Recuperación de la secuencia de edición de Levenshtein

En esta función se pide obtener, además del coste, la secuencia de operaciones de edicón. Esto equivale a una representación gráfica de qué cambios se realizaron sobre una palabra para obtener otra.

Este resultado lo podemos conseguir recorriendo la matriz de estados desde el último elemento hasta llegar al primero. En cada paso comprobaremos el elemento justo encima, justo anterior y en diagonal a la posición actual y saltaremos a la posición que contenga el menor valor. Saltar una columna hacia arriba supone añadir una tupla (xi-1, “”) a la secuencia, saltar hacia atrás supone (“”, yj-1) y el salto en diagonal supone (xi,yj). En el código esto se implementa con tres bucles. Cabe destacar que en el primer bucle el orden de las condiciones if es relevante para obtener el resultado adecuado: si el valor en las posiciones comprobadas es el mismo nos debemos guiar por la posición que nos acerca más a las coordenadas (0,0). Esto nos lleva a tener que elejir dar un salto en diagonal siempre que se pueda.

A medida que recorremos la matriz debemos de añadir las tuplas de elementos de x e y a una lista. Esta es el objeto que devolvemos. Contendrá la secuencia de operaciones de ecición, pero, para que esté ordenada, deberemos darle la vuelta.

Ampliaciones Damerou’Levenshtein Restricted

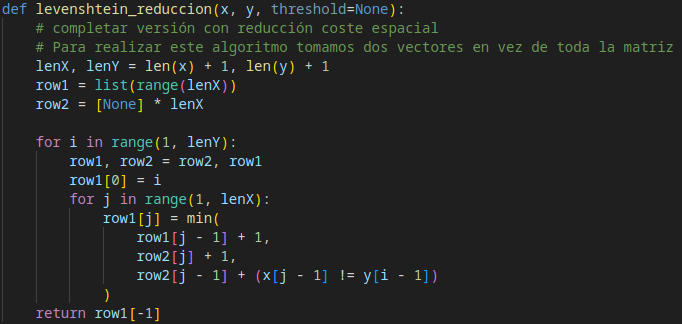
En este caso tendremos que resolver el problema de la distancia Damerou-Levenshtein sin poder reutilizar aquellas letras en las que ya hemos hecho un intercambio del tipo (“ab”,“ba”). Se nos pedía hacer tres funciones distintas que se comentarán a continuación.

La primera es damerou\_restricted\_matrix, que se puede construir a partir de la función levenshtein\_matriz pero añadiendo otra clásula if en el bucle que recorre las posiciones de la matriz. El código dentro de este if se ejecutará si estamos suficientemente “lejos” de una pared y si las letras de las palabras x e y son intercambiables y, si lo son, establecerá el valor de la celda actual como (j-2, i-2) + 1 si es menor que el de las transformaciones de Levenshtein.

La segunda fucnión a crear es damerou\_restriucted\_edicion donde se nos pide la secuencia de edicón de la transformación aplicada a la palabra x. Aquí también podemos reutilizar el código de la función antes comentada añadiéndole un bucle que recorre la matriz desde el último al primer elemento guiándonos por el la posición que contenga el menor valor. Los valores a comprobar son ahora son los mismos que para la secuencia de edición de Levenshtein más el valor en las cordenadas (j-2, i-2). Si hay varias celdas con el valor igual al mínimo nos guiaremos por aquella que nos lleva más cerca a las coordenadas (0,0).

Para la última función que se nos pide de damerou restricted debemos reducir el coste espacial y añadir un threshold. Necesitaremos un mínimo de tres filas para poder hallar la distancia de Damerou-Levenshtein y el resto del proceso será el mismo al explicado anteriormente. Al final de cada iteración tendremos que “rotar” los valores de estas filas y, para implementar la parada por threshold comprobaremos si en las 3 últimas iteraciones los valores de la última fila están por encima de este. Si lo están significará que es imposible conseguir un coste menor o igual que el threshhold y se parará la ejecución devolviendo threshold + 1.

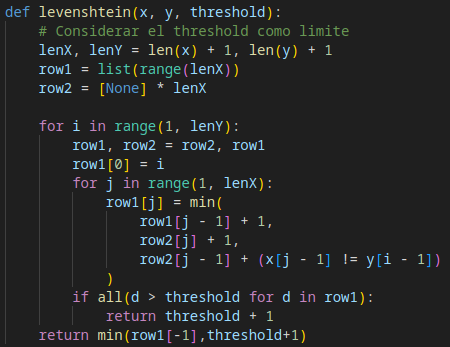
Cabe anotar que para implementar la versión restringida de este algorimto no hemos añadido ninguna comprobación extra ya que no hace falta. Tal y como está diseñado nuestro código no se considerará la opción de repetir una trasposición.



El objetivo de esta función es aplicar el algoritmo de levenshtein con una reducción del coste espacial, dado que solo es necesario tener en memoria las dos últimas filas para poder realizar las comparaciones necesarias para la realización de este algoritmo.

Primero se inicializan las longitudes de la misma manera que en el Levenshtein normal, y inicializamos las dos primeras filas del algoritmo, **row1** como el rango de **lenX** y **row2** como una lista vacía del la misma longitud.

Después comienza el bucle donde compararemos estas 2 filas. Primero las cambiamos para que **row2** sea la base, hacemos que el primer elemento de la fila donde vamos a hacer las comparaciones sea igual a **i**, y iteramos a lo largo del resto de **row1** haciendo las evaluaciones correspondientes a las operaciones de inserción, borrado y sustitución. Guardamos su mínimo y seguimos hasta acabar el bucle, que es cuando devolveremos el último valor obtenido.



Esta función es igual que la de **levenshtein\_reduccion**, excepto que hacemos un uso de **threshold** para dejar de hacer comparaciones en cuanto la distancia de levenshtein se pasa de este **threshold**.

Respecto al código, esto significa hacer una comprobación al final del cálculo de cada fila, de manera que si todos los elementos de la **row1** son estrictamente mayores que **threshold**, entonces dejamos de iterar y devolvemos el **threshold**.