Experiment-7

Student Name: Aditya Patel
UID: 22BCS11543
Branch: BE-CSE
Section/Group: 22BCS-IOT-640-B
Semester: 6th
Date of Performance: 04/03/2025
Subject Name: AP Lab-2
Subject Code: 22CSH-359

1. Aim: Dynamic Programming Problems

2. Problem Statements:

Medium Level:

- **Problem 2.1:** Given an array of non-negative integers, determine if you can reach the last index by jumping at most the value of the current index.
  Leetcode: Jump Game
  Link: https://leetcode.com/problems/jump-game/
- **Problem 2.2:** Find the number of unique paths in an m x n grid where you can only move right or down.
  Leetcode: Unique Paths
  Link: https://leetcode.com/problems/unique-paths/

Hard Level:

- **Problem 2.3:** Find the contiguous subarray with the maximum product.
  Leetcode: Maximum Product Subarray
  Link: https://leetcode.com/problems/maximum-product-subarray/
- **Problem 2.4:** Determine the maximum profit from buying and selling a stock with a cooldown period after selling.
  Leetcode: Best Time to Buy and Sell Stock with Cooldown
  Link: https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/
- **Problem 2.5:** Given a string and a word dictionary, determine if the string can be segmented into one or more dictionary words.
  Leetcode: Word Break
  Link: https://leetcode.com/problems/word-break/

3. Implementation (Code):

Problem 2.1: Jump Game

```
class Solution {
    public boolean canJump(int[] nums) {
        int maxReach = 0;
        for (int i = 0; i < nums.length; i++) {
            if (i > maxReach) return false;
            maxReach = Math.max(maxReach, i + nums[i]);
        }
        return true;
    }
}
```

Problem 2.2: Unique Paths

```
class Solution {
    public int uniquePaths(int m, int n) {
        int[][] dp = new int[m][n];
        for (int i = 0; i < m; i++) dp[i][0] = 1;
        for (int j = 0; j < n; j++) dp[0][j] = 1;
        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
            }
        }
        return dp[m - 1][n - 1];
    }
}
```

Problem 2.3: Maximum Product Subarray

```
class Solution {
    public int maxProduct(int[] nums) {
        int maxProduct = nums[0], minProduct = nums[0], result = nums[0];
        for (int i = 1; i < nums.length; i++) {
            if (nums[i] < 0) {
                int temp = maxProduct;
                maxProduct = minProduct;
                minProduct = temp;
            }
            maxProduct = Math.max(nums[i], maxProduct * nums[i]);
            minProduct = Math.min(nums[i], minProduct * nums[i]);
            result = Math.max(result, maxProduct);
        }
        return result;
    }
```

```
}
```

Problem 2.4: Best Time to Buy and Sell Stock with Cooldown

```
class Solution {
    public int maxProfit(int[] prices) {
        if (prices.length == 0) return 0;
        int sell = 0, prevSell = 0, buy = Integer.MIN_VALUE, prevBuy;
        for (int price : prices) {
            prevBuy = buy;
            buy = Math.max(prevSell - price, buy);
            prevSell = sell;
            sell = Math.max(prevBuy + price, sell);
        }
        return sell;
    }
}
```

Problem 2.5: Word Break

```
import java.util.*;
class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
        Set<String> wordSet = new HashSet<>(wordDict);
        boolean[] dp = new boolean[s.length() + 1];
        dp[0] = true;
        for (int i = 1; i <= s.length(); i++) {
            for (int j = 0; j < i; j++) {
                if (dp[j] && wordSet.contains(s.substring(j, i))) {
                    dp[i] = true;
                    break;
                }
            }
        }
        return dp[s.length()];
    }
}
```

4. Output:

Fig-1 (Jump Game)

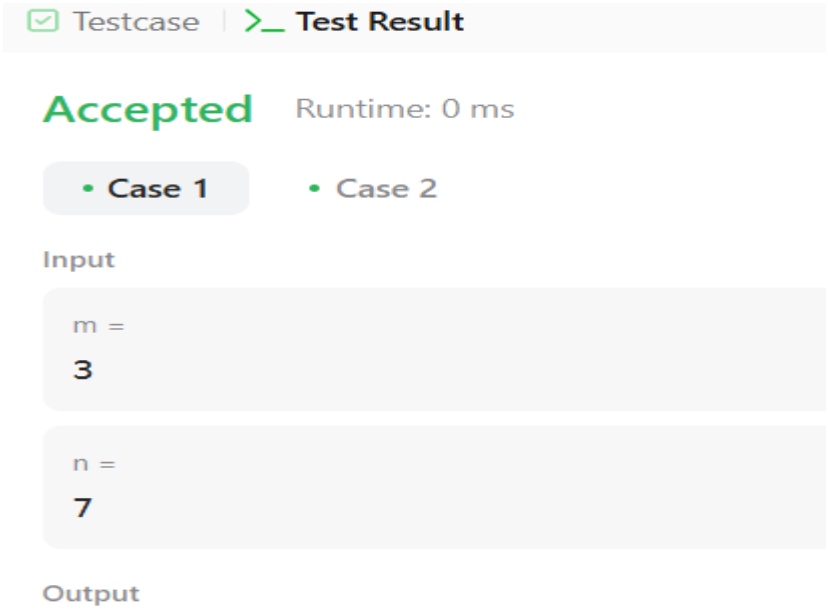

Fig-2 (Unique Paths)

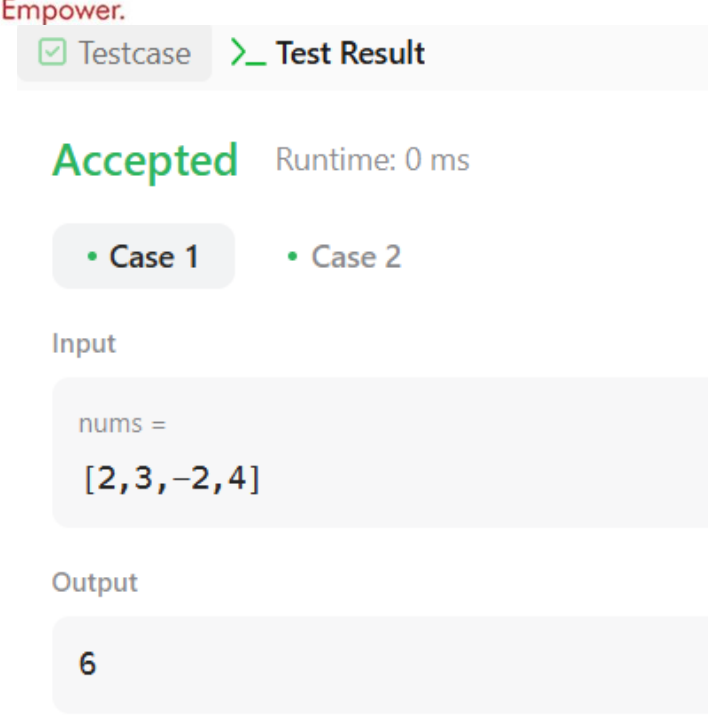

Fig-3 (Maximum Product Subarray)

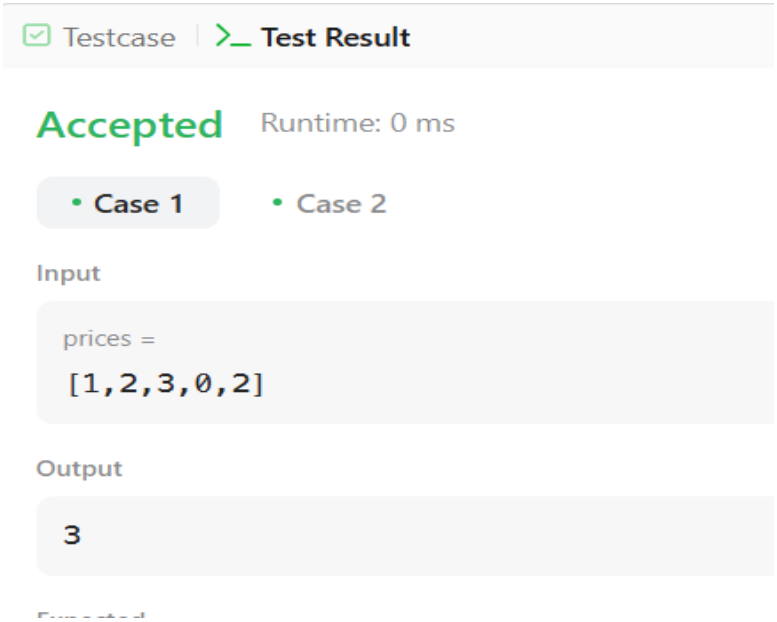

Fig-4 (Best Time to Buy and Sell Stock with Cooldown)

Fig-5 (Word Break)

5. Learning Outcome:

1. Understanding different dynamic programming approaches.
2. Learning optimization techniques like memoization and bottom-up DP.
3. Developing efficient strategies for subproblems and overlapping subproblems.
4. Improving problem-solving skills with real-world scenarios.