

Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

По дисциплине: «Л и ОА в ИЗ»


на тему: «Реализация алгоритма Флойда-Уоршелла поиска кратчайших
путей в графе»

Выполнили студенты группы 22ВВП1:

Изосин М. А.

Принял:

к.э.н доцент Акифьев. И. В.

 22.01.24г.
Хорошо

Пенза, 2023

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

 М.А. Митрохин

«__» __ 20__ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Студенту Митрохину Михаилу Александровичу Группа 22ВВ2.
Тема проекта Реализация алгоритма Флойда-Уоршелла поиска кратчайших путей в графе.

Исходные данные (технические требования) на проектирование

- Разработка алгоритма и программного обеспечения к нему с соответствием требованиям задания курсового проекта.
- Технические требования:
1. Постановка задачи;
 2. Теоретическая часть задания;
 3. Описание алгоритма построения графа;
 4. Пример ручного расчета графа (объясняющий, как выполняется каждая часть работы алгоритма);
 5. Описание самой программы;
 6. Тесты;
 7. Список литературы;
 8. Листы оформления;
 9. Резюме работы программы;

Объем работы по курсу

1. Расчетная часть

Ручной расчет работы алгоритма

2. Графическая часть

Схемы алгоритма в форме блок-схем

3. Экспериментальная часть

Тестирование программы.
Подготовка работы, оформленной на тестовых
картах данных

Срок выполнения проекта по разделам

1. Изучение теоретической части курсов
2. Разработка алгоритмов программы
3. Разработка программы
4. Тестирование и завершение разработки программы
5. Оформление итогового учебного задания
- 6.
- 7.
- 8.

Дата выдачи задания "6" сентября

Дата защиты проекта "22" января

Руководитель Акиреев. И. В.

Задание получил "3" сентября 20 г.

Студент Мусин В. А. (Мусин)

Содержание

Реферат	4
Введение	5
Постановка задачи.....	7
Теоретическая часть задания	8
Описание алгоритма программы	11
Описание программы.....	14
Тестирование	20
Заключение	24
Список литературы	25
Приложение А	26
Приложение В.....	28

Реферат

Отчет 38 стр, 17 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ОРГРАФ, ПОИСК КРАТЧАЙШИХ ПУТЕЙ.

Реализация алгоритма Флойда-Уоршелла.

Цель исследования – разработка программы, выполняющей алгоритм Флойда-Уоршелла.

В работе рассмотрен способ нахождения кратчайших расстояний между всеми вершинами взвешенного графа без циклов с отрицательными весами.

Введение

Курсовая работа посвящена одному из фундаментальных вопросов теории графов и вычислительной практики, а именно — задаче поиска кратчайших путей между всеми парами вершин графа. Эта проблема имеет не только теоретическое значение, пронизывающее глубокие слои математики и информатики, но и широкое практическое применение, начиная от маршрутизации в компьютерных сетях и заканчивая планированием логистики и расписаний транспортных средств. Среди разнообразия алгоритмов, применяемых для решения этой задачи, алгоритм Флойда-Уоршелла занимает особое место. Он представляет собой простой и эффективный метод расчёта кратчайших путей во взвешенном графе, идеально подходящий для плотных графов, где требуется вычислить расстояния между всеми парами вершин. Особенностью данного алгоритма является его универсальность и простота реализации на различных программных платформах.

В рамках данной курсовой работы будет произведен анализ теоретических основ алгоритма Флойда-Уоршелла, особенности его функционирования и этапы реализации. Также будет проведено исследование существующих модификаций и улучшений алгоритма, направленных на повышение его эффективности. Особое внимание будет уделено аспектам оптимизации работы данного алгоритма для обработки больших наборов данных, что актуально для современных приложений. Целью работы является изучение принципов работы алгоритма Флойда-Уоршелла, разработка программного кода для его реализации и экспериментальная проверка эффективности на различных входных данных. В ходе работы будут поставлены и решены следующие задачи: описание теоретической базы алгоритма, разработка программной реализации алгоритма, проведение тестирования программы, анализ результатов и

формирование выводов по работе. Таким образом, данная курсовая работа позволит глубже понять аспекты проектирования алгоритмов на графах и освоить практические навыки разработки и оптимизации алгоритмических процедур, что имеет важное значение в рамках подготовки специалиста в области информационных технологий и вычислительной математики.

Постановка задачи

Требуется разработать программу, которая реализует алгоритм Флойда-Уоршелла.

Ориентированный граф должен задаваться матрицей смежности. Программа должна работать так, чтобы у пользователя был выбор самому вводить данные или случайно генерировать матрицу. После обработки данных на экран должна выводиться матрица смежности кратчайших путей и сами пути.

Устройство ввода- клавиатура.

Задания выполняются в соответствии с вариантом №7.

Теоретическая часть задания

В алгоритме Флойда используется матрица A размером $n \times n$, в которой вычисляются длины кратчайших путей. Элемент $A[i,j]$ равен расстоянию от вершины i к вершине j , которое имеет конечное значение, если существует ребро (i,j) , и равен бесконечности в противном случае.

Основная идея алгоритма. Пусть есть три вершины i, j, k и заданы расстояния между ними. Если выполняется неравенство $A[i,k] + A[k,j] < A[i,j]$, то целесообразно заменить путь $i \rightarrow j$ путем $i \rightarrow k \rightarrow j$. Такая замена выполняется систематически в процессе выполнения данного алгоритма.

Шаг 0. Определяем начальную матрицу расстояния A_0 и матрицу последовательности вершин S_0 . Каждый диагональный элемент обеих матриц равен 0, таким образом, показывая, что эти элементы в вычислениях не участвуют. Полагаем $k = 1$.

Основной шаг k . Задаем строку k и столбец k как ведущую строку и ведущий столбец. Рассматриваем возможность применения замены описанной выше, ко всем элементам $A[i,j]$ матрицы A_{k-1} . Если выполняется неравенство $A[i,k] + A[k,j] < A[i,j]$, ($i \neq k, j \neq k, i \neq j$), тогда выполняем следующие действия:

1. создаем матрицу A_k путем замены в матрице A_{k-1} элемента $A[i,j]$ на сумму $A[i,k] + A[k,j]$;
2. создаем матрицу S_k путем замены в матрице S_{k-1} элемента $S[i,j]$ на k . Полагаем $k = k + 1$ и повторяем шаг k .

Таким образом, алгоритм Флойда делает n итераций, после i -й итерации матрица A будет содержать длины кратчайших путей между любыми двумя парами вершин при условии, что эти пути проходят через вершины от первой до i -й. На каждой итерации перебираются все пары

вершин и путь между ними сокращается при помощи i -й вершины.

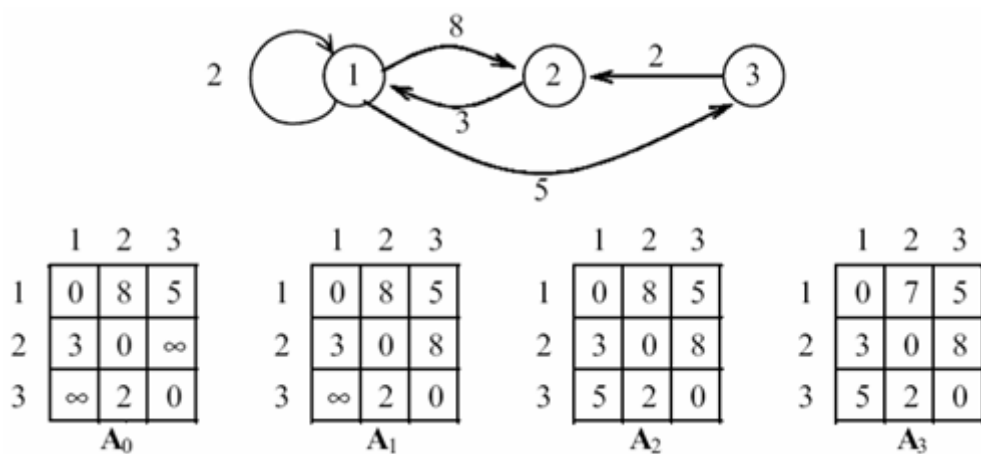


Рисунок 1

Пример работы

1. Дан граф

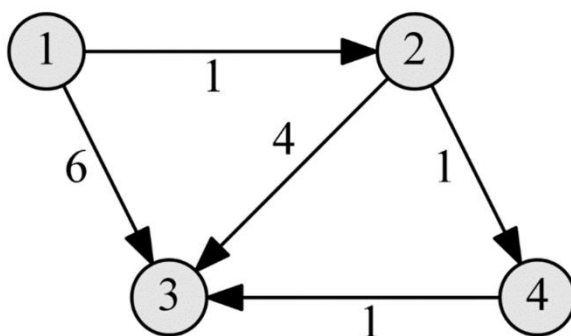


Рисунок 2

2. Выберем вершину

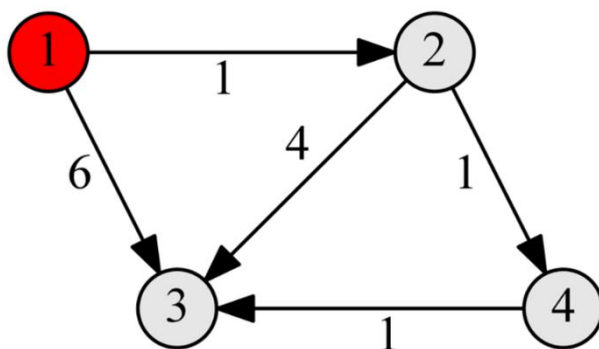


Рисунок 3

3. Выберем потенциально более короткий путь

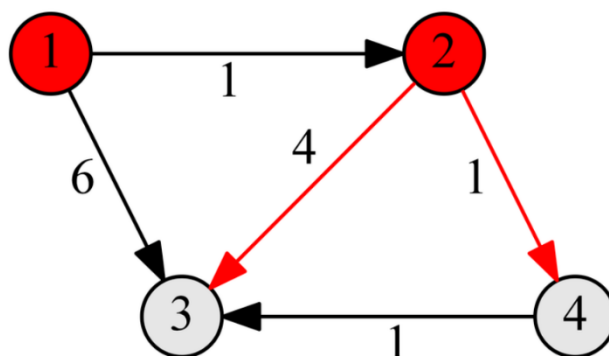


Рисунок 4

4. Выберем потенциально более короткий путь с новой вершины

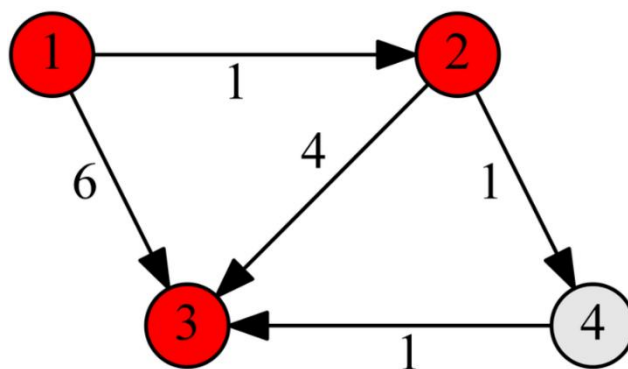


Рисунок 5

5. Самый короткий путь найден

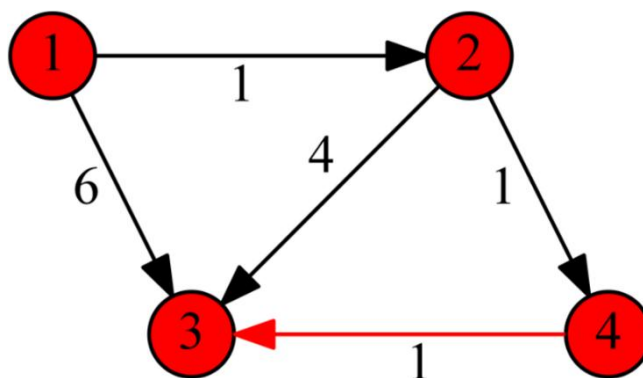


Рисунок 6

Описание алгоритма программы

Для программной реализации алгоритма понадобится указатель на двумерный массив A , размерами $size \times size$, в котором хранятся величины вершин ориентированного взвешенного графа.

Создаем указатель 'p' на матрицу предшествования, размерности $size \times size$. В ней будем хранить информацию о предшествующих вершинах на кратчайших путях/

Инициализируем матрицу 'p'. Если элемент матрицы 'p[i][j]' меньше "бесконечности" (999) и 'i' не равно 'j', то указываем что предшествующим узлом для 'j' является узел 'i'. В противном случае ставим -1.

Внешний цикл ('k') перебирает все вершины в качестве "промежуточных" точек.

Два внутренних цикла ('i' и 'j') перебирают все возможные пары вершин.

Во внутренних циклах, для каждой пары вершин (i, j) алгоритм проверяет, сможет ли он улучшить текущее известное расстояние между этими вершинами, используя вершину k как промежуточную. Если новый путь короче, он обновляет значение $x[i][j]$ и $p[i][j]$.

Выводим все маршруты, используя матрицу предшествования 'p'. Вывод осуществляется в обратном порядке: начиная от конечной вершины и двигаемся в направлении начальной вершины, пока не достигнем ее или не столкнемся с недостижимостью вершины ('u == -1').

Ниже представлен псевдокод функций программы.
floyd()

1. Создаем указатель на указатель 'p' на матрицу, размерами 'y' x 'y', и выделяем под него память.

// Инициализация матрицы предшествования

2. для i от 0 до ' y ':

3. $p[i]$ = выделяем память под каждый элемент.

4. для j от 0 до ' y ':

5. если $x[i][k] < \text{'INF'}$ и i не равно j :

6. $p[i][j] = i$

7. иначе:

8. $p[i][j] = -1$

9. для k от 0 до ' y ':

10. для i от 0 до ' y ':

11. для j от 0 до ' y ':

// если найден более короткий путь между i и j через k и через
вершину k существует путь между i и j

12. если $(x[i][k] < \text{INF} \text{ и } x[k][j] < \text{INF})$

// обновляем значение кратчайшего пути между i и j

13. если $(x[i][j] > (x[i][k] + x[k][j]))$:

14. $x[i][j] = x[i][k] + x[k][j]$

15. $p[i][j] = p[k][j]$

16. конец условия

17. конец условия

18. конец цикла j

19. конец цикла i

20. конец цикла k

21.Вывод "Маршруты между вершинами:",

22.для i от 0 до 'y':

23.для j от 0 до 'y':

24.если i не равно j :

25. вывести "Кратчайший путь из ", $i + 1$, " в ", $j + 1$, ": "

26. целое $u = j$

27.пока u не равно i

28.Если u равно -1

29.Вывести недостижимо

30.Прерывание цикла

31.Вывести $u+1$, "<-"

32. $u = p[i][u]$

33.конец цикла пока

34.Вывести $i+1$

35.конец условия

36.конец цикла j

37.конец цикла i

38.конец функции

Описание программы

Для написания данной программы был использован язык программирования СИ.

Проект был создан в виде консольного приложения в программе VS 2019.

Данная программа является многомодульной, поскольку состоит из нескольких функций: main, creatematrix, floyd, scan, input, scan_key, print.

Работа программы начинается с вызова меню. Если пользователь выберет пункт под номером 1, то будет сгенерирована случайная матрица. Если выбрать пункт 2, то пользователь введёт размер матрицы и запишет матрицу с клавиатуры. Если выбрать пункт 3, то выведется на экран титульный лист. Пункт 0 завершает работу программы.

Функция main

```
int main()
{
    setlocale(LC_ALL, "rus");
    srand(time(NULL));
    int size;      //Размер матрицы
    int** A = 0;   //Указатель на матрицу смежности
    int r = 0;      //Для поиска отрицательного цикла
    int menu = 1;  //переменная меню
    char name[50];  //Наименование файла
    int cycle = 0;  //Для числа отрицательных циклов
    char results[20]; //Для содания файла
    int way[20] = { 999 }; //Массив для пути
    int* wway = way; //Указатель на массив пути
    int q = 0;      //Для изолированной вершины

    //_____
    _____
}
```

```

while (menu != 0)
{
    printf("Выберите пункт:\n");
    printf("Для генерации матрицы случайными числами нажмите 1:\n");
    printf("Для ввода матрицы с клавиатуры нажмите 2:\n");
    printf("Для просмотра титульного листа нажмите 3:\n");
    printf("Для выхода нажмите 0:\n");

    menu = scan();

    while (menu > 4)
    {
        printf("Ошибка!Вы ввели большое число.");
        menu = scan();
    }
    switch (menu)
    {
        case 0:
            return 0;
            break;
        case 1:
            printf("Введите размер матрицы.\n");

            size = scan(); //Проверка ввода

            A = (int**)malloc(size * sizeof(int*));
            for (int i = 0; i < size; i++)
            {
                A[i] = (int*)malloc(size * sizeof(int));    //Создание матрицы
            }

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

```



```

{
    for (int j = 0; j < size; j++)
    {
        A[i][j] = 0;    //Обнуляем матрицу
    }
}

creatematrix(A, size); //Вызываем функцию генерации матрицы

print(A, size); //Выводим матрицу

floyd(A, size); //Вызываем функцию Флойда-Уоршелла

for (int i = 0; i < size; i++)
    if (A[i][i] < 0)
        r++;
if (r >= 1)
{
    printf("Обнаружено: %d отрицательных циклов\n", r);
    cycle = 1;
}

if (!cycle)
    print(A, size); //Выводим матрицу

FILE* t;

t = fopen("results.txt", "w");

fprintf(t, "%d", size);
fprintf(t, "\n");

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

```

```

        for (int j = 0; j < size; j++)
        {
            fprintf(t, "%d ", A[i][j]);
        }
        fprintf(t, "\n");
    }
    fclose(t);
    break;
case 2:
    printf("Введите размер матрицы.\n");

    size = scan(); //Проверка ввода

    A = scan_key(size);

    print(A, size); //Выводим матрицу

    floyd(A, size); //Вызываем функцию Флойда-Уоршелла

    for (int i = 0; i < size; i++)
        if (A[i][i] < 0)
            r++;
    if (r >= 1)
    {
        printf("Обнаружено: %d отрицательных циклов\n", r);
        cycle = 1;
    }

    if (!cycle)
        print(A, size); //Выводим матрицу

    for (int i = 0; i < 20; i++)
    {
        if (wway[i] != 0)

```


Программа пользователю выбрать пункт.

При выборе пункта 1 программа просит указать размер матрицы. Затем, создается случайная матрица смежности ориентированного графа и выводится на экран. После, вызывается функция Флойда-Уоршелла и происходит вывод результирующей матрицы кратчайших путей и сами пути на экран. Затем, результат записывается в файл.

При выборе пункта 2 программа просит указать размер матрицы. Затем, вводится с клавиатуры случайная матрица смежности ориентированного графа и выводится на экран. После, вызывается функция Флойда-Уоршелла и происходит вывод результирующей матрицы кратчайших путей и сами пути на экран. Затем, результат записывается в файл.

При выборе пункта 3 на экран выводится титульный лист.

Потом начинается новый цикл и снова выводится меню, чтобы пользователю не было необходимо перезапускать программу.

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.

Таблица 1

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод меню в консоль	Верно
Генерация матрицы случайным образом и ручным вводом размера	Вывод сообщения о вводе размера матрицы И выводе результатов программы в консоль	Верно
Вывод титульного листа	Вывод титульного листа на экран	Верно

Продолжение таблицы 1

Описание теста	Ожидаемый результат	Полученный результат
Сохранение в файл	Создание файла, содержащего результаты программы	Верно
Выход из программы	Завершение работы программы	Верно

```

C:\ F:\ConsoleApplication41\Debug\ConsoleApplication41.exe
Выберите пункт:
Для генерации матрицы случайными числами нажмите 1:
Для ввода матрицы с клавиатуры нажмите 2:
Для просмотра титульного листа нажмите 3:
Для выхода нажмите 0:
Введите:

```

Рисунок 7 - Вывод меню в консоль

```

C:\ F:\ConsoleApplication41\Debug\ConsoleApplication41.exe
Выберите пункт:
Для генерации матрицы случайными числами нажмите 1:
Для ввода матрицы с клавиатуры нажмите 2:
Для просмотра титульного листа нажмите 3:
Для выхода нажмите 0:
Введите: 1
Введите размер матрицы.
Введите: 6
  0    1    14    17    25    18
999    0    19  999  999    4
20   23    0  999    2    8
19   15   22    0  999    1
24  999    8  999    0  999
999  999  999   19   15    0

```

Рисунок 8 - Случайная генерация матрицы

```

F:\ConsoleApplication41\Debug\ConsoleApplication41.exe

Выберите пункт:
Для генерации матрицы случайными числами нажмите 1:
Для ввода матрицы с клавиатуры нажмите 2:
Для просмотра титульного листа нажмите 3:
Для выхода нажмите 0:
Введите: 2
Введите размер матрицы.
Введите: 4
999 - отсутствие ребра
Введите строку 1 столбец 2:
12
Введите строку 1 столбец 3:
67
Введите строку 1 столбец 4:
0
Введите строку 2 столбец 1:
999
Введите строку 2 столбец 3:
12
Введите строку 2 столбец 4:
2
Введите строку 3 столбец 1:
999
Введите строку 3 столбец 2:
0
Введите строку 3 столбец 4:
1
Введите строку 4 столбец 1:
32
Введите строку 4 столбец 2:
11
Введите строку 4 столбец 3:
0

    0   12   67   0
999   0   12   2
999   0   0   1
32   11   0   0

```

Рисунок 9 - Ручной ввод матрицы

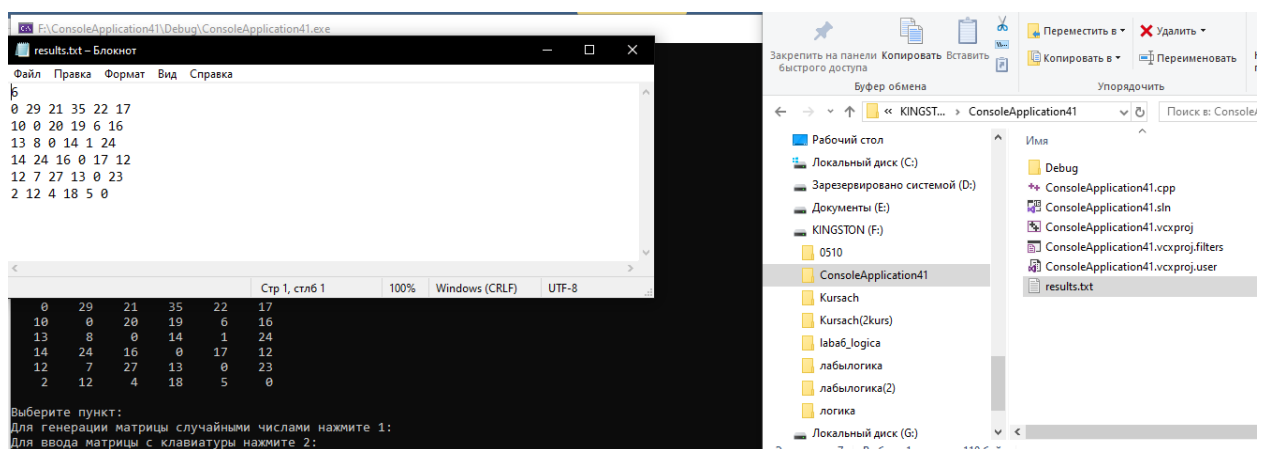


Рисунок 10 - Сохранение результата в файл

```
Выберите пункт:  
Для генерации матрицы случайными числами нажмите 1:  
Для ввода матрицы с клавиатуры нажмите 2:  
Для просмотра титульного листа нажмите 3:  
Для выхода нажмите 0:  
Введите: 0  
  
F:\ConsoleApplication41\Debug\ConsoleApplication41.exe (процесс 42712) завершил работу с кодом 0.  
Нажмите любую клавишу, чтобы закрыть это окно...
```

Рисунок 11 - Выход из программы

Заключение

В процессе создания данного проекта разработана программа, реализующая алгоритм Флойда-Уоршелла для поиска кратчайшего пути в графе в среде Microsoft Visual Studio 2019.

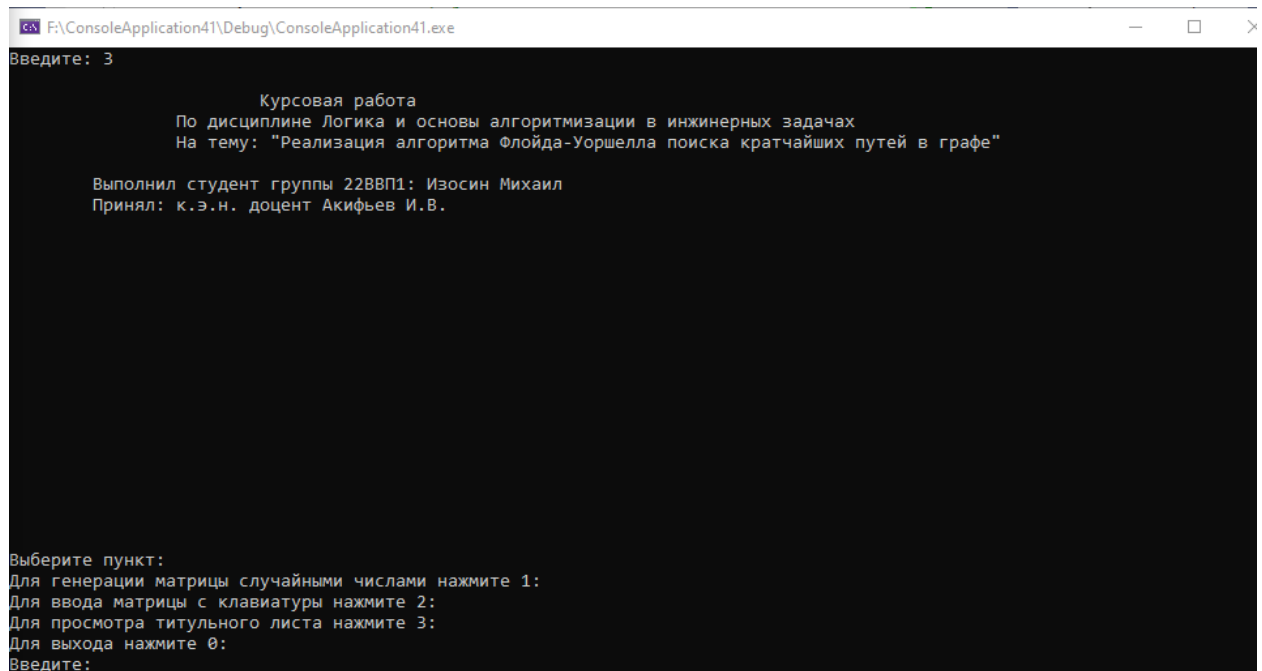
При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей. Приобретены навыки по осуществлению алгоритма Флойда-Уоршелла. Углублены знания языка программирования C.

Список литературы

1. Керниган Б. Ритчи Д. Язык программирования С. 2001
2. И. С. Солдатенко. Основы программирования на языке СИ. 2017
3. В. Г. Давыдов. Программирование и основы алгоритмизации.

2003

Приложение А



```
F:\ConsoleApplication41\Debug\ConsoleApplication41.exe
Введите: 3

Курсовая работа
По дисциплине Логика и основы алгоритмизации в инженерных задачах
На тему: "Реализация алгоритма Флойда-Уоршелла поиска кратчайших путей в графе"

Выполнил студент группы 22ВВР1: Изосин Михаил
Принял: к.э.н. доцент Акифьев И.В.

Выберите пункт:
Для генерации матрицы случайными числами нажмите 1:
Для ввода матрицы с клавиатуры нажмите 2:
Для просмотра титульного листа нажмите 3:
Для выхода нажмите 0:
Введите:
```

Рисунок 12 - Вывод титульника в консоли

```
Выберите пункт:
Для генерации матрицы случайными числами нажмите 1:
Для ввода матрицы с клавиатуры нажмите 2:
Для просмотра титульного листа нажмите 3:
Для выхода нажмите 0:
Введите:
```

Рисунок 13 – Меню

```
Выберите пункт:
Для генерации матрицы случайными числами нажмите 1:
Для ввода матрицы с клавиатуры нажмите 2:
Для просмотра титульного листа нажмите 3:
Для выхода нажмите 0:
Введите: 1
Введите размер матрицы.
Введите: 5
    0  999   11   13   20
    7   0   10   11   16
    3   29   0  999  999
  999   29   25   0   28
    1  999  999  999   0
```

Рисунок 15 - Случайная генерация матрицы

```

Введите: 2
Введите размер матрицы.
Введите: 4
999 - отсутствие ребра
Введите строку 1 столбец 2:
1
Введите строку 1 столбец 3:
0
Введите строку 1 столбец 4:
999
Введите строку 2 столбец 1:
12
Введите строку 2 столбец 3:
8
Введите строку 2 столбец 4:
3
Введите строку 3 столбец 1:
88
Введите строку 3 столбец 2:
12
Введите строку 3 столбец 4:
0
Введите строку 4 столбец 1:
999
Введите строку 4 столбец 2:
1
Введите строку 4 столбец 3:
67

```

0	1	0	999
12	0	8	3
88	12	0	0
999	1	67	0

Рисунок 14 - Запись матрицы через консоль

```

Маршруты между вершинами:
Кратчайший путь из 1 в 2: 2 <- 1
Кратчайший путь из 1 в 3: 3 <- 1
Кратчайший путь из 1 в 4: 4 <- 3 <- 1
Кратчайший путь из 2 в 1: 1 <- 2
Кратчайший путь из 2 в 3: 3 <- 2
Кратчайший путь из 2 в 4: 4 <- 2
Кратчайший путь из 3 в 1: 1 <- 2 <- 4 <- 3
Кратчайший путь из 3 в 2: 2 <- 4 <- 3
Кратчайший путь из 3 в 4: 4 <- 3
Кратчайший путь из 4 в 1: 1 <- 2 <- 4
Кратчайший путь из 4 в 2: 2 <- 4
Кратчайший путь из 4 в 3: 3 <- 2 <- 4

```

0	1	0	0
12	0	8	3
13	1	0	0
13	1	9	0

Рисунок 16 - Вывод результата работы в консоль

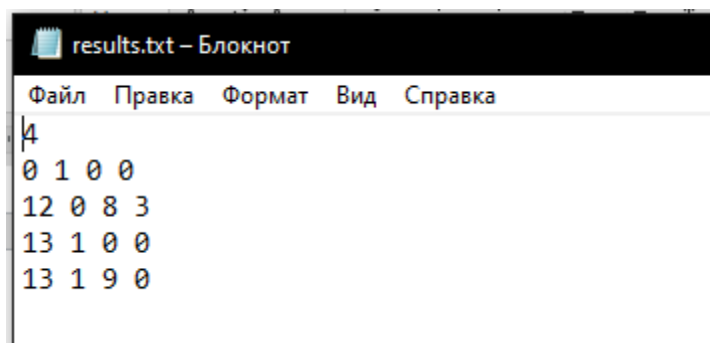


Рисунок 17 - Выгрузка результата работы в файл

Приложение В

Листинг программы.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <locale.h>
#include <malloc.h>
#include <time.h>
#include <stdlib.h>
#include <ctype.h>

#define INF 999          //Нет дорожки из одной вершины в другую

//Функция создания взвешенной ориентированной матрицы смежности
void creatematrix(int** x, int y)    //Принимает указатель на матрицу А и ее размер
size
{
    for (int i = 0; i < y; i++)
    {
        int count = 0; //Чтобы не было изолированной вершины
        for (int j = 0; j < y; j++)
        {
            int N = rand() % 2;    //50% шанс, что будет ребро
            if (N == 0)
            {
                x[i][j] = INF; // вводим бесконечность(999)
                count++;
            }
            else
                x[i][j] = rand() % 31; //Заполняем матрицу числами,
диапазоном [0:15]

            if (count == y - 1)
                x[i][j] = rand() % 31; //Заполняем матрицу числами,
диапазоном [0:15]
        }
    }
}
```

```

        x[i][i] = 0;
    }
}

//Функция, выполняющая алгоритм Флойда-Уоршелла
void floyd(int** x, int y)    //Принимает указатель на матрицу A и ее размер size
{
    int** p = (int**)malloc(y * sizeof(int*));

    // Инициализация матрицы предшествования
    for (int i = 0; i < y; i++) {
        p[i] = (int*)malloc(y * sizeof(int));
        for (int j = 0; j < y; j++) {
            if (x[i][j] < INF && i != j)
                p[i][j] = i;
            else
                p[i][j] = -1;
        }
    }

    for (int k = 0; k < y; k++)
    {
        for (int i = 0; i < y; i++)
        {
            for (int j = 0; j < y; j++)
            {
                if (x[i][k] < INF && x[k][j] < INF)
                {
                    if (x[i][j] > (x[i][k] + x[k][j]))    //Если сумма
веса дорог < веса записанной дороги
                    {
                        x[i][j] = x[i][k] + x[k][j];    //Записываем
новый вес дороги
                    }
                }
            }
        }
    }
}

```

```

p[i][j] = p[k][j]; // Обновление
предшествования
    }
    }
}
}
}

```

```

// Вывод маршрутов
printf("Маршруты между вершинами:\n");
for (int i = 0; i < y; i++) {
    for (int j = 0; j < y; j++) {
        if (i != j) {
            printf("Кратчайший путь из %d в %d: ", i + 1, j + 1);
            int u = j;
            while (u != i) {
                if (u == -1) {
                    printf("недостижимо");
                    break;
                }
                printf("%d <- ", u + 1);
                u = p[i][u];
            }
            printf("%d\n", i + 1);
        }
    }
}
}

```

```

//Функция, выполняющая проверку ввода
int scan() //Принимает размер матрицы size
{
    char input[20];

```

```

int input_success = 0;
int y;

while (!input_success) {
    printf("Введите: ");
    if (scanf("%s", input) == 1) {
        // Проверка, является ли введенный символ числом
        int is_digit = 1;
        for (int i = 0; input[i] != '\0'; ++i) {
            if (!isdigit(input[i])) {
                is_digit = 0;
                break;
            }
        }

        if (is_digit) {
            y = atoi(input);
            input_success = 1;
        }
        else {
            printf("Ошибка! \n");
        }
    }
    else {
        printf("Некорректный ввод. \n");
        // Очистка буфера ввода
        while (getchar() != '\n');
    }
}

return y;
}

//Функция, выполняющая проверку ввода(можно отрицательные числа)
int input() {

```



```

char input[20];
int input_success = 0;
int y;

while (!input_success) {
    if (scanf("%s", input) == 1) {
        // Проверка, является ли введенный символ числом
        int is_digit = 1;
        int start_index = 0;

        // Проверка наличия знака минуса в начале строки
        if (input[0] == '-') {
            start_index = 1;
        }

        for (int i = start_index; input[i] != '\0'; ++i) {
            if (!isdigit(input[i])) {
                is_digit = 0;
                break;
            }
        }

        if (is_digit) {
            y = atoi(input);
            input_success = 1;
        }
        else {
            printf("Ошибка! \n");
        }
    }
    else {
        printf("Некорректный ввод. \n");
        // Очистка буфера ввода
        while (getchar() != '\n');
    }
}

```

```

    }

}

return y;

}

//Функция, выполняющая ввод матрицы с клавиатуры
int** scan_key(int y)
{
    int** A = (int**)malloc(y * sizeof(int*));
    for (int i = 0; i < y; i++)
    {
        A[i] = (int*)malloc(y * sizeof(int)); //Создание матрицы
    }

    for (int i = 0; i < y; i++)
    {
        for (int j = 0; j < y; j++)
        {
            A[i][j] = 0;    //Обнуляем матрицу
        }
    }

    printf("999 - отсутствие ребра\n");
    for (int i = 0; i < y; i++) {
        for (int j = 0; j < y; j++) {
            if (!(i == j)) {
                A[i][j] = 1000;
                while (A[i][j] > 999 || A[i][j] < -999)
                {
                    printf("Введите строку %d столбец %d:\n", i + 1, j
+ 1);

                    A[i][j] = input(); // Используем функцию для
ввода числа
                }
            }
        }
    }
}

```

```

        }
        else {
            A[i][j] = 0;
        }
    }
}

return A;
}

//Функция, выполняющая вывод матрицы на экран
void print(int** x, int y)
{
    for (int i = 0; i < y; i++)
    {
        for (int j = 0; j < y; j++)
        {
            printf("%5d ", x[i][j]);    //Выводим матрицу
        }
        printf("\n");
    }
    printf("\n");
}

int main()
{
    setlocale(LC_ALL, "rus");
    srand(time(NULL));
    int size;    //Размер матрицы
    int** A = 0; //Указатель на матрицу смежности
    int r = 0;    //Для поиска отрицательного цикла
    int menu = 1; //переменная меню
    char name[50];    //Наименование файла
    int cycle = 0; //Для числа отрицательных циклов

```

```
char results[20];      //Для содания файла
int way[20] = { 999 }; //Массив для пути
int* wway = way;      //Указатель на массив пути
int q = 0;             //Для изолированной вершины
```

```
//_____

while (menu != 0)
{
    printf("Выберите пункт:\n");
    printf("Для генерации матрицы случайными числами нажмите 1:\n");
    printf("Для ввода матрицы с клавиатуры нажмите 2:\n");
    printf("Для просмотра титульного листа нажмите 3:\n");
    printf("Для выхода нажмите 0:\n");

    menu = scan();

    while (menu > 4)
    {
        printf("Ошибка!Вы ввели большое число.");
        menu = scan();
    }
    switch (menu)
    {
    case 0:
        return 0;
        break;
    case 1:
        printf("Введите размер матрицы.\n");

        size = scan(); //Проверка ввода
```

матрицы

```
A = (int**)malloc(size * sizeof(int*));
for (int i = 0; i < size; i++)
{
    A[i] = (int*)malloc(size * sizeof(int));    //Создание

}

for (int i = 0; i < size; i++)
{
    for (int j = 0; j < size; j++)
    {
        A[i][j] = 0;    //Обнуляем матрицу
    }
}

creatematrix(A, size); //Вызываем функцию генерации матрицы

print(A, size); //Выводим матрицу

floyd(A, size); //Вызываем функцию Флойда-Уоршелла

for (int i = 0; i < size; i++)
    if (A[i][i] < 0)
        r++;
if (r >= 1)
{
    printf("Обнаружено: %d отрицательных циклов\n", r);
    cycle = 1;
}

if (!cycle)
    print(A, size); //Выводим матрицу
```

```

FILE* t;

t = fopen("results.txt", "w");

fprintf(t, "%d", size);
fprintf(t, "\n");

for (int i = 0; i < size; i++)
{
    for (int j = 0; j < size; j++)
    {
        fprintf(t, "%d ", A[i][j]);
    }
    fprintf(t, "\n");
}
fclose(t);
break;
case 2:
printf("Введите размер матрицы.\n");

size = scan(); //Проверка ввода

A = scan_key(size);

print(A, size); //Выводим матрицу

floyd(A, size); //Вызываем функцию Флойда-Уоршелла

for (int i = 0; i < size; i++)
    if (A[i][i] < 0)
        r++;
if (r >= 1)
{
    printf("Обнаружено: %d отрицательных циклов\n", r);
}

```

```

        cycle = 1;
    }

    if (!cycle)
        print(A, size); //Выводим матрицу

    for (int i = 0; i < 20; i++)
    {
        if (wway[i] != 0)
            printf("%d ", wway[i]);
    }

    FILE* d;

    d = fopen("results.txt", "w");

    fprintf(d, "%d", size);
    fprintf(d, "\n");

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            fprintf(d, "%d ", A[i][j]);
        }
        fprintf(d, "\n");
    }
    fclose(d);

    break;
case 3:
    printf("\n\t\tКурсовая работа\n\t\tПо дисциплине Логика и
основы алгоритмизации в инженерных задачах\n\t\tНа тему: \"Реализация алгоритма
Флойда-Уоршелла поиска кратчайших путей в графе\"\n\n\tВыполнил студент группы

```

