

Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Системы автоматизации проектирования»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проектированию

по курсу «Логика и основы алгоритмизации в инженерных
задачах»

на тему «Алгоритм Форда-Беллмана».

кор. А.
26.12.12

Выполнил:

студент группы 21ВВ2

Хабибулин А.М.

Принял:

Митрохин М.А.

Пенза 2022

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

«___» _____ 20__

ЗАДАНИЕ

на курсовое проектирование по курсу

"Логика и основы алгоритмизации в инженерных задачах"
Студенту Хабидуллину Айрату Маратовичу Группа 21 ВВ 2
Тема проекта Реализация алгоритма Форда - Беллмана

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения
в соответствии с данным заданием курсового проекта
Пояснительная записка должна содержать:

1. Постановку задачи
2. Теоретическую часть задания
3. Описание алгоритма поставленной задачи
4. Пример ручного расчета задачи и вычислений (на небольшом участке работы программы)
5. Описание самой программы
6. Тесты
7. Список литературы
8. Коды программы
9. Результаты работы программы

Объем работы по курсу

1. Расчетная часть

Ручной расчет работы алгоритма

2. Графическая часть

Схема алгоритма в формате блок-схемы

3. Экспериментальная часть

Тестирование программы

Результаты работы программы на тестовых данных

Срок выполнения проекта по разделам

- 1 Исследование теоретической части курсового к
- 2 Разработка алгоритмов программы к
- 3 Разработка программы к
- 4 Тестирование и завершение разработки программы к
- 5 Оформление пояснительной записки к
- 6
- 7
- 8

Дата выдачи задания "15" сентября 2021

Дата защиты проекта "26" 12 2021

Руководитель _____

Задание получил "15" сентября 2021 г.

Студент Хид _____

Содержание

Реферат.....	8
Введение	9
1 Постановка задачи.....	10
2 Теоретическая часть	11
3 Описание алгоритма программы	13
4 Описание программы	15
5 Тестирование	22
6 Ручной расчёт задачи	24
Заключение	26
Список литературы	27
Приложение А Листинг программы.....	28

Реферат

Отчет 34 стр, 13 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.

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.

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 расширении 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 2019, языки программирования C и C++.

2 Теоретическая часть

Граф G (рисунок 1) задается множеством вершин $X_1, X_2, \dots, X_n$. И множеством ребер, соединяющих между собой определенные вершины. Ребра из множества A ориентированы, что показывается стрелкой, которая указывает достижимость данной вершины, граф с такими ребрами называется ориентированным графом.

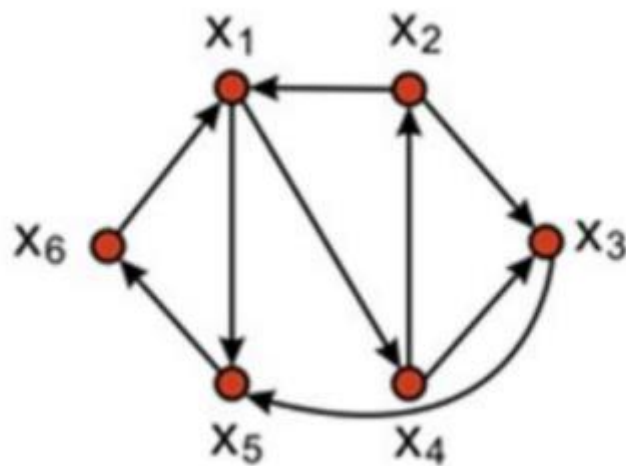


Рисунок 1 – Пример графа

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 $puti[u]$ значением $puti[v] + w$. 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 $puti[]$ 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 `start`. 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 `puti[0...n-1]`, который после отработки алгоритма будет содержать длины кратчайших путей. В начале заполняем массив следующим образом: начальная вершина, с которой начинается поиск, `puti[start] = 0`, все остальные элементы `puti[ ]` 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.

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.

Описание работы функции FordBellman().

1. Заполняем массив **puti** $[0 \dots n-1] = \text{inf}$.
2. Инициализируем **puti** $[\text{start}(\text{стартовая вершина})] = 0$.
3. Инициализируем **x**.
4. Для $i = 0$; пока $i < n-1$; делать $i = i + 1$.
5. $x = 0$
6. Флаг $\text{flag} = 0$.
7. Для $j = 0$; пока $j < e$ (кол-во рёбер); делать $j = j + 1$.
8. Если $\text{puti}[\text{edge}[j].u] > \text{puti}[\text{edge}[j].v] + \text{edge}[j].w$.
9. То $\text{puti}[\text{edge}[j].u] = \text{наибольшее из } \text{puti}[\text{edge}[j].v] + \text{edge}[j].w$.
10. $x = \text{edge}[j].u$
11. Если $\text{flag} \neq 1$
12. Выходим из цикла.
13. Если $x \neq 0$
14. Выводим сообщение об отрицательном цикле
15. Иначе
16. Выводим кратчайшие расстояния
17. Для $i = 0$; пока $i < n$; делать $i = i + 1$.
18. Вывод $\text{puti}[i]$.
19. Возврат **puti**.

4 Описание программы

Для написания данной программы использованы языки программирования Си и C++. Языки программирования Си и C++ - универсальные языки программирования, которые завоевали 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: `main()`, `menu()`, `menu1()`, `delVersh()`, `addVersh()`, `Read()`, `Save()`, `Edit()`, `EditMatrix()`, `createMatrix()`, `printMatrix()`, `FordBellman()`, `SizeOfMatrix()`.

Работа программы начинается с показа меню, где пользователь сам выбирает нужный ему пункт (см. рис. 2).

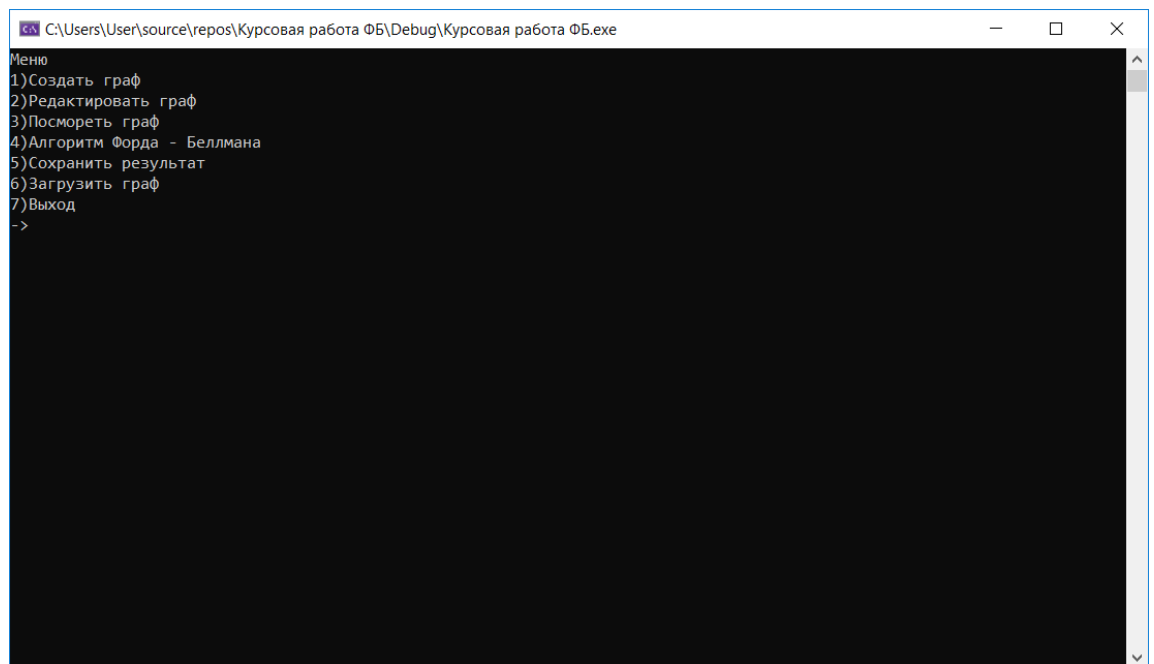


Рисунок 2 – Меню

1. Если пользователь выбрал пункт «1», то программа вызывает функцию `SizeOfMatrix()`, где просит ввести размер матрицы. После того, как пользователь ввёл размер, программа создаёт матрицу смежности **Matrica**, которые пока заполнены нулями (см. рис. 3).

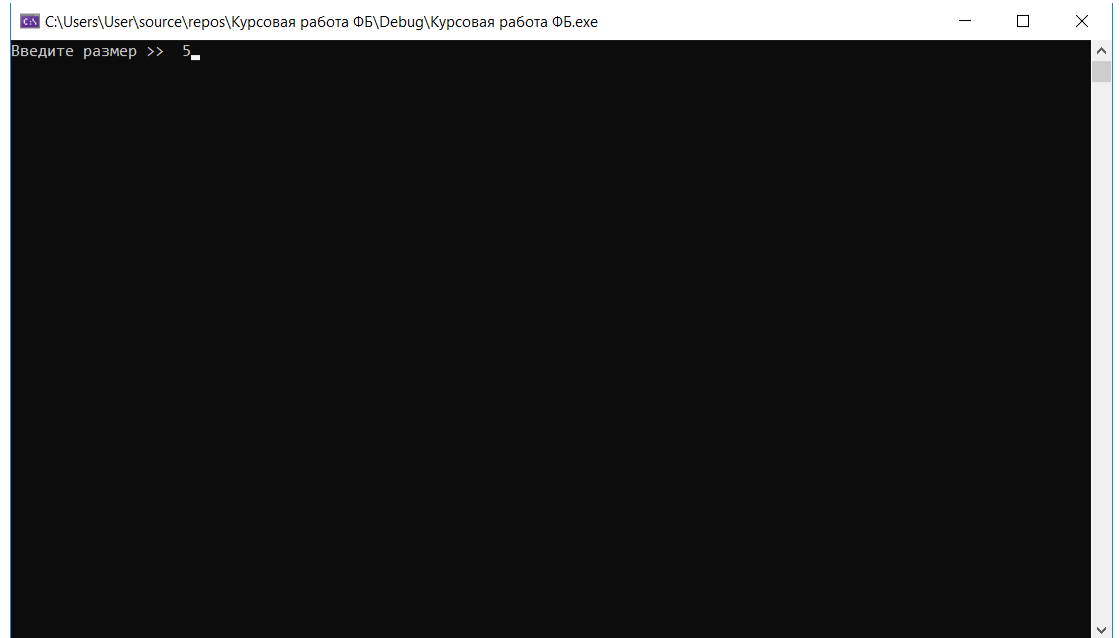


Рисунок 3 – Создание графа

2. Если пользователь выбрал пункт «2», то программа вызывает функцию `menu1()`, где пользователю снова показывается меню редактирования (см. рис. 4).

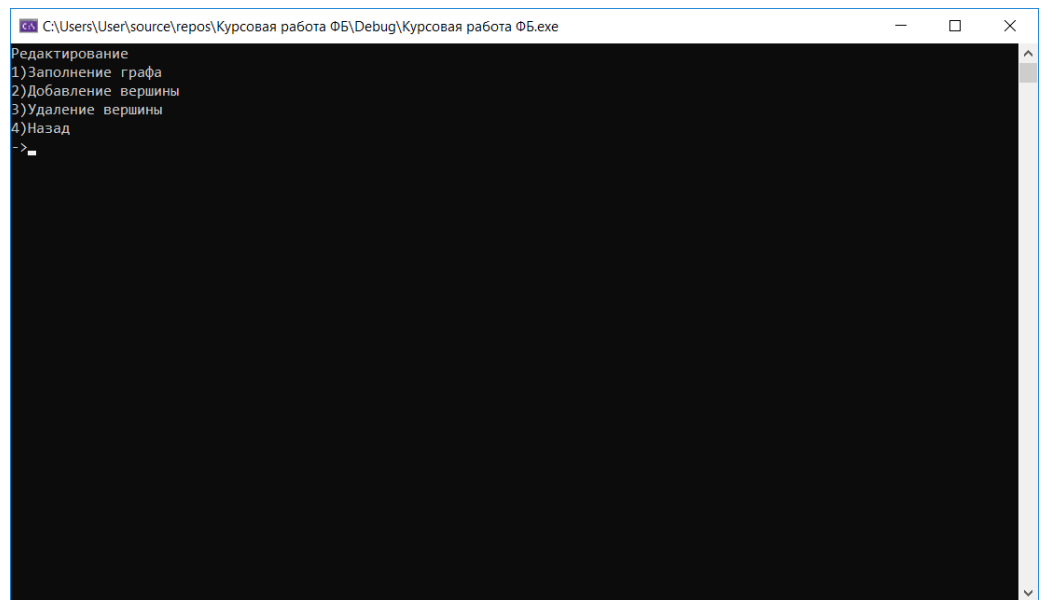


Рисунок 4 – Редактирование

- 2.1. Если пользователь выберет «1», то программа попросит пользователя указать как именно нужно заполнить матрицу смежности: автоматически или вручную. После этого, в зависимости от выбора, программа вызовет либо функцию EditMatrix(), либо Edit() (см. рис. 5).

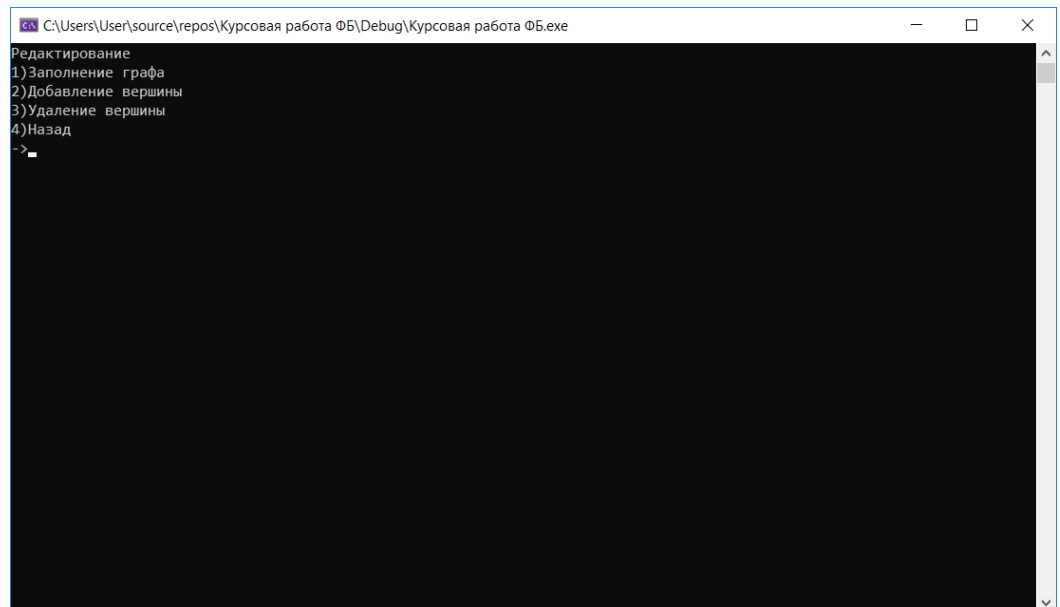
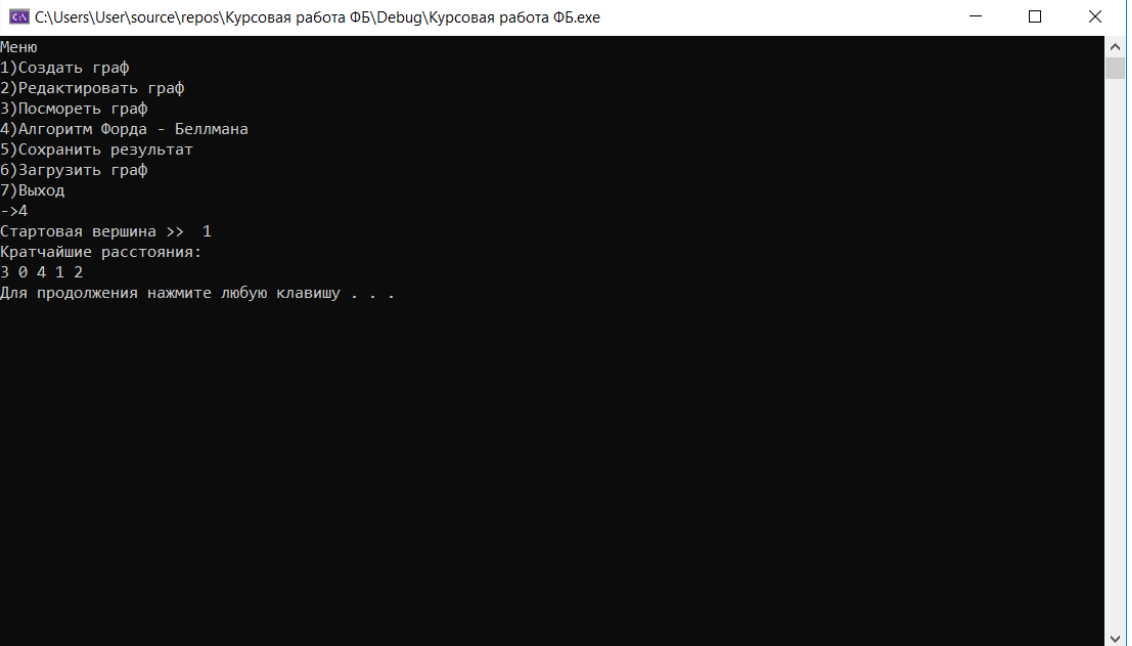


Рисунок 5 – Заполнение матрицы

- 2.2. Если пользователь выберет «2», то программа вызовет функцию SizeOfMatrix() и увеличит размерность матрицы на 1.
- 2.3. Если пользователь выберет «3», то программа спросит пользователя какую именно вершину он хочет удалить и вызовет функцию delVersh().
3. Если пользователь выберет «3», то программа вызовет функцию printMatrix(), и пользователь сможет увидеть свой граф, в виде матрицы смежности.

4. Если пользователь выберет «4», то программа вызовет функцию FordBellman() и, с помощью алгоритма Форда - Беллмана, высчитает кратчайшие расстояния от выбранной вершины до всех остальных и представит результат в виде списка кратчайших расстояний. После этого, на экран выводится сам список кратчайших расстояний **puti**. (см. рис. 6).



```
C:\Users\User\source\repos\Курсовая работа ФБ\Debug\Курсовая работа ФБ.exe
Меню
1)Создать граф
2)Редактировать граф
3)Посмотреть граф
4)Алгоритм Форда - Беллмана
5)Сохранить результат
6)Загрузить граф
7)Выход
->4
Стартовая вершина >> 1
Кратчайшие расстояния:
3 0 4 1 2
Для продолжения нажмите любую клавишу . . .
```

Рисунок 6 – Алгоритм Форда - Беллмана

5. Если пользователь выберет «5», то программа вызовет функцию Save() и сохранит результат в файл «Graph» (см. рис. 7).

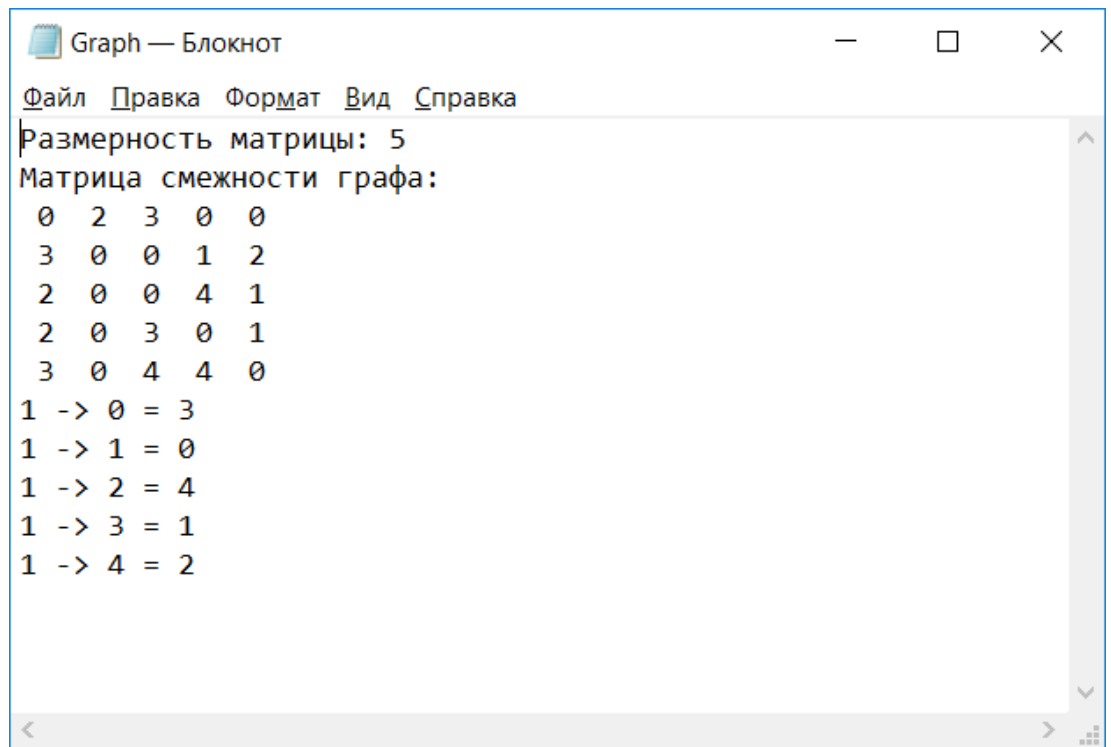


Рисунок 7 - Сохранение результата

6. Если пользователь выберет «б», то программа вызовет функцию `Read()` и из файла «Graph» возьмутся данные для того, чтобы заполнить матрицу смежности **Matrix** и список рёбер **edge** (см. рис. 8).

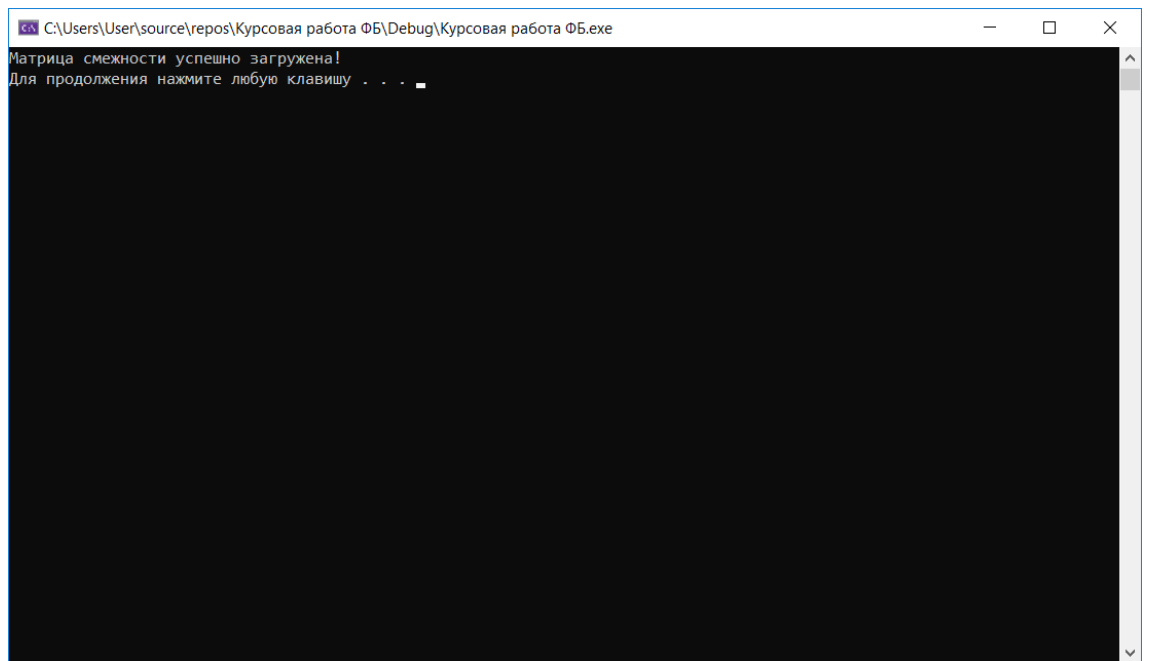


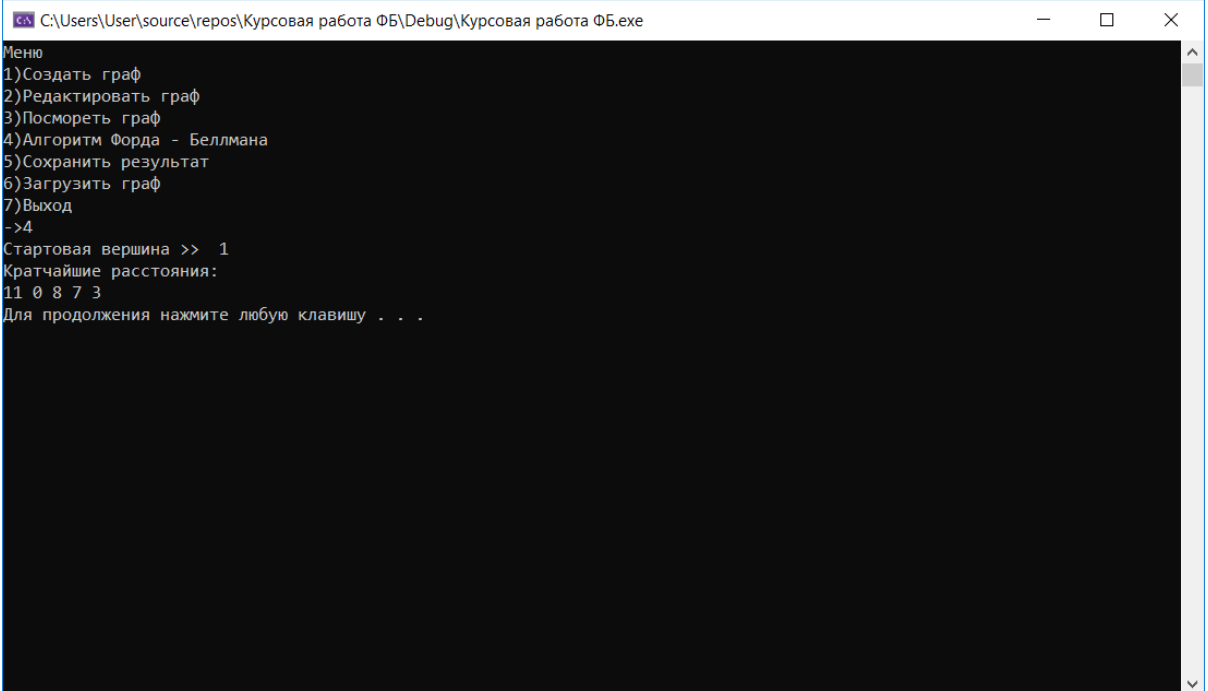
Рисунок 8 – Чтение из файла

5 Тестирование

Среда разработки Microsoft Visual Studio 2019 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

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

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и вывод результата алгоритма Форда – Беллмана. (см. рис. 9,10,11,12).



```
C:\Users\User\source\repos\Курсовая работа ФБ\Debug\Курсовая работа ФБ.exe
Меню
1)Создать граф
2)Редактировать граф
3)Посмотреть граф
4)Алгоритм Форда - Беллмана
5)Сохранить результат
6)Загрузить граф
7)Выход
->4
Стартовая вершина >> 1
Кратчайшие расстояния:
11 0 8 7 3
Для продолжения нажмите любую клавишу . . .
```

Рисунок 9 – Результат работы (5 вершин)

```
C:\Users\User\source\repos\Курсовая работа ФБ\Debug\Курсовая работа ФБ.exe
Меню
1)Создать граф
2)Редактировать граф
3)Посмотреть граф
4)Алгоритм Форда - Беллмана
5)Сохранить результат
6)Загрузить граф
7)Выход
->4
Стартовая вершина >> 1
Кратчайшие расстояния:
7 0 6 6 2 1 2 2 5 4
Для продолжения нажмите любую клавишу . . .
```

Рисунок 10 – Результат работы (10 вершин)

```
C:\Users\User\source\repos\Курсовая работа ФБ\Debug\Курсовая работа ФБ.exe
Меню
1)Создать граф
2)Редактировать граф
3)Посмотреть граф
4)Алгоритм Форда - Беллмана
5)Сохранить результат
6)Загрузить граф
7)Выход
->4
Стартовая вершина >> 1
Кратчайшие расстояния:
6 0 12 7 14 10 7 9 8 16 10 12 3 14 13
Для продолжения нажмите любую клавишу . . .
```

Рисунок 11 – результат работы (15 вершин)

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

6 Ручной расчёт задачи

Проведем проверку программы посредством ручных вычислений на примере ориентированного графа с 5 вершинами и графа с отрицательным циклом, где 4 вершины. (рис. 12, 13).

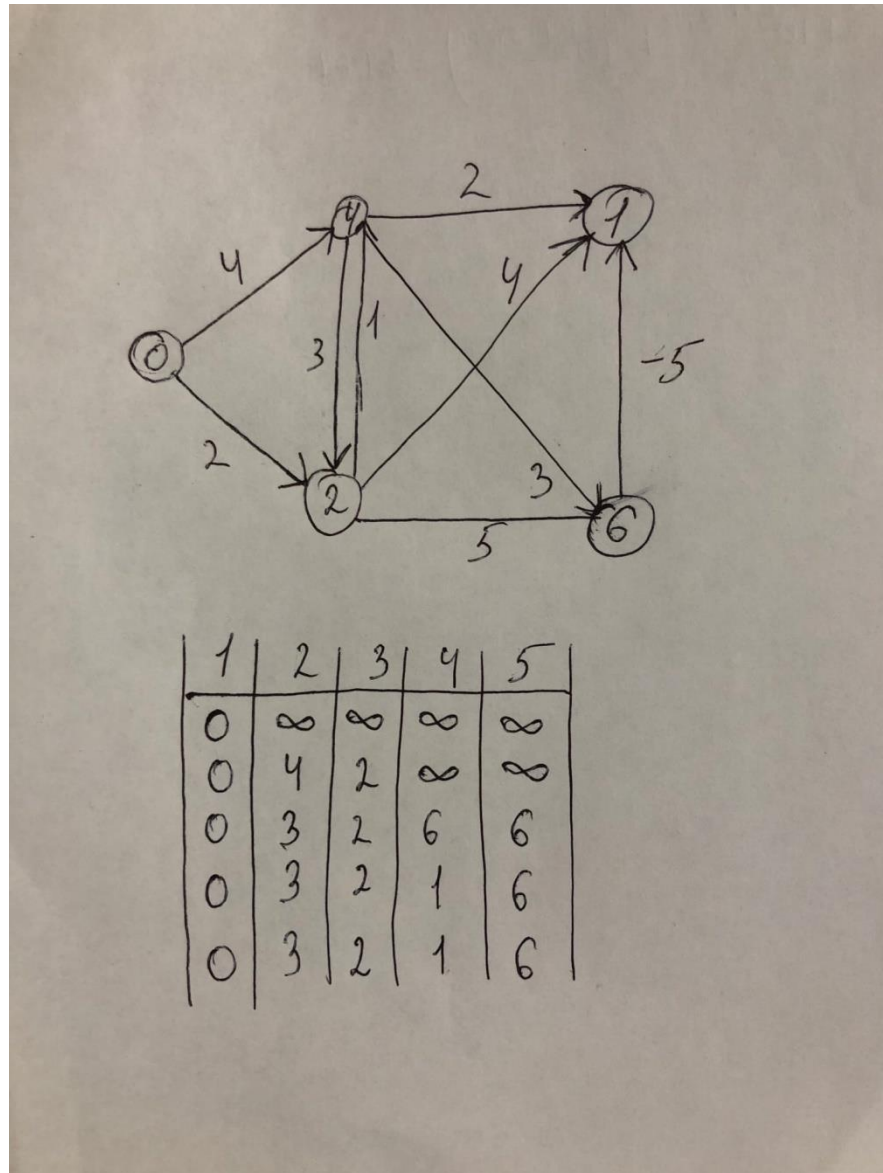


Рисунок 12 – Ориентированный граф.

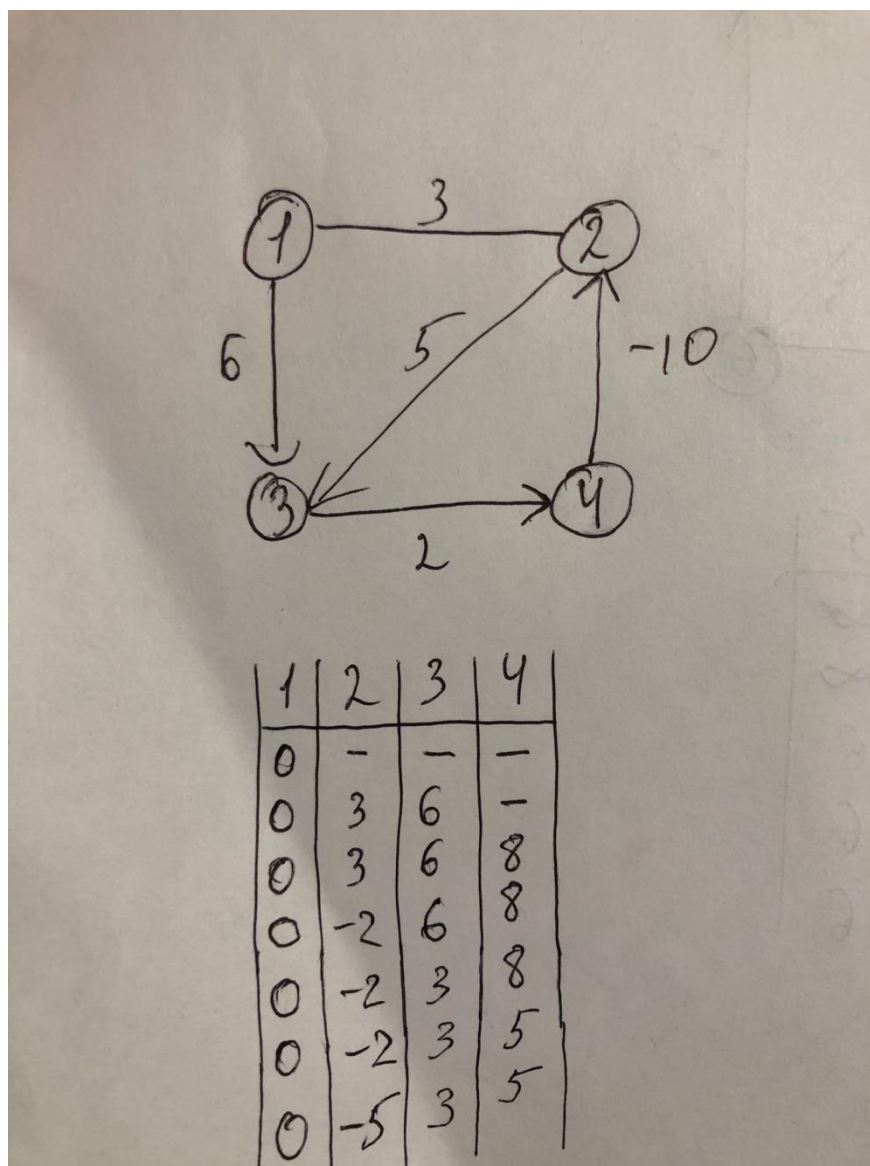


Рисунок 13 – Граф с отрицательным циклом.

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 литературы

1. https://e-maxx.ru/alg/ford_bellman
2. Алгоритм Беллмана — Форда. (kvodo.ru)
3. <https://habr.com/ru/company/otus/blog/484382/>
4. Онлайн-школа Фоксфорд (foxford.ru)
5. MAXimal :: algo :: Нахождение кратчайших путей между заданной вершиной и всеми остальными вершинами алгоритмом Форда-Беллмана за $O(N M)$ (e-maxx.ru)
6. https://ru.wikipedia.org/wiki/Алгоритм_Беллмана_—_Форда.

Приложение А

Листинг программы.

```
#define _CRT_SECURE_NO_WARNINGS
#define _CRT_NONSTDC_NO_WARNINGS
#define HEADER ("Курсовая работа\nТема: Алгоритм Форда
- Беллмана.\nВыполнил: студент группы 21BB2 Хабибулин
А.М.\n")
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <time.h>
#include <conio.h>
#include <Windows.h>
#include <queue>
#define inf 10000; //превосходит все возможные значения
путей

using namespace std;

int** Matrix, **Vis, n, puti[1000], e, x; //n-кол-во
вершин, e-кол-во ребер

struct Edges {
    int v, u, w;
};
Edges edge[10000];

int SizeOfMatrix() {

    int n;
    cout << "Введите размер >> " << " ";
    cin >> n;
    return n;
}

int* FordBellman(int start) // функция алгоритм Форда-
Беллмана
{
```

```

    for (int i = 0; i < n; i++)
    {
        puti[i]= inf;
    }
    puti[start]= 0;

    for (int i = 0; i < n; i++)
    {
        x = 0;
        bool stop = false;
        for (int j = 0; j < e; j++)
            if (puti[edge[j].v] + edge[j].w <
puti[edge[j].u]) {
                puti[edge[j].u] = puti[edge[j].v] +
edge[j].w;
                x = edge[j].u;
                stop = true;
            }
        if (!stop) break;
    }

    if (x != 0) {
        cout << " Ошибка, найден отрицательный цикл" <<
endl;
    }
    else {
        cout << "Кратчайшие расстояния: " << endl;

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

            cout << puti[i] << " ";

        }

        cout << endl;
    }

    return puti;
}

int** createMatrix() // функция создания матрицы
{

```

```

    int** M = (int**) (malloc(n * sizeof(int*))); //
    выделение памяти под матрицу
    for (int i = 0; i < n; i++)
    {
        M[i] = (int*) (malloc(n * sizeof(int)));
        for (int j = 0; j < n; j++)
        {
            M[i][j] = 0; // заполнение матрицы 0
        }
    }
    return M;
}

void printMatrix(int** Matrica) //функция вывода
матрицы на экран
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (Matrica[i][j] == SHRT_MAX)
            {
                cout << NULL << " ";
            }
            else
            {
                cout << Matrica[i][j] << " ";
            }
        }
        cout << endl;
    }
}

void EditMatrix() // функция ручного заполнения матрицы
{
    cout << endl << endl << "Введите расстояния путей
    между вершинами:" << endl;
    for (int row = 0; row < n; row++) {

        for (int col = 0; col < n; col++) {
            cout << endl << "[" << row + 1 << "]" -> ["
<< col + 1 << "]" = ";
            cin >> Matrix[row][col];
        }
    }
}

```

```

    }
}
printMatrix(Matrix);
e = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
    {
        if (Matrix[i][j] != 0) // передача значений
в список ребер
        {
            edge[e].v = i;
            edge[e].u = j;
            edge[e].w = Matrix[i][j];
            e++;
        }
    }
}

}

void Edit() // функция рандомного заполнения матрицы
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i == j)
            {
                Matrix[i][j] = 0;
            }
            else if (rand() % 100 < 50)
            {
                Matrix[i][j] = SHRT_MAX;
            }
            else
            {
                Matrix[i][j] = (rand() % n) + 1;
            }
        }
    }
    e = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
        {

```

```

        if (Matrix[i][j] != 0) // передача значений
в список ребер
        {
            edge[e].v = i;
            edge[e].u = j;
            edge[e].w = Matrix[i][j];
            e++;
        }
    }
}

```

```

void Save(int** Matrica, int* puti, int start) //
сохранение в файл
{
    FILE* G = fopen("Graph", "w");
    fprintf(G, "Размерность матрицы: %d\n", n);
    fprintf(G, "Матрица смежности графа:\n");
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (Matrica[i][j] == SHRT_MAX)
            {
                fprintf(G, " %d ", NULL);
            }
            else
            {
                fprintf(G, " %d ", Matrica[i][j]);
            }
        }
        fprintf(G, "\n");
    }
    if (x != 0) {
        fprintf(G, "Найден отрицательный цикл");
        fclose(G);
        return;
    }
    else {
        for (int i = 0; i < n; i++) {
            if (puti[i] == 10000)

```

```

        fprintf(G, "%d -> %d = Пути не
существует\n", start, i);
    else {
        fprintf(G, "%d -> %d = %d\n", start,
i, puti[i]);
    }
}
fclose(G);
}

FILE* Gg = fopen("GraphMatrix", "w");
fprintf(Gg, "%d\n", n);
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        if (Matrica[i][j] == SHRT_MAX)
        {
            fprintf(Gg, "%d ", NULL);
        }
        else
        {
            fprintf(Gg, "%d ", Matrica[i][j]);
        }
    }
    fprintf(Gg, "\n");
}
fclose(Gg);
}

void Read() // чтение из файла
{
    FILE* Gg;
    if (!fopen("Graph", "r"))
    {
        printf("Невозможно открыть файл!\n");
        system("pause");
        return;
    }
    Gg = fopen("GraphMatrix", "r");
    fscanf(Gg, "%d", &n);
    Matrix = createMatrix();

```

```

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            fscanf(Gg, " %d ", &Matrix[i][j]);
        }
    }
    fclose(Gg);
    e = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
        {
            if (Matrix[i][j] != 0) // передача значений
в список ребер
            {
                edge[e].v = i;
                edge[e].u = j;
                edge[e].w = Matrix[i][j];
                e++;
            }
        }
    }
}

void addVersh() // функция добавления вершины в граф
{
    Matrix = (int**) (realloc(Matrix, (n + 1) *
sizeof(int**)));
    for (int i = 0; i < n; i++)
    {
        Matrix[i] = (int*) (realloc(Matrix[i], (n + 1) *
sizeof(int)));
    }
    Matrix[n] = (int*) (malloc((n + 1) * sizeof(int)));
    n++;
    for (int i = 0; i < n; i++)
    {
        Matrix[n - 1][i] = 0;
        Matrix[i][n - 1] = 0;
    }
}

void delVersh(int v)

```

```

{
    for (int i = v; i < n - 1; i++)
    {
        for (int j = 0; j < n; j++)
        {
            Matrix[i][j] = Matrix[i + 1][j];
        }
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = v; j < n - 1; j++)
        {
            Matrix[i][j] = Matrix[i][j + 1];
        }
    }
    for (int i = 0; i < n; i++)
    {
        Matrix[i] = (int*)(realloc(Matrix[i], (n - 1) *
sizeof(int)));
    }
    free(Matrix[n - 1]);
    n--;
}

void menu1() {

    int k, nom;
    do
    {
        system("cls");
        cout << "Редактирование\n1) Заполнение
графа\n2) Добавление вершины\n3) Удаление
вершины\n4) Назад\n->";
        cin >> k;
        switch (k)
        {
            case (1):

                system("cls");
                cout << "Заполнение
матрицы?\n1) Ручное\n2) Автоматическое\n3) Назад\n->";
                cin >> nom;
                if (nom == 1)

```

```

        {
            EditMatrix();
        }
    else if (nom == 2)
    {
        Edit();
        system("cls");
        cout << "Успех" << endl;
        system("pause");
    }
    else if (nom == 3)
    {
        return;
    }
    else
    {
        cout << "Ошибка" << endl;
        system("pause");
    }break;

case(2):

    system("cls");
    addVersh();
    cout << "Успех" << endl;
    system("pause");
    break;

case(3):

    system("cls");
    cout << "Номер удаляемой вершины >> " <<
endl;
    cin >> nom;
    if (nom < k)
    {
        delVersh(nom);
    }
    else
    {
        system("cls");
        cout << "Вершины нет" << endl;
        system("pause");
    }

```

```

        return;
    }
    break;

case (4) :

    return;

default:

    system("cls");
    cout << "Ошибка" << endl;
    system("pause");
    break;

}

} while (k != 4);

}

void menu()
{
    int k;
    do {
        system("cls");
        cout << "Меню\n1) Создать граф\n2) Редактировать
граф\n3) Посмотреть граф\n4) Алгоритм Форда -
Беллмана\n5) Сохранить результат\n6) Загрузить
граф\n7) Выход\n->";
        cin >> k;
        switch (k)
        {
            case (1) :

                system("cls");
                n = SizeOfMatrix();
                Matrix = createMatrix();
                Vis = createMatrix();
                break;

            case (2) :

```

```

        menu1();
        break;

    case(3):

        system("cls");
        cout << "Матрица смежности:" << endl;
        printMatrix(Matrix);
        system("pause");
        break;

    case(4):
        int start;
        cout << "Стартовая вершина >> " << " ";
        cin >> start;
        FordBellman(start);
        system("pause");
        break;

    case(5):

        system("cls");
        cout << "Результат сохранён в файл Graph."
<< endl;

        Save(Matrix, puti, start);
        system("pause");
        break;

    case (6):

        system("cls");
        Read();
        cout << "Матрица смежности успешно
загружена!" << endl;
        system("pause");
        break;

    case(7):

        exit(NULL);

    default:
        system("cls");

```

```

        cout << "ERROR" << endl;
        system("pause");
        break;
    }
} while (k != 7);
}

int main() {

    setlocale(LC_ALL, "rus");
    srand(time(NULL));
    cout << HEADER;
    system("pause");
    menu();
    system("pause");
    return 0;
}

```