Informe de Laboratorio 08

Tema: Algoritmo de Compresión de Huffman

Nota

Estudiante	Escuela	Asignatura
Victor Gonzalo Maldonado Vilca, Armando Steven Cuno Cahuari vmaldonadov@unsa.edu.pe, acunoc@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Estructura de Datos y Algoritmos Semestre: III Código: 1702122

Tarea	Tema	Duración
08	Algoritmo de Compresión de Huffman	2 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	Del 11/07/2024 – 00:00am	Al 11/07/2024 – 23:59pm

1. Introducción

1. Algoritmo de compresión de Huffman

   - Se trata de un algoritmo que puede ser usado para compresión o encriptación de datos. Este algoritmo se basa en asignar códigos de distinta longitud de bits a cada uno de los caracteres de un fichero.
   - Si se asignan códigos más cortos a los caracteres que aparecen más a menudo se consigue una compresión del fichero.
   - Esta compresión es mayor cuando la variedad de caracteres diferentes que aparecen es menor.

   Por ejemplo: si el texto se compone únicamente de números o mayúsculas, se conseguirá una compresión mayor. Para recuperar el fichero original es necesario conocer el código asignado a cada caracter, así como su longitud en bits, si ésta información se omite, y el receptor del fichero la conoce, podrá recuperar la información original. De este modo es posible utilizar el algoritmo para encriptar ficheros.

2. Mecanismo del algoritmo

   - Contar cuantas veces aparece cada caracter en el fichero a comprimir. Y crear una lista enlazada con la información de caracteres y frecuencias.
   - Ordenar la lista de menor a mayor en función de la frecuencia.

- Convertir cada elemento de la lista en un árbol.
- Fusionar todos estos árboles en uno único, para hacerlo se sigue el siguiente proceso, mientras la lista de árboles contenga más de un elemento:
- ——-• Con los dos primeros árboles formar un nuevo árbol, cada uno de los árboles originales en una rama.
- ——-• Sumar las frecuencias de cada rama en el nuevo elemento árbol.
- ——-• Insertar el nuevo árbol en el lugar adecuado de la lista según la suma de frecuencias obtenida.
- Para asignar el nuevo código binario de cada caracter sólo hay que seguir el camino adecuado através del árbol.
- Si se toma una rama cero, se añade un cero al código, si se toma una rama uno, se añade un uno.
- Se recodifica el fichero según los nuevos códigos.

3. Veamos un ejemplo **Tomemos un texto corto, por ejemplo: *.ªta la jaca a la estaca"***

   a) Contamos las veces que aparece cada caracter y hacemos una lista enlazada: ' '(5), a(9), c(2), e(1), j(1), l(2), s(1), t(2).

   b) Ordenamos por frecuencia de menor a mayor: e(1), j(1), s(1), c(2), l(2), t(2), ' '(5), a(9).

   c) Consideremos ahora que cada elemento es el nodo raíz de un árbol.

   d) Fusionamos los dos primeros nodos (árboles) en un nuevo árbol, sumamos sus frecuencias y lo colocamos en el lugar correspondiente:

   e) Asignamos los códigos, las ramas a la izquierda son ceros, y a la derecha unos, es una regla arbitraria.

2. Objetivos

- Comprender y aplicar el algoritmo de compresión de Huffman.
- Implementar el algoritmo en un lenguaje de programación.
- Analizar la eficiencia del algoritmo.
- Documentar el proceso de implementación.

3. Tarea

Elabore un informe paso a paso de la implementación del algoritmo de compresión mediante el algoritmo de Huffman.

4. Entregables

- Informe hecho en Latex.
- URL: Respositorio GitHub.

5. Equipos, materiales y temas utilizados

- Teoría de la Compresión de Datos.
- Algoritmo de Huffman.
- Estructuras de Datos.
- Java.

6. URL de Repositorio Github

- Link de Repositorio GitHub.
- `https://github.com/Victor-Gonzalo-Maldonado-Vilca/EDA_Lab08.git`

7. Desarrollo del trabajo

El flujo de trabajo gracias al algoritmo de Huffman permite minimizar el espacio ocupado por documentos, entre otros. Para ello, es necesario hacer un recuento de la frecuencia de cada carácter. Luego, debemos ordenar estos caracteres de menor a mayor frecuencia para construir el árbol Heap. A continuación, mediante una iteración de ese ordenamiento, construiremos el árbol uniendo dos nodos a la vez: el hijo izquierdo con un '0' y el derecho con un '1'.

7.1. Clase Node

- **Descripción:** Definir la clase 'Node' es necesario para construir el árbol de Huffman utilizando una cola de prioridad o una lista enlazada. Los valores se organizan en la cola de prioridad y se usarán para construir el árbol de Huffman.
  En esta clase, tendremos como atributos privados el carácter, la frecuencia y dos nodos hijos, izquierdo y derecho. La clase tendrá dos constructores: uno que recibe como parámetros la frecuencia y el carácter, y otro que además incluye los valores de los hijos izquierdo y derecho.
- **Código:**

```java
public class Node {
    char ch;
    int freq;
    Node left = null, right = null;

    Node(char ch, int freq) {
        this.ch = ch;
        this.freq = freq;
    }

    Node(char ch, int freq, Node left, Node right) {
        this.ch = ch;
        this.freq = freq;
        this.left = left;
        this.right = right;
    }
}
```

7.2. Clase HuffmanComparator

- **Descripción:** Esta clase se utilizará para ordenar la cola de prioridad basada en la frecuencia de los caracteres. Tendrá un método de clase llamado 'compare', que tomará como argumento dos nodos. Este método comparará las frecuencias de los dos nodos y decidirá su orden. Si 'a.freq < b.freq', entonces 'a' debe venir antes que 'b'. Si 'a.freq > b.freq', entonces 'a' debe venir después que 'b'. Finalmente, si ambas frecuencias son iguales, no habrá ningún cambio.
- **Código:**

```
1  import java.util.Comparator;
2
3  public class HuffmanComparator implements Comparator<Node> {
4      @Override
5      public int compare(Node a, Node b) {
6          return a.freq - b.freq;
7      }
8  }
```

7.3. Clase HuffmanCoding

- **Descripción:** Esta clase estará encargada de realizar todo el trabajo relacionado con el árbol de Huffman.
- **Código:**

```
1  import java.util.PriorityQueue;
2  import java.util.Map;
3  import java.util.HashMap;
4
5  public class HuffmanCoding {
```

7.3.1. Método encode

- **Descripción:** Método encargado de recorrer el árbol de Huffman y almacenar los códigos de Huffman en un mapa. Este método recibirá como parámetros el nodo raíz del árbol, la palabra a codificar y un mapa que almacenará los códigos. Primero, comprobará si el nodo raíz es nulo. Si el nodo actual no tiene hijos, se almacenará en el mapa el carácter con el código binario acumulado. Finalmente, se realizarán dos llamadas recursivas al mismo método, variando el nodo raíz: una para el hijo izquierdo y otra para el hijo derecho. Este proceso asegurará que cada carácter tenga un código único con respecto a su posición en el árbol de Huffman.
- **Código:**

```
6      public static void encode(Node root, String str, Map<Character, String>
           huffmanCode) {
7          if (root == null) {
8              return;
9          }
10
11         if (root.left == null && root.right == null) {
12             huffmanCode.put(root.ch, str);
13         }
14
```

```
15          encode(root.left, str + "0", huffmanCode);
16          encode(root.right, str + "1", huffmanCode);
17      }
```

7.3.2. Método decode

- **Descripción:** Método encargado de recorrer el árbol de Huffman y decodificar el string.

  1. Primero, comprobará si el nodo raíz es nulo. Si es así, terminará la decodificación.
  2. Luego, verificará si el nodo actual no tiene hijos. En ese caso, imprimirá el carácter y retornará el índice actual.
  3. Después, avanzará al siguiente bit en la cadena codificada y realizará una llamada recursiva según el bit en la posición actual del índice: si es 0, se moverá al subárbol izquierdo; si es 1, se moverá al subárbol derecho.
  4. Finalmente, retornará el índice actual después de decodificar el carácter actual.

  Este proceso permite reconstruir la cadena original a partir de su representación binaria codificada usando el árbol de Huffman.

- **Código:**

```
19      public static int decode(Node root, int index, StringBuilder sb) {
20          if (root == null) {
21              return index;
22          }
23
24          if (root.left == null && root.right == null) {
25              System.out.print(root.ch);
26              return index;
27          }
28
29          index++;
30
31          if (sb.charAt(index) == '0') {
32              index = decode(root.left, index, sb);
33          } else {
34              index = decode(root.right, index, sb);
35          }
36
37          return index;
38      }
```

7.3.3. Método buildHuffmanTree

- **Descripción:** Método encargado en la construcción del árbol Heap.
  Este método consiste en la compresión y descompresión de un texto. Para ello será necesario primero construir el árbol.
  Primero, contaremos la frecuencia de cada carácter en una palabra utilizando un mapa que mantenga esta información. Luego, ordenamos estos valores utilizando una cola de prioridad que almacenará todos los nodos con sus respectivos caracteres y frecuencias. Utilizaremos una clase 'HuffmanComparator' para ordenar los nodos por frecuencia mínima.
  Después, crearemos los nodos del árbol de Huffman. Comenzaremos combinando los dos nodos con menor frecuencia, sumando sus frecuencias para crear un nuevo nodo interno, hasta que

quede un solo nodo que será la raíz del árbol.
Finalmente, implementaremos métodos para codificar y decodificar utilizando el nodo raíz del árbol de Huffman. En el método de codificación, utilizaremos un mapa para almacenar los códigos correspondientes a cada carácter. En el método de decodificación, construiremos una cadena a partir de los códigos Huffman y luego decodificaremos esa cadena utilizando el árbol de Huffman.

- **Código:**

```
41    public static void buildHuffmanTree(String text) {
42        // Contar la frecuencia de aparicion de cada caracter y almacenarlo en un mapa
43        Map<Character, Integer> freq = new HashMap<>();
44        for (char c : text.toCharArray()) {
45            freq.put(c, freq.getOrDefault(c, 0) + 1);
46        }
47
48        // Crear una cola de prioridad para almacenar nodos del arbol de Huffman
49        PriorityQueue<Node> pq = new PriorityQueue<>(new HuffmanComparator());
50
51        // Crear un nodo hoja para cada caracter y anadirlo a la cola de prioridad
52        for (var entry : freq.entrySet()) {
53            pq.add(new Node(entry.getKey(), entry.getValue()));
54        }
55
56        // Hacer hasta que haya mas de un nodo en la cola
57        while (pq.size() != 1) {
58            // Eliminar los dos nodos de mayor prioridad (frecuencia minima)
59            Node left = pq.poll();
60            Node right = pq.poll();
61
62            // Crear un nuevo nodo interno con estos dos nodos como hijos y con una
                  frecuencia
63            // igual a la suma de las frecuencias de los dos nodos. Anadir el nuevo nodo
                  a la cola.
64            int sum = left.freq + right.freq;
65            pq.add(new Node('\0', sum, left, right));
66        }
67
68        // Root stores pointer to root of Huffman Tree
69        Node root = pq.peek();
70
71        // Recorrer el arbol de Huffman y almacenar los codigos de Huffman en un mapa
72        Map<Character, String> huffmanCode = new HashMap<>();
73        encode(root, "", huffmanCode);
74
75        // Print encoded string
76        System.out.println("Codigos de Huffman: " + huffmanCode);
77        System.out.println("Texto original: " + text);
78
79        StringBuilder sb = new StringBuilder();
80        for (char c : text.toCharArray()) {
81            sb.append(huffmanCode.get(c));
82        }
83
84        System.out.println("Texto codificado: " + sb);
85
86        // Decodificar el string binario
```

```
87          int index = -1;
88          System.out.println("Texto decodificado: ");
89          while (index < sb.length() - 1) {
90              index = decode(root, index, sb);
91          }
92      }
```

7.4. Clase Main

- **Descripción:** Ejemplo básico de como funciona el código Huffman mediante la palabra "Codigo Huffman aplicando compresion y descompresion"
- **Código:**

```
41  public class Main {
42      public static void main(String[] args) {
43          String text = "Codigo Huffman aplicando compresion y descompresion";
44          HuffmanCoding.buildHuffmanTree(text);
45      }
46  }
```

8. Conclusiones

El Algoritmo de Compresión de Huffman:

- Asigna códigos más cortos a caracteres frecuentes, logrando una reducción significativa del tamaño del archivo.
- Proporciona una codificación óptima para datos con frecuencias de caracteres conocidas, sin probabilidades dependientes.
- Fácil de implementar y robusto, usando estructuras de datos como listas enlazadas y árboles binarios.
- Útil para diversos tipos de datos y contextos, especialmente en sistemas de archivos y transmisión de datos.
- Menos efectivo cuando la diversidad de caracteres es alta y las frecuencias son uniformes. La tabla de códigos puede aumentar ligeramente el tamaño total del archivo.

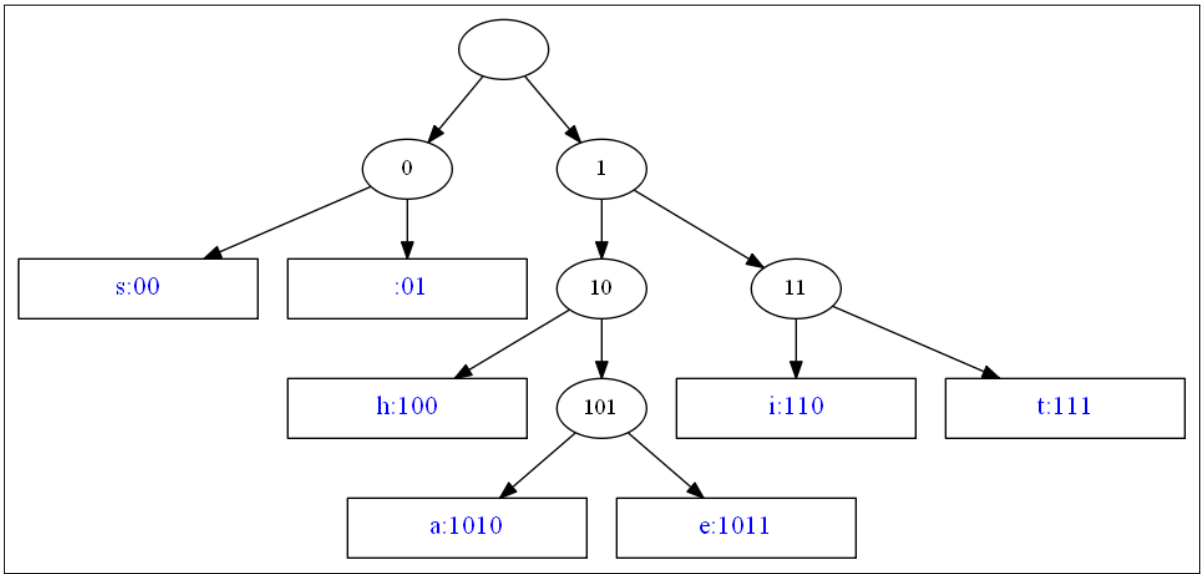

Figura 1: Ejemplo 1

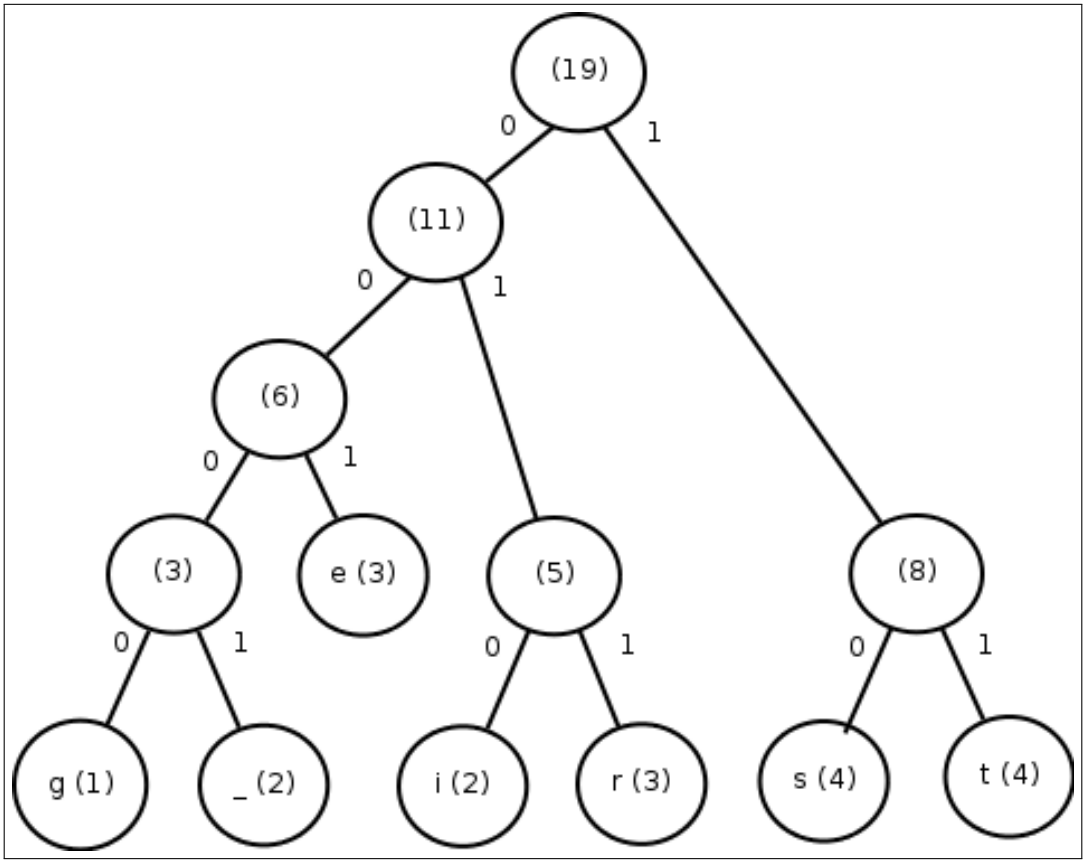

Figura 2: Ejemplo 2

8.1. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 1: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 2: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	4	
Total		20		20	

9. Referencias

- https://www.w3schools.com/java/default.asp
- https://www.java.com/es/download/ie_manual.jsp
- https://conclase.net/blog/item/huffman