

# АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Лекция 5

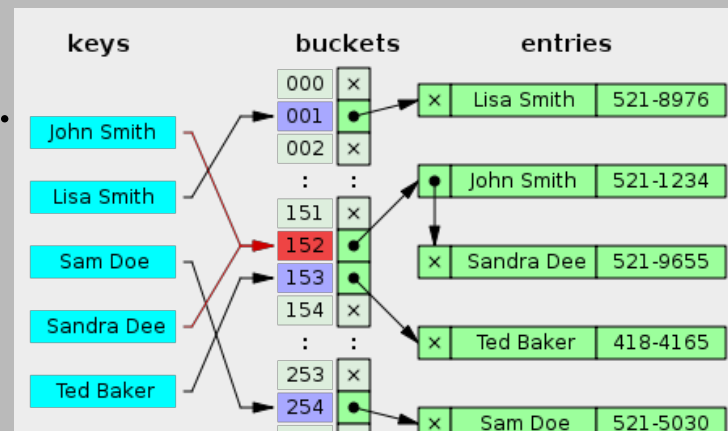


Мацкевич С.Е.

# План лекции 5 «Хеш-таблицы»



- Хеш-функции.
- Хеш-таблица. Стоимость добавления элементов.
- Разрешение коллизий методом цепочек.
- Разрешение коллизий методом открытой адресации.
- Двойное хеширование.



# Быстрый контейнер.

## Постановка задачи.

---



Задача. Хранить ключи в контейнере:

- быстро добавлять,
- быстро удалять,
- быстро проверять наличие.

Решение 1.

Упорядоченный массив:

- длительное добавление —  $O(n)$ ,
- длительное удаление —  $O(n)$ ,
- быстрый поиск —  $O(\log n)$ .

# Быстрый контейнер.

## Постановка задачи.



Задача. Хранить ключи в контейнере:

- быстро добавлять,
- быстро удалять,
- быстро проверять наличие.

Частное решение 2.

Пусть ключи – неотрицательные целые числа в диапазоне  $[0, \dots, n - 1]$ .

Будем хранить  $A$  – массив `bool`.

$A[i] = \text{true} \Leftrightarrow i$  содержится:

- мгновенное добавление –  $O(1)$ ,
- мгновенное удаление –  $O(1)$ ,
- мгновенный поиск –  $O(1)$ .

# Быстрый контейнер. Хеш-таблица.



*Хеширование* – преобразование ключей к числам.

*Хеш-таблица* – массив ключей с особой логикой, состоящей из:

1. Вычисления хеш-функции, которая преобразует ключ поиска в индекс.
2. Разрешения конфликтов, т.к. два и более различных ключа могут преобразовываться в один и тот же индекс массива.

Отношение порядка над ключами не требуется.

**Определение.** Хеш-функция — преобразование по детерминированному алгоритму входного массива данных произвольной длины (один ключ) в выходную битовую строку фиксированной длины (значение).

Результат вычисления хеш-функции называют «хешем».

**Определение.** Коллизией хеш-функции  $H$  называется два различных входных блока данных  $X$  и  $Y$  таких, что  $H(x) = H(y)$ .

Количество возможных значений хеш-функции не больше  $M$  и для любого ключа  $k$ :

$$0 \leq h(k) < M$$

**Важно!** Хорошая хеш-функция должна:

1. Быстро вычисляться.
2. Минимизировать количество коллизий.

HASH = рубить, перемешивать.

## Пример плохой хеш-функции.

$h(k) = [\text{последние [три] цифры } k] = k \% 1000.$

Такая хеш-функция порождает много коллизий, если множество ключей — цены.

Частые значения:

000, 500, 999, 998, 990, 900.





# Хеш-функции. Метод деления.



$$\underline{h(k) = k \bmod M.}$$

$M$  определяет размер диапазона значений:  
[0, .. ,  $M-1$ ].

Как выбрать  $M$ ?

- Если  $M = 2^K$ , то значение хеш-функции не зависит от старших байтов.
- Если  $M = 2^8 - 1$ , то значение хеш-функции не зависит от перестановки байт.

Обычно в качестве  $M$  выбирают простое число, далекое от степеней двойки.

# Хеш-функции.

## Метод деления многочленов.



Пусть

$$K(x) = k_0 + k_1x + \dots + k_nx^n,$$
$$P(x) = p_0 + p_1x + \dots + p_mx^m.$$

Деление многочленов с остатком:

$$K(x) = A(x) \cdot P(x) + R(x),$$

$A(x)$  – частное,  $R(x)$  – остаток,  $\deg R < \deg P = m$ .

Все коэффициенты в поле. Например, в поле  $Z_p$ ,  $p$  – простое.

Делить удобно в столбик.

# Хеш-функции.

## Метод деления многочленов.



Каждый ключ  $K$  определяется числами  $(k_0, k_1, \dots, k_{n-1})$ .

Пусть  $K(x) = k_0 + k_1x + \dots + k_{n-1}x^{n-1}$ .

Пусть задан полином  $P(x) = p_0 + p_1x + \dots + p_mx^m$ .

Многочлен  $P(x)$  называют порождающим многочленом.

Определим хеш-функцию как остаток от деления:

$$\begin{aligned} H_P(K)(x) &= K(x) \bmod P(x) = \\ &= h_0 + h_1x + \dots + h_{m-1}x^{m-1} = \\ &= (h_0, h_1, \dots, h_{m-1}). \end{aligned}$$

Все коэффициенты в поле  $Z_p$ , где  $p$  — простое.

В алгоритмах CRC используется  $p = 2$  и немного другая формула

$$H_P(K)(x) = K(x) \cdot x^m \bmod P(x)$$

# Хеш-функции. Метод умножения.



$$h(k) = [M \cdot \{k \cdot A\}],$$

где  $\{ \}$  — дробная часть,

где  $[ ]$  — целая часть,

$A$  — действительное число,  $0 < A < 1$ ,

$M$  определяет диапазон значений:  $[0, \dots, M-1]$ .

Кнут предложил в качестве  $A$  использовать число, обратное к золотому сечению:

$$A = \phi^{-1} = \left( \frac{\sqrt{5} - 1}{2} \right) = 0.6180339887 \dots$$

Такой выбор  $A$  дает хорошие результаты хеширования.

# Хеш-функции. Метод умножения.



Хеш-функцию  $h(k) = [M \cdot \{k \cdot A\}]$  вычисляют без использования операций с числами с плавающими точками.

Пусть  $M$  — степень двойки.  $M = 2^p, p \leq 32$ .

Вместо действительного числа  $A$  берут близкое к нему  $A = \frac{s}{2^{32}} = \frac{2654435769}{2^{32}}$ . То есть  $s = 2654435769$ .

$$\text{Тогда } h(k) = \left[ 2^p \cdot \left\{ k \cdot \frac{s}{2^{32}} \right\} \right] = \left[ 2^p \cdot \left\{ \frac{r_1 2^{32} + r_0}{2^{32}} \right\} \right] = \left[ 2^p \cdot \frac{r_0}{2^{32}} \right]$$

$$= \left[ \frac{r_0}{2^{32-p}} \right] = \left[ \frac{r_{01} 2^{32-p} + r_{00}}{2^{32-p}} \right] = r_{01} = \text{Старшие } p \text{ бит } r_0.$$

$$\text{Итого, } h(k) = (k \cdot s \bmod 2^{32}) \gg (32 - p).$$

# Хеш-функции строки.



Строка  $s = s_0, s_1, \dots, s_{n-1}$ .

Вариант 1.  $h_1(s) = (s_0 + s_1a + s_2a^2 + \dots + s_{n-1}a^{n-1}) \bmod M$ .

Вариант 2.  $h_2(s) = (s_0a^{n-1} + s_1a^{n-2} + \dots +$

**Теорема.** 1) Если  $a$  и  $M$  не являются взаимно простыми, то  $\{s \cdot a \bmod M, 0 \leq s < M\} \neq \{0, \dots, M - 1\}$ .

2) Если  $a$  и  $M$  взаимно просты, то  $\{s \cdot a \bmod M, 0 \leq s < M\} = \{0, \dots, M - 1\}$ .

**Доказательство.** 1) Пусть  $a$  и  $M$  не являются взаимно простыми. Тогда  $a$  и  $M$  имеют общий делитель  $d$ .

$$a = d \cdot x, M = d \cdot y.$$

Для любого  $s$  остаток от деления  $s \cdot a$  на  $M$  также делится на  $d$ :

$$s \cdot a = M \cdot k + r, r = s \cdot d \cdot x - d \cdot y \cdot k = d(sx - yk).$$

2) От противного. Пусть множество  $\{s \cdot a \bmod M, 0 \leq s < M\}$  имеет меньше  $M$  различных элементов. Тогда существуют  $i$  и  $j$ , что  $ia \equiv ja \pmod{M}$ ,  $i < j < M$ . Следовательно,  $(j - i)a = M \cdot u$ . Из этого следует, что  $j - i$  делится на  $M$ , т.к.  $a$  и  $M$  — взаимно простые. Но  $0 < j - i < M$ . Противоречие.

$h_2(s)$  вычисляется эффективнее, если использовать метод Горнера:

$$h_2(s) = \left( ((s_0 a + s_1) a + s_2) a + \dots + s_{n-2} \right) a + s_{n-1}.$$

$h_1(s)$  можно вычислять аналогично, но начиная с конца строки.

Но в с-строках известен только указатель на начало строки, а размер строки не известен.

Поэтому удобнее вычислять  $h_2(s)$ .





# Хеш-функция строки



```
// Хеш-функция строки.  
int Hash( const char* str, int m )  
{  
    int hash = 0;  
    for( ; *str != 0; ++str )  
        hash = ( hash * a + *str ) % m;  
    return hash;  
}
```

Функции вычисления контрольной суммы также являются хеш-функциями.

**CRC** — циклически избыточный код, Cyclic redundancy check. Использует метод деления многочленов, но не просто остаток от деления многочленов, а:

$$H_P(K)(x) = K(x) \cdot x^m \bmod P(x),$$

коэффициенты в поле  $Z_2$ .

Для разных версий используются многочлены разных степеней, с разными коэффициентами.

# Другие хеш-функции



## Некоторые стандарты CRC

CRC стандарт	Многочлен
CRC-1	$x + 1$
CRC-5-USB	$x^5 + x^2 + 1$
CRC-8	$x^8 + x^7 + x^6 + x^4 + x^2 + 1$
CRC-16(-IBM)	$x^{16} + x^{15} + x^2 + 1$
CRC-32-IEEE 802.3	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
CRC-64-ISO	$x^{64} + x^4 + x^3 + x + 1$

# Другие хеш-функции



**MD1, MD2, MD3, MD4, MD5, MD6** — известные алгоритмы вычисления контрольных сумм.

Message Digest. Один из самых популярных — MD5 — 128-битный алгоритм хеширования. Разработан Рональдом Л. Ривестом в 1991г. Использует битовые операции с блоками длины 128.

Важное преимущество MD — лавинный эффект. Замена одного символа приводит к полному изменению хеша:

$\text{MD5}(\text{"md5"}) = 1\text{BC}29\text{B}36\text{F}623\text{BA}82\text{AA}\text{F}6724\text{FD}3\text{B}16718.$

$\text{MD5}(\text{"md4"}) = \text{C}93\text{D}3\text{B}\text{F}7\text{A}7\text{C}4\text{A}\text{F}\text{E}94\text{B}64\text{E}30\text{C}2\text{C}\text{E}39\text{F}4\text{F}$

# Другие хеш-функции

---



Криптографические хеш-функции.

**SHA-1, SHA-2** — 160, 256/512-битные хеши.

SHA = Secure Hash Code.

Криптографические = нет способа нахождения коллизий.

**Определение.** Хеш-таблица — структура данных, хранящая ключи в таблице. Индекс ключа вычисляется с помощью хеш-функции. Операции: добавление, удаление, поиск.

Пусть хеш-таблица имеет размер  $M$ , количество элементов в хеш-таблице —  $N$ .

**Определение.** Число хранимых элементов, делённое на размер массива (число возможных значений хеш-функции), называется **коэффициентом заполнения хеш-таблицы** (load factor). Обозначим его  $\alpha = \frac{N}{M}$ .

Этот коэффициент является важным параметром, от которого зависит среднее время выполнения операций.

## Парадокс дней рождений.

При вставке в хеш-таблицу размером 365 ячеек всего лишь 23-х элементов вероятность коллизии уже превысит 50 % (если каждый элемент может равновероятно попасть в любую ячейку).

Хеш-таблицы различаются по методу разрешения коллизий.

Основные **методы** разрешения коллизий:

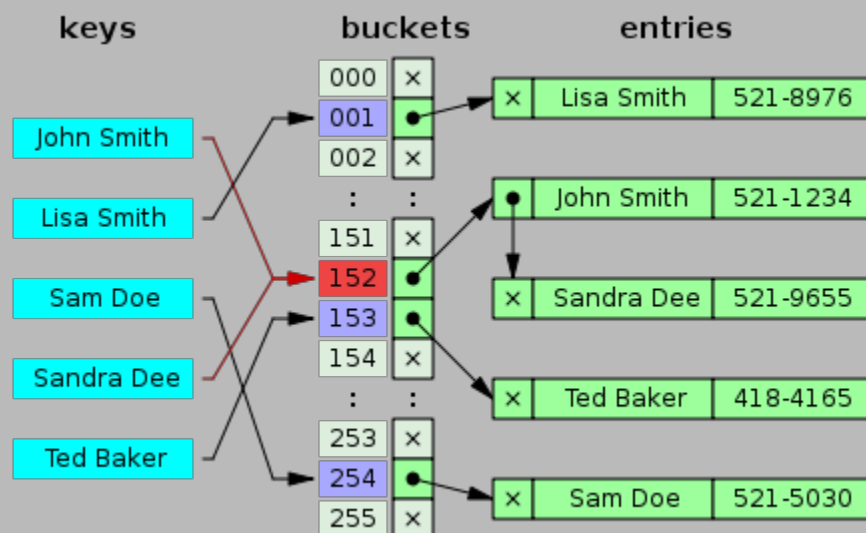
1. Метод цепочек.
2. Метод открытой адресации.

# Хеш-таблицы. Метод цепочек.



Каждая ячейка массива является указателем на связный список (цепочку).

Коллизии приводят к тому, что появляются цепочки длиной более одного элемента.





# Хеш-таблицы. Метод цепочек.



## Добавление ключа.

1. Вычисляем значение хеш-функции добавляемого ключа —  $h$ .
2. Находим  $A[h]$  — указатель на список ключей.
3. Вставляем в начало списка (в конец списка дольше).  
Если запрещено дублировать ключи, то придется просмотреть весь список.

Время работы:

В лучшем случае —  $O(1)$ .

В худшем случае

- если не требуется проверять наличие дубля, то  $O(1)$ ,
- иначе —  $O(N)$ .

# Хеш-таблицы. Метод цепочек.



## Удаление ключа.

1. Вычисляем значение хеш-функции удаляемого ключа —  $h$ .
2. Находим  $A[h]$  — указатель на список ключей.
3. Ищем в списке удаляемый ключ и удаляем его.

Время работы:

В лучшем случае —  $O(1)$ .

В худшем случае —  $O(N)$ .

# Хеш-таблицы. Метод цепочек.



## Поиск ключа.

1. Вычисляем значение хеш-функции ключа —  $h$ .
2. Находим  $A[h]$  — указатель на список ключей.
3. Ищем его в списке.

Время работы:

В лучшем случае —  $O(1)$ .

В худшем случае —  $O(N)$ .

# Хеш-таблицы. Метод цепочек.



## Среднее время работы.

**Теорема.** Среднее время работы операций поиска, вставки (с проверкой на дубликаты) и удаления в хеш-таблице, реализованной методом цепочек —  $O(1 + \alpha)$ , где  $\alpha$  — коэффициент заполнения таблицы.

**Доказательство.** Среднее время работы — математическое ожидание времени работы в зависимости от исходного ключа.

Время работы для обработки одного ключа  $T(k)$  зависит от длины цепочки и равно  $1 + N_{h(k)}$ , где  $N_i$  — длина  $i$ -ой цепочки. Предполагаем, что хеш-функция равномерна, а ключи равновероятны.

Среднее время работы

$$\begin{aligned} T_{\text{ср}}(M, N) &= M(T(k)) = \sum_{i=0}^{M-1} \frac{1}{M} (1 + N_i) = \frac{1}{M} \sum_{i=0}^{M-1} (1 + N_i) = \frac{M + N}{M} \\ &= 1 + \alpha \end{aligned}$$



# Хеш-таблицы. Метод цепочек.



```
// Хеш-функция.
template<class T>
int Hash( T& data );

// Элемент цепочки в хеш-таблице.
template<class T>
struct CHashTableNode {
    T Data;
    CHashTableNode<T>* Next;
};

// Хеш-таблица.
template<class T>
class CHashTable {
public:
    CHashTable( int initialSize );
    bool Has( T& key ) const;
    void Add( T& key );
    bool Delete( T& key );

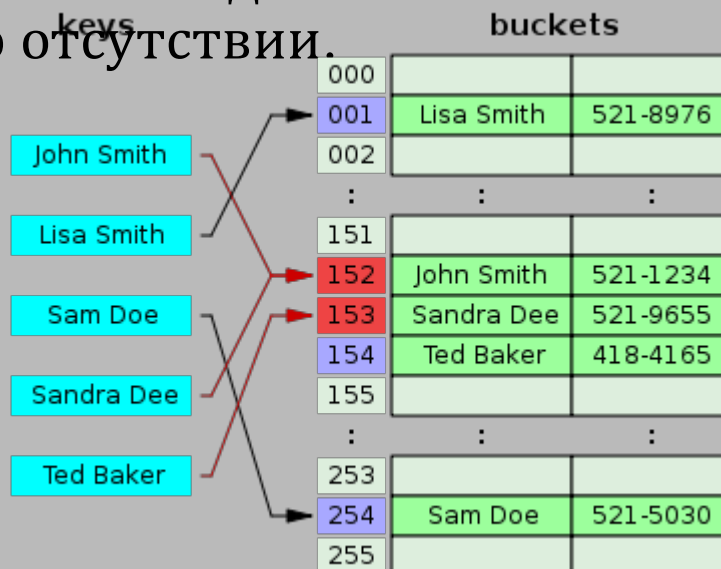
private:
    vector<CHashTableNode<T>*> table;
};
```

# Хеш-таблицы. Открытая адресация.



Все элементы хранятся непосредственно в массиве.  
Каждая запись в массиве содержит либо элемент, либо NIL.

При поиске элемента систематически проверяем ячейки до тех пор, пока не найдем искомый элемент или не убедимся в его отсутствии.



## Вставка ключа.

1. Вычисляем значение хеш-функции ключа —  $h$ .
2. Систематически проверяем ячейки, начиная от  $A[h]$ , до тех пор, пока не находим пустую ячейку.
3. Помещаем вставляемый ключ в найденную ячейку.

В п.2 поиск пустой ячейки выполняется в некоторой последовательности. Такая последовательность называется **«последовательностью проб»**.

# Хеш-таблицы. Открытая адресация.



Последовательность проб зависит от вставляемого в таблицу ключа. Для определения исследуемых ячеек расширим хеш-функцию, включив в нее номер пробы (от 0).

$$h : U \times \{0, 1, \dots, M - 1\} \rightarrow \{0, 1, \dots, M - 1\}.$$

Важно, чтобы для каждого ключа  $k$  последовательность проб

$$\langle h(k, 0), h(k, 1), \dots, h(k, M - 1) \rangle$$

представляла собой перестановку множества  $\langle 0, 1, \dots, M - 1 \rangle$ , чтобы могли быть просмотрены все ячейки таблицы.





# Хеш-таблицы. Открытая адресация.



// Вставка ключа в хеш-таблицу (без учета удаленных элементов).

```
void CHashTable::Insert( T& k )
{
    for( int i = 0; i < tableSize; ++i ) {
        int j = h( k, i );
        if( IsNil( table[j] ) ) {
            table[j] = k;
            return;
        }
    }
    throw CHashTableException( "Overflow" );
}
```

# Хеш-таблицы. Открытая адресация.



## Поиск ключа.

Исследуется та же последовательность, что и в алгоритме вставки ключа.

Если при поиске встречается пустая ячейка, поиск завершается неуспешно, поскольку искомый ключ должен был бы быть вставлен в эту ячейку в последовательности проб, и никак не позже нее.

# Хеш-таблицы. Открытая адресация.



## Удаление ключа.

Алгоритм удаления достаточен сложен.

Нельзя при удалении ключа из ячейки  $i$  просто пометить ее значением `NIL`. Иначе в последовательности проб для некоторого ключа (или некоторых) возникнет пустая ячейка, что приведет к неправильной работе алгоритма поиска.

Решение. Помечать удаляемые ячейки спец. значением «Deleted».

Нужно изменить методы поиска и вставки.

В методе вставки проверять «Deleted», вставлять на его место.

В методе поиска продолжать поиск при обнаружении «Deleted».

## Вычисление последовательности проб.

Желательно, чтобы для различных ключей  $k$  последовательность проб  $\langle h(k, 0), h(k, 1), \dots, h(k, M - 1) \rangle$  давала большое количество последовательностей-перестановок множества  $\langle 0, 1, \dots, M - 1 \rangle$ .

Обычно используются три метода построения  $h(k, i)$ :

1. Линейное пробирование.
2. Квадратичное пробирование.
3. Двойное хеширование.

## Линейное пробирование.

$$h(k, i) = (h'(k) + i) \bmod M.$$

Основная проблема — кластеризация.

Последовательность подряд идущих занятых элементов таблицы быстро увеличивается, образуя кластер.

Попадание в элемент кластера при добавлении гарантирует «одинаковую прогулку» для различных ключей и проб. Новый элемент будет добавлен в конец кластера, увеличивая его.

Если  $h(k_1, i) = h(k_2, j)$ , то  $h(k_1, i + r) = h(k_2, j + r)$  для всех  $r$ .

## Квадратичное пробирование.

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod M.$$

Требуется, чтобы последовательность проб содержала все индексы  $0, \dots, M-1$ . Требуется подбирать  $c_1$  и  $c_2$ .

При  $c_1 = c_2 = 1/2$ , то проба вычисляется рекуррентно:

$$h(k, i + 1) = h(k, i) + i + 1 \pmod{M}.$$

Возникает вторичная кластеризация. Проявляется на ключах с одинаковым хеш-значением  $h'(\cdot)$ .

Если  $h(k_1, 0) = h(k_2, 0)$ , то  $h(k_1, i) = h(k_2, i)$  для всех  $i$ .

Соответствует цепочкам в методе цепочек. Разница лишь в том, что в методе открытой адресации эти цепочки могут еще пересекаться.

## Квадратичное пробирование.

Утверждение. Если  $c_1 = c_2 = 1/2$ , а  $M = 2^p$ , то квадратичное пробирование дает перестановку  $\{0, 1, 2, 3, \dots, M-1\}$ .

Доказательство. От противного. Пусть существуют  $i$  и  $j$ ,  $0 \leq i, j \leq M - 1$ , для которых

$$\frac{i(i+1)}{2} \equiv \frac{j(j+1)}{2} \pmod{2^p}.$$

Тогда

$$\begin{aligned} i^2 + i - j^2 - j &= 2^{p+1}D, \\ (i-j)(i+j+1) &= 2^{p+1}D, \end{aligned}$$

Если  $i$  и  $j$  одинаковой четности, то  $i+j+1$  нечетна, но  $i-j$  не может делиться на  $2^{p+1}$ .

Если  $i$  и  $j$  разной четности, то  $i-j$  нечетна, но  $i+j+1$  не может делиться на  $2^{p+1}$ , т.к.  $0 < i+j+1 < 2^{p+1}$ .

Противоречие.

## Двойное хеширование.

$$h(k, i) = (h_1(k) + ih_2(k)) \bmod M.$$

Требуется, чтобы последовательность проб содержала все индексы  $0, \dots, M-1$ . Для этого все значения  $h_2(k)$  должны быть взаимно простыми с  $M$ .

- $M$  может быть степенью двойки, а  $h_2(k)$  всегда возвращать нечетные числа.
- $M$  простое, а  $h_2(k)$  меньше  $M$ .

Общее количество последовательностей проб  $= O(M^2)$ .



# Хеш-таблицы. Открытая адресация.



## Анализ хеш-таблиц с открытой адресацией.

**Теорема.** Математическое ожидание количества проб при неуспешном поиске в хеш-таблице с открытой адресацией и коэффициентом заполнения  $\alpha = \frac{n}{m} < 1$  в предположении равномерного хеширования не превышает  $\frac{1}{1-\alpha}$ .

Без доказательства.

Время работы методов поиска, добавления и удаления:

В лучшем случае —  $O(1)$ .

В худшем случае —  $O(N)$ .

В среднем —  $O\left(\frac{1}{1-\alpha}\right)$ .

# Хеш-таблицы. Открытая адресация.



## Плюсы.

- + Основное преимущество метода открытой адресации — не тратится память на хранение указателей списка.
- + Нет элементов, хранящихся вне таблицы.

## Минусы.

- Хеш-таблица может оказаться заполненной.  
Коэффициент заполнения  $\alpha$  не может быть больше 1.
- При приближении коэффициента заполнения  $\alpha$  к 1 среднее время работы поиска, добавления и удаления стремится к  $N$ .
- Сложное удаление.

# Динамическая хеш-таблица.



Изначально может быть неизвестно количество хранимых ключей. Коэффициент заполнения  $\alpha$  может приближаться к 1, а в реализации методом цепочек может быть больше 1.

Среднее время работы для метода цепочек:  
 $O(1 + \alpha)$ ,

для открытой адресации  $O(1/(1 - \alpha))$ .

Требуется динамически увеличивать размер таблицы. Аналогично динамическому массиву.

Процесс увеличения размера хеш-таблицы называется «**перехешированием**».

## Перехеширование.

1. Создать новую пустую таблицу. Размер новой таблицы  $\tilde{M}$  может быть равен  $2 \cdot M$ , где  $M$  — размер старой таблицы. Если размер таблицы должен быть простым, то следует использовать простое число близкое к  $2 \cdot M$ .
2. Проитерировать старую таблицу. Каждый ключ старой таблицы перенести в новую. Для добавления в новую таблицу надо использовать другую хеш-функцию, возвращающую значения от 0 до  $\tilde{M} - 1$ .

# Динамическая хеш-таблица.



## Когда выполнять перехеширование?

Для разных хеш-таблиц следует использовать разные стратегии.

Для хеш-таблиц, реализованных методом цепочек:

Например, когда коэффициент заполнения  $\alpha$  достиг 1.

Для хеш-таблиц, реализованных методом открытой адресации:

Например, когда  $\alpha$  достиг значения  $\frac{2}{3}$  или  $\frac{3}{4}$ .

# Хеш-таблицы. Время работы.



	Лучший случай.	В среднем. Метод цепочек.	В среднем. Метод открытой адресации.	Худший случай.
Поиск	$O(1)$	$O(1 + \alpha)$	$O\left(\frac{1}{1 - \alpha}\right)$	$O(N)$
Вставка	$O(1)$	$O(1 + \alpha)$	$O\left(\frac{1}{1 - \alpha}\right)$	$O(N)$
Удаление	$O(1)$	$O(1 + \alpha)$	$O\left(\frac{1}{1 - \alpha}\right)$	$O(N)$

