DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Experiment 7

Name: Mandeep Sharma **Branch: BE-IT**
Uid: 22BET10014 **Semester: 6 TH**
Section/Group: 22BET_IO1_701/A **Date Of Performance: 28/2/2025**
Subject Name: Advanced Programming Lab-2 **Subject Code: 22ITT-367**

- **Aim:**

1. Find how many distinct ways you can climb to the top of a staircase with n steps, taking 1 or 2 steps at a time.
2. Find the contiguous subarray with the largest sum within an array of integers.
3. Determine the maximum money you can rob from houses without robbing two adjacent ones.
4. Determine if you can reach the last index from the first, given jump limits at each position.
5. Count the number of unique paths from the top-left to bottom-right of an m x n grid, moving only right or down.
6. Find the minimum number of coins needed to make a given amount using a set of denominations.
7. Find the length of the longest subsequence where each element is larger than the previous one.
8. Find the contiguous subarray with the largest product.
9. Count the number of ways to decode a string of digits into letters (e.g., "12" → "AB" or "L").
10. Find the minimum number of perfect square numbers (like 1, 4, 9...) that sum up to a given number n.
11. Return all possible ways to break a string into valid words from a dictionary.

- **Code:**

1. Climbing stairs:

```
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];
    }
}
```

2. Maximum Subarray:

```
class Solution {
    public int maxSubArray(int[] nums) {
        int currentSum = nums[0];
        int maxSum = nums[0];
        for (int i = 1; i < nums.length; i++) {
            currentSum = Math.max(nums[i], currentSum + nums[i]);
```

```
            maxSum = Math.max(maxSum, currentSum);
        }
        return maxSum;
    }
}
```

3. House Robber:

```
class Solution {
    public int rob(int[] nums) {
        if (nums.length == 0) return 0;
        if (nums.length == 1) return nums[0];
        int prev1 = 0; // max profit excluding the current house
        int prev2 = 0; // max profit including the current house
        for (int num : nums) {
            int temp = Math.max(prev1, prev2 + num);
            prev2 = prev1;
            prev1 = temp;
        }
        return prev1;
    }
}
```

4. Jump Game:

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

5. 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];
    }
}
```

6. Coin Change:

```
class Solution {
    public int coinChange(int[] coins, int amount) {
        int max = amount + 1;
        int[] dp = new int[amount + 1];
        Arrays.fill(dp, max);
        dp[0] = 0;
        for (int coin : coins) {
            for (int i = coin; i <= amount; i++) {
                dp[i] = Math.min(dp[i], dp[i - coin] + 1);
            }
        }
        return dp[amount] == max ? -1 : dp[amount];
    }
}
```

7. Longest Increasing Subsequence:

```
class Solution {
    public int lengthOfLIS(int[] nums) {
        if (nums.length == 0) return 0;
        int[] dp = new int[nums.length];
        Arrays.fill(dp, 1);
        int maxLength = 1;
        for (int i = 1; i < nums.length; i++) {
            for (int j = 0; j < i; j++) {
                if (nums[i] > nums[j]) {
                    dp[i] = Math.max(dp[i], dp[j] + 1);
                }
            }
            maxLength = Math.max(maxLength, dp[i]);
        }
        return maxLength;
    }
}
```

8. Maximum Product Subarray:

```
class Solution {
```

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

9. Decode Ways:

```
class Solution {
    public int numDecodings(String s) {
        if (s == null || s.length() == 0 || s.charAt(0) == '0') return 0;
        int n = s.length();
        int[] dp = new int[n + 1];
        dp[0] = 1;
        dp[1] = s.charAt(0) == '0' ? 0 : 1;
        for (int i = 2; i <= n; i++) {
            int oneDigit = Integer.parseInt(s.substring(i - 1, i));
            int twoDigits = Integer.parseInt(s.substring(i - 2, i));
            if (oneDigit >= 1 && oneDigit <= 9) {
                dp[i] += dp[i - 1];
            }
            if (twoDigits >= 10 && twoDigits <= 26) {
                dp[i] += dp[i - 2];
            }
        }
        return dp[n];
    }
}
```

10. Perfect Squares:

```
import java.util.Arrays;
class Solution {
    public int numSquares(int n) {
        int[] dp = new int[n + 1];
        Arrays.fill(dp, Integer.MAX_VALUE);
```

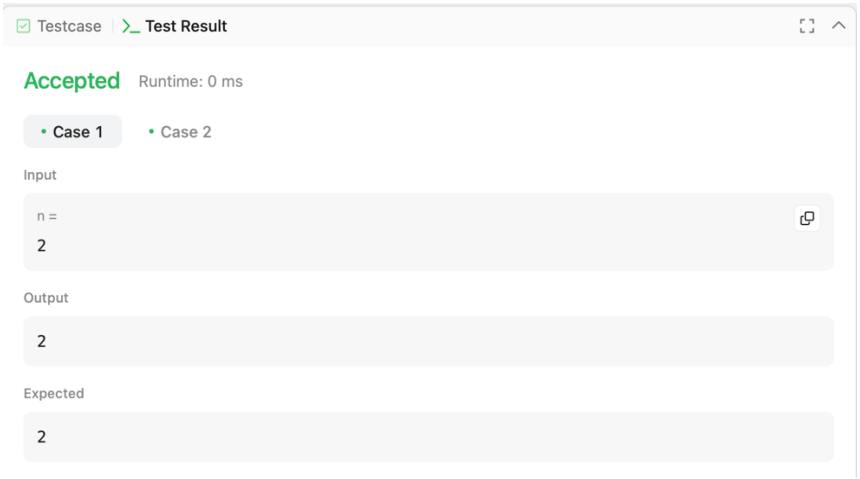
```
        dp[0] = 0;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j * j <= i; j++) {
                dp[i] = Math.min(dp[i], dp[i - j * j] + 1);
            }
        }
        return dp[n];
    }
}
```

11. Word Break:

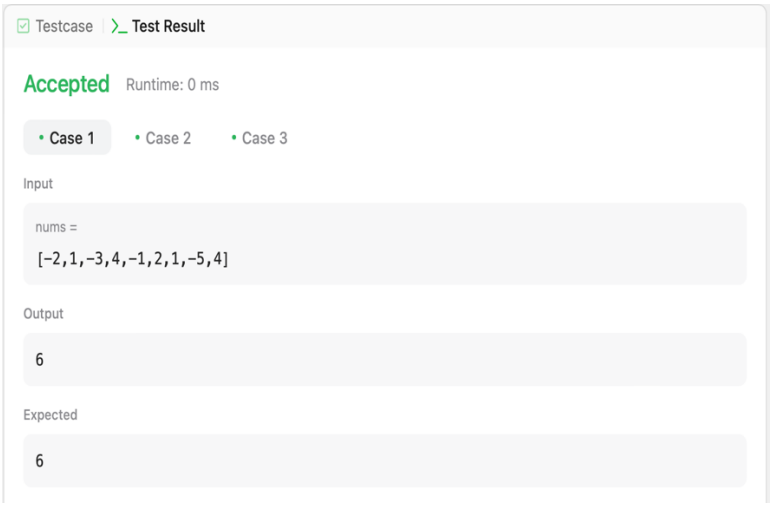
```
import java.util.*;
class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
        Set<String> wordSet = new HashSet<>(wordDict);
        boolean[] dp = new boolean[s.length() + 1];
        dp[0] = true;
        for (int i = 1; i <= s.length(); i++) {
            for (int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

- **Output:**

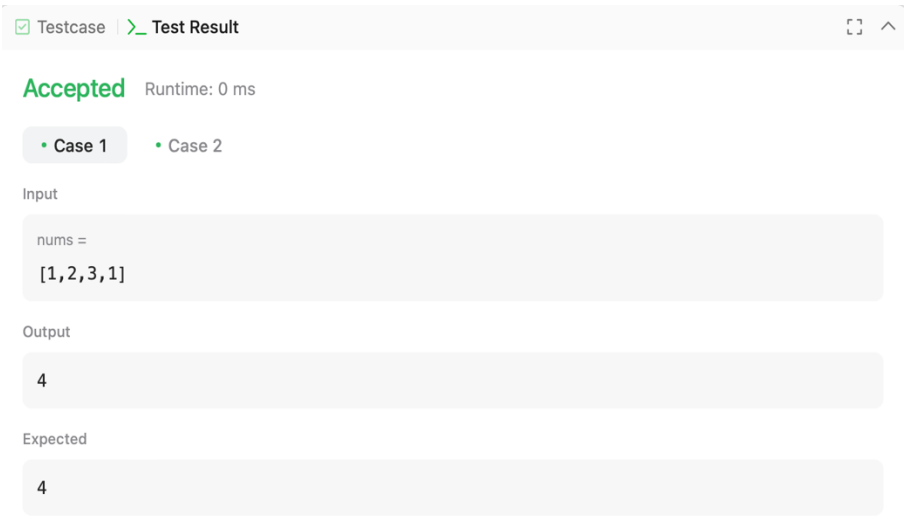
1.



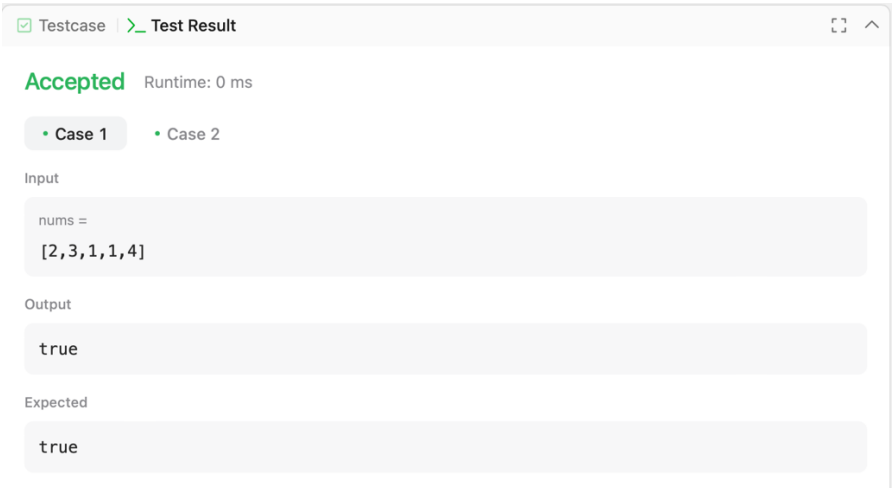
2.


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
nums =
[-2,1,-3,4,-1,2,1,-5,4]
Output
6
Expected
6

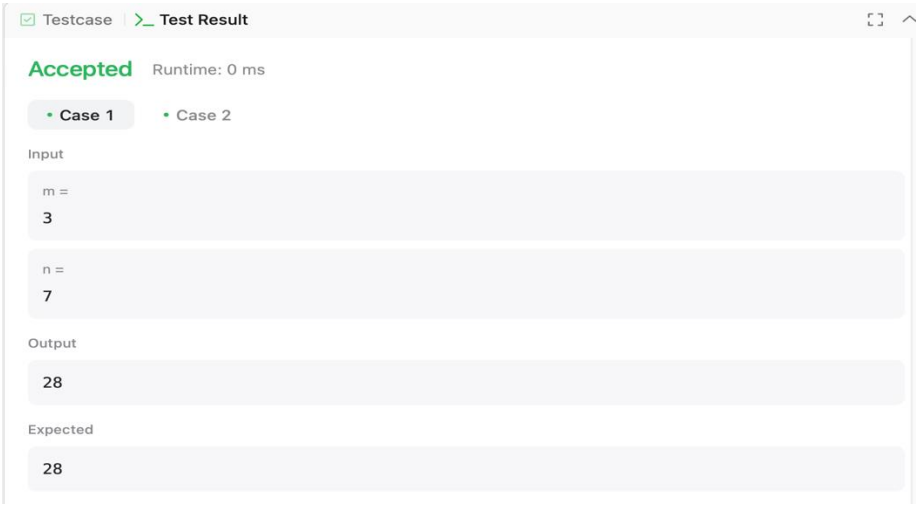
3.


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
nums =
[1,2,3,1]
Output
4
Expected
4

4.


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
nums =
[2,3,1,1,4]
Output
true
Expected
true

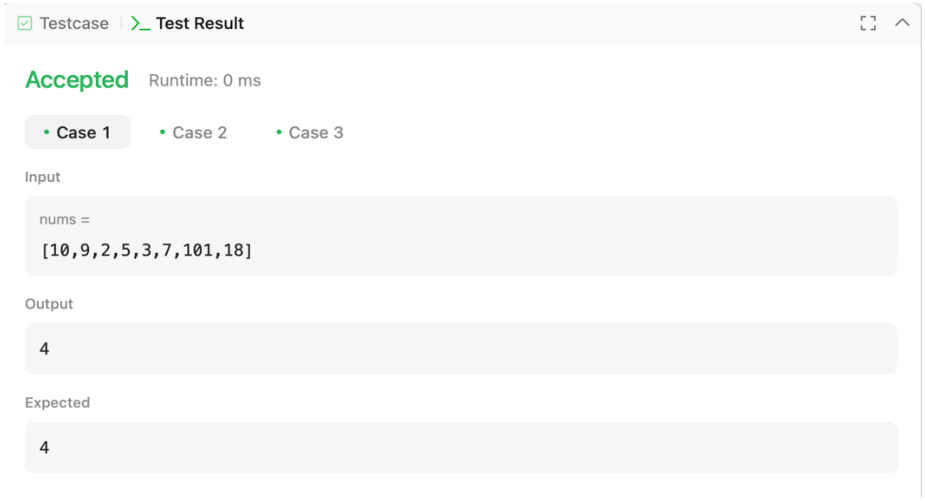
5.


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
m =
3
n =
7
Output
28
Expected
28

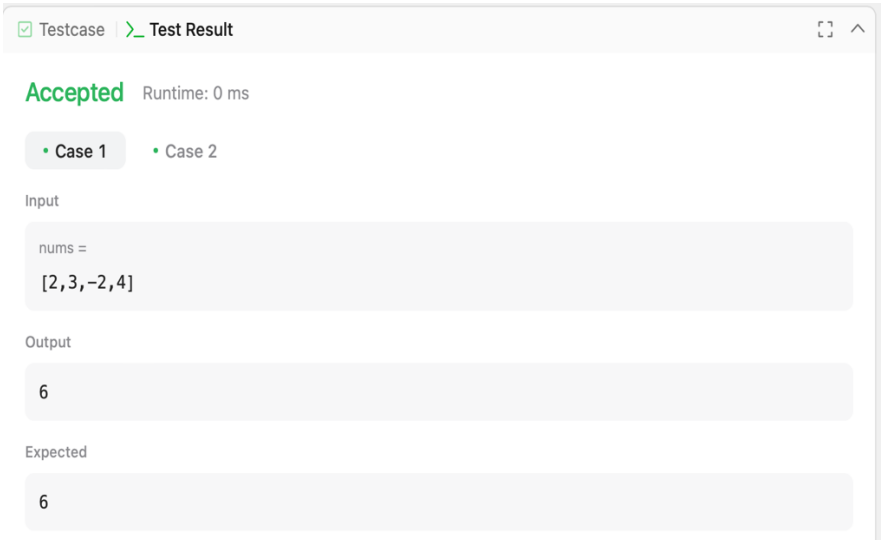
6.


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
coins =
[1,2,5]
amount =
11
Output
3
Expected
3

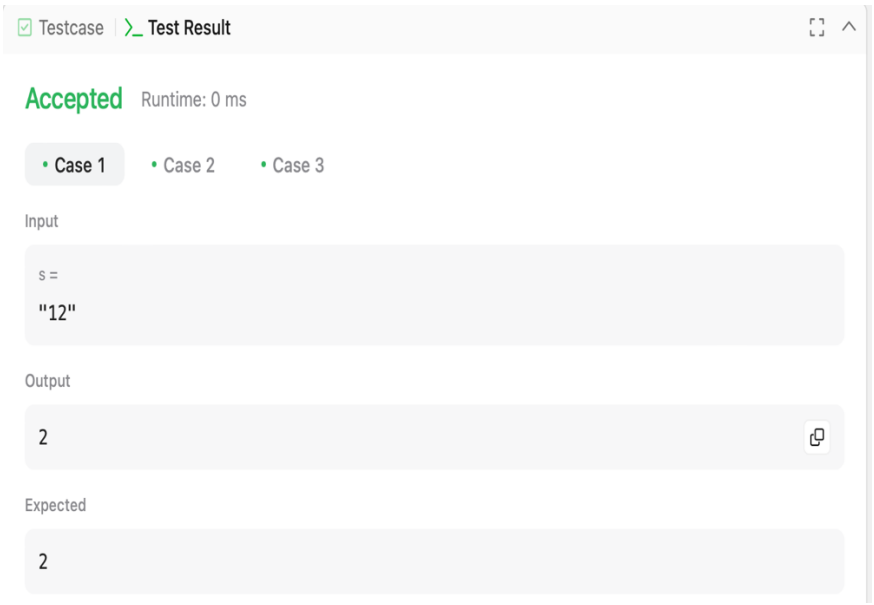
7.


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
nums =
[10,9,2,5,3,7,101,18]
Output
4
Expected
4

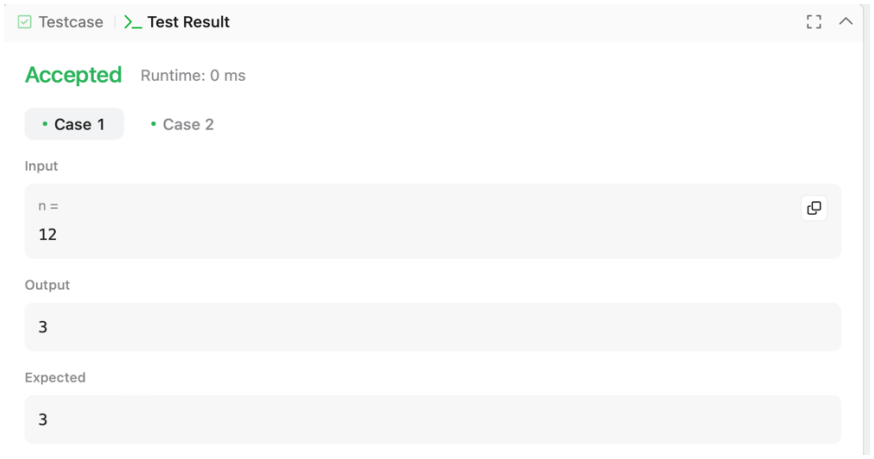
8.


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
nums =
[2,3,-2,4]
Output
6
Expected
6

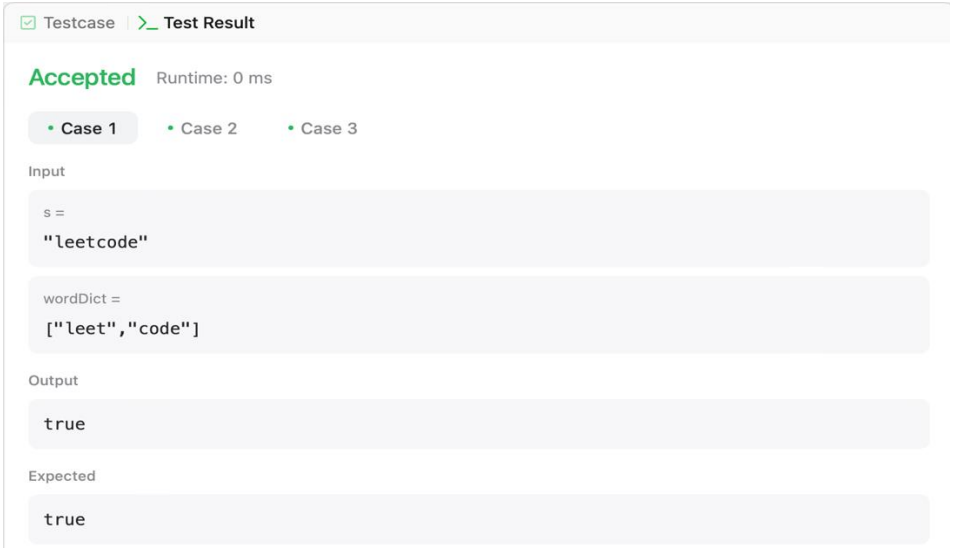
9.


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
s =
"12"
Output
2
Expected
2

10.


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
n =
12
Output
3
Expected
3

11.


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
s =
"leetcode"
wordDict =
["leet","code"]
Output
true
Expected
true