Geometría Computacional – Práctica 1

Miguel Manzano Rodríguez

1. Introducción

La presente práctica de Geometría Computacional tiene como objetivo çexplorar y aplicar los conceptos de código Huffman y el Primer Teorema de Shannon en los alfabetos del inglés y español, con muestras de los idiomas proporcionados en dos archivos de texto.

El código Huffman es una técnica de comprensión de datos que asigna códigos de longitud variable a conjuntos de símbolos para poder representarlos de manera más eficiente. Se basa en la asignación de códigos más cortos o largos a los símbolos más o menos frecuentes, respectivamente. Se utiliza así una representación binaria compacta, mediante la utilización del llamado “árbol de Huffman”, donde cada hoja representa un símbolo.

2. Material usado (método y datos)

En la sección i) del código, se llevará a cabo la obtención de los códigos Huffman binarios para SEng y SEsp, así como el cálculo de sus longitudes medias (L(SEng) y L(SEsp)). Además, estos cálculos servirán para verificar el Primer Teorema de Shannon. Todo esto se realiza mediante varias funciones: primero, una función para crear el código a partir del árbol de la plantilla; seguidamente, una función para calcular la longitud media de la codificación para un documento de los dos proporcionados; finalmente, una función para calcular la entropía.

En la sección ii), se utilizarán los códigos Huffman obtenidos para codificar la palabra "Lorentz" en ambos idiomas, permitiendo así comparar la eficiencia de longitud frente al código binario estándar. Para ambos idiomas, se incorpora una función que permite realizar la codificación de cualquier palabra dada la palabra y la codificación. Además, se utiliza otra función para la codificación en binario, utilizando concretamente la función ‘bytearray’.

Finalmente, en la sección iii), se desarrollará una función para decodificar cualquier palabra utilizando los diccionarios Huffman generados.

3. Resultados

Apartado i): Se muestran las codificaciones para los símbolos de los documentos proporcionados (se muestra en inglés, por ejemplo). Además, se muestran la entropía y longitud media de esta codificación en inglés. De esta manera, comprobamos que se cumple el primer teorema de Shannon, pues la longitud media está entre la entropía y la entropía más una unidad.

Apartado ii): Se muestra la codificación de ‘Lorentz’ en ambos idiomas, además de en binario.

Apartado iii): Se muestra la decodificación de lo anteriormente codificado.

```
In [42]: codigohuffman(tree_en)
Out[42]:
{'A': '000000110',
 '4': '000000111',
 'R': '111100100',
 'E': '111100101',
 'S': '010001110',
 'x': '010001111',
 '-': '00000010',
 'L': '01000110',
 'M': '01000100',
 'T': '01000101',
 'q': '11110011',
 'g': '0000000',
 'k': '11111000',
 'z': '11111001',
 ',': '0000010',
 '.': '0000011',
 'b': '1111000',
 'w': '1111010',
 'y': '1111011',
 'v': '1111101',
 'u': '010000',
 'p': '111111',
 'c': '00001',
 'f': '00010',
 'l': '00011',
 'd': '01001',
 'h': '01100',
 'm': '01101',
 'r': '0101',
 'a': '0111',
 'o': '1000',
 's': '1001',
 't': '1010',
 'n': '1011',
 'i': '1110',
 'e': '001',
 ' ': '110'}
```

```
In [44]: longitud_media(distr_en, d_en)
Out[44]: 4.303754266211604
```

```
In [46]: entropia_shannon(distr_en)
Out[46]: 4.2827303843795885
```

```
In [47]: p = 'Lorentz'

In [48]: p_cod_en = codificacion(p, d_en)

In [49]: display(p_cod_en)
'0100011010000101001101110101111001'

In [50]: p_cod_bin = codificacion_bin(p)

In [51]: display(p_cod_bin)
'100110011011111111001011001011101110111010011110'

In [52]: decodificar(p_cod_en, d_en)
Out[52]: 'Lorentz'

In [53]: p_cod_es = codificacion(p, d_es)

In [54]: display(p_cod_es)
'11111000100101100101000111011101101'

In [55]: decodificar(p_cod_es)
Traceback (most recent call last):
```

```
In [56]: p_cod_es = codificacion(p, d_es)

In [57]: display(p_cod_es)
'11111000100101100101000111011101101'

In [58]: decodificar(p_cod_es, d_es)
Out[58]: 'Lorentz'
```

4. Conclusión

En conclusión, hemos observado cómo podemos codificar un conjunto de símbolos proporcionados, obteniendo sus códigos de Huffman binarios. Además, hemos comprobado en la práctica la veracidad del Primer Teorema de Shannon. Finalmente, hemos codificado alguna palabra sin problema gracias a las codificaciones realizadas, comprobando también que no hay problema en dar marcha atrás a partir de la codificación, y que se obtiene la misma palabra.

En definitiva, esta práctica ha sido de utilidad para desarrollar un entendimiento práctico de estos conceptos teóricos, como el algoritmo para realizar la codificación, o la veracidad del Teorema.

5. Anexo con el script/ código utilizado

```
'''
# Apartado i): A partir de las muestras dadas, hallar el código Huffman binario
 de SEng y SEsp, y sus longitudes medias L(SEng) y L(SEsp). Comprobar que se
 satisface el Primer Teorema de Shannon. (1.50 puntos)
'''

# Primero, creamos el código huffman de las palabras de un idioma, y lo aplicamos
# para los árboles de los dos idiomas que hemos obtenido. Después, hallamos
# L(SEng) y L(SEsp), además de la entropía.

def codigohuffman(tree):

  lista_arbol = list(tree)
  d = {}

  for i in lista_arbol:

    [k0, k1] = list(i.keys())
    [v0, v1] = list(i.values())

    if (len(k0) == 1):
      d[k0] = str(v0)
    if (len(k1) == 1):
      d[k1] = str(v1)

    if (len(k0) > 1):
      for j in k0:
        d[j] = str(v0) + d[j]
```

```
        if (len(k1) > 1):
            for j in k1:
                d[j] = str(v1) + d[j]

    return d

d_en = codigohuffman(tree_en)
d_es = codigohuffman(tree_es)


def longitud_media(distr, d):
    L_S = 0
    caracteres = list(distr['states'])
    probabilidades = list(distr['probab'])
    i = 0
    while i < len(caracteres):
        L_S += len(d[caracteres[i]])*probabilidades[i]
        i += 1
    return L_S


def entropia_shannon(distribucion):
    entropia_res = 0
    probabilidades = list(distribucion['probab'])

    for probabilidad in probabilidades:
        entropia_res += probabilidad * np.log2(probabilidad)

    entropia_res *= -1
    return entropia_res
```

'''

ii) Utilizando los códigos obtenidos en el apartado anterior, codificar la palabra Lorentz para

ambas lenguas. Comprobar la eficiencia de longitud frente al código binario usual. (0.50 puntos)

'''

```
def codificacion(palabra, d):
    cod = ''
    for i in palabra:
        cod = cod + d[i]
    return cod

def codificacion_bin(palabra):
    lista = list(format(letra, 'b') for letra in bytearray(palabra, "utf-8"))
    cod = ''
    for i in lista:
        cod = cod + i
    return cod
```

'''

iii) Realiza un programa para decodificar cualquier palabra y comprueba que funciona con el

resultado del apartado anterior para ambos idiomas. (0.50 puntos)

'''

```
def decodificacion(palabra_codificada, d):
    resultado = ''
    i = 0
```

```
p_temp = ''
claves = list(d.keys())
valores = list(d.values())

while i < len(palabra_codificada):
    p_temp += palabra_codificada[i]

    if p_temp in valores:
        resultado += claves[valores.index(p_temp)]
        p_temp = ''
        i += 1
    else:
        i += 1

return resultado
    if p_temp in valores:
        resultado += claves[valores.index(p_temp)]
        p_temp = ''
        i += 1
    else:
        i += 1

return resultado
```