ACM Summer Challenge 2020

Standard Template Library

Editorial

House Painting

This question is an example of maintaining certain information with the use of unordered maps. As asked in the question, maintain two unordered maps, say, **m1** and **m2**, where **m1** stores the color of houses in rows and **m2** stores the color of houses in columns.
One more information that needs to be stored is the sequence number in which a specific query is run.
Basically, for any row **i**,

m1[i] = {p,A_i} where p denotes the index of query and A_i is the color

For any column **i**,

m2[i] = {p,A_i} where p denotes the index of query and A_i is the color

Now, after all queries are stored in such a way, we need to understand the following aspects of the problem:
For any house defined as **(i,j)** ,

- If **m1[i]** and **m2[j]** are **0**, then it means this house is not painted in any query.
- If exactly one of **m1[i]** and **m2[j]** is **0**, then the house will have the color from **m1[i]** and **m2[j]** which is not **0**.
- If both **m1[i]** and **m2[j]** are not **0**, then the house will have color which has greater **p** (meaning the color that is painted after the other)

This can be done in **O(NM)** assuming unordered maps take **O(1)** time in accessing data.

Survival Test

This question is an example of how to use priority queue. You may refer more about it here.
First of all, store all the fighter strengths in the priority queue stored in descending order.
Now **repeat** the following step until the size of priority queue is more than **2**:

- **Pop the top 2 elements** from the priority queue.
- **Push their difference** back in the priority queue if their difference is not **0**.

After the above process, **if the size of priority queue is 0, your answer will be -1 else, your answer will be the top element of priority queue.**
This can be done in **O(NlogN)**.

Alice and Bob

This question is an example of a set. You may refer more about sets here.

- Store the given data in a **set** so that all duplicate elements get ignored.
- Now create two vectors, say, **arr1** and **arr2**.
- Store all elements of the set in **arr1**.

- Iterate through the elements of **arr1** and now push all elements of **arr1** divisible by **3** in **arr2** and set their **flag** to **1**.
- Iterate again through the elements of **arr1** and now push all elements of **arr1** divisible by **2** and **flag = 0** in **arr2** and set their **flag** to **1**.
- Iterate for the last time through the elements of **arr1** and now push all elements of **arr1** which have **flag = 0** to **arr2**.
- You have obtained the required two sequences asked in the question.
- Now do, the cake walk part of the question as asked in the question.

Since a set is used, further sorting isn't required.
This can be done in **O(NlogN)**.