Análisis de transmisiones de datos y detección de código malicioso en C++

Andre Milan Guzman Ramos - A01199013

Fernando Gael Hernández Salazar - A01029264

Carlos Daniel Lara - A01711690

Ignacio Rey Arslangul - A01178392

Este trabajo presenta el desarrollo de un programa en C++ diseñado para analizar transmisiones digitales y detectar posibles códigos maliciosos. Se utilizan cinco archivos de texto, dos de transmisiones y tres que contienen secuencias sospechosas. El sistema busca subsecuencias para determinar si los códigos maliciosos aparecen dentro de las transmisiones e identifica su posición inicial. También localiza el palíndromo más largo en cada transmisión y reporta su inicio y fin. Además identifica el substring común más largo entre ambas transmisiones y muestra su ubicación en el primer archivo. Finalmente implementa compresión mediante Huffman Coding para calcular la longitud promedio de codificación y evaluar si la codificación de los archivos sospechosos resulta anormalmente alta. El programa combina así la búsqueda de patrones con técnicas de compresión para ofrecer un análisis integral de seguridad en datos digitales.

Palabras clave y frases adicionales: detección de subsecuencias, búsqueda de palíndromos, subcadena común más larga, codificación Huffman, detección de malware, análisis de transmisión de datos, programación en C++

1 INTRODUCCIÓN

El análisis de transmisiones digitales se ha convertido en una tarea esencial para garantizar la seguridad de los sistemas de comunicación. La detección temprana de código malicioso dentro de grandes volúmenes de datos requiere no solo técnicas de búsqueda eficientes, sino también un entendimiento profundo de la complejidad computacional de los algoritmos empleados. En este trabajo se estudian diferentes enfoques aplicados a la identificación de subsecuencias sospechosas, la localización de palíndromos significativos y la comparación de similitudes entre transmisiones mediante la búsqueda del substring común más largo.

Adicionalmente, se incorpora la compresión basada en Huffman Coding como una estrategia para mejorar la eficiencia en tiempo y espacio. Este enfoque no solo permite optimizar el almacenamiento y transmisión de datos, sino que también ofrece un mecanismo complementario para evaluar anomalías en la codificación de posibles secuencias maliciosas. La reflexión incluye la explicación de cada algoritmo aplicado a la situación problema, acompañado de su complejidad computacional y un análisis comparativo que justifica las ventajas de integrar la compresión en el proceso de detección.

2 ALGORITMOS

2.1 Búsqueda de subsecuencias

En esta primera parte del problema, el algoritmo funciona como un buscador de patrones dentro de las transmisiones. La idea es recorrer el archivo de transmisión desde el inicio hasta el final e ir comparando, carácter por carácter, con el código malicioso. Si en algún punto todos los caracteres del código coinciden de manera consecutiva con los de la transmisión, se concluye que efectivamente está presente y se devuelve un "true" junto con la posición exacta donde comienza. Si se recorre toda la transmisión sin encontrar una coincidencia completa, la salida es "false". Esto refleja un enfoque directo y sencillo para verificar si un patrón se encuentra contenido dentro de una secuencia más grande, lo cual tiene sentido en el contexto de detectar fragmentos de malware en un flujo de datos.

La complejidad de este procedimiento depende de dos factores: la longitud de la transmisión y la longitud del código malicioso. En el peor de los casos, el algoritmo intenta alinear el patrón en casi todas las posiciones posibles de la transmisión, y en cada intento puede llegar a comparar carácter por carácter hasta recorrer todo el patrón. Eso da como resultado una complejidad de tiempo proporcional al producto de ambos tamaños, es decir, del orden de $O(n \cdot m)$, donde n es la longitud de la transmisión y m la longitud del código. En el mejor escenario, si la coincidencia se encuentra muy pronto o los caracteres no llegan a coincidir en la primera comparación, el costo se reduce a un recorrido lineal sobre la transmisión. En cuanto a memoria, no requiere estructuras adicionales más allá de unas cuantas variables, lo que lo hace un algoritmo de espacio constante, $O(1)$.

2.2 Búsqueda de palíndromos

Para resolver el problema de identificar el palíndromo más largo dentro de un archivo de transmisión, se implementaron dos versiones del algoritmo: una que considera todos los caracteres de la entrada, incluyendo espacios, saltos de línea y tabulaciones, y otra que ignora los caracteres de espacio en blanco, mapeando el resultado a las posiciones originales en la cadena. Ambas se basan en la técnica de Manacher, un algoritmo especializado que permite encontrar el palíndromo más largo en un string en tiempo lineal.

En la primera versión, se recibe la cadena de entrada tal cual y se transforma para facilitar la búsqueda de palíndromos de longitud par e impar de manera unificada. Para ello se intercalan caracteres separadores (#) entre los caracteres originales y sentinelas al inicio y al final (^, $). De esta forma, la cadena "abba" se convierte en ^#a#b#b#a#$, lo que permite tratar todos los palíndromos como si fueran de longitud impar. Durante la ejecución, el algoritmo mantiene dos valores: el centro del palíndromo que actualmente se expande más a la derecha, y el límite derecho de ese palíndromo. Cada nueva posición se compara con su espejo respecto al centro actual. Con esta simetría, es posible evitar reexplorar regiones ya conocidas, lo que garantiza eficiencia. La posición y longitud del palíndromo máximo se actualizan en cada paso. Al finalizar, se calculan los índices originales (1-based) para dar el resultado. La complejidad temporal de este algoritmo es O(n), donde n es el tamaño de la cadena, y la complejidad espacial también es O(n), ya que se almacena un arreglo con los radios de los palíndromos y la cadena transformada.

La segunda versión extiende la idea anterior pero con un preprocesamiento adicional: antes de ejecutar Manacher, se limpia la cadena eliminando todos los caracteres de espacio, tabulación y salto de línea. Durante este paso también se genera un mapeo de posiciones que relaciona cada carácter de la cadena limpia con su posición en la cadena original.

Posteriormente, sobre esta cadena limpia se ejecuta Manacher de la misma forma que en el primer algoritmo. Una vez encontrado el palíndromo más largo, sus índices se traducen de nuevo a posiciones en la cadena original utilizando el mapeo. Esto asegura que los resultados estén expresados en las coordenadas del archivo de transmisión, pero sin verse afectados por los espacios en blanco. La complejidad temporal de este enfoque también es O(n), ya que el filtrado de espacios y la ejecución de Manacher son ambos lineales respecto al tamaño de la entrada. La complejidad espacial es O(n), por la necesidad de almacenar tanto la cadena limpia como el vector de mapeo y el arreglo auxiliar de radios.

La ventaja de Manacher sobre soluciones más ingenuas, como comparar todos los substrings posibles, radica en que evita el uso de dos ciclos anidados que resultan en una complejidad cuadrática $O(n^2)$. De esta forma, incluso para transmisiones largas, se garantiza un desempeño eficiente. El uso de la versión que ignora espacios es particularmente útil en contextos de análisis de logs o flujos de texto donde los caracteres de formato no son relevantes, permitiendo obtener un resultado más fiel a las secuencias de interés.

2.3 Substring común más largo

Para la tercera parte del problema, se implementó un algoritmo de programación dinámica para conocer el substring común más largo entre los dos archivos de transmisión. El programa obtiene los dos archivos ya leídos y los compara. Primero guarda el tamaño de las cadenas y crea una matriz con base en estos tamaños. Al mismo tiempo, guarda la máxima longitud de la subcadena encontrada hasta el momento, y lleva un registro de la posición en la que esa cadena termina. Los ciclos for, recorrer ambos strings, cada vez que se encuentre una coincidencia, el substring encontrado y los valores de la longitud y de la última posición aumentan en 1. En caso de que estos valores superen el registro previo se guardan. Al final del ciclo se guarda el máximo valor alcanzado tanto de última posición como de longitud.

Una vez terminado el ciclo se calcula la posición inicial del substring, restando la posición final menos la longitud. Después se obtiene la cadena que coincidió a partir de los valores obtenidos de posición inicial y longitud. Finalmente se imprime la cadena, la posición inicial y la final.

Es importante considerar que el algoritmo toma por igual los caracteres de espacio, por lo que al encontrarse con uno, siempre y cuando ambas cadenas lo tengan, lo tomará como una coincidencia, por lo que el carácter de espacio realmente no “rompe” la cadena, más bien puede formar parte de ella. Es por eso que el resultado de esta parte para la comparación de ambos archivos es “AAAABBBBCCCC 1234DEADBEEF5678CAFEBABE9876” tomando en cuenta el espacio como una coincidencia.

Ya que el algoritmo se basa en dos ciclos for para recorrer cada substring respecto a su tamaño, el algoritmo tiene una complejidad de tiempo = $O\ (\ n \cdot m\ )$ y de la misma forma, ya que se genera una matriz a partir de los tamaños de ambos strings tenemos una complejidad de espacio igual a $O\ (\ n \cdot m\ )$.

2.4 Huffman Coding

En esta parte del problema se desarrolló un detector de códigos sospechosos utilizando Huffman Coding. El objetivo principal fue analizar transmisiones de texto y determinar si un mensaje presenta anomalías en función de la distribución de frecuencias de sus caracteres. Para ello, se implementaron diversos algoritmos que trabajan en conjunto: cálculo de frecuencias, construcción del árbol de Huffman, generación de códigos binarios, codificación de mensajes y verificación de condiciones de sospecha

El primer paso consiste en el cálculo de frecuencias, que recorre la transmisión caracter por caracter y cuenta sus apariciones. Este proceso tiene complejidad O(n), siendo n el número de caracteres en la transmisión. A continuación, se realiza la construcción del árbol de Huffman usando una cola de prioridad (min-heap), en la cual se combinan repetidamente los nodos con menor frecuencia hasta formar un único árbol. Este procedimiento presenta complejidad O(m log m), donde m corresponde al número de caracteres distintos. Una vez generado el árbol, se ejecuta un recorrido recursivo para la **generación de códigos binarios**, asignando a cada carácter un código según su posición en el árbol. Este paso requiere **O(m)** operaciones. Después, la **codificación de los mensajes** sustituye cada carácter por su código Huffman, lo que implica un nuevo recorrido del texto original con complejidad **O(n)**. Finalmente, el algoritmo de **detección de sospechosos** compara la longitud real del mensaje con la longitud promedio esperada, y revisa la presencia de caracteres raros o desconocidos. Este análisis también es **O(n)**. En conjunto, el detector implementado tiene una complejidad de **O(n + m log m)**, siendo eficiente ya que en la práctica $m \ll n$.

El enfoque de programación dinámica fue elegido porque permite reducir el trabajo redundante en las comparaciones. En lugar de recalcular subproblemas múltiples veces, la matriz garantiza que los resultados parciales se reutilicen de forma eficiente, lo cual mejora considerablemente el rendimiento. Aunque la complejidad temporal y espacial resultan en $O(n \cdot m)$, esta técnica es más confiable que otros enfoques como fuerza bruta, ya que asegura la obtención del substring más largo de manera correcta y ordenada

Justificación de mejora con Huffman Coding

El uso de Huffman Coding aporta una mejora significativa en comparación con la codificación fija (por ejemplo, ASCII con 8 bits por carácter). Al asignar códigos más cortos a los símbolos frecuentes y más largos a los raros, se obtiene una **longitud promedio menor**, lo cual reduce el espacio requerido para representar el mensaje. Más importante aún, esta característica sirve como base estadística para la detección: si un mensaje ocupa mucho más de lo esperado, o si contiene una proporción elevada de caracteres poco comunes, el sistema puede marcarlo como sospechoso. En conclusión, Huffman Coding no solo optimiza la compresión, sino que también provee una **métrica objetiva y probabilística** para identificar transmisiones anómalas, mejorando la capacidad de detección del sistema.

3 CONCLUSIÓN

El desarrollo del programa permitió integrar de manera efectiva distintas técnicas de análisis y compresión orientadas a la detección de patrones sospechosos en transmisiones digitales. La implementación de algoritmos de búsqueda de subsecuencias, palíndromos y substrings comunes más largos demostró la importancia de seleccionar métodos con diferentes niveles de eficiencia según el problema a resolver. Por otro lado, la incorporación de Huffman Coding demostro cómo los enfoques de compresión no solo optimizan el uso de recursos, sino que también pueden convertirse en una herramienta para identificar anomalías que apunten a la presencia de código malicioso. En conjunto, este trabajo muestra que la seguridad en el manejo de datos requiere soluciones que combinen algoritmos clásicos de búsqueda con técnicas de optimización y análisis estadístico, ofreciendo así un marco más robusto para la. detección temprana de amenazas en entornos digitales.