Министерство науки и образования Российской Федерации Пензенский государственный университет

Кафедра «Вычислительная техника»

Пояснительная записка

К курсовому проектированию

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация алгоритма Дейкстры»

Выполнил:

Студент группы 22ВВС1

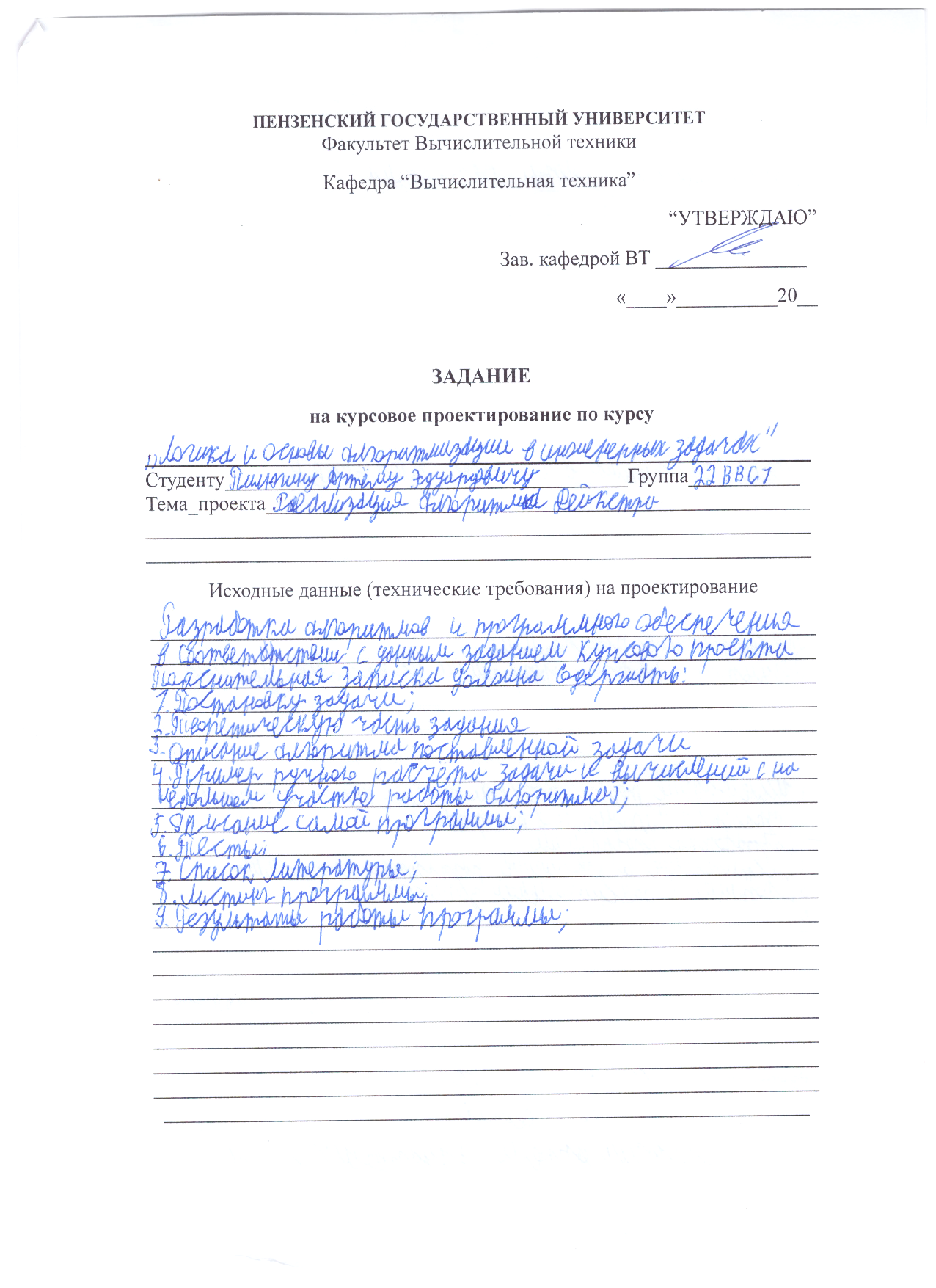
Пилюгин А.Э

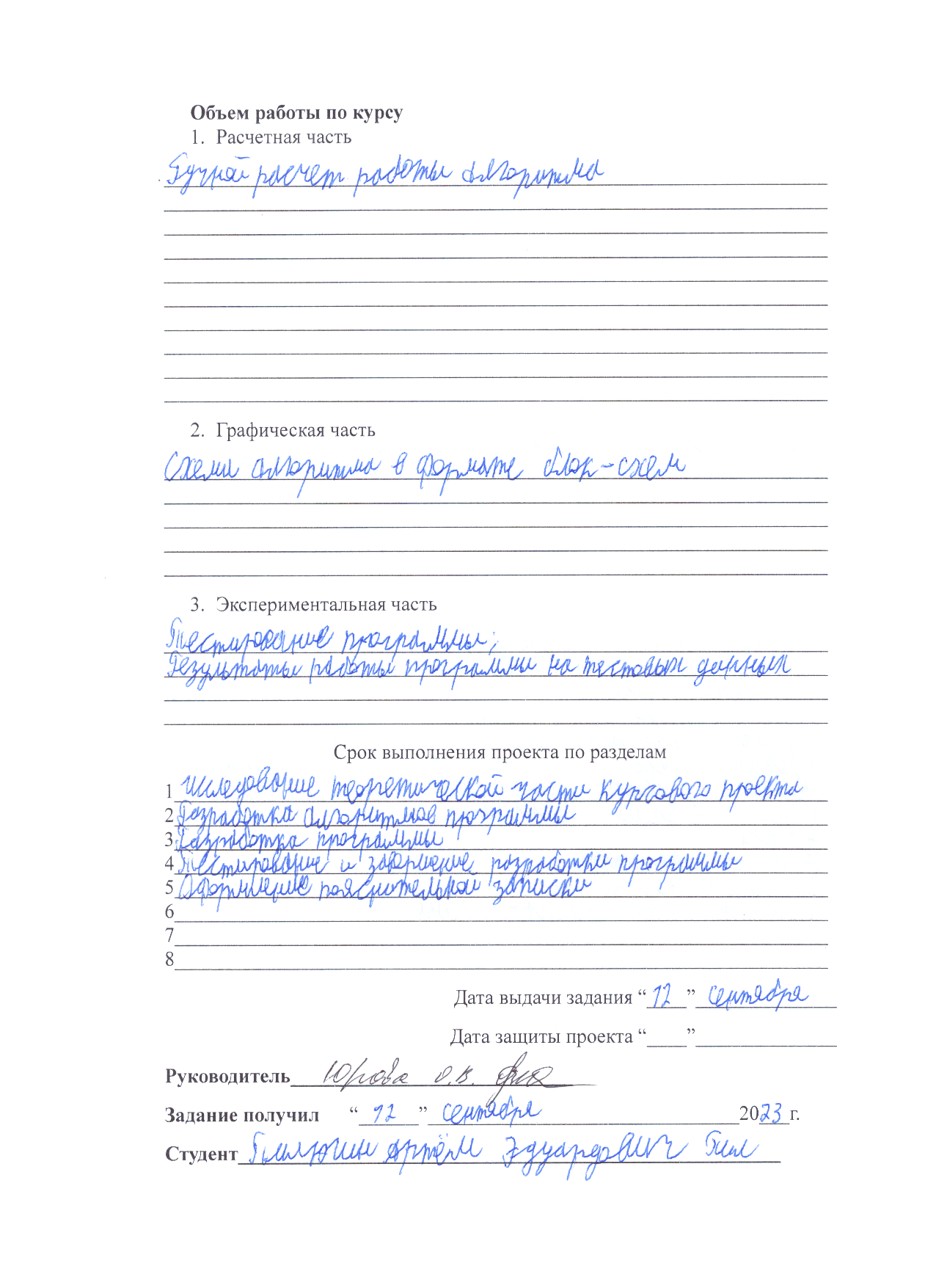
Принял:

Акифьев И.В.

Юрова О.В.

Пенза 2023





## Оглавление

[Реферат 8](#_bookmark0)

[Введение 9](#_bookmark1)

1. [Постановка задачи 10](#_bookmark2)
2. [Теоретическая часть задания 11](#_bookmark3)
3. [Описание алгоритма программы 12](#_bookmark4)
4. [Описание программы 13](#_bookmark5)
5. [Тестирование 15](#_bookmark6)
6. [Ручной расчёт задачи 17](#_bookmark7)

[Заключение 19](#_bookmark8)

[Список литературы 20](#_bookmark9)

[Приложение А 21](#_bookmark10)

# Реферат

Отчёт стр. 20 , рисунков 14, таблиц 1.

ГРАФ, ТЕОРИЯ ГРАФОВ, ДОСТИЖИМОСТЬ, АЛГОРИТМ ДЕЙКСТРЫ.

Цель исследования – разработка программы, в которой реализован алгоритм Дейкстры.

# Введение

Целью данной курсовой работы является создание программы, которая будет искать кратчайший путь до вершины.

Алгоритм Дейкстры — 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, например, его используют протоколы маршрутизации OSPF и IS-IS.

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudio2022, язык программирования – С.

# Постановка задачи

Требуется разработать программу, которая представит граф в виде матрицы смежности, найдет кратчайший путь до вершин.

Программа должна иметь текстовое или графическое меню.

При генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы.

После обработки этих данных пользователь вручную заполнял матрицу. Необходимо сделать возможность случайного заполнения матрицы.

Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Результат выполнения программы должен сохраняться в файл.

# Теоретическая часть задания

Для того, чтобы реализовать алгоритм Дейкстры нужно знать, что граф не должен иметь ребра отрицательного веса.

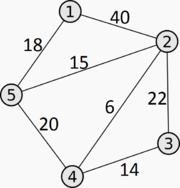


Рисунок 1 – Пример правильного построения графа

# Описание алгоритма программы

Реализация алгоритма Дейкстры выполнена следующим образом:

**Пока** minindex < 10000:

minindex = 10000

min = 10000

**Для каждого** i **от** 0 **до** SIZE:

**Если** вершина i не посещена (v[i] == 1) и расстояние до нее меньше текущего минимума (d[i] < min):

min = d[i]

minindex = i

**Если** minindex **не равно** 10000:

**Для каждого** i от 0 до SIZE:

**Если** есть связь между вершинами minindex и i (a[minindex][i] > 0):

s = min + a[minindex][i]

**Если** новое расстояние s меньше текущего d[i]:

d[i] = s

**Пометить вершину minindex как посещенную** (v[minindex] = 0)

Полный код программы можно увидеть в листинге.

# Описание программы

Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до a.

Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки.

Работа алгоритма завершается, когда все вершины посещены. Программа начинается с вывода в консоль текстового меню.

Пользователю предлагается задать количество вершин в графе. Если

пользователь ввёл меньше 1 вершины, программа выведет в консоль сообщение «Введено неверное значение» и запросит повторный ввод (Рисунок 2).

Рисунок 2 – пример ошибки ввода вершин

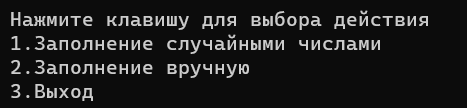
Если пользователь ввел правильно число вершин, то программа выведет текстовое меню. В меню пользователю предлагают выбрать вариант заполнения матрицы смежности: заполнить матрицу случайным образом с помощью функции rand (), нажав клавишу 1, либо заполнить матрицу вручную с клавиатуры, нажав клавишу 2. Если пользователь нажмет клавишу 3 программа завершится. (Рисунок 3).

Рисунок 3 –выбор выхода из программы

Выбор пользователем режима заполнения матрицы был реализован с помощью функции \_getch().

При заполнении матрицы смежности пользователем учитывается, что в графе не может быть отрицательный вес ребра, то есть можно будет ввести только числа больше или равные 0.

После заполнения матрицы программа выводит её в консоль и в файл, который пользователь именует сам (Рисунок 4,5)

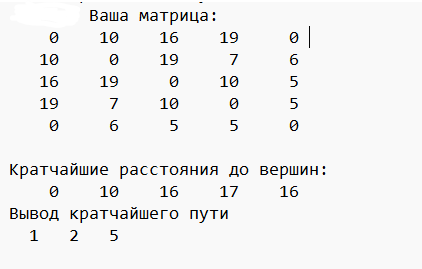
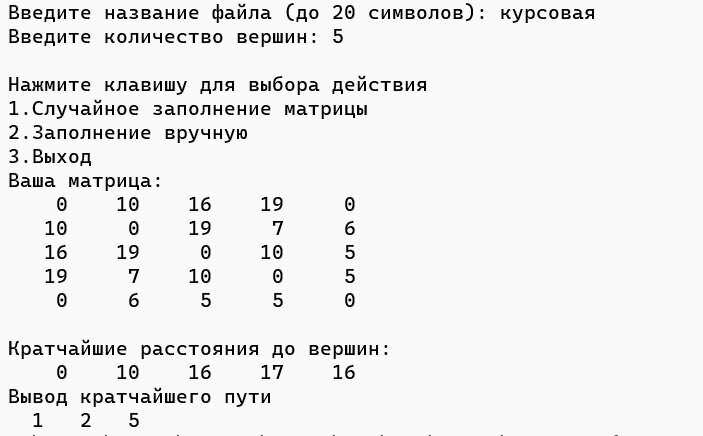
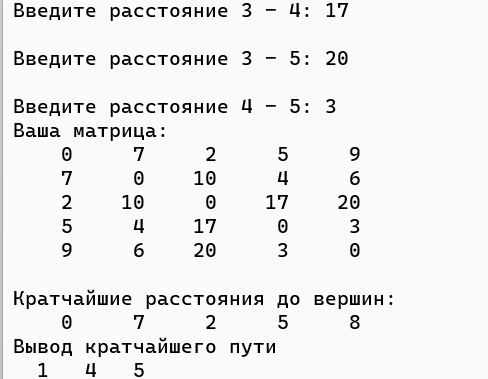
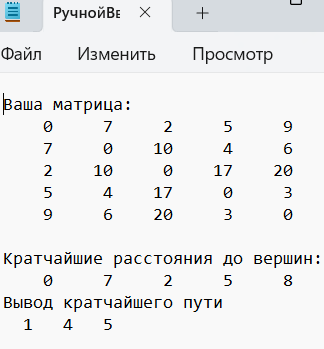


Рисунок 4 – вывод случайно матрицы в консоль и файл

Рисунок 5 – ручной ввод матрицы, вывод в консоль и файл

После создания матрицы программа найдет и выведет кратчайшие пути до каждой вершины, после выведет кратчайший путь до конечной вершины (через какие вершины нужно идти).

# Тестирование

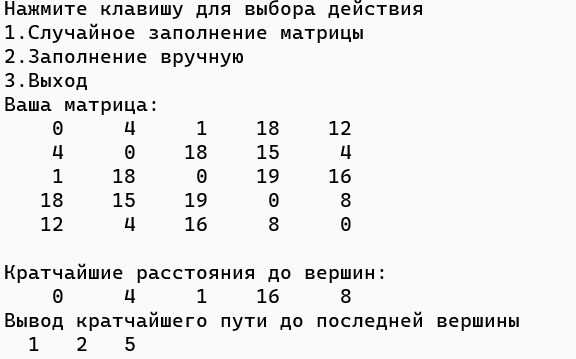
Выполнение программы с 5 вершинами в графе при случайном заполнении матрицы (Рисунок 6)

Рисунок 6 – тестирование случайного заполнения

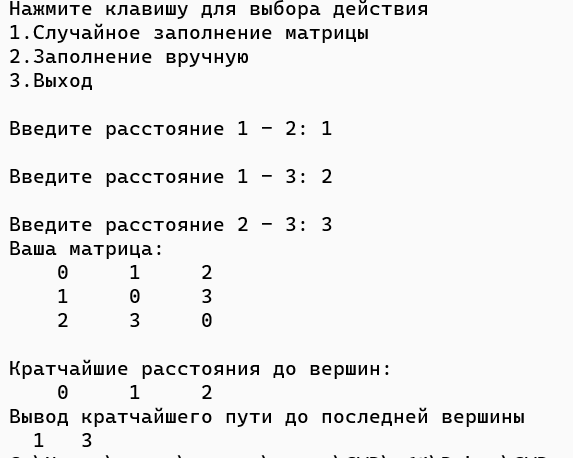
Выполнение программы с 5 вершинами в графе при ручном заполнении матрицы (Рисунок 7)

Рисунок 7 – тестирование ручного заполнение

Выполнение программы с некорректным вводом вершин (Рисунок 8)



Рисунок 8 – тестирование матрицы с неправильным вводом Выполнение программы с некорректным вводом веса рёбер

(Рисунок 9)

Рисунок 9 – Тестирование с неправильным вводом

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод текстового меню с выбором режима генерации матрицы смежности графа | верно |
| Выбор генерации матрицы | Вывод сообщения о выборе количества вершин в графе | верно |
| Выбор количества вершин в графе | Вывод матрицы, если был выбран автоматический метод генерации либо заполнение и вывод матрицы, если был  выбран ручной метод генерации | верно |
| Нахождение кратчайшего пути | Вывод кратчайшего пути | верно |

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.

# Ручной расчёт задачи

Пример работы программы для графа с 3 вершинами (Рисунок 10)

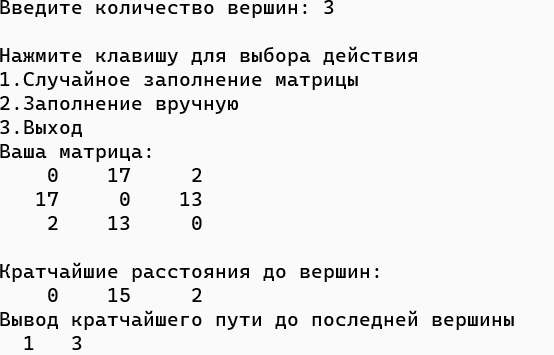
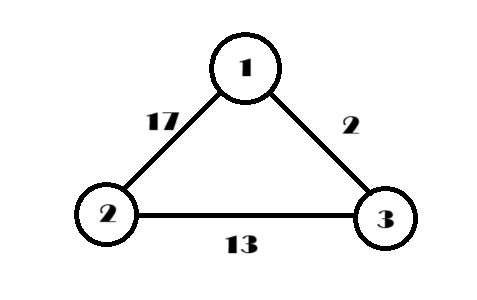
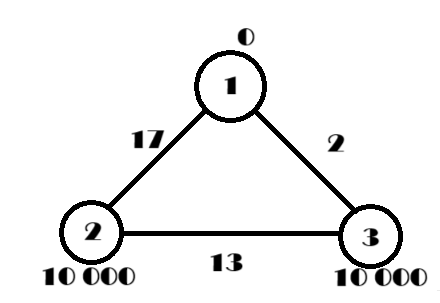
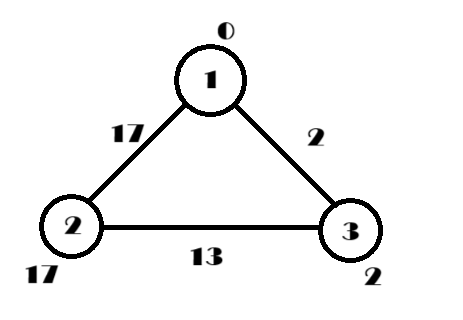
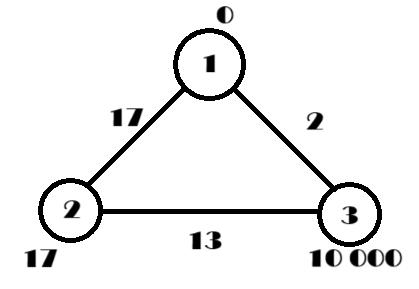


Рисунок 10 – выполнение для графа из 3 вершин Воссоздадим граф по заданной матрице смежности (Рисунок 11)



Рисунок 11 – граф из 3 вершин

 Рисунок 12,13,14 – работа алгоритма Дейкстры

Алгоритм присваивает всем вершинам практически недостижимое число. Используя обход в ширину, проверяет вес пути с тем, который присвоен вершине. Если вес меньше, то вершине присваивается новое число – вес пути.

Ручной расчёт совпал с программным, следовательно, программа работает верно.

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритма Дейкстры в MicrosoftVisualStudio 2022. При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории графов. Углублены знания языка программирования C.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса. Программа имеет небольшой, но достаточный для использования функционал возможностей.

Уже в ходе защиты данной курсовой работы были добавлены возможности взаимодействия с ориентированным и неориентированным графом. Помимо этого, стал возможным выбор вершины, с который начинается поиск кратчайшего пути.

# Список литературы

1. «Грокаем алгоритмы» - Адитья Бхатгава

2. «ЯЗЫК С» Б.В. Керниган, Д.М. Ритчи.

# Приложение А

# Листинг программы

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <conio.h>

#include <locale.h>

#include <time.h>

int main() {

FILE\* fp;

int s = -1, minindex, min, SIZE = 0, menu = 0;

int\*\* a; // матрица связей

int\* d; // минимальное расстояние

int\* v; // посещенные вершины

int begin\_index = 0;

char name[20];

char key;

system("chcp 1251");

system("cls");

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

printf("Введите название файла (до 20 символов): ");

gets\_s(name);

fp = fopen(name, "a");

printf("Введите количество вершин: ");

while (SIZE <= 0) {

scanf\_s("%d", &SIZE);

if (SIZE <= 0) {

printf("Введено неверное значение\n");

printf("Введите количество вершин: ");

}

}

int isRandomFill = 0; // Флаг для определения способа заполнения: 0 - случайное, 1 - ручное

printf("Выберите способ заполнения матрицы:\n0 - Случайное заполнение\n1 - Ручное заполнение\n");

scanf\_s("%d", &isRandomFill);

int isDirected = 0; // Флаг для определения типа графа: 0 - неориентированный, 1 - ориентированный

printf("Выберите тип графа:\n0 - Неориентированный\n1 - Ориентированный\n");

scanf\_s("%d", &isDirected);

printf("Выберите начальную вершину (от 1 до %d): ", SIZE);

while (begin\_index < 1 || begin\_index > SIZE) {

scanf\_s("%d", &begin\_index);

if (begin\_index < 1 || begin\_index > SIZE) {

printf("Неверный номер вершины. Выберите от 1 до %d: ", SIZE);

}

}

begin\_index--; // Уменьшаем на 1 для корректной работы с индексами массивов (индексация с 0)

a = (int\*\*)malloc(SIZE \* sizeof(int\*));

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

a[i] = (int\*)malloc(SIZE \* sizeof(int));

if (isRandomFill == 0) {

if (isDirected == 0) {

// Неориентированный граф (случайное заполнение)

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

a[i][i] = 0;

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

a[i][j] = rand() % 20;

a[j][i] = a[i][j];

}

}

}

else {

// Ориентированный граф (случайное заполнение)

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

a[i][i] = 0;

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

if (i != j) {

a[i][j] = rand() % 20;

}

}

}

}

}

else {

// Ручное заполнение (для ориентированного и неориентированного графа)

if (isDirected == 0) {

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

a[i][i] = 0;

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

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

scanf\_s("%d", &s);

if (s < 0) {

while (s < 0) {

printf("Введено неверное значение");

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

scanf\_s("%d", &s);

}

}

a[i][j] = s;

a[j][i] = s;

}

}

}

else {

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

a[i][i] = 0;

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

printf("\nВведите расстояние от %d к %d: ", i + 1, j + 1);

scanf\_s("%d", &s);

if (s < 0) {

while (s < 0) {

printf("Введено неверное значение");

printf("\nВведите расстояние от %d к %d: ", i + 1, j + 1);

scanf\_s("%d", &s);

}

}

a[i][j] = s;

}

}

}

}

printf("Ваша матрица:\n");

fprintf(fp, "Ваша матрица:\n");

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

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

printf("%5d ", a[i][j]);

fprintf(fp, "%5d ", a[i][j]);

}

printf("\n");

fprintf(fp, "\n");

}

v = (int\*)malloc(SIZE \* sizeof(int));

d = (int\*)malloc(SIZE \* sizeof(int));

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

d[i] = 10000;

v[i] = 1;

}

d[begin\_index] = 0;

do {

minindex = 10000;

min = 10000;

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

if ((v[i] == 1) && (d[i] < min)) {

min = d[i];

minindex = i;

}

}

if (minindex != 10000) {

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

if (a[minindex][i] > 0) {

s = min + a[minindex][i];

if (s < d[i]) {

d[i] = s;

}

}

}

v[minindex] = 0;

}

} while (minindex < 10000);

printf("\nКратчайшие расстояния до вершин: \n");

fprintf(fp, "\nКратчайшие расстояния до вершин: \n");

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

printf("%5d ", d[i]);

fprintf(fp, "%5d ", d[i]);

}

int\* ver;

int end = SIZE - 1;

ver = (int\*)malloc(SIZE \* sizeof(int));

ver[0] = end + 1;

int k = 1;

int weight = d[end];

while (end != begin\_index) {

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

if (a[i][end] != 0) {

int s = weight - a[i][end];

if (s == d[i]) {

weight = s;

end = i;

ver[k] = i + 1;

k++;

}

}

}

}

printf("\nВывод кратчайшего пути до последней вершины\n");

fprintf(fp, "\nВывод кратчайшего пути\n");

for (int i = k - 1; i >= 0; i--) {

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

fprintf(fp, "%3d ", ver[i]);

}

fclose(fp);

free(a);

free(v);

free(d);

free(ver);

return 0;

}