C: Greedy, Observation
The answer will always be the lexicographically smallest or second smallest possible B. And the AND value will always be 0 or 1.

Let's disregard condition 3, notice that the lexicographically smallest B fulfilling all other conditions will always have bitwise AND value 0/1, because Bn will always be 1. To generate the lex smallest B, we start at the end, put the minimum possible B/ maximum possible C and keep going forward. If the bitwise and of lex smallest is 0 then that is the ans, otherwise we check whether that is the only answer or not, if not we generate the second smallest.

Code: https://codeforces.com/contest/322/submission/243513763
Code binary_search: https://codeforces.com/contest/322/submission/243513612

D: Sliding Window / Dp
Let n be the length of the string, and c be the number of different characters in the string.
The intended complexity is O(n*c*c), the constraints are set very tight to reject any solutions slower..

Let's first try to establish a O(n*c*c*c) solution.
Observation 1: See that the Smex for any string will never exceed c.

Let's fix the start index of a substring, how many substrings starting at index i are important to us? it's c*c. Because for each character, the indexes where the frequency of the ending char >c, will never contribute to our SMEX.

So there are total n*c*c important subarrays, if we keep a prefix sum of frequencies of each character, we can calculate the frequency and SMEX of each of those substrings from those in O(c), so the total complexity of the solution is O(n*c*c*c), which is not fast enough.

Observation 2:
Let's compare the important end indexes for subarrays starting at index i and (i+1), see that the edit distance will be 1. So, let the set of important end indexes for the substring starting at i be X, you delete i from x, and add any missing important index for i+1(there will be at most 1). You get the important indexes for i+1.

Observation 3:
See that for all the common important indexes between i and i+1, only the frequency of character at index i will change. So, a single loop through the linked list is enough to update frequency of all the characters in O(c*c).

So till now, we calculated the frequencies of all the characters in all the important substring in a total complexity of O(n*c*c), we need the MEX. For each substring ending at index j, if we keep

a std::set of frequencies not available and then update them with the previous calculations, we can get the MEX too. The complexity for this approach will be O(n*c*c*log(c)). This is too slow, unless you pull off some kind of miracle optimization.

Instead, let's just save mex[j] for each ending index. After each update of frequencies, if the frequency of something less than mex[j] becomes 0, we just set mex[j]=0, if freq[mex[j]] becomes 1 after some operation, we run a loop, from mex[j] to find the mex.

How is the complexity of this approach O(n*c*c)?
See that in the worst case the amount of time the mex loop will run is the total amount of time the frequency is changed for substring ending at j. Now as we are looping from left to right, it can only decrease, and as the frequencies are only important only when they are <c, the total frequency change will be O(c*c). So the complexity of that loop for each end index is O(c*c).

Which gives us total complexity of O(n*c*c)+n*O(c*c) -> O(n*c*c)
This will work.

Code: https://codeforces.com/contest/212/submission/243512648
Alter uses different optimization for MEX, hence the 3s time limit:
https://codeforces.com/contest/312/submission/243512924

E: Math

A=initial value of A
A`=Current value of A
Let's assume we know the number of bits in A(call it i), if we divide A by 2^i, A` becomes 1. Then we can do a binary search. A`*mid, check whether it is more or less than A and then do a A`/mid to go back to A`=1.

See that in the worst-case you'll need 60 queries to find the initial value of A. So do you need to make A`=1 in 4 queries? not actually.

See that for any value B<=A, if you do A' = A*(B+1)/B , A'>A
And for any value B>A , if you do A' = A*(B+1)/B , A==A

Now if we start doing it for each B=pow(2,i)[starting from i=29 downwards], for the first B <= A we will have A'>A is the number of bits A has. Now, does it destroy any bits of A? No actually, because A' will always be A+1, why? [Math goes here]. After that we can do two operation to make A'=A again[Edge case 1e9], and then divide A' by 2^i. Which will make it 1, and then we

do the binary search.

How does it work within 64 queries, let's say that the first on bit is x, see that finding the optimal B will take (29-x)*2 queries, and after that the binary search will take x*2 queries. So total queries = ~60 < 64.

Code: https://codeforces.com/contest/201/submission/243512024

H: Lazy Prop

The answer is always possible. Turn on and off operations can be done with segment tree lazy propagation.We have to keep track of the total number of one's and the sum of the positions of them. You can build an equation with those to find the answer.

Code: https://codeforces.com/contest/333/submission/243514245

I: DSU / Segtree

Code 1: https://codeforces.com/contest/353/submission/243517375
Code 2: https://codeforces.com/contest/353/submission/243517607