1. **--------------------MERGE TWO SORTED LINKED LIST:**

import java.util.Scanner;

class Node{

int data ;

Node next;

Node(int value){

this.data = value;

next = null;

}

}

public class Main {

public static Node merge(Node headA, Node headB) {

Node dummy = new Node(0);

Node current = dummy;

while( headA != null && headB != null){

if(headA.data < headB.data){

current.next = headA;

headA = headA.next;

}

else{

current.next = headB;

headB = headB.next;

}

current = current.next;

}

if(headA != null){

current.next = headA;

}

if(headB != null){

current.next = headB;

}

return dummy.next;

}

public static void printLinkedList(Node head) {

Node current = head;

while (current != null) {

System.out.print(current.data);

if (current.next != null) {

System.out.print("->");

} else {

System.out.print("->NULL");

}

current = current.next;

}

System.out.println();

}

public static Node create(Scanner scanner) {

Node dummy = new Node(0);

Node current = dummy;

int size = scanner.nextInt();

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

int val = scanner.nextInt();

current.next = new Node(val);

current = current.next;

}

return dummy.next;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Node headA = create(sc);

Node headB = create(sc);

Node mergedList = merge(headA, headB);

printLinkedList(mergedList);

sc.close();

}

}

1. **---------------PALINDROME OF LIST:**

**import java.util.Scanner;**

**class Node{**

**int data ;**

**Node next;**

**Node(int value){**

**this.data = value;**

**next = null;**

**}**

**}**

**public class Main {**

**public static boolean palindrome(Node head){**

**Node fast = head;**

**Node slow = head;**

**while(fast != null && fast.next != null){**

**slow = slow.next;**

**fast = fast.next.next;**

**}**

**Node headA = reverse(slow);**

**boolean temp = compare(head,headA);**

**return temp;**

**}**

**public static Node reverse(Node head){**

**Node prev = null;**

**Node curr = head;**

**while(curr != null){**

**Node next = curr.next;**

**curr.next = prev;**

**prev = curr;**

**curr = next;**

**}**

**return prev;**

**}**

**public static boolean compare(Node head,Node headA){**

**Node temp = head;**

**Node temp1 = headA;**

**while(temp != null && temp1 != null){**

**if(temp.data != temp1.data){**

**return false;**

**}**

**else{**

**temp = temp.next;**

**temp1 = temp1.next;**

**}**

**}**

**return true;**

**}**

**public static void printLinkedList(Node head) {**

**Node current = head;**

**while (current != null) {**

**System.out.print(current.data);**

**if (current.next != null) {**

**System.out.print("->");**

**} else {**

**System.out.print("->NULL");**

**}**

**current = current.next;**

**}**

**System.out.println();**

**}**

**public static Node create(Scanner scanner) {**

**Node dummy = new Node(0);**

**Node current = dummy;**

**int size = scanner.nextInt();**

**for (int i = 0; i < size; i++) {**

**int val = scanner.nextInt();**

**current.next = new Node(val);**

**current = current.next;**

**}**

**return dummy.next;**

**}**

**public static void main(String[] args) {**

**Scanner sc = new Scanner(System.in);**

**Node head = create(sc);**

**boolean decision = palindrome(head);**

**if(decision){**

**System.out.println("True");**

**}**

**else{**

**System.out.println("False");**

**}**

**}**

**}**

**3)-------------------------------REORDER LIST:**

**import java.util.Scanner;**

**public class Main {**

**static class ListNode {**

**int val;**

**ListNode next;**

**ListNode(int val) {**

**this.val = val;**

**}**

**}**

**static ListNode head;**

**static void insert(int val) {**

**ListNode newNode = new ListNode(val);**

**newNode.next = head;**

**head = newNode;**

**}**

**static void printList(ListNode head) {**

**while (head != null) {**

**System.out.print(head.val + " ");**

**head = head.next;**

**}**

**System.out.println();**

**}**

**static ListNode reorderList(ListNode head) {**

**if (head == null || head.next == null || head.next.next == null) {**

**return head;**

**}**

**ListNode slow = head;**

**ListNode fast = head;**

**while (fast.next != null && fast.next.next != null) {**

**slow = slow.next;**

**fast = fast.next.next;**

**}**

**ListNode secondHalf = slow.next;**

**slow.next = null;**

**secondHalf = reverseList(secondHalf);**

**ListNode mergedList = mergeLists(head, secondHalf);**

**return mergedList;**

**}**

**static ListNode reverseList(ListNode head) {**

**ListNode prev = null;**

**ListNode current = head;**

**ListNode next = null;**

**while (current != null) {**

**next = current.next;**

**current.next = prev;**

**prev = current;**

**current = next;**

**}**

**return prev;**

**}**

**static ListNode mergeLists(ListNode list1, ListNode list2) {**

**ListNode dummy = new ListNode(0);**

**ListNode current = dummy;**

**while (list1 != null || list2 != null) {**

**if (list1 != null) {**

**current.next = list1;**

**list1 = list1.next;**

**current = current.next;**

**}**

**if (list2 != null) {**

**current.next = list2;**

**list2 = list2.next;**

**current = current.next;**

**}**

**}**

**return dummy.next;**

**}**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**int n = scanner.nextInt();**

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

**int val = scanner.nextInt();**

**insert(val);**

**}**

**head = reorderList(head);**

**printList(head);**

**}**

**}**

**4)-------------------------ROTATE LIST:**

**import java.util.Scanner;**

**public class Main {**

**static class Node {**

**int data;**

**Node next;**

**Node(int data) {**

**this.data = data;**

**this.next = null;**

**}**

**}**

**static Node head;**

**static void insert(int data) {**

**Node newNode = new Node(data);**

**if (head == null) {**

**head = newNode;**

**} else {**

**Node current = head;**

**while (current.next != null) {**

**current = current.next;**

**}**

**current.next = newNode;**

**}**

**}**

**static void printLinkedList(Node head) {**

**while (head != null) {**

**System.out.print(head.data + " ");**

**head = head.next;**

**}**

**System.out.println();**

**}**

**static Node rotateLinkedList(Node head, int rotations) {**

**if (head == null || rotations <= 0) {**

**return head;**

**}**

**int length = getLength(head);**

**rotations = rotations % length;**

**if (rotations == 0) {**

**return head;**

**}**

**Node newHead = head;**

**Node tail = null;**

**for (int i = 0; i < rotations; i++) {**

**tail = newHead;**

**newHead = newHead.next;**

**}**

**tail.next = null;**

**Node temp = newHead;**

**while (temp.next != null) {**

**temp = temp.next;**

**}**

**temp.next = head;**

**return newHead;**

**}**

**static int getLength(Node head) {**

**int length = 0;**

**while (head != null) {**

**length++;**

**head = head.next;**

**}**

**return length;**

**}**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**int data;**

**while ((data = scanner.nextInt()) != -1) {**

**insert(data);**

**}**

**int rotations = scanner.nextInt();**

**System.out.println("Given linked list:");**

**printLinkedList(head);**

**head = rotateLinkedList(head, rotations);**

**System.out.println("Rotated Linked list:");**

**printLinkedList(head);**

**}**

**}**

**5)-------------------------------SEGGREGATE ODD EVEN LIST:**

**import java.util.Scanner;**

**class ListNode {**

**int val;**

**ListNode next;**

**ListNode(int val) {**

**this.val = val;**

**}**

**}**

**public class Main{**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**ListNode head = null;**

**ListNode tail = null;**

**int n = scanner.nextInt();**

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

**int val = scanner.nextInt();**

**ListNode newNode = new ListNode(val);**

**if (head == null) {**

**head = newNode;**

**tail = newNode;**

**} else {**

**tail.next = newNode;**

**tail = newNode;**

**}**

**}**

**Main solution = new Main();**

**ListNode result = solution.oddEvenList(head);**

**while (result != null) {**

**System.out.print(result.val + " ");**

**result = result.next;**

**}**

**}**

**public ListNode oddEvenList(ListNode head) {**

**ListNode oddHead = new ListNode(0);**

**ListNode evenHead = new ListNode(0);**

**ListNode odd = oddHead;**

**ListNode even = evenHead;**

**for (boolean isOdd = true; head != null; head = head.next, isOdd = !isOdd)**

**if (isOdd) {**

**odd.next = head;**

**odd = odd.next;**

**} else {**

**even.next = head;**

**even = even.next;**

**}**

**odd.next = evenHead.next;**

**even.next = null;**

**return oddHead.next;**

**}**

**}**

**6)--------------------------------LONGEST VALID PARENTHESIS:**

**import java.util.\*;**

**class Main{**

**public static void main(String[]args){**

**Scanner sc=new Scanner(System.in);**

**String res=sc.nextLine();**

**int result=longest(res);**

**System.out.println(result);**

**}**

**public static int longest(String res){**

**Stack<Integer> str=new Stack();**

**str.push(-1);**

**int len=0;**

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

**if(res.charAt(i)=='('){**

**str.push(i);**

**}else{**

**str.pop();**

**if(str.isEmpty()){**

**str.push(i);**

**}else{**

**len=Math.max(len,i-str.peek());**

**}**

**}**

**}**

**return len;**

**}**

**}**

**7) --------------------------INFIX TO POSTFIX CONVERSION:**

**import java.util.\*;**

**public class Main{**

**public static void main(String[] args){**

**Scanner sc = new Scanner(System.in);**

**String str = sc.nextLine();**

**longest(str);**

**}**

**public static int priority(char c){**

**if (c == '^')**

**return 3;**

**else if (c == '/' || c == '\*')**

**return 2;**

**else if (c == '+' || c == '-')**

**return 1;**

**else**

**return -1;**

**}**

**public static void longest(String str){**

**StringBuilder result = new StringBuilder();**

**Stack<Character> stk = new Stack();**

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

**char c = str.charAt(i);**

**if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')) {**

**result.append(c);**

**}**

**else if(c == '('){**

**stk.push(c);**

**}**

**else if (c == ')') {**

**while(!stk.isEmpty() && stk.peek() != '(') {**

**result.append(stk.pop());**

**}**

**stk.pop();**

**}**

**else {**

**while (!stk.isEmpty() && (priority(str.charAt(i)) <= priority(stk.peek()) )) {**

**result.append(stk.pop());**

**}**

**stk.push(c);**

**}**

**}**

**while(!stk.isEmpty()){**

**result.append(stk.pop());**

**}**

**System.out.println(result);**

**}**

**}**

**8) ------------------------------IMPLEMENT QUEUE USING STACK:**

**import java.util.Scanner;**

**import java.util.Stack;**

**public class Main {**

**private Stack<Integer> stack1 = new Stack<>();**

**private Stack<Integer> stack2 = new Stack<>();**

**public void enqueue(int element) {**

**stack1.push(element);**

**}**

**public void dequeue() {**

**if (stack2.isEmpty()) {**

**while (!stack1.isEmpty()) {**

**stack2.push(stack1.pop());**

**}**

**}**

**if (!stack2.isEmpty()) {**

**stack2.pop();**

**}**

**}**

**public void displayQueue() {**

**System.out.print("Queue elements are:\n");**

**for (int i = stack2.size() - 1; i >= 0; i--) {**

**System.out.print(stack2.get(i) + " ");**

**}**

**System.out.println();**

**}**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**Main queue = new Main();**

**int n = scanner.nextInt();**

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

**int element = scanner.nextInt();**

**queue.enqueue(element);**

**}**

**int dequeueCount = scanner.nextInt();**

**for (int i = 0; i < dequeueCount; i++) {**

**queue.dequeue();**

**}**

**queue.displayQueue();**

**scanner.close();**

**}**

**}**

**9)------------------------------------IMPLEMENT STACK USING QUEUE:**

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

class StackUsingQueues {

Queue<Integer> queue1 = new LinkedList<>();

Queue<Integer> queue2 = new LinkedList<>();

// Function to push element onto the stack

void push(int x) {

// Add the element to queue1

queue1.offer(x);

}

// Function to pop element from the stack

int pop() {

if (queue1.isEmpty())

return -1; // Stack is empty

// Move all elements from queue1 to queue2 except the last one

while (queue1.size() > 1) {

queue2.offer(queue1.poll());

}

// Pop the last element from queue1 (which is the top element of the stack)

int poppedElement = queue1.poll();

// Swap the queues so that queue2 becomes queue1

Queue<Integer> temp = queue1;

queue1 = queue2;

queue2 = temp;

return poppedElement;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int q = scanner.nextInt();

StackUsingQueues stack = new StackUsingQueues();

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

int queryType = scanner.nextInt();

if (queryType == 1) {

int x = scanner.nextInt();

stack.push(x);

} else if (queryType == 2) {

int poppedElement = stack.pop();

System.out.println(poppedElement);

}

}

}

}

**10)------------------------------DIAMETER OF THE TREE:**

**import java.util.\*;**

**class Node{**

**int data;**

**Node left, right;**

**Node(int val){**

**data = val;**

**left = right = null;**

**}**

**}**

**class BinaryTree{**

**Node build(Node node, int val){**

**if(node == null){**

**return new Node(val);**

**}**

**if(node.data < val){**

**node.right = build(node.right,val);**

**}**

**if(node.data > val){**

**node.left = build(node.left,val);**

**}**

**return node;**

**}**

**public int height(Node root){**

**if(root == null){**

**return 0;**

**}**

**int leftH = height(root.left);**

**int rightH = height(root.right);**

**return (Math.max(leftH, rightH)) + 1;**

**}**

**public int diameter(Node node){**

**if(node == null){**

**return 0;**

**}**

**int inclusive = height(node.left) + height(node.right) + 1;**

**int exclusive = Math.max(diameter(node.left),diameter(node.right));**

**return Math.max(inclusive,exclusive);**

**}**

**}**

**class Main{**

**public static void main(String args[])**

**{**

**Scanner scn = new Scanner(System.in);**

**int val;**

**Node root = null;**

**BinaryTree tree = new BinaryTree();**

**while((val = scn.nextInt()) != -1){**

**root = tree.build(root, val);**

**}**

**System.out.print("Diameter of the given binary tree is " + tree.diameter(root));**

**}**

**}**

**11)-----------------------BINARY TREE ZIGZAG LEVEL :**

**import java.util.\*;**

**class TreeNode {**

**int val;**

**TreeNode left;**

**TreeNode right;**

**TreeNode(int x) {**

**val = x;**

**}**

**}**

**public class Main {**

**public List<Integer> zigzagLevelOrder(TreeNode root) {**

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

**if (root == null) {**

**return result;**

**}**

**Deque<TreeNode> deque = new LinkedList<>();**

**deque.offer(root);**

**boolean leftToRight = true;**

**while (!deque.isEmpty()) {**

**int levelSize = deque.size();**

**LinkedList<Integer> currentLevel = new LinkedList<>();**

**for (int i = 0; i < levelSize; i++) {**

**TreeNode current;**

**if (leftToRight) {**

**current = deque.pollFirst();**

**if (current.left != null) {**

**deque.offerLast(current.left);**

**}**

**if (current.right != null) {**

**deque.offerLast(current.right);**

**}**

**} else {**

**current = deque.pollLast();**

**if (current.right != null) {**

**deque.offerFirst(current.right);**

**}**

**if (current.left != null) {**

**deque.offerFirst(current.left);**

**}**

**}**

**currentLevel.add(current.val);**

**}**

**result.addAll(currentLevel);**

**leftToRight = !leftToRight;**

**}**

**return result;**

**}**

**public static void main(String[] args) {**

**Main solution = new Main();**

**Scanner scanner = new Scanner(System.in);**

**String[] input = scanner.nextLine().split(" ");**

**TreeNode root = null;**

**if (!input[0].equals("N")) {**

**root = new TreeNode(Integer.parseInt(input[0]));**

**Queue<TreeNode> queue = new LinkedList<>();**

**queue.offer(root);**

**int index = 1;**

**while (!queue.isEmpty() && index < input.length) {**

**TreeNode current = queue.poll();**

**if (!input[index].equals("N")) {**

**current.left = new TreeNode(Integer.parseInt(input[index]));**

**queue.offer(current.left);**

**}**

**index++;**

**if (index < input.length && !input[index].equals("N")) {**

**current.right = new TreeNode(Integer.parseInt(input[index]));**

**queue.offer(current.right);**

**}**

**index++;**

**}**

**}**

**List<Integer> result = solution.zigzagLevelOrder(root);**

**for (int val : result) {**

**System.out.print(val + " ");**

**}**

**}**

**}**

**12)---------------------------------SUMROOTS OF LEAF NODES:**

**import java.util.\*;**

**class Node{**

**int data;**

**Node left;**

**Node right;**

**Node(int data){**

**this.data = data;**

**left=null;**

**right=null;**

**}**

**}**

**class Main{**

**static int ans=0;**

**static Node buildTree(String str){**

**if(str.length()==0 || str.charAt(0)=='N'){**

**return null;**

**}**

**String ip[]=str.split(" ");**

**Node root = new Node(Integer.parseInt(ip[0]));**

**Queue<Node> queue = new LinkedList<>();**

**queue.add(root);**

**int i=1;**

**while(queue.size()>0 && i<ip.length){**

**Node currNode = queue.peek();**

**queue.remove();**

**String currVal = ip[i];**

**if(!currVal.equals("N")){**

**currNode.left = new Node(Integer.parseInt(currVal));**

**queue.add(currNode.left);**

**}**

**i++;**

**if(i >= ip.length)**

**break;**

**currVal = ip[i];**

**if(!currVal.equals("N")){**

**currNode.right = new Node(Integer.parseInt(currVal));**

**queue.add(currNode.right);**

**}**

**i++;**

**}**

**return root;**

**}**

**public static void preorderSum(Node root, String sr)**

**{**

**if(root==null)**

**return;**

**sr+=root.data;**

**if(root.left==null && root.right==null)**

**{**

**ans+=Integer.parseInt(sr);**

**return;**

**}**

**preorderSum(root.left,sr);**

**preorderSum(root.right,sr);**

**}**

**public static void main (String[] args){**

**Scanner sc=new Scanner(System.in);**

**String s = sc.nextLine();**

**Node root = buildTree(s);**

**preorderSum(root,"");**

**System.out.print(ans);**

**}**

**}**

**KTH SMALLEST ELEMENT IN BST:**

**import java.util.\*;**

**class TreeNode {**

**int val;**

**TreeNode left, right;**

**TreeNode(int val) {**

**this.val = val;**

**left = right = null;**

**}**

**}**

**public class Main {**

**TreeNode root;**

**int count;**

**public int kthSmallest(TreeNode root, int k) {**

**count = 0;**

**return inorderTraversal(root, k);**

**}**

**public int inorderTraversal(TreeNode root, int k) {**

**if (root == null) {**

**return -1;**

**}**

**// Traverse left subtree**

**int leftResult = inorderTraversal(root.left, k);**

**if (leftResult != -1) {**

**return leftResult;**

**}**

**// Process current node**

**count++;**

**if (count == k) {**

**return root.val;**

**}**

**// Traverse right subtree**

**return inorderTraversal(root.right, k);**

**}**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**Main main = new Main();**

**// Build the binary search tree**

**while (true) {**

**int val = scanner.nextInt();**

**if (val == -1) {**

**break;**

**}**

**main.root = main.insert(main.root, val);**

**}**

**// Input k**

**System.out.println("Enter the kth value:");**

**int k = scanner.nextInt();**

**// Find the kth smallest element**

**int kthSmallest = main.kthSmallest(main.root, k);**

**// Output the result**

**System.out.println("Smallest kth value " + kthSmallest);**

**}**

**public TreeNode insert(TreeNode node, int data) {**

**if (node == null) {**

**return new TreeNode(data);**

**}**

**if (data < node.val) {**

**node.left = insert(node.left, data);**

**} else if (data > node.val) {**

**node.right = insert(node.right, data);**

**}**

**return node;**

**}**

**}**

**13)---------------------------------MINIMUM FUEL COST TO REPORT TO THE CAPITAL CITY:**

**import java.util.\*;**

**public class Main {**

**static class TreeNode {**

**int val;**

**List<TreeNode> children;**

**TreeNode(int val) {**

**this.val = val;**

**this.children = new ArrayList<>();**

**}**

**}**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**int n = scanner.nextInt(); // Number of cities**

**List<int[]> roads = new ArrayList<>();**

**for (int i = 0; i < n - 1; i++) {**

**int a = scanner.nextInt();**

**int b = scanner.nextInt();**

**roads.add(new int[]{a, b});**

**}**

**int[] seats = new int[n];**

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

**seats[i] = scanner.nextInt();**

**}**

**int minFuel = minFuelToReachCapital(n, roads, seats);**

**System.out.println(minFuel);**

**}**

**public static int minFuelToReachCapital(int n, List<int[]> roads, int[] seats) {**

**Map<Integer, TreeNode> graph = new HashMap<>();**

**// Construct the tree graph**

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

**graph.put(i, new TreeNode(seats[i]));**

**}**

**for (int[] road : roads) {**

**int u = road[0];**

**int v = road[1];**

**graph.get(u).children.add(graph.get(v));**

**graph.get(v).children.add(graph.get(u));**

**}**

**return dfs(graph.get(0), null);**

**}**

**public static int dfs(TreeNode node, TreeNode parent) {**

**int fuelRequired = 0;**

**for (TreeNode child : node.children) {**

**if (child != parent) {**

**// Calculate fuel required for subtree rooted at 'child'**

**fuelRequired += Math.max(0, (child.val - 1) / 2);**

**fuelRequired += dfs(child, node);**

**}**

**}**

**return fuelRequired;**

**}**

**}**

**DFS:**

**COURSE SCHEDULE:**

**14)--------------------------------------LETTER COMBINATION OF PHONE NUMBER:**

**import java.util.ArrayList;**

**import java.util.List;**

**import java.util.Scanner;**

**public class Main {**

**private static final String[] LETTERS = {**

**"", // 0**

**"", // 1**

**"abc", // 2**

**"def", // 3**

**"ghi", // 4**

**"jkl", // 5**

**"mno", // 6**

**"pqrs", // 7**

**"tuv", // 8**

**"wxyz" // 9**

**};**

**public static List<String> letterCombinations(String digits) {**

**List<String> result = new ArrayList<>();**

**if (digits == null || digits.length() == 0)**

**return result;**

**backtrack(result, "", digits, 0);**

**return result;**

**}**

**private static void backtrack(List<String> result, String combination, String digits, int index) {**

**if (index == digits.length()) {**

**result.add(combination);**

**return;**

**}**

**String letters = LETTERS[digits.charAt(index) - '0'];**

**for (char letter : letters.toCharArray()) {**

**backtrack(result, combination + letter, digits, index + 1);**

**}**

**}**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**String digits = scanner.nextLine();**

**List<String> combinations = letterCombinations(digits);**

**for (String combination : combinations) {**

**System.out.print(combination + " ");**

**}**

**}**

**}**

**26)----------------------------------WILDCARD PATTERN:**

**import java.util.Scanner;**

**public class Main {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**String text = scanner.nextLine();**

**String pattern = scanner.nextLine();**

**boolean result = isMatch(text, pattern);**

**System.out.println(result); // Print the result**

**scanner.close();**

**}**

**public static boolean isMatch(String text, String pattern) {**

**boolean[][] dp = new boolean[text.length() + 1][pattern.length() + 1];**

**dp[0][0] = true;**

**for (int j = 1; j <= pattern.length(); j++) {**

**if (pattern.charAt(j - 1) == '\*') {**

**dp[0][j] = dp[0][j - 1];**

**}**

**}**

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

**for (int j = 1; j <= pattern.length(); j++) {**

**if (pattern.charAt(j - 1) == '\*') {**

**dp[i][j] = dp[i - 1][j] || dp[i][j - 1];**

**} else if (pattern.charAt(j - 1) == '?' || text.charAt(i - 1) == pattern.charAt(j - 1)) {**

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

**}**

**}**

**}**

**return dp[text.length()][pattern.length()];**

**}**

**}**

**15)------------------------------HOUSE ROBBER:**

**import java.util.\*;**

**public class Main{**

**public static void main(String[] args){**

**Scanner input=new Scanner(System.in);**

**String str =input.nextLine();**

**String[] string=str.split(",");**

**int[] nums=new int[string.length];**

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

**nums[i]=Integer.valueOf(string[i]);**

**}**

**int rob=0;**

**int norob=0;**

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

**int newrob=norob+nums[i];**

**int newnorob=Math.max(norob,rob);**

**rob=newrob;**

**norob=newnorob;**

**}**

**int ans=Math.max(rob,norob);**

**System.out.println(ans);**

**}**

**}**

**--------------------------------------EDIT DISTANCE:**

**16)---------------------------------------------NO.OF LONGEST INCREASING SUBSEQUENCE:**

**import java.util.\*;**

**public class Main**

**{**

**public static void main(String[] args) {**

**Scanner scanner=new Scanner(System.in);**

**String input=scanner.nextLine();**

**String[] numbersString=input.split(",");**

**int [] arr=new int[numbersString.length];**

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

**arr[i]=Integer.parseInt(numbersString[i]);**

**}**

**int counter=0;**

**int max=0;**

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

**if(max<arr[i]) {**

**max=arr[i];**

**counter++;**

**}**

**}**

**System.out.println(counter);**

**}**

**}**

**17)------------------------------------GENERATE PARENTHESIS:**

**import java.util.\*;**

**import java.lang.\*;**

**import java.io.\*;**

**/\* Name of the class has to be "Main" only if the class is public. \*/**

**class Main**

**{**

**static List<String> generateParenthesis(int n){**

**List<String> l= new ArrayList();**

**generateParenthesisHelper(n,0,0,"",l);**

**return l;**

**}**

**static void generateParenthesisHelper(int n, int openCount, int closeCount, String current, List<String> l){**

**if(current.length()==2\*n){**

**l.add(current);**

**return;**

**}**

**if (openCount <n){**

**generateParenthesisHelper(n,openCount+1,closeCount,current+"(",l);**

**}**

**if(closeCount<openCount){**

**generateParenthesisHelper(n,openCount,closeCount+1,current+")",l);**

**}**

**}**

**public static void main (String[] args)**

**{**

**Scanner sc = new Scanner(System.in);**

**int n=sc.nextInt();**

**List<String> l =generateParenthesis(n);**

**for(String ele: l){**

**System.out.println(ele);**

**}**

**}**

**}**

**18)----------------------------------LONGEST SUBSTRING WITHOUT REPEATING CHARACTERS:**

import java.util.HashMap;

import java.util.Scanner;

class LongestSubstringWithoutRepeatingChars {

  public static int lengthOfLongestSubstring(String s) {

    int n = s.length();

    if (n < 2) {

      return n > 0 ? 1 : 0;

    }

    int maxLength = 0;

    int left = 0, right = 0;

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

    while (right < n) {

      char currentChar = s.charAt(right);

      if (charIndexMap.containsKey(currentChar)) {

        left = Math.max(left, charIndexMap.get(currentChar) + 1);

      }

      charIndexMap.put(currentChar, right);

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

      right++;

    }

    return maxLength;

  }

  public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    String str = scanner.nextLine();

    int longestLength = lengthOfLongestSubstring(str);

    if (longestLength == 1 && str.length() == 1) {

      System.out.println("The input string has only one character. Longest substring is the character itself.");

    } else {

      System.out.println(longestLength);

    }

    scanner.close();

  }

}

**19)-------------------------------SHORTEST PALINDROME:**

import java.util.Scanner;

class ShortestPalindrome {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        String str = sc.nextLine();

        System.out.println(shortest(str));

    }

    public static String shortest(String str) {

        int start = 0;

        int end = str.length() - 1;

        while (start < end) {

            if (str.charAt(start) == str.charAt(end)) {

                start++;

                end--;

            } else {

                return buildPalindrome(str, start);

            }

        }

        return str;

    }

    private static String buildPalindrome(String str, int start) {

        StringBuilder prefix = new StringBuilder(str.substring(start + 1));

        prefix.reverse();

        return prefix.toString() + str;

    }

}

**20)---------------------------------TRAPPING RAIN WATER:**

import java.util.Scanner;

class TrappingRainWater {

    public static int trapWater(int[] heights) {

        int n = heights.length;

        if (n == 0) return 0;

        int left = 0, right = n - 1;

        int leftMax = 0, rightMax = 0;

        int water = 0;

        while (left < right) {

            if (heights[left] < heights[right]) {

                if (heights[left] >= leftMax) {

                    leftMax = heights[left];

                } else {

                    water += leftMax - heights[left];

                }

                left++;

            } else {

                if (heights[right] >= rightMax) {

                    rightMax = heights[right];

                } else {

                    water += rightMax - heights[right];

                }

                right--;

            }

        }

        return water;

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        String[] input = scanner.nextLine().split(" ");

        int[] heights = new int[input.length];

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

            heights[i] = Integer.parseInt(input[i]);

        }

        int trappedWater = trapWater(heights);

        System.out.println(trappedWater);

    }

}

**21)--------------------------------------LONGEST PALINDROMIC SUBSTRING:**

import java.util.Scanner;

class LongestPalindromicSubstring {

  public static String longestPalindrome(String str) {

    int n = str.length();

    int maxLength = 0;

    String longestPal = "";

    for (int i = 0; i < n; i++) {

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

        String subStr = str.substring(i, j);

        if (isPalindrome(subStr)) {

          int subStrLen = subStr.length();

          if (subStrLen > maxLength) {

            maxLength = subStrLen;

            longestPal = subStr;

          }

        }

      }

    }

    return longestPal;

  }

  public static boolean isPalindrome(String str) {

    int left = 0, right = str.length() - 1;

    while (left < right) {

      if (str.charAt(left) != str.charAt(right)) {

        return false;

      }

      left++;

      right--;

    }

    return true;

  }

  public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    String str = scanner.nextLine();

    String longestPal = longestPalindrome(str);

    System.out.println(longestPal);

    scanner.close();

  }

}

**22)-----------------------------LONGEST HAPPY PREFIX:**

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class Main

{

    public static String longestAprefix(String s){

        int[] pref = new int[s.length()];

        Arrays.fill(pref,-1);

        int i = 1;

        int j = 0;

        int max = -1;

        while(i<s.length())

        {

            if(s.charAt(i) == s.charAt(j) ){

                pref[i] = j;

                i++;

                j++;

            }

            else if(j>0)

            {

                j=pref[j-1]+1;

            }

            else{

                i++;

            }

        }

        max = pref[s.length()-1];

        return s.substring(0,max+1);

    }

    public static void main(String[]args)

    {

        Scanner sc = new Scanner(System.in);

        String s = sc.nextLine();

        String result = longestAprefix(s);

        System.out.println(result);

    }

}

**23)-------------------------------------PERMUTATIONS:**

import java.util.\*;

class Permutation {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String input = sc.nextLine();

String[] strArr = input.split("\\s+");

int[] arr = new int[strArr.length];

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

arr[i] = Integer.parseInt(strArr[i]);

}

int n = arr.length;

Permutation permutation = new Permutation();

permutation.permute(arr, 0, n - 1);

sc.close();

}

private void permute(int[] arr, int l, int r) {

if (l == r) {

printArray(arr);

} else {

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

swap(arr, l, i);

permute(arr, l + 1, r);

swap(arr, l, i);

}

}

}

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

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

private void printArray(int[] arr) {

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

System.out.print(arr[i]);

if(i != arr.length - 1) {

System.out.print(" ");

}

}

System.out.println();

}

}

**24)-------------------------SPIRAL MATRIX:**

import java.util.\*;

import java.lang.\*;

import java.io.\*;

import java.util.Scanner;

class SpiralMatrix {

  public static void printSpiral(int[][] matrix) {

    int rows = matrix.length;

    int cols = matrix[0].length;

    int top = 0, bottom = rows - 1, left = 0, right = cols - 1;

    int direction = 0; // 0 - right, 1 - down, 2 - left, 3 - up

    while (top <= bottom && left <= right) {

      switch (direction) {

        case 0: // Move right

          for (int col = left; col <= right; col++) {

            System.out.print(matrix[top][col] + " ");

          }

          top++;

          break;

        case 1: // Move down

          for (int row = top; row <= bottom; row++) {

            System.out.print(matrix[row][right] + " ");

          }

          right--;

          break;

        case 2: // Move left

          for (int col = right; col >= left; col--) {

            System.out.print(matrix[bottom][col] + " ");

          }

          bottom--;

          break;

        case 3: // Move up

          for (int row = bottom; row >= top; row--) {

            System.out.print(matrix[row][left] + " ");

          }

          left++;

          break;

      }

      direction = (direction + 1) % 4; // Change direction for next iteration

    }

  }

  public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    int rows = scanner.nextInt();

    int cols = scanner.nextInt();

    int[][] matrix = new int[rows][cols];

    for (int i = 0; i < rows; i++) {

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

        matrix[i][j] = scanner.nextInt();

      }

    }

    printSpiral(matrix);

    scanner.close();

  }

}

**25)--------------------------------MINIMUM DELETIONS TO MAKE ARRAYS DIVISIBLE:**

import java.util.Scanner;

class MinimumDeletions {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        String[] numsInput = scanner.nextLine().split(" ");

        int[] nums = new int[numsInput.length];

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

            nums[i] = Integer.parseInt(numsInput[i]);

        }

        String[] numsDivideInput = scanner.nextLine().split(" ");

        int[] numsDivide = new int[numsDivideInput.length];

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

            numsDivide[i] = Integer.parseInt(numsDivideInput[i]);

        }

        System.out.println(minDeletions(nums, numsDivide));

    }

    public static int minDeletions(int[] nums, int[] numsDivide) {

        int minNum = Integer.MAX\_VALUE;

        for (int num : nums) {

            minNum = Math.min(minNum, num);

        }

        for (int num : numsDivide) {

            if (num % minNum != 0) {

                return -1;

            }

        }

        int deletions = 0;

        for (int num : nums) {

            if (num % minNum != 0) {

                deletions++;

            }

        }

        return deletions;

    }

}