1. Sliding Window Algorithm

LeetCode: 3. Longest Substring Without Repeating Characters

[**[LeetCode] 3. Longest Substring Without Repeating Characters 解题思路**](http://www.cnblogs.com/TonyYPZhang/p/5061202.html)

Given a string, find the length of the longest substring without repeating characters. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1.

问题：找到字符串中最长的无重复字符的子字符串。

这题是求最长连续子字符串，可以使用 滑动窗口算法(**Slide Window** Algorithm)求解，和 [Minimum Size Subarray Sum](http://www.cnblogs.com/TonyYPZhang/p/5059248.html) 相似。

设下标 l 和 r, 把左开右闭 [l, r) 想象成一个窗口。

* 当 s[r] 和窗口内字符重复时， 则 l 向右滑动，缩小窗口。
* 当s[r] 和窗口内字符不重复时，则 r 向右滑动，扩大窗口，此时窗口内的字符串一个无重复子字符串。

在所有的无重复子字符串中，找到长度最大的，即为原问题解。

[复制代码](javascript:void(0);)

int lengthOfLongestSubstring(string s) {

int l = 0;

int r = 1;

unordered\_set<int> setv;

setv.insert(s[l]);

int longest = 1;

while (r < s.size()) {

if (setv.count(s[r]) != 0) {

setv.erase(s[l]);

l++;

}else{

setv.insert(s[r]);

r++;

longest = max(longest, (int)setv.size());

}

}

return longest;

}

[复制代码](javascript:void(0);)

2. int型的数值范围是 -2147483648～2147483647

LeetCode: 7. Reverse Integer

3. 正负号标记法

**解法I 正负号标记法（一趟遍历）**

参考LeetCode Discuss：https://discuss.leetcode.com/topic/64735/java-simple-solution

遍历nums，记当前数字为n（取绝对值），将数字n视为下标（因为a[i]∈[1, n]）

当n首次出现时，nums[n - 1]乘以-1

当n再次出现时，则nums[n - 1]一定＜0，将n加入答案

## **Python代码：**

**class** **Solution**(object):

**def** **findDuplicates**(self, nums):

"""

:type nums: List[int]

:rtype: List[int]

"""

ans = []

**for** n **in** nums:

**if** nums[abs(n) - 1] < 0:

ans.append(abs(n))

**else**:

nums[abs(n) - 1] \*= -1

**return** ans

<http://bookshadow.com/weblog/2016/10/25/leetcode-find-all-duplicates-in-an-array/>

4. 全排列问题

## **Permutations**

Given a collection of distinct numbers, return all possible  
permutations.

For example, [1,2,3] have the following permutations: [1,2,3],  
[1,3,2], [2,1,3], [2,3,1], [3,1,2], and [3,2,1].

思路： 典型的backtracking，注意list在递归的过程一直在变化，所以每次加入的list需要重新创建。**result.add(new ArrayList<Integer>(list))**;

时间复杂度：O(n!)  
空间复杂度：O(n)

**public** **class** **Solution** {

**public** **List**<**List**<Integer>> permute(**int**[] nums) {

**List**<**List**<Integer>> result = **new** ArrayList<**List**<Integer>>();

**List**<Integer> **list** = **new** ArrayList<Integer>();

helper(result, **list**, nums);

**return** result;

}

**private** void helper(**List**<**List**<Integer>> result, **List**<Integer> **list**, **int**[] nums) {

**if** (**list**.size() == nums.length) {

**if** (!result.contains(**list**)) {

result.add(**new** ArrayList<Integer>(**list**));

**return**;

}

}

**for** (**int** i = 0; i < nums.length; i++) {

**if** (**list**.contains(nums[i])) {

**continue**;

}

**list**.add(nums[i]);

helper(result,**list**,nums);

**list**.remove(**list**.size() - 1);

}

}

}

5. 对于这种玩字符串，并且是求极值的问题，十有八九都是用dp来解的