Mcpc18 prep #1

Link to contest: https://open.kattis.com/contests/vsjw27/standings

First step

Understand what the problem is asking (very important step which is under-estimated sometimes)

Input to problem is

> Array of size N (positive elements)
> Integer m

Want to find continuous block within array such that:

1. the continuous block contains an integer m
2. m appears only once in this continuous block
3. m is the minimum element of this continuous block

Solution is guaranteed to exist because m appears at least once in array. Hence continuous block containing only m is a valid solution)

Output

> Sum of elements in said block

Working out a solution

Values which are less than m won't be part of any valid block. Makes sense to throw them away. This leaves us with some number of blocks which contains only numbers greater than or equal to m. We can now focus on this sub-problem.

Example:

> A = 1 3 2 0 6 2 4
> m = 2
>
> This leaves us with the following blocks:
> 3 2

6 2 4

Now, let us say we have one of these blocks which have only values >= m. Within these blocks, the valid outputs are the ones which contain only one occurrence of the value m. So we can scan these blocks from left to right and keep track of the number of m's we have seen so far. A few cases to consider:

1. No m's seen so far, can't update the answer but I will add the current value to a running sum because I might encounter a 'm' going forward
2. I have seen at least one 'm', I can update my answer with sum from current position to previous 2nd occurrence (but excluding) of 'm' going backwards. Here is an example (sum elements in bue)
    a. 1 2 3 4 2 3 4 5 6 2 3 4 9
3. I have seen exactly one 'm', sum I have so far is a valid solution

This can be implemented in linear time. Below is a possible implementation which gets accepted. Of course, there are other ways to implement this. Also note that this solution does not depend on the range of the values in the input array. These can be as high as 2^6 but we haven't used this at all. Maybe there are other ways to exploit this and come up with smarter solutions.

Link to solution (cpp): https://ideone.com/x8jXOY

Alternative solution: Two-Pointers technique

This problem can also be solved using the famous two-pointers technique, you'll need to maintain two pointers (nothing to have with C/C++ pointers) of the current subarray you are dealing with. Let's call the two pointers "L" and "R". Range [L, R) (notice that R is excluded & L is included) .

At any moment in the process, while "L" has not reached the end of the array yet (while(L <= n)) let's try to expand it while keeping in check 2 other variables:

- Weight: total weight of the subarray
- mCount: number of m's in the subarray

We need to keep mCount <= 1 always so that our subarray [L,R). If we have more than 1 occurrence of 'm' then the conditions above are not satisfied.

Now since we have our info on range [L,R), we try greedily to expand it even more to the right to [L, R+1) by trying to add the element at position R, we'll have the following cases:

- a[R] == m & mcount == 1 : we can't add this guy, since if we do, the range will have two m's, which we can't risk x)

- a[R] < m: we can't add this guy either, since if we do, we'll have an element which is smaller than m in our range
- Otherwise: we add it to the weight sum and update mCount if necessary

Once we've expanded all the way to the right (maintaining the conditions), we update the answer by taking the maximum between the previous answer and the answer for the current range [L, R) **ONLY** if mCount == 1 (if not, we don't have an "m" so it's not a valid range).

Now that we are done with the maximum range starting at index "L", let's remove element a[L] from our weight sum (update mCount if necessary) and then try to find the maximum range starting at "L+1" and so on.

Since we are moving L and R to the right only, the complexity of the algorithm is linear.
If you don't understand what the Two-Pointers technique means, make sure to read about it, or watch this video we shared in the group a while before (mother of advertising).

CPP Code: https://ideone.com/ptKQx4

Alternative solutions: Binary Search

Some may argue that this approach is a bit overkill for this kind of problem, but it's much more general, and can be used almost always when two pointer technique is used. This is sometimes easier to implement.

Problem A from Code IT 2018 and the GCD problem from JNJD 2018, can be solved using an idea similar to this.

The idea is simple, you just need to go to each index "i" such that a[i] == m, then find the right most position "R" (included) such that every element "x" in range [i+1, R] fulfil the condition "x > m". Same goes for "L" (the leftmost position included), now the answer is the sum in the range [L,R].

Doing this naively, we can have a $O(n^2)$ algorithm, but we can improve it to O(n log n) which will be nice and enough for this problem.

1) Once you get R and L, you'll need the sum in that range, let's use prefix sum data structure here to get this sum in O(1) time (with O(n) prior processing time of course)
2) The indices R and L are best to be found using binary search, that's the key idea.

3) To find R and L, you'll need sparse table to check that the minimum element in the range is > m.

The complexity then is:

- O(n) preprocessing the prefix sum
- O(n log n) preprocessing the "min" sparse table
- O(log(n)) for each binary search with O(1) spares table queries for each each index "i"

Total complexity is O(n log n)
(only if you use the O(1) per query sparse table, you can experiment the O(log(n)) one, you'll probably find that the total runtime O(n log^2(n)) will pass the time limit easily without a hitch)

CPP Code: 502
But instead of code, we'll show you how to binary search the rightmost "R" when being at a position "i" such that all [i+1, R] are > m: https://ideone.com/xXMPwX

Conclusions

We chose this problem because it is not very hard and there is room for multiple ideas/solutions. If you don't understand the solutions/buzz-words above then it's time to look them up in the internet.

We hope that you learned something new.

Cheers