

Алгоритмы и структуры данных

ЛЕКЦИЯ 15

Хеширование

Валенда Н.А.

Кафедра Программной инженерии,
факультет КН, ХНУРЕ

Для многих приложений необходимо и достаточно реализовывать динамические множества, поддерживающие только стандартные операции :

- *вставка*
- *поиск*
- *удаление.*

Скорость поиска

- Линейный поиск - $T_{prepare} = O(n), T_{find} = O(n)$
- Бинарный поиск и деревья поиска -
 $T_{prepare} = O(n \cdot \log(n)), T_{find} = O(\log(n))$
- можно ли искать за $O(1)$

Прямая адресация

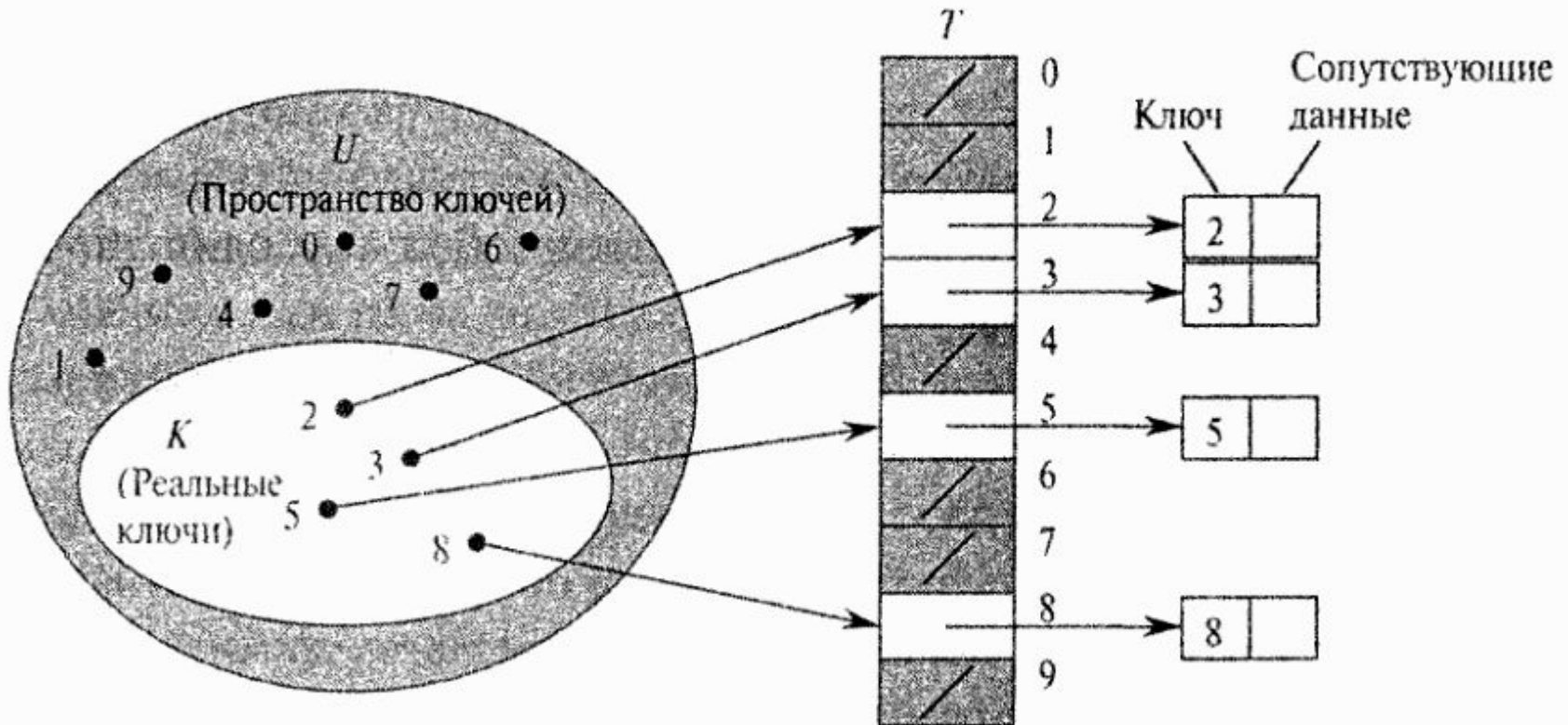
Прямая адресация применяется для небольших множеств ключей.

Требуется задать динамическое множество, каждый элемент которого имеет ключ из множества $U = \{0, 1, \dots, m - 1\}$, где m не слишком велико, никакие два элемента не имеют одинаковых ключей.

Для представления динамического множества используется массив (таблицу с прямой адресацией), $T[0..m - 1]$, каждая позиция, или ячейка, которого соответствует ключу из пространства ключей U .

Ячейка k указывает на элемент множества с ключом k . Если множество не содержит элемента с ключом k , то $T[k] = \text{NULL}$.

Прямая адресация



Операция поиска по ключу занимает время $O(1)$

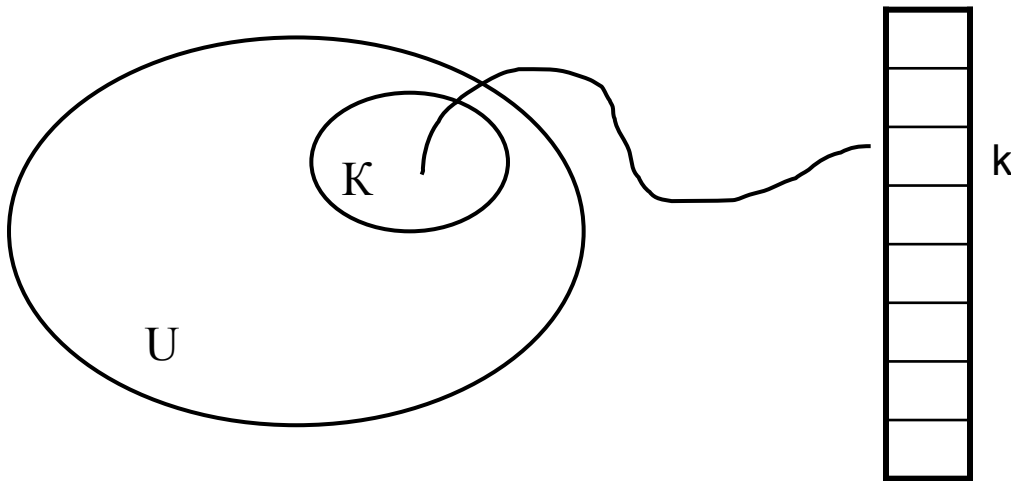
Прямая адресация

Недостатки прямой адресации :

- если пространство ключей U велико, хранение таблицы T размером $|U|$ непрактично, а то и вовсе невозможно — в зависимости от количества доступной памяти и размера пространства ключей
- множество K реально сохраненных ключей может быть мало по сравнению с пространством ключей U , а в этом случае память, выделенная для таблицы T , в основном расходуется напрасно.

Хеш-функция

Хеш-функция – такая функция h , которая определяет местоположение элементов множества U в таблице T .



Хеширование

При хешировании элемент с ключом k хранится в ячейке $h(k)$, хеш-функция h используется для вычисления ячейки для данного ключа k . Функция h отображает пространство ключей U на ячейки хеш-таблицы T $[0..m - 1]$:

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

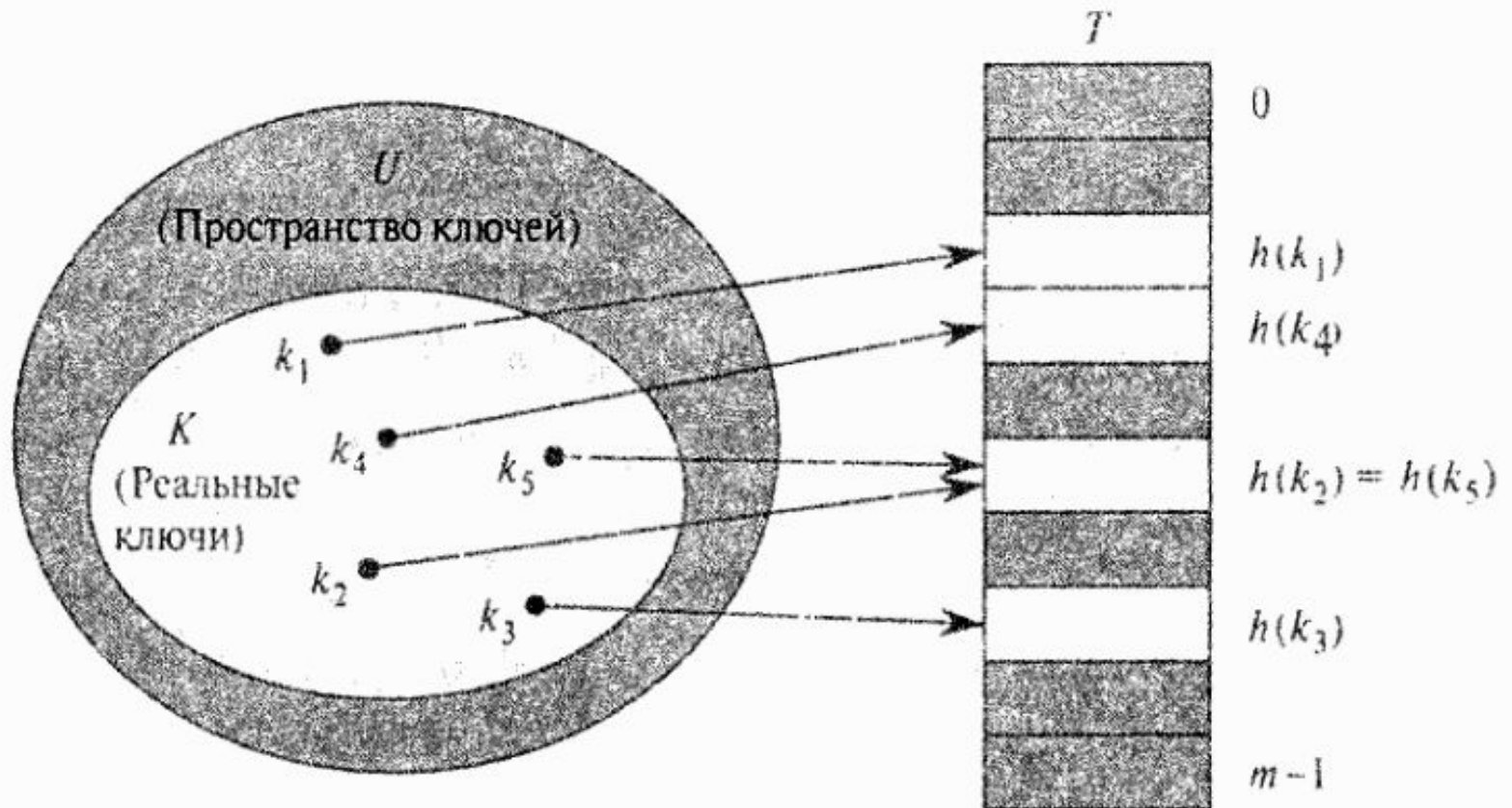
величина $h(k)$ называется хеш-значением ключа k .

Когда множество K хранящихся в словаре ключей гораздо меньше пространства возможных ключей U , хеш-таблица требует существенно меньше места, чем таблица с прямой адресацией.

Цель хеш-функции состоит в том, чтобы уменьшить рабочий диапазон индексов массива, и вместо $|U|$ значений можно обойтись всего лишь m значениями.

Требования к памяти могут быть снижены до $\Theta(|K|)$, при этом время поиска элемента в хеш-таблице остается равным $O(1)$ - это граница среднего времени поиска, в то время как в случае таблицы с прямой адресацией эта граница справедлива для наихудшего случая.

Хеширование



Коллизии

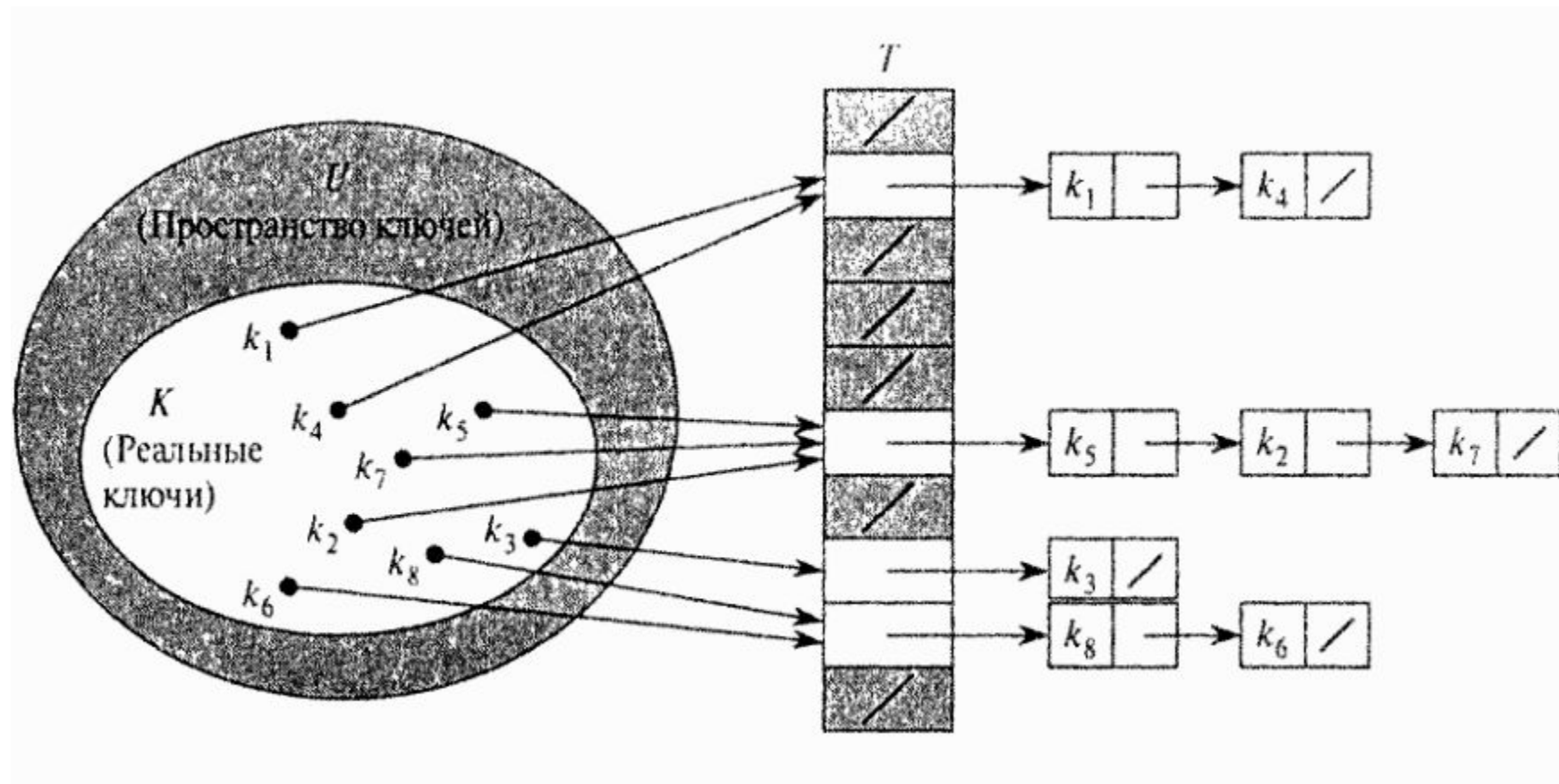
Коллизия – ситуация, когда два ключа отображаются в одну и ту же ячейку .

Например, $h(43) = h(89) = h(112) = k$

Решения коллизий:

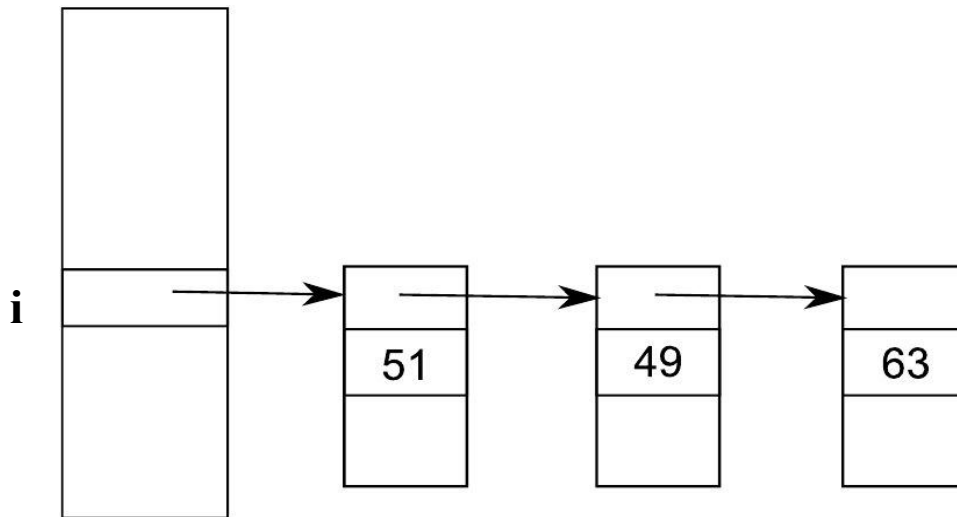
- Метод цепочек
- Открытая адресация

Коллизии



Метод цепочек

Идея: Хранить элементы множества с одинаковым значением хэш-функции в виде списка.



$$h(51) = h(49) = h(63) = i$$

Анализ метода цепочек

Наихудший случай: если хэш-функция для всех элементов множества выдает одно и то же значение. Время доступа равно $\Theta(n)$, при $|U| = n$.

Средний случай: для случая, когда значения хэш-функции равномерно распределены.

Каждый ключ с равной вероятностью может попасть в любую ячейку таблицы, вне зависимости от того куда попали другие ключи.

Анализ метода цепочек

Пусть дана таблица $T[0..m - 1]$, и в ней хранится n ключей.

Тогда, $\alpha = n/m$ - среднее количество ключей в ячейках таблицы.

Время поиска отсутствующего элемента – $\Theta(1 + \alpha)$.

Константное время на вычисления значения хэш-функции плюс время на проход списка до конца, т.к. средняя длина списка - α , то результат равен $\Theta(1) + \Theta(\alpha) = \Theta(1 + \alpha)$

Если количество ячеек таблицы пропорционально количеству элементов, хранящихся в ней, то $n = O(m)$ и, следовательно, $\alpha = n/m = O(m)/m = O(1)$, а значит поиск элемента в хэш-таблице в среднем требует времени $\Theta(1)$.

Операции

- вставка элемента в таблицу
- удаление

также требуют времени $O(1)$

Выбор хэш-функции

- Ключи должны равномерно распределяться по всем ячейкам таблицы.
- Закономерность распределения ключей хэш-функции не должна коррелировать с закономерностями данных. (Например, данные – это четные числа).

Методы:

- Метод деления
- Метод умножения

Метод деления

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

Проблема маленького делителя m

Пример №1. $m = 2$ и все ключи четные $\Rightarrow$ нечетные ячейки не заполнены.

Пример №2. $m = 2^r \Rightarrow$ хэш не зависит от битов выше r .

Метод деления: хорошая эвристика

Выбирать для m простое число не близкое к степеням 2 и 10.

Метод умножения

Пусть $m = 2^r$, ключи являются w -битными словами.

$$h(k) = (A \cdot k \bmod 2^w) \gg (w - r), \text{ где}$$

$$A \bmod 2 = 1 \cap 2^{w-1} < A < 2^w$$

Не следует выбирать A близко к 2^{w-1} и 2^w

Метод умножения: пример

$$m = 8 = 2^3, w = 7$$

							1	0	1	1	0	0	1	A
							1	1	0	1	0	1	1	k
1	0	0	1	0	1	0	0	1	1	0	0	1	1	

1001010

ignore by mod 2^w

011

$h(k)$

0011

ignore by $\gg (w-r)$

Разрешение коллизий: открытая адресация

- Не нужно хранить ссылки
- Будем последовательно проверять ячейки таблиц, пока не найдем пустую.
- $h : U \times \{0, 1, \dots, m - 1\} \rightarrow \{0, 1, \dots, m - 1\}$
- $(h(k, 0), h(k, 1), \dots, h(k, m - 1))$ – это перестановка $(0, 1, \dots, m - 1)$
- $n < m$

Открытая адресация: пример вставки

Пусть дана таблица A:

1	2	3	4	5	6	7	8	9	10
	23	45			78				

Вставим $k = 89$:

1. $h(89, 0) = 3$
2. $h(89, 1) = 2$
3. $h(89, 2) = 9$ – **Успех!**

Открытая адресация: поиск

- Поиск – также последовательное исследование
- Успех, когда нашли значение
- Неудача, когда нашли пустую клетку или прошли всю таблицу.

Стратегии исследования

- Линейная -

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

где $h'(k)$ – обычная хэш-функция

Данная стратегия легко реализуется, однако подвержена проблеме первичной кластеризации, связанной с созданием длинной последовательности занятых ячеек, что увеличивает среднее время поиска.

- Квадратичная

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

где $h'(k)$ – обычная хэш-функция

- Двойное хеширование –

$$h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod m.$$

Двойное хеширование

Этот метод даёт отличные результаты, но $h_2(k)$ должен быть взаимно простым с m .

Этого можно добиться:

1) используя в качестве m степени двойки и сделав так, чтобы $h_2(k)$ выдавала только нечётные числа
 $m = 2^r$ и $h_2(k)$ – нечетная.

2) m - простое число, значения h_2 – *целые положительные числа, меньшие m*

Для простого m можно установить

$$h_1(k) = k \bmod m$$

$$h_2(k) = 1 + (k \bmod m')$$

m' меньше m ($m' = m-1$ или $m-2$)

Открытая адресация: пример вставки

Пусть дана таблица A:

0	1	2	3	4	5	6	7	8	9	10	11	12
	79			69	98		72				50	

Двойное хеширование

$$h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod m$$

$$m=13$$

$$h_1(k)=k \bmod 13$$

$$h_2(k)=1+(k \bmod 11)$$

$$k=14$$

Куда будет встроен элемент?

Открытая адресация: пример вставки

0	1	2	3	4	5	6	7	8	9	10	11	12
	79			69	98		72		14		50	

Анализ открытой адресации

Дополнительное допущение для равномерного хеширования: каждый ключ может равновероятно получить любую из $m!$ перестановок последовательностей исследования таблицы независимо от других ключей.

Поиск отсутствующего элемента, добавление элемента

$$E[\text{количество исследований}] \leq \frac{1}{1 - \alpha},$$

$$\text{где } \alpha < 1 \Leftrightarrow n < m$$

$\alpha = n/m$, где n – число записей, m – размер таблицы.

Число проб при успешном поиске

$$\frac{1}{\alpha} \ln \frac{1}{1 - \alpha}$$

Скорость работы поиска в хэш-таблице при открытой адресации

$$a < 1 — const \Rightarrow O(1)$$

Как же себя ведет a :

- Таблица заполнена 50% $\Rightarrow$ 2 исследования
- Таблица заполнена на 90% $\Rightarrow$ 10 исследований

Список литературы

- Ахо. А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. – М. : Издательский дом «Вильямс», 2000
сс. 116-127
- Кормен Е., Лейзерсон Ч., Ривест., Штайн К. Алгоритмы: построение и анализ, 2-е издание. - М. : Издательский дом «Вильямс»,
2007. сс. 103-104, 282-315