Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе № 10  
по дисциплине: «Поиск расстояний во взвешенном графе»

Выполнили студенты группы 22ВВВ2:  
Беляев Д.

Приняли:  
Акифьев И. В.

Митрохин М. А.

Пенза 2023

**Название**

Поиск расстояний во взвешенном графе

**Цель работы**

Разработать и реализовать алгоритм нахождения расстояний во взвешенном графе

**Лабораторное задание**

Задание 1

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного взвешенного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При  реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**3.**\* Сгенерируйте (используя генератор случайных чисел) матрицу смежности для ориентированного взвешенного графа *G*. Выведите матрицу на экран и осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием.

Задание 2

1. Для каждого из вариантов сгенерированных графов (ориентированного и не ориентированного) определите радиус и диаметр.
2. Определите подмножества периферийных и центральных вершин.

Задание 3\*

1. Модернизируйте программу так, чтобы получить возможность запуска программы с параметрами командной строки (см. описание ниже).  В качестве параметра должны указываться тип графа (взвешенный или нет) и наличие ориентации его ребер (есть ориентация или нет).

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <iostream>

#include <stdlib.h>

#include <time.h>

#include <algorithm>

#include <string>

#include <cstring>

#include <queue>

#define INF 10;

//#define CONSOLE

int\*\* GenerateMatrix(int size\_x, int size\_y);

void ClearMatrix(int\*\* matrix, int size);

void PrintMatrix(int\*\* matrix, int size\_x, int size\_y);

int\*\* GenerateMatrixDirected(int size);

int\*\* GenerateMatrixUnDirected(int size);

void BFS(int\*\* matrix, int size, int in, int out);

int FindRadius(int\*\* matrix, int size);

int FindDiametr(int\*\* matrix, int size);

void FindCenterVert(int\*\* matrix, int size);

void FindPerefVert(int\*\* matrix, int size);

int main(int argc, char\* argv[])

{

srand(time(NULL));

const int size = 4;

int in = 0, out = 0;

int\*\* matrixDir = GenerateMatrixDirected(size);

printf("Directed Matrix: \n");

PrintMatrix(matrixDir, size, size);

printf("\nrad = %d, diam = %d\n", FindRadius(matrixDir, size), FindDiametr(matrixDir, size));

printf("Center vert: ");

FindCenterVert(matrixDir, size);

printf("Peref vert: ");

FindPerefVert(matrixDir, size);

int\*\* matrixUnDir = GenerateMatrixUnDirected(size);

printf("\nUnDirected Matrix: \n");

PrintMatrix(matrixUnDir, size, size);

printf("\nrad = %d, diam = %d\n", FindRadius(matrixUnDir, size), FindDiametr(matrixUnDir, size));

printf("Center vert: ");

FindCenterVert(matrixUnDir, size);

printf("Peref vert: ");

FindPerefVert(matrixUnDir, size);

#ifndef CONSOLE

printf("\nInsert in and out vertex: ");

scanf("%d %d", &in, &out);

#else

if (argc > 1)

{

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

{

if (strcmp(argv[i], "-vertIn") == 0)

in = std::stoi(argv[i + 1]);

else if (strcmp(argv[i], "-vertOut") == 0)

out = std::stoi(argv[i + 1]);

}

}

printf("\nIn and out vertex: %d %d", in, out);

#endif

printf("\nWay in direct matrix: ");

BFS(matrixDir, size, in, out);

printf("Way in undirect matrix: ");

BFS(matrixUnDir, size, in, out);

return 0;

}

#pragma region Matrix work

int\*\* GenerateMatrix(int size\_x, int size\_y)

{

int\*\* matrix = (int\*\*)malloc(size\_x \* sizeof(int\*));

if (matrix == NULL)

return NULL;

for (int i = 0; i < size\_x; i++)

{

matrix[i] = (int\*)malloc(size\_y \* sizeof(int));

if (matrix[i] == NULL)

return NULL;

}

for (int i = 0; i < size\_x; i++)

{

for (int j = 0; j < size\_y; j++)

{

matrix[i][j] = 0;

}

}

return matrix;

}

void ClearMatrix(int\*\* matrix, int size)

{

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

{

free(matrix[i]);

}

free(matrix);

}

void PrintMatrix(int\*\* matrix, int size\_x, int size\_y)

{

for (int i = 0; i < size\_x; i++)

{

for (int j = 0; j < size\_y; j++)

{

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

#pragma endregion

#pragma region MatrixGenerate

int\*\* GenerateMatrixUnDirected(int size)

{

int\*\* matrix = GenerateMatrix(size, size);

if (matrix == NULL)

return NULL;

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

{

for (int j = i; j < size; j++)

{

if (i == j)

continue;

int rnd = rand() % INF;

int isZero = rand() % 2;

if (isZero == 0)

rnd = 0;

matrix[i][j] = rnd;

matrix[j][i] = rnd;

}

}

return matrix;

}

int\*\* GenerateMatrixDirected(int size)

{

int\*\* matrix = GenerateMatrix(size, size);

if (matrix == NULL)

return NULL;

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

{

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

{

if (i == j)

continue;

int rnd = rand() % INF;

int side = rand() % 3;

switch (side)

{

case 0:

matrix[i][j] = rnd;

break;

case 1:

matrix[j][i] = rnd;

break;

case 2:

matrix[i][j] = rnd;

matrix[j][i] = rnd;

break;

}

}

}

return matrix;

}

#pragma endregion

#pragma region FirsNumber

std::vector<int> BFSLogic(int\*\* matrix, int size, int in, int out)

{

std::vector<int> dst (size, INT\_MAX);

std::vector<int> pr (size, -1);

std::priority\_queue<std::pair<int, int>, std::vector<std::pair<int, int>>, std::greater< std::pair<int, int>>> q;

dst[in] = 0;

q.push({ 0, in });

while (!q.empty())

{

std::pair<int, int> tmp = q.top();

q.pop();

int len = tmp.first;

int vert = tmp.second;

if (dst[vert] < len)

continue;

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

{

if (matrix[vert][i] != 0)

{

int cur = i;

int lenCur = matrix[vert][i];

int nLen = len + lenCur;

if (nLen < dst[cur])

{

dst[cur] = nLen;

pr[cur] = vert;

q.push({nLen, cur});

}

}

}

}

int cur = out;

std::vector<int> path;

path.push\_back(cur);

while (pr[cur] != -1)

{

cur = pr[cur];

path.push\_back(cur);

}

int len = 0;

path.push\_back(len);

std::reverse(path.begin(), path.end());

for (int i = 1; i < path.size() - 1; i++)

len += matrix[path[i]][path[i + 1]];

path[0] = len;

return path;

}

void BFS(int\*\* matrix, int size, int in, int out)

{

std::vector<int> path = BFSLogic(matrix, size, in, out);

if (path[0] <= 0)

printf("There is no way");

else

{

printf("Way: ");

for (int i = 1; i < path.size(); i++)

{

printf("%d ", path[i]);

}

printf("Len: %d", path[0]);

}

printf("\n");

}

#pragma endregion

#pragma region SecondNumber

std::vector<int> MaxWeight(int\*\* matrix, int size)

{

std::vector<int> m(size, 0);

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

{

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

{

int len = BFSLogic(matrix, size, i, j)[0];

m[i] = std::max(m[i], len);

}

}

return m;

}

int FindRadius(int\*\* matrix, int size)

{

int rad = INF + 1;

std::vector<int> m = MaxWeight(matrix, size);

for (size\_t i = 0; i < size; i++)

{

rad = std::min(m[i], rad);

}

return rad;

}

int FindDiametr(int\*\* matrix, int size)

{

int diam = 0;

std::vector<int> m = MaxWeight(matrix, size);

for (size\_t i = 0; i < size; i++)

{

diam = std::max(m[i], diam);

}

return diam;

}

void FindCenterVert(int\*\* matrix, int size)

{

std::vector<int> m = MaxWeight(matrix, size);

int rad = FindRadius(matrix, size);

for (size\_t i = 0; i < size; i++)

{

if (m[i] == rad)

printf("%d ", i);

}

printf("\n");

}

void FindPerefVert(int\*\* matrix, int size)

{

std::vector<int> m = MaxWeight(matrix, size);

int diam = FindDiametr(matrix, size);

for (size\_t i = 0; i < size; i++)

{

if (m[i] == diam)

printf("%d ", i);

}

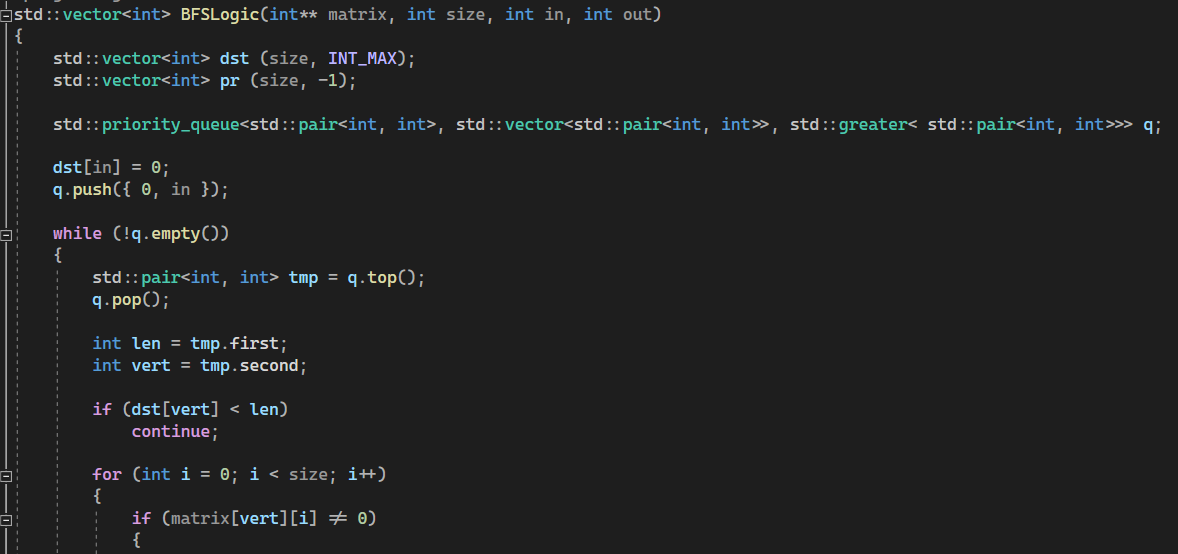
printf("\n");

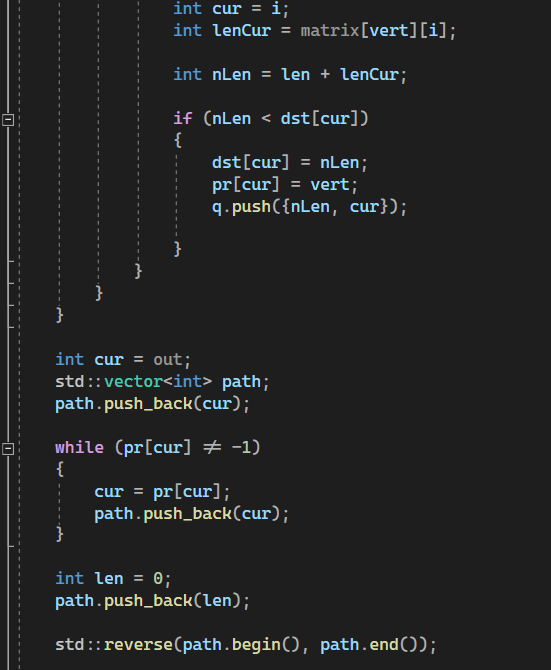
}

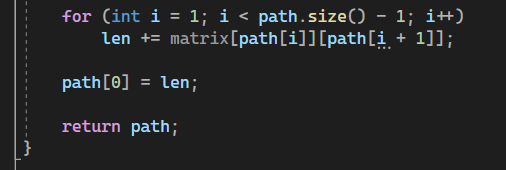
#pragma endregion

**Задания**

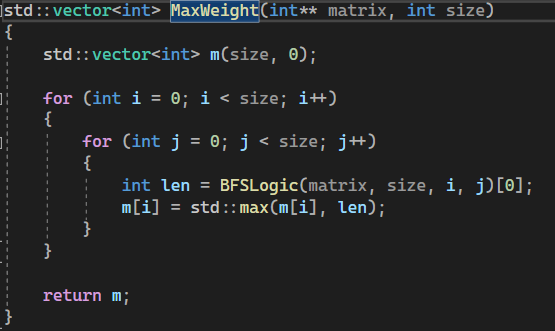
Задание 1

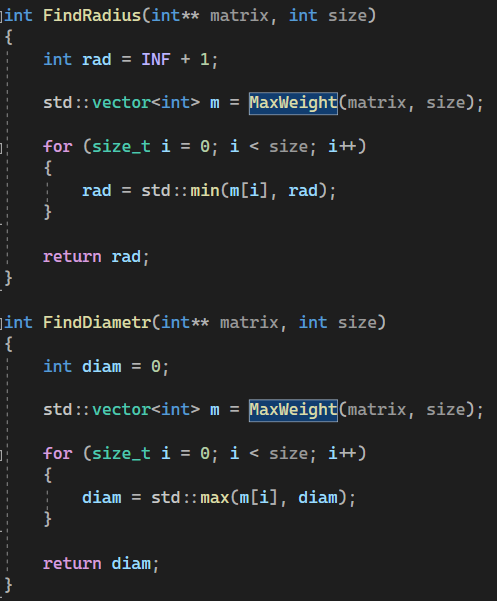
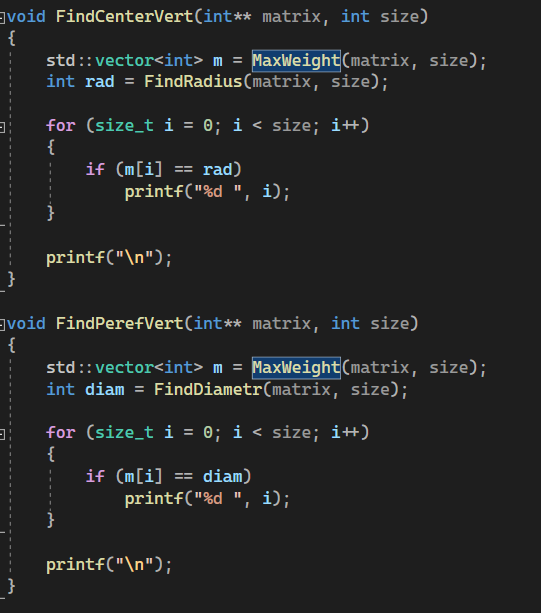




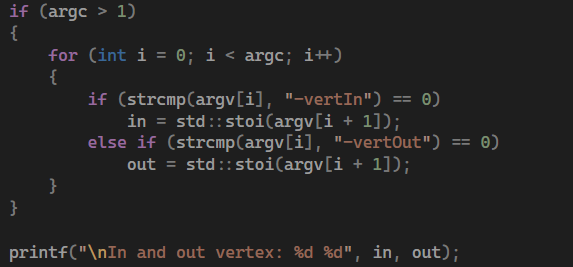


Задание 2

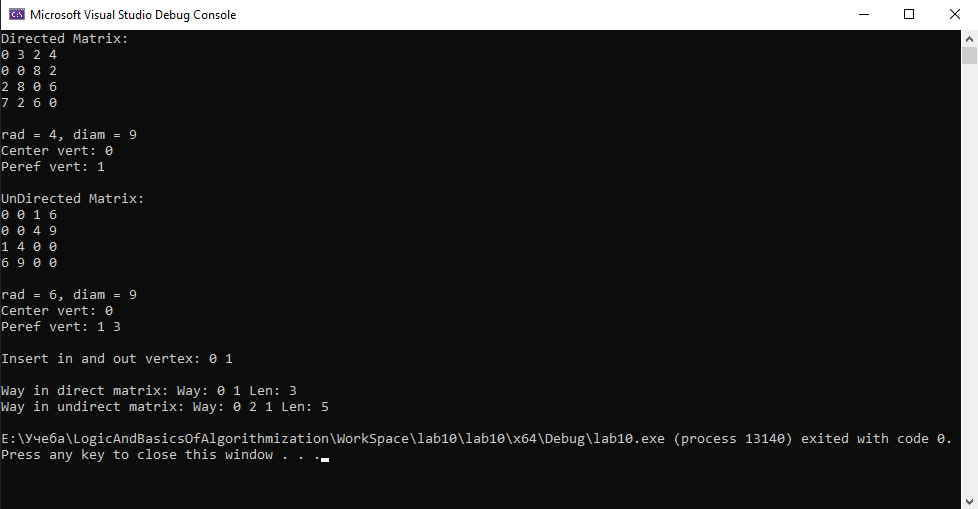


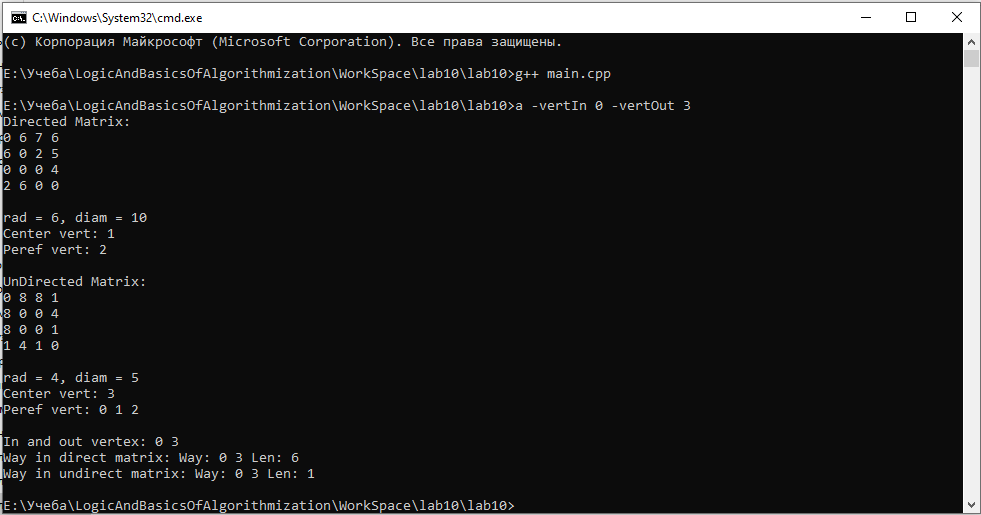
 

Задание 3



**Результат работы программы**

****

****

**Вывод**

Разработал и реализовал алгоритм нахождений расстояний во взвешенном графе