Lenguajes de Programación: Trabajo Práctico I

M. Sc. Saúl Calderón Ramírez
Instituto Tecnológico de Costa Rica,
Escuela de Ingeniería en Computación,
PAttern Recognition and MAchine Learning Group (PARMA-Group)

25 de agosto de 2016

El presente proyecto introduce el uso del lenguaje lógico Prolog, y propone el desarrollo de un programa de compresión de texto basado en el algoritmo de Huffman.

- **Fecha de entrega: 20 de setiembre.**
- **Modo de trabajo**: Grupo de tres personas.
- **Tipo de entrega:** digital, por medio de la plataforma TEC-digital.

Parte I
Motivación

Los sistemas de almacenamiento y comunicación digitales con múltiples propósitos, necesitan optimizar el uso de recursos (energéticos, temporales, de espacio, etc.). Para cumplir con tal objetivo de manera efectiva, desde múltiples disciplinas (ciencias de la computación, ingenieria eléctrica, matemática, etc.) se han formulado múltiples algoritmos para **comprimir y descomprimir** la información contenida en una señal. La compresion de una señal digital (compuesta por 1's y 0's) consiste en aplicar algún método que permita disminuir el tamaño original de la señal. Los algoritmos de compresión **sin pérdida** son capaces de aplicar una serie de pasos para construir la señal comprimida, para posteriormente, cuando la señal original necesite ser leída, se descomprima y recupere la señal original completamente idéntica. Los algoritmos de compresión **con pérdida** en cambio, cuando implementan la descompresión de la señal, no logran recuperar al 100 % la señal original. El algoritmo de Huffman a implementar en la presente tarea programada fue propuesto por científico computacional estadounidense David A. Huffman en 1952, como un enfoque de compresión sin pérdida de datos [1].

Parte II
Compresión de un archivo de texto: el algoritmo de Huffman

1. Introducción al algoritmo de Huffman

El algoritmo de Huffman fue diseñado para comprimir una señal digital. La señal digital a comprimir está compuesta por un conjunto finito de símbolos, donde cada símbolo está definido por una cadena específica de bits

Token	número de apariciones
sed	5
et	4
lorem	3
dolor	3
ipsum	2
amet	2
consectetur	3
elit	1
do	2
incididunt	1
aliqua	1
nostrud	1
nisi	1
labore	2
ut	1

Cuadro 1: Histograma de «tokens» del texto de ejemplo.

(1's y 0's). Así por ejemplo, para el caso de la presente tarea programada, en el que se implementará el algoritmo de Huffman para comprimir **archivos de texto** (de extensión *.txt*), un símbolo está definido por una **palabra o *token***. Para ilustrar el funcionamiento del algoritmo, tomaremos el siguiente texto como ejemplo para ejecutar la compresión con el algoritmo de Huffman:

lorem ipsum dolor sed amet consectetur ipsum elit sed do lorem dolor incididunt sed labore et dolor amet aliqua do enim ad enim veniam, et nostrud consectetur sed labore nisi ut sed et et lorem consectetur

El algoritmo de Huffman busca construir para cada símbolo (en este caso, palabra o «token»), una cadena de bits o código de Huffman, que al reemplazarse por cada «token» correspondiente, reduzca el tamaño total del texto. La codificación de Huffman define entonces un diccionario, donde por cada palabra existirá un código correspondiente. Para construir tal diccionario, el algoritmo de Huffman analiza todo el texto, para construir el **diccionario de frecuencias de aparición, o histograma de tokens,** el cual define una entrada por «token», donde tal «token» constituye la llave, y el valor de tal entrada es definido por el número de veces que el «token» sucede en el texto. Así para el ejemplo, anterior, el histograma de tokens está definido como se ve en el Cuadro 1 :

El algoritmo de Huffman busca generar el código más corto para el símbolo con la mayor cantidad de repeticiones. Para ello es necesario asegurarse que la codificación de todos los tokens no sea ambigua, es decir, sea posible reconstruir el texto original a partir de la señal codificada. Para ello el algoritmo de Huffman utiliza el árbol binario como estructura de datos. Tal estructura se explica en la sección 1.1.

1.1. El árbol binario

Un árbol binario es una estructura de datos, la cual está compuesta por un conjunto de **nodos.** Un nodo se entiende como una estructura de datos que puede contener cualquier tipo de datos en su interior (por ejemplo una hilera de caracteres y un número entero). Los datos dentro del nodo forman la «etiqueta» del mismo. Un nodo en un árbol binario tiene como máximo referencia a dos nodos «hijos», un hijo izquierdo, y un hijo derecho. Esto significa que un nodo puede estar «enlazado», como máximo a dos nodos, los cuales semánticamente están en un nivel jerárquico inferior, como se ilustra en la Figura 1.

Un nodo tiene entonces como propiedades su etiqueta (la cual puede estar definida por un conjunto de datos), un hijo izquierdo y un hijo derecho. Tales propiedades definen sus operaciones básicas: la operación *crearNodo(nodo)*, que recibe como mínimo la etiqueta como argumento, y la inserción de la referencia al hijo izquierdo y

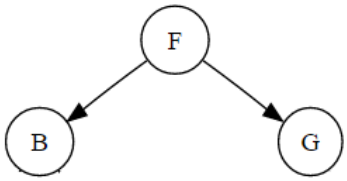

Figura 1: Nodo con la etiqueta «F», con referencias a los nodos hijos de etiqueta «F» y «G».

derecho mediante las operaciones de *insertarHijoIzquierdo(nodo)* e *insertarHijoDerecho(nodo)*, respectivamente. Además se pueden definir otros operadores como por ejemplo *esHoja()* la cual notifica si el nodo operado es una hoja.

La Figura 1 muestra entonces un arbol binario básico, y permite entender los componentes básicos de la estructura de datos abstracta arbol binario:

- **Raíz**: La raíz de un árbol binario se define como el único nodo del conjunto que no desciende de ningún otro nodo. Jerárquicamente es el nodo superior. En el caso de la Figura 1, tal nodo corresponde al de etiqueta «F».
- **Nodos hoja**: En un árbol, un nodo hoja se define como aquel que no tenga enlaces a nodos descendientes, es decir, un nodo hoja es aquel que no tiene hijos.

El árbol binario, como cualquier estructura de datos abstracta, define un conjunto de operadores, los cuales se pueden enlistar como siguen:

- ***insertarRaiz(nodo)*:** Asigna como raíz del árbol al nodo recibido como argumento.
- ***buscarNodo(nodo)***: Busca el nodo dentro del árbol cuya etiqueta coincida con el nodo recibido como argumento.
- ***recorrerArbol()***: Genera una hilera que representa el contenido de un árbol. La representación textual de un árbol puede usar el esquema preorden, posorden e inorden. La representación en preorden especifica que de izquierda a derecha, se muestra primero la etiqueta del nodo padre, luego el contenido del nodo hijo izquierdo, y de último, el contenido del nodo derecho. Así por ejemplo, para el árbol binario de la Figura 1, la representación en preorden vendría dada por: <F, <B>, <G». Los simbolos relacionales de mayor y menor permiten representar cuando inicia un nodo y cuando termina. Para el árbol más complejo de la Figura 2, la representación en preorden es definida como: <F, <B, <A>,<D,<C>,<E», <G,<I,<H»»

Además se pueden definir otras operaciones como la eliminación de un nodo, el cálculo de la profundidad del árbol, etc.

2. El algoritmo de Huffman

El algoritmo consiste en construir «de abajo hacia arriba» un árbol binario para definir el código de cada «token». Al terminar de construir el arbol binario, existirá un nodo por cada «token», y que además contendrá la cantidad total de repeticiones de todos los nodos hijos, en cada nodo. La secuencia de pasos a implementar consiste en:

1. Crear un nodo por cada «token», incluyendo en su etiqueta además el número de repeticiones en el texto analizado (usando el histograma de apariciones). Agregar todos los nodos a una lista.
2. Tomar y remover los dos nodos de menor frecuencia de la lista y unirlos en un nuevo nodo (el cual tendrá los dos nodos con menor frecuencia como hijos). Este nuevo nodo tendrá como etiqueta un «token» nulo (simbolizado por un «#»), y como cantidad de apariciones, la suma de la cantidad de apariciones de los dos nodos hijos. El nuevo nodo se inserta en la lista de nodos.

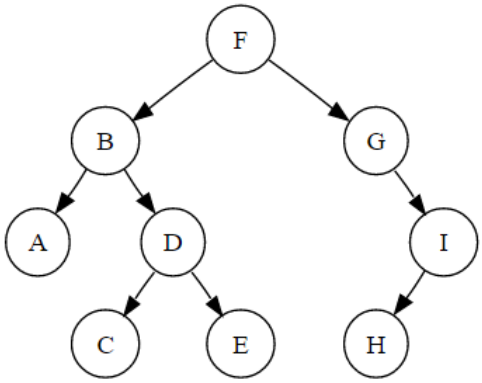

Figura 2: Nodo con la etiqueta «F», con múltiples descendientes.

Token	Número de apariciones	Nodo
do	500	<do-500>
da	200	<da-200>
coisa	10	<coisa-10>
menina	8	<menina-8>

Cuadro 2: Histograma de «tokens» del texto de ejemplo, con el nodo correspondiente en la lista de nodos.

3. Repetir el paso 2 hasta que exista un solo nodo dentro de la lista. Cuando exista un único nodo en la lista de nodos, el mismo se toma como la raíz del **árbol de Huffman.**

El árbol de Huffman es utilizado para construir el código de Huffman para cada «token». Dada la naturaleza del algoritmo, los «tokens» **con mayor cantidad de apariciones en el texto son incluidos en el árbol de último, por lo que estarán más cerca de la raíz del árbol binario**.

Para ilustrar la construcción del árbol de Huffman, se tomará un histograma de «tokens» más pequeño, como se muestra en el Cuadro 2:

Una vez convertidos todos los «tokens» a nodos (con el «token» y su frecuencia como etiquetas), se toman los nodos con menor frecuencia de aparición (coisa y menina), y se agrupan en un solo nodo, como se ve en el cuadro 3.

Luego, se toman de nuevo los dos nodos con menor cantidad de apariciones, para formar uno nuevo (El nodo anónimo con 18 apariciones y el nodo con la etiqueta «da»), como se observa en el cuadro 4.

Finalmente, se toman los dos últimos nodos y se fusionan en uno solo, tal cual se aprecia en el Cuadro 5.

Cuando existe un elemento en la lista, el algoritmo se detiene, y asigna tal elemento como raíz. Gráficamente el árbol expresado en símbolos en el Cuadro 5, se muestra en la Figura 3.

Para construir el código de Huffman para cada «token», se recorre el árbol binario, y se concatena el bit 0 o 1, dependiendo del hijo a iterar (izquierdo o derecho respectivamente). De esta manera, cada vez que se llega a una hoja (correspondiente a un «token»), se obtendrá el código correspondiente. De esta manera se construye el diccionario de Huffman, como se observa en el Cuadro 6.

Nodo
<do-500>
<da-200>
<#-18,<menina-8>, <coisa-10>>

Cuadro 3: Lista de nodos, primera iteración.

Nodo
<do-500>
<#-218,<#-18,<menina-8>, <coisa-10>>,<da-200>>

Cuadro 4: Lista de nodos, segunda iteración.

Nodo
<#-718,<#-218,<#-18,<menina-8>, <coisa-10>>,<da-200>>,<do-500>>

Cuadro 5: Lista de nodos, tercera iteración.

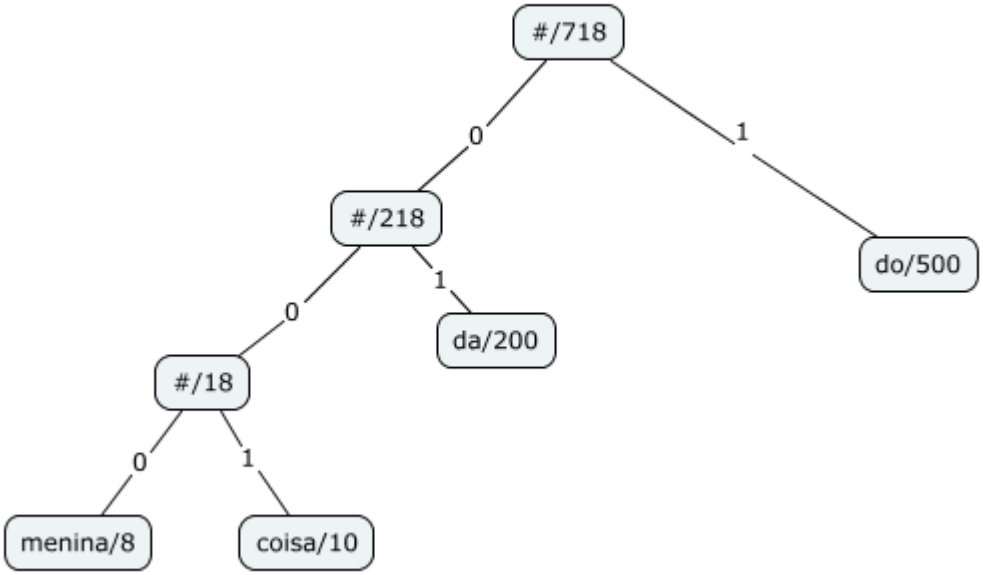

Figura 3: Arbol de Huffman.

Token	Código de Huffman
do	1
da	01
coisa	001
menina	000

Cuadro 6: Diccionario de Huffman.

Observe que el árbol de Huffman garantiza que para los tokens con mayor frecuencia, se asigna el código más corto, y además, que ningún «token» sea prefijo de otro, lo que hace posible la decodificación directa, del texto comprimido (producto del reemplazo de cada «token» por su correspondiente código de Huffman).

El texto comprimido se obtiene entonces al reemplazar cada «token» por el código de Huffman especificado en el diccionario de Huffman. Es importante precisar que en el encabezado de tal archivo, debe incluirse el diccionario de Huffman, de modo que el texto comprimido pueda ser descomprimido posteriormente.

Parte III
Requerimientos del programa

El programa debe proveer una interfaz de línea de comandos que permite comprimir y descomprimir cualquier archivo de texto de extensión .txt. Además debe escribir en otro archivo, el diccionario generado, con el nombre *diccionario.txt*.

3. Implementación

El programa debe implementarse en su totalidad usando Prolog, explotando al máximo las facilidades pertinentes.

Referencias

[1] Mamta Sharma. Compression using huffman coding. *IJCSNS International Journal of Computer Science and Network Security*, 10(5):133–141, 2010.