PYTHON CODING CHALLENGE

NAME : SOUMYADEEP SINHA (DE235)
DATE : 12 - 11 - 2024

---

1) Given an array of integers `nums` and an integer `target`, return the indices of the two numbers such that they add up to `target`. You may assume that each input: would have exactly one solution, and you may not use the same element twice.

Example:
input:
nums = [2, 7, 11, 15]
target = 9
Output: [0, 1]

SOLUTION

```
def question_1():
    nums = list(map(int, input().split()))
    target = int(input())

    indices = {}

    for i, num in enumerate(nums):
        res = target - num

        if res in indices:
            return [indices[res], i]

        indices[num] = i
    return []

print(question_1())
```

OUTPUTS

```
PS D:\HEXAVARSITY\DomainTraining>
2 7 11 15
9
[0, 1]
```

```
PS D:\HEXAVARSITY\DomainTraining>
5 21 4 8
12
[2, 3]
PS D:\HEXAVARSITY\DomainTraining>
4 25 3 6
7
[0, 2]
```

EXPLANATION :

- The function takes user input for a list of numbers and a target value.
- It initialises a dictionary, indices, to store each number with its index.
- For each number in the list, it calculates the required complement (target - num).
- If the complement is found in indices, it returns the indices of the two numbers that add up to the target.
- If the complement isn't found, it adds the current number and index to indices.
- If no match is found by the end, it returns an empty list.

COMPLEXITY

- **Time Complexity: O(n)**
- **Space Complexity: O(n)**

---

2) Given a string `s`, find the length of the longest substring without repeating characters.

Example:
input:
s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.

SOLUTION

```
def question_2():
    s = input()
    left = 0
    right = 0
    maxlen = 0
    substr = ""
    while right != len(s):
        if s[right] in substr:
            left += 1
        else:
```

```
            right += 1

        substr = s[left:right]

        if len(substr) > maxlen:
            maxlen = len(substr)
    return maxlen

print(question_2())
```

OUTPUT

```
PS D:\HEXAVARSITY\DomainTraining>
abcabcbb
3

PS D:\HEXAVARSITY\DomainTraining>
aaaaaaaa
1

PS D:\HEXAVARSITY\DomainTraining>
abcdefgh
8
```

EXPLANATION

- The function takes a string input from the user.
- It initializes left and right pointers, maxlen for the longest substring length, and substr to track the current substring.
- While right hasn't reached the end of the string:
    - If s[right] is already in substr, it increments left (to adjust the window).
    - If not, it increments right.
- Updates substr to the substring between left and right.
- If the length of substr is greater than maxlen, updates maxlen to this new length.
- Returns maxlen as the final length of the longest substring without repeating characters.

COMPLEXITY

- **Time Complexity: O(n)**
- **Space Complexity: O(n)**

---

3) Given an integer array `coins` representing coins of different denominations and an integer `amount`, return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return `-1`.

Example:

input:
coins = [1, 2, 5]
amount = 11
Output: 3 (11 = 5 + 5 + 1)

SOLUTION

```
def question_3():
    coins = list(map(int, input().split()))
    amount = int(input())

    dp = [amount + 1] * (amount + 1)
    dp[0] = 0

    for a in range(1, amount + 1):
        for c in coins:
            if a - c >= 0:
                dp[a] = min(dp[a], dp[a - c] + 1)

    return dp[amount] if dp[amount] != amount + 1 else -1

print(question_3())
```

OUTPUT

```
PS D:\HEXAVARSITY\DomainTraining>
1 2 5
11
3
PS D:\HEXAVARSITY\DomainTraining>
7
3
-1
PS D:\HEXAVARSITY\DomainTraining>
1
0
0
```

EXPLANATION

- The user provides a list of coin denominations (space-separated) and the target amount.
- **Initialize DP Array:**
  - A dp array of size amount + 1 is created, with each element initialized to amount + 1 to represent an unreachable state.
  - dp[0] is set to 0 since no coins are needed to reach an amount of 0.
- **Dynamic Programming:**
  - For each amount a from 1 to the target amount, iterate through the list of coins.
  - If a coin can be used (i.e., a - c >= 0), update dp[a] by taking the minimum value between the current dp[a] and dp[a - c] + 1.
- **Result:**
  - Once all amounts are processed, we check if dp[amount] is still equal to amount + 1 (indicating no valid combination of coins was found).
  - Return the number of coins needed if a solution is found, or -1 if it's not possible to make the amount with the given coins.

COMPLEXITY

- **Time Complexity: O(n * m)**
  - **n is the amount.**
  - **m is the number of coins.**

- **Space Complexity: O(n)**

---

4) You are given two non-empty lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contain a single digit. Add the two numbers and return it as a list.

Example:
input:
l1 = [2, 4, 3]
l2 = [5, 6, 4]
Output: [7, 0, 8]
Explanation: 342 + 465 = 807.

SOLUTION

```
def add_two_numbers():
    list_1 = list(map(int, input().split()))
    list_2 = list(map(int, input().split()))

    result = []
    carry = 0
    i = 0
    j = 0

    while i < len(list_1) or j < len(list_2) or carry:
```

```
        num_1 = list_1[i] if i < len(list_1) else 0
        num_2 = list_2[j] if j < len(list_2) else 0

        total = num_1 + num_2 + carry
        carry = total // 10
        result.append(total % 10)

        i += 1
        j += 1

    return result

print(add_two_numbers())
```

OUTPUT

```
PS D:\HEXAVARSITY\DomainTraining>
2 4 3
5 6 4
[7, 0, 8]
PS D:\HEXAVARSITY\DomainTraining>
9 9 9
8 8 8
[7, 8, 8, 1]
PS D:\HEXAVARSITY\DomainTraining>
6 4 5
5 8 7
[1, 3, 3, 1]
```

EXPLANATION

- **Input**: The function takes two lists of digits from the user, where each list represents a number in reverse order (e.g., [2, 4, 3] represents the number 342).
- **Initialization**: It sets up an empty list result to store the sum and a carry variable to handle values greater than 9.
- **Adding Numbers**: The function loops through both lists and adds corresponding digits, including the carry from previous additions. If one list is shorter, it treats missing digits as 0.
- **Updating Carry**: After adding, the function updates the carry (if the sum is 10 or more) and stores the last digit of the sum in the result.
- **Output**: Once all digits and the carry are processed, it returns the result, which is the sum in reverse order.

COMPLEXITY

- **Time Complexity: O(n + m)**
  - **n is the length of list_1.**
  - **m is the length of list_2.**

- **Space Complexity: O(n + m)**
  - **n is the length of list_1.**
  - **m is the length of list_2.**

---

5) You are given an array of `k` lists, each list is sorted in ascending order. Merge all the linked lists into one sorted list and return it.

Example:
input:
lists = [[1,4,5], [1,3,4], [2,6]]
Output: [1,1,2,3,4,4,5,6]

SOLUTION

```
def merge_two_lists(list_1, list_2):
    res = []
    i = 0
    j = 0

    while i < len(list_1) and j < len(list_2):

        if list_1[i] <= list_2[j]:
            res.append(list_1[i])
            i += 1
        else:
            res.append(list_2[j])
            j += 1

    res.extend(list_1[i:])
    res.extend(list_2[j:])

    return res

def solution(lists):
    if not lists:
        return []

    merged = lists[0]
```

```
    for i in range(1, len(lists)):
        merged = merge_two_lists(merged, lists[i])
    return merged

# test_1 = [[1, 4, 5], [1, 3, 4], [2, 6]]
# test_2 = [[1, 2, 3], [4, 5, 6]]
# test_3 = [[7, 8, 9], [1, 2, 3], [4, 5]]

# print(solution(test_1))
# print(solution(test_2))
# print(solution(test_3))
```

OUTPUT

```
# test_1 = [[1, 4, 5], [1, 3, 4], [2, 6]]
 PS D:\HEXAVARSITY\DomainTraining>
  [1, 1, 2, 3, 4, 4, 5, 6]
# test_2 = [[1, 2, 3], [4, 5, 6]]
 PS D:\HEXAVARSITY\DomainTraining>
  [1, 2, 3, 4, 5, 6]
# test_3 = [[7, 8, 9], [1, 2, 3], [4, 5]]
 PS D:\HEXAVARSITY\DomainTraining>
  [1, 2, 3, 4, 5, 7, 8, 9]
```

EXPLANATION

merge_two_lists function:

- This function merges two sorted lists into one sorted list.
- Initializes an empty result list res and two pointers i and j to traverse list_1 and list_2.
- While loop: Iterates through both lists until one is exhausted:
    - Compare the current elements from both lists.
    - Appends the smaller element to res.
    - Moves the pointer of the list from which the element was taken.
- Extend remaining elements: After the loop, the function appends any remaining elements from either list to res and returns the merged sorted list.

solution function:

- This function merges multiple sorted lists into one sorted list using the **merge_two_lists** function.
- If lists is empty, the function returns an empty list.
- Merge lists:
    - Initializes merged with the first list and then iteratively merges it with the remaining lists.
- After all lists are merged, the function returns the final merged sorted list.

COMPLEXITY

- **Time Complexity: O(n Log k) (Here k is the number of lists)**
- **Space Complexity: O(n)**
