**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ.УЛЬЯНОВА (ЛЕНИНА)**

Кафедра вычислительной техники

Отчет по курсовой работе на тему:

«Графы»

по дисциплине «Алгоритмы и структуры данных»

Колинько П.Г.

Санкт-Петербург

2018

Торопов В. А.

Преподаватель

Студент гр. 7307

Оглавление

[Задание 3](#_Toc530320008)

[Математическая формулировка задачи. 4](#_Toc530320009)

[Выбор и обоснование способа представления данных. 5](#_Toc530320010)

[Описание алгоритма и оценка его временной сложности. 6](#_Toc530320011)

[Набор тестов и результаты проверки алгоритма на ЭВМ. 7](#_Toc530320012)

[Вывод 8](#_Toc530320013)

[Список литературы 9](#_Toc530320014)

[Код программы 10](#_Toc530320015)

# **Задание**

Поиск кратчайшего пути между заданной парой вершин в неориентированном графе с нагруженными неотрицательными рёбрами. (При отрицательных рёбрах возникнут отрицательные циклы, в следствии чего кратчайшего пути не будет существовать)

# **Уточнение задания**

Вес ребра является целым числом для того, чтобы визуальное представление графа было красивым и читабельным при выводе его на экран.

# Математическая формулировка задачи.

G(V, E) – заданный граф с множеством вершин V и множеством рёбер E. Граф неориентированный с нагруженными неотрицательными рёбрами.

i, j V– данные вершины, между которыми необходимо найти кратчайший путь.

u[k,l] E – вес ребра между вершинами k, l.

S[i,j] – путь между вершинами i, j.

Путь – это последовательность попарно смежных вершин.

Найти: min S[i,j].

Минимальный путь между вершинами – это последовательность попарно смежных вершин с минимальной суммой весов рёбер между этими вершинами.

# Выбор и обоснование способа представления данных.

Для решения данной задачи используется весовая матрица, так как граф является неориентированным и с нагруженными рёбрами. Следовательно она оптимально подходит для описания данного графа.

# Описание алгоритма и оценка его временной сложности.

Для решения данной задачи используется алгоритм Дейкстры: алгоритм находит расстояние от одной из вершин графа до всех остальных, а значит и подходит для поиска расстояния от одной вершины до другой конкретной вершины. К дополнению к алгоритму Дейкстры используется массив предков: массив, хранящий номера вершин, путь из которых в данную ячейку, является кратчайшим.

Для хранения чисел используется массив чисел, а для хранения принадлежности элемента к множеству – массив булевых переменных.

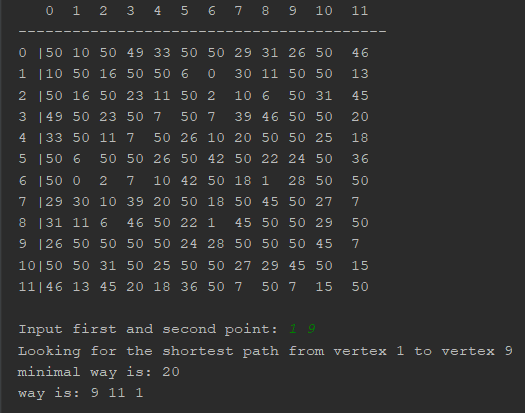
В начале алгоритма расстояние для начальной вершины полагается равным нулю, а все остальные расстояния заполняется максимально возможным. Массив флагов заполняется false. Затем запускается основной цикл.

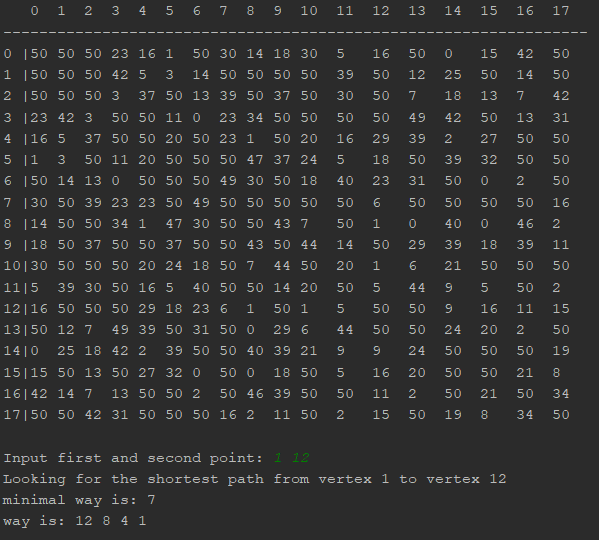
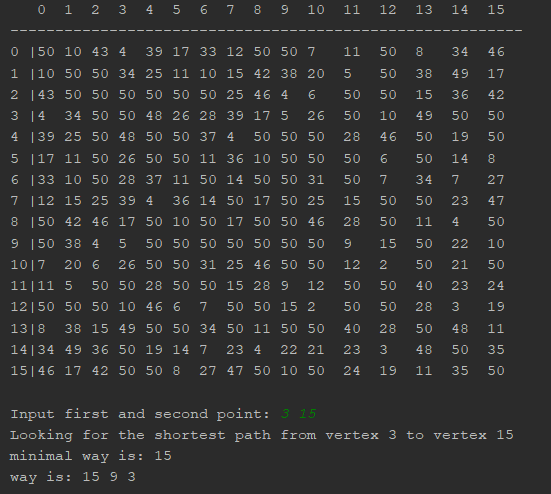
На каждом шаге цикла мы ищем вершину v с минимальным расстоянием и флагом равным нулю. Затем мы устанавливаем в соответствующей ячейке массива флагов значение true и проверяем все соседние вершины u. Если в них расстояние больше, чем сумма расстояния до текущей вершины и длины ребра, то уменьшаем его, а также переписываем соответствующую ячейку в массиве предков. Цикл завершается, когда все вершины пройдены.

Путь получается через цикл по массиву предков: цикл начинается с ячейки, путь до которой нам следовало найти. Выписываем номер этой ячейки, а параметр цикла принимает значение, хранящееся в этой ячейке. Цикл заканчивается в тот момент, когда значение ячейки будет равно первой ячейке, путь из которой следует найти.

Временная сложность всего алгоритма: O(n^2).

# Набор тестов и результаты проверки алгоритма на ЭВМ.





# Вывод

В данной работе были изучены графы, представления их в памяти ЭВМ, а также алгоритмы работы с графами.

# Список литературы

1. Колинько Павел Георгиевич – Алгоритмы и структуры данных Ч.1, Методические указания к самостоятельной работе на ПЭВМ и курсовому проектированию, выпуск 1809.
2. Б/А – Алгоритм Дейкстры, Википедия – свободная энциклопедия

URL: https//ru.wikipedia.org/wiki/Алгоритм\_Дейкстры.

# **Код программы**

Main:

#include "graph.h"  
  
**int** main() {  
 Graph A;  
 **if**(A.generator()){  
 A.show();  
 A.way();  
 **return** 0;  
 } **else return** 1;  
}

graph.h:

#ifndef INC\_4\_GRAPH\_H  
#define INC\_4\_GRAPH\_H  
#include <iostream>  
#include <time.h>  
#include <fstream>  
#endif //INC\_4\_GRAPH\_H  
  
**using namespace** std;  
  
**class** Graph  
{  
 **static int** maxN;  
 **static int** maxS;  
 **int** N;  
 **int** \*\*gr;  
**public**:  
 **bool** generator();  
 **void** show();  
 **int** way();  
 Graph(): gr(**nullptr**) {};  
 ~Graph(){  
 **for**(**int** i = 0; i < N; i++){  
 **delete**(gr[i]);  
 }  
 **delete**(gr);  
 }  
};  
  
**int** Graph:: maxN = 20;  
**int** Graph:: maxS = 50;  
  
  
**bool** Graph:: generator(){  
 cout << "1: Generate from filee\n2: Random generation\nYour choose is: ";  
 **int** choose;  
 cin >> choose;  
 **switch**(choose){  
 **case** 1: {  
 ifstream input;  
 input.open("generator.txt", ios::in);  
 **if**(input) {  
 input >> N;  
 gr = **new int** \*[N];  
 **for** (**int** i = 0; i < N; i++) {  
 gr[i] = **new int**[N];  
 **for** (**int** i1 = 0; i1 < N; i1++) {  
 input >> gr[i][i1];  
 }  
 }  
 input.close();  
 } **else** {  
 cout << "wrong file" << endl;  
 **return false**;  
 }  
 **break**;  
 }  
 **case** 2:{  
 srand(time(**nullptr**));  
 N = rand() % maxN;  
 **while**(N < 10){  
 N = rand() % maxN;  
 }  
 gr = **new int** \* [N];  
 **for**(**int** i = 0; i < N; i++){  
 gr[i] = **new int** [N];  
 }  
 **for**(**int** i = 0; i < N; i++){  
 **for**(**int** i1 = 0; i1 <= i; i1++){  
 **int** a = rand() % 100;  
 **if**(i == i1) gr[i][i1] = maxS; **else  
 if**(a >= 30) {  
 gr[i][i1] = rand() % maxS;  
 } **else** gr[i][i1] = maxS;  
 }  
 }  
 **for**(**int** i = 0; i < N - 1; i++){  
 **for**(**int** i1 = i + 1; i1 < N; i1++){  
 gr[i][i1] = gr[i1][i];  
 }  
 }  
 **break**;  
 }  
 **default**: {  
 cout << "wrong input" << endl;  
 **return false**;  
 }  
 }  
 **return true**;  
}  
  
**void** Graph::show(){  
 cout << " ";  
 **for**(**int** i = 0; i < N; i++) cout << i << " ";  
 cout << endl;  
 cout << "---";  
 **for** (**int** i = 0; i < N; i++) **if**(i < 10) cout << "---"; **else** cout << "----";  
 cout << endl;  
 **for**(**int** i = 0; i < N; i++){  
 **if** (i / 10 == 0) cout << i << " |"; **else** cout << i << "|";  
 **for**(**int** i1 = 0; i1 < N; i1++){  
 **if** (gr[i][i1] / 10 == 0) cout << gr[i][i1] << " ";  
 **else** cout << gr[i][i1] << " ";  
 **if**(i1 / 10 != 0) cout << " ";  
 }  
 cout << endl;  
 }  
 cout << endl;  
}  
  
**int** Graph::way(){  
 **int** first, second;  
 cout << "Input first and second point: ";  
 cin >> first >> second;  
 cout << "Looking for the shortest path from vertex " << first << " to vertex " << second << endl;  
 **int** point[N];  
 **bool** pass[N];  
 **for**(**int** i = 0; i < N; i++){  
 point[i] = maxS;  
 pass[i] = **false**;  
 }  
 **int** k = 0;  
 **int** tp;  
 **int** p[N];  
 tp = first;  
 point[first] = 0;  
 p[first] = first;  
 **while** (k < N){  
 **int** min = maxS;  
 **for**(**int** i = 0; i < N; i++){  
 **if**(!pass[i] && point[i] <= min){  
 min = point[i];  
 tp = i;  
 }  
 }  
 **for**(**int** i = 0; i < N; i++){  
 **if**(point[i] > point[tp] + gr[tp][i]){  
 point[i] = point[tp] + gr[tp][i];  
 p[i] = tp;  
 }  
 }  
 pass[tp] = **true**;  
 k++;  
 }  
  
 cout << "minimal way is: " << point[second] << endl;  
 cout << "way is: ";  
 k = second;  
 cout << second << " ";  
 **while**(k != first){  
 cout << p[k] << " ";  
 k = p[k];  
 }  
 cout << endl;  
  
 **return** N;  
}