem gasi un zero descoperit pe o linie i si o coloana j, il taiem si consideram urmatoarele doua cazuri:

1. **Linia i este cuplata**: Acoperim linia i si descoperim coloana cu care este cuplata. Coloana cu care este cuplata linia i va fi mereu acoperita din cauza modului in care lucreaza algoritmul: in orice moment ori linia ori coloana pe care se afla un zero incercuit vor fi acoperite, deoarece initial toate coloanele cuplate sunt acoperite, si singura operatie pe care o efectuam este descoperirea unei coloane si acoperirea liniei cu care aceasta este cuplata.
2. **Linia i nu este cuplata**: Incercam sa construim un drum de crestere, dupa cum urmeaza:   
   Pornim de pe pozitia pe care tocmai am taiat-o. Daca coloana in care se afla nu este cuplata, cuplam linia i cu coloana j, si drumul de crestere este complet. Daca coloana j este cuplata, continuam drumul de crestere cu zeroul incercuit din coloana j si zeroul taiat care se gaseste pe linia acelui zero. Fie k linia cu care este cuplata coloana j. (k, j) este incercuit, coloana j nu este acoperita, ceea ce inseamna ca linie k este acoperita. Dar o linie nu poate fi acoperita decat in conditiile in care la un pas anterior am gasit un zero descoperit in acea linie, l-am taiat, si am descoperit coloana din care facea parte. In concluzie, vom putea gasi mereu un zero taiat pe linia care ne intereseaza. Deoarece numarul elementelor incercuite este finit, fiecare din ele va fi selectat cel mult o data, si vom putea mereu sa gasim un zero taiat cu care sa continuam drumul de crestere, rezulta ca in cele din urma vom gasi un zero taiat intr-o coloana necuplata, moment in care drumul de crestere va fi complet.

Daca la pasul anterior nu am gasit un drum valid de crestere, vom considera matricea formata din elementele care nu apartin unor linii sau coloane acoperite si vom calcula minimul λ din elementele coloanelor neacoperite si il vom aduna la elementele liniilor acoperite. Deoarece toate zerourile sunt acoperite o singura data conform modului de obtinere a suportului minim, aceste operatii nu vor face sa dispara din zerourile obtinute pana la acest pas, si nici nu vor provoca aparitia unor elemente negative, insa vor produce cel putin inca un zero, pe pozitia pe care a fost gasit minimul. Suportul minim al matricei costurilor va creste cu cel putin o unitate prin aplicarea repetata a acestor pasi. Conform teoremei lui König, aceasta inseamna ca numarul elementelor unui cuplaj maxim al matricei obtinute creste cu cel putin o unitate, deci in cele din urma vom reusi sa gasim un drum de crestere cu ajutorul caruia sa marim cuplajul obtinut. Concluzia pe care o tragem este urmatoarea: la fiecare din cei n pasi vom repeta aplicarea operatiilor descrise pana la obtinerea unui drum de ameliorare, iar cuplajul maxim va creste cu 1. DUpa cei n pasi, vom avea un cuplaj de cardinal n care va corespune transversalei minime din matricea costurilor. Complexitatea algoritmului descris este O(N4), insa se comporta bine in practica, in functie de costurile muchiilor grafului pe care se realizeaza cuplajul.

## Detalii de implementare

Algoritmul lui Kuhn este destul de cunoscut de catre participantii la concursurile de informatica, dar nu este folosit foarte des, deoarece se considera ca este mult prea greu de implementat. In continuare se va vedea ca, cu putina atentie, metoda descrisa mai sus este implementabila in timp de concurs, in cel mult 10 minute.

Avem nevoie de o matrice G care va retine graful propriu-zis. Pentru a nu irosi spatiul, in loc de a memora in alta parte matricea modificata, vom memora in vectorii vr si vc suma valorilor care au fost adaugate pe o anumita linie, respectiv scazute pe o anumita coloana. Vectorii l si r vor retine cu cine este cuplata o anumita linie, respectiv coloana, sau 0 daca acea linie/coloana nu a fost cuplata inca. pi = j are semnificatia ca (i, j) este taiat, iar vectorii cr si cc sunt folositi pentru a afla rapid daca o linie/coloana este acoperita. Fara alte comentarii, voi trece direct la prezentarea codului sursa care implementeaza pas cu pas cele descrise mai sus.

#include <stdio.h>

#include <string.h>

int N, G[256][256], l[256], r[256], p[256], cr[256], cc[256], vr[256], vc[256];

void find\_zero () {

    int i, j, min, t;

    for (min = 1 << 30, i = 1; i <= N; ++ i) if (!cr[i])

        for (j = 1; j <= N; ++ j) if (!cc[j])

            min <?= G[i][j] + vr[i] - vc[j];

    for (i = 1; i <= N; ++ i) if (cr[i]) vr[i] += min;

    for (j = 1; j <= N; ++ j) if (!cc[j]) vc[j] += min;

    for (i = 1; i <= N; ++ i) if (!cr[i])

        for (j = 1; j <= N; ++ j) if (!cc[j] && G[i][j] + vr[i] == vc[j])

            if (r[i]) {

                p[i] = j, cr[i] = 1, cc[r[i]] = 0;

                break;

            } else {

                do t = l[j], r[i] = j, l[j] = i, i = t, j = p[i]; while (t);

                return;

            }

    find\_zero ();

}

int main() {

    int i, j, min, cnt;

    scanf("%d", &N);

    for (i = 1; i <= N; ++ i)

        for (j = 1; j <= N; ++ j) scanf("%d", &G[i][j]);

    memset (vr, 0, sizeof (vr)), memset (vc, 0, sizeof (vc));

    memset (l, 0, sizeof (l)), memset (r, 0, sizeof (r));

    for (cnt = 0; cnt < N; ++ cnt) {

        memset (cr, 0, sizeof (cr)), memset (p, 0, sizeof (p));

        memcpy (cc, l, sizeof (cc));

        find\_zero ();

    }

    return 0;

}

## Optimizari

Codul de mai sus a fost destul de compactat si optimizat incat sa nu mai aiba nevoie decat de urmatorul pas pentru a stoarce si ultima picatura de performanta din el: nu este nevoie sa pornim cu cuplajul de cardinal 0. La inceput scadem valoarea minima din fiecare linie, apoi scadem valoarea minima din fiecare coloana. In acest moment putem sa parcurgem matricea pas cu pas si, cand gasim un element nul a carui linie si coloana nu au fost cuplate il adaugam la cuplajul initial. Astfel, nu vom mai fi nevoiti sa facem n pasi, iar timpul de executie va scadea simtitor.

## Extinderi

Dupa cum am promis, algoritmul lui Kuhn va putea fi aplicat si pe matrice ne-patrate. Daca m ≠ n vom transforma matricea costurilor astfel:

1. pentru m < n, se adauga matricei costurilor initiale n - m linii care contin numai 0.
2. pentru m > n, se adauga matricei costurilor initiale m - n coloane care contin numai 0.

In cazul unor probleme de afectare pentru care se cauta o afectare saturanta care maximizeaza suma ∑i,jxijvij, vom defini o noua matrice a costurilor, ale care elemente sunt: v'ij = v0 - vij, 1 ≤ i ≤ m, 1 ≤ j ≤ n, v0 = maxi,jvij. Observam ca daca matricea costurilor initiala contine un element egal cu ∞ aceasta problema este banala, deci presupunem v0 < ∞. Solutia de cost minim a problemei cu matricea costurilor modificata va coincide cu solutia de cost maxim a problemei de afectare cu matricea costurilor initiala. Demonstratia este imediata, si ramane ca tema cititorului.