Пензенский государственный университет  
Кафедра «Вычислительной техники»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**к курсовому проектированию  
по дисциплине: «Логика и основы алгоритмизации в инженерных задачах»  
на тему: «Реализация алгоритма Дейкстры»

**Выполнил студент группы 19ВВ1:**

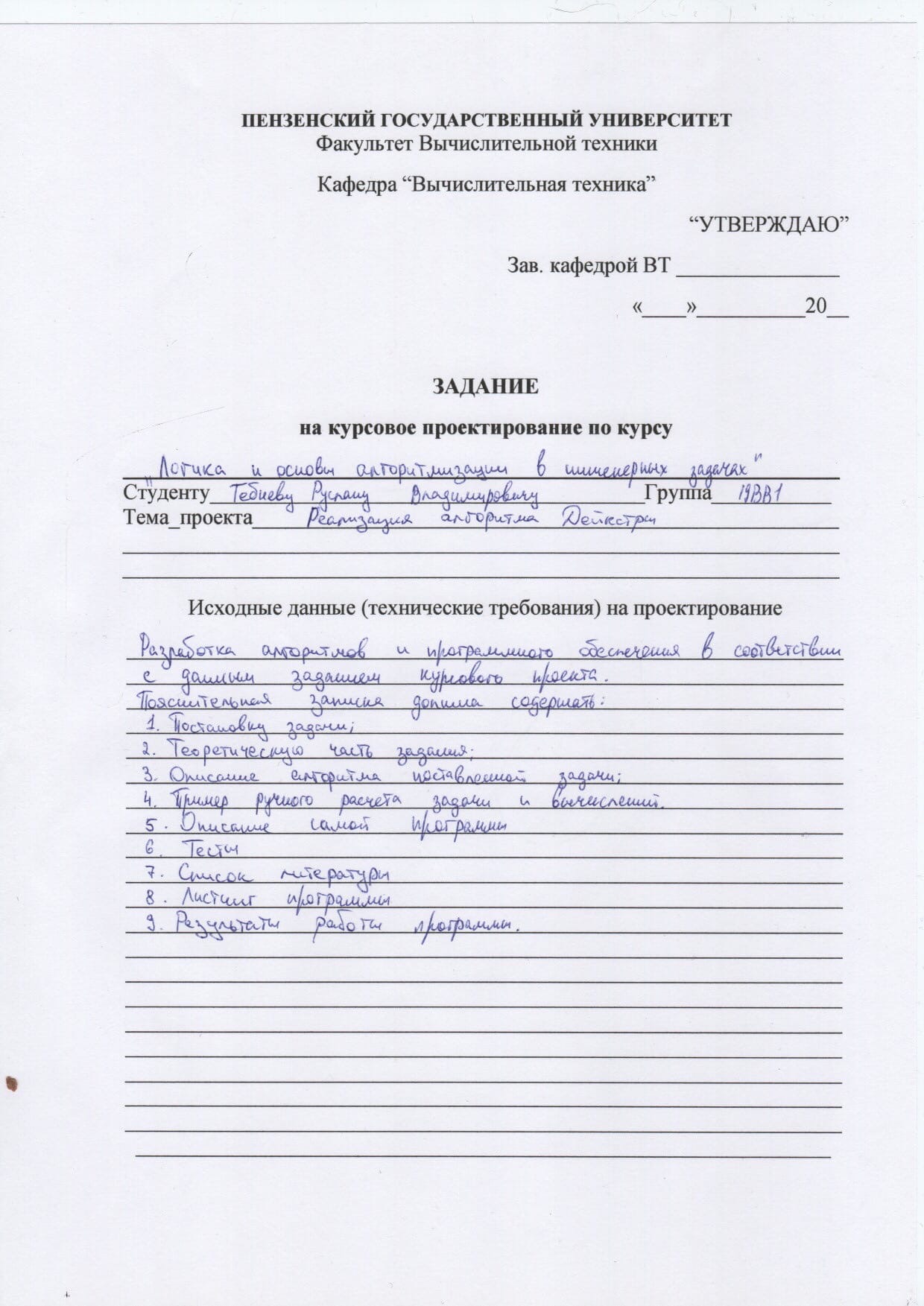
Тебнев Р.

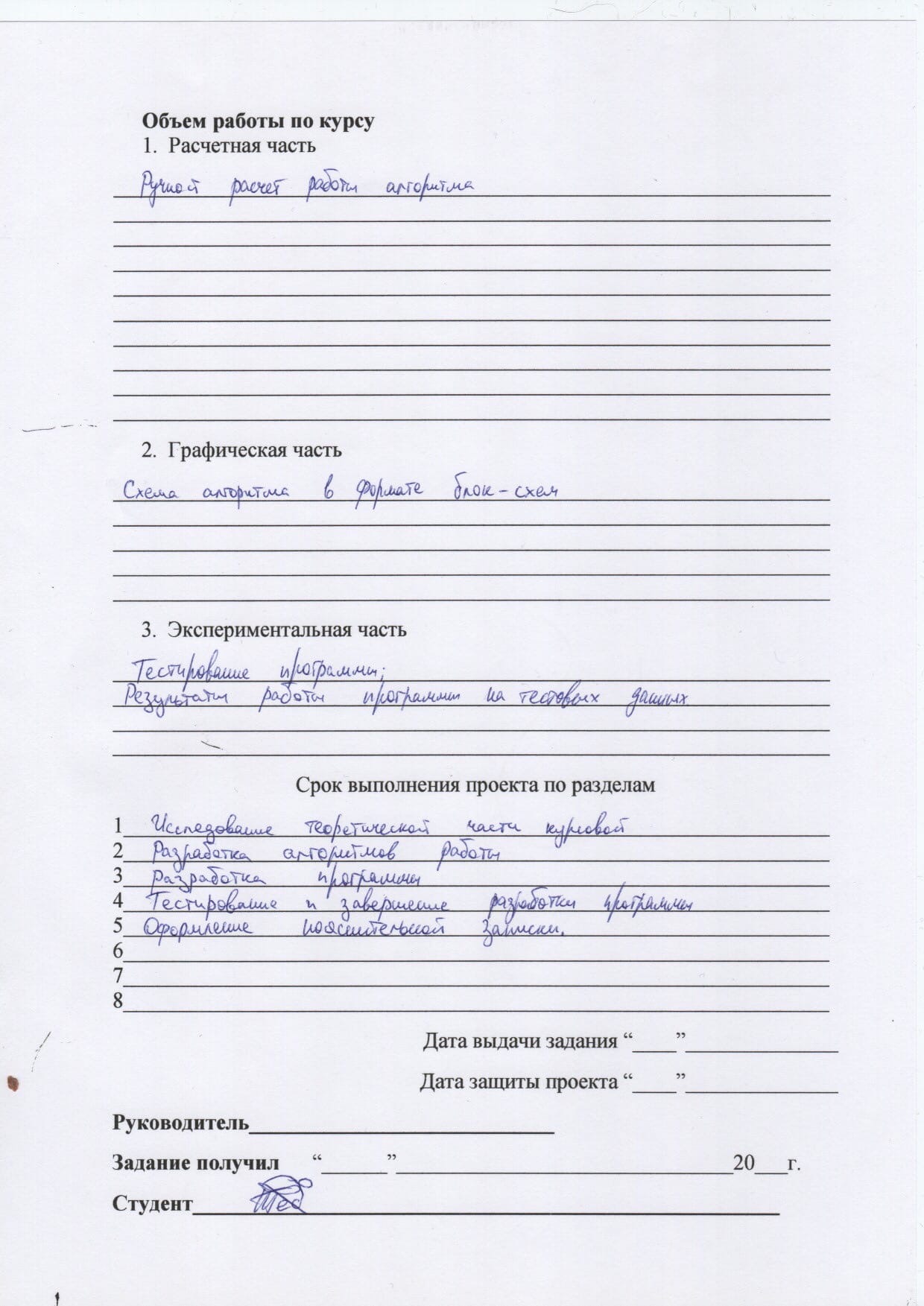
**Приняли:**

д.т.н. профессор Митрохин М. А.

Юрова О.В.

Пенза 2020





**Содержание**

Оглавление

[**Реферат** 5](#_Toc59488096)

[**Введение** 6](#_Toc59488097)

[**Постановка задачи** 7](#_Toc59488098)

[**Теоретическая часть программы** 8](#_Toc59488099)

[**Описание алгоритма программы** 9](#_Toc59488100)

[**Описание программы** 11](#_Toc59488101)

[**Тестирование** 15](#_Toc59488102)

[**Ручной расчет задачи** 19](#_Toc59488103)

[**Заключение** 20](#_Toc59488104)

[**Список литературы** 21](#_Toc59488105)

[**Листинг** 22](#_Toc59488106)

# **Реферат**

Отчет 23 стр., 17 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ДЕЙКСТРЫ.

Цель исследования – разработка программы, способная находить кратчайшие расстояния до всех вершин графа с помощью алгоритма Дейкстры.

В работе рассмотрен алгоритм, на основе которого находят кратчайшее расстояние от одной вершины графа до всех остальных. Работает только для графов без рёбер отрицательного веса.

# **Введение**

**Алгори́тм Де́йкстры** (англ. *Dijkstra’s algorithm*) — алгоритм на графах, изобретённый нидерландским учёным Эдсгером Дейкстрой в 1959 году. Находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании и технологиях, например, его используют протоколы маршрутизации OSPF и IS-IS. Алгоритм Дейкстры относится к так называемым «жадным» алгоритмам.

Программа должна быть разработана для работы в операционной системе Microsoft Windows.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2019, язык программирования – Си.

В процессе выполнения работы были усовершенствованы навыки работы с данным языком программирования.

# **Постановка задачи**

Требуется разработать программу, которая будет выполнять поиск минимального пути от одной выбираемой вершины к другим, с помощью алгоритма Дейкстры.

Пользователь должен иметь возможность выбирать размер графа. В программе должно быть реализовано текстовое меню. Также должна быть возможность выбора случайного или ручного задания графа и возможность сохранить результат работы программы в файл.

Устройство ввода информации - клавиатура и мышь.

Задание выполняется в соответствии с вариантом №13.

# **Теоретическая часть программы**

Дан ориентированный или неориентированный взвешенный граф с n вершинами и n рёбрами. Веса всех рёбер неотрицательны. Указана некоторая стартовая вершина s. Требуется найти длины кратчайших путей из вершины s во все остальные вершины, а также предоставить способ вывода самих кратчайших путей.

При представлении графа матрицей смежности веса рёбер хранятся в квадратной матрице, где присутствие пути из одной вершины в другую обозначается числами отличными от 0, при 0 – путь отсутствует.

Пусть расстояние от начальной вершины start до вершины i хранится в массиве minrast[i]. Начальные значения minrast[start]=0, min=10000 для всех остальных вершин i. То есть в самом начале алгоритму известен путь из вершины start до вершины start длины 0, а до остальных вершин кратчайшие пути неизвестны. Между тем алгоритм будет постепенно улучшать значения в массиве minrast, в результате получит кратчайшие расстояния до всех вершин. Алгоритм заканчивается, когда на очередном шаге не останется не посещённых вершин или если эти вершины являются недостижимыми.

# **Описание алгоритма программы**

Алгоритм находит кратчайшее расстояние от одной из вершин графа до всех остальных и работает только для графов без ребер отрицательного веса.

Каждой вершине приписывается вес – это вес пути от начальной вершины до данной. Также каждая вершина может быть выделена. Если вершина выделена, то путь от нее до начальной вершины кратчайший, если нет – то временный. Обходя граф, алгоритм считает для каждой вершины маршрут, и, если он оказывается кратчайшим, выделяет вершину. Весом данной вершины становится вес пути. Для всех соседей данной вершины алгоритм также рассчитывает вес, при этом ни при каких условиях не выделяя их. Алгоритм заканчивает свою работу, дойдя до конечной вершины, и весом кратчайшего пути становится вес конечной вершины.

Шаг 1. Всем вершинам, за исключением первой, присваивается вес равный бесконечности (в нашем случае 10000), а первой вершине – 0.

Шаг 2. Все вершины отмечаются не помещенными.

Шаг 3. Первая выбранная вершина s объявляется текущей.

Шаг 4. Вес всех невыделенных вершин пересчитывается по формуле: вес невыделенной вершины есть минимальное число из старого веса данной вершины, суммы веса текущей вершины и веса ребра, соединяющего текущую вершину с невыделенной.

Шаг 5. Среди невыделенных вершин ищется вершина с минимальным весом. Если таковая не найдена, то есть вес всех вершин равен бесконечности, то маршрут не существует. Следовательно, выход. Иначе, текущей становится найденная вершина. Она же выделяется.

Шаг 6. Если текущей вершиной оказывается конечная, то путь найден, и его вес есть вес конечной вершины.

Шаг 7. Переход на шаг 4.

Ниже представлен псевдокод алгоритма.

**Вход:** G – матрица смежности графа, s – исходная вершина.

**Выход:** minrast – вектор расстояний до всех вершин от вершины s.

**Алгоритм ПОШ**

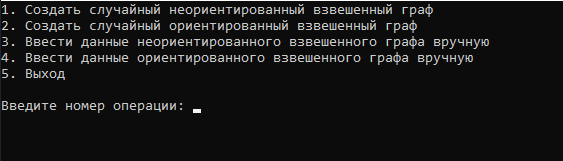
* 1. Для всех i положим minrast[i] = 10000 и пометим вершину как не посещенную posew[i] = -1.
  2. **ВЫПОЛНЯТЬ;**
  3. min = 10000, s = 10000;
  4. **ДЛЯ** i = 1 до size\_G **ВЫПОЛНЯТЬ**
  5. **ЕСЛИ** posew[i] = = -1 И minrast[i] < min
  6. **ТО**
  7. min = minrast[i], s = i;
  8. **ЕСЛИ** s != 10000
  9. **ДЛЯ** i = 1 до size\_G **ВЫПОЛНЯТЬ**
  10. **ЕСЛИ** G(s,i) > 0
  11. **ТО**
  12. Temp = min + G(s,i);
  13. **ЕСЛИ** temp < minrast[i]
  14. **ТО**
  15. minrast[i] = temp;
  16. пометить вершину как «посещенную» posew[s] = true;
  17. **ПОКА** s < 10000 продолжать с 2.1 по новой.

# **Описание программы**

Для написания программы использован язык программирования Си. Проект был создан в видео консольного приложения Win32.

Данная программа является многомодульной, поскольку состоит из нескольких функций: menu, CreateNeOr, CreateOr, CreateRuchNeor, CreateRuchOr, PrintGraph, init, dijkstra, main.

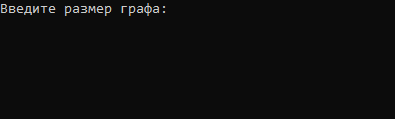
Работа программы начинается с вывода меню на экран:



1 Меню программы

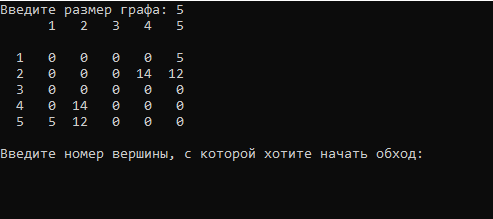
Пользователь может сгенерировать случайный граф или ввести его данные вручную.

При создании случайного графа у пользователя запрашивается его размер, который вводится с клавиатуры:



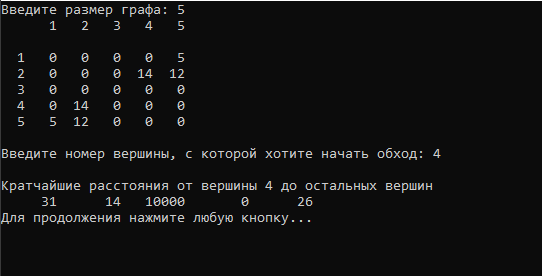
2. Запрос ввода размера графа

После вывода графа на экран запрашивается номер вершины, с которой будет начинаться обход:



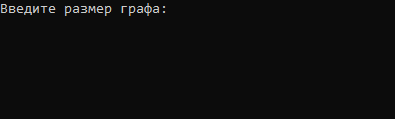
3. Вывод случайно сгенерированного неориентированного взвешенного графа на экран

Далее выводятся кратчайшие расстояния от заданной вершины до всех остальных:



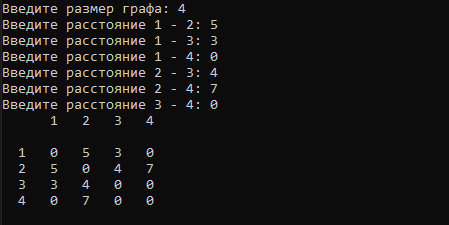
4. Результат случайно сгенерированного неориентированного взвешенного графа

При задании графа вручную сначала запрашивается размер графа:



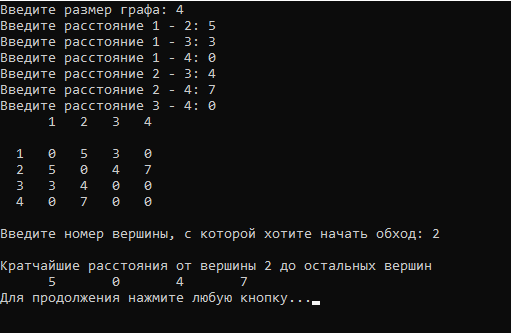
5. Запрос ввода размера графа

Следом запрашиваются расстояния между вершинами:



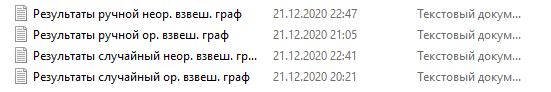
6. Запрос весов ребер и вывод неориентированного взвешенного графа

После вывода графа на экран запрашивается номер вершины, с которой будет начинаться обход, и выводятся кратчайшие расстояния:

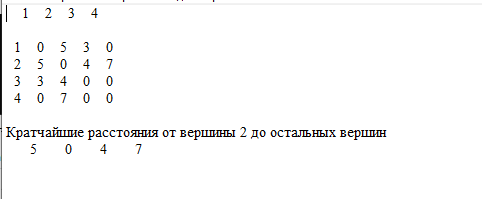


7. Результат неориентированного взвешенного графа введенного вручную

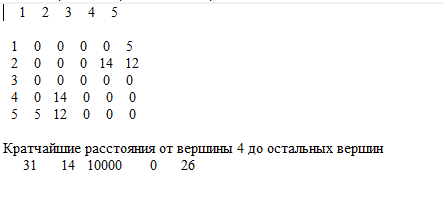
После успешного завершения программы происходит сохранение результата в текстовый документ, в котором данные обновляются каждый раз после очередного завершения.



8. Сохранение результатов в файлы



9. Сохранение неориентированного, взвешенного графа введенного вручную

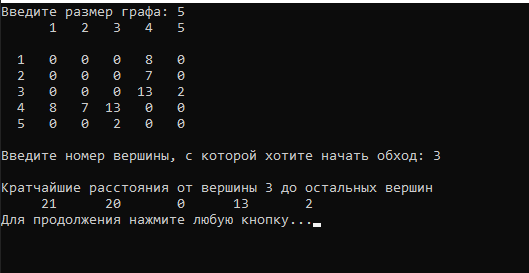


10. Сохранение результата, случайно сгенерированного неориентированного взвешенного графа

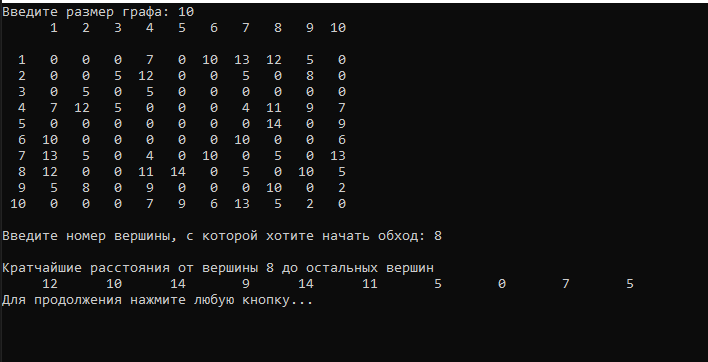
# **Тестирование**

Тестирование проводилось как в процессе разработки, так и после написания программы. В ходе тестирования было выявлены проблемы, связанные с сохранением в файл и вводом данных вручную в ориентированном графе.

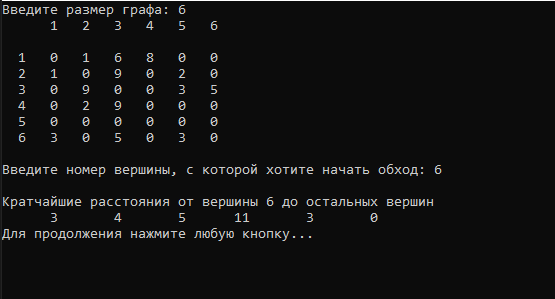
Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и вывод кратчайших путей.



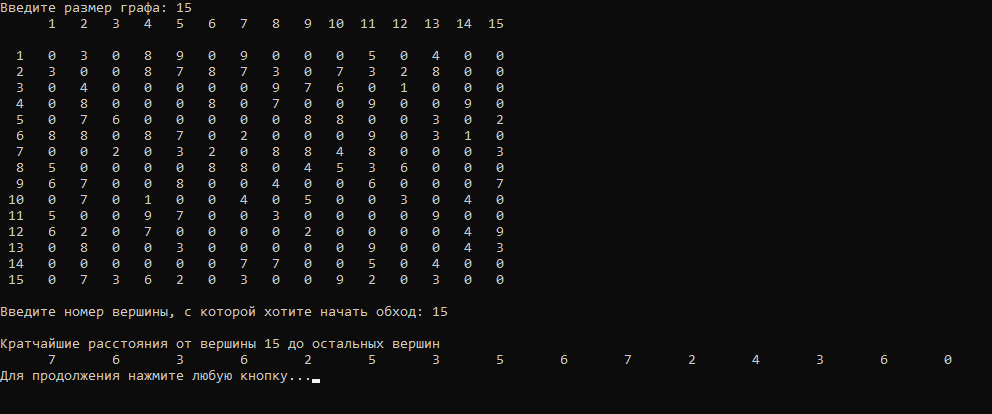
11. Случайно сгенерированный неориентированный взвешенный граф размерностью 5



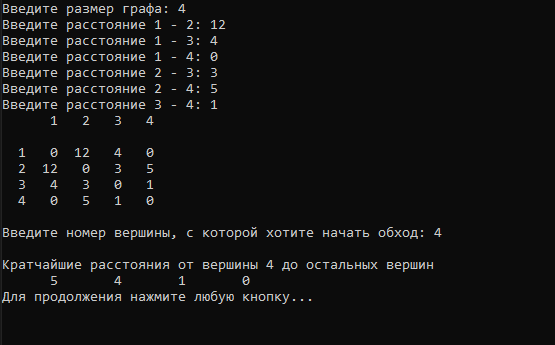
12. Случайно сгенерированный неориентированный взвешенный граф размерностью 10



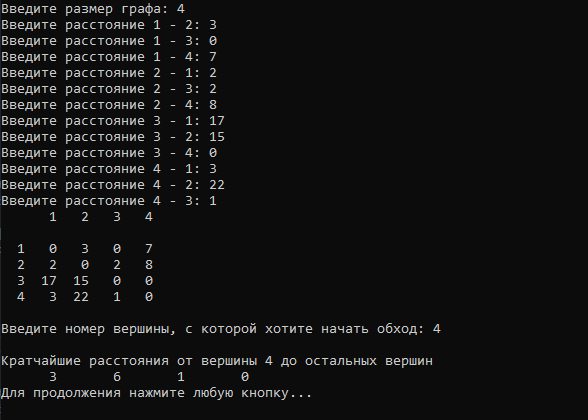
13. Случайно сгенерированный ориентированный взвешенный граф размерностью 6



14. Случайно сгенерированный ориентированный взвешенный граф размерностью 15



15. Неориентированный взвешенный граф введенный вручную размерностью 4



16. Ориентированный взвешенный граф введенный вручную размерностью 4

Таблица 1 - описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Появление меню на экране и сообщения о выборе нужной операции | Верно |
| Выбор генерации графа | Запрос ввода количества вершин | Верно |
| Ручной ввод графа | Запрос ввода расстояния между вершинами | Верно |
| Сохранение результата | Результат записан в файл | Верно |
| Результат работы программы | Верный вывод кратчайших расстояний | Верно |

В результате тестирования было выявлено, что программа работает успешно.

# **Ручной расчет задачи**

Проведем проверку программы посредством ручного расчета на примере графа с 5 вершинами (рисунок 11).

Метка самой вершины 3 полагается равной 0, метки остальных вершин – недостижимо большое число (в идеале — бесконечность). Это отражает то, что расстояния от вершины 3 до других вершин пока неизвестны. Все вершины графа помечаются как не посещённые.

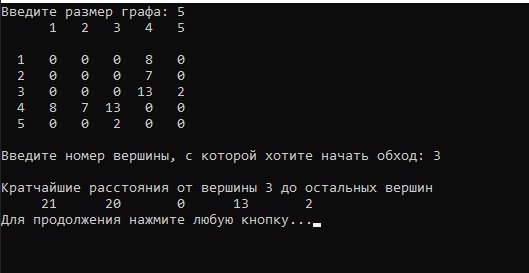
Начинаем обход из вершины 3 в последующие. Если есть путь из 3 в другие вершины, то идем дальше по графу. В нашем случае – в вершину 5, так как длина пути до нее минимальна. Длина пути в нее через вершину 3 равна   
0 + 2 = 2. 2 меньше текущей метки вершины 5, поэтому новая метка 5-ой вершины равна 2.

Аналогично находим пути для других соседей, в данном случае вершина 4. После того, как все соседи 3 вершины проверены, текущее минимальное расстояние до 3 вершины считается окончательным, а вершина 3 помечается «посещенной».

Далее идем в вершину 5. Так как она связана только с 3 вершиной, мы помечаем ее «посещенной» и идем в вершину 4 с меткой 13.

Вершина 4 связана с вершинами 3, 1 и 2. Вершина 3 уже посещена. Следующий сосед вершины 3 – вершина 2, так как имеет минимальную метку из непосещенных вершин. Если идти в нее через 4-ю вершину, то длина пути будет равна 13+7 = 20, так как 20 < 10000, устанавливаем метку 2 вершины равной 20. И последний сосед 4 вершины – вершина 1. Ее метка будет равна 13+8 = 21.

Таким образом, можно сделать вывод, что программа работает корректно.



17. Тестирование программы.

# **Заключение**

В процессе создания данного проекта разработана программа, реализующая поиск кратчайшего пути с помощью алгоритма Дейкстры в Microsoft Visual Studio 2019.

При выполнении данной курсовой работы были получена навыки разработки программ и работы с матрицами смежности. Приобретены навыки по осуществлению алгоритма Дейкстры.

Недостатком программы является примитивный пользовательский интерфейс и отсутствие нарисованного графа для легкого понимания человеку.

# **Список литературы**

1. <https://prog-cpp.ru/deikstra/>
2. <http://khpi-iip.mipk.kharkiv.edu/library/datastr/book_sod/kgsu/din_0123.html>
3. <https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B>
4. <https://www.youtube.com/watch?v=gVG349v5tsE>
5. <https://foxford.ru/wiki/informatika/algoritm-deykstry>

# **Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <time.h>

#include <locale.h>

FILE\* fp1;

void menu() {

printf("1. Создать случайный неориентированный взвешенный граф\n");

printf("2. Создать случайный ориентированный взвешенный граф\n");

printf("3. Ввести данные неориентированного взвешенного графа вручную\n");

printf("4. Ввести данные ориентированного взвешенного графа вручную\n");

printf("5. Выход\n\n");

}

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

srand(time(NULL));

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 CreateOr(int\*\* G, int n) {

srand(time(NULL));

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() % 10;

if (rand() % 100 > 50)

G[j][i] = 0;

else

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

}

else

if (rand() % 100 > 50) {

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

G[i][j] = 0;

}

else {

G[i][j] = 0;

G[j][i] = 0;

}

}

}

}

void CreateRuchNeor(int\*\* G, int n, int temp) {

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

{

G[i][i] = 0;

for (int j = i + 1; j < n; j++) {

printf("Введите расстояние %d - %d: ", i + 1, j + 1);

scanf("%d", &temp);

G[i][j] = temp;

G[j][i] = temp;

}

}

}

void CreateRuchOr(int\*\* G, int n, int temp) {

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

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

if (i == j) {

G[i][j] = 0;

}

else {

printf("Введите расстояние %d - %d: ", i + 1, j + 1);

scanf("%d", &temp);

G[i][j] = temp;

}

}

}

}

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

printf(" ");

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

printf("%4d", i + 1);

fprintf(fp1, "%5d", i + 1);

}

printf("\n\n");

fprintf(fp1, "\n\n");

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

printf("%3d", i + 1);

fprintf(fp1, "%3d", i + 1);

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

printf("%4d", G[i][j]);

fprintf(fp1, "%5d", G[i][j]);

}

printf("\n");

fprintf(fp1, "\n");

}

printf("\n");

}

void init(int\*\* G, int n, int\* minrast, int\* posew) {

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

minrast[i] = 10000;

posew[i] = -1; //вершины помечаются непосещенными

}

}

void dijkstra(int\*\* G, int start\_index, int n, int min, int s, int\* posew, int\* minrast, int temp) {

minrast[start\_index] = 0;

do {

min = 10000;

s = 10000;

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

if ((posew[i] == -1) && (minrast[i] < min)) { // Если вершину ещё не обошли и вес меньше min

min = minrast[i];

s = i;

}

}

if (s != 10000) {

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

if (G[s][i] > 0) {

temp = min + G[s][i]; // Добавляем найденный минимальный вес к текущему весу вершины

if (temp < minrast[i]) { // сравниваем с текущим минимальным весом вершины

minrast[i] = temp;

}

}

}

posew[s] = 0; //вершина помечается посещенной

}

} while (s < 10000);

printf("\nКратчайшие расстояния от вершины %d до остальных вершин\n", start\_index+1);

fprintf(fp1, "\nКратчайшие расстояния от вершины %d до остальных вершин\n", start\_index + 1);

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

printf("%7d ", minrast[i]);

fprintf(fp1, "%7d ", minrast[i]);

}

}

int main() {

int num, min = NULL, s = NULL;

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

int\* minrast, \*posew, start\_index;

int n, temp = NULL, start = NULL, save;

setlocale(LC\_ALL, "rus");

while (true) {

system("cls");

menu();

printf("Введите номер операции: ");

scanf("%d", &num);

if (num == 1) {

system("cls");

printf("Введите размер графа: ");

scanf("%d", &n);

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

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

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

}

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

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

fp1 = fopen("Результаты случайный неор. взвеш. граф.txt", "w");

CreateNeOr(G, n);

PrintGraph(G, n);

printf("Введите номер вершины, с которой хотите начать обход: ");

scanf("%d", &start\_index);

while (start\_index < 1 || start\_index > n) {

printf("Требуемая вершина отсутствует. Введите данные корректно:");

scanf("%d", &start\_index);

}

start\_index--;

init(G, n, minrast, posew);

dijkstra(G, start\_index, n, min, s, posew, minrast, temp);

fclose(fp1);

printf("\nДля продолжения нажмите любую кнопку...");

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

free(G[i]);

}

free(G);

free(minrast);

free(posew);

getchar();

getchar();

}

if (num == 2) {

system("cls");

printf("Введите размер графа: ");

scanf("%d", &n);

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

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

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

}

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

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

fp1 = fopen("Результаты случайный ор. взвеш. граф.txt", "w");

CreateOr(G, n);

PrintGraph(G, n);

printf("Введите номер вершины, с которой хотите начать обход: ");

scanf("%d", &start\_index);

while (start\_index < 1 || start\_index > n) {

printf("Требуемая вершина отсутствует. Введите данные корректно:");

scanf("%d", &start\_index);

}

start\_index--;

init(G, n, minrast, posew);

dijkstra(G, start\_index, n, min, s, posew, minrast, temp);

fclose(fp1);

printf("\nДля продолжения нажмите любую кнопку...");

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

free(G[i]);

}

free(G);

free(minrast);

free(posew);

getchar();

getchar();

}

if (num == 3) {

system("cls");

printf("Введите размер графа: ");

scanf("%d", &n);

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

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

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

}

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

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

fp1 = fopen("Результаты ручной неор. взвеш. граф.txt", "w");

CreateRuchNeor(G, n, temp);

PrintGraph(G, n);

printf("Введите номер вершины, с которой хотите начать обход: ");

scanf("%d", &start\_index);

while (start\_index < 1 || start\_index > n) {

printf("Требуемая вершина отсутствует. Введите данные корректно:");

scanf("%d", &start\_index);

}

start\_index--;

init(G, n, minrast, posew);

dijkstra(G, start\_index, n, min, s, posew, minrast, temp);

fclose(fp1);

printf("\nДля продолжения нажмите любую кнопку...");

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

free(G[i]);

}

free(G);

free(minrast);

free(posew);

getchar();

getchar();

}

if (num == 4) {

system("cls");

printf("Введите размер графа: ");

scanf("%d", &n);

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

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

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

}

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

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

fp1 = fopen("Результаты ручной ор. взвеш. граф.txt", "w");

CreateRuchOr(G, n, temp);

PrintGraph(G, n);

printf("Введите номер вершины, с которой хотите начать обход: ");

scanf("%d", &start\_index);

while (start\_index < 1 || start\_index > n) {

printf("Требуемая вершина отсутствует. Введите данные корректно:");

scanf("%d", &start\_index);

}

start\_index--;

init(G, n, minrast, posew);

dijkstra(G, start\_index, n, min, s, posew, minrast, temp);

fclose(fp1);

printf("\nДля продолжения нажмите любую кнопку...");

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

free(G[i]);

}

free(G);

free(minrast);

free(posew);

getchar();

getchar();

}

if (num == 5) {

break;

}

}

}