Мутацияланууучу(алмашуучу) ДНК

ДНК = Дез Оксирибо Нуклеин кычкылы.

Грейс - Сингапурдагы биоинформатикалык фирмада иштеген биолог. Жумушунун алкагында ал ар кандай организмдердин ДНК тизмелерин талдайт. ДНК тизмеси "A", "T" жана "C" белгилеринен турган сап катары аныкталат. Бул тапшырмада ДНК тизмелеринде **"G" белгиси жок экендигин эске алыңыз.**

Мутацияны, ДНК тизмеси боюнча, ырааттуулуктун эки элементи алмаштырылган операция деп аныктайбыз. Мисалы, бир мутация "A **C** T **A** " белгисин "A" жана "C" белгилерин алмаштыруу менен "A **A** T **C** " түрүнө өзгөртө алат.

Эки ырааттуулуктун ортосундагы мутациялык аралык - бул бир катарды экинчисине которуу үчүн талап кылынган эң аз мутациялар саны, же мутациялардын жардамы менен бир ырааттуулукту экинчисине айлантуу мүмкүн болбосо, (-1).

Грейс 0дөн $(n-1)$ге чейинки индекстери бар n элементтеринен турган a жана b эки ДНК ырааттуулугун талдап жатат. Сиздин милдетиңиз Грейстин q сурооно жооп берүүсүнө жардам берүү: $a[x..y]$ тизмече менен $b[x..y]$ тизмече мутациясынын аралыгы кандай? Мында $s[x..y]$ тизмече ДНК тизмесинин s индекси x менен y кошо турган s ырааттуу белгилердин тизмеги катары аныкталат. Башкача айтканда, $s[x..y]$ = $s[x]s[x+1]$ ... $s[y]$ ырааттуулугу.

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: T**A**C $\rightarrow$ C**A**T, followed by C**AT** $\rightarrow$ C**TA**, 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) a and b consist of characters "A" and "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`.