Mutating DNA

Grace es una bióloga trabajando en una firma de bioinformática en Singapur. Como parte de su trabajo, ella analiza secuencias de ADN de varios organismos. Una secuencia de ADN es definida como un string formado por los caracteres "A", "T" y "C". Note que en esta tarea las secuencias de ADN **no contienen la letra "G"**.

Nosotros definimos una mutación como una operación en una secuencia de ADN donde dos elementos de la secuencia son intercambiados. Por ejemplo una mutación simple puede transformar "A**C**T**A**" en "A**A**T**C**" intercambiando los caracteres resaltados "A" y "C".

La distancia de mutación entre dos secuencias es el número mínimo de mutaciones requeridas para transformar una secuencia en la otra, o -1 si es imposible transformar una secuencia en la otra utilizando mutaciones.

Grace está analizando dos secuencias de ADN a y b, donde ambas contienen n elementos con índices desde 0 hasta $n-1$. Su tarea es ayudar a Grace a responder q preguntas de la forma: ¿cuál es la distancia de mutación entre el substring $a[x..y]$ y el substring $b[x..y]$? Aquí, un substring $s[x..y]$ de una secuencia de ADN s es definido como una secuencia de caracteres consecutivos de s, cuyos índices son x y y inclusive. En otras palabras, $s[x..y]$ es la secuencia $s[x]s[x+1]\ \ldots\ s[y]$.

Detalles de Implementación

Usted deberá implementar los siguientes procedimientos

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

- a, b: strings de longitud n, describiendo las dos secuencias de ADN a ser analizadas.
- Este procedimiento es invocado exactamente una vez, antes de cualquier invocación a `get_distance`.

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

- x, y: índices de inicio y final de los substrings para ser analizados.
- El procedimiento deberá retornar la distancia de mutación entre los substrings $a[x..y]$ y $b[x..y]$.
- Este procedimiento será invocado exactamente q veces.

Ejemplo

Considere la siguiente invocación:

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

Digamos que el grader invoca `get_distance(1, 3)`. Esta invocación deberá retornar la distancia mutación entre $a[1..3]$ y $b[1..3]$, esto es, las secuencias "TAC" y "CTA". "TAC" puede ser transformada en "CTA" via 2 mutaciones: **TA**C $\rightarrow$ **C**A**T**, seguida de C**AT** $\rightarrow$ C**TA**, y la transformación es imposible de hacer con menos de 2 mutaciones.

Entonces, esta llamada deberá retornar 2.

Digamos ahora que el grader invoca `get_distance(4, 5)`. Esta invocación deberá retornar la distancia de mutación entre las secuencias "AT" y "TA". "AT" puede ser transformada en "TA" a través de una simple mutación, y claramente solo una mutación es requerida.

Entonces, esta invocación deberá retornar 1.

Finalmente, digamos que el grader invoca `get_distance(3, 5)`. Dado que **no hay** forma de que la secuencia "CAT" pueda ser transformada en "ATA" por ninguna secuencia de mutaciones, esta invocación deberá retornar -1.

Restricciones

- $1 \le n, q \le 100\ 000$
- $0 \le x \le y \le n - 1$
- Cada caracter de a y b es uno entre "A", "T" y "C".

Subtareas

1. (21 points) $y - x \le 2$
2. (22 points) $q \le 500,\ y - x \le 1000$, cada caracter de a y b es "A" or "T".
3. (13 points) cada caracter en a y b puede ser solamente "A" o "T".
4. (28 points) $q \le 500,\ y - x \le 1000$
5. (16 points) Sin restricciones adicionales.

Evaluador de Ejemplo

El evaluador de ejemplo lee la entrada en el siguiente formato:

- línea 1: n q
- línea 2: a
- línea 3: b
- línea $4 + i$ ($0 \le i \le q - 1$): x y para la i-ésima invocación a `get_distance`.

El evaluador de ejemplo imprime sus respuestas en el siguiente formato:

- línea $1 + i$ ($0 \le i \le q - 1$) : el valor de retorno de la i-th invocación a `get_distance`.