Алгоритмы и анализ сложности

Тема «Хеш-таблицы. Хеш-функции»

Хеш-таблицы. Коллизии.

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Введение

Задача:

Реализовать динамическое множество (key-value storage) со стандартными операциями вставки элемента **(Insert),** удаления элемента **(Delete),** поиск элемента **(Search)**.

Способы реализации :

- **Массивы** – сложность всех операций *O(1)*, т.к. *произвольный* доступ к элементам.
- **Связные списки** – сложность всех операций *O(n)* , т.к. *последовательный* доступ к элементам
- **хеш-таблицы** – сложность всех операций *O(1)*
  - *вписать элементы в пространство меньшей размерности.*
- … деревья …

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Таблицы с прямой адресацией

Пусть требуется реализовать динамическое множество, где каждый элемент имеет ключ из совокупности $U = \{0, 1, \ldots, m-1\}$, при этом m не слишком велико.

Разные элементы имеют разные ключи, т.е. они уникальны.

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Таблицы с прямой адресацией

Пусть требуется реализовать динамическое множество, где каждый элемент имеет ключ из совокупности ***U = {0, 1, … ,m-1}***, при этом ***m*** не слишком велико.

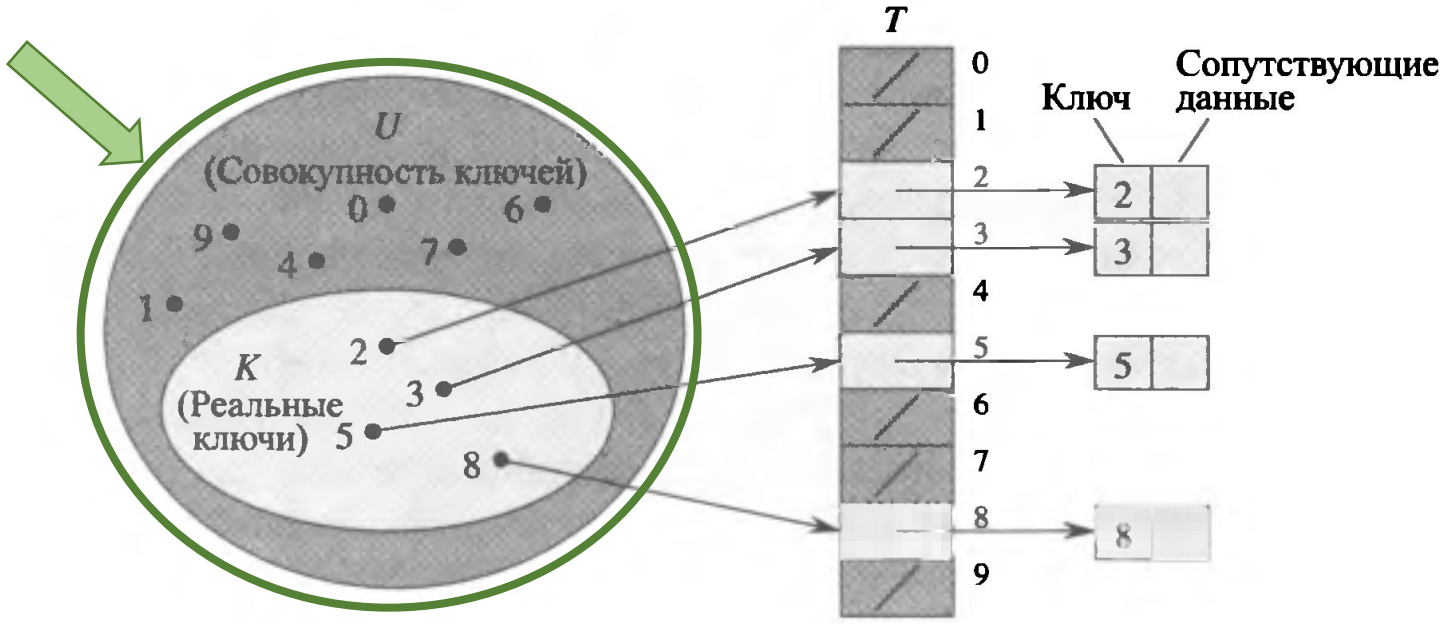
Разные элементы имеют разные ключи (идентификаторы), т.е. они уникальны.

Для реализации множества будем использовать *таблицу с прямой адресацией*:

- Таблицу организуем на массиве размера **m (m - малое)**. Обозначим его ***T[0…m-1]***.
- Никакие два элемента не имеют элементов с двумя одинаковыми ключами.
- Каждая ячейка массива соответствует ключу из совокупности, т.е ***индекс_элемента_массива = ключ_элемента_множества***.
- Если множество не содержит элемент с ключом ***k***, то ***T[k]=NULL***.

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

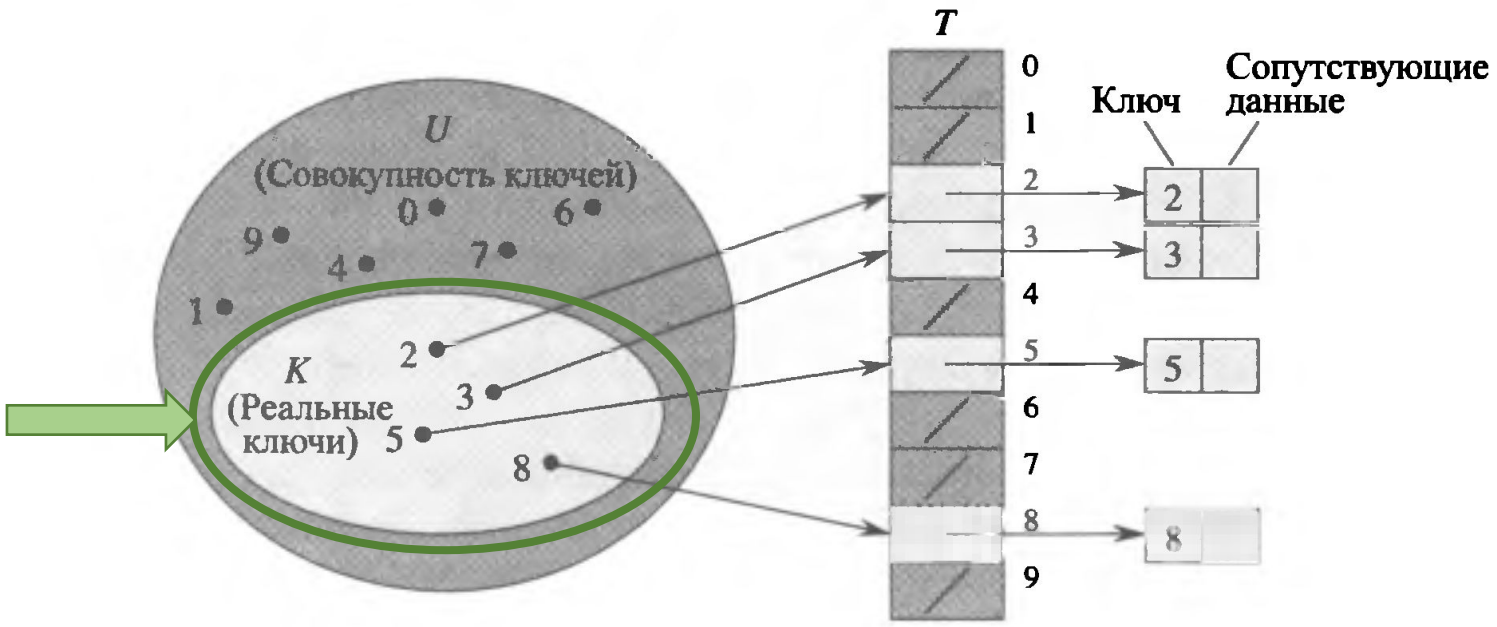
Таблицы с прямой адресацией. Иллюстрация.



U – множество всех *возможных* ключей

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

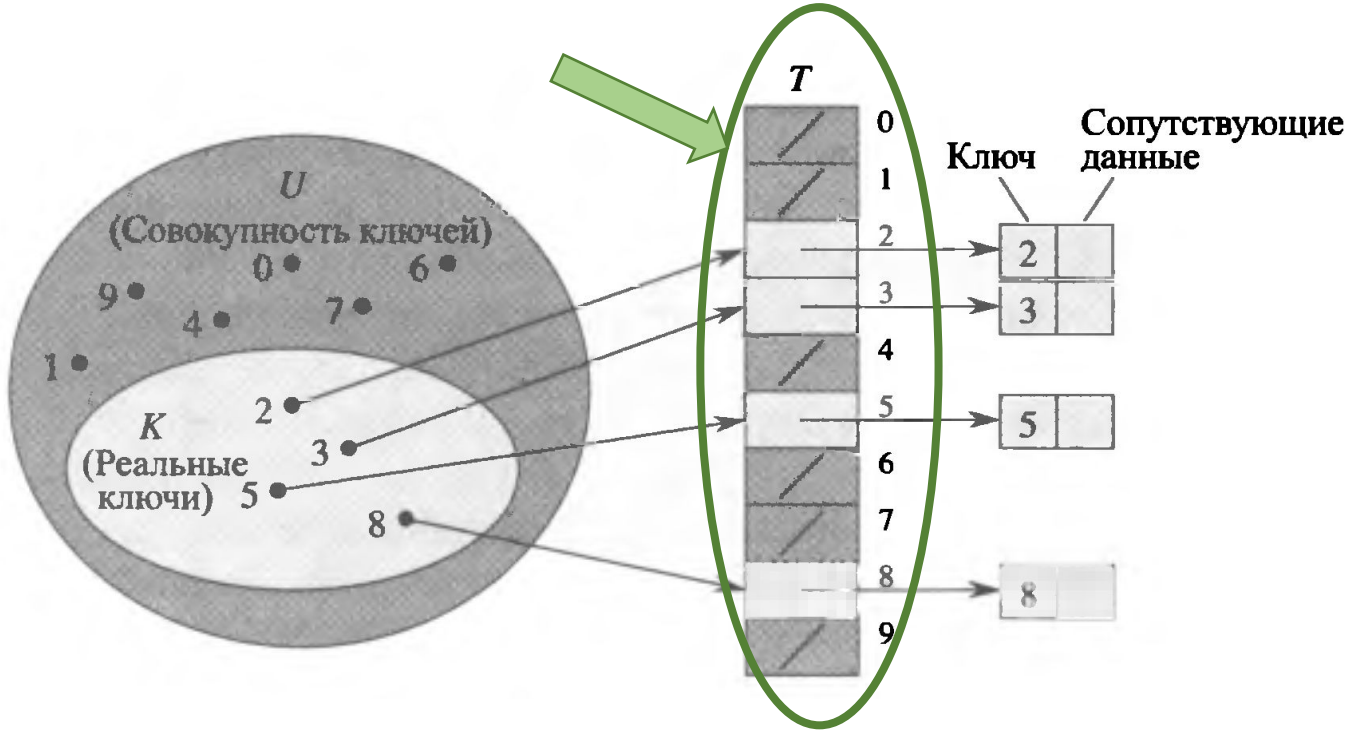
Таблицы с прямой адресацией. Иллюстрация.



K – множество всех *реальных* ключей (имеются объекты с ключами из этого множества)

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Таблицы с прямой адресацией. Иллюстрация.

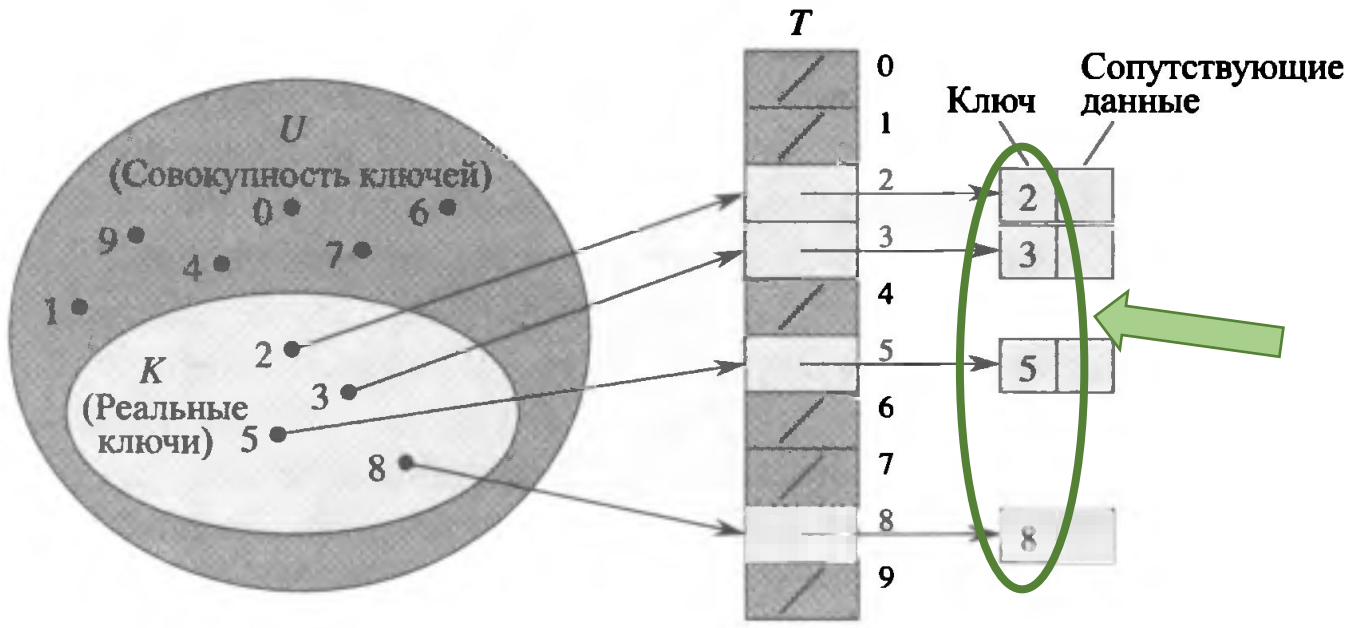


T – таблица с прямой адресацией (массив) размера ***|U|***

Таблица может быть заполнена частично.

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Таблицы с прямой адресацией. Иллюстрация.

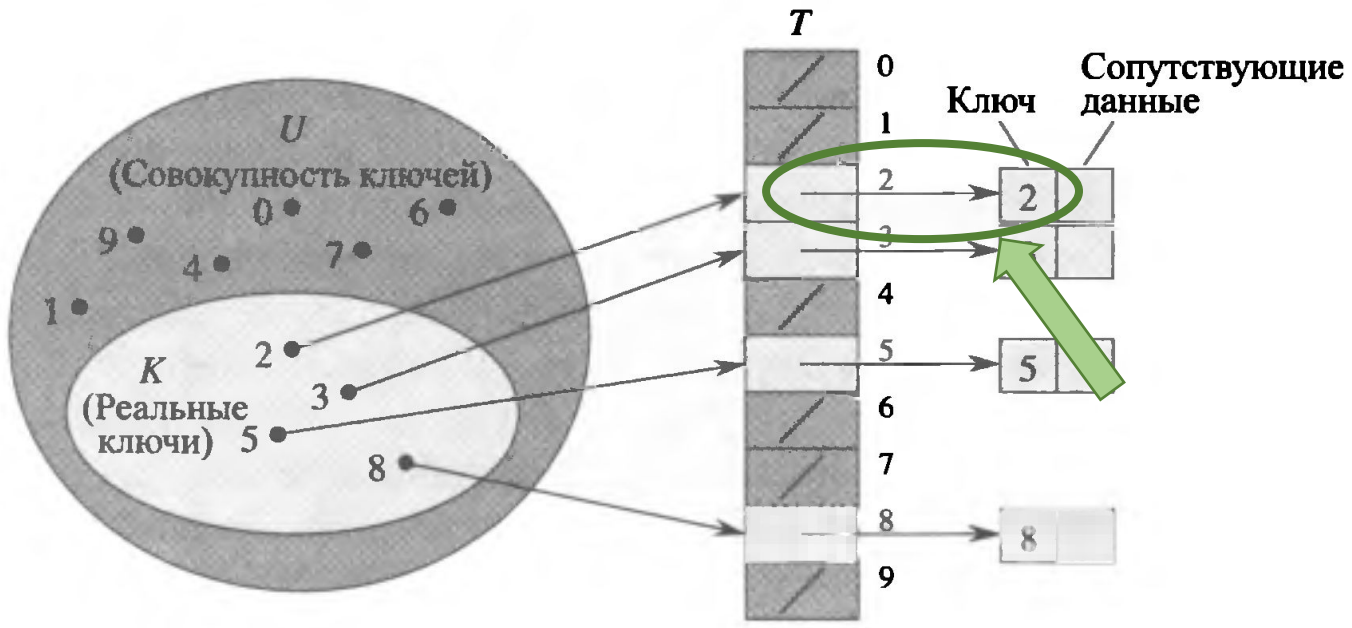


Ключи используются для того,

чтобы уникально определять каждый объект.

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Таблицы с прямой адресацией. Иллюстрация.



В ***i***-ой ячейке содержится указатель на элемент с ключом ***i***.

Каждый элемент помимо ключа содержит сопутствующую информацию.

Если не существует элемента с ключом ***j***, то ***T[j] = NULL***.

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Таблицы с прямой адресацией. Операции.

Реализация операций тривиальна.

Direct-Address-Search(T,k) // T[] – массив, k – ключ искомого элемента

```
return T[k]
```

Direct-Address-Insert(T,x) // x – вставляемый элемент, x.key – ключ

```
T[x.key] = x
```

Direct-Address-Delete(T,x) // x – удаляемый элемент, x.key – ключ

```
T[x.key] = NIL
```

Каждая из этих операций выполняется за $O(1)$.

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Таблицы с прямой адресацией. Вариации.

Если нет необходимости в дополнительных полях объектов динамического множества, сами элементы могут храниться непосредственно в таблице с прямой адресацией.

- Таблица хранит *НЕ указатели на объекты* во внешней по отношению к таблице памяти, *а сами объекты*.
- Пустая ячейка содержит *специальное значение ключа*.

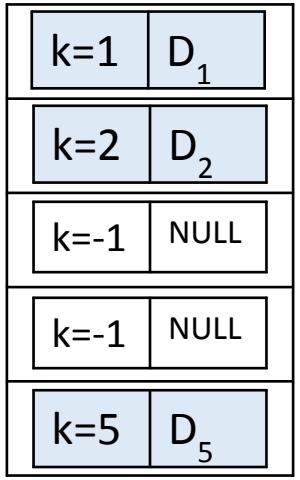


Таблица в ячейках содержит
- заполненные элементы
- пустые элементы

Элемент_массива = объект

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Таблицы с прямой адресацией. Вариации.

В объекте можно не хранить информацию о значении ключа,

т.к. индекс в таблице = значение ключа.

Но в этом случае нужен специальный механизм для того, чтобы помечать пустые ячейки.

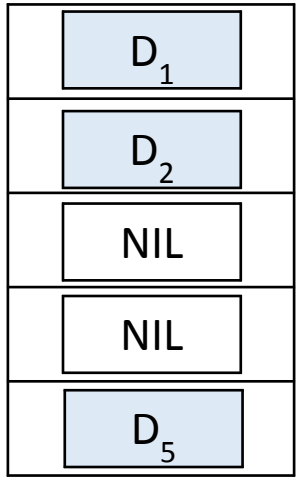


Таблица в ячейках содержит
- заполненные элементы
- пустые элементы

Элемент_массива = данные_в_объекте

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Таблицы с прямой адресацией. Недостатки.

- Если совокупность ключей велика, то хранение таблицы T размером $|U|$ непрактично или невозможно.

- Кроме того, множество K реально сохраненных ключей может быть мало по сравнению с $|U|$, то есть $|K| << |U|$.

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Хеш-таблицы

Когда используются хеш-таблицы:

- множество ***K*** *реально сохраненных ключей* мало по сравнению с ***|U|***.

Требования к памяти - $\Theta(|K|)$

Время поиска элемента – $O(1)$

Опр. **Хеш-функцией** называется такая функция ***h(k)***, которая отображает совокупность ключей ***U*** на ячейки хеш-таблицы ***T[0…m-1]***:

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

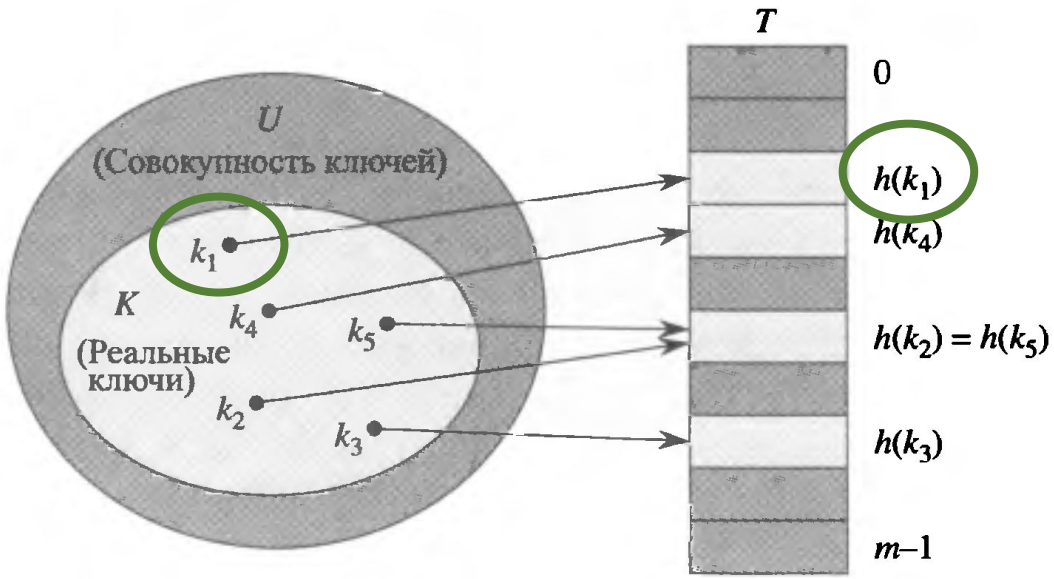
где ***m*** – размер хеш-таблицы, гораздо меньший значения ***|U|***.

Опр. ***h(k)*** называется **хеш-значением** ключа ***k***.

Цель хеш-функции: уменьшение рабочего диапазона индексов массива.

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

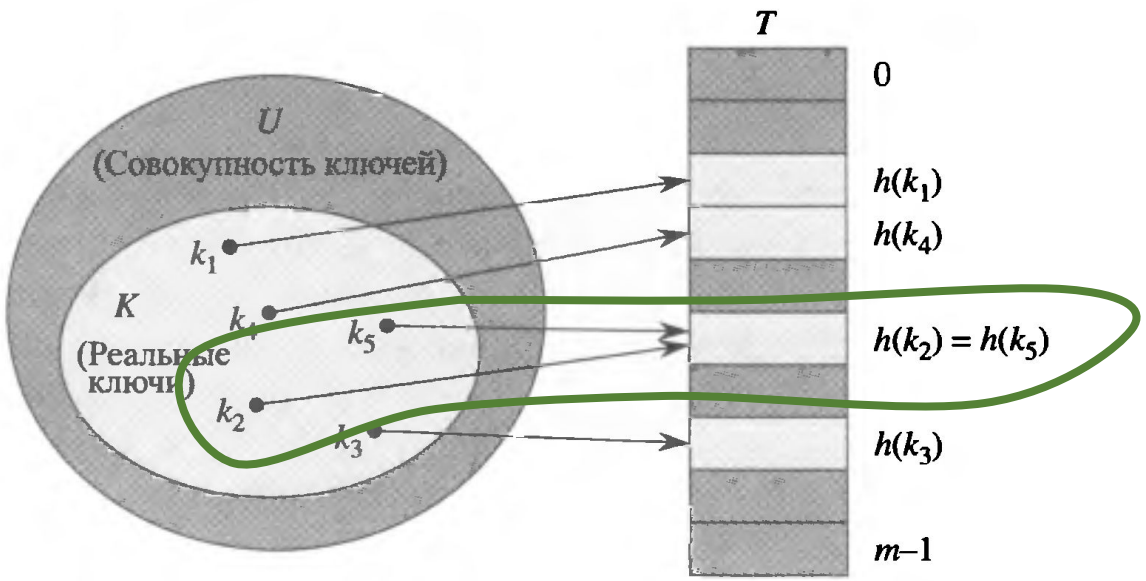
Хеш-таблицы. Иллюстрация.



Объект с ключом k_i сохраняется в таблицу по индексу $h(k_i)$.

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Хеш-таблицы. Иллюстрация.



Возможна ситуация, когда разным ключам соответствуют одинаковые хеш-значения.

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Хеш-таблицы. Коллизии.

Опр. **Коллизией** называется ситуация, когда два ключа имеют одно и то же хеш-значение, а значит, должны быть сохранены одну ячейку таблицы.

Коллизий нельзя избежать из-за неравенства $|U| > m$.

$|U|$ - количество всех возможных ключей

m – размер хеш-таблицы

Требования к хорошей хеш-функции:

- Минимизировать количество коллизий
- Функция обязательно должна быть детерминистической

Введение
Таблицы с прямой адресацией
Хеш-таблицы
Коллизии

Хеш-таблицы. Коллизии.

Методы разрешения коллизий:

- Метод разрешения с помощью цепочек
- Метод открытой адресации

Особенности использования хэш-функций

- Найти хэш-функцию, которая минимизирует коллизии
- Хорошая хэш функция должна использовать всю информацию, которая есть в ключе, чтобы максимизировать количество хэш значений
- Хэш значения должны быть равномерно распределены
- Распределение независимо от популярности данных
- Лучше когда хэш функция создает разные значения для близких значений ключей, это уменьшит коллизии
- Хэш-функция должна быть быстрой

Метод разрешения с помощью цепочек

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Разрешение коллизий с помощью цепочек

Суть метода:

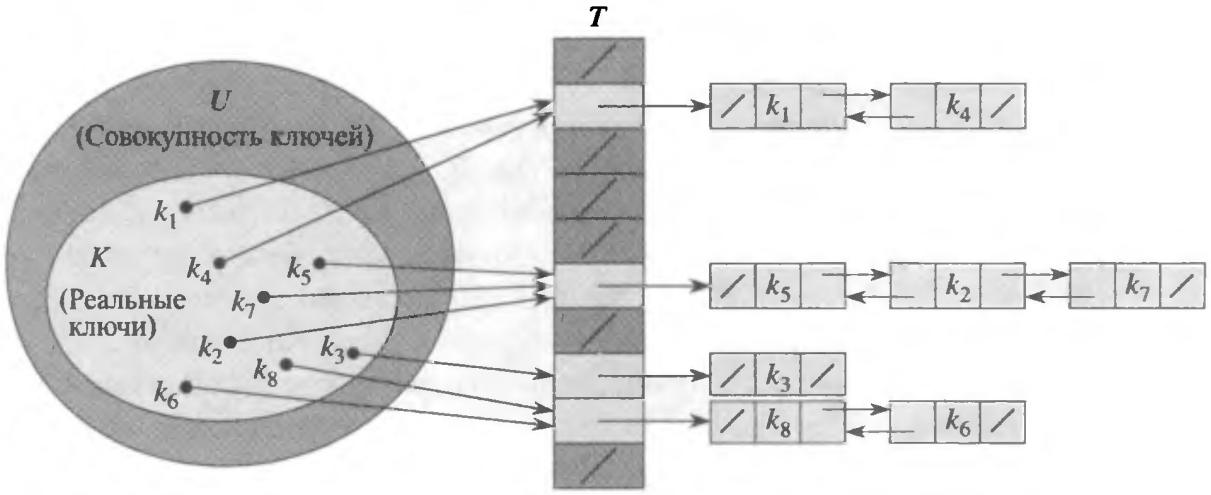
все элементы, хешированные в одну и ту же ячейку, помещаются в *связный список*.

Ячейка ***j*** содержит:

- указатель на заголовок списка всех элементов, хеш-значение ключа которых равно ***j***;
- ячейка содержит значение ***NULL***, если таких элементов нет.

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Разрешение коллизий с помощью цепочек. Иллюстрация



Каждая ячейка ***T[j]*** хеш-таблицы содержит указатель на связный список всех ключей с хеш-значением ***j***.

Например, $h(k_1)=h(k_4)$ и $h(k_5)=h(k_7)=h(k_2)$.

Список может быть одно- и двусвязным.

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Разрешение коллизий с помощью цепочек. Операции

Реализация операций работы с таблицей:

CHAINED-HASH-INSERT(T, x)

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

Элемент вставляется в начало списка, а не в конец. **Время выполнения** – константное.

CHAINED-HASH-SEARCH(T, k)

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

CHAINED-HASH-DELETE(T, x)

1 Удаление x из списка $T[h(x.key)]$

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Разрешение коллизий с помощью цепочек. Операции

Реализация операций работы с таблицей:

CHAINED-HASH-INSERT(T, x)

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

CHAINED-HASH-SEARCH(T, k)

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

CHAINED-HASH-DELETE(T, x)

1 Удаление x из списка $T[h(x.key)]$

1. Определяется нужный список.
2. Проход по списку

Время выполнения – зависит от длины списка.

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Разрешение коллизий с помощью цепочек. Операции

Реализация операций работы с таблицей:

CHAINED-HASH-INSERT(T, x)

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

CHAINED-HASH-SEARCH(T, k)

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

CHAINED-HASH-DELETE(T, x)

1 Удаление x из списка $T[h(x.key)]$

1. Определяется нужный список.
2. Проход по списку
3. Удаляется элемент списка. Переопределяются связи.

Время выполнения – зависит от длины списка.

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Разрешение коллизий с помощью цепочек

Пусть ***n*** элементов хранятся в хеш-таблице ***T*** с ***m*** ячейками.

Будем вычислять ***коэффициент заполнения α*** таблицы ***T*** как ***n/m***.

Коэффициент ***α*** *>0* и может быть:

- <1, когда n < m,
- =1, когда n = m,
- >1, когда n > m.

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Разрешение коллизий с помощью цепочек

В худшем случае: все *n* ключей имеют одинаковое хеш-значение и попадают в один и тот же список.

Время поиска = Θ(n) (как в связном списке) + время вычисления значения хеш-функции

Вывод: производительность хеширования в среднем случае зависит от того, насколько хорошо хеш-функция распределяет множество ключей по m ячейкам.

Основное предположение:

Пусть все элементы хешируются по ячейкам ***равномерно и независимо***.

Хеширование, удовлетворяющее основному предположению, называется ***простым равномерным хешированием.***

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Разрешение коллизий с помощью цепочек

Обозначим длины списков ***T[j]*** для ***j=0,1,…,m-1*** как n_j, так что

$$n_0 + n_1 + \ldots + n_{m-1} = n$$

А ожидаемое значение n_j равно $E[n_j] = n/m = \alpha$.

Утверждение 1:

В хеш-таблице с разрешением коллизий методом цепочек время неудачного поиска (элемент не найден) в среднем случае в предположении простого равномерного хеширования *составляет Θ (1+α).*

Утверждение 2:

В хеш-таблице с разрешением коллизий методом цепочек время успешного поиска (элемент найден) в среднем случае в предположении простого равномерного хеширования *составляет Θ (1+α).*

Доказывается математически

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Разрешение коллизий с помощью цепочек

Примечание.

Если количество элементов ***n*** пропорционально количеству ячеек ***m*** в хэш-таблице, то ***n = O(m)*** и ***α = n/m = O(m)/m = O(1).***

Операции поиска и удаления элемента выполняются за время $O(\alpha)$.

Вывод: все операции в среднем случае выполняются за константное время.

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Разрешение коллизий с помощью цепочек. Важно

Для качественной хеш-функции должно выполняться предположение простого равномерного хеширования.

Помещаемые в хеш-таблицу данные могут *быть зависимыми* (можем этого и не узнать, пока не начнем добавлять объекты в таблицу).

Следовательно, качественная хеш-функция должна минимизировать возможность попадания близких данных в одну ячейку. То есть **близкие данные должны быть хорошо «рассеяны».**

Прим. Идентификаторы pt и ptr. Хорошая хэш-функция распределит их по «неблизким» ячейкам таблицы.

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Разрешение коллизий с помощью цепочек. Строковые ключи

Метод преобразования ключей строкового типа в числовой:

Пусть есть 2 близких ключа: pt и ptr.

1. Переведем каждый символ в значение по таблице ASCII
   - pt = (112,116) и ptr = (112,116,113).
2. Каждому набору чисел сопоставим единственное число в 128-ричной СС с соответствующими разрядами
   - pt = (112*128)+116 = 14452
   - ptr = $112*128^2+116*128+113=1849969$

Итог: числовые ключи 14452 и 1849969 – не близки.

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Разрешение коллизий с помощью цепочек.

Методы вычисления хеш-значения по ключу:

- Метод деления
- Метод умножения
- Универсальное хеширование
- И т.д.

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Хеш-функции. Метод деления.

Суть метода:

Хеш-значение ключа k определяется как остаток от деления на количество ячеек m в хеш-таблице.

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

Пример.

Пусть таблица имеет размер 12, значение ключа = 100, тогда элемента с данным ключом попадет в список с индексом h(100) = 100 mod 12 = 4.

Рекомендации по выбору *m*:

1. НЕ должно быть степенью двойки (если $m=2^p$, то $h(k)$ – это младшие p бит числа k). Если только заранее не известно, что все наборы младших p битов ключей равновероятны, лучше строить хеш-функцию таким образом, чтобы ее результат зависел от всех битов ключа.
2. Рекомендуют выбирать для m простое число, далекое от степени двойки

   Пример. Пусть n=2000, желаемое значение α=3, тогда m=701.

   Число 701 выбрано как простое число, близкое к величине 2000/3 и не являющееся степенью 2

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Хеш-функции. Метод умножения.

Построение хеш-функции выполняется в 2 этапа:

1. Выбираем константу $0 < A < 1$. Умножаем ключ ***k*** на константу ***A*** и выделяем дробную часть результата умножения.
2. Полученное в п.1 число умножаем на ***m*** и к результату применяем функция округление снизу.

$$h(k) = \lfloor m\,(kA \bmod 1) \rfloor$$

Прим. Выражение *“kA mod 1”* означает получение дробной части произведения *kA*.

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Хеш-функции. Метод умножения.

Рекомендация для технической оптимизации:

- m перестает быть критичным
- обычно m выбирается равной степени 2 ($m=2^p$ для некоторого натурального p)
- константа A выбирается как дробь вида $s/2^w$, где s – целое число из диапазона $0<s<2^w$.

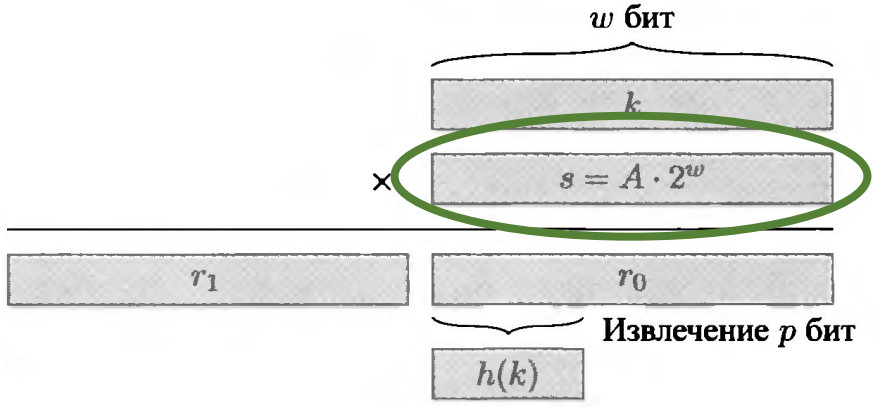
Пусть размер машинного слова составляет w бит и число k помещается в одно слово.

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Хеш-функции. Метод умножения.

Вычисление функции h(k):

1. Умножаем *k* на *w*–битовое целое число $s=A*2^w$
2. Результат умножения – *2w*-битовое число $r_1 2^w + r_0$, где r_1 – старшее слово произведения, а r_0 – младшее.
3. Старшие *p* бит числа r_0 представляют собой искомое *p*–битовое хеш-значение.

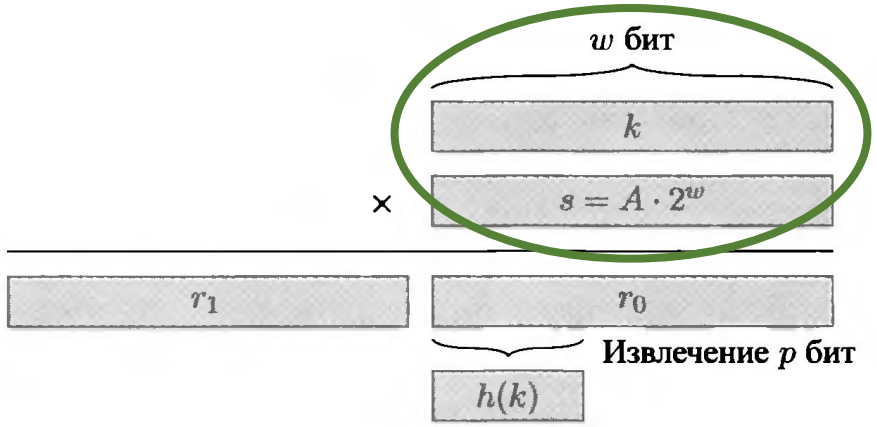


Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Хеш-функции. Метод умножения.

Вычисление функции h(k):

1. Умножаем k на w–битовое целое число $s=A*2^w$
2. Результат умножения – $2w$-битовое число $r_1 2^w+r_0$, где r_1 – старшее слово произведения, а r_0 – младшее.
3. Старшие p бит числа r_0 представляют собой искомое p–битовое хеш-значение.

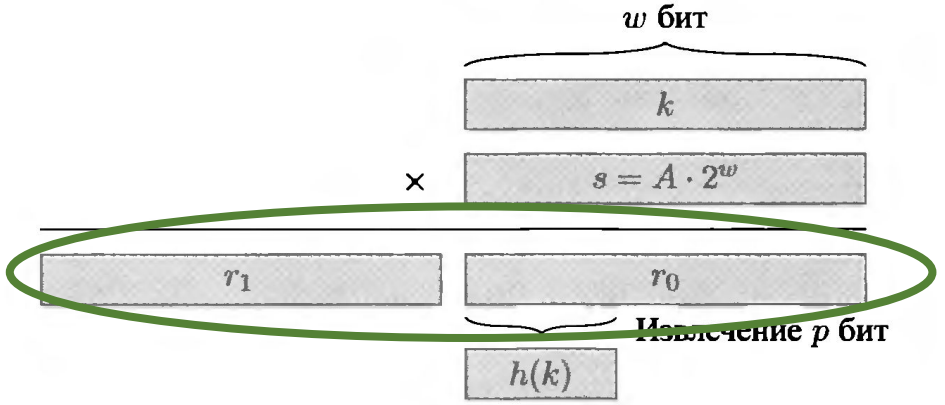


Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Хеш-функции. Метод умножения.

Вычисление функции h(k):

1. Умножаем *k* на *w*–битовое целое число $s=A*2^w$
2. Результат умножения – *2w*-битовое число $r_1 2^w + r_0$, где r_1 – старшее слово произведения, а r_0 – младшее.
3. Старшие *p* бит числа r_0 представляют собой искомое *p*–битовое хеш-значение.

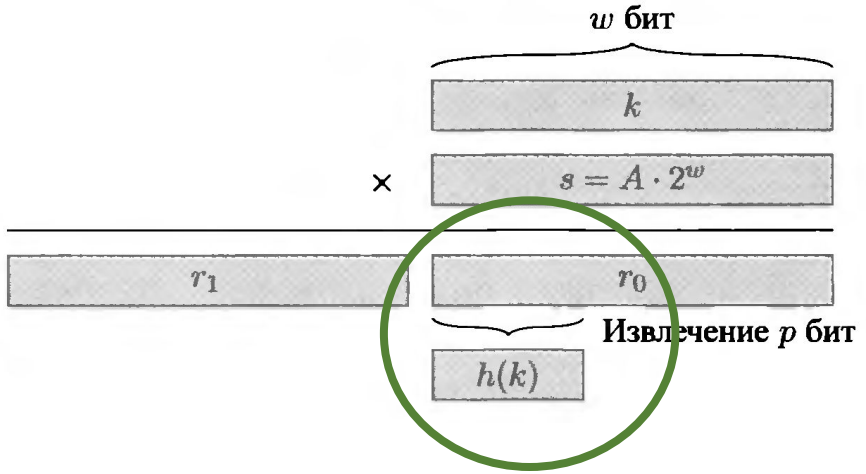


Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Хеш-функции. Метод умножения.

Вычисление функции h(k):

1. Умножаем *k* на *w*–битовое целое число $s=A*2^w$
2. Результат умножения – *2w*-битовое число $r_1 2^w + r_0$, где r_1 – старшее слово произведения, а r_0 – младшее.
3. Старшие *p* бит числа r_0 представляют собой искомое *p*–битовое хеш-значение.



Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Метод умножения. Пример.

Кнут в своей книге рекомендует в качестве константы

$$A \approx (\sqrt{5} - 1)/2 = 0.6180339887\ldots$$

Пример.

Пусть $k = 123456$, $p = 14$, $m = 2^{14} = 16384$ и $w = 32$.

Тогда представим A как дробь $2654435769/2^{32}$, где $s = 2654435769$. Такая дробь будет очень близка к предлагаемому Кнутом значению A.

Тогда $k*s = 327706022297664 = (76300*2^{32}) + 17612864$, т.е.

$r_1 = 76300$, $r_0 = 17612864$,

а старшие 14 бит числа r_0 дают $h(k) = 67$

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Хеш-функции. Универсальное хеширование.

Предположим есть недоброжелатель, знающий, какая именно хеш-функция используется и умышленно выбирающий ключи так, чтобы все ***n*** вставляемых значений попали в одну ячейку таблицы.

Тогда время поиска и удаления элемента будет $\Theta(n)$.

Вывод: хеш-функция должна выбираться случайно для каждой хеш-таблицы и не должна зависеть от набора ключей, с которыми работает.

Такой подход называется ***универсальным хешированием*** и гарантирует в среднем хорошую производительность.

Хеш-функция выбирается из некоторого, заранее определенно класса функций. В силу рандомизации выбора хеш-функции, результаты работы на одних и тех же данных всегда будут разными, что гарантирует отсутствие «всегда плохих» данных.

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Пример универсального хеширования.

Простой пример класса хеш-функций для универсального хеширования:

- Выберем простое число p достаточно большое, чтобы все возможные ключи находились в диапазоне от 0 до $p-1$ включительно. Следовательно, $p>m$.
- Обозначим через Z_p множество $\{0,1,2,\ldots,p-1\}$, а через Z_p^* – множество $\{1,2,\ldots,p-1\}$.
- Определим хеш-функцию h_{ab} для любых $a \in Z_p^*$ и $b \in Z_p$, используя линейное преобразование с последующими приведениями по модулю.

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

Пример: пусть $p=17$ и $m=6$, тогда $h_{34}(8)=5$.

Суть метода
Операции
Метод деления
Метод умножения
Универсальное хеширование

Пример универсального хеширования.

Получили семейство хеш-функций $H_{pm}=\{h_{ab}: a \in Z_p^* \text{ и } b \in Z_p\}$.

Причем размер m выходного диапазона произволен и не обязан быть простым числом.

Поскольку число a можно выбрать $p-1$ способом, а число b - p способами, то всего в семействе H_{pm} содержится $p(p-1)$ хеш-функций.

Метод открытой адресации

Суть метода
Методы исследования

Хэш-функции. Метод открытой адресации

Суть метода:

- В ячейке хеш-таблицы хранится либо элемент динамического множества, либо NULL
- Поиск элемента – проход по хэш-таблице (по определенному алгоритму) пока не найдем, либо пока не убедимся, что элемента нет
- Количество вставляемых в таблицу элементов ограничивается её размером
- Коэффициент заполнения $\alpha \leq 1$
- Прим. Отказ от указателей позволяет увеличить объем хэш-таблицы

Метод открытой адресации. Исследование таблицы

Поиск элемента в таблице – это исследование ячеек в определенной последовательности.

Если просматривать ячейки в прямой порядке 0,…,m-1, то времени требуется Θ(n).

Вывод: требуется определенный алгоритм просмотра хэш-таблицы. Можно совершить не более, чем ***m*** исследований. При этом *номер исследования* становится вторым аргументом хэш-функции.

Хэш-функция имеет вид

$$h : U \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$$

Ключ — Номер исследования — Хеш-значение ключа

Хэш-функции. Метод открытой адресации

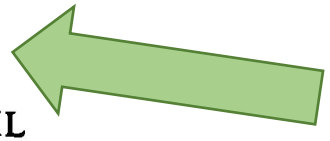
Последовательность исследований для каждого ключа ***k***

$$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$$

есть перестановка на множестве {0,1,…,m-1} .

Реализация функции вставки:

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



Вычисляем хеш-значение ключа с учетом номера исследования

Суть метода

Методы исследования

Хэш-функции. Метод открытой адресации

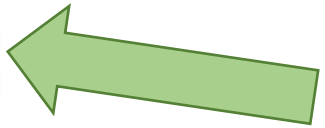
Последовательность исследований для каждого ключа ***k***

$$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$$

есть перестановка на множестве {0,1,…,m-1} .

<u>Реализация функции вставки:</u>

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



- Индекс проверяемой ячейки совпадает с хеш-значением.
- Если найденная ячейка пуста, то в нее помещаем объект с ключом *k*

Суть метода

Методы исследования

Хэш-функции. Метод открытой адресации

Последовательность исследований для каждого ключа ***k***

$$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$$

есть перестановка на множестве {0,1,…,m-1} .

<u>Реализация функции вставки:</u>

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

Если найденная ячейка не пуста, то увеличиваем счетчик и производим следующее исследование

Хэш-функции. Метод открытой адресации

Последовательность исследований для каждого ключа ***k***

$$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$$

есть перестановка на множестве {0,1,…,m-1} .

Реализация функции вставки:

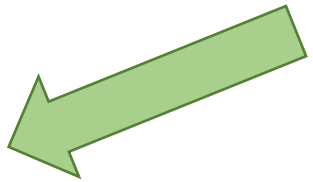
```
HASH-INSERT(T, k)
1  i = 0
2  repeat
3       j = h(k, i)
4       if T[j] == NIL
5            T[j] = k
6            return j
7       else i = i + 1
8  until i == m
9  error "переполнение хеш-таблицы"
```

Если не найдено ни одной пустой ячейки

Хэш-функции. Метод открытой адресации

Реализация функции поиска:

```
HASH-SEARCH(T, k)
1  i = 0
2  repeat
3       j = h(k, i)
4       if T[j] == k
5            return j
6       i = i + 1
7  until T[j] == NIL или i == m
8  return NIL
```



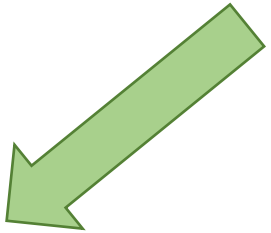
В ходе i–ой пробы в j–ой ячейке таблицы нашли объект с ключом k

Прим. Не забываем предположение о равномерном хешировании: для каждого ключа должны быть равновероятны все *m!* перестановок множества {0,1,…,m-1}.

Хэш-функции. Метод открытой адресации

Реализация функции поиска:

```
HASH-SEARCH(T, k)
1  i = 0
2  repeat
3       j = h(k, i)
4       if T[j] == k
5            return j
6       i = i + 1
7  until T[j] == NIL или i == m
8  return NIL
```



Неблагоприятные исходы:

- Дошли до пустой ячейки, но на предыдущих пробах не нашли искомый элемент
- Прошли по всей таблице

Прим. Не забываем предположение о равномерном хешировании: для каждого ключа должны быть равновероятны все *m!* перестановок множества {0,1,…,m-1}.

Метод открытой адресации. Линейное исследование.

Используется вспомогательная хэш-функция ***h`: U → {0,1,…,m-1}***

Для вычисления последовательности исследований используется хэш-функция

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

где ***i = 0,1,…,m-1***

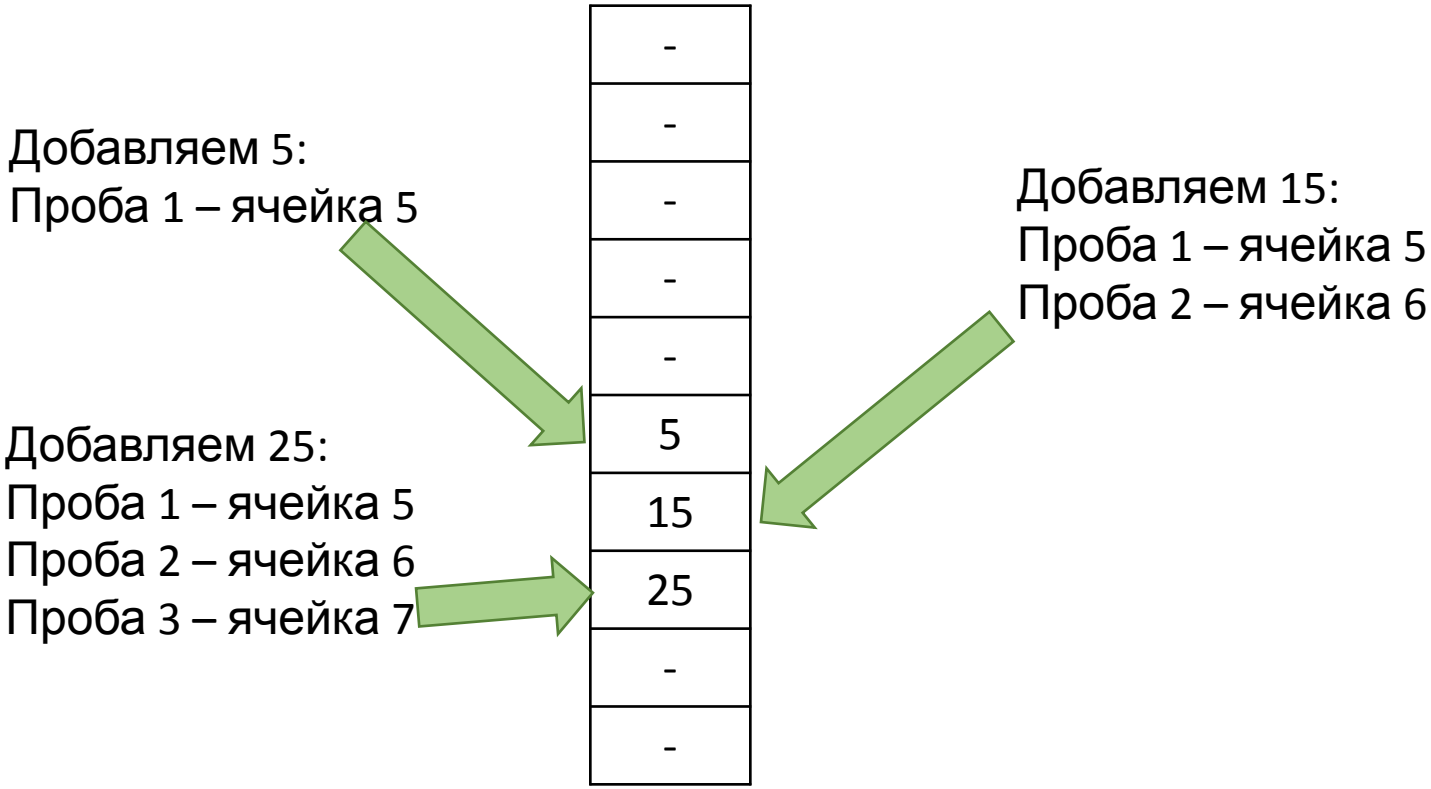
В результате использований хэш-функции исследуются ячейки

T[h`(k)], T[h`(k)+1], …, T[m-1], T[0], T[1], …, T[h`(k)-1]

Метод открытой адресации. Линейное исследование.

Проблема метода: первичная кластеризация.

Пример: добавим в хеш-таблицу размера *m=10* с использованием хеш-функции *h(k)=k mod m* элементы с ключами 5,15,25.



Метод открытой адресации. Квадратичное исследование.

Используется хэш-функция

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

где ***h`*** - вспомогательная хэш-функция, c_1 и c_2 – положительные вспомогательные константы, ***i = 0,1,…,m-1.***

Прим. Требуется тщательный выбор c_1, c_2, m.

Проблема метода: вторичная кластеризация (последовательности исследований у ключей совпадают, если совпадают начальные позиции исследований).

Метод открытой адресации. Двойное хеширование.

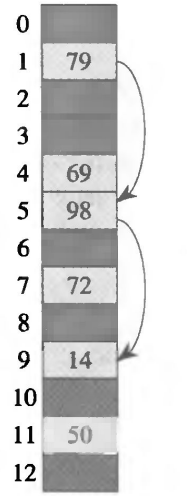
Используется хеш-функция

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

где h_1, h_2 - вспомогательные хеш-функции, $i = 0,1,\ldots,m-1$.

Стартовое значение: $h_1(k)$

Шаг поиска: $h_2(k)$



Вставка при двойном хешировании. Здесь показана хеш-таблица размером 13 ячеек, в которой используются хеш-функции $h_1(k) = k \bmod 13$ и $h_2(k) = 1 + (k \bmod 11)$. Так как $14 \equiv 1 \pmod{13}$ и $14 \equiv 3 \pmod{11}$, ключ 14 вставляется в пустую ячейку 9, после того как при исследовании ячеек 1 и 5 выясняется, что эти ячейки заняты.

Метод открытой адресации. Двойное хеширование.

Пример вспомогательных хеш-функций:

$$h_1(k) = k \bmod m\ ,$$
$$h_2(k) = 1 + (k \bmod m')\ ,$$

Где m` должно быть немного меньше m (например m-1).

Пример.

Пусть k_1=80 и k_2=123456, m=701, m`=700,

тогда $h_1(k_1)=h_1(k_2)$=80 (совпадают стартовые позиции),

$h_2(k_1)$ = 81 и $h_2(k_2)$ = 257.