Лекция по курсу «Алгоритмы и структуры данных» / «Технологии и методы программирования»

ХЕШ-таблицы

Мясников Е.В.

ХЕШ-таблицы

До настоящего времени мы изучили несколько методов поиска в упорядоченных структурах данных, таких как массивы или деревья поиска. Во всех ранее рассмотренных случаях поиск и организация соответствующих структур выполнялись на основе *сравнения непосредственно значений ключей* обрабатываемых записей.
Альтернативой прямому сравнению ключей является построение некоторой *функции, отображающей значения ключей в адреса записей*, хранящих ключи и связанную с ними информацию.
В случае использования такого метода организации данных поиск записи по ключу будет сводиться к расчету значения указанной функции и обращению к ячейке таблицы по вычисленному адресу. Таким образом, при использовании указанной структуры данных поиск будет выполняться за *константное время*.

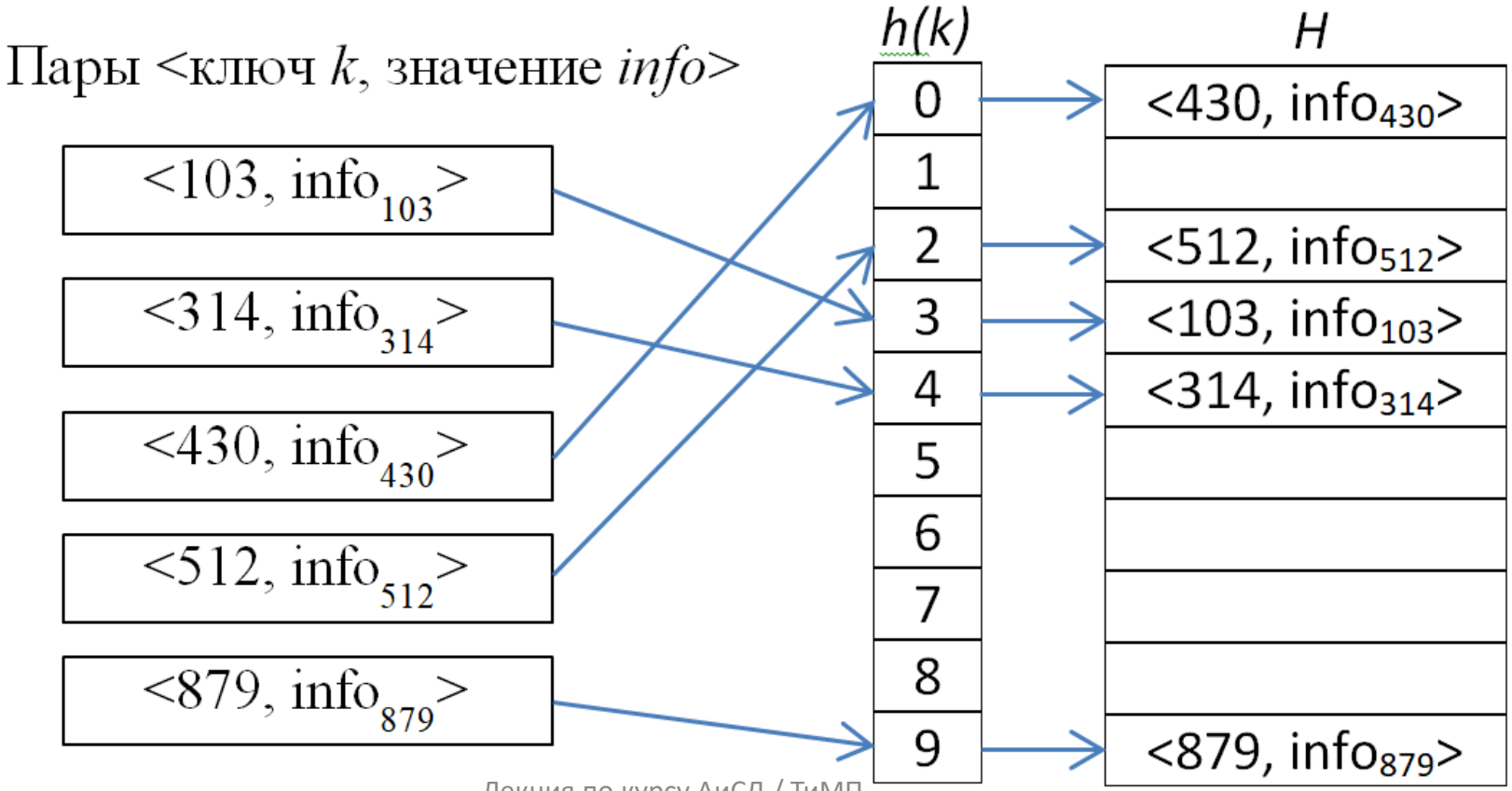
Хеш-таблицей называется структура данных, позволяющая хранить пары вида «ключ-значение», поддерживающая операции поиска, вставки и удаления элемента и использующая ***хеш-функцию*** для вычисления адреса (индекса) пары.

Бытовым аналогом хеширования в данном случае можно считать размещение слов в словаре по алфавиту. Первая буква слова является его хеш-кодом, и при поиске мы просматриваем не весь словарь, а только нужную букву.

ХЕШ-таблицы: пример построения

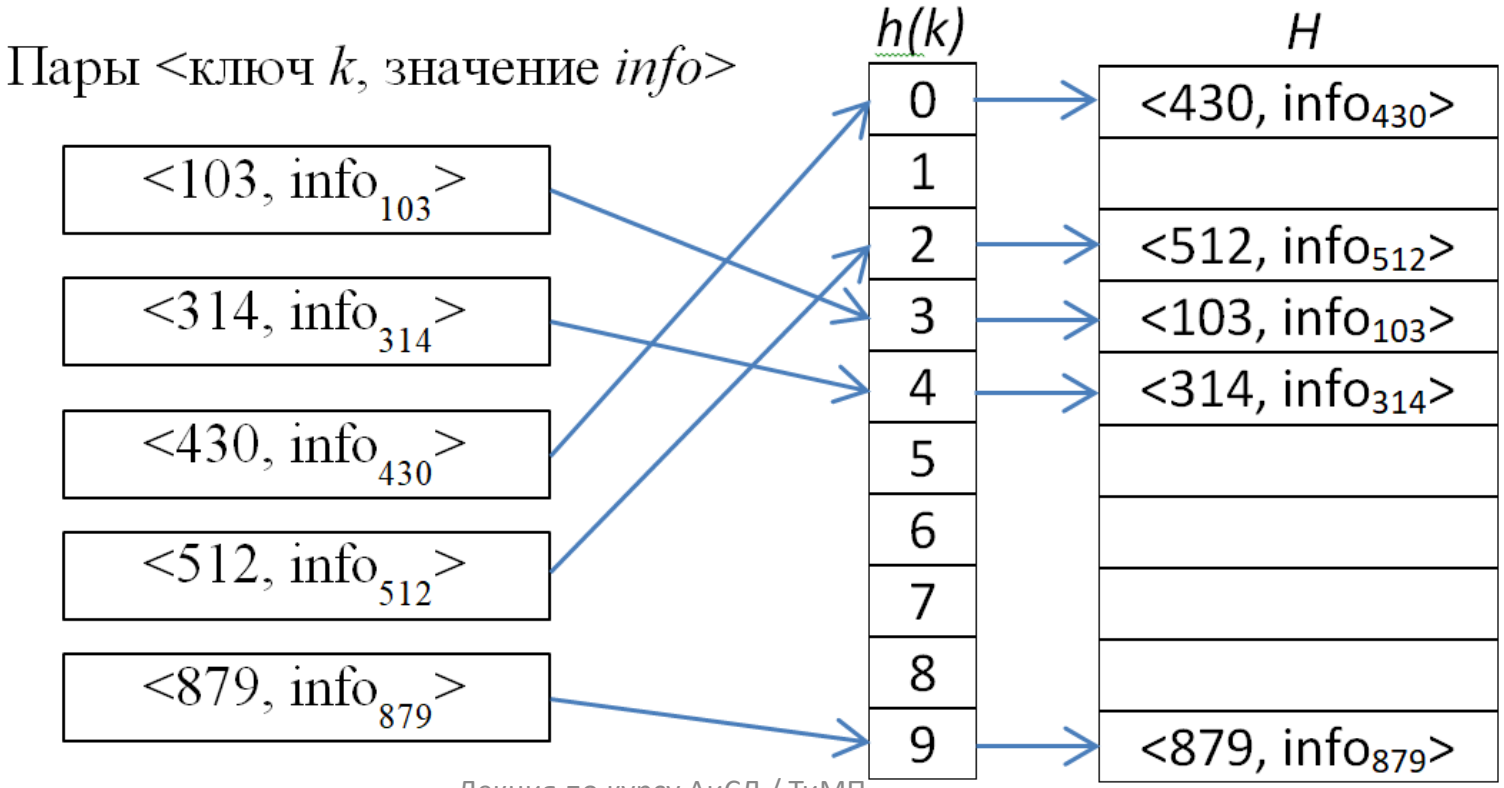
Выполнение операции (поиск, добавление или удаление) в хеш-таблице **H** начинается с вычисления значения **i** хеш-функции **h** по ключу **k** искомой записи: **i = h(k)**. Полученное значение **i** играет роль индекса в массиве **H**.

Рассмотрим простой пример хеш- таблицы, содержащей 5 пар записей с ключами **103, 314, 430, 512 и 879**, хеш-функция в которой имеет вид: **h(k) = k mod 10**, где **mod** – операция вычисления остатка от деления.



ХЕШ-таблицы: пример построения

При вставке очередного элемента, например, пары **<512, info$_{512}$>** ,берется значение ключа ***k*=512**, вычисляется значение хеш-функции
i = h(512) = 512 mod 10 = 2,
которое и используется в качестве индекса ячейки в таблице ***H***, в которую будет сохранена указанная запись. Аналогично, при поиске записи по ключу, например, для ***k=314*** вычисляется ***h(314) = 4*** и берется запись в ячейке с индексом **4**.



ХЕШ-таблицы: коллизии

К сожалению, указанный простой способ построения имеет очевидный *недостаток*: при попытке вставить *запись с таким значением хеш-функции, которое уже присутствует в таблице*, нам придется перестраивать таблицу и искать новую хеш-функцию, которая позволит разместить записи так, чтобы они не мешали друг другу, т.е. не возникало коллизий.

Коллизией называется ситуация, при которой для различных ключей значения хеш функции-совпадают.

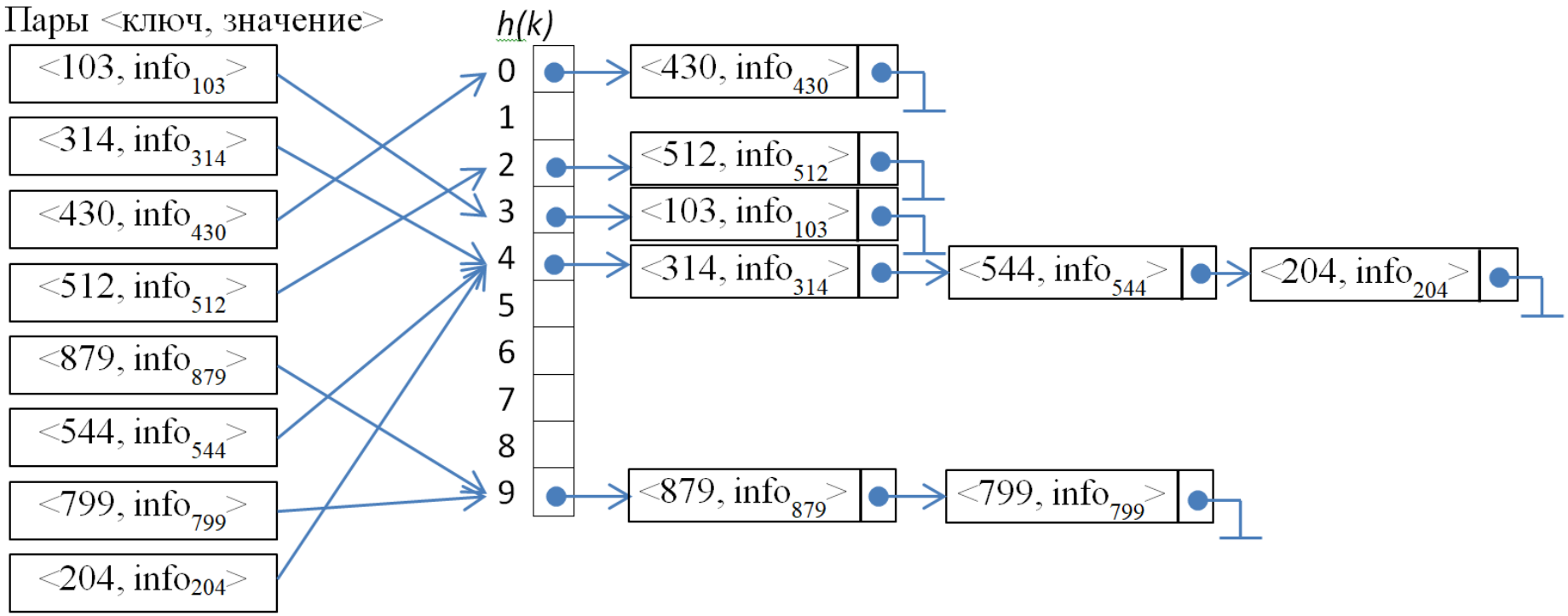
В некоторых случаях, когда заранее известны значения хранимых ключей или они меняются редко, удаётся построить ***совершенную хеш-функцию***, позволяющую распределять ключи в таблице без коллизий (хеш-таблицы с прямой адресацией).

Однако чаще всего коллизии допускаются, и используется один из двух распространенных ***способов разрешения кол***лизий:

- метод цепочек и
- метод открытой адресации.

Метод цепочек

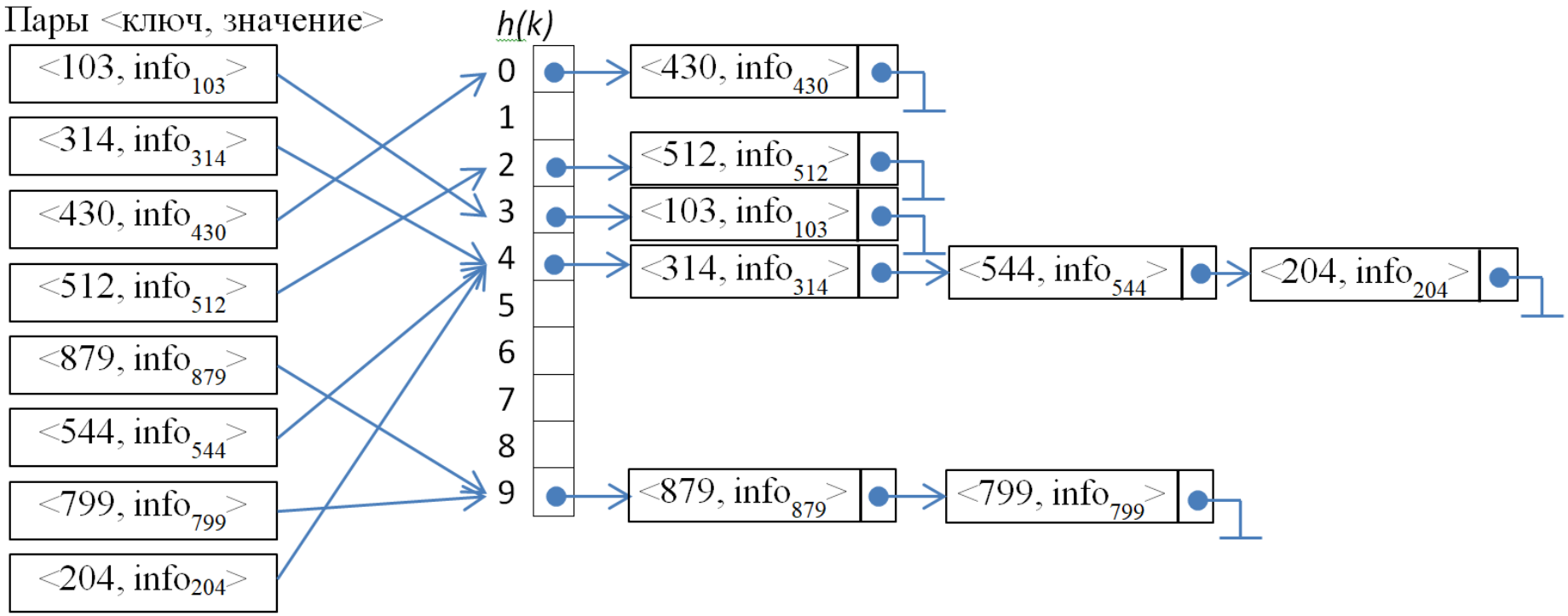
В методе цепочек каждая ячейка таблицы, определяемая значением хеш-функции, содержит не одну пару «ключ-значение», а связный список (цепочку) таких записей с одинаковым значением хеш-функции.
В таком случае, очевидно, *коллизии приводят к образованию списков длиной большей единицы*, а время поиска определяется не только временем расчета хеш-функции и доступа к ячейке таблицы, но и временем поиска в списке по ключу (прямой перебор).



Метод цепочек: добавление элемента

Операция добавления элемента заключается в *расчете значения хеш-функции* с последующей *вставкой нового элемента в начало или конец соответствующего списка*.

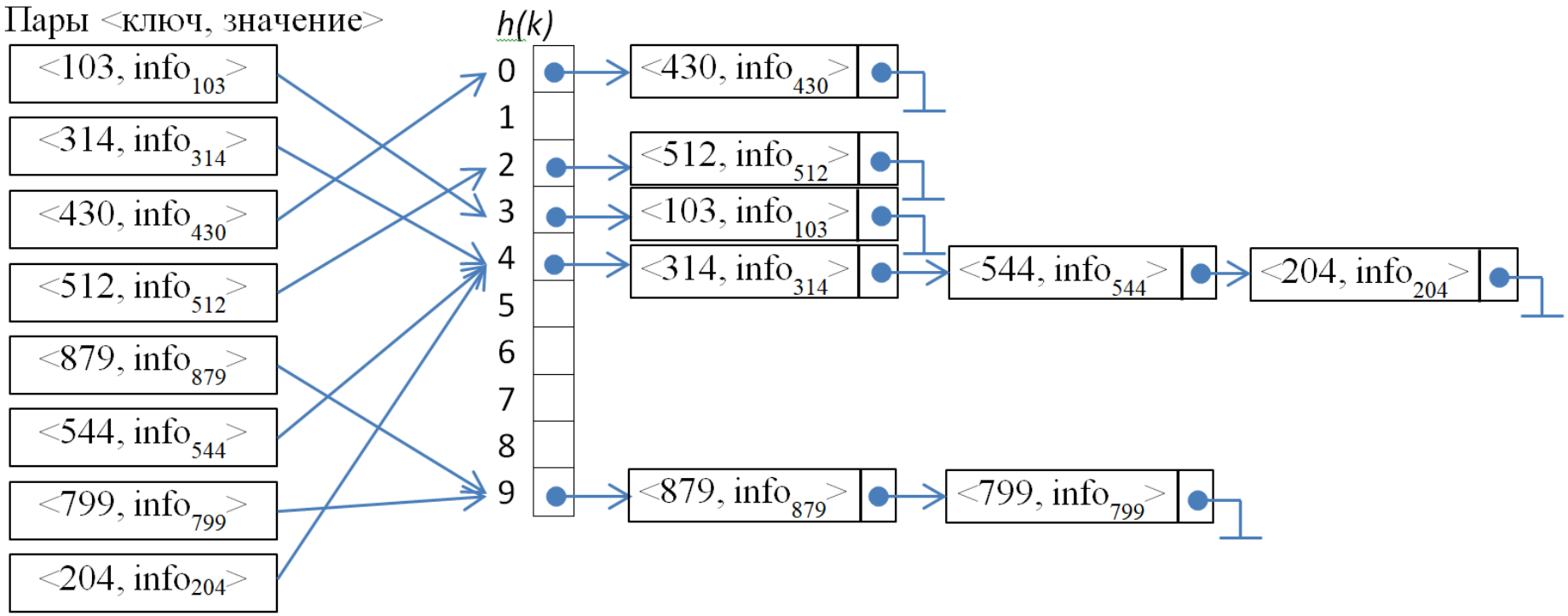
Операция *удаления* элемента - в расчете значения хеш-функции с последующим удалением элемента из списка.



Метод цепочек: добавление и удаление

Операция ***добавления элемента*** заключается в *расчете значения хеш-функции* с последующей *вставкой нового элемента в начало или конец соответствующего списка*.

Операция ***удаления элемента*** - в расчете значения хеш-функции с последующим удалением элемента из списка.



Метод цепочек:

Как правило, при использовании метода цепочек *требуется, чтобы большинство ячеек содержало списки длиной не более единицы*, реже - длиной два или более.

По мере заполнения таблицы (при достижении коэффициентом заполнения предопределенного значения), требуется перестройка хеш-таблицы: увеличение ее размера и соответствующее изменение хеш-функции.

Коэффициентом заполнения хеш-таблицы называется отношение количества хранимых в ней записей к размеру таблицы.

Метод открытой адресации

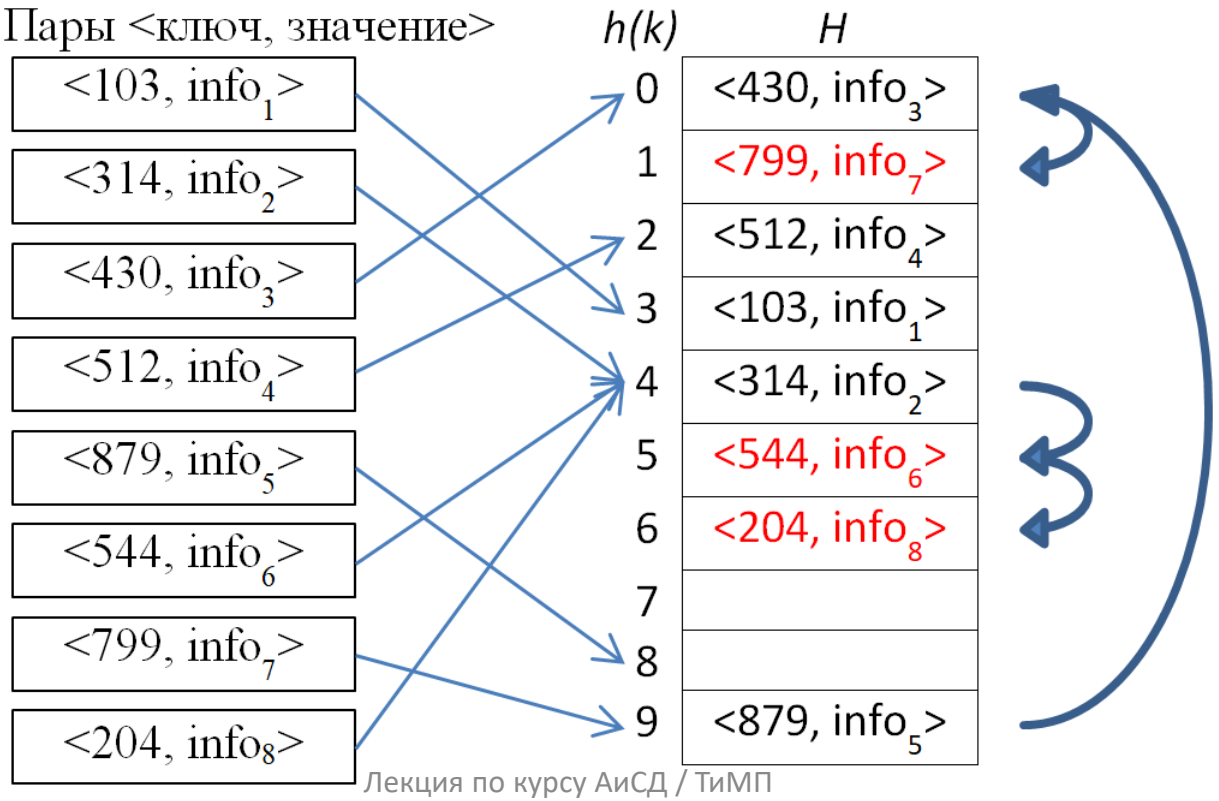
В отличие от рассмотренного выше метода, в методе открытой адресации не используются никакие дополнительные структуры данных, подобные спискам.
Вместо этого при возникновении коллизии добавляемую запись пытаются включить на свободное место в имеющейся таблице. Для этого предварительно договариваются, каким образом будет осуществляться поиск свободного места, и используют для поиска некоторый детерминированный алгоритм, определяющий так называемую ***последовательность про***б.

Если при добавлении записи оказывается что ячейка таблицы, соответствующая ключу вставляемой записи занята, выполняется просмотр ячеек, позиции которых определяются последовательностью проб. Запись при этом вставляется в первую встретившуюся пустую ячейку.

Если пустой ячейки найти не удается, это свидетельствует о заполнении таблицы и необходимости ее перестроения.

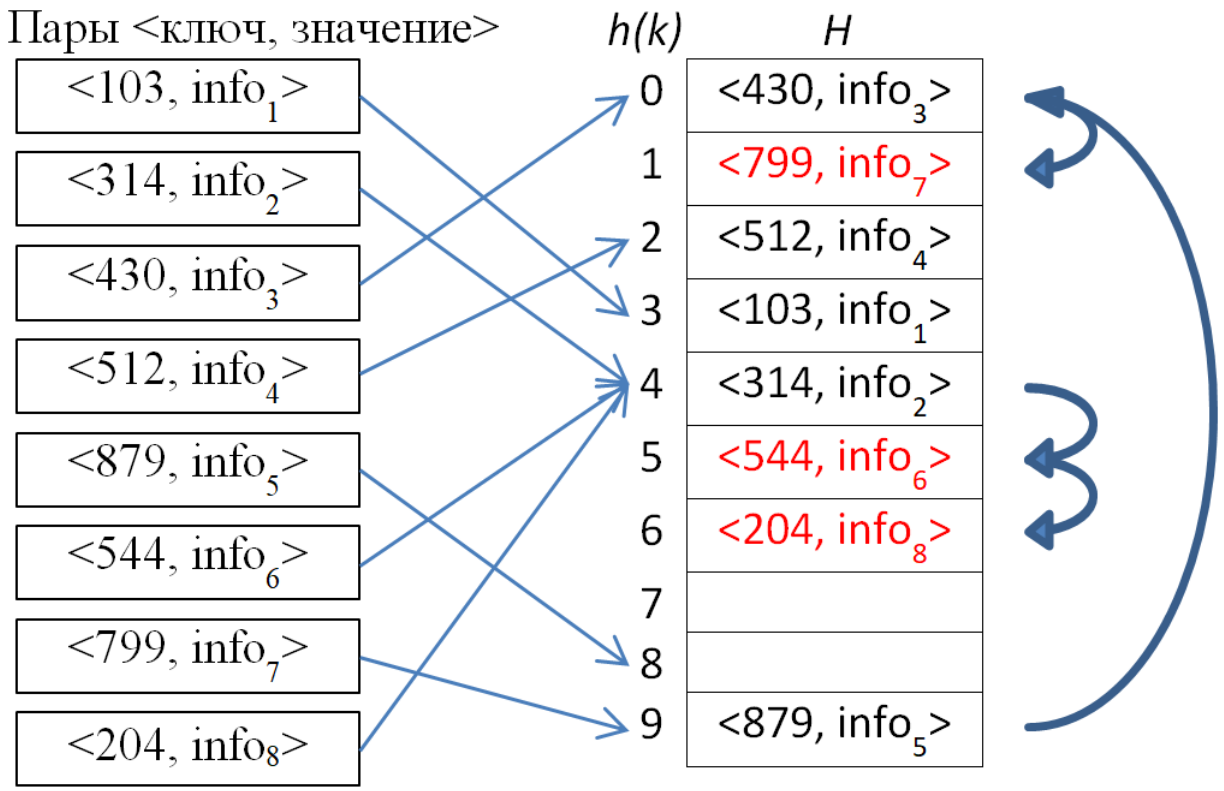
Метод открытой адресации: пример

Рассмотрим пример на приведенном ниже рисунке. На нем используется та же последовательность вставляемых записей, те же хеш-функция и размер хеш-таблицы. Для разрешения коллизий будет использоваться метод открытой адресации с *линейным пробированием с шагом* ***1***, то есть при вставке записи с ключом *k* будут просматриваться ячейки с индексами *(h(k)+i) mod K*, где *i =0..(K-1)* – номер пробы. При вставке первых пяти записей с ключами 103, 314, 430, 512 и 879 коллизий не возникает, и они попадают в ячейки с индексами 3, 4, 0, 2 и 9, соответственно.



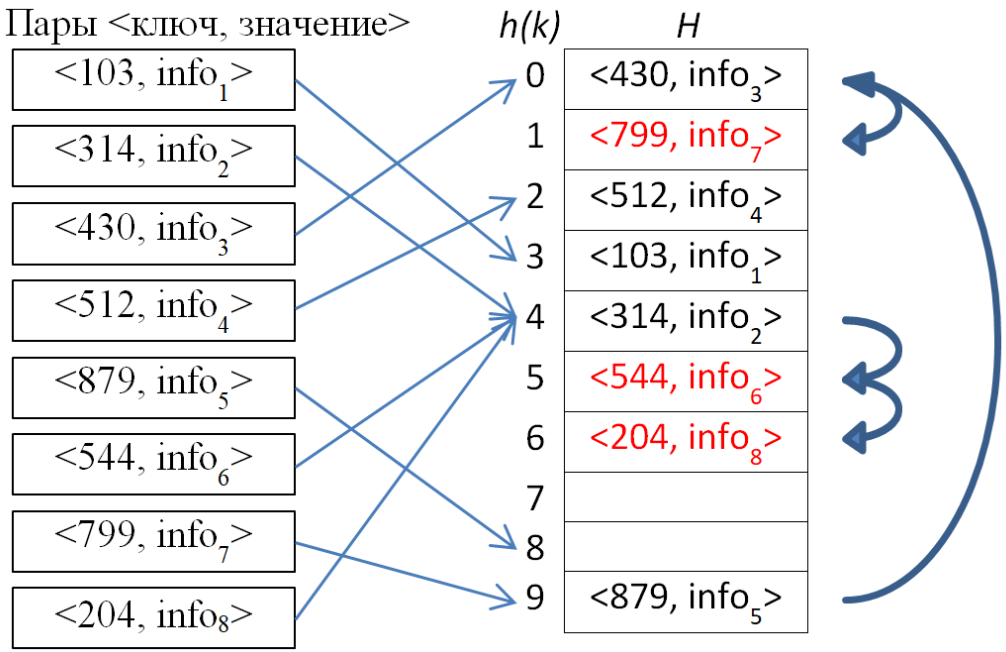
Метод открытой адресации: вставка записей

Далее при вставке записи с индексом ***544*** возникает коллизия с ранее добавленным ключом **314**, находящимся в ячейке с индексом **4**. По этой причине вставка осуществляется в свободную ячейку с индексом ***(h(544)+1) mod 10 = 5*.**



Метод открытой адресации: вставка записей

Вставка следующей записи с ключом **799** также вызывает коллизию с ранее размещенным ключом **879**. По этой причине проверяется ячейка с индексом ***(h(799)+1)mod 10 = 0,***
которая оказывается занята записью с индексом **430**. Следующая ячейка с индексом ***(h(799)+2)mod 10 = 1*** оказывается свободна, и вставка осуществляется в нее.
Аналогичным образом, вставка записи с ключом **204** осуществляется в ячейку с индексом 6 после последовательности проб ячеек с индексами **4** и **5**.

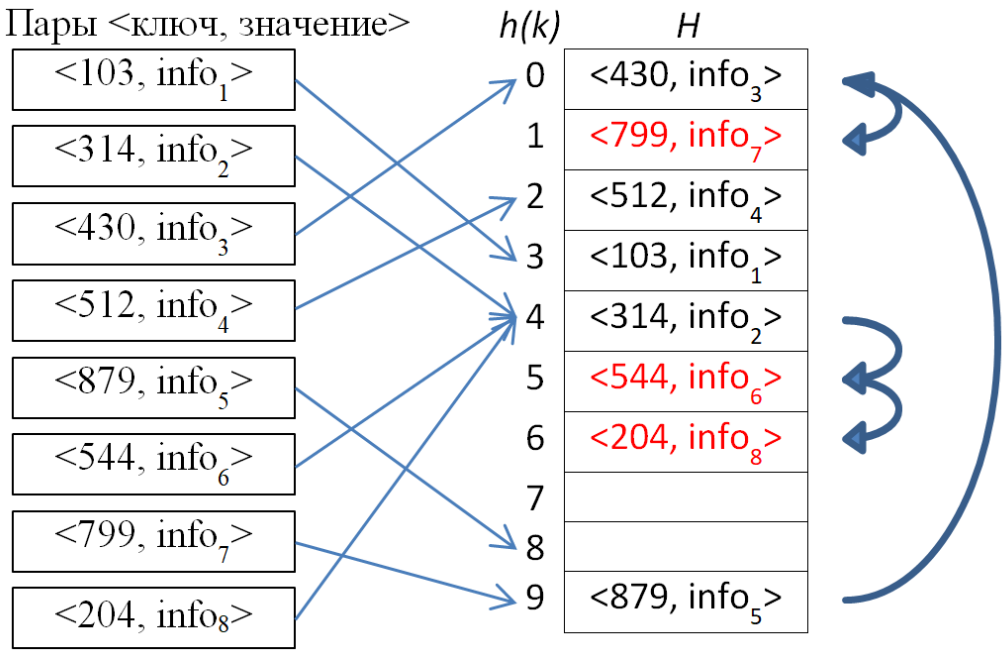


Метод открытой адресации: поиск записей

При поиске записи по ключу ячейки просматриваются по тому же алгоритму, что и при вставке. При этом переход к пустой ячейке свидетельствует об отсутствии искомой записи. Переход же к записи с другим ключом свидетельствует о необходимости продолжить поиск. Например, при поиске записи с ключом **724** мы осуществим последовательность проб ячеек с индексами

h(724)=4, (h(724)+1) mod 10 = 5, (h(724)+2) mod 10 = 6

и, убедившись, что ячейка ***(h(724)+3) mod 10 = 7*** пуста, сделаем вывод об отсутствии записи с ключом 724 в таблице.



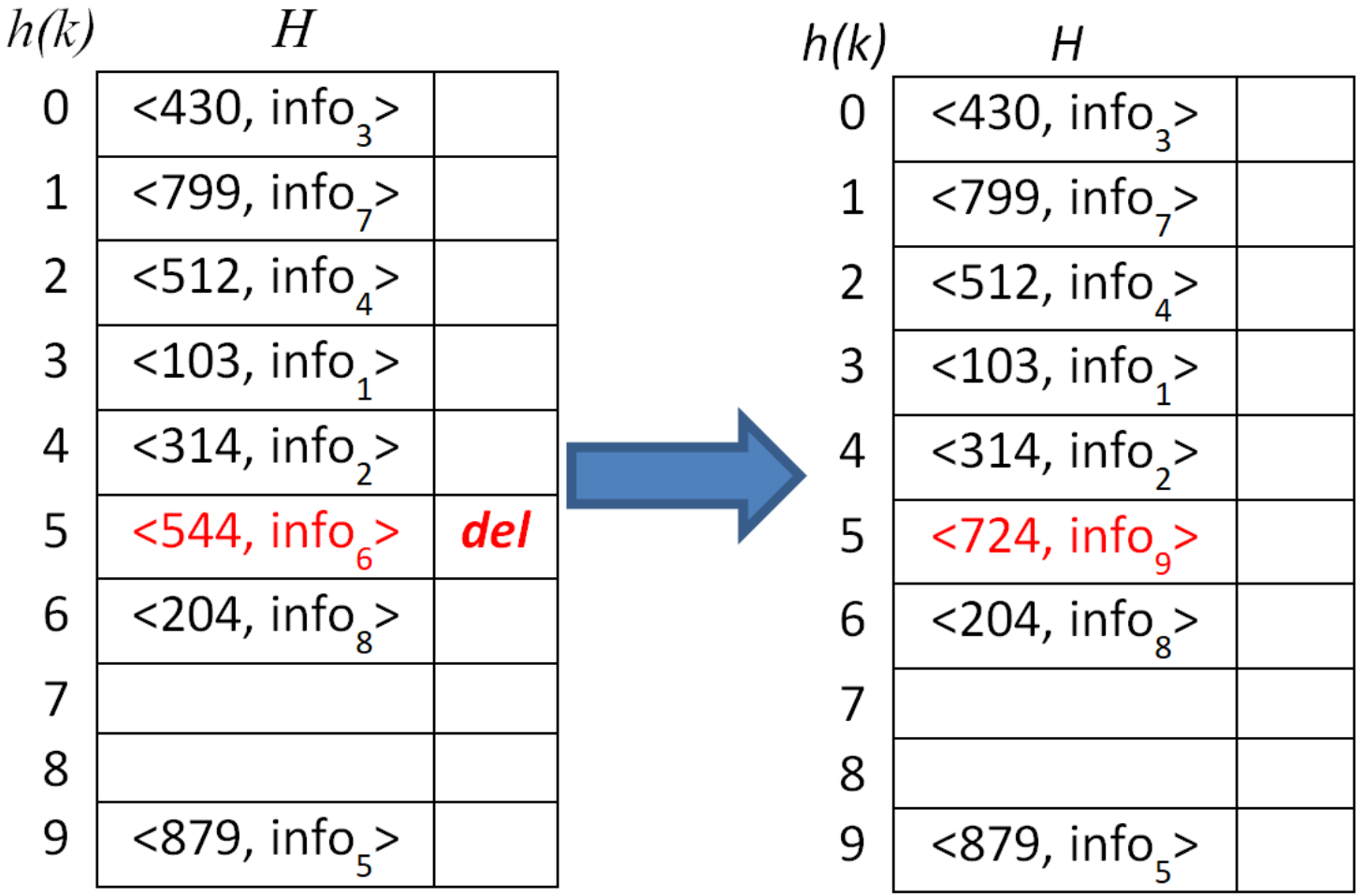
Метод открытой адресации: удаление записей

При *удалении* элемента из хеш-таблицы с открытой адресацией последовательность ячеек при последующем поиске (записи с другим ключом) может быть ошибочно прервана раньше времени из-за удаленной записи и сделан ошибочный вывод об отсутствии искомой записи в таблице. По этой причине при необходимости реализовать удаление в хеш-таблице с открытой адресацией для ячеек таблицы приходиться вводить ***дополнительный флаг, отмечающий удаленные записи.***

В этом случае само *удаление* записи сводится к поиску удаляемой записи с последующим выставлением флага. *Добавление* же записи может осуществляться на место ранее удаленной записи со сбросом флага. *Поиск* записи выполняется также как ранее с тем отличием, что при достижении записи с установленным флагом удаления ее содержимое не учитывается, а последовательность проб продолжается до нахождения пустой ячейки или записи с искомым значением ключа.

Метод открытой адресации: пример удаления

На приведенном ниже рисунке запись с ключом **544** оказывается удалена (выставлен флаг *del*), а потом на ее место вставлена запись с индексом **724** (флаг снят).



Метод открытой адресации: пробирование

Последовательность проб может вычисляться по различным алгоритмам. В частности, распространены линейное и квадратичное пробирование.

При ***линейном пробировании*** для некоторого ключа *k* будут просматриваться ячейки с индексами
(h(k)+ai) mod K,
где ***i =0..(K-1)*** – номер пробы, а ***a*** – шаг просмотра. В приведенном выше примере использовался шаг **a=1**. В любом случае, *a* должно быть взаимно простым с *K,* чтобы все ячейки таблицы были просмотрены по одному разу.
При квадратичном пробировании может использоваться схема
(h(k) + i(i+1)/2) mod *K*.
При такой схеме шаг между просматриваемыми ячейками с каждой пробой увеличивается: ***h(k) mod K, (h(k) + 1) mod K, (hash(k) + 3) mod K,…***
Могут использоваться и более сложные схемы, в которых размер шага определяется второй хеш-функцией, отличной от используемой для вычисления первичного индекса ячейки. Такие схемы называются схемами с двойным хешированием.

Хеш-функции

Использованная нами ранее простая хеш-функция является лишь одним из множества возможных вариантов. Для того чтобы функция могла быть использована при построении хеш-таблиц, помимо очевидного требования к области ее значений (целочисленные значения в диапазоне
0<=h(k)<K, где ***K*** – размер массива хеш-таблицы),
она должна удовлетворять следующим двум основным условиям:
- функция должна быстро вычисляться,
- функция должна минимизировать количество коллизий.

Двумя наиболее распространенными схемами вычисления хеш-функций являются:
Использованная нами ранее простая схема с делением: ***h(k) = k mod m,***
в которой для минимизации коллизий в качестве ***m*** берется простое число,

Хеш-функции: мультипликативная схема

Хеш-функция при использовании ***мультипликативной схемы*** может быть описана выражением

$$h(k) = \left\lfloor K \left(\left(\frac{a}{w} k \right) mod\ 1 \right) \right\rfloor,$$

где ***w*** – размер машинного слова, ***a*** – достаточно большая целая константа, взаимно простая с ***w***, операция "***p mod 1***" возвращает дробную часть числа ***p***, а "**[*p*]**" – целую часть числа p.

Здесь от произведения $\frac{a}{w} k$, представляющего собой некоторое действительное число (зависящее от значения ключа), берется дробная часть, в результате чего получается действительное число в диапазоне от **[0; 1)**. Будучи умноженным на ***K*** (размер массива хеш-таблицы), произведение находится в диапазоне **[0;*K*)**, а его целая часть представляет собой целое в диапазоне **0..(*K*-1)** и используется в качестве значения хеш-функции.

Хеширование строк

Во многих случаях ключ не является просто целочисленным значением, а представляет собой некоторую последовательность (например, символов) $k = k_1,k_2,...k_n$. В этом случае, значение хеш-функции может быть рассчитано с использованием набора $h_1, h_2, ..h_m$ по следующей схеме:
$h(k) = (h_1(k_1)+h_2(k_2)+...h_n(k_n)) \bmod K.$

В случае, когда в качестве ключа поступает последовательность символов неизвестной наперед длины, для построения хеш-функции удобно использовать ***алгоритм Пирсона***, представленный ниже.

```
h = 0

Для каждого символа c из строки S:

        h = T[h xor c]
```

На вход алгоритма поступает строка ***S***, выполняющая роль ключа, и заранее подготовленная таблица перестановок ***T***. На выходе формируется значение хеша ***h***.
Здесь выражение ***h xor c*** представляет собой индекс в таблице перестановок ***T***. Сама таблица может представлять собой случайную перестановку последовательности значений

Хеширование строк: таблица перестановок

```
T = {
 98,  6, 85,150, 36, 23,112,164,135,207,169,  5, 26, 64,165,219,
 61, 20, 68, 89,130, 63, 52,102, 24,229,132,245, 80,216,195,115,
 90,168,156,203,177,120,  2,190,188,  7,100,185,174,243,162, 10,
237, 18,253,225,  8,208,172,244,255,126,101, 79,145,235,228,121,
123,251, 67,250,161,  0,107, 97,241,111,181, 82,249, 33, 69, 55,
 59,153, 29,  9,213,167, 84, 93, 30, 46, 94, 75,151,114, 73,222,
197, 96,210, 45, 16,227,248,202, 51,152,252,125, 81,206,215,186,
 39,158,178,187,131,136,  1, 49, 50, 17,141, 91, 47,129, 60, 99,
154, 35, 86,171,105, 34, 38,200,147, 58, 77,118,173,246, 76,254,
133,232,196,144,198,124, 53,  4,108, 74,223,234,134,230,157,139,
189,205,199,128,176, 19,211,236,127,192,231, 70,233, 88,146, 44,
183,201, 22, 83, 13,214,116,109,159, 32, 95,226,140,220, 57, 12,
221, 31,209,182,143, 92,149,184,148, 62,113, 65, 37, 27,106,166,
  3, 14,204, 72, 21, 41, 56, 66, 28,193, 40,217, 25, 54,179,117,
238, 87,240,155,180,170,242,212,191,163, 78,218,137,194,175,110,
 43,119,224, 71,122,142, 42,160,104, 48,247,103, 15, 11,138,239
}
```

Хеширование строк: пример

С использованием описанного алгоритма вычисление хеш-функции, например, от строки “*abc*” будет производиться за три итерации следующим образом:

h = 0

h = T[0 xor 97] = T[97] = 96 //97 - код символа ‘a’

h = T[96 xor 98] = T[2] = 85 //0110 0000_2 xor 0110 0010_2 = 0000 0010_2 = 2

h = T[85 xor 99] = T[54] = 172 //0101 0101_2 xor 0110 0011_2 = 0011 0110_2 = 54

```
T = {
 98,  6, 85,150, 36, 23,112,164,135,207,169,  5, 26, 64,165,219,
 61, 20, 68, 89,130, 63, 52,102, 24,229,132,245, 80,216,195,115,
 90,168,156,203,177,120,  2,190,188,  7,100,185,174,243,162, 10,
237, 18,253,225,  8,208,172,244,255,126,101, 79,145,235,228,121,
123,251, 67,250,161,  0,107, 97,241,111,181, 82,249, 33, 69, 55,
 59,153, 29,  9,213,167, 84, 93, 30, 46, 94, 75,151,114, 73,222,
197, 96,210, 45, 16,227,248,202, 51,152,252,125, 81,206,215,186,
 39,158,178,187,131,136,  1, 49, 50, 17,141, 91, 47,129, 60, 99,
154, 35, 86,171,105, 34, 38,200,147, 58, 77,118,173,246, 76,254,
133,232,196,144,198,124, 53,  4,108, 74,223,234,134,230,157,139,
189,205,199,128,176, 19,211,236,127,192,231, 70,233, 88,146, 44,
183,201, 22, 83, 13,214,116,109,159, 32, 95,226,140,220, 57, 12,
221, 31,209,182,143, 92,149,184,148, 62,113, 65, 37, 27,106,166,
  3, 14,204, 72, 21, 41, 56, 66, 28,193, 40,217, 25, 54,179,117,
238, 87,240,155,180,170,242,212,191,163, 78,218,137,194,175,110,
 43,119,224, 71,122,142, 42,160,104, 48,247,103, 15, 11,138,239
}
```

Другие виды хэш-функций

Следует отметить, что помимо создания хеш-таблиц, хеш-функции используются также:

- в ***криптографии*** (криптографически стойкие хеш-функции)

Требования:

- невозможность восстановления данных по значению хэш-функции
- невозможность подбора других данных с тем же значением хэш-функции или подбора пары различных блоков данных с одинаковым хэшем
- небольшие изменения в данных должны приводить к существенному изменению значения хэш-функции.
- невозможно предсказать значение хэш-функции при дополнении сообщения

- при ***передаче и хранении данных*** (обнаружение ошибок с использованием контрольных сумм)

Требования:

- высокая скорость работы
- возможность аппаратной реализации

- в ***компьютерной графике и 2D/3D моделировании*** (геометрическое хеширование) и других областях.

Другие виды хэш-функций

При решении ряда проблем, в том числе при построении хеш-таблиц, используется подход на основе множества хеш-функций, при котором выбор конкретной хеш-функции осуществляется по некоторому случайному алгоритму. Такой способ называется ***универсальным хешированием***.