Mutating DNA

Grace este biolog și lucrează la o firmă de bioinginerie în Singapore. Ca parte a job-ului, ea analizează secvențe ADN pentru diverse organisme. O secvență ADN este definită ca un string format din carcterele "A", "T", și "C". Observați că în această problemă secvențele ADN **nu conțin caracterul "G".**

Definim o mutație ca fiind o operație pe o secvență ADN unde două elemente ale secvenței sunt interschimbate. De exemplu o singură mutație poate transforma "A**C**T**A**" în "A**A**T**C**" prin interschimbarea caracterelor îngroșate "A" și "C".

Distanța-mutație între două secvențe este numărul minim de mutații necesare pentru a transforma o secvență în cealaltă sau -1 dacă nu este posibilă transformarea unei secvențe în alta folosind mutații.

Grace analizează două secvențe ADN a și b, ambele constând în n elemente indexate de la 0 la $n-1$

Sarcina voastră este să o ajutați pe Grace să răspundă la q întrebări de forma: care este distanța-mutație între substringul $a[x..y]$ și substringul $b[x..y]$? Aici, un substring $s[x..y]$ a unei secvențe ADN s este definit ca o subsecvență de caractere consecutive ale lui s, a cărei indici sunt de la x la y inclusiv.
Cu alte cuvinte, $s[x..y]$ este subsecvența $s[x]s[x+1]\ \ldots\ s[y]$.

Detalii de implementare

Trebuie să implementați următoarele proceduri:

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

- a, b: șir de caractere de lungime n, descriind cele două secvențe ADN ce urmează să fie analizate.
- Această funcție este apelată exact o dată, înainte de oricare apel `get_distance`.

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

- x, y: indicii de început și de sfârșit ale substringurilor care vor fi analizate.
- Procedura trebuie să returneze distanța-mutație între substringurile $a[x..y]$ și $b[x..y]$.
- Această procedură este apelată de exact q ori.

Exemplu

Considerăm următorul apel:

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

Să presupunem că grader-ul apelează `get_distance(1, 3)`. Acest apel ar trebui să returneze distanța-mutație între $a[1..3]$ și $b[1..3]$, adică, între secvențele "TAC" și "CTA". "TAC" poate fi transformată in "CTA" prin 2 mutații: **TA**C $\rightarrow$ **CA**T, urmată de C**AT** $\rightarrow$ C**TA**, și transformarea nu este posibilă în mai puțin de 2 mutații.

De aceea, acest apel trebuie să returneze 2.

Să presupunem că grader-ul apelează `get_distance(4, 5)`. Acest apel ar trebui să returneze distanța-mutație între secvențele "AT" și "TA". "AT" poate fi transformată în "TA" cu o singură mutație, și evident că este necesară cel puțin o mutație.

De aceea, acest apel trebuie să returneze 1.

În final, să presupunem că grader-ul apelează `get_distance(3, 5)`. Cum este **imposibil** ca secvența "CAT" să fie tansformată în "ATA" prin orice secvență de mutații, acest apel trebuie să returneze -1.

Restricții

- $1 \leq n, q \leq 100\ 000$
- $0 \leq x \leq y \leq n - 1$
- Fiecare caracter din a și b este unul dintre caracterele "A", "T", și "C".

Subtasks-uri

1. (21 puncte) $y - x \leq 2$
2. (22 puncte) $q \leq 500,\ y - x \leq 1000$, fiecare caracter din a și b este "A" sau "T".
3. (13 puncte) fiecare caracter din a și b este "A" sau "T".
4. (28 puncte) $q \leq 500,\ y - x \leq 1000$
5. (16 puncte) Fără restricții suplimentare.

Exemplul de grader

Grader-ul citește datele de intrare în următorul format:

- linia 1: n q
- linia 2: a
- linia 3: b
- linia $4 + i$ ($0 \leq i \leq q - 1$): x y pentru apelul i al funcției `get_distance`.

Grader-ul afișează răspunsul în următorul format:

- linia $1 + i$ ($0 \leq i \leq q - 1$) : valoarea returnată de alpelul i al funcției `get_distance`.