Выполнил:

студент группы 19ВВ2

Павлов А.А.

Принял:

д.т.н. доцент Митрохин М.А.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовому проектированию

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация алгоритма Форда-Беллмана»

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Пензенский государственный университет

Кафедра «Вычислительная техника»

Пенза 2020

**ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**Факультет вычислительной техники

Кафедра “Вычислительная техника”

“УТВЕРЖДАЮ”

Зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_20\_\_

**ЗАДАНИЕ**

на курсовое проектирование по курсу

Логика и основы алгоритмизации в инженерных задачах

**Студенту** Павлову Алексею Александровичу **Группы**    19ВВ2

**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 расчёта задачи и вычислений (на небольшом участке работы алгоритма);

     5. Описание самой программы;

     6. Тесты;

     7. Список литературы;

     8. Листинг программы;

     9. Результаты работы программы;

**Расчетная часть**

Ручной расчёт работы алгоритма.

**Графическая часть**

Схема алгоритма в формате блок - схем.

**Экспериментальная часть**

Тестирование программы                                                                                         Результаты работы программы на тестовых данных.

**Срок выполнения проекта по разделам**

В соответствии с графиком выполнения курсового проекта.

Дата выдачи задания “16” сентября 2020 г.

Дата защиты проекта “ ”              202 г.

**Руководитель**

**Задание получил** “16” сентября 2020 г.

**Студент** Павлов А.А.

Содержание

1. Постановка задачи 6
2. Теоретическая часть7
3. Описание алгоритма поставленной задачи 9
4. Ручной расчёт12
5. Описание программы. 15
6. Тестирование 26
7. Заключение 30
8. Список используемых источников 31
9. Приложение. Листинг программы 32

# 

# 1. Постановка задачи

Необходимо разработать программу, которая будет искать длины кратчайших путей от вершины st до всех остальных вершин, используя алгоритм Беллмана-Форда.

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.

# 2. Теоретическая часть

Граф G (рисунок 1) задается множеством вершин 1, 2, ..., 5(N). 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.

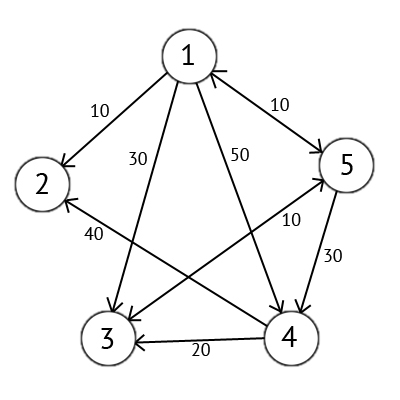


Рисунок 1 – Пример орграфа

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.

**Алгоритм Беллмана-Форда** — алгоритм поиска кратчайшего пути во взвешенном графе. За время O(|st|\*|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.

Дан ориентированный или неориентированный граф **G** со взвешенными рёбрами. Длиной пути назовём сумму весов рёбер, входящих в этот путь. Требуется найти кратчайшие пути от выделенной вершины **st** до всех вершин графа.

Заметим, что кратчайших путей может не существовать. Так, в графе, содержащем цикл с **отрицательным суммарным весом**, существует сколь угодно короткий путь от одной вершины этого цикла до другой (каждый обход цикла уменьшает длину пути). Цикл, сумма весов рёбер которого отрицательна, называется **отрицательным циклом**.

Сам алгоритм Форда-Беллмана представляет из себя несколько фаз. На каждой фазе просматриваются все рёбра графа, и алгоритм пытается произвести релаксацию (relax, ослабление) вдоль каждого ребра **(v,u)** веса **w**. Релаксация вдоль ребра — это попытка улучшить значение nodes[u] значением nodes[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 nodes[] останется равным бесконечности (заведомо большим числом).

# 3. Описание алгоритма поставленной задачи

Для алгоритма Форда-Беллмана, в отличие от многих других графовых алгоритмов, более удобно представлять граф в виде одного списка всех рёбер. В приведённой реализации заводится массив структур данных **edge** для всех рёбер. Входными данными для алгоритма являются числа **N** (кол-во вершин), **e** (кол-во рёбер), список **edge** рёбер, и номер стартовой вершины **st**. Все номера вершин нумеруются с 0 по N-1. Константа **inf** обозначает число "бесконечность" — её надо подобрать таким образом, чтобы она заведомо превосходила все возможные длины путей. Массив расстояний **nodes[0…N-1]**, который после отработки алгоритма будет содержать ответ на задачу. В начале работы мы заполняем его следующим образом: nodes[st] = 0, а все остальные элементы nodes[] 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 (**change**) того, изменилось что-то на текущей фазе или нет, и если на какой-то фазе ничего не произошло, то алгоритм можно останавливать.

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 **parent[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 parent[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. Следовательно, в момент улучшения нам надо просто запоминать в parent[], из какой вершины это улучшение произошло.

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 st до i, но в обратном порядке, поэтому мы вызываем **reverse** от него и затем выводим.

При восстановлении пути может возникнуть момент, когда после первой итерации мы попадаем в вершину, в которой уже были до этого, это свидетельствует о наличие отрицательного цикла. В этом случае останавливаем восстановление пути 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:

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

Ниже представлен псевдокод функции **bellman\_ford()**

1. Открываем/создаём файл bellman\_ford.txt для сохранения результатов.

2. В случае невозможности открытия/создания файла останавливаем программу.

3. Заполняем массив **nodes[0…N-1] = inf**.

4. Инициализируем вектор **parent[0…N-1] = -1**.

5. nodes[**st**(стартовая вершина)] = 0.

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

7. Флаг **change = 0**.

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

9. Если nodes[edge[j].v] + edge[j].w < nodes[edge[j].u].

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

11. parent[edge[j].u] = edge[j].v.

12. change = 1.

13. Если change == 0

14. Выходим из цикла.

15. Для i = 0; пока i < N; i = i + 1.

16. Если nodes[i] == inf

17. Выводим в консоль и файл, что «Путь отсутствует».

18. Иначе

19. Выводим в консоль и файл nodes[i]

20. Если 0 <= nodes[i] < 10

21. Выводим в консоль и файл Пробел.

22. Объявляем вектор **path**.

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

24. Для i2 = 0; пока i2 < размер path; делать i2 = i2 + 1.

25. Если cur == path[i2] и размер path > 1.

26. Увеличиваем на 1 размер path и заносим в конец cur.

27. Выводим предупреждение об отрицательном цикле.

28. Переходим на метку **negative.**

29. Увеличиваем на 1 размер path и заносим в конец cur.

30. Метка **negative:**

31. Меняем порядок элементов path[] на обратный.

32. Цикл вывода path[].

33. Закрываем файл bellman\_ford.txt

# 4. Ручной расчёт.

Проведем проверку программы посредством ручных вычислений на

примере взвешенного ориентированного графа с 5 вершинами (рисунок 2).

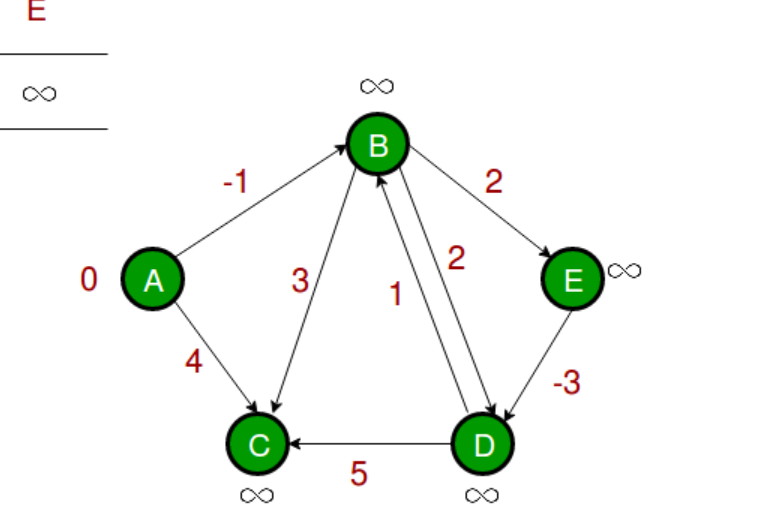


Рисунок 2. Орграф.

Пусть ребра отрабатываются в следующем порядке: (B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D). Мы получаем следующие расстояния, когда проход по ребрам был совершен первый раз. Первая строка показывает начальные расстояния, вторая строка показывает расстояния, когда ребра (B, E), (D, B), (B, D) и (A, B) обрабатываются. Третья строка показывает расстояние при обработке (A, C). Четвертая строка показывает, что происходит, когда обрабатываются (D, C), (B, C) и (E, D).

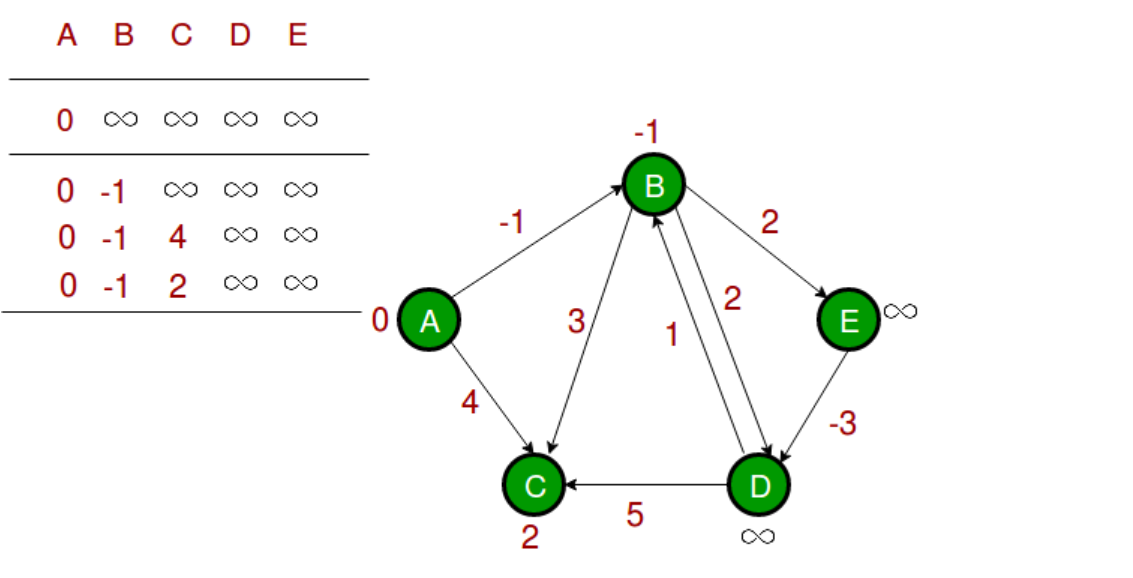


Рисунок 2.1 – Результат первой итерации.

Первая итерация гарантирует, что все самые короткие пути будут не длиннее пути в 1 ребро. Мы получаем следующие расстояния, когда будет совершен второй проход по всем ребрам (в последней строке показаны конечные значения).

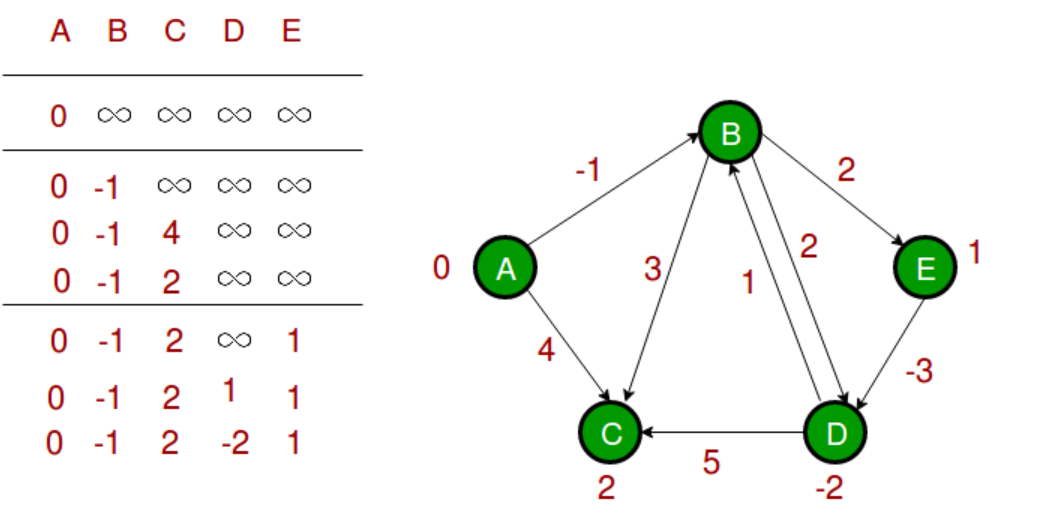


Рисунок 2.2 – Результат второй итерации.

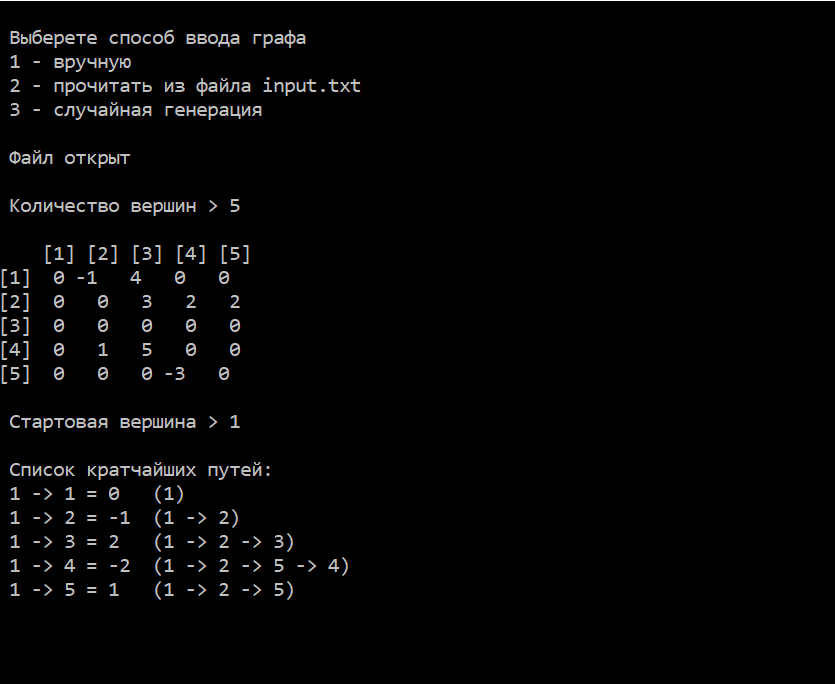


Рисунок 2.3 – Обработка того же графа в программе.

Таким образом, можно сделать вывод, что программа работает верно.

# 5. Описание программы

Для написания данной программы использованы языки программирования С и C++. C - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней. Синтаксис C++ унаследован от языка C. Одним из принципов разработки было сохранение совместимости с C.

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

Данная программа является многомодульной, поскольку состоит из нескольких функций: main(), generation(), bellman\_ford().

Работа программы начинается с запроса генерации матрицы. Пользователю даётся на выбор 3 действия. Для выбора действия пользователю нужно нажать соответствующую клавишу на клавиатуре.

1. Ввести матрицу вручную.

2. Прочитать матрицу из файла.

3. Сгенерировать случайную матрицу.



Рисунок 3 – выбор способа ввода матрицы.

input:

cout << "\n Выберете способ ввода графа";

cout << "\n 1 - вручную";

cout << "\n 2 - прочитать из файла input.txt";

cout << "\n 3 - случайная генерация";

input2:

c = \_getch();

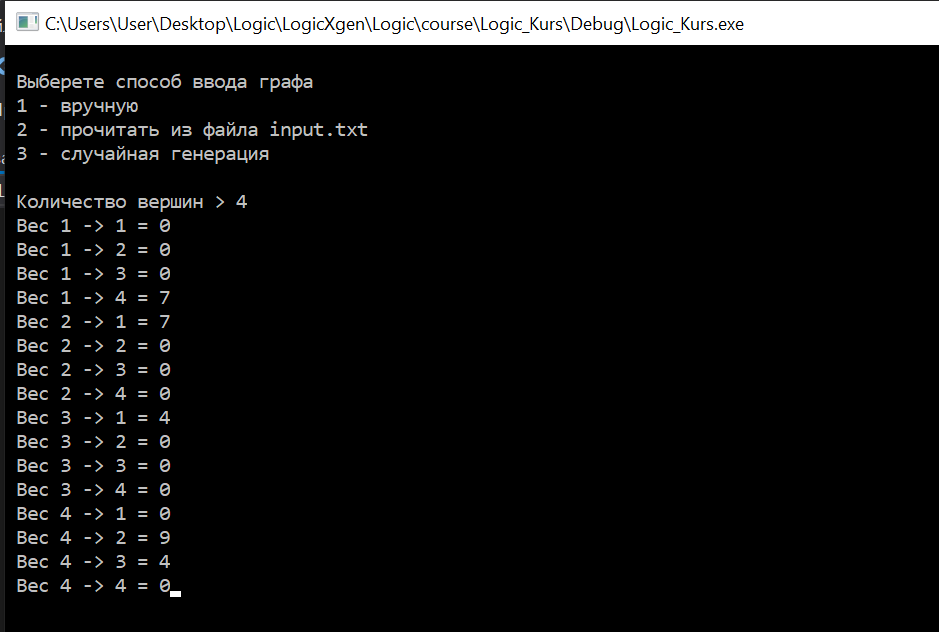


Рисунок 4 – ручной ввод матрицы.

if (c == 49)

{

cout << "\n\n Количество вершин > "; cin >> N;

nodes.resize(N);

M1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

M1[i] = (int\*)malloc(N \* sizeof(int));

}

e = 0;

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

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

{

cout << " Вес " << i + 1 << " -> " << j + 1 << " = "; cin >> w;

M1[i][j] = 0;

if (w != 0)

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = w;

M1[i][j] = w;

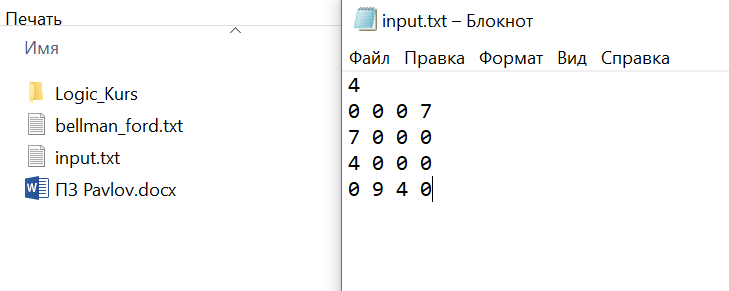
e++;

}

}

cout << "\n";

}



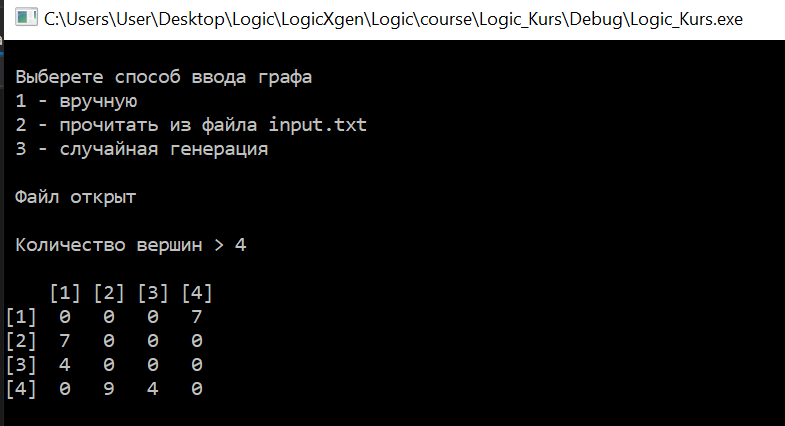


Рисунок 5 – ввод через файл.

else if (c == 50) // считывание из файла

{

ifstream fin;

fin.open("../../input.txt");

if (!fin.is\_open())

{

cout << "\n\n Ошибка открытия файла";

cout << "\n Проверьте существование файла input.txt";

cout << "\n Для корректной работы программы файл должен быть заполнен в таком виде:";

cout << "\n Первая строка - кол-во вершин";

cout << "\n Начиная со второй - матрица смежности";

cout << "\n\n Пример заполнения:";

cout << "\n 3 ";

cout << "\n 0 8 0 ";

cout << "\n 3 0 0 ";

cout << "\n 4 0 0\n ";

fout.close();

fout.open("../../input.txt");

fout.close();

goto input;

}

cout << "\n\n Файл открыт";

fin >> N; // число строк

cout << "\n\n Количество вершин > " << N << endl;

nodes.resize(N);

M1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

M1[i] = (int\*)malloc(N \* sizeof(int));

}

//Считаем матрицу из файла

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

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

fin >> M1[i][j];

fin.close();

e = 0;

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

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

{

if (M1[i][j] != 0)

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = M1[i][j];

e++;

}

}

cout << "\n";

}

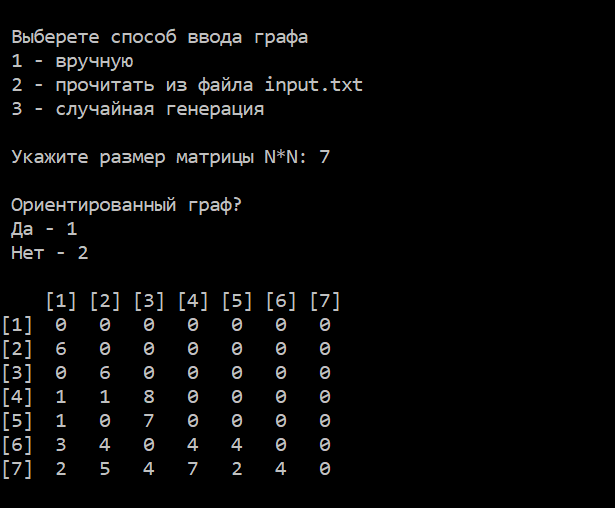


Рисунок 6 – случайная генерация.

else if (c == 51)

{

cout << "\n";

generation();

e = 0;

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

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

{

if (M1[i][j] != 0)

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = M1[i][j];

e++;

}

}

}

void generation() // генерация взвешанных графов

{

int naprav = 0;

char c;

setlocale(LC\_ALL, "russian");

printf("\n Укажите размер матрицы N\*N: ");

scanf("%d", &N);

printf("\n Ориентированный граф?");

printf("\n Да - 1 ");

printf("\n Нет - 2\n\n");

c = \_getch();

if(c == 49)

naprav = 1;

M1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

M1[i] = (int\*)malloc(N \* sizeof(int));

}

nodes.resize(N);

srand(time(NULL));

int ch = 0;

int ch2 = 0;

for (int i = 0; i < N; i++) // генерация матрицы M1 (взвешанный неориентированный)

{

M1[i][i] = 0;

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

{

ch = rand() % 10;

ch2 = rand() % 100;

if (ch < 7)

{

if (ch2 < 1)

ch = -ch;

M1[i][j] = ch;

}

else

M1[i][j] = 0;

M1[j][i] = M1[i][j];

}

}

M2 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

M2[i] = (int\*)malloc(N \* sizeof(int));

}

for (int i = 0; i < N; i++) // генерация матрицы M2 (взвешанный ориентированный граф)

{

M2[i][i] = 0;

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

{

ch = rand() % 15;

ch2 = rand() % 100;

if (ch < 12)

{

if (ch > 9)

ch = ch - 6;

if (ch2 < 1)

ch = -ch;

M2[i][j] = ch;

M2[j][i] = 0;

}

else

{

M2[i][j] = 0;

}

}

}

if(naprav == 1)

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

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

M1[i][j] = M2[i][j];

for (int i = N - 1; i > 0; i--)

{ // очищение памяти

free(M2[i]);

}

free(M2);

}

При нажатии альтернативной клавиши указатель вернётся к месту ввода.

else

goto input2;

Далее выводится введённая матрица.

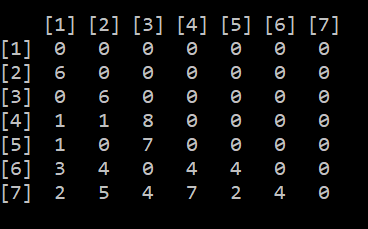


Рисунок 7 – вывод матрицы.

///// вывод матрицы смежности

printf(" ");

fout << " ";

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

{

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

fout << "[" << i + 1 << "] ";

}

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

{

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

fout << "\n[" << i + 1 << "]";

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

{

if ((i + 1) < 10)

{

if(M1[i][j] < 0)

cout << '\b';

if (j < 10)

{

printf("%3d ", M1[i][j]);

fout << setw(3) << M1[i][j] << " ";

}

else

{

printf(" %3d ", M1[i][j]);

fout << " " << setw(3) << M1[i][j] << " ";

}

}

else if ((i + 1) < 100)

{

if (j == 0)

{

printf(" %d", M1[i][j]);

fout << " " << M1[i][j];

}

else if (j < 10)

{

printf(" %d", M1[i][j]);

fout << " " << M1[i][j];

}

else

{

printf(" %d", M1[i][j]);

fout << " " << M1[i][j];

}

}

}

}

/////

Затем вызывается функция bellman\_ford(), где и выполняется основной алгоритм программы.

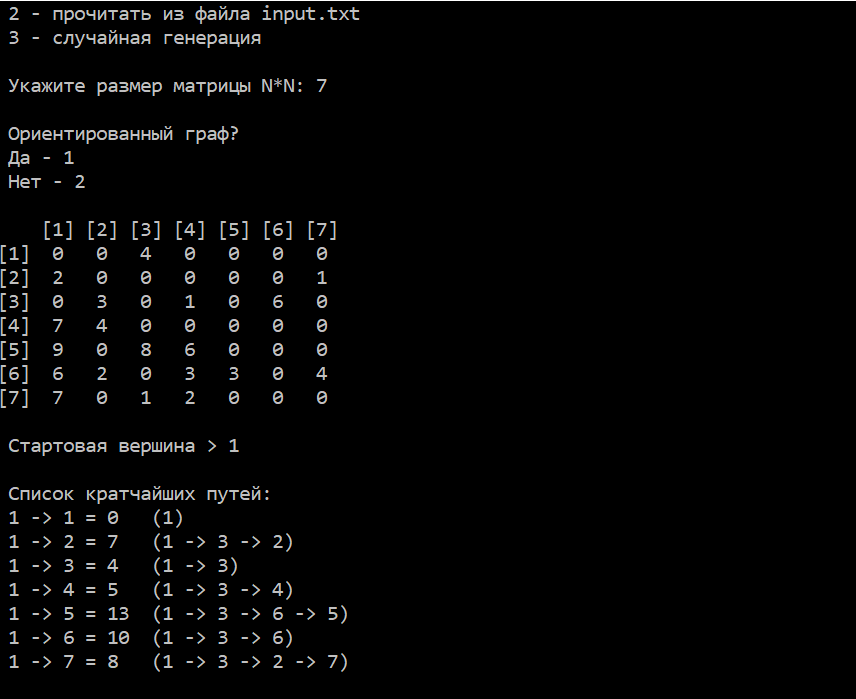


Рисунок 8 – результат работы.

int i, j;

ofstream fout;

fout.open("../../bellman\_ford.txt", ios::app);

if (!fout.is\_open())

{

cout << "\n Ошибка открытия файла";

\_Exit(EXIT\_SUCCESS);

}

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

nodes[i] = inf;

vector<int> parent(N, -1);

nodes[st] = 0;

int x;

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

{

bool change = false;

for (j = 0; j < e; j++) // v -> u, w - вес

if (nodes[edge[j].v] + edge[j].w < nodes[edge[j].u])

{

nodes[edge[j].u] = max(-inf, nodes[edge[j].v] + edge[j].w); // защита от переполнения

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

change = true;

}

if (!change) // если итерация не дала изменений, релаксации не продолжаются

break;

}

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

{

if (nodes[i] == inf)

{

cout << endl << " " << st + 1 << " -> " << i + 1 << " = " << "Путь отсутствует ";

fout << endl << " " << st + 1 << " -> " << i + 1 << " = " << "Путь отсутствует ";

}

else

{

cout << endl << " " << st + 1 << " -> " << i + 1 << " = " << nodes[i];

fout << endl << " " << st + 1 << " -> " << i + 1 << " = " << nodes[i];

// Восстановление пути

if (nodes[i] >= 0 && nodes[i] < 10)

{

cout << " ";

fout << " ";

}

vector<int> path; // вектор для восстановления пути

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

{

for (int i2 = 0; i2 < path.size(); i2++)

if (cur == path[i2] && path.size() > 1) // если после первой итерации попадаем в уже посещённую

{ // вершину - отрицательный цикл

path.push\_back(cur);

cout << " (Отрицательный цикл)";

fout << " (Отрицательный цикл)";

goto negative;

}

path.push\_back(cur);

}

negative:

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

cout << " (";

fout << " (";

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

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

{

cout << path[i] + 1 << " -> ";

fout << path[i] + 1 << " -> ";

}

else

{

cout << path[i] + 1;

fout << path[i] + 1;

}

cout << ")";

fout << ")";

}

}

fout.close();

# 6. Тестирование

Среда разработки Microsoft Visual Studio 2017 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 компонент сильной связности орграфов.

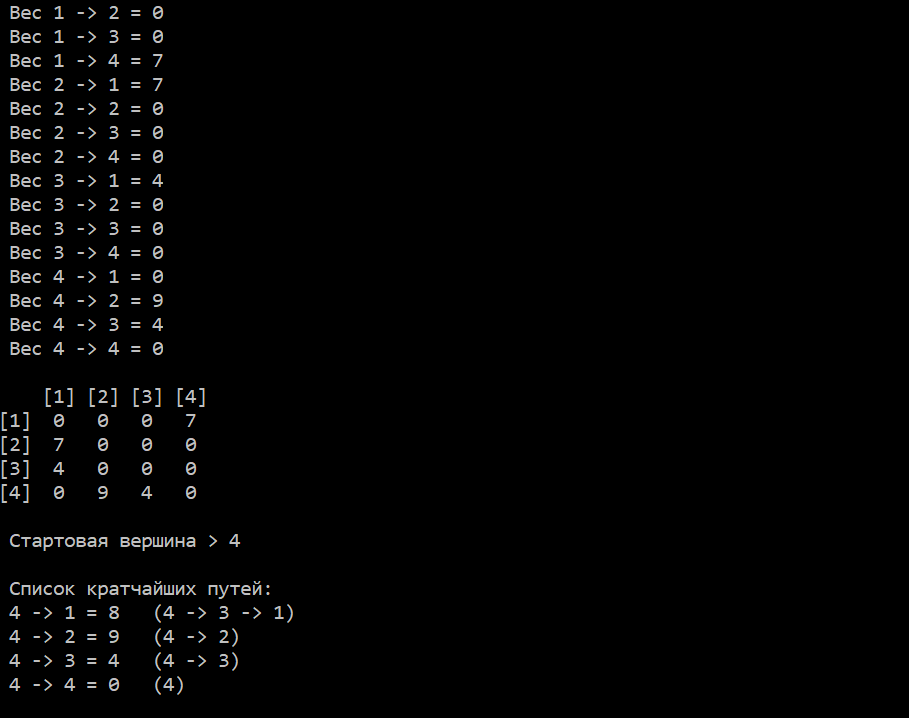
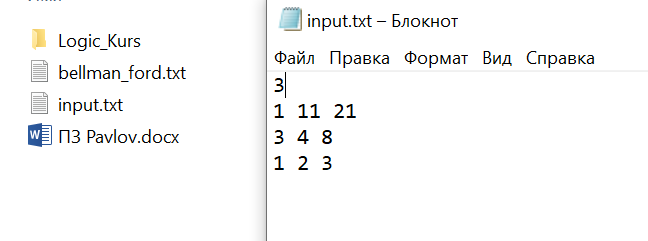


Рисунок 9 – тестирование при ручном вводе.



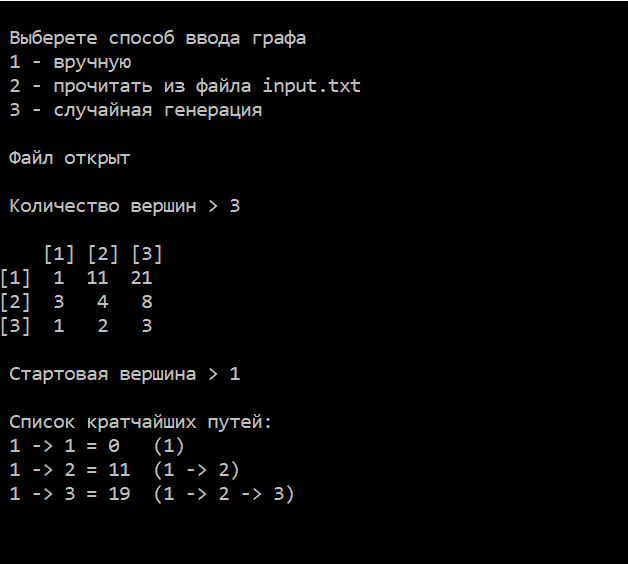


Рисунок 10 – тестирование при считывании из файла.

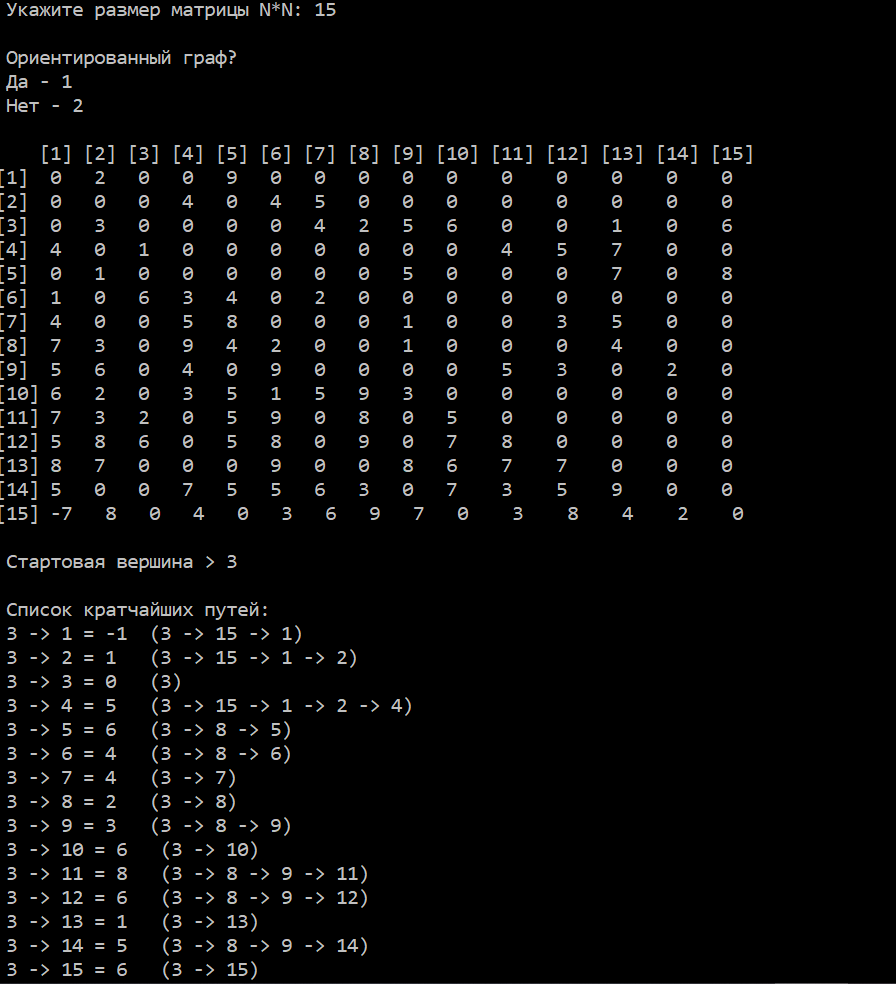


Рисунок 11 – тестирование при случайной генерации.

## 

Рисунок 12 – тестирование при наличии отрицательного цикла.

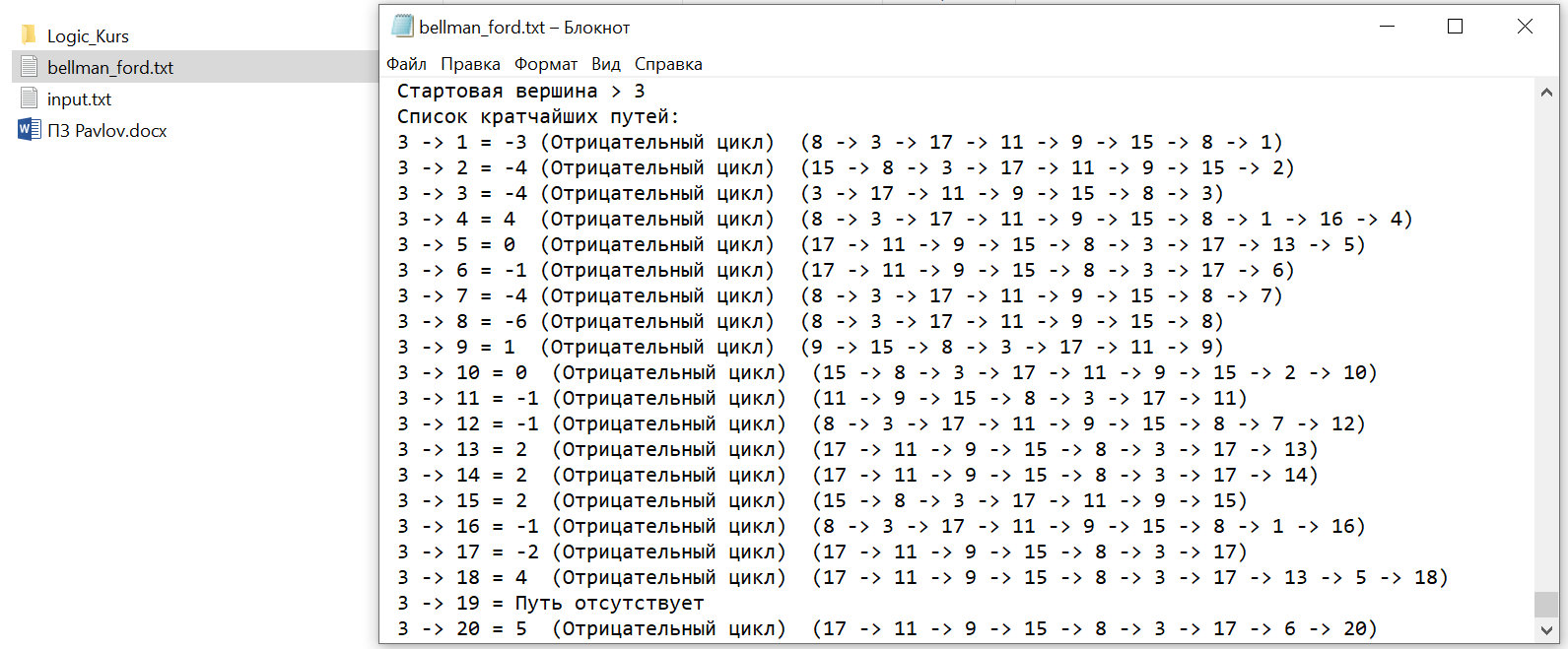


Рисунок 13 – тестирование сохранения результата в файле.

В результате тестирования было выявлено, что программа успешно

проверяет данные на соответствие необходимым требованиям.

# 7. Заключение

Таким образом, в процессе создания данного проекта разработана программа, которая, используя алгоритм Беллмана-Форда, находит длины кратчайших путей от вершины st до всех остальных вершин, запоминает путь до этих вершин, а также находит отрицательные циклы 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. В. Н. Касьянов, В. А. Евстигнеев - Графы в программировании: обработка, визуализация и применение.
2. info-master.su/programming – Поляков А.В. – Основы программирования.
3. Х. Дейтел. П. Дейтел. - Как программировать на C. 2007 г.
4. Роберт Седжвик - Алгоритмы на C++. Анализ структуры данных. Сортировка. Поиск. Алгоритмы на графах.
5. www.geeksforgeeks.org/bellman-ford-algorithm-dp-23 - Bellman–Ford Algorithm | DP-23

# Приложение Листинг программы

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <locale.h>

#include <iostream>

#include <fstream>

#include <iomanip>

#include <vector>

#include <time.h>

#include <stack> // стек

#include <string>

#include <limits>

#include <queue>

#define INFTY INT\_MAX

int\*\* M1, \*\*M2;

int N;

void generation();

void bellman\_ford();

using namespace std;

vector <int> nodes; // вектор расстояний

int st = 0; // стартовая вершина

#define inf 2000000000

using namespace std;

struct Edges {

int u, v, w;

};

const int Emax = 1000;

int i, j, n, e, start;

Edges edge[Emax]; // набор рёбер

void main()

{

setlocale(LC\_ALL, "Rus");

int w, c;

ofstream fout;

fout.open("../../bellman\_ford.txt", ios::app);

if (!fout.is\_open())

{

cout << "\n Ошибка открытия файла";

\_Exit(EXIT\_SUCCESS);

}

fout << endl << endl << endl;

input:

cout << "\n Выберете способ ввода графа";

cout << "\n 1 - вручную";

cout << "\n 2 - прочитать из файла input.txt";

cout << "\n 3 - случайная генерация";

input2:

c = \_getch();

if (c == 49)

{

cout << "\n\n Количество вершин > "; cin >> N;

nodes.resize(N);

M1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

M1[i] = (int\*)malloc(N \* sizeof(int));

}

e = 0;

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

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

{

cout << " Вес " << i + 1 << " -> " << j + 1 << " = "; cin >> w;

M1[i][j] = 0;

if (w != 0)

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = w;

M1[i][j] = w;

e++;

}

}

cout << "\n";

}

else if (c == 50) // считывание из файла

{

ifstream fin;

fin.open("../../input.txt");

if (!fin.is\_open())

{

cout << "\n\n Ошибка открытия файла";

cout << "\n Проверьте существование файла input.txt";

cout << "\n Для корректной работы программы файл должен быть заполнен в таком виде:";

cout << "\n Первая строка - кол-во вершин";

cout << "\n Начиная со второй - матрица смежности";

cout << "\n\n Пример заполнения:";

cout << "\n 3 ";

cout << "\n 0 8 0 ";

cout << "\n 3 0 0 ";

cout << "\n 4 0 0\n ";

fout.close();

fout.open("../../input.txt");

fout.close();

goto input;

}

cout << "\n\n Файл открыт";

fin >> N; // число строк

cout << "\n\n Количество вершин > " << N << endl;

nodes.resize(N);

M1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

M1[i] = (int\*)malloc(N \* sizeof(int));

}

//Считаем матрицу из файла

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

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

fin >> M1[i][j];

fin.close();

e = 0;

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

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

{

if (M1[i][j] != 0)

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = M1[i][j];

e++;

}

}

cout << "\n";

}

else if (c == 51)

{

cout << "\n";

generation();

e = 0;

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

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

{

if (M1[i][j] != 0)

{

edge[e].v = i;

edge[e].u = j;

edge[e].w = M1[i][j];

e++;

}

}

}

else

goto input2;

///// вывод матрицы смежности

printf(" ");

fout << " ";

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

{

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

fout << "[" << i + 1 << "] ";

}

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

{

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

fout << "\n[" << i + 1 << "]";

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

{

if ((i + 1) < 10)

{

if(M1[i][j] < 0)

cout << '\b';

if (j < 10)

{

printf("%3d ", M1[i][j]);

fout << setw(3) << M1[i][j] << " ";

}

else

{

printf(" %3d ", M1[i][j]);

fout << " " << setw(3) << M1[i][j] << " ";

}

}

else if ((i + 1) < 100)

{

if (j == 0)

{

printf(" %d", M1[i][j]);

fout << " " << M1[i][j];

}

else if (j < 10)

{

printf(" %d", M1[i][j]);

fout << " " << M1[i][j];

}

else

{

printf(" %d", M1[i][j]);

fout << " " << M1[i][j];

}

}

}

}

/////

cout << "\n\n Стартовая вершина > "; cin >> st;

fout << "\n\n Стартовая вершина > " << st;

st--;

cout << "\n Список кратчайших путей:";

fout << "\n Список кратчайших путей:";

fout.close();

bellman\_ford();

for (int i = N - 1; i > 0; i--)

{ // очищение памяти

free(M1[i]);

}

free(M1);

getch();

}

void generation() // генерация взвешанных графов

{

int naprav = 0;

char c;

setlocale(LC\_ALL, "russian");

printf("\n Укажите размер матрицы N\*N: ");

scanf("%d", &N);

printf("\n Ориентированный граф?");

printf("\n Да - 1 ");

printf("\n Нет - 2\n\n");

c = \_getch();

if(c == 49)

naprav = 1;

M1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

M1[i] = (int\*)malloc(N \* sizeof(int));

}

nodes.resize(N);

srand(time(NULL));

int ch = 0;

int ch2 = 0;

for (int i = 0; i < N; i++) // генерация матрицы M1 (взвешанный неориентированный)

{

M1[i][i] = 0;

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

{

ch = rand() % 10;

ch2 = rand() % 100;

if (ch < 7)

{

if (ch2 < 1)

ch = -ch;

M1[i][j] = ch;

}

else

M1[i][j] = 0;

M1[j][i] = M1[i][j];

}

}

M2 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

M2[i] = (int\*)malloc(N \* sizeof(int));

}

for (int i = 0; i < N; i++) // генерация матрицы M2 (взвешанный ориентированный граф)

{

M2[i][i] = 0;

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

{

ch = rand() % 15;

ch2 = rand() % 100;

if (ch < 12)

{

if (ch > 9)

ch = ch - 6;

if (ch2 < 1)

ch = -ch;

M2[i][j] = ch;

M2[j][i] = 0;

}

else

{

M2[i][j] = 0;

}

}

}

if(naprav == 1)

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

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

M1[i][j] = M2[i][j];

for (int i = N - 1; i > 0; i--)

{ // очищение памяти

free(M2[i]);

}

free(M2);

}

void bellman\_ford() //алгоритм Беллмана-Форда

{

int i, j;

ofstream fout;

fout.open("../../bellman\_ford.txt", ios::app);

if (!fout.is\_open())

{

cout << "\n Ошибка открытия файла";

\_Exit(EXIT\_SUCCESS);

}

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

nodes[i] = inf;

vector<int> parent(N, -1);

nodes[st] = 0;

int x;

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

{

bool change = false;

for (j = 0; j < e; j++) // v -> u, w - вес

if (nodes[edge[j].v] + edge[j].w < nodes[edge[j].u])

{

nodes[edge[j].u] = max(-inf, nodes[edge[j].v] + edge[j].w); // защита от переполнения

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

change = true;

}

if (!change) // если итерация не дала изменений, релаксации не продолжаются

break;

}

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

{

if (nodes[i] == inf)

{

cout << endl << " " << st + 1 << " -> " << i + 1 << " = " << "Путь отсутствует ";

fout << endl << " " << st + 1 << " -> " << i + 1 << " = " << "Путь отсутствует ";

}

else

{

cout << endl << " " << st + 1 << " -> " << i + 1 << " = " << nodes[i];

fout << endl << " " << st + 1 << " -> " << i + 1 << " = " << nodes[i];

// Восстановление пути

if (nodes[i] >= 0 && nodes[i] < 10)

{

cout << " ";

fout << " ";

}

vector<int> path; // вектор для восстановления пути

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

{

for (int i2 = 0; i2 < path.size(); i2++)

if (cur == path[i2] && path.size() > 1) // если после первой итерации попадаем в уже посещённую

{ // вершину - отрицательный цикл

path.push\_back(cur);

cout << " (Отрицательный цикл)";

fout << " (Отрицательный цикл)";

goto negative;

}

path.push\_back(cur);

}

negative:

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

cout << " (";

fout << " (";

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

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

{

cout << path[i] + 1 << " -> ";

fout << path[i] + 1 << " -> ";

}

else

{

cout << path[i] + 1;

fout << path[i] + 1;

}

cout << ")";

fout << ")";

}

}

fout.close();

}