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 Дейкстры»

Выполнил:

Студент группы 22ВВВ2 Шатин Д.Д.

Принял:

Акифьев И.В.

Пенза 2023

Изображение выглядит как текст, письмо, документ, меню

Автоматически созданное описание

Изображение выглядит как текст, бумага, документ, меню

Автоматически созданное описание

## Оглавление

[Реферат 6](#_bookmark0)

[Введение 7](#_bookmark1)

1. [Постановка задачи 8](#_bookmark2)
2. [Теоретическая часть задания 9](#_bookmark3)
3. [Описание алгоритма программы 10](#_bookmark4)
4. [Описание программы 11](#_bookmark5)
5. [Тестирование 14](#_bookmark6)
6. [Ручной расчёт задачи 17](#_bookmark7)

[Заключение 19](#_bookmark8)

[Список литературы 20](#_bookmark9)

[Приложение А 21](#_bookmark10)

# Реферат

Отчёт стр. 23, рисунков 15, таблиц 1.

ГРАФ, ТЕОРИЯ ГРАФОВ, ДОСТИЖИМОСТЬ, АЛГОРИТМ ДЕЙКСТРЫ.

Цель исследования – разработка программы с реализованным алгоритмом Дейкстры.

# Введение

В качестве цели данной курсовой работы создание программы, которая будет следовать кратчайшему пути к вершине.

В 1959 году нидерландским ученым, которого звали Эдсгер Дейкстра. был разработан алгоритм, основанный на графах, который стал известен как алгоритм Дейкстры. Он способен находить кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм может применяться только для графов с положительным весом. В программировании данный алгоритм широко используется, например, его используют в протоколах маршрутизации OSPF и IS-IS. Этот алгоритм широко используется в транспортных и сетевых системах, маршрутизации пакетов данных и других областях, где необходимо находить оптимальные маршруты в графах с неотрицательными весами рёбер.

В качестве среды разработки была выбрана среда Microsoft Visual Studio2022, 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.

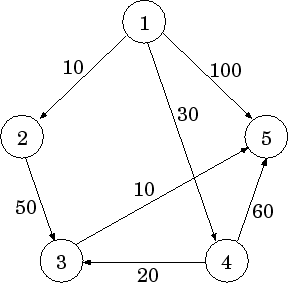


Рисунок 1 – Пример правильного построения графа

Алгоритм Дейкстры — это метод нахождения кратчайших путей от одной вершины графа ко всем остальным. Он был разработан Эдсгером Дейкстрой и эффективно работает в графах с неотрицательными весами ребер.

Алгоритм начинает с начальной вершины, постепенно расширяя область поиска, пока не будет найден кратчайший путь до всех остальных вершин.

Он широко применяется в сетевом проектировании, GPS-навигации, оптимизации маршрутов и других областях, где необходимо находить оптимальные пути в графовых структурах.

# Описание алгоритма программы

Сначала задаются библиотеки, необходимые для работы, далее на консоль выводится титульный лист курсового проекта и задаются переменные. Управление программой производится с помощью клавиатуры в консоли, где можно выбрать количество вершин графа, тип графа (ориентированный или неориентированный) и его заполнение (случайным образом, вручную, либо считывание матрицы из файла). Также в программе учтены различные ошибочные варианты ввода расстояний и количества вершин, что может предотвратить нечаянно нажатый символ, не соответствующий просьбе ввода.

Реализация алгоритма:устанавливаем начальную вершину (узел) в графе с индексом begin\_index - d[begin\_index] = 0 в ноль в массиве d, что обозначает, что расстояние от начальной вершины до самой себя равно нулю.

Этот алгоритм ищет кратчайшие пути от начальной вершины до всех остальных вершин в графе.

**Выбираем вершины с минимальным весом** (min):Алгоритм ищет вершину (minchislo), которая ещё не была обработана (v[i] == 1), и имеет наименьший вес (d[i] < min). Эта вершина выбирается для дальнейшей обработки.

После выбора вершины с минимальным весом, алгоритм обновляет расстояния (d[i]) до смежных с ней вершин (a[minchislo][i] > 0). Если новый путь через эту выбранную вершину короче, чем текущий путь до вершины **i**, то обновляется значение d[i] на это более короткое расстояние.

**Отмечаем вершины как обработанные:** после обработки выбранной вершины, она помечается как обработанная (v[minchislo] = 0;) для предотвращения повторного прохода.

Цикл продолжает работу, пока есть не посещенные вершины (minchislo < 100000), что в данном случае является условием выхода из цикла.

После выполнения алгоритма программа выводит кратчайшие пути до каждой из вершин и самый короткий путь от начальной вершины до конечной. Результат записывается в файл, а также выводится на экран. Программу можно прогнать снова, по выбору пользователя.

**Псевдо-код алгоритма:**

Пусть SIZE - количество вершин в графе

Пусть a[][] - матрица смежности графа (веса ребер)

Пусть v[] - массив для отслеживания посещенных вершин

Пусть d[] - массив кратчайших расстояний от начальной вершины до остальных

Инициализируем массивы d[] и v[]

начальная\_вершина = выбор\_начальной\_вершины()

// Инициализация расстояний

для каждой вершины i от 0 до SIZE - 1 {

d[i] = бесконечность

v[i] = 1

}

d[начальная\_вершина] = 0

повторять {

minchislo = 100000

min = 100000

// Находим вершину с минимальным расстоянием

для каждой вершины i от 0 до SIZE - 1 {

если (v[i] == 1) и (d[i] < min) {

min = d[i]

minchislo = i

}

}

// Обновляем расстояния до смежных вершин

если (minchislo != 100000) {

для каждой вершины i от 0 до SIZE - 1 {

если (a[minchislo][i] > 0) {

s = min + a[minchislo][i]

если (s < d[i]) {

d[i] = s

}

}

}

v[minchislo] = 0

}

} пока (minchislo < 100000)

# 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.

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).

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Автоматически созданное описание

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)Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 4 – вывод случайно матрицы в консоль и файлИзображение выглядит как текст, снимок экрана, Шрифт, черно-белый

Автоматически созданное описание

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 7 вершинами в графе при случайном заполнении матрицы (Рисунок 6)

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рисунок 6 – тестирование случайного заполнения

Выполнение программы с 4 вершинами в графе при ручном заполнении матрицы (Рисунок 7)

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 7 – тестирование ручного заполнение

Выполнение программы с некорректным вводом вершин (Рисунок 8)

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

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

(Рисунок 9)

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

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)

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 10 – выполнение для графа из 4 вершин Воссоздадим граф по заданной матрице смежности (Рисунок 11)

Изображение выглядит как диаграмма, круг, линия

Автоматически созданное описание

Рисунок 11 – граф из 4 вершин

Изображение выглядит как диаграмма, текст, линия, Шрифт

Автоматически созданное описание

Рисунок 12,13,14,15 – 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 выполняется корректно.

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм 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.

# Список литературы

1. Wikipedia.com –- [Электронный ресурс] --- Алгоритм Дейкстры. Режим доступа: [Алгоритм Дейкстры — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/Алгоритм_Дейкстры) (Дата обращения: 19.12.23)

2. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.(Дата обращения: 11.12.23)

3. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006. (Дата обращения: 12.12.23)

**4.GeeksforGeeks**  ---- [Электронный ресурс] --- Режим доступа: <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>. (Дата обращения: 15.12.23)

5**.** Goldberg, Andrew V., и Robert E. Tarjan. "A new approach to the shortest-path problem" // Journal of the ACM (JACM), 1984. (Дата обращения: 16.12.23)

# 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 <ctype.h>

#include <time.h>

void titul() {

printf("\n");

printf(" Министерство науки и высшего образования Российской Федерации\n");

printf(" Пензенский государственный университет\n");

printf(" Кафедра «Вычислительная кафедра»\n");

printf("\n\n");

printf(" КУРСОВОЙ ПРОЕКТ\n");

printf(" по дисциплине:\n ЛОГИКА И ОСНОВЫ АЛГОРИТМИЗАЦИИ В ИНЖЕНЕРНЫХ ЗАДАЧАХ\n");

printf(" тема проекта:\n РЕАЛИЗАЦИЯ АЛГОРИТМА ДЕЙКСТРЫ\n ");

printf("\n\n\n");

printf(" Выполнил:\n");

printf(" студент группы 22ВВП1\n");

printf(" Шатин Д.Д.\n");

printf("\n\n");

printf(" Принял:\n");

printf(" Акифьев И.В.\n");

printf("\n\n\n");

printf(" Пенза 2023\n");

time\_t startTime = time(nullptr);

while (difftime(time(nullptr), startTime) < 5) {}

}

int repeat\_program() {

int repeat;

char choice;

do {

printf("\nХотите запустить программу заново? (y/n): ");

scanf(" %c", &choice);

getchar(); // Очистка буфера ввода

if (choice != 'y' && choice != 'Y' && choice != 'n' && choice != 'N') {

printf("!!! Введите 'y' для повторного запуска или 'n' для завершения!\n\n");

}

} while (choice != 'y' && choice != 'Y' && choice != 'n' && choice != 'N');

repeat = (choice == 'y' || choice == 'Y') ? 1 : 0;

system("cls");

return repeat;

}

void input\_distance(int\*\* a, int SIZE) {

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

a[i][i] = 0;

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

int s = -1;

while (s <= 0 || s >= 20) {

printf("Введите расстояние между вершинами %d и %d (1-19): ", i + 1, j + 1);

scanf("%d", &s);

getchar(); // Очищаем буфер ввода

if (s <= 0 || s >= 20) {

printf("!!! Введите корректное расстояние (1-19)!\n\n");

}

}

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

}

}

}

void count\_vertex(int\* count) {

int flag = 0;

char value[6];

do {

flag = 0;

printf("Введите размерность графа ( 1 < размер < 11 ): ");

scanf("%5s", value);

getchar(); // Чтобы прочитать символ новой строки из буфера ввода

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

if (!isdigit(value[i]) || isspace(value[i])) {

flag = 1;

break;

}

}

if (!flag) {

if (atoi(value) == 1) {

flag = 1;

}

if (atoi(value) > 10) {

flag = 1;

}

}

if (flag == 1) {

printf("!!! Введите корректное число вершин!\n\n");

printf("> Нажмите любую клавишу, чтобы продолжить...");

getchar();

system("cls");

}

else {

\*count = atoi(value);

break;

}

} while (flag);

}

void createRandomDirectedGraph(int\*\* a, int SIZE) {

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

a[i][i] = 0;

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

if (i != j) {

int probability = rand() % 100; // Генерируем вероятность от 0 до 99

if (probability < 50) { // Например, задаем вероятность 50%

a[i][j] = rand() % 20; // Устанавливаем случайное значение от 0 до 19

}

else {

a[i][j] = 0; // Устанавливаем 0 в противном случае

}

}

}

}

}

void createRandomUndirectedGraph(int\*\* a, int SIZE) {

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

a[i][i] = 0;

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

int weight = rand() % 20;

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

}

}

}

void readMatrixFromFile(int\*\* a, int SIZE, const char\* filename) {

FILE\* file;

int success = 0;

do {

file = fopen(filename, "r");

if (file == NULL) {

printf("Ошибка при открытии файла. Пожалуйста, введите верное имя файла: ");

scanf("%s", filename);

}

else {

success = 1;

}

} while (!success);

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

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

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

}

}

fclose(file);

}

int main()

{

setlocale(LC\_ALL, "Russian");

titul();

FILE\* fp;

int s = -1, minchislo, min, menu = 0;

int\*\* a; // матрица связей

int\* d; // минимальное расстояние

int\* v; // посещенные вершины

int begin\_index = 0;

char namefile[20];

char keyboard;

int choice = 0;

int num = 1;

int isDirected = -1;

system("chcp 1251");

system("cls");

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

int SIZE = 0;

count\_vertex(&SIZE);

fp = fopen("Результат.txt", "w");

if (fp == NULL) {

printf("Ошибка при открытии файла.\n");

return 1; // Или обработайте ошибку соответствующим образом

}

int readFromFile = -1;

while (readFromFile == -1) {

printf("\nВыберите источник данных для графа:\n1. Ввод с клавиатуры\n2. Чтение из файла\n");

char keyboard = \_getch();

if (keyboard == 49 || keyboard == 50) {

readFromFile = (keyboard == 49) ? 0 : 1;

}

}

// меню

system("clear || cls");

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

for (int i = 0; i < SIZE; i++) a[i] = (int\*)malloc(SIZE \* sizeof(int));

if (readFromFile) {

char filename[50];

printf("Введите имя файла для чтения данных: ");

scanf("%s", filename);

readMatrixFromFile(a, SIZE, filename);

}

else {

while (isDirected == -1) {

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

char keyboard = \_getch();

if (keyboard == 49 || keyboard == 50) {

isDirected = (keyboard == 49) ? 1 : 0;

}

}

while (menu == 0) {

printf("\nНажмите клавишу для выбора действия\n1.Случайное заполнение\n2.Заполнение вручную\n3.Выход\n");

keyboard = \_getch();

if (keyboard == 49) {

if (isDirected) {

createRandomDirectedGraph(a, SIZE);

}

else {

createRandomUndirectedGraph(a, SIZE);

}

menu = 1;

}

if (keyboard == 50) {

input\_distance(a, SIZE); // Вызов функции для ручного ввода расстояний

menu = 1;

}

if (keyboard == 51) exit(0);

}

}

// Вывод матрицы связей

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");

}

//Инициализация вершин и расстояний

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

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

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

{

d[i] = 100000;

v[i] = 1;

}

d[begin\_index] = 0;

// Шаг алгоритма

do {

minchislo = 100000;

min = 100000;

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

{ // Если вершину ещё не обошли и вес меньше min

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

{ // Переприсваиваем значения

min = d[i];

minchislo = i;

}

}

// Добавляем найденный минимальный вес

// к текущему весу вершины

// и сравниваем с текущим минимальным весом вершины

if (minchislo != 100000)

{

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

{

if (a[minchislo][i] > 0)

{

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

if (s < d[i])

{

d[i] = s;

}

}

}

v[minchislo] = 0;

}

} while (minchislo < 100000);

// Вывод кратчайших расстояний до вершин

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]);

}

// Восстановление пути

int\* ver; // массив посещенных вершин

int end = SIZE - 1; // индекс конечной вершины

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

ver[0] = end + 1; // начальный элемент - конечная вершина

int k = 1; // индекс предыдущей вершины

int weight = d[end]; // вес конечной вершины

while (end != begin\_index) // пока не дошли до начальной вершины

{

for (int i = 0; i < SIZE; i++) // просматриваем все вершины

if (a[i][end] != 0) // если связь есть

{

int s = weight - a[i][end]; // определяем вес пути из предыдущей вершины

if (s == d[i]) // если вес совпал с рассчитанным

{ // значит из этой вершины и был переход

weight = s; // сохраняем новый вес

end = i; // сохраняем предыдущую вершину

ver[k] = i + 1; // и записываем ее в массив

k++;

}

}

}

// Вывод пути (начальная вершина оказалась в конце массива из k элементов)

printf("\nВывод кратчайшего пути\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);

int repeat = repeat\_program();

if (repeat) {

main();

}

else {

system("cls");

}

return 0;

}