A Swapped Signs

Author: Michelle

In this string-parsing problem, we can loop through each index of the two strings in $O(n)$ time to see if their characters are not equal. If so, we increment a counter to let us know that we will have to swap this character out. Moreover, if the character we need to swap in is either B or X, we will have to increment the counter again to signify that we need to contribute a total of two letters to the board.

B Snowy Hill

Author: Yu

We can begin by performing a partial sum on the snow heights on the hill. We can then linear search for the size of the interval from 1 to N since the upper bound of Q is only 100. For each interval size, we binary search for the interval's position by calculating the sum within the interval with the pre-computed partial sum. If sum $< K$, we shift the interval up, otherwise, we shift the interval down.

C Journey to Nome

Author: Jimmy

First, we can find the prime factors of M using a standard algorithm such as this one. Next, we can brute force every number up to M to determine if which of those numbers are relatively prime with M.
To answer queries, we claim that for any number i, i is relatively prime with M if and only if $i\%M$ is relatively prime with M. This allows us to calculate the a_ith number that is relatively prime with M in a straightforward way.
We can find the prime factors of M in $O(\log M)$ time. We can then enumerate every relative prime between 1 and M in $O(\#\text{ prime factors of } M \times M) = O(M \log M)$ time worst case. Each query a_i can be answered in $O(1)$ time (neglecting the time complexity of the modulo operation). All in all, we get $O(M \log M + N)$ time complexity. I can provide a more detailed proof upon request. (I'm quite busy at the time I write this.)

D Sled Ordering

Author: Zeki

Hint: Think about the cases for small k. My (short, but rather ugly) C++ solution can be found here.

E Balto's Training

Author: Bennett

If we can efficiently find which node is K to the left/right of another node, this problem is pretty much solved.

There are a number of ways to do this. Without knowing many tricks, a fairly simple way is to follow paths from all nodes, find cycles, and use math to determine whether we've entered a cycle and if so, which part of the cycle we'll be in.

However, there's a nicer (to code) solution that utilizes the idea of binary lifting. In binary lifting, we store the nodes 2^x above every node for all values of x (up to a certain point). Since every node knows how to skip any power of 2, we can repeatedly skip large distances, covering a total of K units in $O(\log K)$ time.

This is fairly easy to compute as the location 2^K ahead of me is just the location 2^{K-1} ahead of the location 2^{K-1} ahead of me. Since the maximum length is 10^9, we need to compute up to 2^{29} ahead. After computing this for the left and right sides, we simply apply both sides several times to find the desired location.