Trabajo Práctico N° 4

Codificación de fuentes de información

Primer Teorema de Shannon. Rendimiento y redundancia de un código. Códigos de Huffman. Código de Shannon-Fano. Compresión de datos. Métodos de compresión. Métodos de compresión sin pérdida. Algoritmo de Huffman. Run Length Coding (RLC). Codificación para control de errores. Códigos bloque. Detección y corrección de errores por chequeo de paridad.

1. Codificar una función booleana en Python que reciba como parámetros: una lista con la distribución de probabilidades de una fuente, otra lista con palabras código y el orden N de la extensión, y verifique si el código cumple con el Primer Teorema de Shannon.
2. Dada la siguiente fuente de información y su codificación, comprobar si la fuente y su extensión de orden 2 cumplen con el Primer Teorema de Shannon:

Fuente	Probs	Código
S1	0.3	BA
S2	0.1	CAB
S3	0.4	A
S4	0.2	CBA

3. Si se tiene una fuente de información con probabilidades P = { 0.5, 0.2, 0.3 } y los siguientes códigos para la fuente y su extensión de orden 2, verificar el cumplimiento del Primer Teorema de Shannon para ambas codificaciones:
   - C1 = { 11, 010, 00 }
   - C2 = { 10, 001, 110, 010, 0000, 0001, 111, 0110, 0111 }
4. Para una fuente binaria con ω = 0.8, calcular la extensión de orden 3 y proponer una codificación binaria que cumpla con el Primer Teorema de Shannon.
5. Para una fuente binaria con ω = 0.7:
   a. Obtener una codificación mediante el algoritmo de Huffman
   b. Codificar la extensión de orden 2 mediante el algoritmo de Shannon-Fano
   c. Comprobar si las codificaciones cumplen con el Primer Teorema de Shannon
6. Realizar una función en Python que reciba como parámetros: dos listas paralelas con la distribución de probabilidades de una fuente y su codificación, y calcule el rendimiento y la redundancia del código.
7. Comparar los rendimientos y las redundancias de los códigos del ejercicio 3.

8. Comparar los rendimientos y las redundancias de los siguientes códigos:

Fuente	Probs	Código 1	Código 2	Código 3	Código 4
A	0.2	01	00	0110	11
B	0.15	111	01	010	001
C	0.1	110	10	0111	000
D	0.3	101	110	1	10
E	0.25	100	111	00	01

9. Dadas las siguientes fuentes, generar códigos de Huffman y de Shannon-Fano:

Símbolos	S1	S2	S3	S4
Fuente A	0.2	0.2	0.3	0.3
Fuente B	0.4	0.25	0.25	0.1

10. Construir códigos de Huffman y de Shannon-Fano para fuentes de información que emiten los siguientes mensajes representativos:
    a. ABCDABCBDCBAAABBBCBCBABADBCBABCBDBCCCAAABB
    b. AOEAOEOOOOEOAOEOOEOOEOAOAOEOEUUUIEOEOEO
11. Desarrollar dos funciones en Python que reciban como parámetro una lista con la distribución de probabilidades de una fuente de información y generen una lista de cadenas de caracteres con codificaciones binarias de Huffman y de Shannon-Fano.
12. Para una fuente de información con distribución de probabilidades:
    P = { 0.385, 0.154, 0.128, 0.154, 0.179 }
    a. Calcular la entropía de la fuente
    b. Generar una codificación de Huffman
    c. Obtener una codificación de Shannon-Fano
    d. Graficar los árboles binarios que surgen de cada codificación
    e. Comparar la longitud media, el rendimiento y la redundancia de cada código
13. Dada una fuente de información que emite el siguiente mensaje representativo:
    587847845253686698957451236562536989896564521217023002236 59
    a. Calcular la entropía de la fuente
    b. Construir una codificación de Huffman
    c. Generar una codificación de Shannon-Fano
    d. Comparar la longitud media, el rendimiento y la redundancia de cada código
14. De acuerdo a la codificación del ejercicio 3, decodificar los siguientes mensajes:
    a. ABACBAACABABAACBABA
    b. BACBAAABAAACBABACAB
    c. CBAABACBABAAACABABA

15. Implementar funciones en Python que reciban como parámetros: una cadena de caracteres que contenga un alfabeto fuente y una lista de cadenas de caracteres que almacena una codificación en el alfabeto binario, y resuelvan lo siguiente:
    a. Dada una cadena de caracteres con un mensaje escrito en el alfabeto fuente, devolver una secuencia de bytes (bytearray) que contenga el mensaje codificado.
    b. Dada una secuencia de bytes, decodificar y retornar el mensaje original.

    Sugerencia: manipular el mensaje codificado como una cadena de caracteres de unos y ceros, tanto para codificar como para decodificar, y realizar las conversiones entre binarios y enteros con las funciones de casteo correspondientes.
16. Codificar una función en Python que reciba como parámetros: una cadena de caracteres con un mensaje y una secuencia de bytes (bytearray) con ese mensaje codificado y calcule la tasa de compresión.
17. Dada la siguiente tabla de probabilidades:

Símbolo	Prob	Símbolo	Prob	Símbolo	Prob	Símbolo	Prob
espacio	0.175990	D	0.038747	L	0.048149	S	0.058406
,	0.014093	E	0.101604	M	0.021041	T	0.031093
.	0.015034	F	0.004873	N	0.050490	U	0.033240
:	0.000542	G	0.008762	Ñ	0.002018	V	0.008930
;	0.002109	H	0.007953	O	0.073793	W	0.000012
A	0.111066	I	0.049740	P	0.019583	X	0.000706
B	0.015368	J	0.003706	Q	0.010246	Y	0.007851
C	0.030176	K	0.000034	R	0.051446	Z	0.003199

    Utilizando las funciones desarrolladas en los ejercicios 11 y 15, comprimir un mensaje mediante el algoritmo de Huffman y/o Shannon-Fano y almacenarlo en un archivo binario. Enviar el archivo a un compañero, quien tendrá que extraer y descomprimir el mensaje. Ambos deben calcular la tasa de compresión, el rendimiento y la redundancia.
18. Comprimir los siguientes mensajes utilizando el algoritmo RLC:
    a. XXXYZZZZ
    b. AAAABBBCCDAA
    c. UUOOOOAAAIEUUUU
19. Realizar una función en Python que reciba como parámetro una cadena de caracteres con un mensaje y devuelva una secuencia de bytes (bytearray) que contenga el mensaje comprimido con RLC, utilizando un byte para almacenar la representación en código ASCII del carácter y otro byte para el número.
20. Volver a comprimir los mensajes del ejercicio 18 (utilizando la función desarrollada) y calcular la tasa de compresión de cada uno.

21. Dados los siguientes códigos que representan colores:

Color	Código 1	Código 2	Código 3
Rojo	00	000	0000
Amarillo	01	100	0011
Verde	10	101	1010
Azul	11	111	0101

    a. Calcular la distancia de Hamming de cada código
    b. Determinar para cada código cúantos errores se pueden detectar y corregir

22. Implementar una función en Python que reciba una lista de cadenas de caracteres que representa una codificación binaria y devuelva: la distancia de Hamming, la cantidad de errores que se pueden detectar y la cantidad de errores que se pueden corregir.

23. Dados los siguientes códigos que representan colores:

Color	Código 1	Código 2	Código 3
Rojo	0100100	0100100	0110000
Amarillo	0101000	0010010	0000011
Verde	0010010	0101000	0101101
Azul	0100000	0100001	0100110

    a. Obtener sus distancias de Hamming
    b. Especificar cúantos errores se pueden detectar y corregir en cada caso

24. Desarrollar funciones en Python que resuelvan lo siguiente:
    a. Dado un carácter, devolver un byte que represente su código ASCII (7 bits) y utilice el bit menos significativo para almacenar la paridad del código.
    b. Dado un byte que se obtuvo como resultado de la función anterior, verificar si es correcto o tiene errores.

25. Dadas las siguientes matrices que contienen mensajes representados con código ASCII y sus bits de paridad vertical, longitudinal y cruzada, detectar los errores y, en caso de ser posible, recuperar el mensaje original:

a.
```
00100001
10000111
10000010
10100110
10000010
```

b.
```
00101101
10011001
10001010
10011100
10000010
```

c.
```
00101010
10000010
10011010
10011111
10100101
```

d.
```
00010100
10010000
10011110
10011001
10000010
```

e.
```
00110101
10011010
10101011
10100100
10000010
```

f.
```
00001001
10101001
10100101
10001011
10100110
```

g.
```
00011101
10010011
10011101
10001100
10011111
```

h.
```
00111110
10000111
10010000
10000010
10101010
```

26. Codificar funciones en Python que resuelvan lo siguiente:
    a. Dada una cadena de caracteres, generar una secuencia de bytes (bytearray) que contenga su representación con código ASCII y sus bits de paridad vertical, longitudinal y cruzada.
    b. Dada una secuencia de bytes que se obtuvo como resultado de la función anterior, devolver el mensaje original o una cadena de caracteres vacía si no se pueden corregir los errores.

    Sugerencia: generar una matriz de bits para realizar las operaciones, transformando la secuencia de bytes en una lista de cadenas de caracteres binarias y, luego, cada cadena de caracteres en una lista de números enteros que representen los bits.

Resultados:

2. n = 1 $H_3(S) = 1.16$ $L_1 = 1.9$ Cumple
   n = 2 $H_3(S) = 1.16$ $L_2 = 3.8$ No cumple
3. n = 1 $H_2(S) = 1.49$ $L_1 = 2.2$ Cumple
   n = 2 $H_2(S) = 1.49$ $L_2 = 3$ Cumple
5. a. $L_1 = \{ 1, 1 \}$
   b. $L_2 = \{ 1, 3, 2, 3 \}$
   c. n = 1 $H_2(S) = 0.88$ $L_1 = 1.00$ Cumple
      n = 2 $H_2(S) = 0.88$ $L_2 = 1.81$ Cumple
7. $\eta_1 = 0.6752$ $\eta_2 = 0.9903$
   $R_1 = 0.3248$ $R_2 = 0.0097$
8. $\eta_1 = 0.7958$ $\eta_2 = 0.8738$ $\eta_3 = 0.9095$ $\eta_4 = 0.9903$
   $R_1 = 0.2042$ $R_2 = 0.1262$ $R_3 = 0.0905$ $R_4 = 0.0097$
9. $L_A = \{ 2, 2, 2, 2 \}$ $L_B = \{ 1, 3, 2, 3 \}$
10. a. S = { A, B, C, D } P = { 0.26, 0.4, 0.24, 0.1 } L = {2, 1, 3, 3}
    b. S = { A, E, I, O, U } P = { 0.13, 0.28, 0.02, 0.49, 0.08 } L = {3, 2, 4, 1, 4}
12. a. $H_2(S) = 2.1854$ bits
    b. $L_{Huffman} = \{ 1, 3, 3, 3, 3 \}$
    c. $L_{Shannon\text{-}Fano} = \{ 2, 3, 3, 2, 2 \}$
    e. $L_{Huffman} = 2.2300$ bits $\eta_{Huffman} = 0.9800$ $R_{Huffman} = 0.0200$
       $L_{Shannon\text{-}Fano} = 2.2820$ bits $\eta_{Shannon\text{-}Fano} = 0.9576$ $R_{Shannon\text{-}Fano} = 0.0424$
13. a. $H_2(S) = 3.2041$ bits
    b. $L_{Huffman} = \{ 4, 4, 3, 3, 4, 3, 3, 4, 3, 3 \}$
    c. $L_{Shannon\text{-}Fano} = \{ 4, 4, 3, 4, 4, 2, 3, 4, 3, 4 \}$
    d. $L_{Huffman} = 3.2373$ bits $\eta_{Huffman} = 0.9898$ $R_{Huffman} = 0.0102$
       $L_{Shannon\text{-}Fano} = 3.2542$ bits $\eta_{Shannon\text{-}Fano} = 0.9846$ $R_{Shannon\text{-}Fano} = 0.0154$

14. a. S3 S1 S4 S3 S2 S3 S1 S3 S4 S1
    b. S1 S4 S3 S3 S1 S3 S3 S4 S1 S2
    c. S4 S3 S1 S4 S1 S3 S3 S2 S3 S1

18. a. X3Y1Z4
    b. A4B3C2D1A2
    c. U2O4A3I1E1U4

20. a. 4 / 3
    b. 6 / 5
    c. 5 / 4

21. a. $d_1 = 1$ $d_2 = 1$ $d_3 = 2$
    b. **Código 1:** no se pueden detectar ni corregir errores
    Código 2: no se pueden detectar ni corregir errores
    Código 3: se puede detectar un solo error pero no corregir ninguno

23. a. $d_1 = 1$ $d_2 = 2$ $d_3 = 3$
    b. **Código 1:** no se pueden detectar ni corregir errores
    Código 2: se puede detectar un solo error pero no corregir ninguno
    Código 3: se pueden detectar dos errores pero solo corregir uno

25. a. CASA b. LUNA c. AMOR d. HOLA
    e. — f. — g. — h. —