

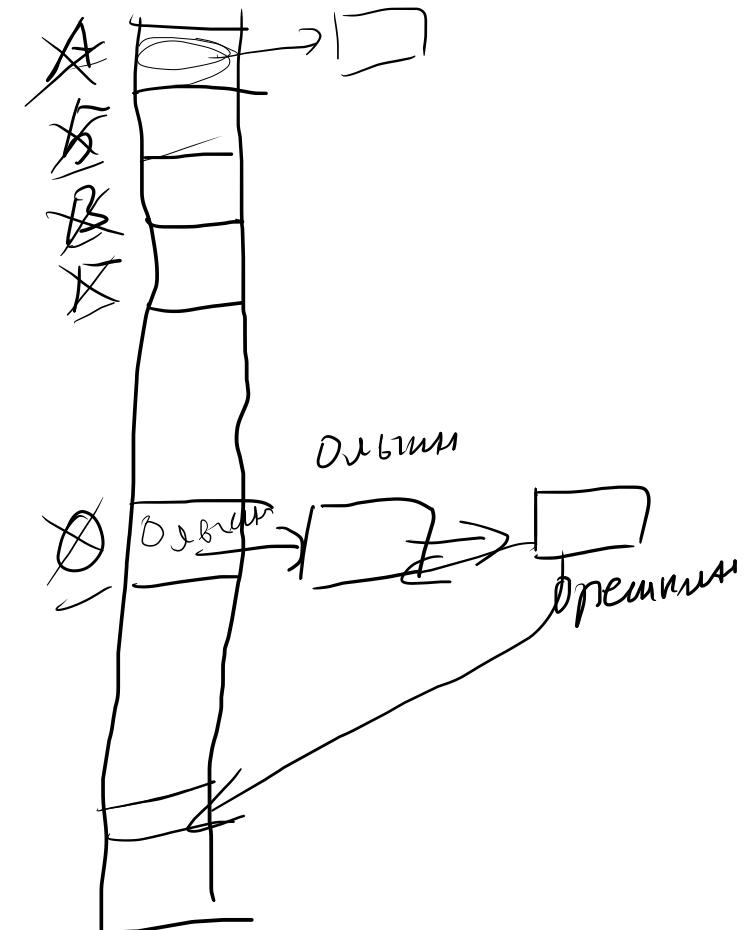
## Хеш-таблицы

Реализуют стандартные операции: **Insert**, **Delete** и **Search**

Эффективны для организации словарей

Diagram illustrating a hash function  $h(x)$  mapping an input  $x$  to an output  $y$ . A box labeled  $h(x)$  contains a circle with  $x$ , with an arrow pointing to a circle with  $y$ .

$$h(x_1) = y$$
$$h(x_2) = y$$



**План:**

- 1) Таблицы с прямой адресацией (нет коллизий)
- 2) Хеш-таблицы (коллизии возможны)
  - a) с закрытой адресацией (метод цепочек)  
способы равномерного распределения данных по ячейкам: методы деления, умножения, универсальное
  - b) с открытой адресацией (все данные записаны в массив с каким-то шагом относительно совпадающих по хешу)  
методы исследования: линейное, квадратичное, двойное хеширование
  - c) идеальное хеширование

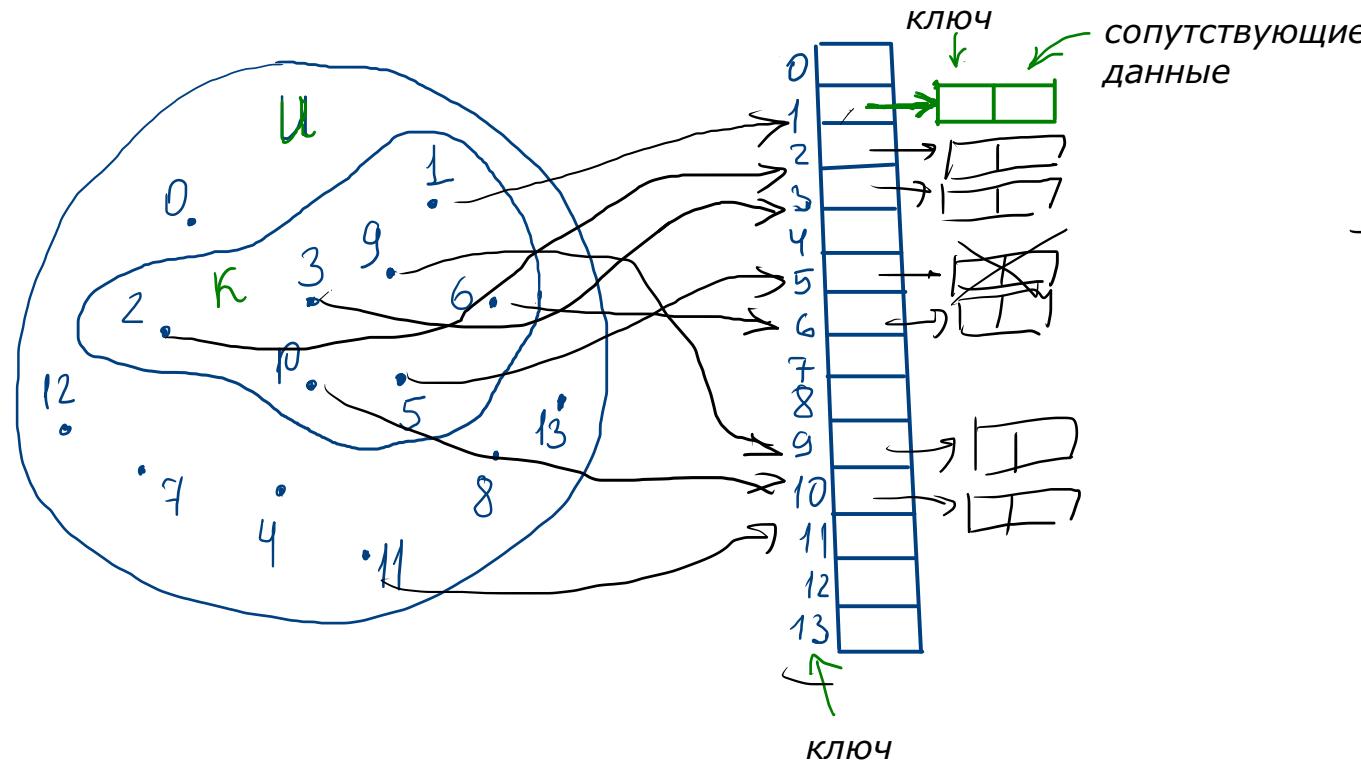
## Таблицы с прямой адресацией

$U = \{0, 1, \dots, m-1\}$  - множество ключей

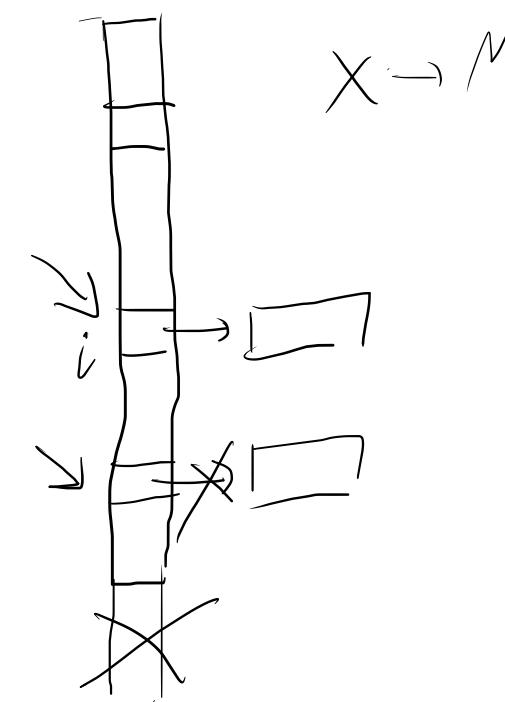
Никакие два элемента не имеют одинаковый ключ



Массив = таблица с прямой адресацией, где ключи - индексы в массиве



$$h(X) = i$$



Операции:

DIRECT-ADDRESS-SEARCH( $T, k$ )  $\Theta(1)$

1 return  $T[k]$

DIRECT-ADDRESS-INSERT( $T, x$ )  $\Theta(1)$

1  $T[x.key] = x$

DIRECT-ADDRESS-DELETE( $T, x$ )  $\Theta(1)$

1  $T[x.key] = NIL$

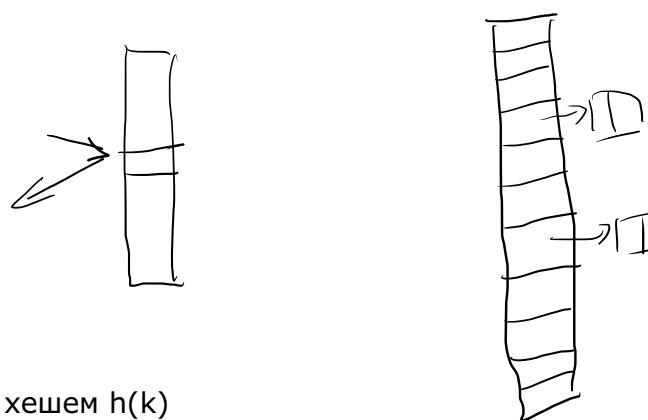
! Можно не хранить ключ, т.к. ключ = индекс в массиве

! Можно экономить память, если хранить объекты прямо в ячейках таблицы

## Хеш-таблицы

Недостаток прямой адресации: если  $|U|$  велика, то хранить данные прямой адресацией непрактично

$k \ll |U|$ , где  $k$  - реально хранимые ключи



Хеш-таблицы снижают требования к памяти до  $O(k)$   
(и поиск осуществляется за  $O(1)$  в среднем)

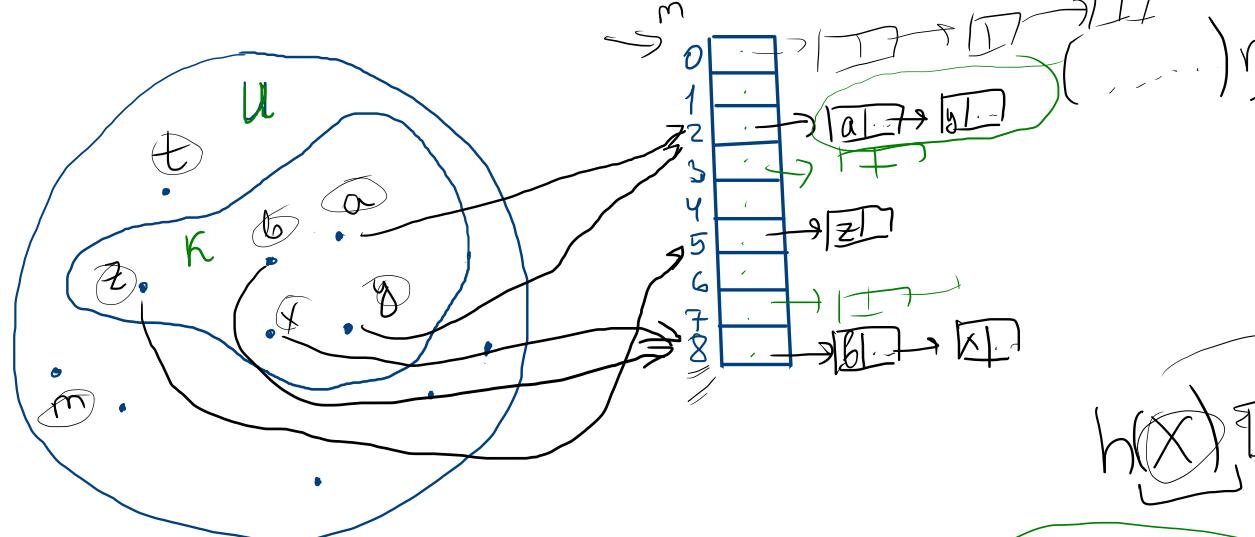
**Суть:** хранение элементов (с ключом  $k$ ) в ячейках с хешем  $h(k)$

То есть для всех добавляемых в таблицу значений (ключей,  $k$ ) может быть вычислена функция  $h(k)$

$h(k)$  показывает, например, индекс в массиве, куда надо записать ключ

$h: U \rightarrow \{0, 1, \dots, m-1\}$ , где  $m$  - число индексов в таблице

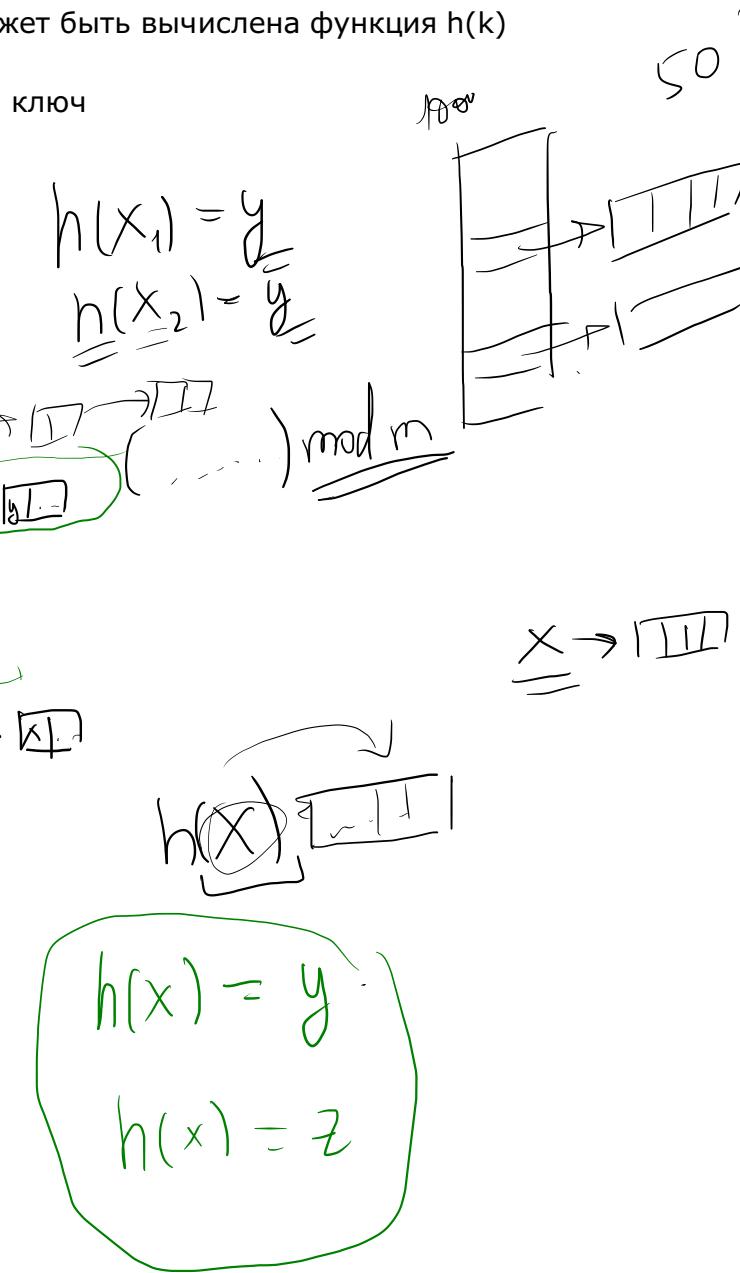
**Цель:** уменьшить рабочий диапазон индексов



**Важно:**  $h$  - случайная и детерминистическая

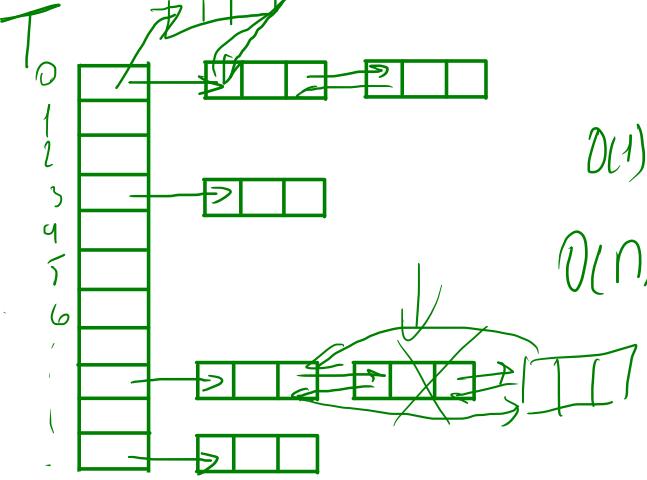
Избежать коллизий в принципе не возможно, если  $|U| > m$

**Хорошая хеш-функция минимизирует число коллизий**



## Хеш-таблицы с закрытой адресацией (метод цепочек)

Теперь из каждой ячейки таблицы ведет указатель на голову двусвязного списка из ключей, чьи значения хеш-функции совпали



$$h(x) = \lfloor x \bmod m \rfloor \quad (\text{для упрощения удаления})$$

Операции:

CHAINED-HASH-INSERT( $T, x$ )

- 1 Вставка  $x$  в заголовок списка  $T[h(x.key)]$

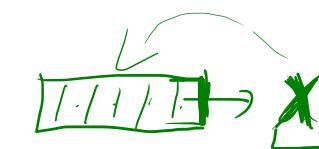
CHAINED-HASH-SEARCH( $T, k$ )

- 1 Поиск элемента с ключом  $k$  в списке  $T[h(k)]$

CHAINED-HASH-DELETE( $T, x$ )

- 1 Удаление  $x$  из списка  $T[h(x.key)]$

$X$  - ключ  
 $y$  - значение  
хеш функции.



$$h(x) = y$$

Про оценку времени поиска конкретного элемента в таблице

Коэффициент заполнения:

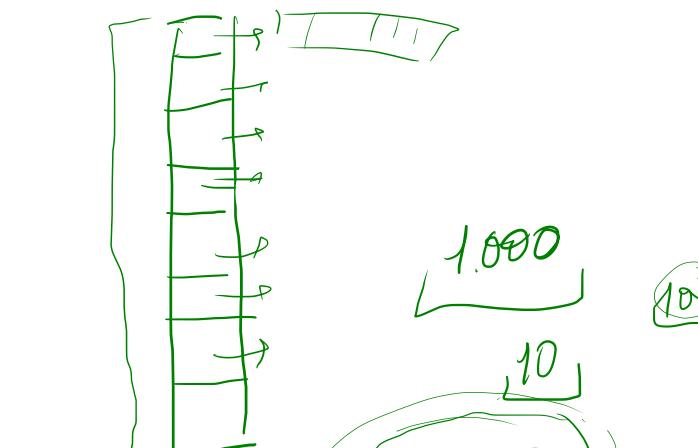
$$\lambda = \frac{n}{m} \rightarrow \text{многоваж}$$

Анализ поиска:

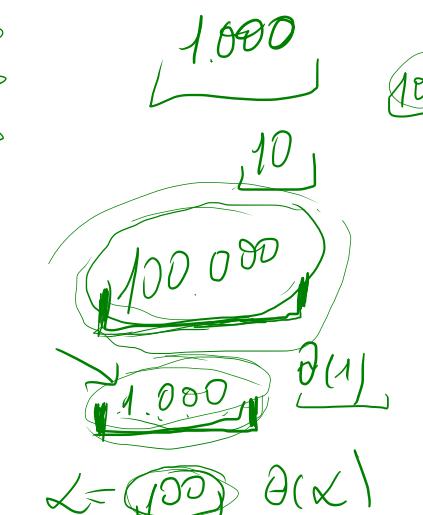
1) худший случай - все элементы оказались в одной ячейке  
время вычисления хеша + поиск по всем элементам

2) идеальный случай - простое равномерное хеширование  
(т.е. для каждого ключа равновероятно его помещение в любую из ячеек)

$$\Theta(1)$$



$$\frac{n}{m}$$



$$\lambda = 100 \quad \Theta(1)$$

Вывод: если число ячеек пропорционально числу элементов, то  $n = O(m) \Rightarrow$   
коэффициент заполнения - константа  
=> поиск требует в среднем константное время

$$\Theta(1 + \lambda)$$

$$+ 100$$

$$\left[ \frac{n}{m} \right]$$

$$\left[ \dots \right]$$

## Минимизация коллизий: методы деления, умножения и универсального хеширования

Качество хеш-функции определяется ее близости к простому равномерному хешированию

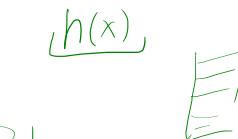
т.е. для каждого ключа равновероятно помещение в любую из  $m$  ячеек

Дано: распределение вероятностей поступающих данных **неизвестно**

Ожидается: хеш-функция никак не коррелирует с закономерностью данных

### 1. Метод деления

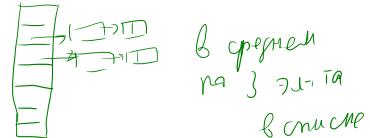
$$h(k) = k \bmod m$$



Ограничения:  $m -$  не  $\geq 2^p$   $h(k) -$  множимо биты

Хороший вариант:  $m -$  пурпурное

Пример:  $n = 2000$   $m = \frac{200}{3} \approx 701$



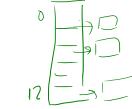
### 2. Метод умножения

$$h(k) = \lfloor m(kA \bmod 1) \rfloor \quad 0 < A < 1 \quad (0, \dots, 1)$$

$KA -$  крайнее значение

Ограничения:  $m -$  пурпурная  $\geq 2^p$  (на  $2^p$  уменьшить лишко)  $\frac{1}{2}, \dots, 1$

Хороший вариант:  $A \approx \frac{\sqrt{5}-1}{2}$



Пример:

$m = 13, A = 0,618033$

Записать 25, 44, 97 в таблицу

1.  $h(k) = 13 * \{25 * 0,618033\} \rfloor = 13 * \{15,450825\} \rfloor = 13 * 0,450825 \rfloor = 13 * 5,860725 \rfloor = 5$
2.  $h(k) = 13 * \{44 * 0,618033\} \rfloor = 13 * \{27,193452\} \rfloor = 13 * 0,193452 \rfloor = 12,514876 \rfloor = 12$
3.  $h(k) = 13 * \{97 * 0,618033\} \rfloor = 13 * \{59,949201\} \rfloor = 13 * 0,949201 \rfloor = 12,339613 \rfloor = 12$

### 3. Универсальное хеширование

Любая фиксированная хеш-функция язвима, так как входные данные могут оказаться в одной ячейке

Решение: случайный выбор хеш-функции

Опр: H универсальное множество хеш-функций, если для любых ключей k и l, число функций,  
для которых  $h(k) = h(l)$  не превышает  $\frac{|H|}{m}$

$$H = \{h_1, h_2, \dots\}$$

т.е. при случайном выборе хеш-функции из H вероятность коллизии между различными ключами не  
превышает вероятности случайного выбора двух одинаковых значений из  $\{0, 1, \dots, m-1\}$

Построение универсального класса хеш-функций:  
 $P > m$  все числа в множестве  $(0, \dots, P-1)$

$$\mathbb{Z}_P = \{0, 1, \dots, P-1\}$$

$$h_{ab}(k) = ((ak+b) \bmod P) \bmod m$$

$$a \in \mathbb{Z}_P^*, b \in \mathbb{Z}_P$$

$P(P-1)$  - всего хеш-функций

$$h_l(k)$$

$$h_n(k)$$

$$h_m(k)$$

$$h_{m-1}(k)$$

$$h_{m-2}(k)$$

$$h_{m-3}(k)$$

$$h_{m-4}(k)$$

$$h_{m-5}(k)$$

$$h_{m-6}(k)$$

$$h_{m-7}(k)$$

$$h_{m-8}(k)$$

$$h_{m-9}(k)$$

$$h_{m-10}(k)$$

$$h_{m-11}(k)$$

$$h_{m-12}(k)$$

$$h_{m-13}(k)$$

$$h_{m-14}(k)$$

$$h_{m-15}(k)$$

Почему множество этих функций универсально?

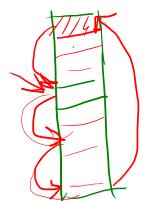
$$\exists k, l \in \mathbb{Z}_P \quad n \neq l$$
$$R = (ak+b) \bmod P$$
$$S = (al+b) \bmod P$$
$$R-S = (a(k-l)) \bmod P \Rightarrow R \neq S \quad \text{т.к. } a \neq 0$$

$$h_{m-16}(k)$$

## Хеш-таблицы с открытой адресацией

Все элементы хранятся в самой таблице

100 000

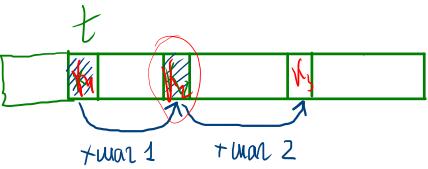


K<sub>1</sub>

$$h(K_1) = h(K_2) = t$$

Позволяет отказаться от указателей

Элементы с одинаковым хешем хранятся в виде **последовательностей** с определенным шагом



Хеш **стартовой** ячейки вычисляется в зависимости от **ключа**

Если ячейка **заполнена**, то проверяется **следующая с определенным шагом**

Шаг может быть как **константным**, так и нет.

$$\begin{aligned} h(K_1) &= t \\ h(K_2) &= t \end{aligned}$$

$\downarrow$

$t + l$

Требуется чтобы шаги формировали последовательность ячеек, являющуюся перестановкой всех ячеек

Запись:  $h(\text{ключ}, \text{номер исследования})$

$\langle h(k, 0), h(k, 1), h(k, 2), \dots, h(k, m-1) \rangle$  - перестановка  $\langle 0, 1, 2, \dots, m-1 \rangle$

$$\begin{aligned} h(K_1) &= t \\ h(K_2) &= t \end{aligned}$$

$\downarrow$

$t + 2l$

Операции:

HASH-INSERT( $T, k$ )

```

1  $i = 0$ 
2 repeat
3    $j = h(k, i)$ 
4   if  $T[j] == \text{NIL}$ 
5      $T[j] = k$ 
6     return  $j$ 
7   else  $i = i + 1$ 
8 until  $i == m$ 
9 error "переполнение хеш-таблицы"

```

HASH-SEARCH( $T, k$ )

```

1  $i = 0$ 
2 repeat
3    $j = h(k, i)$ 
4   if  $T[j] == k$ 
5     return  $j$ 
6    $i = i + 1$ 
7 until  $T[j] == \text{NIL}$  или  $i == m$ 
8 return NIL

```



Вставка и поиск похожи.



**Вставка:** идут по ячейкам исследования вплоть до попадания в **пустую** ячейку

**Поиск:** идут по ячейкам исследования вплоть до попадания в ячейку с **искомым значением**.

**Удаление** - пометить удаляемую ячейку как "Deleted".

В таком случае Hash-Insert рассматривает её как **пустую**, а Hash-Search как **заполненную** и не приостанавливает поиск на ней

$$L = \frac{n}{m}$$

! При использовании метки "Deleted" время поиска перестёт зависеть от **коэффициента заполненности**.

Если надо много удалять из таблицы, то метод цепочек (закрытая адресация) предпочтительней



**Про число последовательностей:** всего возможных перестановок  $m!$



Если есть возможность добиться  $m!$  перестановок, то создается иллюзия **равномерного хеширования**

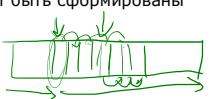
$$\leq 0, \dots, m-1 \geq m \cdot (m-1) \cdot (m-2) \dots = m!$$

## Методы исследования в хеш-таблицах с открытой адресацией

Различные последовательности  $\langle h(k,0), h(k,1), h(k,2), \dots h(k,m-1) \rangle$  могут быть сформированы следующими вариантами исследований:

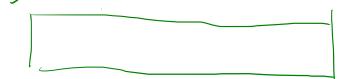
### 1. Линейное исследование

Вспомогательная хеш-функция:  $\exists h' : U \rightarrow \{0, \dots m-1\}$



Сама функция:  $h(k, i) = (h'(k) + i) \bmod m$

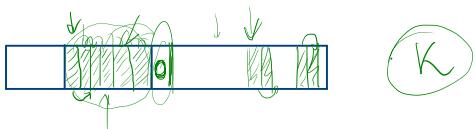
Порядок исследования:  $T[h'(u)] \rightarrow T[h'(u)+1] \rightarrow T[h'(u)+2]$



Всего последовательностей:  $m$

Проблемы: первичная кластеризация - длинные последовательности занятых ячеек => среднее время поиска увеличивается

Из-за чего возникают: вероятность того, что ячейка, которой предшествуют  $i$  заполненных будет заполнена



### 2. Квадратичное исследование

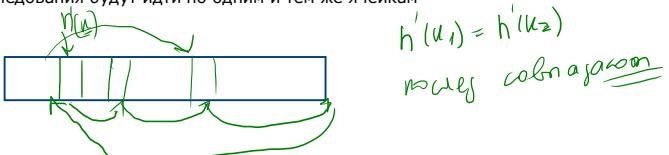
Функция:  $h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$   $c_1$  и  $c_2$  - члены

Лучше линейного, но выбор  $c_1, c_2$  и  $m$  затруднен

Порядок исследования:  $T[h'(u)] \rightarrow T[h'(u) + c_1 + c_2] \rightarrow T[h'(u) + 2c_1 + 4c_2] \dots$

Всего последовательностей:  $m$

Проблемы: вторичная кластеризация - если два ключа имеют одинаковое значение хеша, то и в дальнейшем их списки исследования будут идти по одним и тем же ячейкам



### 3. Двойное хеширование

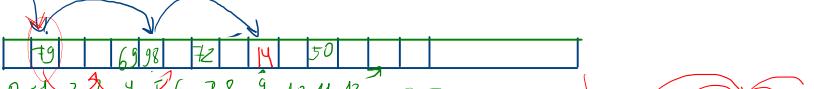
Наилучший способ использования открытой адресации, т.к. перестановки получаются похожими на случайные

Функция:  $h(k, i) = (h_1(k) + i h_2(k)) \bmod m$  беспорядок

Порядок исследования:  $T[h_1(k)] \rightarrow$  смещение  $i(h_2(k) \bmod m)$

Чтобы охватить всю таблицу  $h_2(k)$  взаимно просто с  $m$

Пример:



Вставка 14.

$$h(k, i) = ((k \bmod 13) + i(1 + (k \bmod 11))) \bmod 11$$

$$\begin{array}{ll} h_1 = 1 & h_2 = 1 + 3 = 4 \\ i = 0 & h(14, 0) = 1 \\ i = 1 & h(14, 1) = 5 \bmod 11 \\ i = 2 & h(14, 2) = 9 \bmod 11 \end{array}$$

Число последовательностей:  $m^2$

Как добиться:

1)  $m$  - степень двойки и  $h_2$  возвращает только нечетные значения

2)  $m$  - простое и  $h_2$  возвращает натуральные числа сплошь

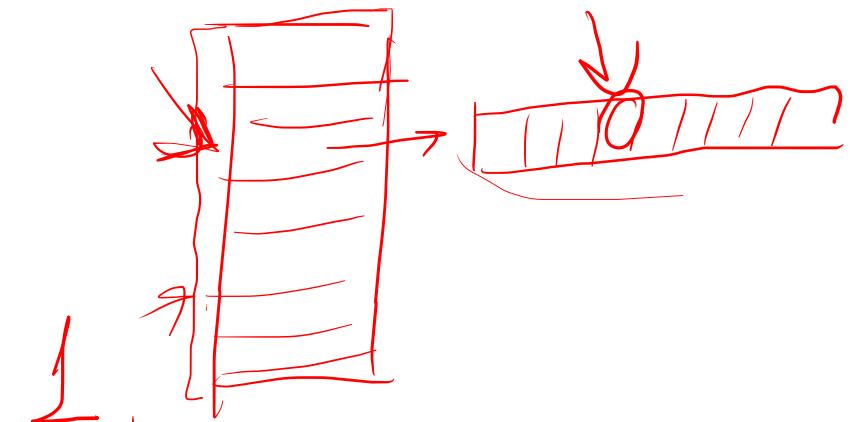
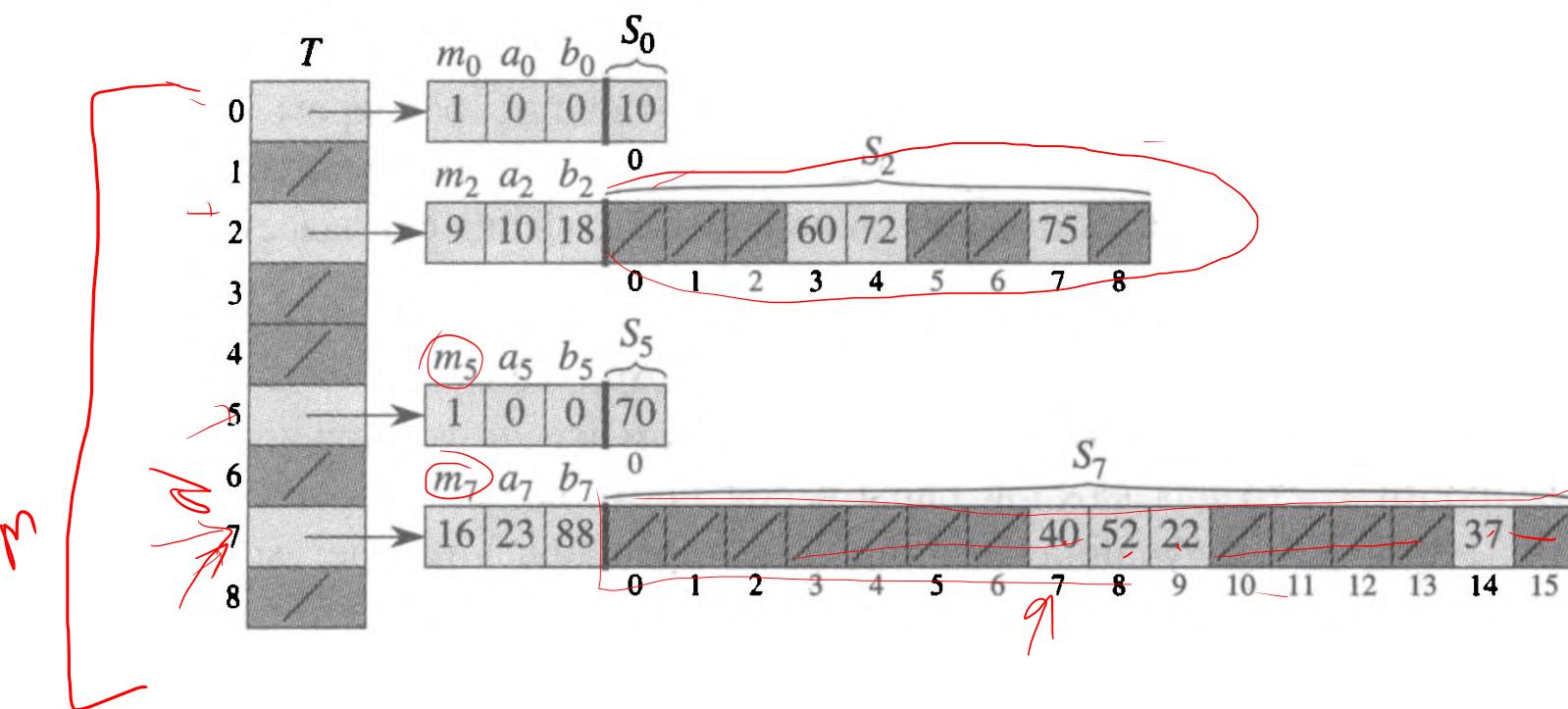
## Идеальное хеширование

Если определено статическое множество ключей

int

static

Пример: зарезервированы слова языка программирования или множество файлов на диске



2 уровня из множества универсальных хеш-функций:

Внешний:  $h(k) = ((ak + b) \bmod p) \bmod m$   $k_1, k_2$

$$h(k_1) = h(k_2)$$

Вторичный:  $h_j(k) = ((a_j k + b_j) \bmod p) \bmod m_j$

$n_j$  - число хешей  
указанных  $h(k) = j$

$$m_j = n^2$$