PUCE Virtual

CLASE

5

Teoría de la Información y la Comunicación

Compresión de datos sin pérdida

QS WORLD UNIVERSITY RANKINGS

Educación de calidad

INTRODUCCIÓN DE LA CLASE

GRADO / POSGRADO / TECNOLOGÍAS

Estudia a tu tiempo
sin interrupciones

EDUCACIÓN EN LÍNEA DE CALIDAD

En la semana anterior se abordaron métricas fundamentales como la entropía, la información mutua y la redundancia, las cuales permiten comprender cómo se distribuye la incertidumbre dentro de un sistema de datos. Estos conceptos no solo tienen un valor teórico, sino que constituyen la base para diseñar mecanismos que optimicen la transmisión de información en entornos digitales.

En esta quinta semana, el enfoque se desplaza hacia la aplicación práctica de estos principios mediante técnicas de compresión sin pérdida. A partir de la identificación de redundancias en los datos, es posible construir esquemas que reduzcan el tamaño de la información sin alterar su contenido original, lo que resulta esencial en sistemas de comunicación eficientes. Se explorarán métodos clásicos y avanzados como la codificación de Huffman y los algoritmos de la familia Lempel-Ziv, los cuales permiten acercarse a los límites teóricos de compresión definidos por la entropía. Estos algoritmos constituyen herramientas clave en múltiples aplicaciones modernas, desde almacenamiento de datos hasta transmisión en redes.

El estudio de estos temas permitirá al estudiante no solo comprender los fundamentos matemáticos de la compresión, sino también diseñar e implementar soluciones que maximicen la eficiencia informativa, contribuyendo directamente al desarrollo del Reto 2.

5. COMPRESIÓN DE DATOS SIN PÉRDIDA

La compresión de datos sin pérdida constituye una de las aplicaciones más relevantes de la teoría de la información, ya que permite optimizar la transmisión y almacenamiento de datos sin sacrificar la integridad de la información original. A diferencia de otros enfoques donde se tolera cierta degradación, en este tipo de compresión el proceso es completamente reversible, lo que garantiza que los datos recuperados sean idénticos a los originales (Alencar, 2015).

La compresión sin pérdida permite reducir el tamaño de los datos sin alterar su contenido original.

5.1 Fundamentos de Compresión de Datos

La compresión de datos consiste en transformar una representación de información en otra que ocupa menos espacio, manteniendo la capacidad de recuperar el contenido original. En el caso de la compresión sin pérdida, esta transformación es completamente reversible (Alencar, 2015).

La compresión sin pérdida elimina redundancia sin afectar la información original.

Modelo general de compresión sin pérdida

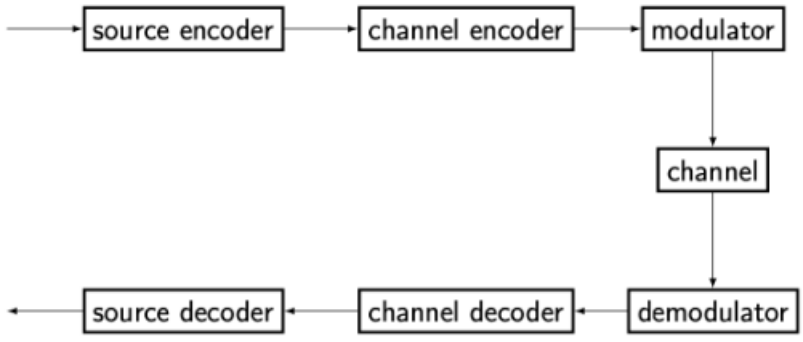
Un sistema de compresión sin pérdida puede describirse mediante tres componentes principales:

1. Fuente de datos: Genera los símbolos con ciertas probabilidades.
2. Codificador: Transforma los símbolos en una representación comprimida.
3. Decodificador: Reconstruye exactamente los datos originales.

Este proceso se puede representar como se muestra en la Figura 1. Este modelo permite comprender que la compresión no es un proceso aislado, sino parte integral de la codificación de la información dentro de sistemas de transmisión eficientes.

Figura 1

Modelo general de un sistema de comunicación digital (incluye compresión)



Nota. La figura muestra los bloques fundamentales de un sistema de comunicación digital, donde se observa cómo la información generada por una fuente es procesada por un codificador —etapa en la que se aplica la compresión sin pérdida—, transmitida a través de un canal y posteriormente reconstruida mediante un decodificador. Adaptado de *Building Blocks of Communication Systems*, University of Hawai'i. Disponible en:

https://descartes.manoa.hawaii.edu/ee643spring2024/morea/02-overview/reading-building-blocks.html

Un sistema de compresión eficiente requiere codificación sin ambigüedad y completamente reversible.

Relación con la entropía

La entropía establece un límite inferior para la compresión:

- No es posible comprimir por debajo de la entropía promedio
- Representa la cantidad mínima de bits necesarios por símbolo

Según Csiszár y Körner (2011), la eficiencia de un algoritmo de compresión se evalúa comparando:

- Longitud promedio del código
- Entropía de la fuente

La entropía define el límite teórico de compresión alcanzable.

Para comprender cómo funciona la compresión sin pérdida, es necesario partir de la representación de una fuente de información discreta, donde cada símbolo tiene asociada una probabilidad de ocurrencia. Esta representación permite calcular la entropía y analizar la redundancia, elementos clave para diseñar esquemas de compresión eficientes.

Tabla 1

Distribución de probabilidad de una fuente discreta

Símbolo	Probabilidad
A	**0.50**
B	**0.25**
C	**0.15**
D	**0.10**

Nota. La tabla presenta una fuente de información discreta con cuatro símbolos y sus respectivas probabilidades de ocurrencia, se observa que el símbolo A tiene la mayor probabilidad de ocurrencia (0.50), lo que indica que aparece con mayor frecuencia en la fuente. En contraste, el símbolo D presenta una probabilidad significativamente menor (0.10), lo que implica que su aparición es poco frecuente. Elaboración propia con base en Alencar (2015).

Las diferencias en probabilidades evidencian la existencia de redundancia en la fuente.

Esta desigualdad en la distribución es precisamente lo que permite aplicar técnicas de compresión sin pérdida. Si todos los símbolos fueran equiprobables, la entropía sería máxima y no existiría margen para optimizar la representación. Sin embargo, en este caso, la presencia de un símbolo dominante genera redundancia estadística.

Desde el punto de vista del diseño de códigos, esta situación sugiere que:

- El símbolo A debe representarse con un código más corto

- Los símbolos menos frecuentes (C y D) pueden representarse con códigos más largos

Asignar códigos más cortos a los símbolos más probables reduce la longitud promedio del mensaje.

Principios fundamentales de la compresión sin pérdida

1. Eliminación de redundancia
La compresión se basa en identificar información repetitiva o predecible dentro de los datos. Esto incluye:

- Repetición de símbolos
- Patrones frecuentes
- Dependencias entre datos

Existen diferentes tipos de redundancia que se debe considerar:
Redundancia estadística: Se presenta cuando ciertos símbolos ocurren con mayor frecuencia que otros. Este tipo de redundancia es explotado por algoritmos como Huffman.
Redundancia estructural: Se refiere a la repetición de patrones o secuencias dentro de los datos. Es aprovechada por algoritmos como Lempel-Ziv.
Redundancia contextual: Surge cuando el valor de un símbolo depende de otros, lo que permite predecir información futura.

Diferentes tipos de redundancia permiten aplicar distintos métodos de compresión.

2. Codificación eficiente
Se asignan códigos más cortos a los símbolos más frecuentes y códigos más largos a los menos frecuentes. Este principio permite reducir la longitud promedio del mensaje.

3. Reversibilidad
El proceso debe garantizar que:

- No exista pérdida de información
- La reconstrucción sea exacta

4. Adaptación a la fuente
Los algoritmos más avanzados se adaptan dinámicamente a las características de los datos, mejorando la eficiencia en escenarios reales.

La compresión sin pérdida depende de la capacidad de modelar correctamente la fuente de información.

5.2 Codificación Huffman

La codificación Huffman es uno de los algoritmos más importantes en la compresión sin pérdida, ya que permite construir códigos de longitud variable óptimos basados en la distribución de probabilidades de una fuente. Su relevancia radica en que logra minimizar la longitud promedio del mensaje, acercándose al límite teórico definido por la entropía (Gorzalka & Deloumeaux, 2012).

Huffman minimiza la longitud promedio del código utilizando las probabilidades de los símbolos.

Construir un árbol donde:

- Los símbolos más probables están más cerca de la raíz (Códigos cortos → símbolos frecuentes)
- Los menos probables están más profundos (Códigos largos → símbolos poco frecuentes)

Esto permite reducir el número total de bits necesarios para representar la información.

La eficiencia de Huffman depende directamente de la distribución de probabilidades.

Algoritmo de Huffman paso a paso

1. Ordenar los símbolos de menor a mayor probabilidad
2. Seleccionar los dos símbolos con menor probabilidad
3. Combinar ambos en un nuevo nodo (sumando probabilidades)
4. Reinsertar el nuevo nodo en la lista
5. Repetir el proceso hasta obtener un solo nodo (raíz del árbol)
6. Asignar códigos binarios (0 izquierda, 1 derecha)

El árbol de Huffman permite generar códigos sin ambigüedad (prefijo libre).

Por ejemplo: supongamos que disponemos de las probabilidades descritas en la Tabla 2

Tabla 2
Distribución de probabilidad para la construcción de un código de Huffman

Símbolo	Probabilidad
A	0.4
B	0.3
C	0.2
D	0.1

Nota. La tabla presenta una fuente discreta con probabilidades asociadas a cada símbolo, utilizada como base para la construcción de un árbol de Huffman y el análisis de compresión sin pérdida; los símbolos están ordenados de mayor a menor o de menor a mayor probabilidad. Elaboración propia con base en Gorzalka y Deloumeaux (2012). Miguel Ortiz Navarrete

Seleccionamos los dos símbolos con menor probabilidad y los combinamos en un nuevo nodo sumando las probabilidades (paso 2 y 3)

$$DC = D + C$$

$$DC = 0.1 + 0.2 = 0.3$$

Incorporamos el nuevo nodo a la lista (paso 4) (DC) = 0.3, B = 0.3, A = 0.4

Repetimos el proceso hasta tener un solo nodo (paso 5)

$$DCB = DC + B$$

$$DCB = 0.3 + 0.6 = 0.6$$

Incorporamos el nuevo nodo a la lista (paso 4) (DCB) = 0.6, A = 0.4

Repetimos el proceso hasta tener un solo nodo (paso 5)

$$DCBA = DCB + A$$

$$DCBA = 0.6 + 0.4 = 1.0$$

Se obtiene la raíz y el árbol como se muestra en la Figura 2 y se asignan los códigos binarios (paso 6) de manera que:

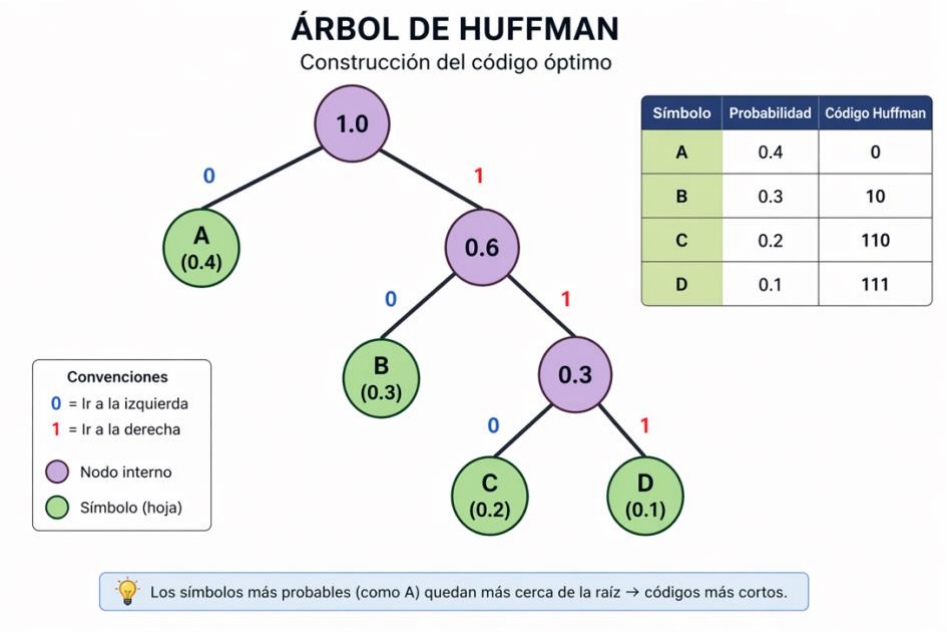
Cada vez que baje por una rama

- Izquierda = 0
- Derecha = 1

Para obtener el código del símbolo A partimos de la raíz, nos dirigimos a la izquierda y llegamos a A; el código de A = 0; igual con B la ruta sería raíz a la derecha y luego a la izquierda; entonces B = 10, para C sería raíz derecha, derecha, izquierda; entonces C = 110, para D sería raíz derecha, derecha, derecha; entonces D = 111. Obteniendo los códigos Huffman como se muestra en la Tabla 3

Figura 2

Árbol de Huffman: Construcción del código óptimo



Nota. La figura muestra la construcción del árbol de Huffman a partir de una distribución de probabilidades, donde se asignan códigos binarios según el recorrido del árbol. Elaboración propia. Miguel Ortiz Navarrete

Tabla 3
Asignación de códigos binarios mediante codificación Huffman

Símbolo	Código
A	**0**
B	**10**
C	**110**
D	**111**

Nota. La tabla presenta los códigos binarios generados a partir del algoritmo de Huffman, donde la longitud del código depende de la probabilidad de cada símbolo. Elaboración propia con base en Gorzalka y Deloumeaux (2012). Miguel Ortiz Navarrete

Longitud Promedio

La longitud promedio se calcula como:

$$L = \sum p(x) \cdot l(x)$$

- $p(x)$ representa la probabilidad.
- $l(x)$ representa el número de bits asignados en el código Huffman para cada símbolo (Tabla 4); esto viene del árbol de Huffman A (0.4) → está más cerca de la raíz → menos pasos → código corto. D (0.1) → está más profundo → más pasos → código largo

Tabla 4
Relación entre probabilidad, código Huffman y longitud de código

Símbolo	p(x)	Código	Longitud
A	0.4	0	1
B	0.3	10	2
C	0.2	110	3
D	0.1	111	3

Nota. La tabla muestra la asignación de códigos binarios mediante el algoritmo de Huffman, donde la longitud de cada código depende de la probabilidad de ocurrencia del símbolo. Elaboración propia con base en Gorzalka y Deloumeaux (2012). Miguel Ortiz Navarrete

$$L = (0.4)(1) + (0.3)(2) + (0.2)(3) + (0.1)(3)$$

$$L = 0.4 + 0.6 + 0.6 + 0.3 = 1.9 \text{ bits}$$

Relación entre entropía, redundancia y compresión

Uno de los principios fundamentales en la teoría de la información establece que la entropía define el límite inferior de compresión. Esto implica que ningún algoritmo puede representar los datos, en promedio, utilizando menos bits que su entropía asociada. Por tanto, la eficiencia de

un esquema de compresión se evalúa comparando la longitud promedio del código generado con la entropía de la fuente.

Calculando la entropía de las probabilidades dadas para cada símbolo obtenemos que:

$$H(X) \approx 1.85 \text{ bits}$$

Huffman se aproxima al límite teórico definido por la entropía.

Enlaces complementarios
Título: Huffman Coding Explained
Descripción: Explica paso a paso la construcción del algoritmo de codificación de Huffman.
Enlace: https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/

5.3 Codificación Huffman Avanzada

La codificación Huffman clásica asume que se conoce previamente la distribución de probabilidades de la fuente. Sin embargo, en escenarios reales —como transmisión en tiempo real, datos dinámicos o flujos de información no estacionarios— esta información no siempre está disponible o cambia constantemente. Por esta razón, se han desarrollado variantes avanzadas que permiten adaptar el algoritmo a condiciones más realistas (Alencar, 2015).

Variantes

1. Huffman adaptativo
2. Huffman por bloques
3. Huffman dinámico

1. Huffman Adaptativo

El Huffman adaptativo construye y actualiza el árbol mientras se procesan los datos, sin necesidad de conocer las probabilidades previamente.

El Huffman adaptativo aprende la distribución de la fuente en tiempo real.

- Inicialmente no se conocen probabilidades
- Se empieza con un árbol básico
- Cada símbolo leído actualiza el árbol

Por ejemplo, si tenemos la siguiente cadena de entrada: A B A C A B

Paso 1: Inicio árbol vacío NYT (Not Yet Transmited)

Paso 2: Leer “A”; No existe en el árbol, se crea el nodo para A

Símbolo	Frecuencia
A	1

Paso 3: Leer “B”; Nuevo símbolo, se añade al árbol

Símbolo	Frecuencia
A	1
B	1

Paso 4: Leer “A”; Ya existe, se incrementa la frecuencia

Símbolo	Frecuencia
A	2
B	1

Paso 5: Leer “C”; Nuevo símbolo, se añade al árbol

Símbolo	Frecuencia
A	2
B	1
C	1

Continúa leyendo el resto de los símbolos, el árbol cambia constantemente, los códigos evolucionan obteniendo

Símbolo	Frecuencia	Probabilidad
A	3	3/6 = 0.5
B	2	2/6 ≈ 0.33
C	1	1/6 ≈ 0.17

Codificando con Huffman se obtiene:

Símbolo	Código	Longitud
A	0	1
B	10	2
C	11	2

Cadena original A B A C A B, cadena codificada 0 | 10 | 0 | 11 | 0 | 10

Longitud total:

- A → 3 veces → 3 × 1 = 3 bits
- B → 2 veces → 2 × 2 = 4 bits
- C → 1 vez → 1 × 2 = 2 bits

Total: 9 bits

El rendimiento del Huffman adaptativo mejora a medida que aprende la fuente.

Las aplicaciones de esta variante van desde Streaming de datos, Redes en tiempo real, Compresión en sistemas sin información previa

2. Huffman por Bloques

En lugar de codificar símbolos individuales, se agrupan en bloques: A, B → AB, BA, AC, etc.

Por ejemplo, supongamos que tenemos esta cadena original A B A C A B

Paso1: Se agrupa en bloque de 2, AB | AC | AB

Paso 2: Calcular la frecuencia y probabilidad:

Bloque	Frecuencia	Probabilidad
AB	2	2/3 ≈ 0.67
AC	1	1/3 ≈ 0.33

Paso 3: Aplicando Huffman

Bloque	Código	Longitud
AB	0	1
AC	1	1

Cadena original A B A C A B, cadena en bloques AB | AC | AB, cadena codificada 0 | 1 | 0

Longitud total: 3 bloques × 1 bit = 3 bits

Tabla 5
Comparison Directa

Método	Bits totales
Sin bloques	9 bits
Con bloques	3 bits

Nota: La compression major en: (9 - 3) / 9 = 66.7%. Eso ocurre porque sin bloques el algoritmo solo analiza símbolos individuales, su redundancia es a nivel individual y no se aprovechan relaciones entre símbolos, no detecta que AB se repite (entropía mayor). Con bloques identifica que AB aparece dos veces, se capturan redundancias contextuales y se detecta que A va seguido de B convirtiéndose en el patrón dominante. Con bloques se reduce la incertidumbre (entropía menor).

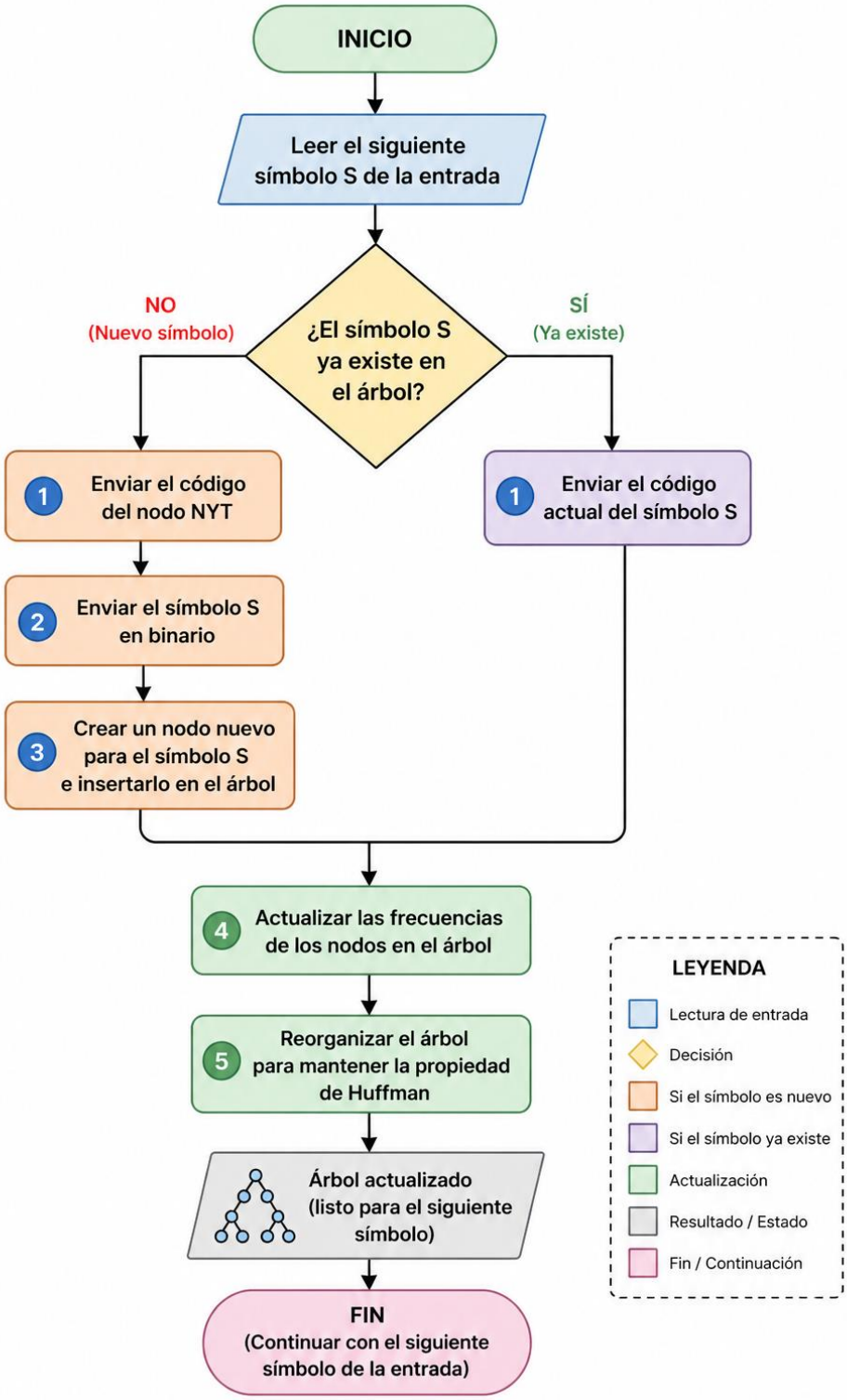
3. Huffman Dinámico

En muchas implementaciones también se le reconoce como adaptativo, a diferencia de Huffman clásico, no se conoce la probabilidad al inicio, y el árbol se construye y se ajusta mientras van llegando los datos.

Al igual que en Huffman adaptativo por cada símbolo leído se gestiona el proceso como se muestra en la Figura 3. La diferencia de Huffman dinámico con Huffman adaptativo se muestra en la Tabla 5

Figura 3

Diagrama de flujo del algoritmo de Huffman dinámico



Nota. La figura representa el proceso de codificación mediante Huffman dinámico, incluyendo la identificación de símbolos nuevos o existentes, la actualización de frecuencias y la reorganización del árbol para mantener la propiedad de Huffman. Elaboración propia. Miguel Ortiz Navarrete

Tabla 6

Comparación conceptual entre Huffman Adaptativo y Huffman Dinámico

Criterio	Huffman Adaptativo	Huffman Dinámico
Naturaleza	Algoritmo específico	Enfoque o concepto general
Definición	Método formal bien definido	Término amplio que agrupa varias estrategias
Actualización del árbol	Se actualiza después de cada símbolo	Puede actualizarse de forma periódica o por bloques
Frecuencia de ajuste	Continua (tiempo real)	Discreta o por intervalos
Estructura del árbol	Siempre mantiene propiedades formales de Huffman	Puede variar según implementación
Algoritmos asociados	FGK (Faller-Gallager-Knuth), Vitter	No tiene un algoritmo único definido
Precisión	Alta (se adapta inmediatamente)	Variable (depende del método usado)
Complejidad	Alta (actualización constante del árbol)	Variable (puede ser menor o mayor)
Flexibilidad	Menor (estructura estricta)	Mayor (permite múltiples enfoques)
Uso típico	Streaming en tiempo real	Procesamiento por lotes o datos por fases
Ejemplo de funcionamiento	Ajusta el código tras cada símbolo leído	Recalcula el árbol cada cierto número de símbolos

Nota. La tabla presenta una comparación entre el algoritmo de Huffman adaptativo y el concepto de Huffman dinámico, destacando sus diferencias en actualización, estructura y aplicación en sistemas de compresión sin pérdida. Elaboración propia con base en Alencar (2015) y Gorzalka y Deloumeaux (2012). Miguel Ortiz Navarrete

5.4 Algoritmos Lempel-Ziv

Los algoritmos de la familia Lempel-Ziv (LZ) constituyen una de las bases más importantes de la compresión sin pérdida moderna (Compresión basada en patrones). A diferencia de métodos como Huffman, que dependen de probabilidades, estos algoritmos trabajan identificando patrones repetidos dentro de los datos y representándolos de forma más compacta; son métodos basados en diccionarios dinámicos (Alencar, 2015).

Lempel-Ziv comprime datos identificando y reutilizando patrones repetidos en lugar de analizar probabilidades.

La idea central de Lempel-Ziv es:

- Detectar subcadenas repetidas
- Reemplazarlas por referencias
- Reducir redundancia estructural

Esto implica que el algoritmo:

- Aprende de los datos

- Construye estructuras internas (ventanas o diccionarios)
- No necesita conocimiento previo de la fuente

Tipos principales de algoritmos LZ

Los más importantes son:

- LZ77 → basado en ventana deslizante
- LZ78 → basado en diccionario explícito

Enlaces complementarios

Título: LZ77 and LZ78: Dictionary-Based Compression
Descripción: Explica de forma clara cómo funcionan los algoritmos LZ77 y LZ78, incluyendo el uso de ventanas deslizantes, diccionarios dinámicos y la detección de patrones repetidos. Presenta ejemplos conceptuales y describe cómo ambos métodos reducen redundancia en los datos.
Enlace: https://www.application-architect.com/posts/lz77-and-lz78-dictionary-based-compression/

1. Algoritmo LZ77 (ventana deslizante)

LZ77 busca patrones repetidos en una ventana de datos ya procesados y los reemplaza por referencias. Cada referencia tiene la forma: (offset, longitud, símbolo)

- offset → cuántos caracteres hacia atrás buscar
- longitud → tamaño del patrón
- símbolo → siguiente carácter

LZ77 trabaja con dos zonas:

- Ventana de búsqueda (pasado)
- Buffer de entrada (futuro)

Y siempre intenta: Buscar la subcadena más larga posible que ya apareció antes

LZ77 siempre busca la coincidencia más larga hacia atrás en la ventana.

Ejemplo

Consideremos la cadena: ABCABCABC
Aplicando el proceso que se explica en el Recurso de profundización 2 en el algoritmos LZ77 se obtiene:

Resultado final
Representación: A B C (3,3) (3,3)

Tabla 7
Interpretación de la representación en LZ77

Parte	Significado
A B C	símbolos iniciales sin compresión
(3,3)	copia “ABC” desde atrás
(3,3)	copia “ABC” nuevamente

Nota. La tabla muestra la interpretación de una secuencia comprimida mediante el algoritmo LZ77, donde los símbolos iniciales se transmiten sin compresión y las referencias posteriores representan repeticiones detectadas en la ventana de búsqueda. Elaboración propia con base en Alencar (2015). Miguel Ortiz Navarrete

2. Algoritmo LZ78 (diccionario dinámico)

LZ78 construye un diccionario de patrones mientras procesa los datos.
Cada entrada tiene la forma: (índice, símbolo) = (índice de la secuencia previa, símbolo nuevo)

LZ78 trabaja así:

1. Lee la cadena de izquierda a derecha
2. Busca la **secuencia más larga que ya esté en el diccionario**
3. Luego:
    - Toma esa secuencia
    - Añade el siguiente símbolo
    - La guarda como nueva entrada
4. Emite: (índice de la secuencia previa, símbolo nuevo)

LZ78 construye patrones cada vez más largos combinando lo conocido con lo nuevo.

Ejemplo

Supongamos que tenemos la cadena ABAABABAAB

Estado inicial

- Diccionario vacío
- Índice inicia en 1

Aplicando el proceso que se explica en el Recurso de profundización 2 en el algoritmos LZ78 se obtiene:

Paso	Cadena	Diccionario	Salida
1	A	{1:A}	(0,A)
2	B	{2:B}	(0,B)
3	A	ya existe	—
4	AA	{3:AA}	(1,A)
5	B	ya existe	—
6	AB	{4:AB}	(2,B)

Resultado final

(0,A) (0,B) (1,A) (2,B)

Alencar, M. S. (2015). Information theory. Momentum Press.

Csiszár, I., & Körner, J. (2011). Information theory: Coding theorems for discrete memoryless **systems (2nd ed.).** Cambridge University Press.

Gorzalka, J. D., & Deloumeaux, P. (2012). Information theory: New research. Nova Science Publishers.

PUCE Virtual

síguenos en:

www.pucevirtual.puce.edu.ec