Proyecto Programado: Compresor de imágenes - Paolo Induni

1. Resumen Ejecutivo

El proyecto consiste en crear un compresor y descompresor de archivos basados en el algoritmo de Huffman, un algoritmo el cual genera arboles de este mismo nombre con el tal de poder disminuir el tamaño de un archivo y compactarlo, sin embargo este mismo algoritmo genera un alfabeto y una distribucion de probabilidades los cuales se usan para volver a generar el archivo original (descomprimirlo).

Para este proyecto lo que se solicita es un compresor y descompresor basados en la codificación o algoritmo de Huffman, pero primero para esto tambien es necesario generar una tabla en un achivo de texto que tenga la altura del árbol, anchura del árbol, cantidad de nodos por nivel y la tabla de frecuencias original (sólo aquellos bytes presentes con frecuencia mayor a cero), el compresor debe ser capaz de comprimir cualquier formato de archivo y el tamaño de este archivo comprimido debe ser menor a el archivo original, adicionalmente, el archivo comprimido debe poder ser descomprimido con otro programa basado en el mismo algoritmo, y este archivo descomprimido debe volver a ser el original y ser funcional.

Para poder realizar este proyecto se necesitan utilizar varias funciones para trabajar con árboles, funciones para recrear o crear árboles a partir de un formato específico, funciones para calcular el tamaño del árbol y funciones para poder trabajar con bits y bytes.

Lo que se hizo para poder realizar los procedimientos anteriormente mencionados fue, de forma general, crear dos distintos programas, el primero siendo el compresor el cual lee un archivo encontrado en la misma carpeta que este, y luego se encarga de comprimirlo, generando asi el archivo huff (siendo el archivo comprimido) y el archivo ".table.[el] cual posee las estadisticas de el archivo, en cuanto al segundo código, el descompresor, este lee el archivo huff y se encarga de convertirlo a su forma original usando la tabla de frecuencias

En este proyecto se han realizado varias pruebas con el tal de poder calcular y diferenciar los tiempos en los que el programa comprime archivos de un cierto formato y tamaño, además se han hecho pruebas de descompresión de los archivos comprimidos con el tal de poder verificar y asegurar el funcionamiento tanto del programa como de los archivos comprimidos, adicional a esto, tambien se realizaron estas pruebas con el tal de poder observar las diferencias de tiempo entre archivo y las diferencias de tiempo entre programa, es decir entre compresión y descompresión.

2. Introducción

Este proyecto consiste en la creación de dos códigos que trabajan en conjunto. El primer código es un comprensor de archivos cuya función principal es crear una copia de un documento, o material multimedia, que sea más compacta y por lo tanto que ocupe un espacio de memoria reducico en comparación al original. La otra parte del código es el complemento necesario, es decir, un descomprensor que sea capaz de leer y restaurar correctamente el archivo original antes de que fuese comprimido.

Para lograr este objetivos nos basamos en los principios de lo árboles de Huffman, junto a la lectura de los bites y bytes que comforman el archivo original.

Para este proyecto tenemos esperado realizar los programas de tal forma que estos puedan comprimir un documento en formato txt como la "Santa Biblia."[en] pocos segundos, y descomprimirla en un tiempo parecido, sin embargo para archivos de mayor tamaño tenemos esperado que el compresor sea más rápido realizando su función que el descompresor.

Lo que se espera del proyecto es la creación de un compresor (y descompresor) de archivos que no presente pérdidas de información al ser utilizado, esto último se puede considerar como un objetivo minimo. Además de eso, otro objetivo consiste en que el compresor cumpla justamente su función y los archivos comprimidos sí ocupen un menor espacio en la memoria de los dispositivos electrónicos.

Este docuemnto cuenta con 8 secciones distintas, siendo la primera el Resumen Ejecutivo"donde se habla sobre lo que se debe realizar y programar, además de el algoritmo en específico y de como se solucionará o realizará el katamino y mencion de los casos de prueba, la segunda sección sería la Introducción, donde se habla sobre de que es el proyecto y de una pequeña introduccion o resumen al documento, la tercera sección es el "Marco Teórico"donde se habla sobre lo que se necesitó para realizar el programa para hacer el proyecto y de la información respectiva de este, la cuarta sección es la "Descripción de la solución"donde se habla de como se consiguió realizar el programa y de su funcionamiento en si, la quinta sección es los Resultados de pruebas"donde se habla y muestra sobre ciertas pruebas realizadas y el como el programa reaccionó ante estas, la sexta sección son las Çonclusiones"donde se habla sobre lo que se logró y sobre lo que se determino de el programa al final, la sétima sección habla sobre los .ªprendizajes"donde se menciona lo que se ha logrado aprender o mejorar gracias a este proyecto, y finalmente la octava y última sección la "Bibliografía"donde se ponen las referencias de donde se obtuvo la información tanto para realizar el proyecto como para el marco teórico.

3. Marco Teórico

3.1. Python

El lenguaje de programación Python fue creado por un programador holandés Guido van Rossum a finales de los 80s e inicios de los 90s, en un inicio se concibe con el fin de manejar exepciones y tener interfaces con Amoeba, siendo sucesor del lenguaje ABC. El día 16 de octubre del 2000 se lanza Python 2.0, esta actualización le trajo al lenguaje nuevas características y completo soporte a Unicode, pero sin duda el mayor avance que tuvo fue que Python empezó a ser verdaderamente desarrollado por la comunidad bajo la dirección de Guido. El 03 de diciembre de 2008 sale de manera oficial la version 3.0 de Python, siendo esta una versión mayor e incompatible con las versiones anteriores

3.2. Pycharm

Pycharm es un entorno de desarrollo integrado multiplataforma el cual se utiliza para desarrollar en el lenguaje de programacion Python. Este IDE es desarrollado por la empresa JetBrains, tiene dos versiones, la Community que es gratuita y orientada en parte a la educación, y en si al desarrollo puro en Python, la otra versión seria la Professional la cual incluye más características.

3.3. Codificación Huffman

La codificación Huffman fue un método propuesto en 1952 por David Huffman, esta codificación se encarga de crear una tabla de códigos de longitud variable para así codificar cada uno de los símbolos de la fuente. La creación de la tabla de codificación y la longitud de los códigos son dependientes de la probabilidad de cada símbolo.

El alfabeto y la distribución de probabilidades de la información codificada es almacenada y/o enviada a el decodificador para que este se encargue de construir el árbol de Huffman equivalente y asi se logre obtener el archivo original.

4. Descripción de la solución

A manera general, el primer código del proyecto que se encarga de la compresión recibe como input un archivo, el tipo de archivo no es muy importante puesto que lo que va a leer tal cual van a ser los bytes del mismo permitiendo así una mayor cobertura. Posterior a eso se inicializa una lista con 256 espacios de longitud, esto debido a la naturaleza de los bytes, en esta lista se va a asignar la frecuencia de cada byte, añadiendo el valor de 1 a la posición de la lista cada vez que se repita dentro del documento.

Una vez que se obtiene la lista de frecuencias se recorre la misma y se crea un árbol en base a esta, tal que a cada posición en el árbol se le asigna su

byte correspondiente junto con el subarbol que vaya asociado y finalmente se acomoda la lista de frecuencias según su valor.

Ahora de este árbol se va a partir para crear un árbol de Huffman que es donde se va a realizar el proceso de compresión del árbol. Para esto se crear un ciclo 'while' que va finalizar hasta que la longitud del árbol sea de 1. Dentro de este 'while' se inicializa una lista en la que se va a almacenar la suma de las frecuencias, junto con esto se sacan el hijo izquierdo y el hijo derecho a través de un: ".pop(0)". Con todo esto se procede a realizar la suma de la frecuencia del hijo izquierdo junto con la frecuencia del hijo derecho y se agruegan a la lista inicializada anteriormente todos los anteriores. Se agrega la lista de frecuencias al final del árbol de Huffman y se le hace un: ".sort()"para darle orden.

Cuando finalice el ciclo se obtendría un arbol donde las raíces son bytes y frecuencias de estos, los nodos son la suma de las frecuencias.

Apenas se consiga el árbol de Huffman se necesita calcular las rutas hacia las hojas para la creación de la tabla de direcciones, donde un giro a la izquierda se representa como un '0' y un giro a la derecha como un '1'. Este proceso se va a ser recursivamente y se va a obtener una ruta binario en formato 'string' cuando llegue a su fin.

Cuando el código termina de obtener las rutas hacia las hojas se va a crear una nueva lista en la que se va a guardar el camino binario al byte en la posición que es igual del byte, por ejemplo: un byte representado como "68"tendrá su ruta binario en la posición 68 de esta nueva lista.

Posteriormente con las funciones vistas en clases se consigue sacar la altura, anchura y los nodos por nivel, y con eso concluiría la parte del compresor.

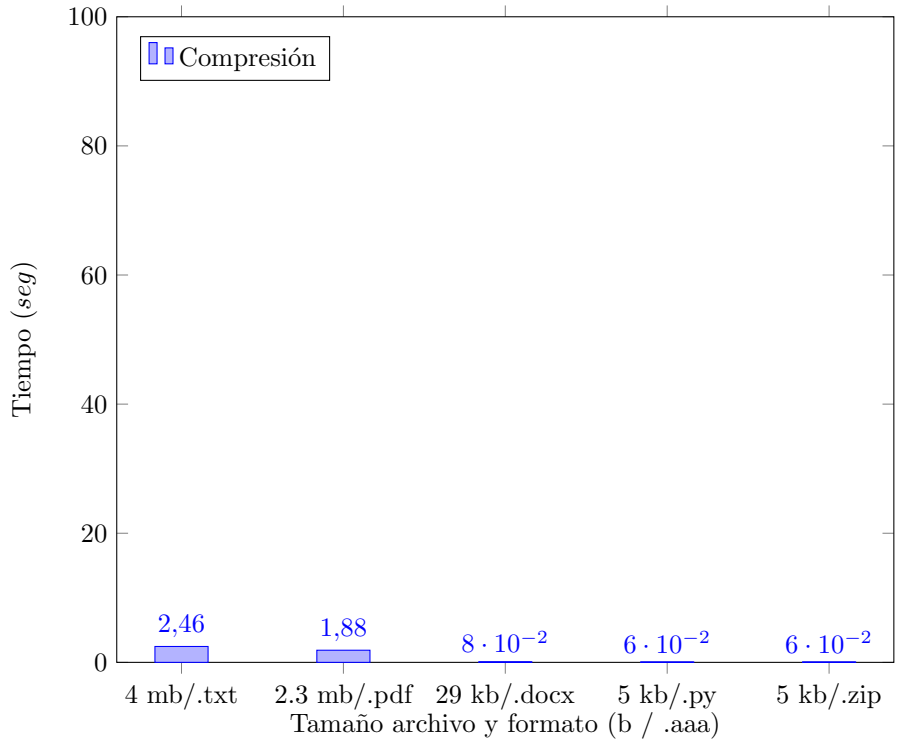
La segunda parte del código es el descompresor de archivos, para esto va a ser necesario el '.huff' y el '.table' que son el archivo comprimido y las lista de direcciones respectivamente obtenidos en el compresor.

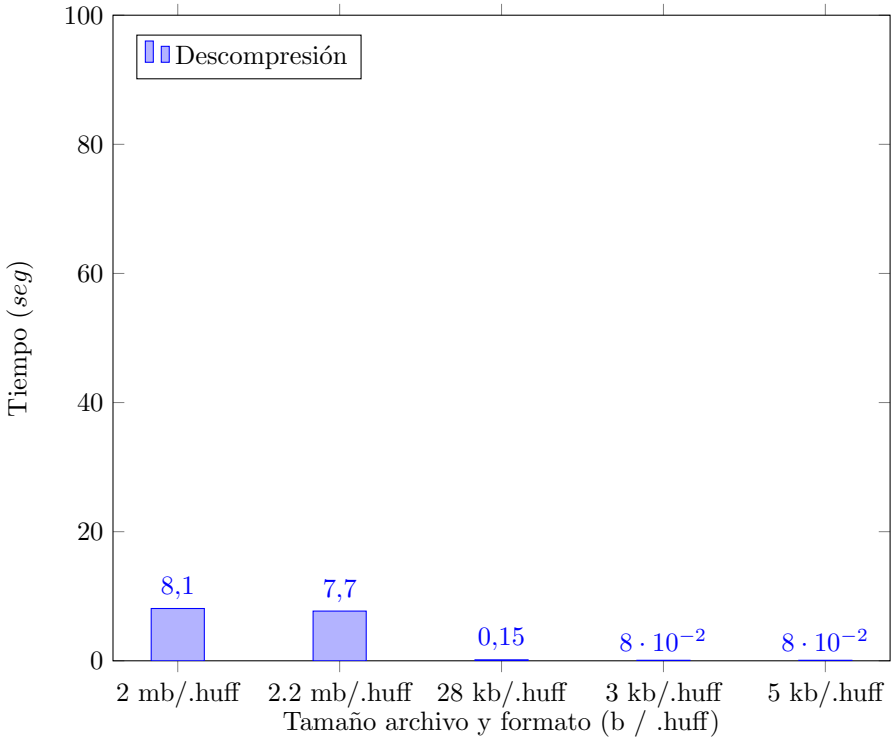
Lo primero que se hace en el lado del descompresor es hacer la lectura de los de la lista de rutas binarios através de la linea: "tabla = open("nombre del archivo.table", r").read().split("") ". Después de eso se procede a reconstruir el árbol con ayuda de la función reconstruir"que funciona de manera recursiva y va rearmando el árbol tanto en la rama izquierda como derecha.

Cuando se consiga el árbol reconstruido por completo otra vez se abre el archivo '.huff', que es el archivo comprimido, para hacer la lectura de los bytes que se va a guardar en la variable 'huff', a esta variable se le va a quitar el último valor que posee y se va a asignar en la variable 'extra'. Posterior a esto se va a llamar a la función recorrer"que va ser la encargada de descomprimir el archivo esto se logra recorriendo todos los bytes en 'huff'y en 'extra'. Finalemente la función va a retornar un archivo de nombre .ªrchivoComprimidoz con eso finaliza el proceso de descompresión.

5. Resultados de pruebas

Se realizaron pruebas con archivos de distintos tamaños y formatos con el tal de ver que tan eficiente y capaz era el programa, de lo cual se muestra el resultado en las siguientes gráficas





5.1. Aclaración

Cabe mencionar, que de las pruebas anteriores, los archivos comprimidos concuerdan con los descomprimidos, siendo el tamaño de el archivo a comprimir (es decir el tamaño original del archivo) el mencionado en la primera gráfica y el tamaño de el archivo ya comprimido mencionado en la segunda gráfica, para saber cual archivo corresponde con cual entre las gráficas, basta con posicionarse en la misma posición para ambas gráficas.

5.2. Hardware

A continuación se especifica el hardware con el que se ejecutaron las pruebas para todos los modos, el programa fue desarrollado con el IDE Pycharm y se corrieron las pruebas desde la consola CMD.

- Procesador i-3 9100f de Intel
- 16 Gigabytes de RAM a 2666 Ghz
- Sistema operativo Windows 10 en 64 bits de Microsoft

6. Conclusiones

Basandonos en todo lo anterior, y principalmente en los resultados de las pruebas realizadas se puede concluir y demostrar que el compresor es más veloz realizando su función que descompresor, puesto que en las pruebas realizadas se puede observar la diferencia de tiempos que hay entre estas dos funciones, si bien no es una diferencia muy grande, si es notoria, otro detalle que se puede observar es que la compresión en base al algoritmo de Huffman no es precisamente una reducción de tamaño muy considerable, sin embargo si se realiza una reducción de tamaño al fin y al cabo.

7. Bibliografía

Ramirez, E.(2017). Fundamentos de Programación. Sin editorial

Centro de Información y Gestión Tecnológica (Holguín, Cuba), Ciencias Holguín.

[2] Y. Cedeño Ocampo. Codificación Huffman Mejorada con Cifrado Datos, Vol. 2. Universidad Nacional, 2016.