Строки

16/03/2024

Для справки

- "abcd" = [61, 62, 63, 64]
- В некоторых языках строки больше похожи на массивы, в некоторых - на объекты в памяти.
- Очень *жизненный* тип данных

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)
```

Зет-функция

Сначала введем зет-функцию. z_i - самая большая такая длина, что $s_{i:i+z_i} = s_{0:z_i}$.

Какая z_2 у $hahaha$?

Алгоритм КМП

Соберем строку $s' = s\#t$, причем символ # уникален в рамках s'.

Тогда для строки $s' = s\#T$ все индексы (кроме нулевого) с $z_i = |s|$ соответствуют вхождению s в T.

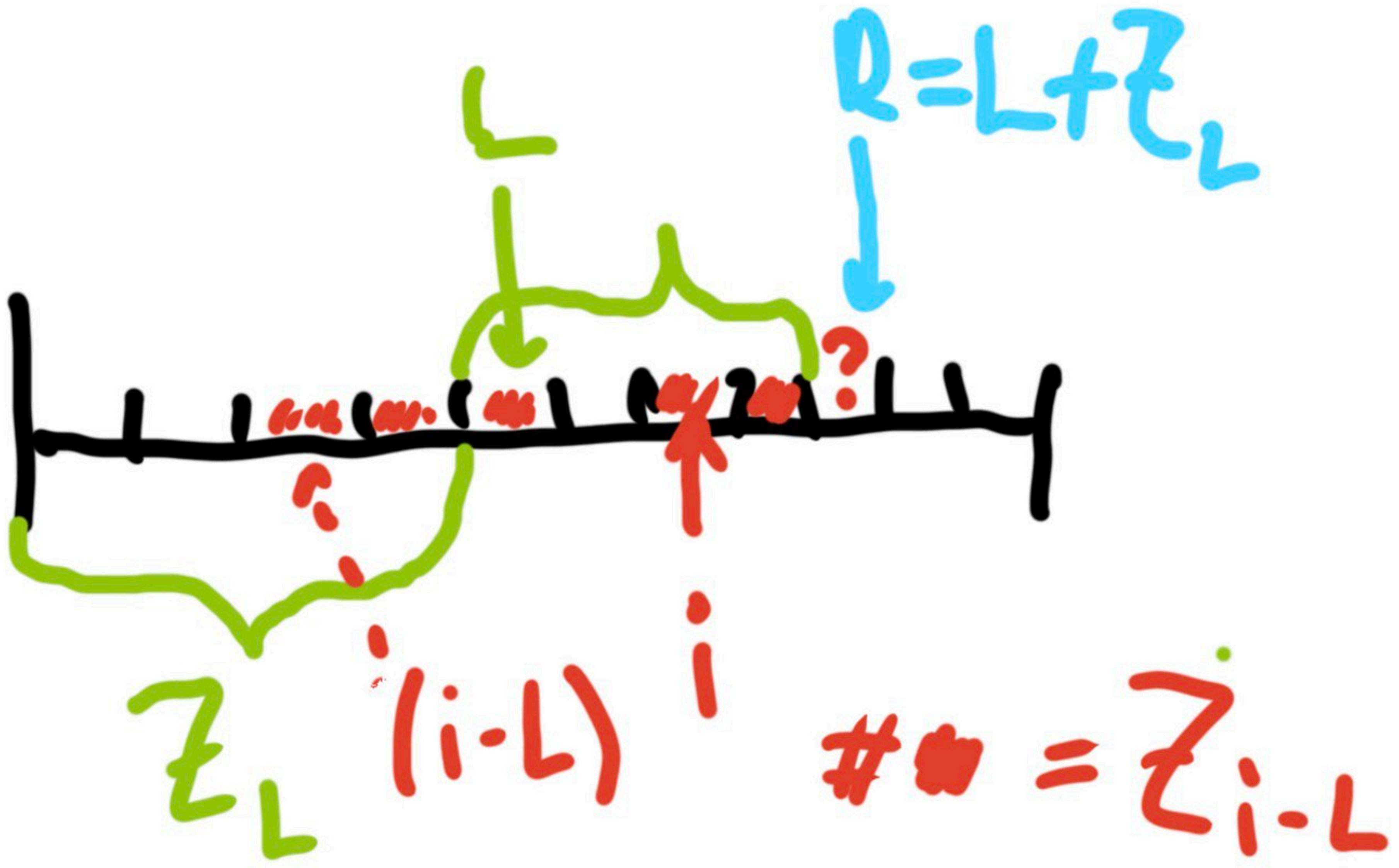
Примечание. Более каноничным вариантом считаеся использование префикс-функции для КМП, они с зет-функцией взаимозаменяемы.

Подсчет зет-функции

Пусть мы посчитали все $z_k, k < i$

И пусть есть какой-то индекс $k > 0$ такой, что $k + z_k \geq i$. То есть текущий индекс лежит внутри какой-то уже обработанной подстроки.

Но тогда $z_i \geq min(z_{i-k},\ z_k + k - i)$


R=L+Z_L
L
?
Z_L
(i-L)
i
= Z_{i-L}

Два указателя

Будем обрабатывать строку двумя указателями. Правым -- читать строку, левым -- подсчитывать зет-функцию.

Тогда в зависимости от $argmin(z_{i-l},\ r - i)$ выходят два случая: либо про позицию i заранее известен ее z_i, либо ее z_i находится правее r, и нужно прочитать больше символов

Реализация

```
z = [0] * n
l = 0
for i in range(1, len(s)):
    z[i] = min(z[i - l], l + z[l] - i)
    while i + z[i] < len(s) and s[z[i]] == s[i + z[i]]:
        z[i] += 1
    if i + z[i] > l + z[l]:
        l = i
```

Асимптотика $O(|s|)$ (в случае КМП $O(|s| + |T|)$) амортизированно, потому что и левый и правый указатель читают каждый символ один раз.

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:

$$h_{t_1, t_2, \dots t_n} \in \mathbb{H}(\mathbb{T_1}, \mathbb{T_2}, \dots, \mathbb{T_n})$$

Вопрос 1

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

Вопрос 2

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

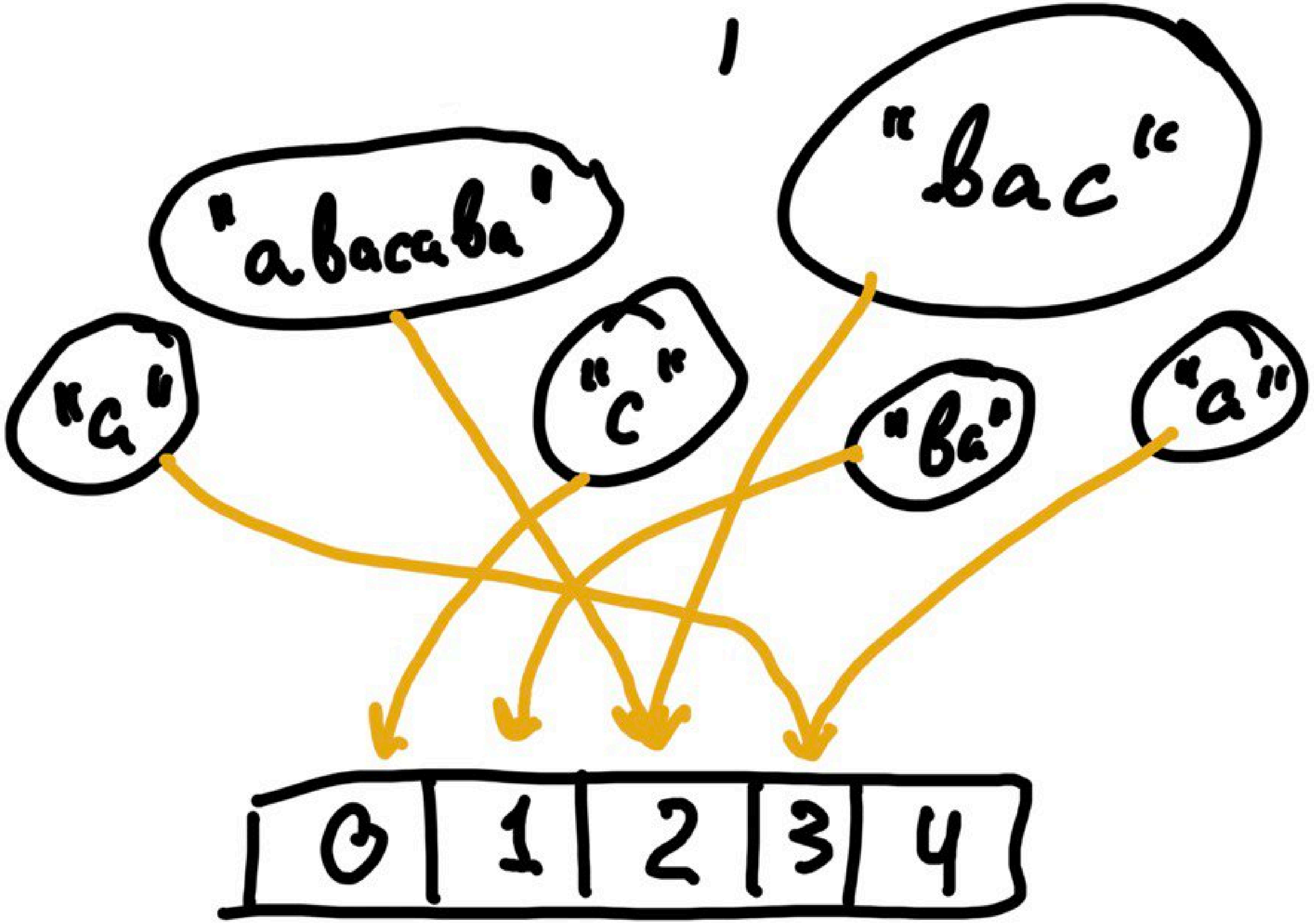
Вопрос 3

- 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


"abacaba"
"bac"
"a"
"c"
"ba"
"a"
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 различных 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.

А, значит, если $h(a) = h(b)$, то с большой вероятностью $a = b$.

Вопрос

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

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

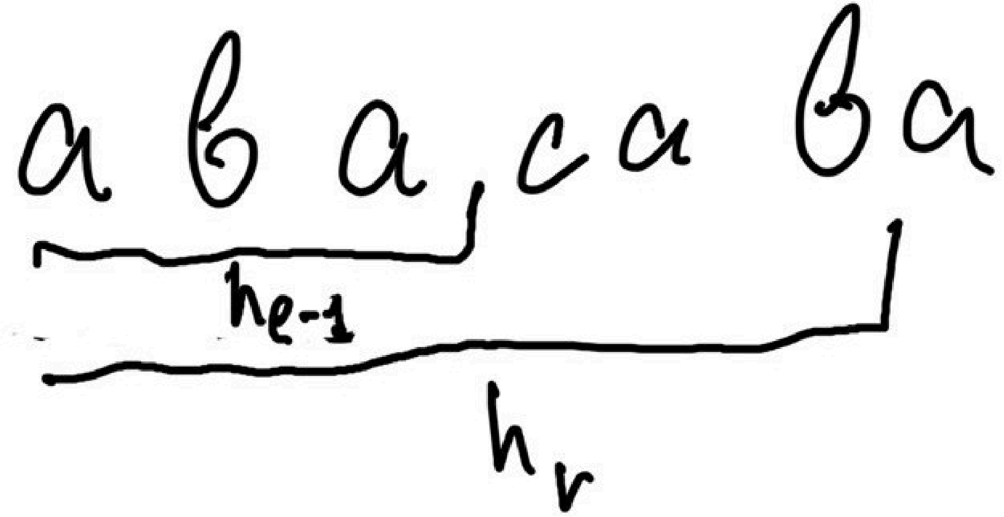
$$h(s_{l,r}) = s_l \cdot t^{r-l} + s_{l+1} \cdot t^{r-l-1} + \ldots + s_{r-1} \cdot t + s_r$$

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 для произвольного отрезка $h(s_{l,r}) = h_r - h_{l-1} \cdot t^{r-l+1}$



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.

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

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

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 $O(|s| \cdot |t|)$

Хеш-таблица

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

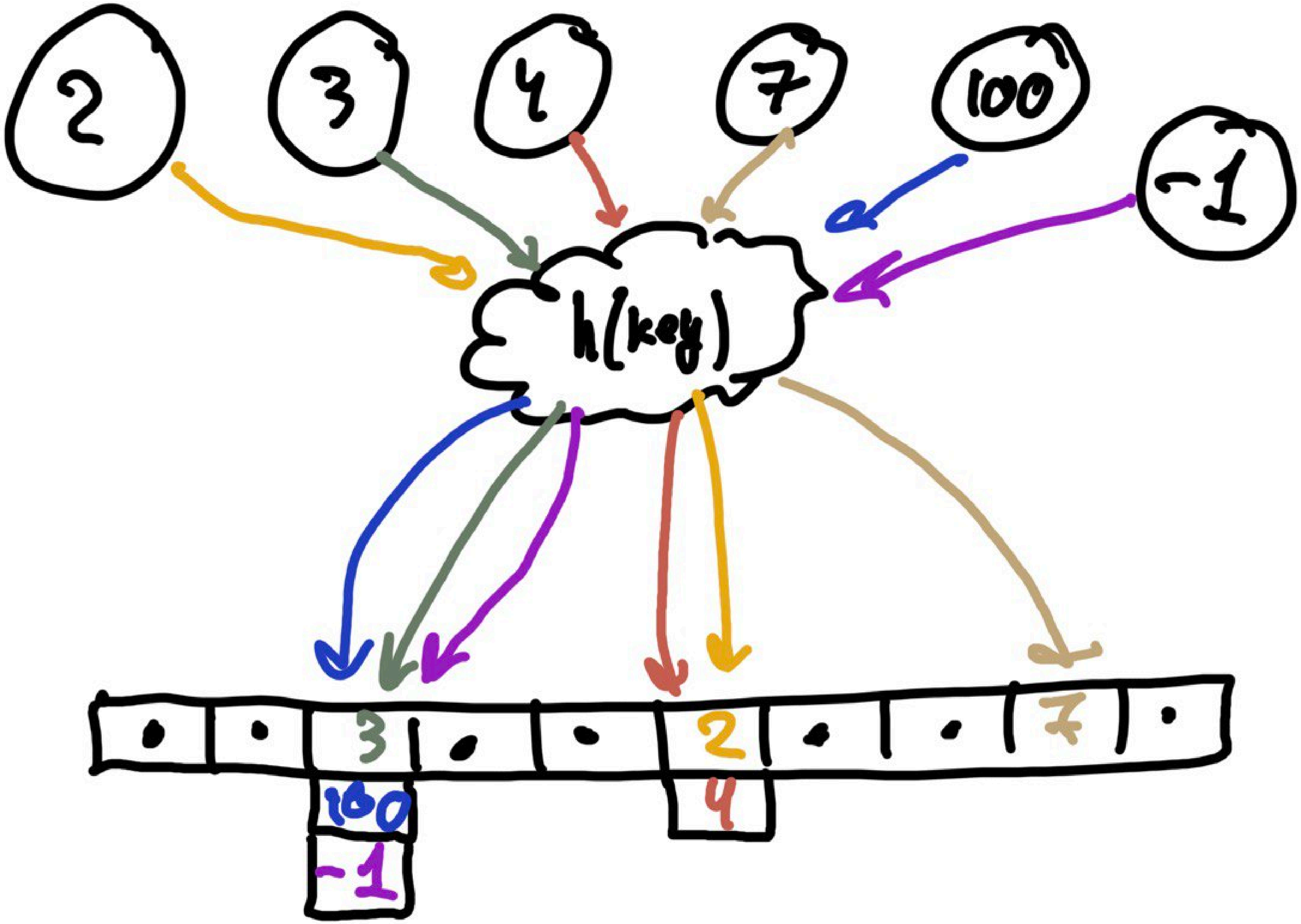
В ячейке можно хранить динамический массив (или список), в который мы будем добавлять элементы

```python
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

Асимптотика

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

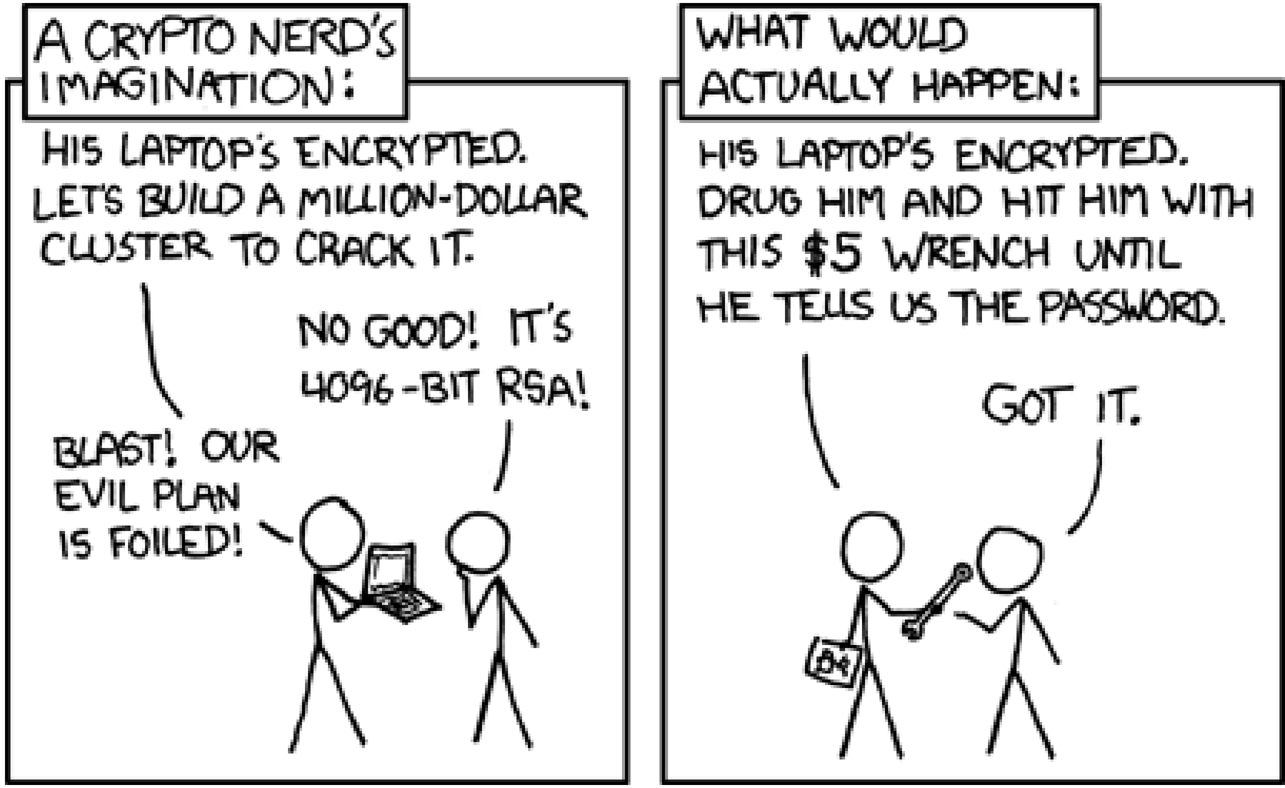
Если $n > C$, то коллизии неизбежны (принцип Дирихле).

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

Поддерживают, например

$$0.25 < \alpha < 0.75$$

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.