TECNOLÓGICO DE MONTERREY®

TC2038 – ANÁLISIS Y DISEÑO DE ALGORITMOS AVANZADOS
GRUPO 603

Actividad Integradora 1

Eunice Santos Galindo – A00831991
Brenda Elena Saucedo González – A00829855

28 de septiembre de 2022

Introducción

Cuando se transmite información de un dispositivo a otro, se transmite una serie sucesiva de bits, que llevan una cabecera, datos y cola. Existe mucha gente mal intencionada, que puede interceptar estas transmisiones, modificar estas partes del envío, y enviarlas al destinatario, incrustando sus propios scripts o pequeños programas que pueden tomar cierto control del dispositivo que recibe la información.

La presente situación problema consiste en solucionar tres subproblemas, en donde tenemos que interceptar código malicioso y código “espejeado” (palíndromo) dentro de una transmisión, además de que también se debe de interceptar el substring más largo y común entre ambas transmisiones enviadas.

En lo que respecta, nuestro programa presenta diversos algoritmos empleados para la solución de la situación problema planteada, los cuales explicaremos a continuación.

Algoritmos Empleados

KMP

La búsqueda de patrones es un problema importante en informática. Cuando buscamos una cadena en un archivo de texto o en el navegador o la base de datos, se utilizan algoritmos de búsqueda de patrones para mostrar los resultados de la búsqueda.

El algoritmo Knuth-Morris-Pratt, más conocido como KMP, es un algoritmo de búsqueda de subcadenas, es decir, su objetivo es buscar la existencia de una subcadena (habitualmente llamada patrón) dentro de otra cadena.

La implementación de nuestro algoritmo toma como parámetro dos variables de tipo string (cadena de caracteres), las cuales son el texto a analizar y el patrón a buscar, y retorna un vector de enteros, en donde se almacenan las posiciones de las coincidencias encontradas. Cabe mencionar que por el tipo de problema, decidimos que el algoritmo tome los saltos de línea como un carácter más, y por consiguiente, como una posición más.

Por el tipo de algoritmo y su implementación, su complejidad es de O(n + m), en donde “n” depende de la longitud del texto a analizar y “m” depende de la longitud del patrón.

Manacher

En la situación problema se planteó que en ocasiones, los códigos maliciosos tienen código "espejeado" (palíndromos de chars), por lo que se vuelve un requerimiento analizar los archivos de textos para buscar palíndromos dentro de este.

El algoritmo de Manacher es un algoritmo de búsqueda de la subcadena palindrómica más larga, es decir, su objetivo es buscar la existencia de una subcadena palindrómica y que esta sea la de mayor longitud, dentro de una cadena dada.

La implementación de nuestro algoritmo toma como parámetro una variable de tipo string, la cual es el texto a analizar, y retorna información sobre el palíndromo encontrado, como lo son su posición inicial y final en el texto, y el palíndromo como tal.

Por el tipo de algoritmo y su implementación, su complejidad es de O(n), en donde “n” depende de la longitud del texto a analizar.

Longest Common Substring (DP approach)

El eje principal de la parte 3 es encontrar la subcadena más larga común entre ambos archivos, partimos de la idea de que todo el texto contenido en cada archivo forma parte de la misma cadena a comparar para buscar el LCS. Intentar un método de fuerza bruta no sería eficiente debido al número de comparaciones entre substrings, además, sería un gasto en memoria innecesario. Para reducir el tiempo de ejecución se plantea el siguiente enfoque de programación dinámica en “Bottom Up” (ascendente) para encontrar el LCS entre dos cadenas particulares.

La idea se basa en encontrar la longitud del sufijo común más largo para todos los substrings de ambos strings y almacenar estas longitudes en una tabla. Establecemos la primer cadena como X y la segunda como Y.

Cada valor LCSuffix[i][j] se obtiene de dos maneras, en caso de que X[i-1] sea igual a Y[j-1], entonces el valor asignado será LCSuffix[i-1][j-1] + 1, si no se cumple la condición de comparación entonces el valor asignado en la tabla es 0. Al hacer este proceso disminuyendo los índices i y j, se debe asegurar que en todo momento su valor sea mayor o igual a cero, y no mayor al tamaño de su cadena correspondiente.

Se cuenta con una variable en la que se irá actualizando el valor máximo de longitud. Este valor se calcula comparando el máximo actual y el calculado en LCSuffix[i][j] y manteniendo el valor mayor.

Adaptación 1: Como en la implementación al hacer el cálculo solo se requiere saber el contenido de la última fila, podemos mantener únicamente dos filas consecutivas y optimizar el espacio (Nueva matriz: len[2][n+1]). Además, usamos una variable que indique cual es la fila actual (currRow).

Adaptación 2: Para conocer las posiciones inicial y final del substring en el archivo de transmisión 1, se ocupa una variable índice final de la cadena X en el que cada vez que se actualiza el valor que guarda el tamaño máximo hasta ahora, también se actualiza el valor del end a i-1.

Al final las posiciones del substring en el archivo de transmisión 1 (X) serán: desde [end-(tamaño máximo) + 1] hasta [end].

Cabe mencionar que por el tipo de problema, decidimos que el algoritmo tome los saltos de línea como un carácter más, y por consiguiente, como una posición más.

Por el tipo de algoritmo y su implementación, su complejidad es de O(m*n), en donde “n” depende de la longitud del primer string y “m” depende de la longitud del segundo string.

Conclusión

Como conclusión, comprendemos que existe una gran diversidad de algoritmos que manejan y manipulan cadenas de texto, de los cuales algunos podrían ser más eficientes que otros, sin embargo, el desarrollo e implementación de los algoritmos de los que como equipo hicimos uso, fueron algunos de los cuales adquirimos conocimiento en el curso actual, los cuales mejoramos para que se pudieran adaptar a la presente situación problema.

Somos conscientes de que estos tipos de algoritmos pueden ser aplicados en una gran diversidad de contextos, no sólo para encontrar coincidencias en códigos que pudiesen ser maliciosos o dañinos, sino que también se pueden aplicar para encontrar patrones, realizar análisis de textos, búsquedas, etc.

Referencias

Anónimo. (s.f.). *Situación problema 1*. Septiembre 21, 2022, de ITESM. Sitio web: https://experiencia21.tec.mx/courses/313041/pages/situacion-problema-1-2?module_item_id=18747241

Anónimo. (s.f.). *Algoritmo KMP para la búsqueda de patrones*. Septiembre 21, 2022, de ITESM. Sitio web: https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/

Anónimo. (s.f.). *Longest common substring: DP-29.* Septiembre 21, 2022, de ITESM. Sitio web: https://www.geeksforgeeks.org/longest-common-substring-dp-29/

Colaboradores de Wikipedia. (s.f.). *Algoritmo Knuth-Morris-Pratt*. Septiembre 21, 2022, de Wikipedia. Sitio web: https://es.wikipedia.org/wiki/Algoritmo_Knuth-Morris-Pratt