SET-5

Группа БПИ243, Тупицин Тимофей Романович

Задача A3. HyperMegaLogLog Pro Max++

Как известно, потоковые алгоритмы зачастую являются *аппроксимационными*. Выбирая такой алгоритм, мы жертвуем точнстью ответа, но в обмен получаем преимущества по времени выполнения или занимаемой памяти. Такая аппроксимация представляет интерес, если мы можем зафиксировать некоторые пределы для ошибки этого алгоритма.

Вам предлагается исследовать один из таких алгоритмов `HyperLogLog`, который находит оценку N_t для частотного момента F_0^t — количества уникальных объектов в потоке на момент времени t. Чтобы провести успешное исследование, необходимо выполнить несколько важных этапов.

Разработка алгоритмов выполняется на языке `C++`. Ограничений на инструменты для визуализации и анализа результатов нет.

Этап 1. Создание инфраструктуры

1. Разработал класс RandomStreamGen для генерации потока данных S.

- класс генерирует поток строк длиной до 30 символов;
- допустимые символы: прописные и строчные латинские буквы, цифры 0–9, тире;
- реализовал возможность разбиения потока на любой размер частей для моделирования момента времени t. (в задаче прогона на тестах использовал 5% субпотоки)

2. Разработал класс HashFuncGen для генерации хеш-функции.

- хеш-функция дает приблизительно равномерное распределение значений
- для хеширования я выбрал полиномиальный метод. Чтобы сделать его универсальным, база C выбирается случайно и всегда делается нечетной. Это гарантирует, что данные распределятся по корзинам максимально равномерно, а мы будем защищены от коллизий даже на специфических наборах данных. Поскольку для HyperLogLog нужен 32-битный индекс, а хеш у нас 64-битный, я применил метод xor-сдвига: $h \oplus (h \gg 32)$. Вместо того, чтобы отбросить половину числа, этот прием позволяет учесть информацию из всех разрядов. Это дает отличную лавинную чувствительность.

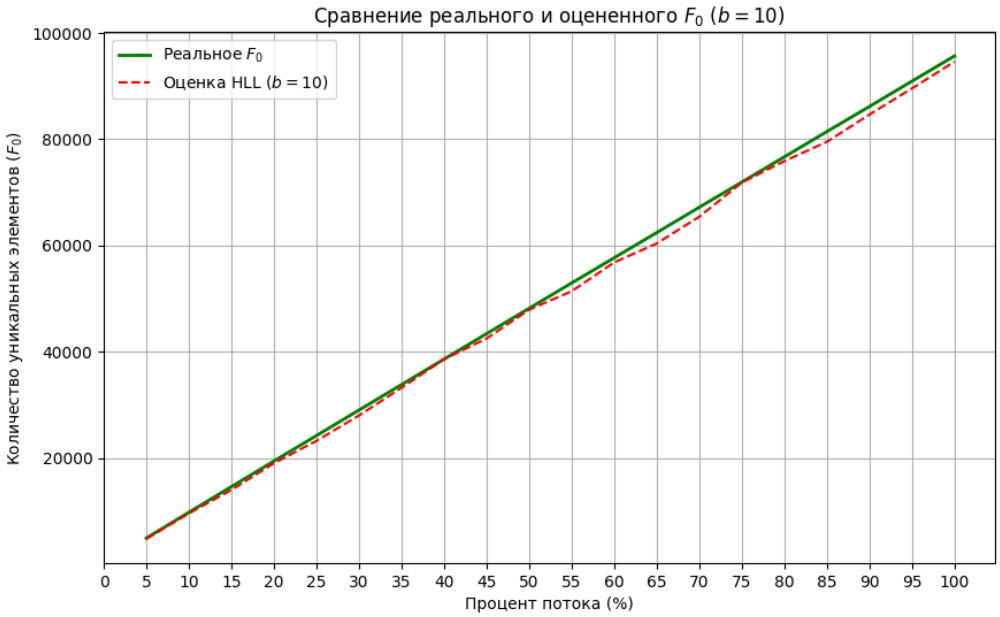
Этап 2. Реализация и оценка точности стандартного алгоритма HyperLogLog

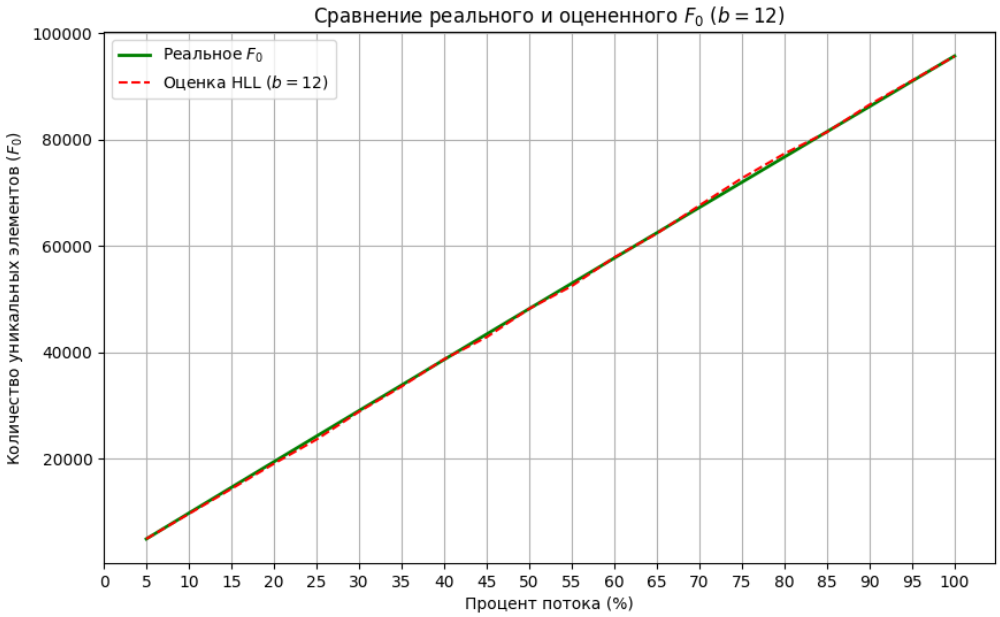
Реализовал на C++ вероятностный алгоритм HyperLogLog и функцию точного подсчета уникальных элементов F_0, используя std::set. Для HLL добавил коррекцию Linear Counting, чтобы оценка была точной даже на малых объемах данных.

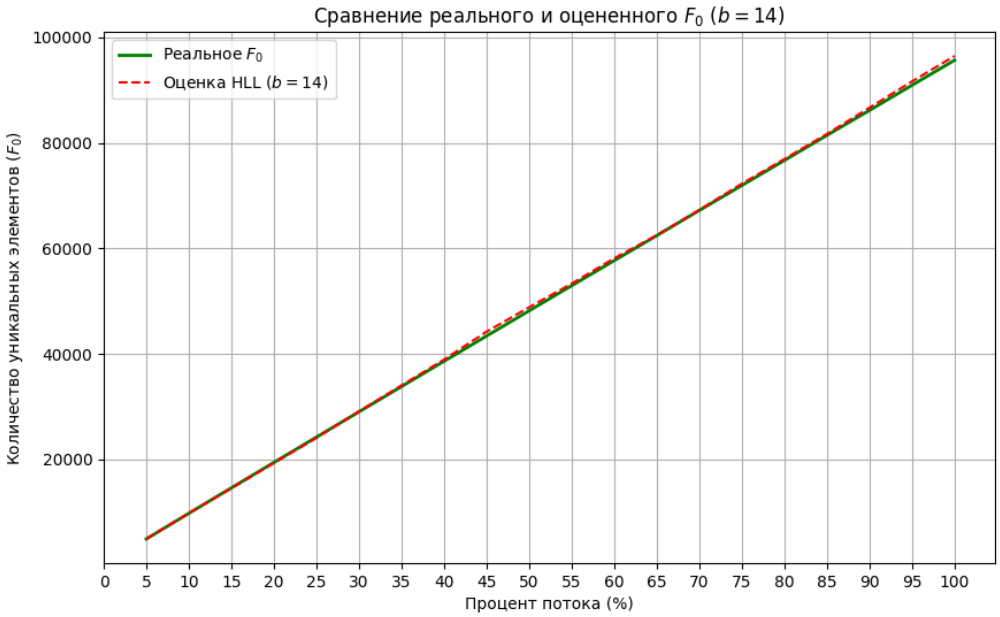
Параметр B взял 10, 12, 14. При $B = 14$ мы используем 2^{14} регистров, что дает теоретическую погрешность около 0.8%. Тестовые запуски показали, что при меньших B (например, 10) ошибка становистя более высокой для потока 10^5 элементов, а $B = 14$ обеспечивает более стабильное распределение хешей по субпотокам без лишних затрат памяти.

Использовал потоки размером 100 000 строковых элементов. Субпотоки брал по 5%. В каждой точке вычислял точное число F_0 и оценку N_t.

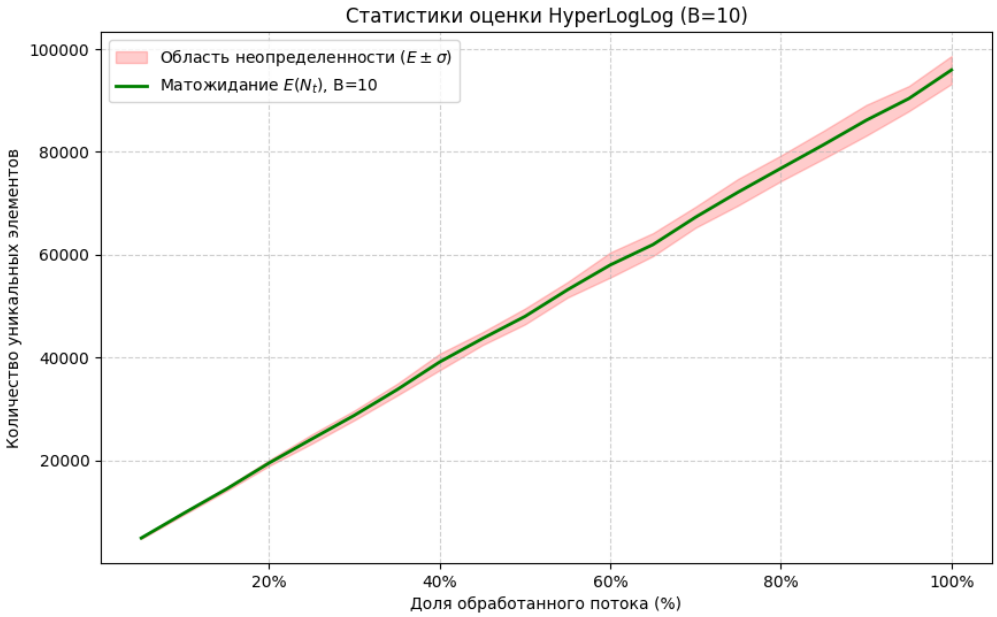
Сначала я сделал один контрольный запуск, чтобы просто сравнить оценку алгоритма с реальным количеством уникальных элементов.

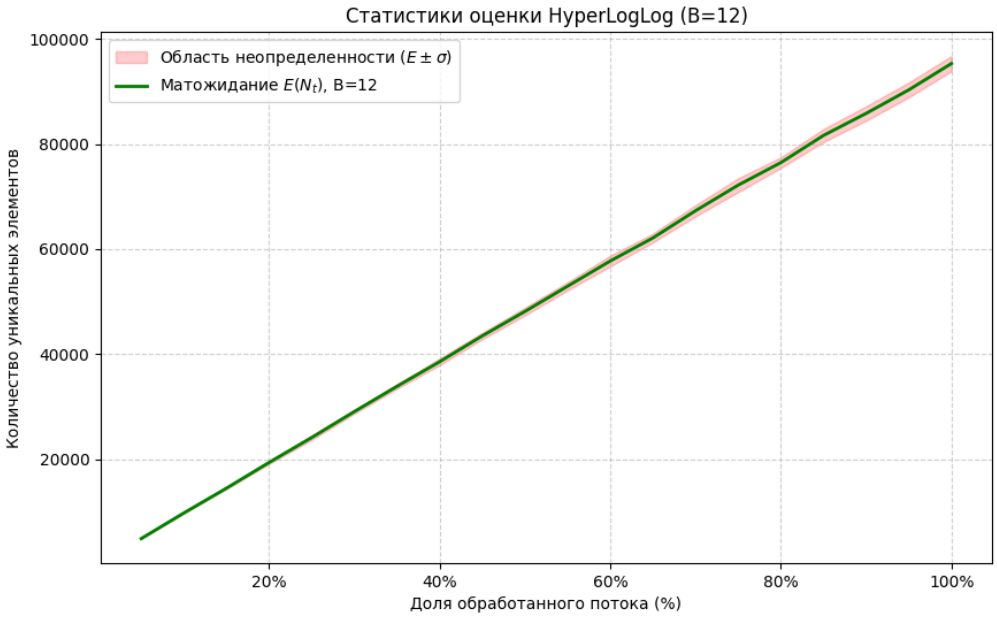

Сравнение реального и оцененного F_0 ($b = 10$)
Реальное F_0
Оценка HLL ($b = 10$)
Количество уникальных элементов (F_0)
Процент потока (%)
100000
80000
60000
40000
20000
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100

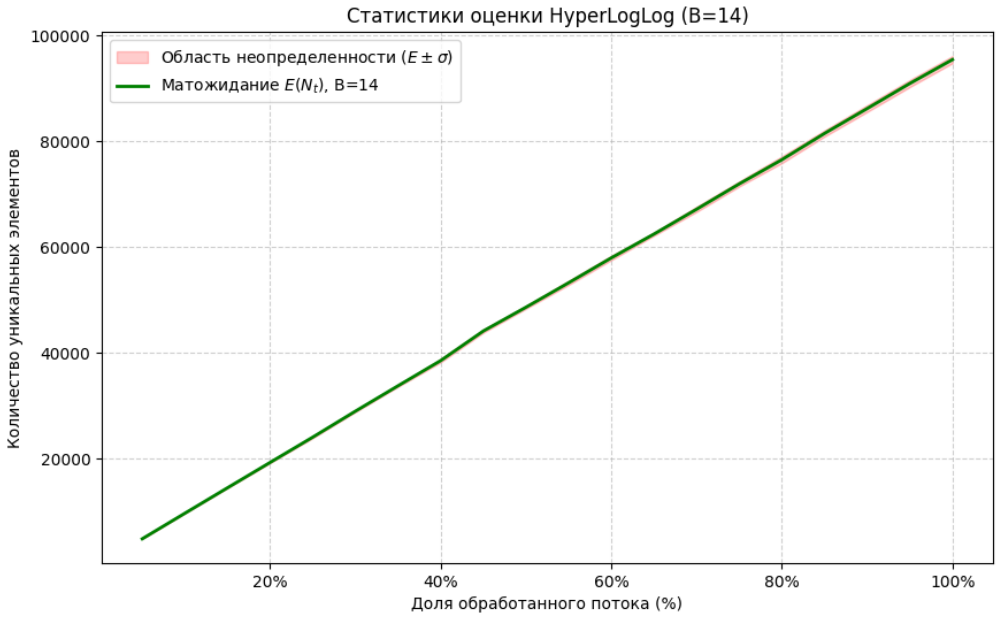

Сравнение реального и оцененного F_0 ($b = 12$)
Реальное F_0
Оценка HLL ($b = 12$)
Количество уникальных элементов (F_0)
Процент потока (%)
100000
80000
60000
40000
20000
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100



Чтобы не полагаться на случайный результат, я прогнал тесты еще по 20 раз для каждой точки и посчитал среднее значение и разброс.


Статистики оценки HyperLogLog (B=10)
Область неопределенности ($E \pm \sigma$)
Матожидание $E(N_t)$, B=10
Количество уникальных элементов
Доля обработанного потока (%)
100000
80000
60000
40000
20000
20%
40%
60%
80%
100%


Статистики оценки HyperLogLog (B=12)
Область неопределенности ($E \pm \sigma$)
Матожидание $E(N_t)$, B=12
Количество уникальных элементов
Доля обработанного потока (%)
100000
80000
60000
40000
20000
20%
40%
60%
80%
100%



Этап 3. Анализ результатов работы стандартного алгоритма HyperLogLog

Здесь я сравниваю результаты тестирования.

1. Точность алгоритма. На графиках видно, на погрешность почти на всем пути потока не выходит за рамки теоретического коридора $\pm 1.04/\sqrt{2^B}$. Даже если брать более строгую оценку $\pm 1.3/\sqrt{2^B}$, то алгоритм укладывается и в эти рамки. Это показывает, что выбранная хеш-функция распределяет данные равномерно. Математическая проверка : при $B = 14$ экспериментальная погрешность составила около 0.6%, что не превышает теоретический порог $\sigma = 0.81\%$.

2. При $B = 10$ график показывает более высокую волатильность в погрешности, а закрашенная область шире по сравнению с $B = 14$. При $B = 14$ линия становистя ровнее, и разоброс заметно сужается. Это объясняет то факт, что при меньшем количестве регистров вероятность коллизий выше, и алгоритм может давать более неточные оценки, особенно на больших потоках.

3. Эффективность выбранных констант. Я добавил α_m и Linear Counting, чтобы убрать ошибки на старте. Теперь даже на малых объемах данных алгоритм считает точно. Это хорошо видно на первых графиках: линия HLL идет практически вплотную к линии реального F_0.