Universidad de Costa Rica

Escuela de Computación e Informatica

CI-1101 Programación I

III Tarea Programada

Estudiante:
Luis Felipe Barquero Jiménez
B40857

Profesor:
M.sc Braulio Solano Rojas

26 de Noviembre 2014

1. Tabla de Contenidos

Índice

2. Introducción

2.1. Descripción del Problema

Un de los grandes paradigmas de la computación es el uso eficiente de la memoria que permita almacenar la información y trasmitirla. Es por esta razón que en la década los infomáticos buscaban inventar un código binario que fuera lo más eficiente posible. David Hufmann en 1951 publica un artículo sobre un método de construcción de códigos con la cantidad mínima de redundancias. Su método fue realmente revolucionario ya que Hufmann demostró que su método produce el código más eficiente.
En este pequeño proyecto se implementará el código Hufmann para comprimir y descomprimir un archivo de texto.

2.2. Descripción de la Metodología

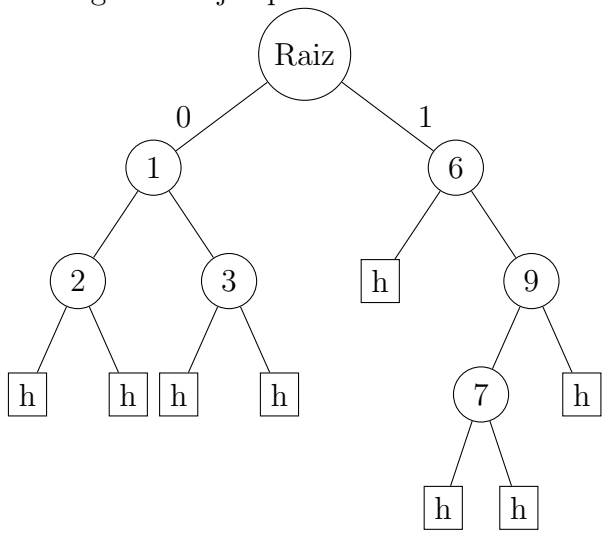
La Metodología utilizada en este proyecto consistió en descomponer el trabajo en pequeñas sub-tareas (la lectura de los caracteres del texto, la creación de la lista, la construcción del arbol binario, la serialización de los objetos,...). La realización de estas sub-tareas fue completamente secuencial, en la manera en la que se plantea el enunciado e instrucciones del problema.

3. Análisis del Problema

El objetivo de los códigos de Hufmann es representar los caracteres de tal manera que los más frecuentes posean la representación más pequeña y los menos frecuentes la más larga, así de esta manera se logra crear un código que sea más eficiente para almacenar la información. En nuestro programa se debe tomar un archivo de texto y crear su código Hufmann.
Analizaremos el problema desde un punto de vista algorítmico: En primer lugar, sea $S = \{s_1, \ldots, s_n\}$ el conjunto de símbolos que puede tener nuestro archivo de texto. Un archivo de texto es una combinación ordenada de estos símbolos.La primera tarea que se debe realizar es contar los símbolos presentes en el archivo, entonces nombramos $a_1, \ldots, a_n$ la cantidad de apariciones que tiene el símbolo i. Note que solo estamos interesados en los caracteres que aparecen en el texto, por eso si $a_i = 0$ lo omitimos . El total de caracteres es $T = \sum_{i=1}^{n} a_i$

Luego, sea $f_i = \frac{a_i}{T}$, la frecuencia con la que aparece s_i. Esta notación nos favorece pues $\sum_{i=1}^{n} f_i = 1$ y además la si seleccionamos un caracter al azar, la probabilidad que tal caracter sea s_i es f_i.
El problema se puede analizar mediante el uso de arboles binario. Si tenemos un arbol como se muestra en la figura 1, podemos asignar 0 al codigo de un hijo que este a la izquierda y un 1 a un hijo a la derecha. Así le podemos asignar a cada hoja un caracter, y de esta manera se obtiene un código que se puede decifrar mediante el arbol.

Figura 1: Ejemplo de arbol binario



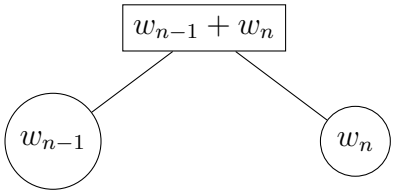
Se define la altura L_n de un nodo como la cantidad de aristas que es necesario recorrer para llegar de la raiz a el nodo n. Si $L_1, \ldots, L_n$ denotan son las alturas de las hojas de un arbol talque: $L_1 \leq \ldots \leq L_n$ y si $w_1, \ldots, w_m$ es una permutación de los $f_i \neq 0$ tal que $W_1 \geq \ldots \geq W_m$ por la desigualdad del reacomodo $\sum_{k=1}^{m} w_k L_k$ minimiza el producto de parejas. El problema entonces se convierte en encontrar valores para las alturas de las hojas de un arbol $L_1 \leq \ldots \leq L_m$ que minimizen $\sum_{k=1}^{m} w_k L_k$.

Proposición 3.1 *$L_1, \ldots, L_n$ son las alturas de las hojas de un arbol binario si y solo si $\sum_{k=1}^{n} 2^{-L_k} = 1$*

Como coloralario de la proposición anterior si $L_1, \ldots, L_n$ son las la altura de las hojas de un arbol binario, entonces $L_{n-1} = L_n$.

El algoritmo de Hufmann: Dada las frecuencias $w_1 \geq \ldots \geq w_n$. Cree una lista ordenada de arboles simples que contengan las frecuencias en la raíz.

1. Si $n = 1$ devuelva el arbol con el elemento en la raíz.
2. Tome w_{n-1} y w_n, cree un arbol de la siguiente manera:



3. Introduzca el arbol creado en la lista, de tal manera que quede ordenado por las frecuencias en la raíz.

Ahora se analizara la manera en que se implemento los códigos de Huffman en Java.

Lectura de los caracteres Para esta tarea se utilizó de la clase `java.io` las clases FileReader, BufferReader, File. Como CHAR puede tomar 256 valores, se creó un vector de 256 entradas donde se contabilizó las apariciones de cada caracter. Luego con estas apariciones y las cantidad total de caracteres se calculó las frecuencia.

Creación de la lista ordenada Para la creación de la lista ordenada se utilizó la clase `ListaOrdenada.java`, primero se crearon árboles con el caracter y su respectiva frecuencia (en un objeto de clase SimboloHuffman comparable) como la raíz y ambos hijos `null`. Luego se insertaron los arboles en la lista ordenados por la frecuencia de la raíz.

Construcción del arbol Aquí comienza el algoritmo de Huffman. Primero si la lista tiene al menos dos elementos (`!estaVacia()`) entonces se toma los dos árboles menores y se crea un nuevo árbol con hijos los dos árboles selecionados y con frencuencia igual a la suma de las frecuencias. Cuando quede solamente un elemento en la lista el metodo devuelve este elemento.

Codificación Para reescribir el mensaje en los códigos de Huffman, se realizó el siguiente proceso: Primero se recorrió todo el árbol asignado códigos a las hojas, además se había guardado una copia (referencia de memoria) de la lista, por lo que para cada caracter se buscó su código en la lista y este fue concatenado en un String llamado mensaje.

Serialización Para la serialización del mensaje primero se convirtió el mensaje (hilera binaria) a un BitSet. Como la clase BitSet implementa Serializable, entonces simplemente se serializa el mensaje en bits (mensajeCodificado). Para serializar el arbol se utilizón un método propio llamado (`serializa()`), este recorre el arbol y concatena la siguiente informacón (El elemento: '?' SI ES NULL, y si no su caracter. De donde proviene # D # de la derecha, # I# de la izquierda, # F# es una hoja.). Finalmente String implementa Serializable, por lo que simplemente se serializa.

Deserialización La deserialización del mensaje en bits(mensajeCodificado) fue fácil pues BitSet es serializable. Para el árbol, primero se deserializó el string del arbol (arbolString) y apartir de este se creó el arbol binario (note que este árbol no tiene las frecuencias ni los códigos, solo los caracteres). Luego con este árbol y con el mensaje se descomprime el mensaje y se crea un archivo de texto con su información.

4. Diseño de Clases

El siguiente diagrama de clases UML, ilustra el diseño orientado a objetos que se utilizó:

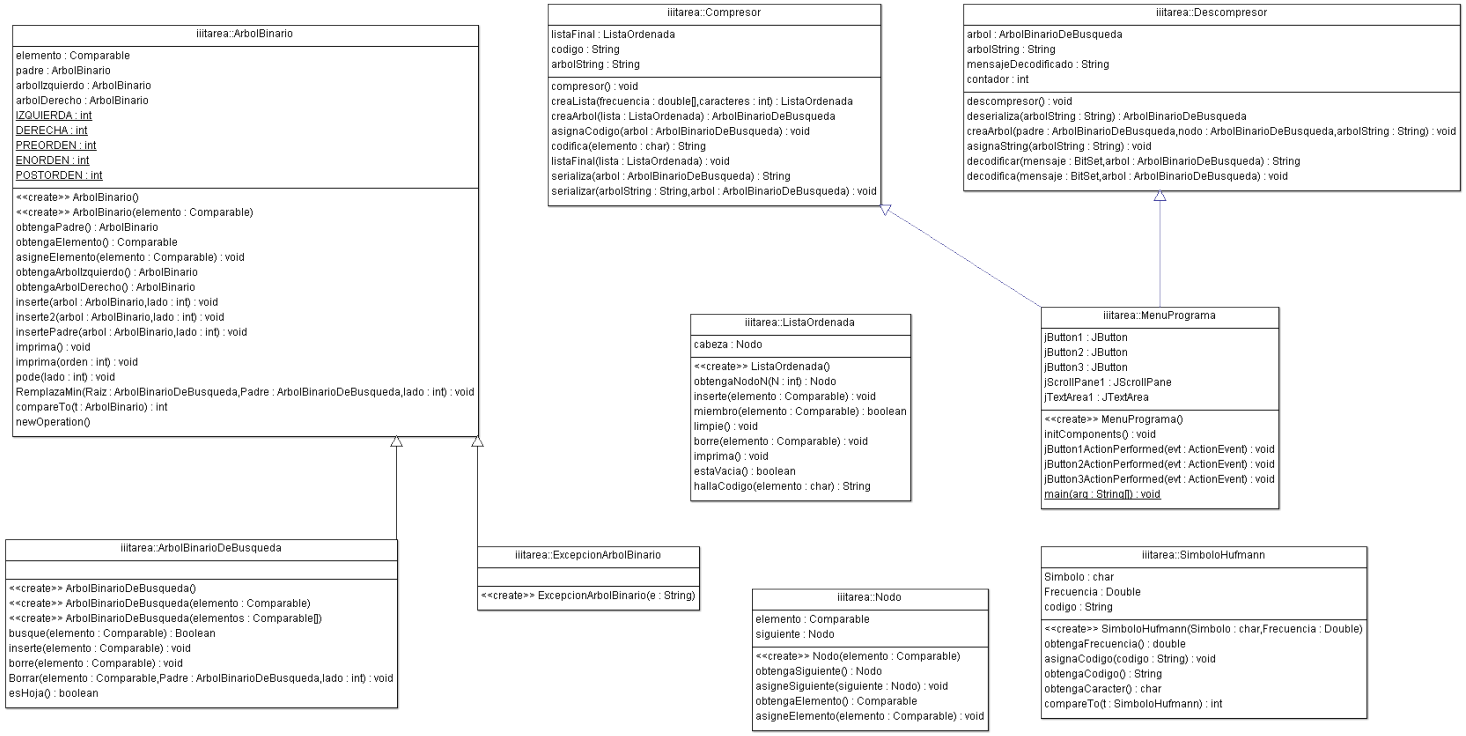


Figura 2: Diseño de Clases.

5. Casos de Prueba

Para probar el programa se eligieron dos archivos de texto, el primero un texto pequeño con muchos caracteres extraños y desordenados. El segundo es un poema de J.R.R. Tolkien, llamado Namarie.

6. Resultados de los Casos de Prueba

La interfaz gráfica funcionó correctamente:

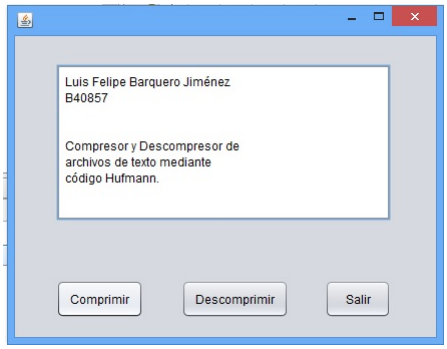


Figura 3: Interfaz

Luego la primer prueba funcionó bien, pues se creó exitosamente los archivos serializados, se deserializaron y el producto final fue identico al inicial. Para la segunda prueba se utilizaron algunos `System.out.print()` para corroborar el funcionamiento de las partes del programa y se obtuvo:

```
Caracter N (código entero 78).
Caracter i (código entero 105).
Caracter m (código entero 109).
Caracter r (código entero 114).     0.03982843137254902
Caracter o (código entero 111).     0.03982843137254902
Caracter d (código entero 100).     0.1482843137254902
Caracter e (código entero 101).     0.011642156862745098
Caracter l (código entero 108).     0.004901960784313725
Caracter   (código entero 32).      0.010416666666666666
Caracter b (código entero 98).      0.001838235294117647
Caracter y (código entero 121).     0.014093137254901961
```

```
-
Nimrodel by J. R. R. Tolkien

An Elven-maid there was of old,
A shining star by day.
Her mantle white was hemmed with gold,
Her shoes of silver-grey.
```

7. Análisis de los Resultados de las Pruebas

Los resultados fueron exitosos, sin embargo si un archivo de texto excede los 40000 caracteres como un dicionario o una novela, el programa se cae por

desbordamiento del buffer.

Referencias

[1] Knuth, D. (1997) *The Art of Computer Programming Volume 1 : Fundamental Algorithms.* Adisson-Wesley, E.E.U.U.