

Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовому проектированию

по курсу «Логика и основы алгоритмизации

в инженерных задачах»

## на тему «Алгоритм Форда-Беллмана»

Выполнил студент группы 21ВВ1.1:

Киреев Д.А.

Принял:

Юрова О.В.

Пенза 2022

**Содержание**

Введение............................................................................................................. 3

Постановка задачи............................................................................................. 4

Теоретическая часть задания ........................................................................... 5

Описание алгоритма программы...................................................................... 6

Описание программы........................................................................................ 8

Тестирование ................................................................................................... 11

Ручной расчёт ……………………………………………………………….. 14

Заключение ...................................................................................................... 17

Список литературы ......................................................................................... 18

Листинг программы ........................................................................................19

**Введение**

Теория о нахождении кратчайшего пути является одной из важнейших классических задач теории графов. Сегодня известно множество алгоритмов для её решения. Значимость данной задачи определяется её различными практическими применениями в разных областях: в GPS –навигаторах для осуществления поиска кратчайшего пути между двумя перекрестками, в картографических сервисах для нахождения путей между физическими объектами на таких картографических сервисах, как карты Google или OpenStreetMap.

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudio2022, 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, осуществляющийпоиск [кратчайших путей](https://en.wikipedia.org/wiki/Shortest_path_problem) во [взвешенном графе](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%BE%D1%81%D1%81%D0%B0%D1%80%D0%B8%D0%B9_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2) с положительным или отрицательным весом ребер.

**Постановка задачи**

Требуется разработать программу, которая будет находить кратчайшие пути между вершинами в графе, используя алгоритм Форда-Беллмана.

Программа должна работать так, чтобы пользователь генерировал граф, как ориентированный, так и неориентированный. После обработки этихданных на экран должна выводиться матрица на основе сгенерированного графа,после вывода матрицы в ней должны находиться все кратчайшие пути до указанной пользователем вершины. Необходимо предусмотреть различныеисходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

**Теоретическая часть задания**

Алгоритм носит имя двух американских учёных: Ричарда Беллмана (Richard Bellman) и Лестера Форда (Lester Ford). Форд фактически изобрёл этот алгоритм в 1956 г. при изучении другой математической задачи, подзадача которой свелась к поиску кратчайшего пути в графе, и Форд дал набросок решающего эту задачу алгоритма. Беллман в 1958 г. опубликовал статью, посвящённую конкретно задаче нахождения кратчайшего пути, и в этой статье он чётко сформулировал алгоритм в том виде, в котором он известен нам сейчас.

**Алгоритм Форда-Беллмана** позволяет найти кратчайшие пути из одной вершины графа до всех остальных, даже для графов, в которых веса ребер могут быть отрицательными.

Алгоритм Форда-Беллмана использует динамическое программирование.

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

**Описание алгоритма программы**

Введем функцию динамического программирования:

F[k][i] — длина кратчайшего пути из начальной вершины до вершины i, содержащего не более k ребер.

Начальные значения зададим для случая k=0. В этом случае F[0][start] = 0, а для всех остальных вершин i F[0][i] = INF, то есть путь, состоящий из нуля ребер существует только от вершины start до вершины start, а до остальных вершин пути из нуля ребер не существует, что будем отмечать значением INF.

Далее будем вычислять значения функции F увеличивая число ребер в пути k, то есть вычислим кратчайшие пути, содержащие не более 1 ребра, кратчайшие пути, содержащие не более 2 ребер и т. д. Если в графе нет циклов отрицательного веса, то кратчайший путь между любыми двумя вершинами содержит не более  ребра ( - число вершин в графе), поэтому нужно вычислить значения F[n-1][i], которые и будут длинами кратчайших путей от вершины start до вершины i).

Рассмотрим, как вычисляется значение F[k][i]. Пусть есть кратчайший маршрут из вершины start до вершины i, содержащий не более k ребер. Пусть последнее ребро этого маршрута есть ребро j-i. Тогда путь до вершины j содержит не более k-1 ребра и является кратчайшим путем из всех таких путей, значит, его длина равна F[k-1][j], а длина пути до вершины i равна F[k-1][j] + W[j][i], где W[j][i] есть вес ребра j-i. Дальше необходимо перебрать все вершины j, которые могут выступать в качестве предыдущих, и выбрать минимальное значение F[k-1][j] + W[j][i].

Получаем следующий алгоритм:

int\*\* MinPath; // массив кратчайших путей

int i, j, k;

int count = 0;

MinPath = (int\*\*)malloc(CountApex \* sizeof(int\*));

for (inti = 0; i<CountApex; i++)

{

MinPath[i] = (int\*)malloc(CountApex \* sizeof(int));

}

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

{

MinPath[0][i] = INF; //изначально кратчайшие пути неизвестны

}

MinPath[0][Start] = 0;

for (k = 1; k <CountApex; k++)

{

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

{

MinPath[k][i] = MinPath[k - 1][i];

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

{

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

if ((MinPath[k - 1][j] != INF) && (MinPath[k - 1][j] + SourceMatrix[j][i] <MinPath[k][i]))

{

MinPath[k][i] = MinPath[k - 1][j] + SourceMatrix[j][i];

}

}

}

}

То есть, по сути, алгоритм Форда-Беллмана можно сформулировать так:

1. Проинициализировать массив F значениями F[start] = 0, F[i] = INF для остальных i.
2. Пройтись по всем ребрам j-i графа, пытаясь срелаксировать ребро j-i.
3. Пункт 2 повторить n-1 раз.

**Описание программы**

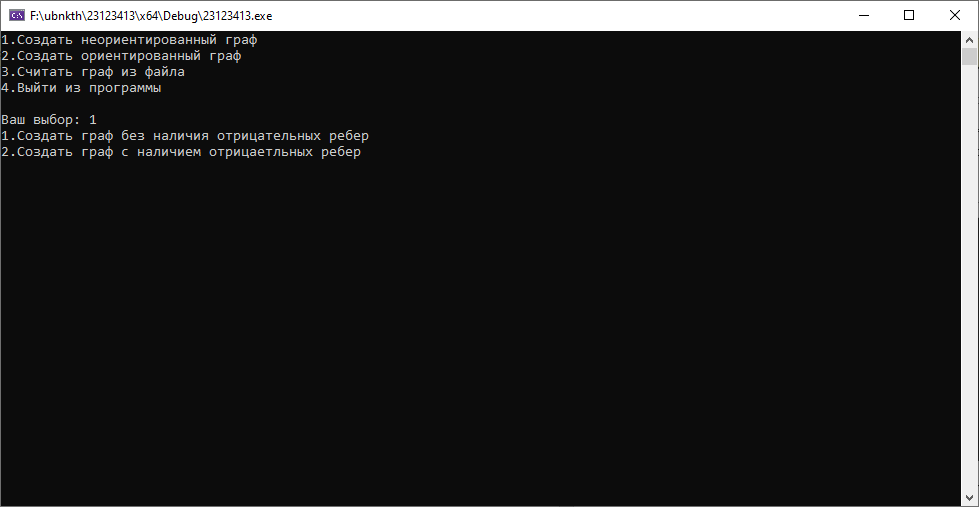
Для написания данной программы использован язык программирования

С#.Язык программирования С# - объектно-ориентированный язык программирования общего назначения.

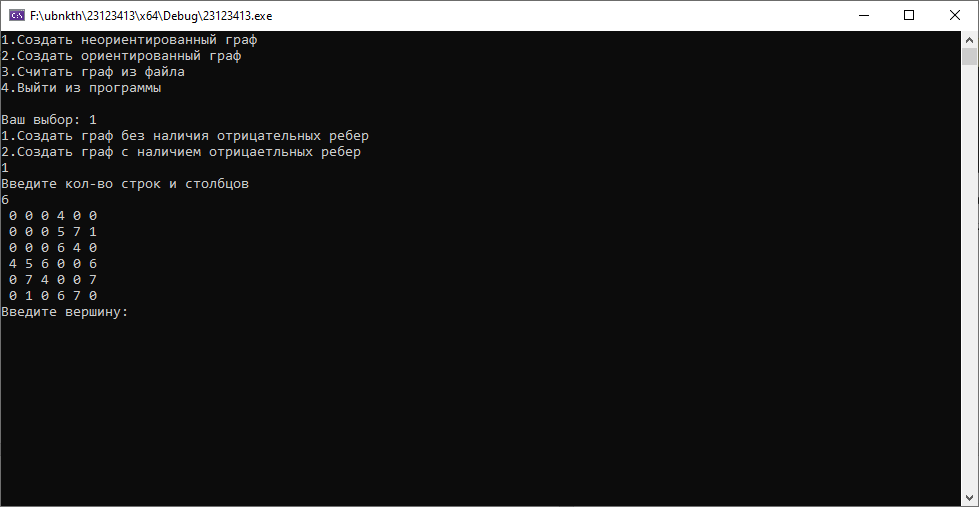
Проект был создан в виде консольного приложения Win32 (Visual C#).

Данная программа является многофункциональной.

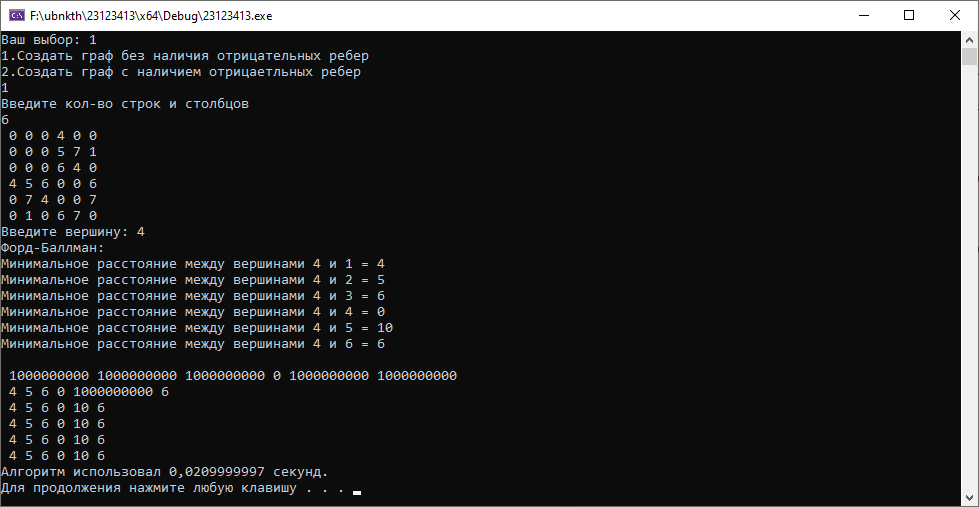
Работа программы начинается сгенерации меню и выбора графа, который хочет создать пользователь: неориентированный, ориентированный или считать свой граф из файла. Также, на выбор пользователя, программа может сделать граф с наличием отрицательных ребер.



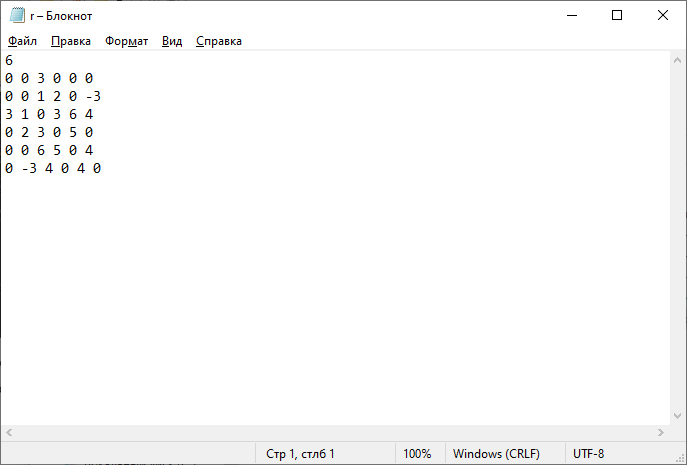
Далее, выбрав нужный вариант меню,программа выводит случайно созданный ею граф, с полученным от пользователя количеством вершин, либо считанный с файла граф.

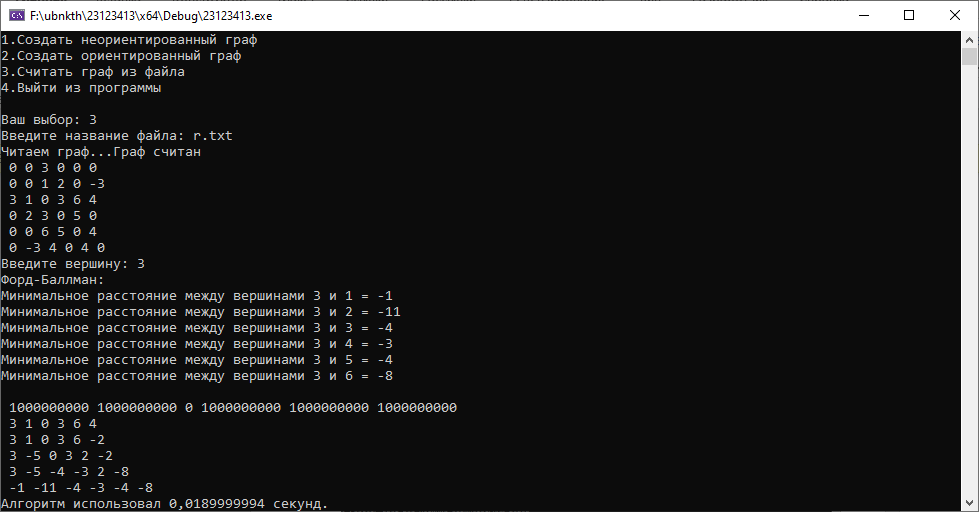


Затем пользователь должен выбрать вершину, до которой он хочет найти кратчайшие пути. Далее программа выведет результат, который будет содержать в себе конечные расстояния до исходной вершины и матрицу расстояний, которые были просчитаны на каждой итерации.



Также программа способна считать граф из файла пользователя и выполнить алгоритм.



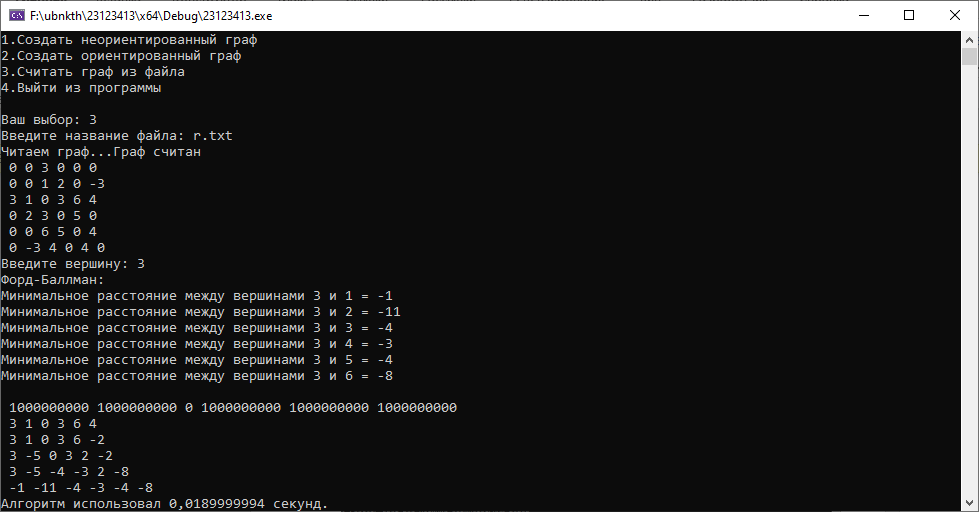


В конце выполнения подсчета алгоритма программа всегда выводит время, за которое алгоритм был просчитан.

**Тестирование**

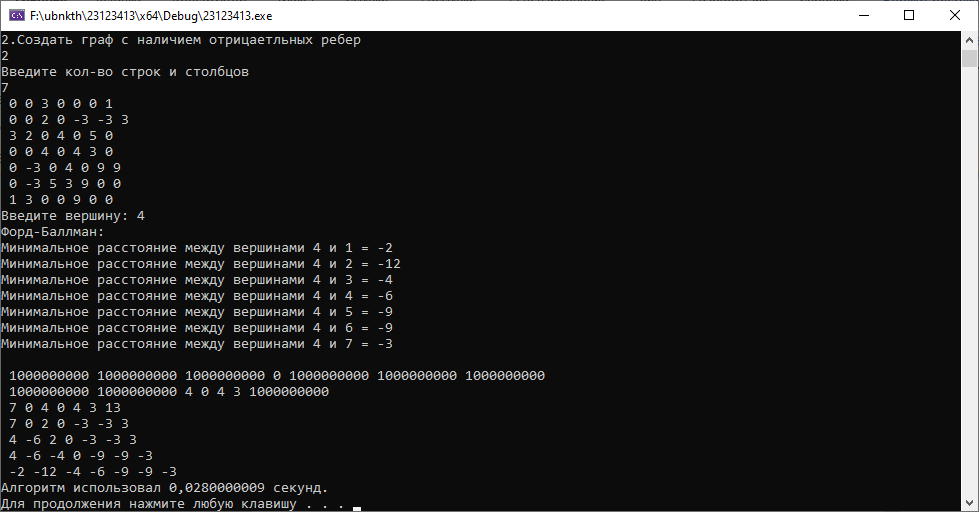
Среда разработки MicrosoftVisualStudio 2022 предоставляет всесредства, необходимые при разработке и отладке многомодульной программы.Тестирование проводилось в рабочем порядке, в процессе разработки,после завершения написания программы. В ходе тестирования было выявлено иисправлено множество проблем, связанных с вводом данных, изменениемдизайна выводимых данных, алгоритмом программы, взаимодействиемфункций.Ниже продемонстрирован результат тестирования программы привводепользователем различных размеров графа.

Ниже продемонстрирован результат тестирования программы при считывании графа из файла



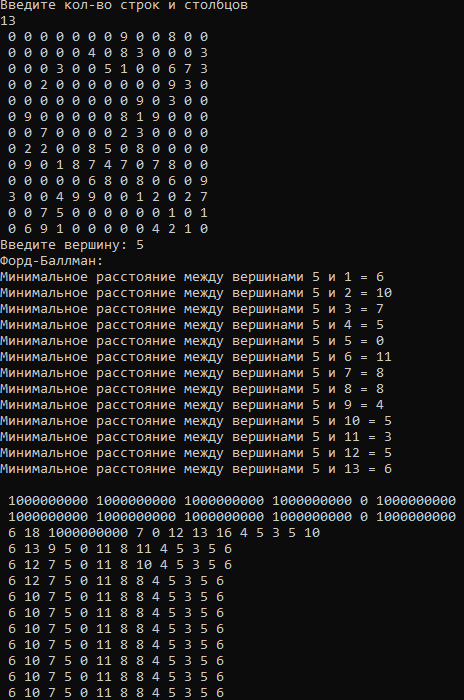
Неориентированный граф с 7 вершинами с наличием отрицательных ребер:

Исходные данные:



Ориентированный граф с положительными ребрами 13х13

Исходные данные:



**Ручной расчёт**

В ручную рассчитаем кратчайшие пути в графе и сравним с тем , что получится в программе на графес 5ю вершинами:



Расстояния в 0 реберот вершины 3до остальных:

Из 3 в 1 = INF

Из 3 в 2 = INF

Из 3 в 3 = 0

Из 3 в 4 = INF

Из 3 в 5 = INF

Расстояния в 1 реброот вершины 3до остальных:

Из 3 в 1 = INF

Из 3 в 2 = 4

Из 3 в 3 = 0

Из 3 в 4 = INF

Из 3 в 5 = 5

Расстояния в 2 ребраот вершины 3до остальных:

Из 3 в 1 = 12

Из 3 в 2 = 4

Из 3 в 3 = 0

Из 3 в 4 = 6

Из 3 в 5 = 5

Расстояния в 3 ребраот вершины 3до остальных:

Из 3 в 1 = 12

Из 3 в 2 = 4

Из 3 в 3 = 0

Из 3 в 4 = 6

Из 3 в 5 = 5

Расстояния в 4 ребраот вершины 3до остальных:

Из 3 в 1 = 12

Из 3 в 2 = 4

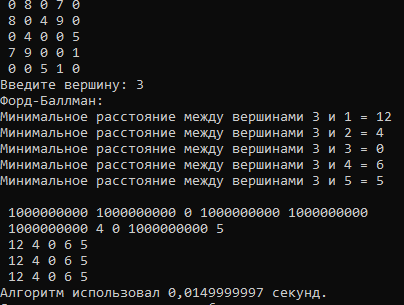
Из 3 в 3 = 0

Из 3 в 4 = 6

Из 3 в 5 = 5

Из 3 в 5 = 5

Результат работы программы:



Ручной расчёт совпадает с работой в программе , следовательно программа работает правильно

**Заключение**

Таким образом, в процессе создания данного проекта разработанапрограмма, реализующая алгоритм Форда-Беллманадля поиска кратчайшего пути вMicrosoftVisualStudio 2022.При выполнении данной курсовой работы были получены навыкиразработки программ и освоены приемы создания графа. Приобретены навыки по осуществлению алгоритма Форда-Беллмана. Углублены знания языка программированияC#.Программа имеет небольшой, но достаточный для использованияфункционал возможностей.

**Список литературы**

**1)**https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC\_%D0%91%D0%B5%D0%BB%D0%BB%D0%BC%D0%B0%D0%BD%D0%B0\_%E2%80%94\_%D0%A4%D0%BE%D1%80%D0%B4%D0%B0

**2)**https://algowiki-project.org/ru/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC\_%D0%91%D0%B5%D0%BB%D0%BB%D0%BC%D0%B0%D0%BD%D0%B0-%D0%A4%D0%BE%D1%80%D0%B4%D0%B0

**Листингпрограммы.**

#include<iostream>

#include<locale>

#include<time.h>

#defineINF 1000000000

//Создание неориентированного графа через матрицу смежности

voidneormatrix(int\*\* arr, intcount, intvibor)

{

int w, t, k = 0;

if (vibor == 1)

{

for (inti = 0; i<count; i++)

{

for (int j = i; j <count; j++)

{

w = rand() % 2;

if (w == 1)

{

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

arr[j][i] = arr[i][j];

}

else {

arr[i][j] = 0;

arr[j][i] = arr[i][j];

}

if (i == j)

{

arr[i][j] = 0;

}

}

}

}

if (vibor == 2)

{

for (inti = 0; i<count; i++)

{

for (int j = i; j <count; j++)

{

w = rand() % 2;

if (w == 1)

{

t = rand() % 2;

if (t == 1)

{

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

arr[j][i] = arr[i][j];

}

else

arr[i][j] = rand() % 10 - 5;

arr[j][i] = arr[i][j];

}

else {

arr[i][j] = 0;

arr[j][i] = arr[i][j];

}

if (i == j)

{

arr[i][j] = 0;

}

}

}

}

for (inti = 0; i<count; i++)

{

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

{

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

}

printf("\n");

}

}

//Создание ориентированного графа через матрицу смежности

voidormatrix(int\*\* arr, intcount,intvibor)

{

int w, t, k = 0;

if (vibor == 1)

{

for (inti = 0; i<count; i++)

{

for (int j = i; j <count; j++)

{

w = rand() % 2;

if (w == 1)

{

t = rand() % 2;

if (t == 1)

{

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

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

}

else

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

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

}

else {

arr[i][j] = 0;

arr[j][i] = arr[i][j];

}

if (i == j)

{

arr[i][j] = 0;

}

}

}

}

if (vibor == 2)

{

for (inti = 0; i<count; i++)

{

for (int j = i; j <count; j++)

{

w = rand() % 2;

if (w == 1)

{

t = rand() % 2;

if (t == 1)

{

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

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

}

else

arr[i][j] = rand() % 10 - 5;

arr[j][i] = rand() % 10 - 5;

}

else {

arr[i][j] = 0;

arr[j][i] = arr[i][j];

}

if (i == j)

{

arr[i][j] = 0;

}

}

}

}

for (inti = 0; i<count; i++)

{

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

{

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

}

printf("\n");

}

}

//Выводалгоритма

voidoutput(intcount, int\*\* result, intapex)

{

inti, j;

printf("Форд-Баллман:\n");

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

{

//еслипутьесть

if (result[count - 1][j] < 10000)

{

printf("Минимальное расстояние между вершинами %d и %d = %d\n", apex + 1, j + 1, result[count - 1][j]);

}

else

printf("Минимальное расстояние между вершинами %d и %d = INF\n", apex + 1, j + 1);

}

printf("\n");

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

{

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

{

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

}

printf("\n");

}

}

//АлгоритмФорда-Беллмана

voidFord\_Ballman(intCountApex, int\*\* SourceMatrix, intStart)

{

int\*\* MinPath; // массивкратчайшихпутей

inti, j, k;

int count = 0;

MinPath = (int\*\*)malloc(CountApex \* sizeof(int\*));

for (inti = 0; i<CountApex; i++)

{

MinPath[i] = (int\*)malloc(CountApex \* sizeof(int));

}

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

{

MinPath[0][i] = INF; //изначально кратчайшие пути неизвестны

}

MinPath[0][Start] = 0;

for (k = 1; k <CountApex; k++)

{

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

{

MinPath[k][i] = MinPath[k - 1][i];

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

{

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

if ((MinPath[k - 1][j] != INF) && (MinPath[k - 1][j] + SourceMatrix[j][i] <MinPath[k][i]))

{

MinPath[k][i] = MinPath[k - 1][j] + SourceMatrix[j][i];

}

}

}

}

output(CountApex, MinPath, Start);

}

intmain()

{

setlocale(LC\_ALL, "Rus");

int\*\* a, n, k = 0, begin = 0,u;

float start, end;

FILE \*in;

charfilename[20];

while (begin != 4)

{

printf("1.Создать неориентированный граф\n2.Создать ориентированный граф\n3.Считать граф из файла\n4.Выйти из программы\n");

printf("\nВашвыбор: ");

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

if (begin == 1)

{

printf("1.Создать граф без наличия отрицательных ребер\n2.Создать граф с наличием отрицаетльных ребер\n");

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

printf("Введите кол-во строк и столбцов\n");

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

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

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

{

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

}

neormatrix(a, n,u);

printf("Введитевершину: ");

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

k = k - 1;

start = clock();

Ford\_Ballman(n, a, k);

end = clock();

printf("Алгоритмиспользовал %.10f секунд.\n", (end - start) / (CLOCKS\_PER\_SEC));

free(a);

system("pause");

}

if (begin == 2)

{

printf("1.Создать граф без наличия отрицательных ребер\n2.Создать граф с наличием отрицаетльных ребер\n");

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

printf("Введите кол-во строк и столбцов\n");

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

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

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

{

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

}

ormatrix(a, n,u);

printf("Введитевершину: ");

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

k = k - 1;

start = clock();

Ford\_Ballman(n, a, k);

end = clock();

printf("Алгоритмиспользовал %.10f секунд.\n", (end - start) / (CLOCKS\_PER\_SEC));

free(a);

system("pause");

}

if (begin == 3)

{

printf("Введитеназваниефайла: ");

scanf\_s("%s", filename, 20);

fopen\_s(&in, filename, "r");

if(!in)

printf("Ошибка! Файл не прочитан\n");

else

printf("Читаемграф...");

fscanf\_s(in, "%d", &n);

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

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

{

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

}

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

{

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

{

fscanf\_s(in, "%d", &a[i][j]);

}

}

fclose(in);

printf("Графсчитан\n");

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

{

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

{

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

}

printf("\n");

}

printf("Введитевершину: ");

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

k = k - 1;

start = clock();

Ford\_Ballman(n, a, k);

end = clock();

printf("Алгоритмиспользовал %.10f секунд.\n", (end - start) / (CLOCKS\_PER\_SEC));

free(a);

system("pause");

}

}

getchar();

getchar();

}