Tecnológico de Monterrey

E1. Actividad integradora 1

Análisis y diseño de algoritmos avanzados

Victor Manuel de la Cueva

Campus Santa Fe

02 de noviembre de 2023

Fernanda Cantú Ortega - A01782232

Alina Rosas Macedo - A01252720

Parte 1 (búsqueda de subsecuencias)

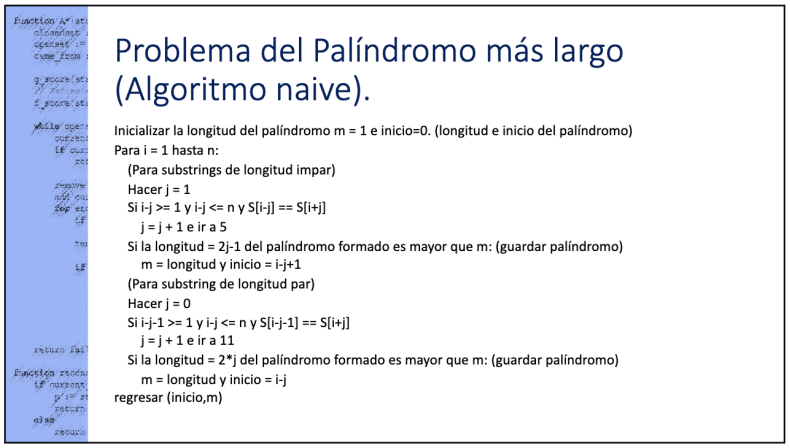
complejidad: $O(n^2)$

Para esta parte del código usamos vectores para leer los archivos porque consideramos que es una buena solución que recorra los vectores a comparación de llamar los archivos múltiples veces, ya que de esta manera es más sencillo imprimir los resultados sin crear variables individuales. En esta función, se utilizan dos loops anidados para recorrer todas las posibles comparaciones entre los archivos de transmisión y los archivos que representan código malicioso y almacena la información de estos en strings. Utiliza el método *"find"* para buscar el string malicioso dentro del string de transmisión y almacena la posición de inicio para posteriormente imprimir true seguido de la posición de inicio de donde se encuentra o false en caso de que no encuentre coincidencias.

Parte 2 (palíndromo más largo en cada archivo)

complejidad: O(n)

En la parte 2, hubo complicaciones al querer implementar el algoritmo Manacher por lo tanto, utilizamos el algoritmo naive. En este algoritmo, se llama el archivo de transmisión que se quiere analizar y recorre el archivo caracter por caracter y se va expandiendo de a la izquierda y a la derecha, comparando si hay caracteres simétricos en *"i"* y "*i-1*". Registra la longitud del palíndromo encontrado y guarda la posición de inicio y finalmente, regresa el rango en el que se encuentra el palíndromo más largo encontrado en el archivo. Es necesario llamar la función dos veces, una por cada archivo que se quiere analizar. Utilizamos como referencia el siguiente pseudocódigo:



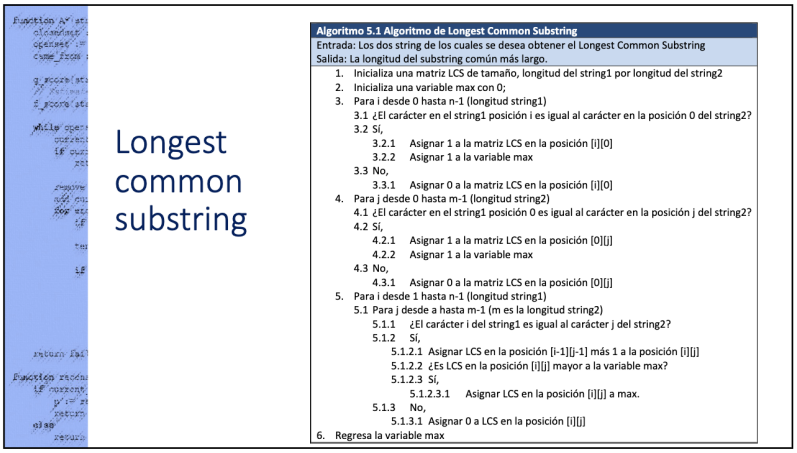
Problema del Palíndromo más largo (Algoritmo naive).

Inicializar la longitud del palíndromo m = 1 e inicio=0. (longitud e inicio del palíndromo)
Para i = 1 hasta n:
  (Para substrings de longitud impar)
  Hacer j = 1
  Si i-j >= 1 y i-j <= n y S[i-j] == S[i+j]
    j = j + 1 e ir a 5
  Si la longitud = 2j-1 del palíndromo formado es mayor que m: (guardar palíndromo)
    m = longitud y inicio = i-j+1
  (Para substring de longitud par)
  Hacer j = 0
  Si i-j-1 >= 1 y i-j <= n y S[i-j-1] == S[i+j]
    j = j + 1 e ir a 11
  Si la longitud = 2*j del palíndromo formado es mayor que m: (guardar palíndromo)
    m = longitud y inicio = i-j
regresar (inicio,m)

Parte 3 (substring más largo en común)

complejidad: O(n * m)

Finalmente para poder encontrar correctamente el substring más largo en común para dos archivos de texto, utilizamos programación dinámica. N y M son las longitudes de los contenidos de los archivos de transmisión. Se crea una matriz para almacenar la longitud del substring más largo y se rastrea la longitud máxima y su posición de inicio. Después se utiliza un loop anidado para comparar los caracteres de ambas cadenas y actualizar la matriz. Finalmente, se imprime el rango en el que se encontró el substring más largo en común o se imprime un mensaje de que no se ha encontrado ninguno en caso de que esto suceda. Utilizamos como referencia el siguiente pseudocódigo:



Longest common substring

Algoritmo 5.1 Algoritmo de Longest Common Substring
Entrada: Los dos string de los cuales se desea obtener el Longest Common Substring
Salida: La longitud del substring común más largo.

1. Inicializa una matriz LCS de tamaño, longitud del string1 por longitud del string2
2. Inicializa una variable max con 0;
3. Para i desde 0 hasta n-1 (longitud string1)
   3.1 ¿El carácter en el string1 posición i es igual al carácter en la posición 0 del string2?
   3.2 Sí,
      3.2.1 Asignar 1 a la matriz LCS en la posición [i][0]
      3.2.2 Asignar 1 a la variable max
   3.3 No,
      3.3.1 Asignar 0 a la matriz LCS en la posición [i][0]
4. Para j desde 0 hasta m-1 (longitud string2)
   4.1 ¿El carácter en el string1 posición 0 es igual al carácter en la posición j del string2?
   4.2 Sí,
      4.2.1 Asignar 1 a la matriz LCS en la posición [0][j]
      4.2.2 Asignar 1 a la variable max
   4.3 No,
      4.3.1 Asignar 0 a la matriz LCS en la posición [0][j]
5. Para i desde 1 hasta n-1 (longitud string1)
   5.1 Para j desde a hasta m-1 (m es la longitud string2)
      5.1.1 ¿El carácter i del string1 es igual al carácter j del string2?
      5.1.2 Sí,
         5.1.2.1 Asignar LCS en la posición [i-1][j-1] más 1 a la posición [i][j]
         5.1.2.2 ¿Es LCS en la posición [i][j] mayor a la variable max?
         5.1.2.3 Sí,
            5.1.2.3.1 Asignar LCS en la posición [i][j] a max.
      5.1.3 No,
         5.1.3.1 Asignar 0 a LCS en la posición [i][j]
6. Regresa la variable max