

# Bloom filter

---

*Григорьев Юрий, ИС-142*

# Начнём с простого

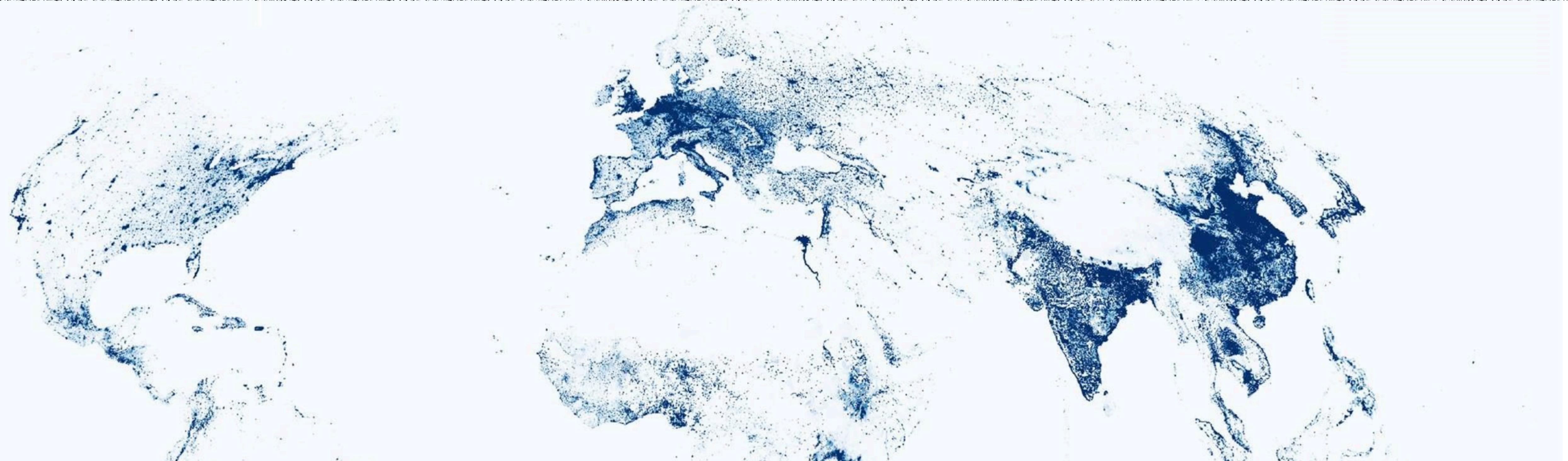
## «Города» (игра):

Игроки друг за другом называют города мира на последнюю букву прошлого названного города

Если никто не оспаривает названный игроком город (не сомневается в том, что такой существует), игра продолжается

Проигрывает (вылетает из игры) тот, кто в течение долгого времени не может назвать в свой ход следующий город

Карта, отражающая плотность населения Земли. Каждая синяя точка — населённый пункт (потенциально — город)



# *Игра в «Города»*

- знает небольшое количество городов
- не может быстро проверить, существует ли город, названный соперником

**Woz**



**VS**

- в памяти огромный массив существующих городов

- по их подобию может выдумать любое название города

**Ryotr**



# *Игра в «Города»*

- изобретательный программист  
с ЭВМ (например, смартфоном)  
в кармане
- ходячая энциклопедия

**Ryotr**



**Woz**

**VS**



# **Как проверить наличие ключа в словаре?**

( используя ЭВМ с производительностью 1000 операций / с )

*всего – 2 700 000 городов*

## **Линейный поиск**

- Занимает время  $O(N)$

*44 минуты на поиск одного города*

## **Хеш-таблица**

- Занимает время  $O(1)$

— Малоэффективна  
при частых коллизиях

— Требуется большой  
объём памяти



Woz

## **Бинарный поиск**

- Занимает время  $O(\log N)$

*0,02 с на поиск города*

+

- Нужно один раз  
отсортировать все элементы

Пример (Quick Sort – avg. case) :

Занимает время  $O(N * \log N)$

*16 часов*

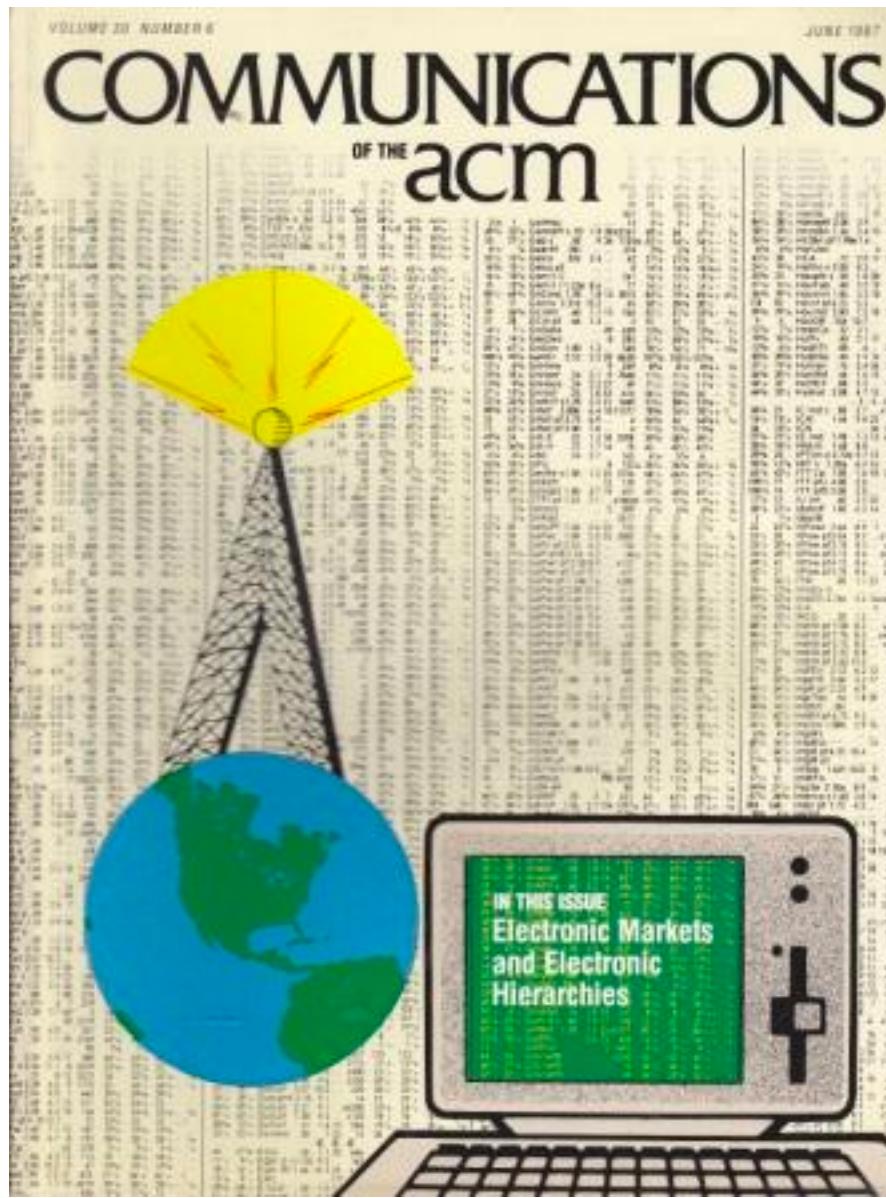


# Чего мы хотим от структуры данных / алгоритма?



1. Нужно проверить только отсутствие элемента в массиве данных
2. Желательно не размещать все объекты в оперативной памяти
3. Время выполнения операций должно быть константным (  $O(1)$  )
4. Можем пойти на риск и в редких случаях получить ложноположительный результат  
(в пользу пространственной и временной сложностей)

**Есть ли что-то подобное?**



и вот к нам на помощь приходит...

## Фильтр Блума!

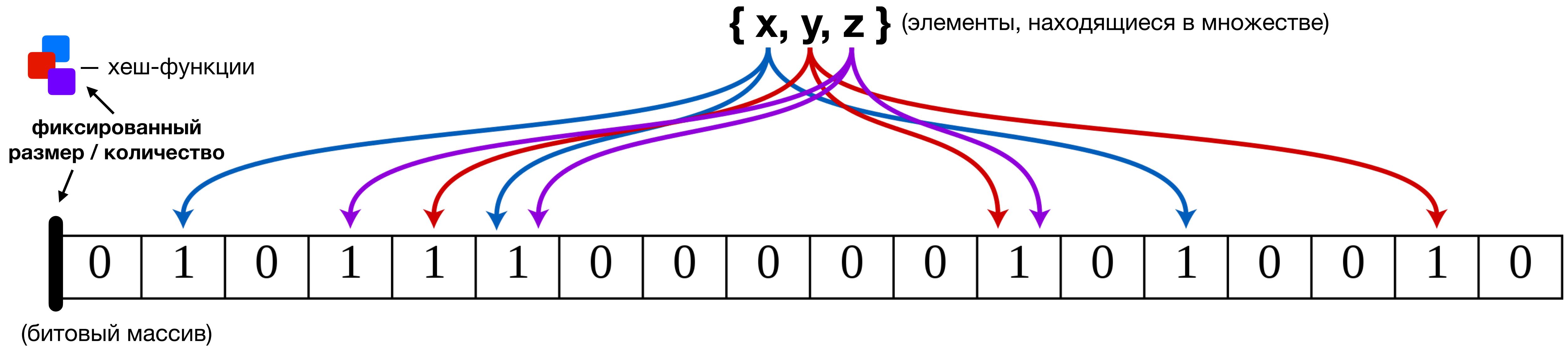
Придуман **Говардом Бертоном Блумом** в **1970** году,  
описан им же в выпуске "*Space/Time Trade-offs in Hash Coding  
with Allowable Errors*" журнала "*Communications of the ACM*"



1. Битовый массив  
фиксированного размера

2. Хеш-функции  
фиксированного количества

# Фильтр Блума



Как **добавить** элемент в фильтр?

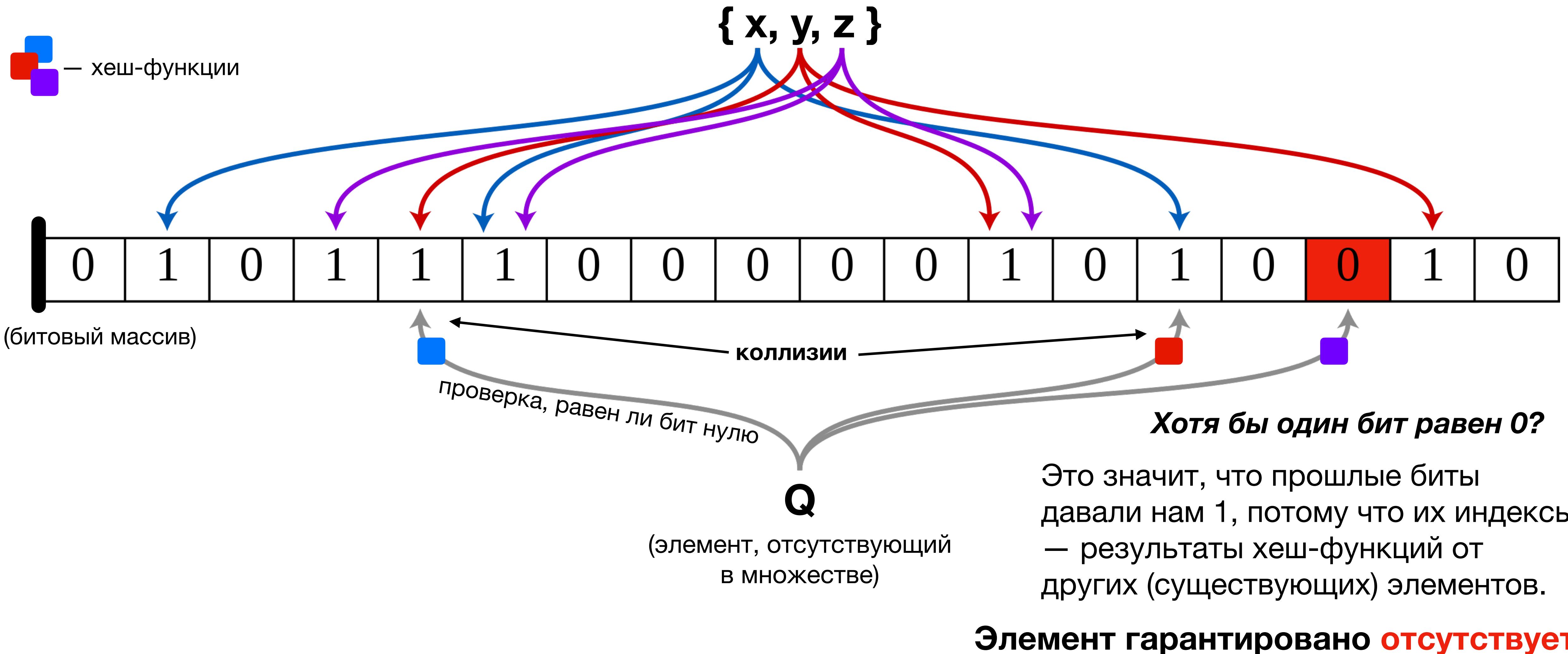
Хеш-функции получают  
индексы для элемента  
в массиве битов



Биты на соответствующих  
позициях приравниваются к 1

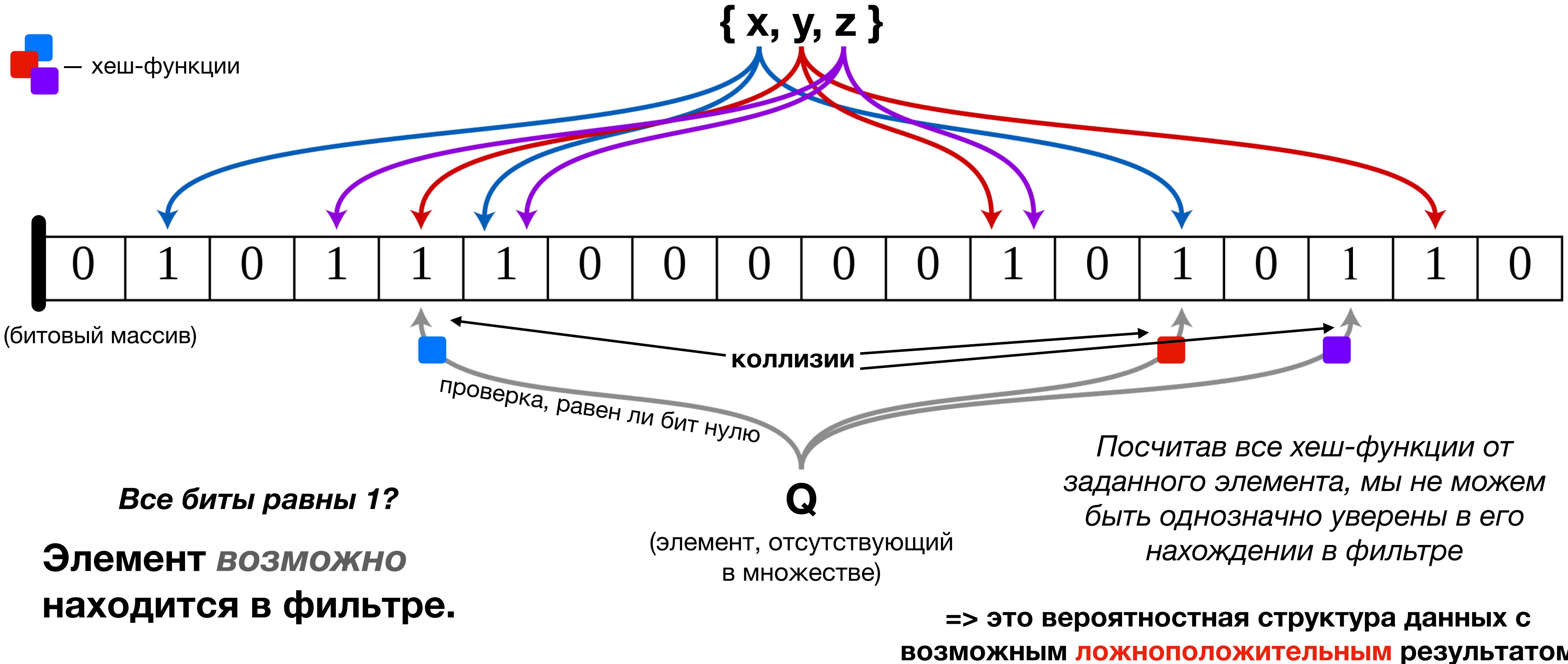
# Фильтр Блума

Как проверить **отсутствие** элемента в фильтре? (случай 1)



# Фильтр Блума

Как проверить **отсутствие** элемента в фильтре? (случай 2)



## **Преимущества?**

**Экономия памяти** – под все элементы выделяется только массив битов фиксированной величины

**Скорость** – все операции выполняются с помощью хеш-функций ( временная сложность  $O(1)$  )

## **Недостатки?**

Вероятность получить **ложноположительный** результат из-за коллизий

## **Как минимизировать недостатки?**

Подобрать **оптимальные** параметры:

1. Количество хеш-функций
2. Размер битового массива

# Какова вероятность получения *false positive* результата?

**n** — количество элементов

**m** — размер фильтра

**k** — количество хеш-функций

1)  $1 - \frac{1}{m}$  — вероятность хеш-функции не установить определённый бит в 1

$$2) \left(1 - \frac{1}{m}\right)^k = \left(\left(1 - \frac{1}{m}\right)^m\right)^{k/m} \approx e^{-k/m}$$

3)  $\left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$  — вероятность того, что после вставки **n** элементов **k** хеш-функциями определённый бит равен 0

$$\lim_{m \rightarrow \infty} \left(1 - \frac{1}{m}\right)^m = \frac{1}{e}$$

вероятность для **k** хеш-функций не установить определённый бит в 1

4)  $1 - \left(1 - \frac{1}{m}\right)^{kn} \approx 1 - e^{-kn/m}$  — вероятность того, что после вставки **n** элементов **k** хеш-функциями определённый бит равен 1

## Вероятность **ложноположительного срабатывания** (p)

5)  $\left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k$  — вероятность того, что для отсутствующего элемента каждая позиция (бит), вычисленная **k** хеш-функциями, будет равна 1

# Как подобрать **оптимальные** параметры?

**n** – количество элементов  
**m** – размер фильтра  
**k** – количество хеш-функций

$$p = \left( 1 - \left[ 1 - \frac{1}{m} \right]^{kn} \right)^k \approx \left( 1 - e^{-kn/m} \right)^k$$

1)  $k = \frac{m}{n} \ln 2$  – **оптимальное** число хеш-функций, минимизирующее **ложноположительные** срабатывания ( при котором  $p \rightarrow \min$  )  
(  $m/n$  – количество битов на элемент )

2) Подставляем в исходную формулу

$$p = \left( 1 - e^{-\left(\frac{m}{n} \ln 2\right) \frac{n}{m}} \right)^{\frac{m}{n} \ln 2}$$

Потенцируем

$$3) \ln p = -\frac{m}{n} (\ln 2)^2$$

Выражаем  $m$  (размер фильтра)

4)

$$m = -\frac{n \ln p}{(\ln 2)^2}$$

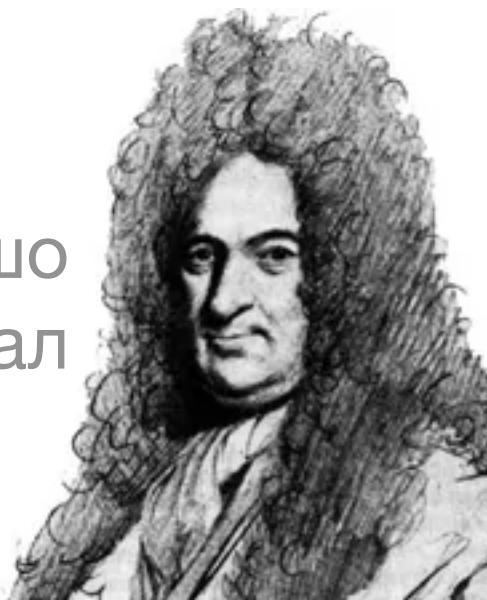
Вуаля!

Мы получили формулы для всех нужных нам **оптимальных** значений фильтра Блума

Кроме того, можно найти **оптимальное** количество бит на один элемент ( соотношение  $m/n$  ) :

$$\frac{m}{n} = -\frac{\ln p}{(\ln 2)^2}$$

хорошо  
посчитал



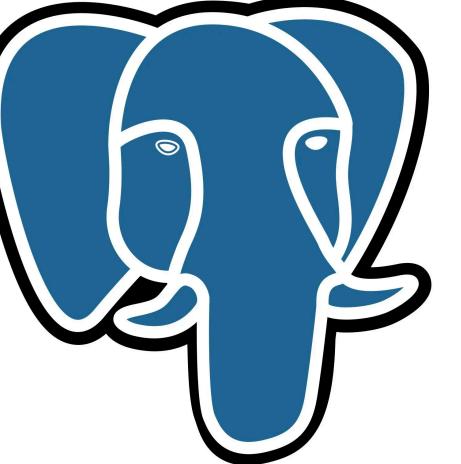
# Итоги

**Фильтр Блума** – это вероятностная структура данных, которая

1. Экономит время и количество обращений к диску
2. Может точно сказать, что некоторого элемента *нет* в множестве
3. Выдаёт *неопределённый* ответ о принадлежности элемента к множеству
4. Применяется в основном как *предварительный* фильтр запросов (для последующего поиска применяются другие алгоритмы)
5. Задаётся исходя из количества элементов (*n*) и желаемой *вероятности ошибки* (*p*) (*ложноположительного* результата) (часто берётся равным 1%)
6. Не поддерживает операцию удаления элемента, т.к. это может затронуть «чужие» результаты хеширования

# Где он используется?

- **Google BigTable, Apache HBase + Cassandra, PostgreSQL** – сокращение времени поиска несуществующих строк или столбцов на диске
- **Google Chrome** – выявление вредоносных URL
- **Venti и WebArchive** (системы архивного хранения) – обнаружение ранее сохраненных данных
- **Medium** – хранение прочитанных статей (чтобы не рекомендовать их в будущем для прочтения)
- **Ethereum** – быстрый поиск логов в блокчейне
- **Grammarly** – поиск слов с орфографическими ошибками



(shocked of how cool  
Bloom filter is)



**Danke für ihre Aufmerksamkeit!**

