# Leetcode 题解 - 哈希表

* [Leetcode 题解 - 哈希表](#leetcode-题解---哈希表)
  + [1. 数组中两个数的和为给定值](#1-数组中两个数的和为给定值)
  + [2. 判断数组是否含有重复元素](#2-判断数组是否含有重复元素)
  + [3. 最长和谐序列](#3-最长和谐序列)
  + [4. 最长连续序列](#4-最长连续序列)

哈希表使用 O(N) 空间复杂度存储数据，并且以 O(1) 时间复杂度求解问题。

* Java 中的 **HashSet** 用于存储一个集合，可以查找元素是否在集合中。如果元素有穷，并且范围不大，那么可以用一个布尔数组来存储一个元素是否存在。例如对于只有小写字符的元素，就可以用一个长度为 26 的布尔数组来存储一个字符集合，使得空间复杂度降低为 O(1)。

Java 中的 **HashMap** 主要用于映射关系，从而把两个元素联系起来。HashMap 也可以用来对元素进行计数统计，此时键为元素，值为计数。和 HashSet 类似，如果元素有穷并且范围不大，可以用整型数组来进行统计。在对一个内容进行压缩或者其它转换时，利用 HashMap 可以把原始内容和转换后的内容联系起来。例如在一个简化 url 的系统中 [Leetcdoe : 535. Encode and Decode TinyURL (Medium)

[Leetcode](https://leetcode.com/problems/encode-and-decode-tinyurl/description/)，利用 HashMap 就可以存储精简后的 url 到原始 url 的映射，使得不仅可以显示简化的 url，也可以根据简化的 url 得到原始 url 从而定位到正确的资源�) / [力扣](https://leetcode-cn.com/problems/encode-and-decode-tinyurl/description/)，利用 HashMap 就可以存储精简后的 url 到原始 url 的映射，使得不仅可以显示简化的 url，也可以根据简化的 url 得到原始 url 从而定位到正确的资源�)

## 1. 数组中两个数的和为给定值

1. Two Sum (Easy)

[Leetcode](https://leetcode.com/problems/two-sum/description/) / [力扣](https://leetcode-cn.com/problems/two-sum/description/)

可以先对数组进行排序，然后使用双指针方法或者二分查找方法。这样做的时间复杂度为 O(NlogN)，空间复杂度为 O(1)。

用 HashMap 存储数组元素和索引的映射，在访问到 nums[i] 时，判断 HashMap 中是否存在 target - nums[i]，如果存在说明 target - nums[i] 所在的索引和 i 就是要找的两个数。该方法的时间复杂度为 O(N)，空间复杂度为 O(N)，使用空间来换取时间。

public int[] twoSum(int[] nums, int target) {  
 HashMap<Integer, Integer> indexForNum = new HashMap<>();  
 for (int i = 0; i < nums.length; i++) {  
 if (indexForNum.containsKey(target - nums[i])) {  
 return new int[]{indexForNum.get(target - nums[i]), i};  
 } else {  
 indexForNum.put(nums[i], i);  
 }  
 }  
 return null;  
}

## 2. 判断数组是否含有重复元素

217. Contains Duplicate (Easy)

[Leetcode](https://leetcode.com/problems/contains-duplicate/description/) / [力扣](https://leetcode-cn.com/problems/contains-duplicate/description/)

public boolean containsDuplicate(int[] nums) {  
 Set<Integer> set = new HashSet<>();  
 for (int num : nums) {  
 set.add(num);  
 }  
 return set.size() < nums.length;  
}

## 3. 最长和谐序列

594. Longest Harmonious Subsequence (Easy)

[Leetcode](https://leetcode.com/problems/longest-harmonious-subsequence/description/) / [力扣](https://leetcode-cn.com/problems/longest-harmonious-subsequence/description/)

Input: [1,3,2,2,5,2,3,7]  
Output: 5  
Explanation: The longest harmonious subsequence is [3,2,2,2,3].

和谐序列中最大数和最小数之差正好为 1，应该注意的是序列的元素不一定是数组的连续元素。

public int findLHS(int[] nums) {  
 Map<Integer, Integer> countForNum = new HashMap<>();  
 for (int num : nums) {  
 countForNum.put(num, countForNum.getOrDefault(num, 0) + 1);  
 }  
 int longest = 0;  
 for (int num : countForNum.keySet()) {  
 if (countForNum.containsKey(num + 1)) {  
 longest = Math.max(longest, countForNum.get(num + 1) + countForNum.get(num));  
 }  
 }  
 return longest;  
}

## 4. 最长连续序列

128. Longest Consecutive Sequence (Hard)

[Leetcode](https://leetcode.com/problems/longest-consecutive-sequence/description/) / [力扣](https://leetcode-cn.com/problems/longest-consecutive-sequence/description/)

Given [100, 4, 200, 1, 3, 2],  
The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length: 4.

要求以 O(N) 的时间复杂度求解。

public int longestConsecutive(int[] nums) {  
 Map<Integer, Integer> countForNum = new HashMap<>();  
 for (int num : nums) {  
 countForNum.put(num, 1);  
 }  
 for (int num : nums) {  
 forward(countForNum, num);  
 }  
 return maxCount(countForNum);  
}  
  
private int forward(Map<Integer, Integer> countForNum, int num) {  
 if (!countForNum.containsKey(num)) {  
 return 0;  
 }  
 int cnt = countForNum.get(num);  
 if (cnt > 1) {  
 return cnt;  
 }  
 cnt = forward(countForNum, num + 1) + 1;  
 countForNum.put(num, cnt);  
 return cnt;  
}  
  
private int maxCount(Map<Integer, Integer> countForNum) {  
 int max = 0;  
 for (int num : countForNum.keySet()) {  
 max = Math.max(max, countForNum.get(num));  
 }  
 return max;  
}