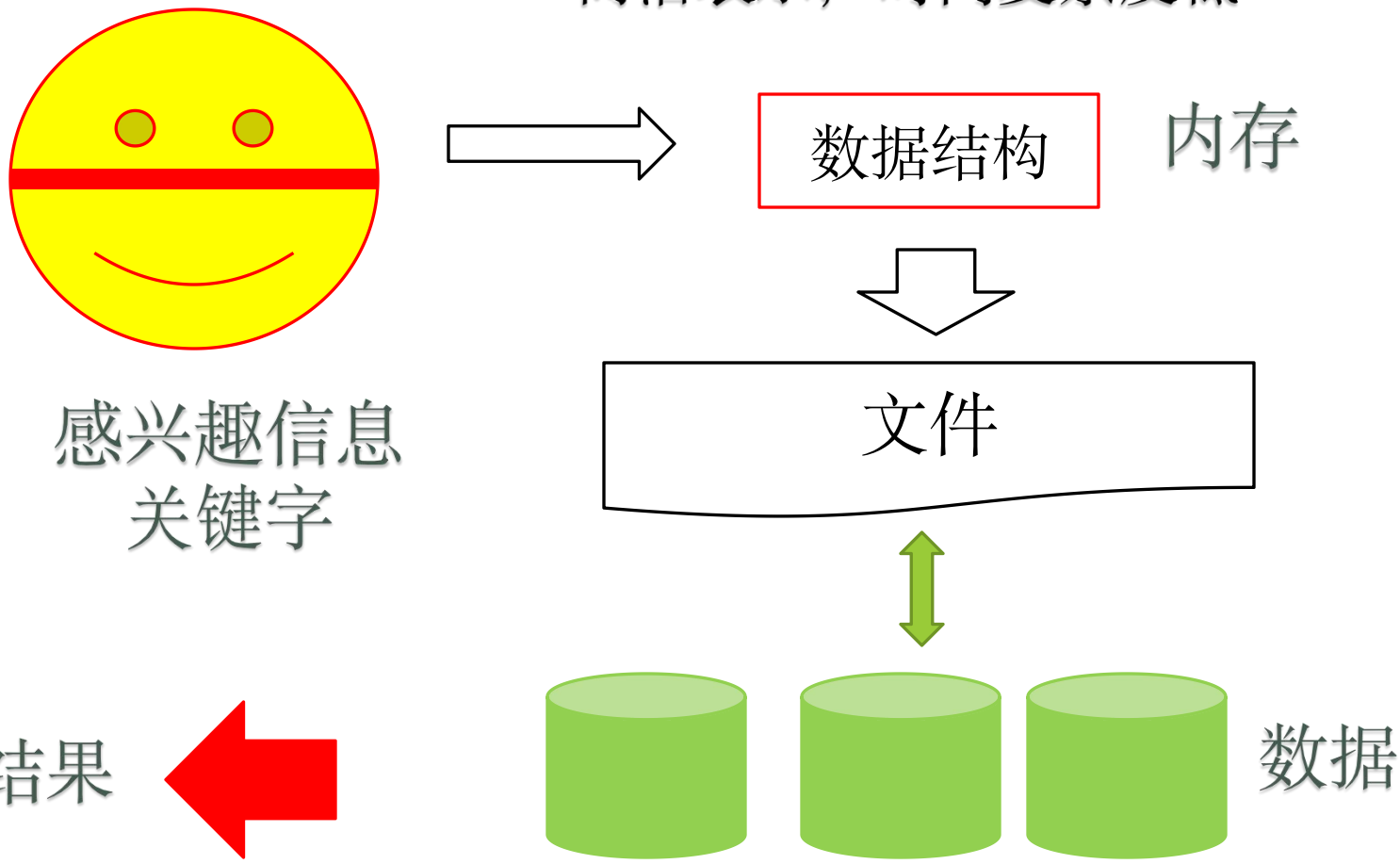
哈希策略

湖南大学信息科学与工程学院

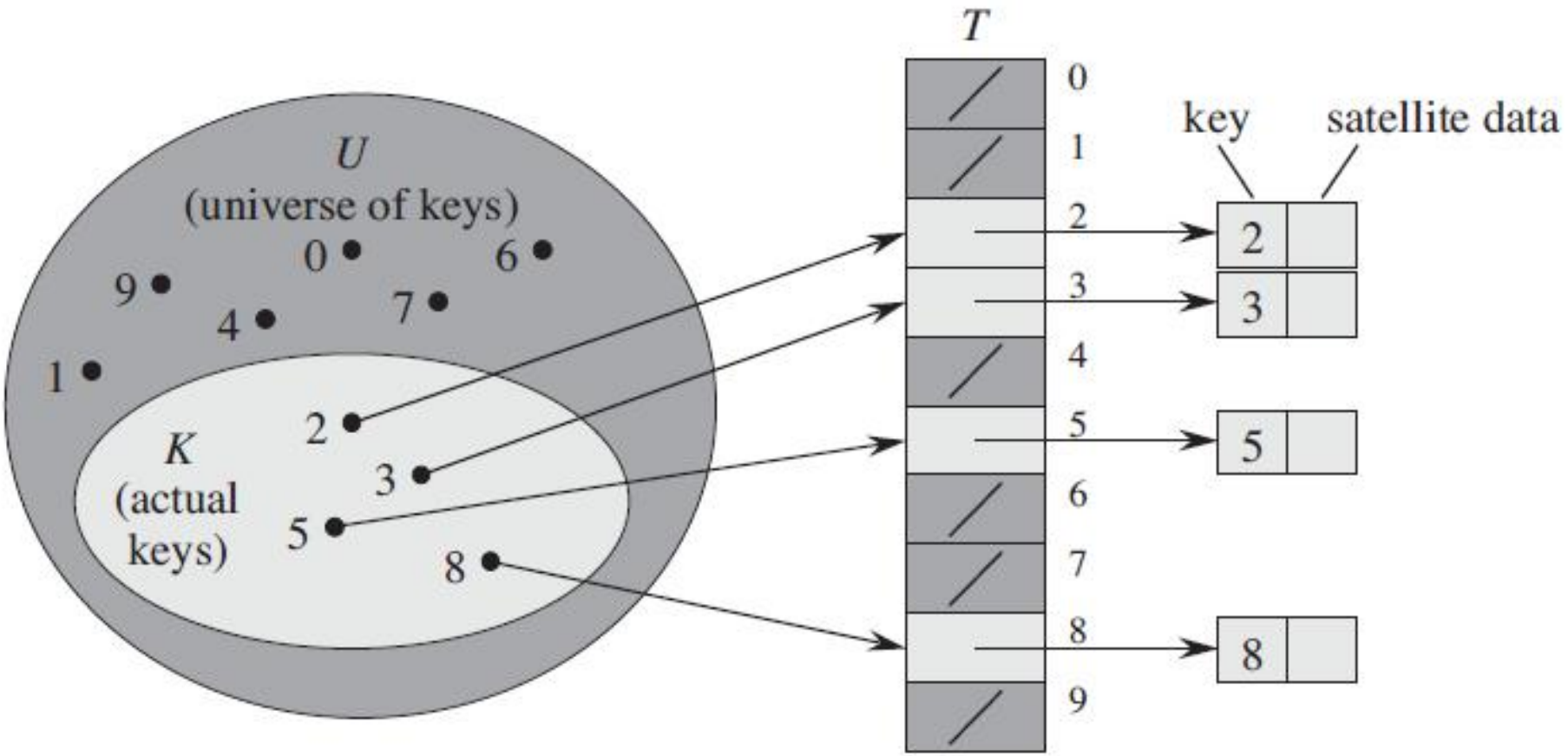
提纲

4.1概述



提纲

4.2 直接寻址表


T
U
(universe of keys)
K
(actual
keys)
key
satellite data
0
1
2
3
4
5
6
7
8
9
2
3
5
8

4.2 直接寻址表

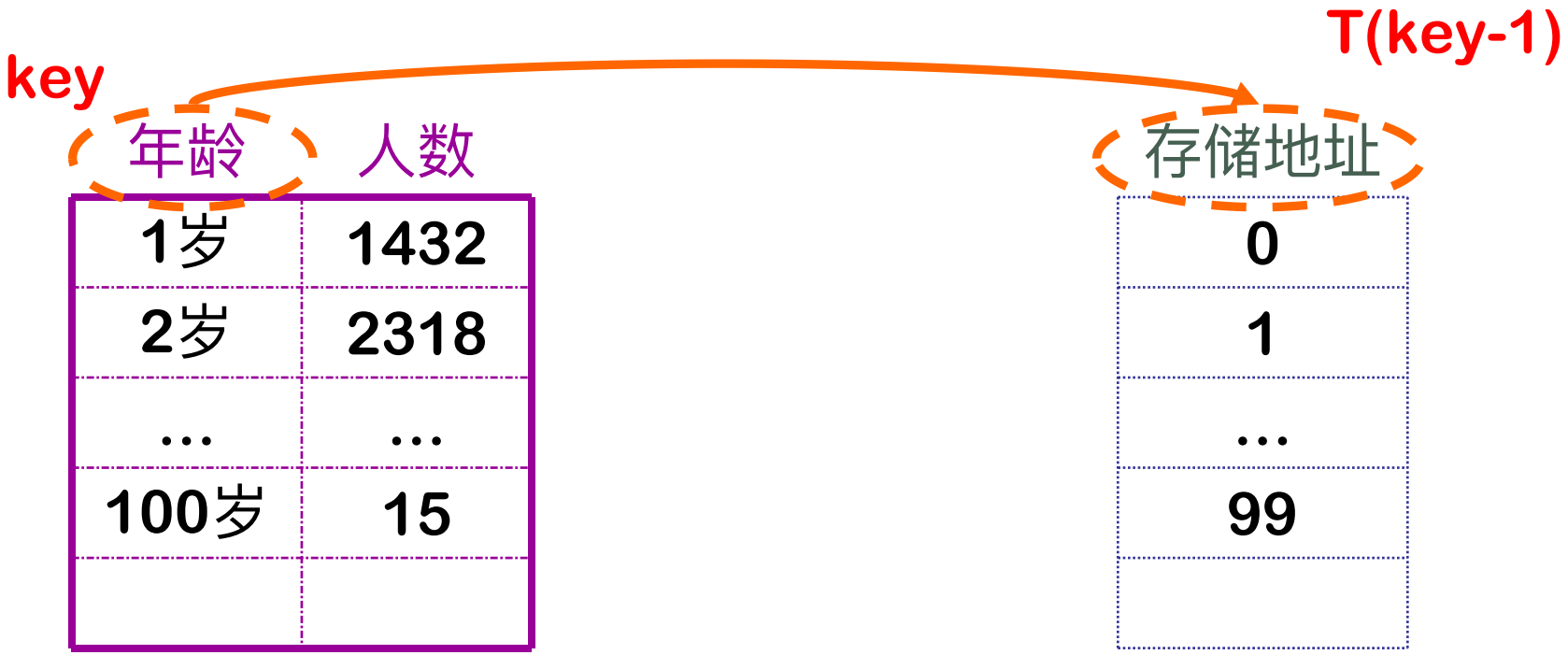
```
DIRECT-ADDRESS-SEARCH(T, k)
1  return T[k]

DIRECT-ADDRESS-INSERT(T, x)
1  T[x.key] = x

DIRECT-ADDRESS-DELETE(T, x)
1  T[x.key] = NIL
```

4.2 直接寻址表

- 举例

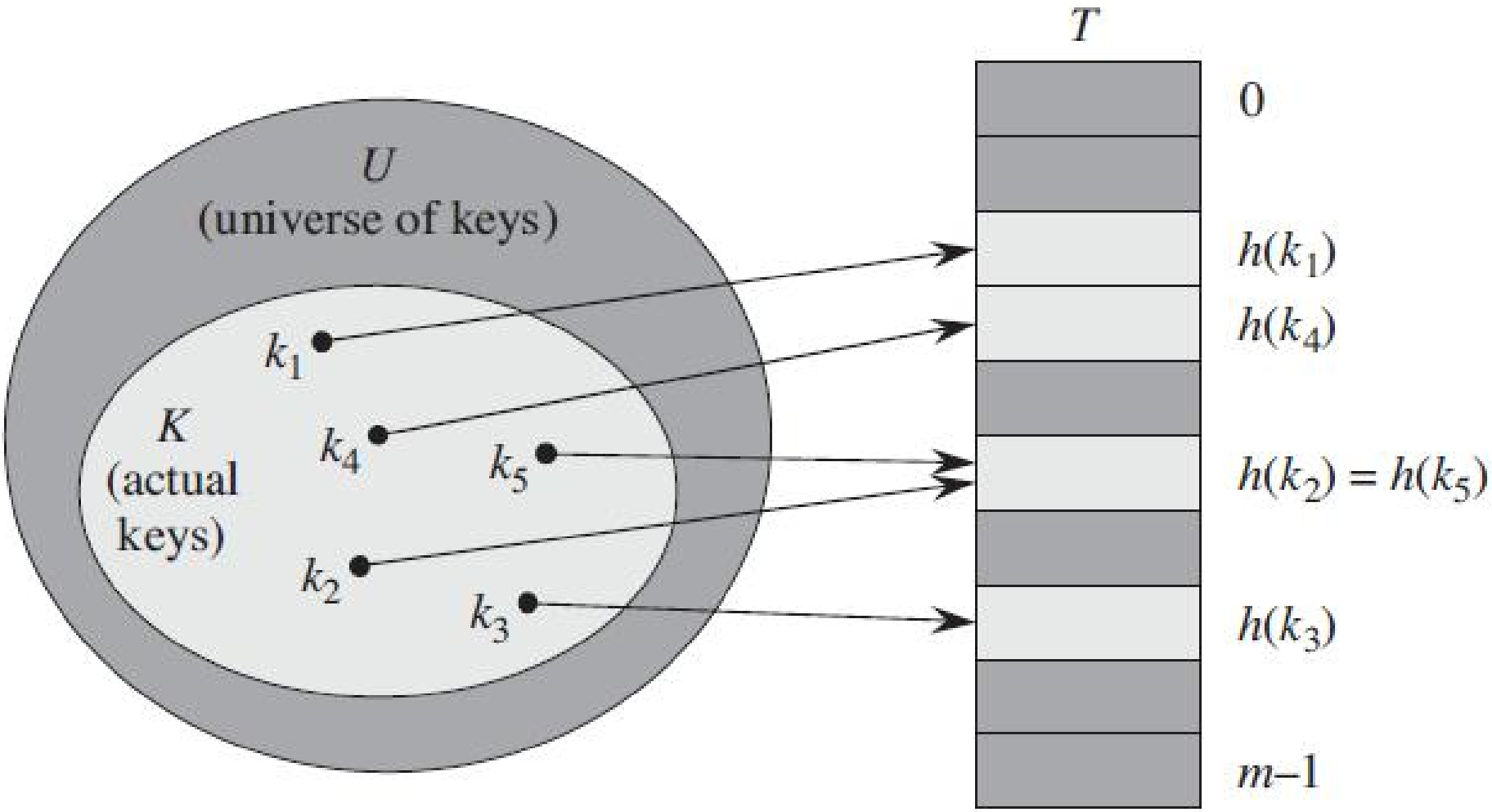


直接寻址的缺点

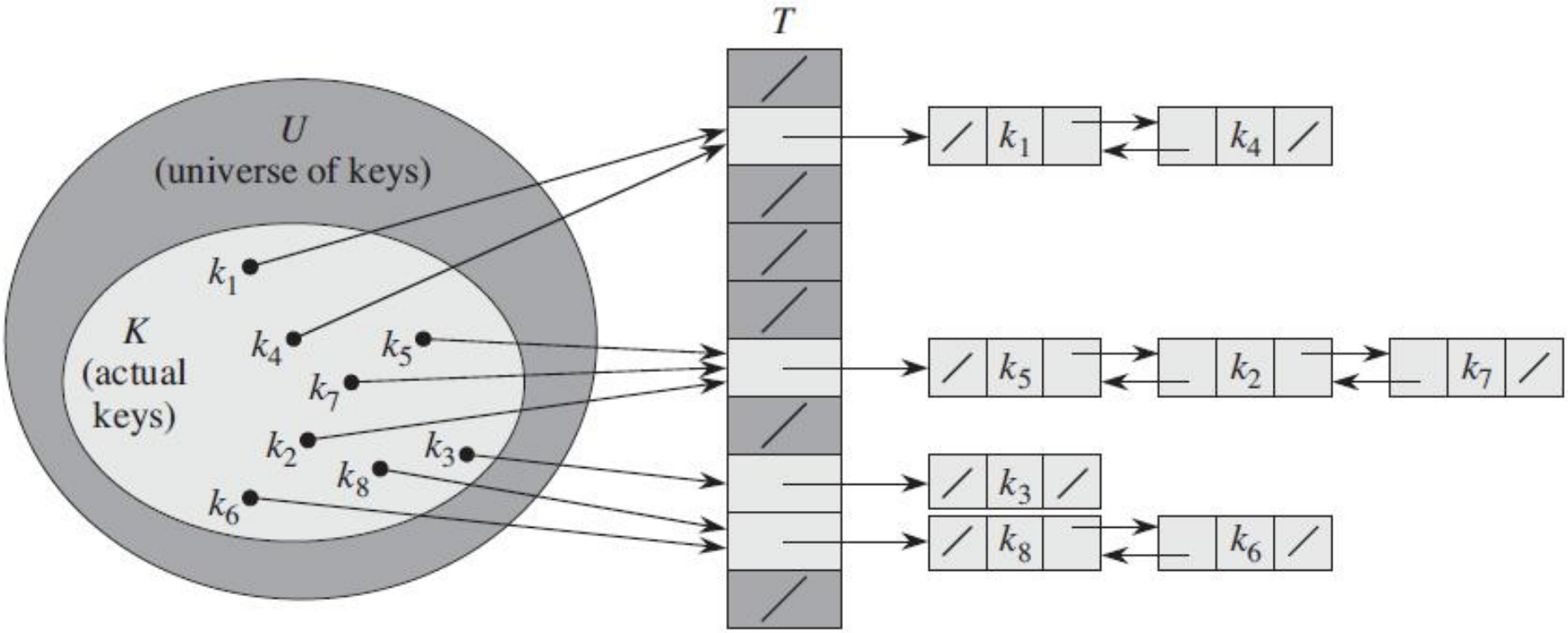
对于全域较大，但是元素却十分稀疏的情况，使用这种存储方式将浪费大量的存储空间。

提纲

4.3 散列表


T
U
(universe of keys)
K
(actual
keys)
k_1
k_4
k_5
k_2
k_3
0
$h(k_1)$
$h(k_4)$
$h(k_2) = h(k_5)$
$h(k_3)$
$m-1$

4.3 散列表–链接法（链地址法）



4.3 散列表–链接法（链地址法）

CHAINED-HASH-INSERT(T, x)

1 insert x at the head of list $T[h(x.key)]$

CHAINED-HASH-SEARCH(T, k)

1 search for an element with key k in list $T[h(k)]$

CHAINED-HASH-DELETE(T, x)

1 delete x from the list $T[h(x.key)]$

4.3 散列表–链接法（链地址法）

链接法散列的分析（算法导论）

①一次不成功查找平均时间分析

在查找不成功的情况下，我们需要遍历链表T[j]的每一个元素，而链表T[j]的长度是α，因此需要时间O(α)，加上索引到T(j)的时间O(1)，总时间为Θ (1 + α)。

② 一次成功查找平均时间分析

在查找成功的情况下，我们无法准确知道遍历到链表T[j]的何处停止，因此我们只能讨论平均情况。平均时间都为Θ(1+α)

4.3 散列表–链接法（链地址法）

链接法散列的分析（数据结构）

表 9.2　不同处理冲突的平均查找长度

处理冲突的方法	平均查找长度	
	查找成功时	查找不成功时
线性探测法	$S_{nl} \approx \frac{1}{2}\left(1+\frac{1}{1-\alpha}\right)$	$U_{nl} \approx \frac{1}{2}\left(1+\frac{1}{(1-\alpha)^2}\right)$
二次探测法与双哈希法	$S_{nr} \approx -\frac{1}{\alpha}\ln(1-\alpha)$	$U_{nr} \approx \frac{1}{1-\alpha}$
链地址法	$S_{nc} \approx 1+\frac{\alpha}{2}$	$U_{nc} \approx \alpha+e^{-\alpha}$

4.3

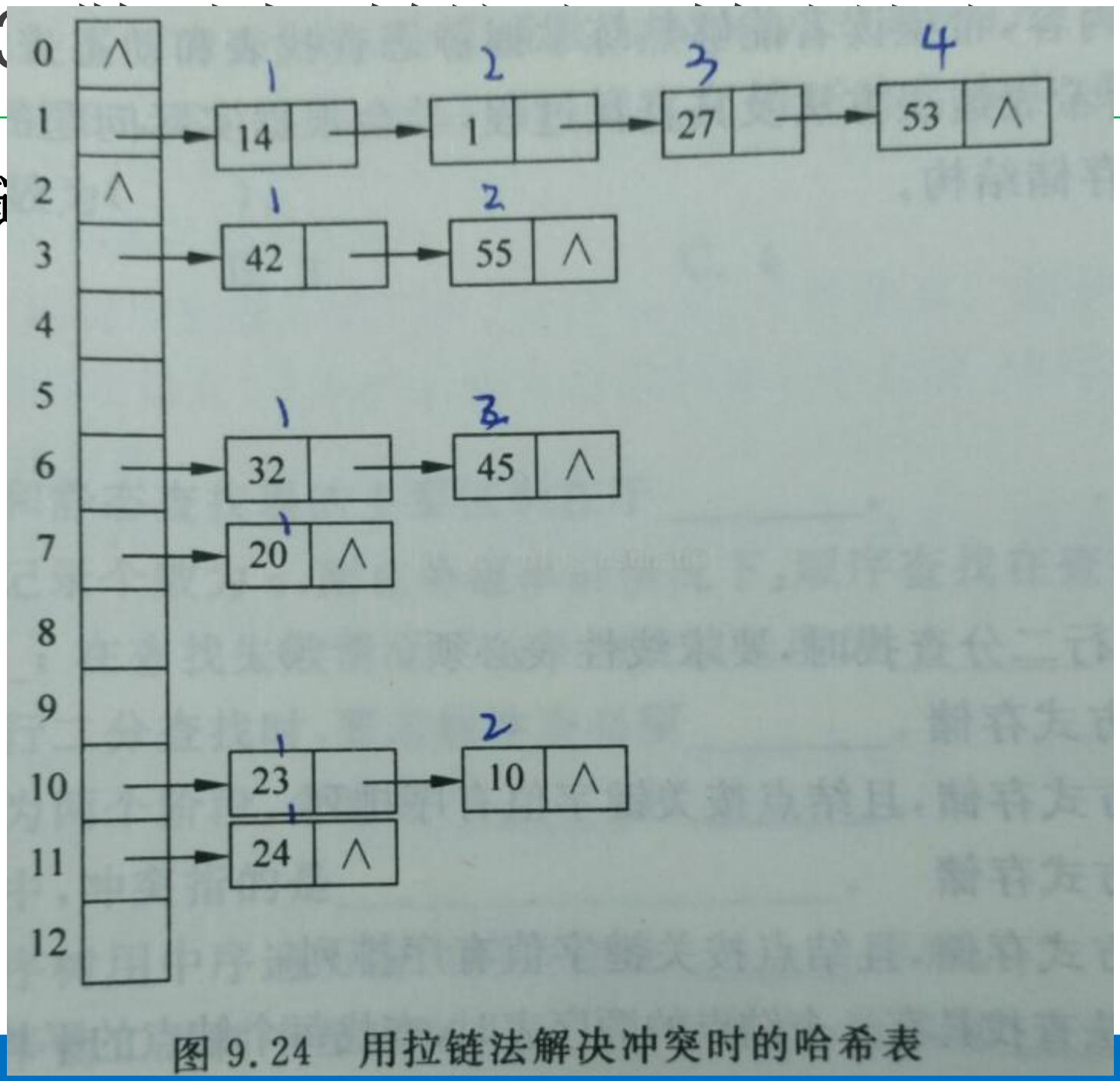


图 9.24　用拉链法解决冲突时的哈希表

提纲

4.1 概述

4.2 直接寻址

4.3 散列表

4.4 散列函数

4.5 开放寻址法

4.6 Cuckoo Hash 布谷鸟哈希

4.7 哈希的扩展应用

4.4 散列函数

The 'magic' of hash functions

PHENOMENAL COSMIC POWERS!!

惊人的、广大无边的、强势（拉丁神）

itty bitty living space

微人模式、小的居住空间（拉丁神灯）

构造散列函数的基本方法

- 基本要求

设关键字集**K**中有**n**个关键字，哈希表长为**m**，即哈希表地址集为**[0,m-1]**，则哈希函数**H**应满足：

1. 对任意$k_i \in K$，**i=1,2,…,n**，有$0 \le H(k_i) \le m-1$；

2. 对任意$k_i \in K$，$H(k_i)$取**[0,m-1]**中任一值的概率相等。

4.4.1 散列函数:除法散列法

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

A	B	C
十进制(k)	二进制(k)	h(k)(其中m=8)
20	10100	4
98	1100010	2
204	11001100	4
71	1000111	7
67	1000011	3
1234	10011010010	2

4.4.1 除法散列法的不足

Problems with division method

对于规则性的键值集合

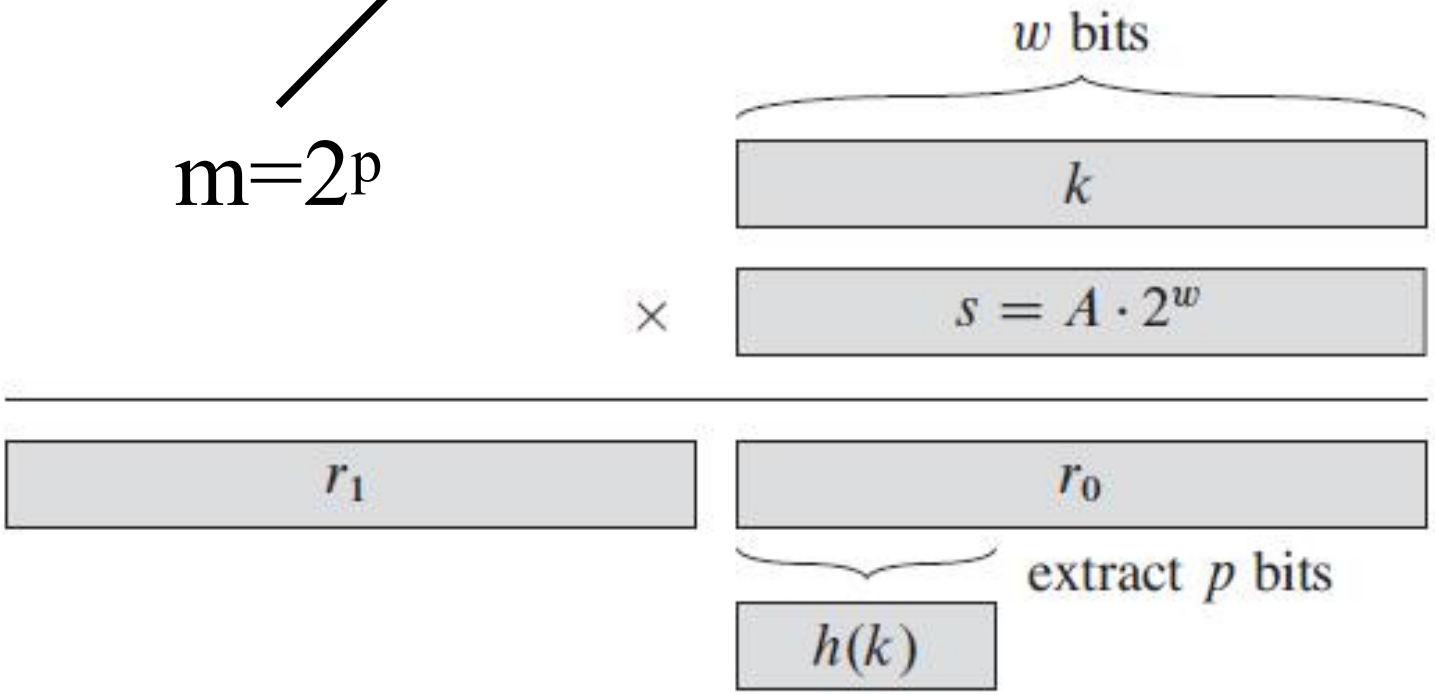
- Regularity
  - Suppose keys are x, 2x, 3x, 4x,…. 假如x与m有个公约数d
  - Suppose x and chosen m have common divisor d
  - Then only use 1/d fraction of table 那么我们将只能有效利用到表格1/d部分空间
    - x series cycles back, leaving d-1 out of d entries blank
    - E.g, m power of 2 and all keys are even, only use half
- So make m a prime number 所以m得是个质数
  - But finding a prime number is hard 但是找个质数很难
  - And now you have to divide (slow)

并且你要去相除（比乘法，位移都慢很多）

4.4.2 散列函数:乘法散列法

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

$m=2^p$



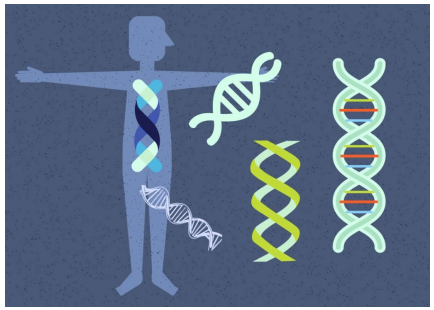
$$A \approx (\sqrt{5} - 1) / 2 = 0.618033988$$

4.4.3散列函数:散列值

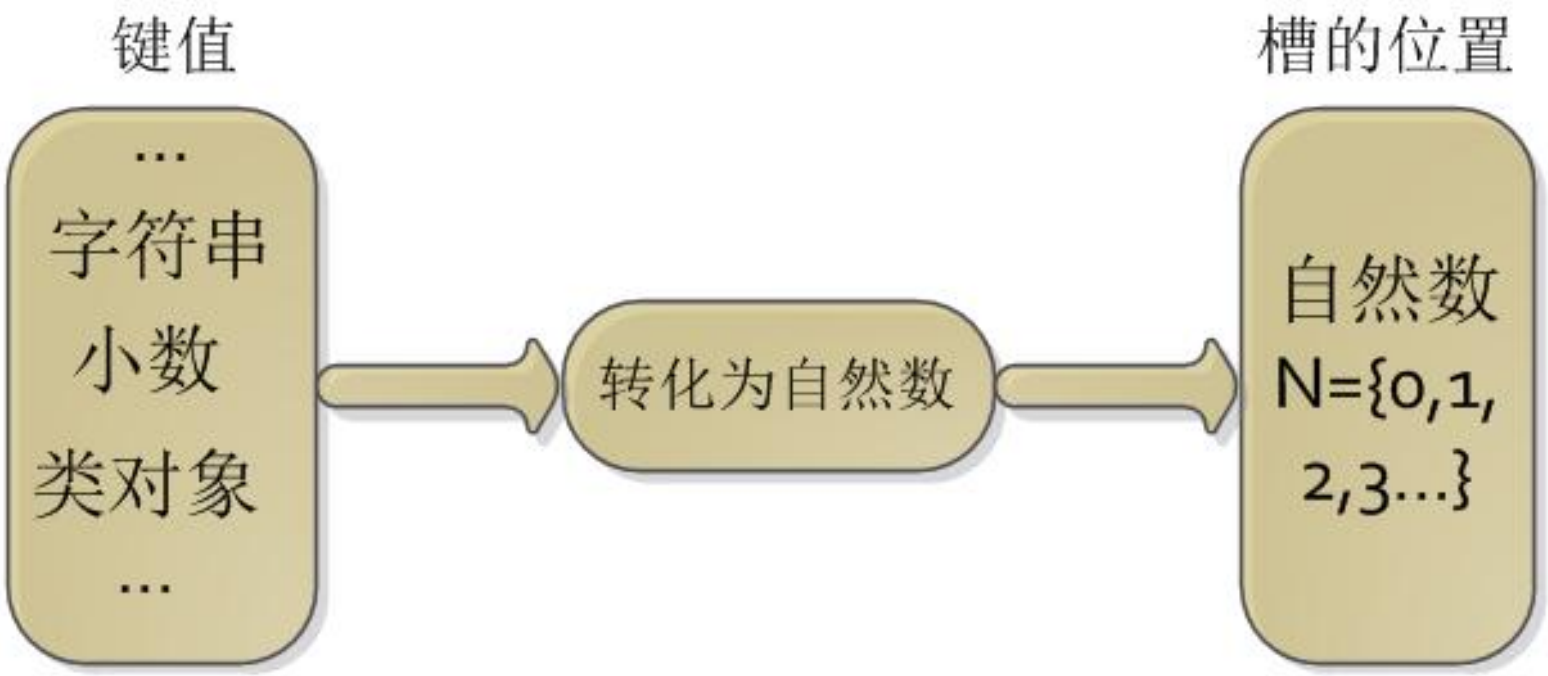
Key不是数字，是字符串呢?

Key=“abcdef”

Hash值



4.4.3散列函数:散列值



Java: hashcode
C++: ASCII码相加

Python:
Key="abcdef"
A=hash(Key)

提纲

4.1 概述

4.2 直接寻址

4.3 散列表

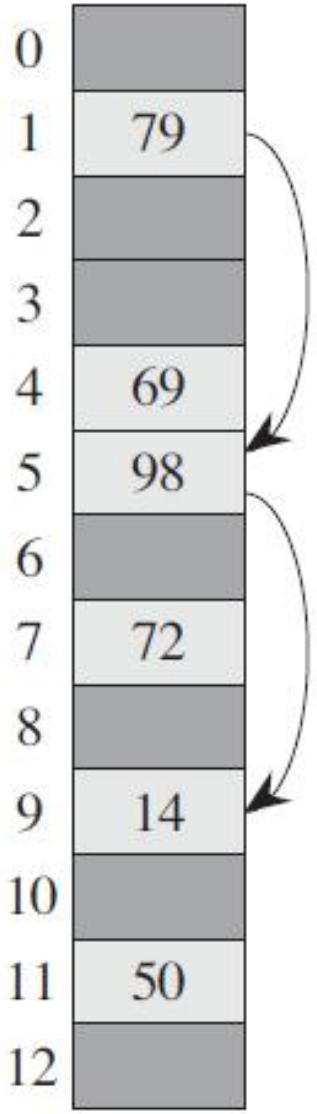
4.4 散列函数

4.5 开放寻址法

4.6 Cuckoo Hash 布谷鸟哈希

4.7 哈希的扩展应用

4.5 开放寻址法



4.5 开放寻址法

```
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 "hash table overflow"
```

4.5 开放寻址法

```
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 or i == m
8  return NIL
```

4.5 开放寻址法

```
HASH–DELETE(T,k)
    i=0
    repeat
            j = h(k,i)
            if ( T[j] == k )
                T[j] = DELETED
                return
             i = i + 1
    until T[j] == NIL or i == m
    return
```

4.5 开放寻址法

```
HASH-INSERT(T, k)
    i = 0
    repeat
        j = h(k,i)
        if ( T[j] == NIL or T[j] == DELETED )
            T[j] = k
            return j
        else i = i + 1
    until i == m
    error "hash table overflow"
```

4.5 开放寻址法

三种常用技术来计算开放寻址法中的探查序列

1.线性探查

2.二次探查

3.双重散列

4.5 开放寻址法–线性探查

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

关键词 (key)	47	7	29	11	9	84	54	20	30
散列地址 h(key)	3	7	7	0	9	7	10	9	8
冲突次数	0	0	1	0	0	3	1	3	6

关键词 (key)	47	7	29	11	9	84	54	20	30
散列地址 h(key)	3	7	7	0	9	7	10	9	8
冲突次数	0	0	1	0	0	3	1	3	6

地址 / 操作	0	1	2	3	4	5	6	7	8	9	10	11	12	说 明
插入47				47										无冲突
插入7				47				7						无冲突
插入29				47				7	29					$d_1 = 1$
插入11	11			47				7	29					无冲突
插入9	11			47				7	29	9				无冲突
插入84	11			47				7	29	9	84			$d_3 = 3$
插入54	11			47				7	29	9	84	54		$d_1 = 1$
插入20	11			47				7	29	9	84	54	20	$d_3 = 3$
插入30	11	30		47				7	29	9	84	54	20	$d_6 = 6$

注意”聚集”现象

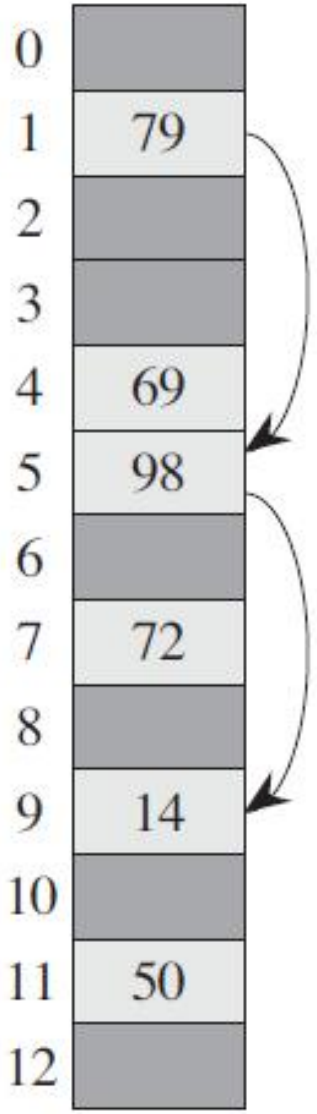
4.5 开放寻址法–二次探查

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

4.5 开放寻址法–双重散列

$$h(k, i) = (h1(k) + ih2(k)) \bmod m$$

4.5 开放寻址法



提纲

4.1 概述

4.2 直接寻址

4.3 散列表

4.4 散列函数

4.5 开放寻址法

4.6 Cuckoo Hash 布谷鸟哈希

4.7 哈希的扩展应用

4.6 Cuckoo Hash 布谷鸟哈希

定义： 一种解决hash冲突的方法，其目的是使用简单的hash 函数来提高hash table的利用率，同时保证O(1)的查询时间。基本思想是使用2个hash函数来处理碰撞，从而每个key都对应到2个位置。

4.6 Cuckoo Hash 布谷鸟哈希

操作：

1）对key值hash，生成两个hash key值，hashk1和hashk2, 如果对应的两个位置上有一个为空，那么直接把key插入即可。

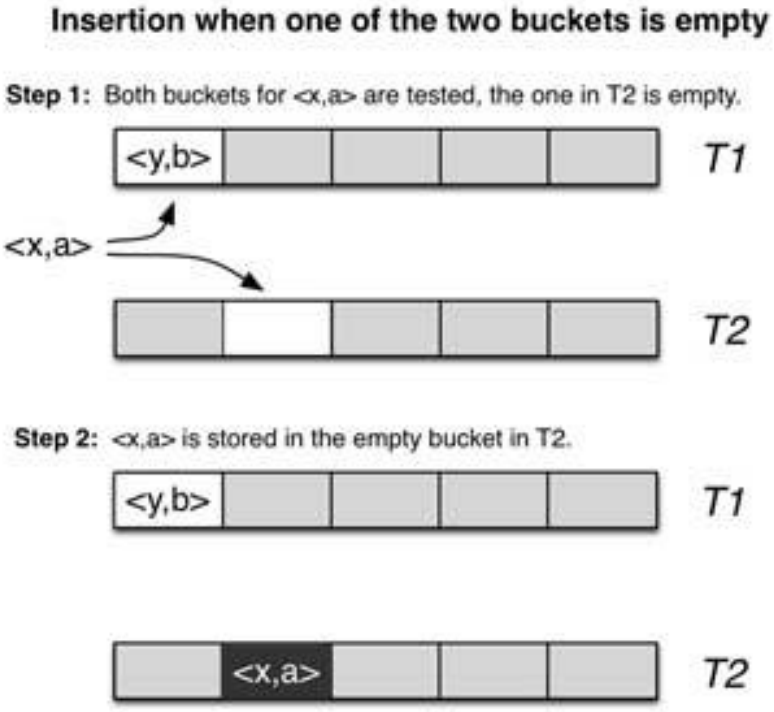
2）否则，任选一个位置，把key值插入，把已经在那个位置的key值踢出来。

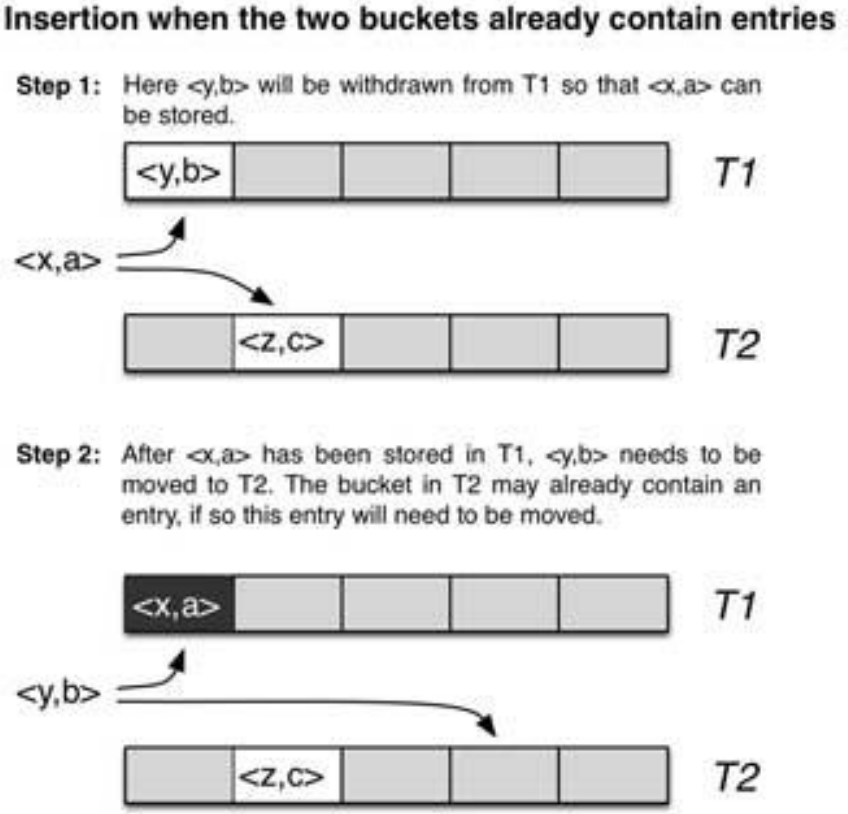
3）被踢出来的key值，需要重新插入，直到没有key被踢出为止。

4.6 Cuckoo Hash 布谷鸟哈希

衍生背景:
Cuckoo中文名叫布谷鸟，这种鸟有一种即狡猾又贪婪的习性，它不肯自己筑巢， 而是把蛋下到别的鸟巢里，而且它的幼鸟又会比别的鸟早出生，布谷幼鸟天生有一种残忍的动作，幼鸟会拼命把未出生的其它鸟蛋挤出窝巢，今后以便独享“养父 母”的食物。
借助生物学上这一典故，cuckoo hashing处理碰撞的方法，就是把原来占用位置的这个元素踢走，不过被踢出去的元素还要比鸟蛋幸运，因为它还有一个备用位置可以安置，如果备用位置上 还有人，再把它踢走，如此往复。直到被踢的次数达到一个上限，才确认哈希表已满，并执行rehash操作。

4.6 Cuckoo Hash 布谷鸟哈希





4.6 Cuckoo Hash 布谷鸟哈希

其他：

1）Cockoo hash 有两种变形。一种通过增加哈希函数进一步提高空间利用率；另一种是增加哈希表，每个哈希函数对应一个哈希表，每次选择多个张表中空余位置进行放置。三个哈希表可以达到80% 的空间利用率。

2）Cockoo hash 的过程可能因为反复踢出无限循环下去，这时候就需要进行一次循环踢出的限制，超过限制则认为需要添加新的哈希函数。

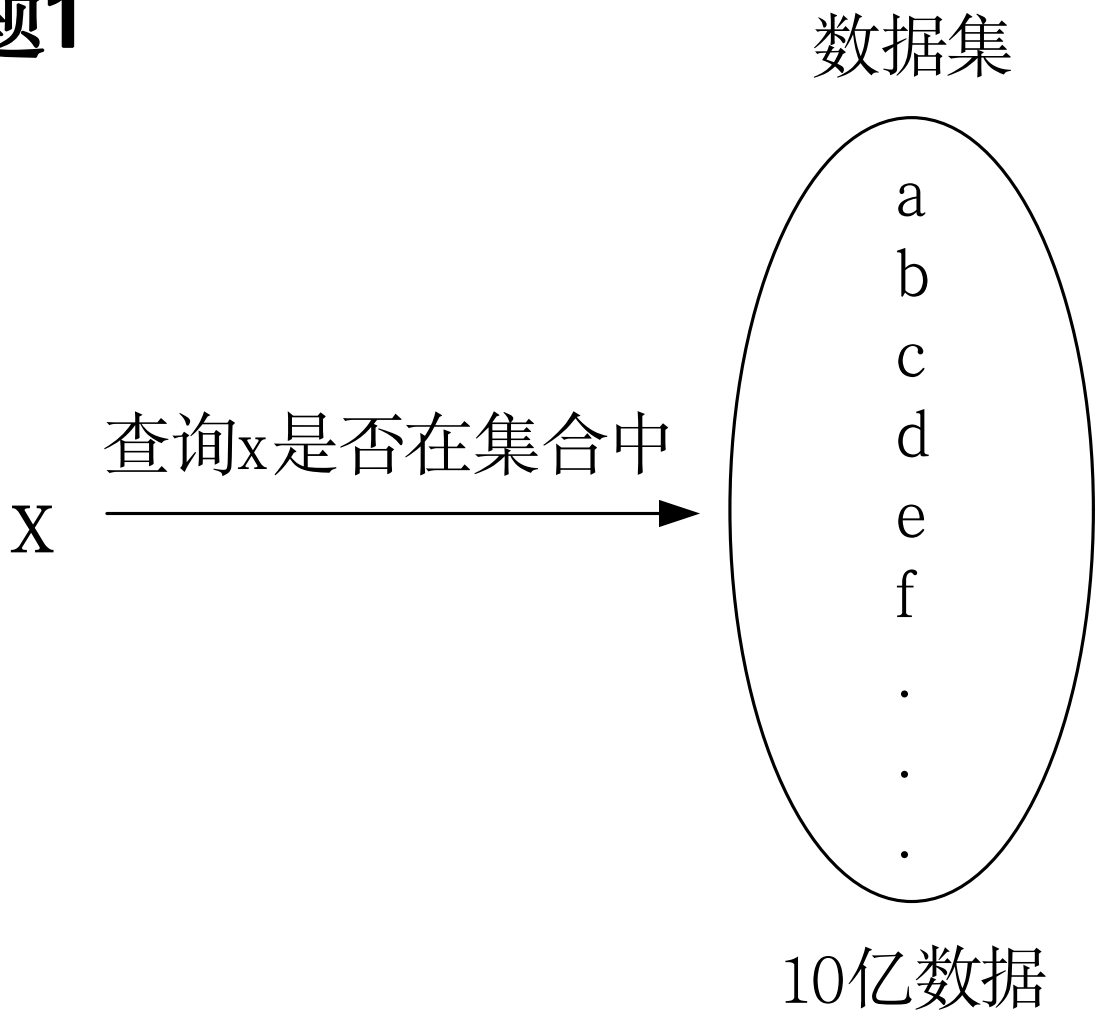
提纲

4.7 哈希的扩展应用

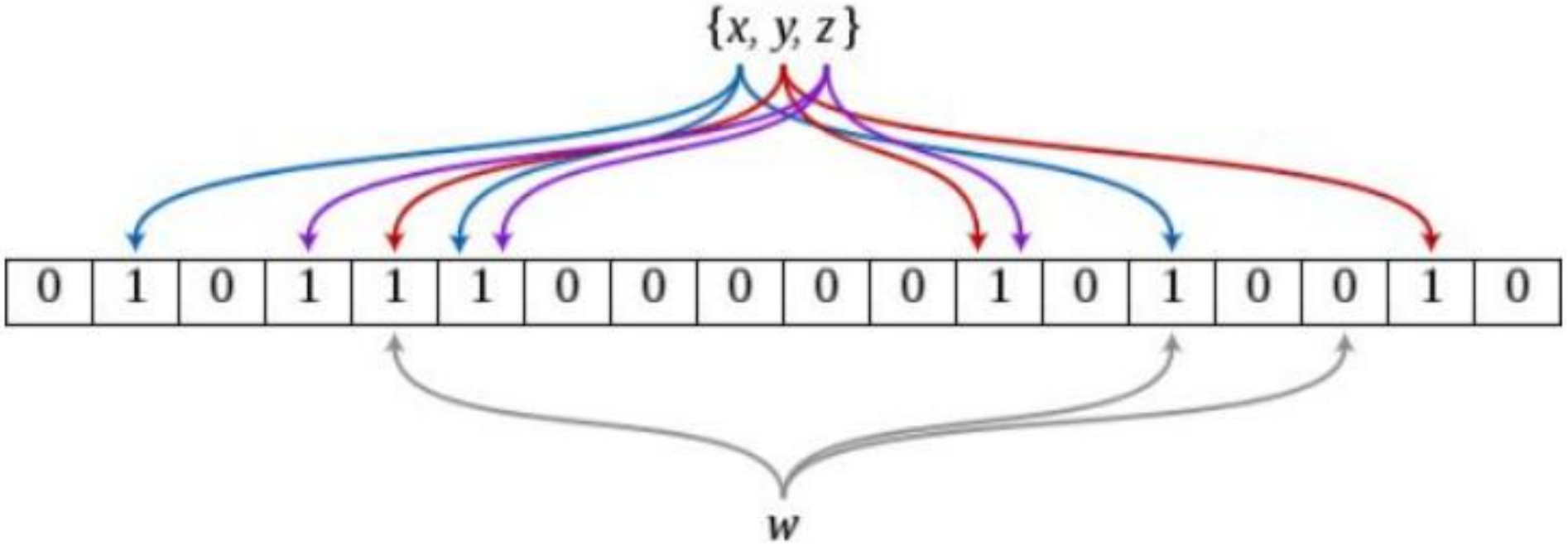
- 布鲁姆过滤器（BloomFilter）
- 最小哈希
- 区块链

4.7 布鲁姆过滤器 (bloomfilter)

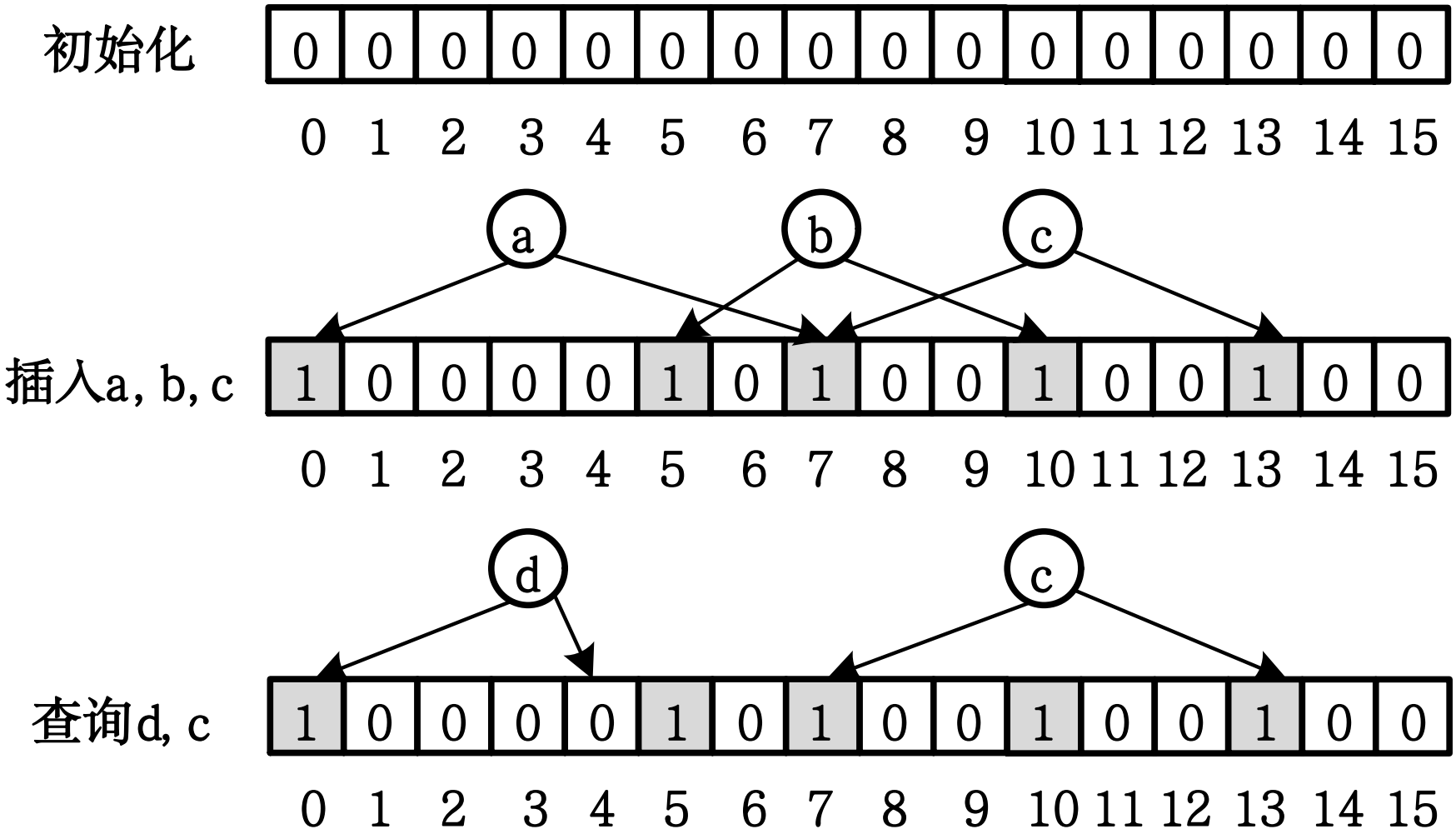
问题1



4.7 布鲁姆过滤器 (bloomfilter)



4.7 布鲁姆过滤器(bloom filter)



布鲁姆过滤器算法关键指标

- 空间复杂度
- 时间复杂度
- **误判率（假阳性）**

将不属于集合的元素误判断成属于集合中的这种假阳性误判称为一次误判。发生误判的概率称为误判率。

4.7 布鲁姆过滤器 (bloomfilter)

误判率分析（假阳性）

(1) 理论分析，假设k个哈希函数、m位bitset，n个元素之后

$$f_{\mathrm{BF}}(m,k,n)=(1-e^{-k\cdot n/m})^{k}$$

4.7 布鲁姆过滤器

误判率分析（假阳性）

(3) 最优分析

$$k = \left\lceil \ln 2 \left(m / n \right) \right\rceil$$

$$k_{\min} = (\ln 2)\left(\frac{m}{n} \right)$$

4.7 最小哈希

MinHash是一种局部敏感哈希技术
可以用来快速估算两个集合的相似度

4.7 Jaccard相似度

Jaccard相似度是用来计算集合相似性，也就是距离的一种度量标准

假如有集合A、B，那么$J(A,B)= |A \cap B| / |A \cup B|$

4.7 最小哈希

h(x): 把x映射成一个整数的哈希函数

hmin(S)： 集合S中的元素经过h(x)哈希后，具有最小哈希值的元素

那么对集合A、B，hmin(A) = hmin(B)成立的条件是A ∪ B 中具有最小哈希值的元素也在 A ∩ B中

4.7 最小哈希性质

假设h(x)是一个良好的哈希函数，它具有很好的均匀性，能够把不同元素映射成不同的整数。

所以有，Pr[hmin(A) = hmin(B)] = J(A,B)，即集合A和B的相似度为集合A、B经过hash后最小哈希值相等的概率。

4.7 区块链

假设h(x)是一个良好的哈希函数，它具有很好的均匀性，能够把不同元素映射成不同的整数。

所以有，Pr[hmin(A) = hmin(B)] = J(A,B)，即集合A和B的相似度为集合A、B经过hash后最小哈希值相等的概率。