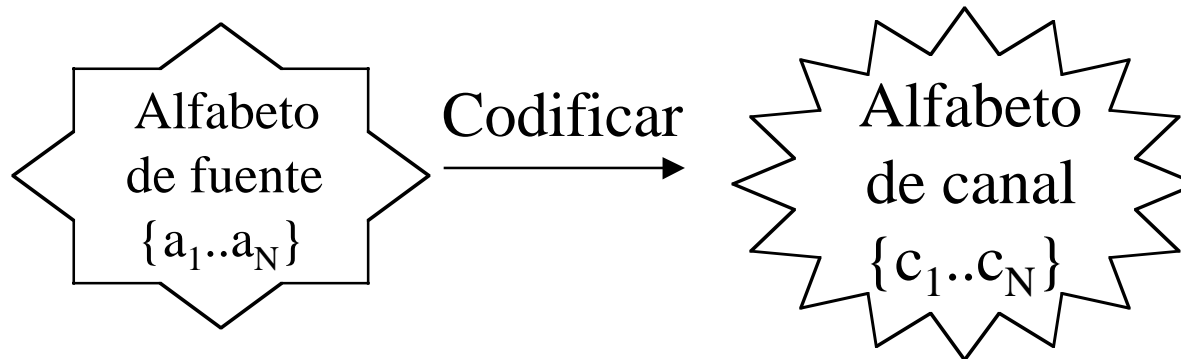

16.36: Ingeniería de sistemas de comunicación

Clase 5: Codificación de la fuente

Eytan Modiano

Codificación de la fuente



- **Símbolos de la fuente**
 - Letras del alfabeto, símbolos ASCII, diccionario de inglés, etc...
 - Voz cuantificada
- **Símbolos del canal**
 - En general puede tener un número arbitrario de símbolos de canal
Típicamente $\{0,1\}$ para un canal binario
- **Objetivos de la codificación de la fuente**
 - Decodabilidad unívoca
 - Compresión
 - Codificar el alfabeto utilizando el menor número posible de símbolos de canal

Compresión

- **Compresión sin pérdidas**
 - Permite la decodificación sin errores
 - Decodabilidad unívoca sin ambigüedad
- **Compresión con pérdidas**
 - Tal vez el código no sea codificable unívocamente, pero hay una probabilidad enorme de que se codifique correctamente

Códigos (libres) prefijo

- **Un código prefijo es un código en el cual ninguna palabra se antepone a otra**
 - Los códigos prefijo son decodificables unívocamente
 - Los códigos prefijo son decodificables instantáneamente
- **La siguiente desigualdad importante se aplica a códigos prefijo y en general a todos los códigos decodificables unívocamente**

Desigualdad de Kraft

Sea n_1, \dots, n_k la longitud de palabras código en un código prefijo (o en cualquiera decodificable unívocamente). Entonces,

$$\sum_{i=1}^k 2^{-n_i} \leq 1$$

Demostración de la desigualdad de Kraft

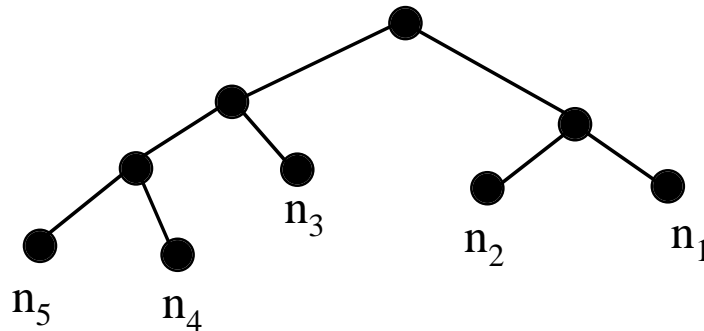
- **Demostración sólo para códigos prefijo**
 - Se puede ampliar a todos los códigos decodificables unívocamente
- **Aplicar palabras código a un árbol binario**
 - Las palabras código deben ser las hojas del árbol
 - Una palabra código de longitud n_i es una hoja a profundidad n_i
- **Sea $n_k \geq n_{k-1} \dots \geq n_1 \Rightarrow$ profundidad del árbol = n_k**
 - En un árbol binario de profundidad n_k , son posibles hasta 2^{n_k} hojas (si todas las hojas están a profundidad n_k)
 - Cada hoja a profundidad $n_i < n_k$ elimina una fracción $1/2^{n_i}$ de las hojas a profundidad $n_k \Rightarrow$ elimina $2^{n_k - n_i}$ de las hojas a profundidad n_k
 - De ahí que,

$$\sum_{i=1}^k 2^{n_k - n_i} \leq 2^{n_k} \Rightarrow \sum_{i=1}^k 2^{-n_i} \leq 1$$

Desigualdad de Kraft - recíproca

- Si un conjunto de números enteros $\{n_1..n_k\}$ satisface la desigualdad de Kraft el código prefijo a se puede hallar con longitudes de palabras código $\{n_1..n_k\}$
 - De ahí que la desigualdad de Kraft sea condición suficiente y necesaria para la existencia de un código decodificable unívocamente
- La demostración se realiza mediante la construcción de un código
 - Dado $\{n_1..n_k\}$, comenzando con n_1 , asignar nodo al nivel n_i para la palabra código de longitud n_i . La desigualdad de Kraft garantiza tal asignación

Ejemplo: $n = \{2,2,2,3,3\}$, (verificar que la desigualdad de Kraft se sostiene)



Longitud media de una palabra código

- La desigualdad de Kraft no nos dice nada sobre la longitud media de una palabra código. El siguiente teorema da una cota reducida estrecha

Teorema: dada una fuente con alfabeto $\{a_1..a_k\}$, probabilidades $\{p_1..p_k\}$, y entropía $H(X)$, la longitud media de un código binario decodificable unívocamente satisface:

$$\bar{n} \geq H(X)$$

Demostración:

$$H(X) - \bar{n} = \sum_{i=1}^{i=k} p_i \log \frac{1}{p_i} - \sum_{i=1}^{i=k} p_i n_i = \sum_{i=1}^{i=k} p_i \log \frac{2^{-n_i}}{p_i}$$

$$\log inequality \Rightarrow \log(X) \leq X - 1 \Rightarrow$$

$$H(X) - \bar{n} \leq \sum_{i=1}^{i=k} p_i \left[\frac{2^{-n_i}}{p_i} - 1 \right] = \sum_{i=1}^{i=k} 2^{-n_i} - 1 \leq 0$$

Longitud media de una palabra código

- ¿Podemos construir códigos próximos a $H(X)$?

Teorema: dada una fuente con alfabeto $\{a_1 \dots a_k\}$, probabilidades $\{p_1 \dots p_k\}$, y entropía $H(X)$, es posible construir un código prefijo (por tanto, unívocamente decodificable) de longitud media que satisfaga:

$$\bar{n} < H(X) + 1$$

Demostración (Códigos Shannon - Fano):

$$\text{Sea } n_i = \left\lceil \log\left(\frac{1}{p_i}\right) \right\rceil \Rightarrow n_i \geq \log\left(\frac{1}{p_i}\right) \Rightarrow 2^{-n_i} \leq p_i$$

$$\Rightarrow \sum_{i=1}^k 2^{-n_i} \leq \sum_{i=1}^k p_i \leq 1$$

\Rightarrow **satisfecha la desigualdad de Kraft**

\Rightarrow **puede hallar un código prefijo con longitudes,**

$$n_i = \left\lceil \log\left(\frac{1}{p_i}\right) \right\rceil < \log\left(\frac{1}{p_i}\right) + 1$$

$$n_i = \left\lceil \log\left(\frac{1}{p_i}\right) \right\rceil < \log\left(\frac{1}{p_i}\right) + 1,$$

Ahora

$$\bar{n} = \sum_{i=1}^k p_i n_i < \sum_{i=1}^k p_i \left[\log\left(\frac{1}{p_i}\right) + 1 \right] = H(X) + 1.$$

Así ,

$$H(X) \leq \bar{n} < H(X) + 1$$

Aproximándose a $H(X)$

- **Tenga en cuenta bloques de N letras código**
 - Existen K^N bloques de letras N posibles (N -tuples)
 - Sea Y el “nuevo” alfabeto fuente de N bloques de letras
 - Si cada una de las letras se genera independientemente,

$$H(Y) = H(x_1..x_N) = N * H(X)$$

- **Codifique Y utilizando el mismo procedimiento anterior para obtener,**

$$\begin{aligned} H(Y) \leq \bar{n}_y < H(Y) + 1 \\ \Rightarrow N * H(X) \leq \bar{n}_y < N * H(X) + 1 \\ \Rightarrow H(X) \leq \bar{n} < H(X) + 1 / N \end{aligned}$$

Donde la última desigualdad se obtiene porque cada letra de Y corresponde a N letras de la fuente original

- **Ahora podemos adaptar la longitud del bloque (N) para que sea arbitrariamente grande y se aproxime arbitrariamente a $H(X)$**

Códigos de Huffman

- Los códigos de Huffman son códigos prefijo especiales que pueden demostrarse óptimos (minimizar la longitud media de las palabras código)



Algoritmo de Huffman:

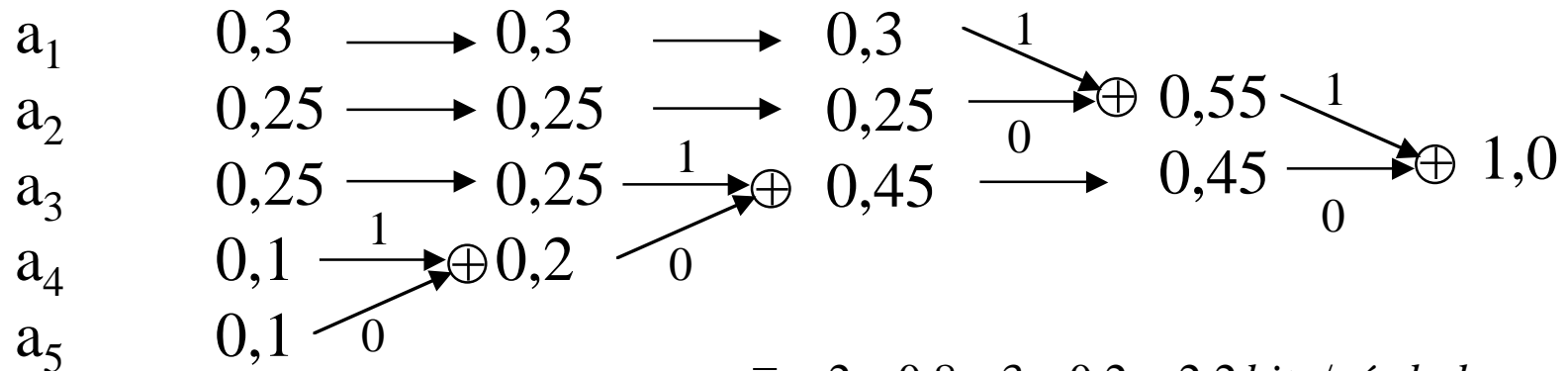
- 1) Dispone las letras fuente en orden decreciente de probabilidad ($p_1 \geq p_2 \dots \geq p_k$)
- 2) Asigna '0' al último dígito de X_k y '1' al último dígito de X_{k-1}
- 3) Combina p_k y p_{k-1} para formar un nuevo conjunto de probabilidades

$$\{p_1, p_2, \dots, p_{k-2}, (p_{k-1} + p_k)\}$$

- 4) Es correcto si sólo le queda una letra, si no, vuelva de nuevo al paso 1

Ejemplo de código Huffman

$A = \{a_1, a_2, a_3, a_4, a_5\}$ y $p = \{0.3, 0.25, 0.25, 0.1, 0.1\}$



$$\bar{n} = 2 \times 0.8 + 3 \times 0.2 = 2.2 \text{ bits / símbolo}$$

<u>Letra</u>	<u>Palabra código</u>
a_1	11
a_2	10
a_3	01
a_4	001
a_5	000

$$H(X) = \sum p_i \log\left(\frac{1}{p_i}\right) = 2.1855$$

$$\text{Códigos Shannon-Fano} \Rightarrow n_i = \left\lceil \log\left(\frac{1}{p_i}\right) \right\rceil$$

$$n_1 = n_2 = n_3 = 2, n_4 = n_5 = 4$$

$$\Rightarrow \bar{n} = 2.4 \text{ bits / símbolo} < H(X) + 1$$

Codificación fuente Lempel-Ziv

- **Las estadísticas de fuente a menudo no son conocidas**
- **La mayoría de las fuentes no son independientes**
 - **Las letras del alfabeto tienen un alto grado de correlación**
P.ej., después de la I va la E, después de la G va la H, etc.
- **Se pueden codificar “bloques” de letras, sin embargo, se requeriría un código muy largo y complejo**
- **Algoritmo Lempel-Ziv**
 - **“Código universal” - funciona sin conocimiento de estadísticas de fuente**
 - **Analiza sintácticamente el archivo de entrada en frases unívocas**
 - **Codifica frases empleando palabras código de longitud fija**
Codificación de longitud variable a fija

Algoritmo de Lempel-Ziv

- **Analizar el archivo de entrada en frases que aún no han aparecido**
 - Entrar frases en un diccionario
 - Numerar su ubicación
- **Observe que cada frase nueva debe ser una frase vieja seguida por un '0' o un '1'**
 - Puede codificar la nueva frase utilizando la ubicación del diccionario de la frase anterior seguida por el '0' o el '1'

Ejemplo de Lempel-Ziv

Entrada: 0010110111000101011110

Frases analizadas: 0, 01, 011, 0111, 00, 010, 1, 01111

Diccionario

Loc	Rep. binaria	Frase	Pal. código	Comentario
0	0000	null		
1	0001	0	0000 0	loc-0 + '0'
2	0010	01	0001 1	loc-1 + '1'
3	0011	011	0010 1	loc-2 + '1'
4	0100	0111	0011 1	loc-3 + '1'
5	0101	00	0001 0	loc-1 + '0'
6	0110	010	0010 0	loc-2 + '0'
7	0111	1	0000 1	loc-0 + '1'
8	1000	01111	0100 1	loc-4 + '1'

Secuencia enviada: 00000 00011 00101 00111 00010 00100 00001 01001

Notas acerca de Lempel-Ziv

- **El decodificador sólo puede decodificar unívocamente la secuencia enviada**
- **El algoritmo no es eficiente para secuencias cortas (datos de entrada)**
- **El rendimiento del código se aproxima a la entropía de fuente para secuencias largas**
- **El tamaño del diccionario debe elegirse con antelación para que se pueda establecer la longitud de la palabra código**
- **Lempel-Ziv se usa frecuentemente para codificar archivos de texto/binarios**
 - **Comprimir/descomprimir bajo unix**
 - **Mismo software de compresión para PC y MAC**