Адрианов Н.М.
Иванов А.Б.

Алгоритмы и структуры данных

Хеш-таблицы. Распределенные хеш-таблицы. Фильтр Блума. Префиксное дерево. Отображенное дерево хеш-массива.

Кукушкино хеширование

- Используются 2 массива A_1, A_2 каждый размером $\frac{m}{2}$
- 2 хеш функции. $h_1, h_2 : U \to \left\{0, \ldots, \frac{m}{2} - 1\right\}$
- Каждый ключ $x \in U$ Может находится в одном из двух мест: в ячейке $h_1(x)$ в первом массиве или в ячейке $h_2(x)$ во втором массиве
- Если вставляем ключ x и место $h_1(x)$ занято ключом y, то "выкидываем"y, кладем x, а y перемещаем в $h_2(y)$ и далее по цепочке
- Если чепочка перемещений длиннее порого, выбираем новые хэш функции и перестраиваем всю таблицу
- Сложность поиска и удаления в худшем случае - $O(1)$
- Сложность вставки при достаточно низком коэффициенте заполнения в среднем - $O(1)$

Хэширование Робин Гуда

- Для каждого ключа x храним дополнительно расстояние до его "законного"места $h(x)$
- При вставке сравниваем расстояния, место достается ключу с большим текущим расстоянием
- Не улучшает теоретическую оценку, но уменьшает дисперсию

Распределенные хэш-таблицы

Объект $x \in U$
Ячейки (сервера): $S = \{S_1, \dots S_n\}$

Необходимо построить хеш-функцию

$$H : U \to S$$

Свойства:

- Равномерное распределение
- Минимальная перестройка при изменении S (добавлении/удалении элемента)

Применение

- Распределение нагрузки
- Распределённые хеш таблицы
- Распределённые БД

Консистентное хеширование (Consistent Hashing/Hash Ring)

- Представляем диапазон $[1 \dots 2^{32}]$ как окружность S^1
- Для каждого S_i генерим m случайных точек $s_{ij} \in S^1$
- Хеш функция $h : U \to S^1$
- Если ближайшая по часовой стрелке к $h(x)$ – точка $s_{kj} \in S^1$, то полагаем $H(x) = S_k$.

Rendezvous хеширование

- Хеш-функция

$$h : U \times S \to [1 \dots 2^{32}]$$

- Для $x \in U$ выбираем ячейку S_k, на которой хеш-функция принимает максимальное значение:

$$H(x) = \underset{s}{\operatorname{argmax}}\ h(x, s)$$

Другими словами

$$H(x) = k, \text{ если } h(x, S_k) \geq h(x, S_i)\ \forall i$$

Фильтр Блума (Bloom filter)

Вероятностая структура данных (множество). Операции:

- Вставка
- Поиск по ключу (бинарный результат)

vs Хеш-таблица:

- $\oplus$ память и скорость!!!
- $\ominus$ нет удалений
- $\ominus$ нельзя хранить значение
- $\ominus$ вероятность ложноположительного поиска

Фильтр Блума - конструкция

- Битовый массив B длины n (например, $n = 8|S|$)
- k хеш функций $h_1, \ldots, h_k$, $h_i : U \to \{1, \ldots, n\}$

Вставка ключа x: $B[h_i(x)] = 1$ для всех $i = 1, \ldots, k$

Поиск ключа x: "да" $\Leftrightarrow$ $B[h_i(x)] = 1$ для всех $i = 1, \ldots, k$

Фильтр Блума - анализ

В предположении равномерности и независимости хеш функций h_i: вероятность того, что конкретный бит установлен в 1 после вставки всего S, равна

$$1 - \left(1 - \frac{1}{n}\right)^{k|S|} \approx 1 - e^{-\frac{k|S|}{n}}$$

Вероятность ложноположительного поиска:

$$\left(1 - e^{-\frac{k|S|}{n}}\right)^k$$

Фильтр Блума - анализ

В предположении равномерности и независимости хеш функций h_i: вероятность того, что конкретный бит установлен в 1 после вставки всего S, равна

$$1 - \left(1 - \frac{1}{n}\right)^{k|S|} \approx 1 - e^{-\frac{k|S|}{n}}$$

Вероятность ложноположительного поиска:

$$\left(1 - e^{-\frac{k|S|}{n}}\right)^k$$

$$n/|S| = 8, \quad k = 2 \quad \longrightarrow \quad < 5\%$$

Фильтр Блума - анализ

В предположении равномерности и независимости хеш функций h_i: вероятность того, что конкретный бит установлен в 1 после вставки всего S, равна

$$1 - \left(1 - \frac{1}{n}\right)^{k|S|} \approx 1 - e^{-\frac{k|S|}{n}}$$

Вероятность ложноположительного поиска:

$$\left(1 - e^{-\frac{k|S|}{n}}\right)^k$$

$$n/|S| = 8, \quad k = 2 \quad \longrightarrow \quad < 5\%$$

$$n/|S| = 9, \quad k = 6 \quad \longrightarrow \quad \approx 2\%$$

Фильтр Блума - анализ

В предположении равномерности и независимости хеш функций h_i: вероятность того, что конкретный бит установлен в 1 после вставки всего S, равна

$$1 - \left(1 - \frac{1}{n}\right)^{k|S|} \approx 1 - e^{-\frac{k|S|}{n}}$$

Вероятность ложноположительного поиска:

$$\left(1 - e^{-\frac{k|S|}{n}}\right)^k$$

$$n/|S| = 8, \quad k = 2 \quad \longrightarrow \quad < 5\%$$

$$n/|S| = 9, \quad k = 6 \quad \longrightarrow \quad \approx 2\%$$

$$n/|S| = 16, \quad k = 11 \quad \longrightarrow \quad < 0.05\%$$

Фильтр Блума - анализ

Вероятность ложноположительного поиска:

$$\left(1 - e^{-\frac{k|S|}{n}}\right)^k$$

Фильтр Блума - анализ

Вероятность ложноположительного поиска:

$$\left(1 - e^{-\frac{k|S|}{n}}\right)^k$$

При фиксированном $n/|S|$ как функция от k достигает минимума при

$$k^* = \frac{n}{|S|} \ln 2$$

Фильтр Блума - анализ

Вероятность ложноположительного поиска:

$$\left(1 - e^{-\frac{k|S|}{n}}\right)^k$$

При фиксированном $n/|S|$ как функция от k достигает минимума при

$$k^* = \frac{n}{|S|} \ln 2$$

$$n/|S| = 9, \quad k^* \approx 6.24, \quad k = 6 \quad \longrightarrow \quad \approx 2\%$$

$$n/|S| = 16, \quad k^* \approx 11.09, \quad k = 11 \quad \longrightarrow \quad < 0.05\%$$

Фильтр Блума - использование

`https://en.wikipedia.org/wiki/Bloom_filter`

Akamai Technologies (CDN) - to prevent "one-hit-wonders"from being stored in its disk caches. One-hit-wonders are web objects requested by users just once, something that Akamai found applied to nearly three-quarters of their caching infrastructure. Using a Bloom filter to detect the second request for a web object and caching that object only on its second request prevents one-hit wonders from entering the disk cache, significantly reducing disk workload and increasing disk cache hit rates.

Google Bigtable, Apache HBase and Apache Cassandra and PostgreSQL - to reduce the disk lookups for non-existent rows or columns.

Google Chrome web browser - to identify malicious URLs

Microsoft Bing (search engine) uses multi-level hierarchical Bloom filters for its search index, BitFunnel. Bloom filters provided lower cost than the previous Bing index, which was

Trie (префиксное дерево)

Структура придумана Rene de la Briandais в 1959.

TRIE = центральные буквы слово reTRIEval. Термин придуман Edward Fredkin в 1960, обыгрывая схожесть со словом tree. Автор термина произносил `[tri:]`, но общепринято произношение `[trai]`, чтобы как раз не путать на слух.

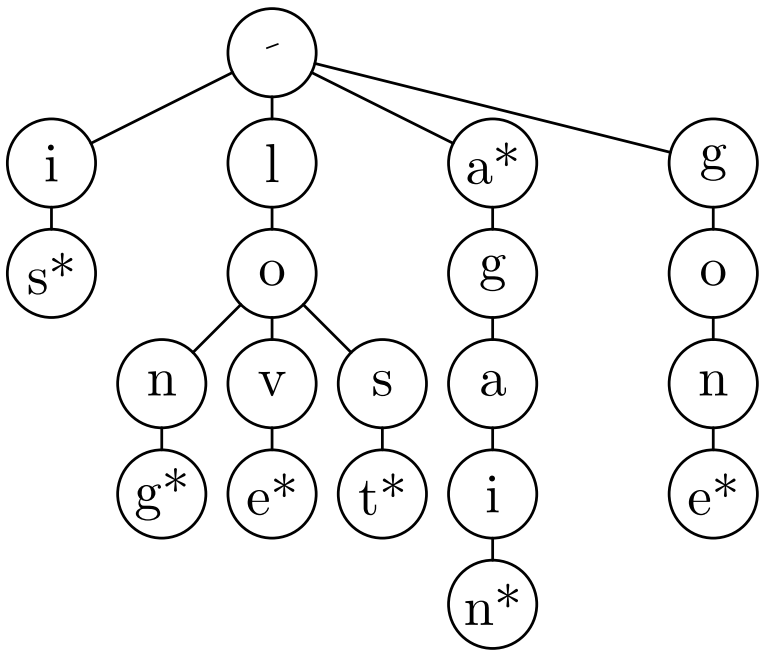
В русских переводах встречаются варианты

- БОР = выБОРка (Кнут, 3й том, 1 издание)
- ЛУЧ = поЛУЧение (Кнут, 3й том, 2 издание)
- нагруженное дерево (Ахо Ульман)

Trie (префиксное дерево)

Структура похожа на дерево поиска с парой отличий:

- в узлах храним буквы, а не строки;
- количество детей может быть до R (количество букв в алфавите).



Отображенное дерево хеш-массива (HAMT)

Структтура придумана Phil Bagwell в 2000.

В определенном смысле гибрид хеш-таблицы и префиксного дерева:

- Считаем 64-битную хеш-функцию
- Рассматриваем значение хеш-функции как слово из 6-битных букв
- Строим префиксное дерево