DSA Practice Problems- Day 7

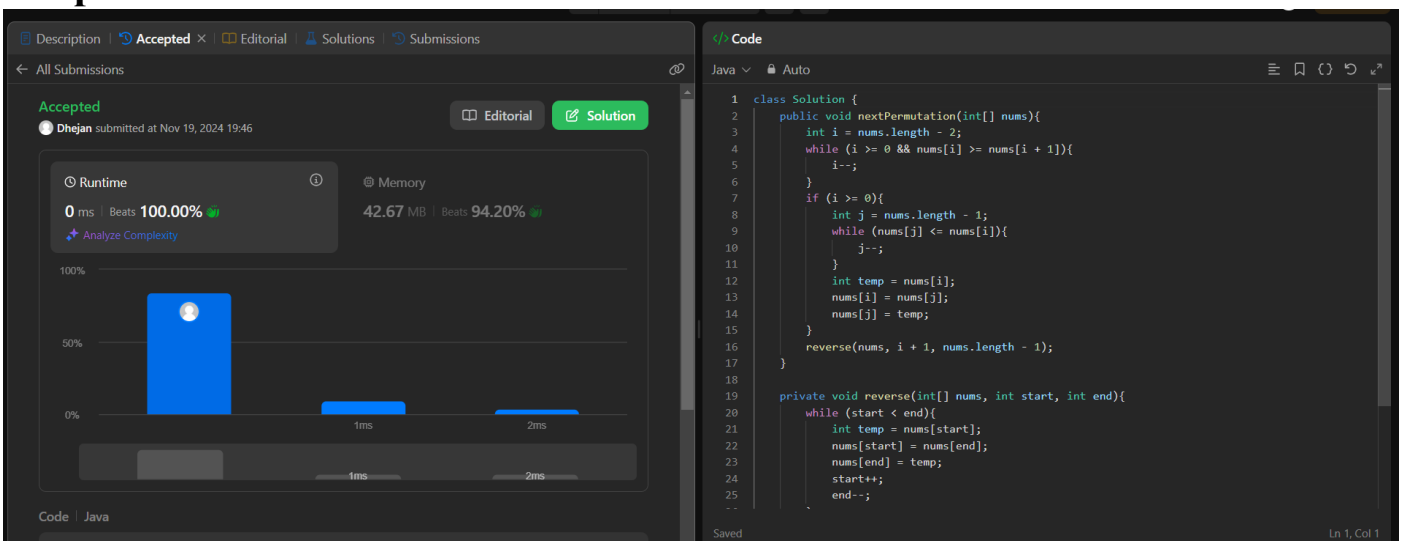
Name: Dhejan R
Reg No: 22IT022
Date: 19/11/2024

1. **Next Permutation**

Code Solution

```
class Solution {
    public void nextPermutation(int[] nums){
        int i = nums.length - 2;
        while (i >= 0 && nums[i] >= nums[i + 1]){
            i--;
        }
        if (i >= 0){
            int j = nums.length - 1;
            while (nums[j] <= nums[i]){
                j--;
            }
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
        }
        reverse(nums, i + 1, nums.length - 1);
    }

    private void reverse(int[] nums, int start, int end){
        while (start < end){
            int temp = nums[start];
            nums[start] = nums[end];
            nums[end] = temp;
            start++;
            end--;
        }
    }
}
```

Output



Time complexity: O(n)
Space complexity: O(1)

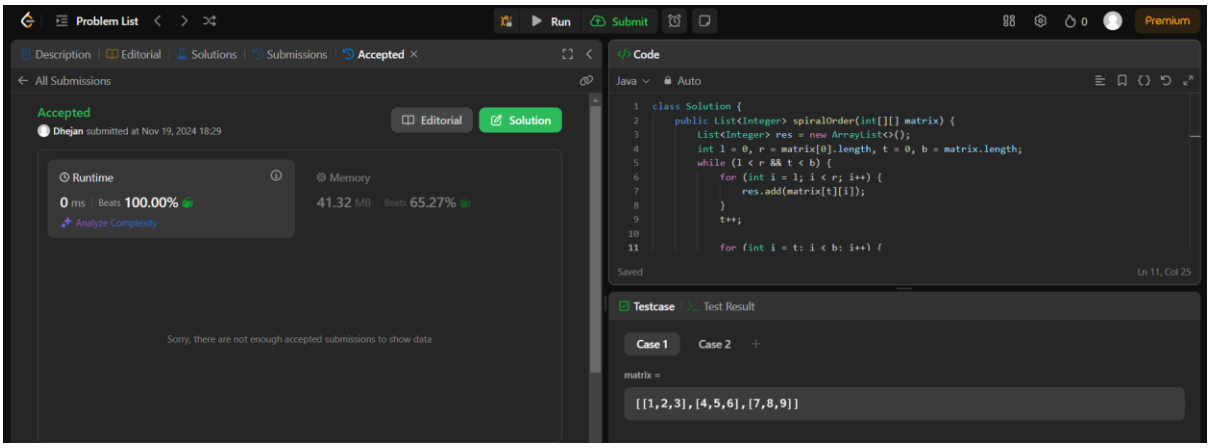
2. Spiral Matrix

Code Solution:

```
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> res = new ArrayList<>();
        int l = 0, r = matrix[0].length, t = 0, b = matrix.length;
        while (l < r && t < b) {
            for (int i = l; i < r; i++) {
                res.add(matrix[t][i]);
            }
            t++;

            for (int i = t; i < b; i++) {
                res.add(matrix[i][r - 1]);
            }
            r--;

            if (!(l < r && t < b)) {
                break;
            }

            for (int i = r - 1; i >= l; i--) {
                res.add(matrix[b - 1][i]);
            }
            b--;

            for (int i = b - 1; i >= t; i--) {
                res.add(matrix[i][l]);
            }
            l++;
        }
```

```
        return res;
    }
}
```

Output



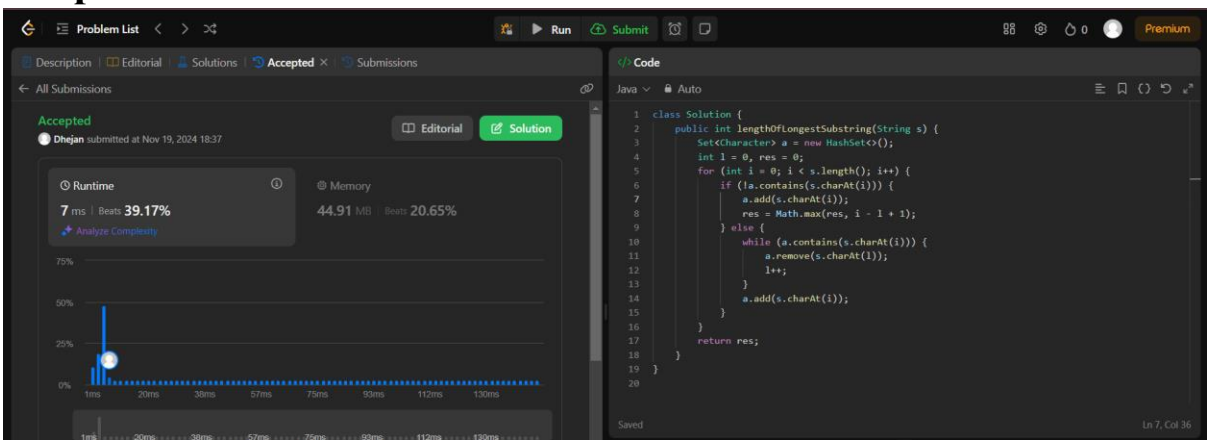
Time complexity: O(MXN)
Space complexity: O(MXN)

3. Longest substring without repeating characters

Code Solution

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        Set<Character> a = new HashSet<>();
        int l = 0, res = 0;
        for (int i = 0; i < s.length(); i++) {
            if (!a.contains(s.charAt(i))) {
                a.add(s.charAt(i));
                res = Math.max(res, i - l + 1);
            } else {
                while (a.contains(s.charAt(i))) {
                    a.remove(s.charAt(l));
                    l++;
                }
                a.add(s.charAt(i));
            }
        }
        return res;
    }
}
```

Output



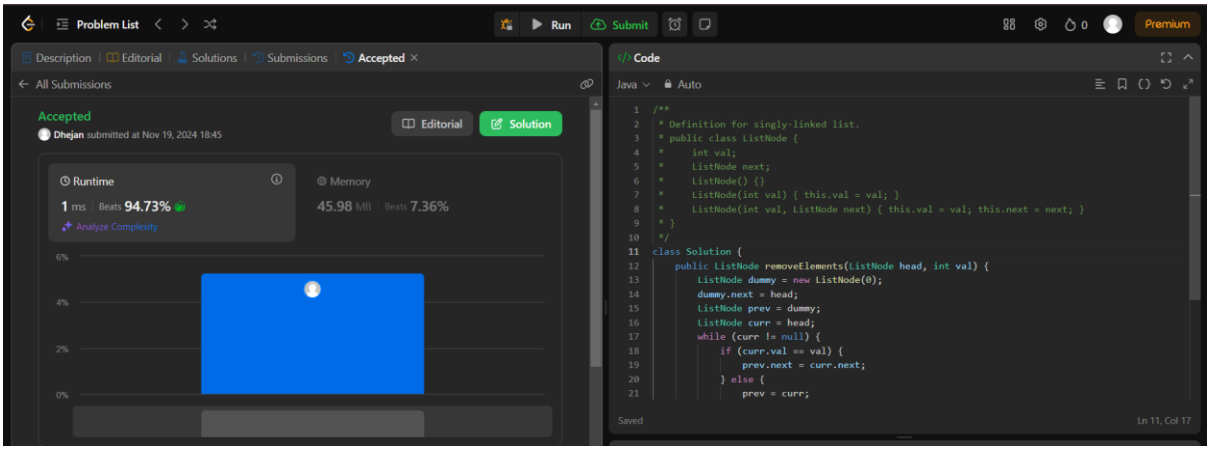
Time complexity: O(n)
Space complexity: O(n)

4. Remove linked list elements

Code solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode prev = dummy;
        ListNode curr = head;
        while (curr != null) {
            if (curr.val == val) {
                prev.next = curr.next;
            } else {
                prev = curr;
            }
            curr = curr.next;
        }
        return dummy.next;
    }
}
```

Output



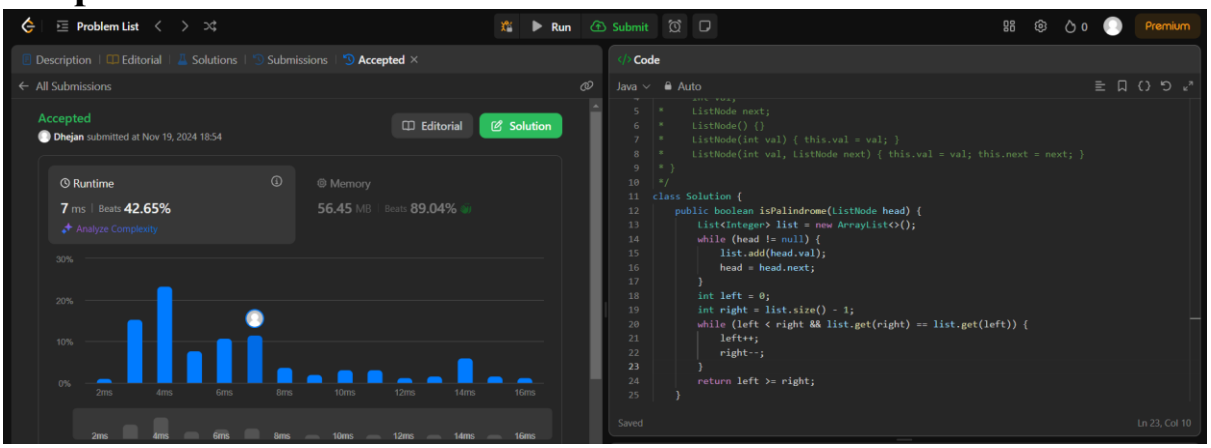
Time complexity: O(n)
Space complexity: O(1)

5. Palindrome linked list

Code Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public boolean isPalindrome(ListNode head) {
        List<Integer> list = new ArrayList<>();
        while (head != null) {
            list.add(head.val);
            head = head.next;
        }
        int left = 0;
        int right = list.size() - 1;
        while (left < right && list.get(right) == list.get(left)) {
            left++;
            right--;
        }
        return left >= right;
    }
}
```

Output

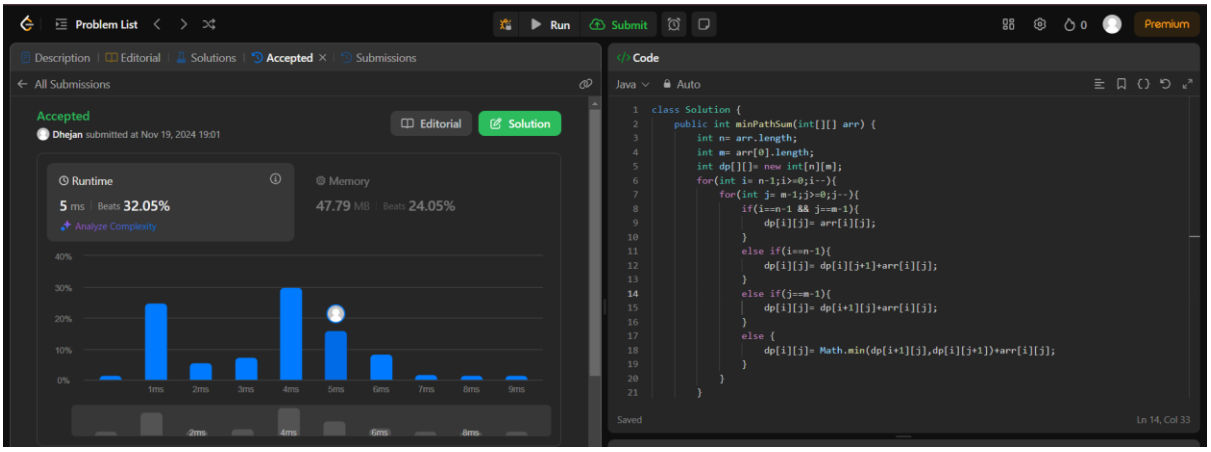


Time complexity: O(n)
Space complexity: O(n)

6. Minimum path sum

Code Solution

```
class Solution {
    public int minPathSum(int[][] arr) {
        int n= arr.length;
        int m= arr[0].length;
        int dp[][]= new int[n][m];
        for(int i= n-1;i>=0;i--){
            for(int j= m-1;j>=0;j--){
                if(i==n-1 && j==m-1){
                    dp[i][j]= arr[i][j];
                }
                else if(i==n-1){
                    dp[i][j]= dp[i][j+1]+arr[i][j];
                }
                else if(j==m-1){
                    dp[i][j]= dp[i+1][j]+arr[i][j];
                }
                else {
                    dp[i][j]= Math.min(dp[i+1][j],dp[i][j+1])+arr[i][j];
                }
            }
        }
        return dp[0][0];
    }
}
```



Time complexity: O(NXM)
Space complexity: O(NXM)

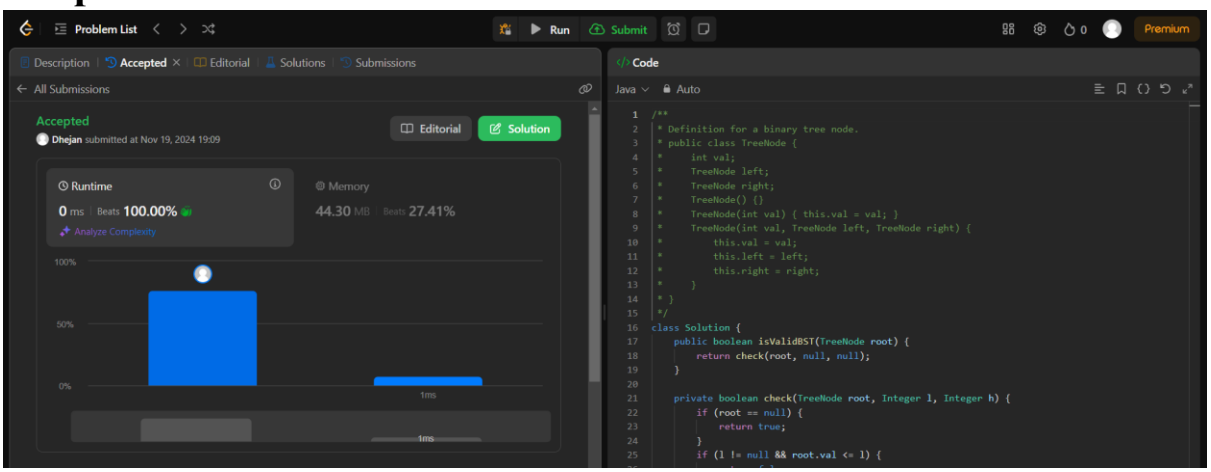
7. Validate binary search tree

Code Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isValidBST(TreeNode root) {
        return check(root, null, null);
    }

    private boolean check(TreeNode root, Integer l, Integer h) {
        if (root == null) {
            return true;
        }
        if (l != null && root.val <= l) {
            return false;
        }
        if (h != null && root.val >= h) {
            return false;
        }
        return check(root.left, l, root.val) && check(root.right, root.val, h);
```

```
    }
}
```

Output



Time complexity: O(n)
Space complexity: O(h)

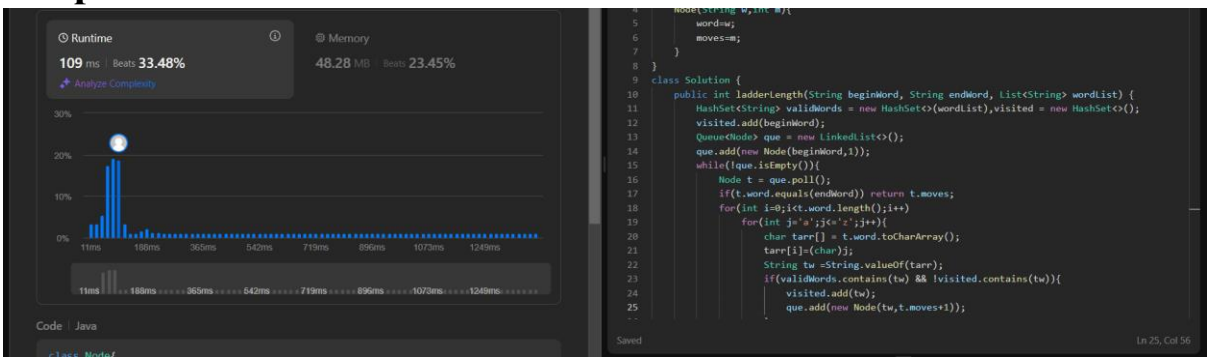
8. Word ladder

Code Solution

```
class Node{
    String word;
    int moves;
    Node(String w,int m){
        word=w;
        moves=m;
    }
}
class Solution {
    public int ladderLength(String beginWord, String endWord, List<String> wordList) {
        HashSet<String> validWords = new HashSet<>(wordList),visited = new HashSet<>();
        visited.add(beginWord);
        Queue<Node> que = new LinkedList<>();
        que.add(new Node(beginWord,1));
        while(!que.isEmpty()){
            Node t = que.poll();
            if(t.word.equals(endWord)) return t.moves;
            for(int i=0;i<t.word.length();i++)
                for(int j='a';j<='z';j++){
                    char tarr[] = t.word.toCharArray();
                    tarr[i]=(char)j;
                    String tw =String.valueOf(tarr);
                    if(validWords.contains(tw) && !visited.contains(tw)){
                        visited.add(tw);
                        que.add(new Node(tw,t.moves+1));
                    }
                }
        }return 0;
    }
```

}

Output



Time complexity: O(NXM)

Space complexity: O(N)

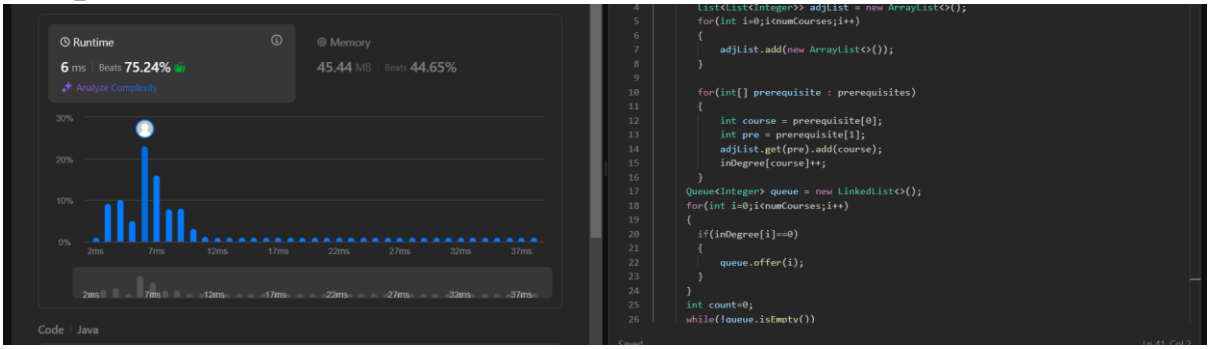
10. Course Schedule

Code Solution

```
class Solution {
   public boolean canFinish(int numCourses, int[][] prerequisites) {
      int[] inDegree = new int[numCourses];
      List<List<Integer>> adjList = new ArrayList<>();
      for(int i=0;i<numCourses;i++)
      {
         adjList.add(new ArrayList<>());
      }

      for(int[] prerequisite : prerequisites)
      {
         int course = prerequisite[0];
         int pre = prerequisite[1];
         adjList.get(pre).add(course);
         inDegree[course]++;
      }
     Queue<Integer> queue = new LinkedList<>();
     for(int i=0;i<numCourses;i++)
     {
      if(inDegree[i]==0)
      {
         queue.offer(i);
      }
     }
     int count=0;
     while(!queue.isEmpty())
     {
      int current = queue.poll();
      count++;
      for(int neighbor:adjList.get(current))
      {
         inDegree[neighbor]--;
```

```
            if(inDegree[neighbor]==0)
            {
                queue.offer(neighbor);
            }
        }
      }
      return count==numCourses;
    }
}
```

Output



Time complexity: O(V+E)
Space complexity: O(V+E)

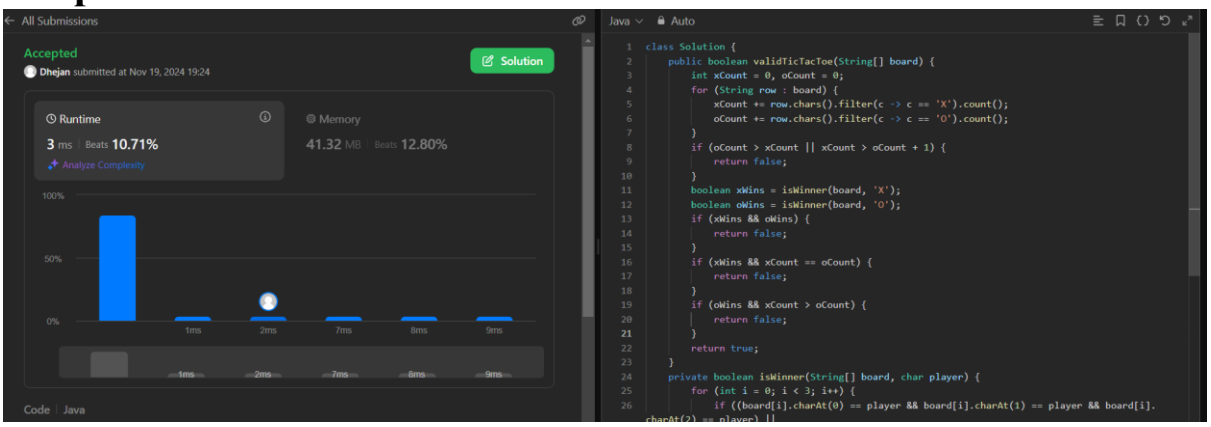
11. Valid Tic Tac Toe

Code Solution

```
class Solution {
    public boolean validTicTacToe(String[] board) {
        int xCount = 0, oCount = 0;
        for (String row : board) {
            xCount += row.chars().filter(c -> c == 'X').count();
            oCount += row.chars().filter(c -> c == 'O').count();
        }
        if (oCount > xCount || xCount > oCount + 1) {
            return false;
        }
        boolean xWins = isWinner(board, 'X');
        boolean oWins = isWinner(board, 'O');
        if (xWins && oWins) {
            return false;
        }
        if (xWins && xCount == oCount) {
            return false;
        }
        if (oWins && xCount > oCount) {
            return false;
        }
        return true;
    }
    private boolean isWinner(String[] board, char player) {
        for (int i = 0; i < 3; i++) {
```

```
        if ((board[i].charAt(0) == player && board[i].charAt(1) == player && board[i].charAt(2) == player)
||
            (board[0].charAt(i) == player && board[1].charAt(i) == player && board[2].charAt(i) == player))
{
            return true;
        }
    }
    if ((board[0].charAt(0) == player && board[1].charAt(1) == player && board[2].charAt(2) == player)
||
        (board[0].charAt(2) == player && board[1].charAt(1) == player && board[2].charAt(0) == player))
{
        return true;
    }
    return false;
  }
}
```

Output



Time complexity: O(1)
Space complexity: O(1)