# Goldman

class Solution {

public List<List<String>> Anagrams(String[] string\_list) {

HashMap<HashMap<Character,Integer>,ArrayList<String>> moto=new HashMap<>();

for(String str:string\_list){

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

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

char ch=str.charAt(i);

nano.put(ch,nano.getOrDefault(ch,0)+1);

}

if(moto.containsKey(nano)==false){

ArrayList<String> temp=new ArrayList<>();

temp.add(str);

moto.put(nano,temp);

}

else{

ArrayList<String> t2=moto.get(nano);

t2.add(str);

}

}

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

for(ArrayList<String> val:moto.values()){

ans.add(val);

}

return ans;

}

}

//Overlap Rectangle

class Solution {

int doOverlap(int L1[], int R1[], int L2[], int R2[]) {

if(L1[0]>R2[0]||L2[0]>R1[0]) //for x axis

return 0;

if(R1[1]>L2[1]||R2[1]>L1[1]) //for y axis

return 0;

return 1;

}

};

//count subarrays

class Solution {

public int countSubArrayProductLessThanK(long a[], long n, long k)

{

int i=0,j=0;

int ans=0;

long multi=1;

while(j<n){

multi=(long) multi \* a[j];

while(multi>=k && i<=j){

multi=(long) multi / a[i];

i++;

}

ans += j - i + 1;

j++;

}

return ans;

}

}

//Nth ugly number

class Solution {

/\* Function to get the nth ugly number\*/

long getNthUglyNo(int n) {

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

dp[1]=1;

int p2=1,p3=1,p5=1;

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

long m2=2\*dp[p2];

long m3=3\*dp[p3];

long m5=5\*dp[p5];

long min=Math.min(Math.min(m2,m3),m5);

dp[i]=min;

if(min==m2){

p2++;

}

if(min==m3){

p3++;

}

if(min==m5) p5++;

}

return dp[n];

}

}

//min subarray sum

class Solution {

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

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

int i=0,j=0;

int sum=0;

int ans=Integer.MAX\_VALUE;

while (j < nums.length) {

sum += nums[j++];

while (sum >= target) {

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

sum -= nums[i++];

}

}

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

}

}

//Given a pattern

class Solution{

static String printMinNumberForPattern(String S){

String ans = "";

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

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

st.push(i + 1);

if (S.charAt(i) == 'I') {

while (!st.empty()) {

ans += String.valueOf(st.pop());

}

}

}

st.push(S.length()+1);

while (!st.empty()) {

ans += String.valueOf(st.pop());

}

return ans;

}

}

//find first missing and repeating

class Solve {

int[] findTwoElement(int arr[], int n) {

int i=0;

int[] ans=new int[2];

while(i<arr.length){

if(arr[i]!=i+1){

int correct=arr[i]-1;

if(arr[i]!=arr[correct]){

swap(arr,i,correct);

}

else i++;

}

else i++;

}

for(int id=0;id<arr.length;id++){

if(arr[id]!=id+1){

ans[0]=arr[id];

ans[1]=id+1;

}

}

return ans;

}

static void swap(int[] arr,int first,int second){

int temp=arr[first];

arr[first]=arr[second];

arr[second]=temp;

}

}

//Question 6

class Solution {

public static String gcdOfStrings(String str1, String str2) {

if (str1.length() < str2.length()) {

return gcdOfStrings(str2, str1);

}

if (!str1.startsWith(str2)) return "";

String reminder = str1.substring(str2.length());

if (reminder.isEmpty()) {

return str2;

}

return gcdOfStrings(str2, reminder);

}

}

//decode string

class Solution

{

public int CountWays(String str)

{

if (str.length() == 0) {

return 0;

}

int n = str.length();

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

dp[0] = 1;

char[] ch=str.toCharArray();

//to decode the first character

if(ch[0]=='0') return 0;

dp[1] = 1;

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

dp[i]=0;

if (ch[i - 1] > '0')

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

if (ch[i - 2] == '1' || (ch[i - 2] == '2' && ch[i - 1] < '7')) {

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

}

}

return dp[n];

}

}

//encode string

class GfG

{

String encode(String str)

{

StringBuilder sb=new StringBuilder();

int n = str.length();

int count = 1;

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

if(str.charAt(i) == str.charAt(i-1)){

count++;

}

else{

sb.append(str.charAt(i-1));

sb.append(count);

count = 1;

}

}

sb.append(str.charAt(n-1));

sb.append(count);

return sb.toString();

}

}

//Array Pair sum division problem

class Solution {

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

if (nums.length % 2 == 1)

return false;

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

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

int remainder = (nums[i] % k);

if (!map.containsKey(remainder)) {

map.put(remainder, 0);

}

map.put(remainder, map.get(remainder) + 1);

}

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

int remainder = (nums[i] % k);

if (2 \* remainder == k) {

// if remainder is k/2 then it must have have even occurance

if (map.get(remainder) % 2 == 1)

return false;

}

if (remainder == 0) {

// if remainder is 0 then it must have have even occurance in order to create pair

if (map.get(remainder) % 2 == 1)

return false;

}

else {

if (map.get(k - remainder) != map.get(remainder))

return false;

}

}

return true;

}

}

# Amazon

## Q-1

// User function Template for Java

class Solution {

static int maxProfit(int K, int N, int A[]) {

int[][] dp=new int[K+1][N+1];

int max=0;

for(int t=1;t<=K;t++){

for(int d=1;d<A.length;d++){

max=dp[t][d-1];

for(int pd=0;pd<d;pd++){

int ptill=dp[t-1][pd];

int pro=A[d]-A[pd];

max=Math.max(max,ptill+pro);

}

dp[t][d]=max;

}

}

return max;

}

}

## Q-2

class Solution {

public int longestMountain(int[] arr) {

int ans = 0;

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

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

int left = i-1;

int right = i+1;

while(left > 0 && arr[left-1] < arr[left]) left--;

while(right <arr.length-1 && arr[right] > arr[right+1]) right++;

ans = Math.max(ans, right-left+1);

}

}

return ans;

}

}

## Q-3 & Q-6

class Solution {

static ArrayList<Integer> max\_of\_subarrays(int arr[], int n, int k) {

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

Deque<Integer> dq=new LinkedList<>();

int i=0,j=0;

int t=0;

while(j<n)

{

while(!dq.isEmpty() && arr[dq.peekLast()]<arr[j])

{

dq.pollLast();

}

dq.offerLast(j);

if(j-i+1==k)

{

res.add(arr[dq.peekFirst()]);

if(dq.peekFirst()==i)

dq.pollFirst();

i++;

}

if((j-i+1)<k)

j++;

}

return res;

}

}

## Q-7

class Solution

{

public String FirstNonRepeating(String A)

{

int[] freq = new int[26];

StringBuilder sb=new StringBuilder();

Queue<Character> q = new LinkedList<Character>();

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

char ch = A.charAt(i);

q.add(ch);

freq[ch - 'a']++;

while (!q.isEmpty()) {

if (freq[q.peek() - 'a'] > 1)

q.remove();

else {

sb.append(q.peek());

break;

}

}

if (q.isEmpty())

sb.append('#');

}

return sb.toString();

}

}

## Q-8

class Solution

{

//Function to count number of ways to reach the nth stair

//when order does not matter.

Long countWays(int m)

{

long[] res = new long[m + 1];

res[0] = 1;

res[1] = 1;

for (int i = 2; i <= m; i++)

res[i] = 1 + res[i - 2];

return res[m];

}

}

## Q-13

class Solution {

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

int i,j,x,y,n=grid.length;

int m=grid[0].length;

int ans=0,fresh=0;

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

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

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

for(j=0;j<m;j++) {

if(grid[i][j]==2) {

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

}

else if(grid[i][j]==1)

fresh++;

}

if(fresh==0) return 0;

int time=-1;

while(!queue.isEmpty()) {

int size=queue.size();

time++;

while(size-->0)

{

int tm[]=queue.poll();

x=tm[0];

y=tm[1];

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

int newx=x+dirs[i][0];

int newy=y+dirs[i][1];

if(newx>=0 && newy>=0 && newx<n && newy<m && grid[newx][newy]==1 ){

queue.add(new int[]{newx,newy});

grid[newx][newy]=2;

fresh--;

}

}

}

}

if(fresh>0)

return -1;

return time;

}

}

## Q-15

class Solution

{

static void linkdelete(Node head, int M, int N)

{

int cnt;

if(head==null) return;

Node curr = head;

Node temp;

while (curr!=null)

{

cnt=1;

while(cnt < M && curr != null){

curr = curr.next;

cnt++;

}

if (curr == null)

return;

temp = curr.next;

cnt=1;

while (cnt <= N && temp != null)

{

temp = temp.next;

cnt++;

}

curr.next = temp;

curr = temp;

}

}

}

## Q-10

class Solution {

void matchPairs(char nuts[], char bolts[], int n){

Arrays.sort(nuts);

Arrays.sort(bolts);

}

}

## Q-12

class Solution

{

String colName (long n)

{

StringBuilder sb = new StringBuilder();

while (n > 0) {

int rem =(int) n % 26;

if (rem == 0) {

sb.append("Z");

n = (n / 26) - 1;

}

else

{

sb.append((char)((rem - 1) + 'A'));

n = n / 26;

}

}

return sb.reverse().toString();

}

}

## Q-11

class Tree

{

class TreeHelper{

int index;

}

//Function to serialize a tree and return a list containing nodes of tree.

public void serialize(Node root, ArrayList<Integer> A)

{

if(root==null){

A.add(-1);

return;

}

A.add(root.data);

serialize(root.left, A);

serialize(root.right, A);

}

//Function to deserialize a list and construct the tree.

public Node deSerialize(ArrayList<Integer> A)

{

TreeHelper th=new TreeHelper();

//starting index of node

th.index=0;

return BinaryTree(A, th);

}

//Function to construct the tree.

public Node BinaryTree(ArrayList<Integer> A, TreeHelper th)

{

//check out of bound

if (th.index == A.size() || A.get(th.index) == -1)

{

th.index += 1;

return null;

}

Node root = new Node(A.get(th.index));

th.index += 1;

//calling function recursively for left and right subtrees.

root.left = BinaryTree(A, th);

root.right = BinaryTree(A, th);

return root;

}

};

## Q-14

class Solution

{

/\*class Node {

int data;

Node left;

Node right;

Node(int data) {

this.data = data;

left = null;

right = null;

}

}\*/

private static Node MapParent(Node root,

HashMap<Node, Node> map, int start) {

Queue<Node> q = new LinkedList<>();

q.offer(root);

Node res = new Node(-1);

while(!q.isEmpty()) {

Node node = q.poll();

if(node.data == start) res = node;

if(node.left != null) {

map.put(node.left, node);

q.offer(node.left);

}

if(node.right != null) {

map.put(node.right, node);

q.offer(node.right);

}

}

return res;

}

private static int findMaxDistance(HashMap<Node, Node> mpp, Node target) {

Queue<Node> q = new LinkedList<>();

q.offer(target);

HashMap<Node,Integer> vis = new HashMap<>();

vis.put(target, 1);

int maxi = 0;

while(!q.isEmpty()) {

int sz = q.size();

int fl = 0;

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

Node node = q.poll();

if(node.left != null && vis.get(node.left) == null) {

fl = 1;

vis.put(node.left, 1);

q.offer(node.left);

}

if(node.right != null && vis.get(node.right) == null) {

fl = 1;

vis.put(node.right, 1);

q.offer(node.right);

}

if(mpp.get(node) != null && vis.get(mpp.get(node)) == null) {

fl = 1;

vis.put(mpp.get(node), 1);

q.offer(mpp.get(node));

}

}

if(fl == 1) maxi++;

}

return maxi;

}

public static int minTime(Node root, int target)

{

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

Node start = MapParent(root, map, target);

int max = findMaxDistance(map, start);

return max;

}

}

## Q-4

class Solution{

static char start;

static StringBuilder sb=new StringBuilder();

static void getString(int i, int j, int n, int[][] bracket,StringBuilder sb){

if (i == j)

{

sb.append(start++);

return;

}

sb.append("(");

getString(i, bracket[i][j], n, bracket,sb);

getString(bracket[i][j] + 1, j, n, bracket,sb);

sb.append(")");

}

static String matrixChainOrder(int p[], int n){

int[][] m = new int[n][n];

int[][] bracket = new int[n][n];

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

m[i][i] = 0;

}

for (int gap = 2; gap < n; gap++)

{

for (int i = 1; i < n - gap + 1; i++)

{

int j = i + gap - 1;

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

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

{

int q = m[i][k] + m[k + 1][j] + p[i - 1] \* p[k] \* p[j];

if (q < m[i][j])

{

m[i][j] = q;

bracket[i][j] = k;

}

}

}

}

start='A';

getString(1, n - 1, n, bracket,sb);

return sb.toString();

}

}

## Q-9

class Solution{

public int isValid(int mat[][]){

int row[][]=new int[9][9];

int col[][]=new int[9][9];

int box[][][]=new int[3][3][9];

int num;

for (int i = 0; i < 9; i++) {

for (int j = 0; j < 9; j++) {

if (mat[i][j] > 0 && mat[i][j] <= 9) {

num = mat[i][j] - 1;

if (row[i][num]!=0) return 0;

row[i][num] = 1;

if (box[i / 3][j / 3][num] !=0) return 0;

box[i / 3][j / 3][num] = 1;

if (col[j][num]!=0) return 0;

col[j][num] = 1;

}

}

}

return 1;

}

}

## Q-5

class Solution{

static class Node

{

HashMap<Character,Node> child;

boolean end;

public Node()

{

child = new HashMap<Character,Node>();

for (char i = 'a'; i <= 'z'; i++)

child.put(i,null);

end = false;

}

}

static Node root;

static void insert(String s)

{

int len = s.length();

Node itr = root;

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

{

Node nextNode = itr.child.get(s.charAt(i));

if (nextNode == null)

{

nextNode = new Node();

itr.child.put(s.charAt(i),nextNode);

}

itr = nextNode;

if (i == len - 1){

itr.end = true;

}

}

}

static void insertIntoTrie(String contacts[])

{

root = new Node();

int n = contacts.length;

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

{

insert(contacts[i]);

}

}

static void displayContactsUtil(Node curNode,

String prefix,ArrayList<String> res)

{

if (curNode.end){

res.add(prefix);

}

for (char i = 'a'; i <= 'z'; i++)

{

Node nextNode = curNode.child.get(i);

if (nextNode != null)

{

displayContactsUtil(nextNode, prefix + i, res);

}

}

}

static ArrayList<ArrayList<String>> displayContacts(int n,

String contact[], String s)

{

insertIntoTrie(contact);

ArrayList<ArrayList<String>> res =new ArrayList<>();

Node prevNode = root;

String prefix = "";

int len = s.length();

int i;

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

{

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

prefix += s.charAt(i);

char lastChar = prefix.charAt(i);

Node curNode = prevNode.child.get(lastChar);

if (curNode == null)

{

ans.add("0");

res.add(ans);

i++;

break;

}

displayContactsUtil(curNode, prefix, ans);

res.add(ans);

prevNode = curNode;

}

while(i<len){

ArrayList<String> zero=new ArrayList<>();

zero.add("0");

res.add(zero);

i++;

}

return res;

}

}