Infosys SP 6 Jul 2024 asked Coding Question -HiringHustle

Question 3

A kingdom has N cities with M roads between them. You are given a 2D array E denoting that there is a road between the cities E[i][0] and E[i][1] with a distance of E[i] [2] for all Osis N-1.

Bob loves traveling very much and wants to travel the kingdom. He can start his trip from any city and end the trip in any city he wants. However, he can't visit the same city more than once.

You can perform the following operation exactly once:

- Take one road with its distance and delete this road. Add it in another place where there is no road, provided that all cities remain connected each other(you can add the road in the same place where it was removed).

Find the minimum distance of the longest trip t Bob can take after performing the above given operation.

Note:

- Its guaranteed that there is a path between every two cities.

Input Format

The first line contains an integer, N, denoting the number of cities in the kingdom.

The next line contains an integer, M, denoting the number of rows in E.

The next line contains an integer, three, denoting the number of columns in E.

Each line i of the M subsequent lines (where 0 ≤ i < M) contains three space separated integers each describing the row E[i].

Constraints

1 <= N <= 2000

N-1 <= M <= N-1

3 <= three <= 3

1 <= E[i][j] <= 10^5

Sample Test Cases

Case 1

Input:

2

1

3

1 2 2

Output:

2

Explanation:

Given N=2,, M = 1, three = 3, E= [[1, 2, 2]].

We can remove the road between cities 1, 2 and add it in the same place. So the minimum distance of the longest trip is 2.

Case 2

Input:

3

2

3

1 2 1

2 3 2

Output:

3

Given N=3, M=2 ,three=3,E=[[1,2,1],[2,3,2]]
We can remove the road between cities 2,3 and it between 1,
3.So the minimum distance of the longest trip is 3.

case 3
input
3
2
3
1 2 2
1 3 3
output
5
Given N=3, M = 2, three = 3, E= [[1,2,2],[1,3,3]]

We can remove the road between cities 1,2 and add 2,3. So the minimum distance of the longest trip is 5.

```
def find_min_longest_trip(N, M, E):
from collections import defaultdict
import heapq
graph = defaultdict(list)
edges = []

# Build the graph
for u, v, w in E:
    graph[u].append((v, w))
    graph[v].append((u, w))
    edges.append((w, u, v))

# Function to find MST using Prim's algorithm
def prim_mst():
```

```
    mst_cost = 0
    visited = [False] * (N + 1)
    min_heap = [(0, 1)]  # (cost, node)

    while min_heap:
        cost, node = heapq.heappop(min_heap)
        if visited[node]:
            continue
        visited[node] = True
        mst_cost += cost
        for neighbor, weight in graph[node]:
            if not visited[neighbor]:
                heapq.heappush(min_heap, (weight, neighbor))

    return mst_cost

# Compute MST cost
mst_cost = prim_mst()

# Find all edges not in MST
non_mst_edges = []
for w, u, v in edges:
    if w > mst_cost:
        non_mst_edges.append((w, u, v))

# If there are no edges not in MST, the result is the MST cos
t
if not non_mst_edges:
    return mst_cost

# Function to find minimum of maximum distances after modifyi
ng an edge
def min_longest_trip_after_modification():
    min_longest_trip = float('inf')

    for w, u, v in non_mst_edges:
```

```
        # Remove edge u-v from the graph temporarily
        graph[u] = [(nv, nw) for nv, nw in graph[u] if nv !=
v]
        graph[v] = [(nu, nw) for nu, nw in graph[v] if nu !=
u]

        # Calculate the maximum distance after this modificat
ion
        max_distance = 0
        visited = [False] * (N + 1)
        def dfs(node, distance):
            nonlocal max_distance
            visited[node] = True
            max_distance = max(max_distance, distance)
            for neighbor, weight in graph[node]:
                if not visited[neighbor]:
                    dfs(neighbor, distance + weight)

        # Start DFS from any node (here 1)
        dfs(1, 0)

        # Restore the graph
        graph[u].append((v, w))
        graph[v].append((u, w))

        # Update min_longest_trip
        min_longest_trip = min(min_longest_trip, max_distanc
e)

    return min_longest_trip

# Find the minimum of the maximum distances
result = min_longest_trip_after_modification()

return result
```

Example usage:

N = 3
M = 2
E = [
[1, 2, 2],
[1, 3, 3]
]

print(find_min_longest_trip(N, M, E)) # Output: 5

Coding Question 2

You are given an array A of size N.

For a non-continuous subsequence S of length K, the beauty is calculated as follows:

- If the length of the subsequence is 1 then the beauty is 0. pretty2020
- If the length of the subsequence is greater than 1 then the beauty is the sum of (S[i + 1] - S[i]) ^ 2 for all 1 <= i < k

Find the maximum possible beauty of a subsequence such that the GCD of the absolute values of S is greater than 1. Since the answer can be large, return it modulo 10 ^ 9 + 7

Input Format

The first line contains an integer, N, denoting the number of elements in A.

Each line i of the N subsequent lines (where 0 <= i < N) contains an integer describing A[i].

Constraints

1 <= N <= 10 ^ 5

- N <= A[i] <= N
  Sample Test Cases

Case 1

Input:
5
1
2
1
2
1
Output:

1

Explanation:

Given N=5, A= [1, 2, 1, 2, 1]

We can choose a subsequence as [1, 2], which consists of only two elements and satisfies the necessary conditions.

Hence, the beauty of this subsequence is equal to 0.

Case 2

Input:

5
5
3
4
2
1

Output:

4

Explanation:

Given N=5, A= [5, 3, 4, 2, 1]
We can choose a subsequence which consists of elements [4, 2].

The beauty of this subsequence is (4-2)^2 which is equal to 4.

Case 3

Input:

6

1

6

2

5

4

- 3

Output:

81

Explanation:

Given N= 6, A = [1, 6, 2, 5, 4, -3]

We can choose a subsequence which consists of elements [6, -3].
The beauty of this subsequence is (6 - (- 3)) ^ 2 which is equal to 81

```
def max_beauty_subsequence(N, A):
    MOD = 10**9 + 7

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

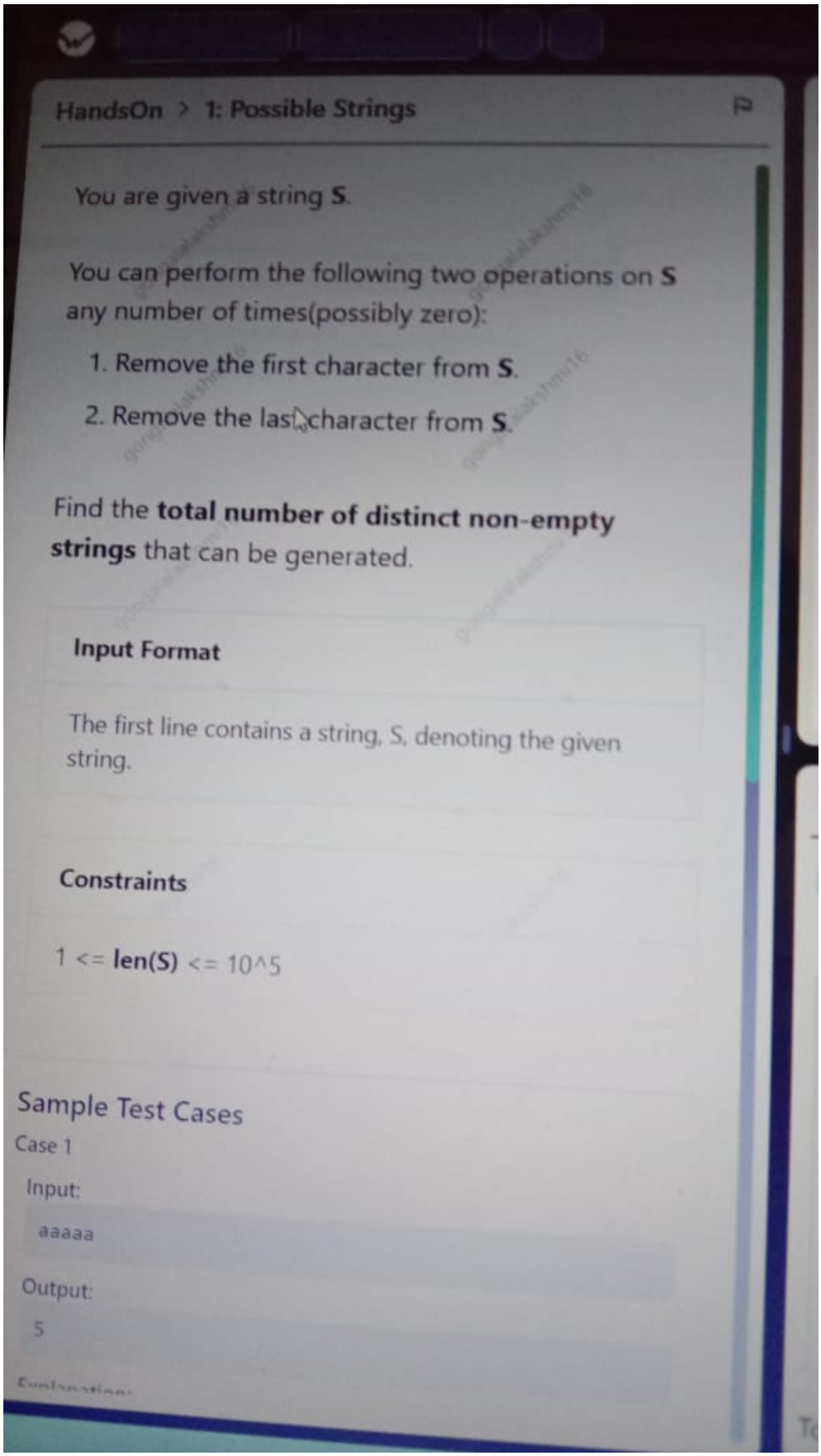
```
        return x

    max_beauty = 0

    # Check each possible subsequence
    for i in range(N):
        for j in range(i + 1, N):
            subseq = A[i:j+1]
            subseq_len = len(subseq)

            # Check GCD condition
            if subseq_len > 1:
                current_gcd = abs(subseq[0])
                for num in subseq[1:]:
                    current_gcd = gcd(current_gcd, abs(num))
                    if current_gcd > 1:
                        break

                if current_gcd > 1:
                    beauty = sum((subseq[k] - subseq[k-1])**2
for k in range(1, subseq_len))
                    max_beauty = max(max_beauty, beauty % MO
D)

    return max_beauty

# Example usage:
N = 5
A = [1, 2, 1, 2, 1]
print(max_beauty_subsequence(N, A))  # Output: 1
```

Example usage:

N = 5
A = [1, 2, 1, 2, 1]

```
print(max_beauty_subsequence(N, A))  # Output: 1
```

Question 1



HandsOn > 1: Possible Strings

You are given a string **S**.

You can perform the following two operations on **S** any number of times(possibly zero):

1. Remove the first character from **S**.
2. Remove the last character from **S**.

Find the **total number of distinct non-empty strings** that can be generated.

Input Format

The first line contains a string, S, denoting the given string.

Constraints

1 <= **len(S)** <= 10^5

Sample Test Cases

Case 1

Input:

aaaaa

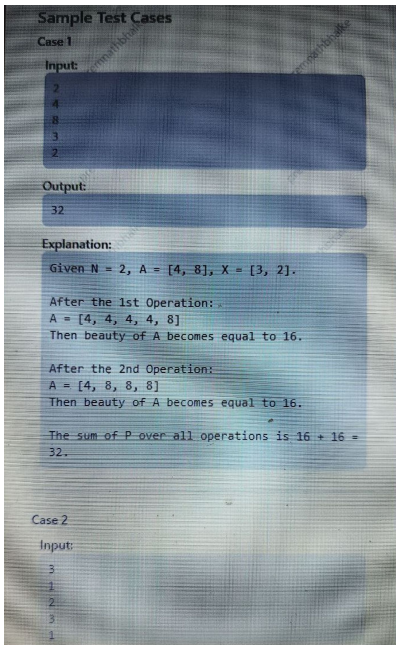
Output:

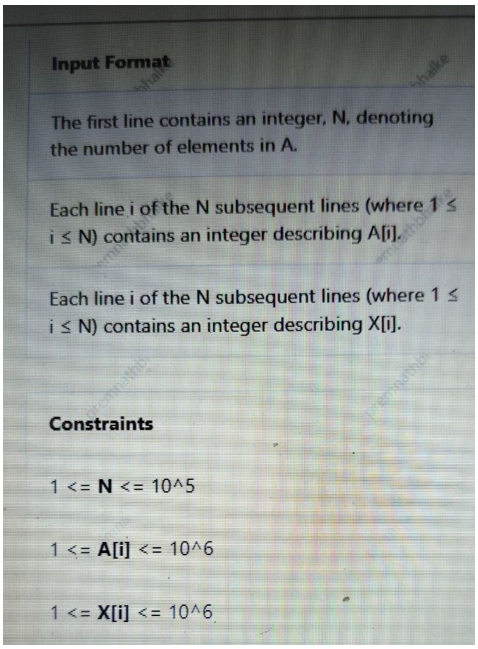
5

Code

```
Set<String> substrings = new HashSet<>();
        int n = S.length();

        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j <= n; j++) {
                substrings.add(S.substring(i, j));
            }
        }

        return substrings.size();
```

Question 4





Input Format

The first line contains an integer, N, denoting the number of elements in A.

Each line i of the N subsequent lines (where 1 ≤ i ≤ N) contains an integer describing A[i].

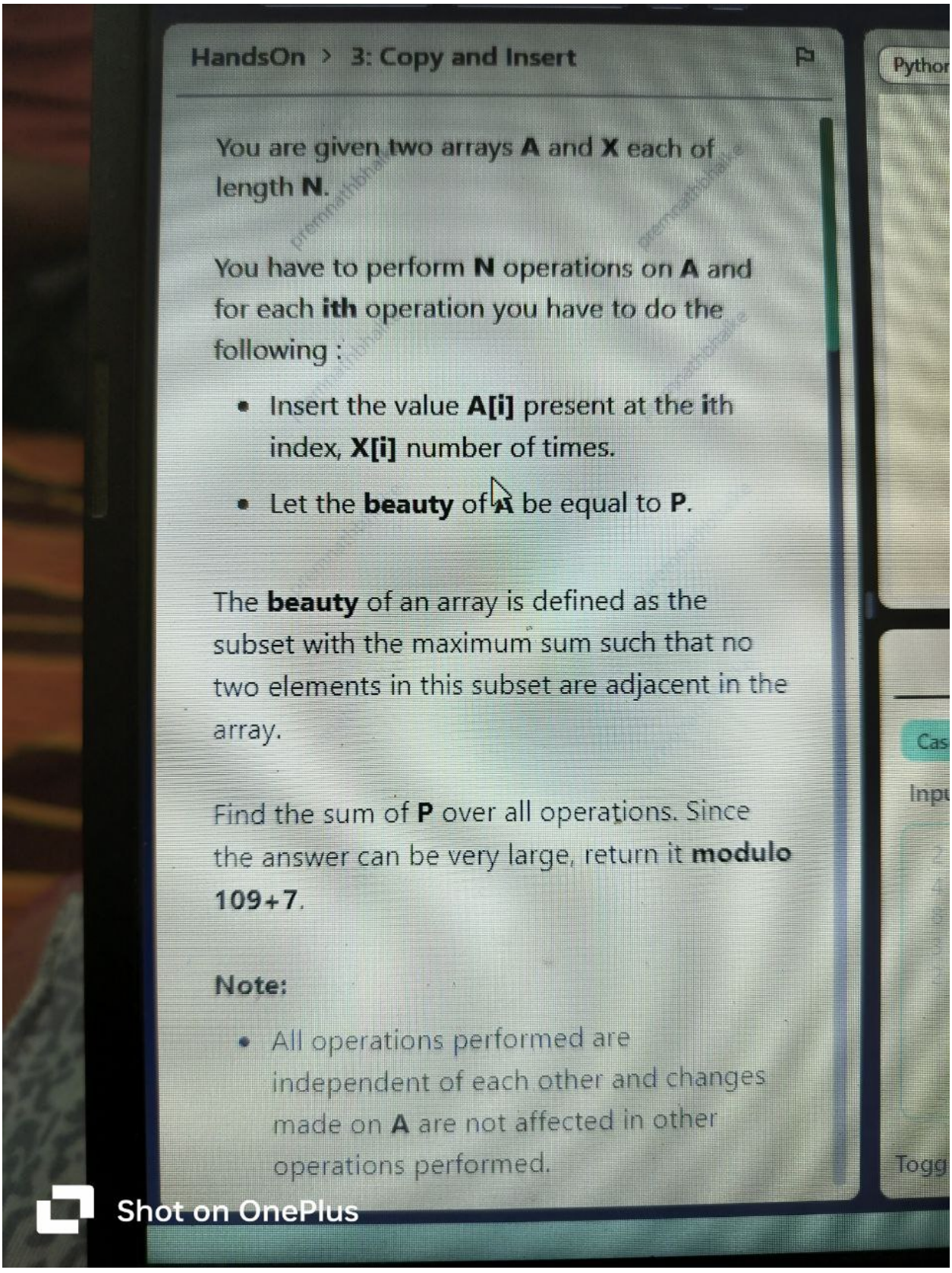
Each line i of the N subsequent lines (where 1 ≤ i ≤ N) contains an integer describing X[i].

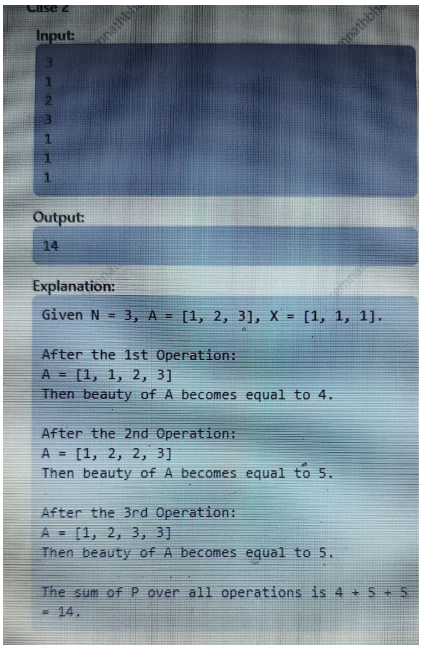
Constraints

1 <= **N** <= 10^5

1 <= **A[i]** <= 10^6

1 <= **X[i]** <= 10^6


HandsOn > 3: Copy and Insert
You are given two arrays A and X each of length N.
You have to perform N operations on A and for each ith operation you have to do the following :
Insert the value A[i] present at the ith index, X[i] number of times.
Let the beauty of A be equal to P.
The beauty of an array is defined as the subset with the maximum sum such that no two elements in this subset are adjacent in the array.
Find the sum of P over all operations. Since the answer can be very large, return it modulo 109+7.
Note:
All operations performed are independent of each other and changes made on A are not affected in other operations performed.
Shot on OnePlus


Case 2

Input:

```
3
1
2
3
1
1
1
```

Output:

```
14
```

Explanation:

```
Given N = 3, A = [1, 2, 3], X = [1, 1, 1].

After the 1st Operation:
A = [1, 1, 2, 3]
Then beauty of A becomes equal to 4.

After the 2nd Operation:
A = [1, 2, 2, 3]
Then beauty of A becomes equal to 5.

After the 3rd Operation:
A = [1, 2, 3, 3]
Then beauty of A becomes equal to 5.

The sum of P over all operations is 4 + 5 + 5
= 14.
```