Mutating DNA

Grace é uma bióloga que trabalha numa empresa de bioinformática de Singapura. Como parte do seu trabalho, ela analisa sequências de DNA de vários organismos. Uma sequência de DNA é definida como sendo uma string de caracteres "A", "T" e "C". Nota que neste problema as sequências de DNA **não contêm o caracter "G"**.

Definimos uma mutação como sendo uma operação na sequência de DNA onde dois elementos da sequência são trocados. Por exemplo, uma única mutação pode transformar "A**C**T**A**" em "A**A**T**C**" ao trocar os caracteres a negrito "A" e "C".

A distância de mutação entre duas sequências é o número mínimo de mutações necessário para transformar uma sequência noutra, ou -1 se não for possível transformar uma sequência noutra usando mutações.

Grace está a analisar duas sequências de DNA a e b, ambas consistindo em n elementos com índices de 0 a $n-1$. A tua tarefa é ajudar Grace a responder a q questões da forma: qual é a distância de mutação entre a substring $a[x..y]$ e a substring $b[x..y]$? Aqui, a substring $s[x..y]$ de uma sequência de DNA s é definida como sendo a sequência de caracters consecutivos de s cujos índices são de x a y, inclusive. Por outros palavras, $s[x..y]$ é a sequência $s[x]s[x+1]\ \ldots\ s[y]$

Detalhes de Implementação

Deves implementar as seguintes funções:

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

- a, b: strings de tamanho n, descrevendo as duas sequências de DNA a serem analisadas.
- Esta função deve ser chamada exatamente uma vez, antes de quaisquer chamadas a `get_distance`.

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

- x, y: os índices de início e fim das substrings a serem analisadas.
- A função deve devolver a distância de mutação entre as substrings $a[x..y]$ e $b[x..y]$.
- Esta função é chamada exatamente q vezes.

Exemplo

Considera a seguinte chamada:

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

Imaginemos que o avaliador chama `get_distance(1, 3)`. Esta chamada deve devolver a distância de mutação entre $a[1..3]$ e $b[1..3]$, ou seja, as sequências "TAC" e "CTA". "TAC" pode ser transformada em "CTA" através de 2 mutações. **TA**C $\rightarrow$ **C**A**T**, seguida de C**AT** $\rightarrow$ C**TA**, e a transformação é impossível de fazer em menos do que 2 mutações.

Deste modo, a chamada deve devolver 2.

Imaginemos que o avaliador chama `get_distance(4, 5)`. Esta chamada deve devolver a distância de mutação entre as sequências "AT" e "TA". "AT" pode ser transformada em "TA" através de uma única mutação, e claramente pelo menos uma mutação é necessária.

Deste modo, a chamada deve devolver 1.

Finalmente, imaginemos que o avaliador chama `get_distance(3, 5)`. Como **não existe** maneira da sequência "CAT" ser transformada em "ATA" através de uma sequência de mutações, esta chamada deve devolver -1.

Restrições

- $1 \leq n, q \leq 100\ 000$
- $0 \leq x \leq y \leq n - 1$
- Cada caracter de a e b é um entre "A", "T" e "C".

Subtarefas

1. (21 pontos) $y - x \leq 2$
2. (22 pontos) $q \leq 500,\ y - x \leq 1000$, cada caracter de a e b é "A" ou "T".
3. (13 pontos) cada caracter de a e b é "A" ou "T".
4. (28 pontos) $q \leq 500,\ y - x \leq 1000$
5. (16 pontos) Nenhuma restrição adicional.

Avaliador Exemplo

O avaliador exemplo lê o input no seguinte formato:

- linha 1: n q
- linha 2: a
- linha 3: b
- linha $4 + i$ ($0 \leq i \leq q - 1$): x y para i-ésima chamada a `get_distance`.

O avaliador exemplo escreve as tuas respostas no seguinte formato:

- linha $1 + i$ ($0 \leq i \leq q - 1$) : o valor de retorno da i-ésima chamada a `get_distance`.