0 - 1 sequences

Philip Stanton

February 2023

1 General

Please see: `https://open.kattis.com/problems/sequences` for the problem description

What makes this problem tricky is that we essentially need to keep track of 2^k possible sequences to know exactly how many inversions there are. Although, there are certain properties we can exploit to get an $O(n)$ algorithm

2 Idea

We first notice that the minimum number of inversions in a sequence is to take all chunks of consecutive ones in the sequence; and multiply the number of ones in each chunk by the number of leading zeros that chunk has.

Example:

1100 has a chunk of 11 in the $(0, 1)$ positions. If we multiply this by number of leading zeros the chunk has, we get $2 * 2 = 4$. It is quite easy to verify this is correct manually.

This leads to the idea that we can process the sequence one character at a time, keeping track of the number of ones, zeros and inversion. We keep the list sorted upon processing the *ith* character. If the character is a 0, then we simply add the total number of 1's to the current number of inversions. Otherwise, we add another 1 to the chunk.

Example:

At iteration i we have

$$000...1111$$

If the next character is 0, we have

$$000...11110$$

If the next character is 1, we have 000...11111

Since the list is sorted, we know exactly how many inversions to apply if a 0 is encountered.

If a '?' is found, we simply double the occurrences of all sequences. half of them append a 0, while the other half append a 1. Therefore, all we have to is double the number of inversions and increase by the number of ones for the 0 case. We also update the number of total ones. This then leads to the final algorithm

3 Algorithm

let $ones$ = total number of ones encountered
let inv = total number of inversions
let k = total number of '?' encountered
let x be input vector s.t. $x \in [1, 0, ?]^n$

```
k ← 0;
inv ← 0;
ones ← 0;
for c ∈ x do
    if c = 1 then
        ones ← ones + 2^k;
    end if
    else if c = 0 then
        inv ← inv + ones;
    end if
    else if c = ? then
        inv ← 2 * inv + ones;
        ones ← 2 * ones + 2^k;
        k ← k + 1;
    end if
end for
return inv
```

Since we simply iterate over the x vector of length n the entire algorithm is $O(n)$, which is more than efficient enough for the given problem constraints

4 Optimizations and stuff

- Note that we do not take the mod after each operation, which is required to pass on kattis.
- We shouldn't naively compute 2^k upon each loop iteration
- multiplying by 2 is better done with bit shifts