

Lecture 28: 散列查找

28.1 散列表基础概念

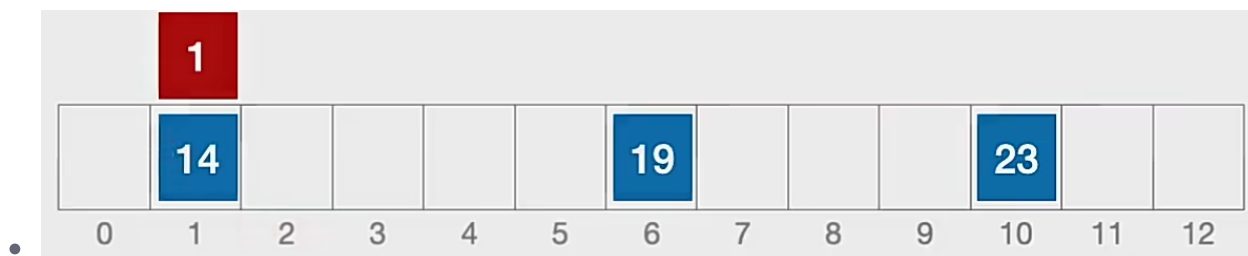
概念：基本概念

- 概念：
 - 散列表 (哈希表, Hash Table);
 - 是一种数据结构;
- 特点：
 - 可以根据数据元素的关键字计算出它在散列表中的存储地址
- 散列函数：
 - 又称哈希函数
 - $\text{Addr} = H(\text{key})$: 建立了 “关键字” $\rightarrow$ “存储地址” 的映射关系;
- 性能：
 - 理想情况下, 在散列表中查找一个元素的时间复杂度为 $O(1)$
- 举例：
 - 查找元素 19: 根据散列函数计算出元素的存储地址 = $19\%13 = 6$, 检查位置 #6 $\rightarrow$ 查找成功;

	14					19				23		
0	1	2	3	4	5	6	7	8	9	10	11	12

哈希冲突

- 概念：
 - 在散列表中插入个数据元素时, 需要根据关键字的值确定其存储地址, 若该地址 **已经存储了其他元素**, 则称这种情况为“冲突 (碰撞);
- 同义词：
 - 若不同的关键字通过散列函数映射到同一个存储地址, 则称它们为“同义词”;
- 改进方法：
 - 1. **构造更适合的散列函数**, 让各个关键字尽可能地映射到不同的存储位置, 从而减少“冲突”;
 - 但无论这个函数再精妙, 冲突的情况总是不可避免地;
 - 2. 对冲突进行处理;
 - (1) 拉链法: 把所有地同义词用链表连起来;
 - (2) 开放定址法: 当两个关键字冲突时, 为冲突地关键字重新确定一个空闲位置;
- 图示：
 - 对于散列函数 $H(\text{key}) = \text{key}\%13$ 来说, 1 和 14 是“**同义词**”



散列查找的效率因素

- 装填因子：

$$\text{装填因子} \alpha = \frac{\text{表中记录数 } n}{\text{散列表长度 } m}$$

- 装填因子越大，发生冲突的概率越大，反之越小；

28.2 散列函数地构造

核心目标：散列函数要尽可能地减少“冲突”；

- 其他目标：
 - 定义域必须涵盖所有可能出现的关键字；
 - 比如： $H(key) = \sqrt{key} \% 13$ 函数不支持关键字为负值；
 - 散列函数地值域不能超过散列表的地址范围；
 - 比如： $H(key) = key \% 15$ 函数可能被非法的映射到 13、14；
 - 尽可能减少冲突。散列函数计算出来的地址应尽可能均匀分布在整个地址空间；
 - 比如： $H(key) = (key \times 13) \% 13$ 函数的全部关键字都会被映射到地址 0；
 - 散列函数应该尽量简单，能够快速计算出任意一个关键字所对因的散列地址；
 - 比如： $H(key) = (key!) \% 13$ 函数再关键字值很大时，计算耗时大；

28.2.1 除留余数法

概念

- 核心：
 - $H(key) = key \% p$
- 概念：
 - 散列表表长为 m ，取一个不大于 m 但最接近或等于 m 的质数 p ；
- 补充：
 - 质数又称素数，指除了 1 和此整数自身外，不能被其他自然数整除的数；
- 适用场景：
 - 较为通用，只要关键字是整数即可；
- 原因：
 - 对质数取余，可以分布更均匀，从而减少冲突；
- 举例：
 - 散列表表长 $m=13$ 。13 是质数，因此可令散列函数 $H(key)=key \% 13$

- 散列表表长 $m=15$ 。不大于 15 且最接近 15 的质数是 13，因此可令散列函数 $H(key)=key\%13$

28.2.2 直接定址法

概念

- 核心：
 - $H(key)=key$ 或 $H(key)=a*key+b$
- 概念：
 - 在 $H(key)=a*key+b$ 中， a 和 b 是常数；
 - 这种方法计算最简单，且不会产生冲突；
 - 若关键字分布不连续，空位较多则会造成存储空间的浪费；
- 适用场景：
 - 关键字分布基本连续；
- 举例：
 - 用散列表存储某班级的 30 个学生信息，班内学生学号为 (1120112176~1120112205)

0	1	2	3	4	5	...	...	...	26	27	28	29
...176	...177	...178	...179	...180	...181	...	...	...	...202	...203	...204	...205

28.2.3 数字分析法

概念

- 核心：
 - 选取数码分布较为均匀的若干位作为散列地址；
- 概念：
 - 设关键字是 r 进制数 (如十进制数)，而 r 个数码在各位上出现的频率不一定相同，可能在某些位上分布均匀一些，每种数码出现的机会均等；
 - 而在某些位上分布不均匀，只有某几种数码经常出现，此时可选取数码分布较为均匀的若干位作为散列地址；
- 适用场景：
 - 关键字集合已知，且关键字的某几个数码位分布均匀；
- 举例：
 - 要求将手机用户的信息存入长度为 10000 的散列表，以“手机号码”作为关键字设计散列函数；

138XXXX2875
138XXXX1682
138XXXX9125
.....

199XXXX1684
199XXXX1236

0	1	2	3	...	...	9998	9999

28.2.4 平方取中法

概念

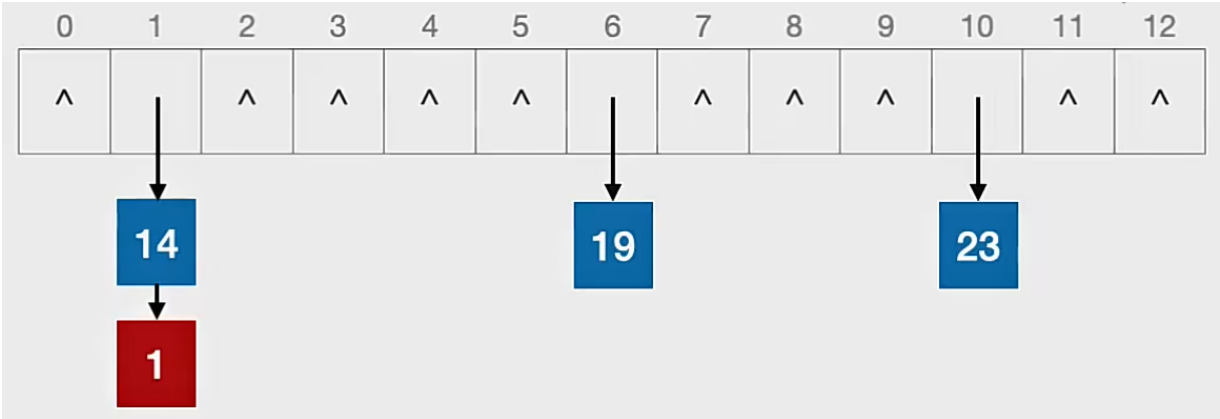
- 核心：
 - 取**关键字的平方值的中间几位**作为散列地址；
- 概念：
 - 具体取多少位要视实际情况而定。这种方法得到的散列地址与关键字的每位都有关系，因此使得**散列地址分布比较均匀**；
 - 关键字的各个数码位都分布不均匀，中间几位的值受每个数码位影响；
- 适用场景：
 - 关键字的**每位取值**都不够均匀；
- 举例：
 - 某汽车品牌，以 (底盘号、发动机号、外观号) 作为一款汽车型号的标识。该品牌有 100 种车型，欲将这这些车型信息存入长度为 100 的散列表，如何设计散列函数？

车型号	平方	取中
605030	→	366061300900
345220	→	119176848400
395225	→	156202800625
605435	→	366551539225

28.3 拉链法

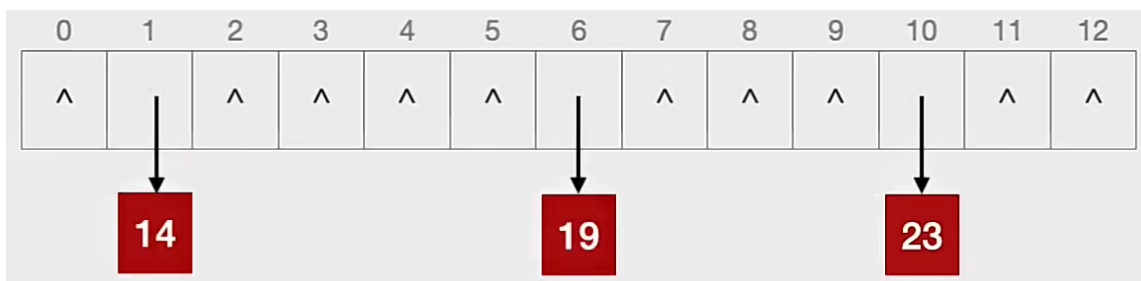
概念

- 核心：
 - 把所有“同义词”存储在一个链表中；
- 举例：
 - 某散列表的长度为 13，散列函数 $H(key)=key\%13$ 。依次将数据元素 19、14、23、1 插入散列表；
- 图示：

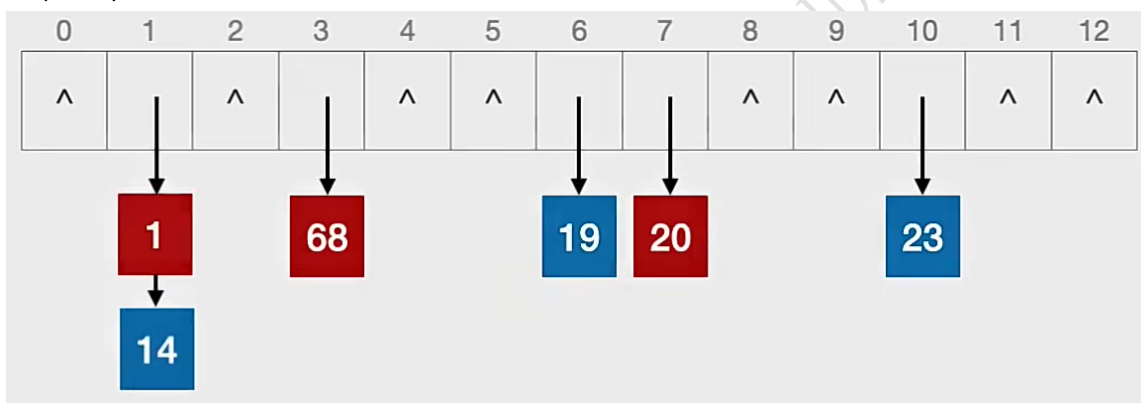


插入

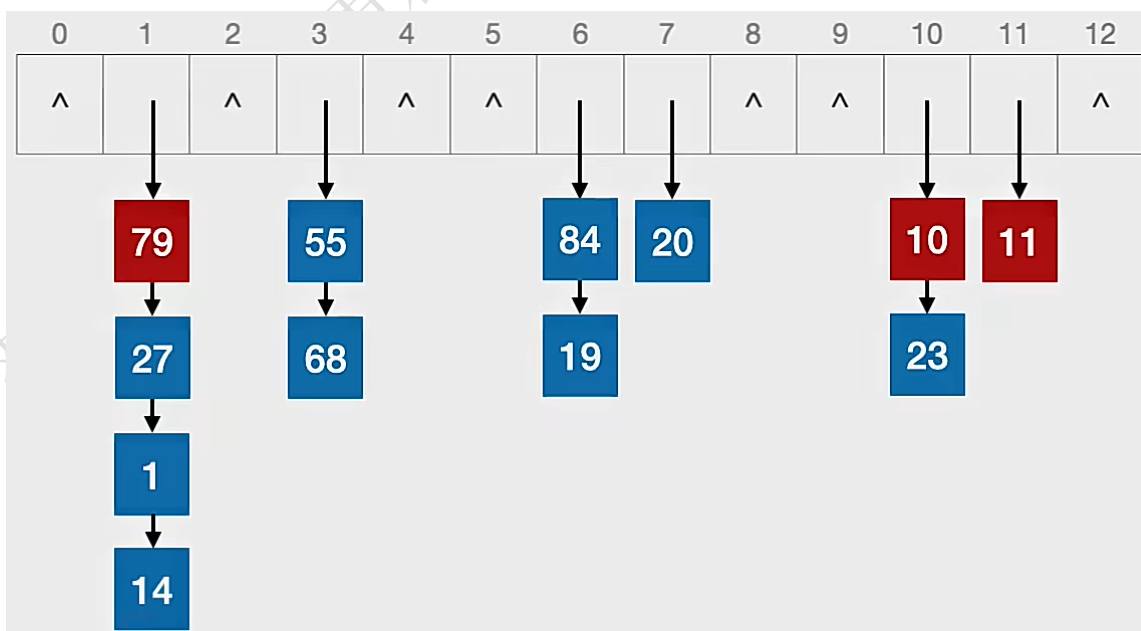
- 方法：
 - 步骤一：
 - 结合散列函数计算新元素的散列地址；
 - 步骤二：
 - 将新元素插入散列地址对应的链表 (可用头插法，也可用尾插法)；
- 举例：头插法
 - 插入前三个元素：



- 插入 1、68、20



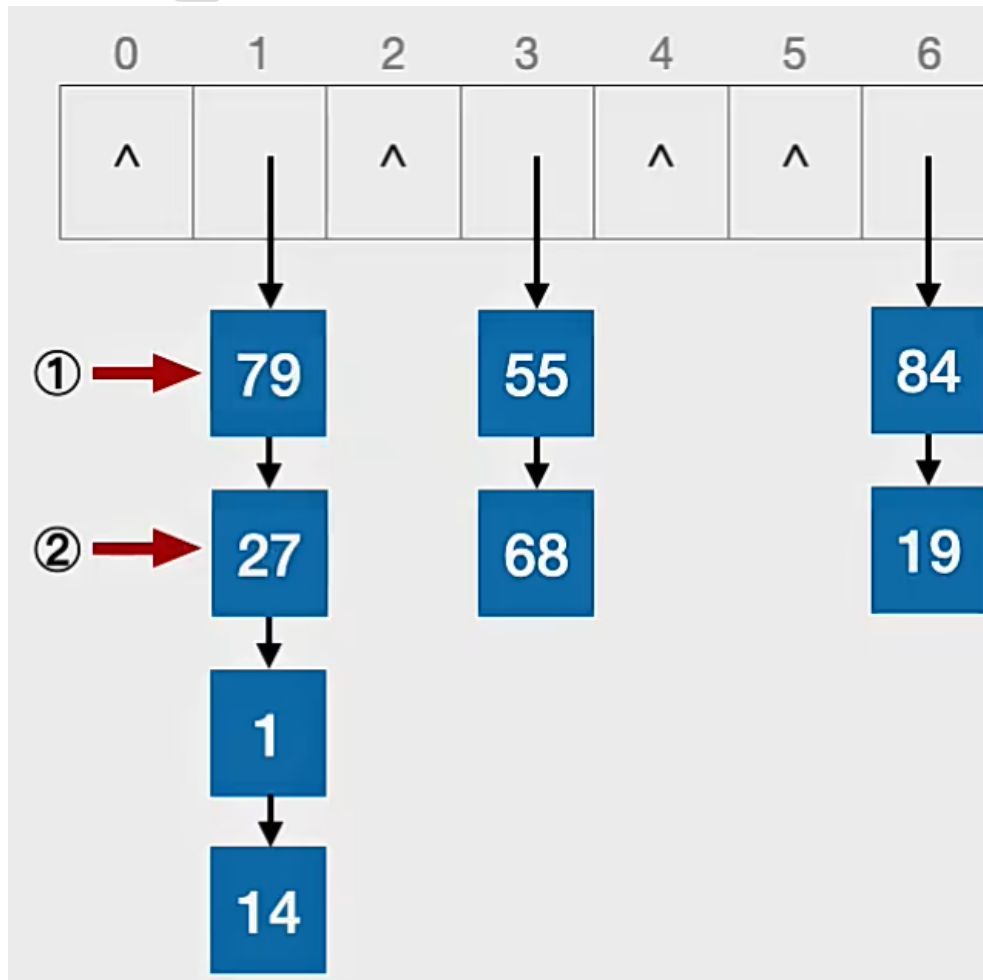
- 其他：



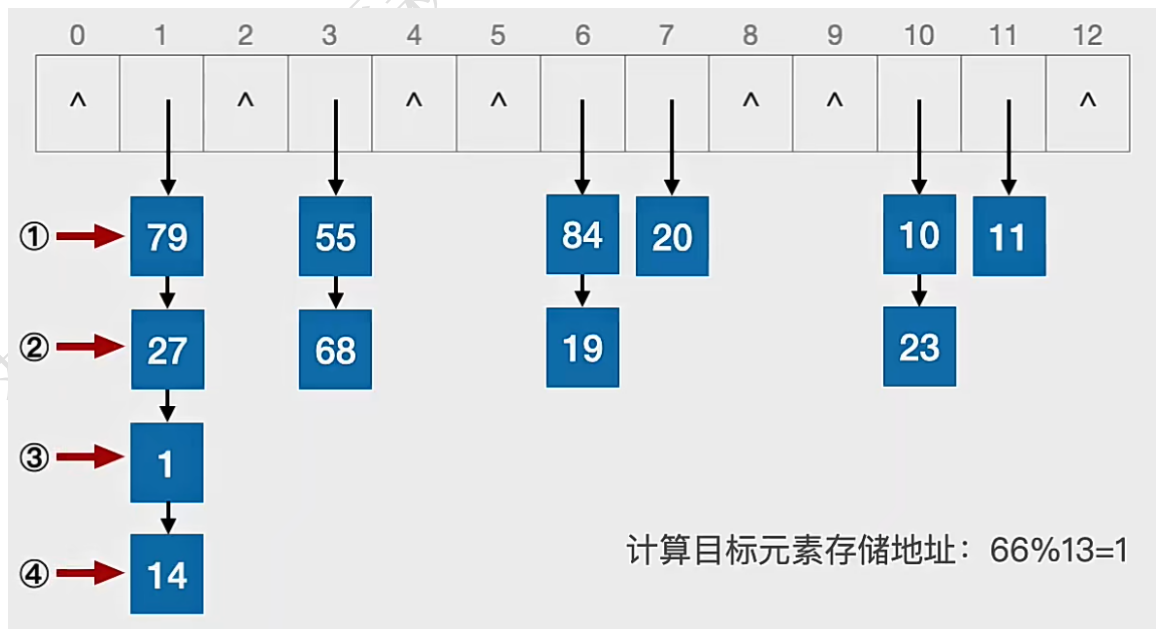
查找

- 注意：
 - 有的教材会把“空指针的对比”也计入查找长度。但考试中默认只统计关键字对比次数；
- 举例：

- 寻找数字 27:
 - 查找了两次 -> 查找长度为 2;



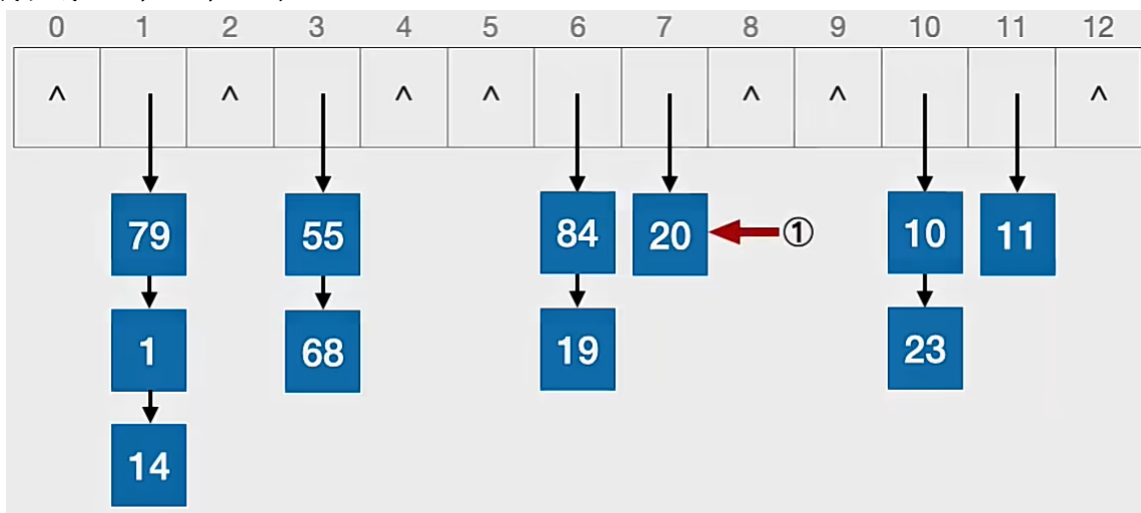
- 查找失败: 找 66, 查找长度为 4



删除

- 概念:
 - 只要能查找成功, 就可以删除成功;
- 举例:

- 若散列表长度为 13，散列函数 $H(key) = key \% 13$ ，用拉链法解决冲突。在下述散列表中依次删除元素 27，20，66；



28.4 开放定址法

概念

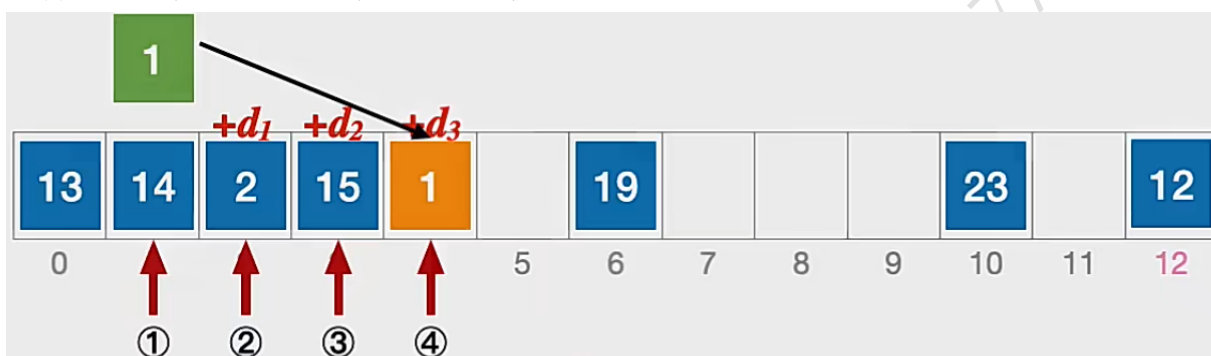
- 思想：
 - 如果发生“冲突”，就给新元素找另一个空闲位置；
 - 开放定址 $\rightarrow$ 一个地址除了可以存储这个地址的同义词，还可以存储不属于这个地址的非同义词；
 - 一个散列地址，既对同义词开放，也对非同义词开放；
 - 此时需确定一个“探测的顺序”，从初始散列地址出发，去寻找下一个空闲位置；
- 方法：
 - 1. 根据散列函数 $H(key)$ ，求得初始散列地址；
 - 2. 若发生冲突，则按照初始散列地址+偏移量，得到新的地址 $\rightarrow$

$$H_i = (H(key) + d_i) \% m;$$
 - $H(key)$ 为初始散列地址；
 - d_i 为偏移量；
 - m 为散列表的表长；
- 四种设计探测序列的方法：
 - 1. 线性探测法： $d_i = 0, 1, 2, 3, \dots, m - 1$
 - 2. 平方探测法： $d_i = 0^2, 1^2, -1^2, 2^2, -2^2, \dots, k^2, -k^2$ 。其中 $k \leq m/2$
 - 3. 双散列法： $d_i = i \times hash_2(key)$ 。其中 $hash_2(key)$ 是另一散列函数；
 - 4. 伪随机序列法： d_i 是一个伪随机序列，如 $d = 0, 5, 3, 11, \dots$
- 注意：
 - 所有的探测序列， $d_0 = 0$
 - 所有的探测序列，最大值不大于 $m - 1$

28.4.1 线性探测法

概念

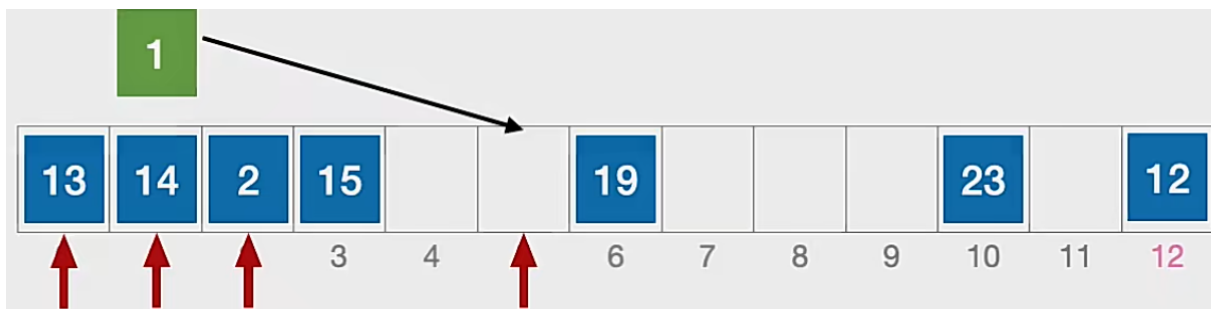
- 探测序列：
 - $d_i = 0, 1, 2, 3, \dots, m - 1$
- 概念：
 - 每次冲突，往后退一格；
 - 查找操作原理类似，根据探测序列依次对比各存储单元内的关键字；
 - 若探测到目标关键字，则查找成功。若探测到空单元，则查找失败；
- 覆盖率：
 - 若散列表表长 $= m$ ，则最多发生 $m - 1$ 次冲突即可“探测”完整个散列表
- 举例：
 - 长度为 13 的散列表状态如下图所示，散列函数 $H(\text{key}) = \text{key} \% 13$ ，采用线性探测法解决冲突；
 - 分析插入元素 1、查找元素 1 的过程；



28.4.2 平方探测法

概念

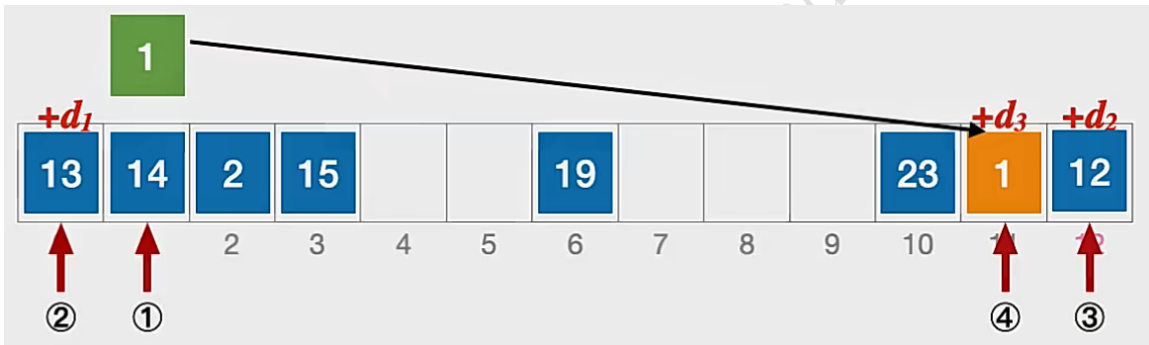
- 探测序列：
 - $d_i = 0^2, 1^2, -1^2, 2^2, -2^2, \dots, k^2, -k^2$
 - 并且 $(k \leq m/2)$
- 探测覆盖率：
 - 采用平方探测法，至少可以探测到散列表中一半的位置这意味着 **即便散列表中有空闲位置，也未必能插入成功**；
- 特点：
 - 优点：可以避免出现堆积的现象；
 - 缺点：不能探测到所有的单元，但至少也能探测到一半的单元；
- 举例：



28.4.3 双散列法

概念

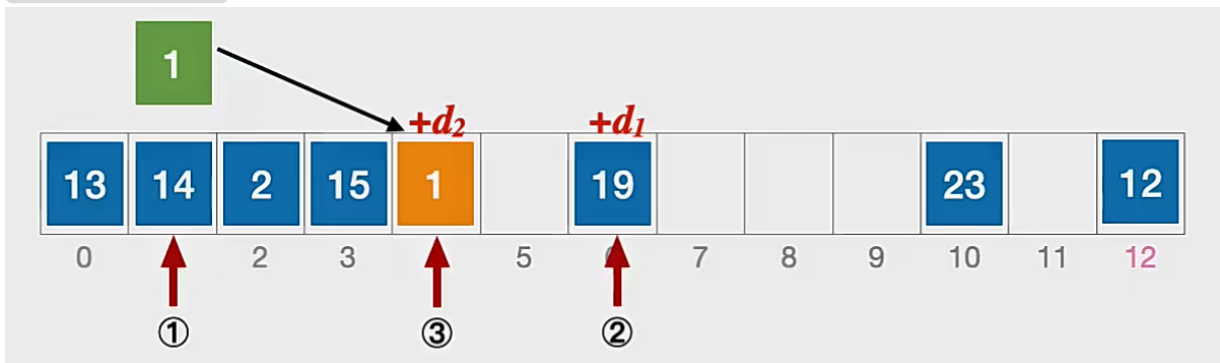
- 探测序列：
 - $d_i = i \times hash_2(key)$
 - 当发生序列不匹配时，依次计算后一个需要计算的位置；
- 探测覆盖率：
 - 双散列法未必能探测到散列表的所有位置；
 - 双散列法的探测覆盖率取决于第二个散列函数 `hash(key)` 设计的是否合理；
 - 若 $hash_2(key)$ 计算得到的值与散列表表长 m 互质，就能保证双散列法可以探测所有单元；
- 结论：
 - $hash_2(key) = m - (key \% m)$
- 举例：
 - 图示：



28.4.4 伪随机法

概念

- 探测序列：
 - 随机数构成 $d_i = 0, 3, 5, 11$
- 探测覆盖率：
 - 取决于其设计是否合理；
- 图示：
 - 初始散列地址 $H_0 = 1 \% 13 = 1$ ，发生冲突 (第 1 次)： $H = (1 + 5) \% 13 = 6$ ，发生冲突 (第 2 次)
 - $H_2 = (1 + 3) \% 13 = 4$ ，未发生冲突，插入位置 #4



28.4.5 删除元素

开放定址法

- 概念：
 - 如果是使用开放定址法，题目一定会具体说清楚采用哪种探测序列（线性探测法、平方探测法、双散列法、伪随机序列法）；
- 实现：
 - 1. 先根据散列函数算出散列地址，并对比关键字是否匹配。若匹配，则“查找成功”；
 - 2. 采用“开放定址法”时，删除元素不能简单地将被删元素的空间置为空,否则将截断在它之后的探测路径，可以**做一个“已删除”标记，进行逻辑删除**；
- 正确示范：删除元素



- 后续删除：
 - 计算元素 1 的初始散列地址= $1\%13=1$ 。对比位置 #1，关键字不等于 1；
 - 根据线性探测法的探测序列，继续对比位置 #2，关键字不等于 1；
 - 根据线性探测法的探测序列，继续对比位置 #3，该位置**原关键字已删，继续探测后一个位置**；