

Name: Adarsh

UID: 22BCS10068

Aim: To solve leet code problems

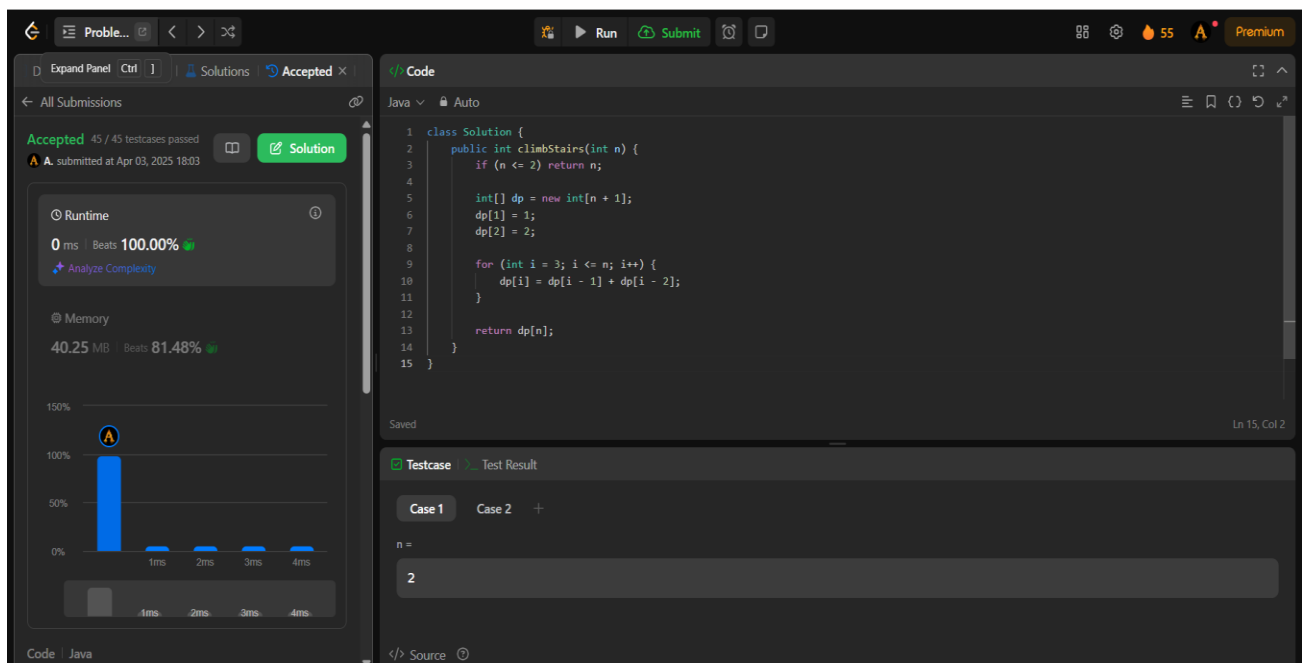
Dynamic programming(Basic problems)

1. Problem : Climbing stairs

Code:

```
class Solution {  
    public int climbStairs(int n) {  
        if (n <= 2) return n;  
  
        int[] dp = new int[n + 1];  
        dp[1] = 1;  
        dp[2] = 2;  
  
        for (int i = 3; i <= n; i++) {  
            dp[i] = dp[i - 1] + dp[i - 2];  
        }  
  
        return dp[n];  
    }  
}
```

Output :

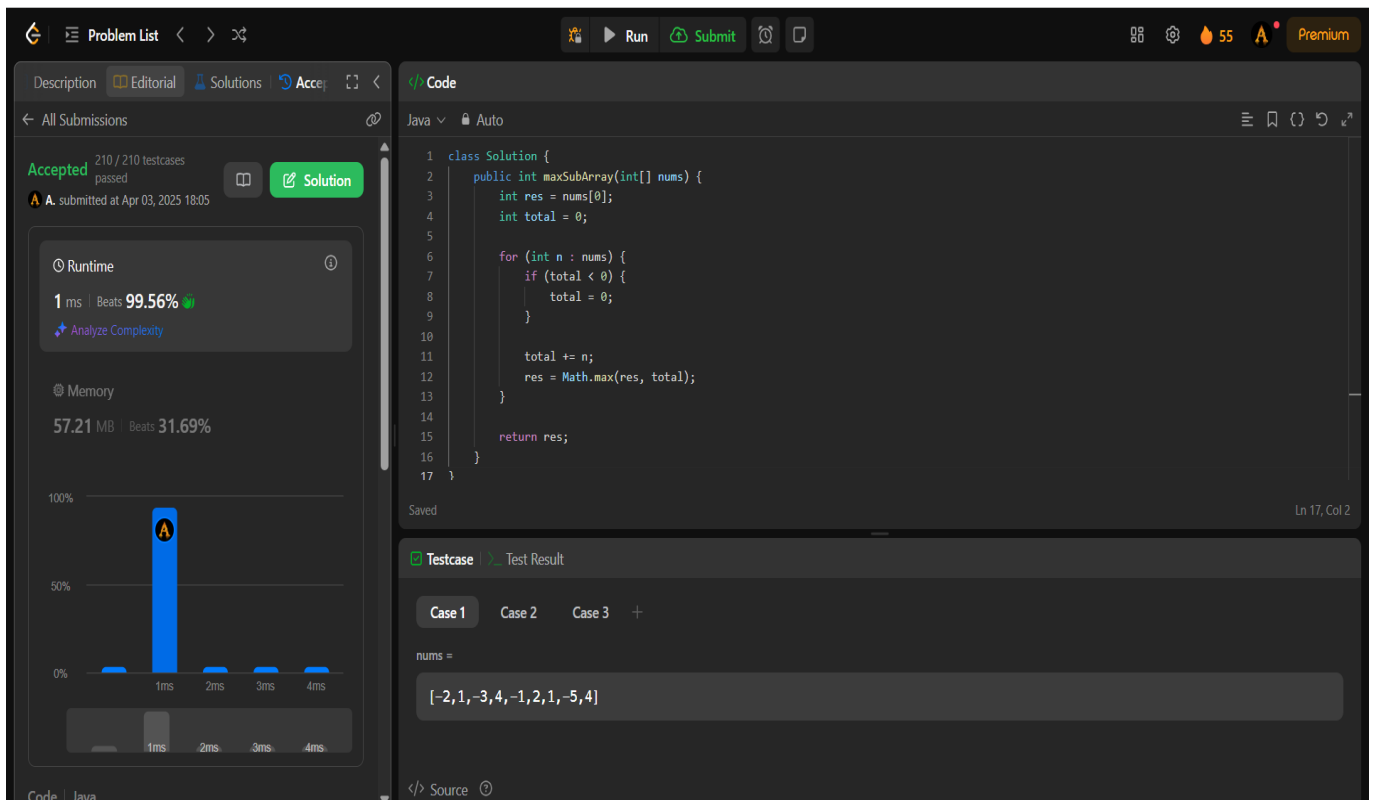


2. Problem: Maximum subarray

Code:

```
class Solution {  
    public int maxSubArray(int[] nums) {  
        int res = nums[0];  
        int total = 0;  
  
        for (int n : nums) {  
            if (total < 0) {  
                total = 0;  
            }  
  
            total += n;  
            res = Math.max(res, total);  
        }  
  
        return res;  
    }  
}
```

Output:



The screenshot displays a code editor interface for the "Maximum Subarray" problem. The left sidebar shows the problem status as "Accepted" with 210/210 testcases passed, submitted at Apr 03, 2025 18:05. The runtime is 1 ms, beating 99.56% of solutions, and the memory is 57.21 MB, beating 31.69%. A bar chart shows the runtime distribution. The main editor displays the Java code for the solution. The bottom right shows the test case input: `nums = [-2,1,-3,4,-1,2,1,-5,4]`.

Problem List < > 🔍

Description Editorial Solutions Accepted [🔍] [🔍]

All Submissions

Accepted 210 / 210 testcases passed

A. submitted at Apr 03, 2025 18:05

Runtime

1 ms | Beats: 99.56%

Analyze Complexity

Memory

57.21 MB | Beats: 31.69%

100% 50% 0%

1ms 2ms 3ms 4ms

Code | Java

Code

Java Auto

```
1 class Solution {  
2     public int maxSubArray(int[] nums) {  
3         int res = nums[0];  
4         int total = 0;  
5  
6         for (int n : nums) {  
7             if (total < 0) {  
8                 total = 0;  
9             }  
10  
11             total += n;  
12             res = Math.max(res, total);  
13         }  
14  
15         return res;  
16     }  
17 }
```

Saved

Ln 17, Col 2

Testcase Test Result

Case 1 Case 2 Case 3 +

nums =

[-2,1,-3,4,-1,2,1,-5,4]

</> Source

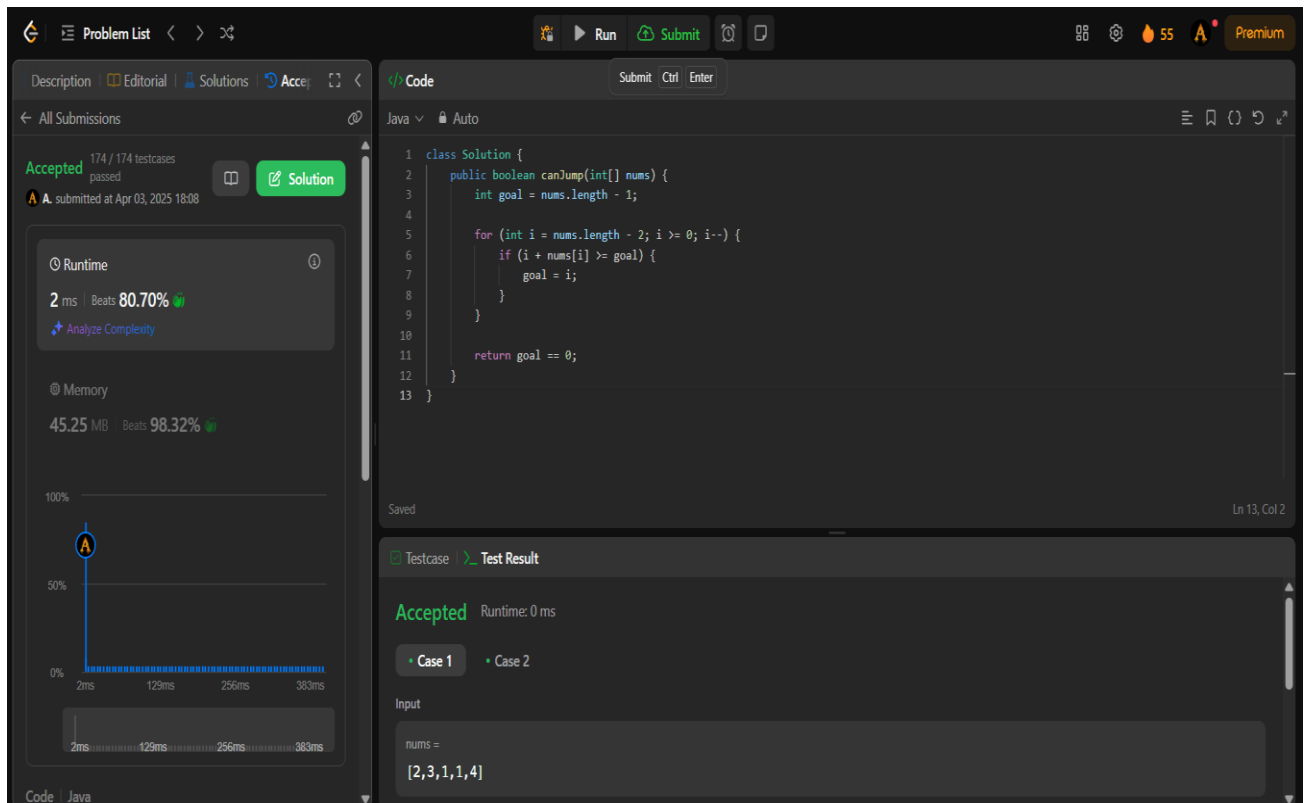
Dynamic Programming (Intermediate Problems)

3. Problem: Jump game

Code:

```
class Solution {  
    public boolean canJump(int[] nums) {  
        int goal = nums.length - 1;  
  
        for (int i = nums.length - 2; i >= 0; i--) {  
            if (i + nums[i] >= goal) {  
                goal = i;  
            }  
        }  
  
        return goal == 0;  
    }  
}
```

Output:



The screenshot displays a code editor interface for a problem titled "Jump game". The code is written in Java and is shown in the "Code" tab. The code defines a class `Solution` with a method `canJump` that takes an array of integers `nums` and returns a boolean. The method calculates the goal as the last index of the array and iterates backwards from the second-to-last index to the first index, updating the goal to the current index if the current index plus the value at that index is greater than or equal to the goal. The method returns `true` if the goal is 0, indicating that the start of the array is the goal.

The left sidebar shows the "Problem List" and "All Submissions" tabs. The "All Submissions" tab is active, showing a submission that is "Accepted" with 174/174 testcases passed. The submission was made on Apr 03, 2025 at 18:08. The runtime is 2 ms, which beats 80.70% of other submissions. The memory usage is 45.25 MB, which beats 98.32% of other submissions. A graph shows the runtime distribution, with a peak at 2 ms.

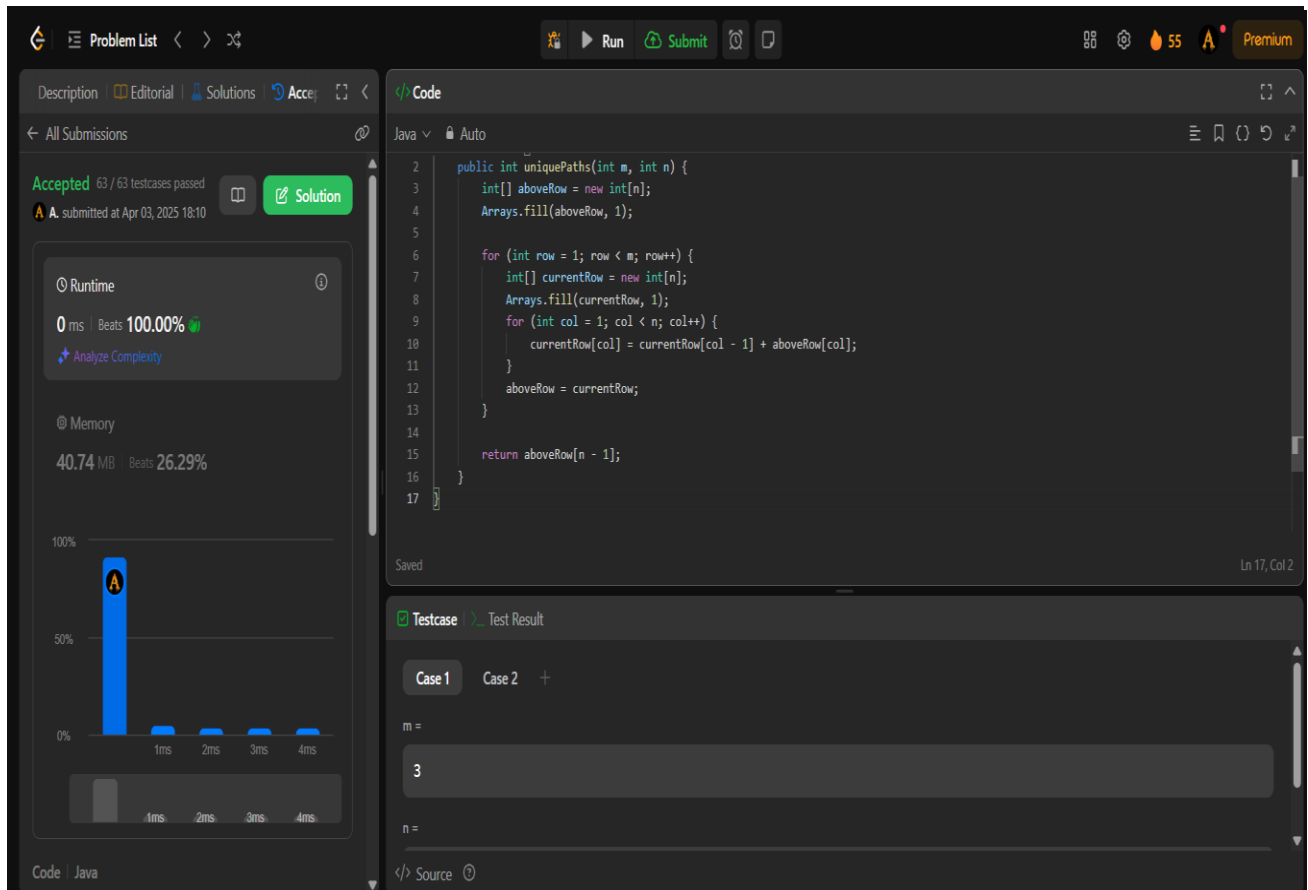
The bottom right section shows the "Testcase" and "Test Result" tabs. The "Test Result" tab is active, showing the input `nums = [2,3,1,1,4]` and the output "Accepted" with a runtime of 0 ms.

4. Problem: Unique Paths

Code:

```
class Solution {  
    public int uniquePaths(int m, int n) {  
        int[] aboveRow = new int[n];  
        Arrays.fill(aboveRow, 1);  
  
        for (int row = 1; row < m; row++) {  
            int[] currentRow = new int[n];  
            Arrays.fill(currentRow, 1);  
            for (int col = 1; col < n; col++) {  
                currentRow[col] = currentRow[col - 1] + aboveRow[col];  
            }  
            aboveRow = currentRow;  
        }  
  
        return aboveRow[n - 1];  
    }  
}
```

Output:



The screenshot displays a coding platform interface with the following components:

- Problem List:** Shows the current problem and navigation options.
- Submissions:** Indicates "Accepted 63 / 63 testcases passed" and "A. submitted at Apr 03, 2025 18:10".
- Runtime:** Shows "0 ms | Beats 100.00%".
- Memory:** Shows "40.74 MB | Beats 26.29%".
- Code Editor:** Contains the Java code for the solution, with line numbers 1 through 17.
- Testcase:** Shows "Case 1" and "Case 2" with input values $m = 3$ and $n =$.

Dynamic Programming (Advanced Problems)

5. Problem: Maximum product subarrays

Code:

```
class Solution {
    public int maxProduct(int[] nums) {
        int res = Integer.MIN_VALUE;
        for (int n : nums) {
            res = Math.max(res, n);
        }

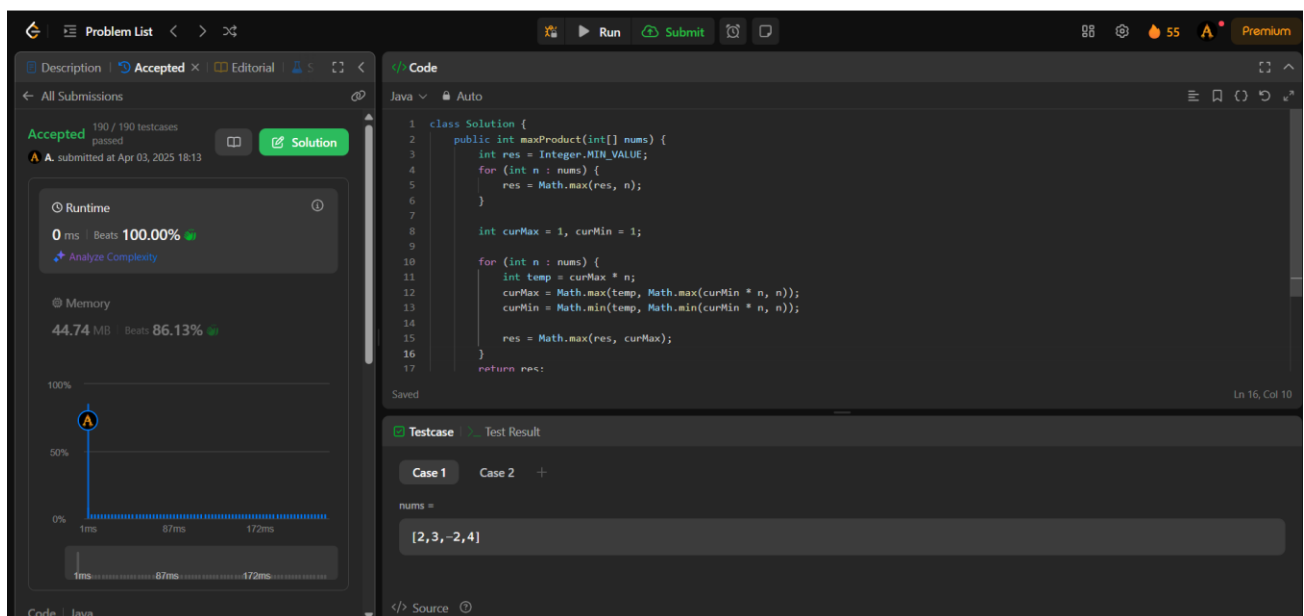
        int curMax = 1, curMin = 1;

        for (int n : nums) {
            int temp = curMax * n;
            curMax = Math.max(temp, Math.max(curMin * n, n));
            curMin = Math.min(temp, Math.min(curMin * n, n));

            res = Math.max(res, curMax);
        }

        return res;
    }
}
```

Output:



The screenshot shows a code editor interface with the following details:

- Problem List:** Accepted (190 / 190 testcases passed). Submitted at Apr 03, 2025 18:13.
- Runtime:** 0 ms, Beats 100.00%.
- Memory:** 44.74 MB, Beats 86.13%.
- Code:** Java. The code is the same as provided in the previous block.
- Testcase:** Case 1. Input: nums = [2, 3, -2, 4].

6. Problem: Decode ways

Code:

```
class Solution {
public int numDecodings(String s) {
    if (s.charAt(0) == '0') {
        return 0;
    }

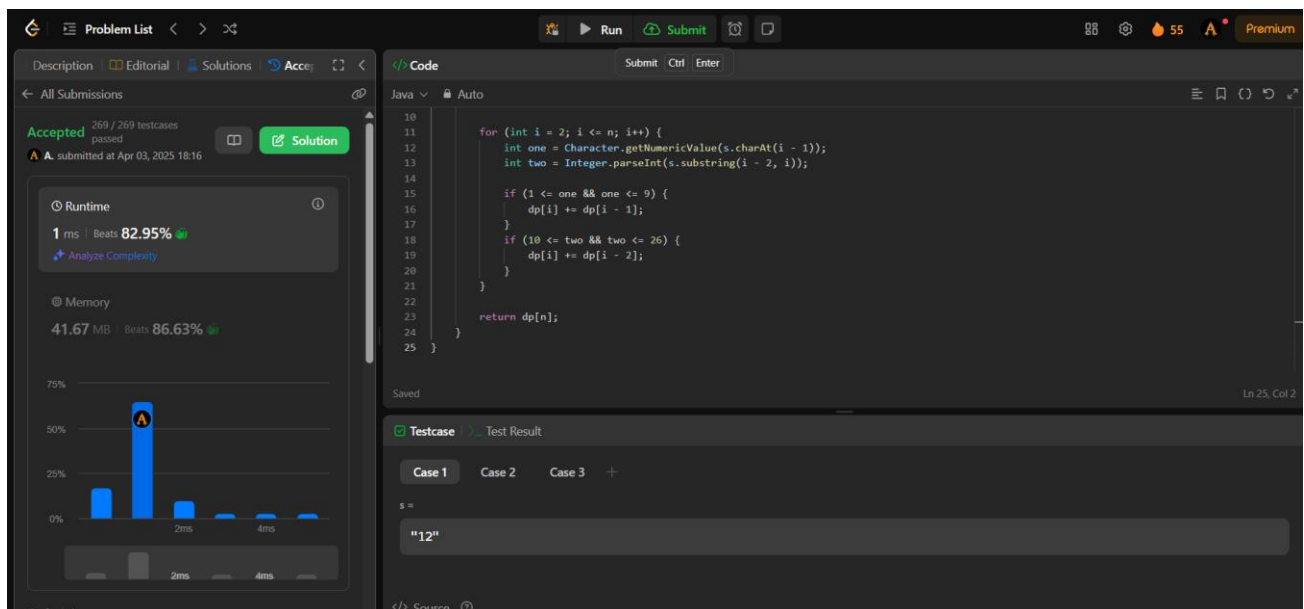
    int n = s.length();
    int[] dp = new int[n + 1];
    dp[0] = dp[1] = 1;

    for (int i = 2; i <= n; i++) {
        int one = Character.getNumericValue(s.charAt(i - 1));
        int two = Integer.parseInt(s.substring(i - 2, i));

        if (1 <= one && one <= 9) {
            dp[i] += dp[i - 1];
        }
        if (10 <= two && two <= 26) {
            dp[i] += dp[i - 2];
        }
    }

    return dp[n];
}
}
```

Output:



The screenshot displays a coding platform interface with the following components:

- Problem List:** Shows the current problem and navigation options.
- Description:** Contains the problem statement and constraints.
- Editorial:** Provides hints and explanations for the problem.
- Solutions:** Lists other users' solutions for comparison.
- Accepted:** Indicates that the solution has been accepted.
- Runtime:** Shows the execution time as 1 ms, beating 82.95% of other solutions.
- Memory:** Shows the memory usage as 41.67 MB, beating 86.63% of other solutions.
- Code:** Displays the Java code for the solution, which is a dynamic programming approach.
- Testcase:** Shows the input string "12" and the expected output.

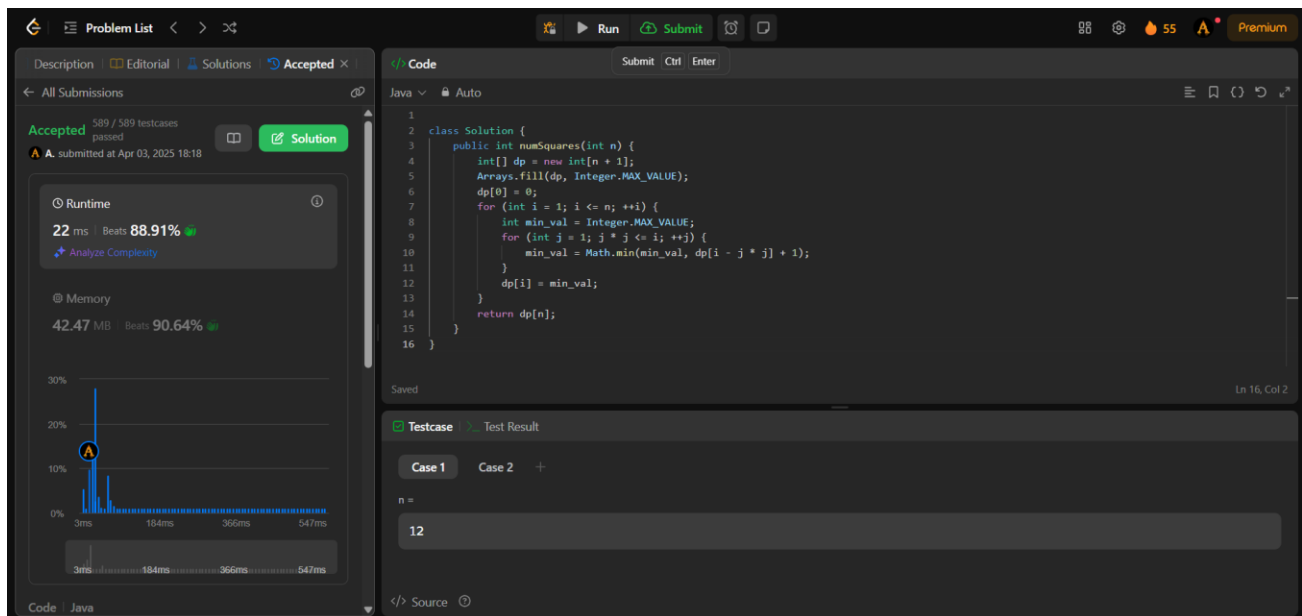
Dynamic Programming (More Challenges)

7. Problem: Perfect Squares

Code:

```
class Solution {
    public int numSquares(int n) {
        int[] dp = new int[n + 1];
        Arrays.fill(dp, Integer.MAX_VALUE);
        dp[0] = 0;
        for (int i = 1; i <= n; ++i) {
            int min_val = Integer.MAX_VALUE;
            for (int j = 1; j * j <= i; ++j) {
                min_val = Math.min(min_val, dp[i - j * j] + 1);
            }
            dp[i] = min_val;
        }
        return dp[n];
    }
}
```

Output:



The screenshot displays a code editor interface for a problem titled "Perfect Squares". The code is written in Java and implements a dynamic programming solution. The left sidebar shows the problem status as "Accepted" with 589/589 testcases passed. The runtime is 22 ms, beating 88.91% of solutions. The memory usage is 42.47 MB, beating 90.64% of solutions. The right pane shows the code for the Solution class. The bottom pane shows the test case input n=12.

```
class Solution {
    public int numSquares(int n) {
        int[] dp = new int[n + 1];
        Arrays.fill(dp, Integer.MAX_VALUE);
        dp[0] = 0;
        for (int i = 1; i <= n; ++i) {
            int min_val = Integer.MAX_VALUE;
            for (int j = 1; j * j <= i; ++j) {
                min_val = Math.min(min_val, dp[i - j * j] + 1);
            }
            dp[i] = min_val;
        }
        return dp[n];
    }
}
```

Testcase: Case 1, Case 2, +

n = 12

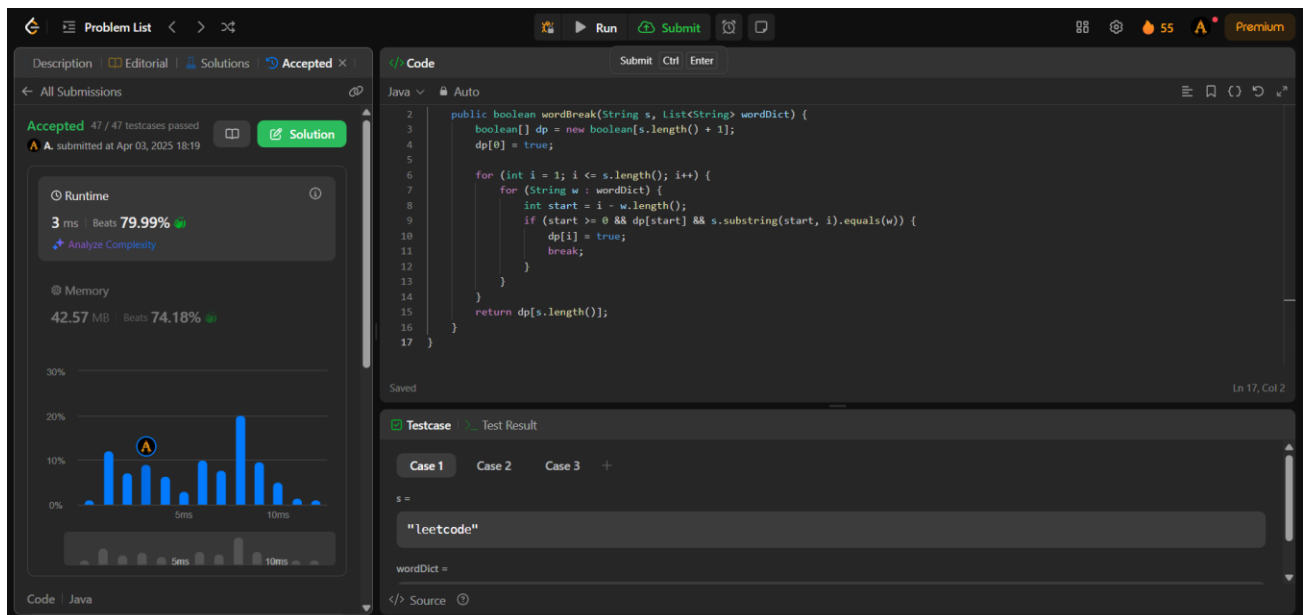
8. Problem: Word Break

Code:

```
class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
        boolean[] dp = new boolean[s.length() + 1];
        dp[0] = true;

        for (int i = 1; i <= s.length(); i++) {
            for (String w : wordDict) {
                int start = i - w.length();
                if (start >= 0 && dp[start] && s.substring(start, i).equals(w)) {
                    dp[i] = true;
                    break;
                }
            }
        }
        return dp[s.length()];
    }
}
```

Output:



The screenshot displays the LeetCode submission page for the 'Word Break' problem. The left sidebar shows the problem status as 'Accepted' with 47/47 testcases passed, submitted on Apr 03, 2025, at 18:19. The runtime is 3 ms, beating 79.99% of solutions, and the memory usage is 42.57 MB, beating 74.18%. A bar chart shows the distribution of runtime and memory usage across different percentiles. The main area shows the Java code for the solution, which is the same as the code provided in the text. The bottom section shows the test results for Case 1, where the input string 's' is 'leetcode' and the word dictionary 'wordDict' is empty.