Министерство образования Российской федерации  
Пензенский государственный университет  
Кафедра “Вычислительная техника”

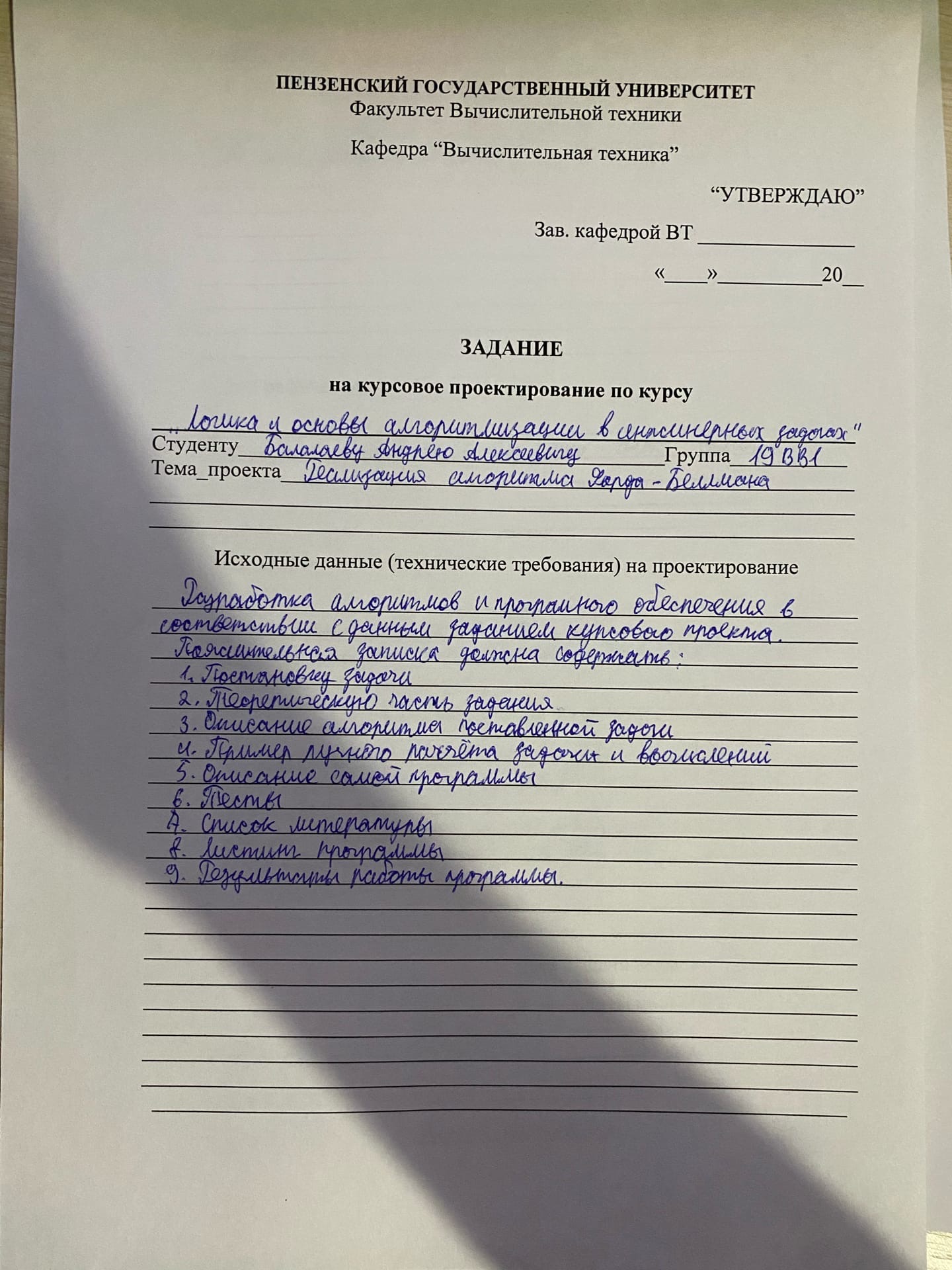
**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**К курсовому проектированию  
по курсу«Логика и основы алгоритмизации  
в инженерных задачах»  
на тему «Алгоритм Форда-Беллмана»

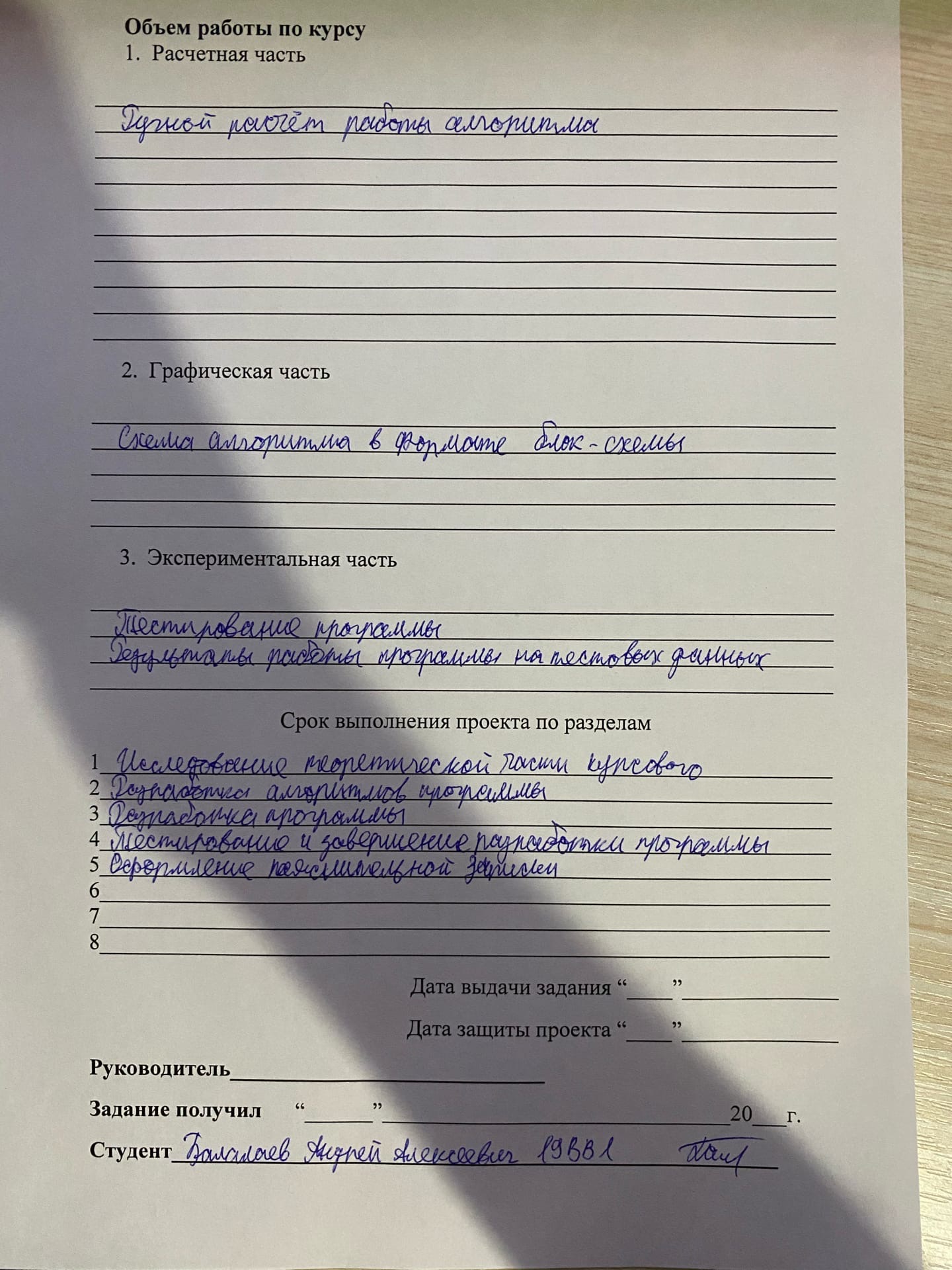
Выполнил:  
студент группы 19ВВ1  
Балалаев Андрей

Приняли:

д.т.н. профессор Митрохин М. А.

к.т.н. Юрова О.В.





**Содержание**

[Введение 5](#_Toc60068384)

[1. Постановка задачи 6](#_Toc60068385)

[2. Теоретическая часть задания 7](#_Toc60068386)

[3. Описание алгоритма программы 9](#_Toc60068387)

[4. Описание программы 11](#_Toc60068388)

[5. Тестирование 20](#_Toc60068389)

[6. Ручной расчёт задачи 23](#_Toc60068390)

[Заключение 25](#_Toc60068391)

[Список литературы 26](#_Toc60068392)

[Приложение 27](#_Toc60068393)

**Реферат**

Отчет 33 стр, 13 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ОРГРАФ, АЛГОРИТМ ФОРДА-БЕЛЛМАНА.

Цель исследования – разработка программы, способная вычислять кратчайшие расстояние между вершинами графа с помощью алгоритма Форда – Беллмана.

# Введение

Алгоритм Беллмана — Форда — алгоритм поиска кратчайшего пути во взвешенном графе. За время O(|V|\*|E|) алгоритм находит кратчайшие пути от одной вершины графа до всех остальных. В отличие от алгоритма Дейкстры, алгоритм Беллмана — Форда допускает рёбра с отрицательным весом. Предложен независимо Ричардом Беллманом и Лестером Фордом.

Алгоритм маршрутизации RIP (алгоритм Беллмана — Форда) был впервые разработан в 1969 году, как основной для сети ARPANET.

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudio2019, язык программирования – Си.

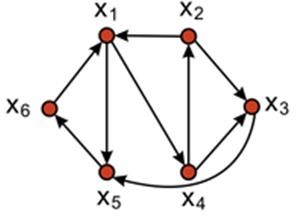
Целью данной курсовой работы является разработка программы на языке Си, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Беллмана — Форда.

# Постановка задачи

Требуется разработать программу, которая найдет кратчайшие пути между вершинами графа, используя алгоритм Форда – Беллмана.

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

# Теоретическая часть задания

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.

*Рисунок 1.1 Пример орграфа.*

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается единицей, иначе нулем.

Существует много алгоритмов на графах, в основе которых лежит систематический перебор вершин графа, такой, что каждая вершина графа просматривается только один раз, и переход от одной вершины к другой осуществляется по ребрам графа. Остановимся на одном из двух стандартных методах такого перебора - поиск в глубину.

Пусть задан орграф G= (X, a) и вершина X1, с которой начинается обход. После посещения вершины X1, следующей за ней будет посещена смежная с X1 вершиной. Далее, эта процедура повториться для вершин смежных с вершинами из множества q, за исключением вершины X1, т.к. она уже была

посещена. Так, продолжая обходить в глубину, алгоритм обойдет все доступные из X1 вершины орграфа G. Алгоритм прекращает свою работу после обхода всех вершин графа, либо в случае выполнения наличествующего условия.

Орграф называется сильно связным, если любые две его вершины сильно связаны. Две вершины X1 и X2 любого орграфа сильно связаны, если существует ориентированный путь из X1 в X2 и ориентированный путь из X2 в X1. Компонентами сильной связности орграфа называются его максимальные по включению сильно связные подграфы.

Любая вершина орграфа сильно связана сама с собой.

# Описание алгоритма программы

Также как и алгоритм Дейкстры, алгоритм Беллмана — Форда вычисляет во взвешенном графе кратчайшие пути от одной вершины до всех остальных. Он подходит для работы с графами, имеющими ребра с отрицательным весом. Но спектр применимости алгоритма затрагивает не все такие графы, ввиду того, что каждый очередной проход по пути, составленному из ребер, сумма весов которых отрицательна (т. е. по отрицательному циклу), лишь улучшает требуемое значение. Бесконечное число улучшений делает невозможным определение одного конкретного значения, являющегося оптимальным. В связи с этим алгоритм Беллмана — Форда ***не применим к графам, имеющим отрицательные циклы***.

Решить задачу, т. е. найти все кратчайшие пути из вершины s до всех остальных, используя алгоритм Беллмана — Форда, это значит воспользоваться методом динамического программирования: разбить ее на типовые подзадачи, найти решение последним, покончив тем самым с основной задачей. Здесь решением каждой из таких подзадач является определение наилучшего пути от одного отдельно взятого ребра, до какого-либо другого.

Для хранения результатов работы алгоритма заведем одномерный массив d[]. В каждом его i-ом элементе будет храниться значение кратчайшего пути из вершины s до вершины i (если таковое имеется). Изначально, присвоим элементам массива d[] значения равные условной бесконечности (например, число заведомо большее суммы всех весов), а в элемент d[s] запишем нуль.

Так мы задействовали известную и необходимую информацию, а именно известно, что наилучший путь из вершины s в нее же саму равен 0, и необходимо предположить недоступность других вершин из s. По мере выполнения алгоритма, для некоторых из них, это условие окажется ложным, и вычисляться оптимальные стоимости путей до этих вершин из s.

Задан граф G=(V, E), n=|V|, а m=|E|. Обозначим смежные вершины этого графа символами v и u (vÎV и uÎV), а вес ребра (v, u) символом w. Иначе говоря, вес ребра, выходящего из вершины v и входящего в вершину u, будет равен w.

Тогда ключевая часть алгоритма Беллмана — Форда примет следующий вид:

Для i от 1 до n-1 выполнять

Для j от 1 до m выполнять

Если d[v] + w(v, u) < d[u] то

d[u] < d[v] + w(v, u)

На каждом n-ом шаге осуществляются попытки улучшить значения элементов массива d[]: если сумма, составленная из веса ребра w(v, u) и веса хранящегося в элементе d[v], меньше веса d[u], то она присваивается последнему.

# Описание программы

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

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

Данная программа является многомодульной, поскольку состоит из нескольких функций: bellman\_ford(), rand\_Zap\_vz\_or(), rand\_Zap\_vz\_nor(), print\_G().

Работа программы начинается с меню. Пользователю предоставляется выбор пути генерации матрицы смежности: ручной ввод, произвольный размер.

int menu1;

printf(" 1. Ввести массив с клавиатуры. \n 2. Создание массива произвольного размера, вводимого с клавиатуры. \n ");

printf("\n\nВыберите пункт меню: ");

scanf("%d", &menu1);

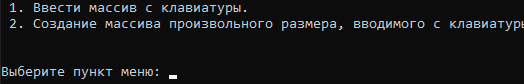
while (menu1 > 2)

{

printf("\nТакого пункта меню нет. Выберите пункт меню: ");

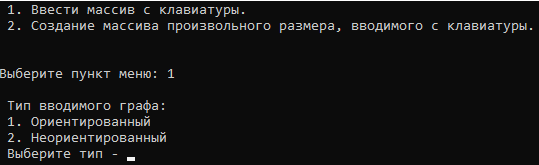
scanf("%d", &menu1);

}



*Рисунок 4.1 Меню 1*

При выборе любого из вариантов пользователю предоставляется выбор типа графа: ориентированный, неориентированный.



*Рисунок 4.2 Меню 2*

**Ввод матрицы с клавиатуры.**

При выборе пункта 1 начального меню (1. Ввести массив с клавиатуры.) пользователю предоставляется выбор типа графа: ориентированный, неориентированный.

switch (menu1)

{

case 1:

{

int i, j, n, type;

printf("\n Тип вводимого графа:\n 1. Ориентированный\n 2. Неориентированный \n Выберите тип - ");

scanf("%d", &type);

Далее пользователь выбирает количество вершин графа:

printf(" Введите количество вершин графа - ");

scanf\_s("%d", &n);

Выделение памяти под массив:

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

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

{

G[i] = (int\*)malloc(n \* sizeof(int));

}

При выборе заполнения ориентированного графа свою работу начинает цикл по заполнению матрицы с клавиатуры ориентированного графа.

if (type == 1)

{

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

{

for (j = i + 1; j < n; j++)

{

printf(" Введите расстояние из %d в %d - ", i + 1, j + 1);

scanf\_s("%d", &G[i][j]);

while (G[i][j] < 0)

{

printf("\n");

printf(" Введите положительное расстояние из %d в %d - ", i + 1, j + 1);

scanf\_s("%d", &G[i][j]);

}

printf(" Введите расстояние из %d в %d - ", j + 1, i + 1);

scanf\_s("%d", &G[j][i]);

while (G[j][i] < 0)

{

printf("\n");

printf(" Введите положительное расстояние из %d в %d - ", j + 1, i + 1);

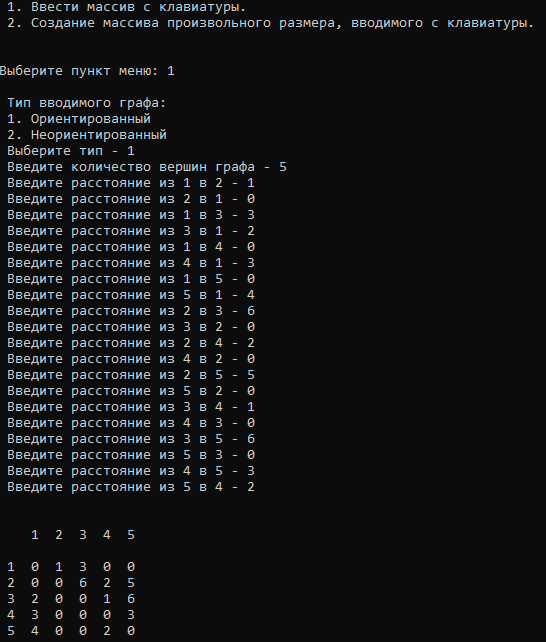
scanf\_s("%d", &G[j][i]);

}

}

}

}



*Рисунок 4.3 Заполнение матрицы смежности ориентированного графа с клавиатуры*

При выборе заполнения неориентированного графа свою работу начинает цикл по заполнению матрицы с клавиатуры неориентированного графа.

if (type == 2)

{

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

{

for (int j = i + 1; j < n; j++)

{

printf(" Введите расстояние из %d в %d - ", i + 1, j + 1);

scanf\_s("%d", &G[i][j]);

while (G[i][j] < 0)

{

printf("\n");

printf(" Введите положительное расстояние из %d в %d - ", i + 1, j + 1);

scanf\_s("%d", &G[i][j]);

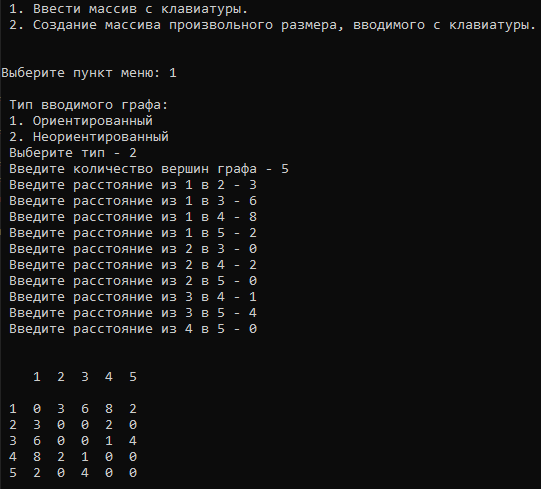
}

G[j][i] = G[i][j];

}

}

}



*Рисунок 4.4 Заполнение матрицы смежности неориентированного графа с клавиатуры*

Далее программа выполняет функцию print\_G(), которая выводит матрицу на экран, а так же записывает ее в файл.

void print\_G(int\*\* Matrix, int n)

{

save = fopen("save.txt", "w");

fprintf(save, " ");

printf(" ");

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

printf("%3d", i + 1);

fprintf(save, "%3d", i + 1);

printf("\n\n");

fprintf(save, "\n\n");

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

printf("%2d", i + 1);

fprintf(save, "%2d", i + 1);

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

printf("%3d", Matrix[i][j]);

fprintf(save, "%3d", Matrix[i][j]);

}

printf("\n");

fprintf(save, "\n");

}

printf("\n");

fprintf(save, "\n");

fclose(save);

}

Данные о матрице смежности записываются в массив структуры Edges для корректной работы алгоритма.

int w = 0;

e = 0;

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

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

{

w = G[i][j];

if (w != 0)

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = w;

e++;

}

}

Следующим шагом будет выбор пользователем стартовой вершины, от которой будут рассчитываться расстояния до других вершин графа.

printf("Введите стартовую вершину: ");

scanf("%d", &start);

printf("\n ");

Далее программа вызывает функцию bellman\_ford(), которая является алгоритмом Форда-Беллмана. Функция также сохраняет результат в файл.

void bellman\_ford(int n, int s)

{

save = fopen("save.txt", "a");

fseek(save, 0L, SEEK\_END);

int i, j;

for (i = 0; i < n; i++) d[i] = inf;

d[s] = 0;

for (i = 0; i < n - 1; i++)

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

if (d[edge[j].u] > d[edge[j].v] + edge[j].w)

d[edge[j].u] = d[edge[j].v] + edge[j].w;

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

if (d[i] == inf)

{

printf("%d -> %d - Нет пути\n", start, i + 1);

fprintf(save, "%d -> %d - Нет пути\n", start, i + 1);

}

else

{

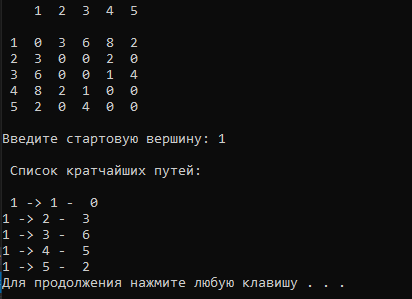
printf("%d -> %d - %d\n", start, i + 1, d[i]);

fprintf(save, "%d -> %d - %d\n", start, i + 1, d[i]);

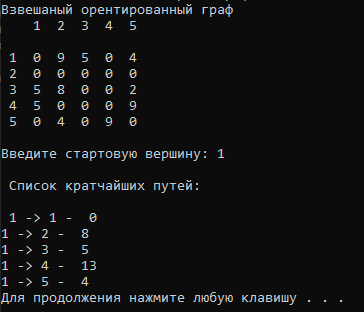
}

fclose(save);

}



*Рисунок 4.5 Работа алгоритма Беллмана-Форда для неориентированного графа*



*Рисунок 4.6 Работа алгоритма Беллмана-Форда для ориентированного графа*

**Заполнение матрицы случайными расстояниями.**

При выборе пункта 2 начального меню (2. Создание массива произвольного размера, вводимого с клавиатуры.) пользователю предоставляется выбор типа графа: ориентированный, неориентированный.

switch (menu1)

{

case 1:

{

int i, j, n, type;

printf("\n Тип вводимого графа:\n 1. Ориентированный\n 2. Неориентированный \n Выберите тип - ");

scanf("%d", &type);

Далее пользователь выбирает количество вершин графа:

printf(" Введите количество вершин графа - ");

scanf\_s("%d", &n);

Выделение памяти под массив:

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

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

{

G[i] = (int\*)malloc(n \* sizeof(int));

}

При выборе заполнения ориентированного графа свою работу начинает функция rand\_Zap\_vz\_or(), которая заполняет ориентированный граф случайными значениями.

void rand\_Zap\_vz\_or(int\*\* Matrix, int n)

{

srand(time(NULL));

printf("Взвешаный орентированный граф\n");

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

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

{

if (i == j)

{

Matrix[i][j] = 0;

}

if (i < j)

{

if (rand() % 100 > 50)

{

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

if (rand() % 100 > 50)

Matrix[j][i] = 0;

else

Matrix[j][i] = Matrix[i][j];

}

else

if (rand() % 100 > 50)

{

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

Matrix[i][j] = 0;

}

else

{

Matrix[i][j] = 0;

Matrix[j][i] = 0;

}

}

}

}

При выборе заполнения неориентированного графа свою работу начинает функция rand\_Zap\_vz\_nor(), которая заполняет неориентированный граф случайными значениями.

void rand\_Zap\_vz\_nor(int\*\* Matrix, int n)

{

srand(time(NULL));

printf("Взвешаный неорентированный граф\n");

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

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

{

if (i == j)

{

Matrix[i][j] = 0;

}

if (i < j)

{

if (rand() % 100 > 50)

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

else

Matrix[i][j] = 0;

Matrix[j][i] = Matrix[i][j];

}

}

}

Данные о матрице смежности записываются в массив структуры Edges для корректной работы алгоритма.

int w = 0;

e = 0;

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

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

{

w = G[i][j];

if (w != 0)

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = w;

e++;

}

}

Следующим шагом будет выбор пользователем стартовой вершины, от которой будут рассчитываться расстояния до других вершин графа.

printf("Введите стартовую вершину: ");

scanf("%d", &start);

printf("\n ");

Далее программа вызывает функцию bellman\_ford(), которая является алгоритмом Форда-Беллмана. Функция также сохраняет результат в файл.

void bellman\_ford(int n, int s)

{

save = fopen("save.txt", "a");

fseek(save, 0L, SEEK\_END);

int i, j;

for (i = 0; i < n; i++) d[i] = inf;

d[s] = 0;

for (i = 0; i < n - 1; i++)

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

if (d[edge[j].u] > d[edge[j].v] + edge[j].w)

d[edge[j].u] = d[edge[j].v] + edge[j].w;

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

if (d[i] == inf)

{

printf("%d -> %d - Нет пути\n", start, i + 1);

fprintf(save, "%d -> %d - Нет пути\n", start, i + 1);

}

else

{

printf("%d -> %d - %d\n", start, i + 1, d[i]);

fprintf(save, "%d -> %d - %d\n", start, i + 1, d[i]);

}

fclose(save);

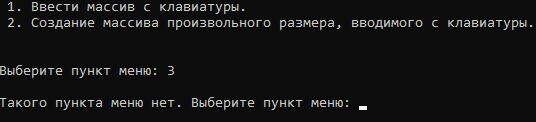
}

# Тестирование

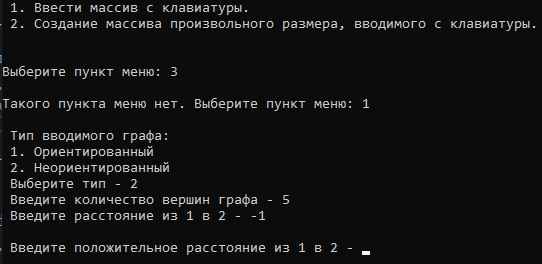
Среда разработки MicrosoftVisualStudio 2019 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

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

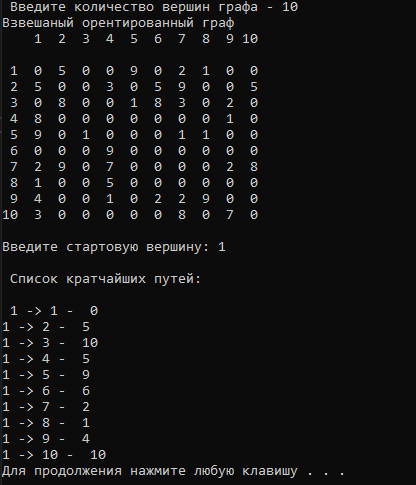
Ниже продемонстрирован результат тестирования программы.



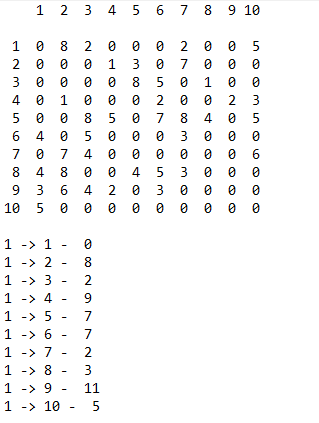
*Рисунок 5.1 Выбор неправильного пункта меню*



*Рисунок 5.2 Ввод отрицательного расстояния между вершинами*



*Рисунок 5.3 Работа программы с различным количеством вершин графа*



*Рисунок 5.4 Файл save.txt*

Таблица 1 – Описание поведения программы при тестировании

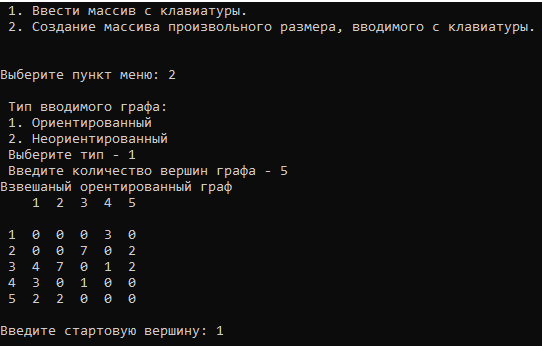
|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод сообщения о выборе: сгенерировать матрицу случайно или с помощью ручного ввода | Верно |
| Выбор генерации матрицы случайно | Вывод сообщения о выборе типа графа: ориентированный или неорентированный. Вывод сгенерированной матрицы  Выбор стартовой вершины. | Верно |
| Выбор генерации матрицы с  помощью ручного ввода | Вывод сообщения о  выборе типа графа:  ориентированный или  неорентированный.  Заполнение матрицы  с клавиатуры.  Вывод  сгенерированной  матрицы.  Выбор стартовой вершины. | Верно |
| Выбор стартовой вершины | Вывод результатов программы. | Верно |

.

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

# Ручной расчёт задачи

Проведем проверку программы посредством ручных вычислений на примере ориентированного графа с 5-ю вершинами:



*Рисунок 6.1 Ориентированный граф с 5-ю вершинами*

Начинаем обход из вершины 1.

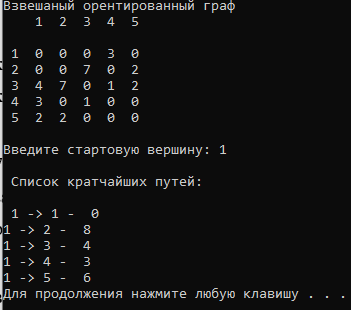
Из вершины 1 в вершину 1 кротчайшее расстояние будет 0.

Из вершины 1 в вершину 2 кротчайшее расстояние будет 8.  
Путь: 1->4->3->5->2. 3+1+2+2 = 8

Из вершины 1 в вершину 3 кротчайшее расстояние будет 4.  
Путь: 1->4->3. 3+1 = 4.

Из вершины 1 в вершину 4 кротчайшее расстояние будет 3.  
Путь: 1->4. 3

Из вершины 1 в вершину 5 кротчайшее расстояние будет 6.  
Путь: 1->4->3->5. 3+1+2 = 6.



*Рисунок 6.2 Результат работы алгоритма Форда-Беллмана*

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Форда - Беллмана в MicrosoftVisualStudio 2010.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей разных видов.

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. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.
  2. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006
  3. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978

# Приложение

Листинг программы.

// course.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <time.h>

#include <queue>

#include <locale.h>

#include <malloc.h>

#include <windows.h>

using namespace std;

struct Edges {

int u, v, w;

};

const int inf = 1000000000;

const int Vmax = 1000;

const int Emax = Vmax \* (Vmax - 1) / 2;

int i, j, n, e, start;

Edges edge[Emax];

int d[Vmax];

FILE\* save;

//алгоритм Беллмана-Форда

void bellman\_ford(int n, int s)

{

save = fopen("save.txt", "a");

fseek(save, 0L, SEEK\_END);

int i, j;

for (i = 0; i < n; i++) d[i] = inf;

d[s] = 0;

for (i = 0; i < n - 1; i++)

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

if (d[edge[j].u] > d[edge[j].v] + edge[j].w)

d[edge[j].u] = d[edge[j].v] + edge[j].w;

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

if (d[i] == inf)

{

printf("%d -> %d - Нет пути\n", start, i + 1);

fprintf(save, "%d -> %d - Нет пути\n", start, i + 1);

}

else

{

printf("%d -> %d - %d\n", start, i + 1, d[i]);

fprintf(save, "%d -> %d - %d\n", start, i + 1, d[i]);

}

fclose(save);

}

void rand\_Zap\_vz\_or(int\*\* Matrix, int n)

{

srand(time(NULL));

printf("Взвешаный орентированный граф\n");

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

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

{

if (i == j)

{

Matrix[i][j] = 0;

}

if (i < j)

{

if (rand() % 100 > 50)

{

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

if (rand() % 100 > 50)

Matrix[j][i] = 0;

else

Matrix[j][i] = Matrix[i][j];

}

else

if (rand() % 100 > 50)

{

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

Matrix[i][j] = 0;

}

else

{

Matrix[i][j] = 0;

Matrix[j][i] = 0;

}

}

}

}

void rand\_Zap\_vz\_nor(int\*\* Matrix, int n)

{

srand(time(NULL));

printf("Взвешаный неорентированный граф\n");

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

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

{

if (i == j)

{

Matrix[i][j] = 0;

}

if (i < j)

{

if (rand() % 100 > 50)

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

else

Matrix[i][j] = 0;

Matrix[j][i] = Matrix[i][j];

}

}

}

void print\_G(int\*\* Matrix, int n)

{

save = fopen("save.txt", "w");

fprintf(save, " ");

printf(" ");

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

{

printf("%3d", i + 1);

fprintf(save, "%3d", i + 1);

}

printf("\n\n");

fprintf(save, "\n\n");

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

printf("%2d", i + 1);

fprintf(save, "%2d", i + 1);

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

printf("%3d", Matrix[i][j]);

fprintf(save, "%3d", Matrix[i][j]);

}

printf("\n");

fprintf(save, "\n");

}

printf("\n");

fprintf(save, "\n");

fclose(save);

}

int main()

{

setlocale(LC\_CTYPE, "rus");

srand(time(NULL));

int menu1;

printf(" 1. Ввести массив с клавиатуры. \n 2. Создание массива произвольного размера, вводимого с клавиатуры. \n ");

printf("\n\nВыберите пункт меню: ");

scanf("%d", &menu1);

while (menu1 > 2)

{

printf("\nТакого пункта меню нет. Выберите пункт меню: ");

scanf("%d", &menu1);

}

switch (menu1)

{

case 1:

{

int i, j, n, type;

printf("\n Тип вводимого графа:\n 1. Ориентированный\n 2. Неориентированный \n Выберите тип - ");

scanf("%d", &type);

while (type > 2)

{

printf("\n Номер такого типа не существует\n Выберите тип заново - ");

scanf("%d", &type);

}

printf(" Введите количество вершин графа - ");

scanf\_s("%d", &n);

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

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

{

G[i] = (int\*)malloc(n \* sizeof(int));

}

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

{

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

{

G[i][j] = 0;

}

}

if (type == 1)

{

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

{

for (j = i + 1; j < n; j++)

{

printf(" Введите расстояние из %d в %d - ", i + 1, j + 1);

scanf\_s("%d", &G[i][j]);

while (G[i][j] < 0)

{

printf("\n");

printf(" Введите положительное расстояние из %d в %d - ", i + 1, j + 1);

scanf\_s("%d", &G[i][j]);

}

printf(" Введите расстояние из %d в %d - ", j + 1, i + 1);

scanf\_s("%d", &G[j][i]);

while (G[j][i] < 0)

{

printf("\n");

printf(" Введите положительное расстояние из %d в %d - ", j + 1, i + 1);

scanf\_s("%d", &G[j][i]);

}

}

}

}

if (type == 2)

{

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

{

for (int j = i + 1; j < n; j++)

{

printf(" Введите расстояние из %d в %d - ", i + 1, j + 1);

scanf\_s("%d", &G[i][j]);

while (G[i][j] < 0)

{

printf("\n");

printf(" Введите положительное расстояние из %d в %d - ", i + 1, j + 1);

scanf\_s("%d", &G[i][j]);

}

G[j][i] = G[i][j];

}

}

}

printf("\n\n");

print\_G(G, n);

int w = 0;

e = 0;

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

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

{

w = G[i][j];

if (w != 0)

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = w;

e++;

}

}

printf("Введите стартовую вершину: ");

scanf("%d", &start);

printf("\n ");

printf("Список кратчайших путей: ");

printf("\n\n ");

bellman\_ford(n, start - 1);

system("pause");

break;

}

case 2:

{

int i, j, n, type;

printf("\n Тип вводимого графа:\n 1. Ориентированный\n 2. Неориентированный \n Выберите тип - ");

scanf("%d", &type);

while (type > 2)

{

printf("\n Номер такого типа не существует\n Выберите тип заново - ");

scanf("%d", &type);

}

printf(" Введите количество вершин графа - ");

scanf\_s("%d", &n);

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

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

{

G[i] = (int\*)malloc(n \* sizeof(int));

}

if (type == 1)

rand\_Zap\_vz\_or(G, n);

if (type == 2)

rand\_Zap\_vz\_nor(G, n);

print\_G(G, n);

int w = 0;

e = 0;

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

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

{

w = G[i][j];

if (w != 0)

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = w;

e++;

}

}

printf("Введите стартовую вершину: ");

scanf("%d", &start);

printf("\n ");

printf("Список кратчайших путей: ");

printf("\n\n ");

bellman\_ford(n, start - 1);

system("pause");

break;

}

}

}