ADN en Mutation

Grace est une biologiste qui travaille dans une entreprise de bioinformatique à Singapour. Dans le cadre de son travail, elle analyse les séquences d'ADN de divers organismes. Une séquence d'ADN est définie comme une chaîne composée des caractères "A", "T" et "C". Notez que dans ce sujet, les séquences d'ADN **ne contiennent pas le caractère "G".**

Nous définissons une mutation comme une opération sur une séquence d'ADN dans laquelle deux éléments de la séquence sont échangés. Par exemple, une seule mutation peut transformer "A**C**T**A**" en "A**A**T**C**" en échangeant les caractères mis en gras "A" et "C".

La distance de mutation entre deux séquences est le nombre minimum de mutations nécessaires pour transformer une séquence en l'autre, ou -1 s'il n'est pas possible de transformer une séquence en l'autre en utilisant des mutations.

Grace analyse deux séquences d'ADN a et b, toutes deux constituées de n éléments avec des indices de 0 à $n-1$. Votre objectif est d'aider ***Grace*** à répondre à q questions de la forme : quelle est la distance de mutation entre la sous-chaîne $a[x..y]$ et la sous-chaîne $b[x..y]$? Ici, une sous-chaîne $s[x..y]$ d'une séquence d'ADN s est définie comme une séquence de caractères consécutifs de s, dont les indices sont de x à y inclus. En d'autres termes, $s[x..y]$ est la séquence $s[x]s[x+1]\ldots s[y]$.

Détails d'implémentation

Vous devez implémenter les fonctions suivantes :

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

- a, b : chaînes de longueur n, décrivant les deux séquences d'ADN à analyser.
- Cette fonction est appelée une seule fois, avant tout appel à `get_distance`.

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

- x, y : indices de début et de fin des sous-chaînes à analyser.
- La fonction doit retourner la distance de mutation entre les sous-chaînes $a[x..y]$ et $b[x..y]$.
- Cette fonction est appelée exactement q fois.

Exemple

Considérons l'appel initial suivant :

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

Considérons que l'évaluateur appelle ensuite `get_distance(1, 3)`. Cet appel doit retourner la distance de mutation entre $a[1..3]$ et $b[1..3]$, c'est-à-dire les séquences "TAC" et "CTA". "TAC" peut être transformé en "CTA" via 2 mutations : **TA**C $\rightarrow$ **C**A**T**, suivi de C**AT** $\rightarrow$ C**TA**, et la transformation est impossible avec moins de 2 mutations.

Par conséquent, cet appel doit retourner 2.

Considérons maintenant que le correcteur appelle `get_distance(4, 5)`. Cet appel doit retourner la distance de mutation entre les séquences "AT" et "TA". "AT" peut être transformé en "TA" par une seule mutation, et il est clair qu'au moins une mutation est requise.

Par conséquent, cet appel doit retourner 1.

Enfin, considérons que le correcteur appelle `get_distance(3, 5)`. Puisqu'il n'y a **aucun moyen** pour que la séquence "CAT" soit transformée en "ATA" via une séquence de mutations, cet appel doit retourner -1.

Contraintes

- $1 \leq n, q \leq 100\ 000$
- $0 \leq x \leq y \leq n - 1$
- Chaque caractère de a et b est l'un des caractères "A", "T" et "C".

Sous-tâches

1. (21 points) $y - x \leq 2$
2. (22 points) $q \leq 500,\ y - x \leq 1000$, chaque caractère de a et b est soit "A" soit "T".
3. (13 points) chaque caractère de a et b est soit "A" soit "T".
4. (28 points) $q \leq 500,\ y - x \leq 1000$
5. (16 points) Aucune contrainte supplémentaire.

Evaluateur d'exemples

L'évaluateur d'exemple lit l'entrée au format suivant :

- ligne 1 : n q
- ligne 2 : a
- ligne 3 : b
- ligne $4 + i$ ($0 \leq i \leq q - 1$) : x y pour le i-ème appel à `get_distance`.

L'évaluateur d'exemple affiche vos réponses au format suivant :

- ligne $1 + i$ ($0 \leq i \leq q - 1$) : la valeur de retour du i-ème appel à `get_distance`.