SET 5. Капогузов Максим
Задача A3

A3. HyperMegaLogLog Pro Max++

Открыто с: суббота, 31 января 2026, 19:30
Срок сдачи: понедельник, 9 февраля 2026, 02:00

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. Разработайте класс **`RandomStreamGen`** для генерации потока данных S:
   - класс должен генерировать поток строк длиной до 30 символов;
   - допустимые символы: прописные и строчные латинские буквы, цифры 0–9, тире;
   - реализовать возможность разбиения потока на части (например, с шагом 5%, 10%, …) для моделирования момента времени t.
2. Разработайте класс **`HashFuncGen`** для генерации хеш-функции $h : U \to M = 2^{32}$, где U — это множество строк, из которых формируется поток данных S:

Этап 1. Создание инфраструктуры

1. Разработайте класс **`RandomStreamGen`** для генерации потока данных S:
   - класс должен генерировать поток строк длиной до 30 символов;
   - допустимые символы: прописные и строчные латинские буквы, цифры 0–9, тире;
   - реализовать возможность разбиения потока на части (например, с шагом 5%, 10%, …) для моделирования момента времени t.
2. Разработайте класс **`HashFuncGen`** для генерации хеш-функции $h : U \to M = 2^{32}$, где U — это множество строк, из которых формируется поток данных S:
   - хеш-функция должна давать приблизительно равномерное распределение значений;
   - рекомендуется использовать подходы из Лекций 14 и 16 (линейные/полиномиальные хеши) или адаптировать существующие: например, MurmurHash (в некоторых версиях лежит в основе **`std::hash`**), FNV или SHA;
   - дополнительно можно протестировать хеш-функцию на равномерность распределения.

Этап 2. Реализация и оценка точности стандартного алгоритма HyperLogLog

Реализуйте любым удобным способом вероятностный алгоритм **`HyperLogLog`** для вычисления оценки N_t, а также функцию, вычисляющую точное число F_0^t уникальных объектов в потоке. Для алгоритма **`HyperLogLog`** требуется самостоятельно выбрать размер индекса субпотока --- первые B бит (регистров). Кратко обоснуйте свой выбор B (для этого может понадобиться проведение нескольких тестовых запусков с разными значениями B для оценки того, как распределяются оставшиеся $L - B$ бит по субпотокам). Сгенерируйте несколько потоков (количество и размеры потоков выберите самостоятельно) и:

1. Для каждой выбранной вами части каждого потока вычислите точное число F_0^t уникальных объектов и оценку N_t.
2. Определите выборочные статистики для всех сгенерированных потоков: среднее значение оценки $\mathbb{E}(N_t)$, а также стандартное отклонение σ_{N_t}.

Постройте следующие графики для визуализации результатов анализа точности **`HyperLogLog`**:

1. **График №1** сравнения оценки N_t и F_0^t:
   - по оси X: номер шага (момент времени t) или размер обработанной части потока;
   - по оси Y: количество уникальных элементов*
   - соответственно, на этот график нанесите две линии: истинное значение F_0^t и оценка N_t.
2. **График №2** статистик оценки:
   - линия $\mathbb{E}(N_t)$ и
   - область неопределенности, которую можно представить в виде закрашенной области $\mathbb{E}(N_t) + \sigma_{N_t}$ и $\mathbb{E}(N_t) - \sigma_{N_t}$.

РЕШЕНИЕ

Этап 1

Генератор потока RandomStreamGen

Генерирует поток строк длиной от 1 до 30 символов.

Символы: латинские буквы и цифры, -

Поток делится на части по шагам времени t: 0.05, 0.10, …, 1.00

Хеш-функция HashFuncGen

Каждая строка переводится в 32-битное число

Используется быстрый хеш с хорошим перемешиванием битов, чтобы значения распределялись примерно равномерно.

Этап 2

В чем суть

Все хеши распределяются по m регистрам.

По каждому регистру запоминается насколько редкий шаблон битов встретился

В конце из состояния регистров вычисляется N(t).

Выбор параметра B

B - число бит для индекса регистра.

m = 2^B - число регистров.

В эксперименте выбрано: B = 14, значит m = 16384.

Память под регистры: 16384 байта

Параметры запуска

Длина потока: N = 200000

Число независимых потоков для статистики: 30

Шаг по времени: 5%

Что измерялось

Для каждого t:

считали точное F0(t)

считали оценку N(t) от HLL

по 30 потокам считали среднее E[N(t)] и разброс

Графики

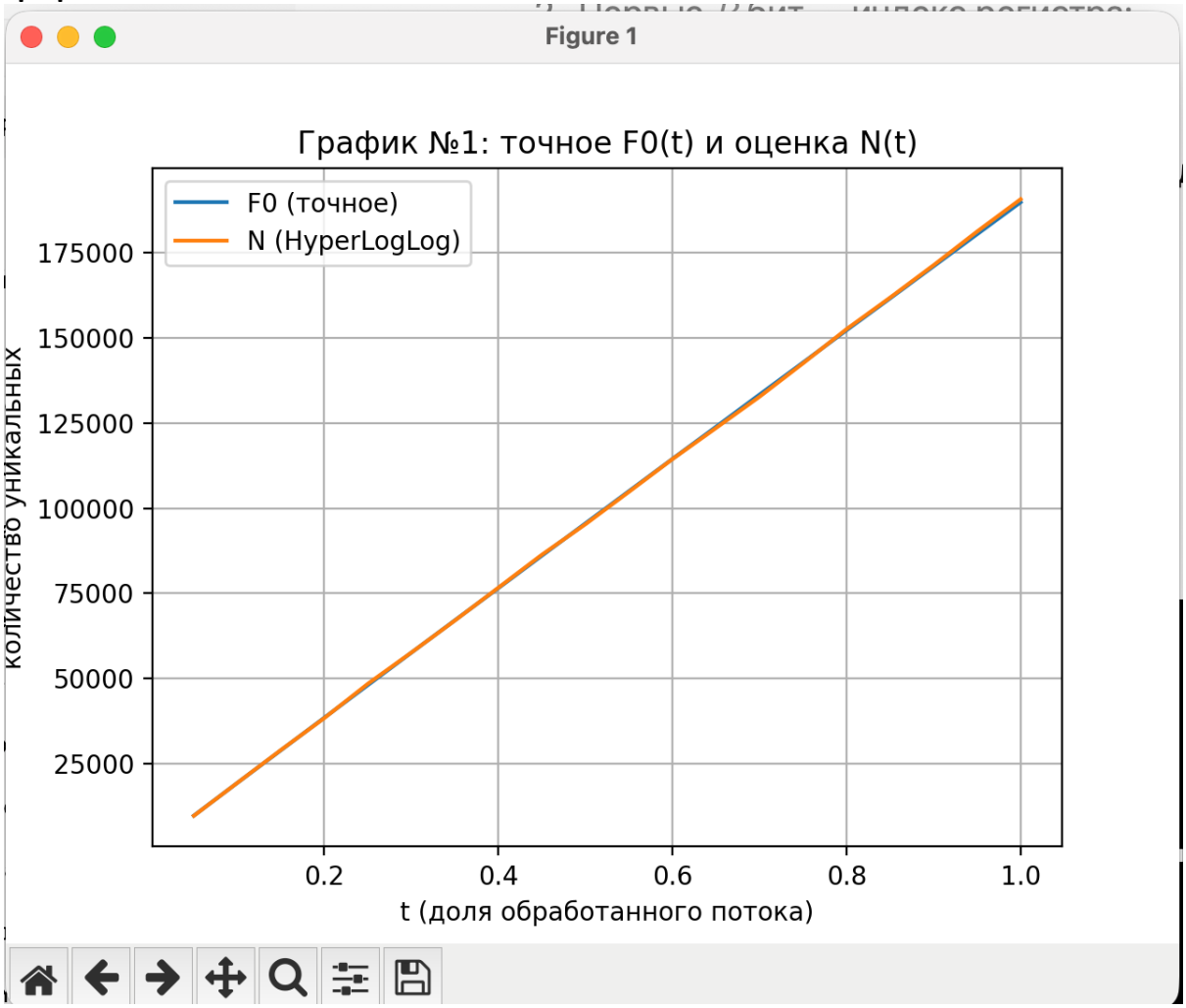

График №1: F0(t) и N(t) для одного потока

Линии почти совпадают на всем интервале t.

Это означает, что алго дает очень близкую оценку к истинному числу уникальных.

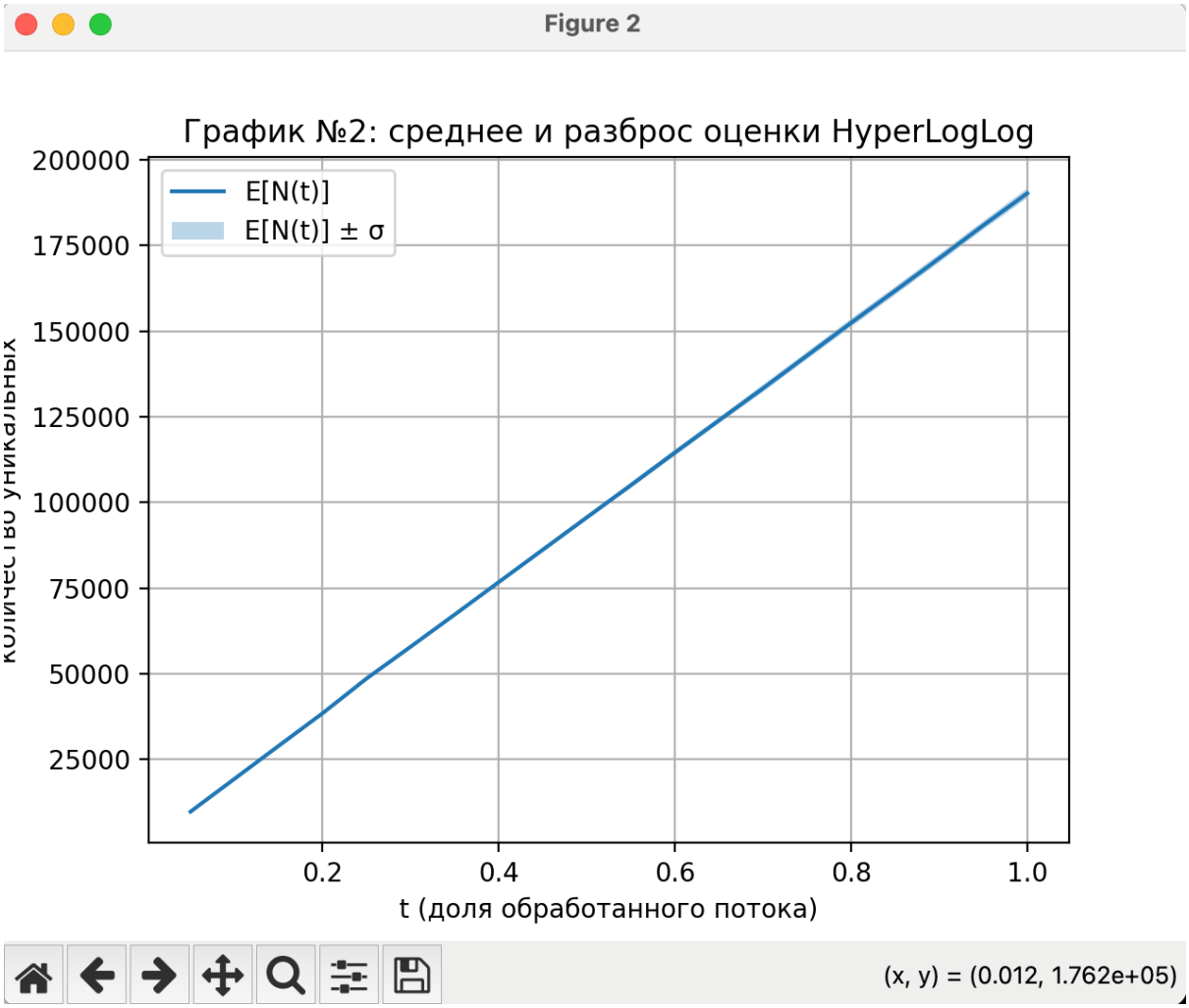

График №2:

Видно, что разброс вокруг среднего очень маленький, полоса почти не заметна

Это говорит о высокой стабильности оценки: разные случайные потоки дают очень похожие N(t)

Этап 3

Теоретические ориентиры ошибки

По выводу программы:

```
stream 30/30 done
Saved graph1.csv and graph2.csv
B=14, m=16384
Theoretical RSE ~ 1.04/sqrt(m) = 0.8125%
Wider bound     ~ 1.32/sqrt(m) = 1.03125%
Registers memory ~ 16384 bytes (+vector overhead)
```

Сравнение с графиками ==>

На графике №1 оценка N(t) практически не отличается от F0(t) визуально, то есть ошибка маленькая, ну это в целом видно как бы

На графике №2 область E[N(t)] ± дисперсия очень узкая, значит дисперсия тоже небольшая

С теорией тоже сходится при m = 16384 ожидается ошибка порядка примерно 1%, и на практике это так же, линии совпадают, разброс очень маленький

Стабильность оценки

Разброс дисперсии заметно меньше самого значения E[N(t)], и визуально выглядит очень небольшим даже к t = 1.0

Это означает, что алгоритм дает повторяемый результат на разных потоках при фиксированном B

Влияние выбранных констант и параметров

Выбор B = 14 оказался удачным компромиссом: памяти около 16 КБ хватает, чтобы получить высокую точность и низкий разброс

При меньшем B ожидалась бы заметно более толстая полоса разброса и более видимое расхождение с F0(t)

Качество хеширования также важно: если хеш распределен плохо, начнутся смещения и ухудшится точность. В нашем случае ухудшения не наблюдается, по графикам линии совпадают

Вывод

Алгоритм **HyperLogLog** при B = 14 (m = 16384) показал:

высокую точность: N(t) близко к F0(t) на всем диапазоне t

высокую стабильность: разброс оценок по потокам очень мал

соответствие теоретическим ожиданиям. ориентир ошибки ~ 0.7-1.1%