Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет  
Кафедра «Вычислительной техники»

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

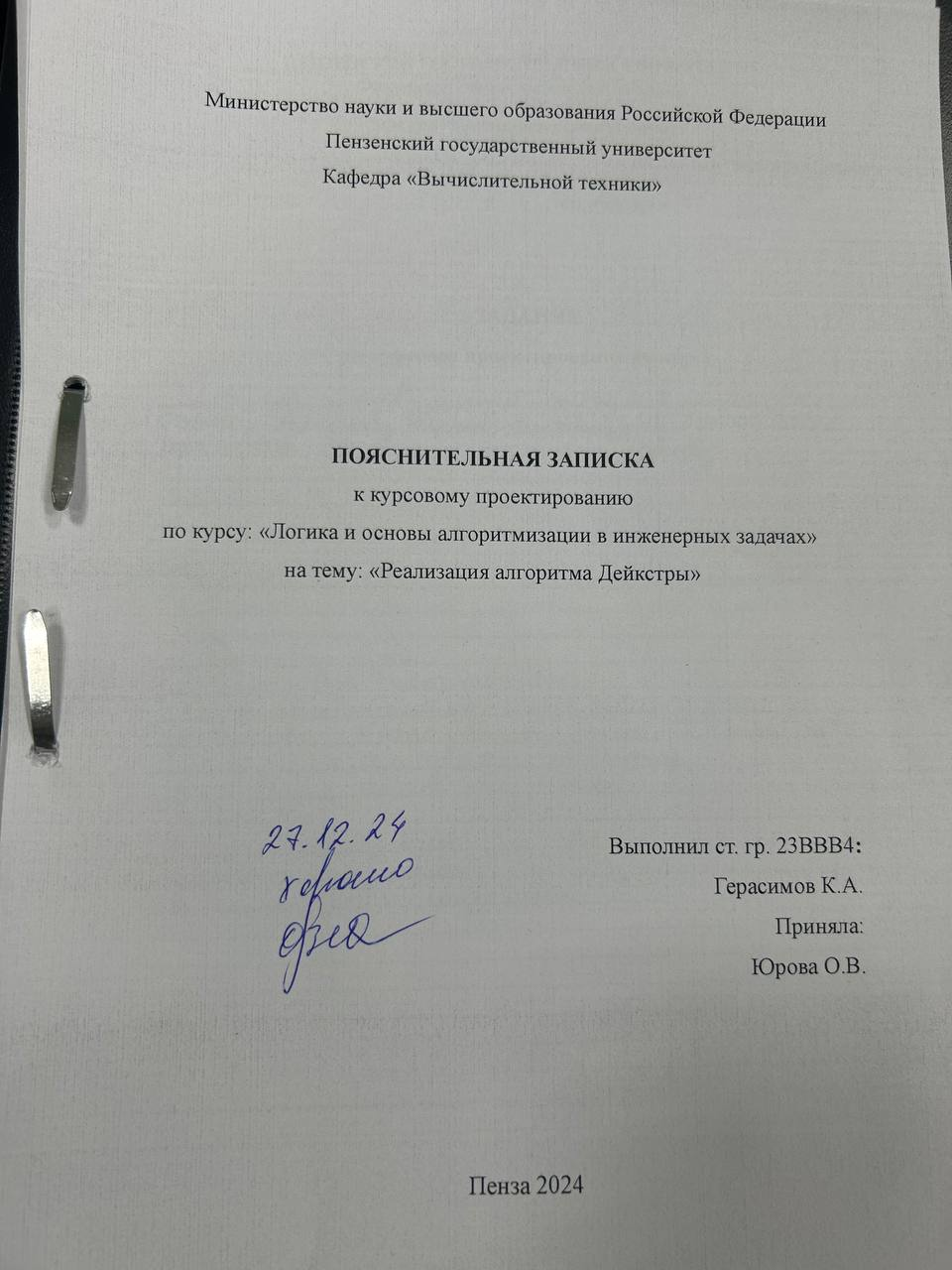
Выполнил ст. гр. 23ВВВ4**:**

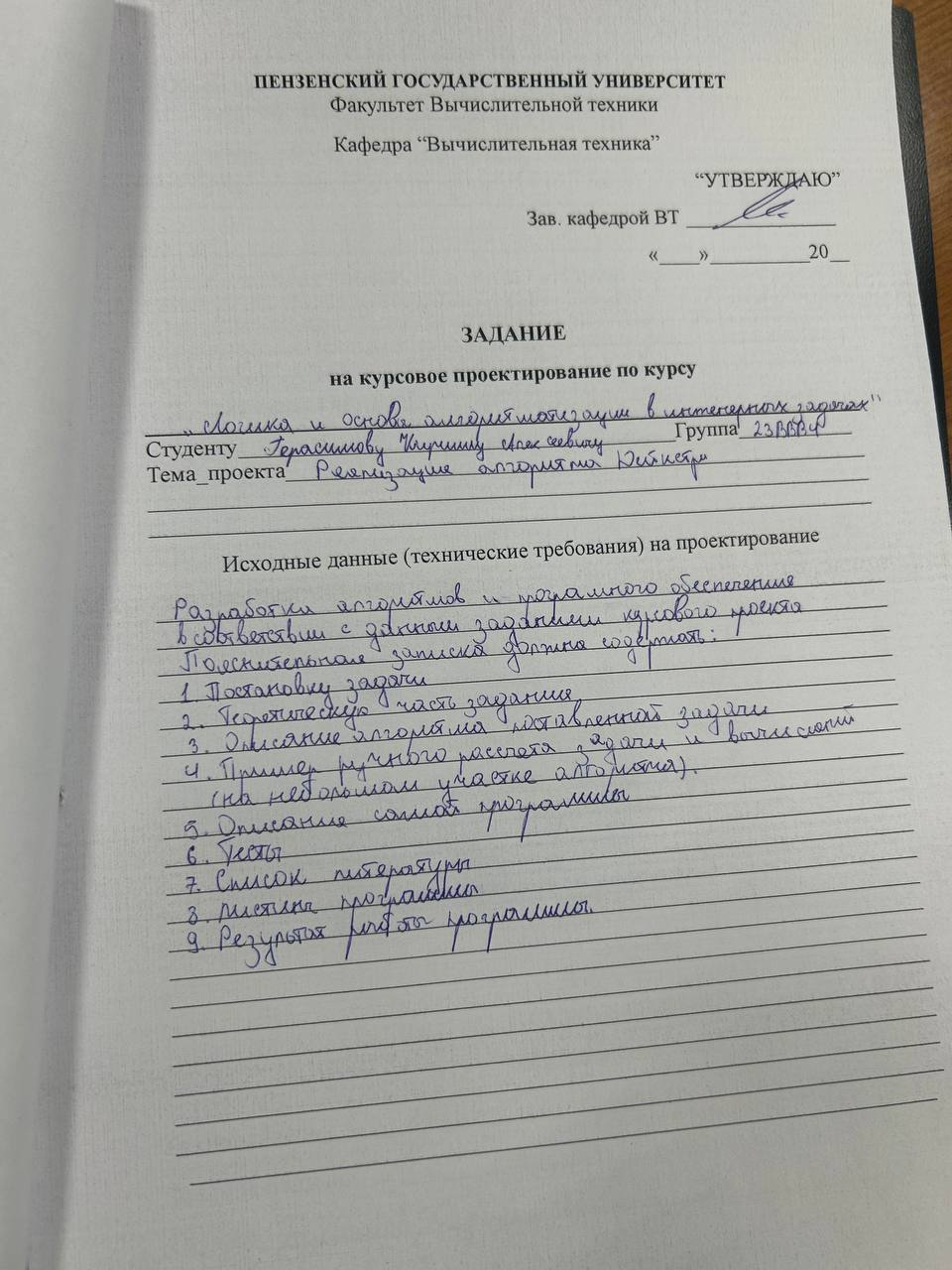
Герасимов К.А.

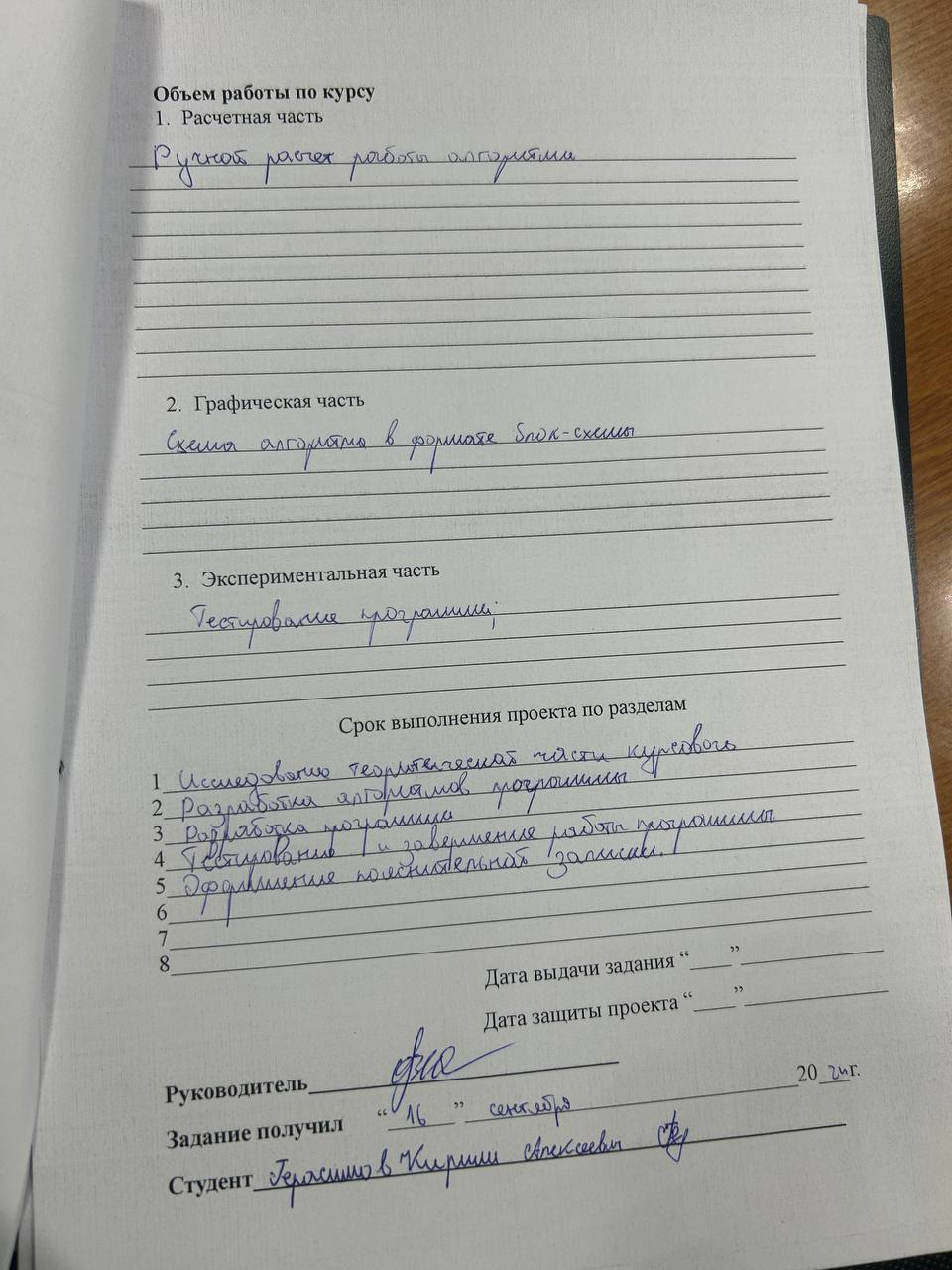
Приняла:

Юрова О.В.

Пенза 2024







**Реферат 3**

[**Введение**](#_4d34og8) **4**

**1.Постановка Задачи** [**5**](#_4d34og8)

[**2.Теоретическая часть задания 7**](#_17dp8vu)

[**3.Описание алгоритма программы**](#_26in1rg) **9**

**4. Описание программы** [**1**](#_26in1rg)**1**

[**5.Тестирование**](#_35nkun2) **17**

**6.Ручной расчёт программы 22**

**Заключение 23**

**Список Литературы 24**

**Приложение А 25**

**Реферат**

**Отчет 27 страниц, 17 рисунков, 1 таблица, 1 приложение**

РЕАЛИЗАЦИЯ АЛГОРИТМА ДЕЙКСТРЫ, ТЕОРИЯ ГРАФОВ, ГРАФ, КРАТЧАЙШИЕ ПУТИ, МАТРИЦА КРАТЧАЙШИХ РАССТОЯНИЙ, ВЕС РЕБЕР.

Цель исследования – разработка программы, предназначенная для нахождения кратчайших путей между всеми парами вершин в графе, как ориентированном, так и неориентированном, используя алгоритм Дейкстры. В ходе курсовой работы была разработана программа, вычисляющая кратчайший путь и сохраняющая его в файл.

# 

# 

# **Введение**

Целью данной курсовой работы является реализация алгоритма Дейкстры для нахождения кратчайших путей в ориентированном взвешенном графе. Алгоритм, разработанный нидерландским учёным Эдсгером Дейкстрой в 1959 году, стал основой для многих современных подходов к решению задач оптимизации на графах. Основная идея алгоритма заключается в последовательном улучшении оценок расстояний до вершин, используя принцип жадного выбора.

Алгоритм Дейкстры предназначен для нахождения кратчайших путей в графе с неотрицательными весами рёбер. Он работает, начиная с одной выбранной вершины (источника), и последовательно определяет минимальное расстояние от этой вершины до всех других вершин графа. Результатом выполнения алгоритма является либо кратчайший путь от начальной вершины до конкретной целевой вершины, либо таблица с минимальными расстояниями до всех вершин графа.

Программа написана на языке программирования C, который предоставляет низкоуровневый доступ к памяти и позволяет детально контролировать процессы вычислений. Для разработки использована среда Microsoft Visual Studio 2022, обеспечивающая удобный интерфейс для написания, тестирования и отладки кода.

**1. Постановка задачи**

Целью данной курсовой работы является разработка программы, которая решает задачу нахождения кратчайших путей в ориентированном взвешенном графе с использованием алгоритма Дейкстры. Программа должна представлять граф в виде матрицы смежности и обеспечивать возможность нахождения кратчайших путей от одной вершины до всех остальных. Пользователь должен иметь возможность взаимодействовать с программой через текстовое или графическое меню. Устройства ввода – клавиатура и мышь.

# 

# **2.Теоретическая часть задания**

Находит кратчайшие пути от одной из вершин графа G (рисунок 1) до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Каждой вершине из множества вершин V сопоставим метку — минимальное известное расстояние от этой вершины до стартовой вершины a.

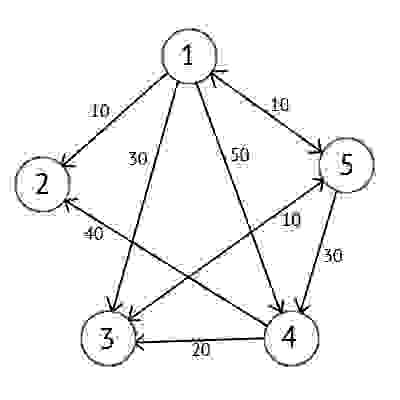


Рисунок 1 - Граф G

Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

**Инициализация.**

Метка самой вершины a полагается равной 0, метки остальных вершин — бесконечности.

Это отражает то, что расстояния от a до других вершин пока неизвестны. Все вершины графа помечаются как непосещённые.

**Шаг алгоритма.**

Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина u, имеющая минимальную метку.

Мы рассматриваем все возможные маршруты, в которых u является предпоследним пунктом. Вершины, в которые ведут рёбра из u, назовём соседями этой вершины. Для каждого соседа вершины u, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки u и длины ребра, соединяющего u с этим соседом.

**Завершение выполнения алгоритма.**

Алгоритм заканчивает работу, когда все вершины посещены. Если в какой-то момент все непосещённые вершины помечены бесконечностью, то это значит, что до этих вершин нельзя добраться (то есть граф несвязный). Тогда алгоритм может быть завершён досрочно.

Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину u как посещённую и повторим шаг алгоритма.

Алгоритм Дейкстры находит множество практических применений в жизни и бизнесе благодаря своей способности решать задачи оптимизации. **Навигационные системы**: Используется в GPS-приложениях для построения кратчайшего маршрута между двумя точками. Например, прокладывание пути для автомобилей, велосипедистов или пешеходов. **Логистика**: Помогает оптимизировать маршруты доставки товаров, минимизируя время или расходы на транспортировку. **Интернет и телекоммуникации**: В сетевых протоколах, таких как OSPF (Open Shortest Path First), для определения оптимального пути передачи данных. Это обеспечивает быструю и эффективную маршрутизацию в компьютерных сетях. **Оптимизация затрат**: Используется для анализа минимальных затрат или рисков при перемещении ресурсов, например, в инвестиционных или торговых стратегиях. **Расчёты в графах транзакций**: Например, для нахождения самого дешёвого пути перевода денег через посредников. **Транспортные системы**: Планирование дорожных сетей или маршрутов общественного транспорта, чтобы сократить пробки и улучшить мобильность.

# 

# 

# 

# **3.Описание алгоритма программы**

Для начала, пользователю необходимо ввести полное имя файла, в который будет сохранен результат. Если указанный файл не существует, он будет создан. Далее пользователю предлагается выбрать способ создания матрицы смежности и ввести количество вершин графа.

Данный алгоритм находит кратчайшее расстояние от одной вершины графа до всех остальных и предназначен для работы с графами, в которых нет рёбер с отрицательным весом. Каждая вершина в графе имеет вес, который соответствует минимальному пути от начальной вершины до данной. Вершины могут быть помечены как обработанные или не обработанные. Если вершина помечена как обработанная, это означает, что найден кратчайший путь от неё до начальной вершины. Если вершина не обработана, это означает, что путь ещё не найден и она является временной.

Процесс работы алгоритма можно описать следующими шагами:

1. Всем вершинам, кроме начальной, присваивается бесконечное значение (10000), а начальной вершине — значение 0.
2. Все вершины считаются необработанными.
3. Первая выбранная вершина («s») становится текущей.
4. Для каждой вершины, которая еще не была обработана, вычисляется новый вес, равный минимуму из старого значения веса, суммы веса текущей вершины и веса ребра, соединяющего её с текущей вершиной.
5. Среди всех необработанных вершин выбирается вершина с наименьшим весом. Если такая вершина не найдена, это означает, что путь невозможен, и алгоритм завершает выполнение. В противном случае выбранная вершина становится текущей и помечается как обработанная.
6. Если текущая вершина совпадает с конечной, значит, кратчайший путь найден, и его вес равен весу конечной вершины.
7. Алгоритм возвращается к шагу 4, повторяя процесс для новых вершин, пока не будет найден кратчайший путь.

**4.Описание программы**

Данная программа написана на языке программирования Си, который отличается универсальностью и сочетанием возможностей низкоуровневого и высокоуровневого программирования. Программа реализована в формате консольного приложения для операционной системы Windows.

Код состоит из нескольких функций, каждая из которых выполняет определённую задачу. Среди них:

* main — управление логикой работы программы,
* printGraphMatrix — вывод матрицы смежности,
* initializeArrays — подготовка массивов расстояний и посещений,
* printDistances — отображение кратчайших расстояний,
* restorePath — восстановление пути между вершинами.

Работа программы начинается с выбора действия из меню. Пользователю предлагается:

1. Создать граф с помощью случайной генерации
2. Ввести матрицу смежности вручную
3. Завершить выполнение программы.

При создании графа пользователь вводит количество вершин, а также выбирает, будет ли граф ориентированным и/или взвешенным. Если пользователь предпочитает ввод данных вручную, программа запрашивает значения матрицы смежности.

После генерации или ввода графа программа отображает его матрицу смежности и рассчитывает кратчайшие расстояния от выбранной начальной вершины до остальных вершин с использованием алгоритма поиска кратчайшего пути. Также предусмотрена возможность восстановления пути от стартовой до конечной вершины.

Результаты работы программы сохраняются в текстовый файл, имя которого указывается пользователем. Итоговая матрица смежности, кратчайшие расстояния и восстановленный путь выводятся как в консоль, так и в файл.

Для сохранения файла нужно ввести его название (Рисунок 2)

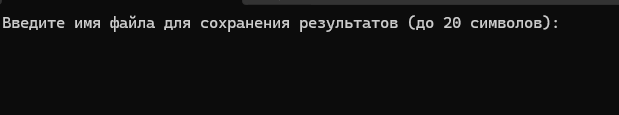


Рисунок 2 - Меню присваивания названия файла

После выбрать количество вершин (Рисунок 3).

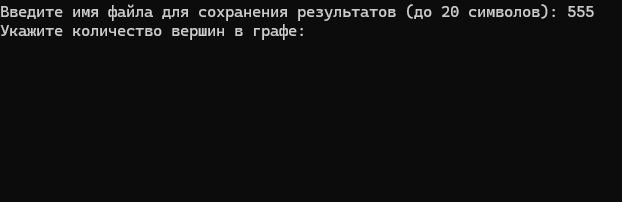


Рисунок 3 - Меню выбора количества вершин в графе

В коде предусмотрен выбор начальной вершины (Рисунок 4)

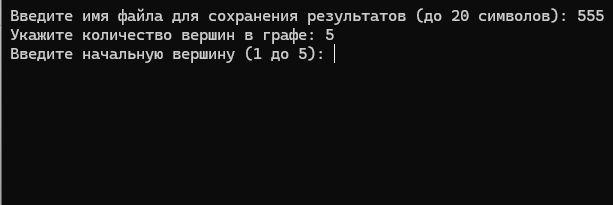


Рисунок 4 - Меню выбора начальной вершины

Можно выбрать между ориентированным и неориентированным графом (Рисунок 5).

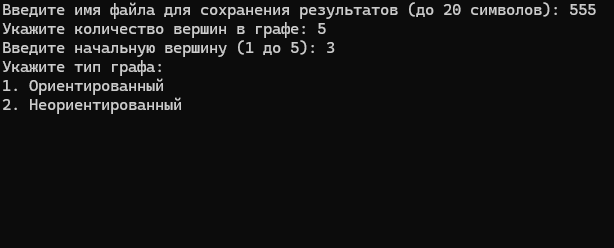


Рисунок 5 - Меню выбора типа графа (ориентированный, неориентированный)

В программе матрица смежности генерируется случайным образом или вручную (Рисунок 6, Рисунок 7, Рисунок 8).

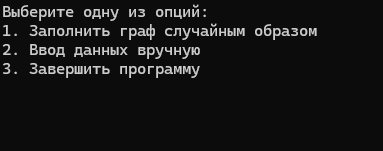


Рисунок 6 - Меню заполнения матрицы

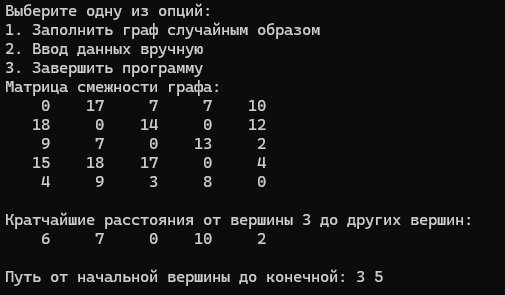


Рисунок 7- Случайное заполнение

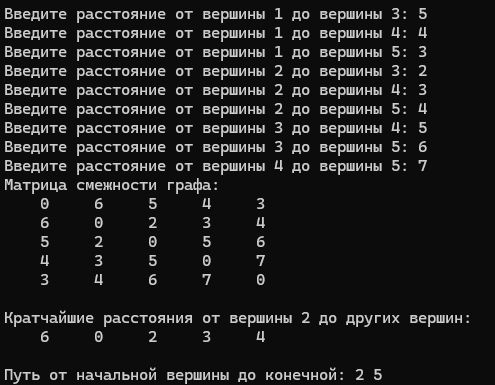


Рисунок 8 - Ручной ввод

Код предоставляет возможность вывести матрицу на экран (Рисунок 9)

void printGraphMatrix(int\*\* graphMatrix, int size, FILE\* file) {

printf("Матрица смежности графа:\n");

fprintf(file, "Матрица смежности графа:\n");

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

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

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

fprintf(file, "%5d ", graphMatrix[i][j]);

}

printf("\n");

fprintf(file, "\n");

}

}

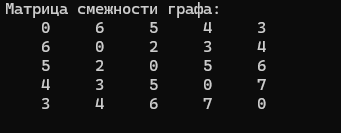


Рисунок 9 - Вывод матрицы

Можно вывести кратчайшие расстояния от заданной вершины до других(Рисунок 10)

void printDistances(int\* distances, int size, int startVertex, FILE\* file) {

printf("\nКратчайшие расстояния от вершины %d до других вершин:\n", startVertex + 1);

fprintf(file, "\nКратчайшие расстояния от вершины %d до других вершин:\n", startVertex + 1);

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

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

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

}

printf("\n");

fprintf(file, "\n");

}



Рисунок 10 - Кратчайшие расстояния

Также можно вывести путь от начальной вершины до конечной(Рисунок 11)

printf("\nПуть от начальной вершины до конечной: ");

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# **5.Тестирование**

Среда разработки Microsoft Visual Studio 2019 предоставляет все необходимые инструменты для разработки и отладки многомодульных программ. Тестирование проводилось как в процессе разработки, так и по завершении написания программы. Во время тестирования были обнаружены и устранены многочисленные ошибки, связанные с вводом данных, чтением файлов и реализацией алгоритма Дейкстры. Ниже приведен результат тестирования программы с вводом количества вершин графа, матрицы смежности, сгенерированной автоматически(Рисунок 13), а также с введенными вручную данными(Рисунок 12).

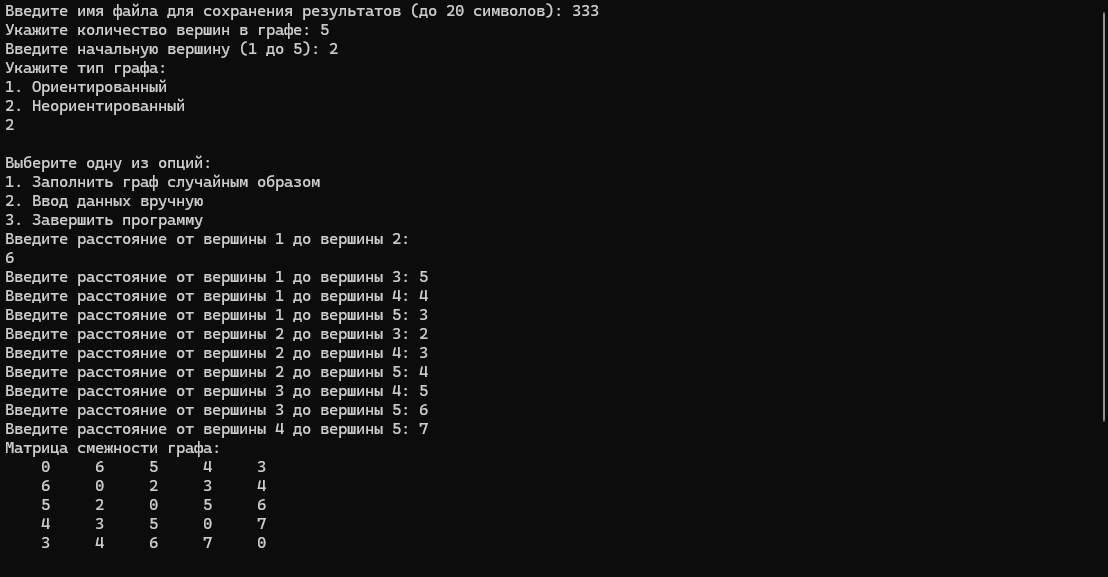


Рисунок 12 - Тестирование при вводе количество вершин в графе = 5, ориентированный граф и ручная генерация матрицы смежности

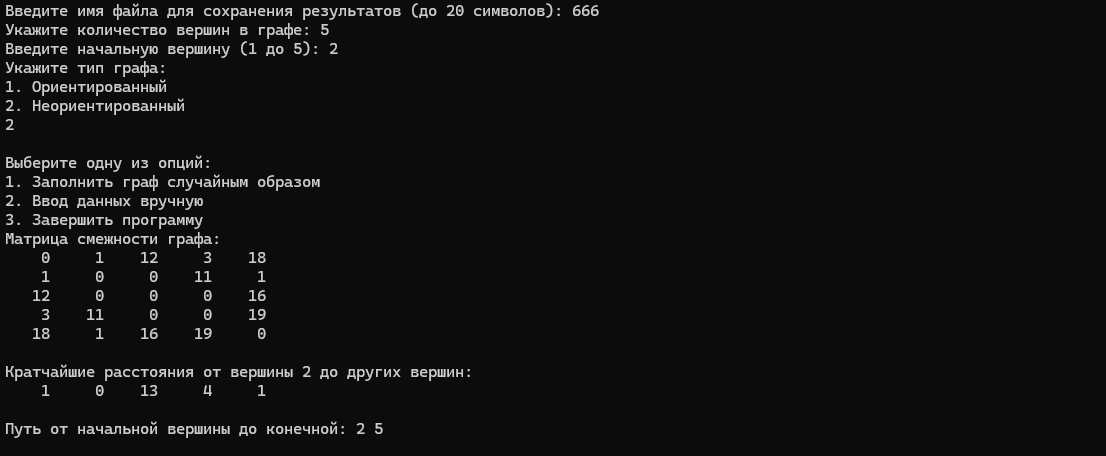


Рисунок 13 - Тестирование при вводе количество вершин в графе = 5, ориентированный граф и случайная генерация матрицы смежности

Таблица 1 – Описание поведения программы при тестировании

| №№ | Описание | Предусловие | Тестирование | Ожидаемый результат |
| --- | --- | --- | --- | --- |
| 1 | Работа меню | Программа запущена | Запускаем программу | Вывод в консоли меню программы |
| 2 | Выбор функции | Программа запущена | Вводим номер функции | Переход к выполнению функции |
| 3 | Определение числа вершин графа | Программа запущена | Ввод числа графа с клавиатуры | Переход к выполнению функции |
| 4 | Создание неориентированного графа | Ввод “1” в меню | Ввод способа создания матрицы | Создание неориентированного графа |
| 5 | Создание ориентированного графа | Ввод “2” в меню | Ввод способа создания матрицы | Создание ориентированного графа |
| 6 | Создание матрицы случайным образом | Ввод “1” в меню | Ввод генерации матрицы случайным образом | Вывод матрицы на экран |
| 7 | Создание матрицы путем ввода с клавиатуры | Ввод “2” в меню | Ввод матрицы ручным способом | Вывод матрицы на экран |
| 8 | Вывод матрицы на экран | Наличие матрицы | Ввод наименования сохранения файла | Вывод матрицы на экран |
| 9 | Сохранение в файл результата | Наличие матрицы | Ввод названия файла сохранения | Появление в файле сохранения результатов |

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

**6. Ручной расчёт программы**

Сделал матрицу смежности с тремя вершинами и заполнил расстояния исходя из матрицы смежности (Рисунок 14), вычислил расстояния от первой вершины.

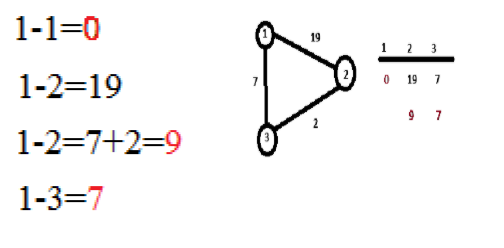


Рисунок 14 – граф из 3 вершин

В программе выбрал 3 вершины и неориентированный граф(Рисунок 15).

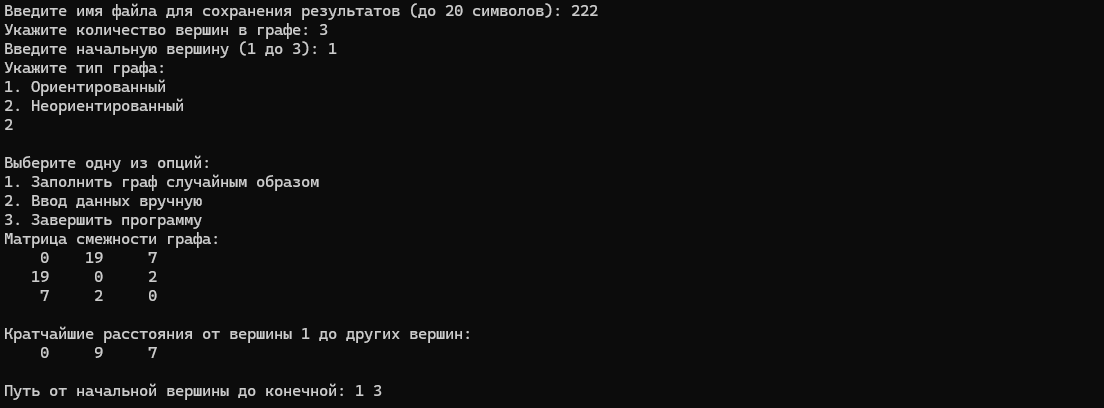


Рисунок 15 - Пример работы с тремя вершинами

Алгоритм сначала присваивает всем вершинам значение, соответствующее практически недостижимому расстоянию. Затем с использованием алгоритма обхода в ширину он сравнивает текущий вес пути с уже присвоенным значением вершины. Если новый путь имеет меньший вес, то вершине присваивается обновленное значение.

Ручной расчет совпал с результатами программы, что подтверждает правильность её работы.

**Заключение**

В ходе разработки данного проекта была создана программа, реализующая алгоритм Дейкстры с использованием Microsoft Visual Studio 2022. В процессе выполнения курсовой работы были приобретены навыки программирования и освоены методы создания матриц смежности, основанные на теории графов. Также были углублены знания в языке программирования C.

Одним из ограничений разработанной программы является её простой пользовательский интерфейс, так как она работает в консольном режиме, что исключает необходимость работы с оконным интерфейсом, увеличивающим сложность программы. Тем не менее, программа обладает достаточным функционалом для выполнения поставленных задач.

# 

# 

# **Список литературы**

1. Алгоритм Дейкстры<https://w.wiki/5BW4>
2. Язык программирования C. С. Прата
3. А.А. Тюгашев. Языки программирования. Учебное пособие. 2018 г.

**Приложение A**

**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>

// Функция для вывода матрицы графа

void printGraphMatrix(int\*\* graphMatrix, int size, FILE\* file) {

printf("Матрица смежности графа:\n");

fprintf(file, "Матрица смежности графа:\n");

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

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

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

fprintf(file, "%5d ", graphMatrix[i][j]);

}

printf("\n");

fprintf(file, "\n");

}

}

// Функция для инициализации массивов

void initializeArrays(int\*\* distances, int\*\* visited, int size, int startVertex) {

\*distances = (int\*)malloc(size \* sizeof(int));

\*visited = (int\*)malloc(size \* sizeof(int));

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

(\*distances)[i] = 10000; // "Бесконечность"

(\*visited)[i] = 1; // Все вершины ещё не посещены

}

(\*distances)[startVertex] = 0; // Расстояние до стартовой вершины

}

// Функция для вывода кратчайших расстояний

void printDistances(int\* distances, int size, int startVertex, FILE\* file) {

printf("\nКратчайшие расстояния от вершины %d до других вершин:\n", startVertex + 1);

fprintf(file, "\nКратчайшие расстояния от вершины %d до других вершин:\n", startVertex + 1);

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

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

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

}

printf("\n");

fprintf(file, "\n");

}

// Функция для восстановления пути

void restorePath(int\* distances, int\*\* graphMatrix, int startVertex, int size, FILE\* file) {

int\* path = (int\*)malloc(size \* sizeof(int));

int endVertex = size - 1;

path[0] = endVertex + 1; // Начинаем с конечной вершины

int pathIndex = 1;

int pathWeight = distances[endVertex];

while (endVertex != startVertex) {

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

if (graphMatrix[i][endVertex] != 0) {

int possibleDist = pathWeight - graphMatrix[i][endVertex];

if (possibleDist == distances[i]) {

pathWeight = possibleDist;

endVertex = i;

path[pathIndex] = i + 1;

pathIndex++;

break;

}

}

}

}

// Вывод пути

printf("\nПуть от начальной вершины до конечной: ");

fprintf(file, "\nПуть от начальной вершины до конечной: ");

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

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

fprintf(file, "%d ", path[i]);

}

printf("\n");

fprintf(file, "\n");

free(path);

}

int main() {

FILE\* file;

int size = 0, minVertex, minDist, inputDistance = -1;

int\*\* graphMatrix;

int\* distances;

int\* visited;

int startVertex = 0;

char filename[20], userInput;

int directed = 0; // Флаг для ориентированности графа

system("chcp 1251");

system("cls");

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

printf("Введите имя файла для сохранения результатов (до 20 символов): ");

gets\_s(filename);

file = fopen(filename, "a");

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

while (size <= 0) {

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

if (size <= 0) {

printf("Некорректное значение. Попробуйте снова: ");

}

}

printf("Введите начальную вершину (1 до %d): ", size);

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

startVertex--;

// Запрашиваем тип графа (ориентированный или нет)

printf("Укажите тип графа:\n1. Ориентированный\n2. Неориентированный\n");

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

if (directed != 1) {

directed = 0; // Если выбран другой вариант, считаем граф неориентированным

}

graphMatrix = (int\*\*)malloc(size \* sizeof(int\*));

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

graphMatrix[i] = (int\*)malloc(size \* sizeof(int));

}

while (1) {

printf("\nВыберите одну из опций:\n1. Заполнить граф случайным образом\n2. Ввод данных вручную\n3. Завершить программу\n");

userInput = \_getch();

if (userInput == '1') {

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

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

if (i == j) {

graphMatrix[i][j] = 0;

}

else {

graphMatrix[i][j] = rand() % 20; // Взвешенный граф

if (!directed) {

graphMatrix[j][i] = graphMatrix[i][j]; // Для неориентированного графа

}

}

}

}

break;

}

if (userInput == '2') {

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

for (int j = i; j < size; j++) {

if (i == j) {

graphMatrix[i][j] = 0;

}

else {

printf("Введите расстояние от вершины %d до вершины %d: ", i + 1, j + 1);

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

while (inputDistance < 0) {

printf("Ошибка! Расстояние не может быть отрицательным. Попробуйте снова: ");

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

}

graphMatrix[i][j] = inputDistance;

if (!directed) {

graphMatrix[j][i] = inputDistance; // Для неориентированного графа

}

}

}

}

break;

}

if (userInput == '3') {

exit(0);

}

}

printGraphMatrix(graphMatrix, size, file);

initializeArrays(&distances, &visited, size, startVertex);

do {

minDist = 10000;

minVertex = 10000;

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

if (visited[i] == 1 && distances[i] < minDist) {

minDist = distances[i];

minVertex = i;

}

}

if (minVertex != 10000) {

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

if (graphMatrix[minVertex][i] > 0) {

inputDistance = minDist + graphMatrix[minVertex][i];

if (inputDistance < distances[i]) {

distances[i] = inputDistance;

}

}

}

visited[minVertex] = 0;

}

} while (minVertex < 10000);

printDistances(distances, size, startVertex, file);

restorePath(distances, graphMatrix, startVertex, size, file);

fclose(file);

free(graphMatrix);

free(visited);

free(distances);

return 0;

}