

**Содержание**

[Реферат 3](#_Toc153478642)

[Введение 4](#_Toc153478643)

[1. Постановка задачи 5](#_Toc153478644)

[2. Теоретическая часть задания 6](#_Toc153478645)

[3. Описание алгоритма программы 7](#_Toc153478646)

[4. Описание программы 10](#_Toc153478647)0

[5. Отладка и тестирование 14](#_Toc153478648)2

[6. Ручной расчет 20](#_Toc153478649)5

[Заключение 23](#_Toc153478652)6

[Список литературы 24](#_Toc153478653)7

[Приложение А. Листинги программы 25](#_Toc153478654)18

# Введение

Алгоритм Дейкстры – это один из фундаментальных алгоритмов теории графов, который находит широкое применение в задачах поиска кратчайшего пути. Основная цель алгоритма – найти минимальный путь от одной выбранной вершины графа до всех остальных, что важно в таких практических областях, как оптимизация маршрутов, планирование сети, навигационные системы и управление транспортными потоками. Алгоритм находит кратчайшие пути последовательно, начиная с заданной стартовой вершины и постепенно добавляя минимально удалённые вершины до тех пор, пока не будут найдены оптимальные пути ко всем вершинам.

Для реализации данного алгоритма была выбрана среда разработки Microsoft Visual Studio 2022, а язык программирования C использован в качестве основного инструмента. Этот выбор обусловлен широким распространением языка C и его применимостью в различных областях программирования.

# Постановка задачи

Необходимо разработать программу для алгоритма Дейктсры.

Программа должна отвечать следующим требованиям:

1. Текстовое или графическое меню для удобства использования программы.

2. Возможность задания пользователем размера графа (множества).

3.Возможность выбора автоматического (случайного) или ручного (с клавиатуры или из файла) задания графа (элементов множества).

4. Возможность сохранения результатов работы программы.

5. Устройство ввода информации: клавиатура/мышь.

# Теоретическая часть задания

Граф — это абстрактный тип данных, предназначенный для реализации концепций неориентированного графа и ориентированного графа из области теории графов в математике.

Структура данных графа состоит из конечного (и, возможно, изменяемого) набора вершин (также называемых узлами или точками) вместе с набором неупорядоченных пар этих вершин для неориентированного графа или набором упорядоченных пар для ориентированного графа. Эти пары известны как ребра (также называемые связями или линиями), а для ориентированного графа также известны как ребра, но также иногда как стрелки или дуги. Вершины могут быть частью структуры графа или могут быть внешними объектами, представленными целочисленными индексами или ссылками.

Алгоритм Дейкстры — это метод нахождения кратчайших путей от одной вершины графа ко всем остальным.

# Описание алгоритма программы

Перед началом разработки программы был проведен анализ требований и определены основные функциональности, которые она должна включать. Основные компоненты программы включают в себя генерацию и отображение графа, обработка его вершин, сохранение графа в файл.

На вход подаётся кол-во вершин для генерации графа. После окончания генерации или копирования из файла, можно выполнить алгоритм Дейкстры или сохранить его в файл.

Была реализована функция dijkstra, которая отвечает за раскраску графа.

**Функция dijkstra:**

void dijkstra(int\*\* graph, int src, int rows) {

int\* dist = (int\*)malloc(rows \* sizeof(int)); // Массив для хранения кратчайших расстояний до вершин

bool\* visited = (bool\*)malloc(rows \* sizeof(bool)); // Массив для отслеживания посещенных вершин

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

dist[i] = INT\_MAX;

visited[i] = false;

}

dist[src] = 0; // Расстояние до начальной вершины равно 0

for (int count = 0; count < rows - 1; count++) {

int u = minDistance(dist, visited, rows); // Находим вершину с минимальным расстоянием

visited[u] = true;

for (int v = 0; v < rows; v++)

if (!visited[v] && graph[u][v] && dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printf("Кратчайшие расстояния от вершины %d:\n", src + 1);

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

printf("Вершина %d -> Расстояние %d\n", i + 1, dist[i]);

free(dist);

free(visited);

}

Ниже представлен псевдокод функции dijkstra:

1. void dijkstra(int\*\* graph, int src, int rows):

a. Создать массив dist для хранения кратчайших расстояний до каждой вершины:

I. для i от 0 до rows:

1. dist[i] = INT\_MAX

2. конец цикла

b. Создать массив visited для отслеживания посещённых вершин:

I. для i от 0 до rows:

1. visited[i] = false

2. конец цикла

c. Установить начальную точку:

I. dist[src] = 0

d. для count от 0 до rows - 1:

I. u = minDistance(dist, visited, rows) // Найти вершину с минимальным расстоянием

II. visited[u] = true

III. для каждой вершины v от 0 до rows:

1. если !visited[v] и graph[u][v] != 0 и dist[u] != INT\_MAX:

a. если dist[u] + graph[u][v] < dist[v]:

i. dist[v] = dist[u] + graph[u][v]

b. конец условия

2. конец условия

IV. конец внутреннего цикла

e. Вывести резу 1. void dijkstra(int\*\* graph, int src, int rows):

a. Создать массив dist для хранения кратчайших расстояний до каждой вершины:

I. для i от 0 до rows:

1. dist[i] = INT\_MAX

2. конец цикла

b. Создать массив visited для отслеживания посещённых вершин:

I. для i от 0 до rows:

1. visited[i] = false

2. конец цикла

c. Установить начальную точку:

I. dist[src] = 0

d. для count от 0 до rows - 1:

I. u = minDistance(dist, visited, rows) // Найти вершину с минимальным расстоянием

II. visited[u] = true

III. для каждой вершины v от 0 до rows:

1. если !visited[v] и graph[u][v] != 0 и dist[u] != INT\_MAX:

a. если dist[u] + graph[u][v] < dist[v]:

i. dist[v] = dist[u] + graph[u][v]

b. конец условия

2. конец условия

IV. конец внутреннего цикла

e. Вывести результаты:

I. для каждой вершины i от 0 до rows:

1. Напечатать dist[i]

2. конец цикла

f. Освободить память, выделенную для dist и visited

2. Конец функциильтаты:

I. для каждой вершины i от 0 до rows:

1. Напечатать dist[i]

2. конец цикла

f. Освободить память, выделенную для dist и visited

2. Конец функции

# Описание программы

Программа состоит из нескольких модулей (многомодульная): Source.cpp, где реализуется вся программа, Header.h – заголовочный файл, Functions.cpp – файл с функциями.

Разработанная программа состоит из нескольких функций модуля, которые реализовывались в следующем порядке:

1. main()– Меню программы.
2. Header.h – библиотека с константами.
3. addSp()– функция для добавления вершины в список смежности.
4. printSp() – функция для вывода на экран списка смежности.
5. inputNumbers() – функция для проверки ввода чисел пользователем.
6. dijkstra () – функция для алгоритма Дейктсры.

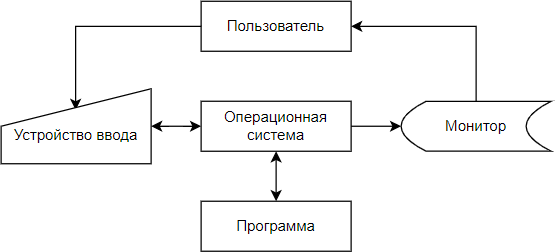
По порядку реализации модулей можно понять, что использовался нисходящей подход реализации программы. Этот метод основан на том, что 5 функций сначала проектируются основные компоненты программы, а затем уже дорабатываются ее мелкие детали. Это позволяет упростить ее тестирование, так как каждая функция (модуль) тестируется сразу же после его реализации, что помогает в создании более структурированных программ.

### При запуске программы происходит вывод заставки-приветствия, после чего пользователю необходимо выбрать пункт меню для дальнейшего использования программы. Выход из меню осуществляется после выбора пользователем нужного пункта. Описание состояний программы выполнено в таблице ниже.

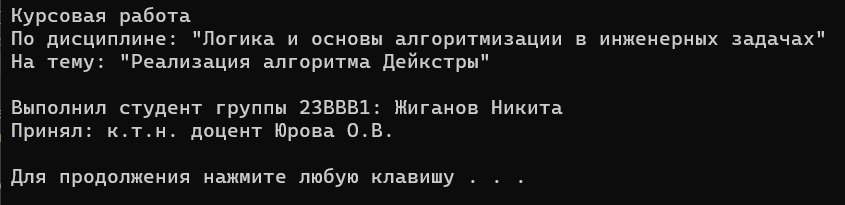
#### Таблица 1 – Главная функция

|  |  |  |
| --- | --- | --- |
| Клавиши,  вызывающие событие | Действие  пользователя | Действие программы |
| 1, Enter | Выбран пункт  «Создать случайный граф» | Запускается диалог создания случайного графа |
| 2, Enter | Выбран пункт  «Ввести граф в ручную» | Запускается диалог  Создания графа |
| 3, Enter | Выбран пункт  «Импортировать граф из файла» | Запускает диалог импортирования графа из файла |
| 4, Enter | Выбран пункт  «Сохранить полученные данные» | Запускается диалог сохранения полученных данных в файл |
| 5, Enter | Выбран пункт  «Выполнить алгоритм Дейкстры» | Запускается алгоритм Дейкстры |
| 6, Enter | Выбран пункт  « Просмотреть граф» | Выводится матрица смежности и список смежности графа, если он существует |
| 7, Enter | Выбран пункт  «Выход» | Программа завершает свою работу |

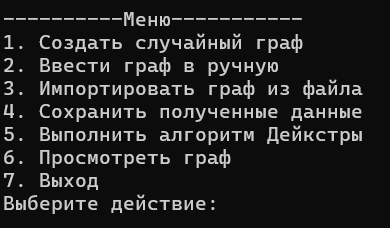
### Далее на рисунке 1 представлена схема данных.



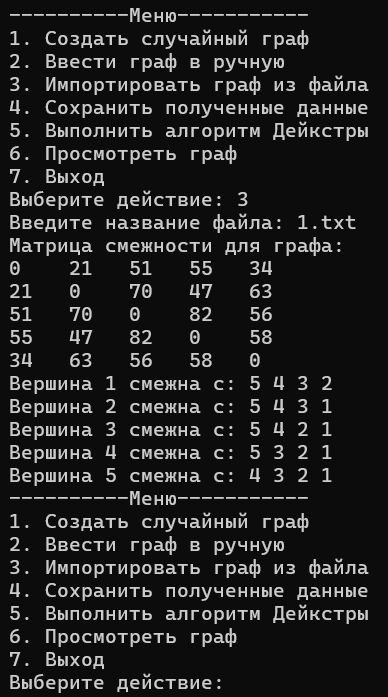
***Рисунок 1 - Схема данных***



При запуске программы открывается консольное окно с меню:



**Рисунок 2 – Главное меню**

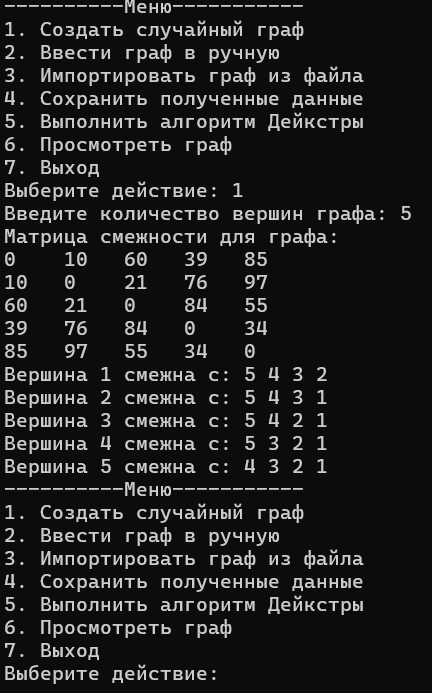
После выбора способа создания графа, выведется запрос на кол-во вершин в графе и после ввода кол-ва выведется матрица смежности для графа и список смежности. Либо(в случае копирования) сразу выведется матрица смежности для графа и список смежности:

**Рисунок 3 – Создание графа**

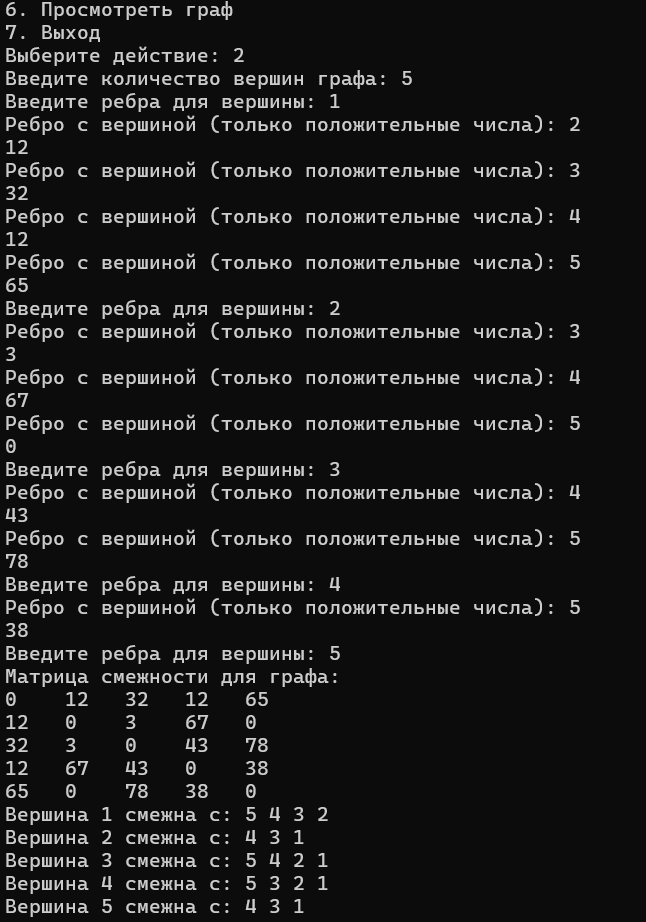
# Отладка и тестирование

В качестве среды разработки была выбрана программа Microsoft Visual Studio 2022. Программа обладает всеми средствами необходимыми при разработке и отладке программы. Для отладки использовались несколько возможностей Visual Studio: точка останова, трассировка, анализ содержимого переменных.

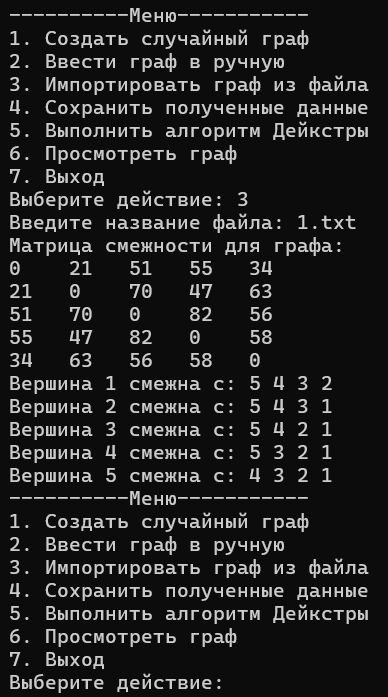
Тестирование проводилось во время разработки и также после завершения разработки. В ходе нее было выявлено огромное количество проблем, связанных с работой с файлами, работой с памятью, работой с полями структуры, с размерами самих массивов и т. д.

Тестирование программы:

**Рисунок 4 – создание графа случайным образом**

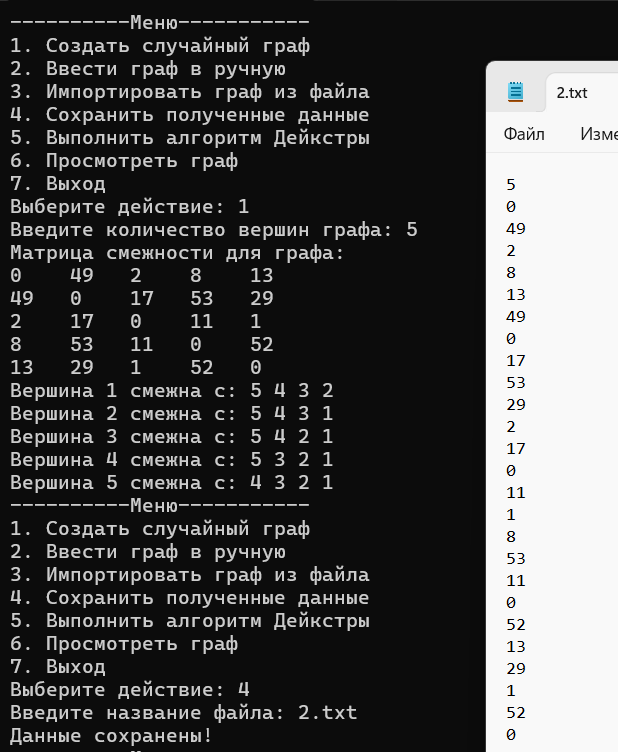


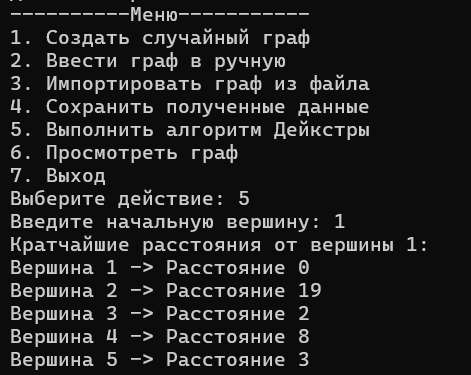
**Рисунок 5 – создание графа вручную**



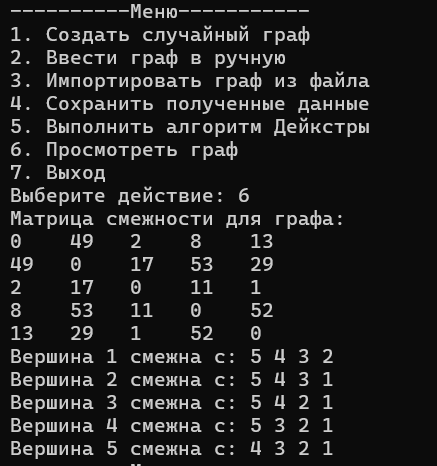


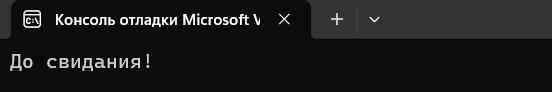
**Рисунок 6 – копирование графа из файла**



**Рисунок 7 – сохранение графа в файл**

**Рисунок 8 – алгоритм Дейктсры**



**Рисунок 9 – просмотр графа**

**Рисунок 10 – выход из программы**

***Таблица 1 – Описание поведения программы при тестировании***

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Ввод кол-ва вершин | Правильное считывание размера, генерация графа нужного размера и вывод графа | Верно |
| Алгоритм Дейкстры | Правильное нахождение кратчайших расстояний. | Верно |
| Сохранение в файл/чтение из файла | Граф верно считывается из файла и верно записывается результат. | Верно |

# Ручной расчет

Обозначим вершины как 1, 2, 3, 4 и 5. В ходе выполнения будем обновлять список расстояний и отмечать посещенные вершины.

**Инициализация:**

* Начальная вершина: 1
* Расстояния до вершин: [0,∞,∞,∞,∞][0, \infty, \infty, \infty, \infty][0,∞,∞,∞,∞]
* Посещенные вершины: ∅\emptyset∅

**Шаг 1:**

* Текущая вершина: 1, расстояние до неё: 0
* Обновляем расстояния до соседей:
  + До вершины 2: 0+49=490 + 49 = 490+49=49
  + До вершины 3: 0+2=20 + 2 = 20+2=2
  + До вершины 4: 0+8=80 + 8 = 80+8=8
  + До вершины 5: 0+13=130 + 13 = 130+13=13
* Обновленные расстояния: [0,49,2,8,13][0, 49, 2, 8, 13][0,49,2,8,13]
* Отмечаем вершину 1 как посещенную: {1}\{1\}{1}

**Шаг 2:**

* Выбираем вершину с наименьшим расстоянием среди непосещенных вершин — это вершина 3 (расстояние 2).
* Обновляем расстояния до её соседей:
  + До вершины 1 уже минимальное расстояние.
  + До вершины 2: 2+17=192 + 17 = 192+17=19 (меньше, чем 49, поэтому обновляем)
  + До вершины 4: 2+11=132 + 11 = 132+11=13 (меньше, чем 8, поэтому не обновляем)
  + До вершины 5: 2+1=32 + 1 = 32+1=3 (меньше, чем 13, поэтому обновляем)
* Обновленные расстояния: [0,19,2,8,3][0, 19, 2, 8, 3][0,19,2,8,3]
* Отмечаем вершину 3 как посещенную: {1,3}\{1, 3\}{1,3}

**Шаг 3:**

* Выбираем вершину с наименьшим расстоянием среди непосещенных вершин — это вершина 5 (расстояние 3).
* Обновляем расстояния до её соседей:
  + До вершины 1 уже минимальное расстояние.
  + До вершины 2: 3+29=323 + 29 = 323+29=32 (больше, чем 19, поэтому не обновляем)
  + До вершины 3 уже минимальное расстояние.
  + До вершины 4: 3+52=553 + 52 = 553+52=55 (больше, чем 8, поэтому не обновляем)
* Обновленные расстояния: [0,19,2,8,3][0, 19, 2, 8, 3][0,19,2,8,3]
* Отмечаем вершину 5 как посещенную: {1,3,5}\{1, 3, 5\}{1,3,5}

**Шаг 4:**

* Выбираем вершину с наименьшим расстоянием среди непосещенных вершин — это вершина 4 (расстояние 8).
* Обновляем расстояния до её соседей:
  + До вершины 1 уже минимальное расстояние.
  + До вершины 2: 8+53=618 + 53 = 618+53=61 (больше, чем 19, поэтому не обновляем)
  + До вершины 3 уже минимальное расстояние.
  + До вершины 5 уже минимальное расстояние.
* Обновленные расстояния: [0,19,2,8,3][0, 19, 2, 8, 3][0,19,2,8,3]
* Отмечаем вершину 4 как посещенную: {1,3,5,4}\{1, 3, 5, 4\}{1,3,5,4}

**Шаг 5:**

* Последняя непосещенная вершина — 2 (расстояние 19).
* Так как это последняя вершина, дополнительные обновления не требуются.
* Обновленные расстояния: [0,19,2,8,3][0, 19, 2, 8, 3][0,19,2,8,3]
* Отмечаем вершину 2 как посещенную: {1,2,3,4,5}\{1, 2, 3, 4, 5\}{1,2,3,4,5}

**Итоговые кратчайшие расстояния от вершины 1 до всех вершин:**

* Вершина 1: 0
* Вершина 2: 19
* Вершина 3: 2
* Вершина 4: 8
* Вершина 5: 3

Таким образом, кратчайшие расстояния от вершины 1 до всех остальных вершин: [0,19,2,8,3][0, 19, 2, 8, 3][0,19,2,8,3].

**Рисунок 11 – ручной расчет**

# 

**Рисунок 12 – расчет программы**

# Заключение

При выполнении данной курсовой работы были получены навыки разработки многомодульных программ. Была освоена работа с файлами, изучены функции работы с консолью и оперативной памятью. Также были получены основные навыки отладки и тестирования программ и программирования в среде Visual Studio 2022 на языке Си.

В рамках курсовой работы была написана программа для реализации алгоритма Дейкстры. Она имеет все необходимые функции для работы с графом.

Важной особенностью программы является реализация текстового пользовательского интерфейса, что обеспечивает удобство использования и повышает интуитивность взаимодействия пользователя с программой.

Код программы структурирован, что улучшает его читаемость и возможность доработки.

# В дальнейшем программу можно улучшить, путём добавления графического интерфейса, что позволит улучшить навигацию по меню программы, путём добавления возможности использования мыши.

# Список литературы

### Э. Таненбаум, Т. Остин. Архитектура компьютера (6-е издание). 2016 г.

### Шилдт Герберт. Полный справочник по С. 2002 г.

### MSDN.

### Дэвид Гриффитс, Дон Гриффитс. Изучаем программирование на С. 2013 г.

### Брайан Керниган. Деннис Ритчи. Язык программирования С. 2009 г.

1. Лекция, Деревья\_потоки\_раскрашивания, Митрохин М.А.

# Приложение А. Листинги программы

**Source.cpp**

#include "Header.h"

void main() {

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

int\*\* arr = 0; // матрица смежности

int\* colors = 0; // массив для цветов

int i, j, rows;

FILE\* file;

Node\*\* arr\_sp = NULL;

char name[255];

int choice, rows1, rows2, what, startVertex;

printf("Курсовая работа\nПо дисциплине: \"Логика и основы алгоритмизации в инженерных задачах\"\nНа тему: \"Реализация алгоритма Дейкстры\"\n\nВыполнил студент группы 23ВВВ1: Жиганов Никита\nПринял: к.т.н. доцент Юрова О.В.\n\n");

system("PAUSE");

system("cls");

do {

printf("----------Меню-----------\n");

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

printf("2. Ввести граф в ручную\n");

printf("3. Импортировать граф из файла\n");

printf("4. Сохранить полученные данные\n");

printf("5. Выполнить алгоритм Дейкстры\n");

printf("6. Просмотреть граф\n");

printf("7. Выход\n");

printf("Выберите действие: ");

choice = inputNumber();

switch (choice) {

case 1: // Создать случайный граф

printf("Введите количество вершин графа: ");

rows = inputNumber();

arr = (int\*\*)malloc(rows \* sizeof(int\*));

if (arr == NULL) {

printf("Не удалось выделить память!\n");

break;;

}

for (i = 0; i < rows; i++) {

arr[i] = (int\*)malloc(rows \* sizeof(int));

}

// генерировать случайные значения для матрицы смежности

for (i = 0; i < rows; i++) {

for (j = i; j < rows; j++) {

if (i == j) {

arr[i][j] = 0; // на главной диагонали нули

}

else {

arr[i][j] = rand() % 100; //(-10) + rand() % 21; // случайные значения 0 или 1

arr[j][i] = arr[i][j]; // симметрично заполнять значения для неориентированного графа

}

}

}

// выводить матрицу смежности на экран

printf("Матрица смежности для графа:\n");

for (i = 0; i < rows; i++) {

for (j = 0; j < rows; j++) {

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

}

printf("\n");

}

// Выделение памяти для массива списков смежности

arr\_sp = (Node\*\*)malloc(rows \* sizeof(Node\*));

if (arr\_sp == NULL) {

printf("Не удалось выделить память!\n");

return;

}

for (i = 0; i < rows; i++) {

arr\_sp[i] = NULL;

}

addSp(arr\_sp, arr, rows);

printSp(arr\_sp, rows);

break;

case 2: // Ввести граф в ручную

printf("Введите количество вершин графа: ");

rows = inputNumber();

arr = (int\*\*)malloc(rows \* sizeof(int\*));

if (arr == NULL) {

printf("Не удалось выделить память!\n");

break;

}

for (i = 0; i < rows; i++) {

arr[i] = (int\*)malloc(rows \* sizeof(int));

}

for (i = 0; i < rows; i++) {

printf("Введите ребра для вершины: %d\n", i + 1);

for (j = i; j < rows; j++) {

if (i == j) {

arr[i][j] = 0;

}

else {

printf("Ребро с вершиной (только положительные числа): %d\n", j + 1);

arr[i][j] = inputNumber();

if (arr[i][j] < 0) {

printf("Алгоритм Дейкстры работает только с положительными числами!");

break;

}

arr[j][i] = arr[i][j];

}

}

}

// выводить матрицу смежности на экран

printf("Матрица смежности для графа:\n");

for (i = 0; i < rows; i++) {

for (j = 0; j < rows; j++) {

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

}

printf("\n");

}

// Выделение памяти для массива списков смежности

arr\_sp = (Node\*\*)malloc(rows \* sizeof(Node\*));

if (arr\_sp == NULL) {

printf("Не удалось выделить память!\n");

return;

}

for (i = 0; i < rows; i++) {

arr\_sp[i] = NULL;

}

addSp(arr\_sp, arr, rows);

printSp(arr\_sp, rows);

break;

case 3: // Импротировать граф из файла

printf("Введите название файла: ");

scanf("%s", &name);

if ((file = fopen(name, "r")) == NULL)

{

printf("Не удалось открыть файл");

getchar();

break;

}

fclose(file);

file = fopen(name, "r");

fscanf(file, "%d", &rows);

arr = (int\*\*)malloc(rows \* sizeof(int\*));

if (arr == NULL) {

printf("Не удалось выделить память!\n");

break;

}

for (i = 0; i < rows; i++) {

arr[i] = (int\*)malloc(rows \* sizeof(int));

}

for (i = 0; i < rows; i++) {

for (j = 0; j < rows; j++) {

fscanf(file, "%d", &arr[i][j]);

}

}

fclose(file);

// выводить матрицу смежности на экран

printf("Матрица смежности для графа:\n");

for (i = 0; i < rows; i++) {

for (j = 0; j < rows; j++) {

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

}

printf("\n");

}

// Выделение памяти для массива списков смежности

arr\_sp = (Node\*\*)malloc(rows \* sizeof(Node\*));

if (arr\_sp == NULL) {

printf("Не удалось выделить память!\n");

return;

}

for (i = 0; i < rows; i++) {

arr\_sp[i] = NULL;

}

addSp(arr\_sp, arr, rows);

printSp(arr\_sp, rows);

break;

case 4: // Сохранить полученные данные

if (arr == NULL) {

printf("Сначала создайте граф!\n");

break;

}

printf("Введите название файла: ");

scanf("%s", &name);

if ((file = fopen(name, "w")) == NULL)

{

printf("Не удалось открыть файл");

getchar();

break;

}

fclose(file);

file = fopen(name, "w");

fprintf(file, "%d\n", rows);

for (i = 0; i < rows; i++) {

for (j = 0; j < rows; j++) {

fprintf(file, "%d\n", arr[i][j]);

}

}

fclose(file);

if (colors == NULL) {

printf("Данные сохранены!\n");

break;

}

file = fopen(name, "a");

fprintf(file, "Результат раскраски:\n");

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

fprintf(file, "Вершина %d -> Цвет %d\n", i + 1, colors[i]);

}

fclose(file);

printf("Данные сохранены!\n");

break;

case 5:

if (arr == NULL) {

printf("Сначала создайте граф!\n");

break;

}

// Выполнить алгоритм Дейкстры

if (arr == NULL) {

printf("Сначала создайте граф!\n");

break;

}

printf("Введите начальную вершину: ");

startVertex = inputNumber() - 1;

if (startVertex < 0 || startVertex >= rows) {

printf("Некорректный номер вершины!\n");

break;

}

dijkstra(arr, startVertex, rows);

break;

case 6: // Просмотреть массив

if (arr == NULL) {

printf("Сначала создайте граф!\n");

break;

}

// выводить матрицу смежности на экран

printf("Матрица смежности для графа:\n");

for (i = 0; i < rows; i++) {

for (j = 0; j < rows; j++) {

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

}

printf("\n");

}

printSp(arr\_sp, rows);

break;

case 7: // Выход

system("cls");

printf("До свидания!\n");

break;

default:

printf("Некорректный выбор!\n");

break;

}

} while (choice != 7);

}

**Header.h**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <time.h>

#include <string.h>

#include <ctype.h>

#include <limits.h>

typedef struct Node {

int inf;

struct Node\* next;

}Node;

int inputNumber();

void colorGraph(int\*\* arr, int rows, int\* colors);

void addSp(Node\*\* arr\_sp, int\*\* arr, int rows);

void printSp(Node\*\* arr\_sp, int rows);

void dijkstra(int\*\* graph, int src, int rows);

**Functions.cpp**

#include "Header.h"

void addSp(Node\*\* arr\_sp, int\*\* arr, int rows) {

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

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

if (arr[i][j] != 0) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->inf = j + 1;

newNode->next = arr\_sp[i];

arr\_sp[i] = newNode;

}

}

}

}

void printSp(Node\*\* arr\_sp, int rows) {

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

Node\* current = arr\_sp[i];

if (current == NULL) {

}

else {

printf("Вершина %d смежна с: ", i + 1);

while (current != NULL) {

printf("%d ", current->inf);

current = current->next;

}

printf("\n");

}

}

}

// Функция для ввода цифр с проверкой

int inputNumber() {

char input[255];

int number;

int isValidInput = 0;

do {

isValidInput = 1; // Предполагаем, что ввод корректен

scanf("%s", input);

// Проверяем каждый символ в строке

for (int i = 0; input[i] != '\0'; i++) {

if (!isdigit(input[i])) {

isValidInput = 0; // Некорректный ввод, если символ не цифра

printf("Ошибка! Введите только цифры.\n");

printf("Выберите действие: ");

break;

}

}

// Очищаем буфер

while (getchar() != '\n');

} while (!isValidInput);

// Преобразуем строку в число

sscanf(input, "%d", &number);

return number;

}

int minDistance(int\* dist, bool\* visited, int rows) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < rows; v++)

if (!visited[v] && dist[v] <= min) {

min = dist[v];

min\_index = v;

}

return min\_index;

}

void dijkstra(int\*\* graph, int src, int rows) {

int\* dist = (int\*)malloc(rows \* sizeof(int)); // Массив для хранения кратчайших расстояний до вершин

bool\* visited = (bool\*)malloc(rows \* sizeof(bool)); // Массив для отслеживания посещенных вершин

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

dist[i] = INT\_MAX;

visited[i] = false;

}

dist[src] = 0; // Расстояние до начальной вершины равно 0

for (int count = 0; count < rows - 1; count++) {

int u = minDistance(dist, visited, rows); // Находим вершину с минимальным расстоянием

visited[u] = true;

for (int v = 0; v < rows; v++)

if (!visited[v] && graph[u][v] && dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printf("Кратчайшие расстояния от вершины %d:\n", src + 1);

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

printf("Вершина %d -> Расстояние %d\n", i + 1, dist[i]);

free(dist);

free(visited);

}