Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «**Алгоритм Флойда**»

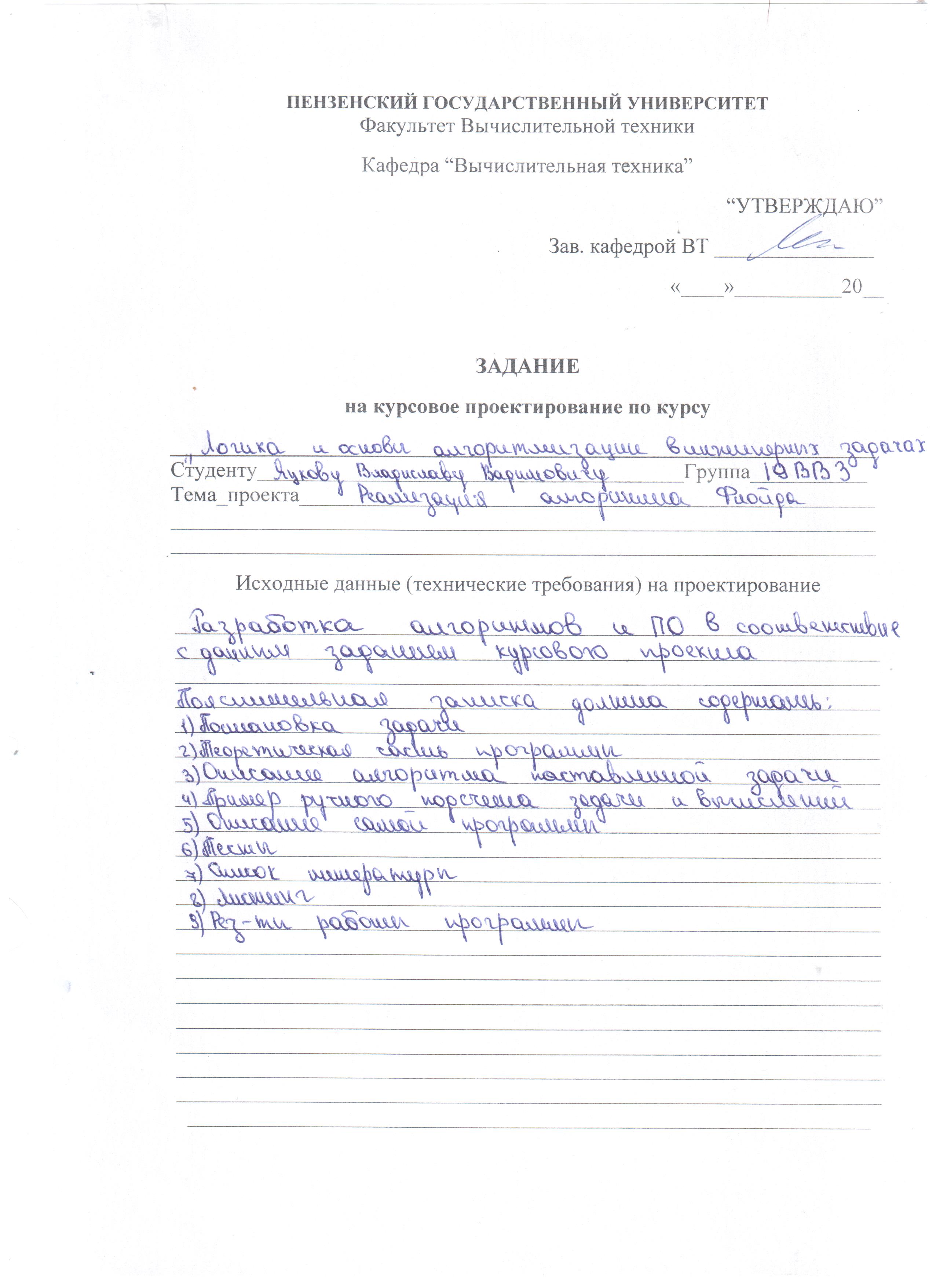
***Выполнил работу студент группы 19ВВ3:***

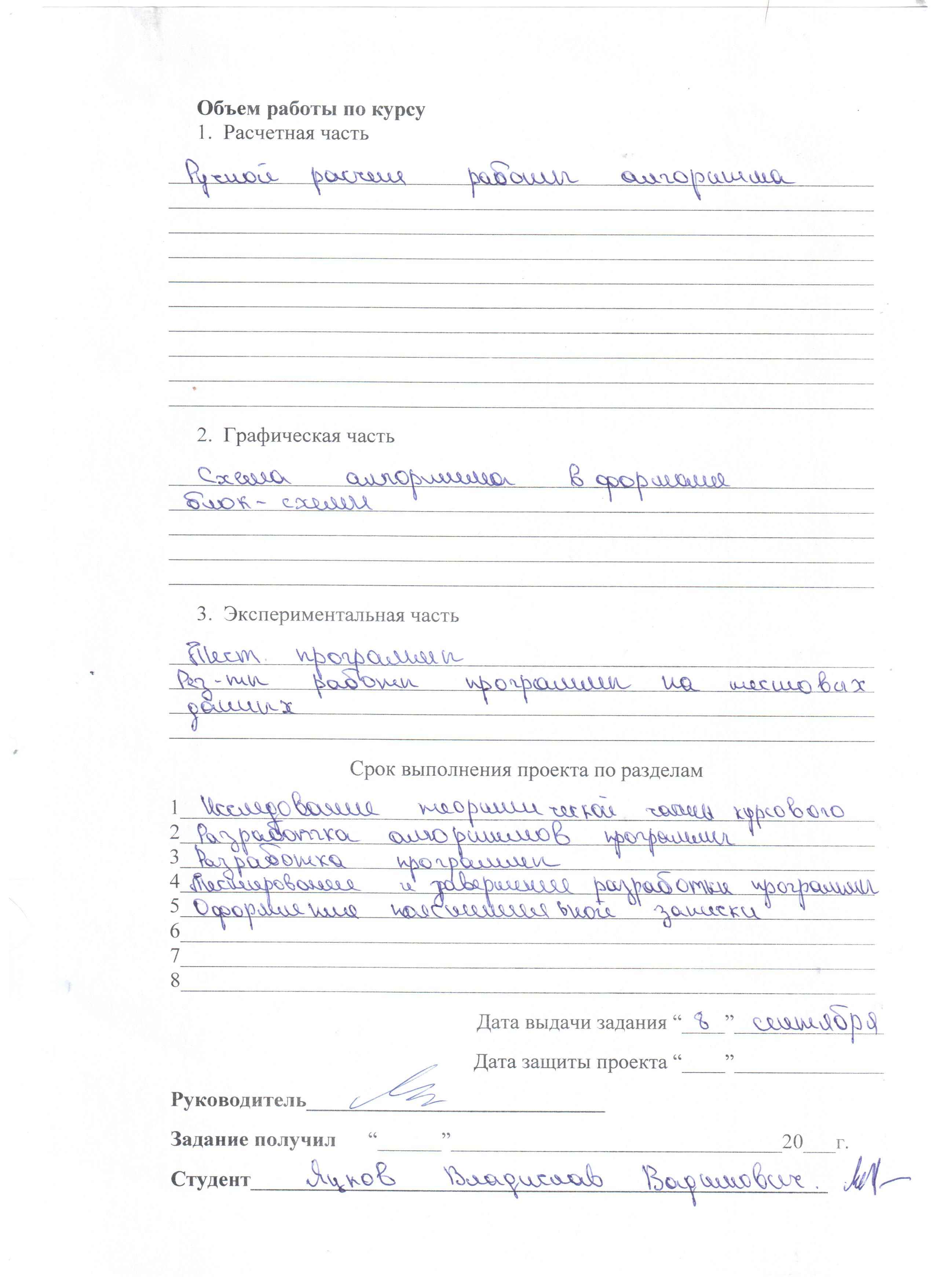
Яцков В.В

**Приняли:**

д.т.н. профессор Митрохин М. А.

Пенза 2020





**Содержание**

[Реферат 5](#_Реферат.)

[Введение 6](#_Введение)

1. [Постановка задачи 7](#_TOC_250007)
2. [Теоретическая часть задания 8](#_TOC_250006)
3. [Описание алгоритма программы 10](#_TOC_250005)
4. [Описание программы 1](#_Описание_алгоритма_программы)2
5. [Тестирование](#_TOC_250003) 24

[Заключение 3](#_TOC_250001)9

[Список литературы 40](#_TOC_250000)

[Приложение A. Листинг программы 41](#_Приложение_А.)

# Реферат.

Отчет по курсовой работе: 47 страниц, 30 рисунков, 1 таблица, 1 приложение и 7 источников.

Цель работы — создание программы для реализации алгоритма Флойда.

Ключевые слова: алгоритм, граф, матрица, массив, поиск кратчайших путей, консоль, цикл, размер матрицы, вершина, вес ребра, файл, функция, программа, аргументы.

Использование алгоритма Флойда – Уоршелла позволяет определить самые короткие пути между любой парой вершин графа. Согласно условию задачи, имеем неориентированный или ориентированный взвешенный граф G, который имеет n вершин. Необходимо определить величину всех кратчайших путей d(i, j) между вершинами i и j, при условии отсутствия в графе циклов с отрицательными весами (в этом случае решения для некоторых пар вершин могут быть бесконечно малыми, то есть несуществующими). Данный алгоритм описали в своих работах независимо друг от друга и примерно в одно и тоже время Роберт Флойд и Стивен Уоршелл в 1962-м году. В их честь и назвали алгоритм, именуемый и сегодня алгоритмом Флойда – Уоршелла. Однако следует заметить, что ещё в 1959-ом году Бернардом Роем был разработан почти аналогичный алогитм, но он не был тогда никем замечен.

# Введение

Алгоритм Флойда — алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного графа без циклов с отрицательными весами с использованием метода динамического программирования. Алгоритм работает за Θ(n3) времени и использует Θ(n2) памяти. В основном Алгоритм Флойда-Уоршелла является эффективным для расчёта всех кратчайших путей в плотных графах, когда имеет место большое количество пар рёбер между парами вершин. Данный программный продукт позволяет автоматизировать расчёт нахождения кратчайшего расстояния между всеми вершинами взвешенного ориентированного графа и призван упростить работу инженерам.

Как и любой базовый алгоритм, алгоритм Флойда — Уоршелла используется очень широко и много где, начиная от поиска транзитивного замыкания графа, заканчивая генетикой и управлением проектами. Но первое что приходит в голову конечно же транспортные и всякие другие сети.  
Скажем если вы возьмете карту города — её транспортная система это граф, соответственно присвоив каждому ребру некую стоимость, рассчитанную скажем из пропускной способности и других важный параметров — вы сможете подвести попутчика по самому короткому/быстрому/дешевому пути.

# 

# Постановка задачи

Требуется создать динамический алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного ориентированного графа, путем реализации алгоритма Флойда. При этом, одной из главных особенностей моего программного продукта является вызов параметров генерации массива из командной строки, рандомное заполнение массива случайными числами, считывания массива из файла и ручной ввод матрицы в консоли, а также сохранение результатов в файл.

Дан взвешенный граф, предполагается, что граф не содержит циклов отрицательного веса (тогда ответа между некоторыми парами вершин может просто не существовать — он будет бесконечно маленьким). Программа должна работать так, чтобы пользователь индивидуально вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться первоначальная матрица и матрица, содержащая кратчайшие пути, найденные методом Флойда. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Целью данной курсовой работы является разработка программы в MicrosoftVisualStudio2019. Языком программирования выбран - Си, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Флойда, осуществляющий поиск кратчайших путей.

Устройство ввода – клавиатура и мышь.

# 

# Теоретическая часть задания

Для определенности положим, что вершины графа последовательно пронумерованы от 1 до n. Алгоритм Флойда использует матрицу А размера n \* n, в которой вычисляются длины кратчайших путей. Вначале A[i, j] = C[i, j] для всех i != j. Если дуга i -» j отсутствует, то C[i, j] = infinity. Каждый диагональный элемент матрицы А равен 0.

Над матрицей А выполняется п итераций. После k-й итерации A[i, j] содержит значение наименьшей длины путей из вершины i в вершину у, которые не проходят через вершины с номером, большим k. Другими словами, между концевыми вершинами пути i и у могут находиться только вершины, номера которых меньше или равны k. На k-й итерации для вычисления матрицы А применяется следующая формула:  
**Ak[i,j]=min(Ak-1[i,j], Ak-1[i,k]+Ak-1[k,j])**

Нижний индекс k обозначает значение матрицы А после k-й итерации, но это не означает, что существует п различных матриц, этот индекс используется для сокращения записи. Для вычисления Ak[i, j] проводится сравнение величины Ak-i[i, j] (т.е. стоимость пути от вершины i к вершине j без участия вершины k или другой вершины с более высоким номером) с величиной Ak-1[i, k] + Ak-1[k, j] (стоимость пути от вершины i до вершины k плюс стоимость пути от вершины k до вершины j). Если путь через вершину k дешевле, чем Ak-1[i, j], то величина Ak[i, j] изменяется. Пример работы программы показан на рисунке 1

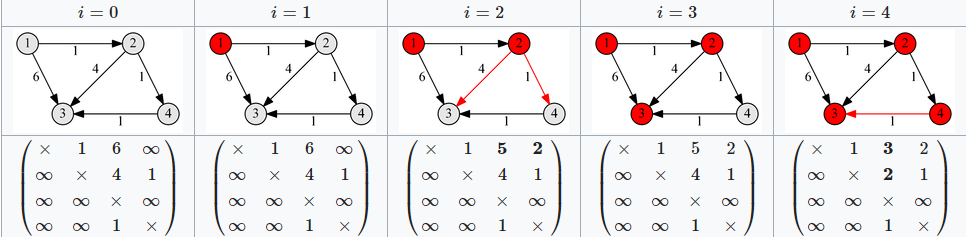


Рисунок 1 – Пример работы.

Время выполнения этой программы, очевидно, имеет порядок 0(V3), поскольку в ней практически нет ничего, кроме вложенных друг в друга трех циклов. Доказательство "правильности" работы этого алгоритма также очевидно и выполняется с помощью математической индукции по k, показывая, что на k-й итерации вершина k включается в путь только тогда, когда новый путь короче старого.

# 

# Описание алгоритма программы

Для реализации алгоритма Флойда – Уоршелла сформируем матрицу смежности **D[][]** графа **G=(V, E)**, в котором каждая вершина пронумерована от **1** до **|V|**. Эта матрица имеет размер**|V|´|V|**, и каждому ее элементу **D[i][j]** присвоен вес ребра, идущего из вершины **i** в вершину **j**. По мере выполнения алгоритма, данная матрица будет перезаписываться: в каждую из ее ячеек внесется значение, определяющее оптимальную длину пути из вершины **i** в вершину **j** (отказ от выделения специального массива для этой цели сохранит память и время).

Теперь, перед составлением основной части алгоритма, необходимо разобраться с содержанием матрицы кратчайших путей. Поскольку каждый ее элемент **D[i][j]** должен содержать наименьший из имеющихся маршрутов, то сразу можно сказать, что для единичной вершины он равен нулю, даже если она имеет петлю (отрицательные циклы не рассматриваются), следовательно, все элементы главной диагонали (**D[i][i]**) нужно обнулить.

А чтобы нулевые недиагональные элементы (матрица смежности могла иметь нули в тех местах, где нет непосредственного ребра между вершинами **i** и **j**) сменили по возможности свое значение, определим их равными бесконечности, которая в программе может являться, например, максимально возможной длинной пути в графе, либо просто – большим числом.

Если в графе есть циклы отрицательного веса, то формально алгоритм Флойда-Уоршелла к такому графу неприменим. Но на самом деле алгоритм корректно сработает для всех пар, пути мужду которыми никогда не проходят через цикл негативной стоимости, а для остальных мы получим какие-нибудь числа, возможно сильно отрицательные.

**Ключевая часть алгоритма, состоя из трех циклов, выражения и условного оператора, записывается довольно компактно:**

**Для k от 1 до |V| выполнять**  
**Для i от 1 до |V| выполнять**  
**Для j от 1 до |V| выполнять**  
**Если D[i][k]+D[k][j]<D[i][j] то D[i][j] ←D[i][k]+D[k][j]**

Кратчайший путь из вершины **i** в вершину **j** может проходить, как только через них самих, так и через множество других вершин **k∈(1, …, |V|)**. Оптимальным из **i** в **j** будет путь или не проходящий через **k**, или проходящий. Заключить о наличии второго случая, значит установить, что такой путь идет из **i** до **k**, а затем из **k** до **j**, поэтому должно заменить, значение кратчайшего пути **D[i][j]** суммой **D[i][k]+D[k][j]**.

Ниже представлен псевдокод функции FLOID:

1. Для i=0 пока i<V делать i++
2. arr[i][i]=0
3. Конец цикла
4. Для k=0 пока k<V делать k++
5. Для i=0 пока i<V делать i++
6. Для j=0 пока j<V делать j++
7. Если с arr[i][k] и arr[k][j] и i не равно j
8. Если сумма arr[i][k] и arr[k][j] меньше arr[i][j] или arr[i][j] равно 0
9. arr[i][j] = arr[i][k] + arr[k][j]
10. Конец условия
11. Конец условия
12. Конец цикла
13. Конец цикла
14. Конец цикла
15. Для i=0 пока i<V делать i++
16. Для j=0 пока j<V делать j++
17. Вывод arr[i][j]
18. Конец цикла
19. Вывод \n
20. Конец цикла

# 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++), и является многомодульным, поскольку состоит из нескольких функций.

Работа программы начинается с вызова аргументов с открытия пользователем командной строки и ввода .\kurs.exe. Переменная int argc отвечает за кол-во, переменная char\* argv отвечает за название аргументов. После пользователю предложены на выбор следующие параметры: ориентированный/неориентированный граф, размер матрицы, вероятность появления ребра, максимальный вес элемента.

int main(int argc, char\* argv[]) {

if (argc <= 3) {

printf("Курсовая работа на тему «Реализация алгоритма Флойда» \n");

printf("Выполнил студент группы 19ВВ3:\nЯцков В.В\n\n");

printf("Вариант работы программы с помощью вызова параметров в командной строки \n\n");

printf("Введите параметры в командной строке через пробел.\n");

printf("Варианты параметров:\n");

printf("Ориентированный граф - route.\n");

printf("Неориентированный граф - nroute.\n");

printf("Размер массива - укажите число.\n");

printf("Вероятность появления ребра - укажите число.\n");

printf("Максимальный вес ребра - укажите число.\n");

return 1;

}

После ввода нужных параметров, программа проверяет правильность их ввода и их кол-во. Первым аргументом для проверки служит выбор ориентированного или неориентированного графа. Далее программа проверяет количество введенных параметров меньше 4рех, то программа еще раз выводит подсказку ввода параметров. Если количество введенных параметров больше 5, программа выведет ошибку с текстом «Слишком много параметров. Попробуйте снова!». Если числовой параметр будет введен неверно, программа выведет ошибку с текстом «Число не обнаружено! Попробуйте снова!»

if ((argc >= 4) && (strcmp(argv[1], "route") == 0)) {

if (argc > 5) {

printf("Слишком много параметров. Попробуйте снова!\n");

return 1;

}

if (argv[2] != NULL) {

n = atoi(argv[2]);

}

else if (n == 0) {

printf("Число не обнаружено! Попробуйте снова!\n");

return (0);

}

if (argv[3] != NULL) {

ver = atoi(argv[3]);

}

else if (ver == 0) {

printf("Число не обнаружено! Попробуйте снова!\n");

return (0);

}

if (argv[4] != NULL) {

max = atoi(argv[4]);

}

else if (max == 0) {

printf("Число не обнаружено! Попробуйте снова!\n");

return (0);

}

Если аргументы введены корректно программа создает динамический массив исходя из указанных параметров. Далее передает этот массив для генерации матрицы указанного типа (ориентированная/неориентированная). Далее эта матрица передается в основную функцию данной программы, функция реализации алгоритма Флойда. Далее следует вывод исходной матрицы и матрицы кратчайшиз путей на экран.

arr = dynamic(n);

generate\_typeV\_route(arr, n);

printf("Матрица смежности взвешенного ориентированного графа:\n\n");

print(arr, n);

printf("Матрица кратчайших путей: \n");

FLOID(arr, n);

Далее программа предлагает переход в основное меню программы. Для этого пользователь вводит 1, а программа в свою очередь проверяет данное условие, если введенное число = 1, то программа переходит в основное меню. Пример работы с аргументами мы можем увидеть на рисунке 2.

printf("Для перехода в режим работы консольного приложения программы введите 1\n");

scanf("%d", &usl);

if (usl == 1) {

printf("Основная часть программы!");

}

}

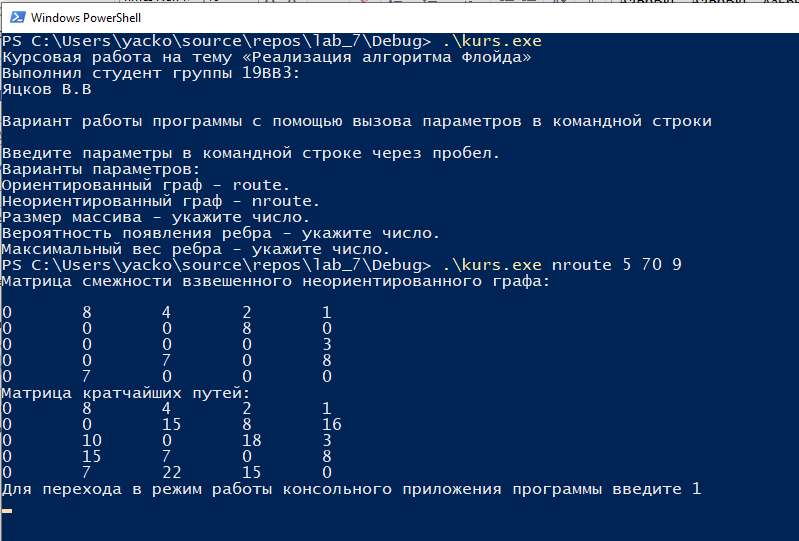


Рисунок 2 – Работа программы с аргументами.

Выводится основное меню программы (Рис.3), для продолжения пользователю предложено ввести пункт с которым он хочет продолжить взаимодействие. Этот пункт записывается в переменную item. После этого программа проверяет введенный символ и отправляет пользователя на ту часть программы которая принадлежит введенному пункту.

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

printf("Курсовая работа на тему «Реализация алгоритма Флойда» \n");

printf("Выполнил студент группы 19ВВ3\n");

printf("Яцков Владислав\n");

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

printf("\t1) Создать массив рандомно\n");

printf("\t2) Открыть массив из файла\n");

printf("\t3) Создать массив вручную в консоли\n");

printf("\t0) Завершить работу\n\n\n");

printf("\tВыберите пункт: ");

unsigned int item = 0;

scanf("%u", &item);

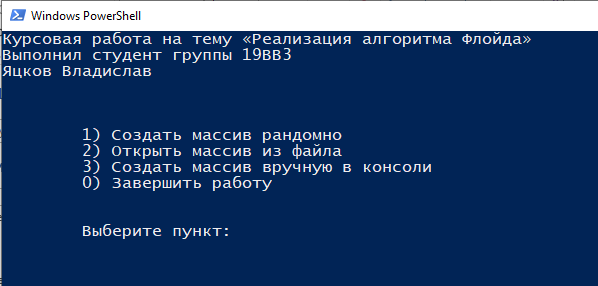


Рисунок 3 – Открытие основного меню.

Допустим пользователь выбирает 1 пункт меню, программа создает или отчищает (если файл уже есть) текстовый файл output. Сразу записывает в этот файл фразу «Первоначальный массив», далее на консоль выводится запрос от пользователя размера массива который будет создан рандомно. Далее выполняется функция malloc, отвечающая за динамическое выделение памяти. Исходя из введенного размера массива, рандомно генерируется массив и заполняется числами от 1 до 99. На экран выводится заданный массив

case 1: {

f\_output = fopen("output.txt", "w");

fprintf(f\_output, "Первоначальный массив:\n");

printf("Выберите размер массива:\n\n\n");

scanf("%d", &n);

num = (int\*)malloc(n \* sizeof(int));

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

num[i] = 0;

}

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

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

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

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

if (j > i) {

arr[i][j] = rand() % 100;

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

}

else if (i == j) arr[i][j] = 0;

}

}

printf("\nЗаданный массив: \n");

printf("");

После этого в программе присутствует цикл, отвечающий за вывод массива в файл в виде матрицы. Для вывода используется табуляция.

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

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

printf("%d \t", arr[i][j]);

fprintf(f\_output, " %d", arr[i][j]);

}

printf("\n");

fprintf(f\_output, "\n");

}

Далее на экран выводится матрица кратчайших путей, а именно исходная матрица после прохода через алгоритм Флойда. Далее также задан цикл, отвечающий за вывод массива в файл в виде матрицы.

{

printf("Матрица кратчайших путей: \n");

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

FLOID(arr, n);

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

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

fprintf(f\_output, " %d", arr[i][j]);

}

fprintf(f\_output, "\n");

}

}

Далее программа предлагает переход в основное меню программы или завершение работы. Для этого пользователь вводит 1 или 0, а программа в свою очередь проверяет данное условие, если введенное число = 1, то программа переходит в основное меню. Пример работы с 1 пунктом меню приведен на рисунке 4.

printf("\n\nДля возрата в главное меню нажмите 1\nДля завершения работы нажмите 0\n");

scanf("%d", &usl);

if (usl == 1) {

break;

}

return (0);

}

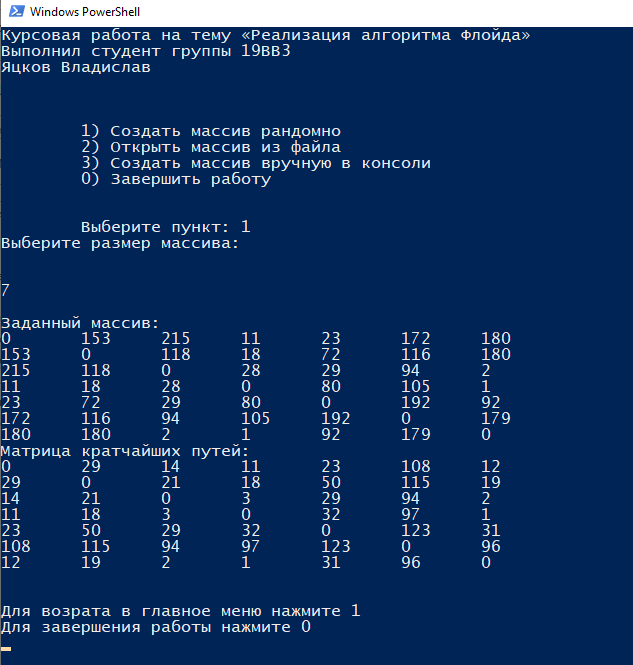


Рисунок 4 – Работа 1 пункта меню.

Далее рассмотрим второй пункт основного меню. Этот пункт отвечает за считывание массива из файла. Именно поэтому сначала программа открывает текстовый файл input в режиме чтения, а текстовый файл output отвечающий за вывод результатов в режиме записи. Далее программа проверяет количество элементов в файле и записывает их в переменную m, != EOF означает что программа дошла до конца файла. После этого вызывается функция rewind, которая переносит указатель на начало файла. Далее находим размер полученный матрицы, путем вычисления квадратного корня из переменной m. После этого, программа безошибочно считывает матрицу из файла. Выводится текст «Массив считан из файла». И вызывается функция print отвечающая за вывод матрицы в консоль. Файл input.txt закрывается.

case 2: {

f\_input = fopen("input.txt", "r");

f\_output = fopen("output.txt", "w");

fprintf(f\_output, "Первоначальный массив:\n");

// считывание данных с файла

while ((fscanf(f\_input, "%d", &s) != EOF)) {

if (!f\_input) break;

m += 1;

}

rewind(f\_input); // перематываем файл для повторного чтения

n = sqrt(m);

arr = dynamic(n);

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

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

fscanf\_s(f\_input, "%d", &arr[i][j]);

}

}

printf("Массив считан из файла.\n");

print(arr, n);

fclose(f\_input);

Далее программа выполняет запись, полученной матрицы кратчайших путей в текстовый файл после реализации алгоритма Флойда и возврата в главное меню. Данные действия описаны в первом пункте основного меню. Работоспособность второго пункта показана на рисунке 5.

printf("Матрица кратчайших путей: \n");

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

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

fprintf(f\_output, " %d", arr[i][j]);

}

fprintf(f\_output, "\n");

}

FLOID(arr, n);

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

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

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

fprintf(f\_output, " %d", arr[i][j]);

}

fprintf(f\_output, "\n");

}

}

printf("\n\nДля возрата в главное меню нажмите 1\nДля завершения работы нажмите 0\n");

scanf("%d", &usl);

if (usl == 1) {

break;

}

return (0);

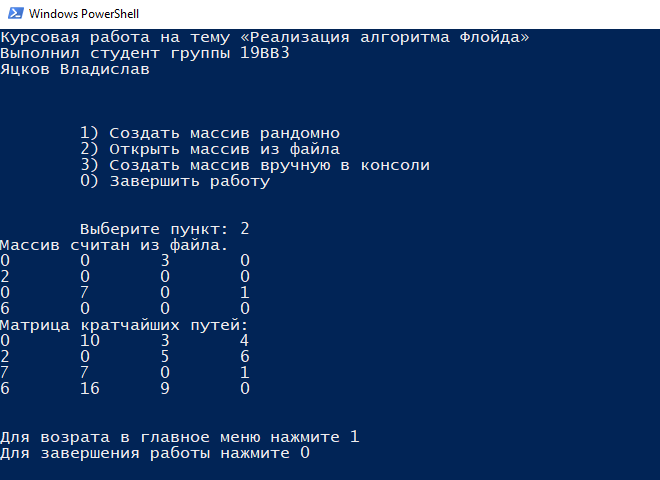


Рисунок 5– Работа 2 пункта меню.

Далее рассмотрим третий пункт основного меню. Этот пункт отвечает за ввод элементов массива в консоли. Программа уже привычно открывает текстовый файл output для записи результатов. Далее происходит запрос от пользователя размера матрицы и записывается в переменную n. Далее происходит создание динамического массива, исходя из размера введённым пользователем, через функцию malloc. Далее программа запрашивает ввести вес ребра для определённого элемента массива. Когда все элементы заданы, на экран выводится матрица и в текстовый файл сохраняется исходная матрица.

case 3: {

f\_output = fopen("output.txt", "w");

fprintf(f\_output, "Первоначальный массив:\n");

printf("Количество вершин в графе > ");

scanf("%d", &n);

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

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

printf("Введите элементы матрицы: \n");

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

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

{

printf("arr[%d][%d] > ", i + 1, j + 1);

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

}

printf("Заданная матрица:\n");

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

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

printf(" %d\t", arr[i][j]);

fprintf(f\_output, " %d", arr[i][j]);

}

printf("\n");

fprintf(f\_output, "\n");

}

Далее программа выполняет запись, полученной матрицы кратчайших путей в текстовый файл после реализации алгоритма Флойда и возврата в главное меню. Данные действия описаны в первом пункте основного меню. Работоспособность третьего пункта показана на рисунке 6.

printf("Матрица кратчайших путей: \n");

FLOID(arr, n);

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

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

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

fprintf(f\_output, " %d", arr[i][j]);

}

fprintf(f\_output, "\n");

}

fclose(f\_output);

printf("\n\nДля возрата в главное меню нажмите 1\nДля завершения работы нажмите 0\n");

scanf("%d", &usl);

if (usl == 1) {

break;

}

return (0);

}

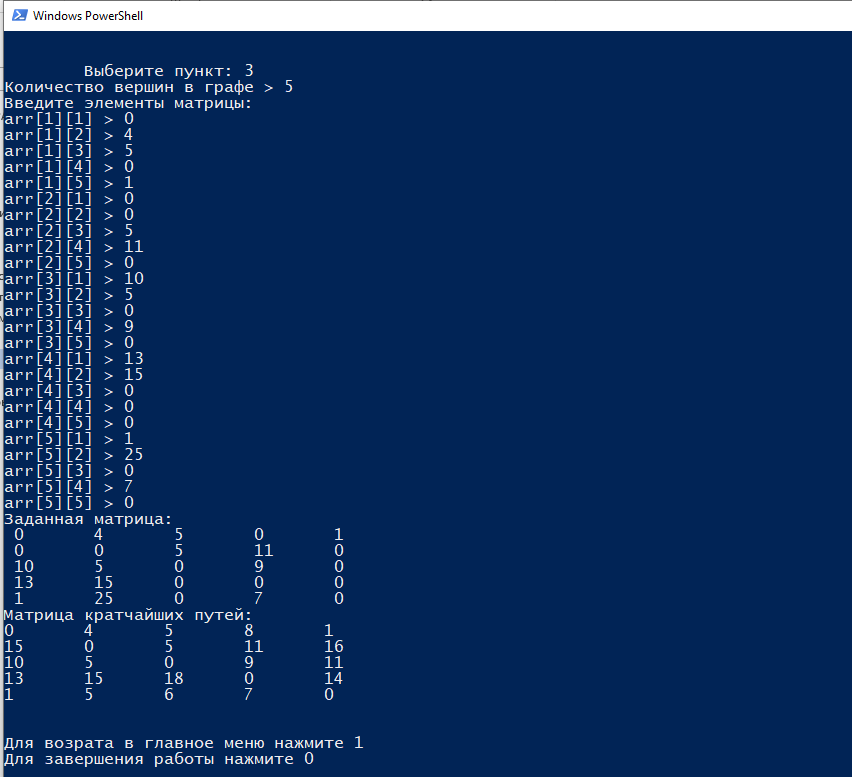


Рисунок 6 – Работа 3 пункта меню.

Пользователь всегда может завершить работу программы нажатием клавиши 0 или еще раз проследовать по уже реализованным пунктам, изменив какие-либо условия генерации матрицы по нажатию клавиши 1. Это можно рассмотреть на рисунке 7.

case 0:

return(0);

}

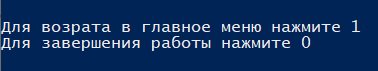


Рисунок 7 – Выход из программы

# Тестирование

Среда разработки MicrosoftVisualStudio 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 и выводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Ниже в таблице 1 продемонстрирован результат тестирования программы при выборе разных пунктов программы.

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |  |
| --- | --- | --- | --- |
| **Описание теста** | **Проделанные действия** | **Ожидаемый результат** | **Полученный результат** |
| Запуск программы | Запуск программы через cmd, ввод .\kurs.exe. | Успешное открытие командной строки и описание вариантов выбора параметров (Рис. 8) | Верно |
| Выбор параметров через cmd для генерации неориентированного графа, с размером 5, вероятностью 10%, максимальным весом ребра 100 | Ввод .\kurs.exe nroute 5 10 100 | Генерация матрицы с размером 5, вероятностью 10%, максимальным весом ребра 100 и матрицы кратчайших путей.  (Рис. 9) | Верно |
| Выбор параметров через cmd для генерации ориентированного графа, с размером 7, вероятностью 70%, максимальным весом ребра 90. | Ввод .\kurs.exe route 7 70 9 | Генерация матрицы с размером 7, вероятностью 70%, максимальным весом ребра 9 и матрицы кратчайших путей. (Рис. 10) | Верно |
| Ввод не всех параметров (т.е меньше 4рех) | Ввод .\kurs.exe nroute 10 | Вызов описания правильного ввода параметров. (Рис. 11) | Верно |
| Ввод лишних параметров (т.е больше 4рех) | Ввод .\kurs.exe nroute 10 20 30 30 30 30 | Вывод: «Слишком много параметров(Больше4). Попробуйте снова!» (Рис. 12) | Верно |
| Неправильный ввод параметра. | Ввод .\kurs.exe nroute 30 f | Вывод: «Число не обнаружено! Попробуйте снова!» (Рис. 13) | Верно |
| Рандомная генерация матрицы с указанием размера графа | Переход в основное меню, выбор пункта 1 (ввод 1), выбор размера матрицы (ввод 6) | Вывод сгенерированной рандомной матрицы с размером 6 и матрицы кратчайших путей (Рис. 14) | Верно |
| Вывод матрицы из файла | Переход в основное меню, выбор пункта 2 (ввод 2) | Вывод самостоятельно заданной матрицы из файла и матрицы кратчайших путей (Рис. 26) | Верно |
| Вывода матрицы сгенерированной вручную в консоли с указанием размера | Переход в основное меню, выбор пункта 3 (ввод 3), выбор размера матрицы (ввод 5) | Вывод самостоятельно заданной матрицы с размером 5 и матрицы кратчайших путей (Рис. 27) | Верно |
| Сохранение результата программы в файл | Выход из пунктов программы после вывода результата программы в консоль. | Успешное сохранение исходной матрицы, с указанием выбора пункта, и матрицы кратчайших путей. (Рис28, Рис.29, Рис.30) | Верно |

.

В результате тестирования было выявлено, что программа успешно реализует алгоритм Флойда во всех вариантах генерации матрицы.

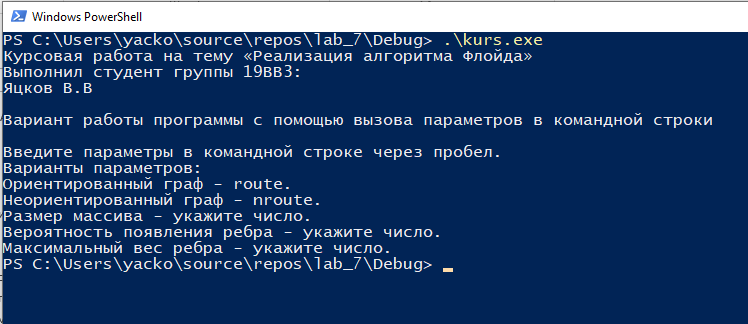


Рисунок 8 – Запуск программы

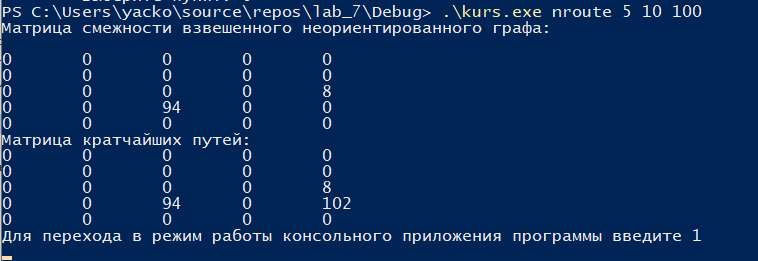


Рисунок 9 – Тестирование при вводе: «неориентированный граф», количество вершин 5, вероятность 10%, максимальный вес ребра 100.

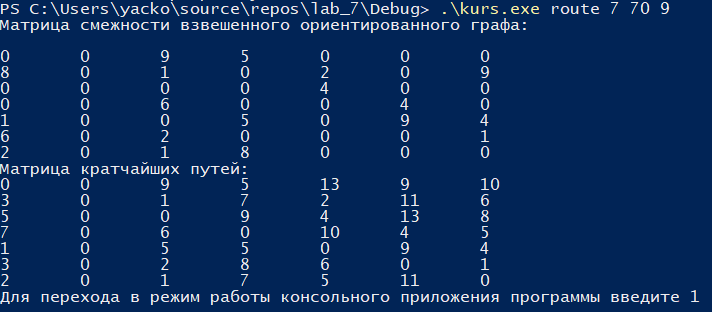


Рисунок 10 – Тестирование при вводе: «ориентированный граф», количество вершин 7, вероятность 70%, максимальный вес ребра 9.

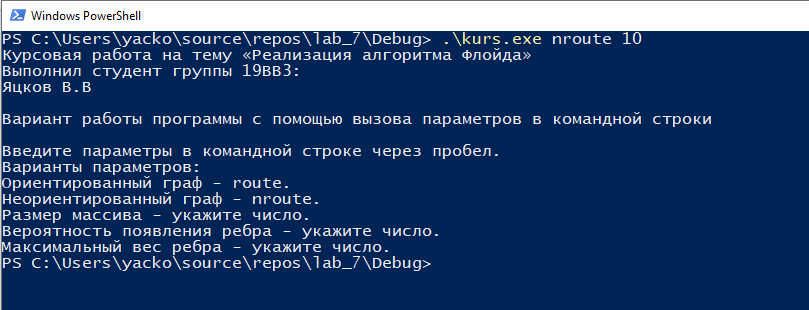


Рисунок 11 – Ввод не всех параметров

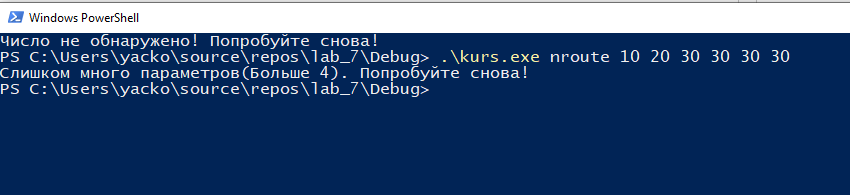


Рисунок 12 – Ввод лишних параметров

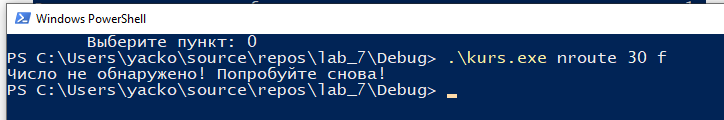


Рисунок 13 – Неправильный ввод параметра.

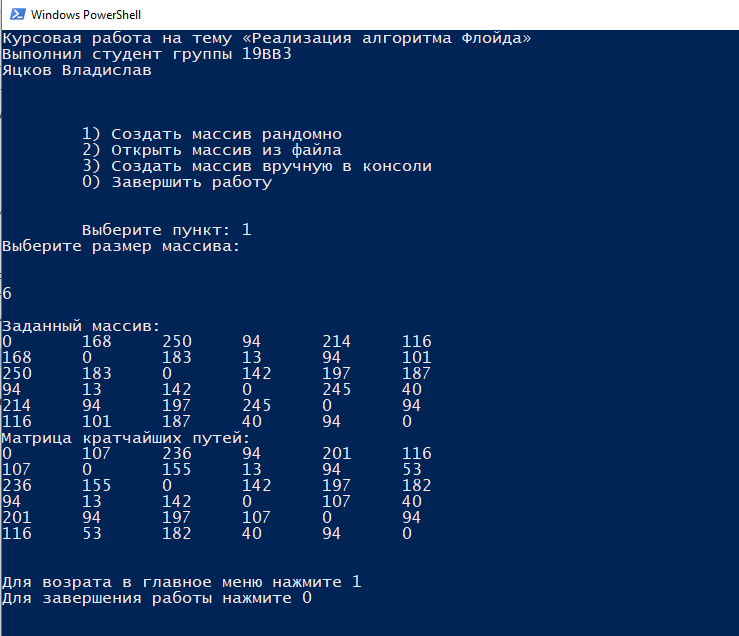
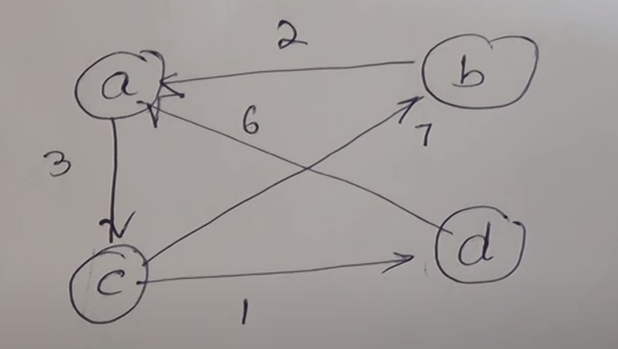


Рисунок 14 – Рандомная генерация матрицы. Заданный размер массива 6.

Для тестирования правильной работоспособности программы, нужно провести ручной просчет алгоритма, для проверки удобно использовать 2 пункт меню программы (ввод массива из файла).

**Ручной просчет программы.**

Проведем проверку программы посредством ручных вычислений на примере ориентированного графа с 5 вершинами (Рисунок 15).

Рисунок 15 –Ручной расчет задачи. Заданный граф.

Тогда начальная весовая матрица будет (Рисунок 16):

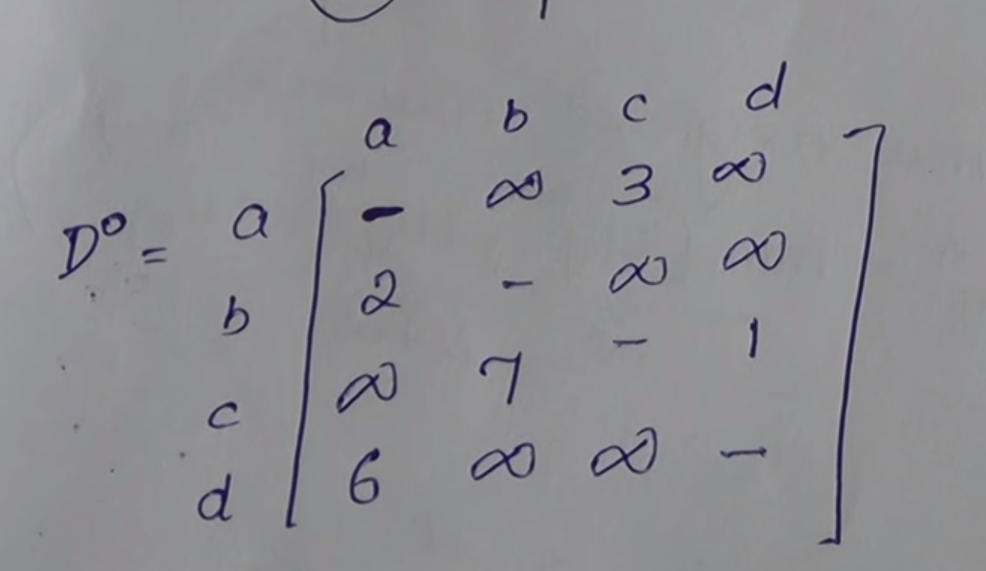


Рисунок 16 –Ручной расчет задачи. Заданная матрица.

Т.к матрица состоит из 4 вершин, для реализации алгоритма потребуется 4 шага, Чтобы записать матрицу с индексом 1, мы должны в исходной матрице выделить первую строчку и первый столбец (Рисунок 17):

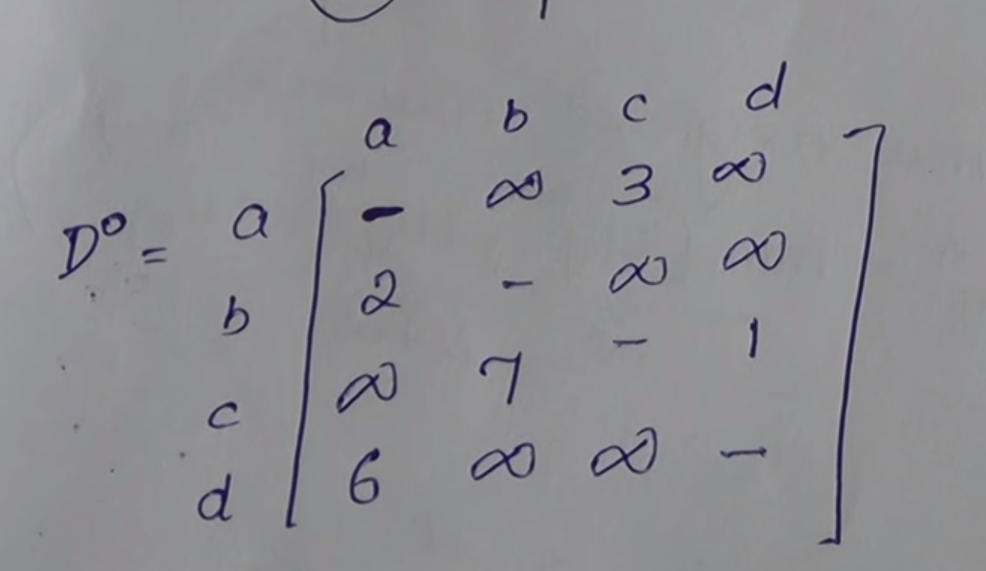


Рисунок 17 –Ручной расчет задачи. Шаг 1.

Далее для всех элементов в первой строчке и первом столбце, которые не равны бесконечности, мы смотрим на места их пересечения (Рисунок 18).

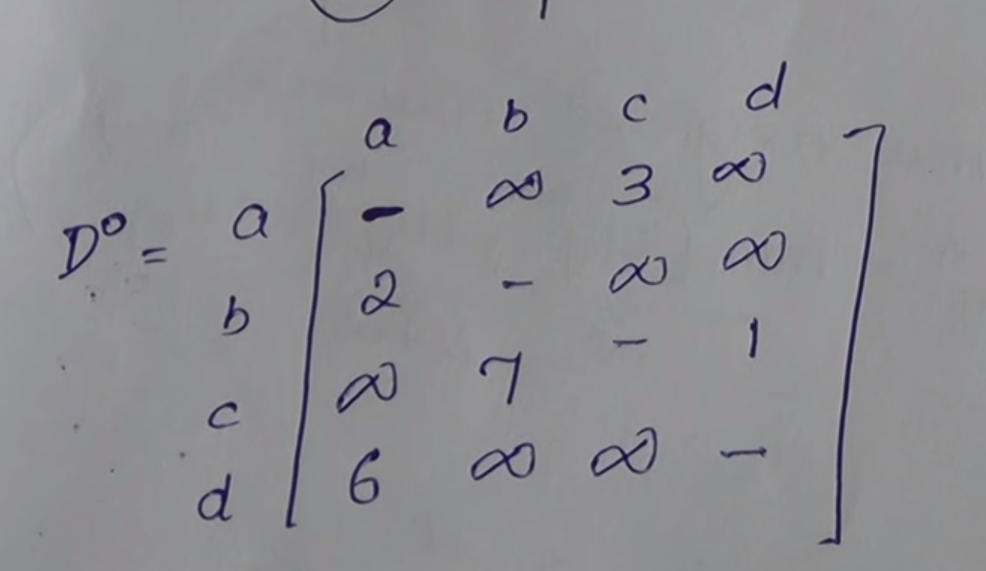


Рисунок 18 –Ручной расчет задачи. Шаг 1.

Элементы пересечения – пересчитываются. Пересчет происходит методом сложения весов ребер индексов. Т.е. выделенные элементы

Vbc = 3 + 2 = 5;

Vdc= 3 + 6 = 9.

Остальные элементы переписываются в матрицу с индексом 1 без изменений как показано на рисунке 19.

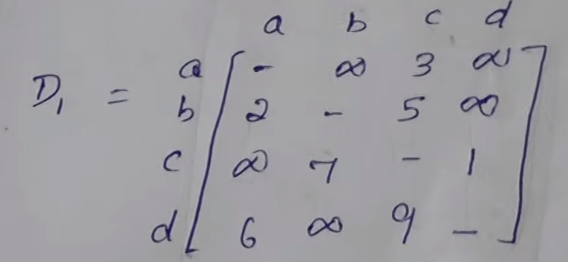


Рисунок 19 –Ручной расчет задачи. Шаг 2.

Далее повторяем операцию с 2 столбцом и второй строчкой, как показано на рисунке 20

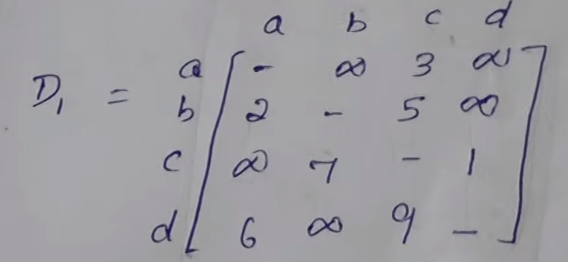


Рисунок 20 –Ручной расчет задачи. Шаг 2.

Vcа = 7 + 2 = 9

Остальные элементы переписываются в матрицу с индексом 2 без изменений, как показано на рисунке 21

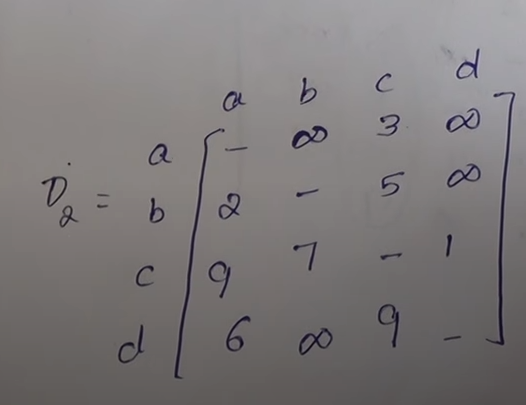


Рисунок 21 –Ручной расчет задачи. Шаг 3.

Далее повторяем операцию с третим столбцом и третий строчкой, как показано на рисунке 22.

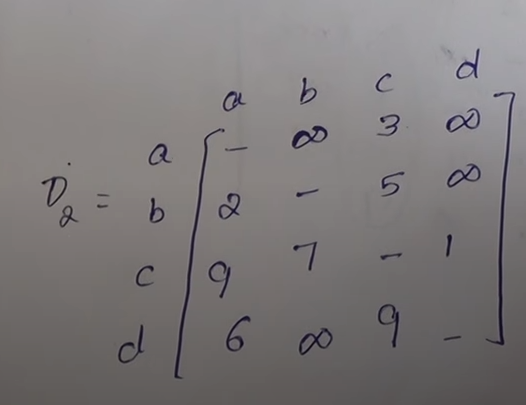


Рисунок 22 –Ручной расчет задачи. Шаг 3.

Vab = 3 + 7 = 10

Vad = 3 + 1 = 4

Vbd = 5 + 1 = 6

Vdb = 7 + 9 = 16

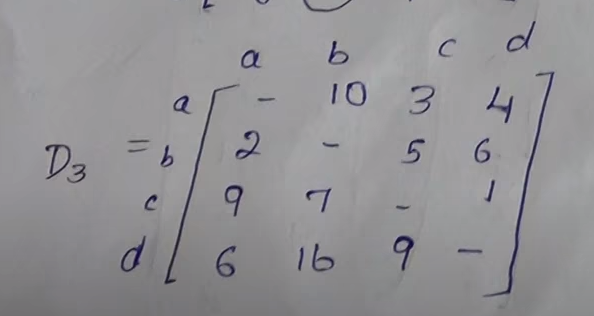
Остальные элементы переписываются в матрицу с индексом 3 без изменений, как показано на рисунке 23

Рисунок 23 –Ручной расчет задачи. Шаг 4.

Далее повторяем операцию с четвертым столбцом и четвертой строчкой, как показано на рисунке 24

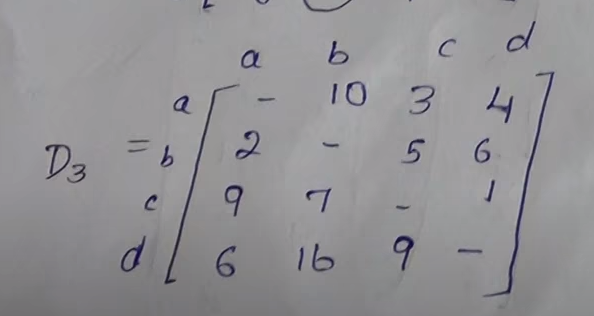


Рисунок 24 –Ручной расчет задачи. Шаг 4.

Vca = 6 + 1 = 7, 7<9, поэтому записываем 7.

Полученная матрица, будет являться матрицей кратчайших путей, найденных методом Флойда (Рис. 25).

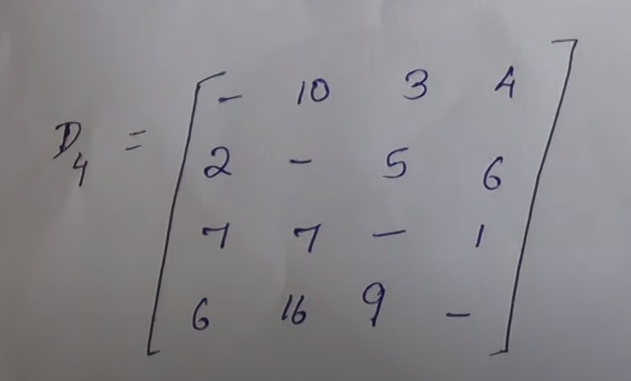


Рисунок 25 –Ручной расчет задачи. Полученная матрица кратчайших путей.

Таким образом, исходя из рисунка 26, можно сделать вывод, что программа работает верно.

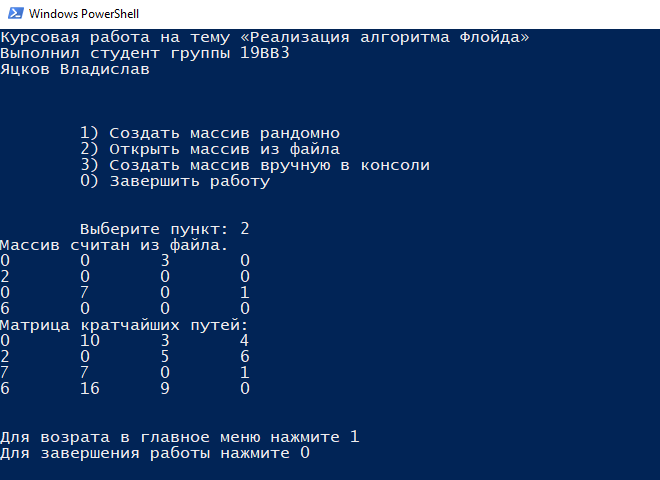


Рисунок 26– Вывод матрицы из файла.

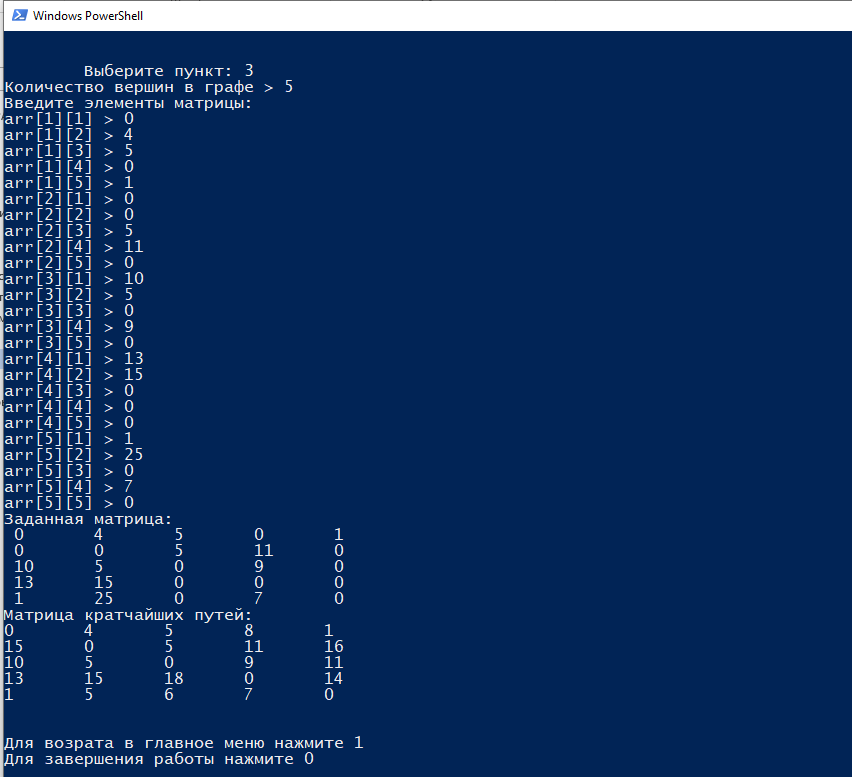


Рисунок 27 – Вывода матрицы сгенерированной вручную в консоли с указанием размера 5.

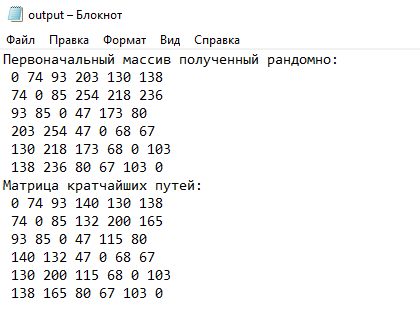


Рисунок 28– Тестирование функции «Запись в файл» из пункта 1

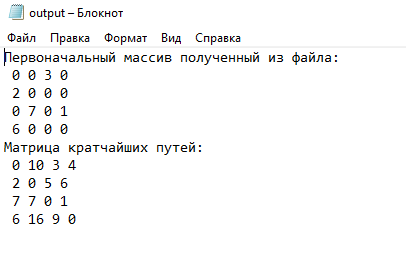


Рисунок 29– Тестирование функции «Запись в файл» из пункта 2

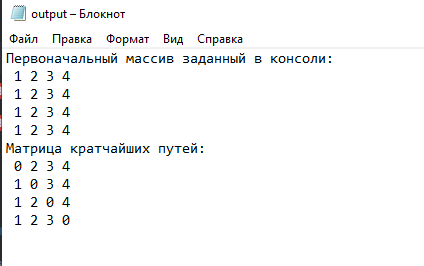


Рисунок 30– Тестирование функции «Запись в файл» из пункта 3

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Флойда-Уоршелла для поиска кратчайших путей в MicrosoftVisualStudio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц. Приобретены навыки по осуществлению алгоритма поиска кратчайших путей, работы с файлами, а также работы с командной строкой. Углублены знания языка программирования Cи.

Недостатком разработанной программы является невозможность работы с циклами с отрицательными весами.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

## Список литературы.

* 1. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978
  2. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.
  3. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.
  4. Татт У. Теория графов**; НФ "Пушкинская библиотека", АСТ Москва, АСТ -** Москва**, 2013. - 229**c.
  5. <https://www.cyberforum.ru/>
  6. <https://algowiki-project.org/ru/Алгоритм_Флойда-Уоршелла>
  7. https://e-maxx.ru/algo/floyd\_warshall\_algorithm

### 

## Приложение А.

**Листинг программы.**

Файл: header.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdlib.h>

#include <stdio.h>

#include <locale.h>

#include <string.h>

#include <stdio.h>

#include <time.h>

#include <windows.h>

#include <conio.h>

#include <queue>

#include <iostream>

#include <fstream>

#include <assert.h>

#include <math.h>

Файл: menu.h

#pragma once

int\* num;

int\*\* arr;

int i, j, n;

FILE\* f\_output;

FILE\* f\_input;

// для case2

int m = 0; // m - кол-во записей

int s, k = 0;

int usl = 0;// условие для перехода в другое меню

int ver; // вероятность

int max; // максимальный вес

Файл: main.h

#pragma once

#include"header.h"

#include "menu.h"

void FLOID(int\*\* arr, int V) // Алгоритм Флойда

{

int k;

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

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

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

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

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

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

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

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

{

for (j = 0; j < V; j++) printf("%d\t", arr[i][j]);

putchar('\n');

}

}

int\*\* dynamic(int count) // Создание динамического массива.

{

int\*\* mass;

mass = (int\*\*)malloc(count \* sizeof(int\*));

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

return mass;

}

void print(int\*\* mass, int count) //Вывод на экран

{

printf("");

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

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

printf("%d \t", mass[i][j]);

}

printf("\n");

}

}

void generate\_typeV\_route(int\*\* mass, int count) { //взвешенный ориентированный

int a; // вероятность

int b;

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

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

a = rand() % 101;

if (a < ver) {

mass[j][i] = rand() % max + 1;

mass[i][j] = 0;

}

else mass[j][i] = 0;

if (i == j) mass[j][i] = 0;

}

}

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

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

b = rand() % 101;

if ((b < 50) && (mass[j][i] >= 1)) {

mass[i][j] = mass[j][i];

mass[j][i] = 0;

}

if (i == j) mass[i][j] = 0;

}

}

}

void generate\_typenv(int\*\* mass, int count) { // взвешенный неориентированный

int a; // вероятность

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

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

a = rand() % 101;

if (a < ver) {

mass[j][i] = rand() % max + 1;

mass[i][j] = 0;

}

else mass[j][i] = 0;

if (i == j) mass[j][i] = 0;

}

}

}

Файл: menu.cpp

#include "header.h"

#include "main.h"

#include "menu.h"

int main(int argc, char\* argv[]) {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int begin;

int n = 0;

srand(time(NULL));

if (argc < 2) {

printf("Курсовая работа на тему «Реализация алгоритма Флойда» \n");

printf("Выполнил студент группы 19ВВ3:\nЯцков В.В\n\n");

printf("Вариант работы программы с помощью вызова параметров в командной строки \n\n");

printf("Введите параметры в командной строке через пробел.\n");

printf("Варианты параметров:\n");

printf("Ориентированный граф - route.\n");

printf("Неориентированный граф - nroute.\n");

printf("Размер массива - укажите число.\n");

printf("Вероятность появления ребра - укажите число.\n");

printf("Максимальный вес ребра - укажите число.\n");

return 1;

}

if ((argc >= 4) && (strcmp(argv[1], "route") == 0)) {

if (argc > 5) {

printf("Слишком много параметров. Попробуйте снова!\n");

return 1;

}

if (argv[2] != NULL) {

n = atoi(argv[2]);

}

else if (n == 0) {

printf("Число не обнаружено! Попробуйте снова!\n");

return (0);

}

if (argv[3] != NULL) {

ver = atoi(argv[3]);

}

else if (ver == 0) {

printf("Число не обнаружено! Попробуйте снова!\n");

return (0);

}

if (argv[4] != NULL) {

max = atoi(argv[4]);

}

else if (max == 0) {

printf("Число не обнаружено! Попробуйте снова!\n");

return (0);

}

arr = dynamic(n);

generate\_typeV\_route(arr, n);

printf("Матрица смежности взвешенного ориентированного графа:\n\n");

print(arr, n);

printf("Матрица кратчайших путей: \n");

FLOID(arr, n);

printf("Для перехода в режим работы консольного приложения программы введите 1\n");

scanf("%d", &usl);

if (usl == 1) {

printf("Основная часть программы!");

}

}

if ((argc >= 4) && (strcmp(argv[1], "nroute") == 0)) {

if (argc > 5) {

printf("Слишком много параметров(Больше 4). Попробуйте снова!\n");

return 1;

}

if (argv[2] != NULL) {

n = atoi(argv[2]);

}

else if (n == 0) {

printf("Число не обнаружено! Попробуйте снова!\n");

return (0);

}

if (argv[3] != NULL) {

ver = atoi(argv[3]);

}

else if (ver == 0) {

printf("Число не обнаружено! Попробуйте снова!\n");

return (0);

}

if (argv[4] != NULL) {

max = atoi(argv[4]);

}

else if (max == 0) {

printf("Число не обнаружено! Попробуйте снова!\n");

return (0);

}

arr = dynamic(n);

generate\_typenv(arr, n);

printf("Матрица смежности взвешенного неориентированного графа:\n\n");

print(arr, n);

printf("Матрица кратчайших путей: \n");

FLOID(arr, n);

printf("Для перехода в режим работы консольного приложения программы введите 1\n");

scanf("%d", &usl);

if (usl == 1) {

printf("Основная часть программы!");

}

}

bool exitPerformed = false;

unsigned int level = 1;

while (!exitPerformed) {

srand(time(NULL));

system("cls");

switch (level)

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

printf("Курсовая работа на тему «Реализация алгоритма Флойда» \n");

printf("Выполнил студент группы 19ВВ3\n");

printf("Яцков Владислав\n");

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

printf("\t1) Создать массив рандомно\n");

printf("\t2) Открыть массив из файла\n");

printf("\t3) Создать массив вручную в консоли\n");

printf("\t0) Завершить работу\n\n\n");

printf("\tВыберите пункт: ");

unsigned int item = 0;

scanf("%u", &item);

switch (item) {

case 1: {

f\_output = fopen("output.txt", "w");

fprintf(f\_output, "Первоначальный массив:\n");

printf("Выберите размер массива:\n\n\n");

scanf("%d", &n);

num = (int\*)malloc(n \* sizeof(int));

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

num[i] = 0;

}

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

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

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

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

if (j > i) {

arr[i][j] = rand() % 256;

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

}

else if (i == j) arr[i][j] = 0;

}

}

printf("\nЗаданный массив: \n");

printf("");

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

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

printf("%d \t", arr[i][j]);

fprintf(f\_output, " %d", arr[i][j]);

}

printf("\n");

fprintf(f\_output, "\n");

}

{

printf("Матрица кратчайших путей: \n");

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

FLOID(arr, n);

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

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

fprintf(f\_output, " %d", arr[i][j]);

}

fprintf(f\_output, "\n");

}

}

fclose(f\_output);

printf("\n\nДля возрата в главное меню нажмите 1\nДля завершения работы нажмите 0\n");

scanf("%d", &usl);

if (usl == 1) {

break;

}

return (0);

}

case 2: {

// условие, при котором удалось открыть исходный файл

f\_input = fopen("input.txt", "r");

f\_output = fopen("output.txt", "w");

fprintf(f\_output, "Первоначальный массив:\n");

// считывание данных с файла

// если переменная равна 'А' => удалось открыть файл

while ((fscanf(f\_input, "%d", &s) != EOF)) {

if (!f\_input) break;

m += 1;

}

rewind(f\_input); // перематываем файл для повторного чтения

n = sqrt(m);

arr = dynamic(n);

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

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

fscanf\_s(f\_input, "%d", &arr[i][j]);

}

}

printf("Массив считан из файла.\n");

print(arr, n);

fclose(f\_input);

printf("Матрица кратчайших путей: \n");

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

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

fprintf(f\_output, " %d", arr[i][j]);

}

fprintf(f\_output, "\n");

}

FLOID(arr, n);

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

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

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

fprintf(f\_output, " %d", arr[i][j]);

}

fprintf(f\_output, "\n");

}

fclose(f\_output);

printf("\n\nДля возрата в главное меню нажмите 1\nДля завершения работы нажмите 0\n");

scanf("%d", &usl);

if (usl == 1) {

break;

}

return (0);

}

case 3: {

f\_output = fopen("output.txt", "w");

fprintf(f\_output, "Первоначальный массив:\n");

printf("Количество вершин в графе > ");

scanf("%d", &n);

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

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

printf("Введите элементы матрицы: \n");

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

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

{

printf("arr[%d][%d] > ", i + 1, j + 1);

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

}

printf("Заданная матрица:\n");

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

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

printf(" %d\t", arr[i][j]);

fprintf(f\_output, " %d", arr[i][j]);

}

printf("\n");

fprintf(f\_output, "\n");

}

printf("Матрица кратчайших путей: \n");

FLOID(arr, n);

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

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

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

fprintf(f\_output, " %d", arr[i][j]);

}

fprintf(f\_output, "\n");

}

fclose(f\_output);

printf("\n\nДля возрата в главное меню нажмите 1\nДля завершения работы нажмите 0\n");

scanf("%d", &usl);

if (usl == 1) {

break;

}

return (0);

}

case 0:

return(0);

}

}

}