查找算法

胡船长
初航我带你，远航靠自己

本章题目

1-应试. Leetcode-01：两数之和

2-应试. Leetcode-35：搜索插入位置

3-应试. Leetcode-217：存在重复元素

4-应试. Leetcode-349：两个数组的交集

5-校招. Leetcode-03：无重复字符的最长子串

6-校招. Leetcode-04：两个正序数组的中位数

7-竞赛. HZOJ-242：最大平均值

8-竞赛. HZOJ-244：奶牛围栏

本期内容

一. 二分算法

二. 跳跃表（Skiplist）

三.哈希表与布隆过滤器

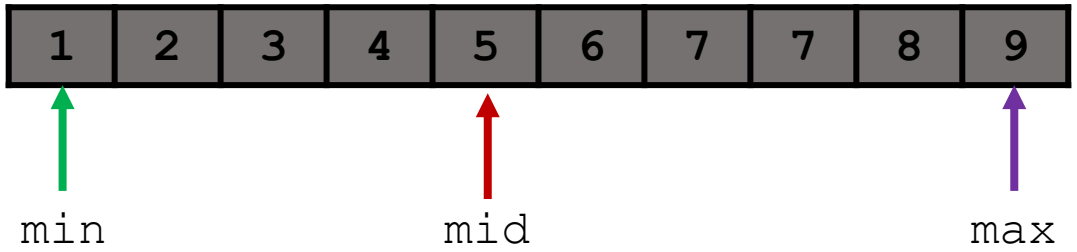
一. 二分算法

二分查找

待查找 x

二分查找

待查找 x



min是头指针；**max**是尾指针；**mid = (min + max) / 2**

调整:

如果arr[mid] < x, min = mid + 1
如果arr[mid] > x, max = mid - 1
如果arr[mid] == x, 找到结果

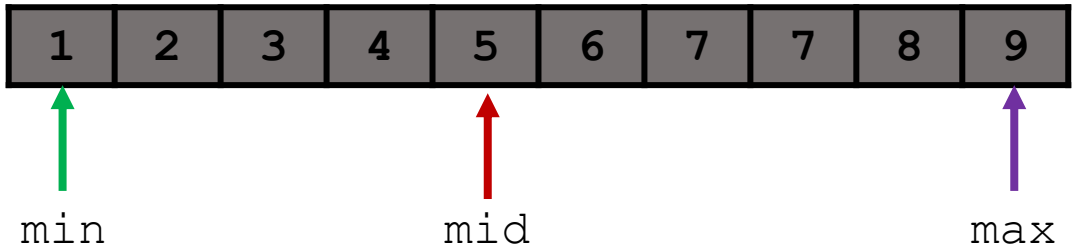
终止条件: min >= max

二分查找

待查找 x

二分查找

待查找 x



min是头指针；**max**是尾指针；**mid = (min + max) / 2**

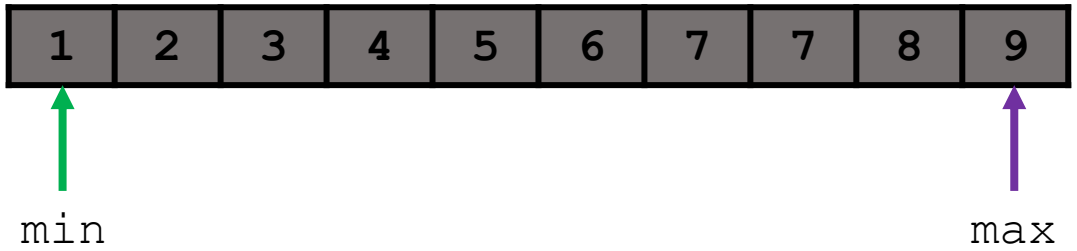
调整:

如果arr[mid] < x, min = mid + 1
如果arr[mid] > x, max = mid - 1
如果arr[mid] == x, 找到结果

终止条件: min >= max

二分查找

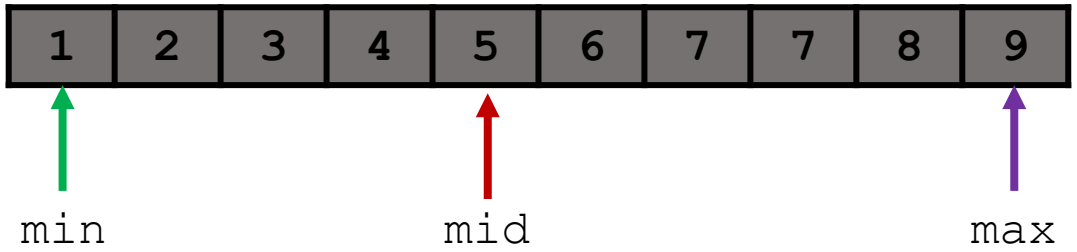
待查找 7



arr[mid] < x, min = mid + 1
重新计算，得到 mid = 6

二分查找

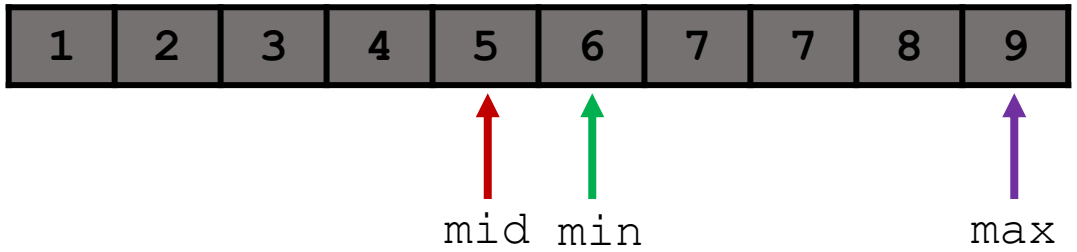
待查找 7



arr[mid] < x, min = mid + 1
重新计算，得到 mid = 6

二分查找

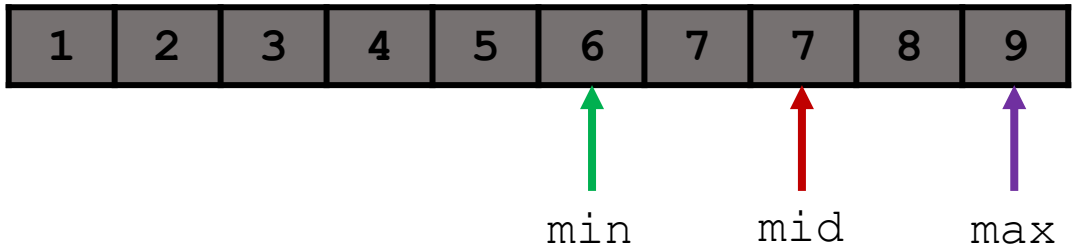
待查找 7



arr[mid] < x, min = mid + 1
重新计算，得到 mid = 6

二分查找

待查找 7

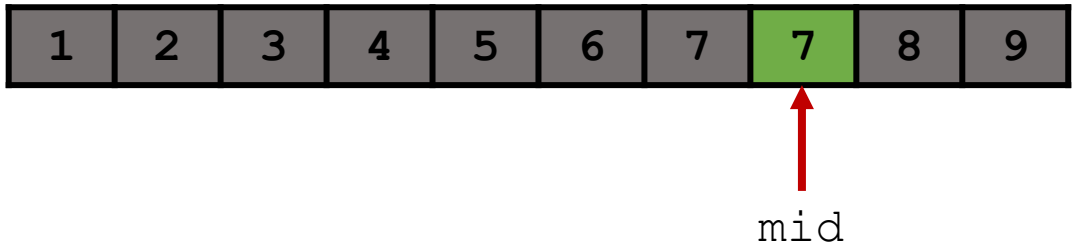


arr[mid] == x，找到结果

总查找次数：2次

二分查找

待查找 7

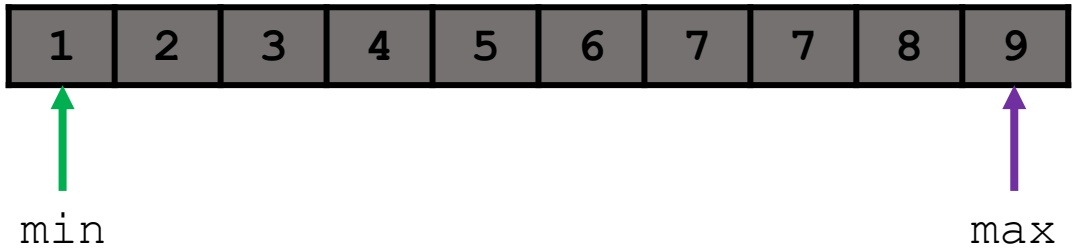


arr[mid] == x，找到结果

总查找次数：2次

二分查找

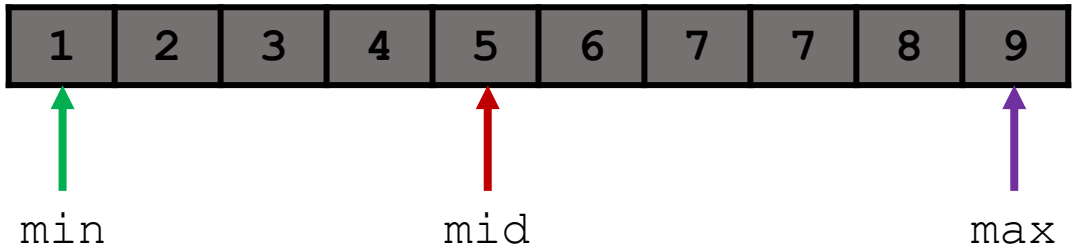
待查找 4



采用二分查找，总查找次数为：?次

二分查找

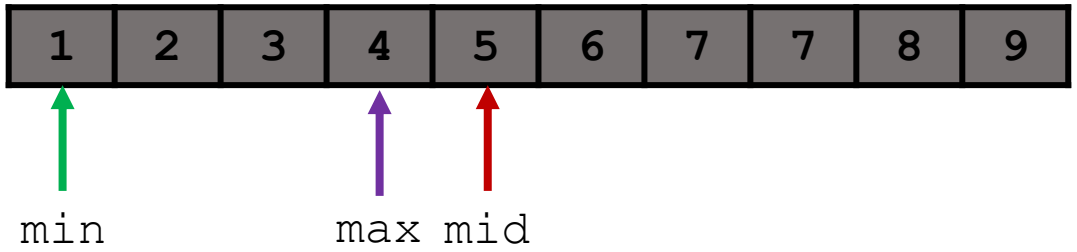
待查找 4



采用二分查找，总查找次数为：?次

二分查找

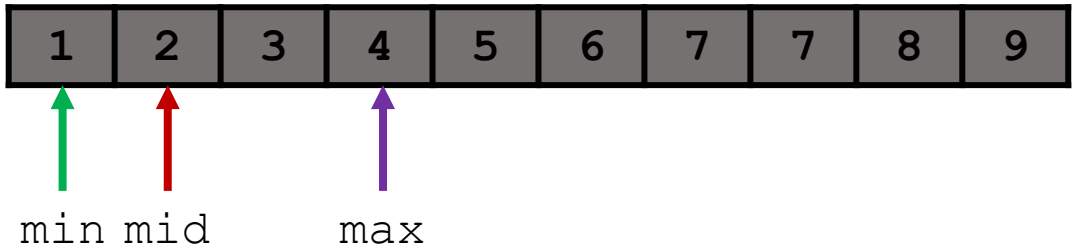
待查找 4



采用二分查找，总查找次数为：?次

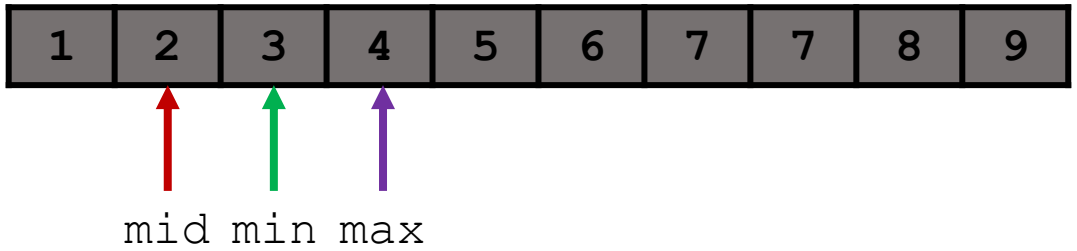
二分查找

待查找 4



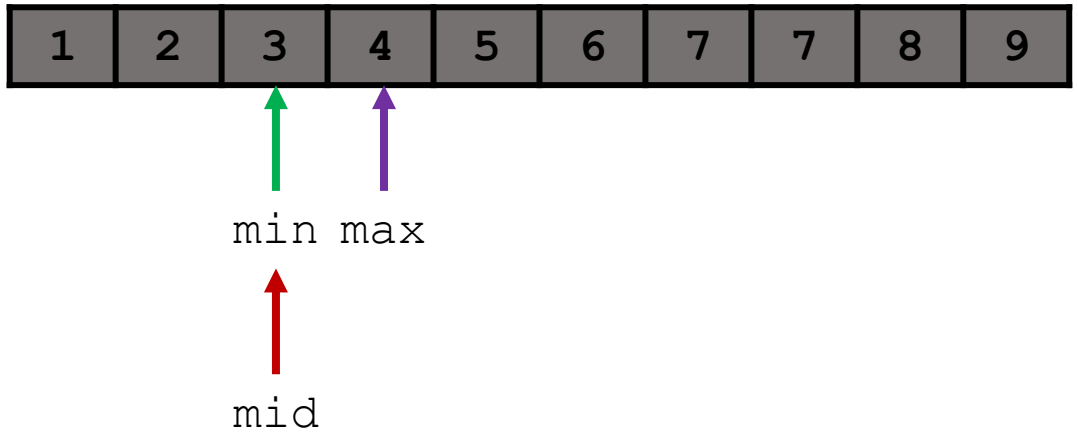
二分查找

待查找 4



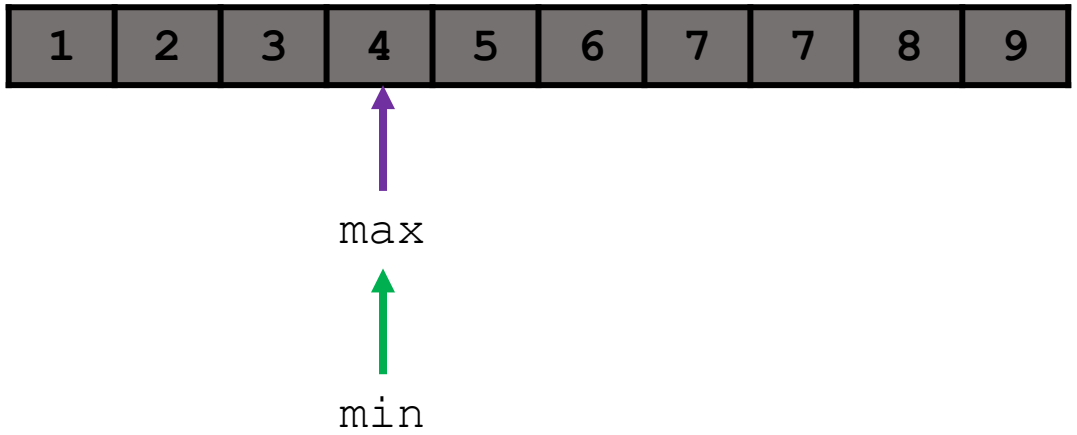
二分查找

待查找 4



二分查找

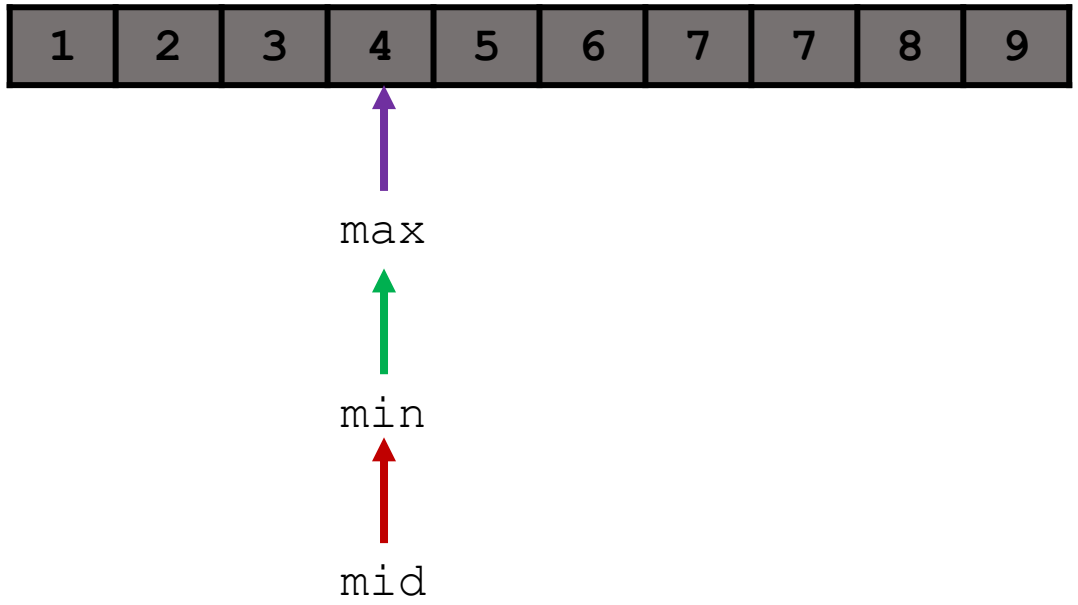
待查找 4



采用二分查找，总查找次数为：4次

二分查找

待查找 4



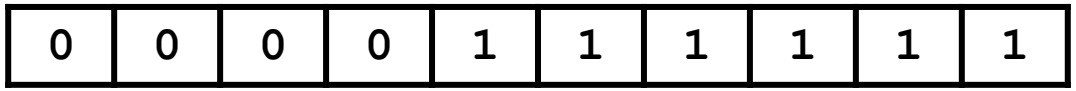
采用二分查找，总查找次数为：4次

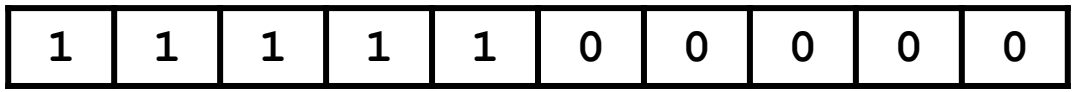
二分查找

待查找 4

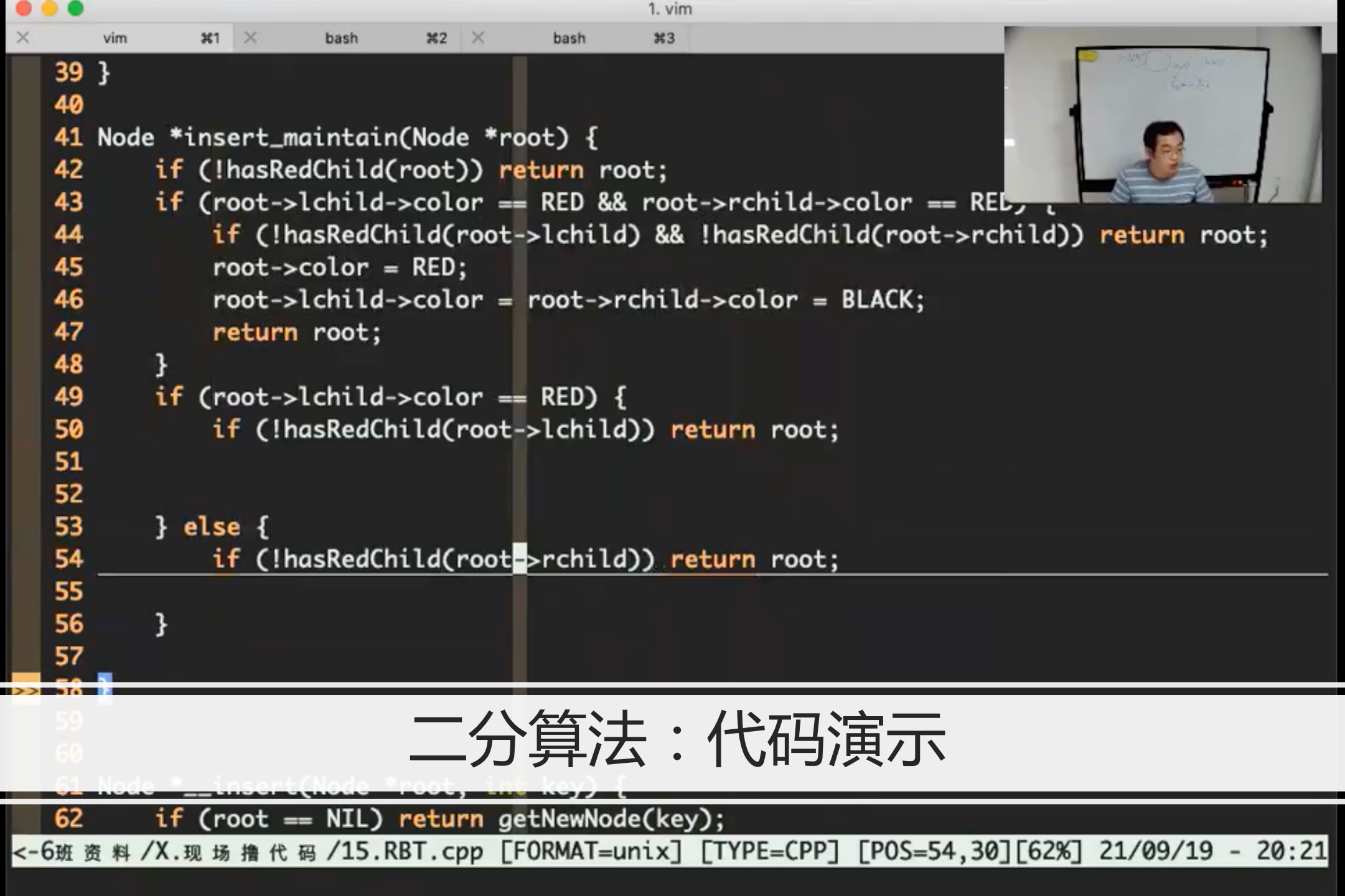
采用二分查找，总查找次数为：4次

二分查找—泛型情况





二分中的数组和函数的关系


1. vim
vim ⌘1
bash ⌘2
bash ⌘3
39 }
40
41 Node *insert_maintain(Node *root) {
42 if (!hasRedChild(root)) return root;
43 if (root->lchild->color == RED && root->rchild->color == RED) {
44 if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45 root->color = RED;
46 root->lchild->color = root->rchild->color = BLACK;
47 return root;
48 }
49 if (root->lchild->color == RED) {
50 if (!hasRedChild(root->lchild)) return root;
51
52
53 } else {
54 if (!hasRedChild(root->rchild)) return root;
55
56 }
57
58 }
59
60
61 Node *__insert(Node *root, int key) {
62 if (root == NIL) return getNewNode(key);
<-6班 资 料 /X.现 场 撸 代 码 /15.RBT.cpp [FORMAT=unix] [TYPE=CPP] [POS=54,30][62%] 21/09/19 - 20:21

二分算法：代码演示

随堂练习题：

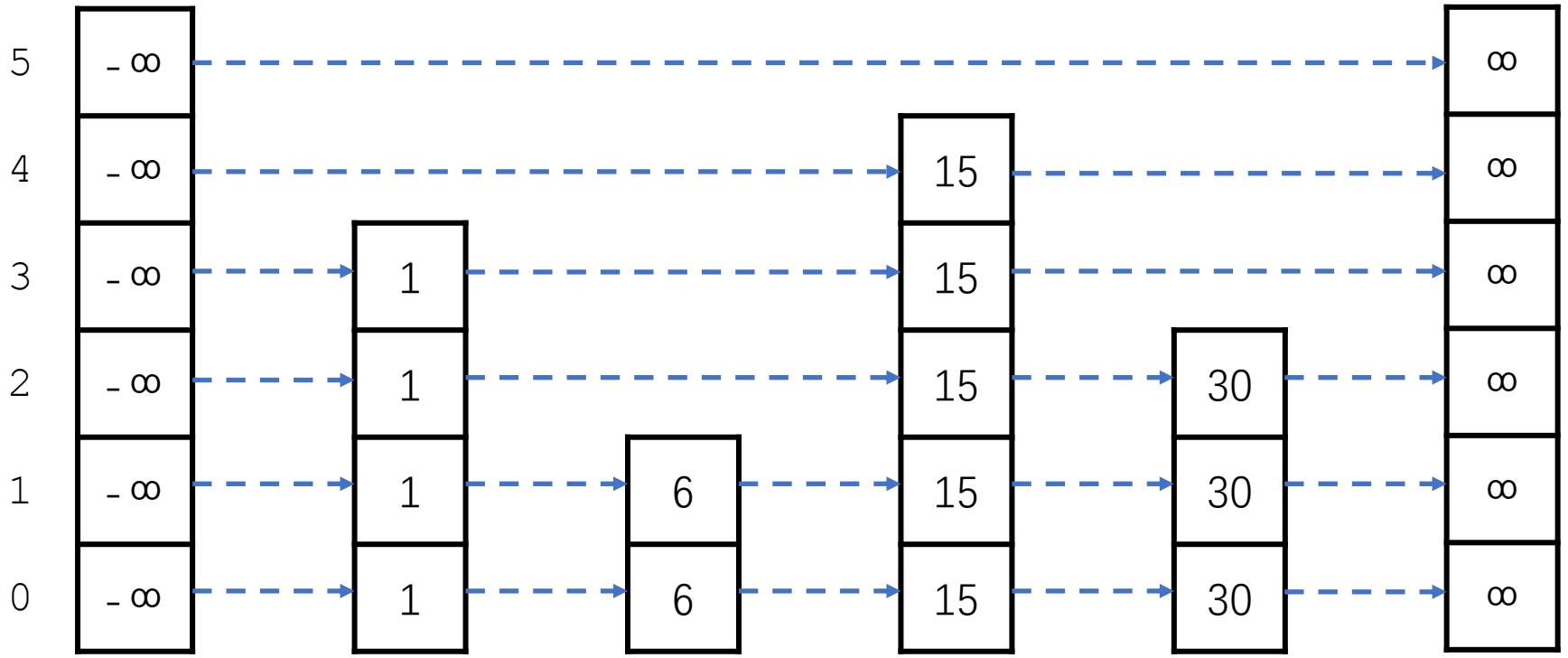
个人所得税，是根据收入进行阶梯设置的，我国个人所得税现行标准如下：

个人所得税税率表三（非居民公司薪金所得等按月换算）			
级数	全年应纳税所得额	税率（%）	速算扣除数
1	不超过3000元的	3	0
2	超过3000元至12000元的部分	10	210
3	超过12000元至25000元的部分	20	1410
4	超过25000元至35000元的部分	25	2660
5	超过35000元至55000元的部分	30	4410
6	超过55000元至80000元的部分	35	7160
7	超过80000元的	45	15160

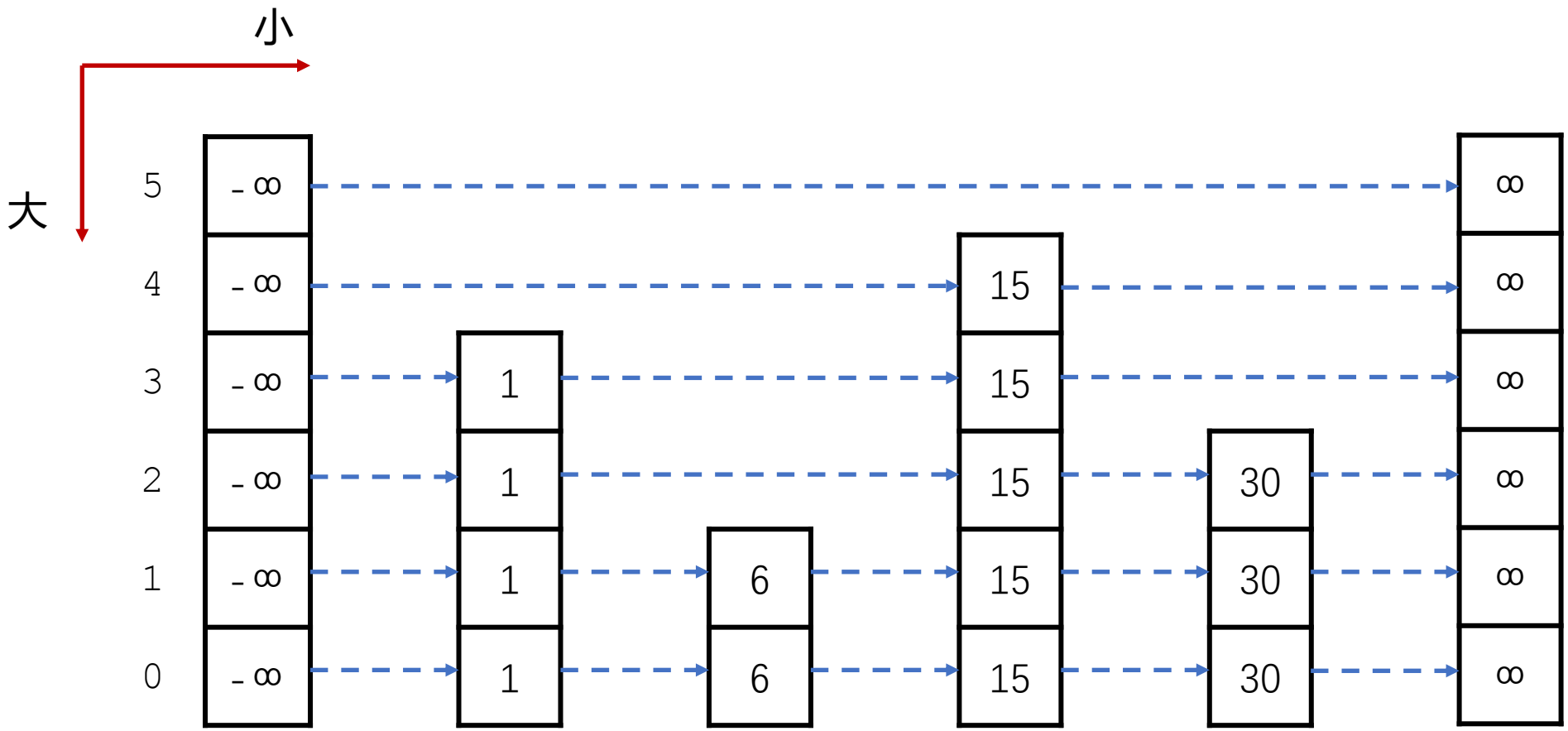
问题：给出税后收入，求税前收入

二. 跳跃表（Skiplist）

跳跃表（Skiplist）

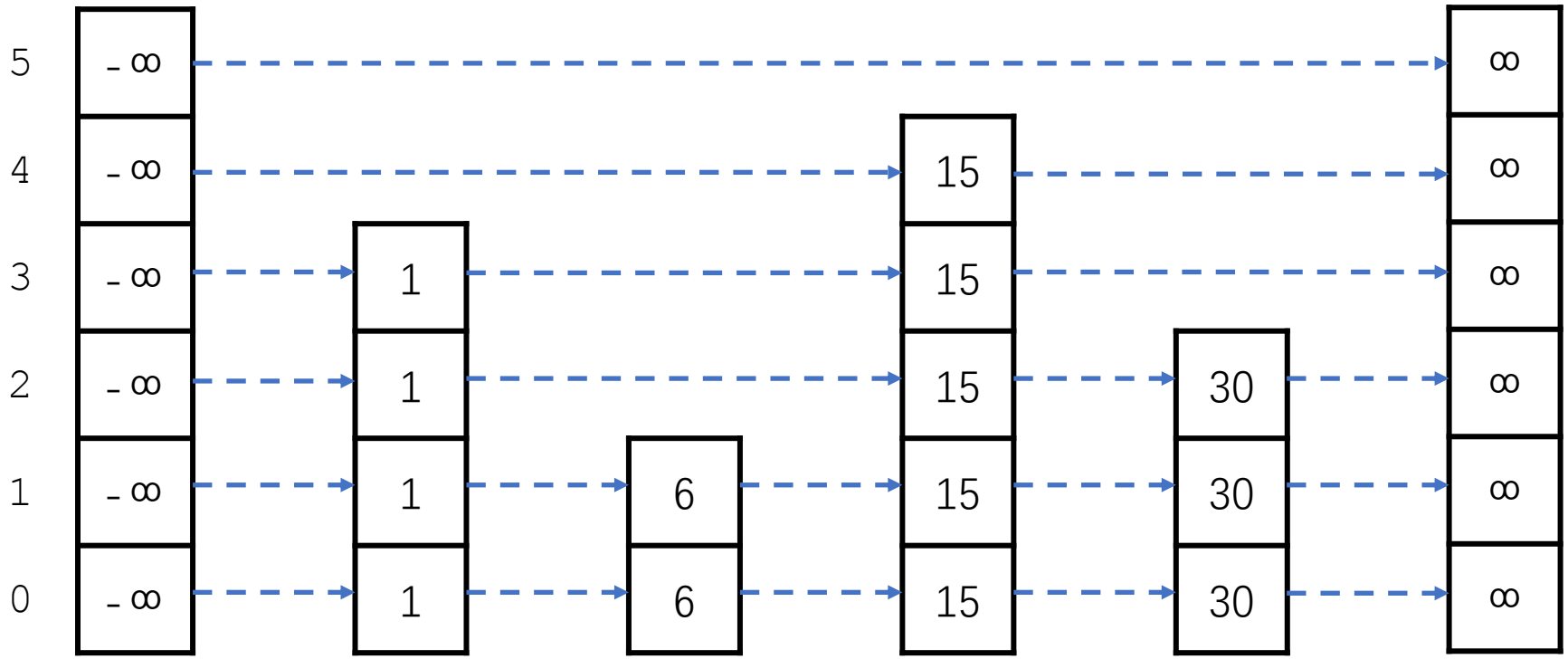


跳跃表（Skiplist）-查找操作



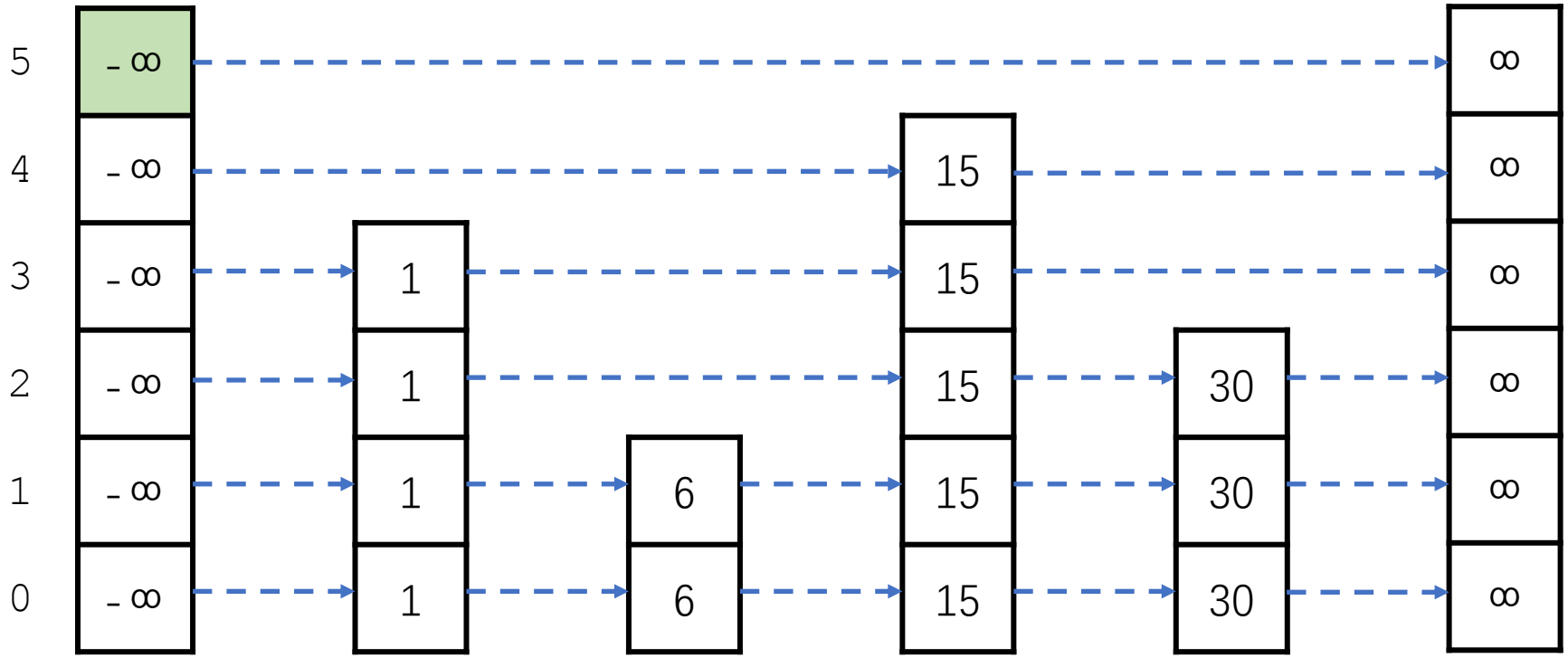
跳跃表（Skiplist）-查找操作

查找 15



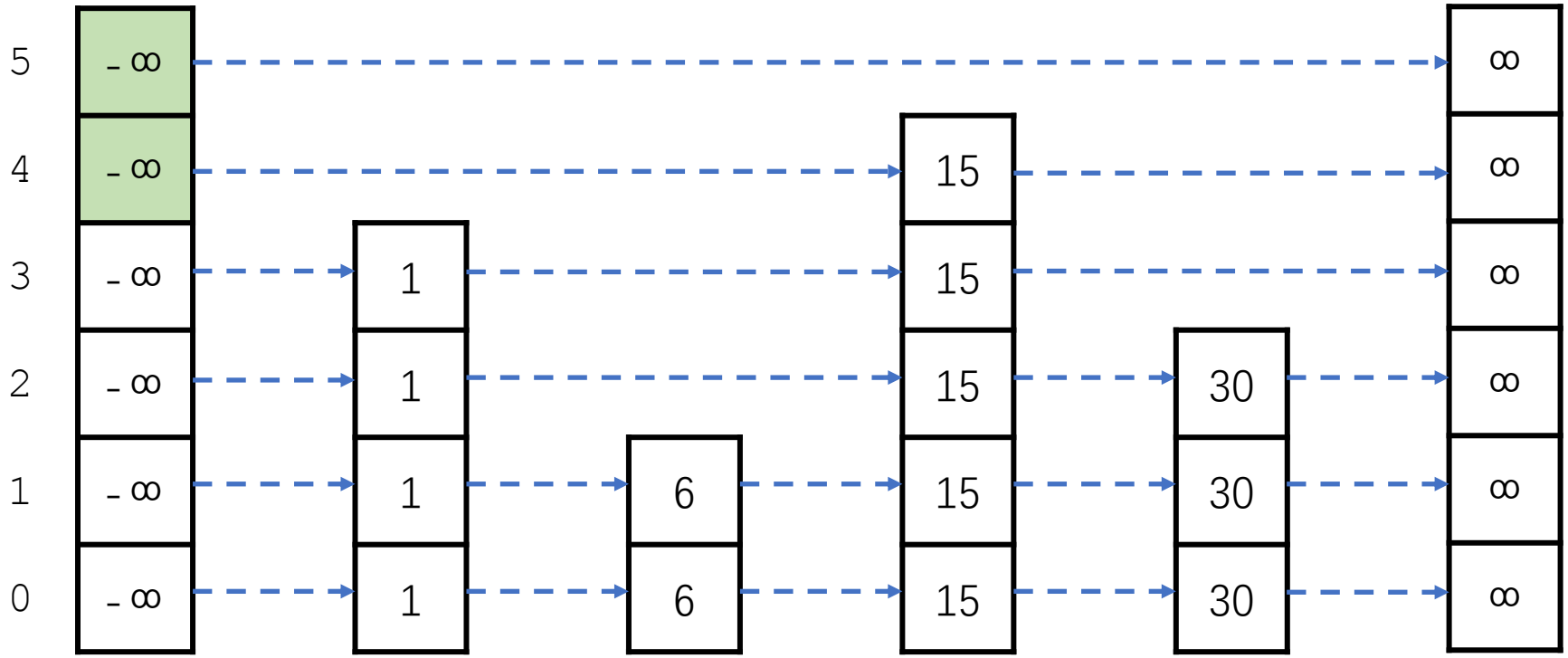
跳跃表（Skiplist）-查找操作

查找 15



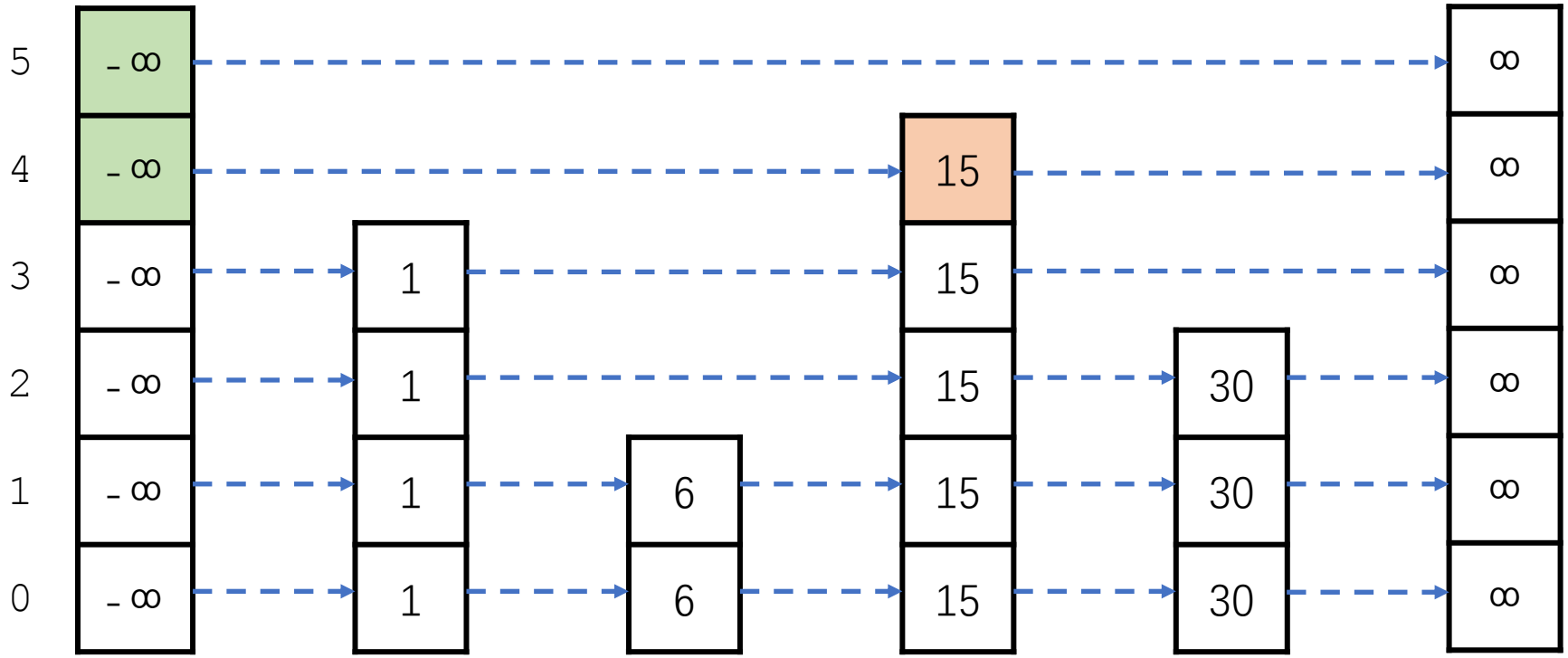
跳跃表（Skiplist）-查找操作

查找 15



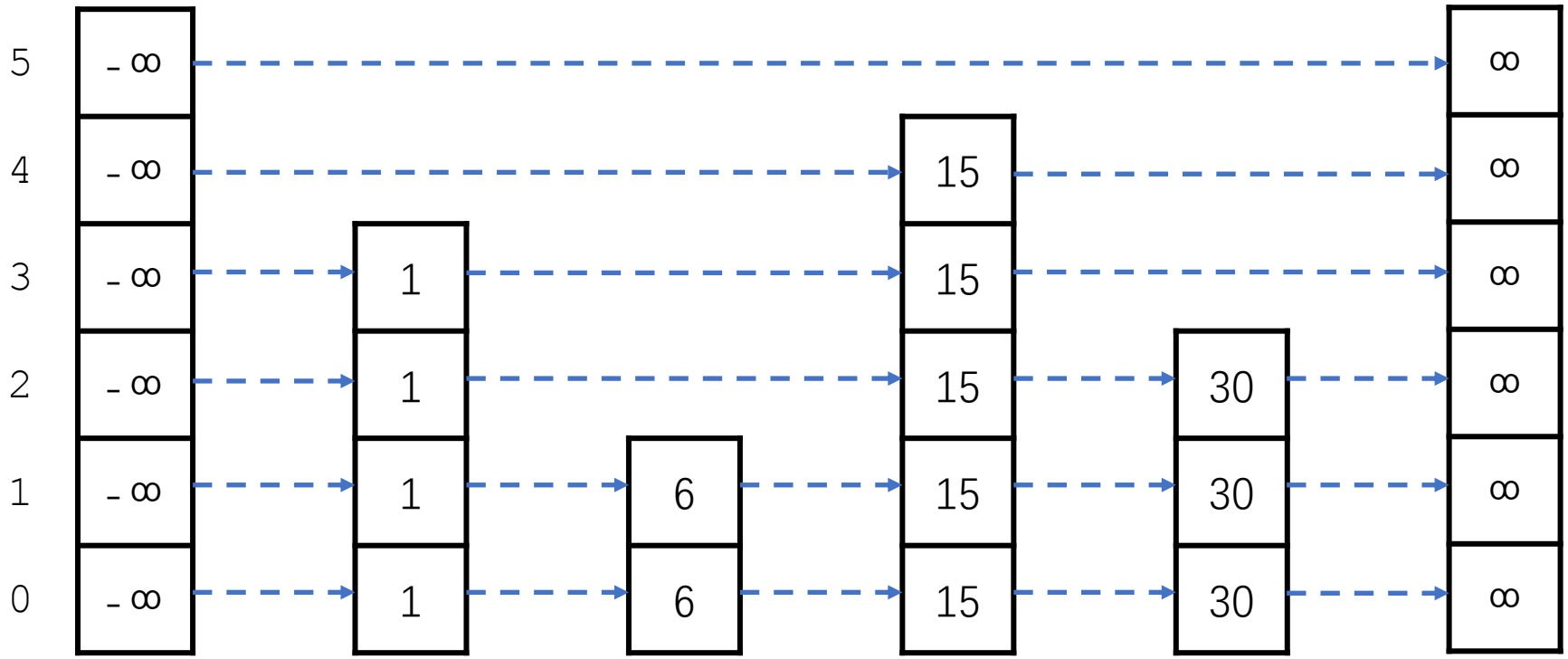
跳跃表（Skiplist）-查找操作

查找 15



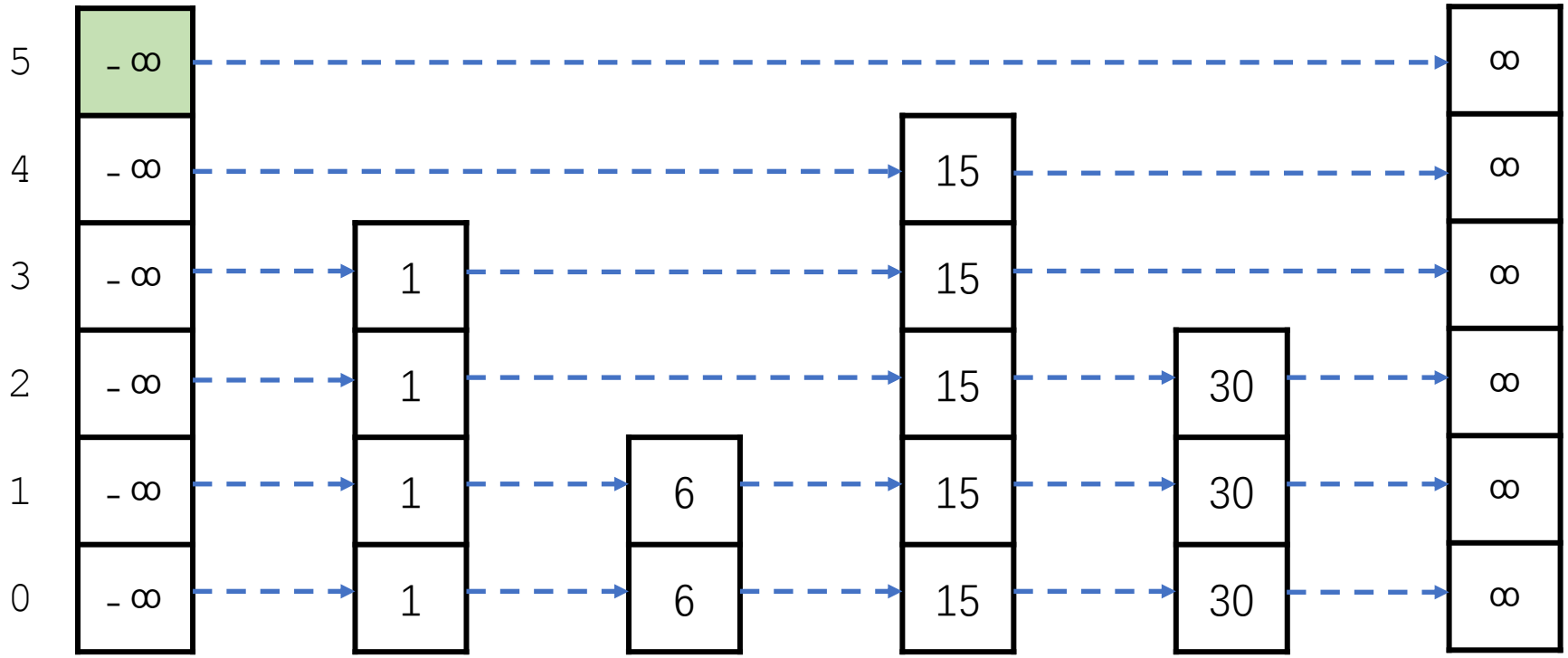
跳跃表（Skiplist）-查找操作

查找 30



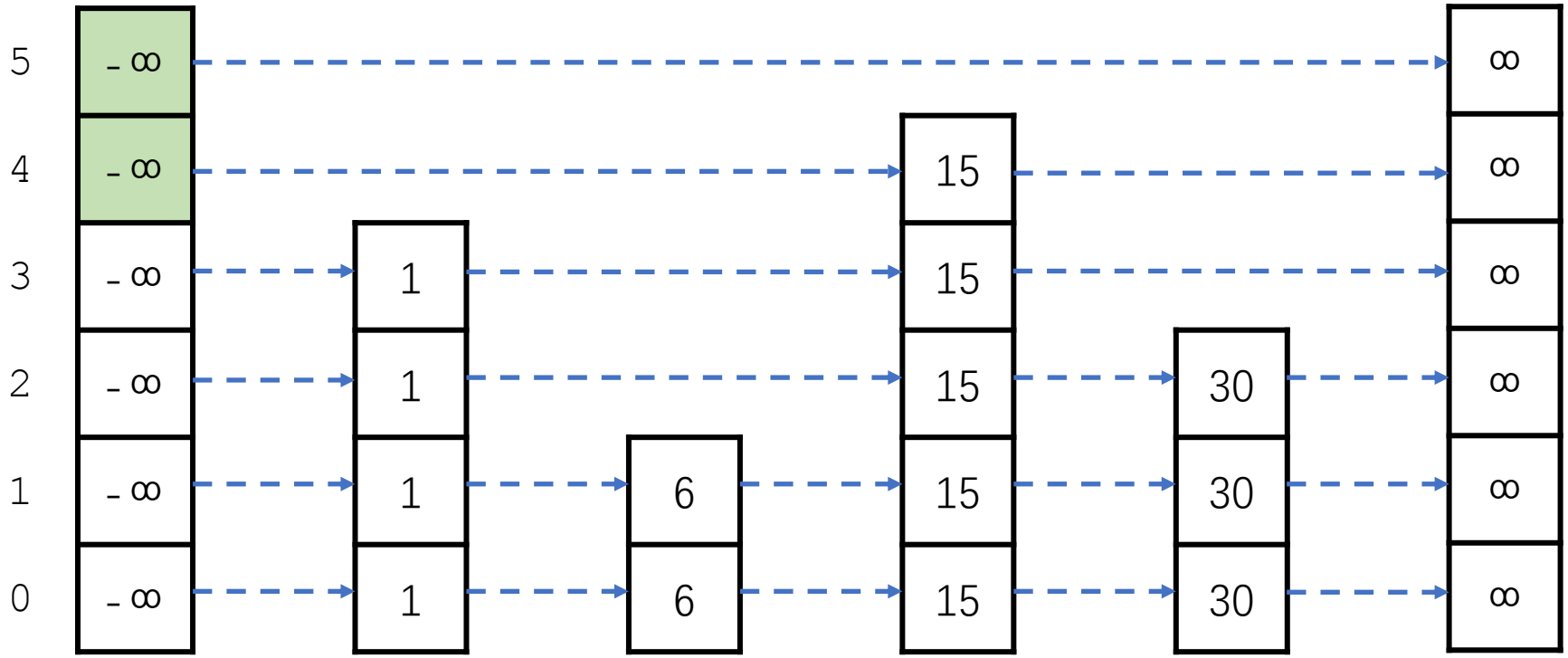
跳跃表（Skiplist）-查找操作

查找 30



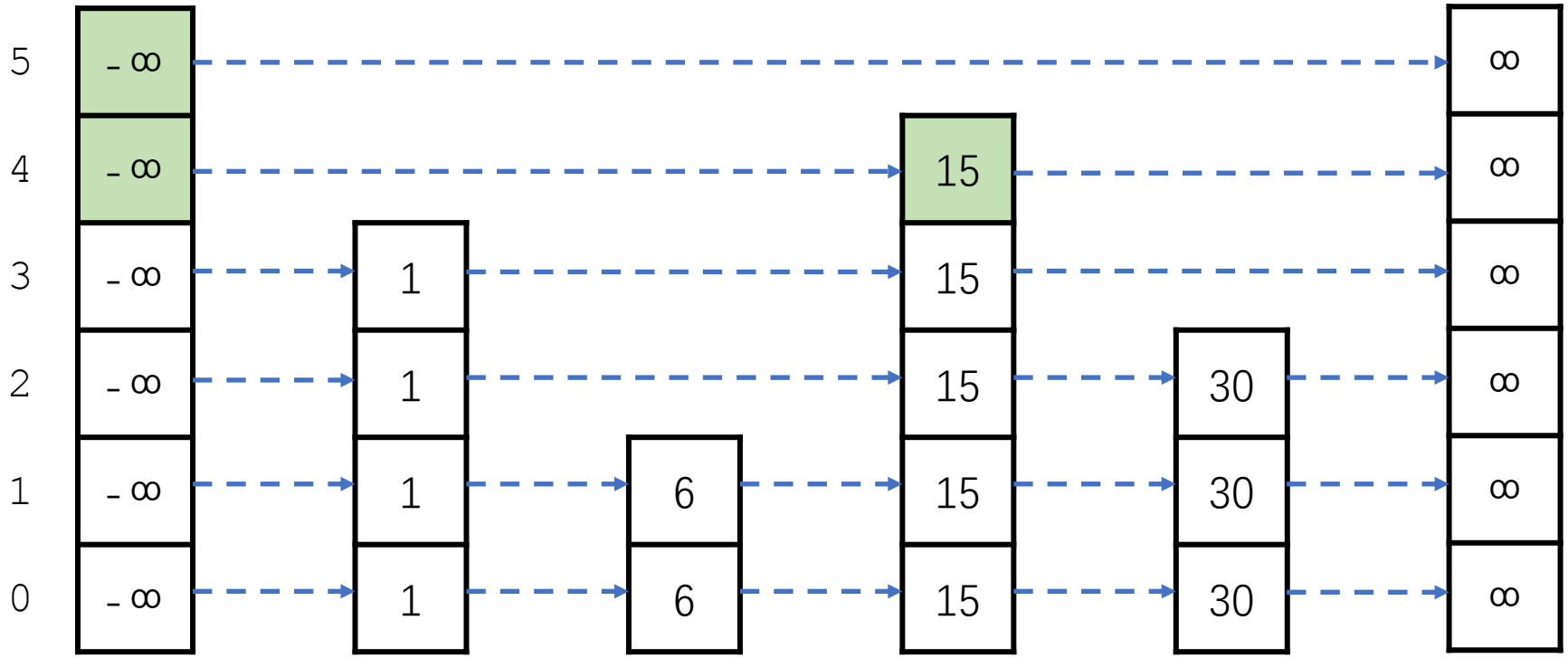
跳跃表（Skiplist）-查找操作

查找 30



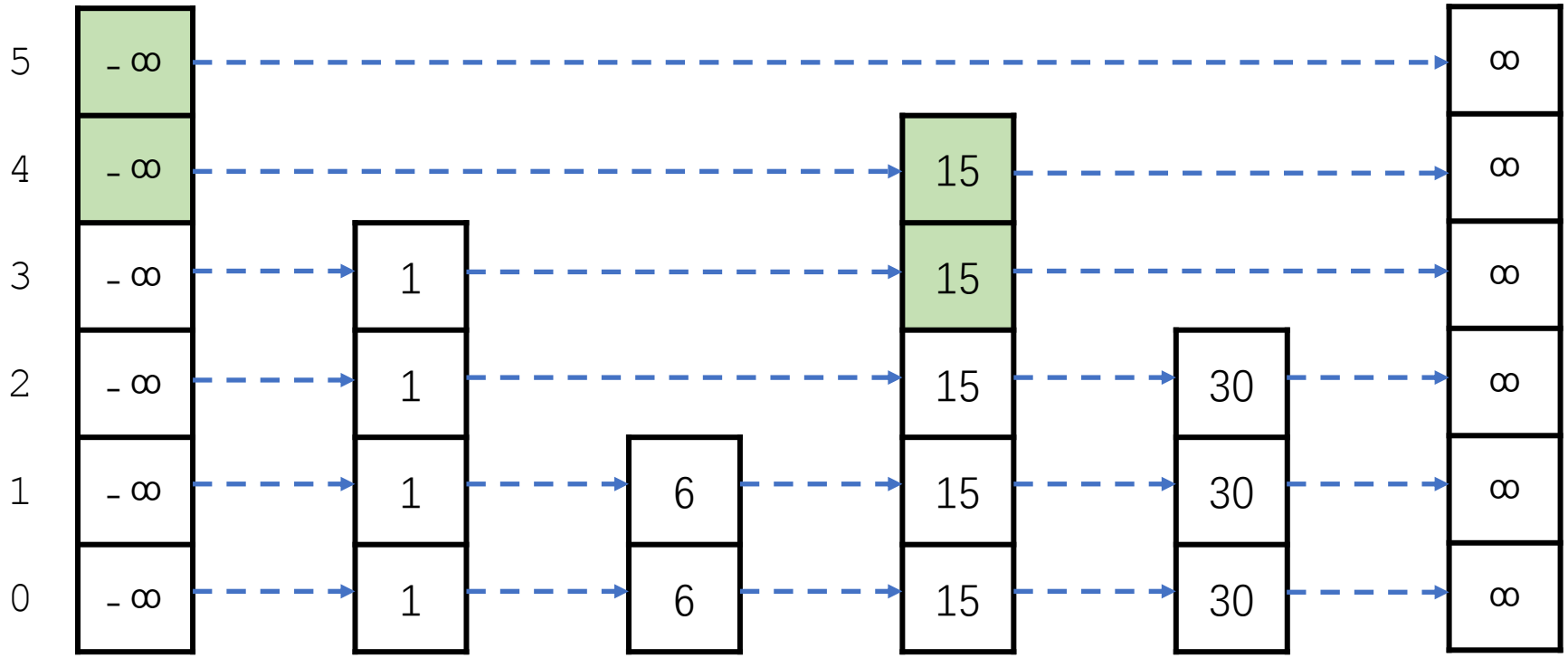
跳跃表（Skiplist）-查找操作

查找 30



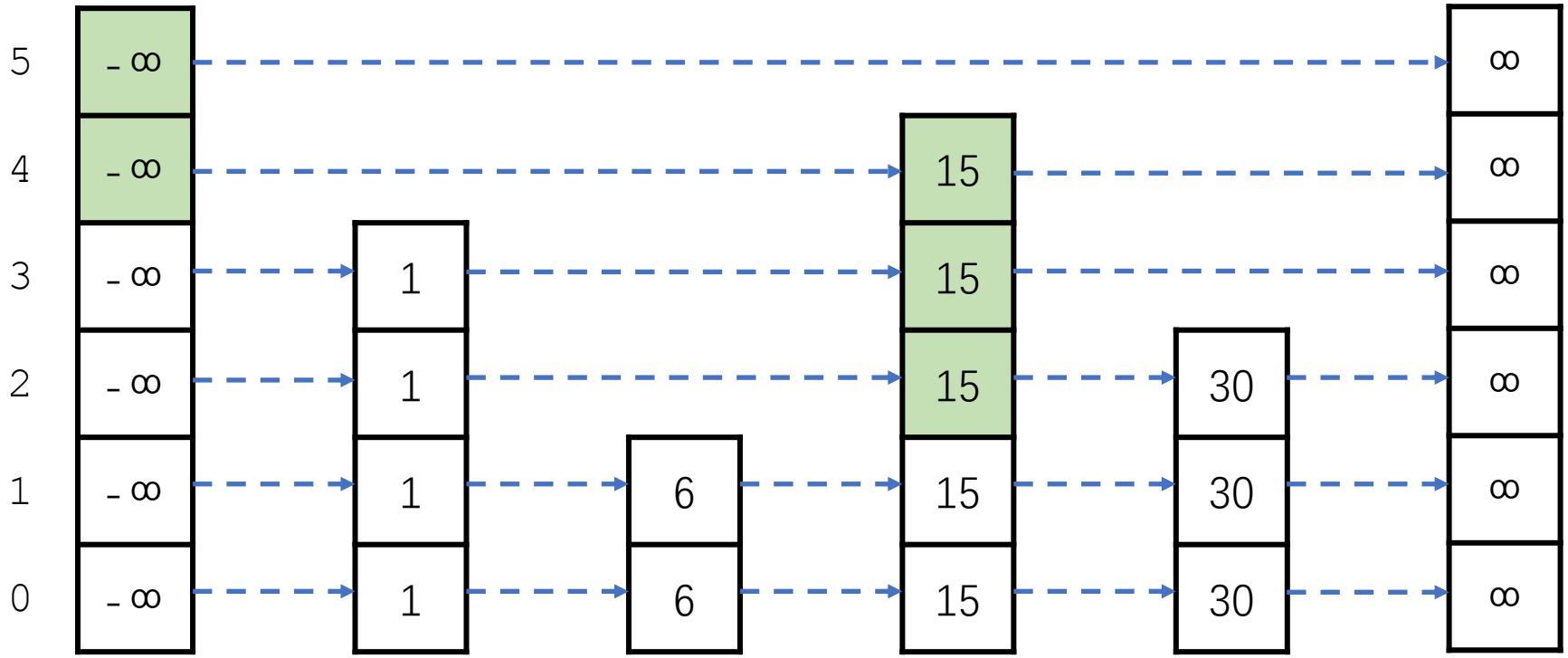
跳跃表（Skiplist）-查找操作

查找 30



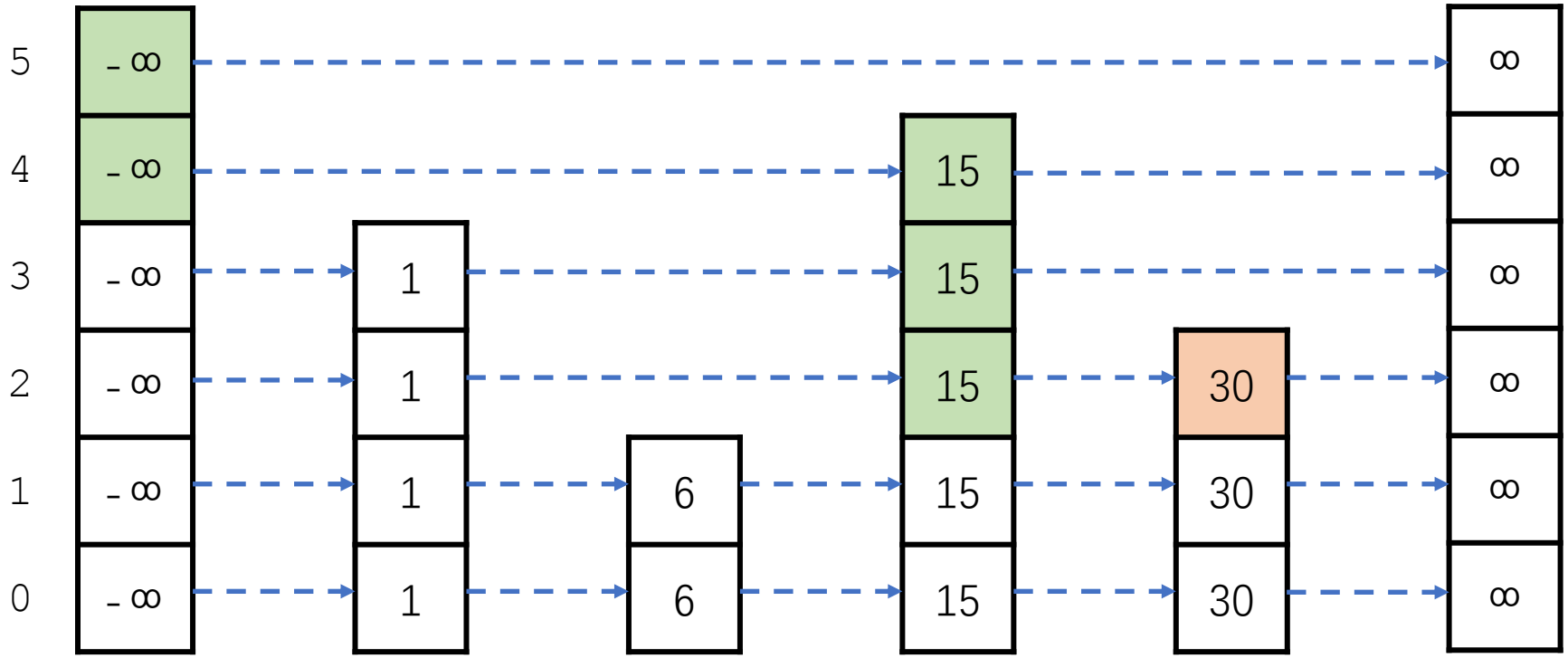
跳跃表（Skiplist）-查找操作

查找 30



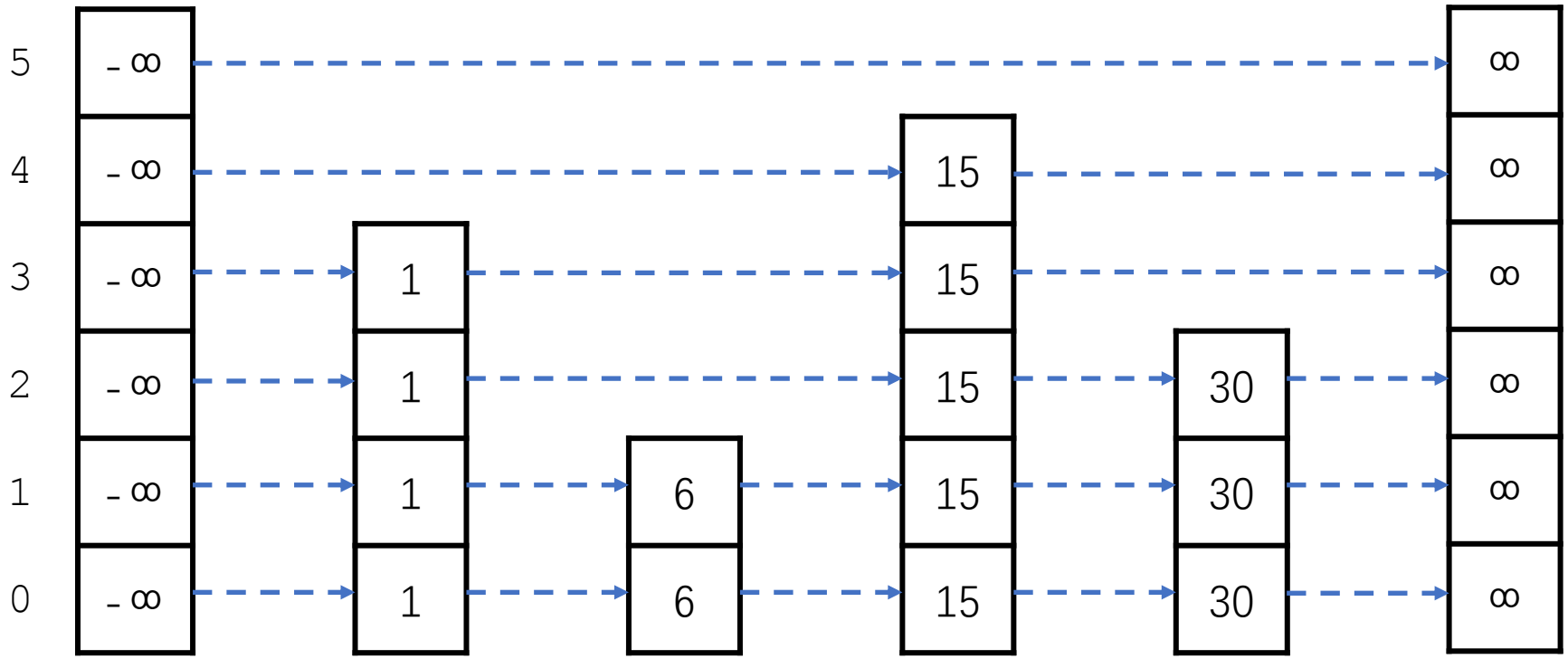
跳跃表（Skiplist）-查找操作

查找 30



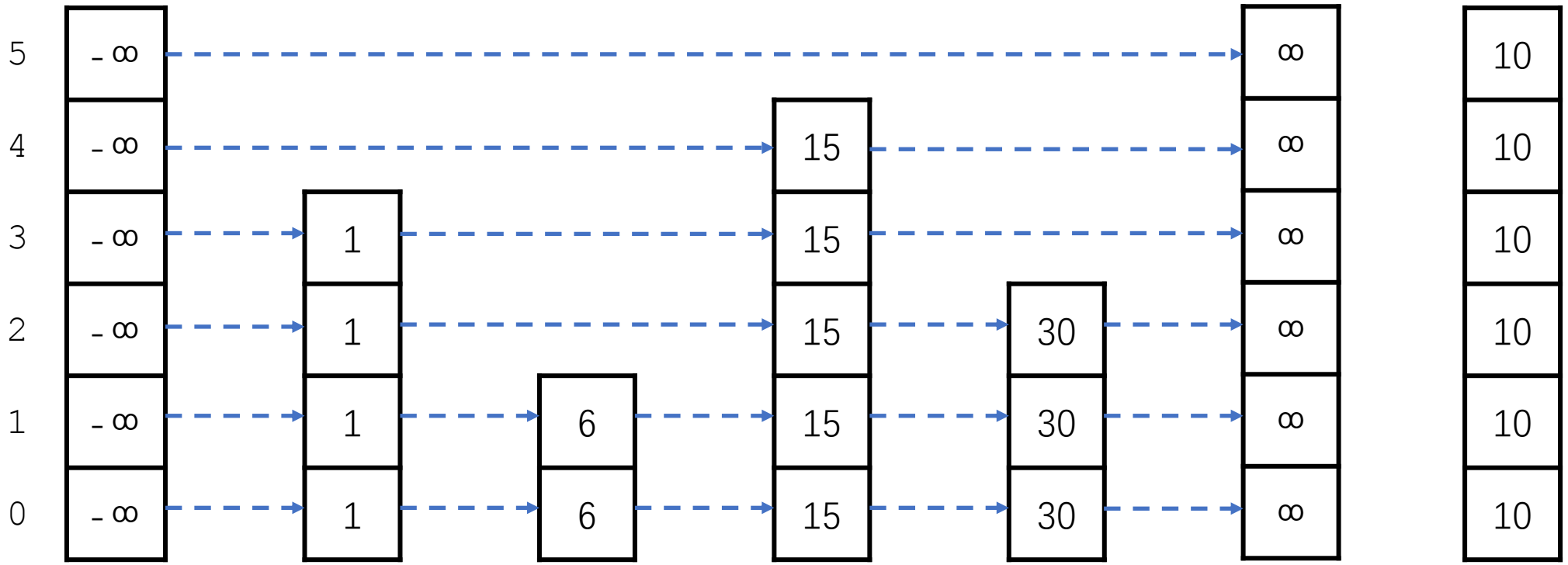
跳跃表（Skiplist）-插入操作

插入 10



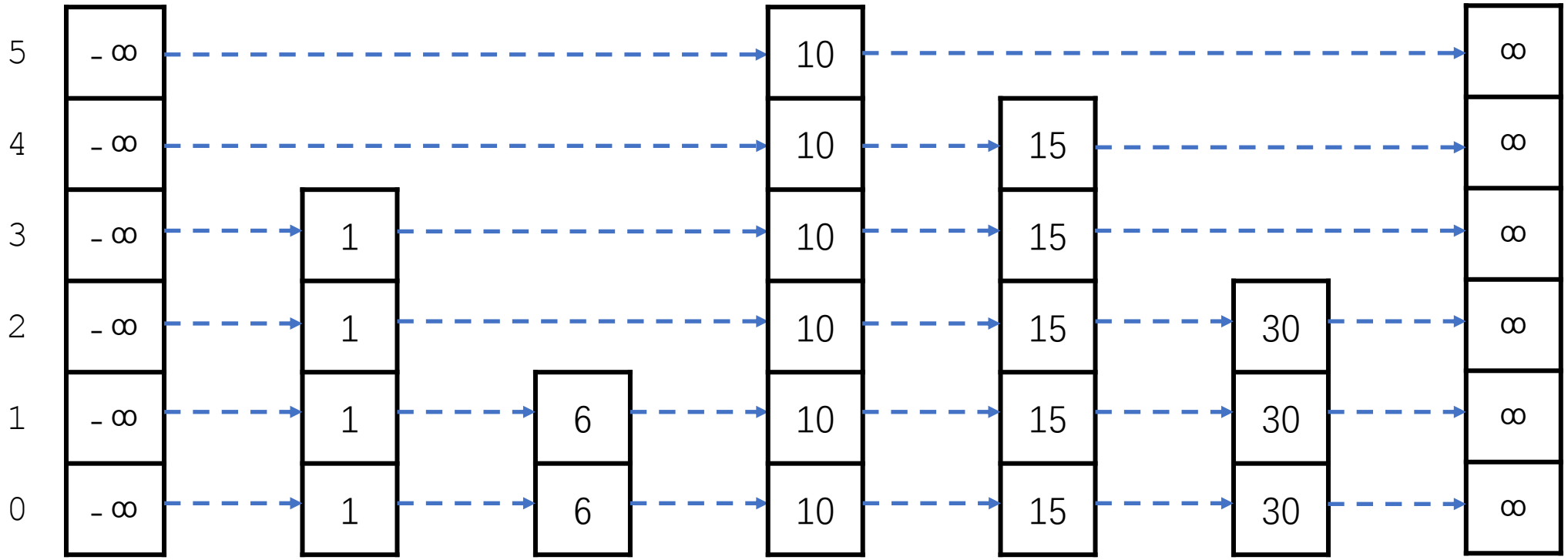
跳跃表（Skiplist）-插入操作

插入 10



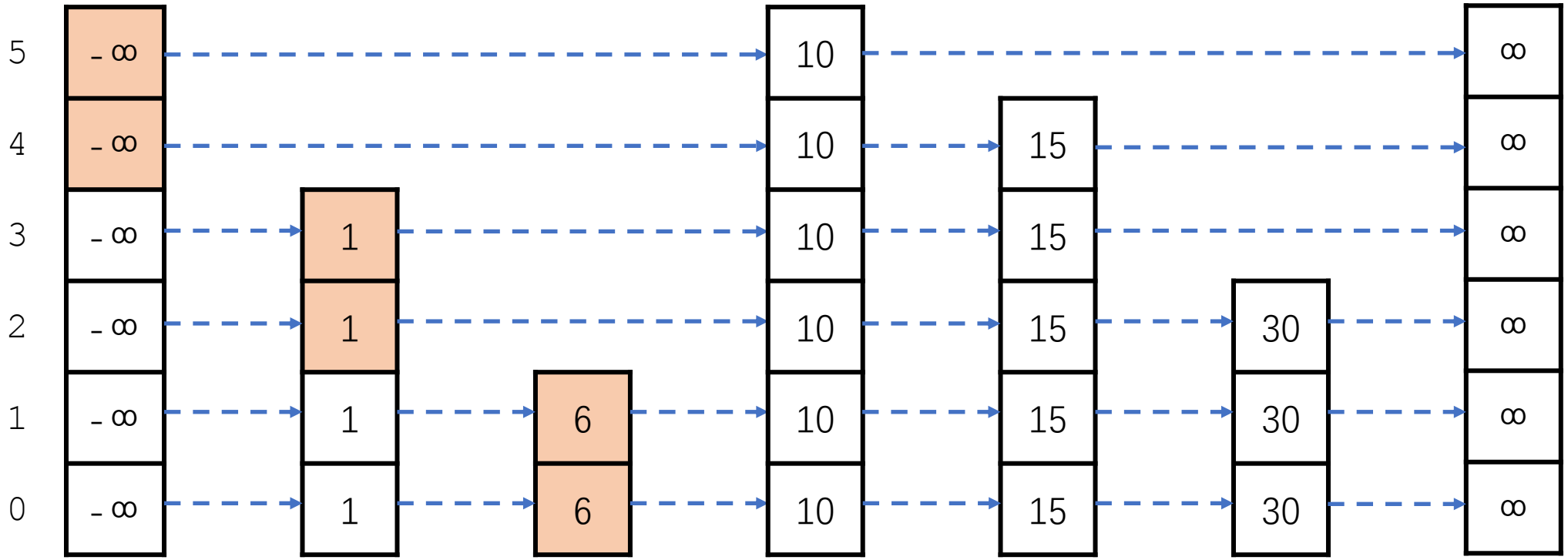
跳跃表（Skiplist）-插入操作

插入 10



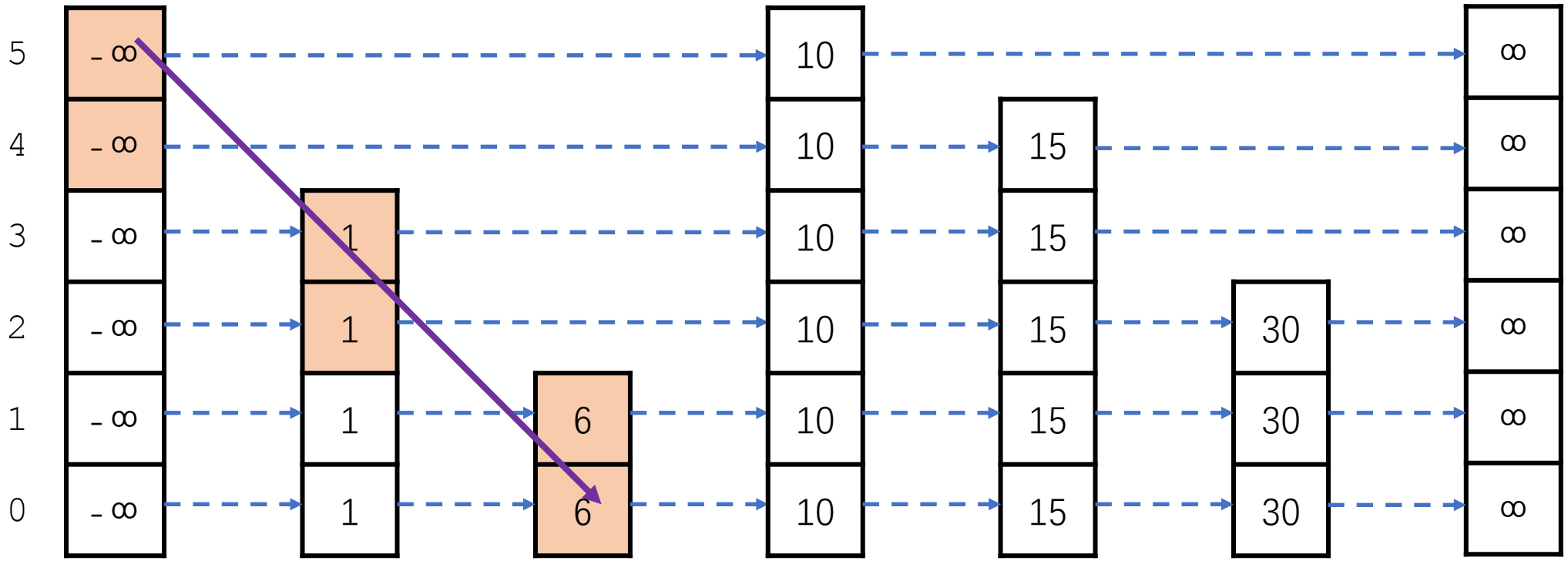
跳跃表（Skiplist）-插入操作

插入 10



跳跃表（Skiplist）-插入操作

插入 10

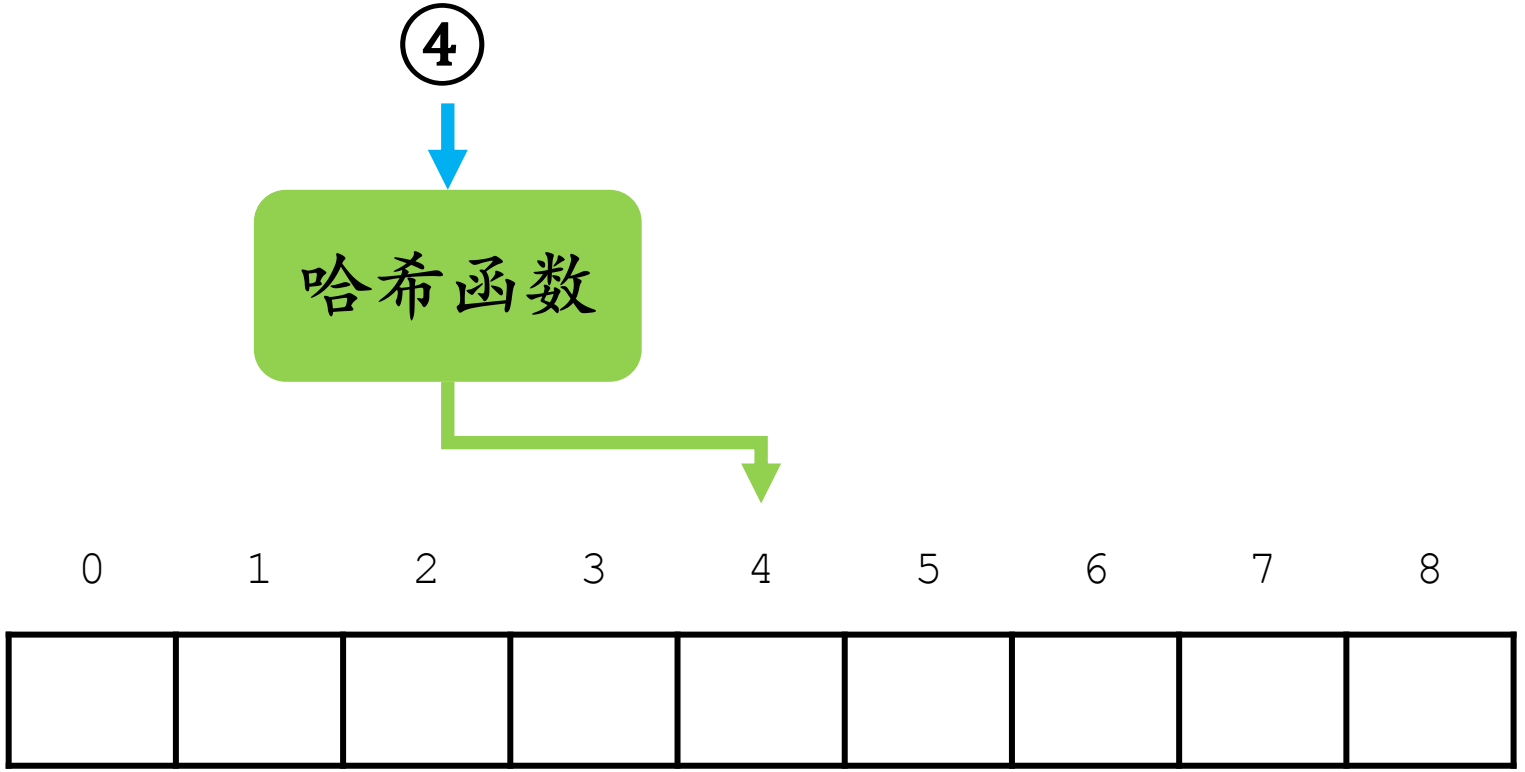



1. vim
vim ⌘1
bash ⌘2
bash ⌘3
39 }
40
41 Node *insert_maintain(Node *root) {
42 if (!hasRedChild(root)) return root;
43 if (root->lchild->color == RED && root->rchild->color == RED) {
44 if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45 root->color = RED;
46 root->lchild->color = root->rchild->color = BLACK;
47 return root;
48 }
49 if (root->lchild->color == RED) {
50 if (!hasRedChild(root->lchild)) return root;
51
52
53 } else {
54 if (!hasRedChild(root->rchild)) return root;
55
56 }
57
58 }
59
60
61 Node *__insert(Node *root, int key) {
62 if (root == NIL) return getNewNode(key);
<-6班 资 料 /X.现 场 撸 代 码 /15.RBT.cpp [FORMAT=unix] [TYPE=CPP] [POS=54,30][62%] 21/09/19 - 20:21

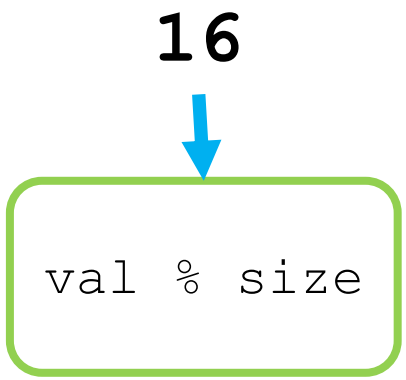
跳跃表：代码演示

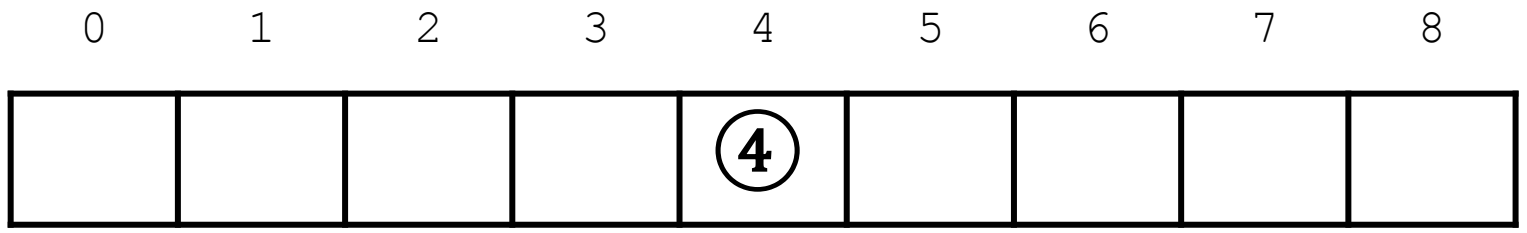
三. 哈希表与布隆过滤器

哈希表

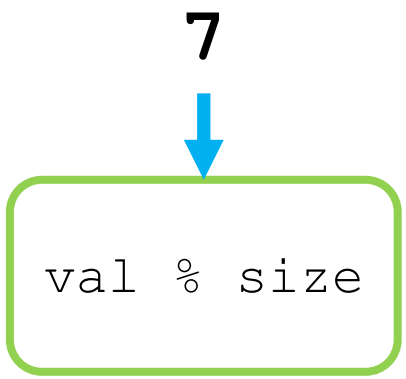


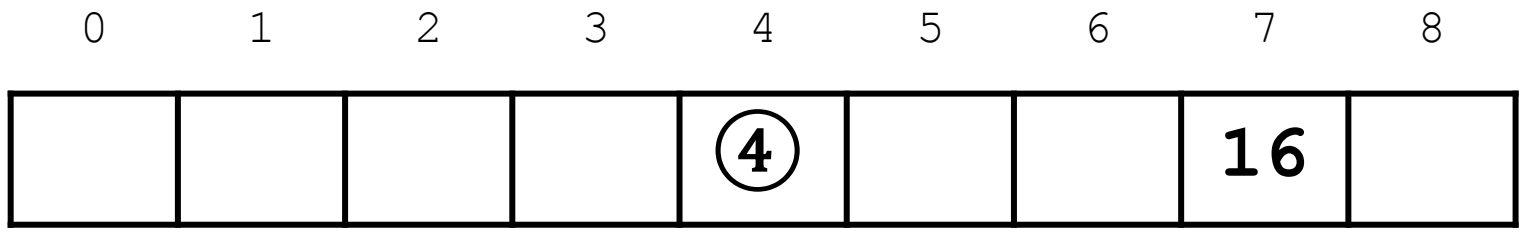
哈希表



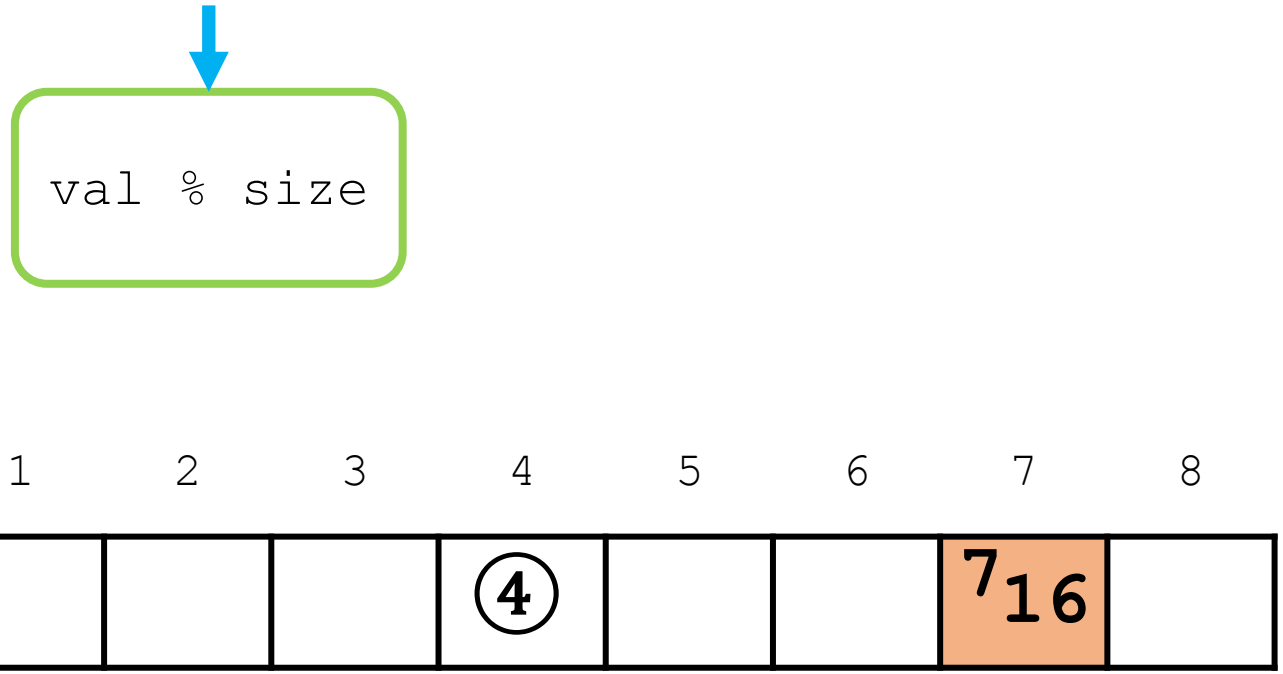


哈希表

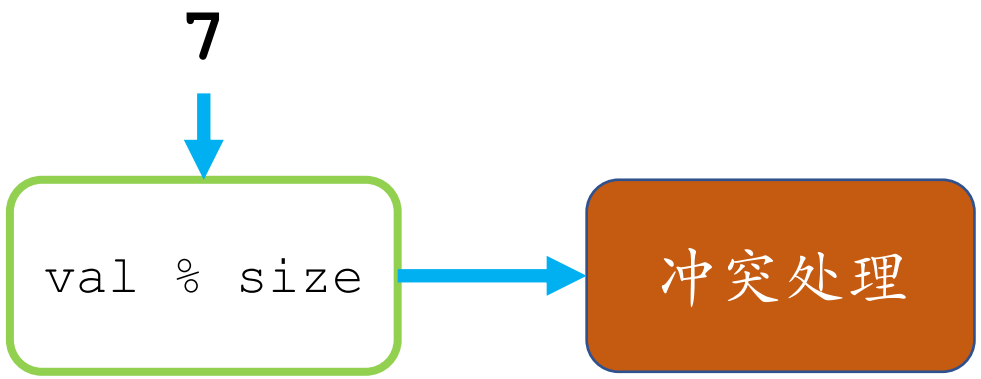


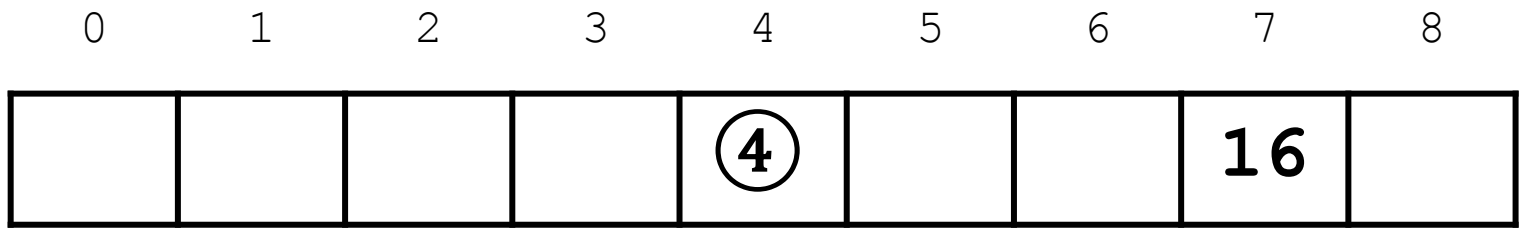


哈希表

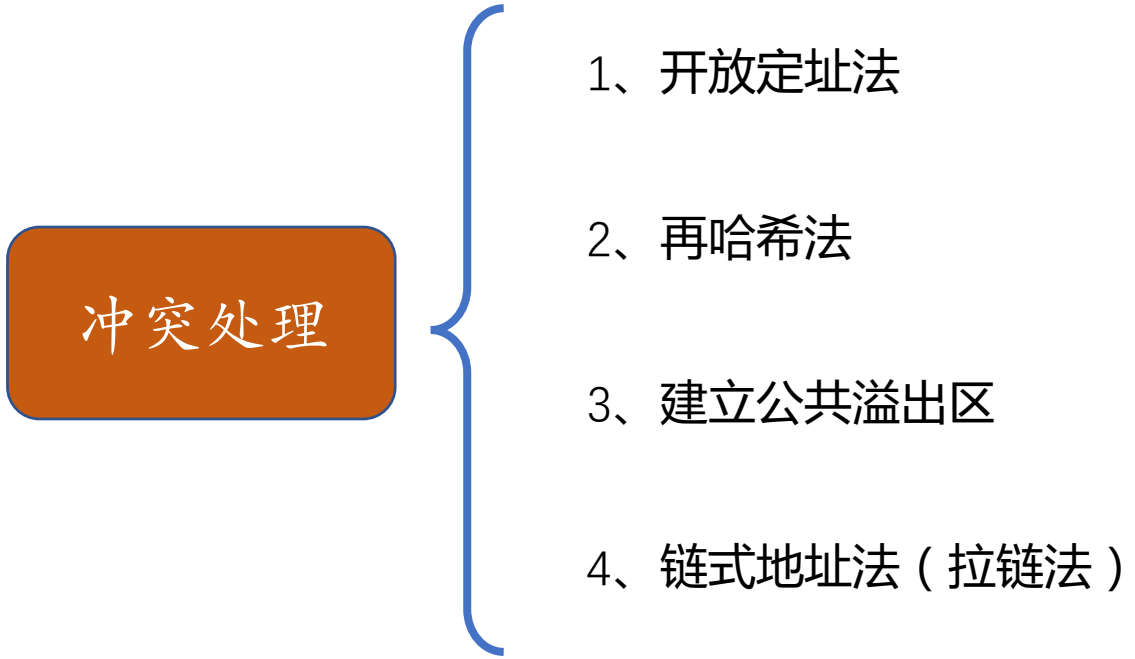


哈希表



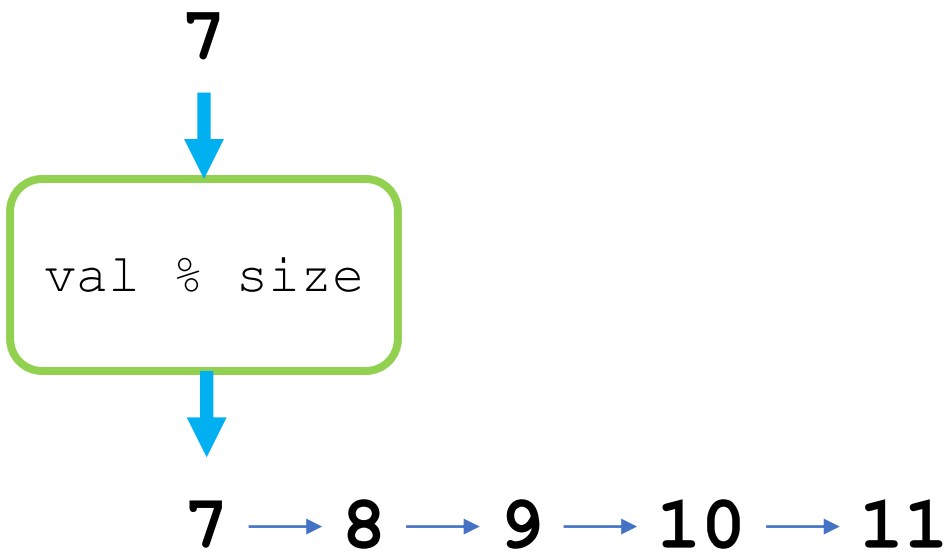


哈希表-冲突处理方法



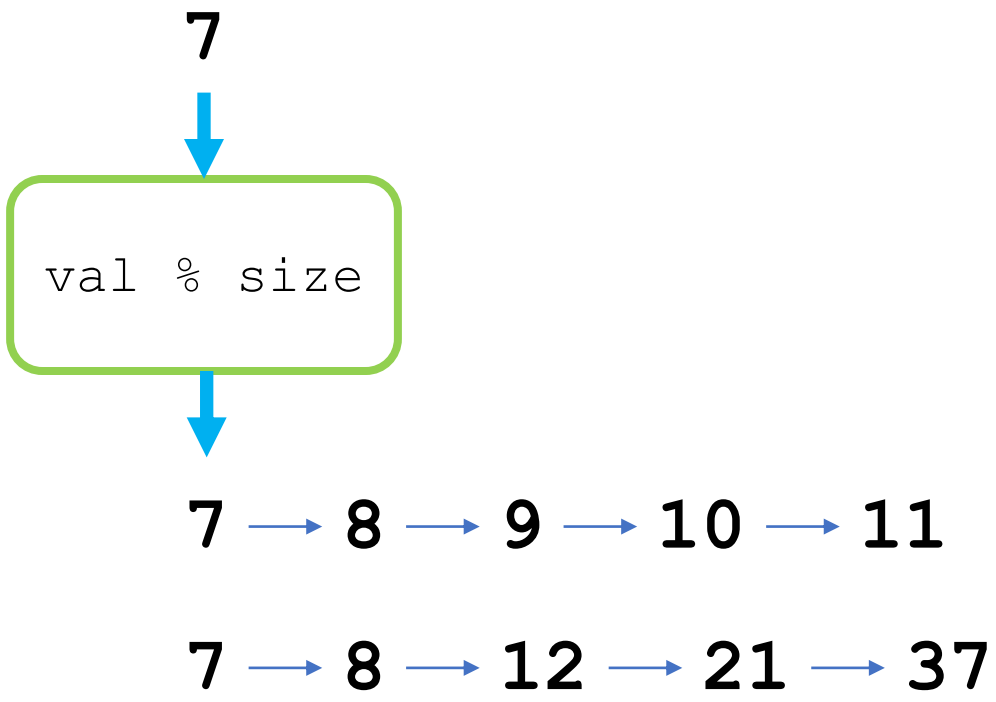
哈希表-冲突处理方法

1、开放定址法

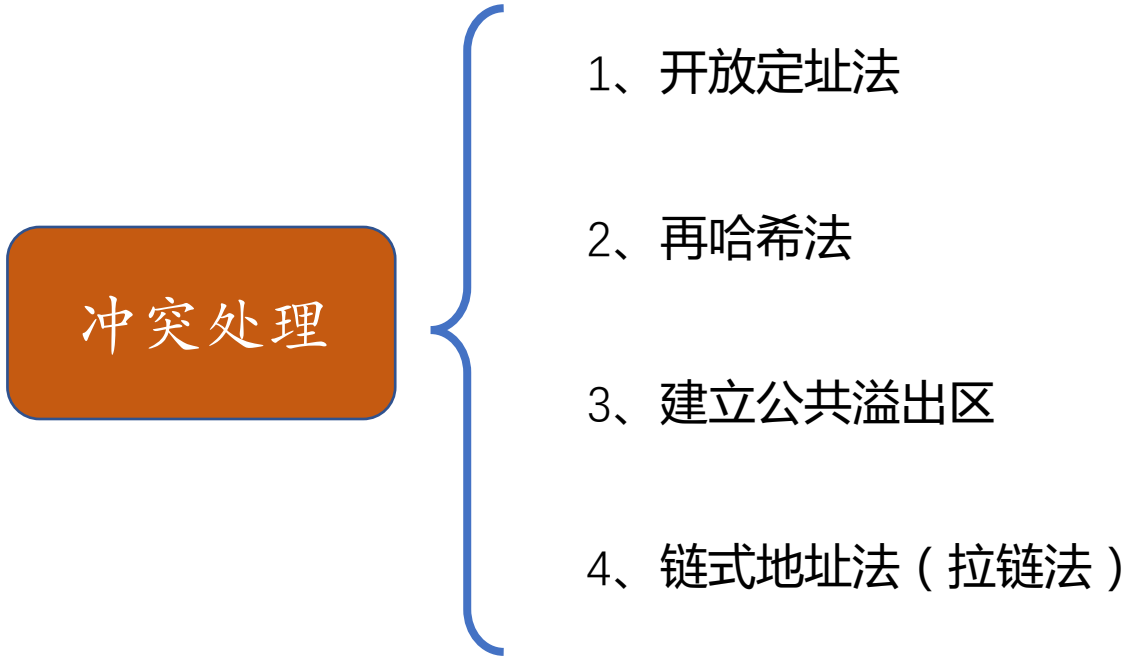


哈希表-冲突处理方法

1、开放定址法

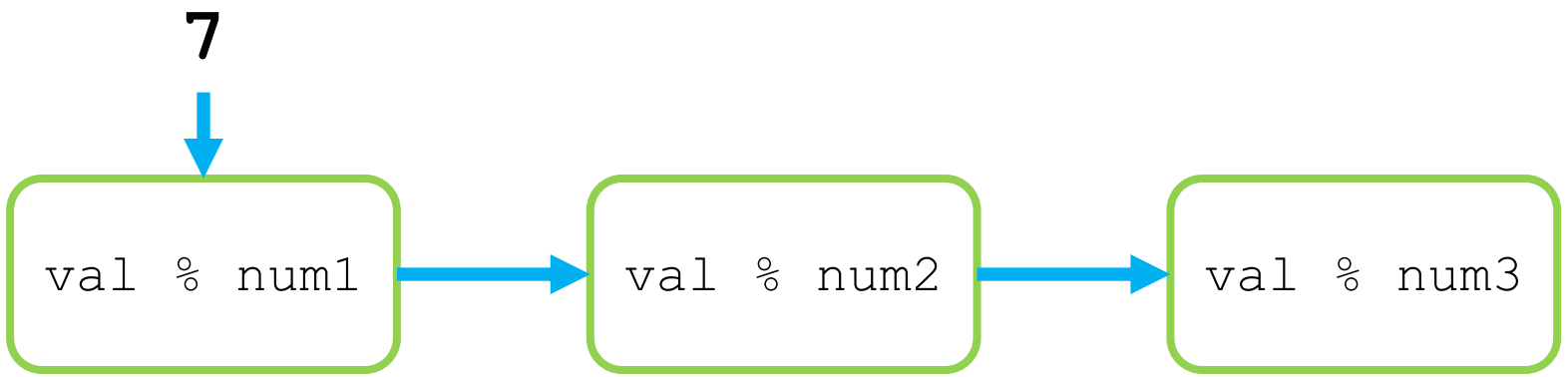


哈希表-冲突处理方法

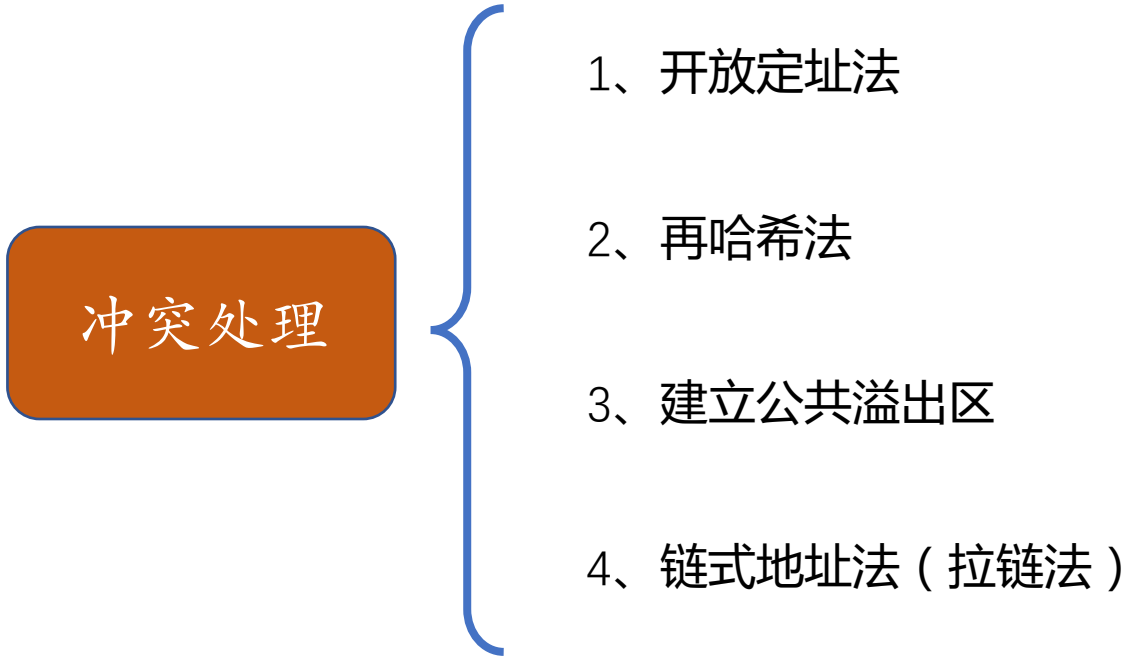


哈希表-冲突处理方法

2、再哈希法

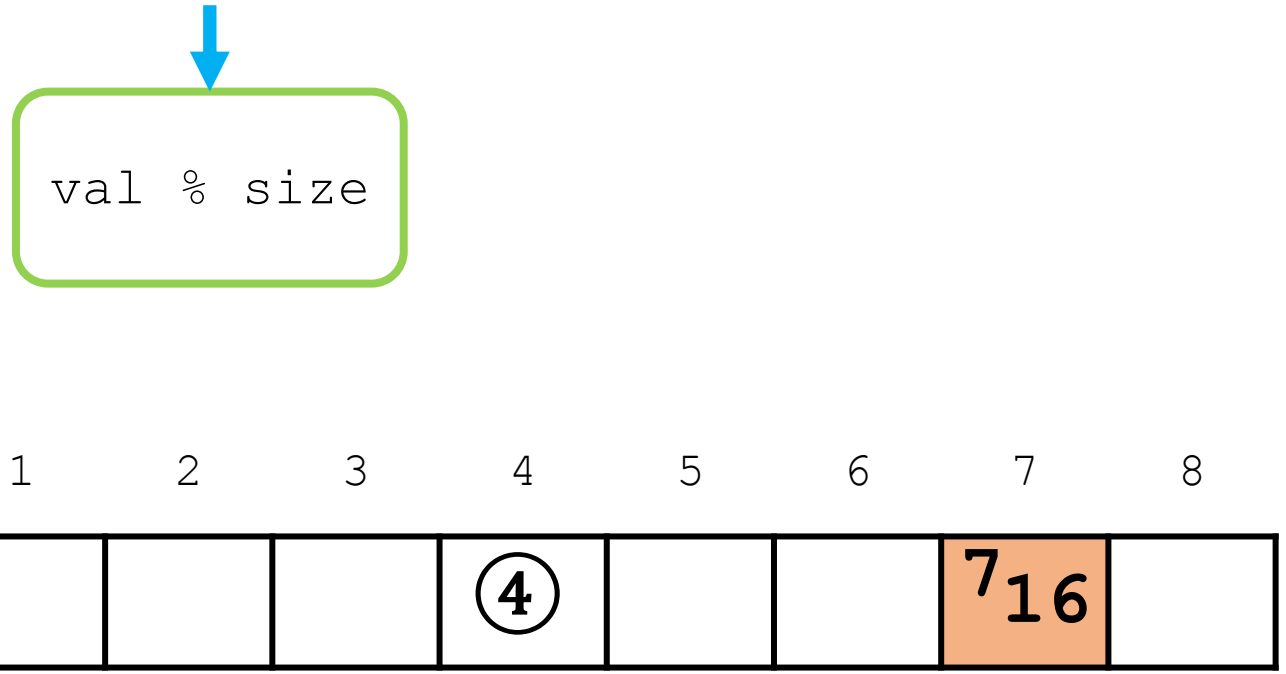


哈希表-冲突处理方法



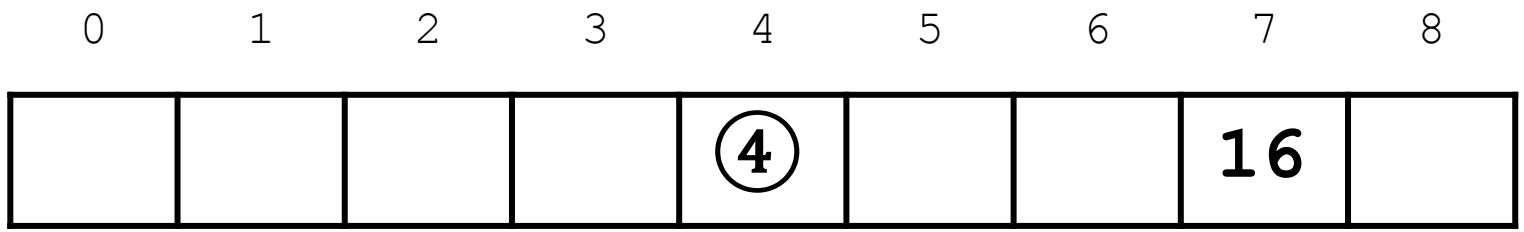
哈希表-冲突处理方法

3、建立公共溢出区

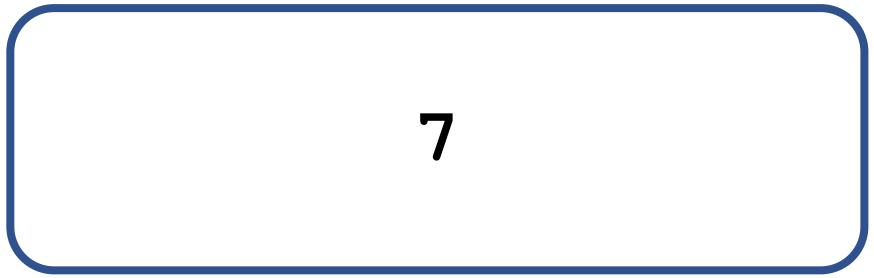


哈希表-冲突处理方法

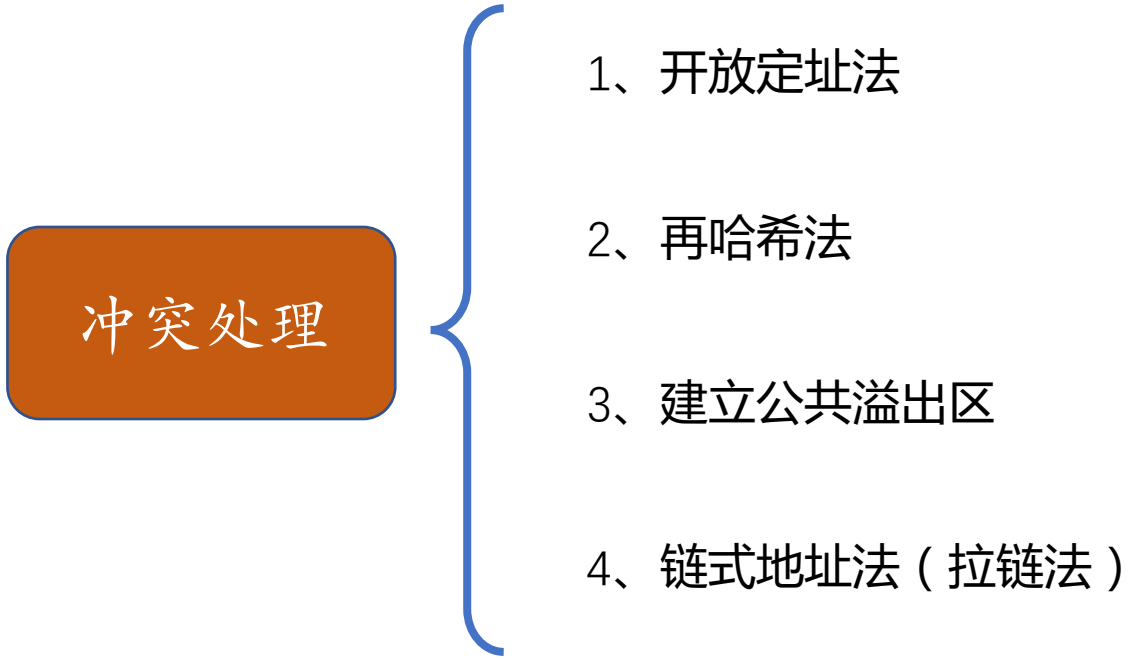
3、建立公共溢出区



溢出缓冲区

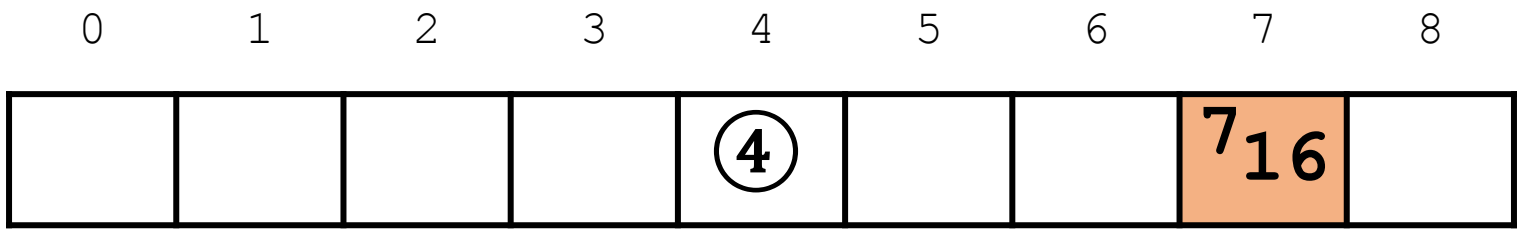


哈希表-冲突处理方法



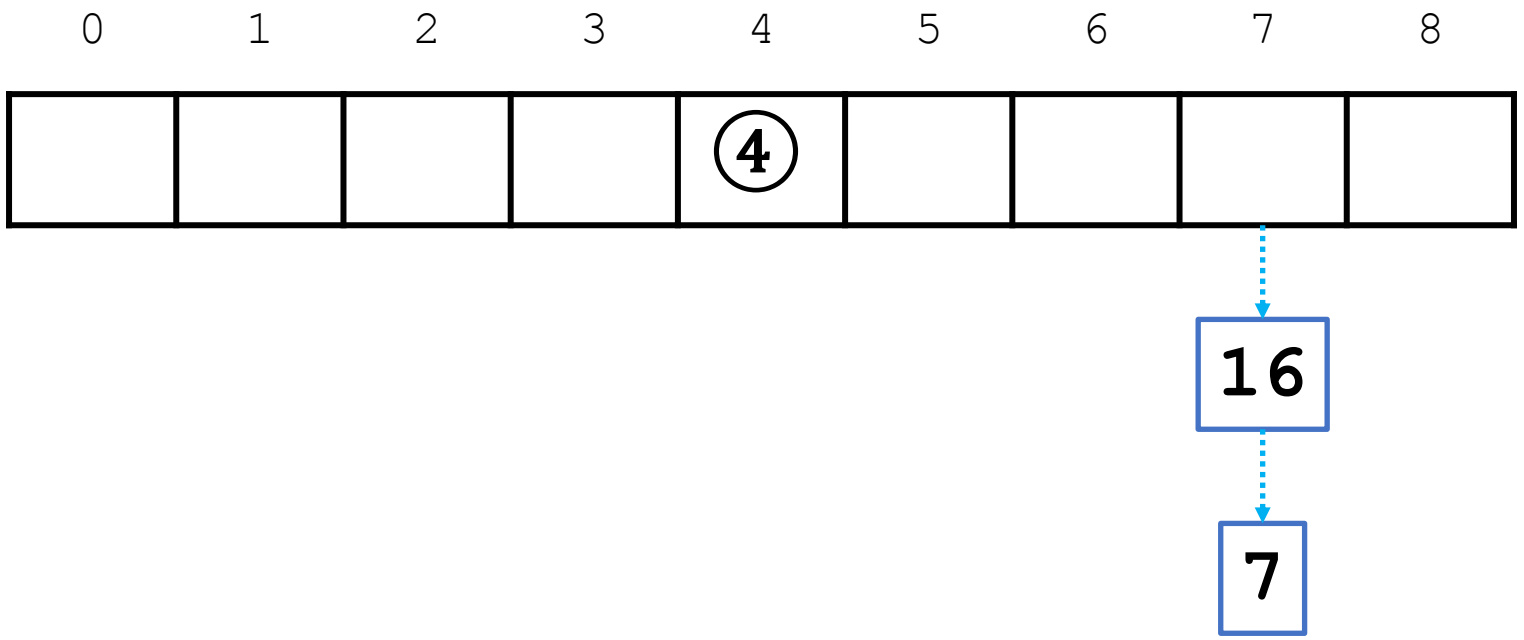
哈希表-冲突处理方法

4、链式地址法（拉链法）



哈希表-冲突处理方法

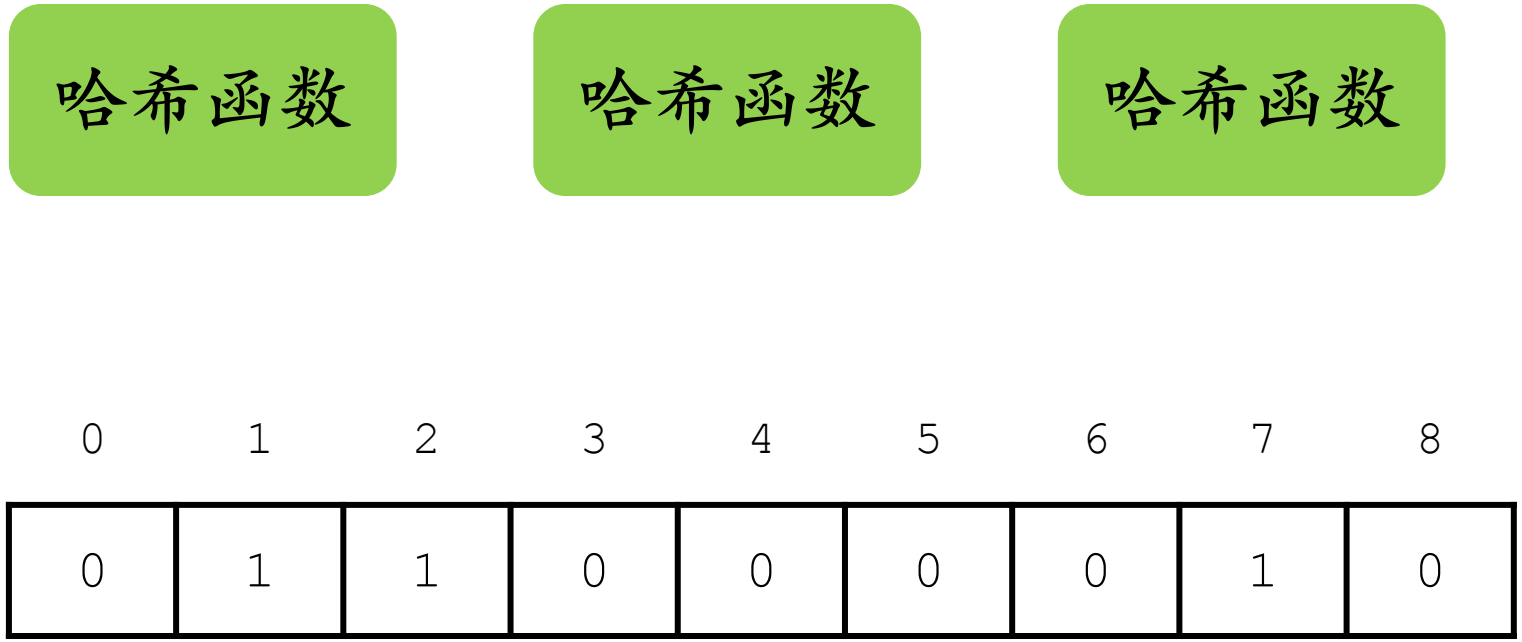
4、链式地址法（拉链法）



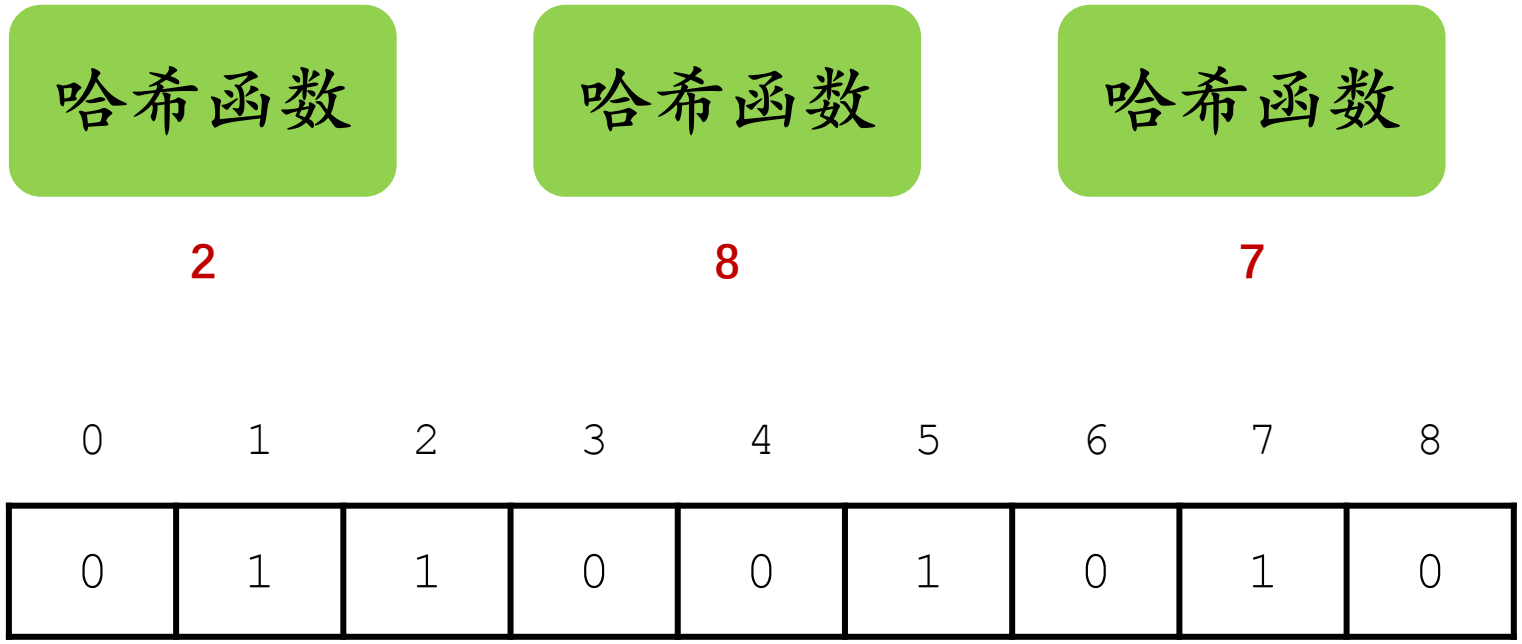
传统哈希表，存储空间与元素数量有关

布隆过滤器，存储空间与元素数量无关

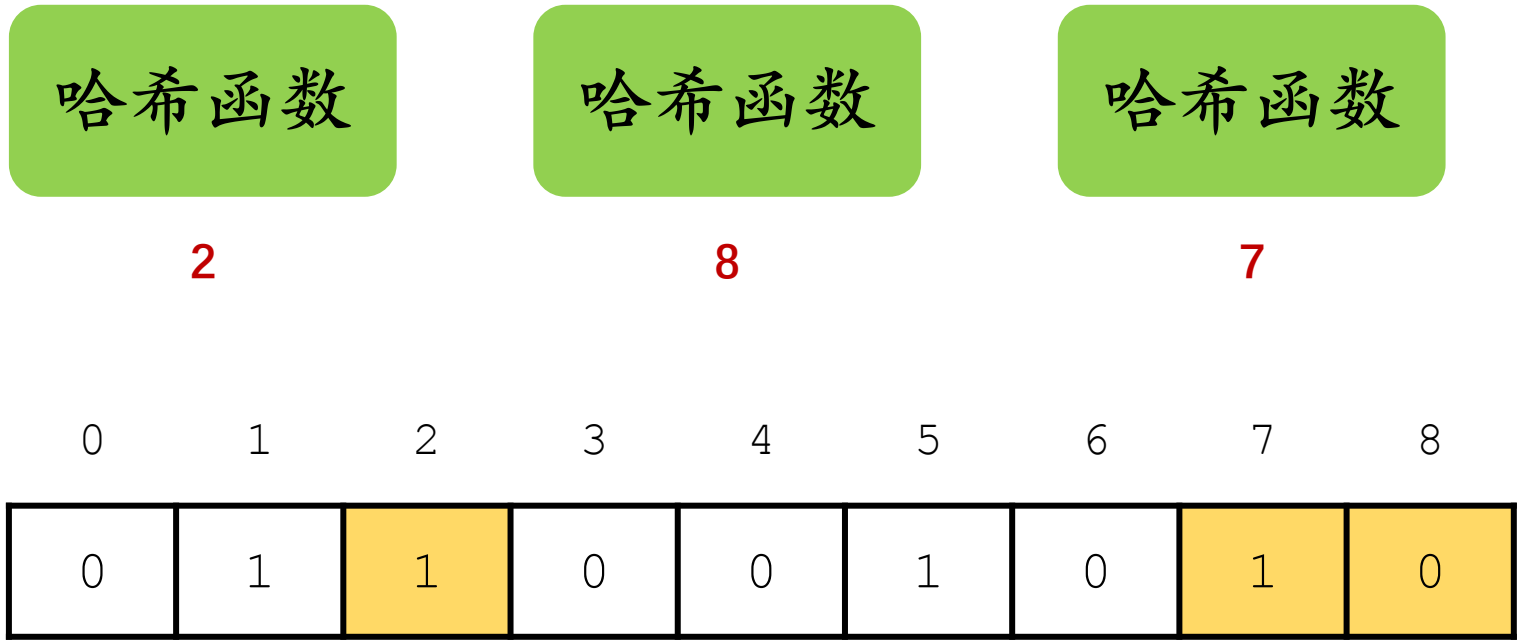
布隆过滤器



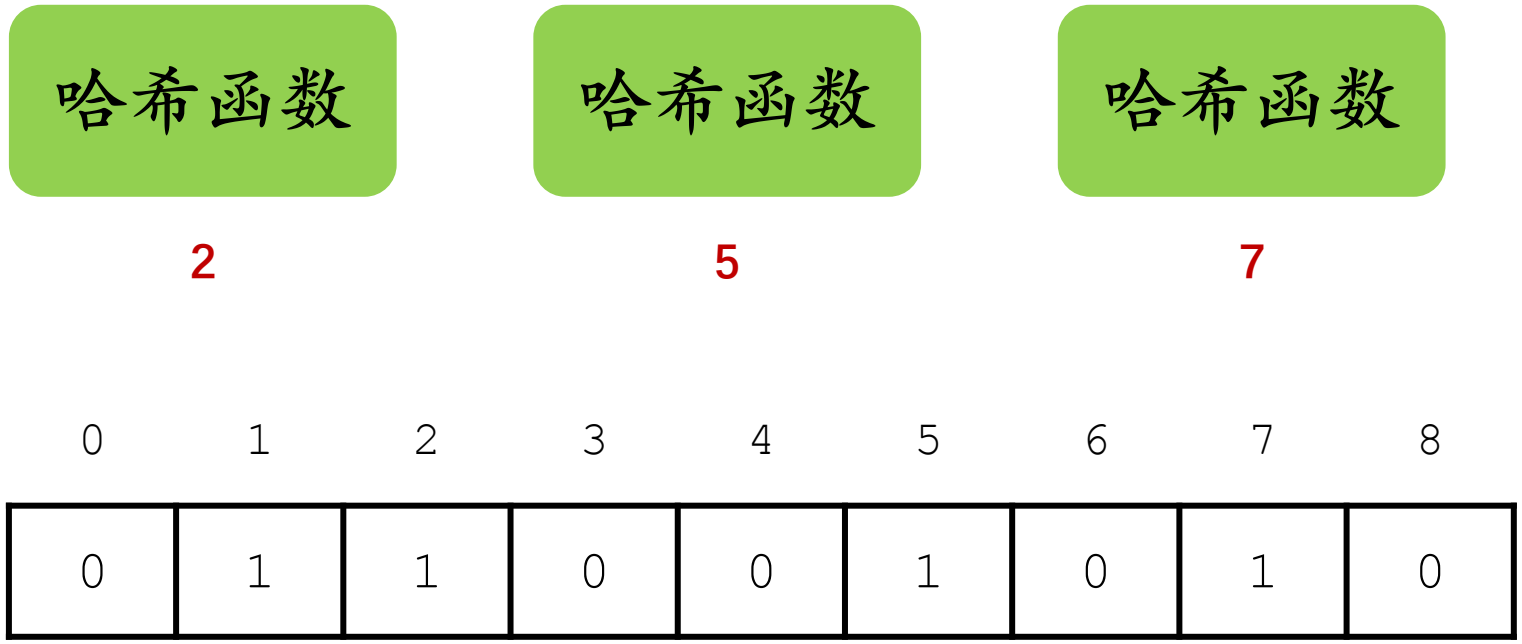
布隆过滤器



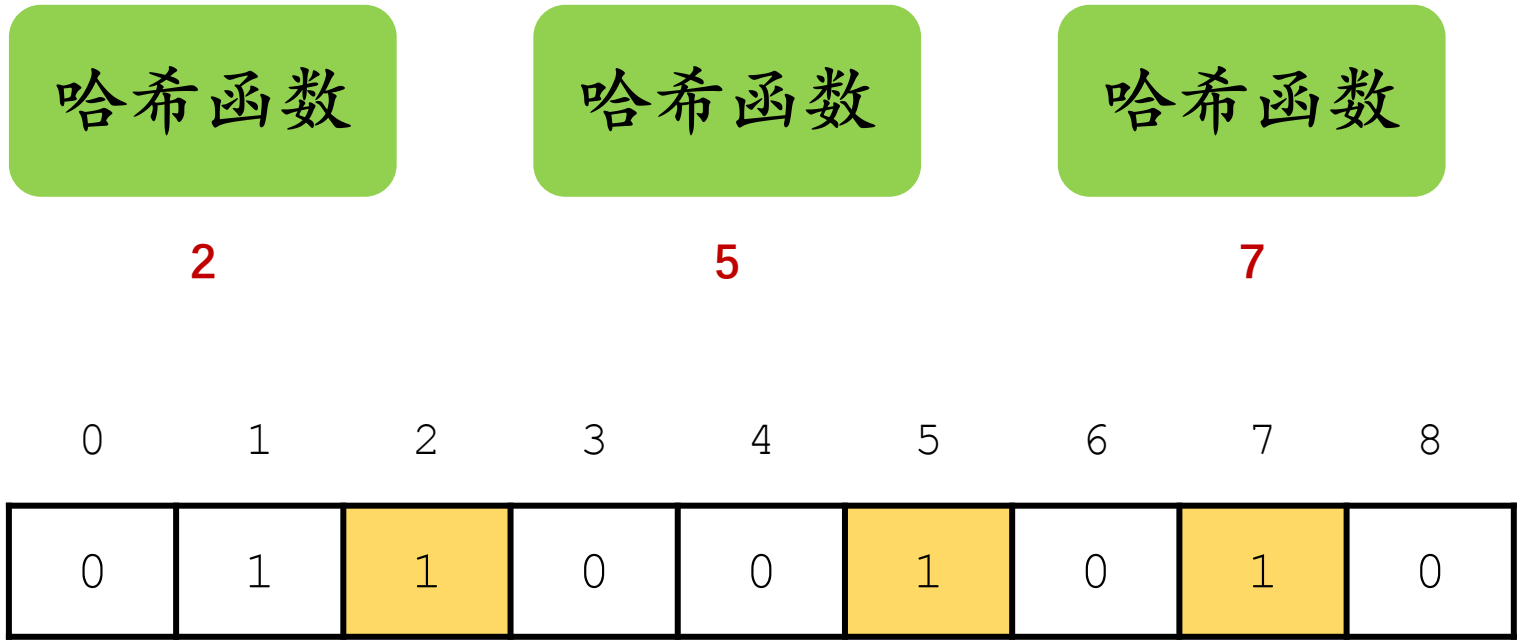
布隆过滤器



布隆过滤器

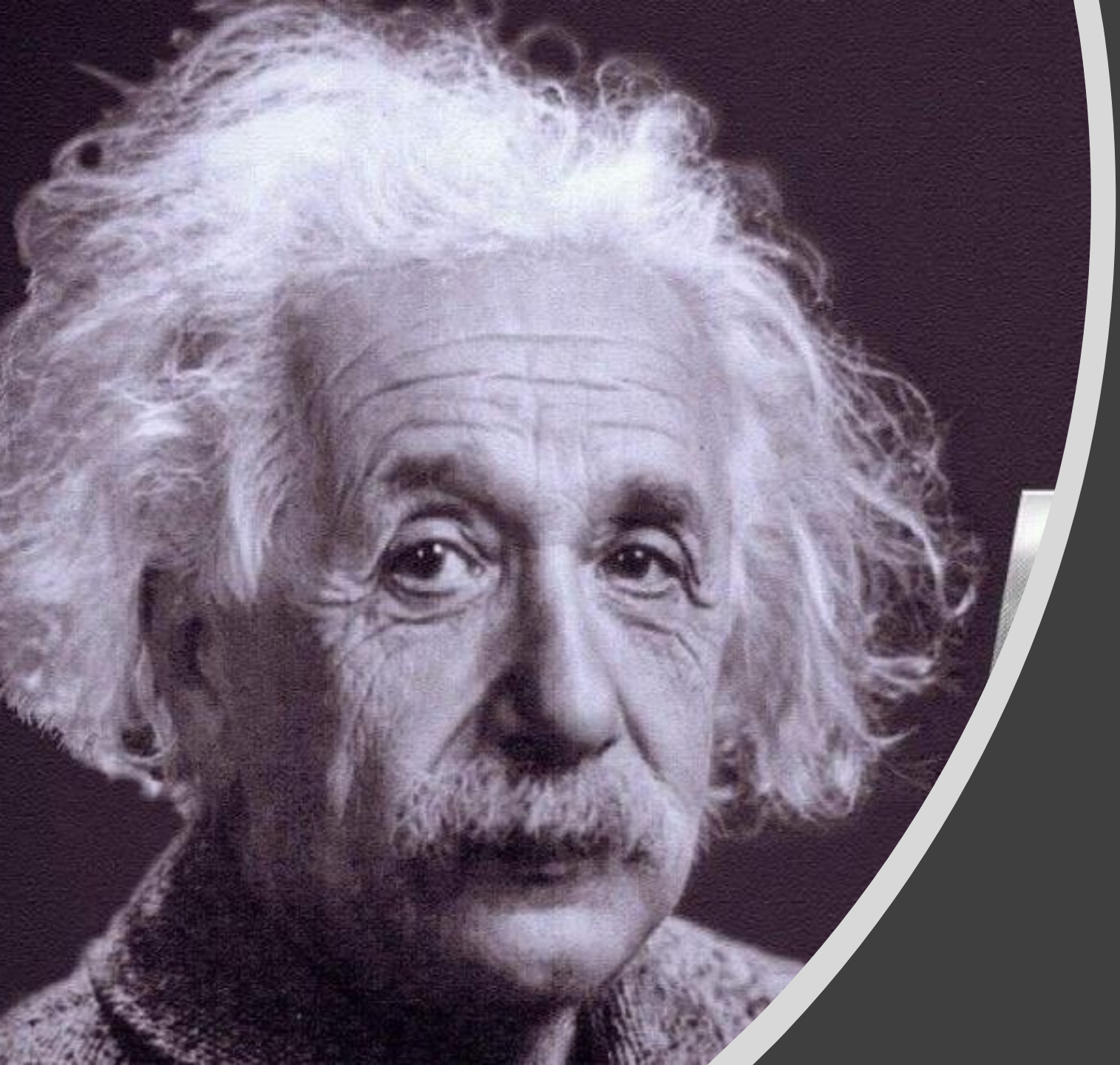


布隆过滤器




1. vim
vim ⌘1
bash ⌘2
bash ⌘3
39 }
40
41 Node *insert_maintain(Node *root) {
42 if (!hasRedChild(root)) return root;
43 if (root->lchild->color == RED && root->rchild->color == RED) {
44 if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45 root->color = RED;
46 root->lchild->color = root->rchild->color = BLACK;
47 return root;
48 }
49 if (root->lchild->color == RED) {
50 if (!hasRedChild(root->lchild)) return root;
51
52
53 } else {
54 if (!hasRedChild(root->rchild)) return root;
55
56 }
57
58 }
59
60
61 Node *__insert(Node *root, int key) {
62 if (root == NIL) return getNewNode(key);
<-6班 资 料 /X.现 场 撸 代 码 /15.RBT.cpp [FORMAT=unix] [TYPE=CPP] [POS=54,30][62%] 21/09/19 - 20:21

哈希表：代码演示



为什么
会出一样的题目？