Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовой работе

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация алгоритма Форда-Фалкерсона»

Выполнил:

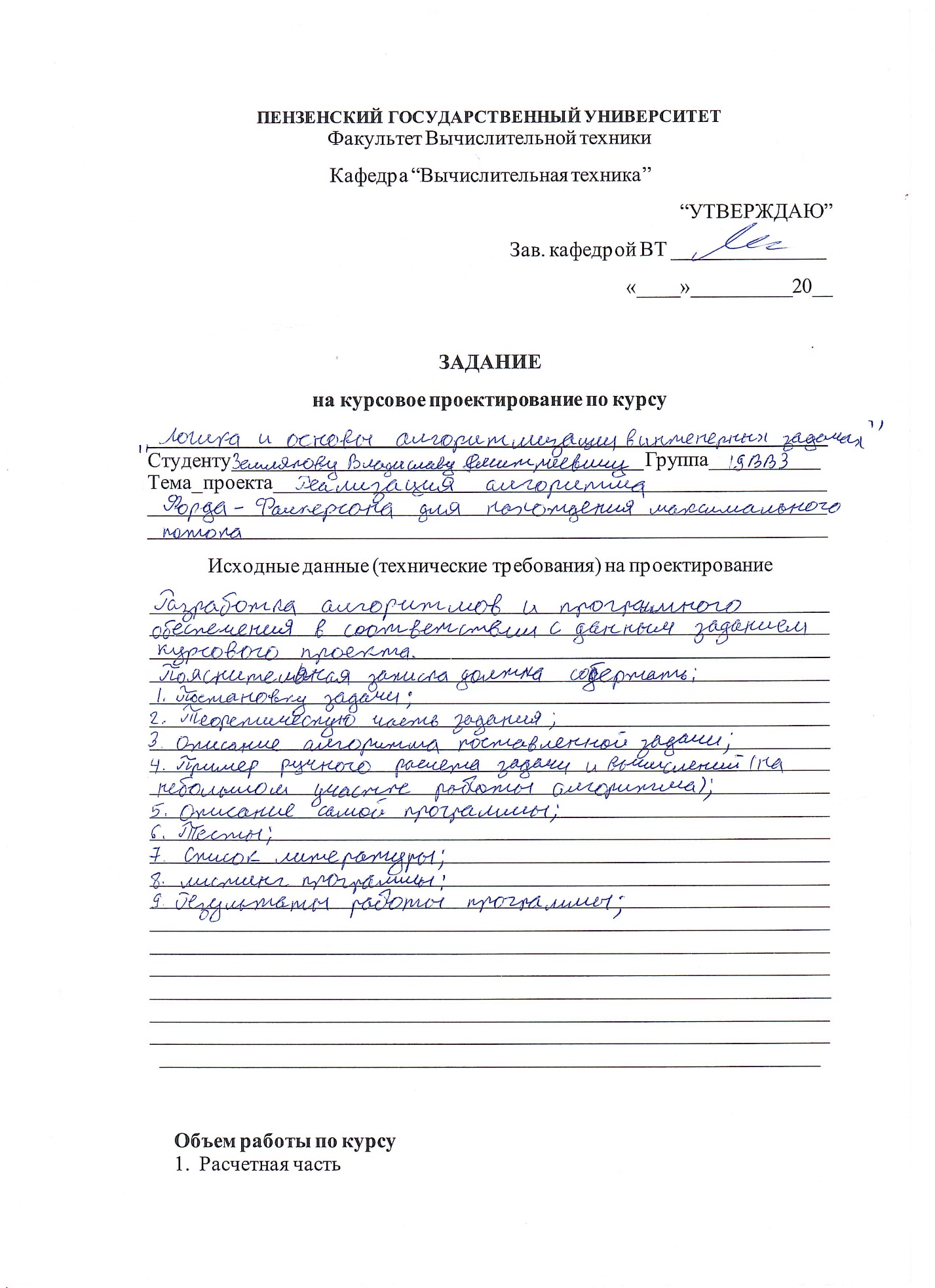
Студент группы 19ВВ3

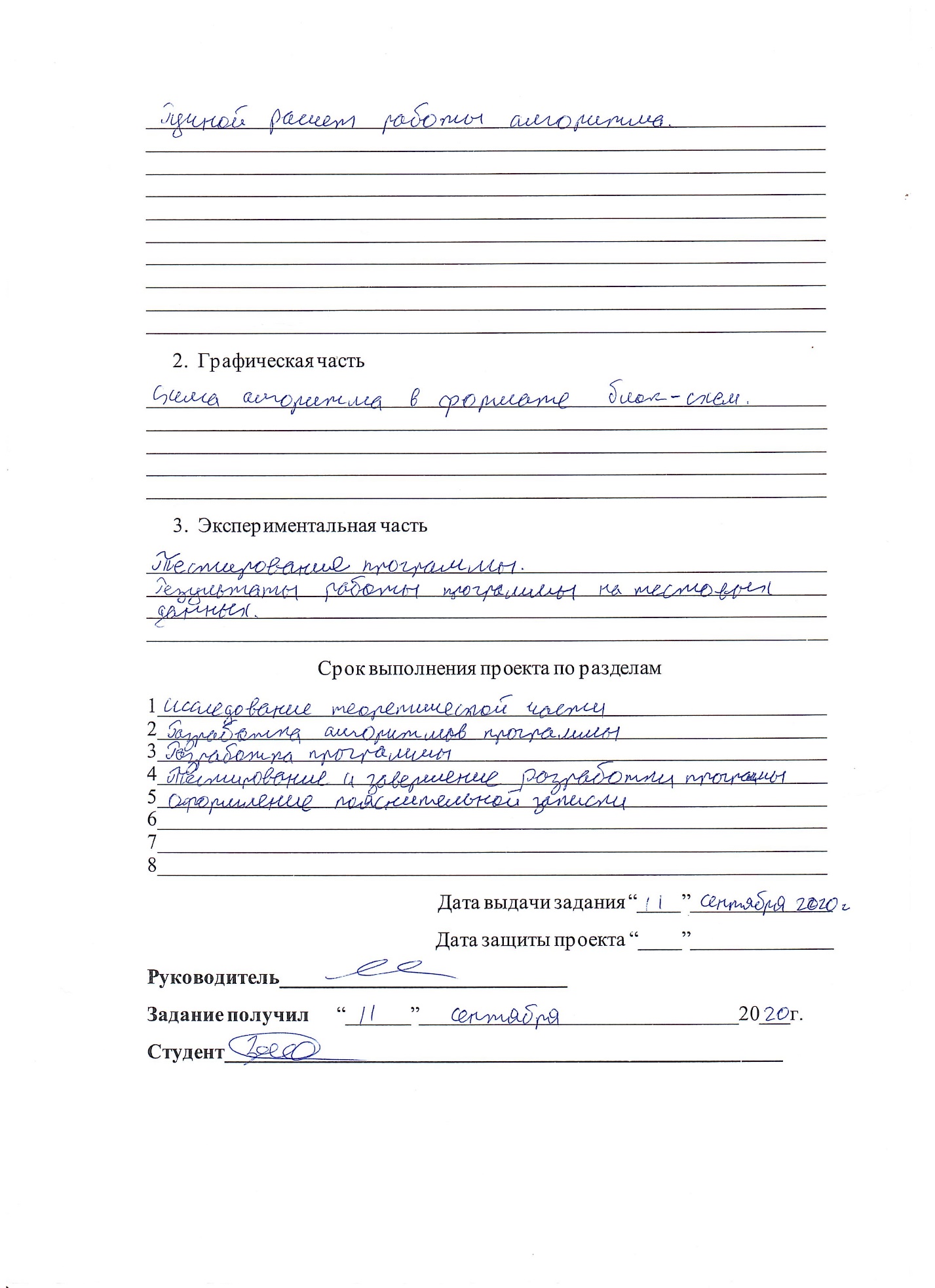
Земляков В.Д.

Принял:

Митрохин М.А.

Пенза 2020





Содержание

**Оглавление**

[**Реферат** 5](#_Toc59175875)

[**Введение** 6](#_Toc59175877)

[**Постановка задачи** 7](#_Toc59175879)

[**1 Теоретическая часть** 8](#_Toc59175880)

[**2 Описание алгоритма программы** 9](#_Toc59175881)

[**3 Пример ручного просчёта задачи** 11](#_Toc59175882)

[**4 Описание программы** 12](#_Toc59175883)

[**6 Тестирование программы** 19](#_Toc59175884)

[**Заключение** 25](#_Toc59175886)

[**Список литературы** 26](#_Toc59175887)

[**Приложение А** 27](#_Toc59175892)

[**Листинг** 27](#_Toc59175893)

[**Header.h** 27](#_Toc59175894)

[**KURS.cpp** 27](#_Toc59175896)

[**menu.cpp** 29](#_Toc59175898)

**Реферат**

**Отчет 32 стр, 16 рисунков.**

Цель исследования – разработка программы, выполняющей алгоритм Форда-Фалкерсона.

В работе рассмотрен способ нахождения максимального потока в сети.

**Введение**

**Алгоритм Форда** — **Фалкерсона** решает [задачу нахождения максимального потока](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE_%D0%BC%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC_%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%B5) в [сети](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BF%D0%BE%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C).

Идея алгоритма заключается в следующем. Выбираем такой путь от источника к стоку, чтобы для каждого ребра остаточная пропускная способность была строго больше нуля. При этом ребра на данном пути могут проходиться как в прямом, так и в обратном направлении. Выбираем минимальное значение среди остаточных пропускных способностей ребер данного пути. Увеличиваем поток на каждом из ребер данного пути на выбранное минимальное значение. Далее ищем следующий аналогичный путь.

Работа алгоритма продолжается до тех пор, пока удается находить данные пути.

Данный алгоритм относится к классу недетерминированных, т.е. каждый следующий шаг алгоритма определен неоднозначно. И время работы (количество шагов) алгоритма зависит от того, как будут выбираться шаги

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudio2010, язык программирования – Си и Си++.

Целью данной курсовой работы является разработка программы на языке Си и С++, которые являются широко используемым. Именно с их помощью в данном курсовом проекте реализуется алгоритм Форда-Фалкерсона, осуществляющий поиск максимального потока в сети.

**Постановка задачи**

Требуется разработать программу, которая осуществит нахождение максимального потока графа, используя алгоритм Форда-Фалкерсона.

Исходный граф в программе должен задаваться взвешенной и ориентированной матрицей смежности. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После этого он вводит начальную и конечную вершину, между которыми программа найдёт максимальный поток между ними, результат выводится на экран. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройства ввода- клавиатура и мышь.

**1 Теоретическая часть**

Граф G (рисунок 1) задается множеством вершин X1, X2, ..., Xn. и

множеством ребер , соединяющих между собой определенные вершины. Ребра

из множества А ориентированы, что показывается стрелкой, которая указывает достижимость данной вершины, граф с такими ребрами называется

ориентированным графом. Так же граф может иметь вес рёбер.

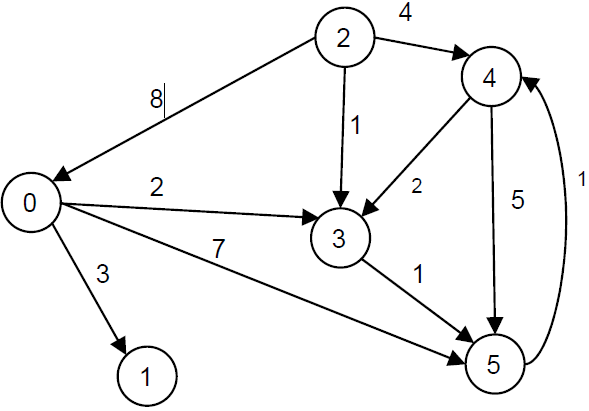


Рисунок 1-Пример графа

При представлении графа матрицей смежности информация о ребрах

графа хранится в квадратной матрице, где присутствие пути из одной вершины

в другую обозначается весом ребра, иначе нулем.

Идея алгоритма заключается в следующем. Изначально величине потока присваивается значение 0: f(u,v)=0 для всех u,v из V. Затем величина потока итеративно увеличивается посредством поиска увеличивающего пути (путь от источника s к стоку t, вдоль которого можно послать ненулевой поток).

**2 Описание алгоритма программы**

Для программной реализации алгоритма понадобится четыре массива: ArrayM1(int) - хранит матрицу смежности, visited (int) – содержит номера посещённых вершин, rArray(int) – включает в себя остаточные мощности, parent(int) – сохраняет в себе путь. Итак, имеется граф G=(V, E). Каждая из вершин входящих во множество V, изначально отмечена как не посещенная, т. е. элементам массива visited присвоено значение 0.

Создаем остаточный граф rArray(int) и заполняем его. Создаём переменную потока max\_flow и обнуляем её. Находим минимальную остаточную ёмкость (path\_flow) рёбер. Путь заполнен с помощью BFS. Обновляем остаточные ёмкости рёбер и обратных рёбер. Добавляем поток к общему пути. В конце работы функция возвращает значение максимального потока.

Ниже представлен псевдокод функции FF.

**Вход**: Матрица смежности Array графа G, источник s и сток t

**Выход:** Максимальный поток max\_flow из s в t

**Алгоритм ПОШ**

1. Max\_flow=0
2. rArray=Array
3. Пока существует путь выполнять
4. Path\_flow=INF
5. Для всех вершин v неравных s выполнить
   1. path\_flow = fmin(path\_flow, rArray[u][v])
6. Для всех вершин v неравных s выполнить
   1. rArray[u][v] -= path\_flow
   2. rArray[v][u] += path\_flow
7. max\_flow += path\_flow

**3 Пример ручного просчёта задачи**

Пусть задан следующий граф.

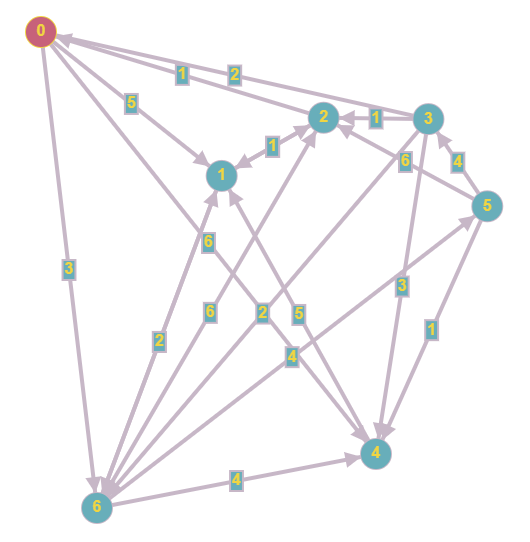


Рисунок 2- Граф для ручного расчёта.

Рассмотрим пути из 0 в 6.

Существует такие пути как: 0→6, 0→1→6 и 0→3→6

Рассмотрим первый путь. Минимальный поток по этому пути будет равен весу ребра, то есть 3.

Второй путь состоит из 2 рёбер. Вес первого равен 5, а второго 4. Поток проходящий по этому пути будет равняться 4.

Веса рёбер на третьем пути равны 2 и 2. Минимальное число 2. Потоком этого пути является 2.

Складывая потоки этих путей, получаем максимальный поток сети.

3+4+2=9

**4 Описание программы**

Для написания данной программы использованы языки программирования Си и Си++.

Проект был создан в виде консольного приложения в программе Visual studio 2019.

Данная программа является многомодульной, поскольку состоит из нескольких файлов: KURS.cpp, menu.cpp, Header.h.

Файл Header.h содержит библиотеки и объявление некоторых функций.

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_CRT\_NONSTDC\_NO\_DEPRECATE

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <conio.h>

#include <locale.h>

#include <time.h>

#include <queue>

#include <iostream>

void output(int\*\* Array, int N);

int GenWeighedOriented(int\*\* Array, int N);

int BFS(int\*\* Array, int s, int t, int\* parent, int N);

int FF(int\*\* Array, int s, int t, int N);

int Save(int\*\* Array,int result,int s,int t, int N);

KURS.cpp включает в себя функции работы с матрицами и файлами.

menu.cpp осуществляет вывод меню в консоль и обработку событий при выборе пункта меню.

В программе имеются следующие функции: output, GenWeighedOriented, BFS, FF, Save, print\_menu, selected, button\_process, menu, main.

Работа программы начинается с вывода в консоли меню.

int print\_menu(int index) {

setlocale(LC\_ALL, "Rus");

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

if (index > 4 || index < 0) return 0;

(index == 1) ? printf(" |\_\_\_\_\_\_Создание\_\_\_\_\_|\n") : printf(" | Создание |\n");

(index == 2) ? printf(" |\_\_\_\_\_\_Алгоритм\_\_\_\_\_|\n") : printf(" | Алгоритм |\n");

(index == 3) ? printf(" |\_\_\_\_\_Сохранение\_\_\_\_|\n") : printf(" | Cохранение |\n");

(index == 4) ? printf(" |\_\_\_\_\_\_\_Выход\_\_\_\_\_\_\_|\n") : printf(" | Выход |\n");

printf(" |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

}

Где пользователь выбирает один из пунктов: Создание, Алгоритм, Сохранение или Выход.

Если пользователь выбирает пункт “Cоздание”, то программа попросит ввести его размерность, после чего сгенерируется и выведется матрица смежности взвешенного неориентированного графа.

int GenWeighedOriented(int\*\* Array, int N) {

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

{

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

{

int z = rand() % 100;

if (z > 50) {

Array[i][j] = rand() % N;

}

else {

Array[i][j] = 0;

}

if (i == j) {

Array[i][j] = 0;

}

}

}

output(Array, N);

return \*\*Array;

}

void output(int\*\* Array, int N) {

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

{

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

{

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

}

printf("\n");

}

}

При выборе пункта “Алгоритм”, программа попросит пользователя ввести вершину источника и стока. В зависимости от вершин и наличия сгенерированной матрицы программа выдаст различные результаты.

system("cls");

if (N == 0) {

printf("Пожалуйста, сгенерируйте матрицу.");

}

else {

output(ArrayM1, N);

printf("Введите исходную вершину:");

scanf("%d", &s);

printf("Введите конечную вершину:");

scanf("%d", &t);

if (s == t) {

printf("Введите другие числа!");

}

else {

result = FF(ArrayM1, s, t, N);

if (result == 0) {

printf("Путей нет!");

}

else

printf("Максимальный возможный поток:%d", result);

}

}

“Сохранение” выполняет запись результатов работы программы в текстовый файл.

FILE\* fp;

char name[] = "result.txt";

if ((fp = fopen(name, "w+")) == NULL)

{

printf("Не удалось открыть файл");

getchar();

return 0;

}

else {

printf("Файл успешно сохранён");

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

{

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

{

fprintf(fp, "%d\t", Array[i][j]);

}

fprintf(fp,"\n");

}

if (result == 0) {

fprintf(fp,"Путей нет!");

}

else

fprintf(fp,"Максимальный возможный поток из %d в %d:%d", s,t, result);

}

return 0;

И последний пункт “Выход”, завершает работу программы с кодом 0.

exit(0);

Ниже можно увидеть оформление после запуска программы и его последующие изменения, после выбора соответствующих пунктов.

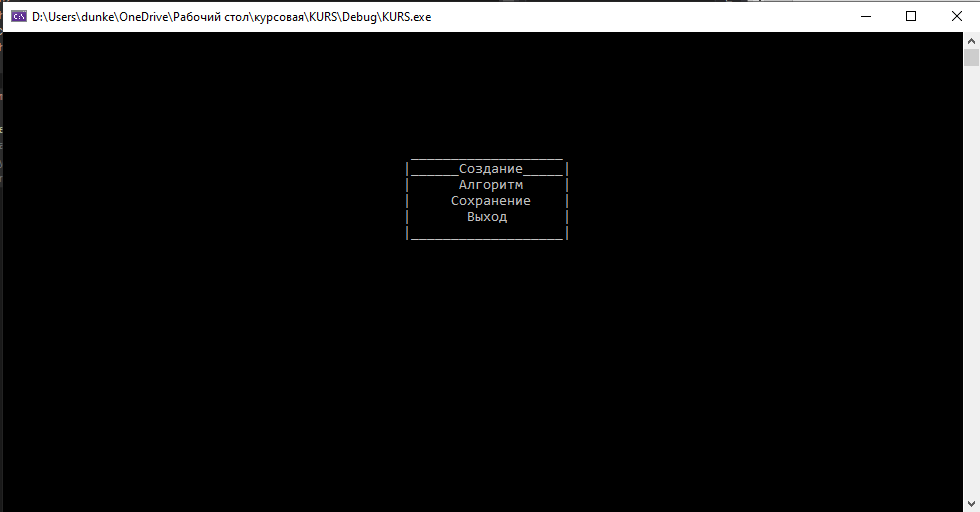


Рисунок 3- Описание работы программы. Вывод меню.

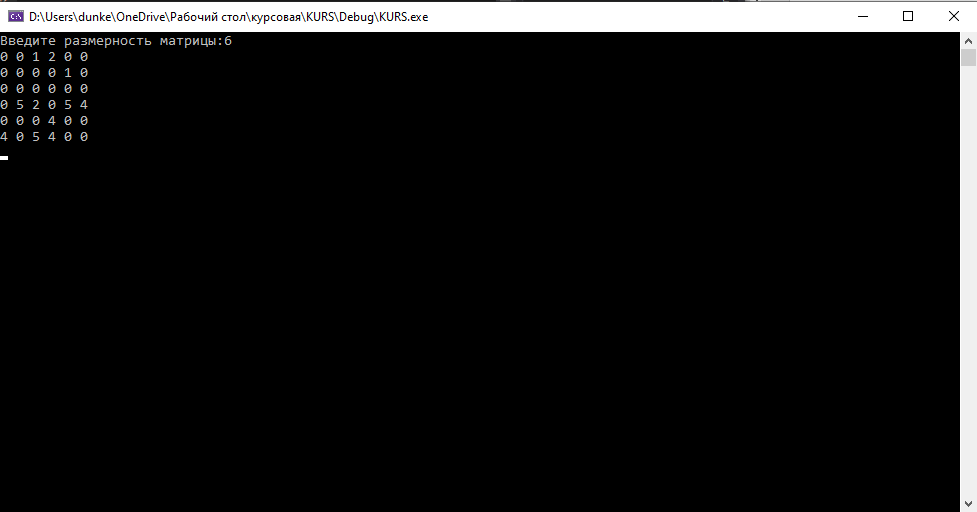


Рисунок 4-Описание работы программы. Генерация матрицы

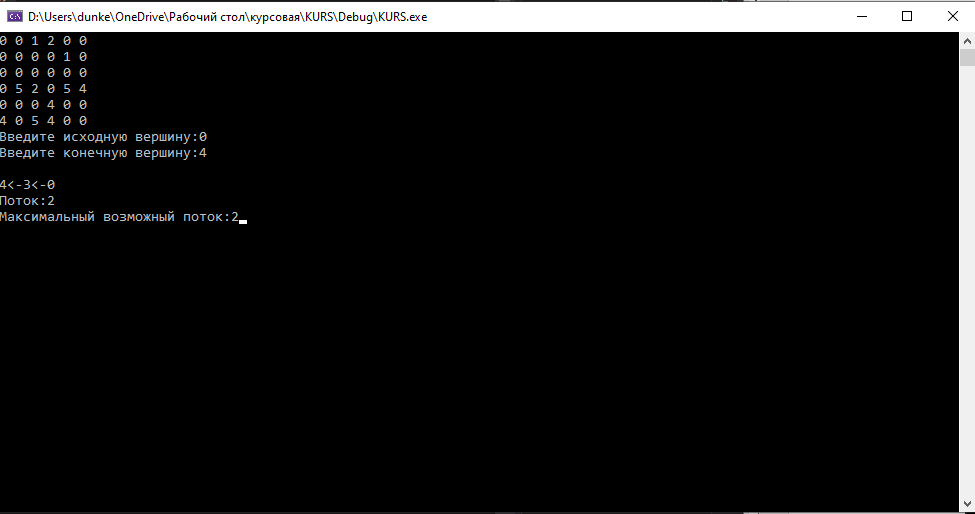


Рисунок 5-Описание работы программы. Результат работы алгоритма.

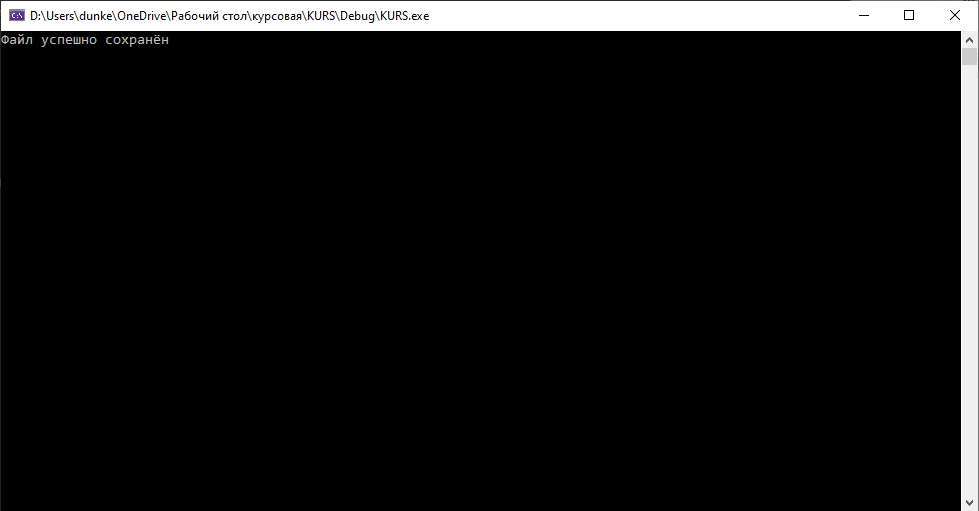


Рисунок 6-Описание работы программы. Сохранение файла

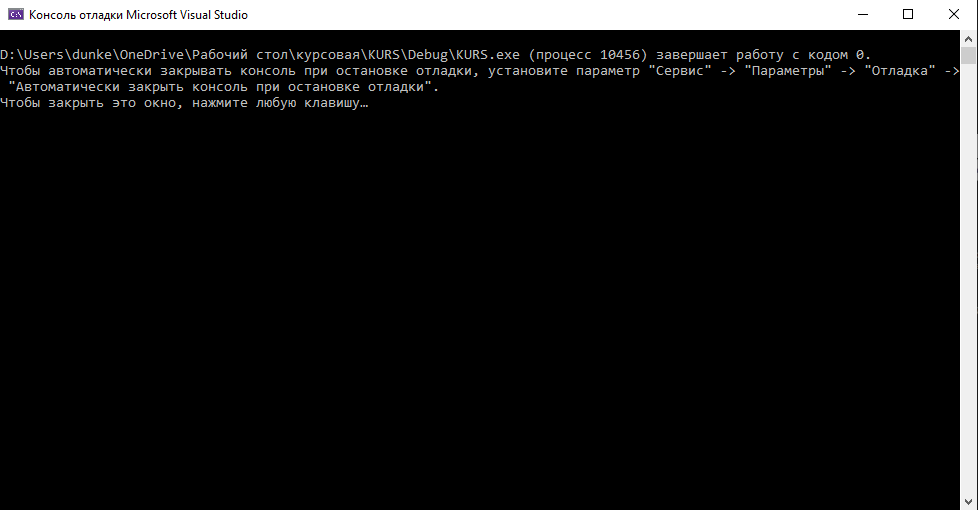


Рисунок 7- Описание работы программы. Выход из программы.

**6 Тестирование программы**

Среда разработки MicrosoftVisualStudio 2010 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.

Таблица 1- Описание поведения программы при тестировании

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер  теста | Описание теста | Последовательность действий(как достичь результата) | Ожидаемый результат. | Полученный результат |
| 1 | Запуск программы | Запустить «KURS.exe» | Вывод в консоль меню | Верно |
| 2 | Генерация матрицы | Выбор пункта меню «Создание» | Вывод сгенерированной матрицы смежности. | Верно |
| 3 | Ошибка при выборе пункта «Алгоритм» | Выбрать пункт «Алгоритм» с отсутствующей матрицей | Вывод сообщения в консоль «Пожалуйста, сгенерируйте матрицу» | Верно |
| 4 | Ошибка при введении вершин | Ввести одну и ту же вершину | Вывод сообщения в консоль «Введите другие числа» | Верно |
| 5 | Отсутствие путей | Ввести вершины, между которыми нет связей | Вывод сообщения в консоль «Путей нет!» | Верно |
| 6 | Корректный вывод путей | Ввести вершины, между которыми существуют связи | Вывод в консоль  Путей, их потоков, и максимальных потоков | Верно |
| 7 | Сохранение в файл | Выбор пункта меню «Сохранение» | Создание файла, содержащего результаты работы программы | Верно |
| 8 | Выход из программы | Выбор пункта меню «Выход» | Завершение работы программы с кодом 0 | Верно |
| 9 | Результат работы программы | Ввести вершины, которые дадут путь | Совпадение результатов работы программы с ручными рассчётами | Верно |

Ниже представлены результаты тестирования. Изначально после запуска программы выведется меню в консоль.

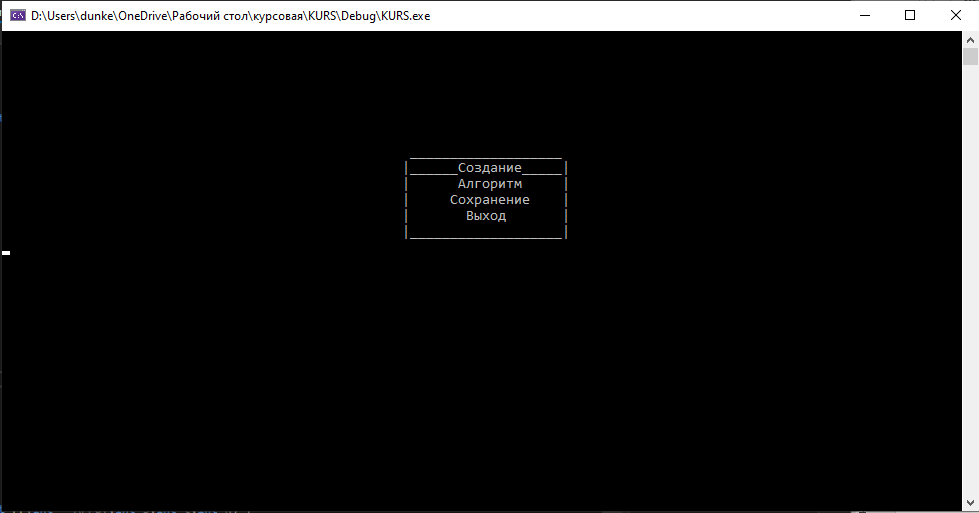


Рисунок 8- Результат 1 теста

После выбора Пункта «Создание» программа предложит пользователю ввести размерность матрицы смежности взвешенного ориентированного графа. После чего она будет сгенерирована и выведена на экран.

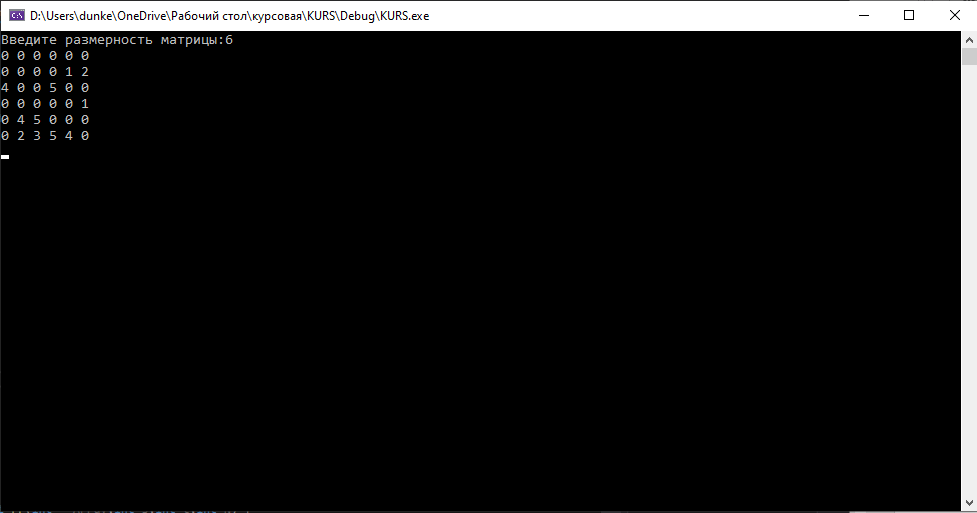


Рисунок 9-Результат работы 2 теста

Если же пользователь не задаст размер матрицы и выберет пункт «Алгоритм», то программа выведет ошибку и попросит его сгенерировать матрицу.

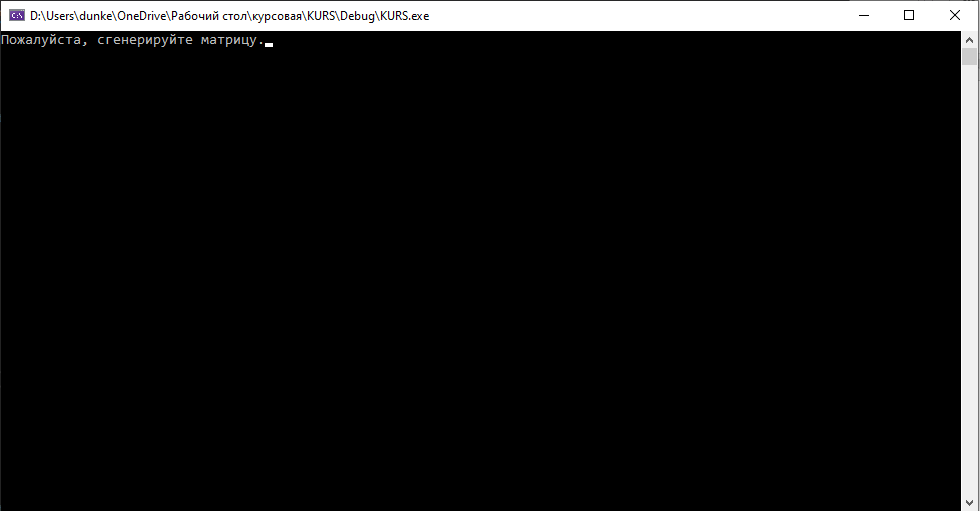


Рисунок 10- Результат работы 3 теста

Из-за того, что в программе не предусмотрено наличие петлей, после получения на выход одной и той же вершины, в консоль будет выведена просьба о введении других.

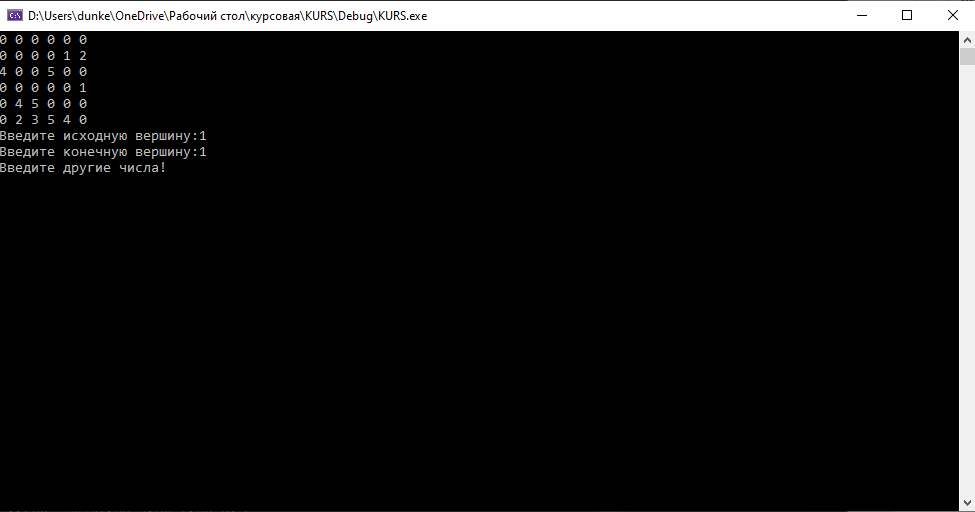


Рисунок 11- Результат работы 4 теста

Если же всё введено правильно, но связей между вершинами нет, программа выведет сообщение об этом.

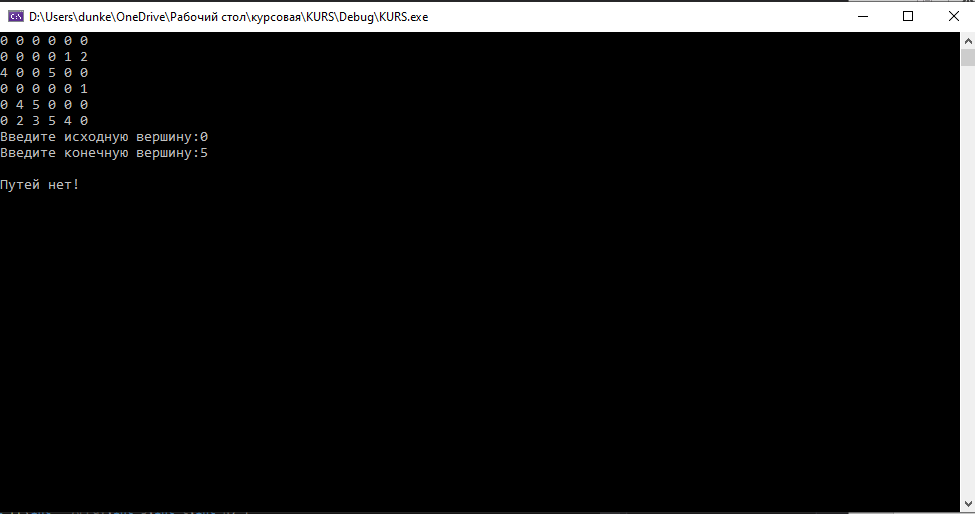


Рисунок 12- Результат работы 5 теста

При наличии между вершинами связей программа выведет пути, их потоки, и максимальный поток.

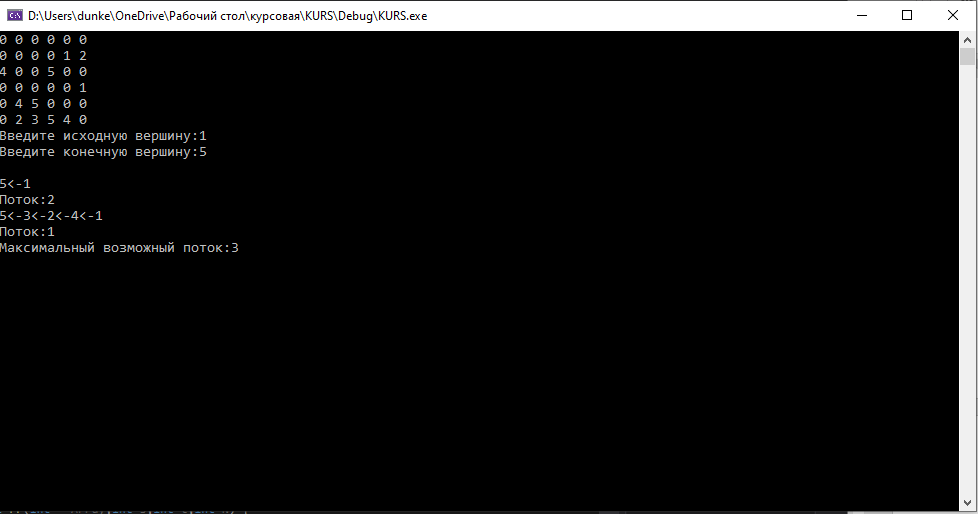


Рисунок 13- Результат работы 6 теста

Когда пользователь выбирает пункт «Сохранение» программа записывает результаты в файл, если его не существует, то создаёт.

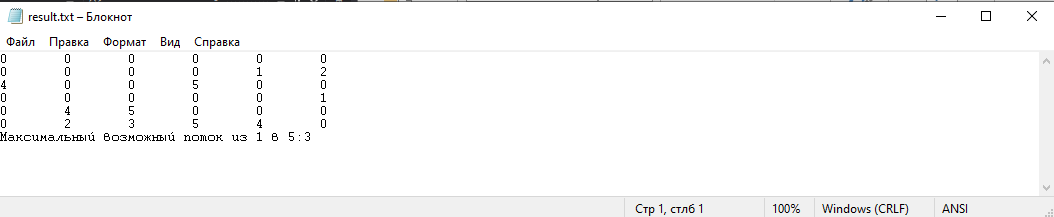
****

Рисунок 14- Результат работы 7 теста

При выборе пункта «Выход» программа завершит работу с кодом 0.

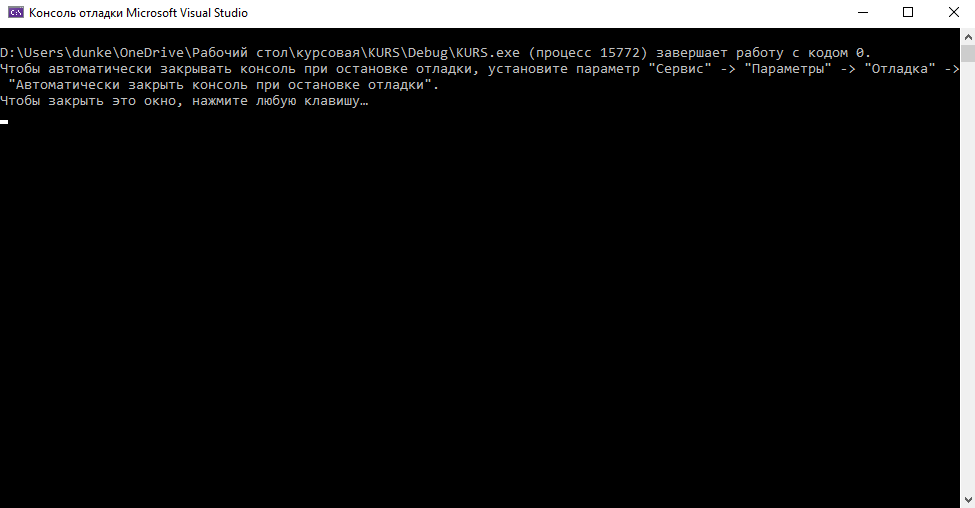


Рисунок 15- Результат работы 8 теста

Сгенерировали граф, выполнили алгоритм и сравнили его с ручными рассчётами.

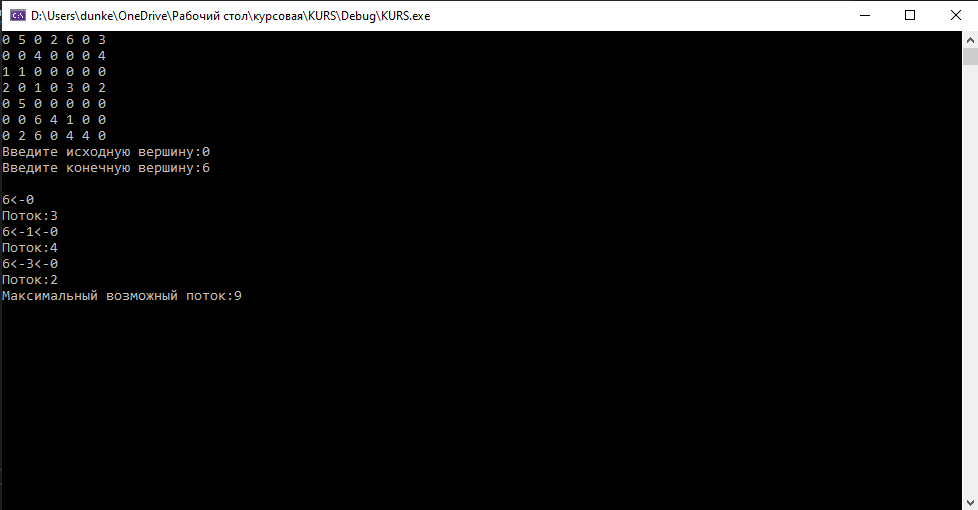


Рисунок 16- Проверка ручного расчёта.

Результаты работы программы совпадают с ручными расчётами, следовательно программа работает корректно.

**Заключение**

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Форда-Фалкерсона для поиска максимального потока в MicrosoftVisualStudio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей. Приобретены навыки по осуществлению алгоритма Форда-Фалкерсона. 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. Т*омас Х. Кормен и др.* Алгоритмы: построение и анализ = INTRODUCTION TO ALGORITHMS. — 2-е изд. — М.: «Вильямс», 2006. — С. 1296. — ISBN 0-07-013151-1
2. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006
3. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.
4. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ - М.: МЦНМО, 2001. - 960 с.

**Приложение А**

**Листинг**

**Header.h**

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_CRT\_NONSTDC\_NO\_DEPRECATE

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <conio.h>

#include <locale.h>

#include <time.h>

#include <queue>

#include <iostream>

void output(int\*\* Array, int N);

int GenWeighedOriented(int\*\* Array, int N);

int BFS(int\*\* Array, int s, int t, int\* parent, int N);

int FF(int\*\* Array, int s, int t, int N);

int Save(int\*\* Array,int result,int s,int t, int N);

**KURS.cpp**

#include "Header.h"

void output(int\*\* Array, int N) {

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

{

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

{

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

}

printf("\n");

}

}

int GenWeighedOriented(int\*\* Array, int N) {

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

{

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

{

int z = rand() % 100;

if (z > 50) {

Array[i][j] = rand() % N;

}

else {

Array[i][j] = 0;

}

if (i == j) {

Array[i][j] = 0;

}

}

}

output(Array, N);

return \*\*Array;

}

int BFS(int\*\* Array, int s, int t, int\* parent, int N) {

int\* visited = NULL;

visited = (int\*)malloc(N \* sizeof(int));

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

visited[i] = 0;

}

std::queue <int> Q;

Q.push(s);

visited[s] = 1;

parent[s] = -1;

while (!Q.empty()) {

int u = Q.front();

Q.pop();

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

if (visited[i]==0 && Array[u][i]>0) {

Q.push(i);

parent[i] = u;

visited[i] = 1;

}

}

}

printf("\n");

return (visited[t] == 1);

}

int FF(int \*\*Array,int s,int t,int N) {

int\*\* rArray = NULL;

int u, v;

int max\_flow = 0;

rArray = (int\*\*)malloc(N \* sizeof(int\*));

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

{

rArray[i] = (int\*)malloc(N \* sizeof(int));

}

for (u = 0; u < N; u++) {

for (v = 0; v < N; v++) {

rArray[u][v] = Array[u][v];

}

}

int\* parent = NULL;

parent = (int\*)malloc(N \* sizeof(int));

while (BFS(rArray, s, t, parent,N)) {

int path\_flow = INT\_MAX;

for (v = t; v != s; v = parent[v]) {

u = parent[v];

printf("%d<-", v);

path\_flow = fmin(path\_flow, rArray[u][v]);

}

printf("%d", s);

printf("\nПоток:%d", path\_flow);

for (v = t; v != s; v = parent[v]){

u = parent[v];

rArray[u][v] -= path\_flow;

rArray[v][u] += path\_flow;

}

max\_flow += path\_flow;

}

return max\_flow;

}

int Save(int \*\*Array,int result, int s, int t,int N) {

FILE\* fp;

char name[] = "result.txt";

if ((fp = fopen(name, "w+")) == NULL)

{

printf("Не удалось открыть файл");

getchar();

return 0;

}

else {

printf("Файл успешно сохранён");

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

{

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

{

fprintf(fp, "%d\t", Array[i][j]);

}

fprintf(fp,"\n");

}

if (result == 0) {

fprintf(fp,"Путей нет!");

}

else

fprintf(fp,"Максимальный возможный поток из %d в %d:%d", s,t, result);

}

return 0;

}

**menu.cpp**

#include "Header.h"

int\*\* ArrayM1 = NULL;

int N = 0;

int s = 0, t = 0;

int result = 0;

static int value = 1;

int print\_menu(int index) {

setlocale(LC\_ALL, "Rus");

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

if (index > 4 || index < 0) return 0;

(index == 1) ? printf(" |\_\_\_\_\_\_Создание\_\_\_\_\_|\n") : printf(" | Создание |\n");

(index == 2) ? printf(" |\_\_\_\_\_\_Алгоритм\_\_\_\_\_|\n") : printf(" | Алгоритм |\n");

(index == 3) ? printf(" |\_\_\_\_\_Сохранение\_\_\_\_|\n") : printf(" | Cохранение |\n");

(index == 4) ? printf(" |\_\_\_\_\_\_\_Выход\_\_\_\_\_\_\_|\n") : printf(" | Выход |\n");

printf(" |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

}

void selected(int button) {

system("cls");

if (button == 1) {

system("cls");

printf("Введите размерность матрицы:");

scanf("%d", &N);

ArrayM1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

ArrayM1[i] = (int\*)malloc(N \* sizeof(int));

}

GenWeighedOriented(ArrayM1,N);

}

else if (button == 2) {

system("cls");

if (N == 0) {

printf("Пожалуйста, сгенерируйте матрицу.");

}

else {

output(ArrayM1, N);

printf("Введите исходную вершину:");

scanf("%d", &s);

printf("Введите конечную вершину:");

scanf("%d", &t);

if (s == t) {

printf("Введите другие числа!");

}

else {

result = FF(ArrayM1, s, t, N);

if (result == 0) {

printf("Путей нет!");

}

else

printf("Максимальный возможный поток:%d", result);

}

}

}

else if (button == 3) {

system("cls");

Save(ArrayM1,result,s,t,N);

}

else if (button == 4) {

exit(0);

}

getch();

}

int button\_process() {

char ch = '0';

while (!kbhit());

ch = getch();

switch (ch) {

case '1':

value = ch - 0x30;

selected(value);

break;

case '2':

value = ch - 0x30;

selected(value);

break;

case '3':

value = ch - 0x30;

selected(value);

break;

case '4':

value = ch - 0x30;

selected(value);

break;

}

if (ch == 72) {

//Arrow up

value -= 1;

if (value == 0) value = 4;

}

else if (ch == 80) {

//Arrow down

value += 1;

if (value == 5) value = 1;

}

else if (ch == 13) {

// Enter button

selected(value);

}

return value;

}

int menu(){

int button = 1;

while (1) {

print\_menu(button);

button = button\_process();

system("cls");

}

system("PAUSE");

}

int main() {

srand(time(NULL));

setlocale(LC\_ALL, "RUS");

menu();

}