

Содержание

[Реферат 5](#_Toc90316383)

[Введение 6](#_Toc90316384)

[1 Постановка задачи 7](#_Toc90316385)

[2 Теоретическая часть 8](#_Toc90316386)

[3 Описание алгоритма программы 10](#_Toc90316387)

[4 Описание программы 12](#_Toc90316388)

[5 Тестирование 17](#_Toc90316389)

[6 Ручной расчёт задачи 20](#_Toc90316390)

[Заключение 24](#_Toc90316391)

[Список литературы 25](#_Toc90316392)

[Приложение A Листинг программы. 26](#_Toc90316393)

# Реферат

Отчёт 34 стр., 13 рис.

Алгоритм Флойда

Цель исследования – разработка программы, способная, с помощью алгоритма Флойда, находить кратчайшие расстояния между вершинами графа.

В работе осуществлена реализация алгоритма Флойда, с помощью которого находятся все кратчайшие расстояния от всех вершин до всех.

# Введение

Алгоритм Флойда – это алгоритм поиска кратчайших расстояний во взвешенном графе. Данный алгоритм позволяет найти кратчайшее расстояние между любыми двумя вершинами в графе, при этом веса ребер могут быть как положительными, так и отрицательными.[1]

Алгоритм Флойда является эффективным для расчёта всех кратчайших расстояний в плотных графах, когда имеет место большое количество пар рёбер между парами вершин. В случае разреженных графов с рёбрами неотрицательного веса лучшим выбором считается использование алгоритма Дейкстры для каждого возможного узла. Если граф разрежен, у него имеются рёбра с отрицательным весом и отсутствуют циклы с отрицательным суммарным весом, то используется алгоритм Джонсона, который имеет ту же сложность, что и вариант с алгоритмом Дейкстры.[3]

Также являются известными алгоритмы с применением алгоритмов быстрого перемножения матриц, которые ускоряют вычисления в плотных графах, но они обычно имеют дополнительные ограничения (например, представление весов рёбер в виде малых целых чисел). Вместе с тем, из-за большого константного фактора времени выполнения преимущество при вычислениях над алгоритмом Флойда проявляется только на больших графах.

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

Целью данной курсовой работы является разработка программы на языке Си, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Флойда, осуществляющий поиск кратчайших расстояний между всеми вершинами графа.

# 1 Постановка задачи

Требуется разработать программу, которая, с помощью алгоритма Флойда, будет находить кратчайшие расстояния от каждой вершины до каждой.

Исходный граф в программе будет задаваться матрицей смежности. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности графа. Необходимо предусмотреть различные исходы алгоритма, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

# 2 Теоретическая часть

Граф G (см. рис. 1) задается множеством вершин X1, X2, ..., Xn. И множеством ребер, соединяющих между собой определенные вершины. Ребра ориентированы, что показывается стрелкой, которая указывает достижимость данной вершины, граф с такими ребрами называется ориентированным графом. Также каждое ребро имеет свой вес. Это делает граф взвешенным.[1]

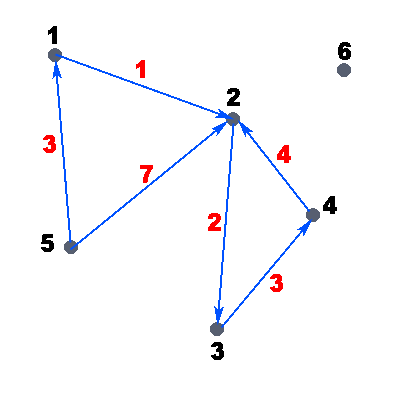


Рисунок 1 – Граф G

Большое число задач, связанных с графами, требует перебора вершин графа, т.е. просмотра каждой вершины в точности один раз (задачи поиска). Для того, чтобы написать алгоритм кратчайших путей, необходимо представить граф в памяти компьютера. Лучше всего представить граф с помощью матрицы смежности. Номер строки или столбца – это номер вершины, а значение элемента данной матрицы – это вес ребра между конкретными двумя вершинами.

Достоинства представления графа матрицей смежности:

1. Простота и наглядность
2. Проверка наличия ребра между вершинами осуществляется за один шаг (Сложность O (1))

Главным недостатком представления графа в виде матрицы смежности является большой расход памяти компьютера.

# 3 Описание алгоритма программы

Для программной реализации алгоритма Флойда понадобятся два двумерных массива. В первом массиве **M** будет представлен граф в виде матрицы смежности, а во втором массиве **A** будут храниться кратчайшие расстояния от всех вершин до всех. При этом, для корректной работы алгоритма Флойда, если между двумя вершинами нет связи, значение элемента в матрице будет равно SHRT\_MAX (нет пути).

Первым шагом, в матрицу **A** переносятся все связи и веса рёбер, которые находятся в матрице смежности **M**. Далее мы сравниваем каждый элемент матрицы **A**(A[i][j]) с суммой тех выделенных элементов, которые находятся либо в одном столбце, либо в одной строке с выбранным элементом (A[i][k] + A[k][j]). Если сумма соответствующих элементов меньше, чем сам элемент матрицы, то значение элемента станет равно сумме соответствующих элементов(A[i][j] = A[i][k] + A[k][j]).

Ниже представлен псевдокод функции Floyd ():

**Floyd (**M, n**)**

M – Матрица смежности.

n – кол-во вершин.

1. Создать матрицу **A**
2. Для i = 0, пока i < n, делать i = i + 1
   1. Для j = 0, пока j < n, делать j = j + 1
      1. A[i][j] = M[i][j]
   2. Конец цикла
3. Конец цикла
4. Для k = 0, пока k < n, делать k = k + 1
   1. Для i = 0, пока i < n, делать i = i + 1
      1. Для j = 0, пока j < n, делать j = j + 1
         1. Если A[i][k] + A[k][j] < A[i][j]
            1. A[i][j] = A[i][k] + A[k][j]
         2. Конец условия
      2. Конец цикла
   2. Конец цикла
5. Конец цикла
6. Вернуть матрицу **A**

# 4 Описание программы

Для написания данной программы использован язык программирования Си. Язык программирования Си - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней. Проект был создан в виде консольного приложения Win32 (Visual C++). Данная программа является многомодульной, поскольку состоит из нескольких функций: main, menu, createMatrix, numVertex, menuEdit, EditA, EditM, addVertex, delVertex, printMatrix, Floyd, Save, Read.

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

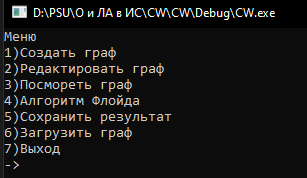


Рисунок 2 – Меню

1. Если пользователь выбрал пункт «1», то программа вызывает функцию numVertex(), где просит ввести размерность матрицы. После того, как пользователь ввёл размерность, программа создаёт матрицу смежности **M** и матрицу кратчайших путей **A**, которые пока заполнены нулями (см. рис. 3).

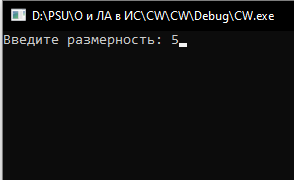


Рисунок 3 – Создание графа

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

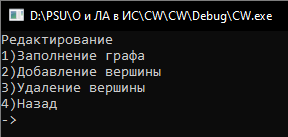


Рисунок 4 – Редактирование

* 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 EditA(), либо EditM() (см. рис. 5).

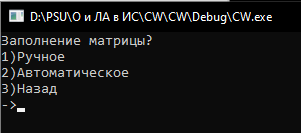


Рисунок 5 – Заполнение матрицы

* 1. Если пользователь выберет «2», то программа вызовет функцию addVertex() и увеличит размерность матрицы на 1.
  2. Если пользователь выберет «3», то программа спросит пользователя какую именно вершину он хочет удалить и вызовет функцию delVertex().

1. Если пользователь выберет «3», то программа вызовет функцию printMatrix(), и пользователь сможет увидеть свой граф, в виде матрицы смежности.
2. Если пользователь выберет «4», то программа вызовет функцию Floyd() и, с помощью алгоритма Флойда, высчитает кратчайшие расстояния от всех вершин до всех и представит результат в виде матрицы. После этого, на экран выводится матрица смежности **M** и матрица кратчайших расстояний **A** (см. рис. 6).

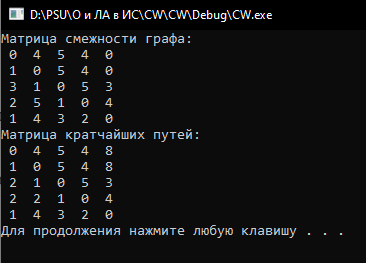


Рисунок 6 – Алгоритм Флойда

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

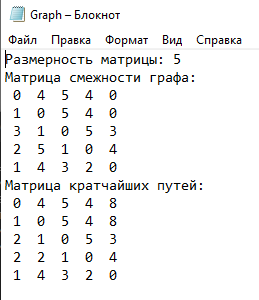


Рисунок 7 - Сохранение результата

1. Если пользователь выберет «6», то программа вызовет функцию Read() и из файла «Graph» возьмутся данные для того, чтобы заполнить матрицу смежности **M** (см. рис. 8).

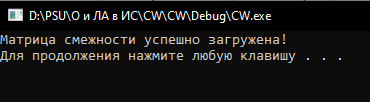


Рисунок 8 – Чтение из файла

# 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 Флойда (см. рис. 9,10,11,12).

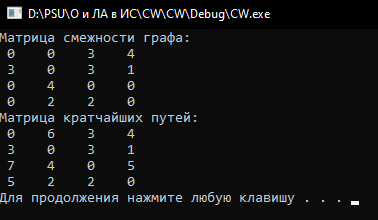


Рисунок 9 – Результат работы (4 вершины)

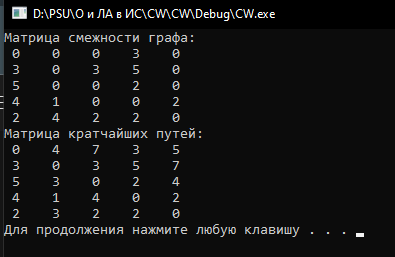


Рисунок 10 – Результат работы (5 вершин)

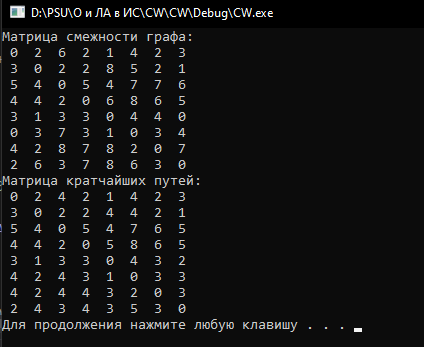


Рисунок 11 – результат работы (8 вершин)

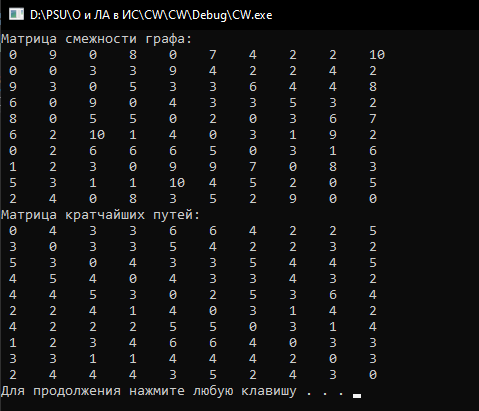


Рисунок 12 – Результат работы (10 вершин)

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 графа с 4 вершинами (рисунок 9).

Изначально матрица кратчайших расстояний будет точно такой же, как и матрица смежности графа. Поэтому при нулевом проходе матрица **A**0 будет просто копией матрицы смежности.

Таблица 1 – **A**0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | V0 | V1 | V2 | V3 |
| V0 | 0 | ∞ | 3 | 4 |
| V1 | 3 | 0 | 3 | 1 |
| V2 | ∞ | 4 | 0 | ∞ |
| V3 | ∞ | 2 | 2 | 0 |

При первом проходе необходимо выделить первую строку и первый столбец матрицы **A0** и на пересечении ненулевых и не бесконечных элементов матрицы по формуле выбирать кратчайшее расстояние.

Таблица 2 – **A**1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | V0 | V1 | V2 | V3 |
| V0 | 0 | ∞ | 3 | 4 |
| V1 | 3 | 0 | 3 или (3 + 3) | 1 или (3 + 4) |
| V2 | ∞ | 4 | 0 | ∞ |
| V3 | ∞ | 2 | 2 | 0 |

Далее, по алгоритму, необходимо заполнить **A**2, исходя из данных предыдущей матрицы. Теперь нужно выделить 2 строку и 2 столбец матрицы **A**1 и также смотреть на пересечение ненулевых и не бесконечных элементов, пытаясь найти кратчайшее расстояние.

Таблица 3 – **A**2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | V0 | V1 | V2 | V3 |
| V0 | 0 | ∞ | 3 | 4 |
| V1 | 3 | 0 | 3 | 1 |
| V2 | ∞ или (3 + 4) | 4 | 0 | ∞ или (4 + 1) |
| V3 | ∞ или (3 + 2) | 2 | 2 или (2 + 3) | 0 |

По такому же алгоритму, построил матрицы A3, A4.

Таблица 4 – **A**3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | V0 | V1 | V2 | V3 |
| V0 | 0 | ∞ или (4 + 3) | 3 | 4 или (3 + 5) |
| V1 | 3 или (7 + 3) | 0 | 3 | 1 или (3 + 5) |
| V2 | 7 | 4 | 0 | 5 |
| V3 | 5 или (7 + 2) | 2 или (4 + 2) | 2 | 0 |

Таблица 5 – A4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | V0 | V1 | V2 | V3 |
| V0 | 0 | 7 или (2 + 4) | 3 или (2 + 4) | 4 |
| V1 | 3 или (5 + 1) | 0 | 3 или (2 + 1) | 1 |
| V2 | 7 или (5 + 5) | 4 или (2 + 5) | 0 | 5 |
| V3 | 5 | 2 | 2 | 0 |

В итоге, я получил финальную матрицу A, которая полностью идентична той матрице, которую построила программа (см. рис 13).

Таблица 6 – **A**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | V0 | V1 | V2 | V3 |
| V0 | 0 | 6 | 3 | 4 |
| V1 | 3 | 0 | 3 | 1 |
| V2 | 7 | 4 | 0 | 5 |
| V3 | 5 | 2 | 2 | 0 |

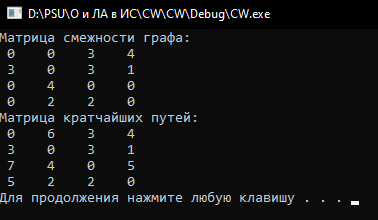


Рисунок 13 – Результат работы (4 вершины)

В результате ручного расчёта, выяснил, что программа работает корректно.

# Заключение

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

При выполнении данной курсовой работы были получены навыки разработки программ и освоен алгоритм Флойда. Приобретены навыки по осуществлению алгоритма поиска кратчайших расстояний. Углублены знания языка программирования Cи.

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. Foxford. Алгоритм Флойда. URL: <https://foxford.ru/wiki/informatika/algoritm-floyda> (дата обращения: 07.12.21\)
2. Medium. Floyd–Warshall algorithm. URL: <https://medium.com/unilecs/floyd-warshall-algorithm-d39dede79e67> (дата обращения: 08.12.21\)
3. Habr. Алгоритм Флойда — Уоршелла. URL: <https://habr.com/ru/post/105825/> (дата обращения: 08.12.21)

# Приложение A Листинг программы.

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_CRT\_NONSTDC\_NO\_WARNINGS

#define HEADER ("Курсовая работа\nТема: Алгоритм Флойда.\nВыполнил: Шмелёв Д.В.\n")

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <time.h>

#include <conio.h>

#include <Windows.h>

#include <queue>

using namespace std;

int\*\* M,\*\* A, n;

int numVertex()

{

int n;

printf("Введите размерность: ");

scanf("%d", &n);

return n;

}

int\*\* createMatrix()

{

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

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;

}

}

return M;

}

void printMatrix(int\*\* M)

{

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

{

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

{

if (M[i][j] == SHRT\_MAX)

{

printf(" %-3d ", NULL);

}

else

{

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

}

}

printf("\n");

}

}

void EditM()

{

int i, j, d, c;

printf("Введите номера смежных вершин и вес ребра между ними:\nДля завершения нажмите '-'\n");

c = getch();

if (c == '-')

{

return;

}

else

{

scanf("%d %d %d", &i, &j, &d);

if (i > n || j > n)

{

printf("Таких вершин в графе нет!\n");

}

else

{

M[i][j] = d;

}

printMatrix(M);

EditM();

}

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

{

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

{

if (M[i][j] == 0 && i != j)

{

M[i][j] = SHRT\_MAX;

}

}

}

}

void EditA()

{

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

{

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

{

if (i == j)

{

M[i][j] = 0;

}

else if (rand() % 100 < 30)

{

M[i][j] = SHRT\_MAX;

}

else

{

M[i][j] = (rand() % n) + 1;

}

}

}

}

void Save(int\*\* M, int\*\* A)

{

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 (M[i][j] == SHRT\_MAX)

{

fprintf(G, " %d ", NULL);

}

else

{

fprintf(G, " %d ", M[i][j]);

}

}

fprintf(G, "\n");

}

fprintf(G, "Матрица кратчайших путей:\n");

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

{

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

{

if (A[i][j] == SHRT\_MAX)

{

fprintf(G, " %d ", NULL);

}

else

{

fprintf(G, " %d ", A[i][j]);

}

}

fprintf(G, "\n");

}

fclose(G);

}

void Read()

{

FILE\* G;

if (!fopen("Graph", "r"))

{

printf("Невозможно открыть файл!\n");

system("pause");

return;

}

G = fopen("Graph", "r");

fseek(G, 21, SEEK\_SET);

fscanf(G, "%d", &n);

M = createMatrix();

fseek(G, 28, SEEK\_CUR);

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

{

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

{

fscanf(G, " %d ", &M[i][j]);

}

}

fclose(G);

}

void printList(int\*\* M)

{

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

{

printf("%d: ", i);

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

{

if (M[i][j] != 0 && M[i][j] != SHRT\_MAX)

{

printf(" %d ", j);

}

}

printf("\n");

}

}

int\*\* Floyd()

{

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

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

{

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

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

{

A[i][j] = M[i][j];

}

}

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

{

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

{

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

{

if (A[i][k] + A[k][j] < A[i][j])

{

A[i][j] = A[i][k] + A[k][j];

}

}

}

}

return A;

}

void addVertex()

{

M = (int\*\*)(realloc(M, (n + 1) \* sizeof(int\*\*)));

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

{

M[i] = (int\*)(realloc(M[i], (n + 1) \* sizeof(int)));

}

M[n] = (int\*)(malloc((n + 1) \* sizeof(int)));

n++;

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

{

M[n - 1][i] = 0;

M[i][n - 1] = 0;

}

}

void delVertex(int v)

{

for (int i = v; i < n - 1; i++)

{

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

{

M[i][j] = M[i + 1][j];

}

}

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

{

for (int j = v; j < n - 1; j++)

{

M[i][j] = M[i][j + 1];

}

}

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

{

M[i] = (int\*)(realloc(M[i], (n - 1) \* sizeof(int)));

}

free(M[n - 1]);

n--;

}

void menuEdit()

{

int a, c;

do

{

system("cls");

printf("Редактирование\n1)Заполнение графа\n2)Добавление вершины\n3)Удаление вершины\n4)Назад\n->");

c = getch();

switch (c)

{

case '1':

system("cls");

printf("Заполнение матрицы?\n1)Ручное\n2)Автоматическое\n3)Назад\n->");

a = getch();

if (a == '1')

{

EditM();

}

else if (a == '2')

{

EditA();

system("cls");

printf("Матрица успешно заполнена!\n");

system("pause");

}

else if (a == '3')

{

return;

else

{

printf("Ошибка!\n");

system("pause");

}

break;

case '2':

system("cls");

addVertex();

printf("Вершина успешно добавлена!\n");

system("pause");

break;

case '3':

system("cls");

printf("Введите номер удаляемой вершины: ");

scanf("%d", &a);

if (a < n)

{

delVertex(a);

}

else

{

system("cls");

printf("Такой вершины нет!\n");

system("pause");

return;

}

break;

case '4':

return;

default:

system("cls");

printf("Ошибка!\n");

system("pause");

break;

}

} while (c != '4');

}

void menu()

{

int c;

do

{

system("cls");

printf("Меню\n1)Создать граф\n2)Редактировать граф\n3)Посмореть граф\n4)Алгоритм Флойда\n5)Сохранить результат\n6)Загрузить граф\n7)Выход\n->");

c = getch();

switch (c)

{

case '1':

system("cls");

n = numVertex();

M = createMatrix();

A = createMatrix();

break;

case '2':

menuEdit();

break;

case '3':

system("cls");

printf("Матрица смежности графа:\n");

printMatrix(M);

system("pause");

break;

case '4':

system("cls");

printf("Матрица смежности графа:\n");

printMatrix(M);

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

A = Floyd();

printMatrix(A);

system("pause");

break;

case '5':

system("cls");

printf("Результат сохранён в файл Graph.\n");

Save(M, A);

system("pause");

break;

case '6':

system("cls");

Read();

printf("Матрица смежности успешно загружена!\n");

system("pause");

break;

case '7':

exit(NULL);

default:

printf("Ошибка!\n");

system("pause");

system("cls");

break;

}

} while (c != '7');

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(NULL));

printf(HEADER);

Sleep(1337);

menu();

system("pause");

return 0;

}