Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе № 10

по дисциплине: "Логика и ОА в ИЗ"

## на тему: " Поиск расстояний во взвешенном графе"

Выполнили:

Трундов Н.А.

Евдокимов Р.E.

Приняли:

Акифьев И.В.

Юрова О.В.

Пенза, 2023

**Общие сведения.**

### 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.

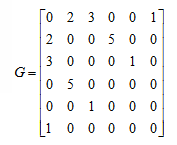
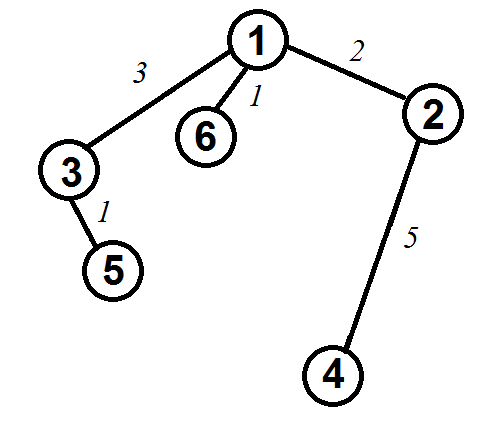


Рисунок 1 – Граф

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] = -1 пометим как "не посещенную";

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) > 0 **И** DIST = = -1

2.10. **ТО**

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

2.12. Обновить вектор расстояний DIST [ i ] = DIST [ v ] + G(v,i);

Реализация состоит из подготовительной части, в которой все вершины помечаются как не поcещенные (п.1.1). Не посещенные вершины помечаются – 1, т.к. значение 0 и 1 могут быть расстояниями. Расстояние 0 – от исходной вершины до самой себя.

В самой процедуре сначала создается пустая очередь (п. 2.1), в которую помещается исходная вершина, из которой начат обход (п.2.2). Расстояние до этой вершины (п.2.3) устанавливается равным 0 (расстояние до самой себя).

Далее итерационно, пока очередь не опустеет, из нее извлекается первый элемент, который становится текущей вершиной (п. 2.5, 2.6). Затем в цикле просматривается **v**-я строка матрицы смежности графа G(v,i). Как только алгоритм встречает смежную с **v** не посещенную вершину (п.2.9), эта вершина помещается в очередь (п.2.11) и для нее обновляется вектор расстояния (п.2.12). Расстояние до новой **i**-й вершины вычисляется как расстояние до текущей **v**-й вершины плюс вес ребра до новой вершины G(v,i).

После просмотра строки матрицы смежности алгоритм делает следующую итерацию цикла 2.4 или заканчивает работу, если очередь пуста.

Если для всех пар вершин графа определены расстояния, то можно вычислить эксцентриситет

Если *G -* граф, содержащий непустое множество *n* вершин *V* и множество ребер *E* и *d(vi, vj) –* расстояние между двумя произвольными вершинами *vi* и*vj*, тогда для фиксированной вершины *v* величина

,

где *v*, *vj* ∈*V* и *j* = 1…*n* называется **эксцентриситетом** вершины *vi*.

Другими словами **эксцентриситет** вершины – расстояние до наиболее удаленной вершины графа.

Максимальный эксцентриситет среди эксцентриситетов всех вершин графа называется **диаметром** графа *G* и обозначается через *D(G).*

Вершина *vi* называется **периферийной***,* если её эксцентриситет равен диаметру графа *e(vi) = d(G).*

Минимальный из эксцентриситетов вершин графа называется его **радиусом** и обозначается через *r(G).*

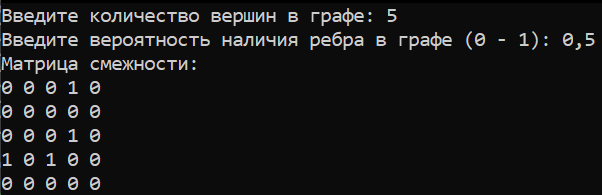
Вершина *vi* называется **центральной***,* если её эксцентриситет равен радиусу графа *e(vi) = r(G).*

Множество всех центральных вершин графа называется его **центром***.* Граф *G* может иметь единственную центральную вершину или несколько центральных вершин.

.

### **Задание 1**

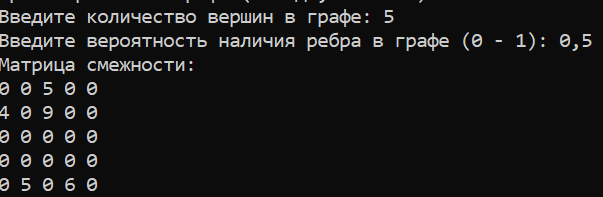
1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного взвешенного графа *G*. Выведите матрицу на экран.



1. 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.



### 

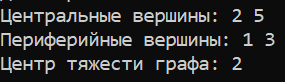
### **Задание 2**

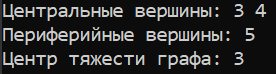
1. Для каждого из вариантов сгенерированных графов (ориентированного и не ориентированного) определите радиус и диаметр.





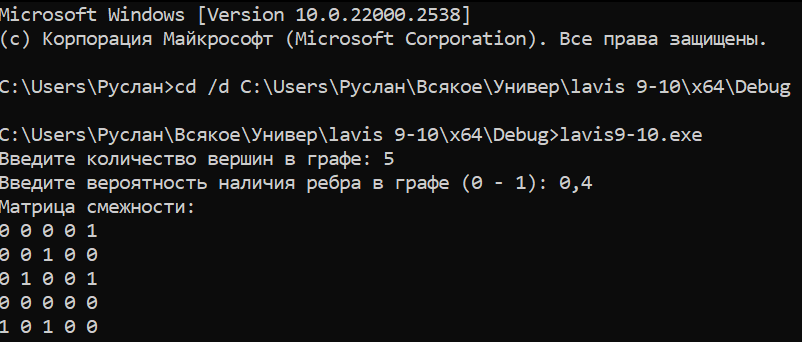
1. Определите подмножества периферийных и центральных вершин.





### **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).



Примечание: задания, помеченные символом \* выполняются по указанию преподавателя.

**Дополнительная информация.**

При создании консольного приложения в языке программирования C/С++ возможно задание параметров функции **main**(). Для этого объявляются параметры **argс** (argument count) и **argv**[] (argument values):

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

  Если запустить такую программу через командную строку, то через эти параметры можно передать ей какую-либо информацию. Параметр **argc** имеет тип данных **int**, и содержит количество параметров, передаваемых в функцию **main**. Причем **argc** всегда не меньше 1, даже если мы не передаем никакой информации в программу – в неё передаются путь и имя программы. Через командную строку в программу передаются данные строкового типа. Поэтому параметр **argv**[] – это массив указателей на строки.

Нулевой параметр **argv**[0] – это путь и имя вызываемой программы. Далее следуют параметры, указываемые в командной строке при вызове программы через пробел. Например, вызов

Program.exe P1 P2 P3

передаст в программу:

**argv**[1] = P1

**argv**[2] = P2

**argv**[3] = P3

Обработка параметров обычно начинается с проверки их количества:

#include "stdafx.h"

#include <iostream>

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

{

if(argc > 1)

{

// Do something

}

return 0;

}

Если аргументы передаются, то с ними совершаются какие-либо действия:

#include "stdafx.h"

#include <iostream>

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

{

setlocale(LC\_CTYPE, "rus");

std::cout << "В программе аргументов: " << argc << " \n\n";

if(argc > 1)

{

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

std::cout << i << "-й аргумент: " << argv[i] << '\n';

}

else

{

std::cout << "Укажите аргументы. ";

}

std::cout << '\n';

system("pause");

return 0;

}

Часто в программах используются составные параметры, где первая часть указывает на тип (название) параметра, а вторая на его значение. В этом случае для обозначения типа используется какой-либо заранее заданный символ ("-", "--", "\_" и т.д.).

Например, в программу

#include "stdafx.h"

#include <iostream>

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

{

setlocale(LC\_CTYPE, "rus");

std::cout << "В программе аргументов: " << argc << " \n\n";

if(argc > 1)

{

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

{

if (strcmp(argv[i], "-name1") == 0)

std::cout << "Аргумент " << argv[i] << " имеет значение: " << argv[i + 1] << '\n';

if (strcmp(argv[i], "-name2") == 0)

std::cout << "Аргумент " << argv[i] << " имеет значение: " << argv[i + 1] << '\n';

}

}

else

{

std::cout << "Укажите аргументы. ";

}

std::cout << '\n';

system("pause");

return 0;

}

можно передавать параметры-name1 и -name2 в любом порядке:

Pogram\_with\_parameters.exe -name1 23 -name2 45

или

Pogram\_with\_parameters.exe -name2 45 -name1 23

на результат работы программы это не повлияет.

Вывод:

Научились проводить расчет расстояния во взвешенном и/или ориентированном графе. Научили искать радиус графа, диаметр, центр и центр тяжести графа.

Приложение А “Листинг”

// Подключение необходимых библиотек

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cstdio>

#include <vector>

#include <queue>

#include <ctime>

#include <cstdlib>

#include <algorithm>

#include <climits>

#include <locale.h>

using namespace std;

// Функция для обхода в ширину с записью расстояний

void BFSD(const vector < vector < int>>& G, int start\_vertex, int target\_vertex, vector<int>& DIST) {

queue<int> Q;

Q.push(start\_vertex);

DIST[start\_vertex] = 0;

while (!Q.empty()) {

int current = Q.front();

Q.pop();

if (current == target\_vertex) // Если достигли целевой вершины, завершаем поиск

break;

for (int i = 1; i < G[current].size(); ++i) { // Исправлено условие на <, чтобы не выходить за пределы вектора

if (G[current][i] > 0 && DIST[i] == -1) {

Q.push(i);

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

}

}

}

}

// Перегруженная функция для обхода в ширину без указания целевой вершины

void BFSD(const vector < vector < int>>& G, int start\_vertex, vector<int>& DIST) {

queue<int> Q;

Q.push(start\_vertex);

DIST[start\_vertex] = 0;

while (!Q.empty()) {

int current = Q.front();

Q.pop();

for (int i = 1; i < G[current].size(); ++i) {

if (G[current][i] > 0 && DIST[i] == -1) {

Q.push(i);

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

}

}

}

}

int findGraphCentroid(const vector < vector < int>>& G, int num\_vertices) {

// Вычисление эксцентриситета для каждой вершины

vector<int> eccentricity(num\_vertices + 1);

for (int v = 1; v <= num\_vertices; ++v) {

vector<int> DIST(num\_vertices + 1, -1);

BFSD(G, v, DIST);

int max\_dist = 0;

for (int dist : DIST) {

if (dist != -1) {

max\_dist = max(max\_dist, dist);

}

}

eccentricity[v] = max\_dist;

}

// Нахождение вершины с минимальным эксцентриситетом

int min\_eccentricity = INT\_MAX;

int centroid = -1;

for (int v = 1; v <= num\_vertices; ++v) {

if (eccentricity[v] < min\_eccentricity) {

min\_eccentricity = eccentricity[v];

centroid = v;

}

}

return centroid;

}

// Функция для генерации и печати матрицы смежности

vector < vector < int>> generateAndPrintMatrix(int num\_vertices, double edge\_probability, bool isDirected, bool isWeighted) {

vector < vector < int>> adjacency\_matrix(num\_vertices + 1, vector<int>(num\_vertices + 1, 0));

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = (isDirected ? 1 : i + 1); j <= num\_vertices; ++j) {

if (i != j && static\_cast<double>(rand()) / RAND\_MAX < edge\_probability) {

if (isWeighted) {

adjacency\_matrix[i][j] = rand() % 10 + 1;

}

else {

adjacency\_matrix[i][j] = 1;

}

if (!isDirected) {

adjacency\_matrix[j][i] = adjacency\_matrix[i][j];

}

}

}

}

// Вывод матрицы смежности на экран

printf("Матрица смежности:\n");

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

printf("%d ", adjacency\_matrix[i][j]);

}

printf("\n");

}

return adjacency\_matrix;

}

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

// Установка локали

setlocale(LC\_ALL, "RUS");

// Инициализация флагов для взвешенного и ориентированного графа

int isWeighted = 0;

int isDirected = 0;

if (argc > 1)

{

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

{

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

isWeighted = 1;

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

isDirected = 1;

}

}

// Проверка наличия аргументов командной строки

if (argc != 3) {

printf("Вы не предоставили аргументы командной строки.\n");

printf("Взвешенный граф? (1 = да, 0 = нет): ");

scanf("%d", &isWeighted);

printf("Ориентированный граф? (1 = да, 0 = нет): ");

scanf("%d", &isDirected);

}

else {

isWeighted = atoi(argv[1]);

isDirected = atoi(argv[2]);

}

// Проверка введенных значений

if ((isWeighted != 0 && isWeighted != 1) || (isDirected != 0 && isDirected != 1)) {

printf("Неверные аргументы. Пожалуйста, следуйте инструкциям.\n");

return 1;

}

// Ввод количества вершин и вероятности ребер

int num\_vertices;

double edge\_probability;

do {

printf("Введите количество вершин в графе: ");

scanf("%d", &num\_vertices);

} while (num\_vertices <= 0);

do {

printf("Введите вероятность наличия ребра в графе (0 - 1): ");

scanf("%lf", &edge\_probability);

} while (edge\_probability < 0.0 || edge\_probability > 1.0);

// Инициализация генератора случайных чисел

srand(static\_cast<unsigned int>(time(nullptr)));

// Генерация и вывод матрицы смежности

auto matrix = generateAndPrintMatrix(num\_vertices, edge\_probability, isDirected, isWeighted);

// Ввод начальной и конечной вершин для поиска пути

int start\_vertex = 1;

int target\_vertex;

do {

printf("Введите конечную вершину: ");

scanf("%d", &target\_vertex);

} while (target\_vertex <= 0 || target\_vertex > num\_vertices);

// Инициализация массива расстояний

vector<int> DIST(num\_vertices + 1, -1);

// Вызов функции поиска кратчайшего пути

BFSD(matrix, start\_vertex, target\_vertex, DIST);

if (DIST[target\_vertex] == -1) {

printf("Предупреждение: Путь до заданной вершины не существует.\n");

}

else {

printf("Расстояние до заданной вершины: %d\n", DIST[target\_vertex]);

}

// Вычисление радиуса и диаметра графа

vector<int> eccentricity(num\_vertices + 1, 0);

int radius = INT\_MAX, diameter = INT\_MIN;

for (int v = 1; v <= num\_vertices; ++v) {

vector<int> DIST(num\_vertices + 1, -1);

BFSD(matrix, v, DIST);

int max\_dist = 0;

for (int dist : DIST) {

if (dist != -1) {

max\_dist = max(max\_dist, dist);

}

}

eccentricity[v] = max\_dist;

radius = min(radius, max\_dist);

diameter = max(diameter, max\_dist);

}

// Вывод радиуса и диаметра графа, центральных и периферийных вершин

printf("Радиус: %d\n", radius);

printf("Диаметр: %d\n", diameter);

printf("Центральные вершины: ");

for (int i = 1; i <= num\_vertices; ++i) {

if (eccentricity[i] == radius) {

printf("%d ", i);

}

}

printf("\n");

printf("Периферийные вершины: ");

for (int i = 1; i <= num\_vertices; ++i) {

if (eccentricity[i] == diameter) {

printf("%d ", i);

}

}

printf("\n");

int centroid = findGraphCentroid(matrix, num\_vertices);

printf("Центр тяжести графа: %d\n", centroid);

return 0;

}