Escuela de Ingenieria en Computacion
Taller de programacion
Grupo 21

Compresíon con Arboles de Huffman

Kendal Chacon Chaves
Carné:2024122026

Prof: Alex Brenes Brenes

17 de Junio de 2024

0.1. Introducción

En el ámbito de la informática, la compresión de datos ha sido una herramienta que ha venido a facilitarnos nuestro diario interactuar con dispositivos digitales, esto debido a que en la informática es esencial el poder recibir y enviar archivos, pero a veces pasa que dichos archivos tienen un peso elevado, por lo cual transmitirlos en su estado original podría ser una tarea que requiera mucho ancho de banda y tiempo por lo mismo.

La compresión de datos reduce el tamaño de un archivo al minimizar los datos redundantes. En un archivo de texto, los datos redundantes pueden producirse con frecuencia caracteres, como el carácter de espacio o los vocales comunes, como las letras e y a; también puede producirse con frecuencia cadenas de caracteres. La compresión de datos crea una versión comprimida de un archivo minimizando estos datos redundantes.[1]

En este proyecto se nos plantea la tarea de generar dos programas en python, un compresor y otro descompresor, los cuales tienen como su nombre lo indican; la tarea de comprimir o descomprimir un archivo cualquiera mediante el Algoritmo de Huffman, los cuales deberán ser programados en Python 3 y además deben incluir un archivo con stats en el cual se muestren:

- Altura del árbol
- Anchura del árbol
- Cantidad de nodos por nivel
- Tabla de frecuencia de caracteres(de aquellos que aparezcan una o mas veces)

En la primera parte de este documento se van a tratar temas tales como datos acerca de los algoritmos de compresión de archivos ya sea con pérdida o sin perdida, un poco de la vida de David A. Huffman y varias estructuras necesarias para completar la tarea que se nos presenta.

En la segunda parte del mismo vamos a estar abordando la solución al problema presentado, se van a detallar las funciones mencionando sus parámetros de entrada, librerías utilizadas, datos que se retornan y el proceso que ocurre dentro de ellas además de como una función se relaciona con otras para llegar al resultado deseado.

0.2. Desarrollo

Descripción del problema

En el presente proyecto, se nos presenta el reto de desarrollar dos programas, un compresor y un descompresor, basados en el Algoritmo de Huffman. Este algoritmo se fundamenta en la generación de tablas de frecuencia y árboles binarios que representan la frecuencia de los caracteres presentes en un archivo, permitiendo así comprimir la información de manera eficiente. Sumado a esto en el programa de compresión se deben generar tres archivos:

- Un archivo con extensión .huff el cual va a ser nuestro archivo ya comprimido
- Un archivo de texto con la tabla de frecuencia de los caracteres con extensión .table
- Un ultimo archivo en el cual se van a presentar las estadísticas del árbol generado en el cual vamos a contener:
  - Altura del árbol
  - anchura del árbol
  - Cantidad de nodos
  - Tabla de frecuencia original

Por su parte el programa de descompresión de archivos debe de recibir como argumento el nombre del archivo comprimido, el nombre del archivo con la tabla de códigos y el nombre del archivo destino en el que se desea guardar el resultado. En la buena teoría nos debería devolver el archivo original antes de ser comprimido sin la más mínima perdida de datos en el proceso.

Biografía de David A. Huffman

Originario de Ohio, David A. Huffman obtuvo su B.S. en ingeniería eléctrica de la Universidad Estatal de Ohio a la edad de 18 años en 1944. Luego sirvió en la Marina de los EE. UU. como oficial de mantenimiento de radar en un destructor que ayudó a limpiar minas en aguas japonesas y chinas después de la Segunda Guerra Mundial. Posteriormente obtuvo su M.S. grado de Estado de Ohio en 1949 y su Ph.D. del MIT en 1953, también en ingeniería eléctrica.

Huffman se unió a la facultad del MIT en 1953. En 1967, fue a la Universidad de California, Santa Cruz, como miembro fundador de la facultad del Departamento de Ciencias de la Computación. Desempeñó un papel importante en el desarrollo de los programas académicos del departamento y la contratación de su personal docente, y se desempeñó como presidente de 1970 a 1973. Se jubiló en 1994, pero permaneció activo como profesor emérito, impartiendo cursos de teoría de la información y análisis de señales.[2]

Dentro de las grandes contribuciones de Huffman a la ciencia de la computación se encuentran:

- El algoritmo de Huffman, fue de los primeros algoritmos de compresión de archivos sin pérdida de datos
- Contribuyo a transmitir sus conocimientos en el campo de la informática como profesor del Instituto de Tecnología de Massachusetts (MIT) y la Universidad de California en Santa Cruz (UCSC).
- Huffman contribuyó al estudio de estos sistemas, que son fundamentales en la teoría de la computación y en la construcción de autómatas y máquinas de estado finito

Debido a su gran influencia en las ciencias de la computación también se le fue premiado con diferentes distinciones entre las cuales destacan

- En 1999, recibió el premio Richard Hamming Medalla del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) en reconocimiento a sus excepcionales contribuciones a las ciencias de la información.[3]
- Medalla Louis E. Levy del Instituto Franklin por su tesis doctoral en los circuitos de conmutación secuencial Premio al Alumno Distinguido de la Universidad Estatal de Ohio.[3]
- Premio W. Wallace McDowell. Él era un recipiente fundador de la Pioneer Award Informática por la IEEE Computer Society.[3]
- Recibió un premio de oro del jubileo de Innovación Tecnológica de la Sociedad de la Teoría de la Información IEEE en 1998.[3]

Compresión con perdida y sin pérdida de datos

La compresión de datos es el proceso de reducir la cantidad de datos necesarios para almacenar o transmitir información. Dicho de otra forma, consiste en codificar la información utilizando menos bits que la representación original. Este proceso es similar a reducir un documento físico a un tamaño más pequeño y manejable sin perder su legibilidad.[4]

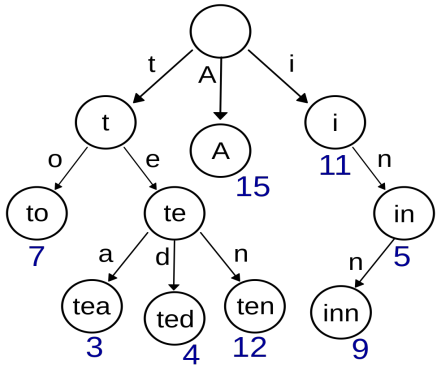
De la misma manera los métodos de compresión se dividen en 2 principalmente:

- **Algoritmos de compresión con perdida:** Se trata de un método de compresión de archivos en el cual para reducir el tamaño de ciertos archivos se procede a eliminar información considerada no importante o de la que el usuario no se vaya a percatar fácilmente, está perdida se puede ver reflejada como por ejemplo en disminución de la definición de una fotografía, alteraciones de la ecualización de un audio,etc..

- **Algoritmos de compresión sin perdida:** Este método de compresión de archivos tiene la particularidad de que en el proceso de compresión y descompresión de un archivo no se pierden datos, Algunos ejemplos incluyen, PNG para imágenes, y FLAC para audio, el requisito para entrar en esta categoría es que al descomprimir el archivo, obtengamos el archivo original

Trie

La estructura de datos Trie es una estructura de datos en forma de árbol que se utiliza para almacenar un conjunto dinámico de cadenas. Se utiliza comúnmente para la recuperación y el almacenamiento eficiente de claves en un gran conjunto de datos. La estructura admite operaciones como inserción, búsqueda y eliminación de claves, lo que la convierte en una herramienta valiosa en campos como la informática y la recuperación de información.[5]



Ejemplo de un trie tomado de Wikipedia.com

Biblioteca bitarray

Esta biblioteca proporciona un tipo de objeto que representa de manera eficiente una matriz de valores booleanos. Los bitarrays son tipos de secuencia y se comportan de forma muy parecida a las listas habituales. Ocho bits están representados por un byte en un bloque de memoria contiguo. El usuario puede seleccionar entre dos representaciones: little-endian y big-endian. Toda la funcionalidad se implementa en C. Se proporcionan métodos para acceder a la representación de la máquina, incluida la capacidad de importar y exportar buffers. Esto permite crear matrices de bits que se asignan a otros objetos, incluidos archivos asignados en memoria.[6] En síntesis lo que esto quiere decir es que Bitarray es una biblioteca que nos permite trabajr con arreglos de bits utilizando modelos de organización de memoria como por ejemplo: Little Endian y Big Endian.
Algunas de las funciones que implementaremos serán:

- **Bitarray:** La función bitarray() se utiliza para crear un objeto bitarray en Python
- **tofile:** Escriba la representación de bytes de bitarray en el objeto de archivo seleccionado.[6]
- **fromfile:** Extiende un bitarray con hasta n bytes leídos del objeto de archivo seleccionado (o cualquier otro flujo binario que admita un método .read(), por ejemplo, io.BytesIO). Cada byte leído agregará ocho bits a la matriz de bits. Cuando n se omite o es negativo, se leen todos los bytes hasta EOF. Cuando n no es negativo pero excede los datos disponibles, se genera EOFError (pero los datos disponibles aún se leen y se agregan).[6]

0.3. Solucion

0.4. Modulo: Compresor

Bibliotecas utilizadas

- bitarray:Esta biblioteca nos permite representar arreglos de bits
- collections.defaultdict: Esta biblioteca tiene la principal función de ser un diccionario, lo utilizamos para contar la frecuencia de los bits
- heaqp:Es popular debido a que es una implementación de cola de prioridad utilizada para árboles en python
- Sys: Es la biblioteca que nos permite utilizar argumentos dentro de nuestro programa

Class nodohufman:

init
Parámetros:

- frecuencia: La frecuencia del nodo
- byte: El byte asociado al nodo
- izquierda: El nodo hijo izquierdo
- derecha: El nodo hijo derecho (

Retorno: Nada porque es una clase como tal, además esta clase también está presente en el descompresor)
Descripción: El método init es el constructor de la clase NodoHuffman. Se utiliza para inicializar los atributos del nodo, incluyendo su frecuencia, byte asociado, y nodos hijos izquierdo y derecho

Lt
Parámetros:

- self: la clase en la que estamos
- otro: cualquier objeto con el que comparemos

Retorno: Un valor booleano que indica si el nodo actual es menor que otro nodo
Descripción: El método lt es un operador mágico de python que hacia la comparación menor que «"

Contar frecuencia Bits
Parámetros:

- La ruta del archivo del que se quiere contar la frecuencia de bytes

Retorno: Un diccionario que contiene la frecuencia de cada byte en el archivo
Descripción: Esta función cuenta la frecuencia de cada byte en un archivo y devuelve un diccionario con los resultados.

Construir árbol de huffman
Parametros:

- Un diccionario que contiene la frecuencia de cada byte en el archivo (defaultdict)

Retorno: El nodo raíz del árbol de Huffman (NodoHuffman)
Descripción: Esta función construye un árbol de Huffman a partir de las frecuencias de bytes en un archivo. Utiliza una cola de prioridad para combinar los nodos con menor frecuencia hasta que solo quede un nodo, que es la raíz del árbol de Huffman.

Generar codigos Huffman
Parámetros:

- Nodo: El nodo actual
- Prefijo: Un string cualquiera que represente el nodo
- libro códigos: Un diccionario que almacena los códigos de cada byte

Retorno:El diccionario que contiene los códigos Huffman para cada byte
Descripcion: Esta función genera los códigos Huffman para cada byte en el árbol de Huffman. Recorre el árbol de Huffman de manera recursiva y asigna un código Huffman a cada byte según su ruta en el árbol.

Comprimir archivo
Parámetros:

- ruta archivo:la ruta del archivo que vamos a comprimir
- Códigos: El diccionario que contiene los códigos Huffman para cada byte

Retorno: Los datos comprimidos en forma de un bitarray
Descripción: comprime un archivo utilizando los códigos Huffman generados anteriormente. Lee el archivo, convierte los bytes en un arreglo de bits, y luego reemplaza cada byte con su código Huffman correspondiente.

Guardar archivo
Parámetros:

- ruta original: la ruta del archivo en forma de string
- datos comprimidos: los datos que fueron comprimidos en el bitarray anteriormente

Retorno: archivo con extension .huff
Descripción: Esta función guarda el archivo comprimido en un nuevo archivo con la misma ruta que el archivo original, pero con una extensión .huff adicional.

Guardar Tabla huffman
Parámetros:

- Ruta archivo: La ruta del archivo forma string
- Codigos:el diccionario de códigos huffman de cada byte

Retorno: Un archivo con extensión .table
Descripción: En el siguiente sector del programa se genera un archivo de texto con la extensión .table en el cual se especifica cada valor ASCII con su respectivo código huffman

Altura de árbol
Parámetros:

- nodo: El nodo del árbol Huffman

Retorno: dato tipo INT que representa la altura del árbol
Descripción: La función utiliza una llamada recursiva para calcular la altura de los subárboles izquierdo y derecho y devuelve el máximo de los dos más uno por la definición de altura

Anchura de árbol
Parámetros:

- nodo: El nodo del árbol Huffman

Retorno: dato tipo INT que representa la anchura del árbol
Descripción: La función utiliza una llamada recursiva para calcular la anchura de los subárboles izquierdo y derecho y devuelve la suma de los dos más uno

contar nodos árbol
Parámetros:

- nodo: El nodo del árbol Huffman

Retorno: dato tipo INT que representa la cantidad de nodos en el árbol
Descripción: Esta función cuenta la cantidad de nodos en el árbol Huffman dado un nodo. La función utiliza una llamada recursiva para contar los nodos en los subárboles izquierdo y derecho y devuelve la suma de los dos más uno

Guardar estadísticas huffman
Parámetros:

- ruta archivo: La ruta del archivo origina
- frecuencias: Las frecuencias de los caracteres
- árbol huffman: El árbol de Huffman construido a partir del archivo

Retorno: Asignar valores a variables Descripción: Esta función guarda las estadísticas del árbol de Huffman en un archivo de texto. La función calcula la altura, anchura y cantidad de nodos del árbol de Huffman utilizando las funciones usando las funciones de altura, anchura y contar nodos mencionadas anteriormente, para escribirlas en un archivo de texto con extension .stats

main
Parámetros:

- nombre del archivo:Un argumento que indica el archivo sobre el cual vamos a realizar la compresion

Retorno: No retorna un dato pero desencadena todo el proceso de compresion
Descripción: es la función principal que realiza la compresión de un archivo utilizando el programa. Primero, cuenta las frecuencias de bytes en el archivo, construye un árbol de Huffman, y genera códigos Huffman. Luego, comprime el archivo utilizando los códigos Huffman y guarda el resultado en un archivo. También guarda la tabla de códigos Huffman y las estadísticas del árbol de Huffman en archivos separados.

0.5. Modulo:Descompresor

Leer tabla huffman
Parámetros:

- ruta tabla: La ubicación de la tabla que contiene los caracteres y sus valores

Retorno: Un diccionario llamado tabla de valores
Descripción: la función lee una tabla de códigos Huffman desde un archivo de texto .table y devuelve un diccionario con los códigos Huffman asociados a cada byte

Reconstruir árbol huffman
Parámetros:

- Tabla de códigos(nuestro archivo .table)

Retorno: La raíz del árbol ya reconstruido
Descripción: toma una tabla de códigos de Huffman, donde las claves son bytes y los valores son las cadenas de bits correspondientes, y construye un árbol de Huffman a partir de esta tabla

descomprimir archivo
Parámetros:

- Ruta comprimida: la ruta del archivo que se va a descomprimir
- árbol huffman: la raíz del árbol que vamos a descomprimir
- Ruta salida: la ruta en la cual vamos a descomprimir el archivo
- derecha: El nodo hijo derecho (

Retorno: Escribir el archivo descomprimido en la ruta especificada
Descripción: lee un archivo comprimido, usa un árbol de Huffman para interpretar los datos comprimidos bit a bit, reconstruye el contenido original y lo escribe en un archivo de salida.

Bloque final de argumentos
Parámetros:

- Argumento 1(Archivo comprimido)
- Argumento 2(Tabla de valores huffman)
- Argumento 3(Archivo sobre el que vamos a escribir)

Retorno: Nada
Descripción: En general se encargan de procesar en un terminal el nombre de varios archivos que estén en la misma ubicación que el descompresor y realiza todo el proceso de descompresión para imprimir el archivo original en el archivo deseado.

0.6. Problemáticas y limitaciones

En la vida hay que ser sincero, he de admitir que realice varias pruebas con el compresor y descompresor y en la gran mayoria de casos devuelve un archivo idéntico al original, pero hubo un caso en el que no fue así, el archivo fue una cadena de texto que decia"Hola, este es un archivo de prueba del algoritmo de huffman.z devolvió después de todo el proceso "Hola, este es un archivo de prueba del algoritmo de huffman.a", en resumen le agrego un .ª.ªl final de la cadena de texto cuando originalmente no estaba. De igual manera cabe resaltar que solo en este caso me ocurrió.

0.7. Conlcusiones

En python existe una librería llamada collections que contiene la clase de diccionario llamado "Defaultdict"que a diferencia del diccionario común, asigna un valor un valor predeterminado a un elemento si no existe en el mismo.

Las clases son una estructura funcionalidad en la programación que nos permite crear un conjunto de datos y funciones que giren entorno a un objeto. Por otro lado. los métodos especiales o mágicos en python,permiten crear clases en el código que se comporten de la forma que nosotros especifiquemos.

Por lo general, los algoritmos de compresión más populares en la actualidad son los "sin perdida", muestra de ello sería por ejemplo los archivos RAR, zip, 7z, FLAC, que en temas de compresión son los más populares en la actualidad

0.8. Bibliografia

[1]Alvinashcraft, “Compresión y descompresión de archivos - Win32 apps,” Microsoft Learn. https://learn.microsoft.com/es-es/windows/win32/fileio/file-compression-and-decompression

[2]J. Zhu-Mai, “David A. Huffman,” IEEE Computer Society, Jun. 05, 2018. https://www.computer.org/profiles/david-huffman

[3]EcuRed, “David Huffman - ECURed.” https://www.ecured.cu/David$_Huffman$

[4]$L.Ballejos, \?`Quéeslacompresiónde datos?-NinjaOne,"NinjaOne, Feb$,13, 2024.$https : //www.ninjaone.com/es/it-hub/it-service-management/que-es-la-compresion-de-datos/$

[5]$GeeksforGeeks, \Triedatastructureinsertandsearch,"GeeksforGeeks, Jun$,07, 2024.$https : //www.geeksforgeeks.org/trie-insert-and-search/$

[6]$\bitarray,"PyPI.https : //pypi.org/project/bitarray/$