

Date: 12/11/24

DSA Practice Problems

1. Anagram

```
class Solution {  
    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']++;  
        }  
        for(int i=0;i<s2.length();i++){  
            count[s2.charAt(i)-'a']--;  
        }  
        for(int i=0;i<26;i++){  
            if(count[i]!=0){  
                return false;  
            }  
        }  
        return true;  
    }  
}
```

The screenshot shows a LeetCode problem solution for 'Anagrams'. The left panel displays 'Compilation Results' with 'Test Cases Passed: 1115 / 1115', 'Attempts: Correct / Total: 2 / 2', 'Accuracy: 100%', and 'Time Taken: 0.38'. The right panel shows the Java code for the 'areAnagrams' function, which uses character counting to determine if two strings are anagrams.

```

1  // User function template for Java
2
3  class Solution {
4      // Function to check whether two strings are anagram of each other or not.
5      public static boolean areAnagrams(String s1, String s2) {
6
7          // Your code here
8          if(s1.length()!=s2.length()){
9              return false;
10         }
11         int[] count=new int[26];
12         for(int i=0;i<s1.length();i++){
13             count[s1.charAt(i)-'a']++;
14         }
15         for(int i=0;i<s2.length();i++){
16             count[s2.charAt(i)-'a']--;
17         }
18         for(int i=0;i<26;i++){
19             if(count[i]!=0){
20                 return false;
21             }
22         }
23         return true;
24     }
25 }

```

Time Complexity: $O(n)$

2. Row with max 1s'

```

class Solution {
    public int rowWithMax1s(int arr[][]) {
        int maxrow=-1;
        int row=arr.length;
        int column=arr[0].length;
        int i=0;
        int j=column-1;
        while(i<row && j >=0){
            if(arr[i][j]==0){
                i++;
            }
            else{
                maxrow=i;
                j--;
            }
        }
        return maxrow;
    }
}

```

The screenshot shows a LeetCode problem solution for 'Row with max 1s'. The left panel displays 'Compilation Results' with 'Test Cases Passed: 85 / 85', 'Attempts: Correct / Total: 2 / 8', 'Accuracy: 25%', and 'Time Taken: 0.84'. The right panel shows the Java code for the 'rowWithMax1s' function, which iterates through the matrix to find the row with the maximum number of 1s.

```

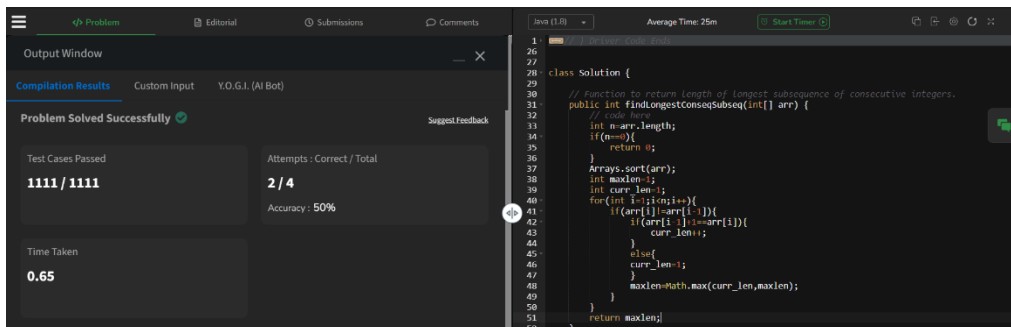
1  // User function template for Java
2
3  class Solution {
4      public int rowWithMax1s(int arr[][]) {
5
6          int maxrow=-1;
7          int row=arr.length;
8          int column=arr[0].length;
9          int i=0;
10         int j=column-1;
11         while(i<row && j >=0){
12             if(arr[i][j]==0){
13                 i++;
14             }
15             else{
16                 maxrow=i;
17                 j--;
18             }
19         }
20         return maxrow;
21     }
22 }

```

Time Complexity: $O(M+N)$

3. Longest consecutive subsequence

```
class Solution {  
    public int findLongestConseqSubseq(int[] arr) {  
        int n=arr.length;  
        if(n==0){  
            return 0;  
        }  
        Arrays.sort(arr);  
        int maxlen=1;  
        int curr_len=1;  
        for(int i=1;i<n;i++){  
            if(arr[i]!=arr[i-1]){  
                if(arr[i-1]+1==arr[i]){  
                    curr_len++;  
                }  
                else{  
                    curr_len=1;  
                }  
                maxlen=Math.max(curr_len,maxlen);  
            }  
        }  
        return maxlen;  
    }  
}
```



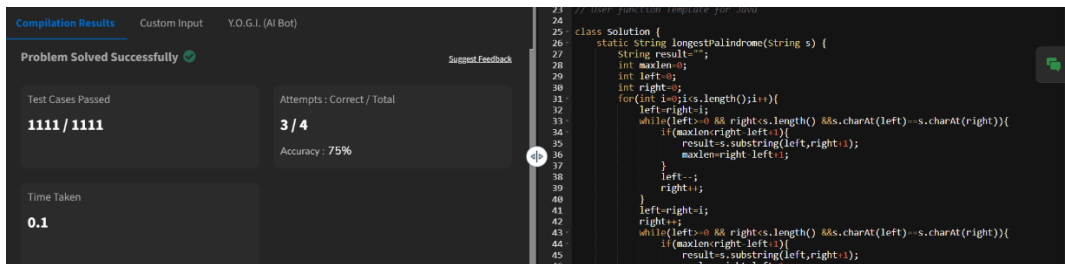
```
1  
26  
27  
28 class Solution {  
29  
30     // Function to return Length of longest subsequence of consecutive integers.  
31     public int findLongestConseqSubseq(int[] arr) {  
32         // Base Case  
33         int n=arr.length;  
34         if(n==0){  
35             return 0;  
36         }  
37         Arrays.sort(arr);  
38         int maxlen=1;  
39         int curr_len=1;  
40         for(int i=1;i<n;i++){  
41             if(arr[i]!=arr[i-1]){  
42                 if(arr[i-1]+1==arr[i]){  
43                     curr_len++;  
44                 }  
45                 else{  
46                     curr_len=1;  
47                 }  
48                 maxlen=Math.max(curr_len,maxlen);  
49             }  
50         }  
51         return maxlen;  
52     }  
53 }
```

Time Complexity: $O(n \log n)$

4. Longest palindrome in a string

```
class Solution {  
    static String longestPalindrome(String s) {  
        String result="";  
        int maxlen=0;  
        int left=0;  
        int right=0;  
        for(int i=0;i<s.length();i++){  
            left=right=i;  
            while(left>=0 && right<s.length() &&s.charAt(left)==s.charAt(right)){  
                if(maxlen<right-left+1){  
                    result=s.substring(left,right+1);  
                    maxlen=right-left+1;  
                }  
                left--;  
                right++;  
            }  
            left=right=i;  
            right++;  
            while(left>=0 && right<s.length() &&s.charAt(left)==s.charAt(right)){  
                if(maxlen<right-left+1){  
                    result=s.substring(left,right+1);  
                    maxlen=right-left+1;  
                }  
                left--;  
                right++;  
            }  
        }  
        return result;  
    }  
}
```

}



The screenshot shows a code editor with a dark theme. On the left, a 'Compilation Results' panel displays 'Problem Solved Successfully' with a green checkmark. Below this, it shows 'Test Cases Passed: 1111 / 1111', 'Attempts: Correct / Total: 3 / 4', 'Accuracy: 75%', and 'Time Taken: 0.1'. On the right, a Java code file is open, showing a class 'Solution' with a static method 'longestPalindrome'. The code uses a recursive approach with a for loop over the string length, a while loop to expand from a center, and a return statement to build the result string. Line numbers 23 through 46 are visible on the left margin of the code editor.

Time Complexity: $O(n^2)$

5. Rat in a maze problem

```
class Solution {  
  
    public ArrayList<String> findPath(int[][] mat) {  
        int N = mat.length;  
  
        if (mat[0][0] == 0 || mat[N - 1][N - 1] == 0) {  
            return new ArrayList<>();  
        }  
  
        Map<String, int[]> dirs = new HashMap<>();  
        dirs.put("U", new int[]{-1, 0});  
        dirs.put("R", new int[]{0, 1});  
        dirs.put("L", new int[]{0, -1});  
        dirs.put("D", new int[]{1, 0});  
  
        boolean[][] visited = new boolean[N][N];  
        ArrayList<String> paths = new ArrayList<>();  
  
        DFS(0, 0, "", mat, visited, paths, dirs, N);  
  
        Collections.sort(paths);  
  
        return paths;  
    }  
}
```

```

private void DFS(int r, int c, String curr, int[][] mat, boolean[][] visited,
ArrayList<String> paths, Map<String, int[]> dirs, int N) {

    if (r == N - 1 && c == N - 1) {

        paths.add(curr);

        return;

    }

    visited[r][c] = true;

    for (Map.Entry<String, int[]> entry : dirs.entrySet()) {

        int nr = r + entry.getValue()[0];

        int nc = c + entry.getValue()[1];

        if (isValid(nr, nc, mat, visited, N)) {

            DFS(nr, nc, curr + entry.getKey(), mat, visited, paths, dirs, N);

        }

    }

    visited[r][c] = false;

}

private boolean isValid(int r, int c, int[][] mat, boolean[][] visited, int N) {

    return r >= 0 && r < N && c >= 0 && c < N && mat[r][c] == 1 &&
!visited[r][c];

}

}

```

The screenshot shows a code editor interface. On the left, the 'Output Window' is open, displaying 'Compilation Results'. It indicates that the problem was solved successfully, with 162 out of 162 test cases passed. The user has used 1 out of 4 attempts, achieved 100% accuracy, and scored 4 out of 4 points. The total score is 34. The main editor area shows a Java solution for a problem named 'Path in a Grid'. The code includes a DFS method and an isValid method. The DFS method explores all four directions from a given cell (r, c) and adds the current path to the list of paths if it reaches the bottom-right corner (N-1, N-1). The isValid method checks if the coordinates (r, c) are within the grid boundaries and if the cell contains a 1 and has not been visited.