

Ministerul Educației al Republicii Moldova

Universitatea Tehnică a Moldovei

Departamentul ISA

RAPORT

Lucrarea de laborator nr.3

Analiza si proiectarea algoritmilor

A efectuat:
st. gr. TI-173

Roșca Florin

A verificat:
lect., univ.

Andrievschi-Bagrin Veronica

Chișinău 2018

LUCRAREA DE LABORATOR NR.3

Tema: Tehnica greedy de proiectare a algoritmilor

Scopul lucrării:

- Studierea tehnicii greedy.
- Analiza și implementarea algoritmilor greedy.

Note de curs

Algoritmii *greedy* (greedy = lacom) sunt în general simpli și sunt folosiți la rezolvarea problemelor de optimizare, cum ar fi: să se găsească cea mai bună ordine de executare a unor lucrări pe calculator, să se găsească cel mai scurt drum într-un graf etc. În cele mai multe situații de acest fel avem:

- mulțime de *candidați* (lucrări de executat, vârfuri ale grafului etc);
 - o funcție care verifică dacă o anumită mulțime de candidați constituie o *soluție posibilă*, nu neapărat optimă, a problemei;
 - o funcție care verifică dacă o mulțime de candidați este *fezabilă*, adică dacă este posibil să completăm această mulțime astfel încât să obținem o soluție posibilă, nu neapărat optimă, a problemei;
1. o *funcție de selecție* care indică la orice moment care este cel mai promițător dintre candidații încă nefolosiți;
 2. o *funcție obiectiv* care dă valoarea unei soluții (timpul necesar executării tuturor lucrărilor într-o anumită ordine, lungimea drumului pe care l-am gasit) aceasta este funcția pe care urmărim să o optimizăm (minimizăm)

SARCINA DE BAZĂ:

1. De studiat tehnica greedy de proiectare a algoritmilor.
2. De implementat într-un limbaj de programare algoritmii Prim și Kruskal.
3. De făcut analiza empirică a algoritmilor Kruskal și Prim.
4. De alcătuit un raport.

Algoritmul lui Kruskal

Arborele de acoperire minim poate fi construit muchie, cu muchie, după următoarea metoda a lui Kruskal (1956): se alege întâi muchia de cost minim, iar apoi se adaugă repetat muchia de cost minim nealeasă anterior și care nu formează cu precedentele un ciclu. Alegem astfel $V-1$ muchii. Este ușor de dedus că obținem în final un arbore.

În algoritmul lui Kruskal, la fiecare pas, graful parțial $\langle V, A \rangle$ formează o pădure de componente conexe, în care fiecare componentă conexă este la rândul ei un arbore de acoperire minim pentru vârfurile pe care le conectează. În final, se obține arborele parțial de cost minim al grafului G .

Pentru a implementa algoritmul, trebuie să putem manipula submulțimile formate din vârfurile componentelor conexe. Folosim pentru aceasta o structura de date pentru mulțimi disjuncte pentru prezentarea mai multor mulțimi de elemente disjuncte [Cormen]. Fiecare mulțime conține vârfurile unui arbore din pădurea curentă. Funcția **Find-Set** (u) returnează un element reprezentativ din mulțimea care îl conține pe u . Astfel, putem determina dacă două vârfuri u și v aparțin aceleiași arbore testând dacă **Find-Set** (u) este egal cu **Find-Set** (v). Combinarea arborilor este realizată de procedura **Union**. În acest caz, este preferabil să reprezentăm graful ca o lista de muchii cu costul asociat lor, astfel încât să putem ordona această listă în funcție de cost. În continuare este prezentat algoritmul:

```
MST - Kruskal( $G, w$ )
1:  $A \leftarrow \emptyset$ 
2: for fiecare vârf  $v \in V[G]$  do
3:   Make-Set( $v$ )
4: sortează muchiile din  $M$  crescător în funcție de cost
5: for fiecare muchie  $(u, v) \in M$  do
6:   if Find-Set( $u$ )  $\neq$  Find-Set( $v$ )
7:     then  $A \leftarrow A \cup \{(u, v)\}$ 
8:       Union( $u, v$ )
9: return  $A$ 
```

```
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 << "Numarul de iteratii : " << count2 << endl;
}
```

Algoritmul lui Prim

Cel de-al doilea algoritm greedy pentru determinarea arborelui de acoperire minimal al unui graf se datorează lui Prim (1957). În acest algoritm, la fiecare pas, mulțimea A de muchii alese împreună cu mulțimea U a vârfurilor pe care le conectează formează un arbore parțial de cost minim pentru subgraful $\langle U, A \rangle$ al lui G . Inițial, mulțimea U a vârfurilor acestui arbore conține un singur vârf oarecare din V , care va fi rădăcina, iar mulțimea A a muchiilor este vidă. La fiecare pas, se alege o muchie de cost minim, care se adaugă la arborele precedent, dând naștere unui nou arbore parțial de cost minim. Arborele parțial de cost minim crește “natural”, cu câte o ramură, până când va atinge toate vârfurile din V , adică până când $U = V$.

Cheia implementării eficiente a algoritmului lui Prim este să procedăm în așa fel încât să fie ușor să selectăm o nouă muchie pentru a fi adăugată la arborele format de muchiile din A . În pseudocodul de mai jos, graful conex G și rădăcina r a arborelui minim de acoperire, care urmează a fi dezvoltat, sunt privite ca date de intrare pentru algoritm. În timpul execuției algoritmului, toate vârfurile care *nu* sunt în arbore se află într-o coadă de prioritate Q bazată pe un câmp key . Pentru fiecare vârf v , $key[v]$ este costul minim al oricărei muchii care îl unește pe v cu un vârf din arbore. Prin convenție, $key[v] = \infty$ dacă nu există o astfel de muchie. Câmpul $\pi[v]$ reține „părintele” lui v din arbore. $Adj[u]$ este lista de adiacență cu vârfurile v din V astfel încât $uv \in E$.

```
-----  
3:   do  $key[u] \leftarrow \infty$   
4:  $key[r] \leftarrow 0$   
5:  $\pi[r] \leftarrow NIL$   
6: while  $Q \neq \emptyset$   
7:   do  $u \leftarrow Extract\_Min(Q)$   
8:   for fiecare vârf  $v \in Adj[u]$   
9:     do if  $v \in Q$  and  $w(u, v) < key[v]$   
10:        then  $\pi[v] \leftarrow u$   
11:            $key[v] \leftarrow w(u, v)$ 
```

```
void PRIM() {  
    int a, b, u, v, ne = 1, min;  
    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 << "Numarul de iteratii : " << count1 << endl;  
    }  
}
```

Codul programului in C++

// Lab APA -3.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

```
#include "pch.h"
#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() {
    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;
            }
            else 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]
                MV[i][j] = MS[i][j];
            }
            else
                MV[i][j] = INF;
        }
    }
}

<< endl;
```

```

    }
    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;
    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) {
    system("cls");
    cout << "1. Cazul favorabil." << endl
        << "2. Cazul mediu." << endl
        << "3. Cazul defavorabil." << endl
        << "0. Stop Program." << endl;
    cout << endl << "Raspuns : ";
    cin >> ChooseMenu;
    system("cls");
    switch (ChooseMenu) {
    case 1: {
        while (true) {
            system("cls");
            cout << "1. Introduceti numarul de virfuri." << endl
                << "2. Algoritmul Prim." << endl
                << "3. Algoritmul Kruskal." << endl
                << "0. Meniul principal." << endl;
            cout << endl << "Raspuns : ";
            cin >> ChooseMenu;
            system("cls");
            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;
    }
}
system("pause");
system("cls");
}
case 2: {
    while (true) {
        system("cls");
        cout << "1-> Introduceti numarul de virfuri." << endl
        << "2 -> Algoritmul Prim." << endl
        << "3-> Algoritmul Kruskal." << endl
        << "0-> Meniul principal." << endl;
        cout << endl << "Raspuns : ";
        cin >> ChooseMenu;
        system("cls");
        switch (ChooseMenu) {
            case 1: {
                nr_virf_meniu();
                costurile_meniu();
                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;
        }
    }

    system("pause");
    system("cls");
}

case 3: {
    while (true) {
        system("cls");
        cout << "1. Introduceti numarul de virfuri." << endl
            << "2. Algoritmul Prim." << endl
            << "3. Algoritmul Kruskal." << endl
            << "0. Meniul principal." << endl;
        cout << endl << "Raspuns : ";
        cin >> ChooseMenu;
        system("cls");
        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;
            }
        }
        system("pause");
        system("cls");
    }
}

case 0: {
    return 0;
}

```

```

        default: {
            cout << "ERORR" << endl;
            break;
        }
    }
}

system("pause");
system("cls");
}
return 0;
}

```

Cazul favorabil

```

C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Muchia 979: 979 -> 980, costul: 9400
Muchia 980: 980 -> 981, costul: 4053
Muchia 981: 981 -> 982, costul: 6436
Muchia 982: 982 -> 983, costul: 3905
Muchia 983: 983 -> 984, costul: 8558
Muchia 984: 984 -> 985, costul: 9696
Muchia 985: 985 -> 986, costul: 1009
Muchia 986: 986 -> 987, costul: 156
Muchia 987: 987 -> 988, costul: 9683
Muchia 988: 988 -> 989, costul: 2028
Muchia 989: 989 -> 990, costul: 8624
Muchia 990: 990 -> 991, costul: 5867
Muchia 991: 991 -> 992, costul: 2416
Muchia 992: 992 -> 993, costul: 9506
Muchia 993: 993 -> 994, costul: 7515
Muchia 994: 994 -> 995, costul: 8
Muchia 995: 995 -> 996, costul: 1883
Muchia 996: 996 -> 997, costul: 4376
Muchia 997: 997 -> 998, costul: 5240
Muchia 998: 998 -> 999, costul: 5338
Muchia 999: 999 -> 1000, costul: 7005

Numarul de iteratii : 1998
Timpul de lucru al algoritmului : 9.311000 sec
Press any key to continue . . .

```

Fig 1. Alg.Prim 1000 noduri

```

C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Muchia 79: 79 -> 80, costul: 5561
Muchia 80: 80 -> 81, costul: 8102
Muchia 81: 81 -> 82, costul: 9266
Muchia 82: 82 -> 83, costul: 5266
Muchia 83: 83 -> 84, costul: 4263
Muchia 84: 84 -> 85, costul: 2380
Muchia 85: 85 -> 86, costul: 9678
Muchia 86: 86 -> 87, costul: 2332
Muchia 87: 87 -> 88, costul: 5232
Muchia 88: 88 -> 89, costul: 9725
Muchia 89: 89 -> 90, costul: 555
Muchia 90: 90 -> 91, costul: 1792
Muchia 91: 91 -> 92, costul: 1985
Muchia 92: 92 -> 93, costul: 456
Muchia 93: 93 -> 94, costul: 1143
Muchia 94: 94 -> 95, costul: 812
Muchia 95: 95 -> 96, costul: 998
Muchia 96: 96 -> 97, costul: 4776
Muchia 97: 97 -> 98, costul: 4957
Muchia 98: 98 -> 99, costul: 524
Muchia 99: 99 -> 100, costul: 7725

Numarul de iteratii : 198
Timpul de lucru al algoritmului : 0.323000 sec
Press any key to continue . . .

```

Fig 2. Alg.Prim 100 noduri

```

C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Muchia 979: 529 -> 530, costul: 9798
Muchia 980: 203 -> 204, costul: 9808
Muchia 981: 641 -> 642, costul: 9808
Muchia 982: 873 -> 874, costul: 9816
Muchia 983: 626 -> 627, costul: 9830
Muchia 984: 224 -> 225, costul: 9833
Muchia 985: 163 -> 164, costul: 9846
Muchia 986: 14 -> 15, costul: 9848
Muchia 987: 564 -> 565, costul: 9850
Muchia 988: 648 -> 649, costul: 9855
Muchia 989: 370 -> 371, costul: 9867
Muchia 990: 748 -> 749, costul: 9875
Muchia 991: 2 -> 3, costul: 9882
Muchia 992: 714 -> 715, costul: 9886
Muchia 993: 471 -> 472, costul: 9904
Muchia 994: 589 -> 590, costul: 9924
Muchia 995: 179 -> 180, costul: 9925
Muchia 996: 256 -> 257, costul: 9960
Muchia 997: 812 -> 813, costul: 9960
Muchia 998: 458 -> 459, costul: 9964
Muchia 999: 386 -> 387, costul: 9994

Numarul de iteratii : 2988
Timpul de lucru al algoritmului : 9.332000 sec
Press any key to continue . . .

```

Fig 3. Alg.Prim 10 noduri

Fig 4. Alg.Kruskal 1000 noduri

```

C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Muchia 79: 3 -> 4, costul: 7189
Muchia 80: 35 -> 36, costul: 7204
Muchia 81: 56 -> 57, costul: 7221
Muchia 82: 99 -> 100, costul: 7725
Muchia 83: 13 -> 14, costul: 7727
Muchia 84: 41 -> 42, costul: 7997
Muchia 85: 52 -> 53, costul: 8054
Muchia 86: 80 -> 81, costul: 8102
Muchia 87: 4 -> 5, costul: 8177
Muchia 88: 11 -> 12, costul: 8397
Muchia 89: 21 -> 22, costul: 8479
Muchia 90: 60 -> 61, costul: 8884
Muchia 91: 40 -> 41, costul: 9182
Muchia 92: 81 -> 82, costul: 9266
Muchia 93: 38 -> 39, costul: 9402
Muchia 94: 2 -> 3, costul: 9514
Muchia 95: 85 -> 86, costul: 9678
Muchia 96: 88 -> 89, costul: 9725
Muchia 97: 37 -> 38, costul: 9795
Muchia 98: 20 -> 21, costul: 9882
Muchia 99: 25 -> 26, costul: 9986

Numarul de iteratii : 293
Timpul de lucru al algoritmului : 0.313000 sec
Press any key to continue . . .

```

Fig 5. Alg.Kruskal 100 noduri

Fig 6. Alg.Kruskal 10 noduri

Cazul defavorabil

```

C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Muchia 979: 403 -> 192, costul: 7
Muchia 980: 607 -> 325, costul: 34
Muchia 981: 258 -> 177, costul: 35
Muchia 982: 63 -> 661, costul: 36
Muchia 983: 973 -> 969, costul: 36
Muchia 984: 396 -> 395, costul: 38
Muchia 985: 1000 -> 550, costul: 38
Muchia 986: 295 -> 955, costul: 39
Muchia 987: 822 -> 299, costul: 39
Muchia 988: 299 -> 849, costul: 33
Muchia 989: 34 -> 298, costul: 41
Muchia 990: 642 -> 994, costul: 41
Muchia 991: 265 -> 829, costul: 42
Muchia 992: 943 -> 752, costul: 42
Muchia 993: 302 -> 794, costul: 44
Muchia 994: 320 -> 847, costul: 44
Muchia 995: 519 -> 123, costul: 44
Muchia 996: 998 -> 515, costul: 45
Muchia 997: 168 -> 95, costul: 47
Muchia 998: 852 -> 506, costul: 47
Muchia 999: 788 -> 832, costul: 48

Numarul de iteratii : 22337
Timpul de lucru al algoritmului : 21.729000 sec
Press any key to continue . . .

```

Fig 7. Alg.Prim 1000 noduri

```

C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Muchia 979: 65 -> 535, costul: 34
Muchia 980: 325 -> 607, costul: 34
Muchia 981: 403 -> 497, costul: 34
Muchia 982: 177 -> 258, costul: 35
Muchia 983: 63 -> 661, costul: 36
Muchia 984: 969 -> 973, costul: 36
Muchia 985: 395 -> 396, costul: 38
Muchia 986: 550 -> 1000, costul: 38
Muchia 987: 295 -> 955, costul: 39
Muchia 988: 299 -> 822, costul: 39
Muchia 989: 34 -> 298, costul: 41
Muchia 990: 642 -> 994, costul: 41
Muchia 991: 265 -> 829, costul: 42
Muchia 992: 752 -> 943, costul: 42
Muchia 993: 123 -> 519, costul: 44
Muchia 994: 302 -> 794, costul: 44
Muchia 995: 320 -> 847, costul: 44
Muchia 996: 515 -> 998, costul: 45
Muchia 997: 95 -> 168, costul: 47
Muchia 998: 506 -> 852, costul: 47
Muchia 999: 788 -> 832, costul: 48

Numarul de iteratii : 436971
Timpul de lucru al algoritmului : 21.731000 sec
Press any key to continue . . .

```

Fig 8. Alg.Kruskal 1000 noduri

```

C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Muchia 79: 92 -> 25, costul: 48
Muchia 80: 92 -> 88, costul: 119
Muchia 81: 10 -> 36, costul: 148
Muchia 82: 26 -> 18, costul: 168
Muchia 83: 67 -> 15, costul: 169
Muchia 84: 40 -> 57, costul: 170
Muchia 85: 83 -> 28, costul: 175
Muchia 86: 2 -> 41, costul: 191
Muchia 87: 41 -> 8, costul: 75
Muchia 88: 41 -> 14, costul: 195
Muchia 89: 59 -> 82, costul: 199
Muchia 90: 82 -> 24, costul: 41
Muchia 91: 49 -> 42, costul: 227
Muchia 92: 31 -> 61, costul: 231
Muchia 93: 18 -> 52, costul: 236
Muchia 94: 52 -> 98, costul: 144
Muchia 95: 47 -> 44, costul: 238
Muchia 96: 17 -> 76, costul: 241
Muchia 97: 60 -> 85, costul: 252
Muchia 98: 60 -> 99, costul: 260
Muchia 99: 12 -> 70, costul: 448

Numarul de iteratii : 2752
Timpul de lucru al algoritmului : 0.331000 sec
Press any key to continue . . .

```

Fig 9. Alg.Prim 100 noduri

```

C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Muchia 79: 3 -> 68, costul: 142
Muchia 80: 5 -> 43, costul: 142
Muchia 81: 39 -> 67, costul: 142
Muchia 82: 52 -> 98, costul: 144
Muchia 83: 4 -> 21, costul: 145
Muchia 84: 10 -> 36, costul: 148
Muchia 85: 18 -> 26, costul: 168
Muchia 86: 15 -> 67, costul: 169
Muchia 87: 40 -> 57, costul: 170
Muchia 88: 28 -> 83, costul: 175
Muchia 89: 2 -> 41, costul: 191
Muchia 90: 14 -> 41, costul: 195
Muchia 91: 59 -> 82, costul: 199
Muchia 92: 42 -> 49, costul: 227
Muchia 93: 31 -> 61, costul: 231
Muchia 94: 18 -> 52, costul: 236
Muchia 95: 44 -> 47, costul: 238
Muchia 96: 17 -> 76, costul: 241
Muchia 97: 60 -> 85, costul: 252
Muchia 98: 60 -> 99, costul: 260
Muchia 99: 12 -> 70, costul: 448

Numarul de iteratii : 6086
Timpul de lucru al algoritmului : 0.309000 sec
Press any key to continue . . .

```

Fig 10. Alg.Kruskal 100 noduri

```

C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Algoritmul lui Prim
Muchia 1: 1 -> 2, costul: 41
Muchia 2: 2 -> 9, costul: 491
Muchia 3: 9 -> 8, costul: 1322
Muchia 4: 8 -> 10, costul: 333
Muchia 5: 10 -> 3, costul: 153
Muchia 6: 9 -> 5, costul: 1538
Muchia 7: 5 -> 4, costul: 292
Muchia 8: 5 -> 7, costul: 1726
Muchia 9: 4 -> 6, costul: 2382

Numarul de iteratii : 84
Timpul de lucru al algoritmului : 0.049000 sec
Press any key to continue . . .

```

Fig 11. Alg.Prim 10 noduri

```
C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Algoritmul Kruskal
Muchia 1: 1 -> 2, costul: 41
Muchia 2: 3 -> 10, costul: 153
Muchia 3: 4 -> 5, costul: 292
Muchia 4: 8 -> 10, costul: 333
Muchia 5: 2 -> 9, costul: 491
Muchia 6: 8 -> 9, costul: 1322
Muchia 7: 5 -> 9, costul: 1538
Muchia 8: 5 -> 7, costul: 1726
Muchia 9: 4 -> 6, costul: 2382

Numarul de iteratii : 38
Timpul de lucru al algoritmului : 0.031000 sec
Press any key to continue . . .
```

Fig 12. Alg.Kruskal 10 noduri

Cazul mediu

```
C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Algoritmul lui Prim
Muchia 1: 1 -> 2, costul: 41
Muchia 2: 1 -> 10, costul: 169
Muchia 3: 10 -> 5, costul: 141
Muchia 4: 5 -> 8, costul: 153
Muchia 5: 8 -> 9, costul: 35
Muchia 6: 8 -> 7, costul: 299
Muchia 7: 9 -> 6, costul: 322
Muchia 8: 6 -> 3, costul: 382
Muchia 9: 3 -> 4, costul: 145

Numarul de iteratii : 45
Timpul de lucru al algoritmului : 0.033000 sec
Press any key to continue . . .
```

Fig 13. Alg.Prim 10 noduri

```
C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Algoritmul Kruskal
Muchia 1: 8 -> 9, costul: 35
Muchia 2: 1 -> 2, costul: 41
Muchia 3: 10 -> 5, costul: 141
Muchia 4: 3 -> 4, costul: 145
Muchia 5: 5 -> 8, costul: 153
Muchia 6: 1 -> 10, costul: 169
Muchia 7: 3 -> 6, costul: 281
Muchia 8: 8 -> 7, costul: 299
Muchia 9: 9 -> 6, costul: 322

Numarul de iteratii : 31
Timpul de lucru al algoritmului : 0.042000 sec
Press any key to continue . . .
```

Fig 14. Alg.Kruskal 10 noduri

```
C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Muchia 79: 35 -> 60, costul: 15
Muchia 80: 60 -> 93, costul: 33
Muchia 81: 4 -> 37, costul: 41
Muchia 82: 53 -> 48, costul: 41
Muchia 83: 45 -> 66, costul: 42
Muchia 84: 66 -> 15, costul: 5
Muchia 85: 66 -> 57, costul: 6
Muchia 86: 57 -> 90, costul: 7
Muchia 87: 90 -> 99, costul: 28
Muchia 88: 66 -> 89, costul: 43
Muchia 89: 51 -> 56, costul: 45
Muchia 90: 45 -> 28, costul: 47
Muchia 91: 16 -> 67, costul: 49
Muchia 92: 96 -> 79, costul: 49
Muchia 93: 80 -> 87, costul: 50
Muchia 94: 30 -> 95, costul: 56
Muchia 95: 16 -> 77, costul: 60
Muchia 96: 55 -> 74, costul: 63
Muchia 97: 46 -> 23, costul: 65
Muchia 98: 51 -> 8, costul: 83
Muchia 99: 95 -> 18, costul: 104

Numarul de iteratii : 5100
Timpul de lucru al algoritmului : 0.331000 sec
Press any key to continue . . .
```

Fig 15. Alg.Prim 100 noduri

```
C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Muchia 79: 54 -> 61, costul: 18
Muchia 80: 10 -> 3, costul: 19
Muchia 81: 68 -> 33, costul: 19
Muchia 82: 4 -> 49, costul: 20
Muchia 83: 35 -> 42, costul: 20
Muchia 84: 98 -> 17, costul: 20
Muchia 85: 99 -> 22, costul: 20
Muchia 86: 26 -> 41, costul: 21
Muchia 87: 34 -> 7, costul: 21
Muchia 88: 36 -> 45, costul: 22
Muchia 89: 40 -> 85, costul: 22
Muchia 90: 46 -> 65, costul: 22
Muchia 91: 50 -> 81, costul: 26
Muchia 92: 83 -> 50, costul: 26
Muchia 93: 1 -> 78, costul: 35
Muchia 94: 5 -> 82, costul: 38
Muchia 95: 56 -> 89, costul: 38
Muchia 96: 37 -> 32, costul: 40
Muchia 97: 53 -> 48, costul: 41
Muchia 98: 45 -> 28, costul: 47
Muchia 99: 46 -> 23, costul: 65

Numarul de iteratii : 6577
Timpul de lucru al algoritmului : 0.308000 sec
Press any key to continue . . .
```

Fig 16. Alg.Kruskal 100 noduri


```

C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Muchia 979: 949 -> 472, costul: 8
Muchia 980: 472 -> 291, costul: 3
Muchia 981: 24 -> 703, costul: 9
Muchia 982: 194 -> 661, costul: 9
Muchia 983: 330 -> 481, costul: 9
Muchia 984: 481 -> 710, costul: 3
Muchia 985: 529 -> 516, costul: 9
Muchia 986: 531 -> 162, costul: 9
Muchia 987: 667 -> 488, costul: 9
Muchia 988: 724 -> 411, costul: 9
Muchia 989: 985 -> 90, costul: 9
Muchia 990: 250 -> 891, costul: 10
Muchia 991: 502 -> 57, costul: 10
Muchia 992: 757 -> 852, costul: 10
Muchia 993: 380 -> 871, costul: 11
Muchia 994: 598 -> 423, costul: 11
Muchia 995: 872 -> 429, costul: 11
Muchia 996: 174 -> 635, costul: 12
Muchia 997: 107 -> 22, costul: 13
Muchia 998: 106 -> 741, costul: 15
Muchia 999: 588 -> 539, costul: 17

Numarul de iteratii : 63899
Timpul de lucru al algoritmului : 57.399000 sec
Press any key to continue . . .

```

Fig 17. Alg.Prim 1000 noduri

```

C:\Users\Admin\Desktop\3 apa\x64\Debug\3 apa.exe
Muchia 979: 891 -> 428, costul: 4
Muchia 980: 904 -> 33, costul: 4
Muchia 981: 922 -> 691, costul: 4
Muchia 982: 925 -> 48, costul: 4
Muchia 983: 13 -> 554, costul: 5
Muchia 984: 16 -> 161, costul: 5
Muchia 985: 47 -> 130, costul: 5
Muchia 986: 63 -> 786, costul: 5
Muchia 987: 148 -> 691, costul: 5
Muchia 988: 166 -> 325, costul: 5
Muchia 989: 245 -> 860, costul: 5
Muchia 990: 265 -> 352, costul: 5
Muchia 991: 314 -> 801, costul: 5
Muchia 992: 391 -> 590, costul: 5
Muchia 993: 661 -> 288, costul: 5
Muchia 994: 839 -> 168, costul: 5
Muchia 995: 20 -> 389, costul: 6
Muchia 996: 215 -> 598, costul: 6
Muchia 997: 718 -> 671, costul: 6
Muchia 998: 571 -> 180, costul: 7
Muchia 999: 789 -> 234, costul: 7

Numarul de iteratii : 609632
Timpul de lucru al algoritmului : 25.051000 sec
Press any key to continue . . .

```

Fig 18. Alg.Kruskal 1000 noduri

Cazul Favorabil

Algoritmul	Nr.de virfuri	Iteratii	Timpul
Kruskal	1000	2988	9.332000
Kruskal	100	293	0.308000
Kruskal	10	23	0.047000
Prim	1000	1998	9.311000
Prim	100	198	0.323000
Prim	10	18	0.052000

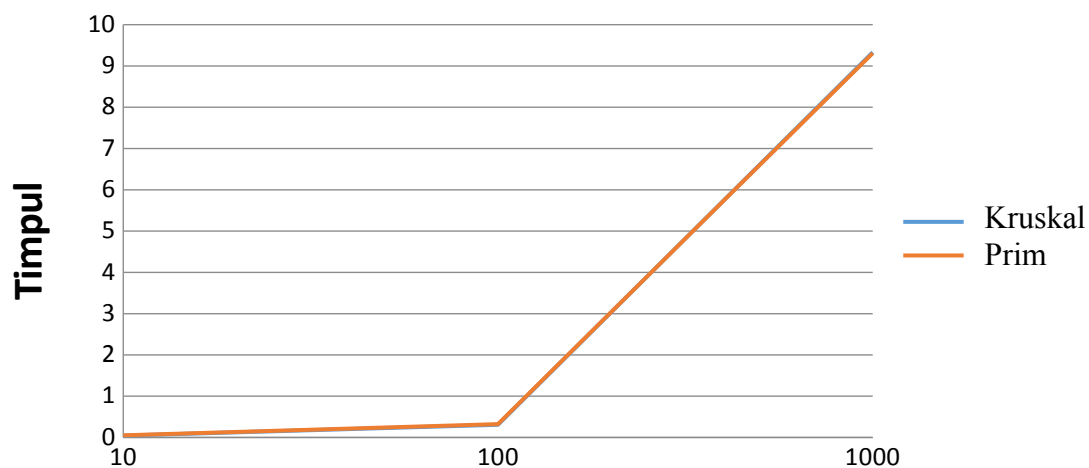
Cazul mediu

Algoritmul	Nr.de virfuri	Iteratii	Timpul
Kruskal	1000	609632	25.051000
Kruskal	100	6577	0.313000
Kruskal	10	31	0.042000
Prim	1000	63899	57.399000
Prim	100	5100	0.331000
Prim	10	45	0.033000

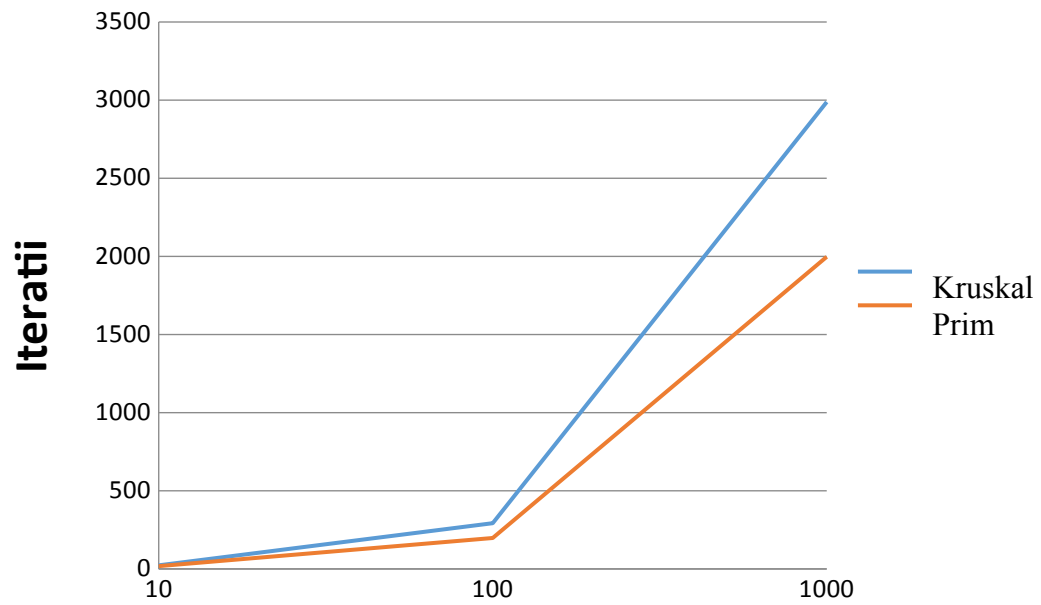
Caz defavorabil

Algoritmul	Nr.de virfuri	Iteratii	Timpul
Kruskal	1000	436971	21.731000
Kruskal	100	6086	0.309000
Kruskal	10	38	0.031000
Prim	1000	22337	21.729000
Prim	100	2752	0.331000
Prim	10	84	0.049000

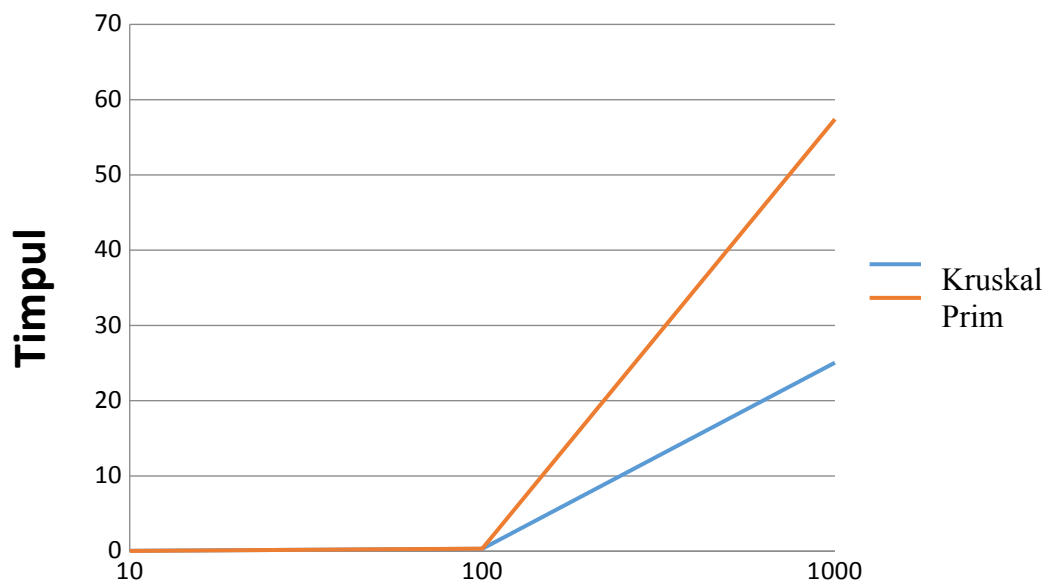
Cazul favorabil



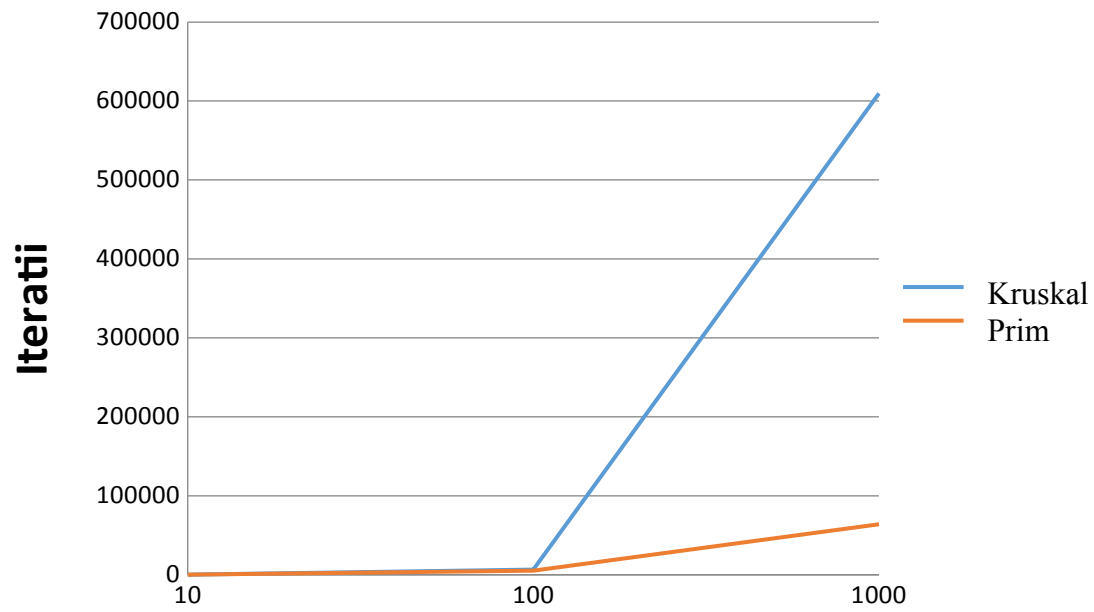
Cazul favorabil



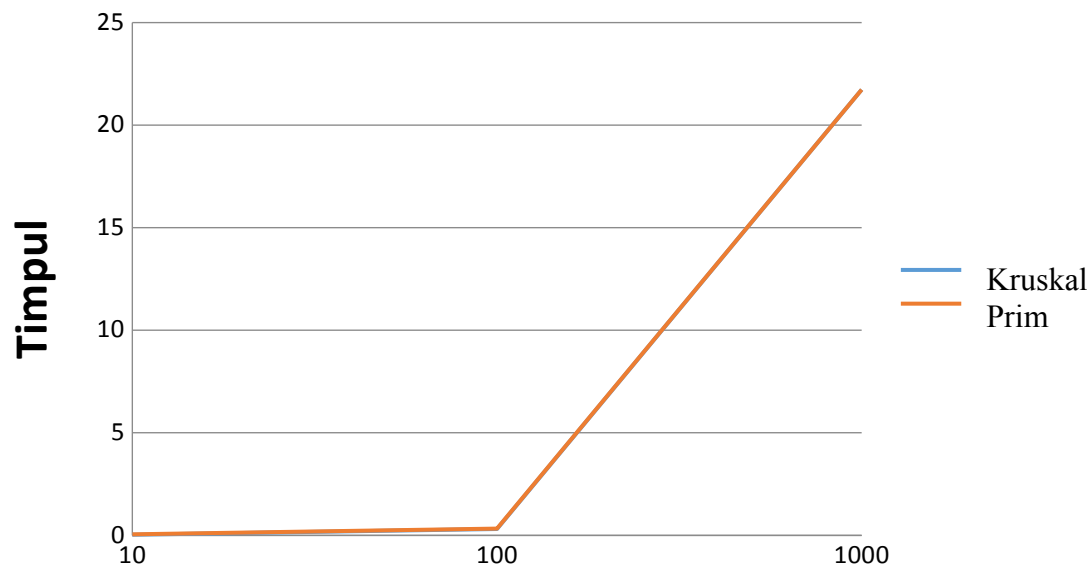
Cazul mediu

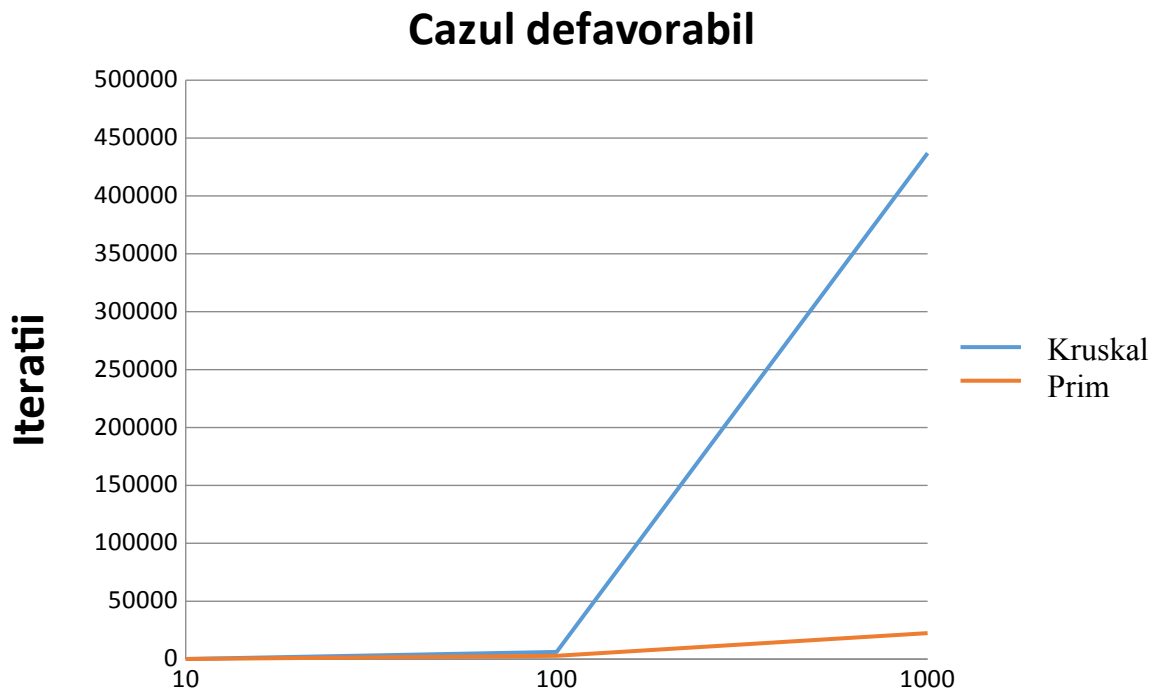


Cazul mediu



Cazul defavorabil





Concluzie

Realizînd laboratorul dat am obținut deprinderi noi în programarea algoritmilor Greedy. După rezultatele obținute se observa ca algoritmul Kruskal este considerabil mai efektiv din punct de vedere al timpului de executie decât algoritmul Prim deoarece programînd algoritmul Prim am folosit tabloul bidimensional dar calculînd complexitatea după pseudocod se observa ca algoritmul Kruskal are un timp mai efektiv decât algoritmul Prim, insa algoritmul Prim este mai efektiv din punct de vedere al iteratiilor , din tabele si grafice se observa ca face mai putine iteratii decit algoritmul Kruskal

