Infosys SP 7 Jul 2024 asked Coding Question -HiringHustle and 6th Jul 2024 question below

Question 1

You are given an array A which consists of N integers.

Find the total number of unique subarray sum that are possible in A.

Note:

- A subarray is a contiguous part of an array.

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[1]

Constraints

1<= N <= 10^5

Sample Test Cases

Case 1

Input

5

7

2

3

2

2

Output: 9

Explanation:

Given N = 5, A [7, 2, 3, 2, 2]

The number of unique sums that can be obtained by taking the sum of elements in all possible subarrays of A are 9 which are [7], [9], [12], [14], [2], [5], [3], [4], [16].

Case 2

Input:

5

4

10

5

2

2

Output:

12

Explanation:

Given N = 5, A [4, 10, 5, 2, 2]

The number of unique sums that can be obtained by taking the sum of elements in all possible subarrays of A are 12.

Case 3

Input:

5

2

10

3

4

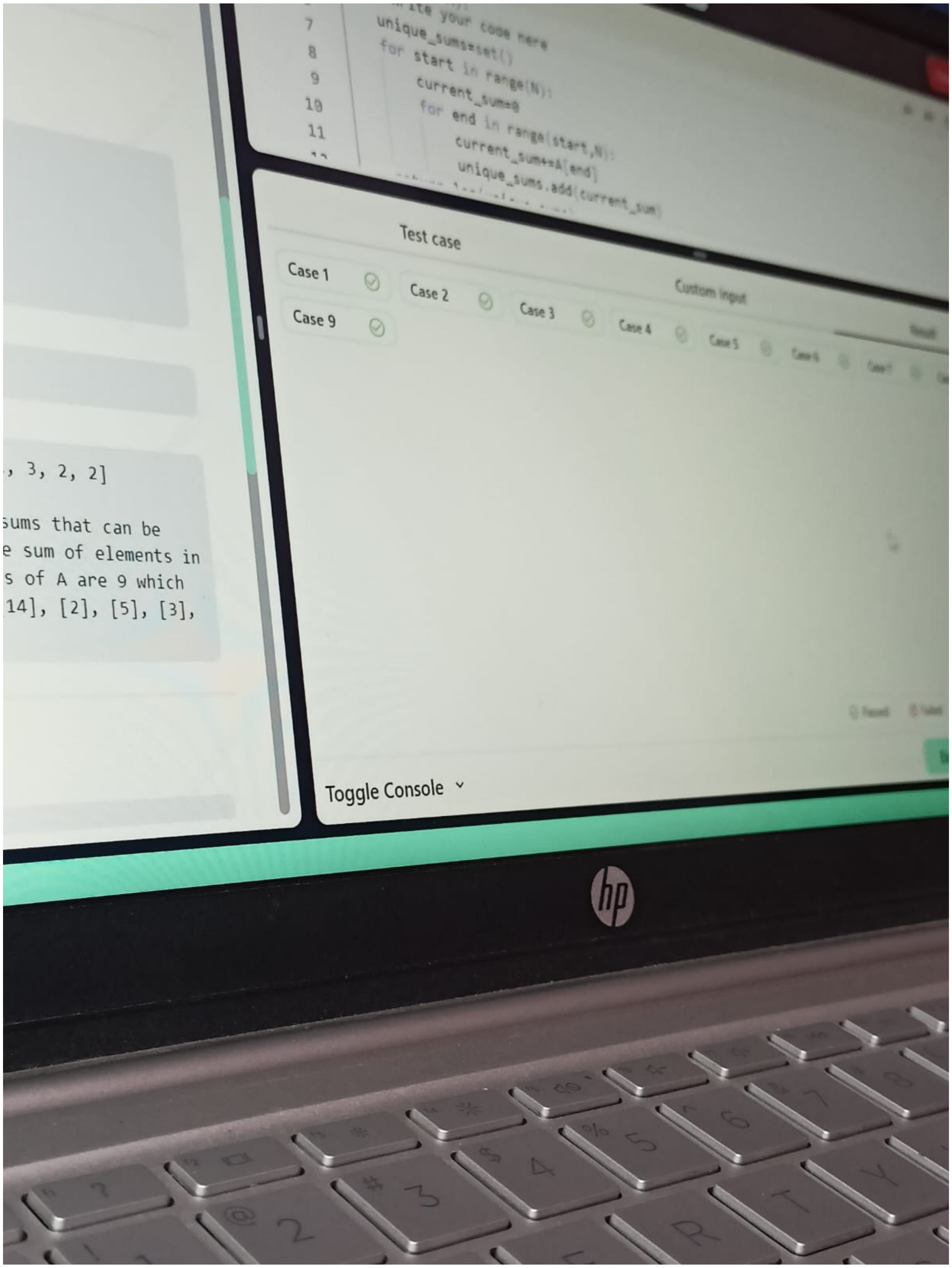
2

Output:

13

def solve(N,A)

```
def solve(N, A):
    unique_sums = set()

    for start in range(N):
        current_sum = 0
        for end in range(start, N):
            current_sum += A[end]
            unique_sums.add(current_sum)

    print(len(unique_sums))
```



```
ite your code here
unique_sums=set()
for start in range(N):
    current_sum=0
    for end in range(start,N):
        current_sum+=A[end]
        unique_sums.add(current_sum)
```

, 3, 2, 2]

sums that can be
e sum of elements in
s of A are 9 which
14], [2], [5], [3],

Question 2

You are given a tree which consists of N nodes and an array A where A[U] represents the value of node U.

You are also given an array P where P[U] denotes the parent of node U.

We define the beauty of the path from U to V as follows:

- Write down the values written on the nodes of the simple path from U to V in the order they appear.
- Find the length of the longest subsequence such that for each pair of adjacent elements, their ged is greater than 1.

Find the maximum possible beauty over all pairs of nodes.

Note:

- It is given that P[1] is equal to 0.

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]

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

Constraints

1 <= N <= 10 ^ 5

1 <= A[i] <= 10 ^ 5

1 <= P[i] <= 10 ^ 5

Sample Test Cases

Case 1

Input:

1

999

0

Output:

1

Q

Explanation:

Given N 1, A [999], P= [0].

There exist only 1 node present in the Tree.

Hence, the answer for this case is equal to 1.

Case 2

Input:

2

43636

46964

0

1

Output:

2

Explanation:

Given N = 2, A [43636, 46964], P = [0, 1].

Since the values of both nodes are even then the gcd of the path is greater than 1.

Hence the answer for this case is equal to 2.

Case 3

Input

4

55305

63807

73880

50840

0

1

1

1

Q

Output:

3

Explanation:

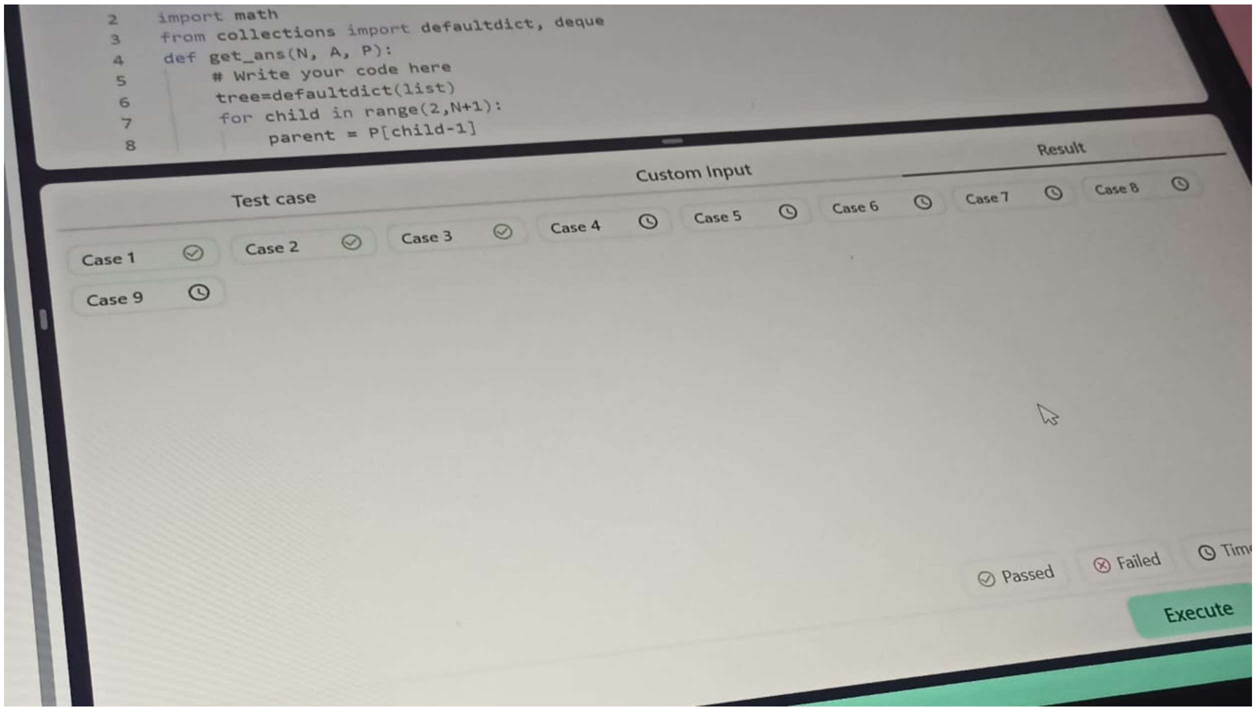
Given N = 4, A [55305, 63807, 73880, 50840], P= [0, 1, 1, 1].

The optimal path consists of nodes [3, 1, 4] which has a gcd greater than 1.

Hence, the answer for this case is equal to 3.

def get_ans(N,A,P)

```
import math
from collections import defaultdict, deque

def get_ans(N, A, P):
    tree = defaultdict(list)
    for child in range(2, N+1):
        parent = P[child - 1]
        tree[parent].append(child)
        tree[child].append(parent)

    def gcd(a, b):
        return math.gcd(a, b)

    def bfs(start):
        queue = deque([(start, [A[start-1]])])
        max_beauty = 1
        visited = set()
        visited.add(start)

        while queue:
            current, path = queue.popleft()
            for neighbor in tree[current]:
                if neighbor not in visited:
                    visited.add(neighbor)
                    new_path = path + [A[neighbor-1]]
                    queue.append((neighbor, new_path))

                    dp = [1] * len(new_path)
                    for i in range(1, len(new_path)):
                        for j in range(i):
                            if gcd(new_path[i], new_path[j])
> 1:
                                dp[i] = max(dp[i], dp[j] + 1)
                    max_beauty = max(max_beauty, max(dp))

        return max_beauty

    overall_max_beauty = 1
    for node in range(1, N+1):
        overall_max_beauty = max(overall_max_beauty, bfs(nod
e))

    return overall_max_beauty
#if u faced gcd eroor
def gcd(x, y):
        while y:
            x, y = y, x % y
        return x
```



```
import math
from collections import defaultdict, deque
def get_ans(N, A, P):
    # Write your code here
    tree=defaultdict(list)
    for child in range(2,N+1):
        parent = P[child-1]
```

Question 3

You are given an array A of length N, which initially contains of only 1's.

We call the subarray from L to R good, if for all pairs of elements in this range have product as a perfect square.

You are also given a 2D array Queries which consists of Q queries. Each query is one of the two types:

Q • Type 1-1 LR: Replace A[L] with R.

* Type 2 - 2 LR: Find the length of the longest good subarray contained in the

range from L to R.

Find the bitwise XOR of all queries of Type 2. Since the answer can be very large, return it modulo 10pow9+7.

Input Format

The first line contains an integer. N. denoting the size of array A

The next line contains an integer, Q. denoting the number of rows in Queries.

The next line contains an integer, Col, denoting the number of columns in Queries

Each ime i of the Q subsequent lines (where 0 <= i < Q) contains Col space separated integers each describing the row Queries[i].

Constraints

1 <= N <= 10 ^ 5

1 <= Q <= 10 ^ 5

3 <= col <= 3

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

Sample Test Cases

Case 1

Input:

5

5

3

213

211

111

112

222

Output:

3

Q

Explanation:

Given M = 5 Q = 5 three = 3, Q = I[2] 1, 3], [2, 1, 1], [1, 1, 1], [1, 1, 2], [2, 2, 2]].

After ist Query A = [1, 1, 1, 1, 1] , then length of longest good subarray in range [1, 3] is equal to 3.

After 2nd Query A = [1, 1, 1, 1, 1] , the length of the longest good subarray in range [1, 1] is equal to 1.

After 3rd Query A = [1, 1, 1, 1, 1]

After 4th Query A = [2, 1, 1, 1, 1]

After 5th Query A = [2, 1, 1, 1, 1] , the length of the longest good suharray in range

12. 21 is equal to 1.

The bitwise xon of answers of queries of type 2 is equal to 1 xom 1 XOR 11.

Case 2

Input
5
5
3

211

1164

12 64

215

Output

7

Q

Case

Input

Explanation

Given N5, Q5, three 1, Q ||2. 1, 3], [2, 1, 1], [1, 1, 64], [1, 2, 64], [2, 1, 5]].

After 1st Query A [1, 1, 1, 1, 1], then length of longest good subarray in range [1, 3] is equal to 3.

After 2nd Query A [1, 1, 1, 1, 1, the
After 2nd Query A = [1, 1, 1, 1, 1] the length of the longest good subarray in range [1, 1] is equal to 1.

After 3rd Query A = [64, 1, 1, 1, 1]

After 4th Query A = [64, 64, 1, 1, 1] .

After 5th Query A = [64, 64, 1, 1, 1] the length of the longest good subarray in range [1, 5] is equal to 5.

The bitwise XOR of answers of queries of type 2 is equal to 3 XOR 1 XOR 5 7.

Case 3

Input

3

5

3

112

132

213

212

223

Output:

1

Explanation:

Given N = 3, Q5, three = 3, Q = [[1, 1, 2], [1, 3, 2], [2, 1, 3], [2,, 2], [2, 2, 3]].

After the first 2 queries, the array is [2, 1, 2], and the answer's are: 1, 1, 1 and their xor is 1.

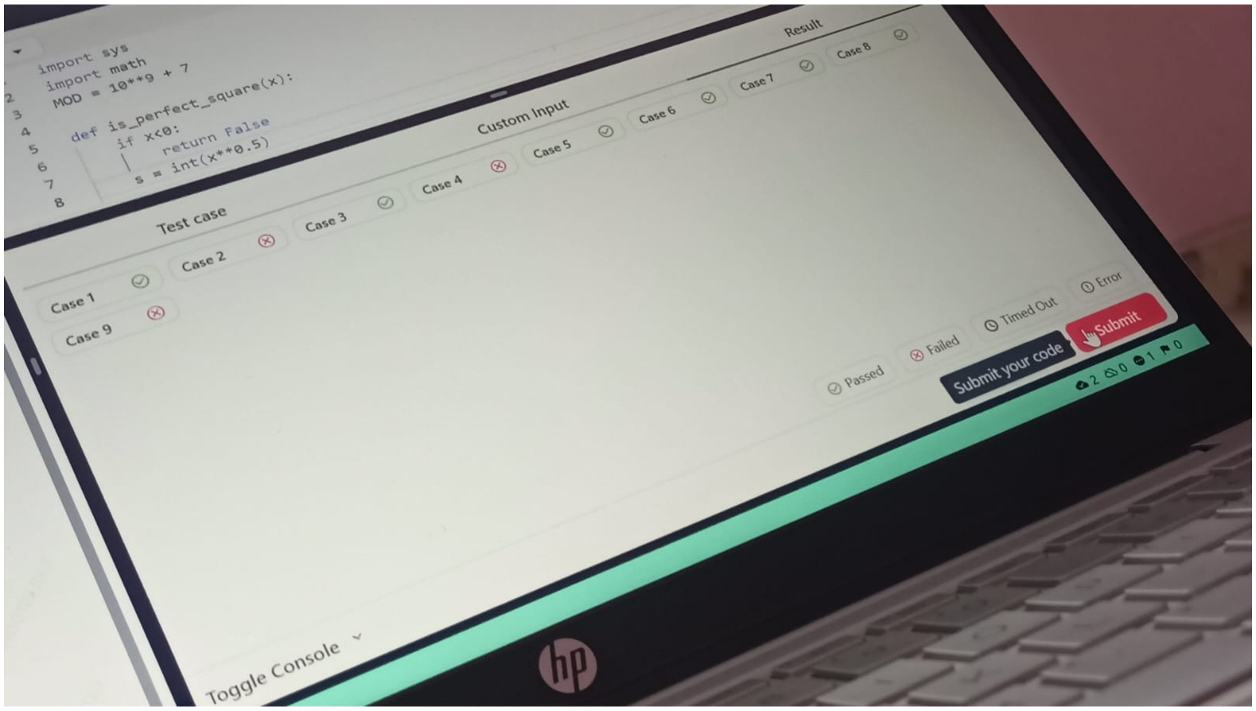
def get(N,Q,Col, Queries)

```
import math

MOD = 10**9 + 7

def is_perfect_square(x):
    s = int(math.isqrt(x))
    return s * s == x

def get(N, Q, Col, Queries):
    A = [1] * N
    xor_result = 0

    def find_longest_good_subarray(L, R):
        max_len = 0
        current_len = 0
        for i in range(L, R + 1):
            if is_perfect_square(A[i - 1]):
                current_len += 1
            else:
                max_len = max(max_len, current_len)
                current_len = 0
        max_len = max(max_len, current_len)
        return max_len

    for query in Queries:
        if query[0] == 1:
            L, R = query[1], query[2]
            A[L - 1] = R
        elif query[0] == 2:
            L, R = query[1], query[2]
            longest_good_length = find_longest_good_subarray
(L, R)
            xor_result ^= longest_good_length

    return xor_result % MOD
```

Sample test cases

N1, Q1, Col1, Queries1 = 5, 5, 3, [[2, 1, 3], [2, 1, 1], [1, 1, 1], [1, 1, 2], [2, 2, 2]]
N2, Q2, Col2, Queries2 = 5, 5, 3, [[2, 1, 3], [2, 1, 1], [1, 1, 64], [1, 2, 64], [2, 1, 5]]
N3, Q3, Col3, Queries3 = 3, 5, 3, [[1, 1, 2], [1, 3, 2], [2, 1, 3], [2, 2, 2], [2, 2, 3]]

print(get(N1, Q1, Col1, Queries1)) # Output: 3
print(get(N2, Q2, Col2, Queries2)) # Output: 7
print(get(N3, Q3, Col3, Queries3)) # Output: 1



```
import sys
import math
MOD = 10**9 + 7

def is_perfect_square(x):
    if x<0:
        return False
    s = int(x**0.5)
```

6 th july below Question

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]

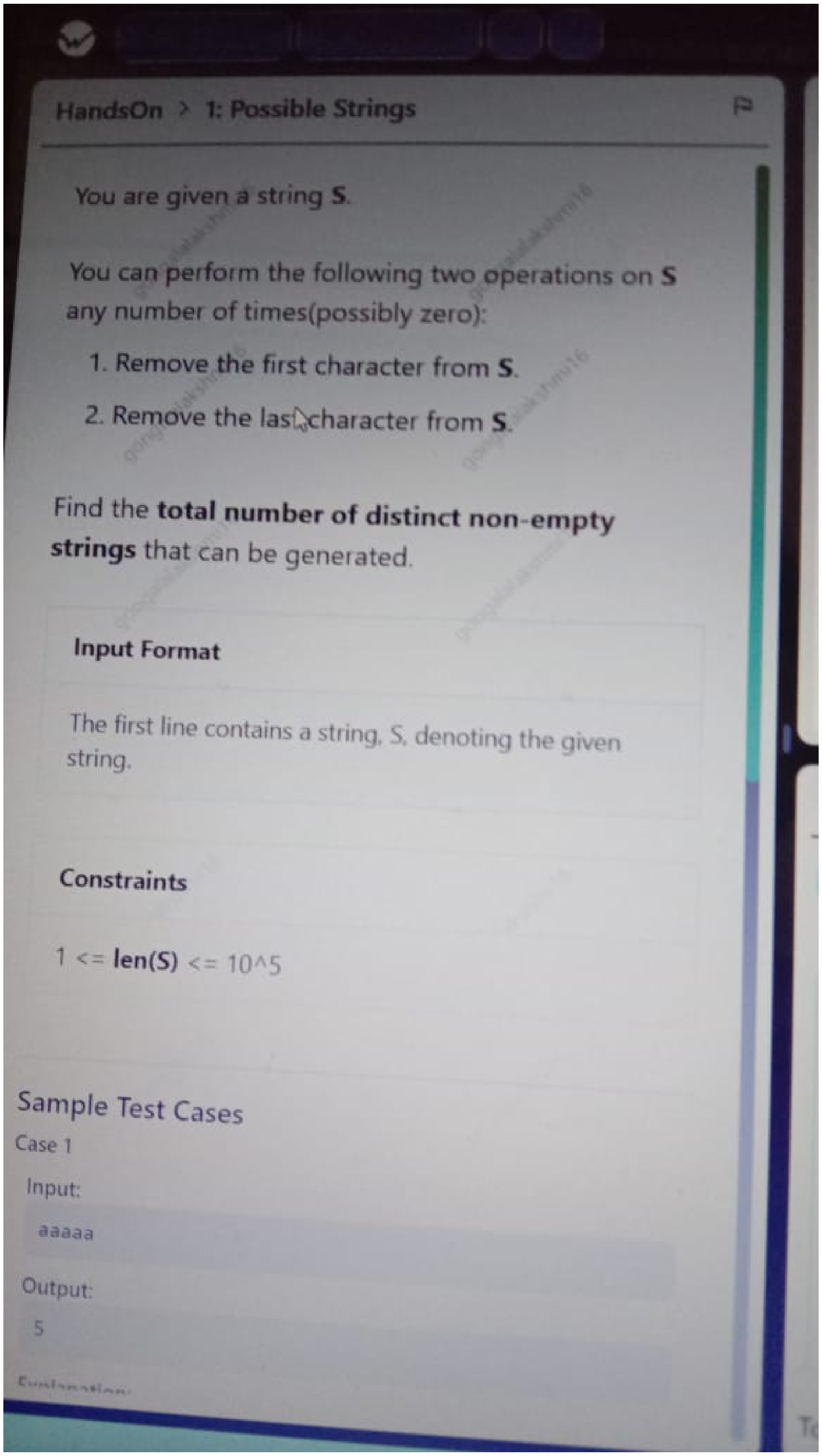
We can choose a subsequence which consists of elements [6, -3].
The beauty of this subsequence is (6 - (- 3)) ^ 2 which is equal to 81

```
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
```

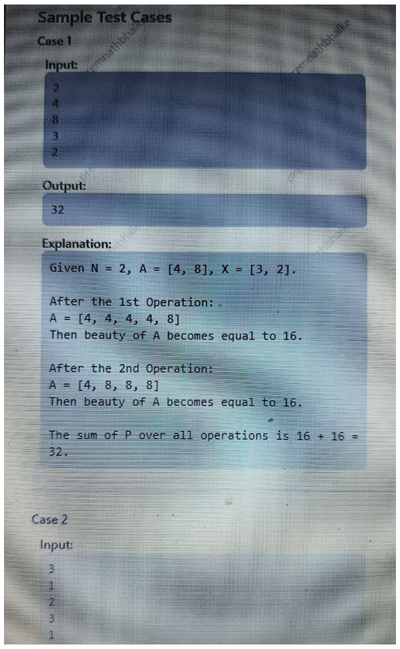
Output:

```
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



Sample Test Cases

Case 1

Input:

```
2
4
8
3
2
```

Output:

```
32
```

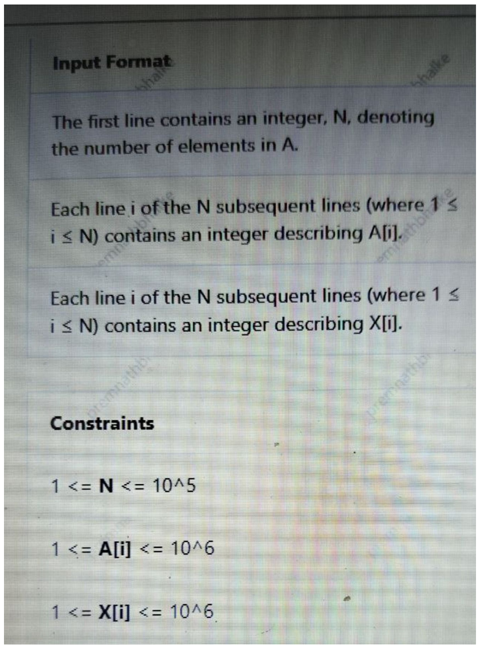
Explanation:

```
Given N = 2, A = [4, 8], X = [3, 2].

After the 1st Operation:
A = [4, 4, 4, 4, 8]
Then beauty of A becomes equal to 16.

After the 2nd Operation:
A = [4, 8, 8, 8]
Then beauty of A becomes equal to 16.

The sum of P over all operations is 16 + 16 =
32.
```

Case 2

Input:

```
3
1
2
3
1
```



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].

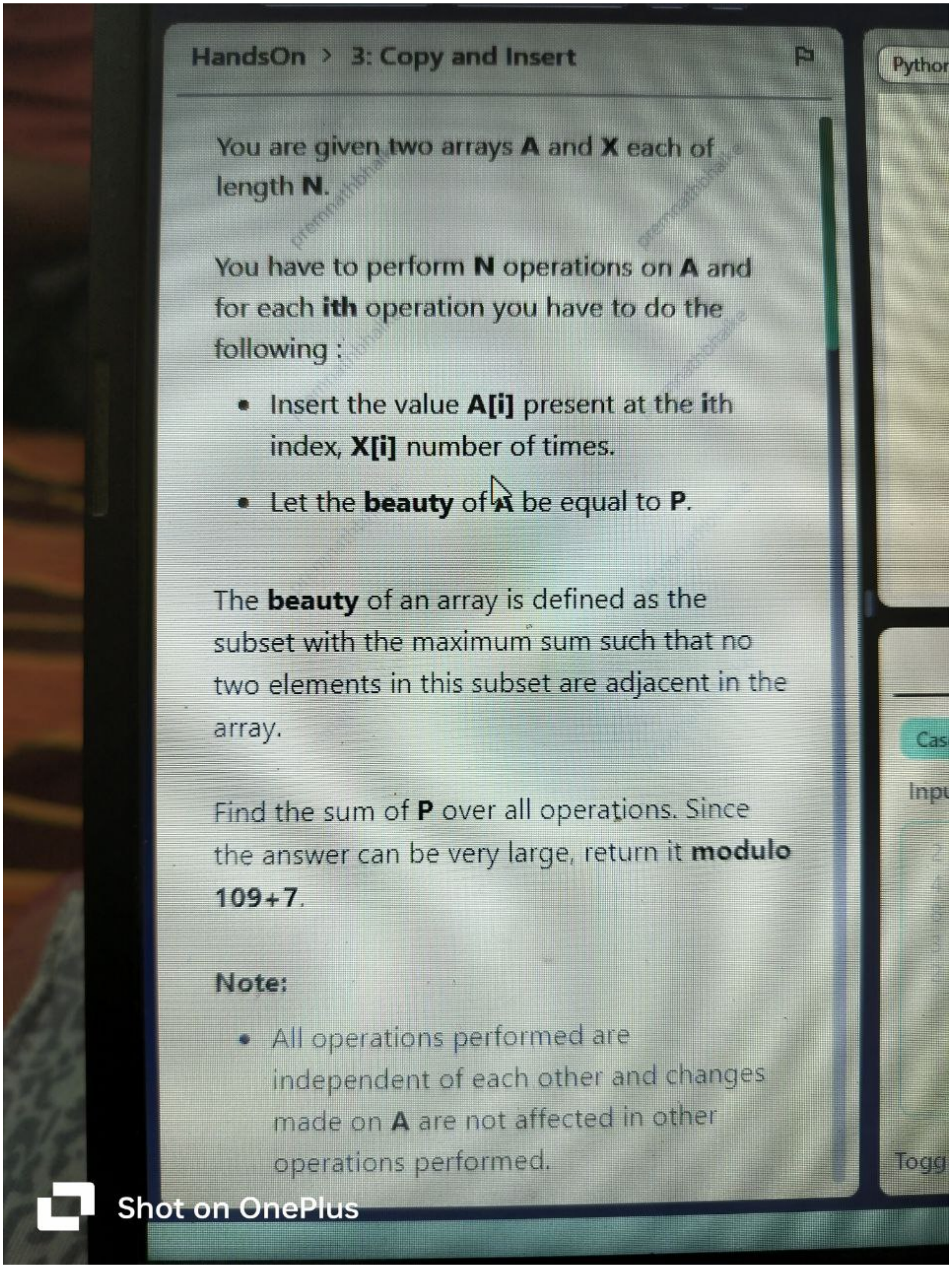
Each line i of the N subsequent lines (where 1 ≤ i ≤ N) contains an integer describing X[i].

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 :

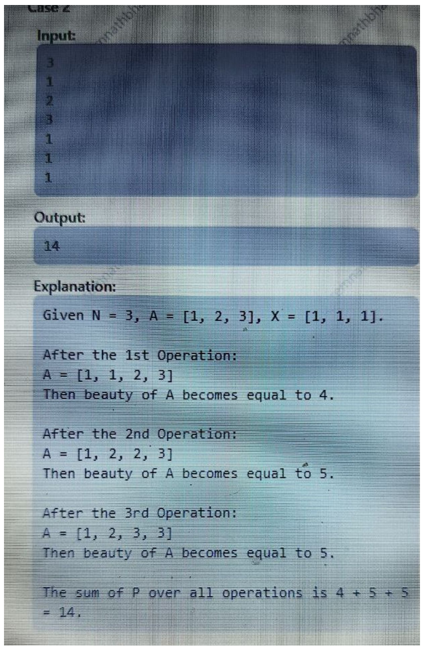
- Insert the value **A[i]** present at the **ith** index, **X[i]** number of times.
- Let the **beauty** of **A** be equal to **P**.

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.



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.
```