Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

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

на тему «Поиск расстояний в графе»

**Выполнили:**

студенты группы 20ВВ3

Пантюшов Егор

**Приняли:**

Юрова О.В.

Митрохин М.А.

Пенза 2021

# Название

Поиск расстояний в графе

**Цель работы** – изучение алгоритма поиска расстояний в графе.

# Методические указания

Поиск расстояний – довольно распространенная задача анализа графов.

Для поиска расстояний можно использовать процедуры обхода графа.

Для этого при каждом переходе в новую вершину необходимо запоминать, сколько шагов до нее мы сделали. При этом вектор, который хранил информацию о посещении вершин становится вектором расстояний.

# Лабораторное задание

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу

смежности для неориентированного 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++.

**Задание 2\***

1. Реализуйте процедуру поиска расстояний на основе обхода в глубину.

**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] = -1 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 != Æ 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) = = 1И DIST = = -1

2.10. ТО

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

2.12. Обновить вектор расстояний DIST [ i ] = DIST [ v ] + 1;

# Листинг

#define \_CRT\_NONSTDC\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNINGS

#define HEADER ("Лабораторная работа №6\nВыполнил:Пантюшов Е.\n\n")

#include <queue>

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

#include <stack>

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] == 1) && (DIST[i] == -1))

{

q.push(i);

DIST[i] = 1+DIST[v];

}

}

}

}

void dfs(int\*\* G, int\* DIST, int v, int a, int n)

{

int rez = a + 1;

DIST[v] = a;

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

{

if (G[v][i] == 1 && DIST[i] == -1)

dfs(G, DIST,i,rez,n);

if (G[v][i] == 1 && a < DIST[i])

dfs(G, DIST, i, rez, n);

}

}

void task\_1\_1(int\*\* G, int n)

{

printf("\nЗадание 1.\n\nПункт 1.\nМатрица смежности:\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;

}

}

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 task\_1\_2(int\*\* G, int n)

{

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

int v;

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

{

DIST[i] = -1;

}

printf("\nПункт 1.2.\nВведите точку входа: ");

scanf("%d", &v);

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

bfs(G, DIST, v, n);

printf("\n");

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

{

if (DIST[i] == -1)

{

DIST[i] = 0;

}

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

}

free(DIST);

}

void task\_2\_1(int\*\* G, int n)

{

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

int v = 0;

int a = 0;

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

{

DIST[i] = -1;

}

printf("\nПункт 2.1.\nВведите точку входа: ");

scanf("%d", &v);

printf("Результат работы алгоритма обхода в глубину(матрица): ");

dfs(G,DIST,v,a,n);

printf("\n");

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

{

if (DIST[i] == -1)

{

DIST[i] = 0;

}

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

}

free(DIST);

}

int main()

{

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\*));

task\_1\_1(G, n);

task\_1\_2(G, n);

task\_2\_1(G, n);

system("PAUSE");

return 0;

}

# Результат работы программы

**Пункт 1.1.**

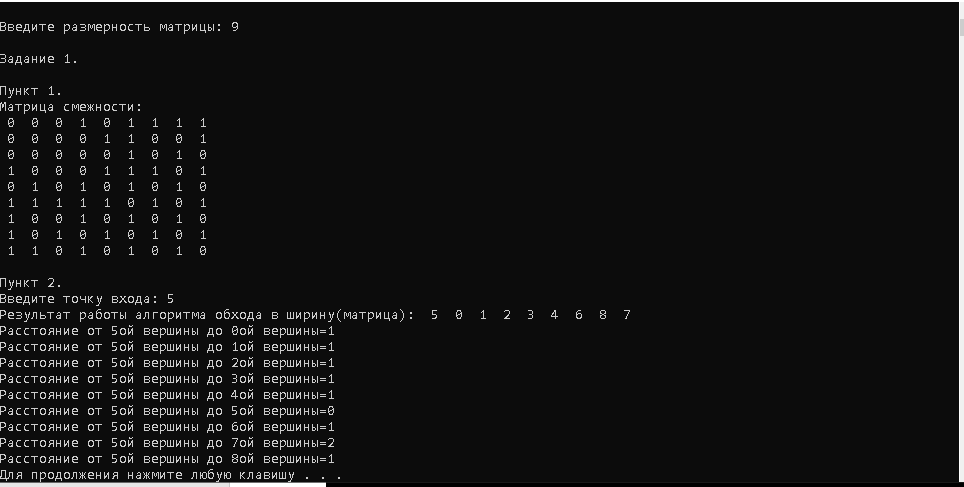
Результаты работы программы показаны на рисунке 1.



**Рисунок 1 – Результат работы программы**

**Пункт 1.2.**

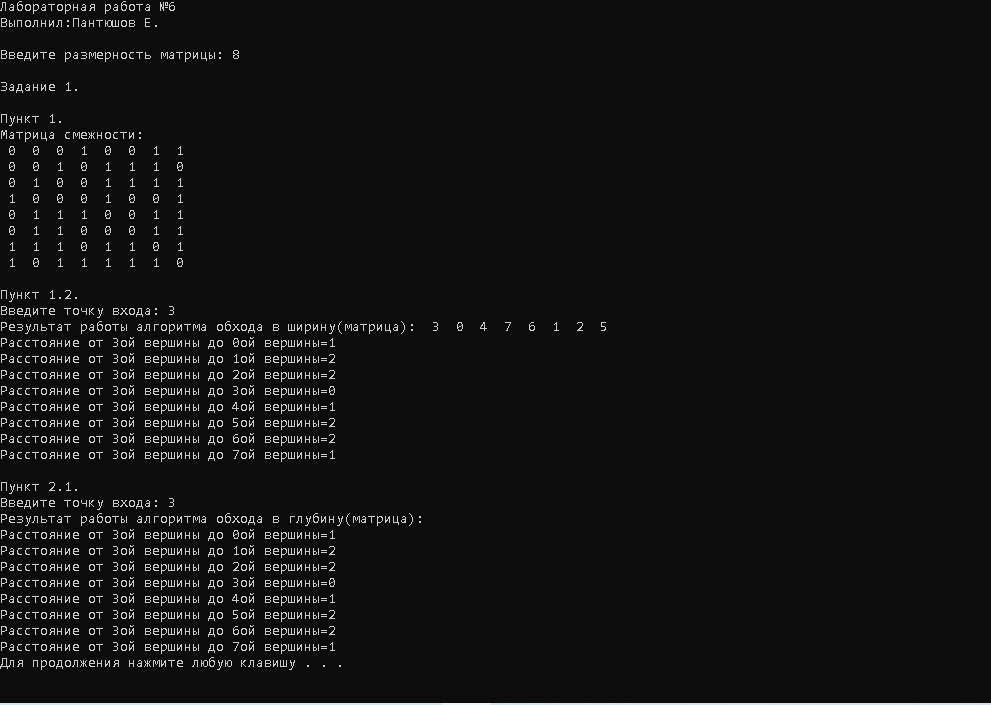
Результаты работы программы показаны на рисунке 2.



**Рисунок 2 – Результат работы программы**

**Пункт 2.1.**

Результаты работы программы показаны на рисунке 3.



**Рисунок 3 – Результат работы программы**

# Вывод

# В ходе выполнения лабораторной работы была разработана программа, в который были реализованы алгоритмы поиска расстояний в графе, используя поиски в ширину и глубину. При разработке данного алгоритма были применены такие структуры данных как очередь и стек.