Хеширование данных

Хеширование - разбиение множества ключей (однозначно характеризующих элементы хранения и представленных, как правило, в виде текстовых строк или чисел) на непересекающиеся подмножества (наборы элементов), обладающие определенным свойством.

Хеширование данных

Hashing или scatter storage (рассеянная память) – крошить, размалывать, рубить и тд (рассеяние, расстановка, рандомизация)

Идея хеширования состоит в использовании некоторой частичной информации, полученной из ключа, в качестве основы поиска, те вычисляется хеш-адрес h(K), который используется для поиска

Хеширование данных

адрес = h (ключ)

таблица (файл базы данных)

Ключ	поле данных	

поля (имеют фиксированный тип)

Хеширование данных

- Пример
- Файл клиентов пункта проката DVD содержит десятизначные номера сотовых телефонов.
- Номер телефона используется в качестве ключа для доступа к конкретной записи файла клиентов.

Хеширование данных

Индекс	Ключ - Номер телефона	Персональные данные
0	9039130722	Тихомиров Д.С., Советская84/3-313; Прометей; Облачный атлас; Голодные игры
1	9234295772	Иванова С.А., Вершинина 48а -423; Джунгли; Дублер
2	.....	

- Временная сложность – O(1)

Хеширование данных

- Идеальной хеш-функцией является такая hash-функция, которая для любых двух неодинаковых ключей дает неодинаковые адреса.

$$k1 \neq k2 \Rightarrow h(k1) \neq h(k2)$$

- (парадокс дней рождения)

Хеширование данных

Позиция	Ключ	Запись
0	4967000	
1		
2	8421002	
3		
⋮	⋮	
395		
396	4618396	
397	4957397	
398		
399	1286399	
400		
401		
⋮	⋮	
990	0000990	
991	0000991	
992	1200992	
993	0047993	
994		
995	9846995	
996	4618996	
997	4967997	
998		
999	0001999	

h(key):= key mod 1000

Выбор хеш-функции

1. **Метод деления.**

Некоторый целый ключ делится на размер таблицы и остаток от деления берется в качестве значения хеш-функции.

2. **Метод середины квадрата.**

Ключ умножается сам на себя и в качестве индекса используется несколько средних цифр этого квадрата.

```
Function h(key: integer): integer;
    Begin
    key:=key*key; {Возвести в квадрат}
    KEY:=key shl 11;{Отбросить 11
    младших бит}
    h:= key mod 1024;{Возвратить 10
    младших бит}
    End;
```

Выбор хеш-функции

3. Аддитивный метод для строк (размер таблицы равен 256)

```
Function h(st: string): integer;
    Var
    Sum: longint;
    I: integer;
    Begin
    For i:=0 to length(st) do
    Sum := sum + ord(st[i]);
    h:=sum mod 256;
    End;
```

Выбор хеш-функции

4. Исключающее ИЛИ для строк (размер таблицы равен 256).

```
Var
    rand8: array[0..255] of integer;

    Procedure init;
    var
    i: integer;
    begin
     randomize;
     for i:=0 to 255 do
      rand8[i]:=random(255);
    End;
```

Выбор хеш-функции

```
Function h(st: string): integer;
  var
  sum: longint;
  I: integer;
  begin
  for i:=0 to length(st) do
    sum := sum + ord(st[i]) xor
                          rand8[i];
  h:=sum mod 256;
  End;
```

Хеширование данных

```
Function H (x: string[10]): 0..n-1;
  var I, sum: integer;
  begin
    sum:= 0;
    for I:=1 to 10 do
      sum := sum + ord ( x[I] );
    h :=sum mod n
  end;
```

Хеширование данных

- Пример реализации несовершенной хеш-функции:

Предположим ключ состоит из четырех символов;

таблица имеет диапазон адресов от 0 до 10000.

Хеширование данных

- **function** hash (key : **string**[4]): **integer**;

```
var f: longint;
begin
f:=ord (key[1]) - ord (key[2]) + ord (key[3])
   -ord (key[4]);
{вычисление функции по значению
   ключа}
f:=f+255*2;
{совмещение начала области значений
   функции с начальным адресом хеш-
   таблицы (a=1)}
```

Хеширование данных

{совмещение начала области значений функции с начальным адресом хеш-таблицы (a=1)}

f:=(f*10000) **div** (255*4);

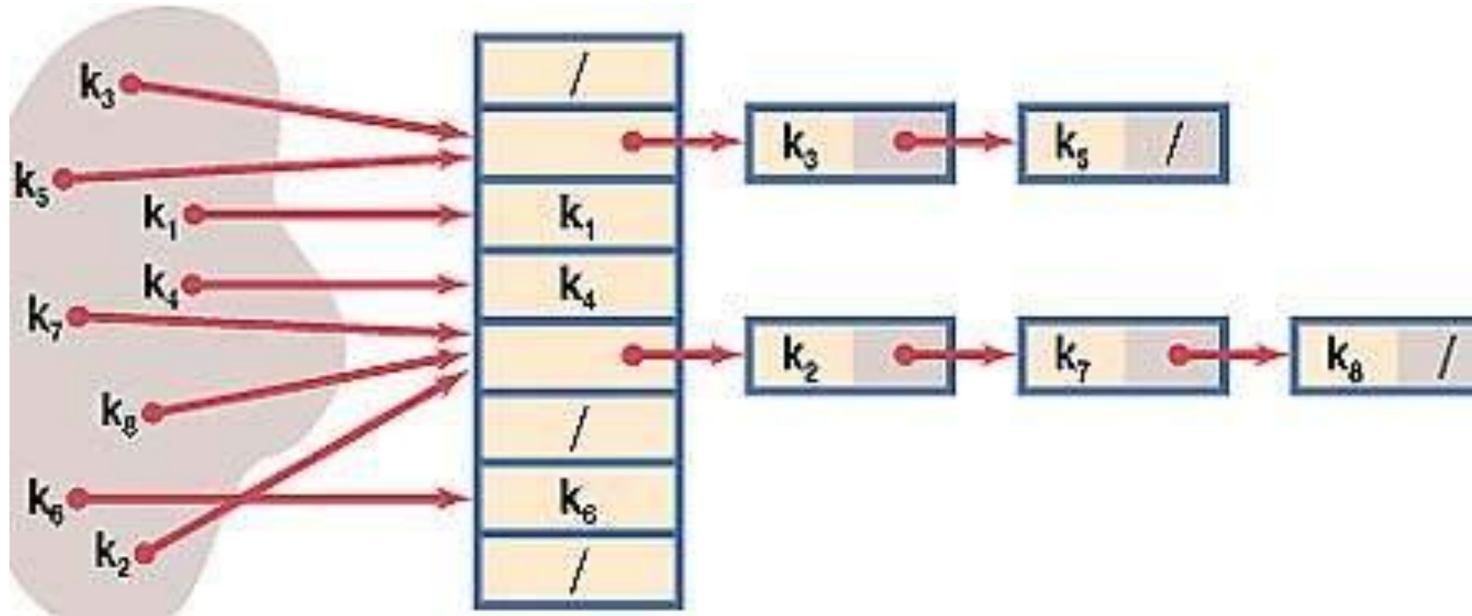
{совмещение конца области значений функции с конечным адресом хеш-таблицы (a=10 000)}

hash:=f

end;

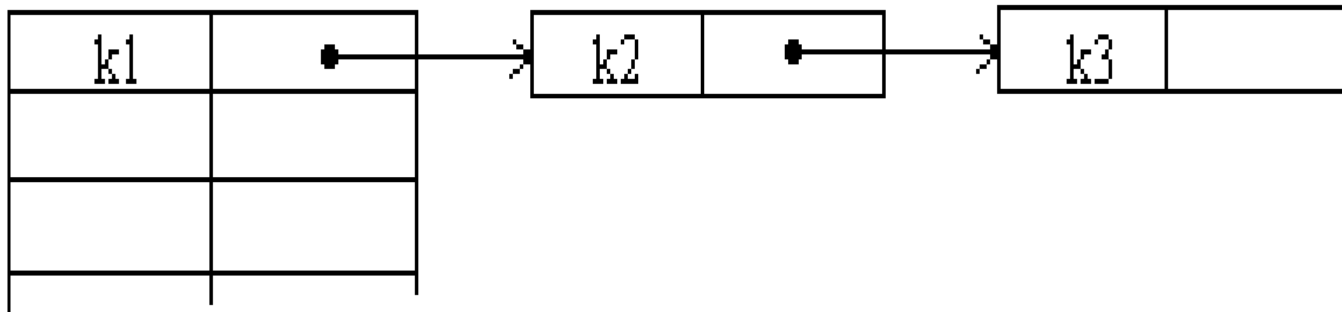
Хеширование данных

Разрешение коллизий: метод цепочек



(www.cs.mcgill.ca/~cs251/OldCourses/1997/topic12).

Разрешение коллизий: метод цепочек



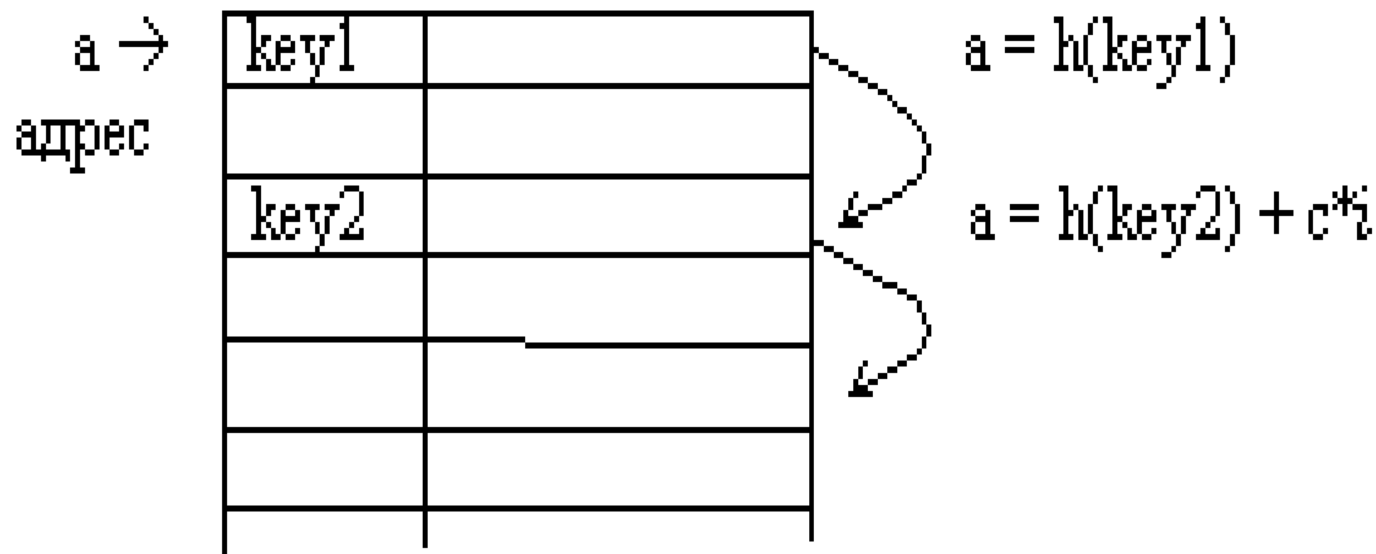
$k1 \neq k2$

$h(k1)=h(k2)$

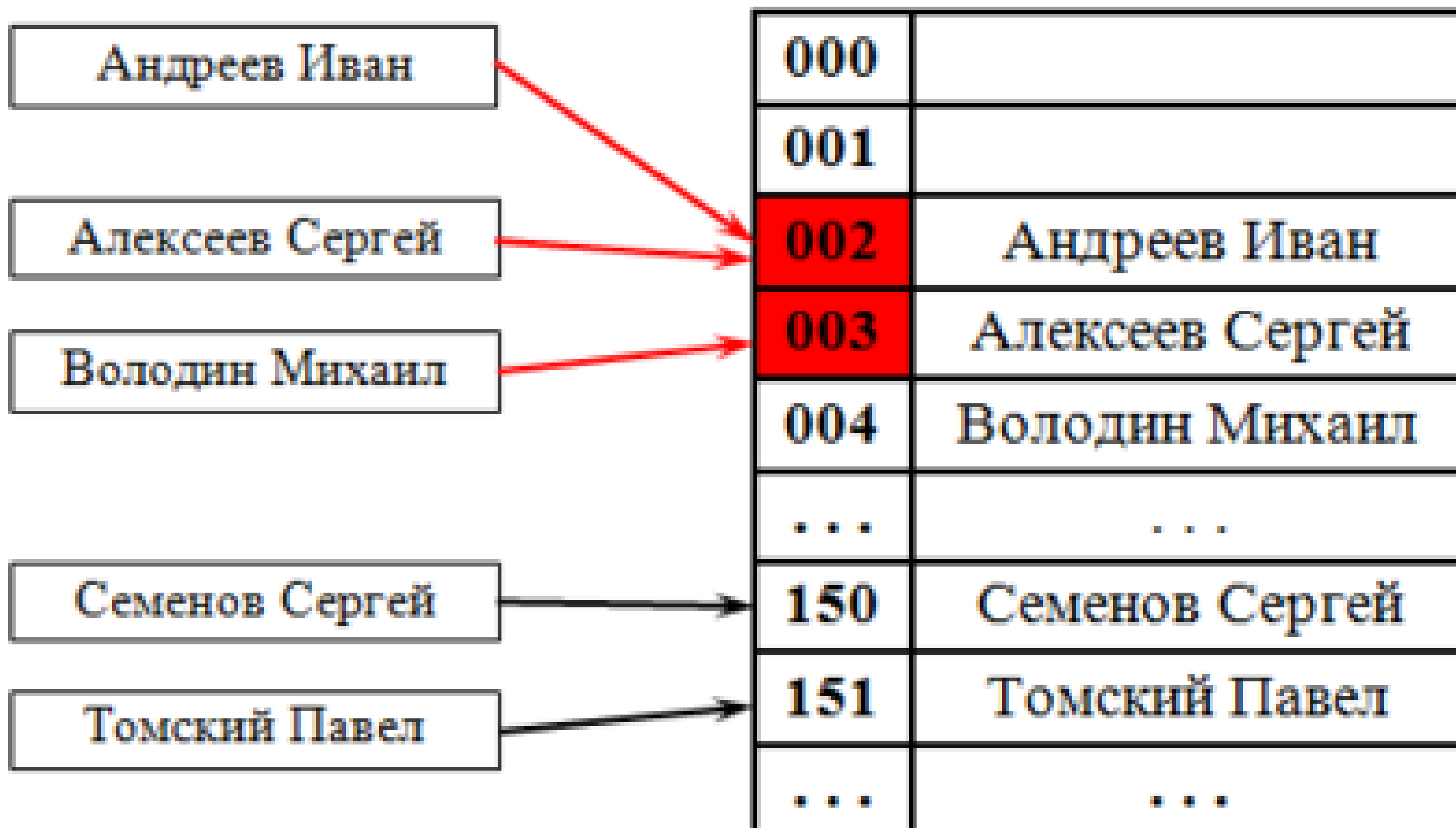
$k3 \neq k1$

$h(k3)=h(k1)$

Разрешение коллизий: линейное опробование

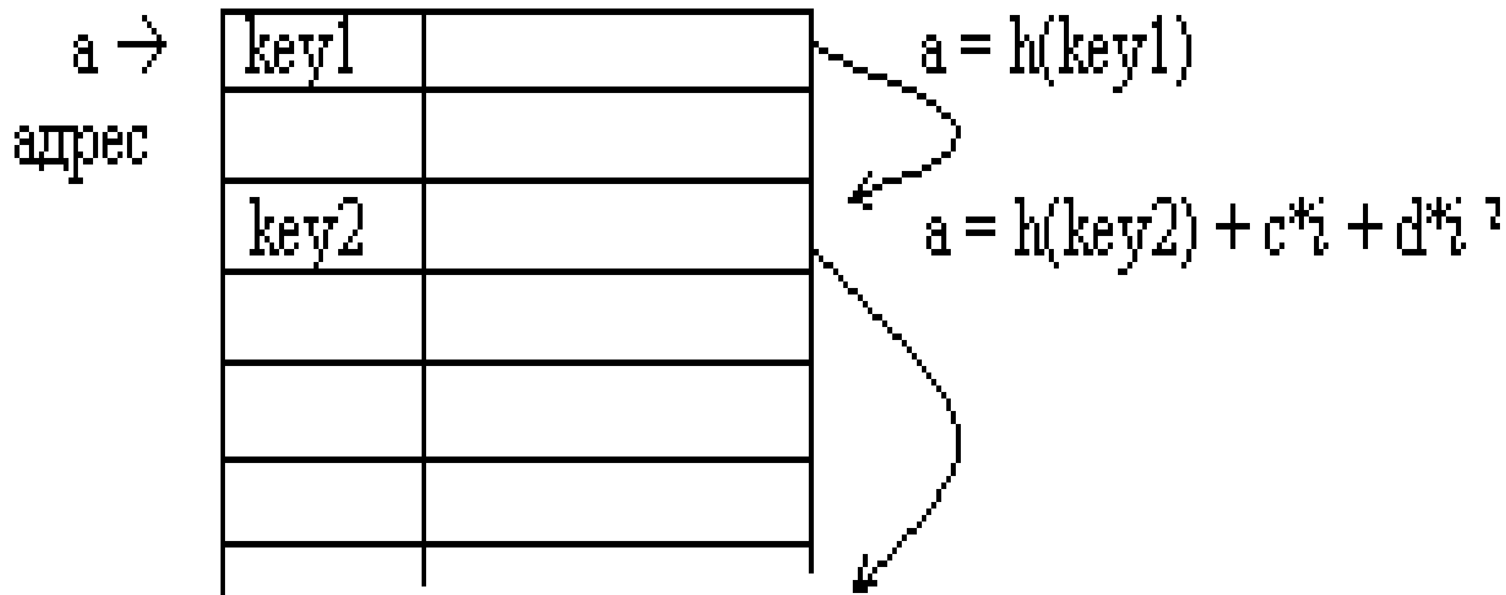


Разрешение коллизий: линейное опробование



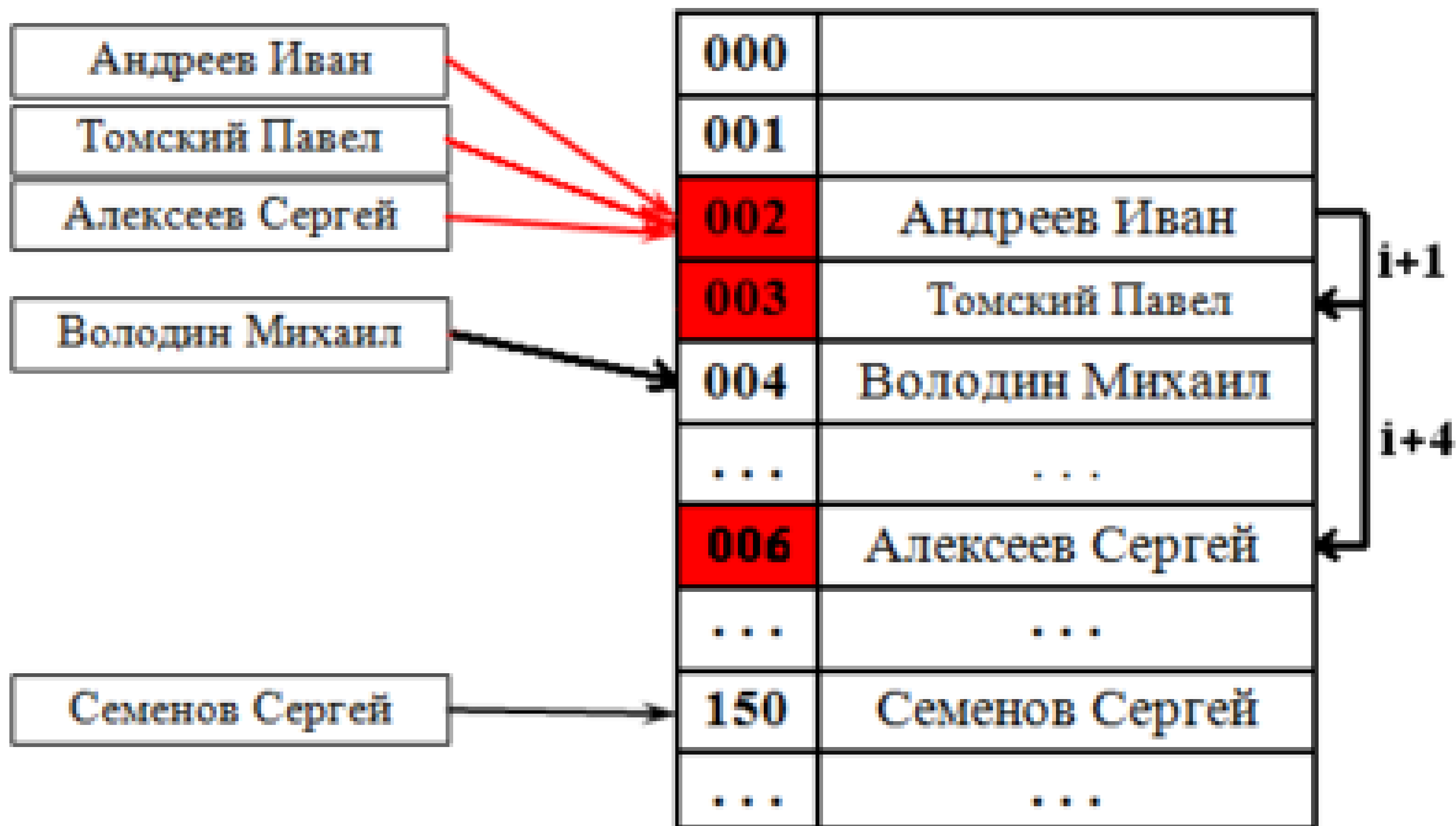
Разрешение коллизий: линейное опробование

Разрешение коллизий: квадратичное опробование



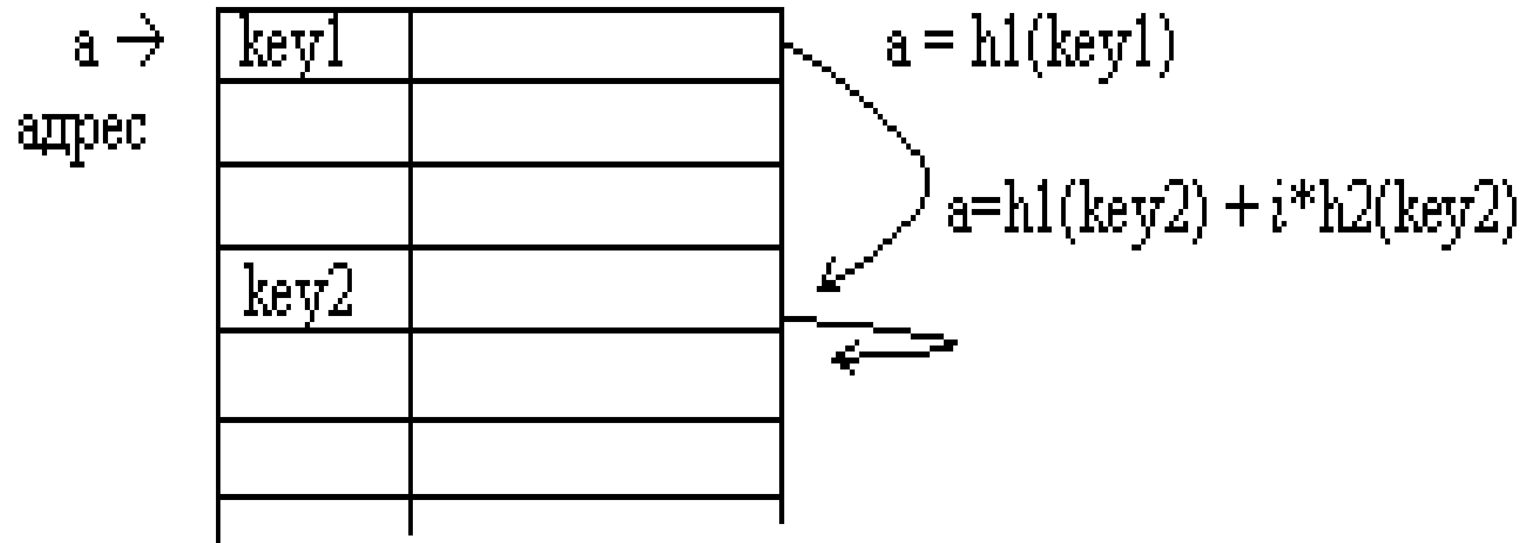
key1 ≠ key2
a=h(key1) = h(key2)

Разрешение коллизий: квадратичное опробование



Разрешение коллизий: двойное хеширование

в) Двойное хеширование



key1 ≠ key2

a=h1(key1) = h1(key2)

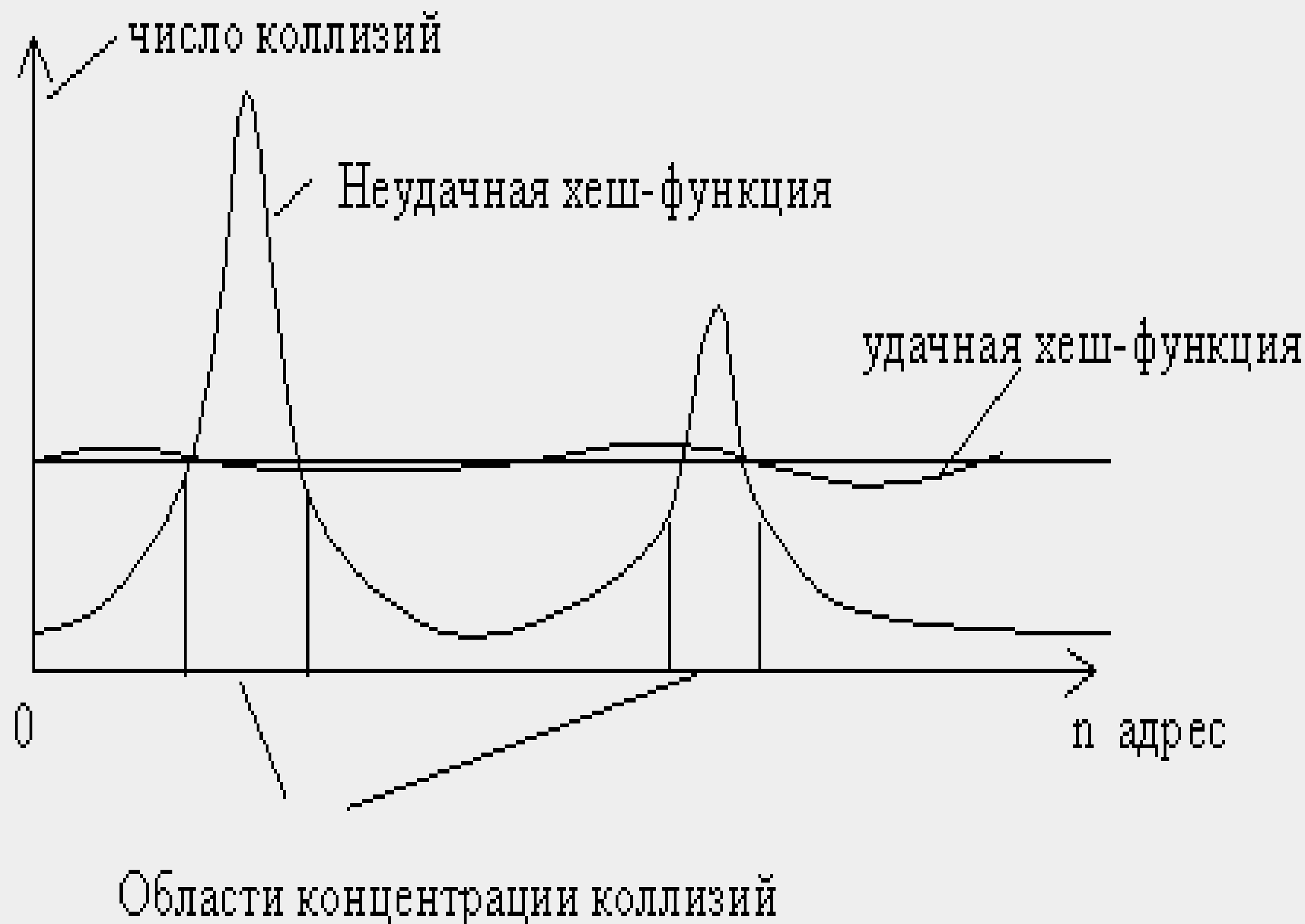
Линейное опробование: вставка

1. $i = 0$
2. $a = h(key) + i*c$
3. Если $t(a)$ = свободно, то $t(a) = key$, записать элемент, ***стоп*** ***элемент добавлен***
4. $i = i + 1$, перейти к шагу 2

Линейное опробование: поиск

1. **i = 0**
2. **a = h(key) + i*c**
3. Если **t(a) = key**, то ***стоп элемент найден***
4. Если **t(a) = свободно**, то ***стоп элемент не найден***
5. i = i + 1, перейти к шагу 2

Оценка качества хеш-функции



Хеш-таблицы представляют собой распространенный механизм для хранения пар ключ/элемент.

Они обладают такими достоинствами, как универсальность и простота, а также высокая эффективность при хорошо продуманной генерации хеш-кода.

Класс Hashtable расширяет Dictionary и реализует интерфейс Map. Он обладает определенной емкостью и средствами, определяющими момент увеличения таблицы.

Расширение хеш-таблицы требует повторного хеширования всех ее элементов в соответствии с их новым положением в увеличенной таблице, так что важно обеспечить однократное изменение таблицы

Класс HashTable

- Значение хеш-кода возвращается методом hashCode() для объекта, являющегося ключом.
- По умолчанию каждый объект имеет уникальный хеш-код.
- Использование в качестве ключей случайно выбранных объектов приводит к порождению различных хеш-кодов.
- Классы String, BitSet и большинство других, переопределяющих метод equals(), обычно переопределяют и hashCode()

Хеш-код для класса String

```
int hash = 0;
for (int i = 0; i < length(); i++)
   hash = 31 * hash + charAt(i);
```

Строка	Хэш-код
Hello	140207504
Harry	140013338
Hacker	884756206

Класс HashTable

```
String s = "Ok";
StringBuffer sb = new StringBuffer(s);
System.out.println(s.hashCode() + " " + sb.hashCode());
String t = new String("Ok");
StringBuffer tb = new StringBuffer(t);
System.out.println(t.hashCode() + " " + tb.hashCode());
```

Объект	**Хэш-код**
s	3030
sb	20526976
t	3030
tb	20527144

Класс HashTable

Объекты Hashtable автоматически увеличиваются, когда они становятся слишком заполненными.

Под выражением “слишком заполненными” понимается превышение *показателя загрузки* таблицы, который представляет собой отношение количества элементов к текущей емкости таблицы.

Когда таблица увеличивается, ее новая емкость примерно вдвое превышает текущую.

Для повышения эффективности следует выбирать емкость, представленную простым числом, чтобы при увеличении объекта Hastable также было выбрано ближайшее простое число.

Исходная емкость хеш-таблицы и показатель загрузки могут задаваться в конструкторах Hashtable:

Класс HashTable

- public **Hashtable()**

Конструирует новую, пустую хеш-таблицу с принятой по умолчанию исходной емкостью и показателем загрузки, равным 0, 75.

- public **Hashtable (int initialCapacity)**

Конструирует новую, пустую хеш-таблицу с заданной емкостью initial Capacity и принятым по умолчанию показателем загрузки, равным 0,75.

- public **Hashtable(int initialCapacity, float loadFactor)**

Example