

**Содержание**

[Содержание 4](#_Toc9206)

[Реферат 5](#_Toc19963)

[Введение 6](#_Toc5774)

[1 Постановка задачи 7](#_Toc22238)

[2 Теоретическая часть задания 8](#_Toc15877)

[3 Описание алгоритма поставленной задачи 12](#_Toc28087)

[4 Описание программы 14](#_Toc22240)

[5 Тестирование программы 16](#_Toc31645)

[5.1 Методика тестирования 16](#_Toc20974)

[5.2 Пример ручного расчёта задачи и вычислений 17](#_Toc4559)

[Заключение 18](#_Toc6240)

[Список используемых источников 19](#_Toc17921)

[Приложение А. Листингпрограммы 20](#_Toc32084)

[Приложение Б. Блок-схема 23](#_Toc18984)

**Реферат**

Отчёт 19 стр, 2 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ОРГРАФ, АЛГОРИТМ ФЛОЙДА, ВЗВЕШЕННЫЙ ГРАФ.

Цель исследования – разработка программы, которая реализует алгоритм Флойда, осуществляющий поиск кратчайших расстояний между всеми вершинами взвешенного ориентированного графа.

При выполнении данной работы были получены навыки разработки программ и освоены приёмы создания матриц, основанных на теории графов.

# Введение

Благодаря своему широкому применению, теория о нахождении кратчайших расстояний в последнее время интенсивно развивается.Задача о поиске кратчайшего пути на графе может быть интерпретирована по-разному и применяться в различных областях: в картографических сервисах для нахождения путей между физическими объектами, недетерминированных машинах для поиска оптимальной последовательности решений для достижения главной цели, для поиска кратчайшего расстояния в сети дорог.

Компьютерная техника и информационные технологии позволяют моделировать объекты, явления и процессы, тем самым упрощая решение различных задач.

Для реализации компьютерной модели, большое значение имеет такое научное направление, как программирование. Без него компьютер это просто набор различных устройств, микросхем, который не может быть полезным.

Целью данной курсовой работы является разработка программы на языке С, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Флойда, осуществляющий поиск кратчайших расстояний между всеми вершинами взвешенного ориентированного графа.

# 1 Постановка задачи

Необходимо осуществить программную реализацию алгоритма поиска кратчайших расстояний между всеми вершинами взвешенного ориентированного графа (Алгоритм Флойда).

Программа должна работать так, чтобы пользователь мог самостоятельно выбрать порядок случайно созданного ориентированного графа. После обработки этих данных на экран должна выводиться матрица весов рёбер графа и матрица кратчайших путей, затем будет возможность сохранить полученные результаты в файл. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь. Необходимо научиться различать события, возникающие в процессе работы с устройствами ввода. Необходимо однозначно идентифицировать и выполнять те или иные действия в зависимости от выбранной команды.

Данная программа может использоваться в дискретной математике для исследования графов или в качестве наглядного пособия, демонстрирующего применение алгоритма Флойда на практике.

# 2 Теоретическая часть задания

Наиболее часто используемое название, метод «Алгоритм Флойда – Уоршелла» получил в честь двух американских исследователей Роберта Флойда и Стивена Уоршелла, одновременно открывших его в 1962 году. Реже встречаются другие варианты наименований: алгоритм Рой – Уоршелла или алгоритм Рой – Флойда. Рой – фамилия профессора, который разработал аналогичный алгоритм на 3 года раньше коллег (в 1959 г.), но это его открытие осталось безвестным.

Алгоритм Флойда – Уоршелла – динамический алгоритм вычисления значений кратчайших путей для каждой из вершин графа. Метод работает на взвешенных графах, с положительными и отрицательными весами ребер, но без отрицательных циклов, являясь, таким образом, более общим в сравнении с алгоритмом Дейкстры, т. к. последний не работает с отрицательными весами ребер, и к тому же классическая его реализация подразумевает определение оптимальных расстояний от одной вершины до всех остальных.

Ключевая идея алгоритма — разбиение процесса поиска кратчайших путей на фазы.

Перед -ой фазой () считается, что в матрице расстояний  сохранены длины таких кратчайших путей, которые содержат в качестве внутренних вершин только вершины из множества  (вершины графа мы нумеруем, начиная с единицы).



Иными словами, перед -ой фазой величина  равна длине кратчайшего пути из вершины  в вершину , если этому пути разрешается заходить только в вершины с номерами, меньшими  (начало и конец пути не считаются).



Легко убедиться, что чтобы это свойство выполнилось для первой фазы, достаточно в матрицу расстояний  записать матрицу смежности графа:  — стоимости ребра из вершины  в вершину . При этом, если между какими-то вершинами ребра нет, то записать следует величину "бесконечность" . Из вершины в саму себя всегда следует записывать величину , это критично для алгоритма.



Пусть теперь мы находимся на -ой фазе, и хотим пересчитать матрицу  таким образом, чтобы она соответствовала требованиям уже для -ой фазы. Зафиксируем какие-то вершины  и . У нас возникает два принципиально разных случая:



* Кратчайший путь из вершины  в вершину , которому разрешено дополнительно проходить через вершины , совпадает с кратчайшим путём, которому разрешено проходить через вершины множества .



В этом случае величина  не изменится при переходе с -ой на -ую фазу.



* "Новый" кратчайший путь стал лучше "старого" пути.

Это означает, что "новый" кратчайший путь проходит через вершину . Сразу отметим, что мы не потеряем общности, рассматривая далее только простые пути (т.е. пути, не проходящие по какой-то вершине дважды).



Тогда заметим, что если мы разобьём этот "новый" путь вершиной  на две половинки (одна идущая , а другая — ), то каждая из этих половинок уже не заходит в вершину . Но тогда получается, что длина каждой из этих половинок была посчитана ещё на -ой фазе или ещё раньше, и нам достаточно взять просто сумму , она и даст длину "нового" кратчайшего пути.



Объединяя эти два случая, получаем, что на -ой фазе требуется пересчитать длины кратчайших путей между всеми парами вершин  и  следующим образом:



new\_d[i][j] = min (d[i][j], d[i][k] + d[k][j]);

Таким образом, вся работа, которую требуется произвести на -ой фазе — это перебрать все пары вершин и пересчитать длину кратчайшего пути между ними. В результате после выполнения -ой фазы в матрице расстояний  будет записана длина кратчайшего пути между  и , либо , если пути между этими вершинами не существует.



Последнее замечание, которое следует сделать, — то, что можно не создавать отдельную матрицу  для временной матрицы кратчайших путей на -ой фазе: все изменения можно делать сразу в матрице . В самом деле, если мы улучшили (уменьшили) какое-то значение в матрице расстояний, мы не могли ухудшить тем самым длину кратчайшего пути для каких-то других пар вершин, обработанных позднее.



# 3 Описание алгоритма поставленной задачи

На вход программе подаётся граф, заданный в виде матрицы смежности — двумерного массива  размера , в котором каждый элемент задаёт длину ребра между соответствующими вершинами.



Требуется, чтобы выполнялось  для любых .



for (int k=0; k<n; ++k)

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

for (int j=0; j<n; ++j)

d[i][j] = min (d[i][j], d[i][k] + d[k][j]);

Предполагается, что если между двумя какими-то вершинами **нет ребра**, то в матрице смежности было записано какое-то большое число (достаточно большое, чтобы оно было больше длины любого пути в этом графе); тогда это ребро всегда будет невыгодно брать, и алгоритм сработает правильно.

Правда, если не принять специальных мер, то при наличии в графе рёбер **отрицательного веса**, в результирующей матрице могут появиться числа вида , , и т.д., которые, конечно, по-прежнему означают, что между соответствующими вершинами вообще нет пути. Поэтому при наличии в графе отрицательных рёбер алгоритм Флойда лучше написать так, чтобы он не выполнял переходы из тех состояний, в которых уже стоит "нет пути":



for (int k=0; k<n; ++k)

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

for (int j=0; j<n; ++j)

if (d[i][k] < INF && d[k][j] < INF)

d[i][j] = min (d[i][j], d[i][k] + d[k][j]);

Полный код программы можно увидеть в Приложении А.

Блок-схему программы можно увидеть в приложении Б.

# 4 Описание программы

Для написания данной программы использован язык программирования Си. Си - язык программирования общего назначения, который очень известен своей эффективностью, экономичностью ресурсов и переносимостью на других программные платформы. Все эти преимущества обеспечивают качество разработки программного обеспечения, их быстроту и сравнительно небольшой размер.

В качестве среды программирования был выбран программный продукт Microsoft Visual Studio 2019. Проект был создан в виде консольного приложения Win32 (Visual C++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: main, printM, copyM, FU, orgraf, saveorg1, saveorg2.

Работа программы начинается с объявления о создании матрицы динамически и запроса введения порядка орграфа. Затем выводятся элементы массива, а именно матрица весов рёбер графа и матрица кратчайших путей. Ниже можно увидеть оформление.

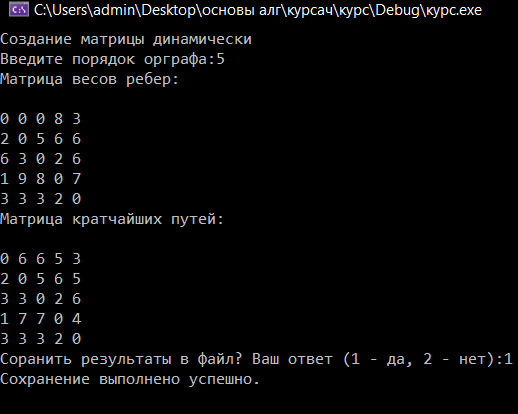


Рисунок 1 – Вид программы

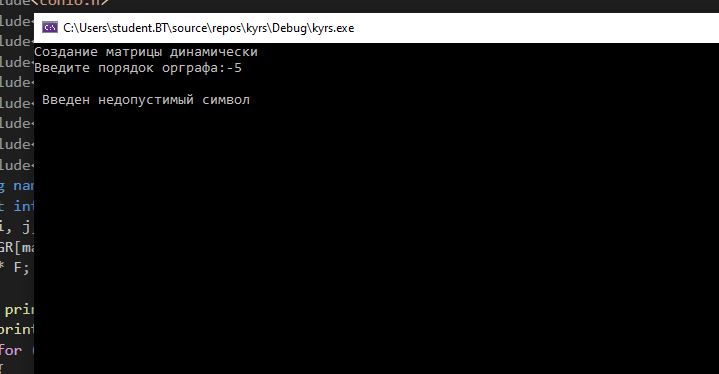


Рисунок 2 – Ввод числа со знаком «-»

Далее предоставляется возможность сохранить полученные результаты в файл. Так выглядит файл с результатами:

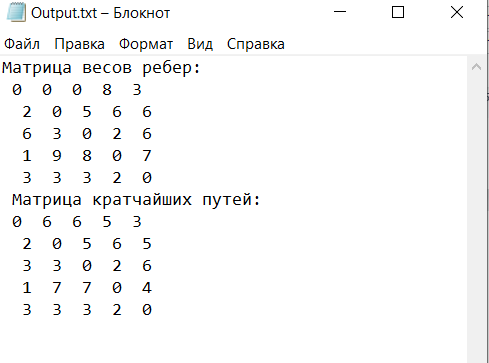


Рисунок 2 – Файл с результатами

# 5 Тестирование программы

## 5.1 Методика тестирования

Среда разработки Microsoft Visual Studio 2019 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. Также были предусмотрены выводы предупреждений об ошибках при недопустимых действиях пользователя.

Таблица 1 – Описание поведения программы при различных действиях пользователя

|  |  |
| --- | --- |
| Действие пользователя | Ожидаемый результат |
| Введение корректного количества вершин графа | Инициализация матрицы и вывод матрицы весов рёбер и матрицы кратчайших путей |
| Введение отрицательного количества вершин графа | Вывод сообщения об ошибке и повторение запроса о вводе корректного количества вершин графа |

## 5.2 Пример ручного расчёта задачи и вычислений

Проведём проверку программы посредством ручных вычислений на примере графа с тремя вершинами.

Метод не предусматривает работу с петлями, поэтому кратчайшие расстояния от вершины до самой себя будут равняться нулю.

Для первой вершины. Путь от первой вершины ко второй составляет - 1, к третьей - 2. Следовательно, они и будут являться кратчайшими.

Для второй вершины. Из второй вершины есть путь к первой и третьей, равные 1 и 1. Следовательно, они и будут кратчайшим путями.

Для третьей вершины. Есть путь только к первой вершине равный - 8. Он и будет кратчайшим, а путь ко второй будет складываться из пути к первой - 8 и из первой во вторую - 1. В итоге - 9.

Таким образом, можно сделать вывод, что программа работает верно.

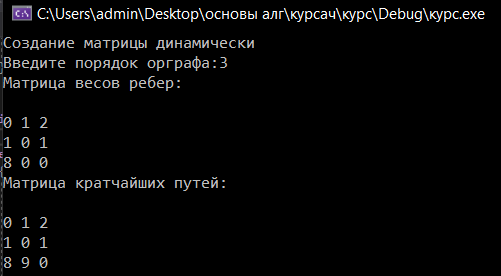


Рисунок 3 – Тестирование работы программы

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Флойда в Microsoft Visual Studio 2019 C++.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приёмы создания матриц, основанных на теории графов. Приобретены навыки по осуществлению алгоритма поиска кратчайших путей. Углублены знания языка программирования C.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Это связано с тем, что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

В дальнейшем программу можно улучшить, добавив псевдографический интерфейс, вывод сообщений обо всех ошибках и подсказок при неверном действии пользователя, оптимизировав код.

**Список литературы**

1. Алексеев В.В., Гаврилов Г.П., Сапоженко А.А. (ред.) Теория графов. Покрытия, укладки, турниры. Сборник переводов - М. : Мир, 1974.— 224 с.

2. Основные понятия графов сайт https://skysmart.ru/articles/mathematic/osnovnye-ponyatiya-teorii-grafov

3. Алексеев В.Е., Захарова Д.В. Теория графов. Учебное пособие 2017 г.

4. Алгоритмы и программы. Язык С++. Е. Конова, Г. Поллак

5. Демидович Е.М. Основы алгоритмизации и программирования. Язык СИ :учебн. Пособие. – СПб.: БХВ-Петербург, 2006.

6. Дэвис С. С++ для «чайников». – К. : Диалектика, 2005

# Приложение А. Листингпрограммы

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdafx.h>

#include<conio.h>

#include<locale.h>

#include<time.h>

#include<iostream>

#include<queue>

#include<stack>

#include<windows.h>

#include<fstream>

#include<iostream>

using namespace std;

const int maxV=100;

int i, j, n, variant;

int GR[maxV][maxV], D[maxV][maxV];

FILE \*F;

void printM(int M[][maxV]) {

printf("\n");

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

{

for (int j = 0; j < n; j++)

{

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

}

printf("\n");

}

}

void copyM(int M[][maxV], int M1[][maxV]) {

for (char i = 0; i < n; i++)

{

for (char j = 0; j < n; j++)

{

M1[i][j] = M[i][j];

}

}

}

void FU(int R[][maxV], int V)

{

int k;

copyM(R,D);

for (i=0; i<V; i++) D[i][i]=0;

for (k=0; k<V; k++)

for (i=0; i<V; i++)

for (j=0; j<V; j++)

if (D[i][k] && D[k][j] && i!=j)

if (D[i][k]+D[k][j]<D[i][j] || D[i][j]==0)

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

}

void orgraf(int R[maxV][maxV]) {

printf("Введите порядок орграфа:");

cin>>n;

if (n <0)

{

printf("\n Введен недопустимый символ\n");

\_getch();

system("cls");

printf("Введите порядок орграфа:");

cin>>n;

}

if (n >100)

{

printf("\n Введенное число превосходит допустимое значение(100) \n");

\_getch();

system("cls");

printf("Введите порядок орграфа:");

cin>>n;

}

srand(time (NULL));

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

{

for (int j = 0; j < n; j++)

{

if (i == j)

R[i][j] = 0;

else

R[i][j] = rand()%10;

}

}

}

void saveorg1(int R[][maxV]) {

if ((F = fopen("Output.txt", "a")) == 0)

printf("Невозможно открыть/создать для записи файл 'Output.txt'\n");

else

{

fprintf(F,"Матрица весов ребер: \n");

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

{

for (int j = 0; j < n; j++)

{

fprintf(F," %d ", R[i][j]);

}

fprintf(F,"\n ");

}

}

fclose(F);

}

void saveorg2(int R[][maxV]) {

if ((F = fopen("Output.txt", "a")) == 0)

printf("Невозможно открыть/создать для записи файл 'Output.txt'\n");

else

{

fprintf(F,"Матрица кратчайших путей: \n");

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

{

for (int j = 0; j < n; j++)

{

fprintf(F," %d ", R[i][j]);

}

fprintf(F,"\n ");

}

}

fclose(F);

}

int main()

{

setlocale(LC\_ALL, "Rus");

printf("Создание матрицы динамически \n");

orgraf(GR);

cout<<"Матрица весов ребер:\n";

printM(GR);

cout<<"Матрица кратчайших путей:"<<endl;

FU(GR, n);

printM(D);

printf("Сохранить результаты в файл? Ваш ответ (1 - да, 2 - нет):");

cin>>variant;

if (variant!=1 && variant!=2){

printf("Введен недопустимый символ. Введите ваш ответ: ");

cin>>variant;}

else {

if (variant == 1) {

saveorg1(GR);

saveorg2(D);

cout<<"Сохранение выполнено успешно.\n";

}

}

system("pause>>void");

}

**Приложение Б. Блок-схема**

