Хеширование, хеш-таблицы

04/03/2023

В предыдущих сериях

Мы обсудили разные ordered-структуры, которые явно сохраняют порядок

Ordered-структуры

Обычно можно сделать примерно такой интерфейс, все за $O(\log n)$

- `add`
- `remove`
- `get`
- `lower_bound`
- `next`

Дальше, на самом деле, особо некуда двигаться: Ordered-структура реализует кучу, и если ускорить все операции, то можно будет сортировать массив быстрее $O(n \log n)$, а это теоретически невозможно.

Еще немного про контейнеры

- Иногда контейнеры реализуют интерфейс `add`, `get`, `remove` явно - тогда это *множества* (*set*)
- Иногда контейнеры ведут себя как массивы с произвольными ключами - то есть, где-то рядом с ключом хранится еще и значение. Такое key-value хранилище иногда называют *map*, иногда *dictionary*.

Новый интерфейс

Давайте уберем самое ненужное нам требование на упорядоченность

- `add`
- `get`
- `remove`

Хеш-функция

$$h: \ X \to [0; C]$$

Свойства:

- Определено: для какого-то произвольного объекта, зависит от функции
- Значения: целые числа от 0 до C.
- Если $h(x_1) = a$ и $x_2 = x_1$, то $h(x_2) = a$

Еще дополнительно хорошо, если есть параметризуемость:

$$h_{t_1,t_2,\ldots t_n} \in \mathbb{H}(\mathbb{T}_1, \mathbb{T}_2, \ldots, \mathbb{T}_n)$$

Вопрос 1

- Пример хеш-функции для множества $[0; 15]$

Вопрос 2

- Пример хеш-функции для множества $\mathbb{N}$

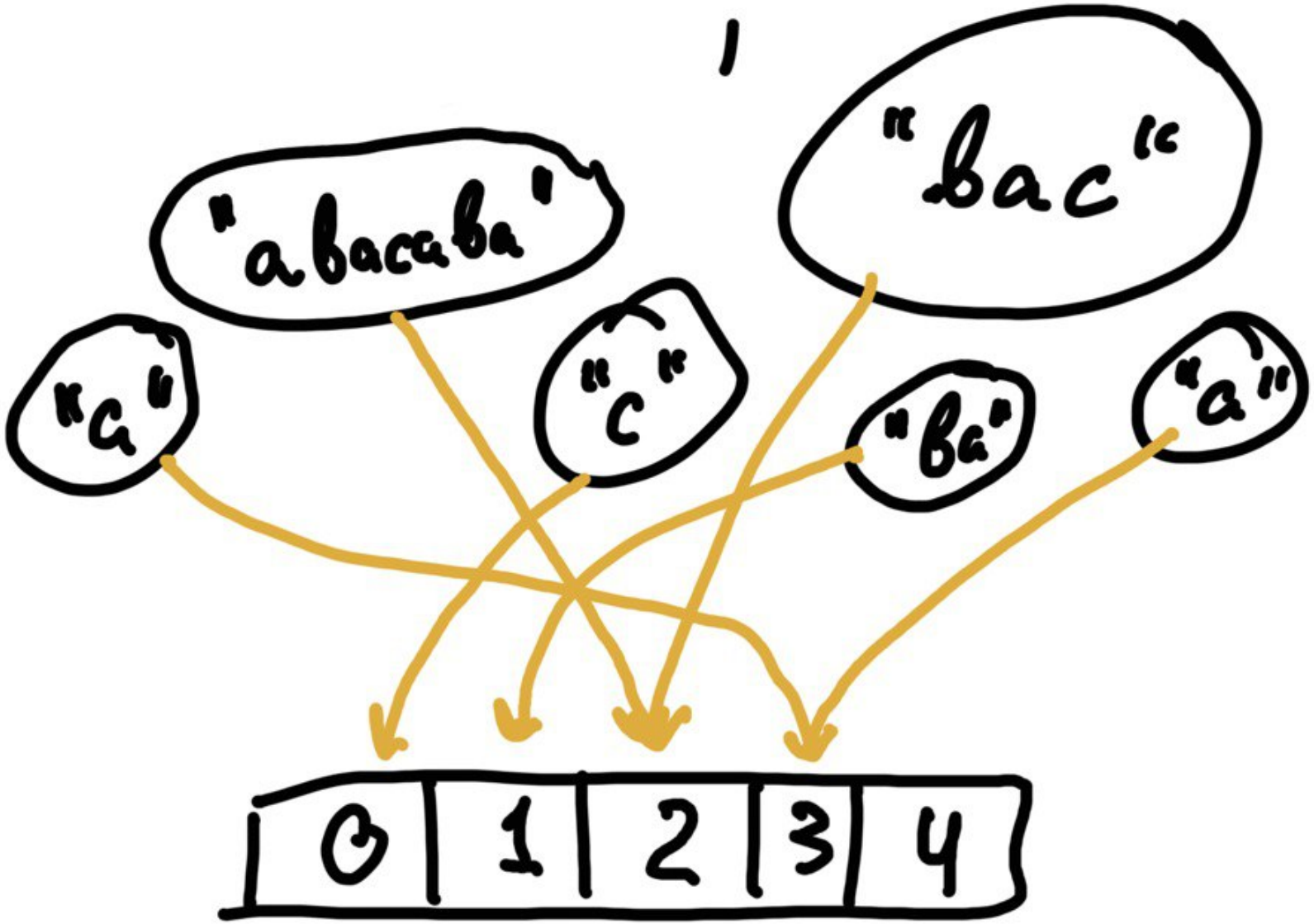
Вопрос 3

- Когда третье свойство не выполняется?

Идея

Мы хотим переводить "большие" и "тяжелые" объекты в самый простой дискретный объект - целое число.

Ограниченность нужна, чтобы хеш-функция уменьшала множество


"авасава"
"вас"
"а"
"с"
"ва"
"а"
0
1
2
3
4

Вопрос

Является ли $h(x) = 0$ хеш-функцией?

"Хорошие" хеш-функции

Мы хотим, чтобы хеш-функция давала объектам равномерно распределенное значение.

Тогда, если у нас есть n объектов из очень большого пространства (например, строки имеют размерность $256^{\mathbb{N}}$) мы можем раскидать объекты по группам, каждая из которых будет содержать $\frac{n}{C}$ элементов.

"Безопасные" хеш-функции

Дополнительное требование, которое нам пока что не обязательно - чтобы внесение "шума" в аргумент хеш-функции могло очень сильно изменить значение.

Вопрос

- Является ли хеш-функция от целых чисел $h(x) = x \mod 10$ хорошей?
- А безопасной?

Хеш-таблица

Идея хеш-таблицы в том, чтобы равномерно распределить элементы по ячейкам.

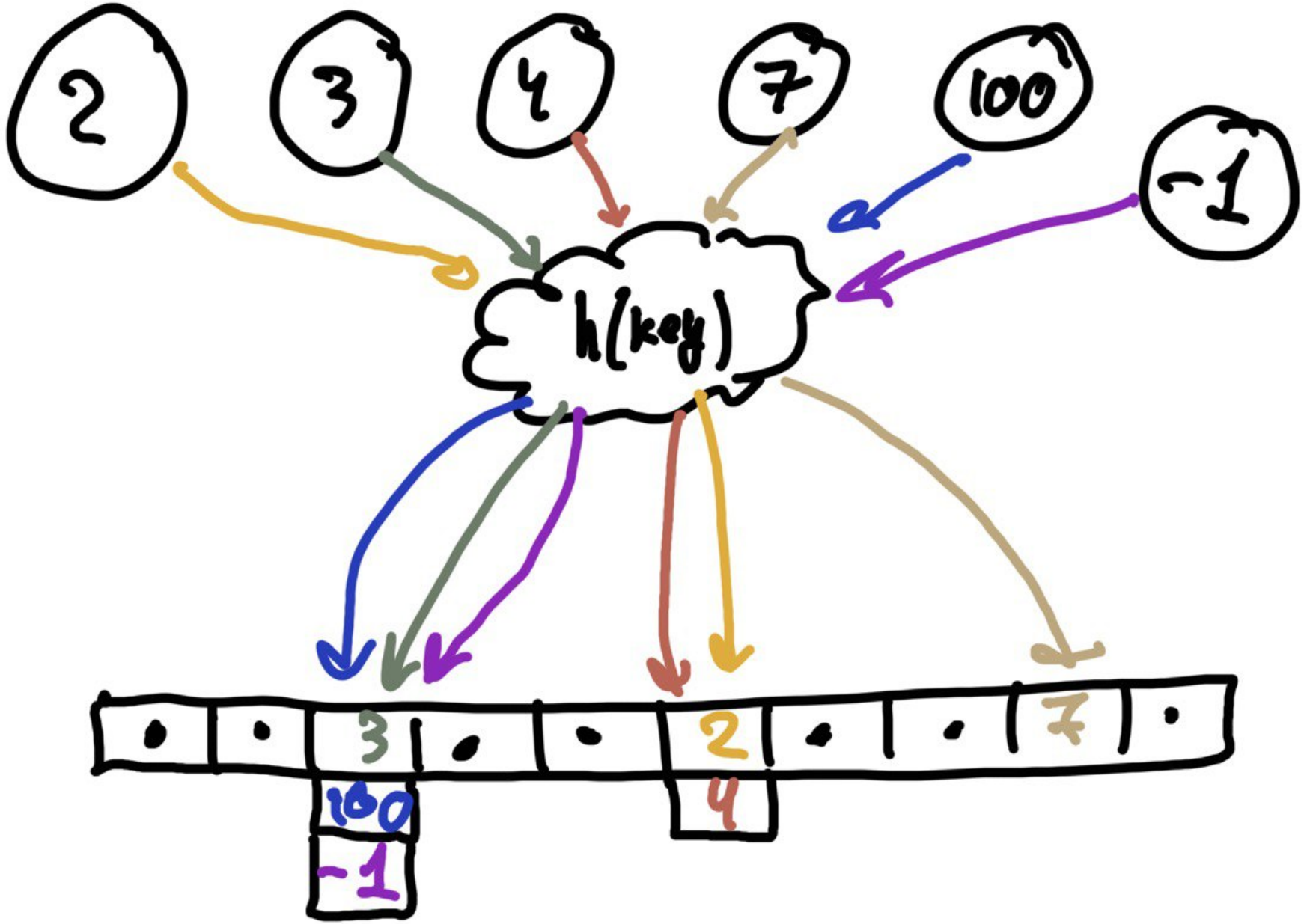
В ячейке можно хранить динамический массив (или список), в который мы будем добавлять элементы

```
class HashTable:
    def __init__(self):
        self.size = 1000
        self.data = [[] for _ in range(self.size)]

    def hash(self,elem):
        return elem % self.size

    def insert(elem):
        self.data[self.hash(elem)].append(elem)
```

Поиск

```
class HashTable:
    # ...
     def find(self, elem):
         for x in self.data[self.hash(elem)]:
             if x == elem:
                 return True
         return False
```

В чем недостаток?

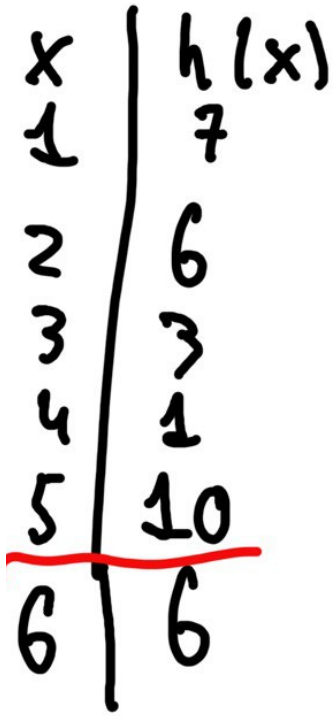

2
3
4
7
100
-1
h(key)
3
2
7
100
-1
4

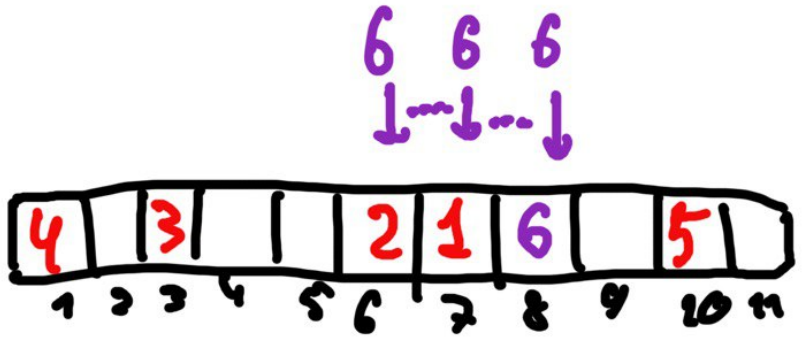
Закрытая адресация

Тот подход, который мы написали выше, называется хеш-таблицей с закрытой адресацией

Открытая адресация

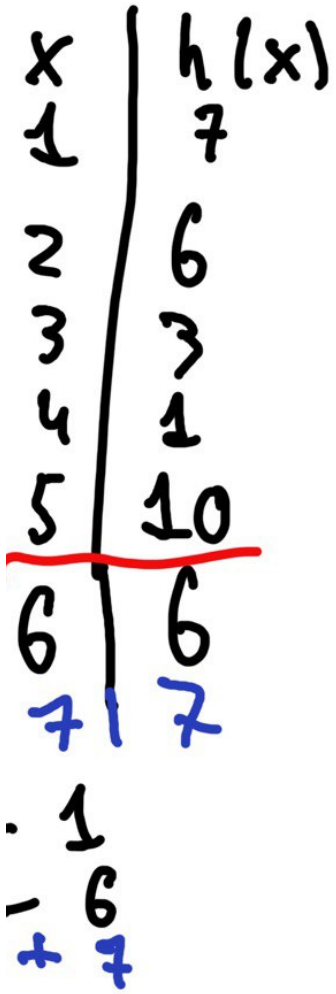
Вместо того, чтобы хранить ячейки явно, можно в ячейке хранить один элемент, и при поиске идти в следующий

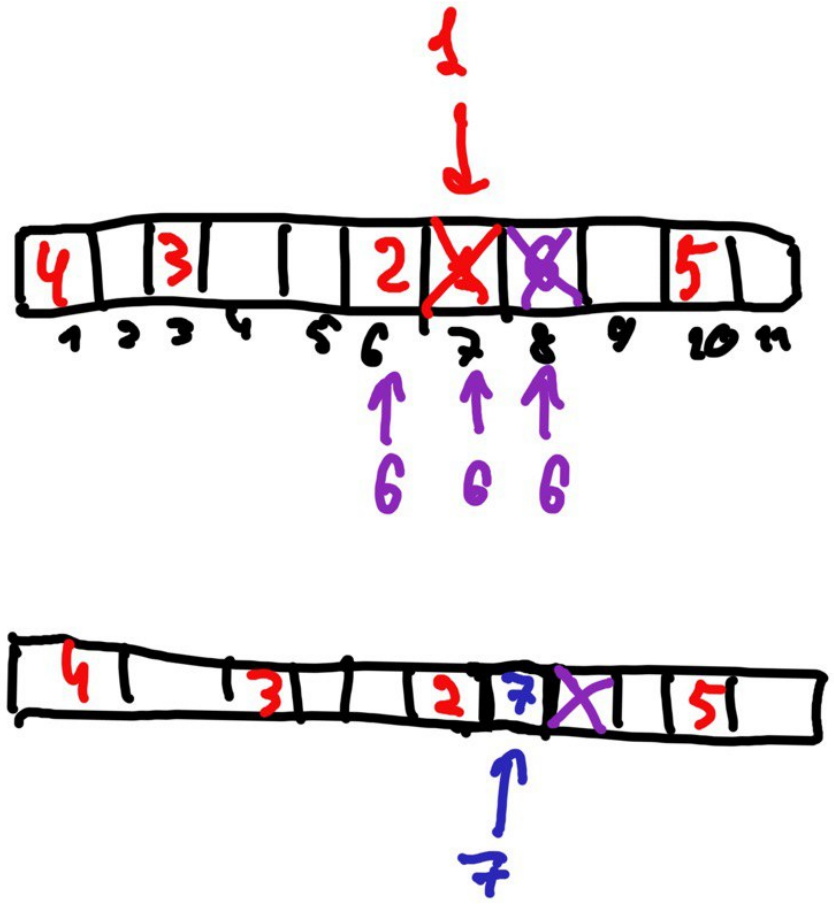




Удаление в открытой адресации

Если мы хотим что-то удалить, то мы не удаляем, а записываем специальную пометку, что следующий элемент может нас перезаписать.





Асимптотика

В среднем добавление работает за количество коллизий. Понятно, что если коллизий нет, то все добавления за $O(1)$.

Если $n > C$, то коллизии неизбежны (принцип Дирихле).

$$\alpha = \frac{n}{C}$$

Поддерживают, например

$$0.25 < \alpha < 0.75$$

Хеш-функция для int: пример

$$t, p = random(),\ random()$$
$$h(x) = (x \oplus t) \cdot p \pmod{C}$$

Хорошую хеш-функцию обычно берут из *семейства* и инициализируют параметрами, чтобы никто снаружи (например, автор тестов) не знал функцию наверняка.

Вопрос

Как захешировать произвольный объект в памяти компьютера?

Полиномиальное хеширование

Техника для строки, которая позволяет найти не только хеш строки, но и хеш всех ее подстрок

Чем-то похоже на префиксные суммы.

Явная формула

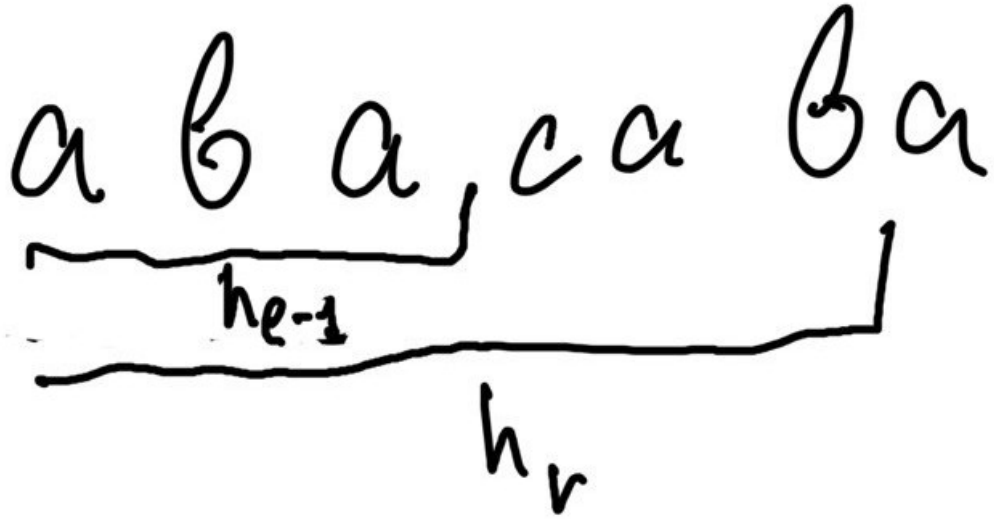
$$h(s_{l,r}) = s_l \cdot t^{r-l} + s_{l+1} \cdot t^{r-l-1} + \ldots + s_{r-1} \cdot t + s_r$$

Почему именно такая?

$$h(s_{l,r+1}) = h(s_{l,r}) \cdot t + s_{r+1}$$

Префиксные хеши

- Считаем массив $h_r = h(s_{0,r})$
- Тогда
  $h(s_{l,r}) = h_r - h_{l-1} \cdot t^{r-l+1}$



Вероятность коллизии

Наша проблема в том, что иногда подстроки не равны, а их хеши равны. Надо понять, с какой вероятностью это происходит.

После того, как мы посчитали для строки s_i ее уникальную хеш-функцию, мы как бы заняли один слот из C.

$$f(n, C) = \prod_{i=1}^{n}(1 - \frac{i}{C}) \sim \prod_{i=1}^{n} e^{\frac{-i}{C}} = e^{\frac{-n(n+1)}{2C}}$$

$$e^{\frac{-n(n+1)}{2C}} = \frac{1}{2} \iff n \sim \sqrt{C}$$

Грязные хаки: два модуля

Иногда бывает так, что 32-битного модуля не хватает для того, чтобы коллизий не случалось.

Тогда на помощь приходит 64-битный модуль! Или 128! И так далее.

В языках без длинной арифметики можно делать два взаимно простых модуля.

По *некоторым* инженерным соображениям в языках типа C++ хорошо работает пара $(2^{64}, 10^9 + 7)$.

Поиск подстроки в строке

Хотим: найти s в t

```
for i in range(len(s) - len(t)):
    if s[i:i +len(t)] == t:
        print(i)
```

Поиск подстроки в строке

Можем перебрать позицию i и проверить, что $h(t_{i,i+|s|}) = h(s)$

Получаем асимптотику $O(|s| + |t|)$ вместо тривиальной $O(|s| \cdot |t|)$

Чек-суммы

Такие хеш-функции, как MD5 или SHA256, могут использоваться для превращения файла в маленькую строку

```
$ md5sum 1.wav
c30b4b28c773ef796f1cb919e02b98f5  1.wav
```

Можно использовать для чексумм, чтобы проверить, что содержимое файлов совпадает.

А можно хранить пароль!

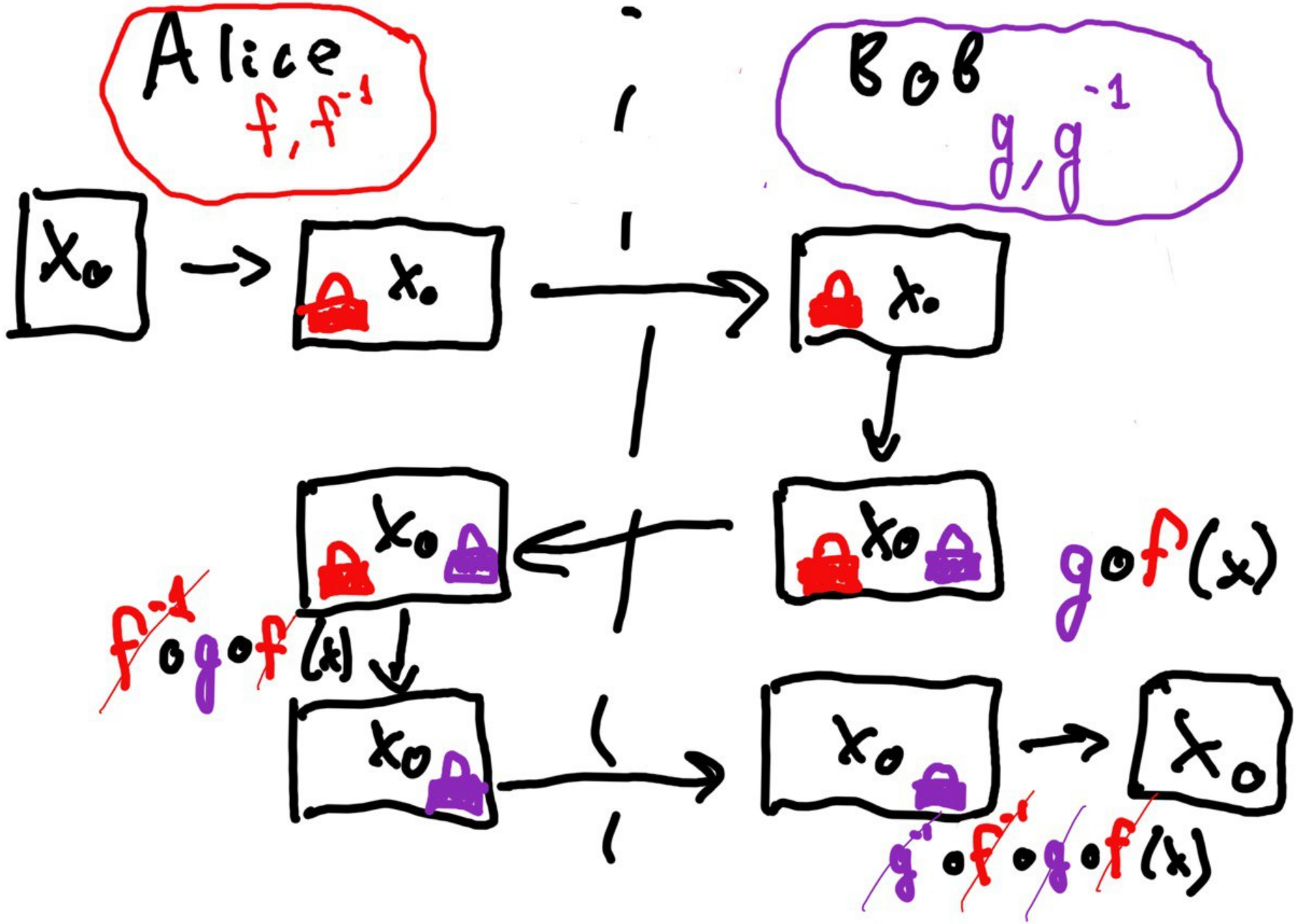
RSA, Диффи-Хэлман

Сделаем $f(x)$ и $g(x) = f^{-1}(x)$. То есть $g(f(x)) = x$. $f(x)$ называется публичным ключом, а $g(x)$ приватным.

Пусть я сказал всем $f(x)$, а сам знаю $g(x)$. Тогда, если проверяющий возьмет секретный x_0, посчитает $f(x_0)$ расскажет всем и спросит, чему был равен x_0, то никто не поймет, а я посчитаю $g(f(x_0)) = x_0$.

Так проверяющий узнает, что я действительно знаю приватный ключ, который был создан в паре с публичным ключом $f(x)$.

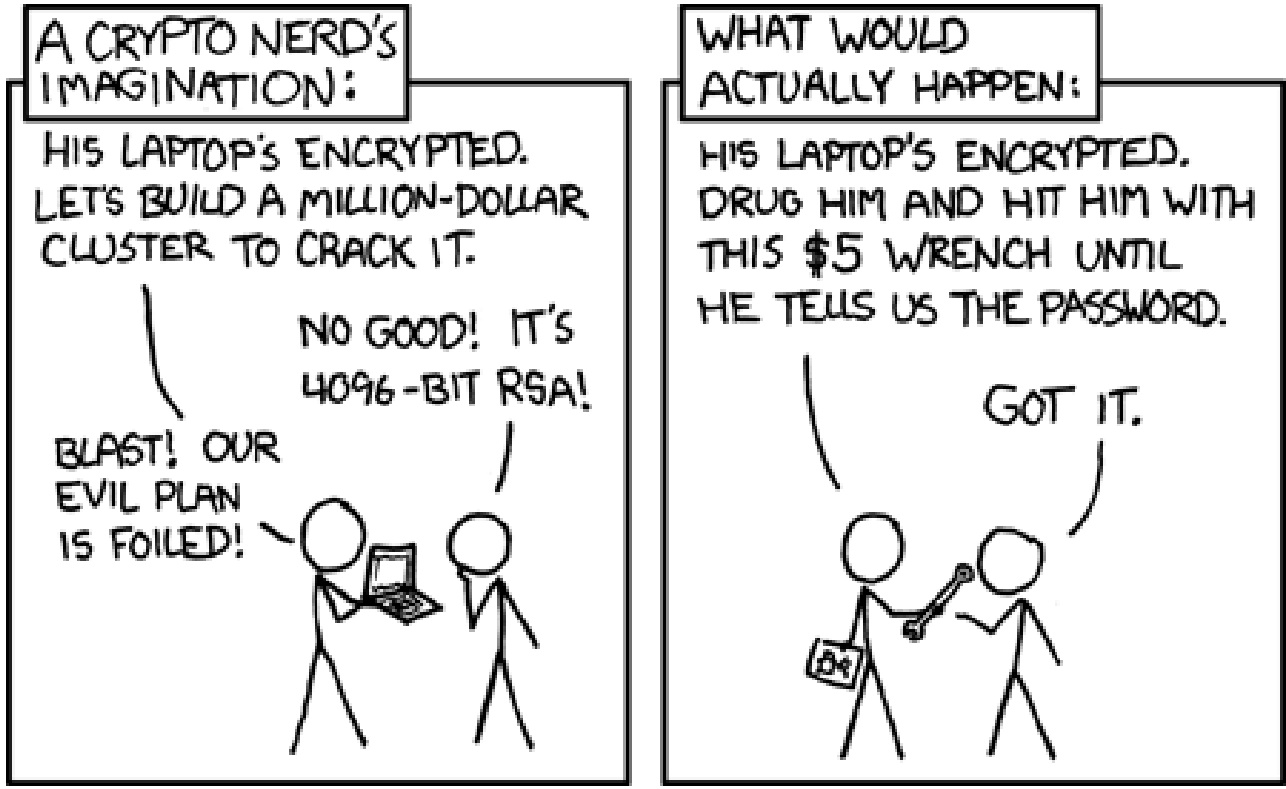
Конкретно в случае RSA f, g это теоретико-числовые функции.


Alice
f, f^{-1}
Bob
g, g^{-1}
x_0
x_0
x_0
x_0
x_0
$f^{-1} \circ g \circ f(x)$
$g \circ f(x)$
x_0
x_0
x_0
$g^{-1} \circ f^{-1} \circ g \circ f(x)$

Цифровая подпись

С помощью RSA достаточно просто верифицировать маленькие сообщения, но это плохо подходит для больших сообщений. Поэтому для подтверждения владения иногда подписывают и верифицируют хеш-функцию от файлов.

Мем


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

Пет-проект

Можно попробовать прочитать про RSA подробнее и придумать свою обратимую (возможно, дурацкую) функцию, сгенерировать приватный и публичный ключ, написать честного собеседника, который проверил наше знание ключа, и взломщика, который несколько раз подтвердил наше знание подписи и на основе этих данных наш приватный ключ подобрал.

Например, можно поиграть с $f(x) = x + K \pmod{10}$, а потом с настоящими функциями, которые используются для шифрования в жизни.

Дополнительное чтение

- RSA
- https://ru.algorithmica.org/cs/hashing/isomorphism/
- Дискретное логарифмирование