

**Содержание**

[Реферат 4](#_Toc123068610)

[Введение 5](#_Toc123068611)

[1. Постановка задачи 6](#_Toc123068612)

[2. Теоретическая часть задания 7](#_Toc123068613)

[3. Описание алгоритма программы 8](#_Toc123068614)

[4. Описание программы 9](#_Toc123068615)

[5. Тестирование 10](#_Toc123068616)

[6. Ручной расчёт задачи 12](#_Toc123068617)

[Заключение 23](#_Toc123068618)

[Список литературы 24](#_Toc123068619)

[Листинг программы 25](#_Toc123068620)

# Реферат

Отчет 35 стр, 28 рисунков

ГРАФ, ТЕОРИЯ ГРАФОВ, ОРГРАФ, АЛГОРИТМ БЕЛЛМАНА-ФОРДА, ПОИСК КРАТЧАЙШИХ РАССТОЯНИЙ.

Цель исследования – разработка программы, которая будет находить кратчайшие пути от одной вершины графа до всех остальных, используя алгоритм Беллмана-Форда.

В работе рассмотрен алгоритм Беллмана-Форда, предназначенный для поиска кратчайшего пути в графе. Установлено, что с помощью данного алгоритма можно делать восстановление пути, а также проверять наличие циклов отрицательного веса в графе.

# Введение

Алгоритм Беллмана–Форда — это алгоритм, который вычисляет кратчайший путь от одной исходной вершины ко всем другим вершинам в взвешенном орграфе. Он медленнее, чем алгоритм Дейкстры для той же задачи, но более универсален, так как способен обрабатывать графы, в которых некоторые веса ребер являются отрицательными числами.

В качестве среды разработки мною была выбрана  
среда Microsoft Visual Studio 2019, язык программирования – С++.

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

# Постановка задачи

Исходный граф в программе должен задаваться матрицей смежности, причём при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, пользователь вводит количество вершин для генерации матрицы смежности, выбирает способ заполнения матрицы смежности, случайным способом или заполнить самому. После обработки этих данных на экран должна выводиться матрица смежности графа, все компоненты связности графа, результат окраски графа. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

Задания выполняются в соответствии с вариантом №14.

# Теоретическая часть задания

Учитывая исходную вершину s из множества вершин V во взвешенном ориентированном Graph, где его реберные веса w(u, v) может быть отрицательным, найти веса кратчайших путей d(s, v) из источника s для всех вершин v присутствует на Graphе. Если граф содержит цикл отрицательного веса, сообщите об этом.

Рассмотрим в качестве примера данный Graph.

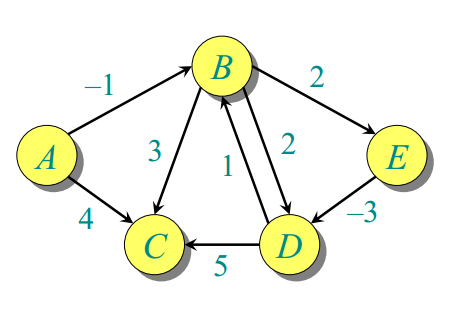


Рисунок 1 - пример графа

Идея состоит в том, чтобы использовать Алгоритм Беллмана – Форда для вычисления кратчайших путей от одной исходной вершины ко всем остальным вершинам данного взвешенного орграфа. Алгоритм Беллмана-Форда медленнее, чем Алгоритм Дейкстры, но он может обрабатывать ребра с отрицательным весом в Graph, в отличие от Дейкстры.

Если граф содержит “отрицательный цикл” (т. е. цикл, сумма ребер которого дает отрицательное значение), достижимый из источника, то кратчайшего пути не существует. Любой путь, имеющий точку на отрицательном цикле, можно сделать дешевле, если еще раз пройти по отрицательному циклу. Алгоритм Беллмана-Форда может легко обнаружить любые отрицательные циклы в Graph.

Алгоритм инициализирует расстояние до источника до 0 и все остальные узлы INFINITY. Затем для всех ребер, если расстояние до пункта назначения можно сократить, взяв ребро, расстояние обновляется до нового меньшего значения. На каждой итерации i что ребра просматриваются, алгоритм находит все кратчайшие пути максимальной длины i края. Так как самый длинный путь без цикла может быть V-1 края, края должны быть отсканированы V-1 раз, чтобы убедиться, что кратчайший путь найден для всех узлов. Выполняется окончательное сканирование всех ребер, и если какое-либо расстояние обновляется, то путь длиной |V| найдены ребра, что возможно только в том случае, если в Graph существует хотя бы один отрицательный цикл.

# 3. Описание алгоритма программы

Для заданного графа и исходной вершины src в графе найти кратчайшие пути от src до всех вершин данного графа. Граф может содержать ребра с отрицательным весом. Этот шаг инициализирует расстояния от источника до всех вершин как бесконечные, а расстояние до самого источника как 0. Создайте массив dist[] размера |V| со всеми значениями как бесконечными, кроме dist[src], где src — исходная вершина.

На этом шаге вычисляются кратчайшие расстояния. Выполните следующие |V|-1 раз, где |V| - количество вершин в данном графе. Сделайте следующее для каждого края uv.

Если dist[v] > dist[u] + вес ребра uv, то обновить dist[v] до dist[v] = dist[u] + вес ребра uv.

Этот шаг сообщает, есть ли на графике отрицательный весовой цикл. Снова пройдите через каждое ребро и сделайте следующее для каждого ребра uv.

Если dist[v] > dist[u] + вес ребра uv, тогда «График содержит отрицательный весовой цикл».

Идея шага 3 состоит в том, что шаг 2 гарантирует кратчайшие расстояния если граф не содержит отрицательный весовой цикл. Если мы еще раз пройдемся по всем ребрам и получим более короткий путь для любой вершины, то получится цикл с отрицательным

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[].

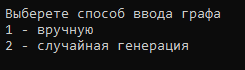
33. Закрываем файл bellman\_ford.txt

# Описание программы

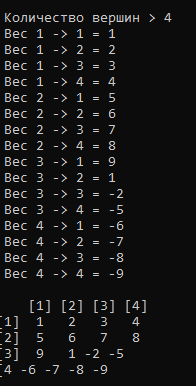
Для написания данной программы использован язык программирования С++. Язык программирования С++ - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

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

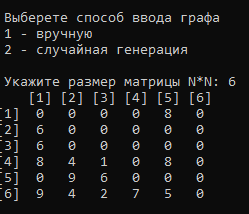
Работа программы начинается с главного меню, где пользователю предлагается выбор.

  
Рисунок 2 – меню программы

Если пользователь выбирает ручной ввод, то ему предоставляется возможность ввести все данные самостоятельно

  
Рисунок 3 – ручной ввод

Если пользователь хочет случайную генерацию, то должен выбрать 2й пункт и размер матрицы.

  
Рисунок 4 – случайная генерация

# 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 вывода результата.

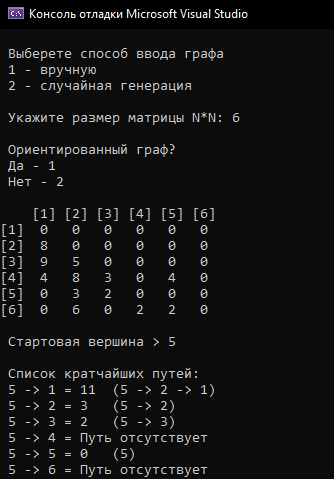


Рисунок 5 – тестирование при случайном заполнении графа   
при 6 вершинах

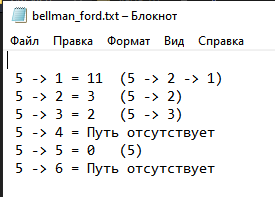


Рисунок 6 – вывод тестирования сверху в файл

Если путь был не найден, то программа выдаст сообщение пользователю.

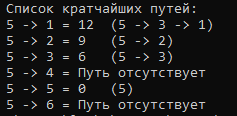


Рисунок 7 – оповещение пользователя об отсутствии пути.

Таблица 1 – результаты тестирования

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод сообщения о вводе кол-ва вершин | Верно |
| Описание теста | Ожидаемый результат | Полученный результат |
| Выбор генерации матрицы | Вывод меню программы и выбор пользователем нужного пункта | Верно |
| Заполнение матрицы случайными числами | Ввод вероятности и вывода матрица смежности | Верно |
| Самостоятельное и случайное заполнение матрицы смежности, корректный вывод результата | Вывод матрицы смежности с n вершинами. Вывод результата в виде ряда вершин с минимальной суммой весов | Верно |
| Запись результата в файл | Запись результата работы программы в файл | Верно |
| Проверка на отрицательный цикл | Вывод пользователю сообщения, что при работе программы цикл стал отрицательным | Верно |
| Проверка на отсутствие путей | Вывод пользователю сообщения, что при работе программы путь был не найден | Верно |
| Нахождение кратчайших путей | Оповещение пользователя о том, что кратчайшие пути от начальной вершины были найдены | Верно |

В результате тестирования было выявлено, что программа успешно выдает результат.

# Ручной расчёт задачи

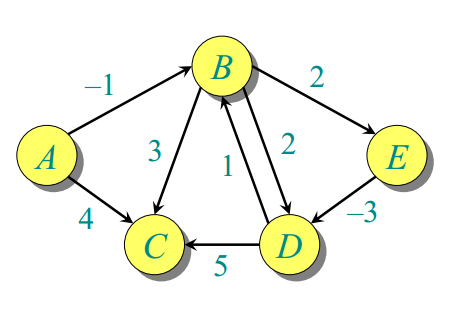
Проведем проверку программы посредством ручных вычислений на примере графа самостоятельного ввода с 5ю вершинами и случайного заполнения с вершинами.

Рисунок 8 – Начальный граф

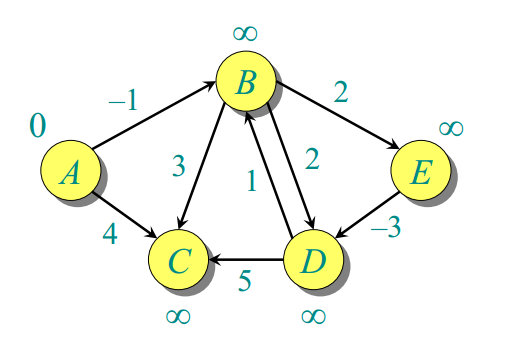
Сначала инициализируется расстояние от источника до всех вершин как бесконечные, а расстояние до самого источника как 0.

Рисунок 9 – Инициализация графа

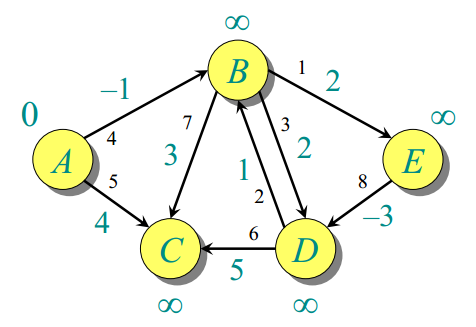
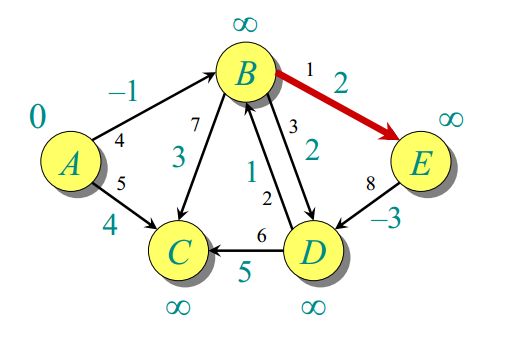
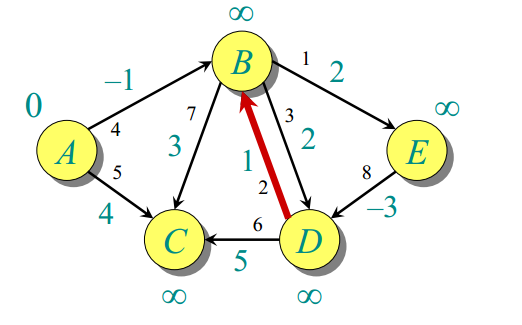
Дальше идет проверка на отрицательные веса в графе.

Рисунок 10 – Проверка на отрицательность

После проверки на отрицательные веса происходит первая итерация, которая ищет все доступные пути.

Рисунок 11 – Проверка ребра BE



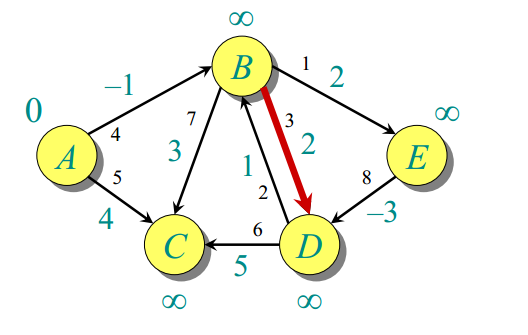
Рисунок 12 – Проверка DB

Рисунок 13 – Проверка ребра BD

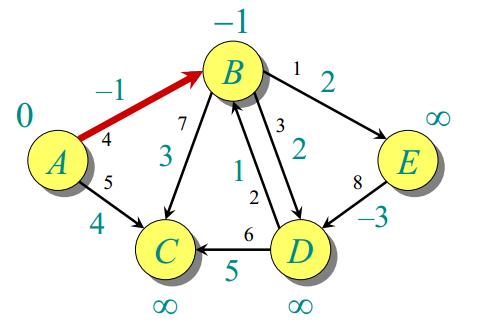
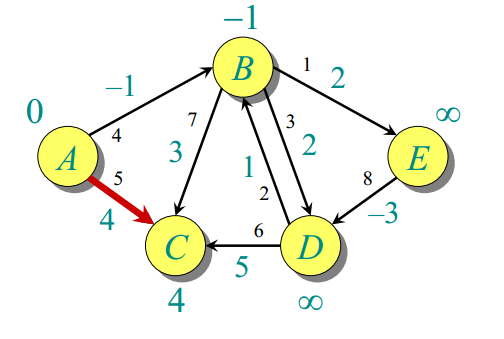
Рисунок 14 – Проверка ребра AB

Рисунок 15 – Проверка ребра AC

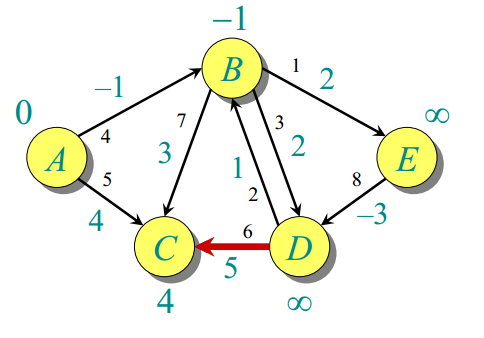
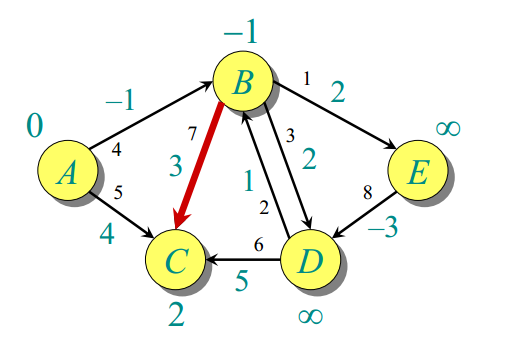
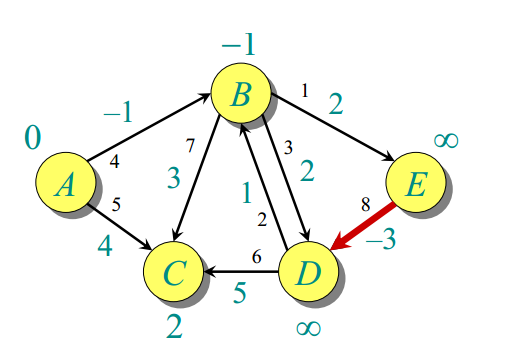
Рисунок 16 – Проверка ребра DC

Рисунок 17 - Проверка ребра BC

Рисунок 18 - проверка ребра ED

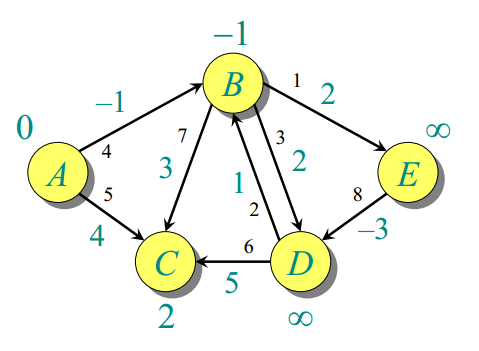
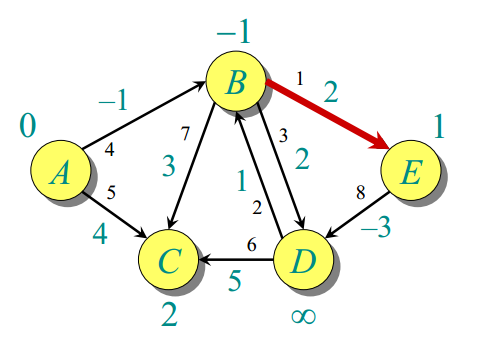


Рисунок 19 – Конец первой итерации

После 1й итерации идет вторая, которая уже учитывает отрицательные веса ребер  
 

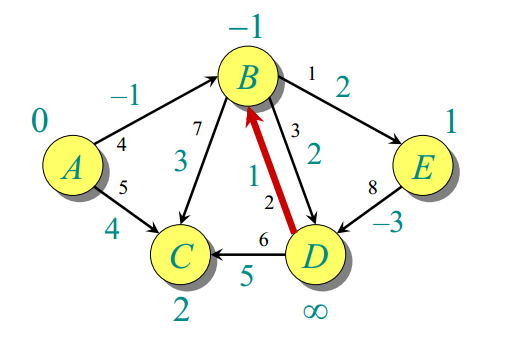
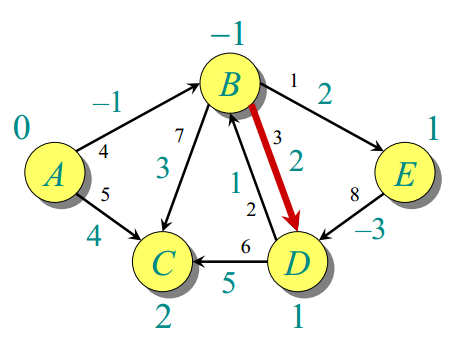
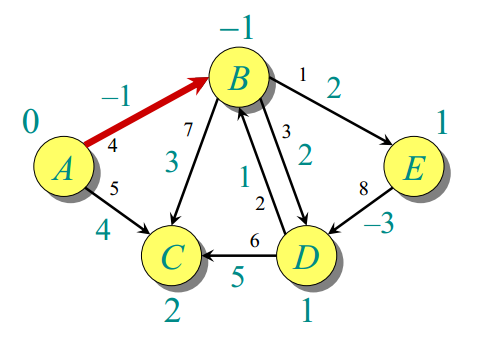
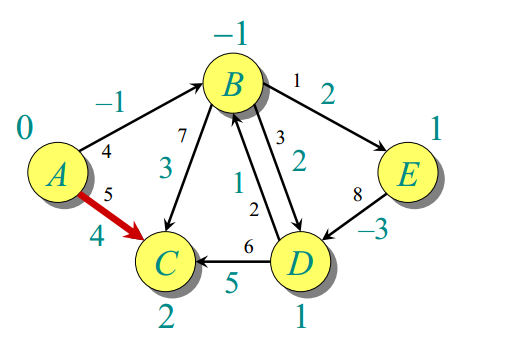
Рисунок 20 – Проверка ребра BE

Рисунок 21 – Проверка ребра DB

Рисунок 22 – Проверка ребра BD

Рисунок 23 – Проверка ребра AB



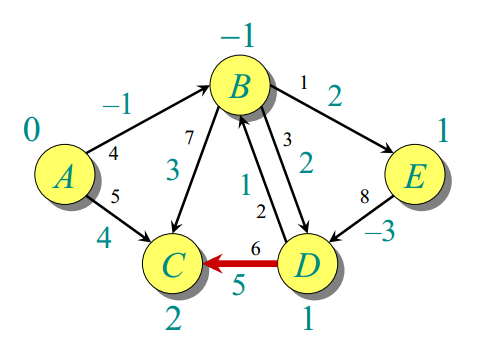
Рисунок 24 – Проверка ребра AC

Рисунок 25 – Проверка ребра DC

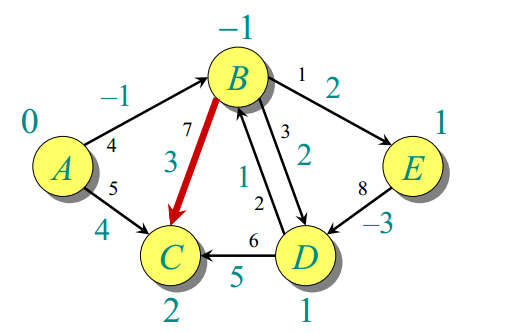


Рисунок 26 – Проверка ребра BC

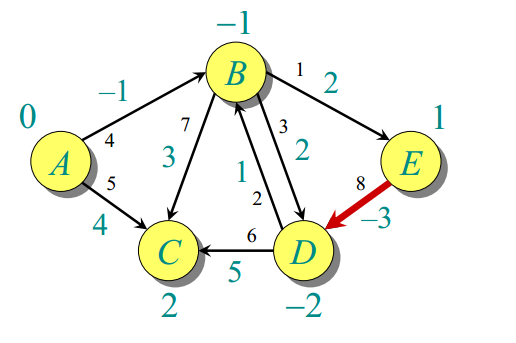


Рисунок 27 – Проверка ребра ED

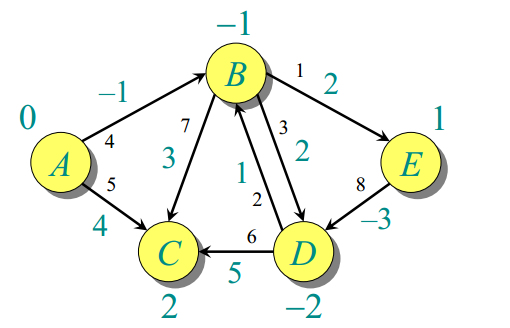


Рисунок 28 – Конец 2й итерации

После прохождения 2й итерации алгоритм проверяет ещё раз граф на отрицательные веса, если он их не находит, то он останавливается на 2й итерации.

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм раскраски графа в Microsoft Visual Studio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории неопределенных графов. Приобретены навыки по осуществлению алгоритма раскраски графа. Углублены знания языка программирования C++.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

# Список литературы

1. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978
2. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006
3. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.
4. Лекция по дискретной математике, по теме Раскраска графа - <https://en.ppt-online.org/45056>

# 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");

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 - 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)

{

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);

}

void generation() // генерация взвешанных графов

{

int naprav = 0;

char c;

setlocale(LC\_ALL, "russian");

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

scanf("%d", &N);

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");

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();

}