

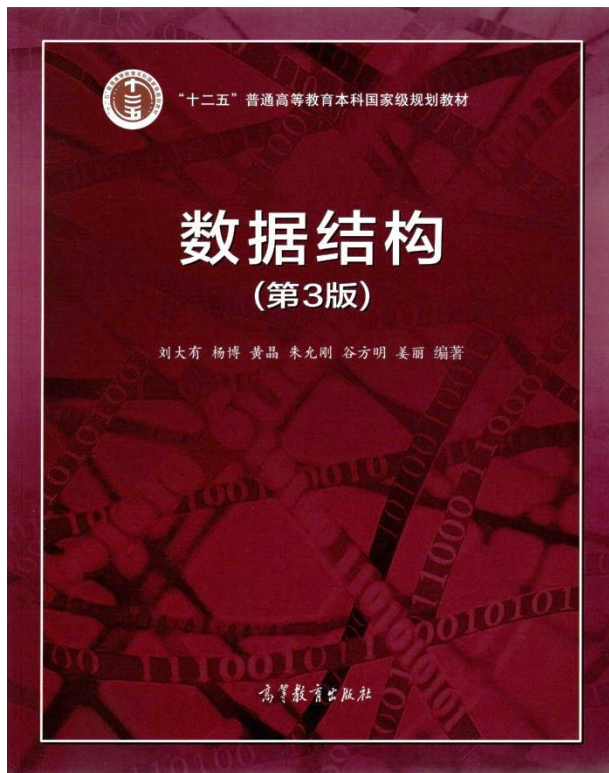


计算机学院王湘浩班
2024级




散列查找

- 散列定义
- 散列函数
- 冲突处理方法
- 性能分析与其他应用



数据之法
结构之美
算法之道



大学期间我主要在做三件事情：
一是写代码，因为我是搞技术的；
二是看书，看了很多很多书；
三是帮人修电脑，这让我交到很多朋友。

——张一鸣

南开大学学士
字节跳动创始人

动机

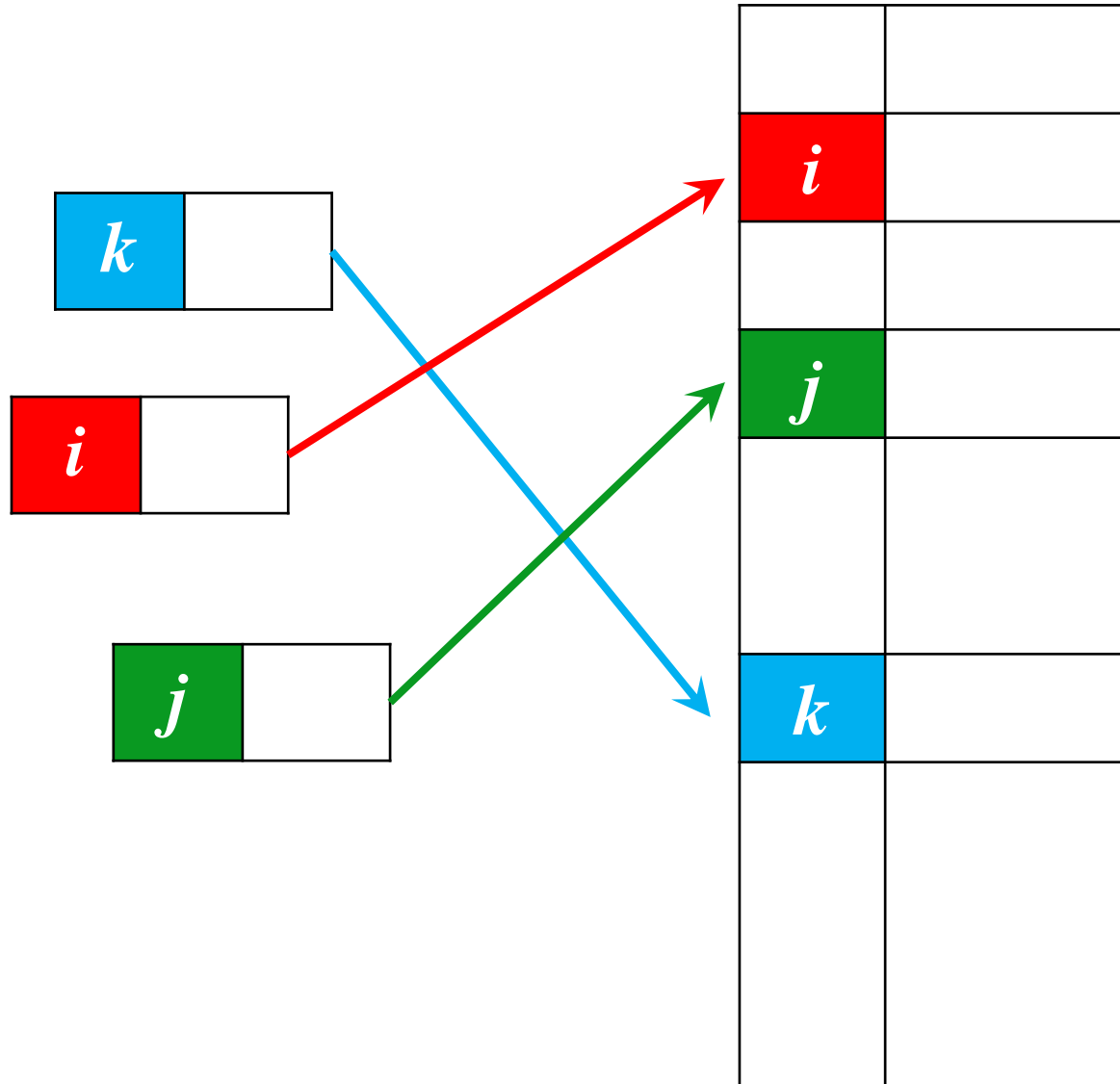
- 前面所介绍的查找方法都是基于关键词的比较，在找到想要的元素之前，需要检查若干数目的元素。
- 当数据规模 n 很大时，上述方法的时间效率仍可能使得用户无法忍受。
- **理想的情况：**根据给定关键词，直接定位到元素的存储地址，而不需要与各元素逐个比较。

散列方法提出者



Hans Peter Luhn
(1896-1964)
IBM公司研究员

散列表



```
struct Student{  
    int id;  
    char name[100];  
    char gender;  
    int age;  
    double score;  
};
```

散列 (Hash, 亦称哈希、杂凑)

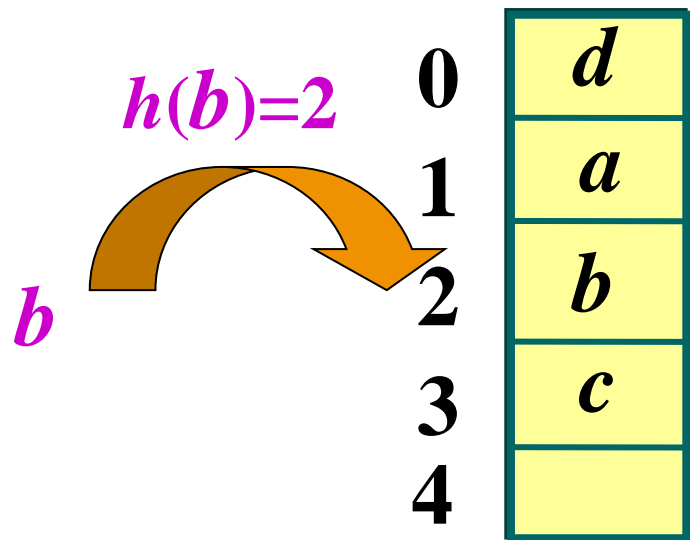
散列函数

$$h('a')=1$$

$$h('b')=2$$

$$h('c')=3$$

$$h('d')=0$$



散列表 (Hash Table)
通常是一个一维数组，散列地址即数组的下标

散列函数 h

- ✓ 自变量 K : 关键词
- ✓ 函数值 $h(K)$: 元素在散列表中的存储地址 (亦称散列地址)
- ✓ 作用: 把关键词值映射到散列地址

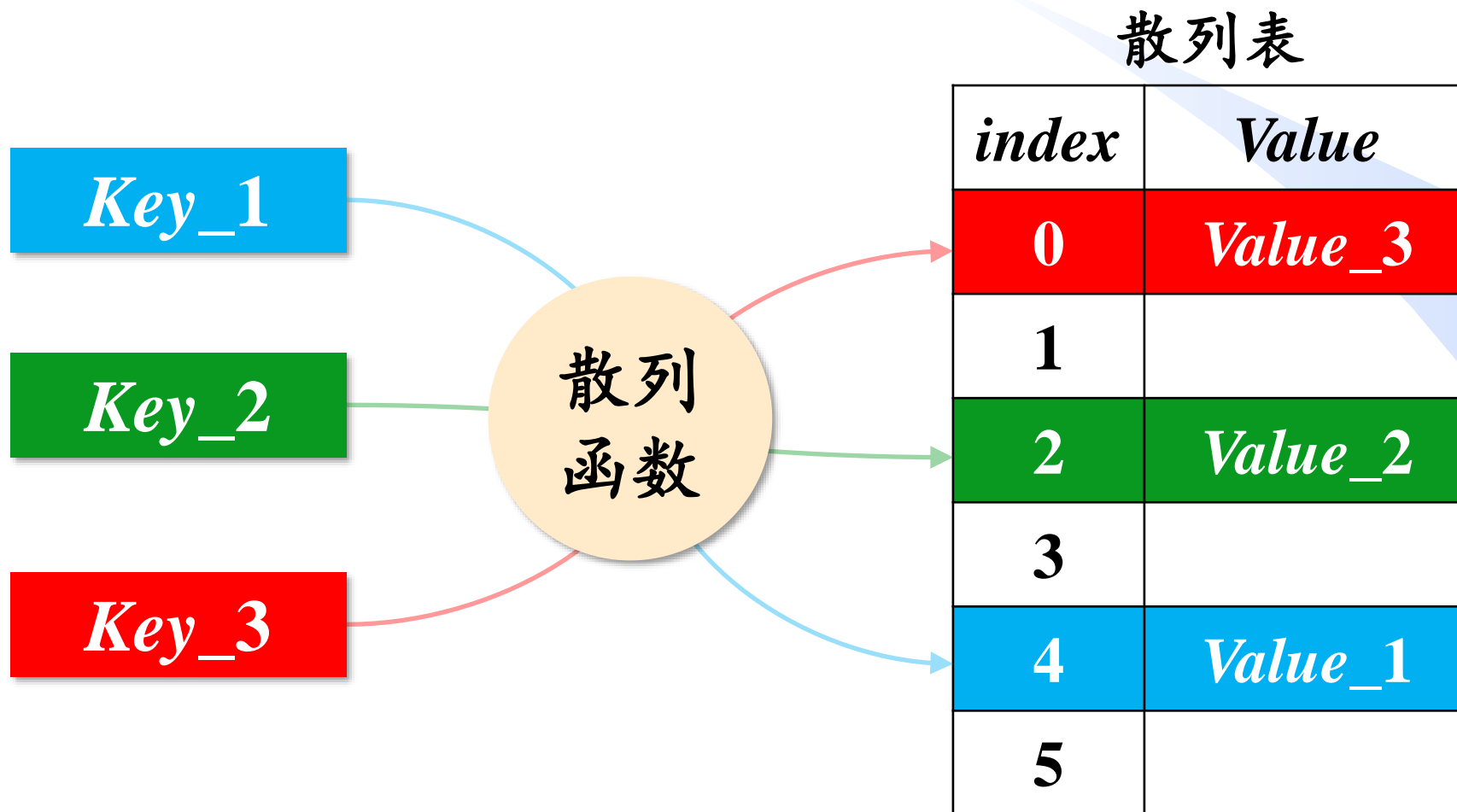
$h('e')=1$ 怎么办?

冲突: 多个不同的关键词具有相同的散列函数值, 即 $K_1 \neq K_2, h(K_1)=h(K_2)$

散列方法的核心问题

散列函数设计

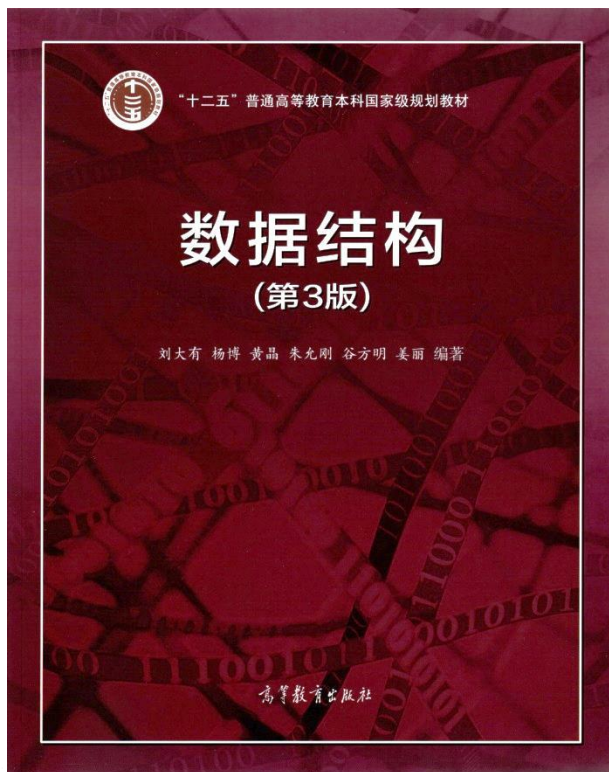
冲突处理方法





散列查找

- 散列定义
- **散列函数**
- 冲突处理方法
- 性能分析与其他应用



数据之法
结构之美
算法之道

散列函数定义及原则

- 散列函数：把关键词值映射到散列地址，通常用 h 来表示。
- $h(\text{Key}) = \text{Address of Hash Table}$
- 散列函数的选取原则
 - ✓ 便于快速计算
 - ✓ 极少出现冲突：不同关键词的散列函数值尽可能不同
 - 函数值域在散列表长范围内，即 $0 \leq h(K) < M$ ， M 为散列表长度。
 - 关键词映射到散列表各位置的概率尽量相等，使散列函数值在区间 $[0, M-1]$ 内均匀分布。

除法取余（除余法）

- $h(K) = K \% M$ 。其中 M 为散列表长度， $\%$ 为取余运算，也表示为 mod 。
- 对于理想随机的序列， M 取值无关紧要。
- 实际应用中的数据序列远非理想随机，一般来说 M 为素数时，数据对散列表的覆盖最充分、分布最均匀。
- 例如：对于序列 $a+d, a+2d, a+3d, \dots$ （公差为 d 的等差数列）
当 $\text{GCD}(d, M)=1$ 时， $(a+xd)\%M$ 均匀分布在 $[0, M-1]$ 区间内。

K	1	7	13	19	25	31	37
$h(K), M=9$	1	7	4	1	7	4	1
$h(K), M=7$	1	0	6	5	4	3	2

- M 取值的进一步建议：<https://planetmath.org/goodhashtableprimes>

MAD法 (Multiply-Add-Divide)



➤ $h(K) = (a \times K + b) \% M$

➤ M 为素数, $a > 1, b > 0, M \nmid a$

乘法散列函数

- 给定一个实数 θ , $0 < \theta < 1$, 乘法散列函数为:

$$h(K) = \lfloor M(K\theta \bmod 1) \rfloor$$

其中 $\bmod 1$ 指取小数部分, M 为散列表长。

- Knuth指出, 一般来说当 $\theta = 0.6180339887$ 或 $\theta = 0.3819660113$ (黄金分割点) 时散列函数值能均匀地分布在区间 $[0, M-1]$ 上。

平方取中法

➤ 取 K^2 的中间若干位作为 $h(K)$ 的值。

关键词	平方	取中间三位
123	1 512 9	512
1234567	1524155677489	556

抽取（数字分析）法

- 抽取关键词中的某几位，构成地址。例如

$$h(1234567)=1357$$

//取十进制的奇数位

$$h(\text{THE})=h(101000100000101)=(101)_2=5 \quad //\text{取二进制的第3位和后2位}$$

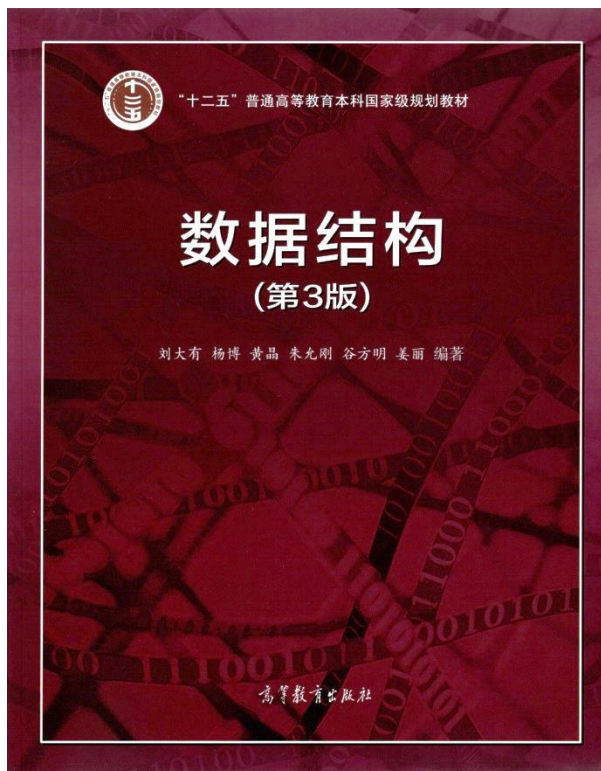
- 函数值仅依赖关键词的部分位，而不是依赖整个关键词，容易出现冲突，如 $h(\text{THE})=h(\text{FROM})=h(\text{ONE})=h(\text{WE})=(101)_2=5$ ， $h(1234567)=h(1639527)=h(1536517)=1357$





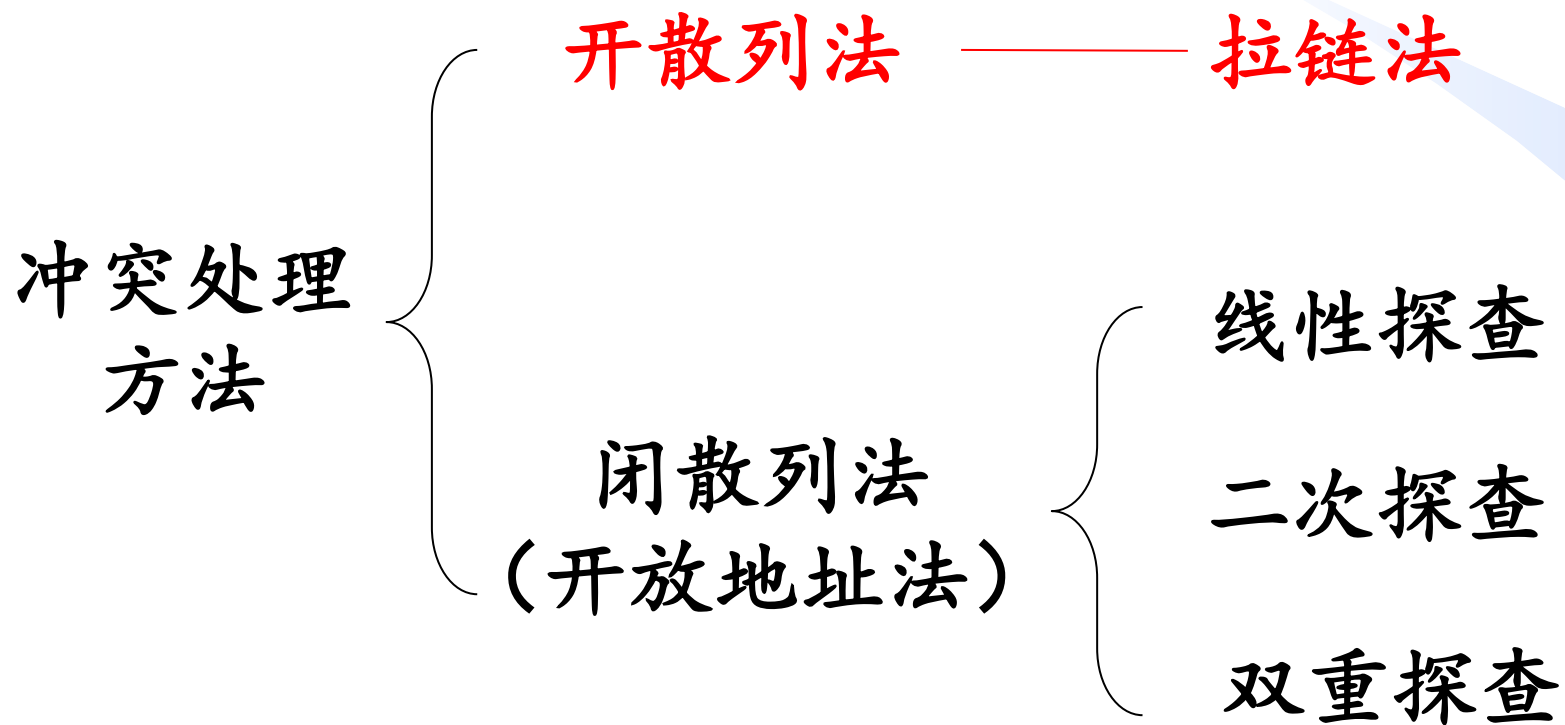
散列查找

- 散列定义
- 散列函数
- **冲突处理方法**
- 性能分析与应用

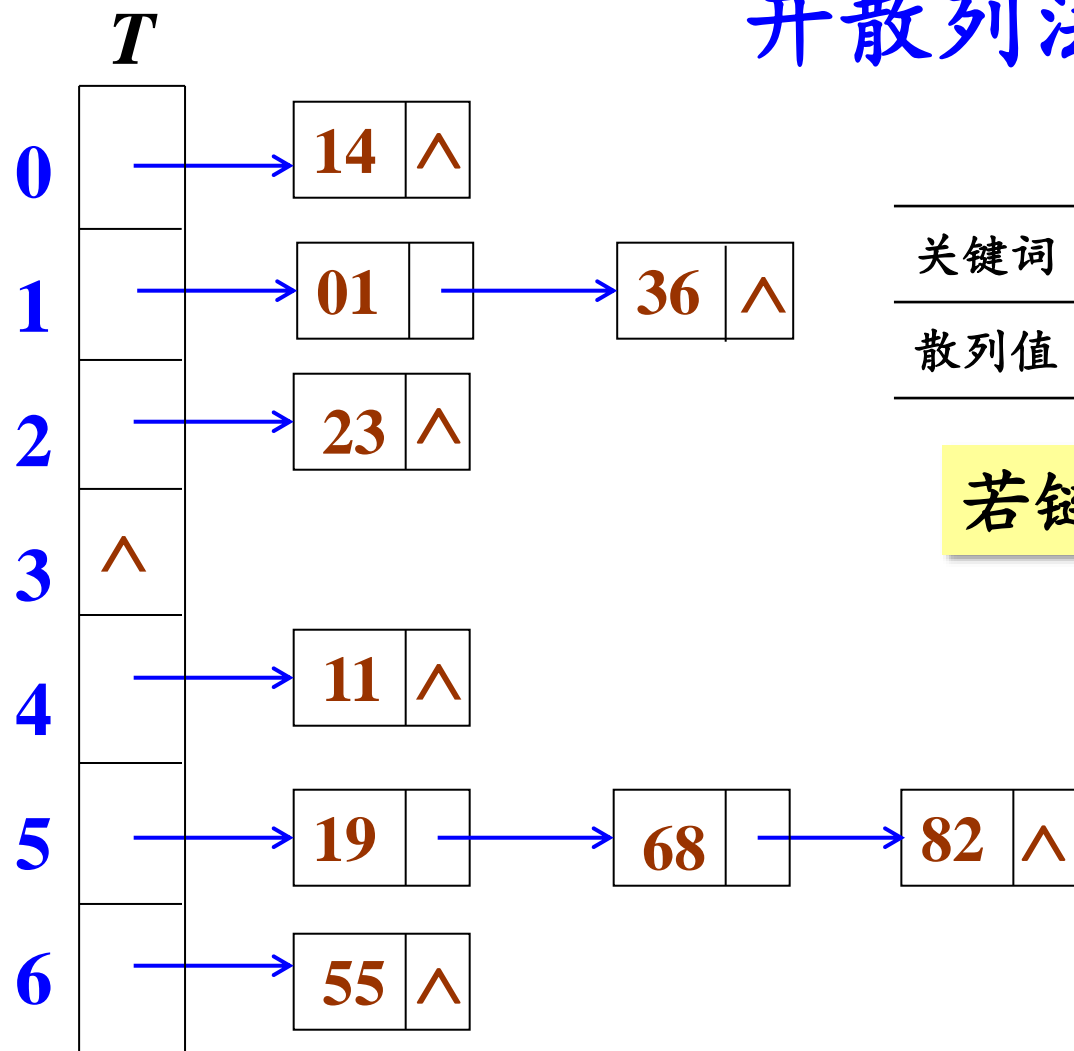


数据之美
结构之美
算法之道

冲突处理方法



开散列法——拉链法



散列函数 $h(K) = K \% 7$

关键词	19	01	23	14	55	68	11	82	36
散列值	5	1	2	0	6	5	4	5	1

若链表很长，可将其替代为跳表或查找树

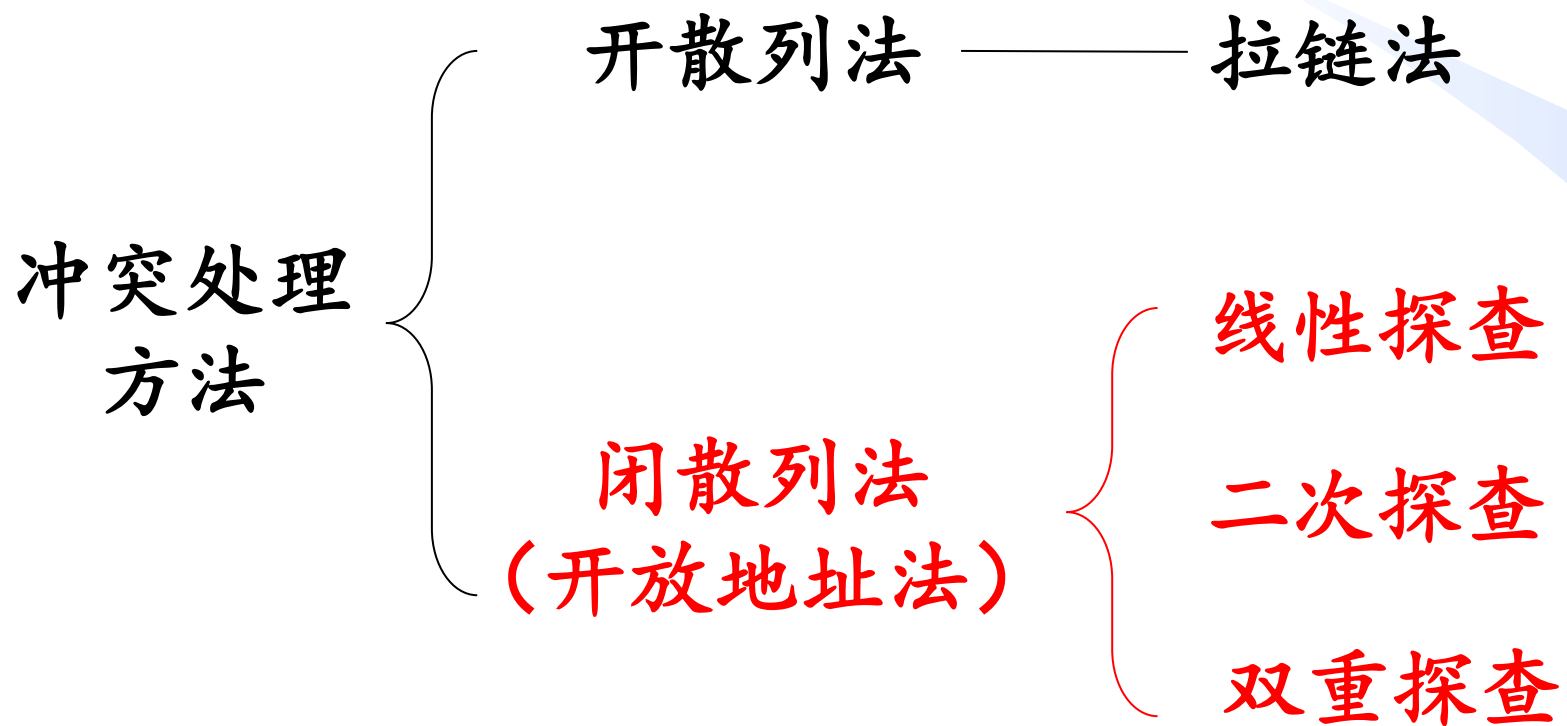
✓ C++ STL: unordered_map

✓ Java: HashMap

链表长度大于8时转化为红黑树

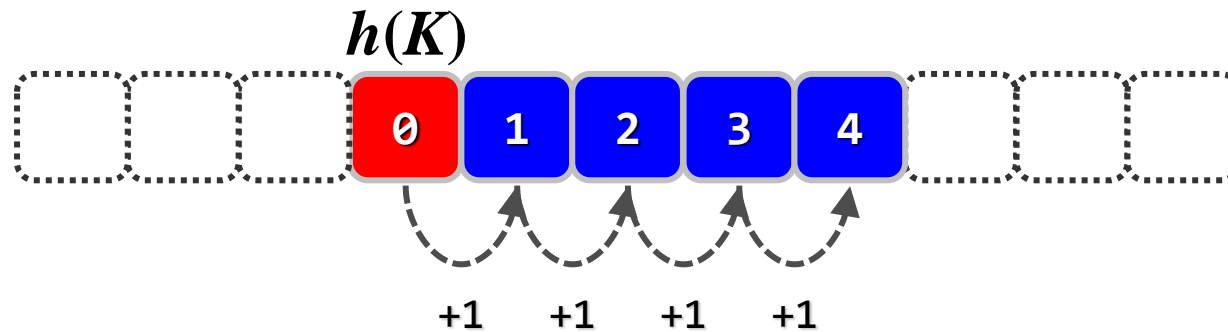
拉链就是令散列地址等于 i 的记录组成一个链表，且指针 $T[i]$ 是该单链表的头指针。

冲突处理方法



线性探查（亦称线性探测）

- 当发生冲突时，以固定的次序查找表中的记录，直到找到一个关键词为 K 的结点或者找到一个空位置。
- 其循环探查路径： $h_i = (h(K) + i) \% M$
 $h(K), h(K)+1, h(K)+2, h(K)+3, \dots, M-1, 0, 1, \dots, h(K)-1$
- 检查相邻元素，缓存命中率高。



线性探查：循环探查路径 $h(K), h(K)+1, \dots, M-1, 0, 1, \dots, h(K)-1$

散列函数： $h(K)=K \% 10$

关键词 K	20	17	2	23	32	66	6	1
散列地址 $h(K)$	0	7	2	3	2	6	6	1

0	1	2	3	4	5	6	7	8	9
20	1	2	23	32		66	17	6	

装填因子
 $\alpha=8/10=0.8$

缺点：聚集现象，即很多元素连成一片，使探查次数增加。如查找31

设散列表的长度为 M ，填入表中的元素个数是 N ，则称 $\alpha = N / M$ 为散列表的“装填因子（Load Factor）”。

- 实际应用时，常将散列表大小设计为 $\alpha=0.5 \sim 0.8$ 为宜。
- Java HashMap的 $\alpha=0.75$ ，超过此值将自动进行表扩容。
- C++ STL unordered map的 α 默认值1.0，可人为修改。

二次探查

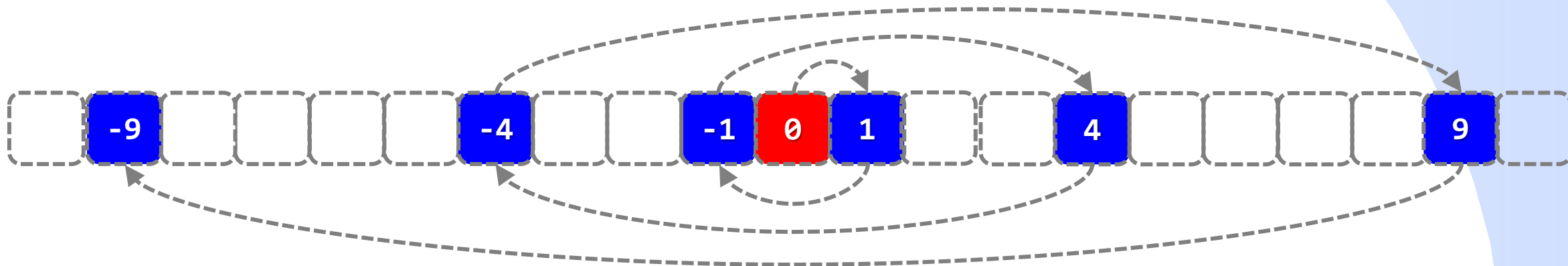
探测序列: $h_i = (h(K) \pm i^2) \% M$, 其中 $i=1, 2, \dots, (M-1)/2$, M 为散列表长

其具体循环探查路径:

$h(K), h(K)+1^2, h(K)-1^2, h(K)+2^2, h(K)-2^2, h(K)+3^2, h(K)-3^2, \dots$

以上各值均在 $\% M$ 意义下

特点: 一旦冲突, 能更快的跳离“是非之地”, 避免聚集



二次探查

散列函数: $h(K)=K \% 11$

关键词 K	19	1	23	14	55	68	11	82	36
散列值 $h(K)$	8	1	1	3	0	2	0	5	3

0	1	2	3	4	5	6	7	8	9	10
55	1	23	14	36	82	68		19		11

其具体循环探查路径:

$h(K), h(K)+1^2, h(K)-1^2, h(K)+2^2, h(K)-2^2, h(K)+3^2, h(K)-3^2, \dots$

以上各值均在 $\% M$ 意义下

双重探查

➤ 从 $h(K)$ 开始，寻找空地址时，所前进的步长不是固定的，而与 K 有关，即用 $\delta(K)$ 代替线性探查的前进步长1 ($1 \leq \delta(K) < M$)。

➤ 循环探查路径：

$$h(K), h(K) + \delta(K), h(K) + 2\delta(K), \dots$$

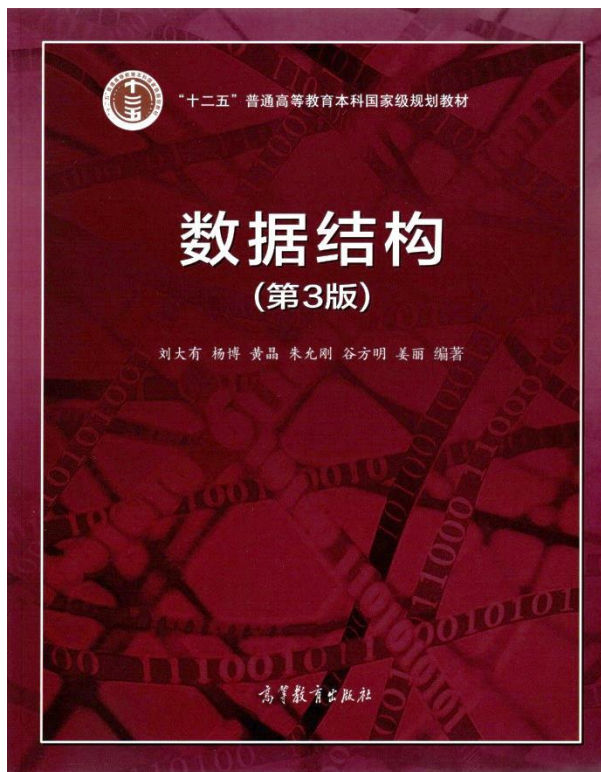
以上各值均在 $\% M$ 意义下

➤ 为了确保表中的每一个地址都能探查到，要求 $\delta(K)$ 和 M 互质，因此 M 应选作素数。



散列查找

- 散列定义
- 散列函数
- 冲突处理方法
- **性能分析与其他应用**



数据之法
结构之美
算法之道

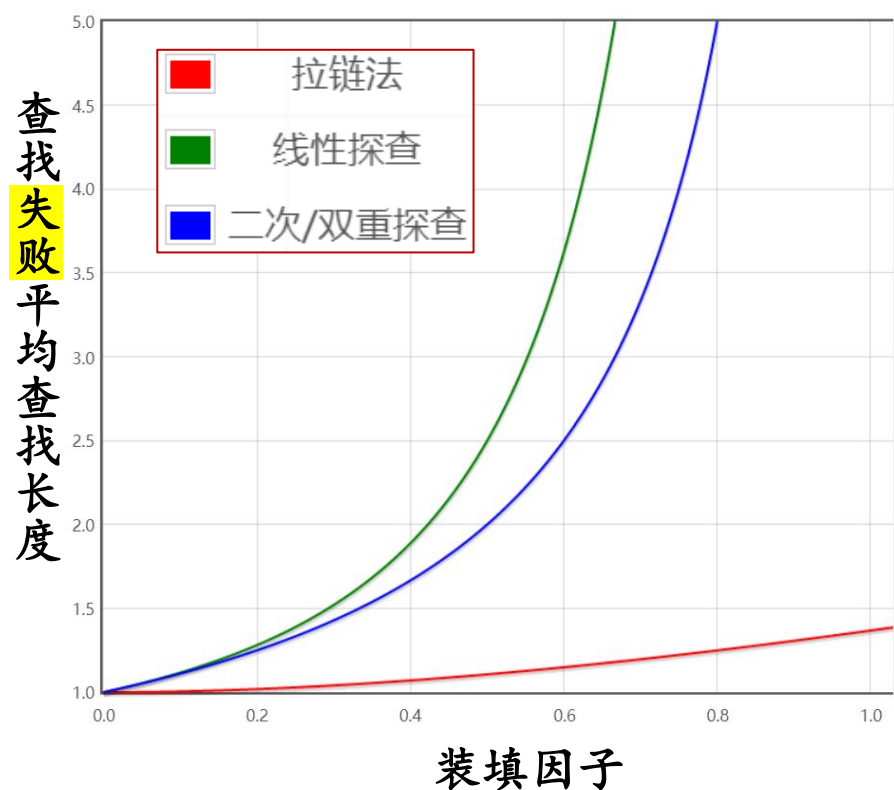
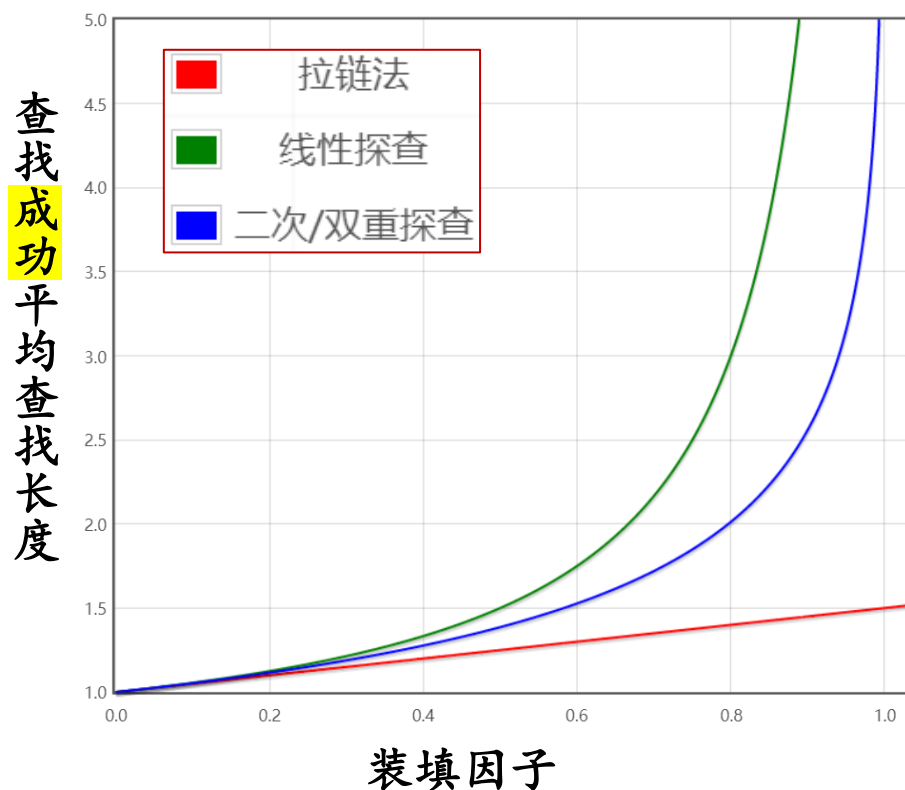
散列表的性能分析

解决冲突的策略	平均查找长度	
	查找成功	查找不成功
拉链法	$1 + \frac{\alpha}{2}$	$\alpha + e^{-\alpha}$
线性探查	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$
二次探查 双重探查	$\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$	$\frac{1}{1-\alpha}$

当装填因子 $\alpha = 0.5$ 时

- 对线性探查法，每次成功的查找操作平均需要1.5次探查，每次不成功的查找和插入平均需要2.5次探查。
- 对二次探查和双重探查法，每次成功的查找操作平均需要1.39次探查，每次不成功的查找和插入平均需要2次探查。

散列表的性能分析

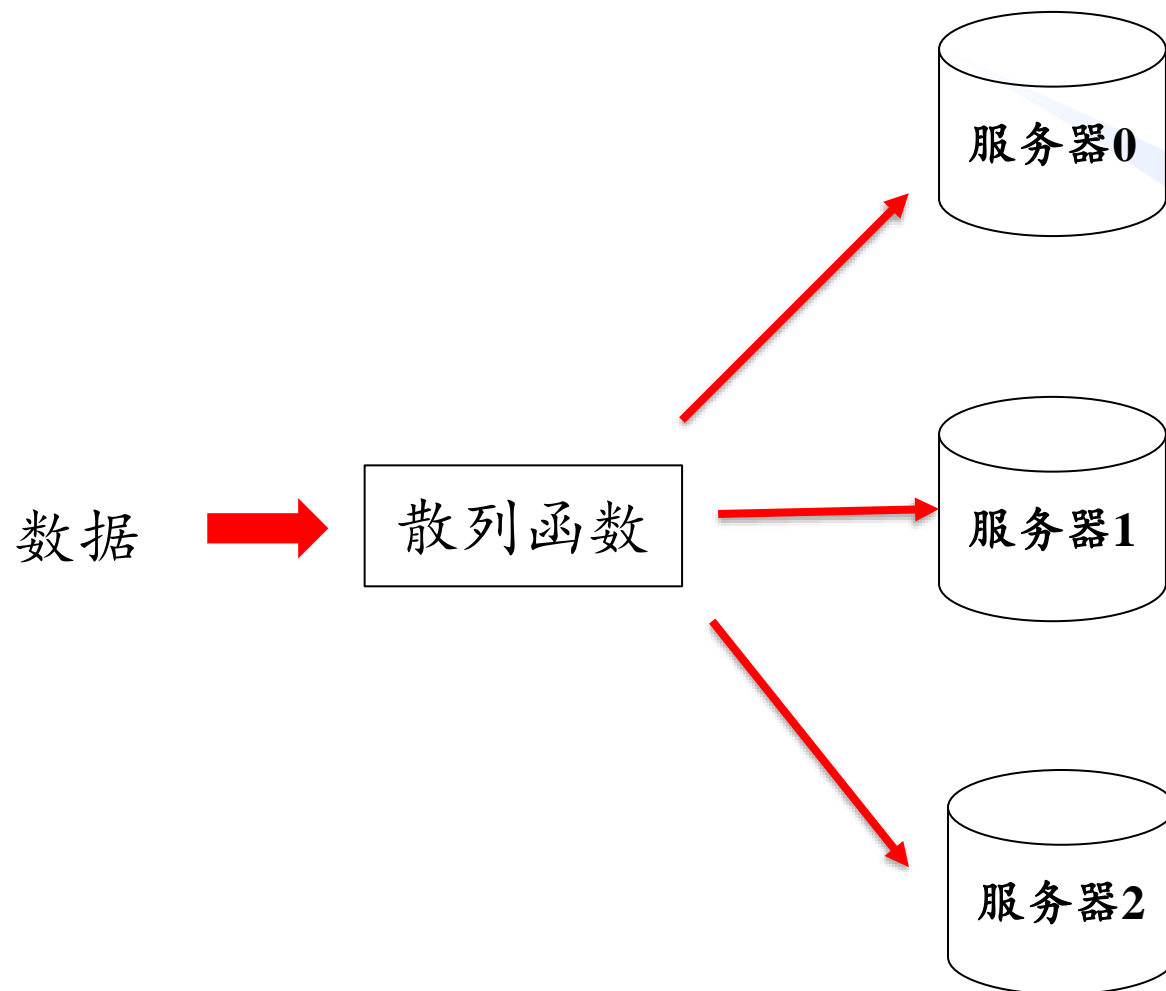


- 装填因子越大，平均探查次数越多。
- 装填因子超过0.5时，散列表的性能急剧下降。
- 拉链法效率最高，实际系统中使用的散列大多采用拉链法，而且该方法易于实现，不会产生聚集现象，删除也方便。
- 线性探查法的数据访问具有很好的空间局部性，缓存命中率高

散列表 vs 线性表查找和树形查找

- 前面介绍的查找算法时间为 $O(\log n)$ ，而散列查找的时间理论上为 $O(1)$ ，为什么有如此高效的查找方法还不放弃低效率的查找方法？
- 平均情况下，散列表时间更快。
- 但 $O(1)$ 只是散列表理想情况下的性能，实际应用中时间性能随数据量、数据分布情况的变化而变化，最坏情况下时间可达 $O(n)$ 。
- 散列表占空间大。需要控制装载因子（保证散列表中有一定的空闲单元），是以牺牲空间来换取时间。
- 在散列表中，查找失败后，仅能知道所找的关键词 K 不在表中。而以比较为基础的查找方法还能得到更多额外信息，如小于等于 K 的最大关键词和/或大于等于 K 的最小关键词，这在很多应用中是重要的。
- 查找树能更好的维护数据的“序”信息，如在BST中找最小元素/第 k 小元素只需 $O(\log n)$ 时间，而在散列表中则需要 $O(n)$ 时间。

散列表的其他应用——分布式存储



散列表的其他应用——信息安全

➤ 安全访问（密码散列化后存入数据库）



散列表的其他应用——数字指纹

➤ 数字指纹（网盘秒传，MD5、SHA1）

文件



散列函数



921988ba001dc8e14a3b108f3fa6cb6d



Ronald Rivest

1992年设计MD5散列函数

图灵奖获得者

美国科学院、工程院院士

麻省理工学院教授

虽然我不愿看到MD5倒下，但人们必须推崇真理。

——Ronald Rivest



王小云

2004年首次破解MD5

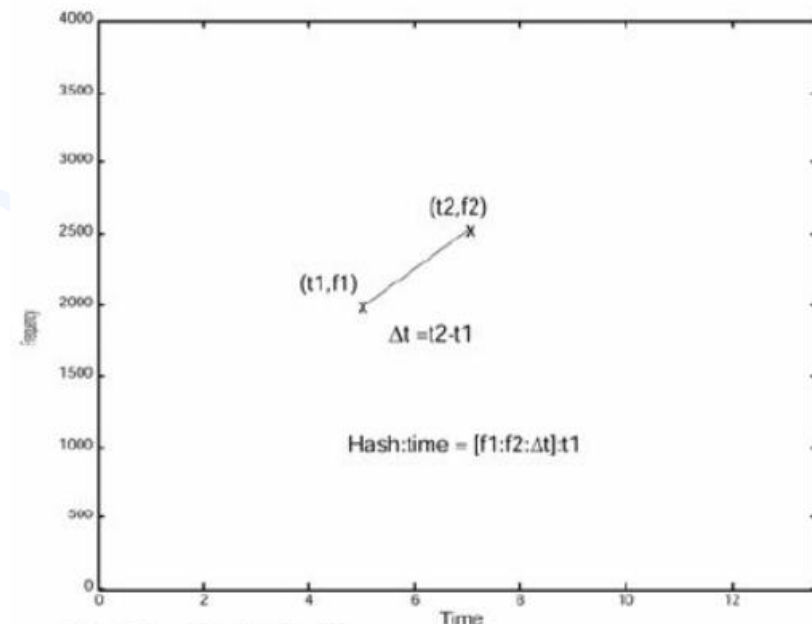
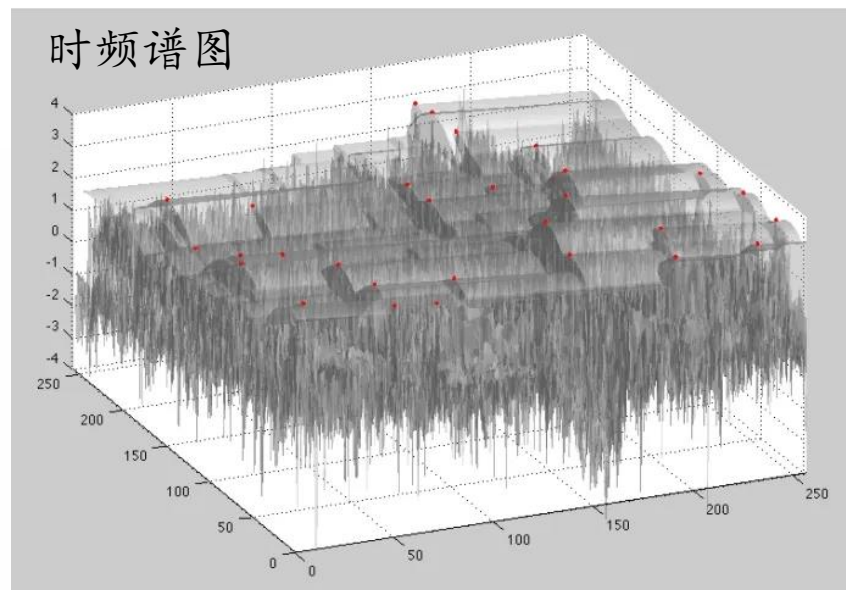
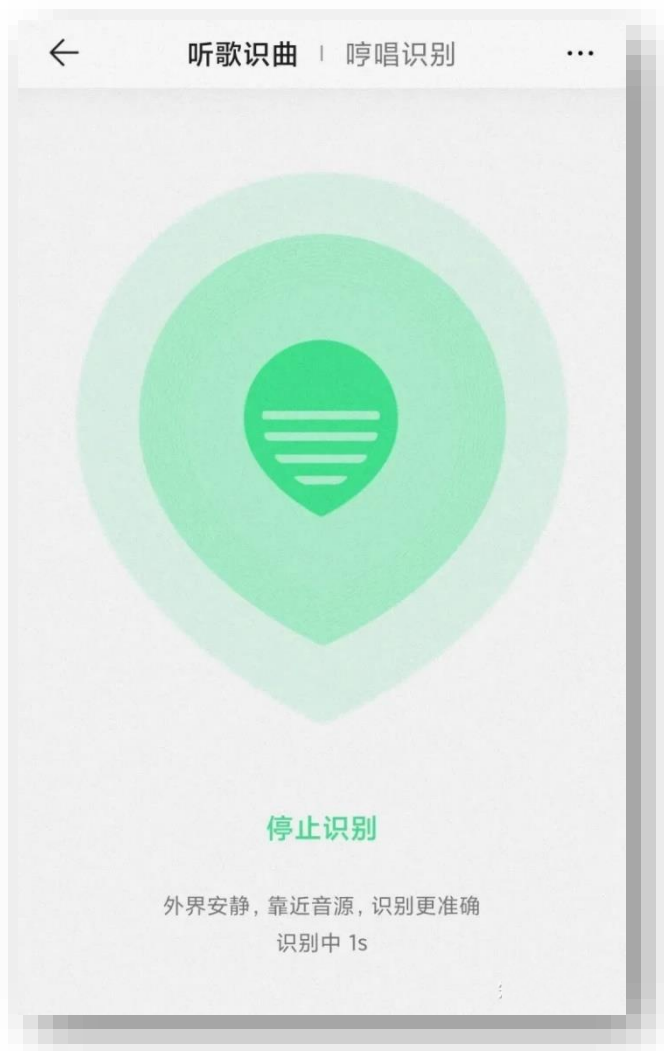
中国科学院院士

清华大学教授

山东大学教授

散列表的其他应用——听歌识曲

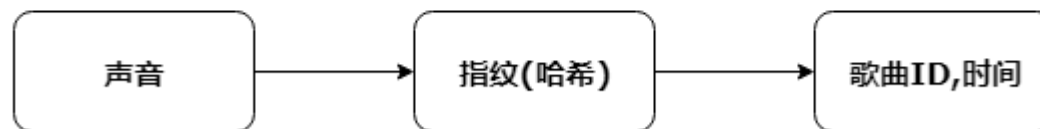
➤ 音频指纹：



1. 很尖的小号声
2. 音高是F#到E
3. 四分音符

8796513218

"Miles Davis - So What"
1分55秒



QQ音乐听歌识曲技术分享：基于Landmark的音频指纹系统

[1] <https://zhuanlan.zhihu.com/p/81994831>

[2] <https://zhuanlan.zhihu.com/p/82299663>

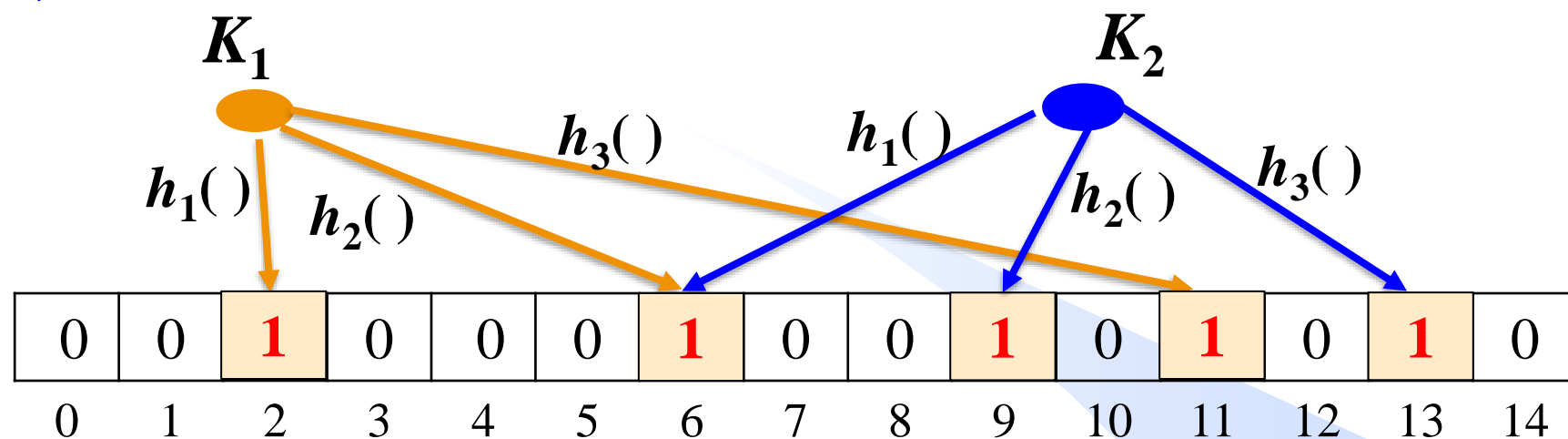
散列表的其他应用——浏览器恶意URL识别 布隆过滤器 (Bloom Filter)

D

将恶意网站的URL存入散列表，当用户准备访问某网址时，在哈希表里查找该网址？

缺点：恶意URL很多，散列表占较大内存。

希望：散列表无需存储关键词或数据元素。



- 查找成功：哈希值对应位都为1。查找失败：有1位为0。
- 存在误判：若查找结果是某个元素在表中，可能误判。若查找结果是某个元素不在表中，不会误判。
- 不能删除：因为某位可能多个关键词共用。
- Chrome浏览器：当用户访问某网址时，在本地Bloom过滤器查找该网址，若不在表中（不存在误判）则网址安全。若在表中（可能误判），则连接远程服务器做进一步检查。

课下思考

编写程序实现一个散列表。【某年腾讯校园招聘笔试最后一道大题】