PRACTICE SET – 7

Meena Rhoshini C

Nov 19, 24

1. **Next Permutation**

```
import java.util.*;

public class NextPermutation {

    public static void findNextPermutation(int[] nums) {
        int n = nums.length;
        int i = n - 2;
        while (i >= 0 && nums[i] >= nums[i + 1]) {
            i--;
        }
        if (i >= 0) {
            int j = n - 1;
            while (nums[j] <= nums[i]) {
                j--;
            }
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
        }
        int start = i + 1, end = n - 1;
        while (start < end) {
            int temp = nums[start];
            nums[start] = nums[end];
            nums[end] = temp;
            start++;
            end--;
        }
    }

    public static void main(String[] args) {
        int[] nums1 = {1, 2, 3};
        findNextPermutation(nums1);
        System.out.println(Arrays.toString(nums1));

        int[] nums2 = {3, 2, 1};
        findNextPermutation(nums2);
        System.out.println(Arrays.toString(nums2));

        int[] nums3 = {1, 1, 5};
        findNextPermutation(nums3);
        System.out.println(Arrays.toString(nums3));
    }
```

```
}
```

OUTPUT:

```
C:\Users\Rhoshini\Desktop\dsa>javac NextPermutation.java

C:\Users\Rhoshini\Desktop\dsa>java NextPermutation
[1, 3, 2]
[1, 2, 3]
[1, 5, 1]
```

Time complexity: O(n)
Space complexity: O(1)

---

2. Spiral Matrix

```
import java.util.*;

public class SpiralMatrix {

    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;
        int 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;
    }

    public static void main(String[] args) {
        int[][] matrix1 = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
        System.out.println(spiralOrder(matrix1));

        int[][] matrix2 = {
            {1, 2, 3, 4},
            {5, 6, 7, 8},
            {9, 10, 11, 12}
        };
        System.out.println(spiralOrder(matrix2));
    }
}
```

OUTPUT:

```
C:\Users\Rhoshini\Desktop\dsa>javac SpiralMatrix.java

C:\Users\Rhoshini\Desktop\dsa>java SpiralMatrix
[1, 2, 3, 6, 9, 8, 7, 4, 5]
[1, 2, 3, 4, 8, 12, 11, 10, 9, 5, 6, 7]
```

Time complexity: O(m*n)
Space complexity: O(1)

---

3. Longest Substring Without Repeating Characters

```
import java.util.*;

public class LongestSubstring {

    public static int lengthOfLongestSubstring(String s) {
        Set<Character> set = new HashSet<>();
        int left = 0, right = 0, maxLength = 0;

        while (right < s.length()) {
            if (!set.contains(s.charAt(right))) {
                set.add(s.charAt(right));
                maxLength = Math.max(maxLength, right - left + 1);
                right++;
            } else {
                set.remove(s.charAt(left));
                left++;
            }
```

```
        }

        return maxLength;
    }

    public static void main(String[] args) {
        System.out.println(lengthOfLongestSubstring("abcabcbb"));  // Output: 3
        System.out.println(lengthOfLongestSubstring("bbbbb"));     // Output: 1
        System.out.println(lengthOfLongestSubstring("pwwkew"));    // Output: 3
    }
}
```

OUTPUT:

```
C:\Users\Rhoshini\Desktop\dsa>javac LongestSubstring.java

C:\Users\Rhoshini\Desktop\dsa>java LongestSubstring
3
1
3
```

Time complexity: O(n)
Space complexity: O(k)

---

4. **Remove Linked List Elements**

```
class MyListNode {
    int val;
    MyListNode next;
    MyListNode() {}
    MyListNode(int val) { this.val = val; }
    MyListNode(int val, MyListNode next) { this.val = val; this.next = next; }
}

public class MySolution {
    public MyListNode removeElements(MyListNode head, int val) {
        MyListNode dummy = new MyListNode(0);
        dummy.next = head;
        MyListNode current = dummy;

        while (current.next != null) {
            if (current.next.val == val) {
                current.next = current.next.next;
            } else {
                current = current.next;
            }
        }

        return dummy.next;
    }
```

```
    public static void main(String[] args) {
        MySolution solution = new MySolution();

        MyListNode head = new MyListNode(1, new MyListNode(2, new MyListNode(6, new
MyListNode(3, new MyListNode(4, new MyListNode(5, new MyListNode(6)))))));
        MyListNode result = solution.removeElements(head, 6);
        printList(result);  // Output: [1, 2, 3, 4, 5]

        MyListNode head2 = null;
        result = solution.removeElements(head2, 1);
        printList(result);  // Output: []

        MyListNode head3 = new MyListNode(7, new MyListNode(7, new MyListNode(7, new
MyListNode(7))));
        result = solution.removeElements(head3, 7);
        printList(result);  // Output: []
    }

    public static void printList(MyListNode head) {
        MyListNode current = head;
        while (current != null) {
            System.out.print(current.val + " ");
            current = current.next;
        }
        System.out.println();
    }
}
```

OUTPUT:

```
C:\Users\Rhoshini\Desktop\dsa>javac MySolution.java

C:\Users\Rhoshini\Desktop\dsa>java MySolution
1 2 3 4 5
```

Time complexity: O(n)
Space complexity: O(1)

---

5. Palindrome Linked List

```
public class PalindromeListChecker {

    public boolean isPalindrome(ListNode1 head) {
        if (head == null || head.next == null) {
            return true;
        }

        ListNode1 slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
```

```
            fast = fast.next.next;
        }

        ListNode1 prev = null;
        while (slow != null) {
            ListNode1 nextNode = slow.next;
            slow.next = prev;
            prev = slow;
            slow = nextNode;
        }

        ListNode1 left = head, right = prev;
        while (right != null) {
            if (left.val != right.val) {
                return false;
            }
            left = left.next;
            right = right.next;
        }

        return true;
    }

    public static void main(String[] args) {
        PalindromeListChecker checker = new PalindromeListChecker();

        ListNode1 head1 = new ListNode1(1);
        head1.next = new ListNode1(2);
        head1.next.next = new ListNode1(2);
        head1.next.next.next = new ListNode1(1);
        System.out.println(checker.isPalindrome(head1));

        ListNode1 head2 = new ListNode1(1);
        head2.next = new ListNode1(2);
        System.out.println(checker.isPalindrome(head2));
    }
}

class ListNode1 {
    int val;
    ListNode1 next;
    ListNode1() {}
    ListNode1(int val) { this.val = val; }
    ListNode1(int val, ListNode1 next) { this.val = val; this.next = next; }
}
```

OUTPUT:

```
C:\Users\Rhoshini\Desktop\dsa>javac PalindromeListChecker.java

C:\Users\Rhoshini\Desktop\dsa>java PalindromeListChecker
true
false
```

Time complexity: O(n)
Space complexity: O(1)

---

6. Minimum Path Sum

```
public class MinPathSum {

    public int minPathSum(int[][] grid) {
        int m = grid.length, n = grid[0].length;

        for (int i = 1; i < m; i++) {
            grid[i][0] += grid[i - 1][0];
        }
        for (int j = 1; j < n; j++) {
            grid[0][j] += grid[0][j - 1];
        }

        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) {
        MinPathSum solution = new MinPathSum();
        int[][] grid1 = {{1, 3, 1}, {1, 5, 1}, {4, 2, 1}};
        System.out.println(solution.minPathSum(grid1)); // Output: 7

        int[][] grid2 = {{1, 2, 3}, {4, 5, 6}};
        System.out.println(solution.minPathSum(grid2)); // Output: 12
    }
}
```

OUTPUT:

```
C:\Users\Rhoshini\Desktop\dsa>javac MinPathSum.java

C:\Users\Rhoshini\Desktop\dsa>java MinPathSum
7
12
```

Time complexity: O(m*n)
Space complexity: O(1)

---

7. **Validate Binary Search Tree**

```
public class BSTValidator {

    public boolean isValidBST(TreeNode root) {
        return isValid(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    private boolean isValid(TreeNode node, long min, long max) {
        if (node == null) return true;
        if (node.val <= min || node.val >= max) return false;
        return isValid(node.left, min, node.val) && isValid(node.right, node.val, max);
    }

    public static void main(String[] args) {
        BSTValidator validator = new BSTValidator();

        TreeNode root1 = new TreeNode(2);
        root1.left = new TreeNode(1);
        root1.right = new TreeNode(3);
        System.out.println(validator.isValidBST(root1));

        TreeNode root2 = new TreeNode(5);
        root2.left = new TreeNode(1);
        root2.right = new TreeNode(4);
        root2.right.left = new TreeNode(3);
        root2.right.right = new TreeNode(6);
        System.out.println(validator.isValidBST(root2));
    }
}

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode() {}
    TreeNode(int val) { this.val = val; }
    TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}
```

OUTPUT:

```
C:\Users\Rhoshini\Desktop\dsa>javac BSTValidator.java

C:\Users\Rhoshini\Desktop\dsa>java BSTValidator
true
false
```

Time complexity: O(n)
Space complexity: O(h)

---

8. Word Ladder

```
import java.util.*;

public class WordLadder {

   public int ladderLength(String beginWord, String endWord, List<String> wordList) {
      if (!wordList.contains(endWord)) return 0;

      Set<String> wordSet = new HashSet<>(wordList);
      Queue<String> queue = new LinkedList<>();
      queue.offer(beginWord);
      int level = 1;

      while (!queue.isEmpty()) {
         int size = queue.size();
         for (int i = 0; i < size; i++) {
            String word = queue.poll();
            char[] wordArray = word.toCharArray();
            for (int j = 0; j < word.length(); j++) {
               char originalChar = wordArray[j];
               for (char c = 'a'; c <= 'z'; c++) {
                  wordArray[j] = c;
                  String newWord = new String(wordArray);
                  if (newWord.equals(endWord)) return level + 1;
                  if (wordSet.contains(newWord)) {
                     queue.offer(newWord);
                     wordSet.remove(newWord);
                  }
               }
               wordArray[j] = originalChar;
            }
         }
         level++;
      }
      return 0;
   }
   public static void main(String[] args) {
      WordLadder wl = new WordLadder();

      List<String> wordList1 = Arrays.asList("hot", "dot", "dog", "lot", "log", "cog");
      System.out.println(wl.ladderLength("hit", "cog", wordList1));

      List<String> wordList2 = Arrays.asList("hot", "dot", "dog", "lot", "log");
      System.out.println(wl.ladderLength("hit", "cog", wordList2));
   }
}
```

OUTPUT:

```
C:\Users\Rhoshini\Desktop\dsa>javac WordLadder.java

C:\Users\Rhoshini\Desktop\dsa>java WordLadder
5
0
```

Time complexity: O(m*n*26)
Space complexity: O(m*n)

---

9. Word Ladder II

```
import java.util.*;

public class WordLadderII {

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

        Set<String> wordSet = new HashSet<>(wordList);
        Map<String, List<String>> graph = new HashMap<>();
        Queue<String> queue = new LinkedList<>();
        Set<String> visited = new HashSet<>();
        queue.offer(beginWord);
        visited.add(beginWord);
        boolean found = false;

        while (!queue.isEmpty() && !found) {
            Set<String> levelVisited = new HashSet<>();
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                String word = queue.poll();
                char[] wordArray = word.toCharArray();
                for (int j = 0; j < word.length(); j++) {
                    char originalChar = wordArray[j];
                    for (char c = 'a'; c <= 'z'; c++) {
                        wordArray[j] = c;
                        String newWord = new String(wordArray);
                        if (wordSet.contains(newWord)) {
                            if (!visited.contains(newWord)) {
                                queue.offer(newWord);
                                levelVisited.add(newWord);
                            }
                            graph.computeIfAbsent(word, k -> new ArrayList<>()).add(newWord);
                        }
                    }
                    wordArray[j] = originalChar;
```

```
            }
        }
        visited.addAll(levelVisited);
        if (!levelVisited.isEmpty()) found = true;
    }

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

    return result;
}
private void dfs(String current, String endWord, Map<String, List<String>> graph,
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)) {
        path.add(neighbor);
        dfs(neighbor, endWord, graph, path, result);
        path.remove(path.size() - 1);
    }
}
public static void main(String[] args) {
    WordLadderII wl = new WordLadderII();

    List<String> wordList1 = Arrays.asList("hot", "dot", "dog", "lot", "log", "cog");
    System.out.println(wl.findLadders("hit", "cog", wordList1));

    List<String> wordList2 = Arrays.asList("hot", "dot", "dog", "lot", "log");
    System.out.println(wl.findLadders("hit", "cog", wordList2));
}
}
```

OUTPUT:

```
C:\Users\Rhoshini\Desktop\dsa>javac WordLadderII.java

C:\Users\Rhoshini\Desktop\dsa>java WordLadderII
[]
[]
```

Time complexity: O(N * M + N * M * log N)
Space complexity: O(N*M)

10. Course Schedule

```
import java.util.*;

public class CourseScheduler {

    public boolean canCompleteCourses(int totalCourses, int[][] prereqs) {
        List<List<Integer>> courseGraph = new ArrayList<>();
        int[] courseIndegree = new int[totalCourses];

        for (int i = 0; i < totalCourses; i++) {
            courseGraph.add(new ArrayList<>());
        }

        for (int[] prereq : prereqs) {
            int course = prereq[0];
            int prereqCourse = prereq[1];
            courseGraph.get(prereqCourse).add(course);
            courseIndegree[course]++;
        }

        Queue<Integer> availableCourses = new LinkedList<>();
        for (int i = 0; i < totalCourses; i++) {
            if (courseIndegree[i] == 0) {
                availableCourses.offer(i);
            }
        }

        int processedCourses = 0;
        while (!availableCourses.isEmpty()) {
            int course = availableCourses.poll();
            processedCourses++;

            for (int nextCourse : courseGraph.get(course)) {
                courseIndegree[nextCourse]--;
                if (courseIndegree[nextCourse] == 0) {
                    availableCourses.offer(nextCourse);
                }
            }
        }

        return processedCourses == totalCourses;
    }

    public static void main(String[] args) {
        CourseScheduler scheduler = new CourseScheduler();

        int totalCourses1 = 2;
        int[][] prereqs1 = {{1, 0}};
        System.out.println(scheduler.canCompleteCourses(totalCourses1, prereqs1));
```

```
        int totalCourses2 = 2;
        int[][] prereqs2 = {{1, 0}, {0, 1}};
        System.out.println(scheduler.canCompleteCourses(totalCourses2, prereqs2));
    }
}
```

OUTPUT:

```
C:\Users\Rhoshini\Desktop\dsa>javac CourseScheduler.java

C:\Users\Rhoshini\Desktop\dsa>java CourseScheduler
true
false
```

Time complexity: O(v+e)
Space complexity: O(v+e)

---

11. Design Tic-Tac-Toe

```
public class TicTacToeGame {
    private int size;
    private int[][] counts;

    public TicTacToeGame(int size) {
        this.size = size;
        counts = new int[2][(size << 1) + 2];
    }

    public int makeMove(int row, int col, int player) {
        int[] currentPlayerCounts = counts[player - 1];
        ++currentPlayerCounts[row];
        ++currentPlayerCounts[size + col];
        if (row == col) {
            ++currentPlayerCounts[size << 1];
        }
        if (row + col == size - 1) {
            ++currentPlayerCounts[(size << 1) | 1];
        }
        if (currentPlayerCounts[row] == size || currentPlayerCounts[size + col] == size ||
currentPlayerCounts[size << 1] == size || currentPlayerCounts[(size << 1) | 1] == size) {
            return player;
        }
        return 0;
    }

    public static void main(String[] args) {
        TicTacToeGame game = new TicTacToeGame(3);

        // Example moves
```

```
        System.out.println(game.makeMove(0, 0, 1));
        System.out.println(game.makeMove(0, 1, 2));
        System.out.println(game.makeMove(1, 1, 1));
        System.out.println(game.makeMove(0, 2, 2));
        System.out.println(game.makeMove(2, 2, 1));
    }
}
```

OUTPUT:

```
C:\Users\Rhoshini\Desktop\dsa>java TicTacToeGame
0
0
0
0
1
```

Time complexity: O(1)
Space complexity: O(n)