Tecnológico de Monterrey

Campus Queretaro

Situación Problema

Actividad Integradora 1

Curso:

Análisis y diseño de algoritmos avanzados

Grupo 601

Luis Enrique Gutierrez Mendioroz - A00837481

Iñaki Salvador Perez Lozada -

Claudio Gabriel Lopez Briceño - A01710963

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

Para abordar esta problemática nuestra solución implementa tres algoritmos que realizan análisis fundamentales para evitar ataques cibernéticos.

En este documento se encontrará detalladamente las aplicaciones de cada uno de los algoritmos con el siguiente orden:

1. Identificación de patrones maliciosos (Algoritmo de Knuth Morris Pratt)
2. Identificación de código espejado (Algoritmo de Manacher)
3. Identificación de código común entre transmisiones (LCS - Longest common substring)

Búsqueda de Subsecuencias

Introducción:

Se emplea el algoritmo de Knuth Morris Pratt (KMP) cuyo objetivo es encontrar patrones en cadenas de texto. En la aplicación de dicho algoritmo se revisa que patrón se repite en los archivos dados y por ende podemos sacar conclusiones sobre si el archivo contiene algo malicioso dentro.
Tenemos como entrada cinco archivos y seis salidas de true o false dependiendo el caso.

Implementación:

Matriz LPS

La entrada consiste en cinco archivos y la salida son seis valores booleanos (true o false), dependiendo de si se detecta el patrón buscado.
Un elemento clave en la implementación del algoritmo es la matriz LPS (Longest Proper Prefix which is also Suffix). Esta matriz optimiza el proceso de búsqueda de un patrón, ya que permite al algoritmo avanzar posiciones sin necesidad de repetir comparaciones innecesarias.

Funcionamiento:

El funcionamiento se basa en el uso de prefijos y sufijos de la cadena:

- Prefijo: parte inicial de la cadena (puede incluir la palabra completa).
- Sufijo: caracteres que constituyen el final de la cadena.
- Prefijo propio: igual que el prefijo, pero excluyendo el último carácter.

El algoritmo compara y almacena cuántos caracteres coinciden entre prefijos y sufijos de la cadena. Gracias a esto, en caso de encontrar un desajuste, no es necesario reiniciar la búsqueda desde el comienzo, sino que se retoma desde el punto más óptimo. Esto convierte al KMP en un algoritmo eficiente y adecuado para detectar rápidamente patrones que puedan representar amenazas de ciberseguridad.

Por último, se utiliza directamente en el algoritmo KMP donde al ir recorriendo las cadenas de texto y en caso de encontrar concordancia guardaremos el índice donde existe el patrón.

Complejidad Temporal

En este algoritmo utilizamos dos variables '*n*' (corresponde a la longitud del texto) y '*m*' (corresponde a la longitud del patrón).
Donde el tiempo de espera(complejidad) es proporcional a la longitud de ambos, es la suma de ambos componentes, O(m+n). Lo que quiere decir que al crear la matriz utilizamos O(m) y la búsqueda en el texto utiliza O(n).
En las pruebas realizadas, demostró ser un algoritmo muy rápido para analizar los texto y devolver los resultados.

Conclusión del algoritmo

El algoritmo KMP cumplió con las expectativas en términos de complejidad, ya que permite analizar cadenas de texto de manera rápida y eficiente. A diferencia de otros métodos de búsqueda, como el algoritmo de Boyer–Moore, cuya complejidad en el peor caso puede llegar a O(n×m), KMP mantiene un tiempo de ejecución garantizado de O(n+m). Esto refuerza la idea de que KMP es una elección adecuada para tareas

en las que se requiere un rendimiento constante y confiable en la detección de patrones dentro de grandes volúmenes de texto.

Búsqueda de Palíndromos

Introducción

Se emplea el algoritmo de Manacher cuyo objetivo es encontrar el palíndromo más largo dentro de una cadena de texto. En la aplicación de dicho algoritmo se identifica la subsecuencia más extensa que se lee igual de izquierda a derecha que de derecha a izquierda, lo cual puede ser útil para detectar patrones simétricos en transmisiones de datos o identificar secuencias repetidas que puedan indicar anomalías.

La entrada consiste en archivos de transmisión y la salida son dos valores enteros que representan las posiciones de inicio y fin del palíndromo más largo encontrado.

Implementación

Transformación de la cadena

Un elemento clave en la implementación del algoritmo de Manacher es la transformación de la cadena original mediante la inserción de caracteres separadores. Esta transformación permite tratar de manera uniforme tanto palíndromos de longitud par como impar.

Funcionamiento:

La cadena original se transforma insertando un carácter especial (en este caso '|') entre cada par de caracteres. Por ejemplo:

- Cadena original: `"aba"`

- Cadena procesada: `"|a|b|a|"`

Esta transformación garantiza que todos los palíndromos tengan un centro explícito, simplificando el proceso de búsqueda.

Vector de radios

El algoritmo mantiene un vector llamado `longitudes` que almacena, para cada posición de la cadena procesada, el radio del palíndromo centrado en esa posición. El radio representa cuántos caracteres a cada lado del centro forman parte del palíndromo.

Optimización mediante centro y borde derecho

El algoritmo utiliza dos variables clave para optimizar el proceso:

- **centroActual**: posición del centro del palíndromo que alcanza más a la derecha
- **bordeActual**: límite derecho del palíndromo más extenso encontrado hasta el momento

Gracias a estas variables, cuando se procesa una nueva posición:

1. Si la posición está dentro del borde derecho actual, se puede aprovechar información previamente calculada mediante simetría
2. Se calcula la posición simétrica respecto al centro actual
3. Se utiliza el valor mínimo entre el radio de la posición simétrica y la distancia al borde

Esto evita repetir comparaciones innecesarias y convierte a Manacher en un algoritmo lineal.

Expansión del palíndromo

Una vez determinada la longitud inicial aprovechando la simetría, el algoritmo intenta expandir el palíndromo comparando caracteres hacia ambos lados:

```C/C++
while (izq >= 0 && der < tamProcesado && texto[izq] ==
texto[der]) {
    longitudInicial++;
    izq--;
    der++;
}
```

Este proceso se detiene cuando se encuentra un desajuste o se alcanzan los límites de la cadena.

Búsqueda del máximo

Finalmente, se recorre el vector de longitudes para identificar el palíndromo más largo:

- Se encuentra la posición con el radio máximo
- Se convierte esta posición de la cadena procesada a índices de la cadena original
- Se calcula el inicio como: `(posicionMaxima - longitudMaxima) / 2`

- Se calcula el fin como: `inicio + longitudMaxima - 1`

Complejidad Temporal

En este algoritmo se utiliza la variable '*n*' que corresponde a la longitud del texto de entrada.

La complejidad temporal del algoritmo de Manacher es **O(n)**, donde cada posición de la cadena procesada se visita a lo sumo dos veces:

1. Una vez durante el recorrido principal del bucle
2. Potencialmente una segunda vez durante la expansión

Aunque existen bucles anidados en apariencia, el algoritmo garantiza que cada carácter se compara un número constante de veces gracias a la optimización del centro y borde derecho. Esto hace que la complejidad total sea lineal.

Desglose de complejidades:

- Transformación de la cadena: O(n)
- Procesamiento con Manacher: O(n)
- Búsqueda del máximo: O(n)
- **Total: O(n)**

En las pruebas realizadas, el algoritmo demostró ser extremadamente eficiente incluso con cadenas largas, procesando miles de caracteres en milisegundos.

Conclusión del algoritmo

El algoritmo de Manacher cumplió con las expectativas en términos de complejidad, ya que permite encontrar el palíndromo más largo de manera óptima con complejidad lineal O(n). A diferencia de enfoques más simples que requieren $O(n^2)$ o incluso $O(n^3)$ para verificar todos los posibles palíndromos, Manacher aprovecha las propiedades de simetría para evitar trabajo redundante.

Esta eficiencia lo convierte en la elección ideal para aplicaciones que requieren detección de patrones simétricos en grandes volúmenes de datos, como análisis de transmisiones, búsqueda de secuencias repetidas en ADN, o detección de anomalías en streams de datos donde los patrones palindrómicos puedan indicar comportamientos sospechosos o errores de transmisión.

La implementación desarrollada mantiene un código claro y modular, con nombres de variables descriptivos que facilitan su comprensión y mantenimiento, sin sacrificar el rendimiento óptimo del algoritmo original.

Función para encontrar el substring más largo común entre dos archivos

Descripción

La función substringMasLargoComun encuentra la subcadena más larga que aparece exactamente igual en dos archivos de transmisión. Además, indica la posición inicial y final (empezando en 1) donde se encuentra esa subcadena dentro del primer archivo.

Funcionamiento

1. **Lectura de archivos:**
   El contenido de ambos archivos se lee y se almacena como cadenas de texto.

2. **Algoritmo utilizado:**
   Se emplea el algoritmo de **programación dinámica** conocido como "Longest Common Substring" (LCS).

   - Se construye una matriz donde cada celda [i][j] representa la longitud del substring común más largo que termina en la posición i del primer archivo y en la posición j del segundo archivo.
   - Si los caracteres coinciden, se suma 1 al valor diagonal anterior; si no, se coloca 0.
   - Se lleva un registro de la longitud máxima encontrada y la posición donde termina en el primer archivo.

3. **Cálculo de posiciones:**
   Una vez encontrado el substring más largo común, se calcula su posición inicial y final en el primer archivo (usando índices que empiezan en 1).

4. **Salida:**
   La función retorna un par de enteros con la posición inicial y final del substring más largo común. Si no existe tal substring, retorna {-1, -1}.

Complejidad

- **Tiempo:**
  La complejidad temporal es **O(n × m)**, donde n es la longitud del primer archivo y m la del segundo archivo. Esto se debe a que se recorre una matriz de tamaño n × m.

- **Espacio:**
  La complejidad espacial también es **O(n × m)**, ya que se utiliza una matriz auxiliar para almacenar los resultados parciales de la comparación.

Utilidad

Esta función es útil para:

- Analizar la similitud entre archivos.
- Detectar fragmentos compartidos o posibles plagios.
- Identificar patrones comunes en transmisiones de datos.

Ejemplo de uso

Supongamos que tenemos el siguiente archivo de transmisión:

Archivo 1:

```
None
70616e20646520e6172716e6a61
50616e20646520e6172616e6a61
```

La función indicará la posición inicial y final del substring más largo común encontrado en el primer archivo, comparándolo con el segundo archivo.

Referencias

Comment, info, M., K

kartik, K, Kartik, & Follow. (2025, August 27). *KMP algorithm for pattern searching*. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/kmp-algorithm-for-pattern-searching/

Das, A. (2021, August 31). *Preprocessing algorithm for KMP Search (LPS array algorithm)*. Medium. https://medium.com/@aakashjsr/preprocessing-algorithm-for-kmp-search-lps-array-algorithm-50e35b5bb3cb

GeeksforGeeks. (2025, July 23). *Knuth-Morris-Pratt in C++*. https://www.geeksforgeeks.org/cpp/knuth-morris-pratt-in-c/

Nivardo. (2024, April 11). *archivos en C++ → cómo USAR EN C++* . Oregoom.com. https://oregoom.com/cpp/archivos/