Trabajo Práctico Integrador 1

Integrantes:

- de Lellis, Lucas (lucasdelellis2002@gmail.com)
- Ramos Kees, Teo (teoramites@gmail.com)
- Romeo, Renzo Agustin (renzoromeo44@gmail.com)

Índice

Repositorio Git: link.

Resumen

A partir de un archivo con 10.000 caracteres aleatorios de un alfabeto {A, B, C} se calcularon las probabilidades condicionales de aparición de cada símbolo para conocer si se trata de una fuente de memoria nula o no nula. Luego, se generó la extensión de orden 20 con estos símbolos y se calculó su entropía.

Luego, se consideró que las cadenas del archivo representaban palabras de un código de 3, 5 y 7 caracteres de longitud. Para cada palabra se calculó la cantidad de información y luego la entropía. Se plantearon las inecuaciones de Kraft y McMillan, se obtuvo la longitud media para cada código, y luego se verificó si se trataba de códigos compactos. Además, se calculó el rendimiento y la redundancia de cada uno. Por último, se codificaron los símbolos de los códigos anteriores utilizando el algoritmo de Huffman y se reconstruyó el archivo original utilizando estas codificaciones compactas.

Introducción

Este trabajo se realizó con el objetivo de aplicar los temas tratados en la materia. El primer concepto tratado es el de cantidad de información de un evento, que está definida por la siguiente fórmula:

$$I(E) = log(\frac{1}{P(E)})$$

Fórmula 1: Cantidad de información de un evento.

Luego se trató el concepto de probabilidad condicional entre los símbolos, es decir, la probabilidad de que aparezca un símbolo determinado habiendo aparecido otro previamente. Utilizando esta medida, se puede determinar si se trata de una fuente de memoria nula o no, ya que en una fuente de memoria nula la probabilidad de aparición de un símbolo no depende de los símbolos que aparecieron previamente.

El siguiente concepto tratado es el de entropía, que representa la cantidad media de información por símbolo de la fuente, y se calcula de la siguiente forma:

$$H(S) = \sum_{S} P(S_i)\, log(\frac{1}{P(S_i)})$$

Fórmula 2: Entropía de una fuente.

El concepto de extensión de una fuente representa todas las combinaciones de una determinada longitud de los símbolos pertenecientes a la fuente original. A esta longitud se la llama orden de la extensión. Su entropía puede calcularse de la siguiente forma:

$$H(S^n) = \sum_{S^n} P(\sigma_i)\, log(\frac{1}{P(S_i)})$$

Fórmula 3: Entropía de una extensión de orden N.

Siendo, la probabilidad de cada palabra:

$$P(\sigma_i) = P_{i1} P_{i2} \dots P_{in}$$

Fórmula 4: Probabilidad de una palabra de una extensión de orden N.

Se puede demostrar que la siguiente fórmula es verdadera:

$$H(S^n) = nH(S)$$

Fórmula 5: Entropía de una extensión de orden N.

Esta fórmula permite calcular la entropía de la extensión de orden n de una fuente sin tener que generar la extensión.

En la segunda parte, se trataron conceptos relacionados a los códigos. Teniendo un alfabeto fuente y un alfabeto código, se define un código como una correspondencia entre todas las posibles combinaciones de un alfabeto fuente a secuencias de símbolos de otro alfabeto. Un código bloque es aquel que asigna a cada símbolo del alfabeto fuente una secuencia fija de símbolos del alfabeto código. También se puede definir un código como no singular cuando es un código bloque y todas sus palabras son distintas. Por otra parte, los códigos unívocamente decodificables son aquellos no singulares que permiten distinguir los

símbolos del alfabeto original a partir de una secuencia de símbolos del alfabeto código. Por último, se dice que un código es instantáneo si y sólo si ninguna palabra del alfabeto código es prefijo de la otra. Esta propiedad permite que para decodificar un símbolo no haya necesidad de esperar la aparición de los símbolos que lo suceden.

Luego, se desarrollaron las inecuaciones de Kraft y McMillan. Ambas se representan por la misma inecuación matemática, pero tienen diferentes interpretaciones.

$$\sum_{i=1}^{q} r^{-l_i} \leq 1$$

Fórmula 6: Inecuaciones de Kraft y McMillan.

Siendo:

- q: cantidad de palabras del código.
- r: cantidad de símbolos diferentes que constituyen el código.
- l_i: longitud de cada palabra del código.

Por un lado, el cumplimiento de la inecuación de Kraft es condición necesaria y suficiente para la existencia de un código instantáneo. Por otro lado, McMillan demostró que si existe un código unívocamente decodificable, entonces se cumple la inecuación de McMillan.

Por otra parte, la longitud media de un código está dada por la siguiente fórmula:

$$L = \sum_{i=1}^{q} p_i l_i$$

Fórmula 7: Longitud media del código.

Esta medida tiene la característica de ser siempre mayor o igual a la entropía, gracias al Primer Teorema de Shannon:

$$H_r(S) \leq L < H_r(S) + 1$$

Fórmula 8: Primer teorema de Shannon.

Y se utiliza en el cálculo del rendimiento y de la redundancia del código:

$$\eta = \frac{H_r(S)}{L}$$

Fórmula 9: Rendimiento.

$$1 - \eta = \frac{L - H_r(S)}{L}$$

Fórmula 10: Redundancia.

Mayor redundancia implica mayor información.

Por otro lado, se dice que un código es compacto cuando su longitud media es la mínima. Para lograr esto se debe cumplir la siguiente relación entre la probabilidad de la palabra y su longitud.

$$P_i = \left(\frac{1}{r}\right)^{l_i}$$

Fórmula 11: Condición de código compacto para una palabra.

Por último, se hizo uso del algoritmo de Huffman para generar un código binario compacto en base al código dado. Para llevar a cabo el algoritmo, se ordenan las palabras del alfabeto original en orden descendente de probabilidad. Luego, las dos palabras con menor probabilidad se agrupan y se suman sus probabilidades. Se realiza este proceso hasta obtener un alfabeto con solo dos palabras, y se les asigna un dígito binario a cada una. Luego, se realiza el proceso en reversa, agregando un dígito binario más a las dos palabras

que fueron agrupadas anteriormente, hasta finalmente obtener un código para el alfabeto original. La característica principal del código generado, es que es un código compacto que asigna a las palabras más frecuentes un código de menor longitud. Este proceso se puede aplicar tanto en fuentes de memoria nula como no nula.

Estos conceptos fueron aplicados durante el desarrollo del Trabajo Práctico sobre un archivo de 10.000 caracteres aleatorios compuesto por tres símbolos.

Desarrollo

Primer parte

Sobre el archivo de 10.000 caracteres, fue necesario calcular las probabilidades condicionales de cada símbolo para poder determinar si se trata de una fuente de memoria nula o no. Para calcular estas probabilidades fue necesario recorrer el archivo con los símbolos, almacenando la frecuencia de cada símbolo y a su vez la frecuencia de aparición de un símbolo habiéndose dado otro. Para almacenar estas frecuencias se utilizó una matriz de transición, que en cada posición contiene la frecuencia de aparición del símbolo de la fila i habiéndose dado el símbolo de la columna j. Luego, para obtener las probabilidades de los símbolos se dividió cada frecuencia por la cantidad total de símbolos presentes en el archivo, es decir, 10.000. Por otra parte, para obtener las probabilidades condicionales se dividió cada elemento de la matriz de transición por la suma de su columna correspondiente, que representa la cantidad total de apariciones de un símbolo habiéndose dado el símbolo de la columna. Entonces, la suma de todos los elementos de la matriz da un carácter menos que los caracteres del archivo ya que el último elemento no tiene ningún siguiente.

A continuación, se muestra el pseudocódigo que representa este proceso:

```
procedimiento leerArchivo(String nombreArchivo, double probabilidades[], double mat[][])
{
  archivo = abrir(nombreArchivo);
  anterior = leerChar(archivo);
  mientras (!finArchivo())
  {
    siguiente = leerChar(archivo);
    probabilidades[anterior]++;
    si (siguiente != null)
      mat[siguiente][anterior];
    anterior = siguiente;
  }
  sumarColumnas(sumas, mat); //suma la frecuencia de cada columna.
  calcularProbabilidadCondicional(sumas, mat); //divide cada columna por su frecuencia.
  calculaProbablidadSimbolos(probabilidades); //divide para cada simbolo por cant caracteres.
}
```

Luego, se compararon las probabilidades condicionales de cada símbolo con sus probabilidades independientes.

Probabilidad de cada símbolo	Matriz de transición:		
P(A) = 0.3978	0.396682	0.394349	0.407483
P(B) = 0.407	0.402966	0.412531	0.403895
P(C) = 0.1952	0.200352	0.193120	0.188621

Dado que las probabilidades son una aproximación, ya que fueron obtenidas a partir de un archivo finito, se observa una mínima diferencia entre el valor de probabilidad de cada símbolo y su probabilidad condicional. Esta diferencia no fue lo suficientemente significativa como para considerarla una fuente de memoria no nula.

Como la fuente resultó ser de memoria nula, fue necesario calcular la entropía de la extensión de orden 20 y de la fuente original. Para calcular la entropía de la fuente original, se utilizó la definición de entropía vista en la fórmula 2. El resultado obtenido fue 0.957087.

Luego, para calcular la entropía de la extensión se utilizó la fórmula 3. La parte más compleja del proceso fue la generación de la extensión de orden 20. Para generar dicha extensión, se utilizó un vector del tamaño igual al orden que inicialmente tenía todos sus

elementos igualados a cero. Luego, se utilizó un proceso análogo a contar en la base determinada por la cantidad de símbolos para poder calcular la siguiente palabra. Para cada palabra se calculó su probabilidad utilizando la fórmula 4 y se fue acumulando. Este proceso está plasmado en el siguiente pseudocódigo:

```
funcion entropiaExtension(double probabilidades[], integer orden)
{
  cantCiclos = elevar CANT_SIMBOLOS al orden;
  inicializarVector(extension);
  para i = 0 hasta cantCiclos
  {
    para j = 0 hasta orden
    {
      probabilidadExtension *= probabilidades[extension[j]];
    }
    entropiaOrdenN += probabilidadExtension * (-(log(probabilidadExtension) / log(CANT_S
IMBOLOS)));

    j = 0;
    mientras(j sea menor a orden y extension[j] sea mayor a CANT_SIMBOLOS)
    {
      extension[j] = 0; //Se inicializa de nuevo la letra de la palabra y se avanza
      j++;
    }
    Si (j es menor a orden)
      extension[j]++; //itero a la siguiente letra en dicha posicion
  }
}
```

El resultado obtenido para la entropía de la extensión de orden 20 fue 19.141749. Se puede observar que este resultado concuerda con la fórmula 5.

Segunda parte

En primer lugar, fue necesario calcular las probabilidades de aparición de cada palabra perteneciente a la extensión. Para esto, se leyeron los caracteres del archivo en grupos de la longitud correspondiente al orden de la extensión, se acumuló la frecuencia de cada palabra y finalmente se dividió por la cantidad total de apariciones. En el caso de no completar la última palabra, esta se ignoró.

En base a estas probabilidades, se calculó la cantidad de información y la entropía de cada palabra. Para obtener la cantidad de información de cada palabra se utilizó la fórmula 1, y la entropía se calculó utilizando la fórmula 2.

Todos los códigos analizados son de la misma longitud y distintos entre sí, por lo que ninguno puede ser prefijo del otro. Esta es una condición suficiente para afirmar que se trata de códigos instantáneos, y por lo tanto, unívocos, no singulares y códigos bloque.

Luego, se desarrollaron las inecuaciones de Kraft y McMillan, expresadas en la fórmula 6. El resultado obtenido para todos los códigos fue exactamente 1. Esto se puede deducir con el siguiente razonamiento. La cantidad de palabras del código (q) es igual a la cantidad de símbolos elevada a la longitud de la extensión. Como el término dentro de la sumatoria es constante, ya que todas las palabras del código tienen la misma longitud, el término se vuelve la cantidad de palabras elevada a menos la longitud de la extensión. Entonces, la fórmula se vuelve un producto de factores cuyos exponentes son inversos y da 1. Al cumplirse la inecuación de Kraft, se confirmó que los códigos analizados son instantáneos. Por otra parte, como se sabía de antemano que se trataba de códigos instantáneos, era esperable que se cumpliera la inecuación de McMillan.

Luego se calculó la longitud media para cada código utilizando la fórmula 7. Este resultado también se puede deducir ya que l_i es constante para todas las palabras del mismo código, por lo que se puede extraer como factor común, y la sumatoria de todas las probabilidades es igual a 1. Esto significa que la longitud media del código es igual al orden de la extensión.

Luego, para verificar si el código es compacto se debe cumplir la fórmula 11. En este caso, ninguno de los códigos es compacto ya que no cumplen esta condición. Esto es esperable ya que en los códigos compactos, las palabras con mayor probabilidad son las de menor longitud. En nuestro caso, todas las palabras tienen la misma longitud pero no son equiprobables.

Para los cálculos de rendimiento y redundancia se hizo uso de las fórmulas 9 y 10. Se puede observar que a medida que aumenta el orden de la extensión, la redundancia del código aumenta también. Esto se debe a que a medida que aumenta el orden de la extensión, y por ende la longitud de la palabra, hay una menor cantidad de palabras presentes en el archivo original. Esto significa que disminuirá la información presente en el mismo.

Por último, se reconstruyó el archivo original utilizando los códigos generados por el algoritmo de Huffman. Este algoritmo se implementó siguiendo dos pasos. Primero fue necesario crear un árbol de Huffman a partir de los códigos con su probabilidad. Este árbol tiene en cada nodo el código, su probabilidad y sus dos hijos. Para armarlo, se utiliza una cola de prioridades, es decir, una cola que está ordenada ascendentemente. Luego, se extraen los dos elementos de la cabeza de la cola y se agrupan en un nuevo nodo que tiene como probabilidad la suma de ambos, como palabra un “-” indicando que es un nodo que proviene de una fusión y como hijos los dos extraídos. Este nodo se agrega nuevamente a la cola. Este proceso se repite hasta que queda un solo elemento en la cola, que se

establece como la raiz del arbol. Como segundo paso, este árbol se recorre recursivamente para generar el código Huffman. Se parte de la raíz y, recorriendo el árbol en preorden, se agrega un ‘0’ cuando se va hacia un hijo izquierda y un ‘1’ cuando se va hacia un hijo derecha. Al llegar a una hoja, la cadena generada es el código Huffman para esa palabra, por lo que se almacena. El pseudocódigo que representa este algoritmo es el siguiente:

```
procedimiento algoritmoHuffman(Codigo codigos[]) {
  codigoHuffman = nuevo mapa de String a String

  q = nueva cola de prioridad

  para cada codigo
    nodo = crear nuevo nodo de Huffman
    nodo.palabra = codigo.palabra
    nodo.probabilidad = codigo.probabilidad
    agregar nodo a q

  mientras tamano de q > 1
    nodoA = quitar el primer elemento de q
    nodoB = quitar el primer elemento de q

    nodo = crear nuevo nodo de Huffman
    nodo.palabra = "-"
    nodo.probabilidad = nodoA.probabilidad + nodoB.probabilidad
    nodo.izq = nodoA
    nodo.der = nodoB

    agregar nodo a q

  raiz = quitar el primer elemento de q

  guardarCodigo(raiz, "", codigoHuffman) // Segundo paso del algoritmo

  devolver codigoHuffman
}
```

Para reconstruir el archivo se escribió cada símbolo de las palabras del código como un dígito binario. De esta forma, el espacio que ocupa es mínimo. Este archivo regenerado ocupa un tamaño considerablemente menor que el archivo original.

Conclusiones

A partir de los ejercicios resueltos en la parte uno se llegó a la conclusión de que el código brindado resultó en una fuente de memoria nula ya que las probabilidades condicionales obtenidas se podían calcular con la probabilidad de cada símbolo por separado. Además, se logró comprobar que calcular la entropía mediante cada palabra de la extensión de orden 20 concuerda con el producto de la entropía de la fuente original y el orden de la extensión.

Por otro lado, en la parte dos se pudo comprobar que los códigos analizados eran instantáneos ya que ninguna palabra era prefijo de otra. Además, al cumplirse la inecuación de Kraft se pudo verificar dicha afirmación. Por otra parte, dado que todas las palabras del código tenían la misma longitud y no eran equiprobables, era de esperarse que la longitud media sea igual al orden de la extensión y no se cumpla la condición de código compacto. Además, se observa que el rendimiento disminuye a medida que la longitud de la palabra crece. Esto se debe a que se encuentran menos palabras dentro del archivo ya que tiene una cantidad fija de caracteres. Por lo tanto la cantidad de información es menor. A partir del algoritmo de Huffman se pudo generar códigos compactos para los códigos de 3, 5 y 7 caracteres. Con esto fue posible reconstruir el archivo original de forma comprimida.