跳表：Redis中如何实现有序集合？

困难/跳表、映射表

学习目标

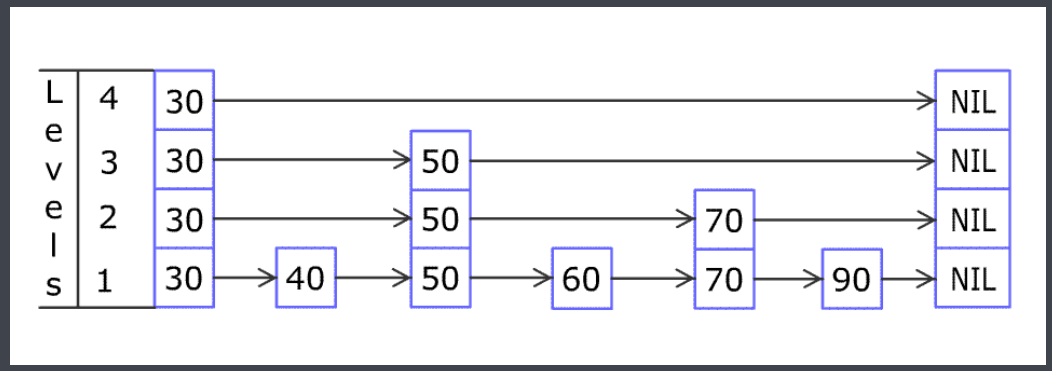
拉勾教育
—互联网人实战大学—

- 了解算法题的解题思路
- 跳表的结构和特点
- 跳表的操作流程

题目描述

不使用任何库函数，设计一个跳表。

跳表是在 O(log(n)) 时间内完成增加、删除、搜索操作的数据结构。跳表相比于树堆与红黑树，其功能与性能相当，并且跳表的代码长度更短，其设计思想与链表相似。例如，一个跳表包含 [30, 40, 50, 60, 70, 90]，然后增加 80、45 到跳表中，以下图的方式操作：



- 跳表中有很多层，每一层是一个短的链表
- 你的设计应该要包含这些函数：
  - void add(int num): 插入一个元素到跳表
  - bool search(int target) : 返回target是否存在于跳表中。
  - bool erase(int num): 在跳表中删除一个值，如果 num 不存在，直接返回false。如果存在多个 num ，删除其中任意一个即可。

题目描述

拉勾教育
—互联网人实战大学—

约束条件

- 0 <= num, target <= 20000
- 最多调用 50000 次 search, add, 以及 erase操作。

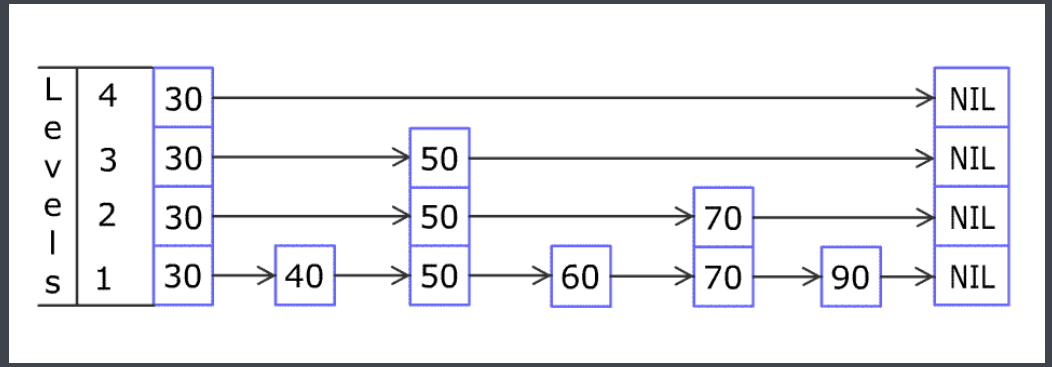
样例：

```
Skiplist skiplist = new Skiplist();

skiplist.add(1);
skiplist.add(2);
skiplist.add(3);
skiplist.search(0);   // 返回 false
skiplist.add(4);
skiplist.search(1);   // 返回 true
skiplist.erase(0);    // 返回 false，0 不在跳表中
skiplist.erase(1);    // 返回 true
skiplist.search(1);   // 返回 false，1 已被擦除
```

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

题目主干要求



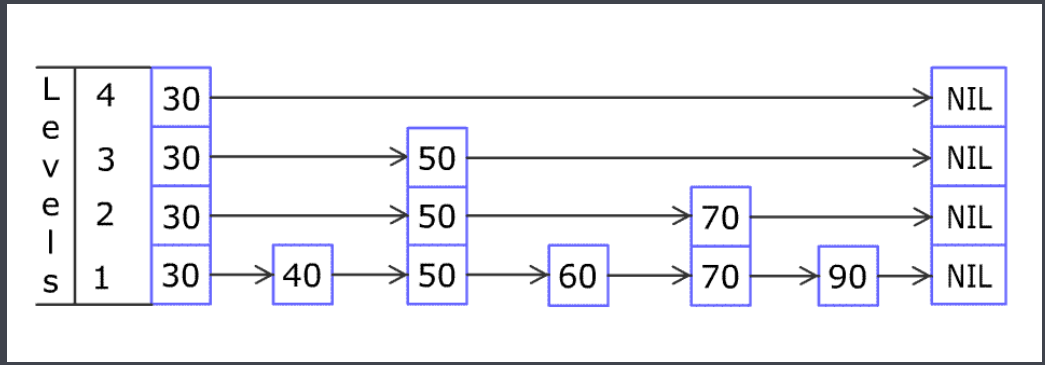
- 不能使用任何库函数
- 跳表由很多层有序的链表结构组成
- 最底层（Level 1）称为原链表，包含所有元素
- 上面每一层（Level i）链表都叫索引层，上级索引是下一级的子集
- 如果一个元素出现在 Level i 的链表中，则它在 Level i 之下的链表也都会出现
- 每个节点包含两个指针，一个指向同一链表中的后一个元素，一个指向下面一层的元素
- 跳表的每一个操作都是从最高层链表的头节点开始，从左往右，从上往下

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

题目附加信息

- 添加元素允许重复
- 删除重复元素中的一个即可
- 添加元素后，用扔硬币的方式决定是否创建索引
- 删除元素时，应考虑索引的删除问题
- 跳表中存储的数据范围是：[0, 20000]



一. Comprehend 理解题意

解题思路

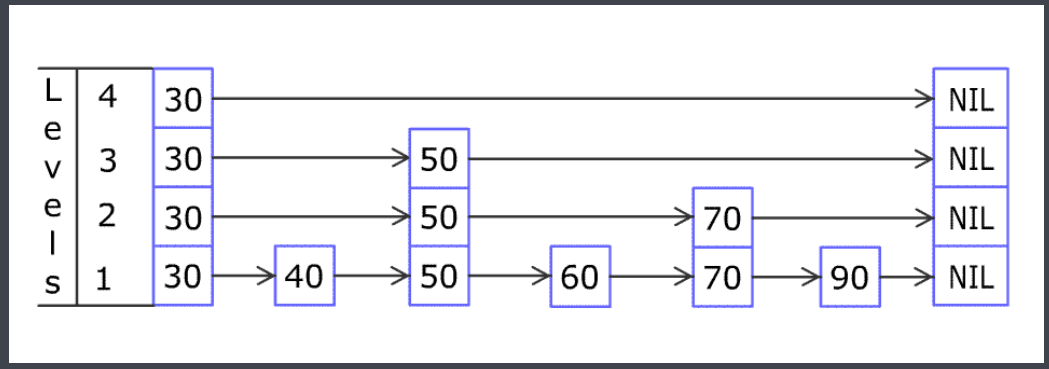
- 跳表（Skiplist）类的成员：
  - Node head：跳表的头节点指针
  - void add(int num)：插入
  - boolean search(int target)：查询
  - boolean erase(int num)：删除
- 节点（Node）类的成员：
  - int val：数据域
  - Node right，down：指针域，向右，向下

```
class Skiplist {
    final int HEAD_VALUE = -1;  // 链表头节点的值
    final Node HEAD = new Node(HEAD_VALUE);
    Node head;          // 最左上角的头节点，所有操作的开始位置
    int levels;         // 当前层级，即 head 节点所在的最高层数
    public Skiplist() {
        head = HEAD;
        levels = 1;
    }
    class Node {
        int val;           // 数据域
        Node right, down; // 指针域：向右，向下

        Node(int val) {
            this(val, null, null);
        }

        Node(int val, Node right, Node down) {
            this.val = val;
            this.right = right;
            this.down = down;
        }
    }
}
```

二. Choose 数据结构及算法思维选择

数据结构：

- 跳表的每一层都是链表



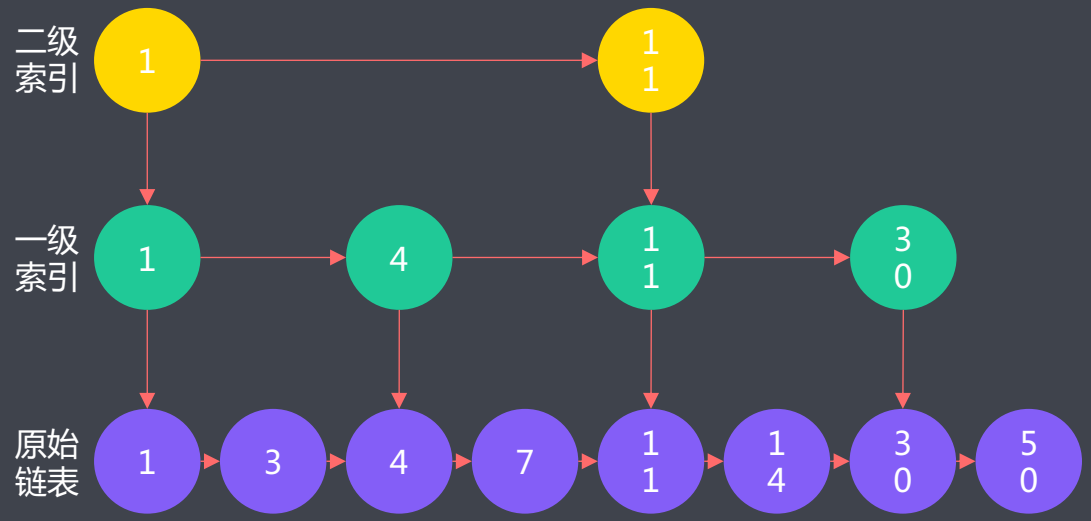
算法思维：

- 遍历

二. Choose 数据结构及算法思维选择

关键知识点：跳表

- Skip List，跳跃表，简称跳表
- 实质是一种可以进行二分查找的链表
- 在原有的有序链表上面增加了多级索引，通过索引来实现快速查找，以空间换时间



跳表的特点

- 多层结构，每一层随机概率产生
- 每一层都是有序链表，默认升序，最底层包含所有元素，即原链表
- 每个节点包含两个指针：向右（right，同级链表）、向下（down，下级链表）

三. Code 基本解法及编码实现

步骤一：实现一个不含索引的跳表

1. 创建跳表时初始化头节点和当前层级

   head = HEAD; // -1，right=null，down=null

2. 完成查找功能 ：

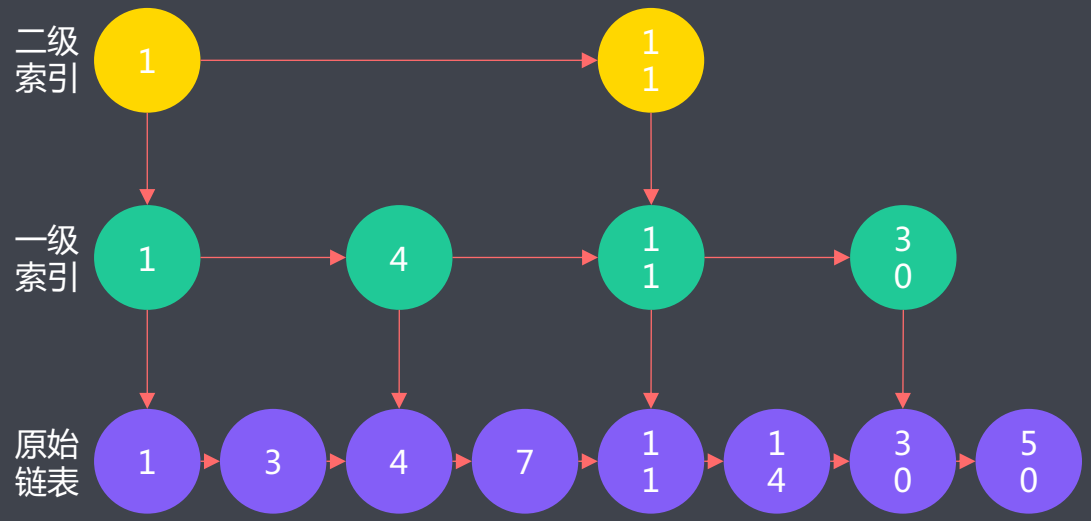
   boolean search(int target)：找到返回true

   从 head 开始，从左到右、从上到下依次查找

   如果 right 指向null，即尾节点，下移

   如果节点值等于 target，返回

   小于往右，大于往下



三. Code 基本解法及编码实现

步骤一：实现一个不含索引的跳表

3. 完成删除功能：

   boolean erase(int num)：删除成功返回true

   查找到要删除的元素，方式与 search 相同

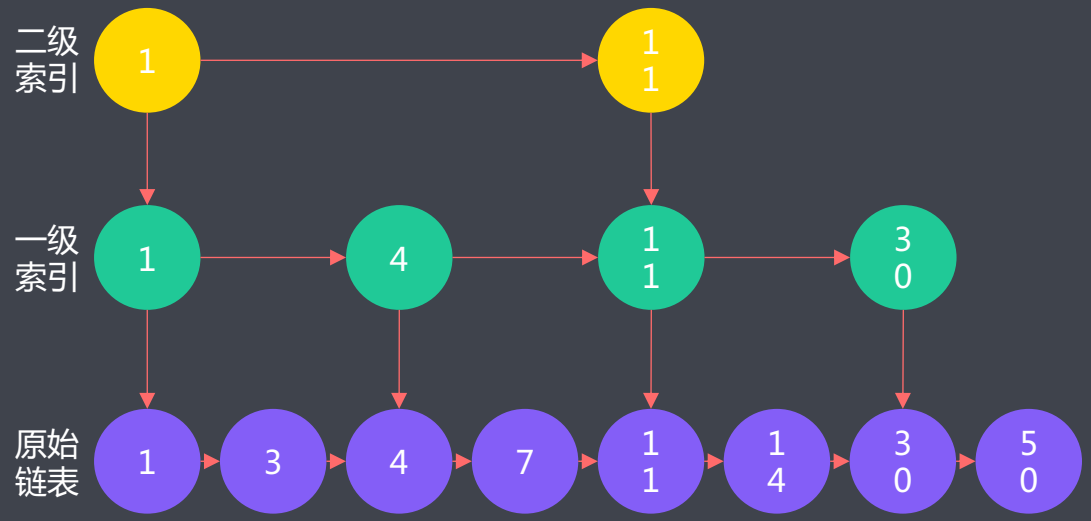
   执行删除操作：前面节点指向后面节点

4. 完成插入功能：

   void add(int num)：插入原链表，支持重复

   定位原链表中要插入的位置，方式与 search 相同，新元素放到重复元素后

   新建节点，执行插入操作



三. Code 基本解法及编码实现

步骤二：实现有索引的插入和删除

3. 完成删除功能：

   下移，删除每一层

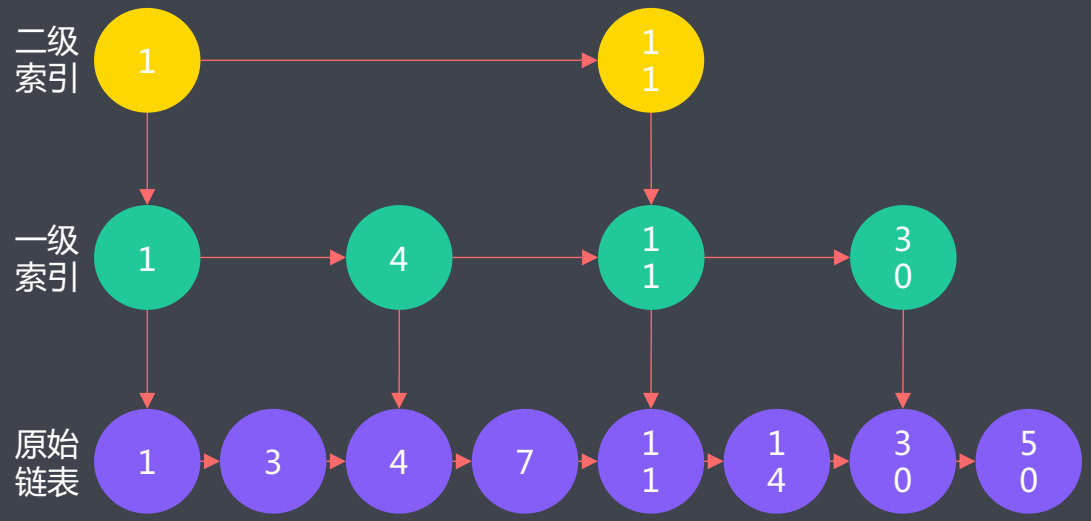
4. 完成插入功能：

   抛硬币生成（多层）索引：

   正面朝上（随机数1），生成索引

   背面朝上（随机数0），返回

索引层不存在，新建索引链表的头节点和索引节点



三. Code 基本解法及编码实现


```java
public boolean search(int target) {
    Node n = head;
    while (n != null) {
        // 1.在同一层级上向右查找，直到链表结尾，或者找到
        while (n.right != null && n.right.val < target) {
            n = n.right;
        }
        // 2.若找到，返回true
        Node right = n.right; // 要查找的节点
        if (right != null && right.val == target) {
           return true;
        }
        // 3.若右侧数据较大，向下一层
        n = n.down;
    }
    return false;
}
```

```java
public boolean erase(int num) {
    boolean exist = false;
    Node n = head;
    while (n != null) {
        // 1.获取该指定数据节点的前一个节点
        while (n.right != null && n.right.val < num) {
            n = n.right;
        }
        // 2.与当前层链表断开
        Node right = n.right; // 要删除的节点
        if (right != null && right.val == num) {
            n.right = right.right;
            right.right = null; // help GC
            exist = true;
        }
        // 3.删除下一层
        n = n.down;
    }
    return exist;
}
```

三. Code 基本解法及编码实现


```java
public void add(int num) {
    // 1.定位插入位置：原链表中 >= num 的最小节点前
    Node node = head; // 从 head 开始查找
    // 节点向下，可能是生成索引的位置，使用数组记录这些节点
    Node[] nodes = new Node[levels];
    int i = 0; // 操作上述数组
    while (node != null) { // node==null 时，到达原链表
        // 在同一层级上向右查找，直到链表结尾，或者找到
        while (node.right != null && node.right.val < num) {
            node = node.right;
        }
        // 右侧为结尾 or 右侧值大 or 右侧值相同
        nodes[i++] = node;
        // 继续查找下一层的位置
        node = node.down;
    }

    // 2.插入新节点
    node = nodes[--i]; // nodes中最后一个元素
    Node newNode = new Node(num, node.right, null);
    node.right = newNode;

    // 3.根据扔硬币决定（是否）生成索引
    addIndicesByCoinFlip(newNode, nodes, i);
}
```

```java
private void addIndicesByCoinFlip(Node target, Node[] nodes, int
indices) {
    Node downNode = target;
    Random random = new Random();
    int coins = random.nextInt(2); // 0 or 1, 50% 概率
    // 1.抛硬币，在现有跳表层数范围内建立索引
    while (coins == 1 && indices > 0) {
        Node prev = nodes[--indices]; // 数组的倒数第二个元素，level 2
        Node newIndex = new Node(target.val, prev.right, downNode);
        prev.right = newIndex;

        downNode = newIndex;
        coins = random.nextInt(2);
    }
    // 2.抛硬币，决定是否建立一层超出跳表层数的索引层
    if (coins == 1) { // 新建一个索引层级
        // 新建索引节点和 head 节点
        Node newIndex = new Node(target.val, null, downNode);
        Node newHead = new Node(HEAD_VALUE, newIndex, head);
        head = newHead; // head 指针上移
        levels++; // 跳表层数加 1
    }
}
```

四. Consider 思考更优解

1. 剔除无效代码或优化空间消耗
   - 查找和删除方法定位元素的方式是相同的，可以优化
   - 跳表的效率跟索引紧密相关，可以调整索引的生成策略来尝试优化
2. 寻找更好的算法思维
   - 借鉴其它算法

五. Code 最优解思路及编码实现

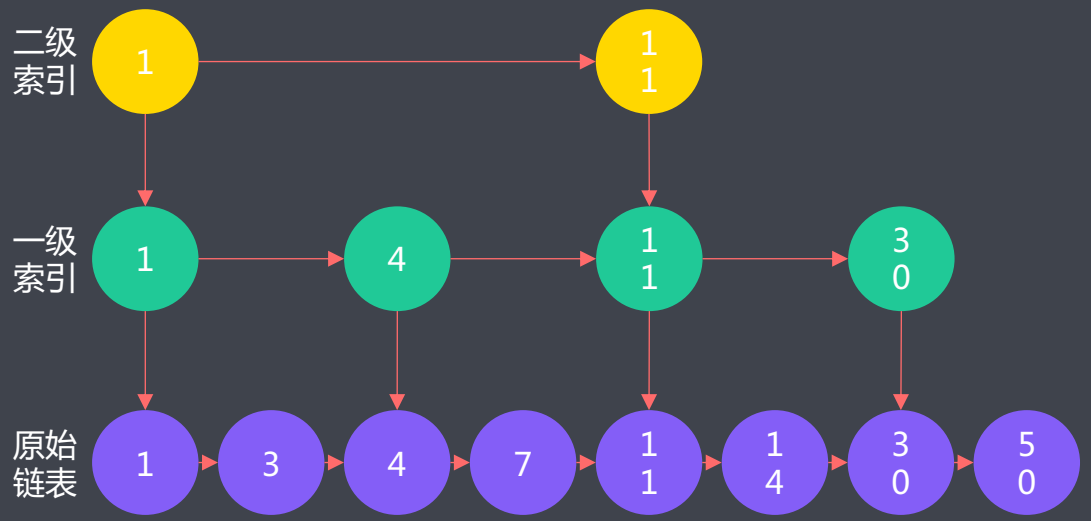
拉勾教育
—互联网人实战大学—

```
public Node get(int target) {
    return get(target, head);
}

public Node get(int target, Node from) {
    Node n = from;
    while (n != null) {
        // 1.在同一层级上向右查找，直到链表结尾，或者找到
        while (n.right != null && n.right.val < target) {
            n = n.right;
        }
        // 2.若找到，返回true
        Node right = n.right; // 要查找的节点
        if (right != null && right.val == target) {
            return n; // 返回要查找的节点的前一个
        }
        // 3.若右侧数据较大，向下一层
        n = n.down;
    }
    return null;
}
```

```
public boolean search(int target) {
    Node node = get(target);
    return node != null;
}
```

```
public boolean erase(int num) {
    boolean exist = false;
    Node node = get(num, head);
    while (node != null) {
        Node right = node.right; // 要删除的节点
        node.right = right.right;
        right.right = null; // help GC
        exist = true;

        node = get(num, node.down);
    }
    return exist;
}
```

五. Code 最优解思路及编码实现



五. Code 最优解思路及编码实现


```
private void addIndicesByCoinFlip(Node target, Node[] nodes, int
indices) {
    Node downNode = target;
    Random random = new Random();
    int coins = random.nextInt(2); // 0 or 1, 50% 概率
    // 1.抛硬币，在现有跳表层数范围内建立索引
    while (coins == 1 && levels < (length >> 6)) {
        if (indices > 0) {
            // 数组的倒数第二个元素，level 2
            Node prev = nodes[--indices];
            Node newIndex = new Node(target.val, prev.right, downNode);
            prev.right = newIndex;

            downNode = newIndex;
            coins = random.nextInt(2);
        } else { // 新建一个索引层级
            // 新建索引节点和 head 节点
            Node newIndex = new Node(target.val, null, downNode);
            Node newHead = new Node(HEAD_VALUE, newIndex, head);
            head = newHead; // head 指针上移
            levels++; // 跳表层数加 1
        }
    }
}
```

平均时间复杂度：O(log(n))

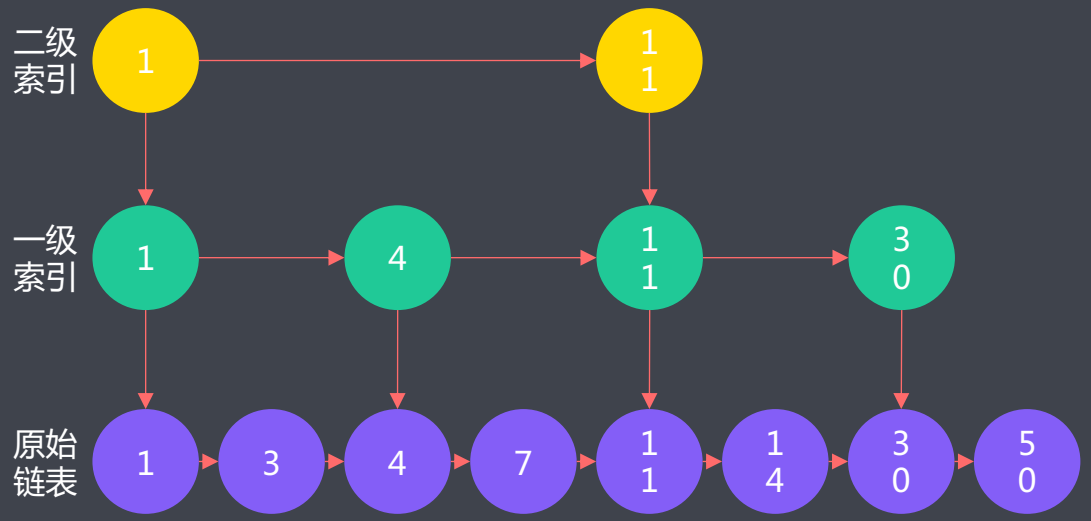
- 查找的时间复杂度：O(log(n))
- 删除的时间复杂度：O(log(n)) + O(k)
- 插入的时间复杂度：O(log(n)) + O(k)

空间复杂度：O(n)

- 原链表：O(n)
- 索引：O(n)

执行耗时:19 ms,击败了86.68% 的Java用户
内存消耗:44.6 MB,击败了40.63% 的Java用户

五. Code 最优解思路及编码实现



平均时间复杂度：O(log(n))

- 查找的时间复杂度：O(log(n))
- 删除的时间复杂度：O(log(n)) + O(k)
- 插入的时间复杂度：O(log(n)) + O(k)

空间复杂度：O(n)

- 原链表：O(n)
- 索引：O(n)

执行耗时:19 ms,击败了86.68% 的Java用户
内存消耗:44.6 MB,击败了40.63% 的Java用户

六. Change 变形延伸

拉勾教育
—互联网人实战大学—

题目变形

- ConcurrentSkipListMap：TreeMap的并发版本
- ConcurrentSkipListSet：TreeSet的并发版本
- 给定一个链表，如何转成跳表结构？

延伸扩展

- Redis，LevelDB，Lucene

本题来源

- Leetcode 1206 https://leetcode-cn.com/problems/design-skiplist/

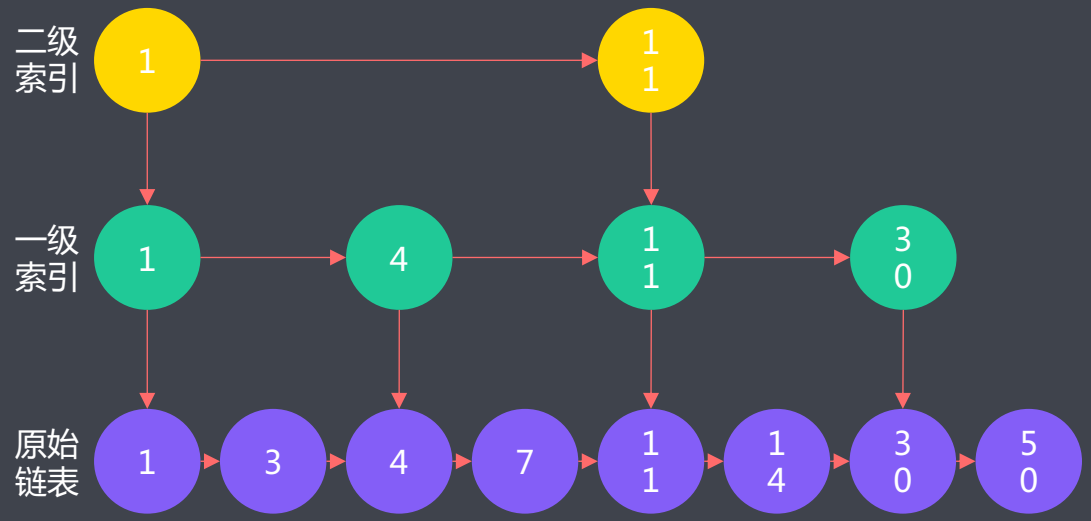
总结

拉勾教育
—互联网人实战大学—

6C解题法

关键知识点：跳表

- Skip List，跳跃表，简称跳表
- 实质是一种可以进行二分查找的链表
- 在原有的有序链表上面增加了多级索引，通过索引来实现快速查找，以空间换时间



跳表的特点

- 多层结构，每一层随机概率产生
- 每一层都是有序链表，默认升序，最底层包含所有元素，即原链表
- 每个节点包含两个指针：向右（right，同级链表）、向下（down，下级链表）

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容