Хеши

что это вообще такое и зачем они нужны

Задача:

Дано множество студентов, у каждого из которых есть ФИО, возраст, курс, группа, etc. Вам дают двух студентов. Нужно выяснить, являются ли они одним и тем же человеком

Как решать?

Задача:

Дано множество студентов, у каждого из которых есть ФИО, возраст, курс, группа, etc. Вам дают двух студентов. Нужно выяснить, являются ли они одним и тем же человеком

Как решать?

- честно сравнить строки с данными - долго

Задача:

Дано множество студентов, у каждого из которых есть ФИО, возраст, курс, группа, etc. Вам дают двух студентов. Нужно выяснить, являются ли они одним и тем же человеком

Как решать?

- честно сравнить строки с данными - долго
- как-то отсортировать и искать совпадение бинпоиском - все еще долго

Задача:

Дано множество студентов, у каждого из которых есть ФИО, возраст, курс, группа, etc. Вам дают двух студентов. Нужно выяснить, являются ли они одним и тем же человеком

Логический вывод:

Все взаимодействия со строками напрямую - это долго

Как решать?

- честно сравнить строки с данными - долго
- как-то отсортировать и искать совпадение бинпоиском - все еще долго

Что мы умеем сравнивать быстрее, чем строки?

Что мы умеем сравнивать быстрее, чем строки?

Решение:

Давайте сопоставим каждому студенту какой-то номер (номер ису) и запишем это все в **c++ map**

Теперь при очередном запросе будем находить в словаре студентов **a** и **b** и сравнивать полученные по ним числа

Что мы умеем сравнивать быстрее, чем строки?

Решение:

Давайте сопоставим каждому студенту какой-то номер (номер ису) и запишем это все в **c++ unordered_map**

Теперь при очередном запросе будем находить в словаре студентов **a** и **b** и сравнивать полученные по ним числа

Асимптотика: O(1) на запрос

Хеш-функция

Хеш-функция - особое преобразование любого объема информации, в результате которого получается некое отображение, **образ**, называемый **хешем (hash)** - уникальное короткое значение, которое присуще только этому массиву входящей информации

Похоже по уникальности на:

- отпечаток пальца человека
- структуру ДНК человека

Свойства:

- уникальность
- необратимость
- изменяемость
- быстрая вычислимость

Хорошая хеш-функция

- Быстро вычисляется
- Минимизирует число коллизий (позже разберем, что это)

Хорошая:

- (x * const ^ a) % m

Плохая:

- Разложим число на простые делители, возьмем произведение нечетных (долго)
- (x * const ^ a) **% 2** (мало различных хешей)

Хеш-таблица

Хеш-таблица - структура данных, реализующая интерфейс **ассоциативного массива**. Представляет собой эффективную структуру данных для реализации словарей, а именно позволяет хранить пары (ключ, значение)

Хеш-таблица

Хеш-таблица - структура данных, реализующая интерфейс **ассоциативного массива**. Представляет собой эффективную структуру данных для реализации словарей, а именно позволяет хранить пары (ключ, значение)

Операции:

- **добавление** новой пары **O(1)**
- **поиск** значения по ключу **O(1)**
- **удаление** значения по ключу **O(1)**

Сравним с массивом

Массив (таблица с прямой адресацией):

- резервируется много неиспользуемой памяти (например, номера ису возрастают с каждым годом -> удаленный человек == неиспользованная память)
- ограничение по возможным типам индексов
- долгий поиск при неизвестных ключах **O(n)**

Хеш-таблица:

- требования к памяти снижаются до **M** - размер таблицы
- основа реализации ассоциативного массива
- все операции быстрые
- благодаря свойствам применима в большом кластере задач

Сравнение с другими структурами

Структура \ Операция	Вставка	Удаление	Поиск
Массив	O(n)	O(n)	O(n)
Список	O(1)	O(1)	O(n)
Отсортированный массив	O(n)	O(n)	O(logn)
Бинарное дерево поиска	O(logn)	O(logn)	O(logn)
Хеш-таблица	O(1)	O(1)	O(1)

Где используется?

- хранение паролей
- защита медиафайлов: защита от нелегального распространения
- используется для поимки вирусов
- защита от фальсификации передаваемой информации
- электронная подпись
- блокчейн
- ускорение поиска данных в БД

Известные алгоритмы:

SHA-1, SHA-2, SHA-3, SHA-256, MD6, MD5, MD4

Виды хеш-функций

Метод деления

hash(key) = key % m

- **key** - целое числовое значение ключа
- **m** - размер таблицы, простое число

Работа с символами (аддитивный)

while (*str): hash += (*str)++

- не различаются похожие слова и анаграммы, т е **hash(XY) = hash(YX)**
- **m = 256** (обычно)
- модификация: использовать побитовый **xor**

Виды хеш-функций

Метод умножения

hash(key) = (m * (key * A % 1))

- **0 < A < 1** - константа, рекомендуют золотое сечение:

  A = (√5 - 1) / 2 ≈ 0.618

- **m** - размер таблицы

Виды хеш-функций

Метод умножения

hash(key) = (m * (key * A % 1))

- **0 < A < 1** - константа, рекомендуют золотое сечение:

  A = (√5 - 1) / 2 ≈ 0.618

- **m** - размер таблицы

key = 123456

m = 10000

A = 0,6180339887499 = 0,618…

h(key) = [10000× ((123456×*0,618*…) mod 1)]

= [10000× (**76500,004**...×mod 1)] =

= [10000×**0,0041151**…] = [**41,151…**] = 41

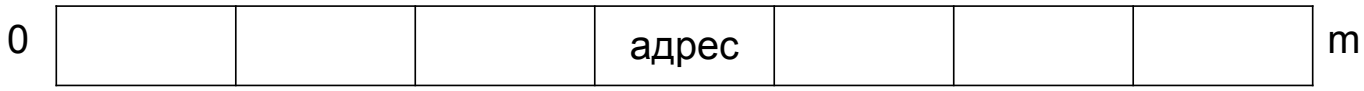
Виды хеш-функций

Универсальное хеширование

- подразумевает **случайный** выбор хеш-функции из некоторого множества во время **выполнения** программы
- например, случайные **A** в методе умножения

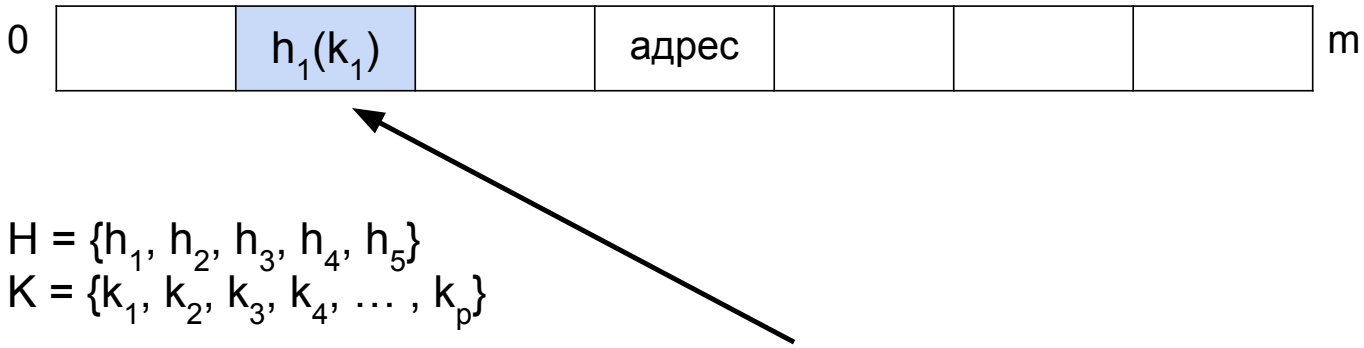
$H = \{h_1, h_2, h_3, h_4, h_5\}$

H - набор хеш-функций



$H = \{h_1, h_2, h_3, h_4, h_5\}$
$K = \{k_1, k_2, k_3, k_4, \dots, k_p\}$

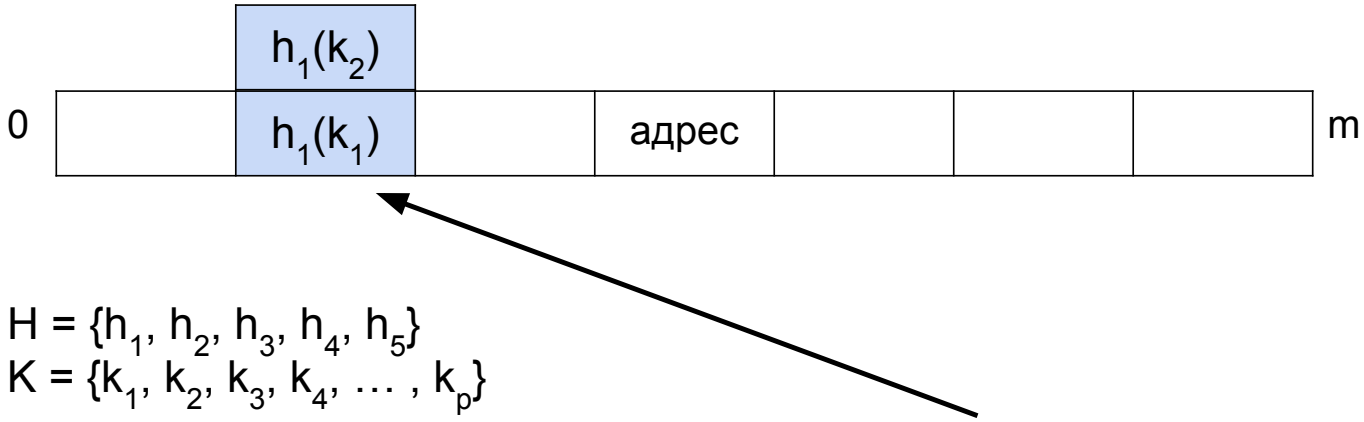
H - набор хеш-функций
K - набор данных для хеширования



$H = \{h_1, h_2, h_3, h_4, h_5\}$
$K = \{k_1, k_2, k_3, k_4, \dots, k_p\}$

H - набор хеш-функций
K - набор данных для хеширования

Хешируем k_1.
Теперь k_1 лежит по адресу $h_1(k_1)$

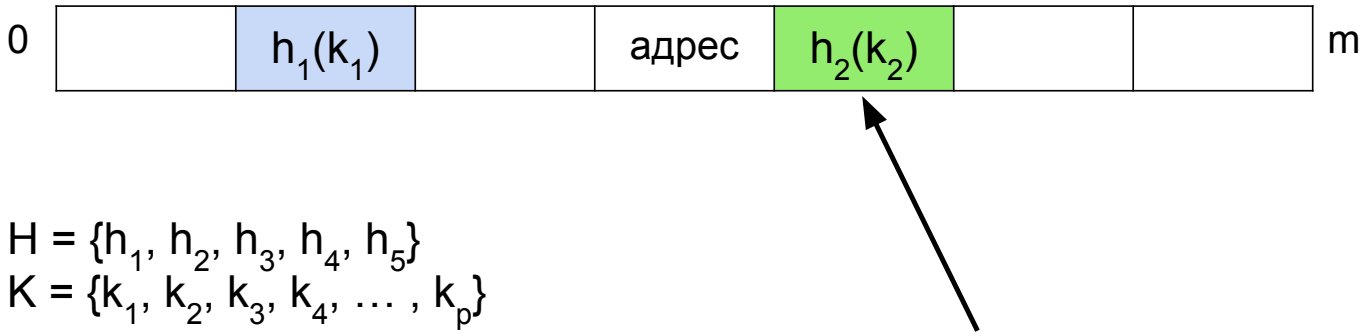


$H = \{h_1, h_2, h_3, h_4, h_5\}$
$K = \{k_1, k_2, k_3, k_4, \dots, k_p\}$

H - набор хеш-функций
K - набор данных для хеширования

Хешируем k_2, оно совпало с k_1

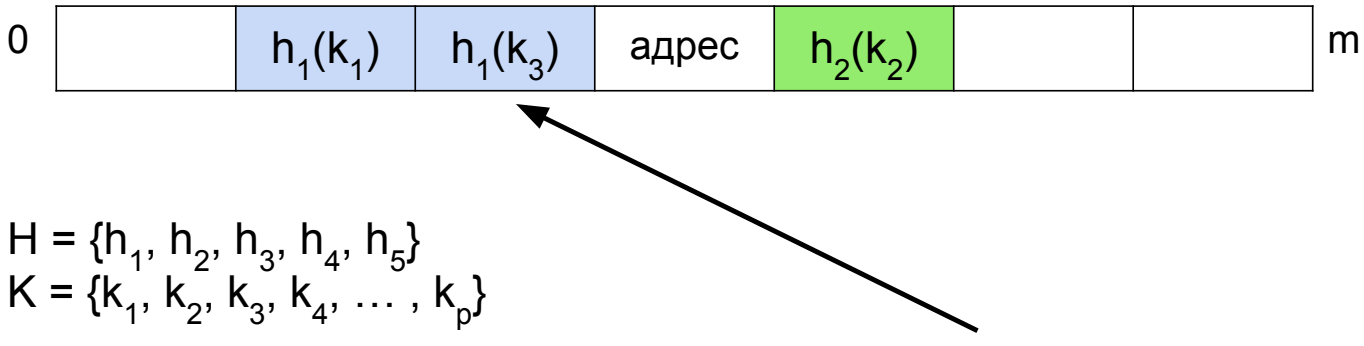
Что делать?



$H = \{h_1, h_2, h_3, h_4, h_5\}$
$K = \{k_1, k_2, k_3, k_4, \dots, k_p\}$

H - набор хеш-функций
K - набор данных для хеширования

Хешируем k_2 через h_2



$H = \{h_1, h_2, h_3, h_4, h_5\}$
$K = \{k_1, k_2, k_3, k_4, \dots, k_p\}$

H - набор хеш-функций
K - набор данных для хеширования

значения хеш-функций

$h_4(k_4)$	$h_1(k_1)$	$h_1(k_3)$	$h_3(k_5)$	$h_2(k_2)$	$h_1(k_6)$	$h_2(k_7)$
k_4	k_1	k_3	k_5	k_2	k_6	k_7

сами вставляемые значения

$H = \{h_1, h_2, h_3, h_4, h_5\}$
$K = \{k_1, k_2, k_3, k_4, \dots, k_p\}$

H - набор хеш-функций
K - набор данных для хеширования

Допустим, получилось что-то такое
Как теперь выполнить поиск нужного элемента?

$h_4(k_4)$	$h_1(k_1)$	$h_1(k_3)$	$h_3(k_5)$	$h_2(k_2)$	$h_1(k_6)$	$h_2(k_7)$
k_4	k_1	k_3	k_5	k_2	k_6	k_7

$H = \{h_1, h_2, h_3, h_4, h_5\}$
$K = \{k_1, k_2, k_3, k_4, \dots, k_p\}$

H - набор хеш-функций
K - набор данных для хеширования

Честно будем проверять всю последовательность хешей для поискового элемента, пока он не совпадет с тем, который лежит по адресу

Например, для поиска $\mathbf{k_4}$ пришлось бы проверить, что лежит по адресам $\mathbf{h_1(k_4)}$, $\mathbf{h_2(k_4)}$ **и** $\mathbf{h_3(k_4)}$. Элементы по этим адресам не совпадают с $\mathbf{k_4}$, а вот когда мы проверяем элемент по адресу $\mathbf{h_4(k_4)}$, он совпадает -> элемент найден

$h_4(k_4)$	$h_1(k_1)$	$h_1(k_3)$	$h_3(k_5)$	$h_2(k_2)$	$h_1(k_6)$	$h_2(k_7)$
k_4	k_1	k_3	k_5	k_2	k_6	k_7

$H = \{h_1, h_2, h_3, h_4, h_5\}$
$K = \{k_1, k_2, k_3, k_4, \dots, k_p\}$

H - набор хеш-функций
K - набор данных для хеширования

А что делать, если в H закончатся функции для хеширования?

Коллизии

$\exists\ x \neq y : hash(x) = hash(y)$

Как бороться с коллизиями?

Стараться предотвратить:

- искать другую хеш-функцию, использовать другие методы формирования
- подобрать более оптимальную, особенно если есть сведения о свойствах входящих данных или закономерностях

Методы разрешения коллизий:

- закрытая адресация (метод цепочек)
- открытая адресация и ее модификации

Хеш-функции: оптимизации

Метод середины квадрата:

- ключ возводится в квадрат (умножается сам на себя)
- в качестве индекса используются несколько цифр полученного значения

```
int h ( int  key ) {
key *= key;
key >>= 11; //
Отбрасываем 11 младших
бит
return key % 1024; //
Возвращаем 10 младших
бит
}
```

Например:

Ключом является целое 32-битное число, а хеш-функция возвращает средние 10 бит его квадрата

Хеш-функции: оптимизации

Полиномиальный хеш:

$\mathbf{hash_s = (s_1 + s_1 * k + s_2 * k^2 + \ldots + s_n * k^n) \% p}$

- $\mathbf{s_i}$ - код символа в строке
- **k** - произвольное число, большее размера алфавита
- **p** - размер таблицы, взаимно просто с **k**

k = 31
p = 1579
(int)'a' = 61
(int)'b' = 62
(int)'c' = 63

hash(abacaba) = $(61 + 62 * 31 + 61 * 31^2 + 63 * 31^3 + 61 * 31^4 + 62 * 31^5 + 61 * 31^6) \% 1579 =$ **62**

Хеш-функции: оптимизации

Полиномиальный хеш:

$$\mathbf{hash_s = (s_1 + s_1 * k + s_2 * k^2 + \ldots + s_n * k^n) \% p}$$

- $\mathbf{s_i}$ - код символа в строке
- **k** - произвольное число, большее размера алфавита
- **p** - размер таблицы, взаимно просто с **k**

k = 31
p = 1579
(int)'a' = 61
(int)'b' = 62
(int)'c' = 63

Можно писать степени в обратном порядке, от большей к меньшей, от этого незначительно меняется реализация

hash(abacaba) = $(61 + 62 * 31 + 61 * 31^2 + 63 * 31^3 + 61 * 31^4 + 62 * 31^5 + 61 * 31^6) \% 1579 =$ **62**

Хеш-функции: оптимизации

Универсальная строка

Для строки K, состоящей из L символов: $K = (x_1 x_2 x_3 \ldots x_L)$ можно предложить вычисление

$hash(K) = (h_1(x_1) + h_1(x_2) + \ldots + h_L(x_L))$

Хеш-функции: оптимизации

```
h := 0
for each c in C loop
    h := T[h xor c]
end loop
return h
```

Хеширование Пирсона

- использует 8-битный **шифр подстановки**, реализованный через 256-байтную таблицу перестановок
- предназначен для быстрого выполнения на процессорах с 8-разрядным регистром
- нет простого класса входов, для которых очень вероятны коллизии

Выше представлен псевдокод, который вычисляет хеш сообщения **C**, используя таблицу перестановок **T**

Виды хеш-таблиц

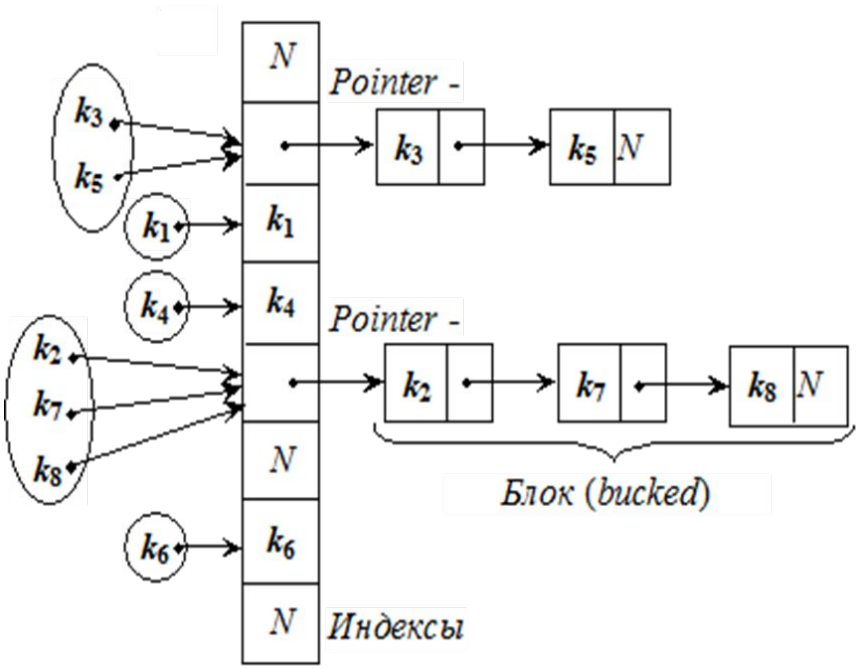
Закрытая адресация:

- в ячейках хранятся не сами элементы, а динамические списки элементов с соответствующим хешем
- более гибкий и хорошо расширяемый
- получив хеш, можно просто перебрать элементы списка и получить сразу все значения с таким хешем

Открытая адресация:

- в ячейках хранятся сами элементы
- при добавлении нового элемента пытаемся найти какую-то незанятую ячейку и записать в нее
- фиксированное потребление памяти
- при многочисленных коллизиях более низкая эффективность

Метод цепочек (закрытая адресация)



- используем динамические структуры данных (например, списки)
- исходная таблица может быть значительно меньше размера входных данных
- память выделяется по необходимости за счет дин структур
- менее чувствителен к типу ключей
- выгоднее при частом удалении
- удобно комбинировать структуры для оптимизации

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

значения хеш-функций

h(x)	0	1	2	3	4	5	6	7	8	9	10
x											

сами вставляемые значения

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		**23**									

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23									

1

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23									

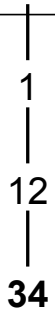
1

12

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23									



Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	**2**								

1

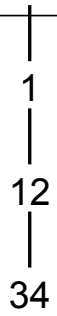
12

34

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

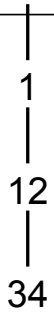
h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	2			**5**					



Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	2			5		**7**			



Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	**121**	23	2			5		7			

1

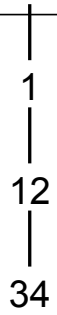
12

34

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

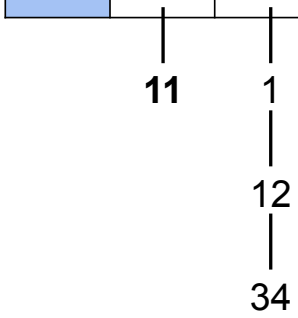
h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			**21**



Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

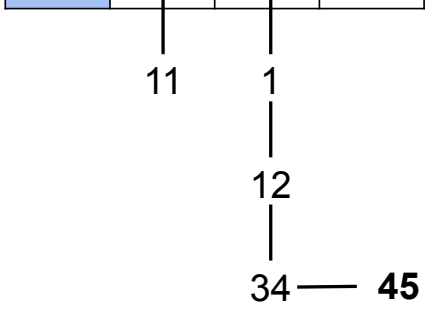
h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21



Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

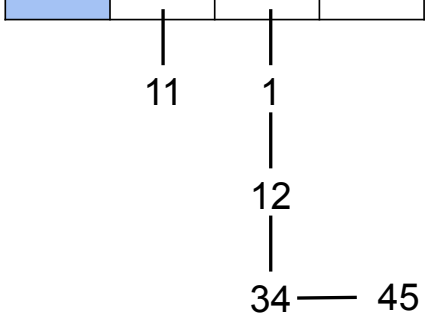
h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21



Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21

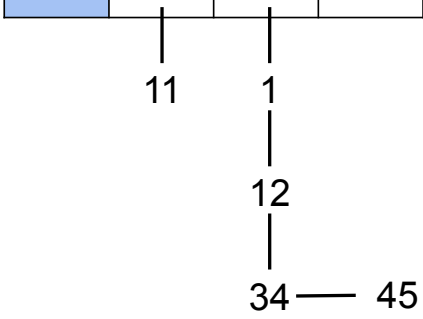


Как найти нужный элемент?

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21

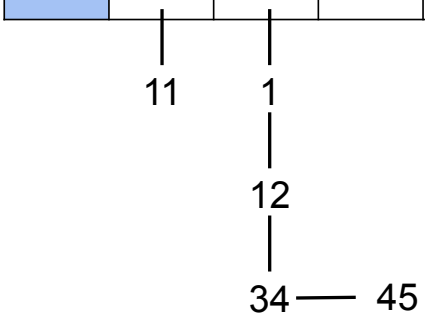


Допустим, хотим найти элемент **12**

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21



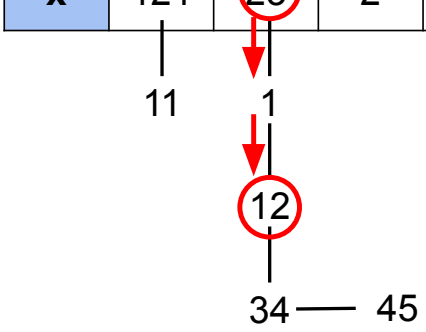
Допустим, хотим найти элемент **12**

h(12) = 12 % 11 = 1

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21



Допустим, хотим найти элемент **12**

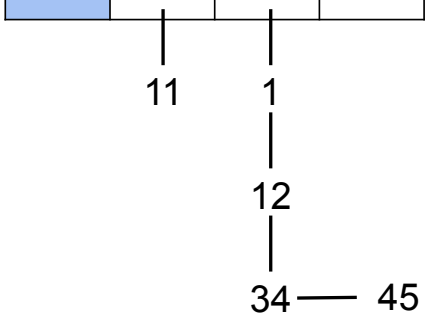
h(12) = 12 % 11 = 1

В ячейке с индексом **1** лежит указатель на голову списка -> честно проходимся по всему этому списку, пока не найдем **12** или список не закончится

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21

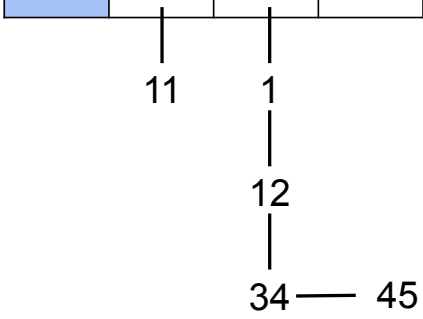


А что, если нужного элемента нет?

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21

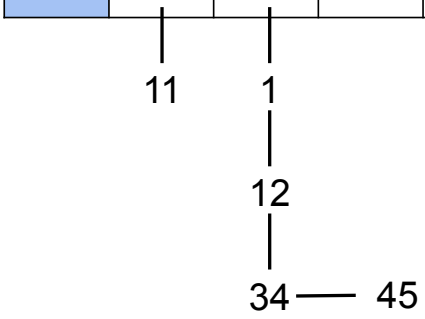


Допустим, хотим найти элемент **56**

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21



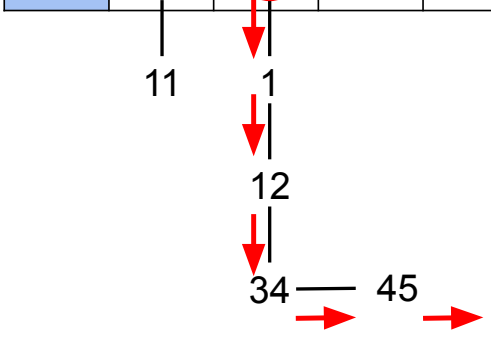
Допустим, хотим найти элемент **56**

h(56) = 56% 11 = 1

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21



Допустим, хотим найти элемент **56**

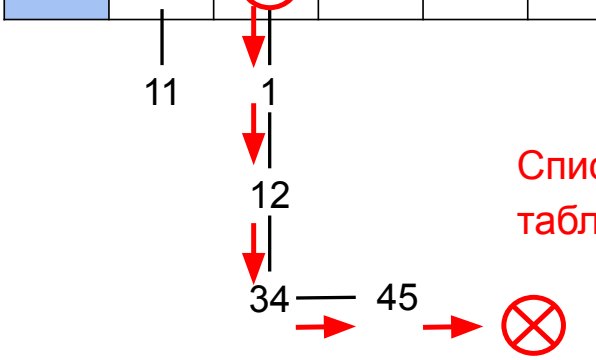
h(56) = 56 % 11 = 1

В ячейке с индексом **1** лежит указатель на голову списка -> честно проходимся по всему этому списку, пока не найдем **56** или список не закончится

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21

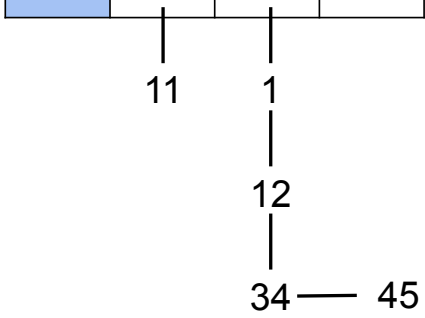


Список закончился -> элемента **56** в нашей таблице нет

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21

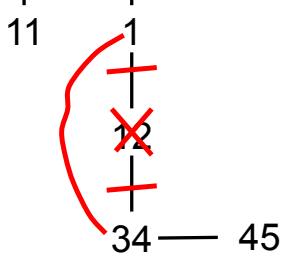


А как удалить?

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21

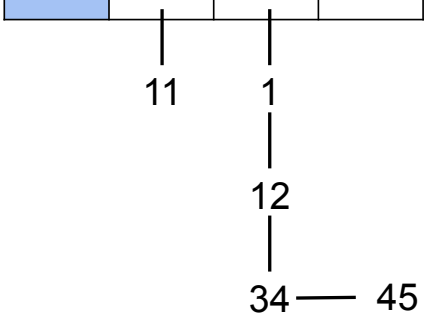


За счет того, что мы используем списки, удаление == поиск + замена указателей

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21

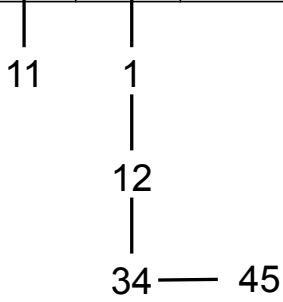


За сколько работают операции вставки, удаления и поиска?

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21



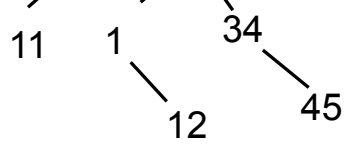
- Вставка: **O(1)** (если хранить указатель на хвост)
- Удаление: **O(1) -> O(n)** из-за поиска
- Поиск: **O(n)** в худшем случае (когда все элементы имеют одинаковый хеш)

Как модифицировать?

Метод цепочек: пример реализации

Задана хеш-функция **h(x) = x % 11**. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2			5		7			21

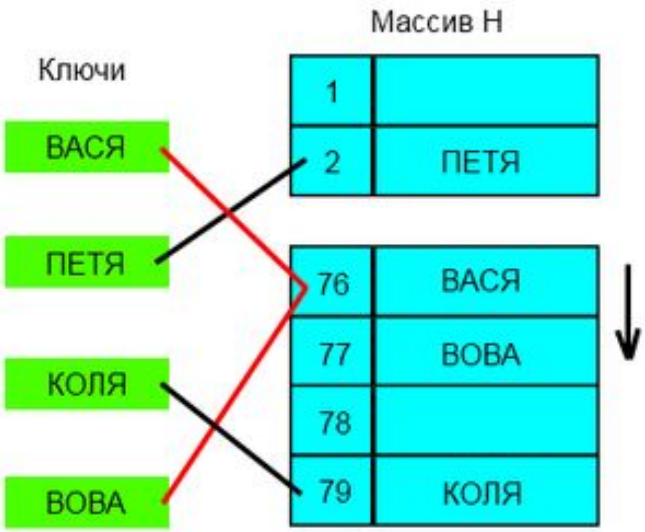


- Вставка: **O(1)**
- Удаление: **O(1) -> O(log(n))** из-за поиска
- Поиск: **O(log(n))** в худшем случае, но при достаточно равномерно распределенных данных может быть **O(1)**

Сделаем вместо обычного списка какое-нибудь дерево поиска

Методы открытой адресации

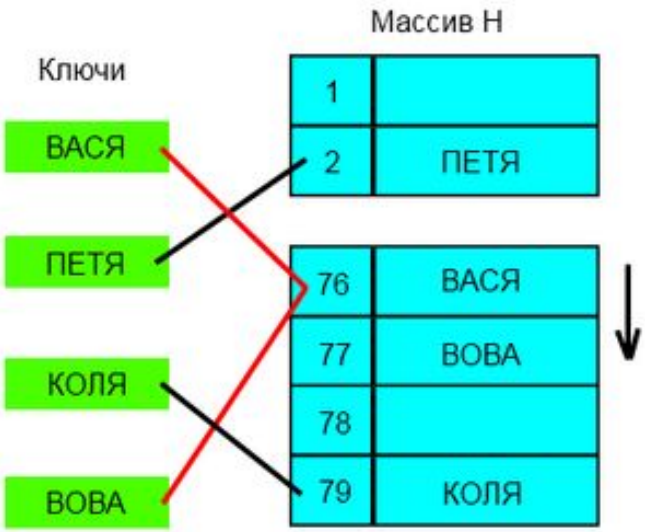
Все элементы хранятся непосредственно в хеш-таблице



- таблица фиксирована и непрерывна
- исследование быстрее
- большой выбор методов исследования: много оптимальных методов, выведенных опытным путем
- проще в реализации и внедрении доп ограничений
- нет “висячих” ссылок и сложностей с управлением дин структурами

Методы открытой адресации

Все элементы хранятся непосредственно в хеш-таблице



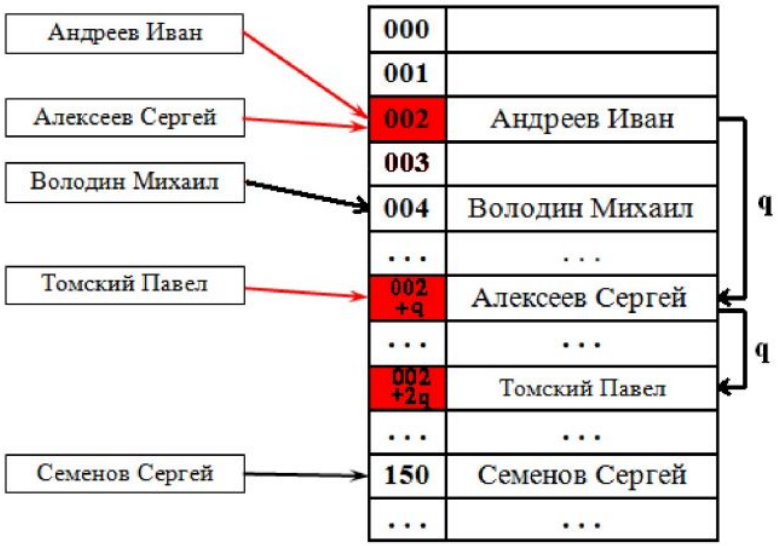
Методы исследований:

- последовательный
- линейный
- квадратичный
- двойное хеширование
- метод “кукушки”

Линейный метод

Линейный метод

При попытке добавить элемент в занятую ячейку начинаем последовательно просматривать ячейки по принципу
h(key, i) = (h(key) + i * q) % m, где **i** - номер попытки от **0** до **m - 1**



Просматриваем ячейки:

- h(key) + q, **i = 1**
- h(key) + 2q, **i = 2**
- h(key) + 3q, **i = 3**
- …

Нашли свободную ячейку **->** записали элемент

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей:

23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45

h(x)	0	1	2	3	4	5	6	7	8	9	10
x											

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		**23**									

h(23) = 1

Ячейка с индексом **1** свободна -> вставляем в нее

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23			**1**						

h(1) = 1

Ячейка с индексом **1** занята -> …

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23			**1**						

h(1) = 1

Ячейка с индексом **1** занята -> берем ячейку с индексом
(h(1) + 3) % 11 = 4 -> свободна -> вставляем

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23			1			**12**			

h(12) = 1

Ячейка с индексом **1** занята -> …

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23			1			**12**			

h(12) = 1

Ячейка с индексом **1** занята -> берем ячейку с индексом **(h(12) + 3) % 11 = 4** -> занята -> …

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23			1			**12**			

h(12) = 1

Ячейка с индексом **1** занята -> берем ячейку с индексом
(h(12) + 3) % 11 = 4 -> занята -> берем ячейку с индексом
(h(12) + 3 * 2) % 11 = 7 -> свободна -> вставляем

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23			1			12			**34**

h(34) = 1

Ячейка с индексом **1** занята -> …

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23			1			12			**34**

h(34) = 1

Ячейка с индексом **1** занята -> берем ячейку с индексом **(h(34) + 3) % 11 = 4** -> занята -> …

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23			1			12			**34**

h(34) = 1

Ячейка с индексом **1** занята -> берем ячейку с индексом
(h(34) + 3) % 11 = 4 -> занята -> берем ячейку с индексом
(h(34) + 3 * 2) % 11 = 7 -> занята -> …

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23			1			12			**34**

h(34) = 1

Ячейка с индексом **1** занята -> берем ячейку с индексом
(h(34) + 3) % 11 = 4 -> занята -> берем ячейку с индексом
(h(34) + 3 * 2) % 11 = 7 -> занята -> берем ячейку с индексом
(h(34) + 3 * 3) % 11 = 10 -> свободна -> вставляем

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	**2**		1			12			34

h(2) = 2

Ячейка с индексом **2** свободна -> вставляем

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	2		1	**5**		12			34

h(5) = 5

Ячейка с индексом **5** свободна -> вставляем

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	2		1	5		12	**7**		34

h(7) = 7

Ячейка с индексом **7** занята -> …

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	2		1	5		12	**7**		34

h(7) = 7

Ячейка с индексом **7** занята -> берем ячейку с индексом **(h(7) + 3) % 11 = 10** -> занята -> …

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	2		1	5		12	**7**		34

h(7) = 7

Ячейка с индексом **7** занята -> берем ячейку с индексом
(h(7) + 3) % 11 = 10 -> занята -> берем ячейку с индексом
(h(7) + 3 * 2) % 11 = 2 -> занята -> …

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	2		1	5		12	**7**		34

h(7) = 7

Ячейка с индексом **7** занята -> берем ячейку с индексом
(h(7) + 3) % 11 = 10 -> занята -> берем ячейку с индексом
(h(7) + 3 * 2) % 11 = 2 -> занята -> берем ячейку с индексом
(h(7) + 3 * 3) % 11 = 5 -> занята -> …

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	2		1	5		12	**7**		34

h(7) = 7

Ячейка с индексом **7** занята -> берем ячейку с индексом
(h(7) + 3) % 11 = 10 -> занята -> берем ячейку с индексом
(h(7) + 3 * 2) % 11 = 2 -> занята -> берем ячейку с индексом
(h(7) + 3 * 3) % 11 = 5 -> занята -> берем ячейку с индексом
(h(7) + 3 * 4) % 11 = 8 -> свободна -> вставляем

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	**121**	23	2		1	5		12	7		34

h(121) = 0

Ячейка с индексом **0** свободна -> вставляем

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2	**21**	1	5		12	7		34

h(21) = 10

Ячейка с индексом **10** занята -> …

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2	**21**	1	5		12	7		34

h(21) = 10

Ячейка с индексом **10** занята -> берем ячейку с индексом
(h(21) + 3) % 11 = 2 -> занята -> берем ячейку с индексом
(h(21) + 3 * 2) % 11 = 2 -> занята ->
… -> берем ячейку с индексом
(h(21) + 3 * 6) % 11 = 3 -> свободна -> вставляем

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2	21	1	5	**11**	12	7		34

h(11) = 0

Ячейка с индексом **0** занята -> …

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2	21	1	5	**11**	12	7		34

h(11) = 0

Ячейка с индексом **0** занята -> берем ячейку с индексом
(h(11) + 3) % 11 = 3 -> занята -> берем ячейку с индексом
(h(11) + 3 * 2) % 11 = 6 -> свободна -> вставляем

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2	21	1	5	11	12	7	**45**	34

h(45) = 1

Ячейка с индексом **1** занята -> …

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2	21	1	5	11	12	7	**45**	34

h(45) = 1

Ячейка с индексом **1** занята -> берем ячейку с индексом
(h(45) + 3) % 11 = 4 -> занята -> берем ячейку с индексом
(h(45) + 3 * 2) % 11 = 7 -> занята ->
… -> берем ячейку с индексом
(h(45) + 3 * 10) % 11 = 9 -> свободна -> вставляем

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2	21	1	5	11	12	7	45	34

Ура, мы заполнили всю таблицу!

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2	21	1	5	11	12	7	45	34

Ура, мы заполнили всю таблицу!

А почему у нас это получилось?

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2	21	1	5	11	12	7	45	34

Ура, мы заполнили всю таблицу!

А почему у нас это получилось?
Правда ли, что нам просто повезло, и числа попались хорошие?

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2	21	1	5	11	12	7	45	34

Ответ: нет, неправда

У нас получилось записать все числа в таблицу за счет того, что размер таблицы и число **q** **взаимно простые**
-> не получается циклов при подсчете очередного индекса

ПОИСК

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2	21	1	5	11	12	7	45	34

Поиск выполняется по тому же методу, что и в закрытой адресации

Линейный метод: пример работы

Снова задана хеш-функция **h(x) = x % 11, h(x, i) = (h(x) + i * 3) % 11**.
Выполните вставку следующего списка ключей: **23**, **1**, **12**, **34**, **2**, **5**, **7**, **121**, **21**, **11**, **45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	2	21	1	5	11	12	7	45	34

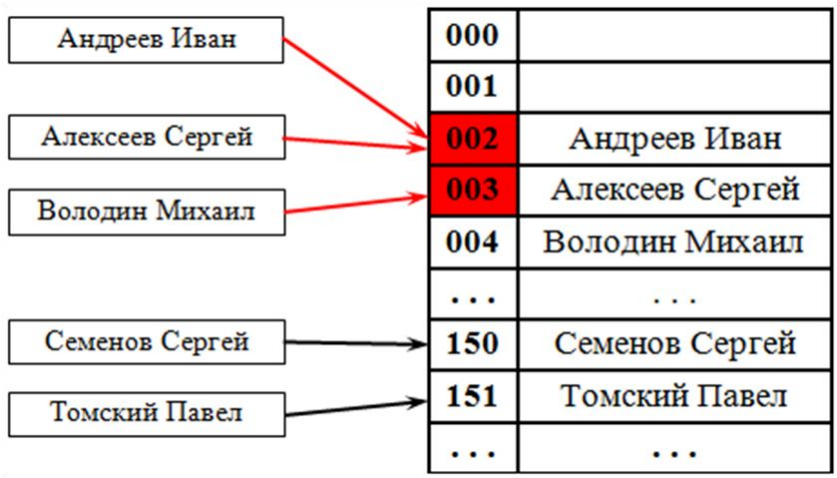
Поиск выполняется по тому же методу, что и в закрытой адресации

Хотим найти элемент -> считаем по очереди по формуле все хеши и сравниваем нужный элемент с тем, который лежит по полученному адресу

Последовательный метод исследования

Последовательный метод

То же самое, что линейный метод, но **q = 1**



Просматриваем ячейки:

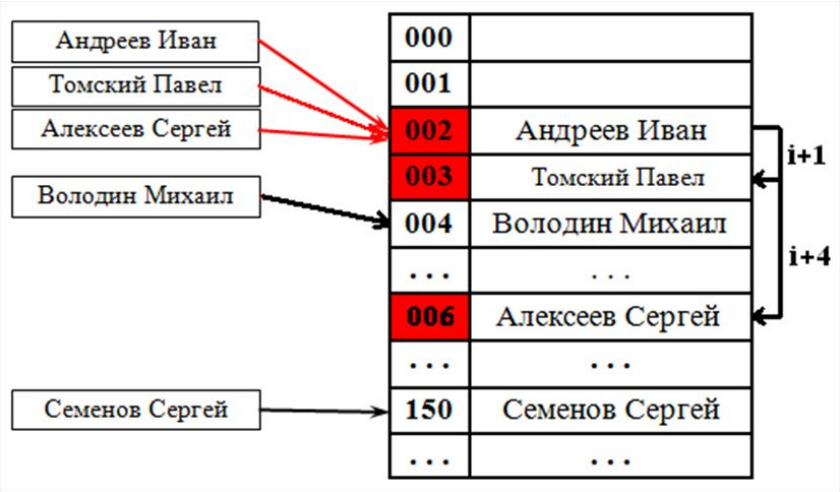
- h(key) + i, i = 1
- h(key) + i, i = 2
- h(key) + i, i = 3
- …

Нашли свободную ячейку -> записали элемент

Квадратичный метод исследования

Квадратичный метод

При попытке добавить элемент в занятую ячейку начинаем последовательно просматривать ячейки по принципу **h(key, i) = (h(key) + c1 * i + c2 * i^2) % m**, где **i** - номер попытки от **0** до **m - 1**, по умолчанию **c1, c2 = 1**



Просматриваем ячейки:

- h(key) + i^2, **i = 1**
- h(key) + i^2, **i = 2**
- h(key) + i^2, **i = 3**
- …

Нашли свободную ячейку -> записали элемент

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x											

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		**23**									

h(23) = 1

Ячейка с индексом **1** свободна -> вставляем в нее

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	**1**								

h(1) = 1

Ячейка с индексом **1** занята -> …

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	**1**								

h(1) = 1

Ячейка с индексом **1** занята -> берем ячейку с индексом **(h(1) + 1) % 11 = 2** -> свободна -> вставляем

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	1			**12**					

h(12) = 1

Ячейка с индексом **1** занята ->...

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	1			**12**					

h(12) = 1

Ячейка с индексом **1** занята -> берем ячейку с индексом
(h(12) + 1) % 11 = 2 -> занята -> берем ячейку с индексом
(h(12) + 4) % 11 = 5 свободна -> вставляем

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	1			12					**34**

h(34) = 1

Ячейка с индексом **1** занята -> …

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	1			12					**34**

h(34) = 1

Ячейка с индексом **1** занята -> берем ячейку с индексом
(h(12) + 1) % 11 = 2 -> занята -> берем ячейку с индексом
(h(12) + 4) % 11 = 5 -> занята -> берем ячейку с индексом
(h(12) + 9) % 11 = 10 -> свободна -> вставляем

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	1	**2**		12					34

h(2) = 2

Ячейка с индексом **2** занята -> …

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	1	**2**		12					34

h(2) = 2

Ячейка с индексом **2** занята -> берем ячейку с индексом **(h(2) + 1) % 11 = 3** -> свободна -> вставляем

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	1	2		12	**5**				34

h(5) = 5

Ячейка с индексом **5** занята -> берем ячейку с индексом **(h(5) + 1) % 11 = 6** -> свободна -> вставляем

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x		23	1	2		12	5	**7**			34

h(7) = 7

Ячейка с индексом **7** свободна -> вставляем

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	**121**	23	1	2		12	5	7			34

h(121) = 0

Ячейка с индексом **0** свободна -> вставляем

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	1	2		12	5	7	**21**		34

h(21) = 10

Ячейка с индексом **10** занята -> берем ячейку с индексом
(h(21) + 1) % 11 = 0 -> занята -> берем ячейку с индексом
(h(21) + 4) % 11 = 3 -> занята -> берем ячейку с индексом
(h(21) + 9) % 11 = 8 -> свободна -> вставляем

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	1	2	**11**	12	5	7	21		34

h(11) = 0

Ячейка с индексом **0** занята -> берем ячейку с индексом
(h(11) + 1) % 11 = 1 -> занята -> берем ячейку с индексом
(h(11) + 4) % 11 = 4 -> свободна -> вставляем

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	1	2	11	12	5	7	21		34

Можно проверить достаточно простым кодом, что даже при **i = 10000** элемент **45** не встает никуда в данной таблице.

Квадратичный метод: пример работы

В очередной раз задана хеш-функция

h(x) = x % 11, h(x, i) = (h(x) + i^2) % 11, c1 = 0, c2 = 1. Выполните вставку следующего списка ключей: **23, 1, 12, 34, 2, 5, 7, 121, 21, 11, 45**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	1	2	11	12	5	7	21		34

Можно проверить достаточно простым кодом, что даже при **i = 10000** элемент **45** не встает никуда в данной таблице.

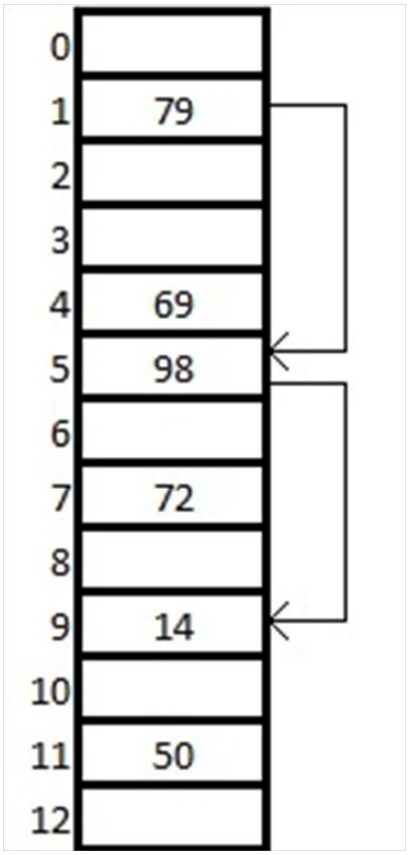
Эта проблема решается ***перехешированием***, которое мы разберем чуть позже…

Двойное хеширование

Двойное хеширование

Метод основан на использовании двух независимых хеш-функций

h(key) = (h1(key) + i * h2(key)) % m, где **i** - номер попытки от **0** до **m - 1**

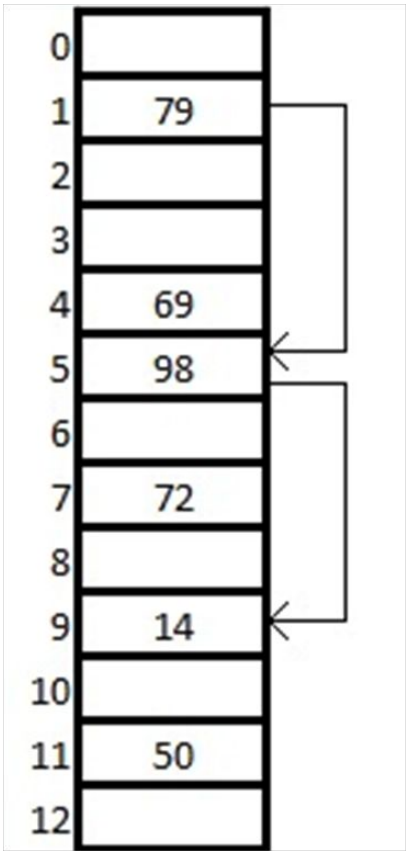


Двойное хеширование

Метод основан на использовании двух независимых хеш-функций

h(key) = (h1(key) + i * h2(key)) % m, где **i** - номер попытки от **0** до **m - 1**

- **h1** - адрес начала исследования

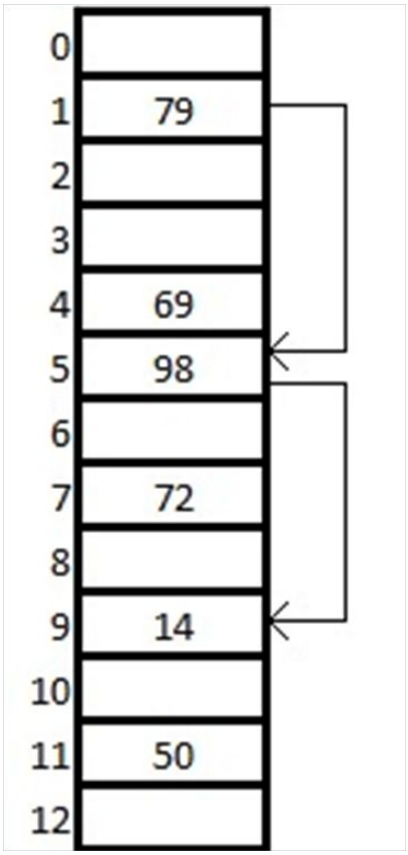


Двойное хеширование

Метод основан на использовании двух независимых хеш-функций

h(key) = (h1(key) + i * h2(key)) % m, где **i** - номер попытки от **0** до **m - 1**

- **h1** - адрес начала исследования
- **h2** - шаг исследования, индивидуальный для каждого ключа

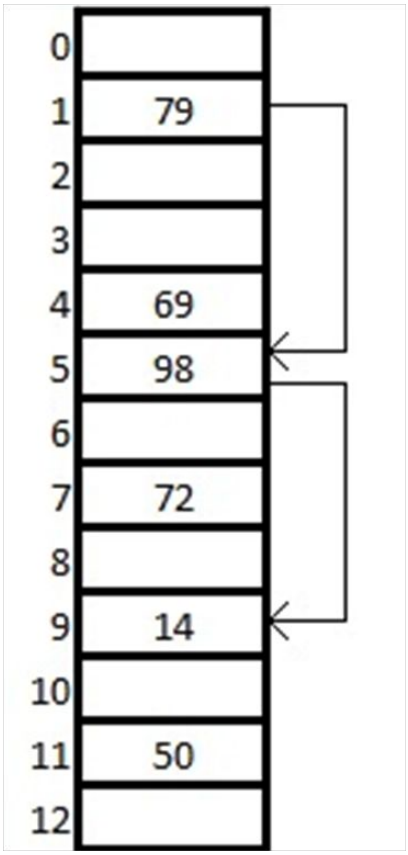


Двойное хеширование

Метод основан на использовании двух независимых хеш-функций

h(key) = (h1(key) + i * h2(key)) % m, где **i** - номер попытки от **0** до **m - 1**

- **h1** - адрес начала исследования
- **h2** - шаг исследования, индивидуальный для каждого ключа
- **h1, h2** обязательно **независимы** друг от друга

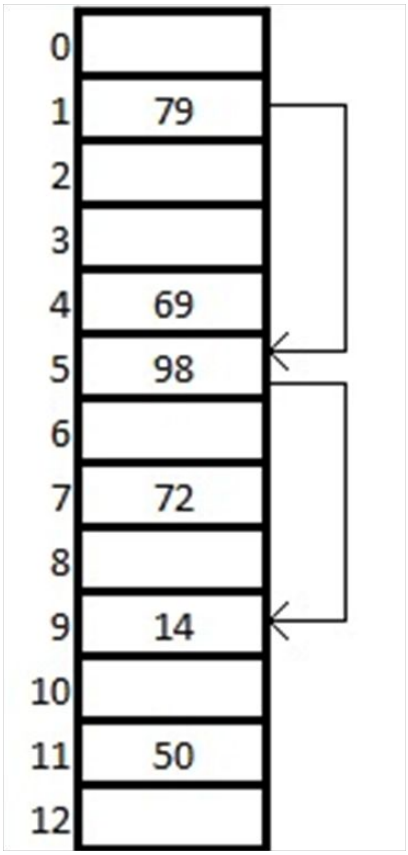


Двойное хеширование

Метод основан на использовании двух независимых хеш-функций

h(key) = (h1(key) + i * h2(key)) % m, где **i** - номер попытки от **0** до **m - 1**

- **h1** - адрес начала исследования
- **h2** - шаг исследования, индивидуальный для каждого ключа
- **h1, h2** обязательно независимы друг от друга
- **h1** - обычная хеш-функция, для того, чтобы последовательность исследования могла охватить всю таблицу, **h2** должна возвращать значения:
  - не равные **0**
  - независимые от **h1**
  - взаимно простые с величиной хеш-таблицы



Двойное хеширование

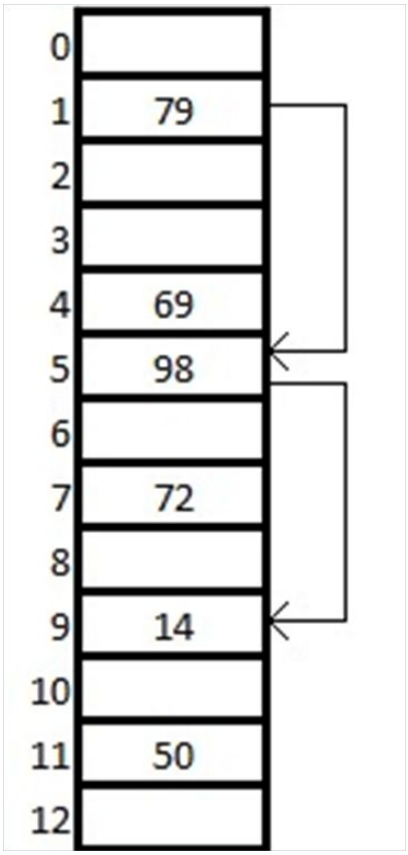
Метод основан на использовании двух независимых хеш-функций

h(key) = (h1(key) + i * h2(key)) % m, где **i** - номер попытки от **0** до **m - 1**

Плюс этого алгоритма в том, что даже при высокой заполненности таблицы при хорошей функции **h2** не случится циклов. Более того, количество шагов до вставки очередного элемента достаточно мало вне зависимости от распределения и входных данных.

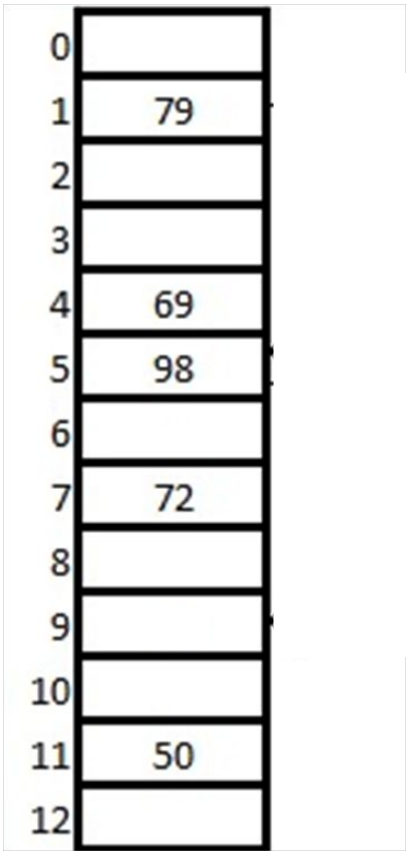
В частности, мат. ожидание числа шагов до вставки нового элемента = **1 / (1 - n / m)**, где

- **n** - число уже вставленных элементов
- **m** - размер таблицы



Двойное хеширование: пример

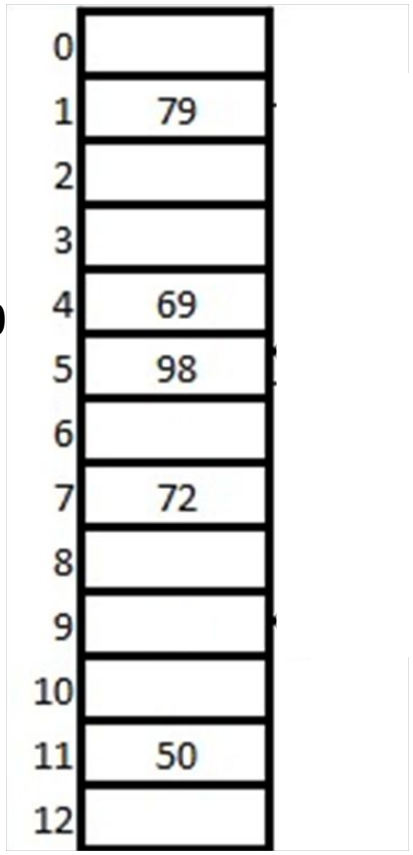
- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**



Двойное хеширование: пример

- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**

Пусть мы хотим вставить ключ **14**. Изначально **i = 0**

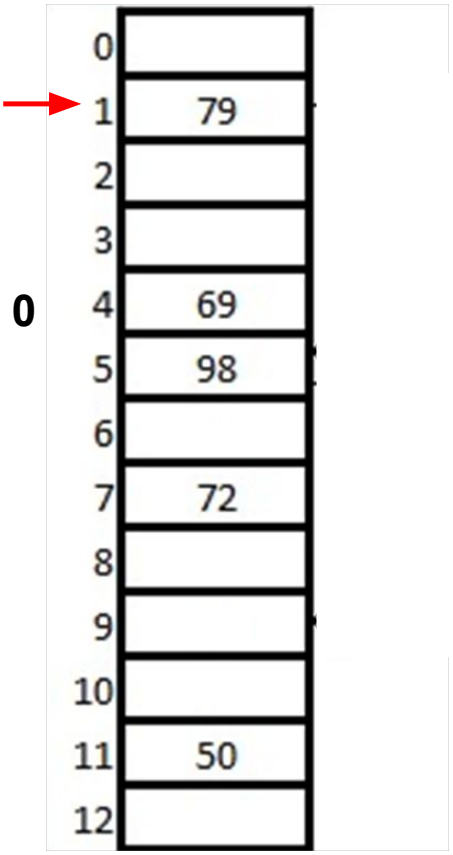


Двойное хеширование: пример

- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**

Пусть мы хотим вставить ключ **14**. Изначально **i = 0**

h(14, 0) = (h1(14) + 0 * h2(14)) % 13 = 1



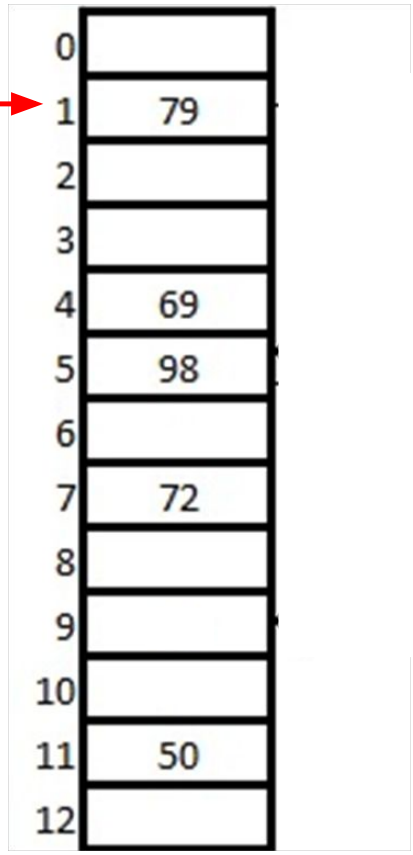
Двойное хеширование: пример

- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**

Пусть мы хотим вставить ключ **14**. Изначально **i = 0**

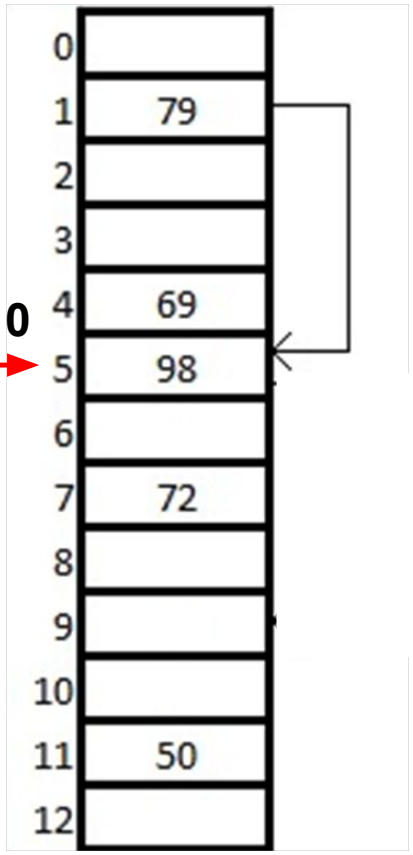
h(14, 0) = (h1(14) + 0 * h2(14)) % 13 = 1

Но ячейка с номером **1** уже занята, **i++**



Двойное хеширование: пример

- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**



Пусть мы хотим вставить ключ **14**. Изначально **i = 0**

h(14, 0) = (h1(14) + 0 * h2(14)) % 13 = 1

Но ячейка с номером **1** уже занята, **i++**

При **i = 1**, **h(14, 1) = 5**

Ячейка с номером **5** тоже занята, **i++**

Двойное хеширование: пример

- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**

Пусть мы хотим вставить ключ **14**. Изначально **i = 0**

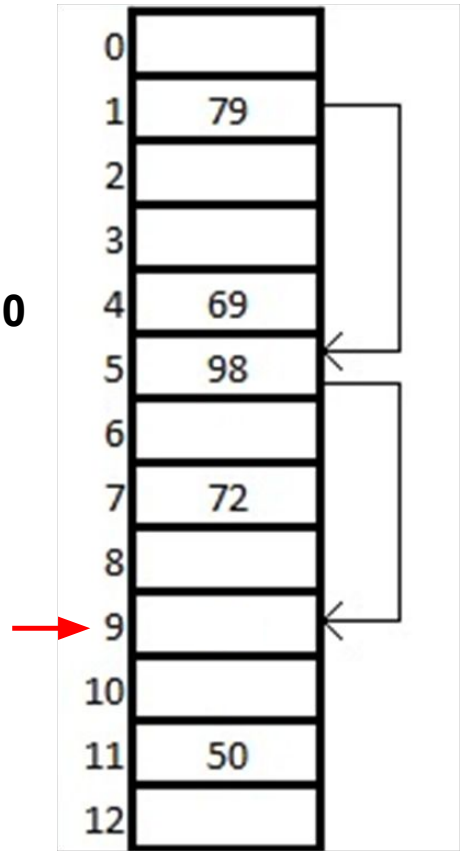
h(14, 0) = (h1(14) + 0 * h2(14)) % 13 = 1

Но ячейка с номером **1** уже занята, **i++**

При **i = 1, h(14, 1) = 5**

Ячейка с номером **5** тоже занята, **i++**

При **i = 2, h(14, 2) = (h1(14) + 2 * h2(14)) % 13 = 9**



Двойное хеширование: пример

- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**

Пусть мы хотим вставить ключ **14**. Изначально **i = 0**

h(14, 0) = (h1(14) + 0 * h2(14)) % 13 = 1

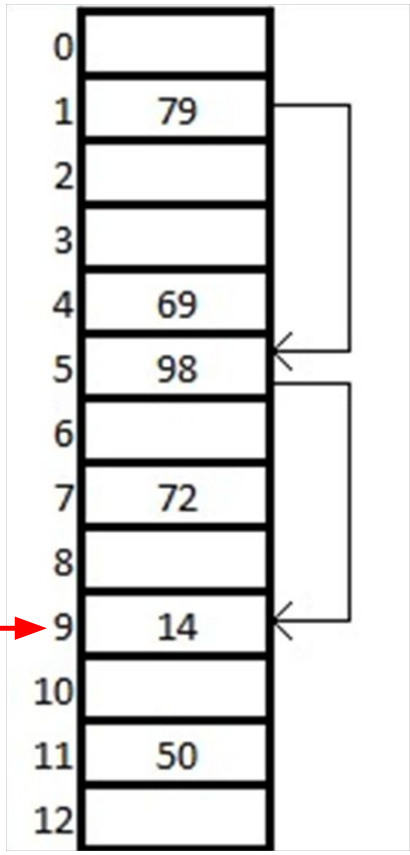
Но ячейка с номером **1** уже занята, **i++**

При **i = 1, h(14, 1) = 5**

Ячейка с номером **5** тоже занята, **i++**

При **i = 2, h(14, 2) = (h1(14) + 2 * h2(14)) % 13 = 9**

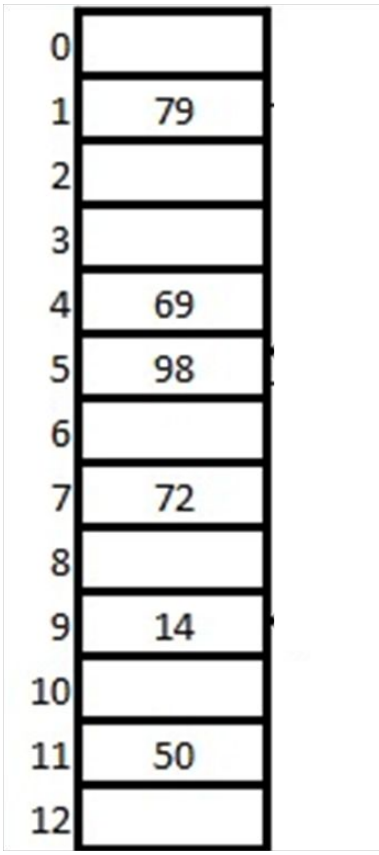
Ячейка с номером **9 свободна** -> можем вписать туда **14**



Двойное хеширование: пример

- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**

Пусть мы хотим вставить ключ **17**. Изначально **i = 0**



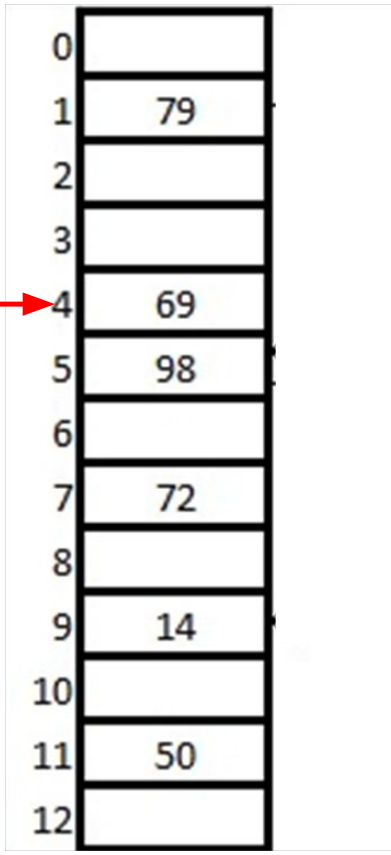
Двойное хеширование: пример

- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**

Пусть мы хотим вставить ключ **17**. Изначально **i = 0**

h(17, 0) = (h1(17) + 0 * h2(17)) % 13 = 4

Но ячейка с номером **4** уже занята, **i++**



Двойное хеширование: пример

- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**

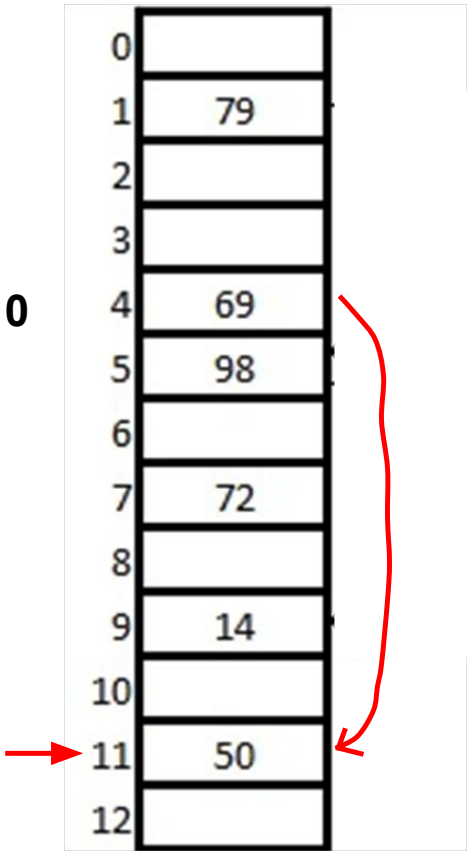
Пусть мы хотим вставить ключ **17**. Изначально **i = 0**

h(17, 0) = (h1(17) + 0 * h2(17)) % 13 = 4

Но ячейка с номером **4** уже занята, **i++**

При **i = 1**, **h(17, 1) = 11**

Ячейка с номером **11** тоже занята, **i++**



Двойное хеширование: пример

- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**

Пусть мы хотим вставить ключ **17**. Изначально **i = 0**

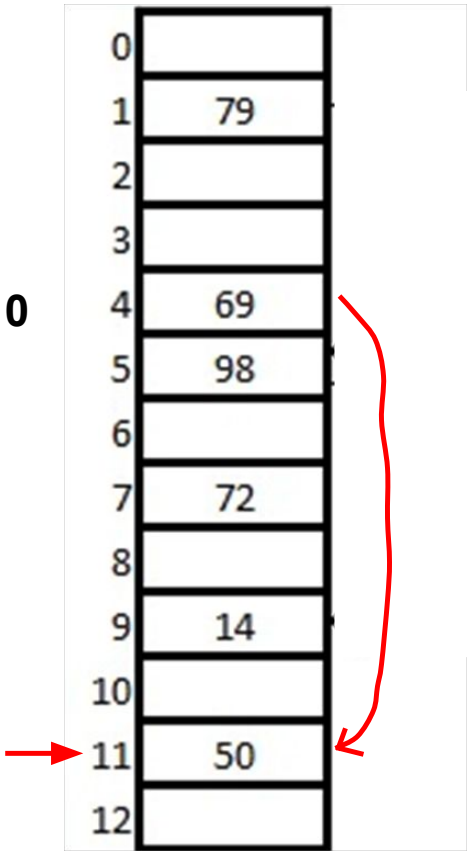
h(17, 0) = (h1(17) + 0 * h2(17)) % 13 = 4

Но ячейка с номером **4** уже занята, **i++**

При **i = 1**, **h(17, 1) = 11**

Ячейка с номером **11** тоже занята, **i++**

Как можно заметить, шаг при вставке поменялся относительно предыдущего элемента, потому что изменилось значение h2(key)



Двойное хеширование: пример

- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**

Пусть мы хотим вставить ключ **17**. Изначально **i = 0**

h(17, 0) = (h1(17) + 0 * h2(17)) % 13 = 4

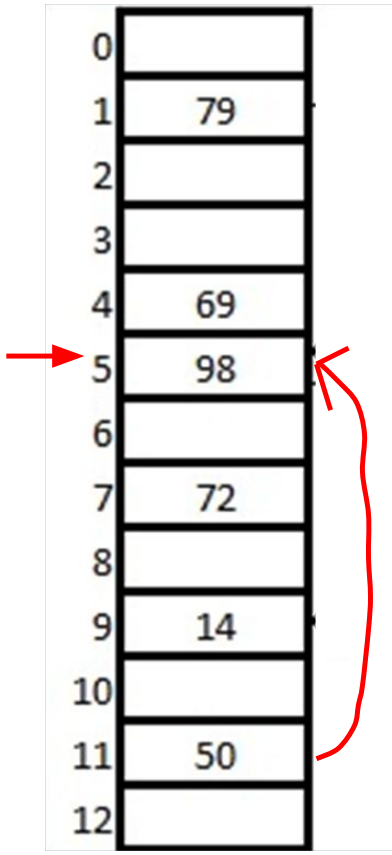
Но ячейка с номером **4** уже занята, **i++**

При **i = 1, h(17, 1) = 11**

Ячейка с номером **11** тоже занята, **i++**

При **i = 2, h(14, 2) = (h1(14) + 2 * h2(14)) % 13 = 5**

Ячейка с номером **5** тоже занята, **i++**



Двойное хеширование: пример

- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**

h(17, 0) = (h1(17) + 0 * h2(17)) % 13 = 4

Но ячейка с номером **4** уже занята, **i++**

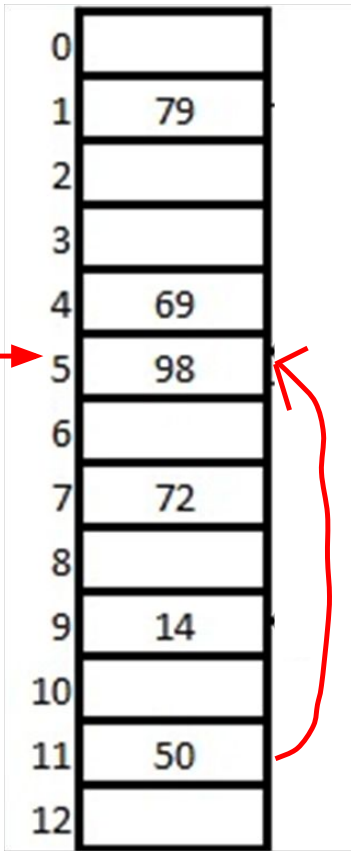
При **i = 1, h(17, 1) = 11**

Ячейка с номером **11** тоже занята, **i++**

При **i = 2, h(14, 2) = (h1(14) + 2 * h2(14)) % 13 = 5**

Ячейка с номером **5** тоже занята, **i++**

Однако шаг не меняется в пределах одного и того же ключа



Двойное хеширование: пример

- **h1(k, i) = (h1(k) + i * h2(k)) % 13**
- **h1(k) = k % 13**
- **h2(k) = k % 11 + 1**

Пусть мы хотим вставить ключ **17**. Изначально **i = 0**

h(17, 0) = (h1(17) + 0 * h2(17)) % 13 = 4

Но ячейка с номером **4** уже занята, **i++**

При **i = 1, h(17, 1) = 11**

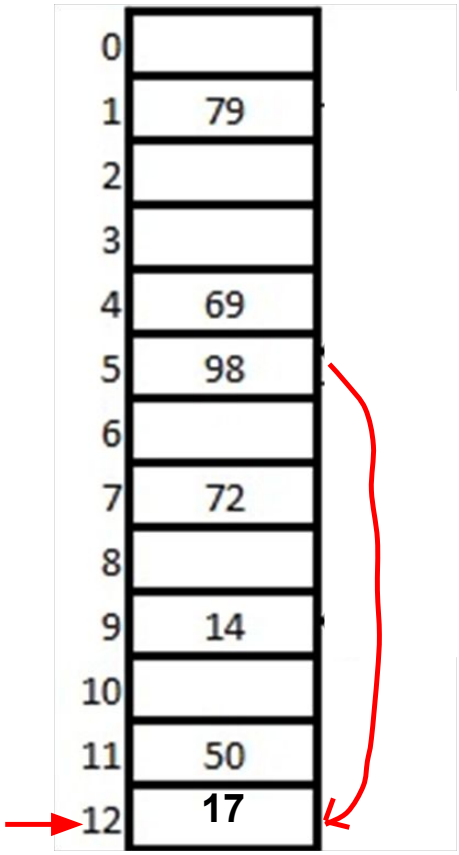
Ячейка с номером **11** тоже занята, **i++**

При **i = 2, h(14, 2) = (h1(14) + 2 * h2(14)) % 13 = 5**

Ячейка с номером **5** тоже занята, **i++**

При **i = 3, h(14, 2) = (h1(14) + 3 * h2(14)) % 13 = 12**

Ячейка с номером **12** свободна -> вставляем в нее



ПОИСК

Удаление при открытой адресации

Удаление из таблицы

Восстановление свойств таблицы

- При удалении элемента сдвигаем все последующие на **q** позиций назад
- Элемент с другим хешем должен оставаться на своем месте
- В цепочке не должно оставаться “дырок”

Удаление из таблицы

Использование флага “удалено” = true

- Помечаем удаленные элементы как **deleted**
- **Insert** будет рассматривать такие ячейки как **пустые** и может вставлять на их место
- **Search** будет рассматривать такую ячейку как **занятую** и продолжать поиск дальше

Удаление из таблицы

Использование флага “удалено”

Помечаем удаленные элементы как **deleted**

- **Insert** будет рассматривать такие ячейки как **пустые** и может вставлять на их место
- **Search** будет рассматривать такую ячейку как **занятую** и продолжать поиск дальше

Восстановление свойств таблицы

- При удалении элемента сдвигаем все последующие на **q** позиций назад
- Элемент с другим хешем должен оставаться на своем месте
- В цепочке не должно оставаться “дырок”

вставка (флаг удаления)

Двойное хеширование с удалением

Вставка

```
function add(Item item):
      x = h1(item.key)
      y = h2(item.key)
      for (i = 0..m)
         if table[x] == null or deleted[x]
            table[x] = item
            deleted[x] = false
            return
         x = (x + y) mod m
      table.resize()// ошибка, требуется увеличить размер таблицы
```

Двойное хеширование с удалением

Хотим вставить элемент **4**,
hash(4) = 4

Вставка

```
function add(Item item):
      x = h1(item.key)
      y = h2(item.key)
      for (i = 0..m)
         if table[x] == null or deleted[x]
            table[x] = item
            deleted[x] = false
            return
         x = (x + y) mod m
      table.resize()// ошибка, требуется увеличить размер таблицы
```

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	X	1	2	X	12	5	7	21		34

Двойное хеширование с удалением

Вставка

```
function add(Item item):
      x = h1(item.key)
      y = h2(item.key)
      for (i = 0..m)
         if table[x] == null or deleted[x]
            table[x] = item
            deleted[x] = false
            return
         x = (x + y) mod m
      table.resize()// ошибка, требуется увеличить размер таблицы
```

Вставка **4**, **hash(4) = 4**
и
deleted[hash(4)] == true,

поэтому ячейка **[4]**
просто перезапишется

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	**X**	1	2	**X**	12	5	7	21		34

Двойное хеширование с удалением

Вставка

```
function add(Item item):
     x = h1(item.key)
     y = h2(item.key)
     for (i = 0..m)
        if table[x] == null or deleted[x]
           table[x] = item
           deleted[x] = false
           return
        x = (x + y) mod m
     table.resize()// ошибка, требуется увеличить размер таблицы
```

Хотим вставить элемент **4**, **deleted[hash(4)] == true**, поэтому ячейка просто перезапишется

Здесь **hash** - это функция, возвращающая место, куда должна встать **4**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	**X**	1	2	**4**	12	5	7	21		34

поиск (флаг удаления)

Двойное хеширование с удалением

Поиск

```
Item search(Item key):
     x = h1(key)
     y = h2(key)
     for (i = 0..m)
        if table[x] != null
           if table[x].key == key and !deleted[x]
              return table[x]
        else
           return null
        x = (x + y) mod m
     return null
```

Двойное хеширование с удалением

Поиск

```
Item search(Item key):
     x = h1(key)
     y = h2(key)
     for (i = 0..m)
        if table[x] != null
           if table[x].key == key and !deleted[x]
              return table[x]
        else
           return null
        x = (x + y) mod m
     return null
```

Поиск происходит до первой **пустой** ячейки, поэтому при

- **h1(x) = x % 11**
- **h2(x) = x % 7**
- **h(x) = (h1(x) + i * h2(x)) % 11**

и поиске элемента **11** **?**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	**X**	1	2	**X**	12	5	7	11		34

Двойное хеширование с удалением

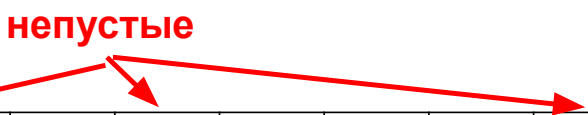
Поиск

```
Item search(Item key):
      x = h1(key)
      y = h2(key)
      for (i = 0..m)
         if table[x] != null
            if table[x].key == key and !deleted[x]
               return table[x]
         else
            return null
         x = (x + y) mod m
      return null
```

Поиск происходит до первой **пустой** ячейки, поэтому при

- **h1(x) = x % 11**
- **h2(x) = x % 7**
- **h(x) = (h1(x) + i * h2(x)) % 11**

и поиске элемента **11** мы посмотрим на ячейки с индексами **0, 4, 8** и найдем искомый элемент



h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	X	1	2	X	12	5	7	11		34

удаление (флаг удаления)

Двойное хеширование с удалением

Удаление

```
function remove(Item key):
      x = h1(key)
      y = h2(key)
      for (i = 0..m)
         if table[x] != null
            if table[x].key == key
               deleted[x] = true
         else
            return
         x = (x + y) mod m
```

Двойное хеширование с удалением

Удаление

```
function remove(Item key):
      x = h1(key)
      y = h2(key)
      for (i = 0..m)
         if table[x] != null
            if table[x].key == key
               deleted[x] = true
         else
            return
         x = (x + y) mod m
```

Хотим удалить **12**
Нашли **12** в таблице -> …

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	**X**	1	2	**X**	12	5	7	11		34

Двойное хеширование с удалением

Удаление

```
function remove(Item key):
      x = h1(key)
      y = h2(key)
      for (i = 0..m)
         if table[x] != null
            if table[x].key == key
               deleted[x] = true
         else
            return
         x = (x + y) mod m
```

Хотим удалить **12**
Нашли **12** в таблице **[5]** -> поставили флаг **deleted[find(12)] = true**

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	**X**	1	2	**X**	**X**	5	7	11		34

Удаление с восстановлением свойств таблицы

Удаление: Восстановление свойств таблицы

Допустим, нужно восстановить свойства таблицы после удаления **x**.

- в случае линейного и ему подобных методов **все элементы**, имеющие тот же хеш, что и **x** и идущие после **x** (идем с шагом **q** и берем все найденные элементы), сдвигаются на **q** позиций назад

Удаление: Восстановление свойств таблицы

Допустим, нужно восстановить свойства таблицы после удаления **x**.

- в случае линейного и ему подобных методов **все элементы**, имеющие тот же хеш, что и **x** и идущие после **x** (идем с шагом **q** и берем все найденные элементы), сдвигаются на **q** позиций назад
- в случае двойного хеширования восстановление свойств таблицы достигается только полным перехешированием всех оставшихся элементов

Удаление: Восстановление свойств таблицы

Допустим, нужно восстановить свойства таблицы после удаления **x**.

- в случае линейного и ему подобных методов **все элементы**, имеющие тот же хеш, что и **x** и идущие после **x** (идем с шагом **q** и берем все найденные элементы), сдвигаются на **q** позиций назад
- в случае двойного хеширования восстановление свойств таблицы достигается только полным перехешированием всех оставшихся элементов

Вывод: восстановление свойств таблицы в любом случае очень дорогостоящая операция

Перехеширование

Перехеширование

Метод цепочек:

Условие: когда в хеш-таблицу добавлено **4m / 3** элементов, где **m** - размер таблицы

Открытая адресация:

Условие: когда хеш-таблица заполнена на **m / 2**, где **m** - размер таблицы

Перехеширование

Метод цепочек:

Условие: когда в хеш-таблицу добавлено **4m / 3** элементов, где **m** - размер таблицы

Открытая адресация:

Условие: когда хеш-таблица заполнена на **m / 2**, где **m** - размер таблицы

- создать новую хеш-таблицу размера **2m**
- сменить хеш-функцию так, чтобы она выдавала значения **0…2m - 1**
- последовательно переместить в новую таблицу по новой хеш-функции все элементы из старой таблицы

Перехеширование

Почему вообще нужно делать перехеширование?

Перехеширование

Почему вообще нужно делать перехеширование?

Вспомним ситуацию, когда нам нужно было вставить 45, а мы не могли этого сделать

h(x)	0	1	2	3	4	5	6	7	8	9	10
x	121	23	1	2	11	12	5	7	21		34

Перехеширование решает эту проблему

Перехеширование

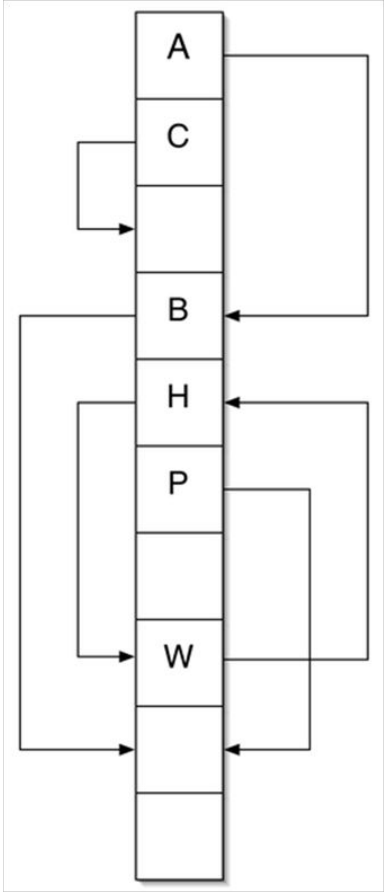
Почему ВЫГОДНО делать перехеширование?

Метод КУКУШКИ

Хеширование кукушкой

Две вариации:

- Две независимые хеш функции
- Две таблицы для двух хеш – функций



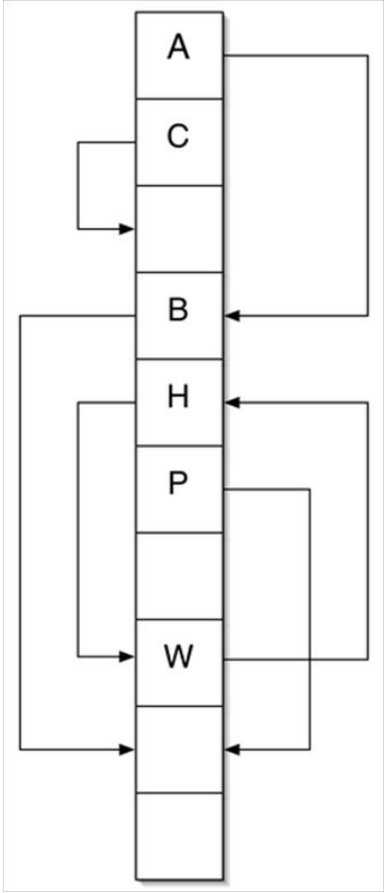
Хеширование кукушкой

Две вариации:

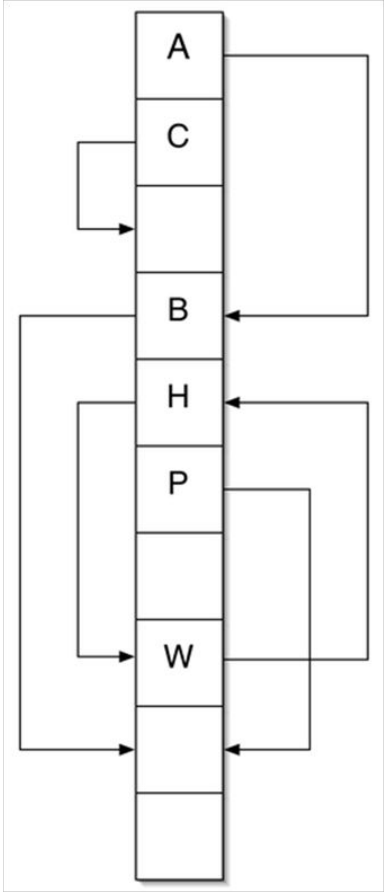
- Две независимые хеш функции
- Две таблицы для двух хеш – функций

Алгоритм **insert**:

- Если одна из ячеек с индексами **h1(x)** или **h2(x)** свободна, кладем в нее
- Иначе запоминаем элемент из занятой ячейки и помещаем туда новый
- Проверяем от сохраненного элемента вторую хеш – функцию
- Если занято, то сохраняем элемент, записываем ранее сохраненный и продолжаем дальше поиск



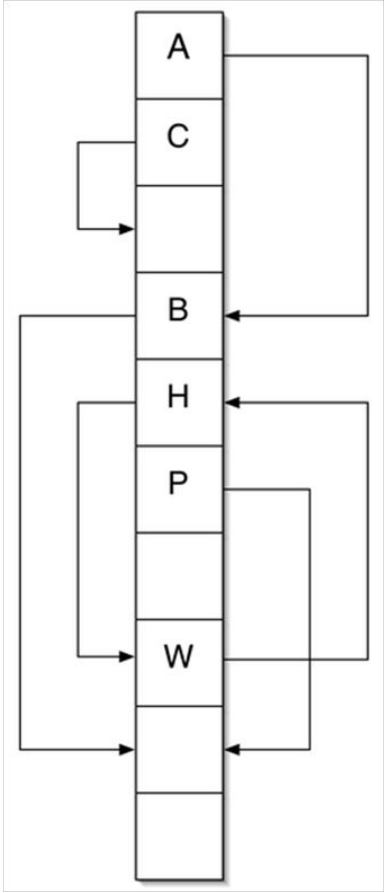
Хеширование кукушкой



Пример хеширования кукушки:

- Стрелки показывают второе возможное место элементов.
- Если нам надо будет вставить новый элемент на место **A**, то мы поместим **A** в его вторую ячейку, занятую **B**, а **B** переместим в его вторую ячейку, которая сейчас свободна.
- А вот помещение нового элемента на место **H** не получится: так как **H** — часть цикла, добавленный элемент будет вытеснен после прохода по циклу.

Хеширование кукушкой



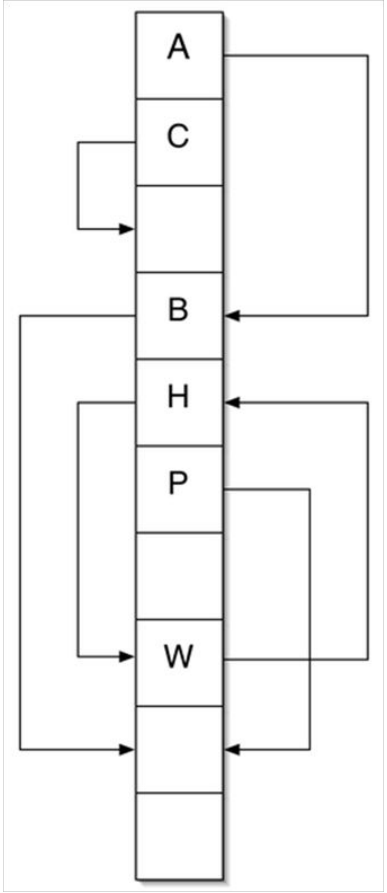
Алгоритм **remove:**

- Смотрим на ячейку с индексами **h1(x)** и **h2(x)**
- Если в одной из них есть нужный элемент, помечаем как свободную

Алгоритм **contains:**

- Смотрим на ячейку с индексами **h1(x)** и **h2(x)**
- Если в одной из них есть нужный элемент, возвращаем **true**
- Иначе возвращаем **false**

Хеширование кукушкой



При добавлении элемента может возникнуть зацикливание.

Например, оно возникнет, если добавить в таблицу 3 элемента **x, y, z** такие, что

- **h1(x) = h1(y) = h1(z)**
- **h2(x) = h2(y) = h2(z)**

Одним из способов решения проблемы зацикливания является смена хэш-функции