```
public class Striver_Revision_SDE_Sheet {
    class Day_1_Arrays{
    // Set Matrix Zeroes
    class Solution{
        //Brute Force-
        class Solution {
            public void setZeroes(int[][] matrix) {
                int val=-1;
                int n=matrix.length;
                int m=matrix[0].length;
                for(int i=0;i<n;i++){
                    for(int j=0;j<m;j++){
                        if(matrix[i][j]==0){FillRowsAndCols(i,j,matrix,val,n,m);}
                    }
                }
                for(int i=0;i<n;i++){
                    for(int j=0;j<m;j++){
                        if(matrix[i][j]==val){matrix[i][j]=0;}
                    }
                }
            }
            public void FillRowsAndCols(int i,int j,int [][]matrix,int val,int n,int m){
                for(int row=0;row<n;row++){
                    if(matrix[row][j]!=0)matrix[row][j]=val;
                    else if(matrix[row][j]==0)continue;
                }
                for(int col=0;col<m;col++){
                    if(matrix[i][col]!=0)matrix[i][col]=val;
                    else if(matrix[i][col]==0)continue;
                    }
            }
        }
        //Optimised 1
        class Solution {
            public void setZeroes(int[][] matrix) {
                int val=-1;
                int n=matrix.length;
                int m=matrix[0].length;
                int rowSpace[]=new int[n];
                int colSpace[]=new int[m];
                for(int i=0;i<n;i++){
                    for(int j=0;j<m;j++){
                        if(matrix[i][j]==0){
                            rowSpace[i]=1;
                            colSpace[j]=1;
                        }
                    }
                }
                for(int i=0;i<n;i++){if(rowSpace[i]==1){FillRow(i,matrix,m);}}
```

```
            for(int i=0;i<m;i++){if(colSpace[i]==1){FillCol(i,matrix,n);}}
        }
        public void FillRow(int row,int matrix[][],int m){
            for(int i=0;i<m;i++){matrix[row][i]=0;}
        }
        public void FillCol(int col,int matrix[][],int n){
            for(int i=0;i<n;i++){matrix[i][col]=0;}
        }
    }
    //Optimised 2
    class Solution{
        static void setZeroes(int[][] matrix){
                int col0=1;
                int rows=matrix.length;
                int cols=matrix[0].length;
                for(int i=0;i<rows;i++){
                    if(matrix[i][0]==0)col0 = 0;
                    for(int j=1;j<cols;j++)
                        if(matrix[i][j]==0)
                            matrix[i][0]=matrix[0][j]=0;
                }

                for (int i=rows-1;i>=0;i--){
                    for(int j=cols-1;j>=1;j--)
                        if(matrix[i][0]==0||matrix[0][j]==0)
                            matrix[i][j]=0;
                    if(col0==0)matrix[i][0] = 0;
                }
            }
    }
}
    // Pascal's Triangle
    class Solution{
        //Variation I
        //Variation II
        //Variation III
    }
    // Next Permutation
    // Kadane's Algorithm
    // Sort an array of 0's 1's 2's
    // Stock buy and Sell
    }
    class Day_2_Arrays{
    // Rotate Matrix
    class Solution{
        //Brute force
        class Solution{
            static int[][] rotate(int[][] matrix) {
                int n=matrix.length;
```

```
            int rotated[][]=new int[n][n];
            for (int i=0;i<n;i++) {
                for (int j=0;j<n;j++){
                    rotated[j][n-i-1]=matrix[i][j];
                }
            }
            return rotated;
        }
    }
    //Optimised -Transpose and Swap Rows
    class Solution{
        static void rotate(int[][] matrix) {
            for(int i=0;i<matrix.length;i++) {
                for (int j=i;j<matrix[0].length;j++){
                    int temp=0;
                    temp=matrix[i][j];
                    matrix[i][j]=matrix[j][i];
                    matrix[j][i]=temp;
                }
            }
            for(int i=0;i<matrix.length;i++){
                for (int j=0;j<matrix.length/2;j++) {
                    int temp=0;
                    temp=matrix[i][j];
                    matrix[i][j]=matrix[i][matrix.length-1-j];
                    matrix[i][matrix.length-1-j]=temp;
                }
            }
        }
    }
}
  // Merge Overlapping Subintervals
class Solution{
    //Brute Force
    //Optimised
    class Solution{
    static ArrayList<List<Integer>> merge(ArrayList<List<Integer>> intervals){
        Collections.sort(intervals,(a,b)->a.get(0)-b.get(0));
        ArrayList<List<Integer>> merged =new ArrayList<>();
        for (int i=0;i<intervals.size();i++) {
            if (merged.isEmpty()||merged.get(merged.size()-1).get(1)<intervals.get(i).
get(0)){
                ArrayList<Integer> v=new ArrayList<>();
                v.add(intervals.get(i).get(0));
                v.add(intervals.get(i).get(1));
                merged.add(v);
            } else {
                merged.get(merged.size()-1).set(1,Math.max(merged.get(merged.size()-1).
get(1),intervals.get(i).get(1)));
```

```
            }
        }
        return merged;
    }
  }
}
    // Merge two sorted Arrays without extra space
    // Find the duplicate in an array of N+1 integers.
  class Solution{
      //Brute Force-First Sort Then Check adjacent elements
      class Solution{
          static int findDuplicate(int[] arr){
              int n=arr.length;
              Arrays.sort(arr);
              for(int i=0;i<n-1;i++){
                  if(arr[i]==arr[i+1]){
                      return arr[i];
                  }
              }
              return 0;
          }
      }
      //Using Frequency Array-O(2N),O(N)
      class Solution{
          static int findDuplicate(int[] arr){
              int n=arr.length;
              int freq[]=new int[n+1];
              for(int i=0;i<n;i++){
                  if(freq[arr[i]]==0){freq[arr[i]]+=1;}
                  else {return arr[i];}
              }
              return 0;
          }
      }
      //Linked List Tortoise Method
      class Solution{
          public static int findDuplicate(int[] nums){
              int slow=nums[0];
              int fast=nums[0];
              do{
                  slow=nums[slow];
                  fast=nums[nums[fast]];
              }while(slow!=fast);
              fast=nums[0];
              while(slow!=fast){
                  slow=nums[slow];
                  fast=nums[fast];
              }
              return slow;
```

```
            }
        }
}
        // Repeat and Missing Number
        //Freq array-O(2N),O(N)
        //Maths solution-n,n^2;
        //XOR-Method

    // Inversion of Array (Pre-req: Merge Sort)
}
class Day_3_Arrays{
    // Search in a 2d Matrix
    // Pow(X,n)
    // Majority Element (>N/2 times)
    // Majority Element (>N/3 times)
    // Grid Unique Paths
    // Reverse Pairs (Leetcode)
}
class Day_4_Arrays{
    // 2-Sum-Problem
    class Solution{
        //Brute
        public int twoSum(int arr[],int target){
            int n=arr.length;
            for(int i=0;i<n;i++){
                for(int j=i+1;j<n;j++){
                    if(arr[i]+arr[j]==target){return true;}
                }
            }
            return false;
        }
       //Two Pointer
        public int twoSum(int arr[],int target){
            int n=arr.length;
            int start=0,end=n-1;
            while(start<end){
                if(arr[start]+arr[end]==target){return true;}
                else if(arr[start]+arr[end]<target){start++;}
                else{end--;}
            }
            return false;
        }
         //Hashing
        public int twoSum(int arr[],int target){
            int n=arr.length;
            for(int i=0;i<n;i++){
                if(mp.containsKey(target-arr[i])){return true;}
                else{mp.put(arr[i],i);}
            }
```

```
            return false;
        }
    }
    // 4-sum-Problem
    class Solution{
        //Three Loops+Binary Search

    }
    // Longest Consecutive Sequence
    // Largest Subarray with 0 sum
    // Count number of subarrays with given Xor K
    // Longest Substring without repeat
}
class Day_5_{
    //Reverse a LinkedList
    class Solution{
        public ListNode reverseList(ListNode head) {
            ListNode curr=head;
            ListNode prev=null;
            while(curr!=null){
                ListNode temp=curr.next;
                curr.next=prev;
                prev=curr;
                curr=temp;
            }
            return prev;
        }
    }
      //Find the middle of LinkedList
    class Solution {
        public ListNode middleNode(ListNode head) {
           ListNode slow=head;
           ListNode fast=head;
           while(fast!=null&&fast.next!=null){
               slow=slow.next;
               fast=fast.next.next;
           }
            return slow;
        }
    }
      //Merge two sorted Linked List (use method used in mergeSort)
      //Remove N-th node from back of LinkedList
      //Add two numbers as LinkedList
    class Solution {
        public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
            ListNode dummy=new ListNode();
            ListNode temp=dummy;
            int sum=0,carry=0;
            while(l1!=null|| l2!=null || carry==1){
```

```
            if(l1!=null){
                sum+=l1.val;
                l1=l1.next;
            }
            if(l2!=null){
                sum+=l2.val;
                l2=l2.next;
            }
            sum+=carry;
            carry=sum/10;
            ListNode h=new ListNode(sum%10);
            temp.next=h;
            temp=h;
            sum=0;
        }
        return dummy.next;
    }
}
  //Delete a given Node when a node is given.(0(1) solution)
class Solution {
    public void deleteNode(ListNode node) {
        if(node==null)return;
        if(node.next!=null) {
            int nextValue=node.next.val;
            node.next=node.next.next;
            node.val=nextValue;
        }
    }
}
}
class Day_6_{
    //Find intersection point of Y LinkedList
  //Detect a cycle in Linked List
    class Solution{
        static boolean cycleDetect(Node head) {
            if(head==null) return false;
            Node fast=head;
            Node slow=head;
            while(fast.next!=null&&fast.next.next!=null){
                fast=fast.next.next;
                slow=slow.next;
                if(fast==slow) return true;
            }
        return false;
    }}
  //Reverse a LinkedList in groups of size k.
    class Solution {
        public ListNode reverseKGroup(ListNode head,int k){
            if(head==null)return null;
```

```
        ListNode curr=head;
        ListNode nexN=null;
        ListNode prev=null;
        int count=0;
        while(count<k&&curr!=null){
            nexN=curr.next;
            curr.next=prev;
            prev=curr;
            curr=nexN;
            count++;
        }
        if(nexN!=null)head.next=reverseKGroup(nexN,k);
        return prev;
    }
}
  //Check if a LinkedList is palindrome or not.
class Solution {
    public boolean isPalindrome(ListNode head) {
        ListNode slow=head,slowprev=head,fast=head;
        while(fast!=null&&fast.next!=null){
            slowprev=slow;
            slow=slow.next;
            fast=fast.next.next;
        }
        ListNode revHead=reverse(slowprev);
        ListNode startForward=head,tailBackward=revHead;
        while(startForward!=null){
            if(startForward.val!=tailBackward.val){return false;}
            else{startForward=startForward.next;tailBackward=tailBackward.next;}
            }
            return true;
    }
    public ListNode reverse(ListNode head){
        ListNode prev=null;
        ListNode nextN=head;
        ListNode curr=head;
        while(curr!=null){
            nextN=curr.next;
            curr.next=prev;
            prev=curr;
            curr=nextN;
        }
        return prev;
    }
}
  //Find the starting point of the Loop of LinkedList
  class Solution {
    public ListNode detectCycle(ListNode head) {
      ListNode slow=head;
```

```
        ListNode fast=head;
        int flag=0;
        while(fast!=null&&fast.next!=null){
            slow=slow.next;
            fast=fast.next.next;
            if(slow==fast){
                flag=1;
                break;
            }
        }
        if(flag==0){
            return null;
        }
        ListNode first=head;
        ListNode second=slow;
        while(first!=second){
            first=first.next;
            second=second.next;
        }
        return first;
    }
}
//Flattening of a LinkedList
class Solution{
    class Node{
    int data;
    Node next;
    Node bottom;
        Node(int d){
            data=d;
            next=null;
            bottom=null;
        }
    }
Node flatten(Node root){
        if(root==null||root.next==null)return root;
        root.next=flatten(root.next);
        root=mergeTwoLists(root,root.next);
        return root;
}
 Node mergeTwoLists(Node a,Node b){

    Node temp=new Node(0);
    Node res=temp;

    while(a!=null&&b!=null){
        if(a.data<b.data){
            temp.bottom=a;
            temp=temp.bottom;
```

```
            a=a.bottom;
        }
        else{
            temp.bottom=b;
            temp=temp.bottom;
            b=b.bottom;
        }
    }

    if(a!=null)temp.bottom = a;
    else temp.bottom=b;
    return res.bottom;
  }
}
}
class Day_7_{
    //Rotate a LinkedList
    //Clone a Linked List with random and next pointer
    //3 sum
    //Trapping rainwater
    //Remove Duplicate from Sorted array
    //Max consecutive ones
}
class Day_8_{
  //N meetings in one room
  class Solution {
    class meeting{
        int start,end,pos;
        meeting(int s,int e,int p){
            this.start=s;
            this.end=e;
            this.pos=p;
        }
    }
    class meetingComparator implements Comparator<meeting>
    {
        @Override
        public int compare(meeting o1,meeting o2)
        {
            if(o1.end<o2.end)return -1;
            else if(o1.end>o2.end)return 1;
            else if(o1.pos<o2.pos)return -1;
            else return 1;
        }
    }
    public static int maxMeetings(int start[], int end[], int n)
    {
        ArrayList<meeting> meet=new ArrayList<>();
        for(int i=0;i<n;i++){meet.add(new meeting(start[i],end[i],i+1));}
```

```
        meetingComparator mc=new meetingComparator();
        Collections.sort(meet,mc);
        ArrayList<Integer> ans=new ArrayList<>();
        ans.add(meet.get(0).pos);
        int limit=meet.get(0).end;
        for(int i=1;i<n;i++){
            if(meet.get(i).start>limit){
                limit=meet.get(i).end;
                ans.add(meet.get(i).pos);
            }
        }
        return ans.size();
    }
}
  //Minimum number of platforms required for a railway
class Solution{
    static int findPlatform(int arr[],int dep[],int n){
// add your code here
        Arrays.sort(arr);
        Arrays.sort(dep);
        int plat=1,res=1,i=1,j=0;
        while(i<n&&j<n){
            if(arr[i]<=dep[j]){plat++;i++;}
            else if(arr[i]>dep[j]){plat--;j++;}
            if(res<plat){res=plat;}
        }
            return res;
        }
    }
//Job sequencing Problem
class Solution{
    //Function to find the maximum profit and the number of jobs done.
    int[] JobScheduling(Job arr[], int n)
    {
        // Your code here
        Arrays.sort(arr,(a,b)->(b.profit-a.profit));
        int maxi=0;
        for(int i=0;i<n;i++){maxi=Math.max(maxi,arr[i].deadline);}
        int result[]=new int[maxi+1];
        for(int j=0;j<maxi+1;j++){result[j]=-1;}
        int profit=0,countJobs=0;
        for(int k=0;k<n;k++){
            for(int m=arr[k].deadline;m>=1;m--){
                if(result[m]==-1){
                    result[m]=k;
                    countJobs++;
                    profit+=arr[k].profit;
                    break;
                }
```

```
            }
        }
        int fin[]={countJobs,profit};
        return fin;
    }
}
  //Fractional Knapsack Problem
class Solution{
    class itemComparator implements Comparator<Item>{
        @Override
        public int compare(Item a,Item b){
            double r1=(double)a.value/(double)a.weight;
            double r2=(double)b.value/(double)b.weight;
            if(r1<r2){return 1;}
            else if(r1>r2){return -1;}
            else return 0;
        }
    }
    double fractionalKnapsack(int W, Item arr[], int n){
        Arrays.sort(arr,new itemComparator());
        int currWeight=0;
        double maxProfit=0.0;
        for(int i=0;i<n;i++){
            if(currWeight+arr[i].weight<=W){
                currWeight+=arr[i].weight;
                maxProfit+=arr[i].value;
            }
            else{
                int remain=W-currWeight;
                maxProfit+=((double)arr[i].value/(double)arr[i].weight)*(double)remain;
                break;
            }
        }
        return maxProfit;
    }
}
  //Greedy algorithm to find minimum number of coins
class Solution{
    public static int findMinimumCoins(int V)
        {
            // Write your code here.
        ArrayList<Integer> ans=new ArrayList<>();
        int coins[]={1,2,5,10,20,50,100,500,1000};
        int n = coins.length;
        for (int i=n-1;i>=0;i--){
            while(V>=coins[i]) {
                V-=coins[i];
                ans.add(coins[i]);
            }
```

```
            }
            return ans.size();
            }
        }
       //Activity Selection (it is the same as N meeting in one room)
   }
   class Day_9_Recursion{
      // Subset Sums -O(2^N+2^Nlog(2^N)),O(2^N)
      class Solution{
      static void subsetSumsHelper(int ind,int sum,ArrayList<Integer> arr,int N,
ArrayList<Integer> sumSubset) {
         if(ind==N){
            sumSubset.add(sum);
            return;
         }
         subsetSumsHelper(ind+1,sum+arr.get(ind),arr,N,sumSubset);//Pick
         subsetSumsHelper(ind+1,sum,arr,N,sumSubset);//Not Pick
      }
      static ArrayList<Integer> subsetSums(ArrayList<Integer> arr,int N){
         ArrayList<Integer> sumSubset=new ArrayList<>();
         subsetSumsHelper(0,0,arr,N,sumSubset);
         Collections.sort(sumSubset);
         return sumSubset;
      }
   }
      // Subset-II
      class Solution{
      //Brute force-O(2^N+mlogm(m=2^N)),O(mlogm)
      class Solution{
         static void printAns(List<String>ans){
            System.out.println("The unique subsets are ");
            System.out.println(ans.toString().replace(",", " "));
         }
         public static void fun(int[] nums,int index,List<Integer> ds,HashSet<String>res){
            if(index==nums.length){
               Collections.sort(ds);
               res.add(ds.toString());
               return;
            }
            ds.add(nums[index]);
            fun(nums,index+1,ds,res);
            ds.remove(ds.size()-1);
            fun(nums,index+1,ds,res);
         }
         public static List<String> subsetsWithDup(int[] nums){
            List<String> ans=new ArrayList<>();
            HashSet<String> res=new HashSet<>();
            List<Integer> ds=new ArrayList<>();
            fun(nums,0,ds,res);
```

```
            for(String it: res) {
                ans.add(it);
            }
            return ans;
        }
    }
    //Optimal
    class Solution{
        static void printAns(List<List<Integer>> ans){
            System.out.println("The unique subsets are ");
            System.out.println(ans.toString().replace(","," "));
        }
         public static void findSubsets(int ind,int[] nums,List<Integer> ds,
List<List<Integer>> ansList) {
                ansList.add(new ArrayList<>(ds));
                for(int i=ind;i<nums.length;i++){
                    if(i!=ind&&nums[i]==nums[i-1])continue;
                    ds.add(nums[i]);
                    findSubsets(i+1,nums,ds,ansList);
                    ds.remove(ds.size()-1);
                }
            }
            public static List<List<Integer>> subsetsWithDup(int[] nums){
                Arrays.sort(nums);
                List<List<Integer>> ansList = new ArrayList<>();
                findSubsets(0,nums,new ArrayList<>(),ansList);
                return ansList;
            }
        }
    }
    // Combination sum-1
    class Solution{
    //Optimised-O(2^t*K),O()
    class Solution {
        private void findCombinations(int ind,int[] arr,int target,List<List<Integer>> ans,
List<Integer>ds) {
            if(ind==arr.length) {
                if(target==0){ans.add(new ArrayList<>(ds));}
                return;
            }
            if(arr[ind]<=target){
                ds.add(arr[ind]);
                findCombinations(ind,arr,target-arr[ind],ans,ds);
                ds.remove(ds.size()-1);
            }
            findCombinations(ind+1,arr,target,ans,ds);
        }
        public List<List<Integer>> combinationSum(int[] candidates, int target){
            List<List<Integer>> ans =new ArrayList<>();
```

```
            findCombinations(0,candidates,target,ans,new ArrayList<>());
            return ans;
        }
    }
  }
    // Combination sum-2
    class Solution{
    //Brute Using Set(O(2^t*K*log(s)))|O(2^N*k),O(k*x)
    //Optimised
    class Solution{
        static void findCombinations(int ind,int[] arr,int target,List<List <Integer>> ans,
List<Integer> ds) {
            if(target==0){
                ans.add(new ArrayList<>(ds));
                return;
            }
            for(int i=ind;i<arr.length;i++){
                if (i>ind&&arr[i]==arr[i-1])continue;
                if (arr[i]>target)break;
                ds.add(arr[i]);
                findCombinations(i+1,arr,target-arr[i],ans,ds);
                ds.remove(ds.size()-1);
            }
        }
     }
    }
    // Palindrome Partitioning
    class Solution{
        //Optimised
        class Solution{
            public static List<List<String>> partition(String s){
                List<List<String>> res=new ArrayList<>();
                List<String> path=new ArrayList<>();
                partitionHelper(0,s,path,res);
                return res;
            }
            static void partitionHelper(int index,String s,List<String> path,
List<List<String>> res) {
                if(index==s.length()){
                    res.add(new ArrayList<>(path));
                    return;
                }
                for (int i=index;i<s.length();++i) {
                    if(isPalindrome(s, index,i)){
                        path.add(s.substring(index,i+1));
                        partitionHelper(i+1,s,path,res);
                        path.remove(path.size()-1);
                    }
                }
```

```
            }
            static boolean isPalindrome(String s,int start,int end){
                while(start<=end){
                    if(s.charAt(start++)!=s.charAt(end--))
                        return false;
                }
                return true;
            }
        }
    }
    // K-th permutation Sequence
}
class Day_10_RecursionBackTracking{
    //Print all permutations of a string/array
    class Solution{
        //Extra Space Complexity-With HashMap
        //TC=O(n!*n)SC-O(2n)
        class Solution {
            public List<List<Integer>> permute(int[] nums) {
                List<List<Integer>> ans=new ArrayList<>();
                List<Integer> ds=new ArrayList<>();
                int n=nums.length;
                boolean freq[]=new boolean[n];
                permuterecur(ans,ds,nums,n,freq);
                return ans;
            }
            public void permuterecur(List<List<Integer>> ans,List<Integer> ds,int []nums,
int n,boolean freq[]){
                if(ds.size()==n){
                    List<Integer> put=new ArrayList<>(ds);
                    ans.add(put);
                    return;
                }
                for(int i=0;i<n;i++){
                    if(freq[i]==false){
                        freq[i]=true;
                        ds.add(nums[i]);
                        permuterecur(ans,ds,nums,n,freq);
                        ds.remove(ds.size()-1);
                        freq[i]=false;
                    }
                }
            }
        }
        //Optimised -Swapping
        //TC=O(n!*n)SC-O(n)
        class Solution {
            private void recurPermute(int index,int[] nums,List<List<Integer>> ans){
                if(index==nums.length){
```

```
                // copy the ds to ans
                List<Integer> ds=new ArrayList<>();
                for(int i=0;i<nums.length;i++){ds.add(nums[i]);}
                ans.add(new ArrayList<>(ds));
                return;
            }
            for(int i=index;i<nums.length;i++){
                swap(i,index,nums);
                recurPermute(index+1,nums,ans);
                swap(i,index,nums);
            }
        }
        private void swap(int i,int j,int[] nums) {
            int t=nums[i];
            nums[i]=nums[j];
            nums[j]=t;
        }
        public List<List<Integer>> permute(int[] nums){
            List<List<Integer>> ans=new ArrayList<>();
            recurPermute(0,nums,ans);
            return ans;
        }
    };
}
//N queens Problem
class Solution{
    public static List<List<String>> solveNQueens(int n){
        char[][] board=new char[n][n];
        for (int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                board[i][j]='.';
        List<List<String>> res=new ArrayList<List<String>>();
        dfs(0,board,res);
        return res;
    }
    static boolean validate(char[][] board,int row,int col){
        int duprow=row;
        int dupcol=col;
        while(row>=0&&col>=0){
            if(board[row][col]=='Q')return false;
            row--;
            col--;
        }
        row=duprow;
        col=dupcol;
        while(col>=0){
            if(board[row][col]=='Q')return false;
            col--;
        }
```

```
        row=duprow;
        col=dupcol;
        while(col>=0&&row<board.length){
            if(board[row][col]=='Q')return false;
            col--;
            row++;
        }
        return true;
    }
    static void dfs(int col,char[][] board,List<List<String>> res){
        if(col==board.length){
            res.add(construct(board));
            return;
        }

        for(int row=0;row<board.length;row++){
            if(validate(board,row,col)){
                board[row][col]='Q';
                dfs(col+1,board,res);
                board[row][col]='.';
            }
        }
    }
    static List<String> construct(char[][] board){
        List<String> res=new LinkedList<String>();
        for(int i=0;i<board.length;i++){
            String s=new String(board[i]);
            res.add(s);
        }
        return res;
    }
}
//Sudoku Solver
class Solution {
    public void solveSudoku(char[][] board) {
        solveSudokuUtil(board);
    }
    public boolean solveSudokuUtil(char board[][]){
        for(int i=0;i<9;i++){
            for(int j=0;j<9;j++){
                if(board[i][j]=='.'){
                    for(char c='1';c<='9';c++){
                        if(isValid(board,i,j,c)){
                            board[i][j]=c;
                            if(solveSudokuUtil(board)==true){
                                return true;
                            }
                            else
                            board[i][j]='.';
```

```
                    }
                }
                return false;
            }
        }
    }
    return true;
}
public boolean isValid(char board[][],int row,int col,char ch){
    for(int i=0;i<9;i++){
        if(board[row][i]==ch){
            return false;
        }
        if(board[i][col]==ch){
            return false;
        }
        if(board[3*(row/3)+(i/3)][3*(col/3)+i%3]==ch){
            return false;
        }
    }
    return true;
}
}
//M coloring Problem
class Solution{
    public boolean graphColoring(boolean graph[][], int m, int n) {
        int color[]=new int[n];
        for(int i=0;i<n;i++)
        {
            color[i] = 0;
        }
        if(graphColoringUtil(graph,m,color,0,n)==false){
            return false;
        }
        return true;
    }
    boolean graphColoringUtil(boolean graph[][],int m,int color[],int ind,int n){
        if(ind==n)
        {return true;}
        for(int c=1;c<=m;c++){
            if(isSafe(ind,graph,color,c,n)){
                color[ind]=c;
                if(graphColoringUtil(graph,m,color,ind+1,n) == true)
                    return true;
                color[ind]=0;
            }
        }
        return false;
    }
```

```
        boolean isSafe(int ind,boolean graph[][],int color[],int c,int n){
            for (int i=0;i<n;i++)
                if(graph[ind][i]&&c==color[i])
                {return false;}
            return true;
        }
    }
    //Rat in a Maze
    class Solution {
        public static void solve(int i,int j,int[][] m,int vis[][],ArrayList<String> ans,String
move,int n){
            if((i==n-1)&&(j==n-1)){
                ans.add(move);
                return;
            }
            if(i+1<n&&vis[i+1][j]==0&&m[i+1][j]==1){
                vis[i][j]=1;
                solve(i+1,j,m,vis,ans,move+"D",n);
                vis[i][j]=0;
            }
            if(j-1>=0&&vis[i][j-1]==0&&m[i][j-1]==1){
                vis[i][j]=1;
                solve(i,j-1,m,vis,ans,move+"L",n);
                vis[i][j]=0;
            }
            if(j+1<n&&vis[i][j+1]==0&&m[i][j+1]==1){
                vis[i][j]=1;
                solve(i,j+1,m,vis,ans,move+"R",n);
                vis[i][j]=0;
            }
            if(i-1>=0&&vis[i-1][j]==0&&m[i-1][j]==1){
                vis[i][j]=1;
                solve(i-1,j,m,vis,ans,move+"U",n);
                vis[i][j]=0;
            }

        }
        public static ArrayList<String> findPath(int[][] m, int n) {
            // Your code here
            int vis[][]=new int[n][n];
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    vis[i][j]=0;
                }
            }
            ArrayList<String> ans=new ArrayList<>();
            if(m[0][0]==1){
                solve(0,0,m,vis,ans,"",n);
            }
```

```
            return ans;
        }

    //************Code 2*************Rat in a
Maze********************************************************************** */
    class Solution {
        public static void solve(int i,int j,int[][] m,int vis[][],ArrayList<String> ans,String
move,int n,int dx[],int dy[]){
            if((i==n-1)&&(j==n-1)){
                ans.add(move);
                return;
            }
            String base="DLRU";
            for(int p=0;p<4;p++){
                int nexti=i+dx[p];
                int nextj=j+dy[p];

if(nexti>=0&&nexti<n&&nextj>=0&&nextj<n&&vis[nexti][nextj]==0&&m[nexti][nextj]==1){
                    vis[i][j]=1;
                    solve(nexti,nextj,m,vis,ans,move+base.charAt(p),n,dx,dy);
                    vis[i][j]=0;
                }
            }

        }
        public static ArrayList<String> findPath(int[][] m, int n) {
            // Your code here
            int vis[][]=new int[n][n];
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    vis[i][j]=0;
                }
            }
            int dx[]={1,0,0,-1};
            int dy[]={0,-1,1,0};
            ArrayList<String> ans=new ArrayList<>();
            if(m[0][0]==1){
                solve(0,0,m,vis,ans,"",n,dx,dy);
            }
            return ans;
        }
        }
    }
    //Word Break (print all ways)
    class Solution{
        class Solution {
            public boolean wordBreak(String s, List<String> wordDict) {
                int m=wordDict.size(),n=s.length();
                String dictionary[]=new String[m];
```

```
            int k=0;
            for(String temp: wordDict){dictionary[k++]=temp;}
            boolean wb[]=new boolean[n+1];
            return wordbreak(s,wb,n,dictionary);
        }
        public boolean wordbreak(String str,boolean wb[],int n,String dictionary[]){
            int y=str.length();
            if(y==0){return true;}
            for(int i=1;i<=n;i++){
                if(wb[i]==false&&check(str.substring(0,i),dictionary)==true){
                    wb[i]=true;
                }
                if(wb[i]==true){
                    if(i==n){return true;}
                    for(int j=i+1;j<=n;j++){
                        if(wb[j]==false&&check(str.substring(i,j),dictionary)==true)
{wb[j]=true;}
                        if(j==n&&wb[j]==true){return true;}
                    }
                }
            }
            return false;
        }
        public boolean check(String s,String dictionary[]){
            for(String temp: dictionary){
                if(temp.equals(s)==true){return true;}
            }
            return false;
        }
    }
    class Solution {
        public List<String> wordBreak(String s, List<String> wordDict) {
            List<String> res=new ArrayList<>();
            int n=s.length();
            String ans="";
            wordBreakUtil(n,s,wordDict,ans,res);
            return res;
        }


      static void wordBreakUtil(int n,String s,List<String> dict,String ans,List<String>
res){
            for(int i=1;i<=n;i++){
                String prefix=s.substring(0,i);
                if(dict.contains(prefix))
                {
                    if(i==n){ans+=prefix;an.add(ans);return;}
                    wordBreakUtil(n-i,s.substring(i,n),dict,ans+prefix+" ",res);
                }
```

```
                }
            }
        }
    }
}
class Day_11_BinarySearch{
    // The N-th root of an integer
    class Solution{
        //O(log(N*10^d)*M)
        class Solution{
            private static double multiply(double number, int n) {
                double ans=1.0;
                for(int i=1;i<=n;i++){
                    ans=ans*number;
                }
                return ans;
            }
            private static void getNthRoot(int n, int m){
                double low=1,double high=m,double eps=1e-7;
                while((high-low)>eps){
                    double mid =(low+high)/2.0;
                    if(multiply(mid,n)<m){low=mid;}
                    else {high=mid;}
                }
                System.out.println(n+"th root of "+m+" is "+low);
    }
}
}
    // Matrix Median-
    class Solution{
        //O(log(2^32)*N*log(M)),O(1)
        class Solution {
            public int findMedian(ArrayList<ArrayList<Integer>> A) {
            int low=1,int high=1000000000,int N=A.size(),int M=A.get(0).size();
            while(low<=high){
                int mid=(low+high)/2;
                int cnt=0;
                for(int i=0;i<N;i++){cnt+=countEleLessThanVal(A.get(i),mid);}
                if(cnt<=((N*M)/2)){low=mid+1;}
                else{high=mid-1;}
                }
                return low;
            }
            public int countEleLessThanVal(ArrayList<Integer> J,int target){
                int low=0;int high=J.size()-1;
                while(low<=high){
                    int mid=(low+high)/2;
                    if(J.get(mid)<=target){low=mid+1;}
                    else{high=mid-1;}
```

```
                }
                return low;
            }
        }
    }
    // Find the element that appears once in a sorted array, and the rest element
appears twice (Binary search)
    class Solution {
        //Even Odd Index- Left Half /Ele/ Right Half
        //O(Log(N))
        public int singleNonDuplicate(int[] nums) {
            int low=0,high=nums.length-2;
            while(low<=high){
                int mid=(low+high)/2;
                if(mid%2==0){
                    if(nums[mid]==nums[mid+1]){low=mid+1;}
                    else{high=mid-1;}
                }
                else{
                    if(nums[mid]==nums[mid-1]){low=mid+1;}
                    else{high=mid-1;}
                }
            }
            return nums[low];
        }
    }
    // Search element in a sorted and rotated array/ find pivot where it is rotated
    class Solution {
        public int search(int[] nums, int target) {
           int low=0,high=nums.length-1;
           while(low<=high)
            {
                int mid=(low+high)/2;
                if(nums[mid]==target){
                    return mid;
                }
                if(nums[low]<=nums[mid]){
                    if(nums[low]<=target&&nums[mid]>=target){
                        high=mid-1;
                    }
                    else{
                        low=mid+1;
                    }
                }
                else{
                if(nums[high]>=target&&nums[mid]<=target){
                        low=mid+1;
                    }
                    else{
```

```
                high=mid-1;
            }
        }
    }
    return -1;
  }
}
// Median of 2 sorted arrays

// K-th element of two sorted arrays
// Allocate Minimum Number of Pages
class Solution {
    public int books(ArrayList<Integer> A, int B) {
        if(A.size()<B)
            return -1;
        int low=A.get(0),high=0;
        for(int i=0;i<A.size();i++){
            high=high+A.get(i);
            low=Math.min(A.get(i),low);
        }
        int res=-1;
        while(low<=high){
            int mid=(low+high)/2;
            if(isPossible(A,mid,B)){
                res=mid;
                high=mid-1;
            }
            else{
                low=mid+1;
            }
        }
        return low;
    }
    public boolean isPossible(ArrayList<Integer> A,int pages,int B){
        int sumAllocated=0,cnt=0;
        for(int i=0;i<A.size();i++){
            if(sumAllocated+A.get(i)>pages)
            {
                cnt++;
                sumAllocated=A.get(i);
                if(sumAllocated>pages){
                    return false;
                }
            }
            else{
                sumAllocated+=A.get(i);
            }
        }
    if(cnt<B){
```

```
            return true;
        }
        return false;
        }
    }
    // Aggressive Cows
    class Solution {
        static boolean isPossible(int a[],int n,int cows,int minDist) {
            int cntCows=1;
            int lastPlacedCow=a[0];
            for(int i=1;i<n;i++) {
                if (a[i]-lastPlacedCow>=minDist){
                    cntCows++;
                    lastPlacedCow=a[i];
                }
            }
            if (cntCows>=cows)
            return true;
            else
            return false;
        }
        public static void main(String args[]) {
            int n = 5, cows = 3;
            int a[]={1,2,8,4,9};
            Arrays.sort(a);
            int low=1,high=a[n-1]-a[0];
            while(low<=high) {
                int mid=(low+high)/2;
                if(isPossible(a,n,cows,mid)){
                    low=mid+1;
                }else{
                    high=mid-1;
                }
            }
            System.out.println("The largest minimum distance is " + high);


        }
    }
}
class Day_12_{
    //Max heap, Min Heap Implementation (Only for interviews)
    class Solution{
    Min Heap Implementation
    PriorityQueue<Integer> pQueue=new PriorityQueue<Integer>();
    Max Heap Implementation
    PriorityQueue<Integer> pQueue=new PriorityQueue<Integer>(Collections.
reverseOrder());
    }
```

```
//Kth Largest Element
class Solution{
    public static int kthLargest(int[] arr,int l,int r,int k){
        //Your code here
        PriorityQueue<Integer> pq=new PriorityQueue<>();
        for(int i=l;i<=r;i++){
            pq.add(arr[i]);
            if(pq.size()>k){
                pq.remove();
            }
        }
        return pq.peek();
}
}
//Maximum Sum Combination
public class Solution {
    public class Pair{
        int l;
        int m;
        Pair(int _l,int _m){
            this.l=_l;
            this.m=_m;
        }
    }
    public class PairSum{
        int sum;
        int l;
        int m;
        PairSum(int _sum,int _l,int _m){
            this.sum=_sum;
            this.l=_l;
            this.m=_m;
        }
    }
    public ArrayList<Integer> solve(ArrayList<Integer> A, ArrayList<Integer> B, int
C) {
    PriorityQueue<PairSum> pq=new PriorityQueue<>((p,q)-> q.sum-p.sum);
    HashSet<Pair> hs=new HashSet<>();
    int n=A.size();
    Collections.sort(A);
    Collections.sort(B);
    int l=n-1,m=n-1;
    pq.add(new PairSum(A.get(l)+B.get(m),l,m));
    hs.add(new Pair(l,m));
    ArrayList<Integer> ans=new ArrayList<>();
    for(int i=0;i<C;i++){
        PairSum curr=pq.remove();
        ans.add(curr.sum);
        int L=curr.l;
```

```
            int M=curr.m;
            if(l>=0&&m>=0&&!hs.contains(new Pair(L-1,M))){
                pq.add(new PairSum(A.get(L-1)+B.get(M),L-1,M));
                hs.add(new Pair(L-1,M));
            }
            if(l>=0&&m>=0&&!hs.contains(new Pair(L,M-1))){
                pq.add(new PairSum(A.get(L)+B.get(M-1),L,M-1));
                hs.add(new Pair(L,M-1));
            }
        }
        return ans;
        }
    }
    //Find Median from Data Stream
    //Merge K sorted arrays
    //K most frequent elements
}
class Day_13_{
    //Implement Stack Using Arrays
    class Solution{
        class stack {
            int size=10000;
            int arr[]=new int[size];
            int top=-1;
            void push(int x){
                top++;
                arr[top]=x;
            }
            int pop(){
                int x=arr[top];
                top--;
                return x;
            }
            int top(){
                return arr[top];
            }
            int size(){
                return top+1;
            }
        }
    }
    //Implement Queue Using Arrays
    class Solution{
        class Queue {
            private int arr[];
            private int start,end,currSize,maxSize;
            public Queue(){
                arr=new int[16];
                start=-1;
```

```
        end=-1;
        currSize=0;
    }
    public Queue(int maxSize) {
        this.maxSize=maxSize;
        arr=new int[maxSize];
        start=-1;
        end=-1;
        currSize=0;
    }
    public void push(int newElement){
        if(currSize==maxSize){
            System.out.println("Queue is full\nExiting...");
            System.exit(1);
        }
        if(end==-1) {
            start=0;
            end=0;
        }else
            end=(end+1)%maxSize;
            arr[end]=newElement;
            System.out.println("The element pushed is " + newElement);
            currSize++;
    }
    public int pop(){
        if(start==-1){
            System.out.println("Queue Empty\nExiting...");
            System.exit(1);
        }
        int popped=arr[start];
        if(currSize==1){
            start=-1;
            end=-1;
        }else
            start=(start+1)%maxSize;
        currSize--;
        return popped;
    }
    public int top(){
        if(start ==-1){
            System.out.println("Queue is Empty");
            System.exit(1);
        }
        return arr[start];
    }
    public int size(){
        return currSize;
    }
}
```

```
    }
    //Implement Stack using Queue (using single queue)
    //Implement Queue using Stack (0(1) amortized method)
    //Check for balanced parentheses
    class Solution{
        public static boolean isValid(String s){
                Stack<Character> st=new Stack<Character>();
                for(char it: s.toCharArray()){
                    if(it=='('||it=='['||it=='{')
                        st.push(it);
                    else {
                        if(st.isEmpty()) return false;
                        char ch=st.pop();
                        if((it==')'&&ch =='(')||(it==']'&&ch=='[')||(it =='}'&&ch=='{')) continue;
                        else return false;
                    }
                }
                return st.isEmpty();
            }
        }
    //Next Greater Element
    class Solution{
        public static int[] nextGreaterElements(int[] nums){
            int n=nums.length;
            int nge[]=new int[n];
            Stack<Integer> st=new Stack<>();
            for(int i=2*n-1;i>=0;i--){
                while(st.isEmpty()==false&&st.peek()<=nums[i%n]) {
                    st.pop();
                }

                if(i<n){
                    if(st.isEmpty()==false)nge[i]=st.peek();
                    else nge[i]=-1;
                }
                st.push(nums[i%n]);
            }
            return nge;
        }
    }
    //Sort a Stack
}
class Day_14_{
    //Next Smaller Element
      //LRU cache (IMPORTANT)
      //LFU Cache
      //Largest rectangle in a histogram
      //Sliding Window maximum
      //Implement Min Stack
```

```
        //Rotten Orange (Using BFS)
        //Stock Span Problem
        //Find the maximum of minimums of every window size
        //The Celebrity Problem
    }
    class Day_15_{
        //Reverse Words in a String
        //Longest Palindrome in a string
        //Roman Number to Integer and vice versa
        //Implement ATOI/STRSTR
        //Longest Common Prefix
        //Rabin Karp
    }
    class Day_16_{
        //Z-Function
        //KMP algo / LPS(pi) array
        //Minimum characters needed to be inserted in the beginning to make it
palindromic
        //Check for Anagrams
        //Count and Say
        //Compare version numbers
    }
    class Day_17_BinaryTree{
        // Inorder Traversal
        class Solution {
            public List<Integer> inorderTraversal(TreeNode root) {
                ArrayList<Integer> result=new ArrayList<>();
                inOrder(root,result);
                return result;
            }
            public void inOrder(TreeNode root,ArrayList<Integer> result){
                if(root==null){
                    return ;
                }
                inOrder(root.left,result);
                result.add(root.val);
                inOrder(root.right,result);
            }
        }
        // Preorder Traversal
        class Solution {
            public List<Integer> preorderTraversal(TreeNode root) {
                ArrayList<Integer> result=new ArrayList<>();
                preOrder(root,result);
                return result;
            }
            public void preOrder(TreeNode root,ArrayList<Integer> result){
                if(root==null){
                    return ;
```

```
        }
        result.add(root.val);
        preOrder(root.left,result);
        preOrder(root.right,result);
    }
}
// Postorder Traversal
class Solution {
    public List<Integer> postorderTraversal(TreeNode root) {
        ArrayList<Integer> result=new ArrayList<>();
        postOrder(root,result);
        return result;
    }
    public void postOrder(TreeNode root,ArrayList<Integer> result){
        if(root==null){
            return ;
        }
        postOrder(root.left,result);
        postOrder(root.right,result);
        result.add(root.val);
    }
}
// Morris Inorder Traversal
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> inorder=new ArrayList<Integer>();
        TreeNode cur=root;
        while(cur!=null){
            if(cur.left==null){
                inorder.add(cur.val);
                cur=cur.right;
            }
            else{
                TreeNode prev=cur.left;
                while(prev.right!=null&&prev.right!=cur){
                    prev=prev.right;
                }
                if(prev.right==null) {
                    prev.right=cur;
                    cur=cur.left;
                }
                else{
                    prev.right=null;
                    inorder.add(cur.val);
                    cur=cur.right;
                }
            }
        }
        return inorder;
```

```
    }

}
// Morris Preorder Traversal
class Solution{
    static ArrayList<Integer> preorderTraversal(Node root){
        ArrayList<Integer> preorder=new ArrayList<>();
        Node cur=root;
        while(cur!=null){
            if(cur.left==null){
                preorder.add(cur.data);
                cur=cur.right;
            } else {
                Node prev=cur.left;
                while(prev.right!=null&&prev.right!=cur) {
                    prev = prev.right;
                }

                if(prev.right==null){
                    prev.right=cur;
                    preorder.add(cur.data);
                    cur=cur.left;
                }else{
                    prev.right=null;
                    cur=cur.right;
                }
            }
        }
        return preorder;
    }
}
// LeftView Of Binary Tree
class Solution{
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result=new ArrayList<Integer>();
        rightView(root,result,0);
        return result;
    }

    public void rightView(TreeNode curr,List<Integer> result,int currDepth){
        if(curr==null){return;}
        if(currDepth==result.size()){result.add(curr.val);}
        rightView(curr.right,result,currDepth+1);
        rightView(curr.left,result,currDepth+1);
    }
    public List<Integer> lightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
        leftView(root,result,0);
        return result;
```

```
        }

        public void leftView(TreeNode curr,List<Integer> result,int currDepth){
            if(curr==null){return;}
            if(currDepth==result.size()){result.add(curr.val);}
            leftView(curr.left,result,currDepth + 1);
            leftView(curr.right,result,currDepth + 1);
        }
    }
    // Bottom View of Binary Tree
    class Solution{
        static ArrayList<Integer> BottomView(Node root)
        {
            ArrayList<Integer> ans=new ArrayList<>();
            if(root==null)return ans;
            Map<Integer,Integer> map=new TreeMap<>();
            Queue<Pair> q=new LinkedList<Pair>();
            q.add(new Pair(root,0));
            while(!q.isEmpty()){
                Pair it=q.remove();
                int hd=it.hd;
                Node temp=it.node;
                map.put(hd,temp.data);
                if(temp.left!=null){q.add(new Pair(temp.left,hd-1));}
                if(temp.right!=null){q.add(new Pair(temp.right,hd+1));}
            }
            for (Map.Entry<Integer,Integer> entry : map.entrySet()) {
                ans.add(entry.getValue());
            }
            return ans;

    }
}
    // Top View of Binary Tree
    class Solution{
        static ArrayList<Integer> topView(Node root)
        {
            ArrayList<Integer> ans=new ArrayList<>();
            if(root==null)return ans;
            Map<Integer,Integer> map=new TreeMap<>();
            Queue<Pair> q=new LinkedList<Pair>();
            q.add(new Pair(root,0));
            while(!q.isEmpty()){
                Pair it=q.remove();
                int hd=it.hd;
                Node temp=it.node;
                if(map.get(hd)==null)map.put(hd,temp.data);
                if(temp.left!=null){q.add(new Pair(temp.left,hd-1));}
                if(temp.right!=null){q.add(new Pair(temp.right,hd+1));}
```

```
            }
            for (Map.Entry<Integer,Integer> entry : map.entrySet()) {
                ans.add(entry.getValue());
            }
            return ans;

        }}
        // Preorder inorder postorder in a single traversal
        // Vertical order traversal
        class Solution{
            class Tuple{
                TreeNode node;
                int row;
                int col;
                Tuple(TreeNode _node,int _row,int _col){
                    this.node=_node;
                    this.row=_row;
                    this.col=_col;
                }
            }
            public List<List<Integer>> verticalTraversal(TreeNode root){
                List<List<Integer>> list=new ArrayList<>();
                TreeMap<Integer,TreeMap<Integer,PriorityQueue<Integer>>> map=new
TreeMap<>();
                q.offer(new Tuple(root,0,0));
                while(!q.isEmpty()){
                    Tuple tup=q.peek();
                    TreeNode Node=tup.node;
                    int x=tup.row;
                    int y=tup.col;

                    if(!map.containsKey(x)){
                        map.put(x,new TreeMap<>());
                    }
                    if(map.get(x).containsKey(y)){
                    map.get(x).put(y,new PriorityQueue<Integer>());
                    }
                    map.get(x).get(y).offer(Node.val);
                    if(Node.left!=null){
                        q.offer(new Tuple(Node.left,x-1,y+1));
                    }
                    if(Node.right!=null){
                        q.offer(new Tuple(Node.left,x+1,y+1));
                    }
                }
                for(TreeMap<Integer,PriorityQueue<Integer>> ys: map.values()){
                    list.add(new ArrayList<>());
                    for(PriorityQueue<Integer>> nodes: ys.values())
                        while(!nodes.isEmpty()){
```

```
                System.out.println(nodes.peek());
                list.get(list.size()-1).add(nodes.poll());
            }
        }
        return list;
    }
}
// Root to node path in a Binary Tree
class Solution{
    static boolean getPath(Node root, ArrayList < Integer > arr, int x) {
        if(root==null)return false;
        arr.add(root.data);
        if(root.data==x)
            return true;
        if(getPath(root.left,arr,x)||getPath(root.right,arr,x))
            return true;
        arr.remove(arr.size()-1);
        return false;
    }
}
// Max width of a Binary Tree
class Solution{
    class Pair{
        TreeNode node;
        int num;
        Pair(TreeNode _node,int _num) {
            num=_num;
            node=_node;
        }
    }
    public static int widthOfBinaryTree(TreeNode root){
        if(root==null)return 0;
        int ans=0;
        Queue<Pair> q=new LinkedList<>();
        q.offer(new Pair(root,0));
        while(!q.isEmpty()){
            int size=q.size();
            int mmin=q.peek().num;    //to make the id starting from zero
            int first=0,last=0;
            for(int i=0;i<size;i++){
                int cur_id=q.peek().num-mmin;
                TreeNode node=q.peek().node;
                q.poll();
                if(i==0) first=cur_id;
                if(i==size-1) last=cur_id;
                if(node.left!=null)
                    q.offer(new Pair(node.left,cur_id*2+1));
                if(node.right != null)
                    q.offer(new Pair(node.right,cur_id*2+2));
```

```
            }
            ans=Math.max(ans,last-first+1);
        }
        return ans;
    }
  }
}
class Day_18_BinaryTree{
  // Level order Traversal / Level order traversal in spiral form
  class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        Queue<TreeNode> q=new LinkedList<TreeNode>();
        List<List<Integer>> wraplist=new LinkedList<List<Integer>>();
        if(root==null){
            return wraplist;
        }
        q.offer(root);
        while(!q.isEmpty()){
            int size=q.size();
            List<Integer> sublist=new LinkedList<Integer>();
            for(int i=0;i<size;i++){
                if(q.peek().left!=null){q.offer(q.peek().left);}
                if(q.peek().right!=null){q.offer(q.peek().right);}
                sublist.add(q.poll().val);
            }
            wraplist.add(sublist);
        }
        return wraplist;
    }
  }
  // Height of a Binary Tree
  class Solution{
    public int heightOfBinaryTree(TreeNode root){
        if(root==null){
            return 0;
        }
        int lh=heightOfBinaryTree(root.left);
        int rh=heightOfBinaryTree(root.right);
        return 1+Math.max(lh,rh);
    }
  }
  // Diameter of Binary Tree
  class Solution{
    public int diameterOfBinaryTree(TreeNode root){
        int []diameter=new int[]{0};
        heightOfBinaryTree(root,diameter);
        return diameter[0];
    }
    public int heightOfBinaryTree(TreeNode root,int diameter[]){
```

```
            if(root==null){
                return 0;
            }
            int lh=heightOfBinaryTree(root.left,diameter);
            int rh=heightOfBinaryTree(root.right,diameter);
            diameter[0]=Math.max(diameter[0],lh+rh);
            return 1+Math.max(lh,rh);
        }
    }
    // Check if the Binary tree is height-balanced or not
    class Solution{
        public boolean isBalanced(TreeNode root){
            return dfsHeight(root)!=-1
        }
        public int dfsHeight(TreeNode root){
            if(root==null){
                return 0;
            }
            int lh=dfsHeight(root.left);
            if(lh==-1)return -1;
            int rh=dfsHeight(root.right);
            if(rh==-1)return -1;
            if(Math.abs(lh-rh)>1)return -1;
            return 1+Math.max(lh,rh);
        }
    }
    // LCA in Binary Tree
    class Solution{
        public TreeNode lowestCommonAncestor(TreeNode root,TreeNode p,
TreeNode q){
            //base case
            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){

                                                                    return
right;
            }
            else if(right==null){
                return left;
            }
            else { //both left and right are not null, we found our result
                return root;
            }
        }
    }
```

```
// Check if two trees are identical or not
class Solution {
    static boolean isIdentical(Node node1,Node node2){
        if(node1==null&&node2==null)
            return true;
        else if(node1==null||node2==null)
            return false;

        return ((node1.data==node2.data)&&isIdentical(node1.left,node2.left)
&&isIdentical(node1.right,node2.right));
    }
}
// Zig Zag Traversal of Binary Tree
class Solution {
    public static ArrayList<ArrayList<Integer>> zigzagLevelOrder(Node root){
        Queue<Node> queue=new LinkedList<Node>();
        ArrayList<ArrayList< Integer>> wrapList=new ArrayList<>();

        if (root == null) return wrapList;

        queue.offer(root);
        boolean flag=true;
        while(!queue.isEmpty()){
            int levelNum=queue.size();
            ArrayList<Integer> subList=new ArrayList<Integer>(levelNum);
            for (int i=0;i<levelNum;i++) {
                int index=i;
                if (queue.peek().left!=null) queue.offer(queue.peek().left);
                if (queue.peek().right!=null) queue.offer(queue.peek().right);
                if (flag==true)subList.add(queue.poll().val);
                else subList.add(0,queue.poll().val);
            }
            flag=!flag;
            wrapList.add(subList);
        }
        return wrapList;
    }
}
// Boundary Traversal of Binary Tree
class Solution{
    static Boolean isLeaf(Node root){
        return (root.left==null)&&(root.right==null);
    }

    static void addLeftBoundary(Node root,ArrayList<Integer> res){
        Node cur=root.left;
        while(cur!=null){
            if(isLeaf(cur)==false)res.add(cur.data);
            if(cur.left!=null)cur=cur.left;
```

```
            else cur=cur.right;
        }
    }
    static void addRightBoundary(Node root,ArrayList<Integer> res){
        Node cur=root.right;
        ArrayList<Integer> tmp=new ArrayList <Integer>();
        while(cur!=null){
            if(isLeaf(cur)==false)tmp.add(cur.data);
            if(cur.right!=null)cur=cur.right;
            else cur=cur.left;
        }
        int i;
        for(i=tmp.size()- 1;i>=0;--i){
            res.add(tmp.get(i));
        }
    }

    static void addLeaves(Node root, ArrayList<Integer>res){
        if(isLeaf(root)){
            res.add(root.data);
            return;
        }
        if(root.left!=null)addLeaves(root.left,res);
        if(root.right!=null)addLeaves(root.right,res);
    }
    static ArrayList<Integer> printBoundary(Node node) {
        ArrayList<Integer> ans=new ArrayList<Integer>();
        if (isLeaf(node)==false)ans.add(node.data);
        addLeftBoundary(node,ans);
        addLeaves(node,ans);
        addRightBoundary(node,ans);
        return ans;
    }
  }
}
class Day_19_BinaryTree{
  //Maximum path sum
  class Solution{
    public static int maxPathSum(Node root){
        int maxValue[]=new int[1];
        maxValue[0]=Integer.MIN_VALUE;
        maxPathDown(root,maxValue);
        return maxValue[0];
    }

    public static int maxPathDown(Node node,int maxValue[]){
        if (node==null) return 0;
        int left=Math.max(0,maxPathDown(node.left,maxValue));
        int right=Math.max(0,maxPathDown(node.right,maxValue));
```

```
            maxValue[0]=Math.max(maxValue[0],left+right+node.val);
            return Math.max(left,right)+node.val;
        }
    }
     //Construct Binary Tree from inorder and preorder
    class Solution{
        static TreeNode buildTree(int[] preorder,int[] inorder) {
          Map<Integer,Integer> inMap=new HashMap<Integer,Integer>();
          for (int i=0;i<inorder.length;i++) {
            inMap.put(inorder[i],i);
          }
          TreeNode root=buildTree(preorder,0,preorder.length - 1,inorder,0,inorder.
length-1,inMap);
          return root;
        }
        static TreeNode buildTree(int[] preorder,int preStart,int preEnd,int[]inorder,int
inStart,int inEnd,Map < Integer, Integer > inMap) {
          if (preStart>preEnd||inStart>inEnd) return null;
          TreeNode root=new TreeNode(preorder[preStart]);
          int inRoot=inMap.get(root.val);
          int numsLeft=inRoot-inStart;
          root.left=buildTree(preorder,preStart+1,preStart+numsLeft,inorder,inStart,
inRoot-1,inMap);
          root.right=buildTree(preorder,preStart+numsLeft+1,preEnd,inorder,inRoot+1,
inEnd,inMap);
          return root;
        }
    }
     //Construct Binary Tree from Inorder and Postorder
    class Solution{
        public TreeNode buildTree(int[] inorder,int[] postorder){
            if (inorder==null||postorder==null||inorder.length!=postorder.length)
                return null;
            HashMap<Integer,Integer> hm=new HashMap<Integer,Integer>();
            for (int i=0;i<inorder.length;++i)
                hm.put(inorder[i],i);
            return buildTreePostIn(inorder,0,inorder.length-1,postorder,0,postorder.
length-1,hm);
        }

        private TreeNode buildTreePostIn(int[] inorder,int is,int ie,int[] postorder,int ps,
int pe,HashMap<Integer,Integer> hm){
            if(ps>pe||is>ie) return null;
            TreeNode root=new TreeNode(postorder[pe]);
            int inroot=hm.get(postorder[pe]);
            int numsLeft=inroot-is;
            root.left=buildTreePostIn(inorder,is,inroot-1,postorder,ps,ps+numsLeft-1,hm);
            root.right=buildTreePostIn(inorder,inroot+1,ie,postorder,ps+numsLeft,pe-1,
hm);
```

```
            return root;
        }
    }
     //Symmetric Binary Tree
    class Solution{
        public boolean isSymmetric(TreeNode root){
            return root==null||isSymmetricHelp(root.left,root.right);
        }
        private boolean isSymmetricHelp(TreeNode left,TreeNode right){
            if(left==null||right==null)return left==right;
            if(left.val!=right.val)return false;
            return isSymmetricHelp(left.left,right.right)&&isSymmetricHelp(left.right,right.
left);
        }
    }
     //Flatten Binary Tree to LinkedList
    class Solution {
        static Node prev=null;
        static void flatten(Node root){
            if(root==null)return;
            flatten(root.right);
            flatten(root.left);
            root.right=prev;
            root.left=null;
            prev=root;
         }
        static Node prev=null;
        static void flatten(Node root){
            if(root==null)return;
            Stack<Node > st=new Stack<>();
            st.push(root);
            while(!st.isEmpty()){
                Node  cur=st.peek();
                st.pop();

                if(cur.right!=null){
                st.push(cur.right);
                }
                if(cur.left!=null){
                st.push(cur.left);
                }
                if(!st.isEmpty()){
                cur.right=st.peek();
                }
                cur.left=null;
            }
        }
        static ArrayList<Integer> preorderTraversal(Node root){
            ArrayList<Integer> preorder=new ArrayList<>();
```

```
            Node cur=root;
            while(cur!=null){
                if(cur.left==null){
                    preorder.add(cur.data);
                    cur=cur.right;
                }else{
                    Node prev=cur.left;
                    while(prev.right!=null&&prev.right!=cur){
                        prev=prev.right;
                    }

                    if (prev.right==null){
                        prev.right=cur;
                        preorder.add(cur.data);
                        cur=cur.left;
                    }else{
                        prev.right=null;
                        cur=cur.right;
                    }
                }
            }
            return preorder;
        }
}
     //Check if Binary Tree is the mirror of itself or not
    class Solution {
        // Function to convert a binary tree into its mirror tree.
            Node mirrorutil(Node node){
            if(node==null){
                return node;
            }
            Node L=mirrorutil(node.left);
            Node R=mirrorutil(node.right);

            node.left=R;
            node.right=L;

            return node;
            }
        }
      //Check for Children Sum Property
    class Solution{
        static void reorder(Node root){
          if (root==null) return;
          int child=0;
          if(root.left!=null){
            child+=root.left.data;
          }
          if(root.right!=null){
```

```
            child+=root.right.data;
          }

          if(child<root.data){
            if(root.left!=null) root.left.data=root.data;
            else if(root.right!=null) root.right.data=root.data;
          }

          reorder(root.left);
          reorder(root.right);

          int tot = 0;
          if (root.left!=null) tot+=root.left.data;
          if (root.right!=null) tot+=root.right.data;
          if (root.left!=null || root.right!=null)root.data=tot;
        }}
}
class Day_20_BinarySearchTree{
    //Populate Next Right pointers of Tree
    class Solution{
        public Node connect(Node root) {
        Queue<Node> q=new LinkedList<>();
            q.add(root);
            Node temp=null;
            while(!q.isEmpty()){
                int n=q.size();
                for(int i=0;i<n;i++){
                    Node prev=temp;
                    temp=q.poll();
                    if(i==0){prev=temp;}
                    if(i>0)prev.next = temp;
                    if(temp!=null){
                        if(temp.left!=null)
                            q.add(temp.left);
                        if(temp.right!=null)
                            q.add(temp.right);
                    }
                }
                if(temp!=null)temp.next=null;
            }
            return root;
        }
    }
    //Search given Key in BST
    class Solution{
        //O(Log N)
        public TreeNode searchBST(TreeNode root, int val) {
            while(root!=null&&root.val!=val){
                root=val<root.val?root.left:root.right;
```

```
        }
        return root;
    }
}
//Construct BST from given keys
class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        TreeNode root=createArraytoBST(nums,0,nums.length-1);
        return root;
    }
    public TreeNode createArraytoBST(int []nums,int start,int end){
        if(start>end){
            return null;
        }
        int mid=(start+end)/2;
        TreeNode root=new TreeNode(nums[mid]);
        root.left=createArraytoBST(nums,start,mid-1);
        root.right=createArraytoBST(nums,mid+1,end);
        return root;
    }
}
//Construct BST from preorder traversal
class Solution {
    public TreeNode bstFromPreorder(int[] A) {
        return bstFromPreorder(A,Integer.MAX_VALUE,new int[]{0});
    }
    public TreeNode bstFromPreorder(int[] A,int bound,int[] i){
        if(i[0]==A.length||A[i[0]]>bound) return null;
        TreeNode root=new TreeNode(A[i[0]++]);
        root.left=bstFromPreorder(A,root.val,i);
        root.right=bstFromPreorder(A,bound,i);
        return root;
    }
}
//Check is a BT is BST or not
class Solution {
    private boolean checkBST(TreeNode node,long min,long max) {
        if(node==null)return true;
        if(node.val<=min||node.val>=max)return false;

        if(checkBST(node.left,min,node.val)&&checkBST(node.right,node.val,max)){
            return true;
        }
        return false;
    }
    public boolean isValidBST(TreeNode root) {
        return checkBST(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }
}
```

```
    //Find LCA of two nodes in BST
    class Solution {
        public TreeNode lowestCommonAncestor(TreeNode root,TreeNode p,
TreeNode q){
            if(root==null)return null;
            int curr=root.val;
            if(curr<p.val&&curr<q.val) {
                return lowestCommonAncestor(root.right,p,q);
            }
            if(curr>p.val&&curr>q.val) {
                return lowestCommonAncestor(root.left,p,q);
            }
            return root;
        }
    }
    //Find the inorder predecessor/successor of a given Key in BST.
    class Solution {
        public TreeNode inorderSuccessor(TreeNode root,TreeNode p){
            TreeNode successor = null;
            while (root != null) {
                if (p.val>=root.val) {
                    root=root.right;
                }else{
                    successor=root;
                    root=root.left;
                }
            }
            return successor;
        }
        public TreeNode inorderPredecessor(TreeNode root,TreeNode p){
            TreeNode predecessor=null;
            while(root!=null) {
                if (p.val<=root.val) {
                    root=root.left;
                }else{
                    predecessor=root;
                    root=root.right;
                }
            }
            return predecessor;
        }
    }
}
class Day_21_BinarySearchTree{
    //Floor in a BST
    class Solution {
        public static int floorInBST(TreeNode<Integer> root,int key) {
            int floor=-1;
            while(root!=null){
```

```
            if(root.data==key){
                floor=root.data;
                return floor;
            }
            if(key>root.data) {
                floor=root.data;
                root=root.right;
            }
            else {
                root=root.left;
            }
        }
        return floor;
    }
}
//Ceil in a BST
class Solution {
    public static int findCeil(TreeNode<Integer> root,int key) {
        int ceil=-1;
        while(root!=null){
            if(root.data==key){
                ceil=root.data;
                return ceil;
            }

            if(key>root.data){
                root=root.right;
            }
            else{
                ceil=root.data;
                root=root.left;
            }
        }
        return ceil;
    }
}
//Find K-th smallest element in BST
class Solution{
    static Node kthsmallest(Node root,int k[])
    {
        if(root==null){return null;}
        Node left=kthsmallest(root.left,k);
        if(left!=null){return left;}
        k[0]--;
        if(k[0]==0){return root;}
        return kthsmallest(root.right,k);
    }
}
//Find K-th largest element in BST
```

```
class Solution{
    static Node kthlargest(Node root,int k[])
    {
    if(root==null)return null;
    Node right=kthlargest(root.right,k);
    if(right!=null){return right;}
    k[0]--;
    if(k[0]==0){return root;}
    return kthlargest(root.left,k);
    }
}
//Find a pair with a given sum in BST
class Solution{
    class BSTIterator {
        private Stack<TreeNode> stack=new Stack<TreeNode>();
        boolean reverse=true;

        public BSTIterator(TreeNode root,boolean isReverse){
            reverse=isReverse;
            pushAll(root);
        }
        /** @return whether we have a next smallest number */
        public boolean hasNext() {
            return !stack.isEmpty();
        }

        /** @return the next smallest number */
        public int next() {
            TreeNode tmpNode=stack.pop();
            if(reverse==false) pushAll(tmpNode.right);
            else pushAll(tmpNode.left);
            return tmpNode.val;
        }

        private void pushAll(TreeNode node){
            while(node!=null){
                stack.push(node);
                if(reverse==true){
                    node=node.right;
                }else{
                    node=node.left;
                }
            }
        }
    }
    class Solution {
        public boolean findTarget(TreeNode root,int k){
            if(root==null) return false;
            BSTIterator l=new BSTIterator(root,false);
```

```
            BSTIterator r=new BSTIterator(root,true);

            int i=l.next();
            int j=r.next();
            while(i<j){
                if(i+j==k) return true;
                else if(i+j<k) i=l.next();
                else j=r.next();
            }
            return false;
        }
    }
}
//BST iterator
class Solution{
    public class BSTIterator {
        private Stack<TreeNode> stack = new Stack<TreeNode>();
        public BSTIterator(TreeNode root){
            pushAll(root);
        }
        /** @return whether we have a next smallest number */
        public boolean hasNext(){
            return !stack.isEmpty();
        }
        /** @return the next smallest number */
        public int next(){
            TreeNode tmpNode=stack.pop();
            pushAll(tmpNode.right);
            return tmpNode.val;
        }

        private void pushAll(TreeNode node) {
            for (;node!=null;stack.push(node),node=node.left);
        }
    }
}
//Size of the largest BST in a Binary Tree
class Solution {
    class NodeValue {
        public int maxNode, minNode, maxSize;
        NodeValue(int minNode,int maxNode,int maxSize){
            this.maxNode=maxNode;
            this.minNode=minNode;
            this.maxSize=maxSize;
        }
    }
    private NodeValue largestBSTSubtreeHelper(TreeNode root){
        if(root==null) {
            return new NodeValue(Integer.MAX_VALUE,Integer.MIN_VALUE,0);
```

```
            }
            NodeValue left = largestBSTSubtreeHelper(root.left);
            NodeValue right = largestBSTSubtreeHelper(root.right);
            if (left.maxNode < root.val && root.val < right.minNode) {
                return new NodeValue(Math.min(root.val,left.minNode),Math.max(root.val,
right.maxNode),left.maxSize+right.maxSize+1);
            }
            return new NodeValue(Integer.MIN_VALUE,Integer.MAX_VALUE,Math.
max(left.maxSize, right.maxSize));
        }
        public int largestBSTSubtree(TreeNode root) {
            return largestBSTSubtreeHelper(root).maxSize;
        }
    }
    //Serialize and deserialize Binary Tree
    class Solution{
        public String serialize(TreeNode root){
            if(root==null)return "";
            Queue<TreeNode> q=new LinkedList<>();
            StringBuilder res=new StringBuilder();
            q.add(root);
            while (!q.isEmpty()){
                TreeNode node=q.poll();
                if(node==null) {
                    res.append("n ");
                    continue;
                }
                res.append(node.val+" ");
                q.add(node.left);
                q.add(node.right);
            }
            return res.toString();
        }

        public TreeNode deserialize(String data) {
            if(data=="") return null;
            Queue<TreeNode> q=new LinkedList<>();
            String[] values=data.split(" ");
            TreeNode root=new TreeNode(Integer.parseInt(values[0]));
            q.add(root);
            for (int i=1;i<values.length;i++){
                TreeNode parent=q.poll();
                if(!values[i].equals("n")) {
                    TreeNode left=new TreeNode(Integer.parseInt(values[i]));
                    parent.left=left;
                    q.add(left);
                }
                if (!values[++i].equals("n")) {
                    TreeNode right=new TreeNode(Integer.parseInt(values[i]));
```

```
                parent.right=right;
                q.add(right);
            }
        }
        return root;
    }
  }
}
class Day_22_BinarySearchTree{
    //Binary Tree to Double Linked List
    class Solution {
        TreeNode head=null;
       public  void flatten(TreeNode root){
          //Write code here
          if(root==null){return;}
          flatten(root.right);
          flatten(root.left);
          root.right=head;
          root.left=null;
          head=root;
       }
   }
    //Find median in a stream of running integers.
    //K-th largest element in a stream.
    //Distinct numbers in Window.
    //K-th largest element in an unsorted array.
    //Flood-fill Algorithm
}
class Day_23_Graphs{
    //Clone a graph (Not that easy as it looks)
    class Solution {
        public void dfs(Node node ,Node copy ,Node[] visited){
            visited[copy.val] = copy;
            for(Node n : node.neighbors){
                if(visited[n.val] == null){
                    Node newNode = new Node(n.val);
                    copy.neighbors.add(newNode);
                    dfs(n,newNode,visited);
                }
                else{
                    copy.neighbors.add(visited[n.val]);
                }
            }

        }
        public Node cloneGraph(Node node) {
            if(node==null)return null;
            Node copy=new Node(node.val);
            Node[] visited=new Node[101];
```

```
            Arrays.fill(visited,null);
            dfs(node,copy ,visited);
            return copy;
        }
    }
     //DFS
    class Solution{
        public void dfs(int src,boolean []vis,ArrayList<ArrayList<Integer>> adj,
ArrayList<Integer> dfs){
            vis[src]=true;
            dfs.add(src);
            for(Integer it: adj.get(src)){
                if(visited[it]==false){
                    dfs(it,vis,adj,dfs);
                }
            }
        }
        public ArrayList<Integer> dfsOfGraph(int V,ArrayList<ArrayList<Integer>> adj){
            ArrayList<Integer> dfs=new ArrayList<>();
            boolean vis[]=new boolean[V];
            for(int i=0;i<V;i++){
                vis[i]=false;
            }
            visited[src]=true;
            dfs(src,vis,adj,dfs);
            return dfs;
        }
        }
     //BFS
    class Solution{
        public ArrayList<Integer> bfsOfGraph(int V,ArrayList<ArrayList<Integer>> adj){
        ArrayList<Integer> bfs=new ArrayList<>();
        boolean vis[]=new boolean[V];
        for(int i=0;i<V;i++){
            vis[i]=false;
        }
        Queue<Integer> q=new LinkedList<>();
        q.add(src);
        vis[src]=true;
        while(!q.isEmpty()){
            Integer node=q.poll();
            bfs.add(node);
            for(Integer it: adj.get(node)){
                if(vis[it]==false){
                    vis[it]=true;
                    q.add(it);
                }
            }
        }
```

```
        return bfs;
    }
    }
     //Detect A cycle in Undirected Graph using BFS
    class Solution{
        public boolean checkforCycle(int src,int V,boolean vis[],
ArrayList<ArrayList<Integer>> adj,boolean vis[]){
            vis[src]=true;
            Queue<Pair> q=new LinkedList<>();
            q.add(new Pair(src,-1));
            while(!q.isEmpty()){
                int node=q.peek().first;
                int parent=q.peek().second;
                q.remove();
                for(Integer adjacentNode: adj.get(node))
                {
                    if(vis[adjacentNode]==false){
                        vis[adjacentNode]=true;
                        q.add(new Pair(adjacentNode,node));
                    }
                    else if(parent!=adjacentNode){
                        return true;
                    }
                }
            }
            return false;
        }
        public boolean isCyclic(int V,ArrayList<ArrayList<Integer>> adj){
        boolean vis[]=new boolean[V];
        for(int i=0;i<V;i++){
            vis[i]=false;
        }
        for(int i=0;i<V;i++){
            if(vis[i]==false){
                if(checkCycle(i,V,adj,vis)==true)
                return true;
            }
        }
        return false;
        }
    }
     //Detect A cycle in Undirected Graph using DFS
    class Solution{
        public boolean dfs(int node,int parent,boolean vis[],
ArrayList<ArrayList<Integer>> adj){
            vis[src]=true;
            for(Integer adjacentNode : adj.get(node)){
                if(dfs(adjacentNode,node,vis,adj)==true){
                    return true;
```

```
                }
                else if((adjacentNode!=parent)){
                    return true;
                }
            }
            return false;
        }
        public boolean isCyclic(int V,ArrayList<ArrayList<Integer>> adj){
        boolean vis[]=new boolean[V];
        for(int i=0;i<V;i++){
            vis[i]=false;
        }
        for(int i=0;i<V;i++){
            if(vis[i]==false){
                if(dfs(i,-1,adj,vis)==true)
                return true;
            }
        }
        return false;
        }
    }
      //Detect A cycle in a Directed Graph using DFS
    class Solution{
        private boolean dfsCheck(int node,ArrayList<ArrayList<Integer>> adj,boolean
vis[],boolean pathVis[]) {
        vis[node]=true;
        pathVis[node]=true;
        for(Integer it : adj.get(node)) {
            if(vis[it]==false){
                if(dfsCheck(it,adj,vis,pathVis)==true)
                    return true;
            }
            else if(pathVis[it]==true){
                return true;
            }
        }
        pathVis[node]=false;
        return false;
    }
    public boolean isCyclic(int V,ArrayList<ArrayList<Integer>> adj) {
        boolean vis[]=new boolean[V];
        boolean pathVis[] = new boolean[V];

        for(int i=0;i<V;i++){
            if(vis[i]==false) {
                if(dfsCheck(i,adj,vis,pathVis)==true)return true;
            }
        }
        return false;
```

```
    }
}
    //Detect A cycle in a Directed Graph using BFS
    class Solution {
        public boolean isCyclic(int N,ArrayList<ArrayList<Integer>> adj){
            // int topo[] = new int[N];
            int indegree[]=new int[N];
            for(int i=0;i<N;i++){
                for(Integer it : adj.get(i)){
                    indegree[it]++;
                }
            }

            Queue<Integer> q=new LinkedList<Integer>();
            for(int i=0;i<N;i++){
                if(indegree[i]==0){
                    q.add(i);
                }
            }
            int cnt=0;
            while(!q.isEmpty()){
                Integer node=q.poll();
                cnt++;
                for (Integer it : adj.get(node)){
                    indegree[it]--;
                    if (indegree[it]==0){
                        q.add(it);
                    }
                }
            }
            if(cnt==N)
                return false;
            return true;
        }
    }
    //Topological Sort BFS
    class Solution{
        public boolean topoSort(int V,ArrayList<ArrayList<Integer>> adj){
            int indegree[]=new int[V];
            for(int i=0;i<V;i++){
                for(Integer it:adj.get(i)){
                    indegree[it]++;
                }
            }
            Queue<Integer> q=new LinkedList<>();
            for(int i=0;i<V;i++){
                if(indegree[i]==0){
                    q.add(i);
                }
```

```
            }
            int i=0;
            int topo[]=new int[V];
            while(!q.isEmpty()){
                int node=q.peek();
                q.remove();
                topo[i++]=node;
                for(Integer it: adj.get(node)){
                    indegree[it]--;
                    if(indegree[it]==0){q.add(it);}
                }
            }
            return topo;
        }
    }
      //Topological Sort DFS
    class Solution{
        public void dfs(int node,boolean vis[],ArrayList<ArrayList<Integer>> adj,
Stack<Integer> st){
            vis[node]=true;
            for(Integer it: adj.get(node))
            {
                if(vis[it]==false){
                    dfs(it,vis,adj,st);
                }
            }
            st.push(node);
        }
        public ArrayList<Integer> dfsOfGraph(int V,ArrayList<ArrayList<Integer>> adj){
        boolean vis[]=new boolean[V];
        Stack<Integer> st=new Stack<>();
        for(int i=0;i<V;i++){
            vis[i]=false;
        }
        for(int i=0;i<V;i++){
            if(vis[i]==false){
                dfs(i,vis,adj,st);
            }
        }
        int ans[]=new int[V];
        int i=0;
        while(!st.isEmpty()){
            ans[i++]=st.peek();
            st.pop();
        }
        return ans;
    }
    }
      //Number of islands(Do in Grid and Graph Both)
```

```
        class Solution {
            class Pair{
                int row;
                int col;
                Pair(int _row,int _col){
                this.row=_row;
                this.col=_col;
                }
            }
            public void dfs(int row,int col,int [][]grid,boolean [][]visited,int []dx,int []dy,
ArrayList<String> res,int sr,int sc,int n,int m){
                    visited[row][col]=true;
                    res.add(toString(row-sr,col-sc));

                    for(int i=0;i<4;i++){
                        int nr=row+dx[i];
                        int nc=col+dy[i];

if(nr>=0&&nr<n&&nc>=0&&nc<m&&visited[nr][nc]==false&&grid[nr][nc]==1){
                            dfs(nr,nc,grid,visited,dx,dy,res,sr,sc,n,m);
                        }
                    }
                }
                public String toString(int r,int c){
                    return Integer.toString(r)+" "+Integer.toString(c);
                }
            int countDistinctIslands(int[][] grid) {
                // Your Code here
                    int n=grid.length;
                    int m=grid[0].length;
                    boolean visited[][]=new boolean[n][m];
                    for(int i=0;i<n;i++){
                        for(int j=0;j<m;j++){
                            visited[i][j]=false;
                        }
                    }
                    int dx[]={-1,0,1,0};
                    int dy[]={0,1,0,-1};
                    HashSet<ArrayList<String>> hs=new HashSet<>();
                    for(int i=0;i<n;i++){
                        for(int j=0;j<m;j++){
                            if(visited[i][j]==false && grid[i][j]==1){
                            ArrayList<String> res=new ArrayList<>();
                            dfs(i,j,grid,visited,dx,dy,res,i,j,n,m);
                            hs.add(res);
                        }
                    }
                }
                return hs.size();
```

```
            }
        }
          //Bipartite Check using BFS
        class Solution
{
    public boolean check(int start,int V,ArrayList<ArrayList<Integer>> adj,int []color){
        Queue<Integer> q=new LinkedList<>();
        q.add(start);
        color[start]=0;
        while(!q.isEmpty()){
            int node=q.peek();
            q.remove();

            for(Integer it: adj.get(node)){
                if(color[it]==-1){
                    color[it]=1-color[node];
                    q.add(it);
                }
                else if(color[it]==color[node]){
                    return false;
                }
            }
        }
        return true;
    }
    public boolean isBipartite(int V, ArrayList<ArrayList<Integer>>adj)
    {
        // Code here
        int color[]=new int[V];
        for(int i=0;i<V;i++)
        {
            color[i]=-1;
        }
        for(int i=0;i<V;i++){
            if(color[i]==-1){
                if(check(i,V,adj,color)==false){
                return false;
                }
            }
        }
        return true;
    }
}
          //Bipartite Check using DFS
        class Solution{
            public boolean check(int start,int c,ArrayList<ArrayList<Integer>> adj,int []color){
                color[start]=c;
                for(Integer it: adj.get(start)){
                    if(color[it]==-1){
```

```
                if(check(it,1-c,adj,color)==false){
                    return false;
                    }
                }
            else if(color[it]==color[start]){
                    return false;
                }
            }
        return true;
    }
    public boolean isBipartite(int V, ArrayList<ArrayList<Integer>>adj)
    {
        // Code here
        int color[]=new int[V];
        for(int i=0;i<V;i++)
        {
            color[i]=-1;
        }
        for(int i=0;i<V;i++){
            if(color[i]==-1){
                if(check(i,0,adj,color)==false){
                return false;
                }
            }
        }
        return true;
    }
  }
  class Day_24_Graphs{
    //Strongly Connected Component(using KosaRaju's algo)
    class Solution {
        private void dfs(int node,int []vis,ArrayList<ArrayList<Integer>> adj,
Stack<Integer> st){
            vis[node]=1;
            for(Integer it: adj.get(node)){
                if(vis[it]==0){
                    dfs(it,vis,adj,st);
                }
            }
            st.push(node);
        }
        private void dfs3(int node,int[] vis,ArrayList<ArrayList<Integer>> adjT){
            vis[node]=1;
            for(Integer it: adjT.get(node)){
                if(vis[it]==0){
                    dfs3(it,vis,adjT);
                }
            }
```

```
        }
        //Function to find number of strongly connected components in the graph.
        public int kosaraju(int V, ArrayList<ArrayList<Integer>> adj) {
            int[] vis=new int[V];
            Stack<Integer> st=new Stack<Integer>();
            for (int i=0;i <V;i++){if(vis[i]==0){dfs(i,vis,adj,st);}}
            ArrayList<ArrayList<Integer>> adjT = new ArrayList<ArrayList<Integer>>();
            for(int i=0;i<V;i++){adjT.add(new ArrayList<Integer>());}
            for (int i=0;i<V;i++){
                vis[i]=0;
                for(Integer it: adj.get(i)){
                    // i -> it
                    // it -> i
                    adjT.get(it).add(i);
                }
            }
            int scc=0;
            while(!st.isEmpty()) {
                int node=st.peek();
                st.pop();
                if(vis[node]==0){
                    scc++;
                    dfs3(node,vis,adjT);
                }
            }
            return scc;
        }
    }
     //Dijkstra's Algorithm
    class Solution
{
    class Pair{
        int distance;
        int node;
        Pair(int _dis,int _node){
            this.distance=_dis;
            this.node=_node;
        }
    };
    //Function to find the shortest distance of all the vertices
    //from the source vertex S.
    static int[] dijkstra(int V, ArrayList<ArrayList<ArrayList<Integer>>> adj, int S)
    {
        // Write your code here
        PriorityQueue<Pair> pq=new PriorityQueue<Pair>((x,y)->x.distance-y.distance);
        int distance[]=new int[V];
        for(int i=0;i<V;i++){
            distance[i]=(int)(1e9);
        }
```

```
        distance[S]=0;
        pq.add(new Pair(0,S));
        while(pq.size()!=0){
            int dis=pq.peek().distance;
            int node=pq.peek().node;
            pq.remove();
            for(int i=0;i<adj.get(node).size();i++){
                int edgeW=adj.get(node).get(i).get(1);
                int adjNode=adj.get(node).get(i).get(0);

                if(dis+edgeW<distance[adjNode]){
                    distance[adjNode]=dis+edgeW;
                    pq.add(new Pair(dis+edgeW,adjNode));
                }
            }
        }
        return distance;
    }
}
     //Bellman-Ford Algo
    class Solution {
        static int[] bellman_ford(int V, ArrayList<ArrayList<Integer>> edges, int S) {
            // Write your code here
            int[] distance=new int[V];
            for(int i=0;i<V;i++){
                distance[i]=(int)(1e8);
            }
            distance[S]=0;

            for(int i=0;i<V-1;i++) {
                for(ArrayList<Integer> it: edges){
                    int u=it.get(0);
                    int v=it.get(1);
                    int wt=it.get(2);
                    if(distance[u]!=(int)(1e8)&&distance[u]+wt<distance[v]){
                        distance[v]=distance[u]+wt;
                    }
                }
            }
            for(ArrayList<Integer> it: edges){
                    int u=it.get(0);
                    int v=it.get(1);
                    int wt=it.get(2);
                    if(distance[u]!=(int)(1e8)&&distance[u]+wt<distance[v]){
                        int temp[]=new int[1];
                        temp[0]=-1;
                        return temp;
                        }
                    }
```

```
        return distance;
    }
}
  //Floyd Warshall Algorithm
class Solution{
    public void shortest_distance(int[][] matrix)
    {
        // Code here
        int n=matrix.length;
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if(i==j){
                    matrix[i][j]=0;
                }
                else if(matrix[i][j]==-1){
                    matrix[i][j]=(int)(1e9);
                }
            }
        }
        for(int k=0;k<n;k++){
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    matrix[i][j]=Math.min(matrix[i][j],matrix[i][k]+matrix[k][j]);
                }
            }
        }

        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if(matrix[i][j]==(int)(1e9)){
                    matrix[i][j]=-1;
                }
            }
        }
        return;
    }
}
  //MST using Prim's Algo
class Solution{
    static class Pair{
        int node;
        int distance;
        Pair(int _first,int _second){
            this.node=_first;
            this.distance=_second;
        }
    }
    //Function to find sum of weights of edges of the Minimum Spanning Tree.
    static int spanningTree(int V, ArrayList<ArrayList<ArrayList<Integer>>> adj)
```

```
        {
            // Add your code here
            PriorityQueue<Pair> pq=new PriorityQueue<Pair>((x,y)-> x.distance-y.
distance);
            boolean vis[]=new boolean[V];
            for(int i=0;i<V;i++){
                vis[i]=false;
            }
            pq.add(new Pair(0,0));
            int sum=0;
            while(!pq.isEmpty()){
                int wt=pq.peek().distance;
                int node=pq.peek().node;

                pq.remove();

                if(vis[node]==true)
                    continue;
                vis[node]=true;
                sum+=wt;

                for(int i=0;i<adj.get(node).size();i++){
                    int eW=adj.get(node).get(i).get(1);
                    int adjNode=adj.get(node).get(i).get(0);

                    if(vis[adjNode]==false){
                        pq.add(new Pair(adjNode,eW));
                    }

                }
            }
            return sum;

        }
    }
      //MST using Kruskal's Algo
    class Solution{
        class DisjointSet {
        List<Integer> rank = new ArrayList<>();
        List<Integer> parent = new ArrayList<>();
        List<Integer> size = new ArrayList<>();
        public DisjointSet(int n) {
            for(int i = 0;i<=n;i++) {
                rank.add(0);
                parent.add(i);
                size.add(1);
            }
        }
        public int findUPar(int node) {
```

```
        if(node == parent.get(node)) {
            return node;
        }
        int ulp = findUPar(parent.get(node));
        parent.set(node, ulp);
        return parent.get(node);
    }
    public void unionByRank(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if(ulp_u == ulp_v) return;
        if(rank.get(ulp_u) < rank.get(ulp_v)) {
            parent.set(ulp_u, ulp_v);
        }
        else if(rank.get(ulp_v) < rank.get(ulp_u)) {
            parent.set(ulp_v, ulp_u);
        }
        else {
            parent.set(ulp_v, ulp_u);
            int rankU = rank.get(ulp_u);
            rank.set(ulp_u, rankU + 1);
        }
    }
    public void unionBySize(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if(ulp_u == ulp_v) return;
        if(size.get(ulp_u) < size.get(ulp_v)) {
            parent.set(ulp_u, ulp_v);
            size.set(ulp_v, size.get(ulp_v) + size.get(ulp_u));
        }
        else {
            parent.set(ulp_v, ulp_u);
            size.set(ulp_u, size.get(ulp_u) + size.get(ulp_v));
        }
    }
}
class Edge implements Comparable<Edge>{
        int src,dest,weight;
        Edge(int _src,int _dest,int _weight){
            this.src=_src;this.dest=_dest;this.weight=_weight;
        }
        public int compareTo(Edge compareEdge){
            return this.weight-compareEdge.weight;
        }
    }
    //Function to find sum of weights of edges of the Minimum Spanning Tree.
    static int spanningTree(int V, ArrayList<ArrayList<ArrayList<Integer>>> adj)
    {
```

```
            List<Edge> edges=new ArrayList<Edge>();
            for(int i=0;i<V;i++){
                for(int j=0;j<adj.get(i).size();j++){
                 int adjN=adj.get(i).get(j).get(0);
                 int wt=adj.get(i).get(j).get(1);
                 int node=i;
                 Edge temp=new Edge(i,adjN,wt);
                 edges.add(temp);
                }
            }
            DisjointSet ds=new DisjointSet(V);
            Collections.sort(edges);
            int mstwt=0;
            for(int i=0;i<edges.size();i++){
                int wt=edges.get(i).weight;
                int u=edges.get(i).src;
                int v=edges.get(i).dest;

                if(ds.findUPar(u)!=ds.findUPar(v)){
                    mstwt+=wt;
                    ds.unionBySize(u,v);
                }
            }
            return mstwt;
        }
    }
}
class Day_25_DP{
    //Max Product Subarray
      //Longest Increasing Subsequence
    class Solution{
        class TUF{
            static int longestIncreasingSubsequence(int arr[],int n){
                int dp[][]=new int[n+1][n+1];
                for(int ind=n-1;ind>=0;ind --){
                    for (int prev_index=ind-1;prev_index>=-1;prev_index --){
                        int notTake=0+dp[ind+1][prev_index +1];
                        int take=0;
                        if(prev_index==-1||arr[ind] > arr[prev_index]){
                            take=1+dp[ind+1][ind+1];
                        }
                        dp[ind][prev_index+1]=Math.max(notTake,take);
                    }
                }
                return dp[0][0];
             }
            static int longestIncreasingSubsequence(int arr[],int n){
                int dp[]=new int[n];
                Arrays.fill(dp,1);
```

```
            for(int i=0;i<=n-1;i++){
                for(int prev_index=0;prev_index<=i-1;prev_index ++){
                    if(arr[prev_index]<arr[i]){
                        dp[i]=Math.max(dp[i],1+dp[prev_index]);
                    }
                }
            }
            int ans=-1;
            for(int i=0;i<=n-1;i++){
                ans=Math.max(ans, dp[i]);
            }
            return ans;
        }
        static int longestIncreasingSubsequence(int arr[],int n){
            int[] dp=new int[n];
            Arrays.fill(dp,1);
            int[] hash=new int[n];
            Arrays.fill(hash,1);

            for(int i=0;i<=n-1;i++){
                hash[i]=i; // initializing with current index
                for(int prev_index=0;prev_index<=i-1;prev_index ++){
                    if(arr[prev_index]<arr[i]&&1+dp[prev_index]>dp[i]){
                        dp[i]=1+dp[prev_index];
                        hash[i]=prev_index;
                    }
                }
            }
            int ans = -1;
            int lastIndex =-1;
            for(int i=0;i<=n-1;i++){
                if(dp[i]>ans){
                    ans=dp[i];
                    lastIndex=i;
                }
            }
            ArrayList<Integer> temp=new ArrayList<>();
            temp.add(arr[lastIndex]);
            while(hash[lastIndex]!=lastIndex){ // till not reach the initialization value
                lastIndex=hash[lastIndex];
                temp.add(arr[lastIndex]);
            }
            for(int i=temp.size()-1; i>=0; i--){
                System.out.print(temp.get(i)+" ");
            }
            return ans;
        }
}}
  //Longest Common Subsequence
```

```
class Solution{
    //****************************Memoization***************************** */
    class TUF{
        static int lcsUtil(String s1,String s2,int ind1,int ind2,int[][] dp){
            if(ind1<0||ind2<0)return 0;
            if(dp[ind1][ind2]!=-1)return dp[ind1][ind2];
            if(s1.charAt(ind1)==s2.charAt(ind2))
                return dp[ind1][ind2]=1+lcsUtil(s1,s2,ind1-1,ind2-1,dp);
            else
                return dp[ind1][ind2]=0+Math.max(lcsUtil(s1,s2,ind1,ind2-1,dp),lcsUtil(s1,
s2,ind1-1,ind2,dp));
        }
        static int lcs(String s1,String s2) {
            int n=s1.length();
            int m=s2.length();
            int dp[][]=new int[n][m];
            for(int rows[]: dp)
            Arrays.fill(rows,-1);
            return lcsUtil(s1,s2,n-1,m-1,dp);
        }
    }
    //****************************Tabulation********************************** */
    class TUF{
        static int lcs(String s1,String s2) {
            int n=s1.length();
            int m=s2.length();
            int dp[][]=new int[n+1][m+1];
            for(int rows[]: dp)
            Arrays.fill(rows,-1);
            for(int i=0;i<=n;i++){
                dp[i][0] = 0;
            }
            for(int i=0;i<=m;i++){
                dp[0][i] = 0;
            }
            for(int ind1=1;ind1<=n;ind1++){
                for(int ind2=1;ind2<=m;ind2++){
                    if(s1.charAt(ind1-1)==s2.charAt(ind2-1))
                        dp[ind1][ind2]=1+dp[ind1-1][ind2-1];
                    else
                        dp[ind1][ind2]=0+Math.max(dp[ind1-1][ind2],dp[ind1][ind2-1]);
                }
            }
            return dp[n][m];
        }
}
}
  //0-1 Knapsack
class Solution{
```

```
        class TUF{
            static int knapsackUtil(int[] wt,int[] val, int ind, int W,int[][] dp){
                if(ind == 0){
                    if(wt[0] <=W) return val[0];
                    else return 0;
                }
                if(dp[ind][W]!=-1)return dp[ind][W];
                int notTaken=0+knapsackUtil(wt,val,ind-1,W,dp);
                int taken=Integer.MIN_VALUE;
                if(wt[ind]<=W)
                    taken=val[ind]+knapsackUtil(wt,val,ind-1,W-wt[ind],dp);

                return dp[ind][W]=Math.max(notTaken,taken);
            }
        }
            class TUF{
                static int knapsack(int[] wt,int[] val, int n, int W){
                    int dp[][]=new int[n][W+1];
                    for(int i=wt[0];i<=W;i++){
                        dp[0][i]=val[0];
                    }
                    for(int ind=1;ind<n;ind++){
                        for(int cap=0;cap<=W;cap++){
                            int notTaken=0+dp[ind-1][cap];
                            int taken=Integer.MIN_VALUE;
                            if(wt[ind]<=cap)
                                taken=val[ind]+dp[ind-1][cap-wt[ind]];
                            dp[ind][cap]=Math.max(notTaken,taken);
                        }
                    }
                    return dp[n-1][W];
                }
            }
        }
          //Edit Distance
        class Solution{
            static int editDistanceUtil(String S1,String S2,int i,int j,int[][] dp){
                if(i<0)return j+1;
                if(j<0)return i+1;
                if(dp[i][j]!=-1) return dp[i][j];
                if(S1.charAt(i)==S2.charAt(j))
                    return dp[i][j]=0+editDistanceUtil(S1,S2,i-1,j-1,dp);
                // Minimum of three choices
                else return dp[i][j]=1+Math.min(editDistanceUtil(S1,S2,i-1,j-1,dp),Math.
min(editDistanceUtil(S1,S2,i-1,j,dp),editDistanceUtil(S1,S2,i,j-1,dp)));
            }
            static int editDistance(String S1, String S2){
                int n=S1.length();
                int m=S2.length();
```

```
            int[][] dp=new int[n+1][m+1];
             for(int i=0;i<=n;i++){
                dp[i][0]=i;
            }
            for(int j=0;j<=m;j++){
                dp[0][j]=j;
            }
            for(int i=1;i<n+1;i++){
                for(int j=1;j<m+1;j++){
                    if(S1.charAt(i-1)==S2.charAt(j-1))
                        dp[i][j]=0+dp[i-1][j-1];
                    else dp[i][j]=1+Math.min(dp[i-1][j-1],Math.min(dp[i-1][j],dp[i][j-1]));
                }
            }
            return dp[n][m];
        }
    }
      //Maximum sum increasing subsequence

      //Matrix Chain Multiplication
    class Solution{
        static int f(int arr[],int i,int j,int[][] dp){
            if(i==j)return 0;
            if(dp[i][j]!=-1)return dp[i][j];
            int mini=Integer.MAX_VALUE;
            for(int k=i;k<=j-1;k++){
            int ans=f(arr,i,k,dp)+f(arr,k+1,j,dp)+arr[i-1]*arr[k]*arr[j];
            mini=Math.min(mini,ans);
            }
            return mini;
        }
    }
}
class Day_26_DP{
    //Minimum sum path in the matrix, (count paths and similar type do, also backtrack
to find the Minimum path)
      class Solution{
        static int minSumPathUtil(int i,int j,int[][] matrix,int[][] dp) {
            if(i==0&&j==0)
              return matrix[0][0];
            if(i<0||j<0)
              return (int)Math.pow(10,9);
            if(dp[i][j]!=-1) return dp[i][j];
            int up=matrix[i][j]+minSumPathUtil(i-1,j,matrix,dp);
            int left=matrix[i][j]+minSumPathUtil(i,j-1,matrix,dp);
            return dp[i][j]=Math.min(up,left);
         }
         static int minSumPath(int n, int m, int[][] matrix){
            int dp[][]=new int[n][m];
```

```
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(i==0&&j==0) dp[i][j]=matrix[i][j];
                else{
                    int up=matrix[i][j];
                    if(i>0) up+=dp[i-1][j];
                    else up+=(int)Math.pow(10,9);
                    int left=matrix[i][j];
                    if(j>0)left+=dp[i][j-1];
                    else left+=(int)Math.pow(10,9);
                    dp[i][j] = Math.min(up,left);
                }
            }
        }
        return dp[n-1][m-1];
    }
}
//Coin change
class Solution{
    class TUF{
        static int minimumElementsUtil(int[] arr,int ind,int T,int[][] dp){
            if(ind==0){
                if(T%arr[0]==0) return T/arr[0];
                else return (int)Math.pow(10,9);
            }
            if(dp[ind][T]!=-1)
                return dp[ind][T];
            int notTaken=0+minimumElementsUtil(arr,ind-1,T,dp);
            int taken=(int)Math.pow(10,9);
            if(arr[ind]<=T)
                taken=1+minimumElementsUtil(arr,ind,T-arr[ind],dp);
            return dp[ind][T]=Math.min(notTaken,taken);
        }
    }
        class TUF{
            static int minimumElements(int[] arr,int T){
                int n=arr.length;
                int dp[][]=new int[n][T+1];
                for(int i=0;i<=T;i++){
                    if(i%arr[0]==0)
                        dp[0][i]=i/arr[0];
                    else dp[0][i]=(int)Math.pow(10,9);
                }
                for(int ind=1;ind<n;ind++){
                    for(int target=0;target<=T;target++){

                        int notTake=0+dp[ind-1][target];
                        int take=(int)Math.pow(10,9);
                        if(arr[ind]<=target)
```

```
                    take=1+ dp[ind][target-arr[ind]];
                dp[ind][target]=Math.min(notTake, take);
            }
        }
        int ans=dp[n-1][T];
        if(ans>=(int)Math.pow(10,9)) return -1;
        return ans;
    }
}
}
  //Subset Sum
class Solution{
    class TUF{
        static boolean subsetSumUtil(int ind,int target,int[] arr,int[][] dp){
            if(target==0)return true;
            if(ind==0)return arr[0]==target;
            if(dp[ind][target]!=-1)return dp[ind][target]==0?false:true;
            boolean notTaken = subsetSumUtil(ind-1,target,arr,dp);
            boolean taken = false;
            if(arr[ind]<=target)
                taken = subsetSumUtil(ind-1,target-arr[ind],arr,dp);
                dp[ind][target]=notTaken||taken?1:0;
            return notTaken||taken;
        }
    }
    class TUF{
        static boolean subsetSumToK(int n,int k,int[] arr){
            boolean dp[][]= new boolean[n][k+1];
            for(int i=0;i<n;i++){
                dp[i][0]=true;
            }
            if(arr[0]<=k)
                dp[0][arr[0]]=true;
            for(int ind=1;ind<n;ind++){
                for(int target=1;target<=k;target++){
                    boolean notTaken=dp[ind-1][target];
                    boolean taken = false;
                        if(arr[ind]<=target)
                            taken=dp[ind-1][target-arr[ind]];
                    dp[ind][target]= notTaken||taken;
                }
            }
            return dp[n-1][k];
        }
    }
}
  //Rod Cutting
class Solution{
    class TUF{
```

```
        static int cutRodUtil(int[] price,int ind,int N,int[][] dp){
            if(ind==0){return N*price[0];}
            if(dp[ind][N]!=-1)return dp[ind][N];
            int notTaken=0+cutRodUtil(price,ind-1,N,dp);
            int taken=Integer.MIN_VALUE;
            int rodLength=ind+1;
            if(rodLength<=N)
                taken=price[ind]+cutRodUtil(price,ind,N-rodLength,dp);
            return dp[ind][N] = Math.max(notTaken,taken);
        }
    }
        class TUF{
            static int cutRod(int[] price,int N) {
                int dp[][]=new int[N][N+1];
                for(int row[]:dp)
                Arrays.fill(row,-1);
                for(int i=0; i<=N; i++){
                    dp[0][i] = i*price[0];
                }
                for(int ind=1;ind<N;ind++){
                    for(int length=0;length<=N;length++){
                        int notTaken=0+dp[ind-1][length];
                        int taken=Integer.MIN_VALUE;
                        int rodLength=ind+1;
                        if(rodLength<=length)
                           taken=price[ind]+dp[ind][length-rodLength];
                        dp[ind][length]=Math.max(notTaken,taken);
                    }
                }
                return dp[N-1][N];
            }
        }
    }
  //Egg Dropping
  //Word Break
  //Palindrome Partitioning (MCM Variation)
class Solution{
    int f(int i,String str){
        if(i==str.length())return 0;
        if(dp[i]!=-1)return dp[i];
        String temp="";
        int minCost=Integer.MAX_VALUE;
        for(int j=i;j<str.length();j++){
            temp=temp+str.charAt(j);
            if(isPalindrome(temp)==true){
                int cost=1+f(j+1,str);
            }
            minCost=Math.min(minCost,cost);
        }
```

```
        return dp[i]=minCost;
    }

    int f(int i,String str){
        int dp[]=new int[n+1];
        for(int i=1;i<n;i++){
            dp[i]=0;
        }
        int n=str.length();
        for(int i=n-1;i>=1;i--){
            int minCost=Integer.MAX_VALUE;
            for(int j=i;j<n;j++){
                if(isPalindrome(i,j,str)==true){
                    int cost=1+dp[j+1];
                    minCost=Math.min(minCost,cost);
                }
            }
            dp[i]=minCost;
        }
        return dp[0]-1;
}}
  //Maximum profit in Job scheduling
class Solution{
    class jobComparator implements Comparator<Job> {
        public int compare(Job j1, Job j2){
            if(j1.profit>j2.profit)return -1;
            if(j1.profit<j2.profit)return 1;
            return 0;
        }
    }
    /*
    class Job {
        int id, profit, deadline;
        Job(int x, int y, int z){
            this.id = x;
            this.deadline = y;
            this.profit = z;
        }
    }
            }
        }
    }*/
    class Solution
    {
        //Function to find the maximum profit and the number of jobs done.
        int[] JobScheduling(Job arr[], int n)
        {
            Arrays.sort(arr,new jobComparator());
```

```
            int res=0,count=0;
            int[] result=new int[n];
            boolean[] slot=new boolean[n];
            Arrays.fill(slot,false);
            for(int i=0;i<n;i++)
            {
                for(int j=Integer.min(n, arr[i].deadline)-1;j>=0;j--)
                {
                    if(slot[j]==false)
                    {
                        result[j]=i;
                        slot[j]=true;
                        break;
                    }
                }
            }
            for(int i=0;i<n;i++)
            {
                if(slot[i]==true){
                    count++;
                    res+=arr[result[i]].profit;
                }
            }
            int[] ans=new int[2];
            ans[0]=count;
            ans[1]=res;
            return ans;
        }
    }
}

}
```