Logotipo

Descripción generada automáticamenteImagen que contiene Logotipo

Descripción generada automáticamenteInstituto Politécnico Nacional

Escuela Superior de Cómputo

**Práctica 4**

**Códigos Huffman**

Acevedo Juárez Sebastián Ávila Mejía Daniel Said

Prof.: Miriam Pescador-Rojas

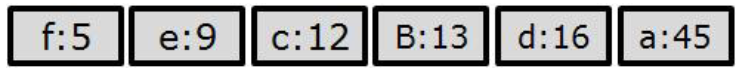
Análisis y Diseño de Algoritmos

**Introducción:**

Para comprimir la información se busca representar un conjunto de símbolos (o cadena de símbolos) obtenidos a partir de un cierto alfabeto, usando el menor número de bits posible, pero preservando en todo momento la capacidad de descomprimir o decodificar la información. En general, el sistema que realiza el proceso directo lo llamamos compresor o codificador, mientras que el que reconstruye los datos originales (o una aproximación a ellos si realizamos compresión con pérdidas) lo llamamos descompresor o decodificador. El algoritmo de codificación/compresión Huffman se propuso en 1952 como una forma sencilla y optima de mapear cada símbolo de un alfabeto con un código (codeword) de longitud óptima. De esta forma, para comprimir cada símbolo de la cadena, simplemente debemos usar el código que se ha calculado mediante Huffman. Para conseguir esta asignación óptima, los símbolos se representan con códigos cuya longitud es inversamente proporcional a la probabilidad del símbolo. De esta forma, los símbolos cuya frecuencia es menor en la cadena se representan con códigos más largos, mientras que los símbolos con mayor frecuencia de apariciones representan con códigos más cortos.

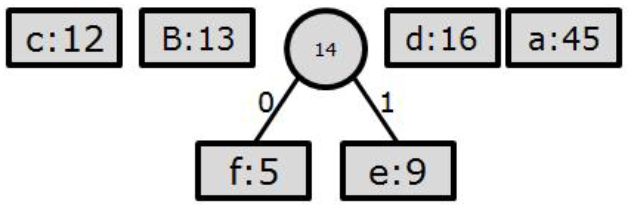
**Marco Teórico**

Consiste en la creación de un árbol binario en el que se etiquetan los nodos hoja con los caracteres, junto a sus frecuencias, y de forma consecutiva se van uniendo cada pareja de nodos que menos frecuencia sumen, pasando a crear un nuevo nodo intermedio etiquetado con dicha suma. Se procede a realizar esta acción hasta que no quedan nodos hoja por unir a ningún nodo superior, y se ha formado el árbol binario. Posteriormente se etiquetan las aristas que unen cada uno de los nodos con ceros y unos (hijo derecho e izquierdo, respectivamente, por ejemplo. El código resultante para cada carácter es la lectura, siguiendo la rama, desde la raíz hacia cada carácter (o viceversa) de cada una de las etiquetas delas aristas.

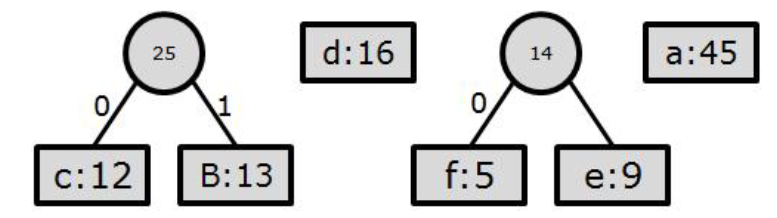


*Representación de lo antes mencionado:*

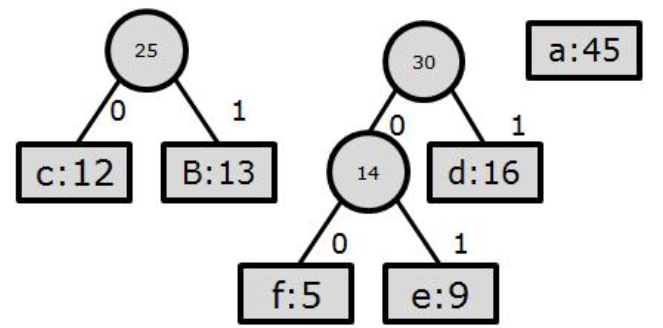
Seleccionamos los dos símbolos con menor frecuencias y los agregamos a un nodo con la suma de ambos, recalcando que el de menor frecuencia ira a la izquierda con el valor 0 en su rama.



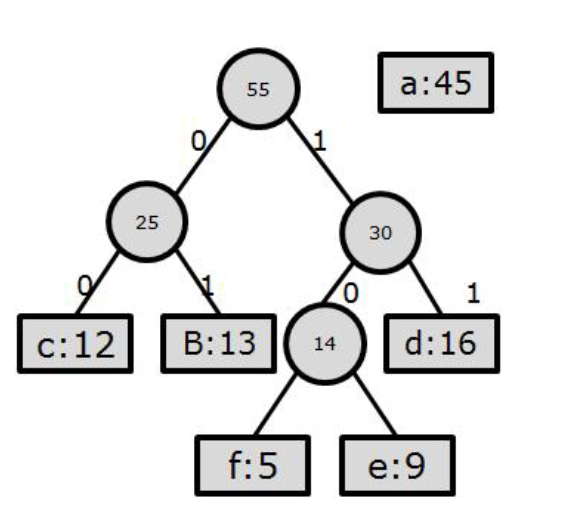
Repetimos el proceso de seleccionar los dos menores sumando sus cantidades.



Observemos que, ahora los dos menores incluyen a un nodo que hemos creado así que usamos este junto con el segundo menor y se crea una nueva rama y un nuevo nivel.



Seguimos con este proceso hasta terminar con los nodos restantes y obtener el árbol completo



**Implementación:**

Para poder implementar este problema, realizamos los siguientes pasos:

1. Desplegar menú con los archivos disponibles a codificar.
2. Ejecutar función madre Huffman() del archivo seleccionado
3. Obtener frecuencia de los caracteres.
4. Crear un árbol binario en base a las frecuencias.
5. Crear un código en base al árbol previamente creado.
6. Codificar el archivo seleccionado.
7. Decodificar la compresión y crear un nuevo archivo con el mensaje decodificado.

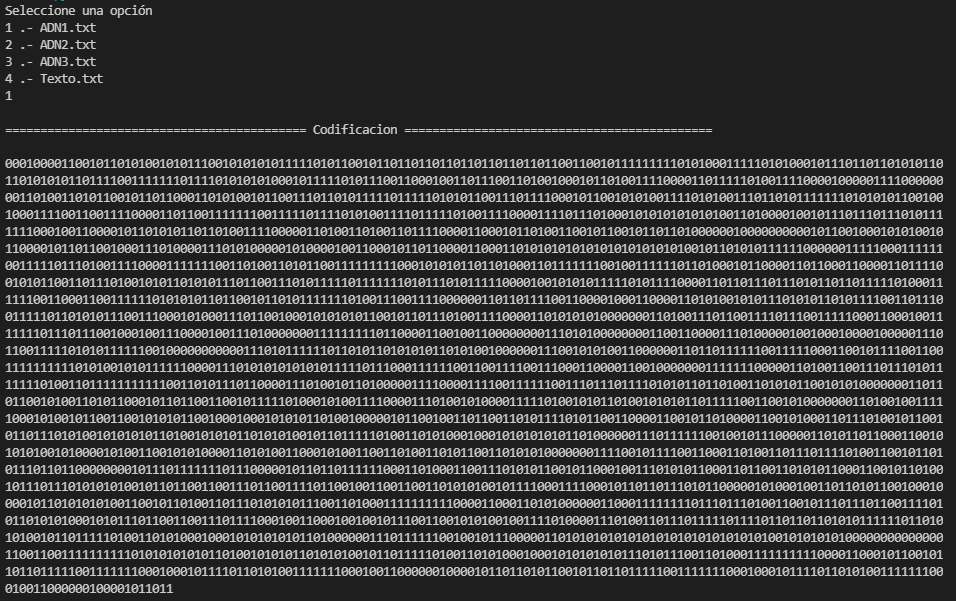
Al usar una imagen, los pasos tienen algún cambio, debido a que primero se tiene que convertir la imagen a un mapeado, para así poder escribirlo en un txt y realizar los pasos mencionados.

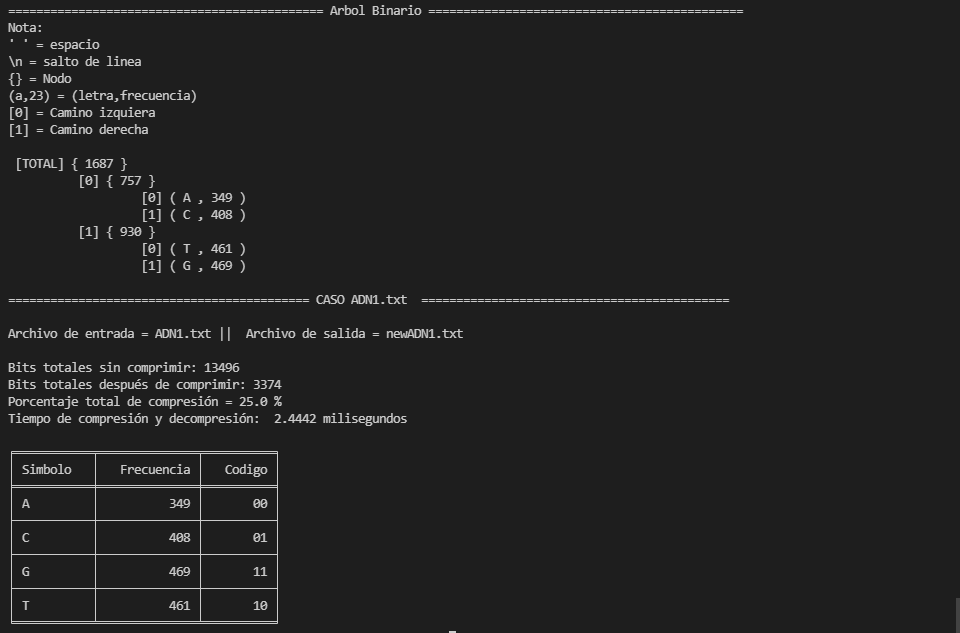
La imagen es representada mediante una matriz que representan los pixeles de una imagen. Al trabajar con una foto en escala de grises, cada celda de la matriz representa la intensidad de color. Este mapa se imprime en un nuevo archivo, al cual se le aplica Huffman.

Al ejecutar la función Huffman(), el archivo de entrada es un archivo ya existente. Sin embargo, el archivo de salida no existe si no se ha ejecutado previamente el programa. En este archivo, se encuentra el mensaje decodificado.

**Ejecución:**

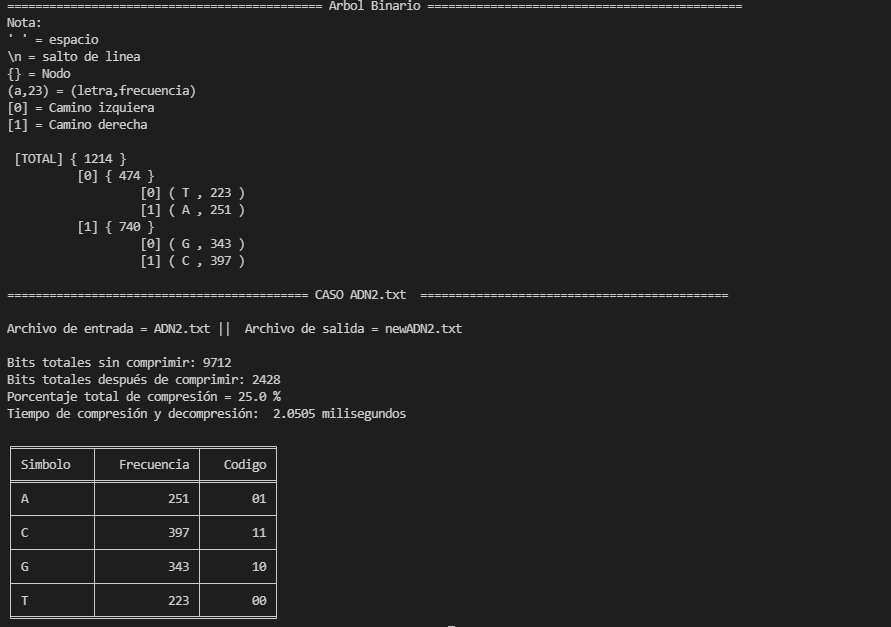
Para “ADN1.txt”





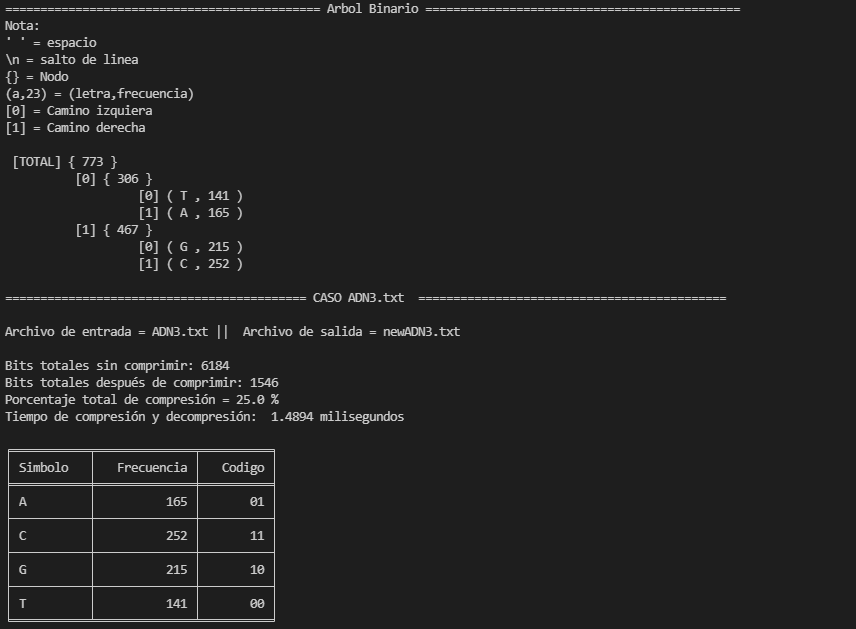
Para “ADN2.txt”





Para “ADN3.txt”



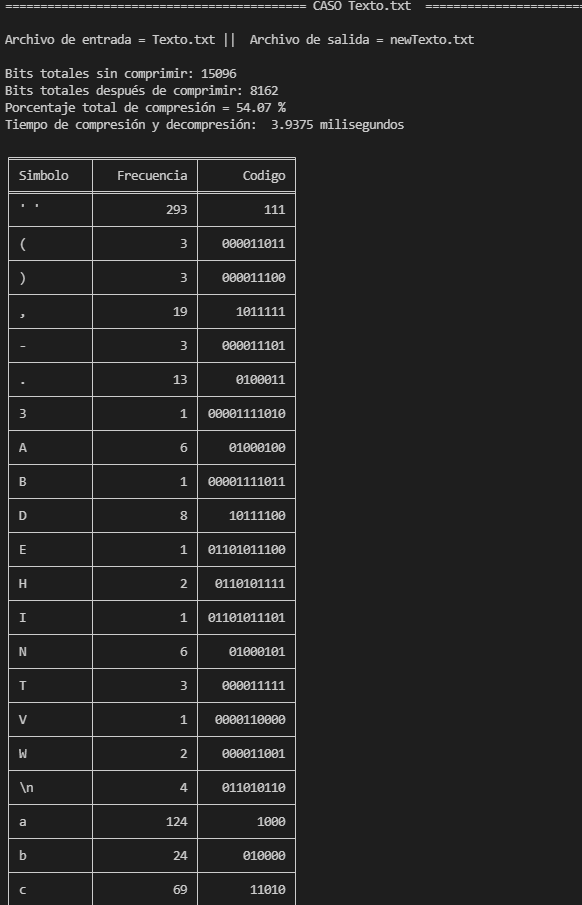


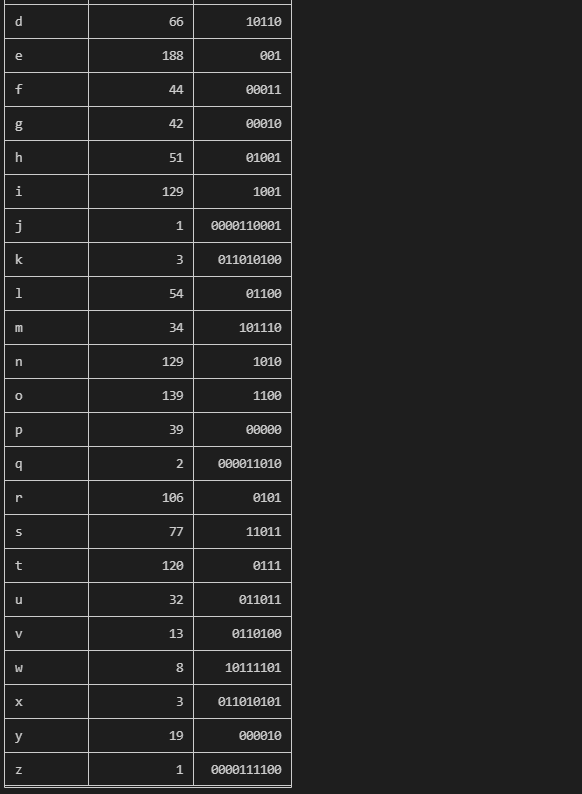
Para “Texto.txt”



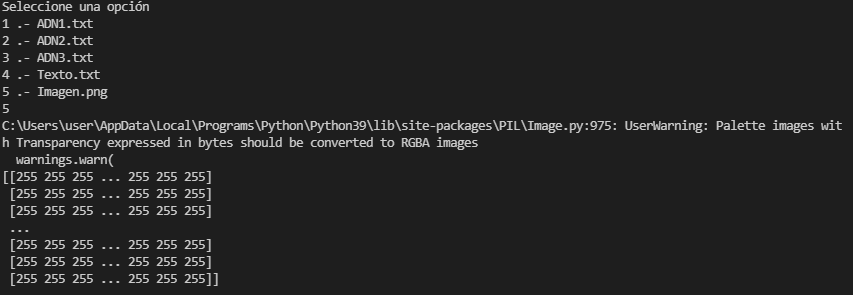






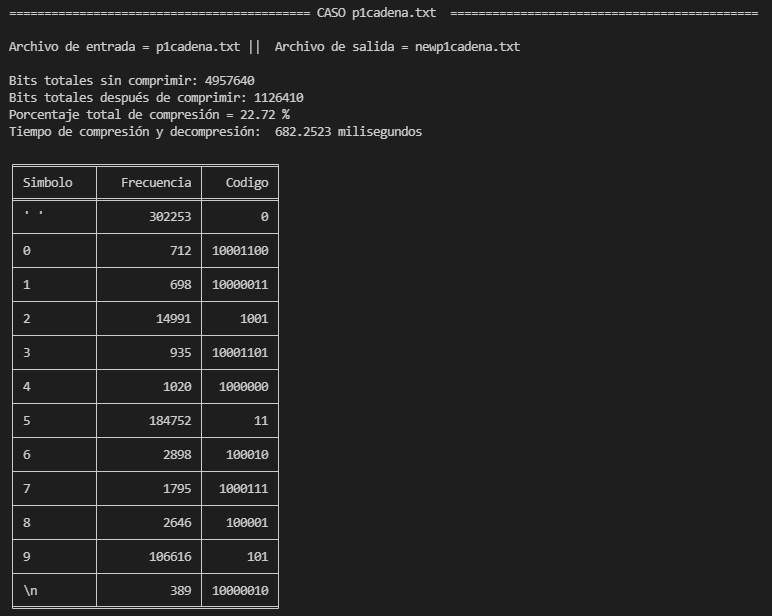


Para “Imagen.PNG”

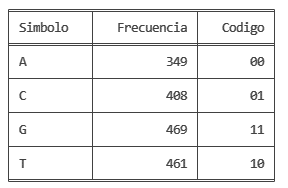








Tablas de Resultados



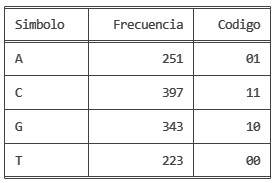
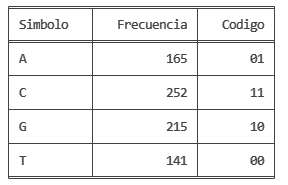
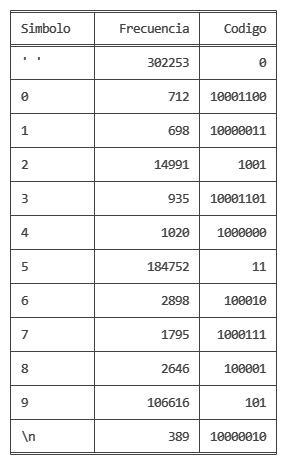


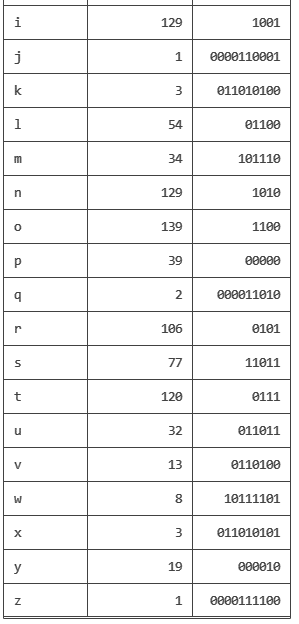
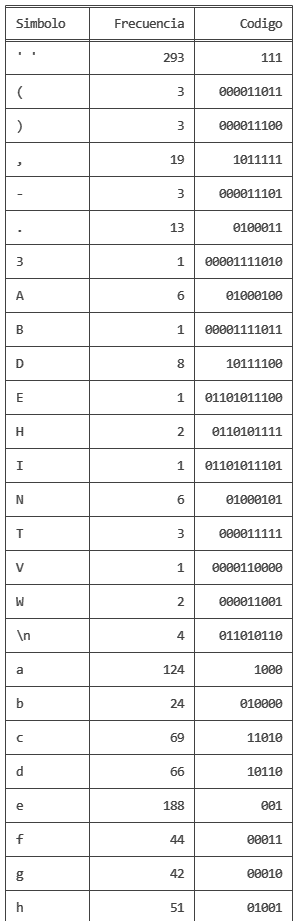
Imagen.PNG

ADN3.txt

ADN2.txt

ADN1.txt

Texto.txt



**Conclusión**

En la práctica se llevó a cabo un método de codificación (compresión y descompresión) para el cual se formuló una serie de algoritmos recursivos y por demás para poder tener una compresión satisfactoria, nos percatamos que es una codificación sin pérdida de datos. Teniendo un porcentaje de compresión promedio del 30%.La asignación de claves para cada carácter depende de la cantidad de veces que se repita cada carácter en el texto mientras se repita más veces el carácter, tendrá una clave más pequeña mientras menos se repita el carácter tendrá una clave más grande. Independientemente del archivo a comprimir, lo que determina que tan compacto sea el archivo resultante, será el número de veces que se repita los caracteres y símbolos que estén en el mismo. Con esto es evidente que una compresión sumamente eficiente es cuando la gran mayoría de los caracteres se repiten. De forma particular tomando en cuenta el lenguaje que se utilizó (Python) nos apoyamos en algunas librerías que facilitaron el análisis de datos, así como su acomodo e impresión de estos. Observamos que de igual forma se pudieron comprimir imágenes PNG con el mismo principio a coste de operaciones mayor.