

Date: 12/11/24

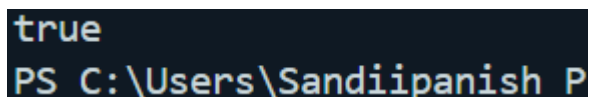
DSA Practice Problems

1. Anagram Program

Code:

```
public class AnagramCheck {  
  
    public static boolean areAnagrams(String s1, String s2) {  
  
        if (s1.length() != s2.length()) return false;  
  
        int[] count = new int[26];  
  
        for (int i = 0; i < s1.length(); i++) {  
  
            count[s1.charAt(i) - 'a']++;  
  
            count[s2.charAt(i) - 'a']--;  
  
        }  
  
        for (int c : count) {  
  
            if (c != 0) return false;  
  
        }  
  
        return true;  
  
    }  
  
    public static void main(String[] args) {  
  
        System.out.println(areAnagrams("geeks", "kseeg"));  
  
    }  
}
```

Output:



```
true  
PS C:\Users\Sandiipnish P
```

Time Complexity: $O(n)$

2. Row with max 1's

Code:

```
public class MaxOnesRow {

    public static int rowWithMaxOnes(int[][] arr) {

        int n = arr.length;

        int m = arr[0].length;

        int maxRowIndex = -1;

        int j = m - 1;

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

            while (j >= 0 && arr[i][j] == 1) {

                j--;

                maxRowIndex = i;

            }

        }

        return maxRowIndex;

    }

    public static void main(String[] args) {

        int[][] arr1 = {

            {0, 1, 1, 1},

            {0, 0, 1, 1},

            {1, 1, 1, 1},

            {0, 0, 0, 0}

        };

        System.out.println(rowWithMaxOnes(arr1));

    }

}
```

```
}
```

Output:

```
2
PS C:\Users\Sandiipani P
```

Time Complexity: $O(n+m)$

3. Longest consecutive subsequence

Code:

```
import java.util.HashSet;
```

```
public class LongestConsecutiveSubsequence {

    public static int findLongestConsecutiveSubsequence(int[] arr) {

        HashSet<Integer> set = new HashSet<>();

        for (int num : arr) {

            set.add(num);

        }

        int maxLength = 0;

        for (int num : arr) {

            if (!set.contains(num - 1)) {

                int currentNum = num;

                int currentLength = 1;

                while (set.contains(currentNum + 1)) {

                    currentNum++;

                    currentLength++;

                }

            }

        }

    }

}
```

```

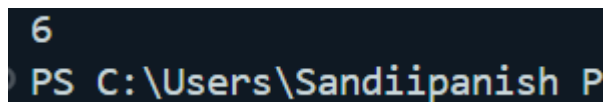
        maxLength = Math.max(maxLength, currentLength);
    }
}

return maxLength;
}

public static void main(String[] args) {
    int[] arr1 = {2, 6, 1, 9, 4, 5, 3};
    System.out.println(findLongestConsecutiveSubsequence(arr1));
}
}

```

Output:



```

6
PS C:\Users\Sandiipani P

```

Time Complexity: $O(n)$

4. Longest Palindrome in a String

Code:

```

public class LongestPalindromicSubstring {
    public static String longestPalindrome(String s) {
        if (s == null || s.length() < 1) return "";

        int start = 0, end = 0;

        for (int i = 0; i < s.length(); i++) {
            int len1 = expandAroundCenter(s, i, i);
            int len2 = expandAroundCenter(s, i, i + 1);

```

```

        int len = Math.max(len1, len2);

        if (len > end - start) {

            start = i - (len - 1) / 2;

            end = i + len / 2;

        }

    }

    return s.substring(start, end + 1);

}

private static int expandAroundCenter(String s, int left, int right) {

    while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {

        left--;

        right++;

    }

    return right - left - 1;

}

public static void main(String[] args) {

    System.out.println(longestPalindrome("aaaabbaa"));

}

}

```

Output:



```

aabbbaa
PS C:\Users\Sandiipanish P

```

Time Complexity: $O(n^2)$

5. Rat in a Maze

Code:

```
import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class RatInMaze {

    public static List<String> findPaths(int[][] mat, int n) {

        List<String> paths = new ArrayList<>();

        if (mat[0][0] == 0 || mat[n - 1][n - 1] == 0) return paths;

        boolean[][] visited = new boolean[n][n];

        findPathsUtil(mat, n, 0, 0, "", visited, paths);

        Collections.sort(paths);

        return paths;

    }

    private static void findPathsUtil(int[][] mat, int n, int row, int col, String path, boolean[][] visited,
List<String> paths) {

        if (row == n - 1 && col == n - 1) {

            paths.add(path);

            return;

        }

        if (row < 0 || col < 0 || row >= n || col >= n || mat[row][col] == 0 || visited[row][col]) return;
```

```

        visited[row][col] = true;

        findPathsUtil(mat, n, row + 1, col, path + "D", visited, paths);
        findPathsUtil(mat, n, row - 1, col, path + "U", visited, paths);
        findPathsUtil(mat, n, row, col + 1, path + "R", visited, paths);
        findPathsUtil(mat, n, row, col - 1, path + "L", visited, paths);

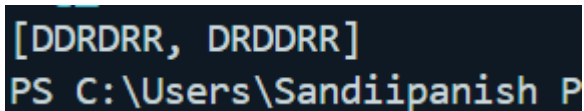
        visited[row][col] = false;
    }

    public static void main(String[] args) {
        int[][] mat1 = {
            {1, 0, 0, 0},
            {1, 1, 0, 1},
            {1, 1, 0, 0},
            {0, 1, 1, 1}
        };

        System.out.println(findPaths(mat1, 4));
    }
}

```

Output:



```

[DDRDRR, DRDDR]
PS C:\Users\Sandiipani P

```

Time Complexity: $O(3^{(n^2)})$