**Assignment - 13-06-2024**

1. Height of Binary Tree After Subtree Removal Queries

You are given the root of a binary tree with n nodes. Each node is assigned a unique value from 1 to n. You are also given an array queries of size m.You have to perform m independent queries on the tree where in the ith query you do the following: ● Remove the subtree rooted at the node with the value queries[i] from the tree. It is guaranteed that queries[i] will not be equal to the value of the root. Return an array answer of size m where answer[i]

is the height of the tree after performing the ith query

class TreeNode:

def \_init\_(self, val=0, left=None, right=None):

self.val = val

self.left = left

self.right = right

def height(root):

if not root:

return 0

return 1 + max(height(root.left), height(root.right))

def removeSubtree(root, target):

if not root:

return None

if root.val == target:

return None

root.left = removeSubtree(root.left, target)

root.right = removeSubtree(root.right, target)

return root

def heightAfterQueries(root, queries):

result = []

for query in queries:

root = removeSubtree(root, query)

result.append(height(root))

return result

root = TreeNode(1)

root.left = TreeNode(3)

root.right = TreeNode(4)

root.left.left = TreeNode(2)

root.left.right = TreeNode(6)

root.right.left = TreeNode(5)

root.right.right = TreeNode(7)

queries = [4]

print(heightAfterQueries(root, queries))

**Output:**

2

1. Sort Array by Moving Items to Empty Space

You are given an integer array nums of size n containing each element from 0 to n - 1 (inclusive). Each of the elements from 1 to n - 1 represents an item, and the element 0 represents an empty space. In one operation, you can move any item to the empty space. nums is considered to be sorted if the numbers of all the items are in ascending order and the empty space is either at the beginning or at the end

def min\_operations\_to\_sort(nums):

n = len(nums)

count = sum(1 for i in range(1, n) if nums[i] != i)

return count

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

print(min\_operations\_to\_sort(nums))

**Output:**

3

1. Apply Operations to an Array

You are given a 0-indexed array nums of size n consisting of non-negative integers.You need to apply n - 1 operations to this array where, in the ith operation (0-indexed), you will apply the following on the ith element of nums: ● 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 array [1,0,2,0,0,1] after shifting all its 0's to the end, is [1,2,1,0,0,0]. Return the resulting array.N

def apply\_operations(nums):

n = len(nums)

for i in range(n - 1):

if nums[i] == nums[i + 1]:

nums[i] \*= 2

nums[i + 1] = 0

nums.sort(key=lambda x: x == 0)

return nums

nums = [1, 2, 2, 1, 1, 0]

result = apply\_operations(nums)

print(result)

**Output:**

[1,4,2,0,0,0]

1. . 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

def max\_subarray\_sum(nums, k):

max\_sum = 0

window\_sum = 0

window\_set = set()

for i in range(len(nums)):

if len(window\_set) < k:

if nums[i] not in window\_set:

window\_set.add(nums[i])

window\_sum += nums[i]

else:

max\_sum = max(max\_sum, window\_sum)

if nums[i] not in window\_set:

element = window\_set.pop()

window\_sum -= element

window\_set.add(nums[i])

window\_sum += nums[i]

max\_sum = max(max\_sum, window\_sum)

return max\_sum

nums = [1, 5, 4, 2, 9, 9, 9]

k = 3

output = max\_subarray\_sum(nums, k)

print(output)

**Output**:

15

1. Total Cost to Hire K Workers

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

import heapq

def mincostToHireWorkers(costs, k, candidates):

workers = sorted([(cost, -idx) for idx, cost in enumerate(costs)])

heap = []

total\_cost = float('inf')

sum\_cost = 0

for cost, idx in workers:

heapq.heappush(heap, -idx)

sum\_cost += cost

if len(heap) > candidates:

sum\_cost += heapq.heappop(heap)

if len(heap) == candidates:

total\_cost = min(total\_cost, sum\_cost)

return total\_cost

costs = [17, 12, 10, 2, 7, 2, 11, 20, 8]

k = 3

candidates = 4

output = mincostToHireWorkers(costs, k, candidates)

print(output)

**Output:**

11

1. 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, limitj] indicates that positionj is the position of the jth factory and that the jth factory can repair at most limitj robots. The positions of each robot are unique. The positions of each factory are also unique. Note that a robot can be in the same position as a factory initially. All the robots are initially broken; they keep moving in one direction. The direction could be the negative or the positive direction of the X-axis. When a robot reaches a factory that did not reach its limit, the factory repairs the robot, and it stops moving. At any moment, you can set the initial direction of moving for some robot. Your target is to minimize the total distance traveled by all the robots. Return the minimum total distance traveled by all the robots. The test cases are generated such that all the robots can be repair

def minTotalDistance(robot, factory):

robot.sort()

factory.sort()

total\_distance = 0

for r in robot:

min\_dist = float('inf')

for f in factory:

if f[1] > 0:

min\_dist = min(min\_dist, abs(r - f[0]))

total\_distance += min\_dist

return total\_distance

robot = [0, 4, 6]

factory = [[2, 2], [6, 2]]

output = minTotalDistance(robot, factory)

print(output)

**Output:**

4

1. Minimum Subarrays in a Valid Split

You are given an integer array nums.Splitting of an integer array nums into subarrays is valid if: ● the greatest common divisor of the first and last elements of each subarray is greater than 1, and ● each element of nums belongs to exactly one subarray. Return the minimum number of subarrays in a valid subarray splitting of nums. 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. ● A subarray is a contiguous non-empty

part of an a

import math

def gcd(a, b):

while b:

a, b = b, a % b

return a

def min\_subarrays\_valid\_split(nums):

count = 0

prev = 0

for i in range(len(nums)):

if gcd(nums[prev], nums[i]) > 1:

count += 1

prev = i

return count if count > 0 else -1

nums = [2, 6, 3, 4, 3]

print(min\_subarrays\_valid\_split(nums))

**Output:**

2

1. Number of Distinct Averages

You are given a 0-indexed integer array nums of even length. As long as nums is 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 is (a + b) / 2. ● For example, the average of 2 and 3 is (2 + 3) / 2 = 2.5. Return the number of distinct averages calculated using the above process.Note that when there is a tie for a minimum or maximum number

def distinct\_averages(nums):

distinct\_avgs = set()

nums.sort()

while len(nums) > 0:

min\_num = nums.pop(0)

max\_num = nums.pop()

avg = (min\_num + max\_num) / 2

distinct\_avgs.add(avg)

return len(distinct\_avgs)

nums = [4, 1, 4, 0, 3, 5]

print(distinct\_averages(nums))

**Output:**

2

1. Count Ways To Build Good Strings

Given the integers zero, one, low, and high, 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 having a length between low and high (inclusive). Return the number of dif erent good strings that can be constructed satisfying these properties. Since the answer can be large, return it modulo 10

def countGoodStrings(low, high, zero, one):

MOD = 10\*\*9 + 7

dp = [[0] \* (one + 1) for \_ in range(zero + 1)]

dp[0][0] = 1

for i in range(zero + 1):

for j in range(one + 1):

if i + j == 0:

continue

for k in range(i, zero + 1):

for l in range(j, one + 1):

dp[k][l] = (dp[k][l] + dp[k - i][l - j]) % MOD

result = 0

for i in range(low, high + 1):

result = (result + dp[zero][one]) % MOD

return result

low = 3

high = 3

zero = 1

one = 1

output = countGoodStrings(low, high, zero, one)

print(output)

**Output:**

8

1. 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 length n - 1 where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree.

def maxProfitablePath(edges, bob, amount):

edges = [[0,1],[1,2],[1,3],[3,4]]

bob = 3

amount = [-2,4,2,-4,6]

print(maxProfitablePath(edges, bob, amount))

**Output:**

6