

# Вероятностные структуры данных: когда приблизительно очень даже хорошо

---

Марат Ахин

26 ноября 2018 г.

Санкт-Петербургский политехнический университет

- Если у вас много данных, у вас много вопросов
  - Есть ли у нас уже такой элемент?
  - Сколько разных элементов у нас есть?
  - Какие элементы встречаются чаще всего?

- Если у вас много данных, у вас много проблем
  - Как анализировать?
  - Как хранить?

- Если у вас много данных, у вас много проблем
  - Как анализировать?
  - Как хранить?

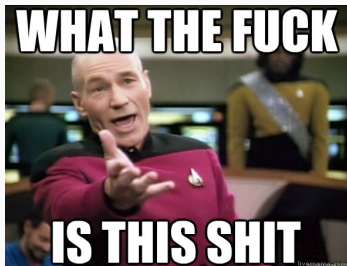
*Изи ваще, все в хэш-таблицу и посчитаем! (с) анон*



- Допустим, мы хотим анализировать Интернет
  - $10^9$  уникальных сайтов
  - 4 Гб данных
  - Очень грубая оценка снизу

*Да у меня в ноуте оперативы больше! (с) анон*

- Есть ли у нас уже такой элемент?
  - Наше решение?
    - Массив? (4 Гб)
    - Список? (12 Гб)
    - Хэш-таблица? (16-40 Гб)



- Можем ли мы сделать лучше?..



- Можем ли мы сделать лучше?..



- ...если не ослабим ограничения на задачу



- Есть ли у нас уже такой элемент?
  - Иногда можно ошибаться

*Это же почти как хэш-функция?!*

- Массив бит размера  $m$
- Хэш-функция  $hash(e) \in [0, m)$

- Есть ли у нас уже такой элемент?
  - Иногда можно ошибаться

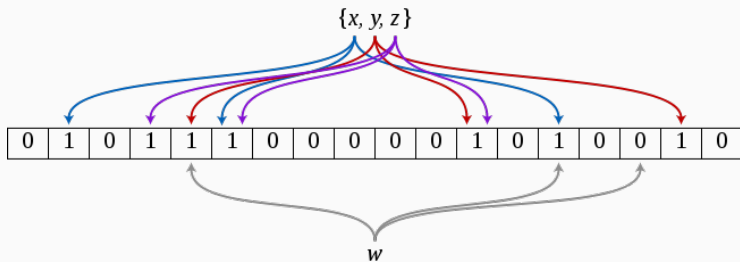
*Это же почти как хэш-функция?!*

- Массив бит размера  $m$
- Хэш-функция  $hash(e) \in [0, m)$

*А что с ошибками?*

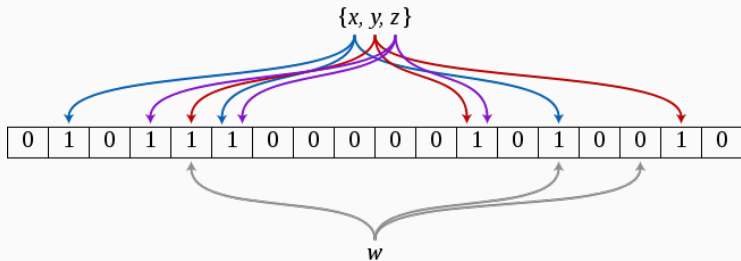
## Bloom filter

- А если взять не одну, а много хэш-функций?
- Массив бит размера  $m$
- $k$  хэш-функций  $hash_i(e) \in [0, m)$



# Bloom filter

- А если взять не одну, а много хэш-функций?
- Массив бит размера  $m$
- $k$  хэш-функций  $hash_i(e) \in [0, m)$



*А что с ошибками?*



$$m = -n \frac{\ln \delta}{\ln^2 2}$$

$$k = \frac{m}{n} \ln 2$$

*Все равно, наверное, много памяти надо! (с) анон*

- $\delta = 5\%$ 
  - $m \approx 6.2 \times 10^9 \approx 0.75\text{GB}$
  - $k \approx 4$
- $\delta = 10\%$ 
  - $m \approx 0.9 \times 10^9 \approx 0.11\text{GB}$
  - $k \approx 3$

- А что с удалением элементов?

- А что с удалением элементов?
- Простой ответ: его нет
- Сложный ответ: два фильтра Блума лучше, чем один
  - Один фильтр на добавление
  - Второй фильтр на удаление
    - А что с повторным добавлением?

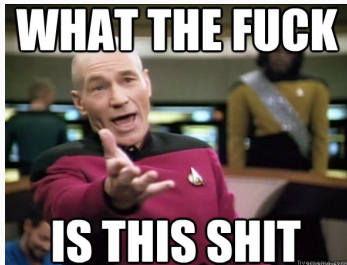


- А что с удалением элементов?
- Простой ответ: его нет
- Сложный ответ: два фильтра Блума лучше, чем один
  - Один фильтр на добавление
  - Второй фильтр на удаление
    - А что с повторным добавлением?
    - Простой ответ: его нет
    - Сложный ответ: ...



## Exact cardinality estimation

- Сколько разных элементов у нас есть?
  - Наше решение?
    - Массив? (4 Гб)
    - Список? (12 Гб)
    - Хэш-таблица? (16-40 Гб)



- Сколько разных элементов у нас есть?
  - Плюс-минус 5 процентов

*Хм, разные элементы с вероятностью ошибки...*

*Где же я это только что слышал?..*

- Фильтр Блума может использоваться для оценки мощности множества

$$n' = \frac{m}{k} \ln \left( 1 - \frac{Ones}{m} \right)$$

- Сколько разных элементов у нас есть?
  - Плюс-минус 5 процентов

*Хм, разные элементы с вероятностью ошибки...*

*Где же я это только что слышал?..*

- Фильтр Блума может использоваться для оценки мощности множества

$$n' = \frac{m}{k} \ln \left( 1 - \frac{Ones}{m} \right)$$

*А что с ошибками?*

- Допустим, что мы рассматриваем равномерно распределенные случайные числа
  - $1/2$  будет иметь вид  $1xxx \dots xxx$
  - $1/4$  будет иметь вид  $01xx \dots xxx$
  - $1/8$  будет иметь вид  $001x \dots xxx$
  - ...
- Получается, что позиция старшей единицы может служить для очень грубой, но оценки мощности множества наших чисел

*Подождите, тут один элемент решает все?!*

- Разобьем наше множество на  $m$  подмножеств, а потом объединим результаты

$$e = x_n x_{n-1} \dots x_{b+1} x_b \dots x_2 x_1$$

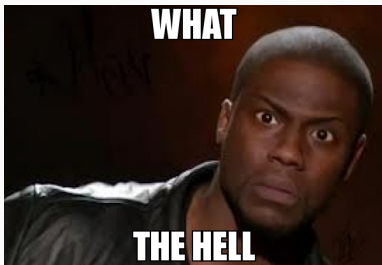
$$j = x_b \dots x_1$$

$$p = \text{leftmost\_one}(x_n \dots x_{b+1})$$

$$M(j) = \max(M(j), p)$$

Как объединять результаты для подмножеств?

$$Z = \frac{1}{\sum_{j=1}^m 2^{-M(j)}}$$





- Давайте посчитаем...

The background is a light green field filled with various mathematical formulas and diagrams, including:

- Algebraic identities:  $a^{-n} = \frac{1}{a^n}$ ,  $(ab)^n = a^n b^n$ ,  $a^m \times a^n = a^{m+n}$ ,  $\left(\frac{a}{b}\right)^n = \frac{a^n}{b^n}$ ,  $\frac{a^m}{a^n} = a^{m-n}$ ,  $a^2 - b^2 = (a+b)(a-b)$ ,  $\frac{a^m}{a^n} = a^{m-n}$ .
- Geometry:
  - Circles:  $C = 2\pi r$ ,  $A = 2\pi r^2$ .
  - Triangles:  $A = \frac{1}{2}bh$ .
  - Rectangles:  $A = \frac{1}{2}bh$ .
  - Cubes:  $A = 2\pi r^2$ .
- Diagrams: Circles with radius  $r$ , triangles with base  $b$  and height  $h$ , and a 3D cube.

- В итоге, получаем следующее выражение для оценки мощности множества

$$n' = m^2 Z$$



- В итоге, получаем следующее выражение для оценки мощности множества

$$n' = \alpha m^2 Z$$

*А что с точностью и памятью? Все же плохо?! (с) анон*

$$\sigma \approx \frac{1.04}{\sqrt{m}}$$

- $3\sigma = 5\%$ 
  - $m \approx 3894$
  - Каждый регистр размера 8 бит (можно и меньше)
  - Итого: 4 Кб
- $\sigma = 5\%$ 
  - $m \approx 433$
  - Каждый регистр размера 8 бит (можно и меньше)
  - Итого: 433 байта

- $3\sigma = 5\%$ 
  - $m \approx 3894$
  - Каждый регистр размера 8 бит (можно и меньше)
  - Итого: 4 Кб
- $\sigma = 5\%$ 
  - $m \approx 433$
  - Каждый регистр размера 8 бит (можно и меньше)
  - Итого: 433 байта

*Подождите, там что-то было про равномерно распределенные...*

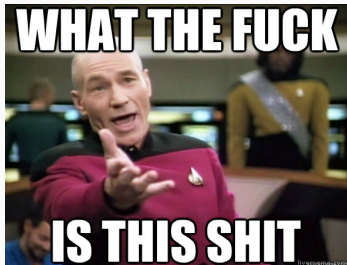
- Хорошие хэш-функции порождают значения, близкие к равномерно распределенным случайным числам
- Берем элементы исходного множества и пропускаем через хорошую хэш-функцию
- ...
- PROFIT!





## Exact frequency estimation

- Какие элементы встречаются чаще всего?
  - Наше решение?
    - Массив? (4 Гб)
    - Список? (12 Гб)
    - Хэш-таблица? (16-40 Гб)





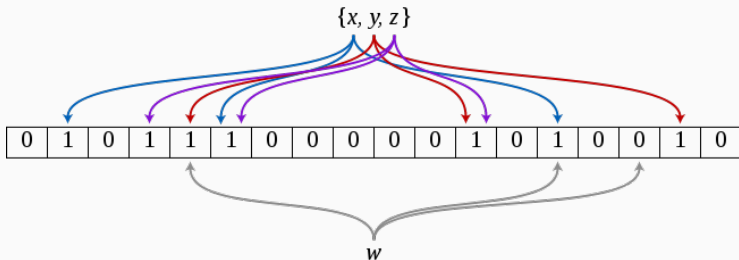
- Какие элементы встречаются чаще всего?
  - Иногда можно ошибаться

*Стоп, опять фильтр Блума? Он же как раз определяет, встречали ли мы элемент или нет...*

- Фильтр Блума не умеет считать, сколько раз встретился элемент

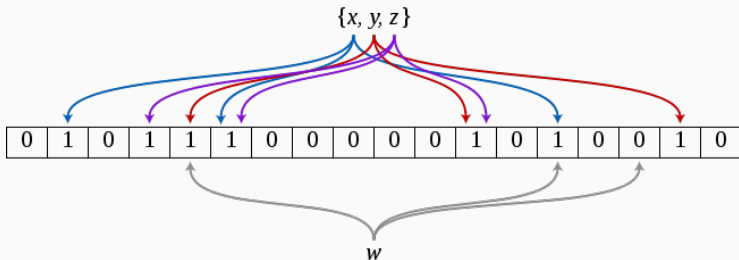
## Frequency estimation

- Давайте вместо массива бит использовать массив счетчиков



## Frequency estimation

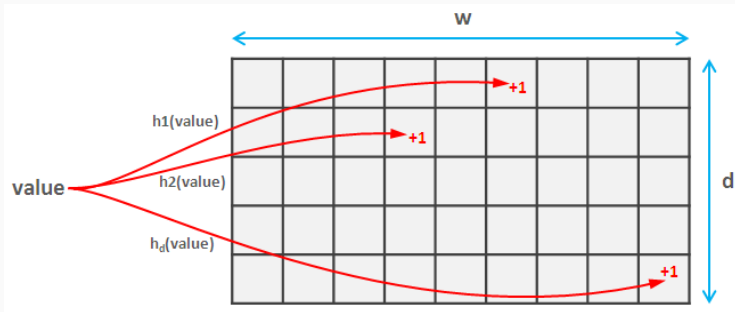
- Давайте вместо массива бит использовать массив счетчиков



*А что с ошибками?*

# Count-min sketch

- Давайте вместо одного массива использовать  $d$  массивов



- Оценка для элемента — минимум из всех его ячеек
- Почему минимум?

$$w = \frac{e}{\epsilon}$$

$$d = \ln \frac{1}{\delta}$$

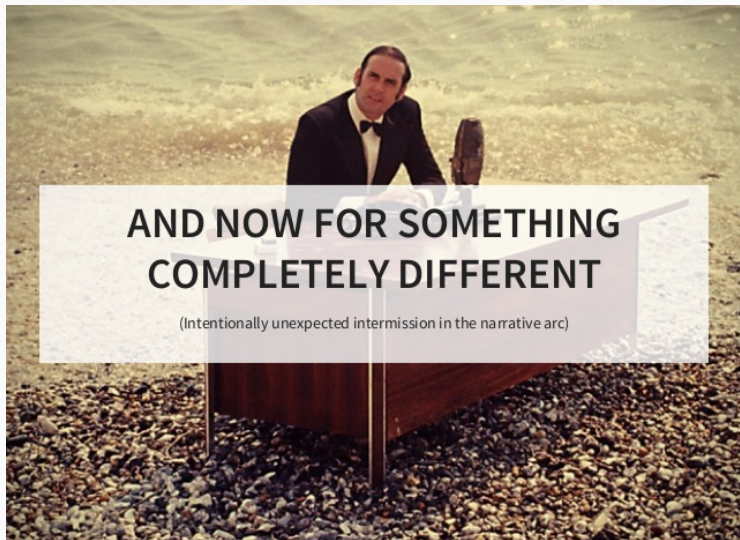
- $\epsilon = 5\%, \delta = 1\%$ 
  - $w \approx 55$
  - $d \approx 5$
  - Итого: 2200 байт
- $\epsilon = 1\%, \delta = 5\%$ 
  - $w \approx 270$
  - $d \approx 3$
  - Итого: 6480 байт

$$w = \frac{e}{\epsilon}$$

$$d = \ln \frac{1}{\delta}$$

- $\epsilon = 5\%, \delta = 1\%$ 
  - $w \approx 55$
  - $d \approx 5$
  - Итого: 2200 байт
- $\epsilon = 1\%, \delta = 5\%$ 
  - $w \approx 270$
  - $d \approx 3$
  - Итого: 6480 байт

*Ой-вей, а как узнать с помощью этой структуры данных, кто был чаще всего?*



## А что еще можно сделать с вероятностью?

- Сделать случайный список!..



- ...хорошо, не совсем уж случайный



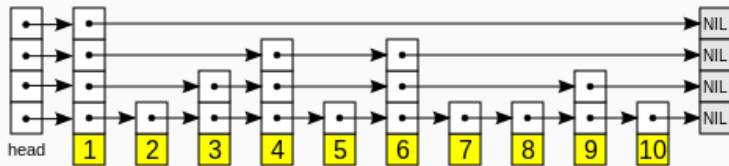
## Чем плох обычный список?

- Очень плохо работает на доступ к произвольному элементу
- Потому что нет быстрого способа добраться до какого-либо элемента



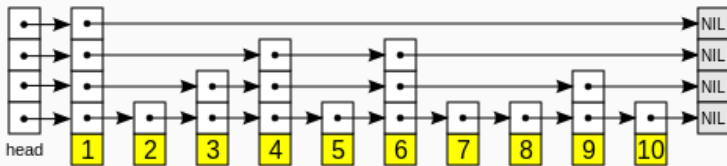
# Skip list

- Сделаем “магистрالی” вперед, в обход обычной последовательности



# Skip list

- Сделаем “магистраль” вперед, в обход обычной последовательности



- $\log n$  уровней
- Константное количество шагов на каждом уровне
- Итого: сложность операций  $O(\log n)$

*А я еще индексироваться люблю, а тут фиг! (с) анон*

- Добавим к каждой магистрали то, сколько элементов вперед она перескакивает
- Тривиально поддерживается при операциях со списком

*А что еще классного в skip-листе?*

- Хорошо обрабатывается параллельно
- Может обрабатываться вообще без блокировок (lock-free)

