Algoritmos de compresión de datos

Exposición I

Velazquez Balandra Diego

diego.velazquezb@ingenieria.unam.edu

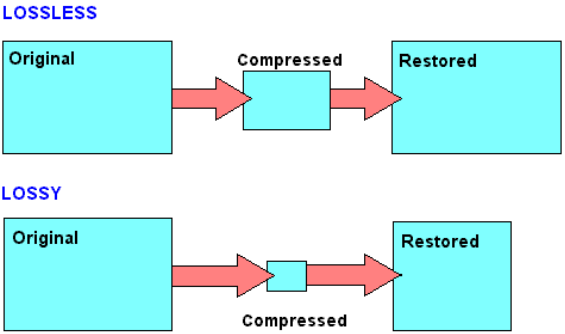
Sistemas Operativos Grupo 06, Septiembre 2024

Agenda

Introducción

La compresión de datos ha representado un papel crucial en los sistemas operativos y sigue siendo importantísimo en estos, aparte de diversas aplicaciones. Hoy en día la generación de datos es exponencial, la necesidad de almacenar, transmitir y procesar información de forma eficiente ha cobrado mayor importancia que nunca. La compresión de datos, ahora, también resulta fundamental para la computación en la nube, la transmisión de multimedia y las comunicaciones en línea, donde los grandes volúmenes de datos son esenciales para conocer la información que pueda ayudar para reducir costos, mejorar la experiencia del usuario y mucha más información.

Tipos de Compresión de Datos



Representación de la comparación de tipos de compresión de datos y sus dimensiones de forma visual. Tomado de *Lossless compression*, por PCmag, https://www.pcmag.com/encyclopedia/term/lossless-compression

Codificación Huffman

La codificación Huffman es una técnica de compresión sin pérdida que se usa ampliamente para comprimir archivos de texto. Desarrollado por David Huffman en 1952, el algoritmo crea códigos de longitud variable para cada carácter en función de su frecuencia de aparición. A los caracteres más utilizados se les asignan códigos más cortos, mientras que a los caracteres menos frecuentes se les asignan códigos más largos. Este enfoque reduce significativamente el tamaño general del archivo, especialmente cuando hay mucha repetición.

Cómo funciona la codificación Huffman

El algoritmo comienza contando la frecuencia de cada carácter en el conjunto de datos. Tomando por ejemplo, un archivo de texto, se construye un árbol binario, llamado árbol Huffman, donde cada carácter se convierte en un nodo de hoja. Los caracteres con las frecuencias más bajas se combinan primero, formando gradualmente un árbol.
Se asigna un código binario a cada carácter en función de su posición en el árbol. Los caracteres que aparecen con mayor frecuencia reciben códigos binarios más cortos, mientras que los caracteres poco frecuentes reciben códigos más largos (Scharf, 2022).

Cómo funciona la codificación Huffman

Algorithm 1 Algoritmo de huffman

```
1:  procedure HUFFMAN(c)
2:      n ← |c|
3:      Q ← c
4:      for i ← 1 to n − 1 do
5:          temp ← get node()
6:          left[temp] ← Get_min(Q)
7:          right[temp] ← Get_min(Q)
8:          a ← left[temp]
9:          b ← right[temp]
10:         F[temp] ← f[a] + f[b]
11:         insert(Q, temp)
        return Get_min(Q)
```

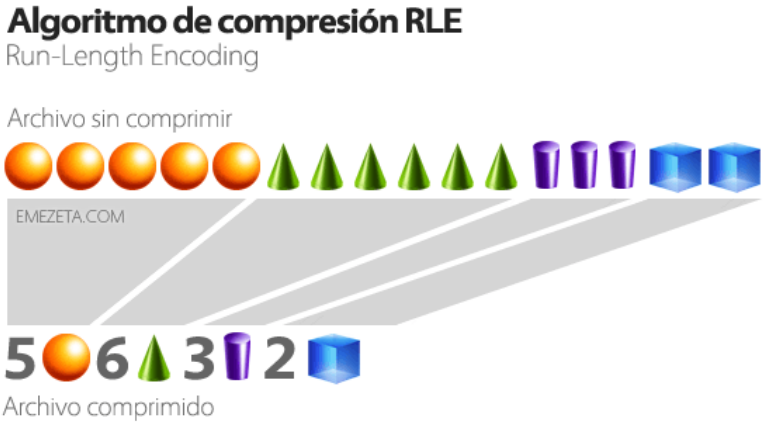
Algoritmo de Huffman

Por ejemplo, si el texto ***"BEEEP"*** se comprime utilizando la codificación Huffman, el algoritmo asignaría códigos binarios más cortos a la "E" que aparece con frecuencia y códigos más largos a las "B" y "P" menos frecuentes. El resultado comprimido final constará de menos bits que el texto original, lo que ahorrará espacio de almacenamiento.

Codificación de longitud de ejecución (RLE)

La codificación de longitud de ejecución (RLE) *Run Length Encoding* es un algoritmo de compresión sin pérdida simple pero efectivo que se utiliza para comprimir datos que contienen muchos elementos repetidos consecutivos, o "ejecuciones". La idea detrás de RLE es almacenar una única instancia del elemento repetido, seguida del número de veces que se repite. Este método es particularmente útil para comprimir imágenes, texto y otras formas de datos donde los patrones repetidos son comunes.

Codificación de longitud de ejecución (RLE)



Codificación RLE. Tomado de *Compresión con pérdida y sin pérdida*, por comprimeme.wordpress, https://comprimeme.wordpress.com/compresion-con-perdida-y-sin-perdida/

Cómo funciona RLE

RLE escanea los datos y agrupa elementos idénticos consecutivos. En lugar de almacenar cada elemento individualmente, registra el elemento una vez, seguido de un recuento de cuántas veces aparece consecutivamente (geeksforgeeks.org, 2023a).

Cómo funciona RLE

```
// Programa Javascript para implementar codificaci n RLE
    function printRLE(str)
    {
        let n = str.length;
        for (let i = 0; i < n; i++)
        {
            // Contar las ocurrencias del car cter actual
            let count = 1;
            while (i < n - 1 && str[i] == str[i+1])
            {
                count++;
                i++;
            }

            // Imprimir car cter y su conteo
            document.write(str[i]);
            document.write(count);
        }
    }
```

Codificación RLE. Tomado de *Compresión con pérdida y sin pérdida*, por comprimeme.wordpress, https://comprimeme.wordpress.com/compresion-con-perdida-y-sin-perdida/

Por ejemplo, considerando la siguiente secuencia de caracteres: ***AAAAAABBBBCCDAA***. RLE la comprimiría como: ***A6B4C2D1A2***

Lempel-Ziv (LZ77, LZW)

Los algoritmos Lempel-Ziv (LZ77 y LZW) se encuentran entre las técnicas de compresión sin pérdida más utilizadas y forman la base de muchos formatos de compresión, como ZIP y GIF. Estos algoritmos exploran patrones en los datos al reemplazar secuencias repetidas con referencias más cortas a ocurrencias anteriores de la misma secuencia (Salomon y Motta, 2010).

Pseudocódigo en general de Lempel-Ziv

Algorithm 2 Pseudocódigo de Lempel-Ziv

```
Initialize table with single character strings
P ← first input character
while not end of input stream do
    C ← next input character
    if P + C is in the string table then
        P ← P + C
    else
        output the code for P
        add P + C to the string table
        P ← C
output code for P
```

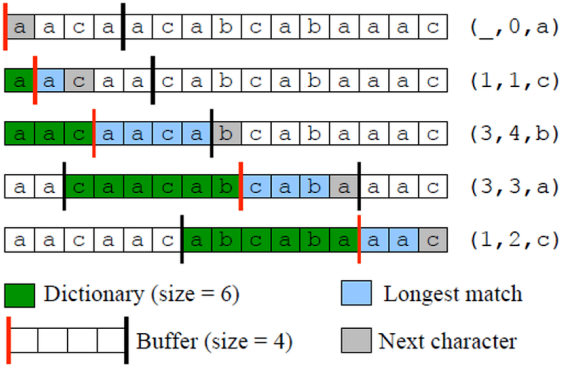
(geeksforgeeks.org, 2024)

Cómo funciona LZ77

Desarrollado por Abraham Lempel y Jacob Ziv en 1977, LZ77 funciona escaneando los datos e identificando secuencias repetidas. En lugar de almacenar la secuencia nuevamente, almacena un puntero a la ocurrencia anterior y la longitud de la secuencia (Sayood, 2012).

Por ejemplo, considerando la cadena: ***ABABABABA***. LZ77 codificaría esto como: ***(0,0,A) (0,0,B) (2,2,A) (4,3,B).*** Aquí, ***(0,0,A)*** representa la primera aparición de "A", mientras que (2,2,A) se refiere a una "A" repetida después de dos caracteres, lo que reduce la cantidad de almacenamiento necesaria para patrones repetidos. Ejemplo en la Figura 16.

Cómo funciona LZ77



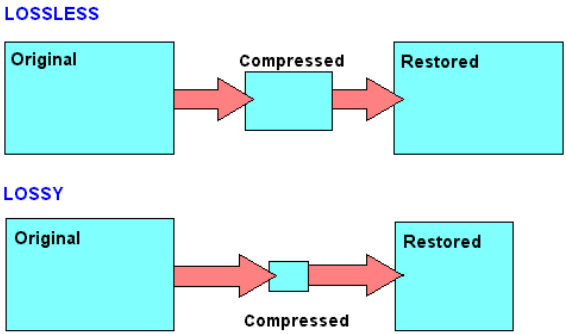
Algoritmos de Compresión con Pérdida

Los algoritmos de compresión con pérdida están diseñados para reducir el tamaño de los archivos descartando algunos de los datos originales. Estas técnicas se utilizan ampliamente en el área de multimedia, donde pérdidas leves de calidad son mayormente imperceptibles para el ojo o el oído humano. En esta sección, se exploran dos métodos de compresión con pérdida:

- La transformada discreta del coseno (DCT), que se utiliza habitualmente en la compresión de imágenes.
- Los algoritmos de codificación de vídeo, como:
  - H.264
  - HEVC

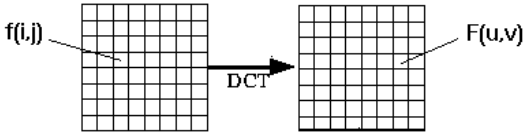
(Sayood, 2012)

Algoritmos de Compresión con Pérdida



Ejemplo de comparación entre una imagen sin pérdida (*lossless* y con pérdida (*lossy*)). Tomado de *Lossy Compression*, por Taylor C., 2018, https://cyberhoot.com/cybrary/lossy-compression/

Cómo funciona la DCT



Transformación de una señal o imagen del dominio espacial al dominio de frecuencia. Tomado de *The Discrete Cosine Transform (DCT)*, por Marshall D., 2001, https://https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html#DCTenc

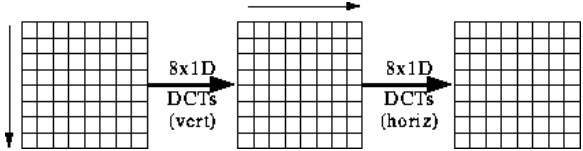
Cómo funciona la DCT

El funcionamiento básico de la DCT es el siguiente:

- La imagen de entrada es N por M;
- f(i,j) es la intensidad del píxel en la fila i y la columna j;
- F(u,v) es el coeficiente de la DCT en la fila k1 y la columna k2 de la matriz DCT.
- En la mayoría de las imágenes, gran parte de la energía de la señal se encuentra en frecuencias bajas; estas aparecen en la esquina superior izquierda de la DCT.
- La compresión se logra porque los valores inferiores a la derecha representan frecuencias más altas y, a menudo, son pequeños, lo suficientemente pequeños como para ignorarlos con poca distorsión visible.
- La entrada de la DCT es una matriz de 8 por 8 números enteros. Esta matriz contiene el nivel de escala de grises de cada píxel;
- Los píxeles de 8 bits tienen niveles de 0 a 255.

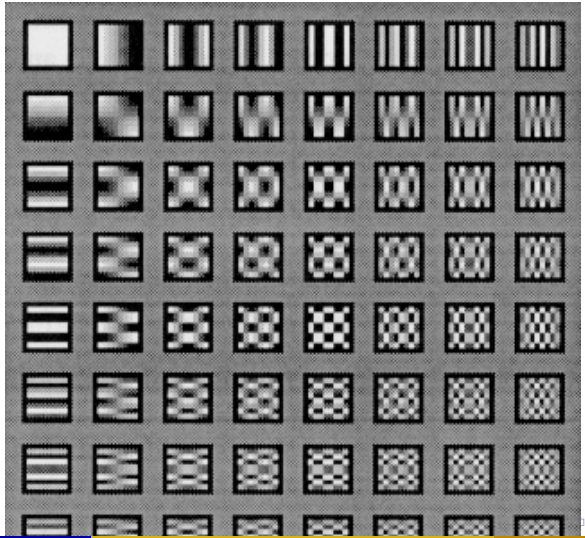
(Marshall, 2001)

Cómo funciona la DCT

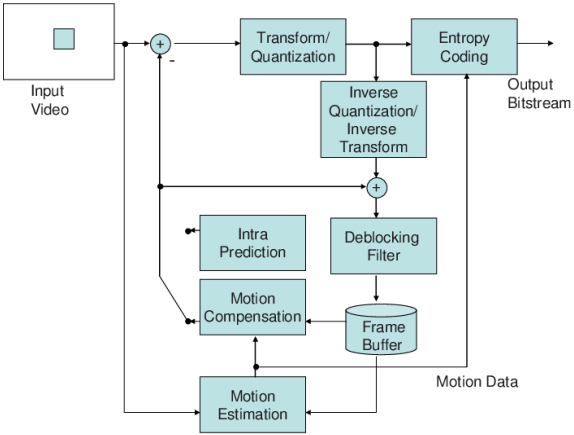


Cálculo de la DCT 2D. Tomado de *The Discrete Cosine Transform (DCT)*, por Marshall D., 2001, https://https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.htmlDCTenc

Cómo funciona la DCT



Cómo funciona H.264



Algoritmo de codificación H.264/AVC. Tomado de *Video Coding on Multicore Graphics Processors*, por Cheung et al., 2010,
https://www.researchgate.net/figure/H264-AVC-encoding-algorithm-5_fig5_224127502

Cómo funciona HEVC

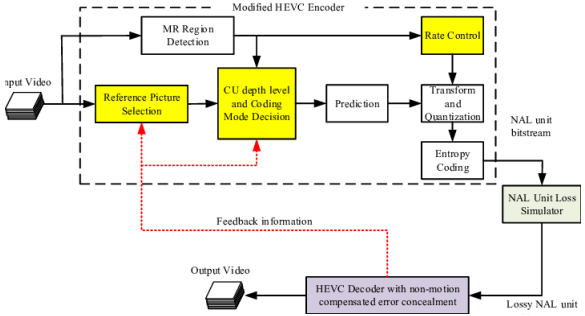


Diagrama de bloques general del sistema de transmisión de vídeo HEVC. Tomado de *Region-of-interest based error resilient method for HEVC video transmission*, por Maung et al., 2015, https://www.researchgate.net/figure/Overall-block-diagram-of-HEVC-video-transmission-system$_{f}ig2_{3}04268810$

Aplicaciones y Uso en Sistemas Operativos

La compresión de datos desempeña un papel importante en la optimización del rendimiento de los sistemas operativos al reducir los requisitos de almacenamiento, acelerar la transmisión de datos y administrar de manera eficiente la memoria. Las técnicas de compresión permiten que los sistemas gestionen grandes cantidades de datos de manera más fácil, lo que garantiza una mejor utilización de los recursos y más rápidos tiempos de procesamiento (Silberschatz et al., 2018).

Referencias

- Scharf, L. (2022). A First Course in Electrical and Computer Engineering [[Accesado 06-09-2024]]. https://cnx.org/contents/fpkWedRh@2.3:tQa_RWkY@3/Dedication-of-A-First-Course-in-Electrical-and-Computer-Engineering
- geeksforgeeks.org. (2023a). Run Length Encoding and Decoding [[Accesado 06-09-2024]]. https://www.geeksforgeeks.org/run-length-encoding/
- Salomon, D., & Motta, G. (2010). *Data Compression: The Complete Reference* (4th). Springer.
- geeksforgeeks.org. (2024). Run Length Encoding and Decoding [[Accesado 06-09-2024]]. https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/
- Sayood, K. (2012). *Introduction to Data Compression* (4th). Morgan Kaufmann.
- Marshall, D. (2001). The Discrete Cosine Transform (DCT) [[Accesado 06-09-2024]]. https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html#DCTenc
- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th). John Wiley & Sons.
- Adedeji, K. B. (2020). Performance Evaluation of Data Compression Algorithms for IoT-Based

Referencias

Referencias