Problem 4: March Sadness (20 points)

By Sacheth Sathyanarayanan

Princeton University's men's basketball team made it to the Sweet 16 Round of March Madness 2023 but lost the match that would have qualified them to the Elite Eight. Alice is a sports analyst who was hired by the Basketball coach to see what went wrong. Alice reasons that Princeton could have selected a better team from its pool of basketball players and decides to develop a framework to think through selecting the "best" team.

Part (a) — How consistent is he? (3 points) [Codeforces]

Consistency is a very important measure of how good a basketball player is. Alice creates the following experiment to compute a suitable proxy for consistency: she picks a player and has them attempt to shoot a basketball a total of n times. If the player makes the shot, she writes down 1 and if they do not make the shot, she writes down 0. Thus, after the player makes n shots, she constructs a bit string of length n. She gives you this bitstring and asks you to compute the maximum number of consecutive 1s. This serves as an estimate for consistency.

Input: The first line contains a positive integer $n < 10^8$. The next line contains a bitstring of length n.

Output: The output should be the maximum number of consecutive ones.

Example:

Input:

```
14
11011111100011
```

Output:

```
6
```

Explanation: The values of all positions between 3 and 8 (inclusive) are all 1 and this is maximum.

Part (b) — What if things were different? (5 points) [Email Submission]

As you work on this problem, you notice that changing a 0 to a 1 or a 1 to a 0 has the potential to significantly change the maximum number of consecutive 1s. In order to further test this hypothesis, you decide to make your implementation robust to bit flips. To be more specific, you need to design three functions: `init`(n, s), `bit-flip`(x) and `num-max`().

The function `init`(n, s) is like a constructor: it takes in n, the number of shots the player takes and s, the bitstring of length n corresponding to the hits / misses of the player and returns nothing. You can use `init`(n, s) to initialize your data structures. The function `bit-flip`(x) takes as argument a position x (assume that $0 \leq x \leq n-1$) and flips the value in the bitstring at position x. The function `num-max`() computes and returns the maximum number of consecutive 1s.

You are guaranteed that `init` will be called first. After this, `bit-flip` and `num-max` could be called in any order any number of times. For full credit, you need `init` to run in $O(n)$ time and both `bit-flip` and `num-max` to run in $O(\log n)$ time per call. You are required to write up a complete description of each of these three functions and provide a brief justification of both correctness and the time bound.

Note: A naive solution would be to have `bit-flip`(x) just change the value of the bitstring and `num-max`() do the same thing as it did in part (a). This takes $O(1)$ time per call of `bit-flip`(x) but $O(n)$ time per call of `num-max`(). For full credit, your implementation needs to take $O(\log n)$ time per call to `bit-flip`(x) and `num-max`().

Part (c) — Which player should I select? (12 points) [Email Submission]

Now assume that Alice has done her analysis (with the help of your code!) on a total of n players $p_0, p_1, \ldots, p_{n-1}$. Armed with the insights from parts (a) and (b), she assigns a consistency score c_i, which is a positive real number, to each player p_i that corresponds to how consistent they are (a higher score implies a higher consistency). She knows, however, that picking a team is not just about maximizing consistency — team dynamics are also very crucial. To gather insights on team dynamics, she decides to survey each player, giving them a choice to select at most one *best friend*. For a player p_i, their best friend is defined as a player p_j such that $j > i$ and p_i refuses to be a part of the team unless p_j is a part of the team. Note that a player need not have a best friend; for instance, it is impossible for p_{n-1} to have a best friend (since there is no index greater than $n-1$). To reiterate, if Alice decides to have a player p_i on the team, then she must also have p_j, who is p_i's best friend, on the team. Note that this constraint proceeds recursively; thus, she must also include p_j's best friend on the team and so on.

Your task is to design these four functions: `init`(n, c, b), `min-consistency`(i), `update-consistency`(i, x) and `update-best-friend`(i, j). The function `init`(n, c, b), as in the previous case, functions like a constructor. It takes as input n, the number of players; c, an array of size n where the i^{th} component $c[i]$ is the consistency score of player p_i and b, an array of size n where the i^{th} component $b[i]$ is the index of the best friend of player p_i. By convention, if player p_i does not have a best friend, $b[i] = -1$. Note that by the constraints of the problem, if $b[i] \neq -1$, then $b[i] > i$. You can use the `init` function to initialize your data structures.

The function `min-consistency`(i) takes an integer i between 0 and $n-1$ (inclusive) as input and returns the minimum consistency score of a player that Alice must include on the team as a (potentially indirect) consequence of including player p_i on the team. The function `update-consistency`(i, x) updates the consistency score c_i of player p_i to a positive real number x. The function `update-best-friend`(i, j) updates the best friend of player p_i to be p_j.

We guarantee that the function `init` is called once at the beginning. After that, any of the three functions `min-consistency`, `update-consistency` and `update-best-friend` can be called in any order in any number of times. Your task is to ensure that the `init` function runs in $O(n)$ time and to minimize the amortized cost per call to `min-cost`, `update-consistency` and `update-best-friend`. For any significant credit, all of these functions must run in $o(n)$ amortized cost.

How to Submit

Email submissions to part (b) and (c) separately to coscon.submit@gmail.com. If you must resubmit, *respond to the thread where you sent your original submission; we cannot guarantee that your resubmission will be graded otherwise.*

Part (a)
Submit your code on Codeforces.

Part (b)
Write the description of your functions as well an analysis of correctness. Don't forget to prove your time bound! Write this in a file called *Problem4bSubmission* and add it as an attachment.

Part (c)
Write the description of your functions as well an analysis of correctness. Don't forget to prove your time bound! Write this in a file called *Problem4cSubmission* and add it as an attachment.