Министерство науки и высшего 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 Флойда»

Выполнил: студент группы 22ВВВ1

Коннов А.Д.

Приняли:

Акифьев И.В.

Юрова О.В.

Пенза – 2023

**Содержание**

[Реферат 4](#_Toc154156949)

[Введение 5](#_Toc154156950)

[1. Постановка задачи 6](#_Toc154156951)

[2. Описание алгоритма программы 7](#_Toc154156952)

[3. Описание программы 9](#_Toc154156953)

[4. Тестирование 16](#_Toc154156954)

[5. Ручной расчёт задачи 20](#_Toc154156955)

[Заключение 23](#_Toc154156956)

[Список литературы 24](#_Toc154156957)

[Листинг программы. 25](#_Toc154156958)

# Реферат

Отчёт 29 стр., 10 рисунков.

АЛГОРИТМ ФЛОЙДА ДЛЯ НАХОЖДЕНИЯ КРАТЧАЙШИХ ПУТЕЙ МЕЖДУ ВСЕМИ ПАРАМИ ВЕРШИН В ОРИЕНТИРОВАННОМ ГРАФЕ.

Цель исследования – разработка программы, способной находить кратчайшие пути между всеми парами вершин в ориентированном графе.

В работе рассмотрены правила поиска кратчайших путей, на основе которых выявляются минимальные расстояния между всеми парами вершин. Установлено, что с помощью данного алгоритма можно эффективно выявить минимальные расстояния между всеми парами вершин в ориентированном графе.

# Введение

Алгоритм Флойда, также известный как алгоритм Флойда-Уоршелла, представляет собой эффективный метод решения задачи нахождения кратчайших путей между всеми парами вершин в ориентированном графе.

Основной идеей алгоритма является пошаговое улучшение длин всех возможных путей между парами вершин, используя промежуточные вершины. В ходе итераций алгоритма находятся все кратчайшие пути, что делает его эффективным для решения задачи даже в графах с отрицательными весами рёбер.

В качестве среды разработки мною была выбрана среда

Microsoft Visual Studio 2022, язык программирования – Си.

Целью данной курсовой работы является разработка программы на языке Си, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Флойда, осуществляющий нахождение длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе.

# 1. Постановка задачи

Требуется разработать программу, которая осуществляет поиск кратчайших путей между всеми парами вершин во взвешенном ориентированном графе.

Исходный граф в программе должен задаваться матрицей смежности, причѐм при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

# 2. Описание алгоритма программы

Для программной реализации алгоритма понадобиться двумерный массив: graph[i][j], который содержит веса рёбер между вершинами графа. После этого создаётся матрица dist[i][j], которая является копией матрицы смежности graph[i][j]. Переменная aопределяет размер матрицы. После создания матрицы происходит перебор всех вершин и обновление кратчайших путей: k–вершина, через которую происходит релаксация путей между iи j. Если путь от iдо jчерез вершину kкороче, чем текущий путь от iдо j, то обновляем dist[i][j]. В результате выводится матрица расстояний между всеми парами вершин.

Ниже представлен псевдокод функции floyd().

Функция floyd(a, graph):

dist = выделить\_память(a строк, каждая строка содержит a столбцов)

Для каждого i от 0 до a:

Для каждого j от 0 до a:

dist[i][j] = graph[i][j]

Для каждого k от 0 до a:

Для каждого i от 0 до a:

Для каждого j от 0 до a:

Если (dist[i][k] + dist[k][j]) меньшеdist[i][j]:

dist[i][j] = dist[i][k] + dist[k][j]

Вывести "\nМатрица расстояний:"

Вывести(" ", каждый i от 0 до a, с шириной столбца -4)

Вывести новую строку

Для каждого i от 0 до a:

Вывести(i + 1, с шириной столбца -4)

Для каждого j от 0 до a:

Еслиdist[i][j] равно INF:

Вывести "INF "

Иначе:

Вывести dist[i][j], с шириной столбца -4

Вывести новую строку

Для каждого i от 0 до a:

Освободить память, выделенную под dist[i]

Освободить память, выделенную под dist

# 3. Описание программы

Для написания данной программы использован язык программирования Си. Язык программирования Си - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Проект был создан в виде консольного приложения Win32 (Visual C++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: printMatrix, generateRandomGraph, manualInputGraph, fileInputGraph, floyd, saveToFile, main.

Работа программы начинается с запроса генерации матрицы: ввод в консоли или из файла. (рис. 1).

while (1) {

printf("Выберите действие:\n");

printf("1. Ввести матрицу смежности\n");

printf("2. Загрузить матрицу смежности из файла\n");

printf("0. Завершить программу\n");

int choice;

int fillChoice;

do {

printf("Ваш выбор: ");

if (scanf("%d", &choice) != 1) {

printf("Ошибка ввода. Пожалуйста, введите корректные данные.\n");

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

choice = -1;

}

} while (choice < 0);

switch (choice) {

case 1:

do {

printf("Введите размер матрицы: ");

if (scanf("%d", &a) != 1 || a <= 0) {

printf("Ошибка ввода. Пожалуйста, введите корректные данные.\n");

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

a = -1;

}

} while (a < 0);

// Выделение памяти под матрицу

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

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

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

}

do {

printf("Выберите способ заполнения матрицы:\n");

printf("1. Заполнить случайными значениями\n");

printf("2. Заполнить вручную\n");

printf("Ваш выбор: ");

if (scanf("%d", &fillChoice) != 1 || (fillChoice != 1 && fillChoice != 2)) {

printf("Ошибка ввода. Пожалуйста, введите корректные данные.\n");

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

fillChoice = -1;

}

} while (fillChoice < 0);

switch (fillChoice) {

case 1:

generateRandomGraph(a, graph);

break;

case 2:

manualInputGraph(a, graph);

break;

default:

printf("Ошибка: неверный выбор заполнения матрицы.\n");

return 1;

}

// Вывод матрицы

printf("Матрица смежности:\n");

printMatrix(a, graph);

Если пользователь выбрал ввод в консоли, то далее предлагается способ заполнения графа: случайным образом или полностью вручную. После этого предлагается ввести размер матрицы. В зависимости от выбора пользователя, выводится случайно сгенерированная матрица смежности, либо введённая пользователем вручную (рис.2 и рис. 3).

srand(time(NULL));

// Заполнение матрицы случайными весами, 0 по диагонали

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

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

if (i == j) {

graph[i][j] = 0;

}

else {

graph[i][j] = rand() % 9 + 1;

}

}

}

}

printf("Введите матрицу смежности(%dx%d):\n", a, a);

printf("Введите матрицу смежности(%dx%d):\n", a, a);

// Ввод элементов матрицы с проверкой на корректность

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

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

while (1) {

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

if (scanf("%d", &graph[i][j]) == 1 && graph[i][j] <= BIG) {

break;

}

else {

printf("Ошибка ввода. Пожалуйста, введите корректные данные (не больше %d).\n", BIG);

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

}

}

}

}

}

Если же пользователь отказался от ввода графа в консоли в пользу ввода её из файла, то ему необходимо указать название файла, из которого хочет ввести матрицу. (рис. 4).

FILE\* file;

char filename[256];

do {

printf("Введите имя файла: ");

if (scanf("%s", filename) != 1) {

printf("Ошибка ввода. Пожалуйста, введите корректные данные.\n");

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

filename[0] = '\0';

}

} while (filename[0] == '\0');

file = fopen(filename, "r");

while (file == NULL) {

printf("Ошибка открытия файла. Пожалуйста, введите корректное имя файла: ");

if (scanf("%s", filename) != 1) {

printf("Ошибка ввода. Пожалуйста, введите корректные данные.\n");

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

filename[0] = '\0';

}

file = fopen(filename, "r");

}

// Выделение памяти под матрицу

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

for (int i = 0; i < \*a; i++) {

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

// Чтение элементов матрицы из файла

for (int j = 0; j < \*a; j++) {

if (fscanf(file, "%d", &(\*graph)[i][j]) != 1) {

printf("Ошибка чтения элементов матрицы из файла.\n");

exit(1);

}

// Пропустить разделители (пробелы, табуляции)

int delimiter;

do {

delimiter = fgetc(file);

} while (delimiter == ' ' || delimiter == '\t');

// Вернуть разделитель в поток

ungetc(delimiter, file);

}

}

// Закрытие файла

fclose(file);

После каждого вывода итоговой матрицы расстояний пользователю предлагается сохранить результат работы программы в файл (рис. 5).

void saveToFile(int a, int\*\* graph) {

FILE\* file;

char filename[256];

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

scanf("%s", filename);

// Открытие файла для записи

file = fopen(filename, "w");

if (file == NULL) {

printf("Ошибка открытия файла для записи.\n");

exit(1);

}

// Сначала сохраняем матрицу смежности

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

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

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

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

// Вставляем пробелы между элементами матрицы

if (j < a - 1) {

fprintf(file, " ");

}

}

fprintf(file, "\n");

}

// Затем сохраняем матрицу расстояний (просто копируем код вывода матрицы расстояний)

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

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

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

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

dist[i][j] = graph[i][j];

}

}

fprintf(file, "\nМатрица расстояний:\n");

fprintf(file, " ");

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

fprintf(file, "%-4d", i + 1);

}

fprintf(file, "\n");

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

fprintf(file, "%-4d", i + 1);

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

fprintf(file, "%-4d", dist[i][j]);

}

fprintf(file, "\n");

}

// Освобождение памяти

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

free(dist[i]);

}

free(dist);

// Закрытие файла

fclose(file);

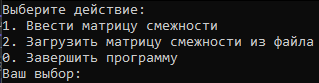


Рисунок 1 – Начальный выбор пользователя

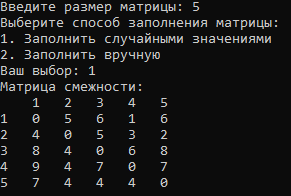


Рисунок 2 – Случайная генерация матрицы

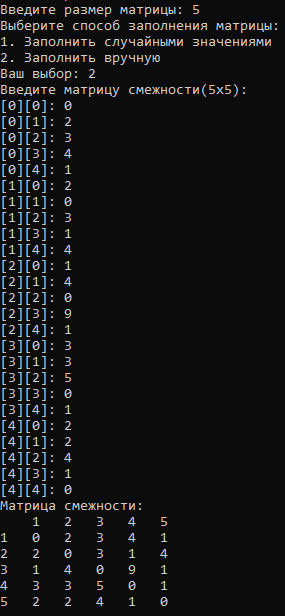


Рисунок 3 – Ввод элементов матрицы вручную

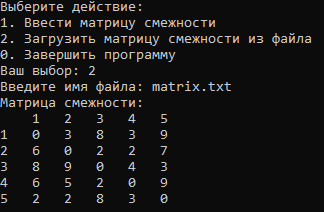


Рисунок 4 – Ввод матрицы из файла

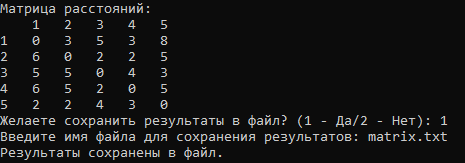


Рисунок 5 – Сохранение результата в файл

# 4. Тестирование

Среда разработки Microsoft Visual Studio 2022 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных параметров ввода матрицы смежности.

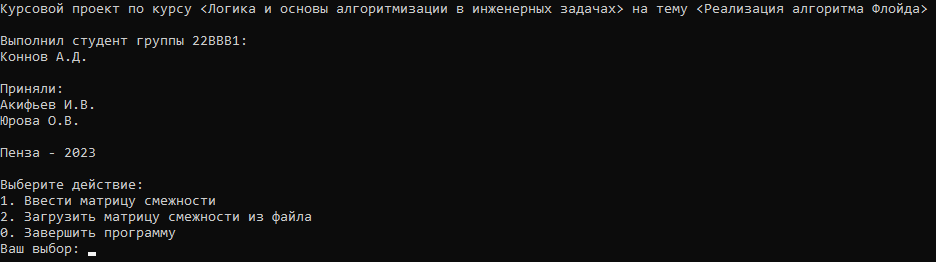


Рисунок 6 – Тестирование запуска программы

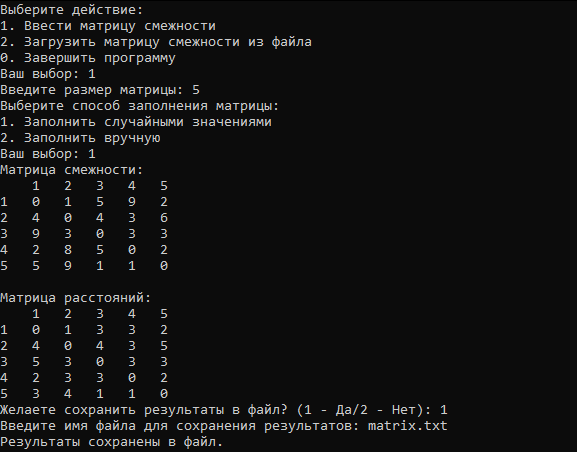


Рисунок 7 – Тестирование генерации матрицы смежности случайным образом и сохранение результата в файл

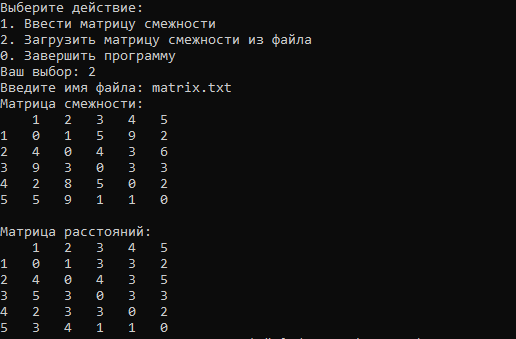


Рисунок 8 –Тестирование ввода матрицы из файла

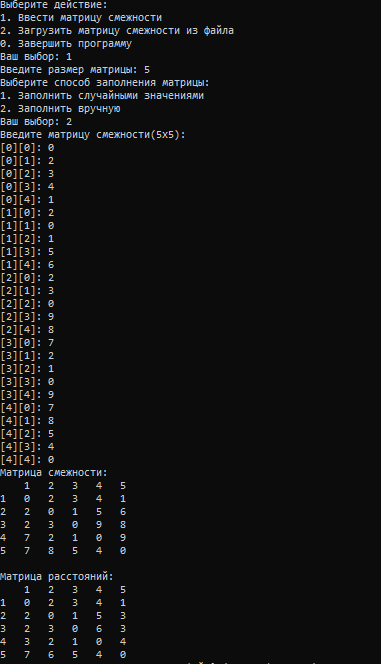


Рисунок 9 – Тестирование ручного ввода матрицы

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод сообщения о выборе: ввести матрицу смежности или загрузить матрицу смежности из файла | Верно |
| Выбор ввода матрицы смежности | Вывод сообщения о просьбе ввести размер матрицы и выбрать способ заполнения матрицы: случайно или вручную | Верно |
| Выбор загрузки матрицы смежности из файла | Вывод просьбы ввести имя файла, после чего выводится матрица из файла и матрица расстояний для неё | Верно |
| Выбор заполнения матрицы смежности случайными элементами | Выводится матрица смежности, заполненная случайными элементами и матрица расстояний для неё | Верно |
| Выбор заполнения матрицы смежности вручную | Пользователю предоставляется возможность ввести каждый элемент матрицы вручную с клавиатуры, после чего выводится сама введённая матрица и матрица расстояний для неё | Верно |
| Выбор: сохранить результат работы программы в файл или нет | Вывод выбора о сохранении результата работы программы в файл | Верно |

.

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

# 5. Ручной расчёт задачи

Проведем проверку программы посредством ручных вычислений на примере графа с 5 вершинами (рис. 10).

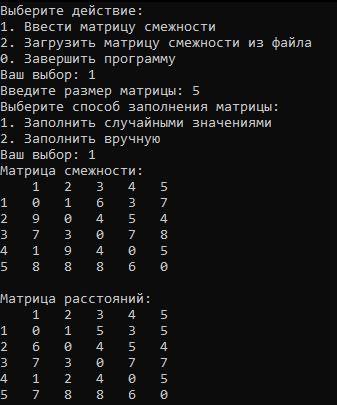
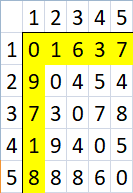
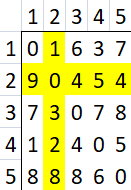


Рисунок 10 – Матрица смежности для ручного расчёта

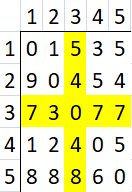
Нам дана исходная матрица, введём промежуточную вершину k = 1. Это означает, что вершина 1 является промежуточной.



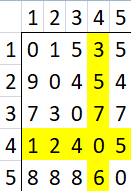
После этого смотрим на пересечения вершин 2-3, 2-4, 2-5 и так с последующими. На пересечениях мы смотрим на исходный элемент и сравниваем его с суммой элементов, которые у нас даны в промежуточной вершине. К примеру, при k = 1 у нас изменяются значения элемента 3-2 с 9 на 2, т.к. сумма элементов в промежуточных вершинах равна 2 (1 + 1), что меньше начального значения 9. Больше при k = 1 нет изменений. Берём k = 2, т.е. промежуточной становится вершина 2.



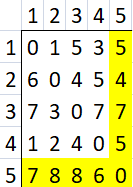
Поступаем по аналогии и видим, что у нас изменились элементы: 1-3 с 6 на 5, потому что сумма элементов в промежуточной вершине равна 5 (1+4), что меньше 6; 1-5 с 7 на 5, т.к. у нас сумма элементов в промежуточной вершине 5 (1+4), что меньше 7; 3-5 с 8 на 7, т.к. сумма элементов равна 7, что меньше 8. Обновляем матрицу.



Принимаем вершину 3 в качестве промежуточной, т.е. k = 3. Поступаем тем же путём, по которому у нас получились так, что никаких изменений не получает матрица. Меняем промежуточную вершину на 4, т.е. k = 4.



Здесь у нас изменения: элемент 2-1 изменился 9 на 6, т.к. сумма элементов в промежуточной вершине равна 6 (1 + 5); элемент 5-1 изменился с 8 на 7, т.к. сумма элементов в промежуточной вершине равна 7 (1 + 6). Больше изменений нет, подставляем последнюю промежуточную вершину k = 5.



Здесь никаких изменений не последует. Мы получаем итоговую матриц расстояний и делаем вывод, что программа рассчитывает всё верно.

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Флойда в Microsoft Visual Studio 202.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей. Приобретены навыки по осуществлению алгоритма поиска минимальных расстояний между вершинами в графе. Углублены знания языка программирования Cи.

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.

# 

# Список литературы

1. Реализация Алгоритма Флойда-Уоршелла [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Алгоритм_Флойда-Уоршелла> (Дата обращения: 21.12.2023)
2. Алгоритм Флойда-Уоршелла. [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/105825> (Дата обращения: 21.12.2023)
3. Алгоритм Флойда [Электронный ресурс]. – Режим доступа: <https://youtu.be/uVtDIxA9BQ4?si=YHF1HT0-T_OyLs1t> (Дата обращения: 21.12.2023)

**Приложение А.**

# Листинг программы.

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <Windows.h>

#include <locale.h>

#define BIG 999999999

// Функция для вывода матрицы смежности

void printMatrix(int a, int\*\* graph) {

printf(" ");

// Вывод заголовка с номерами столбцов

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

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

}

printf("\n");

// Вывод матрицы смежности

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

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

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

if (graph[i][j] > BIG) {

printf("big");

}

else {

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

}

}

printf("\n");

}

}

// Функция для генерации случайного графа

void generateRandomGraph(int a, int\*\* graph) {

srand(time(NULL));

// Заполнение матрицы случайными весами, 0 по диагонали

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

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

if (i == j) {

graph[i][j] = 0;

}

else {

graph[i][j] = rand() % 9 + 1;

}

}

}

}

// Функция для ручного ввода матрицы смежности

void manualInputGraph(int a, int\*\* graph) {

printf("Введите матрицу смежности(%dx%d):\n", a, a);

// Ввод элементов матрицы с проверкой на корректность

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

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

while (1) {

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

if (scanf("%d", &graph[i][j]) == 1 && graph[i][j] <= BIG) {

break;

}

else {

printf("Ошибка ввода. Пожалуйста, введите корректные данные (не больше %d).\n", BIG);

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

}

}

}

}

}

// Функция для загрузки матрицы смежности из файла

void fileInputGraph(int\* a, int\*\*\* graph) {

FILE\* file;

char filename[256];

do {

printf("Введите имя файла: ");

if (scanf("%s", filename) != 1) {

printf("Ошибка ввода. Пожалуйста, введите корректные данные.\n");

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

filename[0] = '\0';

}

} while (filename[0] == '\0');

file = fopen(filename, "r");

while (file == NULL) {

printf("Ошибка открытия файла. Пожалуйста, введите корректное имя файла: ");

if (scanf("%s", filename) != 1) {

printf("Ошибка ввода. Пожалуйста, введите корректные данные.\n");

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

filename[0] = '\0';

}

file = fopen(filename, "r");

}

// Выделение памяти под матрицу

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

for (int i = 0; i < \*a; i++) {

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

// Чтение элементов матрицы из файла

for (int j = 0; j < \*a; j++) {

if (fscanf(file, "%d", &(\*graph)[i][j]) != 1) {

printf("Ошибка чтения элементов матрицы из файла.\n");

exit(1);

}

// Пропустить разделители (пробелы, табуляции)

int delimiter;

do {

delimiter = fgetc(file);

} while (delimiter == ' ' || delimiter == '\t');

// Вернуть разделитель в поток

ungetc(delimiter, file);

}

}

// Закрытие файла

fclose(file);

}

// Функция для выполнения алгоритма Флойда

void floyd(int a, int\*\* graph) {

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

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

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

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

dist[i][j] = graph[i][j];

}

}

// Алгоритм Флойда

for (int k = 0; k < a; k++) {

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

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

if (dist[i][k] != 0 && dist[k][j] != 0 && dist[i][k] + dist[k][j] < dist[i][j])

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

// Вывод матрицы расстояний после алгоритма Флойда

printf("\nМатрица расстояний:\n");

printf(" ");

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

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

}

printf("\n");

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

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

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

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

}

printf("\n");

}

// Освобождение памяти

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

free(dist[i]);

}

free(dist);

}

// Функция для сохранения результатов в файл

void saveToFile(int a, int\*\* graph) {

FILE\* file;

char filename[256];

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

scanf("%s", filename);

// Открытие файла для записи

file = fopen(filename, "w");

if (file == NULL) {

printf("Ошибка открытия файла для записи.\n");

exit(1);

}

// Сначала сохраняем матрицу смежности

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

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

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

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

// Вставляем пробелы между элементами матрицы

if (j < a - 1) {

fprintf(file, " ");

}

}

fprintf(file, "\n");

}

// Затем сохраняем матрицу расстояний (просто копируем код вывода матрицы расстояний)

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

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

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

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

dist[i][j] = graph[i][j];

}

}

fprintf(file, "\nМатрица расстояний:\n");

fprintf(file, " ");

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

fprintf(file, "%-4d", i + 1);

}

fprintf(file, "\n");

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

fprintf(file, "%-4d", i + 1);

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

fprintf(file, "%-4d", dist[i][j]);

}

fprintf(file, "\n");

}

// Освобождение памяти

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

free(dist[i]);

}

free(dist);

// Закрытие файла

fclose(file);

}

// Основная функция программы

int main() {

int a;

int\*\* graph = NULL;

setlocale(LC\_ALL, "Rus"); // Установка русской локали

printf("Курсовой проект по курсу «Логика и основы алгоритмизации в инженерных задачах» на тему «Реализация алгоритма Флойда»\n");

printf("\n");

printf("Выполнил студент группы 22ВВВ1:\n");

printf("Коннов А.Д.\n");

printf("\n");

printf("Приняли:\n");

printf("Акифьев И.В.\n");

printf("Юрова О.В.\n");

printf("\n");

printf("Пенза - 2023\n");

printf("\n");

// Основной цикл программы

while (1) {

printf("Выберите действие:\n");

printf("1. Ввести матрицу смежности\n");

printf("2. Загрузить матрицу смежности из файла\n");

printf("0. Завершить программу\n");

int choice;

int fillChoice;

do {

printf("Ваш выбор: ");

if (scanf("%d", &choice) != 1) {

printf("Ошибка ввода. Пожалуйста, введите корректные данные.\n");

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

choice = -1;

}

} while (choice < 0);

switch (choice) {

case 1:

do {

printf("Введите размер матрицы: ");

if (scanf("%d", &a) != 1 || a <= 0) {

printf("Ошибка ввода. Пожалуйста, введите корректные данные.\n");

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

a = -1;

}

} while (a < 0);

// Выделение памяти под матрицу

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

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

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

}

do {

printf("Выберите способ заполнения матрицы:\n");

printf("1. Заполнить случайными значениями\n");

printf("2. Заполнить вручную\n");

printf("Ваш выбор: ");

if (scanf("%d", &fillChoice) != 1 || (fillChoice != 1 && fillChoice != 2)) {

printf("Ошибка ввода. Пожалуйста, введите корректные данные.\n");

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

fillChoice = -1;

}

} while (fillChoice < 0);

switch (fillChoice) {

case 1:

generateRandomGraph(a, graph);

break;

case 2:

manualInputGraph(a, graph);

break;

default:

printf("Ошибка: неверный выбор заполнения матрицы.\n");

return 1;

}

// Вывод матрицы

printf("Матрица смежности:\n");

printMatrix(a, graph);

// Вывод матрицы расстояний

floyd(a, graph);

// Сохранение результатов в файл

char saveChoice;

do {

printf("Желаете сохранить результаты в файл? (1 - Да/2 - Нет): ");

if (scanf(" %c", &saveChoice) != 1 || (saveChoice != '1' && saveChoice != '2')) {

printf("Ошибка ввода. Пожалуйста, введите корректные данные.\n");

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

saveChoice = '0';

}

} while (saveChoice == '0');

if (saveChoice == '1') {

saveToFile(a, graph);

printf("Результаты сохранены в файл.\n");

}

// Освобождение памяти

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

free(graph[i]);

}

free(graph);

break;

case 2:

fileInputGraph(&a, &graph);

// Вывод матрицы

printf("Матрица смежности:\n");

printMatrix(a, graph);

// Вывод матрицы расстояний

floyd(a, graph);

// Сохранение результатов в файл

char saveChoice2;

printf("Желаете сохранить результаты в файл? (1 - Да/2 - Нет): ");

scanf(" %c", &saveChoice2);

if (saveChoice2 == '1') {

saveToFile(a, graph);

printf("Результаты сохранены в файл.\n");

}

// Освобождение памяти

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

free(graph[i]);

}

free(graph);

break;

case 0:

return 0; // Завершение программы

default:

printf("Ошибка: неверный выбор.\n");

return 1;

}

}

}