EXPERIMENT - 2

Student Name: JEET BHARTI **UID: 22BCS14804**
Branch: BE-CSE **Section: FL_IOT-602 'A'**
Semester: 6th **DOP: 21/01/2025**
Subject: Advanced Programming Lab-II **Subject Code:22CSP-351**

1. Aim:

Problem 1.2.1: Two Sum

- **Problem Statement:** Given an array of integers nums and an integer target, return the indices of the two numbers such that they add up to target. Each input has exactly one solution, and you cannot use the same element twice.

Problem 1.2.2: Jump Game II

- **Problem Statement:** You are given a 0-indexed array nums of length n. You are initially positioned at nums[0]. Each element nums[i] represents the maximum length of a forward jump from index i. Return the minimum number of jumps to reach nums[n - 1].

Problem 1.2.3: Simplify Path

- **Problem Statement**: Given a string path, which is an absolute path to a file or directory in a Unix-style file system, convert it to the simplified canonical path.

2. Objective:

Objective: The objective of this experiment is to enhance problem-solving and algorithmic skills by solving fundamental programming problems like Two Sum, Jump Game II, and Simplify Path. It focuses on designing efficient algorithms, implementing solutions in code, and validating them with test cases. Additionally, the experiment emphasizes analyzing time and space complexity to ensure optimal performance.

3. Algorithm:

1. Initialize an empty hash map (dict).
2. Iterate through the nums array:
    - For each element num, calculate the complement: complement = target - num.
    - Check if the complement exists in the hash map:
        - If it does, return the indices of the complement and the current number.
        - If it doesn't, add the current number and its index to the hash map.
3. Return the indices of the two numbers that add up to the target.

4. Implementation / Code:

Code: 1.2.1

```python
class Solution:
    def twoSum(self, nums, target):
        seen = {}
        for i, num in enumerate(nums):
            complement = target - num
            if complement in seen:
                return [seen[complement], i]
            seen[num] = i

# ROSH
solution = Solution()

nums1 = [2, 7, 11, 15]
target1 = 9
print(solution.twoSum(nums1, target1))

nums2 = [3, 2, 4]
target2 = 6
print(solution.twoSum(nums2, target2))

nums3 = [3, 3]
target3 = 6
print(solution.twoSum(nums3, target3))
```

Output:

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

nums =
[2,7,11,15]

target =
9

Stdout

```
[0, 1]
[1, 2]
[0, 1]
```

Output

[0,1]

Expected

[0,1]

CODE: 1.2.2

```python
class Solution:
    def jump(self, nums):
        n = len(nums)
        jumps = 0
        current_end = 0
        farthest = 0

        for i in range(n - 1):
            farthest = max(farthest, i + nums[i])
            if i == current_end:
                jumps += 1
                current_end = farthest
                if current_end >= n - 1:
                    break

        return jumps

# ROSH
solution = Solution()

nums1 = [2, 3, 1, 1, 4]
print(solution.jump(nums1))

nums2 = [2, 3, 0, 1, 4]
print(solution.jump(nums2))
```

OUTPUT:

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

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

Stdout

2
2

Output

2

Expected

2

Discover. Learn. Empower.

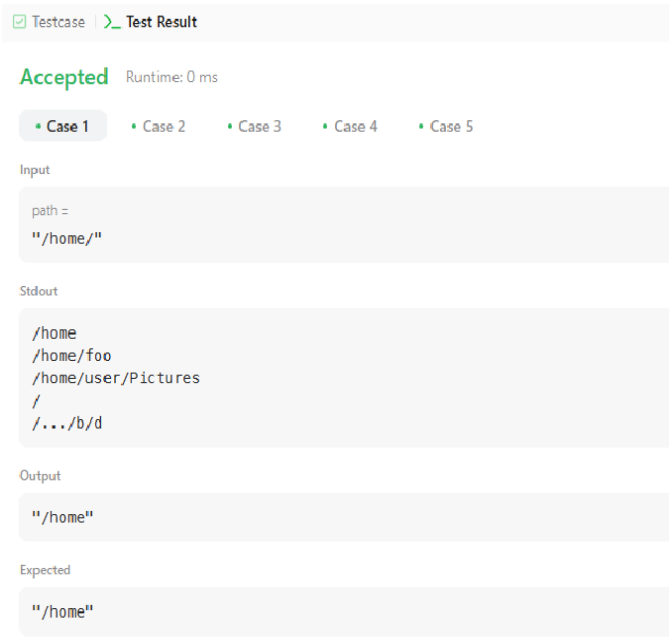
CODE: 1.2.3

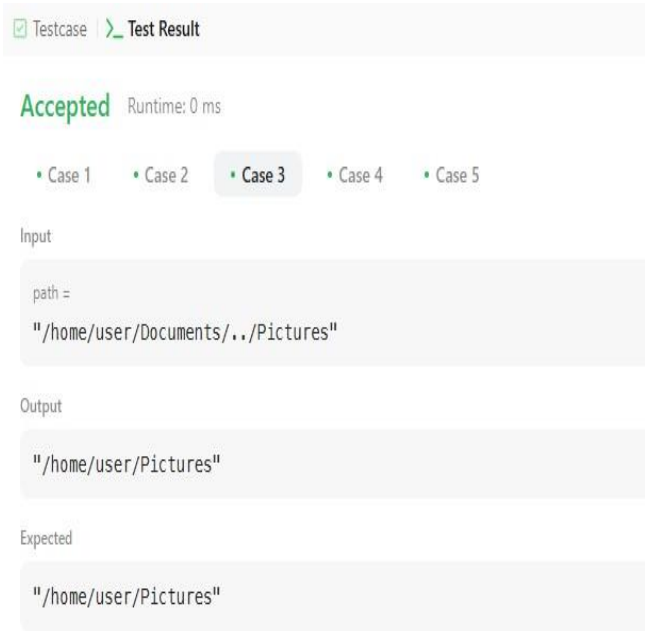
</> Code

Python3 Auto

```python
    def simplifyPath(self, path):
        stack = []
        parts = path.split('/')
        for part in parts:
            if part == '..':
                if stack:
                    stack.pop()
            elif part and part != '.':
                stack.append(part)
        return '/' + '/'.join(stack)

# ROSH
solution = Solution()

path1 = "/home/"
print(solution.simplifyPath(path1))

path2 = "/home//foo/"
print(solution.simplifyPath(path2))

path3 = "/home/user/Documents/../Pictures"
print(solution.simplifyPath(path3))

path4 = "/../"
print(solution.simplifyPath(path4))

path5 = "/.../a/../b/c/../d/./"
print(solution.simplifyPath(path5))
```

OUTPUT:





5. Learning Outcomes:

1. **Problem-Solving with Arrays & Greedy Algorithms** – Understand how to manipulate arrays efficiently and apply greedy strategies to optimize solutions, especially in problems like **Two Sum** and **Jump Game II**.
2. **Data Structures & Hashing** – Gain hands-on experience with **hash maps** to achieve optimal time complexity, particularly in **Two Sum**, where lookup operations can be optimized to **O(1)**.
3. **Graph Traversal & Dynamic Programming** – Develop an understanding of **minimum jump problems**, exploring **BFS-like approaches** and **dynamic programming** to find the optimal path in **Jump Game II**.
4. **String Manipulation & Stack Implementation** – Learn how to efficiently parse and process paths in **Simplify Path** using **stack-based approaches** to handle directory navigation correctly.
5. **Optimization & Edge Case Handling** – Strengthen debugging skills by handling edge cases such as negative numbers, large inputs, special characters in file paths, and performance constraints for real-world applications.