Code Jam 2021 - Round 1C

Analysis: Double or NOTing

Test Set 1

Let's construct a graph where the nodes are numbers and the edges are the operations described in the statement. The problem asks us to find the shortest path from node $\mathbf{S}$ to node $\mathbf{E}$ in this graph. The graph is infinite, but we can inspect only a limited set of nodes to find the answer.

When thinking about binary representations, the double operation can be described as "add a trailing zero". Note that it is never optimal to add more zeroes than we have bits in $\mathbf{E}$: because the NOT operation can only drop leading digits, any additional zeroes will ultimately have to be dropped anyway.

We can use this fact to limit the maximal length to the sum of the lengths of $\mathbf{S}$ and $\mathbf{E}$, and then use a Breadth-first search algorithm to find the shortest path.

Test Set 2

Let's define a *bit group* as a maximally long group of consecutive zeroes or consecutive ones (that is, a group of all bits b that does not have a b adjacent to it). Let's assume that there are K bit groups in $\mathbf{S}$ and L bit groups in $\mathbf{E}$.

The double operation appends a bit to the right end of the binary string, and the NOT operation complements and possibly drops bits from the left end. Therefore, the result of applying operations to $\mathbf{S}$ comes from some (possibly empty) suffix of $\mathbf{S}$ (possibly complemented) and then some extra bits that were created using double. Let us call the bits that come from a bit in $\mathbf{S}$ *reused*.

First, let's consider the case when we do not reuse any bits from $\mathbf{S}$. In such a case we need to construct $\mathbf{E}$ naively using only the bits added by double operations and removing all the bits that came from $\mathbf{S}$ originally. We can do this by adding zeroes with double operations and applying a NOT operation whenever we need to start a new bit group. We also need to apply additional NOT operations to eliminate all bits that came from $\mathbf{S}$ originally. If we end up with $\mathbf{E}$, then let's count this as a possible answer. For example, to get from 101_2 to 11100_2 we would need to do perform the following sequence of operations (we use a space to separate bits originated from $\mathbf{S}$ from the extra bits added with double operations):

Operation	Result
double (to add an extra bit)	101 0
double (to add an extra bit)	101 00
double (to add an extra bit)	101 000
NOT (because the length of first bit group in $\mathbf{E}$ is 3, and we have added 3 zeros already, now we need to start a new bit group)	10 111
double (to add an extra bit)	10 1110
double (to add an extra bit)	10 11100
NOT (because we already have the desired $\mathbf{E}$ as suffix, now we just need to remove extra bits from $\mathbf{S}$)	1 00011
NOT (to remove the leftover bit of $\mathbf{S}$)	11100

If $\mathbf{S}$ ends in 0, we need to perform an extra NOT operation at the beginning so that our first double operation creates a brand new bit group. After we finish removing all bits from $\mathbf{S}$, we might need another NOT operation, if the removal process left us with the complement of $\mathbf{E}$.

Now, let's try to reuse some bits from $\mathbf{S}$.

The NOT operation removes one bit group from the left end of the binary representation and complements the rest. It is never optimal to apply a NOT operation more than $K + 1$ times, as in such a case we will either be looping between 0 and 1, or will be removing bits which were added with the double operation (which we could have just not added in the first place if they were not useful).

Let's assume that the answer will have NOT operations applied exactly X times. Let's apply X NOT operations to $\mathbf{S}$ to obtain S'. If S' is not a prefix of $\mathbf{E}$, then it is not possible to get to $\mathbf{E}$ from $\mathbf{S}$ using X NOT operations, as the double operation will not change the prefix of the string.

If S' is a prefix of $\mathbf{E}$, let's see whether we can construct the suffix. We will need exactly $Y = \text{len}(\mathbf{E}) - \text{len}(S')$ double operations. Let's say there are M bit groups in the suffix we need to create, then we will also need to apply M or $M + 1$ NOT operations, depending on the parity of M and the first bit of the prefix. Let's denote the number as Z. If Z is greater than X, then this case does not work, as we would apply more than X NOT operations, violating the assumption. Otherwise, the answer is $X + Y$, as even if X is greater than Z, extra NOT operations won't affect the suffix (we can perform them before adding any extra bits with a double operation).

We can now iterate through all possible values of X from 0 to $K + 1$, inclusive, and choose the minimum answer between all of those and the non-reuse case we considered first. If none of the cases work, we output `IMPOSSIBLE`.