

本节内容

# 处理冲突的 方法

## —开放定址法

# 知识总览

处理冲突的方法  
——开放定址法

开放定址法的原理

四种常用方法

线性探测法

平方探测法

双散列法

伪随机序列法

# 如何处理“冲突”？——开放定址法



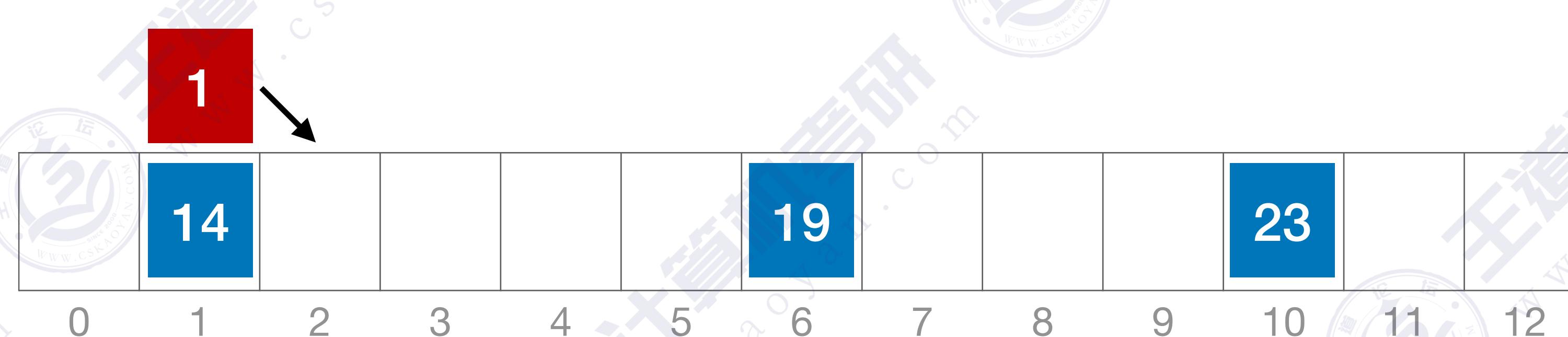
例：某散列表的长度为13，散列函数  $H(key)=key \% 13$ 。依次将数据元素 19、14、23、1 插入散列表：

$$19 \% 13 = 6$$

$$14 \% 13 = 1$$

$$23 \% 13 = 10$$

$$1 \% 13 = 1$$



开放定址法：如果发生“冲突”，就给新元素找另一个空闲位置。

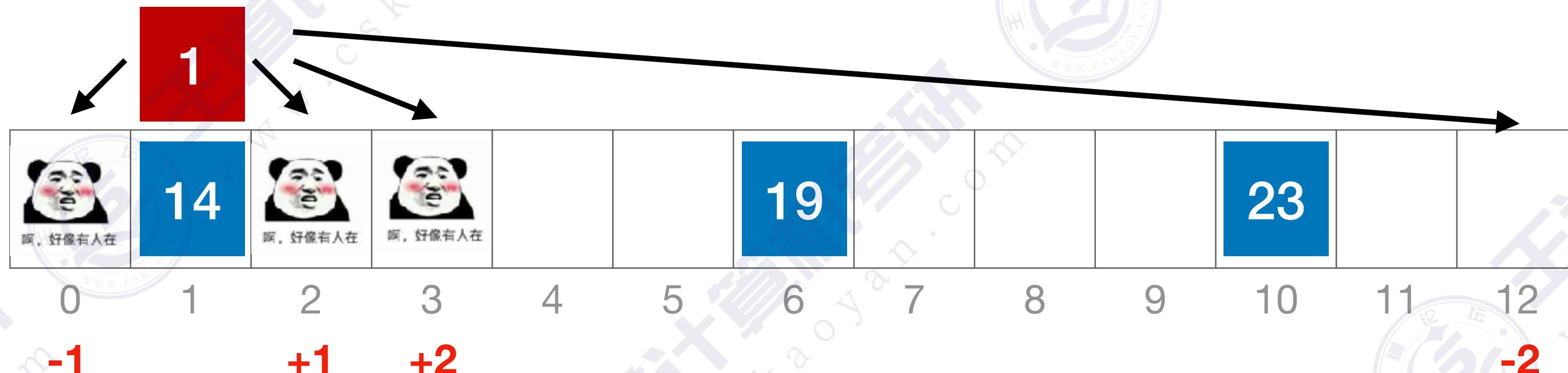
为什么叫“开放定址”？——一个散列地址，既对同义词开放，也对非同义词开放。

# 开放定址法的基本原理



开放定址法：如果发生“冲突”，就给新元素找另一个空闲位置。

$$\begin{aligned}19\%13 &= 6 \\14\%13 &= 1 \\23\%13 &= 10 \\1\%13 &= 1\end{aligned}$$



待解决的问题：用什么规则确定“另一个空闲位置”？

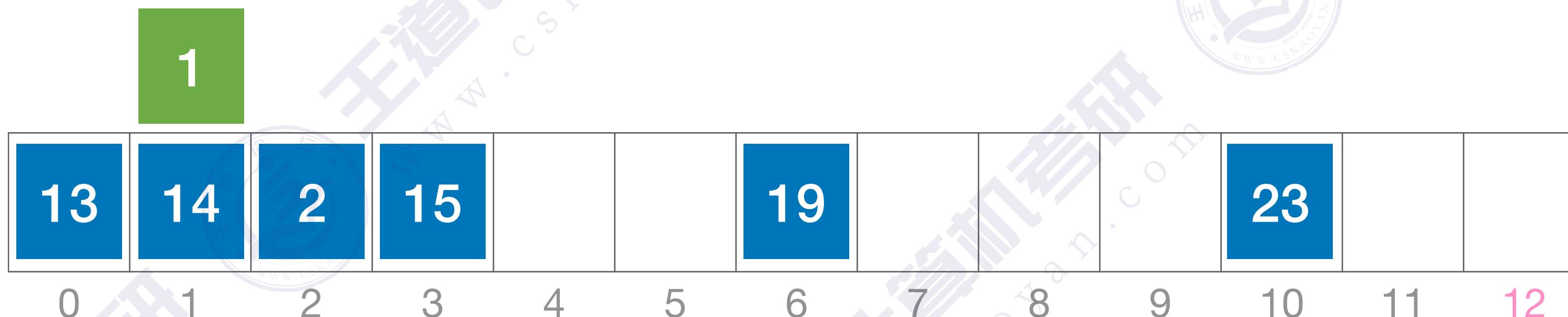
思路：需确定一个“探测的顺序”，从初始散列地址出发，去寻找下一个空闲位置。

eg:  $d_0=0, d_1=1, d_2=-1, d_3=2, d_4=-2, \dots$

注： $d_i$  表示第  $i$  次发生冲突时，下一个探测地址与初始散列地址的相对偏移量。

# 开放定址法的基本原理

根据散列函数  $H(key)$ , 求得初始散列地址。若发生冲突, 如何找到“另一个空闲位置”?



发生第  $i$  次冲突  
时的散列地址

散列表  
表长

$$H_i = (H(key) + d_i) \% m$$

初始散  
列地址

偏移量

四种常用方法  
构造探测序列  $d_i$

注:  $0 \leq i \leq m-1$

探测序列/增量序列

线性探测法

$$d_i = 0, 1, 2, 3, \dots, m-1$$

平方探测法

$$d_i = 0^2, 1^2, -1^2, 2^2, -2^2, \dots, k^2, -k^2 \text{。其中 } k \leq m/2$$

双散列法

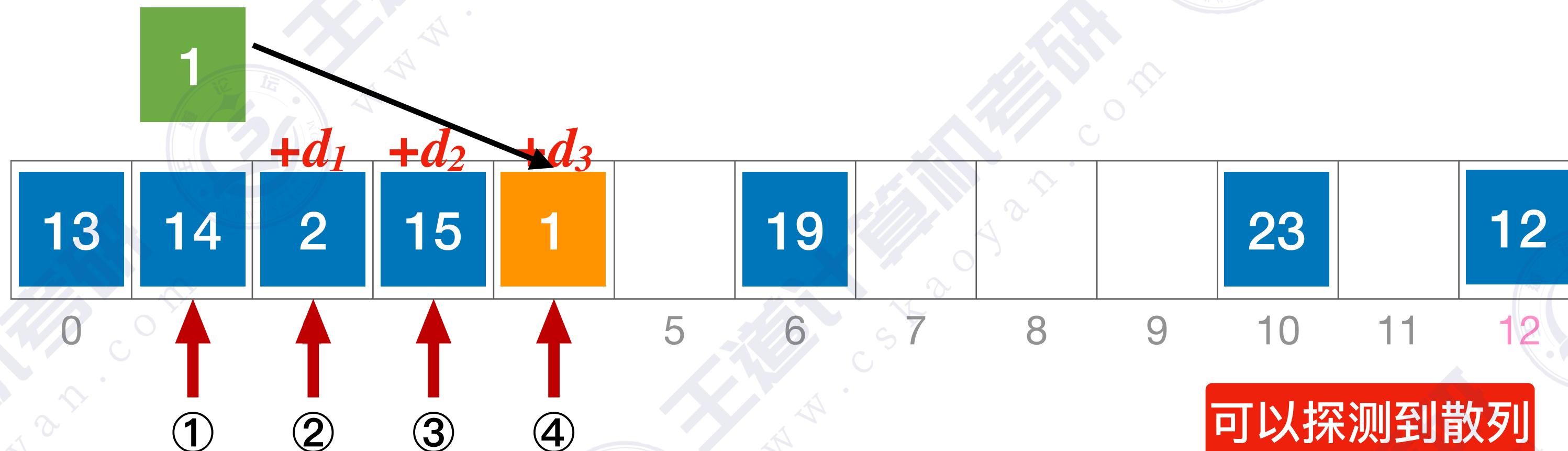
$$d_i = i \times \text{hash}_2(key) \text{。其中 } \text{hash}_2(key) \text{ 是另一散列函数}$$

伪随机序列法

$d_i$  是一个伪随机序列, 如  $d_i = 0, 5, 3, 11, \dots$

# 线性探测法 (插入、查找操作)

例：长度为13的散列表状态如下图所示，散列函数  $H(key)=key \% 13$ ，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



发生第*i*次冲突时的散列地址

散列表表长

$$H_i = (H(key) + d_i) \% m$$

初始散列地址

偏移量

初始散列地址  $H_0 = 1 \% 13 = 1$ ，发生冲突(第1次)

$H_1 = (1 + 1) \% 13 = 2$ ，发生冲突(第2次)

$H_2 = (1 + 2) \% 13 = 3$ ，发生冲突(第3次)

$H_3 = (1 + 3) \% 13 = 4$ ，未发生冲突，插入位置#4

注：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。

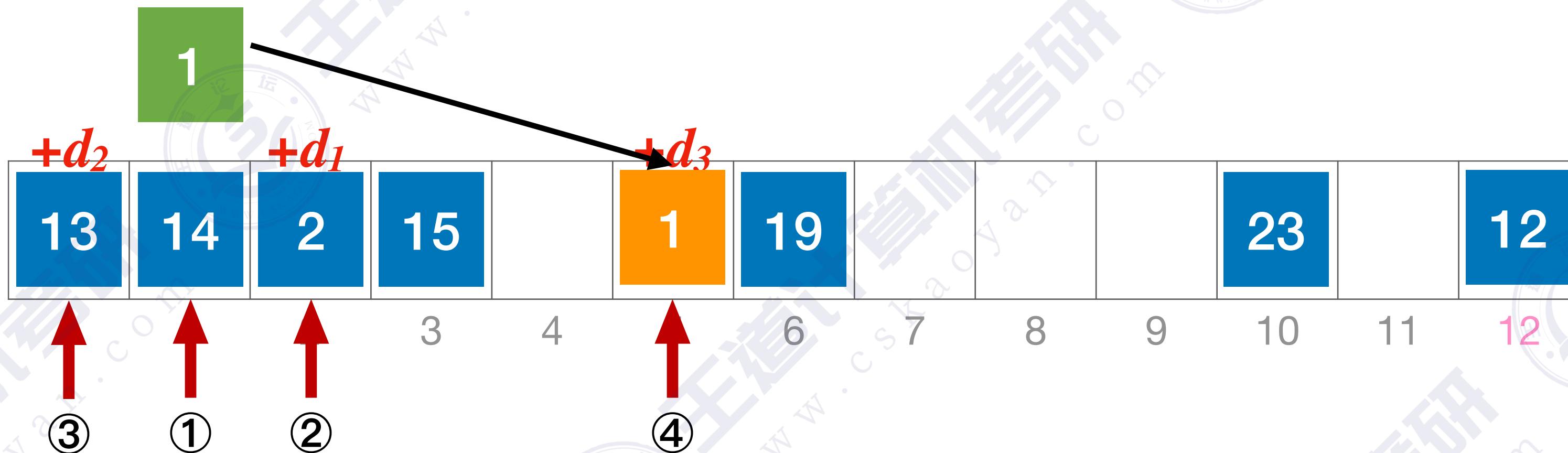
若探测到目标关键字，则查找成功。

若探测到空单元，则查找失败。

又称“二次探测法”

# 平方探测法 (插入、查找操作)

例：长度为13的散列表状态如下图所示，散列函数  $H(key)=key \% 13$ ，采用平方探测法解决冲突。分析：插入元素1、查找元素1的过程



发生第*i*次冲突  
时的散列地址

散列表  
表长

$$H_i = (H(key) + d_i) \% m$$

初始散  
列地址

偏移量

平方探测法， $d_i = 0^2, 1^2, -1^2, 2^2, -2^2, \dots, k^2, -k^2$ 。  
其中  $k \leq m/2$

初始散列地址  $H_0 = 1 \% 13 = 1$ ，发生冲突(第1次)

$H_1 = (1 + 1) \% 13 = 2$ ，发生冲突(第2次)

$H_2 = (1 + -1) \% 13 = 0$ ，发生冲突(第3次)

$H_3 = (1 + 4) \% 13 = 5$ ，未发生冲突，插入位置#5

注：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。

# 双散列法 (插入、查找操作)

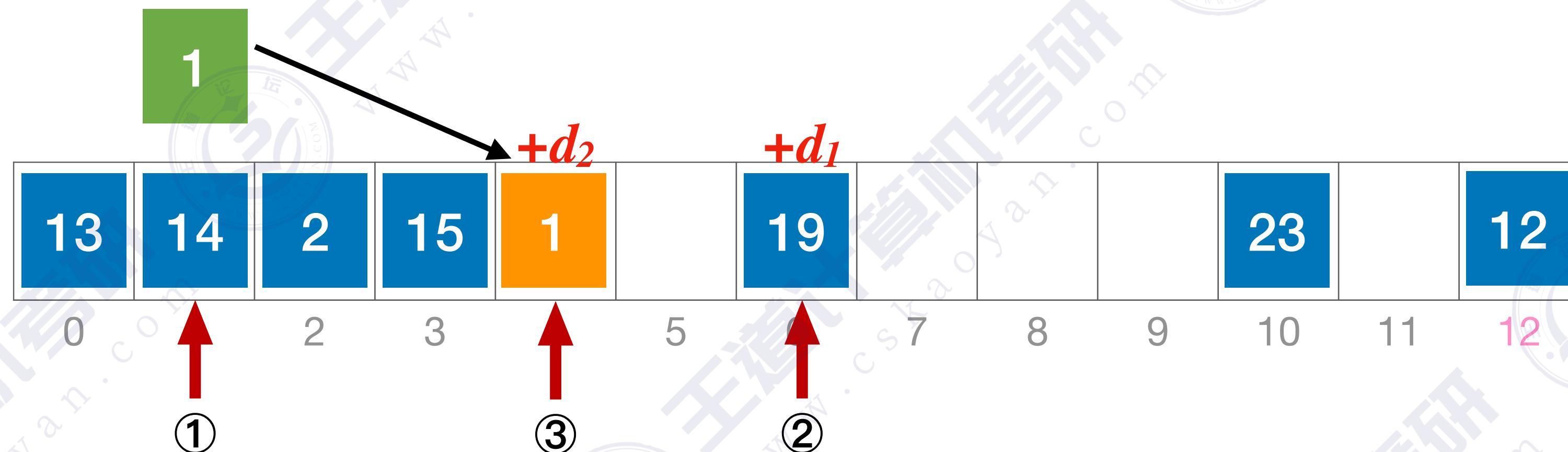
例：长度为13的散列表状态如下图所示，散列函数  $H(key)=key \% 13$ ，采用双散列法解决冲突，假设  $hash_2(key)=13-(key \% 13)$ 。分析：插入元素1、查找元素1的过程



注：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。

# 伪随机序列法 (插入、查找操作)

例：长度为13的散列表状态如下图所示，散列函数  $H(key)=key \% 13$ ，采用伪随机序列法解决冲突，假设伪随机序列  $d_i=0, 5, 3, 11, \dots$ ，其中  $d_i$  表示第  $i$  次发生冲突时的增量。分析：插入元素1、查找元素1的过程



发生第  $i$  次冲突时的散列地址

$$H_i = (H(key) + d_i) \% m$$

初始散列地址

散列表表长

偏移量

$d_i$  是一个伪随机序列，由题目可知  $d_i=0, 5, 3, 11, \dots$

初始散列地址  $H_0=1 \% 13=1$ ，发生冲突(第1次)

$H_1 = (1 + 5) \% 13 = 6$ ，发生冲突(第2次)

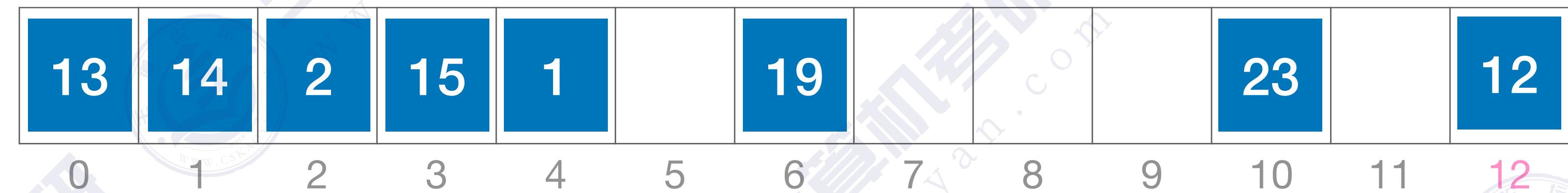
$H_2 = (1 + 3) \% 13 = 4$ ，未发生冲突，插入位置#4

注：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。

# 如何删除一个元素？

注：题目一定会说明具体是采用哪种探测序列（线性探测法、平方探测法、双散列法、伪随机序列法）

例：长度为13的散列表状态如下图所示，散列函数  $H(key)=key \% 13$ ，采用开放定址法解决冲突。



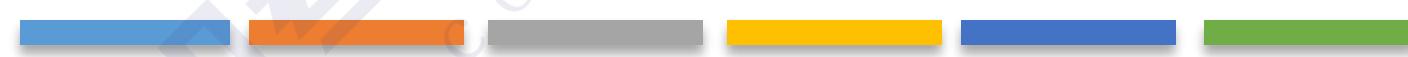
## 如何删除一个元素：

Step 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”

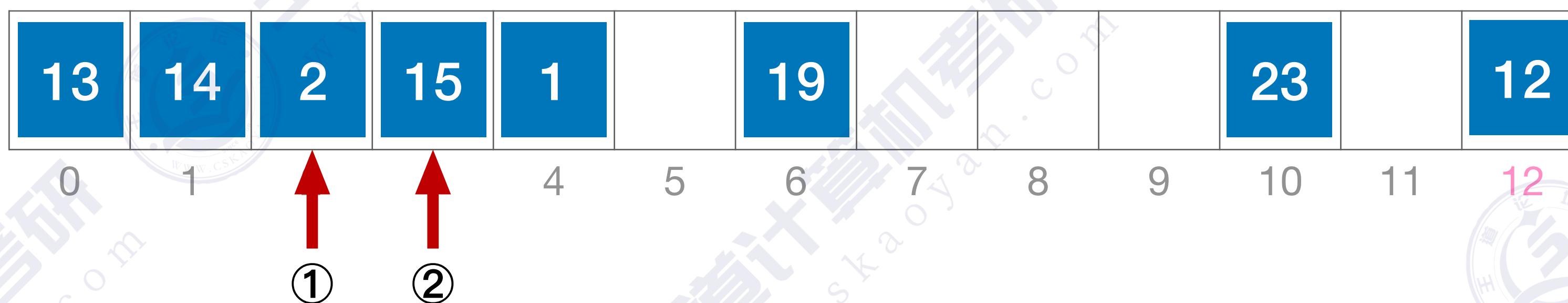
Step 2：若关键字不匹配，则根据“探测序列”对比下一个地址的关键字，直到“查找成功”或“查找失败”

Step 3：若“查找成功”，则删除找到的元素

# 特别注意：关于删除操作



例：长度为13的散列表状态如下图所示，散列函数  $H(key)=key \% 13$ ，采用线性探测法解决冲突。



## 错误示范：删除元素15

- 计算元素15 的初始散列地址= $15 \% 13 = 2$ 。对比位置#2，关键字不等于15；
- 根据线性探测法的探测序列，继续对比位置#3，关键字等于15；
- **删除元素15，清空位置#3**

# 特别注意：关于删除操作



例：长度为13的散列表状态如下图所示，散列函数  $H(key)=key \% 13$ ，采用线性探测法解决冲突。



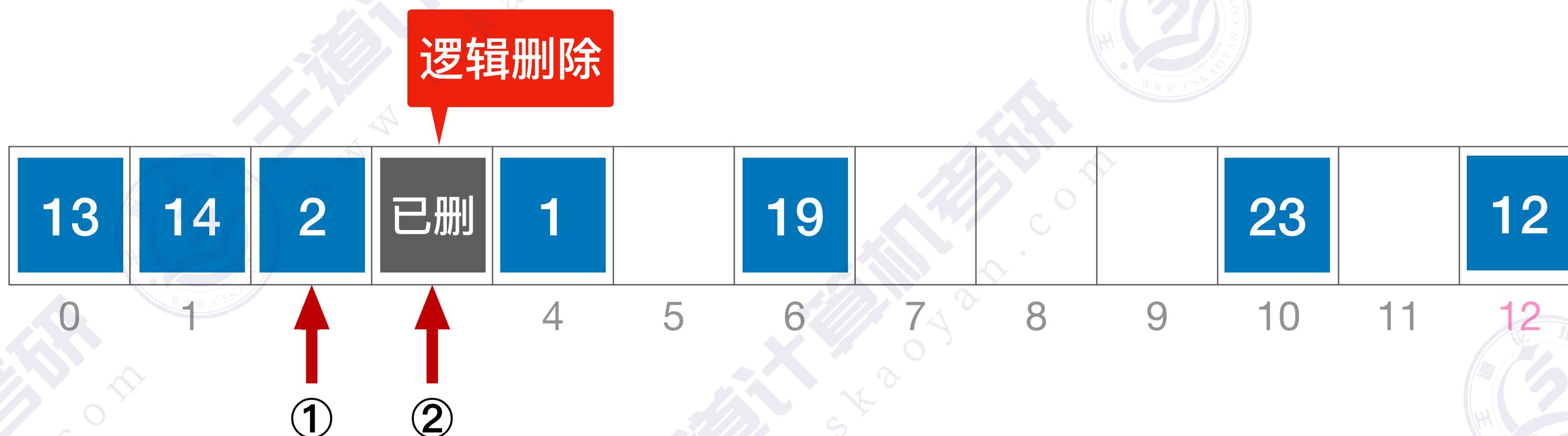
## 错误示范：查找元素1

- 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，探测到空单元，查找失败。

# 特别注意：关于删除操作



例：长度为13的散列表状态如下图所示，散列函数  $H(key)=key \% 13$ ，采用线性探测法解决冲突。



## 正确示范：删除元素15

- 计算元素15的初始散列地址= $15 \% 13=2$ 。对比位置#2，关键字不等于15；
- 根据线性探测法的探测序列，继续对比位置#3，关键字等于15；
- 逻辑删除元素15，将位置#3标记为“已删除”

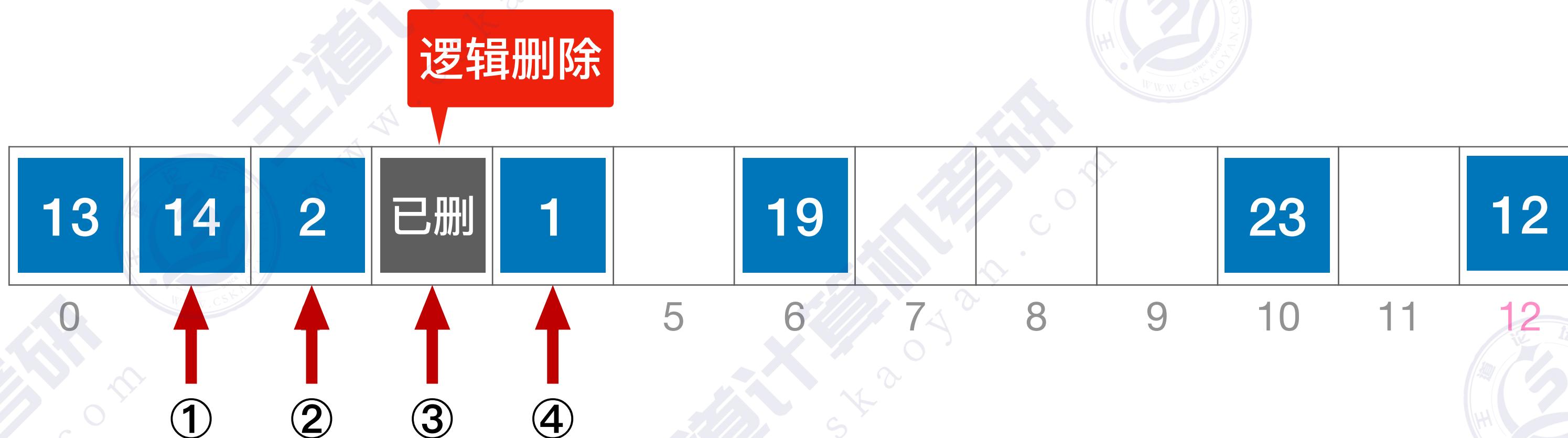
注：无论线性探测法、平方探测法、双散列法、伪随机序列法原理都一样。删除元素时，只能逻辑删除

注意：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。

# 特别注意：关于删除操作



例：长度为13的散列表状态如下图所示，散列函数  $H(key)=key \% 13$ ，采用线性探测法解决冲突。



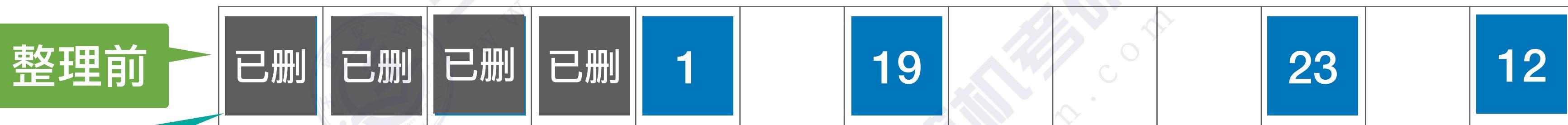
## 正确示范：查找元素1

- 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；
- 根据线性探测法的探测序列，继续对比位置#4，关键字等于1，查找成功。

# 特别注意：关于删除操作



注意：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。

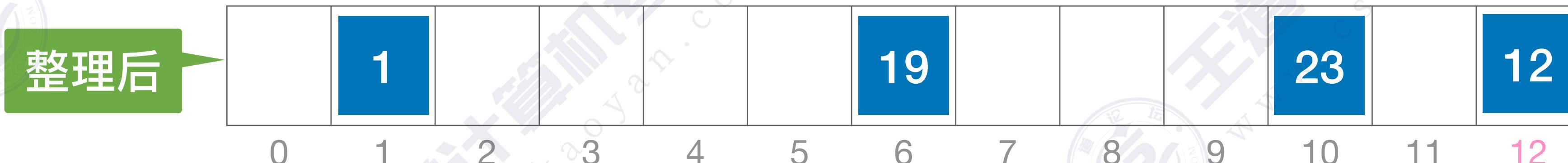


注：新元素也可以插入到已被“逻辑删除”的地址

带来的问题：查找效率低下，散列表看起来很满，实则很空。



Tips：可以不定期整理散列表内的数据。



# 知识回顾与重要考点

$$H_i = (H(key) + d_i) \% m$$

处理冲突的方法  
——开放定址法

基本原理

当初始散列地址发生冲突时，根据“探测序列”  $d_i$  探测下一个地址

插入 —— 根据散列函数算出初始散列地址，若发生冲突，就“探测”下一个地址，直到找到一个空闲地址，即可插入元素

元素的操作  
查找 —— 根据散列函数算出初始散列地址，对比关键字，若关键字不匹配，就“探测”下一个地址，直到关键字匹配(成功)或探测到一个空位置(失败)

① 删除 —— 先按照“查找操作”的规则找到目标元素。若查找成功，就把目标元素“逻辑删除”。注意不能“物理删除”

四种常用的“探测序列”

★ 线性探测法

$$d_i = 0, 1, 2, 3, \dots, m-1$$

平方探测法

$$d_i = 0^2, 1^2, -1^2, 2^2, -2^2, \dots, k^2, -k^2。其中 k \leq m/2$$

双散列法

$$d_i = i \times hash_2(key)。其中 hash_2(key) 是另一散列函数$$

伪随机序列法

$d_i$  是一个伪随机序列，如  $d_i = 0, 5, 3, 11, \dots$

# 拓展：线性探测法的“探测覆盖率”



发生第  $i$  次冲突  
时的散列地址

散列表  
表长

$$H_i = (H(key) + d_i) \% m$$

初始散  
列地址

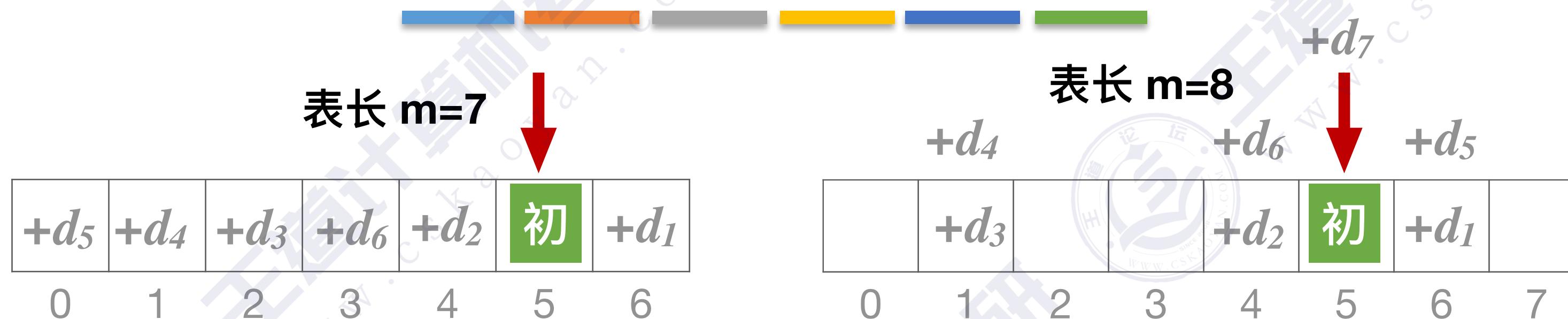
偏移量

线性探测法,  $d_i = 0, 1, 2, 3, \dots, m-1$

采用线性探测法, 一定可以探测到散列表的每个位置  
只要散列表中有空闲位置, 就一定可以插入成功

理想情况下, 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。

# 拓展：平方探测法的“探测覆盖率”



可以探测到  
所有位置

$$\begin{aligned}d_0 &= 0 \quad \text{—— } H_0=5 \\d_1 &= 1 \quad \text{—— } H_1=6 \\d_2 &= -1 \quad \text{—— } H_2=4 \\d_3 &= 4 \quad \text{—— } H_3=2 \\d_4 &= -4 \quad \text{—— } H_4=1 \\d_5 &= 9 \quad \text{—— } H_5=0 \\d_6 &= -9 \quad \text{—— } H_6=3\end{aligned}$$

$$\begin{aligned}d_0 &= 0 \quad \text{—— } H_0=5 \\d_1 &= 1 \quad \text{—— } H_1=6 \\d_2 &= -1 \quad \text{—— } H_2=4 \\d_3 &= 4 \quad \text{—— } H_3=1 \\d_4 &= -4 \quad \text{—— } H_4=1 \\d_5 &= 9 \quad \text{—— } H_5=6 \\d_6 &= -9 \quad \text{—— } H_6=4 \\d_7 &= 16 \quad \text{—— } H_7=5\end{aligned}$$

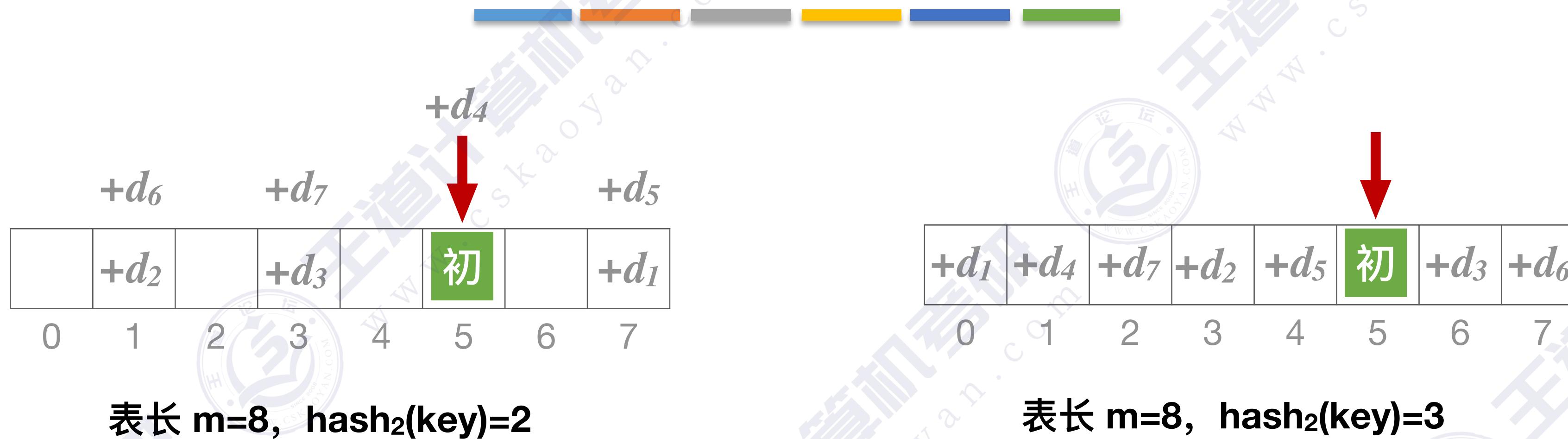
不能探测到  
所有位置

平方探测法,  $d_i = 0^2, 1^2, -1^2, 2^2, -2^2, \dots, k^2, -k^2$ 。其中  $k \leq m/2$ ,  $i \leq m-1$

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

若散列表长度  $m$  是一个可以表示成  $4j + 3$  的素数(如 7、11、19), 平方探测法就能探测到所有位置

# 拓展：双散列法的“探测覆盖率”



双散列法， $d_i = i \times \text{hash}_2(\text{key})$

双散列法未必能探测到散列表的所有位置。

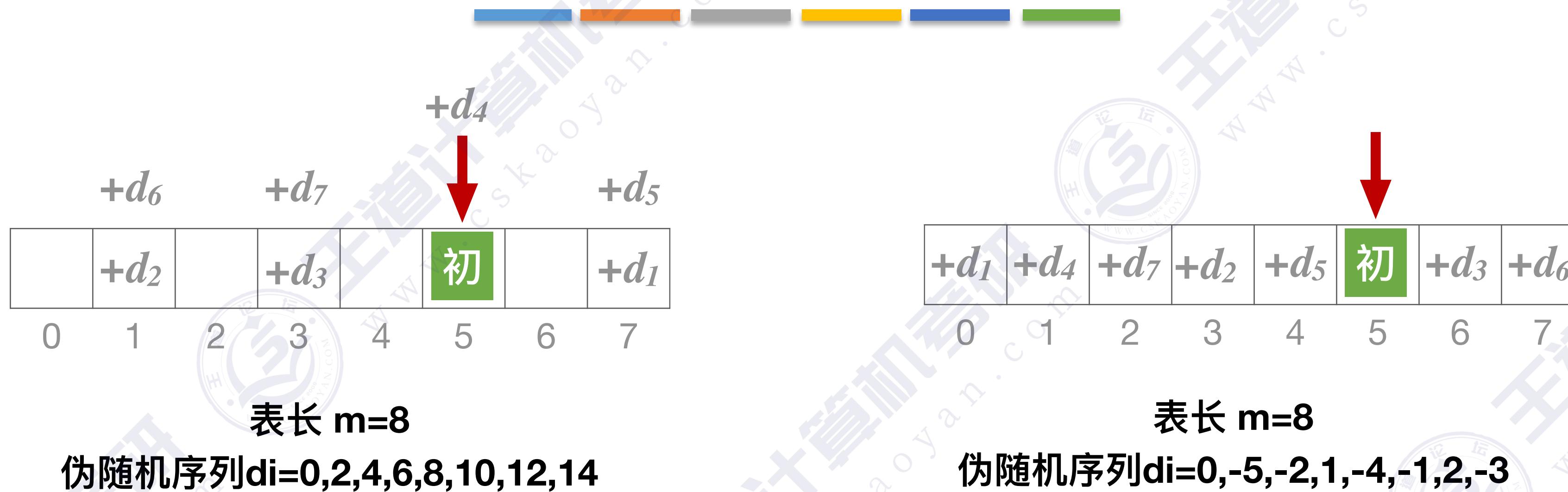
双散列法的探测覆盖率取决于第二个散列函数  $\text{hash}_2(\text{key})$  设计的是否合理。

若  $\text{hash}_2(\text{key})$  计算得到的值与散列表表长  $m$  互质，就能保证双散列发可以探测所有单元

双散列法常用套路：令表长  $m$  本身就是质数， $\text{hash}_2(\text{key}) = m - (\text{key} \% m)$

无论  $\text{key}$  值是多少，  
 $\text{hash}_2(\text{key})$  和  $m$  一定互质

# 拓展：伪随机序列法的“探测覆盖率”



伪随机序列法： $d_i$  是一个伪随机序列，由程序员人为设计

采用伪随机序列法，是否能探测到散列表中全部位置，取决于伪随机序列的设计是否合理

# 拓展：四种探测序列的“探测覆盖率”



## 四种增量序列的“覆盖率”

- 线性探测法 —— 经过  $m-1$  次冲突，一定能探测到散列表的所有单元
- 平方探测法 —— 一般来说，探测序列至少能覆盖到散列表的一半单元。若能保证表长  $m$  是一个可以表示成  $4X+3$  的素数，则可以探测到散列表的所有单元
- 双散列法 —— 若能保证  $\text{hash2}(\text{key})$  的值和表长  $m$  互质，则经过  $m-1$  次冲突，一定能探测到散列表的所有单元
- 伪随机序列法 —— 只要伪随机序列设计合理，就能探测到全部单元