

### Experiment-7

**Name: Sahil Kashyap**

**UID: 22BET10302**

**Branch: BE-IT**

**Section/Group: 22BET\_702-A**

**Semester: 6**

**Date of Performance: 20-03-25**

**Subject Name: Advanced Programming Lab-2 Subject Code: 22ITP-351**

#### **Problem 1- House Robber**

**Aim** -You are given an array nums representing the amount of money in each house along a street. You cannot rob two adjacent houses. Find the maximum amount of money you can rob without alerting the police.

#### **Code:**

```
class Solution
{ public:
    int rob(vector<int>& nums)
    { int n = nums.size();

        if (n == 1)
        { return
            nums[0];
        }

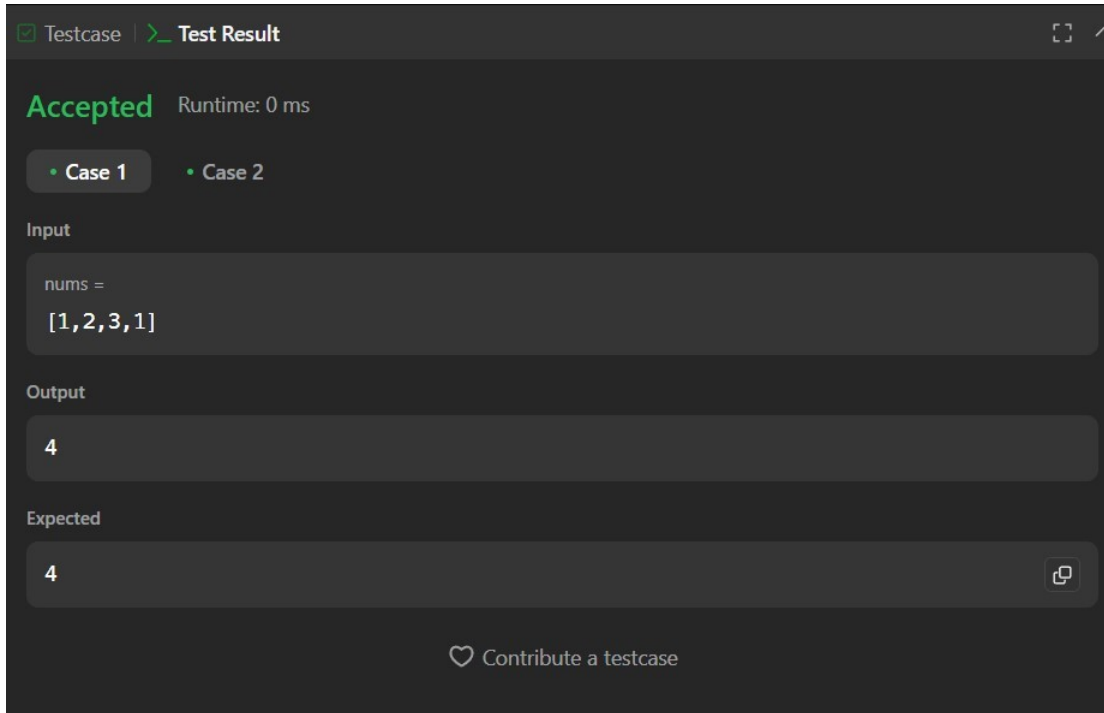
        vector<int> dp(n, 0);

        dp[0] = nums[0];
        dp[1] = max(nums[0], nums[1]);

        for (int i = 2; i < n; i++) {
            dp[i] = max(dp[i - 1], nums[i] + dp[i - 2]);
        }

        return dp[n - 1];
    }
};
```

## Output:



## Problem 2-Jump Game

**Aim :-** Given an array `nums` where `nums[i]` represents the maximum jump length from index `i`, determine if you can reach the last index starting from index 0.

### Code:

```
class Solution
{ public:
    bool canJump(vector<int>& nums)
    { int goal = nums.size() - 1;
      for (int i = nums.size() - 1; i >= 0; i--)
      { if (i + nums[i] >= goal)
        { goal = i;
        }
      }
    }
}
```

```
        return goal == 0;
    }
};
```

### Output:

☒ Testcase | [Test Result](#)

**Accepted** Runtime: 0 ms

- Case 1
- Case 2

Input


nums =  
[2,3,1,1,4]

Output

true

Expected

true

 [Contribute a testcase](#)

### Problem 3:-Unique Paths

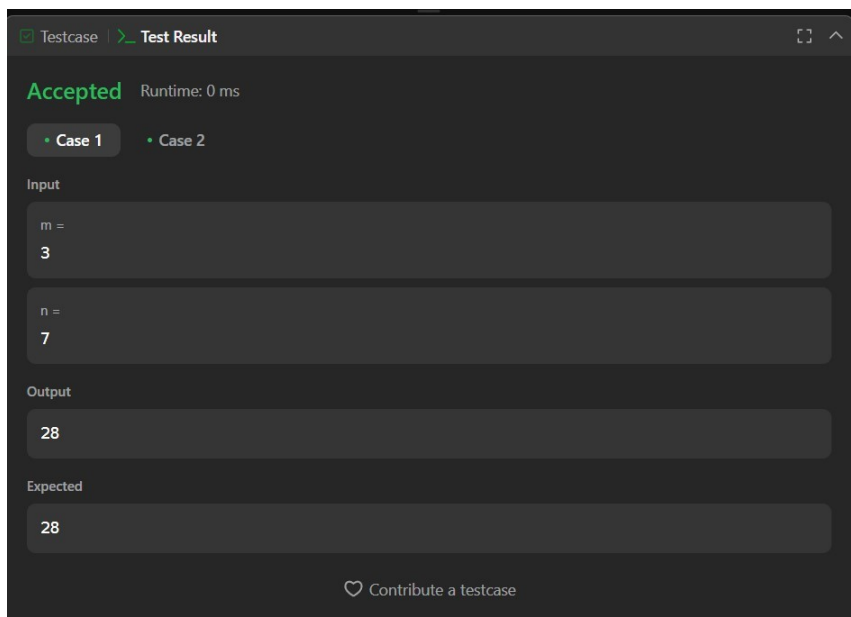
**Aim-** Given the two integers m and n, return the number of possible unique paths that the robot can take to reach the bottom-right corner.

### Code:

```
class Solution
{ public:
//    int f(int m,int n,vector<vector<int>>& dp){
//        if(m==0 && n==0) return 1;
//        if(m<0 || n<0) return 0;
//        if(dp[m][n]!=-1) return dp[m][n];
//        int up = f(m-1,n,dp);
```

```
//      int left = f(m,n-1,dp);
//      return dp[m][n] = up+left;
//  }
int uniquePaths(int m, int n)
{ int dp[m][n];
  // tabulation
  dp[0][0]=1;
  for(int i = 0 ; i<m ;
    i++){ for(int j = 0 ; j<n ;
    j++){
      if(i==0 && j==0) continue;
      int up=0,left=0;
      if(i>0) up = dp[i-1][j];
      if(j>0) left = dp[i][j-1];
      dp[i][j] = up + left;
    }
  }
  return dp[m-1][n-1];
}
```

## Output:



### Problem 4:- Coin Change

**Aim-** Given a list of coins with different denominations and an integer amount, find the fewest number of coins needed to make up that amount. If it's not possible, return -1.

#### Code:

```
class Solution
{ public:
    int coinChange(vector<int>& coins, int amount)
    { int n=coins.size();
      int t[n+1][amount+1];
      for(int i=0;i<n+1;i++){
          for(int
              j=0;j<amount+1;j++){ if(i==
              0){
                  t[i][j]=INT_MAX-1;
              }
              if(j==0){ t[i]
                  [j]=0;
              }
          }
      }
      for(int
          i=1;i<amount+1;i++){ if(i%coins[0]
          ==0) t[1][i]=i/coins[0]; else
          t[1][i]=INT_MAX-1;
      }
      for(int
          i=2;i<n+1;i++){ for(in
          t
          j=1;j<amount+1;j++){ if(c
          oins[i-1]<=j){
              t[i][j]=min(1+t[i][j-coins[i-1]],t[i-1][j]);
          }
          else{
              t[i][j]=t[i-1][j];
          }
      }
```

```
        }  
    }  
    if( t[n][amount]==INT_MAX-  
        1){ return -1;  
    }  
    else{  
        return t[n][amount];  
    }  
}  
};
```

## Output:

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

[• Case 1](#) [• Case 2](#) [• Case 3](#)

Input

coins =  
[1,2,5]


amount =  
11

Output

3

Expected

3

 [Contribute a testcase](#)

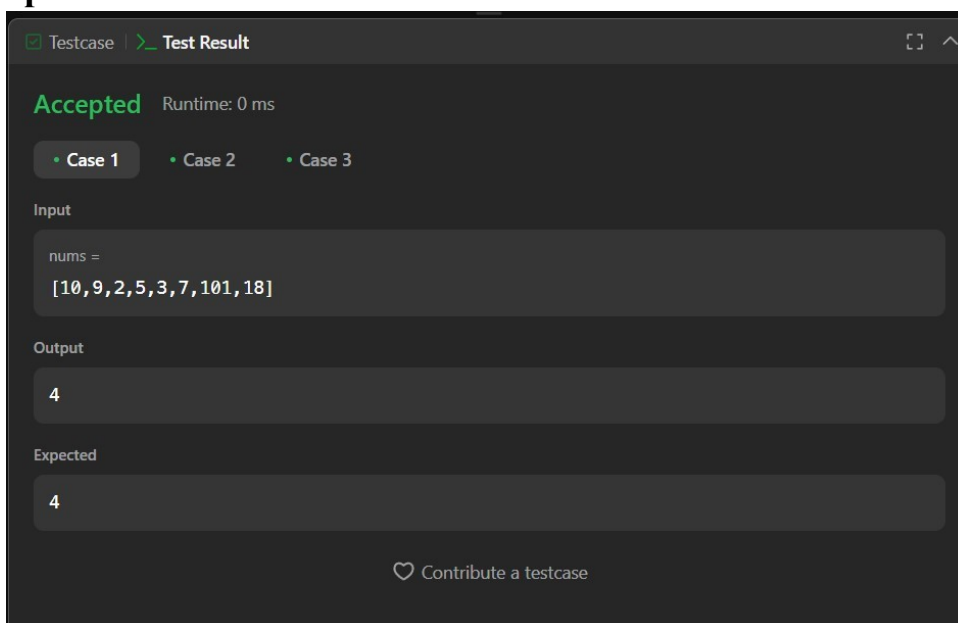
## Problem 5. Longest Increasing Subsequence

**Aim-**Given an integer array nums, return the length of the longest increasing subsequence (LIS). A subsequence is a sequence derived from an array by deleting some or no elements without changing the order.

### Code:

```
class Solution { // 256 ms, faster than 42.84%
public:
    int lengthOfLIS(vector<int>& nums)
    { int n = nums.size();
      vector<int> dp(n, 1);
      for (int i = 0; i < n; ++i)
          for (int j = 0; j < i; ++j)
              if (nums[i] > nums[j] && dp[i] < dp[j] + 1)
                  dp[i] = dp[j] + 1;
      return *max_element(dp.begin(), dp.end());
    }
};
```

### Output:



## Problem 6. Maximum Product Subarray

**Aim-**Given an integer array nums, find the contiguous subarray (of at least one number) that has the largest product and return the product.

### Code:

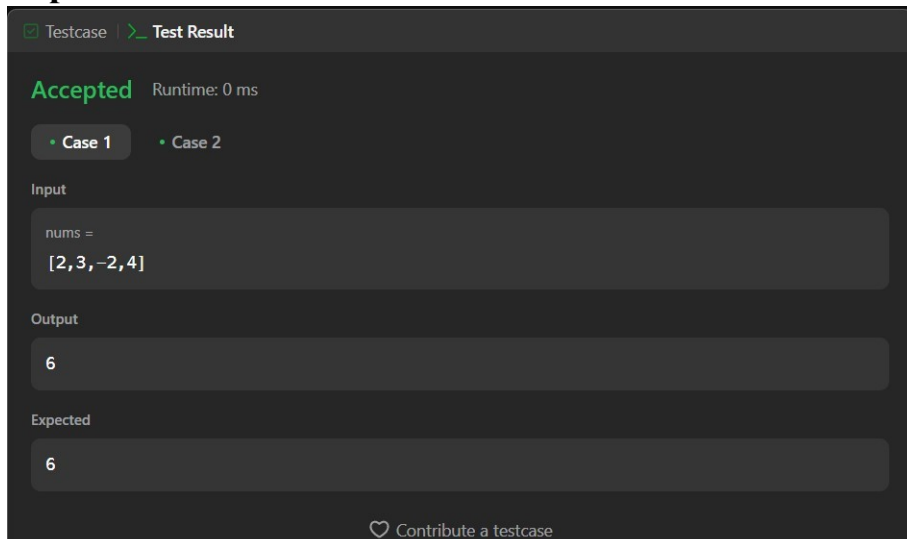
```
class Solution
{ public:
    int maxProduct(vector<int>& nums) {
        int res = *max_element(nums.begin(), nums.end());
        int curMax = 1, curMin = 1;

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

            res = max(res, curMax);
        }

        return res;
    }
};
```

### Output:



Testcase Test Result

**Accepted** Runtime: 0 ms

Case 1 Case 2

Input

nums =  
[2, 3, -2, 4]

Output

6

Expected

6

Contribute a testcase



### Problem 7:- Decode Ways

**Aim-**Given a string  $s$  containing digits, return the number of ways to decode it. Each digit (1-26) maps to A-Z (1  $\rightarrow$  A, 2  $\rightarrow$  B, ..., 26  $\rightarrow$  Z).

#### Code:

```
class Solution
{ public:
    int numDecodings(string s)
    { if (s[0] == '0') {
        return 0;
    }

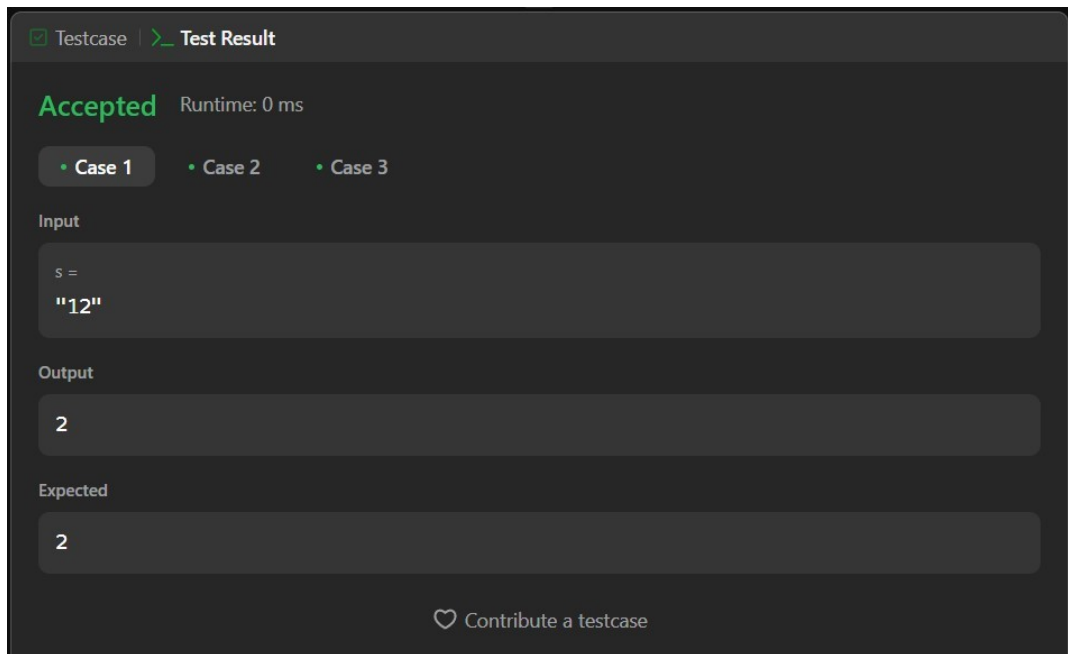
    int n = s.length();
    vector<int> dp(n + 1, 0);
    dp[0] = dp[1] = 1;

    for (int i = 2; i <= n; i++)
    { int one = s[i - 1] - '0';
      int two = stoi(s.substr(i - 2, 2));

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

    return dp[n];
}
};
```

## Output:



## Problem 8:-Best time to buy and Sell a Stock with Cooldown

**Aim-**You are given an array prices where prices[i] is the price of a given stock on day i. You may buy and sell the stock multiple times, but after selling, you must wait one day (cooldown) before buying again. Find the maximum profit you can achieve.

## Code:-

```
class Solution
{
public:
    int maxProfit(vector<int>& prices) {

        int coolDown=0, sell=0, hold=std::numeric_limits<int>::min();

        for( int stockPrice_Day_i : prices){

            int prevCoolDown = coolDown, prevSell = sell, prevHold = hold;
            coolDown = max(prevCoolDown, prevSell);
            sell = prevHold + stockPrice_Day_i;
            hold = max(prevHold, prevCoolDown - stockPrice_Day_i);
        }
    }
};
```

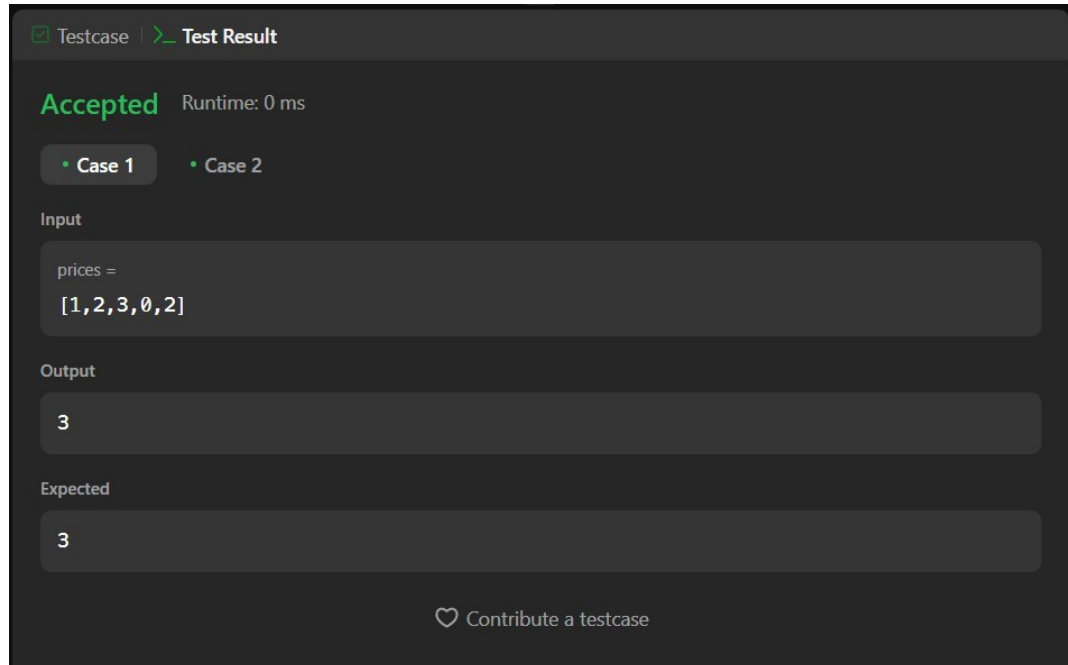
```

    }

    return max( sell, coolDown );
}
}

```

**Output:-**



## Problem 9:- Perfect Squares

**Aim-** Given an integer n, return the least number of perfect square numbers (1, 4, 9, 16, ...) that sum to n.

**Code:-**

```

class Solution
{ public:
    int numSquares(int n) {
        vector<int> dp(n + 1, INT_MAX); dp[0]
        = 0;
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j * j <= i; ++j){
                dp[i] = min(dp[i], dp[i - j * j] + 1);
            }
        }
    }
}

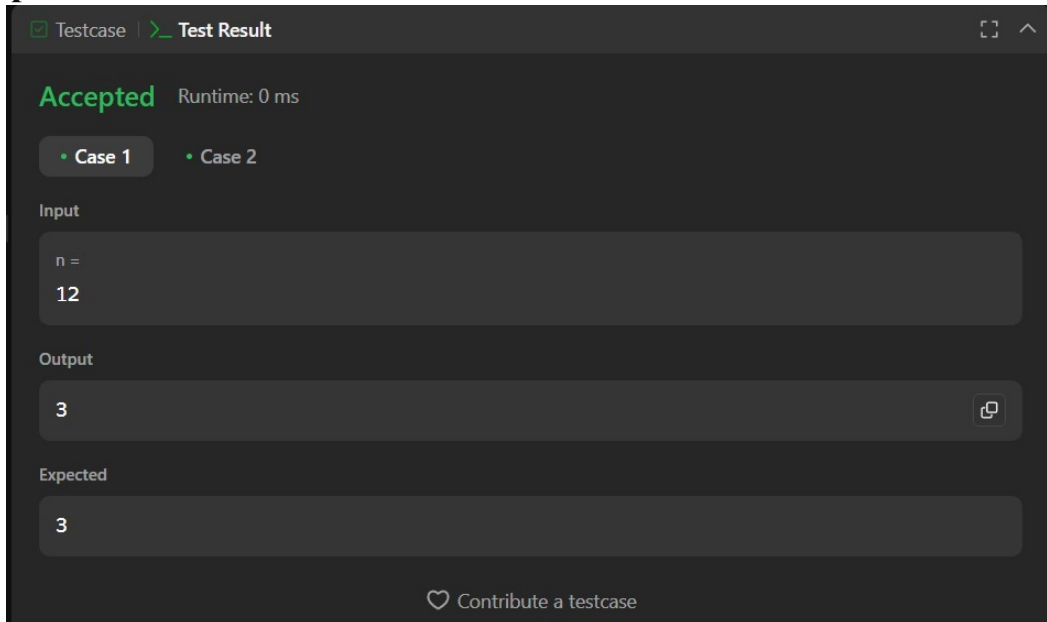
```

```

    }
  }
  return dp[n];
}
};

```

**Output:-**



## Problem 10:- Word Break

**Aim-**Given a string *s* and a dictionary of strings *wordDict*, return true if *s* can be segmented into a space-separated sequence of one or more dictionary words.

**Code:-**

```

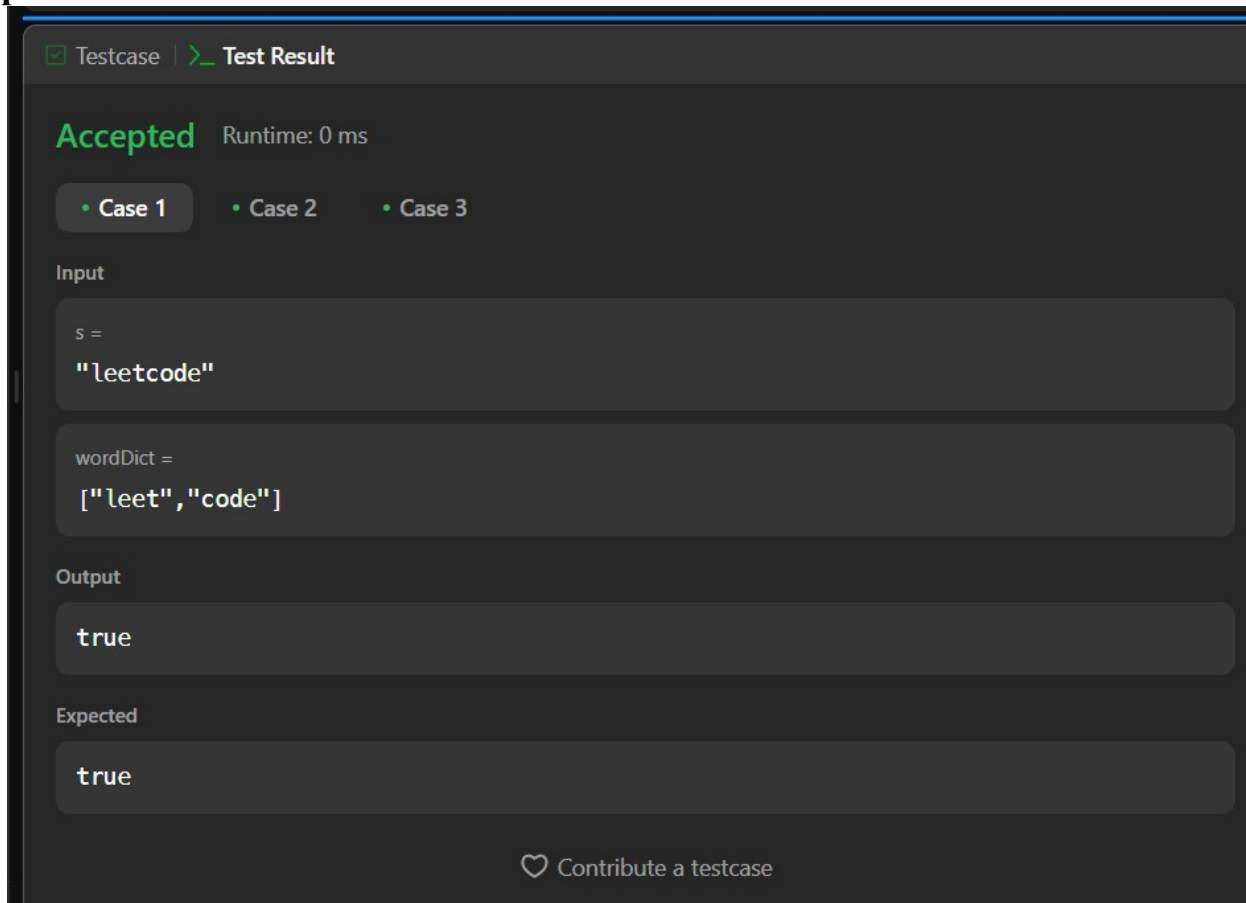
class Solution
{ public:
    bool wordBreak(string s, vector<string>& wordDict)
    { vector<bool> dp(s.size() + 1, false);
      dp[0] = true;

      for (int i = 1; i <= s.size(); i++)
        { for (const string& w : wordDict)
            { int start = i - w.length();

```

```
        if (start >= 0 && dp[start] && s.substr(start, w.length()) == w)
            { dp[i] = true;
              break;
            }
        }
    }
    return dp[s.size()];
}
};
```

**Output:-**



The screenshot shows a test result interface for a coding problem. At the top, there are tabs for 'Testcase' and 'Test Result'. The 'Test Result' tab is active, showing a green 'Accepted' status and a runtime of '0 ms'. Below this, there are three tabs for 'Case 1', 'Case 2', and 'Case 3', with 'Case 1' selected. The 'Input' section contains two text boxes: the first with 's =' and 'leetcode', and the second with 'wordDict =' and '["leet", "code"]'. The 'Output' section shows a text box with 'true'. The 'Expected' section also shows a text box with 'true'. At the bottom, there is a heart icon and the text 'Contribute a testcase'.

Testcase | Test Result

**Accepted** Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

s =  
"leetcode"

wordDict =  
["leet", "code"]

Output

true

Expected

true

♥ Contribute a testcase

### Problem 11:- Word Break 2

**Aim** - Given a string *s* and a dictionary of strings *wordDict*, add spaces in *s* to construct a sentence where each word is a valid dictionary word. Return all such possible sentences in any order.

#### Code:-

```
class Solution
{ public:
    void solve(string s, vector<string>& res, unordered_set<string>& st,
vector<string>&temp){
        if(s.length() ==
            0){ string str = "";
                for(auto
it:temp){ str += it +
                    " ";
                }
                str.pop_back();
                res.push_back(str);
                return;
            }
            for(int i=0;i<s.length();
i++){ if(st.count(s.substr(0,
i+1))) {
                temp.push_back(s.substr(0, i+1));
                solve(s.substr(i+1), res, st, temp);
                temp.pop_back();
            }
        }
    }
vector<string> wordBreak(string s, vector<string>& wordDict)
{ vector<string>res, temp;
    unordered_set<string>st(wordDict.begin(), wordDict.end());

    solve(s, res, st, temp);
    return res;
}
};
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Output-

☒ Testcase | Test Result

**Accepted** Runtime: 0 ms

• Case 1

• Case 2

• Case 3

Input

s =

"catsanddog"

wordDict =

["cat","cats","and","sand","dog"]

Output

["cat sand dog","cats and dog"]

Expected

["cats and dog","cat sand dog"]

Contribute a testcase