Instituto Politécnico Nacional
Escuela Superior de Cómputo

Análisis de Algoritmos, Sem: 2022-2, 3CV11, Práctica 5, 28 de mayo de 2022

Práctica 5: Algoritmos Greedy

Luis Francisco Renteria Cedillo, Denzel Omar Vazquez Perez.

*lrenteriac*1400@*alumno.ipn.mx*, *dvazquezp*1600@*alumno.ipn.mx*

Resumen: En el presente documento se muestra la la aplicación del algoritmo "código de Huffman" en imágenes en escala de grises con extensión BMP, con el objetivo de comparar el porcentaje de compresión de las mismas respecto a la original.

Palabras Clave: Matlab, Huffman, compresión, codificación, arboles binarios.

1 Introducción

Los algoritmos Greedy, también conocidos como algoritmos voraces, tratan de resolver un problema mediante la siguiente estrategia: se realiza una búsqueda para elegir una solución óptima particular con el objetivo de aproximarse a una solución general óptima. Por lo tanto, son ampliamente usados en problemas de optimización.

Los algoritmos Greedy no siempre generan soluciones óptimas globales, debido a que no se realiza una operación exhaustiva de todos los datos, sin embargo, su aplicación radica en su velocidad de encontrar aproximaciones en términos de optimización.

En cuanto a algunas de sus aplicaciones en problemas de la vida real se encuentran: planificación de actividades, compresión de volúmenes de información, minimización de tiempos de espera, optimización de recursos en cajeros automáticos, encontrar el camino más corto en redes de computadoras, entre otras.

2 Conceptos teóricos

2.1 Formato BMP

Un bitmap o una imagen BMP es uno de varios tipos de formatos de archivo para imagen, originalmente este tipo de archivos llevan la extensión *.BMP* . Las computadoras siempre usan bits de 1 y 0 para almacenar datos, tomando esto en cuenta se podría decir que una imagen BMP es literalmente un mapa de bits que forman una imagen particular cuando se representan en una pantalla como un monitor de computadora.

2.2 Teoría de la información

Dentro de la teoría de la probabilidad, existe una rama conocida como teoría de la información, la cual estudia la relación entre la información, compresión de datos, telecomunicaciones y criptografía.

2.3 Entropía de la información

En pocas palabras, la entropía se refiere a la probabilidad o incertidumbre que contiene un sistema, dicho de otro modo, es una medida del nivel de aleatoriedad de una fuente de información. Sin embargo, Claude E. Shannon, padre de la teoría de la información, establece que la entropía tiene dos propiedades:

1. Si se modifica la frecuencia de un elemento en una cantidad mínima, entonces el cambio de la entropía también es mínima.

2. Si todos los elementos tienen la misma probabilidad de ocurrir, entonces el nivel de entropía es máximo.

2.4 Compresión de datos

Tiene como objetivo la reducción de volumen espacial de un conjunto de datos, empleando una codificación a partir de la frecuencia de muestreo de los datos. Sin embargo, al comprimir la información se debe de realizar la operación inversa para recuperar la información original y lograr que no se vea afectada la calidad de la misma. Desde este punto de vista existen dos tipos de compresión de datos:

1. Compresión sin pérdidas: Al comprimir y descomprimir, la información mantiene su integridad

2. Comprensión con pérdidas: Al comprimir y descomprimir, la información obtenida es solo una aproximación a la original, ya que se elimina información que no es apreciable. Generalmente se aplican a videos o imágenes.

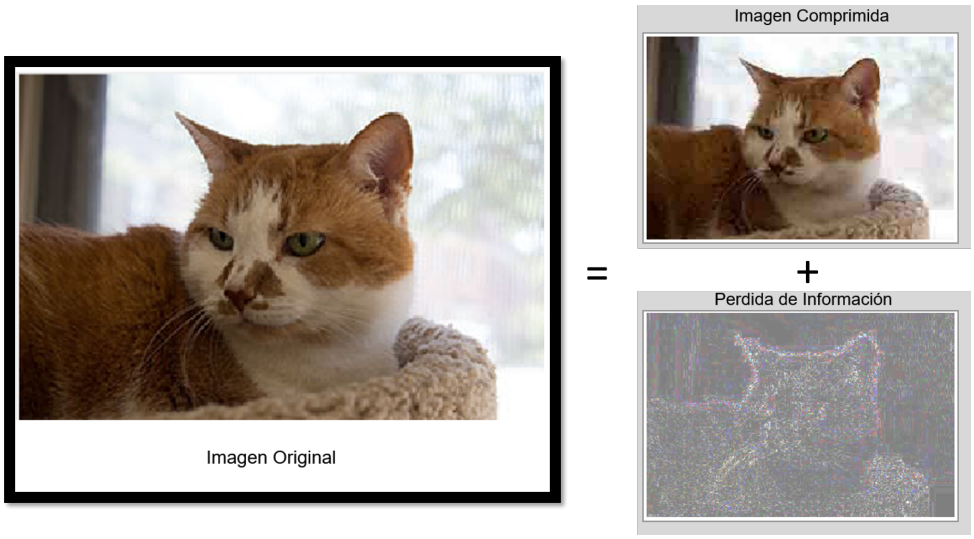


Figura 1: Ejemplo de compresión de datos con perdida (JPG)

2.5 Códigos de Huffman

David Huffman fue un pionero en su tiempo en el campo de ciencias de la computación, ya que realizó importantes atribuciones en diseño de señales en telecomunicaciones, pero el más popular fue su algoritmo de compresión de datos sin pérdida de información, conocido como codificación de Huffman.

El algoritmo de Huffman crea un árbol binario a partir de las frecuencias de caracteres o símbolos en una cadena. Una vez terminado del árbol se crea una tabla donde cada símbolo le corresponde un código binario único, donde los símbolos con mayor frecuencia tienen un código binario de menor longitud respecto a los códigos de los símbolos de menor frecuencia. Finalmente se construye una nueva cadena sustituyendo los símbolos por su respectivo código logrando con ello una compresión de datos importante para los símbolos con mayor frecuencia.

2.6 Algoritmo de Huffman

Vease anexo

2.7 Histograma de una imagen

Un histograma es una representación gráfica de los niveles de exposición que tiene una imagen. Si una imagen es a color, esta tendría 3 histogramas para cada uno de sus canales: rojo, verde y azul. Por el contrario, si solo tiene un canal estamos hablando de una imagen a escala de grises. Cada píxel de la imagen tiene el tamaño de un byte, por lo que su valor está en el rango de 0 a 255, entre mayor es el número, mayor exposición tiene el píxel. Por lo tanto, el histograma es una función que mide la frecuencia de todos los píxeles de la imagen y nos indica el nivel de exposición global de la imagen.

3 Experimentación y resultados

Se implementó el código de Huffman en Matlab R2020b, ya que nos facilita el manejo de matrices, imágenes e histogramas.

El funcionamiento del código es el siguiente:

1. Cargamos una imagen BMP en escala de grises y se muestra en pantalla.
2. Calculamos el histograma y se muestra su gráfica.
3. Preguntamos al usuario cuantos valores desea escoger de la imagen.
4. Creamos una nueva imagen con la cantidad de valores seleccionados.
5. Calculamos los códigos de Huffman y se muestran en pantalla.
6. Se codifica cada fila de la imagen con la tabla de códigos de Huffman.
7. Mostramos la Imagen Codificada.
8. Descodificamos la imagen.
9. Mostramos la imagen Descodificada.
10. Se imprime el valor de compresión respecto a la imagen original.

Para nuestro primer caso escogimos una imagen de 16x16 píxeles para poder mostrar paso a paso su ejecución. A continuación se muestra la figura junto con sus valores de cada píxel.

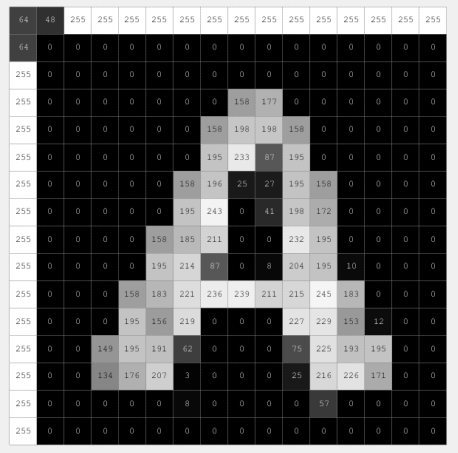


Figura 2: Imagen de 16x16 píxeles

La siguiente figura muestra la imagen original, la imagen con valores reducidos a 8 y la imagen codificada, cada una con su respectivo histograma.

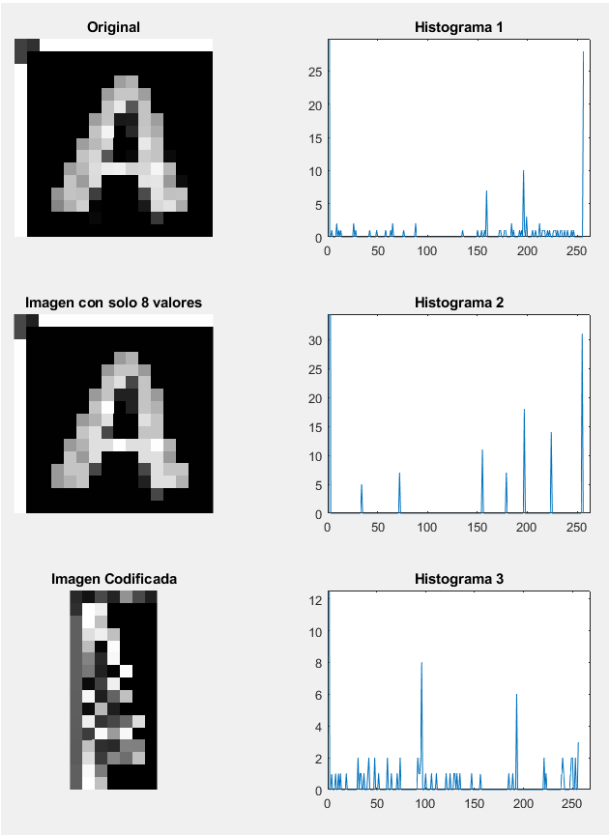


Figura 3: Imágenes e histogramas, ejemplo 1

Como puede observarse en los histogramas, lo que se logra al codificar la figura es que el histograma tiende a tener un comportamiento plano, ya que la entropía aumenta debido a que cada valor aumenta su probabilidad. Otro punto que destaca la imagen codificada es que se aprecia como si se tratara de ruido blanco.
Posteriormente se muestra los códigos de Huffman para cada uno de los 8 valores seleccionados, logrando una compresión al 43.75 por ciento respecto a la imagen original.

```
196->000
33->00100
71->00101
223->0011
254->010
0->0110
178->01110
154->01111
1->1
```

Figura 4: Códigos de Huffman, ejemplo 1

El siguiente ejemplo muestra un caballo con diferentes tonos de grises mientras que el fondo de la imagen es color negro.

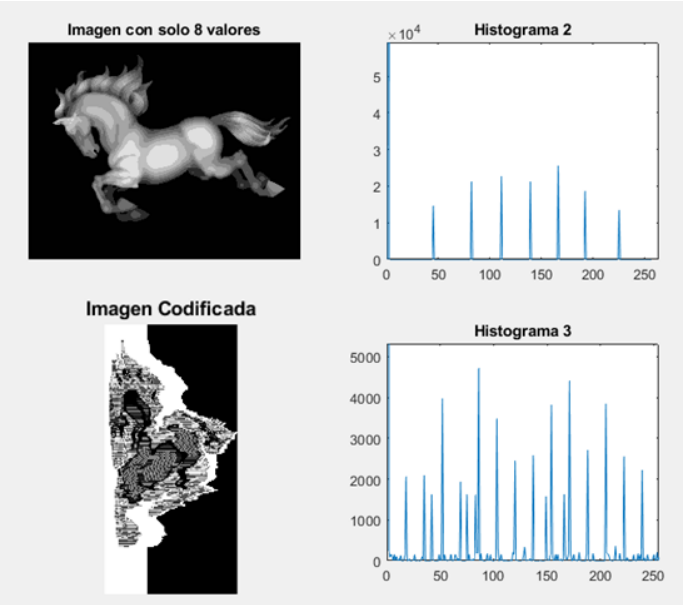


Figura 5: Imágenes e histogramas, ejemplo 2

A continuación se muestran los códigos de Huffman junto con el nivel de compresión respecto a la imagen original.

```
165->000
0->00100
224->00101
44->0011
191->0100
81->0101
138->0110
110->0111
1->1
Compresion:39.25%
```

Figura 6: Códigos de Huffman del ejemplo 2

Ahora realizaremos la codificación de Huffman para una imagen BMP donde la mayoría de sus píxeles son de un blanco puro.

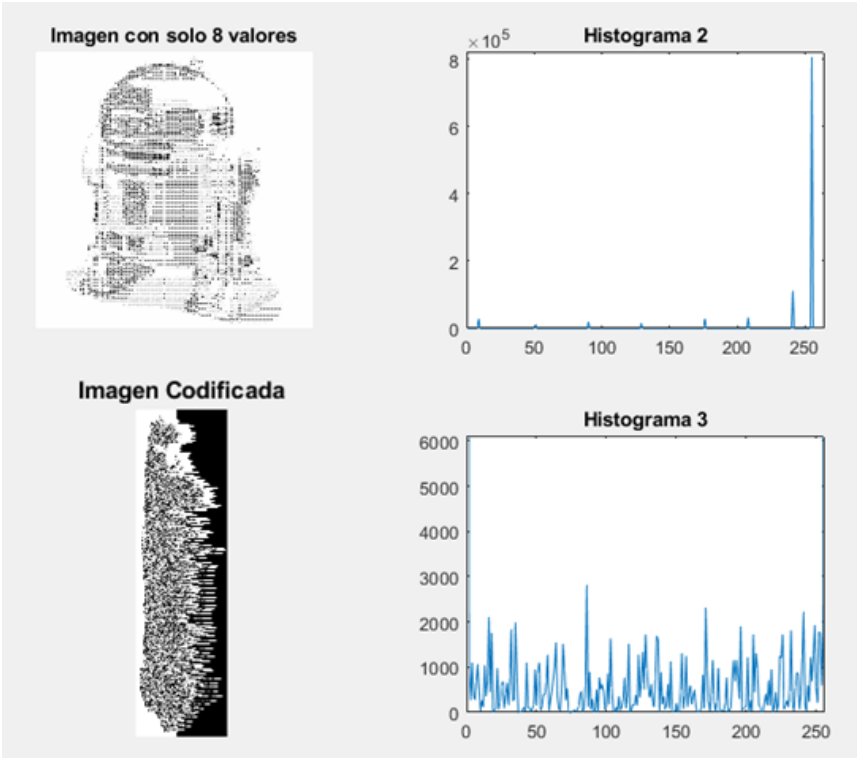


Figura 7: Imágenes e histogramas, ejemplo 3

A continuación se presentan los códigos de Huffman para el ejemplo 3:

```
240->00
8->0100
175->0101
207->0110
89->01110
0->0111100
50->0111101
128->011111
254->1
Compresion:27.9297%
```

Figura 8: Códigos de Huffman del ejemplo 3

Como puede observarse en los ejemplos anteriores, la compresión aumenta cuando el histograma tiene un pico muy importante en un valor especifico, como lo puede ser el negro o el blanco, sin embargo, cuando se aumenta el numero de valores seleccionados y el histograma de la imagen original tiene un comportamiento plano, entonces el nivel de compresión disminuye.

En cuanto a la decodificación de las imágenes fue clave mandar como argumentos tanto los códigos de Huffman como el ancho de la imagen a la función descodificar(), ya que sin ellos no sería posible recuperar la imagen original, por lo que se tiene el comportamiento de un sistema de criptografía si los datos mencionados fueran privados.

A continuación se mostraran las imágenes decodificadas respecto a las imágenes originales:

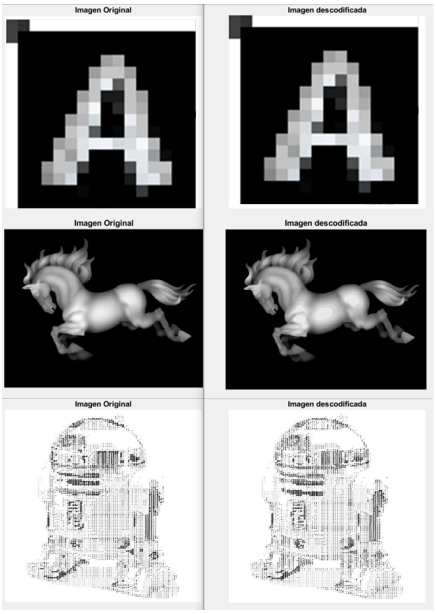


Figura 9: Decodificación de las imágenes de ejemplos 1, 2 y 3

En cuanto a mejoras de este algoritmo podemos mencionar que a la implementación de códigos de Huffman utilizamos el valor 0 como fin de linea, por ello este símbolo está dentro de la codificación de Huffman, y para lograr esto, antes de codificar la imagen nos aseguramos de que todos los píxeles con valor cero fueran sustituidos por unos. De esta manera, la decodificación se logra sin importar los valores que se hayan escogido.

Dado que se ha optado en esta practica por codificar linea a linea, se ha observado que las imágenes codificadas pierden la información visual y no se distinguen correctamente las formas ni el color, por lo que podemos enfocarnos en maximizar la compresión de la imagen sin importar su visualización en estado codificado, es decir, en vez de codificar linea por linea, codificar toda la imagen como si se tratara de una sola linea, ya que con este procedimiento podemos comprimir más la imagen debido a los espacios en negro que cada linea contenía al final de la misma.

4 Conclusiones

Luis Francisco Renteria Cedillo

Se ha observado que la compresión de datos utilizando codificación de Huffman es muy efectiva en cuanto reducción del volumen de información, sin embargo, para imágenes, la imagen codificada pierde la forma, el color y la textura, por lo que no se identifican los elementos presentes en ella. Por otra parte, la compresión JPG aplica la transformada discreta de cosenos a subregiones de la imagen, comúnmente de 8x8 píxeles, logrando deshacerse de las frecuencias más bajas, y al realizar la transformada inversa, no se perciben cambios notables en la imagen, debido a que el cerebro humano reconoce más fácil cambios bruscos de exposición, es decir, las frecuencias altas. Sin embargo, si se tratara de comprimir texto plano, el formato JPG no es factible dado a que se perdería

totalmente la información del texto plano, en este caso, la codificación de Huffman es un claro ganador.

Denzel Omar Vazquez Perez

Dada la complejidad del algoritmo de Huffman y al ser un algoritmo de codificación en prefijo, se asegura que al instante de descodificar la cadena de bits generada por esta no se presentara ninguna ambigüedad, sin embargo su aplicación en imágenes muestra que formas inusuales, como ver pinturas abstractas en una galería en pleno siglo XXI, y es que a simple vista no se identifica la imagen original en la codificación, sin embargo se podría mencionar que se muestra un mapa de intensidades de la imagen.

Las mejoras impuestas en permitieron seguir con esa secuencia en prefijo, lo que permitió inmediatamente la descodificación de lo antes hecho con Huffman.

Así resulta interesante cada uno de los resultados presentados en la practica abriendo cada vez mas el panorama a mas aplicaciones relacionadas a los algoritmo voraces vistos en clase, en lo particular nunca había trabajado con imágenes y se me es emocionante y un nuevo campo con el que seguir trabajando desde hoy en día.

5 Anexo

5.1 Orden de complejidad del algoritmo de mochila fraccionaria

Dado el conjunto de **N** artículos cada uno con un **B** beneficio con **w** peso y la capacidad total de la mochila , donde el objetivo es encontrar el valor máximo de fracciones de artículos que pueden caber en la mochila. Por tanto, como se muestra en la Figura 10 se tiene el siguiente algoritmo para cumplir este problema, así se muestra el análisis por bloques de código que determina que la complejidad del algoritmo es $T(n) \in \mathcal{O}(n\log(n))$ ante el peor caso de mochila fraccionaria.

```
1  mochila_Fraccionaria(b[0,...,n-1],w[0,...,n-1],P)
2      n=len(b)=len(w)
3      s=0
4      i=0
5      x=arreglo de tamaño n igual a 0
6      y=[0,...,n-1]
7      Quicksort(b,w,y) ---------- O(n log n)
8      while s<P && i<n do
9          k=selecciona(y)
10         i++
11         if s+w[k]<=P
12             X[k]=1
13             s+=w[k]
14         else
15             x[k]=(P-s)/w[k]
16             s=P
```

O(n): líneas 8–16 (bloque while)

O(n log n): líneas 7–16

Figura 10: Algoritmo de mochila fraccionaria

5.2 Contraejemplo del algoritmo de mochila fraccionaria

Sol.
Si se cuenta con una mochila de peso maximo P=8 y 4 objetos enteros no fraccionables, donde cada articulo tiene un peso w_i y un beneficio b_i, se tiene los siguientes pesos [4,3,5,2] y beneficios [10,40,30,20].

Por tanto el algoritmo voraz selecciona $W_{1,3}$ ya que son los objetos que entran dentro de la mochila para obtener

$$M_w = 3 + 2 = 5 \ y \ M_b = 40 + 20 = 60$$

A simple vista se observa que la el peso obtenido no alcanza el máximo que puede soportar la mochila, ya que excederá esta misma al agregar otro articulo.

Sin embargo existe una solución óptima escogiendo $w_{1,2}$

$$M_w = 3 + 5 = 8 \ y \ M_b = 40 + 30 = 70$$

Por tanto se demuestra que el algoritmo voraz de mochila fraccionaria al elegir objetos enteros no genera soluciones óptimas.

5.3 ¿Cuál sería la mejor función de selección voraz en el caso en el que todos los objetos tuvieran el mismo valor?

Para obtener una solución óptima se tendrán que elegir aquellos artículos con el mayor beneficio por el peso que puede soportar la mochila.

5.4 ¿Cuál sería la mejor función de selección voraz en el caso en el que todos los objetos tuvieran el mismo peso?

Para obtener una solución óptima se tendrán que elegir aquellos artículos con el mayor peso por el beneficio que adquiere la mochila.

5.5 Construir la codificación de Huffman para la cadena: ciencias de la tierra

Sol. Dada la cadena *"ciencias de la tierra"* se contruye la tabla de frecuencias de los caracteres contenidos en este, vease Tabla 1.

Tabla 1: Tabla de frecuencias de los caracteres de la cadena *ciencias de la tierra*

		a	c	d	e	i	l	n	r	s	t
Frecuencia	3	3	2	1	3	3	1	1	2	1	1

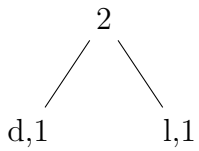
Así dados los resultados de la Tabla 1, se procede a ordenar los caracteres de mayor a menor frecuencia, véase Tabla 2.

Tabla 2: Tabla de frecuencias ordenada

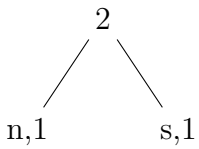
		a	e	i	c	r	d	l	n	s	t
Frecuencia	3	3	3	3	2	2	1	1	1	1	1

Por tanto haciendo usos de la Tabla 2 , se construye el árbol de frecuencias

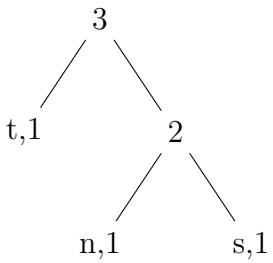
d	l	n	s	t	c	r		a	e	i
1	1	1	1	1	2	2	3	3	3	3



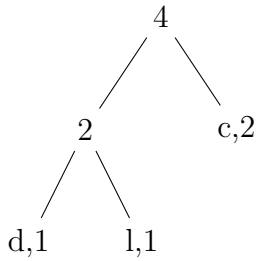
n	s	t	d,l	c	r		a	e	i
1	1	1	2	2	2	3	3	3	3



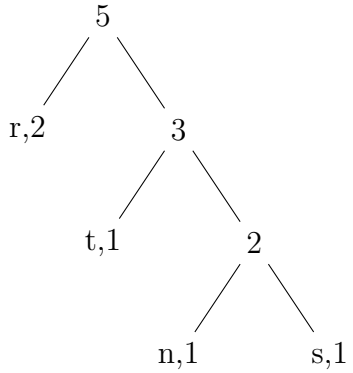
t	n,s	d,l	c	r		a	e	i
1	2	2	2	2	3	3	3	3



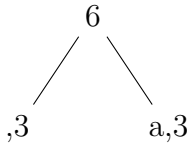
d,l	c	r	t,n,s		a	e	i
2	2	2	3	3	3	3	3



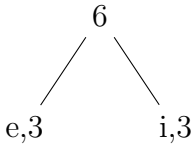
r	t,n,s		a	e	i	d,l,c
2	3	3	3	3	3	4



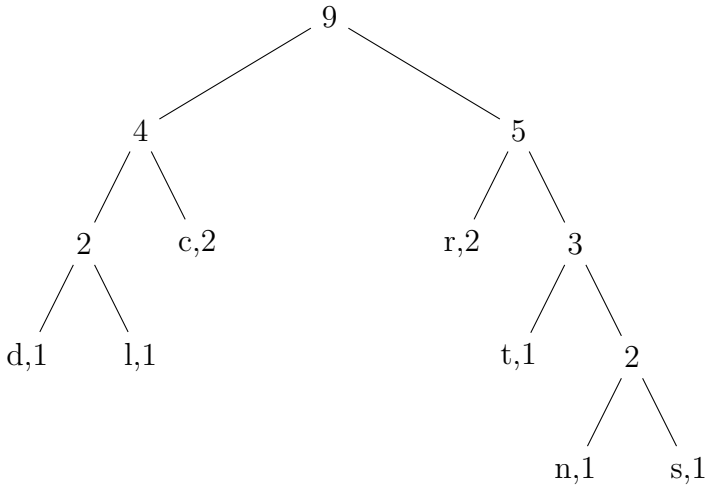
	a	e	i	d,l,c	r,t,n,s
3	3	3	3	4	5



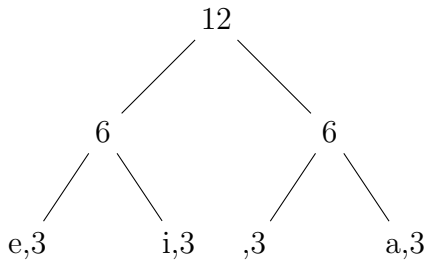
e	i	d,l,c	r,t,n,s	,a
3	3	4	5	6



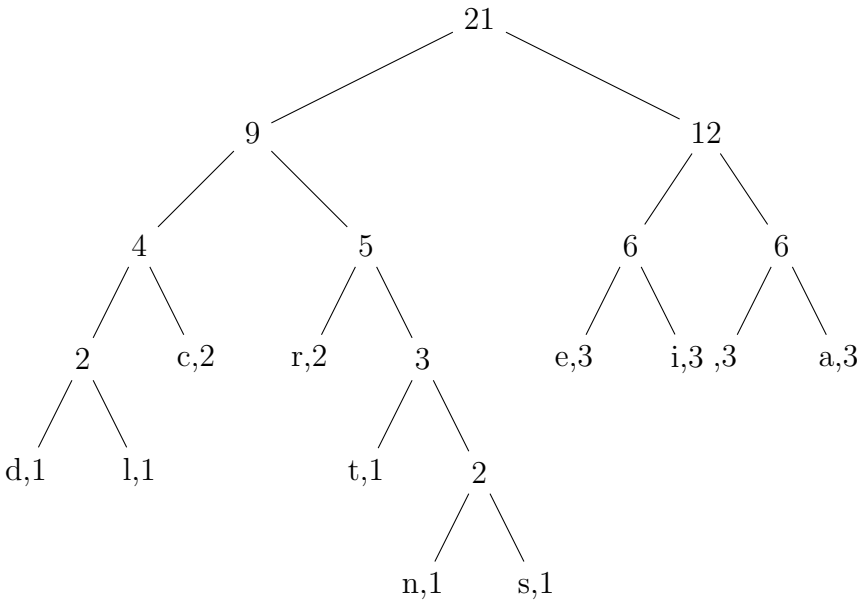
d,l,c	r,t,n,s	e,i	,a
4	5	6	6



e,i	,a	d,l,c,r,t,n,s
6	6	9



d,l,c,r,t,n,s	e,i, ,a
9	12



Se observa que cada nodo raíz puede solo tener un nodo hijo izquierdo como derecho, por lo que colocando un 0 y 1 respectivamente, se obtiene la siguiente codificación de Huffman, véase Tabla 13.

Tabla 13: Codificación Huffman

carácter	código
c	001
r	010
e	100
i	101

	110
a	111
d	0000
l	0001
t	0110
n	01110
s	01111

Finalmente la codificación de Huffman de la cadena *"ciencias de la tierra"* a partir de la Tabla 13 es:

ciencias de la tierra

001101100011100011011110111**110**0000100**110**0001111**110**0110101100010010111

5.6 Orden de complejidad del algoritmo de Huffman

El algoritmo de Huffman hace la compresión de datos sin perdidas asignando códigos en prefijo de longitud variable a los caracteres que reciba este, estos se generan a partir de frecuencias de los caracteres únicos de la cadena. Donde el carácter más frecuente obtiene el código más prequeño y el carácter menos frecuente obtiene el código más grande, de este modo se asegura que no exista la ambigüedad al decodificar la cadena de bits generada. Así se muestra en la Figura 11 el análisis por bloques de código determina que la complejidad del algoritmo es $T(n) \in \mathcal{O}(n \log(n))$ ante el peor caso de Huffman.

```
1  huffman(C:todos los caracteres)
2          n=|C|
3          Q=cola_prioridad(C)                    Θ(n log n)
4          for i=1 to i <= n do                   Θ(n)
5                  x=Q.extraerMin()
6                  y=Q.extraerMin()
7                  z=nuevo_nodo                                  Θ(n log n)
8                  z.izq=x                        Θ(1)    Θ(n)
9                  z.der=y
10                 z.freq=x.freq+y.freq
11                 Q.insertar(z)
12         return Q
```

Figura 11: Algoritmo de Huffman

5.7 Orden de complejidad del algoritmo de Kruskal

El algoritmo de Kruskal representa un árbol de expansión mńimo que hace usos de un grafo encontrando un subconjunto de aristas de este que formen un árbol con todos los nodos obteniendo la suma de costos mínimos entre todos los arboles que puede generar para obtener un óptiomo local, que pueda ser el global. Así se muestra en la Figura 12 el análisis por bloques de código que determina que la complejidad del algoritmo es $T(n) \in \mathcal{O}(n \log(n))$ ante el peor caso de este.

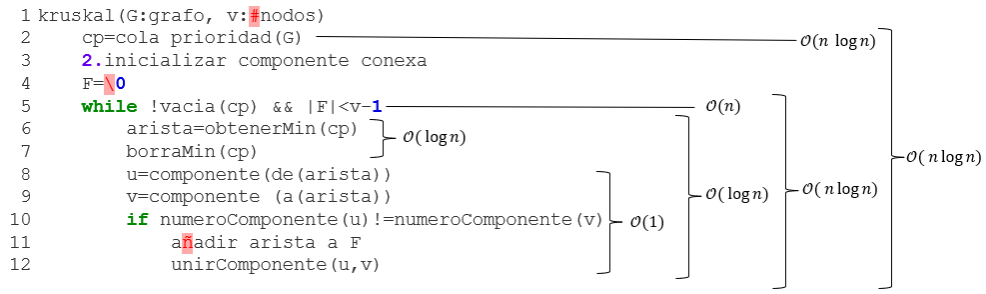


Figura 12: Algoritmo de Kruskal

5.8 Algoritmo Prim

El algoritmo de Prim incrementa el árbol de expasión mínimo comenzando por un vértice inicial al que sucesivamente se le aãden vértices cuya distancia a los padres sea mínima, así cada arista que aparece es por la coincidencia de vertices en el árbol. El presente pseudo-código determina que la complejidad del algoritmo es $T(n) \in \mathcal{O}(n \log(n))$ ante el peor caso de este.

Algorithm 1 Prim(G:grafo)

```
1: for 0 to |V| − 1 do
2:     D[u]=INFINITO
3:     Parent[u]=NULL
4:     Insert(Q,u)
5: end for
6: D[u]=0
7: while !isempty(Q) do
8:     u=min(Q)
9:     for each ∈ Adj[u] do
10:        if (v ∈ Q) && (D[v]) > w(u, v) then
11:            Parent[v]=u
12:            D[v]=w(u,v)
13:            update(Q,v,D[v])
14:        end if
15:    end for
16: end while
```

5.9 Orden de complejidad del algoritmo de Dijkstra

El algoritmo de Dijkstra, es una util herramienta para encontrar el camino mas corto o con menor costo entre todos lo nodos del grafo, a partir de un *"nodo inicial"*, a diferencia del árbol de expasión mínimo este no hace uso de todos lo vértices del grafo. Así se muestra en la Figura 13 el análisis por bloques de código que determina que la complejidad del algoritmo es $T(n) \in \mathcal{O}(n^2))$ ante el peor caso del algoritmo.

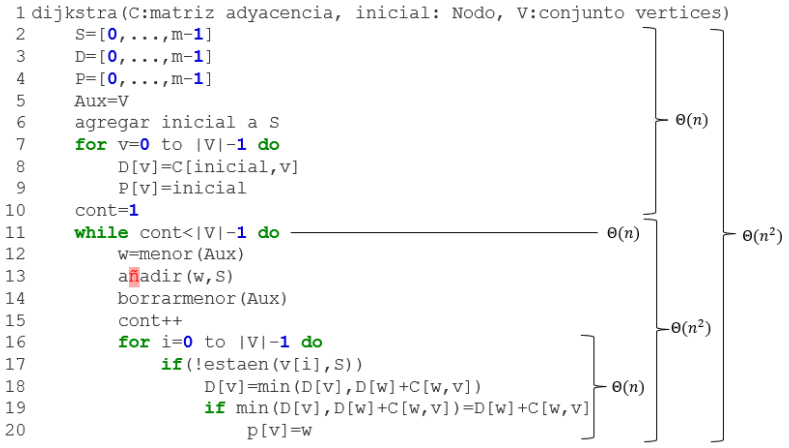


Figura 13: Algoritmo de Dijkstra

6 Bibliografía

Brassard, G. (1997). *Fundamentos de Algoritmia*. España: Ed. Prentice Hall.

Cormen, E. A. (2022). *Introduction To Algorithms*, 3Rd Ed. Phi.

Sánchez, P. J. I. (2006). *Análisis y diseño de algoritmos: un enfoque teórico y práctico*. Servicio de Publicaciones y Divulgación Científica de la UMA.

Nipun Ramakrishnan, Reducible (Julio 2021), *Huffman Codes: An Information Theory Perspective*, Youtube, https://www.youtube.com/watch?v=B3y0RsVCyrw