INFORME DE TRABAJO PRÁCTICO

INFORMACIÓN BÁSICA					
ASIGNATURA:	*Laboratorio EDA*				
TITULO DEL TRABAJO:	*Lab 08*				
NÚMERO DE TRABAJO:	*8*	**AÑO LECTIVO:**	*2*	**NRO. SEMESTRE:**	*3*
FECHA DE PRESENTACIÓN	*11 de julio*	**HORA DE PRESENTACIÓN**	*23:55*		
INTEGRANTE (s) *Quispe Huaman, Rodrigo Ferdinand*				**NOTA (0-20)**	
DOCENTE(s): *Edson Luque*					

INTRODUCCIÓN

Implementar el algoritmo de Huffman:

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.

Tarea:

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

MARCO CONCEPTUAL

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.

SOLUCIONES Y PRUEBAS

Clase HuffmanNode<T>:

- Representa un nodo en el árbol de Huffman. Cada nodo puede ser una hoja (contiene un carácter) o un nodo interno (contiene la suma de frecuencias de sus hijos).

Java

```java
public class HuffmanNode<T> implements Comparable<HuffmanNode<T>> {
    T character;
    int frequency;
    HuffmanNode<T> left;
    HuffmanNode<T> right;

    public HuffmanNode(T character, int frequency) {
        this.character = character;
        this.frequency = frequency;
        this.left = null;
        this.right = null;
    }

    @Override
    public int compareTo(HuffmanNode<T> node) {
        return this.frequency - node.frequency;
    }
}
```

Clase HuffmanTree<T>:

- Construye el árbol de Huffman a partir de un mapa de frecuencias y genera los códigos binarios para cada carácter.

- **Método constructor:**
  - Constructor que crea el árbol de Huffman utilizando una cola de prioridad para fusionar los nodos.
  - Crea un nodo hoja para cada carácter con su frecuencia, añade todos los nodos a una cola de prioridad y fusiona los dos nodos con menor frecuencia hasta obtener un solo árbol.

```java
public HuffmanTree(Map<T, Integer> frequencies) {
    PriorityQueue<HuffmanNode<T>> priorityQueue = new PriorityQueue<>();

    for (Map.Entry<T, Integer> entry : frequencies.entrySet()) {
        priorityQueue.add(new HuffmanNode<>(entry.getKey(), entry.getValue()));
    }

    while (priorityQueue.size() > 1) {
        HuffmanNode<T> left = priorityQueue.poll();
        HuffmanNode<T> right = priorityQueue.poll();
        HuffmanNode<T> newNode = new HuffmanNode<>(null, left.frequency +
right.frequency);
        newNode.left = left;
        newNode.right = right;
        priorityQueue.add(newNode);
    }

    this.root = priorityQueue.poll();
}
```

- **Método public Map<T, String> generateCodes():**
  - Genera un mapa con los códigos binarios para cada carácter.
  - Llama al método recursivo generateCodes pasando la raíz del árbol, un string vacío y un mapa vacío. Este método recursivo recorre el árbol y asigna un código binario a cada carácter.

```java
public Map<T, String> generateCodes() {
    Map<T, String> codes = new HashMap<>();
    generateCodes(this.root, "", codes);
    return codes;
```

```
}
```

- **Método private void generateCodes(HuffmanNode<T> node, String code, Map<T, String> codes):**
  - Método recursivo que asigna códigos binarios a los caracteres recorriendo el árbol.
  - Si el nodo no es nulo y es una hoja, añade el carácter y su código al mapa. Si no, llama recursivamente a la izquierda añadiendo '0' al código y a la derecha añadiendo '1'.

```java
private void generateCodes(HuffmanNode<T> node, String code, Map<T, String> codes) {
    if (node != null) {
        if (node.character != null) {
            codes.put(node.character, code);
        }
        generateCodes(node.left, code + "0", codes);
        generateCodes(node.right, code + "1", codes);
    }
}
```

- **Método public HuffmanNode<T> getRoot():**
  - Devuelve la raíz del árbol.
  - Método accesor para obtener la raíz del árbol.

```java
public HuffmanNode<T> getRoot() {
    return root;
}
```

Clase HuffmanCompressor<T>:

- Proporciona métodos para comprimir y descomprimir texto utilizando los códigos de Huffman generados.

- **Método public static <T> String compress(String text, Map<T, String> huffmanCodes):**
  - Comprime el texto utilizando los códigos de Huffman.
  - Recorre el texto original y reemplaza cada carácter por su código de Huffman correspondiente, concatenando los códigos en una cadena.

```java
public static <T> String compress(String text, Map<T, String> huffmanCodes) {
    StringBuilder compressed = new StringBuilder();
    for (char c : text.toCharArray()) {
        compressed.append(huffmanCodes.get((T) Character.valueOf(c)));
    }
    return compressed.toString();
}
```

- **Método public static <T> String decompress(String compressedText, HuffmanNode<T> root):**
  - Descomprime el texto utilizando el árbol de Huffman.
  - Recorre la cadena comprimida bit a bit, navegando por el árbol de Huffman. Cuando se alcanza una hoja, se añade el carácter correspondiente a la cadena descomprimida y se vuelve a la raíz del árbol.

```java
public static <T> String decompress(String compressedText, HuffmanNode<T> root) {
    StringBuilder decompressed = new StringBuilder();
    HuffmanNode<T> currentNode = root;
    for (char bit : compressedText.toCharArray()) {
        currentNode = bit == '0' ? currentNode.left : currentNode.right;
        if (currentNode.left == null && currentNode.right == null) {
            decompressed.append(currentNode.character);
            currentNode = root;
        }
    }
    return decompressed.toString();
}
```

LECCIONES APRENDIDAS Y CONCLUSIONES

Durante la implementación del algoritmo de Huffman en Java, he aprendido cómo las estructuras de datos como las colas de prioridad y los árboles binarios son esenciales para la compresión eficiente de datos. El proceso de compresión y descompresión me ha permitido ver cómo se pueden reducir los tamaños de los archivos sin pérdida de información al asignar códigos de distinta longitud según la frecuencia de aparición de los caracteres.

Descomponer el problema en subproblemas manejables, como contar frecuencias, construir el árbol, generar códigos, y comprimir y descomprimir texto, ha sido clave para una implementación exitosa. En conclusión, este proyecto me ha proporcionado una valiosa oportunidad para aplicar conceptos teóricos en una solución práctica, incrementando tanto mi comprensión del algoritmo de Huffman como mi habilidad para implementar técnicas de compresión en la programación en Java.

REFERENCIAS Y BIBLIOGRAFÍA

https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

https://docs.oracle.com/javase/8/docs/api/java/util/Set.html

https://docs.oracle.com/en/java/java-components/index.html