# LEETCODE SUMARRY

# BIT PART

## 29. Divide Two Integers

// RT O(logn)

//need consider case : (min\_vale, -1) and (min\_value, 1 )

public class Solution {

public int divide(int dividend, int divisor) {

if (divisor == 0) {return Integer.MAX\_VALUE;}

int sign = 1;

if ((dividend < 0 && divisor > 0) || (dividend > 0 && divisor < 0)){sign = -1;}

long div = (long) Math.abs((long)dividend);

long dis = (long) Math.abs((long)divisor);

int i = 31;

long ret = 0;

while (div >= dis){

while ( dis > (div >> i)){

i--;

}

ret ^= 1 << i;

div -= (dis << i);

}

if (sign == 1 && ((ret >> 31) & 1) == 1){return Integer.MAX\_VALUE;}

else {return sign \* (int)ret;}

}

}

## 67 [Add Binary](https://leetcode.com/problems/add-binary)

public class Solution {

public String addBinary(String a, String b) {

StringBuilder sb = new StringBuilder();

int carry = 0;

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

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

int m = i < 0 ? 0 : a.charAt(i) - '0';

int n = j < 0 ? 0 : b.charAt(j) - '0';

int nums = m + n + carry;

if (nums >= 2){sb.append(nums%2);carry = 1; i--;j--;}

else {sb.append(nums);carry = 0; i--;j--;}

}

if (carry == 1){sb.append(carry);}

return sb.reverse().toString();

}

}

## 136. Single Number

// ^ same is 0 , different is 1

// appeat twice became 0

// 0 ^ any number is that number;

//// RT: O(n) ; Space O(1)

public class Solution {

public int singleNumber(int[] nums) {

int check = 0;

for (int i : nums){

check ^= i;

}

return check;

}

}

## 137. Single Number II

//hint cunt the number in each bit

//if appear 3 times, to make it 0

// RT: O(n) ; Space O(1)

public class Solution {

public int singleNumber(int[] nums) {

int answer = 0;

int len = nums.length;

for (int i = 0; i < 32; i++){

int sum = 0;

for (int j : nums){

sum += (j >> i) & 1;

}

if (sum % 3 != 0 ){

answer ^= 1 << i;

}

}

return answer;

}

}

## 190 Reverse Bits

public class Solution {

// you need treat n as an unsigned value

public int reverseBits(int n) {

int m = 0, answer = 0;

for (int i = 0; i < 32; i++){

m = (n >> i) & 1;

answer |= m << (31 - i);

}

return answer;

}

}

## 191 Number of 1 Bits

public class Solution {

// you need to treat n as an unsigned value

public int hammingWeight(int n) {

int sum = 0;

for (int i = 0; i < 32; i++){

sum += (n >> i) & 1;

}

return sum;

}

}

## 260. Single Number III

//find the different bit of two numbers

// RT: O(n) ; Space O(1)

// 0 ^ number is number itself

public class Solution {

public int[] singleNumber(int[] nums) {

int check = 0, record = 0;

for (int i : nums){

check ^= i;

}

for (int i = 0; i < 32; i++){

if (((check >> i) & 1) == 1){

record = i;

break;

}

}

int num1 = 0, num2 = 0;

for (int i : nums){

if (((i >> record) & 1) == 1){num1 ^= i;}

else {num2 ^=i;}

}

return new int[]{num1,num2};

}

}

## 371 Sum of two Integers

public class Solution {

public int getSum(int a, int b) {

while (b != 0){

int carry = a & b;

a ^= b;

b = carry << 1;

}

return a ;

}

}

## 461 Hamming Distance

public class Solution {

public int hammingDistance(int x, int y) {

int val = x ^ y, dist = 0;

for (int i = 0; i < 32; i++){

dist += (val >> i) & 1;

}

return dist;

}

}

# BFS/DFS

## 127 Word Ladder

public class Solution {

public int ladderLength(String beginWord, String endWord, List<String> wordList) {

int level = 0, cap = 0;

if (beginWord.equals(endWord) || wordList.size() == 0){return 0;}

HashSet<String> set= new HashSet<>();

for (String i : wordList){set.add(i);}

if (!set.contains(endWord)){return 0;}

Queue <String> queue = new LinkedList<>();

queue.offer(beginWord);

while (!queue.isEmpty()){

cap = queue.size();

//level travel

while (cap-- > 0){

HashSet <String> temp = new HashSet<>(set);

String s = queue.poll();

if (s.equals(endWord)){return level + 1;}

//O(lens)

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

char [] cs = s.toCharArray();

for (char c = 'a'; c <= 'z'; c++){

cs[i] = c;

String t = new String (cs);

if (temp.contains(t)){

queue.offer(t);

set.remove(t);

}

}

}

}

level++;

}

return 0;

}

## 329. Longest Increasing Path in a Matrix

//(DFS + DP) 非常经典

// O(mn) RT and Space

public class Solution {

public int longestIncreasingPath(int[][] matrix) {

int row = matrix.length;

if (row == 0) return 0;

int col = matrix[0].length;

if (col == 0) return 0;

// dp is longest distance the point can go from this point

int [][] dp = new int [row][col];

int dist = 0;

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

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

dist = Math.max(dist, dfs (matrix, dp, i, j,Integer.MIN\_VALUE));

}

}

return dist;

}

public int dfs(int [][] matrix, int [][] dp, int i, int j, int prev){

int row = matrix.length;

int col = matrix[0].length;

if (i < 0 || i >= row || j < 0 || j >= col) return 0;

if (matrix[i][j] <= prev) return 0;

if (dp[i][j] != 0) return dp[i][j];//this is very important

int a = dfs (matrix, dp, i + 1, j,matrix[i][j]);

int b = dfs (matrix, dp, i - 1, j,matrix[i][j]);

int c = dfs (matrix, dp, i, j + 1,matrix[i][j]);

int d = dfs (matrix, dp, i, j - 1,matrix[i][j]);

dp[i][j] = Math.max(Math.max(a,b),Math.max(c,d)) + 1;

return dp[i][j];

}

}

## 397. Integer Replacement

//2. USE BFS

//1. Care of the Integer.MAX\_VALUE

//RT log(n)

public class Solution {

public int integerReplacement(int n) {

int level = 0;

int cap = 0;

Queue <Long> queue = new LinkedList<>();

HashSet<Long> visited = new HashSet<>();

queue.offer((long)n);

visited.add((long)n);

while (!queue.isEmpty()){

cap = queue.size();

while(cap-- > 0){

long cur = queue.poll();

System.out.println(cur);

if (cur == 1){return level;}

else if (cur % 2 != 0){

if (!visited.contains(cur + 1)){queue.offer(cur + 1);visited.add(cur + 1);}

if (!visited.contains(cur - 1)){queue.offer(cur - 1);visited.add(cur - 1);}

}

else {

if (!visited.contains(cur / 2)){queue.offer(cur / 2);visited.add(cur / 2);}

}

}

level++;

}

return -1;

}

}

## 547. Friend Circles

//USE BFS RT: O(n)

public class Solution {

int maxCycle = 0;

public int findCircleNum(int[][] M) {

int row = M.length;

if (row == 0){return 0;}

int col = M[0].length;

if (col == 0){return 0;}

if (col != row){return 0;}

HashSet<Integer> visited = new HashSet<>();

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

if (visited.contains(i)){continue;}

else{

maxCycle++;

bfs(visited, i, M);

}

}

return maxCycle;

}

public void bfs(HashSet<Integer> visited, int pos, int [][]M){

Queue<Integer> queue = new LinkedList<>();

queue.offer(pos);

visited.add(pos);

while(!queue.isEmpty()){

int nPos = queue.poll();

int [] nums = M[nPos];

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

if (M[nPos][j] == 1 && !visited.contains(j)){

visited.add(j);

queue.offer(j);

}

}

}

}

}

# Queue/Stack

## 155 Min Stack

public class MinStack {

/\*\* initialize your data structure here. \*/

int min;

Stack <Integer> stack;

Stack <Integer> storeMin;

public MinStack() {

stack = new Stack<>();

storeMin = new Stack<>();

}

public void push(int x) {

stack.push(x);

if (storeMin.isEmpty() || x <= min){

storeMin.push(x);

min = x;}

}

public void pop() {

int cur = stack.pop();

if (cur == storeMin.peek()){

storeMin.pop();

if (!storeMin.isEmpty()){min= storeMin.peek();}

}

}

public int top() {

return stack.peek();

}

public int getMin() {

return min;

}

}

/\*\*

\* Your MinStack object will be instantiated and called as such:

\* MinStack obj = new MinStack();

\* obj.push(x);

\* obj.pop();

\* int param\_3 = obj.top();

\* int param\_4 = obj.getMin();

\*/

## 225 Implement Stack using Queues

public class MyStack {

Queue <Integer> queue;

int cur;

/\*\* Initialize your data structure here. \*/

public MyStack() {

queue = new LinkedList();

}

/\*\* Push element x onto stack. \*/

public void push(int x) {

queue.offer(x);

cur = x;

}

/\*\* Removes the element on top of the stack and returns that element. \*/

public int pop() {

int i = queue.size();

while (i > 1){

int val = queue.poll();

queue.offer(val);

if (i == 2){cur = val;}

i--;

}

return queue.poll();

}

/\*\* Get the top element. \*/

public int top() {

return cur;

}

/\*\* Returns whether the stack is empty. \*/

public boolean empty() {

return queue.isEmpty();

}

}

/\*\*

\* Your MyStack object will be instantiated and called as such:

\* MyStack obj = new MyStack();

\* obj.push(x);

\* int param\_2 = obj.pop();

\* int param\_3 = obj.top();

\* boolean param\_4 = obj.empty();

\*/

## 232 Implement Queue using Stacks

public class MyQueue {

Stack<Integer> stack1;

Stack<Integer> stack2;

int cur;

/\*\* Initialize your data structure here. \*/

public MyQueue() {

stack1 = new Stack<Integer>();

stack2 = new Stack<Integer>();

}

/\*\* Push element x to the back of queue. \*/

public void push(int x) {

if (stack1.isEmpty()){cur = x;}

stack1.push(x);

}

/\*\* Removes the element from in front of queue and returns that element. \*/

public int pop() {

int val;

int i = stack1.size();

while (i > 1){

stack2.push(stack1.pop());

i--;

}

val = stack1.pop();

if (!stack2.isEmpty()){cur = stack2.peek();}

while (!stack2.isEmpty()){

stack1.push(stack2.pop());

}

return val;

}

/\*\* Get the front element. \*/

public int peek() {

return cur;

}

/\*\* Returns whether the queue is empty. \*/

public boolean empty() {

return stack1.isEmpty();

}

}

/\*\*

\* Your MyQueue object will be instantiated and called as such:

\* MyQueue obj = new MyQueue();

\* obj.push(x);

\* int param\_2 = obj.pop();

\* int param\_3 = obj.peek();

\* boolean param\_4 = obj.empty();

\*/

# Array

## Single Element in a Sorted Array

// Input: [3,3,7,7,10,11,11]

// Output: 10

public class Solution {

public int singleNonDuplicate(int[] nums) {

int len = nums.length;

//boundary check

if (nums[0] != nums[1]){return nums[0];}

if (nums[len - 1] != nums[len - 2]){return nums[len - 1];}

int i = 2, j = len - 3, mid = 0;

while (i <= j){

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

if (nums[mid] != nums[mid - 1] && nums[mid] != nums[mid + 1]){

return nums[mid];

}

else if (mid % 2 == 0){

if (nums[mid] == nums[mid + 1]){

i = mid + 2;

}

else {j = mid - 2;}

}

else {

if (nums[mid] == nums[mid - 1]){

i = mid + 1;

}

else {j = mid - 1;}

}

}

return 0;

}

}

## 11 Container With Most Water

public class Solution {

public int maxArea(int[] height) {

int len = height.length;

if (len <= 1) {return 0;}

int max = Integer.MIN\_VALUE;

int i = 0, j = len - 1;

while (i < j){

if (height[i] > height[j]){max = Math.max(max, (j - i) \* height[j--]);}

else {max = Math.max(max, (j - i) \* height[i++]);}

}

return max;

}

}

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

public class Solution {

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

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

Arrays.sort(nums);

int len = nums.length;

HashSet <List<Integer>> set = new HashSet<>();

if (len < 3){return rev;}

int i = 0;

int k = len - 1;

while (i < k){

int j = i + 1;

while (j < k){

if (nums[i] + nums[j] + nums[k] == 0){

set.add (new ArrayList <>(Arrays.asList(nums[i],nums[j],nums[k])));

j++;k--;

}

else if (nums[i] + nums[j] + nums[k] > 0){k--;}

else {j++;}

}

i++;k=len-1;

}

for (List<Integer> l : set){

rev.add(l);

}

return rev;

}

}

//running time : O (n^2), space O(1)

## 16 [3Sum Closest](https://leetcode.com/problems/3sum-closest)

public class Solution {

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

int len = nums.length;

if (len <= 2 ){

return Integer.MAX\_VALUE;

}

Arrays.sort(nums);

int s = 0;

int i = s + 1;

int e = len - 1;

int cls = nums [0] + nums[1] + nums[2];

while (s < e){

while (i < e){

int k = nums [s] + nums[e] + nums[i] ;

if (k == target){

return target;

}

else if (k > target){

cls = Math.abs(k - target) < Math.abs(cls - target) ? k : cls;

e--;

}

else {

cls = Math.abs(k - target) < Math.abs(cls - target) ? k : cls;

i++;

}

}

s++;

i = s + 1;

e = len - 1;

}

return cls;

}

}

## 18 [4Sum](https://leetcode.com/problems/4sum)

// be care of {0,0,0,0,0,0,0} use haseset to remove the duplicate;

public class Solution {

public List<List<Integer>> fourSum(int[] nums, int target) {

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

int len = nums.length;

Arrays.sort(nums);

if (len < 4){

return rev;

}

HashSet <List<Integer>> set = new HashSet <> ();

int start = 0;

int end = len - 1;

while (start < end){

int i = start + 1;

int j = end - 1;

while (i < end) {

while (i < j){

if (nums[i] + nums[j] + nums[start] + nums[end] == target){

set.add(Arrays.asList(nums[start],nums[i],nums[j],nums[end]));

i++;j--;

}

else if (nums[i] + nums[j] + nums[start] + nums[end] > target){

j--;

}

else {i++;}

}

end--;

j = end - 1;

i = start + 1;

}

start++;

end = len - 1;

}

for (List<Integer> l : set){

rev.add(l);

}

return rev;

}

}

//running time O(n^3)

## 26 Remove Duplicates from Sorted Array

public class Solution {

public int removeDuplicates(int[] nums) {

if (nums.length == 0 || nums.length ==1){return nums.length;}

int count =1;

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

if (nums[i] !=nums[i-1]) {nums[count++]=nums[i];}

}

return count;

}

}

## 27 Remove Element

public class Solution {

public int removeElement(int[] nums, int val) {

int cnt = 0;

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

if (nums[s] != val){

nums[cnt] = nums[s];

cnt++;

}

}

return cnt;

}

}

## 33 Search in Rotated Sorted Array

public class Solution {

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

int len = nums.length;

if (len == 0){return -1;}

int i = 0, j = len - 1;

while (i <= j){

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

int left = nums[i];

int right = nums[j];

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

if (left == target){return i;}

if (right == target){return j;}

if (left == right){return left == target ? mid: -1;}

else if (left > right){

if (nums[mid] >= left && (target > nums[mid] || target < left)){i = mid + 1;}

else if (nums[mid] < left && target < right && target > nums[mid]){i = mid + 1;}

else {j = mid - 1;}

}

else {

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

else {i = mid + 1;}

}

}

return -1;

}

}

## 34 Search for a Range

// idea find the fist elemement >= target

public class Solution {

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

int len = nums.length;

if (len == 0){return new int []{-1,-1};}

int cur = binarySearch (nums, target);

if (nums[cur] != target){return new int []{-1,-1};}

int next = binarySearch (nums, target + 1);

if (nums[next] == target){return new int []{cur,next};}

else {return new int []{cur,next - 1};}

}

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

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

while (i < j){

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

if (nums[mid] >= target){

j = mid;

}

else {

i = mid + 1;

}

}

return j;

}

}

## 53 Maximum Subarray

public class Solution {

public int maxSubArray(int[] nums) {

int len = nums.length;

if (len == 0) return len;

int preSum = 0;

int maxValue = Integer.MIN\_VALUE;

for (int i : nums){

preSum = Math.max(preSum + i, i);

maxValue = Math.max(preSum, maxValue);

}

return maxValue;

}

}

## 55 Jump Game

// A = [2,3,1,1,4], return true.

// A = [3,2,1,0,4], return false.

public class Solution {

public boolean canJump(int[] nums) {

int len = nums.length, max = 0;

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

if (max >= len - 1){return true;}

else {

if (i <= max){

max = Math.max(max,nums[i] + i);

}

}

}

return false;

}

}

75 Sort Colors

public class Solution {

public void sortColors(int[] nums) {

int len = nums.length;

if (len <= 1){return;}

int n0 = 0, n1 =0;

for (int i : nums){

if (i == 0){n0++;}

else if (i == 1){n1++;}

}

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

if (n0 != 0){nums[i] = 0; n0--;}

else if (n1 != 0){nums[i] = 1; n1--;}

else {nums[i] = 2;}

}

}

}

## 78 Subsets

public class Solution {

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

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

helpfunction (new ArrayList<Integer>(), 0, nums);

return rev;

}

public void helpfunction (List<Integer> sub, int pos, int [] nums){

rev.add(sub);

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

List<Integer> nsub = new ArrayList<Integer>(sub);

nsub.add(nums[i]);

helpfunction (nsub, i + 1, nums);

}

}

}

## 80 [Remove Duplicates from Sorted Array II](https://leetcode.com/problems/remove-duplicates-from-sorted-array-ii)

public class Solution {

public int removeDuplicates(int[] nums) {

if (nums.length < 3) {

return nums.length;

}

int j = 2;

for ( int i = 2; i < nums.length ; i++){

if (nums[j-2] != nums[i]){

nums[j] = nums[i];

j++;

}

}

return j;

}

}

## 84 Largest Rectangle in Histogram

public class Solution {

public int largestRectangleArea(int[] heights) {

int len = heights.length;

if (len == 0) {return 0;}

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

int maxArea = Integer.MIN\_VALUE;

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

if (i < len){

if (stack.isEmpty() || heights[stack.peek()] <= heights[i]){stack.push(i);}

else {

while (!stack.isEmpty()){

int top = stack.pop();

if (stack.isEmpty()){maxArea = Math.max(maxArea, heights[top] \* i);}

else {

maxArea = Math.max(maxArea, heights[top] \* (i - stack.peek() - 1));

if (heights[stack.peek()] <= heights[i]){break;}

}

}

stack.push(i);

}

}

else {

while (!stack.isEmpty()){

int top = stack.pop();

if (stack.isEmpty()){maxArea = Math.max(maxArea, heights[top] \* i);}

else {

maxArea = Math.max(maxArea, heights[top] \* (i - stack.peek() - 1));

}

}

}

}

return maxArea;

}

}

## 88 Merge Sorted Array

public class Solution {

public void merge(int[] nums1, int m, int[] nums2, int n) {

int k = m + n-1;

int index1 = m - 1;

int index2 = n - 1;

while (index1 > -1 && index2 > -1){

if (nums1[index1]<nums2[index2]) {

nums1[k] = nums2[index2];

k--;

index2--;

}

else{

nums1[k] = nums1[index1];

k--;

index1--;

}

}

System.out.println(k);

System.out.println(index1);

System.out.println(index2);

while (index1>-1){

nums1[k] = nums1[index1];

k--;

index1--;

}

while (index2>-1){

nums1[k] = nums2[index2];

k--;

index2--;

}

}

}

## 90 Subsets II

// If nums = [1,2,2], a solution is:

public class Solution {

HashSet<List<Integer>> rev = new HashSet<>();

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

Arrays.sort(nums);

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

help (sub,0,nums);

return new ArrayList<List<Integer>>(rev);

}

public void help (List<Integer> sub, int pos,int [] nums){

rev.add(sub);

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

List<Integer> nsub = new ArrayList<>(sub);

nsub.add(nums[i]);

help (nsub,i + 1,nums);

}

}

}

## 128 Longest Consecutive Sequence

public class Solution {

public int longestConsecutive(int[] nums) {

int len = nums.length;

if (len == 0){return 0;}

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

for (int i : nums){

set.add(i);

}

int max = Integer.MIN\_VALUE;

for (int i : set){

if (set.contains(i - 1)){continue;} // skip the value

int length = 0;

while (set.contains(i)){

length++;

i++;

}

max = Math.max(max, length);

}

return max;

}

}

## 139 Word Break

public class Solution {

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

int len = s.length();

HashSet<String> set = new HashSet<>(wordDict);

boolean [] dp = new boolean [len + 1]; dp[0] = true;

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

for (String sub : wordDict){

int k = i - sub.length() + 1;

dp[i + 1] = (k >=0 && set.contains(s.substring(k, i + 1)) && dp[k]) || dp[i + 1];

}

}

return dp[len];

}

}

## 152 Maximum Product Subarray

import java.lang.Math.\*;

public class Solution {

public int maxProduct(int[] nums) {

int max = nums[0];

int min = nums[0];

int maxFinal = nums[0];

int maxTemp;

int minTemp;

if (nums.length <2){

return maxFinal;

}

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

maxTemp= Math.max(Math.max(max\*nums[i],min\*nums[i]), nums[i]);

minTemp= Math.min(Math.min(max\*nums[i],min\*nums[i]), nums[i]);

max = maxTemp;

min = minTemp;

maxFinal = Math.max(maxFinal,max);

}

return maxFinal;

}

}

## 153 Find Minimum in Rotated Sorted Array

public class Solution {

public int findMin(int[] nums) {

int len = nums.length;

if (len == 1){return nums[0];}

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

int min = Integer.MAX\_VALUE;

while (i <= j){

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

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

if (nums [mid] >= nums [j]){min = Math.min(min,nums[j]); i = mid + 1;}

else {min = Math.min(min,nums[mid]); j = mid - 1;}

}

else {

min = Math.min(min,nums[i]);

break;

}

}

return min;

}

}

## 154 Find Minimum in Rotated Sorted Array II

public class Solution {

public int findMin(int[] nums) {

int len = nums.length;

if (len == 1){return nums[0];}

int i = 0, j = len - 1;

int min = Integer.MAX\_VALUE;

while (i <= j){

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

int m = nums[mid], l = nums[i], r = nums[j];

min = Math.min(min, l);

min = Math.min(min, r);

min = Math.min(min, m);

if (m == l && m == r){i++;j--;}

else if (l >= r) {

if (m >= l){i = mid + 1;}

else {j = mid - 1;}

}

else {break;}

}

return min;

}

}

## 162 Find Peak Element

public class Solution {

public int findPeakElement(int[] nums) {

int len = nums.length;

if (len == 0){return -1;}

else if (len == 1){return 0;}

else {

if (nums[0] > nums [1]){return 0;}

else if (nums[len - 2] < nums [len - 1]){return len - 1;}

}

int i = 1;

while (i - 1 >=0 && i + 1<= len - 1){

if (nums[i] > nums[i - 1] && nums[i] > nums[i + 1]){

return i;

}

i++;

}

return -1;

}

}

## 167 Two Sum II - Input array is sorted

public class Solution {

public int[] twoSum(int[] numbers, int target) {

int start = 0;

int end = numbers.length-1;

while (start <end){

if (numbers[start] + numbers [end] > target){

end--;

}

else if (numbers[start] + numbers [end] < target){

start++;

}

else {

break;

}

}

return new int [] {start+1,end+1};

}

}

## 169 Majority Element

public class Solution {

public int majorityElement(int[] nums) {

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

int len = nums.length;

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

int j = map.getOrDefault(nums[i], 0) + 1;

if (j > len / 2){return nums[i];}

map.put(nums[i],j);

}

return nums[0];

}

}

## 189 Rotate Array

public class Solution {

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

k %= nums.length;

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

rotateArray (nums,0,k-1);

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

}

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

while (start < end){

int temp = nums [start];

nums [start] = nums [end];

nums [end] = temp;

start++ ;

end-- ;

}

}

}

## 209 Minimum Size Subarray Sum

//[2,3,1,2,4,3] and s = 7,

//[4,3]

public class Solution {

public int minSubArrayLen(int s, int[] nums) {

if (nums.length == 0){return 0;}

int i = 0, j = 0, sum = 0, min = Integer.MAX\_VALUE;

while (i < nums.length){

sum += nums[i];

while (sum >= s){

//System.out.println(sum);

min = Math.min(min,i - j + 1);

sum -= nums[j];

j++;

}

i++;

}

return min == Integer.MAX\_VALUE ? 0 : min;

}

}

## 215 Kth Largest Element in an Array

public class Solution {

public int findKthLargest(int[] nums, int k) {

int s = 0;

int e = nums.length - 1;

int cur = partition (nums, s, e);

while (cur != nums.length - k){

if (cur < nums.length - k){ s = cur + 1;}

else {e = cur - 1;}

cur = partition (nums, s, e);

}

return nums[cur];

}

public int partition (int [] nums, int low, int high){

int i = low - 1;

for (int j = low; j < high; j++){

if (nums[j] < nums[high]){

i++;

swap(nums, i, j);

}

}

swap (nums, i + 1, high);

return (i + 1);

}

public void swap (int [] nums, int i, int j){

int temp = nums[i];

nums[i] = nums[j];

nums[j] = temp;

}

}

## 216 Combination Sum III

public class Solution {

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

public List<List<Integer>> combinationSum3(int k, int n) {

helpfunction (new ArrayList<Integer>(), 1, 0, k, n);

return rev;

}

public void helpfunction (List<Integer> sub, int pos, int sum, int k, int n){

if (sub.size() == k && sum == n){rev.add(sub);}

else if (sub.size() < k && sum < n){

for (int i = pos; i <= 9; i++){

List<Integer> nsub = new ArrayList<Integer>(sub);

nsub.add(i);

helpfunction (nsub, i + 1, sum + i, k, n);

}

}

}

}

## 228 Summary Ranges

//[0,1,2,4,5,7], return ["0->2","4->5","7"]

public class Solution {

public List<String> summaryRanges(int[] nums) {

List <String> rev = new ArrayList<>();

int len = nums.length;

if (len == 0){return rev;}

if (len == 1){rev.add(String.valueOf(nums[0]));return rev;}

int cnt = 1;

String s = "" + nums[0];

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

if (i == len - 1){

if (nums[i] == nums[i - 1] + 1){

rev.add(s + "->" + nums[i]);

}

else{

if (cnt == 1){

rev.add("" + nums[i - 1]);

}

else{

rev.add(s + "->" + nums[i - 1]);

}

rev.add("" + nums[i]);

}

}

else{

if (nums[i] == nums[i - 1] + 1){

cnt++;

}

else{

if (cnt == 1){

rev.add("" + nums[i - 1]);

}

else{

rev.add(s + "->" + nums[i - 1]);

cnt = 1;

}

s = "" + nums[i];

}

}

}

return rev;

}

}

## 229 Majority Element II

public class Solution {

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

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

int len = nums.length;

if (len == 0){return rev;}

int cnt1 = 0, cnt2 = 0, num1 = nums[0], num2 = nums[0];

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

if (nums[i] == num1){cnt1++;}

else if (nums[i] == num2){cnt2++;}

else if (cnt1 == 0){num1 = nums[i]; cnt1 = 1;}

else if (cnt2 == 0){num2 = nums[i]; cnt2 = 1;}

else{cnt1--;cnt2--;}

}

cnt1 = 0;

cnt2 = 0;

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

if (nums[i] == num1) {cnt1++;}

else if (nums[i] == num2) {cnt2++;}

}

if (cnt1 > len / 3) {rev.add(num1);}

if (cnt2 > len / 3) {rev.add(num2);}

return rev;

}

}

## 238 Product of Array Except Self

public class Solution {

public int[] productExceptSelf(int[] nums) {

int len = nums.length;

if (len <= 1){ return new int [0];}

int [] rev = new int [len];

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

if (i == 0) {rev[i] = 1;}

else {

rev[i] = rev[i - 1] \* nums[i - 1];

}

}

int right = 1;

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

rev[i] \*= right;

right \*= nums[i];

}

return rev;

}

}

## 239 Sliding Window Maximum

public class Solution {

public int[] maxSlidingWindow(int[] nums, int k) {

int len = nums.length;

if (len == 0){return new int [0];}

int [] rev = new int [len - k + 1];

ArrayDeque <Integer> queue = new ArrayDeque<>();

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

while (!queue.isEmpty() && queue.peekLast() < nums[i]){

queue.pollLast();

}

if (queue.isEmpty()){queue.offer(nums[i]);}

else if (queue.peekLast() >= nums[i]){queue.offer(nums[i]);}

}

rev[0] = queue.peek();

int count1 = 0, count2 = 0;

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

//consider the go out number

if (nums[i - k] == queue.peek()){queue.poll();}

//consider the inside number

while (!queue.isEmpty() && queue.peekLast() < nums[i]){queue.pollLast();}

if (queue.isEmpty()){queue.offer(nums[i]);}

else if (queue.peekLast() >= nums[i]){queue.offer(nums[i]);}

rev[i - k + 1] = queue.peek();

}

return rev;

}

}

## 287 Find the Duplicate Number

public class Solution {

public int findDuplicate(int[] nums) {

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

if (nums[Math.abs(nums[i])] < 0) {

return Math.abs(nums[i]);

}

else {

nums[ Math.abs(nums[i])] = -nums[ Math.abs(nums[i])];

}

}

return -1;

}

}

## 330 Patching Array

// nums = [1, 5, 10], n = 20

// Return 2.

// The two patches can be [2, 4].

// nums = [1, 3], n = 6

// Return 1. patch 2

public class Solution {

public int minPatches(int[] nums, int n) {

if (n <= 0) {return 0;}

int len = nums.length;

long maxField = 1;// pay attention to the overflow

int count = 0, i = 0;

while (maxField <= n){

if (i < len){

if (nums[i] > maxField){count++; maxField \*= 2;}

else {maxField += nums[i]; i++;}

}

else {

count++; maxField \*= 2;

}

}

return count;

}

}

## 350 Intersection of Two Arrays II

import java.util.\*;

public class Solution {

public int[] intersect(int[] nums1, int[] nums2) {

Arrays.sort (nums1);

Arrays.sort (nums2);

int i=0;

int j=0;

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

while (i < nums1.length && j < nums2.length ){

if (nums1[i] < nums2[j]){

i++;

}

else if (nums1[i] > nums2[j]){

j++;

}

else {

List.add(nums2[j]);

i++;

j++;

}

}

int [] nums = new int [List.size()];

for (int k=0 ; k < List.size(); k++){

nums[k] = List.get(k);

}

return nums;

}

}

## 357 Top K Frequent Elements

// Given [1,1,1,2,2,3] and k = 2, return [1,2].

public class Solution {

public List<Integer> topKFrequent(int[] nums, int k) {

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

int len = nums.length;

if (len == 0) {return rev;}

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

for (int i : nums){map.put(i, map.getOrDefault(i, 0) + 1);}

List<Integer> [] bucket = new List [len + 1];

for (int i : map.keySet()){

if (bucket[map.get(i)] == null){bucket[map.get(i)] = new ArrayList<>();}

bucket[map.get(i)].add(i);

}

for (int pos = len; pos >= 0; pos--){

if (k <= 0){break;}

if (bucket[pos] != null){

if (bucket[pos]. size() <= k ){

rev.addAll(bucket[pos]);

k -= bucket[pos].size();

}

else{

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

rev.add(bucket[pos].get(i));

}

}

}

}

return rev;

}

}

## 376 Wiggle Subsequence

// Input: [1,17,5,10,13,15,10,5,16,8]

// Output: 7

//RT: O(n); Space: O(1)

public class Solution {

public int wiggleMaxLength(int[] nums) {

int len = nums.length;

if (len <= 1){return len;}

int recordLow = len + 1, recordHigh = len + 1;

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

if (nums[i] > nums[0]){recordHigh = i; break;}

if (nums[i] < nums[0]){recordLow = i; break;}

}

int count = 0;

if (recordHigh == recordLow){return 1;}

else if (recordLow < recordHigh){count = 0;}

else {count = 1;}

int countLength = 1;

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

//increasing, find the maximum value

if (count % 2 != 0){

if (i == len - 1){countLength++; break;}

else {

if (nums[i] >= nums[i - 1] && nums[i] <= nums[i + 1]){continue;}

else {countLength++;count++;}

}

}

//decreasing, find the minimum value

else {

if (i == len - 1){countLength++; break;}

else {

if (nums[i] <= nums[i - 1] && nums[i] >= nums[i + 1]){continue;}

else {countLength++;count++;}

}

}

}

return countLength;

}

}

## 303 Range Sum Query - Immutable

public class NumArray {

int [] nums;

public NumArray(int[] nums) {

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

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

this.nums = nums;

}

public int sumRange(int i, int j) {

if (i > j){return 0;}

if (i <= 0){return nums[j];}

else {return nums[j] - nums[i - 1];}

}

}

/\*\*

\* Your NumArray object will be instantiated and called as such:

\* NumArray obj = new NumArray(nums);

\* int param\_1 = obj.sumRange(i,j);

\*/

## 392 Is Subsequence

//s = "abc", t = "ahbgdc" true

//s = "axc", t = "ahbgdc"

//use two pointers

public class Solution {

public boolean isSubsequence(String s, String t) {

int i=0;

if (s.length() > t.length ()){

return false;

}

for (int j = 0; j < t.length(); j++){

if (i <= s.length() -1 && t.charAt(j) == s.charAt(i)){

i++;

}

}

return (i == s.length());

}

}

## 410 Split Array Largest Sum

public class Solution {

public int splitArray(int[] nums, int m) {

int max = 0;

long sum = 0;

for (int num : nums){

max= Math.max(max, num);

sum += num;

}

long i = max, j = sum, mid = 0;

while (i < j){

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

if (help(nums, m, mid)){

j = mid;

}

else{

i = mid + 1;

}

}

return (int)j;

}

public boolean help (int [] nums, int m, long val){

int sum = 0, count = 0;

for (int num : nums){

sum += num;

if (sum > val){

sum = num;

count++;

if (count + 1 > m){return false;}

}

}

return true;

}

}

## 413 Arithmetic Slices

public class Solution {

public int numberOfArithmeticSlices(int[] A) {

int len = A.length;

if (len <= 2) {return 0;}

int [] dp = new int [len];

int sum = 0;

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

if (A[i] - A[i - 1] == A[i - 1] - A[i - 2]){

if (dp[i - 1] > 0){dp[i] = 1 + dp[i - 1];}

else {dp[i] = 1;}

sum += dp[i];

}

}

return sum;

}

}

## 442 Find All Duplicates in an Array

public class Solution {

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

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

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

int index = Math.abs(nums[i]);

if (nums[index -1] > 0) {

nums[index -1] = -nums[index -1];

}

else {

rev.add(Math.abs(nums[i]));

}

}

return rev;

}

}

## 448 Find All Numbers Disappeared in an Array

public class Solution {

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

HashSet <Integer> set0 = new HashSet <Integer> ();

for (int i : nums){

set0.add(i);

}

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

int n = nums.length;

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

if (!set0.contains(j)){

nList.add(j);

}

}

return nList;

}

}

## 453 Minimum Moves to Equal Array Elements

public class Solution {

public int minMoves(int[] nums) {

int min = nums[0];

int sum=0;

for (int i : nums){

if (i<min){

min= i;

}

}

for (int j : nums){

sum += j-min;

}

return sum;

}

}

## 454 4SumII

public class Solution {

public int fourSumCount(int[] A, int[] B, int[] C, int[] D) {

int len = A.length;

if (len == 0){return 0;}

Arrays.sort(A);

Arrays.sort(B);

Arrays.sort(C);

Arrays.sort(D);

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

int count = 0;

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

for (int j = 0; j < B.length; j++){

map.put(A[i] + B[j], map.getOrDefault(A[i] + B[j], 0) + 1);

}

}

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

for (int j = 0; j < D.length; j++){

count += map.getOrDefault(-(C[i] + D[j]), 0);

}

}

return count;

}

}

## 494 Target Sum

// Input: nums is [1, 1, 1, 1, 1], S is 3.

// Output: 5

// P is positive symbol set

// N is negative symbol set

// P - N = target => 2P = target + (P + N)

public class Solution {

public int findTargetSumWays(int[] nums, int S) {

int len = nums.length;

if (len == 0) {return 0;}

int sum = 0;

for (int i : nums){sum += i;}

if ((S + sum) % 2 != 0 || Math.abs(S) > sum){return 0;}

int val = (S + sum) / 2;

int [] dp = new int [val + 1];

dp [0] = 1;

for (int i : nums){

for (int j = val; j >= i; j--){

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

}

}

return dp[val];

}

}

## 496 [Next Greater Element I](https://leetcode.com/problems/next-greater-element-i/)

//idea 8 4 3 5

// then pop 4,3

// Input: nums1 = [4,1,2], nums2 = [1,3,4,2].

// Output: [-1,3,-1]

// RT: O(n)

public class Solution {

public int[] nextGreaterElement(int[] findNums, int[] nums) {

if (nums.length == 0 || findNums.length == 0){return new int [0];}

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

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

for (int num: nums){

while (!sk.isEmpty() && sk.peek() < num){map.put(sk.pop(),num);}

sk.push(num);

}

int [] rev = new int [findNums.length];

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

rev[i] = map.getOrDefault(findNums[i],-1);

}

return rev;

}

}

## 503 Next Greater Element II

// Input: [1,2,1]

// Output: [2,-1,2]

public class Solution {

public int[] nextGreaterElements(int[] nums) {

int len = nums.length;

if (len == 0){return new int [0];}

int [] rev = new int [len];

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

Arrays.fill (rev,-1);

for (int i = 0; i < len \* 2; i++){

while (!stack.isEmpty() && nums[stack.peek()] < nums[i % len]){

rev[stack.pop()] = nums[i % len];

}

if (i < len){stack.push(i);}

}

return rev;

}

}

## 556. Next Greater Element III

//1234

//4321

public class Solution {

public int nextGreaterElement(int n) {

String s = String.valueOf(n);

int len = s.length();

int pos = -1, change = -1;

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

for (int i = len - 2; i>= 0; i--){

rev.add(i + 1);

if (s.charAt(i) < s.charAt(i + 1)){pos = i; break;}

}

if (pos == -1) return -1;

char [] cs = s.toCharArray();

for (int i : rev){

if (cs[i] > cs[pos]){change = i; break;}

}

char temp = cs[pos];cs[pos] = cs[change]; cs[change] = temp;

Arrays.sort(cs, pos + 1, len);

Long newVal = Long.parseLong(new String(cs));

return newVal > Integer.MAX\_VALUE ? -1 : Integer.parseInt(new String(cs));

}

}

## 506 Relative Ranks

// [1,1,1,2,1]

// ["Silver Medal","Bronze Medal","4","Gold Medal","5"]

public class Solution {

public String[] findRelativeRanks(int[] nums) {

int len = nums.length;

Integer [] rev = new Integer [len];

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

rev[i] = i;

}

System.out.println(Arrays.toString(rev));

Arrays.sort (rev, (a, b) -> (nums[b] - nums[a]));

System.out.println(Arrays.toString(rev));

String [] result = new String [len];

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

if (i == 0){result[rev[i]] = "Gold Medal";}

else if (i == 1){result[rev[i]] = "Silver Medal";}

else if (i == 2){result[rev[i]] = "Bronze Medal";}

else {result[rev[i]] = String.valueOf(i + 1);}

}

return result;

}

}

## 521 Longest Uncommon Subsequence I

// Input: "aba", "cdc"

// Output: 3

public class Solution {

public int findLUSlength(String a, String b) {

if (a.equals(b)){return -1;}

else {

return Math.max(a.length(), b.length());

}

}

}

## 523 [Continuous Subarray Sum](https://leetcode.com/problems/continuous-subarray-sum)

public class Solution {

public boolean checkSubarraySum(int[] nums, int k) {

int len = nums.length;

if (len <= 1){return false;}

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

//need to consider {6,6,6} type or {0,0,1,2,3} k = 0 type

map.put(0,-1);

int sum = 0;

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

sum += nums[i];

sum = k == 0 ? sum : sum % k;

if (map.containsKey(sum)){

if (i -map.get(sum) > 1) {return true;}

}

else {

map.put(sum,i);

}

}

return false;

}

}

## 525 Contiguous Array

public class Solution {

public int findMaxLength(int[] nums) {

int max = 0;

if (nums.length <=1){

return max;

}

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

if (nums[i] == 0){

nums[i] = -1;

}

}

int sum = 0;

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

map.put(0,-1);

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

sum += nums[i];

if (map.containsKey(sum)){

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

}

else {

map.put(sum,i);

}

}

return max;

}

}

## 532 K-diff Pairs in an Array

import java.lang.Math.\*;

public class Solution {

public int findPairs(int[] nums, int k) {

if (nums.length <2 || k < 0){

return 0;

}

Arrays.sort(nums);

int i = 0;

int count = 0;

int j;

while (i < nums.length -1) {

j = i + 1;

while (j < nums.length){

if (nums[j] - nums [i] == k){

count++;

break;

}

else {

j++;

}

}

i++;

while ( (i+1) < nums.length && nums[i-1] == nums [i]){

i++;

}

}

return count;

}

}

## 560 Subarray Sum Equals K

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

// Output: 2

//sum[0,i]

//sum[0,j]

//sum[i+1,j]

//000 k = 0 answer is 6

public class Solution {

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

int len = nums.length;

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

map.put(0,1);

int cnt = 0, sum = 0;

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

sum += nums[i];

if (map.containsKey(sum - k)){

cnt += map.get(sum - k);

}

map.put(sum,map.getOrDefault(sum, 0) + 1);

}

return cnt;

}

}

## 561 [Array Partition I](https://leetcode.com/problems/array-partition-i)

public class Solution {

public int arrayPairSum(int[] nums) {

int len = nums.length;

if (len == 0){return 0;}

Arrays.sort(nums);

int sum = 0;

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

sum += i % 2 == 0 ? nums[i] : 0;

}

return sum;

}

}

## 571 Shortest Unsorted Continuous Subarray

public class Solution {

public int findUnsortedSubarray(int[] nums) {

int len = nums.length;

int [] temp = new int [len];

for (int k = 0; k < len; k++){temp[k] = nums[k];}

Arrays.sort(temp);

int i = 0 , j = len - 1;

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

if (temp[k] != nums[i]){break;}

i++;

}

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

if (temp[len - 1 - k] != nums[j]){break;}

j--;

}

return j >= i ? j - i + 1 : 0;

}

}

## 599 Minimum Index Sum of Two Lists

public class Solution {

public String[] findRestaurant(String[] list1, String[] list2) {

int len1 = list1.length, len2 = list2.length;

if (len1 == 0 || len2 == 0){return new String [0];}

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

HashMap <String, Integer> store = new HashMap<>();

int min = Integer.MAX\_VALUE;

for (int i = 0; i < len1; i++){

map.put(list1[i], i);

}

for (int i = 0; i < len2; i++){

if (map.containsKey(list2[i])){

store.put(list2[i], i + map.get(list2[i]));

min = Math.min (min, i + map.get(list2[i]));

}

}

if (min == Integer.MAX\_VALUE){return new String [0];}

else {

List<String> rev = new ArrayList<>();

for (String s : store.keySet()){

if (store.get(s) == min){rev.add(s);}

}

String [] newString = new String [rev.size()];

return rev.toArray(newString);

}

}

}

# Matrix

## 48 Rotate Image

// RT O(n^2) no extra space

public class Solution {

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

int row = matrix.length;

if (row == 0){return;}

for (int i = 0; i <= (row - 1) / 2; i++){

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

int temp = matrix[i][j];

matrix[i][j] = matrix[row - 1 - j][i];

matrix[row - 1 - j][i] = matrix[row - 1 - i][row - 1 - j];

matrix[row - 1 - i][row - 1 - j] = matrix[j][row - 1 - i];

matrix[j][row - 1 - i] = temp;

}

}

}

}

## 54 Spiral Matrix

public class Solution {

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

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

int row = matrix.length;

if (row == 0){return rev;}

int col = matrix[0].length;

if (col == 0){return rev;}

int left = 0, right = col - 1, up = 0, bottom = row - 1;

int dir = 0;

while (rev.size() < row \* col){

// go-right

if (dir == 0){

int i = left;

while (i <= right){rev.add(matrix[up][i++]);}

up++;

dir = 1;

}

// go-down

else if (dir == 1){

int i = up;

while (i <= bottom){rev.add(matrix[i++][right]);}

right--;

dir = 2;

}

// go-left

else if (dir == 2){

int i = right;

while (i >= left){rev.add(matrix[bottom][i--]);}

bottom--;

dir = 3;

}

// go-up

else {

int i = bottom;

while (i >= up){rev.add(matrix[i--][left]);}

left++;

dir = 0;

}

}

return rev;

}

}

## 59 [Spiral Matrix II](https://leetcode.com/problems/spiral-matrix-ii/)

public class Solution {

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

// if (n == 0){return new int [0][0];}

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

int col = n, row = n, count = 0, dir = 0;

int left = 0, right = col - 1, up = 0, bottom = row - 1;

while (count < n \* n){

// go-right

if (dir == 0){

int i = left;

while (i <= right){matrix[up][i++] = ++count;}

up++;

dir = 1;

}

// go-down

else if (dir == 1){

int i = up;

while (i <= bottom){matrix[i++][right] = ++count;}

right--;

dir = 2;

}

// go-left

else if (dir == 2){

int i = right;

while (i >= left){matrix[bottom][i--] = ++count;}

bottom--;

dir = 3;

}

// go-up

else {

int i = bottom;

while (i >= up){matrix[i--][left] = ++count;}

left++;

dir = 0;

}

}

return matrix;

}

}

## 63 [Unique Paths II](https://leetcode.com/problems/unique-paths-ii/)

public class Solution {

public int uniquePathsWithObstacles(int[][] obstacleGrid) {

int row = obstacleGrid.length;

if (row == 0){return 0;}

int col = obstacleGrid[0].length;

if (col == 0){return 0;}

// System.out.println(row);

// System.out.println(col);

int [][] grid = new int [row][col];

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

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

int right = j == col - 1 ? 0 : grid[i][j + 1];

int down = i == row - 1 ? 0: grid[i + 1][j];

if (obstacleGrid[i][j] == 1) {

grid[i][j] = 0;

}

else if (i == row - 1 && j == col - 1){grid[i][j] = 1;}

else {grid[i][j] = right + down;}

}

}

return grid[0][0];

}

}

## 64 Minimum Path Sum

public class Solution {

public int minPathSum(int[][] grid) {

int row = grid.length;

if (row == 0){return -1;}

int col = grid[0].length;

if (col == 0) {return -1;}

int [][] dp = new int [row][col];

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

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

int right = j == 0 ? Integer.MAX\_VALUE : dp[i][j - 1];

int down = i == 0 ? Integer.MAX\_VALUE : dp[i - 1][j];

dp[i][j] = i == 0 && j == 0 ? grid[i][j] : Math.min(right, down) + grid[i][j];

//System.out.println("right " + right + " down " + down + " " + dp[i][j] );

}

}

return dp[row - 1][col - 1];

}

}

## 73 [Set Matrix Zeroes](https://leetcode.com/problems/set-matrix-zeroes/)

public class Solution {

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

int row = matrix.length;

if (row == 0) {return;}

int col = matrix[0].length;

if (col == 0){return;}

boolean r = false;

boolean c = false;

for (int j = 0; j < col; j++){if (matrix[0][j] == 0){r = true; break;}}

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

if (matrix[i][0] == 0){c = true; break;}

}

for (int i = row - 1; i >= 1; i--){

for (int j = col - 1; j >= 1; j--){

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

matrix[0][j] = 0;

matrix[i][0] = 0;

}

}

}

for (int j = col - 1; j >= 1; j--){

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

int i = 0;

while (i < row){matrix[i++][j] = 0;}

}

}

for (int i = row - 1; i >= 1; i--){

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

int j = 0;

while (j < col){matrix[i][j++] = 0;}

}

}

if (r){int j = 0; while (j < col){matrix[0][j++] = 0;}}

if (c){int i = 0; while (i < row){matrix[i++][0] = 0;}}

}

}

## 74 Search a 2D Matrix

// consider [[]] and [] conditions

public class Solution {

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

int row = matrix.length;

if (row == 0){return false;}

int col = matrix[0].length;

if (col == 0){return false;}

int i = 0; int j = row - 1;

while (i <= j){

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

if (target < matrix[mid][0]){j = mid - 1;}

else if (target > matrix[mid][col - 1]){i = mid + 1;}

else {

return serchOneRow (matrix[mid], target);

}

}

return false;

}

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

int len = nums.length;

if (len == 0){return false;}

int i = 0, j = len - 1;

while (i <= j){

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

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

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

else {i = mid + 1;}

}

return false;

}

}

## 79. Word Search

public class Solution {

public boolean exist(char[][] board, String word) {

int row = board.length;

if (row == 0) return false;

int col = board[0].length;

if (col == 0) return false;

if (word.length() > row \* col) return false;

int [][] visited = new int [row][col];

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

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

if (dfs(board, word, i, j, 0)) return true;

}

}

return false;

}

public boolean dfs (char[][] board, String word, int i, int j, int k){

int row = board.length;

int col = board[0].length;

if (i < 0 || i >= row || j < 0 || j >= col || board[i][j] != word.charAt(k)){

return false;

}

else if (k == word.length() - 1) return true;

else {

char c = board[i][j];

board[i][j] = '\*';

if (dfs(board,word,i+1,j,k+1)||dfs(board,word,i-1,j,k+1)||dfs(board,word,i,j+1,k+1)||dfs(board,word,i,j-1,k+1)){

return true;}

board[i][j]= c;return false;

}

}

}

## 85 Maximal Rectangle

public class Solution {

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

int row = matrix.length;

if (row == 0){return 0;}

int col = matrix[0].length;

if (col == 0){return 0;}

int [] nums = new int [col];

int maxRect = Integer.MIN\_VALUE;

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

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

nums[j] = matrix[i][j] == '0' ? 0 : nums[j] + matrix[i][j] - '0';

}

maxRect = Math.max(maxRect, largestRectangleArea(nums));

}

for (int i : nums){System.out.println(i);}

return maxRect;

}

public int largestRectangleArea(int[] heights) {

int len = heights.length;

if (len == 0) {return 0;}

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

int maxArea = Integer.MIN\_VALUE;

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

if (i < len){

if (stack.isEmpty() || heights[stack.peek()] <= heights[i]){stack.push(i);}

else {

while (!stack.isEmpty()){

int top = stack.pop();

if (stack.isEmpty()){maxArea = Math.max(maxArea, heights[top] \* i);}

else {

maxArea = Math.max(maxArea, heights[top] \* (i - stack.peek() - 1));

if (heights[stack.peek()] <= heights[i]){break;}

}

}

stack.push(i);

}

}

else {

while (!stack.isEmpty()){

int top = stack.pop();

if (stack.isEmpty()){maxArea = Math.max(maxArea, heights[top] \* i);}

else {

maxArea = Math.max(maxArea, heights[top] \* (i - stack.peek() - 1));

}

}

}

}

return maxArea;

}

}

## 118 [Pascal's Triangle](https://leetcode.com/problems/pascals-triangle)

public class Solution {

public List<List<Integer>> generate(int numRows) {

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

List <Integer> sub0 = new ArrayList<Integer> ();

if (numRows == 0){return rev;}

sub0.add(1); rev.add(sub0);

if (numRows == 1){return rev;}

for (int j = 2; j <= numRows; j++){

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

int lenRev = rev.size() - 1;

for (int i = 0; i < rev.get(lenRev).size(); i++){

if (i == 0){sub.add(rev.get(lenRev).get(i));}

else {sub.add(rev.get(lenRev).get(i) + rev.get(lenRev).get(i - 1));}

}

sub.add(1);

rev.add(sub);

}

return rev;

}

}

## 119 [Pascal's Triangle](https://leetcode.com/problems/pascals-triangle) II

public class Solution {

public List<Integer> getRow(int rowIndex) {

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

List <Integer> sub0 = new ArrayList<Integer> ();

sub0.add(1); rev.add(sub0);

if (rowIndex == 0){return rev.get(0);}

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

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

int lenRev = rev.size() - 1;

for (int i = 0; i < rev.get(lenRev).size(); i++){

if (i == 0){sub.add(rev.get(lenRev).get(i));}

else {sub.add(rev.get(lenRev).get(i) + rev.get(lenRev).get(i - 1));}

}

sub.add(1);

rev.add(sub);

}

return rev.get(rowIndex);

}

}

## 130 Surrounded Regions

public class Solution {

public void solve(char[][] board) {

int row = board.length;

if (row == 0){return;}

int col = board[0].length;

if (col == 0){return;}

boolean [][] dp = new boolean [row][col];

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

if (board[0][j] == 'O' && dp[0][j] == false){

bfs(board, dp, 0, j);

}

if (board[row - 1][j] == 'O' && dp[row - 1][j] == false){

bfs(board, dp, row - 1, j);

}

}

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

if (board[i][0] == 'O' && dp[i][0] == false){

bfs(board, dp, i, 0);

}

if (board[i][col - 1] == 'O' && dp[i][col - 1] == false){

bfs(board, dp, i, col - 1);

}

}

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

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

if (board[i][j] == 'O' && dp [i][j] == false){

board[i][j] = 'X';

}

}

}

}

public void bfs (char [][] board, boolean [][] dp, int m, int n){

int row = board.length;

int col = board[0].length;

Queue<int []> queue = new LinkedList<>();

int [][] dir = {{-1,0},{1,0},{0,1},{0,-1}};

queue.offer(new int [] {m,n});

dp[m][n] = true;

while (!queue.isEmpty()){

int [] nums = queue.poll();

for (int [] t : dir){

int lr = t[0] + nums[0];

int ud = t[1] + nums[1];

if (lr >=0 && lr < row && ud >=0 && ud < col && board[lr][ud] == 'O' && dp[lr][ud] == false){

queue.offer(new int []{lr,ud});

dp[lr][ud] = true;

}

}

}

}

}

## 200 Number of Islands

public class Solution {

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

Queue <int []> largeQueue = new LinkedList<>();

Queue <int []> smallQueue = new LinkedList<>();

int row = grid.length;

if (row == 0) {return 0;}

int col = grid[0].length;

if (col == 0){return 0;}

int [][] dirs = {{1, 0},{-1, 0},{0, 1},{0, -1}};

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

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

if (grid[i][j] == '1'){largeQueue.offer(new int [] {i,j});}

}

}

int count = 0;

while (!largeQueue.isEmpty()){

int [] nums = largeQueue.poll();

if (grid[nums[0]][nums[1]] == '0'){continue;}

smallQueue.offer(nums);

grid[nums[0]][nums[1]] = '0';

//bfs

while (!smallQueue.isEmpty()){

int [] newNums = smallQueue.poll();

for (int [] t : dirs){

int m = newNums[0] + t[0];

int n = newNums[1] + t[1];

if (m < 0 || m >= row || n < 0 || n >= col || grid[m][n] == '0'){continue;}

else {

smallQueue.offer(new int [] {m, n});

grid[m][n] = '0';

}

}

}

count++;

}

return count;

}

}

## 221 Maximal Square

//idea dp[i][j] is the maxLength can active by using position[i][j]

// dp[i][j] = matrix[i][j] == '0' ? 0: Math.min(Math.min(dp[i - 1][j], dp[i][j - 1]), dp[i - 1][j - 1]) + 1;

public class Solution {

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

int row = matrix.length;

if (row == 0){return 0;}

int col = matrix[0].length;

if (col == 0) {return 0;}

int maxLength = Integer.MIN\_VALUE;

int [][] dp = new int [row][col];

//initialize

for (int i = 0; i < row; i++){dp[i][0] = matrix[i][0] - '0'; maxLength = Math.max(maxLength, dp[i][0]);}

for (int i = 0; i < col; i++){dp[0][i] = matrix[0][i] - '0'; maxLength = Math.max(maxLength, dp[0][i]);}

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

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

dp[i][j] = matrix[i][j] == '0' ? 0 : Math.min(Math.min(dp[i - 1][j], dp[i][j - 1]), dp[i - 1][j - 1]) + 1;

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

}

}

return maxLength \* maxLength;

}

}

## 279 [Perfect Squares](https://leetcode.com/problems/perfect-squares)

//dp solution

//dp[n] = min(dp[n - i \* i] + 1)

// n - i \* i >=0 && i >=1

public class Solution {

public int numSquares(int n) {

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

for (int i = 1; i < n + 1; i++){dp[i] = Integer.MAX\_VALUE;}

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

int i = 1;

while (j - i \* i >= 0){

dp[j] = Math.min(dp[j - i \* i] + 1,dp[j]);

i++;

}

}

return dp[n];

}

}

//running time O(n^1.5)

## 367 [Valid Perfect Square](https://leetcode.com/problems/valid-perfect-square)

public class Solution {

public boolean isPerfectSquare(int num) {

if (num == 0){return true;}

if (num == 1){return true;}

long val = (long) num;

long i = 0, j = val/2;

while (i <= j){

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

System.out.println(mid);

long temp = mid \* mid;

if(temp == val){return true;}

else if (temp < val) {i = mid + 1;}

else {j = mid - 1;}

}

return false;

}

}

## 391 Perfect Rectangle

// rectangles = [

// [1,1,3,3],

// [3,1,4,2],

// [3,2,4,4],

// [1,3,2,4],

// [2,3,3,4]

// ]

//two condition: 1: total sum of small rectangle = large rectangle

// 2: total four corners used in large rectangle is contained

// need consider total sum = 0;

public class Solution {

public boolean isRectangleCover(int[][] rectangles) {

int row = rectangles.length;

if (row == 0){return true;}

int col = rectangles[0].length;

if(col == 0){return true;}

int leftx = Integer.MAX\_VALUE, lefty = Integer.MAX\_VALUE;

int upx = Integer.MIN\_VALUE, upy = Integer.MIN\_VALUE;

int sum = 0;

HashSet<String> set = new HashSet<>();

for (int [] nums : rectangles){

sum += (nums[2] - nums[0]) \* (nums[3] - nums[1]);

if (nums[0] <= leftx && nums[1] <= lefty){leftx = nums[0]; lefty = nums[1];}

if (nums[2] >= upx && nums[3] >= upy){upx = nums[2]; upy = nums[3];}

String ld = nums[0] + "" + nums[1];

String lu = nums[0] + "" + nums[3];

String rd = nums[2] + "" + nums[1];

String ru = nums[2] + "" + nums[3];

String [] strs = {ld, lu, rd, ru};

for (String s : strs){

if (set.contains(s)){set.remove(s);}

else {set.add(s);}

}

System.out.println(set.size());

}

String ld = leftx + "" + lefty;

String lu = leftx + "" + upy;

String rd = upx + "" + lefty;

String ru = upx + "" + upy;

String [] strs = {ld, lu, rd, ru};

boolean flag = false;

for (String s : strs){ flag = flag || !set.contains (s);}

if(sum == 0){return true;}

else if (set.size() != 4) {return false;}

else if (flag){return false;}

else {return sum == (upx - leftx)\* (upy - lefty);}

}

}

## 419 [Battleships in a Board](https://leetcode.com/problems/battleships-in-a-board/)

public class Solution {

public int countBattleships(char[][] board) {

int row = board.length;

if (row == 0){return 0;}

int col = board[0].length;

if (col == 0){return 0;}

int count = 0;

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

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

if (board[i][j] == '.'){continue;}

if (i > 0 && board[i - 1][j] == 'X'){continue;}

if (j > 0 && board[i][j - 1] == 'X'){continue;}

count++;

}

}

return count;

}

}

## 474 Ones and Zeros

//RT: len\*(mn) space O(mn)

public class Solution {

public int findMaxForm(String[] strs, int m, int n) {

int len = strs.length;

if (len == 0) {return 0;}

int [][] dp = new int [m + 1][n + 1];

dp [0][0] = 0;

for (String s : strs){

int [] count = counting (s);

for (int i = m; i >= count[0]; i--){

for (int j = n; j >= count[1];j--){

dp[i][j] = Math.max(dp[i][j], dp[i - count[0]][j - count[1]] + 1);

}

}

}

return dp[m][n];

}

public int [] counting (String s){

int [] count = new int [2];

if (s.length() == 0){return count;}

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

if (s.charAt(i) == '0'){count[0]++;}

else {count[1]++;};

}

return count;

}

}

## 498 [Diagonal Traverse](https://leetcode.com/problems/diagonal-traverse)

public class Solution {

public int[] findDiagonalOrder(int[][] matrix) {

int row = matrix.length;

if (row == 0) {return new int [0];}

int col = matrix[0].length;

int [] nums = new int [row \* col];

int i = 0, j = 0;

for (int k = 0; k < row \* col; k++){

//System.out.println ("i is " + i + " j is " + j);

//System.out.println ("k is " + k);

nums[k] = matrix[i][j];

if ((i + j) % 2 == 0){ // up

if (i == 0 && j == col -1){i++;}

else if (i == 0){j++;}

else if (j == col -1){i++;}

else {i--;j++;}

}

else{ // down

// System.out.println ("i is " + i);

// System.out.println ("j is " + j);

if (j == 0 && i == row - 1){j++;}

else if (j == 0){i++;}

else if (i == row - 1) {j++;}

else {i++;j--;}

}

}

return nums;

}

}

## 542 [01 Matrix](https://leetcode.com/problems/01-matrix/)

public class Solution {

public int[][] updateMatrix(int[][] matrix) {

int m = matrix.length;

int n = matrix[0].length;

Queue<int[]> queue = new LinkedList<>();

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

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

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

queue.offer(new int[] {i, j});

}

else {

matrix[i][j] = Integer.MAX\_VALUE;

}

}

}

int[][] dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

while (!queue.isEmpty()) {

int[] cell = queue.poll();

for (int[] d : dirs) {

int r = cell[0] + d[0];

int c = cell[1] + d[1];

if (r < 0 || r >= m || c < 0 || c >= n ||

matrix[r][c] <= matrix[cell[0]][cell[1]] + 1) continue;

queue.add(new int[] {r, c});

matrix[r][c] = matrix[cell[0]][cell[1]] + 1;

}

}

return matrix;

}

}

## 566 Reshape the Matrix

// [[1,2,3],[4,5,6]] 3 2

//[[1,2],[3,4],[5,6]]

public class Solution {

public int[][] matrixReshape(int[][] nums, int r, int c) {

if (nums.length == 0){

return nums;

}

int row = nums.length;

int col = nums[0].length;

if (row \* col != r \* c ){

return nums;

}

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

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

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

rev.add(nums[i][j]);

}

}

int [] [] newNums = new int [r][c];

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

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

newNums[i][j] = rev.get(c \* i + j);

}

}

return newNums;

}

}

//runing time Analysis: O(rc)

## 598 Range AdditionII

public class Solution {

public int maxCount(int m, int n, int[][] ops) {

int row = ops.length;

if (row == 0) {return m \* n;}

int col = ops[0].length;

if (col == 0) {return m \* n;}

long n1 = m;

long n2 = n;

for (int [] nums : ops){

n1 = Math.min(n1, nums[0]);

n2 = Math.min(n2, nums[1]);

}

return (int)(n1 \* n2);

}

}

# String

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

public class Solution {

public int lengthOfLongestSubstring(String s) {

if (s.length() <= 1){

return s.length();

}

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

int j = 0, max = 0;

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

if (map.containsKey(s.charAt(i))){

j = Math.max(j,map.get(s.charAt(i)) + 1);

}

map.put(s.charAt(i),i);

max = Math.max(max, i - j + 1);

}

return max;

}

}

## 5 Longest Palindromic Substring

public class Solution {

public String longestPalindrome(String s) {

if (s.length() <= 1){

return s;

}

int len = s.length();

boolean [][] dp = new boolean [len][len];

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

int i = 0;

int j = k;

while (i < len -k){

if (j - i == 0){

dp[i][j] = true;

}

if (j - i == 1){

dp[i][j] = s.charAt(j) == s.charAt(i);

}

if (j - i > 1){

dp[i][j] = (s.charAt(j) == s.charAt(i)) && dp [i+1][j-1];

}

i++;j++;

}

}

int max = 0;

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

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

if (dp[i][j] == true && j >=i){

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

}

}

}

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

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

if (dp[i][j] == true && j - i == max){

return s.substring(i,j+1);

}

}

}

return s;

}

}

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

public class Solution {

public int strStr(String haystack, String needle) {

if (haystack.contains(needle)){

return haystack.indexOf(needle);

}

else{

return -1;

}

}

}

## 46 Permutations

public class Solution {

HashSet<List<Integer>> set = new HashSet<>();

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

if (nums.length == 0){return new ArrayList<List<Integer>>();}

helpFun (nums, new ArrayList<Integer>(), new int [nums.length]);

return new ArrayList<List<Integer>>(set);

}

public void helpFun (int [] nums, List<Integer> sub, int [] used){

if (sub.size() == nums.length){set.add(sub); return;}

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

List<Integer> nsub = new ArrayList <>(sub);

if (used[i] == 0){

used[i] = 1;

nsub.add(nums[i]);

helpFun (nums, nsub, used);

used[i] = 0;

}

}

}

}

## 49. Group Anagrams

// RT and Spcae : O(n)

public class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

List<List<String>> rev = new ArrayList<>();

int count = 0;

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

for (String s : strs){

String temp = help(s);

if (!map.containsKey(temp)){

map.put(temp, count++);

List<String> sub = new ArrayList<>();

sub.add(s);rev.add(sub);

}

else{

int pos = map.get(temp);

rev.get(pos).add(s);

}

}

return rev;

}

public String help (String s){

char [] cs = new char [26];

Arrays.fill(cs,'0');

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

cs[s.charAt(i) - 'a']++;

}

return new String(cs);

}

}

## 58 Length of Last Word

public class Solution {

public int lengthOfLastWord(String s) {

if (s.length() == 0){

return 0;

}

int cnt = 0;

for (int i = s.length() -1 ; i>=0 ; i--){

if (s.charAt(i) != ' '){

cnt++;

}

else if (cnt == 0){

continue;

}

else {

break;

}

}

return cnt;

}

}

## 125 Valid Palindrome

import java.util.\*;

public class Solution {

public boolean isPalindrome(String s) {

s = s.toLowerCase();

if (s.length() == 0){

return true;

}

int i = 0;

int j = s.length() -1;

while (i <= j){

if (!Character.isLetterOrDigit(s.charAt(i))){

i++;

}

else if (!Character.isLetterOrDigit(s.charAt(j))){

j--;

}

else {

if (s.charAt(i) !=s.charAt(j)){

return false;

}

else {

i++;

j--;

}

}

}

return true;

}

}

## 131 Palindrome Partitioning

public class Solution {

List<List<String>> rev = new ArrayList<>();

public List<List<String>> partition(String s) {

int len = s.length();

if (len == 0) {return rev;}

help (new ArrayList<String>(), s, 0);

return rev;

}

public void help (List<String> sub, String s, int pos){

if (pos > s.length() - 1){rev.add(sub);return;}

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

if (isPalindrome(s, pos, i)){

List<String> nsub = new ArrayList<>(sub);

nsub.add(s.substring(pos, i + 1));

help (nsub, s, i + 1);

}

}

}

public boolean isPalindrome (String s, int i, int j){

if (s.length() == 0){return true;}

while (i < j){

if (s.charAt(i++) != s.charAt(j--)){return false;}

}

return true;

}

}

## 151 Reverse Words in a String

public class Solution {

public String reverseWords(String s) {

int len = s.length();

if (len == 0){return s;}

String [] strs = s.split("\\s+");

StringBuilder sb = new StringBuilder();

for (int i = strs.length - 1; i >= 0; i--){

sb.append(strs[i]);sb.append(" ");

}

return sb.toString().trim();

}

}

## 205 [Isomorphic Strings](https://leetcode.com/problems/isomorphic-strings)

//Given "paper", "title", return true.

public class Solution {

public boolean isIsomorphic(String s, String t) {

if (s.length() != t.length()){

return false;

}

if (s.length () <= 1){

return true;

}

HashMap <Character,Character> map = new HashMap<>();

map.put(s.charAt(0),t.charAt(0));

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

if (map.containsKey(s.charAt(i))){

if (map.get(s.charAt(i)) != t.charAt(i)){

return false;

}

}

else {

if (map.containsValue(t.charAt(i))){

return false;

}

map.put(s.charAt(i),t.charAt(i));

}

}

return true;

}

}

## 214 [Shortest Palindrome](https://leetcode.com/problems/shortest-palindrome/)

// Given "aacecaaa", return "aaacecaaa".

// Given "abcd", return dcbabcd

// Attention stringBuilder inser. O(n)

// Attention stringBuilder append. O(1)

public class Solution {

public String shortestPalindrome(String s) {

if (s.length() <= 1){

return s;

}

int len = s.length();

int i = 0;

int j= len - 1;

StringBuilder sb = new StringBuilder();

while (j >= 0){

if (s.charAt(i) == s.charAt(j) && isPalindrome(s, i, j)){

for (int k = len - 1; k > j; k--){

sb.append(s.charAt(k));

}

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

sb.append(s.charAt(k));

}

break;

}

else{j--;}

}

return sb.toString();

}

public boolean isPalindrome (String s, int i, int j) {

while (i < j){

if (s.charAt(i) != s.charAt(j)){

return false;

}

else {i++; j--;}

}

return true;

}

}

## 242 Valid Anagram

public class Solution {

public boolean isAnagram(String s, String t) {

if (s.length() != t.length ()){

return false;

}

int [] nums = new int [26];

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

nums[t.charAt(i) - 'a']++;

}

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

nums[s.charAt(i) - 'a']--;

}

boolean b = true;

for (int i : nums){

b = b && (i==0);

}

return b;

}

}

## 278 First Bad Version

/\* The isBadVersion API is defined in the parent class VersionControl.

boolean isBadVersion(int version); \*/

public class Solution extends VersionControl {

public int firstBadVersion(int n) {

int low = 1;

int high = n;

int min = n;

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

while (low <= high){

if (isBadVersion (mid) == true){

min = mid;

high = mid - 1;

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

}

else {

low = mid + 1;

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

}

}

return min;

}

}

## 300 Longest Increasing Subsequence

//Given [10, 9, 2, 5, 3, 7, 101, 18],

//The longest increasing subsequence is [2, 3, 7, 101], therefore the length is 4.

public class Solution {

public int lengthOfLIS(int[] nums) {

if (nums.length <= 1){

return nums.length;

}

int len = nums.length;

int [] rev = new int [len];

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

rev[i] = 1;

}

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

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

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

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

}

}

}

int max = 0;

for (int i : rev){

max = Math.max(max,i);

}

return max;

}

}

## 318 Maximum Product of Word Lengths

//compare every two word

// check the bit

//RT: O(n \* n + O(word.length))

//Attention (checkBit[i] & checkBit[j]) == 0 "()""

// max value initial 0 is better, if use min\_VALUE, may return min\_value in some cases

public class Solution {

public int maxProduct(String[] words) {

int len = words.length;

if (len <= 1){return 0;}

int [] checkBit = new int [len];

int max = 0;

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

for (int j = 0; j < words[i].length(); j++){

checkBit[i] |= 1 << (words[i].charAt(j) - 'a');

}

}

for (int i = 0; i < len - 1; i++){

for (int j = i + 1; j < len; j++){

if ((checkBit[i] & checkBit[j]) == 0){

max = Math.max(words[i].length() \* words[j].length(), max);

}

}

}

return max;

}

}

## 345 Reverse Vowels of a String

public class Solution {

public String reverseVowels(String s) {

int len = s.length();

List<Character> rev = new ArrayList<>(Arrays.asList('a','e','u','e','o','i','A','E','U','E','O','I'));

//string can not set value at by index; but ArrayList can use .set(index,value);

if (len <= 1) {

return s;

}

int i = 0, j = len - 1;

char [] chars = s.toCharArray();

while (i < j) {

// left vowel

while (i <= len-1) {

if (rev.contains(chars[i])){

break;

}

i++;

}

//right vowel

while (j >= 0) {

if (rev.contains(chars[j])){

break;

}

j--;

}

if (i < j){

// swap left and right vowel

char temp = chars[i];

chars[i] = chars[j];

chars[j] = temp;

}

i++;

j--;

}

return new String (chars);

}

}

// O(len), no extra space;

## 387 First Unique Character in a String

import java.util.\*;

public class Solution {

public int firstUniqChar(String s) {

int [] nums = new int [26];

char [] c = s.toCharArray ();

for (int j = 0 ; j < c.length ; j++) {

nums[c[j] - 'a']++;

}

for (int j = 0 ; j < c.length ; j++) {

if (nums[c[j] - 'a'] == 1)

return j ;

}

return -1;

}

}

## 394 Decode String

// s = "3[a2[c]]", return "accaccacc".

// "3[a10[bc]]"

public class Solution {

public String decodeString(String s) {

int len = s.length();

if (len == 0){return s;}

Stack <String> stack1 = new Stack<>();

Stack <Integer> stack2 = new Stack <>();

int i = 0; String rev = "";

while (i < s.length()){

if (Character.isLetter(s.charAt(i))){

rev += s.charAt(i++);

}

else if (Character.isDigit(s.charAt(i))){

int count = 0;

while(Character.isDigit(s.charAt(i))){

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

}

stack2.push(count);

}

else if (s.charAt(i) == '['){

stack1.push(rev);i++;

rev = "";

}

else {

StringBuilder sb = new StringBuilder(stack1.pop());

int repeat = stack2.pop();

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

sb.append(rev);

}

rev = sb.toString();

i++;

}

}

return rev;

}

}

## 395 [Longest Substring with At Least K Repeating Characters](https://leetcode.com/problems/longest-substring-with-at-least-k-repeating-characters/)

//Worst case RT O(n^2)

public class Solution {

public int longestSubstring(String s, int k) {

if (k <= 0 || s.length() < k) {return 0;}

return help (s, k, 0, s.length());

}

public int help (String s, int k, int pos1, int pos2){

if (pos1 >= pos2){return 0;}

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

List<Integer> [] cs = new List [26];

int min = Integer.MAX\_VALUE, pos = 0;

char record = 'A';

for (int i = pos1; i < pos2; i++){

char c = s.charAt(i);

map.put(c, map.getOrDefault(c, 0) + 1);

}

for (char c : map.keySet()){

if (min >= map.get(c)){

min = map.get(c); record = c;

}

}

for (int i = pos1; i < pos2; i++){

if (s.charAt(i) == record){pos = i; break;}

}

//System.out.println("min " + min);

if (min >= k){return pos2 - pos1;}

else {return Math.max(help(s, k, pos1, pos),help(s, k, pos + 1, pos2));}

}

}

## 402 Remove K Digits

// Input: num = "10200", k = 1

// Output: "200"

// Input: num = "1432219", k = 3

// Output: "1219"

//1123 3

public class Solution {

public String removeKdigits(String num, int k) {

if (k <= 0){return num;}

int sum = 0, len = num.length();

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

if (num.charAt(i) != '0'){sum++;}

}

if (len <= k || sum <= k){return "0";}

Stack<Character> stack = new Stack<>();

//13378 2

int j = 0;

while (j < len){

if (k > 0){

if (stack.isEmpty()){stack.push(num.charAt(j++));}

else if (stack.peek() > num.charAt(j)){stack.pop();k--;}

else {stack.push(num.charAt(j++));}

}

else {stack.push(num.charAt(j++));}

}

while (k > 0){stack.pop();k--;}

StringBuilder sb = new StringBuilder();

while (!stack.isEmpty()){sb.append(stack.pop());}

sb = sb.reverse();

int record = -1;

for (int i = 0; i < sb.length();i++){if (sb.charAt(i) != '0') {record = i; break;}}

if (record == -1){return "0";}

else {return sb.substring(record).toString();}

}

}

## 409 Longest Palindrome

public class Solution {

public int longestPalindrome(String s) {

int [] nums = new int [52];

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

char c = s.charAt(i);

if (c >= 'a' && c <= 'z'){

nums[c - 'a']++;

}

else {

nums[26+c - 'A']++;

}

}

int oddNo = 0;

int sum = 0;

for (int i : nums){

sum += i;

if (i%2 !=0){

oddNo++;

}

}

if (oddNo>0) {

oddNo--;

}

return (sum - oddNo);

}

}

## 415 [Add Strings](https://leetcode.com/problems/add-strings)

public class Solution {

public String addStrings(String num1, String num2) {

int len1 = num1.length() - 1;

int len2 = num2.length() - 1;

int carry = 0;

StringBuilder s = new StringBuilder ();

while (len1 >= 0 || len2 >= 0){

int f1 = len1 < 0 ? 0: num1.charAt(len1) - '0';

int f2 = len2 < 0 ? 0: num2.charAt(len2) - '0';

int digit = f1 + f2 + carry;

carry = digit > 9 ? 1 : 0;

digit %= 10;

len1--;len2--;

s.append(digit);

}

if (carry == 1){s.append(carry);}

return s.reverse().toString();

}

}

## 434 Number of Segments in a String

public class Solution {

public int countSegments(String s) {

if (s.length() == 0){

return 0;

}

String newS = s.toLowerCase();

int count = 0;

boolean prev = false;

boolean cur = false;

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

char c = newS.charAt(i);

cur = (c != ' ');

if (prev == false && cur == true){

count++;

}

prev = cur;

}

return count;

}

}

## 438 Find All Anagrams in a String

public class Solution {

public List<Integer> findAnagrams(String s, String p) {

List <Integer> List0 = new ArrayList<Integer> ();

int lenP = p.length();

int lenS = s.length();

for (int i =0; i< lenS+1-lenP; i++){

String sub = s.substring (i,i+lenP);

if (isAnagrams (sub,p)){

List0.add(i);

}

}

return List0;

}

public boolean isAnagrams (String s, String p){

int [] nums = new int [26];

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

nums[s.charAt(i) - 'a']++;

}

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

nums[p.charAt(i) - 'a']--;

}

boolean b = true;

for (int i : nums){

b = b && (i ==0);

}

return b;

}

}

## 451 Sort Characters By Frequency

// RT O(n) Space O(n)

public class Solution {

public String frequencySort(String s) {

int len = s.length();

if (len == 0){return s;}

char [] cs = s.toCharArray();

List<Character> [] bucket = new List [len + 1];

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

// find the frequecy of each char

for (char c : cs){map.put(c, map.getOrDefault(c, 0) + 1);}

for (char c : map.keySet()){

if (bucket[map.get(c)] == null){bucket[map.get(c)] = new ArrayList<Character>();}

bucket[map.get(c)].add(c);

}

int i = 0;

for (int pos = len; pos >= 0; pos--){

if (bucket[pos] != null){

for (int j = 0; j < bucket[pos].size(); j++){

int freq = pos;

while (freq-- > 0){cs[i++] = bucket[pos].get(j);}

}

}

}

return new String(cs);

}

}

## 459 Repeated Substring Pattern

public class Solution {

public boolean repeatedSubstringPattern(String s) {

if (s.length() == 0){

return true;

}

String str = s + s;

return str.substring(1,str.length()-1).contains(s);

}

}

## 467 Unique Substrings in Wraparound String

// find the longest continus substring

// find the number end with each characters

public class Solution {

public int findSubstringInWraproundString(String p) {

int len = p.length();

if (len == 0) {return 0;}

int [] dp = new int [26];

int max = 0;

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

if (i > 0 && (p.charAt(i) - p.charAt(i - 1) == 1 || p.charAt(i - 1) - p.charAt(i) == 25)){

max++;

}

else {max = 1;}

dp[p.charAt(i) - 'a'] = Math.max(dp[p.charAt(i) - 'a'], max);

}

int sum = 0;

for (int i = 0; i < 26; i++){

sum += dp[i];

}

return sum;

}

}

## 491 Increasing Subsequences

//should consider the duplicate case

public class Solution {

HashSet<List<Integer>> rev = new HashSet<>();

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

if (nums.length < 2){return new ArrayList<List<Integer>> ();}

help (new ArrayList<Integer> (), nums, 0);

return new ArrayList<List<Integer>>(rev);

}

public void help (List<Integer> sub, int []nums, int pos){

if (sub.size() > 1){rev.add(sub);}

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

List<Integer> nsub = new ArrayList<>(sub);

if (nsub.size() == 0 || nums[i] >= sub.get(nsub.size() - 1)){

nsub.add(nums[i]);

}

help (nsub, nums, i + 1);

}

}

}

## 524 Longest Word in Dictionary through Deleting

// Input:

// s = "abpcplea", d = ["ale","apple","monkey","plea"]

// Output:

// "apple"

public class Solution {

public String findLongestWord(String s, List<String> d) {

List<String> rev = new ArrayList<>();

int lens = s.length();

int lend = d.size();

int max = 0;

if (lend == 0 || lens == 0){return "";}

Collections.sort(d, (a,b) -> b.length() - a.length());

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

if (lens < d.get(i).length()){break;}

else {

if (isContains(s,d.get(i)) && d.get(i).length() >= max){

rev.add(d.get(i));

max = d.get(i).length();}

}

}

if (rev.size()>= 1) {Collections.sort(rev);return rev.get(0);}

else {return "";}

}

public boolean isContains(String s, String t){

int j = 0;

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

if (s.charAt(i) == t.charAt(j)){

j++;

if (j == t.length()){

return true;

}

}

}

return false;

}

}

## 541 Reverse String II

import java.lang.Math.\*;

public class Solution {

public String reverseStr(String s, int k) {

char [] c = s.toCharArray();

if (s.length () == 0 || k <= 0) {

return s;

}

int i = 0;

int j = 0 + 2\*k-1;

while (j < s.length()){

int low = i;

int high = (i+k-1);

while (low <= high){

char temp = c[low];

c[low] = c [high];

c [high] = temp;

low++;

high--;

}

i = j+1;

j += 2\*k;

}

if (i < s.length()){

int low = i;

int high = Math.min (i+k-1,s.length()-1);

while (low <= high){

char temp = c[low];

c[low] = c [high];

c [high] = temp;

low++;

high--;

}

}

return new String (c);

}

}

## 557 Reverse Words in a String III

import java.util.\*;

public class Solution {

public String reverseWords(String s) {

if (s.length() ==0) {

return s;

}

char [] c = s.toCharArray();

int cnt = 0;

int i = 0;

while ( i < s.length() ){

if (c[i] != ' '){

i++;

cnt++;

}

else {

int high = i-1;

int low = i-cnt;

while(low <= high){

char temp = c[low];

c[low] = c[high];

c[high] = temp;

low++;

high--;

}

cnt = 0;

i++;

}

}

int high = i-1;

int low = i-cnt;

while(low <= high){

char temp = c[low];

c[low] = c[high];

c[high] = temp;

low++;

high--;

}

return new String (c);

}

}

## 567 Permutation in String

public class Solution {

public boolean checkInclusion(String s1, String s2) {

int len1 = s1.length();

int len2 = s2.length();

if (len1 > len2){return false;}

else if (len1 == 0){return true;}

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

HashSet<Character> set = new HashSet<>();

for (int i = 0; i < len1; i++){

char c = s1.charAt(i);

map.put(c,map.getOrDefault(c,0) + 1);

set.add(c);

}

for (int i = len1 - 1; i < len2; i++){

if (set.contains(s2.charAt(i)) && set.contains(s2.charAt(i - len1 + 1))){

HashMap<Character,Integer> nmap = new HashMap<>(map);

if (isSame(nmap,s2,i - len1 + 1, i)){return true;}

}

}

return false;

}

public boolean isSame(HashMap<Character,Integer> map, String s2, int start, int end){

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

char c = s2.charAt(i);

if (map.get(c) == null || map.get(c) == 0){return false;}

else {map.put(c, map.get(c) - 1);}

}

return true;

}

}

# Greedy

## 134 Gas Station

public class Solution {

public int canCompleteCircuit(int[] gas, int[] cost) {

int sumGas = 0, sumCost = 0;

for (int i : gas){sumGas += i;}

for (int i : cost){sumCost += i;}

if (sumGas < sumCost){return -1;}

int start = 0, curGas = 0, curCost = 0;

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

curGas += gas[i];

curCost += cost[i];

if (curGas< curCost){start = i + 1;curGas = 0;curCost = 0;}

}

return start;

}

}

# Tree

## 94 [Binary Tree Inorder Traversal](https://leetcode.com/problems/binary-tree-inorder-traversal/)

import java.util.\*;

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public List<Integer> inorderTraversal(TreeNode root) {

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

helpfunction (root, rev);

return rev;

}

public void helpfunction (TreeNode root, List<Integer> rev){

if (root == null) {

return;

}

helpfunction(root.left,rev);

rev.add(root.val);

helpfunction(root.right,rev);

}

}

## 95. Unique Binary Search Trees II

//(recursion)

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public List<TreeNode> generateTrees(int n) {

if (n <= 0) {return new ArrayList<TreeNode>();}

return buildTree(1, n);

}

public List<TreeNode> buildTree (int start, int end){

List<TreeNode> rev = new ArrayList<>();

if (start > end) {rev.add(null); return rev;}

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

List<TreeNode> leftChild = buildTree(start, i - 1);

List<TreeNode> rightChild = buildTree(i + 1, end);

for (TreeNode l : leftChild){

for (TreeNode r : rightChild){

TreeNode root = new TreeNode(i);

root.left = l;

root.right = r;

rev.add(root);

}

}

}

return rev;

}

}

## 96. Unique Binary Search Trees

//O(n) RT and Space;

//dp[3] = dp [0] \* dp[2] + dp[1]\*dp[1] + dp[2]\*dp[0]

public class Solution {

public int numTrees(int n) {

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

dp[0] = 1;

dp[1] = 1;

for (int i = 2; i <= n; i++){

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

dp[i] += dp[j] \* dp[i - 1 - j];

}

}

if (n <= 0) return 0;

else return dp[n];

}

}

## 98. Validate Binary Search Tree

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

//utilize the property of inorder

//RT O(n), Space O(n)

// no equal case

public class Solution {

public boolean isValidBST(TreeNode root) {

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

inorder(root, rev);

if (rev.size() <= 1) return true;

else {

int i = 1;

while (i < rev.size()){

if (rev.get(i) <= rev.get(i - 1)){return false;}

i++;

}

return true;

}

}

public void inorder (TreeNode root, List<Integer> rev){

if (root == null) return;

inorder (root.left, rev);

rev.add(root.val);

inorder(root.right,rev);

}

}

## 99. Recover Binary Search Tree

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

//utilize the property of inorder

// RT : O(n) Space O(1);

public class Solution {

TreeNode first = null, second = null, prev = null;

public void recoverTree(TreeNode root) {

inorder (root);

int temp = first.val;

first.val = second.val;

second.val = temp;

}

public void inorder (TreeNode root){

if (root == null) return;

inorder (root.left);

if (prev != null && prev.val >= root.val){

if (first == null) first = prev;

second = root;

}

prev = root;

inorder(root.right);

}

}

## 102 [Binary Tree Level Order Traversal](https://leetcode.com/problems/binary-tree-level-order-traversal)

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public List<List<Integer>> levelOrder(TreeNode root) {

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

if (root == null){return l;}

Queue<TreeNode> q = new LinkedList<>();

q.offer(root);

//BFS

while (!q.isEmpty()){

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

int size= q.size();

int i=0;

while (i<size){

TreeNode cur= q.poll();

level.add(cur.val);

if(cur.left != null){q.offer(cur.left);}

if (cur.right != null){q.offer(cur.right);}

i++;

}

l.add(level);

}

return l;

}

}

## 103 Binary Tree Zigzag Level Order Traversal

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public List<List<Integer>> zigzagLevelOrder(TreeNode root) {

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

Queue <TreeNode> q = new LinkedList<>();

if (root ==null) {return l;}

q.offer(root);

int count =0;

while (!q.isEmpty()){

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

int size = q.size();

int i=0;

while (i<size){

TreeNode cur = q.poll();

if (count%2 ==0) {

level.add(cur.val);

}

if (count%2 !=0) {

level.add(0,cur.val);

}

if (cur.left !=null) {q.offer(cur.left);}

if (cur.right !=null) {q.offer(cur.right);}

i++;}

count++;

l.add(level);

}

return l;

}

}

## 104 Maximum Depth of Binary Tree

import java.lang.Math.\*;

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public int maxDepth(TreeNode root) {

if (root ==null) {return 0;}

else {

return 1+ Math.max(maxDepth(root.right),maxDepth(root.left));

}

}

}

## 105. Construct Binary Tree from Preorder and Inorder Traversal

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

// 假设树的先序遍历是12453687，中序遍历是42516837。root is 1

//preorder can get the root first;

//inorder can divide into two part;

// RT: O(n) Space O(n)

public class Solution {

public TreeNode buildTree(int[] preorder, int[] inorder) {

int len = preorder.length;

if (len == 0) return null;

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

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

map.put(inorder[i], i);

}

return help (preorder, 0, inorder, 0, len - 1, map);

}

public TreeNode help (int[] preorder, int preS, int[] inorder, int inS, int inE, Map<Integer, Integer> map) {

if (inS > inE) return null;

int val = preorder[preS];

int index = map.get(val);

TreeNode root = new TreeNode(val);

root.left = help (preorder,preS + 1, inorder, inS, index - 1, map);

root.right = help (preorder,preS + index - inS + 1, inorder, index + 1, inE, map);

return root;

}

}

## 106. Construct Binary Tree from Inorder and Postorder Traversal

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

// inorder left root right

// postorder left right root

// O(N) RT and Space

public class Solution {

public TreeNode buildTree(int[] inorder, int[] postorder) {

int len = inorder.length;

if (len == 0) return null;

else {

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

int i = 0;

while (i < len){map.put(inorder[i],i++);}

return help (postorder, len - 1, inorder, 0, len - 1, map);

}

}

public TreeNode help (int [] postorder, int end, int [] inorder, int inS, int inE, Map<Integer, Integer> map){

if (inS > inE) return null;

int val = postorder[end];

int index = map.get(val);

TreeNode root = new TreeNode(val);

root.left = help (postorder, end - (inE - index) - 1, inorder, inS , index - 1, map);

root.right = help (postorder, end - 1 , inorder, index + 1 , inE, map);

return root;

}

}

## 107 [Binary Tree Level Order Traversal II](https://leetcode.com/problems/binary-tree-level-order-traversal-ii)

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public List<List<Integer>> levelOrderBottom(TreeNode root) {

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

if (root== null) {return l;}

Queue <TreeNode> q = new LinkedList<> ();

q.offer (root);

//bfs

while (!q.isEmpty()) {

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

int size = q.size();

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

System.out.println(q.size());

TreeNode cur = q.poll();

level.add(cur.val);

//add cur node children

if (cur.left != null ){q.offer(cur.left);}

if (cur.right != null ){q.offer(cur.right);}

}

l.add(0,level);

}

return l;

}

}

## 108 Convert Sorted Array to Binary Search Tree

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public TreeNode sortedArrayToBST(int[] nums) {

if (nums == null) {

return null;

}

else {

return BuildTree (nums, 0, nums.length-1);

}

}

public TreeNode BuildTree (int [] nums, int start, int end){

if (start > end) {

return null;

}

TreeNode node = new TreeNode (nums[(start + end)/2]);

node.left = BuildTree (nums,start, (start + end)/2 -1 );

node.right = BuildTree (nums,(start + end)/2 + 1, end );

return node;

}

}

## 109. Convert Sorted List to Binary Search Tree

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

//RT: O(n) Space O(n)

public class Solution {

public TreeNode sortedListToBST(ListNode head) {

if (head == null) return null;

else {

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

while (head != null){

rev.add(head.val);

head = head.next;

}

return help (rev, 0, rev.size() - 1);

}

}

public TreeNode help (List<Integer> rev, int start, int end){

if (start > end) return null;

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

TreeNode node = new TreeNode (rev.get(mid));

node.left = help(rev, start, mid - 1);

node.right = help(rev, mid + 1, end);

return node;

}

}

## 100. Same Tree

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

//O(n)

public class Solution {

public boolean isSameTree(TreeNode p, TreeNode q) {

if (p == null && q == null) {return true;}

else if (p != null && q!= null){

if (p.val != q.val) return false;

else {

return isSameTree(p.left, q.left) && isSameTree(p.right,q.right);

}

}

else {return false;}

}

}

## 101. Symmetric Tree

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public boolean isSymmetric(TreeNode root) {

if (root == null) return true;

return help(root.left, root.right);

}

public boolean help (TreeNode l, TreeNode r){

if (l == null && r == null) return true;

if (l != null && r != null) {

if (l.val != r.val) return false;

return help(l.left, r.right) && help(l.right, r.left);

}

else {return false;}

}

}

## 110 Balanced Binary Tree

import java.lang.Math.\*;

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public boolean isBalanced(TreeNode root) {

if (root == null) {return true;}

if (Math.abs(heightoftree(root.left)-heightoftree(root.right))<=1)

{

return ( isBalanced (root.left) && isBalanced (root.right));

}

else {

return false;

}

}

public int heightoftree (TreeNode root){

if (root ==null) {return 0;}

else {

return 1 + Math.max(heightoftree(root.left), heightoftree(root.right));

}

}

}

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

//O(n) solution

public class Solution {

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

public boolean isBalanced(TreeNode root) {

if (root == null) return true;

dfs(root);

int a = map.getOrDefault(root.left, -1); //System.out.println("a " + a);

int b = map.getOrDefault(root.right, -1);//System.out.println("b" + b);

if (Math.abs(a - b) <=1 && isBalanced(root.left) && isBalanced(root.right)) return true;

return false;

}

public int dfs(TreeNode root){

if (root == null) return -1;

int height = Math.max(dfs(root.left),dfs(root.right)) + 1;

map.put(root, height);

//System.out.println(height);

return height;

}

}

## 111. Minimum Depth of Binary Tree

//BFS O(N)

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

// RT: O(n)

public class Solution {

public int minDepth(TreeNode root) {

if (root == null) return 0;

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

queue.offer(root);

int level = 0;

while(!queue.isEmpty()){

int size = queue.size();

while (size-- > 0){

root = queue.poll();

if (root.left == null && root.right == null) return level + 1;

if (root.left != null) queue.offer(root.left);

if (root.right != null) queue.offer(root.right);

}

level++;

}

return -1;

}

}

## 112. Path Sum

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public boolean hasPathSum(TreeNode root, int sum) {

if (root == null) return false;

else if (root.val == sum && root.left == null && root.right == null) return true;

else return hasPathSum(root.left, sum - root.val) || hasPathSum(root.right, sum - root.val);

}

}

## 113 Path Sum II

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

// O(N) runnint time;

public class Solution {

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

public List<List<Integer>> pathSum(TreeNode root, int sum) {

if(root == null) return rev;

search(root, new ArrayList<Integer>(),sum);

return rev;

}

public void search (TreeNode root, List<Integer> list, int sum){

list.add(root.val);

sum -= root.val;

if (root.left == null && root.right == null &&sum == 0){

rev.add(list);return;}

if (root.left != null) search (root.left, new ArrayList<Integer>(list), sum);

if (root.right != null) search (root.right, new ArrayList<Integer>(list), sum);

}

}

## 437. Path Sum III

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

int count = 0;

public int pathSum(TreeNode root, int sum) {

if (root == null) return 0;

help(root, sum, new HashMap<Integer, Integer> ());

return count;

}

public void help (TreeNode root, int sum, Map<Integer, Integer> map){

if (root == null) return;

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

if (map.size() != 0){

//updateMap

for (int i : map.keySet()){

nMap.put(i - root.val, map.get(i));

}

}

nMap.put(sum - root.val, nMap.getOrDefault(sum - root.val, 0) + 1);

count += nMap.getOrDefault(0, 0);

help(root.left, sum, nMap);

help(root.right, sum, nMap);

}

}

## 124 Binary Tree Maximum Path Sum

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

// max1 : max value in subtree including node;

// max2 : max value from child to root

int max1=Integer.MIN\_VALUE;

public int maxPathSum(TreeNode root) {

if (root == null){

return 0;

}

postorder(root);

return max1;

}

public int postorder (TreeNode root){

if (root == null) {

return 0;

}

int left = postorder(root.left);

int right = postorder(root.right);

int ro = root.val;

int l = left + ro;

int r = right + ro;

int t = left + right + ro;

int max2 = Math.max(Math.max(l,r),ro);

max1 = Math.max(Math.max(Math.max(max2,t),ro),max1);

return max2;

}

}

## 129. Sum Root to Leaf Numbers

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

//O(n)

public class Solution {

public int sumNumbers(TreeNode root) {

return help (root, 0);

}

public int help (TreeNode root, int presum){

if (root == null) return 0;

int k = presum \* 10 + root.val;

return root.left == null && root.right == null ? k : help(root.left, k) + help(root.right, k);

}

}

## 133. Clone Graph

// O(n)

/\*\*

\* Definition for undirected graph.

\* class UndirectedGraphNode {

\* int label;

\* List<UndirectedGraphNode> neighbors;

\* UndirectedGraphNode(int x) { label = x; neighbors = new ArrayList<UndirectedGraphNode>(); }

\* };

\*/

public class Solution {

public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {

if (node == null) {return null;}

Stack<UndirectedGraphNode> stack = new Stack<>();

stack.push(node);

Map<UndirectedGraphNode, UndirectedGraphNode> visited = new HashMap<>();

visited.put(node, new UndirectedGraphNode(node.label));

while(!stack.isEmpty()){

UndirectedGraphNode root = stack.pop();

for (UndirectedGraphNode n : root.neighbors){

if (!visited.containsKey(n)){

stack.add(n);

visited.put(n, new UndirectedGraphNode(n.label));

}

visited.get(root).neighbors.add(visited.get(n));

}

}

return visited.get(node);

}

}

## 144 Binary Tree Preorder Traversal

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public List<Integer> preorderTraversal(TreeNode root) {

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

Stack <TreeNode> stack = new Stack <TreeNode>();

if (root != null){

stack.push(root);

}

//dfs

while (!stack.isEmpty()){

root = stack.pop();

rev.add(root.val);

if (root.right !=null){

stack.push(root.right);

}

if (root.left !=null){

stack.push(root.left);

}

}

return rev;

}

}

## 199 Binary Tree Right Side View

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public List<Integer> rightSideView(TreeNode root) {

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

if (root == null){return l;}

Queue <TreeNode> q = new LinkedList <>();

q.offer(root);

while (!q.isEmpty()){

int i=0;

int size = q.size();

while (i<size){

TreeNode cur = q.poll();

if (cur.left !=null){q.offer(cur.left);}

if (cur.right !=null){q.offer(cur.right);}

if (i==size-1){l.add(cur.val);System.out.println("OK" + size);}

i++;

}

}

return l;

}

}

## 222 Count Complete Tree Nodes

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public int countNodes(TreeNode root) {

if (root == null){

return 0;

}

int l = 0, r =0;

TreeNode leftT = root, rightT = root;

while (leftT != null) {

l++; leftT = leftT.left;

}

while (rightT != null) {

r++; rightT = rightT.right;

}

if (l == r){

return (1<<l) -1 ;

}

else {

return countNodes(root.left) + countNodes(root.right) +1;

}

}

}

## 226 Invert Binary Tree

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public TreeNode invertTree(TreeNode root) {

if (root == null) {return root;}

else {

root.left = invertTree(root.left);

root.right = invertTree(root.right);

TreeNode temp;

temp=root.left;

root.left = root.right;

root.right = temp;

return root;

}

}

}

## 230 Kth Smallest Element in a BST

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public int kthSmallest(TreeNode root, int k) {

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

inorder(root,rev);

return rev.get(k - 1);

}

public void inorder (TreeNode root, List<Integer> rev){

if (root == null){return;}

else {

inorder(root.left,rev);

rev.add(root.val);

inorder(root.right,rev);

}

}

}

## 235. Lowest Common Ancestor of a Binary Search Tree

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

if (isInTree (root, p) && isInTree (root, q)) {

while (((root.val < p.val) && (root.val < q.val)) || ((root.val > p.val) && (root.val > q.val))) {

root = p.val < root.val ? root.left : root.right;

}

return root;

}

return null;

}

public boolean isInTree(TreeNode root, TreeNode p){

if (root == null) return false;

else if (root.val == p.val) return true;

else if (root.val > p.val) return isInTree (root.left, p);

else return isInTree(root.right, p);

}

}

## 236 Lowest Common Ancestor of a Binary Tree

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

if (root == null || root ==p || root == q ) {return root;}

TreeNode left = lowestCommonAncestor(root.left,p,q);

TreeNode right = lowestCommonAncestor(root.right,p,q);

if (left != null && right !=null) {return root;}

else if (left ==null) {return right;}

else {return left;}

}

}

## 257 Binary Tree Paths

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public List<String> binaryTreePaths(TreeNode root) {

List <String> rev = new ArrayList<String> ();

searchLeaf(root,"",rev);

return rev;

}

public void searchLeaf(TreeNode root, String s, List <String> rev){

if (root == null) {

return ;

}

if (root.left == null && root.right == null) {

rev.add(s + root.val);

}

if (root.left != null){

searchLeaf(root.left, s + root.val + "->", rev);

}

if (root.right != null){

searchLeaf(root.right, s + root.val + "->", rev);

}

}

}

## 310 Minimum Height Trees

public class Solution {

public List<Integer> findMinHeightTrees(int n, int[][] edges) {

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

for (int i = 0; i < n; i++){map.put(i, new HashSet<Integer>());}

int row = edges.length;

//store edges for every node

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

map.get(edges[i][0]).add(edges[i][1]);

map.get(edges[i][1]).add(edges[i][0]);

}

HashSet <Integer> leaves = new HashSet <>();

for (int i : map.keySet()){

if (map.get(i).size() == 1){leaves.add(i);}

}

while (map.size() > 2){

HashSet<Integer> newLeaves = new HashSet<>();

//update graph

for (int i : leaves){

System.out.println(i);

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

map.get(j).remove(i);

if (map.get(j).size() == 1){newLeaves.add(j);}

}

map.remove(i);

}

leaves = newLeaves;

}

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

for (int i : map.keySet()){

list.add(i);

}

return list;

}

}

## 404 Sum of Left Leaves

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

class Solution {

public int sumOfLeftLeaves(TreeNode root) {

if (root == null) return 0;

int left = 0;

if (root.left != null && root.left.left ==null && root.left.right == null) left = root.left.val;

else left = sumOfLeftLeaves(root.left);

return left + sumOfLeftLeaves(root.right);

}

}

## 450 [Delete Node in a BST](https://leetcode.com/problems/delete-node-in-a-bst)

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public TreeNode deleteNode(TreeNode root, int key) {

if (root == null) {

return root;

}

else if (root.val == key ){

if (root.left == null && root.right == null){

return null;

}

else if (root.left == null){

return root.right;

}

else if (root.right == null){

return root.left;

}

else {

TreeNode tempL = root.left;

TreeNode tempR = root.right;

while (tempR.left != null) {

tempR = tempR.left;

}

tempR.left = tempL;

return root.right;

}

}

else if (root.val < key) {

root.right = deleteNode (root.right,key);

return root;

}

else {

root.left = deleteNode (root.left,key);

return root;

}

}

}

## 508 Most Frequent Subtree Sum

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

import java.lang.Math.\*;

public class Solution {

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

int freq = 0;

public int[] findFrequentTreeSum(TreeNode root) {

if (root == null) {

return new int [] {};

}

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

postorderSum (root);

for (int i :map.keySet()){

if (map.get(i) == freq){

rev.add(i);

}

}

int [] nums = new int [rev.size()] ;

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

nums[i] = rev.get(i);

}

return nums;

}

public int postorderSum (TreeNode root){

if (root == null) {

return 0;

}

int left = postorderSum (root.left);

int right = postorderSum (root.right);

int sum = left + right + root.val;

int count = map.getOrDefault(sum,0) + 1;

freq = Math.max (freq,count);

map.put(sum,count);

return sum;

}

}

## 501 Find Mode in Binary Search Tree

**import java.lang.Math.\*;**

**/\*\***

**\* Definition for a binary tree node.**

**\* public class TreeNode {**

**\* int val;**

**\* TreeNode left;**

**\* TreeNode right;**

**\* TreeNode(int x) { val = x; }**

**\* }**

**\*/**

**public class Solution {**

**public int[] findMode(TreeNode root) {**

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

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

**int max = 0;**

**//BFS**

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

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

**queue.offer(root);**

**}**

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

**root = queue.poll();**

**if (!map.containsKey(root.val)){**

**map.put(root.val,1);**

**max = Math.max(max,map.get(root.val));**

**}**

**else {**

**map.put(root.val, map.get(root.val) +1);**

**max = Math.max(max,map.get(root.val));**

**}**

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

**queue.offer(root.left);**

**}**

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

**queue.offer(root.right);**

**}**

**}**

**for (int key : map.keySet()){**

**if (map.get(key) == max){**

**rev.add(key);**

**}**

**}**

**System.out.println ("max is " + max);**

**System.out.println ("size is " + rev.size());**

**int [] nums = new int [rev.size()];**

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

**nums[i] = rev.get(i);**

**System.out.println ("getvalue is " + rev.get(i));**

**}**

**return nums;**

**}**

**}**

## 513 Find Bottom Left Tree Value

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public int findBottomLeftValue(TreeNode root) {

//bfs

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

if (root !=null){

queue.offer(root);

}

while (!queue.isEmpty()){

root = queue.poll();

if (root.right !=null){

queue.offer(root.right);

}

if (root.left !=null){

queue.offer(root.left);

}

}

return root.val;

}

}

## 515 Find Largest Value in Each Tree Row

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public List<Integer> largestValues(TreeNode root) {

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

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

Queue <Integer> level = new LinkedList <Integer>();

int max = 0;

if (root !=null) {

queue.offer(root);

level.offer(0);

max =root.val;

}

int curLevel =0;

int temp = 0;

while (!queue.isEmpty()){

root = queue.poll();

curLevel = level.poll();

if (curLevel == temp){

max = Math.max(max,root.val);

}

else{

rev.add(max);

max = root.val;

}

temp = Math.max(temp,curLevel);

if (root.left != null){

queue.offer(root.left);

level.offer(curLevel +1);

}

if (root.right != null){

queue.offer(root.right);

level.offer(curLevel +1);

}

}

if (root !=null){

rev.add(max);

}

return rev;

}

}

## 530 Minimum Absolute Difference in BST

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

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

int min = Integer.MAX\_VALUE;

public int getMinimumDifference(TreeNode root) {

if (root == null) {

return -1;

}

inorder (root);

for (int i : rev){

System.out.println(i);

}

for (int i = 1;i < rev.size(); i ++){

min = Math.min(rev.get(i)-rev.get(i-1),min);

}

return min;

}

public void inorder (TreeNode root){

if (root == null) {

return;

}

inorder (root.left);

rev.add(root.val);

inorder (root.right);

}

}

## 543 Diameter of Binary Tree

import java.lang.Math.\*;

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

int max = 0;

public int diameterOfBinaryTree(TreeNode root) {

height(root);

return max;

}

public int height (TreeNode root){

if (root == null){

return 0;

}

else {

int left = height(root.left);

int right = height(root.right);

max = Math.max(max,left + right);

return (1 + Math.max(left,right));

}

}

}

## 563 Binary Tree Tilt

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

int sum =0;

public int findTilt(TreeNode root) {

help(root);

return sum;

}

public int help (TreeNode root){

if (root ==null) {

return 0;

}

int left = help(root.left);

int right = help(root.right);

if (left > right){

sum += left -right;

}

else {

sum += right - left;

}

return root.val + left + right;

}

}

## 572 Subtree of Another Tree

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

// RT: worst Case O(s.size \* t.size)

public class Solution {

public boolean isSubtree(TreeNode s, TreeNode t) {

if (isSame (s, t)){return true;}

else {

if (s == null) {return false;}

else {

return isSubtree(s.left,t) || isSubtree(s.right,t);

}

}

}

public boolean isSame (TreeNode s, TreeNode t){

if (s == null && t == null){

return true;

}

else if (s != null && t != null){

if (s.val != t.val){return false;}

else {return isSame(s.left,t.left) && isSame(s.right,t.right);}

}

else {return false;}

}

}

# DP

## 72 Edit Distance

public class Solution {

public int minDistance(String word1, String word2) {

int len1 = word1.length();

int len2 = word2.length();

//basecase

int [][] dp = new int [len1 + 1][len2 + 1];

for (int i = 0; i < len1; i++){dp[i + 1][0] = i + 1;}

for (int i = 0; i < len2; i++){dp[0][i + 1] = i + 1;}

for (int i = 0; i < len1; i++){

for (int j = 0; j < len2; j++){

if (word1.charAt(i) == word2.charAt(j)){dp[i+1][j+1] = dp[i][j];}

else {

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

}

}

}

return dp[len1][len2];

}

}

co

## 97. Interleaving String

//RT and space O(mn)

//Be care of the position [0][0] and position of string

public class Solution {

public boolean isInterleave(String s1, String s2, String s3) {

int len1 = s1.length(), len2 = s2.length(), len3 = s3.length();

if (len1 + len2 != len3) return false;

boolean [][]dp = new boolean [len1 + 1][len2 + 1];

dp[0][0] = true;// Base Case

for (int i = 0 ; i <= len1; i++){

for (int j = 0; j <= len2; j++){

if (i == 0 && j == 0) continue;

boolean m = i == 0 ? false : dp[i - 1][j] && s1.charAt(i - 1) == s3.charAt(i + j - 1);;

boolean n = j == 0 ? false : dp[i][j - 1] && s2.charAt(j - 1) == s3.charAt(i + j - 1);

dp[i][j] = m || n;

}

}

return dp[len1][len2];

}

}

## 120 [Triangle](https://leetcode.com/problems/triangle/)

//dp

public class Solution {

public int minimumTotal(List<List<Integer>> triangle) {

int row = triangle.size();

int col = triangle.get(row - 1).size();

int [][] dp = new int [row][col];

// base case

for (int j = 0; j < col; j++){dp[row - 1][j] = triangle.get(row - 1).get(j);}

for (int i = row - 2; i >= 0; i--){

for (int j = 0; j < triangle.get(i).size(); j++){

int m = triangle.get(i).get(j);

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

}

}

return dp[0][0];

}

}

## 198 House Robber

public class Solution {

public int rob(int[] nums) {

int len = nums.length;

if (len == 0) {return 0;}

if (len == 1) {return nums[0];}

if (len == 2) {return Math.max(nums[0],nums[1]);}

int [] dp = new int [len];

//base case;

dp[0] = nums[0];

dp[1] = Math.max(nums[0],nums[1]);

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

dp[i] = Math.max(dp[i - 1], dp[i - 2] + nums[i]);

}

return dp[len - 1];

}

}

## 213 House Robber II

public class Solution {

public int rob(int[] nums) {

int len = nums.length;

if (len == 0){return 0;}

if (len == 1){return nums[0];}

if (len == 2){return Math.max(nums[0],nums[1]);}

int [][] dp = new int [2][len];

// the first line: position 0 is not robbed;

// the second line: otherwise

dp[0][0] = 0; dp[0][1] = nums[1];

dp[1][0] = nums[0];dp[1][1] = nums[0];

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

dp[0][i] = Math.max(nums[i] +dp[0][i - 2], dp[0][i - 1]);

dp[1][i] = i == len - 1? Math.max(dp[1][i - 2], dp[1][i - 1]) :Math.max(nums[i] +dp[1][i - 2], dp[1][i - 1]);

}

return Math.max(dp[0][len - 1], dp[1][len - 1]);

}

}

## 264 [Ugly Number II](https://leetcode.com/problems/ugly-number-ii)

public class Solution {

public int nthUglyNumber(int n) {

// dp solution Time complexity O(n)

//dp[0] = 1;

//dp[1] = min[dp[0]\*2,dp[0]\*3,dp[0]\*5] 2 = dp[0]\*2

//dp[2] = min[dp[1]\*2,dp[0]\*3,dp[0]\*5] 2 = dp[0]\*2

if (n <= 0){

return -1;

}

if (n == 1){

return n;

}

int [] dp = new int [n];

dp [0] = 1;

int i = 0, j = 0, k = 0;

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

dp[m] = Math.min(Math.min(2 \* dp[i], 3 \* dp[j]), 5 \* dp[k]);

if (dp[m] == 2 \* dp[i]){

i++;

}

if (dp[m] == 3 \* dp[j]){

j++;

}

if (dp[m] == 5 \* dp[k]){

k++;

}

}

return dp[n-1];

}

}

## 322. Coin Change

public class Solution {

public int coinChange(int[] coins, int amount) {

int len = coins.length;

if (len == 0 || amount <= 0) return 0;

int [] dp = new int [amount + 1];

Arrays.fill(dp, -1);

dp [0] = 0;

for (int c : coins){

for (int i = c; i <= amount ; i++){

if (dp [i] == -1) {

if (dp[i - c] != -1) {dp[i] = dp[i - c] + 1;}

}

else {

if (dp[i - c] != -1) {dp[i] = Math.min(dp[i],dp[i - c] + 1);}

}

}

}

return dp[amount];

}

}

## 518 [Coin Change 2](https://leetcode.com/problems/coin-change-2/)

public class Solution {

public int change(int amount, int[] coins) {

int [] dp = new int [amount + 1];

dp[0] = 1;

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

for (int j = coins[i]; j <= amount; j++){

dp[j] += dp[j - coins[i]];

}

}

return dp[amount];

}

}

// RT: O(amount\*len); spcace O(amount);

## 583 [Delete Operation for Two Strings](https://leetcode.com/problems/delete-operation-for-two-strings/)

public class Solution {

public int minDistance(String word1, String word2) {

int len1 = word1.length();

int len2 = word2.length();

if (len1 == 0){return len2;}

if (len2 == 0){return len1;}

int [][] dp = new int [len1][len2];

for (int i = 0; i < len1; i++){

for (int j = 0; j < len2; j++){

if (word1.charAt(i) == word2.charAt(j)){

int m = i - 1 < 0 || j - 1 < 0 ? 0 : dp[i - 1][j - 1];

dp[i][j] = m + 1;

}

else{

int n = i - 1 < 0 ? 0 : dp[i - 1][j];

int k = j - 1 < 0 ? 0 : dp[i][j - 1];

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

}

}

}

return len1 + len2 - 2\* dp[len1 - 1][len2 - 1];

}

}

# LinkedList

## 19 Remove Nth Node From End of List

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

public class Solution {

public ListNode removeNthFromEnd(ListNode head, int n) {

if (head == null || n <= 0) {

return head;

}

int length = 0;

List<ListNode> rev = new ArrayList<>();

ListNode checkLength = head;

while (checkLength!= null){

rev.add(checkLength);

length++;

checkLength = checkLength.next;

}

if (n >=length || rev.size() == 1){

return head.next;

}

if (n == 1){

rev.get(length-n-1).next = null;

}

else {

System.out.println(length);

System.out.println(rev.get(length - n -1).val);

System.out.println(rev.get(length - n +1).val);

rev.get(length - n -1).next = rev.get(length - n +1);

}

return rev.get(0);

}

}

## 21 [Merge Two Sorted Lists](https://leetcode.com/problems/merge-two-sorted-lists)

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

public class Solution {

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

if (l1 == null){

return l2;

}

if (l2 == null){

return l1;

}

ListNode newHead;

if (l1.val < l2.val){

newHead = l1;

newHead.next = mergeTwoLists (l1.next,l2);

}

else {

newHead = l2;

newHead.next = mergeTwoLists (l1,l2.next);

}

return newHead;

}

}

## 23 Merge k Sorted Lists

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

public class Solution {

public ListNode mergeKLists(ListNode[] lists) {

if (lists.length == 0) {

return null;

}

return partition(lists, 0, lists.length - 1);

}

public ListNode partition (ListNode [] lists, int start, int end){

if (start == end){

if (lists[start] == null) {

return null;

}

return lists[start];

}

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

ListNode l1 = partition (lists, start, mid);

ListNode l2 = partition (lists, mid + 1, end);

//merge two list

ListNode dummy = new ListNode (0);

ListNode temp = dummy;

while (l1 != null && l2 != null) {

if (l1.val < l2.val){

dummy.next = l1;

dummy = l1;

l1 = l1.next;

}

else {

dummy.next = l2;

dummy = l2;

l2 = l2.next;

}

}

dummy.next = l1 != null ? l1 : l2;

return temp.next;

}

}

//Time complexity: divider conquer T(k) = O(TotalNodes) + 2T(k/2) then O(TotalNodes) \*log(k)

## 24 Swap Nodes in Pairs

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

public class Solution {

public ListNode swapPairs(ListNode head) {

if (head == null || head.next == null){

return head;

}

if( head.next.next == null){

ListNode second = head.next;

second.next = head;

head.next = null;

return second;

}

ListNode newHead = head.next.next;

ListNode second = head.next;

second.next = head;

head.next = swapPairs(newHead);

return (second);

}

}

## 25 Reverse Nodes in k-Group

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

// For example,

// Given this linked list: 1->2->3->4->5

// For k = 2, you should return: 2->1->4->3->5

// For k = 3, you should return: 3->2->1->4->5

public class Solution {

public ListNode reverseKGroup(ListNode head, int k) {

if (head == null || head.next == null || k == 1){

return head;

}

int count = 0;

ListNode kHead = head;

while (kHead != null){

count++;

if (count == k){

break;

}

kHead = kHead.next;

}

if (count < k) {

return head;

}

//cut linkedlist

ListNode newHead = reverseKGroup (kHead.next,k);

kHead.next = null;

ListNode revHead = reverse (head);

//connnect two linkedlist

head.next = newHead;

return revHead;

}

public ListNode reverse(ListNode head){

ListNode prev = null, cur = head, temp = head;

while (cur != null){

temp = cur.next;

cur.next = prev;

prev = cur;

cur = temp;

}

return prev;

}

}

// O(n)

## 61 Rotate List

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

// Given 1->2->3->4->5->NULL and k = 2,

// return 4->5->1->2->3->NULL.

public class Solution {

public ListNode rotateRight(ListNode head, int k) {

if (head == null || head.next == null || k <= 0){

return head;

}

ListNode checkLength = head;

int length = 1;

while (checkLength.next != null){

length++;

checkLength = checkLength.next;

}

if (k > length){

k = k % length;

}

if (k == length || k == 0){

return head;

}

// k is less than length

// left half :

length = length - k;

ListNode newHead = head;

ListNode oldTail = head;

int count = 1;

while (oldTail != null && count < length){

count++;

oldTail = oldTail.next;

}

newHead = oldTail.next;

checkLength.next = head;

oldTail.next = null;

return newHead;

}

}

## 82 [Remove Duplicates from Sorted List II](https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/)

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

//Given 1->2->3->3->4->4->5, return 1->2->5

// 1 1 1 1 1

public class Solution {

public ListNode deleteDuplicates(ListNode head) {

if (head == null || head.next == null) {

return head;

}

if (head.val == head.next.val){

ListNode newHead = deleteDuplicates (head.next);

if (newHead == null){

return null;

}

return head.val == newHead.val ? newHead.next : newHead;

}

else {

ListNode newHead = deleteDuplicates (head.next);

head.next = newHead;

return head;

}

}

}

//time complexicy : O (n).

## 86 Partition List

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

//Given 1->4->3->2->5->2 and x = 3,

//return 1->2->2->4->3->5.

public class Solution {

public ListNode partition(ListNode head, int x) {

if (head == null){

return head;

}

List<ListNode> less = new ArrayList<>();

List<ListNode> larger = new ArrayList<>();

ListNode temp = head;

while (head != null){

if (head.val < x){

less.add(head);

}

else {

larger.add(head);

}

head = head.next;

}

if (less.size() == 0 || larger.size() == 0){

return temp;

}

ListNode lessHead = less.get(0);

ListNode lessTail = less.get(less.size() - 1 );

ListNode largerHead = larger.get(0);

ListNode largerTail = larger.get(larger.size() - 1 );

if (less.size() >= 2){

for (int i = 0; i < less.size() -1; i++){

ListNode first = less.get(i);

ListNode second = less.get(i+1);

first.next = second;

}

}

if (larger.size() >= 2){

for (int i = 0; i < larger.size() -1; i++){

ListNode first = larger.get(i);

ListNode second = larger.get(i+1);

first.next = second;

}

}

//conection of two list

lessTail.next = largerHead;

largerTail.next = null;

return lessHead;

}

}

## 92 Reverse Linked List II

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

public class Solution {

public ListNode reverseBetween(ListNode head, int m, int n) {

ListNode Pre1 = null;

ListNode Cur1 = head;

int count = 1;

while (count < m){

Pre1 = Cur1;

Cur1 = Cur1.next;

count++;

}

ListNode Cur2 = Cur1;

ListNode Pre2 = Pre1;

ListNode Next = null;

while (count <= n){

Next = Cur2.next;

Cur2.next = Pre2;

Pre2 = Cur2;

Cur2 = Next;

count++;

}

// connection

if (m !=1){

Pre1.next = Pre2;

Cur1.next = Cur2;

return head;

}

else {

Cur1.next = Cur2;

return Pre2;

}

}

}

## 114 [Flatten Binary Tree to Linked List](https://leetcode.com/problems/flatten-binary-tree-to-linked-list/)

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

public void flatten(TreeNode root) {

if (root == null) {

return;

}

else if (root.left == null){

flatten(root.right);

}

else {

flatten(root.right);

flatten(root.left);

TreeNode tempR = root.right;

TreeNode tempL = root.left;

root.left =null;

root.right=tempL;

while (tempL.right !=null){

tempL = tempL.right;

}

tempL.right =tempR;

}

}

}

## 141 Linked List Cycle

/\*\*

\* Definition for singly-linked list.

\* class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) {

\* val = x;

\* next = null;

\* }

\* }

\*/

public class Solution {

public boolean hasCycle(ListNode head) {

ListNode fast = head;

ListNode slow = head;

if (head == null) {

return false;

}

while (fast.next !=null && fast.next.next != null){

fast = fast.next.next;

slow = slow.next;

if (fast == slow) {

return true;

}

}

return false;

}

}

## 142 Linked List Cycle I’I

/\*\*

\* Definition for singly-linked list.

\* class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) {

\* val = x;

\* next = null;

\* }

\* }

\*/

public class Solution {

public boolean hasCycle(ListNode head) {

ListNode fast = head;

ListNode slow = head;

if (head == null) {

return false;

}

while (fast.next !=null && fast.next.next != null){

fast = fast.next.next;

slow = slow.next;

if (fast == slow) {

return true;

}

}

return false;

}

}

## 143 Reorder List

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

//　[1,2,3,4,5,6]

// [1,2,3]

// [4,5,6] -> [6,5,4]

//merge

public class Solution {

public void reorderList(ListNode head) {

if (head == null || head.next == null){

return ;

}

ListNode fast = head.next, preMid = head;

while ( fast != null && fast.next != null){

fast = fast.next.next;

preMid = preMid.next;

}

ListNode l2 = reverse(preMid.next);

preMid.next = null;

ListNode newHead = new ListNode (0);

while (head !=null && l2 != null){

newHead.next = head;

newHead = head;

head = head.next;

newHead.next = l2;

newHead = l2;

l2 = l2.next;

}

newHead.next = (head != null) ? head : l2;

}

public ListNode reverse (ListNode head){

ListNode pre = null , cur = head, temp = head;

while (head != null){

temp = head.next;

head.next = pre;

pre = head;

head = temp;

}

return pre;

}

}

## 147 Insertion Sort List

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

public class Solution {

public ListNode insertionSortList(ListNode head) {

if (head == null || head.next == null){

return head;

}

ListNode cur = head.next;

while (cur != null){

ListNode ins = head;

while (ins != cur){

if (cur.val < ins.val){

//swap

int temp = ins.val;

ins.val = cur.val;

cur.val = temp;

}

ins = ins.next;

}

cur = cur.next;

}

return head;

}

}

## 148 [Sort List](https://leetcode.com/problems/sort-list)

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

public class Solution {

public ListNode sortList(ListNode head) {

if (head == null || head.next == null){

return head;

}

ListNode temp = new ListNode (0), fast = head;

temp.next = head;

//find mid

while (fast != null && fast.next != null){

temp = temp.next;

fast = fast.next.next;

}

ListNode mid = temp.next;

temp.next = null;

ListNode l1 = sortList(head);

ListNode l2 = sortList(mid);

//merge two list

ListNode temp2 = new ListNode (0);

ListNode finalNode = temp2;

while (l1 != null && l2 !=null){

if (l1.val < l2.val){

temp2.next = l1;

temp2 = temp2.next;

l1 = l1.next;

}

else {

temp2.next = l2;

temp2 = temp2.next;

l2 = l2.next;

}

}

temp2.next = (l1 != null) ? l1 : l2;

return finalNode.next;

}

}

## 160 Intersection of Two Linked Lists

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) {

\* val = x;

\* next = null;

\* }

\* }

\*/

public class Solution {

public ListNode getIntersectionNode(ListNode headA, ListNode headB) {

if (headA == null || headB == null){

return null;

}

HashSet <ListNode> node = new HashSet <ListNode> ();

while (headA !=null){

node.add(headA);

headA = headA.next;

}

while (headB != null){

if (node.contains(headB)){

return headB;

}

else {

headB = headB.next;

}

}

return null;

}

}

## 203 [Remove Linked List Elements](https://leetcode.com/problems/remove-linked-list-elements/)

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

public class Solution {

public ListNode removeElements(ListNode head, int val) {

if (head == null){

return null;

}

head.next = removeElements(head.next,val);

if (head.val == val){

return head.next;

}

else {

return head;

}

}

}

## 234 Palindrome Linked List

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

public class Solution {

public boolean isPalindrome(ListNode head) {

int length = 0;;

if (head == null){

return true;

}

ListNode right = head;

ListNode left = head;

while (head !=null){

length++;

head = head.next;

}

int newLength = length;

while (newLength > length/2){

right = right.next;

newLength--;

}

right = reverse(right);

while (right !=null){

System.out.println(right.val);

if (left.val != right.val){

return false;

}

else {

left = left.next;

right = right.next;

}

}

return true;

}

public ListNode reverse(ListNode head) {

ListNode Prev = null;

ListNode Next = null;

while (head != null) {

Next= head.next;

head.next = Prev;

Prev = head;

head = Next;

}

return Prev;

}

}

## 237 Delete Node in a Linked List

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

public class Solution {

public void deleteNode(ListNode node) {

node.val = node.next.val;

node.next = node.next.next;

}

}

## 328 Odd Even Linked List

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

public class Solution {

public ListNode oddEvenList(ListNode head) {

if (head == null){

return null;

}

ListNode odd = head;

ListNode even = head.next;

ListNode evenH = head.next;

while (odd.next != null && odd.next.next != null){

odd.next = odd.next.next;

odd = odd.next;

even.next = even.next.next;

even = even.next;

}

if (even == null){

System.out.println("True");

}

else {

if (even.next ==null)

{System.out.println("False"); }

}

//connection

odd.next = evenH;

return head;

}

}

# Others

## 2 [Add Two Numbers](https://leetcode.com/problems/add-two-numbers)

import java.io.\*;

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

public class Solution {

public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

StringBuilder s1 = new StringBuilder ();

StringBuilder s2 = new StringBuilder ();

ListNode dummy = l1;

while (dummy.next != null){

s1.append(dummy.val);

dummy = dummy.next;

}

s1.append(dummy.val);

dummy.next = l2;

while (l2 != null){

s2.append(l2.val);

l2 = l2.next;

}

s1.reverse();s2.reverse();

String s = addTwo(s1,s2);

System.out.println(s);

ListNode dummy2 = new ListNode(0);

ListNode head = dummy2; dummy2.next = l1;

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

l1.val = s.charAt(i) - '0';

l1 = l1.next;

dummy2 = dummy2.next;

}

dummy2.next = null;

return head.next;

}

public String addTwo (StringBuilder s1, StringBuilder s2){

StringBuilder s3 = new StringBuilder ();

int i = s1.length() - 1;

int j = s2.length() - 1;

int carry = 0;

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

int f1 = i < 0 ? 0 : s1.charAt(i) - '0';

int f2 = j < 0 ? 0 : s2.charAt(j) - '0';

int digit = f1 + f2 + carry;

if (digit > 9) {

digit -= 10;

carry = 1;

}

else {carry = 0;}

i--; j--;

s3.append(digit);

}

if (carry == 1){s3.append(carry);}

return s3.toString();

}

}

## 7 Reverse Integer

import java.lang.Math.\*;

public class Solution {

public int reverse(int x) {

if (x == 0){

return x;

}

long rev = 0;

while (x / 10 != 0){

int reminder = x % 10 ;

rev = (rev +reminder) \* 10;

x = x / 10;

}

rev = rev + x;

if (rev > Integer.MAX\_VALUE || rev < Integer.MIN\_VALUE ){

return 0;

}

return (int) rev;

}

}

## 9 Palindrome Number

//idea 1: to String--lost space

//idea 2: get the reverse and compare

//RT O(logn) Space O(1)

// Be careful of the Minius case

public class Solution {

public boolean isPalindrome(int x) {

if (x < 0) return false;

int temp = x;

int sum = 0;

while (x > 0){

sum = 10 \* sum + x % 10;

x /= 10;

}

return temp == sum;

}

}

## 12 [Integer to Roman](https://leetcode.com/problems/integer-to-roman)

//XXIV 29

//XXVI 26

public class Solution {

public String intToRoman(int num) {

if (num == 0){return "";}

StringBuilder sb = new StringBuilder ();

String [] ss = {"M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"};

int [] nums = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};

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

while (num >= nums[i]){

sb.append(ss[i]);

num -= nums[i];

}

}

return sb.toString();

}

}

## 22 Generate Parentheses

public class Solution {

public List<String> generateParenthesis(int n) {

List <String> ll = new ArrayList <String> ();

String s= new String ();

if (n<=0){return ll;}

else {

helper (ll,s,n,n);

}

return ll;

}

public static void helper (List<String> ll ,String s, int l, int r){

if (l==0 && r==0) {ll.add(s);}

if (l>0) {

helper (ll,s+'(',l-1,r);

}

if (r>l){

helper (ll,s+')',l,r-1);

}

}

}

## 36. Valid Sudoku

public class Solution {

public boolean isValidSudoku(char[][] board) {

int row = board.length;

if (row == 0) return true;

int col = board[0].length;

if (col == 0) return true;

HashSet <String> set = new HashSet<>();

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

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

if (board[i][j] !='.'){

char c = board[i][j];

if (!set.add(c+"row" + i)||!set.add(c+"col"+j)||!set.add(c+""+i/3 +"block"+j/3))

return false;

}

}

}

return true;

}

}

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

//1, 11, 21, 1211, 111221, ...

public class Solution {

public String countAndSay(int n) {

StringBuilder cur = new StringBuilder("1");

StringBuilder pre = new StringBuilder();

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

pre = cur;

cur = new StringBuilder();

char say = pre.charAt(0);

int count = 1;

int len = pre.length();

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

if (pre.charAt(j) != say){

cur.append(count).append(say);

count = 1;

say = pre.charAt(j);

}

else{count++;}

}

cur.append(count).append(say);

}

return cur.toString();

}

}

## 39 Combination Sum

public class Solution {

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

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

Arrays.sort(candidates);

int len = candidates.length;

if (len == 0){return rev;}

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

subComb (sub,candidates,target,0,0);

return rev;

}

public void subComb (List<Integer> sub, int [] c, int t, int start, int sum){

if (sum <= t){

if (sum == t){rev.add(sub);}

else{

for (int i = start; i < c.length; i++){

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

cur.add(c[i]);

subComb (cur,c,t,i,sum + cur.get(cur.size() - 1));

}

}

}

}

}

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

public class Solution {

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

public List<List<Integer>> combinationSum2(int[] candidates, int target) {

Arrays.sort(candidates);

int len = candidates.length;

if (len == 0) {return rev;}

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

subComb (sub, candidates, target, 0, 0);

HashSet <List<Integer>> set = new HashSet <List<Integer>>(rev);

return new ArrayList<List<Integer>>(set);

}

public void subComb (List<Integer> sub, int [] c, int target, int start, int sum){

if (sum <= target){

if (sum == target){rev.add(sub);}

else{

for (int i = start; i < c.length; i++){

List<Integer> subN = new ArrayList<>(sub);

subN.add(c[i]);

subComb (subN, c, target, i + 1, sum + c[i]);

}

}

}

}

}

## 41 First Missing Positive

public class Solution {

public int firstMissingPositive(int[] nums) {

int n = nums.length;

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

if (nums [i] <= 0){

nums [i] = Integer.MAX\_VALUE;

}

}

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

if ( Math.abs(nums[i]) -1 >= 0 && Math.abs(nums[i]) -1 <= n -1){

if (nums[Math.abs(nums[i]) -1] >= 0) {

nums[Math.abs(nums[i]) -1] = -nums[Math.abs(nums[i]) -1];

}

}

}

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

if (nums[i] > 0){

return (i+1);

}

}

return nums.length +1;

}

}

## 42 Trapping Rain Water

public class Solution {

public int trap(int[] height) {

int len = height.length;

if (len <= 2){return 0;}

int sum = 0, l = 0, r = len - 1;

while (l < r && height[l + 1] >= height[l]){l++;}//left edge

while (l < r && height[r - 1] >= height[r]){r--;}//right edge

while (l < r){

int left = height[l];

int right = height[r];

if (left <= right){

while(height[++l] < left){

sum += left - height[l];

}

}

else {

while (height[--r] <= right){

sum += right - height[r];

}

}

}

return sum;

}

}

## 45 Jump Game II

// RT: O(n)

// creat new Field and old Field

public class Solution {

public int jump(int[] nums) {

int len = nums.length;

if (len <= 1){return 0;}

int oldField = nums[0], newField = -1, step= 1;

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

if (oldField >= len - 1){return step;}

else{

newField = Math.max(newField, nums[i] + i);

if(i == oldField) {oldField = newField;step++;}

}

}

return step;

}

}

## 50 [Pow(x, n)](https://leetcode.com/problems/powx-n)

//RT: log(n)

// 注意 (n/2) 与 (-n)/2 的区别。

// 如果 n 是 Integer.MIN\_VALUE, -n 会overflow

public class Solution {

public double myPow(double x, int n) {

if(n == 0) {return 1;}

//unsigned 2^n - 1

//singen 2^(n-1) -1 -(2^(n-1))

else if ( n < 0){

return n % 2 == 0 ? myPow((1/x)\*(1/x),-(n/2)): (1/x)\*myPow((1/x)\*(1/x),-(n/2));

}

else {return n % 2 == 0 ? myPow(x\*x,n/2): x\*myPow(x\*x,n/2);}

}

}

## 56. Merge Intervals

/\*\*

\* Definition for an interval.

\* public class Interval {

\* int start;

\* int end;

\* Interval() { start = 0; end = 0; }

\* Interval(int s, int e) { start = s; end = e; }

\* }

\*/

//RT O(nlogn)

public class Solution {

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

List<Interval> rev = new ArrayList<>();

Collections.sort(intervals, (a, b) -> (a.start - b.start));

if (intervals.size() <= 1) return intervals;

Interval prev = intervals.get(0);

rev.add(prev);

for (int i = 1; i < intervals.size(); i++){

Interval cur = intervals.get(i);

if (cur.start <= prev.end){

prev.end = Math.max(prev.end, cur.end);

}

else {

prev = cur;rev.add(prev);

}

}

return rev;

}

}

## 57. Insert Interval

/\*\*

\* Definition for an interval.

\* public class Interval {

\* int start;

\* int end;

\* Interval() { start = 0; end = 0; }

\* Interval(int s, int e) { start = s; end = e; }

\* }

\*/

public class Solution {

public List<Interval> insert(List<Interval> intervals, Interval newInterval) {

intervals.add(newInterval);

Collections.sort(intervals, (a, b) -> a.start - b.start);

Interval prev = intervals.get(0);

List<Interval> rev = new ArrayList<>();

rev.add(prev);

for (Interval cur : intervals){

if (cur.start <= prev.end){prev.end = Math.max(cur.end, prev.end);}

else {prev = cur; rev.add(prev);}

}

return rev;

}

}

## 66 [Plus One](https://leetcode.com/problems/plus-one)

//[1,1,0,9]

//[1,1,1,0]

public class Solution {

public int[] plusOne(int[] digits) {

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

int i = digits.length - 1;

int carry = 1;

while (i >= 0){

int num = digits[i] + carry;

carry = num == 10 ? 1 : 0;

rev.add(num%10);

i--;

}

if (carry == 1){rev.add(carry);}

int [] nums = new int [rev.size()];

for (int j = 0; j < rev.size(); j++){

nums[j] = rev.get(rev.size() - 1 - j);

}

return nums;

}

}

## 69 [Sqrt(x)](https://leetcode.com/problems/sqrtx)

public class Solution {

public int mySqrt(int x) {

if (x < 0) return -1;

if (x == 0) return 0;

int num = x / 2;

int i = 0, j = x;

while (i <= j){

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

if ((long)mid \* (long)mid <= (long) x && (long)(mid + 1) \* (long)(mid + 1) > (long)x){

return mid;

}

else if ((long)mid \* (long)mid > (long) x){j = mid - 1;}

else {

i = mid + 1;

}

}

return -1;

}

}

## 70 Climbing Stairs

public class Solution {

public int climbStairs(int n) {

int [] memo = new int [n+2];

//basecase

memo[0] = 0;

memo[1] = 1;

return dynamic(n,memo);

}

public int dynamic (int n , int [] memo){

if (n < 0) {

return memo[0];

}

else if (n == 0){

return memo[1];

}

else if (memo[n+1] == 0) {

memo[n+1] = dynamic (n-1, memo) + dynamic (n-2, memo);

return memo[n+1];

}

else {

return memo[n+1];

}

}

}

## 77 Combinations

public class Solution {

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

public List<List<Integer>> combine(int n, int k) {

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

subCombine (sub,1,n,k);

return rev;

}

public void subCombine(List<Integer> sub, int start, int n, int k){

if (sub.size() == k){rev.add(sub);}

if (sub.size() < k){

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

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

cur.add(i);

subCombine(cur,i + 1,n,k);

}

}

}

}

## 121 Best Time to Buy and Sell Stock

public class Solution {

public int maxProfit(int[] prices) {

int len = prices.length;

if (len <= 1){return 0;}

int max = Integer.MIN\_VALUE;

int min = prices[0];

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

max = Math.max(max, prices[i] - min);

min = Math.min(min, prices[i]);

}

return max > 0 ? max : 0;

}

}

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

// [3,8,1,5,6]

// return 10

public class Solution {

public int maxProfit(int[] prices) {

int len = prices.length;

if (len <= 1){return 0;}

int sum = 0;

int min = prices[0];

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

if (prices[i] < prices[i - 1]){

sum += prices[i-1] - min;

min = prices[i];

}

else{

min = Math.min(min,prices[i]);

}

}

if (prices[len - 1] - min > 0){

sum += prices[len - 1] - min;

}

return sum;

}

}

## 135 Candy

//case 122

//case 221

//RT O(n) ; Space O(n)

public class Solution {

public int candy(int[] ratings) {

int len = ratings.length;

if (len <= 1){return len;}

int [] nums = new int [len];

//at least 1 candy for each child

Arrays.fill(nums, 1);

//right side higher ranking get more candies

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

if (ratings[i] > ratings[i - 1]){nums[i] = nums[i - 1] + 1;}

}

//left side higher ranking get more candies

for (int i = len - 2; i >= 0; i--){

if (ratings[i] > ratings[i + 1]){nums[i] = Math.max(nums[i], nums[i + 1] + 1);}

}

int sum = 0;

for (int i : nums){sum += i;}

return sum;

}

}

## 165 [Compare Version Numbers](https://leetcode.com/problems/compare-version-numbers/)

// 0.1 < 1.1 < 1.2 < 13.37

public class Solution {

public int compareVersion(String version1, String version2) {

String [] strs1 = version1.split("\\.");

String [] strs2 = version2.split("\\.");

int len = Math.max(strs1.length, strs2.length);

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

Integer a1 = i < strs1.length ? Integer.parseInt(strs1[i]) : 0;

Integer a2 = i < strs2.length ? Integer.parseInt(strs2[i]) : 0;

if (a1.compareTo(a2) != 0){

return a1.compareTo(a2);

}

}

return 0;

}

}

## 168 [Excel Sheet Column Title](https://leetcode.com/problems/excel-sheet-column-title)

public class Solution {

public String convertToTitle(int n) {

StringBuilder sb = new StringBuilder();

while (n != 0){

char c = (char)((n - 1)%26 + 65);

System.out.println(c);

sb.append(c);

n = (n - 1)/26;

}

return sb.reverse().toString();

}

}

## 171 [Excel Sheet Column Number](https://leetcode.com/problems/excel-sheet-column-number)

public class Solution {

public int titleToNumber(String s) {

int len = s.length();

int sum = 0;

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

sum = 26 \* sum + s.charAt(i) - 'A' + 1;

}

return sum;

}

}

## 172 [Factorial Trailing Zeroes](https://leetcode.com/problems/factorial-trailing-zeroes)

public class Solution {

public int trailingZeroes(int n) {

if (n == 0){return 0;}

else {

return n/5 + trailingZeroes(n/5);

}

}

}

## 179 Largest Number

public class Solution {

public String largestNumber(int[] nums) {

if (nums.length == 0){return "";}

String [] strs = new String [nums.length];

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

strs[i] = String.valueOf(nums[i]);

}

Comparator<String> comp = new Comparator<String> (){

@Override

public int compare (String s1, String s2){

return (s2 + s1).compareTo(s1 + s2);

}

};

Arrays.sort(strs,comp);

if (strs[0].equals("0") ) {return "0";}

StringBuilder sb = new StringBuilder ();

for (String s : strs){sb.append(s);}

return sb.toString();

}

}

## 187 [Repeated DNA Sequences](https://leetcode.com/problems/repeated-dna-sequences)

public class Solution {

public List<String> findRepeatedDnaSequences(String s) {

List<String> rev = new ArrayList<>();

if (s.length() < 10){

return rev;

}

HashSet <String> set = new HashSet <>();

HashSet <String> newset = new HashSet <>();

for (int i= 0; i <= s.length () - 10; i++){

String sub = s.substring(i,i+10);

if (!set.add(sub)){

newset.add(sub);

}

}

rev.addAll(newset);

return rev;

}

}

## 202 [Happy Number](https://leetcode.com/problems/happy-number)

public class Solution {

public boolean isHappy(int n) {

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

while (!set.contains(n)){

set.add(n);

n = help(n);

if (n == 1){

return true;

}

}

return false;

}

public int help (int n){

int sum = 0;

while (n > 9){

sum += (n%10) \* (n%10);

n = n/10;

}

sum += n \* n;

return sum;

}

}

## 204 Count Primes

public class Solution {

public int countPrimes(int n) {

int count = 0;

boolean [] nums = new boolean [n];

for (int i = 2 ; i < n ; i++){

if (nums[i] == false){

count++;

int j = 2;

while ( i\*j <n){

nums[i\*j] = true;

j++;

}

}

}

return count;

}

}

## 205 Isomorphic Strings

public class Solution {

public boolean isIsomorphic(String s, String t) {

if (s.length() != t.length()){return false;}

int len = s.length();

HashMap <Character, Integer> maps = new HashMap<>();

HashMap <Character, Integer> mapt = new HashMap<>();

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

if (!maps.containsKey(s.charAt(i))){

if (mapt.containsKey(t.charAt(i))){return false;}

else {

maps.put(s.charAt(i), i);

mapt.put(t.charAt(i), i);

}

}

else {

if (!mapt.containsKey(t.charAt(i))){return false;}

else {

if (maps.get(s.charAt(i)) != mapt.get(t.charAt(i))){return false;}

}

}

}

return true;

}

}

## 207. Course Schedule

//Toplogical sort + BFS

//O(n^2)

public class Solution {

public boolean canFinish(int numCourses, int[][] prerequisites) {

if (numCourses == 0) return true;

int [] indegree = new int [numCourses];

int [][] matrix = new int [numCourses][numCourses];

for (int [] n : prerequisites){

int cur = n[0], prev = n [1];

//consider the duplicate case

if (matrix[prev][cur] == 0){

matrix[prev][cur] = 1;

indegree[cur]++;

}

}

//BFS

Queue <Integer> queue = new LinkedList<>();

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

if (indegree[i] == 0) queue.offer(i);

}

int count = 0;

while(!queue.isEmpty()){

int prev = queue.poll();

count++;

//search line

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

if (matrix[prev][i] == 1){

if (--indegree[i] == 0) queue.offer(i);

}

}

}

return count == numCourses;

}

}

## 210. Course Schedule II

public class Solution {

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

if (numCourses == 0) return new int [0];

int row = numCourses;

int [] indegree = new int [row];

int [][] matrix = new int [row][row];

for (int [] n : prerequisites){

int cur = n[0], prev = n[1];

if (matrix[prev][cur] == 0){matrix[prev][cur] = 1; indegree[cur]++;}

}

Queue<Integer> queue = new LinkedList<>();

for (int i = 0; i < row; i++){if (indegree[i] == 0) queue.offer(i);}

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

while(!queue.isEmpty()){

int prev = queue.poll();

rev.add(prev);

for (int cur = 0; cur < row; cur++){

if (matrix[prev][cur] == 1){if (--indegree[cur] == 0) queue.offer(cur);}

}

}

if (rev.size() != row) {return new int [0];}

// if use rev.toArray(), must be (Integer [] nums)

int [] nums = new int [rev.size()];

for (int i = 0 ; i < rev.size(); i++){nums[i] = rev.get(i);}

return nums;

}

}

## 217 Contains Duplicate

public class Solution {

public boolean containsDuplicate(int[] nums) {

if (nums.length <= 1 ){

return false;

}

HashSet set = new HashSet ();

for (int i : nums){

set.add(i);

}

return set.size() != nums.length;

}

}

## 219 Contains Duplicate II

public class Solution {

public boolean containsNearbyDuplicate(int[] nums, int k) {

//case 1

if (nums.length <=1){

return false;

}

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

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

if (map.containsKey(nums[i])){

if (i - map.get(nums[i]) <= k)

return true;

}

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

}

return false;

}

}

## Contains Duplicate III

//Budget sort O(n)

public class Solution {

public boolean containsNearbyAlmostDuplicate(int[] nums, int k, int t) {

if (k <= 0 || t < 0) return false;

long length = (long) t + 1;

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

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

long id = getId((long)nums[i], length);

if (map.containsKey(id)) return true;

else if (map.containsKey(id - 1) && Math.abs(nums[i] - map.get(id - 1)) < length) return true;

else if (map.containsKey(id + 1) && Math.abs(nums[i] - map.get(id + 1)) < length) return true;

map.put(id, (long)nums[i]);

if (i >= k) map.remove(getId((long)nums[i - k], length));//remove budget

}

return false;

}

public long getId (long number, long length){

return number < 0 ? (number + 1)/length - 1 : number /length;

}

}

## 223 Rectangle Area

public class Solution {

public int computeArea(int A, int B, int C, int D, int E, int F, int G, int H) {

int Area1 = (C - A) \* (D - B);

int Area2 = (G - E) \* (H - F);

int left = Math.max(A, E);

int right = Math.min(C, G);

int bottom = Math.max(B, F);

int up = Math.min(D, H);

int interSection = 0;

if (left <= right && bottom <= up){

interSection = (right - left) \* (up - bottom);

}

return Area1 + Area2 - interSection;

}

}

## 231 [Power of Two](https://leetcode.com/problems/power-of-two/)

import java.lang.Math.\*;

public class Solution {

public boolean isPowerOfTwo(int n) {

if ( n <= 0){

return false;

}

if ( n == 1){

return true;

}

if (n % 2 != 0){

return false;

}

while ( (n / 2) !=1 ){

if ((n / 2) % 2 !=0 ){

return false;

}

n = n/2;

}

return true;

}

}

## 241. Different Ways to Add Parentheses

public class Solution {

public List<Integer> diffWaysToCompute(String input) {

int len = input.length();

if (len == 0){return new ArrayList<Integer>();}

else {

return help(input, 0, len - 1);

}

}

public List<Integer> help (String input, int start, int end){

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

if (start > end){rev.add(0);return rev;}

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

if (!Character.isDigit(input.charAt(i))){

List<Integer> leftChild = help (input, start, i - 1);

List<Integer> rightChild = help (input, i + 1, end);

for (int l : leftChild){

for (int r : rightChild){

if (input.charAt(i) == '+'){rev.add(l + r);}

else if (input.charAt(i) == '-'){rev.add(l - r);}

else {rev.add(l \* r);}

}

}

}

}

if (rev.size() == 0){rev.add(Integer.parseInt(input.substring(start, end + 1)));}

return rev;

}

}

## 258 [Add Digits](https://leetcode.com/problems/add-digits)

//digit root problem

public class Solution {

public int addDigits(int num) {

return 1 + (num - 1)%9;

}

}

## 274. H-Index

//O(n) RT and Space

public class Solution {

public int hIndex(int[] citations) {

int len = citations.length;

int [] nums = new int [len + 1];

for (int i : citations){

if (i >= len) {nums[len]++;}

else {nums[i]++;}

}

int count = 0;

for (int i = len; i > 0; i--){

count += nums[i];

if (count >= i) return i;

}

return 0;

}

}

## 275. H-Index II

public class Solution {

public int hIndex(int[] citations) {

int len = citations.length;

if (len == 0) return 0;

int low = 0, high = len - 1;

int max = 0;

while (low <= high){

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

if (citations[mid] >= len - mid){

max = Math.max(max, len - mid);

high = mid - 1;}

else {

low = mid + 1;

}

}

return max;

}

}

## 283 [Move Zeroes](https://leetcode.com/problems/move-zeroes/)

public class Solution {

public void moveZeroes(int[] nums) {

if (nums == null) {

return ;

}

int count = 0;

for (int i : nums){

if (i != 0) {

nums[count] = i;

count++;

}

}

while (count <= nums.length -1){

nums[count++] =0;

}

}

}

## 290 Word Pattern

public class Solution {

public boolean wordPattern(String pattern, String str) {

//pattern = "abba", str = "dog dog dog dog" should return false.

String [] words = str.split (" ");

System.out.println("1 "+ pattern.length());

System.out.println("2 "+ words.length);

if (pattern.length() != words.length){

return false;

}

if (pattern.length () == 1){

return true;

}

int count =0;

HashMap <Character,String> map = new HashMap<>();

map.put(pattern.charAt(0),words[0]);

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

if (map.containsKey(pattern.charAt(i))){

if (!map.get(pattern.charAt(i)).equals(words[i])){

count ++;

System.out.println(map.get(pattern.charAt(i)).equals(words[i]));

System.out.println(map.get(pattern.charAt(i)));

System.out.println(words[i]);

return false;

}

}

else {

if (map.containsValue(words[i])){

return false;

}

map.put(pattern.charAt(i),words[i]);

}

}

return true;

}

}

## 292 Nim Game

public class Solution {

public boolean canWinNim(int n) {

return n % 4 != 0;

}

}

## 299 [Bulls and Cows](https://leetcode.com/problems/bulls-and-cows/)

public class Solution {

public String getHint(String secret, String guess) {

//igore secret and guess have different lengh

int len = secret.length();

int bull = 0, cow = 0;

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

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

char c = secret.charAt(i);

map.put(c, map.getOrDefault(c, 0) + 1);

}

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

char c = guess.charAt(i);

if (secret.charAt(i) == c){bull++;}

if (map.containsKey(c)){

if (map.get(c) > 0){map.put(c, map.get(c) - 1); cow++;}

}

}

cow -= bull;

return bull + "A" + cow + "B";

}

}

## 326 Power of Three

public class Solution {

public boolean isPowerOfThree(int n) {

if (n==1) {

return true;

}

if (n <= 0 || (n % 2) == 0 || (n % 3) !=0) {

return false;

}

while ( (n /3) != 1 ){

if ((n /3) % 3 !=0) {

return false;

}

n = n/3;

System.out.println (n);

}

if ( (n % 3) !=0) {

return false;

}

return true;

}

}

## 342 Power of Four

public class Solution {

public boolean isPowerOfFour(int num) {

if (num == 1){

return true;

}

if ( num <=0 || num % 4 !=0) {

return false;

}

return isPowerOfFour (num/4);

}

}

## 374 Guess Number Higher or Lower

/\* The guess API is defined in the parent class GuessGame.

@param num, your guess

@return -1 if my number is lower, 1 if my number is higher, otherwise return 0

int guess(int num); \*/

public class Solution extends GuessGame {

public int guessNumber(int n) {

int start = 1 ;

int end = n;

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

while (start <= end) {

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

if (guess(mid) == 0){

break;

}

else if (guess(mid) == 1){

start = mid + 1;

}

else {

end = mid -1;

}

}

return mid;

}

}

## 383 Ransom Note

public class Solution {

public boolean canConstruct(String ransomNote, String magazine) {

int lenR = ransomNote.length();

int lenM = magazine.length();

if ( lenR == 0){

return true;

}

if ( lenR > lenM){

return false;

}

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

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

map.put(magazine.charAt(i),map.getOrDefault(magazine.charAt(i),0) + 1);

}

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

if (map.get(ransomNote.charAt(i)) == null || map.get(ransomNote.charAt(i)) == 0){

return false;

}

map.put(ransomNote.charAt(i),map.get(ransomNote.charAt(i)) - 1);

}

return true;

// runing time O(Math.max(lenR,lenM))

}

}

## 389 Find the Difference

public class Solution {

public char findTheDifference(String s, String t) {

int sum1=0;

int sum2=0;

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

sum1 += (int)s.charAt(i);

}

for (int j=0;j<t.length();j++){

sum2 += (int)t.charAt(j);

}

return ((char)(sum2-sum1));

}

}

## 400 [Nth Digit](https://leetcode.com/problems/nth-digit)

public class Solution {

public int findNthDigit(int n) {

int start = 1;

long len = 9;

int count = 1;

if (n < 10) {return n;}

//find position

while(n > len \* count){

n -= len \* count;

start \*= 10;

count++;

len \*= 10;

}

System.out.println(n);

int num = n / count;

int digit = n % count;

if (digit == 0){

return String.valueOf(start + num - 1).charAt(count - 1) - '0';

}

else {

return String.valueOf(start + num).charAt(digit - 1) - '0';

}

}

}

## 412 Fizz Buzz

public class Solution {

public List<String> fizzBuzz(int n) {

List <String > l = new ArrayList ();

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

String s ="";

if (i%3==0){

s += "Fizz";

}

if ( i%5 ==0){

s += "Buzz";

}

if (i%3!=0 && i%5 !=0)

{

s = new Integer(i).toString();

}

l.add(s);

}

return l;

}

}

## 414 Third Maximum Number

public class Solution {

public int thirdMax(int[] nums) {

int max1 = nums[0];

for (int i0 : nums){

if (i0 < max1){

max1 = i0;

}

}

int max2 = max1;

int max3 = max1;

for (int i : nums){

if (i > max1){

max1 = i;

}

}

System.out.println(max1);

for (int j : nums){

if (j < max1 && j > max2){

max2 = j;

}

}

System.out.println(max2);

for (int k : nums){

if (k < max2 && k > max3){

max3 = k;

}

}

System.out.println(max3);

if (max1 > max2 && max2 > max3){

return max3;

}

return max1;

}

}

## 441 [Arranging Coins](https://leetcode.com/problems/arranging-coins)

public class Solution {

public int arrangeCoins(int n) {

int sum = 0 ;

if (n ==0){

return 0;

}

long m = (long) n;

return binarySearch (m,1,m);

}

public int binarySearch (long n, long start, long end){

while (start <= end){

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

long sum0 = (mid + 1 ) \* mid /2;

long sum1 = sum0 + mid + 1;

if (sum0 <= n && sum1 > n){

return (int) mid;

}

else if (sum1 <= n){

start = mid +1;

}

else{

end = mid -1;

}

}

return -1;

}

}

## 447 Number of Boomerangs

public class Solution {

public int numberOfBoomerangs(int[][] points) {

int row = points.length;

if (row == 0){return 0;}

int col = points[0].length;

if (col == 0){return 0;}

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

int count = 0;

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

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

if (i == j){continue;}

else {

int d = distance(points,i,j);

map.put(d, map.getOrDefault(d,0) + 1);

}

}

for (int val : map.values()){

count += val \* (val - 1);

}

map.clear();

}

return count;

}

public int distance (int [][] points, int i, int j){

int dx = points[i][0] - points[j][0];

int dy = points[i][1] - points[j][1];

return dx \* dx + dy \* dy;

}

}

## 455 Assign Cookies

public class Solution {

public int findContentChildren(int[] g, int[] s) {

if (g == null || s == null) {

return 0;

}

Arrays.sort (g);

Arrays.sort (s);

int cnt =0;

int j = 0;

for (int i : g){

while (j < s.length){

if (i <= s[j]){

cnt++;

break;

}

j++;

}

j++;

}

return cnt;

}

}

## 476 [Number Complement](https://leetcode.com/problems/number-complement/)

import java.lang.Math.\*;

public class Solution {

public int findComplement(int nums) {

int a = 0;

int b = nums;

while (nums !=0) {

nums >>=1;

a <<=1;

a +=1;

}

return (a ^ b);

}

}

## 485 Max Consecutive Ones

import java.lang.Math.\*;

public class Solution {

public int findMaxConsecutiveOnes(int[] nums) {

int count =0;

int max = 0;

if (nums == null){

return 0;

}

for (int num : nums){

if (num == 1){

count++;

max = Math.max(max,count);

}

else {

count =0;

}

}

return max;

}

}

## 486 Predict the Winner

public class Solution {

public boolean PredictTheWinner(int[] nums) {

int len = nums.length;

if (len == 1) {return true;}

if (len == 2) {return Math.max(nums[0],nums[1]) >= Math.min(nums[0],nums[1]);}

int [][] dpf = new int [len][len];

int [][] dps = new int [len][len];

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

dpf[i][i] = nums[i];

dps[i][i] = 0;

}

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

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

//posiiton dp[i][i + j]

if (nums[i + j] + dps[i][ i + j - 1] > nums[i] + dps[i + 1][i + j]){

dpf[i][i + j] = nums[i + j] + dps[i][ i + j - 1];

dps[i][i + j] = dpf[i][ i + j - 1];

}

else {

dpf[i][i + j] = nums[i] + dps[i + 1][i + j];

dps[i][i + j] = dpf[i + 1][ i + j];

}

}

}

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

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

//System.out.println("i " + i + " j " + j + " "+ dpf[i][j] + " "+ dps[i][j]);

}

}

return dpf[0][len - 1] >= dps[0][len - 1];

}

}

## 492 Construct the Rectangle

import java.lang.Math.\*;

public class Solution {

public int[] constructRectangle(int area) {

int low = (int) Math.sqrt(area);

if (area == low \* low){

return new int [] {low, low};

}

else {

while (low >=1){

if (area == low \* (area / low) ){

return new int [] {area / low, low};

}

else {

low-- ;

}

}

return new int [] {0,0};

}

}

}

## 495. Teemo Attacking

public class Solution {

public int findPoisonedDuration(int[] timeSeries, int duration) {

int len = timeSeries.length;

if (len == 0 || duration == 0) return 0;

if (len == 1) return duration;

int stop = timeSeries[0] + duration;

int count = duration;

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

if (stop <= timeSeries[i]) {

count += duration; stop = timeSeries[i] + duration;

}

else {

int newStop = timeSeries[i] + duration;

count += newStop - stop;

stop = newStop;

}

}

return count;

}

}

## 500 [Keyboard Row](https://leetcode.com/problems/keyboard-row)

public class Solution {

public String[] findWords(String[] words) {

List<Character> r1 = new ArrayList<>(Arrays.asList('Q','W','E','R','T','Y','U','I','O','P','q','w','e','r','t','y','u','i','o','p'));

List<Character> r2 = new ArrayList<>(Arrays.asList('A','S','D','F','G','H','J','K','L','a','s','d','f','g','h','j','k','l'));

List<Character> r3 = new ArrayList<>(Arrays.asList('Z','X','C','V','B','N','M','z','x','c','v','b','n','m'));

HashSet <Character> set1 = new HashSet<>(); set1.addAll(r1);

HashSet <Character> set2 = new HashSet<>(); set2.addAll(r2);

HashSet <Character> set3 = new HashSet<>(); set3.addAll(r3);

List<String> rev = new ArrayList<>();

if (words.length == 0){

return words;

}

for (String k : rev ){

System.out.println("k is" + k);

}

for (String word : words){

//System.out.println(word);

System.out.println(isInset(set1,word));

System.out.println(isInset(set2,word));

if (isInset(set1,word) || isInset(set2,word) || isInset(set3,word)){

//

//

// System.out.println(isInset(set3,word));

rev.add(word);

}

}

for (String word : rev ){

System.out.println("word is " + word);

}

return rev.toArray(new String[rev.size()]);

}

public boolean isInset (HashSet <Character> set, String word) {

int len = word.length();

if (len == 0){

return true;

}

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

if (!set.contains(word.charAt(i))){

return false;

}

}

return true;

}

}

## 504 Base 7

public class Solution {

public String convertToBase7(int num) {

if (num == 0) {return "0";}

StringBuilder sb = new StringBuilder();

boolean flag = false;

flag = num < 0 ? true : false;

while (Math.abs(num) > 0){

sb.append(Math.abs(num % 7));

num = Math.abs(num / 7);

}

if (flag){sb.append("-");}

return sb.reverse().toString();

}

}

## 507 Perfect Number

import java.lang.Math.\*;

public class Solution {

public boolean checkPerfectNumber(int num) {

int sum = 1;

int i = 2;

if (num == 1){

return false;

}

while (i <= Math.sqrt(num)){

if (num % i == 0){

sum += i + num / i;

}

i++;

}

return (sum == num);

}

}

## 520 [Detect Capital](https://leetcode.com/problems/detect-capital)

public class Solution {

public boolean detectCapitalUse(String word) {

if (word.length() <=1){

return true;

}

boolean b = true;

char [] charArray = word.toCharArray();

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

if (isLowerCase(charArray[0])){

b = b && isLowerCase(charArray[i]);

}

else if (isUpperCase(charArray[0])&&isLowerCase(charArray[1])){

if (i>=1){

b = b && isLowerCase(charArray[i]);

}

}

else if (isUpperCase(charArray[0])&&isUpperCase(charArray[1])){

b = b && isUpperCase(charArray[i]);

}

else {

b=false;

}

}

return b;

}

public boolean isLowerCase (char c){

return (c >= 'a' && c <= 'z');

}

public boolean isUpperCase (char c){

return (c >= 'A' && c <= 'Z');

}

}

## 526 Beautiful Arrangement

public class Solution {

int count = 0;

int step = 0;

public int countArrangement(int N) {

int [] used = new int [N + 1];

int [] nums = new int [1];

helpFun (N, 1, used, nums);

// System.out.println("count is " + count + "; " + "nums is " + nums[0]);

// System.out.println("Final Step Arrays is " + Arrays.toString(used));

return nums[0];

}

public void helpFun (int N, int pos, int [] used, int [] nums){

if (pos > N){count++; nums[0]++; return; }

else {

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

if (used[i] == 0 && (i % pos == 0 || pos % i == 0)){

used[i] = 1;

// System.out.println("Step nums is " + nums[0] + " Step " + (++step));

// System.out.println("Step Arrays is " + Arrays.toString(used));

helpFun(N, pos + 1, used, nums);

used[i] = 0;

}

}

}

}

}

## 539 Minimum Time Difference

public class Solution {

public int findMinDifference(List<String> timePoints) {

int len = timePoints.size();

int [] nums = new int [len];

// compare to the mid-night;

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

String s = timePoints.get(i);

int sum1 = (s.charAt(0) - '0') \* 10 \* 60 + (s.charAt(1) - '0') \* 60;

int sum2 = + (s.charAt(3) - '0') \* 10 + (s.charAt(4) - '0');

nums[i] = 24\*60 - sum1 - sum2;

}

Arrays.sort(nums);

int min = Integer.MAX\_VALUE;

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

min = Math.min(min,nums[i]-nums[i-1]);

}

// first element and last element comparing:

min = Math.min(min,24\*60-nums[len-1] + nums[0]);

return min;

}

}

## 551 [Student Attendance Record I](https://leetcode.com/problems/student-attendance-record-i/)

import java.lang.Math.\*;

public class Solution {

public boolean checkRecord(String s) {

if (s.length() <= 1){

return true;

}

int noOfA = 0;

int noOfContL = 0;

int maxContL = 0;

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

if (s.charAt(i) == 'A'){

noOfA++;

}

else if (i+1<s.length() && s.charAt(i) == 'L' && s.charAt(i+1) == 'L'){

noOfContL++;

maxContL = Math.max (noOfContL,maxContL);

}

else {

noOfContL = 0;

}

}

System.out.println (maxContL);

if( (noOfA >=2) || (maxContL >=2) ){

return false;

}

return true;

}

}

## 553 Optimal Division

public class Solution {

public String optimalDivision(int[] nums) {

int len = nums.length;

if (len == 0){

return "";

}

else if (len == 1){

return ""+ nums[0];

}

else if (len == 2){

return nums[0] + "/" + nums[1];

}

else {

String s = "(";

for (int i = 1; i < len -1; i++){

s += nums[i] + "/";

}

s = nums[0] + "/" + s + nums[len -1] + ")";

return s;

}

}

}

## 554 Brick Wall

//need consider [[1],[1],[1]] -> 3

public class Solution {

public int leastBricks(List<List<Integer>> wall) {

if (wall.size() == 0){return 0;}

if (wall.get(0).size() == 0) {return 0;}

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

for (List<Integer> w : wall){

int sum = 0;

for (int i = 0; i < w.size() - 1; i++){

sum += w.get(i);

map.put(sum, map.getOrDefault(sum, 0) + 1);

}

}

int minStick = Integer.MAX\_VALUE;

for (int i : map.values()){

minStick = Math.min(minStick, wall.size() - i);

}

return minStick == Integer.MAX\_VALUE ? wall.size() : minStick;

}

}

## 575 Distribute Candies

public class Solution {

public int distributeCandies(int[] candies) {

int len = candies.length;

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

for (int i : candies){set.add(i);}

return set.size() >= len / 2? len / 2 : set.size();

}

}

## 594 Longest Harmonious Subsequence

//[1,1,1,1] -> 0

public class Solution {

public int findLHS(int[] nums) {

int len = nums.length;

if (len == 0){return 0;}

int maxLength = 0;

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

for (int i : nums){

map.put(i, map.getOrDefault(i, 0) + 1);

}

for (int i : map.keySet()){

if (!map.containsKey( i - 1) && !map.containsKey( i + 1)) {continue;}

int lower = (map.containsKey( i - 1) ? map.get(i - 1) : 0) + map.get(i);

int higher = (map.containsKey( i + 1) ? map.get(i + 1) : 0) + map.get(i);

maxLength = Math.max(Math.max(lower, higher), maxLength);

}

return maxLength;

}

}

## 605. Can Place Flowers

//[1] 0 this case need to be considered

public class Solution {

public boolean canPlaceFlowers(int[] flowerbed, int n) {

int len = flowerbed.length;

if (len < n) return false;

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

boolean left = i - 1 < 0 ? true : flowerbed[i - 1] == 0;

boolean right = i + 1 > len - 1 ? true : flowerbed[i + 1] == 0;

if (right && left && flowerbed[i] == 0) {flowerbed[i] = 1;n--;}

if (n <= 0) return true;

}

return false;

}

}

## Minimum Genetic Mutation

public class Solution {

public int minMutation(String start, String end, String[] bank) {

if (start == end){return 0;}

Queue<String> queue = new LinkedList<>();

HashSet <String> visited = new HashSet<>();

HashSet <String> check = new HashSet<>();

for (String s : bank){check.add(s);}

if (!check.contains(end)){return -1;}

// BFS

queue.offer(start);

visited.add(start);

int cap = 0;

int level = 0;

char [] list = {'A', 'C', 'G', 'T'};

while (!queue.isEmpty()){

cap = queue.size();

//level travel

while (cap-- > 0){

String test = queue.poll();

if (test.equals(end)){return level;}

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

char [] cs = test.toCharArray();

for (char c : list){

cs[i] = c;

String t = new String (cs);

if (!visited.contains(t) && check.contains(t)){

queue.offer(t);

visited.add(t);

}

}

}

}

level++;

}

return -1;

}

}

面试题目:

## 1. 乱序，按频率大小输出，相同时，按最早出现顺序输出

//给一个乱序Array,[2,5,1,3,2,5,2,1,6] - >[2,5,1,3,6]

// 按照出现的评论， 如果次数相同，就按先出现的

package Array;

import java.util.\*;

//给一个乱序Array,[2,5,1,3,2,5,2,1,6] - >[2,5,1,3,6]

// 按照出现的评论， 如果次数相同，就按先出现的

//bucket sort O(n)

public class FreqAppear {

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

public void bucketSort (int [] nums){

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

int len = nums.length;

int [] count = new int [len];

List<Integer> [] bucket = new List [len + 1];

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

if (!map.containsKey(nums[i])) {map.put(nums[i], i);}

count[map.get(nums[i])]++;

}

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

//System.out.println(count[i]);

if (count[i] != 0 ){

if ( bucket[count[i]] == null){

bucket[count[i]] = new ArrayList<>();

}

bucket[count[i]].add(nums[i]);

}

}

for (int i = len; i >= 0; i--){

if (bucket[i] != null) {

rev.addAll(bucket[i]);

}

}

}

}