# Recent Questions

## 283 Move Zeroes

Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

For example, given nums = [0, 1, 0, 3, 12], after calling your function, nums should be [1, 3, 12, 0, 0].

**Note**:

1. You must do this **in-place** without making a copy of the array.
2. Minimize the total number of operations.

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

**public** **void** moveZeroes(**int**[] nums) {

**int** temp;

**for** (**int** lastNonZeroFoundAt = 0, cur = 0; cur < nums.length; cur++) {

**if** (nums[cur] != 0) {

temp = nums[lastNonZeroFoundAt];

nums[lastNonZeroFoundAt++] = nums[cur];

nums[cur] = temp;

}

}

}

}

思路：重点是非零数字，把他们依次移到前面，原位置置为零即可，如果本来就是相应位置，则不动。O(n)

## 301 Remove Invalid Parentheses

Remove the minimum number of invalid parentheses in order to make the input string valid. Return all possible results.

Note: The input string may contain letters other than the parentheses ( and ).

**Examples:**

"()())()" -> ["()()()", "(())()"]

"(a)())()" -> ["(a)()()", "(a())()"]

")(" -> [""]

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

**public** List<String> removeInvalidParentheses(String s) {

List<String> ans = **new** ArrayList<>();

remove(s, ans, 0, 0, **new** **char**[] { '(', ')' });

**return** ans;

}

**public** **void** remove(String s, List<String> ans, **int** last\_i, **int** last\_j,

**char**[] par) {

**for** (**int** stack = 0, i = last\_i; i < s.length(); ++i) {

**if** (s.charAt(i) == par[0])

stack++;

**if** (s.charAt(i) == par[1])

stack--;

**if** (stack >= 0)

**continue**;

**for** (**int** j = last\_j; j <= i; ++j)//从上个去除位置开始找

**if** (s.charAt(j) == par[1]

&& (j == last\_j || s.charAt(j - 1) != par[1]))//是要去除的符号且（是起始位置或前面不是“）”，以免重复）。

remove(s.substring(0, j) + s.substring(j + 1, s.length()),

ans, i, j, par);

**return**;

}

String reversed = **new** StringBuilder(s).reverse().toString();

**if** (par[0] == '(')

remove(reversed, ans, 0, 0, **new** **char**[] { ')', '(' });//反过来时，正反括号也反

**else**

ans.add(reversed);//正反都完成，得到答案

}

}

思路：正扫一遍，去掉多余的），去掉任意一个，对于连续出现的“）”，总是去掉第一个。然后把字符串反过来，再来一次。O(n^2)

## 15 [3Sum](https://leetcode.com/problems/3sum)

Given an array *S* of *n* integers, are there elements *a*, *b*, *c* in *S* 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.

For example, given array S = [-1, 0, 1, 2, -1, -4],

A solution set is:

[ [-1, 0, 1],

[-1, -1, 2]]

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

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

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

**if**(nums.length < 3) **return** result;

Arrays.*sort*(nums);

**int** i = 0;

**while**(i < nums.length - 2) {

**if**(nums[i] > 0) **break**;

**int** j = i + 1;

**int** k = nums.length - 1;

**while**(j < k) {

**int** sum = nums[i] + nums[j] + nums[k];

**if**(sum == 0) result.add(Arrays.*asList*(nums[i], nums[j], nums[k]));

**if**(sum <= 0) **while**(nums[j] == nums[++j] && j < k);

**if**(sum >= 0) **while**(nums[k--] == nums[k] && j < k);

}

**while**(nums[i] == nums[++i] && i < nums.length - 2);

}

**return** result;

}

}

思路1：先排序，然后查每一个数后面两数之和为当前数相反数的所有可能（夹逼法）。O(n^2)

思路2（未实现）：不排序，先做2数和的倒排索引，查每一个数是否有相应的2数和为其相反数。O(n^3)

## 17 Letter Combinations of a Phone Number

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.



**Input:**Digit string "23"

**Output:** ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

**Note:**  
Although the above answer is in lexicographical order, your answer could be in any order you want.

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

**public** List<String> letterCombinations(String digits) {

List<String> resultList = **new** ArrayList<String>();

if(digits.length() == 0)

return resultList;

Map<String, List<String>> digitalMap = **new** HashMap<String, List<String>>();

digitalMap.put("2",

**new** ArrayList<String>(Arrays.*asList*("a", "b", "c")));

digitalMap.put("3",

**new** ArrayList<String>(Arrays.*asList*("d", "e", "f")));

digitalMap.put("4",

**new** ArrayList<String>(Arrays.*asList*("g", "h", "i")));

digitalMap.put("5",

**new** ArrayList<String>(Arrays.*asList*("j", "k", "l")));

digitalMap.put("6",

**new** ArrayList<String>(Arrays.*asList*("m", "n", "o")));

digitalMap.put("7",

**new** ArrayList<String>(Arrays.*asList*("p", "q", "r", "s")));

digitalMap.put("8",

**new** ArrayList<String>(Arrays.*asList*("t", "u", "v")));

digitalMap.put("9",

**new** ArrayList<String>(Arrays.*asList*("w", "x", "y", "z")));

*getLetterCombinations*(digits.toCharArray(), **new** StringBuilder(), 0,

digitalMap, resultList);

**return** resultList;

}

**private** **static** **void** getLetterCombinations(**char**[] c, StringBuilder sbSoFar,

**int** curP, Map<String, List<String>> digitalMap,

List<String> resultList) {

**if** (curP == c.length) {

resultList.add(sbSoFar.toString());

**return**;

}

**char** cur = c[curP];

**for** (String letter : digitalMap.get(String.*valueOf*(cur))) {

StringBuilder sb = **new** StringBuilder();

sb.append(sbSoFar);

sb.append(letter);

*getLetterCombinations*(c, sb, curP + 1, digitalMap, resultList);

}

}

}

思路：backtracking，每一种可能都添加到结果中。O(n^k)

**class** Solution {

**public** List<String> letterCombinations(String digits) {

LinkedList<String> ans = **new** LinkedList<String>();

String[] mapping = **new** String[] { "0", "1", "abc", "def", "ghi", "jkl",

"mno", "pqrs", "tuv", "wxyz" };

ans.add("");

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

**int** x = Character.*getNumericValue*(digits.charAt(i));

**while** (ans.peek().length() == i) {

String t = ans.remove();

**for** (**char** s : mapping[x].toCharArray())

ans.add(t + s);

}

}

**return** ans;

}

}

思路：backtracking，每一种可能都添加到结果中。O(n^k)

## 67 Add Binary

Given two binary strings, return their sum (also a binary string).

For example,  
a = "11"  
b = "1"  
Return "100".

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

**public** String addBinary(String a, String b) {

StringBuilder sb = **new** StringBuilder();

**int** i = a.length() - 1, j = b.length() -1, carry = 0;

**while** (i >= 0 || j >= 0) {

**int** sum = carry;

**if** (j >= 0) sum += b.charAt(j--) - '0';

**if** (i >= 0) sum += a.charAt(i--) - '0';

sb.append(sum % 2);

carry = sum / 2;

}

**if** (carry != 0) sb.append(carry);

**return** sb.reverse().toString();

}

}

思路：小学加法竖式，从低位加到高们（考虑进位）。

## 311 [Sparse Matrix Multiplication](https://leetcode.com/problems/sparse-matrix-multiplication)

Given two [sparse matrices](https://en.wikipedia.org/wiki/Sparse_matrix) **A** and **B**, return the result of **AB**.

You may assume that **A**'s column number is equal to **B**'s row number.

**Example:**

**A** = [

[ 1, 0, 0],

[-1, 0, 3]

]

**B** = [

[ 7, 0, 0 ],

[ 0, 0, 0 ],

[ 0, 0, 1 ]

]

| 1 0 0 | | 7 0 0 | | 7 0 0 |

**AB** = | -1 0 3 | x | 0 0 0 | = | -7 0 3 |

| 0 0 1 |

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

**public** **int**[][] multiply(**int**[][] A, **int**[][] B) {

**int** m = A.length, n = A[0].length, nB = B[0].length;

**int**[][] C = **new** **int**[m][nB];

**for** (**int** i = 0; i < m; i++)

**for** (**int** k = 0; k < n; k++)

**if** (A[i][k] != 0) {

**for** (**int** j = 0; j < nB; j++)

**if** (B[k][j] != 0)

C[i][j] += A[i][k] \* B[k][j];

**return** C;

}

}

思路：暴力，无它。

## 621 Task Scheduler

Given a char array representing tasks CPU need to do. It contains capital letters A to Z where different letters represent different tasks.Tasks could be done without original order. Each task could be done in one interval. For each interval, CPU could finish one task or just be idle.

However, there is a non-negative cooling interval **n** that means between two **same tasks**, there must be at least n intervals that CPU are doing different tasks or just be idle.

You need to return the **least** number of intervals the CPU will take to finish all the given tasks.

**Example 1:**

**Input:** tasks = ["A","A","A","B","B","B"], n = 2

**Output:** 8

**Explanation:** A -> B -> idle -> A -> B -> idle -> A -> B.

**Note:**

1. The number of tasks is in the range [1, 10000].
2. The integer n is in the range [0, 100].

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

**public** **int** leastInterval(**char**[] tasks, **int** n) {

**int**[] map = **new** **int**[26];

**for** (**char** c : tasks)

map[c - 'A']++;

Arrays.*sort*(map);

**int** time = 0;

**while** (map[25] > 0) {

**int** i = 0;

**while** (i <= n) {

**if** (map[25] == 0)

**break**;

**if** (i < 26 && map[25 - i] > 0)

map[25 - i]--;

time++;

i++;

}

Arrays.*sort*(map);

}

**return** time;

}

}

思路：尽量使用别的（数量多的）任务填充idle空缺，如果没有更多任务则idle。每次填完一个周期，重新排序，重复弄直到没有任务。因排序是有限集（26个），所以为常数。O(n)

## 1 [Two Sum](https://leetcode.com/problems/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.

**Example:**

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

Because nums[**0**] + nums[**1**] = 2 + 7 = 9,

return [**0**, **1**].

**class** Solution {

**public** **int**[] twoSum(**int**[] nums, **int** target) {

Map<Integer, Integer> m = **new** HashMap<Integer,Integer>();

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

**int** num = target - nums[i];

**if**(m.containsKey(num))

**return** **new** **int**[]{m.get(num), i };

m.put(nums[i], i);

}

**throw** **new** RuntimeException("no answer!");

}

}

思路：查找时，建立索引（Hash查找）或进行排序（二分查找）。本题缓存可在找的过程中建立索引，故一个循环可以求出解（总是使用未使用元素查找使用元素，可以保证每一对都被检索到）。Indexing/ordering is the first step to search questions.

## 76 [Minimum Window Substring](https://leetcode.com/problems/minimum-window-substring)

Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

For example,  
**S** = "ADOBECODEBANC"  
**T** = "ABC"

Minimum window is "BANC".

**Note:**  
If there is no such window in S that covers all characters in T, return the empty string "".

If there are multiple such windows, you are guaranteed that there will always be only one unique minimum window in S.

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

**public** String minWindow(String s, String t) {

**int**[] map = **new** **int**[256];

**for** (**char** c : t.toCharArray())

map[c]++;

**int** counter = t.length(), begin = 0, end = 0, d = Integer.***MAX\_VALUE***,

head = 0;

**while** (end < s.length()) {

**if** (map[s.charAt(end++)]-- > 0)

counter--; // in t

**while** (counter == 0) { // valid

**if** (end - begin < d) {

head = begin;

d = end - head;

}

**if** (map[s.charAt(begin++)]++ == 0)

counter++; // make it invalid

}

}

**return** d == Integer.***MAX\_VALUE*** ? "" : s.substring(head, head + d);

}

}

思路：Sliding Window，先找到包含T的第一个字串，然后把左边界缩小到最小但包含T的状态，这是第一个Candidate。然后左边界前移，重复第一步，找到所有Candidates。最终找到最小值并返回。O(n)

## 125 Valid Palindrome

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

For example,  
"A man, a plan, a canal: Panama" is a palindrome.  
"race a car" is *not* a palindrome.

**Note:**  
Have you consider that the string might be empty? This is a good question to ask during an interview.

For the purpose of this problem, we define empty string as valid palindrome.

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

**public** **boolean** isPalindrome(String s) {

**if** (s.isEmpty()) {

**return** **true**;

}

**int** head = 0, tail = s.length() - 1;

**char** cHead, cTail;

**while** (head < tail) {

cHead = s.charAt(head);

cTail = s.charAt(tail);

**if** (!Character.*isLetterOrDigit*(cHead)) {

head++;

} **else** **if** (!Character.*isLetterOrDigit*(cTail)) {

tail--;

} **else** {

**if** (Character.*toLowerCase*(cHead) != Character

.*toLowerCase*(cTail)) {

**return** **false**;

}

head++;

tail--;

}

}

**return** **true**;

}

}

思路：从两头进行比较，忽略非字母数字的字符。O(n)

## 253 Meeting Rooms II

Given an array of meeting time intervals consisting of start and end times [[s1,e1],[s2,e2],...] (si < ei), find the minimum number of conference rooms required.

For example,  
Given [[0, 30],[5, 10],[15, 20]],  
return 2.

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

**public** **int** minMeetingRooms(Interval[] intervals) {

**int**[] starts = **new** **int**[intervals.length];

**int**[] ends = **new** **int**[intervals.length];

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

starts[i] = intervals[i].start;

ends[i] = intervals[i].end;

}

Arrays.*sort*(starts);

Arrays.*sort*(ends);

**int** rooms = 0;

**int** endsItr = 0;

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

**if** (starts[i] < ends[endsItr])

rooms++;

**else**

endsItr++;

}

**return** rooms;

}

}

思路：如果需要更多会议室，则结束点大于另一个会议的起始点。排序起始和终止点，找出结束点大于起始点（除自身）的个数。O(nlogn)

## 273 [Integer to English Words](https://leetcode.com/problems/integer-to-english-words)

Convert a non-negative integer to its english words representation. Given input is guaranteed to be less than 231 - 1.

For example,

123 -> "One Hundred Twenty Three"

12345 -> "Twelve Thousand Three Hundred Forty Five"

1234567 -> "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"

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

**private** **final** String[] LESS\_THAN\_20 = { "", "One", "Two", "Three", "Four",

"Five", "Six", "Seven", "Eight", "Nine", "Ten", "Eleven", "Twelve",

"Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen",

"Eighteen", "Nineteen" };

**private** **final** String[] TENS = { "", "Ten", "Twenty", "Thirty", "Forty",

"Fifty", "Sixty", "Seventy", "Eighty", "Ninety" };

**private** **final** String[] THOUSANDS = { "", "Thousand", "Million", "Billion" };

**public** String numberToWords(**int** num) {

**if** (num == 0)

**return** "Zero";

**int** i = 0;

String words = "";

**while** (num > 0) {

**if** (num % 1000 != 0)

words = helper(num % 1000) + THOUSANDS[i] + " " + words;

num /= 1000;

i++;

}

**return** words.trim();

}

**private** String helper(**int** num) {

**if** (num == 0)

**return** "";

**else** **if** (num < 20)

**return** LESS\_THAN\_20[num] + " ";

**else** **if** (num < 100)

**return** TENS[num / 10] + " " + helper(num % 10);

**else**

**return** LESS\_THAN\_20[num / 100] + " Hundred " + helper(num % 100);

}

}

思路：根据英语特点，1000及以上3个数分一组，1000以内的100、20分别是两条线。每组对应几种情况。O(n)

## 297 [Serialize and Deserialize Binary Tree](https://leetcode.com/problems/serialize-and-deserialize-binary-tree)

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

For example, you may serialize the following tree

1

/ \

2 3

/ \

4 5

as "[1,2,3,null,null,4,5]", just the same as [how LeetCode OJ serializes a binary tree](https://leetcode.com/faq/#binary-tree). You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.

**Note:** Do not use class member/global/static variables to store states. Your serialize and deserialize algorithms should be stateless.

**public** **class** Codec {

**public** String serialize(TreeNode root) {

StringBuilder sb = **new** StringBuilder();

*serialize*(root, sb);

**return** sb.toString().trim();

}

**private** **static** **void** serialize(TreeNode root, StringBuilder sb) {

**if** (root == **null**) {

sb.append("# ");

**return**;

}

sb.append(root.val);

sb.append(" ");

*serialize*(root.left, sb);

*serialize*(root.right, sb);

}

**public** TreeNode deserialize(String data) {

String[] vals = data.split(" ");

**return** *deserialize*(**new** ValHolder(vals, -1));

}

**class** ValHolder {

String[] vals;

**int** i;

**int** n;

**public** ValHolder(String[] vals, **int** i) {

**this**.vals = vals;

**this**.i = i;

n = vals.length;

}

}

**private** **static** TreeNode deserialize(ValHolder vh) {

String cur = vh.vals[++vh.i];

**if** ("#".equals(cur))

**return** **null**;

TreeNode root = **new** TreeNode(Integer.*valueOf*(cur));

**if** (vh.i < vh.n - 1)

root.left = *deserialize*(vh);

**if** (vh.i < vh.n - 1)

root.right = *deserialize*(vh);

**return** root;

}

}

思路：先序遍历存入数组输出，每到叶结点添加#以区分。恢复时也使用先序遍历，如果遇到#则说明上个结点为叶子结点。O(n)

## 314 [Binary Tree Vertical Order Traversal](https://leetcode.com/problems/binary-tree-vertical-order-traversal)

Given a binary tree, return the *vertical order* traversal of its nodes' values. (ie, from top to bottom, column by column).

If two nodes are in the same row and column, the order should be from **left to right**.

**Examples:**

Given binary tree [3,9,20,null,null,15,7],

3

/\

/ \

9 20

/\

/ \

15 7

return its vertical order traversal as:

[ [9],

[3,15],

[20],

[7]]

Given binary tree [3,9,8,4,0,1,7],

3

/\

/ \

9 8

/\ /\

/ \/ \

4 01 7

return its vertical order traversal as:

[ [4],

[9],

[3,0,1],

[8],

[7]]

Given binary tree [3,9,8,4,0,1,7,null,null,null,2,5] (0's right child is 2 and 1's left child is 5),

3

/\

/ \

9 8

/\ /\

/ \/ \

4 01 7

/\

/ \

5 2

return its vertical order traversal as:

[ [4],

[9,5],

[3,0,1],

[8,2],

[7]]

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

**public** List<List<Integer>> verticalOrder(TreeNode root) {

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

**if** (root == **null**)

**return** cols;

**int**[] range = **new** **int**[] { 0, 0 };

getRange(root, range, 0);

**for** (**int** i = range[0]; i <= range[1]; i++)

cols.add(**new** ArrayList<Integer>());

Queue<TreeNode> queue = **new** LinkedList<>();

Queue<Integer> colQueue = **new** LinkedList<>();

queue.add(root);

colQueue.add(-range[0]);

**while** (!queue.isEmpty()) {

TreeNode node = queue.poll();

**int** col = colQueue.poll();

cols.get(col).add(node.val);

**if** (node.left != **null**) {

queue.add(node.left);

colQueue.add(col - 1);

}

**if** (node.right != **null**) {

queue.add(node.right);

colQueue.add(col + 1);

}

}

**return** cols;

}

**public** **void** getRange(TreeNode root, **int**[] range, **int** col) {

**if** (root == **null**) {

**return**;

}

range[0] = Math.*min*(range[0], col);

range[1] = Math.*max*(range[1], col);

getRange(root.left, range, col - 1);

getRange(root.right, range, col + 1);

}

}

思路：假设根为0，则左/右结点为-1/1，遍历找出最左和最右的范围，建立相应的List，然后再遍历，把每个结点放入对应的List。

## 639 [Decode Ways II](https://leetcode.com/problems/decode-ways-ii)

A message containing letters from A-Z is being encoded to numbers using the following mapping way:

'A' -> 1

'B' -> 2

...

'Z' -> 26

Beyond that, now the encoded string can also contain the character '\*', which can be treated as one of the numbers from 1 to 9.

Given the encoded message containing digits and the character '\*', return the total number of ways to decode it.

Also, since the answer may be very large, you should return the output mod 109 + 7.

**Example 1:**

**Input:** "\*"

**Output:** 9

**Explanation:** The encoded message can be decoded to the string: "A", "B", "C", "D", "E", "F", "G", "H", "I".

**Example 2:**

**Input:** "1\*"

**Output:** 9 + 9 = 18

**Note:**

1. The length of the input string will fit in range [1, 105].
2. The input string will only contain the character '\*' and digits '0' - '9'.

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

**int** M = 1000000007;

**public** **int** numDecodings(String s) {

**long** first = 1, second = s.charAt(0) == '\*' ? 9 : s.charAt(0) == '0' ? 0 : 1;

**for** (**int** i = 1; i < s.length(); i++) {

**long** temp = second;

**if** (s.charAt(i) == '\*') {

second = 9 \* second;

**if** (s.charAt(i - 1) == '1')

second = (second + 9 \* first) % M;

**else** **if** (s.charAt(i - 1) == '2')

second = (second + 6 \* first) % M;

**else** **if** (s.charAt(i - 1) == '\*')

second = (second + 15 \* first) % M;

} **else** {

second = s.charAt(i) != '0' ? second : 0;

**if** (s.charAt(i - 1) == '1')

second = (second + first) % M;

**else** **if** (s.charAt(i - 1) == '2' && s.charAt(i) <= '6')

second = (second + first) % M;

**else** **if** (s.charAt(i - 1) == '\*')

second = (second + (s.charAt(i) <= '6' ? 2 : 1) \* first)

% M;

}

first = temp;

}

**return** (**int**) second;

}

}

思路：从头到尾查看相邻两数，在1-26范围内的都是可能，变数在\*（1-9），但仍适用于以上规则。

## 75 [Sort Colors](https://leetcode.com/problems/sort-colors)

Given an array with *n* objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

**Note:**  
You are not suppose to use the library's sort function for this problem.

[click to show follow up.](https://leetcode.com/problems/sort-colors/)

**Follow up:**  
A rather straight forward solution is a two-pass algorithm using counting sort.  
First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.

Could you come up with an one-pass algorithm using only constant space?

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

**public** **void** sortColors(**int**[] nums) {

**int** r = 0;

**int** w = 0;

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

**if**(0==nums[i])

++r;

**if**(1==nums[i])

++w;

}

**for**(**int** i=0;i<r;++i)

nums[i] = 0;

**for**(**int** i=r;i<r+w;++i)

nums[i] = 1;

**for**(**int** i=r+w;i<nums.length;++i)

nums[i] = 2;

}

}

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

**public** **void** sortColors(**int**[] nums) {

**int** p1 = 0, p2 = nums.length - 1, index = 0;

**while** (index <= p2) {

**if** (nums[index] == 0) {

nums[index] = nums[p1];

nums[p1] = 0;

p1++;

}

**if** (nums[index] == 2) {

nums[index] = nums[p2];

nums[p2] = 2;

p2--;

index--;

}

index++;

}

}

}

思路1：数各值数量，再填充。O(n)

思路2：因为只有3种颜色，遍历夹逼法，把0和2放弃到两头。

## 91 [Decode Ways](https://leetcode.com/problems/decode-ways)

A message containing letters from A-Z is being encoded to numbers using the following mapping:

'A' -> 1

'B' -> 2

...

'Z' -> 26

Given an encoded message containing digits, determine the total number of ways to decode it.

For example,  
Given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12).

The number of ways decoding "12" is 2.

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

**public** **int** numDecodings(String s) {

**int** len = s.length();

**if** (len == 0)

**return** 0;

**int**[] dp = **new** **int**[len + 1];

dp[0] = 1;

dp[1] = s.charAt(0) == '0' ? 0 : 1;

**for** (**int** i = 2; i <= len; i++) {

**char** c0 = s.charAt(i - 1);

**char** c1 = s.charAt(i - 2);

**if** (c0 != '0')

dp[i] += dp[i - 1];

**if** (c1 == '1' || (c1 == '2' && c0 <= '6'))

dp[i] += dp[i - 2];

}

**return** dp[len];

}

}

思路：每次看相邻两数的组合情况（第一数单独以及和第二数结合，第二数单独留到下一次循环中处理）。O(n)

## 98 [Validate Binary Search Tree](https://leetcode.com/problems/validate-binary-search-tree)

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

* The left subtree of a node contains only nodes with keys **less than** the node's key.
* The right subtree of a node contains only nodes with keys **greater than** the node's key.
* Both the left and right subtrees must also be binary search trees.

**Example 1:**

2

/ \

1 3

Binary tree [2,1,3], return true.

**Example 2:**

1

/ \

2 3

Binary tree [1,2,3], return false.

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

**public** **boolean** isValidBST(TreeNode root) {

**return** isValidBST(root, Long.***MIN\_VALUE***, Long.***MAX\_VALUE***);

}

**public** **boolean** isValidBST(TreeNode root, **long** minVal, **long** maxVal) {

**if** (root == **null**)

**return** **true**;

**if** (root.val >= maxVal || root.val <= minVal)

**return** **false**;

**return** isValidBST(root.left, minVal, root.val)

&& isValidBST(root.right, root.val, maxVal);

}

}

思路：根据二叉搜索树的条件，递归查找不符合条件的，如有则不是，遍历完成没有则是。O(n)

## 133 [Clone Graph](https://leetcode.com/problems/clone-graph)

Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.  
**OJ's undirected graph serialization:**

Nodes are labeled uniquely.

We use # as a separator for each node, and , as a separator for node label and each neighbor of the node.

As an example, consider the serialized graph {0,1,2#1,2#2,2}.

The graph has a total of three nodes, and therefore contains three parts as separated by #.

1. First node is labeled as 0. Connect node 0 to both nodes 1 and 2.
2. Second node is labeled as 1. Connect node 1 to node 2.
3. Third node is labeled as 2. Connect node 2 to node 2 (itself), thus forming a self-cycle.

Visually, the graph looks like the following:

1

/ \

/ \

0 --- 2

/ \

\\_/

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

**public** UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {

**if** (node == **null**)

**return** **null**;

UndirectedGraphNode newNode = **new** UndirectedGraphNode(node.label);

HashMap<Integer, UndirectedGraphNode> map = **new** HashMap<>();

map.put(newNode.label, newNode);

LinkedList<UndirectedGraphNode> queue = **new** LinkedList<>();

queue.add(node);

**while** (!queue.isEmpty()) {

UndirectedGraphNode n = queue.pop();

**for** (UndirectedGraphNode neighbor : n.neighbors) {

**if** (!map.containsKey(neighbor.label)) {

map.put(neighbor.label,

**new** UndirectedGraphNode(neighbor.label));

queue.add(neighbor);

}

map.get(n.label).neighbors.add(map.get(neighbor.label));

}

}

**return** newNode;

}

}

思路：深度优先遍历，每次把新结点放入将访问队列，通过Map记录已访问结点，避免复制。O(nk) ，n结点数，k邻居数。

## 139 [Word Break](https://leetcode.com/problems/word-break)

Given a **non-empty** string *s* and a dictionary *wordDict* containing a list of **non-empty** words, determine if *s* can be segmented into a space-separated sequence of one or more dictionary words. You may assume the dictionary does not contain duplicate words.

For example, given  
*s* = "leetcode",  
*dict* = ["leet", "code"].

Return true because "leetcode" can be segmented as "leet code".

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

**public** **boolean** wordBreak(String s, List<String> wordDict) {

**boolean**[] f = **new** **boolean**[s.length() + 1];

f[0] = **true**;

**for** (**int** i = 1; i <= s.length(); i++) {

**for** (**int** j = 0; j < i; j++) {

**if** (f[j] && wordDict.contains(s.substring(j, i))) {

f[i] = **true**;

**break**;

}

}

}

**return** f[s.length()];

}

}

思路：动态规划，每次都利用已处理部分可分信息看到当前位置是否可分，如果可分置True。O(n^3)

## 158 [Read N Characters Given Read4 II - Call multiple times](https://leetcode.com/problems/read-n-characters-given-read4-ii-call-multiple-times)

The API: int read4(char \*buf) reads 4 characters at a time from a file.

The return value is the actual number of characters read. For example, it returns 3 if there is only 3 characters left in the file.

By using the read4 API, implement the function int read(char \*buf, int n) that reads *n* characters from the file.

**Note:**  
The read function may be called multiple times.

**public** **class** Solution **extends** Reader4 {

**private** **int** buffPtr = 0;

**private** **int** buffCnt = 0;

**private** **char**[] buff = **new** **char**[4];

**public** **int** read(**char**[] buf, **int** n) {

**int** ptr = 0;

**while** (ptr < n) {

**if** (buffPtr == 0) {

buffCnt = read4(buff);

}

**if** (buffCnt == 0)

**break**;

**while** (ptr < n && buffPtr < buffCnt) {

buf[ptr++] = buff[buffPtr++];

}

**if** (buffPtr >= buffCnt)

buffPtr = 0;

}

**return** ptr;

}

}

思路：不断读取直到达到规定数量n，二次读取从上次读取位置开始，故保存每次读取完成时的位置。O(n)

## 200 [Number of Islands](https://leetcode.com/problems/number-of-islands)

Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

***Example 1:***

11110  
11010  
11000  
00000

Answer: 1

***Example 2:***

11000  
11000  
00100  
00011

Answer: 3

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

**private** **int** n;

**private** **int** m;

**public** **int** numIslands(**char**[][] grid) {

**int** count = 0;

n = grid.length;

**if** (n == 0)

**return** 0;

m = grid[0].length;

**for** (**int** i = 0; i < n; i++) {

**for** (**int** j = 0; j < m; j++)

**if** (grid[i][j] == '1') {

DFSMarking(grid, i, j);

++count;

}

}

**return** count;

}

**private** **void** DFSMarking(**char**[][] grid, **int** i, **int** j) {

**if** (i < 0 || j < 0 || i >= n || j >= m || grid[i][j] != '1')

**return**;

grid[i][j] = '0';

DFSMarking(grid, i + 1, j);

DFSMarking(grid, i - 1, j);

DFSMarking(grid, i, j + 1);

DFSMarking(grid, i, j - 1);

}

}

思路：遍历矩阵中所有点，如果当前点不为0，则是一个岛，DFS求解当前岛，并把遇到的位置全部置0，岛数量+1。O(n)

## 277 [Find the Celebrity](https://leetcode.com/problems/find-the-celebrity)

Suppose you are at a party with n people (labeled from 0 to n - 1) and among them, there may exist one celebrity. The definition of a celebrity is that all the other n - 1 people know him/her but he/she does not know any of them.

Now you want to find out who the celebrity is or verify that there is not one. The only thing you are allowed to do is to ask questions like: "Hi, A. Do you know B?" to get information of whether A knows B. You need to find out the celebrity (or verify there is not one) by asking as few questions as possible (in the asymptotic sense).

You are given a helper function bool knows(a, b) which tells you whether A knows B. Implement a function int findCelebrity(n), your function should minimize the number of calls to knows.

**Note**: There will be exactly one celebrity if he/she is in the party. Return the celebrity's label if there is a celebrity in the party. If there is no celebrity, return -1.

**public** **class** Solution **extends** Relation {

**public** **int** findCelebrity(**int** n) {

**int** candidate = 0;

**for** (**int** i = 1; i < n; i++) {

**if** (knows(candidate, i))

candidate = i;

}

**for** (**int** i = 0; i < n; i++) {

**if** (i != candidate && (knows(candidate, i) || !knows(i, candidate)))

**return** -1;

}

**return** candidate;

}

}

思路：所有人都知道c，而c不知道任何人，所以如果0是c，则它不可能知道任何其他人，如果0不是c，则他一定知道c，以此类推，找到唯一的一个可能是c的。然后判断是否符合条件。O(n)

## 278 [First Bad Version](https://leetcode.com/problems/first-bad-version)

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which will return whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

**public** **class** Solution **extends** VersionControl {

**public** **int** firstBadVersion(**int** n) {

**int** left = 1;

**int** right = n;

**while** (left < right) {

**int** mid = left + (right - left) / 2;

**if** (isBadVersion(mid)) {

right = mid;

} **else** {

left = mid + 1;

}

}

**return** left;

}

}

思路：典型的二分查找。O(lgn)

## 282 [Expression Add Operators](https://leetcode.com/problems/expression-add-operators)

Given a string that contains only digits 0-9 and a target value, return all possibilities to add **binary** operators (not unary) +, -, or \*between the digits so they evaluate to the target value.

Examples:

"123", 6 -> ["1+2+3", "1\*2\*3"]

"232", 8 -> ["2\*3+2", "2+3\*2"]

"105", 5 -> ["1\*0+5","10-5"]

"00", 0 -> ["0+0", "0-0", "0\*0"]

"3456237490", 9191 -> []

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

**public** List<String> addOperators(String num, **int** target) {

List<String> rst = **new** ArrayList<String>();

**if** (num == **null** || num.length() == 0)

**return** rst;

helper(rst, "", num, target, 0, 0, 0);

**return** rst;

}

**public** **void** helper(List<String> rst, String path, String num, **int** target,

**int** pos, **long** eval, **long** multed) {

**if** (pos == num.length()) {

**if** (target == eval)

rst.add(path);

**return**;

}

**for** (**int** i = pos; i < num.length(); i++) {

**if** (i != pos && num.charAt(pos) == '0')

**break**;

**long** cur = Long.*parseLong*(num.substring(pos, i + 1));

**if** (pos == 0) {

helper(rst, path + cur, num, target, i + 1, cur, cur);

} **else** {

helper(rst, path + "+" + cur, num, target, i + 1, eval + cur,

cur);

helper(rst, path + "-" + cur, num, target, i + 1, eval - cur,

-cur);

helper(rst, path + "\*" + cur, num, target, i + 1,

eval - multed + multed \* cur, multed \* cur);

}

}

}

}

思路：回溯法，走每一种可能的路径，每次都需要注意0，记录下次乘的被乘数。注意溢出情况。O(n!)

## 325 [Maximum Size Subarray Sum Equals k](https://leetcode.com/problems/maximum-size-subarray-sum-equals-k)

Given an array *nums* and a target value *k*, find the maximum length of a subarray that sums to *k*. If there isn't one, return 0 instead.

**Note:**  
The sum of the entire *nums* array is guaranteed to fit within the 32-bit signed integer range.

**Example 1:**

Given *nums* = [1, -1, 5, -2, 3], *k* = 3,  
return 4. (because the subarray [1, -1, 5, -2] sums to 3 and is the longest)

**Example 2:**

Given *nums* = [-2, -1, 2, 1], *k* = 1,  
return 2. (because the subarray [-1, 2] sums to 1 and is the longest)

**Follow Up:**  
Can you do it in O(*n*) time?

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

**public** **int** maxSubArrayLen(**int**[] nums, **int** k) {

**int** sum = 0, max = 0;

HashMap<Integer, Integer> map = **new** HashMap<Integer, Integer>();

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

sum = sum + nums[i];

**if** (sum == k)

max = i + 1;

**else** **if** (map.containsKey(sum - k))

max = Math.*max*(max, i - map.get(sum - k));

**if** (!map.containsKey(sum))

map.put(sum, i);

}

**return** max;

}

}

思路：用Map保存从0到i的sum及i。如果Sum是k，则为1个候选结果；如果Sum-k已经在Map中，则从Sum-k结束到i为一个候选，取所有候选中的最大长度。O(n)

## 1001

## 29 [Divide Two Integers](https://leetcode.com/problems/divide-two-integers)

Divide two integers without using multiplication, division and mod operator.

If it is overflow, return MAX\_INT.

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

**public** **int** divide(**int** dividend, **int** divisor) {

**if** (divisor == 0)

**throw** **new** java.lang.ArithmeticException("/ by zero");

**long** result = divideLong(dividend, divisor);

**return** result > Integer.***MAX\_VALUE*** ? Integer.***MAX\_VALUE*** : (**int**) result;

}

**public** **long** divideLong(**long** dividend, **long** divisor) {

**boolean** negative = dividend < 0 != divisor < 0;

**if** (dividend < 0)

dividend = -dividend;

**if** (divisor < 0)

divisor = -divisor;

**if** (dividend < divisor)

**return** 0;

**long** sum = divisor;

**long** divide = 1;

**while** ((sum + sum) <= dividend) {

sum += sum;

divide += divide;

}

**return** negative ? -(divide + divideLong((dividend - sum), divisor))

: (divide + divideLong((dividend - sum), divisor));

}

}

思路：模拟除法，每次被除数加倍，则结果加倍（从1开始，2、4、8……）。O((lgn)^2)

## 33 [Search in Rotated Sorted Array](https://leetcode.com/problems/search-in-rotated-sorted-array)

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

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

**public** **int** search(**int**[] A, **int** target) {

**int** lo = 0;

**int** hi = A.length - 1;

**if** (hi < 0)

**return** -1;

**while** (lo < hi) {

**int** mid = (lo + hi) / 2;

**if** (A[mid] == target)

**return** mid;

**if** (A[lo] <= A[mid]) {

**if** (target >= A[lo] && target < A[mid])

hi = mid - 1;

**else**

lo = mid + 1;

} **else** {

**if** (target > A[mid] && target <= A[hi])

lo = mid + 1;

**else**

hi = mid - 1;

}

}

**return** A[lo] == target ? lo : -1;

}

}

思路：一个变形的二分查找，判断条件：先和0点比，可以判断是怎么Rotate的，然后再判断应该落在哪个区间。O(lgn)

## 43 [Multiply Strings](https://leetcode.com/problems/multiply-strings)

Given two non-negative integers num1 and num2 represented as strings, return the product of num1 and num2.

**Note:**

1. The length of both num1 and num2 is < 110.
2. Both num1 and num2 contains only digits 0-9.
3. Both num1 and num2 does not contain any leading zero.
4. You **must not use any built-in BigInteger library** or **convert the inputs to integer** directly.

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

**public** String multiply(String num1, String num2) {

**int** m = num1.length(), n = num2.length();

**int**[] pos = **new** **int**[m + n];

**for** (**int** i = m - 1; i >= 0; i--) {

**for** (**int** j = n - 1; j >= 0; j--) {

**int** mul = (num1.charAt(i) - '0') \* (num2.charAt(j) - '0');

**int** p1 = i + j, p2 = i + j + 1;

**int** sum = mul + pos[p2];

pos[p1] += sum / 10;

pos[p2] = (sum) % 10;

}

}

StringBuilder sb = **new** StringBuilder();

**for** (**int** p : pos)

**if** (!(sb.length() == 0 && p == 0))

sb.append(p);

**return** sb.length() == 0 ? "0" : sb.toString();

}

}

思路：模仿竖式法，逐位计算。结果位数为m+n，其中每个结果所在索引为(i+j,i+j+1)。复杂度O(mn)

## 47 [Permutations II](https://leetcode.com/problems/permutations-ii)

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

For example,  
[1,1,2] have the following unique permutations:

[ [1,1,2],

[1,2,1],

[2,1,1]]

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

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

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

Arrays.*sort*(nums);

backtrack(list, **new** ArrayList<>(), nums, **new** **boolean**[nums.length]);

**return** list;

}

**private** **void** backtrack(List<List<Integer>> list, List<Integer> tempList,

**int**[] nums, **boolean**[] used) {

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

list.add(**new** ArrayList<>(tempList));

} **else** {

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

**if** (used[i] || i > 0 && nums[i - 1] == nums[i] && !used[i - 1])

**continue**;

tempList.add(nums[i]);

used[i] = **true**;

backtrack(list, tempList, nums, used);

used[i] = **false**;

tempList.remove(tempList.size() - 1);

}

}

}

}

思路：回溯遍历所有情况。去重方法：使用数组记录某元素是否已经使用，已使用则重复；如果当前元素与前元素相等，如果前面元素未被使用，则重复。O(n^n)

## 56 [Merge Intervals](https://leetcode.com/problems/merge-intervals)

Given a collection of intervals, merge all overlapping intervals.

For example,  
Given [1,3],[2,6],[8,10],[15,18],  
return [1,6],[8,10],[15,18].

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

**public** List<Interval> merge(List<Interval> intervals) {

**if** (intervals.size() <= 1)

**return** intervals;

intervals.sort((i1, i2) -> Integer.*compare*(i1.start, i2.start));

List<Interval> result = **new** LinkedList<Interval>();

**int** start = intervals.get(0).start;

**int** end = intervals.get(0).end;

**for** (Interval interval : intervals) {

**if** (interval.start <= end)

end = Math.*max*(end, interval.end);

**else** {

result.add(**new** Interval(start, end));

start = interval.start;

end = interval.end;

}

}

result.add(**new** Interval(start, end));

**return** result;

}

}

思路：以左值排序点，如果后面点的左值小于当前点右值，则右值为后面点，直到不满足此条件，得到一个Interval。

## 65 [Valid Number](https://leetcode.com/problems/valid-number)

Validate if a given string is numeric.

Some examples:  
"0" => true  
" 0.1 " => true  
"abc" => false  
"1 a" => false  
"2e10" => true

**Note:** It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one.

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

**public** **boolean** isNumber(String s) {

**if** (s == **null**)

**return** **false**;

s = s.trim().toLowerCase();

**int** n = s.length();

**if** (n == 0)

**return** **false**;

**int** signCount = 0;

**boolean** hasE = **false**;

**boolean** hasNum = **false**;

**boolean** hasPoint = **false**;

**for** (**int** i = 0; i < n; i++) {

**char** c = s.charAt(i);

**if** (!isValid(c))

**return** **false**;

**if** (c >= '0' && c <= '9')

hasNum = **true**;

**if** (c == 'e') {

**if** (hasE || !hasNum)

**return** **false**;

**if** (i == n - 1)

**return** **false**;

hasE = **true**;

}

**if** (c == '.') {

**if** (hasPoint || hasE)

**return** **false**;

**if** (i == n - 1 && !hasNum)

**return** **false**;

hasPoint = **true**;

}

**if** (c == '+' || c == '-') {

**if** (signCount == 2)

**return** **false**;

**if** (i == n - 1)

**return** **false**;

**if** (i > 0 && s.charAt(i - 1) != 'e')

**return** **false**;

signCount++;

}

}

**return** **true**;

}

**boolean** isValid(**char** c) {

**return** c == '.' || c == '+' || c == '-' || c == 'e'

|| c >= '0' && c <= '9';

}

}

思路：先罗列出所有可能的数字，然后根据情况判断是正负数、指数及组合等。O(n)

## 121 [Best Time to Buy and Sell Stock](https://leetcode.com/problems/best-time-to-buy-and-sell-stock)

Say you have an array for which the *i*th element is the price of a given stock on day *i*.

If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

**Example 1:**

Input: [7, 1, 5, 3, 6, 4]

Output: 5

max. difference = 6-1 = 5 (not 7-1 = 6, as selling price needs to be larger than buying price)

**Example 2:**

Input: [7, 6, 4, 3, 1]

Output: 0

In this case, no transaction is done, i.e. max profit = 0.

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

**public** **int** maxProfit(**int**[] prices) {

**int** maxCur = 0, maxSoFar = 0;

**for** (**int** i = 1; i < prices.length; i++) {

maxCur = Math.*max*(0, maxCur += prices[i] - prices[i - 1]);

maxSoFar = Math.*max*(maxCur, maxSoFar);

}

**return** maxSoFar;

}

}

思路：股价永远是正值，从头到尾不断累加收益，如果小于零则归零（此时值小于上一个累加起点的值），每次累加的结果是一个极大值，取极大值中的最大值。O(n)

## 188 [Best Time to Buy and Sell Stock IV](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv)

Say you have an array for which the *i*th element is the price of a given stock on day *i*.

Design an algorithm to find the maximum profit. You may complete at most **k** transactions.

**Note:**  
You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

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

**public** **int** maxProfit(**int** k, **int**[] prices) {

**int** n = prices.length;

**if** (n <= 1)

**return** 0;

**if** (k >= n / 2) {

**int** maxPro = 0;

**for** (**int** i = 1; i < n; i++) {

**if** (prices[i] > prices[i - 1])

maxPro += prices[i] - prices[i - 1];

}

**return** maxPro;

}

**int**[][] dp = **new** **int**[k + 1][n];

**for** (**int** i = 1; i <= k; i++) {

**int** localMax = dp[i - 1][0] - prices[0];

**for** (**int** j = 1; j < n; j++) {

dp[i][j] = Math.*max*(dp[i][j - 1], prices[j] + localMax);//不卖掉、卖掉得钱

localMax = Math.*max*(localMax, dp[i - 1][j] - prices[j]);//不购买、购买

}

}

**return** dp[k][n - 1];

}

}

思路：如果次数不小于天数一半，则可任意多次交易，只要上涨就计入即为最大收益。如果次数不到天数一半，二维动态规划：外层买卖次数，内层位置，每次取卖与不卖的最大值（结果是一个极值，存入dp），然后在极值中取最大值。O(kn)

## 189 [Rotate Array](https://leetcode.com/problems/rotate-array)

Rotate an array of *n* elements to the right by *k* steps.

For example, with *n* = 7 and *k* = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].

**Note:**  
Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.

[[show hint]](https://leetcode.com/problems/rotate-array/)

**Hint:**  
Could you do it in-place with O(1) extra space?

Related problem: [Reverse Words in a String II](https://leetcode.com/problems/reverse-words-in-a-string-ii/)

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

**public** **void** rotate(**int**[] nums, **int** k) {

k %= nums.length;

reverse(nums, 0, nums.length - 1);

reverse(nums, 0, k - 1);

reverse(nums, k, nums.length - 1);

}

**public** **void** reverse(**int**[] nums, **int** start, **int** end) {

**while** (start < end) {

**int** temp = nums[start];

nums[start] = nums[end];

nums[end] = temp;

start++;

end--;

}

}

}

思路：先整体翻转，再对各部分翻转。O(n)

## 210 [Course Schedule II](https://leetcode.com/problems/course-schedule-ii)

There are a total of *n* courses you have to take, labeled from 0 to n - 1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite **pairs**, return the ordering of courses you should take to finish all courses.

There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

For example:

2, [[1,0]]

There are a total of 2 courses to take. To take course 1 you should have finished course 0. So the correct course order is [0,1]

4, [[1,0],[2,0],[3,1],[3,2]]

There are a total of 4 courses to take. To take course 3 you should have finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0. So one correct course order is [0,1,2,3]. Another correct ordering is[0,2,1,3].

**Note:**

1. The input prerequisites is a graph represented by **a list of edges**, not adjacency matrices. Read more about [how a graph is represented](https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs).
2. You may assume that there are no duplicate edges in the input prerequisites.

[click to show more hints.](https://leetcode.com/problems/course-schedule-ii/)

**Hints:**

1. This problem is equivalent to finding the topological order in a directed graph. If a cycle exists, no topological ordering exists and therefore it will be impossible to take all courses.
2. [Topological Sort via DFS](https://class.coursera.org/algo-003/lecture/52) - A great video tutorial (21 minutes) on Coursera explaining the basic concepts of Topological Sort.
3. Topological sort could also be done via [BFS](http://en.wikipedia.org/wiki/Topological_sorting#Algorithms).

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

**public** **int**[] findOrder(**int** numCourses, **int**[][] prerequisites) {

List<List<Integer>> adj = **new** ArrayList<List<Integer>>(numCourses);

**for** (**int** i = 0; i < numCourses; i++)

adj.add(i, **new** ArrayList<>());

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

adj.get(prerequisites[i][1]).add(prerequisites[i][0]);

**int**[] visited = **new** **int**[numCourses];

Stack<Integer> stack = **new** Stack<>();

**for** (**int** i = 0; i < numCourses; i++) {

**if** (!topologicalSort(adj, i, stack, visited))

**return** **new** **int**[0];

}

**int** i = 0;

**int**[] result = **new** **int**[numCourses];

**while** (!stack.isEmpty()) {

result[i++] = stack.pop();

}

**return** result;

}

**private** **boolean** topologicalSort(List<List<Integer>> adj, **int** v,

Stack<Integer> stack, **int**[] visited) {

**if** (visited[v] == 2)

**return** **true**;

**if** (visited[v] == 1)

**return** **false**;

visited[v] = 1;

**for** (Integer u : adj.get(v)) {

**if** (!topologicalSort(adj, u, stack, visited))

**return** **false**;

}

visited[v] = 2;

stack.push(v);

**return** **true**;

}

}

思路：建立图，从根结点开始，拓扑排序看有没有循环依赖（需要两次访问已访问结点），如果有，则无法完成；如果没有，给出一种排序即可。过程中有3个状态：0是未访问，1是已访问但是未结束（可能死循环），2是已访问且结束。

## 349 [Intersection of Two Arrays](https://leetcode.com/problems/intersection-of-two-arrays)

Given two arrays, write a function to compute their intersection.

**Example:**  
Given *nums1* = [1, 2, 2, 1], *nums2* = [2, 2], return [2].

**Note:**

* Each element in the result must be unique.
* The result can be in any order.

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

**public** **int**[] intersection(**int**[] nums1, **int**[] nums2) {

Set<Integer> set = **new** HashSet<>();

Set<Integer> intersect = **new** HashSet<>();

**for** (**int** i = 0; i < nums1.length; i++)

set.add(nums1[i]);

**for** (**int** i = 0; i < nums2.length; i++)

**if** (set.contains(nums2[i]))

intersect.add(nums2[i]);

**int**[] result = **new** **int**[intersect.size()];

**int** i = 0;

**for** (Integer num : intersect)

result[i++] = num;

**return** result;

}

}

思路：先把一个数组中出现的数字存入Set，然后遍历另一数组，如果在Set中出现，则添加到结果中。

## 560 [Subarray Sum Equals K](https://leetcode.com/problems/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**.

**Example 1:**

**Input:**nums = [1,1,1], k = 2

**Output:** 2

**Note:**

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

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

**public** **int** subarraySum(**int**[] nums, **int** k) {

**int** count = 0, sum = 0;

Map<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;

}

}

思路：任何连续相加为k的子数组必定起始点为1或结束点为当前位置。利用Map存Accumulate Sum及其出现次数。那么每当Map中有sum-k（相当从i于向前查找），则为一种解，求和即可。O(n)

## 1002

## 1003

## 1004

## 1005

## 1006

## 3 [Longest Substring Without Repeating Characters](https://leetcode.com/problems/longest-substring-without-repeating-characters)

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

**Examples:**

Given "abcabcbb", the answer is "abc", which the length is 3.

Given "bbbbb", the answer is "b", with the length of 1.

Given "pwwkew", 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.

**class** Solution {

**public** **int** lengthOfLongestSubstring(String s) {

**char**[] sc = s.toCharArray();

Map<Character, Integer> cm = **new** HashMap<Character, Integer>();

**int** j = 0, maxLen = 0;

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

**char** cur = sc[i];

**if**(cm.containsKey(cur)) {

maxLen = Math.*max*(i - j, maxLen);

j = Math.*max*(j, cm.get(cur) + 1);

}

cm.put(cur, i);

}

**return** Math.*max*(sc.length - j, maxLen);

}

}

思路：其实只需要前面出现过的重复字符的下标即可算出此段不重复子段的长度，核心操作其实是向前检索重复字符。需要注意的是最后循环完成后，需要再算一下没有计算的那段的长度，在这些子段中取最长的。O(n)

## 10 [Regular Expression Matching](https://leetcode.com/problems/regular-expression-matching)

Implement regular expression matching with support for '.' and '\*'.

'.' Matches any single character.

'\*' Matches zero or more of the preceding element.//meaning, string won’t start with \*

The matching should cover the **entire** input string (not partial).

The function prototype should be:

bool isMatch(const char \*s, const char \*p)

Some examples:

isMatch("aa","a") ? false

isMatch("aa","aa") ? true

isMatch("aaa","aa") ? false

isMatch("aa", "a\*") ? true

isMatch("aa", ".\*") ? true

isMatch("ab", ".\*") ? true

isMatch("aab", "c\*a\*b") ? true

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

**public** **boolean** isMatch(String str, String regex) {

**boolean**[][] dp = **new** **boolean**[str.length() + 1][regex.length() + 1];

dp[0][0] = **true**;

**for**(**int** i = 1; i < regex.length() + 1; i++) {

**if**(regex.charAt(i - 1) == '\*') dp[0][i] = dp[0][i-2];

}

**for**(**int** i = 1; i < dp.length; i++) {

**for**(**int** j = 1; j < dp[0].length; j++) {

**if**(match(str.charAt(i-1), regex.charAt(j-1))) {

dp[i][j] = dp[i-1][j-1];

}**else**{

**if**(regex.charAt(j-1) == '\*') {

dp[i][j] = dp[i][j-2];

**if**(match(str.charAt(i-1), regex.charAt(j-2))) {

dp[i][j] |= dp[i-1][j];

}

}

}

}

}

**return** dp[str.length()][regex.length()];

}

**private** **boolean** match(**char** c1, **char** r) {

**return** c1 == r || r == '.';

}

}

思路：二维动态规划：如果P(regex,i)和S(str,j)当前字符相等（或P是.），则结果与上一位置(i-1,j-1)相同。如果当前位置不等，则为False。如果当前位置P值为\*，如果P上一位置与S当前相等或为.，则结果与上一P位置相同(i-1,j)。

## 28 [Implement strStr()](https://leetcode.com/problems/implement-strstr)

Implement strStr().

Returns the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

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

**public** **int** strStr(String haystack, String needle) {

**if** (haystack == **null** || needle == **null**)

**return** -1;

**int** m = haystack.length();

**int** n = needle.length();

Set<String> ns = **new** HashSet<String>(){{add(needle);}};

**for**(**int** i = 0; i < m-n+1; ++i) {

String cur = haystack.substring(i, i + n);

**if**(ns.contains(cur))

**return** i;

}

**return** -1;

}

}

思路：取得haystack中每一个与needle等长的子串，取Hash与needle相比，然后再计算是否相等。 O(m+n)

不使用hash，可以使用数字签名，简单实现：

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

**public** **int** strStr(String haystack, String needle) {

**if** (haystack == **null** || needle == **null**)

**return** -1;

**int** n = haystack.length();

**int** m = needle.length();

**if** (n < m)

**return** -1;

List<Character> sq1 = **new** LinkedList<>();

List<Character> sq2 = **new** LinkedList<>();

**int** signature = 0, sigCur = 0;

**int** i = 0;

**while** (i < m) {

**char** c1 = haystack.charAt(i);

**char** c2 = needle.charAt(i++);

signature += c2 - 'A' + 1;

sigCur += c1 - 'A' + 1;

sq1.add(c1);

sq2.add(c2);

}

**if** (sigCur == signature) {

**if** (isEqual(sq1, sq2))

**return** i - m;

}

**while** (i < n) {

**char** c1 = haystack.charAt(i);

sigCur += c1 - 'A' + 1;

sq1.add(c1);

sq1.remove(0);

**char** c2 = haystack.charAt(i - m);

sigCur -= c2 - 'A' + 1;

++i;

**if** (sigCur == signature) {

**if** (isEqual(sq1, sq2))

**return** i - m;

}

}

**return** -1;

}

**private** **boolean** isEqual(List<Character> sq1, List<Character> sq2) {

**for** (**int** i = 0; i < sq1.size(); ++i) {

**if** (sq1.get(i) != sq2.get(i))

**return** **false**;

}

**return** **true**;

}

}

## 38 [Count and Say](https://leetcode.com/problems/count-and-say)

The count-and-say sequence is the sequence of integers with the first five terms as following:

1. 1

2. 11

3. 21

4. 1211

5. 111221

1 is read off as "one 1" or 11.  
11 is read off as "two 1s" or 21.  
21 is read off as "one 2, then one 1" or 1211.

Given an integer *n*, generate the *n*th term of the count-and-say sequence.

Note: Each term of the sequence of integers will be represented as a string.

**Example 1:**

**Input:** 1

**Output:** "1"

**Example 2:**

**Input:** 4

**Output:** "1211"

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

**public** String countAndSay(**int** n) {

**if** (n == 1) {

**return** "1";

}

**return** countAndSay(countAndSay(n - 1));

}

**public** String countAndSay(String str) {

StringBuilder sb = **new** StringBuilder();

**char**[] cs = str.toCharArray();

**int** i = 0, j = 0;

**for** (; i < cs.length; i = j) {

**char** cur = cs[i];

j = i + 1;

**while** (j < cs.length && cur == cs[j]) {

++j;

}

sb.append(j - i);

sb.append(cur);

}

**return** sb.toString();

}

}

思路：递归求解，每次把问题降低一个数，直到数值为“1”。每次Count返回对应的字符串。O(n(m^2))，m为数字长度。

## 39 [Combination Sum](https://leetcode.com/problems/combination-sum)

Given a **set** of candidate numbers (***C***) **(without duplicates)** and a target number (***T***), find all unique combinations in ***C*** where the candidate numbers sums to ***T***.

The **same** repeated number may be chosen from ***C*** unlimited number of times.

**Note:**

* All numbers (including target) will be positive integers.
* The solution set must not contain duplicate combinations.

For example, given candidate set [2, 3, 6, 7] and target 7,   
A solution set is:

[ [7],

[2, 2, 3]]

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

**public** List<List<Integer>> combinationSum(**int**[] candidates, **int** target) {

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

**int** n = candidates.length;

**for** (**int** i = 0; i < n; ++i) {

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

candidateL.add(candidates[i]);

*backTrack*(candidates, i, candidates[i], candidateL, target, r);

}

**return** r;

}

**private** **static** **void** backTrack(**int**[] nums, **int** i, **int** sum,

List<Integer> candidateL, **int** target, List<List<Integer>> r) {

**if** (sum > target)

**return**;

**if** (sum == target) {

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

newL.addAll(candidateL);

r.add(newL);

**return**;

}

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

candidateL.add(Integer.*valueOf*(nums[j]));

sum += nums[j];

*backTrack*(nums, j, sum, candidateL, target, r);

candidateL.remove(Integer.*valueOf*(nums[j]));

sum -= nums[j];

}

}

}

思路：回溯法尝试每种组合，结束条件：如果和已经大于等于目标值。O(n^k)

## 46 [Permutations](https://leetcode.com/problems/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],

[3,2,1]]

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

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

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

*backtrack*(list, **new** ArrayList<>(), nums);

**return** list;

}

**private** **static** **void** backtrack(List<List<Integer>> list,

List<Integer> tempList, **int**[] nums) {

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

list.add(**new** ArrayList<>(tempList));

} **else** {

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

**if** (tempList.contains(nums[i]))

**continue**; // element already exists, skip

tempList.add(nums[i]);

*backtrack*(list, tempList, nums);

tempList.remove(tempList.size() - 1);

}

}

}

}

思路：回溯法尝试所有可能，通过一个List（Set可能更好）排除已经使用的数值。O(n^n)

## 51 [N-Queens](https://leetcode.com/problems/n-queens)

The *n*-queens puzzle is the problem of placing *n* queens on an *n*×*n* chessboard such that no two queens attack each other.



Given an integer *n*, return all distinct solutions to the *n*-queens puzzle.

Each solution contains a distinct board configuration of the *n*-queens' placement, where 'Q' and '.' both indicate a queen and an empty space respectively.

For example,  
There exist two distinct solutions to the 4-queens puzzle:

[[".Q..", // Solution 1

"...Q",

"Q...",

"..Q."],

["..Q.", // Solution 2

"Q...",

"...Q",

".Q.."]]

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

**public** List<List<String>> solveNQueens(**int** n) {

List<List<String>> r = **new** ArrayList<List<String>>();

**boolean**[] col = **new** **boolean**[n];

**boolean**[] lr = **new** **boolean**[n \* 2];

**boolean**[] rl = **new** **boolean**[n \* 2];

*backtrack*(0, n, col, lr, rl, r, **new** ArrayList<String>());

**return** r;

}

**private** **static** **void** backtrack(**int** row, **int** n, **boolean**[] col, **boolean**[] lr,

**boolean**[] rl, List<List<String>> r, List<String> l) {

**if** (row == n) {

List<String> resultL = **new** ArrayList<String>();

resultL.addAll(l);

r.add(resultL);

}

**if** (row == n)

**return**;

**for** (**int** i = 0; i < n; ++i) {

**int** lrp = row - i + n;

**int** rlp = n \* 2 - i - row - 1;

**if** (col[i] || lr[lrp] || rl[rlp]) {

**continue**;

}

StringBuilder sb = **new** StringBuilder();

**for** (**int** i1 = 0; i1 < n; ++i1) {

**if** (i == i1) {

sb.append("Q");

**continue**;

}

sb.append(".");

}

l.add(sb.toString());

col[i] = **true**;

lr[lrp] = **true**;

rl[rlp] = **true**;

*backtrack*(row + 1, n, col, lr, rl, r, l);

l.remove(l.size() - 1);

col[i] = **false**;

lr[lrp] = **false**;

rl[rlp] = **false**;

}

}

}

思路：回溯法逐行（逐位）尝试，每次置“Q”后，将其所对应的列（行已经处理了）、两个纵行设置为已访问。O((n^2)^2)

## 53 [Maximum Subarray](https://leetcode.com/problems/maximum-subarray)

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array [-2,1,-3,4,-1,2,1,-5,4],  
the contiguous subarray [4,-1,2,1] has the largest sum = 6.

[click to show more practice.](https://leetcode.com/problems/maximum-subarray/)

**More practice:**

If you have figured out the O(*n*) solution, try coding another solution using the divide and conquer approach, which is more subtle.

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

**public** **int** maxSubArray(**int**[] nums) {

**int** max = Integer.***MIN\_VALUE***;

**int** sum = 0;

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

**int** cur = nums[i];

sum += cur;

max = Math.*max*(max, sum);

**if**(sum < 0) {

sum = 0;

**continue**;

}

}

**return** max;

}

}

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

**private** **class** ArrayContext {

**int** max;

**int** lMax;

**int** rMax;

**int** sum;

}

**public** ArrayContext getArrayContext(**int**[] nums, **int** l, **int** r) {

ArrayContext ctx = **new** ArrayContext();

**if** (l == r) {

ctx.max = nums[l];

ctx.lMax = nums[l];

ctx.rMax = nums[l];

ctx.sum = nums[l];

} **else** {

**int** m = (l + r) / 2;

ArrayContext lCtx = getArrayContext(nums, l, m);

ArrayContext rCtx = getArrayContext(nums, m + 1, r);

ctx.max = Math.*max*(Math.*max*(lCtx.max, rCtx.max),

lCtx.rMax + rCtx.lMax);

ctx.lMax = Math.*max*(lCtx.lMax, lCtx.sum + rCtx.lMax);

ctx.rMax = Math.*max*(rCtx.rMax, rCtx.sum + lCtx.rMax);

ctx.sum = lCtx.sum + rCtx.sum;

}

**return** ctx;

}

**public** **int** maxSubArray(**int**[] nums) {

**if** (nums.length == 0) {

**return** 0;

}

ArrayContext ctx = getArrayContext(nums, 0, nums.length - 1);

**return** ctx.max;

}

}

思路1：类似股票题目，逐一求和，如果小于0则再从0开始。

思路2：二分法。

## 73 [Set Matrix Zeroes](https://leetcode.com/problems/set-matrix-zeroes)

Given a *m* x *n* matrix, if an element is 0, set its entire row and column to 0. Do it in place.

[click to show follow up.](https://leetcode.com/problems/set-matrix-zeroes/)

**Follow up:**

Did you use extra space?  
A straight forward solution using O(*mn*) space is probably a bad idea.  
A simple improvement uses O(*m* + *n*) space, but still not the best solution.  
Could you devise a constant space solution?

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

**public** **void** setZeroes(**int**[][] matrix) {

**boolean** fr = **false**,fc = **false**;

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

**for**(**int** j = 0; j < matrix[0].length; j++) {

**if**(matrix[i][j] == 0) {

**if**(i == 0) fr = **true**;

**if**(j == 0) fc = **true**;

matrix[0][j] = 0;

matrix[i][0] = 0;

}

}

}

**for**(**int** i = 1; i < matrix.length; i++) {

**for**(**int** j = 1; j < matrix[0].length; j++) {

**if**(matrix[i][0] == 0 || matrix[0][j] == 0) {

matrix[i][j] = 0;

}

}

}

**if**(fr) {

**for**(**int** j = 0; j < matrix[0].length; j++) {

matrix[0][j] = 0;

}

}

**if**(fc) {

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

matrix[i][0] = 0;

}

}

}

}

思路：利用首行、列存储归零信息。对于首行、列是否归零则存在两个Boolean变量中。然后依次处理剩余点和首行、列。O(mn)

## 78 [Subsets](https://leetcode.com/problems/subsets)

Given a set of **distinct** integers, *nums*, return all possible subsets.

**Note:** The solution set must not contain duplicate subsets.

For example,  
If ***nums*** = [1,2,3], a solution is:

[ [3],

[1],

[2],

[1,2,3],

[1,3],

[2,3],

[1,2],

[]]

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

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

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

backtrack(list, **new** ArrayList<>(), nums, 0);

**return** list;

}

**private** **void** backtrack(List<List<Integer>> list, List<Integer> tempList,

**int**[] nums, **int** start) {

list.add(**new** ArrayList<>(tempList));

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

tempList.add(nums[i]);

backtrack(list, tempList, nums, i + 1);

tempList.remove(tempList.size() - 1);

}

}

}

思路：回溯查找所有可能情况。O(n^n)

## 79 [Word Search](https://leetcode.com/problems/word-search)

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

For example,  
Given **board** =

[ ['A','B','C','E'],

['S','F','C','S'],

['A','D','E','E']]

**word** = "ABCCED", -> returns true,  
**word** = "SEE", -> returns true,  
**word** = "ABCB", -> returns false.

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

**public** **boolean** exist(**char**[][] board, String word) {

**if** (word == **null** || word.length() == 0) {

**return** **true**;

}

**char**[] chs = word.toCharArray();

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

**for** (**int** j = 0; j < board[0].length; j++) {

**if** (dfs(board, chs, 0, i, j)) {

**return** **true**;

}

}

}

**return** **false**;

}

**private** **boolean** dfs(**char**[][] board, **char**[] words, **int** idx, **int** x, **int** y) {

**if** (idx == words.length) {

**return** **true**;

}

**if** (x < 0 || x == board.length || y < 0 || y == board[0].length) {

**return** **false**;

}

**if** (board[x][y] != words[idx]) {

**return** **false**;

}

board[x][y] ^= 256;

**boolean** exist = dfs(board, words, idx + 1, x, y + 1)

|| dfs(board, words, idx + 1, x, y - 1)

|| dfs(board, words, idx + 1, x + 1, y)

|| dfs(board, words, idx + 1, x - 1, y);

board[x][y] ^= 256;

**return** exist;

}

}

思路，DFS四向查找结果，使用过的点记录为^256，完成该轮再^256恢复。O(m^2\*n^2)

## 88 [Merge Sorted Array](https://leetcode.com/problems/merge-sorted-array)

Given two sorted integer arrays *nums1* and *nums2*, merge *nums2* into *nums1* as one sorted array.

**Note:**  
You may assume that *nums1* has enough space (size that is greater or equal to *m* + *n*) to hold additional elements from *nums2*. The number of elements initialized in *nums1* and *nums2* are *m* and *n* respectively.

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

**public** **void** merge(**int**[] nums1, **int** m, **int**[] nums2, **int** n) {

**for** (**int** i = nums1.length - 1; i >= n; --i) {

nums1[i] = nums1[i - n];

}

**int** k = 0;

**for** (**int** i = n, j = 0; i < m + n || j < n;) {

**if** (i >= m + n) {

nums1[k++] = nums2[j++];

} **else** **if** (j >= n) {

nums1[k++] = nums1[i++];

} **else** {

nums1[k++] = nums1[i] > nums2[j] ? nums2[j++] : nums1[i++];

}

}

}

}

思路：逐一将较小的数存入num1即可。O(m+n)

## 93 [Restore IP Addresses](https://leetcode.com/problems/restore-ip-addresses)

Given a string containing only digits, restore it by returning all possible valid IP address combinations.

For example:  
Given "25525511135",

return ["255.255.11.135", "255.255.111.35"]. (Order does not matter)

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

**public** List<String> restoreIpAddresses(String s) {

List<String> ret = **new** ArrayList<>();

StringBuffer ip = **new** StringBuffer();

**for** (**int** a = 1; a < 4; ++a)

**for** (**int** b = 1; b < 4; ++b)

**for** (**int** c = 1; c < 4; ++c)

**for** (**int** d = 1; d < 4; ++d) {

**if** (a + b + c + d == s.length()) {

**int** n1 = Integer.*parseInt*(s.substring(0, a));

**int** n2 = Integer.*parseInt*(s.substring(a, a + b));

**int** n3 = Integer

.*parseInt*(s.substring(a + b, a + b + c));

**int** n4 = Integer.*parseInt*(s.substring(a + b + c));

**if** (n1 <= 255 && n2 <= 255 && n3 <= 255

&& n4 <= 255) {

ip.append(n1).append('.').append(n2).append('.')

.append(n3).append('.').append(n4);

**if** (ip.length() == s.length() + 3)

ret.add(ip.toString());

ip.delete(0, ip.length());

}

}

}

**return** ret;

}

}

思路：IP每位长度0-4，所以逐一尝试看是否能凑足4个值在0-255间的数，如果满足则记录。O(1)

## 124 [Binary Tree Maximum Path Sum](https://leetcode.com/problems/binary-tree-maximum-path-sum)

Given a binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path must contain **at least one node** and does not need to go through the root.

For example:  
Given the below binary tree,

1

/ \

2 3

Return 6.

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

**int** maxValue;

**public** **int** maxPathSum(TreeNode root) {

maxValue = Integer.***MIN\_VALUE***;

maxPathDown(root);

**return** maxValue;

}

**private** **int** maxPathDown(TreeNode node) {

**if** (node == **null**)

**return** 0;

**int** left = Math.*max*(0, maxPathDown(node.left));

**int** right = Math.*max*(0, maxPathDown(node.right));

maxValue = Math.*max*(maxValue, left + right + node.val);

**return** Math.*max*(left, right) + node.val;

}

}

思路：遍历时记录每一个点的最值。最值有三种，从左、右子树到当前和从左子树经过当前再到右子树。O(n)

## 138 [Copy List with Random Pointer](https://leetcode.com/problems/copy-list-with-random-pointer)

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a deep copy of the list.

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

**public** RandomListNode copyRandomList(RandomListNode head) {

**if** (head == **null**)

**return** **null**;

Map<RandomListNode, RandomListNode> map = **new** HashMap<RandomListNode, RandomListNode>();

RandomListNode node = head;

**while** (node != **null**) {

map.put(node, **new** RandomListNode(node.label));

node = node.next;

}

node = head;

**while** (node != **null**) {

map.get(node).next = map.get(node.next);

map.get(node).random = map.get(node.random);

node = node.next;

}

**return** map.get(head);

}

}

思路：遍历两次原链表，第一次利用Map记录新Node，第二次利用Map为新Node添加next和random指向。O(n)

## 143 [Reorder List](https://leetcode.com/problems/reorder-list)

Given a singly linked list *L*: *L*0?*L*1?…?*Ln*-1?*L*n,  
reorder it to: *L*0?*Ln*?*L*1?*Ln*-1?*L*2?*Ln*-2?…

You must do this in-place without altering the nodes' values.

For example,  
Given {1,2,3,4}, reorder it to {1,4,2,3}.

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

**public** **void** reorderList(ListNode head) {

**if** (head == **null** || head.next == **null**)

**return**;

ListNode p1 = head;

ListNode p2 = head;

**while** (p2.next != **null** && p2.next.next != **null**) {

p1 = p1.next;

p2 = p2.next.next;

}

ListNode preMiddle = p1;

ListNode preCurrent = p1.next;

**while** (preCurrent.next != **null**) {

ListNode current = preCurrent.next;

preCurrent.next = current.next;

current.next = preMiddle.next;

preMiddle.next = current;

}

p1 = head;

p2 = preMiddle.next;

**while** (p1 != preMiddle) {

preMiddle.next = p2.next;

p2.next = p1.next;

p1.next = p2;

p1 = p2.next;

p2 = preMiddle.next;

}

}

}

思路：三步走，先找到中点，然后把后半个链表倒置，然后再得到结果。O(n)

## 146 [LRU Cache](https://leetcode.com/problems/lru-cache)

Design and implement a data structure for [Least Recently Used (LRU) cache](https://en.wikipedia.org/wiki/Cache_replacement_policies#LRU). It should support the following operations: get and put.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.  
put(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

**Follow up:**  
Could you do both operations in **O(1)** time complexity?

**Example:**

LRUCache cache = new LRUCache( 2 /\* capacity \*/ );

cache.put(1, 1);

cache.put(2, 2);

cache.get(1); // returns 1

cache.put(3, 3); // evicts key 2

cache.get(2); // returns -1 (not found)

cache.put(4, 4); // evicts key 1

cache.get(1); // returns -1 (not found)

cache.get(3); // returns 3

cache.get(4); // returns 4

**public** **class** LRUCache {

**private** LinkedHashMap<Integer, Integer> map;

**private** **final** **int** CAPACITY;

**public** LRUCache(**int** capacity) {

CAPACITY = capacity;

map = **new** LinkedHashMap<Integer, Integer>(capacity, 0.75f, **true**) {

@Override

**protected** **boolean** removeEldestEntry(Map.Entry eldest) {

**return** size() > CAPACITY;

}

};

}

**public** **int** get(**int** key) {

**return** map.getOrDefault(key, -1);

}

**public** **void** put(**int** key, **int** value) {

map.put(key, value);

}

}

思路：利用LinkedHashMap的移除最老元素。

## 155 [Min Stack](https://leetcode.com/problems/min-stack)

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

* push(x) -- Push element x onto stack.
* pop() -- Removes the element on top of the stack.
* top() -- Get the top element.
* getMin() -- Retrieve the minimum element in the stack.

**Example:**

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); --> Returns -3.

minStack.pop();

minStack.top(); --> Returns 0.

minStack.getMin(); --> Returns -2.

**public** **class** MinStack {

Stack<Integer> stack = **new** Stack<Integer>();

Stack<Integer> minStack = **new** Stack<Integer>();

**public** **void** push(**int** x) {

stack.push(x);

**if** (minStack.isEmpty())

minStack.push(x);

**else**

**if** (minStack.peek() > x)

minStack.push(x);

**else**

minStack.push(minStack.peek());

}

**public** **void** pop() {

**if** (!stack.isEmpty()) {

stack.pop();

minStack.pop();

}

}

**public** **int** top() {

**return** stack.peek();

}

**public** **int** getMin() {

**return** minStack.peek();

}

}

思路：使用另一个Stack存当前最小值。

## 173 [Binary Search Tree Iterator](https://leetcode.com/problems/binary-search-tree-iterator)

Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.

Calling next() will return the next smallest number in the BST.

**Note:**next() and hasNext() should run in average O(1) time and uses O(*h*) memory, where *h* is the height of the tree.

**public** **class** BSTIterator {

**private** Stack<TreeNode> stack = **new** Stack<TreeNode>();

**public** BSTIterator(TreeNode root) {

pushAll(root);

}

**public** **boolean** hasNext() {

**return** !stack.isEmpty();

}

**public** **int** next() {

TreeNode tmpNode = stack.pop();

pushAll(tmpNode.right);

**return** tmpNode.val;

}

**private** **void** pushAll(TreeNode node) {

**for** (; node != **null**; stack.push(node), node = node.left)

;

}

}

思路：二叉搜索树特点：左孩子总是较小，右孩子总是较大，所以每次要到最远的左结点，缓存到Stack中，如果弹出当前元素，则将对应右结点及相应的左枝缓存入Stack中。

## 206 [Reverse Linked List](https://leetcode.com/problems/reverse-linked-list)

Reverse a singly linked list.

[click to show more hints.](https://leetcode.com/problems/reverse-linked-list/)

**Hint:**

A linked list can be reversed either iteratively or recursively. Could you implement both?

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

**public** ListNode reverseList(ListNode head) {

ListNode prev = **null**;

ListNode curr = head;

**while** (curr != **null**) {

ListNode nextTemp = curr.next;

curr.next = prev;

prev = curr;

curr = nextTemp;

}

**return** prev;

}

}

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

**public** ListNode reverseList(ListNode head) {

**if** (head == **null** || head.next == **null**)

**return** head;

ListNode p = reverseList(head.next);

head.next.next = head;//反向指针

head.next = **null**;//删除原指针

**return** p;

}

}

思路1：逐一反向指针。O(n)

思路2：也是逐一反向指针，只不过是从链表尾部开始。O(n)

## 211 [Add and Search Word - Data structure design](https://leetcode.com/problems/add-and-search-word-data-structure-design)

Design a data structure that supports the following two operations:

void addWord(word)

bool search(word)

search(word) can search a literal word or a regular expression string containing only letters a-z or .. A . means it can represent any one letter.

For example:

addWord("bad")

addWord("dad")

addWord("mad")

search("pad") -> false

search("bad") -> true

search(".ad") -> true

search("b..") -> true

**Note:**  
You may assume that all words are consist of lowercase letters a-z.

[click to show hint.](https://leetcode.com/problems/add-and-search-word-data-structure-design/)

You should be familiar with how a Trie works. If not, please work on this problem: [Implement Trie (Prefix Tree)](https://leetcode.com/problems/implement-trie-prefix-tree/) first.

**public** **class** WordDictionary {

**public** **class** TrieNode {

**public** TrieNode[] children = **new** TrieNode[26];

**public** String item = "";

}

**private** TrieNode root = **new** TrieNode();

**public** **void** addWord(String word) {

TrieNode node = root;

**for** (**char** c : word.toCharArray()) {

**if** (node.children[c - 'a'] == **null**) {

node.children[c - 'a'] = **new** TrieNode();

}

node = node.children[c - 'a'];

}

node.item = word;

}

**public** **boolean** search(String word) {

**return** match(word.toCharArray(), 0, root);

}

**private** **boolean** match(**char**[] chs, **int** k, TrieNode node) {

**if** (k == chs.length)

**return** !node.item.equals("");

**if** (chs[k] != '.') {

**return** node.children[chs[k] - 'a'] != **null**

&& match(chs, k + 1, node.children[chs[k] - 'a']);

} **else** {

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

**if** (node.children[i] != **null**) {

**if** (match(chs, k + 1, node.children[i])) {

**return** **true**;

}

}

}

}

**return** **false**;

}

}

思路：使用Trie树存字典便于查找。

## 215 [Kth Largest Element in an Array](https://leetcode.com/problems/kth-largest-element-in-an-array)

Find the **k**th largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

For example,  
Given [3,2,1,5,6,4] and k = 2, return 5.

**Note:**  
You may assume k is always valid, 1 ? k ? array's length.

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

**public** **int** findKthLargest(**int**[] nums, **int** k) {

**final** **int** N = nums.length;

Arrays.*sort*(nums);

**return** nums[N - k];

}

}

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

**public** **int** findKthLargest(**int**[] nums, **int** k) {

**final** PriorityQueue<Integer> pq = **new** PriorityQueue<>();

**for** (**int** val : nums) {

pq.offer(val);

**if** (pq.size() > k) {

pq.poll();

}

}

**return** pq.peek();

}

}

思路1：先排序再查找。O(nlogn)

思路2：使用优先队列缓存k个数，最后最小的即是第K大。O(nlogk)

## 237 [Delete Node in a Linked List](https://leetcode.com/problems/delete-node-in-a-linked-list)

Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.

Supposed the linked list is 1 -> 2 -> 3 -> 4 and you are given the third node with value 3, the linked list should become 1 -> 2 -> 4 after calling your function.

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

**public** **void** deleteNode(ListNode node) {

node.val = node.next.val;

node.next = node.next.next;

}

}

思路：拷贝后面元素值到当前元素，删除后面元素。

## 257 [Binary Tree Paths](https://leetcode.com/problems/binary-tree-paths)

Given a binary tree, return all root-to-leaf paths.

For example, given the following binary tree:

1

/ \

2 3

\

5

All root-to-leaf paths are:

["1->2->5", "1->3"]

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

**public** List<String> binaryTreePaths(TreeNode root) {

List<String> answer = **new** ArrayList<String>();

**if** (root != **null**)

searchBT(root, "", answer);

**return** answer;

}

**private** **void** searchBT(TreeNode root, String path, List<String> answer) {

**if** (root.left == **null** && root.right == **null**)

answer.add(path + root.val);

**if** (root.left != **null**)

searchBT(root.left, path + root.val + "->", answer);

**if** (root.right != **null**)

searchBT(root.right, path + root.val + "->", answer);

}

}

思路：前序遍历。

## 259 [3Sum Smaller](https://leetcode.com/problems/3sum-smaller)

Given an array of *n* integers *nums* and a *target*, find the number of index triplets i, j, k with 0 <= i < j < k < n that satisfy the condition nums[i] + nums[j] + nums[k] < target.

For example, given *nums* = [-2, 0, 1, 3], and *target* = 2.

Return 2. Because there are two triplets which sums are less than 2:

[-2, 0, 1]

[-2, 0, 3]

**Follow up:**  
Could you solve it in *O*(*n*2) runtime?

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

**public** **int** threeSumSmaller(**int**[] nums, **int** target) {

Arrays.*sort*(nums);

**int** sum = 0;

**for** (**int** i = 0; i < nums.length - 2; i++) {

sum += twoSumSmaller(nums, i + 1, target - nums[i]);

}

**return** sum;

}

**private** **int** twoSumSmaller(**int**[] nums, **int** startIndex, **int** target) {

**int** sum = 0;

**int** left = startIndex;

**int** right = nums.length - 1;

**while** (left < right) {

**if** (nums[left] + nums[right] < target) {

sum += right - left;

left++;

} **else**

right--;

}

**return** sum;

}

}

思路：先排序，然后逐一数字看有没有另外两个数字可以满足条件。O(n^2)

## 268 [Missing Number](https://leetcode.com/problems/missing-number)

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

For example,  
Given *nums* = [0, 1, 3] return 2.

**Note**:  
Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?

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

**public** **int** missingNumber(**int**[] nums) {

**int** xor = 0, i = 0;

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

xor = xor ^ i ^ nums[i];

**return** xor ^ i;

}

}

思路：利用^^是本身，如果一个数字没在其中，那么0^i^0即为i。O(n)

## 285 [Inorder Successor in BST](https://leetcode.com/problems/inorder-successor-in-bst)

Given a binary search tree and a node in it, find the in-order successor of that node in the BST.

**Note**: If the given node has no in-order successor in the tree, return null.

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

**public** TreeNode inorderSuccessor(TreeNode root, TreeNode p) {

**if** (root == **null**)

**return** **null**;

**if** (root.val <= p.val) {

**return** inorderSuccessor(root.right, p);

} **else** {

TreeNode left = inorderSuccessor(root.left, p);

**return** (left != **null**) ? left : root;

}

}

}

思路：中序遍历结果是第一个比该值大的值是其后续。O(n)

## 308 [Range Sum Query 2D - Mutable](https://leetcode.com/problems/range-sum-query-2d-mutable)

Given a 2D matrix *matrix*, find the sum of the elements inside the rectangle defined by its upper left corner (*row*1, *col*1) and lower right corner (*row*2, *col*2).

  
The above rectangle (with the red border) is defined by (row1, col1) = **(2, 1)** and (row2, col2) = **(4, 3)**, which contains sum = **8**.

**Example:**

Given matrix = [

[3, 0, 1, 4, 2],

[5, 6, 3, 2, 1],

[1, 2, 0, 1, 5],

[4, 1, 0, 1, 7],

[1, 0, 3, 0, 5]]

sumRegion(2, 1, 4, 3) -> 8

update(3, 2, 2)

sumRegion(2, 1, 4, 3) -> 10

**Note:**

1. The matrix is only modifiable by the *update* function.
2. You may assume the number of calls to *update* and *sumRegion* function is distributed evenly.
3. You may assume that *row*1 ≤ *row*2 and *col*1 ≤ *col*2.

**public** **class** NumMatrix {

**int**[][] tree;

**int**[][] nums;

**int** m;

**int** n;

**public** NumMatrix(**int**[][] matrix) {

**if** (matrix.length == 0 || matrix[0].length == 0)

**return**;

m = matrix.length;

n = matrix[0].length;

tree = **new** **int**[m + 1][n + 1];

nums = **new** **int**[m][n];

**for** (**int** i = 0; i < m; i++)

**for** (**int** j = 0; j < n; j++)

update(i, j, matrix[i][j]);

}

**public** **void** update(**int** row, **int** col, **int** val) {

**if** (m == 0 || n == 0)

**return**;

**int** delta = val - nums[row][col];

nums[row][col] = val;

**for** (**int** i = row + 1; i <= m; i += i & (-i))

**for** (**int** j = col + 1; j <= n; j += j & (-j))

tree[i][j] += delta;

}

**public** **int** sumRegion(**int** row1, **int** col1, **int** row2, **int** col2) {

**if** (m == 0 || n == 0)

**return** 0;

**return** sum(row2 + 1, col2 + 1) + sum(row1, col1) - sum(row1, col2 + 1)

- sum(row2 + 1, col1);

}

**public** **int** sum(**int** row, **int** col) {

**int** sum = 0;

**for** (**int** i = row; i > 0; i -= i & (-i)) {

**for** (**int** j = col; j > 0; j -= j & (-j)) {

sum += tree[i][j];

}

}

**return** sum;

}

}

思路：利用累加信息，中间的方块是由两个子方块（一个靠上边，一个靠左边）重叠部分组成，所以其和为大方块（右下角到（0，0）-其左面方块-其上面方块再加上这两个方块的重叠部分（因为其被减了2次）。

## 341 [Flatten Nested List Iterator](https://leetcode.com/problems/flatten-nested-list-iterator)

Given a nested list of integers, implement an iterator to flatten it.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

**Example 1:**  
Given the list [[1,1],2,[1,1]],

By calling *next* repeatedly until *hasNext* returns false, the order of elements returned by *next* should be: [1,1,2,1,1].

**Example 2:**  
Given the list [1,[4,[6]]],

By calling *next* repeatedly until *hasNext* returns false, the order of elements returned by *next* should be: [1,4,6].

**public** **class** NestedIterator **implements** Iterator<Integer> {

Queue<Integer> l;

**public** NestedIterator(List<NestedInteger> nestedList) {

l = **new** LinkedList<Integer>();

flattenNestedList(nestedList);

}

**private** **void** flattenNestedList(List<NestedInteger> list) {

**for**(NestedInteger ni:list) {

**if**(ni.isInteger())

l.add(ni.getInteger());

**else**

flattenNestedList(ni.getList());

}

}

@Override

**public** Integer next() {

**return** l.poll();

}

@Override

**public** **boolean** hasNext() {

**return** !l.isEmpty();

}

}

思路1：初始化时将结果存入一维数组。

**public** **class** NestedIterator **implements** Iterator<Integer> {

List<NestedInteger> l = **new** LinkedList<>();

**public** NestedIterator(List<NestedInteger> nestedList) {

l.addAll(nestedList);

}

@Override

**public** Integer next() {

**while**(!l.isEmpty()) {

NestedInteger cur = l.remove(0);

**if**(cur.isInteger())

**return** cur.getInteger();

**for**(**int** i = cur.getList().size()-1;i>=0;--i) {

NestedInteger v = cur.getList().get(i);

l.add(0, v);

}

}

**throw** **new** NullPointerException();

}

@Override

**public** **boolean** hasNext() {

**while**(!l.isEmpty()) {

NestedInteger cur = l.get(0);

**if**(cur.isInteger())

**return** **true**;

l.remove(0);

**for**(**int** i = cur.getList().size()-1;i>=0;--i) {

NestedInteger v = cur.getList().get(i);

l.add(0, v);

}

}

**return** **false**;

}

}

思路2：next前把当前元素（如果是数组）重新按元素顺序放到缓存数组最前。

## 348 [Design Tic-Tac-Toe](https://leetcode.com/problems/design-tic-tac-toe)

Design a Tic-tac-toe game that is played between two players on a *n* x *n* grid.

You may assume the following rules:

1. A move is guaranteed to be valid and is placed on an empty block.
2. Once a winning condition is reached, no more moves is allowed.
3. A player who succeeds in placing *n* of their marks in a horizontal, vertical, or diagonal row wins the game.

**Example:**

Given *n* = 3, assume that player 1 is "X" and player 2 is "O" in the board.

TicTacToe toe = new TicTacToe(3);

toe.move(0, 0, 1); -> Returns 0 (no one wins)

|X| | |

| | | | // Player 1 makes a move at (0, 0).

| | | |

toe.move(0, 2, 2); -> Returns 0 (no one wins)

|X| |O|

| | | | // Player 2 makes a move at (0, 2).

| | | |

toe.move(2, 2, 1); -> Returns 0 (no one wins)

|X| |O|

| | | | // Player 1 makes a move at (2, 2).

| | |X|

toe.move(1, 1, 2); -> Returns 0 (no one wins)

|X| |O|

| |O| | // Player 2 makes a move at (1, 1).

| | |X|

toe.move(2, 0, 1); -> Returns 0 (no one wins)

|X| |O|

| |O| | // Player 1 makes a move at (2, 0).

|X| |X|

toe.move(1, 0, 2); -> Returns 0 (no one wins)

|X| |O|

|O|O| | // Player 2 makes a move at (1, 0).

|X| |X|

toe.move(2, 1, 1); -> Returns 1 (player 1 wins)

|X| |O|

|O|O| | // Player 1 makes a move at (2, 1).

|X|X|X|

**public** **class** TicTacToe {

**private** **int**[] rows;

**private** **int**[] cols;

**private** **int** diagonal;

**private** **int** antiDiagonal;

**public** TicTacToe(**int** n) {

rows = **new** **int**[n];

cols = **new** **int**[n];

}

**public** **int** move(**int** row, **int** col, **int** player) {

**int** toAdd = player == 1 ? 1 : -1;

rows[row] += toAdd;

cols[col] += toAdd;

**if** (row == col)

diagonal += toAdd;

**if** (col == (cols.length - row - 1))

antiDiagonal += toAdd;

**int** size = rows.length;

**if** (Math.*abs*(rows[row]) == size || Math.*abs*(cols[col]) == size

|| Math.*abs*(diagonal) == size

|| Math.*abs*(antiDiagonal) == size) {

**return** player;

}

**return** 0;

}

}

思路：记录四个方向（左右、上下、斜穿）的个数，如果达到则赢。

## 350 [Intersection of Two Arrays II](https://leetcode.com/problems/intersection-of-two-arrays-ii)

Given two arrays, write a function to compute their intersection.

**Example:**  
Given *nums1* = [1, 2, 2, 1], *nums2* = [2, 2], return [2, 2].

**Note:**

* Each element in the result should appear as many times as it shows in both arrays.
* The result can be in any order.

**Follow up:**

* What if the given array is already sorted? How would you optimize your algorithm?
* What if *nums1*'s size is small compared to *nums2*'s size? Which algorithm is better?
* What if elements of *nums2* are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

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

**public** **int**[] intersect(**int**[] nums1, **int**[] nums2) {

Map<Integer, Integer> set = **new** HashMap<>();

Map<Integer, Integer> intersect = **new** HashMap<>();

**int** count = 0;

**for** (**int** i = 0; i < nums1.length; i++)

set.put(nums1[i], set.getOrDefault(nums1[i], 0) + 1);

**for** (**int** i = 0; i < nums2.length; i++) {

**if** (set.containsKey(nums2[i]) && set.get(nums2[i]) > intersect

.getOrDefault(nums2[i], 0)) {

intersect.put(nums2[i],

intersect.getOrDefault(nums2[i], 0) + 1);

++count;

}

}

**int**[] result = **new** **int**[count];

**int** j = 0;

**for** (Integer num : intersect.keySet())

**for** (**int** i = 0; i < intersect.get(num); ++i)

result[j++] = num;

**return** result;

}

}

思路：用Map记录某一个数组里每个数出现的次数，遍历另一个数组时，如果出现重复则存入另一个Map(计数不超过第一个Map)。O(m+n)

## 381 [Insert Delete GetRandom O(1) - Duplicates allowed](https://leetcode.com/problems/insert-delete-getrandom-o1-duplicates-allowed)

Design a data structure that supports all following operations in *average* **O(1)** time.

**Note: Duplicate elements are allowed.**

1. insert(val): Inserts an item val to the collection.
2. remove(val): Removes an item val from the collection if present.
3. getRandom: Returns a random element from current collection of elements. The probability of each element being returned is **linearly related** to the number of same value the collection contains.

**Example:**

// Init an empty collection.

RandomizedCollection collection = new RandomizedCollection();

// Inserts 1 to the collection. Returns true as the collection did not contain 1.

collection.insert(1);

// Inserts another 1 to the collection. Returns false as the collection contained 1. Collection now contains [1,1].

collection.insert(1);

// Inserts 2 to the collection, returns true. Collection now contains [1,1,2].

collection.insert(2);

// getRandom should return 1 with the probability 2/3, and returns 2 with the probability 1/3.

collection.getRandom();

// Removes 1 from the collection, returns true. Collection now contains [1,2].

collection.remove(1);

// getRandom should return 1 and 2 both equally likely.

collection.getRandom();

**public** **class** RandomizedCollection {

ArrayList<Integer> nums;

HashMap<Integer, Set<Integer>> locs;

java.util.Random rand = **new** java.util.Random();

**public** RandomizedCollection() {

nums = **new** ArrayList<Integer>();

locs = **new** HashMap<Integer, Set<Integer>>();

}

**public** **boolean** insert(**int** val) {

**boolean** contain = locs.containsKey(val);

**if** (!contain)

locs.put(val, **new** LinkedHashSet<Integer>());

locs.get(val).add(nums.size());

nums.add(val);

**return** !contain;

}

**public** **boolean** remove(**int** val) {

**boolean** contain = locs.containsKey(val);

**if** (!contain)

**return** **false**;

**int** loc = locs.get(val).iterator().next();

locs.get(val).remove(loc);

**if** (loc < nums.size() - 1) {

**int** lastone = nums.get(nums.size() - 1);

nums.set(loc, lastone);

locs.get(lastone).remove(nums.size() - 1);

locs.get(lastone).add(loc);

}

nums.remove(nums.size() - 1);

**if** (locs.get(val).isEmpty())

locs.remove(val);

**return** **true**;

}

**public** **int** getRandom() {

**return** nums.get(rand.nextInt(nums.size()));

}

}

思路：使用数组保存数值，使用Map<int,list>保存数的位置，删除时，把要删除元素与最后一个元素互换，删除最后一个元素。

## 393 [UTF-8 Validation](https://leetcode.com/problems/utf-8-validation)

A character in UTF8 can be from **1 to 4 bytes** long, subjected to the following rules:

1. For 1-byte character, the first bit is a 0, followed by its unicode code.
2. For n-bytes character, the first n-bits are all one's, the n+1 bit is 0, followed by n-1 bytes with most significant 2 bits being 10.

This is how the UTF-8 encoding would work:

Char. number range | UTF-8 octet sequence

(hexadecimal) | (binary)

--------------------+---------------------------------------------

0000 0000-0000 007F | 0xxxxxxx

0000 0080-0000 07FF | 110xxxxx 10xxxxxx

0000 0800-0000 FFFF | 1110xxxx 10xxxxxx 10xxxxxx

0001 0000-0010 FFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Given an array of integers representing the data, return whether it is a valid utf-8 encoding.

**Note:**  
The input is an array of integers. Only the **least significant 8 bits** of each integer is used to store the data. This means each integer represents only 1 byte of data.

**Example 1:**

data = [197, 130, 1], which represents the octet sequence: **11000101 10000010 00000001**.

Return **true**.

It is a valid utf-8 encoding for a 2-bytes character followed by a 1-byte character.

**Example 2:**

data = [235, 140, 4], which represented the octet sequence: **11101011 10001100 00000100**.

Return **false**.

The first 3 bits are all one's and the 4th bit is 0 means it is a 3-bytes character.

The next byte is a continuation byte which starts with 10 and that's correct.

But the second continuation byte does not start with 10, so it is invalid.

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

**public** **boolean** validUtf8(**int**[] data) {

**if**(data==**null** || data.length==0) **return** **false**;

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

**if**(data[i]>255) **return** **false**; // 1 after 8th digit, 100000000

**int** numberOfBytes = 0;

**if**((data[i] & 128) == 0) { // 0xxxxxxx, 1 byte, 128(10000000)

numberOfBytes = 1;

} **else** **if**((data[i] & 224) == 192) { // 110xxxxx, 2 bytes, 224(11100000), 192(11000000)

numberOfBytes = 2;

} **else** **if**((data[i] & 240) == 224) { // 1110xxxx, 3 bytes, 240(11110000), 224(11100000)

numberOfBytes = 3;

} **else** **if**((data[i] & 248) == 240) { // 11110xxx, 4 bytes, 248(11111000), 240(11110000)

numberOfBytes = 4;

} **else** {

**return** **false**;

}

**for**(**int** j=1;j<numberOfBytes;j++) { // check that the next n bytes start with 10xxxxxx

**if**(i+j>=data.length) **return** **false**;

**if**((data[i+j] & 192) != 128) **return** **false**; // 192(11000000), 128(10000000)

}

i=i+numberOfBytes-1;

}

**return** **true**;

}

}

## 415 [Add Strings](https://leetcode.com/problems/add-strings)

Given two non-negative integers num1 and num2 represented as string, return the sum of num1 and num2.

**Note:**

1. The length of both num1 and num2 is < 5100.
2. Both num1 and num2 contains only digits 0-9.
3. Both num1 and num2 does not contain any leading zero.
4. You **must not use any built-in BigInteger library** or **convert the inputs to integer** directly.

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

**public** String addStrings(String num1, String num2) {

StringBuilder sb = **new** StringBuilder();

**int** carry = 0;

**for** (**int** i = num1.length() - 1, j = num2.length() - 1; i >= 0 || j >= 0

|| carry == 1; i--, j--) {

**int** x = i < 0 ? 0 : num1.charAt(i) - '0';

**int** y = j < 0 ? 0 : num2.charAt(j) - '0';

sb.append((x + y + carry) % 10);

carry = (x + y + carry) / 10;

}

**return** sb.reverse().toString();

}

}

思路：竖式加法，从最后位开始累加存入StringBuilder，最后反转String即为结果。O(n)

## 477 [Total Hamming Distance](https://leetcode.com/problems/total-hamming-distance)

The [Hamming distance](https://en.wikipedia.org/wiki/Hamming_distance) between two integers is the number of positions at which the corresponding bits are different.

Now your job is to find the total Hamming distance between all pairs of the given numbers.

**Example:**

**Input:** 4, 14, 2

**Output:** 6

**Explanation:** In binary representation, the 4 is 0100, 14 is 1110, and 2 is 0010 (just

showing the four bits relevant in this case). So the answer will be:

HammingDistance(4, 14) + HammingDistance(4, 2) + HammingDistance(14, 2) = 2 + 2 + 2 = 6.

**Note:**

1. Elements of the given array are in the range of 0 to 10^9
2. Length of the array will not exceed 10^4.

**class** Solution {

**public** **int** totalHammingDistance(**int**[] nums) {

**int** total = 0, n = nums.length;

**for** (**int** j = 0; j < 32; j++) {

**int** bitCount = 0;

**for** (**int** i = 0; i < n; i++)

bitCount += (nums[i] >> j) & 1;

total += bitCount \* (n - bitCount);

}

**return** total;

}

}

思路：每一位上0和1的个数相乘即为该位上的Hamming Distance。O(n)

## 525 [Contiguous Array](https://leetcode.com/problems/contiguous-array)

Given a binary array, find the maximum length of a contiguous subarray with equal number of 0 and 1.

**Example 1:**

**Input:** [0,1]

**Output:** 2

**Explanation:** [0, 1] is the longest contiguous subarray with equal number of 0 and 1.

**Example 2:**

**Input:** [0,1,0]

**Output:** 2

**Explanation:** [0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of 0 and 1.

**Note:** The length of the given binary array will not exceed 50,000.

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

**public** **int** findMaxLength(**int**[] nums) {

Map<Integer, Integer> map = **new** HashMap<>();

map.put(0, -1);

**int** maxlen = 0, count = 0;

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

count = count + (nums[i] == 1 ? 1 : -1);

**if** (map.containsKey(count)) {

maxlen = Math.*max*(maxlen, i - map.get(count));

} **else** {

map.put(count, i);

}

}

**return** maxlen;

}

}

思路：计算累计和（1则+1，0则-1），如果有两个累和相等，则中间的0和1个数相等。O(n)

## 543 [Diameter of Binary Tree](https://leetcode.com/problems/diameter-of-binary-tree)

Given a binary tree, you need to compute the length of the diameter of the tree. The diameter of a binary tree is the length of the **longest**path between any two nodes in a tree. This path may or may not pass through the root.

**Example:**  
Given a binary tree

1

/ \

2 3

/ \

4 5

Return **3**, which is the length of the path [4,2,1,3] or [5,2,1,3].

**Note:** The length of path between two nodes is represented by the number of edges between them.

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

**public** **int** diameterOfBinaryTree(TreeNode root) {

**if** (root == **null**) {

**return** 0;

}

**int** curMax = getMaxDepth(root.left, 0) + getMaxDepth(root.right, 0);

**int** leftRightMax = Math.*max*(diameterOfBinaryTree(root.left),

diameterOfBinaryTree(root.right));

**return** Math.*max*(curMax, leftRightMax);

}

**private** **int** getMaxDepth(TreeNode root, **int** dia) {

**if** (root == **null**) {

**return** dia;

} **else** {

**return** Math.*max*(getMaxDepth(root.left, dia + 1),

getMaxDepth(root.right, dia + 1));

}

}

}

思路：遍历计算所有点的Diameter。O(nm)

## 572 [Subtree of Another Tree](https://leetcode.com/problems/subtree-of-another-tree)

Given two non-empty binary trees **s** and **t**, check whether tree **t** has exactly the same structure and node values with a subtree of **s**. A subtree of **s** is a tree consists of a node in **s** and all of this node's descendants. The tree **s** could also be considered as a subtree of itself.

**Example 1:**  
Given tree s:

3

/ \

4 5

/ \

1 2

Given tree t:

4

/ \

1 2

Return **true**, because t has the same structure and node values with a subtree of s.

**Example 2:**  
Given tree s:

3

/ \

4 5

/ \

1 2

/

0

Given tree t:

4

/ \

1 2

Return **false**.

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

**public** **boolean** isSubtree(TreeNode s, TreeNode t) {

**if** (isSubtreeComp(s, t)) {

**return** **true**;

}

**if** (s == **null**) {

**return** **false**;

}

**return** isSubtree(s.left, t) || isSubtree(s.right, t);

}

**public** **boolean** isSubtreeComp(TreeNode s, TreeNode t) {

**if** (s == **null** && t == **null**) {

**return** **true**;

}

**if** (s == **null** || t == **null**) {

**return** **false**;

}

**return** (s.val == t.val && isSubtreeComp(s.left, t.left)

&& isSubtreeComp(s.right, t.right));

}

}

思路：递归查看是否为某个结点起的子树。O(n^2)

## 640 [Solve the Equation](https://leetcode.com/problems/solve-the-equation)

Solve a given equation and return the value of x in the form of string "x=#value". The equation contains only '+', '-' operation, the variable x and its coefficient.

If there is no solution for the equation, return "No solution".

If there are infinite solutions for the equation, return "Infinite solutions".

If there is exactly one solution for the equation, we ensure that the value of x is an integer.

**Example 1:**

**Input:** "x+5-3+x=6+x-2"

**Output:** "x=2"

**Example 2:**

**Input:** "x=x"

**Output:** "Infinite solutions"

**Example 3:**

**Input:** "2x=x"

**Output:** "x=0"

**Example 4:**

**Input:** "2x+3x-6x=x+2"

**Output:** "x=-1"

**Example 5:**

**Input:** "x=x+2"

**Output:** "No solution"

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

**public** String coeff(String x) {

**if** (x.length() > 1 && x.charAt(x.length() - 2) >= '0'

&& x.charAt(x.length() - 2) <= '9')

**return** x.replace("x", "");

**return** x.replace("x", "1");

}

**public** String solveEquation(String equation) {

String[] lr = equation.split("=");

**int** lhs = 0, rhs = 0;

**for** (String x : breakIt(lr[0])) {

**if** (x.indexOf("x") >= 0) {

lhs += Integer.*parseInt*(coeff(x));

} **else**

rhs -= Integer.*parseInt*(x);

}

**for** (String x : breakIt(lr[1])) {

**if** (x.indexOf("x") >= 0)

lhs -= Integer.*parseInt*(coeff(x));

**else**

rhs += Integer.*parseInt*(x);

}

**if** (lhs == 0) {

**if** (rhs == 0)

**return** "Infinite solutions";

**else**

**return** "No solution";

}

**return** "x=" + rhs / lhs;

}

**public** List<String> breakIt(String s) {

List<String> res = **new** ArrayList<>();

String r = "";

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

**if** (s.charAt(i) == '+' || s.charAt(i) == '-') {

**if** (r.length() > 0)

res.add(r);

r = "" + s.charAt(i);

} **else**

r += s.charAt(i);

}

res.add(r);

**return** res;

}

}

思路：计算左右两边的和及x个数，如果x全部被消且两边相等，则有无穷多解；如果x全部被消且两边不等，则无解。

## 647 Palindromic Substrings

Given a string, your task is to count how many palindromic substrings in this string.

The substrings with different start indexes or end indexes are counted as different substrings even they consist of same characters.

**Example 1:**

**Input:** "abc"

**Output:** 3

**Explanation:** Three palindromic strings: "a", "b", "c".

**Example 2:**

**Input:** "aaa"

**Output:** 6

**Explanation:** Six palindromic strings: "a", "a", "a", "aa", "aa", "aaa".

**Note:**

1. The input string length won't exceed 1000.

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

**int** count = 0;

**public** **int** countSubstrings(String s) {

**if** (s == **null** || s.length() == 0)

**return** 0;

**for** (**int** i = 0; i < s.length(); i++) { // i is the mid point

extendPalindrome(s, i, i); // odd length;

extendPalindrome(s, i, i + 1); // even length

}

**return** count;

}

**private** **void** extendPalindrome(String s, **int** left, **int** right) {

**while** (left >= 0 && right < s.length()

&& s.charAt(left) == s.charAt(right)) {

count++;

left--;

right++;

}

}

}

思路：从每一个字符展开查找可能的互文串，每增加一点长度，则多一种可能。O(n^2)

## 654 Maximum Binary Tree

Given an integer array with no duplicates. A maximum tree building on this array is defined as follow:

1. The root is the maximum number in the array.
2. The left subtree is the maximum tree constructed from left part subarray divided by the maximum number.
3. The right subtree is the maximum tree constructed from right part subarray divided by the maximum number.

Construct the maximum tree by the given array and output the root node of this tree.

**Example 1:**

**Input:** [3,2,1,6,0,5]

**Output:** return the tree root node representing the following tree:

6

/ \

3 5

\ /

2 0

\

1

**Note:**

1. The size of the given array will be in the range [1,1000].

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

**public** TreeNode constructMaximumBinaryTree(**int**[] nums) {

**return** construct(nums, 0, nums.length);

}

**public** TreeNode construct(**int**[] nums, **int** l, **int** r) {

**if** (l == r)

**return** **null**;

**int** max\_i = max(nums, l, r);

TreeNode root = **new** TreeNode(nums[max\_i]);

root.left = construct(nums, l, max\_i);

root.right = construct(nums, max\_i + 1, r);

**return** root;

}

**public** **int** max(**int**[] nums, **int** l, **int** r) {

**int** max\_i = l;

**for** (**int** i = l; i < r; i++)

**if** (nums[max\_i] < nums[i])

max\_i = i;

**return** max\_i;

}

}

思路：每次从数组中找最大值作为根，把数组分成两部分，左、面边作为左、右子树。O(nlogn)

## 1007

## 1008

## 1009

## 1110

## 1111

## 1112

## 1113

## 1114

## 1115

## 1116

## 1117

## 1118

## 1119

## 1120

## 1121

## 670 [Maximum Swap](https://leetcode.com/problems/maximum-swap)

Given a non-negative integer, you could swap two digits **at most** once to get the maximum valued number. Return the maximum valued number you could get.

**Example 1:**

**Input:** 2736

**Output:** 7236

**Explanation:** Swap the number 2 and the number 7.

**Example 2:**

**Input:** 9973

**Output:** 9973

**Explanation:** No swap.

**Note:**

1. The given number is in the range [0, 108]

**class** Solution {

**public** **int** maximumSwap(**int** num) {

**char**[] A = Integer.*toString*(num).toCharArray();

**int**[] last = **new** **int**[10];

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

last[A[i] - '0'] = i;

}

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

**for** (**int** d = 9; d > A[i] - '0'; d--) {

**if** (last[d] > i) {

**char** tmp = A[i];

A[i] = A[last[d]];

A[last[d]] = tmp;

**return** Integer.*valueOf*(**new** String(A));

}

}

}

**return** num;

}

}

思路：先找出每个数最后一个Index。遍历数字每一位，找最大值的最后一位，如非最前位置，则和最前位置置换即得结果。O(n)

## 1122

## 1123

# 近期没有出现，但是在LC中有标签的：

## 689

## 157

## 23

## 597

## 161

## 252

## 602

## 57

## 636

## 236

## 494

## 13

## 680

## 238

## 642

## 523

## 398

## 218

## 49

## 209

## 286

## 102

## 208

## 554

## 90

## 71

## 128

## 20

## 377

## 127

## 44

## 334

## 269

## 380

## 461

## 235

## 50

## 85

## 117

## 69

## 68

## 26

## 221

## 274

## 404

## 25

## 234

## 168

## 714

## 80

## 261

## 410

## 535

## 265

## 721

## 578

## 637

## 653

## 673

## 674

## 275

# 2016年旧题

## 309

## 16

## 116

## 108

## 109

## 4

## 122

## 123

## 309

## 81

## 199

## 239

## 92

## 5

## 62

## 63

## 114

## 159

## 37

## 108

## 22

## 191

## 214

## 198

## 152

## 32

## 300

## 224

## 310

## 111

## 151

## 395