|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
|  | | |
| Практическое задание № 1 | | |
| по дисциплине «МПиА» | | |
|  | | |
| **Сравнение Алгоритмов** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМ-81 |
| Бригада: | 7 |
| Студенты: | Бородавкин Кирилл, |
|  | Параев Павел |
|  | Редут Анатолий |
| Преподаватель: | Домников Петр Александрович |
|  |  |
|  | | |
| Новосибирск | | |
| 2019 | | |

1. **Задание к практической работе**

Сравнить варианты алгоритма Дейкстры для поиска всех кратчайших путей при хранении графа в виде матрицы смежности и списков смежности.

1. **Текст программы**

#include <iostream>

#include <fstream>

#include <vector>

#include <stack>

#include <queue>

#include <set>

#include <ctime>

#include <windows.h>

using namespace std;

vector <vector<pair <int, int>>> k;

priority\_queue <pair <int, int>> T;

vector < vector<int>> vecForMatrix;

priority\_queue <pair <int, int>> S;

void Diikstra();

void DiikstraForMatrix();

int maxSize;

void DenseGraphGeneration(int graphSize);

void SparseGraphGeneration(int graphSize);

void DenseGraphGenerationList(int graphSize);

void SparseGraphGenerationList(int graphSize);

int main() // Ввод графа

{

setlocale(LC\_ALL, "Russian");

int m, n, f, num;

fstream fin("C://graph.txt");

if (!fin.is\_open())

cout << "File not open";

else

{

cout << "1 - список смежности" << endl;

cout << "2 - матрица смежности" << endl;

cout << "3 - насыщенный граф(матрица смежности)" << endl;

cout << "4 - разряженный граф(матрица смежности)" << endl;

cout << "5 - насыщенный граф(список смежности)" << endl;

cout << "6 - разряженный граф(список смежности)" << endl;

cin >> num;

switch (num)

{

case 1:

{

fin >> maxSize; // количество рёбер

while (!fin.eof())

{

fin >> m >> n >> f; // f - длина пути / m, n - вершины

k.resize(maxSize);

k[m].push\_back(make\_pair(f, n));

k[n].push\_back(make\_pair(f, m));

}

clock\_t start = clock();

Diikstra();

clock\_t end = clock();

cout << (double)(end - start) / CLOCKS\_PER\_SEC << endl;

break;

}

case 2:

{

fin >> maxSize; // размер матрицы maxSize \* maxSize

vecForMatrix.resize(maxSize);

for (int i = 0; i < maxSize && !fin.eof(); i++)

{

vecForMatrix[i].resize(maxSize);

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

{

fin >> m;

vecForMatrix[i][j] = m;

}

}

clock\_t start = clock();

DiikstraForMatrix();

clock\_t end = clock();

cout << (double)(end - start) / CLOCKS\_PER\_SEC << endl;

break;

}

case 3:

{

cout << "Введите размер графа" << endl;

cin >> maxSize;

DenseGraphGeneration(maxSize);

clock\_t start = clock();

DiikstraForMatrix();

clock\_t end = clock();

cout << (double)(end - start) / CLOCKS\_PER\_SEC << endl;

fin.clear();

break;

}

case 4:

{

cout << "Введите размер графа" << endl;

cin >> maxSize;

SparseGraphGeneration(maxSize);

clock\_t start = clock();

DiikstraForMatrix();

clock\_t end = clock();

cout << (double)(end - start) / CLOCKS\_PER\_SEC << endl;

fin.clear();

break;

}

case 5:

{

cout << "Введите размер графа" << endl;

cin >> maxSize;

DenseGraphGenerationList(maxSize);

clock\_t start = clock();

Diikstra();

clock\_t end = clock();

cout << (double)(end - start) / CLOCKS\_PER\_SEC << endl;

fin.clear();

break;

}

case 6:

{

cout << "Введите размер графа" << endl;

cin >> maxSize;

SparseGraphGenerationList(maxSize);

clock\_t start = clock();

Diikstra();

clock\_t end = clock();

cout << (double)(end - start) / CLOCKS\_PER\_SEC << endl;

fin.clear();

break;

}

}

}

fin.close();

}

void Diikstra()

{

fstream fout;

fout.open("C://out.txt");

if (!fout.is\_open())

cout << "File not open";

else

{

vector <int> dl(k.size(), 0), sel(k.size(), 0);

sel[0] = 0;

vector <bool> visited(k.size(), 0);

T.push(make\_pair(0, 0));

while (!T.empty())

{

int f = T.top().second;

int cur = -T.top().first;

if (sel[f] != -1 && !visited[f] && f != 0)

fout << f + 1 << " " << dl[f] << endl;

visited[f] = true;

T.pop();

for (int j = 0; j < k[f].size(); ++j)

{

int i = k[f][j].second, len = k[f][j].first;

if (!visited[i] && (dl[i] == 0 || dl[f] + len < dl[i]))

{

dl[i] = dl[f] + len;

sel[i] = f;

T.push(make\_pair(-dl[i], i));

}

}

}

fout.clear();

fout.close();

}

}

void DiikstraForMatrix()

{

fstream fout;

fout.open("C://out.txt");

if (!fout.is\_open())

cout << "File not open";

else

{

vector <int> dl(maxSize, 0), sel(maxSize, -1);

sel[0] = 0;

vector <bool> visited(maxSize, 0);

S.push(make\_pair(vecForMatrix[0][0], 0));

while (!S.empty())

{

int f = S.top().second;

int cur = -S.top().first;

if (sel[f] != -1 && !visited[f] && f != 0)

fout << f + 1 << " " << dl[f] << endl;

visited[f] = 1;

S.pop();

for (int j = 0; j < vecForMatrix[f].size(); ++j)

{

int len = vecForMatrix[f][j], i = j;

if (!visited[j] && (dl[j] == 0 || dl[f] + len < dl[j]) && vecForMatrix[f][j] != 0)

{

dl[j] = dl[f] + len;

sel[j] = f;

S.push(make\_pair(-dl[i], i));

}

}

f++;

}

fout.close();

}

}

void DenseGraphGeneration(int graphSize)

{

vecForMatrix.resize(graphSize);

srand(time(nullptr));

int cur = 0;

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

{

vecForMatrix[i].resize(graphSize);

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

{

if (i == 0 && j == 0) // выполняется один раз

{

cur = 1 + rand() % (graphSize - 1);

vecForMatrix[i][cur] = 100 + rand() % 100; // сделать чтобы соединяло с рандомной

}

else

{

if (i == j || i == 0 && j != cur)

vecForMatrix[i][j] = 0;

else

vecForMatrix[i][j] = 100 + rand() % 100;

}

}

}

}

void SparseGraphGeneration(int graphSize)

{

vecForMatrix.resize(graphSize);

srand(time(nullptr));

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

{

vecForMatrix[i].resize(graphSize);

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

{

if (i == j)

vecForMatrix[i][j] = 0;

else

{

if (i + 1 == j)

vecForMatrix[i][j] = rand() % 100;

else

vecForMatrix[i][j] = 0;

}

}

}

}

void DenseGraphGenerationList(int graphSize)

{

srand(time(NULL));

k.resize(graphSize);

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

{

k[i].resize(graphSize);

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

{

if (i != j)

{

int f = rand() % 100;

if (f != 0)

{

k[i].push\_back(make\_pair(f + 1, j));

k[j].push\_back(make\_pair(f + 1, i));

}

}

}

}

}

void SparseGraphGenerationList(int graphSize)

{

srand(time(0));

k.resize(graphSize);

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

{

k[i].resize(graphSize);

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

{

if (i + 1 == j)

{

int f = rand() % 100;

if (f != 0)

{

k[i].push\_back(make\_pair(f, j));

k[j].push\_back(make\_pair(f, i));

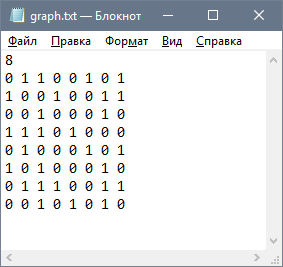
}

}

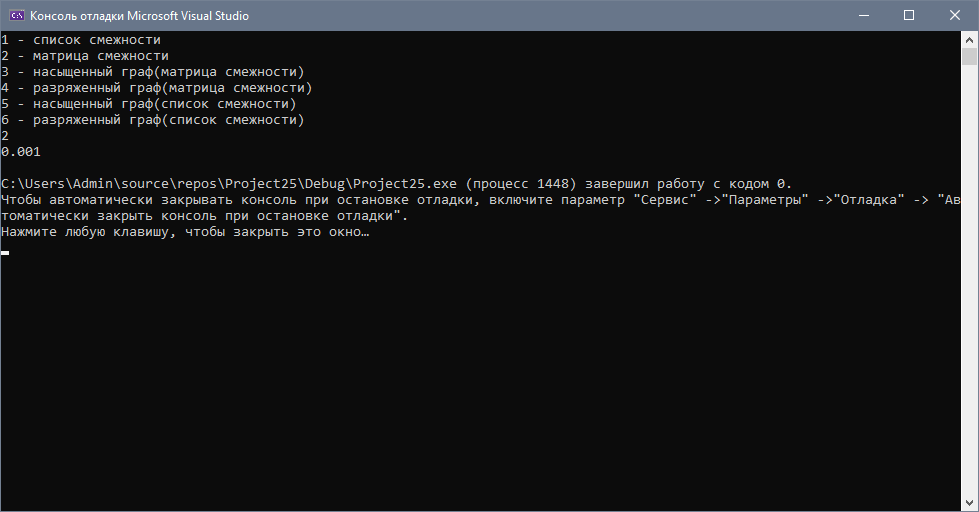
}

}

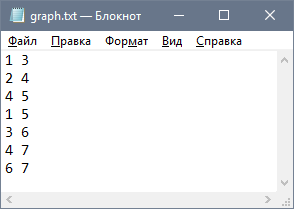
}

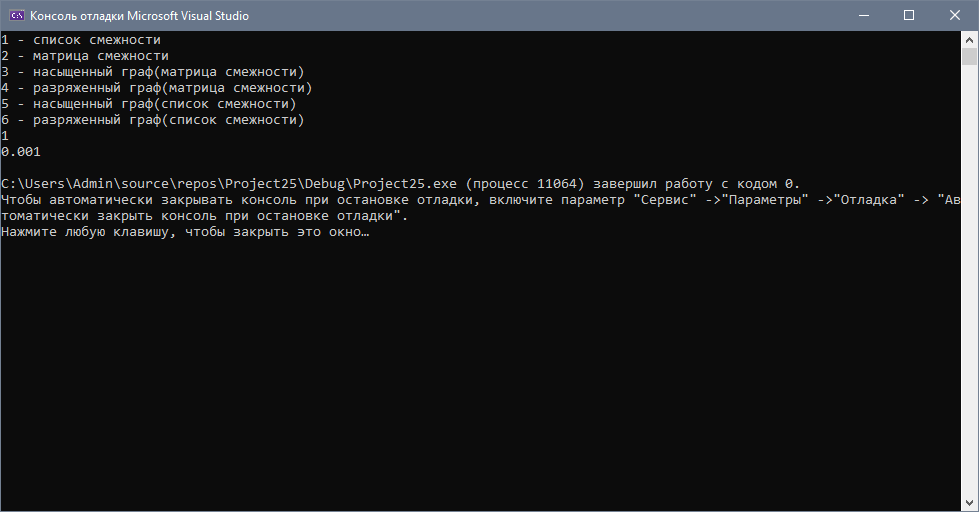
1. **Тесты**

Матрица смежности – размерность 8.



Работает исправно.



Список смежности – 7 вершин. 

Работает исправно.

|  |  |  |  |
| --- | --- | --- | --- |
| **Насыщенный граф** | | | |
| Матрица смежности | | Список смежности | |
| Размер | Время, с | Размер | Время, с |
| 10 | 0.001 | 10 | 0.002 |
| 100 | 0.017 | 100 | 0.13 |
| 1000 | 1.609 | 1000 | 13.449 |
| 5000 | 27.608 | 5000 | 418.146 |
| 10000 | 112.103 | 10000 |  |