Ministerul Educaţiei și Cercetării

al Republicii Moldova   
Universitatea Tehnică a Moldovei

Faculatea Calculatoare, Infromatică și Microelectronică   
  
  
  
  
  
  
  
RAPORT

# LUCRARE DE LABORATOR NR. 4

**la Analiza şi Proiectarea Algoritmilor**

**Tema:** Metoda programării dinamice

A efectuat: Popa Cătălin

st. gr. TI-211

A verificat: asist. univ.Andrieschi-Bagrin Veronica

UTM, Chișinău 2022

**Tema:**

Metoda programării dinamice.

**Scopul lucrării:**

1. Studierea metodei programării dinamice.

2. Analiza şi implementarea algoritmilor de programare dinamică.

3. Compararea tehnicii greedy cu metoda de programare dinamică.

**Sarcina de bază:**

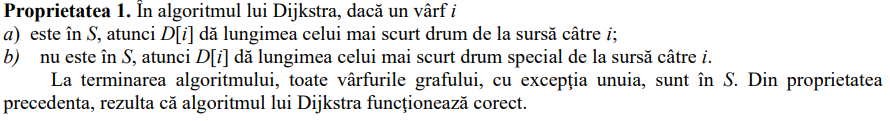
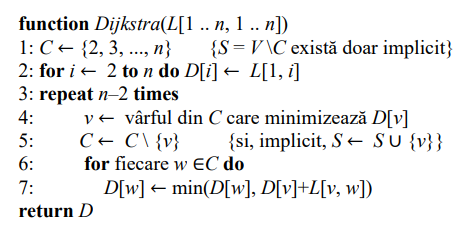
1. De studiat metoda programării dinamice de proiectare a algoritmilor.

2. De implementat într-un limbaj de programare algoritmii prezentaţi mai sus.

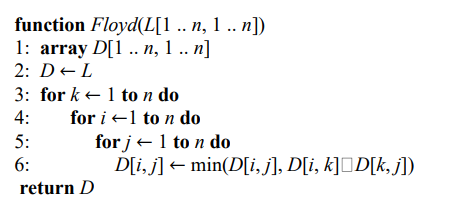
3. De făcut analiza empirică a acestor algoritmi pentru un graf rar şi pentru un graf dens.

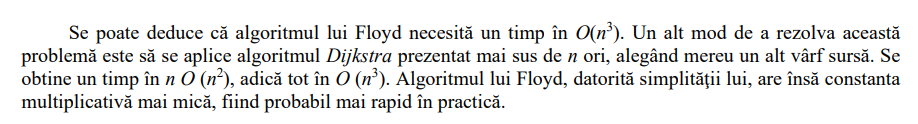
4. De alcătuit un raport.

**Primul algoritm:** Dijkstra**.**



**Al doilea algoritm:** Floyd**.**





**Codul C++**

**Dijkstra**

#include <bits/stdc++.h>

using namespace std;

#define INF 0x3f3f3f3f

long int it = 0;

typedef pair<int, int> iPair;

class Graph

{

    int V;

    list<pair<int, int>> \*adj;

public:

    Graph(int V)

    {

        this->V = V;

        it++;

        adj = new list<iPair>[V];

        it++;

    }

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

    void shortestPathingraph(int s);

};

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

{

    adj[u].push\_back(make\_pair(v, w));

    it++;

    adj[v].push\_back(make\_pair(u, w));

    it++;

}

void Graph::shortestPathingraph(int src)

{

    priority\_queue<iPair, vector<iPair>, greater<iPair>> pq;

    vector<int> dist(V, INF);

    pq.push(make\_pair(0, src));

    dist[src] = 0;

    it++;

    while (!pq.empty())

    {

        int u = pq.top().second;

        it++;

        pq.pop();

        list<pair<int, int>>::iterator i;

        for (i = adj[u].begin(); i != adj[u].end(); ++i)

        {

            it += 3;

            int v = (\*i).first;

            int weight = (\*i).second;

            it += 2;

            if (dist[v] > dist[u] + weight)

            {

                it++;

                dist[v] = dist[u] + weight;

                it++;

                pq.push(make\_pair(dist[v], v));

            }

        }

    }

    cout << "\nVertex \tDistance from source\n";

    for (int i = 0; i < V; ++i)

    {

        cout << i << "------------- " << dist[i] << endl;

    }

}

void defavorabil\_case(int V)

{

    Graph g(V);

    // Defavorabil

    for (int i = 0; i < V; i++)

    {

        for (int j = i; j < V; j++)

        {

            if (i == j)

            {

                g.addEdge(i, j, 0);

            }

            else

            {

                int r = rand() % 10000;

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

                g.addEdge(i, j, val);

            }

        }

    }

    g.shortestPathingraph(0);

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

}

void random\_case(int V)

{

    Graph g(V);

    // Random

    for (int i = 0; i < V; i++)

    {

        for (int j = i; j < V; j++)

        {

            if (i == j)

            {

                g.addEdge(i, j, 0);

            }

            else

            {

                int r = rand() % 10000;

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

                if (val < 500)

                {

                    g.addEdge(i, j, 0);

                    g.addEdge(j, i, 0);

                }

                else

                {

                    g.addEdge(i, j, val);

                    g.addEdge(j, i, val);

                }

            }

        }

    }

    g.shortestPathingraph(0);

    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++)

        {

            if (i == j)

            {

                g.addEdge(i, j, 0);

                g.addEdge(j, i, 0);

            }

            else

            {

                int r = rand() % 10000;

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

                g.addEdge(i, j, val);

                g.addEdge(j, i, val);

            }

        }

    }

    g.shortestPathingraph(0);

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

}

int main()

{

    int arr[] = {5, 25, 45, 65, 85, 100};

    for (int i = 0; i < 6; i++)

    {

        cout << "\nNumarul de varfuri: " << arr[i] << endl;

        defavorabil\_case(arr[i]);

        cout << "\n";

    }

    for (int i = 0; i < 6; i++)

    {

        cout << "\nNumarul de varfuri: " << arr[i] << endl;

        random\_case(arr[i]);

        cout << "\n";

    }

    for (int i = 0; i < 6; i++)

    {

        cout << "\nNumarul de varfuri: " << arr[i] << endl;

        favorabil\_case(arr[i]);

        cout << "\n";

    }

    // g.shortestPathingraph(0); // call the function to find shortest path of graph

    return 0;

}

**Floyd**

#include <iostream>

#include <conio.h>

using namespace std;

#define INF 999

#define V 10

long int it = 0;

void floyds(int b[][V])

{

    for (int k = 0; k < V; k++)

    {

        it += 3;

        for (int i = 0; i < V; i++)

        {

            it += 3;

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

            {

                it += 3;

                if ((b[i][k] \* b[k][j] != 0) && (i != j))

                {

                    it += 3;

                    if ((b[i][k] + b[k][j] < b[i][j]) || (b[i][j] == 0))

                    {

                        it += 3;

                        b[i][j] = b[i][k] + b[k][j];

                        it += 2;

                    }

                }

            }

        }

    }

    for (int i = 0; i < V; i++)

    {

        cout << "\nMinimum Cost With Respect to Node:" << i << endl;

        for (int j = 0; j < V; j++)

        {

            cout << b[i][j] << "\t";

        }

    }

}

void defavorabil\_case()

{

    // Defavorabil

    int b[V][V];

    for (int i = 0; i < V; i++)

    {

        for (int j = i; j < V; j++)

        {

            if (i == j)

            {

                b[i][j] = 0;

            }

            else

            {

                int r = rand() % 1000;

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

                b[i][j] = val;

                b[j][i] = val;

            }

        }

    }

    cout << "\nMatricea: \n";

    for (int i = 0; i < V; i++)

    {

        for (int j = 0; j < V; j++)

        {

            cout << "  " << b[i][j];

        }

        cout << "\n";

    }

    floyds(b);

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

}

void random\_case()

{

    // Random

    int b[V][V];

    for (int i = 0; i < V; i++)

    {

        for (int j = i; j < V; j++)

        {

            if (i == j)

            {

                b[i][j] = 0;

            }

            else

            {

                int r = rand() % 1000;

                int val = (r == V) ? 0 : r;

                if (val < 500)

                {

                    b[i][j] = 0;

                    b[j][i] = 0;

                }

                else

                {

                    b[i][j] = val;

                    b[j][i] = val;

                }

            }

        }

    }

    cout << "\nMatricea: \n";

    for (int i = 0; i < V; i++)

    {

        for (int j = 0; j < V; j++)

        {

            cout << "  " << b[i][j];

        }

        cout << "\n";

    }

    floyds(b);

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

}

void favorabil\_case()

{

   int b[V][V];

    // favorabil

    for (int i = 0; i < V; i++)

    {

        for (int j = i; j < V; j++)

        {

            if (i == j)

            {

                b[i][j] = 0;

                b[j][i] = 0;

            }

            else

            {

                int r = rand() % 10000;

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

                b[i][j] = val;

                b[j][i] = val;

            }

        }

    }

    cout << "\nMatricea: \n";

    for (int i = 0; i < V; i++)

    {

        for (int j = 0; j < V; j++)

        {

            cout << "  " << b[i][j];

        }

        cout << "\n";

    }

    floyds(b);

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

}

int main()

{

  //  defavorabil\_case();

    cout << "\n\n";

    //favorabil\_case();

    cout << "\n\n";

    random\_case();

}

**Rezultatul**

**Concluzie**

In lucrarea data, am realizat lucrarea de laborator nr.4, cu tema: ,,Metoda programarii dinamice’’. Scopul lucrarii a fost de a studia metoda programarii dinamice, analiza si implementarea algoritmilor si de asemenea compararea tehnicii greedy cu metoda de programare dinamica. Pentru a intelege eficienta fiecarui algortim, am relizat printarea numarului de iteratii pentru diferite cantitati de varfuri. In rezultat, observam ca algoritmul Dijkstra efectueaza un numar de iteratii mult mai mic in toate cele trei cazuri(favorabil,defavorabil,random), ceea ce inseamna ca in cazul dat este mai eficient si mai complex. Algortimul Floyd are un numar de iteratii deosebit de mare, chiar in comparatie si cu algoritmii greedy.