CSA-0674 DESIGN AND ANALYSIS OF ALGORITHM

ASSIGNMENT (13.06.24)

KORNANA.GOTHAM

192311169

1.Height of Binary Tree After Subtree Removal Queries

You are given `root` theof a binary tree with `n` nodes. Each node is assigned a unique value from `1` to `n`. You `m`.You have to perform `m` are also given an array `queries` of size independent queries on the tree where in the `ith` query you do the following:

- Remove the subtree rooted at the node `queries[i]` with the valuefrom the tree. It is `queries[i]` guaranteed thatwill not be equal to the value of the root.

Return `answer` *of size* `m` *an arraywhere* `answer[i]` *is the height of the tree after performing thequery*. `ith`

Note:

- The queries are independent, so the tree returns to its initial state after each query.
- The height of a tree is the number of edges in the longest simple path from the root to some node in the tree.

Input: root = [5,8,9,2,1,3,7,4,6], queries = [3,2,4,8]
Output: [3,2,3,2]

File Edit Format Run Options Window Help

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def find_height(node):
    if not node:
        return 0
    left_height = find_height(node.left)
    right_height = find_height(node.right)
    return max(left_height, right_height) + 1

def remove_subtree(root, value):
    if not root:
        return None
    if root.left and root.left.val == value:
        root.left = None
    elif root.right and root.right.val == value:
        root.right = None
    else:
        remove_subtree(root.left, value)
        remove_subtree(root.right, value)
    return root

def height_after_removal(root, queries):
    answer = []
    for value in queries:
        root = remove_subtree(root, value)
        height = find_height(root)
        answer.append(height)
    return answer

# Example usage:
# Construct the binary tree
#      1
#     / \
#    2   3
#   / \
#  4   5
root = TreeNode(1)
root.left = TreeNode(2, TreeNode(4), TreeNode(5))
root.right = TreeNode(3)

queries = [2, 3]
print(height_after_removal(root, queries))  # Output: [1, 1]
```

Output:

```
== RESTART: C:/Users/bmahe/AppData/Local/Programs/
[2, 1]
```

2. Sort Array by Moving Items to Empty Space

You are given an integer `nums` `n` arrayof size containing each element from `0` to `n - 1` (inclusive). Each `n - 1` of the elements from `1` torepresents an item, and the element `0` represents an empty space.

In one operation, you can move any item to the empty `nums` space.is considered to be sorted if the numbers of all the items are in ascending order and the empty space is

```
File  Edit  Format  Run  Options  Window  Help
def min_operations_to_sort(nums):
    n = len(nums)
    zero_pos = nums.index(0)
    operations = 0

    for i in range(n):
        while nums[i] != i:
            target_pos = nums[i]
            # Swap the element at nums[i] with the element at zero_pos
            nums[target_pos], nums[zero_pos] = nums[zero_pos], nums[target_pos]
            zero_pos = target_pos  # Update the position of 0 to the target position
            operations += 1

    return operations

# Example usage:
nums = [4, 0, 2, 1, 3]
print(min_operations_to_sort(nums))  # Output: 4
```

Constraints:

- `n == nums.length`
- `2 <= n <= 105`
- `0 <= nums[i] < n`
- All the values of `nums` are unique.

3. Apply Operations to an Array

You are given a 0-indexed array `nums` of size `n` consisting of non-negative integers.You need to `n - 1` `ith` applyoperations to this array where, in `ith` element of`nums` theoperation (0-indexed), you will apply the following on the:

- If `nums[i] == nums[i + 1]`, then multiply `nums[i]` by `2` and set `nums[i + 1]` to `0`. Otherwise, you skip this operation.

After performing all the operations, shift all the `0`'s to the end of the array.

- For example, the `[1,0,2,0,0,1]` arrayafter shifting all its `0`'s to the end, is. `[1,2,1,0,0,0]`

Return *the resulting array*.Note that the operations are applied sequentially, not all at once.

-

```
File  Edit  Format  Run  Options  Window  Help
def apply_operations(nums):
    n = len(nums)

    # Apply the operations
    for i in range(n - 1):
        if nums[i] == nums[i + 1]:
            nums[i] *= 2
            nums[i + 1] = 0

    # Shift all zeros to the end
    result = [num for num in nums if num != 0]  # Filter out non-zero e
    result.extend([0] * (n - len(result)))  # Append the appropriate nu

    return result

# Example usage:
nums = [2, 2, 0, 4, 4, 8]
print(apply_operations(nums))  # Output: [4, 4, 8, 0, 0, 0]
```

Output:

```
cc.py

= RESTART: C:/Users/gowth/App
cc.py
[4, 8, 8, 0, 0, 0]
>
```

4. Maximum Sum of Distinct Subarrays With Length K

You are given an integer array `nums` and an integer `k`. Find the maximum subarray sum of all the subarrays of `nums` that meet the following conditions:

- The length of the subarray is `k`, and
- All the elements of the subarray are distinct.

Return *the maximum subarray sum of all the subarrays that meet the conditions.* If no subarray meets the conditions, return `0`. *A subarray is a contiguous non-empty sequence of elements within an array.*

```
File  Edit  Format  Run  Options  Window  Help
def max_sum_of_distinct_subarrays(nums, k):
    if k > len(nums):
        return 0

    n = len(nums)
    max_sum = 0
    current_sum = 0
    start = 0
    seen = set()

    for end in range(n):
        while nums[end] in seen:
            seen.remove(nums[start])
            current_sum -= nums[start]
            start += 1

        seen.add(nums[end])
        current_sum += nums[end]

        if end - start + 1 == k:
            max_sum = max(max_sum, current_sum)
            seen.remove(nums[start])
            current_sum -= nums[start]
            start += 1

    return max_sum

# Example usage:
nums = [1, 2, 7, 3, 2, 4, 5]
k = 3
print(max_sum_of_distinct_subarrays(nums, k))  # Output: 14 (subarray [2, 7, 3] or [7, 3, 4])
```

Output:

```
= RESTART: C:/U
/Python312/ccc.
12
>>>
```

- 1 <= k <= nums.length <= 105
  1 <= nums[i] <= 105

5. Total Cost to Hire K Workers

You costs where costs[i] is the cost of hiring the ith
are given a 0-indexed integer array and candidates . W want to hire exactly k
worker.You are also given two integers k
workers according to the following rules:

```
File  Edit  Format  Run  Options  Window  Help
import heapq

def total_cost_to_hire_k_workers(costs, k, candidates):
    n = len(costs)

    if candidates * 2 >= n:
        # All workers should be considered
        all_workers = sorted([(cost, i) for i, cost in enumerate(costs)])
        return sum(cost for cost, _ in all_workers[:k])

    left_heap = [(costs[i], i) for i in range(candidates)]
    right_heap = [(costs[n - 1 - i], n - 1 - i) for i in range(candidates)]

    heapq.heapify(left_heap)
    heapq.heapify(right_heap)

    left_index = candidates
    right_index = n - 1 - candidates

    total_cost = 0

    for _ in range(k):
        if left_heap and (not right_heap or left_heap[0] <= right_heap[0]):
            cost, idx = heapq.heappop(left_heap)
            total_cost += cost
            if left_index <= right_index:
                heapq.heappush(left_heap, (costs[left_index], left_index))
                left_index += 1
        else:
            cost, idx = heapq.heappop(right_heap)
            total_cost += cost
            if right_index >= left_index:
                heapq.heappush(right_heap, (costs[right_index], right_index))
                right_index -= 1

    return total_cost

# Example usage:
```

Output:

```
= RESTART: C:/Users/c
a/Local/Programs/Pyth
2/ccc.py
9
>
```

6. Minimum Total Distance Traveled

There are some robots and factories on the X-axis. You are given an integer array robot where robot[i] is the position of the ith robot. You are also given a 2D integer array factory where factory[j] = [positionj, positionj limitj] indicates thatis the position of the jth factory and that the jth limitj factory can repair at mostrobots.

```
File  Edit  Format  Run  Options  Window  Help
def min_total_distance_traveled(robot, factory):
    # Sort robots and factories by their positions
    robot.sort()
    factory.sort()

    # Keep track of total distance and the current index in factory
    total_distance = 0
    factory_index = 0

    # List to track the number of robots repaired by each factory
    factory_limits = [0] * len(factory)

    for r in robot:
        # Move to the next available factory if the current one is full
        while factory_limits[factory_index] >= factory[factory_index][1]:
            factory_index += 1

        # Calculate the distance from the robot to the factory
        distance = abs(r - factory[factory_index][0])
        total_distance += distance

        # Update the number of robots repaired by this factory
        factory_limits[factory_index] += 1

    return total_distance

# Example usage:
robot = [0, 4, 6]
factory = [[2, 2], [6, 2]]
print(min_total_distance_traveled(robot, factory))  # Output: 4
```

Output:

```
>
= RESTART: C:/Use
a/Local/Programs/P
2/ccc.py
4
>
```

7. Minimum Subarrays in a Valid Split

You are given an integer `nums`.Splitting of an integer array `nums` arrayinto subarrays is valid if:

- the *greatest common divisor* of the first and last elements of each subarray is greater than `1`, and
- each element `nums` ofbelongs to exactly one subarray.

Return *the minimum number of* *subarrays in a valid subarray* `nums` *splitting of*. If a valid subarray splitting is not possible, return `-1`.

Note that:

- The greatest common divisor of two numbers is the largest positive integer that evenly divides both numbers.

```
File  Edit  Format  Run  Options  Window  Help
import math

def min_subarrays_in_valid_split(nums):
    n = len(nums)

    # Edge case for single element
    if n == 1:
        return -1 if nums[0] == 1 else 1

    # dp array to store the minimum number of subarrays to split nums[0:i+1]
    dp = [float('inf')] * n
    dp[0] = 0  # Base case: 0 splits needed for the first element

    # Function to calculate gcd
    def gcd(x, y):
        while y:
            x, y = y, x % y
        return x

    # Fill the dp array
    for i in range(n):
        for j in range(i, n):
            if gcd(nums[i], nums[j]) > 1:
                if i == 0:
                    dp[j] = 1
                else:
                    dp[j] = min(dp[j], dp[i - 1] + 1)

    # Result is the minimum number of subarrays to split the whole array
    return dp[-1] if dp[-1] != float('inf') else -1

# Example usage:
nums = [2, 6, 3, 4, 8]
print(min_subarrays_in_valid_split(nums))  # Output: 2
```

Output:

```
= RESTART: C:/Use
a/Local/Programs/
2/ccc.py
1
>>
```

- 1 <= nums.length <= 1000
  1 <= nums[i] <= 105

8. Number of Distinct Averages

You are given a 0-indexed integer `nums` arrayof even length.

As long `nums` asis not empty, you must repetitively:

- Find the minimum number in `nums` and remove it.
- Find the maximum number in `nums` and remove it.
- Calculate the average of the two removed numbers.

The average of two numbers `a` and `b` `(a + b) / 2` is.

(2 + 3) / 2 = 2.5

```
File  Edit  Format  Run  Options  Window  Help
def number_of_distinct_averages(nums):
    # Step 1: Sort the array
    nums.sort()

    # Step 2: Initialize pointers and a set to track distinct averages
    left, right = 0, len(nums) - 1
    distinct_averages = set()

    # Step 3: Iterate through the array using two pointers
    while left < right:
        avg = (nums[left] + nums[right]) / 2
        distinct_averages.add(avg)
        left += 1
        right -= 1

    # Step 4: Return the number of distinct averages
    return len(distinct_averages)

# Example usage:
nums = [1, 2, 3, 4, 5, 6]
print(number_of_distinct_averages(nums))  # Output: 3
```

output:

```
a/Local/Programs/
2/ccc.py
1
>>
```

- 2 <= nums.length <= 100
- nums.length is even.
- 0 <= nums[i] <= 100

9. Count Ways To Build Good Strings

Given the integers one , low , and high zero,, we can construct a string by starting with an empty string, and then at each step perform either of the following:

- Append the character '0' zero times.
- Append the character '1' one times.

This can be performed any number of times.A good string is a string constructed by the above process low and high having a length between(inclusive).

Return *the number of different good strings that can be constructed satisfying these properties.* Since the answer can be large, return it 109 + 7 modulo.

Example 1:

Input: low = 3, high = 3, zero = 1, one = 1
Output: 8
Explanation:
One possible valid good string is "011".
It can be constructed as follows: "" -> "0" -> "01" -> "011".

```
File  Edit  Format  Run  Options  Window  Help
def count_good_strings(zero, one, low, high):
    MOD = 10**9 + 7

    # Step 1: Initialize the dp array
    dp = [0] * (high + 1)
    dp[0] = 1  # Base case: one way to create an empty string

    # Step 2: Fill the dp array
    for i in range(1, high + 1):
        if i - zero >= 0:
            dp[i] = (dp[i] + dp[i - zero]) % MOD
        if i - one >= 0:
            dp[i] = (dp[i] + dp[i - one]) % MOD

    # Step 3: Calculate the result by summing the number of ways for lengths between low and high
    result = 0
    for i in range(low, high + 1):
        result = (result + dp[i]) % MOD

    return result

# Example usage:
zero = 1
one = 2
low = 2
high = 3
print(count_good_strings(zero, one, low, high))  # Output: 5
```

Output:

```
a/Local/Progra
2/ccc.py
5
>>
```

10. Most Profitable Path in a Tree

There is an undirected tree with n nodes labeled from 0 to n - 1, rooted at node 0. You are given a 2D integer array edges of n - 1 lengthwhere edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree.

At every node i, there is a gate. You are also given an array of even integers, amount where amount[i] represents:

- the price needed to open the gate at node i, if amount[i] is negative, or, ● the cash reward obtained on opening the gate at node i, otherwise.

```
File  Edit  Format  Run  Options  Window  Help
def most_profitable_path(n, edges, amount):
    from collections import defaultdict

    # Step 1: Build the graph
    graph = defaultdict(list)
    for a, b in edges:
        graph[a].append(b)
        graph[b].append(a)

    # Step 2: Initialize variables for DFS
    max_profit = float('-inf')

    def dfs(node, parent, current_sum):
        nonlocal max_profit
        current_sum += amount[node]

        # If it's a leaf node, update max_profit
        if len(graph[node]) == 1 and node != 0:  # node != 0 ensures it's not the root with only one conne
            max_profit = max(max_profit, current_sum)
            return

        # Recursively call DFS for all children
        for neighbor in graph[node]:
            if neighbor != parent:
                dfs(neighbor, node, current_sum)

    # Step 3: Start DFS from the root node (0)
    dfs(0, -1, 0)

    return max_profit

# Example usage:
n = 5
edges = [[0, 1], [0, 2], [1, 3], [1, 4]]
amount = [-2, 4, 2, 1, 3]
print(most_profitable_path(n, edges, amount))  # Output: 7
```

Output:

```
a/Local/Programs,
2/ccc.py
5
>
```