Министерство образования Российской Федерации

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 студенты группы 20ВВ3:

Мальков И.А.

Педай Н.Д.

Приняли:

Митрохин М.А.

Юрова О.В.

Пенза 2021

**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).

**Ход работы:**

1. Код программы

#include <stdio.h>

#include <iostream>

#include <locale.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <queue>

using namespace std;

void BFSD(int\*\* G, int n, int\* dist, int s)

{

queue<int> q;

q.push(s);

dist[s] = 0;

while (!q.empty())

{

s = q.front();

q.pop();

cout << s << " ";

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

{

if (G[s][j] > 0 && dist[j] > dist[s] + G[s][j])

{

dist[j] = dist[s] + G[s][j];

q.push(j);

}

}

}

}

int\*\* create(int n)

{

int\*\* G, k = 1;

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

srand(time(NULL));

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

{

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

G[i][i] = 0;

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

{

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

}

k++;

}

k = 1;

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

{

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

{

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

}

k++;

}

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

cout << "V" << i << " ";

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

{

cout << endl;

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

{

cout << G[i][j] << " ";

}

}

return G;

}

int\*\* create\_o(int n)

{

int\*\* G, k = 1;

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

srand(time(NULL));

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

{

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

G[i][i] = 0;

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

{

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

}

k++;

}

k = 1;

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

{

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

{

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

}

k++;

}

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

cout << "V" << i << " ";

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

{

cout << endl;

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

{

cout << G[i][j] << " ";

}

}

return G;

}

int\*\* create\_on(int n)

{

int\*\* G, k = 1;

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

srand(time(NULL));

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

{

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

G[i][i] = 0;

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

{

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

}

k++;

}

k = 1;

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

{

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

{

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

}

k++;

}

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

cout << "V" << i << " ";

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

{

cout << endl;

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

{

cout << G[i][j] << " ";

}

}

return G;

}

int\*\* create\_n(int n)

{

int\*\* G, k = 1;

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

srand(time(NULL));

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

{

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

G[i][i] = 0;

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

{

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

}

k++;

}

k = 1;

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

{

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

{

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

}

k++;

}

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

cout << "V" << i << " ";

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

{

cout << endl;

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

{

cout << G[i][j] << " ";

}

}

return G;

}

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

{

int n, s, \*\* t, \*\* t1, \*\* t2, \*\* t3;

cout << "Enter array size: ";

cin >> n;

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

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

dist[i] = INT16\_MAX;

if(argc >= 1){

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

if ((strcmp(argv[i], "0") == 0)){ //неориентированный взвешенный

cout << "неориентированный взвешенный";

cout << endl;

t = create(n);

cout << endl;

cout << "Enter vertex: ";

cin >> s;

cout << "BFSD result: ";

BFSD(t, n, dist, s);

cout << endl;

}

if ((strcmp(argv[i], "1") == 0)){ //ориентированный взвешенный

cout << "ориентированный взвешенный";

cout << endl;

t1 = create\_o(n);

cout << endl;

cout << "Enter vertex: ";

cin >> s;

cout << "BFSD result: ";

BFSD(t1, n, dist, s);

cout << endl;

}

if ((strcmp(argv[i], "2") == 0)){ //ориентированный невзвешенный

cout << "ориентированный невзвешенный";

cout << endl;

t2 = create\_on(n);

cout << endl;

cout << "Enter vertex: ";

cin >> s;

cout << "BFSD result: ";

BFSD(t2, n, dist, s);

cout << endl;

}

if ((strcmp(argv[i], "3") == 0)){ //неориентированный невзвешенный

cout << "неориентированный невзвешенный";

cout << endl;

t3 = create\_n(n);

cout << endl;

cout << "Enter vertex: ";

cin >> s;

cout << "BFSD result: ";

BFSD(t3, n, dist, s);

cout << endl;

}

}

}

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

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

cout << endl;

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

free(t3[i]);

free(t3);

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

free(t2[i]);

free(t2);

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

free(t1[i]);

free(t1);

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

free(t[i]);

free(t);

free(dist);

system("pause");

return 0;

}

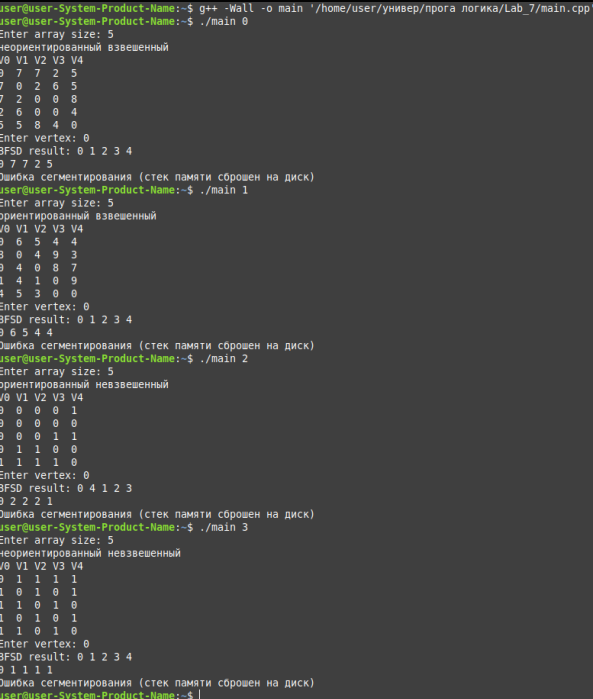
1. Запуск программы с параметрами через терминал:

Неориентрованный взвешенный

Ориентированный взвешенный

Ориентированный невзвешенный

Неориентрованный невзвешенный



Вывод: в ходе выполнения данной лабораторной работы разработали программу, согласно заданию.