DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

Experiment - 7

Name: Adarsh **UID: 22BCS10068**

Aim: To solve leet code problems

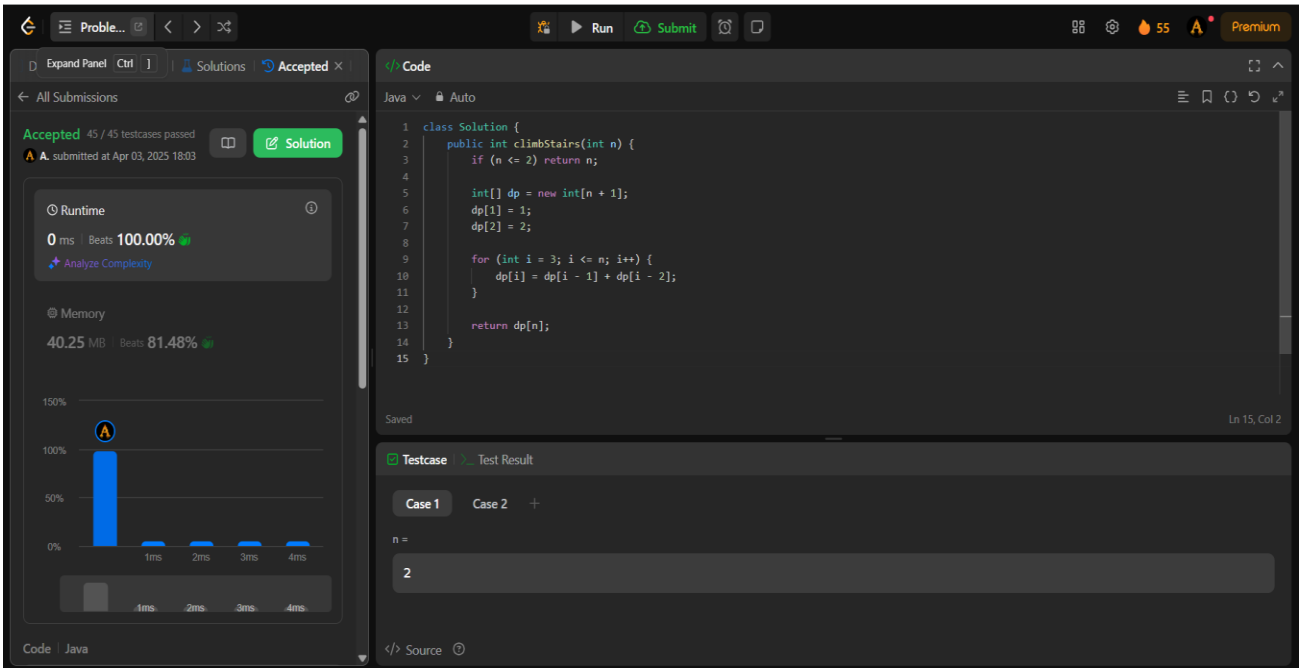
Dynamic programming(Basic problems)

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 :

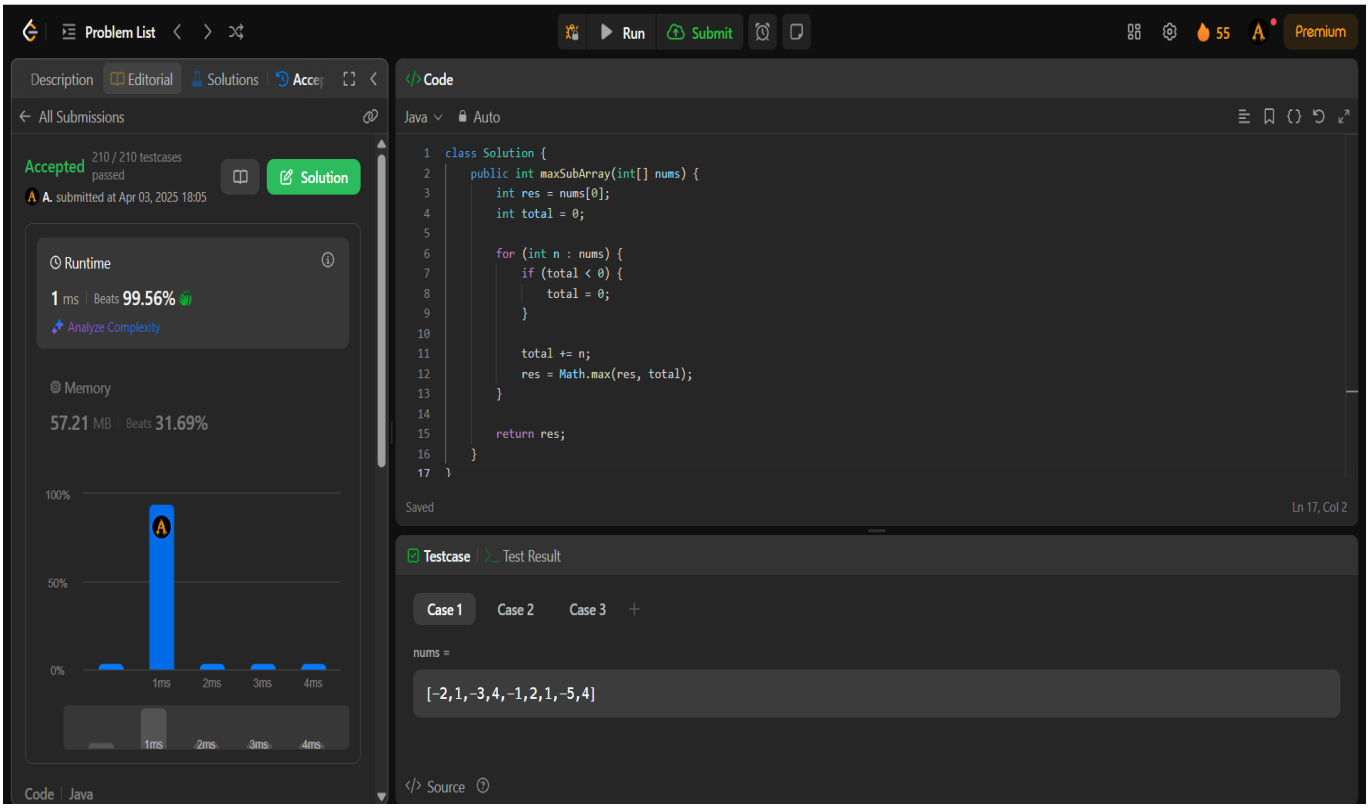


2. Problem: Maximum subarray

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:



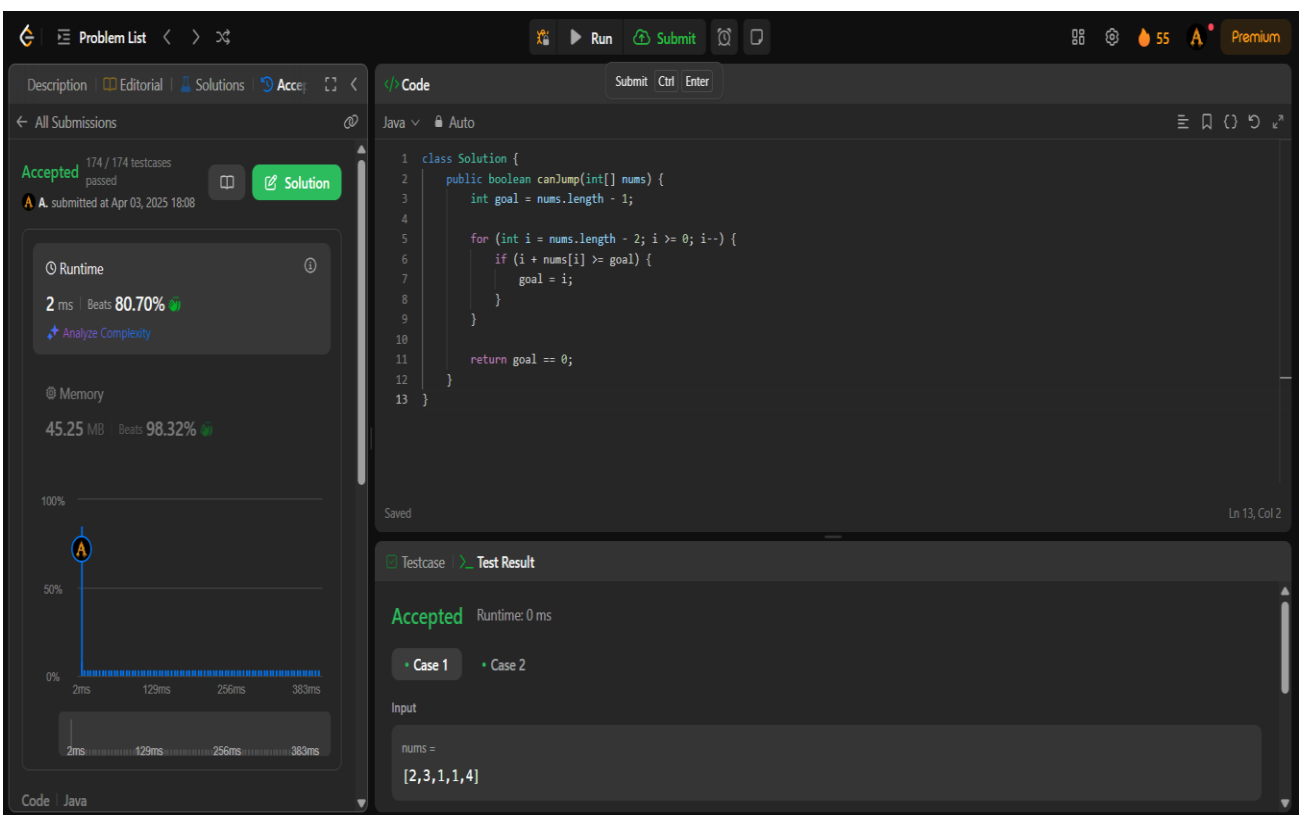
Dynamic Programming (Intermediate Problems)

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:

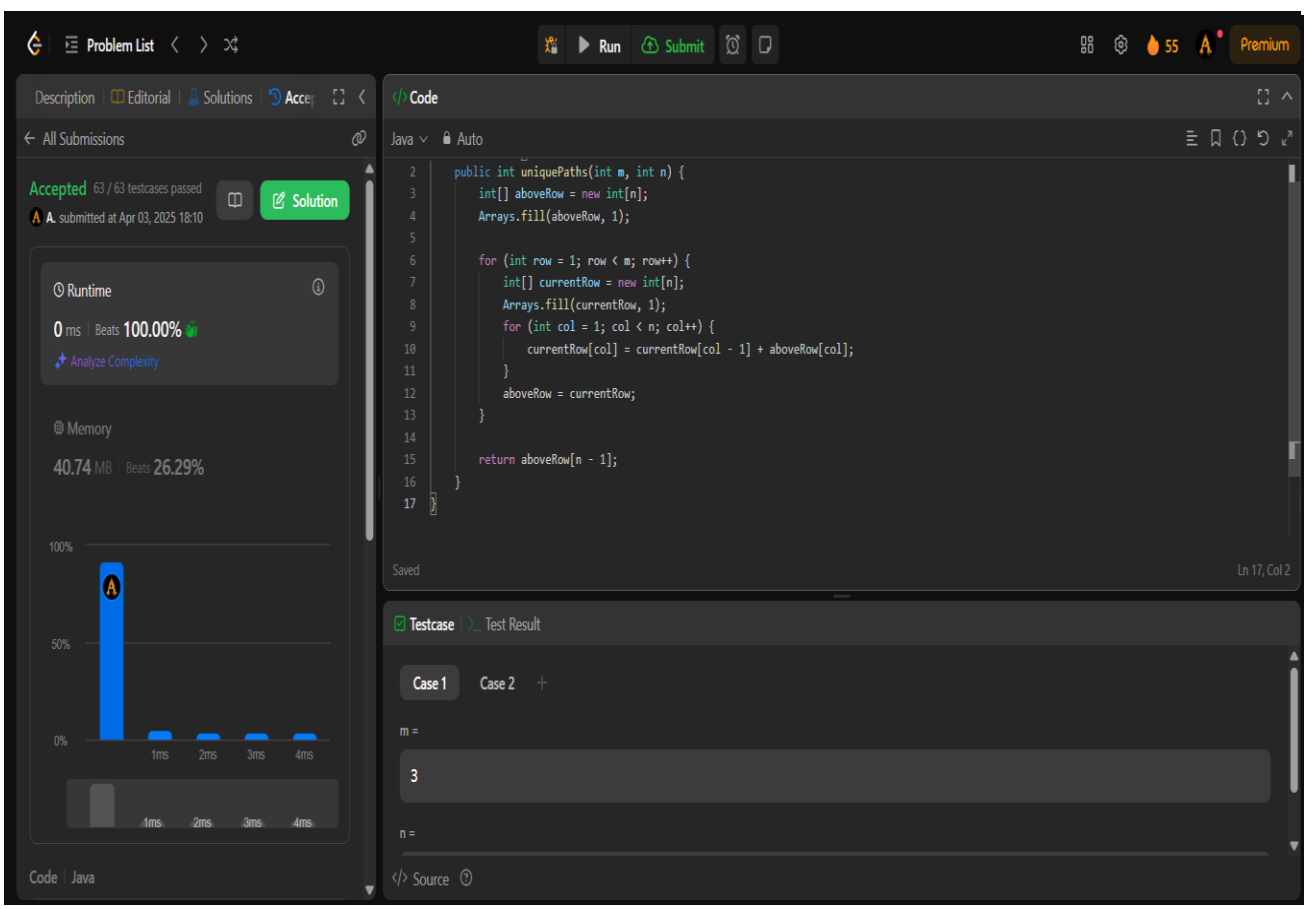


4. Problem: Unique Paths

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:



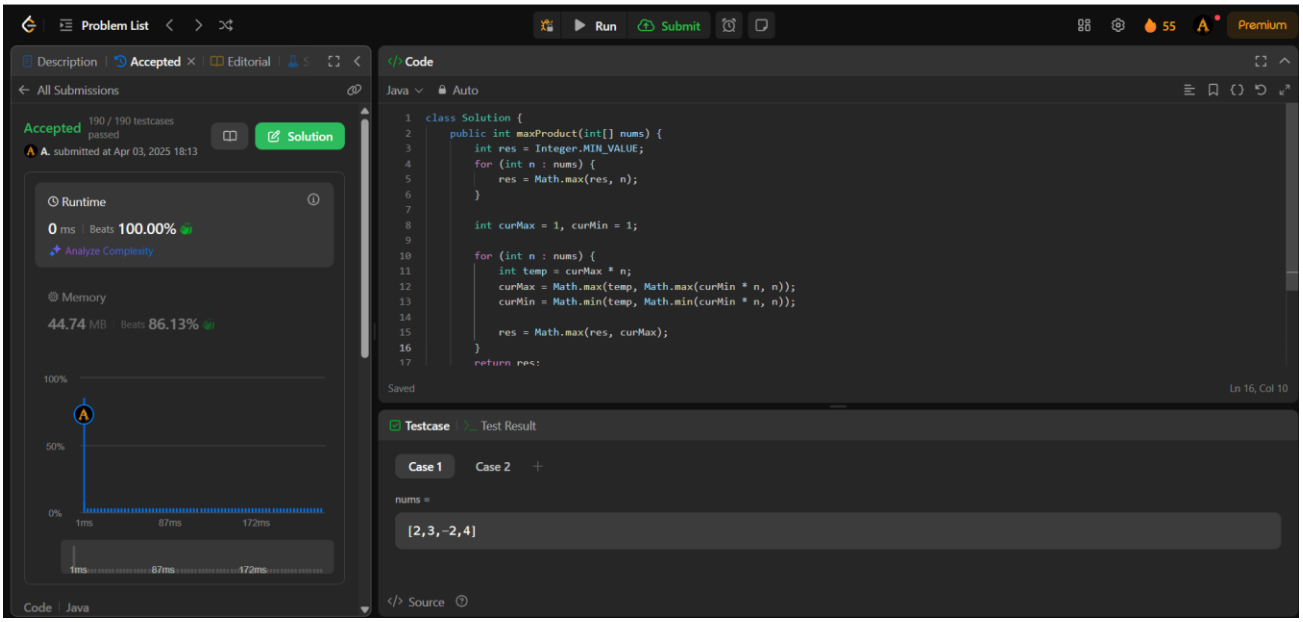
Dynamic Programming (Advanced Problems)

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:

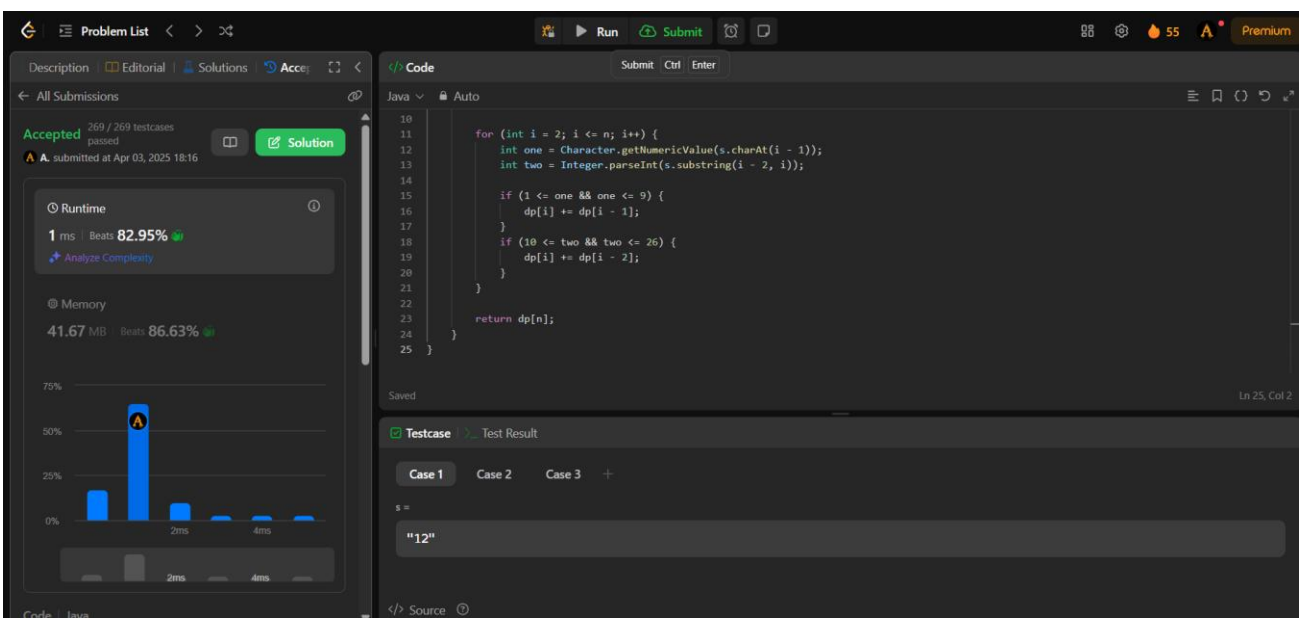


6. Problem: Decode ways

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:



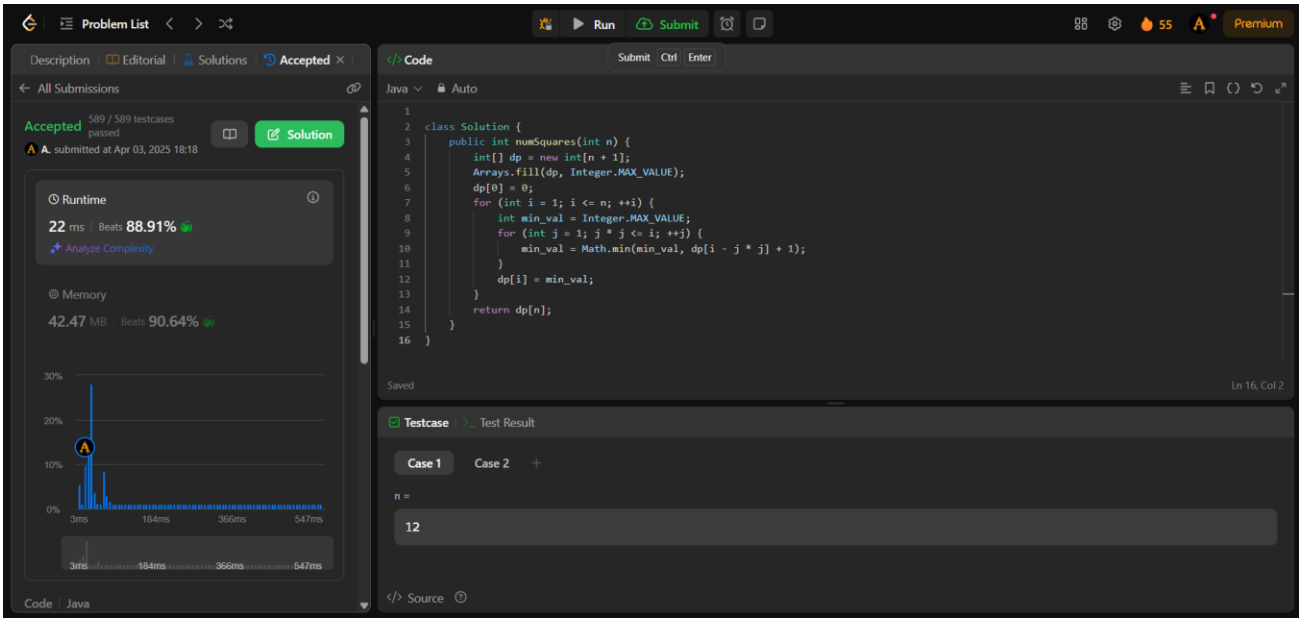
Dynamic Programming (More Challenges)

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:



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: