Code Jam 2019 - Qualification Round

Analysis: Dat Bae

Test set 1

In this problem, we need to somehow identify which workers are not returning the bits that we send to them. Let's see how strings of bits change when some of the data is lost.

Imagine that we have ten workers, and we send them the following five random strings of bits (the *i*-th bit in each string is sent to the *i-th* worker):

```
0101010110
0101010101
0010100100
0110110101
0100100100
```

Also, imagine that workers 3 and 6 are broken. In this case, we will lose the following bits (highlighed in bold):

```
0101010110
0101010101
0010100100
0110110101
0100100100
```

In the result, we will receive the following bit strings:

```
01001110
01001101
00110100
01111101
01010100
```

Notice, how in this representation, we lost *columns* of bits because of the broken workers. It would be nice to be able to tell which columns we lost – then we will be able to determine which workers did not return the data!

Let's see if we can make all the columns different – then it will be easy to tell which ones are missing. In test set 1 we have up to **N** = 1024 workers, so we will need up to 1024 different columns. We can send up to **F** = 10 bit strings, which means our columns will consist of up to 10 bits.

Fortunately for us, 2^{10} = 1024, so we can make each column represent a unique number in the range from 0 to 1023. For example, we could make the *i*-th column represent the number *i*, in which case the first five columns, that represent numbers from 0 to 4, could look like this:

```
01010...
00110...
00001...
00000...
00000...
00000...
00000...
```

```
00000...
00000...
00000...
```

Now we can use a construction like this to see which workers are broken. If the *i*-th worker is broken, the bit column representing the number *i* will be missing in the result we receive.

Test set 2

In the second test set, we can only send up to **F** = 5 bit strings, which means our columns will consist of only up to 5 bits. $2^5 = 32$, so we can no longer use a previous approach of making each column unique.

Notice how we also know that **B** ≤ min(15, **N**-1), even though we have not used this fact in our solution so far. How does this additional constraint change the problem? The first thing to notice is that only a small fraction of columns will be missing when **N** = 1024, but these columns can still be in any positions.

Let's see what we can do with 32 numbers that we can represent with 5 bits. If we put these numbers in an order, that is

`0, 1, ..., 31`

we can notice that since **B**, which is less than 15, is also less than 32, this whole block of 32 numbers will never disappear completely. Let's see how we can make further use of this fact. If we have several blocks like this:

`0, 1, ..., 31, 0, 1, ..., 31, 0, 1, ..., 31,...`

none of the blocks of numbers from 0 to 31 will disappear completely, and for each the remaining numbers, we will always be able to identify which block it is from.

Let's examine this in more detail. Numbers inside each block are in an increasing order. Notice that even after some numbers disappear, when one block ends and the next one starts, numbers go down:

`0, 1, ..., 31, 0, 1, ...,` **`27, 5`** `, ..., 31,...` (numbers between 27 and 5 disappeared, but 27 is still bigger than 5)

Assume this is not the case, and after some number `X` from one block goes a number `Y` from the next block such that `X` ≤ `Y`. But this is impossible since there were at least 31 numbers between any such `X` and `Y`, and these numbers could not disappear altogether.

With these observations at hand, we are ready for a final solution. Let the bit columns of the strings we send to the database represent repeating blocks of numbers from 0 to 31:

`0, 1, ..., 31, 0, 1, ..., 31, 0, 1, ..., 31, ...` (**N** numbers total)

After we receive the responses from the database, we can iterate through the remaining numbers, noting that the new block starts when the current number is smaller than the previous one, and keeping track of how many blocks we have seen so far. Knowing the position of the numbers inside the block and the number of blocks we have seen, we can uniquely identify all the numbers we see: for example number 16 in the fifth block is `(5 - 1) * 32 + 17 = 145-th` number in the whole sequence. And if we know which numbers we have seen in the whole sequence, we can find out which numbers are missing, and output them as the numbers of the missing workers.

Finally, we used the fact that **B** $< 2^5$ to make the approach above work. But **B** $< 2^4$ too! That means we can use the same approach with only four strings. In this case, it is possible that two consecutive numbers from different blocks are equal, since there are $2^4 - 1 = 15$ other numbers in between them, which might all get deleted. Therefore, to detect block changes, we must use $\geq$ instead of $>$ between consecutive numbers.