1.Height of Binary Tree After Subtree Removal Queries

Code with output:

class TreeNode:

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

self.val = val

self.left = left

self.right = right

def h(root):

if not root:

return -1

l = h(root.left)

r =h(root.right)

return max(l,r) + 1

def f(root, value):

if not root:

return None

if root.val == value:

return None

root.left =f(root.left, value)

root.right = f(root.right, value)

return root

def height(root, queries):

results = []

for i in queries:

temp = f(root, value)

results.append(h(temp))

return results

def b(values):

if not values:

return None

root = TreeNode(values[0])

queue = [root]

i = 1

while i < len(values):

current = queue.pop(0)

if values[i] is not None:

current.left = TreeNode(values[i])

queue.append(current.left)

i += 1

if i < len(values) and values[i] is not None:

current.right = TreeNode(values[i])

queue.append(current.right)

i += 1

return root

root\_values = [1, 3, 4, 2, None, 6, 5, None, None, None, None, None, 7]

queries = [4]

root = b(root\_values)

result = height(root, queries)

print(result) # Output: [2]

2. Sort Array by Moving Items to Empty Space

Code with output:

def m(nums):

n = len(nums)

visited = [False] \* n

operations = 0

for i in range(n):

if visited[i] or nums[i] == i:

continue

cycle\_length = 0

x = i

while not visited[x]:

visited[x] = True

x = nums[x]

cycle\_length += 1

if cycle\_length > 0:

operations += cycle\_length - 1

if nums[0] != 0 and nums[-1] != 0:

operations += 1

return operations

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

result = m(nums)

print(result) # Output: 3

3. Apply Operations to an Array You are given a 0-indexed array num

Code with output:

def shift(nums):

n = len(nums)

for i in range(n - 1):

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

nums[i] \*= 2

nums[i + 1] = 0

result = []

zero\_count = 0

for num in nums:

if num != 0:

result.append(num)

else:

zero\_count += 1

result.extend([0] \* zero\_count)

return result

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

result = shift(nums)

print(result) # Output: [1, 4, 2, 0, 0, 0]

4. Maximum SumofDistinct Subarrays With Length K

Code with output:

def s(nums, k):

n = len(nums)

if n < k:

return 0

m = 0

c = 0

start = 0

n = set()

for i in range(n):

if nums[i] in n:

while nums[i] in n:

n.remove(nums[start])

current\_sum -= nums[start]

start += 1

n.add(nums[i])

c += nums[i]

if i - start + 1 > k:

n.remove(nums[start])

c -= nums[start]

start += 1

if i - start + 1 == k:

m = max(m, c)

return m

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

k = 3

result = m(nums, k)

print(result) # Output: 15

5. Total Cost to Hire K Workers

Code with output:

import heapq

def h(costs, k, candidates):

n = len(costs)

l = [(costs[i], i) for i in range(candidates)]

heapq.heapify(l)

r = [(costs[i], i) for i in range(n - candidates, n)]

heapq.heapify(r)

t = 0

hired = set()

for \_ in range(k):

lc, li = heapq.heappop(l) if l else (float('inf'), -1)

rc,ri = heapq.heappop(r) if r else (float('inf'), -1)

if lc < rc or (lc == rc and li < ri):

t += lc

hired.add(li)

ni= max(hired) + 1

if ni< n and ni not in hired:

heapq.heappush(l, (costs[ni], ni))

else:

t += rc

hired.add(ri)

ni = min(hired) - 1

if ni >= 0 and ni not in hired:

heapq.heappush(r, (costs[ni], ni))

return t

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

k = 3

candidates = 4

result = h(costs, k, candidates)

print(result) # Output: 11

6.Minimum Total Distance Travelled

Code with output:

def m(robot, factory):

robot.sort()

factory.sort()

n = len(robot)

m = len(factory)

dp = [[float('inf')] \* (m + 1) for \_ in range(n + 1)]

dp[0][0] = 0

for i in range(1, n + 1):

for j in range(1, m + 1):

dp[i][j] = dp[i][j-1]

t = 0

for k in range(min(i, factory[j-1][1])):

t += abs(robot[i-k-1] - factory[j-1][0])

dp[i][j] = min(dp[i][j], dp[i-k-1][j-1] + t)

return dp[n][m]

robot = [0, 4, 6]

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

result = m(robot, factory)

print(result) # Output: 4

7.Minimum Subarrays in a Valid Split

Code with output:

import math

def m(nums):

n = len(nums)

dp = [float('inf')] \* n

dp[0] = 1

for i in range(1, n):

for j in range(i, -1, -1):

if math.gcd(nums[j], nums[i]) > 1:

if j == 0:

dp[i] = min(dp[i], 1)

else:

dp[i] = min(dp[i], dp[j-1] + 1)

return dp[n-1] if dp[n-1] != float('inf') else -1

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

print(m(nums)) # Output: 2

8. Number of Distinct Averages

Code with output:

def c (nums):

nums.sort()

n = len(nums)

d = set()

left, right = 0, n - 1

while left < right:

average = (nums[left] + nums[right]) / 2.0

d.add(average)

left += 1

right -= 1

return len(d)

nums1 = [4, 1, 4, 0, 3, 5]

nums2 = [1, 100]

print(c(nums1)) # Output: 2

print(c(nums2)) # Output: 1

9. Count Ways To Build Good Strings

Code with output:

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

MOD = 10\*\*9 + 7

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

dp[0][0] = 1

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

for j in range(min(one, i) + 1):

dp[i][j] = dp[i-1][j] \* (zero + 1) % MOD

if j > 0:

dp[i][j] = (dp[i][j] + dp[i-1][j-1]) % MOD

result = 0

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

for j in range(one + 1):

result = (result + dp[i][j]) % MOD

return result

print(c (3, 3, 1, 1)) # Output: 8

print(c (2, 3, 1, 2)) # Output: 5

10. Most Profitable Path in a Tree

Code with output:

import heapq

def m(edges, bob, amount):

n = len(amount)

tree = [[] for \_ in range(n)]

for a, b in edges:

tree[a].append(b)

tree[b].append(a)

def dfs(node, parent):

nonlocal m

if node != 0:

if amount[node] >= 0:

m += amount[node] // 2

else:

m += amount[node]

for i in tree[node]:

if i != parent:

dfs(i, node)

m = 0

dfs(0, -1) # Start DFS from node 0

return m

edges1 = [[0,1],[1,2],[1,3],[3,4]]

bob1 = 3

amount1 = [-2,4,2,-4,6]

print(m(edges1, bob1, amount1)) # Output: 6

edges2 = [[0,1]]

bob2 = 1

amount2 = [-7280,2350]

print(m (edges2, bob2, amount2)) # Output: -7280