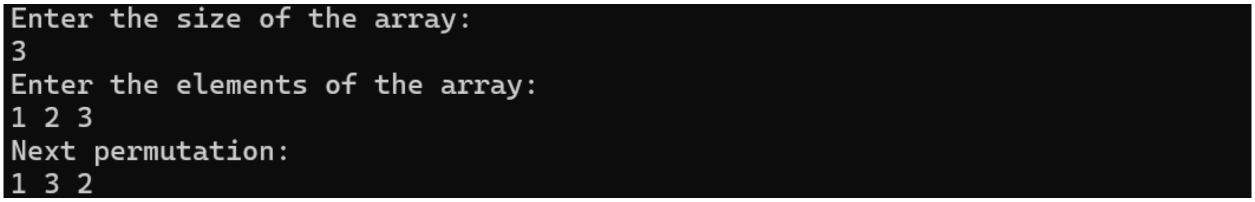
PRACTICE SET – 6 **19-11-2024**

DSA

1.Next Permutation

```
import java.util.*;

public class Problem1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of the array:");
        int n = sc.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) nums[i] = sc.nextInt();
        nextPermutation(nums);
        System.out.println("Next permutation:");
        for (int num : nums) System.out.print(num + " ");
    }

    public static void nextPermutation(int[] nums) {
        int i = nums.length - 2;
        while (i >= 0 && nums[i] >= nums[i + 1]) i--;
        if (i >= 0) {
            int j = nums.length - 1;
            while (nums[j] <= nums[i]) j--;
            swap(nums, i, j);
        }
        reverse(nums, i + 1);
    }

    private static void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }

    private static void reverse(int[] nums, int start) {
        int end = nums.length - 1;
        while (start < end) {
            swap(nums, start, end);
            start++;
            end--;
        }
    }
}
```

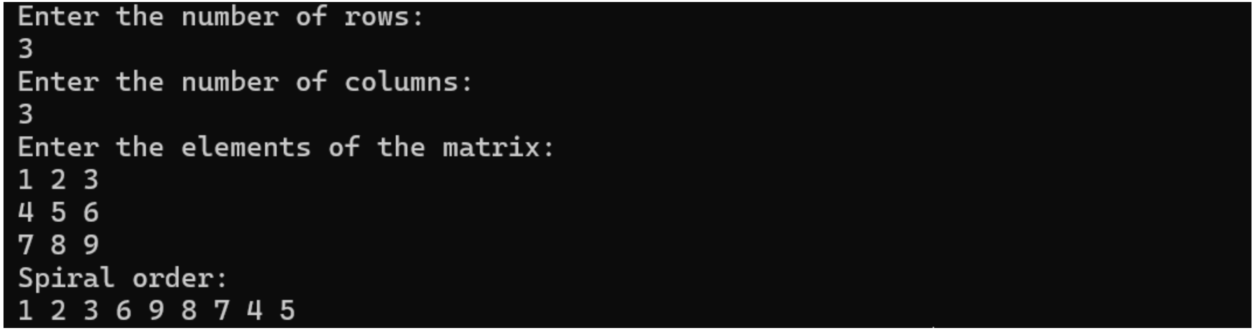


```
Enter the size of the array:
3
Enter the elements of the array:
1 2 3
Next permutation:
1 3 2
```

2.Spiral Matrix

```
import java.util.*;

public class Problem2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of rows:");
        int rows = sc.nextInt();
        System.out.println("Enter the number of columns:");
        int cols = sc.nextInt();
        int[][] matrix = new int[rows][cols];
        System.out.println("Enter the elements of the matrix:");
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                matrix[i][j] = sc.nextInt();
        List<Integer> result = spiralOrder(matrix);
        System.out.println("Spiral order:");
        for (int num : result) System.out.print(num + " ");
    }

    public static List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> result = new ArrayList<>();
        if (matrix.length == 0) return result;
        int top = 0, bottom = matrix.length - 1, left = 0, right = matrix[0].length - 1;
        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; i++) result.add(matrix[top][i]);
            top++;
            for (int i = top; i <= bottom; i++) result.add(matrix[i][right]);
            right--;
            if (top <= bottom) for (int i = right; i >= left; i--) result.add(matrix[bottom][i]);
            bottom--;
            if (left <= right) for (int i = bottom; i >= top; i--) result.add(matrix[i][left]);
            left++;
        }
        return result;
    }
```

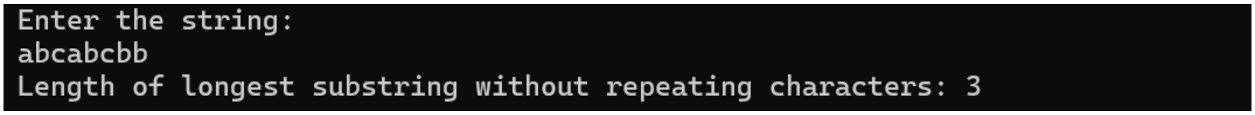
```
}
```



```
Enter the number of rows:
3
Enter the number of columns:
3
Enter the elements of the matrix:
1 2 3
4 5 6
7 8 9
Spiral order:
1 2 3 6 9 8 7 4 5
```

3. Longest Substring Without Repeating Characters

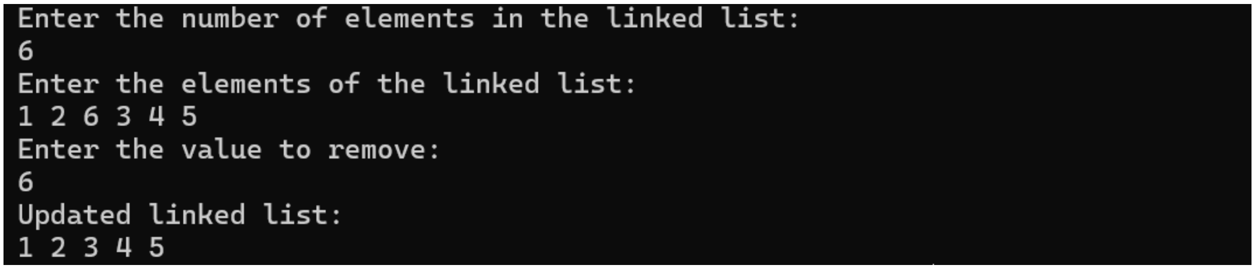
```
import java.util.*;


public class Problem3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the string:");
        String s = sc.nextLine();
        System.out.println("Length of longest substring without repeating characters: " + lengthOfLongestSubstring(s));
    }


    public static int lengthOfLongestSubstring(String s) {
        Set<Character> set = new HashSet<>();
        int maxLength = 0, left = 0;
        for (int right = 0; right < s.length(); right++) {
            while (set.contains(s.charAt(right))) {
                set.remove(s.charAt(left));
                left++;
            }
            set.add(s.charAt(right));
            maxLength = Math.max(maxLength, right - left + 1);
        }
        return maxLength;
    }
}
```



```
Enter the string:
abcabcbb
Length of longest substring without repeating characters: 3
```

4. Remove Linked List Elements

```
import java.util.*;

public class Problem4 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements in the linked list:");
        int n = sc.nextInt();
        System.out.println("Enter the elements of the linked list:");
        ListNode head = new ListNode(sc.nextInt()), curr = head;
        for (int i = 1; i < n; i++) {
            curr.next = new ListNode(sc.nextInt());
            curr = curr.next;
        }
        System.out.println("Enter the value to remove:");
        int val = sc.nextInt();
        head = removeElements(head, val);
        System.out.println("Updated linked list:");
        while (head != null) {
            System.out.print(head.val + " ");
            head = head.next;
        }
    }

    public static ListNode removeElements(ListNode head, int val) {
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode curr = dummy;
        while (curr.next != null) {
            if (curr.next.val == val) curr.next = curr.next.next;
            else curr = curr.next;
        }
        return dummy.next;
    }

    static class ListNode {
        int val;
        ListNode next;
        ListNode(int x) { val = x; }
    }
}
```

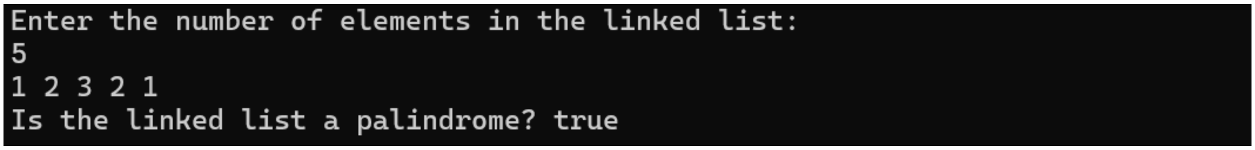


```
Enter the number of elements in the linked list:
6
Enter the elements of the linked list:
1 2 6 3 4 5
Enter the value to remove:
6
Updated linked list:
1 2 3 4 5
```

5. Palindrome Linked List

```
import java.util.*;
```

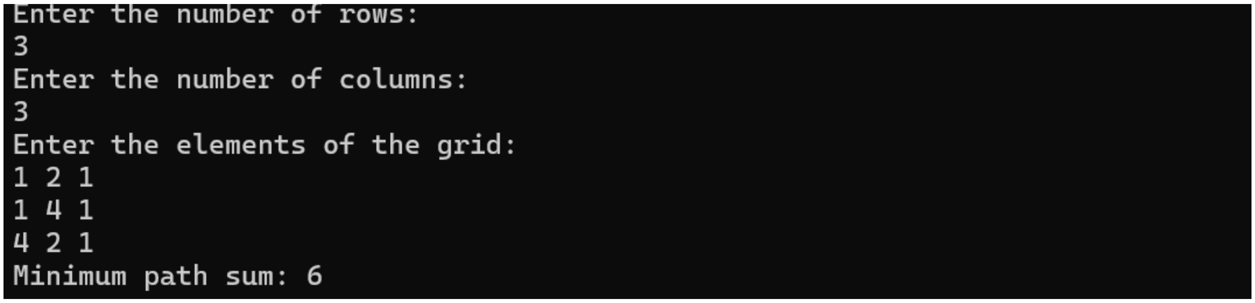
```
public class Problem5 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements in the linked list:");
        int n = sc.nextInt();
        ListNode head = new ListNode(sc.nextInt()), curr = head;
        for (int i = 1; i < n; i++) {
            curr.next = new ListNode(sc.nextInt());
            curr = curr.next;
        }
        System.out.println("Is the linked list a palindrome? " + isPalindrome(head));
    }

    public static boolean isPalindrome(ListNode head) {
        ListNode slow = head, fast = head, prev = null;
        while (fast != null && fast.next != null) {
            fast = fast.next.next;
            ListNode temp = slow;
            slow = slow.next;
            temp.next = prev;
            prev = temp;
        }
        if (fast != null) slow = slow.next;
        while (prev != null && prev.val == slow.val) {
            prev = prev.next;
            slow = slow.next;
        }
        return prev == null;
    }
    static class ListNode {
        int val;
        ListNode next;
        ListNode(int x) { val = x; }
    }
}
```



```
Enter the number of elements in the linked list:
5
1 2 3 2 1
Is the linked list a palindrome? true
```

6. Minimum Path Sum

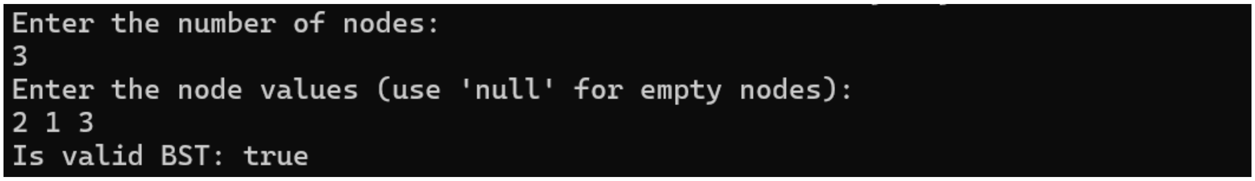
```
import java.util.*;

public class Problem6 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of rows:");
        int rows = sc.nextInt();
        System.out.println("Enter the number of columns:");
        int cols = sc.nextInt();
        int[][] grid = new int[rows][cols];
        System.out.println("Enter the elements of the grid:");
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                grid[i][j] = sc.nextInt();
        System.out.println("Minimum path sum: " + minPathSum(grid));
    }
    public static int minPathSum(int[][] grid) {
        for (int i = 1; i < grid.length; i++) grid[i][0] += grid[i - 1][0];
        for (int j = 1; j < grid[0].length; j++) grid[0][j] += grid[0][j - 1];
        for (int i = 1; i < grid.length; i++)
            for (int j = 1; j < grid[0].length; j++)
                grid[i][j] += Math.min(grid[i - 1][j], grid[i][j - 1]);
        return grid[grid.length - 1][grid[0].length - 1];
    }
}
```



```
Enter the number of rows:
3
Enter the number of columns:
3
Enter the elements of the grid:
1 2 1
1 4 1
4 2 1
Minimum path sum: 6
```

7. Validate Binary Search Tree

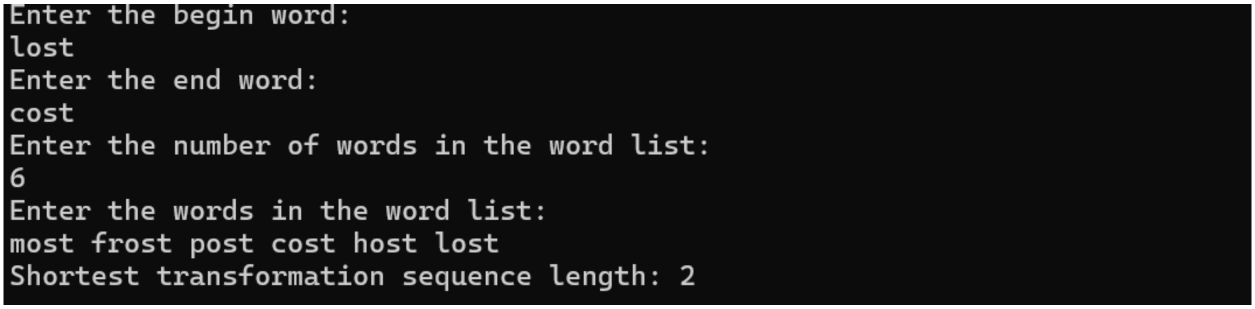
```
import java.util.*;

public class Problem7 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of nodes:");
        int n = sc.nextInt();
        Integer[] nodes = new Integer[n];
        System.out.println("Enter the node values (use 'null' for empty nodes):");
        for (int i = 0; i < n; i++) {
            String input = sc.next();
            nodes[i] = input.equals("null") ? null : Integer.parseInt(input);
        }
        TreeNode root = buildTree(nodes, 0);
        System.out.println("Is valid BST: " + isValidBST(root));
    }

    public static TreeNode buildTree(Integer[] nodes, int index) {
        if (index >= nodes.length || nodes[index] == null) return null;
        TreeNode node = new TreeNode(nodes[index]);
        node.left = buildTree(nodes, 2 * index + 1);
        node.right = buildTree(nodes, 2 * index + 2);
        return node;
    }

    public static boolean isValidBST(TreeNode root) {
        return validate(root, null, null);
    }

    public static boolean validate(TreeNode node, Integer low, Integer high) {
        if (node == null) return true;
        if ((low != null && node.val <= low) || (high != null && node.val >= high)) return false;
        return validate(node.left, low, node.val) && validate(node.right, node.val, high);
    }

    static class TreeNode {
        int val;
        TreeNode left, right;
        TreeNode(int x) { val = x; }
    }
}
```



```
Enter the number of nodes:
3
Enter the node values (use 'null' for empty nodes):
2 1 3
Is valid BST: true
```

8. Word Ladder

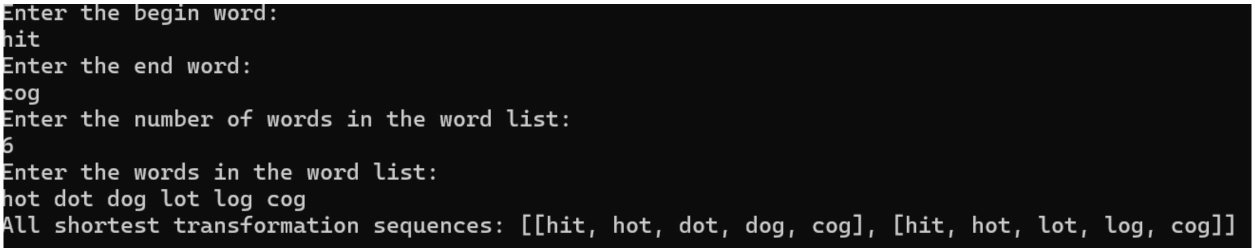
```
import java.util.*;


public class Problem8 {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the begin word:");
    String beginWord = sc.next();
    System.out.println("Enter the end word:");
    String endWord = sc.next();
    System.out.println("Enter the number of words in the word list:");
    int n = sc.nextInt();
    List<String> wordList = new ArrayList<>();
    System.out.println("Enter the words in the word list:");
    for (int i = 0; i < n; i++) wordList.add(sc.next());
    System.out.println("Shortest transformation sequence length: " + ladderLength(beginWord, endWord, wordList));
  }


  public static int ladderLength(String beginWord, String endWord, List<String> wordList) {
    Set<String> wordSet = new HashSet<>(wordList);
    if (!wordSet.contains(endWord)) return 0;
    Queue<String> queue = new LinkedList<>();
    queue.add(beginWord);
    int steps = 1;
    while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                String word = queue.poll();
                char[] chars = word.toCharArray();
                for (int j = 0; j < chars.length; j++) {
                    char original = chars[j];
                    for (char c = 'a'; c <= 'z'; c++) {
                        chars[j] = c;
                        String nextWord = new String(chars);
                        if (nextWord.equals(endWord)) return steps + 1;
                        if (wordSet.contains(nextWord)) {
                            queue.add(nextWord);
                            wordSet.remove(nextWord);
                        }
                    }
                    chars[j] = original;
                }
            }
            steps++;
        }
        return 0;
    }
}
```



```
Enter the begin word:
lost
Enter the end word:
cost
Enter the number of words in the word list:
6
Enter the words in the word list:
most frost post cost host lost
Shortest transformation sequence length: 2
```

9.Word ladder 2

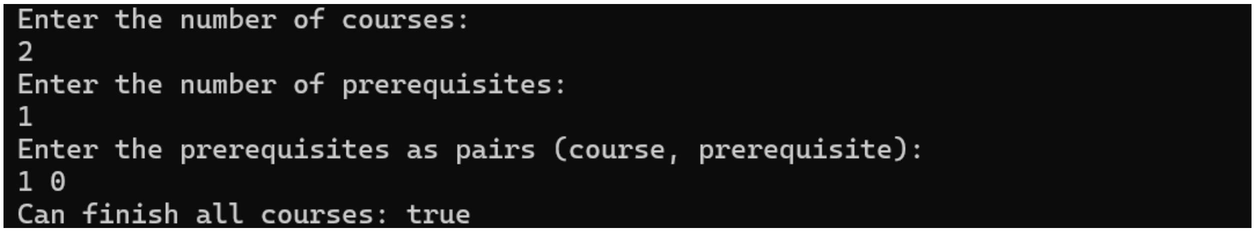
```
import java.util.*;

public class Problem9 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the begin word:");
        String beginWord = sc.next();
        System.out.println("Enter the end word:");
        String endWord = sc.next();
        System.out.println("Enter the number of words in the word list:");
        int n = sc.nextInt();
        List<String> wordList = new ArrayList<>();
        System.out.println("Enter the words in the word list:");
        for (int i = 0; i < n; i++) wordList.add(sc.next());

        List<List<String>> result = findLadders(beginWord, endWord, wordList);
        System.out.println("All shortest transformation sequences: " + result);
    }

    public static List<List<String>> findLadders(String beginWord, String endWord, List<String> wordList) {
        Set<String> wordSet = new HashSet<>(wordList);
        List<List<String>> result = new ArrayList<>();
        if (!wordSet.contains(endWord)) return result;

        Map<String, List<String>> graph = new HashMap<>();
        Queue<String> queue = new LinkedList<>();
        queue.add(beginWord);
        Map<String, Integer> distance = new HashMap<>();
        distance.put(beginWord, 0);

        while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                String current = queue.poll();
                for (String neighbor : getNeighbors(current, wordSet)) {
                    if (!distance.containsKey(neighbor)) {
                        distance.put(neighbor, distance.get(current) + 1);
                        queue.add(neighbor);
                    }
                    graph.computeIfAbsent(current, k -> new ArrayList<>()).add(neighbor);
                }
            }
        }

        List<String> path = new ArrayList<>();
        path.add(beginWord);
        dfs(beginWord, endWord, graph, distance, path, result);
        return result;
    }

    private static void dfs(String current, String endWord, Map<String, List<String>> graph,
                            Map<String, Integer> distance, List<String> path, List<List<String>> result) {
        if (current.equals(endWord)) {
            result.add(new ArrayList<>(path));
            return;
        }
        if (!graph.containsKey(current)) return;

        for (String neighbor : graph.get(current)) {
            if (distance.get(neighbor) == distance.get(current) + 1) {
                path.add(neighbor);
                dfs(neighbor, endWord, graph, distance, path, result);
                path.remove(path.size() - 1);
            }
        }
    }

    private static List<String> getNeighbors(String word, Set<String> wordSet) {
        List<String> neighbors = new ArrayList<>();
        char[] chars = word.toCharArray();
        for (int i = 0; i < chars.length; i++) {
            char original = chars[i];
            for (char c = 'a'; c <= 'z'; c++) {
                chars[i] = c;
                String newWord = new String(chars);
                if (wordSet.contains(newWord) && !newWord.equals(word)) {
                    neighbors.add(newWord);
                }
            }
            chars[i] = original;
        }
        return neighbors;
    }
}
```



```
Enter the begin word:
hit
Enter the end word:
cog
Enter the number of words in the word list:
6
Enter the words in the word list:
hot dot dog lot log cog
All shortest transformation sequences: [[hit, hot, dot, dog, cog], [hit, hot, lot, log, cog]]
```

10.Course Schedule

```
import java.util.*;

public class Problem10 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of courses:");
        int numCourses = sc.nextInt();
        System.out.println("Enter the number of prerequisites:");
        int n = sc.nextInt();
        int[][] prerequisites = new int[n][2];
        System.out.println("Enter the prerequisites as pairs (course, prerequisite):");
        for (int i = 0; i < n; i++) {
            prerequisites[i][0] = sc.nextInt();
            prerequisites[i][1] = sc.nextInt();
        }
        System.out.println("Can finish all courses: " + canFinish(numCourses, prerequisites));
    }

    public static boolean canFinish(int numCourses, int[][] prerequisites) {
        Map<Integer, List<Integer>> graph = new HashMap<>();
        int[] indegree = new int[numCourses];
        for (int[] pre : prerequisites) {
            graph.putIfAbsent(pre[1], new ArrayList<>());
            graph.get(pre[1]).add(pre[0]);
            indegree[pre[0]]++;
        }
        Queue<Integer> queue = new LinkedList<>();
        for (int i = 0; i < numCourses; i++) if (indegree[i] == 0) queue.add(i);
        int count = 0;
        while (!queue.isEmpty()) {
            int curr = queue.poll();
            count++;
            if (graph.containsKey(curr)) {
                for (int next : graph.get(curr)) {
                    indegree[next]--;
                    if (indegree[next] == 0) queue.add(next);
                }
            }
        }
        return count == numCourses;
    }
}
```



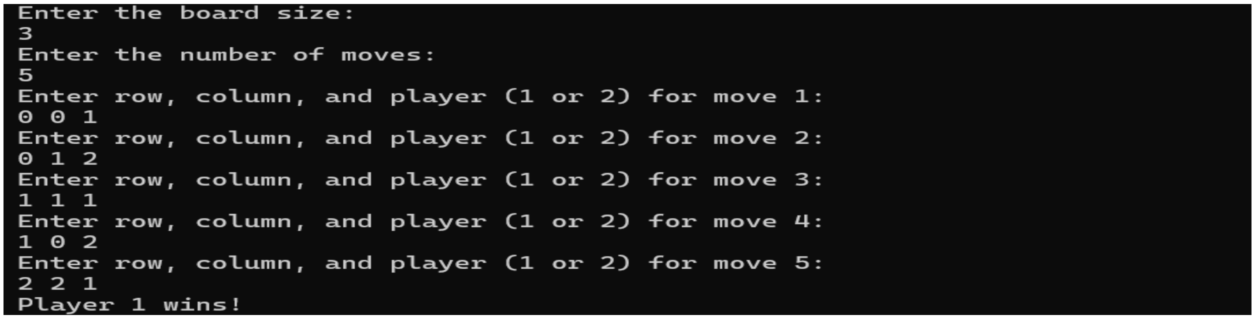
```
Enter the number of courses:
2
Enter the number of prerequisites:
1
Enter the prerequisites as pairs (course, prerequisite):
1 0
Can finish all courses: true
```

11. Design Tic Tac Toe

```
import java.util.Scanner;

public class Problem11 {
```

```
static class TicTacToe {
    private int[] rows, cols;
    private int diagonal, antiDiagonal, n;

    public TicTacToe(int n) {
        this.n = n;
        rows = new int[n];
        cols = new int[n];
        diagonal = 0;
        antiDiagonal = 0;
    }

    public int move(int row, int col, int player) {
        int add = (player == 1) ? 1 : -1;
        rows[row] += add;
        cols[col] += add;

        if (row == col) diagonal += add;
        if (row + col == n - 1) antiDiagonal += add;

        if (Math.abs(rows[row]) == n || Math.abs(cols[col]) == n ||
            Math.abs(diagonal) == n || Math.abs(antiDiagonal) == n) {
            return player;
        }
        return 0;
    }
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
```

```
        System.out.println("Enter the board size:");
        int n = sc.nextInt();
        TicTacToe game = new TicTacToe(n);
        System.out.println("Enter the number of moves:");
        int moves = sc.nextInt();

        for (int i = 0; i < moves; i++) {
            System.out.println("Enter row, column, and player (1 or 2) for move " + (i + 1) + ":");
            int row = sc.nextInt();
            int col = sc.nextInt();
            int player = sc.nextInt();

            int result = game.move(row, col, player);
            if (result == 1) {
                System.out.println("Player 1 wins!");
                return;
            } else if (result == 2) {
                System.out.println("Player 2 wins!");
                return;
            }
        }
        System.out.println("No winner after all moves.");
    }
}
```



```
Enter the board size:
3
Enter the number of moves:
5
Enter row, column, and player (1 or 2) for move 1:
0 0 1
Enter row, column, and player (1 or 2) for move 2:
0 1 2
Enter row, column, and player (1 or 2) for move 3:
1 1 1
Enter row, column, and player (1 or 2) for move 4:
1 0 2
Enter row, column, and player (1 or 2) for move 5:
2 2 1
Player 1 wins!
```