王争的算法训练营

习题课：哈希表

习题课：哈希表

王争的算法训练营

配套习题：

两数之和 （简单）**例题**

15. 三数之和（中等）**例题**

160. 相交链表（简单）

141. 环形链表（简单） 判断链表中是否存在环

面试题 02.01. 移除重复节点（中等）

面试题 16.02. 单词频率 （简单）

面试题 01.02. 判定是否互为字符重排（简单）

242. 有效的字母异位词（简单）

49. 字母异位词分组（中等）

剑指 Offer 03. 数组中重复的数字 （简单）

136. 只出现一次的数字（简单）

349. 两个数组的交集 （简单）

1122. 数组的相对排序（中等）

面试题 16.21. 交换和（中等）

706. 设计哈希映射（简单）实现HashMap

146. LRU 缓存机制 （中等）标准的LRU **例题**

题型说明

不难，容易掌握　　查找、统计、判重

纯粹考察哈希表的题目不难，也不多。大部分情况下，哈希表只不过是一个小配角，配合解决其他算法类型的题目。用到哈希表的场景也比较明确，就是为了提高查找的效率，让查找的时间复杂度降为O(1)。

布隆过滤器、位图作为哈希表的延伸，往往用在大数据处理中，如果面试中考到，大部分都是讲讲思路就够了，不会让候选人动手编程。

习题课：哈希表

[两数之和](#)（简单） 两数之和 **例题**

给定一个整数数组 `nums` 和一个整数目标值 `target`，请你在该数组中找出 **和为目标值** 的那 **两个** 整数，并返回它们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

你可以按任意顺序返回答案。

示例 1：

```
输入：nums = [2,7,11,15], target = 9
输出：[0,1]
解释：因为 nums[0] + nums[1] == 9 ，返回 [0, 1] 。
```

示例 2：

```
输入：nums = [3,2,4], target = 6
输出：[1,2]
```

习题课：哈希表

两数之和 （简单） 两数之和

```
class Solution {
    public int[] twoSum(int[] nums, int target) {
      int n = nums.length;
      // 哈希表,key数本身,value是下标
      HashMap<Integer, Integer> hashTable = new HashMap<>();
      for (int i = 0; i < n; ++i) {
        hashTable.put(nums[i], i);
      }

      for (int i = 0; i < n; i++) {
        if (hashTable.containsKey(target-nums[i])) {
          int value = hashTable.get(target-nums[i]);
          if (value != i) {
            return new int[] {i, value};
          }
        }
      }
      return new int[0];
    }
}
```

习题课：哈希表

王争的算法训练营

15. 三数之和（中等）三数之和 **例题**

给你一个包含 `n` 个整数的数组 `nums`，判断 `nums` 中是否存在三个元素 *a*，*b*，*c*，使得 *a* + *b* + *c* = 0？请你找出所有和为 `0` 且不重复的三元组。

注意：答案中不可以包含重复的三元组。

多种重复：1，1，5，3
1、(3, 5)、(5, 3)
2、(1, 3)、(1, 3)

示例 1：

```
输入：nums = [-1,0,1,2,-1,-4]
输出：[[-1,-1,2],[-1,0,1]]
```

示例 2：

```
输入：nums = []
输出：[]
```

示例 3：

```
输入：nums = [0]
输出：[]
```

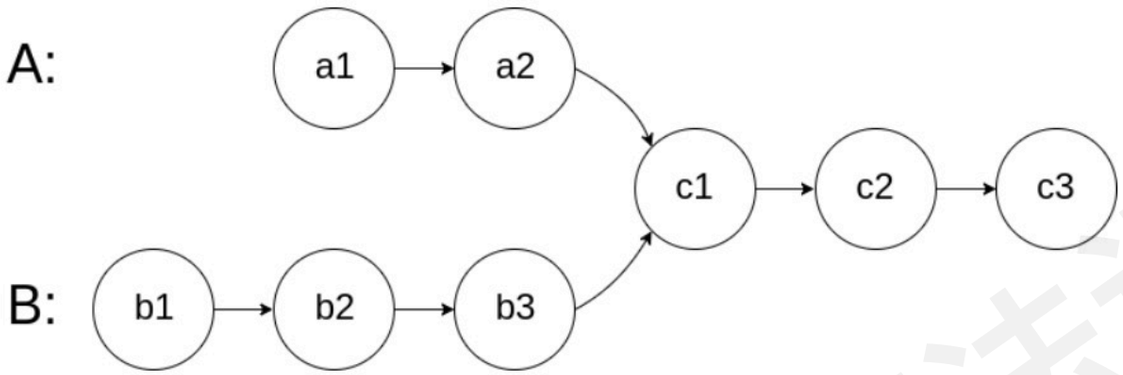
```java
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
      Arrays.sort(nums); // 避免重复
      List<List<Integer>> result = new ArrayList<>();
      int n = nums.length;
      HashMap<Integer, Integer> hashMap = new HashMap<>();
      for (int i = 0; i < n; ++i) {
        hashMap.put(nums[i], i);
      }
      for (int i = 0; i < n; ++i) {
        if (i != 0 && nums[i] == nums[i-1]) continue; // 避免a重复, 1,1,3…
        for (int j = i+1; j < n; ++j) {
          if (j != i+1 && nums[j] == nums[j-1]) continue; // 避免b重复 1,2,2,…
          int target = -1*(nums[i]+nums[j]);
          if (!hashMap.containsKey(target)) {
            continue;
          }
          int k = hashMap.get(target);
          if (k > j) { // 避免重复
            List<Integer> resultItem = new ArrayList<>();
            resultItem.add(nums[i]);
            resultItem.add(nums[j]);
            resultItem.add(nums[k]);
            result.add(resultItem);
          }
        }
      }
      return result;
    }
}
```

习题课：哈希表

160. 相交链表（简单）

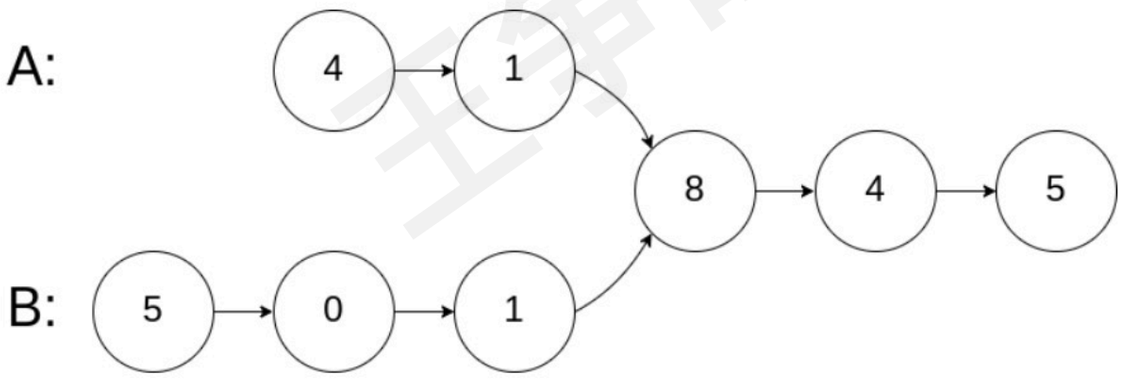
编写一个程序，找到两个单链表相交的起始节点。

如下面的两个链表：



在节点 c1 开始相交。

示例 1：



习题课：哈希表

160. 相交链表（简单）

```
public class Solution {
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        HashSet<ListNode> hashTable = new HashSet<>();
        ListNode p = headA;
        while (p != null) {
            hashTable.add(p);
            p = p.next;
        }
        p = headB;
        while (p != null) {
            if (hashTable.contains(p)) {
                return p;
            }
            p = p.next;
        }
        return null;
    }
}
```

习题课：哈希表

141. 环形链表（简单） 判断链表中是否存在环

给定一个链表，判断链表中是否有环。

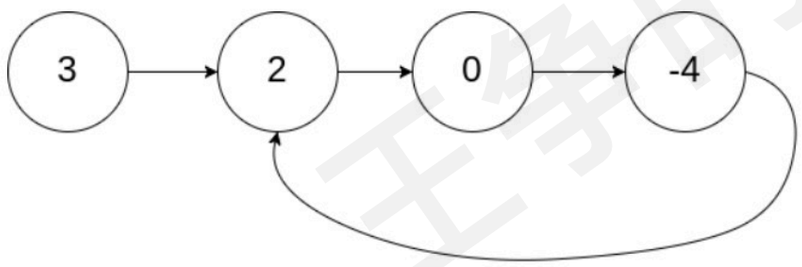
如果链表中有某个节点，可以通过连续跟踪 `next` 指针再次到达，则链表中存在环。 为了表示给定链表中的环，我们使用整数 `pos` 来表示链表尾连接到链表中的位置（索引从 0 开始）。 如果 `pos` 是 `-1`，则在该链表中没有环。**注意：`pos` 不作为参数进行传递**，仅仅是为了标识链表的实际情况。

如果链表中存在环，则返回 `true` 。 否则，返回 `false` 。

进阶：

你能用 *O(1)*（即，常量）内存解决此问题吗？

示例 1：




```
输入：head = [3,2,0,-4], pos = 1
输出：true
解释：链表中有一个环，其尾部连接到第二个节点。
```

习题课：哈希表

141. 环形链表（简单） 判断链表中是否存在环

```
public class Solution {
    public boolean hasCycle(ListNode head) {
        HashSet<ListNode> hashTable = new HashSet<>();
        ListNode p = head;
        while (p != null) {
            if (hashTable.contains(p)) {
                return true;
            } else {
                hashTable.add(p);
            }
            p = p.next;
        }
        return false;
    }
}
```

习题课：哈希表

[面试题 02.01. 移除重复节点](#)（中等）

编写代码，移除未排序链表中的重复节点。保留最开始出现的节点。

示例1:

```
输入：[1, 2, 3, 3, 2, 1]
输出：[1, 2, 3]
```

示例2:

```
输入：[1, 1, 1, 1, 2]
输出：[1, 2]
```

提示：

1. 链表长度在[0, 20000]范围内。
2. 链表元素在[0, 20000]范围内。

习题课：哈希表

面试题 02.01. 移除重复节点（中等）

```
class Solution {
    public ListNode removeDuplicateNodes(ListNode head) {
        if (head == null) return head;
        Set<Integer> set = new HashSet<>();
        ListNode newHead = new ListNode();
        ListNode tail = newHead;
        ListNode p = head;
        while (p != null) {
            ListNode tmp = p.next;
            if (!set.contains(p.val)) {
                set.add(p.val);
                tail.next = p;
                tail = p;
                tail.next = null;
            }
            p = tmp;
        }
        return newHead.next;
    }
}
```

习题课：哈希表

王争的算法训练营

[面试题 16.02. 单词频率](#)（简单）

除了hash，还有其他解决方案吗?

设计一个方法，找出任意指定单词在一本书中的出现频率。

你的实现应该支持如下操作：

- `WordsFrequency(book)` 构造函数，参数为字符串数组构成的一本书
- `get(word)` 查询指定单词在书中出现的频率

示例：

```
WordsFrequency wordsFrequency = new WordsFrequency({"i", "have",
"an", "apple", "he", "have", "a", "pen"});
wordsFrequency.get("you"); //返回0，"you"没有出现过
wordsFrequency.get("have"); //返回2，"have"出现2次
wordsFrequency.get("an"); //返回1
wordsFrequency.get("apple"); //返回1
wordsFrequency.get("pen"); //返回1
```

提示：

- `book[i]` 中只包含小写字母
- `1 <= book.length <= 100000`
- `1 <= book[i].length <= 10`
- `get` 函数的调用次数不会超过100000

习题课：哈希表

面试题 16.02. 单词频率 （简单）

```
class WordsFrequency {
    private Map<String, Integer> map = new HashMap<>();

    public WordsFrequency(String[] book) {
        for (String word : book) {
            int count = 1;
            if (map.containsKey(word)) {
                count += map.get(word);
            }
            map.put(word, count);
        }
    }

    public int get(String word) {
        if (!map.containsKey(word)) {
            return 0;
        }
        return map.get(word);
    }
}
```

习题课：哈希表

王争的算法训练营

[面试题 01.02. 判定是否互为字符重排](#)（简单）

给定两个字符串 `s1` 和 `s2`，请编写一个程序，确定其中一个字符串的字符重新排列后，能否变成另一个字符串。

示例 1：

```
输入: s1 = "abc", s2 = "bca"
输出: true
```

示例 2：

```
输入: s1 = "abc", s2 = "bad"
输出: false
```

说明：

- `0 <= len(s1) <= 100`
- `0 <= len(s2) <= 100`

王争的算法训练营

```
class Solution {
    public boolean CheckPermutation(String s1, String s2) {
        HashMap<Character, Integer> s1ht = new HashMap<>();
        for (int i = 0; i < s1.length(); ++i) {
            char c = s1.charAt(i);
            int count = 1;
            if (s1ht.containsKey(c)) {
                count += s1ht.get(c);
            }
            s1ht.put(c, count);
        }

        // s2去跟s1匹配
        for (int i = 0; i < s2.length(); ++i) {
            char c = s2.charAt(i);
            if (!s1ht.containsKey(c)) {
                return false;
            }
            int count = s1ht.get(c);
            if (count == 0) return false;
            s1ht.put(c, count-1);
        }

        // 检查s1ht是否为空
        for (int i = 0; i < s1.length(); ++i) {
            char c = s1.charAt(i);
            if (s1ht.get(c) != 0) return false;
        }

        return true;
    }
}
```

习题课：哈希表

242. 有效的字母异位词（简单）

给定两个字符串 *s* 和 *t* ，编写一个函数来判断 *t* 是否是 *s* 的字母异位词。

示例 1:

```
输入: s = "anagram", t = "nagaram"
输出: true
```

示例 2:

```
输入: s = "rat", t = "car"
输出: false
```

说明:
你可以假设字符串只包含小写字母。

进阶:
如果输入字符串包含 unicode 字符怎么办？你能否调整你的解法来应对这种情况？

习题课：哈希表

王争的算法训练营

242. 有效的字母异位词（简单）

```
class Solution {
    public boolean isAnagram(String s, String t) {
        if (s.length() != t.length()) {
            return false;
        }
       int[] nums1 = new int[26];
        for (int i = 0; i < s.length(); ++i) {
            char c = s.charAt(i);
            nums1[c-'a']++;
        }
        int[] nums2 = new int[26];
        for (int i = 0; i < t.length(); ++i) {
            char c = t.charAt(i);
            nums2[c-'a']++;
        }
        for (int i = 0; i < 26; ++i) {
            if (nums1[i] != nums2[i]) {
                return false;
            }
        }
        return true;
    }
}
```

习题课：哈希表

[49. 字母异位词分组](#)（中等）

给定一个字符串数组，将字母异位词组合在一起。字母异位词指字母相同，但排列不同的字符串。

示例:

统计单词出现频率+判断异位词

```
输入: ["eat", "tea", "tan", "ate", "nat", "bat"]
输出:
[
  ["ate","eat","tea"],
  ["nat","tan"],
  ["bat"]
]
```

说明:

- 所有输入均为小写字母。
- 不考虑答案输出的顺序。

习题课：哈希表

49. 字母异位词分组（中等）

```
class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        Map<String, List<String>> map = new HashMap<String, List<String>>();
        for (String str : strs) {
            char[] array = str.toCharArray();
            Arrays.sort(array);
            String key = new String(array);
            List<String> list = map.getOrDefault(key, new ArrayList<String>());
            list.add(str);
            map.put(key, list);
        }
        return new ArrayList<List<String>>(map.values());
    }
}
```

习题课：哈希表

[剑指 Offer 03. 数组中重复的数字](#) （简单）

1、排序
2、哈希表
3、位图

找出数组中重复的数字。

在一个长度为 n 的数组 nums 里的所有数字都在 0～n-1 的范围内。数组中某些数字是重复的，但不知道有几个数字重复了，也不知道每个数字重复了几次。请找出数组中任意一个重复的数字。

示例 1：

```
输入：
[2, 3, 1, 0, 2, 5, 3]
输出：2 或 3
```

限制：

`2 <= n <= 100000`

习题课：哈希表

剑指 Offer 03. 数组中重复的数字 （简单）

```
class Solution {
    public int findRepeatNumber(int[] nums) {
        HashSet<Integer> set = new HashSet<>();
        for (int i = 0; i < nums.length; ++i) {
            if (set.contains(nums[i])) return nums[i];
            set.add(nums[i]);
        }
        return -1;
    }
}
```

习题课：哈希表

136. 只出现一次的数字（简单）

给定一个**非空**整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

说明：

你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？

1、排序
2、哈希表
3、位运算

示例 1:

```
输入: [2,2,1]
输出: 1
```

示例 2:

```
输入: [4,1,2,1,2]
输出: 4
```

习题课：哈希表

136. 只出现一次的数字（简单）

```
class Solution {
    public int singleNumber(int[] nums) {
        HashMap<Integer, Integer> hashTable = new HashMap<>();
        for (int i = 0; i < nums.length; ++i) {
            int count = 1;
            if (hashTable.containsKey(nums[i])) {
                count += hashTable.get(nums[i]);
            }
            hashTable.put(nums[i], count);
        }
        for (int i = 0; i < nums.length; ++i) {
            int count = hashTable.get(nums[i]);
            if (count == 1) return nums[i];
        }
        return -1;
    }
}
```

习题课：哈希表

王争的算法训练营

[349. 两个数组的交集](#) （简单）

1、排序+双指针
2、哈希表

给定两个数组，编写一个函数来计算它们的交集。

示例 1：

```
输入：nums1 = [1,2,2,1], nums2 = [2,2]
输出：[2]
```

示例 2：

```
输入：nums1 = [4,9,5], nums2 = [9,4,9,8,4]
输出：[9,4]
```

说明：

- 输出结果中的每个元素一定是唯一的。
- 我们可以不考虑输出结果的顺序。

扩展: k个数组的交集

习题课：哈希表

349. 两个数组的交集 （简单）

```
class Solution {
    public int[] intersection(int[] nums1, int[] nums2) {
        HashSet<Integer> hashTable = new HashSet<>();
        for (int i = 0; i < nums1.length; ++i) {
            hashTable.add(nums1[i]);
        }
        List<Integer> result = new ArrayList<>();
        for (int i = 0; i < nums2.length; ++i) {
            if (hashTable.contains(nums2[i])) {
                hashTable.remove(nums2[i]);
                result.add(nums2[i]);
            }
        }
        int[] resultArr = new int[result.size()];
        for (int i = 0; i < result.size(); ++i) {
            resultArr[i] = result.get(i);
        }
        return resultArr;
    }
}
```

习题课：哈希表

1122. 数组的相对排序（中等）

给你两个数组，`arr1` 和 `arr2`，

- `arr2` 中的元素各不相同
- `arr2` 中的每个元素都出现在 `arr1` 中

对 `arr1` 中的元素进行排序，使 `arr1` 中项的相对顺序和 `arr2` 中的相对顺序相同。未在 `arr2` 中出现过的元素需要按照升序放在 `arr1` 的末尾。

示例：

```
输入：arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]
输出：[2,2,2,1,4,3,3,9,6,7,19]
```

提示：

- `1 <= arr1.length, arr2.length <= 1000`
- `0 <= arr1[i], arr2[i] <= 1000`
- `arr2` 中的元素 `arr2[i]` 各不相同
- `arr2` 中的每个元素 `arr2[i]` 都出现在 `arr1` 中

```java
class Solution {
    public int[] relativeSortArray(int[] arr1, int[] arr2) {
        // arr2中每个数字在arr1中出现的次数
        HashMap<Integer, Integer> counts = new HashMap<>();
        // 先用arr2构建hash表
        for (int i = 0; i < arr2.length; ++i) {
            counts.put(arr2[i], 0);
        }
        // 扫描arr1统计arr2中每个数字在arr1中出现的次数
        for (int i = 0; i < arr1.length; ++i) {
            if (counts.containsKey(arr1[i])) {
                int oldCount = counts.get(arr1[i]);
                counts.put(arr1[i], oldCount+1);
            }
        }
        int[] result = new int[arr1.length];
        int k = 0;
        // 将counts的数据按照arr2的顺序输出
        for (int i = 0; i < arr2.length; ++i) {
            int count = counts.get(arr2[i]);
            for (int j = 0; j < count; ++j) {
                result[k+j] = arr2[i];
            }
            k += count;
        }
        // 将arr1中未出现在arr2中的数据有序输出到result
        Arrays.sort(arr1);
        for (int i = 0; i < arr1.length; ++i) {
            if (!counts.containsKey(arr1[i])) {
                result[k++] = arr1[i];
            }
        }
        return result;
    }
}
```

习题课：哈希表

[面试题 16.21. 交换和](#)（中等）

给定两个整数数组，请交换一对数值（每个数组中取一个数值），使得两个数组所有元素的和相等。

返回一个数组，第一个元素是第一个数组中要交换的元素，第二个元素是第二个数组中要交换的元素。若有多个答案，返回任意一个均可。若无满足条件的数值，返回空数组。

示例:

```
输入: array1 = [4, 1, 2, 1, 1, 2], array2 = [3, 6, 3, 3]
输出: [1, 3]
```

示例:

```
输入: array1 = [1, 2, 3], array2 = [4, 5, 6]
输出: []
```

提示:

- `1 <= array1.length, array2.length <= 100000`

```java
class Solution {
    public int[] findSwapValues(int[] array1, int[] array2) {
        int n = array1.length;
        int m = array2.length;
        // 计算数组1的和
        int sum1 = 0;
        for (int i = 0; i < n; ++i) {
            sum1 += array1[i];
        }
        // 计算数组2的和，并且将元素放到哈希表中
        int sum2 = 0;
        HashSet<Integer> hashTable = new HashSet<>();
        for (int i = 0; i < m; ++i) {
            sum2 += array2[i];
            hashTable.add(array2[i]);
        }
        // sum1+sum2是奇数，那无解
        int sum = sum1+sum2;
        if (sum % 2 == 1) return new int[0];
        // 遍历数组1中的每个元素，在哈希表中查找
        int diff = sum/2 - sum1;
        for (int i = 0; i < n; ++i) {
            int target = array1[i] + diff;
            if (hashTable.contains(target)) {
                return new int[] {array1[i], target};
            }
        }
        return new int[0];
    }
}
```

习题课：哈希表

王争的算法训练营

706. 设计哈希映射（简单）

不使用任何内建的哈希表库设计一个哈希映射（HashMap）。

实现 `MyHashMap` 类：

- `MyHashMap()` 用空映射初始化对象
- `void put(int key, int value)` 向 HashMap 插入一个键值对 `(key, value)` 。如果 `key` 已经存在于映射中，则更新其对应的值 `value` 。
- `int get(int key)` 返回特定的 `key` 所映射的 `value` ；如果映射中不包含 `key` 的映射，返回 `-1` 。
- `void remove(key)` 如果映射中存在 `key` 的映射，则移除 `key` 和它所对应的 `value` 。

示例：

```
输入：
["MyHashMap", "put", "put", "get", "get", "put", "get", "remove",
"get"]
[[], [1, 1], [2, 2], [1], [3], [2, 1], [2], [2], [2]]
输出：
[null, null, null, 1, -1, null, 1, null, -1]
```

```java
class MyHashMap {
    private class Pair {
        public int key;
        public int value;

        public Pair(int key, int value) {
            this.key = key;
            this.value = value;
        }
    }

    private static final int SLOTS_COUNT = 3535;
    private LinkedList<Pair>[] slots;

    public MyHashMap() {
        slots = new LinkedList[SLOTS_COUNT];
    }

    private int hash(int key) {
        return key % SLOTS_COUNT;
    }

    public void put(int key, int value) {
        LinkedList<Pair> slot = slots[hash(key)];
        if (slot == null) {
            slots[hash(key)] = new LinkedList<>();
            slot = slots[hash(key)];
        }
        for (int i = 0; i < slot.size(); ++i) {
            Pair pair = slot.get(i);
            if (pair.key == key) {
                slot.remove(i);
                break;
            }
        }
        slot.add(new Pair(key, value));
    }
}
```

```java
    public int get(int key) {
        LinkedList<Pair> slot = slots[hash(key)];
        if (slot == null) {
            return -1;
        }
        for (int i = 0; i < slot.size(); ++i) {
            Pair pair = slot.get(i);
            if (pair.key == key) {
                return pair.value;
            }
        }
        return -1;
    }

    public void remove(int key) {
        LinkedList<Pair> slot = slots[hash(key)];
        if (slot == null) {
            return;
        }
        for (int i = 0; i < slot.size(); ++i) {
            Pair pair = slot.get(i);
            if (pair.key == key) {
                slot.remove(i);
                break;
            }
        }
    }
}
```

习题课：哈希表

146. LRU 缓存机制 （中等） **例题**

运用你所掌握的数据结构，设计和实现一个 LRU (最近最少使用) 缓存机制 。
实现 `LRUCache` 类：

- `LRUCache(int capacity)` 以正整数作为容量 `capacity` 初始化 LRU 缓存
- `int get(int key)` 如果关键字 `key` 存在于缓存中，则返回关键字的值，否则返回 `-1` 。
- `void put(int key, int value)` 如果关键字已经存在，则变更其数据值；如果关键字不存在，则插入该组「关键字-值」。当缓存容量达到上限时，它应该在写入新数据之前删除最久未使用的数据值，从而为新的数据值留出空间。

进阶：你是否可以在 `O(1)` 时间复杂度内完成这两种操作?

示例：

```
输入
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
输出
[null, null, null, 1, null, -1, null, -1, 3, 4]
```

习题课：哈希表

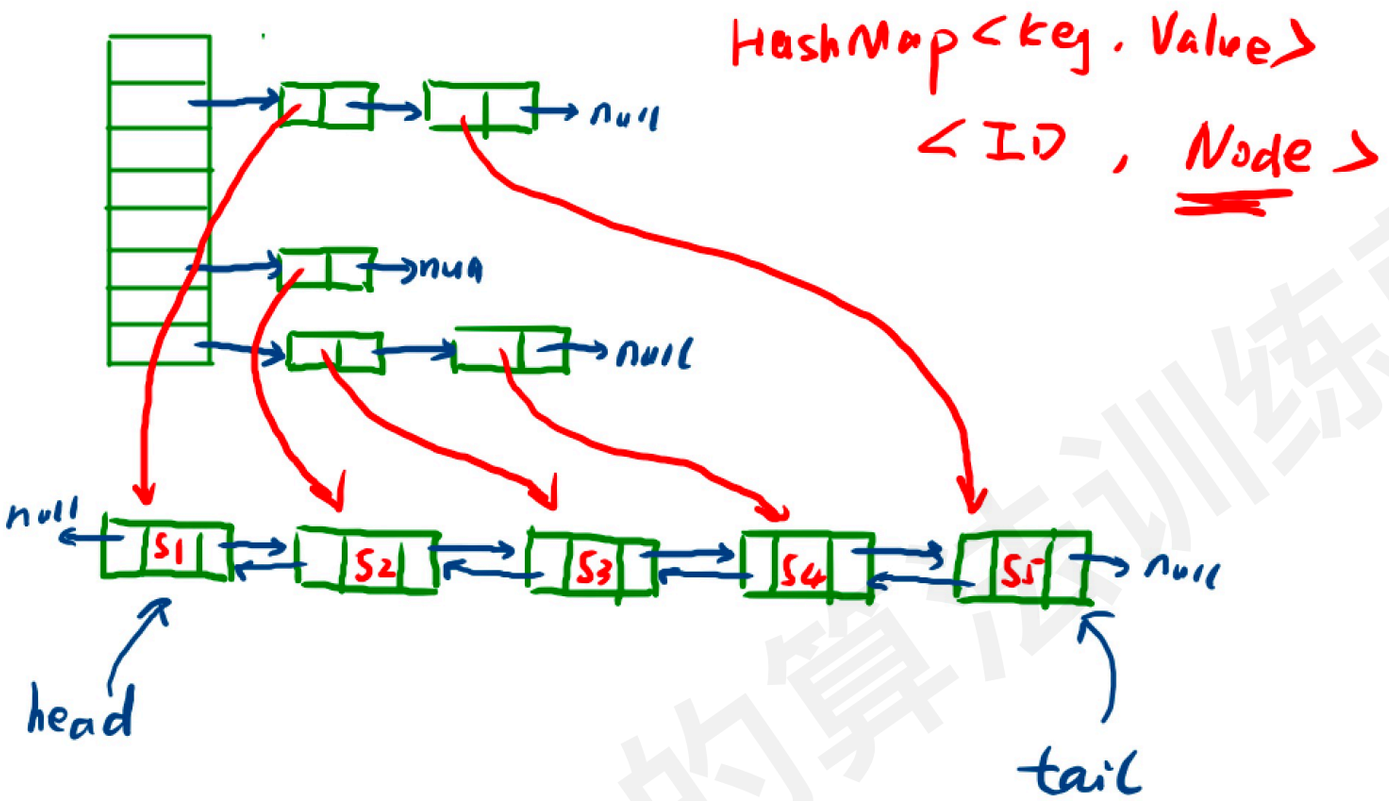
缓存主要包含3个操作：

1. 在缓存中查找一个数据；get
2. 从缓存中删除一个数据；remove
3. 往缓存中添加一个数据；put

基于哈希表+双向有序链表的实现方案：

1. 借助哈希表，快速得到要查找、要删除的结点。
2. 借助双向有序链表，维护数据的有序性（按照访问时间）。

习题课：哈希表



缓存主要包含3个操作：

1. 在缓存中查找一个数据；
2. 从缓存中删除一个数据；
3. 往缓存中添加一个数据；

双向有序链表：

1、头放最新的数据，尾放最老的数据

2、头放最老的诗句，尾放最新的数据

```java
class LRUCache {
  private class DLinkedNode {
    public int key;
    public int value;
    public DLinkedNode prev;
    public DLinkedNode next;
    public DLinkedNode(int key, int value) {
      this.key = key;
      this.value = value;
    }
  }

  private Map<Integer, DLinkedNode> hashtable = new HashMap<Integer, DLinkedNode>();
  private int size;
  private int capacity;
  private DLinkedNode head;
  private DLinkedNode tail;

  public LRUCache(int capacity) {
    this.size = 0;
    this.capacity = capacity;
    this.head = new DLinkedNode(-1, -1); // guard node
    this.tail = new DLinkedNode(-1, -1); // guard node
    this.head.prev = null;
    this.head.next = tail;
    this.tail.prev = head;
    this.tail.next = null;
  }
}
```

```
public int get(int key) { //1、在缓存中查找数据
  if (size == 0)
    return -1;
  DLinkedNode node = hashtable.get(key);
  if (node == null)
    return -1;
  removeNode(node);
  addNodeAtHead(node);
  return node.value;
}

public void remove(int key) { //2、从缓存删除数据
  DLinkedNode node = hashtable.get(key);
  if (node != null) {
    removeNode(node);
    hashtable.remove(key);
    size--;
    return;
  }
}

private void removeNode(DLinkedNode node) {
  node.next.prev = node.prev;
  node.prev.next = node.next;
}

private void addNodeAtHead(DLinkedNode node) {
  node.next = head.next;
  head.next.prev = node;
  head.next = node;
  node.prev = head;
}
```

缓存主要包含3个操作：

1. 在缓存中查找一个数据；
2. 从缓存中删除一个数据；
3. 往缓存中添加一个数据；

特殊情况：

- 删除尾结点一》虚拟尾结点
- 删除最后一个结点-》虚拟头结点

```java
//往缓存中添加一个数据
public void put(int key, int value) {
    DLinkedNode node = hashtable.get(key);
    if (node != null) {
        node.value = value;
        removeNode(node);
        addNodeAtHead(node);
        return;
    }

    if (size == capacity) {
        hashtable.remove(tail.prev.key);
        removeNode(tail.prev);
        size--;
    }
    DLinkedNode newNode = new DLinkedNode(key, value);
    addNodeAtHead(newNode);
    hashtable.put(key, newNode);
    size++;
}
```

缓存主要包含3个操作：

1. 在缓存中查找一个数据；
2. 从缓存中删除一个数据；
3. 往缓存中添加一个数据；

关注微信公众号“**小争哥**”，

后台回复“**PDF**”获取独家算法资料