DSA PRACTICE – DAY 3

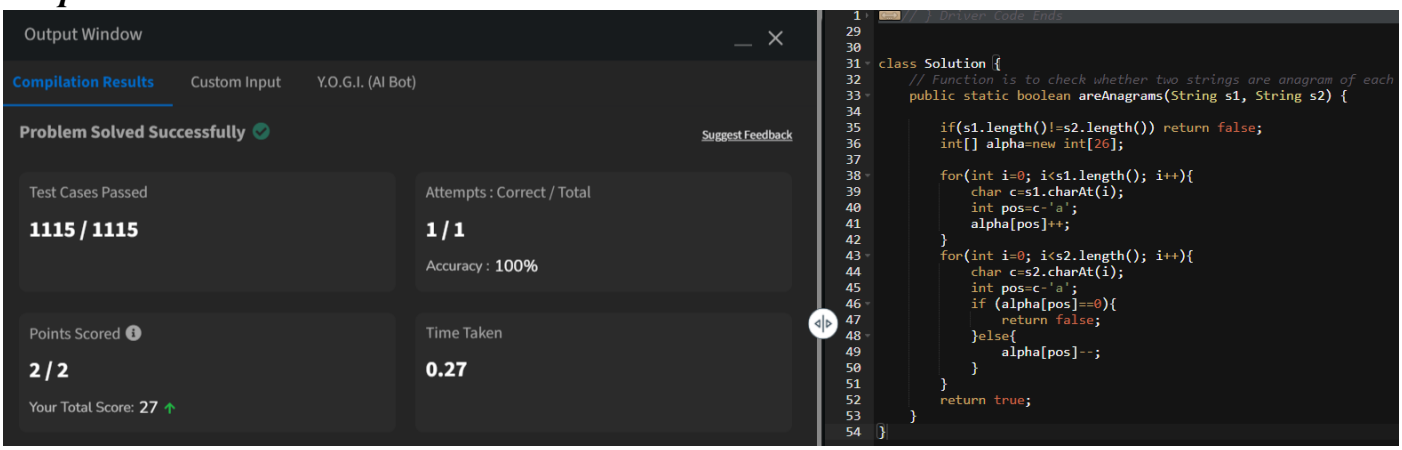
Name: Dhejan R
Reg No: 22IT022
Date: 12/11/2024

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:



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>
```

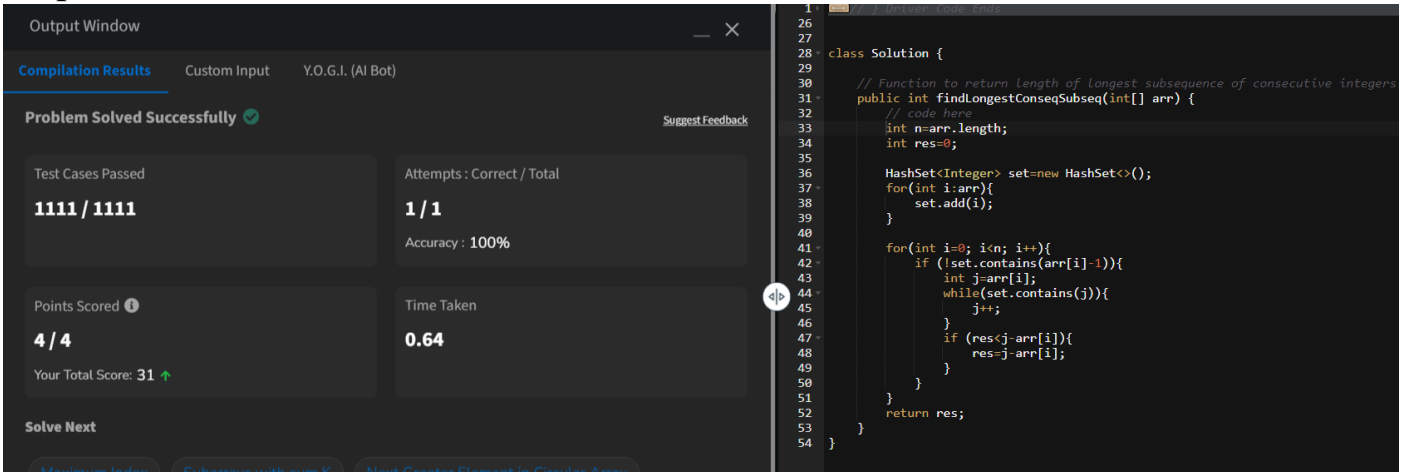
Time Complexity: O (n+m)
Space Complexity: O (1)

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:



Time Complexity: O (n)

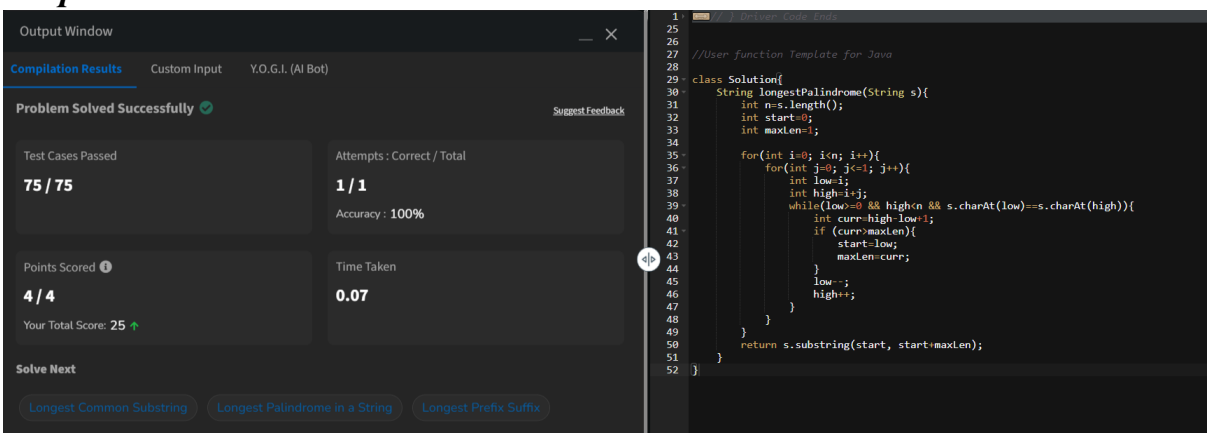
Space Complexity: O (n)

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:



Time Complexity: O (n^2)

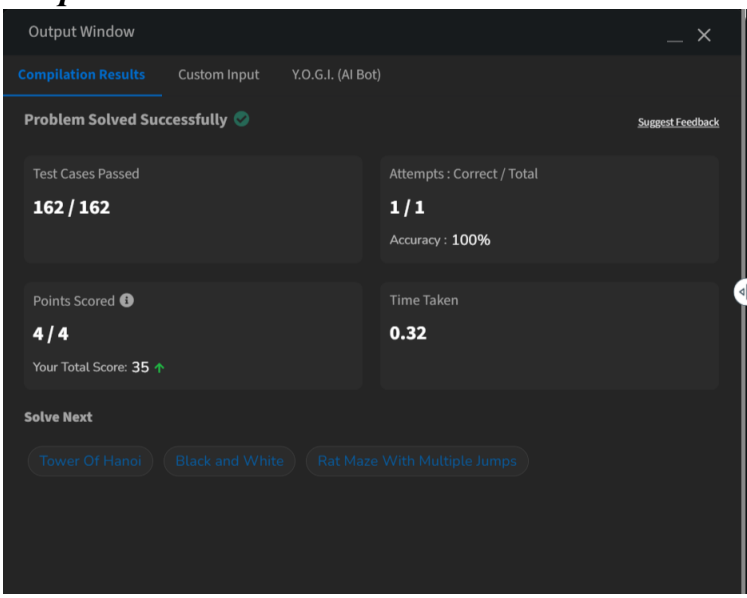
Space Complexity: O (1)

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:




```java
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<Str
        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];
    }
}
```

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