CSA0674 - DAA ASSIGNMENT 6

1) HEIGHT OF BINARY TREE AFTER SUBTREE REMOVAL QUERIES :

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def treeHeightAfterQueries(root, queries):
    def height(node):
        if not node: return 0
        return 1 + max(height(node.left), height(node.right))
    def remove_subtree(node, val):
        if not node: return None
        if node.val == val: return None
        node.left = remove_subtree(node.left, val)
        node.right = remove_subtree(node.right, val)
        return node
    results = []
    for q in queries:
        temp_root = remove_subtree(root, q)
        results.append(height(temp_root) - 1)
    return results
root = TreeNode(1, TreeNode(3, TreeNode(2)), TreeNode(4, TreeNode(6), TreeNode(5, None, TreeNode(7))))
queries = [4]
print(treeHeightAfterQueries(root, queries))
```

2) SORT ARRAY BY MOVING ITEMS TO EMPTY SPACE :

```python
def minOperationsToSort(nums):
    empty = nums.index(0)
    operations = 0
    for i in range(len(nums)):
        while nums[i] != i and nums[i] != 0:
            nums[empty], nums[i] = nums[i], 0
            empty = i
            operations += 1
    return operations
nums = [4, 2, 0, 3, 1]
print(minOperationsToSort(nums))
```

3) APPLY OPERATIONS TO AN ARRAY :

```
def applyOperations(nums):
    for i in range(len(nums) - 1):
        if nums[i] == nums[i + 1]:
            nums[i], nums[i + 1] = nums[i] * 2, 0
    return [num for num in nums if num] + [0] * nums.count(0)
nums = [1, 2, 2, 1, 1, 0]
print(applyOperations(nums))
```

4) MAXIMUM SUM OF DISTINCT SUBARRAYS WITH LENGTH K :

```
def maxDistinctSubarraySum(nums, k):
    max_sum, curr_sum, count = 0, 0, {}
    for i in range(len(nums)):
        if nums[i] in count: count[nums[i]] += 1
        else: count[nums[i]] = 1
        curr_sum += nums[i]
        if i >= k:
            count[nums[i - k]] -= 1
            if count[nums[i - k]] == 0: del count[nums[i - k]]
            curr_sum -= nums[i - k]
        if len(count) == k: max_sum = max(max_sum, curr_sum)
    return max_sum
nums = [1, 5, 4, 2, 9, 9, 9]
k = 3
print(maxDistinctSubarraySum(nums, k))
```

5) TOTAL COST TO HIRE K WORKERS :

```python
def minCostToHireWorkers(costs, k, candidates):
    from heapq import heappop, heappush, heapify
    first_k = costs[:candidates]
    last_k = costs[-candidates:]
    heap = [(cost, i) for i, cost in enumerate(first_k + last_k)]
    heapify(heap)
    total_cost, hired = 0, set()
    while k > 0:
        cost, index = heappop(heap)
        if index in hired: continue
        hired.add(index)
        total_cost += cost
        k -= 1
    return total_cost
costs = [17, 12, 10, 2, 7, 2, 11, 20, 8]
k = 3
candidates = 4
print(minCostToHireWorkers(costs, k, candidates))
```

6) MINIMUM TOTAL DISTANCE TRAVELLED :

```python
def minTotalDistance(robot, factory):
    robot.sort()
    factory.sort()
    total_distance = 0
    for i in range(len(robot)):
        total_distance += abs(robot[i] - factory[i][0])
    return total_distance
robot = [0, 4, 6]
factory = [[2, 2], [6, 2]]
print(minTotalDistance(robot, factory))
```

7) MINIMUM SUBARRAYS IN A VALID SPLIT :

```python
def minSubarrays(nums):
    from math import gcd
    n, last_gcd, subarrays = len(nums), 0, 0
    for i in range(n):
        current_gcd = gcd(last_gcd, nums[i])
        if current_gcd == 1:
            subarrays += 1
            last_gcd = 0
        last_gcd = gcd(last_gcd, nums[i])
    return subarrays
nums = [2, 6, 3, 4, 3]
print(minSubarrays(nums))
```

8) NUMBER OF DISTINCT AVERAGES :

```python
def distinctAverages(nums):
    seen = set()
    while nums:
        min_num, max_num = min(nums), max(nums)
        nums.remove(min_num)
        nums.remove(max_num)
        seen.add((min_num + max_num) / 2)
    return len(seen)
nums = [4, 1, 4, 0, 3]
print(distinctAverages(nums))
```

9) COUNT WAYS TO BUILD GOOD STRINGS :

```python
def countGoodStrings(low, high, zero, one):
    MOD = 10**9 + 7
    dp = [0] * (high + 1)
    dp[0] = 1
    for length in range(1, high + 1):
        if length >= zero:
            dp[length] += dp[length - zero]
        if length >= one:
            dp[length] += dp[length - one]
        dp[length] %= MOD
    return sum(dp[low:high+1]) % MOD
low, high, zero, one = 3, 3, 1, 1
print(countGoodStrings(low, high, zero, one))  # Output: 8
low, high, zero, one = 2, 3, 1, 2
print(countGoodStrings(low, high, zero, one))  # Output: 5
```

10) MOST PROFITABLE PATH IN A TREE :

```python
def maxNetIncome(n, edges, amount, bob):
    from collections import defaultdict, deque
    tree = defaultdict(list)
    for u, v in edges:
        tree[u].append(v)
        tree[v].append(u)
    def find_path(node):
        path = []
        stack = [(0, [])]
        visited = set()
        while stack:
            cur, p = stack.pop()
            if cur == node:
                return p + [cur]
            if cur not in visited:
                visited.add(cur)
                for neighbor in tree[cur]:
                    stack.append((neighbor, p + [cur]))
        return []
    bob_path = find_path(bob)
    bob_path_set = set(bob_path)
    def dfs(node, parent, depth, income):
        nonlocal max_income
        if node not in bob_path_set or bob_path[depth] != node:
            new_income = income + amount[node]
        else:
            new_income = income + amount[node] // 2
        max_income = max(max_income, new_income)
        for neighbor in tree[node]:
            if neighbor != parent:
                dfs(neighbor, node, depth + 1, new_income)
    max_income = float('-inf')
    dfs(0, -1, 0, 0)
    return max_income
n = 7
edges = [[0, 1], [1, 2], [1, 3], [2, 4], [2, 5], [3, 6]]
amount = [0, 10, -10, 20, -10, 30, -10]
bob = 4
```