Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №7

# по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний во взвешенном графе»

**Выполнили:**

студент группы 20ВВ3

Пантюшов Егор

**Приняли:**

Юрова О.В.

Митрохин М.А.

Пенза 2021

# Название

Поиск расстояний во взвешенном графе

**Цель работы** – изучение алгоритма поиска расстояний во взвешенном графе.

# Методические указания

Отличие от поиска расстояний в не взвешенном графе будет состоять в том, что при обновлении расстояния до вершины при ее посещении оно будет увеличиваться не на 1, а на величину веса ребра.

# 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. 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).

**Псевдокод**

Вход: G – матрица смежности графа, v – исходная вершина.

Выход: DIST – вектор расстояний до всех вершин от исходной.

Алгоритм ПОШ

1.1. для всех i положим DIST [i] = 1000 пометим как &quot;не посещенную&quot;;

1.2. ВЫПОЛНЯТЬ BFSD (v).

1.3 для всех i вывести DIST [i] на экран;

Алгоритм BFSD(v):

2.1. Создать пустую очередь Q = {};

2.2. Поместить v в очередь Q.push(v);

2.3. Обновить вектор расстояний DIST [ x ] = 0;

2.4. ПОКА Q != ∅ очередь не пуста ВЫПОЛНЯТЬ

2.5. v = Q.front() установить текущую вершину;

2.6. Удалить первый элемент из очереди Q.pop();

2.7. вывести на экран v;

2.8. ДЛЯ i = 1 ДО size\_G ВЫПОЛНЯТЬ

2.9. ЕСЛИ G(v,i) &gt; 0 И DIST [ i ] &gt; DIST [ v ] + G(v,i)

2.10. ТО

2.11. Поместить i в очередь Q.push(i);

2.12. Обновить вектор расстояний DIST [ i ] = DIST [ v ] +

G(v,i);

# Листинг

#define \_CRT\_NONSTDC\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNINGS

#define HEADER ("Лабораторная работа №7\nВыполнил:Пантюшов Е.\n\n")

#include <queue>

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

#include <stack>

#include <limits.h>

using namespace std;

void bfs(int\*\* G, int\* DIST, int v, int n)

{

queue <int> q;

q.push(v);

DIST[v] = 0;

while (!q.empty())

{

v = q.front();

q.pop();

printf(" %d ", v);

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

{

if ((G[v][i] > 0) && (DIST[i] > DIST[v] + G[v][i]))

{

q.push(i);

DIST[i] = DIST[v] + G[v][i];

}

}

}

}

void v\_graph(int\*\* G, int n)

{

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

{

G[i] = (int\*)malloc(n \* sizeof(int));

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

{

G[i][j] = rand() % 10;

if (G[i][j] < 5)

{

G[i][j] = 0;

}

else

{

G[i][j] = rand() % 5;

}

}

}

}

void nv\_graph(int\*\* G, int n)

{

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

{

G[i] = (int\*)malloc(n \* sizeof(int));

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

{

G[i][j] = rand() % 2;

}

}

}

void no\_graph(int\*\* G, int n)

{

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

{

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

{

if (i == j)

{

G[i][j] = 0;

}

else

{

G[i][j] = G[j][i];

}

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

}

printf("\n");

}

}

void o\_graph(int\*\* G, int n)

{

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

{

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

{

if (i == j)

{

G[i][j] = 0;

}

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

}

printf("\n");

}

}

void task(int\*\* G, int n)

{

int\* DIST = (int\*)malloc(n \* sizeof(int));

int v;

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

{

DIST[i] = INT\_MAX;

}

printf("\nВведите точку входа: ");

scanf("%d", &v);

printf("Результат работы алгоритма обхода в ширину(матрица):\n ");

bfs(G, DIST, v, n);

printf("\n");

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

{

if (DIST[i] == INT\_MAX)

{

DIST[i] = 0;

}

printf("Кратчайшее расстояние от %dой вершины до %dой вершины=%d\n", v, i, DIST[i]);

}

free(DIST);

}

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

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

printf(HEADER);

printf("Введите размерность матрицы: ");

int n;

scanf("%d", &n);

int\*\* G = (int\*\*)malloc(n \* sizeof(int\*));

if ((strcmp(argv[1], "-v") == 0) && (argc > 1))

{

v\_graph(G, n);

}

else

{

nv\_graph(G, n);

}

if ((strcmp(argv[2], "-o") == 0) && (argc > 1))

{

o\_graph(G, n);

}

else

{

no\_graph(G, n);

}

task(G, n);

system("PAUSE");

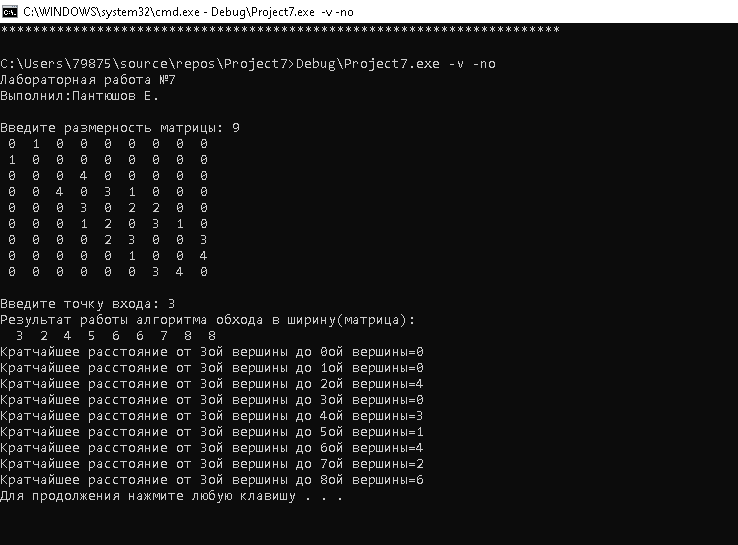
return 0;

}

# Результат работы программы

**Пункт 1.1-2 и 2.1**

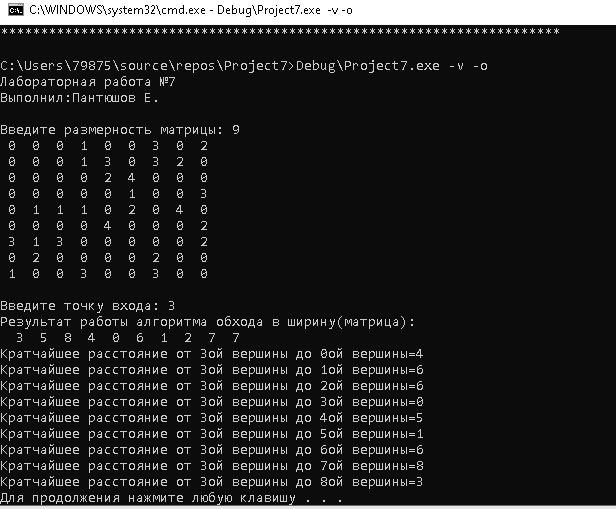
Результаты работы программы показаны на рисунке 1.



**Рисунок 1 – Результат работы программы**

**Пункт 1.3.-2.1**

Результаты работы программы показаны на рисунке 2.



**Рисунок 2 – Результат работы программы**

# Вывод

# В ходе выполнения лабораторной работы была разработана программа, в который был реализован алгоритм поиска расстояний в графа на основе алгоритма поиска в ширину. Также были изучены основные параметры функции main() и были приобретены навыки по запуску программы с выбранными параметрами командной строки.