Ministerul Educaţiei și Cercetării

al Republicii Moldova   
Universitatea Tehnică a Moldovei

Faculatea Calculatoare, Infromatică și Microelectronică   
  
  
  
  
  
  
  
RAPORT

# LUCRARE DE LABORATOR NR. 3

**la Analiza şi Proiectarea Algoritmilor**

**Tema: Algoritmi greedy**

A efectuat:

st. gr. TI-211 Popa Cătălin   
  
A verificat:

asist. univ. Andrieschi-Bagrin Veronica

UTM, Chișinău 2022

**Tema:**

Algoritmi greedy.

**Scopul lucrării:**

1.Studierea tehnicii greedy.

2. Analiza şi implementarea algoritmilor greedy.

**Sarcina de bază:**

1.De studiat tehnica greedy de proiectare a algoritmilor.

2.De implementat intr-un limbaj de programare algrotimii Kruskal, Prim.

3.De facut analiza empirica a algoritmilor Kruskal si Prim.

4.De alcatuit un raport.

**Codul C++**

**Algoritmul lui Kruskal**

#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

#define INF 9999

#define edge pair<int, int>

long int it = 0;

class Graph

{

private:

    vector<pair<int, edge>> G;

    vector<pair<int, edge>> T;

    int \*parent;

    int V;

public:

    Graph(int V);

    void AddWeightedEdge(int u, int v, int w);

    int find\_set(int i);

    void union\_set(int u, int v);

    void kruskal();

    void print();

};

Graph::Graph(int V)

{

    parent = new int[V];

    it++;

    for (int i = 0; i < V; i++)

        parent[i] = i;

    it += 4;

    G.clear();

    T.clear();

}

void Graph::AddWeightedEdge(int u, int v, int w)

{

    G.push\_back(make\_pair(w, edge(u, v)));

    it += 4;

}

int Graph::find\_set(int i)

{

    if (i == parent[i])

        return i;

    else

        return find\_set(parent[i]);

}

void Graph::union\_set(int u, int v)

{

    parent[u] = parent[v];

    it++;

}

void Graph::kruskal()

{

    int i, uRep, vRep;

    sort(G.begin(), G.end());

    it += 2;

    for (i = 0; i < G.size(); i++)

    {

        uRep = find\_set(G[i].second.first);

        vRep = find\_set(G[i].second.second);

        it += 5;

        if (uRep != vRep)

        {

            T.push\_back(G[i]);

            union\_set(uRep, vRep);

            it += 3;

        }

    }

}

void Graph::print()

{

    int cost\_min = 0;

    it++;

    cout << "Varful :"

         << " Cost" << endl;

    for (int i = 0; i < T.size(); i++)

    {

        cout << T[i].second.first << " - " << T[i].second.second << " : " << T[i].first;

        cost\_min += T[i].first;

        it += 4;

        cout << endl;

    }

}

void defavorabil\_case(int V)

{

    Graph g(V);

    it++;

    // Defavorabil

    for (int i = 0; i < V; i++)

    {

        for (int j = i; j < V; j++)

        {

            it += 6;

            if (i == j)

            {

                g.AddWeightedEdge(i, j, 0);

                it += 2;

            }

            else

            {

                int r = rand() % 10000;

                int val = (r == V) ? INF : r;

                it += 6;

                g.AddWeightedEdge(i, j, val);

                it++;

            }

        }

    }

    g.kruskal();

    g.print();

    cout << "\nNr de iteratii: " << it;

}

void random\_case(int V)

{

    Graph g(V);

    it++;

    // Random

    for (int i = 0; i < V; i++)

    {

        for (int j = i; j < V; j++)

        {

            it += 6;

            if (i == j)

            {

                g.AddWeightedEdge(i, j, 0);

                it += 2;

            }

            else

            {

                int r = rand() % 10000;

                int val = (r == V) ? INF : r;

                it += 5;

                if (val < 500)

                {

                    g.AddWeightedEdge(i, j, INF);

                    it += 2;

                }

                else

                {

                    g.AddWeightedEdge(i, j, val);

                    it++;

                }

            }

        }

    }

    g.kruskal();

    g.print();

    cout << "\nNr de iteratii: " << it;

}

void favorabil\_case(int V)

{

    Graph g(V);

    it++;

    // favorabil

    for (int i = 0; i < V; i++)

    {

        for (int j = i; j < V; j++)

        {

            it += 6;

            if (i == j)

            {

                g.AddWeightedEdge(i, j, 0);

                g.AddWeightedEdge(j, i, 0);

                it += 2;

            }

            else

            {

                int r = rand() % 10000;

                int val = (r == V) ? INF : r;

                it += 6;

                g.AddWeightedEdge(i, j, val);

                g.AddWeightedEdge(j, i, val);

                it++;

            }

        }

    }

    g.kruskal();

    g.print();

    cout << "\nNr de iteratii: " << it;

}

int main()

{

    int V;

    cout << "\nIntroduceti numarul de varfuri: ";

    cin >> V;

    defavorabil\_case(V);

    cout <<"\n";

    random\_case(V);

    cout <<"\n";

    favorabil\_case(V);

    return 0;

}

**Algoritmul lui Prim**

#include <cstring>

#include <iostream>

#include <stdio.h>

#include <conio.h>

using namespace std;

#define INF 9999999

int main()

{

    long int it = 0;

    int V;

    cout << "\nIntroduceti numarul de varfuri: \n";

    cin >> V;

    int G[V][V];

    // favorabil

    for (int i = 0; i < V; i++)

    {

        for (int j = i; j < V; j++)

        {

            if (i == j)

            {

                G[i][j] = 0;

                G[j][i] = 0;

            }

            else

            {

                int r = rand() % 1000;

                int val = (r == V) ? INF : r;

                G[i][j] = val;

                G[j][i] = val;

            }

        }

    }

    // Random

    // for (int i = 0; i < V; i++)

    // {

    //     for (int j = i; j < V; j++)

    //     {

    //         if (i == j)

    //         {

    //             G[i][j] = 0;

    //         }

    //         else

    //         {

    //             int r = rand() % 1000;

    //             int val = (r == V) ? INF : r;

    //             G[i][j] = val;

    //             G[j][i] = val;

    //             if(G[i][j] < 500)

    //                 {

    //                     G[i][j] = 0;

    //                     G[j][i] = 0;

    //                 }

    //             else

    //                 {

    //                     G[i][j] = val;

    //                     G[j][i] = val;

    //                 }

    //         }

    //     }

    // }

    // Defavorabil

    // for (int i = 0; i < V; i++)

    // {

    //     for (int j = i; j < V; j++)

    //     {

    //         if (i == j)

    //         {

    //             G[i][j] = 0;

    //         }

    //         else

    //         {

    //             int r = rand() % 1000;

    //             int val = (r == V) ? INF : r;

    //             G[i][j] = val;

    //             G[j][i] = val;

    //         }

    //     }

    // }

    // for (int i = 0; i < V; i++)

    // {

    //     it += 3;

    //     for (int j = 0; j < V; j++)

    //     {

    //         cout << "  " << G[i][j];

    //         it += 4;

    //     }

    //     cout << "\n";

    //     it++;

    // }

    int no\_edge;

    int selected[V];

    memset(selected, false, sizeof(selected));

    no\_edge = 0;

    selected[0] = true;

    int x; //  numarul de randuri

    int y; //  numarul de coloane

    cout << "Edge"

         << " : "

         << "Weight";

    cout << endl;

    int temp = 0;

    while (no\_edge < V - 1)

    {

        it += 2;

        int min = INF;

        x = 0;

        y = 0;

        it += 3;

        for (int i = 0; i < V; i++)

        {

            it += 3;

            if (selected[i])

            {

                it++;

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

                {

                    it += 3;

                    if (!selected[j] && G[i][j])

                    {

                        it += 2;

                        if (min > G[i][j])

                        {

                            it++;

                            min = G[i][j];

                            x = i;

                            y = j;

                            it += 3;

                        }

                    }

                }

            }

        }

        cout << x << " - " << y << " :  " << G[x][y];

        cout << endl;

        selected[y] = true;

        no\_edge++;

        // temp += G[x][y];

        // it++;

    }

    // cout << "\nCostul minim: " << temp;

    cout << "\nNumarul de iteratii: " << it;

    return 0;

}

**Rezultatul**

**Concluzie**

In cadrul lucrării date de laborator am efectuat analiza empirica a doi algoritmi de tip Greedy – Prim si Kruskal, care determina graful de acoperire de cost minim.

Mai intai am reprezentat graful sub forma de matrice de adiacenta, in care ponderea este data aleatoriu dupa trei cazuri:favorabil,defavorabil si random. Apoi am implementat pe rand algortimii. Rezultatele obtinute le-am introdus in diagrame pentru a observa care algoritm efectueaza mai multe iteratii si este mai eficient. In cazul meu, cele mai multe interatii sunt la algroitmul lui Prim, cu o deosebire destul de mare fata de cel a lui Kruskal.De asemenea se obsderva ca la algoritmul Kruskal si Prim, ca cazurile favorabile si defavorabile efectueaza mai multe iteratii spre desosebire de cazul mediu. Respectiv, mai eficient este algoritmul Kruskal.