

DSA PRACTICE – DAY 3

Name: Dhejan R

Reg No: 22IT022

Date: 12/11/2024

1. Anagram Program

Code Solution:

```
public class Anagram{
    public static boolean stringChecker(String s1, String s2){
        if(s1.length()!=s2.length()) return false;
        int[] alpha=new int[26];

        for(int i=0; i<s1.length(); i++){
            char c=s1.charAt(i);
            int pos=c-'a';
            alpha[pos]++;
        }
        for(int i=0; i<s2.length(); i++){
            char c=s2.charAt(i);
            int pos=c-'a';
            if (alpha[pos]==0){
                return false;
            }else{
                alpha[pos]--;
            }
        }
        return true;
    }
}
```

Output:

The screenshot displays a coding environment with two main panels. The left panel, titled 'Output Window', shows 'Compilation Results' and a green checkmark indicating 'Problem Solved Successfully'. It also displays performance metrics: 'Test Cases Passed: 1115 / 1115', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 2 / 2', and 'Your Total Score: 27'. The right panel shows the code solution for the Anagram Program, which is a Java class named 'Anagram' with a static method 'stringChecker' that checks if two strings are anagrams by comparing character frequencies in an array of size 26.

Metric	Value
Test Cases Passed	1115 / 1115
Attempts : Correct / Total	1 / 1
Accuracy	100%
Points Scored	2 / 2
Time Taken	0.27
Your Total Score	27

```
1 // Driver Code Starts
29
30
31 class Solution {
32     // Function is to check whether two strings are anagram of each
33     public static boolean areAnagrams(String s1, String s2) {
34
35         if(s1.length()!=s2.length()) return false;
36         int[] alpha=new int[26];
37
38         for(int i=0; i<s1.length(); i++){
39             char c=s1.charAt(i);
40             int pos=c-'a';
41             alpha[pos]++;
42         }
43         for(int i=0; i<s2.length(); i++){
44             char c=s2.charAt(i);
45             int pos=c-'a';
46             if (alpha[pos]==0){
47                 return false;
48             }else{
49                 alpha[pos]--;
50             }
51         }
52         return true;
53     }
54 }
```

Time Complexity: $O(1)$

Space Complexity: $O(1)$

2. Row with Max 1's

Code Solution:

```
public class Finder{
    public static int findingMaxOne(int[][] matrix){
        int row=0;
        int col=matrix[0].length-1;
        int res=0;

        while (row>=0 && col>=0 && row<matrix.length &&
col<matrix[0].length){
            if (matrix[row][col]==0){
                row++;
            }
            else{
                res=row;
                col--;
            }
        }
        return res+1;
    }

    public static void main(String[] args){
        int[][] matrix = {{0, 0, 0, 1},
        {0, 1, 1, 1},
        {1, 1, 1, 1},
        {0, 0, 0, 0}};
        System.out.println(findingMaxOne(matrix));
    }
}
```

Output:

```
Microsoft Windows [Version 10.0.22631.4391]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Legion\Desktop\SDE\DSA Coding Questions>javac Finder.java

C:\Users\Legion\Desktop\SDE\DSA Coding Questions>java Finder
3

C:\Users\Legion\Desktop\SDE\DSA Coding Questions>
```

Time Complexity: $O(n+m)$

Space Complexity: $O(1)$

3. Longest Consecutive Subsequence

Code Solution:

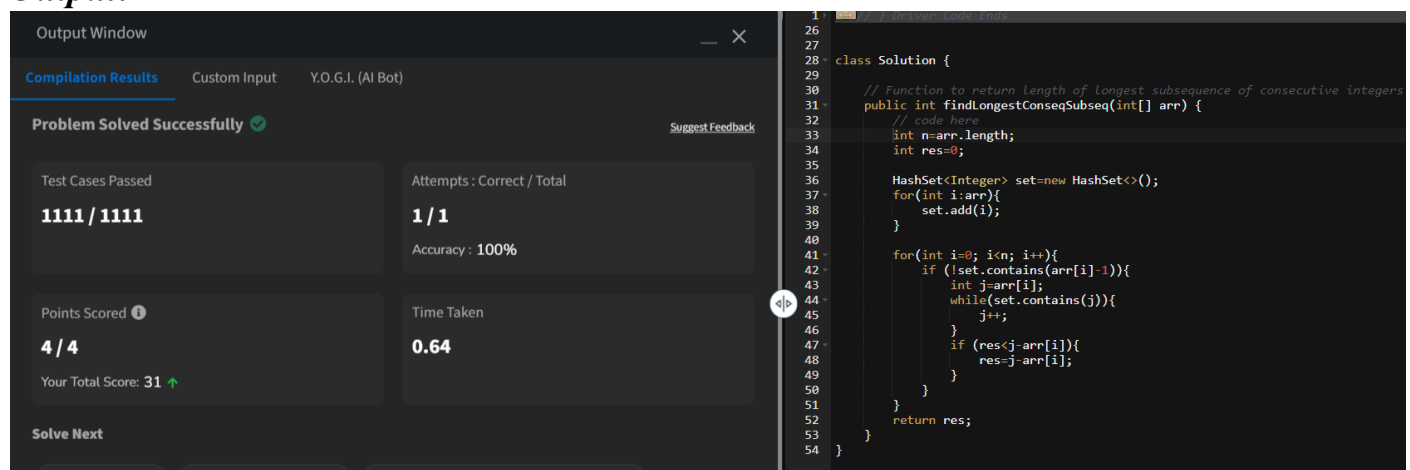
```
import java.util.*;

public class longestConsecutiveSubsequence{
    public static int longestSubsequence(int[] arr){
        int n=arr.length;
        int res=0;

        HashSet<Integer> set=new HashSet<>();
        for(int i:arr){
            set.add(i);
        }

        for(int i=0; i<n; i++){
            if (!set.contains(arr[i]-1)){
                int j=arr[i];
                while(set.contains(j)){
                    j++;
                }
                if (res<j-arr[i]){
                    res=j-arr[i];
                }
            }
        }
        return res;
    }
}
```

Output:



The screenshot shows a coding platform interface. On the left, the 'Output Window' displays 'Compilation Results' for 'Y.O.G.I. (AI Bot)'. It indicates 'Problem Solved Successfully' with a green checkmark. Below this, it shows 'Test Cases Passed: 1111 / 1111', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 4 / 4', and 'Your Total Score: 31'. On the right, the code editor shows the Java solution for the 'Longest Consecutive Subsequence' problem. The code defines a 'Solution' class with a 'findLongestConseqSubseq' method that uses a HashSet to track elements and a loop to find the longest consecutive subsequence.

Time Complexity: $O(n)$

Space Complexity: $O(n)$

4. Longest Palindrome in a String

Code Solution:

```
class Solution{
    String longestPalindrome(String s){
        int n=s.length();
        int start=0;
        int maxLen=1;

        for(int i=0; i<n; i++){
            for(int j=0; j<=1; j++){
                int low=i;
                int high=i+j;
                while(low>=0 && high<n && s.charAt(low)==s.charAt(high)){
                    int curr=high-low+1;
                    if (curr>maxLen){
                        start=low;
                        maxLen=curr;
                    }
                    low--;
                    high++;
                }
            }
        }
        return s.substring(start, start+maxLen);
    }
}
```

Output:

The screenshot displays a coding platform interface. On the left, the 'Output Window' shows 'Compilation Results' with a green checkmark indicating 'Problem Solved Successfully'. It also displays 'Test Cases Passed: 75 / 75', 'Attempts: 1 / 1', 'Accuracy: 100%', 'Points Scored: 4 / 4', and 'Time Taken: 0.07'. At the bottom, there are buttons for 'Solve Next' and 'Longest Common Substring', 'Longest Palindrome in a String', and 'Longest Prefix Suffix'. On the right, the code editor shows the Java solution for the 'Longest Palindrome in a String' problem, which matches the code provided in the 'Code Solution' block.

Time Complexity: $O(n^2)$

Space Complexity: $O(1)$

5. Rat in a Maze

Code Solution:

```
class Solution {
    public ArrayList<String> findPath(int[][] mat) {
        ArrayList<String> paths=new ArrayList<>();
        int n=mat.length;
        if (mat[0][0]==0 || mat[n-1][n-1]==0) {
            return paths;
        }

        boolean[][] visited=new boolean[n][n];
        backtrack(mat, 0, 0, n, "", paths, visited);

        Collections.sort(paths);
        return paths;
    }

    private void backtrack(int[][] mat, int x, int y, int n, String path, ArrayList<String>
paths, boolean[][] visited) {
        if (x==n-1 && y==n-1) {
            paths.add(path);
            return;
        }
        visited[x][y]=true;
        if (isSafe(mat, x+1, y, n, visited)) {
            backtrack(mat, x+1, y, n, path+"D", paths, visited);
        }
        if (isSafe(mat, x, y-1, n, visited)) {
            backtrack(mat, x, y-1, n, path+"L", paths, visited);
        }
        if (isSafe(mat, x, y+1, n, visited)) {
            backtrack(mat, x, y+1, n, path+"R", paths, visited);
        }
        if (isSafe(mat, x-1, y, n, visited)) {
            backtrack(mat, x-1, y, n, path+"U", paths, visited);
        }
        visited[x][y] = false;
    }

    private boolean isSafe(int[][] mat, int x, int y, int n, boolean[][] visited) {
        return x>=0 && x<n && y>=0 && y<n && mat[x][y]==1 && !visited[x][y];
    }
}
```

Output:

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully

Test Cases Passed

162 / 162

Attempts : Correct / Total

1 / 1

Accuracy : 100%

Points Scored

4 / 4

Your Total Score: 35

Solve Next

[Tower Of Hanoi](#) [Black and White](#) [Rat Maze With Multiple Jumps](#)

```
35 class Solution {
36     public ArrayList<String> findPath(int[][] mat) {
37         ArrayList<String> paths=new ArrayList<>();
38         int n=mat.length;
39         if (mat[0][0]==0 || mat[n-1][n-1]==0) {
40             return paths;
41         }
42
43         boolean[][] visited=new boolean[n][n];
44         backtrack(mat, 0, 0, n, "", paths, visited);
45
46         Collections.sort(paths);
47         return paths;
48     }
49
50     private void backtrack(int[][] mat, int x, int y, int n, String path, ArrayList<String> paths) {
51         if (x==n-1 && y==n-1) {
52             paths.add(path);
53             return;
54         }
55         visited[x][y]=true;
56         if (isSafe(mat, x+1, y, n, visited)) {
57             backtrack(mat, x+1, y, n, path+"D", paths, visited);
58         }
59         if (isSafe(mat, x, y-1, n, visited)) {
60             backtrack(mat, x, y-1, n, path+"L", paths, visited);
61         }
62         if (isSafe(mat, x, y+1, n, visited)) {
63             backtrack(mat, x, y+1, n, path+"R", paths, visited);
64         }
65         if (isSafe(mat, x-1, y, n, visited)) {
66             backtrack(mat, x-1, y, n, path+"U", paths, visited);
67         }
68         visited[x][y] = false;
69     }
70
71     private boolean isSafe(int[][] mat, int x, int y, int n, boolean[][] visited) {
72         return x>=0 && x<n && y>=0 && y<n && mat[x][y]!=1 && !visited[x][y];
73     }
74 }
```

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

Space Complexity: $O(n^2)$