Tecnológico de Monterrey

Actividad Integradora 1

Jaime Esteban Ochoa De La Torre - A01643234

Edgardo Medina Miranda - A01614376

Patricio Blanco Rafols - A01642057

Análisis y diseño de algoritmos avanzados (Gpo 604)

Adolfo Ernesto Arroyo Alanis

Fecha de entrega:
21 de Septiembre del 2025

Tecnológico de Monterrey

Introducción

El documento tiene como objetivo analizar las soluciones algorítmicas implementadas para resolver la "Actividad Integradora 1". Se abordará cada una de las tres partes del problema, describiendo el algoritmo utilizado, su complejidad computacional (Big O), y se discutirán algoritmos alternativos que podrían ofrecer una mayor eficiencia, especialmente al trabajar con conjuntos de datos a gran escala.

Parte 1: Búsqueda de Código Malicioso (Búsqueda de Subcadenas)

Algoritmo Implementado

Para determinar si el contenido de los archivos mcode se encontraba dentro de los archivos transmission, se utilizó el método find de la biblioteca estándar. Esta función recorre la cadena principal en busca de la primera ocurrencia de la subcadena proporcionada.

Complejidad Computacional

El algoritmo de fuerza bruta, que es la base de implementaciones sencillas como find, tiene una complejidad computacional en el peor de los casos de **O(n * m)**, donde:

- n es la longitud de la cadena de transmisión.
- m es la longitud del código malicioso a buscar.

Este escenario ocurre cuando se requieren comparaciones repetidas a lo largo de la cadena principal.

Algoritmo Alternativo (Óptimo)

Un enfoque más eficiente es el **Algoritmo Knuth-Morris-Pratt (KMP)**. Este algoritmo mejora drásticamente el rendimiento al preprocesar la subcadena a buscar (mcode) para crear una tabla de "prefijos parciales" (LPS). Esta tabla permite que, al ocurrir una discrepancia, el algoritmo sepa exactamente cuántos caracteres puede saltar hacia adelante sin necesidad de volver a comparar caracteres que ya se sabe que coincidirán.

- **Complejidad de KMP: O(n + m)**. Esta complejidad lineal representa una optimización significativa, ya que el tiempo de ejecución crece en proporción a la suma de las longitudes de las cadenas, no a su producto.

Parte 2: Búsqueda del Palíndromo Más Largo

Algoritmo Implementado

La solución implementada utiliza un enfoque de fuerza bruta. Consiste en generar sistemáticamente cada posible subcadena dentro del texto de transmisión y, para cada una, verificar si es un palíndromo. Se mantiene un registro de la subcadena palindrómica más larga encontrada hasta el momento.

Complejidad Computacional

Este método es computacionalmente costoso. La generación de todas las subcadenas requiere dos bucles anidados, lo que resulta en una complejidad de $O(n^2)$. La verificación de si cada subcadena (de longitud k) es un palíndromo toma O(k) tiempo. Por lo tanto, la complejidad total de este enfoque es **$O(n^3)$**.

Algoritmo Alternativo (Óptimo)

El algoritmo más eficiente conocido para este problema es el **Algoritmo de Manacher**. Es una solución sofisticada que logra encontrar el palíndromo más largo en tiempo lineal. Lo consigue aprovechando la simetría de los palíndromos y reutilizando información de palíndromos ya encontrados para expandirse y evitar cálculos redundantes.

- **Complejidad de Manacher: O(n)**. Esta es la solución óptima y es drásticamente más rápida que la fuerza bruta, especialmente para archivos de transmisión de gran tamaño.

Parte 3: Búsqueda del Substring Común Más Largo

Algoritmo Implementado

Para esta tarea, se implementó un algoritmo de **Programación Dinámica**. Se construyó una matriz bidimensional (tabla DP) de tamaño (n+1) x (m+1), donde n y m son las longitudes de las dos cadenas de transmisión. Cada celda tabla[i][j] almacena la longitud del substring común que termina exactamente en los caracteres s1[i-1] y s2[j-1].

Complejidad Computacional

El enfoque de programación dinámica requiere llenar cada celda de la matriz, realizando una comparación por celda. Por lo tanto, su complejidad computacional es **O(n * m)**.

Evaluación del Algoritmo

A diferencia de los casos anteriores, la solución de Programación Dinámica implementada es el enfoque estándar y ampliamente aceptado para resolver el problema del Substring Común Más Largo. Aunque existen otras estructuras de datos avanzadas como los Árboles de Sufijos (Suffix Trees) que pueden resolver el problema en tiempo O(n+m), la implementación con DP es robusta, más sencilla de codificar y muy eficiente para la mayoría de los casos prácticos.

Conclusión

La actividad permitió implementar soluciones funcionales para tres problemas clásicos en ciencias de la computación. El análisis de complejidad revela que, si bien las soluciones para las Partes 1 y 2 son correctas, existen algoritmos (KMP y Manacher) considerablemente más eficientes que serían cruciales en un entorno de producción con datos masivos. Para la Parte 3, la solución de Programación Dinámica implementada ya representa un estándar de alta eficiencia. Esta reflexión subraya la importancia crítica de seleccionar el algoritmo adecuado para garantizar no solo la corrección, sino también la escalabilidad y el rendimiento de una solución de software.