**Индивидуальное задание**

**«Теория графов»**

***«Дискретная математика»***

|  |
| --- |
| **Выполнил:** |
| Студент 1-го курса механико-математического факультета  \_\_ Семенов Сергей,  группа ПМИ-1,2. |

**Пермь 2021**

**Постановка задачи**

*Нахождение минимального расстояния между двумя заданными вершинами (алгоритм*

*Дейкстры) с указанием пути.*

Входные данные. В первой строке записано одно число n – количество вершин в

графе. Далее располагается матрица расстояний графа (n строк по n чисел в каждой).

Отсутствие ребра в матрице отмечается символом тильды «~». Граф связный. Длины

ребер – натуральные числа. В последней строке файла располагаются два числа –

номера вершин, кратчайший путь между которыми требуется найти.

Выходные данные: в первой строке – минимальное расстояние между заданными

вершинами, во второй строке – сам кратчайший путь (в виде последовательности

номеров вершин).

# Определение идеи алгоритма, выбор структур данных

Для решения задачи была выбрана система программирования Visual Studio 2019 и язык программирования C++.

Основные моменты:

* Ввод данных
* Работа алгоритма Дейкстры (Заполнение массива минимальных расстояний до каждой вершины + массива с “Предыдущими” вершинами)
* Вывод

**Основные переменные и функции:**

* InputSize - функция отвечает за инициализацию переменной n (кол-во вершин).
* InputData - функция отвечает за ввод исходных данных.
* WorkDijkstra – функция отвечает за реализацию работы алгоритма Дейкстры.
* TakeNextW – функция отвечает за выбор следующей рассматриваемой вершины.
* Way – функция отвечает за вывод вершин, через которые проходит прокладывается минимальный путь.
* int a – исходная вершина.
* int b – конечная вершина.
* int w – текущая рассматриваемая вершина.
* vector<vector<int>> mas(n, vector<int>(n)); - матрица смежности с расстояниями.
* vector <int> distance(n); - массив с минимальными расстояниями.
* vector <int> p(n); - массив для последующего восстановления пути.
* vector <bool> checked(n); - массив для проверки просмотра вершины.
* В качестве бесконечности была использована переменная const int INF = 9999;

**Рассмотрим принцип работы алгоритма на примере из задания.**

input.txt output.txt

4 3

0 3 8 1 1 4 3

3 0 4 ~

8 4 0 2

1 ~ 2 0

1 3

Первым шагом считываем все необходимые данные из файла input.txt. Т.к. индексы в массиве нумеруются с нуля, переменные a и b запоминаем на единицу меньше. distance[a] присваиваем значение 0, т.к. путь a => a весит 0 у.е.

До начала работы функции WorkDijkstra “таблица” выглядит следующим образом (в столбец checked будем заносить вершины, значение которых, в массиве checked = true. А так же опустим индексацию с нуля и будем рассматривать, исключительно для удобства, пример с обычными вершинами, нумерация которых начинается с 1 и заканчивается n).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **checked** | **w** | **D[1]** | **D[2]** | **D[3]** | **D[4]** |
| { } |  | 0 | ∞ | ∞ | ∞ |

После запуска в алгоритме сработает цикл n раз. Так же в функции есть вложенный цикл, который будет проверять, является ли путь через рассматриваемую вершину(w) к текущей(i) короче предыдущего и, соответственно, изменять значения массивов distance и p(в этот массив по текущему индексу(i) мы будем заносить значение рассматриваемой вершины(w) ), в случае положительного результата. Само условие выглядит следующим образом: if (checked[i] == false && mas[w][i] + distance[w] < distance[i]).

После одного “круга” наша таблица принимает следующий вид:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **checked** | **w** | **D[1]** | **D[2]** | **D[3]** | **D[4]** |
| { } |  | 0 | ∞ | ∞ | ∞ |
| {1} | 1 | 0 | 3 | 8 | 1 |

А массив p:

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **3** | **4** |
| -1 | 1 | 1 | 1 |

Далее при помощи функции TakeNextW(проходим по массиву distance, рассматриваем вершины, значение в массиве checked которых равно false, находим минимальное, и возвращаем индекс этой вершины) определяем вершину, путь до которой на данный момент является минимальным, присваиваем это значение переменной w и повторяем цикл.

По окончании работы функции WorkDijkstra таблица выглядит следующим образом:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **checked** | **w** | **D[1]** | **D[2]** | **D[3]** | **D[4]** |
| { } |  | 0 | ∞ | ∞ | ∞ |
| {1} | 1 | 0 | 3 | 8 | 1 |
| {1,4} | 4 | 0 | 3 | 3 | 1 |
| {1,4,2} | 2 | 0 | 3 | 3 | 1 |
| {1,4,2,3} | 3 | 0 | 3 | 3 | 1 |

А массив P:

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **3** | **4** |
| -1 | 1 | 4 | 1 |

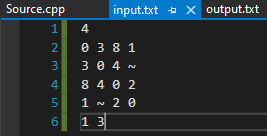
Далее выводим минимальное расстояние до нужной нам вершины distance[b].

Восстанавливаем путь с помощью функции Way(создаем deque<int> res и “Закидываем” значение переменной b + 1, т.е. номер последней вершины. Заводим переменную int temp = b. После присваиваем переменной temp значение p[temp] и кладем в начало очереди значение переменной temp + 1. Так делаем до тех пор, пока p[temp] не будет равно -1, то есть пока не дойдем до исходной вершины. Выводим все элементы очереди с начала).

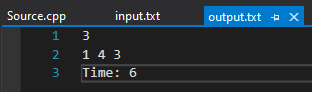
**Тестирование программы**

Тест№ 1

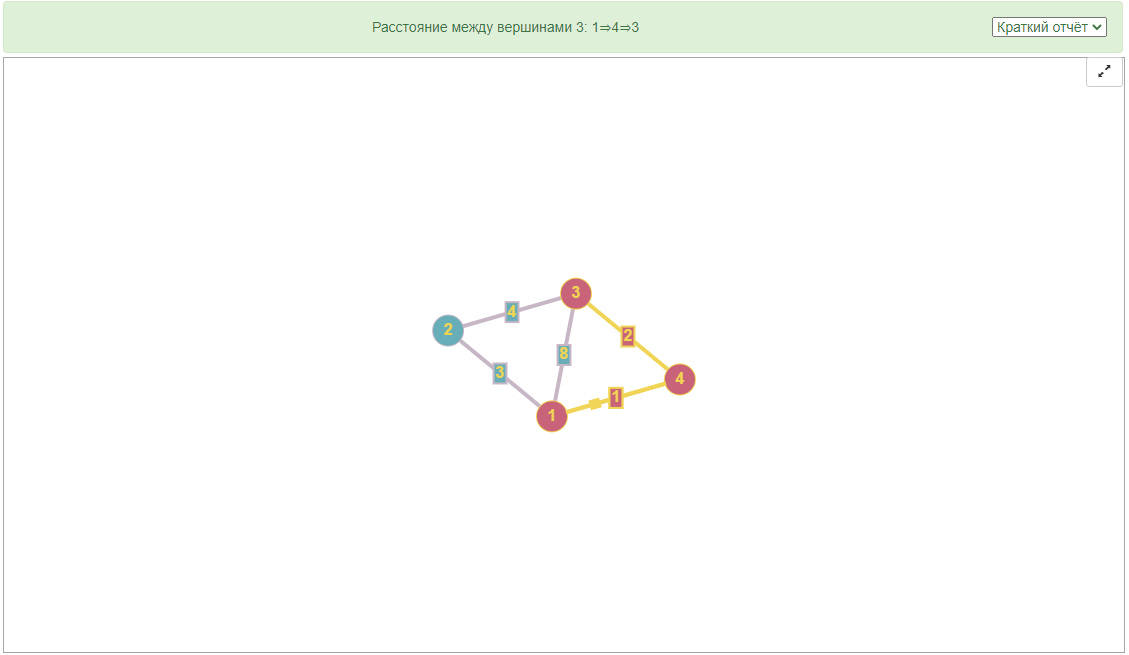
Входные данные:



Результат работы программы:

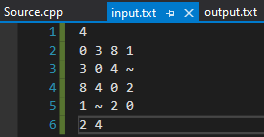


Сверяем с graphonline.ru:

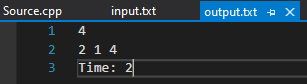


Тест№ 2

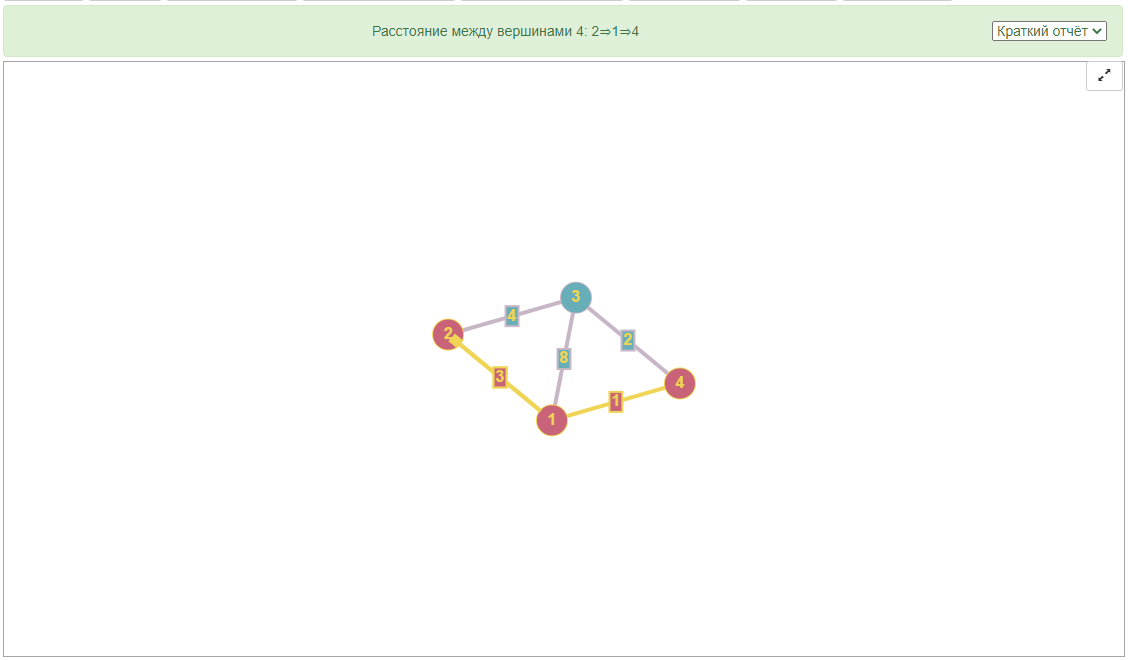
Входные данные:



Результат работы программы:

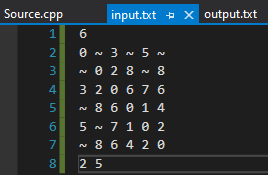


Сверяем с graphonline.ru:

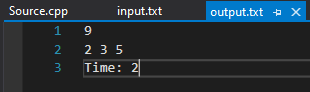


Тест№ 3

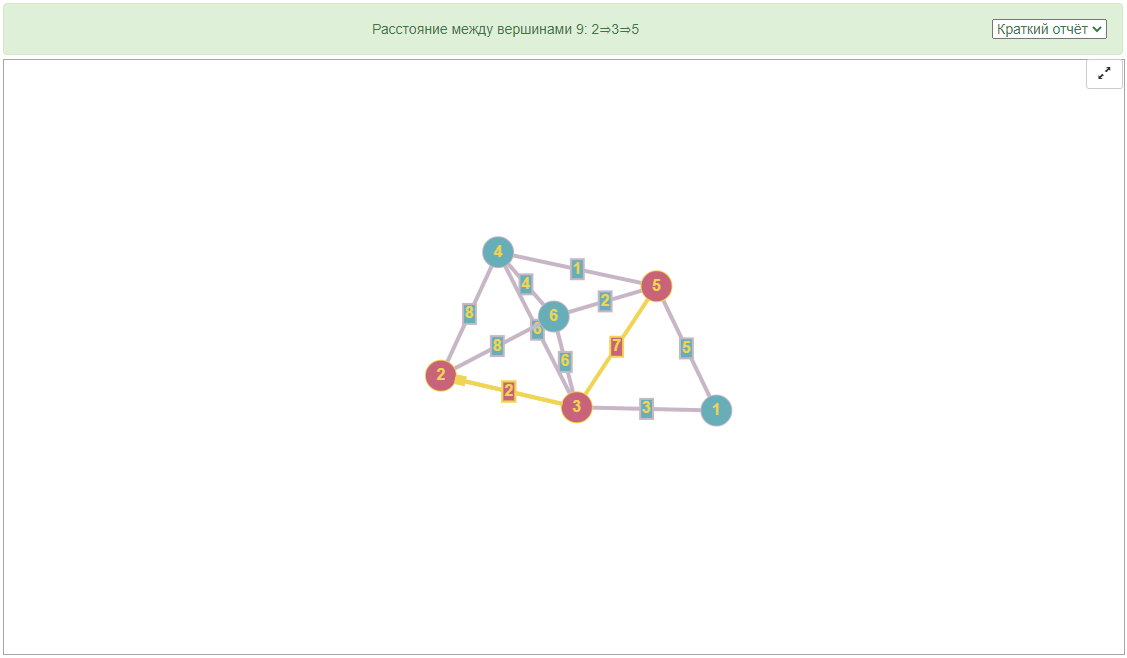
Входные данные:



Результат работы программы:

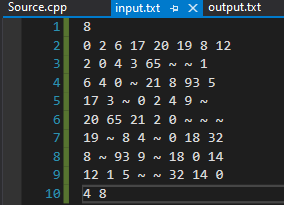


Сверяем с graphonline.ru:

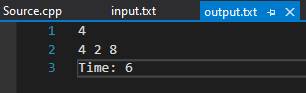


Тест№ 4

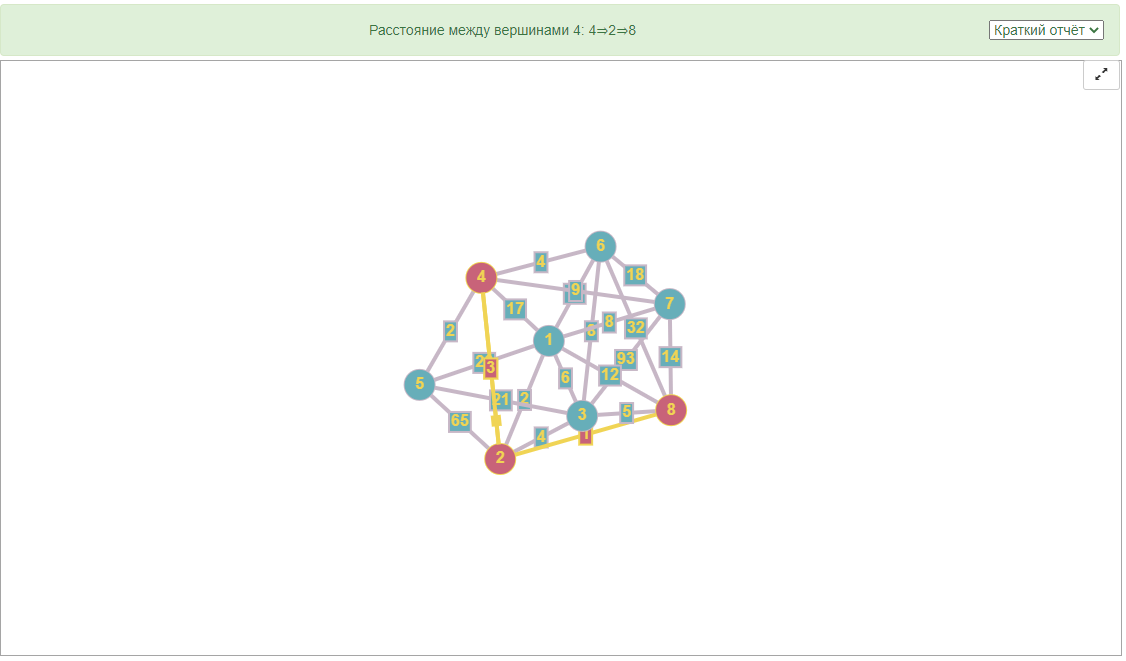
Входные данные:



Результат работы программы:

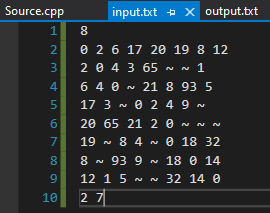


Сверяем с graphonline.ru:

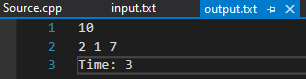


Тест№ 5

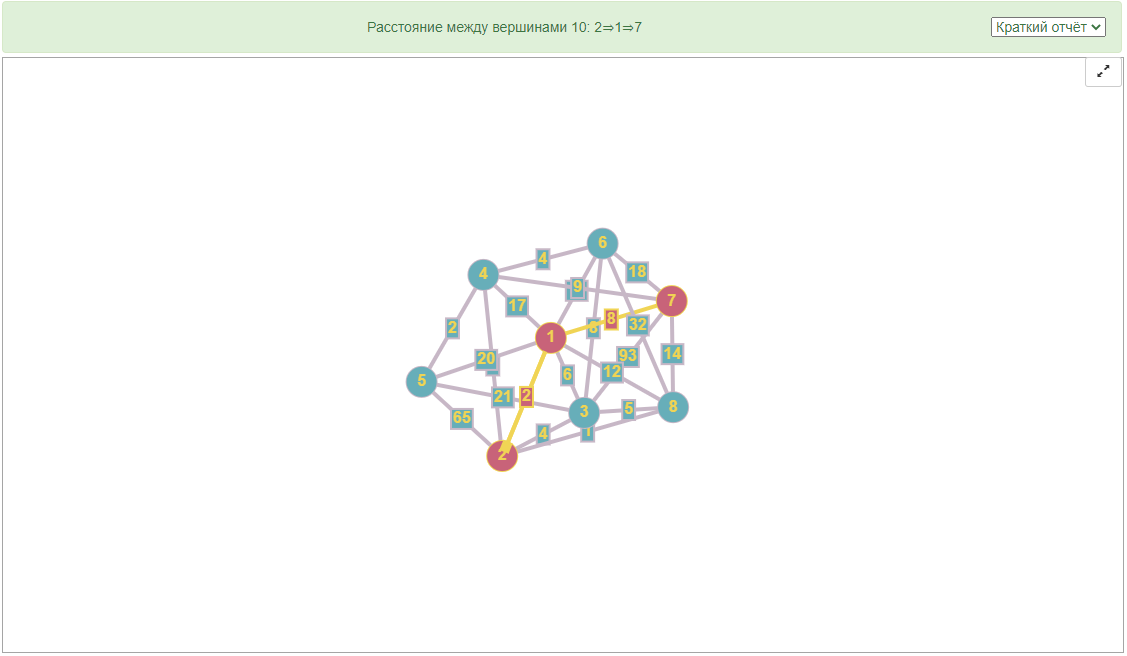
Входные данные:



Результат работы программы:

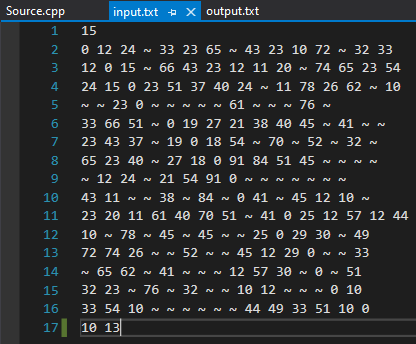


Сверяем с graphonline.ru:

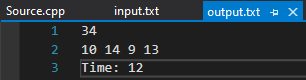


Тест№ 6

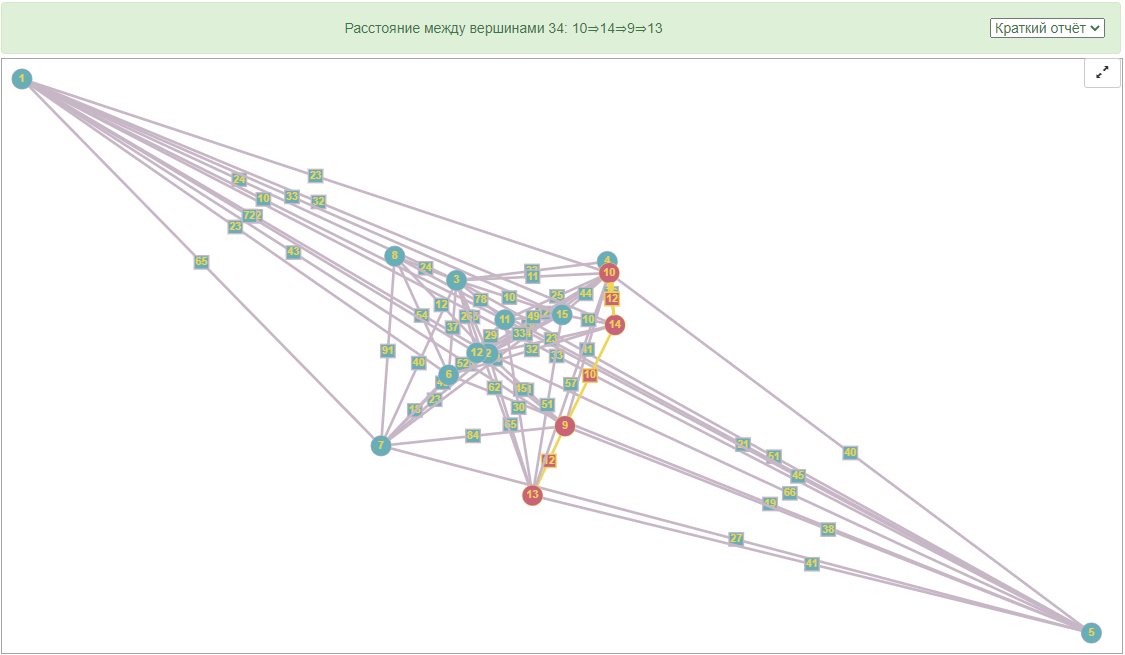
Входные данные:



Результат работы программы:

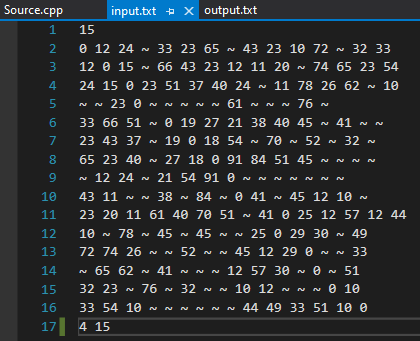


Сверяем с graphonline.ru:

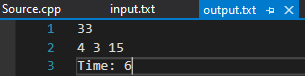


Тест№ 7

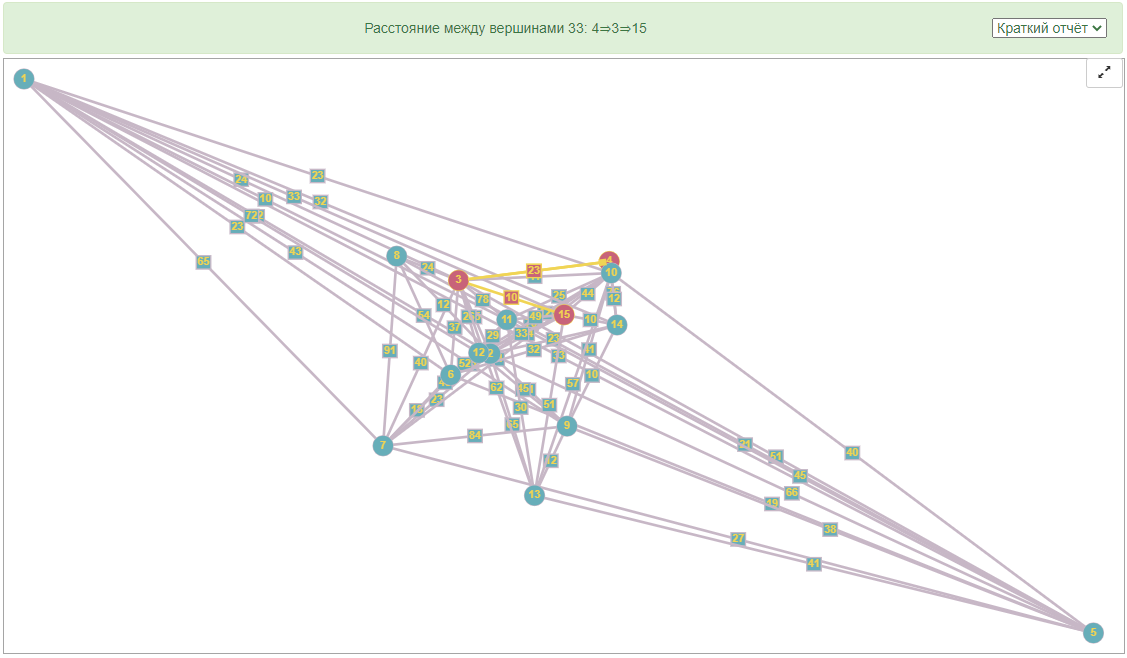
Входные данные:



Результат работы программы:

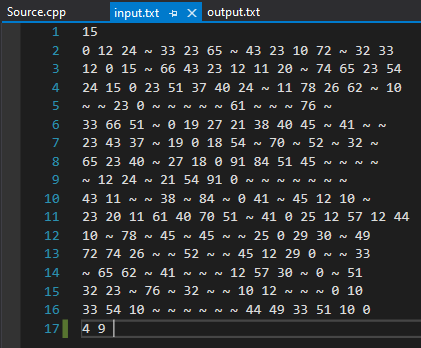


Сверяем с graphonline.ru:

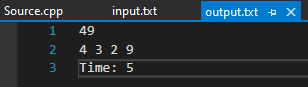


Тест№ 8

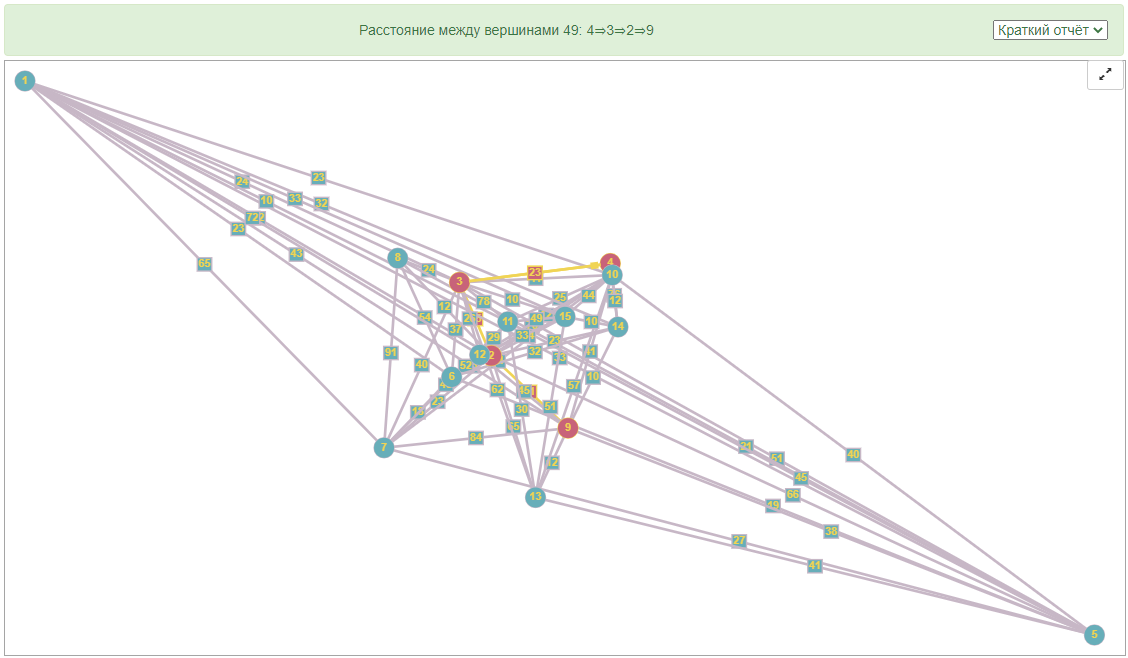
Входные данные:



Результат работы программы:



Сверяем с graphonline.ru:



**Исходный код**

#include <iostream>

#include <string>

#include <fstream>

#include <vector>

#include <deque>

using namespace std;

const int INF = 9999; // "бесконечность"

void InputSize(ifstream& input, int& n) { // кол-во вершин

string x;

input >> x;

n = stoi(x);

}

void InputData(int& n, vector<vector<int>>& mas, vector<int>& distance, vector<int>& p, int& a, int& b, ifstream& input) {

string x;

// матрица смежности с расстояниями

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

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

input >> x;

if (x != "~") mas[i][j] = stoi(x);

else mas[i][j] = INF;

}

input >> x;

a = stoi(x) - 1; // индекс исходной вершины

input >> x;

b = stoi(x) - 1; // индекс конечной вершины

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

distance[i] = INF; // массив с расстояниями

p[i] = -1; // для хранения пути

}

distance[a] = 0;

}

int TakeNextW(vector<int>& distance, int n, vector<bool>& checked) {

int minW = 0, min = INF;

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

if (distance[i] < min && checked[i] == false) {

minW = i;

min = distance[i];

}

return minW;

}

void WorkDijkstra(vector<vector<int>>& mas, vector<int>& distance, vector<int>& p, vector<bool>& checked, int n, int a) {

int w = a; // текущая рассматриваемая вершина

checked[w] = true;

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

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

if (checked[i] == false && mas[w][i] + distance[w] < distance[i]) {

distance[i] = mas[w][i] + distance[w];

p[i] = w;

}

w = TakeNextW(distance, n, checked);

checked[w] = true;

}

}

void Way(int b, vector<int>& p, ofstream& output) {

deque<int> res;

res.push\_back(b + 1);

int temp = b;

while (p[temp] != -1) {

temp = p[temp];

res.push\_front(temp + 1);

}

for (int elem : res) output << elem << ' ';

}

int main() {

int start = clock();

int n, a, b;

ifstream input("input.txt");

ofstream output("output.txt");

string x;

InputSize(input, n);

vector<vector<int>> mas(n, vector<int>(n));

vector <int> distance(n);

vector <int> p(n);

vector <bool> checked(n);

InputData(n, mas, distance, p, a, b, input);

WorkDijkstra(mas, distance, p, checked, n, a);

output << distance[b] << endl;

Way(b, p, output);

int end = clock();

output << endl << "Time: " << (end - start); // время работы в секундах

input.close();

output.close();

}