DSA Practice Solutions – 19.11.24

NAME: N J Meenakshi

DEPARTMENT: CSBS

ROLL NO.: 22CB028

1. Minimum Path Sum

   Code:

```
import java.util.Scanner;
class MinimumPathSum {
    public static int minPathSum(int[][] grid) { int m = grid.length, n = grid[0].length;
for (int j = 1; j < n; j++) {
        grid[0][j] += grid[0][j - 1];
    }
    for (int i = 1; i < m; i++) grid[i][0] += grid[i - 1][0];
    for (int i = 1; i < m; i++) {
        for (int j = 1; j < n; j++) {
            grid[i][j] += Math.min(grid[i - 1][j], grid[i][j - 1]);
        }
    }
    return grid[m - 1][n - 1];
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter m:"); int m = scanner.nextInt();
        System.out.println("Enter n:"); int n = scanner.nextInt();
        int[][] grid = new int[m][n];
        System.out.println("Enter the grid values row by row:");
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                grid[i][j] = scanner.nextInt();
            }
        }
        int result = minPathSum(grid);
        System.out.println("The minimum path sum is: " + result); scanner.close();
    }
}
```

   Output:

```
Enter m:
3
Enter n:
3
Enter the grid values row by row:
1 3 1
1 5 1
```

```
4 2 1
The minimum path sum is: 7
```

Time Complexity: O(m*n)

2. **Validate Binary Search Tree**
   Code:

```java
class Node {
    int data;
    Node left, right;
    Node(int value) {
        data = value;
        left = right = null;
    }
}
public class ValidateBST {
    public static boolean solution(Node root, int min, int max) {
        if (root == null) return true;
        if (root.data < min || root.data > max) return false;
        return solution(root.left, min, root.data - 1) && solution(root.right, root.data +
1, max);
    }
    public static void main(String[] args) {
        Node root = new Node(4);
        root.left = new Node(2);
        root.right = new Node(5);
        root.left.left = new Node(1);
        root.left.right = new Node(3);
        if (solution(root,Integer.MIN_VALUE,Integer.MAX_VALUE)) {
            System.out.println("True");
        }
        else {
            System.out.println("False");
        }
    }
}
```

Output: True
Time Complexity: O(n)

3. **Word ladder**
   Code:

```java
import java.util.*;
class WordLadder {
    static int solution(String start, String target, Set<String> D) {
        if (start.equals(target))
            return 0;
        if (!D.contains(target))
            return 0;
        int level = 0, wordLength = start.length();
        Queue<String> Q = new LinkedList<>();
        Q.add(start);
        while (!Q.isEmpty()) {
            ++level;
            int sizeOfQ = Q.size();
            for (int i = 0; i < sizeOfQ; ++i) {
                char[] word = Q.peek().toCharArray();
                Q.remove();
                for (int pos = 0; pos < wordLength; ++pos) {
                    char origChar = word[pos];
                    for (char c = 'a'; c <= 'z'; ++c) {
                        word[pos] = c;
                        if (String.valueOf(word).equals(target))
                            return level + 1;
                        if (!D.contains(String.valueOf(word)))
                            continue;
                        D.remove(String.valueOf(word));
                        Q.add(String.valueOf(word));
                    }
                    word[pos] = origChar;
                }
            }
        }
        return 0;
    }
    public static void main(String[] args) {
        Set<String> D = new HashSet<>();
        D.add("poon");
        D.add("plee");
        D.add("same");
        D.add("poie");
        D.add("plie");
        D.add("poin");
        D.add("plea");
        String start = "toon";
        String target = "plea";
        System.out.print(solution(start, target, D));
    }
}
```

Output: 7
Time Complexity: O(N*M)

4. **Word ladder -II**
Code:

```java
import java.util.*;
public class WordLadder2 {
    static List<List<String>> solution(String beginWord, String endWord, List<String>
wordList) {
        List<List<String>> ans = new ArrayList<>();
        Set<String> vis = new HashSet<>(wordList);
        List<String> usedOnLevel = new ArrayList<>();
        Queue<List<String>> qu = new LinkedList<>();
        List<String> temp = new ArrayList<>();
        temp.add(beginWord);
        qu.add(temp);
        int level = 0;
        while (!qu.isEmpty()) {
            List<String> vec = qu.poll();
            if (vec.size() > level) {
                level++;
                for (String str : usedOnLevel) {
                    vis.remove(str);
                }
                usedOnLevel.clear();
            }
            String last = vec.get(vec.size() - 1);
            if (last.equals(endWord)) {
                if (ans.size() == 0) {
                    ans.add(vec);
                }
                else if (ans.get(0).size() == vec.size()) {
                    ans.add(vec);
                }
            }
            for (int i = 0; i < last.length(); i++) {
                char[] arr = last.toCharArray();
                char org = arr[i];
                for (char ch = 'a'; ch <= 'z'; ch++) {
                    arr[i] = ch;
                    String newStr = new String(arr);
                    if (vis.contains(newStr)) {
                        List<String> tempVec = new ArrayList<>(vec);
                        tempVec.add(newStr);
                        qu.add(tempVec);
                        usedOnLevel.add(newStr);
                    }
                }
                arr[i] = org;
                last = new String(arr);
            }
        }
        return ans;
    }
    public static void main(String[] args) {
        List<String> wordList = new ArrayList<>();
        wordList.add("poon");
        wordList.add("plee");
        wordList.add("same");
        wordList.add("poie");
        wordList.add("plie");
        wordList.add("poin");
        wordList.add("plea");
        String start = "toon";
        String target = "plea";
        List<List<String>> ans = solution(start, target, wordList);
        for (List<String> a : ans) {
            for (String s : a) {
                System.out.print(s + " ");
            }
            System.out.println("\nLength of sequence is " + a.size());
        }
    }
}
```

Output:
toon poon poin poie plie plee plea
Length of sequence is 7
Time Complexity: O(N*M*26)

5. **Course schedule**
   Code:

```
import java.util.*;
public class CourseSchedule {
    static class Pair {
        int first, second;
        Pair(int first, int second) {
            this.first = first;
            this.second = second;
        }
    }
    static ArrayList<ArrayList<Integer>> makeGraph(int numTasks, Vector<Pair>
prerequisites) {
        ArrayList<ArrayList<Integer>> graph = new ArrayList<>(numTasks);
        for (int i = 0; i < numTasks; i++) {
            graph.add(new ArrayList<>());
        }
        for (Pair pre : prerequisites) {
            graph.get(pre.second).add(pre.first);
        }
        return graph;
    }
    static boolean dfsCycle(ArrayList<ArrayList<Integer>> graph, int node, boolean[]
onPath, boolean[] visited) {
        if (visited[node]) {
            return false;
        }
        onPath[node] = visited[node] = true;
        for (int neighbor : graph.get(node)) {
            if (onPath[neighbor] || dfsCycle(graph, neighbor, onPath, visited)) {
                return true;
            }
        }
        onPath[node] = false;
        return false;
    }
    static boolean solution(int numTasks, Vector<Pair> prerequisites) {
        ArrayList<ArrayList<Integer>> graph = makeGraph(numTasks, prerequisites);
        boolean[] onPath = new boolean[numTasks];
        boolean[] visited = new boolean[numTasks];
        for (int i = 0; i < numTasks; i++) {
            if (!visited[i] && dfsCycle(graph, i, onPath, visited)) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        int numTasks = 4;
        Vector<Pair> prerequisites = new Vector<>();
        prerequisites.add(new Pair(1, 0));
        prerequisites.add(new Pair(2, 1));
        prerequisites.add(new Pair(3, 2));
        if (solution(numTasks, prerequisites)) {
            System.out.println("Possible to finish all tasks");
        }
        else {
            System.out.println("Impossible to finish all tasks");
        }
    }
}
```

Output: YES
Time Complexity: O(V+E)