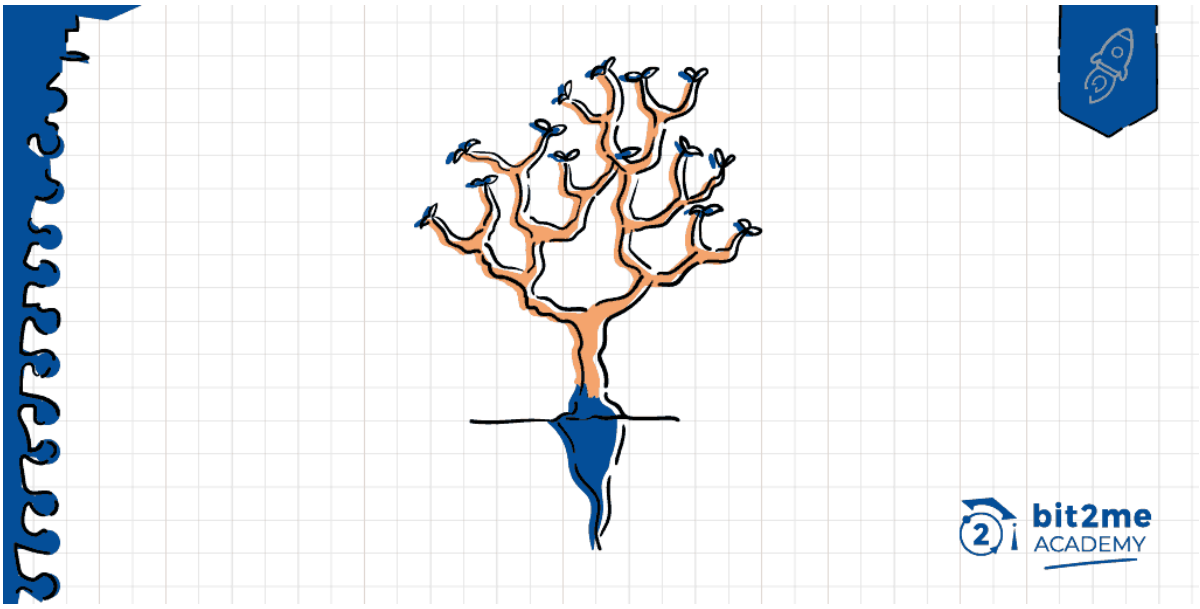
Reporte Final (Semana 3)

ICI2241-1 – Análisis y Diseño de Algoritmos

José Lara Arce

▼ **Descripción del problema: Árbol de Merkle**

El problema que resuelve el algoritmo de construcción del árbol de Merkle es garantizar la integridad de un conjunto de datos que se almacenan y transmiten de manera segura. Dado un conjunto de datos, el objetivo es crear una estructura de árbol de hash, conocido como árbol de Merkle, que permita verificar la integridad de los datos y detectar posibles cambios o corrupciones en ellos.



▼ **Entradas, restricciones y salidas**

1. **Entradas:**
    - Datos Originales: El algoritmo necesita los datos originales que se desean verificar en términos de integridad y autenticidad. Estos datos pueden ser un archivo, un conjunto de transacciones, o cualquier otro conjunto de información que se quiera asegurar que no haya sido alterado.
2. **Restricciones:**

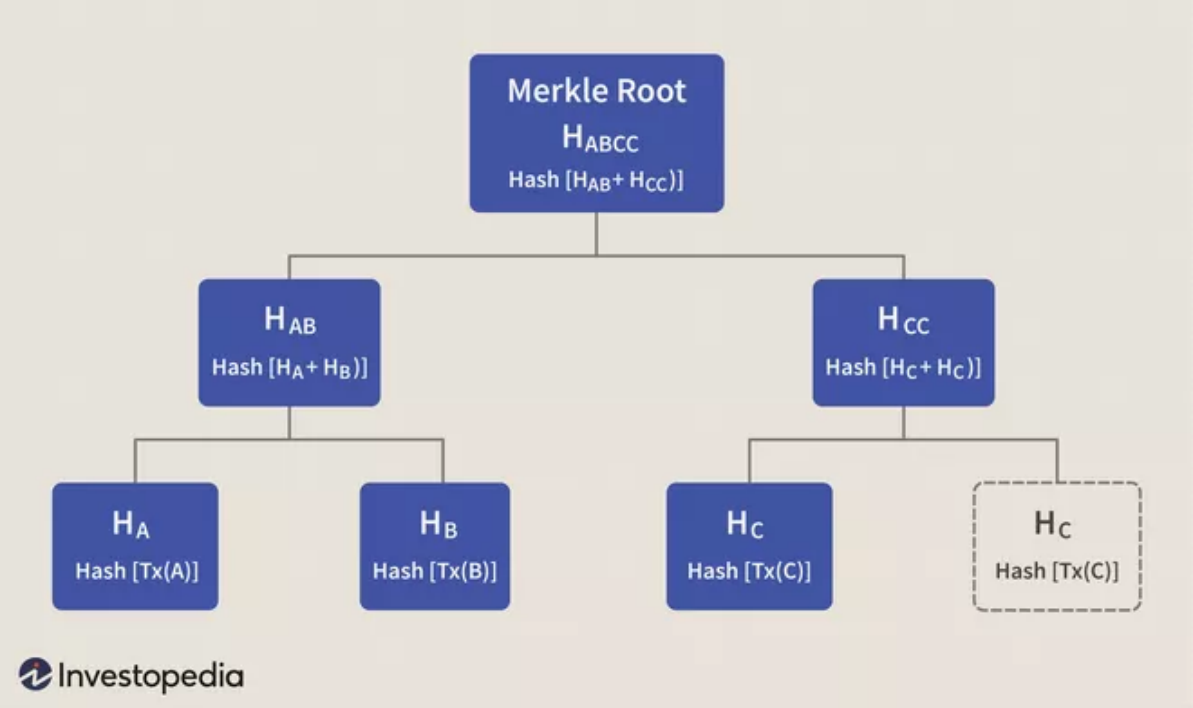
- Precisión de Hash: El algoritmo se basa en funciones de hash criptográficas para generar los valores hash de los bloques de datos. La precisión y la resistencia criptográfica de estas funciones son cruciales para garantizar la seguridad del proceso.
- Integridad de Datos Iniciales: Las entradas deben ser precisas y no estar corrompidas en el momento en que se calculan los hashes iniciales. Cualquier corrupción en los datos originales afectaría la confiabilidad de los resultados.

3. **Salidas:**

- Hash de la Raíz de Merkle: El resultado esperado del algoritmo es un único valor hash conocido como la "raíz de Merkle" (Merkle root). Este valor representa toda la estructura de datos y verifica la integridad y autenticidad de los datos originales. La raíz de Merkle se utiliza para verificar si los datos han sido alterados en tránsito o si la información es auténtica.

▼ **Descripción del algoritmo**

El árbol de Merkle es una estructura de datos jerárquica y eficiente que se utiliza para garantizar la integridad y autenticidad de conjuntos de datos. Este algoritmo se basa en el principio de usar funciones de hash criptográficas para construir una estructura en forma de árbol, donde cada nodo hoja contiene un hash de un bloque de datos y cada nodo interno contiene el hash de los nodos hijos. La raíz de este árbol, llamada la "raíz de Merkle," se utiliza para verificar si los datos originales han sido alterados durante la transmisión o si son auténticos.



▼ **Explicación del paradigma**

En el algoritmo del árbol de Merkle, el paradigma "divide y vencerás" se aplica de la siguiente manera:

- **División:** El conjunto de datos originales se divide en bloques más pequeños. Cada bloque se convierte en un nodo hoja en el árbol de Merkle.
- **Cálculo de Hashes:** Se resuelve un subproblema al calcular un hash único para cada bloque individual.
- **Combinación:** Los valores hash de los bloques se combinan para formar los valores hash de los nodos intermedios. Esto se realiza en niveles sucesivos del árbol hasta llegar a la raíz.

▼ **Explicación paso a paso**

1. **División y Cálculo de Hashes:**
   En este primer paso, se toma el conjunto de datos originales que se desea proteger y se divide en bloques más pequeños. Cada bloque de datos se pasa a través de una función de hash criptográfica, como SHA-256 (Secure Hash Algorithm 256 bits). Esta función toma los datos de entrada y produce un valor hash único, que es una representación alfanumérica fija de longitud fija. Los valores hash resultantes de cada bloque se convierten en los nodos hoja del árbol.

2. **Construcción del Árbol:**
   Los nodos hoja (los valores hash de los bloques) se organizan en niveles para formar una estructura de árbol binario. Esto significa que cada nodo hoja se convierte en un nodo en el nivel más bajo del árbol. Los nodos intermedios y la raíz se construirán en niveles superiores. Los nodos intermedios se generan calculando el valor hash combinado de los nodos hoja que son sus hijos directos. Esto se hace concatenando o combinando de alguna manera los valores hash de los nodos hijos y luego pasándolos a través de la función de hash.

3. **Raíz de Merkle:**
   La raíz del árbol, también conocida como raíz de Merkle, es el nodo en el nivel superior del árbol. Se calcula combinando los valores hash de los nodos intermedios de manera similar a como se hizo en el paso anterior. La raíz de Merkle representa, de manera compacta, la integridad de todos los datos originales. Cualquier cambio en los datos, incluso en un solo bit de un bloque, se propagaría hacia arriba en la estructura del árbol y cambiaría los valores hash en los nodos correspondientes. Como resultado, la raíz de Merkle también cambiaría, lo que indicaría que los datos se han modificado de alguna manera.

▼ **Ejemplo: Verificación de Integridad de Archivo**

Entradas:

- Archivo Original: Un archivo de texto llamado "documento.txt" que contiene el texto "Hola, este es un documento de prueba."

Pasos del Algoritmo:

1. **División en Bloques y Cálculo de Hashes:**
   - El archivo original se divide en bloques más pequeños. Por ejemplo, podríamos dividirlo en bloques de 8 caracteres.
   - Para cada bloque, se calcula un valor hash utilizando una función de hash criptográfica como SHA-256.

2. **Construcción del Árbol:**
   - Los valores hash calculados se convierten en los nodos hoja del árbol.
   - Los nodos intermedios se calculan combinando los valores hash de sus nodos hijos en niveles sucesivos del árbol. El árbol se construye hacia arriba, fusionando pares de nodos hash en nodos intermedios.

3. **Raíz de Merkle:**

- La raíz del árbol, conocida como la raíz de Merkle, se convierte en el valor hash final que representa la integridad del archivo original.

Salida:

- La raíz de Merkle generada: Por ejemplo, el valor hash generado es "e19f571e55d1f50c6c39b5b20140c6091c6c8909133ab2150167cd5d13ff4990".

Verificación de Integridad:
Supongamos que alguien intenta modificar el contenido del archivo original introduciendo un espacio adicional después de la palabra "Hola". Esto cambiaría el contenido y, por lo tanto, debería cambiar el valor hash del archivo.

- Nuevo Archivo Modificado: "Hola, este es un documento de prueba."
- Nuevo Valor Hash Calculado: "b6b03c1612c0213ff8fcabef5a69288a21a75d8c72c67880ea227bf567ab22d0"

Comparación con la Raíz de Merkle Generada:

- Al comparar el nuevo valor hash calculado con la raíz de Merkle generada previamente, se puede determinar que el archivo ha sido alterado. Los valores no coincidirán, lo que indica una modificación en los datos originales.

▼ **Árbol de Merkle: Análisis de Correctitud por Inducción**

1. **Identificación del Caso Base:**
   El caso base en el análisis de la correctitud del algoritmo del árbol de Merkle se refiere a la situación en la que solo tenemos un bloque de datos, lo que equivale a tener un solo nodo hoja en el árbol. En este caso, la raíz de Merkle sería simplemente el hash del único bloque.
2. **Hipótesis de Inducción:**
   Supongamos que el algoritmo del árbol de Merkle funciona correctamente para un conjunto de datos con *n* bloques, es decir, para un árbol de Merkle con n nodos hoja.
3. **Paso de Inducción:**
   Ahora, debemos demostrar que si la hipótesis de inducción es cierta para *n*, entonces también es cierta para *n + 1*. En otras palabras, si el algoritmo funciona correctamente para *k* bloques, también funcionará correctamente para n + *1* bloques.

   Demostración del Paso de Inducción:
   Supongamos que hemos demostrado que el algoritmo del árbol de Merkle es correcto para un árbol con n nodos hoja. Ahora, agreguemos un nuevo bloque de datos para formar un árbol con *n + 1* nodos hoja. El algoritmo seguirá los mismos pasos: dividirá en bloques, calculará los valores hash, construirá el árbol y generará la raíz de Merkle.

Dado que el algoritmo se basa en funciones de hash criptográficas, un cambio en un bloque de datos alteraría su valor hash y, por lo tanto, afectaría todos los nodos intermedios y la raíz de Merkle. Dado que la hipótesis de inducción ya nos dice que el algoritmo es correcto para *k* bloques, si todo el proceso se ejecuta correctamente para *n+1* bloques, se garantizará que la raíz de Merkle refleje cualquier cambio en los datos.

4. **Prueba de las Funciones Auxiliares:**
   En el árbol de Merkle, las funciones auxiliares son las funciones de hash criptográficas utilizadas para calcular los valores hash de los bloques de datos y los nodos intermedios. Para demostrar la correctitud, se requiere que estas funciones sean resistentes a la colisión y produzcan resultados únicos para datos diferentes.

5. **Conclusión:**
   Si hemos demostrado tanto el caso base como el paso de inducción, podemos concluir, por inducción, que el algoritmo del árbol de Merkle es correcto para todos los tamaños de conjunto de datos. La estructura jerárquica del árbol y el uso de funciones de hash criptográficas garantizan que cualquier cambio en los datos originales sea detectado en la raíz de Merkle.

▼ **Desempeño del algoritmo del Árbol de Merkle en la Verificación de Integridad de Datos**

Se llevará a cabo una evaluación del desempeño del Algoritmo del Árbol de Merkle en la verificación de la integridad de datos. Para lograrlo, tendremos un conjunto diverso de casos de prueba, variando el tamaño y contenido de las entradas. Nuestro objetivo es comprender cómo el algoritmo se comporta en situaciones que abarcan desde datos pequeños hasta conjuntos de datos más grandes y complejos. Al probar con diferentes entradas, podremos analizar la eficiencia y la rapidez del algoritmo en la construcción del árbol de Merkle y la generación de las raíces Merkle correspondientes. Esta evaluación nos proporcionará una visión integral de la capacidad del algoritmo para garantizar la integridad de los datos en diversos escenarios.

▼ Pruebas

▼ Experimento 1: **Prueba con datos pequeños**

```
Datos = [”Hello, “World”] arreglo de cadenas de carácteres de 2 elementos.
```

El algoritmo del árbol de Merkle opera eficientemente con entradas pequeñas, como en este caso donde tenemos dos elementos: `'Hello'` y `'World'`. Debido a la cantidad limitada de datos, el proceso de construcción del árbol de Merkle es simple y rápido. Cada cadena se convierte en un hash único utilizando una función de hash criptográfico.

Finalmente, este sería la raíz de Merkle para este experimento:

e3caa45a951457b84493e3adec8265f99311c3b1b4f28befbb067a1912efab9e

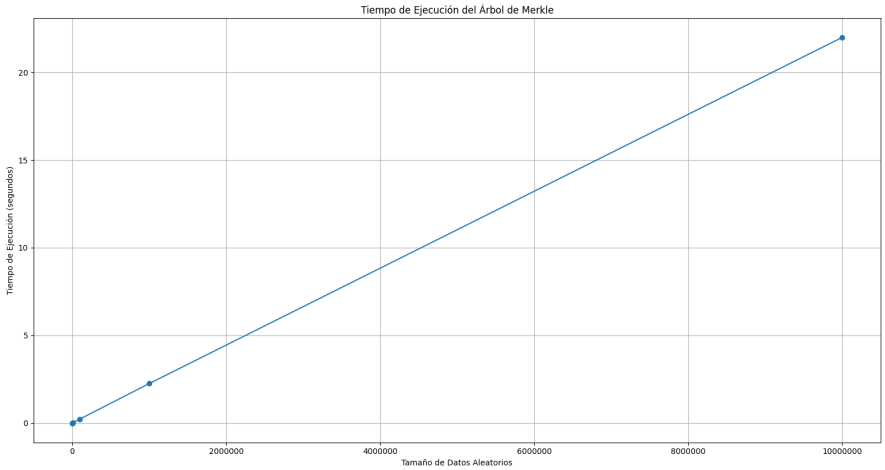
▼ Experimento 2: **Prueba con datos aleatorios y aumento lineal de entrada**

El comportamiento del algoritmo del árbol de Merkle con datos aleatorios revela su capacidad para manejar diferentes tipos de contenido de manera eficiente. Al emplear conjuntos de datos aleatorios de distintos tamaños, podemos observar cómo el algoritmo construye los niveles del árbol y genera las raíces Merkle correspondientes.

A medida que aumenta el tamaño de los datos, el algoritmo continúa operando con eficiencia, ya que el tiempo de construcción del árbol de Merkle depende principalmente del número de elementos en la entrada y no de su contenido en sí. Dado que los hashes se calculan de manera independiente para cada elemento, la variabilidad en los datos aleatorios no afecta significativamente la velocidad del algoritmo.

Al aumentar la entrada hasta 10 millones de cadenas aleatorias, el tiempo de ejecución se vuelve considerablemente más largo en comparación con conjuntos de datos más pequeños. Este comportamiento refleja cómo el algoritmo responde al aumento en la cantidad de datos a procesar y cómo la complejidad de las operaciones necesarias influye en el tiempo requerido para completar el proceso de construcción del árbol de Merkle.

Se puede observar que la complejidad de este algoritmo es O(n).



▼ Experimento 3: **Cambio mínimo**

Supongamos que tenemos una lista de cadenas de texto: `['manzana', 'banana', 'ignacio', 'hola', 'jajaja']`. Calculamos el árbol de Merkle para esta lista y obtenemos una raíz Merkle específica.

> 🌳 Raiz de Merkle: 438bfeaf446901ac59babcd2c1740886e81a668ba4e49e009e34dc64c4f51f96

Cambio Mínimo; Ahora, hagamos un cambio mínimo en la segunda cadena ‘banana’. Cambiaremos la cadena para que ahora sea ‘banina’. Esta alteración es sutil pero suficiente para generar un hash completamente diferente y, por lo tanto, una raíz Merkle diferente:

> 🌳 Raiz de merkle : 298661bd9dd6728217f48b03cc6e4af94f1c29769307577f4a1691985c34ab16

Al introducir alteraciones mínimas en uno o más elementos de la entrada original, observamos que el algoritmo genera una respuesta rápida y precisa al detectar estos cambios. Incluso un pequeño ajuste en un solo carácter de una cadena resultará en un hash completamente diferente, lo que a su vez conduce a una raíz Merkle distinta.

Este comportamiento demuestra la sensibilidad del árbol de Merkle a cambios, sin importar cuán pequeños sean. La prueba resalta la fortaleza del algoritmo como

herramienta para verificar la integridad de los datos. Al realizar cambios mínimos en los datos originales, el árbol de Merkle rápidamente identifica la alteración, y la comparación de las raíces Merkle resultantes revela la diferencia en la integridad de los datos. En esencia, la Prueba de Cambio Mínimo confirma que el algoritmo es capaz de detectar incluso las modificaciones más pequeñas en los datos y resaltar su papel fundamental en la seguridad y verificación de la integridad de los mismos.

▼ Experimento 4: **Datos idénticos**

Supongamos que tenemos una lista de cadenas de texto que consiste en la repetición de la misma cadena: `['apple', 'apple', 'apple', 'apple', 'apple']`.

El resultado será que todos los hashes individuales para cada cadena serán iguales, ya que son idénticas. Al combinar estos hashes, el algoritmo generará niveles superiores del árbol de Merkle hasta llegar a la raíz, que será un hash único que representa la integridad de los datos originales. Aunque todas las cadenas son iguales, el algoritmo aún es capaz de construir un árbol y calcular una raíz Merkle que cumple su propósito criptográfico de verificar la integridad de los datos. Esto demuestra la consistencia y robustez del algoritmo incluso cuando se enfrenta a entradas que contienen valores idénticos. A pesar de que todos los elementos son iguales, el algoritmo del árbol de Merkle sigue siendo eficaz en la generación de la raíz Merkle.

▼ **Mejoras**

- Se agregaron imágenes ilustrativas
- Se modificó el posicionamiento de la categoría de entradas, restricciones y salidas
- Se mejoró la explicación del paso a paso.