Hufco (HUFman COmpressor)

Descripción

Este documento es la memoria del trabajo de fin de grado sobre un programa compresor de textos en castellano usando el algoritmo de compresión de Huffman.

Resumen

El objetivo de este trabajo de fin de grado es explicar la elaboración de un programa compresor y descompresor de textos en castellano, o archivos de texto plano por medio del algoritmo de codificación de Huffman, basado en la frecuencia de uso de los caracteres para crear códigos de longitud variable, demostrando así la capacidad de compresión de éste frente al código ASCII en el cual todas los códigos de codificación de caracteres tienen longitud fija.

Para llegar a este programa, se demostrará el recorrido para llegar al algoritmo de codificación de Huffman, su validez, y eficiencia mediante las explicaciones y demostraciones matemáticas que serán vistas más adelante.

En el documento se estructurará en 4 bloques principales, el primero un tema introductorio sobre el por qué de utilizar el algoritmo de Huffman para codificación de texto; el segundo, la teoría matemática para llegar y demostrar el algoritmo de Huffman, pasando por los siguientes temas: Introducción a códigos instantáneos, teorema de Kraft-McMillan [CONTINUAR] . El segundo bloque se centrará en el programa elaborado y constará de las siguientes partes: Introducción, esquemas, funcionalidades básicas, desarrollo del programa y futuras mejoras. Para finalizar, en el tercer tema se expondrán conclusiones finales.

Abstract

[en inglés]

Palabras clave: código, código de Huffman, código óptimo, código instantáneo, compresor, descompresor.

Introducción

Aplicaciones

Un compresor de archivos digitales de texto supone una gran ventaja que puede ser utilizada en muchos campos:
Libros en txt: Una biblioteca de libros virtual que podría llegar a ocupar la mitad de espacio de almacenamiento.
Código fuente: Los archivos de código fuente, también son texto plano, por lo que pueden ser comprimidos. Un ejemplo de uso es un servidor de código fuente como Github tiene millones de archivos de código fuente.
Archivos de registro (logs): Como son por lo ejemplo los archivos registro de los servidores web que indican todos los cambios, peticiones, etc

Cualquier compresión indica una menor necesidad de espacio de almacenamiento, lo cual se traduce directamente como un ahorro económico al no tener que comprar y mantener tanto hardware de almacenamiento.

Descripción metodológica

Introducción a códigos instantáneos

CÓDIGO INSTANTÁNEO: Un código instantáneo es aquel en el que una palabra-código no forma parte del comienzo de otra, esto se ve muy claro por ejemplo con los números de teléfono, ya que si por ejemplo tuviéramos los números 959127 y 95912783, si quisiéramos llamar al segundo, al llegar a marcar el 7, se produciría una llamada al primero, por lo que obligatoriamente, deben de ser códigos instantáneos para que esto no ocurra.

A la hora de construir un código que codifique un texto, con idea de minimizar el espacio que ocupa este, nos interesa que las palabras-código[i] sean de la menor longitud posible, y la mejor manera de lograr esto, es elegir esa longitud en función de la frecuencia de uso de la fuente a codificar. Si miramos por ejemplo el código Morse (aún sin ser código instantáneo) veremos que la letra '**E**' se codificar como un solo punto mientras que la '**J**' consta de un punto y cuatro rayas seguidas, esto se debe a que la '**E**' es mucho más frecuente que la '**J**' en los textos que analizaron para crear el código Morse.

Llamaremos A al alfabeto formado por el conjunto {a1, a2…an}
Ejemplo: A=alfabeto en minúscula, a1='a' ; a2='b' ; an='z'

Llamaremos Pk a la probabilidad de un elemento ak del conjunto A aparezca
Ejemplo P2 = Probabilidad(a2) = Probabilidad('b') = 0.013

Llamaremos C al conjunto de palabras código {c1…ck} al que se transcribe para ak

Ejemplo c2 = palabra-código(a2) = palabra-código('b') = 1101111

Llamaremos Lk a la longitud de la palabra-código ck
Ejemplo L(c2) = 7

Por último llamaremos L(C) a la longitud media del código, mientras menor sea la longitud media de un código, menor espacio ocupará, y esta es obtenida según la siguiente fórmula:

$$\sum P_k * L_k$$

Teorema de Kraft-McMillan

El teorema de Kraft-McMillan se descompone en dos partes.

q = La base del código, en nuestro caso siempre será 2 (binaria)

1. Si C es un código instantáneo cuyas longitudes de palabras código son L1, L2 … Ln entonces se cumple la siguiente desigualdad:

$$\sum \frac{1}{q^{L_k}} \leq 1$$

Ej: A={a, b, c} C={00,01,11} L={2,2,2} ➔ $\left(\frac{1}{2^2}\right) + \left(\frac{1}{2^2}\right) + \left(\frac{1}{2^2}\right) = \frac{3}{4} < 1$

por lo que podemos decir que C es un código instantáneo

2. Si se verifica la desigualdad anterior, podemos decir que existe un código instantáneo cuyas palabras-código tienen por lo longitudes los números Lk

Ej: Las longitudes de un código dado son L = {3, 3, 2, 2, 4}

➔ $\left(\frac{1}{2^3}\right) + \left(\frac{1}{2^3}\right) + \left(\frac{1}{2^2}\right) + \left(\frac{1}{2^2}\right) + \left(\frac{1}{2^4}\right) = \frac{13}{16} < 1$

por lo que puede existir un código instantáneo en base 2 para esas longitudes dadas

Demostración

Usando el ejemplo anterior, ordenaremos las longitudes de menor a mayor L1<=L2..<=Ln, siendo Ln la mayor longitud, la llamaremos Lmax. $2^{L_{max}}$ = número de variaciones con repetición de los bits (ya que nuestra base es 2) tomados de Lmax en Lmax (4 volviendo al ejemplo anterior, dando como resultado un total de 16 posibles repeticiones)

B1	B2	B3	B4
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Llamaremos α_j al número de palabras código con longitud J

$$\alpha_1 = 0, \alpha_2 = 2, \alpha_3 = 2, \alpha_4 = 1$$

No podremos formar un código instantáneo si no se verifica la siguiente desigualdad:

$$\alpha_1 \leq q$$

Ahora para continuar con las siguientes q^2 palabras-código de longitud 2, debemos eliminar las que comienzan por alguna palabra-código de α_1

$$\alpha_2 \leq q^2 - \alpha_1 q$$

Equivalentemente

$$\alpha_1 q + \alpha_2 \leq q^2$$

Si continuamos expandiendo la fórmula:

$$\alpha_1\, q^{L-1} + \alpha_2\, q^{L-2} + \cdots + \alpha_L \leq q^L$$

Al dividir todo entre q^L (el cual sabemos que es un número positivo):

$$\alpha_1\, q^{-1} + \alpha_2\, q^{-2} + \cdots + \alpha_L q^{-L} \leq 1$$

Escrito de otro modo:

$$\sum_{i=1}^{L} \frac{\alpha_i}{q^i} \leq 1$$

Dado que α_i es el número de veces que existen palabras código de longitud Li, podemos hacer la siguiente sustitución llegando a lo que queríamos demostrar:

$$\sum \frac{1}{q^{L_k}} \leq 1$$

Programa

Introducción

Se ha desarrollado un programa a fin de poder utilizar el algoritmo de Huffman para la compresión y descompresión de archivos de texto en castellano o, en general, que utilicen el sistema de codificación de texto ISO 8859-1, también llamado Latín 1.

Manera estándar
La manera tradicional de codificar los archivos de texto txt es mediante el uso del código ASCII, el cual usa los 7 últimos bits de un byte para codificar cada carácter, incluyéndose algunos caracteres de control como son el salto de línea, la tabulación o el retorno de carro.

En el caso concreto del castellano, el ASCII nos resulta insuficiente, ya que no incluye caracteres necesarios para nosotros como son la "ñ" o cualquier vocal tildada, por lo que nos es necesario ampliarlo usando el estándar de codificación ISO 8859-1 que por medio del uso del bit restante, sí incluye estos caracteres.

En estos dos estándares, la longitud de la palabra código que codifica cada letra es invariable, es decir, todos los caracteres miden lo mismo.

Las funciones básicas del programa son las siguientes

- Analizar la frecuencia de uso de caracteres en un archivo de texto en formato de texto plano que esté codificado en ASCII o use la codificación de texto ISO 8859-1[1].
- Habiendo analizado un texto, crear un archivo diccionario para la compresión/descompresión de textos.
- Comprimir archivos txt utilizando un archivo diccionario.
- Descomprimir textos (comprimidos con el programa) utilizando un archivo diccionario.

[1] Norma de la ISO que define la codificación del alfabeto latino, incluyendo los diacríticos (como letras acentuadas, ñ, ç), y letras especiales (como ß, Ø), necesarios para la escritura de las siguientes lenguas originarias de Europa occidental: afrikáans, alemán, español, catalán, euskera, aragonés, asturiano, danés, escocés, feroés, finés, francés, gaélico, gallego, inglés, islandés, italiano, holandés, noruego, portugués y sueco.

Esquemas

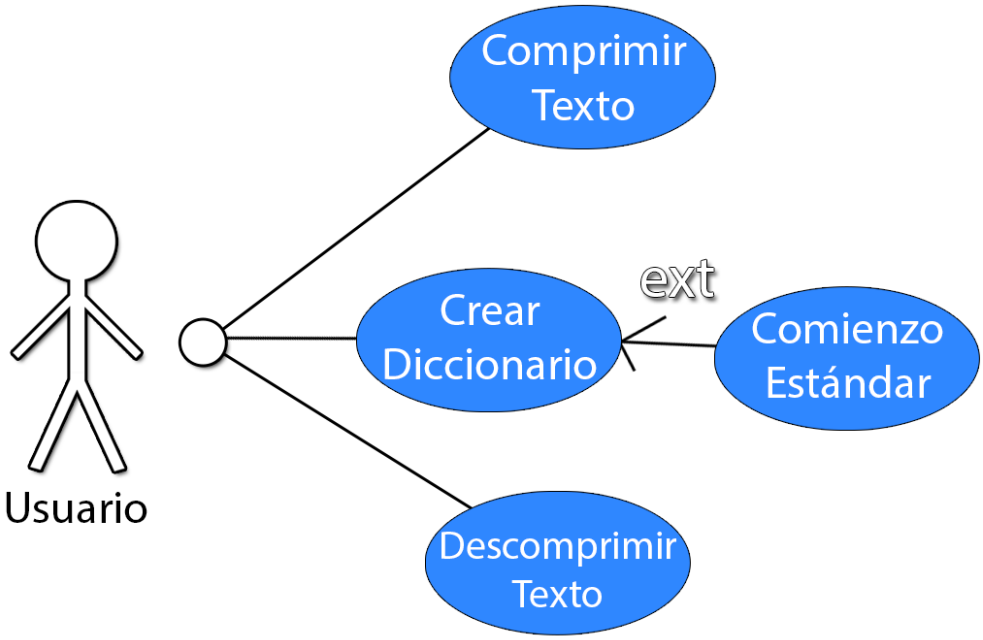

Ilustración 1. Diagrama de casos de uso

Diagramas de clase

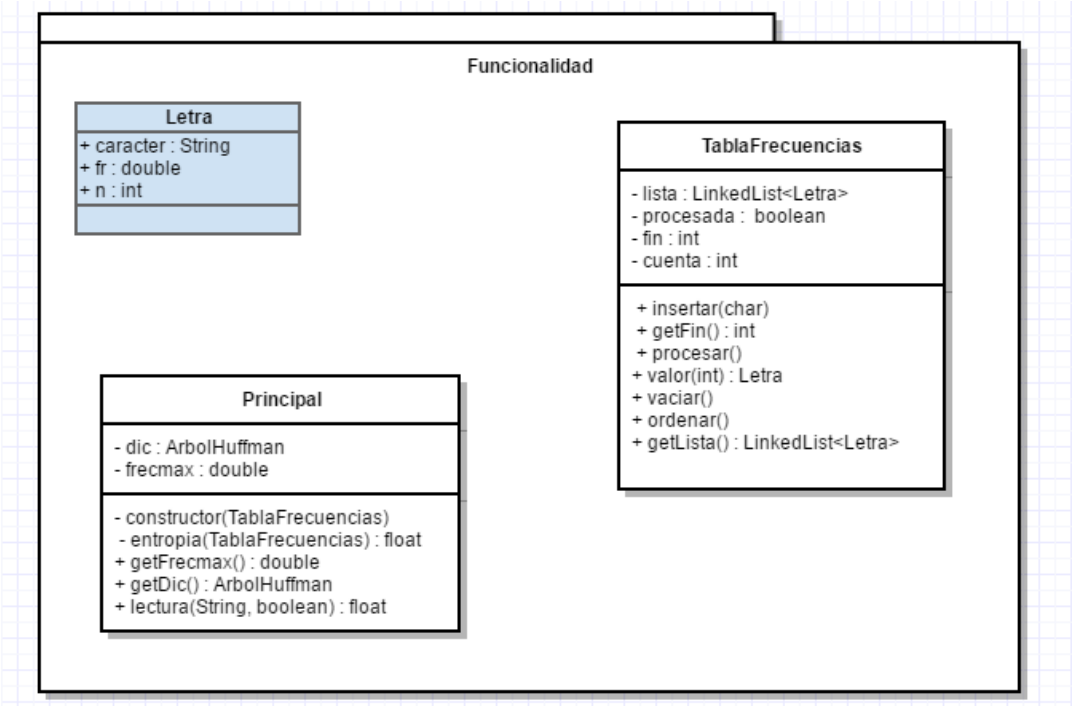

Ilustración 2. Diagrama de clases de paquete Funcionalidad

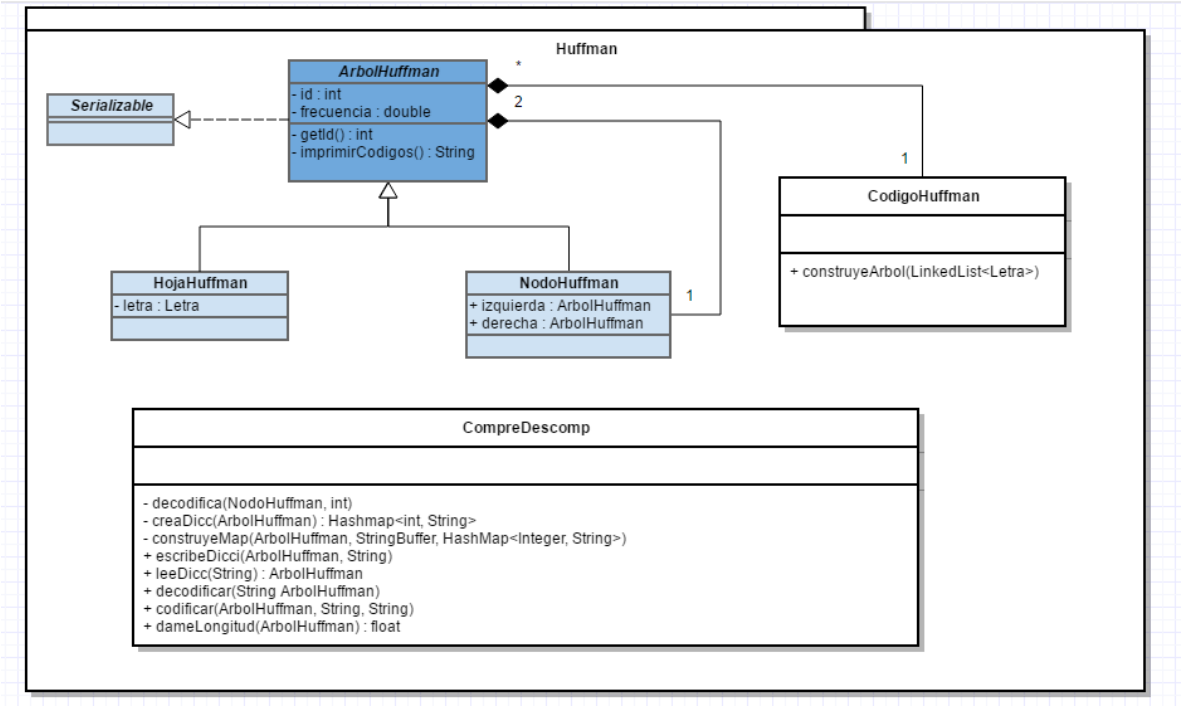

Ilustración 3. Diagrama de clases de paquete Huffman

Archivos

El programa hará uso de tres clases de archivos:

- Archivos .txt que contendrán los textos antes y después de ser comprimidos.
- Archivos .huf que contendrán el texto comprimido. Internamente un archivo .huf no es más que una cadena binaria formada a partir del diccionario y el texto.
- Archivos .dhu que contienen el árbol Huffman usado tanto para comprimir como descomprimir el texto.

Sobre el desarrollo

El programa ha sido desarrollado haciendo uso del lenguaje de programación **Java** junto al IDE[2] **Netbeans 8.1** debido a la facilidad de implementación, documentación, detección de errores, depuración de éstos y portabilidad del programa.

Además de por las características intrínsecas al lenguaje y el IDE, Java cuenta con una comunidad extensa que proporciona muchas librerías y ayudas a los desarrolladores.

[2] Siglas en inglés de Entorno de Desarrollo Integrado

Creación de diccionario:

1. Seleccionamos la pestaña del mismo nombre
2. Seleccionamos el txt a analizar
3. Al hacer click en el botón "Abrir", se crea un objeto ListaFrecuencias que está formado por un array de objetos Letra y unos métodos que serán explicados más adelante en su contexto.
4. El programa lee una a una cada letra creando para cada una un objeto Letra, consistente en el carácter leído, su representación decimal y su número de repeticiones, que aumenta cada vez que se lee el carácter

   4.1 Si se ha marcado la opción de incluir diccionario estándar, el programa, antes de comenzar a leer, introducirá en esta lista de objetos Letras, una repetición de cada carácter que puede leer, que son los caracteres legibles propios del ASCII junto a los de la codificación ISO 8859-1. Más adelante se explicará el por qué de incluir esta opción.

5. Al terminar de leer todos los caracteres del documento, se inserta también carácter cuyo código decimal es "3", es un carácter no legible propio del ASCII que sirve para marcar el final del documento, después se explicará por qué es importante también.

6. La TablaFrecuencias se ordenar de mayor a menor con un método interno y se "procesa" lo cual cambia el valor del número repeticiones de cada Letra por el porcentaje de veces encontrada en el documento.

7. Se toma el primer valor de la tabla, el cual será el porcentaje más alto de repeticiones, y se envía a la interfaz, que lo usa para mostrar la cota superior que nos indica la posible mayor longitud media de nuestra futura codificación

8. Se envía la tabla de frecuencias al método constructor del árbol Huffman que, por el momento, se mantiene en memoria.

9. Se calcula la entropía de Shannon o de la información, la cual es a su vez el otro valor de cota, la mínima, entre la que estará nuestra longitud media.

10. Estableceremos un nombre para el diccionario en el campo correspondiente y al hacer click el botón "crear diccionario" se creará en la carpeta del programa un archivo con el nombre elegido y la extensión .dhf (Diccionario HuFman). Este archivo contiene el árbol Huffman.

Compresión de texto.

1. El usuario elige la pestaña de este mismo nombre
2. Éste selecciona el archivo a comprimir
3. Se selecciona el archivo .dhf con el diccionario a utilizar para la compresión
4. Al hacer click en el botón comprimir el programa comenzará a leer una a una cada letra del archivo de texto seleccionado, buscará su palabra código en el diccionario seleccionado y, haciendo uso de la librería bitoutputstream escribirá un archivo

binario consistente en todas las traducciones a palabras código de las letras leídas en orden.
4.1 La opción de incluir diccionario estándar sirve para evitar que aparezca un error al buscar una letra que no existe en el diccionario.

5. Al terminar de leer todas las letras, se incluye el carácter de fin de texto mencionado anteriormente.

Descompresión de texto

1. El usuario selecciona la pestaña del mismo nombre
2. Selecciona el archivo de texto descomprimir
3. Selecciona el diccionario con el que ha sido comprimido el texto
4. Al hacer click en descomprimir, el programa lee uno a uno cada bit de los que está compuesto el archivo hasta que llega al carácter "fin de texto". Si este carácter no fuera incluido, el programa seguiría leyendo hasta que se acabaran los bits de los que está compuesto el archivo. La unidad mínima de tamaño de un archivo son los bytes, porque si nuestro último bit de carácter terminara antes de acabar el byte, el resto se leería como código basura y podría dar lugar a la inclusión en nuestro texto descomprimido de un carácter que no existía en el texto original.
5. Cada vez que se lee un bit, nos movemos a un nodo del árbol huffman, si llegamos a un nodo hoja, que contiene una letra, esta se escribe en nuestro texto descomprimido, y volvemos a la raíz del árbol para continuar leyendo.

Resultados:

Futuras mejoras:

- Lectura de varios documentos o de una carpeta con archivos de texto para conseguir una tabla de frecuencias más medida
- Inclusión de un modo de programa en terminal
- Un comienzo de archivo comprimido que indique a la hora de la compresión si el diccionario seleccionado es correcto o no.
- Compresión de varios documentos o carpetas a la vez

Ayudas externas

Owen Astrachan: librerías bit(input/output)stream usadas para leer/escribir respectivamente archivos binarios bit a bit.
https://rosettacode.org/wiki/Huffman_coding#Java: clase que contiene el algoritmo de Huffman que ha sido modificado para que las entradas y salidas coincidan con las necesarias para el programa.

[i] Palabra código. Si por el ejemplo el carácter 'G' se traduce al código 00101, éste último es la palabra-código.