Давиденко Алексей Ильич, БПИ214

КДЗ №3 по курсу “Построение и анализ алгоритмов”

Отчёт

Состав файлов

Все алгоритмы поиска и замеры времени алгоритмов приведены в единственном main.cpp. Архив содержит данный отчёт, 3 папки, в каждой из которой 21 текстовый файл с исходными графиками, 4 csv файла, main.py для генерации графов, main.cpp для подсчёта времени и .ipynb файл для генерации графиков. Все файлы, в том числе и этот отчёт находятся в репозитории.
https://github.com/davalex2003/construction-and-analysis-of-algorithms/tree/main/kdz_3

Ход работы

Сначала я написал алгоритм на питоне, который генерирует 21 граф по 3 вида каждого (полные, с коэффициентом плотности 0,5 и разреженные) . Получаются txt файлы, в которых на первой строке записано количество ребер, а последующие строки имеют вид первая вершина, вторая вершина, вес (рандомное число от 1 до 10). Затем на C++ я написал 4 алгоритма поиска кратчайшего пути (3 по заданию и 1 дополнительный, который будет описан ниже) и замерил время работы для всех графов. После этого, с помощью Jupyter Notebook я построил графики по полученным результатам.

Оптимизированный алгоритм Беллмана-Форда

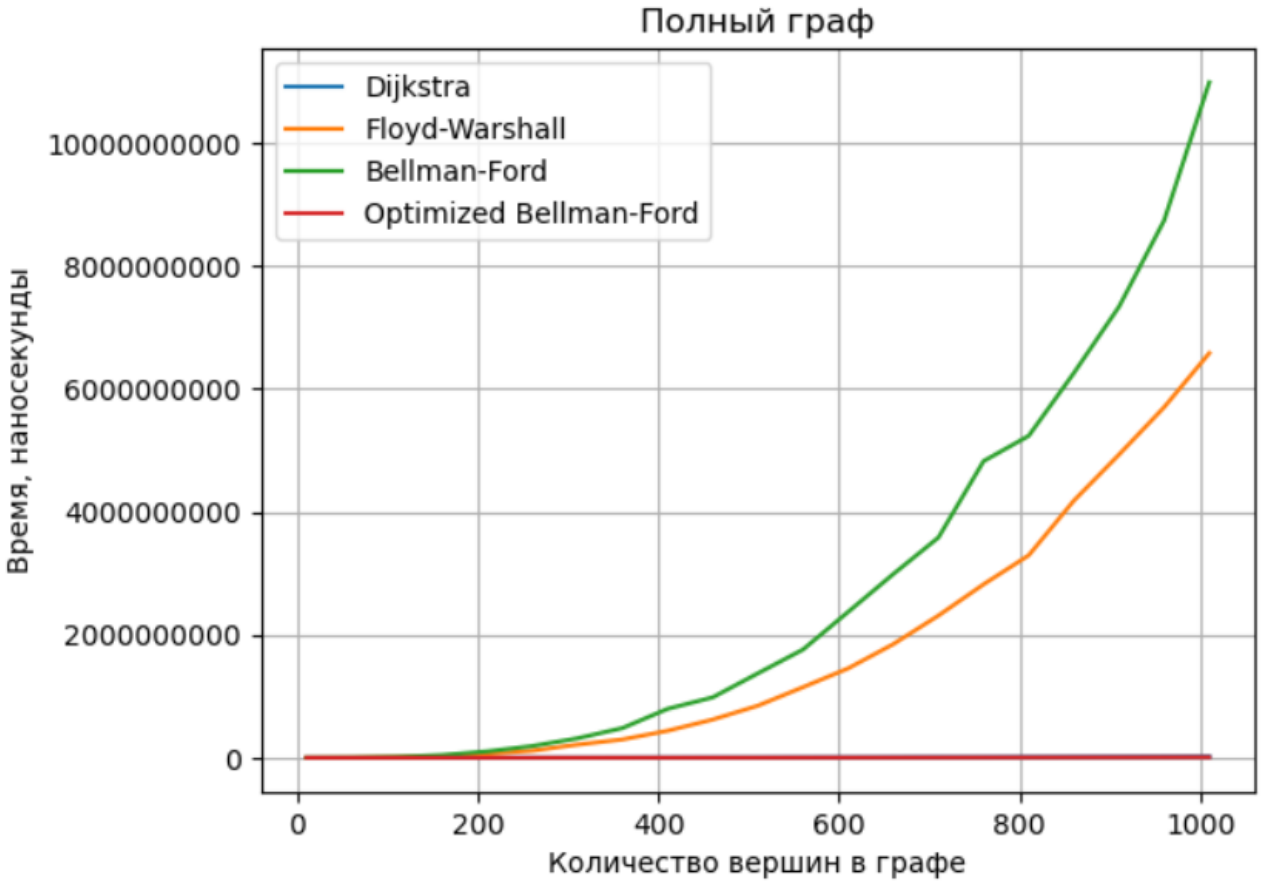
Алгоритм улучшает обычный алгоритм Беллмана-Форда благодаря булевой переменной, которая указывает были ли улучшены пути между вершинами и если нет, то алгоритм завершается. Ниже будет видно, что алгоритм действительно работает быстрее стандартного алгоритма Беллмана-Форда.

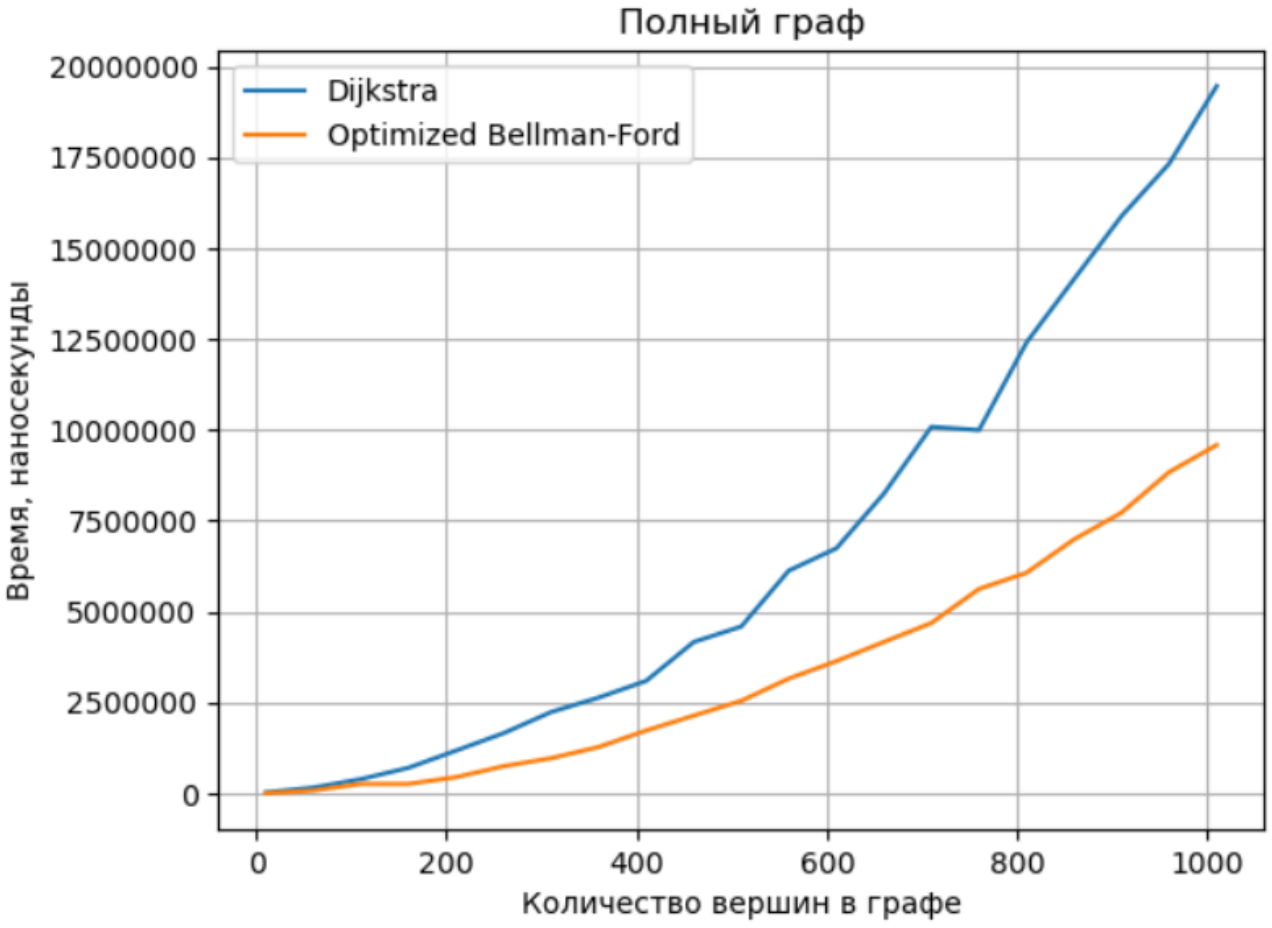
Полученные графики

Результаты замеров времени выводились в csv файлы. Графики строились с помощью Jupyter Notebook. Достаточно было для всех файлов считать значения и построить графики. Далее будут описаны результаты.

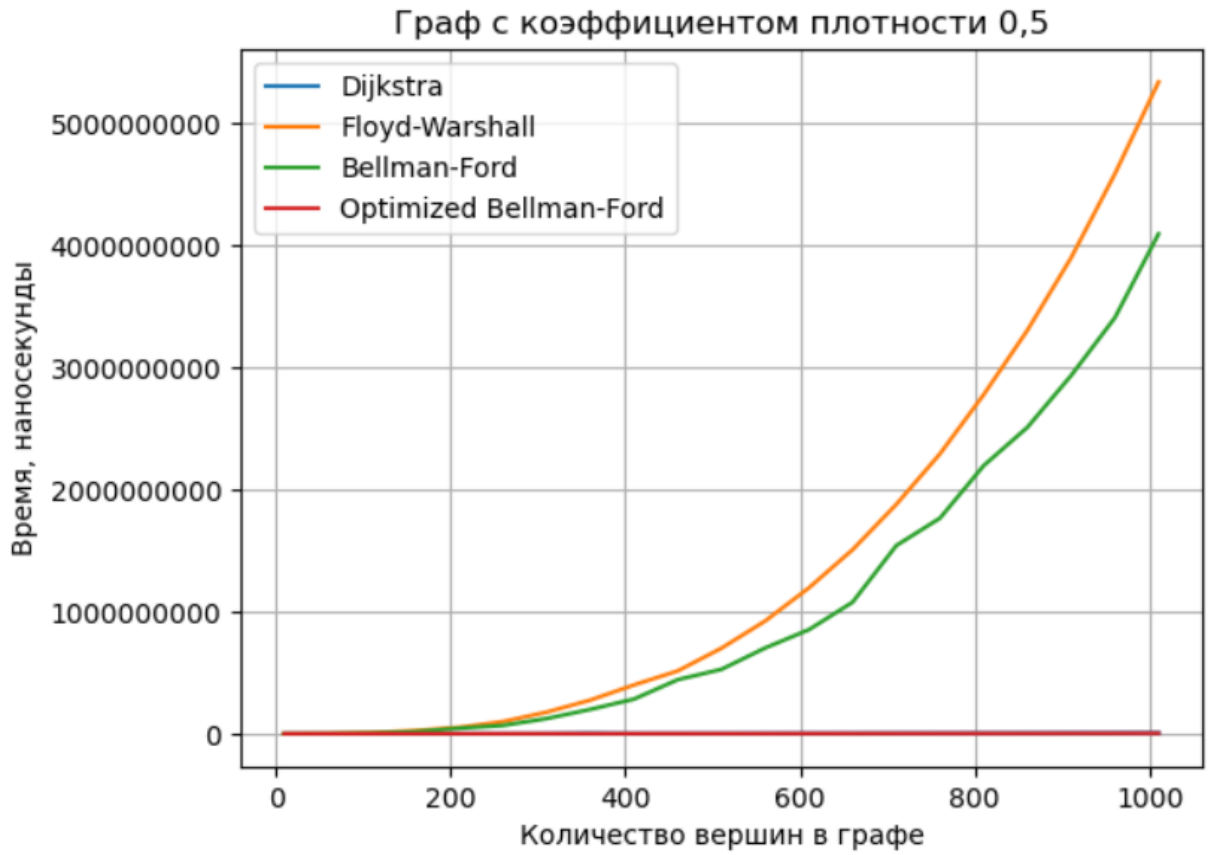
Вывод

Так как некоторые алгоритмы сильно отличались, я разделял некоторые графики, чтобы лучше было видно отличие.

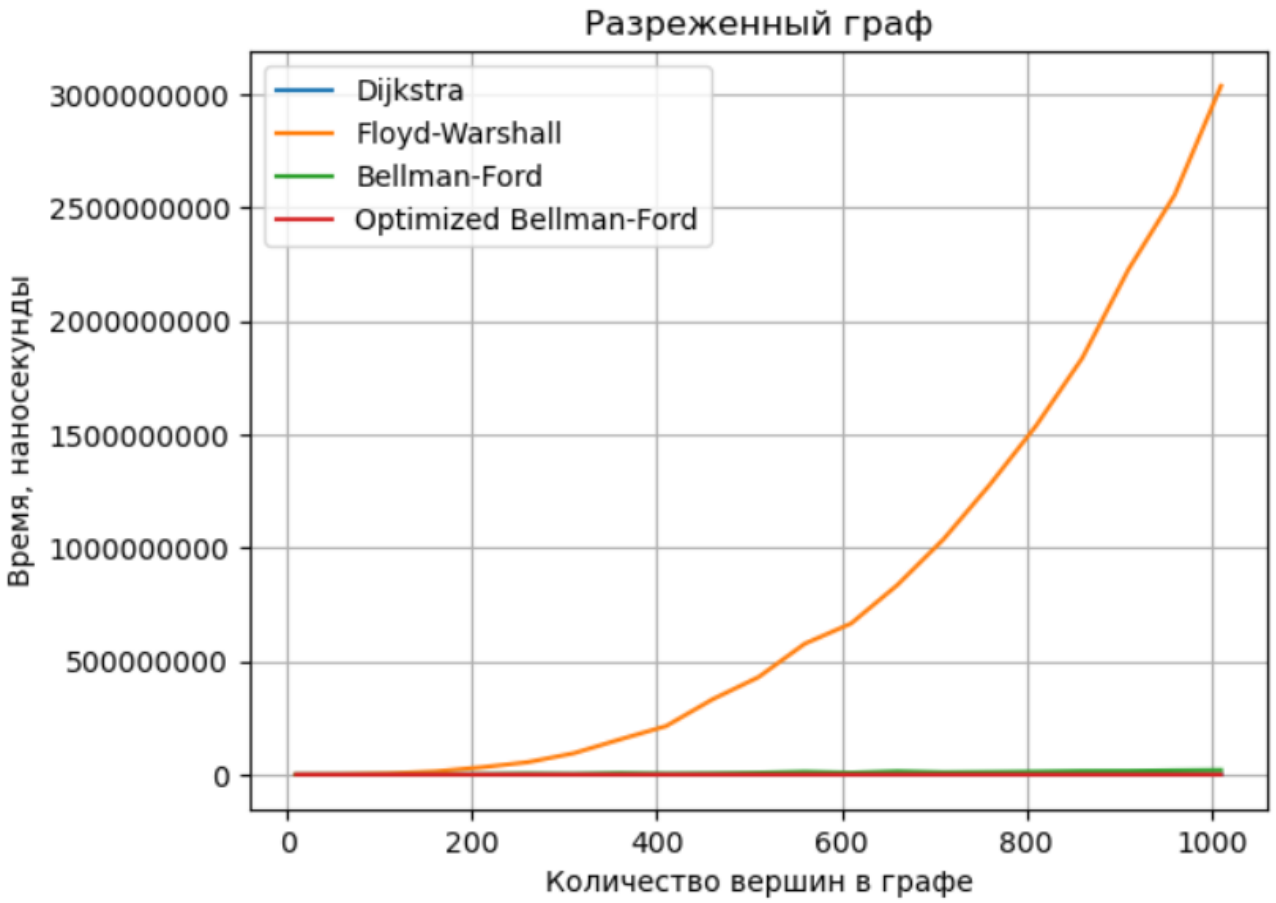

Полный граф
Dijkstra
Floyd-Warshall
Bellman-Ford
Optimized Bellman-Ford
10000000000
8000000000
6000000000
4000000000
2000000000
0
Время, наносекунды
0
200
400
600
800
1000
Количество вершин в графе

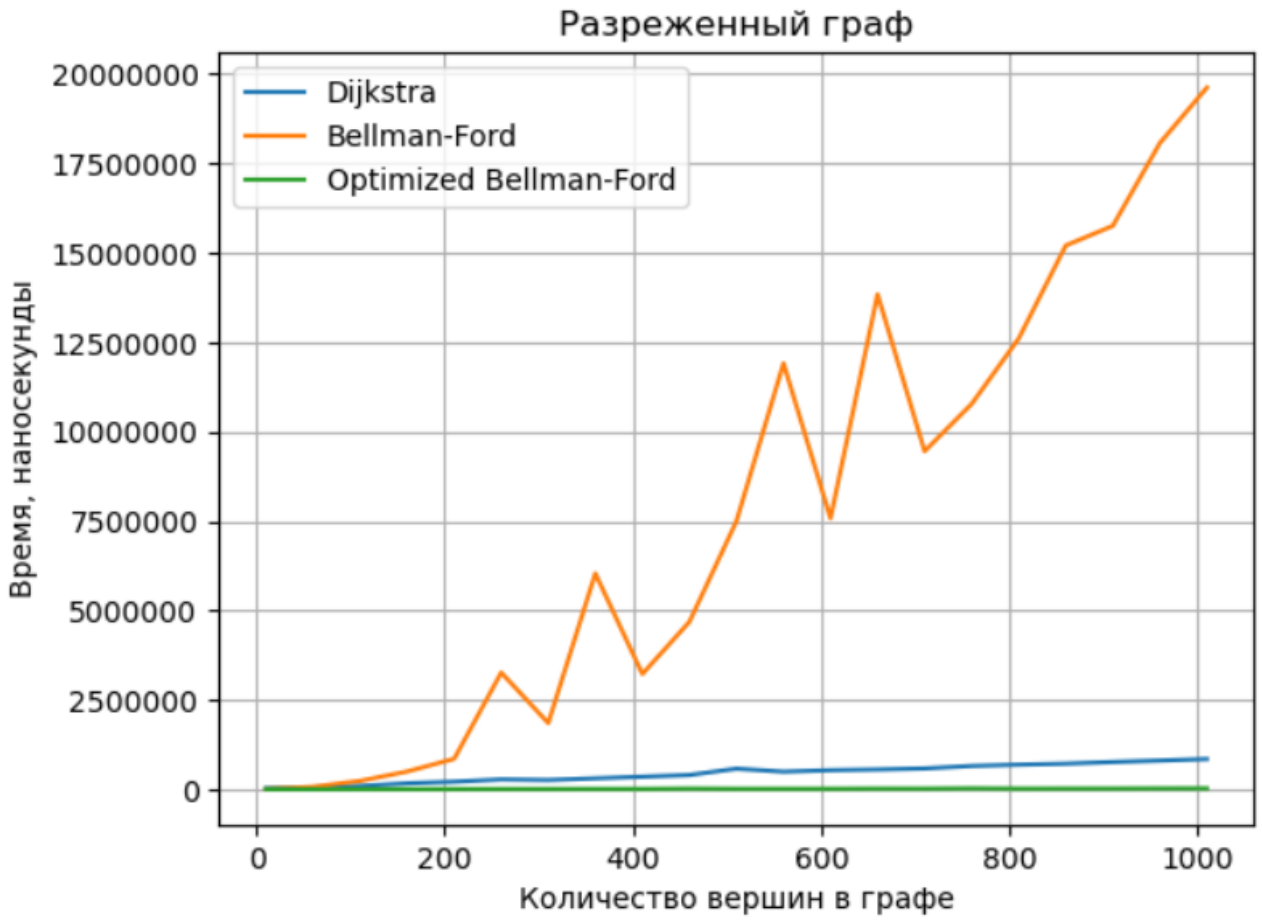

Полный граф
Dijkstra
Optimized Bellman-Ford
20000000
17500000
15000000
12500000
10000000
7500000
5000000
2500000
0
Время, наносекунды
0
200
400
600
800
1000
Количество вершин в графе

Начнём с полных графов. На первом графике сразу видно отличие между проигрывающими алгоритмами Беллмана-Форда и Флойда-Уоршелла, причём не оптимизированный Беллман-Форд показывает худшее время. Лучшее же время показывают алгоритм Дейкстры и оптимизированный Беллман-Форд.

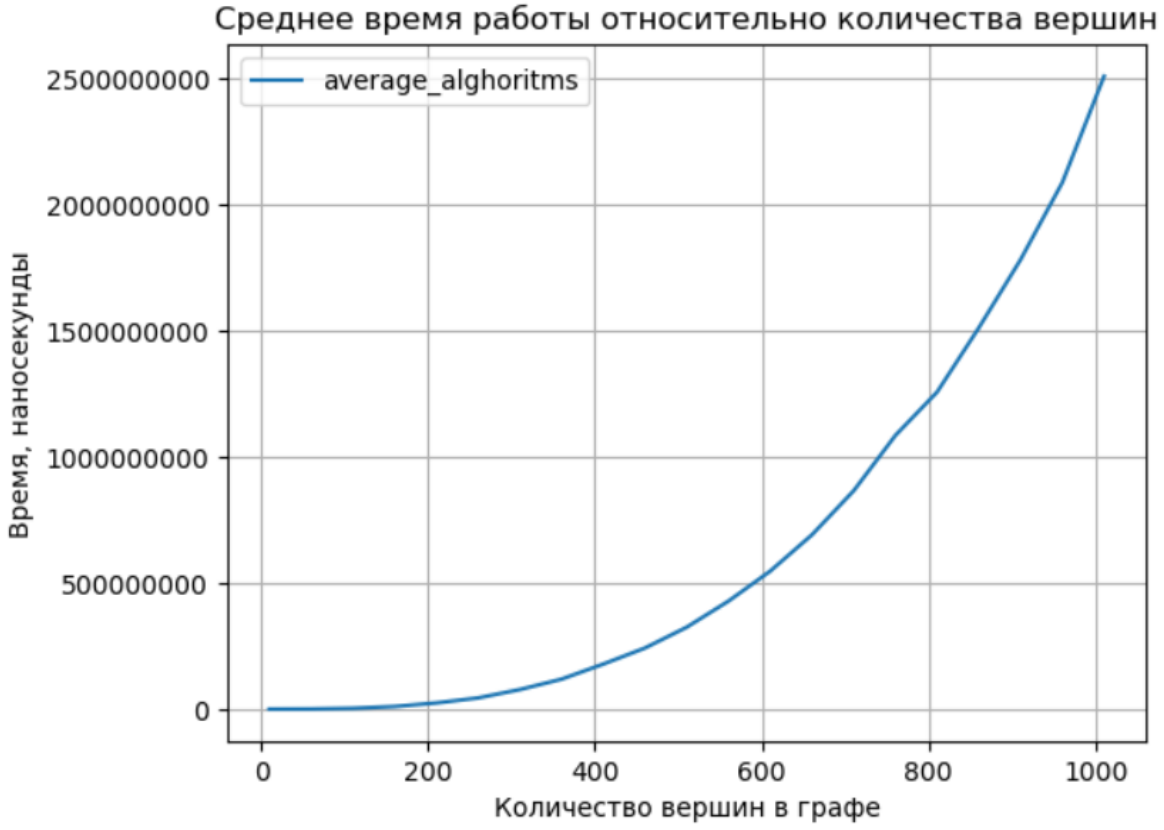


В случае связных графов с коэффициентом плотности 0,5 ситуация очень похожая, не учитывая, что теперь худшее время показывает алгоритм Флойда-Уоршелла, алгоритмы аналогично разбиваются по парам.


Разреженный граф
Dijkstra
Floyd-Warshall
Bellman-Ford
Optimized Bellman-Ford
3000000000
2500000000
2000000000
1500000000
1000000000
500000000
0
Время, наносекунды
0
200
400
600
800
1000
Количество вершин в графе


Разреженный граф
Dijkstra
Bellman-Ford
Optimized Bellman-Ford
20000000
17500000
15000000
12500000
10000000
7500000
5000000
2500000
0
Время, наносекунды
0
200
400
600
800
1000
Количество вершин в графе

При разреженных графах алгоритм Флойда-Уоршелла показывает худшее время, и в сравнении с ним остальные графы показывают отличное время, при ближайшем рассмотрении, выделяется алгоритм Беллман-Форда.



Алгоритм Дейкстра при всех видах графов показывает отличное время, так как он единственный ищет путь между конкретными двумя вершинами, а не просчитывает пути от одной до остальных вершин. Алгоритм Беллмана-Форда служит для поиска кратчайшего пути в том числе и в графах с отрицательным весом ребер, но в нашем случае все графы содержит только натуральные веса ребер, поэтому он закономерно проигрывает по времени. Алгоритм Флойда-Уоршелла всегда работает за O(n^3), поэтому показывает плохое время на всех видах графиков. Оптимизация алгоритма Беллмана-Форда: так как хоть этот алгоритм и просчитывает пути ко всем вершинам, но благодаря остановке при отсутствии нахождения более короткого пути, этот алгоритм показывает лучшее время для всех видов графиков и размеров.

В последних двух графиках я усреднял время между 4 алгоритмами, для того чтобы найти зависимость между количеством вершин (ребер) и временем работы алгоритмов в среднем. В итоге видно, что количество ребер в графе и время работы связаны линейно, а количество вершин ухудшает время по степенной функции. (O(n^2) или O(n^3)).