Universitatea Tehnică a Moldovei

Facultatea - Calculatoare Informatică și Microelectronică

Disciplina: *Analiza și proiectarea algoritmilor*

**Raport**

Lucrarea de laborator Nr.3

Tema: Algoritmi greedy

A efectuat: st.gr. TI-207 Bunescu Gabriel

A verificat: lect.univ. Bîtca Ernest

Chișinău 2021

**Cuprins**

[I. Scopul lucrării 2](#_Toc88582074)

[II. Sarcina lucrării 2](#_Toc88582075)

[III. Considerații teoretice 3](#_Toc88582076)

[IV. Analiza empirică a algortimilor 4](#_Toc88582077)

[V. Concluzie: 5](#_Toc88582078)

# I. Scopul lucrării

1. Studierea tehnicii greedy

2. Analiza și implimentarea algortimilor greedy

# II. Sarcina lucrării

1. De implimentat algoritmii Kruskal și Prim, în baza a 3 tipuri de grafuri conexe (dens, rar și introdus de utilizator).
2. De realizat analiza empirică a algortimilor.

# III. Considerații teoretice

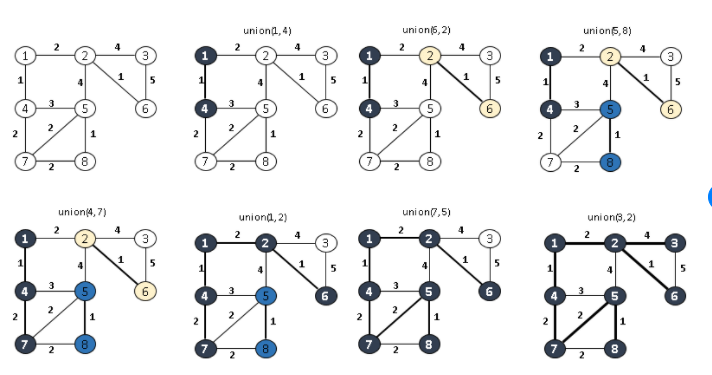
Algoritimul Kruskal

Algoritmul lui Kruskal este un algoritm în teoria grafurilor care găsește arborele parțial de cost minim pentru un graf conex ponderat. Cu alte cuvinte, găsește submulțimea muchiilor care formează un arbore care include toate vârfurile și care este minimizat din punct de vedere al costului. Dacă graful nu este conex, atunci algoritmul găsește o pădure parțială de cost minim (un arbore parțial de cost minim pentru fiecare componentă conexă). Algoritmul lui Kruskal este un exemplu de algoritm greedy.

Algoritmul funcționează în felul următor:

* creează o pădure *F* (o mulțime de arbori), unde fiecare vârf din graf este un arbore separat
* creează o mulțime *S* care conține toate muchiile din graf
* atât timp cât *S* este nevidă
* elimină o muchie de cost minim din *S*
* dacă acea muchie conectează doi arbori distincți, atunci adaugă muchia în pădure, combinând cei doi arbori într-unul singur
* altfel, ignoră muchia

        La sfârșitul algoritmului, pădurea are doar o componentă care reprezintă un arbore parțial de cost minim al grafului.

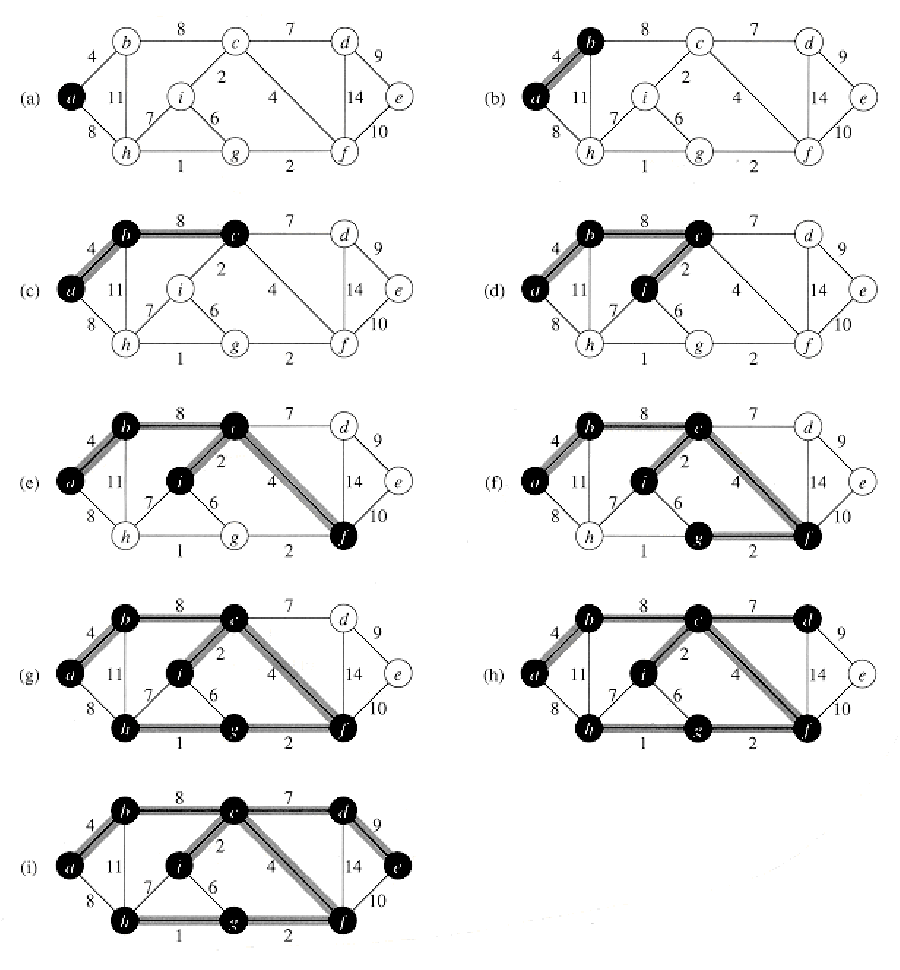


Algoritmul Prim

Considerăm un graf neorientat ponderat (cu costuri) conex G. Se numește **arbore parțial** un graf parțial al lui G care este arbore. Se numește **arbore parțial de cost minim** un arbore parțial pentru care suma costurilor muchiilor este minimă. Dacă graful nu este conex, vorbim despre o **pădure parțială de cost minim**. **Algoritmul lui Prim** permite determinarea unui arbore parțial de cost minim (APM) într-un graf ponderat cu N noduri.

Determinarea APM-ului se face astfel:

* se stabilește un nod de plecare; acesta va fi rădăcina arborelui, care se va crea pas cu pas, prin adăugarea de noi noduri;
* în mod repetat:
* se alege un nod neadăugat încă în arborele curent pentru care muchia dintre el și un nod din arbore are cost minim;
* se adăugă nodul în arbore;
* când nu se mai poate face alegerea unui asemenea nod, fie au fost adăugate toate nodurile, fie graful nu este conex și au fost adăugate în arbore toate nodurile din componenta conexă a nodul inițial;
* dacă graful nu este conex, continuăm cu următoarea componentă conexă.



# IV. Analiza empirică a algortimilor

Tabelul 1. Caz favorabil

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vârfuri | | 10 | 50 | 100 | 250 | 500 | 750 | 1000 |
| Prim | Iteratii | 18 | 98 | 198 | 498 | 998 | 1498 | 1998 |
| Timp | 0.000054 | 0.000558 | 0.003596 | 0.056434 | 0.305453 | 1.566004 | 2.626997 |
| Kruskal | Iteratii | 24 | 141 | 284 | 743 | 1493 | 2240 | 2990 |
| Timp | 0.000054 | 0.000478 | 0.003841 | 0.052974 | 0.377381 | 1.253975 | 2.833898 |

Fig1.1.Cazul favorabil,Iterații

Fig1.2.Cazul favorabil,Timpul

Tabelul 2. Caz mediu

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vârfuri | | 10 | 50 | 100 | 250 | 500 | 750 | 1000 |
| Prim | Iteratii | 64 | 964 | 4652 | 11264 | 33668 | 58074 | 60714 |
| Timp | 0.000088 | 0.001821 | 0.009032 | 0.191165 | 2.431631 | 9.093601 | 20.107653 |
| Kruskal | Iteratii | 55 | 939 | 2654 | 25266 | 177650 | 292601 | 585296 |
| Timp | 0.000082 | 0.001256 | 0.004422 | 0.168923 | 1.163046 | 3.861011 | 9.223161 |

Fig2.1.Cazul mediu,Iterații

Fig2.2.Cazul mediu,Timpul

Tabelul 3.Caz defavorabil

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vârfuri | | 10 | 50 | 100 | 250 | 500 | 750 | 1000 |
| Prim | Iteratii | 76 | 973 | 3914 | 18322 | 47379 | 54058 | 110824 |
| Timp | 0.000068 | 0.002453 | 0.027881 | 0.315246 | 1.964565 | 6.333052 | 21.512278 |
| Kruskal | Iteratii | 88 | 1133 | 6700 | 17882 | 90536 | 313689 | 604067 |
| Timp | 0.000061 | 0.001138 | 0.016179 | 0.09016 | 0.845998 | 3.420427 | 8.230892 |

Fig3.1.Cazul defavorabil,Iterații

Fig3.2.Cazul defavorabil,Timpul

* Complexitatea algoritmilor folosiți depinde foarte mult de structura de date folosită. În acest laborator s-a folosit o matrice de adiacență, ceea ce înseamnă că complexitatea algoritmului Prim este O(|V|2), unde V este numărul de vârfuri. Implementări care folosesc binary heap, lista de adiacență, fibonacci heap, obțin o complexitate logaritmică.
* Complexitatea pentru algoritmul Kruskal pentru matricea de adiacență este O(V2 + ElogE), deoarece pentru a găsi muchiile e nevoie de O(V2) și O(ElogE) pentru a le sorta. E - edges

|  |  |
| --- | --- |
| Prim | Kruskal |
| Începe să construiască MSP de la orice vârf din graf | Începe să construiască MSP de la cea mai mică pondere |
| Graful trebuie sa fie conex | Nu e necesar ca graful sa fie conex |

# V. Concluzie:

În această lucrare de laborator s-au studiat algoritmii după tehnica greedy, Prim și Kruskal, conform timpului de execuție și a numărului de iterații. Algoritmii Greedy aleg soluția optimă pentru fiecare etapă în speranța de a găsi un optim global, iar algoritmii cercetați aleg muchiile fie cu ponderea cea mai mică după o sortare fără a forma un ciclu, fie începând cu un vârf se alege muchia cu cea mai mică pondere la acel moment.

În cazul meu eu ma verificat trei cazurii de complexitare: favorabil, mediu și defavorabil, pentru diferite valori ale vîrfurilor de la 10 pînă la 1000.

Problemele care cercetează arborii minimi de acoperire au întrebuițări în viața cotidiană în telecomunicații, rețele de transport, electrice. Algoritmi care se bazează pe MSP sunt aplicați pentru recunoașterea scrisului de mână cu expresii matematice, proiectarea circuitelor și măsurarea omogenității materialelor bidimensionale.

**VI Codul sursă:**

#include <iostream>

#include <stdlib.h>

#include <time.h>

#define max 10000

using namespace std;

int MS[max ][max ], MV[max][max], MVprim[max][max], MVkruskal[max][max], n; const int INF = 10000;

int count1 = 0, count2 = 0;

int parent[max] = { 0 }, visited[max] = { 0 }; void RESET() {

count1 = 0;

count2 = 0;

for (int i = 0; i < n; i++) {

parent[i] = 0;

visited[i] = 0;

for (int j = 0; j < n; j++) {

MVprim[i][j] = MV[i][j];

MVkruskal[i + 1][j + 1] = MV[i][j];

}

}

}

void nr\_virf\_defavorabil() {

int j;

cout << "Numarul de virfuri: ";

cin >> n;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

//cin >> MS[i][j]; if (j > i) {

MS[i][j] = rand() % 10000;

}

if (i > j) { MS[i][j] = MS[j][i];}

}

}

void costurile\_defavorabil() {

cout << "Costurile muchiilor \n";

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (MS[i][j] && i != j) {

//cout << i + 1 << " -> " << j + 1 << " : " << MS[i][j]<< endl;

MV[i][j] = MS[i][j];

}

else

MV[i][j] = INF;

}

}

RESET();

}

void nr\_virf\_favorabil() {

cout << "Numarul de virfuri: ";

cin >> n;

cout << "Matricea de adiacenta\n";

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

//cin >> MS[i][j]; //de la tastatura

MS[i][i + 1] = rand() % 1000;

}

}

}

void costurile\_favorabil() {

cout << "Costurile muchiilor \n";

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (MS[i][j] && i != j) {

cout << i + 1 << " -> " << j + 1 << " : " << MS[i][j]<< endl;//

MV[i][j] = MS[i][j];

}

else

MV[i][j] = INF;

}

}

RESET();

}

void nr\_virf\_mediu() {

cout << "Numarul de virfuri: ";

cin >> n;

//cout << "Introduceti matricea de adiacenta\n";

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

//cin >> MS[i][j];

if (i % 2 == 0 && j % 2 == 0) {

j = j + 1;

MS[i][j] = rand() % 1000;

}

else if (i % 2 != 0 && j % 2 != 0) {

j = j + 1;

MS[i][j] = rand() % 1000;

}

else if (i > j) {

MS[i][j] = MS[j][i];

}

}

}

}

void costurile\_mediu() {

cout << "Costurile muchiilor \n";

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (MS[i][j] && i != j) {

//cout << i + 1 << " -> " << j + 1 << " : " << MS[i][j] <<endl;

MV[i][j] = MS[i][j];

}

else

MV[i][j] = INF;

}

}

RESET();

}

int FIND(int i) {

while (parent[i]) {

i = parent[i]; count2++;

}

return i;

}

bool UNION(int i, int j) {

count2++;

if (i != j) {

parent[j] = i;

return true;

}

return false;

}

void PRIM() {

int a, b, u, v, ne = 1, min,costul;;

visited[0] = 1;

cout << "Algoritmul lui Prim " << endl << endl; while (ne < n) {

count1++;

min = INF;

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

if (MVprim[i][j] < min)

if (visited[i] != 0) {

count1++;

min = MVprim[i][j];

a = u = i;

b = v = j;

}

if (visited[u] == 0 || visited[v] == 0) {

cout << "Muchia " << ne++ << ": " << a+1 << " -> " << b+1 <<",costul= " << min << endl;

visited[b] = 1;

}

MVprim[a][b] = MVprim[b][a] = INF;

}

cout << endl << "Nr. Iteratii : " << count1 << endl;

}

void KRUSKAL() {

int a, b, u, v, ne = 1, min;

cout << "Algoritmul Kruskal" << endl << endl; while (ne < n) {

count2++;

min = INF;

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

if (MVkruskal[i][j] < min) {

count1++;

min = MVkruskal[i][j];

a = u = i;

b = v = j;

}

}

}

u = FIND(u);

v = FIND(v);

if (UNION(u, v)) {

cout << "Muchia " << ne++ << ": " << a << " -> " << b << ", costul = " << min << endl;

}

MVkruskal[a][b] = MVkruskal[b][a] = INF;

}

cout << endl << "Nr. Iteratii : " << count2 << endl;

}

int main() {

double t1, t2;

int ChooseMenu;

int x;

x: while (true) {

cout << "1. Cazul favorabil." << endl

<< "2. Cazul mediu." << endl

<< "3. Cazul defavorabil." << endl

<< "0. Stop Program." << endl;

cout << endl << "Raspuns : ";

cin >> ChooseMenu;

switch (ChooseMenu) { case 1: {

while (true) {

cout << "1. Introduceti numarul de virfuri." << endl

<< "2. Algoritmul Prim." << endl

<< "3. Algoritmul Kruskal." << endl

<< "0. Meniul principal." << endl;

cout << endl << "Raspuns : ";

cin >> ChooseMenu;

switch (ChooseMenu) { case 1: {

nr\_virf\_favorabil(); costurile\_favorabil(); break;

}

case 2: {

t1 = clock();

PRIM();

t2 = clock();

cout << "Timpul de executie : " << fixed << (t2 - t1) / CLOCKS\_PER\_SEC << " sec" << endl;

break;

}

case 3: {

t1 = clock();

KRUSKAL();

t2 = clock();

cout << "Timpul de executie : " << fixed << (t2 - t1) / CLOCKS\_PER\_SEC << " sec" << endl;

break;

}

case 0: {

goto x;

}

default: {

cout << "ERORR!" << endl;

break;

}

}

}

case 2: {

while (true) {

cout << "1-> Introduceti numarul de virfuri." << endl

<< "2 -> Algoritmul Prim." << endl

<< "3-> Algoritmul Kruskal." << endl

<< "0-> Meniul principal." << endl;

cout << endl << "Raspuns : ";

cin >> ChooseMenu;

switch (ChooseMenu) { case 1: {

nr\_virf\_mediu(); costurile\_mediu(); break;

}

case 2: {

t1 = clock(); PRIM();

t2 = clock();

cout << "Timpul de executie : " << fixed << (t2 - t1) /

CLOCKS\_PER\_SEC << " sec" << endl;

break;

}

case 3: {

t1 = clock();

KRUSKAL();

t2 = clock();

cout << "Timpul de executie : " << fixed << (t2 - t1) / CLOCKS\_PER\_SEC << " sec" << endl;

break;

}

case 0: {

goto x;

}

default: {

cout << "ERORR!" << endl;

break;

}

}

}

case 3: {

while (true) {

cout << "1. Introduceti numarul de virfuri." << endl

<< "2. Algoritmul Prim." << endl

<< "3. Algoritmul Kruskal." << endl

<< "0. Meniul principal." << endl;

cout << endl << "Raspuns : ";

cin >> ChooseMenu;

switch (ChooseMenu) { case 1: {

nr\_virf\_defavorabil(); costurile\_defavorabil(); break;

}

case 2: {

t1 = clock(); PRIM();

t2 = clock();

cout << "Timpul de executie : " << fixed << (t2 - t1) /CLOCKS\_PER\_SEC << " sec" << endl;

break;

}

case 3: {

t1 = clock();

KRUSKAL();

t2 = clock();

cout << "Timpul de executie : " << fixed << (t2 - t1) / CLOCKS\_PER\_SEC << " sec" << endl;

break;

}

case 0: {

goto x;

}

default: {

cout << "ERORR!" << endl;

break;

}

}

}

case 0: {

return 0;

}

default: {

cout << "ERORR" << endl;

break;

}

}

}

}

}

}

return 0;

}