Date: 12-11-2024

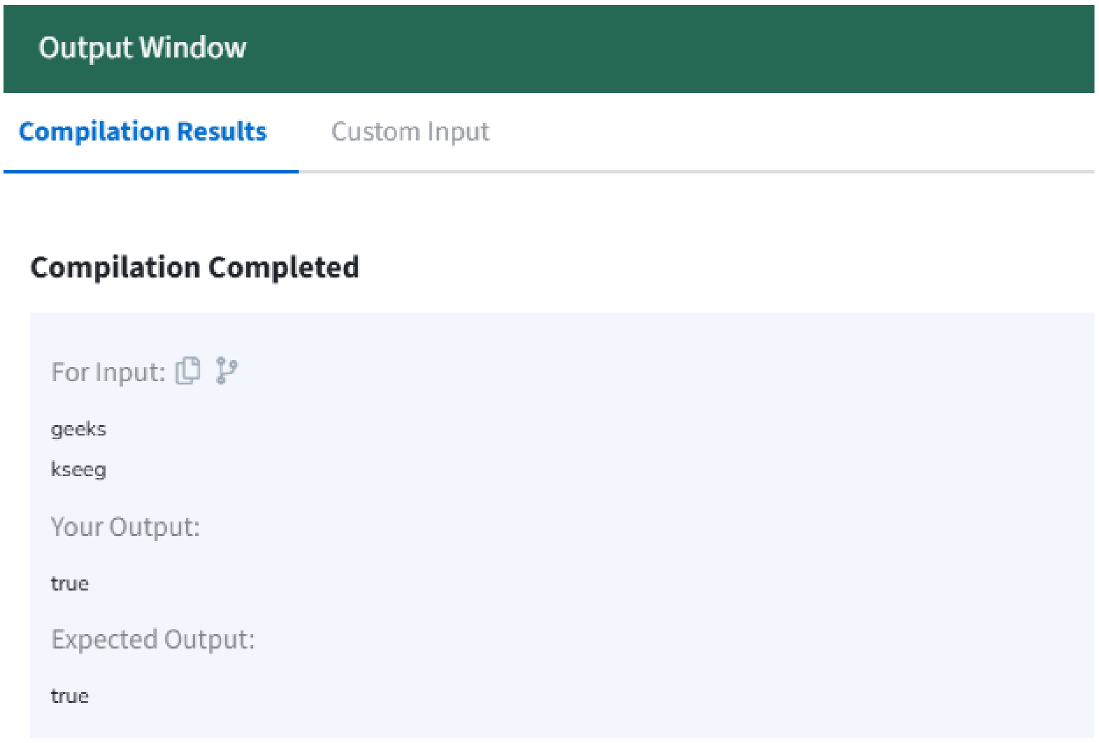
CODING PRACTICE PROBLEMS – DAY 3

1. Anagram

```java
class Solution {
    public static boolean areAnagrams(String s1, String s2) {
        if (s1.length() != s2.length()) {
            return false;
        }
        int[] charCount = new int[26];

        for (int i = 0; i < s1.length(); i++) {
            charCount[s1.charAt(i) - 'a']++;
            charCount[s2.charAt(i) - 'a']--;
        }
        for (int count : charCount) {
            if (count != 0) {
                return false;
            }
        }
        return true;
    }
}
```

Time Complexity: O(N)

OUTPUT:



Compilation Completed

For Input:

```
geeks
kseeg
```

Your Output:

```
true
```

Expected Output:

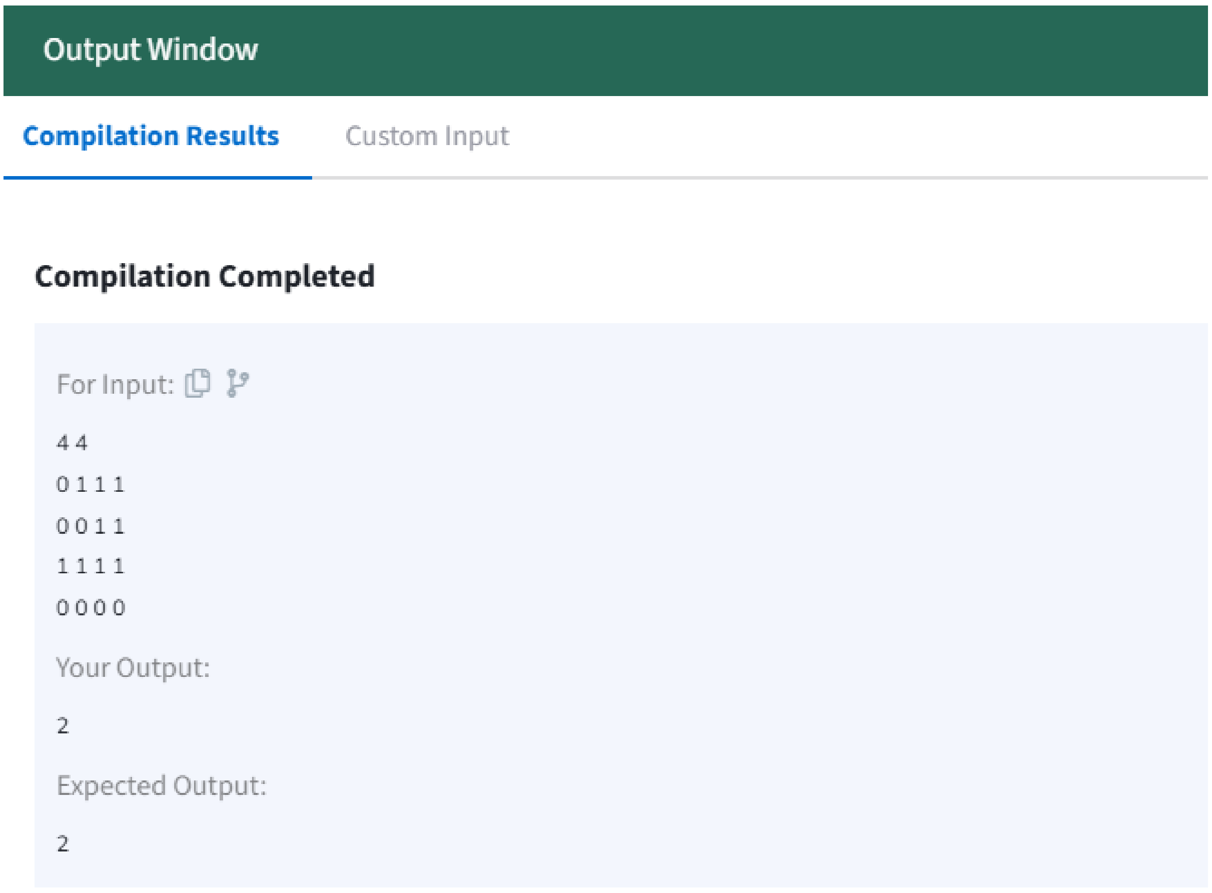
```
true
```

2. Row with max 1s

```java
class Solution {
    public int rowWithMax1s(int arr[][]) {
        int n = arr.length;
        int m = arr[0].length;
        int maxRowIndex = -1;
        int maxCount = 0;

        int row = 0, col = m - 1;

        while (row < n && col >= 0) {
            if (arr[row][col] == 1) {
                col--;
                maxRowIndex = row;
            } else {
                row++;
            }
        }

        return maxRowIndex;
    }
}
```

Time Complexity: O(N+M)

OUTPUT:



Output Window

Compilation Results   Custom Input

Compilation Completed

For Input:

```
4 4
0 1 1 1
0 0 1 1
1 1 1 1
0 0 0 0
```

Your Output:

```
2
```

Expected Output:

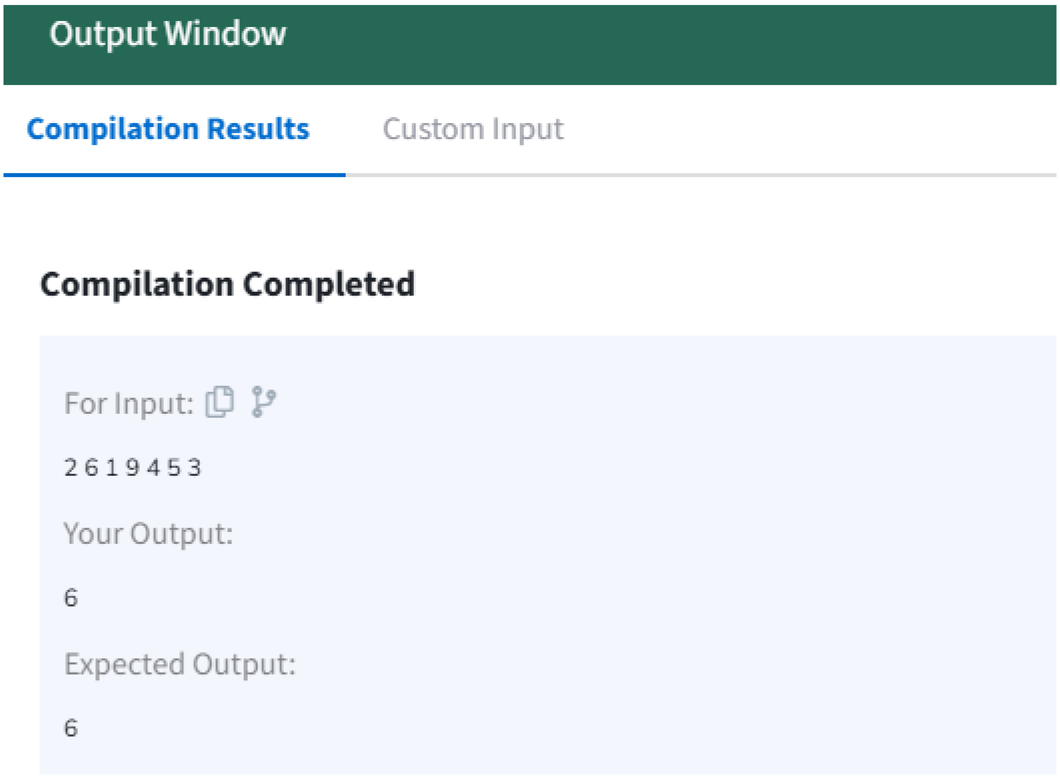
```
2
```

3. Longest consecutive subsequence

```java
class Solution {
    public int findLongestConseqSubseq(int[] arr) {
        Set<Integer> set = new HashSet<>();
        for (int num : arr) {
            set.add(num);
        }
        int longestStreak = 0;

        for (int num : arr) {
            if (!set.contains(num - 1)) {
                int currentNum = num;
                int currentStreak = 1;
                while (set.contains(currentNum + 1)) {
                    currentNum += 1;
                    currentStreak += 1;
                }
                longestStreak = Math.max(longestStreak, currentStreak);
            }
        }
        return longestStreak;
    }
}
```

Time Complexity: O(N)

OUTPUT:



Output Window

Compilation Results   Custom Input

Compilation Completed

For Input:

```
2 6 1 9 4 5 3
```

Your Output:

```
6
```

Expected Output:

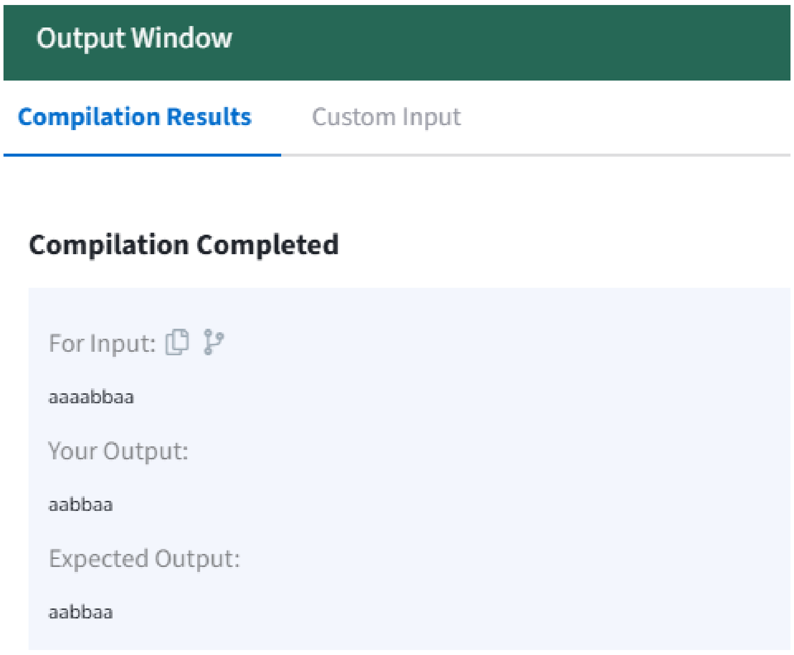
```
6
```

4. Longest palindrome in a string

```java
class Solution {
    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;
    }
}
```

Time Complexity: O(N^2)

OUTPUT:



Compilation Completed

For Input:

```
aaaabbaa
```

Your Output:

```
aabbaa
```

Expected Output:

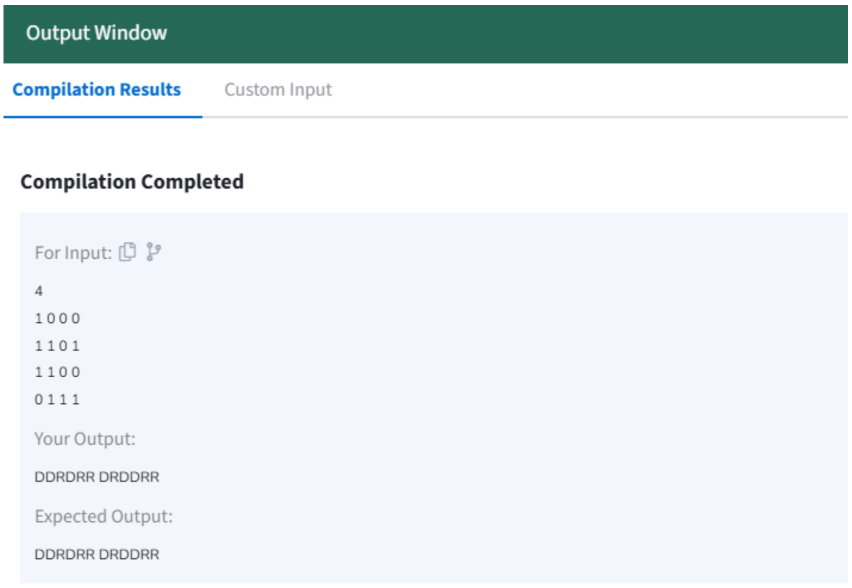
```
aabbaa
```

5. Rat in a maze problem

```java
class Solution {
    public ArrayList<String> findPath(int[][] mat) {
        ArrayList<String> paths = new ArrayList<>();
        int n = mat.length;
        if (mat[0][0] == 0) return paths;
        boolean[][] visited = new boolean[n][n];
        dfs(mat, 0, 0, n, "", visited, paths);
        Collections.sort(paths);
        return paths;
    }
    private void dfs(int[][] mat, int i, int j, int n, String path,
boolean[][] visited, ArrayList<String> paths) {
        if (i == n - 1 && j == n - 1) {
            paths.add(path);
            return;
        }
        visited[i][j] = true;
        int[] row = {1, 0, 0, -1};
        int[] col = {0, -1, 1, 0};
        char[] dir = {'D', 'L', 'R', 'U'};
        for (int k = 0; k < 4; k++) {
            int newRow = i + row[k];
            int newCol = j + col[k];
            if (isSafe(newRow, newCol, n, mat, visited)) {
                dfs(mat, newRow, newCol, n, path + dir[k], visited, paths);
            }
        }
        visited[i][j] = false;
    }
    private boolean isSafe(int i, int j, int n, int[][] mat, boolean[][]
visited) {
        return i >= 0 && i < n && j >= 0 && j < n && mat[i][j] == 1 &&
!visited[i][j];
    }
}
```

Time Complexity: O(N^2)

OUTPUT:



Output Window

Compilation Results Custom Input

Compilation Completed

For Input:

```
4
1 0 0 0
1 1 0 1
1 1 0 0
0 1 1 1
```

Your Output:

```
DDRDRR DRDDRR
```

Expected Output:

```
DDRDRR DRDDRR
```