Отчет по проведению эксперимента

Гордиенко Егор, 371 группа, 02.11.2020

В ходе эксперимента был произведен анализ производительности CFPQ-запросов, реализованных через:

1. Алгоритм Хеллингса
2. Алгоритм с перемножением матриц
3. Алгоритм с тензорным произведением матриц (на входе не преобразованная грамматика)
4. Алгоритм с тензорным произведением матриц (на вход подавалась грамматика в ОНФХ)

В качестве исходных данных были взяты датасеты FullGraph, MemoryAliases, WorstCase.
К сожалению, на датасете SparseGraph замеры произвести не удалось, поскольку даже на самом маленьком по объему графе время работы алгоритма Хеллингса оказалось слишком большим. Скорее всего, алгоритм с перемножением матриц отработал бы гораздо быстрее, но для корректного сравнения хотелось получить данные о времени выполнения всех алгоритмов, поэтому данный датасет было решено не брать во внимание в этом эксперименте.

Замеры производились на компьютере со следующими характеристиками:
процессор Inter(R) Core i7-7700HQ CPU @ 2.80GHz, 16.0 Gb RAM DDR4, Ubuntu 18.04 (WSL) under Windows 10.

Запуск каждого алгоритма производился по 3 раза (кроме тех данных, где время, потраченное на выполнение алгоритма, оказывалось слишком большим). Для замеров использовалась библиотека time, данные округлялись до миллисекунд.

Также результаты работы всех алгоритмов сравнивались между собой, чтобы убедиться в их корректности.

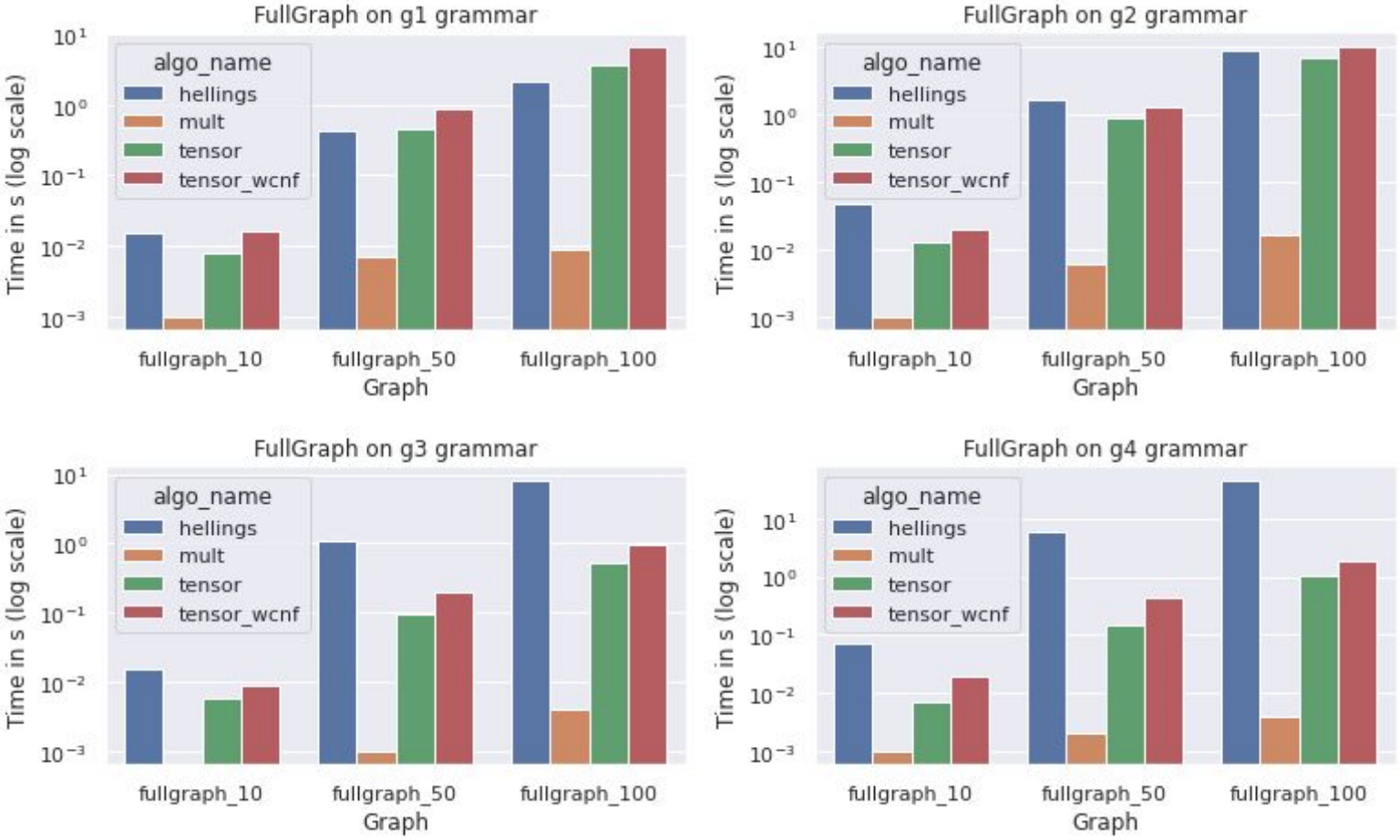
Данные для каждого датасета собирались в следующем формате:

	test	graph	grammar	algo_name	algo_time
0	FullGraph	fullgraph_10	g1	hellings	0.015
1	FullGraph	fullgraph_10	g1	mult	0.001
2	FullGraph	fullgraph_10	g1	tensor	0.008
3	FullGraph	fullgraph_10	g1	tensor_wcnf	0.016
4	FullGraph	fullgraph_10	g2	hellings	0.047

Здесь время указано в секундах, обозначения алгоритмов следующие:

- hellings - алгоритм 1.
- mult - алгоритм 2.
- tensor - алгоритм 3.
- tensor_wcnf - алгоритм 4.

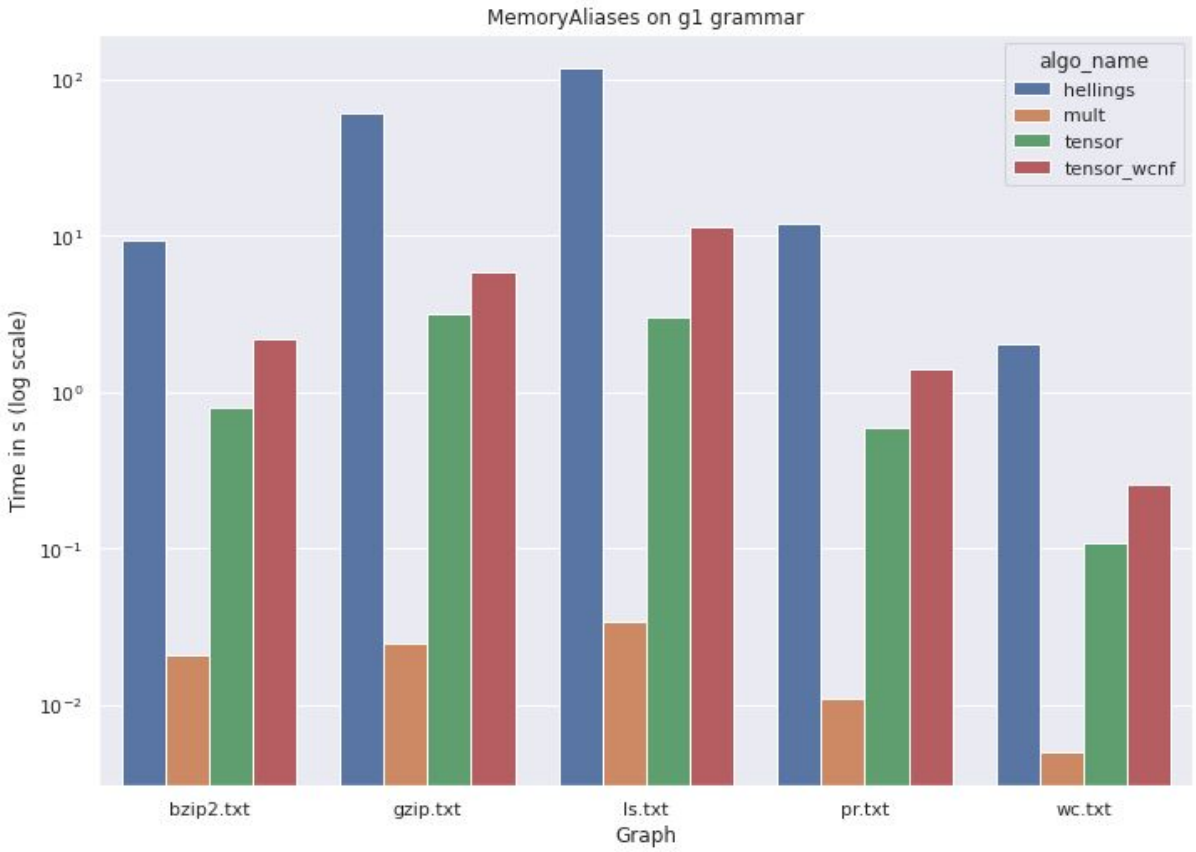
Рассмотрим результат работы алгоритмов на датасете **FullGraph**:

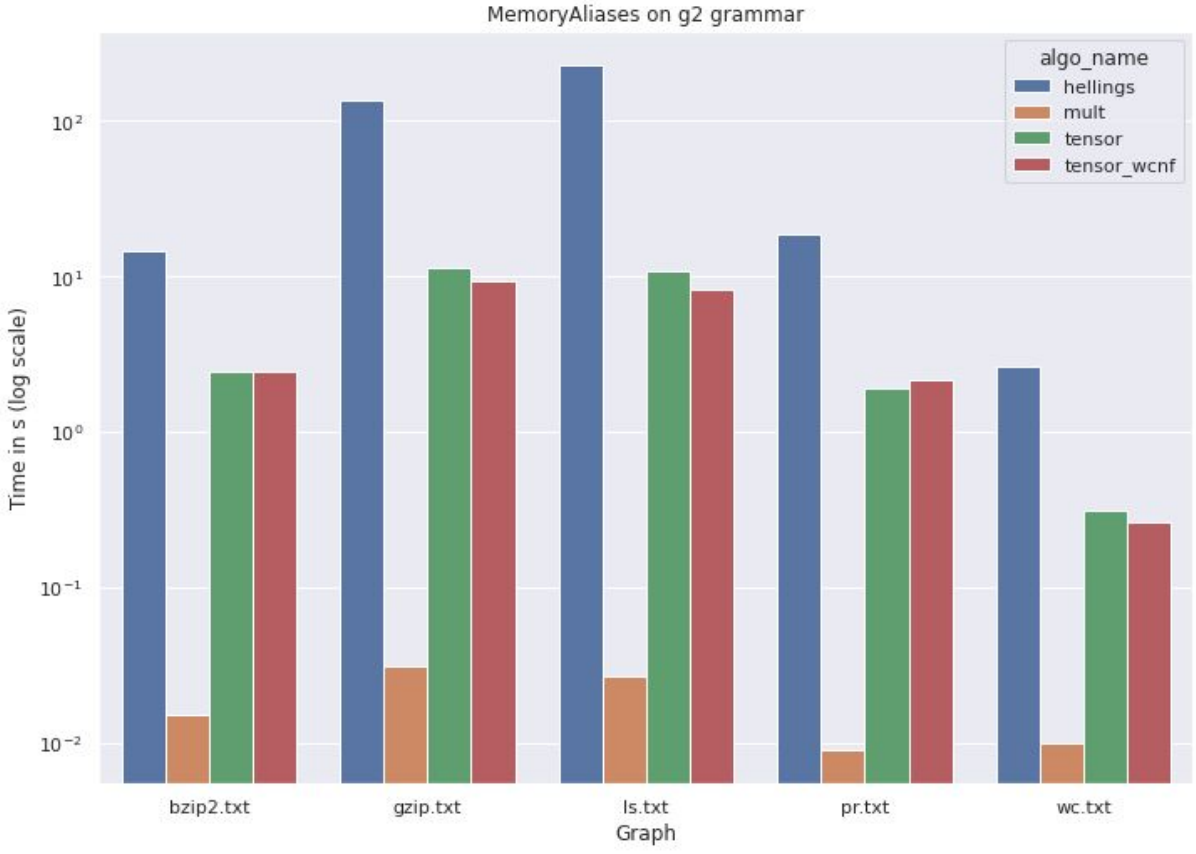


На данном датасете уже на графе fullgraph_200 не удалось дождаться, пока отработают все алгоритмы.
На графиках мы видим, что алгоритм с перемножением матриц отрабатывал в разы быстрее остальных. Самым медленным оказался алгоритм Хеллингса. Также можно отметить разницу, когда тензорному алгоритму на вход подается грамматика, не преобразованная и преобразованная в ОНФХ - время работы с ОНФХ немного больше, поскольку в этой форме размер грамматики увеличивается, что отрицательно влияет на производительность алгоритма.

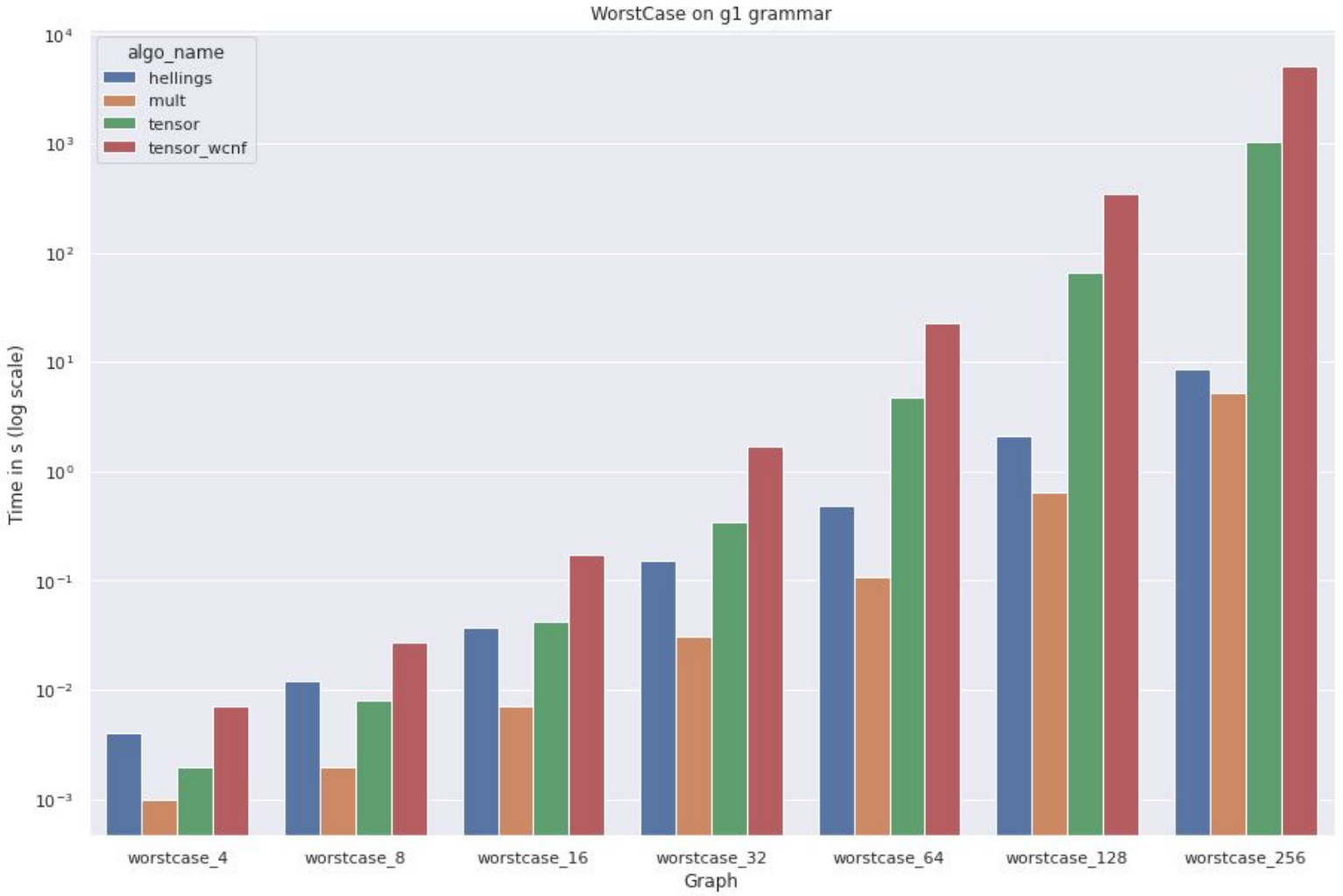
Результат работы на **MemoryAliases**:

Для начала посмотрим на график с грамматикой g1. Как можно увидеть ниже, на данном датасете алгоритм с произведением матриц все так же на порядок быстрее остальных. Алгоритм Хеллингса работает по-прежнему дольше остальных, и аналогично FullGraph, видна разница между тензорным алгоритмом с ОНФХ и без. Грамматика g2 же интересна тем, что правая часть тела продукции задается регулярным выражением. Из-за особенностей реализации считывания такой грамматики (большое число продукций, сравнимое с числом продукций в ОНФХ), разницы между тензорным алгоритмом с ОНФХ и без почти нет.


MemoryAliases on g1 grammar
algo_name
hellings
mult
tensor
tensor_wcnf
Time in s (log scale)
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
bzip2.txt
gzip.txt
ls.txt
pr.txt
wc.txt
Graph


MemoryAliases on g2 grammar
algo_name
hellings
mult
tensor
tensor_wcnf
Time in s (log scale)
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
bzip2.txt
gzip.txt
ls.txt
pr.txt
wc.txt
Graph

Результат работы на **WorstCase**:



Можно заметить, что с увеличением вершин в графе алгоритм с произведением матриц становится ненамного быстрее алгоритма Хеллингса. Это может быть связано с тем, что данный датасет представляет худший случай, когда преобразование в ОНФХ добавляет много новых продукций. Алгоритм Хеллингса же, в отличии от предыдущих замеров, оказался быстрее тензорного алгоритма. Все так же видна разница между тензорным алгоритмом с ОНФХ и без.

Вывод: матричный алгоритм (с перемножением) оказался самым быстрым на всех наборах данных. Этому может способствовать эффективная реализация перемножения разреженных матриц в библиотеке pygraphblas. Тензорный алгоритм везде уступает матричному, а также мы убедились, что алгоритм работает быстрее с грамматикой не в ОНФХ форме, так как его скорость зависит от размера грамматики. Алгоритм Хеллингса тоже медленнее матричного, при этом в некоторых случаях он заметно медленнее и тензорного алгоритма, а в других - опережает его. В любом случае, ни один из алгоритмов не показал большей эффективности, чем матричный, на каждом тесте, который удалось запустить.