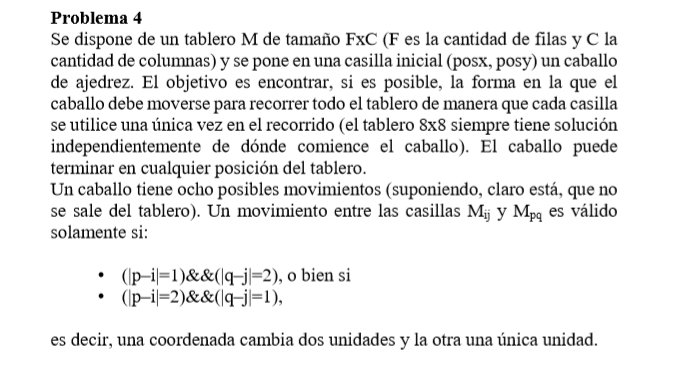
PL5

JAVIER MARTÍN GÓMEZ

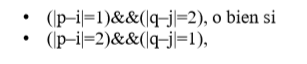
47231977M

\*\*\*\*\*CORREGIDO\*\*\*\*\*

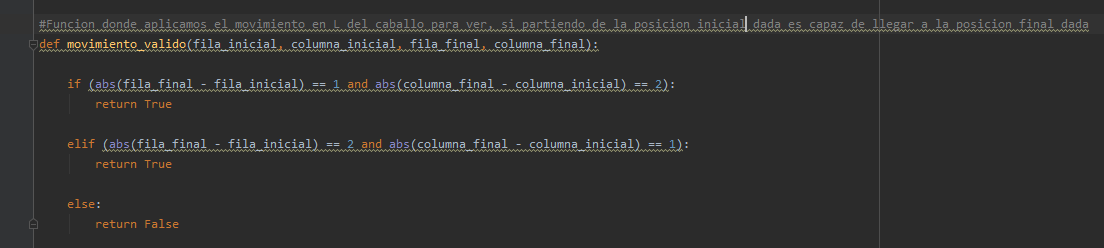


Queremos averiguar si, partiendo de una posición inicial y de un tablero con x,y coordenadas, un caballo (con su movimiento característico en L) es capaz de recorrer todas las posiciones.

Para ello, primero creamos una función (movimiento\_valido) donde introducimos como parámetro una posición inicial y una final y nos dice si es capaz de llegar a la final desde la inicial con el movimiento en L. Para ello, utilizamos la siguiente fórmula:

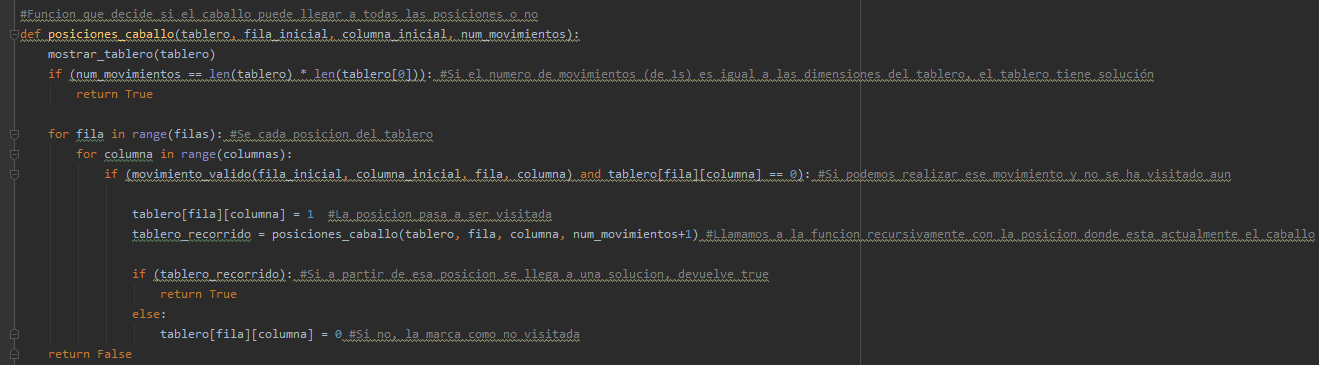


Por ejemplo, si estamos en la posición 0,0 y queremos llegar a la 1,2. Hacemos 1-0 = 1 (correcto) y 2-0 = 2 (correcto), por lo que el movimiento es correcto.

Esta sería la función:

Ahora, para saber si el caballo puede llegar a recorrer todo el tablero utilizamos la función posiciones\_caballo, donde se introducen como parámetros el tablero, la posición inicial y el número de movimientos (inicialmente a 1, ya que la posición inicial ya se ha visitado).

Recorremos todas las posiciones del tablero y vemos si desde la posición inicial se puede llegar. Utilizamos un bucle for para ir recorriendo. Cuando encontramos una a la que podemos llegar desde la posición inicial, llamamos recursivamente a la función con esa posición como “nueva inicial” y vemos si partiendo de esa posición, podemos llegar a una solución. Si llegamos, retornamos True y habremos encontrado la solución y si no, marcamos la casilla como vacía y volvemos a ver las siguientes posiciones (backtracking). En el momento que el número de movimientos sea igual a las dimensiones del tablero habremos encontrado la solución. Si ha recorrido todo el tablero y no ha encontrado solución, devuelve False.

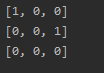
La función es la siguiente:

Veamos ejemplos. Para el tablero 3x3 nunca va a encontrar solución, ya que a la posición del medio es imposible acceder. Veamos cómo se desarrollaría el árbol:

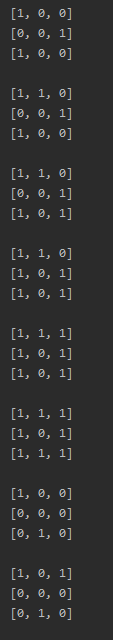
Parte de este tablero inicial:

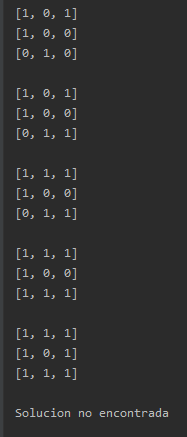


La primera posición a la que accede es la siguiente:



A partir de ahí, el árbol se iría desarrollando de la siguiente forma, volviendo marcha atrás cuando no encuentre solución.:

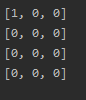




Como vemos, recorre todo el tablero para todas las posiciones y no encuentre solución, por lo que devuelve solución no encontrada

Ahora, para el tablero 4x3 partiendo de la 0,0 va a encontrar solución

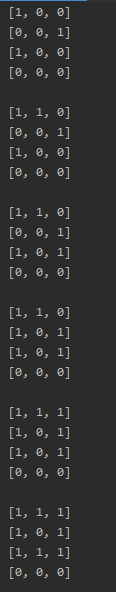
Parte de este tablero inicial:



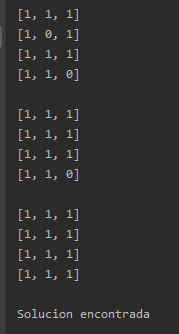
Accede a esta posición en primer lugar:



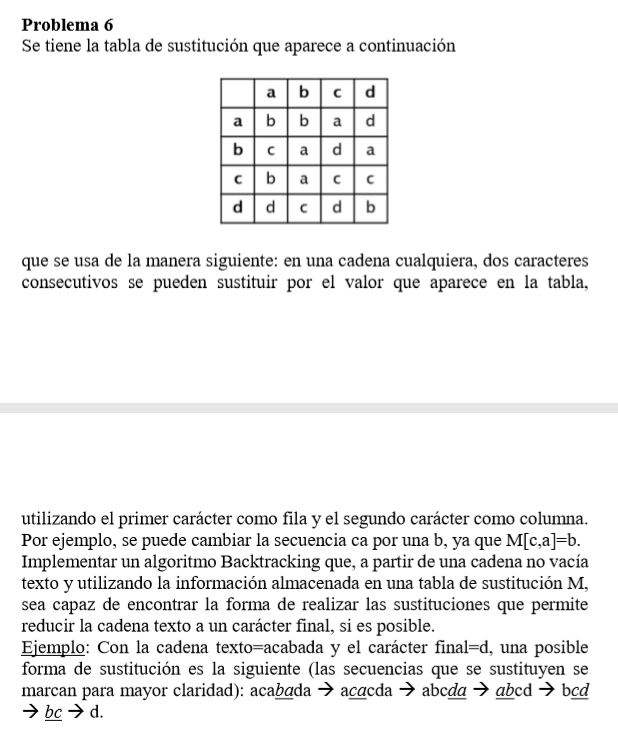
Ahora, el árbol iría teniendo las siguientes ramas con las posiciones a las que puede llegar partiendo de esa posición:



Sigue recorriendo el tablero hasta que encuentra una posible solución:



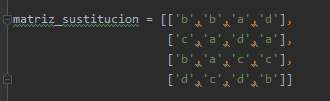
\*\*\*\*\*CORREGIDO\*\*\*\*\*



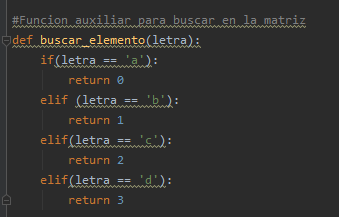
Queremos averiguar si, a partir de una matriz de sustitución de caracteres y una cadena de entrada, podemos llegar a un carácter pedido.

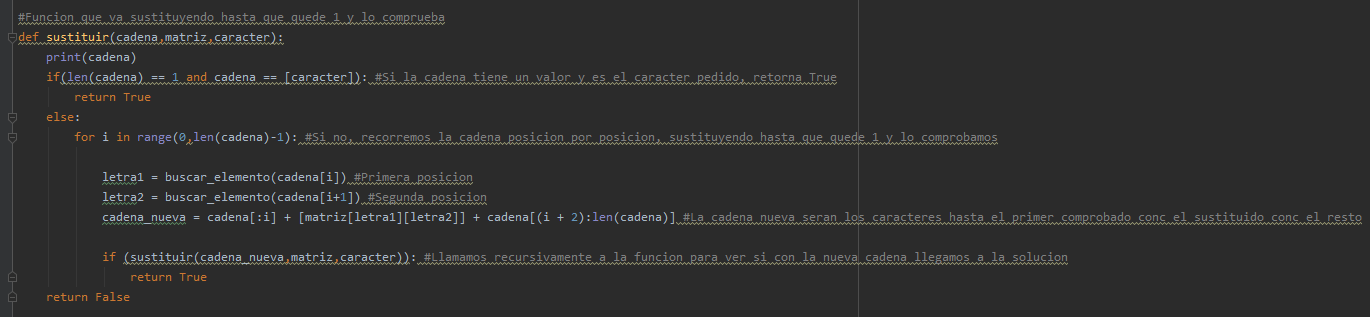
Primero de todo tenemos la matriz de sustitución (como la del enunciado) que consiste en sustituir un par de caracteres por un carácter. Por ejemplo, si tenemos el par “bb” la sustitución será “a”.

Inicializamos una matriz:



En la que la posición 0 de las filas y columnas coincide con a, la 1 con b y así sucesivamente, por lo que creamos una función auxiliar que busca cada sustitución en la matriz dependiendo del par de caracteres:



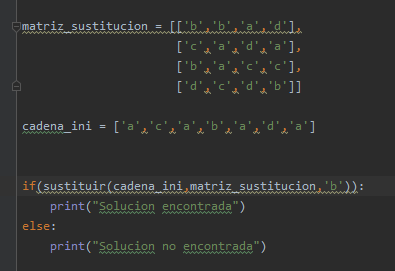
La función principal es la siguiente:

Podemos ver que se le introducen como parámetros, la cadena de caracteres, la matriz de sustitución y el carácter que queremos hallar sustituyendo pares de caracteres.

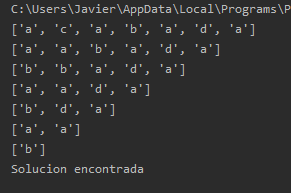
Primero de todo, comprobamos que, si la cadena tiene longitud 1 y es igual al carácter introducido, nos devuelve True, por lo que la solución se ha encontrado. Si no, a través de un bucle for, vamos recorriendo posición por posición de la cadena empezando por la primera posición. Hallamos la posición de cada letra para buscar la sustitución en la matriz y lo sustituimos en la cadena, por lo que la cadena resultante será todo lo anterior al par de caracteres sustituidos concatenado con la sustitución concatenado con el resto.

Una vez hallada la cadena resultante, llamamos recursivamente a la función para comprobar si a partir de esa cadena podemos llegar a una solución. Cuando quede un carácter en la cadena, lo comprobará con el carácter introducido y si no son iguales, vuelve a comprobar la cadena anterior con la siguiente posición (backtracking), y así sucesivamente hasta que encuentre una solución o, en caso contrario, termine el bucle for y devuelva False.

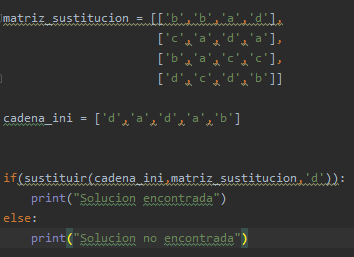
Para el caso del enunciado con los siguientes datos:



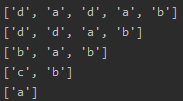
El árbol tendría las siguientes ramas y encontraría solución sin la necesidad de backtracking:



Para este enunciado vamos a hacer uso de backtracking:



El árbol sería de la siguiente forma:



Llega a una solución que no es la correcta por lo que vuelve hacia atrás (bab y en vez de coger ba, coge ab):



Tampoco es correcto, por lo que sigue yendo hacia atrás, ahora a ddab y en vez de coger dd, coge da:



Sigue sin ser correcta la solución, por lo que va hacia atrás y en vez de coger dd, coge db:



db se convierte en c y dc en d y llegamos a la solución pedida.