**Missing / Duplicate Numbers**

268. 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.

题解：

0~n个数字，每个只出现1次，这样可以直接算sum，然后相减。

public int missingNumber(int[] nums) {

if (nums == null) return 0;

int sum = 0, n = nums.length;

for (int num:nums) sum += num;

return n\*(1 + n)/2 - sum;

}

163. Missing Ranges

Given a sorted integer array where the range of elements are in the inclusive range [lower, upper], return its missing ranges.

Eg, given [0, 1, 3, 50, 75], lower = 0 and upper = 99, return ["2", "4->49", "51->74", "76->99"].

287. Find the Duplicate Number

Given an array nums containing n + 1 integers where each integer is between 1 and n (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.

**S1. Binary Search**

取一个中间值，每次判断duplicate one 是在前半段还是后半段。

binary search本身是O(logn), 加上中间计数的部分是O(n), 所以是O(nlogn).

public int findDuplicate2(int[] nums) {

if (nums == null || nums.length < 2) return -1;

int min = 1, max = nums.length - 1;

while(min < max){

int mid = min + (max - min) / 2;

int count = 0;

*// 计算总数组中有多少个数小于等于中间数*

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

if(nums[i] <= mid) count++;

}

*// 如果小于等于中间数的数量小于中间数，说明后半部分必有重复*

if(count <= mid) min = mid + 1;

*// 否则前半部分必有重复*

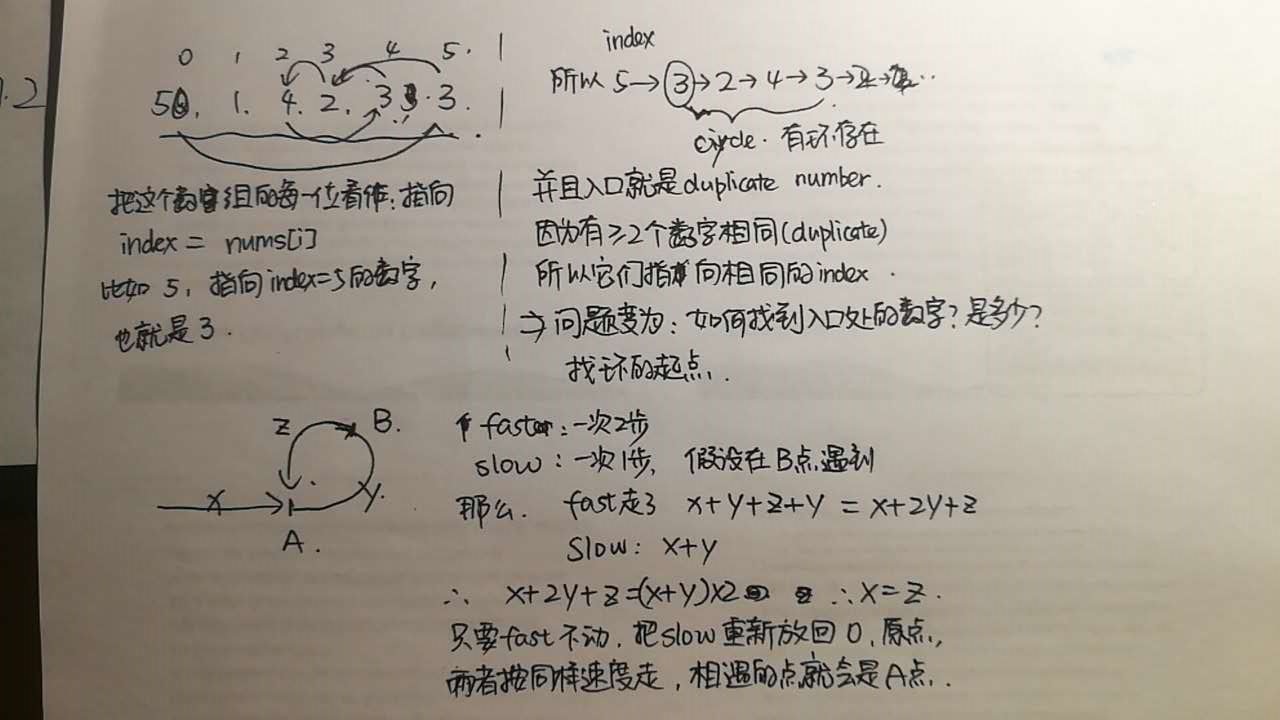
else max = mid;

}

return min;

}

**S2. Fast, slow pointers time O(n)**



public int findDuplicatee(int[] nums) {

if (nums == null || nums.length < 2) return -1;

int slow = nums[0];

int fast = nums[nums[0]];

while (slow != fast) {

slow = nums[slow];

fast = nums[nums[fast]];

}

slow = 0;

while (slow != fast) {

slow = nums[slow];

fast = nums[fast];

}

return slow;

}

**LIS 单调栈**

300. Longest Increasing Subsequence

Given an unsorted array of integers, find the length of longest increasing subsequence.

For example, Given [10, 9, 2, 5, 3, 7, 101, 18], The longest increasing subsequence is [2, 3, 7, 101], therefore the length is 4. Note that there may be more than one LIS combination, it is only necessary for you to return the length.

Your algorithm should run in O(n2) complexity.

Follow up: Could you improve it to O(n log n) time complexity?

Follow up: print out the LIS.

**分三部分：输出LIS的长度，输出任意一个LIS，输出所有的LIS。**

注意：LIS类问题，只需要是increasing subsequence, 不要求各个数字相连/consecutive

**S1.DP**

普适的做法，maxSoFar, maxEndHere分别代表全局最优和局部最优。time O(n^2)

dp[i]: max # of LIS end at i, 并且nums[i]一定包括在内，

public int lengthOfLIS(int[] nums) {

if (nums == null || nums.length == 0) return 0;

int n = nums.length;

int[] dp = new int[n]; *// maxEndHere*

int max = 1;

Arrays.fill(dp, 1);

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

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

if (nums[i] > nums[j]) {

dp[i] = Math.max(dp[i], dp[j] + 1);

max = Math.max(max, dp[i]);

}

}

}

return max;

}

**S2.Binary search**

time: O(n logn)

-记录目前最长的list的每个值，判断新的nums[i]是否能延长它，

-如果不能，也就是nums[i] < largest of LIS, 就update指定位置的值，list的size也就是LIS.

public static int lengthOfLIS2(int[] nums) {

if (nums == null || nums.length == 0) return 0;

List<Integer> res = new ArrayList<>();

for (int num:nums) {

// list为空，或者>当前LIS最大值

if (res.size() == 0 || res.get(res.size() - 1) < num) {

res.add(num);

} else {

int index = Collections.binarySearch(res, num);

if (index < 0) index = -(index + 1);

res.set(index, num);

}

}

return res.size();

}

**Follow up: 输出任意一个LIS:**

**S1.DP的基础上**

建立一个数组parent, parent[i]放比nums[i]小的，sequence中前一个数字的位置，

就可以从后往前遍历，依次找到。

public static int[] longestLIS(int[] nums) {

if (nums == null || nums.length == 0) return null;

int n = nums.length;

int[] dp = new int[n]; *// maxEndHere*

int[] parent = new int[n]; *// store parent id, the previous one;*

int max = 1;

int maxIdx = 0;

Arrays.fill(dp, 1);

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

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

int len = dp[j] + 1;

if (nums[i] > nums[j] && len > dp[i]) {

dp[i] = len;

parent[i] = j;

max = dp[i];

maxIdx = i;

}

}

}

*// now maxIdx point to the last index of LIS*

int[] res = new int[max];

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

res[i] = nums[maxIdx];

maxIdx = parent[maxIdx];

}

return res;

}

**S2.Binary search**

用res矩阵记录LIS，并且返回的int[]是所有的LIS中各个位最小的。

public static int[] longestLIS2(int[] nums) {

if (nums == null || nums.length == 0) return null;

List<Integer> res = new ArrayList<>();

List<Integer> cur = new ArrayList<>();

for (int num:nums) {

if (res.size() == 0 || res.get(res.size() - 1) < num) {

cur.add(num);

res = cur;

} else {

int index = Collections.binarySearch(cur, num);

if (index < 0) index = -(index + 1);

cur.set(index, num);

if (index == cur.size() - 1) res = cur;

}

}

int[] out = new int[res.size()];

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

out[i] = res.get(i);

}

return out;

}

**FollowUp 2: 返回所有LIS**

走到这里已经和backtracking的三种经典做法很像了，事实上也是这样：

分为两个步骤：

1. 找到所有满足LIS的值，比如输入是1, 7, 5, 101, LIS可以是1, 5, 101或者1,7,101

那么在把7 set成5之前，先存入原先的数字到map，<index, number that works>

1， 2（3， 4）, 5（6， 20）, 95（101），每个位置上都有多种可能性，那么可能性的对应就存在map中。

1. 像上面的例子一样，每一位都有多种可能的取值，用DFS找出所有的组合。

public static List<List<Integer>> longestAllLIS2(int[] nums) {

if (nums == null || nums.length == 0) return null;

List<List<Integer>> out = new ArrayList<>();

*// <index, list of int that works*

Map<Integer, Set<Integer>> map = new HashMap<>();

List<Integer> cur = new ArrayList<>();

List<Integer> res = new ArrayList<>();

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

int num = nums[i];

if (res.size() == 0 || res.get(res.size() - 1) < num) {

cur.add(num);

res = cur;

out.clear();

} else {

int index = Collections.binarySearch(cur, num);

if (index < 0) index = -(index + 1);

*// save previous number at index k*

if (!map.containsKey(index)) map.put(index, new HashSet<>());

map.get(index).add(cur.get(index));

*// update index*

cur.set(index, num);

if (index == cur.size() - 1) res = cur;

}

}

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

if (!map.containsKey(i)) map.put(i, new HashSet<>());

map.get(i).add(res.get(i));

}

addLists(out, res, new ArrayList<>(), map, 0);

return out;

}

// DFS，找到所有值，并加入out.

// out: output, res: one of LIS,

// cur: LIS that is working on, the index before start has finished.

// start: the index to work on this iteration, unprocessed.

private static void addLists(List<List<Integer>> out, List<Integer> res, List<Integer> cur, Map<Integer, Set<Integer>> map, int start) {

if (start == res.size()) {

out.add(new ArrayList<>(cur));

return;

}

int size = res.size();

for (int i = start;i < size;i++) {

for (int value:map.get(i)) {

if (cur.isEmpty() || value > cur.get(cur.size() - 1)) {

cur.add(value);

addLists(out, res, cur, map, start + 1);

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

}

}

}

}

单调栈 – 单调栈例题

有N个人，顺次排列，他们的身高分别为H[i]，每个人向自己后方看，他能看到的人

是在他后面/数组右侧离他最近的且比他高的人。请依次输出每个人能看到的人的编号

Next[i]，如果他后面不存在比他高的人，则输出-1。

想找 "从当前元素向某一方向的第一个 (大于 / 小于) 自己的元素"，就要靠单调栈来维护单调性，对应的是 (递减 / 递增)。

**S1.Stack**

stack用来存放index，如果某个index还没有找到右边比自己高的，那么就会保留在stack里。

每走到一个位置i, 只可能更新res[0, i-1]的值，判断当前的heights[i] 是否大于stack中存放的index对应的height。如果大于，满足条件，则会把stack中的index弹出来，并且赋值为i。到最后仍在stack里的，说明右侧没有更高的人。

for 循环是O(n)，并且每个数字只可能被push进stack一次，pop一次，所以是O(n)

public int[] getHeight(int[] heights) {

int n = heights.length;

int[] res = new int[n];

// numbers stored in stack: idx of nums haven’t found a higher person behind them.

// 并且stack中存进的数字只可能越来越小，或者不变，不可能越来越大

Stack<Integer> s = new Stack<>();

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

while (!s.isEmpty() && heights[i] > heights[s.peek()]) {

res[s.poll()] = heights[i];

}

s.push(i);

}

while (!s.isEmpty()) res[s.poll()] = -1;

return res;

}

**S2 从右向左走**

用一个list保留已经走过的所有heights, 然后每次往左走到一个新的num，就在list中binary search, 找比它大一点的那个数字，有就赋值，没有就为-1.

time O(nlogn)

85. Maximal Rectangle

Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area. For example, given the following matrix: Return 6.

1 0 1 0 0

1 0 **1 1 1**

1 1 **1 1 1**

1 0 0 1 0

**S1.DP time O(mn)**

考虑以第i行为底边的长方形，每一行：

先从左到右遍历，取得每个位置j对应的height，并且用left来跟踪连续的1的左侧边界，存的是index，

然后从右向左遍历，并且用right来跟踪每个位置j对应的连续1的右侧边界，存index

并在更新右边界的同时，通过height, left来update max，当前点的面积是(right[j] – left[j]+1)\*height[j]

初始化：左边界需要通过取max获得，所以初始值设为0，右边界需要通过取min，初始值设为Integer.MAX\_VALUE.

public int maximalRectangle(char[][] matrix) {

if (matrix == null || matrix.length == 0) return 0;

int m = matrix.length, n = matrix[0].length;

int[] left = new int[n], right = new int[n];

int[] height = new int[n];

Arrays.fill(right, Integer.MAX\_VALUE);

int max = 0;

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

// 新的一行，初始化curLeft = 0

int curLeft = 0; // 存储的是当前行走到j时，j左侧的1的边界

for (int j = 0;j < n;j++) {

if (matrix[i][j] == '1') {

height[j]++;

// 此处left[j]是上一行的左侧边界，curLeft是这一行的左侧边界，用max来取交集

left[j] = Math.max(left[j], curLeft);

} else {

height[j] = 0;

left[j] = 0;

curLeft = j + 1;

}

}

int curRight = n - 1; // 初始化curRight = n-1;

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

if (matrix[i][j] == '1') {

right[j] = Math.min(right[j], curRight);

// left[j], right[j]左右边界，i下边界，i + height[j]-1上边界

max = Math.max(max, (right[j] - left[j] + 1)\*height[j]);

} else {

right[j] = Integer.MAX\_VALUE;

curRight = j - 1;

}

}

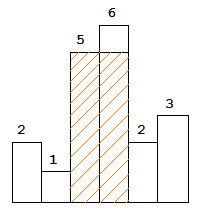
}

return max;

}

84. Largest Rectangle in Histogram

Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.

   
Given heights = [2,1,5,6,2,3],  
return 10.

**S1. Naive enumeration**

从每个heights[i]，把height[i]当做计算的高度，向左向右扫，找到左右边界，也就是小于heights[i]的位置为止。

time O(n^2)

注意边界条件：left最小值可能是-1, right最大值可能是n，并且left, right都是exclusive

所以在计算area时(right – left - 1)\*height

public int largestRectangleArea(int[] heights) {

if (heights == null || heights.length == 0) return 0;

int max = 0, n = heights.length;

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

int left = i - 1, right = i + 1;

while (left >= 0 && heights[left] >= heights[i]) left--;

while (right < n && heights[right] >= heights[i]) right++;

max = Math.max(max, (right - left - 1)\*heights[i]);

}

return max;

}

**S2.Divide and Conquer O(nlogn)**

把矩阵拆成左右两部分，有三种可能性：最大面积完全属于右边，完全属于左边，跨越两边。第三种可能性：一开始加入两边的边界，然后持续加入两侧中较大的，更新面积。

public int largestRectangleArea(int[] heights) {

return recursion(heights, 0, heights.length);

}

public int recursion(int[] heights, int left, int right) {

if (left == right) return 0;

if (left + 1 == right) return heights[left];

int mid = (left + right) / 2;

int a1 = recursion(heights, left, mid);

int a2 = recursion(heights, mid, right);

int a3 = 0, i = mid - 1, j = mid;

int h = Integer.MAX\_VALUE;

while (i >= left || j < right) {

if (i < left || (j < right && heights[j] > heights[i])) {

h = Math.min(h, heights[j++]);

} else {

h = Math.min(h, heights[i--]);

}

a3 = Math.max(a3, (j - i - 1)\*h);

}

return Math.max(Math.max(a1, a2), a3);

}

**S3. Stack 可以优化到O(n)**

在从左到右扫描histogram的过程中计算left，维护一个stack，stack 中保存的是index，保持stack中的i对应的heights[i]递减，到每个i时，所有>= 当前heights[i]的元素都会被弹出栈，只有小于栈顶元素的才会保留。

访问heights[i]时的栈顶元素即为对应的left位置，用数组保存left值，再从右到左遍历获取right值。

worst case; O(n^2).

public int largestRectangleArea(int[] heights) {

if (heights == null || heights.length == 0) return 0;

int max = 0, n = heights.length;

Stack<Integer> s = new Stack<>();

int[] left = new int[n];

s.push(-1);

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

while (s.peek() >= 0 && heights[s.peek()] >= heights[i]) s.pop();

left[i] = s.peek();

s.push(i);

}

s = new Stack<>();

s.push(n);

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

while (s.peek() < n && heights[s.peek()] >= heights[i]) s.pop();

int right = s.peek();

max = Math.max(max, (right - left[i] - 1)\*heights[i]);

s.push(i);

}

return max;

}

**继续优化：O(n)**

其实不需要array保存left，从左到右再从右到左，可以在stack中保存left值，

也就是说，假设高度是1 3 6 5 4 2, 那么

* 我们在stack中只保存递增的值，1,3,6，
* 当发现下一个为5， < heights[s.peek()], 说明以6为高度来计算的left，right都已经取到了，right即为5的index，h = heights[s.peek()], 然后把6 pop后的s.peek就是left。
* 所以以6为高度，进行计算的area = (index of 5 – s.peek after pop - 1)\*6
* 有可能最后stack中还有剩余的值，它们后面的所有值都比它们高，就是直到最后，right都没有找到的，所以他们的right都是n，left是自己index - 1。
* init 先把-1放入stack，用于判断边界，不然每次都要判断是否为空。

每个值都push进一次，拿出去一次，所以是O(n)。

public int largestRectangleArea(int[] heights) {

Stack<Integer> s = new Stack<>();

int max = 0, n = heights.length;

s.push(-1);

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

while (s.peek() >= 0 && heights[s.peek()] >= heights[i]) {

int area = heights[s.pop()] \* (i - s.peek() - 1);

max = Math.max(max, area);

}

s.push(i);

}

while (s.peek() >= 0) {

int area = heights[s.pop()] \* (n - s.peek() - 1);

max = Math.max(max, area);

}

return max;

}

334. Increasing Triplet Subsequence

Given an unsorted array return whether an increasing subsequence of length 3 exists or not in the array. Formally the function should:

Return true if there exists i, j, k such that arr[i] < arr[j] < arr[k] given 0 ≤ i < j < k ≤ n-1 else return false.

Your algorithm should run in O(n) time complexity and O(1) space complexity.

**S1. Binary Search** 和LIS的binary Search一个做法 time O(nlogn)

dp中只存逐渐增大的数字，dp中最后一位是最大的，

每取到一个新的nums[i]，

* 如果大于dp最大值，直接append
* 否则在dp中做binary search，更新相应位的值。

public boolean increasingTriplet(int[] nums) {

if (nums == null || nums.length < 3) return false;

int n = nums.length;

List<Integer> dp = new ArrayList<>();

dp.add(nums[0]);

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

if (nums[i] > dp.get(dp.size() - 1)) {

if (dp.size() >= 2) return true;

dp.add(nums[i]);

} else {

int pos = Collections.binarySearch(dp, nums[i]);

if (pos < 0) pos = -(pos + 1);

dp.set(pos, nums[i]); // nums[i]比原来这个位置的值更小

}

}

return false;

}

**S2. ArrayLIS特性 O(n)**

这里的单调栈长度=3，非常小，所以存两个数即可，分别代表单调栈的第一个，第二个位置。碰到第三个位置直接返回true即可。

可以推广到任意较小的常数k。

public boolean increasingTriplet(int[] nums) {

if (nums == null || nums.length < 3) return false;

int n = nums.length;

int min1 = Integer.MAX\_VALUE, min2 = Integer.MAX\_VALUE;

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

if (nums[i] <= min1) {

min1 = nums[i];

} else if (nums[i] <= min2) {

min2 = nums[i];

} else return true;

}

return false;

}

354. Russian Doll Envelopes

You have a number of envelopes with widths and heights given as a pair of integers (w, h). One envelope can fit into another if and only if both the width and height of one envelope is greater than the width and height of the other envelope.

What is the **maximum number of envelop**es can you Russian doll? (put one inside other)

Given envelopes = [[5,4],[6,4],[6,7],[2,3]], the maximum number of envelopes you can Russian doll is 3 ([2,3] => [5,4] => [6,7]).

**S1. LIS**

**先按照width升序排序，如果width相同，就按照height降序排序**，保证同样width的不会被取到两个。

之后就相当于是求height的longest increasing subsequence的长度，用300-LIS的做法，time O(nlogn).

public int maxEnvelopes(int[][] envelopes) {

if (envelopes == null || envelopes.length == 0 || envelopes[0].length != 2) return 0;

Arrays.sort(envelopes, new Comparator<int[]>() {

public int compare(int[] a1, int[] a2) {

if (arr1[0] == arr2[0]) return arr2[1] - arr1[1]; *// 从大到小*

else return arr1[0] - arr2[0]; *// 从小到大*

}

});

List<Integer> dp = new ArrayList<>();

for (int[] envelop:envelopes) {

if (dp.isEmpty() || envelop[1] > dp.get(dp.size() - 1)) {

dp.add(envelop[1]);

} else {

int idx = Collections.binarySearch(dp, envelop[1]);

if (idx < 0) idx = -(idx + 1);

dp.set(idx, envelop[1]);

}

}

return dp.size();

}

368. Largest Divisible Subset

Given a set of distinct positive integers, find the largest subset such that **every pair (Si, Sj) of elements** in this subset satisfies: Si % Sj = 0 or Sj % Si = 0.

If there are multiple solutions, return any subset is fine.

Example 1: nums: [1,2,3] Result: [1,2] (of course, [1,3] will also be ok)

**S1.LIS time O(n^2)**

首先sort, 这样在计算mod的时候，只要计算较大值mod较小值是否为0即可。

用parent存前一个数字的index，便于后续返回list。

count[i] : maxEndHere, stores the largest number of LDS including nums[i]. 并更新maxSoFar.

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

if (nums == null) return null;

int maxSoFar = 0, maxSoFarIdx = -1, n = nums.length;

int[] maxEndHere = new int[n], parent = new int[n];

Arrays.sort(nums);

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

maxEndHere[i] = 1; // 初始化

parent[i] = -1;

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

// when: meet the requirement && could be larger than existing solution:

if (nums[i] % nums[j] == 0 && maxEndHere[j] + 1 > maxEndHere[i]) {

maxEndHere[i] = maxEndHere[j] + 1;

parent[i] = j;

}

}

if (maxEndHere[i] > maxSoFar) {

maxSoFar = maxEndHere[i];

maxSoFarIdx = i;

}

}

List<Integer> out = new ArrayList<>();

while (maxSoFarIdx != -1) {

out.add(0, nums[maxSoFarIdx]);

maxSoFarIdx = parent[maxSoFarIdx];

}

return out;

}

**Binary Search**

240. Search a 2D Matrix II

Write an efficient algorithm **that searches for a value in an m x n matrix**. This matrix has the following properties:

Integers in each row are sorted in ascending from left to right.

Integers in each column are sorted in ascending from top to bottom.

For example, Consider the following matrix:

[ [1, 4, 7, 11, 15],

[2, 5, 8, 12, 19],

[3, 6, 9, 16, 22],

[10, 13, 14, 17, 24],

[18, 21, 23, 26, 30]]

Given target = 5, return true. Given target = 20, return false.

题目的重点是：行和列都是sort过的

**S1.Binary Search**

创意的思路，从左下角开始走，如果< target，就往右，如果> target, 就往上。

Time O(m+n), 每次都可以排除一行，或者一列

public boolean searchMatrix(int[][] matrix, int target) {

if(matrix == null || matrix.length == 0) return false;

int m = matrix.length, n = matrix[0].length;

int x = m - 1, y = 0; // 左下角

while (x >= 0 && y < n) {

int cur = matrix[x][y];

if (cur == target) return true;

else if (cur < target) y++;

else x--;

}

return false;

}

Matrix 2D Search

比刚刚的题简单，第一行的所有值都一定小于第二行，小于第三行......

**S1. Binary Search**

要考虑的点只是2d index转换为1d

public boolean matrixBinarySearch(int[][] matrix, int target) {

if (matrix == null || matrix.length == 0) return false;

int m = matrix.length, n = matrix[0].length;

if (target < matrix[0][0] || target > matrix[m-1][m-1]) return false;

int front = 0, back = m\*n - 1;

while (front <= back) {

int mid = (front) + (back - front)/2;

int row = mid / m, col = mid % m;

if (matrix[row][col] == target) return true;

if (matrix[row][col] < target) {

front = mid + 1;

} else {

back = mid - 1;

}

}

return false;

}

34. Search for a Range

Given an array of integers sorted in ascending order, find **the starting and ending position of a given target value.**

Your algorithm's runtime complexity must be in the order of O(log n).

If the target is not found in the array, return [-1, -1].

For example, Given [5, 7, 7, 8, 8, 10] and target value 8, return [3, 4].

**S1.Binary Search**

首先搜索最左边的target，再作为参数传入，搜索最右边的target。

最左边的target的条件：nums[i] == target && (i == 0 || nums[i-1] < target)

最右边的target的条件：nums[i] == target && (i == n-1 || nums[i+1] > target)

Time O(logn)

public int[] searchRange(int[] nums, int target) {

if (nums == null || nums.length == 0) return new int[]{-1, -1};

int front = binarySearch(nums, target, -1);

int back = front == -1 ? -1 : binarySearch(nums, target, front);

return new int[]{front, back};

}

// left == -1: 找最左边的target, else 找最右边的target

public int binarySearch(int[] nums, int target, int left) {

int i = left < 0 ? 0 : left;

int j = nums.length - 1;

while (i < j) {

int mid = i + (j - i)/2;

if (nums[mid] > target) j = mid - 1;

else if (nums[mid] < target) i = mid + 1;

else {

if (left < 0) { *// searching for left bound*

if (mid == 0 || nums[mid-1] < target) return mid;

else j = mid - 1;

} else {

if (mid == nums.length - 1 || nums[mid+1] > target) return mid;

else i = mid + 1;

}

}

} *// break only when i == j*

return nums[i] == target ? i : -1;

}

33. Search in Rotated Sorted Array

Suppose a sorted array 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.

**S1. Binary Search**

构造一个helper function：已知sub-array最左边和最右边的两个值，判断target是不是在这个sub-array中：有两种情况，sub-array中有转折点和没有转折点：

没有转折点要满足的条件：left <= target && target <= right

有转折点，要满足的条件：right < left && (target >= left || target <= right)

通过这个helper函数来判断应该更新left还是right指针。

*// range:[left, right-1], right not included.*

*// 这里的left, right是具体的数字，而不是index*

boolean inRange(int left, int right, int target) {

// 两种情况满足任意一种，都在[left, right)中

boolean normal = left <= target && target <= right;

boolean mix = left > right && (target >= left || target <= right);

return normal || mix;

}

public int search(int[] nums, int target) {

if (nums == null || nums.length == 0) return -1;

int i = 0, j = nums.length - 1;

while (i < j) {

int mid = i + (j - i) / 2;

if (nums[mid] == target) return mid;

else if (inRange(nums[i], nums[mid], target)) j = mid - 1;

else i = mid + 1;

}

return nums[i] == target ? i : -1;

}

81. Search in Rotated Sorted Array II

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

Write a function to determine if a given target is in the array. **The array may contain duplicates**.

和上道题的区别就是，允许有重复数字

**S1. Binary Search**

其实有没有duplicate并不影响搜索过程。时间仍然是O(logn)

*// this won't affect time complexity, still O(logn).*

*// 7 8 9 1 2 3 4 5 6*

public boolean search(int[] nums, int target) {

if (nums == null || nums.length == 0) return false;

int low = 0, high = nums.length - 1;

while (low < high) {

int mid = (low + high) / 2;

if (nums[mid] == target) return true;

if (nums[mid] > nums[high]) {

if (nums[mid] > target && nums[low] <= target) high = mid;

else low = mid + 1;

} else if (nums[mid] < nums[high]) {

if (nums[mid] < target && nums[high] >= target) low = mid + 1;

else high = mid;

} else {

high--;

}

}

return nums[low] == target;

}

153. Find Minimum 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).

Find the minimum element.

You may assume no duplicate exists in the array.

**S1.Binary Search**

对于rotated sorted array, 找最小值其实也就是找pivot的过程，

只有当: left < right && nums[left] > nums[right]的时候，pivot才在[left, right]范围中

否则最小值就是nums[left]

public int findMin(int[] nums) {

int i = 0, j = nums.length - 1;

while (i < j && nums[i] > nums[j]) {

int mid = i + (j - i) / 2;

if (nums[mid] < nums[j]) j = mid; // 因为找的是最小值，可能就是mid, 所以j = mid

else i = mid + 1;

}

return nums[i];

}

154. Find Minimum in Rotated Sorted Array II

Follow up for "Find Minimum in Rotated Sorted Array": What if duplicates are allowed?

Would this affect the run-time complexity? How and why?

**S1. Binary Search with duplicates**

影响就是，刚刚的条件不满足了，可能会有nums[left] == nums[right]，数组两端的值相等时不能砍掉一半，只能从其中某一边往中间走，不能两边都往中间走，因为有可能这两个边上的值其实是唯二的最小值。

worst case time O(n)

public int findMin(int[] nums) {

if (nums == null || nums.length == 0) return -1;

int front = 0, back = nums.length - 1;

while (front < back && nums[front] >= nums[back]) {

if (nums[front] == nums[back]) {

front++;

} else {

int mid = front + (back - front) / 2;

if (nums[mid] <= nums[back]) back = mid;

else front = mid + 1;

}

}

return nums[front];

}

162. Find Peak Element

A peak element is an element that is greater than its neighbors.

Given an input array where num[i] ≠ num[i+1], find a peak element and return its index.

The array may contain multiple peaks, in that case return index to any one of the peaks is fine.

You may imagine that num[-1] = num[n] = -∞.

For example, in array [1, 2, 3, 1], 3 is a peak element and your function should return the index number 2.

**S1.Binary Search, time O(nlogn)**

重要条件：相邻数字绝不重复 + 数组两端有-inf

所以在[0, n-1]内一定存在peak，可以通过binary search逐渐缩小有效区间的大小。

通过check mid，

* 如果nums[mid] < nums[left], 说明mid可以替代数组右端的-inf作用，在[left, mid-1]中一定有peak，可以缩小区间，right = mid-1
* 如果nums[mid] < nums[right], 说明mid可以替代数组左端的-inf作用
* 同理，如果nums[mid] < nums[mid - 1], 则mid也可以替代右端-inf
* nums[mid] < nums[mid + 1], 则mid可以替代左端-inf

**为了保证3,4的有效性，left, right不能相邻，需要满足left + 1 < right**.

public int findPeakElement(int[] nums) {

int left = 0, right = nums.length - 1;

while (left + 1 < right) {

int mid = (left + right) / 2;

if (nums[mid] < nums[left] || nums[mid] < nums[mid - 1]) {

right = mid - 1;

} else if (nums[mid] < nums[right] || nums[mid] < nums[mid + 1]) {

left = mid + 1;

} else {

return mid;

}

}

return nums[left] > nums[right] ? left : right;

}

4 Median of Two Sorted Arrays

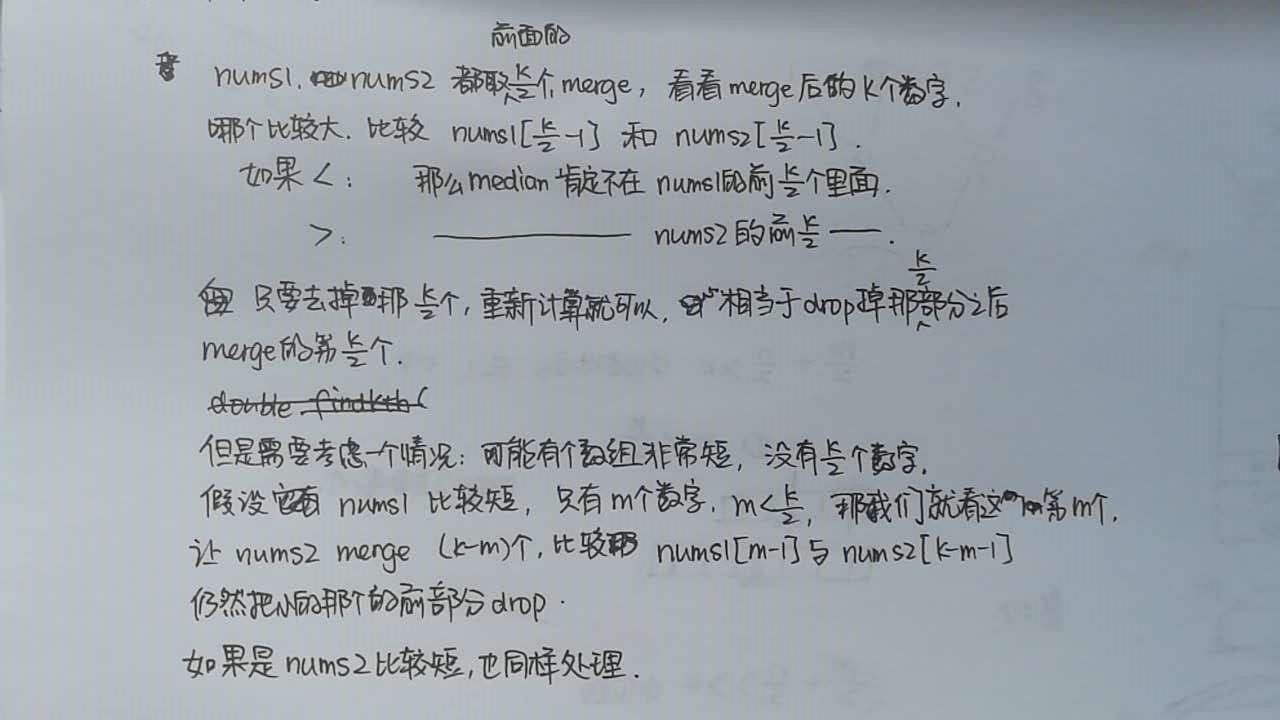
There are two sorted arrays nums1 and nums2 of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

**S1. Binary Search**

假设nums1长度是m, nums2长度是n, 找到这两者的median，其实也就是找第这m+n个数中间的那一个的取值。

* 如果(m + n) 是奇数，取中间一个即可；
* 如果是偶数，取中间两个，取平均即可。



public double findMedianSortedArrays(int[] nums1, int[] nums2) {

int m = nums1.length, n = nums2.length;

int len = m + n;

if (len % 2 == 1) {

return findKth(nums1, 0, m, nums2, 0, n, len / 2 + 1);

} else {

return (findKth(nums1, 0, m, nums2, 0, n, len / 2)

+ findKth(nums1, 0, m, nums2, 0, n, len / 2 + 1)) / 2;

}

}

*// assume m <= n*

*// m/n: length of nums1, nums2. k: k-th number to search for.*

*// start from start1 and start2.*

double findKth(int[] nums1, int start1, int m, int[] nums2, int start2, int n, int k) {

if (m > n) return findKth(nums2, start2, n, nums1, start1, m, k);

if (m == 0) return nums2[k - 1]; *// k-th number of nums2*

if (k == 1) return Math.min(nums1[start1], nums2[start2]);

*// divide into 2 parts, considered the condition m < k/2:*

int mida = Math.min(k/2, m), midb = k - mida;

if (nums1[start1 + mida - 1] >= nums2[start2 + midb - 1]) {

return findKth(nums1, start1, m, nums2, start2 + midb, n - midb, k - midb);

} else {

return findKth(nums1, start1 + mida, m - mida, nums2, start2, n, k - mida);

}

}

69. Sqrt(x)

Implement int sqrt(int x).

Compute and return the square root of x.

**S1. loop**

从1开始一个个往上接近

**S2. Binary Search**

x最大的取值是Integer.MAX\_VLUAE, 那么只要保证mid \* mid > x的判断可以覆盖Integer.MAX\_VLUAE即可。

两端的值可以分别取1，Integer.MAX\_VALUE, 然后往中间靠拢。

需要考虑当没有恰好的sqrt时，应该取floor值，所以在判断的时候是在mid\*mid > x 时取j = mid – 1

public int mySqrt(int x) {

if (x <= 0) return 0;

long i = 1, j = Integer.MAX\_VALUE;

while (i < j) {

long mid = i + (j-i)/2;

if (mid \* mid > x) {

j = mid - 1; *// floor*

} else {

if ((mid + 1) \* (mid + 1) > x) return (int)mid;

i = mid + 1;

}

}

return (int)i;

}

278. 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.

**S1. Binary Search**

就是非常典型的binary search，不停的移动，通过mid来判断左右。

public int firstBadVersion(int n) {

if (n <= 0) return 0;

return helper(0, n);

}

private int helper(int start, int end) {

if (start == end) return start;

int mid = start + (end - start)/2;

if (isBadVersion(mid)) {

return helper(0, mid);

} else {

return helper(mid + 1, end);

}

}

PocketGems - Find first occurrences of an element in a sorted array

具体题目在这里：<http://www.geeksforgeeks.org/find-first-last-occurrences-element-sorted-array/>

**S1. Naive 从左到右for loop, O(n)**

**S2. Binary Search O(logn)**

相比最简单的find index of element, 区别只是加了一个first occurance, 所以找到nums[i] == k

之后不直接返回，判断一下是不是first occurranence 然后要不要继续找即可。

public static int firstOccurance(int[] nums, int k) {

if (nums == null || nums.length == 0) return -1;

int low = 0, high = nums.length - 1;

while (low <= high) {

int mid = low + (high - low)/2;

if (nums[mid] == k) {

if (mid == 0 || nums[mid - 1] < k) return mid;

else high = mid - 1;

} else if (nums[mid] < k) { *// in right side*

low = mid + 1;

} else {

high = mid - 1;

}

}

return -1;

}

PocketGems: most left T in an array

题目是一个matrix，有T和F，要求找到最左边列的T。Rule是每一行，从左向右，只要有T，后面都是T

**S1. Binary Search**

第一行，先用binary search, 假设一共有n列，logn，假设求得最左边的是i

第二行，只需要找[0, i-1]之间就好

找到最后一行返回，或者index = 0直接返回

public static int leftMostT(char[][] grid) {

if (grid == null || grid.length == 0 || grid[0].length == 0) return -1;

int m = grid.length, n = grid[0].length;

int index = binarySearch(grid[0], 0, n-1);

int minIndex = index;

for (int i = 1;i < m;i++) {

int curIndex = binarySearch(grid[i], 0, minIndex - 1);

minIndex = Math.min(minIndex, curIndex);

if (minIndex == 0) break;

}

return minIndex;

}

*// find first appearance of 'T' in row, from [start, end]*

static int binarySearch(char[] row, int start, int end) {

while (start <= end) {

int mid = start + (end - start)/2;

if (row[mid] == 'T') {

if (mid == 0 || row[mid - 1] != 'T') return mid;

else end = mid - 1;

} else if (row[mid] != 'T') { *// in right side*

start = mid + 1;

}

}

return Integer.MAX\_VALUE;

}

Houzz - find a lower number

an array nums whose length is n satisfy the following condition:

nums[0] >= nums[1], nums[n-1]>= nums[n-2], find a number that satisfy:

nums[i] <= nums[i-1] and nums[i] <= nums[i+1]

**S1 for循环 O(n)**

**S1 Binary Search O(logn)**

每次选取中间点mid, 判断mid与两侧点的关系，

1. 如果满足条件，直接返回，
2. 如果只满足一边的条件，那么另一边一定有解，
3. 如果两边都不满足，那么两边都一定有解，随意选择一边即可。

为什么2成立？

比如2 1 ...1 2 3 4 5 6, 假设中间的2是mid点，左边不满足条件，右边满足条件，所以我们单看左边：

2 1 ... 1 2, 因为本身满足nums[0] > nums[1], 并且nums[mid - 1] < nums[mid], 所以这个subarray本身是从大到小再到大，所以这里面一定存在最低点，满足题目的条件，所以一定有解。

**Majority Element + Voting Algorithm**

169. Majority Element

Given an array of size n, find the majority element.

The majority element is the element that appears more than**⌊n/2⌋** times.

You may assume that the array is non-empty and the majority element always exist in the array.

229. Majority Element II

Given an integer array of size n, find all elements that appear more **than ⌊ n/3 ⌋** times. The algorithm should run in linear time and in O(1) space.

Lintcode - Majority Number III

Given an array of integers and a number k, the majority number is the number that occurs more **than 1/k of the** size of the array.

Find it.

http://www.lintcode.com/en/problem/majority-number-iii/

**S. Moore’s voting**

解决任意1/k的majority number:

满足出现次数大于1/k的majority number最多有k-1个，所以保存一个长度为k – 1的数组count[]，

对nums遍历一次：

* if数组中有nums[i], 则该数count ++
* else if数组中有空位，则放入count, 且该位的count置为1
* else if数组中有某个count为0，则用nums[i]替换这个数, 该位count置为1
* else 所有数字的count--

剩下的数就是candidate，再扫一遍数组，确认每个candidate的真正出现次数

关键是，为什么可以这么做？

比如[5,5,0,0,0,5,0,0,5... 不管majority candidate是谁，

假设放进nums，并保存 count的本身是众数，那么每次非candidate出现时，都会抵消一个candidate的count，

同理，假设非candidate被放进nums中，并计算count，那么每次走到candidate时，也会抵消一个

总之，非candidate和candidate的数量会互相抵消。而两个candidate之间不会互相抵消。

并且candidate的数量一定大于非candidate，所以抵消到最后，留下的一定是candidate.

public int majorityElement(int[] nums) {

int num = nums[0];

int count = 1;

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

if (num == nums[i]) {

count++;

} else if (count == 0) {

num = nums[i];

count = 1;

} else {

count--;

}

}

return num;

}

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

List<Integer> res = new ArrayList<>();

if (nums == null || nums.length == 0) return res;

int num1 = 0, num2 = 0;

int count1 = 0, count2 = 0;

int len = nums.length;

for (int i = 0;i < len;i++) {

if (num1 == nums[i]) {

count1++;

} else if (num2 == nums[i]) {

count2++;

} else if (count1 == 0) {

num1 = nums[i];

count1++;

} else if (count2 == 0) {

num2 = nums[i];

count2++;

} else {

count1--;

count2--;

}

}

count1 = count2 = 0;

for (int num:nums) {

if (num == num1) count1++;

else if (num == num2) count2++;

}

if (count1 > len/3) res.add(num1);

if (count2 > len/3) res.add(num2);

return res;

}

扩展，当推广到K时：

一共遍历两遍 O(2\*n + k), 用countMap保存candidate的num和对应的count，

用zeroIndex的hash set来保存count == 0的数字，注意在count++的时候，会有count原本为0的加到1，需要从zeroIndex中删掉。

public static int majorityNumber(int[] nums, int k) {

if (nums == null || nums.length == 0) return -1;

Map<Integer, Integer> countMap = new HashMap<>(); // <number, count>

int size = nums.length;

*// store number whose count reaches 0.*

Set<Integer> zeroIndex = new HashSet<>();

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

int cur = nums[i];

if (countMap.size() < k - 1 && !countMap.containsKey(cur)) {

countMap.put(cur, 1);

} else if (countMap.containsKey(cur)) {

countMap.put(cur, countMap.get(cur) + 1);

zeroIndex.remove(cur);

} else if (!zeroIndex.isEmpty()) { *// there are count==0*

int idx = zeroIndex.iterator().next();

countMap.remove((Integer)idx);

zeroIndex.remove(idx);

countMap.put(cur, 1);

} else {

for (Map.Entry<Integer, Integer> entry:countMap.entrySet()) {

int num = entry.getKey();

int count = entry.getValue() - 1;

countMap.put(num, count);

if (count == 0) zeroIndex.add(num);

}

}

}

for (int key :countMap.keySet()) {

countMap.put(key, 0);

}

double threthold = (double)size / k;

for (int num:nums) {

if (!countMap.containsKey(num)) continue;

int count = countMap.get(num) + 1;

if (count > threthold) return num;

countMap.put(num, count);

}

return -1;

}

**Matrix Inplace Operations**

73. 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.

Could you devise a constant space solution?

题解：

这道题的重点就在于优化空间：

可以用两个boolean分别记录第一行，第一列有没有0，然后再用matrix的第一行和第一列记录其他行列是否有0。

public void setZeroes(int[][] matrix) {

if (matrix == null || matrix.length == 0) return;

int m = matrix.length, n = matrix[0].length;

boolean firstRow = false, firstCol = false;

*// 1.check if 0 exist in first row and first column:*

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

if (matrix[i][0] == 0) {

firstCol = true;

break;

}

}

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

if (matrix[0][i] == 0) {

firstRow = true;

break;

}

}

*// 2.now use first row/col to set flag for other rows/cols*

for (int i = 1;i < row;i++) {

for (int j = 1;j < col;j++) {

if (matrix[i][j] == 0) matrix[i][0] = matrix[0][j] = 0;

}

}

*// 3.start setting zeros:*

for (int i = 1;i < row;i++) {

for (int j = 1;j < col;j++) {

if (matrix[i][0] == 0 || matrix[0][j] == 0) matrix[i][j] = 0;

}

}

if (firstRow) Arrays.fill(matrix[0], 0);

if (firstCol) {

for (int i = 0;i < row;i++) matrix[i][0] = 0;

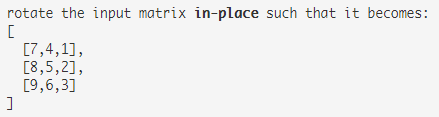
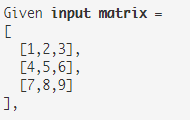
}

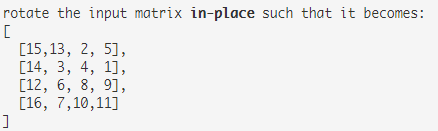
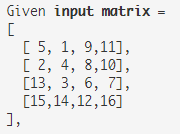
}

48. Rotate Image

You are given an n x n 2D matrix representing an image. Rotate the image by 90 degrees (clockwise).

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.





**S1 Math trick**

/\*

\* clockwise rotate

\* first reverse up to down, then swap the symmetry

\* 1 2 3 7 8 9 7 4 1

\* 4 5 6 => 4 5 6 => 8 5 2

\* 7 8 9 1 2 3 9 6 3

\*

\* anticlockwise rotate

\* first reverse left to right, then swap the symmetry

\* 1 2 3 3 2 1 3 6 9

\* 4 5 6 => 6 5 4 => 2 5 8

\* 7 8 9 9 8 7 1 4 7

\*/

public void rotate(int[][] matrix) {

if (matrix == null || matrix.length < 2) return;

int mid = matrix.length / 2;

int m = matrix.length, n = matrix[0].length;

for (int i = 0;i < mid;i++) {

for (int j = 0;j < n;j++) {

swap(matrix, i, j, m-1-i, j);

}

}

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

for (int j = i + 1;j < matrix[i].length;j++) {

swap(matrix, i, j, j, i);

}

}

}

private void swap(int[][] matrix, int i1, int j1, int i2, int j2) {

int tmp = matrix[i1][j1];

matrix[i1][j1] = matrix[i2][j2];

matrix[i2][j2] = tmp;

}

54. Spiral Matrix

Given a matrix of m x n elements (m rows, n columns), return all elements of the matrix in spiral order.

Given the following matrix: You should return [1,2,3,6,9,8,7,4,5].



题解：

通过count来维护spiral的过程中，每次开始的和结束的位置。

public List<Integer> spiralOrder(int[][] matrix) {

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

if (matrix == null || matrix.length == 0 || matrix[0].length == 0) return list;

int m = matrix.length, n = matrix[0].length;

int num = m\*n;

int i = 0, j = 0, count = 0;

while (list.size() < num) {

for (j = count;j < n - count;j++) {

list.add(matrix[i][j]);

}

if (list.size() == num) break;

for (i++, j--;i < m - count;i++) {

list.add(matrix[i][j]);

}

if (list.size() == num) break;

for (i--, j--;j >= count;j--) {

list.add(matrix[i][j]);

}

if (list.size() == num) break;

for (i--, j++;i > count;i--) {

list.add(matrix[i][j]);

}

count++;

i++;

}

return list;

}

59. Spiral Matrix II

Given an integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.

Given n = 3, You should return the following matrix:



S1

和上道题的区别在于，matrix一定是正方形，所以不需要在while loop内部再判断size

public int[][] generateMatrix(int n) {

int[][] matrix = new int[n][n];

int i = 0, j = 0, count = 0;

int num = n\*n;

int idx = 1;

while (idx <= num) {

for (j = count;j < n - count;j++) {

matrix[i][j] = idx++;

}

for (i++, j--;i < n - count;i++) {

matrix[i][j] = idx++;

}

for (j--, i--;j >= count;j--) {

matrix[i][j] = idx++;

}

for (i--, j++;i > count;i--) {

matrix[i][j] = idx++;

}

count++;

i++;

}

return matrix;

}

65. 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.

**S1.just be cautious**

1.check all the valid character and its valid conditions:

- +/-: at index 0 || one position after e

- e: after number, and only once, and there should be number after e.

- . : after number, and only once, and cannot be after e

- digital number: none

then we need to keep track of: if number, dot, e have been seen before, and

public boolean isNumber(String s) {

s = s.trim();

boolean pointSeen = false, eSeen = false;

boolean numberSeen = false;

boolean numberAfterE = true;

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

char cur = s.charAt(i);

if (cur >= '0' && cur <= '9') {

numberSeen = true;

numberAfterE = true;

} else if (cur == '.') {

if (eSeen || pointSeen) return false;

pointSeen = true;

} else if (cur == 'e') {

if (eSeen || !numberSeen) return false;

numberAfterE = false;

eSeen = true;

} else if (cur == '-' || cur == '+') {

if (i != 0 && s.charAt(i - 1) != 'e') return false;

} else {

return false;

}

}

return numberSeen && numberAfterE;

}

8. String to Integer (atoi)

Implement atoi to convert a string to an integer.

Hint: Carefully consider all possible input cases. If you want a challenge, please do not see below and ask yourself what are the possible input cases.

Notes: It is intended for this problem to be specified vaguely (ie, no given input specs). You are responsible to gather all the input requirements up front.

**S1. Just consider all the conditions**

**这里处理了overflow的情况，注意对overflow的判断。**

在这里，“ -12a43”会在遇到a时break，只取之前的有效数字，所以返回的是-12.

所以需要处理的特殊情况有：

前面的空格，正负号，overflow，非数字

public int myAtoi(String str) {

if (str == null || str.isEmpty()) return 0;

int i = 0, n = str.length();

boolean signSeen = false;

int sign = 1;

int num = 0;

while (str.charAt(i) == ' ') i++; *// leading spaces*

while (i < n) {

char c = str.charAt(i);

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

if (signSeen) return 0;

signSeen = true;

sign = c == '-' ? -1 : 1;

} else if (Character.isDigit(c)) {

if (num > Integer.MAX\_VALUE/10

|| (num == Integer.MAX\_VALUE/10 && c > '7')) {

return sign == 1 ? Integer.MAX\_VALUE : Integer.MIN\_VALUE;

}

num = 10\*num + c - '0';

} else {

break;

}

i++;

}

return sign \* num;

}

**Houzz -** Convert String to double if valid

if not, return 0.0. For example:

"12.3459" -> 12.3459

" 12.3459 " -> 12.3459

"12.3 459" -> Exception

"1 2.3459" -> Exception

"12.34x59" -> Exception

"12.34.59" -> Exception

**以及follow up: 只允许从头到尾走一遍，连trim也不可以用。**

和valid number的套路一致，

* 小数点：dotSeen, 最多一次，并且必须在数字后面，遇见则记录index。
* -/+: symbolSeen只能见一次，且必须是第一位
* 其他字符出现都算异常

在parse的过程中，遇到异常立刻返回。negative记录正负， dotIndex记录数字从第几位开始，也就是前面有没有正负符号。

只有在确定valid的情况下转换，整数从左到右，小数从右到左。

public static double convertStringToDouble(String s) {

if (s == null || s.length() == 0) return 0.0;

s = s.trim();

boolean symbolSeen = false;

boolean negative = false;

boolean dotSeen = false;

int dotIndex = s.length(), start = 0;

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

char c = s.charAt(i);

if (c == '.') {

if (dotSeen) return 0.0;

dotSeen = true;

dotIndex = i;

} else if (c == '+' || c == '-') {

if (symbolSeen || i != 0) return 0.0;

else negative = c == '-' ? true : false;

start = 1;

} else if (!Character.isDigit(c)){

return 0.0;

}

}

int integer = 0;

for (int i = start;i < dotIndex;i++) {

integer = integer \* 10 + s.charAt(i) - '0';

}

double decimal = 0.0; *// right to left.*

for (int i = s.length() - 1;i > dotIndex;i--) {

decimal = 0.1 \* decimal + s.charAt(i) - '0';

}

return (integer + decimal\*0.1) \* (negative ? -1 : 1);

}

**Follow up:**

一开始用while循环跳过spaces, 最后也用while 循环跳过spaces.

根据dotSeen的true/false判断是小数部分还是整数部分，整数可以直接算，

小数不仅要和整数做同样计算，还要算出要除的数字，比如1.23, 不仅要记录23, 还要记录100，来计算小数23/100. 因为没法直接从左往右算出。

public static Double convertStringToDouble(String s) {

if (s == null || s.length() == 0) return null;

int n = s.length();

boolean dotSeen = false;

boolean signSeen = false;

boolean negative = false;

boolean spacesLeft = false;

int whole = 0;

int decimal = 0;

int multiple = 1;

int spaceIdx = 0;

while (s.charAt(spaceIdx) == ' ') spaceIdx++;

for (int i = spaceIdx;i < n;i++) {

char c = s.charAt(i);

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

if (signSeen) return null;

signSeen = true;

negative = c == '-' ? true : false;

} else if (c == '.') {

if (dotSeen) return null;

dotSeen = true;

} else if (Character.isDigit(c)) {

if (dotSeen) { *// decimal part*

decimal = 10 \* decimal + c - '0';

multiple \*= 10;

} else { *// integer part*

whole = 10 \* whole + c - '0';

}

} else {

*// valid part end, check if there are invalid char later.*

spacesLeft = true;

spaceIdx = i;

break;

}

}

*// skip spaces in the end*

while (spacesLeft && spaceIdx < n) {

if (s.charAt(spaceIdx) == ' ') spaceIdx++;

else return null;

}

return (negative ? -1 : 1) \* (whole + (double)decimal/multiple);

}

7. Reverse Integer

Reverse digits of an integer.

Example1: x = 123, return 321，Example2: x = -123, return -321

The input is assumed to be a 32-bit signed integer. Your function should return 0 when the reversed integer overflows.

题解：

一般情况下会直接result = result\*10 + x%10,

但在这里，需要判断overflow，所以先传给tmp，判断没有overflow的情况后再传给result

*// -123*

public int reverse(int x) {

int result = 0;

while (x != 0) {

int tmp = result \* 10 + x % 10;

x /= 10;

*// check overflow:*

if (tmp/10 != result) return 0;

result = tmp;

}

return result;

}

**Houzz -** 给一个string s，找出其中出现次数最多的数字

/最大的top k个数字

**S DFA**

遍历字符，isNegative判定数字正负。

corner case：string中的数字数目 < k, string本身无效，返回的top k是否允许重复。

出现次数最多：

public static int findMostFrequentNumberInString(String s) {

if (s == null || s.length() == 0) return -1;

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

boolean isNegative = false;

Integer cur = 0;

int n = s.length(), max = -1, maxCount = -1;

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

char c = s.charAt(i);

if (Character.isDigit(c)) {

cur = cur == null ? c - '0' : cur \* 10 + c - '0';

if (i == n-1 || !Character.isDigit(s.charAt(i + 1))) {

cur = isNegative ? -1\*cur : cur;

int num = map.getOrDefault(cur, 0) + 1;

map.put(cur, num);

if (num > maxCount) {

maxCount = num;

max = cur;

}

cur = null;

}

} else if (c == '-') {

isNegative = true;

} else {

isNegative = false;

}

}

return max;

}

top k:

public static List<Integer> findTopKNumberInString(String s, int k) {

if (s == null || s.length() == 0) return null;

PriorityQueue<Integer> pqueue = new PriorityQueue<>(Collections.reverseOrder());

Integer cur = 0;

boolean isNegative = false;

int n = s.length();

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

char c = s.charAt(i);

if (Character.isDigit(c)) {

cur = cur == null ? c - '0' : cur \* 10 + c - '0';

if (i == n-1 || !Character.isDigit(s.charAt(i + 1))) {

cur = isNegative ? -1\*cur : cur;

pqueue.offer(cur);

cur = null;

}

} else if (c == '-') {

isNegative = true;

} else {

isNegative = false;

}

}

List<Integer> res = new ArrayList<>();

for (int i = 0;i < k && !pqueue.isEmpty();i++) {

res.add(pqueue.poll());

}

return res;

}