АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

ЛЕКЦИЯ 12

Фильтры Блума и HyperLogLog

Блум проверяет возможное членство, HyperLogLog оценивает число разных ключей.

12

ИЗ 15 ЛЕКЦИЙ



МОДУЛЬ 3 Динамическое программирование

1 КУРС / 2 АКАДЕМИЧЕСКИХ ЧАСА / PYTHON 3

Цель и содержание лекции

ЦЕЛЬ И РЕЗУЛЬТАТЫ ОБУЧЕНИЯ

различать проверку принадлежности и оценку числа уникальных элементов, выполнить вручную обновления двух вероятностных структур и интерпретировать их ошибки.

- Объяснять ложноположительный ответ фильтра Блума.
- Трассировать установку битов и обновление регистров.
- Выбирать структуру для нужного вероятностного запроса.

ПЛАН ЛЕКЦИИ

1. **Задачи принадлежности и подсчета уникальных элементов**
2. **Добавление и запрос в фильтре Блума**
3. **Регистр HyperLogLog и оценка мощности множества**
4. **Слияние потоков, сложность и выбор параметров**

Фильтр Блума помнит позиции, а не ключи

СХЕМА

	0	1	2	3	4	5	6	7
1. Добавили A: позиции 2 и 5	0	0	1	0	0	1	0	0
2. Запрос B: позиции 2 и 6	0	0	1	0	0	1	0	0
3. Запрос C: позиции 2 и 5	0	0	1	0	0	1	0	0

У B одна позиция равна 0: B точно не добавляли. У C обе позиции равны 1: ответ «возможно».

Два вопроса об одном потоке

01 ТЕМА

Пусть сервер записывает идентификаторы посетителей в порядке `a`, `b`, `a`, `c`, `b`. Для проверки, появлялся ли `c`, подходит точное множество `set`. Размер этого множества равен числу разных идентификаторов, то есть трем. Оба ответа точны, но для их получения нужно хранить все различные ключи. В потоке с миллионами значений память на точное множество растет вместе с числом уникальных элементов.

Фильтр Блума хранит компактное представление множества для запросов «встречался ли данный ключ?». Отрицательный ответ означает отсутствие ключа, положительный — возможное присутствие. HyperLogLog (HLL) оценивает, сколько разных ключей встретилось. Размер фильтра Блума подбирают по ожидаемому числу ключей и допустимой доле ложноположительных ответов. Для HLL число регистров определяет типичную погрешность оценки. Перед хешированием нужно определить, какие входы считаются одинаковыми. Например, система может считать `Alice` и `alice` одним идентификатором или двумя, в зависимости от правил источника.

Как работает фильтр Блума

Фильтр состоит из массива `m` битов и `k` хеш-отображений в индексы от `0` до `m - 1`. Изначально все биты нулевые. При вставке вычисляют `k` индексов и записывают в каждый единицу. Для запроса повторяют вычисление. Если среди нужных позиций есть ноль, ключ отсутствует. Если все позиции заняты, ключ мог быть вставлен. Ложноположительный ответ возникает, когда его биты установили другие ключи.

Рассмотрим `m = 8`, `k = 2`. В учебной трассировке значения индексов уже вычислены. После вставки `a` отмечены позиции `1` и `4`. После вставки `b` отмечены `4` и `6`. Массив от позиции `0` до `7` равен `[0, 1, 0, 0, 1, 0, 1, 0]`. Запрос `c` с позициями `(1, 6)` дает ложноположительный ответ. Запрос `d` с позициями `(1, 7)` дает отрицательный ответ. Вставленный `b` дает положительный ответ, потому что его позиции сохранились.

Как работает фильтр Блума

02 ПРОДОЛЖЕНИЕ

Ложноотрицательный результат исключен, пока вставка и запрос используют одинаковые параметры и отмеченные биты сохраняются. Обнуление позиции `4` при удалении `a` уничтожило бы след вставки `b`. Для удаления применяют специальные варианты, например счетный фильтр Блума, где позиции содержат счетчики. При удалении нужно знать, что ключ действительно был вставлен.

При модели независимых равномерных хешей и `n` разных ключах вероятность ложноположительного ответа приближенно равна `(1 - exp(-kn/m))**k`. Доля занятых битов растет, и при заполнении почти всех позиций фильтр перестает хорошо отсеивать отсутствующие ключи. Близкое к оптимальному значение `k` в этой модели — `(m/n)·ln(2)` с округлением до положительного целого. Частый сценарий применения — предварительная проверка запроса к диску. Ответ «точно отсутствует» экономит чтение, ответ «возможно присутствует» ведет к обычной точной проверке.

Как работает фильтр Блума

02 ПРИМЕР PYTHON

12_01.py

```
from hashlib import sha256


def positions(key, size, hashes):
    for number in range(hashes):
        payload = number.to_bytes(2, "big") + key.encode("utf-8")
        digest = sha256(payload).digest()
        yield int.from_bytes(digest[:8], "big") % size


size, hashes = 128, 3
bits = bytearray((size + 7) // 8)
for key in ("a", "b", "a", "c"):
    for index in positions(key, size, hashes):
        bits[index // 8] |= 1 << (index % 8)

for key in ("a", "b", "c", "d"):
    maybe = all(bits[index // 8] & (1 << (index % 8))
                for index in positions(key, size, hashes))
    print(key, maybe)
```

РЕЗУЛЬТАТ ЗАПУСКА

```
a True
b True
c True
d False
```

Как работает фильтр Блума

02 ПРОДОЛЖЕНИЕ

В примере каждый байт содержит восемь битов. Деление на восемь дает номер байта, остаток — позицию внутри байта. Разные префиксы задают воспроизводимые варианты хеширования. Для ключей фиксированной длины вставка и запрос выполняют `O(k)` операций, память под битовый массив составляет `O(m)` бит. Для строк переменной длины нужно дополнительно учитывать время чтения каждого ключа при вычислении хешей.

HyperLogLog обновляет максимум в каждом регистре

СХЕМА

Шесть битов хеша	Корзина	Ранг	Регистры 0, 1, 2, 3
Начальное состояние	—	—	[0, 0, 0, 0]
01 | 0010	1	3	[0, 3, 0, 0]
10 | 1000	2	1	[0, 3, 1, 0]
01 | 0001	1	4	[0, 4, 1, 0]

Первые два бита задают корзину. Ранг равен числу ведущих нулей в остатке плюс один.

Регистр 1 повысился с 3 до 4. Повтор тех же хешей не изменит регистры: максимум останется прежним.

Регистры HyperLogLog

Представим равномерный двоичный хеш. Вероятность, что он начинается с `r` нулей, равна `2**(-r)`. Длинная серия нулей встречается реже короткой. Если среди разных ключей появился длинный префикс из нулей, это дает основание предположить, что ключей было много. Один максимум слишком чувствителен к случайности. HLL делит хеши на `m = 2**p` корзин по первым `p` битам и хранит максимум в каждом регистре.

У оставшихся битов ранг равен числу ведущих нулей плюс один. Если весь остаток нулевой, ранг равен длине остатка плюс один. Начальное значение регистра равно нулю. Возьмем шесть битов хеша и `p = 2`. Хеш `01|0010` попадает в регистр `1` с рангом `3`. Хеш `10|1000` обновляет регистр `2` рангом `1`. Хеш `01|0001` повышает регистр `1` до `4`. Получаем `[0, 4, 1, 0]`. Повтор любого из трех хешей оставляет все максимумы прежними.

Регистры HyperLogLog

03 ПРОДОЛЖЕНИЕ

Базовая оценка имеет вид `alpha · m² / sum(2**(-R[j]))`, где `R[j]` — значение регистра, а `alpha` зависит от числа регистров. Через сумму обратных степеней двоек вычисление учитывает все регистры, уменьшая влияние одного аномально большого ранга. При небольшом числе элементов полезна поправка `m · ln(m/V)`, где `V > 0` — число пустых регистров. Когда все регистры пусты, эта оценка равна нулю. При мощности, сравнимой с числом возможных хешей, коллизии требуют отдельной обработки. Для объемов данных в учебном примере достаточно 64-битного хеша.

Регистры HyperLogLog

03 ПРИМЕР PYTHON

12_02.py / фрагмент 1 из 2

```
from hashlib import sha256
from math import log


def make_registers(keys, precision=8):
    registers = [0] * (1 << precision)
    suffix_width = 64 - precision
    suffix_mask = (1 << suffix_width) - 1
    for key in keys:
        digest = sha256(key.encode("utf-8")).digest()
        value = int.from_bytes(digest[:8], "big")
        bucket = value >> suffix_width
        suffix = value & suffix_mask
        rank = suffix_width - suffix.bit_length() + 1
        registers[bucket] = max(registers[bucket], rank)
    return registers
```

Регистры HyperLogLog

03 ПРИМЕР PYTHON

12_02.py / фрагмент 2 из 2

```
def estimate(registers):
    size = len(registers)
    alpha = 0.7213 / (1 + 1.079 / size)
    raw = alpha * size ** 2 / sum(2.0 ** (-rank) for rank in registers)
    empty = registers.count(0)
    if raw <= 2.5 * size and empty:
        return size * log(size / empty)
    return raw


first = [f"user-{i}" for i in range(700)]
second = [f"user-{i}" for i in range(400, 1000)]
merged = [max(a, b) for a, b in zip(make_registers(first),
                                    make_registers(second))]
assert merged == make_registers(first + second)
print("Точное количество:", len(set(first + second)))
print("Оценка:", round(estimate(merged)))
```

РЕЗУЛЬТАТ ЗАПУСКА

```
Точное количество: 1000
Оценка: 928
```

Регистры HyperLogLog

03 ПРОДОЛЖЕНИЕ

После округления оценка занижена на `72` ключа, или `7.2%`. Приближенное значение зависит от хеширования и параметров. Равенство, проверенное `assert`, относится к самим регистрам, поэтому выполняется точно. Для коэффициента `alpha` в таком виде используют `m ≥ 128`. В примере `p = 8`, значит `m = 256`.

Поток, память и точность

04 ТЕМА

Порегистровый максимум объединяет HLL двух потоков, если совпадают хеширование, кодирование и число регистров. Обычное сложение оценок завысило бы число при пересечении множеств. В примере первая часть содержит `700` ключей, вторая — `600`, а пересечение — `300`. После слияния дубликаты между частями влияют на регистры так же, как дубликаты внутри одной части.

При фиксированной длине ключа обновление HLL выполняется за `O(1)`, оценка и слияние — за `O(m)`. Память составляет `O(m)` регистров. Типичная относительная стандартная ошибка при подходящих предпосылках составляет около `1.04/√m`. При `m = 256` это около `6.5%`. Увеличение числа регистров в четыре раза уменьшает стандартную ошибку примерно вдвое. Это характеристика распределения ошибок, а конкретный запуск может дать большую ошибку. В компактной реализации каждый регистр занимает несколько битов. Список целых чисел Python из примера хранится менее плотно.

Что нужно уметь после лекции

ИТОГИ

- Фильтр Блума отвечает на запрос о конкретном ключе и может дать ложноположительный ответ.
- HyperLogLog оценивает число разных ключей по редким битовым шаблонам и допускает слияние совместимых сводок.
- Параметры структур связывают объем памяти с точностью разных типов ответов.

> Блум проверяет возможное членство, HyperLogLog оценивает число разных ключей.

Проверьте понимание

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Почему удаление бита может скрыть другой вставленный ключ?
2. Что происходит с фильтром Блума при многократном превышении ожидаемого числа ключей?
3. Почему повтор хеша не меняет регистр HyperLogLog после первого обновления?
4. При каких условиях можно объединить два массива регистров?

Самостоятельная работа

ПРАКТИКА

1. Для учебного фильтра из восьми битов перечислите пары различных позиций, для которых запрос сейчас даст положительный ответ.
2. Начните с регистров `[0, 4, 1, 0]`. Последовательно добавьте хеши `00|0011`, `10|0100` и `01|1111`. Запишите состояние после каждого шага.
3. Сравните ошибки оценки при `p = 8`, `p = 10` и `p = 12` на трех потоках одинаковой мощности с разными идентификаторами.

Источники и документация

ЧТЕНИЕ

ОСНОВНОЕ ЧТЕНИЕ

А. Бхаргава. Грокаем алгоритмы

СПб.: Питер, 2017. Глава 11. «Фильтры Блума и HyperLogLog». С. 263–266.

ДОПОЛНИТЕЛЬНО

Bloom B. H. Space/Time Trade-offs in Hash Coding with Allowable Errors. https://doi.org/10.1145/362686.362692

Flajolet P., Fusy É., Gandouet O., Meunier F. HyperLogLog: The Analysis of a Near-Optimal Cardinality Estimation Algorithm. https://dmtcs.episciences.org/en/articles/3545