```
public class LoveBabbarDSASheet {
    class Arrays(36){
        // Reverse the array
        // Find the maximum and minimum element in an array
        // Find the “Kth” max and min element of an array
        // Given an array which consists of only 0, 1 and 2. Sort the array without using any sorting algo
        // Move all the negative elements to one side of the array
        // Find the Union and Intersection of the two sorted arrays.
        // Write a program to cyclically rotate an array by one.
        // Find Largest sum contiguous Subarray [V. IMP]
        // Minimize the maximum difference between heights [V.IMP]
        // Minimum no. of Jumps to reach end of an array
        // Find duplicate in an array of N+1 Integers
        // Merge 2 sorted arrays without using Extra space.
        // Kadane’s Algo [V.V.V.V.V IMP]
        // Merge Intervals
        // Next Permutation
        // Count Inversion
        // Best time to buy and Sell stock
        // Find all pairs on integer array whose sum is equal to given number
        // Find common elements In 3 sorted arrays
        // Rearrange the array in alternating positive and negative items with O(1) extra space
        // Find if there is any subarray with sum equal to 0
        // Find factorial of a large number
        // Find maximum product subarray
        // Find longest consecutive subsequence
        // Given an array of size n and a number k, fin all elements that appear more than ” n/k ” times. Link NA
        // Maximum profit by buying and selling a share at most twice
        // Find whether an array is a subset of another array
        // Find the triplet that sum to a given value
        // Trapping Rain water problem
        // Chocolate Distribution problem
        // Smallest Subarray with sum greater than a given value
        // Three way partitioning of an array around a given value
        // Minimum swaps required bring elements less equal K together
        // Minimum no. of operations required to make an array palindrome
        // Median of 2 sorted arrays of equal size
        // Median of 2 sorted arrays of different size
    }
    class Matrix(10){}
    class Strings(43){
        //Reverse a String
        //Check whether a String is Palindrome or not
        //Find Duplicate characters in a string Link NA
        ////Why strings are immutable in Java?
        //Write a Code to check whether one string is a rotation of another
```

```
class Solution{
    public static boolean areRotations(String s1, String s2 )
    {
        int l1=s1.length();
        int l2=s2.length();
        String base=s1+s1;
        if(l1!=l2){return false;}
        if(base.contains(s2)==true){return true;}
        else return false;
    }
}
////Write a Program to check whether a string is a valid shuffle of two strings or not
class Solution{
    static boolean f(String s1,String s2,String s){
        int i=0,j=0,k=0;
        int l1=s1.length(),l2=s2.length();l3=s3.length();
        if(l1+l2!=l3){return false;}
        while(k<l3){
            if(i<l1&&s1.charAt(i)==s.charAt(k)){i++;k++;}
            else if(j<l2&&s2.charAt(j)==s.charAt(k)){j++;k++;}
            else{return false;}
        }
        return true;
    }
}
//Count and Say problem
class Solution{
    static String lookandsay(int n) {
        //your code here
        if(n==1){return "1";}
        if(n==2){return "11";}
        String s="11";
        for(int i=3;i<=n;i++){
            String t="";
            s+="&";
            int c=1;
            for(int j=1;j<s.length();j++){
                if(s.charAt(j)!=s.charAt(j-1)){
                    t=t+Integer.toString(c);
                    t=t+s.charAt(j-1);
                    c=1;
                }
                else{c++;}
            }
            s=t;
        }
        return s;
    }
}
```

```
//Write a program to find the longest Palindrome in a string.[ Longest palindromic Substring]
class Solution{
    static String longestPalin(String S){
        // code here
        int l1=S.length();
        int l,h,start=0,end=1;
        for(int i=1;i<l1;i++){
            //Even Palindrome
            l=i-1;
            h=i;
            while(l>=0&&h<l1&&S.charAt(l)==S.charAt(h)){
                if(h-l+1>end){
                    start=l;
                    end=h-l+1;
                }
                l--;
                h++;
            }
            //Odd Substring
            l=i-1;
            h=i+1;

            while(l>=0&&h<l1&&S.charAt(l)==S.charAt(h)){
                if(h-l+1>end){
                    start=l;
                    end=h-l+1;
                }
                l--;
                h++;
            }
        }
        return S.substring(start,start+end);
    }
}
//Find Longest Recurring Subsequence in String

////Print all Subsequences of a string.
class Solution{
    public void f(int i,int n,String s,String temp){
        if(i==n){
            System.out.println(temp);
            return ;
        }
        f(i+1,n,s,temp);
        f(i+1,n,s,temp+s.charAt(i));
        return ;
    }
}
```

```
        //Print all the permutations of the given string
        class Solution{

        }
        ////Split the Binary string into two substring with equal 0’s and 1’s
        //Word Wrap Problem [VERY IMP].
        //EDIT Distance [Very Imp]
        //Find next greater number with same set of digits. [Very Very IMP]
        //Balanced Parenthesis problem.[Imp]
        //Word break Problem[ Very Imp]
        //Rabin Karp Algorithm
        //KMP Algorithm
        //Convert a Sentence into its equivalent mobile numeric keypad sequence.
        //Minimum number of bracket reversals needed to make an expression balanced.
        //Count All Palindromic Subsequence in a given String.
        //Count of number of given string in 2D character array
        //Search a Word in a 2D Grid of characters.
        //Boyer Moore Algorithm for Pattern Searching.
        //Converting Roman Numerals to Decimal
        //Longest Common Prefix
        //Number of flips to make binary string alternate
        //Find the first repeated word in string.
        //Minimum number of swaps for bracket balancing.
        //Find the longest common subsequence between two strings.
        //Program to generate all possible valid IP addresses from given  string.
        //Write a program to find the smallest window that contains all characters of string
itself.
        //Rearrange characters in a string such that no two adjacent are same
        //Minimum characters to be added at front to make string palindrome
        //Given a sequence of words, print all anagrams together
        //Find the smallest window in a string containing all characters of another string
        //Recursively remove all adjacent duplicates
        //String matching where one string contains wildcard characters
        //Function to find Number of customers who could not get a computer
        //Transform One String to Another using Minimum Number of Given Operation
        //Check if two given strings are isomorphic to each other
        //Recursively print all sentences that can be formed from list of word lists
        //Function to check if two strings are rotations of each other or not.



    }
    class Searching and Sorting(36){}
    class Linked List(36){}
    class Bit Manipulation(10){
        //Count set bits in an integer
        class Solution {
            static int setBits(int N) {
                // code here
        int num=N,count=0;
        while(num>0){
            count+=num&1;
            num=num>>1;
        }
        return count;
    }
}
//Find the two non-repeating elements in an array of repeating elements
class Solution{
    public int getFirstSetBitinXOR(int xor){
        int n=1;
        for(int i=0;i<32;i++){
            int bit=1<<i;
            int check=bit&xor;
            if(check!=0){return i;}
        }
        return 0;
    }
    public int[] singleNumber(int[] nums)
    {
        // Code here
        int xor=0;
        int n=nums.length;
        for(int i=0;i<n;i++){xor=xor^nums[i];}
        int firstDigit=getFirstSetBitinXOR(xor);
        int firstele=0,secondele=0;
        for(int i=0;i<n;i++){
            int bitmask=1<<firstDigit;
            int check=nums[i]&bitmask;
            if(check!=0){firstele=nums[i]^firstele;}
            else {secondele=nums[i]^secondele;}
        }
        int [] ans=new int[]{firstele,secondele};
        Arrays.sort(ans);
        return ans;
    }
}
//Count number of bits to be flipped to convert A to B
class Solution{
    // Function to find number of bits needed to be flipped to convert A to B
    public static int countBitsFlip(int a, int b){
        int x1=a,x2=b;
        int count=0,bit=31;
        while(bit>0){
            int xor=x1^x2;
            int check=xor&1;
            if(check!=0){count++;}
            x1=x1>>1;
```

```
                x2=x2>>1;
                bit--;
            }
            return count;
        }
    }
    //Count total set bits in all numbers from 1 to n

    //Program to find whether a no is power of two
    class Solution{
        public static boolean isPowerofTwo(long n){
            if(n==0){return false;}
            long check=(n)&(n-1);
            return (check==0);
        }
    }
    //Find position of the only set bit
    class Solution {
        static int findPosition(int N) {
            // code here
            int count=0,n=N;
            int ind=0;
            for(int i=0;i<32;i++){
                int y=n>>i;
                int check=y&1;
                if(check!=0){count++;if(count==1){ind=i+1;}}
            }
            if(count==1)return ind;
            else return -1;
        }
    };
    //Copy set bits in a range
    class Solution {
        static int setAllRangeBits(int N , int L , int R) {
            // code here
            int res=N;
            for(int i=L-1;i<R;i++){
                int bitmask=1<<i;
                res=res|bitmask;
            }
            return res;
        }
    };
}
class Greedy(35)
class Backtracking(19){
    //Rat in a maze Problem
    class Solution{
        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 Problem using Backtracking
        //Printing All Solutions in N queen
        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;
    }
}
// Remove Invalid Parentheses
// Sudoku Solver
// M Coloring Problem
// Print all palindromic partitions of a string
```

```
    // Subset Sum Problem
    // The Knight’s tour problem
    // Tug of War
    // Find shortest safe route in a path with landmines
    // Combinational Sum
    // Find Maximum number possible by doing at-most K swaps
    // Print all permutations of a string
    // Find if there is a path of more than k length from a source
    // Longest Possible Route in a Matrix with Hurdles
    // Print all possible paths from top left to bottom right of a mXn matrix
    // Partition of a set into K subsets with equal sum
    // Find the K-th Permutation Sequence of first N natural numbers
}
class Dynamic Programming(60){
    //Coin Change Problem
    class Solution{
        class TUF{
            static long countWaysToMakeChangeUtil(int[] arr,int ind, int T,long[][] dp){
                if(ind == 0){
                    if(T%arr[0]==0)
                    return 1;
                    else
                    return 0;
                }
                if(dp[ind][T]!=-1)return dp[ind][T];
                long notTaken=countWaysToMakeChangeUtil(arr,ind-1,T,dp);
                long taken=0;
                if(arr[ind]<=T)
                    taken=countWaysToMakeChangeUtil(arr,ind,T-arr[ind],dp);

                return dp[ind][T]=notTaken+taken;
            }
        }
            class TUF{
                static long countWaysToMakeChange(int[] arr,int n,int T){
                    long dp[][]=new long[n][T+1];
                    for(int i=0;i<=T;i++){
                        if(i%arr[0]==0)
                            dp[0][i]=1;
                    }
                    for(int ind=1;ind<n;ind++){
                        for(int target=0;target<=T;target++){
                            long notTaken=dp[ind-1][target];
                            long taken=0;
                            if(arr[ind]<=target)
                                taken=dp[ind][target-arr[ind]];
                            dp[ind][target] = notTaken + taken;
                        }
                    }
                return dp[n-1][T];
            }
        }
    }
// Knapsack Problem
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];
            }
    }
}
// Binomial Coefficient Problem
// Permutation Coefficient Problem
// Program for nth Catalan Number
// Matrix Chain Multiplication
// Edit Distance
// Subset Sum Problem
// Friends Pairing Problem
// Gold Mine Problem
// Assembly Line Scheduling Problem
// Painting the Fence problem
// Maximize The Cut Segments
// Longest Common Subsequence
// Longest Repeated Subsequence
// Longest Increasing Subsequence
// Space Optimized Solution of LCS
// LCS (Longest Common Subsequence) of three strings
// Maximum Sum Increasing Subsequence
// Count all subsequences having product less than K
// Longest subsequence such that difference between adjacent is one
// Maximum subsequence sum such that no three are consecutive
// Egg Dropping Problem
// Maximum Length Chain of Pairs
// Maximum size square sub-matrix with all 1s
// Maximum sum of pairs with specific difference
// Min Cost Path Problem
// Maximum difference of zeros and ones in binary string
// Minimum number of jumps to reach end
// Minimum cost to fill given weight in a bag
// Minimum removals from array to make max –min <= K
// Longest Common Substring
// Count number of ways to reach a given score in a game
// Count Balanced Binary Trees of Height h
// LargestSum Contiguous Subarray [V>V>V>V IMP ]
// Smallest sum contiguous subarray
// Unbounded Knapsack (Repetition of items allowed)
// Word Break Problem
// Largest Independent Set Problem
// Partition problem
// Longest Palindromic Subsequence
// Count All Palindromic Subsequence in a given String
// Longest Palindromic Substring
// Longest alternating subsequence
// Weighted Job Scheduling
// Coin game winner where every player has three choices
// Count Derangements (Permutation such that no element appears in its original position) [ IMPORTANT ]
// Maximum profit by buying and selling a share at most twice [ IMP ]
// Optimal Strategy for a Game
// Optimal Binary Search Tree
// Palindrome Partitioning Problem
// Word Wrap Problem
// Mobile Numeric Keypad Problem [ IMP ]
// Boolean Parenthesization Problem
// Largest rectangular sub-matrix whose sum is 0
// Largest area rectangular sub-matrix with equal number of 1’s and 0’s [ IMP ]
// Maximum sum rectangle in a 2D matrix
// Maximum profit by buying and selling a share at most k times
// Find if a string is interleaved of two other strings
```

```
    // Maximum Length of Pair Chain
}
class Stacks and Queues(38){}
class Binary Trees(35){
    //Level order traversal
    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;
        }
    }
    // Reverse Level Order traversal
    class Solution{
        public ArrayList<Integer> reverseLevelOrder(Node root)
        {
            // code here
                    Queue<Node> q=new LinkedList<Node>();
                    Stack<Node> st=new Stack<>();
                    q.offer(root);
                    while(!q.isEmpty()){
                        int size=q.size();
                        for(int i=0;i<size;i++){
                            if(q.peek().right!=null){q.offer(q.peek().right);}
                            if(q.peek().left!=null){q.offer(q.peek().left);}
                            st.add(q.poll());
                        }
                    }
                    ArrayList<Integer> arr=new ArrayList<>();
                    while(!st.isEmpty()){
                        arr.add(st.pop().data);
                    }
                    return arr;

        }
}
// Height of a 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 a 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);
    }
}
// Mirror of a tree
class Solution {
    // Function to convert a binary tree into its mirror tree.
        void mirror(Node node) {
            mirrorutil(node);
        }
        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;
        }
    }
// Inorder Traversal of a tree both using recursion and Iteration
// Preorder Traversal of a tree both using recursion and Iteration
// Postorder Traversal of a tree both using recursion and Iteration
// Left View of a 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);
    }
}
// Right View of 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);
        }
    }
    // Top View of a 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;

    }
}
    // Bottom View of a 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;

            }
        }
            // Zig-Zag traversal of a binary tree
            // Check if a tree is balanced or not
            // Diagonal Traversal of a Binary tree
            // Boundary traversal of a Binary tree
            // Construct Binary Tree from String with Bracket Representation
            // Convert Binary tree into Doubly Linked List
            // Convert Binary tree into Sum tree
            // Construct Binary tree from Inorder and preorder traversal
            // Find minimum swaps required to convert a Binary tree into BST
            // Check if Binary tree is Sum tree or not
            class Solution{
          boolean isSumTree(Node root)
          {
                // Your code here
                return isSumTreeUtil(root)!=-1?true:false;
          }
          int isSumTreeUtil(Node root){
              if(root==null){return 0;}
              int left=isSumTreeUtil(root.left);if(left==-1)return -1;
              int right=isSumTreeUtil(root.right);if(right==-1)return -1;
              if(!isLeaf(root)){if(root.data!=left+right){return -1;}}
              return left+right+root.data;
          }
          boolean isLeaf(Node root){
              if(root.left==null&&root.right==null){return true;}
              else return false;
          }
    }
            // Check if all leaf nodes are at same level or not
            // Check if a Binary Tree contains duplicate subtrees of size 2 or more [ IMP ]
            // Check if 2 trees are mirror or not
            // Sum of Nodes on the Longest path from root to leaf node
            // Check if given graph is tree or not.  [ IMP ]
            // Find Largest subtree sum in a tree
            // Maximum Sum of nodes in Binary tree such that no two are adjacent
            // Print all “K” Sum paths in a Binary tree
            // Find LCA in a Binary tree
            // Find distance between 2 nodes in a Binary tree
            // Kth Ancestor of node in a Binary tree
            // Find all Duplicate subtrees in a Binary tree [ IMP ]
            // Tree Isomorphism Problem

        }
        class BinarySearchTree(2){}
        class Graphs(44){
```

//Create a Graph, print it
// Implement BFS algorithm

// Implement DFS Algo

// Detect Cycle in Directed Graph using BFS/DFS Algo

// Detect Cycle in UnDirected Graph using BFS/DFS Algo

// Search in a Maze

// Minimum Step by Knight

// Flood fill algo

// Clone a graph
// Making wired Connections
// Word Ladder

// Dijkstra algo

// Implement Topological Sort

// Minimum time taken by each job to be completed given by a Directed Acyclic Graph

// Find whether it is possible to finish all tasks or not from given dependencies

// Find the no. of Islands

// Given a sorted Dictionary of an Alien Language, find order of characters

// Implement Kruksal'sAlgorithm
// Implement Prim's Algorithm
// Total no. of Spanning tree in a graph
// Implement Bellman Ford Algorithm

// Implement Floyd warshall Algorithm

// Travelling Salesman Problem
// Graph Colouring Problem
// Snake and Ladders Problem
// Find bridge in a graph
// Count Strongly connected Components(Kosaraju Algo)
// Check whether a graph is Bipartite or Not
// Detect Negative cycle in a graph
// Longest path in a Directed Acyclic Graph
// Journey to the Moon
// Cheapest Flights Within K Stops

```
        // Oliver and the Game
        // Water Jug problem using BFS
        // Find if there is a path of more thank length from a source
        // M-Colouring Problem
        // Minimum edges to reverse to make path from source to destination
        // Paths to travel each nodes using each edge(Seven Bridges)
        // Vertex Cover Problem
        // Chinese Postman or Route Inspection
        // Number of Triangles in a Directed and Undirected Graph
        // Minimise the cashflow among a given set of friends who have borrowed money
from each other
        // Two Clique Problem
    }
    class Heap(18){
        // Implement a Maxheap/MinHeap using arrays and recursion.
        // Sort an Array using heap. (HeapSort)
        // Maximum of all subarrays of size k.
        // “K” largest element in an array
        // Kth smallest and largest element in an unsorted array
        // Merge “K” sorted arrays. [ IMP ]
        // Merge 2 Binary Max Heaps
        // Kth largest sum continuous subarrays
        // Leetcode- reorganize strings
        // Merge “K” Sorted Linked Lists [V.IMP]
        // Smallest range in “K” Lists
        // Median in a stream of Integers
        // Check if a Binary Tree is Heap
        // Connect “n” ropes with minimum cost
        class Solution{
        //Function to return the minimum cost of connecting the ropes.
        long minCost(long arr[], int n)
        {
            // your code here
            PriorityQueue<Long> pq=new PriorityQueue<>();
            for(int i=0;i<n;i++){pq.add(arr[i]);}
            long sum=0;
            while(pq.size()>=2){
                long first=pq.remove();
                long second=pq.remove();
                long newrope=first+second;
                sum+=newrope;
                pq.add(newrope);
            }
            return sum;
        }
    }
        // Convert BST to Min Heap
        // Convert min heap to max heap
        // Rearrange characters in a string such that no two adjacent are same.
        // Minimum sum of two numbers formed from digits of an array
    }
    class Trie(6){}
}
```