

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА
по дисциплине «Построение и анализ алгоритмов»
Тема: Алгоритм Флойда-Уоршелла

Студентка гр. 8303

Потураева М.Ю.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студентка Потураева М.Ю.

Группа 8303

Тема работы : алгоритм Флойда-Уоршелла

Исходные данные:

Исследование зависимости времени работы алгоритма Флойда-Уоршелла от входных данных.

Дата сдачи курсовой работы:

Дата защиты курсовой работы:

Студентка

Потураева М.Ю.

Преподаватель

Фирсов М.А.

АННОТАЦИЯ

В данной работе рассмотрены генерация случайного графа по количеству вершин и ребер и исследование алгоритма Флойда-Уоршелла.

Программный код написан на языке программирования C++.

Результат работы программы выводится в файл.

SUMMARY

In this paper, we consider the generation of a random graph by the number of vertices and edges and the study of the Floyd-Warshall algorithm.

The program code is written in the C++programming language.

The result of the program is output to a file.

СОДЕРЖАНИЕ

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ	2
АННОТАЦИЯ	3
ВВЕДЕНИЕ	5
ЦЕЛЬ РАБОТЫ	6
ПОСТАНОВКА ЗАДАЧИ	6
СПЕЦИФИКАЦИЯ ПРОГРАММЫ	6
ОПИСАНИЕ АЛГОРИТМА	7
ОПИСАНИЕ ОСНОВНЫХ ФУНКЦИЙ И СТРУКТУР ДАННЫХ	8
ТЕСТИРОВАНИЕ	9
ИССЛЕДОВАНИЕ АЛГОРИТМА	11
ЗАКЛЮЧЕНИЕ	15
ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ.	17

ВВЕДЕНИЕ

Алгоритм Флойда — Уоршелла — динамический алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного ориентированного графа.

Пусть вершины графа $G=(V,E)$, $|V|=n$ пронумерованы от 1 до n и введено обозначение d_{ij}^k для длины кратчайшего пути от i до j , который кроме самих вершин i,j проходит только через вершины $1 \dots k$. Очевидно, что d_{ij}^0 — длина (вес) ребра (i,j) , если таковое существует (в противном случае его длина может быть обозначена как ∞).

Существует два варианта значения d_{ij}^k , $k \in (1 \dots n)$:

1. Кратчайший путь между i,j не проходит через вершину k , тогда $d_{ij}^k = d_{ij}^{k-1}$
2. Существует более короткий путь между i,j , проходящий через k , тогда он сначала идёт от i до k , а потом от k до j . В этом случае, очевидно, $d_{ij}^k = d_{ik}^{k-1} + d_{kj}^{k-1}$

Таким образом, для нахождения значения функции достаточно выбрать минимум из двух обозначенных значений.

Тогда рекуррентная формула для d_{ij}^k имеет вид:

d_{ij}^0 — длина ребра (i,j)

$$d_{ij}^k = \min (d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$$

ЦЕЛЬ РАБОТЫ

Написать программу, с помощью которой можно будет генерировать граф по количеству вершин и ребер, а также анализировать время работы алгоритма в зависимости от входных данных.

ПОСТАНОВКА ЗАДАЧИ

Исследование алгоритма Флойда-Уоршелла на большом количестве входных данных.

СПЕЦИФИКАЦИЯ ПРОГРАММЫ

Программа написана на языке C++. Считывание происходит из терминала. Пользователь вводит количество вершин и ребер, далее по ним строится граф. Результат работы программы помещается в файл.

ОПИСАНИЕ АЛГОРИТМА

На вход алгоритму подается количество вершин N и количество ребер $edges$, которое может быть в диапазоне $[1; N*(N-1)]$. Далее происходит заполнение матрицы смежности -1 , т.к. граф еще не заполнен. Формируется массив случайно сгенерированных чисел для дальнейшей записи их как вес ребер. В цикле $edges$ раз генерируем ребро (кроме диагональных и уже созданных) и берем значение его веса из массива случайных чисел.

Далее алгоритм Флойда-Уоршелла обрабатывает матрицу смежности, и на выходе получаем матрицу с итоговыми кратчайшими расстояниями в графе.

Чтобы рассчитать время работы программы используется функция `clock()`, перед работой алгоритма и после.

Сложность алгоритма по времени: три вложенных цикла содержат операцию, исполняемую за константное время $O(1)$, то есть алгоритм имеет кубическую сложность $O(n*n*n)$, где n -количество вершин.

Сложность алгоритма по памяти: так как в структуре графа хранится только двумерный массив, хранящий информацию о ребрах, то сложность $O(n*n)$, где n -количество вершин.

ОПИСАНИЕ ОСНОВНЫХ ФУНКЦИЙ И СТРУКТУР ДАННЫХ

```
class Graph {  
private:  
    int** matrix;  
public:  
    int N;  
    int edges;  
};
```

Структура для хранения графа, matrix-матрица смежности, N-количество вершин, edges-количество ребер.

```
void FloydWarshall()
```

Функция, реализующая алгоритм Флойда-Уоршелла.

```
Graph()
```

Конструктор графа, в котором происходит считывание данных из терминала и генерация графа.

```
void Print(bool flag)
```

Функция для вывода графа в файл, в зависимости от флага выбирается файл, в который будет записана матрица.

ТЕСТИРОВАНИЕ

№	Input	Output
1	0 50 19 5 0 20 41 6 0	0 25 19 5 0 20 11 6 0
2	0 47 24 -1 0 -1 26 48 0	0 47 24 -1 0 -1 26 48 0
3	1 2 3 4 5 2 3 4 4 2 3 4 4 3 4 2 3 4 2 3 3 2 3 4 3	0 2 3 4 4 2 0 4 4 2 3 4 0 3 4 2 3 4 0 3 3 2 3 4 3
4	0 43 10 31 7 -1 0 43 32 39 9 48 0 50 17 48 44 32 0 2 35 25 24 13 0	0 32 10 20 7 -1 0 43 32 34 9 41 0 29 16 35 27 26 0 2 33 25 24 13 0
5	0 42 2 25 11 25 28 29 0 35 48 42 48 -1 37 18 0 43 29 33 20 10 36 21 0 50 45 -1 1 10 42 3 0 39 42 31 27 15 36 13 0 1 2 23 36 12 10 49 0	0 20 2 14 11 25 22 29 0 31 43 40 48 -1 22 18 0 32 29 33 20 10 30 12 0 21 35 -1 1 10 3 3 0 26 23 3 21 5 13 11 0 1 2 20 4 12 10 27 0

```

Консоль отладки Microsoft Visual Studio
Enter the number of vertex in the range [1;10000]:
10
Enter the number of edges :
5
Runtime of program: 0.02
The result of the program is in the file after.txt

C:\Users\potur\source\repos\curs\curs\Debug\curs.exe (процесс 6852) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрывать консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...
  
```

Рисунок 1. Построение графа с 10 вершинами и 5 ребрами

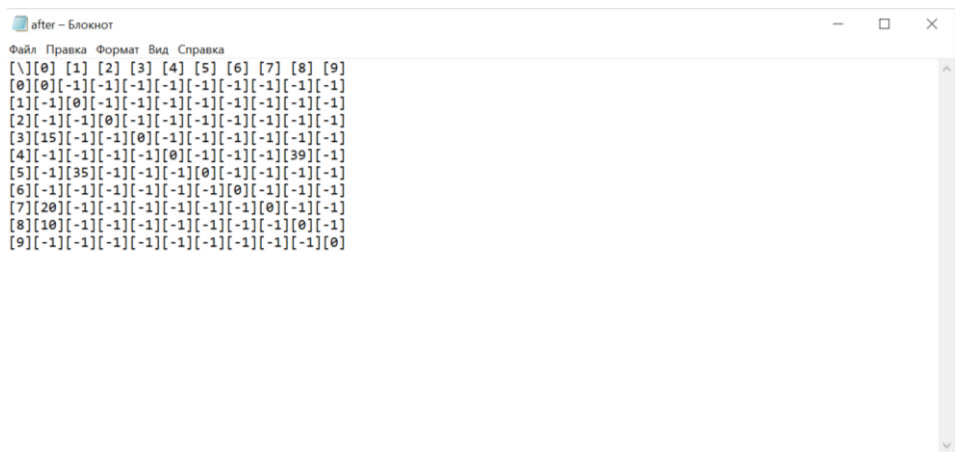


Рисунок 2. Построение графа с 10 вершинами и 5 ребрами

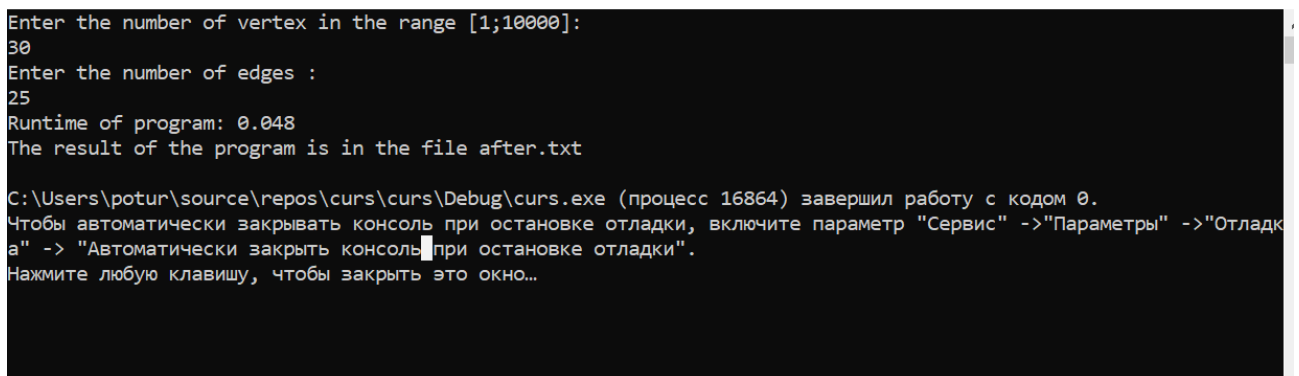


Рисунок 3. Построение графа с 30 вершинами и 25 ребрами

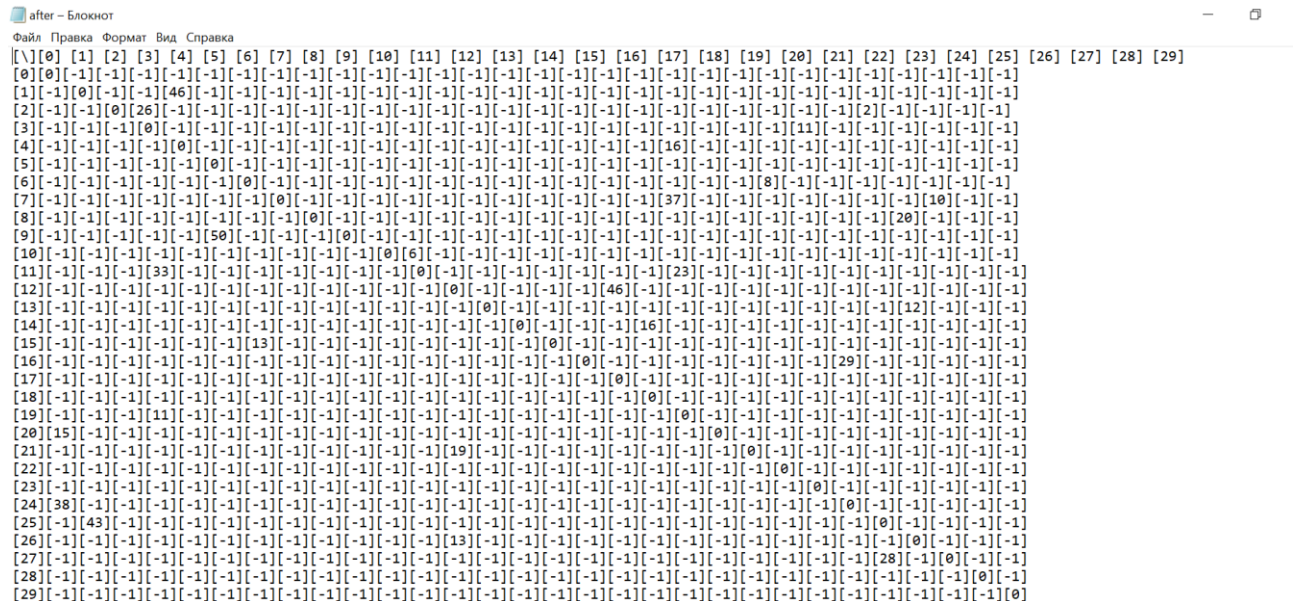


Рисунок 4. Построение графа с 30 вершинами и 25 ребрами

```
Консоль отладки Microsoft Visual Studio
Enter the number of vertex in the range [1;1000]:
100
Enter the number of edges :
20
Runtime of program: 0.103
The result of the program is in the file after.txt

C:\Users\potur\source\repos\curs\curs\Debug\curs.exe (процесс 20708) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...
```

Рисунок 5. Построение графа с 100 вершинами и 20 ребрами

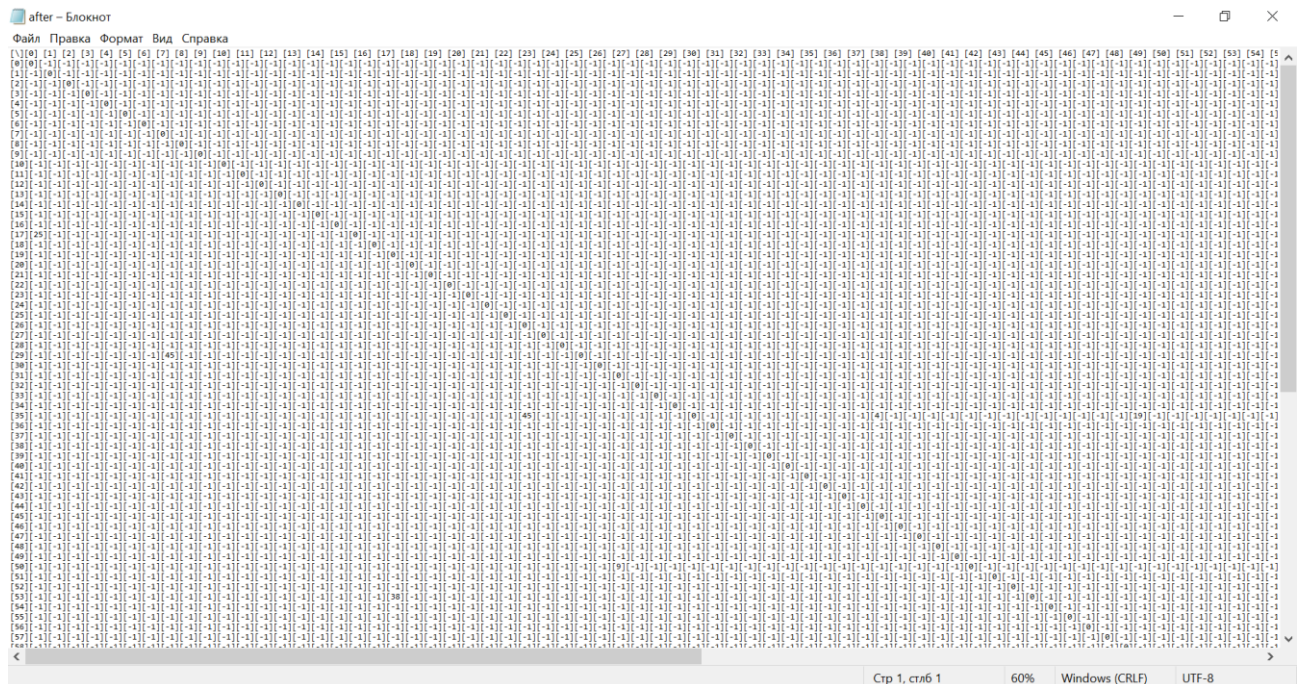


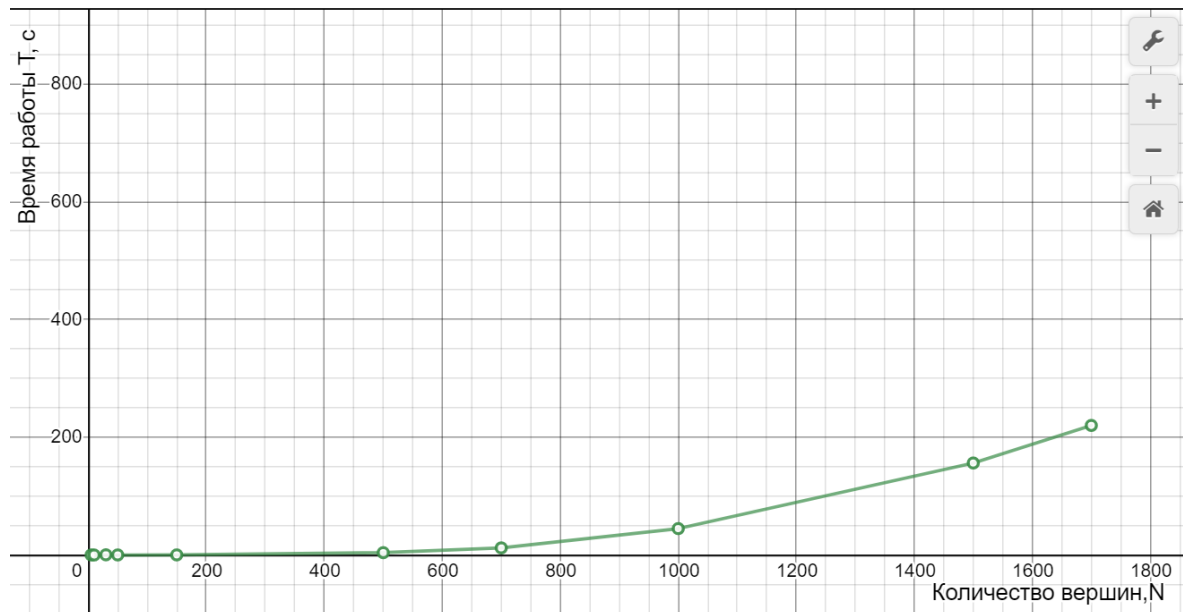
Рисунок 6. Построение графа с 100 вершинами и 20 ребрами

ИССЛЕДОВАНИЕ АЛГОРИТМА

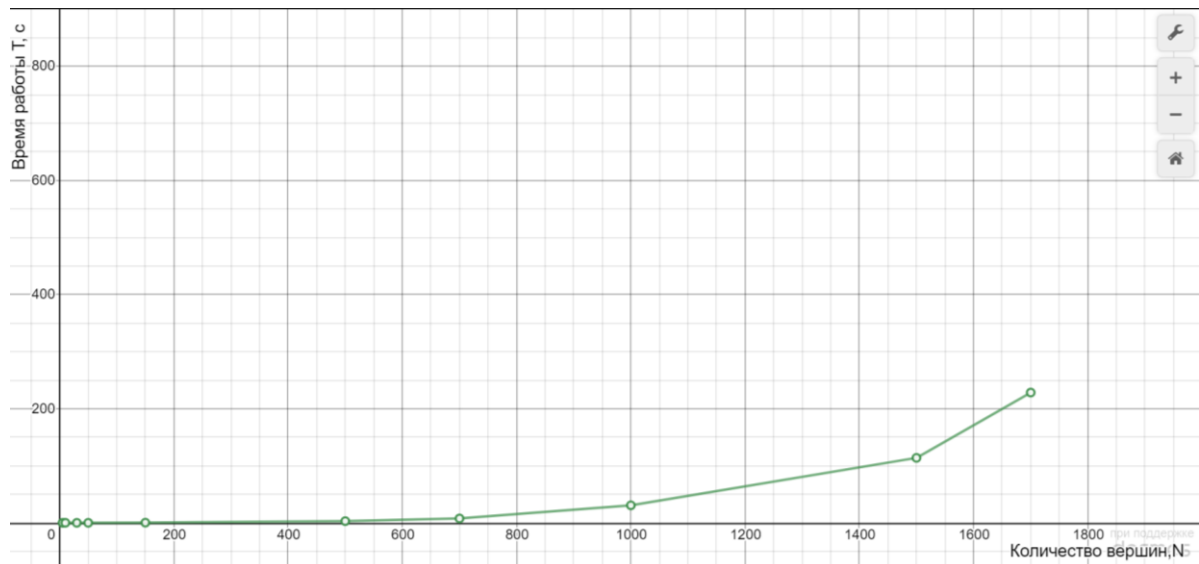
Проведем исследование времени работы алгоритма при разной плотности графа, которая является величиной, значение которой равно отношению числа ребер в анализируемом графе к числу ребер в полном графе с тем же количеством вершин.

Ниже изображена зависимость времени от всех тестовых данных. График функции построен по набору данных [5,10,30,50,150,500,700,1000,2000,3000] при плотности графа 100%. Далее во всех экспериментах плотность графа определяется как отношения количества ребер в сгенерированном графе к количеству ребер в полном графе с таким же количеством вершин.

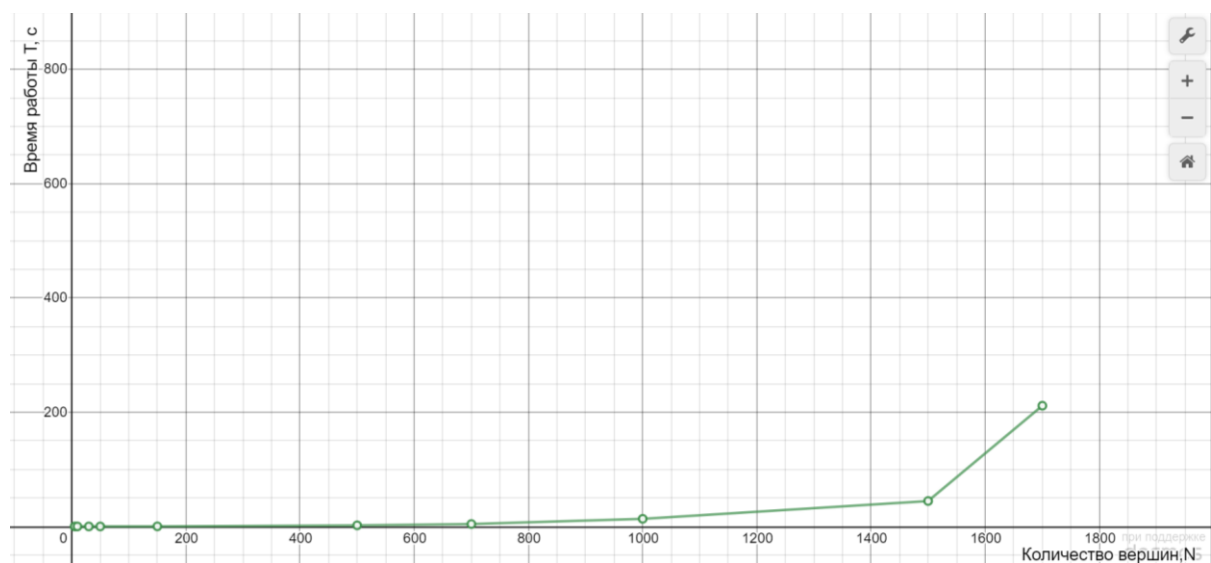
Следовательно, в данном случае количество ребер в графе максимально. Время работы алгоритма практически во всех случаях увеличивается пропорционально количеству вершин в графе, т.к. граф обрабатывает в тройном цикле каждую вершину и ищет кратчайший путь. Возрастание функции схоже с экспоненциальным ростом. Результат работы для каждого количества вершин получился в диапазоне [0.158;220.066].



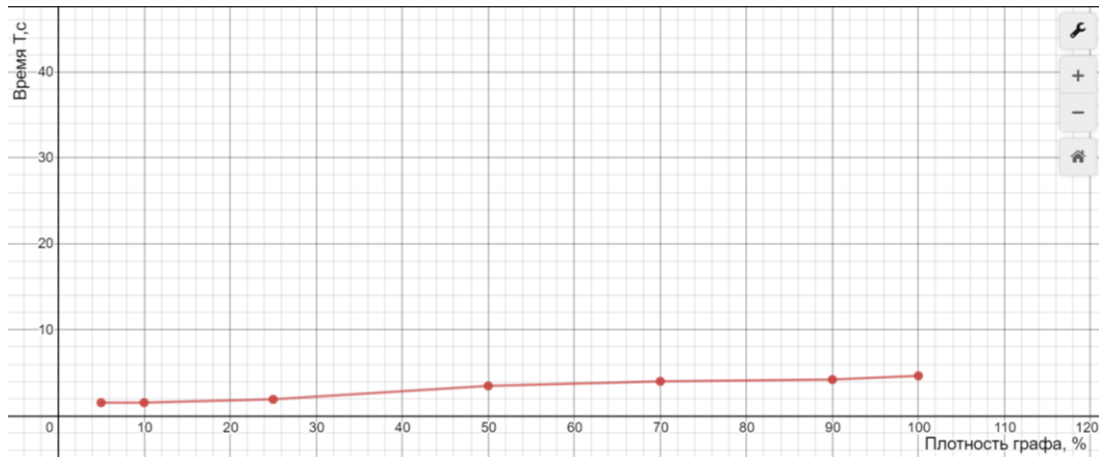
Далее проверим алгоритм на том же наборе данных, но уменьшим плотность графа до 50%. В этом случае количество ребер в сгенерированном графе равно половине от максимально возможного ($E=N*N/2$, где N -количество вершин графа). При уменьшении ребер в графе, соответственно и уменьшается количество обрабатываемых ячеек в матрице и пересчета их расстояний. Следовательно, в большинстве случаев уменьшается и время работы программы, которое напрямую зависит от их количества. Результат работы для каждого количества вершин получился в диапазоне [0.047;228.268].



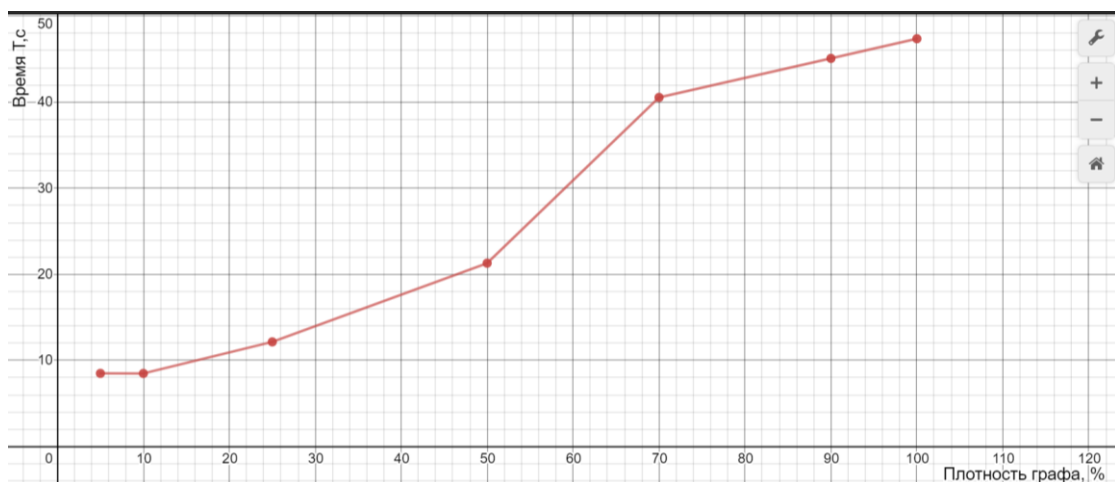
И последнее исследование проведем с плотностью графа 20%, следовательно, $E = N * N / 5$, где N-количество вершин графа. В данном случае время работы программы сильно отличается от значений первого исследования. Результат работы для каждого количества вершин получился в диапазоне [0.057;211.323]. Так как время работы алгоритма при одинаковом количестве вершин в случае с 100% насыщенностью меньше, чем время работы алгоритма с 50% насыщенностью, можно сделать вывод, что время работы алгоритма Флойда-Уоршелла, обратно пропорционально зависит от количества вершин и ребер в графе, следовательно, от его насыщенности.



Также проведем исследования зависимости времени от плотности графа при фиксированном количестве вершин. Ниже изображена зависимость времени от всех тестовых данных. График функции построен по набору данных [5,10,25,50,70,90,100] при количестве вершин равном 500. В данном случае время работы также увеличивается с увеличением плотности графа. По данному графику нельзя точно сказать, какой является данная функция, степенной или же экспоненциальной, как это было в прошлых исследованиях.



Далее проверим алгоритм на том же наборе данных, но увеличим количество вершин до 1000. При увеличении вершин в графе, соответственно и увеличивается количество обрабатываемых ячеек в матрице и пересчета их расстояний. Следовательно, в большинстве случаев увеличивается и время работы программы, которое напрямую зависит от их количества. В данном случае по графику можно сказать, что функция близка к степенной.



ЗАКЛЮЧЕНИЕ

В ходе данной курсовой работы была написана программа для генерации случайного графа и исследования алгоритма Флойда-Уоршелла, также было выяснено, что время работы алгоритма, обратно пропорционально зависит от насыщенности исследуемого графа.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://ru.wikipedia.org/wiki/Алгоритм_Флойда_—_Уоршелла
2. https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Флойда
3. <https://habr.com/ru/post/105825/>

ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <stdlib.h>
#include <time.h>
using namespace std;
class Graph
{
private:
    int** matrix;
public:
    int N;
    int edges;

    ~Graph() {
        for (int i = 0; i < N; ++i)
            delete[] matrix[i];
        delete[] matrix;
    }

    void FloydWarshall() {
        int i, j, k;
        for (i = 0; i < N; i++)
            if (matrix[i][i] != -1)
                matrix[i][i] = 0;

        for (k = 0; k < N; k++) {
            for (i = 0; i < N; i++) {
                for (j = 0; j < N; j++) {
                    if (matrix[i][k] <= 0 || matrix[k][j] <= 0 || matrix[i][j] == -1)
                        continue;
                    if ((matrix[i][k] + matrix[k][j] < matrix[i][j] || matrix[i][j] == 0) &&
                        (i != j)) {
                        matrix[i][j] = matrix[i][k] + matrix[k][j];
                    }
                }
            }
        }
    }

    Graph() {
        cout << "Enter the number of vertex in the range [1;10000]:" << endl;
        cin >> N;
        cout << "Enter the number of edges : " << endl;
        cin >> edges;
        while (N > 10000 || N < 0) {
            cout << "Wrong value for vertexes, enter number again:" << endl;
            cin >> N;
        }
        while (edges <= 0 || edges > N * (N - 1)) {
            cout << "Wrong value for edges, enter number again:" << endl;
            cin >> edges;
        }
        int i, j;
        int tmp = 0;
        cout << "If you want to enter the graph manually - press '1', if you want to  
generate a graph - press '2'. " << endl;
        cin >> tmp;
        matrix = new int* [N];
```

```

        for (int i = 0; i < N; i++) // создание каждого одномерного массива в динамическом
двумерном массиве, или иначе - создание столбцов размерность n
            matrix[i] = new int[N];

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            matrix[i][j] = -1;
            if (i == j)
                matrix[i][j] = 0;
        }
    }

    if (tmp == 2) {
        const int R_MIN = -1;
        const int R_MAX = 50;
        int i, j;
        int* randomArray = new int[edges];
        srand(time(NULL));

        for (int i = 0; i < edges; i++) {
            randomArray[i] = rand() % (R_MAX - R_MIN + 1) + R_MIN;
            if (randomArray[i] == 0) {
                i--;
            }
        }

        for (int i = 0; i < edges; i++) {
            int k = rand() % (N);
            int j = rand() % (N);
            if (matrix[k][j] > 0 || k == j) {
                i--;
            }
            else {
                matrix[k][j] = randomArray[i];
            }
        }
    }
    else if (tmp == 1) {
        cout << "Fill in the adjacency matrix:" << endl;
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
                {
                    cout << "matrix[" << i << "][" << j << "] = ";
                    cin >> matrix[i][j];
                }
            }
        }
    }
}

void Print(bool flag) {
    FILE* f;
    if (flag)
        f = fopen("before.txt", "w");
    else
        f = fopen("after.txt", "w");
    fprintf(f, "[\\]");
    for (int i = 0; i < N; ++i)
        fprintf(f, "[%d] ", i);
    fprintf(f, "\\n");
    for (int i = 0; i < N; i++) {
        fprintf(f, "[%d]", i);
    }
}

```

```

        for (int j = 0; j < N; j++) {
            fprintf(f, "[%d]", matrix[i][j]);
        }
        putc('\n', f);
    }
    fclose(f);
}

};

int main()
{
    Graph graph;
    unsigned int tmp1 = clock();
    graph.Print(true);
    graph.FloydWarshall();
    graph.Print(false);
    unsigned int tmp2 = clock();
    cout << "Runtime of program: " << (tmp2 - tmp1) / 1000.0 << endl;
    cout << "The result of the program is in the file after.txt" << endl;
    system("PAUSE");
    return 0;
}

```