PRÁCTICA-II

Códigos de Huffman Binarios

Veremos ahora un procedimiento para generar códigos bloque instantáneos y óptimos, a partir de una fuente de información de memoria nula, en el caso en el que el alfabeto código es binario, {0,1} (el de mayor interés tecnológico), que puede ser extendido al caso con un alfabeto código general, obtenido por Huffman en el año 1952.

Caso binario

Consideremos una fuente de memoria nula $\mathcal{F}_1 = (\mathcal{A}_1, p_1)$, con $\mathcal{A}_1 = \{a_1,...,a_n\}$, n > 2. Supongamos los símbolos ordenados $a_1,...,a_n$ de modo que

$$p_1(a_1) \geq p_1(a_2) \geq ... \geq p_1(a_{n-1}) \geq p_1(a_n).$$

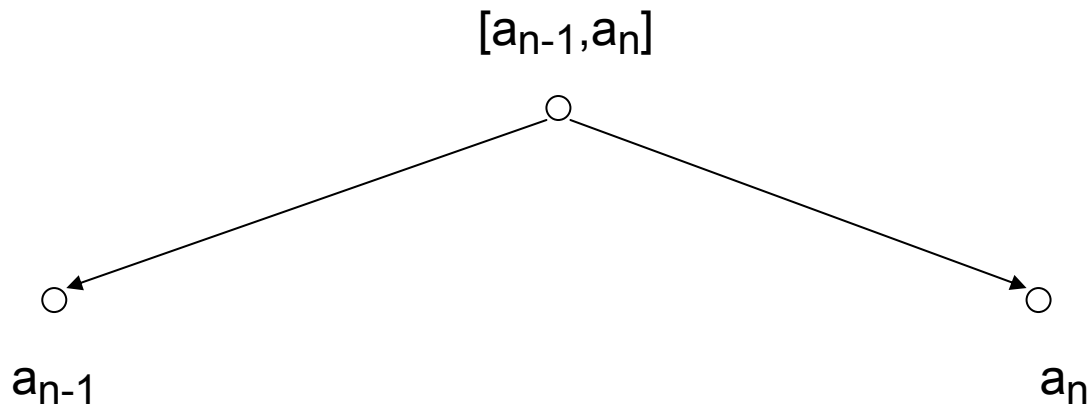
Ahora, los dos últimos símbolos (o en general, dos de los símbolos con las menores probabilidades) se unen para formar un nuevo símbolo, $[a_{n-1},a_n]$, con probabilidad $p([a_{n-1},a_n]) = p_1(a_{n-1}) + p_1(a_n)$.

De esta forma definimos una nueva fuente $\mathcal{F}_2 = (\mathcal{A}_2,p_2)$, con $\mathcal{A}_2 = \{a_1,...,a_{n-2},[a_{n-1},a_n]\}$, y probabilidades

- $p_2(a_i) = p_1(a_i)$, i = 1, ... , n – 2
- $p_2([a_{n-1},a_n]) = p_1(a_{n-1}) + p_1(a_n)$.

Continuándose de esta manera se obtendrá una secuencia finita de fuentes de memoria nula, cada una con un símbolo menos que la anterior, hasta llegar a una fuente, $\mathcal{F}_{n-1} = (\mathcal{A}_{n-1},p_{n-1})$, con un alfabeto de dos símbolos.

Este proceso de reducción puede estructurarse gráficamente en un árbol binario.



El siguiente paso consiste simplemente en notar que el código óptimo instántaneo binario para la última fuente reducida, $\mathcal{F}_{n-1}$, está formado por las palabras `0`, `1`. A partir de aquí, descendiendo por el árbol binario de reducción, se obtiene el código final.

Supóngase dos fuentes consecutivas, $\mathcal{F}_k = (\mathcal{A}_k, p_k)$ y $\mathcal{F}_{k+1} = (\mathcal{A}_{k+1}, p_{k+1})$, de la secuencia anterior, donde

$$\mathcal{A}_k = \mathcal{A}' \cup \{a', a''\} \qquad \text{y} \qquad \mathcal{A}_{k+1} = \mathcal{A}' \cup \{[a', a'']\}$$

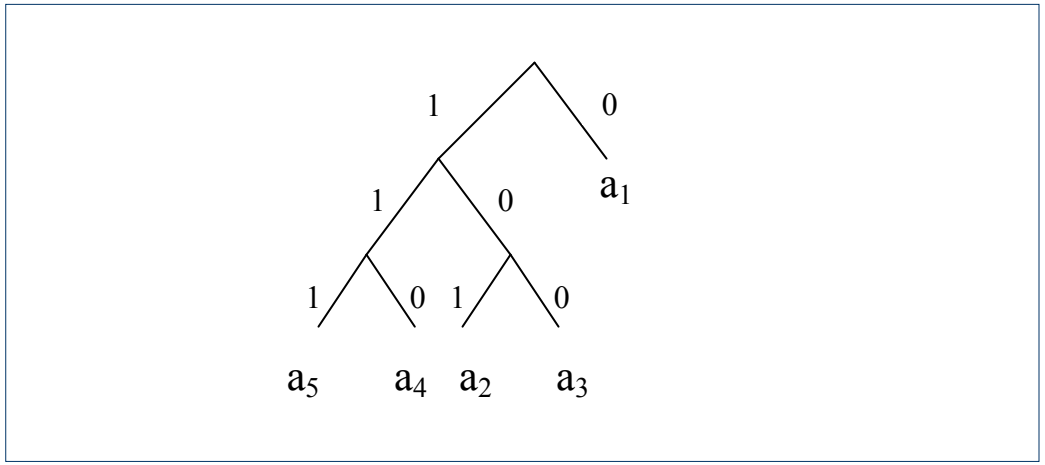
Supóngase asignado el código $\mathbf{C}_{k+1}$ a la fuente $\mathcal{F}_{k+1}$. A la fuente $\mathcal{F}_k$ se le asignaría, por ejemplo, el código $\mathbf{C}_k$ de la manera siguiente:

- $\forall a \in \mathcal{A}'$: $\mathbf{C}_k(a) = \mathbf{C}_{k+1}(a)$
- $\mathbf{C}_k(a') = \mathbf{C}_{k+1}([a', a''])0$
- $\mathbf{C}_k(a'') = \mathbf{C}_{k+1}([a', a''])1$

Ejemplos:

Sea la fuente ($\mathcal{A}$,p) con $\mathcal{A}$ = {a_1,a_2,a_3,a_4,a_5} y p(a_1) = 0.5, p(a_2) = 0.15, p(a_3) = 0.15, p(a_4) = 0.1 y p(a_5) = 0.1.

Un árbol y código de Huffman asociado a esta fuente es el siguiente:



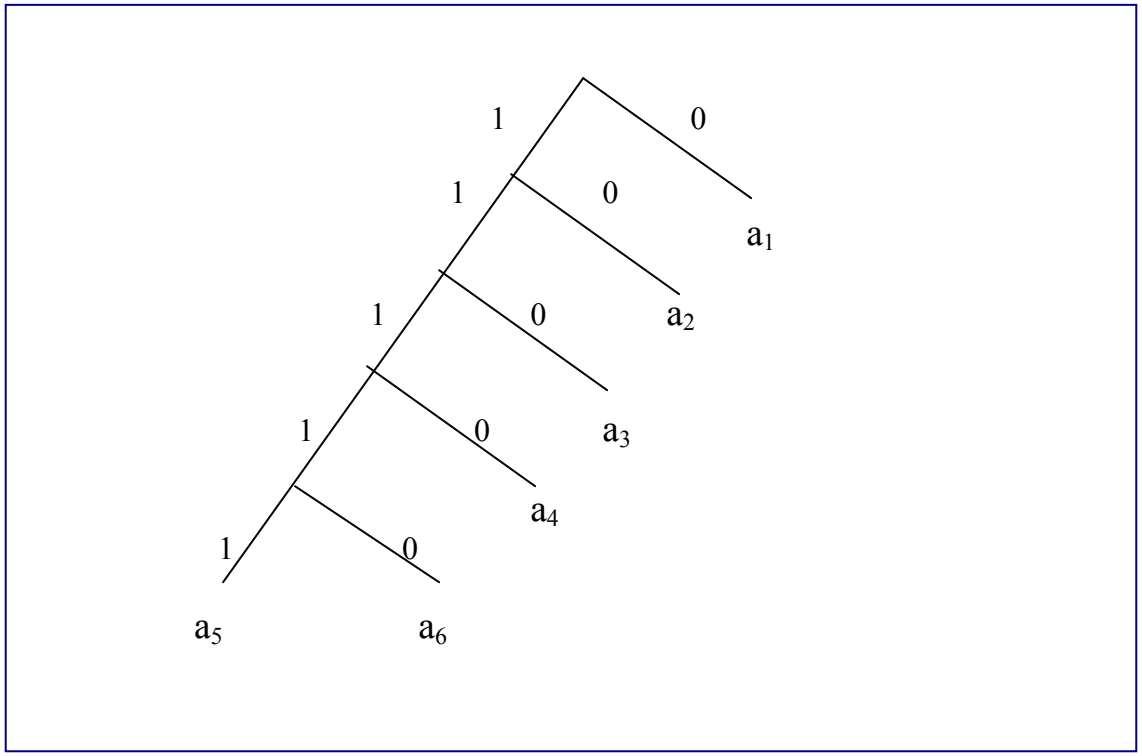
f(a_1) = 0 f(a_2) = 101

f(a_3) = 100 f(a_4) = 110

f(a_5) = 111

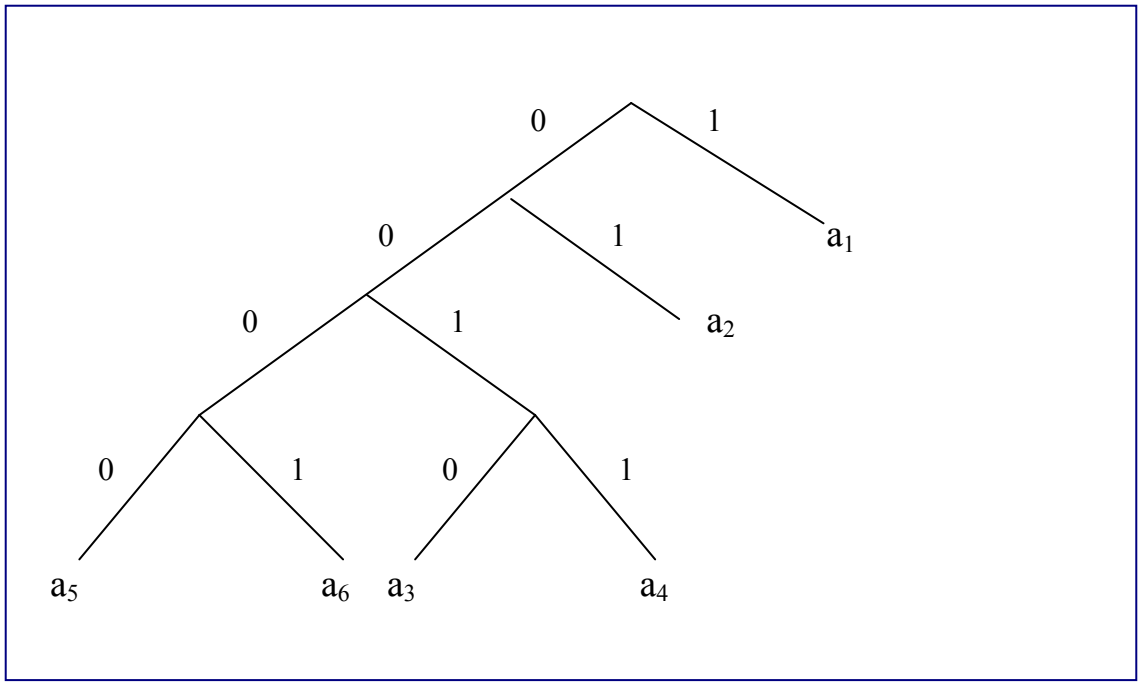
Para la fuente ($\mathcal{A}$,p) con $\mathcal{A}$ = {a_1,a_2,a_3,a_4,a_5,a_6} con p(a_1) = 0.4, p(a_2) = 0.3, p(a_3) = 0.1, p(a_4) = 0.1, p(a_5) = 0.06 y p(a_6) = 0.04.

Un árbol y código de Huffman asociado a esta fuente es el siguiente:



f(a_1) = 0 f(a_2) = 10 f(a_3) = 110 f(a_4) = 1110 f(a_5) = 11111 f(a_6) = 11110

A esta fuente, también podemos, alternativamente asociarle el árbol y el código siguientes:



$f(a_1) = 1 \quad f(a_2) = 01 \quad f(a_3) = 0010 \quad f(a_4) = 0011 \quad f(a_5) = 0000 \quad f(a_6) = 0001$

Se pide desarrollar los programas necesarios, debidamente documentados, para implementar y probar una de las siguientes opciones.

Opción-1 (Simple)

Un módulo que reciba como datos de entrada una fuente de memoria nula y proporcione como salida un código de Huffman binario asociado a la misma.

Opción-2 (Intermedia)

Un conjunto de módulos definido como sigue:

1. Un módulo que a partir del contenido de un fichero, cuya ruta recibe como entrada, calcule las frecuencias o probabilidades de cada símbolo del mismo, asociándole la correspondiente fuente de memoria nula.

2. Un módulo que reciba como datos de entrada una fuente de memoria nula y proporcione como salida un código de Huffman binario asociado a la misma.

3. Un módulo que codifique el contenido del fichero inicial con el código obtenido y lo almacene en un fichero con el mismo nombre que el original y extensión `huf`.

Opción-3 (Completa)

Un conjunto de módulos definido como sigue:

1. Un módulo que a partir del contenido de un fichero, cuya ruta recibe como entrada, calcule las frecuencias o probabilidades de cada símbolo del mismo, asociándole la correspondiente fuente de memoria nula.

2. Un módulo que reciba como datos de entrada una fuente de memoria nula y proporcione como salida un código de Huffman binario asociado a la misma.

3. Un módulo que codifique el contenido del fichero inicial con el código obtenido y lo almacene en un fichero con el mismo nombre que el original y extensión `huf`, adicionalmente en este fichero se debe también incluir, mediante algún formato predefinido*, la codificación utilizada.

4. Un módulo que a partir de un fichero con extensión `huf` decodifique su contenido y lo escriba en un fichero cuya ruta se le proporcionará también como dato de entrada.

* Una manera de introducir y manejar la codificación es a partir del siguiente seudocódigo. Supóngase convenido que en cualquier árbol binario, como es el caso que nos ocupa, de un código de Hufffman binario el `0` etiqueta la rama izquierda y el `1` la derecha, en estas condiciones podemos operar como sigue:

Algoritmo para serializar el árbol del código

```
serializarÁrbol(secuencia,árbol-código)
    obtener el nodo-raíz del árbol-código
    serializarÁrbol(secuencia,nodo-raíz)
```

. . . . . .

```
serializarÁrbol(secuencia,nodo)
    si nodo no es una hoja
      entonces
        añadir un bit 0 a la secuencia
        obtener descendiente nodo-izquierda
        serializarÁrbol(secuencia,nodo-izquierda)
        obtener descendiente nodo-derecha
        serializarÁrbol(secuencia,nodo-derecha)
      en otro caso
        añadir un bit 1 a la secuencia
        añadir a la secuencia el código instantáneo predefinido para el
        símbolo del alfabeto fuente que etiqueta la hoja
```

Algoritmo para deserializar el árbol del código

```
nodo: deserializarÁrbol(secuencia)
        crear nodo-raíz
        deserializarÁrbol(secuencia,nodo-raíz)
        retornar nodo-raíz
```

```
deserializarÁrbol(secuencia,nodo)
  mientras haya bits que leer en la secuencia
    leer el bit b que corresponda en la secuencia
    si b == 1
      entonces
        el nodo es un hoja del árbol
        etiquetar el nodo con el símbolo del
          alfabeto fuente correspondiente
      en otro caso
        crear y añadir al árbol un nuevo
            nodo-izquierdo
        deserializarÁrbol(secuencia,
                                  nodo-izquierdo)
        crear y añadir al árbol un nuevo
            nodo-derecho
        deserializarÁrbol(secuencia,nodo-derecho)
```