TEORÍA DE LA INFORMACIÓN

Tratamiento probabilístico de la Información

Compresión de datos

INGENIERÍA DE SISTEMAS

Cursada 2016

COMPRESIÓN Y REDUNDANCIA

- Los datos suelen contener información **no relevante o redundante o repetitiva** (especialmente los datos multimedia: imágenes, video, sonido)

COMPRESIÓN Y REDUNDANCIA

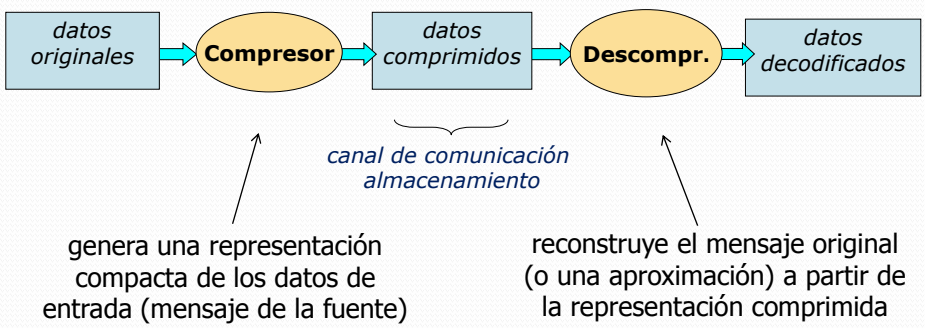
- **El objetivo de la compresión es reducir la cantidad de datos necesarios para representar la información, eliminando (o reduciendo) la redundancia**
- **Permite ahorrar espacio de almacenamiento y reducir costos de transmisión**
- **Enfoques de compresión:**
  - ❖ los datos pueden procesarse para obtener una representación de la *misma* información en menos espacio

    → **métodos de compresión sin pérdida**

  - ❖ algunos datos irrelevantes (o menos importantes) se pueden eliminar para generar una *aproximación* de la información

    → **métodos de compresión con pérdida**

COMPRESIÓN / DESCOMPRESIÓN



MEDIDAS DE PERFORMANCE

compiten entre sí

Tasa de compresión → Relación del tamaño de los datos originales (t_O) vs. tamaño de los datos comprimidos (t_C)

Se expresa generalmente como ***N : 1*** donde $N = t_O / t_C$

Ejemplo archivo original: 100 Mb - archivo comprimido: 20 Mb
→ compresión 5:1 (reducción del 20%)

Tiempo de compresión/descompresión

Los algoritmos pueden ser simétricos o asimétricos (dependiendo si involucran igual costo computacional o no)

Distorsión → error o pérdida de calidad entre datos originales y reconstruidos (para los métodos irreversibles)

Ej: error medio (absoluto o cuadrático) y otras medidas derivadas

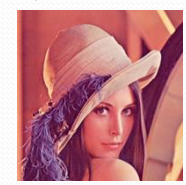
MÉTODOS DE COMPRESIÓN

Sin Pérdida o reversibles (loosless compression)

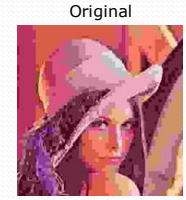
- Mantienen la integridad de la información (los datos descomprimidos *coinciden* con los datos originales)
- Se aplica a texto, bases de datos, código fuente, imágenes críticas, …
- Tasas de compresión moderadas: *2:1* (texto) *4:1* (imágenes)
- La longitud media del código está acotada por la entropía

Con Pérdida o irreversibles (lossy compression)

- No mantienen la integridad de la información (los datos descomprimidos son *aproximación* a los originales)
- Se plantean métodos aplicables a imágenes, video, sonido
- Tasas de compresión altas: *30:1* (imágenes) *200:1* (video)
- No tienen como límite la entropía (pérdida de información)



Original

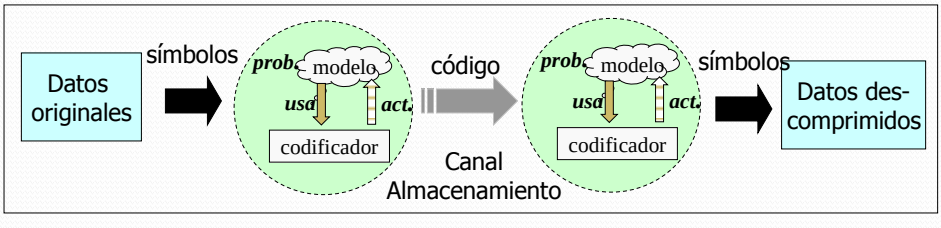


Descomprimida (128:1)

MÉTODOS DE COMPRESIÓN

SIN PÉRDIDA	CON PÉRDIDA
• Métodos estadísticos (codifican un símbolo por vez): ○ Shannon, Fano ○ Huffman estático y semiestático ○ Huffman dinámico (FGK) ○ Codificación aritmética … • Basados en diccionario o sustitucionales: ○ Algoritmos LZ (codifican referencias a cadenas de símbolos previos): LZ77, LZ78 y sus variantes (pkzip, arj, winrar, compress ,pdf, png) • Run-Length (codifican secuencias de símbolos coincidentes)	• JPEG • Run-Lenght c/pérdida • Compresión Fractal • VQ (Cuantificación vectorial) • MPEG • …

COMPRESIÓN : MODELO/CODIFICADOR



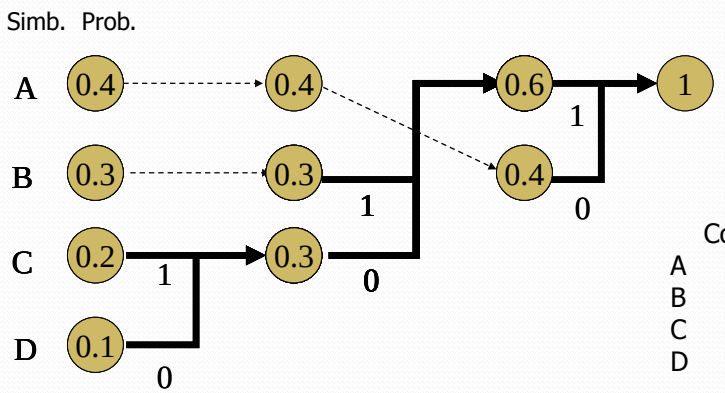
- **Modelo:** Se basa en las *probabilidades* de los símbolos de la fuente (sin memoria, con memoria, extensión a orden n)
- A su vez puede ser:
  - **Estático** → es fijo, es conocido por codificador y decodificador (universal)
  - **Semi-estático** → es fijo, pero se construye a partir de los datos a comprimir
  - **Dinámico o adaptativo** → el modelo se actualiza durante el proceso
- **Codificador:** Codifica los datos de entrada usando el modelo
- El codificador y el decodificador se basan en el mismo modelo

MODELOS: CARACTERÍSTICAS

- **Estático**
  (+) Requiere una única pasada por los datos para codificar
  (-) La distribución de probabilidades (fija) puede diferir de los datos a codificar
  (+) No se debe transmitir/almacenar la distrib. de prob. al decodificador
- **Semi-estático**
  (-) Requiere dos pasadas por los datos: para armar la tabla y para codificar
  (+) La distribución de probabilidades se ajusta a los datos a comprimir
  (-) Se debe transmitir/almacenar la distrib. de prob. al decodificador
- **Dinámico**
  (+) Requiere una única pasada por los datos para codificar
  (+) La distribución de probabilidades se va ajustando a los datos al procesar y no debe transmitirse/almacenarse
  (-) Algoritmos más complejos, de mayor costo computacional

ALGORITMO de HUFFMAN

David Huffman (1950)



Código	
A	0
B	11
C	101
D	100

Ejemplo In: AABADDC Out: 00110100100101

IMPORTANTE: Para que la compresión sea efectiva se debe hacer un procesamiento de los sucesivos códigos a nivel bit

ALGORITMO de HUFFMAN

- Genera un código óptimo (mínima longitud media $<L>$) para el conjunto de símbolos de una fuente S
- Para fuentes especiales, con probabilidades $\sim 1/2^a$, $<L> \sim H(S)$
- No suele alcanzar el límite $H(S)$ porque no considera ´bits fraccionales´ (necesita al menos un bit para codificar un símbolo)

  Ejemplo: S= {s1, s2} con prob. {0.99, 0.01} → $H(S) \approx 0{,}081$
  pero el código asignará 1 bit a cada símbolo

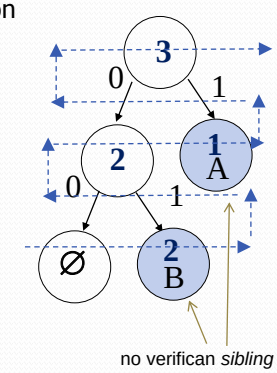
- Si $H(S)$ es grande, la diferencia fraccional $<L> - H(S)$ es despreciable, pero si $H(S)$ es pequeña la diferencia puede ser considerable
- Para la extensión de orden m se cumple que: $H(S) \leq <L_m> < H(S) + 1/m$
  → La performance del código mejora a expensas de un aumento exponencial del tamaño de la tabla de códigos
- Forma parte de la especificación de formatos y estándares de compresión (JPEG, MP3, MPEG, GZIP, …)

HUFFMAN DINÁMICO – ALGORITMO FGK

Faller(1973), Gallager(1978), Knuth(1985)

- El árbol de código inicialmente está vacío y se construye dinámicamente a medida que se procesa el mensaje
- A cada nodo del árbol se le asocia un símbolo y un peso (frecuencia)
- Nodo especial –nodo ∅ - asociado al punto de inserción
- Para que sea un árbol de Huffman debe cumplir siempre la <u>propiedad de *sibling*</u>:

  si se recorren los nodos de abajo hacia arriba y de izquierda a derecha, los pesos de los nodos deben aparecen en orden no decreciente



HUFFMAN DINÁMICO – ALGORITMO FGK

Faller(1973), Gallager(1978), Knuth(1985)

Algoritmo:

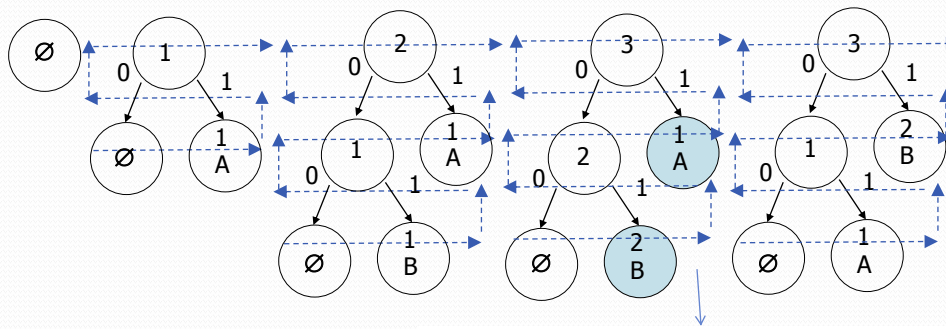
1) Leer el próximo símbolo del mensaje
2) Generar el código:
   - si el símbolo está en el árbol → emitir el código binario asociado a este símbolo
   - si no está → emitir el código del nodo ∅ seguido del símbolo literal nuevo (para el primer símbolo del mensaje no hay código para el nodo ∅)
3) Actualizar el árbol:
   - Si el símbolo está en el árbol → incrementar la frecuencia del nodo y de los nodos en los niveles superiores acordemente
   - Si no está → dividir el nodo ∅ en dos: a la izquierda se deja el nodo ∅ y a la derecha se inserta el nuevo símbolo con frecuencia 1
4) Verificar la propiedad de *sibling* :
   - Si no se cumple → reestructurar el árbol, intercambiando los 2 nodos en conflicto (con sus subárboles si los tuvieran):
     - el nodo de peso mayor y más próximo al inicio del recorrido
     - con el nodo de peso menor y más cercano a la raíz
5) Si no terminó el mensaje volver al paso 1)

HUFFMAN DINÁMICO – ALGORITMO FGK

In: A
Out: **A**

In: B
Out: **0 B**

In: B
Out: **01**



No verifica la propiedad de *sibling*:
Intercambiar el de peso mayor y más cercano al inicio del recorrido con el de peso menor más cercano a la raíz

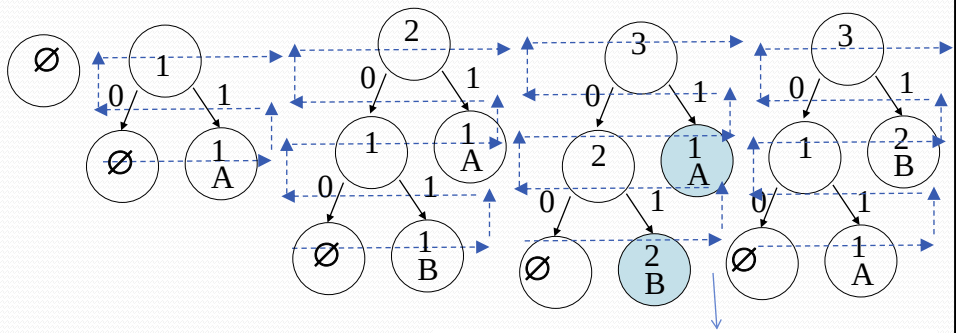
HUFFMAN DINÁMICO – ALGORITMO FGK

El decodificador recibe la salida del codificador, construye el árbol y decodifica siguiendo los mismos pasos que realizó el codificador

In: A
Out: **A**

In: 0 B
Out: **B**

In: 01
Out: **B**



No verifica la propiedad de *sibling* (intercambiar nodos)

RUN LENGHT CODING (RLC)

- Se codifican secuencias de símbolos consecutivos iguales con un par *Símbolo, Repeticiones (longitud de la secuencia)* → utiliza información de contexto
- Se debe fijar el tamaño de la representación de ambos valores

Ejemplo 1:

In: A A A A A A A A B B B C C C C C (ORIGINAL → 16 bytes)

Out: A 8 B 3 C 5 (COMPRIMIDO → 6 bytes ☺)

Suponiendo 1 byte para símbolo y 1 byte para longitud

Ejemplo 2:

In: A B C A B C B A B A C A A A A A (ORIGINAL → 16 bytes)

Out: A 1 B 1 C 1 A 1 B 1 C 1 B 1 A 1 B 1 A 1 C 1 A 5 (COMPRIMIDO → 24 bytes ☹)

- Mejora: fijar un número mínimo de repeticiones para representar una secuencia como un par *S R* y anteponer un bit *flag* para indicar si sigue un par codificado o un símbolo sin codificar (según sea conveniente)
- Aplicable en datos que contienen largas repeticiones de símbolos (ejemplo: gráficos con fondo y objetos de color plano)
- Forma parte de estándares de compresión (ej: JPEG, MPEG, codif. Fax ITU-T T4 Group 3)

RUN LENGHT CODING (RLC)

- En imágenes Blanco/Negro no se codifica el tono (sería redundante, ya que es una secuencia de '0' y '1' alternados)
- Por convención la longitud inicial siempre es para el símbolo '0' (negro)

Ejemplo 1:

In: 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 .....

Out: 8 5 9 7 .....

Ejemplo 2:

In: 1 1 1 0 0 0 0 0 .....

Out: 0 3 5 .....

- En este caso tener en cuenta que las imágenes B/N asignan un bit por tono

BIBLIOGRAFÍA

Cover T., Thomas J., ***Elements of Information Theory,*** 2nd ed., John Wiley & Sons, 2006

Nelson M., Gailly J., ***The Data Compression Book,*** 2nd ed., M&T Books,1996

Bell T., Cleary J., Witten I., ***Text Compression***, Prentice Hall, 1990

Gonzalez R., Woods R., ***Digital Image Processing,*** 2nd ed., Prentice Hall, 2002