Campus Monterrey

Escuela de ingeniería y ciencias

Unidad de formación TC2038.601

Profesor:
Luis Humberto González Guerra

Evidencia 1

Grupo:601:

Fernando Morán Fougerat A01284623

Actividad realizada junto a:
Julen Hoppenstedt Mandiola A01174098

Fecha de entrega 6 de octubre del 2024

Reflexión Individual

Dentro del presente entregable trabaje de manera colaborativa junto con mi compañero Julen para resolver el problema de detectar códigos maliciosos dentro de diversos archivos de texto que representaron el envío de datos de un dispositivo a otro. Mi experiencia dentro de mis estudios me permitió entender que en este caso, la manera en la que podíamos realizar nuestros pedidos era con algoritmos de String que habíamos visto en clase, más precisamente el algoritmo de Knuth-Morris-Pratt (KMP), el de Manacher, y el de Longest Common Substring . Todos los algoritmos que acabo de mencionar involucraron la técnica de Programación Dinámica (DP), los cuales nos permitieron llegar a un resultado de manera más eficiente y con un resultado correcto.

Inicialmente, se nos pidió detectar, para cada línea de texto dentro de un archivo malicioso, la cantidad de incidencias en cada uno de los tres archivos de trasmisión dados. Para resolver este algoritmo, usamos el de KMP, el cual tiene una complejidad de tiempo y espacio de O(m + n) m siendo la longitud del primer texto a comparar y n siendo la longitud del segundo (código malicioso). En este, se opta por tener una tabla de prefijos (lpsArray) para evitar comparaciones redundantes cuando no se puede tener una palabra en común, y continuamos la búsqueda desde el lugar correcto sin reiniciar la búsqueda completa en el texto. En nuestra función main, después de buscar todos los registros del string completo, generamos subsecuencias de 1 char menos, y usamos de igual manera KMP para detectar cuál fue el substring más repetido, cuantas veces y en donde hay más. Esto funcionará para detectar los posibles códigos maliciosos en nuestros mensajes, de una manera veloz y eficiente.

Para la siguiente tarea, se nos encargó detectar código "espejeado" , es decir palíndromos de chars. Para esto, usamos el algoritmo de Manacher, el cual primero crea un string donde cada letra del original está separado por un char especial no usado originalmente (en nuestro caso podría ser cualquier letra después de la F alfabéticamente, pero optamos por seguir el modelo original y usamos ‘#’) para hacer siempre impar nuestro string, y de ahí buscar del centro el mayor palíndromo. Para cada posición, si la posición actual está dentro del palíndromo más largo encontrado hasta ese punto delimitado por valor de variable right centrados en nuestro índice center, se usa el valor ya calculado de su espejo para reducir el número de comparaciones. Después, el palíndromo se intenta expandir desde el centro hacia los lados, actualizando nuestros límites si se encuentra un palíndromo mayor en los límites (son iguales ambos nuevos chars). Este algoritmo funciona con complejidad de tiempo y espacio O(n), ya que da barrido lineal a string gracias al uso de DP mediante el vector P para inicializar valores de palíndromo actual y continuar con pasada. Aún si es de doble tamaño el string (2n por agregar el char especial), como se aumenta de manera lineal según el tamaño, el valor de ambas complejidades en Big O notation es solo O(n).

Por último, se nos pidió analizar que tan similares son los tres archivos de transmisión con combinaciones de 2. Para resolver este ejercicio usamos el algoritmo de Longest Common Substring, usando una matriz de tamaño nxm (tamaño de string1* tamaño string2) para almacenar las longitudes de las coincidencias encontradas, llenando primero las filas y

columnas con valores iniciales de nuestros string y luego iteramos sobre ambas cadenas, actualizando la matriz y rastreando la longitud máxima de la LCS y su posición final, comparando si es mayor para actualizar el índice (en un string) y tamaño (para construir el subst al terminar nuestra función con valor final). Al ir pasando, vamos agregando a nuestra matriz en cada elemento el valor de la pasada (para que vaya aumentando y reducir volver a computar todo el tamaño) o lo regresamos a 0 si se rompe una subcadena. Fue una solución eficiente a nuestro problema, y logramos encontrar correctamente nuestros mayores substrings en los diferentes casos que buscamos.

Esta actividad fue bastante buena para reforzar mis conocimientos sobre algoritmos de String, y principios de programación dinámica, y estoy seguro que el seguir practicando me ayudará a desarrollar competencias clave como lo son Problem solving, y analisis analitico para encontrar la mejor solución con el catálogo de algoritmos que he estudiado.