Assignment7(13/6/24)

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. Note: ● Thequeries are independent, so the tree returns to its initial state after each query. ● Theheight of a tree is the number of edges in the longest simple path from the root to some node in the tree.**

   Code:

```
class Node:
	def __init__(self, data):
		self.data = data
		self.left = None
		self.right = None
def maxDepth(node):
	if node is None:
		return 0

	else:
		lDepth = maxDepth(node.left)
		rDepth = maxDepth(node.right)
		if (lDepth > rDepth):
			return lDepth+1
		else:
			return rDepth+1

root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
print("Height of tree is %d" % (maxDepth(root)))
```

   output:

```
>>
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/height of subtree.
py
Height of tree is 3
>>
```

2. **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 of the array. For example, if n = 4, nums is sorted if:

● nums = [0,1,2,3] or nums = [1,2,3,0] ...and considered to be unsorted otherwise.Return the minimum number of operations needed to sort nums.

Example 1:

Input: nums = [4,2,0,3,1]
Output: 3

Code:

```
def sort(arr, n):
  i = 0
  while(i < n):
    correct = arr[i]-1
    if arr[correct] != arr[i]:
      swap(arr, i, correct)
    else:
      i = i + 1
def swap(arr, first, second):
  temp = arr[first]
  arr[first] = arr[second]
  arr[second] = temp
arr = [3, 2, 5, 6, 1, 4]
n = len(arr)
sort(arr, n)
for i in range(0, n):
  print(arr[i], end=" ")
```

output:

```
>>
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/sort an array.py
1 2 3 4 5 6
>>
```

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 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.Note that the operations are applied sequentially, not all at once.

code:

```
def minOp (arr, n) :
        sm = sum(arr)
        small = min(arr)
        minOperation = sm - (n * small)
        return minOperation
arr = [5, 6, 2, 4, 3]
n = len(arr)
print( "Minimum Operation = ", minOp (arr, n))
```

output:

```
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/minnimum opration.
py
Minimum Operation =  10
```

4.Maximum SumofDistinct 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 ● Allthe 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.

Example 1:

Input: nums = [1,5,4,2,9,9,9], k = 3

Output: 15

Code:

```
from collections import defaultdict
def helper(arr, k):
        mp = defaultdict(int)
        currentSum = 0
        maxSum = 0
        n = len(arr)
        left = 0
        i = 0
        while i < k and i < n:
                currentSum += arr[i]
                mp[arr[i]] += 1

                i += 1
        if len(mp) == k:
                maxSum = currentSum
        for i in range(k, n):
                mp[arr[i]] += 1
                mp[arr[left]] -= 1
                if mp[arr[left]] == 0:
                        del mp[arr[left]]
                currentSum += arr[i]
```

```
            currentSum -= arr[left]
            if len(mp) == k:
                maxSum = max(maxSum, currentSum)
            left += 1
        return maxSum
if __name__ == "__main__":
    arr = [1, 5, 4, 2, 9, 9, 9]
    k = 3
    print(helper(arr, k))
```

output:

```
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/max sum in subarr
y.py
15
```

5. 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 rules:

● You will run k sessions and hire exactly one worker in each session.

● Ineachhiring session, choose the worker with the lowest cost from either the first candidates workers or the last candidates workers. Break the tie by the smallest index. ○ ○ For example, if costs = [3,2,7,7,1,2] and candidates = 2, then in the first hiring session, we will choose the 4th worker because they have the lowest cost [3,2 ,7,7,1,2].

Code:

```
def solve(days, costs, i, validity, N):
    if i >= N:
        return 0

    if days[i] <= validity:
        return solve(days, costs, i + 1, validity, N)
    else:
        ch1 = costs[0] + solve(days, costs, i + 1, days[i], N)
        ch2 = costs[1] + solve(days, costs, i + 1, days[i] + 6, N)
        ch3 = costs[2] + solve(days, costs, i + 1, days[i] + 29, N)
        return min(ch1, min(ch2, ch3))
def MinCost(days, cost, N):
    return solve(days, cost, 0, 0, N)
if __name__ == '__main__':
    arr = [2, 4, 6, 7, 8, 10, 17]
    cost = [3, 8, 20]
    N = len(arr)
    print(MinCost(arr, cost, N))
```

output:

```
>
==== RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/min cost.py ==
14
```

6. MinimumTotalDistanceTraveled ThEREaresomerobotsandfactoriesontheX-axis.Youaregivenanintegerarrayrobot whererobot[i]isthepositionoftheithrobot.Youarealsogivena2Dintegerarrayfactory wherefactory[j]=[positionj,limitj]indicatesthatpositionjisthepositionofthejthfactory andthatthejthfactorycanrepairatmostlimitjrobots. Thepositionsofeachrobotareunique.Thepositionsofeachfactoryarealsounique.Note thatarobotcanbeinthesamepositionasafactoryinitially

Code:

```
def minimum_total_distance(robot, factory):
    robot.sort()
    factory.sort()
    total_distance = 0
    robot_index = 0
    n = len(robot)
    m = len(factory)
    for i in range(m):
        factory_position, factory_limit = factory[i]
        while factory_limit > 0 and robot_index < n:
            total_distance += abs(factory_position - robot[robot_index])
            robot_index += 1
            factory_limit -= 1

        if robot_index == n:
            break

    return total_distance
robots = [1, 3, 6]
factories = [[2, 2], [5, 1]]
print(minimum_total_distance(robots, factories))
```

output:

```
>
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/total distance.py
3
>
```

7. .MinimumSubarraysinaValidSplit Youaregivenanintegerarraynums.Splittingofanintegerarraynumsintosubarraysisvalid if: ●

thegreatestcommondivisorofthefirstandlastelementsofeachsubarrayisgreater than1,and ● eachelementofnumsbelongstoexactlyonesubarray.
Returntheminimumnumberofsubarraysinavalidsubarraysplittingofnums.Ifavalid subarraysplittingisnotpossible,return-1.

Code:

```
def is_partition_possible(arr, N, K):


    pre = [0] * (N + 1)


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

        pre[i] = arr[i - 1]


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

        pre[i] = pre[i] + pre[i - 1]

    dp = [[0] * (N + 1) for _ in range(N + 1)]

    for i in range(N + 1):

        dp[i][i] = 1

    for size in range(1, N + 1):

        for i in range(1, N - size + 2):

            j = i + size - 1

            for k in range(i, j):

                if ((k - i == 0 or pre[k] - pre[i - 1] >= K) and

                        (j - k - 1 == 0 or pre[j] - pre[k] >= K)):

                    dp[i][j] = dp[i][j] | (dp[i][k] & dp[k + 1][j])

    return dp[1][N]

if __name__ == "__main__":

    # Input

    N = 5

    K = 6

    arr = [2, 3, 3, 2, 3]
```

```
print(is_partition_possible(arr, N, K))
```

output:

```
= RESTART: C:/Users/Neda Anjum/Documents/1lab experiments daa/min subarray valid
.py
1
```

8. 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

Code:

```
import math as mt
def countWindowDistinct(win, K):

    dist_count = 0
    for i in range(K):
        j = 0
        while j < i:
            if (win[i] == win[j]):
                break
            else:
                j += 1
        if (j == i):
            dist_count += 1

    return dist_count
def countDistinct(arr, N, K):
```

```
    for i in range(N - K + 1):
        print(countWindowDistinct(arr[i:K + i], K)
if __name__=='__main__':
 arr = [1, 2, 1, 3, 4, 2, 3]
K = 4
N = len(arr)
countDistinct(arr, N, K)
```

output:

```
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/count distinct ele
ment.py
3
4
4
3
```

9.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

Code:

```
def transform(A, B):
    if len(A) != len(B):
        return -1
    m = {}
    n = len(A)
    for i in range(n):
        if A[i] in m:
            m[A[i]] += 1
        else:
            m[A[i]] = 1
    for i in range(n):
        if B[i] in m:
```

```
            m[B[i]] -= 1
    for key in m:
        if m[key] != 0:
            return -1
        i, j = n-1, n-1
    res = 0
    while i >= 0 and j >= 0:
        while i >= 0 and A[i] != B[j]:
            res += 1
            i -= 1
        i -= 1
        j -= 1

    return res
A = "EACBD"
B = "EABCD"
print("Minimum number of opera", transform(A, B))
```

output:

```
== RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/count string.py
Minimum number of operations required is 3
```

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 length n- 1 where 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.**

Code:

```
def most_profitable_path(n, edges, amount):
```

```
    from collections import defaultdict
    tree = defaultdict(list)
    for a, b in edges:
        tree[a].append(b)
        tree[b].append(a)
    max_profit = float('-inf')
    def dfs(node, parent, current_profit):
        nonlocal max_profit
        current_profit += amount[node]
        if len(tree[node]) == 1 and node != 0:
            max_profit = max(max_profit, current_profit)
        for neighbor in tree[node]:
            if neighbor != parent:
                dfs(neighbor, node, current_profit)
    dfs(0, -1, 0)

    return max_profit
n = 5
edges = [[0, 1], [0, 2], [1, 3], [1, 4]]
amount = [5, -3, 4, 2, -1]
print(most_profitable_path(n, edges, amount))
```

output:

```
== RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/profit path.py
9
>
```