***PPMC***

***Dos implementaciones de Compresor***

**Grupo “Rey Neno”**

Daniel Trinchero

Nazareno Romani

Sergio Soria

Carlos Carbone

Leonardo Petrora

**Documentación**

## Introducción

## La compresión de datos consiste en la reducción del volumen de información tratable (procesar, transmitir o grabar). En principio, con la compresión se pretende transportar la misma información, pero empleando la menor cantidad de espacio.

El espacio que ocupa una información codificada (datos, señal digital, etc.) sin compresión es el cociente entre la frecuencia de muestreo y la resolución. Por tanto, cuantos más bits se empleen mayor será el tamaño del archivo. No obstante, la resolución viene impuesta por el sistema digital con que se trabaja y no se puede alterar el número de bits a voluntad; por ello, se utiliza la compresión, para transmitir la misma cantidad de información que ocuparía una gran resolución en un número inferior de bits.  
La compresión de datos se basa fundamentalmente en buscar repeticiones en series de datos para después almacenar solo el dato junto al número de veces que se repite. Así, por ejemplo, si en un fichero aparece una secuencia como "AAAAAA", ocupando 6 bytes se podría almacenar simplemente "6A" que ocupa solo 2 bytes, en algoritmo RLE.

## Codificación aritmética

--------------------------------------------------ELIMINAR------------------------------------------------------------------

Es un método para [compresión de datos lossless](http://www.worldlingo.com/ma/enwiki/es/Lossless_data_compression). Normalmente, una cadena de caracteres tales como las palabras “hola allí” se representa usando un número fijo de pedacitos por carácter, como en [ASCII](http://www.worldlingo.com/ma/enwiki/es/ASCII) código. Como [Codificación de Huffman](http://www.worldlingo.com/ma/enwiki/es/Huffman_coding), la codificación aritmética es una forma de [variable-length](http://www.worldlingo.com/ma/enwiki/es/Variable-length_code) [codificación de la entropía](http://www.worldlingo.com/ma/enwiki/es/Entropy_encoding) ese convierte una secuencia en otra representación que represente los caracteres con frecuencia usados que usan pocos pedacitos y los caracteres infrecuentemente usados que usan más pedacitos, con la meta de usar pocos pedacitos en total. En comparación con otras técnicas de codificación de la entropía que separen el mensaje de la entrada en sus símbolos componentes y substituyan cada símbolo por una palabra de código, la codificación aritmética codifica el mensaje entero en un solo número.

--------------------------------------------------ELIMINAR------------------------------------------------------------------

Este método se basa, al igual que Huffman, en la distribución no uniforme de la frecuencia de aparición de cada símbolo. El método de Huffman resulta óptimo si y sólo si la probabilidad de aparición de cada símbolo es una potencia entera de ½, lo que usualmente no ocurre. El método de compresión aritmética no tiene esta restricción.

Primeramente se crea una lista con la frecuencia de aparición de cada símbolo. Luego se va comprimiendo los datos, representado los mismos en un único número real entre 0 y 1, donde se destina a cada símbolo un intervalo proporcional a su frecuencia de aparición. Cuanto mayor sea la cadena a comprimir, mayor será la cantidad de decimales necesarios para almacenar la información y menor será el intervalo para representarlo.

Por razones de eficiencia, no se trabaja con números de punto flotante para la representación de los números, ya que se desperdiciarían los bits usados por la mantisa, sino que se trabaja con números enteros. Existen varias formas de implementar el algoritmo. Una de ellas es hacer un cambio de escala y codificar en número entre 0 y MAX\_INT en vez de entre 0 y 1, en cuyo caso habría que pasar a un nuevo entero cuando se acaban los bits disponibles. Otra técnica, más eficiente pero más lenta, consiste en ir almacenando el número de a poco, teniendo en memoria tan sólo los bits que pueden cambiar de un extremo a otro del intervalo, y almacenando los bits que ya quedan fijos. Por ejemplo, si los extremos del intervalo son 0.10101100100 y 0.10101100111, se almacenan los bits 0.101011001, que son comunes a ambos intervalos, y se trabaja entre los extremos 0.00 y 0.11, que son los bits que difieren.

**PPMC**

Debido a los avances tecnológicos en áreas como la electrónica y la computación, la cantidad de datos a manejar ha crecido de tal forma que es preciso utilizar algoritmos de compresión que ofrezcan una buena razón de compresión y al mismo tiempo una velocidad aceptable según su aplicación. El algoritmo PPM presenta una buena razón de compresión, pero su velocidad puede llegar a ser inadecuada para algunas aplicaciones como la transmisión de datos.

El algoritmo PPM mantiene un diccionario con un modelo estadístico de los datos, asignando probabilidades a los símbolos y mandando estas probabilidades a un codificador aritmético.

El modelo estadístico en su forma más simple cuenta el número de veces que ocurre cada símbolo en el pasado y le asigna una probabilidad con base en ese número. Un modelo más complejo es el basado en *contexto*, donde no sólo la frecuencia del símbolo es usada para predecir sino que también la frecuencia del símbolo ocurrido cuando una secuencia de símbolos lo preceden inmediatamente. Dichos símbolos son llamados *contexto* y el número de ellos es el *orden* del contexto. Por ejemplo, en la frase: “compresió” se puede predecir que el siguiente símbolo en la cadena es “n”. La frase es el contexto y el orden es 9, porque es el número de símbolos que la conforman.

Como regla general, entre más grande es el orden del modelo mejor será la predicción.

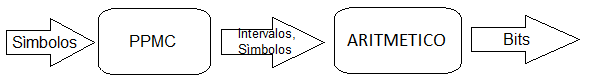
Desafortunadamente, la magnitud del espacio requerido para almacenar un contexto tan largo es prácticamente imposible de manejar. Aún restringiendo el contexto a 4 caracteres, habría (usando el típico byte de 8 bits) más de 4 billones de contextos posibles.

1. **Implementación**

Se implementó el algoritmo PPMC en dos compresores con optimizaciones en sus estructuras de datos. Uno con la tabla almacenada en forma de Árbol B+ y el otro en forma de Dispersión Extensible. Se modificaron las estructuras presentadas en la primera entrega, para que puedan ser utilizadas según lo requerido por los diferentes algoritmos de compresión.

Se agregaron utilidades para facilitar las diferentes operaciones de bits, así como también, otras para la conversión de cadenas de caracteres a una “tira de bits” y viceversa.

Los compresores que se implementan utilizan un aritmético como base para la compresión.



1. **Estructura**

**Aritmético**

*-Estructura*

**Clase Aritmetic**: en esta clase se encuentran los algoritmos que manejan la compresión y la descompresión mediante el sistema aritmético. Encapsula métodos que detectan y manejan el overflow y underflow, entre otros.

**Clase DistributionTable**: esta clase contiene un mapa que relaciona un carácter determinado con su frecuencia. Esta tabla la utilizan los algoritmos para la codificación/decodificación de los símbolos presentes en el texto utilizado para la tarea a realizar.

**PPMC**

*-Estructura*

**Clase Context**: clase que almacena el contexto del símbolo a ser codificado. Este contexto es utilizado por las tablas correspondientes al PPMC.

**Clase PPMC**: en esta clase se encuentran los algoritmos que manejan la compresión y la descompresión mediante el sistema PPMC . Contiene las tablas con las probabilidades de los caracteres según orden y contexto. En cada paso de compresión y descompresión pasa la tabla correspondiente al aritmético para transformar la información en bits.

1. **Funcionalidad**

Haciendo uso de los algoritmos implementados, podemos nombrar como funcionalidades más importantes a la compresión/descompresión de un archivo determinado, ver estadísticas de compresión/descompresión, ver los diferentes caracteres ASCII a ser comprimidos, etc.

Se contará con dos ejecutables, uno para estructura (Arbol B+ y Hash), a los cuales se le podrán ejecutar los siguientes comandos:

• Comprimir: c[orden del compresor]

• Descomprimir: d

• Destino: f[path relativo al archivo de trabajo]

• Estadísticas: e

• Ver todos los datos: v

• Ver datos particulares: vv[Número de Bloque]