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 «Реализация алгоритма Беллмана - Форда»

Выполнил:

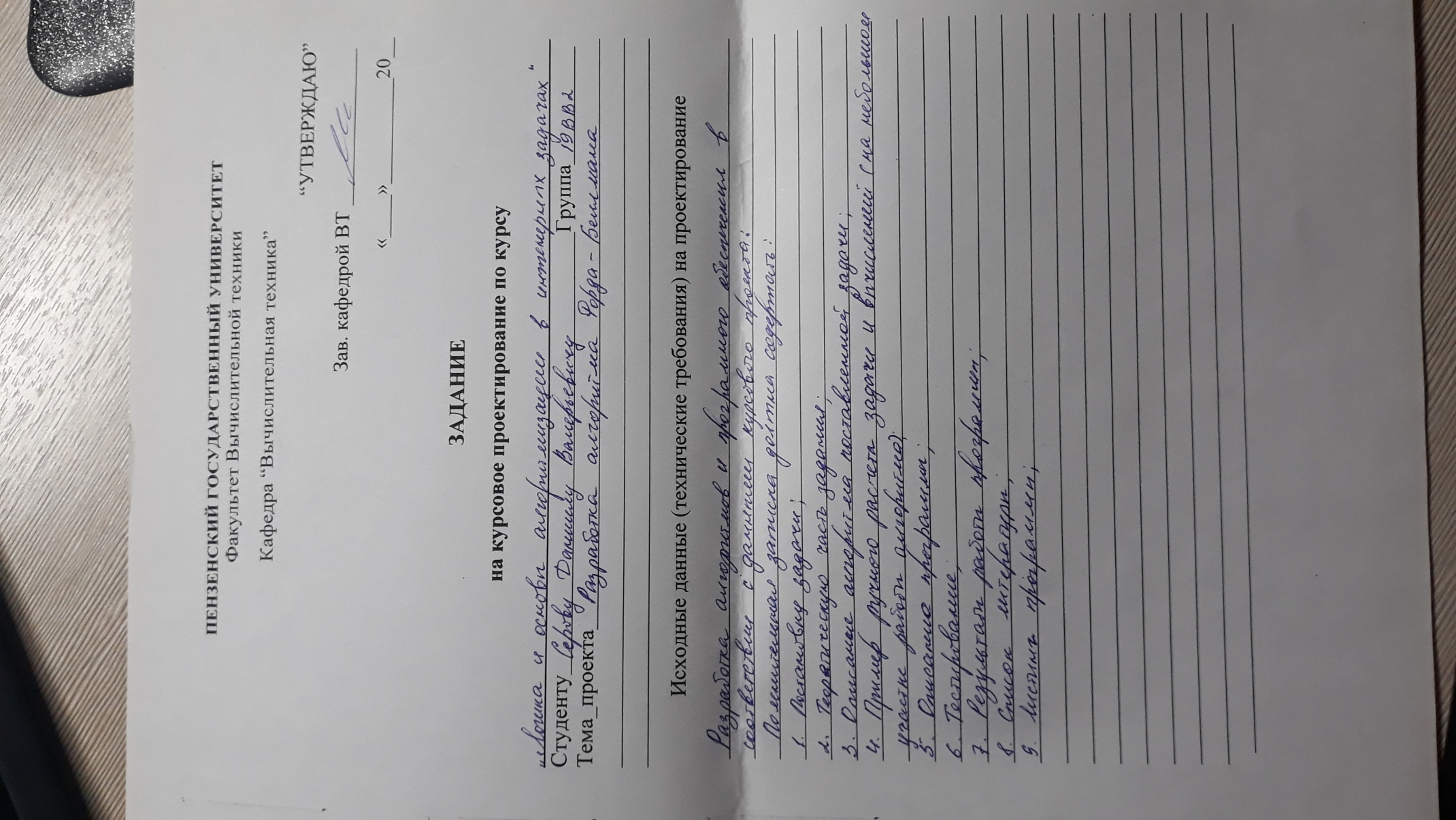
студент группы 19ВВ2

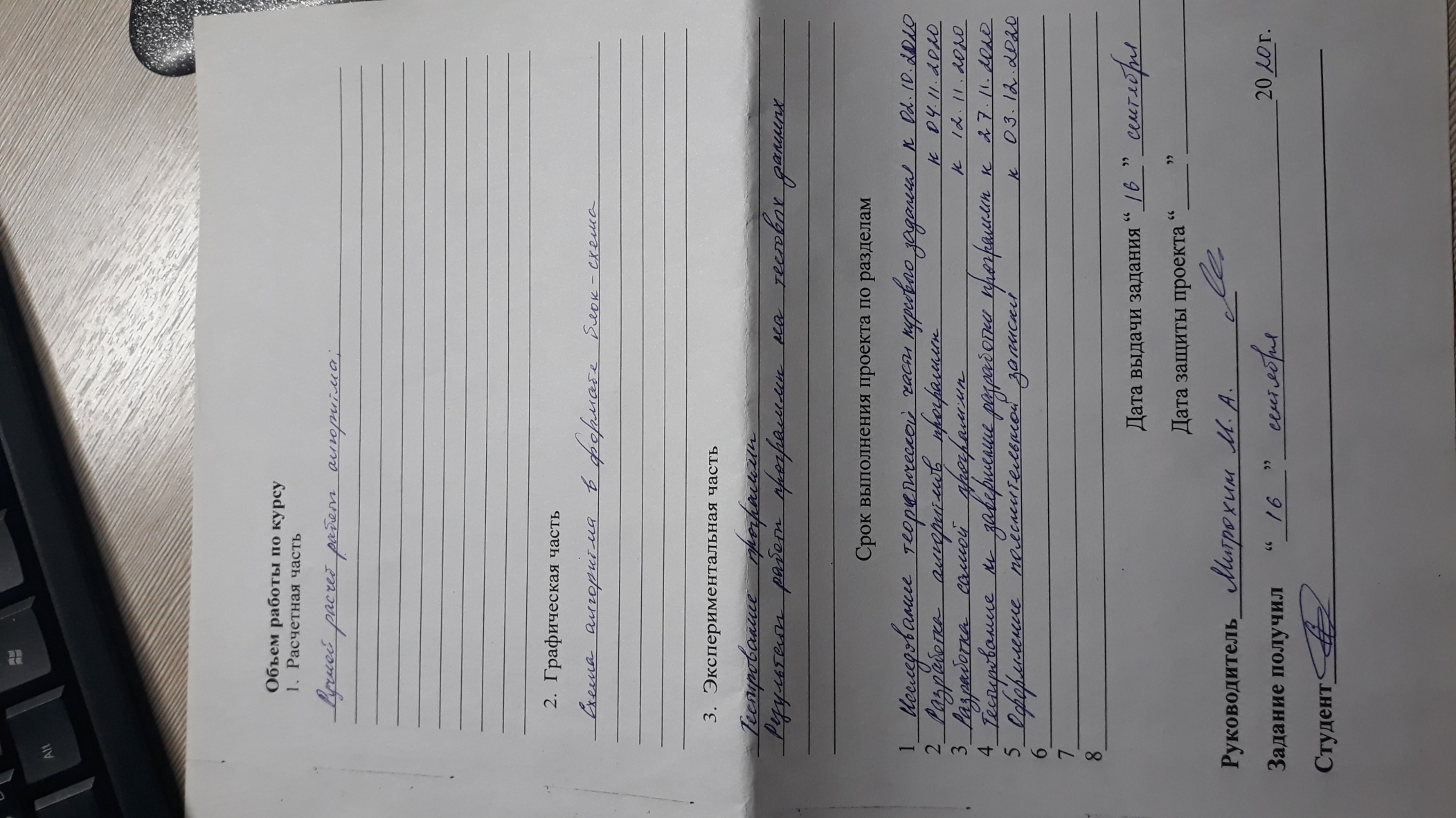
Серов Д.В.

Принял:

Митрохин М.А.

Пенза 2020





Оглавление

[Реферат 3](#_Toc59751900)

[Введение 4](#_Toc59751901)

[1. Постановка задачи 5](#_Toc59751902)

[2. Теоретическая часть задания 6](#_Toc59751903)

[3. Описание алгоритма программы 8](#_Toc59751904)

[4. Описание программы 11](#_Toc59751905)

[5. Ручной расчёт 20](#_Toc59751906)

[6. Тестирование 22](#_Toc59751907)

[Заключение 29](#_Toc59751908)

[Список используемых источников 30](#_Toc59751909)

[Приложение А. Листинг программы. 31](#_Toc59751910)

# 

# **Реферат**

Отчет 37 стр, 19 рисунков.

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

Цель исследования – разработка программы, реализующую алгоритм Беллмана-Форда. Получить навыки в разработке консольного приложения с пользовательским интерфейсом. Сохранить результаты работы программы в файл.

В работе рассмотрен алгоритм Беллмана-Форда. С помощью данного алгоритма можно находить кратчайшее расстояние от одной вершины графа до всех остальных, восстанавливать пути, также искать отрицательный цикл (если он есть).

# **Введение**

История алгоритма связана сразу с тремя независимыми математиками: Лестером Фордом, Ричардом Беллманом и Эдвардом Муром. Форд и Беллман опубликовали алгоритм в 1956 и 1958 годах соответственно, а Мур сделал это в 1957 году. И иногда его называют алгоритмом Беллмана – Форда – Мура. Метод используется в некоторых протоколах дистанционно-векторной маршрутизации, например в RIP (Routing Information Protocol – Протокол маршрутной информации).

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, но он позволяет определить наличие таковых, о чем будет сказано позже.

# **1. Постановка задачи**

Разработать программу, которая ищет кратчайшее расстояние от заданной вершины до всех остальных, на основе алгоритма Беллмана-Форда.

Программа должна предоставлять возможность пользователю выбирать ввод матрицы: автоматический (случайная генерация) или ручной(в самой программе). Граф должен задаваться матрицей смежности. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После вывода матрицы смежности, на экран должен выводиться список кратчайших путей и сохранение результата в файл. Если пользователь введет номер несуществующей команды, программа должна вывести соответствующее сообщение с просьбой заново ввести требуемые данные.

Для реализации этих требований необходимо разработать интуитивно понятный интерфейс.

Многомодульность программы. Это позволяет поделить программу на логические модули, также это помогает при расширение программы и локализации ошибок при отладке.

Устройство ввода – клавиатура и мышь.

Программа должна быть разработана для работы в операционной системе Microsoft Windows.

Средой разработки является Microsoft Visual Studio 2017, языки программирования C и C++.

# **2.** **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.

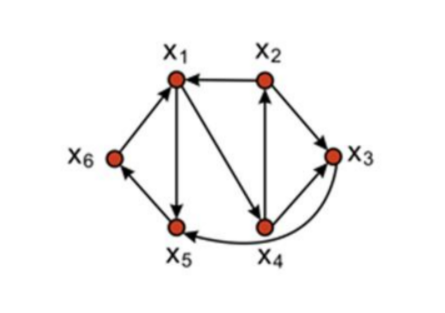


Рисунок 1 - Пример графа

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.

Алгоритм Беллмана-Форда применим, как к обычным графам, так и к графам, содержащим рёбра отрицательного веса. Впрочем, если граф содержит отрицательный цикл, то кратчайшего пути до некоторых вершин может не существовать (вес кратчайшего пути должен быть равен минус бесконечности); но, этот алгоритм можно модифицировать, чтобы он сигнализировал о наличии цикла отрицательного веса, или выводил сам этот цикл.

Этот алгоритм проходит в несколько фаз. На каждой фазе просматриваются все рёбра графа, и алгоритм пытается произвести релаксацию (ослабление) вдоль каждого ребра (v,u) веса w. Релаксация вдоль ребра — это попытка улучшить значение dist[u] значением dist[v] + w. Фактически это значит, что мы пытаемся улучшить ответ для вершины u, пользуясь ребром (v,u) и текущим ответом для вершины v.

Утверждается, что достаточно N-1 фазы алгоритма, чтобы корректно посчитать длины всех кратчайших путей в графе (при условии, что граф не содержит циклов отрицательного веса). Для недостижимых вершин расстояние dist[ ] останется равным бесконечности (заведомо большим числом).

# **3.** **Описание алгоритма программы**

Алгоритм Форда-Беллмана более удобно представлять граф в виде списка рёбер. В данной программе создается массив структур данных edge для рёбер. Входными данными для алгоритма являются числа n (кол-во вершин), e (кол-во рёбер), список edge рёбер, и номер стартовой вершины s. Все номера вершин нумеруются с 0 по N-1. Константа inf число "бесконечность" — она подобрана таким образом, чтобы она заведомо превосходила все возможные длины путей. Массив расстояний dist[0…n-1], который после отработки алгоритма будет содержать ответ на задачу. В начале заполняем массив следующим образом: начальная вершина, с которой начинается поиск, dist [s] = 0, все остальные элементы dist [ ] равны бесконечности (inf).

Этот алгоритм можно несколько ускорить: зачастую ответ находится уже за несколько фаз, а оставшиеся фазы никакой полезной работы не происходит, лишь впустую просматриваются все рёбра. Поэтому будем хранить флаг(flag) изменения на текущей фазе, и если на какой-то фазе ничего не произошло, то алгоритм можно останавливать. Но данная модификация не улучшает асимптотику, т.е. на некоторых графах по-прежнему будут нужны все n-1 фаза, но значительно ускоряет поведение алгоритма "в среднем", т.е. на случайных графах.

Алгоритм Форда-Беллмана модифицирован, чтобы он не только находил длины кратчайших путей, но и позволял восстанавливать сами кратчайшие пути. Для этого заведён ещё один вектор p[0…n-1], в котором для каждой вершины хранится её "предкок", т.е. предпоследняя вершина в кратчайшем пути, ведущая в неё. В самом деле, кратчайший путь до какой-то вершины v является кратчайшим путём до какой-то вершины p[v], к которому приписали в конец вершину v. Заметим, что алгоритм Форда-Беллмана работает по такой же логике: он, предполагая, что кратчайшее расстояние до одной вершины уже посчитано, пытается улучшить кратчайшее расстояние до другой вершины.

В цикле восстановления путей мы сначала проходимся по предкам, начиная с вершины i, и сохраняем весь пройденный путь в векторе path. В этом векторе получается кратчайший путь от s до i, но в обратном порядке, поэтому мы вызываем reverse от него и затем выводим.

При поиске кратчайшего пути может возникнуть момент, когда мы попадаем в вершину, в которой уже были до этого, это свидетельствует о наличие отрицательного цикла. В этом случае останавливаем работу алгоритма и выводим сообщение о некорректности результата.

Поскольку при наличии отрицательного цикла за N итераций алгоритма расстояния могли уйти далеко в минус, в коде приняты дополнительные меры против такого целочисленного переполнения:

dist[edge[j].u] = max(-inf, dist[edge[j].v] + edge[j].w);

**Описание работы функции bellman\_ford()**

Заполняем массив **dist [0…n-1] = inf**.

Инициализируем **dist** [**s**(стартовая вершина)] = 0.

Инициализируем вектор **p[0…n-1] = -1**.

Инициализируем **x**.

Для i = 0; пока i < n-1; делать i = i + 1.

x = -1

Флаг flag = 0.

Для j = 0; пока j < e (кол-во рёбер); делать j = j + 1.

Если dist[edge[j].u]> dist[edge[j].v] + edge[j].w.

То dist[edge[j].u] = наибольшее из (-inf либо dist[edge[j].v] + edge[j].w).

p[edge[j].u] = edge[j].v.

flag = 1.

x = edge[j].u

Если flag != 1

Выходим из цикла.

Если x != -1

Выводим сообщение об отрицательном цикле

Вызываем функцию **out()**

Открываем/создаём файл Форд-Беллман.txt для сохранения результатов.

Для i = 0; пока i < n; i = i + 1.

Если dist[i] == inf

Выводим в консоль и файл, что «нет пути».

Иначе

Выводим в консоль и файл dist [i]

Объявляем вектор **path**.

Для cur = i; пока cur ≠ -1; делать cur = p[cur].

Увеличиваем на 1 размер path и заносим в конец cur.

Меняем порядок элементов path[] на обратный.

Цикл вывода path[].

# **4. 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 Visual C++ .

Данная программа является многомодульной, поскольку состоит из нескольких функций: main(), random(), hand(), vyvod(), save(), out(), bllman\_ford().

Работа программы начинается с вывода меню на экран. Пользователю даётся на выбор 3 действия. Для выбора действия пользователю нужно нажать соответствующую клавишу на клавиатуре. Если выбран пункт, которого нет в списке, то выводится соответствующее сообщение и предлагается выбрать нужный пункт заново.

1. Создать массив случайных чисел.

2. Ручной ввод.

3. Выход.

while (true)//Главный цикл

{

system("cls");

printf("\n\n\t\tМеню\n\n");

printf("\t1.Создать массив случайных чисел\n");

printf("\t2.Ручной ввод\n");

printf("\t3.Выход\n\n");

printf("\tВыберите пункт меню (1-3): ");

scanf("%d", &choose\_number);

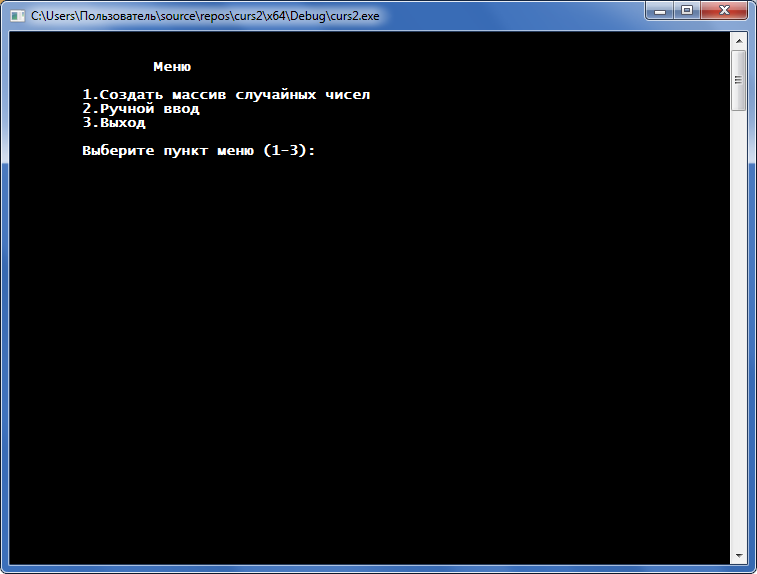


Рисунок 2 – Меню программы.

Если пользователь выбирает номер пнкта, который не предусмотрен в меню, выводится сообщение об ошибке, пользователю предлагается выбрать нужный ему пункт заново.

while ((choose\_number<1)|| (choose\_number > 3))

{

printf("\n\tВыбран пункт, которого нет в меню!!!\n");

printf("\n\tВыберите пункт меню (1-3): ");

scanf("%d", &choose\_number);

}

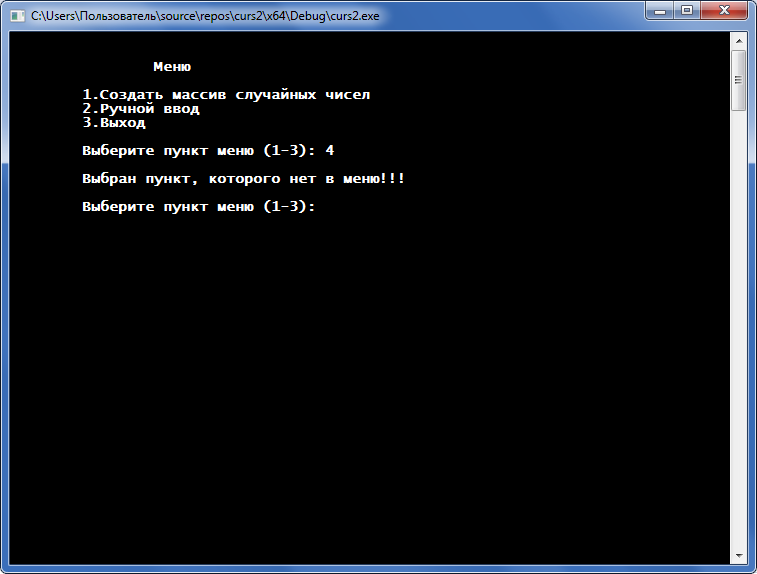


Рисунок 3 – Сообщение об ошибке в меню программы.

При выборе первого пункта, пользователь попадет в меню автоматического создания матрицы. Реализация этого пункта содержится в функции random().

**Функция random():**

void random() //функция создания графа случайных чисел

{

printf("\n\n Количество вершин: ");

scanf("%d", &n);

srand(time(NULL));

mass = new int\*[n]; //выделение памяти для массива

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

mass[i] = new int[n];

}

e = 0;

for (i = 0; i < n; i++) {// генерация массива

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

{

R = rand() % 100;

if (R > 60) {

int ran = (rand() % 10 - 5);

mass[i][j] = ran;

}

else {

int ran = rand() % 10;

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

}

if (i == j) {

mass[i][j] = 0;

}

if (mass[i][j] != 0)// передача значений в список ребер

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = mass[i][j];

e++;

}

}

}

}

Далее за функцией random() вызывается функция vyvod(), которая выводит на экран сгенерированный массив.

**Функция vyvod():**

void vyvod() //функция вывода графа на экран

{

printf("\n\n");

printf(" ");

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

{

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

}

printf("\n\n");

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

{

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

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

{

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

}

printf("\n\n");

}

}

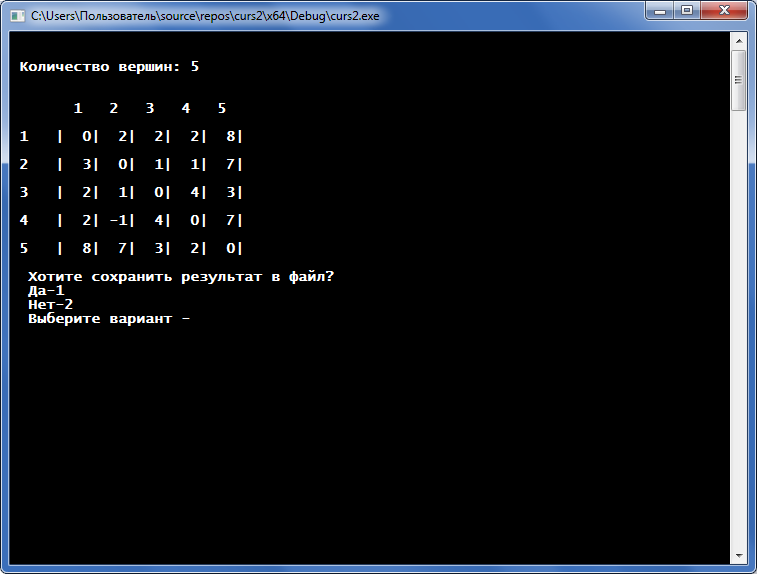


Рисунок 3 – Создание и вывод массива.

После вывода сгенерированного массива, пользователю предлагается выбор, хочет он сохранить результат, который будет в данном пункте, либо нет. При нажатии клавиши «1» пользователь соглашается сохранить результат (результат будет сохранен в папку с программой под названием «Форд-Беллман.txt»), при нажатии клавиши «2» пользователь отказывается от этой операции.

**Функция save():**

void save() //функция сохранения

{

save\_res = fopen("Форд-Беллман.txt", "w");//создание\открытие файла для записи

fprintf(save\_res, "\n Колличество вершин %d \n\n", n);

fprintf(save\_res, " ");// сохранение массива

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

{

fprintf(save\_res, " %2d ", j + 1);

}

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

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

{

fprintf(save\_res, " %d |", i + 1);

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

{

fprintf(save\_res, "%3d|", mass[i][j]);

}

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

}

fprintf(save\_res, "Список кратчайших путей:");}//сохранение списка кратчайших путей

При выборе пункта, который не предусмотрен в программе, пользователю выведется соответствующее сообщение и будет предложено ввести заново нужный вариант.

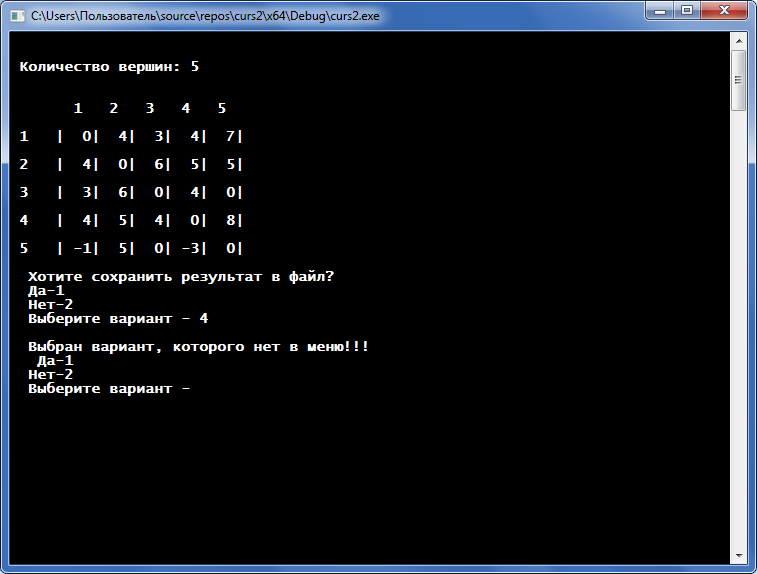
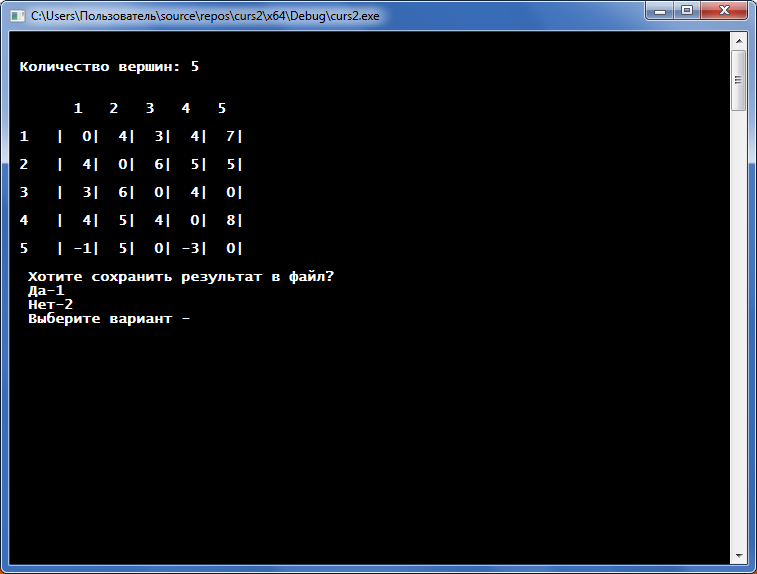


Рисунок 4 – Сохранение результата и вывод ошибки.

Затем выводится результат поиска кратчайших путей с помощью алгоритма Беллмана-Форда. Сам алгоритм реализова в функции bellman\_ford(), а вывод результата работы алгоритма реализован в функции out(). В функции out() также реализован поиск отрицательного цикла и восстановление путей.

**Функция bellman\_ford():**

void bellman\_ford(int n, int s) //алгоритм Беллмана-Форда

{

int i, j;

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

{

dist[i] = inf;

}

dist[s] = 0;

vector<int> p(n, -1);

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

{

bool flag = false;

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

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

{

dist[edge[j].u] = max(-inf, dist[edge[j].v] + edge[j].w);

p[edge[j].u] = edge[j].v;

flag = true;

}

if (!flag) break;

}

out(n, dist,start,p);

}

**Функция out():**

void out(int n,int\* dist,int start, vector<int> p) //функция сохранения

{

save\_res = fopen("Форд-Беллман.txt", "a");

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

{

if (dist[i] == inf)

{

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

fprintf(save\_res,"\n [%d]-[%d]= нет пути", start, i + 1);

}

else

{

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

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

vector<int> path;// Восстановление путь

for (int cur = i; cur != -1; cur = p[cur])

{

path.push\_back(cur);

}

reverse(path.begin(), path.end());

printf(" (");

fprintf(save\_res, " (");

for (size\_t i = 0; i < path.size(); ++i)

if ((i + 1) != path.size())

{

printf("%d->", path[i] + 1);

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

}

else

{

printf("%d", path[i] + 1);

fprintf(save\_res, "%d", path[i] + 1);

}

printf(")");

fprintf(save\_res, ")");

}

printf("\n\n");

}

}

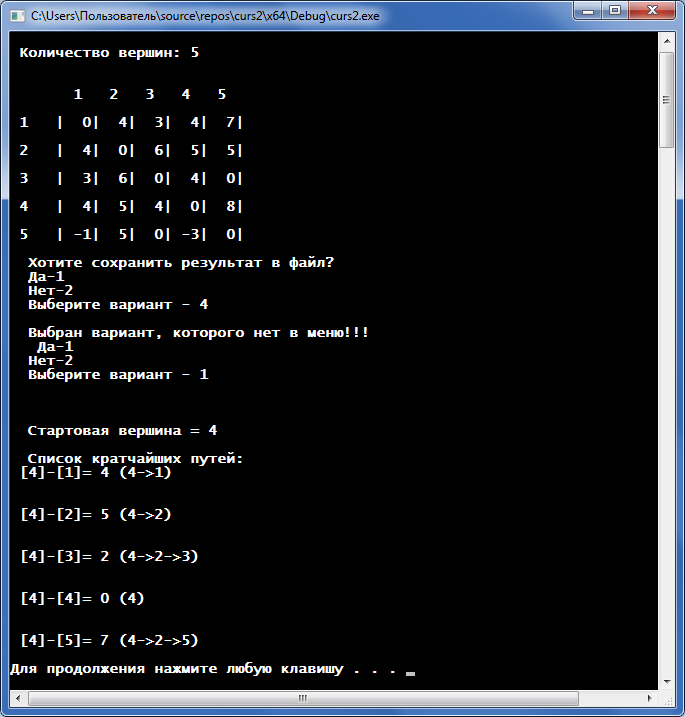


Рисунок 5 – Результат работы алгоритма.

Затем пользователь нажатием любой клавиши возвращается в меню, там при выборе пункта 2 он переходит в меню с ручным вводом массива. Ввод веса ребер реализвон в функции hand(). Пользователю предлагается ввести для каждого ребра вес, который может быть не только положительным, но и отрицательным. После ввода всех ребер, как и в пункте с случайным вводом, на экран выводица матрица, затем пользователю предлагается сохранить результ и вывод результата.

**Функция hand():**

void hand() //функция ручного ввода графа

{

printf("\n\n Количество вершин: ");

scanf("%d", &n);

mass = new int\*[n]; //выделение памяти для массива

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

mass[i] = new int[n];

}

e = 0;

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

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

{

printf(" Вес %d->%d= ", i + 1, j + 1);

scanf(" %d", &mass[i][j]); //запоминание массива

if (mass[i][j] != 0)// передача значений в список ребер

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = mass[i][j];

e++;

}

}

}

}

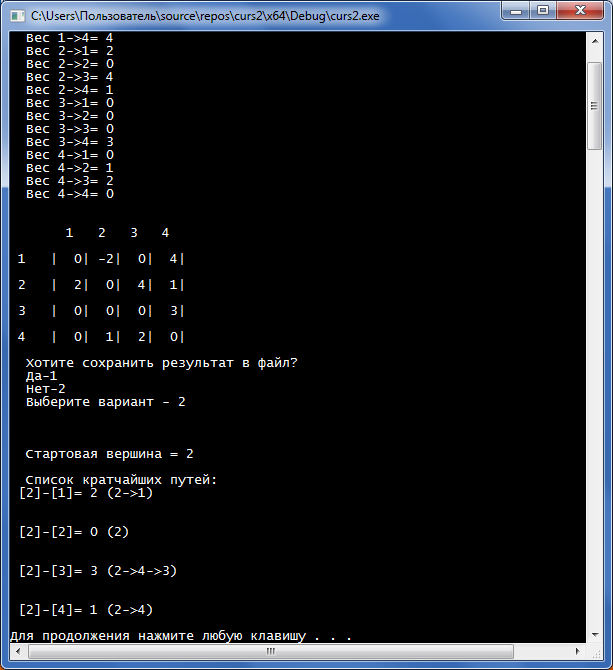


Рисунок 6 – Результат работы в пункте с ручным вводом.

После завершения работы в пункте 2, пользователь может вернутся в меню нажатием любой клавиши, и для выхода из программы пользователю нужно выбрать 3 пункт.

if (choose\_number == 3)//завершение работы

{

break;

}

# **5. Ручной расчёт**

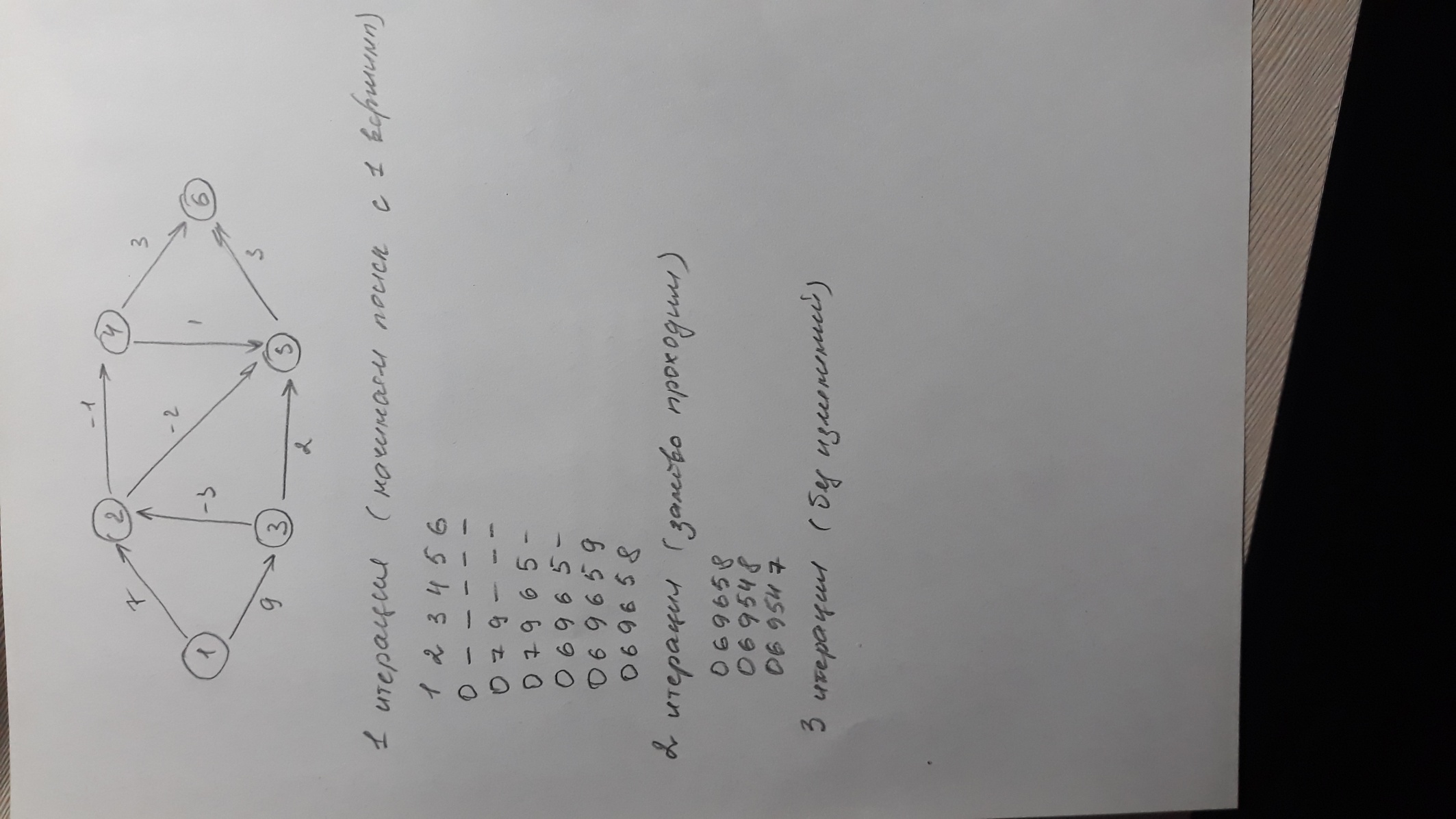


Рисунок 7 – Ориентированный граф.

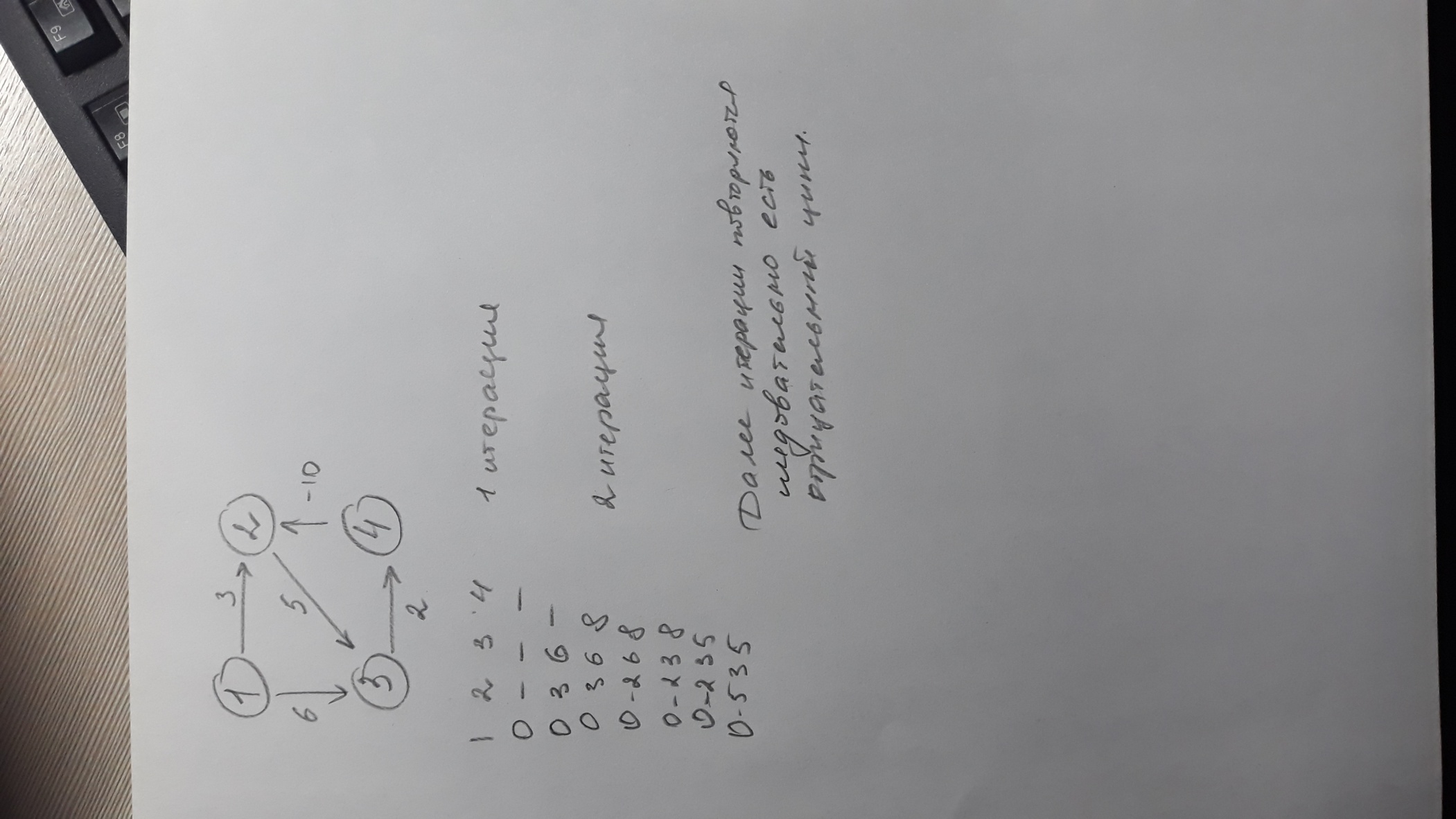


Рисунок 8 – Граф с отрицательным циклом.

# **6.** **Тестирование**

|  |  |  |  |
| --- | --- | --- | --- |
| Описание теста | Порядок действий | Ожидаемый результат | Полученный результат |
| Запуск программы | Запустить «curs2.exe» | Вывод меню в консоль | Верно |
| Ошибка при выборе пункта, которого нет в меню | Ввести номер пункта меньше 1 либо больше 3 | Вывод сообщения в консоль «Выбран пункт, которого нет в меню!!!» | Верно |
| Генерация графа | Выбрать первый пункта меню «Создать массив случайных чисел», ввести количество вершин | Вывод в консоль графа | Верно |
| Сохранение результата случайного графа в файл «Форд-Беллман.txt» | Выбрать вариант 1 для сохранения результата | При открытии файла «Форд-Беллман.txt», мы увидим сохраненный результат | Верно |
| Ошибка при введении неправильного варианта для сохранения | Выбрать номер пункта меньше 1 либо больше 2 | Вывод сообщения в консоль «Выбран вариант, которого нет в меню!!! » | Верно |
| Ввод стартовой вершины для поиска минимальных путей | Ввести вершину от которой нужно начать поиск | Вывод в консоль кратчайших путей | Верно |
| Ошибка при введении несуществующей вершины | Ввести вершину больше чем размер графа | Вывод сообщения в консоль «Выбрана вершина, которой не существует!!!» | Верно |
| Отрицательный цикл | Ввести вершину, при которой найдется отрицательный цикл | Вывод сообщения “Отрицательный цикл” | Верно |
| Ручной ввод графа | Выбрать второй пункта меню, ввести вес ребра | Вывод в консоль массив введенных чисел | Верно |
| Выход из программы | Выбор третьего пункта меню | Завершение работы программы с кодом 0 | Верно |

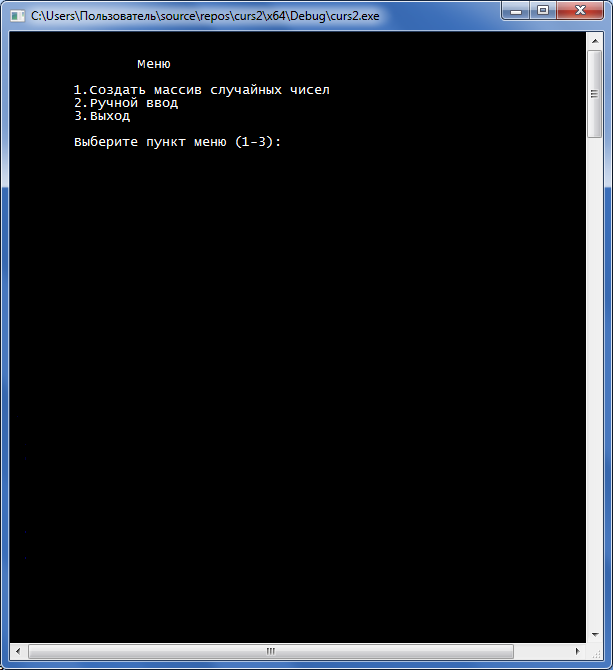


Рисунок 9– Тест запуска программы.

При запуске программы пользователю выводится меню, в котором он выбирает нужный ему пункт.

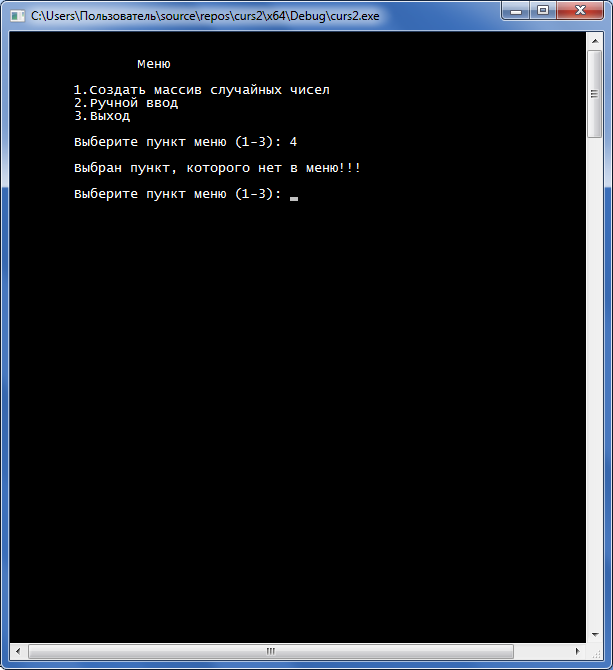


Рисунок 10– Тест вывода ошибки.

Если пользователь введет номер пункта, которого нет в меню, выведется сообщение об ошибке и будет предложенно заново ввести номер пункта.

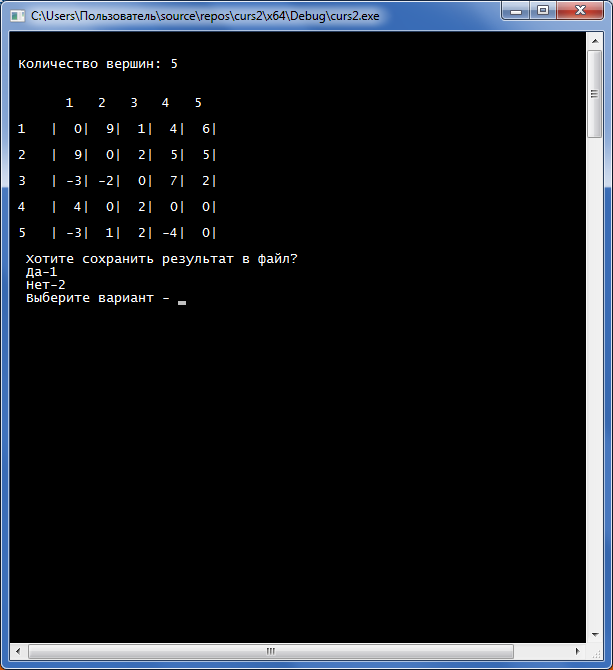


Рисунок 11– Тест генерации графа.

Если пользователь выбрал первый пункт меню, ему будет предложено ввести количество вершин в графе, которое он хочет, и матрица автоматически сгенерируется и выведится в консоль.

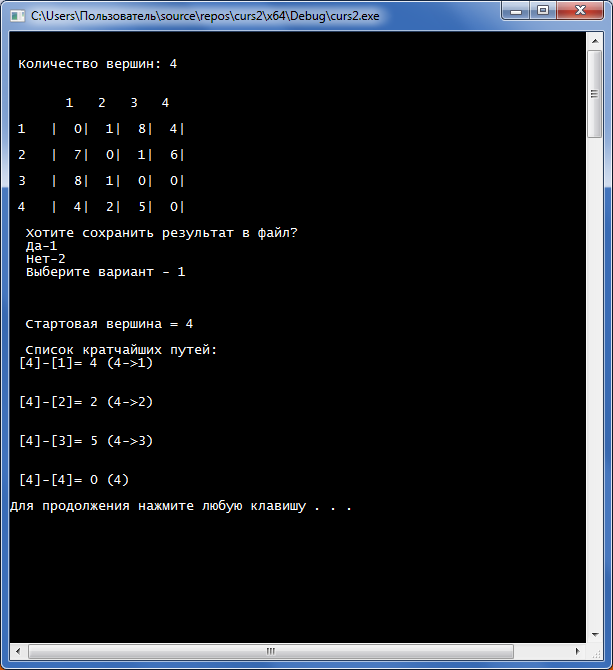
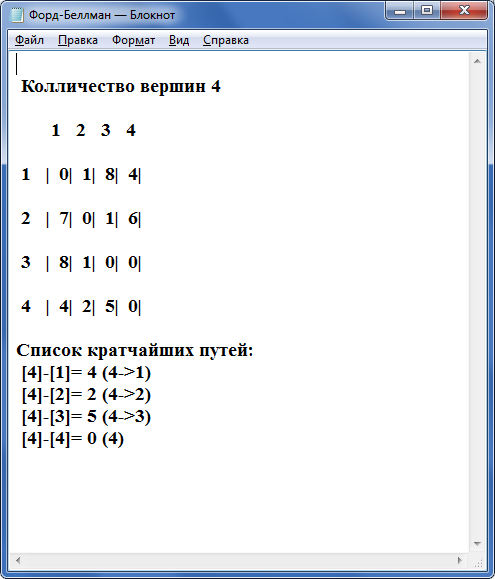
 

Рисунок 12 – Тест сохранения результата в файл.

После вывода матрицы, пользователю будет предложенно 2 варианта, сохранить матрицу и последующий результат в файл либо же нет.

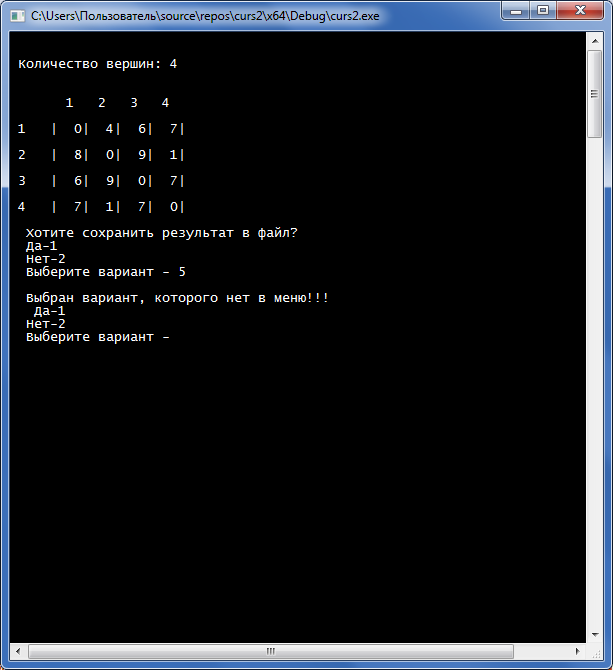


Рисунок 13– Тест вывода сообщения об ошибке.

Если пользователь ввел номер варианта, которого нет будет выведено сообщение об ошибке и предложено заново выбрать нужный вариант.

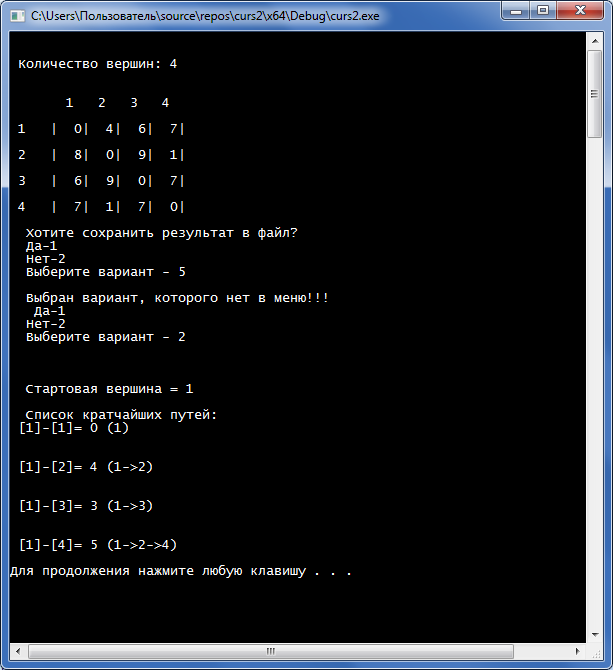


Рисунок 14– Тест ввода стартовой вершины для начала поиска.

После вывода матрицы и сохранения, пользователю будет предложенно ввести номер вершины, с которой он хочет начать поиск.

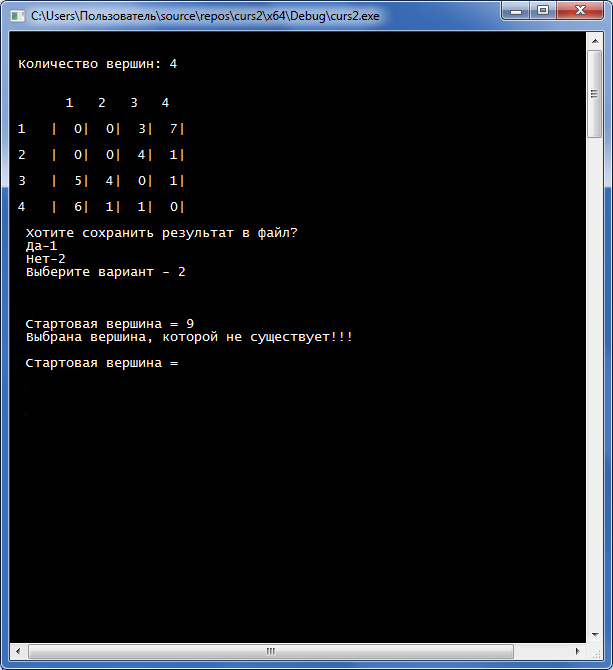


Рисунок 15– Тест вывода ошибки о неправильном вводе вершины.

Если введена вершина, которой не существует, будет выведено сообщение об ошибке и предложенно заново ввести нужную вершину.

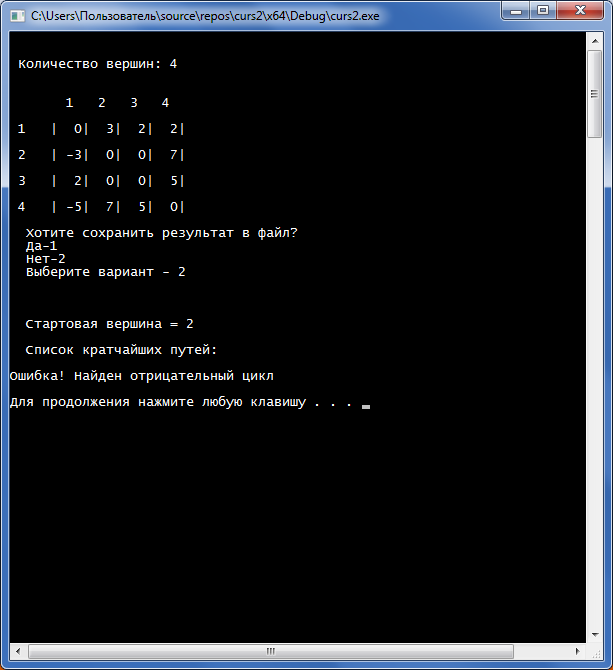


Рисунок 16– Тест вывода сообщения об отрицательном цикле.

Если в графе был обнаружен отрицательный цикл, пользователю выводится сообщение об этом, и дальнейшая работа по поиску путей прекращается так, как ответ будет некоректным.

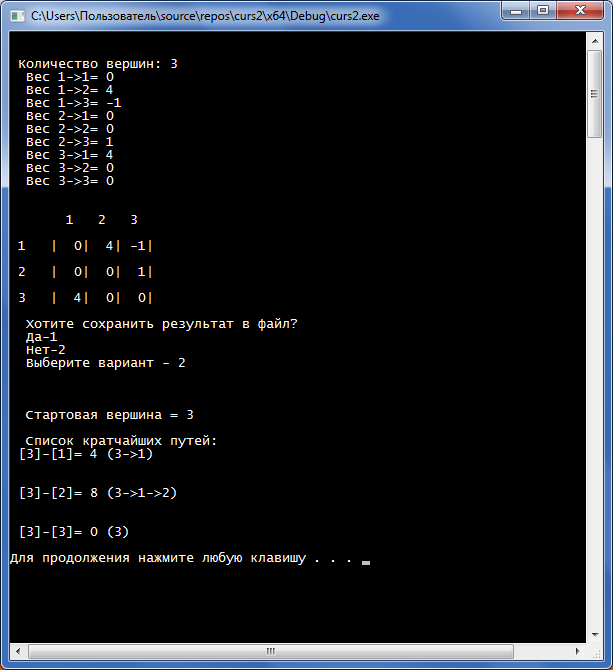
.

Рисунок 17 – Тест ручного ввода графа.

Если пользователь выбирает второй пункт меню, ему предлагоется ввести количество вершин и их вес самостоятельно, после чего предыдущие операции повторятся.

Если пользователь выберет третий пункт меню, программа завершит свою работу и закроется.

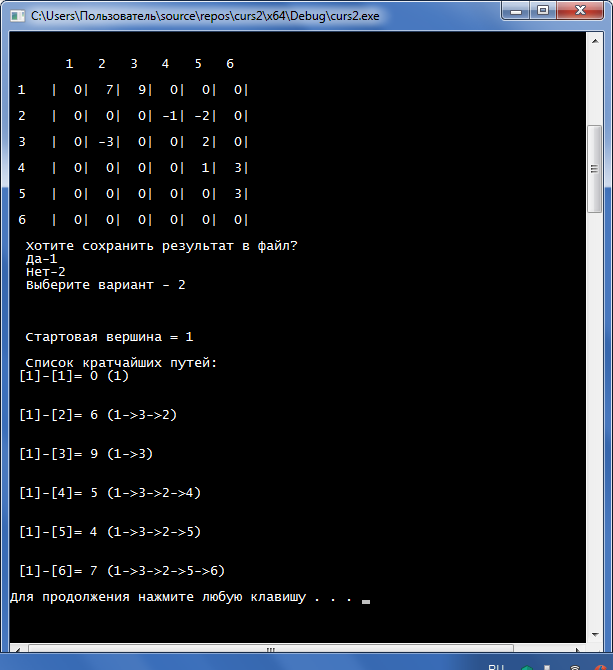


Рисунок 18 – Результат работы алгоритма совпадает с ручным расчётом(рис.7).

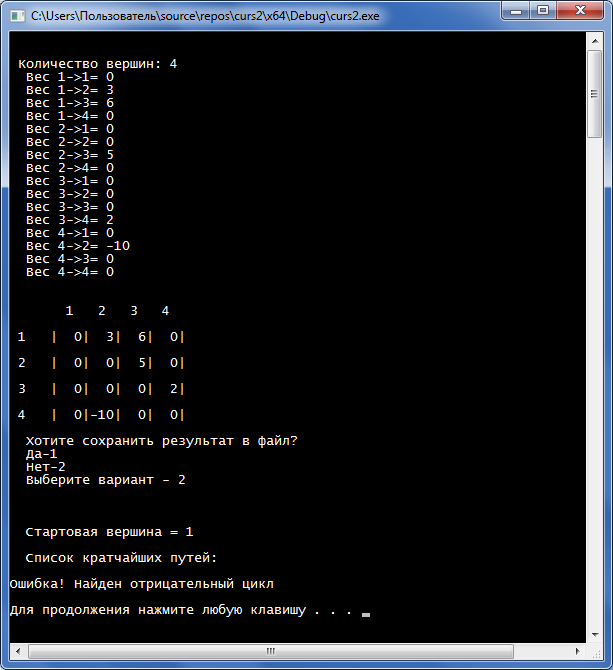


Рисунок 19 – Результат работы алгоритма с отрицательным циклом совпадает с ручным расчётом(рис.8).

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

# **Заключение**

Таким образом, в процессе создания данного проекта разработана программа, которая, используя алгоритм Беллмана-Форда, находит длины кратчайших путей от стартовой вершины до всех остальных вершин, запоминает путь до этих вершин, а также находит циклы отрицательного веса при их наличии.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории орграфов. Приобретены навыки по осуществлению алгоритма Беллмана-Форда. Углублены знания языков программирования C и C++.

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

Программа имеет небольшой, но достаточный для использования функционал возможностей.

# **Список используемых источников**

1. https://e-maxx.ru/algo/ford\_bellman
2. https://habr.com/ru/company/otus/blog/484382/
3. К. Поляков, Программирование на языке Си. 2012
4. https://ru.wikipedia.org/wiki/Алгоритм\_Беллмана\_—\_Форда.

# **Приложение А. Листинг программы.**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include "stdio.h"

#include "malloc.h"

#include "locale.h"

#include "time.h"

#include <iostream>

#include <vector>

#define inf 100000000

using namespace std;

struct Edges {int u, v, w;};

const int Vmax = 1000;

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

int i, j, n, e, start;

Edges edge[Emax];

int dist[Vmax];

int R;

int\*\* mass;

FILE\* save\_res;

void out(int n,int\* dist,int start, vector<int> p) //функция сохранения

{

save\_res = fopen("Форд-Беллман.txt", "a");

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

{

if (dist[i] == inf)

{

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

fprintf(save\_res,"\n [%d]-[%d]= нет пути", start, i + 1);

}

else

{

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

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

vector<int> path;// Восстановление путь

for (int cur = i; cur != -1; cur = p[cur])

{

path.push\_back(cur);

}

reverse(path.begin(), path.end());

printf(" (");

fprintf(save\_res, " (");

for (size\_t i = 0; i < path.size(); ++i)

if ((i + 1) != path.size())

{

printf("%d->", path[i] + 1);

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

}

else

{

printf("%d", path[i] + 1);

fprintf(save\_res, "%d", path[i] + 1);

}

printf(")");

fprintf(save\_res, ")");

}

printf("\n\n");

}

}

void bellman\_ford(int n, int s) //алгоритм Беллмана-Форда

{

int i, j;

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

{

dist[i] = inf;

}

dist[s] = 0;

vector<int> p(n, -1);

int x;

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

{

x = -1;

bool flag = false;

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

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

{

dist[edge[j].u] = max(-inf, dist[edge[j].v] + edge[j].w);

p[edge[j].u] = edge[j].v;

x = edge[j].u;

flag = true;

}

if (!flag) break;

}

if(x!=-1){

save\_res = fopen("Форд-Беллман.txt", "a");

for (i = 0; i < e; ++i) {

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

{

printf("\n\nОшибка! Найден отрицательный цикл\n\n");

fprintf(save\_res,"\n\nОшибка! Найден отрицательный цикл\n");

return;

}

}

}

out(n, dist,start,p);

}

void random() //функция создания графа случайных чисел

{

printf("\n\n Количество вершин: ");

scanf("%d", &n);

srand(time(NULL));

mass = new int\*[n]; //выделение памяти для массива

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

mass[i] = new int[n];

}

e = 0;

for (i = 0; i < n; i++) {// генерация массива

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

{

R = rand() % 100;

if (R > 60) {

int ran = (rand() % 10-5 );

mass[i][j] = ran;

}

else {

int ran = rand() % 10;

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

}

if (i == j) {

mass[i][j] = 0;

}

if (mass[i][j] != 0)// передача значений в список ребер

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = mass[i][j];

e++;

}

}

}

}

void hand() //функция ручного ввода графа

{

printf("\n\n Количество вершин: ");

scanf("%d", &n);

mass = new int\*[n]; //выделение памяти для массива

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

mass[i] = new int[n];

}

e = 0;

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

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

{

printf(" Вес %d->%d= ", i + 1, j + 1);

scanf(" %d", &mass[i][j]); //запоминание массива

if (mass[i][j] != 0)// передача значений в список ребер

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = mass[i][j];

e++;

}

}

}

}

void vyvod() //функция вывода графа на экран

{

printf("\n\n");

printf(" ");

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

{

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

}

printf("\n\n");

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

{

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

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

{

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

}

printf("\n\n");

}

}

void save() //функция сохранения

{

save\_res = fopen("Форд-Беллман.txt", "w");//создание\открытие файла для записи

fprintf(save\_res, "\n Колличество вершин %d \n\n", n);

fprintf(save\_res, " ");// сохранение массива

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

{

fprintf(save\_res, " %2d ", j + 1);

}

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

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

{

fprintf(save\_res, " %d |", i + 1);

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

{

fprintf(save\_res, "%3d|", mass[i][j]);

}

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

}

fprintf(save\_res, "Список кратчайших путей:");//сохранение списка кратчайших путей

}

int main() //главная функция

{

setlocale(LC\_ALL, "Rus");

int choose\_number, choose\_number2;

while (true)//Главный цикл

{

system("cls");

printf("\n\n\t\tМеню\n\n");

printf("\t1.Создать массив случайных чисел\n");

printf("\t2.Ручной ввод\n");

printf("\t3.Выход\n\n");

printf("\tВыберите пункт меню (1-3): ");

scanf("%d", &choose\_number);

while ((choose\_number<1)|| (choose\_number > 3))

{

printf("\n\tВыбран пункт, которого нет в меню!!!\n");

printf("\n\tВыберите пункт меню (1-3): ");

scanf("%d", &choose\_number);

}

if (choose\_number == 1)//рандомный массив

{

system("cls");

random();

vyvod();

printf(" Хотите сохранить результат в файл?\n");

printf(" Да-1\n");

printf(" Нет-2\n");

printf(" Выберите вариант - ");

scanf("%d", &choose\_number2);

printf(" \n");

while ((choose\_number2 < 1)|| (choose\_number2 > 2))

{

printf(" Выбран вариант, которого нет в меню!!!\n ");

printf(" Да-1\n");

printf(" Нет-2\n");

printf(" Выберите вариант - ");

scanf("%d", &choose\_number2);

printf(" \n");

}

if (choose\_number2 == 1)

{

save();

}

printf("\n\n Стартовая вершина = ");

scanf("%d", &start);

while (start > n) {

printf(" Выбрана вершина, которой не существует!!!\n ");

printf("\n Стартовая вершина = ");

scanf("%d", &start);

}

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

bellman\_ford(n, start - 1);

system("pause");

continue;

}

if (choose\_number == 2)//ручной ввод

{

system("cls");

hand();

vyvod();

printf(" Хотите сохранить результат в файл?\n");

printf(" Да-1\n");

printf(" Нет-2\n");

printf(" Выберите вариант - ");

scanf("%d", &choose\_number2);

printf(" \n");

while ((choose\_number2 < 1) || (choose\_number2 > 2))

{

printf(" Выбран вариант, которого нет в меню!!!\n ");

printf(" Да-1\n");

printf(" Нет-2\n");

printf(" Выберите вариант - ");

scanf("%d", &choose\_number2);

printf(" \n");

}

if (choose\_number2 == 1)

{

save();

}

printf("\n\n Стартовая вершина = ");

scanf("%d", &start);

while (start > n) {

printf(" Выбрана вершина, которой не существует!!!\n ");

printf("\n Стартовая вершина = ");

scanf("%d", &start);

}

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

bellman\_ford(n, start - 1);

system("pause");

continue;

}

if (choose\_number == 3)//завершение работы

{

break;

}

}

}