Complejidad Computacional

Parte 1: Malicious Text

La solución a esta parte de la actividad tiene 3 funciones: *findPatternPosition*, *readFileM* y *findAndDisplayPatter*. De estas funciones la que lleva a cabo la solución es *findPatternPosition*. Esta función recibe un texto (transmisión) y un patrón(mcode), primero checa si el patrón es más largo que el texto, en ese caso regresa *-1*. Utiliza un for loop para recorrer el texto y un if para comparar el patrón con el texto.

Complejidad: int findPatternPosition(const string& text, const string& pattern)

- La comparación del texto de longitud *n* con el patrón(*m)* toma O(m), esta comparación se hace de i a n-m. La complejidad de esta función queda como $O(n * m)$

Parte 2: Palíndromos

Para el desarrollo del punto 2 del proyecto se creó un header llamado "palindrome.h". Dentro de este header podemos encontrar dos funciones. La primera se llama even() y lo que hace es insertar en un vector todos los componentes de los archivos intercalados por "0" para asegurar que sea impar. La segunda función es la que se encarga de encontrar palíndromos dentro del texto e identificar el más largo.

Complejidad even()

Esta función contiene un for loop que se llevará a cabo n veces dependiendo de qué tan grande es el texto.

$Big\ O\ (n)$

Complejidad findPalindrome()

Esta función se caracteriza por iterar sobre todos los elementos del vector y comparar el char derecho e izquierdo para ver si es palindromo. Para realizar estas operaciones se implementó un while loop dentro del for loop. Este while loop continuará su implementación hasta que ya no se encuentre un palíndromo.

$Big\ O\ (n^{\ 2})$

Parte 3: Longest Substring

En esta parte de la actividad se busca encontrar el substring común más largo entre los dos archivos de transmisión. Este substring debe ser continuo en todos los archivos pero no necesariamente del mismo tamaño. La función que resuelve esta parte recibe dos vectores de caracteres y crea una matriz de vectores *int* de acuerdo a los tamaños de ambos vectores. Toma en cuenta la longitud máxima del substring conforme se vayan recorriendo los vectores de caracteres. Utilizando for loops anidados checa si cada carácter es igual y en ese caso le suma 1 a esa posición en la matriz, en caso de que el valor de esa posición sea más grande que la longitud máxima entonces se actualiza y se guarda el índice de la posición final actual.

Complejidad: string LCS(const vector<char>& S1, const vector<char>& S2)

- Crear la matriz tiene una complejidad de O(m*n) donde m es la longitud de S1 y n es la longitud de S2.
- Los for loops anidados tienen una complejidad de m para el for loop más interno y n para el for loop más externo. Por lo tanto su complejidad es de O(m*n)
- Encontrar el substring común más largo también tiene una complejidad de O(m*n)

En general la función tiene una complejidad de tiempo de $O(m * n)$ donde *m* y *n* son las longitudes de S1 y S2 respectivamente.