Пензенский государственный университет  
Кафедра «Вычислительной техники»

**Отчет**по лабораторной работе №7  
по дисциплине: «Логика и основы алгоритмизации в инженерных задачах»  
на тему: «Поиск расстояний во взвешенном графе»

**Выполнил студент группы 19ВВ1:**

Тебнев Р.

**Приняли:**

д.т.н. профессор Митрохин М. А.

Юрова О.В.

Пенза 2020

**Цель работы:** изучить поиск расстояний во взвешенном графе с помощью обхода в ширину.

**Теория**

Во взвешенном графе в отличие от не взвешенного каждое ребро имеет

вес, отличный от нуля. Поэтому в матрице смежности взвешенного графа

содержится информация не только о наличии ребра, но и о его весе.

Поиск расстояний между вершинами в таком графе также возможно

построить используя процедуры обхода графа. 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.

Таким образом, можно предложить следующую реализацию алгоритма

обхода в ширину.

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);

**Код программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <time.h>

#include <locale.h>

#include <queue>

using namespace std;

void NewGraphNeOr(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;

}

if (i < j) {

if (rand() % 100 > 50) {

G[i][j] = rand() % 15;

}

else {

G[i][j] = 0;

}

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

}

}

}

}

void NewGraphOr(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;

}

if (i < j) {

if (rand() % 100 > 50) {

G[i][j] = rand() % 15;

}

else {

G[j][i] = rand() % 15;

}

}

}

}

}

void PrintGraph(int\*\* G, int n) {

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

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

printf("%3d ", G[i][j]);

}

printf("\n");

}

printf("\n");

}

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

queue <int> q;

q.push(s);

dist[s] = 0;

while (!q.empty()) {

s = q.front();

q.pop();

printf("%d ", s);

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

if ((G[s][i] > 0) && (dist[i] == 1000)) {

q.push(i);

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

}

}

}

printf("\n");

}

int main() {

int\*\* G; // указатель на указатель на строку элементов

int\* dist;

int i, j, n;

setlocale(LC\_ALL, "rus");

srand(time(NULL));

printf("Введите размер графа: ");

scanf("%d", &n);

printf("\n");

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

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

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

}

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

printf("Неориентированный взвешенный граф \n");

NewGraphNeOr(G, n);

PrintGraph(G, n);

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

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

dist[i] = 1000;

}

printf("от вершины %d: ", j);

BFSD(j, G, dist, n);

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

if (dist[i] != 1000)

printf("%d\t", dist[i]);

else

printf("нет прохода\t");

}

printf("\n\n");

}

printf("Ориентированный взвешенный граф \n");

NewGraphOr(G, n);

PrintGraph(G, n);

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

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

dist[i] = 1000;

}

printf("от вершины %d: ", j);

BFSD(j, G, dist, n);

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

if (dist[i] != 1000)

printf("%d\t", dist[i]);

else

printf("нет прохода\t");

}

printf("\n\n");

}

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

free(G[i]);

}

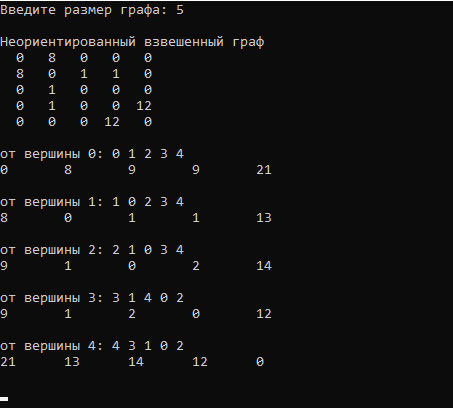
free(G);

free(dist);

getchar();

getchar();

}**Результат программы**



**Вывод:** изучил поиск расстояния во взвешенном неориентированном графе между вершинами с помощью обхода в ширину.