Trabajo Práctico Especial 2

Teoría de la Información

Codificación

Fecha de entrega: 02/06/2022
Grupo 3
Integrantes:

Raffin, Giuliana

Zárate, Francisca Pilar

Ejercicio a) Obtener para cada señal la distribución de probabilidades de los valores de la señal (considerada como una fuente sin memoria) y su codificación según el método de Huffman. Almacenarlos en un archivo.

Para este ejercicio, se nos pidió obtener la probabilidad de emisión para cada símbolo de las señales provistas y su codificación obtenida mediante el método de Huffman. Para la primera parte, se calculó la frecuencia de cada símbolo y, teniendo la cantidad de símbolos emitidos en cada señal, se obtuvo la distribución de probabilidades solicitada dividiendo para cada símbolo su frecuencia por la cantidad total de emisiones de cada señal por separado.

Luego de haber calculado la distribución de probabilidades se procedió a implementar el algoritmo de Huffman para obtener la codificación para cada símbolo de las señales. Este algoritmo busca reducir el tamaño de una señal al comprimir sus valores mediante la transformación de los mismos a un código binario basado en la probabilidad de emisión o frecuencia de cada uno de ellos. Se busca, de esta manera, que a cada valor se le asigne un código en particular que no pertenezca a otro de la misma fuente. Así, es posible la codificación y decodificación de ciertas señales sin perder datos debido a la ambigüedad. Para este ejercicio, el algoritmo de Huffman se implementó utilizando una cola de prioridad que ordena los valores comparando las probabilidades y así, obteniendo el mínimo cada vez que se accede a ella. Por otro lado y para su utilización en el ejercicio siguiente, se eligió almacenar la longitud de cada código, la cuál depende de la cantidad de pasos que realice el algoritmo para llegar desde cierto valor hasta la raíz.

El pseudocódigo planteado para la resolución del inciso se muestra a continuación.

```
Nodo {
       float data
       int c
       Nodo izq
       Nodo der
}

//comparador de los nodos basado en su valor para que el treeSet este ordenado de menor a
mayor, el primero que sale (con q.first) es el menor
Comparator<Nodo> {
       compare(Nodo x, Nodo y) {
       return Float.compare(x.data, y.data)
       }
}

Huffman {
       imprimirArbol(Nodo raiz, String s,HashMap<Integer,Integer> longitud){ //en s guardo el
código de 1s y 0s
       if (raiz.c != -1) {
          longitud.put(raiz.c, s.length())
          if (raiz.izq == null and raiz.der == null) {
              print(raiz.c + ":" + s + " longitud: " + s.length())
              return
          }
       }
       //izq 0, der 1 para el código
       imprimirArbol(raiz.izq, s + "0",longitud)
       imprimirArbol(raiz.der, s + "1",longitud)
}

main () {
       tam = #simbolos //tamaño
       fuente = {} //fuente
       prob = {} //distribución de probabilidades de la fuente
       HashMap<Integer, Integer> longitud //guardo la longitud de cada elemento
       PriorityQueue<Nodo> q(new MyComparator())
       for (i = 0 to #simbolos únicos) {
```

```
            Nodo hn //nodo del árbol de Huffman
            hn.c = fuente[i] //símbolo
            hn.data = prob[i] //probabilidad
            hn.izq = null
            hn.der = null
            q.add(hn)
      }
      Nodo raiz = null
      while (q.size() > 1){ //Busco el valor mínimo hasta que llegue a la raíz
            // primer mínimo
            Nodo x = q.peek()
            q.poll()
            //segundo mínimo
            Nodo y = q.peek()
            q.poll()
            assert y != null
            Nodo f
            f.data = x.data + y.data // sumo las probabilidades
            f.c = -1
            f.izq = x
            f.der = y
            raiz = f
            if (!q.contains(f))
                  q.add(f)
      }
      assert raiz != null
      imprimirArbol(raiz, "", longitud)
}
```

Ejercicio b) Obtener para cada señal la longitud total en bits de la señal codificada con Huffman y comparar respecto del tamaño del archivo original.

La longitud en bits de cada símbolo de cada señal codificada con Huffman se obtuvo con el algoritmo de Huffman implementado en el ejercicio anterior. Utilizando los códigos obtenidos y las longitudes, se multiplicó la frecuencia de emisión de cada símbolo con la longitud de código correspondiente para obtener la longitud total en bits de ambas señales. De igual manera se sumaron la cantidad de de dígitos de 1 byte que tiene cada símbolo de las dos señales. La fuente Beethoven original pesa 40000 bits, mientras que codificándola con Huffman se obtuvo un peso total de 9591 bits, por otra parte la fuente de L-Gante original pesa 28168 bits y su codificación con Huffman pesa 6703 bits. Como era de esperarse, en ambos casos la codificación de Huffman reduce en gran medida el tamaño total de las señales lo que supone una alta optimización al momento de almacenarlas o transmitirlas.

A continuación se muestra el pseudocódigo planteado para la resolución del inciso.

```
longitudTotalEnBits() {
    for ∀ símbolo ∈ señal
      l = l + símbolo.longitud*8; //multiplicado por 8 para pasar de bytes a bits
    return l;
}

longitudHuffmanEnBits() {
    for ∀ símbolo_S ∈ señal //para cada símbolo de la señal
      for ∀ símbolo_D ∈ distribución //para cada símbolo único de la señal
        if símbolo_S == símbolo_D
          l += longitud[símbolo]; //longitud obtenida aplicando Huffman
    return l;
}
```

Ejercicio c) Obtener para cada señal el rendimiento del código de Huffman obtenido en a) respecto del valor de la entropía de la señal. Explicar de qué manera se podría mejorar dicho rendimiento.

El rendimiento se obtuvo dividiendo la longitud media de cada señal por sus respectivas entropías. La longitud media se calculó como la sumatoria del producto entre la longitud del código de cada símbolo y su probabilidad de emisión ($L = \sum_i p_i * l_i$) y la entropía como menos la sumatoria del producto entre la probabilidad de aparición de un símbolo y el logaritmo en base dos de la misma ($H = -\sum_i p_i log_2 p_i$). La entropía se puede interpretar de varias maneras: la longitud de código promedio que se necesita para codificar una señal, el número de preguntas binarias que se deben hacer en promedio para identificar un elemento o símbolo de una señal.

Para la señal de Beethoven, el rendimiento es de 99.9377% y el de la señal L-Gante es de 99.503426%. En ambos casos, mediante el algoritmo de Huffman, se obtienen rendimientos realmente altos y se evidencia la gran eficiencia del método, ya que por el resultado se ve que la longitud media de código necesaria es muy cercana a la entropía, que es el ideal. Los rendimientos obtenidos son, en nuestra opinión, más que deseables, aunque podrían haber sido más bajos si la cantidad de símbolos repetidos hubiera sido menor. Por ejemplo, si la señal tuviera 1000 símbolos de los cuales ninguno es repetido, el rendimiento que se podría lograr sería más limitado que el obtenido para las señales dadas. Consideramos que el rendimiento podría ser mejorado en el caso de que las señales tengan más repeticiones de un mismo símbolo, con lo que habría menos códigos y más cortos, mejorando aún más el rendimiento.

A continuación se muestra el pseudocódigo propuesto.

```
log2 (probabilidad){
  resultado = log(probabilidad) / log(2); //se aplicó la regla matemática log2N = log N / log 2
  return - resultado; //se devuelve el negativo de resultado para que sea positivo
}

entropia(){
  for ∀ símbolo ∈ distribución
    entropia = entropia + simbolo.probabilidad * log2(símbolo.probabilidad);
  return  entropia; //se devuelve el negativo de óptima para que sea positiva
}

longitudMedia(){
  for ∀ símbolo ∈ distribución
    media = media + símbolo.probabilidad * longitud[símbolo];
  return media;
}

float rendimiento(){
  return entropia / longitudMedia;
}
```

Conclusiones

Luego de realizado el trabajo, podemos concluir con seguridad que el método de Huffman supone una mejora notable para la optimización del tamaño que ocupa una señal para su posterior almacenamiento o transmisión. Fue evidenciado que, para ambas señales, Huffman provee un rendimiento de un porcentaje verdaderamente alto, con lo que se reduce en gran medida el tamaño de la señal original. Para ingresar al código en Replit haga click aquí.