Правительство Российской Федерации

Федеральное государственное автономное учреждение высшего профессионального образования «Национальный исследовательский университет «Высшая школа экономики»
Факультет программной инженерии

Контрольное домашнее задание «Максимальные потоки в транспортной сети»

Работу выполнил
Самойлов Никита Андреевич
студент 2 курса ПИ ФКН
БПИ 171-1

Москва 2019

Оглавление

Постановка задачи

Для успешного выполнения работы необходимо:

- Изучить задачу поиска максимального потока в транспортной сети и методы ее решения.

- Разработать с использованием языка C++ программу, реализующую следующие алгоритмы расчета значения максимального потока в заданной транспортной сети:
    1. Базовая реализация метода Форда-Фалкерсона
    2. Метода Форда-Фалкерсона в версии Эдмондса-Карпа
    3. Алгоритм Ефима Диница

- Провести вычислительный эксперимент с целью оценки реализованных алгоритмов расчёта максимального потока.

- Подготовить отчет по итогам работы

Описание структуры проекта

Проект состоит из следующего набора файлов:

GraphReader.hpp
algs
graph
main.cpp
time

./algs:
Algorithm.hpp
DinicAlgorithm.hpp
EdmondsKarpAlgorithm.hpp
FordFulkersonAlgorithm.hpp

./graph:
Graph.hpp
MaxFlowGraph.hpp

./time:
CPPTimeMeasurer.hpp
ChronoTimeMeasurer.hpp
ClockTimeMeasurer.hpp
RDTSCTimeMeasurer.hpp
TimeMeasurer.hpp

GraphReader.hpp - описывает класс-шаблон для считывания графа из файла
algs - хранит классы алгоритмов
graph - хранит классы графов
main.cpp - входная точка программы (main())
time - хранит классы для подсчета времени работы алгоритмов

Algorithm.hpp - описывает абстрактный класс-шаблон для реализации алгоритмов
DinicAlgorithm.hpp - содержит класс-шаблон представляющий алгоритм Диницы
EdmondsKarpAlgorithm.hpp - содержит класс-шаблон представляющий алгоритм Эдмондса-Карпа
FordFulkersonAlgorithm.hpp - содержит класс-шаблон представляющий алгоритм Форда-Фалкерсона

Graph.hpp - описывает граф
MaxFlowGraph.hpp - описывает граф с истоком и стоком.

CPPTimeMeasurer.hpp - содержит метод измерения времени с помощью ф-ии time() из time.h
ChronoTimeMeasurer.hpp - содержит метод измерения времени с помощью библиотеки chrono
ClockTimeMeasurer.hpp - содержит метод измерения времени с помощью ф-ии clock() из time.h
RDTSCTimeMeasurer.hpp - содержит метод измерения времени с помощью команды процессора RDTSC
TimeMeasurer.hpp - содержит абстрактный класс для методов измерения времени.

Алгоритмы

Алгоритмы представлены в виде классов наследующих класс **Algorithm** и реализующие *operator(const Graph<>&)* (т.е. в виде функторов).

Граф

Граф представлен в виде класса шаблона **Graph** в основе которого лежит матрица смежностей. Также он содержит функционал:

- по добавлению новой вершины (*addVertex()*)
- *addVertex_functional* для дополнения реализации *addVertex()* в классах-наследниках
- обращению к элементу матрицы (*operator()(int,int)*)
- и др

MaxFlowGraph расширяет функционал **Graph**, добавляя:

- Сток и исток (*getSink() getSource()*)
- Поиск стока и истока (*refresh_source_and_sink()*). Если он находит больше чем 1 стока или 1 истока, автоматически создаются вершины описывающие сток и исток.

Измерение времени

В основе измерения времени работы алгоритмов лежит абстрактный класс **TimeMeasurer**, у которого есть:

- функция *measure(Algorithm& , Graph&)* для измерения времени работы алгоритма на данном графе (возвращает время)
- функция *getResult()* для получения последнего результата работы алгоритма (только после использования *measure*)

А также есть 2 абстрактные protected функции *_startMeasure()* и *_endMeasure()*, которые нужно реализовать в классе-наследнике, тем самым реализуя определенный метод подсчета времени работы алгоритма.

Описание алгоритмов

Базовая реализация метода Форда-Фалкерсона (Ford–Fulkerson algorithm) [2]

Алгоритм реализован в виде класса FordFulkersonAlgorithm<> .Начинает он свою работу с того, что создает копию графа (rgraph), которая будет остаточным графом. Далее находим любой увеличивающий путь с помощью алгоритма поиска в глубину и насыщаем путь минимальным найденным потоком. Повторяем до тех пор, пока существует этот путь.

Метод Форда-Фалкерсона в версии Эдмондса-Карпа (Edmonds–Karp algorithm) [2]

Алгоритм реализован в виде класса EdmondsKarpAlgorithm<>, который наследуется от FordFulkersonAlgorithm<>, переопределяет метод поиска увеличивающего пути, проводя поиск с помощью алгоритма поиска в ширину.

Алгоритм Ефима Диница (Dinic's/Dinitz's algorithm). [3]

Алгоритм реализован в виде класса DinicAlgorithm<>. Суть алгоритма состоит в том, чтобы с помощью поиска в ширину расставить уровни (vector<> *level*), которые представляют расстояние от истока до стока, для вершин. Далее строим граф остаточной сети (у меня он не строится, а просто принимается во внимание содержимое вектора level). Проходимся по остаточному графу до стока с помощью алгоритма поиска в глубину и насыщаем по пути ребра минимальным потоком. Так делаем до тех пор, пока существует путь от истока к стоку. Если его нет, строим новый граф поиском в ширину до тех пор пока на исходном графе есть путь из истока в сток и опять выполняем поиск в глубину.

Описание плана эксперимента

Для проведения эксперимента необходимо:

- Запустить все алгоритмы на всех исходных данных
- Избежать получения недостоверных данных

Вследствие этого, был выбран следующий способ получения данных:

1. Прочитать файл
2. Выполнить 10 раз каждый из алгоритмов
3. Найти среднее время выполнения каждого алгоритма
4. Записать полученные результаты в файл
5. Повторить

Измерение времени будет производится с помощью ассемблерных вставок (инструкции RDTSC) [1]. На выходе мы получаем количество тактов процессора.

Результаты эксперимента

N - кол-во вершин в графе
FF - алгоритм Форда-Фалкерсона
EK - алгоритм Эдмондса-Карпа
DN - алгоритм Диница

Результаты представлены в наносекундах. (такты переведены в наносекунды)

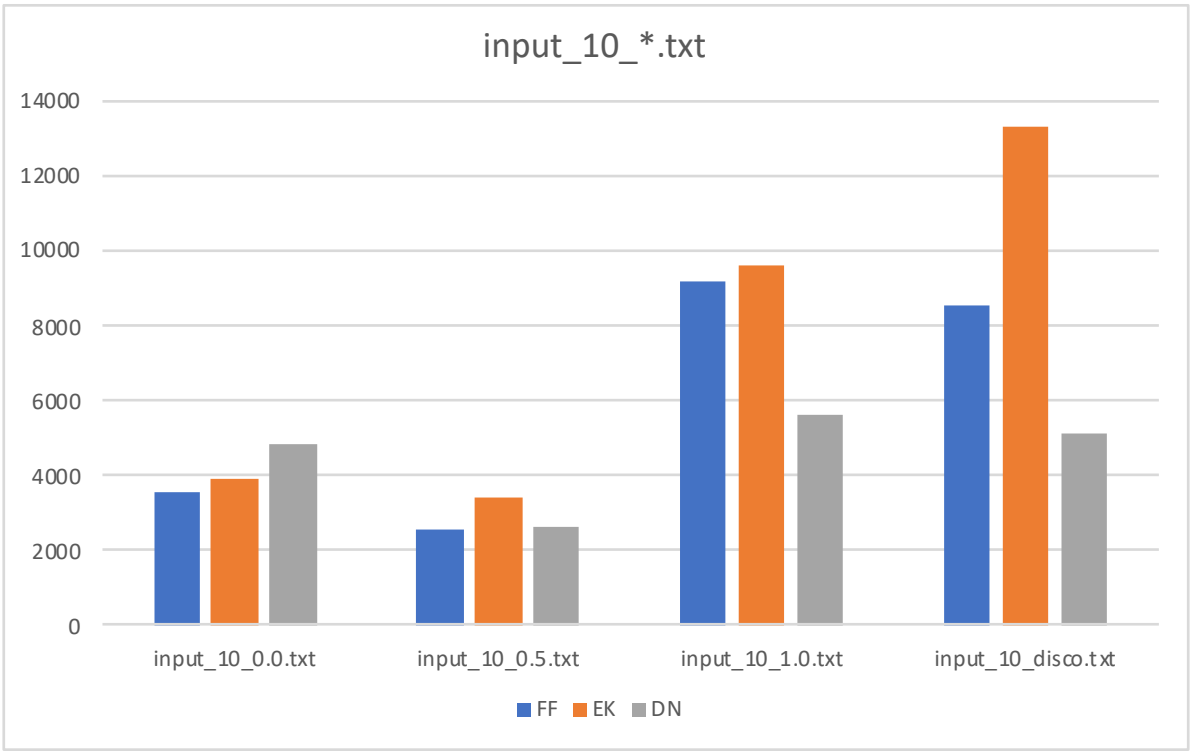
File	FF	EK	DN
input_10_0.0.txt	3545,61	3937,68	4838,71
input_10_0.5.txt	2545,42	3421,74	2642,55
input_10_1.0.txt	9172,65	9635,65	5641,87
input_10_disco.txt	8547,13	13331,3	5129,94
input_310_0.0.txt	8,64E+06	1,55E+06	3,83E+06
input_310_0.5.txt	6,56E+08	4,05E+07	1,33E+08
input_310_1.0.txt	1,26E+09	8,37E+07	2,26E+07
input_310_disco.txt	7,21E+08	5,46E+07	6,60E+07
input_610_0.0.txt	4,81E+07	7,26E+06	1,28E+07
input_610_0.5.txt	5,23E+09	3,10E+08	1,92E+09
input_610_1.0.txt	1,05E+10	5,66E+08	1,29E+08
input_610_disco.txt	5,43E+09	3,12E+08	7,55E+08
input_910_0.0.txt	1,92E+08	2,83E+07	4,09E+07
input_910_0.5.txt	1,64E+10	1,04E+09	9,43E+09
input_910_1.0.txt	3,42E+10	1,87E+09	4,21E+08
input_910_disco.txt	2,12E+10	1,11E+09	3,28E+09
input_1210_0.0.txt	3,50E+08	5,33E+07	1,12E+08
input_1210_0.5.txt	3,92E+10	2,38E+09	2,88E+10
input_1210_1.0.txt	7,95E+10	4,35E+09	9,27E+08
input_1210_disco.txt	4,78E+10	2,53E+09	1,36E+10
input_1510_0.0.txt	3,46E+08	5,36E+07	1,06E+08
input_1510_0.5.txt	7,93E+10	4,76E+09	6,58E+10
input_1510_1.0.txt	1,54E+11	8,31E+09	1,75E+09
input_1510_disco.txt	9,53E+10	5,34E+09	2,21E+10
input_1810_0.0.txt	7,33E+08	7,01E+07	1,30E+08
input_1810_0.5.txt	1,35E+11	8,00E+09	1,38E+11
input_1810_1.0.txt	2,68E+11	1,46E+10	3,28E+09
input_1810_disco.txt	1,84E+11	8,97E+09	6,76E+10
input_2110_0.0.txt	5,56E+08	8,28E+07	2,04E+08

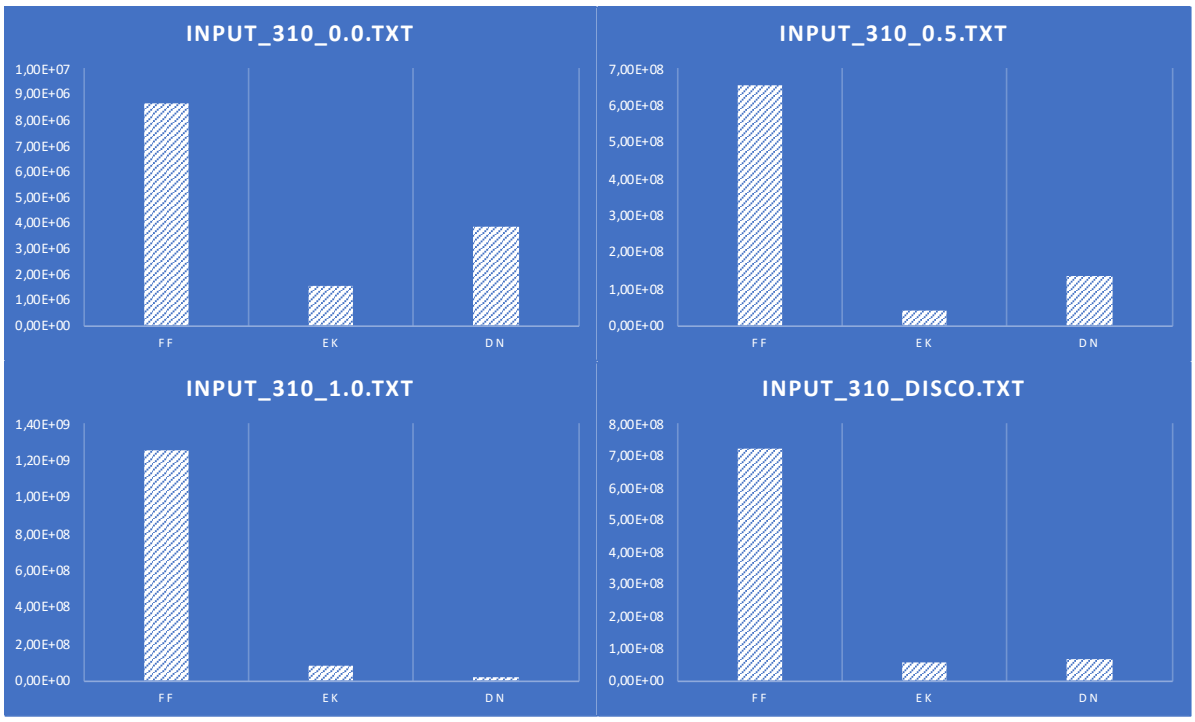
input_2110_0.5.txt	2,14E+11	1,30E+10	2,75E+11
input_2110_1.0.txt	4,07E+11	2,28E+10	5,37E+09
input_2110_disco.txt	2,96E+11	1,51E+10	9,44E+10
input_2410_0.0.txt	4,03E+08	6,49E+07	1,81E+08

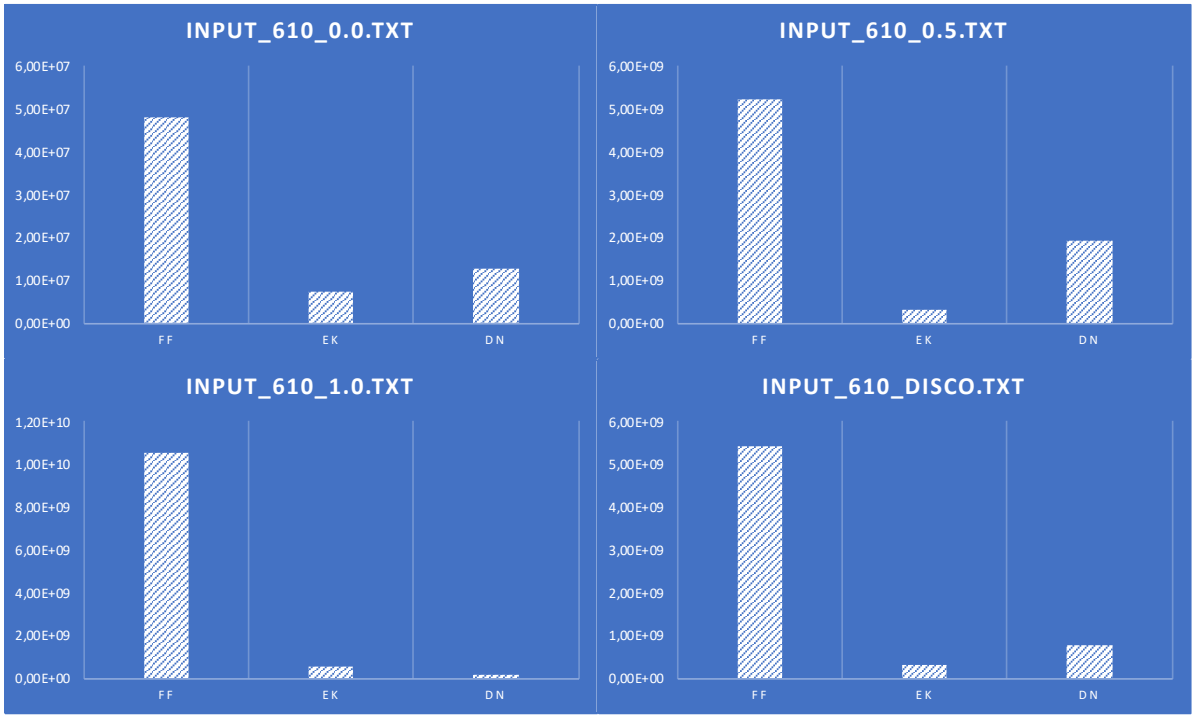
Результаты работы алгоритмов

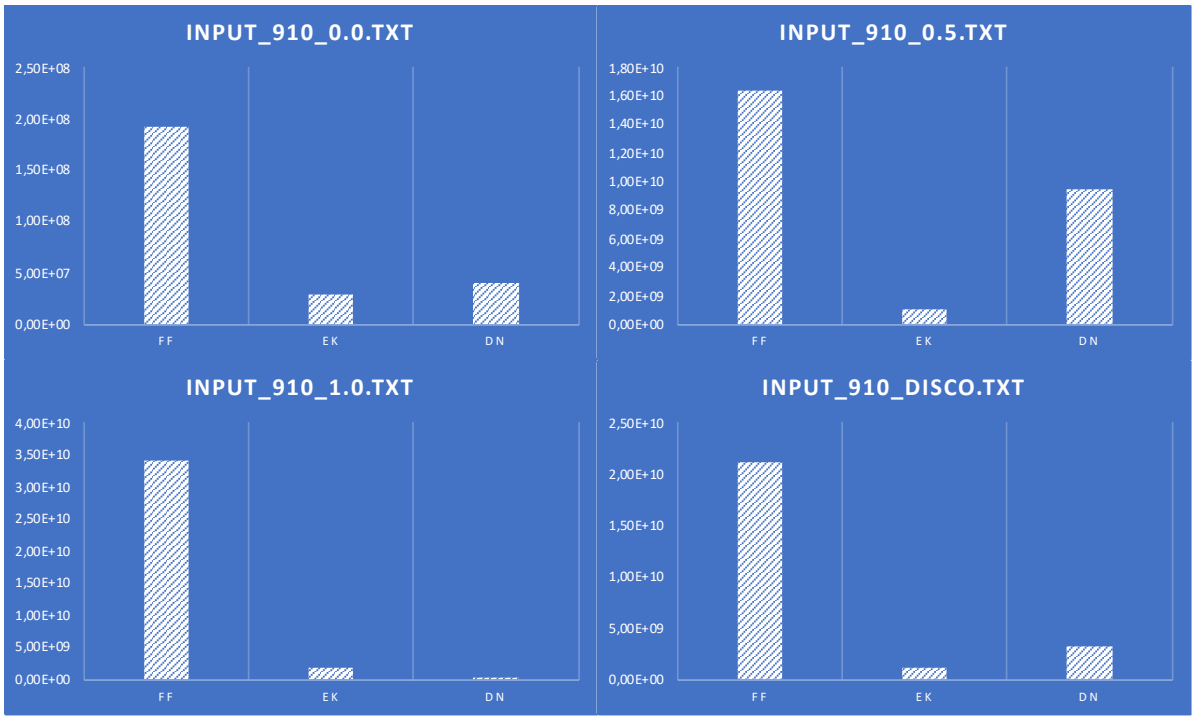
File	Result	File	Result
input_10_0.0.txt	31	input_1510_0.0.txt	88
input_10_0.5.txt	45	input_1510_0.5.txt	41844
input_10_1.0.txt	486	input_1510_1.0.txt	80504
input_10_disco.txt	338	input_1510_disco.txt	38754
input_310_0.0.txt	63	input_1810_0.0.txt	122
input_310_0.5.txt	7817	input_1810_0.5.txt	49487
input_310_1.0.txt	16157	input_1810_1.0.txt	99508
input_310_disco.txt	7656	input_1810_disco.txt	48794
input_610_0.0.txt	87	input_2110_0.0.txt	67
input_610_0.5.txt	16592	input_2110_0.5.txt	58419
input_610_1.0.txt	33270	input_2110_1.0.txt	111565
input_610_disco.txt	14500	input_2110_disco.txt	57884
input_910_0.0.txt	150	input_2410_0.0.txt	42
input_910_0.5.txt	23492	input_2410_0.5.txt	64995
input_910_1.0.txt	49212	input_2410_1.0.txt	129297
input_910_disco.txt	24059	input_2410_disco.txt	62977
input_1210_0.0.txt	140	input_2710_0.0.txt	134
input_1210_0.5.txt	31993	input_2710_0.5.txt	74659
input_1210_1.0.txt	65717	input_2710_1.0.txt	145476
input_1210_disco.txt	30665	input_2710_disco.txt	72513

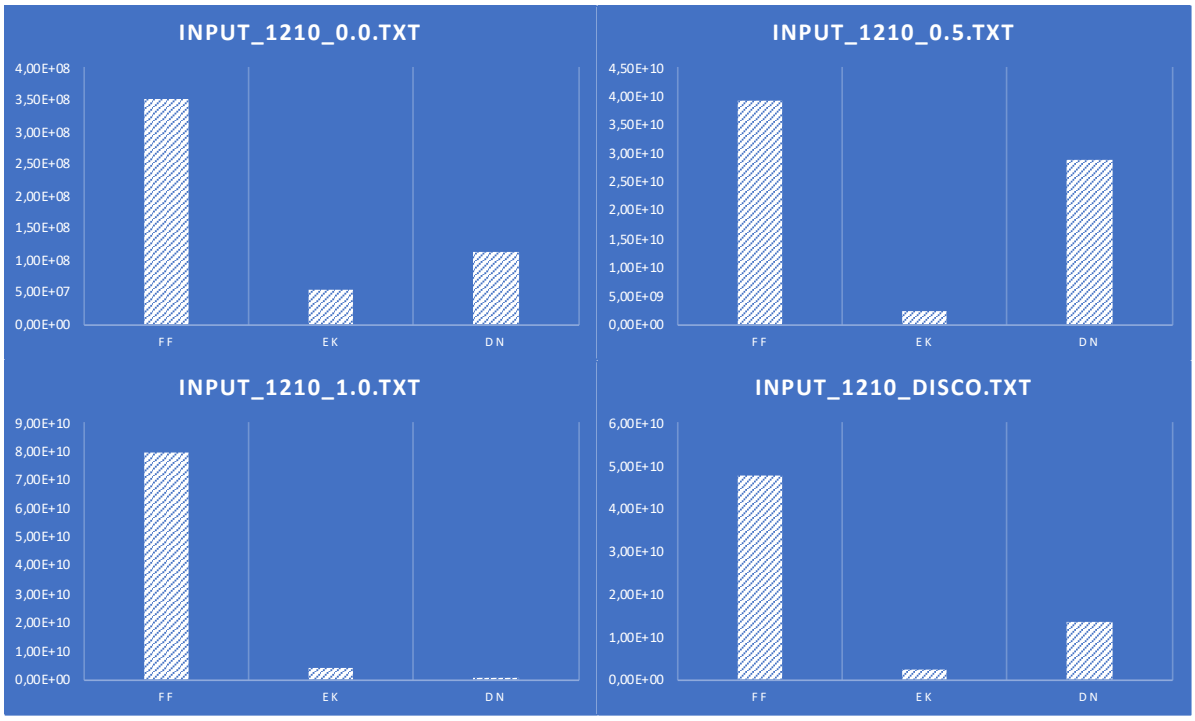
Гистограммы, отражающие время выполнения алгоритмов на одних исходных данных.

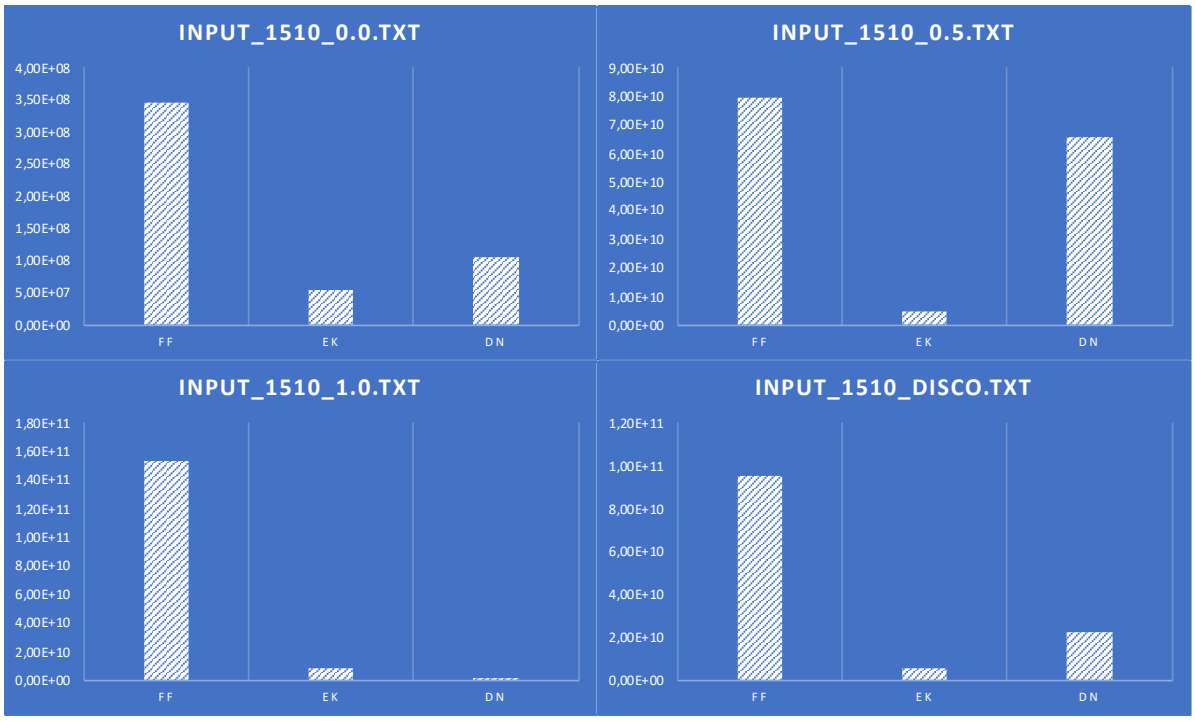


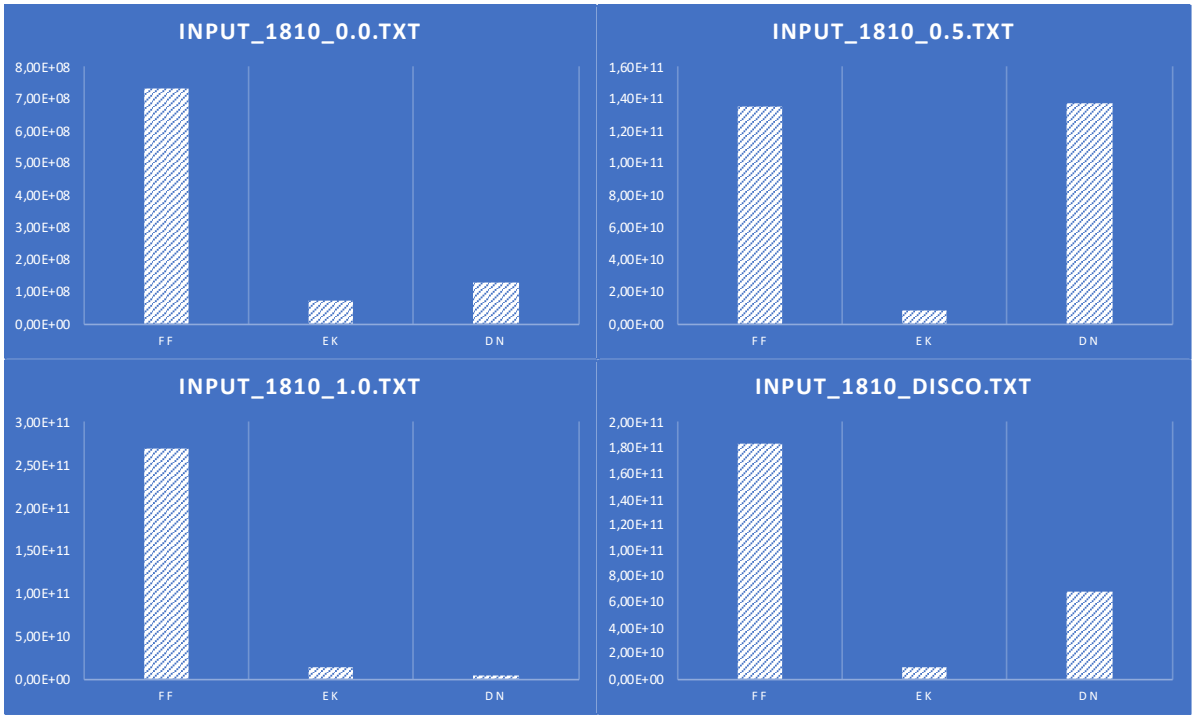


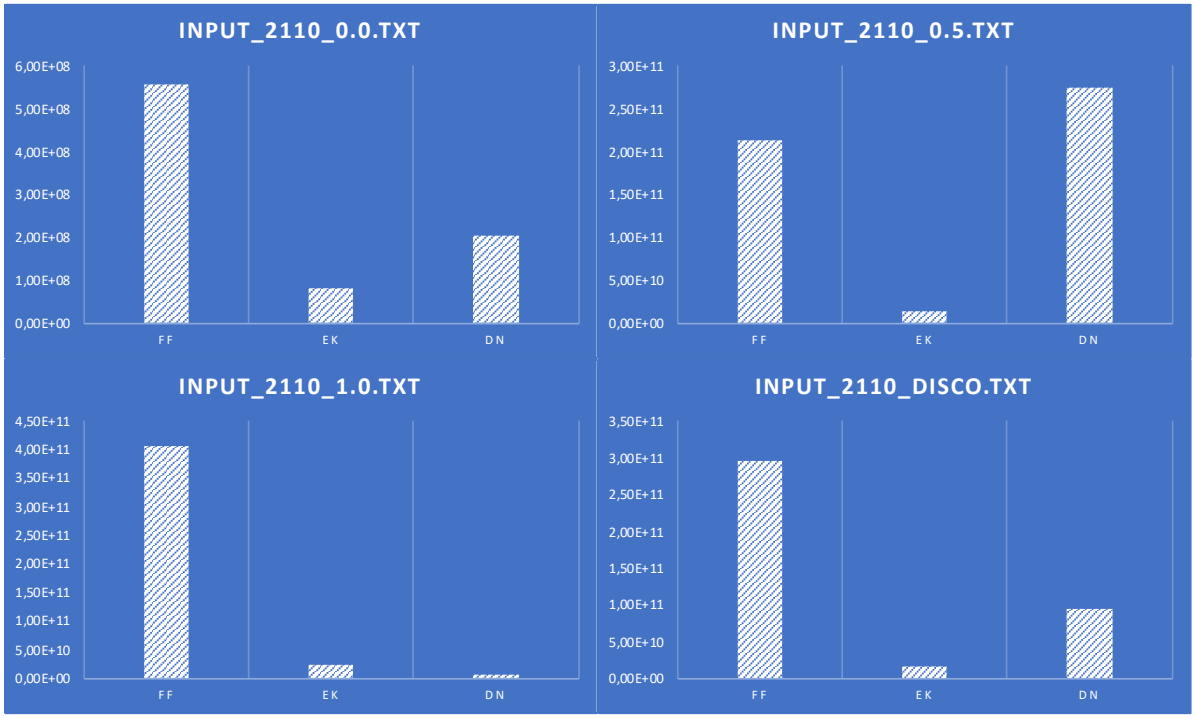

INPUT_610_0.0.TXT
6,00E+07
5,00E+07
4,00E+07
3,00E+07
2,00E+07
1,00E+07
0,00E+00
FF
EK
DN
INPUT_610_0.5.TXT
6,00E+09
5,00E+09
4,00E+09
3,00E+09
2,00E+09
1,00E+09
0,00E+00
FF
EK
DN
INPUT_610_1.0.TXT
1,20E+10
1,00E+10
8,00E+09
6,00E+09
4,00E+09
2,00E+09
0,00E+00
FF
EK
DN
INPUT_610_DISCO.TXT
6,00E+09
5,00E+09
4,00E+09
3,00E+09
2,00E+09
1,00E+09
0,00E+00
FF
EK
DN


INPUT_910_0.0.TXT
2,50E+08
2,00E+08
1,50E+08
1,00E+08
5,00E+07
0,00E+00
FF
EK
DN
INPUT_910_0.5.TXT
1,80E+10
1,60E+10
1,40E+10
1,20E+10
1,00E+10
8,00E+09
6,00E+09
4,00E+09
2,00E+09
0,00E+00
FF
EK
DN
INPUT_910_1.0.TXT
4,00E+10
3,50E+10
3,00E+10
2,50E+10
2,00E+10
1,50E+10
1,00E+10
5,00E+09
0,00E+00
FF
EK
DN
INPUT_910_DISCO.TXT
2,50E+10
2,00E+10
1,50E+10
1,00E+10
5,00E+09
0,00E+00
FF
EK
DN

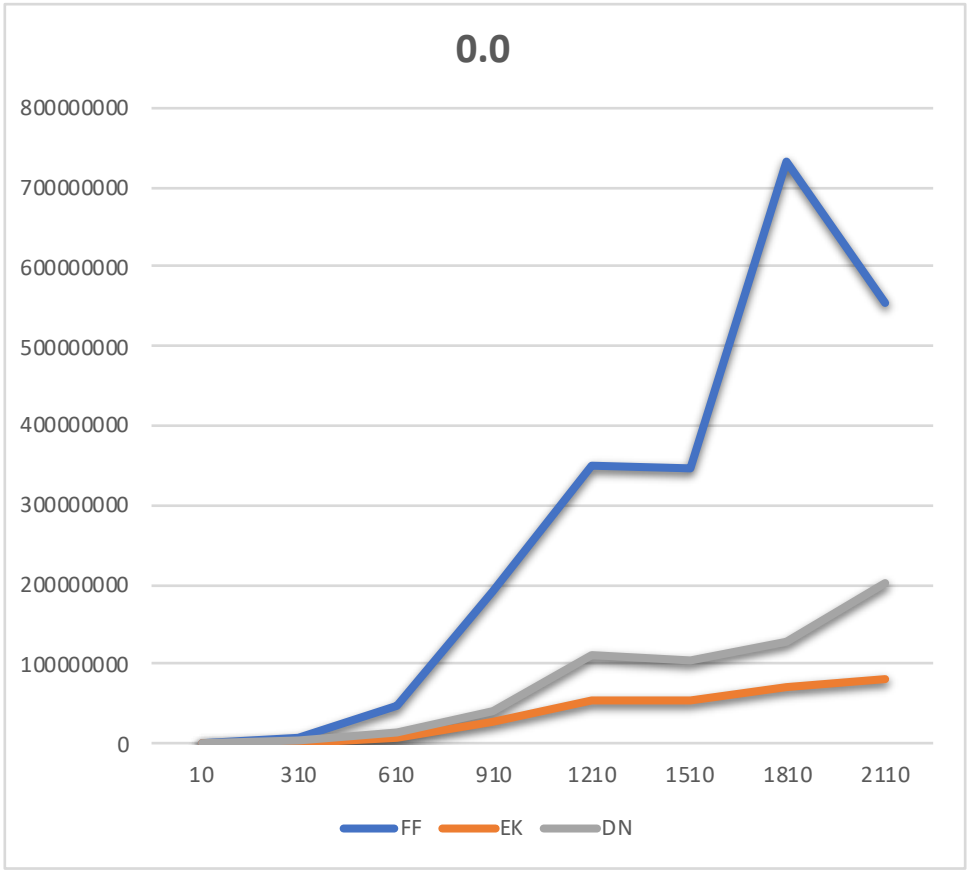

INPUT_1210_0.0.TXT
4,00E+08
3,50E+08
3,00E+08
2,50E+08
2,00E+08
1,50E+08
1,00E+08
5,00E+07
0,00E+00
FF
EK
DN
INPUT_1210_0.5.TXT
4,50E+10
4,00E+10
3,50E+10
3,00E+10
2,50E+10
2,00E+10
1,50E+10
1,00E+10
5,00E+09
0,00E+00
FF
EK
DN
INPUT_1210_1.0.TXT
9,00E+10
8,00E+10
7,00E+10
6,00E+10
5,00E+10
4,00E+10
3,00E+10
2,00E+10
1,00E+10
0,00E+00
FF
EK
DN
INPUT_1210_DISCO.TXT
6,00E+10
5,00E+10
4,00E+10
3,00E+10
2,00E+10
1,00E+10
0,00E+00
FF
EK
DN

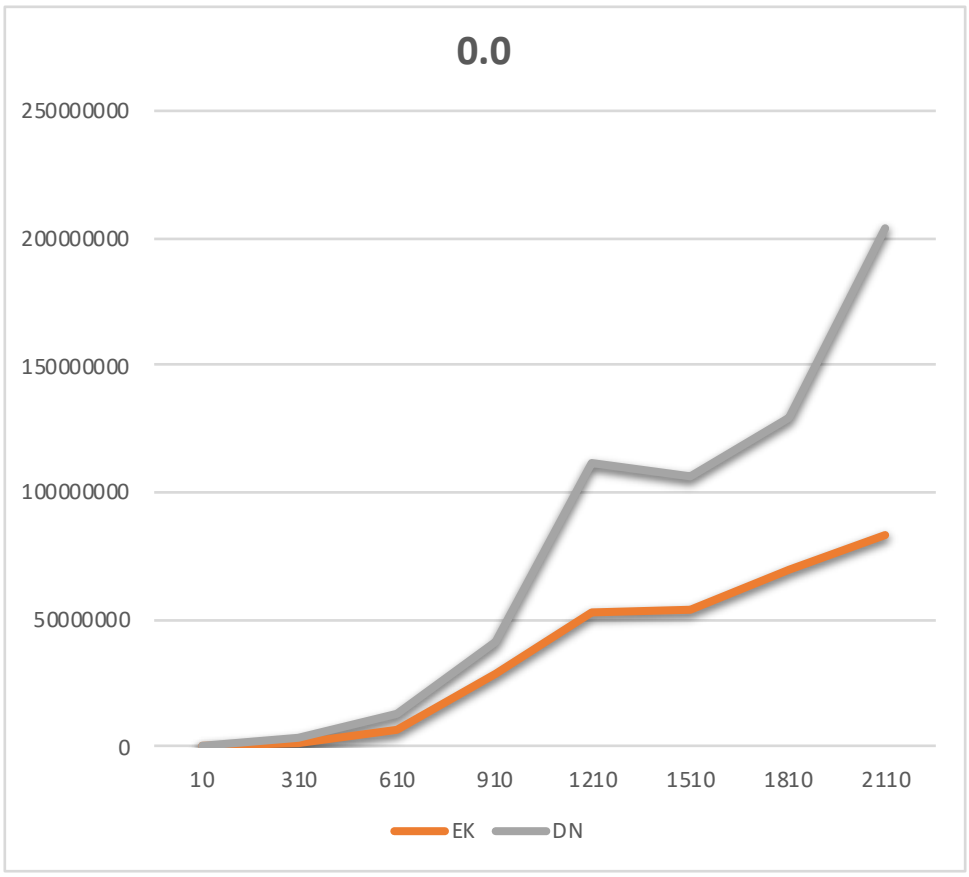

INPUT_1510_0.0.TXT
4,00E+08
3,50E+08
3,00E+08
2,50E+08
2,00E+08
1,50E+08
1,00E+08
5,00E+07
0,00E+00
FF
EK
DN
INPUT_1510_0.5.TXT
9,00E+10
8,00E+10
7,00E+10
6,00E+10
5,00E+10
4,00E+10
3,00E+10
2,00E+10
1,00E+10
0,00E+00
FF
EK
DN
INPUT_1510_1.0.TXT
1,80E+11
1,60E+11
1,40E+11
1,20E+11
1,00E+11
8,00E+10
6,00E+10
4,00E+10
2,00E+10
0,00E+00
FF
EK
DN
INPUT_1510_DISCO.TXT
1,20E+11
1,00E+11
8,00E+10
6,00E+10
4,00E+10
2,00E+10
0,00E+00
FF
EK
DN

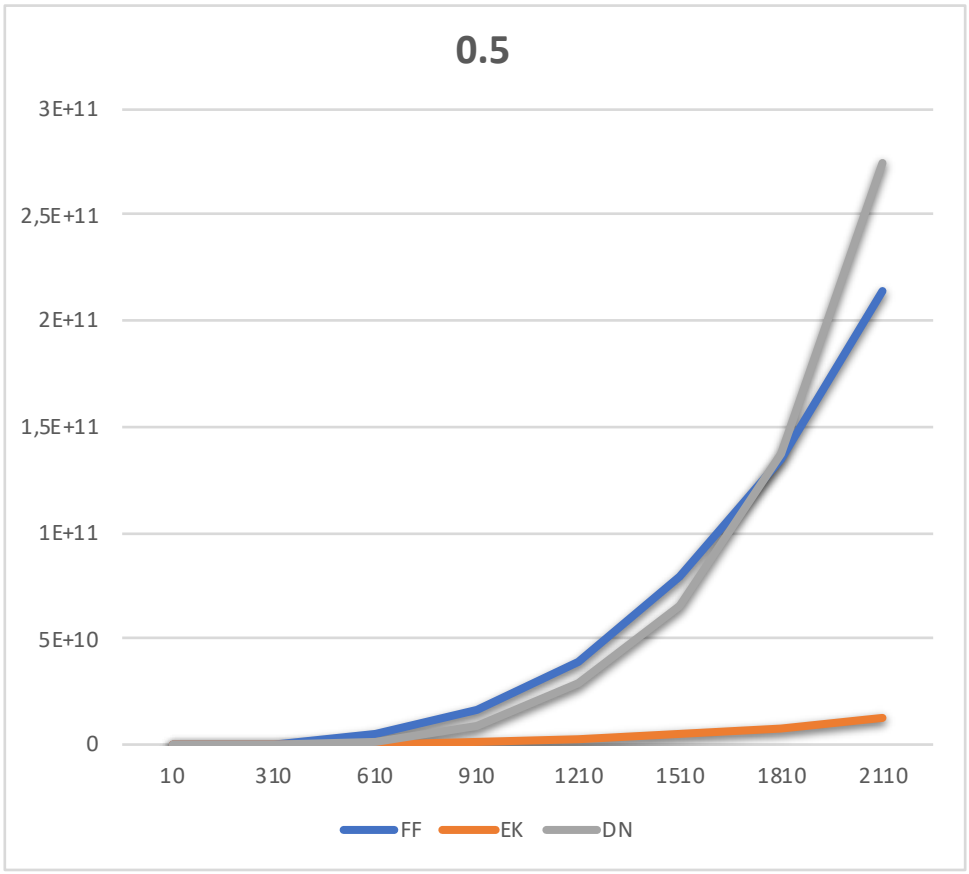

INPUT_1810_0.0.TXT
8,00E+08
7,00E+08
6,00E+08
5,00E+08
4,00E+08
3,00E+08
2,00E+08
1,00E+08
0,00E+00
FF
EK
DN
INPUT_1810_0.5.TXT
1,60E+11
1,40E+11
1,20E+11
1,00E+11
8,00E+10
6,00E+10
4,00E+10
2,00E+10
0,00E+00
FF
EK
DN
INPUT_1810_1.0.TXT
3,00E+11
2,50E+11
2,00E+11
1,50E+11
1,00E+11
5,00E+10
0,00E+00
FF
EK
DN
INPUT_1810_DISCO.TXT
2,00E+11
1,80E+11
1,60E+11
1,40E+11
1,20E+11
1,00E+11
8,00E+10
6,00E+10
4,00E+10
2,00E+10
0,00E+00
FF
EK
DN

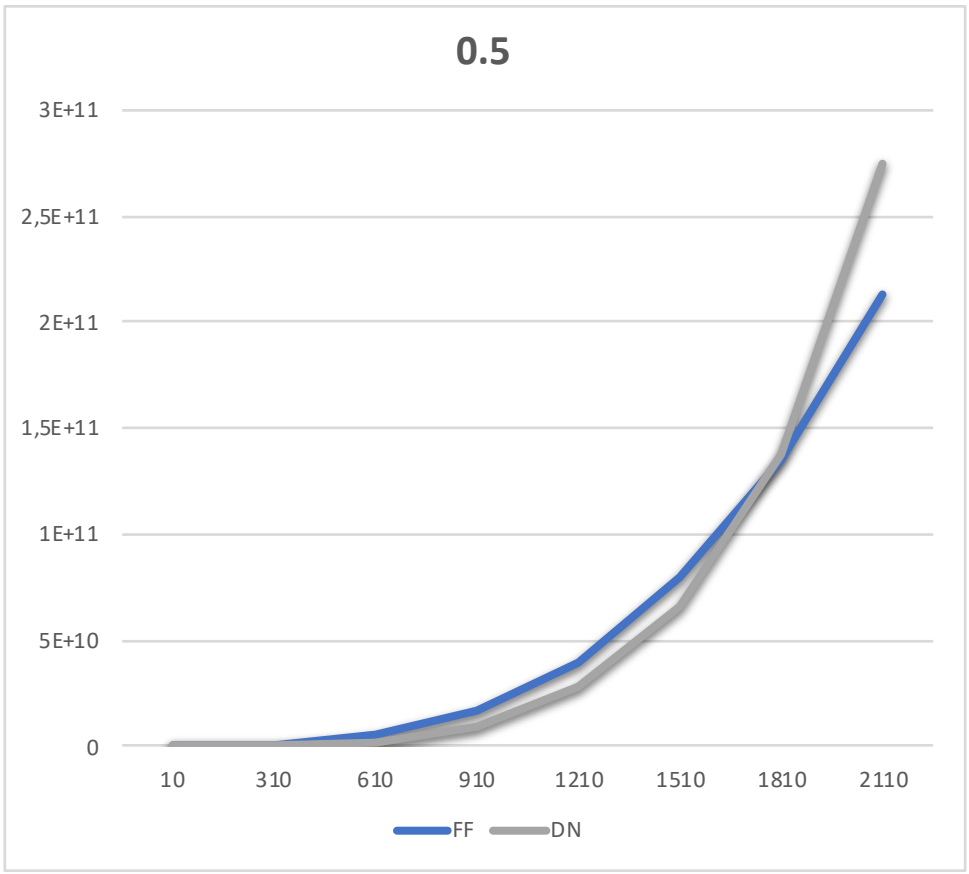

INPUT_2110_0.0.TXT
6,00E+08
5,00E+08
4,00E+08
3,00E+08
2,00E+08
1,00E+08
0,00E+00
FF
EK
DN
INPUT_2110_0.5.TXT
3,00E+11
2,50E+11
2,00E+11
1,50E+11
1,00E+11
5,00E+10
0,00E+00
FF
EK
DN
INPUT_2110_1.0.TXT
4,50E+11
4,00E+11
3,50E+11
3,00E+11
2,50E+11
2,00E+11
1,50E+11
1,00E+11
5,00E+10
0,00E+00
FF
EK
DN
INPUT_2110_DISCO.TXT
3,50E+11
3,00E+11
2,50E+11
2,00E+11
1,50E+11
1,00E+11
5,00E+10
0,00E+00
FF
EK
DN

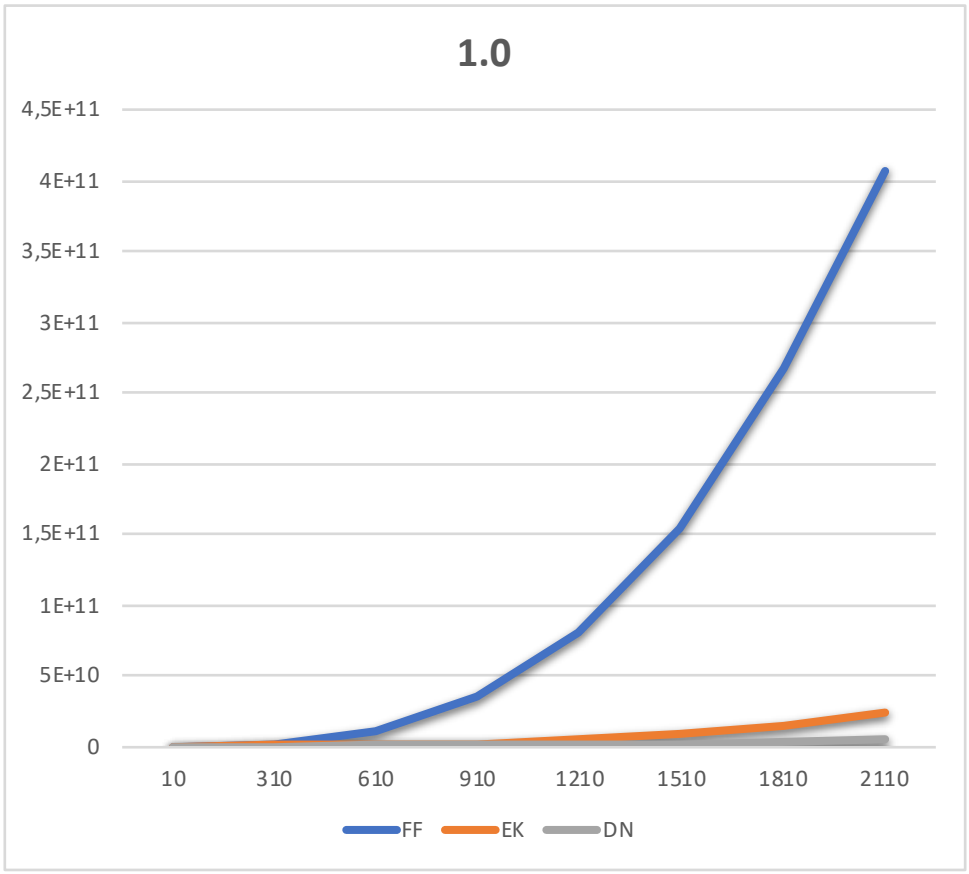
Ниже приведены графики замеров времени работы алгоритмов для каждого из типов графа.

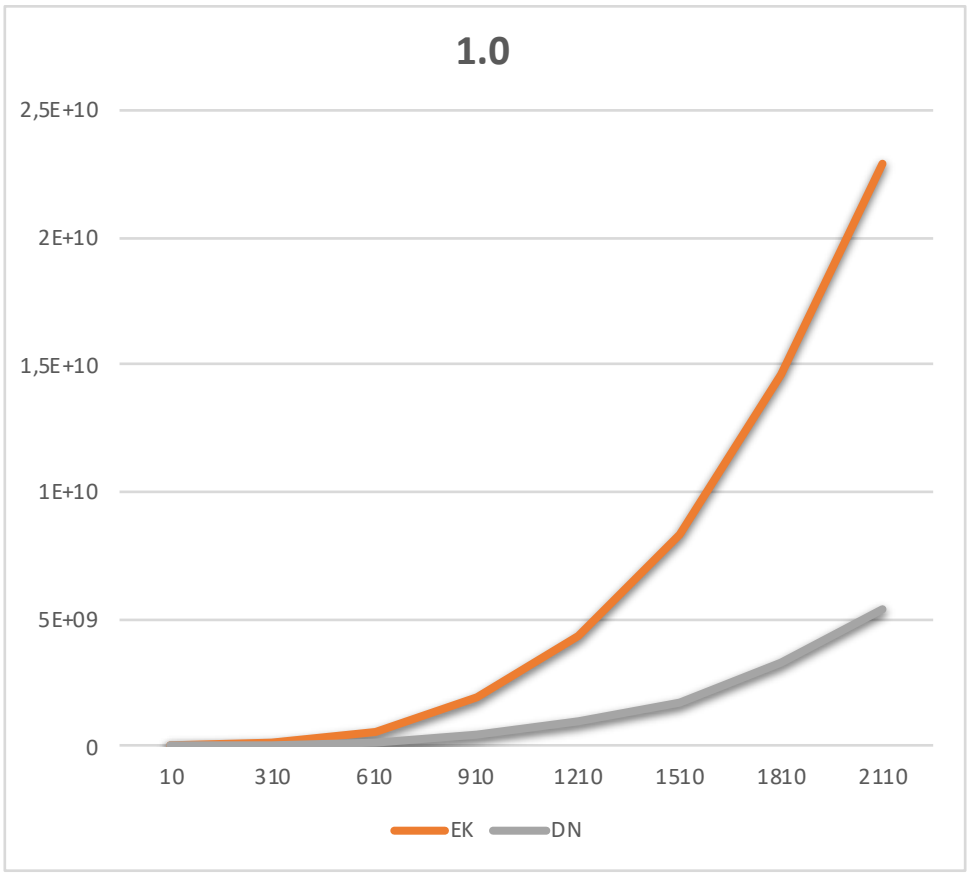


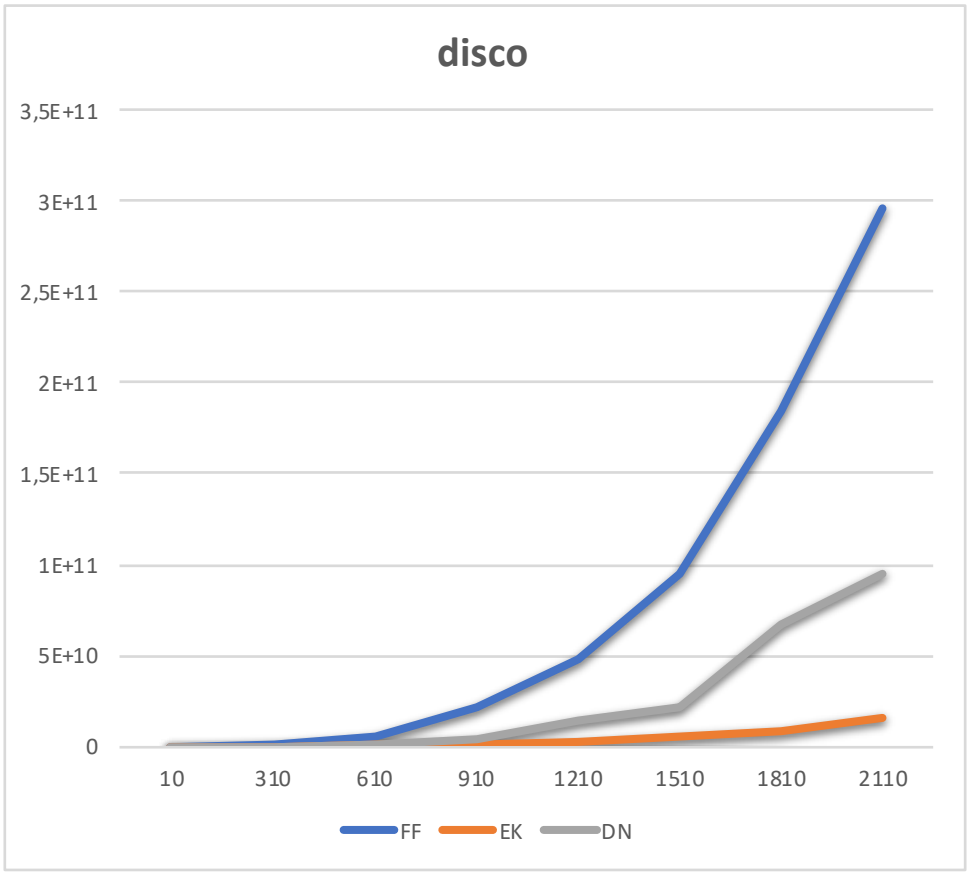


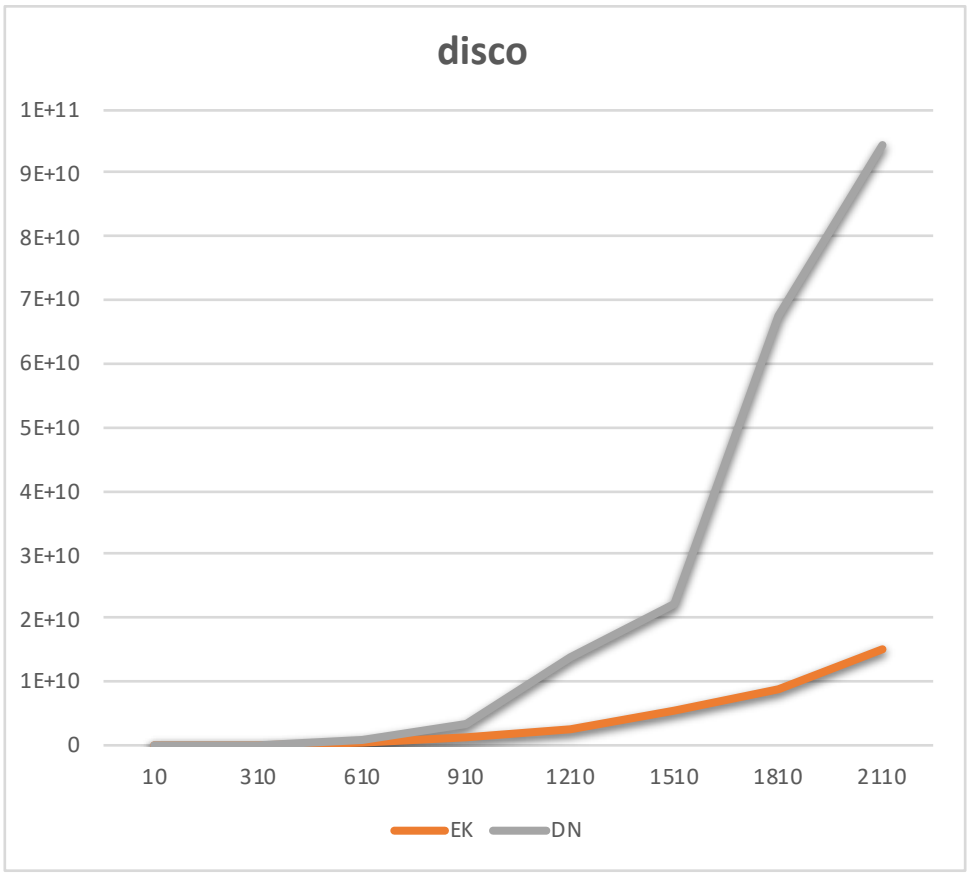

0.5
3E+11
2,5E+11
2E+11
1,5E+11
1E+11
5E+10
0
10
310
610
910
1210
1510
1810
2110
FF
EK
DN


0.5
3E+11
2,5E+11
2E+11
1,5E+11
1E+11
5E+10
0
10
310
610
910
1210
1510
1810
2110
FF
DN


1.0
4,5E+11
4E+11
3,5E+11
3E+11
2,5E+11
2E+11
1,5E+11
1E+11
5E+10
0
10
310
610
910
1210
1510
1810
2110
FF
EK
DN


1.0
2,5E+10
2E+10
1,5E+10
1E+10
5E+09
0
10
310
610
910
1210
1510
1810
2110
EK
DN


disco
3,5E+11
3E+11
2,5E+11
2E+11
1,5E+11
1E+11
5E+10
0
10
310
610
910
1210
1510
1810
2110
FF
EK
DN


disco
1E+11
9E+10
8E+10
7E+10
6E+10
5E+10
4E+10
3E+10
2E+10
1E+10
0
10
310
610
910
1210
1510
1810
2110
EK
DN

Сравнительный анализ алгоритмов

Все алгоритмы используют различный подходи к решению задачи. И в связи с этим имеют разную сложность, следовательно, имеют различия во времени выполнения на разных исходных данных.

Скорость работы алгоритма Форда-Фалкерсона зависит от значения максимального потока. Если оно небольшое, то время выполнения будет неплохое.

От этой зависимости избавляет другая реализация этого алгоритма, использующая обход в ширину (алгоритм Эдмонда-Карпа). И это видно по результатам эксперимента. Время выполнения этого алгоритма на несколько порядков меньше чем у базового Форда-Фалкерсона.

Алгорит Диница имеет большее преимущество на графах высокой плотности за счет наиболее эффективного поиска увеличивающего пути. Но из-за того, что на графах большой размерности и меньшей плотности ему приходится строить новый остаточный граф поиском в ширину, он проигрывает другим алгоритмам, потому как они хоть и не имеют быстрого поиска, им не приходится постоянно строить новый граф.

Вывод

Как видно по графикам выше на разряженных графах лучше всего показывает себя алгоритм Эдмондса-Карпа, а на графах близких к полным алгоритм Диница имеет существенные результаты по сравнению с остальными. Алгоритм Форда-Фалкерсона же быстрее остальных лишь на некоторых исходных графах небольшой размерности и малой плотности. Все это объясняется сложностью данных алгоритмов:

Алгоритма Форда-Фалкерсона имеет сложность O(E*max|f|), что означает что на больших данных он будет работать медленней всех.

Алгоритм Эдмонд-Карпа имеет сложность O(VE^2) , что объясняет его быстроту на разряженных графах.

Алгоритм Диница имеет сложность O(V^2E) и O(V* sqrt(E)) для единичных пропускных способностей, что объясняет быстроту на графах большой плотности.

Использованные источники

1. [Описание метода измерения времени] https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf

2. [Алгоритм Форда-Фалкерсона] Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и
анализ, 3-е изд.: Пер. с англ. - М.: ООО “И.Д. Вильямс”, 2016. - Глава 26. Задача о максимальном потоке.

3. [Алгоритм Диницы] https://en.wikipedia.org/wiki/Dinic%27s_algorithm