OMI 2022 - Solución para *5-5 Feliz encuentro*

$Q = 1$. La respuesta es 1 ó 0.

La subtarea especifica que la respuesta a lo más es 1. Significa que como máximo puede haber dos símbolos iguales juntos. **OJO: Es importante tomar en cuenta que la pregunta cubre sólo un rango, de modo que pueden existir parejas de símbolos iguales que estén fuera del rango.**

Una posible solución a esta subtarea es recorrer de la posición L a la posición $R - 1$ y preguntar en cada caso si $T[i] == T[i + 1]$, si en algún momento esta condición evalúa a *verdadero* se debe escribir un 1 y si no un 0.

$T, Q, L_i, R_i \leq 1000$.

Esta subtarea tiene límites que permiten para cada pregunta (Q preguntas), recorrer el rango de L_i a R_i (como máximo medirá 1000 ya que ese el el largo máximo del texto) y contar la cantidad de parejas de posiciones que tienen el mimso símbolo.

El siguiente ciclo es suficiente para resolver esta subtarea:

```
    res = 0;
    for (int i = l + 1; i <= r; ++i) {
      if (st[i] == st[i - 1]) ++res;
    }

    std::cout << res << "\n";
```

Para cada pregunta el ciclo puede resolver como máximo todo el texto, por lo tanto, la complejidad de la solución es $O(TQ)$.

$L_i = 1$ para todas las preguntas.

Supón que te piden contestar las siguientes 2 preguntas: $[1, 5]$ y $[1, 10]$. Para la primera pregunta tienes que contestar cuántas parejas iguales hay entre las posiciones 1 y 5. Para la segunda cuántas parejas iguales hay entre las posiciones 1 y 10. Si se usa el método de la subtarea anterior estarás contando dos veces el intervalo de 1 a 5. ¿Por qué no mejor, para el rango $[1, 10]$ usar lo que ya habías calculado del rango $[1, 5]$? Después de todo, las parejas en el rango $[1, 5]$ también están en el rango $[1, 10]$.

La técnica usada para resolver esta subtarea se llama *precálculo de prefijos* y consiste en precalcular, antes de contestar ninguna pregunta, el resultado de todos los prefijos posibles (un prefijo es un intervalo que va desde el primer elemento de la lista hasta un elemento intermedio). Una vez teniendo los precálculos se puede contestar una pregunta en tiempo constante.

Para hacer el *precálculo de prefijos* eficiente debes utilizar los prefijos cuyo resultado ya conoces. Definamos $prefijo[i]$ como la cantidad de parejas iguales que hay en el prefijo que va de la posición 1 a la posición i. Se empieza entonces de $prefijo[1] = 0$ ya que en el intervalo $[1, 1]$ sólo hay una letra y es imposible que exista una pareja igual y posteriormente se va calculando el $prefijo[i]$ en base al prefijo anterior de la siguiente manera:

```
  for (int i = 2; i <= t; ++i) {
    prefijo[i] = prefijo[i - 1];  // AL MENOS LOS MISMOS QUE EL PREFIJO ANTERIOR
    if (texto[i] == texto[i - 1])
      ++prefijo[i];  // SI EL NUEVO SIMBOLO ES IGUAL AL ANTERIOR, UNA PAREJA MAS
  }
```

Para contesar la pregunta $[1, R_i]$ basta con escribir $prefijo[R_i]$.

El *precálculo de prefijos* se hace recorriendo una vez el texto, contestar las preguntas se hace en tiempo constante. La complejidad de la solución es $O(T + Q)$

$100 > (R_i - L_i) - A_i$ donde A_i es la respuesta a la i-ésima pregunta.

Esta subtarea puede ser algo confusa. Analizándola puedes ver que significa lo siguiente: En un intervalo $[L_i, R_i]$ va a haber una cantidad de parejas iguales muy similar al largo del intervalo, de hecho, la diferencia máxima puede ser de 100. Es decir, si el intervalo mide $50,000$ tiene que haber al menos $49,900$ parejas de símbolos iguales.

Observación: Cada pareja de posiciones que no es *feliz encuentro* es un cambio de símbolo.

La observación anterior junto con las condiciones de la subtarea implican que en cada pregunta habrá a lo más 100 cambios de símbolo, es decir, a lo más 100 grupos distintos de posiciones con el mismo símbolo en cada intervalo de pregunta.

La solución entonces para esta subtarea es dividir el texto en grupos de un mismo símbolo. Se sabe que cada pregunta abarcará a lo más 100 de estos grupos, por lo que si tenemos una forma eficiente de encontrar el primer grupo que intersecta con el intervalo de la pregunta, podemos recorrer uno por uno los grupos e ir contando el total de parejas. Una manera de encontrar eficientemente el primer grupo que intersecta con la pregunta es mantener los grupos en un `set` ordenados por la posición del último elemento del grupo. El siguiente código muestra la solución para esta subtarea:

```
// HAZ UNA LISTA DE RANGOS DE CARACTERES IGUALES
st += '*';  // SE AGREGA UN * AL FINAL PARA USAR COMO MARCADAOR DE FINAL
l = r = 0;
for (int i = 1; i <= t; ++i) {
  if (st[i] == st[i - 1])
    ++r;  // SI ESTE CARACTER ES IGUAL AL ANTERIOR, AUMENTA EL RANGO
  else {
    // AGREGA EL RANGO SIEMPRE Y CUANDO SEA MAYOR QUE 1, LOS DE 1 NO TIENEN
    // ENCUENTROS Y NO HACE SENTIDO AGREGARLOS
    if (l != r) rangos.emplace_hint(rangos.end(), r, l);
    l = r = i;
  }
}

// PARA CADA PREGUNTA BUSCA LOS RANGOS DONDE SE CRUZA.
while (q--) {
  std::cin >> l >> r;
  --l;
  --r;
  res = 0;

  // BUSCA EL PRIMER RANGO CUYO FINAL ESTE DESPUES (O IGUAL) QUE EL INICIO DE
  // LA PREGUNTA.
  auto it = rangos.lower_bound({l, 0});
  while (it != rangos.end()) {
    // EL NUMERO DE FELICES ENCUENTROS DE ESTE RANGO ES IGUAL AL TAMAÑO DE LA
    // INTERSECION ENTRE EL RANGO Y LA PREGUNTA MENOS UNO.
    int interseccion = min((*it).fin, r) - max((*it).inicio, l) + 1;
    if (interseccion > 0) res += interseccion - 1;

    it++; // AVANZA AL SIGUIENTE RANGO
    if ((*it).inicio > r) break; // SI YA SE PASA DE LA PREGUNTA, TERMINA
  }
  std::cout << res << "\n";
}
```

La construcción del `set` de grupos se hace en tiempo lineal. Cada pregunta requiere un tiempo logarítmico para encontrar el primer grupo y posteriormente un máximo de 100 avances en el `set` para sacar el total. La complejidad de la solución es $O(T + Q(100 + log_2 T))$

Solución de 100 puntos, ninguna restricción.

Regresemos a la solución que utiliza el *precálculo de prefijos*. El precálculo te permite contestar de forma eficiente la pregunta: *¿cuántas parejas hay entre el inicio del texto y la posición i?* Observa que si quieres contesar la pregunta de cuántas parejas hay en el intervalo $[L, R]$ puedes hacerlo restando de las parejas que hay entre el inicio y la posición R las parejas que hay entre el inicio y la posición L, es decir, quitándole las del *pedazo* que no quieres.

Usando el *precálculo de prefijos* de la subtarea anterior se puede contesar la pregunta $[L, R] = prefijo[R] - prefijo[L]$.

Esta técnica se conoce normalmente como *acumulado de prefijos* y se utiliza en problemas en los que se desea contestar preguntas sobre un intervalo de elementos cuyo valor se mantiene fijo entre pregunta y pregunta.

El código para contestar las respuestas es el siguiente:

```
while (q--) {
  std::cin >> l >> r;
  --l;
  --r;
  std::cout << prefijo[r] - prefijo[l] << "\n";
}
```

El precálculo se hace en tiempo lineal y contestar cada pregunta en tiempo constante. La complejidad de la solución es $O(T + Q)$.