

IOI '21 P4 - Mutating DNA

Time limit: 2.0s **Memory limit:** 512M

Grace is a biologist working in a bioinformatics firm in Singapore. As part of her job, she analyses the DNA sequences of various organisms. A DNA sequence is defined as a string consisting of characters "A", "T", and "C". Note that in this task DNA sequences **do not contain character "G"**.

We define a mutation to be an operation on a DNA sequence where two elements of the sequence are swapped. For example a single mutation can transform "ACTA" into "AATC" by swapping the highlighted characters "A" and "C".

The mutation distance between two sequences is the minimum number of mutations required to transform one sequence into the other, or -1 if it is not possible to transform one sequence into the other by using mutations.

Grace is analysing two DNA sequences a and b , both consisting of n elements with indices from 0 to $n - 1$. Your task is to help Grace answer q questions of the form: what is the mutation distance between the substring $a[x..y]$ and the substring $b[x..y]$? Here, a substring $s[x..y]$ of a DNA sequence s is defined to be a sequence of consecutive characters of s , whose indices are x to y inclusive. In other words, $s[x..y]$ is the sequence $s[x]s[x + 1] \dots s[y]$.

Implementation Details

You should implement the following procedures:

```
void init(std::string a, std::string b)
```

- a, b : strings of length n , describing the two DNA sequences to be analysed.
- This procedure is called exactly once, before any calls to `get_distance`.

```
int get_distance(int x, int y)
```

- x, y : starting and ending indices of the substrings to be analysed.
- The procedure should return the mutation distance between substrings $a[x..y]$ and $b[x..y]$.
- This procedure is called exactly q times.

Example

Consider the following call:

```
init("ATACAT", "ACTATA")
```

Let's say the grader calls `get_distance(1, 3)`. This call should return the mutation distance between $a[1..3]$ and $b[1..3]$, that is, the sequences "TAC" and "CTA". "TAC" can be transformed into "CTA" via 2 mutations: **TAC** $\rightarrow$ **CAT**, followed by **CAT** $\rightarrow$ **CTA**, and the transformation is impossible with fewer than 2 mutations.

Therefore, this call should return 2.

Let's say the grader calls `get_distance(4, 5)`. This call should return the mutation distance between sequences "AT" and "TA". "AT" can be transformed into "TA" through a single mutation, and clearly at least one mutation is required.

Therefore, this call should return 1.

Finally, let's say the grader calls `get_distance(3, 5)`. Since there is **no way** for the sequence "CAT" to be transformed into "ATA" via any sequence of mutations, this call should return -1 .

Constraints

- $1 \leq n, q \leq 100\,000$
- $0 \leq x \leq y \leq n - 1$
- Each character of a and b is one of "A", "T", and "C".

Subtasks

1. (21 points) $y - x \leq 2$
2. (22 points) $q \leq 500, y - x \leq 1000$, each character of a and b is either "A" or "T".
3. (13 points) each character of a and b is either "A" or "T".
4. (28 points) $q \leq 500, y - x \leq 1000$
5. (16 points) No additional constraints.

Sample Grader

The sample grader reads the input in the following format:

- line 1: $n\ q$
- line 2: a
- line 3: b
- line $4 + i$ ($0 \leq i \leq q - 1$): $x\ y$ for the i -th call to `get_distance`.

The sample grader prints your answers in the following format:

- line $1 + i$ ($0 \leq i \leq q - 1$): the return value of the i -th call to `get_distance`.

Attachment Package

The sample grader and sample test cases are available here: [dna.zip](#).