Chapter 2: Array

N sum

Two Sum
Given an array of integers, return indices of the two numbers such that they add up to a specific target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

Examples:

```
Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,
return [0, 1].
```

Brute Force解法

- 穷举所有的组合（How？）- Nested Loop(双重循环)
- Time Complexity: O(n^2)
- Space Complexity: O(1)

Two-pass Hash Table 解法

- 空间换时间从而使得穷举所用时间复杂度更少
- Array的查询复杂度为: O(n)
- Hashmap的查询复杂度为:O(1)
- 如何快速找到所需元素的index？利用Hashmap来快速定位元素的index -> Mapping {value: index}
- {2: 0, 7: 1, 11: 2, 15: 3}
- Time Complexity: O(n) // O(2n)
- Space Complexity: O(n)

One-pass Hash Table 解法

- 遍历一遍数组就可以穷举出所有的组合，因为顺序没有关系 -》对排列组合的理解
- Time Complexity: O(n)
- Space Complexity: O(n)

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

```
    }
    throw new IllegalArgumentException("No two sum solution");
}
```

Difference between int and Integer?
int is **primitive type**
Integer is **object type**

2 Pointer解法

- **Sort数组** -》递增数组
- [1, 2, 5, 6, 7, 11], target = 9
- Head = 0, Tail = 5 -> 1 + 11 = 12 > 9
- Head = 0, Tail = 4 -> 1 + 7 = 8 < 9
- Head = 1, Tail = 4 -> 2 + 7 = 9 == 9
- 利用头尾指针快速定位到所需元素
- 缺陷：需要hashmap来保存index
- Time Complexity: O(nlogn)
- Space Complexity: O(n)

```
# 假设数组已经sort好
public int[] twoSum(int[] numbers, int target) {
	if (numbers == null || numbers.length == 0)
		return null;

	int i = 0;
	int j = numbers.length - 1;

	while (i < j) {
		int x = numbers[i] + numbers[j];
		if (x < target) {
			++i;
		} else if (x > target) {
			j--;
		} else {
			return new int[] { i + 1, j + 1 };
		}
	}
	return null;
}
```

2 Pointer with Binary Search

- [1, 2, 5, 6, 7, 11], target = 9
- Time Complexity: O(nlogn)
- Space Complexity: O(1)

3 Sum

Given an array nums of n integers, are there elements a, b, c in nums such that a + b + c = 0? Find all unique triplets in the array which gives the sum of **zero**.

Note:
The solution set must not contain duplicate triplets.

Example:

```
Given array nums = [-1, 0, 1, 2, -1, -4],

A solution set is:
[
  [-1, 0, 1],
  [-1, -1, 2]
]
```

Brute Force解法

- 穷举所有的组合（How？）-> Nested Loop(三重循环)
- Time Complexity: O(n^3)
- Space Complexity: O(1)

利用2 Sum解法（HashMap/Set）

- 穷举所有的组合（How？）
- 难点：如何处理duplicate elements
- Time Complexity: O(n^2)
- Space Complexity: O(1)

```
public List<List<Integer>> threeSum(int[] nums) {
    Arrays.sort(nums);
    List<List<Integer>> toreturn = new LinkedList<List<Integer>>();

    HashMap<Integer, Integer> hm = new HashMap<Integer, Integer>();
    HashSet<List<Integer>> check = new HashSet<List<Integer>>();

    for (int i = 0; i < nums.length; i++)   hm.put(nums[i], i);
    for (int i = 0; i < nums.length - 1; i++)
        for (int j = i + 1; j < nums.length; j++)
            if(hm.get(-nums[i]-nums[j]) != null && check.add(Arrays.asList(nums[i],
nums[j])) && hm.get(-nums[i]-nums[j]) > j)
                toreturn.add(Arrays.asList(nums[i], nums[j], -nums[i]-nums[j]));
    return toreturn;
}
```

利用 2 Point 解法

- Time Complexity: O(n^2)
- Space Complexity: O(1)

```java
public List<List<Integer>>  threeSum(int[] nums){
    List<List<Integer>> ans = new ArrayList<>();

    Arrays.sort(nums);
    for(int i = 0; i < nums.length-1; i++){
        if(i > 0 && nums[i] == nums[i-1]) continue;
        int begin = i+1;
        int end = nums.length-1;
        while(begin < end){
            if(nums[i] + nums[begin] + nums[end] == 0){
                List<Integer> list = new ArrayList<Integer>();
                list.add(nums[i]);
                list.add(nums[begin]);
                list.add(nums[end]);
                ans.add(list);
                begin++;
                end--;
                while(begin < end && nums[begin] == nums[begin-1])
                    begin++;
                while(begin < end && nums[end] == nums[end+1])
                    end--;
            }else if(nums[i] + nums[begin] + nums[end] > 0){
                end--;
            }else{
                begin++;
            }
        }
    }
    return ans;
}
```

Ref: TianTian Ji (2019-04 期学生)

[4 Sum]

Given an array nums of n integers and an integer target, are there elements a, b, c, and d in nums such that a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.

Note:
The solution set must not contain duplicate quadruplets.

Example:

```
Given array nums = [1, 0, -1, 0, -2, 2], and target = 0.

A solution set is:
[
  [-1,  0, 0, 1],
  [-2, -1, 1, 2],
  [-2,  0, 0, 2]
]
```

Brute Force解法

- 穷举所有的组合（How？）-> Nested Loop(四重循环)
- Time Complexity: O(n^4)
- Space Complexity: O(1)

利用2 Sum解法（HashMap）

- 穷举所有的组合（How？）
- Time Complexity: O(n^3)
- Space Complexity: O(1)

Generalized K sum solution:
https://leetcode.com/problems/4sum/discuss/161661/JAVAEasy-to-understand-of-kSums

其它相似问题：
Two Sum II - Input array is sorted

Subarray Sum Equals K

Given an array of integers and an integer k, you need to find the total number of **continuous subarrays** whose sum equals to k.

Note:
The length of the array is in range [1, 20,000].
The range of numbers in the array is [-1000, 1000] and the range of the integer k is [-1e7, 1e7].

Example:

```
Input:nums = [1,1,1], k = 2
Output: 2
```

[0, 1, 2, 3] 我们可以sort数组吗？

Brute Force解法
- 穷举所有的组合（How？）-> Nested Loop(三重循环)
- Time Complexity: O(n^3)
- Space Complexity: O(1)

Cumulative Sum(累计和)解法
- 空间换时间
- 使用cumulative sum array来节省重复计算Subarray Sum的时间
- Time Complexity: O(n^2)
- Space Complexity: O(n)

Cumulative Sum(累计和)解法 - HashMap
- 使用HashMap快速定位某个cumulative sum的位置(类似转换为2 Sum问题了)
- Time Complexity: O(n)
- Space Complexity: O(n)

[3, 4, 7, 2, -3, 1, 4, 2] k = 7

Init: Map {(0, 1)}, Sum 0 -> Count 0
3 -> Map {(0, 1), (3, 1)}, Sum 3 -> Count 0
4 -> Map {**(0, 1)**, (3, 1), (7, 1)}, Sum 7 -> Count 1
7 -> Map {(0, 1), (3, 1), **(7, 1)**, (14, 1)}, Sum 14 -> Count 2
2 -> Map {(0, 1), (3, 1), (7, 1), (14, 1), (16, 1)}, Sum 16 -> Count 2
-3 ->Map {(0, 1), (3, 1), (7, 1), (14, 1), (16, 1), (13, 1)}, Sum 13 -> Count 2
1 ->Map {(0, 1), (3, 1), **(7, 1)**, (14, 2), (16, 1), (13, 1)}, Sum 14 -> Count 3
4 ->Map {(0, 1), (3, 1), (7, 1), (14, 2), (16, 1), (13, 1), (18, 1)}, Sum 18 -> Count 3
2 ->Map {(0, 1), (3, 1), (7, 1), (14, 2), (16, 1), **(13, 1)**, (18, 1)}, Sum 20 -> Count 4

红色的 pair 表示能形成 k = 7 的 continuous array

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

Solution Ref: https://leetcode.com/problems/subarray-sum-equals-k/solution/

TODO:
https://leetcode.com/problems/contiguous-array/

Missing Number

Missing Number
Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.

Examples:

```
Input: [3,0,1]
Output: 2

Input: [9,6,4,2,3,5,7,0,1]
Output: 8
```

Brute Force解法
- N次遍历数组直到找到不存在的数字 -》双重循环
- Time Complexity: O(n^2)
- Space Complexity: O(1)

HashSet解法
- 空间换时间
- 使用**HashSet**来保存Input数组中的数字
- Time Complexity: O(n)
- Space Complexity: O(n)

HashSet 查询时间 -> O(1)

Sorting解法
- Sort数组，找出不连续数字即为Missing Number
- Time Complexity: O(nlogn)
- Space Complexity: O(1)

Sorting; in-place(O(1))

Java pass by value vs. pass by reference?
Java一直都是 pass by value
Note: java中数组的值是 内存地址
https://stackoverflow.com/questions/12757841/are-arrays-passed-by-value-or-passed-by-reference-in-java

头尾指针问题（Head & Tail Pointer）

3 Sum, 4 Sum（参考之前课件）

3Sum Closest
Given an array nums of n integers and an integer target, find three integers in nums such that the sum is closest to target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

Example:

```
Given array nums = [-1, 2, 1, -4], and target = 1.
The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
```

Brute Force解法

- 如何穷举出所有组合？-》三重循环
- Time Complexity: O(n^3)
- Space Complexity: O(1)

头尾指针解法

- Sorting
- Current, Head, Tail（三个指针来遍历）
- Time Complexity: O(n^2)
- Space Complexity: O(1)

```
public int threeSumClosest(int[] num, int target) {
        int result = num[0] + num[1] + num[num.length - 1];
        Arrays.sort(num);
        for (int i = 0; i < num.length - 2; i++) {
            int start = i + 1, end = num.length - 1;
            while (start < end) {
                int sum = num[i] + num[start] + num[end];
                if (sum > target) {
                    end--;
                } else {
                    start++;
                }
                if (Math.abs(sum - target) < Math.abs(result - target)) {
                    result = sum;
                }
            }
        }
        return result;
    }
}
```

Reverse Vowels of a String
Write a function that takes a string as input and reverse only the vowels of a string. (a,e,i,o,u)

Example:

```
Input: "hello"
Output: "holle"

Input: "leetcode"
Output: "leotcede"
```

头尾指针解法

- 使用一个HashSet/String来存vowels，利用Head和Tail指针遍历String并完成互换
- 考验Coding
- Time Complexity: O(n)
- Space Complexity: O(1)

快慢指针（Fast & Slow Pointer）

Remove Duplicates from Sorted Array
Given a **sorted** array nums, remove the duplicates in-place such that each element appear only once and return the new length.

Do not allocate extra space for another array, **you must do this by modifying the input array in-place with O(1) extra memory.**

Example:

```
Given nums = [1,1,2],
Your function should return length = 2, with the first two elements of nums being 1
and 2 respectively.
It doesn't matter what you leave beyond the returned length.

Given nums = [0,0,1,1,1,2,2,3,3,4],
Your function should return length = 5, with the first five elements of nums being
modified to 0, 1, 2, 3, and 4 respectively.
It doesn't matter what values are set beyond the returned length.
```

快慢指针解法

- Slow指针指向不重复数组的最后一位，Fast指针一直向后遍历，如果发现不重复数字，则同时移动Slow和Fast指针
- Time Complexity: O(n)
- Space Complexity: O(1)

Similar Questions:
Remove Element
Remove Duplicates from Sorted Array II（solution ref）

Is Subsequence

Given a string s and a string t, check if s is subsequence of t.

You may assume that there is only lower case English letters in both s and t. t is potentially a very long (length ~= 500,000) string, and s is a short string (<=100).

Example:

```
s = "abc", t = "ahbgdc"
Return true

s = "axc", t = "ahbgdc"
Return false
```

快慢指针解法

- 慢指针遍历string s, 快指针遍历string t, 直到其中一个指针遍历完，则可以分辨出s是否为t的子序列
- Time Complexity: O(n)
- Space Complexity: O(1)

```
public static boolean isSubsequence(String s, String t) {
	if (s.length() == 0) {
		return true;
	}
	char[] ch1 = s.toCharArray();
	char[] ch2 = t.toCharArray();
	int i = 0;
	int j = 0;
	while (i < ch1.length && j < ch2.length ) {
		if (ch1[i] != ch2[j]) {
			j++;
		} else {
			i++;
			j++;
		}
	}
	return i == ch1.length;
}
```

Longest Word in Dictionary through Deleting

Given a string and a string dictionary, find the longest string in the dictionary that can be formed by deleting some characters of the given string. If there are more than one possible results, return the longest word with **the smallest lexicographical order**. If there is no possible result, return the empty string.

Note:

- All the strings in the input will only contain lower-case letters.
- The size of the dictionary won't exceed 1,000.
- The length of all the strings in the input won't exceed 1,000.

Example:

```
Input:
s = "abpcplea", d = ["ale","apple","monkey","plea"]

Output:
"apple"

Input:
s = "abpcplea", d = ["a","b","c"]

Output:
"a"
```

Brute Force解法

- 怎么穷举出所有可能性？-》参见之后的组合优化章节
- Time Complexity: O(2^s.length)
- Space Complexity: -

快慢指针解法

- 用Is Subsequence的解法
- Time Complexity: O(t*s + s*logs)
- Space Complexity: O(1)

滑动窗口（Sliding Window）

Longest Substring Without Repeating Characters

Given a string, find the length of the longest substring without repeating characters.

Example:

```
Input: "abcabcbb"
Output: 3
Explanation: The answer is "abc", which the length is 3.

Input: "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

```
Input: "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3. Note that the answer must
be a substring, "pwke" is a subsequence and not a substring.
```

Brute Force解法

- 穷举所有的substring，判断每个substring是不是没有重复字符串 =》双重循环
- Time Complexity: O(n^2)
- Space Complexity: O(1)

滑动窗口解法

- 两个指针Maintain一个窗口，并使用HashSet来存滑动窗口内的字符
- Time Complexity:
- Space Complexity:

Init: “pwwkew”
i = 0, j = 0, set = {}, ans = 0
i = 0, j = 1, set = {p}, ans = 1
i = 0, j = 2, set = {p, w}, ans = 2
i = 1, j = 2, set = {w}, ans = 2
i = 2, j = 2, set = {}, ans = 2
i = 2, j = 3, set = {w}, ans = 2
i = 2, j = 4, set = {w, k}, ans = 2
i = 2, j = 5, set = {w, k, e}, ans = 3
i = 3, j = 5, set = {k, e}, ans = 3
i = 3, j = 6, set = {k, e, w}, ans = 3

```java
public int lengthOfLongestSubstring(String s) {
    int n = s.length();
    Set<Character> set = new HashSet<>();
    int ans = 0, i = 0, j = 0;
    while (i < n && j < n) {
        // try to extend the range [i, j]
        if (!set.contains(s.charAt(j))){
            set.add(s.charAt(j++));
            ans = Math.max(ans, j - i);
        }
        else {
            set.remove(s.charAt(i++));
        }
    }
    return ans;
}
```

Permutation in String

Given two strings s1 and s2, write a function to return true if s2 contains the permutation of s1. In other words, one of the first string's permutations is the substring of the second string.

Example:

```
Input:s1 = "ab" s2 = "eidbaooo"
Output:True
Explanation: s2 contains one permutation of s1 ("ba").

Input:s1= "ab" s2 = "eidboaoo"
Output: False
```

Brute Force解法:

- 穷举所有string s2的substring(如何穷举 -> 见后续组合数学)，判别substring是不是string s1的permutation
- Time Complexity: O(n!)

滑动窗口解法:

- 固定长度则不需要用两个指针来Maintain窗口，一个指针即可，并使用 HashMap 来存滑动窗口内的字符
- 可以使用字母数组：[a, b, c, d, … , z]

```
public boolean checkInclusion(String s1, String s2) {
    if (s1.length() > s2.length())
        return false;
    int[] s1map = new int[26];
    int[] s2map = new int[26];
    for (int i = 0; i < s1.length(); i++) {
        s1map[s1.charAt(i) - 'a']++;
        s2map[s2.charAt(i) - 'a']++;
    }
    for (int i = 0; i < s2.length() - s1.length(); i++) {
        if (matches(s1map, s2map))
            return true;
        s2map[s2.charAt(i + s1.length()) - 'a']++;
        s2map[s2.charAt(i) - 'a']--;
    }
    return matches(s1map, s2map);
}
public boolean matches(int[] s1map, int[] s2map) {
    for (int i = 0; i < 26; i++) {
        if (s1map[i] != s2map[i])
            return false;
    }
    return true;
}
```

Extra

[Trapping Rain Water]
Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

Example:

```
Input: [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
```

Brute Force解法

- 找到每一个bar的左边和右边最高的bar，以此算出当前这个bar可以容纳多少水
- Time Complexity: O(n^2)
- Space Complexity: O(1)

头尾指针解法

- 如何省去每一次都需要向左和向右遍历数组？
- 维系Head, Tail指针，同时用两个变量来记录当前Head和Tail指针所遇到的最高的height

```
int trap(vector<int>& height)
{
    int left = 0, right = height.size() - 1;
    int ans = 0;
    int left_max = 0, right_max = 0;
    while (left < right) {
        if (height[left] < height[right]) {
            height[left] >= left_max ? (left_max = height[left]) : ans += (left_max
- height[left]);

            ++left;
        } else {
            height[right] >= right_max ? (right_max = height[right]) : ans +=
(right_max - height[right]);

            --right;
        }
    }
    return ans;
}
```

[First Missing Positive]
Given an unsorted integer array, find the smallest missing positive integer.

Note:
Your algorithm should run in O(n) time and uses constant extra space.

Examples:

```
Input: [1,2,0]
Output: 3

Input: [3,4,-1,1]
Output: 2

Input: [7,8,9,11,12]
Output: 1
```

数组坐标与元素对应解法

- 挪动数组元素到其对应坐标点
- 大于数组个数整数可以不考虑，负数也可以不考虑
- Time Complexity: O(n)
- Space Complexity: O(1)

```
int firstMissingPositive(int A[], int n)
{
    for(int i = 0; i < n; ++ i)
        while(A[i] > 0 && A[i] <= n && A[A[i] - 1] != A[i])
            swap(A[i], A[A[i] - 1]);

    for(int i = 0; i < n; ++ i)
        if(A[i] != i + 1)
            return i + 1;

    return n + 1;
}
```

https://leetcode.com/problems/first-missing-positive/discuss/17071/My-short-c++-solution-O(1)-space-and-O(n)-time

Similar:
Find the Duplicate Number
https://leetcode.com/problems/set-mismatch

Find All Numbers Disappeared in an Array
Given an array of integers where 1 ≤ a[i] ≤ n (n = size of array), some elements appear twice and others appear once. Find all the elements of [1, n] inclusive that do not appear in this array.

Could you do it without extra space and in O(n) runtime? You may assume the returned list does not count as extra space.

Examples:

```
Input:
```

```
[4,3,2,7,8,2,3,1]

Output:
[5,6]
```

Brute Force解法

- N次遍历数组直到找到不存在的数字 -》双重循环
- Time Complexity: O(n^2)
- Space Complexity: O(1)

HashSet解法

- 空间换时间
- 使用HashSet来保存Input数组中的数字
- Time Complexity: O(n)
- Space Complexity: O(n)

我们可以使用Sorting吗？

数组坐标与元素对应解法

- 挪动数组元素到其对应坐标点
- Time Complexity: O(n)
- Space Complexity: O(1)

Init: [4,3,2,7,8,2,3,1]
4 -> nums[0] != 0 + 1 -> swap(4, [4,3,2,7,8,2,3,1])

- [4,3,2,4,8,2,3,1], i = 7
- [4,3,2,4,8,2,7,1], i = 3
- [4,3,3,4,8,2,7,1], i = 2
- [4,2,3,4,8,2,7,1], i = 3

2 -> nums[1] == 1 + 1
3 -> nums[2] == 2 + 1
4 -> nums[3] == 3 + 1
8 -> nums[4] != 4 + 1 -> swap(8, [4,2,3,4,8,2,7,1])

- [4,2,3,4,8,2,7,8], i = 1
- [1,2,3,4,8,2,7,8], i = 4

2 -> nums[5] != 5 + 1 -> swap(2, [1,2,3,4,8,2,7,8])

- [1,2,3,4,8,2,7,8]

7 -> nums[6] == 6 + 1
8 -> nums[7] == 7 + 1

```
public List<Integer> findDisappearedNumbers(int[] nums) {
  List<Integer> res = new ArrayList<Integer>();
  if(nums==null || nums.length==0) return res;
  int len = nums.length;
  for(int i=0; i<len; i++) if(nums[i]!=i+1) swap(nums[i], nums);
  for(int i=0; i<len; i++) if(nums[i]!=i+1) res.add(i+1);
```

```
  return res;
}

public void swap(int val, int[] nums){
  while(val != nums[val-1]){
     int temp = nums[val-1];
     nums[val-1] = val;
     val = temp;
  }
}
```

https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/discuss/159325/Super-Simple-Java-O(n)-O(1)-space