

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Российский химико-технологический университет имени Д.И.
Менделеева»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

Выполнил студент группы КС-36: Золотухин А.А.

Ссылка на репозиторий: [https://github.com/
MUCTR-IKT-CPP/
ZolotukhinAA_36_ALG](https://github.com/MUCTR-IKT-CPP/ZolotukhinAA_36_ALG)

Принял: Крашенников Роман Сергеевич

Дата сдачи: 24.03.2025

Москва
2025

Оглавление

Описание задачи	1
Описание метода/модели	2
Выполнение задачи	3
Выводы	14

Описание задачи

1. Создайте взвешенный граф, состоящий из $[10, 20, 50, 100]$ вершин.
 - каждая вершина графа связана со случайным количеством вершин, минимум с $[3, 4, 10, 20]$;
 - веса рёбер задаются случайным значением от 1 до 20;
 - каждая вершина графа должна быть доступна, т.е. до каждой вершины графа должен обязательно существовать путь до каждой вершины, необязательно прямой;
 - (Можно использовать генератор из предыдущей лабораторной работы.)
2. Выведите получившийся граф в виде матрицы смежности.
3. Для каждого графа требуется провести серию из 5-10 тестов, в зависимости от времени, затраченного на выполнение одного теста. Необходимо:
 - найти кратчайшие пути между всеми вершинами графа и их длину с помощью алгоритма Флойда-Уоршелла.
4. В рамках каждого теста необходимо замерить потребовавшееся время на выполнение задания из пункта 3 для каждого набора вершин. По окончании всех тестов построить график, используя полученные замеры времени, где на ось абсцисс (X) нанести N - количество вершин, а на ось ординат (Y) - значения затраченного времени.

Описание метода/модели

Алгоритм Флойда-Уоршелла предназначен для получения кратчайших путей между всеми парами вершин, существующих в графе. Результатом работы этого алгоритма является матрица $N \times N$, где N - количество вершин. Особенностью является то, что в матрицах смежности для взвешенного графа нельзя использовать 0 для указания несуществующего пути, лучше использовать максимальное значение.

Ход алгоритма:

1. Поиск-всех-кратчайших-путей-Флойда(матрица смежности)
2. Инициируем 3 счётчика, 2 счётчика для матрицы и 1 для номера промежуточной вершины
3. Итерируем по счётчику промежуточных вершин до тех пор, пока не кончатся вершины
4. Итерируем по первому счётчику вершин до тех пор, пока не кончатся вершины
5. Итерируем по второму счётчику вершин до тех пор, пока не кончатся вершины
6. Считаем длину пути, используя промежуточную вершину как узловую точку, т.е. от вершины по первому счётчику до узловой плюс от узловой до вершины по второму счётчику
7. Проверяем, будет ли новый путь меньше предыдущего, уже найденного через другую вершину, и записываем новое значение, если оно подходит.

Анализ сложности алгоритма Флойда-Уоршелла:

- **Лучший вариант:** $O(N^3)$;
- **Средний вариант:** $O(N^3)$;
- **Худший вариант:** $O(N^3)$;

Преимущества:

- универсален;
- простота использования;
- эффективность.

Недостатки:

- сложность понимания;
- работает медленно на больших графах.

Выполнение задачи

Алгоритм Флойда-Уоршелла реализован на языке *C++*. Построение графиков проводились с помощью программы *GNUpot*. Построение графов демонстрировались с помощью *Graphviz*.

"main"

```
1 int main() {
2     double executionTimes[Constants::nTests] = {0.0};
3
4     for (int sizeIndex = 0; sizeIndex < Constants::nTests; sizeIndex++) {
5         GraphGenerator generator(Constants::minVertices[sizeIndex], Constants::
6         maxVertices[sizeIndex], Constants::minEdges[sizeIndex], Constants::maxEdges[
7         sizeIndex]);
8
9         double totalTime = 0.0;
10
11        for (int graphIndex = 0; graphIndex < Constants::nGraphs; graphIndex++) {
12            Graph graph = generator.generate(sizeIndex);
13
14            std::cout << "Graph " << graphIndex + 1 << " with " << Constants::
15            minVertices[sizeIndex] << " vertices:" << std::endl;
16            std::cout << "Adjacency matrix:" << std::endl;
17            graph.printAdjacencyMatrix();
18
19            auto start = std::chrono::high_resolution_clock::now();
20            graph.floydWarshall();
21            auto end = std::chrono::high_resolution_clock::now();
22
23            std::chrono::duration<double> elapsed = end - start;
24            totalTime += elapsed.count();
25            std::cout << "Time taken: " << elapsed.count() << " seconds" << std::endl;
26
27            std::stringstream dotFilename;
28            dotFilename << Constants::graphNdot << "size_" << Constants::minVertices[
29            sizeIndex] << "_" << graphIndex + 1 << ".dot";
30            graph.generateDotFile(dotFilename.str());
31
32            std::stringstream pngFilename;
33            pngFilename << Constants::graphNpng << "size_" << Constants::minVertices[
34            sizeIndex] << "_" << graphIndex + 1 << ".png";
35            std::string command = "dot -Tpng " + dotFilename.str() + " -o " +
36            pngFilename.str();
37            system(command.c_str());
38        }
39        executionTimes[sizeIndex] = totalTime / Constants::nGraphs;
40    }
41
42    std::ofstream resultsFile(Constants::floyd_marshall);
43    if (resultsFile.is_open()) {
44        for (int i = 0; i < Constants::nTests; i++)
45            resultsFile << Constants::minVertices[i] << " " << executionTimes[i] << std
46            ::endl;
47        resultsFile.close();
48    } else std::cerr << "Error: Unable to open results file." << std::endl;
49
50    return 0;
51 }
```

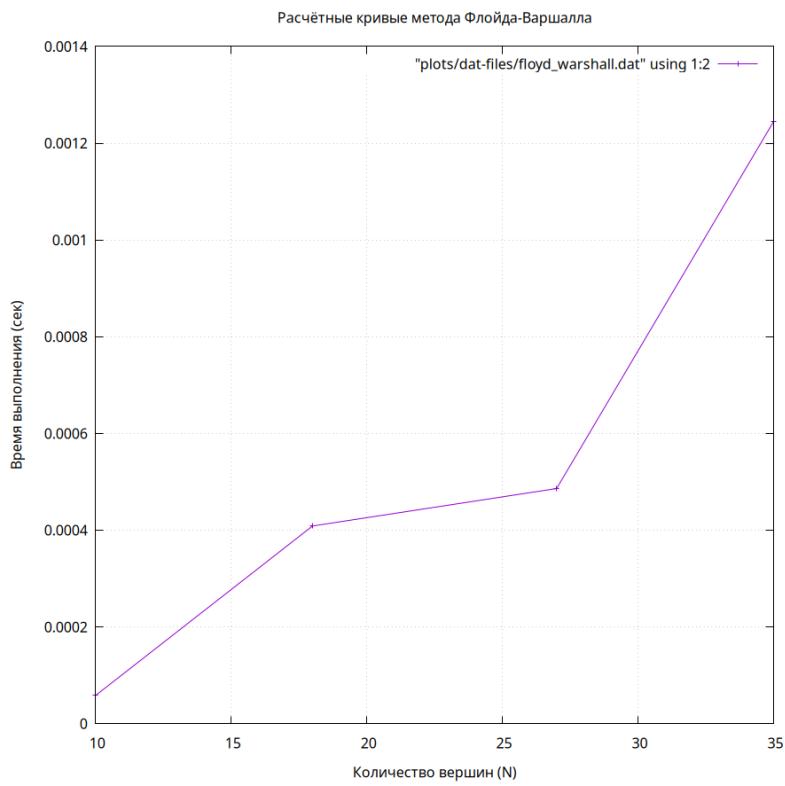



Рис. 1: График.

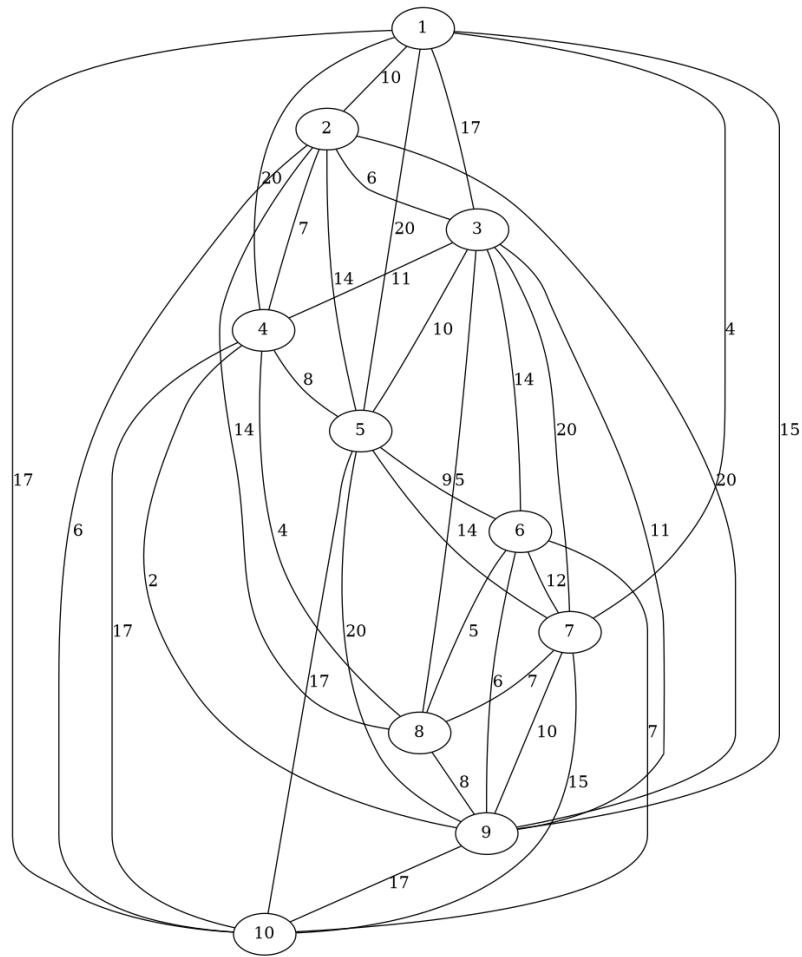


Рис. 2: Граф 1 с 10 вершинами.

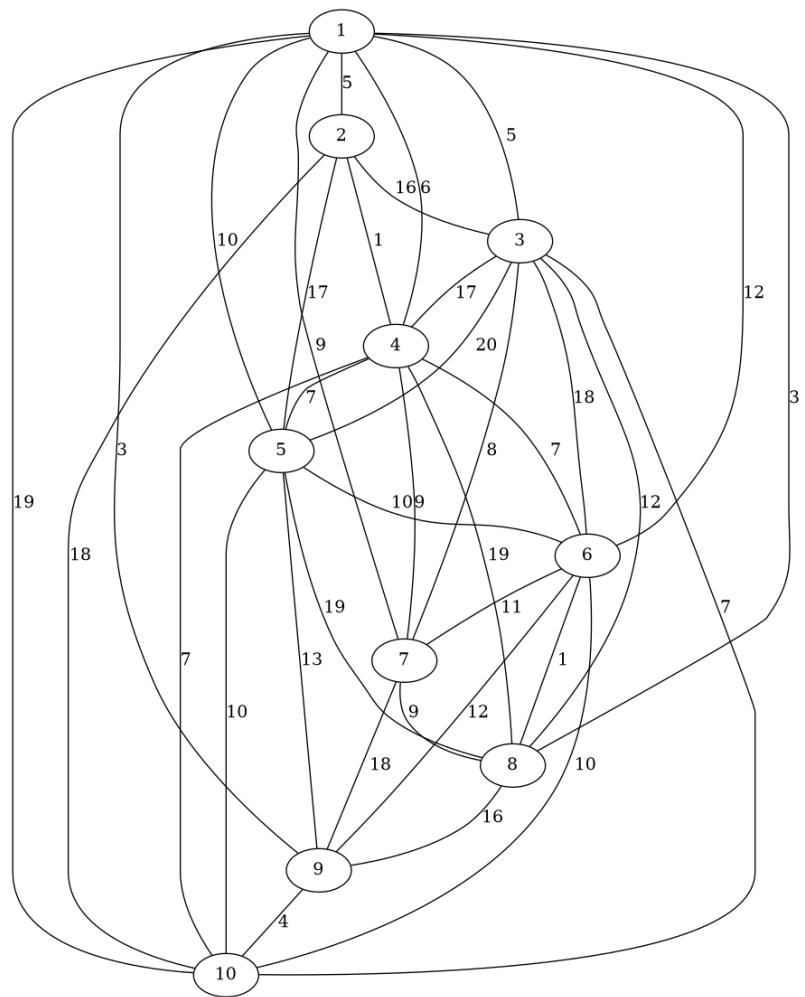


Рис. 3: Граф 2 с 10 вершинами.

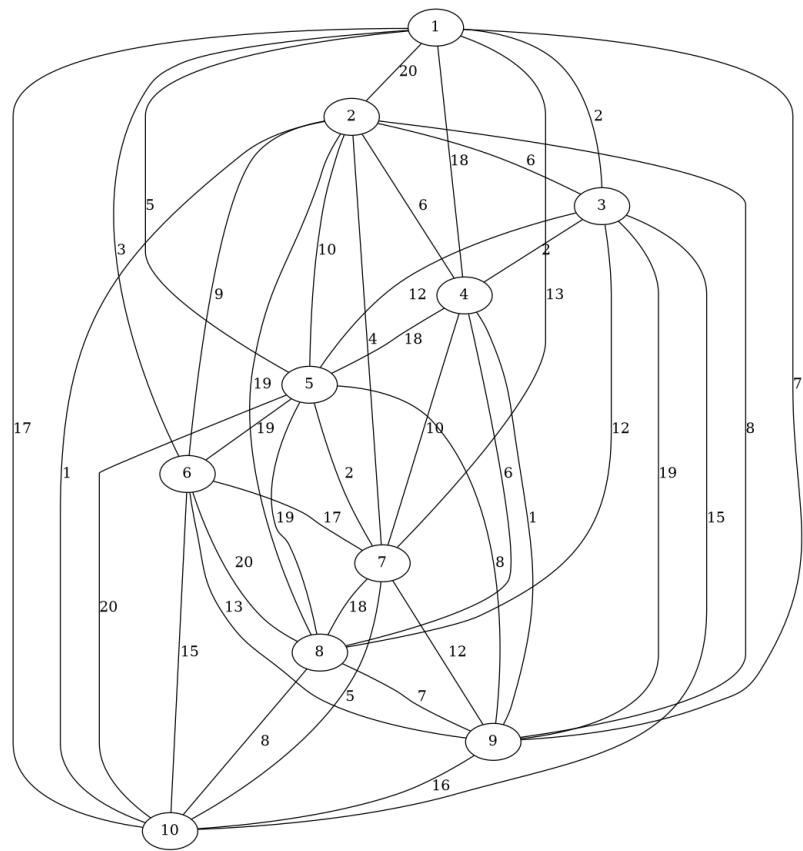


Рис. 4: Граф 3 с 10 вершинами.

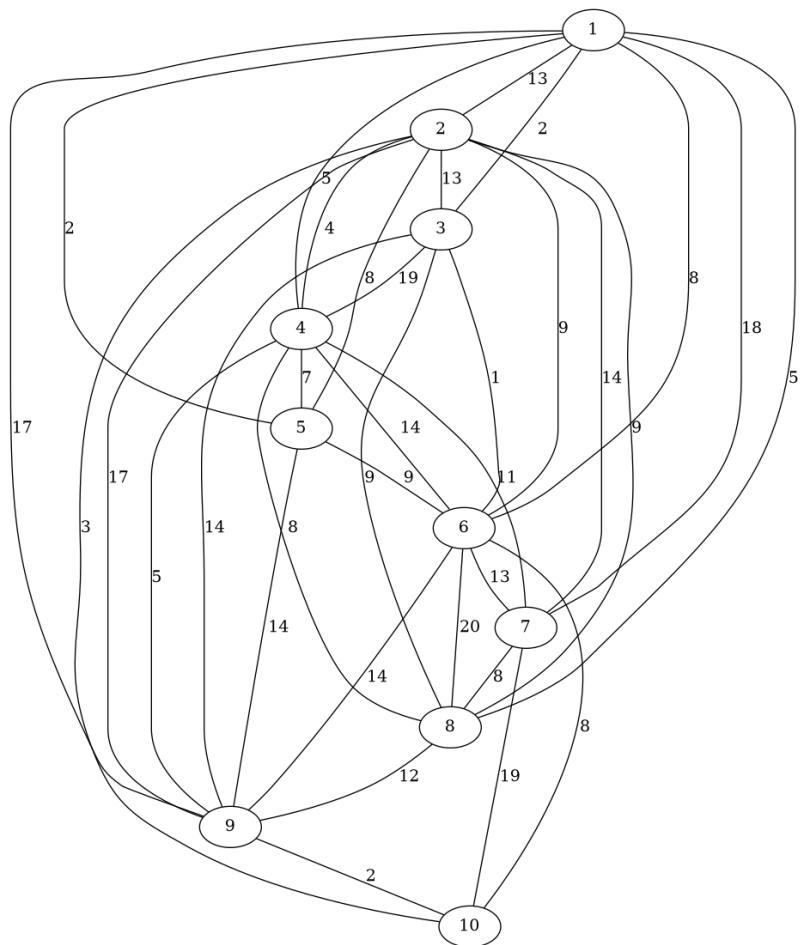


Рис. 5: Граф 4 с 10 вершинами.

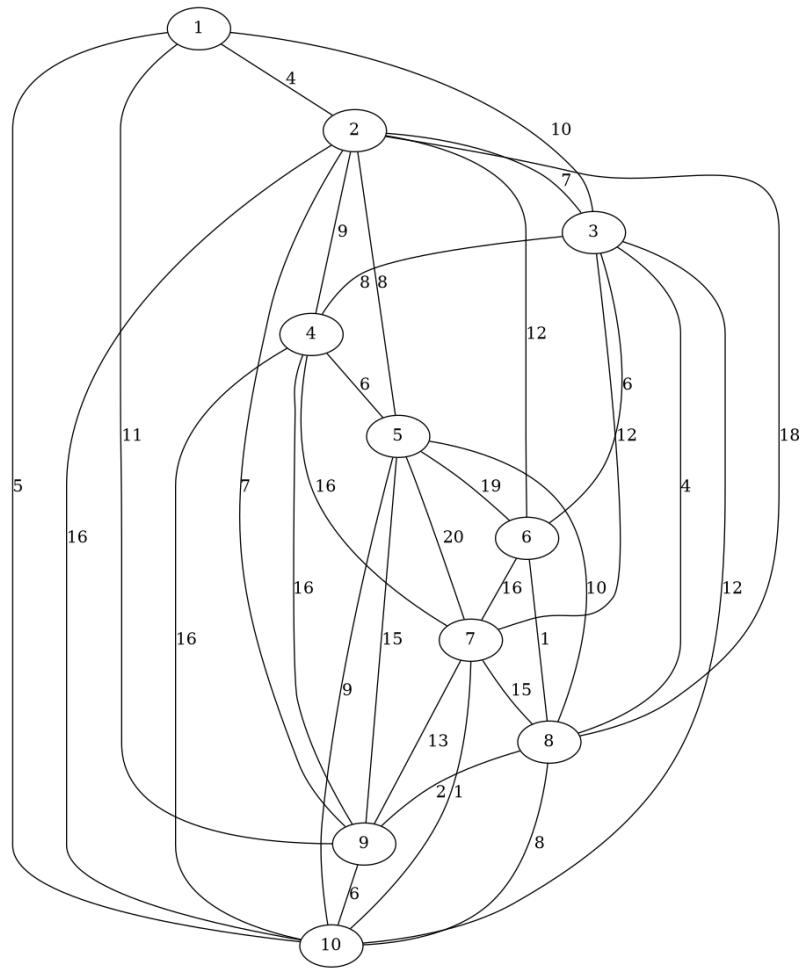


Рис. 6: Граф 5 с 10 вершинами.

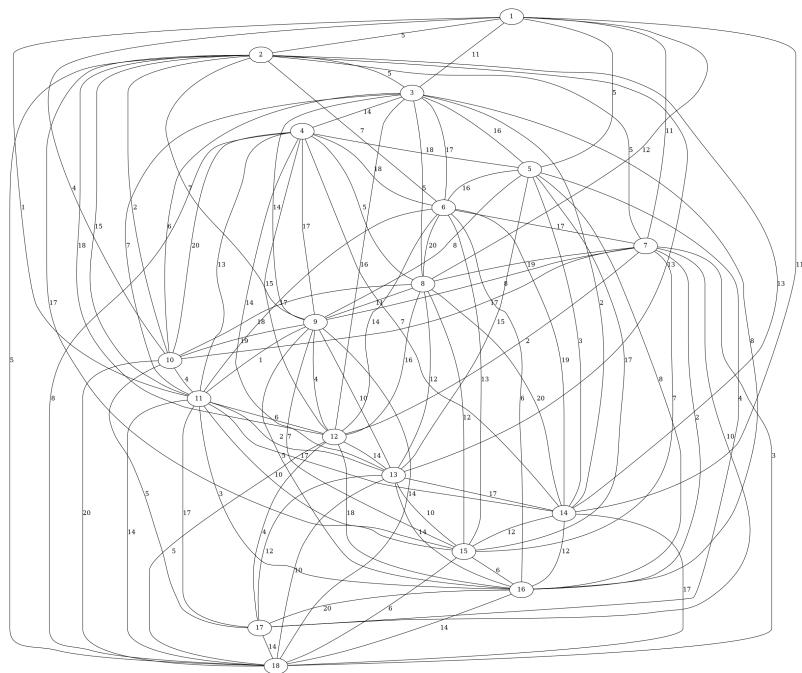


Рис. 7: Граф 1 с 18 вершинами.

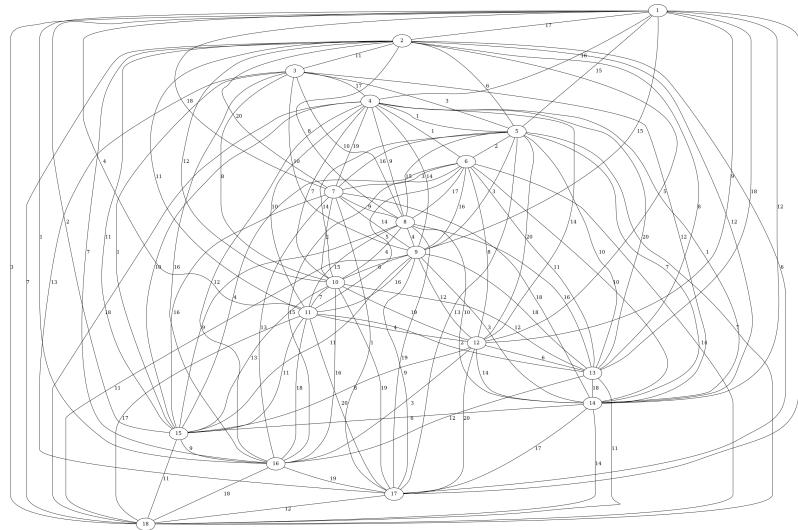


Рис. 8: Граф 2 с 18 вершинами.

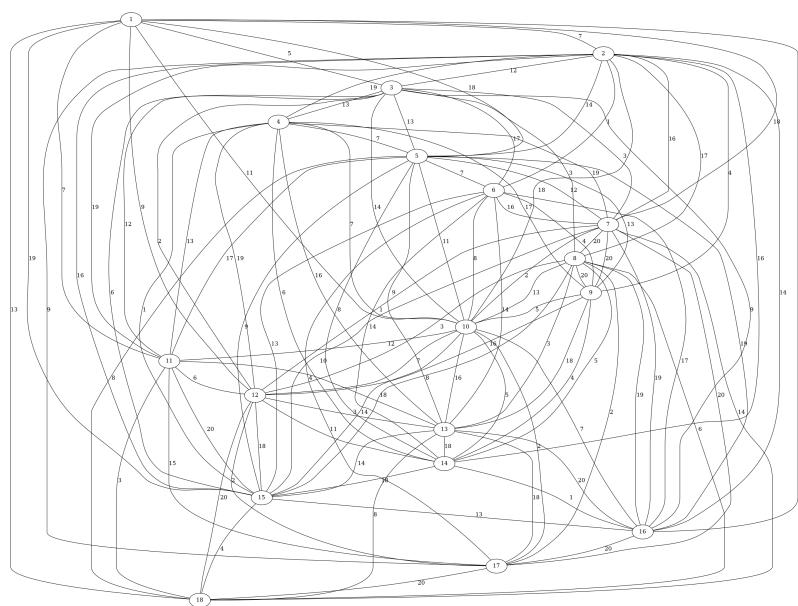


Рис. 9: Граф 3 с 18 вершинами.

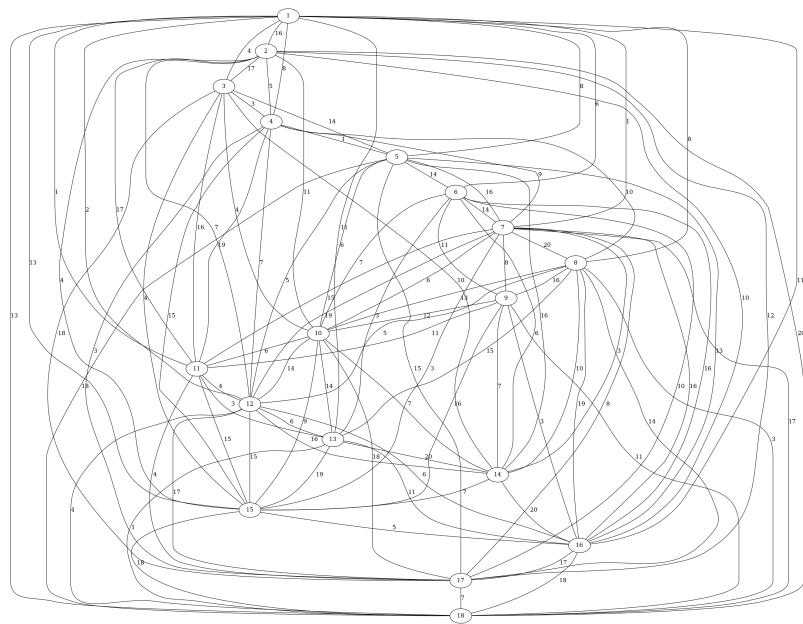


Рис. 10: Граф 4 с 18 вершинами.

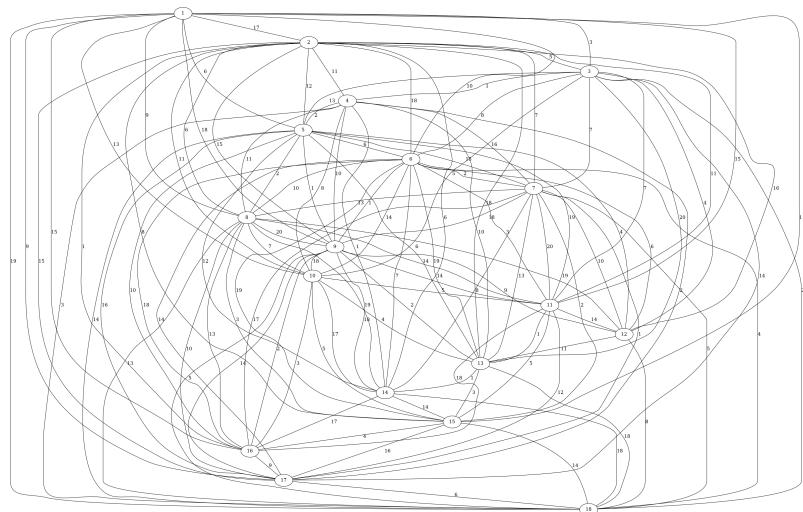


Рис. 11: Граф 5 с 18 вершинами.



Рис. 12: Граф 1 с 27 вершинами.

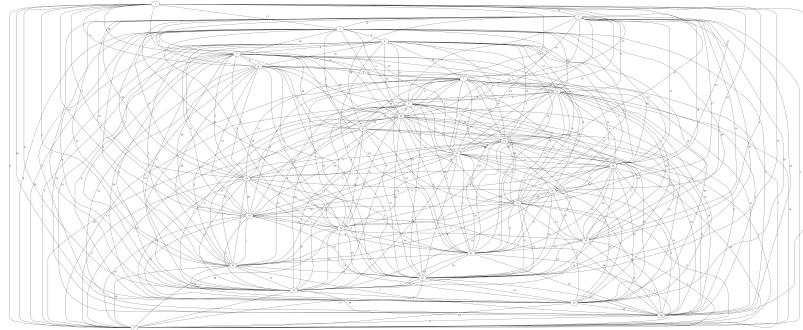


Рис. 13: Граф 2 с 27 вершинами.

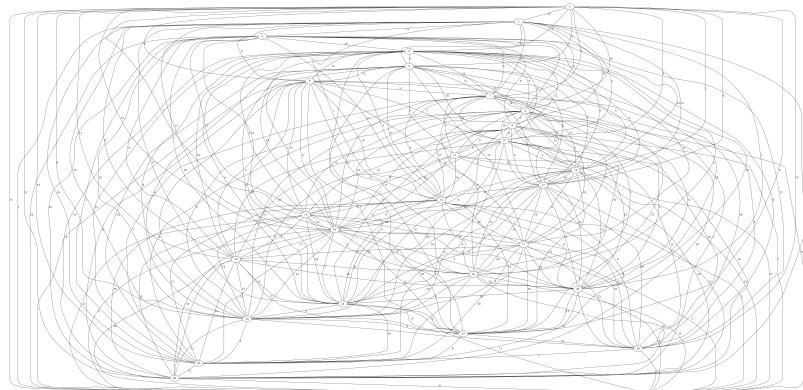


Рис. 14: Граф 3 с 27 вершинами.

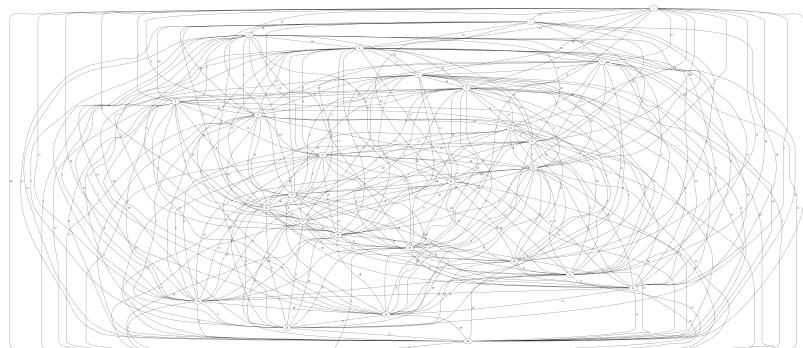


Рис. 15: Граф 4 с 27 вершинами.

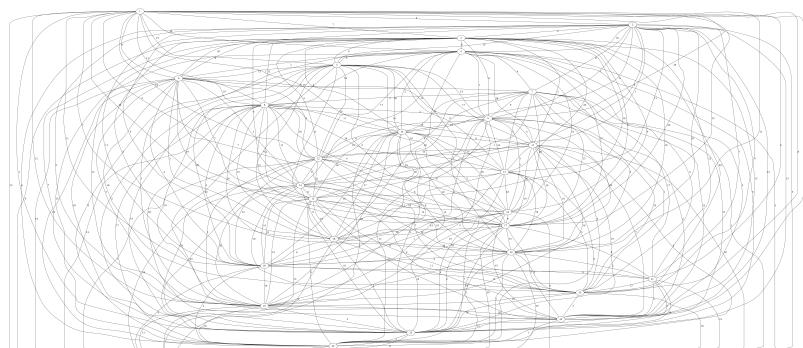


Рис. 16: Граф 5 с 27 вершинами.

Выводы

В ходе лабораторной работы я познакомился с алгоритмом Флойда-Уоршелла, который предназначен для получения кратчайших путей между всеми парами вершин, существующих в графе.

Из графика оказалось видно, что время выполнения увеличивается согласно асимптотической сложности алгоритма, т.е. N^3 с увеличением количества вершин в графе.

Поэтому он довольно эффективен для графов с небольшим количеством вершин.