



Experiment 1

Student Name: Sachin

Branch: CSE

Semester: 6th

Subject: AP-LAB-2

UID: 22BCS15411

Section: NTPP_602-A

DOP: 20/01/25

Subject Code: 22CSP-351

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.

Algorithm:

1. Two Sum Problem (Algorithm)

1. Create a hash map to store numbers and their indices.
2. Iterate through the array `nums`:
 - For each number `num`, compute the difference `diff = target - num`.
 - Check if `diff` exists in the hash map.
 - If yes, return `[index of diff, current index]`.
 - Otherwise, store `num` and its index in the hash map.

2. Jump Game II (Algorithm)

1. Initialize `jumps = 0`, `current_end = 0`, and `farthest = 0`.
2. Traverse the array from index 0 to `len(nums) - 1`:
 - Update `farthest = max(farthest, i + nums[i])`.
 - If you reach `current_end`:
 - Increment `jumps`.
 - Update `current_end = farthest`.
3. Return `jumps` after reaching the end of the array.

Simplify Path (Algorithm)

1. Split the input `path` by `/` to get components.
2. Use a stack to process the components:
 - For each component:
 - If it is `.` or empty, skip it.
 - If it is `..`, pop from the stack (if the stack is not empty).
 - Otherwise, push the component onto the stack.
3. Join the elements in the stack with `/` and prepend `/` to construct the simplified path.
4. Return the result.

Code: 1.2.1

```
</> Code
Python3  Auto
1 class Solution:
2     def twoSum(self, nums, target):
3         seen = {}
4         for i, num in enumerate(nums):
5             complement = target - num
6             if complement in seen:
7                 return [seen[complement], i]
8             seen[num] = i
9
10 # ROSH
11 solution = Solution()
12
13 nums1 = [2, 7, 11, 15]
14 target1 = 9
15 print(solution.twoSum(nums1, target1))
16
17 nums2 = [3, 2, 4]
18 target2 = 6
19 print(solution.twoSum(nums2, target2))
20
21 nums3 = [3, 3]
22 target3 = 6
23 print(solution.twoSum(nums3, target3))
24
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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]
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• Case 1 • **Case 2** • Case 3

Input:

nums =
[3,2,4]

target =
6

Output

[1,2]

Expected

[1,2]

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• Case 1 • Case 2 • **Case 3**

Input

nums =
[3,3]

target =
6

Output

[0,1]

Expected

[0,1]

CODE: 1.2.2

```
</> Code
Python3  Auto
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:



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

☒ 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
```

☒ Testcase | [> Test Result](#)

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

```
nums =  
[2,3,0,1,4]
```

Output

```
2
```

Expected

```
2
```

</> Code

Python3   Auto

```
2     def simplifyPath(self, path):
3         stack = []
4         parts = path.split('/')
5         for part in parts:
6             if part == '..':
7                 if stack:
8                     stack.pop()
9             elif part and part != '.':
10                stack.append(part)
11        return '/' + '/'.join(stack)
12
13    # ROSH
14    solution = Solution()
15
16    path1 = "/home/"
17    print(solution.simplifyPath(path1))
18
19    path2 = "/home//foo/"
20    print(solution.simplifyPath(path2))
21
22    path3 = "/home/user/Documents/../Pictures"
23    print(solution.simplifyPath(path3))
24
25    path4 = "/../"
26    print(solution.simplifyPath(path4))
27
28    path5 = "/.../a/..b/c/...d/.."
29    print(solution.simplifyPath(path5))
```

OUTPUT:



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• **Case 1** • Case 2 • Case 3 • Case 4 • Case 5

Input

```
path =  
"/home/"
```

Stdout

```
/home  
/home/foo  
/home/user/Pictures  
/  
/.../b/d
```

Output

```
"/home"
```

Expected

```
"/home"
```

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• Case 1 • **Case 2** • Case 3 • Case 4 • Case 5

Input

```
path =  
"/home//foo/"
```

Output

```
"/home/foo"
```

Expected

```
"/home/foo"
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• Case 1 • Case 2 • **Case 3** • Case 4 • Case 5

Input

```
path =  
"/home/user/Documents/../Pictures"
```

Output

```
"/home/user/Pictures"
```

Expected

```
"/home/user/Pictures"
```

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3 • **Case 4** • Case 5

Input

```
path =  
"/../"
```

Output

```
"/"
```

Expected

```
"/"
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

☒ Testcase ☒ Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3 • Case 4 • **Case 5**

Input

```
path =  
"/.../a/..b/c/..d/./"
```

Output

```
"/.../b/d"
```

Expected

```
"/.../b/d"
```

Learning Outcomes:

- Understand how to efficiently solve the Two Sum Problem using a hash map to optimize time complexity.
- Learn to implement a greedy algorithm for Jump Game II to minimize jumps while maintaining linear time complexity.
- Acquire skills in processing and simplifying file paths using stacks in the Simplify Path problem.
- Develop problem-solving techniques for handling string manipulation and edge cases in Unix-style file systems.
- Enhance algorithmic thinking by applying data structures like hash maps and stacks to solve real-world problems effectively.