Permutations of a string

public ArrayList<ArrayList<Integer>> permute(int[] num) {

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

    permute(num, 0, result);

    return result;

}

void permute(int[] num, int start, ArrayList<ArrayList<Integer>> result) {

    if (start >= num.length) {

        ArrayList<Integer> item = convertArrayToList(num);

        result.add(item);

    }

    for (int j = start; j <= num.length - 1; j++) {

        swap(num, start, j);

        permute(num, start + 1, result);

        swap(num, start, j);

    }

}

private ArrayList<Integer> convertArrayToList(int[] num) {

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

    for (int h = 0; h < num.length; h++) {

        item.add(num[h]);

    }

    return item;

}

private void swap(int[] a, int i, int j) {

    int temp = a[i];

    a[i] = a[j];

    a[j] = temp;

}

Longest Substring

public class Solution {

public int lengthOfLongestSubstring(String s) {

int n = s.length();

Set<Character> set = new HashSet<>();

int ans = 0, i = 0, j = 0;

while (i < n && j < n) {

// try to extend the range [i, j]

if (!set.contains(s.charAt(j))){

set.add(s.charAt(j++));

ans = Math.max(ans, j - i);

}

else {

set.remove(s.charAt(i++));

}

}

return ans;

}

}

Given a sorted circularly linked list of Nodes that store integers and a new Node, insert the new Node into the correct position. (Duplicates allowed)

How many times will a robot need to retrieve bins if it is given an array of bin ID's and it can only hold N bins at a time? When the robot is already holding N bins, it will return the least recently retrieved bin and store the new bin.

Given an ArrayList of Nodes, with each Node having an ID and a parent ID, determine whether the List is given in preorder.

find a pattern 'p' in a string 's'

Priority queues and how you implement it.

2 a) Stacks  
2 b) Tree Traversal

Division without divide operator

public int divide(int dividend, int divisor) {

//handle special cases

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

if(divisor==-1 && dividend == Integer.MIN\_VALUE)

return Integer.MAX\_VALUE;

//get positive values

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

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

int result = 0;

while(pDividend>=pDivisor){

//calculate number of left shifts

int numShift = 0;

while(pDividend>=(pDivisor<<numShift)){

numShift++;

}

//dividend minus the largest shifted divisor

result += 1<<(numShift-1);

pDividend -= (pDivisor<<(numShift-1));

}

if((dividend>0 && divisor>0) || (dividend<0 && divisor<0)){

return result;

}else{

return -result;

}

}

find-all-anagrams-in-a-string

public List<Integer> findAnagrams(String s, String p) {

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

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

int[] hash = new int[256]; //character hash

//record each character in p to hash

for (char c : p.toCharArray()) {

hash[c]++;

}

//two points, initialize count to p's length

int left = 0, right = 0, count = p.length();

while (right < s.length()) {

//move right everytime, if the character exists in p's hash, decrease the count

//current hash value >= 1 means the character is existing in p

if (hash[s.charAt(right)] >= 1) {

count--;

}

hash[s.charAt(right)]--;

right++;

//when the count is down to 0, means we found the right anagram

//then add window's left to result list

if (count == 0) {

list.add(left);

}

//if we find the window's size equals to p, then we have to move left (narrow the window) to find the new match window

//++ to reset the hash because we kicked out the left

//only increase the count if the character is in p

//the count >= 0 indicate it was original in the hash, cuz it won't go below 0

if (right - left == p.length()) {

if (hash[s.charAt(left)] >= 0) {

count++;

}

hash[s.charAt(left)]++;

left++;

}

}

return list;

}

find common ancestor for a n-ary tree

public Node LCA(Node root, Node a, Node b) {

if(root == null) return null;

if(a == root || b == root) return root;

Node ancestor = null;

for(Node child: root.children) {

Node found = LCA(child, a, b);

if(found != null) {

if(ancestor == null) ancestor = found;

else return root; // if two nodes were found in different subtrees of root, then root is the closest common ancestor

}

if(ancestor != null) return ancestor; // only 1 node between a and b is found in subtree of root. root is not a common ancestor, return the node found

return null; //the subtree has neither of a and b

}

Write breadth-first search in a matrix

writing a program to tell if a binary tree is a symmetric tree

public boolean isSymmetric(TreeNode root) {

return isMirror(root, root);

}

public boolean isMirror(TreeNode t1, TreeNode t2) {

if (t1 == null && t2 == null) return true;

if (t1 == null || t2 == null) return false;

return (t1.val == t2.val)

&& isMirror(t1.right, t2.left)

&& isMirror(t1.left, t2.right);

}

Rotate a matrix and reverse the second half linked list.

List down the paths in a binary tree that sum up to the given sum

public boolean hasPathSum(TreeNode root, int sum) {

    if (root == null)

        return false;

    if (root.val == sum && (root.left == null && root.right == null))

        return true;

    return hasPathSum(root.left, sum - root.val)

            || hasPathSum(root.right, sum - root.val);

}

public class Solution {

public boolean hasPathSum(TreeNode root, int sum) {

if(root == null) return false;

LinkedList<TreeNode> nodes = new LinkedList<TreeNode>();

LinkedList<Integer> values = new LinkedList<Integer>();

nodes.add(root);

values.add(root.val);

while(!nodes.isEmpty()){

TreeNode curr = nodes.poll();

int sumValue = values.poll();

if(curr.left == null && curr.right == null && sumValue==sum){

return true;

}

if(curr.left != null){

nodes.add(curr.left);

values.add(sumValue+curr.left.val);

}

if(curr.right != null){

nodes.add(curr.right);

values.add(sumValue+curr.right.val);

}

}

return false;

}

}

* Given a list of weighted edges between nodes, find the minimum cost spanning tree
* Given the upper left and lower right coordinates of two rectangles, determine if they overlap
* fizz buzz and variations on it.  
  Implementing queue with stack.
* private int front;
* public void push(int x) {
* if (s1.empty())
* front = x;
* while (!s1.isEmpty())
* s2.push(s1.pop());
* s2.push(x);
* while (!s2.isEmpty())
* s1.push(s2.pop());
* }
* public void pop() {
* s1.pop();
* if (!s1.empty())
* front = s1.peek();
* }
* public boolean empty() {
* return s1.isEmpty();
* }
* public int peek() {
* return front;
* }
* //using two stacks
* private Stack<Integer> s1 = new Stack<>();
* private Stack<Integer> s2 = new Stack<>();
* // Push element x to the back of queue.
* public void push(int x) {
* if (s1.empty())
* front = x;
* s1.push(x);
* }
* public void pop() {
* if (s2.isEmpty()) {
* while (!s1.isEmpty())
* s2.push(s1.pop());
* }
* s2.pop();
* }
* public boolean empty() {
* return s1.isEmpty() && s2.isEmpty();
* }
* public int peek() {
* if (!s2.isEmpty()) {
* return s2.peek();
* }
* return front;
* }
* finding intersection of 2 lists.
* Find the longest palindromic substring.

public String longestPalindrome(String s) {

int start = 0, end = 0;

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

int len1 = expandAroundCenter(s, i, i);

int len2 = expandAroundCenter(s, i, i + 1);

int len = Math.max(len1, len2);

if (len > end - start) {

start = i - (len - 1) / 2;

end = i + len / 2;

}

}

return s.substring(start, end + 1);

}

private int expandAroundCenter(String s, int left, int right) {

int L = left, R = right;

while (L >= 0 && R < s.length() && s.charAt(L) == s.charAt(R)) {

L--;

R++;

}

return R - L - 1;

}  
- Return the k-closest points to the center of a cartesian plane given an array of coordinates.

* public static class Point implements Comparable<Point> {
* public double x;
* public double y;
* public Point(final double x, final double y) {
* this.x = x;
* this.y = y;
* }
* public double getDist(){
* return x\*x+y\*y;
* }
* @Override
* public int compareTo(Point o) {
* int c = Double.compare(getDist(), o.getDist());
* if(c == 0){
* c = Double.compare(x, o.x);
* if(c == 0){
* c = Double.compare(y, o.y);
* }
* }
* return c;
* }
* @Override
* public String toString() {
* return "(" + x + "," + y + ")";
* }
* }
* public static Point[] closestk(final Point points[], final int k) {
* //max heap
* final PriorityQueue<Point> kClosest = new PriorityQueue<>(k, Collections.reverseOrder());
* for (int i = 0; i < points.length; i++) {
* if (kClosest.size() < k) {
* kClosest.add(points[i]);
* } else if (points[i].getDist() < kClosest.peek().getDist()) {
* kClosest.remove();
* kClosest.add(points[i]);
* }
* }
* return kClosest.toArray(new Point[k]);
* - Create and return a deep copy of a singly linked list where each node also has an additional pointer to a random node in the list.
* Given an input stream of strings, find the most frequent string.
* Given an input stream of strings, find the first, unique string.

Given an array input of N number of strings. [ cat, tac, pot, top, meow, act ]  
Return the output : [[act,cat,act], [pot,top], [meow]]

class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

if (strs.length == 0) return new ArrayList();

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

for (String s : strs) {

char[] ca = s.toCharArray();

Arrays.sort(ca);

String key = String.valueOf(ca);

if (!ans.containsKey(key)) ans.put(key, new ArrayList());

ans.get(key).add(s);

}

return new ArrayList(ans.values());

}

}

Finding intersection of two lists of strings.

* Write a function that returns true if a string of parentheses is balanced or not, such as "(())()", which would return true.  [3 Answers](https://www.glassdoor.com/Interview/Write-a-function-that-returns-true-if-a-string-of-parentheses-is-balanced-or-not-such-as-which-would-return-true-QTN_1981697.htm)
* private static boolean checkBalancedParentheses(String input){
* Stack<String> stack = new Stack<String>();
* boolean isBalanaced = false;
* for(int i=0; i < input.length(); i++){
* String str = ""+input.charAt(i);        //store characters as String
* //if opening bracket then push into stack
* if(str.equals("(") || str.equals("[") || str.equals("{")){
* stack.push(str);
* }
* //if closing bracket, pop bracket and compare if its a pair
* if(str.equals(")") || str.equals("]") || str.equals("}")){
* //if stack becomes empty in between then also its not balanced
* if(stack.isEmpty()){
* return false;
* }
* String opening = stack.peek();
* if(str.equals(")") && opening.equals("(")){
* stack.pop();
* }
* if(str.equals("]") && opening.equals("[")){
* stack.pop();
* }
* if(str.equals("}") && opening.equals("{")){
* stack.pop();
* }
* }
* }
* //if stack is empty at end, then its balanced
* if(input.length() > 0 && stack.isEmpty()){
* isBalanaced = true;
* }
* return isBalanaced;
* }
* Write a function that rotates a 2-dimensional array clockwise or counter clockwise 90 degrees depending on a given parameter, which I believe was either -1 or 1, which told you which way to rotate it. You are given the 2D array as a parameter as well.  [2 Answers](https://www.glassdoor.com/Interview/Write-a-function-that-rotates-a-2-dimensional-array-clockwise-or-counter-clockwise-90-degrees-depending-on-a-given-paramete-QTN_1981698.htm)
* for (int layer = 0; layer < arr.length/2; layer++) {
* int first = layer;
* int last = arr.length - layer - 1;
* for (int i = first; i < last; i++) {
* int offset = i - first;
* //save top
* int top = arr[first][i];
* //top -> left
* arr[first][i] = arr[last-offset][first];
* //left -> bottom
* arr[last-offset][first] = arr[last][last-offset];
* //bottom -> right
* arr[last][last-offset] = arr[i][last];
* //right -> top
* arr[i][last] = top;
* }
* }
* Given a sorted array of integers in increasing order (can contain duplicates), return the last index of a specified target integer, or return -1 otherwise. Thus 1, 2, 2, 3, 4 and the target is 2, the function should return 2.  [3 Answers](https://www.glassdoor.com/Interview/Given-a-sorted-array-of-integers-in-increasing-order-can-contain-duplicates-return-the-last-index-of-a-specified-target-QTN_1981699.htm)

Use Binary Search

* The final question was given a binary tree, find the max height of the tree. I solved this using recursion and he wanted to know how I would do it without recursion

public int findHeight(TreeNode root) {

if(root == null){

return 0;

}

int leftHeight = 0;

if(root.left != null) {

leftHeight = findHeight(root.left);

}

int rightHeight = 0;

if(root.right != null){

rightHeight = findHeight(root.right);

}

return Math.max(leftHeight+1, rightHeight+1);

}

* Design a function that reverses the second half of a singly-linked list  [Answer Question](https://www.glassdoor.com/Interview/Design-a-function-that-reverses-the-second-half-of-a-singly-linked-list-QTN_1880711.htm)
* public static ListNode reverse(ListNode start)
* {
* int counter = 0;
* ListNode node = start;
* ListNode pre = start;
* ListNode result = start;
* while (node!= null)// for count how many elements in linked list
* {
* counter += 1;
* node = node.next;
* }
* for (int i=0; i< (counter / 2) ; i++)//no matter counter is even or odd, when it divided by 2, the result is even
* {
* pre = start;
* start = start.next;
* }
* ListNode temp = null;
* ListNode preNext = null;// this variable is used to track the next val behind pre
* // for example, 2->1->3->4->5->6->7->8
* // at this moment, pre:4, start:5
* // I treated 5->6->7->8 as an independent linkedlist
* // I reversed the linkedlist
* // Finally, set the pre node's next value to the reversed linkedlist's head
* // The first half and second half have been connected together
* while (start != null)
* {
* temp = start.next;
* start.next = preNext;
* preNext = start;
* start = temp;
* }
* pre.next = preNext;
* return start;
* }
* Design a function that finds a tree path with the lowest total value.
* private static void path(Node root, ArrayList<Integer> list,int s) {
* if(root==null) {
* return;
* } else {
* list.add(root.info);
* s = s+root.info;
* }
* if ((root.left == null && root.right == null)) {
* System.out.println(list);
* if(maxSum>s) {
* maxSum = s;
* finalList = new ArrayList<>(list);
* }
* return;
* }
* path(root.left, new ArrayList<Integer>(list),s);
* path(root.right, new ArrayList<Integer>(list),s);
* }