25 | Aplicación de varias estructuras: Codificación de Huffman

Meta

Implementar un algoritmo en forma eficiente utilizando las técnicas y estructuras de datos vistas a lo largo del curso.

Objetivos

Al finalizar la práctica el alumno será capaz de:

- Determinar la estructura apropiada para cada etapa de un algoritmo.
- Comprimir archivos de texto.

Antecedentes

En 1951, David Huffman, tratando de evitar un examen final del curso de *Teoría de la Información* recibió como tarea encontrar el *código binario más eficiente* para transmitir secuencias de símbolos de un conjunto, sabiendo la frecuencias con que aparece cada símbolo. [1]

Especificación del problema de codificación

Entradas

- El alfabeto de símbolos $A = \{a_1, a_2, ..., a_n\}$.

[1] https://es.wikipedia.org/wiki/Codificaci%C3%B3n_Huffman

- El conjunto $W = \{w_1, w_2, ..., w_n\}$ de pesos, proporcionales a la probabilidad de que aparezca el símbolo correspondiente, es decir, $w_i = peso(a_i), 1 \leqslant i \leqslant n$.

Salidas

- El código $C(A, W) = \{c_1, c_2, ..., c_n\}$, donde cada c_i es la palabra del código binario el símbolo $a_i, 1 \leqslant i \leqslant n$.

Algoritmo

1. Crear un nodo hoja para cada símbolo, asociandole un peso según su frecuencia de aparición e insertarlo en una estructura tal que pueda devolver los nodos uno por uno, según su frecuencia de menor a mayor.
2. Mientras haya más de un nodo pendiente:
   a) Extraer los dos nodos con menor probabilidad.
   b) Crear un nuevo nodo interno que tome a los anteriores como hijos izquierdo y derecho, de menor a mayor; su peso será la suma de los pesos de sus hijos.
   c) Insertar el nodo nuevo en la estructura, en el lugar que le corresponda según su peso.
3. El nodo que quede es el nodo raíz del árbol de codificación.

Codificación

Para determinar los bits que representarán a cada símbolo, se etiquetan las aristas a lo largo del camino desde la raíz hasta el caracter de interés. Cuando el arista apunta al hijo izquierdo se le asocia el cero, cuando apunta al hijo derecho se le asocia el uno.

Ejemplo

Sea la frase:

Vísteme despacio, que tengo prisa.

Al contar cuántas veces aparece cada caracter se obtiene la siguiente tabla de frecuencias en Tabla 25.1. Siguiendo el algoritmo, esta tabla de frecuencias genera el árbol de la Figura 25.1.

Tabla 25.1 Número de veces que aparece cada caracter en el texto a codificar. Se agregó un caracter al que llamamos EOF para indicar el final del archivo, en la práctica lo codificamos con `Character.MIN_VALUE`.

char	Frec.	char	Frec.	char	Frec.	char	Frec.
" "	4	c	1	m	1	r	1
,	1	d	1	n	1	s	3
.	1	e	5	o	2	t	2
V	1	g	1	p	2	u	1
a	2	i	2	q	1	í	1
EOF	1						

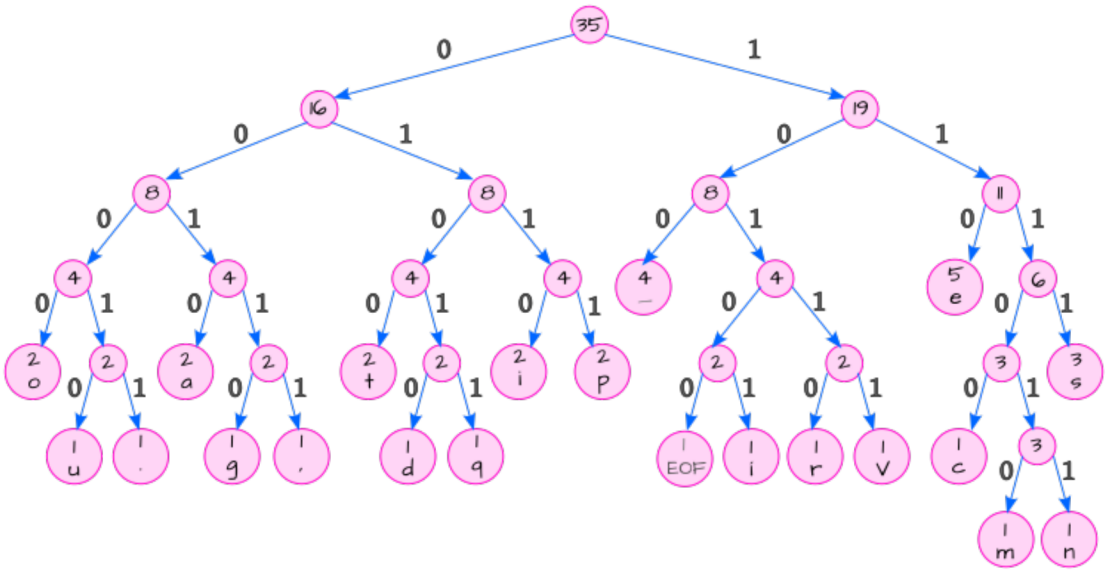

Figura 25.1 Árbol de codificación de Huffman generado con la tabla de frecuencias del ejemplo.

Aunque esto no es particulamente eficiente, representemos de momento el contenido del árbol en formato XML. Esto nos permitirá almacenar y recuperar el árbol en un archivo de texto. Claramente este no es un formato comprimido, pero en esta práctica nos concentraremos sólo en comprimir el texto de interés. Un fragmento del código que representaría al árbol de la Figura 25.1 se incluye a continuación:

```
<nodo peso="35">
 <nodo peso="16">
  <nodo peso="8">
   <nodo peso="4">
    <nodo peso="2">o</nodo>
    <nodo peso="2">
     <nodo peso="1">u</nodo>
     <nodo peso="1">.</nodo>
    </nodo>
   </nodo>
   <nodo peso="4">
   ...
   </nodo>
  </nodo>
  <nodo peso="8">
   <nodo peso="4">
    ...
   </nodo>
   <nodo peso="4">
    ...
   </nodo>
  </nodo>
 </nodo>
 <nodo peso="19">
  <nodo peso="8">
   ...
  </nodo>
  <nodo peso="11">
   ...
  </nodo>
 </nodo>
</nodo>
```

Observa como cada nodo se representa con una etiqueta que contiene dentro de sí la frecuencia que le corresponde como un atributo; si es un nodo interno entre las etiquetas que abren y cierran están las etiquetas de sus nodos hijos, mientras que si se trata de una hoja está el símbolo que representa.

Para guardar entonces el texto en un archivo comprimido se realiza la escritura bit por bit. En un archivo de tipo binario se codifica la secuencia de caracteres utilizando los bits que van de la raíz hasta el caracter de interés. Por ejemplo: al leer la letra *V*, se escriben los bits `10101` pues la ruta desde la raíz es:

$$\text{derecha} \rightarrow \text{izquierda} \rightarrow \text{derecha} \rightarrow \text{izquierda} \rightarrow \text{derecha}$$

al leer la letra *í*, se escriben los bits `11000`. Obsérvese que las letras más frecuentes, como la *e*, tienen códigos más cortos: `111`. De este modo, la cantidad de bits utilizados para escribir el texto completo será mínimo.

Desarrollo

Utiliza la frase siguiente para probar el código final:

> VÍSTEME DESPACIO QUE TENGO PRISA: Con esta frase encarecemos a otro a que proceda con sosiego al realizar algo complicado porque el apresuramiento, lejos de abreviar, suele entorpecer e incluso malograr los mejores propósitos.[2]

Escríbela dentro del archivo de texto que utilizarás para tus pruebas.

1. Crea un `Makefile` que te permita compilar y ejecutar tu código. No olvides utilizar la estructura de archivos donde los archivos fuente van dentro de la carpeta `src` y los archivos compilados se guardan dentro de la carpeta `build`.

2. Programa la clase `CodificadorHuffman`, dentro del paquete:

   `ed.aplicaciones.huffman`,

   que en su constructor reciba una cadena de texto y la utilice para generar el árbol de codificación.

   Programa las clases auxiliares `Árbol` y `Nodo` para representar y construir el árbol codificador, poniéndolas dentro del mismo paquete. `CodificadorHuffman` los utilizará para realizar su trabajo y deberá tener un método de lectura que devuelva una referencia a este árbol.

3. `CodificadorHuffman` deberá tener un método `codifica` que reciba como parámetro el nombre de un archivo sin terminación, le agregarás la terminación `.huff`.

[2] https://www.cervantesvirtual.com/obra-visor/dichos-populares-su-significado/html/

Utiliza la clase auxiliar que se te entrega `ed.io.BitBufferedOutputStream` para que escriba el texto codificado en el archivo indicado. Este flujo es un filtro, para enviar bits al archivo necesitarás usar un `FileOutputStream`, revisa la documentación correspondiente.

Si requieres métodos auxiliares que te ayuden con tu trabajo o clases extra para administrar mejor tus datos puedes agregarlas.

TIP: Como los caracteres se encuentran en la base del árbol, no será fácil encontrar el código dado un caracter. Elige una estructura auxilar para almacenar los ćodigos dado su caracter, recorre el árbol recolectando todos los códigos y almacenándolos en esta estructura. Considera: ¿qué tipo de recorrido necesitas? ¿cómo es más fácil implementarlo? ¿qué estructura auxiliar es la más adecuada?

TIP: Como la cadena codificada no necesariamente tendrá una longitud múltiplo de ocho, y los archivos no pueden tener menos que un byte, seguramente al final serán agregados ceros que no tienen significado. Para detectar correctamente el final del archivo, se te sugiere agregar un símbolo a la tabla de frecuencias, cuya frecuencia sea uno y que represente el final del archivo [Figura 25.1]; añadirás el código que el algoritmo asigne a este símbolo al final de la cadena codificada; así, cuando quieras recuperar el texto, sabrás que has terminado cuando reconozcas este código y podrás ignorar los ceros que vengan después.

4. Programa una clase de uso llamada `Codifica` dentro del paquete:

   `ed.aplicaciones.huffman`.

   Cuando sea invocada pasándole como parámetro la dirección de un archivo desde el que deberá leer el contenido, crear un objeto tipo `CodificadorHuffman` y ordenarle que codifique el texto. Como resultado se deberán producir dos archivos: uno con el texto codificado, al que guardarás con terminación `.huff` y otro con el árbol de codificación en formato `XML` y terminación `.xml`, utiliza la raíz del nombre del archivo original para generar estos otros.

   Recuerda que puedes poner métodos dentro de esta clase para que realicen la lectura y escritura en archivos, de modo que no quede toda la lógica aglomerada en el método `main`.

5. Programa ahora una clase `DecodificadorHuffman`, que en su constructor reciba un arreglo de bytes y un árbol codificador. Tendrá un método `decodifica` que devolverá una cadena con el texto decodificado.

6. Agrega otra clase de uso llamada `Decodifica`, que reciba como argumentos los archivos donde se encuentran el texto codificado, el árbol en formato xml y dónde se desea guardar el texto decodificado; deberá leer la información, reconstruir el árbol, utilizar esto para invocar a `DecodificadorHuffman` y crear el archivo con el texto decodificado.

7. Edita el archivo `Readme.md` para explicar cómo ejecutar tu código.

TIP: Observa que puedes programar `CodificadorHuffman` y `DecodificadorHuffman` sin realizar manipulaciones de archivos. Las clases de uso pueden comenzar teniendo la información en variables locales. Una vez que ya funcione tu algoritmo de Huffman comienza a trabajar en leer y escribir archivos.

Preguntas

1. ¿Qué estructura de datos elegiste para generar la tabla de frecuencias? ¿Por qué es la adecuada?
2. ¿Qué estructura te permite ordenar los nodos según su frecuencia?