- https://leetcode-cn.com/problems/two-sum/

求两个数的和

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，数组中同一个元素不能使用两遍。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]
```

示例 3：

```
输入：nums = [3,3], target = 6
输出：[0,1]
```

解法一：暴力破解

对每一个数 x ，在数组里寻找 target - x，由于 x 不能重复使用，所以只能在 x 后面的数寻找，代码如下：

```
private static int[] twoSum(int[] nums, int target) {
    for (int i = 0; i < nums.length; i++) {
        for (int j = i+1; j < nums.length; j++) {
            if (nums[i] + nums[j] == target) {
                return new int[] {i, j};
            }
        }
     }
    return new int[] {0};
}
```

复杂度分析

- 时间复杂度 O(N2) 其中 N 是数组中元素个数，最坏情况下数组中没两个数都要匹配一次。
- 空间复杂度：O(1)

方法二：借助 Map

借助 Map，循环一次，每次判断 map 中是否存在 target - x 这个值，如果有则从 map 中取出下标的值，如果没有则将值 x 和下标 i 加入 map。代码如下：

```
private static int[] twoSum2(int[] nums, int target) {
    Map<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < nums.length; i ++) {
        if (map.containsKey(target - nums[i])) {
            return new int[] {i, map.get(target - nums[i])};
        } else {
            map.put(nums[i], i);
        }
    }
    return new int[] {0};
}
```

复杂度分析

- 时间复杂度：O(N), N 为数组数字个数。
- 空间复杂度: O(N), N 是数组数字个数，是 Hash 表的开销。