



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO



## ANÁLISIS DE ALGORITMOS

PROFESOR: CRISTHIAN ALEJANDRO ÁVILA  
SÁNCHEZ

PROYECTO:

PROBLEMA NP-COMPLETO – LONGEST  
PALINDROMIC SUBSEQUENCE

ALUMNO: GARAYOA FLORES ROBERTO  
ALESSANDRO

GRUPO: 3CM11

## 1. Introducción

¿Qué son los problemas NP-Completos?

El concepto de "NP-completo" fue introducido por Stephen Cook en un artículo titulado «The complexity of theorem-proving procedures» en las páginas 151-158 de Proceedings of the 3rd Annual ACM Symposium on Theory of Computing en 1971, aunque el término "NP-completo" como tal no aparece en el documento. En la conferencia de ciencias de la computación hubo un intenso debate entre los científicos de la computación sobre si los problemas NP-completos podían ser resueltos en tiempo polinómico o en una máquina de Turing determinista. John Hopcroft llevó a todos los asistentes de la conferencia a consenso concluyendo que el estudio sobre si los problemas NP-completos son resolubles en tiempo polinómico debería ser pospuesto ya que nadie había conseguido probar formalmente sus hipótesis ni en un sentido ni en otro.

NP toma su nombre de “nondeterministic polynomial time”, un término que remonta a las raíces de la Teoría de la Complejidad. Intuitivamente, significa que se puede encontrar y verificar una solución a cualquier problema de búsqueda en tiempo polinomial mediante un tipo de algoritmo especial (y bastante poco realista), llamado “Algoritmo no Determinista”. Tal algoritmo tiene el poder de adivinar correctamente en cada paso.

Subsecuencia Palindrómica más larga

Primero para un poco de contexto, debemos de conocer lo que es un palíndromo y es que, la palabra “Palíndromo” viene del griego palin dromein que significa literalmente “que recorre a la inversa”. Es decir, que la última letra de una frase es la misma que la primera, que la penúltima es idéntica a la segunda, etcétera, etcétera. Igual que los números y las fechas capicúas (2002, 23.832, 11.11.11...), pero con las letras. Por supuesto es el pasatiempo favorito de los supersticiosos, de los locos de la simetría y de los amantes de las letras.

## 2. Planteamiento del problema

*Como otro ejemplo, si la secuencia dada es "BBABCBCAB", entonces la salida debería ser 7 ya que "BABCBAB" es la subsecuencia palindrómica más larga en ella. "BBBBB" y "BBCBB" también son subsecuencias palindrómicas de la secuencia dada, pero no las más largas.*

*La solución ingenua para este problema es generar todas las subsecuencias de la secuencia dada y encontrar la subsecuencia palindrómica más larga. Esta solución es exponencial en términos de complejidad temporal. Veamos cómo este problema posee las dos propiedades importantes de un problema de programación dinámica (DP) y puede resolverse de manera eficiente utilizando la programación dinámica.*

1) Subestructura óptima:

Sea  $X[0..n-1]$  la secuencia de entrada de longitud  $n$  y  $L(0, n-1)$  la longitud de la subsecuencia palindrómica más larga de  $X[0..n-1]$ .

Si el último y el primer carácter de  $X$  son iguales, entonces  $L(0, n-1) = L(1, n-2) + 2$ . De lo contrario,  $L(0, n-1) = \max(L(1, n-1), L(0, n-2))$ .

También se pueden encontrar otras condiciones tales como:

// Cada carácter individual es un palíndromo de longitud 1

$L(i, i) = 1$  para todos los índices  $i$  en secuencia dada

// Si el primer y último carácter no son iguales

Si  $(X[i] \neq X[j])$   $L(i, j) = \max\{L(i+1, j), L(i, j-1)\}$

// Si solo hay 2 caracteres y ambos son iguales

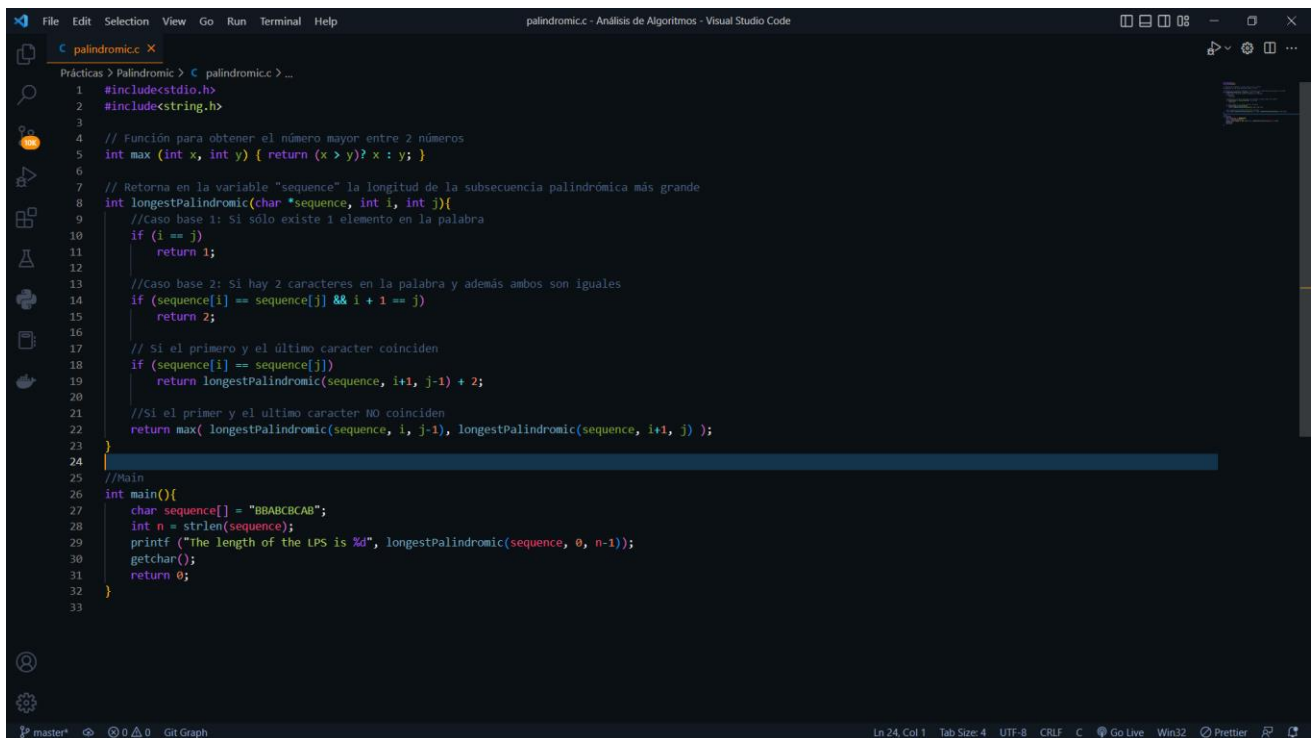
De lo contrario, si  $(j == i + 1)$   $L(i, j) = 2$

// Si hay más de dos caracteres, primero y último

// los caracteres son iguales

De lo contrario  $L(i, j) = L(i+1, j-1) + 2$

### 3. Implementación



```
File Edit Selection View Go Run Terminal Help
palindromic - Analisis de Algoritmos - Visual Studio Code

Prácticas > Palindromic > C palindromic > ...
1 #include<stdio.h>
2 #include<string.h>
3
4 // Función para obtener el número mayor entre 2 números
5 int max (int x, int y) { return (x > y)? x : y; }
6
7 // Retorna en la variable "sequence" la longitud de la subsecuencia palindrómica más grande
8 int longestPalindromic(char *sequence, int i, int j){
9     //Caso base 1: Si sólo existe 1 elemento en la palabra
10    if (i == j)
11        return 1;
12
13    //Caso base 2: Si hay 2 caracteres en la palabra y además ambos son iguales
14    if (sequence[i] == sequence[j] && i + 1 == j)
15        return 2;
16
17    // Si el primero y el último caracter coinciden
18    if (sequence[i] == sequence[j])
19        return longestPalindromic(sequence, i+1, j-1) + 2;
20
21    //Si el primer y el ultimo caracter NO coinciden
22    return max( longestPalindromic(sequence, i, j-1), longestPalindromic(sequence, i+1, j) );
23 }
24
25 //Main
26 int main(){
27     char sequence[] = "BBABCB CAB";
28     int n = strlen(sequence);
29     printf ("The length of the LPS is %d", longestPalindromic(sequence, 0, n-1));
30     getchar();
31     return 0;
32 }
33
```

```
#include<stdio.h>
#include<string.h>

// Función para obtener el número mayor entre 2 números
int max (int x, int y) { return (x > y)? x : y; }

// Retorna en la variable "sequence" la longitud de la subsecuencia palindrómica más grande
int longestPalindromic(char *sequence, int i, int j){
    //Caso base 1: Si sólo existe 1 elemento en la palabra
    if (i == j)
        return 1;

    //Caso base 2: Si hay 2 caracteres en la palabra y además ambos son iguales
    if (sequence[i] == sequence[j] && i + 1 == j)
        return 2;

    // Si el primero y el último caracter coinciden
    if (sequence[i] == sequence[j])
        return longestPalindromic(sequence, i+1, j-1) + 2;

    //Si el primer y el ultimo caracter NO coinciden
    return max( longestPalindromic(sequence, i, j-1), longestPalindromic(sequence,
i+1, j) );
}
```

```
//Main
int main(){
    char sequence[] = "BBABCBCAB";
    int n = strlen(sequence);
    printf ("The length of the LPS is %d", longestPalindromic(sequence, 0, n-1));
    getchar();
    return 0;
}
```

## 4. Análisis de Complejidad

Comenzamos con el análisis de la complejidad con la primera línea del código:

```
// Función para obtener el número mayor entre 2 números
int max (int x, int y) { return (x > y)? x : y; }
```

Tenemos una función para hacer una comparación entre 2 números y obtener el mayor entre ellos, por lo que esto solamente se ejecutará 1 sola vez en todo el código, por lo que su complejidad es constante, o sea:

$O(1)$

Pasando al “Caso Base 1”, como prácticamente cualquier palabra o letra puede ser ingresada en este programa, hay que hacer una condición para las palabras formadas por 1 solo carácter; si en este caso mi contador  $i$  es el mismo que mi contador  $j$  pues entonces tendrá que retornar el valor de 1.

```
int longestPalindromic(char *sequence, int i, int j){
    //Caso base 1: Si sólo existe 1 elemento en la palabra
    if (i == j)
        return 1;
```

Su complejidad:

$O(1)$

Caso Base 2: Al igual que arriba, teníamos la condición de que la palabra estuviera compuesta por 1 carácter, en este caso veremos la condición para las palabras de 2 caracteres y además ambos sean iguales.

```
//Caso base 2: Si hay 2 caracteres en la palabra y además ambos son iguales
    if (sequence[i] == sequence[j] && i + 1 == j)
        return 2;
```

Complejidad:

$O(1)$

Si en dado caso de que el primer y el último caracter coincidan, esta condición me va a retornar la longitud de la secuencia

```
// Si el primero y el último caracter coinciden
    if (sequence[i] == sequence[j])
        return longestPalindromic(sequence, i+1, j-1) + 2;
```

Complejidad

$T(n - 1) + O(1)$

Para el caso de que el primer y el último caracter NO coincidan me retornará la longitud de la secuencia ya comparada para saber si la palabra fue de menos caracteres, regresar es número de caracteres.

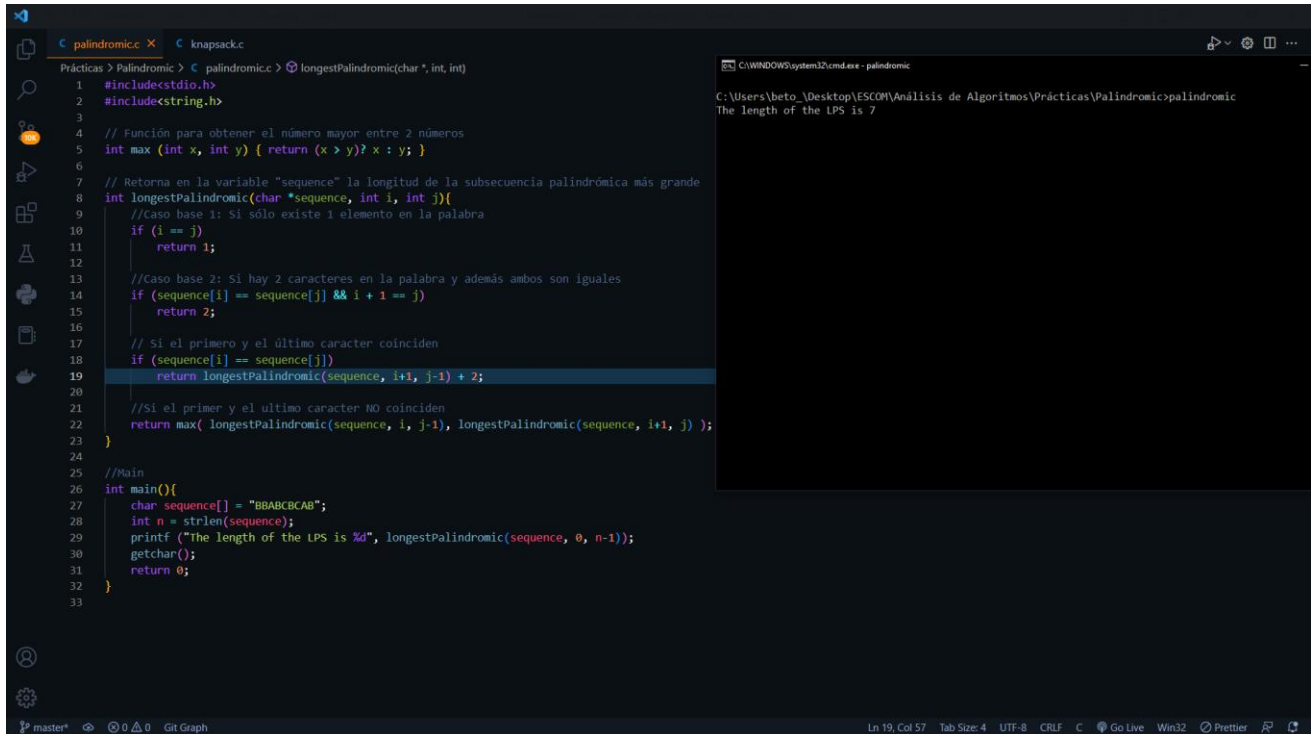
```
//Si el primer y el ultimo caracter NO coinciden
    return max( longestPalindromic(sequence, i, j-1), longestPalindromic(sequence,
i+1, j) );
```

Complejidad

$T(n - 1) + O(1)$

## 5. Pruebas

1er prueba: Tenemos como primer prueba la palabra "BBABCBCAB", entonces la salida debería ser 7 ya que "BABCBAB" es la subsecuencia palindrómica más larga en ella. "BBBBB" y "BBCBB" también son subsecuencias palindrómicas de la secuencia dada, pero no las más largas.



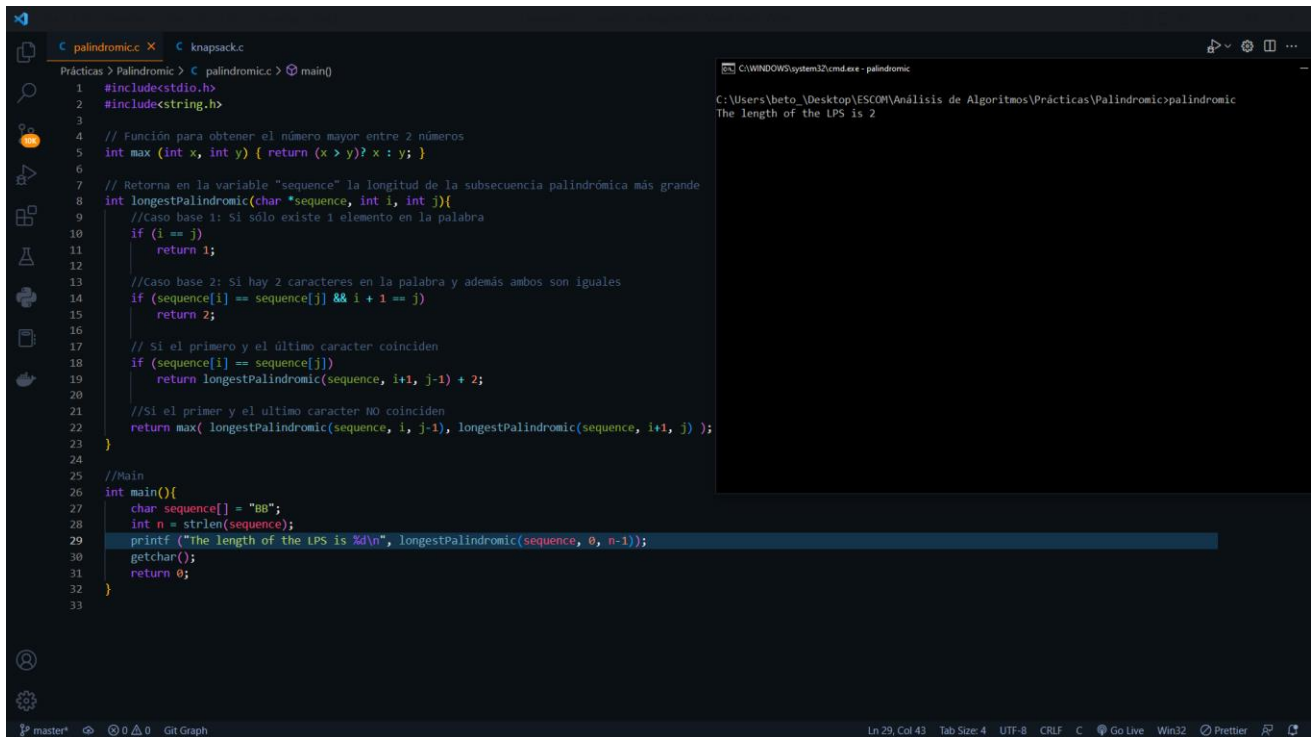
```
1 #include<stdio.h>
2 #include<string.h>
3
4 // Función para obtener el número mayor entre 2 números
5 int max (int x, int y) { return (x > y)? x : y; }
6
7 // Retorna en la variable "sequence" la longitud de la subsecuencia palindrómica más grande
8 int longestPalindromic(char *sequence, int i, int j){
9     //Caso base 1: Si sólo existe 1 elemento en la palabra
10     if (i == j)
11         return 1;
12
13     //Caso base 2: Si hay 2 caracteres en la palabra y además ambos son iguales
14     if (sequence[i] == sequence[j] && i + 1 == j)
15         return 2;
16
17     // Si el primero y el último caracter coinciden
18     if (sequence[i] == sequence[j])
19         return longestPalindromic(sequence, i+1, j-1) + 2;
20
21     //Si el primer y el ultimo caracter NO coinciden
22     return max( longestPalindromic(sequence, i, j-1), longestPalindromic(sequence, i+1, j) );
23 }
24
25 //Main
26 int main(){
27     char sequence[] = "BBABCBCAB";
28     int n = strlen(sequence);
29     printf ("The length of the LPS is %d", longestPalindromic(sequence, 0, n-1));
30     getchar();
31     return 0;
32 }
33
```

C:\WINDOWS\system32\cmd.exe - palindromic

C:\Users\beto\Desktop\ESCOM\Análisis de Algoritmos\Prácticas\Palindromic\palindromic  
The length of the LPS is 7

Ln 19, Col 57 Tab Size: 4 UTF-8 CRLF C Go Live Win32 Prettier

2da prueba: Tenemos una nueva palabra de 2 caracteres, "BB" al detectar que son únicamente 2 caracteres dentro de la función, este tendrá que regresar un "2".



```
1 #include<stdio.h>
2 #include<string.h>
3
4 // Función para obtener el número mayor entre 2 números
5 int max (int x, int y) { return (x > y)? x : y; }
6
7 // Retorna en la variable "sequence" la longitud de la subsecuencia palindrómica más grande
8 int longestPalindromic(char *sequence, int i, int j){
9     //Caso base 1: Si sólo existe 1 elemento en la palabra
10     if (i == j)
11         return 1;
12
13     //Caso base 2: Si hay 2 caracteres en la palabra y además ambos son iguales
14     if (sequence[i] == sequence[j] && i + 1 == j)
15         return 2;
16
17     // Si el primero y el último caracter coinciden
18     if (sequence[i] == sequence[j])
19         return longestPalindromic(sequence, i+1, j-1) + 2;
20
21     //Si el primer y el ultimo caracter NO coinciden
22     return max( longestPalindromic(sequence, i, j-1), longestPalindromic(sequence, i+1, j) );
23 }
24
25 //Main
26 int main(){
27     char sequence[] = "BB";
28     int n = strlen(sequence);
29     printf ("The length of the LPS is %d\n", longestPalindromic(sequence, 0, n-1));
30     getchar();
31     return 0;
32 }
33
```

```
C:\WINDOWS\system32\cmd.exe - palindromic
C:\Users\beto\Desktop\ESCOM\Análisis de Algoritmos\Prácticas\Palindromic>palindromic
The length of the LPS is 2
```

## 6. Conclusiones

Puede que en algún momento hayamos tenido problemas para dar cambio a alguien, que te quedas sin cambio o que no hay cambio, o que tienes mucho cambio pero no sabes qué monedas darle al cliente. Es por eso que se puede considerar un problema NP-completo. El algoritmo te dice cuántas formas puedes completar la tarea de dar cambio.

## 7. Bibliografías

- ✓ (Dasgupta, July 18, 2006, p. 318)
- ✓ <https://www.youtube.com/watch?v=iw-UVkxqHiA>
- ✓ <https://www.youtube.com/watch?v=3THUtOvAFLU>