Algoritmia básica - Practica 2 (2016), pp. 1–2
Manuel Lagunas Arto

¿Cuánto cuesta transformar una cadena en palíndromo?

Manuel Lagunas[1]

[1]Universidad de Zaragoza, Algoritmia básica

Abstract
El siguiente documento contiene la información referente a la práctica 2 de la asignatura de Algoritmia básica. El objetivo de esta es calcular el coste entendido como inserciones, borrados o cambios de letras en una cadena para crear un palindromo. Las siguientes lineas exponen la estrategia seguida para su desarrollo asi como las decisiones tomadas. El código ha sido desarrollado por el propio autor de este documento.
En el documento se usa $len(x)$ para hacer referencia a la longitud de x.

1. Mayor subsecuencia palindrómica

Para el desarrollo del algóritmo global se ha seguido una estrategía que parte del resultado de la mayor subsecuencia palindrómica de la cadena de entrada. [Wik] Se ha seguido una estrategia de programación dinámica para abordar el problema. Los subresultados del método se han almacenado en una matriz de dimensiones $n+1 * n+1 | n = len(cadena)$, la primera fila estará inicializada a 0, la segunda a 1 y la tercera con *nil*. Para una entrada *alg* se generará una estructura como la tabla 1.

0	0	0	0
1	1	1	1
nil	nil	nil	nil
nil	nil	nil	nil

Table 1: *Inicialización de la estructura de datos que guarda las sub-ejecuciones de la función para calcular la subsequencia palindrómica más larga*

1.1. Algoritmo

Una vez la tabla esta inicializada se iterará sobre dos bucles que la recorreran, el primer índice i comienza en la posicion 2 hasta $len(cadena)$, el segundo, j tendrá valor 0 hasta i (se entiende que el bucle i es el que contiene el bucle j). Dentro del bucle se compararán posiciones contiguas de los carácteres de la cadena de entrada de manera que si ambos son iguales se actualice la posicion de la tabla i, j para que su valor sea $[i-2, j+1]+2$ dada la inicialización de la tabla este valor será 1 en la primera ejecución ($j = 0$ y busca en $j+1$, la primera fila se inicializa en 1s) al que debemos sumar 2 dado que hemos encontrado en la cadena 2 carácteres iguales $len(subsecuencia) = carcteres_a adidos + len(subsecuencia - cracteres_a adidos)$. El coste asíntotico de dicho bucle será la multiplicación del coste da cada uno, que podemos aproximar a $O(n^2)$.

El hecho de que la tabla disponga de esa fila de 1s en la segunda fila es debido a que lo que se busca en la subsequencia palíndromica con mayor número de elementos, esto implica que ese resultado seá impar siempre (el número de elementos de la subsecuencia), dado que si esta es par indicá que tiene parejas de carácteres a los que se les puede añadir uno en medio sin alterar el resultado. La fila con ceros es necesaria para comenzar la ejecución sin errores y podría ser un resultado si no buscaramos la subsequencia más larga. Tras acabar las iteraciones encontramos en la última fila, en la primera posición el resultado del tamaño de dicha cadena tal y como puede verse en la tabla 2.

0	0	0	0	0
1	1	1	1	1
1	1	1	nil	nil
3	3	nil	nil	nil
3	nil	nil	nil	nil

Table 2: *Tabla con sub-ejecuciones al acabar los bucles anidados. Se observa el resultado en la posición de la última fila, primera columna. La tabla esta medio vacia por optimizaciones a la hora de ejecutar el algoritmo, pues si tienes una palabra no es necesario volver a comparar su segunda mitad con la primera, pues ya estaba hecho antes.*

1.2. Generar la subsecuencia

Para ahora obtener un resultado que nos devuelva la subsequencia palindrómica que buscamos tendremos que volver a recorrer la entrada una sola vez, y por tanto, con coste lineal $O(n)$. El bucle se ejecutara de la parte final de la cadena al comienzo, además tendremos un elemento w que apuntara al principio, de manera que

podamos comparar carácteres dos a dos y añadirlos al palindromo si estos coinciden. Durante el bucle w solo se actualizará en el momento que los dos caráctares comparandose no coincidan y se compruebe en la tabla de sub-ejecuciones que la posición en la misma columna pero en la fila inferior es mayor (mayor número de elemntos en esa sub-ejecución, esto controlará los elementos que debemos añadir al palindromo). Tras acabar esta ejecución el método devolverá tanto el número de carácteres, como el palíndromo.

2. Obtención del palindromo

Una vez se tiene la palabra de entrada y su mayor subsecuencia palindrómica, es inmediato calcular el número de operaciones necesarias, que será $len(entrada) - len(subsecuencia)$. Se conoce que los elementos de la subsecuencia palindrómica estan contenidos en la entrada, por lo tanto habra que iterar sobre los elementos de esta, con un coste $O(n)$, mirando cuales se encuentran en la entrada y eleminando los elementos que no pertenezcan a esta. Existe un caso particular en la ejecución, en el cual los elementos que quedan por observar de la secuencia palindrómica en la entrada son mayores que los elementos que hay en la entrada, en ese caso en vez de borrar elementos habrá que hacer un cambio de los elementos que queden en la entrada por los elementos de la subsecuencia palindrómica, haciendo que el resultado sea algo más eficiente en cuanto a pasos a dar para obtener el palíndromo final.

3. Resultados

El agoritmo desarrollado no es el que devuelve el resultado óptimo, sino que devuelve una aproximación a este. El trabajo fúturo a desarrollar sería continuar haciendo que el resultado obtenido se optimice de acuerdo a el número de movimientos a realizar, haciendo uso de algoritmos como la distancia de Levenshtein [**?**].

References

[Wik] WIKIPEDIA: *Longest common subsequence problem*. https://en.wikipedia.org/wiki/Longest_common_subsequence_problem. 1

```
Number of operations -- 1
Input string -- mannaam
Delete pos 6 -- mannam
Final result: mannam
'mannam'

Number of operations -- 10
Input string -- algoritmia basica
Delete pos 2 -- agoritmia basica
Delete pos 2 -- aoritmia basica
Delete pos 2 -- aritmia basica
Delete pos 2 -- aitmia basica
Delete pos 3 -- aimia basica
Delete pos 3 -- aiia basica
Delete pos 6 -- aiia asica
Delete pos 7 -- aiia aica
Change pos 8 for i -- aiia aiia
Final result: aiia aiia
'aiia aiia'

Input string -- Esto es una cadena larga
Number of operations -- 17
Add a in pos 0 -- aEsto es una cadena larga
Delete pos 2 -- asto es una cadena larga
Delete pos 2 -- ato es una cadena larga
Delete pos 2 -- ao es una cadena larga
Delete pos 2 -- a es una cadena larga
Delete pos 3 -- a s una cadena larga
Delete pos 3 -- a  una cadena larga
Delete pos 4 -- a  na cadena larga
Delete pos 7 -- a  na adena larga
Delete pos 8 -- a  na aena larga
Delete pos 8 -- a  na ana larga
Delete pos 9 -- a  na an larga
Delete pos 10 -- a  na an arga
Delete pos 10 -- a  na an rga
Delete pos 10 -- a  na an ga
Change pos 10 for   -- a  na an  a
Final result: a  na an  a

Number of operations -- 5
Input string -- 12342331234
Delete pos 1 -- 2342331234
Delete pos 1 -- 342331234
Delete pos 2 -- 32331234
Delete pos 5 -- 3233234
Delete pos 7 -- 323323
Final result: 323323
'323323'
```

Figure 1: *Ejemplos de ejecución del algoritmo desarrollado*