Национальный исследовательский университет
«Высшая школа экономики»

Контрольная домашняя работа
по предмету
«Алгоритмы и структуры данных»

Студент: Королев Дмитрий Павлович
Группа: БПИ-173 (1)

Москва, 2019

ОГЛАВЛЕНИЕ

Постановка задачи

- Задание: реализовать три алгоритма расчета максимального потока в транспортной сети, провести экспериментальное исследование их эффективности и анализ полученных результатов.

Тестируемые алгоритмы:

1. Базовая реализация метода Форда-Фалкерсона
2. Метод Форда-Фалкерсона в версии Эдмонда-Карпа
3. Алгоритм Ефима-Диница

Задачей было разработать программу на языке *C++*, позволяющую удобным способом протестировать каждый из алгоритмов на данных в условии графах.

Введем определения типов графов, которые будет далее использовать:

Rarefied – разреженные графы (*0.0.txt)
Medium – средние по плотности графы(*0.5.txt)
Tight – плотные графы (*1.0.txt)
Disco – несвязные графы (*disco.txt)

Использовалась версия языка C++14. Ниже приведены технические характеристики устройства, на котором проводился эксперимент:

- Intel Core i7-8700 CPU @ 3.20GHz
- Asus Prime Z370-A

Описание алгоритмов и использованных структур данных

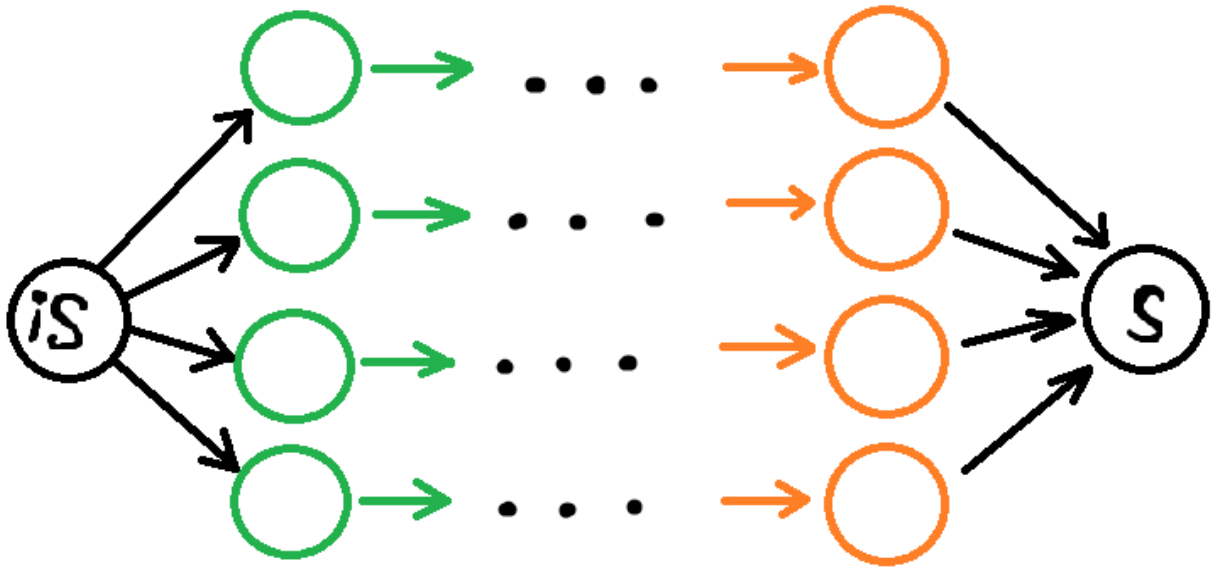
Каждый из данных графов был переведен в матрицу смежности. Реализовывалась матрица смежности в виде *vector<vector<int>>*.

Для каждого алгоритма сперва проводилась операция поиска истоков и стоков. В данной работе истоки и стоки определялись так:

Исток – та вершина, в которую не приходит ни одно ребро.

Сток – та вершина, из которой не исходит ни одно ребро.

На файлах типа *rarefied, medium* и *tight* алгоритм лишь выбирал единственный исток и единственный сток. В случае файлов *disco* данный алгоритм также добавлял в матрицу две вершины. Одна из них является ***фейковым-истоком***, который имеет ребра, идущие в все истоки. Другая из них является ***фейковым-стоком***, в который идут ребра, идущие из всех стоков. Иначе говоря, преобразуем граф таким образом:



На рисунке черные вершины – фейковые исток и сток
Зеленые – истоки, рыжие – стоки

Далее запускался каждый из алгоритмов. Важно понимать, что вес ребра в матрице является количеством единиц потока, которые ЕЩЕ можно послать по данному ребру. Иначе говоря, в многих определениях у ребра *u* есть *u.f* – поток, идущий по ребру, и *u.c* – величина потока, которую в теории можно пустить (capacity). Вес ребра в матрице в таком случае: (*u.c* – *u.f*)

- **Алгоритм Форда-Фалкерсона**

  Изначально создается переменная full_flow, значение которой равно нулю. Затем с помощью запуска обхода графа в глубину (dfs) находится путь от истока к стоку, и совместно с этим считается минимум потока, который можно пустить. Это минимум из всех значений весов ребер пути, который идет от истока к стоку. Далее значение минимума потока вычитается из каждого из этих ребер, а по обратным из них это значение прибавляется. Таким образом «блокируется» этот путь, ведь алгоритм не пойдет по нему, т.к. какое-то из ребер имеет вес 0. Значение переменной full_flow инкрементируется на полученное значение обходом в глубину. Обход в глубину запускается снова, при этом используемый вектор «посещений» каждой из вершин снова обнуляется – в ином случае алгоритм не сможет пройти по вершинам, использованным ранее, хотя возможно по ним можно провести еще один путь.

  Кратко:

  1. Ищется путь от истока к стоку с помощью **DFS**, по которому можно пустить поток. В ином случае (не найден путь) алгоритм завершает свою работу.
  2. К общему значению величины потока прибавляется поток, который был запущен по найденному пути.
  3. Возвращаемся к пункту 1.

  UPD: Алгоритм был изменен после продолжительных тестов. Рекурсия при запуске алгоритма «ела» очень много памяти и уничтожала скорость работы. Поэтому было применено использование обхода в глубину с помощью структуры *std::stack*. При каждом вызове метода dfs() создавался стек, который заполнялся вершинами. Алгоритм стал существенно быстрее работать, а также получилось избежать создания вектора посещений, а соответственно и его повторного инициализирования.

  Асимптотика:

  Увеличивающий путь может быть найден за $O(|E|)$

  Минимальный инкрементирующий поток -1

  Поэтому общее время работы $- O(|E| * f)$, где f $-$ величина потока

- **Алгоритм Эдмонда-Карпа**

  Это иная версия алгоритма Форда-Фалкерсона. Используется другой способ нахождения пути от истока до стока – **BFS** (обход графа в ширину). По аналогии с «обновленным» алгоритмом Форда-Фалкерсона я использовал структуру данных *std::queue*. Во время обхода происходит запоминание очереди посещенных вершин. Когда обход графа завершается, идет проверка – был ли посещен сток. Если не был, то метод bfs возвращает 0. В ином случае в векторе, который использовался для запоминания очереди посещенных вершин, хранится путь от истока к стоку. Тогда ищется минимальный вес ребра, который есть на этом пути, а затем у каждого ребра пути вычитается этот вес, а к обратным он прибавляется.

  Кратко:

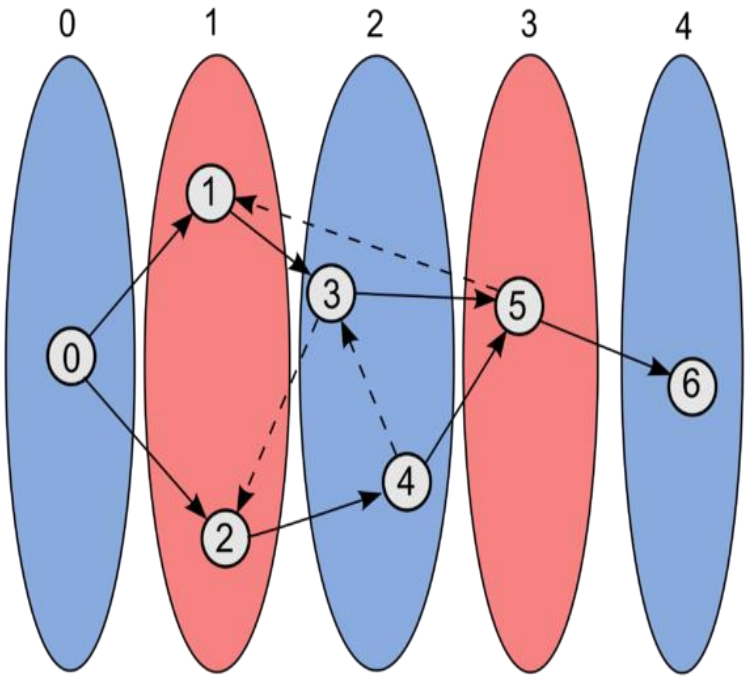
  1. Ищется путь от истока к стоку с помощью **BFS**, по которому можно пустить поток. В ином случае (не найден путь) алгоритм завершает работу.
  2. К общему значению величины потока прибавляется поток, который был запущен по найденному пути.
  3. Возвращаемся к пункту 1.

  Асимптотика:

  Каждая итерация цикла *while* выполнима за $O(VE)$
  Требуемая инициализация и запись ребер $- O(E)$
  Поэтому общее время работы $- O(VE^2)$

- **Алгоритм Ефима-Диница**

  Этот алгоритм сложнее по своей идее, чем предыдущие. Сперва, используя обход графа в ширину, строится слоистая сеть.



Картинка с АлгоВики

Слоистая сеть – то множество ребер, которые идут от вершины на уровне *x* к вершине на уровне *x+1*. Ребра, нарисованные прерывистой линией, не входят в слоистую сеть. При этом исток находится на нулевом уровне.

После построения слоистой сети запускается проход графа в глубину. Обход происходит только по тем ребрам, которые входят в слоистую сеть! Это позволяет максимально быстро искать блокирующие потоки.

Обход в глубину происходит до тех пор, пока получается найти блокирующий поток. Когда не удалось, происходит перестроение слоистой сети. В алгоритме bfs построения слоистой сети в этот же момент происходит проверка достижимости стока из истока. Если сток недостижим, то выполнение алгоритма завершается.

Асимптотика:

Алгоритм совершает не более чем n $-$ 1 проходов, ведь путь от истока до стока не превосходит n $-$ 1. Поэтому алгоритм выполняется за $O(V^2E)$

Описание плана эксперимента

Замеры времени делались с использованием библиотеки *std::chrono*.

В программе были написаны 3 функции, которые позволяют запустить работу каждого из алгоритмов. При этом в качестве параметра туда передавалась группа файлов – *rarefied, medium, tight* или *disco*.

Так как алгоритмы работали очень долго, то было принято решение проводить замеры лишь для файлов до 1810 элементов (включительно), а также при времени работы алгоритма над графом превосходящим 200 секунд, не высчитывалось среднее время работы, а это значение и было результатом. При этом среднее по алгоритмам Эдмондса-Карпа и Ефима-Диница высчитывалось по 10 замерам, а в случае Форда-Фалкерсона – лишь по 6-ти после некоторого количества тестирования скорости работы алгоритмов.

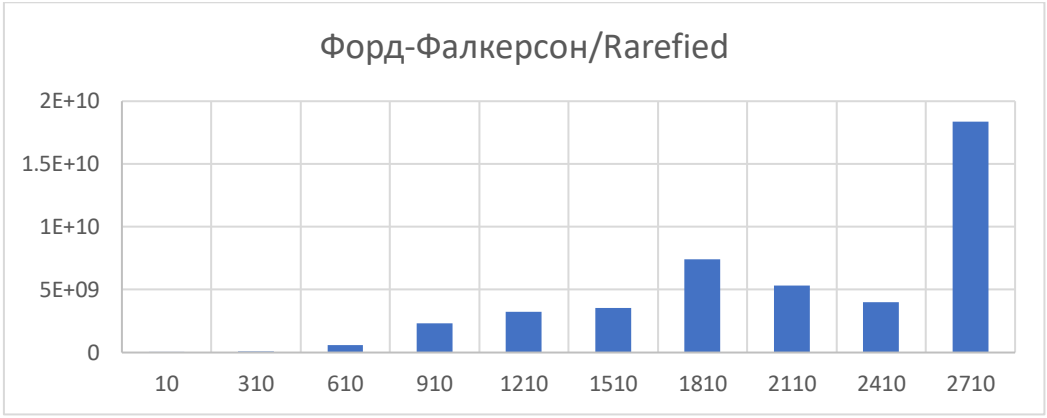
Структура проекта:

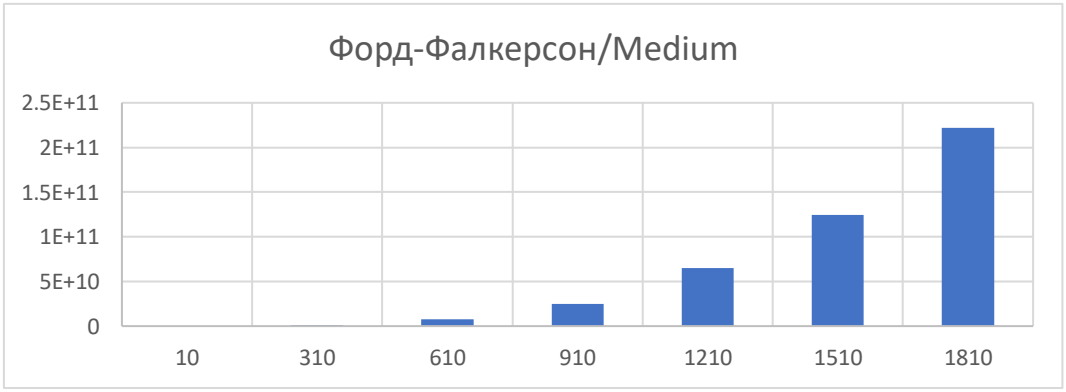
- Папка *input* – входные файлы
- Папка *docs* – документы с заданиями. Этот отчет находится в этой папке.
- Папка *output* – основной Excel-файл *Results* и папки для каждого из алгоритмов. В них в каждом файле находятся выходные данные, состоящие из 3-х частей:

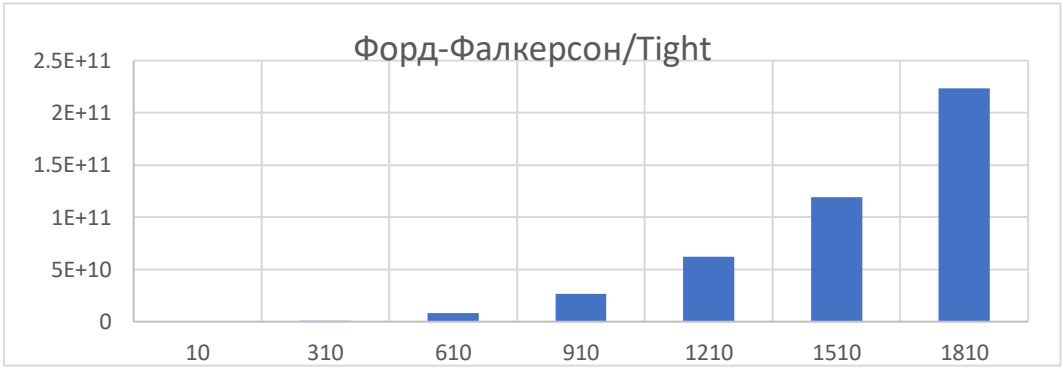
  1) количество наносекунд, потраченных на вычисление потока
  2) величина потока – результат работы алгоритмов
  3) матрица, в которой каждое ребро *uv* - величина потока, запущенная по данному ребру.

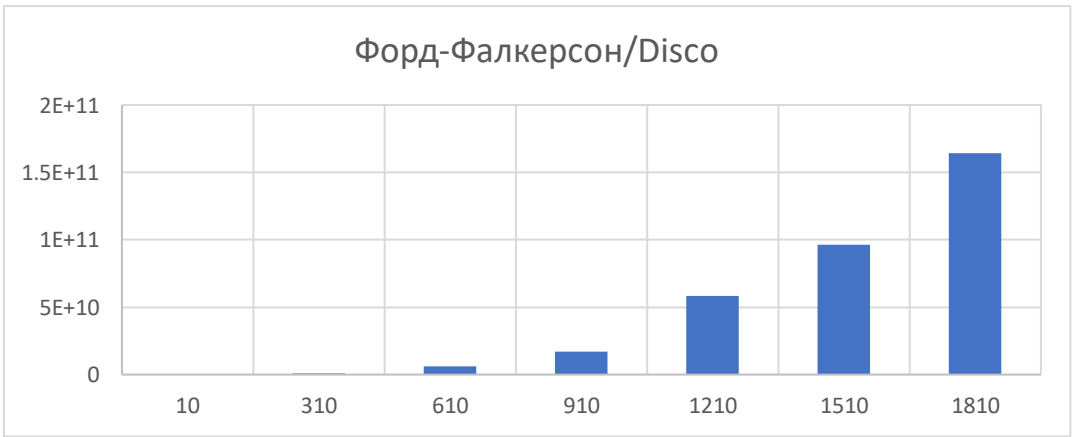
Результаты экспериментов

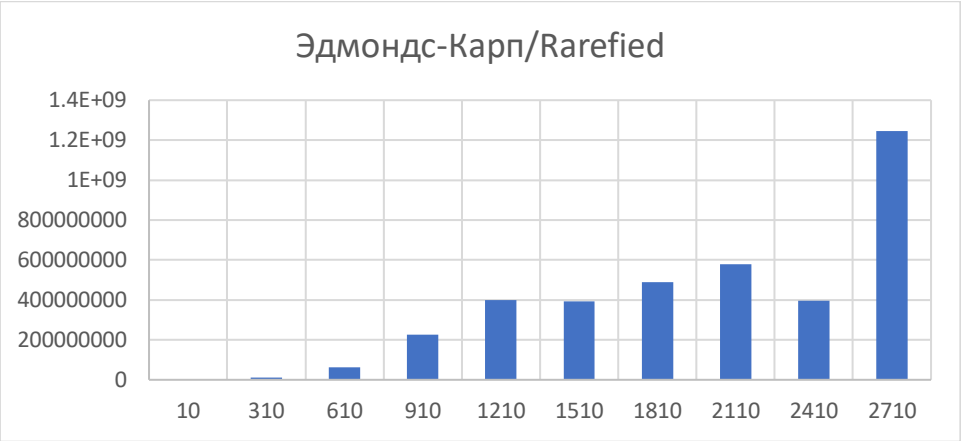
На каждом графике ось абсцисс – количество элементов в графе.
Ось ординат – количество наносекунд, затраченное на поиск потока.
Наименование диаграммы обозначает тип файлов и используемый алгоритм.

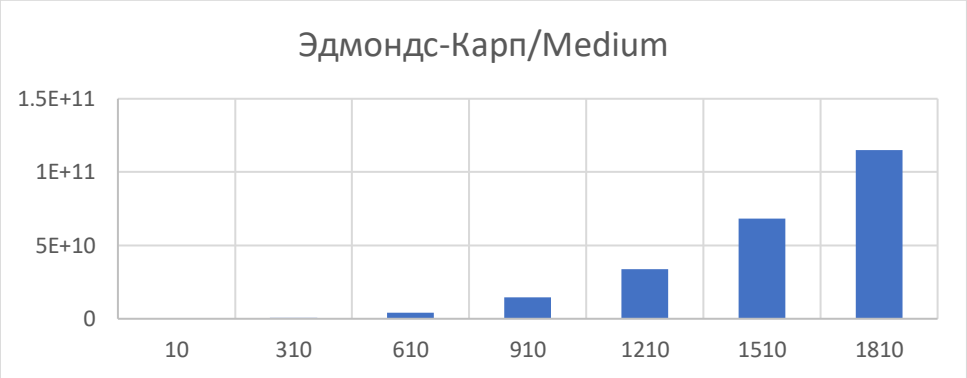


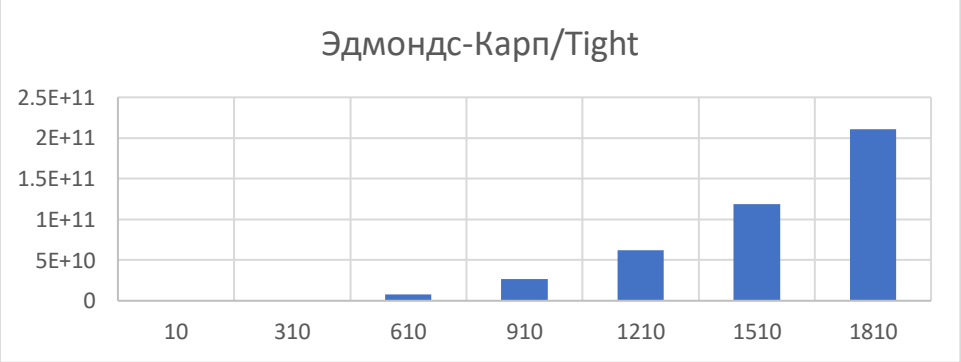


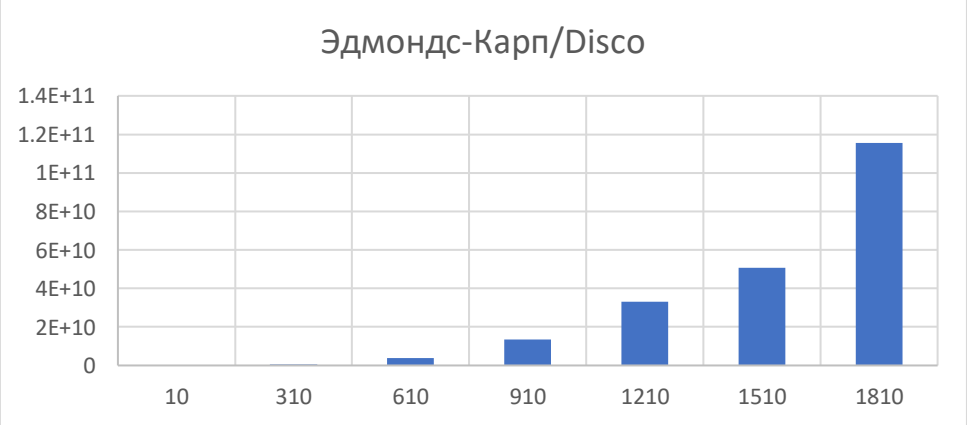


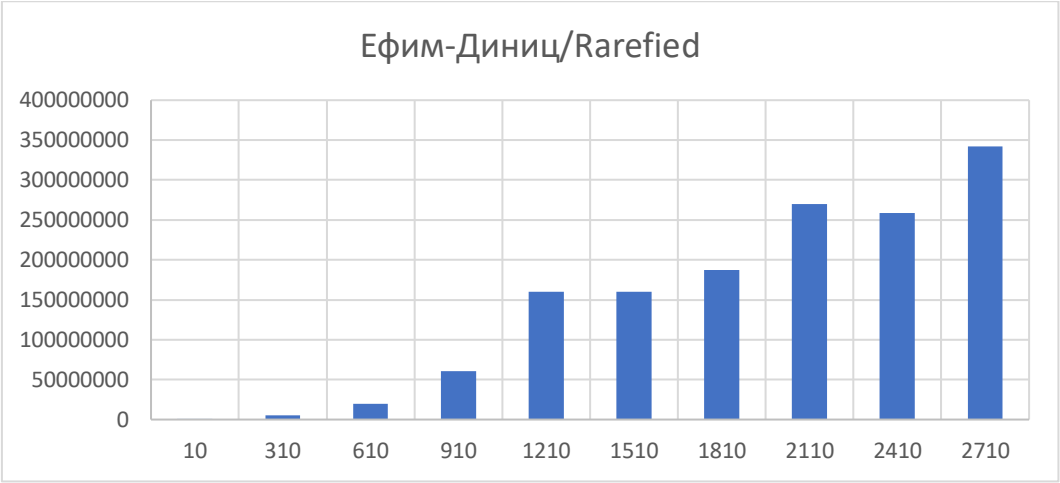


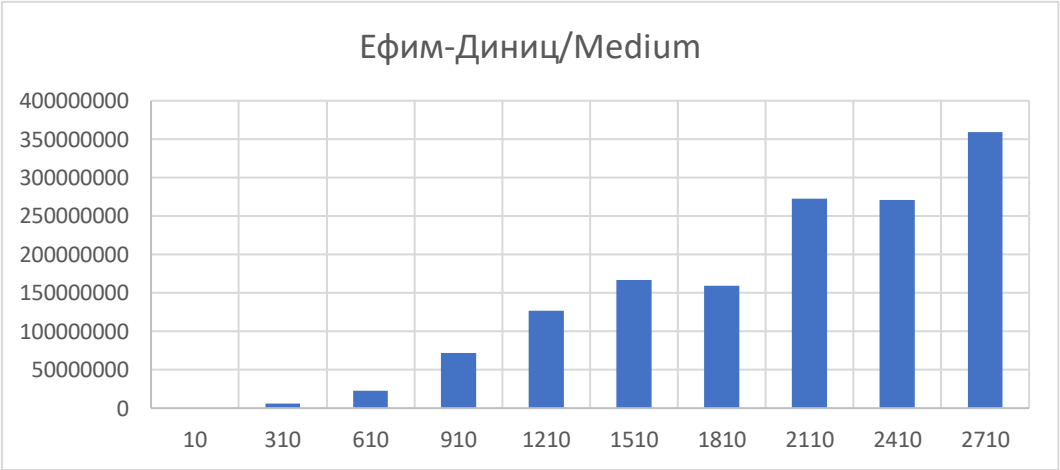

Эдмондс-Карп/Rarefied
1.4E+09
1.2E+09
1E+09
800000000
600000000
400000000
200000000
0
10
310
610
910
1210
1510
1810
2110
2410
2710

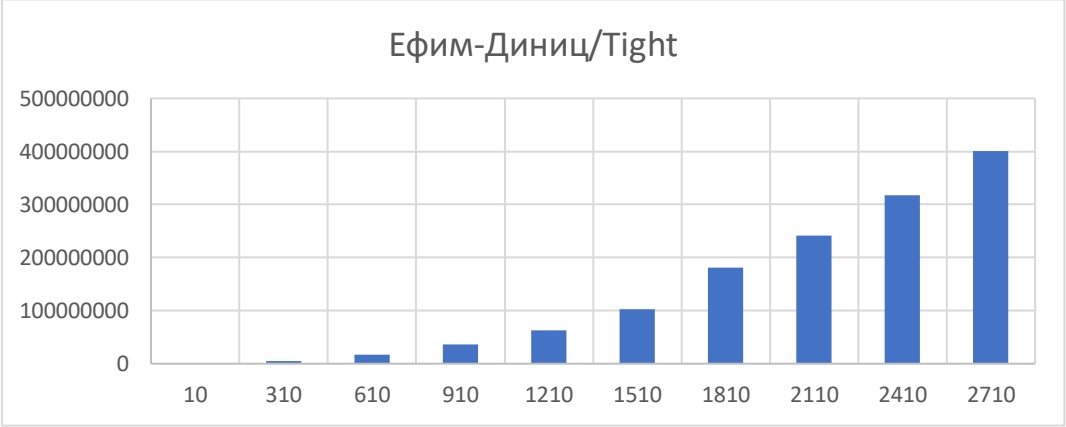

Эдмондс-Карп/Medium
1.5E+11
1E+11
5E+10
0
10
310
610
910
1210
1510
1810

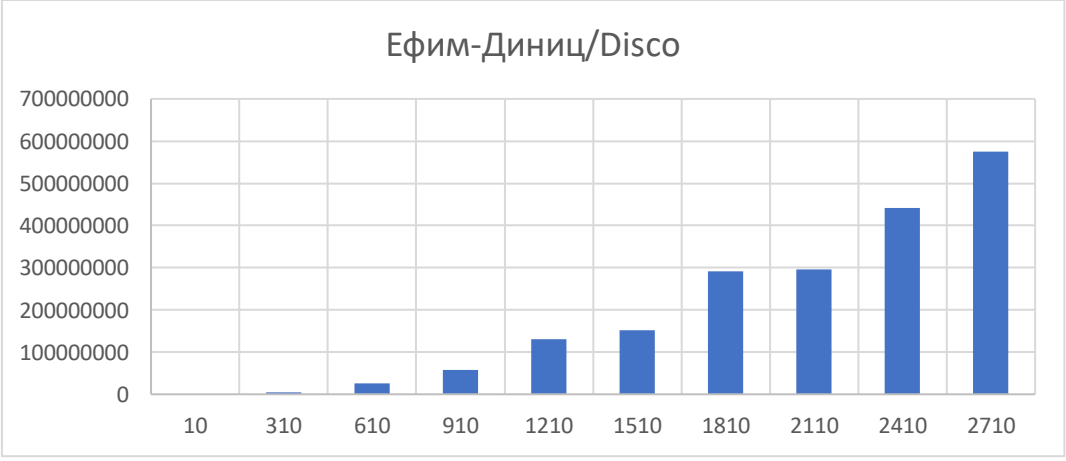

Эдмондс-Карп/Tight
2.5E+11
2E+11
1.5E+11
1E+11
5E+10
0
10
310
610
910
1210
1510
1810


Эдмондс-Карп/Disco
1.4E+11
1.2E+11
1E+11
8E+10
6E+10
4E+10
2E+10
0
10
310
610
910
1210
1510
1810

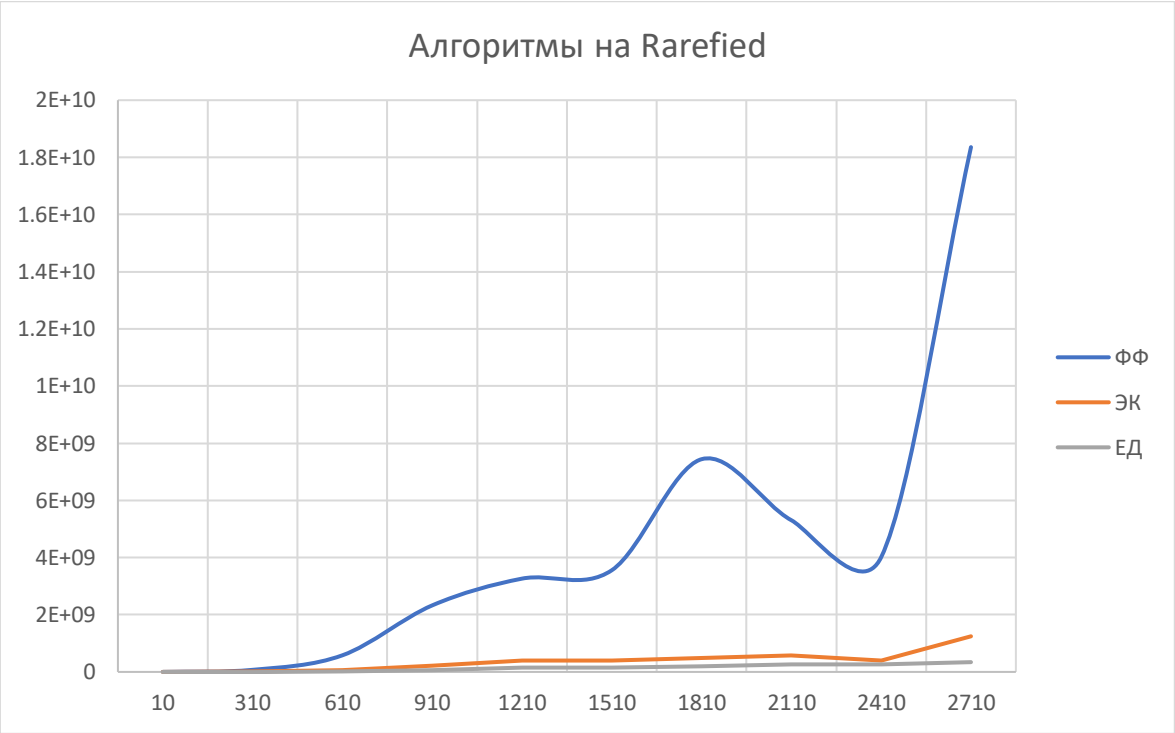

Ефим-Диниц/Rarefied
400000000
350000000
300000000
250000000
200000000
150000000
100000000
50000000
0
10
310
610
910
1210
1510
1810
2110
2410
2710

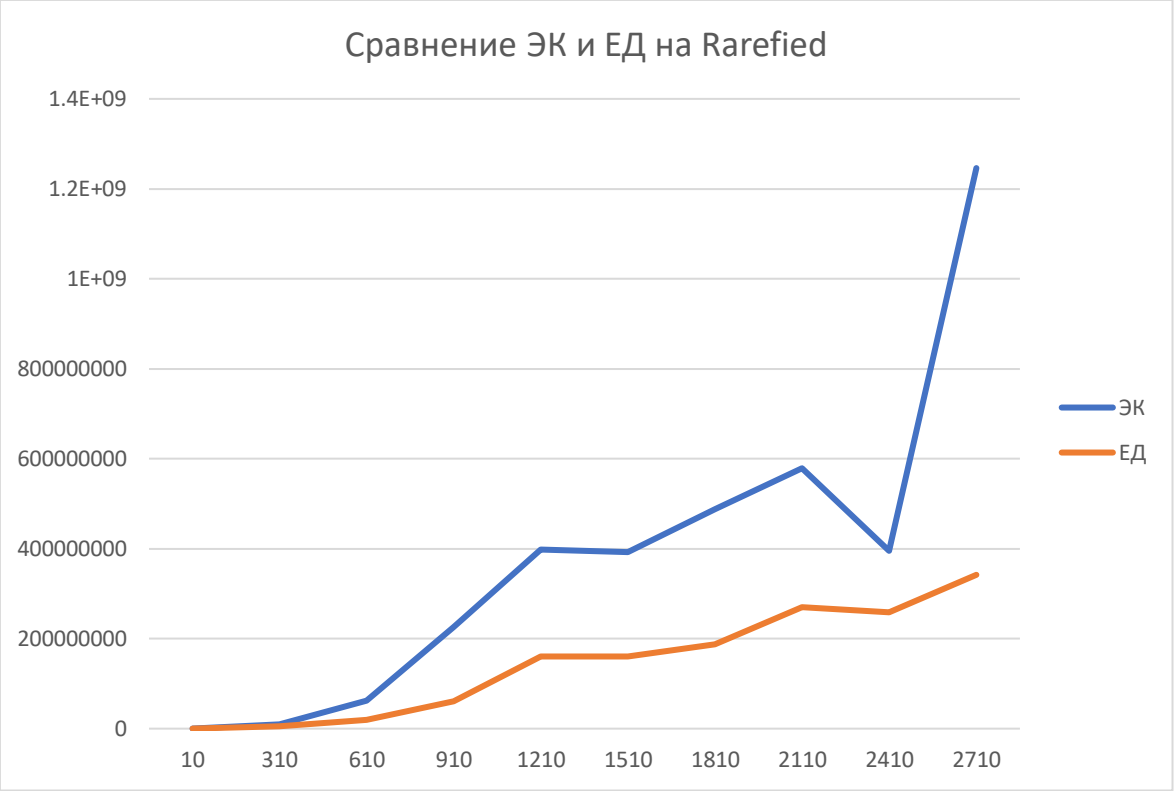

Ефим-Диниц/Medium
400000000
350000000
300000000
250000000
200000000
150000000
100000000
50000000
0
10
310
610
910
1210
1510
1810
2110
2410
2710

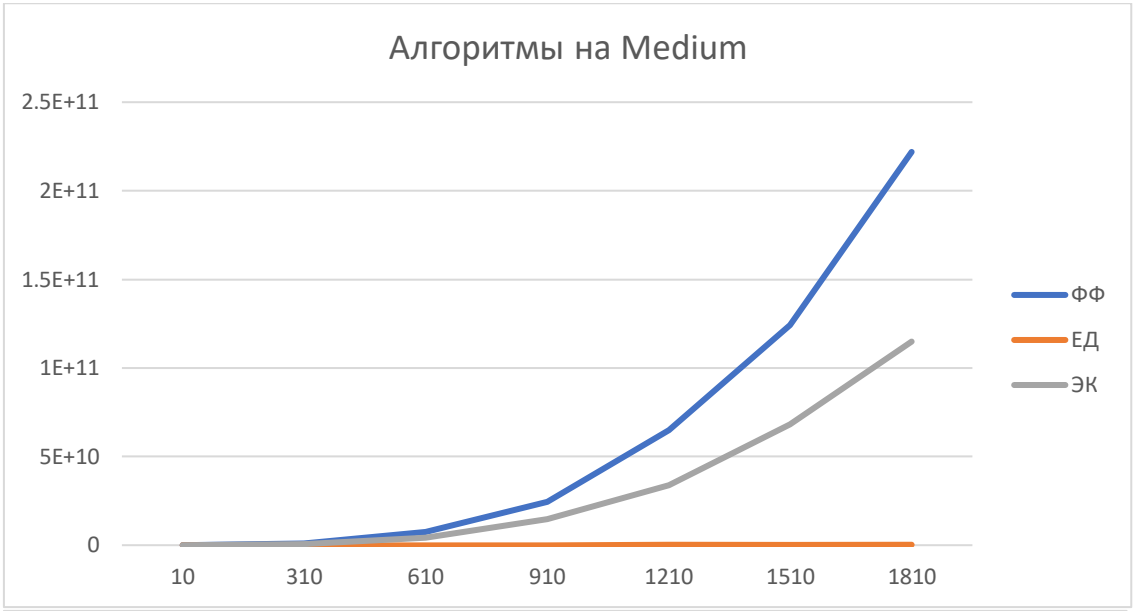

Ефим-Диниц/Tight
500000000
400000000
300000000
200000000
100000000
0
10
310
610
910
1210
1510
1810
2110
2410
2710

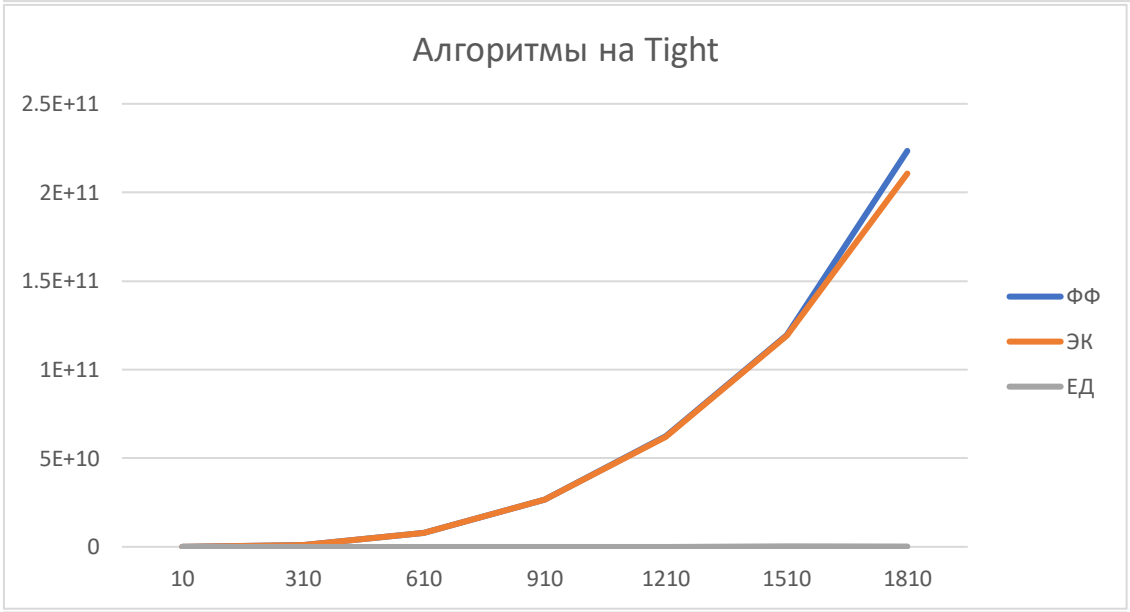

Ефим-Диниц/Disco
700000000
600000000
500000000
400000000
300000000
200000000
100000000
0
10
310
610
910
1210
1510
1810
2110
2410
2710

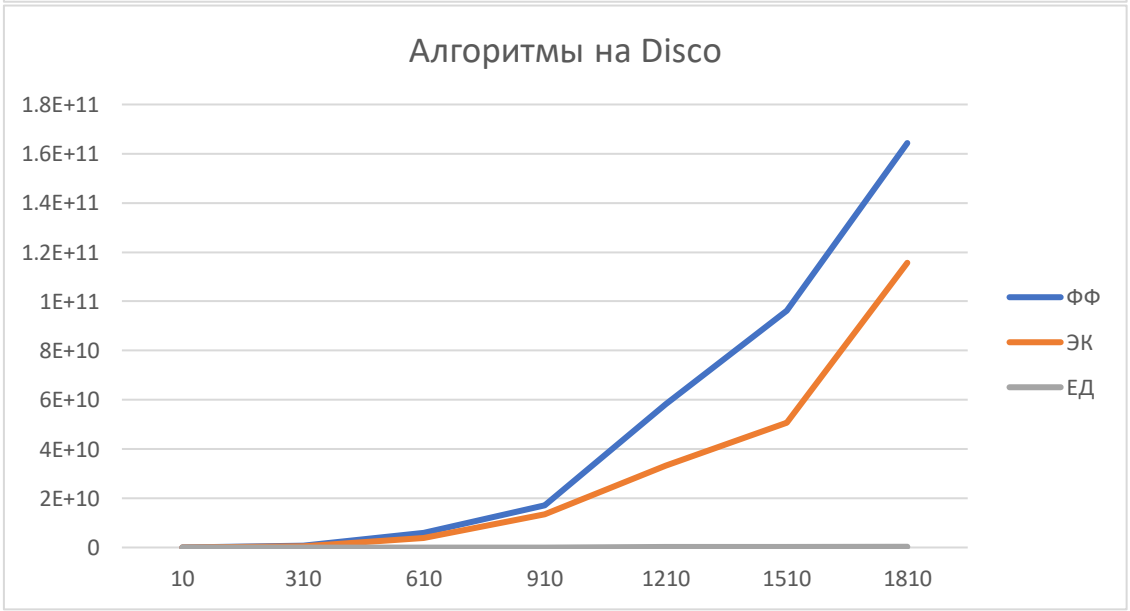
Имя файла	Форд-Фалкерсон (нс)	Эдмондс-Карп (нс)	Ефим-Диниц (нс)
Rarefied			
10	199400	153420	98543
310	73009600	10083777	5534222
610	5,82E+08	62839111	19865777
910	2,33E+09	2,25E+08	60846666
1210	3,26E+09	3,99E+08	1,6E+08
1510	3,56E+09	3,92E+08	1,6E+08
1810	7,44E+09	4,88E+08	1,87E+08
2110	5,31E+09	5,79E+08	2,7E+08
2410	4,01E+09	3,95E+08	2,59E+08
2710	1,84E+10	1,25E+09	3,42E+08
Medium			
10	0	0	0
310	1,07E+09	5,48E+08	5875888
610	7,47E+09	4,34E+09	22300777
910	2,46E+10	1,47E+10	71624444
1210	6,48E+10	3,38E+10	1,27E+08
1510	1,24E+11	6,81E+10	1,66E+08
1810	2,22E+11	1,15E+11	1,59E+08
Tight			
10	199000	0	111000
310	1,04E+09	1,02E+09	5427333
610	8,03E+09	7,97E+09	17479000
910	2,65E+10	2,65E+10	36403888
1210	6,25E+10	6,22E+10	63185000
1510	1,19E+11	1,19E+11	1,03E+08
1810	2,23E+11	2,11E+11	1,81E+08
Disco			
10	0	0	0
310	6,66E+08	4,76E+08	4545666
610	5,92E+09	3,81E+09	25745000
910	1,71E+10	1,34E+10	57753222
1210	5,82E+10	3,32E+10	1,3E+08
1510	9,63E+10	5,08E+10	1,51E+08
1810	1,64E+11	1,16E+11	2,92E+08


Алгоритмы на Rarefied
2E+10
1.8E+10
1.6E+10
1.4E+10
1.2E+10
1E+10
8E+09
6E+09
4E+09
2E+09
0
10
310
610
910
1210
1510
1810
2110
2410
2710
ФФ
ЭК
ЕД


Сравнение ЭК и ЕД на Rarefied
1.4E+09
1.2E+09
1E+09
800000000
600000000
400000000
200000000
0
10
310
610
910
1210
1510
1810
2110
2410
2710
ЭК
ЕД


Алгоритмы на Medium
2.5E+11
2E+11
1.5E+11
1E+11
5E+10
0
10
310
610
910
1210
1510
1810
ФФ
ЕД
ЭК


Алгоритмы на Tight
2.5E+11
2E+11
1.5E+11
1E+11
5E+10
0
10
310
610
910
1210
1510
1810
ФФ
ЭК
ЕД


Алгоритмы на Disco
1.8E+11
1.6E+11
1.4E+11
1.2E+11
1E+11
8E+10
6E+10
4E+10
2E+10
0
10
310
610
910
1210
1510
1810
ФФ
ЭК
ЕД

Сравнительный анализ алгоритмов

- На основе графиков сравнения каждого из алгоритмов на одном и том же типе файлов очевидно, что алгоритм Ефима-Диница не имеет конкурентов в скорости. Лишь на файлах *rarefied* этот алгоритм имеет скорость выполнения чуть меньше, чем у алгоритма Эдмондса-Карпа. На соседнем графике это удобно посмотреть.

- На *rarefied* графах алгоритм Форда-Фалкерсона заметно уступает обоим конкурентам.

- На *medium* графах Форд-Фалкерсон не сильно уступает Эдмондсу-Карпу.

- На *tight* графах Форд-Фалкерсон имеет практически такую же скорость, что и Эдмондс-Карп.

- На *disco* графах Форд-Фалкерсон немного уступает алгоритму Эдмондса-Карпа.

Источники

1. Схема алгоритма Диница
2. Схема алгоритма Форда-Фалкерсона
3. Схема алгоритма Эдмондса-Карпа