UNIVERSIDAD EAFIT

Informe Final

Implementación del Algoritmo de Codificación de Huffman

Sergio Atehortua Ceferino & Santiago Montoya Angarita

30/05/2014

ST0247 - ESTRUCTURA DATOS Y ALGORITMOS2

1. **Enunciado:**

   Consiste en la implementación de un algoritmo archiconocido de codificación, El algoritmo de Huffman es un algoritmo para la construcción de códigos de Huffmann, desarrollado por David A. Huffmann en 1952 y descrito en “A Method for the Construction of Minimum-Redundancy Codes”. Este algoritmo toma un alfabeto de n símbolos, junto con sus frecuencias de aparición asociadas, y produce un código de Huffman para ese alfabeto y esas frecuencias.

   El algoritmo de huffman es utilizado para convertir una cantidad dada de símbolos en código huffman, utilizando como herramienta un árbol binario, que contendrá todos los árboles asociados a cada símbolo implicado, el cual después de realizar el recorrido del árbol para llegar a cada símbolo, permitirá obtener el código huffman del determinado símbolo con el que se esté trabajando.

   Después de que se posee las equivalencias de cada letra se procede a reemplazar en la frase o conjunto de símbolos que se proporcionaron desde el principio, por sus correspondientes equivalencias, obteniendo así el conjunto de símbolos dados traducido a código huffman el cual se da en binario.

2. **Modelo conceptual:**

3. **Algoritmos**:

-Generar el árbol:

```
    public static Node genTree(ArrayList<Node> list){
        if(list.size()>1){
            list.add(new Node(
                    (char)0,

list.get(0).getHowMany()+list.get(1).getHowMany(),
                    list.get(0),
                    list.get(1)));
            list.remove(0);
            list.remove(0);
            Collections.sort(list);
            return genTree(list);
        }
        return list.get(0);
    }
```

-Crear Lista:

```
    public static ArrayList<Node> List(String text){
        ArrayList<Node> list = new ArrayList<>();
        list.add(new Node((char)0,0));
        for(int i = 0; i < text.length(); i++){
            char letter = text.charAt(i);
            int num = pos(list, letter);
            if (num!= 0){

list.get(num).setHowMany(list.get(num).getHowMany()+1);
            }
            else {
                list.add(new Node(letter, 1));
            }
        }
        list.remove(0);
        Collections.sort(list);
        return list;
    }
```

-Codificar Cadena:

```
    public static String encode(ArrayList<Node> doIt, String
text){
        doIt(text);
        String ret = "";
        for (int i = 0; i < text.length();i++) {
```

```
            ret             +=              doIt.get(posn(doIt,
text.charAt(i))).getHuffcode();
        }
        return ret;
    }
```

-Poner Código Huffman a cada Nodo:

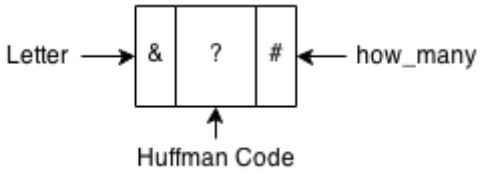
```
    public  static  Node  evaluateNode(Node  nodo,  String
huffcode){
        nodo.setHuffcode(huffcode);
        if(nodo.getLeft()!=null){
            Node    left    =    evaluateNode(nodo.getLeft(),
nodo.getHuffcode()+"0");
            nodo.setLeft(left);
        }
        if(nodo.getRight()!=null){
            Node    right    =    evaluateNode(nodo.getRight(),
nodo.getHuffcode()+"1");
            nodo.setRight(right);
        }
        return nodo;
    }
```

4. **Pruebas:**

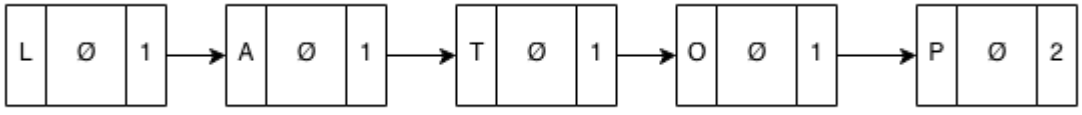
Para nuestra prueba dibujaremos “manualmente” el árbol y el código que debe generar y lo compararemos con el que genera el programa, para estre ejemplo codificaremos la palabra “LAPTOP”:

-Cadena:

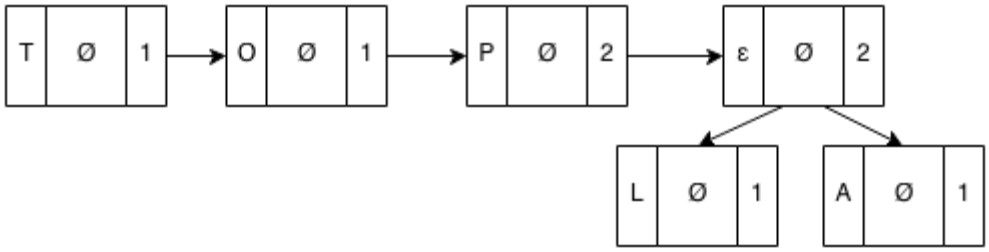
-Estructura básica de los nodos:



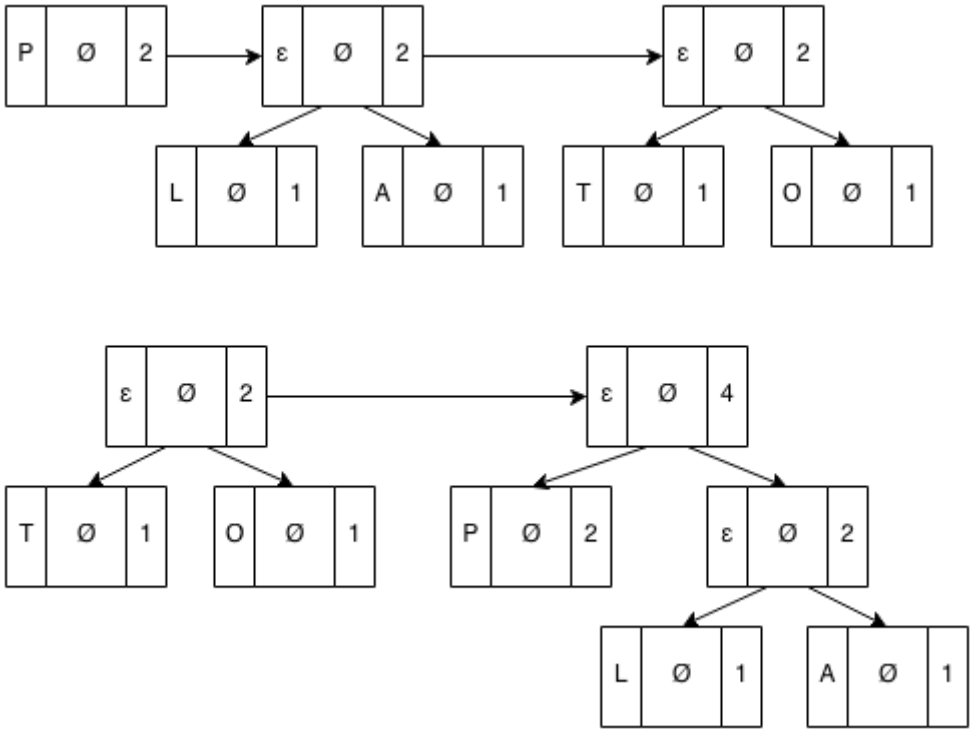
-Lista ordenada según las veces que aparece un carácter con “carácter”, Código Huffman nulo y el número de veces que aparece:

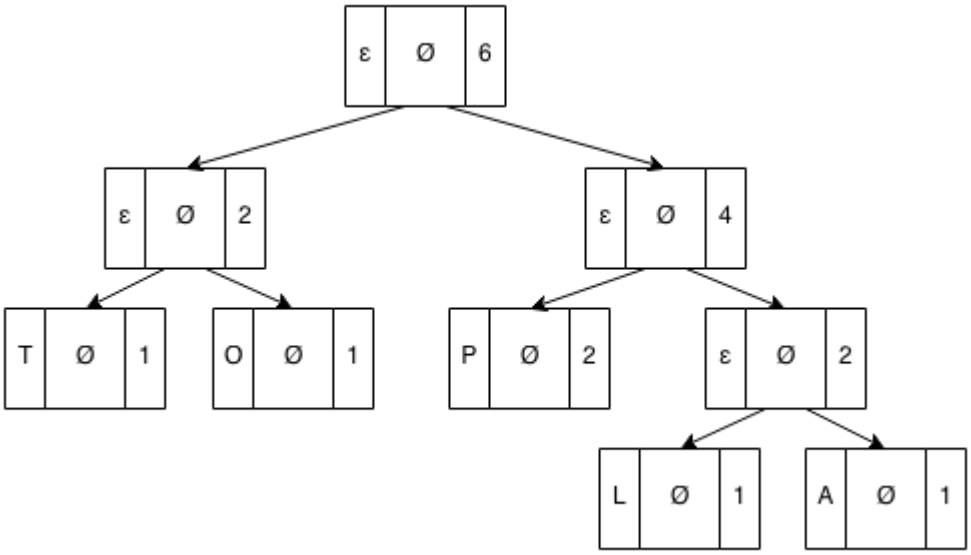


-Creamos un nuevo nodo con un carácter vacío (representado con la letra épsilon), Código Huffman nulo además ponemos como hijo izquierdo al primer Nodo de la lista y como hijo derecho el segundo Nodo de la lista y por último el “número de veces” en el nuevo nodo será la suma del número de veces de sus dos hijos. Sacamos los hijos de la lista, añadimos el nuevo nodo a la lista y la ordenamos según el “número de veces”:

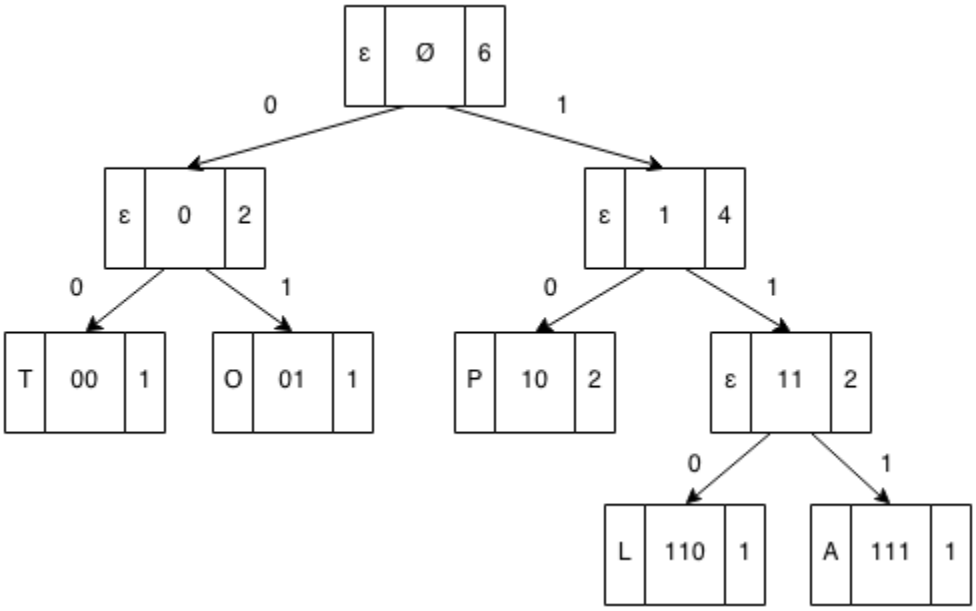


-Repetimos el proceso hasta tener solo un Nodo en la lista, el cual será la raíz de nuestro árbol:

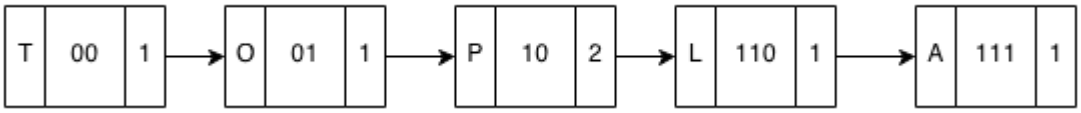




-Asignamos el Código Huffman a cada nodo del árbol, dejando el nodo raíz vacío y asignando a los demás el código Huffman de su padre más un “0” si son el hijo izquierdo o un “1” si son el hijo derecho.



-Luego creamos una lista con los nodos con un carácter diferente al vacío y la usaremos para codificar el texto:

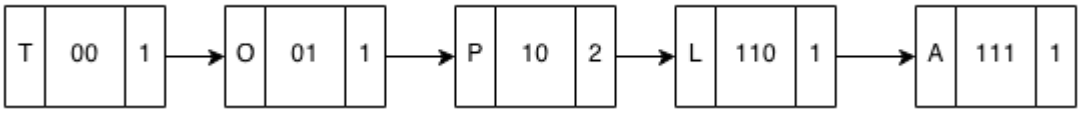


-Recorremos carácter por carácter en el texto y lo cambiamos por su Código Huffman:
L-A-P-T-O-P
110-111-10-00-01-10

Resultado: **11011110000110**

-Pruebas “Manuales” vs Implementación del Programa:

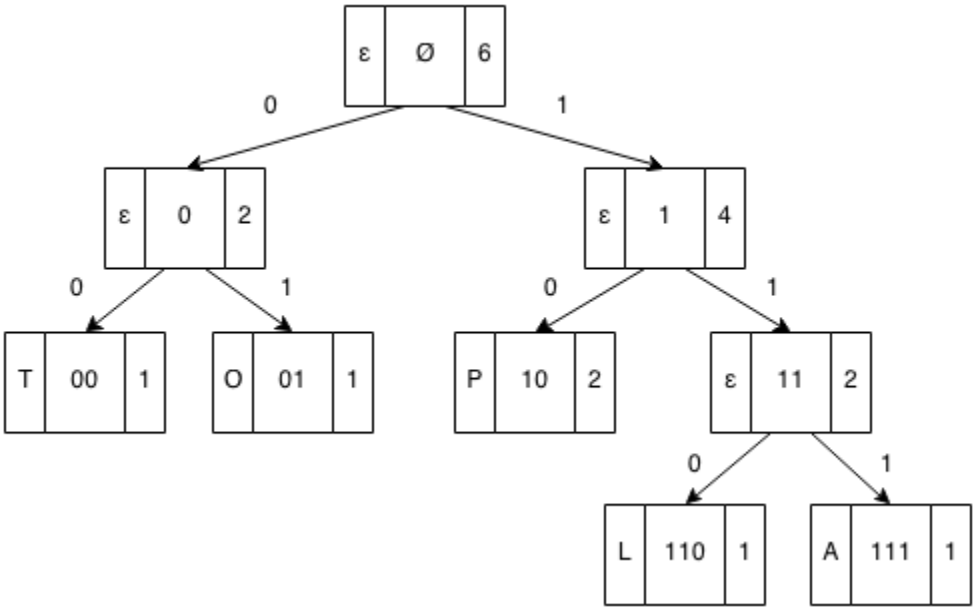
•Tabla Manual:



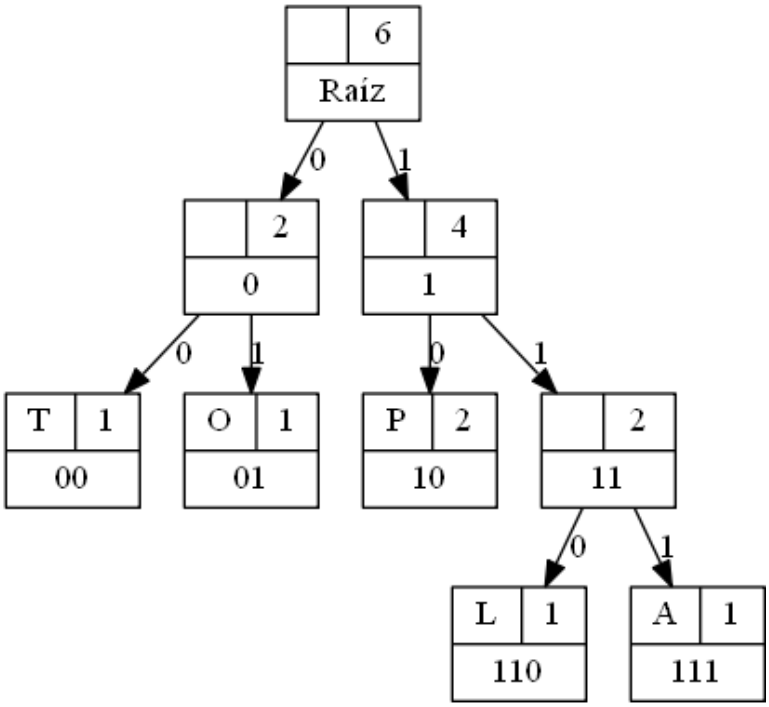
•Tabla Implementación:

Equivalencias		
Simbolo	Repetido	Cifrado
T	1	00
O	1	01
P	2	10
L	1	110
A	1	111

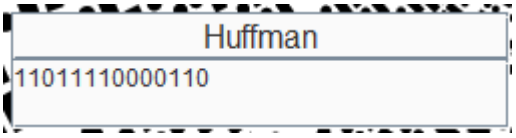
•Árbol Manual:



•Árbol Implementación:



•Códigos Huffman (Implementación vs Manual):



vs **11011110000110**

5. **Código**:

   Los códigos fuentes, imágenes y demás archivos se encuentran en la carpeta /Huffman/src/ en conjunto todos son un proyecto del IDE "NetBeans" en su versión 7.3 y las clases se encuentran dentro del paquete co.edu.eafit.huffman

6. **Ejecutable**:

   El archivo ejecutable "Huffman.jar" (≈360KB) se encuentra ubicado en /Huffman/, Si se desea generar arboles es necesario poner la carpeta "bin" que se entre en el CD al lado de la carpeta "Huffman" puesto que esta contiene el compilador de imágenes del árbol y es necesario añadir que para usar la opción puesto que se usa el programa "GraphViz" de generación de gráficos para este cometido y que se vea correctamente en su computador.

7. **Conclusiones:**

   1) Éste algoritmo toma un alfabeto de n símbolos, junto con sus frecuencias de aparición asociadas, y construye un árbol binario cuyos hijos izquierdos de nodos se les asocia un 0 y los hijos izquierdos se les asocia un 1, los cuales al recorrer el árbol le proporcionan el código huffman a cada símbolo.

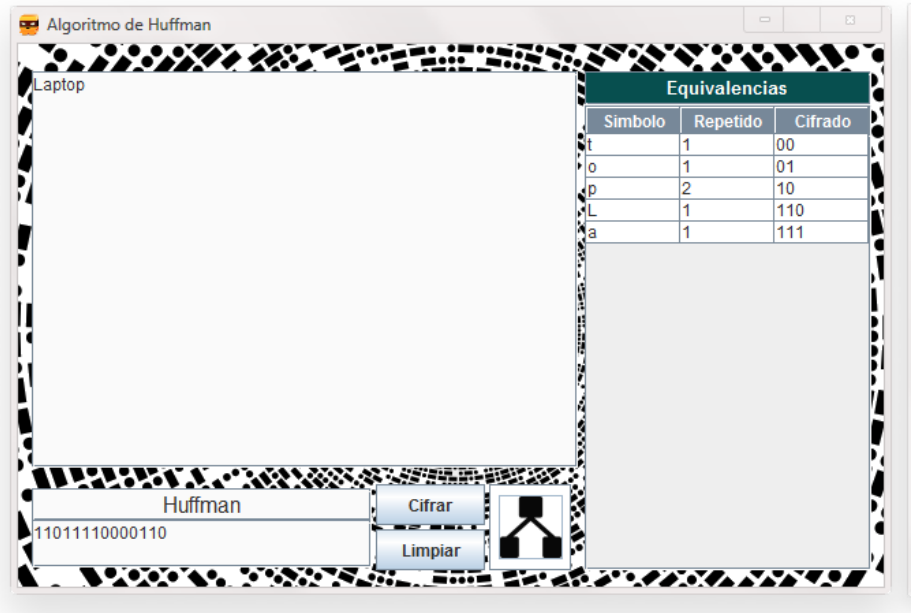
   2) Para codificar un número dado de símbolos es necesario que los símbolos suministrados sean mínimo 2 de ellos diferentes.

3) En caso de que se quiera decodificar una expresión, se debe proporcionar la tabla de equivalencias codificadas de cada símbolo.

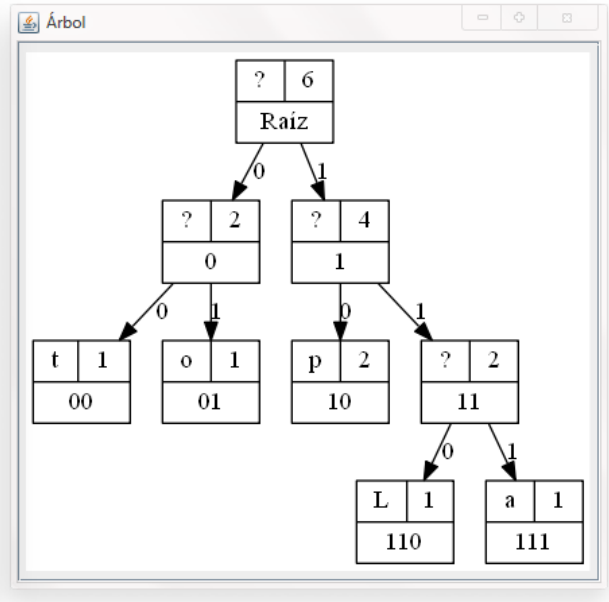
4) Es un algoritmo interesante ya que es muy útil a la hora de proteger la información que queramos, y además de que utiliza árboles B para realizar su funcionamiento.

8. Interfaz Gráfica:

Ventana:



Arbol:



9. JavaDoc:

Se encuentra en la carpeta: "/Huffman/dist/javadoc" Todas las clases y métodos se encuentran debidamente documentadas.