Escuela de Ingeniería en Computación
Taller de programación
Grupo 21

Compresión con árboles de Huffman

Justin Morris
Carné: 2024143181
Jose Mario Castro
Carné: 2022437423

Profesor: Alex Brenes Brenes

17 de Junio de 2024

0.1. Introducción

Este proyecto hablará principalmente acerca de la compresión y descompresión, pero aplicado en este caso haciendo uso de otro tema del que se hablará bastante, que serían los árboles de Huffman, temas muy importantes en la computación y que son claves para diferentes procesos que realizamos día a día en nuestra sociedad actual.
Para empezar, hay que definir los procesos de los que se hablarán en este proyecto, la compresión, o en este caso el çomprimir"se define según la RAE(Real Academia Española) se define como [1].ºprimir, apretar, estrechar, reducir a menor volumen", en computación, se aplica generalmente la tercera palabra, reducir, y se aplica reduciendo el tamaño de un archivo, por el contrario, descompresión, o "descomprimir", según la Real Academia Española(RAE) se define cómo [2].ªminorar o anular la compresión en un cuerpo o en un espacio cerrado..en otras palabras, es un método de revertir la compresión.
finalmente, el árbol de Huffman, es un método de compresión "sin perdidas"(esta definición se explicará más adelante) [3]que se usa para la compresión y encriptación de datos mediante el análisis de frecuencia de aparición de caracteres.
Ya habiendo definido estas palabras, podemos pasar a explicar el proyecto, que consiste en dos funciones principales, un compresor y un descompresor, el compresor recibe 1 archivo de texto y retorna 3 archivo diferentes, un archivo.huff en el que se encuentra el código binario de huffman ya procesado, un archivo .table con los prefijos de cada código, y un tercer archivo .stats que contiene, la altura del árbol de Huffman desarrollado, su anchura, la cantidad de nodos por nivel, y la frecuencia de cada elemento, y el descompresor, recibe los dos primeros archivos generados por el compresor(.huff y .table) y un tercer archivo .txt, en el que se descomprimirá el resultado de comparar el .huff con los prefijos del .table.

0.2. Desarrollo

Como mencionamos anteriormente, el problema consiste en desarrollar dos funciones, una de compresión y otra de descompresión, todo esto usando el algoritmo de Huffman, aplicando el árbol de Huffman, por lo que hay que adaptar primero el funcionamiento de este árbol a código, y después realizar una función que cree los 3 archivos solicitados, y luego hacer otra función aparte que lea el documento con el código de Huffman y lo desencripte con la tabla de prefijos que está en el documento .table.

Ahora, para profundizar más acerca de los árboles de Huffman, es necesario hablar de la persona que creó el algoritmo, David A. Huffman, este nació el 9 de agosto de 1925 en Ohio, Hufmman, que se graduó del colegio a la edad de 15 años, decía que el "pensaba que era mayormente valorado por su mente"[4], él entro a la Universidad Estatal de Ohio el año siguiente, y a los 19 se graduó de su primera ingeniería y se convirtió en un oficial de la Navia en el Pacifico, tuvo muchos momentos duros ahí, porque su superior le tenía despreció, le daba tareas de más, con que tuvieran que ver un mínimo con el radar o el sonar, sólo sufrió una herida de guerra, después de dos años de servicio lo dejaron irse, y este fue a sacar una maestría en ingeniería eléctrica, tiempo después, para fortuna de él, lo aceptaron en MIT, donde pudo ejercer, mientras estaba aquí, él, al haber fallado un examen importante y estando en verano, él se apuntó, (por accidente) a un curso, el cual le introdujo a una de las partes más importantes de su vida, que sería lo digital y discreto, todo este interés, desembocó en la realización del procedimiento del código de Huffman.

Él tuvo 3 hijos con su exesposa, y 2 hijastros con su esposa, El murió en un hospital local, un jueves, 10 de octubre, después de 10 meses de combatir el cáncer, a los 74 años [5].

Ahora, cómo mencionamos antes, hay que explicar que es la compresión de archivos sin perdidas, que es en donde se encuentra el árbol de huffman y con pérdidas, que sería la contraparte, para empezar, la compresión con pérdidas, o lossy, en palabras de adobe es ün algoritmo mayormente usado en archivos que puedan permitirse perder algunos datos, o que necesiten liberar espacio"[6], en otras palabras son un tipo de dato que se usa en situaciones en las que no poder retornar el archivo a exactamente lo que era antes, no importa, o no es necesario. por el lado contrario, la compresión sin perdidas es cuando los datos del archivo no son dañados o eliminados en la compresión, y pueden ser reversibles, en otras, palabras, la compresión sin perdidas es el tipo de compresión que se puede descomprimir.

Trie: los Tries son árboles de búsqueda, donde las claves de búsqueda son interpretadas partidas en porciones de claves. Estas porciones son utilizadas para armar la estructura de datos de búsqueda [7] Estos son grado m ¿= 2, y en estos la ramificación a cualquier nivel es determinada no por el valor completo de la clave, sino por una porción de ella.

Bitarray: La biblioteca Bitarray otorga un tipo de objeto que representa de manera eficiente un vector de booleanos [8], en otras palabras, una manera de representar "bitarrays", los cuales son un tipo de secuencia que actúa de manera

similar a una lista, en esta biblioteca, se encuentran las 3 funciones usadas, que serían:

Bitarray(): esta función es la que se usa para usar los bitarrays, en esta se ponen los bitarrays que se quieren usar, se puede utilizar para interpretar información cómo bitarray y hacer posible que el resto de las funciones de esta biblioteca puedan usarse.

.tofile: esta función se usa para escribir la representación de los bytes en un archivo

.fromfile: esta función es similar a read, en otras palabras, esta función se encarga de leer los bits presentes en un documento.

Funciones Usadas:

Nombre: contenido archivo

Módulo: compresión

Parámetros: filename: parametro posicional que solicita el nombre del archivo

Retorno: texto

Resumen: retorna el texto dentro del documento leído

Nombre: ascii

Módulo: compresión

Parámetros: text: posicional, solicita el texto .

Retorno: lista de frecuencia de caracteres ascii(f).

Resumen: recibe texto, y retorna una lista con todos los valores ascii, representada con ceros, cada vez que detecta una letra, aumenta el contador de dicha tecla en uno.

Nombre: frequencia caracteres

Módulo: compresión

Parámetros: f: posicional, solicita el total de apariciones de cada elemento que haya en la lista

Retorno: charfreq: conjunto de tuplas con el elemento y su respectiva cantidad de apariciones

Resumen: recibe la lista generada en ascii, y construye tuplas con el elemento y su respectiva frecuencia

Nombre: huffmantree

Módulo: compresión

Parámetros: charfreq: posicional, brindar la frecuencia para poder construir el árbol

Retorno: charfreq[0], en otras palabras, la lista original pero adaptada al árbol de huffman

Resumen: va sumando los valores con frecuencia de menor valor, usando el modelo del algoritmo de huffman, creando así el árbol de Hufman.

Nombre: codigo

Módulo: compresión

Parámetros: T: posicional, recibe un árbol, name: posicional, recibe el nombre del documento, camino: con valor predeterminado, lista vacía para acumular los caminos, res: valor predeterminado, una lista en la que se acumulan los prefijos

Retorno: retorna res, una lista vacía en la que se guardan los prefijos, y genera un documento .table

Resumen: recibe el árbol de huffman y el nombre del archivo, a partir de ahí genera una lista con el prefijo de cada uno de los elementos de presentes

Nombre: compresión

Módulo: compresión

Parámetros: text: posicional, recibe el texto que se quiere codificar, caminos: posicional, recibe los caminos de los prefijos name: posicional, recibe el nombre original del archivo para poder generar el .huff

Retorno: retorna el código de huffman, en un documento cuyo nombre es, el nombre original del archivo con .huff

Resumen: recibe la palabra original y los prefijos, y a partir de ahí construye el código en un documento.huff

Nombre: altura árbol

Módulo: compresión

Parámetros: árbol: posicional, recibe el árbol al que le va a sacar la altura

Retorno: la altura del árbol, o 0 en caso de no haber nodos

Resumen: retorna la altura del árbol, esta función esta específicamente construida para recibir este tipo de árboles de huffman

Nombre: anchura árbol

Módulo: compresión

Parámetros: árbol: posicional, recibe el árbol del que se va a determinar la anchura

Retorno: "max anchura", un contador en el que se acumula la cantidad de nodos por nivel, o 0 en caso de no haber nodos

Resumen: va acumulando la cantidad de nodos en "max anchura" hasta encontrar el nivel con mayor cantidad, retornando este valor

Nombre: nodos por nivel

Módulo: compresión

Parámetros: árbol. posicional, solicita el árbol del que se va a sacar los nodos por nivel

Retorno: nodos nivel, una lista en la que se encuentra la cantidad de nodos que tiene cada nivel del arbol o una lista vacia en caso de que no haya nodos

Resumen: cuenta los nodos que hay por nivel y los añade a la lista de nodos nivel hasta que ya no hay niveles

Nombre: escribir resultado en archivo

Módulo: compresión

Parámetros:filename: posicional, recibe el nombre del archivo original para usarlo en el nuevo text: recibe texto, funciones: recibe una lista con las funciones a poner en el documento valores: el valor a ser evaluado por las funciones

Retorno: un documento.stats con el nombre que esté en filename, con el valor dado en texto, con el resultado de las funciones dadas

Resumen: escribe los resultados de comparar las funciones de "funcionesçon los elementos que estén en "valoresz los imprime en un documento .stats que tiene el nombre del documento original

Nombre: main

Módulo: compresión

Parámetros: no tiene

Retorno: el resultado de la compresión

Resumen: esta función es para conectar todo el código y que el programa funcione

Nombre: interpretación huff

Módulo: descompresión

Parámetros: huff file: posicional, recibe el documento .huff en donde se encuentra el código huffman

Retorno: Text, lee el archivo y lo pasa a texto str

Resumen: interpreta un archivo .huff, que está en binario, y lo pasa a texto.

Nombre: interpretacion table

Módulo: descompresión

Parámetros: table file: posicional, un archivo.table con una tabla de prefijos

Retorno: codes, una lista en donde se evalúan los prefijos de la tabla cómo literales

Resumen: recibe un .table, y lo convierte a literales para poder usarlos en la decodificación

Nombre: descompresión

Módulo: descompresión

Parámetros: text: posicional, recibe un código binario para pasarlo a str. codes: posicional tabla de prefijos interpretada como un literal. new file: posicional, documento en donde se pondrá el resultado

Retorno: se retorna el new file con el código de text decodificado

Resumen: se usa la tabla de prefijos para usarla en la decodificación del código hecho

Nombre: main

Módulo: descompresión

Parámetros: input del usuario

Retorno: archivo.txt con el código decodificado

Resumen: función para conectar las demás funciones

0.3. Problemáticas y Limitaciones

La mayor problemática que tuvimos con el proyecto es que algunas veces se marcan 0 de más al descomprimir el archivo, ocasionando que el problema en algunas pruebas dé el resultado con letras de más, más allá de eso, no tuvimos alguna otra problemática notada por nosotros mismos.

0.4. Conclusiones

En este proyecto se aprendió a interpretar el cómo funcionan los bitarrays, cómo usar las biblioteca de la propia bitarray, el cómo se usaban sus funciones, sus limitaciones, cómo por ejemplo que sólo se pueden usar documentos que funcionen en bits, o con terminología .bin para poder usar, por ejemplo, tofile y fromfile.

También, se aprendió a representar de manera correcta el algoritmo de huffman, usarlo para crear su árbol, y usarlo en archivos para reducir su tamaño, a causa de esto, también aprendimos a realizar codificación y decodificación básica, por lo menos, en lo que respecta a símbolos de ASCII.
En síntesis, y para concluír, los árboles de Huffman son una gran herramienta que nos permite codificar a nivel básico, y bitarray es una librería ideal para trabajar al nivel al que se trabaja con los árboles de Huffman, y nos ayuda a entender de manera más sencilla el funcionamiento de los bits.

0.5. Bibliografía

[1] Real Academia Española, Çomprimir,"Diccionario de la lengua española, 22ª ed., Madrid, Spain, 2001. [Online]. Available: https://www.rae.es/drae2001/comprimir. [Accessed: Jun. 15, 2024].

[2] Real Academia Española, "descomprimir,"Diccionario de la lengua española, 22ª ed., Madrid, Spain, 2001. [Online]. Available: https://dle.rae.es/descomprimir. [Accessed: Jun. 16, 2024]

[3]Xakataciencia(A. Jimenez),."Algoritmo de Huffman", 2006. [online]. Available:https://www.xatakaciencia.com de-huffman.[accessed:jun.15,2024]

[4] "David Huffman Biography,"HuffmanCoding.com. [Online]. Available: https://www.huffmancoding.com/my uncle/david-bio. [Accessed: June 17, 2024].

[5]J. burns ."Eminent UCSC computer scientist David Huffman dies at age 74ÜC SANTA CRUZ, para, october 8, 1999.[online], Available: https://acortar.link/VGm7us
[6] Adobe, "Lossy vs. Lossless Compression,."Adobe Creative Cloud. [Online]. Available: https://www.adobe.com/uk/creativecloud/photography/discover/lossy-vs-lossless.html. [Accessed: June 17, 2024].

[7] F. Engels,."Estructura de Datos : Arboles Trie"Facultad de Ciencias Exactas, Ingeniería y Agrimensura (FCEIA), Universidad Nacional de Rosario.[Online].Available:https://n9.cl/op89go[A June 17, 2024]

[8] "bitarray - PyPI,"Python Package Index (PyPI). [Online]. Available: https://pypi.org/project/bitarray/. [Accessed: June 17, 2024].

Enlace al GitHub:
https://github.com/Doubl3J/Proyecto-3