Luis Rubiano
Sebastián Gaona
Isaac Bermúdez

Algoritmo de solución:

En la solución planteada principalmente hay dos grandes casos: cuando la subsecuencia contiene dos repeticiones de la misma letra y cuando son dos letras distintas.

Para la identificación de estos dos casos se utilizó la función esTriangular() que básicamente compara los caracteres de la subsecuencia para verificar si son iguales.

Con la función esOptimo() se pretende evaluar si ya se ha conseguido el máximo número de ocurrencias posibles tomando como referencia para el primer caso la fórmula de los números triangulares y para el segundo la fórmula de la sucesión matemática de los cuartos de cuadrado que era la que más se ajustaba a los casos de prueba que se realizaron manualmente.

El método subrutina básicamente busca la posición donde se favorece el reemplazo de la letra y lo realiza según las apariciones de los caracteres de la subcadena en la cadena principal.

Para evaluar el número de ocurrencias de la subcadena se desarrolló la función contar() en donde a partir de la distinción entre los casos mencionados se ejecutan dos métodos de solución: para el primer caso básicamente es que a partir del número de apariciones de un carácter en el string se retorna el triangular asociado. Para el otro caso, se realiza la cuenta del número de apariciones de cada una de las letras de la subcadena y se va incrementando el número de apariciones conjuntas para retornar las ocurrencias.

El método mejorar aplica la función subrutina para el string pasado como parámetro y aplica el método contar evaluando el resultado y devolviendo una cadena que aumenta el número de ocurrencias de la subcadena.

Finalmente, para aplicar el algoritmo se itera sobre el número de cambios disponibles y si el caso es triangular se realiza el cálculo la función inversa de los números triangulares para así encontrar el número de apariciones del primer carácter de la subcadena sumándole 1 y con él las apariciones de la subcadena, se evalúa si son las óptimas y se decide si se reemplaza o no un carácter. Para el caso contrario, se invoca la función mejorar y se cuentan las ocurrencias en el string retorno, a partir de la evaluación de optimalidad de este string se decide si se reemplaza o no un carácter. Así se continua hasta no tener más cambios disponibles

Complejidad:

La complejidad temporal del algoritmo es **O(len(X)*m)** (m siendo el número de reemplazos), la operación que toma más tiempo es contar(), la cual se ejecuta en tiempo O(len(X)), y cuando se itera sobre el número de reemplazos la complejidad total del algoritmo acaba siendo **O(len(X)*m).**

La complejidad espacial es **O(len(X)*m)**, la peor operación es mejorar() la cual crea copias del string original, lo cual toma espacio O(len(X)), al iterar sobre los reemplazos la complejidad acaba siendo **O(len(X)*m)**.

Escenarios:

ESCENARIO 1: Suponga ahora que un movimiento no aplica solo a una posición, sino a todo X. Es decir, un movimiento cambiaría toda ocurrencia de un carácter x en X. Por ejemplo, para $X = "abacachi"$ un posible movimiento seria cambiar el carácter "a" por el carácter "u", dando como resultado $X = "ubucuchi"$.

- Este escenario cambia el comportamiento de los movimientos, cuando realizo un movimiento, modifico todos los caracteres elegidos, y sumaría a lo sumo |e2| * |R| subsecuencias, donde |e2| es el conteo del número de ocurrencias del segundo carácter del patrón, y |R| es el número de ocurrencias que cambie. Y lo mismo para el cambio para e1.
-

ESCENARIO 2: YA NO HAY LÍMITE DE MOVIMIENTOS

Si ya no hay límite de movimientos, el número que se va a retornar es el óptimo, existen dos casos para esto.

- Caso 1 (Y[0] = Y[1] ej: Y = 'aa'): en este caso, el óptimo va a ser el número triangular asociado al tamaño de X, es decir (len(X)*len(X-1))//2
- Caso 2 (Y[0] ≠ Y[1] ej: Y = 'ab'): en este caso, el óptimo va a ser OEIS A002620 (Quarter-squares) de len(X), es decir, floor(n^2 / 4)