UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ingeniería

Algoritmos de compresión de datos

Sistemas Operativos

Alumno: Velazquez Balandra Diego
421089824

Profesor: Ing. Gunnar Eyal Wolf Iszaevich

Grupo: 06

12 de septiembre de 2024

Índice

Alumno:

Velazquez Balandra Diego

Grupo 6

Algoritmos de compresión de datos

1. 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.

Existen dos principales formas de compresión de datos: sin pérdida y con pérdida. La compresión sin pérdida reduce el tamaño de los archivos sin ninguna pérdida en calidad, lo que la hace ideal para textos, software y datos delicados. Por otro lado, la compresión con pérdida logra mayor reducción al sacrificar mínima precisión en los datos, por lo que resulta mejor para medios como imágenes, audio y video, donde estas pequeñas pérdidas en calidad son totalmente aceptables. Asimismo, este tipo de compresión con pérdida, permite alcanzar un elevado porcentaje de compresión, aunque posteriormente los archivos comprimidos no podrán descomprimirse exactamente al estado original.

El cometido de este escrito para la exposición misma del tema, es explorar los principales algoritmos de compresión de datos, de los tipos con y sin pérdida, para comprender cómo funcionan y sus aplicaciones. Al examinar estos algoritmos, se informará sobre la eficacia de la compresión, la velocidad y la integridad de la información.

2. Tipos de Compresión de Datos

La compresión de datos se divide en dos categorías: compresión sin pérdidas, es decir, que no pierde información, y compresión con pérdidas, que sacrifica detalles mínimos para mejorar la eficiencia de la compresión misma. Estos diferentes métodos tienen sus ventajas y desventajas en cuanto a rapidez, precisión de resultados y protección de datos (Keepcoding, 2024).

2.1. Compresión sin pérdida

Esta técnica es muy relevante cuando se trata de archivos de texto, software o datos importantes, porque la pérdida de información puede ser fatal. Con este tipo de compresión, los datos se pueden restaurar

Figura 1: Representación de la compresión de datos. Tomado de *COMPRESIÓN DE ARCHIVOS*, por informaticamoderna.com, 2020, https://www.informaticamoderna.com/Compresion.html

a su estado original exactamente como estaban antes de la compresión.

Se puede encontrar compresión sin pérdidas en archivos zip para documentos, archivos png para imágenes y archivos flac para música. Los archivos zip son carpetas que comprimen todos los archivos en una especie de paquete eliminando cualquier material adicional que sea repetido. Las imágenes png son archivos capaces almacenar gráficos sin perder ningunas de sus propiedades visibles de la imagen. Son ideales para mostrar imágenes detalladas, como logotipos o íconos. Flac es una herramienta que preserva claridad de los archivos de audio, siendo la indicada para apreciar sonido de alta calidad sin pérdida de la misma(Sayood, 2012).

2.2. Compresión con pérdida

La compresión con pérdida reduce el tamaño de sus archivos eliminando parte de la información, siendo no tan efectiva como la compresión sin pérdida. El objetivo de la compresión con pérdida es deshacerse de información extra o menos importante, especialmente datos que podrían pasar desapercibidos para los sentidos humanos. Es por eso que se usa para todo tipo de archivos, como imágenes, música y videos, porque incluso si hay una pequeña pérdida de calidad, el ahorro de espacio es mucho más beneficioso.

La compresión con pérdida se produce cuando se eliminan algunos detalles de un archivo para reducir su espacio, como cuando se convierte una imagen de alta calidad a menor espacio sin perder demasiada calidad. La compresión Mp3 es elimina los sonidos extra que no son perceptibles al oído humano, haciendo el archivo más pequeño sin perder calidad. Jpeg es elimina los pequeños detalles de una imagen que ni siquiera podemos ver cuando la observamos de cerca, lo que la hace idónea para compartir en Internet(Sayood, 2012).

2.3. Comparación entre compresión con y sin pérdida

Tanto la compresión **con**, como **sin** pérdida tienen sus propias ventajas y desventajas. La compresión sin pérdida es ideal para situaciones en las que la integridad de los datos es crítica, ya que garantiza integridad total de la información. Los métodos sin pérdida suelen ofrecer reducciones menos importantes en el tamaño de archivo en comparación con las técnicas con pérdida. La compresión con pérdida puede lograr índices de compresión mucho más altos, lo que la hace una mejor opción para archivos multimedia grandes, donde las limitaciones de almacenamiento son una preocupación. La contraparte es que se sacrifica algo de calidad, situación que puede no ser aceptable en aplicaciones profesionales que exigen alta precisión o fidelidad.

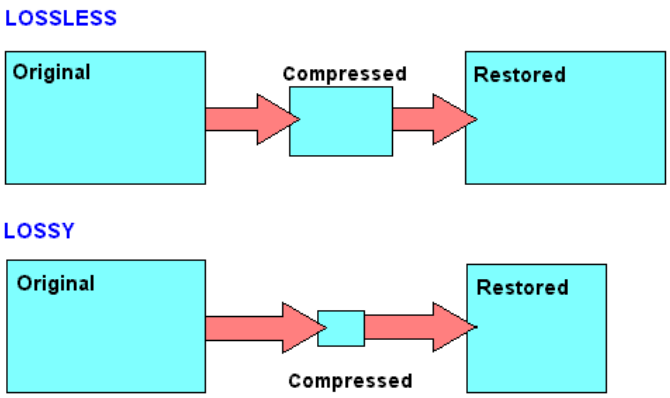

Figura 2: Representación de la comparación de tipos de compresión de datos y sus dimensiones de forma visual. Tomado de *Lossless compression*, por PCmag, https://www.pcmag.com/encyclopedia/term/lossless-compression

La elección entre compresión con y sin pérdida depende del caso de uso específico y de la importancia de preservar la calidad original frente al ahorro de espacio de almacenamiento.

3. Algoritmos de Compresión Sin Pérdida

Los algoritmos de compresión sin pérdida son esenciales en casos en que la integridad de los datos es crucial, ya que permiten reconstruir completamente los datos originales a partir de la versión comprimida. Esta sección explora tres algoritmos de compresión sin pérdida de uso común:

- Codificación Huffman
- Codificación Run-Length (RLE)
- Lempel-Ziv (LZ77, LZW)

Cada algoritmo tiene su enfoque único para reducir el tamaño de los archivos sin perder información, lo que los convierte en parte integral de aplicaciones como la compresión de texto, el almacenamiento de archivos y los formatos de imagen.

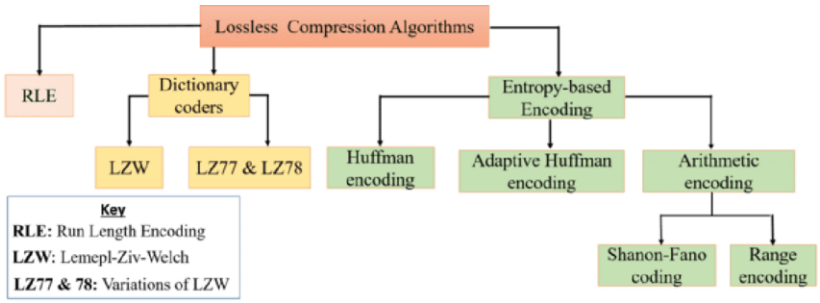

Figura 3: Mapa de los algoritmos de compresión sin pérdida. Tomado de *Performance Evaluation of Data Compression Algorithms for IoT-Based Smart Water Network Management Applications*, por Adedeji, 2020, https://www.researchgate.net/figure/Types-of-lossless-data-compression-algorithm$_fig3_346527862$

3.1. 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 (Sayood, 2012).

3.1.1. 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).

Pseudocódigo Algoritmo 1. (geeksforgeeks.org, 2023a).

Por ejemplo, si el texto ***"BEEEP"*** se comprime utilizando la codificación Huffman, el algoritmo

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)
```

asignaría códigos binarios más cortos a la .E que aparece con frecuencia y códigos más largos a las "Bƶ "P"menos frecuentes. El resultado comprimido final constará de menos bits que el texto original, lo que ahorrará espacio de almacenamiento.

3.1.2. Aplicaciones

La codificación Huffman se utiliza ampliamente en formatos como DEFLATE (que forma parte de los formatos de archivo ZIP y GZIP) y en la compresión multimedia como JPEG y MP3.

3.1.3. Ventajas y desventajas

- Ventajas: la codificación de Huffman es óptima cuando los caracteres del conjunto de datos tienen frecuencias variables. Asegura que no haya pérdida de información y puede lograr una buena compresión para archivos con mucho texto.
- Desventajas: el algoritmo requiere la construcción de un árbol, lo que puede resultar costoso en términos computacionales para conjuntos de datos muy grandes. Además, puede que no funcione bien en conjuntos de datos donde los caracteres tienen frecuencias aproximadamente iguales.

3.2. 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(Salomon y Motta, 2010).

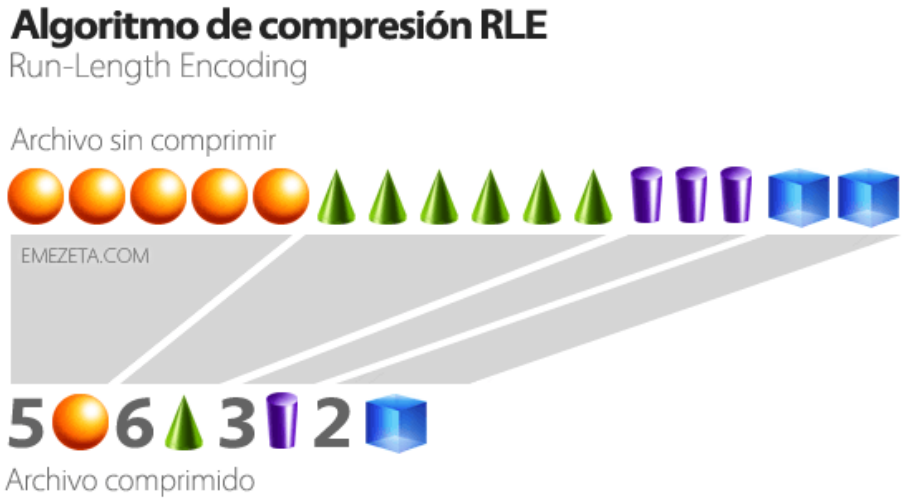

Figura 4: Codificación RLE. Tomado de *Compresión con pérdida y sin pérdida*, por comprimeme.wordpress, https://comprimeme.wordpress.com/compresion-con-perdida-y-sin-perdida/

3.2.1. 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.

```
<script>
// 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);
        }
    }

    let str = "AAAAAABBBBCCDAA";
```

```
    printRLE(str);

    // C digo contribuido por rag2127 en Run Length Encoding and Decoding
        // https://www.geeksforgeeks.org/run-length-encoding/
</script>
```

Listing 1: Ejemplo de codificación de longitud de ejecución en JavaScript

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

La secuencia original tiene 14 caracteres, mientras que la versión comprimida solo tiene 10 (6 letras y 4 recuentos), lo que representa una reducción significativa en el tamaño.

3.2.2. Aplicaciones

RLE se usa comúnmente en formatos de imagen como BMP y TIFF, donde es particularmente eficaz para comprimir imágenes simples con grandes áreas de color uniforme. También se usa en máquinas de fax, donde las imágenes a menudo contienen grandes bloques de píxeles blancos o negros.

3.2.3. Ventajas y desventajas

- Ventajas: RLE es increíblemente fácil de implementar y funciona bien en datos con largas series de valores idénticos. Requiere recursos computacionales mínimos, lo que lo convierte en un algoritmo rápido y eficiente.
- Desventajas: RLE tiene un rendimiento deficiente en datos que carecen de repetición. Si los datos son muy variados, el archivo comprimido puede terminar siendo más grande que el original, ya que el algoritmo aún necesita almacenar un recuento para cada elemento.

3.3. 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).

3.3.1. Pseudocódigo en general de Lempel-Ziv

En Algoritmo 2. (geeksforgeeks.org, 2024).

Algorithm 2 Pseudocódigo de Lempel-Ziv

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

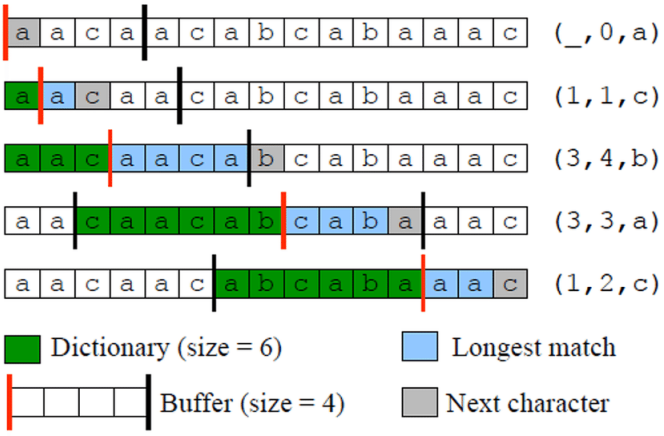

Figura 5: Ejemplo de codificación LZ77. Tomado de *An example of LZ77 encoding fig.4*, por Shi et al., 2018, https://www.researchgate.net/figure/An-example-of-LZ77-encoding$_f ig4_3$22296027

3.3.2. 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 .Arepetida después de dos caracteres, lo que reduce la cantidad de almacenamiento necesaria para patrones repetidos. Ejemplo en la Figura 5.

3.3.3. Cómo funciona LZW

Lempel-Ziv-Welch (LZW), desarrollado por Terry Welch en 1984, se basa en LZ77 pero introduce un enfoque basado en diccionario. LZW asigna códigos a patrones en los datos y almacena estos códigos en un diccionario. A medida que procesa los datos, reemplaza los patrones repetidos con códigos de diccionario, lo que reduce el tamaño del archivo (geeksforgeeks.org, 2024).

3.3.4. Aplicaciones

- LZ77 se usa ampliamente en los formatos de compresión ZIP y GZIP, que son esenciales para la compresión de archivos en los sistemas operativos.
- LZW se usa principalmente en el formato de imagen GIF, donde comprime datos gráficos de manera eficiente.

3.3.5. Ventajas y desventajas

- Ventajas: Los algoritmos Lempel-Ziv son muy efectivos para comprimir grandes conjuntos de datos con patrones repetidos. Son adaptativos, lo que significa que no requieren un conocimiento previo de los datos para comprimirlos. Esto los hace adecuados para una amplia gama de tipos de archivos, incluidos texto, imágenes y ejecutables.
- Desventajas: si bien LZ77 y LZW funcionan bien con datos con patrones, pueden ser menos efectivos con archivos muy aleatorios o ya comprimidos. Además, LZW a veces puede inflar el tamaño del archivo si el diccionario crece demasiado o si no hay suficientes repeticiones en los datos.

4. 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)

Figura 6: 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/

4.1. Transformada discreta del coseno (DCT)

La transformada discreta del coseno (DCT) es una técnica matemática utilizada para transformar los datos en un dominio de frecuencia, lo que facilita la identificación y eliminación de información innecesaria. Es especialmente eficaz en compresión de imágenes y es un componente fundamental del estándar de compresión de imágenes JPEG.

4.1.1. Cómo funciona la DCT

La DCT funciona dividiendo una imagen en bloques más pequeños (normalmente de 8 x 8 píxeles) y convirtiendo los valores de los píxeles de cada bloque en dominio espacial al dominio de frecuencia, como se representa en la Figura 7. En el dominio de frecuencia, la DCT separa la imagen en dos partes: componentes de baja frecuencia (que contienen la mayoría de los detalles visuales importantes) y componentes de alta frecuencia (que a menudo corresponden a detalles finos o ruido).

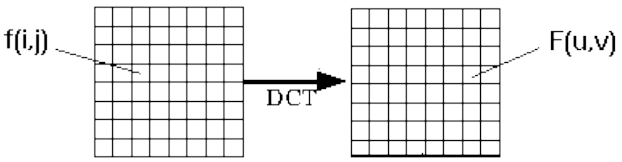

Figura 7: Transformación de una señal o imagen del dominio espacial al dominio de frecuencia. Tomado de *The Discrete Cosine Transform (DCT)*, por Marshall D., 2001, https://https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.htmlDCTenc

La ecuación general para una DCT 1D (N elementos de datos) se define mediante la siguiente ecuación:

$$F(u) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \Lambda(i).cos\left[\frac{\pi.u}{2.N}(2i+1)\right] f(i)$$

La ecuación general para una DCT 2D (imagen N por M) se define mediante la siguiente ecuación (Marshall, 2001):

$$F(u,v) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \left(\frac{.....}{(}2i+1)\right] cos\left[\frac{\pi.v}{2.M}(2j+1)\right].f(i,j)$$

(Marshall, 2001).

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.

Cálculo de la DCT 2D en Figura 8.
La factorización reduce el problema a una serie de DCT 1D:

- Aplicar la DCT 1D (verticalmente) a las columnas
- Aplicar la DCT 1D (horizontalmente) a la DCT vertical resultante anterior.
- o alternativamente, aplicar de forma horizontal a vertical.

Es computacionalmente más fácil de implementar y más eficiente considerar la DCT como un con-

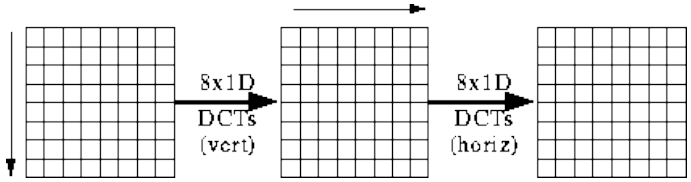

Figura 8: Cálculo de la DCT 2D. Tomado de *The Discrete Cosine Transform (DCT)*, por Marshall D., 2001, https://https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.htmlDCTenc

junto de funciones base que, dado un tamaño de matriz de entrada conocido de (8 x 8), se pueden calcular previamente y almacenar. Los valores se calculan simplemente a partir de la fórmula de la DCT. Las 64 funciones base de la DCT (8 x 8) se ilustran en la figura 9 (Marshall, 2001).

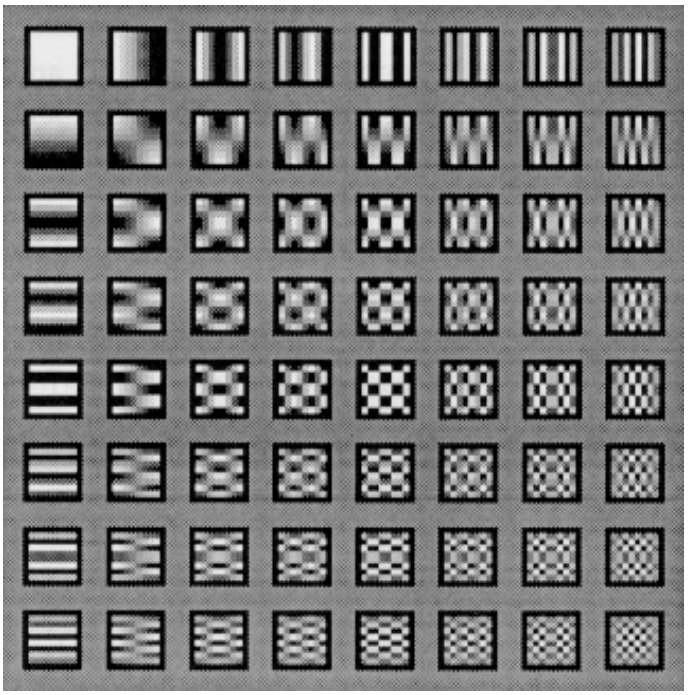

Figura 9: Las 64 funciones base de una matriz de 8 por 8. Tomado de *The Discrete Cosine Transform (DCT)*, por Marshall D., 2001, https://https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.htmlDCTenc

En la compresión JPEG, después de aplicar la DCT, los componentes de alta frecuencia, que son menos perceptibles para el ojo humano, se reducen o se descartan por completo. Los componentes de baja frecuencia restantes se cuantifican y codifican utilizando métodos como la codificación Huffman, dando como resultado una compresión significativa. La compresión JPEG utiliza la DCT para reducir el tamaño de archivo sin causar una pérdida notable en la calidad de la imagen. Por ejemplo, en una imagen JPEG típica, se eliminan pequeñas variaciones en el color o el brillo que el ojo humano puede no detectar, lo que permite reducciones sustanciales en el tamaño del archivo manteniendo la fidelidad visual. Un JPEG de alta

calidad puede conservar el 90 % de los datos de la imagen original, pero reducir el tamaño del archivo en un 70-80 % (Marshall, 2001).

4.1.2. Ventajas y Desventajas

- Ventajas:
  - Compresión eficiente para imágenes con cambios de color graduales o grandes áreas de color uniforme.
  - Equilibra la reducción del tamaño de archivo con una pérdida aceptable de calidad visual.
- Desventajas:
  - En casos de compresión intensa, pueden aparecer artefactos JPEG (distorsiones en bloques), lo que reduce el atractivo visual de la imagen.

4.2. Codificación de vídeo (H.264, HEVC)

Los algoritmos de codificación de vídeo H.264 y HEVC/H.265 utilizan una combinación de compresión espacial y temporal para reducir el tamaño de los archivos de vídeo. Estos algoritmos se utilizan ampliamente para transmitir contenido de vídeo, almacenar vídeos de alta definición (HD) y comprimir archivos de vídeo para su transmisión por Internet.

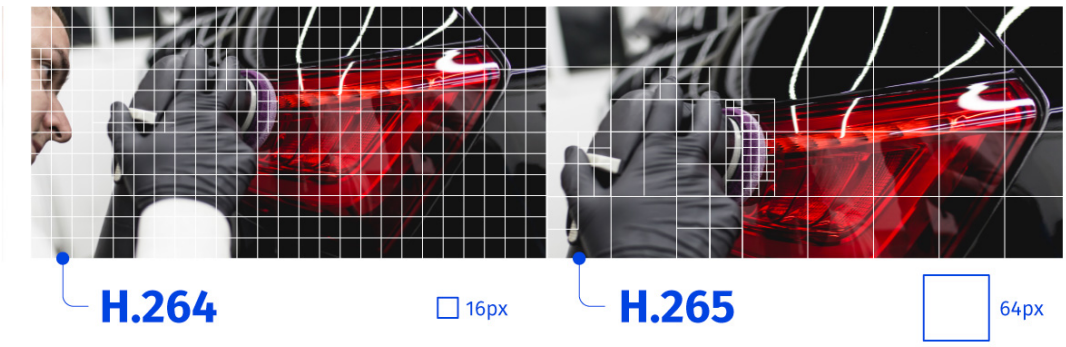

Figura 10: Comparación visual de H.264 y HEVC/H.265. Tomado de *H.264 y H.265 - AVC y HEVC - ¿Cuál es la diferencia?*, por flussonic., 2021, https://flussonic.com/es/blog/news/h264-vs-h265/

4.2.1. Cómo funciona H.264

H.264, también conocido como codificación de vídeo avanzada (AVC), es un estándar de compresión de vídeo popular que utiliza una combinación de compresión intracuadro e intercuadro. La compresión

intracuadro funciona aplicando técnicas como DCT dentro de cada cuadro (similar a cómo JPEG comprime las imágenes), mientras que la compresión intercuadro reduce la redundancia entre cuadros consecutivos codificando únicamente las diferencias (o el movimiento) entre cuadros.

La Figura 11 muestra el diagrama de bloques del codificador. La imagen de entrada se divide en diferentes bloques, y cada bloque puede experimentar una predicción intra utilizando píxeles reconstruidos vecinos en el mismo cuadro como predictor. H.264 admite tamaños de bloque de predicción intra de 16×16, 8×8 y 4×4, y permite diferentes formas de construir las muestras de predicción a partir de los píxeles reconstruidos adyacentes.
Alternativamente, el bloque de entrada puede experimentar una predicción inter utilizando los bloques reconstruidos en los cuadros de referencia como predictor. La predicción inter puede basarse en un tamaño de partición de 16×16, 16×8, 8×16, 8×8, 8×4, 4×8 o 4×4.

La señal residual de predicción de la predicción intra o interpredicción se sometería entonces a una transformación para descorrelacionar los datos. En H.264, se utiliza una transformación de enteros separables 4×4, que es similar a la DCT 4×4 pero evita la falta de correspondencia entre la transformación directa y la inversa. A continuación, los coeficientes de transformación se cuantificarían escalarmente y se escanearían en zigzag. La codificación de longitud variable adaptativa al contexto (CAVLC) puede emplearse a continuación para codificar por entropía los coeficientes de transformación escaneados (Cheung et al., 2010).

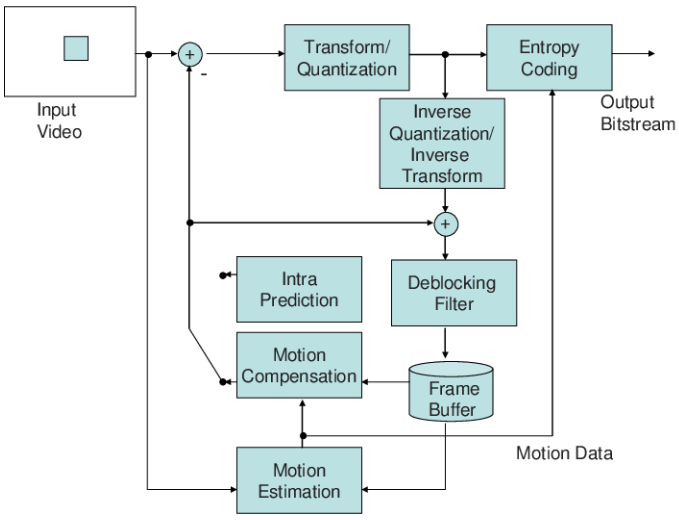

Figura 11: Algoritmo de codificación H.264/AVC. Tomado de *Video Coding on Multicore Graphics Processors*, por Cheung et al., 2010, https://www.researchgate.net/figure/H264-AVC-encoding-algorithm-5$_fig5_2$4127502

Esta combinación de compresión espacial (dentro de un cuadro) y temporal (entre cuadros) permite

que H.264 alcance altos índices de compresión manteniendo una buena calidad de video. Por ejemplo, al transmitir una película, H.264 reduce el tamaño del archivo al eliminar la información visual repetida en los cuadros, lo que significa que se necesitan transmitir menos datos.

4.2.2. Cómo funciona HEVC

HEVC, o H.265, es un estándar de compresión de video todavía más avanzado. Diseñado para ofrecer una compresión incluso mejor que H.264 manteniendo una calidad de video similar o mejor. Utiliza bloques de codificación más grandes, predicción de movimiento más sofisticada y técnicas de cuantificación mejoradas para lograr una mayor eficiencia de compresión (Maung et al., 2015).

Para entender el funcionamiento y transmisión de este tipo de compresión, observar la Figura 12.

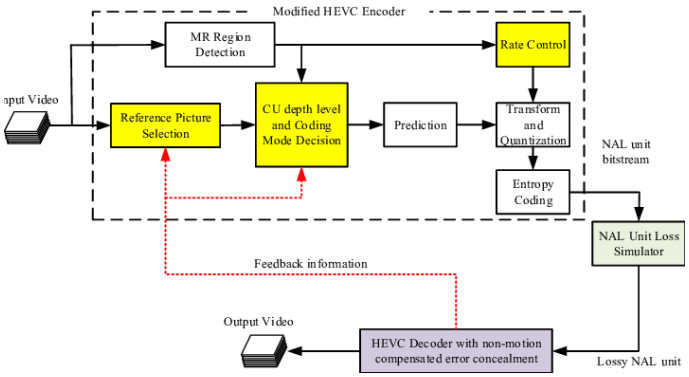

Figura 12: Diagrama de bloques general del sistema de transmisión de vídeo HEVC. Tomado de *Region-of-interest based error resilient method for HEVC video transmission*, por Maung et al., 2015, https://www.researchgate.net/figure/Overall-block-diagram-of-HEVC-video-transmission-system$_fig2_304268810$

HEVC es particularmente eficaz para comprimir videos 4K y de alta definición, donde puede reducir los tamaños de archivo en un 50 % en comparación con H.264 sin una pérdida de calidad notable. Esto lo hace ideal para servicios de *streaming* como Netflix, YouTube y otros, donde la transmisión de video eficiente es esencial para reducir el uso del ancho de banda.

4.2.3. Aplicaciones

- H.264 se utiliza ampliamente en aplicaciones como videoconferencias (Zoom, Skype), servicios de streaming (YouTube, Netflix) y almacenamiento de vídeo (discos Blu-ray).
- HEVC, con su mayor eficiencia, se utiliza en formatos de vídeo de alta resolución como 4K, lo que garantiza una transmisión fluida sin necesidad de un ancho de banda excesivo.

5. 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).

5.1. Compresión en el almacenamiento

Los sistemas operativos suelen utilizar la compresión para minimizar la cantidad de espacio que ocupan los datos en los dispositivos de almacenamiento, como discos duros o SSD. Al comprimir archivos, el sistema operativo puede almacenar más datos en el mismo espacio físico, lo que extiende la vida útil del hardware de almacenamiento y reduce el costo de la administración de datos. Los sistemas de archivos como NTFS en Windows y Btrfs en Linux admiten la compresión transparente, donde el sistema comprime y descomprime automáticamente los archivos sin la intervención del usuario. Esto permite un uso sin problemas de los datos comprimidos, lo que garantiza que los archivos permanezcan accesibles mientras ocupan menos espacio en el disco.

5.2. Compresión en la transmisión de datos

La compresión también es crucial para mejorar la velocidad de transmisión de datos a través de las redes. En muchos casos, enviar datos sin comprimir daría como resultado tiempos de transferencia más prolongados y un mayor uso del ancho de banda. Los sistemas operativos emplean técnicas de compresión como gzip y *DEFLATE* para reducir el tamaño de los archivos que se envían a través de Internet o de redes locales. Por ejemplo, al navegar por la web, muchos servidores comprimen los datos del sitio web antes de enviarlos al navegador del usuario, que luego descomprime el contenido. Esto, permitiendo que exista una carga más rápida de las páginas y un menor consumo de ancho de banda.

La compresión de datos es particularmente importante en las soluciones de almacenamiento en la nube y de copias de seguridad, donde existe la trasnmisión y almacenamiento de grandes volúmenes de datos de forma remota. La compresión de archivos antes de cargarlos reduce el tiempo necesario para las operaciones de copia de seguridad y disminuye el costo del almacenamiento en la nube, ya que se requiere menos espacio para almacenar las versiones comprimidas.

5.3. Compresión en memoria

Además del almacenamiento y la transmisión, la compresión también se puede aplicar a la memoria del sistema (RAM) para mejorar el rendimiento. Los sistemas operativos como Linux utilizan técnicas de compresión de memoria para almacenar más datos en la RAM comprimiendo la información a la que se accede con menos frecuencia. Cuando se comprime la memoria, el sistema puede mantener más aplicaciones y datos en la RAM, lo que reduce la necesidad de intercambiar datos con dispositivos de almacenamiento más lentos, como los discos duros. Esto da como resultado tiempos de acceso más rápidos y un mejor rendimiento general del sistema, especialmente en sistemas con recursos de memoria limitados (Silberschatz et al., 2018).

Por ejemplo, las funciones *zswap* y *zram* de Linux permiten la compresión en memoria, lo que puede resultar útil en sistemas con poca memoria o dispositivos integrados. Estas técnicas ayudan a reducir la cantidad de datos que se intercambian en el disco, lo que acelera la capacidad de respuesta del sistema bajo una carga pesada.

5.4. Ejemplos de compresión de archivos en Linux

Los sistemas operativos Linux ofrecen varias utilidades para la compresión de archivos, incluidas *gzip* y *bzip2*, que se utilizan ampliamente para comprimir archivos y archivos.

- **gzip**: Es una de las herramientas de compresión más usadas en Linux, que utiliza el algoritmo *DEFLATE*. Se suele utilizar para comprimir archivos de texto, registros o paquetes de software. Por ejemplo, al descargar un archivo tarball (.tar.gz), el archivo se comprime normalmente con gzip, lo que hace que sea más rápido de descargar y más fácil de almacenar.

- **bzip2**: Otra herramienta, que proporciona índices de compresión más altos que *gzip*, pero a costa de tiempos de compresión más lentos. Es especialmente útil cuando el espacio de almacenamiento es escaso, sirviendo para comprimir grandes conjuntos de datos o archivos. Por ejemplo, un archivo comprimido con *bzip2* (*.bz2*) normalmente será más pequeño que el mismo archivo comprimido con *gzip*, aunque puede llevar más tiempo comprimirlo.

(Silberschatz et al., 2018).

6. Conclusión

La compresión de datos es una tecnología esencial en la computación, que desempeña un papel fundamental en la optimización del almacenamiento, la transmisión de datos y el uso de la memoria. Permite el manejo más eficientemente de grandes cantidades de datos, lo que hace posible almacenar más información en menos espacio y transferir archivos rápidamente a través del internet.

Los algoritmos de compresión sin pérdida, como la codificación Huffman y LZ77, conservan todos los datos originales, mientras que los métodos con pérdida, como DCT y H.264, sacrifican algunos detalles para lograr mayores reducciones de tamaño, lo que resulta útil en aplicaciones del área de multimedia.

Vamos hacia una era de generación de datos cada vez mayor, ldonde la importancia de la compresión seguirá creciendo. Cadavez es mayor la distribución de videos de alta resolución, simulaciones más complejas y análisis de datos a gran escala, se hacen necesarios los algoritmos de compresión más avanzados para administrar grandes volúmenes de información. En un futuro, será posible que técnicas de compresión más inteligentes se adapten dinámicamente al tipo de datos que se procesan, aumentando la eficiencia y sin comprometer la calidad. La compresión de datos sigue siendo y seguirá siendo un aspecto crucial de la computación, indispensable en años futuros.

Referencias

Keepcoding. (2024). ¿Qué son los algoritmos de compresión de datos? [[Accesado 06-09-2024]]. https://keepcoding.io/blog/algoritmos-de-compresion-de-datos/

Sayood, K. (2012). *Introduction to Data Compression* (4th). Morgan Kaufmann.

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). Huffman Coding | Greedy Algo-3 [[Accesado 06-09-2024]]. https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/

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/

Marshall, D. (2001). The Discrete Cosine Transform (DCT) [[Accesado 06-09-2024]]. https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html#DCTenc

Cheung, N.-M., Fan, X., Au, O., & Kung, M.-C. (2010). Video Coding on Multicore Graphics Processors. *Signal Processing Magazine, IEEE*, *27*, 79-89. https://doi.org/10.1109/MSP.2009.935416

Maung, H., Aramvith, S., & Miyanaga, Y. (2015, octubre). Region-of-interest based error resilient method for HEVC video transmission. https://doi.org/10.1109/ISCIT.2015.7458352

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 Smart Water Network Management Applications. *Journal of Applied Science Process Engineering*, *7*, 554-563. https://doi.org/10.33736/jaspe.2272.2020

Shi, P., Li, B., Thike, P. H., & Ding, L. (2018). A knowledge-embedded lossless image compressing method for high-throughput corrosion experiment. *International Journal of Distributed Sensor Networks*, *14*, 155014771775037. https://doi.org/10.1177/1550147717750374

flussonic. (2021). H.264 y H.265 - AVC y HEVC - ¿Cuál es la diferencia? [[Accesado 06-09-2024]]. https://flussonic.com/es/blog/news/h264-vs-h265/

Taylor, C. (2022). Lossy Compression [[Accesado 06-09-2024]]. https://cyberhoot.com/cybrary/lossy-compression/

geeksforgeeks.org. (2023b). Run Length Encoding and Decoding [[Accesado 06-09-2024]]. https://www.geeksforgeeks.org/run-length-encoding/

comprimeme.wordpress. (s.f.). Compresión con pérdida y sin pérdida [[Accesado 06-09-2024]]. https://comprimeme.wordpress.com/compresion-con-perdida-y-sin-perdida/