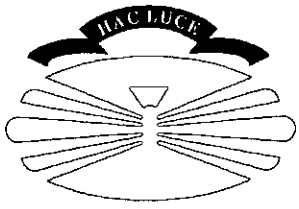
TESIS DOCTORAL

Una nueva técnica de compresión de textos con soporte de *Text Retrieval* y su adaptación a lenguas romances

Eva Lorenzo Iglesias

A Coruña, 2003



Universidade da Coruña

Departamento de Computación

Una nueva técnica de compresión de textos con soporte de *Text Retrieval* y su adaptación a lenguas romances

A Coruña, Junio de 2003

Doctoranda
Eva Lorenzo Iglesias

Directores
Nieves R. Brisaboa y José R. Paramá Gabía

Tesis doctoral dirigida por

Nieves Rodríguez Brisaboa
Departamento de Computación
Facultade de Informática
Universidade da Coruña
15071 A Coruña (España)
Tel: +34 981 167000 ext. 1243
Fax: +34 981 167160
brisaboa@udc.es

José Ramón Paramá Gabía
Departamento de Computación
Facultade de Informática
Universidade da Coruña
15071 A Coruña (España)
Tel: +34 981 167000 ext. 1241
Fax: +34 981 167160
parama@udc.es

Agradecimientos

Esta tesis se la debo a mis directores por la confianza que han puesto en mí y en mi capacidad para llevarla a cabo. Siempre han sabido orientar mi trabajo y hacerme sentir que algún día llegaría este momento.

La tesis se la dedico a mis padres y a mis hermanos, pero no puedo olvidarme de mi segunda familia que son todos los miembros del Laboratorio de Bases de Datos de la Universidade da Coruña y mis compañeros de Ourense. Hacia ellos no puedo tener nada más que elogios porque realmente son geniales y siempre formarán una parte muy importante en mi vida. Lo mejor de esta tesis ha sido conocerlos y compartir el día a día con ellos. Por su aportación concreta en este trabajo me gustaría agradecérselo de un modo especial a Jose, Fernando, Miguel, Floro, Fariña y Orge.

Quiero agradecer también a mis amigos (en especial a Ana, Tere, Fran y Emilio) por su pregunta habitual: "¿Cuándo acabas la tesis?".

Por último quiero dar las gracias también a los miembros del tribunal por haber accedido a la evaluación de este trabajo.

Espero responder a cada uno de ellos como esperan de mí, e incluso más.

Gracias a todos

A mis padres
A Susana
A mis grandes debilidades Víctor, Tamara y Nadia

Índice general

1

Introducción

1.1. Motivación

En la última década estamos asistiendo a un espectacular desarrollo de las tecnologías de la información y las comunicaciones. Esto ha favorecido el crecimiento en número y tamaño de las colecciones de textos en lenguaje natural. Actualmente, las bibliotecas digitales, bases de datos textuales y colecciones de textos de propósito general contienen cientos de gigabytes y la Web se mide en terabytes.

Cuando se manejan grandes volúmenes de información es necesario que los documentos puedan ser localizados y recuperados eficientemente. Los sistemas tradicionales sólo permitían realizar búsquedas sobre listas predeterminadas de palabras. Sin embargo, el aumento de la capacidad de los sistemas informáticos actuales ha facilitado la posibilidad de aplicar técnicas de *Recuperación de Textos* [7, 14]. Las técnicas de Recuperación de Textos permiten realizar búsquedas exactas o aproximadas de cualquier palabra (o conjunto de palabras) que forme parte del documento.

Ahora bien, no sólo es importante conseguir una recuperación eficiente, sino que también es necesario alcanzar eficiencia en términos de espacio de almacenamiento y/o de transmisión. La cantidad de espacio requerido para almacenar los textos se puede reducir significativamente utilizando *técnicas*

de compresión. Estas técnicas se basan en modificar la representación de un documento de forma que, gracias a la reducción de la *redundancia* presente en el texto original [1], la nueva representación ocupe menos espacio y que pueda obtenerse el texto original rápidamente [8].

El problema de la representación eficiente de la información no es algo reciente. Siempre ha habido interés en buscar nuevos medios para comprimir los documentos. Las formas más simples de compresión de textos fueron las aplicadas por los códigos *Braille* y *Morse*. El código Braille, que data originalmente de 1820, se basa en la utilización de matrices de puntos para representar las letras, dígitos y símbolos de puntuación, de modo que algunas de las palabras más comunes se representan simplemente con dos o tres caracteres. Samuel Morse desarrolló en 1837 un método de codificación que comprimía los datos utilizando representaciones más cortas para los símbolos más comunes. En su primera versión empleaba rayas cortas y largas, de modo que cada secuencia de rayas representaba un número, y cada número representaba una palabra (no un carácter). Más tarde, esta forma de codificación Morse se sustituiría por los conocidos puntos y rayas.

Hoy en día, estos métodos tienen únicamente interés histórico ya que son poco eficientes y no pueden competir con las técnicas de compresión que fueron desarrollándose posteriormente. Entre ellas, una de las más utilizadas para la compresión de textos y su almacenamiento automático es la *codificación Huffman* [19]. El método de Huffman emplea el mismo principio que la codificación Morse: los símbolos originales más comunes se codifican con códigos más cortos, mientras que los símbolos más raros se representan con códigos más largos. Desde su publicación en los años 50, la codificación Huffman fue considerada la mejor técnica de compresión hasta la aparición a finales de los 70 de los métodos *Ziv-Lempel* [43, 44] y *compresión aritmética* [21] que conseguían mejores ratios de compresión y no requerían grandes cantidades de memoria.

Sin embargo, las técnicas de compresión citadas no alcanzan el rendimiento adecuado cuando se aplican a grandes colecciones de textos en lenguaje natural. Los esquemas de compresión basados en codificación Huffman [19] dan lugar a ratios de compresión bajos, mientras que el método de Ziv-Lempel [43, 44] obtiene un ratio de compresión mejor pero la búsqueda

de una palabra sobre el texto comprimido es ineficiente [30].

En los últimos años se han desarrollado métodos de compresión especialmente indicados para su aplicación sobre grandes *corpus* de textos. En [37], Moura *et al.* presentan un esquema basado en la codificación Huffman que permite buscar una palabra directamente sobre el texto comprimido sin necesidad de descomprimirlo, consiguiendo que ciertas consultas puedan llegar a ser ocho veces más rápidas que en el caso de utilizar herramientas comerciales como *Agrep*. La idea clave del trabajo mencionado (y de otros como el de Moffat y Turpin [27]) es considerar las palabras, en lugar de los caracteres, como los símbolos atómicos a codificar. Dado que las palabras también son utilizadas como átomos por los métodos de Recuperación de Textos, las técnicas de compresión citadas son particularmente adecuadas en este campo. Empleando estos algoritmos es posible realizar directamente sobre el texto comprimido: (i) búsquedas exactas de palabras y frases aplicando cualquier algoritmo convencional de emparejamiento de patrones, (ii) búsquedas aproximadas y (iii) descompresión eficiente de porciones arbitrarias del texto.

El esquema de compresión de Moura *et al.* sugiere dos tipos de codificación. La primera de ellas, denominada *Plain Huffman*, consigue mejores ratios de compresión que la segunda, la codificación *Tagged Huffman*, aunque a costa de tener una eficiencia más baja a la hora de realizar búsquedas. La codificación Tagged Huffman agiliza considerablemente la recuperación pero se produce un incremento de alrededor del 11 % en el tamaño del fichero comprimido [36, 37].

A pesar de los buenos resultados obtenidos por estos métodos, es posible mejorar la compresión alcanzada por la codificación Tagged Huffman manteniendo todas sus ventajas ya que, según la teoría de la información, presenta todavía redundancia. Así, en este trabajo se presenta una nueva técnica de compresión de textos que consigue mejores ratios de compresión que dicha codificación y tiene un menor coste computacional, conservando plenamente las capacidades de búsqueda directa sobre el texto comprimido de palabras exactas, frases y búsqueda aproximada.

Dentro del ámbito de la compresión de textos, nuestro interés se centra especialmente en mejorar la eficiencia en el almacenamiento y recuperación

de documentos escritos en lenguas romances. Hasta la actualidad, los estudios de compresión de las técnicas basadas en palabras se han llevado a cabo principalmente sobre *corpus* de textos en inglés [36, 37]. En este trabajo se demuestra cómo la aplicación de las técnicas basadas en palabras actuales no resulta tan eficiente para la compresión y búsqueda de documentos escritos en lenguas romances. Esta afirmación se basa, principalmente, en dos hechos. El primero de ellos es que el número de entradas del vocabulario para las lenguas romances es mayor que el correspondiente a un texto del mismo tamaño en inglés, debido a que las palabras en lenguas romances tienen una amplia variedad gramatical. El segundo aspecto que afecta a la compresión es que la distribución de las frecuencias de las palabras en el vocabulario no es tan sesgada como para el inglés. A lo largo del trabajo se demuestra cómo ambos aspectos afectan negativamente en la compresión y, sobre todo, en la eficiencia de la recuperación.

El problema de manejar documentos con amplios vocabularios ya ha sido tratado en el ámbito de la Recuperación de Textos. Como consecuencia, se han desarrollado herramientas para el preprocesado de los textos y la reducción del vocabulario, entre las que destacan los *lematizadores* que permiten unir todas las variaciones morfológicas de una palabra bajo un mismo término (*lema*) [7, 14]. El problema de la lematización es que no permite volver a recrear el texto original dado que las palabras son sustituidas por sus lemas. Por ello no resulta una solución viable para su aplicación sobre textos comprimidos.

Sin embargo, partiendo de la idea de la lematización, se presenta una adaptación de las técnicas Huffman basadas en palabras de modo que resulten más adecuadas a lenguas con un número importante de variaciones morfológicas como es el caso de las lenguas romances. El método se basa igualmente en el preprocesado de los textos para lo cual, dada la imposibilidad de utilizar los algoritmos de lematización actuales, se ha desarrollado una nueva herramienta lingüística que permite reducir el tamaño del vocabulario e incrementar el número de palabras frecuentes. Los estudios experimentales realizados demuestran la efectividad del método propuesto, tanto en el aspecto de la compresión como de la realización de búsquedas aproximadas sobre el texto comprimido.

1.2. Objetivos

La línea de investigación bajo la que se enmarca este trabajo se orienta hacia el desarrollo de nuevas técnicas de compresión de textos que aumenten la eficiencia en la recuperación y almacenamiento de grandes *corpus* de documentos.

Como punto de partida se asumen las siguientes cuestiones:

- Los símbolos atómicos a codificar son las palabras que componen los documentos, no los caracteres. Esto es así puesto que de este modo aumenta la tasa de compresión de los documentos, como queda demostrado ya en la literatura [23, 26, 39, 37, 40, 45], y se ajusta mejor a las necesidades de búsqueda empleadas en el ámbito de la Recuperación de Textos.
- El criterio de evaluación de los esquemas analizados a lo largo de este trabajo es el adecuado balance entre el porcentaje de compresión alcanzado y la eficiencia de los algoritmos involucrados en la tareas de recuperación.

Una vez presentadas las consideraciones iniciales, pueden establecerse claramente los objetivos perseguidos:

- Presentación de un nuevo esquema de codificación, denominado *codificación Densa con Post-Etiquetado*, que mejora el balance ratio de compresión / eficiencia de recuperación de los esquemas de codificación Plain Huffman y Tagged Huffman propuestos por Moura *et al.*
- Adaptación de las codificaciones Huffman basadas en palabras y la nueva codificación Densa con Post-Etiquetado para su aplicación sobre textos con una mayor variedad morfológica que la lengua inglesa, como es el caso de las lenguas romances.

La consecución de estos objetivos se ha llevado a cabo del siguiente modo. Tras el estudio de los métodos de compresión de textos en lenguaje natural más relevantes en la actualidad, se ha definido el nuevo esquema de

codificación Densa con Post-Etiquetado. Su descripción se complementa con un estudio analítico para comparar la bondad del nuevo esquema respecto a otras técnicas similares (en concreto, las codificaciones Plain Huffman y Tagged Huffman). Habida cuenta de la imposibilidad de realizar este análisis teórico para todas las distribuciones posibles, se ha complementado con un estudio experimental donde se comparan los resultados obtenidos por los tres métodos en diversos corpus de textos en lengua inglesa.

En lo que se refiere al segundo objetivo, en primer lugar se demostraron las hipótesis que confirman la necesidad de adaptar las técnicas de compresión a textos con amplios vocabularios. Para ello, se ha extendido el estudio empírico anterior realizando la compresión de un conjunto de textos en lenguas romances (en concreto, en castellano, portugués y gallego) y el análisis de la distribución y del tamaño de los vocabularios de este tipo de textos.

Una vez confirmadas las hipótesis anteriores, se ha desarrollado una adaptación de las técnicas Huffman basadas en palabras y de nuestra codificación Densa con Post-Etiquetado, de modo que resulten más adecuadas a lenguas con amplios vocabularios. La nueva adaptación se ha apoyado con resultados empíricos sobre textos en lenguaje natural.

1.3. Estructura general de este trabajo

En el Capítulo 2 se presenta una revisión histórica de las técnicas de compresión de textos más conocidas y utilizadas. En el estudio realizado se hace especial hincapié en aquellos aspectos de cada esquema de compresión que son conservados o mejorados con nuestra propuesta. Las codificaciones Plain y Tagged Huffman, de Moura *et al.* [37], son tratadas con más detalle dado que son las codificaciones que más se asemejan a la planteada aquí y se emplean a lo largo del trabajo restante a modo de comparativa.

Una vez introducido el ámbito en el que se encuadra este trabajo, en el Capítulo 3 se describe el nuevo esquema de codificación Densa con Post-Etiquetado desarrollado. Los estudios analíticos y experimentales realizados sobre *corpus* en inglés se detallan en el Capítulo 4. En los estudios se

han incluido, a modo de comparativa, las técnicas de compresión Huffman basadas en palabras lo cual ha servido para demostrar las ventajas aportadas por el nuevo método, tanto a nivel de mejora en el porcentaje de compresión como en eficiencia de recuperación.

En el Capítulo 5 se presentan los resultados obtenidos tras la compresión de *corpus* de textos en lenguas romances (español, gallego y portugués, concretamente) con la nueva codificación Densa con Post-Etiquetado y las codificaciones Plain y Tagged Huffman. Los parámetros estudiados sirven para apoyar la necesidad de mejorar las técnicas de compresión dentro de este ámbito, por lo que se presenta la solución propuesta y la mejora obtenida tras su aplicación sobre los *corpus* anteriores.

Finalmente, en el Capítulo 6 se resume el trabajo realizado, se detallan las conclusiones alcanzadas y se plantea el posible marco de trabajo futuro.

2

Revisión de los métodos de Compresión de Textos

2.1. Introducción a la compresión de datos

Las técnicas de compresión buscan la reducción del número de símbolos necesario para almacenar o transmitir información [8]. Dentro del ámbito de las bases de datos documentales, las bibliotecas digitales y la Web, la compresión es un opción muy atractiva pues permite reducir los requerimientos de espacio y tiempo de transmisión.

A lo largo de los años se han desarrollado numerosos métodos de compresión, desde técnicas *ad hoc* hasta métodos más formales. Aunque sus resultados son de diferente carácter y cada uno de ellos resulta adecuado en determinados entornos, todos los métodos se basan en el mismo principio: realizan la compresión mediante la eliminación de la *redundancia* presente en los datos originales [34]. Si la compresión es efectiva, la longitud de la concatenación de todos los códigos de salida será menor que la longitud de la secuencia de símbolos inicial. Esto depende fundamentalmente de que exista suficiente redundancia en la cadena a comprimir.

El concepto de redundancia es fundamental en el ámbito de la compresión. Se relaciona con la existencia de cierta correlación entre los

elementos de información dentro de una colección. Esta correlación puede ser explotada para conseguir una representación más pequeña de los datos. Así, por ejemplo, en la mayoría de los textos en inglés el carácter E aparece con mucha frecuencia, mientras que la Z es poco habitual. Esto es lo que se denomina *redundancia alfabética*, y sugiere la asignación de códigos de tamaño variable a las letras, de modo que el código correspondiente a la E sea más corto que el código para la Z. Un ejemplo de redundancia en las imágenes se puede observar en el hecho de que los píxeles adyacentes tienden a tener colores similares.

En general, los compresores de datos se pueden dividir en dos familias:

- *Compresores con pérdida* ("lossy"): son utilizados para comprimir representaciones digitales de señales analógicas, como por ejemplo sonidos o imágenes, donde una pequeña pérdida de información no es apreciada por el sistema auditivo/visual humano. Dado que los datos expandidos (descomprimidos) no tienen por qué ser una réplica exacta de los datos originales, se pueden obtener ratios de compresión superiores.

- *Compresores sin pérdida* ("lossless"): garantizan la regeneración exacta de los datos originales tras la descompresión, ya que no se ha producido la pérdida de ningún tipo de información. Esta es la compresión utilizada sobre ficheros de texto, registros de una base de datos, bases de datos documentales, colecciones de texto de propósito general, información de una hoja de cálculo, etc. Ya que estos datos son comúnmente llamados *datos de texto*, dichos compresores se denominan también *compresores de texto*.

Uno de los mayores obstáculos para el almacenamiento de textos en formato comprimido es la necesidad de aplicar sobre ellos técnicas de Recuperación de Textos. Para poder obtener una palabra que forma parte de un texto comprimido mediante los compresores comerciales actuales es necesario descomprimir el texto desde el principio hasta alcanzar dicha palabra.

Actualmente están surgiendo nuevos métodos de compresión que resultan

más apropiados para el entorno de la Recuperación de Textos como, por ejemplo, los propuestos por Moura *et al.* en [36, 37]. Estos métodos se caracterizan por considerar las palabras como los símbolos a codificar, y no los caracteres como en el caso de los métodos clásicos. Utilizando las palabras como símbolos, además de alcanzar mejores ratios de compresión, se facilita el acceso a las mismas dentro del texto comprimido.

Relacionado también con la recuperación, otro elemento de gran interés es la posibilidad de realizar emparejamiento de patrones sobre el texto comprimido sin descomprimirlo en lo que comúnmente se denomina *emparejamiento de patrones comprimido*[37]. Aplicando emparejamiento comprimido las búsquedas se pueden acelerar de modo considerable ya que el texto que se debe analizar es mucho más pequeño.

En este capítulo se presenta una revisión de las técnicas de compresión de textos más conocidas y próximas a nuestro trabajo. Se ha estructurado en una primera sección donde, tras introducir la terminología que se empleará en adelante, se trata en profundidad la idea de redundancia y se define una serie de conceptos propios de la teoría de la información que se emplean habitualmente para la evaluación de los métodos de compresión.

Se dedican las Secciones 2.3 y 2.4 a una revisión de diferentes técnicas de compresión de textos, clasificándolas en *métodos de diccionario* y *estadísticos*. Básicamente, los métodos *de diccionario* utilizan un diccionario para realizar la codificación, mientras que los métodos *estadísticos* estiman las probabilidades de ocurrencia de los símbolos a codificar y asignan un código de compresión corto a los símbolos más frecuentes a costa de asignar un código de compresión largo a los símbolos más raros. Se pueden encontrar más detalles en [21, 34, 40].

En el resto del capítulo se estudian las técnicas de compresión con soporte de Recuperación de Textos propuestas por Moura *et al.* ya mencionadas anteriormente. Dichas técnicas se estudian con gran detalle por ser las más próximas al nuevo método de codificación desarrollado en este trabajo. Además se incluye una sección dedicada a presentar los diferentes tipos de búsqueda que se pueden aplicar sobre los textos comprimidos con estas técnicas, y algunos algoritmos propuestos para la búsqueda directa sobre el texto comprimido aplicando emparejamiento de patrones.

Finalmente, en la Sección 2.7 se recogen las conclusiones principales extraídas del estudio realizado.

2.2. Teoría de la información

2.2.1. Definiciones

Se denominan *símbolos de entrada* a cada una de las unidades básicas (caracteres, por ejemplo) en que se divide la cadena a comprimir, y *vocabulario de entrada* al conjunto de símbolos de entrada diferentes (y sin repeticiones) que componen dicha cadena. Llamaremos n a la longitud total de la cadena, es decir, el número total de símbolos de entrada a codificar, y v al número de símbolos que componen el vocabulario de entrada.

Ejemplo 2.2.1 En la cadena de entrada: $ABCDEDEDDDFABBBBC$ hay 17 símbolos de entrada:

$$\{A, B, C, D, E, D, E, D, D, D, F, A, B, B, B, B, C\}$$

y el vocabulario está formado por 6 símbolos: $\{A, B, C, D, E, F\}$ □

El *proceso de codificación* hace corresponder a cada símbolo de entrada un *código* que lo representará en la cadena de salida. Cada código estará formado por una secuencia de *símbolos de salida* que pertenecen a un *vocabulario de salida*. Llamaremos d al número de símbolos que componen dicho vocabulario.

El conjunto de símbolos de salida asignados a un vocabulario de entrada forman la *codificación* de la cadena de entrada.

Ejemplo 2.2.2 Una posible codificación para el vocabulario de entrada del ejemplo anterior podría ser de la forma:

$$\{A(0), B(1), C(10), D(11), E(1100), F(1111)\}$$

donde el vocabulario de salida está formado por $d = 2$ símbolos: 0 y 1. De este modo, la cadena codificada quedaría: 01101111001111001111111110111110

□

El *proceso de decodificación* permite realizar la operación inversa a la codificación; es decir, obtener la cadena original a partir de la secuencia codificada.

Denominaremos *esquema de codificación* al método que establece cómo llevar a cabo los procesos de codificación y decodificación.

Una codificación es *decodificable de forma única* si cada código es identificable dentro de la cadena codificada.

Ejemplo 2.2.3 Dada una codificación:

$$\{A(0), B(1), C(10), D(11)\}$$

la cadena 11 puede decodificarse como D o como BB y, por lo tanto, la codificación no es decodificable de forma única. □

Un código decodificable de forma única es de *prefijo único o libre* si ningún código es prefijo de otro.

Ejemplo 2.2.4 La codificación:

$$\{A(1), B(1000), C(00)\}$$

es decodificable de forma única (obsérvese que una cadena de la forma 100000 sólo puede decodificarse como BC); sin embargo, no es de prefijo libre (dado que el código asignado a A es prefijo del código correspondiente a B).

Por el contrario:

$$\{A(1), B(01), C(00)\}$$

sí es una codificación de prefijo libre. □

Toda codificación de prefijo libre es *decodificable de forma inmediata*; es decir, la cadena codificada puede dividirse en códigos sin necesidad de hacer un análisis "hacia adelante". Podemos reconocer directamente un código sin inspeccionar los siguientes códigos que forman la cadena comprimida.

Ejemplo 2.2.5 Para decodificar la cadena 10000001, donde la codificación es de la forma:

$$\{A(1), B(1000), C(00)\}$$

la lectura del primer símbolo 1 no es suficiente para confirmar que el símbolo de entrada sea A. De hecho, en este caso, no se puede determinar hasta que se haya leído el último símbolo (si la cadena de ceros hubiese sido de longitud impar entonces el primer código sería 1000 y, por lo tanto, correspondería al símbolo de entrada B).

Este problema no sucede en las codificaciones de prefijo libre, dado que nunca un código puede estar contenido en otro. Una vez reconocido un código dentro del texto se tiene la certeza de que es un código final, sin necesidad de tener que comprobar si podría ser parte de otro de mayor longitud. □

Una codificación de prefijo libre es *de prefijo mínimo* si, dado un símbolo (o secuencia de símbolos) de salida X que sea prefijo de algún código, Xy debe ser necesariamente un código o un prefijo de otro código, donde y representa a cada uno de los símbolos del vocabulario de salida.

Ejemplo 2.2.6 La codificación:

$$\{A(00), B(01), C(10)\}$$

es un ejemplo de codificación de prefijo libre que no es mínima. Esto se debe a que como 1 es prefijo del código 10 se requeriría también que 11 fuese código o prefijo, y esto no sucede así. Intuitivamente, la restricción de minimalidad evita el uso de códigos más largos de lo necesario.

En el ejemplo anterior, el código 10 puede sustituirse por 1, obteniéndose así una codificación de prefijo mínimo con códigos más cortos. □

Todas las codificaciones que se tratarán a lo largo de este trabajo son codificaciones de prefijo mínimo.

2.2.2. Evaluación de los métodos de compresión de datos

A la hora de evaluar las técnicas de compresión de datos deben tenerse en cuenta tres factores principales: la *redundancia* [35], la *complejidad de los algoritmos de codificación y decodificación* y el *porcentaje (ratio) de compresión* alcanzado. Todas estas medidas asumen que el documento a comprimir está formado por una secuencia de n símbolos pertenecientes a un vocabulario de v elementos. A cada símbolo i le corresponde una probabilidad constante p_i determinada por la frecuencia de aparición del símbolo i en la totalidad del documento.

Redundancia

El objetivo de la compresión de datos es, como ya se ha mencionado anteriormente, reducir la redundancia sin perder información. Por lo tanto, es crucial disponer de una medida que permita cuantificar la información asociada a un suceso (por ejemplo, la aparición de un símbolo determinado en un texto). Esta medida existe y se conoce como *cantidad de información* [1]. Su definición está relacionada con la disminución de la incertidumbre acerca de la ocurrencia de un suceso. Así, si nos dicen que el número que ha salido en un dado es menor que dos, nos dan más información que si nos dicen que el número que ha salido es par.

La cantidad de información que obtenemos al conocer un suceso es proporcional al número valores posibles que éste tenga *a priori*. Así, si inicialmente existen diez posibilidades equiprobables, conocer el suceso nos proporciona más información que si inicialmente tuviéramos dos. El concepto de *cantidad de información* proporcionada por un símbolo i es cuantificable y se puede definir de la forma $(-\log_d p_i)$ donde, recordemos, d es el número de símbolos del vocabulario de salida. Su interpretación es intuitiva; en caso de que $p_i = 1$ (probabilidad máxima) entonces i no es informativa ya que tiene que ocurrir siempre y, por tanto, la cantidad de información que nos aporta sería igual a 0. Igualmente, cuanto más pequeño sea el valor de p_i (es decir, es más improbable que suceda) mayor es su cantidad de información. Esto es lógico, ya que el estado de incertidumbre se reduce en mayor medida [1, 22].

La *cantidad media de información* H del vocabulario de entrada, conocido como *entropía*, es la media ponderada de la cantidad de información asociada a cada símbolo del vocabulario de entrada, considerando que los pesos en este promedio son las probabilidades de ocurrencia. La expresión de la entropía es de la forma:

$$H = \sum_{i=1}^{v} - p_i \log_d p_i$$

La definición de cantidad de la información se debe a Shannon. La derivación del concepto de entropía hacia la teoría de la información se presenta en [35].

La longitud del código asignado a cada símbolo de entrada i debe ser suficiente para representar su cantidad de información. Dado que la entropía H impone el límite inferior del número de símbolos necesarios para la codificación y que su valor no suele ser un entero, para aproximarse a la entropía deben emplearse códigos de longitud variable.

Muy relacionada con la entropía está la medida de la *redundancia* que se define como:

$$\sum_{i=1}^{v} p_i \, long_i - \sum_{i=1}^{v} - p_i \log_d p_i$$

donde $long_i$ es la longitud del código que representa el i-ésimo símbolo de entrada. El primer sumando representa las longitudes de los códigos pesados en base a sus probabilidades de aparición (es decir, la longitud media de los códigos). La segunda expresión es la entropía. Así, la redundancia es una medida de la diferencia entre la longitud media del código y la cantidad media de información. Si una codificación tiene una longitud media de códigos mínima para una determinada distribución de probabilidad discreta, se dice que es una codificación *óptima* o *de redundancia mínima.* Dado que la entropía es constante para una distribución de probabilidad dada, al minimizar la longitud media de los códigos se minimiza la redundancia.

Una codificación es *asintóticamente óptima* si tiene la propiedad de que la

redundancia tiende a 0 cuando la entropía de la distribución de probabilidad es muy grande. Es decir, la optimalidad asintótica garantiza que la longitud media de los códigos se aproxima al mínimo teórico.

Ratio de compresión

La cantidad de compresión alcanzada por un esquema de codificación se puede medir como un porcentaje de compresión. El término *ratio de compresión* C compara la longitud de la cadena obtenida tras la codificación respecto a la longitud de la cadena de entrada. Así, si la longitud de la cadena de entrada es u bytes y la de la cadena de salida es de c bytes, el porcentaje de compresión se obtiene como $(100 * c / u)$.

Complejidad

Aunque la meta principal de la compresión de datos es desarrollar técnicas que mejoren cada vez más el porcentaje de compresión, otro aspecto que no se debe descuidar es la *complejidad* de los algoritmos asociados a la técnica de compresión. La eficacia de las técnicas de compresión viene dada no sólo por el porcentaje de compresión alcanzado sino también por la eficiencia computacional (tanto temporal como espacial) de los algoritmos que llevan a cabo los procesos de codificación, compresión y descompresión.

2.3. Métodos estadísticos

Los primeros métodos de compresión, como el código Braille, asignaban códigos de longitud fija a los símbolos que forman la cadena a comprimir. Las codificaciones de tamaño fijo son útiles por su sencillez pero son redundantes.

Con el fin de conseguir mejores ratios de compresión disminuyendo en lo posible la redundancia (y, por lo tanto, aproximándose al mínimo teórico determinado por la entropía) se adoptó la *Ley general de compresión*, basada en "asignar códigos de salida cortos a los símbolos de entrada más comunes, y códigos largos a los menos habituales". Este es el principio seguido por

los métodos estadísticos, de modo que los códigos generados son de longitud variable.

Los métodos estadísticos (también llamados *de sustitución de orden cero* o *basados en símbolos*) pueden describirse en términos de un *modelo* asociado a la cadena de datos a comprimir y un *esquema de codificación*. El modelo asigna probabilidades a los símbolos que forman la cadena de entrada y, posteriormente, el esquema de codificación genera un código de salida para cada símbolo de entrada en función de estas probabilidades.

Los modelos de datos se pueden clasificar en:

- *Modelos no adaptativos* (*estáticos*): son rígidos y no modifican sus operaciones, parámetros o tablas en función de los datos a comprimir. Utilizan tablas de probabilidades prefijadas de antemano, independientes de la cadena de entrada. El uso de compresores basados en modelos no adaptativos es muy específico; resultan útiles en caso de que todos los datos sean de un mismo tipo. Un ejemplo podría ser el estándar JPEG para compresión de imágenes.

  El problema de estos modelos es el riesgo de que se apliquen sobre una cadena de entrada que no se ajuste perfectamente al tipo de datos para el que fueron creados. Es lo que sucede, por ejemplo, con la utilización del código Morse para codificar un fichero formado en su mayor parte por valores numéricos. En este caso, dado que los números tienen poca probabilidad de aparición en un texto están asociados a códigos más largos.

- *Modelos semiadaptativos* (*probabilísticos*): utilizan probabilidades obtenidas a partir de la cadena a comprimir. Se realizan dos pasadas: en la primera se lee la cadena completa para calcular la frecuencia de aparición de cada símbolo y en la segunda el esquema realiza la codificación. Por ello, los datos a comprimir deben estar almacenados o transmitirse dos veces, lo cual puede ser un inconveniente en ciertos entornos.

  Este tipo de modelos tiene dos inconvenientes. Por un lado, se necesita consumir un tiempo extra para calcular las probabilidades de aparición de los símbolos en la cadena a comprimir. Por otro lado, el tamaño

del fichero comprimido se ve incrementado debido a la necesidad de transmitir al decodificador el modelo utilizado (además del texto codificado), dado que la codificación de un símbolo varía para cada cadena de entrada.

- *Modelos adaptativos* (*dinámicos*): realizan una estimación de las probabilidades de los símbolos que componen el vocabulario de entrada en el momento de la codificación. El codificador "aprende" las características de la cadena de entrada. Un modelo adaptativo podría comenzar por considerar, por ejemplo, que todos los símbolos tienen la misma frecuencia; a medida que se va comprimiendo la cadena de entrada cuenta las apariciones de cada símbolo y va cambiando sus frecuencias (y, por lo tanto, el modelo).

  Las técnicas de compresión basadas en modelos adaptativos combinan dos aspectos clave: codifican en una sola pasada la cadena de entrada y pueden comprimir distintas cadenas de entrada sin necesidad de trabajar con un tipo de datos particular (como textos en inglés, por ejemplo).

El resto de esta sección se dedica a presentar diferentes técnicas de compresión estadísticas basadas en modelos semiadaptativos (en concreto, se estudian las codificaciones Shannon-Fano, Huffman y Aritmética). Se dedica una mayor atención a la codificación Huffman por ser el esquema más empleado actualmente en el ámbito de interés de este trabajo, la compresión de textos en lenguaje natural.

2.3.1. Codificación de Shannon-Fano

La codificación de Shannon-Fano fue el primer método estadístico que logró una buena codificación de longitud variable.

El modo de construir los códigos es el siguiente: los símbolos de entrada i y sus probabilidades p_i se ordenan de forma decreciente. La lista obtenida se divide para formar dos grupos, de modo que la probabilidad total de ambos grupos tenga un valor similar. A cada símbolo del primer grupo se le asigna un 1 como primer dígito, y a los símbolos de la segunda

Símbolo	Prob.	Paso 1	Paso 2	Paso 3
A	0,25	1	1	
B	0,25	1	0	
C	0,125	0	1	1
D	0,125	0	1	0
E	0,125	0	0	1
F	0,125	0	0	0

Tabla 2.1: Ejemplo de codificación Shannon-Fano

mitad se les asignan códigos comenzando en 0. Este proceso se realiza recursivamente subdividiendo cada grupo en base al mismo criterio hasta que cada subconjunto posea un solo elemento.

Ejemplo 2.3.1 En la Tabla 2.1 se muestra la codificación Shannon-Fano para un vocabulario de entrada formado por 6 símbolos: $\{A, B, C, D, E, F\}$ y cuyas probabilidades de aparición en el texto son las que se muestran en dicha tabla.

En el *Paso* 1 se divide el vocabulario en dos subconjuntos con frecuencia similar: $\{A, B\}$ y $\{C, D, E, F\}$. A los dos símbolos del primer conjunto se les asigna un 1 como primer símbolo de salida, mientras que a los símbolos del segundo conjunto se les asigna un 0. En el *Paso* 2 se divide cada conjunto anterior en dos subconjuntos y se sigue el mismo procedimiento. De este modo, el conjunto $\{A, B\}$ se divide en $\{A\}$ y $\{B\}$ y se asigna un nuevo símbolo de salida a cada uno (con valor 1 y 0, respectivamente). Igualmente se divide el conjunto $\{C, D, E, F\}$ en dos subconjuntos $\{C, D\}$ y $\{E, F\}$ y estos, a su vez, (en el Paso 3) se subdividen hasta que todos los conjuntos estén formados por un único elemento.

El código de salida para cada símbolo de entrada estará formado por la concatenación de los diferentes símbolos asignados en cada paso, quedando una codificación de la forma:

$$\{A(11), B(10), C(011), D(010), E(001), F(000)\}$$

□

El algoritmo de Shannon-Fano da lugar a un código de prefijo mínimo, aunque no siempre se trata de una codificación óptima. La longitud media de los códigos se encuentra entre la entropía H y la entropía más un símbolo, dado que la longitud de cada código i oscila entre $-log_d\ p_i$ (si los subgrupos tienen exactamente la misma probabilidad) y $(-log_d\ p_i\ +\ 1)$.

Ejemplo 2.3.2 La codificación obtenida para el ejemplo anterior es óptima, debido a que todos los grupos tenían exactamente la misma probabilidad. Así, la longitud media de los códigos es:

$$2\ (0,25 \times 2) + 4\ (0,125 \times 3) = 2,5$$

(hay 2 símbolos de entrada (A y B) cuya probabilidad es 0,25 y donde el número de símbolos que componen el código de salida es 2, y hay 4 símbolos (C, D, E, F) con probabilidad 0,125 y cuyos códigos de salida son de longitud 3), y el valor de la entropía es:

$$-[2\ (0,25\ \log_2\ 0,25) + 4\ (0,125\ \log_2\ 0,125)] = 2,5$$

□

2.3.2. Codificación Huffman

La codificación Huffman (también denominada *codificación de redundancia mínima*) fue descrita por primera vez por David A. Huffman en 1952 [19]. Debido a su facilidad de cómputo, es ampliamente utilizada en programas de compresión como *gzip*, máquinas de fax y en esquemas de compresión de imágenes como JPEG.

Al igual que sucedía para la codificación Shannon-Fano, Huffman asigna a los símbolos de entrada más frecuentes un código de compresión de menor longitud a costa de asignar códigos de mayor longitud a los menos frecuentes. La diferencia entre ambos métodos es que, mientras que la generación de códigos en Shannon-Fano comienza para los símbolos de entrada más frecuentes, la codificación Huffman construye los códigos de abajo a arriba (desde los símbolos de entrada menos probables hasta los más probables).

En general, la codificación Huffman da lugar a mejores codificaciones que Shannon-Fano.

El proceso de codificación de Huffman se compone de varias fases. En primer lugar es necesario calcular la frecuencia de aparición de las palabras, para posteriormente construir del árbol de Huffman que asignará un código a cada símbolo del vocabulario de entrada (palabra). Finalmente, se recorre una vez más el texto a comprimir sustituyendo cada elemento del texto por el código asignado mediante el árbol.

A continuación se presentan con más detalle cada una de estas fases.

Construcción del árbol Huffman

Para construir un código Huffman normalmente se utiliza un árbol que se construye a partir de las probabilidades de los símbolos de entrada que deseamos codificar y una lista donde se mantienen los nodos que todavía no han sido procesados. El proceso de construcción de dicho árbol se realiza de la siguiente forma:

1. Crear tantos nodos (que serán las hojas del árbol) como símbolos tenga el vocabulario de entrada. En cada nodo se almacena el símbolo y su probabilidad. Estos nodos forman la lista inicial de nodos sin procesar.

2. Extraer d nodos de la lista de nodos sin procesar que tengan las probabilidades más bajas, siendo d el número de símbolos que componen el vocabulario de salida. Así, para una codificación orientada a bit, donde el vocabulario de salida es $\{0, 1\}$, se toman los dos nodos de menor probabilidad.

3. Crear en el árbol un nuevo nodo que sea padre de los nodos seleccionados e incluirlo en la lista de nodos sin procesar. Este nodo no tiene asociado ningún símbolo y su probabilidad será la suma de las probabilidades de sus nodos hijo.

4. Repetir los pasos 2 y 3 hasta que tengamos en la lista un número de nodos sin procesar inferior a d. La suma de las probabilidades de estos

nodos debe ser igual a 1, y serán los hijos de un nuevo nodo en el árbol denominado *nodo raíz del árbol Huffman*. Puesto que en cada iteración extraemos d nodos de la lista e insertamos otro, el número de iteraciones necesarias para construir un árbol con v hojas (el número de símbolos que deseamos codificar) es $\lceil \frac{v}{d-1} \rceil$.

Ejemplo 2.3.3 Veamos cómo construir el árbol Huffman para un vocabulario de entrada formado por 5 símbolos: $\{A, B, C, D, E\}$ con pesos $0,40$, $0,18$, $0,15$, $0,15$ y $0,12$, respectivamente, y donde el vocabulario de salida es $\{0, 1\}$. El proceso se realiza en los 5 pasos siguientes(representados en la Figura 2.1):

1. El primer paso (Figura 2.1-A) consiste en crear una lista de nodos sin procesar formada por todos los nodos hoja.

2. Se extraen los $d = 2$ nodos de menor peso correspondientes a los símbolos D y E.

   Es importante indicar aquí que también podría haberse optado por seleccionar el nodo C en lugar del D, lo que daría lugar a un árbol diferente. Esta circunstancia se da siempre que haya símbolos equiprobables. Por lo tanto, la codificación Huffman no es única. Sin embargo, desde el punto de vista de la compresión de datos el resultado es equivalente. Dado que los símbolos intercambiables (C y D en este caso) tienen exactamente la misma probabilidad, la compresión alcanzada para la secuencia de entrada será la misma.

   Siguiendo el ejemplo, dado que los pesos de los nodos D y E son $0,15$ y $0,12$ respectivamente, insertamos un nuevo nodo en la lista, sin símbolo asociado y con un peso igual a $0,27$ (Figura 2.1-B).

3. Para construir el segundo subárbol se extraen los nodos B y C (Figura 2.1-C) y se genera un nuevo nodo intermedio con valor $0,33$ que se introduce en la lista de nodos sin procesar.

4. A continuación se extraen de la lista los nodos de peso $0,27$ y $0,33$, de modo que los dos subárboles forman un árbol mayor (Figura 2.1-D) cuyo nodo padre (que también se introduce en la lista de nodos sin procesar) toma el valor $0,60$.

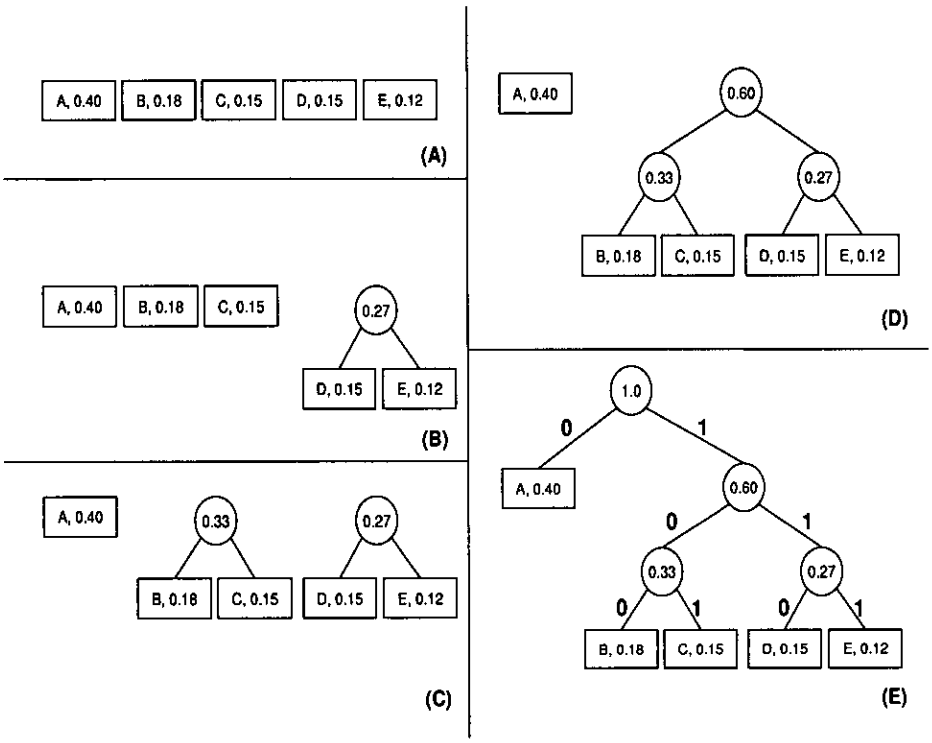


Figura 2.1: Ejemplo de construcción de un árbol Huffman

5. Por último, se extraen los dos nodos restantes (el correspondiente al símbolo A y el nodo intermedio de valor $0,60$) y se genera el nodo raíz de valor 1. El árbol Huffman completo se presenta en la Figura 2.1-E.

□

Procesos de codificación, compresión y descompresión

Una vez construido el árbol ya se puede llevar a cabo el proceso de codificación. Para ello, en cada nivel del árbol se etiquetan las ramas con los diferentes valores del vocabulario de salida. El código correspondiente a cada símbolo de entrada estará formado por la secuencia de símbolos de

salida asignados a las ramas que unen la raíz con la hoja que representa dicho símbolo de entrada.

Ejemplo 2.3.4 Siguiendo el Ejemplo 2.3.3, podemos etiquetar las ramas izquierdas del árbol con el símbolo de salida 0 y las ramas derechas con el símbolo de salida 1 (ver Figura 2.1-E). De esta forma, la codificación quedaría:

$$\{A(0), B(100), C(101), D(110), E(111)\}$$

□

Como puede observarse en este ejemplo, la codificación obtenida es de prefijo libre pues ningún código está contenido en otro. Esta propiedad se cumple siempre, dado que los símbolos de entrada están representados por nodos hoja y no por nodos internos del árbol.

Una vez calculados los códigos, el proceso de compresión consiste en generar una cadena de salida leyendo de nuevo la cadena de entrada y reemplazando cada uno de sus símbolos con el código asignado.

Ejemplo 2.3.5 Para la cadena de entrada $ABACDE$, la secuencia de códigos de salida resultante sería: 01000101110111. □

El proceso de decodificación se debe realizar utilizando el mismo árbol Huffman generado durante la codificación. Dado que la compresión Huffman no es única, tal como ya se ha mencionado, para que en el momento de la decodificación se pueda reconstruir exactamente el mismo árbol, es necesario que la cadena comprimida posea suficiente información para poder conocer el código asignado a cada símbolo de entrada. Aunque se podría optar por enviar en la cadena de salida el vocabulario de entrada y sus códigos (además del texto comprimido), esta solución penaliza considerablemente el porcentaje de compresión sobre todo cuando la cadena a comprimir no es muy grande. En este caso, el porcentaje de espacio ocupado por el vocabulario puede ser mayor incluso al del texto comprimido. Existen otras soluciones más eficientes, como se comentará con detalle en el apartado dedicado a los árboles Huffman canónicos. En este caso no se envía en la

cadena de salida el árbol Huffman, sino únicamente unos cientos de bytes de información extra.

Una vez regenerado el árbol, el proceso de decodificación es simple. El texto comprimido se lee símbolo a símbolo mientras se recorre el árbol partiendo de la raíz y siguiendo la rama cuyo valor coincida con el símbolo leído. Cuando se alcanza una hoja, se obtiene el símbolo de entrada asociado.

Gracias a la propiedad de prefijo libre, la decodificación de un código puede realizarse sin necesidad de analizar los códigos que le siguen dado que se reconoce inmediatamente su final.

Ejemplo 2.3.6 Utilizando el árbol de la Figura 2.1-E, vamos a decodificar la cadena comprimida del ejemplo anterior: 01000101110111. El símbolo de salida 0 nos indica que el primer símbolo de entrada es A. Para obtener el siguiente símbolo de entrada nos situamos de nuevo en la raíz del árbol y leemos carácter a carácter hasta alcanzar de nuevo una hoja. En este caso, una vez leído 100 habremos alcanzado la hoja correspondiente al símbolo de entrada es B. Este proceso se repite hasta que no quedan más símbolos por decodificar. □

Redundancia del código

El algoritmo de Huffman siempre produce una codificación óptima de redundancia mínima [12]. Su diferencia respecto a la entropía es debida únicamente a que Huffman no puede generar un código utilizando partes fraccionarias de símbolos de salida. Así, si la entropía de un símbolo de entrada es $2,5$, el algoritmo de Huffman generará un código formado por 3 símbolos de salida. Generalizando, si la entropía correspondiente a un símbolo de entrada i con probabilidad p_i es $(-log_d p_i)$, Huffman utilizará $\lceil -log_d p_i \rceil$ símbolos; es decir, nunca se utilizará más de un símbolo extra por código debido a los ajustes por redondeo mencionados.

Por lo tanto, la longitud media de los códigos Huffman asignados a un texto cuyo vocabulario de entrada está formado por v símbolos es:

$$\sum_{i=1}^{v} p_i \lceil -log_d p_i \rceil \leq 1 + \sum_{i=1}^{v} p_i - log_d p_i$$

Ejemplo 2.3.7 Vamos a analizar la redundancia de la codificación generada para el Ejemplo 2.3.3.

Bajo codificación Huffman, la longitud media de los códigos se puede calcular del mismo modo que el utilizado en el Ejemplo 2.3.2:

$$(1 \times 0,40) \; + \; (3 \times 0,18) \; + \; (3 \times 0,15) \; + \; (3 \times 0,15) \; + \; (3 \times 0,12) \; = \; 2,2$$

o, por tratarse de un árbol donde el peso de sus nodos es la suma del peso de sus hijos (frecuencias de las palabras asociadas), también puede calcularse sumando los valores asignados a los nodos internos del árbol:

$$0,27 + 0,33 + 0,60 + 1 \; = \; 2,2 \text{ símbolos}$$

Por otro lado, la entropía es:

$$-[(0,40 \log_2 0,40) + (0,18 \log_2 0,18) + (0,15 \log_2 0,15) + (0,15 \log_2 0,15) + (0,12 \log_2 0,12)] \; = \; 2,16 \text{ símbolos}$$

Como se puede observar, la diferencia entre la longitud media y la entropía es inferior a 1 símbolo. □

La eficiencia de la codificación Huffman se ve particularmente afectada cuando existen algunos símbolos de entrada con muy alta probabilidad. De forma más general, Gallager [15] ha demostrado que el límite superior de la redundancia de la codificación Huffman binaria (es decir, cuando el número de símbolos de salida es 2) es:

$$p_b \; + \; \log_2 \; [\; (2 \; \log_2 \; e) \; / \; e \;] \simeq (p_b + 0,086)$$

donde p_b es la probabilidad del símbolo más probable. Capocelli *et al.* determinaron límites más ajustados que los de Gallager para algunas distribuciones de probabilidad [11].

Por otro lado, Buro [10] estableció el límite superior de la longitud de un código Huffman en base a las probabilidades de los dos símbolos menos probables p_1 y p_2, como:

$$\text{mín}\left\{\left\lfloor \log_\Phi\left(\frac{\Phi+1}{p_1\Phi+p_2}\right)\right\rfloor, v-1\right\}$$

siendo $\Phi = (1+\sqrt{5})/2$.

Árbol Huffman canónico

Como ya se ha mencionado anteriormente, la codificación Huffman no es única. Así, en el Ejemplo 2.3.3 podría haberse seleccionado el nodo C en lugar del D, en cuyo caso el árbol resultante sería diferente. El intercambio de los subárboles de un nodo intermedio da lugar a un árbol distinto siempre que los subárboles sean diferentes en estructura, aunque la longitud media final del código no se vea afectada.

Para garantizar que, dado un vocabulario de entrada, siempre se obtiene el mismo árbol Huffman (y, por lo tanto, los mismos códigos), muchas aplicaciones optan por generar un *árbol canónico* que impone un orden particular de codificación [7, 18, 40].

Un árbol Huffman es canónico cuando el peso del subárbol izquierdo de cualquier nodo nunca es menor que el del subárbol derecho, y todas las hojas se colocan de izquierda a derecha en orden creciente de probabilidad.

Ejemplo 2.3.8 El árbol canónico para el Ejemplo 2.3.3 se presenta en la Figura 2.2. La codificación canónica queda la forma:

$$\{C(000), B(001), E(010), D(011), A(1)\}$$

□

La codificación canónica se caracteriza porque el conjunto de códigos de la misma longitud son las representaciones binarias de enteros consecutivos.

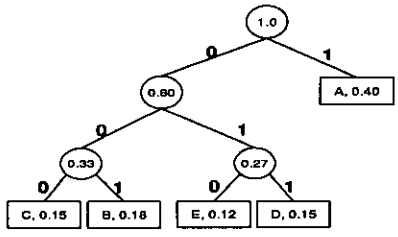


Figura 2.2: Árbol canónico para el Ejemplo 2.3.3

Esto puede observarse en el ejemplo anterior, donde a los 4 símbolos de entrada codificados con 3 bits (C, B, E, D) se les han asignado los códigos cuyo valores enteros son 0, 1, 2 y 3, respectivamente.

Esta característica de secuencia numérica aporta importantes ventajas a la codificación. Utilizando un árbol canónico no es necesario construir el árbol Huffman para la codificación ni para la decodificación, mejorando consecuentemente la eficiencia de estos procesos tanto en tiempo de computación como en espacio. De hecho, para la construcción de la codificación no es necesario llevar a cabo la generación de los códigos. Basta con conocer la longitud del código asignado a cada símbolo de entrada y cuál es el valor del código más pequeño para cada longitud. Un algoritmo para el cálculo de las longitudes de los códigos Huffman para los símbolos que forman una cadena de entrada puede verse en [40].

De este modo, la codificación canónica puede representarse mediante un array S de pares de la forma (x_j, y_j) siendo $1 \leq j \leq l$, donde x_j representa el número de códigos de longitud j, y_j el valor numérico del primer código de longitud j, y l es la altura del árbol.

Ejemplo 2.3.9 Para el ejemplo anterior, el árbol podría representarse de la forma:

$$S = \{(1,\ 1), (0,\ \infty), (4,\ 0)\}$$

porque hay 1 símbolo de entrada (A) cuyo código es de longitud 1 y su código es 1, no hay códigos de longitud 2, y hay 4 códigos de longitud 3 que

comienzan por el valor 0. □

La información aportada por el array anterior, junto con una lista de los símbolos que componen el vocabulario de entrada ordenados decrecientemente por frecuencia, es suficiente para la generación de la codificación Huffman canónica en cualquier instante. Gracias a las propiedades del árbol canónico, no es necesario almacenar el vocabulario con sus códigos, propiedad especialmente interesante cuando se trabaja con grandes vocabularios ya que, como ya se ha mencionado, el vocabulario tiene que almacenarse con el texto, de modo que el tamaño total del fichero comprimido será la suma del tamaño del texto comprimido y del vocabulario.

2.3.3. Codificación aritmética

La *codificación aritmética* es una técnica sofisticada que alcanza una compresión muy próxima a la determinada por la entropía.

Bajo esta codificación, la cadena de entrada se representa por un único número real que pertenece al intervalo [0, 1). El método parte de una lista desordenada con los símbolos que componen el vocabulario de entrada y sus probabilidades de aparición en el texto a comprimir. A medida que se realiza la codificación del texto, el intervalo inicial [0, 1) se divide en subintervalos más pequeños. El comienzo de cada subintervalo y su tamaño dependen de la posición que ocupe el símbolo de entrada en el vocabulario y de su probabilidad. Veámoslo con un ejemplo:

Ejemplo 2.3.10 Supongamos un vocabulario de entrada formado por 5 símbolos $\{A, B, C, D, E\}$ cuyas probabilidades de aparición en el texto se muestran en la Tabla 2.2.

A cada símbolo de entrada se le hace corresponder la parte proporcional del intervalo [0, 1) en función de la posición que ocupa en el vocabulario y de su probabilidad inicial. Así, tal como se muestra en la última columna de la Tabla 2.2, al símbolo A le corresponden los primeros 2/10 del intervalo (es decir, el intervalo [0, 0,2)), al símbolo B le corresponden los 4/10 siguientes (es decir, el intervalo [0,2, 0,6)), y así sucesivamente.

Una vez determinado el rango del intervalo que corresponde a cada símbolo puede comenzar la codificación.

En la Figura 2.3 se muestran los diferentes subintervalos generados para la codificación de una cadena de entrada de la forma $AADBE$. Tomando en orden los símbolos que componen la cadena de entrada, los pasos seguidos por la codificación aritmética son los siguientes:

$A \Rightarrow$ dado que al símbolo A le corresponden los primeros 2/10 del intervalo, entonces el intervalo inicial [0, 1) se reduce a [0, 0,2).

$A \Rightarrow$ se reduce el intervalo anterior [0, 0,2) a sus 2/10 primeras partes; es decir, al intervalo [0, 0,04).

$D \Rightarrow$ dado que el rango del símbolo D (tal como se indica en la Tabla 2.2) es [0,7, 0,9), el intervalo obtenido en el paso anterior debe reducirse a sus 2/10 partes, pero comenzando en el 70% de la distancia desde el extremo izquierdo del intervalo. Es decir, el nuevo intervalo es [0,028, 0,036).

$B \Rightarrow$ al símbolo B le corresponden 4/10 partes del intervalo anterior [0,028, 0,036), comenzando en el 20% de la distancia desde el extremo izquierdo. Es decir, el nuevo intervalo es [0,0296, 0,0328).

$E \Rightarrow$ se determina el intervalo final como el último 1/10 del intervalo anterior. Es decir [0,03248, 0,0328).

El propio intervalo [0,03248, 0,0328), o cualquier valor c dentro del mismo, puede usarse para representar la cadena de entrada. □

Para la decodificación debe conocerse el modelo usado por el codificador (es decir, el vocabulario de entrada y las probabilidades de sus símbolos) y el número c elegido por el codificador dentro del intervalo. La decodificación consiste en una serie de comparaciones del valor c con los rangos de los símbolos del vocabulario de entrada, simulando las operaciones del codificador.

Ejemplo 2.3.11 Para el ejemplo anterior, si el codificador hubiese elegido el valor de $c = 0,0325$, como está entre 0 y 0,2, el decodificador sabría

Símbolo	Prob.	Rango
A	2/10=0,2	[0, 0,2)
B	4/10=0,4	[0,2, 0,6)
C	1/10=0,1	[0,6, 0,7)
D	2/10=0,2	[0,7, 0,9)
E	1/10=0,1	[0,9, 1,0)

Tabla 2.2: Ejemplo de primera partición de la codificación aritmética

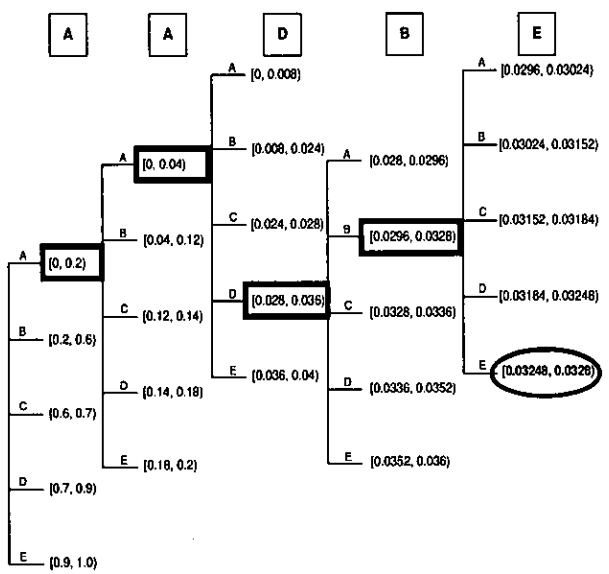


Figura 2.3: Codificación aritmética de la cadena *AADBE*

que el primer símbolo es una A, pues el rango [0, 0,2) corresponde a la A. El decodificador ahora puede deducir que el siguiente símbolo se corresponderá con alguno de los intervalos [0, 0,04) si es A, [0,04, 0,12) si es B, [0,12, 0,14) si es C, [0,14, 0,18) si es D, y [0,18, 0,2) si se trata de E (ver Figura 2.3). Como c se encuentra en el intervalo [0, 0,04) el segundo símbolo será una A, y así sucesivamente. □

Aunque la codificación Huffman tiende a ser más rápida que la codificación aritmética, la codificación aritmética alcanza una compresión más próxima a la óptima para una distribución de probabilidades dada. El problema es que, debido a su naturaleza incremental, no se puede realizar la decodificación de un trozo intermedio del texto comprimido. Para decodificar un símbolo en el medio de un fichero comprimido con codificación aritmética es necesario decodificar el texto completo desde el principio hasta la palabra deseada. Esta característica hace que la codificación aritmética resulte inadecuada en el ámbito de la Recuperación de Textos.

2.4. Métodos de diccionario

Los métodos que hemos presentado hasta el momento se caracterizan por utilizar un modelo estadístico que asigna probabilidades a los símbolos que forman la cadena de entrada, y un esquema de codificación que genera un código de salida (de longitud variable) para cada símbolo de entrada en función de estas probabilidades.

Los métodos *de diccionario* no utilizan un modelo estadístico ni códigos de longitud variable. En su lugar sustituyen subcadenas del texto original por códigos que identifican dicha subcadena en un diccionario. Un ejemplo muy simple podría ser un método que realizase la compresión de textos en castellano utilizando directamente un diccionario en esta lengua. Para realizar la codificación se extraen secuencialmente las palabras del texto a comprimir y se buscan en el diccionario. Cada vez que se localiza una palabra en el diccionario se escribe en la cadena de salida un índice a la entrada correspondiente en el diccionario.

Los diccionarios pueden ser *estáticos* o *dinámicos(adaptativos)*. Un

diccionario estático está formado por un conjunto fijo y prestablecido de entradas (aunque en ocasiones permite añadir nuevas). Un ejemplo de diccionario estático es el empleado en la codificación ASCII o cuando, por ejemplo, sustituimos "Diciembre" por 12 o TR para referirnos a *Text Retrieval*.

El problema de los diccionarios estáticos es que no suelen adaptarse bien a cualquier texto. Por ello los compresores de texto comerciales optan por una solución más elegante y efectiva que es la utilización de diccionarios dinámicos. Estos diccionarios se crean para cada texto a codificar.

Casi todas las técnicas de diccionario adaptativas están basadas en uno de los dos métodos Ziv-Lempel que se detallan a continuación.

2.4.1. Codificación Ziv-Lempel

Dos investigadores israelíes, Jacob Ziv y Abraham Lempel, desarrollaron en la década de los 70 los primeros métodos de compresión basados en diccionarios adaptativos, LZ77 y LZ78 [43, 44]. Sus ideas han sido básicas para el desarrollo de muchos métodos actuales para la compresión sin pérdida de textos (como *gzip* o *compress*), sonidos e imágenes.

Ambos métodos (LZ77 y LZ78) se basan en el mismo principio: las subcadenas de texto se sustituyen por punteros a ocurrencias previas de las mismas en el texto ya analizado. De este modo, el diccionario está formado por el texto analizado, y los códigos se representan por punteros. La utilización de un diccionario de este tipo suele resultar muy adecuado pues el texto analizado generalmente está formado por símbolos del mismo alfabeto (de la misma lengua si hablamos de texto en lenguaje natural) que el texto que falta por codificar. Además, de este modo el diccionario se transmite implícitamente en el texto codificado sin ningún coste de almacenamiento extra. Las diferencias entre los métodos LZ77 y LZ78 vienen dadas básicamente por la forma en que se representan los punteros y el tipo de subcadenas a las que pueden apuntar, por lo que a partir de aquí nos centraremos en LZ78.

El método LZ78 utiliza un diccionario cuyas entradas son subcadenas. En

cada paso de la codificación se extrae una subcadena del texto no analizado, se añade al diccionario una entrada formada por esta subcadena y el código asignado, y se escribe dicho código en el texto comprimido.

El algoritmo de codificación divide la cadena de entrada en un conjunto de subcadenas de longitud gradualmente incremental. En cada paso de la codificación, a la subcadena más larga del texto sin analizar que empareje con una entrada Υ existente en el diccionario se le asigna un código (Υ, c), siendo c el símbolo siguiente a dicha subcadena en el texto, y se añade al diccionario.

Ejemplo 2.4.1 Una cadena de entrada de la forma $ABABAAABA$ da lugar a una codificación como se muestra en la Tabla 2.3. Al principio el diccionario contiene una única entrada con la subcadena nula. El algoritmo comienza leyendo la subcadena A y buscando en el diccionario alguna entrada que la contenga. Como no hay ninguna crea una nueva entrada en el diccionario formada por esta subcadena y le asigna un código (0, A). Este código representa la cadena "null A" (una concatenación de la cadena nula correspondiente a la entrada 0 con la subcadena A).

En el siguiente paso, el algoritmo lee la subcadena B y realiza la misma operación.

Posteriormente extrae la subcadena AB porque encuentra la subcadena A en el diccionario (entrada 1) pero no encuentra ninguna subcadena AB. Crea entonces la nueva entrada AB y le asigna un código (1, B) indicando que la subcadena actual es la concatenación de la de la entrada 1 (A) y el símbolo B.

Este proceso se realiza sucesivamente hasta finalizar el análisis del texto. □

En cada paso de la codificación, la subcadena añadida al diccionario está formada por una subcadena ya codificada y un nuevo símbolo; por lo tanto, cada nueva entrada será de igual o mayor longitud que las anteriores. El tamaño máximo que puede alcanzar el diccionario puede ser prefijado de antemano o estar limitado a la cantidad de memoria disponible. Cuanto mayor sea este tamaño se podrán almacenar subcadenas más largas

entrada	subcadena de entrada	Codif.Ziv-Lempel
0	null	
1	A	(0, A)
2	B	(0, B)
3	AB	(1, B)
4	AA	(1, A)
5	ABA	(3, A)

Tabla 2.3: Ejemplo de codificación LZ78

permitiendo emparejamientos de mayor longitud, pero también aumenta el número de punteros lo que hace más lenta la búsqueda sobre el diccionario.

Los métodos Ziv-Lempel ofrecen en general una buena compresión, son rápidos y no requieren mucha memoria. Sin embargo, resultan ineficientes para documentos pequeños, pues comienzan asignando códigos más largos que otros métodos vistos. Además, al tratarse de una codificación de longitud fija, si el texto original no es lo suficientemente grande la eficiencia del método podría reducirse considerablemente.

Se trata de una codificación asintóticamente óptima, lo que significa que la redundancia tiende a 0 cuando el tamaño del vocabulario de entrada tiende a infinito.

2.5. La compresión de textos en lenguaje natural

Los métodos clásicos presentados en la sección anterior no son adecuados para bases de datos textuales. La codificación Huffman binaria es poco utilizada por su escaso ratio de compresión. Por otro lado, el método Ziv-Lempel obtiene una buena compresión (entre un 30 % y un 40 % en la práctica) pero la búsqueda de una palabra sobre el texto comprimido es muy ineficiente. Así, los estudios empíricos realizados [2, 3, 28] demuestran que la búsqueda sobre textos comprimidos bajo codificación Ziv-Lempel se puede llevar a cabo en la mitad de tiempo que realizar la descompresión del texto y búsqueda posterior, pero es el doble de lenta que la búsqueda directamente

sobre el texto sin comprimir. Dado que uno de los retos principales de los sistemas de Recuperación de Textos es proporcionar un acceso rápido a los datos, se necesita mejorar el rendimiento que ofrecen las técnicas mencionadas.

En 1989, Moffat propuso la idea de utilizar las palabras como los símbolos a comprimir en lugar de los caracteres [26, 40]. El motivo principal del planteamiento es que un texto es mucho más fácil de comprimir cuando se considera como una secuencia de palabras y no como un conjunto de caracteres. Además, dado que las palabras son consideradas átomos por los métodos de Recuperación de Textos, una estrategia de compresión basada en palabras resulta particularmente adecuada en este entorno.

Actualmente se han desarrollado nuevas técnicas de compresión basadas en palabras como las presentadas por Manber [23], Turpin [39] o Moura *et al.* [37, 45]. Sobre los textos comprimidos con estas técnicas es posible realizar búsquedas directas, evitando así la necesidad de descomprimir antes de buscar.

A continuación, se presenta con detalle el método de Moura *et al.* por ser el que más se asemeja a la nueva técnica de compresión desarrollada en este trabajo. Nuestra codificación conserva las principales ventajas de esta técnica utilizando algoritmos más sencillos y rápidos para la codificación, compresión y descompresión, como se verá en los siguientes capítulos.

2.5.1. Compresión Huffman orientada a byte basada en palabras

En [37, 45], Moura *et al.* presentan una técnica de compresión basada en palabras que utiliza codificación Huffman. Resultados empíricos han demostrado que, desde el punto de vista de la compresión, las técnicas que emplean codificación Huffman basada en caracteres reducen el tamaño de textos en inglés a aproximadamente un 60 % de su tamaño original, mientras que los métodos que emplean codificación Huffman basada en palabras pueden alcanzar un porcentaje de compresión próximo al 30 %. Esto es debido a que la distribución de palabras es considerablemente más sesgada que la de caracteres pues, en la mayoría de los textos en lenguaje

natural, existe un conjunto de palabras que se repiten mucho mientras que las restantes tienen una probabilidad de aparición mucho menor.

Otra de las ventajas de la utilización de compresión basada en palabras es que esta técnica permite la búsqueda eficiente de palabras y frases sobre el texto comprimido, sin necesidad de descomprimirlo. En [37] se demuestra que las búsquedas sobre textos comprimidos con estas técnicas pueden llegar a ser 8 veces más rápidas (para ciertas consultas) que utilizando herramientas de búsqueda (como *Agrep*[41]) sobre el texto sin comprimir.

La técnica de Moura *et al.* utiliza un modelo *semiadaptativo*, como sucede con la técnica de compresión Huffman clásica. Es decir, el codificador hace una primera pasada sobre el texto para obtener la frecuencia de todas las palabras y, en una segunda pasada, realiza la codificación. Durante esta última fase, se sustituyen los símbolos de entrada (palabras en este caso) por sus códigos. Para cada palabra en el texto habrá un único código cuya longitud varía dependiendo de la frecuencia de aparición de la palabra en el texto. Utilizando el algoritmo de compresión de Huffman, se asignan los códigos más cortos a las palabras más frecuentes y los códigos más largos a las menos frecuentes.

Otra diferencia con el método clásico de Huffman está en los símbolos utilizados para la codificación. Ahora el código Huffman asignado a cada palabra tiene uno o más bytes en lugar de un número variable de bits. Las pruebas experimentales realizadas (ver [37, 45]) demuestran que no existe una degradación significativa en el porcentaje de compresión utilizando bytes en lugar de bits, mientras que la búsqueda y descompresión son más rápidas ya que se evita la manipulación de bits.

Moura *et al.* proponen dos tipos de codificación. La primera de ellas, denominada *Plain Huffman*, consiste en la aplicación del algoritmo Huffman utilizando como símbolos de salida los $d = 2^8 = 256$ valores que puede tomar un byte. Por lo tanto, emplea en la codificación los $b = 8$ bits que componen un byte, de modo que el árbol Huffman resultante tiene grado 256.

Aplicando codificación Plain Huffman se consiguen buenos ratios de compresión, como se ha mencionado, pero no resulta muy eficiente a la hora de realizar búsquedas directamente sobre el texto comprimido. El problema

es que aunque se trata de una codificación de prefijo libre (y, por lo tanto, ningún código es prefijo de otro) no se puede reconocer dónde comienza cada código dentro de la cadena codificada. Por lo tanto, la decodificación debe comenzar desde el principio del texto (o, al menos, desde algún punto del texto comprimido donde se identifique el comienzo de un código).

Existe una segunda codificación, denominada *Tagged Huffman*, que se caracteriza por la utilización de marcas para permitir la localización de los códigos en el texto comprimido de forma inmediata. Para ello, se reserva el bit más significativo de cada byte como *bit de marca* para señalar el comienzo de cada código comprimido. En concreto, el primer byte de cada código tiene su bit de marca a 1, mientras que los bytes restantes tienen su bit de marca a 0. Por lo tanto, se emplean únicamente $b = 7$ bits de cada byte para la codificación, reservando el octavo como marca. Sobre estos 7 bits se continúa aplicando codificación Huffman, pues el bit de marca por sí mismo no hace que el código sea de prefijo libre. Como consecuencia, el árbol Huffman resultante ahora tiene grado 128.

La utilización de un bit de marca facilita las búsquedas comenzando en cualquier punto del texto comprimido y la descompresión de porciones del texto. De este modo es posible realizar, por ejemplo, una rápida descompresión de los fragmentos que contienen los resultados de una búsqueda, aspecto de especial interés en los sistemas de Recuperación de Textos.

Esta importante ventaja tiene, sin embargo, un coste en términos de rendimiento en la compresión: se almacenan bytes completos pero sólo se emplean 7 bits para la codificación. Como consecuencia de ello el tamaño del fichero comprimido crece aproximadamente en un 11 % respecto a la aproximación Plain Huffman [37].

En los siguientes apartados se muestran ciertas particularidades de la codificación Huffman orientada a byte para, posteriormente, presentar varios ejemplos donde se reflejan las diferencias entre las codificaciones Plain y Tagged Huffman.

Proceso de codificación

Para realizar el proceso de codificación es necesario construir un árbol Huffman no binario. En concreto, el número de símbolos que componen el vocabulario de salida es $d = 2^b = 2^8 = 256$ en caso de la codificación Plain Huffman, y $d = 2^{b-1} = 2^7 = 128$ en caso de la codificación Tagged Huffman, donde b representa el número de bits empleado para la codificación.

La construcción de un árbol no binario exige tener en cuenta ciertos detalles para evitar que los primeros niveles del árbol no tengan nodos vacíos. Veámoslo con un ejemplo.

Ejemplo 2.5.1 Supongamos que se desea aplicar codificación Huffman sobre un texto cuyo vocabulario de entrada son 512 palabras de igual probabilidad utilizando un vocabulario de salida formado por 256 símbolos (del 0 al 255). En este caso, si aplicamos directamente el algoritmo de Huffman, se crearán 2 nodos padre (uno para cada 256 palabras) y estos, a su vez, (junto con 254 nodos vacíos) darán lugar al nodo raíz (ver Figura 2.4 (A)).

De este modo, la longitud de los códigos para las 512 palabras será de 2 bytes, y no habrá ningún código de longitud 1. □

Como se puede observar en la Figura 2.4 (A), el nodo raíz tiene 254 espacios vacíos que podrían haber sido ocupados por palabras que se encuentran en el segundo nivel del árbol, de modo que las longitudes de sus códigos se reducirían de 2 bytes a 1.

Una forma de asegurar que los nodos vacíos siempre se encuentran en el nivel más profundo del árbol (y, por lo tanto, les corresponderán los códigos más largos) es conocer previamente el número de nodos vacíos, para poder combinar éstos con los símbolos de entrada (palabras) de menor probabilidad. El número de palabras que se deben combinar con los nodos vacíos se calcula como

$$1 + ((v - d) mod (d - 1))$$

siempre que $((v - d) mod (d - 1)) > 0$

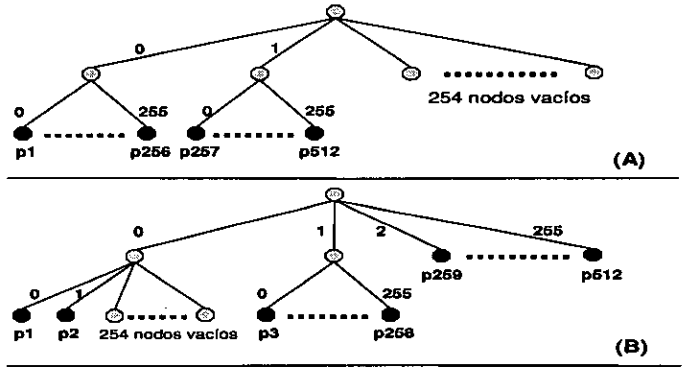


Figura 2.4: Ejemplo de árbol Huffman no binario

Ejemplo 2.5.2 Para el ejemplo anterior, el número de palabras que se deben combinar en la primera iteración del algoritmo de Huffman es:

$$1 + ((512 - 256) mod (256 - 1)) = 2$$

Por lo tanto, en el primer paso se combinarán 2 palabras con 254 nodos vacíos, en el segundo paso se calculará el padre de las 256 palabras siguientes y, por último, se tomarán las 254 palabras restantes junto con los 2 nodos padre generados para dar lugar al nodo raíz del árbol (ver Figura 2.4 (B)). □

Ejemplos de codificación Plain y Tagged Huffman

Los ejemplos que se presentan a continuación muestran el resultado de aplicar las codificaciones Plain y Tagged Huffman sobre dos distribuciones de palabras. En el primer ejemplo se considera que todas las palabras tienen la misma probabilidad de aparición mientras que en el segundo siguen una distribución exponencial.

Por simplicidad, se supone que cada "byte" está formado únicamente por

$b = 3$ bits. Por lo tanto, el esquema Plain Huffman emplea los 3 bits para la codificación ($d = 2^3 = 8$ símbolos), mientras que Tagged Huffman utiliza 1 bit como marca y 2 para la codificación ($d = 2^{3-1} = 4$ símbolos). Los bits de marca aparecen resaltados.

Las distribuciones elegidas son hipotéticas y no se corresponden a lo que sucede habitualmente en los textos en lenguaje natural. Sin embargo, se utilizan aquí por tratarse de casos extremos a través de los que se pueden observar claramente las diferencias entre ambas codificaciones.

Ejemplo 2.5.3 La Tabla 2.4 muestra las diferencias entre los códigos generados por los esquemas de codificación Plain y Tagged Huffman para una distribución de probabilidad uniforme (es decir, $p_i = p_j \ \forall_{i,j}$, donde p_k es la probabilidad de la k-ésima palabra del vocabulario). El vocabulario de entrada está formado por 16 palabras. En la Figura 2.5 se representan los árboles Huffman resultantes para ambos esquemas de codificación.

Como puede observarse en la Figura 2.5, el número de palabras combinadas con nodos vacíos bajo el algoritmo Plain Huffman han sido:

$$1 + ((v - d) mod (d - 1)) = 1 + ((16 - 8) mod (8 - 1)) = 2$$

En caso de la codificación Tagged Huffman no hay nodos vacíos, puesto que

$$((v - d) mod (d - 1)) = ((16 - 4) mod (4 - 1)) = 0$$

Por lo tanto, en la primera iteracción del algoritmo se combinan los $d = 4$ símbolos de menor probabilidad. □

Ejemplo 2.5.4 La Tabla 2.5 muestra los códigos obtenidos aplicando de nuevo ambos esquemas de codificación pero, en este caso, considerando una distribución exponencial (de la forma $p_i = 2^{-i}$). Sin pérdida de generalidad, se ha incrementado la frecuencia de la palabra menos probable para que las probabilidades sumen 1.

En la Figura 2.6 se pueden observar los árboles Huffman asociados. □

Palabra	Probab.	Plain Huffman	Tagged Huffman
A	1/16	000 000	100 000
B	1/16	000 001	100 001
C	1/16	000 010	100 010
D	1/16	000 011	100 011
E	1/16	000 100	101 000
F	1/16	000 101	101 001
G	1/16	000 110	101 010
H	1/16	000 111	101 011
I	1/16	001 000	110 000
J	1/16	001 001	110 001
K	1/16	010	110 010
L	1/16	011	110 011
M	1/16	100	111 000
N	1/16	101	111 001
O	1/16	110	111 010
P	1/16	111	111 011

Tabla 2.4: Codificaciones Plain Huffman y Tagged Huffman para una distribución uniforme de palabras

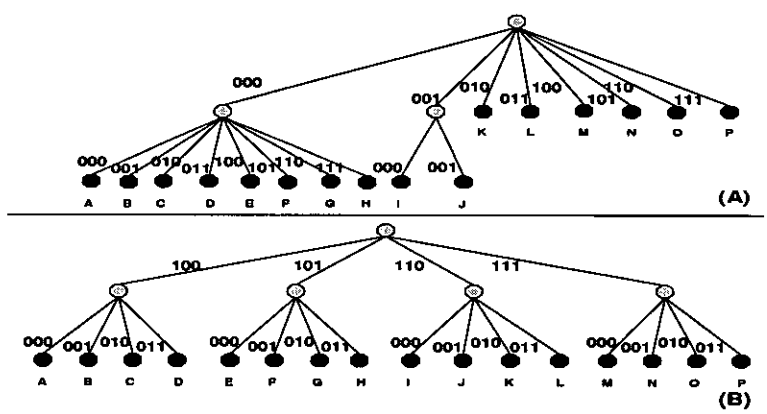


Figura 2.5: Árboles Plain Huffman y Tagged Huffman para la distribución uniforme de la Tabla 2.4

Palabra	Probab.	Plain Huffman	Tagged Huffman
A	1/2	111	111
B	1/4	110	110
C	1/8	101	101
D	1/16	100	100 011
E	1/32	011	100 010
F	1/64	010	100 001
G	1/128	001	100 000 011
H	1/256	000 111	100 000 010
I	1/512	000 110	100 000 001
J	1/1024	000 101	100 000 000 011
K	1/2048	000 100	100 000 000 010
L	1/4096	000 011	100 000 000 001
M	1/8192	000 010	100 000 000 000 011
N	1/16384	000 001	100 000 000 000 010
O	1/32768	000 000 001	100 000 000 000 001
P	1/32768	000 000 000	100 000 000 000 000

Tabla 2.5: Codificaciones Plain Huffman y Tagged Huffman para una distribución exponencial de palabras

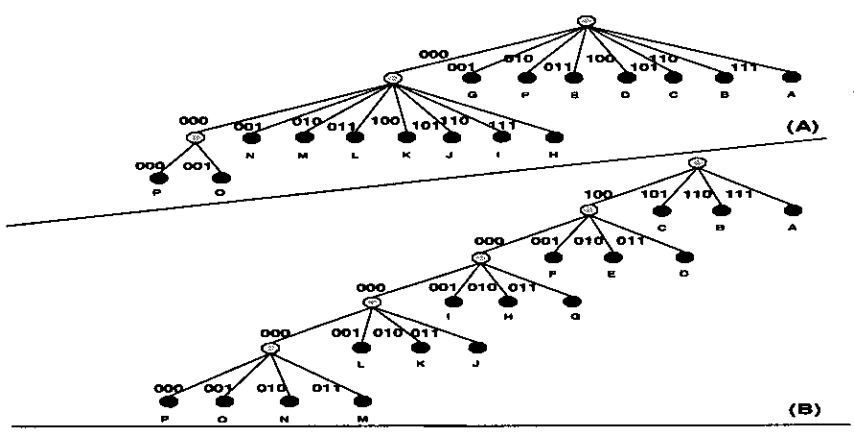


Figura 2.6: Árboles Plain Huffman y Tagged Huffman para la distribución exponencial de la Tabla 2.5

Como se puede observar en las Tablas 2.4 y 2.5, cuando se aplica la codificación Tagged Huffman el codificador emplea únicamente 2^{b-1} de los $d = 2^b$ posibles valores que puede tomar un byte formado por b bits. En concreto, el primer byte de un código debe comenzar siempre con un bit a 1 para indicar que es comienzo de código, por lo que sus valores posibles oscilan entre 100 y 111 en el ejemplo anterior (en el caso real ($b = 8$) serían 10000000 y 11111111). El segundo byte y sucesivos deben comenzar siempre con un bit a 0 para indicar que no son comienzo de código. Por lo tanto, los valores para estos bytes oscilan entre 000 y 011 para el ejemplo anterior (en el caso real los valores estarían entre 00000000 y 01111111).

2.6. Búsquedas sobre texto comprimido

Las nuevas técnicas de compresión de textos en lenguaje natural basadas en palabras no han sido desarrolladas únicamente para alcanzar mejores ratios de compresión que las técnicas de compresión tradicionales (como Shannon-Fano, Huffman, aritmética o Ziv-Lempel) sino también para aumentar la eficiencia de las búsquedas, permitiendo su ejecución directamente sobre el texto comprimido sin necesidad de descomprimirlo.

Esta sección se dedica a presentar los diferentes tipos de búsqueda que se pueden aplicar sobre textos codificados mediante técnicas de compresión basadas en palabras, como la explicada en la sección anterior. Estos tipos de búsqueda resultan de gran interés aquí ya que también pueden aplicarse directamente sobre los textos comprimidos con la nueva codificación Densa con Post-Etiquetado desarrollada en este trabajo.

Previamente se dedica un primer apartado al estudio de los principales algoritmos de *emparejamiento de patrones* por ser los más habituales para la realización de búsquedas sobre textos. La técnica de emparejamiento de patrones se basa en el concepto de *patrón* y permite recuperar porciones de texto que cumplen una cierta propiedad.

La búsqueda por patrones puede utilizarse como herramienta para facilitar la realización de consultas por palabra u otras más complejas (por frase o por proximidad), que forman lo que se denominan *consultas básicas*.

Las consultas básicas, y sus combinaciones utilizando expresiones booleanas, se presentan en el Apartado 2.6.2.

Finalmente, en esta sección se estudian los algoritmos de *emparejamiento comprimido* que permiten la realización de consultas básicas directamente sobre textos comprimidos utilizando técnicas basadas en palabras (concretamente, mediante las codificaciones Plain y Tagged Huffman).

2.6.1. Emparejamiento de patrones (*Pattern Matching*)

La técnica de consulta mediante emparejamiento de patrones consiste en localizar todas las ocurrencias (o la primera) de un patrón m en un documento [14, 20, 38]. Un *patrón* define un conjunto de características sintácticas que deben ocurrir en el texto. La cadena de texto que satisfaga estas características se dice que "empareja" con el patrón.

Los patrones pueden ser muy simples (palabras, por ejemplo) o complejos (como expresiones regulares). Los tipos de patrones más habituales son:

- palabras: secuencia de caracteres que debe dar lugar a una palabra en el texto.
- prefijos: cadena con la que debe comenzar una palabra. Ej: *cant* devolvería documentos que contengan palabras como *cantaba* o *cantante*.
- sufijos: cadena con la que debe finalizar una palabra. Ej: *ión* devolvería los documentos que contengan *camión*, *nación* o *pasión*.
- subcadenas: cadena que puede aparecer dentro de una palabra. Ej: *enc* devolvería textos que incluyan palabras como *eficiencia*, *reticencia*, *encuadernar* o *creencia*. La subcadena puede incluir separadores de palabras (",", blanco, ";", ":", etc.). Ej: *ado par* devolvería documentos que incluyan *analizado parcialmente*, *estudiado particularmente* o *citado para*.
- rangos: en este caso el patrón está formado por dos cadenas, de modo que se obtienen los documentos que contengan palabras que se

encuentren (alfabéticamente) entre las mismas. Ej: *lado lodo* devuelve textos con palabras como *lamer*, *lectura* o *local*.

- permitiendo errores: este tipo de búsqueda resulta de interés cuando se intenta evitar errores tipográficos (de tecleo, fallos de OCR, etc.). En este caso se recuperan los documentos que incluyen palabras "similares" al patrón. Aunque existen varios modelos de similitud entre palabras, el más empleado en el campo de la Recuperación de Textos es la *distancia de edición*. La distancia entre dos cadenas es el número mínimo de inserciones, borrados y sustituciones necesarias para hacerlas iguales. De este modo, la consulta debe especificar el número máximo de errores permitidos para que una palabra empareje con el patrón. Ej: una consulta *casa (distancia 1)* devuelve documentos con cadenas de la forma *ca sa*, *cosa* o *cara*.

- expresiones regulares: una expresión regular es un patrón más general formado por cadenas simples y los operadores:

  - unión: si e_1 y e_2 son expresiones regulares, $(e_1|e_2)$ devuelve las cadenas emparejadas con e_1 o con e_2.
  - concatenación: si e_1 y e_2 son expresiones regulares, las ocurrencias de (e_1e_2) están formadas por las ocurrencias de e_1 seguidas inmediatamente por las de e_2. De este modo, las cadenas simples pueden verse como la concatenación de caracteres simples.
  - repetición: si e es una expresión regular, (e^*) empareja con secuencias de 0 o más ocurrencias contiguas de e.

  Ej. de expresión regular: *prob(blema|teina)(s|ϵ)(0|1|2)** (donde ϵ representa la cadena vacía) devuelve documentos que contengan las palabras *problemas* o *proteina02*.

- patrones extendidos: son subconjuntos de expresiones regulares que se expresan con una sintaxis más simple. Como cada sistema tiene sus propios patrones extendidos no hay una definición formal para los mismos.

- expresiones condicionales: parte del patrón puede o no aparecer.

- combinaciones para permitir que algunas partes del patrón emparejen exactamente y otras admitan errores.

Algoritmos de emparejamiento de patrones

A lo largo de los años se han implementado diferentes algoritmos que permiten el emparejamiento de patrones sobre un texto. Entre ellos podemos destacar los que se presentan a continuación (ver [7, 14, 38] para más detalle sobre los mismos).

Para una mejor explicación de los algoritmos, denominaremos $m[i]$ al i-ésimo carácter de un patrón $m = m[0]...m[k-1]$ cuya longitud es k, y llamaremos $t[j]$ al j-ésimo carácter de una cadena de texto $t = t[0]...t[n-1]$ de longitud $n > k$. Se dice que el patrón m está alineado con el texto t en la posición x si el carácter $m[0]$ está alineado con $t[x]$, $m[1]$ con $t[x+1]$ y $m[i]$ con $t[x+i]$ para todo $i \leq (k-1)$. Es decir,

TEXTO:	...	$t[x-1]$	$t[x]$	$t[x+1]$	$t[x+2]$	...	$t[x+k-1]$	$t[x+k]$	...
patrón:			$m[0]$	$m[1]$	$m[2]$	...	$m[k-1]$		

Además, utilizaremos como ejemplo de todos los algoritmos que se presentan a continuación el caso en que queremos localizar el patrón $m =$ *alababa* en un texto $t =$ *alalalababa*.

El algoritmo de emparejamiento de patrones más simple, denominado *straightforward(SF)* o *fuerza bruta*, consiste en alinear el patrón m al extremo izquierdo del texto (es decir, a la posición $x = 0$) y comparar los caracteres del patrón $m[0]m[1]...m[k-1]$ con los caracteres del texto $t[x]t[x+1]...t[x+k-1]$ de izquierda a derecha. En caso de que se detecte algún error de emparejamiento, el patrón se desplaza una posición a la derecha incrementando x en 1, y comienza de nuevo el análisis desde el carácter $m[0]$ del patrón.

Ejemplo 2.6.1 Algoritmo de búsqueda SF:

TEXTO:	a	l	a	l	a	l	a	b	a	b	a
paso 1.	a	l	a	**b**							
paso 2.		a									
paso 3.			a	l	a	**b**					
paso 4.				a							
paso 5.					a	l	a	b	a	b	a

En cada iteración del algoritmo el patrón se desplaza una posición a la derecha hasta que se produce el emparejamiento completo. □

Se han desarrollado algoritmos más eficientes que esta aproximación simple. Dos de los más conocidos son el de Knuth-Morris-Pratt (KMP) y el de Boyer-Moore (BM). Ambos algoritmos tienen un comportamiento lineal en el peor caso, frente a la complejidad cuadrática del algoritmo SF.

El *algoritmo Knuth-Morris-Pratt (KMP)*[20] trabaja del mismo modo que SF, pero cuando se detecta un error de emparejamiento, entre $m[i]$ y $t[x+i]$ por ejemplo, en lugar de hacer un único desplazamiento hacia la derecha, el patrón se mueve δ posiciones hasta alinear el prefijo emparejado $m[0]m[1]...m[i-1]$ (o una parte) con su siguiente ocurrencia en la subcadena de texto ya analizada $t[x]t[x+1]...t[x+i-1]$. Una vez realizado el desplazamiento, el análisis continúa a partir del elemento $m[i]$ del patrón que produjo el error.

Ejemplo 2.6.2 Algoritmo de búsqueda KMP

TEXTO:	a	l	a	l	a	l	a	b	a	b	a
paso 1.	a	l	a	**b**							
paso 2.			a	l	a	**b**					
paso 3.					a	l	a	b	a	b	a

En este caso, en el paso 2 se producen $\delta = 2$ desplazamientos porque no hay nuevas ocurrencias de $m[0]m[1]m[2] = ala$ ni de $m[0]m[1] = al$ en la cadena de texto $t[0]t[1]t[2] = ala$ analizada, pero sí de $m[0] = a$ que se encuentra en $t[2]$. □

El *algoritmo Boyer-Moore (BM)* [9, 30] cambia la dirección de análisis del patrón, haciéndolo en este caso de derecha a izquierda (es decir, desde la posición $m[k-1]$ hasta $m[0]$). Al igual que KMP, cuando se produce un error, en la posición $m[k-i]$ por ejemplo, utiliza la información de los emparejamientos anteriores para realizar el mayor desplazamiento posible del patrón m hacia la derecha. Para ello, se localizan en el patrón dos posiciones (partiendo siempre desde $m[k-i]$ hacia $m[0]$):

- δ_1 : primera ocurrencia del carácter del texto que produjo el error (es decir, $t[x+k-i]$) que se encuentre a la izquierda de $m[k-i]$).
- δ_2 : siguiente ocurrencia del sufijo del patrón ya emparejado $m[k-i+1]...m[k]$ de modo que su carácter a la izquierda no sea el que produjo el error ($m[k-i]$).

El desplazamiento se hará en base al mayor de ambos valores. Veámoslo con un ejemplo.

Ejemplo 2.6.3 Algoritmo de búsqueda BM

TEXTO:	a	l	a	l	a	l	a	b	a	b	a
paso 1.	a	**l**	a	b	a	**b**	a				
paso 2.					a	l	a	b	a	b	a

En este caso, después de emparejar el sufijo $m[6] = a$ del patrón se ha producido un fallo de emparejamiento por el símbolo $t[5] = l$. Entonces se calcula δ_1 como el número de posiciones (4, en este caso) donde se encuentra la siguiente ocurrencia de l en el patrón (partiendo desde $m[5]$ hacia $m[0]$). Por otro lado, se calcula δ_2 como el número de posiciones donde se ubica la siguiente ocurrencia de $m[6] = a$ que no esté precedida por $m[5] = b$ (partiendo igualmente desde $m[5]$ hacia $m[0]$). La ocurrencia se encuentra en $m[2]$ por lo que δ_2 también vale 4.

Finalmente, el patrón se desplaza máx$[\delta_1, \delta_2] = 4$ posiciones. □

En la práctica, sobre textos básicos en inglés, el algoritmo BM es unas 3 veces más rápido que SF y KMP, y mejora cuanto más grande sea el patrón de búsqueda.

Por último, presentamos los *algoritmos de Sunday* [38] que están basados en BM. En general, Sunday utiliza el mismo valor de δ_2 pero el valor de δ_1 cambia. Ahora, δ_1 es la distancia a la siguiente ocurrencia de $t[x+k]$ en el patrón, comenzando desde la posición $m[k-1]$ hacia $m[0]$.

Ejemplo 2.6.4 Algoritmo de búsqueda de Sunday

TEXTO:	a	l	a	l	a	l	a	**b**	a	**b**	a
paso 1.	a	l	a	b	a	**b**	a				
paso 2.					a	l	a	b	a	b	a

Tras el fallo de emparejamiento entre el carácter $t[5] = l$ y $m[5] = b$ se calculan los valores de δ_1 y δ_2.

En este caso, δ_1 vale 2, porque la primera ocurrencia de $t[7] = b$ en el patrón (partiendo de $m[6]$ hacia $m[0]$) está en $m[5]$.

Por otro lado, δ_2 vale 4 ya que la siguiente ocurrencia de $m[6] = a$ cuyo carácter a su izquierda no sea $m[5] = b$ está en la posición $m[2]$.

Por lo tanto, el número de desplazamientos es máx$[\delta_1,\ \delta_2]$ = máx$[2,\ 4] = 4$. □

2.6.2. Realización de consultas básicas

Mediante emparejamiento de patrones es posible realizar distintos tipos de búsquedas como las que se detallan a continuación.

El modo más simple de formular una consulta es solicitando aquellos documentos que incluyen una palabra o un conjunto de palabras concreto. La búsqueda por *palabra clave* es la más habitual ya que es intuitiva, fácil de formular y permite una consulta rápida. Algunos sistemas potencian esta búsqueda incorporando la capacidad de localizar palabras en un contexto determinado, es decir, que se encuentren próximas. En este caso podemos distinguir dos tipos de consultas: *por frase*, donde una frase es una secuencia de palabras, o *por proximidad* de modo que se permite que entre las palabras haya una cierta distancia.

Ejemplo 2.6.5 Un caso de búsqueda por frase sería localizar los documentos donde aparezca exactamente *sistema operativo*.

Una búsqueda por proximidad sería localizar los textos donde aparezcan las palabras *compresión* y *textos* con una distancia inferior a 5 palabras. En este caso, se recuperarían documentos con ocurrencias del tipo *compresión de textos* o *compresión y búsqueda sobre textos*. □

El resultado obtenido mediante las consultas de palabra clave puede mejorarse utilizando operadores booleanos. Una *consulta booleana* está compuesta por átomos (consultas básicas del tipo palabra clave) y un conjunto de operadores booleanos (OR, AND y NOT).

Sin embargo, las consultas booleanas son muy restrictivas cuando se aplican sobre grandes bases de datos textuales. No admiten un emparejamiento parcial entre un documento y una consulta. Así, *palabra1 AND palabra2* exige la existencia de las dos palabras, *palabra1 OR palabra2* la aparición de al menos una de ellas, mientras que *palabra1 NOT palabra2* recupera únicamente aquellos documentos que contienen *palabra1* pero no *palabra2*.

Además, es ampliamente aceptado que los usuarios no informáticos tienen dificultades para utilizar los operadores booleanos. Por ello se han propuesto un conjunto de operadores booleanos *difusos*. La idea es que los operadores AND y OR sean menos restrictivos, de modo que en vez de forzar a que aparezcan todos los operandos (AND) o al menos uno (OR), se recuperan los documentos donde aparezcan algunos operandos (en este caso, AND podría requerir más coincidencias que OR).

Tanto las búsquedas simples como las booleanas y difusas se pueden aplicar sobre textos utilizando los algoritmos de emparejamiento de patrones anteriores, pero para su utilización sobre textos comprimidos se precisan nuevos algoritmos, denominados *de emparejamiento comprimido*, como los que se presentan en el siguiente apartado.

2.6.3. Búsqueda sobre texto comprimido sin necesidad de descomprimirlo

El *problema del emparejamiento comprimido* se define como la tarea de realizar emparejamiento de patrones sobre texto comprimido sin descomprimirlo. Dado un texto T, la cadena comprimida correspondiente Z, y un patrón no comprimido M, el problema del emparejamiento comprimido consiste en localizar todas las ocurrencias de M en T, utilizando únicamente M y Z.

El modo de realizar las búsquedas sobre textos comprimidos utilizando la técnica Huffman orientada a byte basada en palabras difiere según el tipo de codificación elegida (Plain o Tagged Huffman). Bajo codificación Tagged Huffman es posible realizar búsquedas comenzando en cualquier punto del texto comprimido gracias a la existencia del bit de marca. Basta con comprimir el patrón de búsqueda y aplicar un algoritmo convencional de emparejamiento de patrones, como los presentados en el apartado anterior.

Esto no es posible mediante codificación Plain Huffman ya que el patrón puede existir en el texto y, sin embargo, no tratarse de un código. Este problema se origina debido a que la concatenación de partes de códigos consecutivos puede ser igual al código correspondiente a otra palabra del vocabulario. Veámoslo con un ejemplo.

Ejemplo 2.6.6 Supongamos que se desea comprimir un texto cuyo vocabulario está formado por las palabras *A, B, C, D*, y supongamos que el algoritmo Huffman asigna los códigos de la Tabla 2.6 a las palabras originales. (De nuevo, por claridad hemos considerado bytes "especiales" de 3 bits). Puede observarse que este código es de prefijo libre y puede obtenerse aplicando el algoritmo de Huffman.

Consideremos ahora la siguiente porción de texto comprimido, bajo la codificación anterior, para la secuencia *ABAD*:

$$\ldots 000\ 010\ \mathbf{000}\ 110\ \mathbf{000}\ldots$$

Por último, supongamos que se desea localizar las ocurrencias de la

A	000
B	010
C	100
D	110 000

Tabla 2.6: Ejemplo de codificación Huffman basada en palabras

palabra *A*. Si la búsqueda comenzara en el tercer byte (000), el algoritmo de emparejamiento de patrones detectaría dos ocurrencias de *A* en el texto (aparecen resaltadas en el ejemplo). Sin embargo, la segunda no representa realmente una ocurrencia de *A* sino que es parte del código correspondiente a *D*. Se debe, por tanto, incluir un procesamiento extra que verifique cada ocurrencia potencial, hecho que arruina claramente la simplicidad y el rendimiento esperado. □

En [37, 45] Moura *et al.* presentan tres algoritmos que permiten la búsqueda directamente sobre textos comprimidos mediante las codificaciones Plain y Tagged Huffman, y se explican a continuación.

Algoritmos de búsqueda

El algoritmo más sencillo propuesto por Moura *et al.* se puede aplicar sobre texto comprimido bajo codificación Tagged Huffman. Comienza comprimiendo el patrón de búsqueda, es decir, localizando cada palabra buscada en el vocabulario del texto para obtener el código Huffman que la representa. En general, esta búsqueda es muy eficiente ya que el vocabulario suele ser pequeño comparado con el tamaño del texto. Posteriormente se busca el código obtenido en el texto comprimido mediante algún algoritmo convencional de emparejamiento de patrones. La búsqueda puede comenzar en cualquier punto del texto pues, como ya se ha mencionado, los códigos se reconocen porque su primer byte tiene el bit de marca a 1. Esta técnica de búsqueda se denomina *búsqueda directa*, y puede extenderse para permitir otras consultas básicas (por frase, por proximidad y booleanas) sin suponer un coste computacional extra. Para realizar búsquedas más complejas (empleando expresiones regulares o con patrones extendidos) es necesario

obtener una lista de los códigos correspondientes a todos los símbolos del vocabulario de entrada (las palabras) que emparejan con el patrón, y utilizar un algoritmo de búsqueda multi-patrón como el propuesto por Baeza-Yates *et al.* [6]. Este algoritmo es una extensión del de Sunday presentado en la sección anterior.

El segundo algoritmo permite la realización de búsquedas sobre códigos Plain Huffman y se basa en el algoritmo *Shift-Or orientado a palabras* [5]. Este algoritmo simula un autómata no determinista cuyos estados son las palabras a buscar y dondè la transición entre estados se produce cada vez que se reconoce una palabra del patrón de búsqueda en el texto.

El tercer algoritmo es una combinación de los anteriores, y se basa en comprimir el patrón y realizar la búsqueda directamente sobre el texto comprimido (como en el primer algoritmo) pero se puede aplicar sobre códigos Plain Huffman. Para ello, utiliza el segundo algoritmo que analiza el área circundante para validar los emparejamientos detectados. Para alcanzar un mejor rendimiento, en el momento de la compresión los textos se dividen en *bloques*. Los códigos se alinean al comienzo de los bloques, de modo que nunca un código forma parte de dos bloques. De este modo, sólo se descomprime el texto desde el comienzo del bloque en que supuestamente se haya encontrado un emparejamiento.

Hoy en día, los sistemas de Recuperación de Textos precisan mayor flexibilidad a la hora de realizar búsquedas. Resulta de gran interés la búsqueda aplicando expresiones regulares, permitiendo errores o utilizando patrones extendidos (mencionados anteriormente). En [37, 45] se describe con más detalle las posibilidades de aplicación de estos tipos de búsquedas sobre textos comprimidos mediante las codificaciones Plain y Tagged Huffman utilizando los tres algoritmos anteriores.

2.7. Conclusiones del capítulo

Este capítulo se ha dedicado a realizar una revisión de las principales técnicas de compresión de textos. Entre ellas, caben destacar las diferentes codificaciones Huffman empleadas para comprimir texto:

Codificación Huffman binaria basada en caracteres [19]: Se asigna una secuencia de bits a cada carácter del texto original. Se asignan los códigos a los caracteres originales aplicando el algoritmo de Huffman, de modo que a los símbolos más frecuentes les corresponden los códigos más cortos.

Codificación Huffman binaria basada en palabras [37]: Se asigna una secuencia de bits a cada palabra del texto original. Se asignan los códigos a las palabras originales aplicando el algoritmo de Huffman.

Cuando se aplica sobre textos en lenguaje natural, este método consigue mejores ratios de compresión que el anterior. El motivo es que la codificación Huffman resulta más ventajosa cuando existe un número de símbolos a codificar de alta probabilidad respecto a los restantes, y esto es lo que sucede con las palabras pues, generalmente, su distribución es mucho más sesgada que la de los caracteres.

Codificación Plain Huffman [37]: Se asigna una secuencia de bytes a cada palabra del texto original, en lugar de una secuencia de bits. Para la codificación se emplean los 8 bits que componen cada byte. La asignación de códigos se realiza aplicando el algoritmo de Huffman. El árbol de Huffman tiene grado 256.

Codificación Tagged Huffman [37]: Se asigna una secuencia de bytes a cada palabra del texto original. Para la codificación se emplean 7 bits de cada byte, y el octavo bit es una marca para indicar si el byte es el primero de un código o no. La asignación de códigos se realiza aplicando el algoritmo de Huffman. El árbol de Huffman tiene grado 128.

Las dos últimas codificaciones resultan mucho más ventajosas que los métodos de compresión de textos clásicos cuando se aplican sobre textos en lenguaje natural. Así, por un lado, alcanzan mejores ratios de compresión (entre un 30 % y un 35 %, frente al 40 % de herramientas como *Compress* y *Gzip* que utilizan métodos de compresión basados en caracteres) tal como se podrá observar en el Capítulo 4 dedicado a los estudios experimentales realizados. Por otro lado, resultan más eficientes a la hora de realizar consultas ya que permiten la búsqueda de palabras directamente sobre

el texto comprimido. Resultados empíricos [37] han demostrado que la búsqueda sobre el texto comprimido es incluso más rápida que la realizada sobre el texto original.

Se han revisado los principales tipos de búsquedas que se pueden aplicar sobre textos comprimidos bajo las codificaciones Plain y Tagged Huffman. Se han cubierto desde los tipos más simples (consultas por palabra clave) hasta las nuevas capacidades de búsqueda que están surgiendo, como es la búsqueda con patrones utilizando expresiones regulares o los potentes patrones extendidos. Como conclusión del estudio puede extraerse que la codificación Tagged Huffman permite realizar búsquedas más eficientes que la codificación Plain Huffman gracias a la existencia del bit de marca mediante el cual se pueden reconocer los códigos directamente dentro del texto comprimido. El problema de esta codificación es que el porcentaje de compresión se ve sensiblemente afectado debido a la "pérdida" de dicho bit para la codificación.

El esquema de compresión propuesto en este trabajo posee las ventajas de la codificación Tagged Huffman en el aspecto de la recuperación pero es más rápido y consigue mejores ratios de compresión, como se demostrará en los siguientes capítulos.

3

Esquema de Codificación Densa con Post-Etiquetado

3.1. Introducción

En los últimos años se han desarrollado nuevas técnicas de compresión especialmente indicadas para su aplicación sobre textos en lenguaje natural. Estas técnicas surgen con la premisa de permitir la realización de búsquedas directamente en el texto comprimido, evitando así la necesidad de descomprimir antes de buscar. Todas ellas se basan en la utilización de las palabras como los símbolos a comprimir [26] (en lugar de los caracteres, como sucede en los métodos clásicos). Esta idea encaja perfectamente con los requerimientos de los algoritmos de Recuperación de Textos, dado que las palabras son generalmente los átomos de dichos sistemas.

Como hemos visto en el capítulo anterior, en [37, 45] Moura *et al.* presentan una técnica de compresión basada en palabras que utiliza Huffman para la codificación. El modelo utilizado es semiadaptativo; esto es, en el proceso de codificación se realiza una primera pasada para obtener las frecuencias de las v palabras distintas del texto original y, en una segunda pasada, se lleva a cabo la compresión del texto. Cada palabra del texto original se reemplaza por un código que es una secuencia de símbolos de un

vocabulario de salida.

La técnica de Moura *et al.* sigue la idea principal del método Huffman clásico [19]; es decir, codifica el texto original de forma que se asignen códigos más cortos a aquellos símbolos más frecuentes. Los códigos generados son de *prefijo libre*, lo que significa que ningún código es prefijo de otro. Así, si 01 es el código asignado a un símbolo de entrada A, ningún otro código podrá comenzar por 01.

La ventaja de las codificaciones de prefijo libre es, como ya se mencionó en el capítulo anterior, que cualquier código puede ser descomprimido sin referenciar a los posteriores debido a que el fin de un código siempre es reconocible. De este modo, siguiendo el ejemplo, si la cadena a descomprimir comienza por 01 tenemos la certeza de que corresponde al símbolo A, sin necesidad de tener que comprobar si podría tratarse del comienzo de otro código de mayor longitud.

La codificación Huffman clásica emplea dos símbolos para la codificación: 0 y 1; es decir, el código asignado a cada símbolo del texto original está formado por uno o más bits. Por el contrario, el esquema de compresión propuesto por Moura *et al.* asigna a cada palabra uno o más bytes. Por ello, esta técnica de codificación se denomina *Codificación Huffman orientada a byte basada en palabras.*

La técnica de Moura *et al.* propone dos tipos de codificación. En la primera, denominada *Plain Huffman*, se emplean para la codificación los 8 bits que componen el byte consiguiendo mejores ratios de compresión que la segunda, denominada *Tagged Huffman* (que sólo utiliza 7 bits), aunque a costa de tener una eficiencia más baja a la hora de realizar búsquedas. La codificación Tagged Huffman se caracteriza por la utilización de marcas que permiten distinguir claramente el comienzo de cada código dentro del texto comprimido. En concreto, el primer byte de cada código tiene un bit de marca con valor 1, mientras que los bytes restantes del código tienen su bit de marca a 0. El uso de marcas permite la realización de búsquedas sobre sobre porciones del texto comprimido sin necesidad de comenzar por el principio del texto. El problema es que el porcentaje de compresión se ve claramente afectado, produciéndose un incremento de más del 11 % en el tamaño del fichero comprimido.

En este capítulo se presenta una técnica de compresión orientada a byte que emplea igualmente un modelo semiadaptativo basado en palabras y una nueva codificación que hemos denominado *codificación Densa con Post-Etiquetado*. Este esquema de codificación conserva las características de las codificaciones anteriormente citadas:

- Es una codificación de prefijo libre.
- Soporta la descompresión eficiente de porciones arbitrarias de texto gracias a la incorporación de un bit de marca en cada uno de los bytes que componen los códigos comprimidos.
- Es posible realizar consultas básicas (búsquedas por palabra clave, por frase o por proximidad) o más complejas (permitiendo errores o utilizando expresiones regulares) directamente sobre el texto comprimido aplicando algoritmos de búsqueda basados en emparejamiento de patrones como los presentados en la Sección 2.6.3.

Al mismo tiempo, la codificación Densa con Post-Etiquetado aporta importantes ventajas adicionales frente a los esquemas basados en Huffman:

- El código generado es más compacto que el obtenido mediante la codificación Tagged Huffman. El aprovechamiento de los códigos es mayor, logrando alcanzar mejores ratios de compresión mientras se continúa manteniendo la propiedad (y, por tanto, las ventajas) del prefijo libre.
- Aunque también es un método semiadaptativo, la generación y asignación de códigos es mucho más sencilla que la realizada por los esquemas de codificación Huffman anteriormente citados. Esto da lugar a una importante reducción en el coste computacional asociado al proceso de codificación y, consecuentemente, a los procesos de compresión y descompresión.

Al igual que sucede con las codificaciones Plain y Tagged Huffman, el nuevo esquema de compresión presentado aquí alcanza ratios de compresión "interesantes" cuando los textos originales tienen un tamaño de al menos 10

MB, debido a la necesidad de tener que almacenar el vocabulario con el texto comprimido. Esto se verá con más detalle en el siguiente capítulo, donde se presentan los resultados experimentales tras la compresión de un conjunto de *corpus* de textos en inglés.

En este capítulo, se explica cómo se lleva a cabo la compresión y búsqueda mediante la nueva codificación Densa con Post-Etiquetado. Así, tras introducir el proceso de codificación, en la Sección 3.3 se presentan los algoritmos asociados a la compresión y descompresión de textos. Por último, en la Sección 3.4 se comentan los tipos de búsquedas que se pueden aplicar directamente sobre textos comprimidos con esta nueva técnica.

3.2. Codificación Densa con Post-Etiquetado

La codificación Densa con Post-Etiquetado es una codificación basada en palabras donde cada código está formado por una secuencia de bytes. Como ya se ha mencionado, de forma similar a la codificación Tagged Huffman, se utiliza un bit de cada byte (en concreto, el bit más significativo) como marca para poder reconocer los códigos dentro del texto comprimido. La única diferencia estriba en que, en lugar de utilizarlo como marca de comienzo de palabra, ahora el bit indica si el byte es el último de un código o no. En concreto, el bit de marca es 1 para el último byte de cada código y 0 para los restantes.

En la Tabla 3.1 se puede observar la diferencia entre los formatos de los códigos correspondientes a la codificación Tagged Huffman frente a la nueva codificación Densa con Post-Etiquetado para diferentes longitudes de códigos.

Se trata de un cambio muy simple pero, sin embargo, tiene importantes consecuencias. En la Tabla 3.1 anterior se puede observar claramente como, por construcción, el hecho de colocar el bit de marca al final del código incorpora automáticamente la característica de prefijo libre a la codificación, independientemente del resto de los bits de cada byte en el código. Dado que únicamente el bit de marca del último byte de cada código vale 1, ningún código podrá ser prefijo de otro. Es decir, dados dos códigos X e Y, si la

N°Bytes	Codif.Tagged Huffman	Codif.Densa con Post-Etiquetado
1	1xxxxxxx	1xxxxxxx
2	1xxxxxxx 0xxxxxxx	0xxxxxxx 1xxxxxxx
3 bytes	1xxxxxxx 0xxxxxxx 0xxxxxxx	0xxxxxxx 0xxxxxxx 1xxxxxxx
...	...	...
n	1xxxxxxx 0xxxxxxx ... 0xxxxxxx	0xxxxxxx ... 0xxxxxxx 1xxxxxxx

Tabla 3.1: Formato de los códigos generados mediante codificación Tagged Huffman y Densa con Post-Etiquetado

longitud (el número de bytes) de X ($long_X$) es menor a la de Y, X nunca podrá ser prefijo de Y porque el último byte de X tiene su bit de marca a 1, mientras que el $long_X$-ésimo byte de Y tiene su bit de marca a 0.

Ejemplo 3.2.1 Los códigos de longitud 2 son de la forma 0xxxxxxx 1xxxxxxx, por lo que nunca podrán ser prefijos de otro código de longitud 3 o superior, porque el segundo byte de estos códigos siempre será de la forma 0xxxxxxx (ver Tabla 3.1). □

Dado que el bit de marca garantiza la propiedad de prefijo libre, ya no es necesario aplicar Huffman sobre los 7 bits restantes (como sucedía para Tagged Huffman) para garantizar una codificación de prefijo libre. Por lo tanto, para la codificación se pueden utilizar todas las combinaciones de valores posibles para cada byte. En concreto, teniendo en cuenta que el bit más significativo es el empleado como bit de marca, los valores mayores de 128 (el caso en el que el primer bit vale 1) se utilizan cuando el byte es el último de un código, y los valores menores de 128 (aquellos cuyo primer bit del byte vale 0) si el byte ocupa otra posición diferente dentro del código.

Veamos ahora cómo se realiza la asignación de códigos a las palabras que componen el texto en función de sus frecuencias de aparición en el mismo.

3.2.1. Proceso de codificación

Una vez eliminada la necesidad de utilizar codificación Huffman, el problema reside en encontrar una asignación de códigos óptima. La idea es

realizar una codificación que minimice en lo posible la longitud de la cadena de salida. Se sigue conservando, por tanto, la idea de asignar códigos más cortos a las palabras más frecuentes, aspecto propio de todas las técnicas de compresión basadas en símbolos como la que nos ocupa.

El proceso de codificación es, como ya se ha mencionado, más sencillo y rápido que el utilizado para aplicar el método de Huffman. No es preciso construir un árbol. La asignación de códigos es secuencial aprovechando todos los valores posibles de cada byte y reservando el bit de marca. Así, el proceso se lleva a cabo del siguiente modo:

1. Se ordenan las palabras distintas (el vocabulario) contenidas en el documento a comprimir. La ordenación se realiza de forma decreciente atendiendo a la frecuencia de aparición de las palabras en el texto.

2. Se asignan secuencialmente los valores 128 a 255 (10000000 a 11111111) a las $2^7 = 128$ palabras más frecuentes. El valor inicial es 128 debido a que, como ya se ha mencionado, el primer bit debe mantenerse a 1 para indicar que se trata del último byte (e inicio en este caso) de código.

3. Los códigos correspondientes a las palabras 129 y siguientes están formados por:

   a) uno o más bytes con valores entre 0 y 127 (es decir, su primer bit es 0)

   b) un byte final con valor 128 o superior (es decir, su primer bit es 1)

   La asignación de códigos se realiza de un modo secuencial utilizando 7 bits de cada byte. Así, la palabra 129 se codifica como 00000000:10000000, la 130 como 00000000:10000001, la 131 como 00000000:10000010, y así sucesivamente como si se tratasen de números de 14 bits.

La Tabla 3.2 muestra los códigos correspondientes a las palabras del texto según su posición en el vocabulario (una vez ordenadas decrecientemente

Posición Palabra	Cod.Densa con Post-Etiquetado	N°Bytes	N°Pal
1	10000000	1	
2	10000001	1	
3	10000010	1	
...	...	...	
$2^7 = 128$	11111111	1	2^7
$2^7 + 1 = 129$	00000000:10000000	2	
130	00000000:10000001	2	
131	00000000:10000010	2	
...	...	...	
256	00000000:11111111	2	
257	00000001:10000000	2	
258	00000001:10000001	2	
259	00000001:10000010	2	
...	...	...	
$2^72^7 + 2^7 = 16512$	01111111:11111111	2	2^72^7
$2^72^7 + 2^7 + 1 = 16513$	00000000:00000000:10000000	3	
16514	00000000:00000000:10000001	3	
16515	00000000:00000000:10000010	3	
...	...	...	
$(2^7)^3 + (2^7)^2 + 2^7$	01111111:01111111:11111111	3	$(2^7)^3$
...	...	...	

Tabla 3.2: Asignación de códigos mediante codificación Densa con Post-Etiquetado

por frecuencia). La tercera columna indica la longitud del código asignado (número de bytes) y la última indica el número total de palabras codificadas para cada tamaño del código (es decir, número de palabras codificadas con 1 byte, número de palabras codificadas con 2 bytes, etc.)

Como se puede observar en la Tabla 3.2, la codificación es extremadamente simple: basta con ordenar las palabras del vocabulario atendiendo a su frecuencia de aparición y asignar secuencialmente los códigos. En consecuencia, la fase de codificación es mucho más rápida que la correspondiente a cualquiera de las técnicas basadas en Huffman (los procesos de codificación y decodificación se explican con más detalle en el siguiente apartado).

Además, es importante destacar el hecho de que *el código de cada*

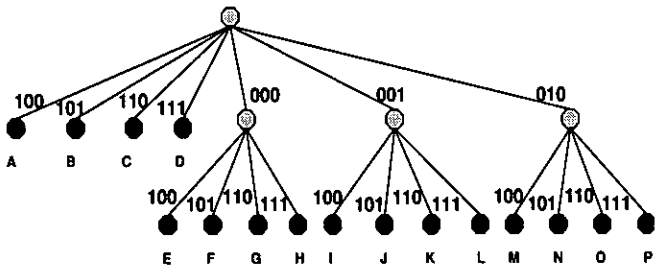


Figura 3.1: Árbol asociado a la codificación Densa con Post-Etiquetado de la Tabla 3.3

palabra depende de su posición en el vocabulario (ordenado decrecientemente por frecuencia), no de su frecuencia real. Es decir, dadas cuatro palabras A, B, C, D con frecuencias $0,20$, $0,15$, $0,14$ y $0,10$, respectivamente, la codificación será la misma que si sus frecuencias fuesen $0,9$, $0,09$, $0,009$ y $0,001$. El resultado, por tanto, es independiente del sesgo de la distribución de las palabras que componen el documento de entrada (a diferencia de lo que sucede en otras codificaciones como Plain o Tagged Huffman). Obsérvese que, tanto si se trata de una distribución uniforme como exponencial (los dos casos más extremos) los códigos generados son exactamente los mismos.

Ejemplo 3.2.2 En la Tabla 3.3 se muestra la codificación Densa con Post-Etiquetado para una distribución de 16 palabras (utilizando bytes "especiales" formados únicamente por 3 bits).

Aunque para la codificación Densa no es necesario construir un árbol ya que la asignación de códigos es secuencial, para mayor claridad en la Figura 3.1 se muestra el árbol de codificación asociado. □

Como se puede observar en la Figura 3.1, la codificación Densa con Post-Etiquetado siempre da lugar a un árbol lo más balanceado posible,

Palabra	Orden	Código
A	1	100
B	2	101
C	3	110
D	4	111
E	5	000 100
F	6	000 101
G	7	000 110
H	8	000 111
I	9	001 100
J	10	001 101
K	11	001 110
L	12	001 111
M	13	010 100
N	14	010 101
O	15	010 110
P	16	010 111

Tabla 3.3: Ejemplo de codificación Densa con Post-Etiquetado para un vocabulario de 16 palabras

independientemente de las frecuencias del vocabulario. Esto es debido a que, dado que la asignación de códigos es secuencial, durante la codificación nunca aumentará la longitud de los códigos (es decir, el número de niveles del árbol) mientras no se hayan utilizado las $2^8 = 256$ ramas posibles de todos los nodos del nivel actual, de las cuales 128 corresponderán a bytes "finales" (con valores entre 128 y 255) y 128 a bytes "intermedios" (con valores entre 0 y 127).

3.3. Procesos de compresión y descompresión bajo codificación Densa con Post-Etiquetado

La codificación Densa con Post-Etiquetado realiza, como hemos visto, una asignación secuencial de códigos utilizando 7 bits de cada byte y reservando el octavo como bit de marca. La sencillez de la codificación hace que no sea necesario almacenar físicamente en el fichero comprimido los códigos asignados a cada palabra. Es posible obtener el código

correspondiente a cada palabra realizando unas simples operaciones una vez se dispone del vocabulario de entrada ordenado decrecientemente por frecuencia. No es necesaria ninguna información extra como sucede para las codificaciones Huffman (recordemos que, en este caso, aún empleando árboles canónicos hay que almacenar un array con información sobre el número de códigos y el valor numérico del primer código para cada longitud (ver Sección 2.3.2)).

A continuación vamos a presentar los algoritmos para llevar a cabo la generación de los códigos a partir del vocabulario y la decodificación (es decir, la obtención de la palabra correspondiente a un código determinado). Posteriormente veremos cómo la eficiencia de estos algoritmos mejora el tiempo necesario para la compresión y descompresión de un texto empleando nuestra codificación.

3.3.1. Algoritmos de codificación y decodificación

Algoritmo de codificación

Tal como se ha visto en la sección anterior, el código correspondiente a una palabra del texto depende de la posición que ésta ocupe en el vocabulario, una vez que las palabras han sido ordenadas en forma decreciente.

Para realizar la codificación puede seguirse un proceso similar al utilizado cuando se llevan a cabo cambios de valores de una base de numeración mayor a otra menor. La idea consiste en realizar divisiones sucesivas del número a codificar (en nuestro caso, la posición i que ocupa la palabra en el vocabulario) entre un cociente que depende del número de símbolos de salida. Así, si se tratase de una conversión de decimal a binario el cociente sería 2, mientras que para nuestra codificación es $2^7 = 128$ (dado que el octavo bit debe reservarse como marca). El resto obtenido tras cada división corresponde al valor de un símbolo (byte) del código final. El algoritmo genera los símbolos que forman el código comenzando por el menos significativo hasta el más significativo. Al byte menos significativo (el último del código) se le coloca el bit de marca a 1 sumándole 128 al valor obtenido.

A continuación se muestra el pseudocódigo del algoritmo que permite la codificación de una palabra que ocupe la posición i en el vocabulario correspondiente a un texto, y un ejemplo de su utilización.

Codificar (i)
\\ cálculo del valor del último byte del código (cuyo bit de marca es 1)
$i \leftarrow i - 1$
output $(i \bmod 128) + 128$ \\ se añade la marca
$i \leftarrow i \text{ div } 128$
\\ cálculo del valor de los bytes restantes del código (con bit de marca a 0)
while $i > 0$ do
    $i \leftarrow i - 1$
    output $(i \bmod 128)$
    $i \leftarrow i \text{ div } 128$
FinCodificar

Ejemplo 3.3.1 Basándonos en la codificación especificada en la Tabla 3.3, veamos cómo generar el código correspondiente a la palabra que ocupa la posición $i = 10$ (es decir, la palabra *J*).

En este ejemplo, dado que los bytes están formados únicamente por 3 bits, el número de símbolos a considerar es $2^{3-1} = 2^2 = 4$ frente al valor $2^7 = 128$ habitual.

Codificar (10)
$i \leftarrow 10 - 1 = 9$
output $(9 \bmod 4) + 4 = 5 \Rightarrow$ byte 101
$i \leftarrow 9 \text{ div } 4 = 2$
$i > 0 \Rightarrow i \leftarrow 2 - 1 = 1$
    output $(1 \bmod 4) = 1 \Rightarrow$ byte 001
    $i \leftarrow 1 \text{ div } 4 = 0$
$i \leq 0 \Rightarrow$ FinCodificar

□

En base al algoritmo anterior, el tiempo necesario para generar el código correspondiente a una palabra que ocupe la posición i en el vocabulario es del orden $O(\log i)$, pues en cada iteracción del bucle i disminuye a $i/128$. La complejidad de este algoritmo también puede expresarse como orden $O(\ell)$, donde ℓ es la longitud (el número de bytes) del código, pues en cada iteracción se obtiene 1 byte del código de salida.

Algoritmo de decodificación

Con respecto a la decodificación, para conocer la palabra correspondiente a un código se puede ejecutar un sencillo algoritmo que devuelve la posición i que ocupa dicha palabra dentro del vocabulario de entrada. El algoritmo es similar al utilizado para realizar conversiones de valores desde una base numérica a otra mayor, y que se presenta en el siguiente ejemplo:

Ejemplo 3.3.2 El valor decimal p correspondiente al número binario 0100101 se puede calcular de la forma: $p = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 + 0 \times 2^6 = 37$ □

En nuestro caso, se analiza igualmente el código byte a byte, comenzando por el que ocupa la posición menos significativa, multiplicando el valor de cada byte por una potencia de 128 que depende de la posición que dicho byte ocupa en el código.

A continuación se presenta el pseudocódigo del algoritmo de decodificación:

Decodificar ()
$r \leftarrow 0$ \\ posic. 1ª palabra para cada long. de código
$p \leftarrow 0$ \\ valor entero del código
$\ell \leftarrow 0$ \\ nº bytes analizados
input x \\ lee el último byte k del código
do
$p \leftarrow p + (x \bmod 128) \times 128^{\ell}$
$r \leftarrow r + 128^{\ell}$

$\ell \leftarrow \ell + 1$
input x \\ lee el byte $k-1$
while $x < 128$
\\ finaliza cuando encuentra el último byte del código anterior
return la posición $i = r + p$
FinDecodificar

Para mayor claridad, veamos cómo funciona el algoritmo con un ejemplo:

Ejemplo 3.3.3 Siguiendo con la codificación de la Tabla 3.3, supongamos que se desea decodificar el código "010 101". El proceso que se lleva a cabo es el siguiente:

Decodificar (010101)
$r \leftarrow 0, p \leftarrow 0, \ell \leftarrow 0$
$x \leftarrow 5$ \\ valor entero del byte menos significativo
$p \leftarrow 0 + (5 \bmod 4) \times 4^0 = 1$
$r \leftarrow 0 + 4^0 = 1$
$\ell \leftarrow 0 + 1 = 1$
$x \leftarrow 2$ \\ valor entero del byte más significativo
$p \leftarrow 1 + (2 \bmod 4) \times 4^1 = 9$
$r \leftarrow 1 + 4^1 = 5$
$\ell \leftarrow 1 + 1 = 2$
return la posición $i = 5 + 9 = 14$
FinDecodificar

□

Este algoritmo permite realizar la decodificación de un código c en un tiempo del orden $O(\log c) = O(\ell)$ ya que, igual que sucedía para la codificación, en cada iteracción del bucle se analiza un byte y el valor de c se divide entre 128.

3.3.2. Tiempos de compresión y descompresión

Una vez descrito el proceso de codificación y decodificación podemos estimar el tiempo necesario para realizar la compresión y descompresión de un texto mediante el esquema de codificación Densa con Post-Etiquetado.

Para comprimir el texto se debe realizar una primera pasada en la cual se extraen las palabras que lo componen y se obtienen sus frecuencias de aparición. Si se emplea una estructura de datos *trie* [7] o un árbol, las palabras se pueden almacenar en una tabla hash en un tiempo en promedio del orden $O(n)$, siendo n el número total de palabras del texto. La ordenación de las diferentes palabras en base a sus frecuencias se puede realizar, utilizando un algoritmo de ordenación rápida, en un tiempo en promedio del orden $O(v \log v)$, siendo v el número de palabras que componen el vocabulario.

Después de la ordenación de las palabras, el siguiente paso es la obtención de sus códigos. Utilizando el algoritmo de codificación presentado en el apartado anterior, mediante el cual es posible obtener el código correspondiente a una palabra del vocabulario, la generación de los códigos del vocabulario puede llevarse a cabo en un tiempo del orden $O(v \log v)$. Finalmente, la compresión del texto (es decir, la búsqueda del código correspondiente a cada palabra del texto original en la tabla hash y su almacenamiento en el fichero comprimido) se lleva a cabo en un tiempo del orden $O(n)$.

Con respecto a la descompresión, ésta comienza leyendo el vocabulario (que se encontrará almacenado en el fichero comprimido junto con el texto codificado) y llevándolo a memoria con un coste del orden $O(v)$. Una vez hecho esto, el tiempo necesario para realizar la decodificación del texto comprimido (utilizando el algoritmo de decodificación del apartado anterior) y la escritura de las palabras correspondientes en disco tiene un coste del orden $O(n)$.

La principal desventaja de la codificación Densa con Post-Etiquetado (al igual que para las codificaciones Huffman) es el requerimiento de espacio necesario para realizar la compresión y descompresión del texto. Aunque no es necesario reservar espacio extra para construir el árbol Huffman,

el vocabulario debe encontrarse en memoria principal para poder realizar tanto la compresión como la descompresión. Por lo tanto, la complejidad de espacio para nuestro método es del orden $O(v)$, lo cual puede resultar un inconveniente para ciertas aplicaciones.

Sin embargo, a pesar de la demanda de memoria de la nueva codificación consideramos que las ventajas que aporta (búsquedas eficientes sobre texto comprimido y rápida descompresión de fragmentos) compensan en gran medida este problema.

3.4. Búsqueda sobre texto comprimido bajo codificación Densa con Post-Etiquetado

Cuando se comprime un texto mediante la codificación Densa con Post-Etiquetado es posible realizar búsquedas directamente sobre la versión comprimida del mismo. Las búsquedas se basan en aplicar *técnicas de emparejamiento comprimido* similares a las empleadas sobre textos comprimidos bajo codificación Tagged Huffman, y ya comentadas en la Sección 2.6.3.

Toda búsqueda debe comenzar con una primera fase en la que se realiza la compresión del patrón (palabra a localizar). Para ello, basta con conocer la posición que ocupa el patrón en el vocabulario, y aplicar el mismo algoritmo de codificación utilizado para comprimir el texto (y cuyo pseudocódigo se ha presentado en la sección anterior). Una vez hecho esto, se puede localizar el código correspondiente dentro del texto comprimido aplicando cualquier algoritmo convencional de emparejamiento de patrones (como los presentados en la Sección 2.6.1) aunque con alguna particularidad como veremos a continuación.

Veamos un ejemplo de utilización de un algoritmo de emparejamiento de patrones, en concreto el *algoritmo de búsqueda de Sunday*, sobre un texto comprimido bajo codificación Densa con Post-Etiquetado:

Ejemplo 3.4.1 Consideremos el vocabulario de entrada de la Tabla 3.3. Supongamos que se desea localizar la primera ocurrencia de la palabra P

sobre el texto comprimido "000 010 111 000 100 010 111 111".

Lo primero que se debe hacer es comprimir la palabra, es decir, conocer su código Denso con Post-Etiquetado. Así, tras aplicar el algoritmo de codificación, obtenemos que su código es 010 111.

Los pasos seguidos por el algoritmo de Sunday son los siguientes:

	$t[0]$	$t[1]$	$t[2]$	$t[3]$	$t[4]$	$t[5]$	$t[6]$	$t[7]$
TEXTO:	000	010	111	000	100	010	111	111
paso 1.	010	111						
paso 2.		010	111					
paso 3.				010	111			
paso 4.						010	111	

- paso 1: El patrón m se alinea al principio del texto. El algoritmo realiza un análisis comenzando por el último byte del patrón ($m[k-1]$, siendo k la longitud del patrón) hacia la izquierda. Por lo tanto, se compara el byte $m[1]$ ="111" con el byte $t[1]$ ="010" del texto. Dado que hay un fallo de emparejamiento se calcula el valor de δ_1 (distancia a la siguiente ocurrencia del símbolo $t[k]$ ="111" en m partiendo del final hacia la izquierda), cuyo valor es 1. Además, se calcula el valor de δ_2 (posición de la siguiente ocurrencia del sufijo del patrón emparejado, comenzando en la posición que produjo el error hacia la izquierda, de modo que su byte a la izquierda no sea el que produjo el error ($m[0]$)). En este caso, δ_2 vale 0, ya que al producirse el error en $m[k-1]$ no hay ningún sufijo emparejado. Finalmente, el desplazamiento será máx$[\delta_1, \delta 2] = 1$

- paso 2: Se comparan los bytes $m[1]$ ="111" y $m[0]$ ="010" con $t[2]$ y $t[1]$, respectivamente. Se produce emparejamiento en ambos casos pero, sin embargo, no se puede confirmar todavía que se trate de una ocurrencia del patrón. Es necesario comprobar si la secuencia $t[1]t[2]$ es realmente código Denso o sólo es parte de otro código de mayor longitud (como en este caso). Debe tenerse en cuenta que, como el patrón se analiza de derecha a izquierda, el primer emparejamiento se lleva a cabo siempre entre bytes "finales" (con bit de marca a 1), con

lo que sabemos que el emparejamiento comienza siempre por un final de código en el texto. Sin embargo, no tenemos la certeza de dónde comienza el código en el texto hasta que hemos localizado el final del código anterior. Por ello, siguiendo el ejemplo, debemos comprobar si el byte $t[0]$ es un byte final. En este caso no es así (ya que vale "000"), lo que indica que la subcadena emparejada con el patrón forma parte de un código diferente (de mayor longitud) al que buscamos. Dado que hay un fallo de emparejamiento, se calcula δ_1 de nuevo, que en este caso vale 2 (no hay ocurrencias de $t[3]$ ="000" en el patrón), y δ_2 que vale 0. Por lo tanto, se hacen 2 desplazamientos.

- paso 3: El emparejamiento entre $m[1]$ ="111" y $t[4]$ ="100" falla. Se calcula δ_1 que vale 2 (la siguiente ocurrencia de $t[5]$ ="010" en m se encuentra a una distancia de 2) y de δ_2 que de nuevo es 0. Por lo tanto, se realizan 2 desplazamientos.

- paso 4: Por último, se comparan los bytes $m[1]$ ="111" y $m[0]$ ="010" con $t[6]$ y $t[5]$, respectivamente. Se produce emparejamiento en ambos casos, por lo que se debe comprobar si la cadena $t[5]t[6]$ forma un código o no. En este caso se confirma ya que el byte anterior $t[4]$ ="100" tiene su bit de marca a 1.

□

En el ejemplo anterior puede observarse como, gracias a la existencia del bit de marca que señala el final de cada código, es posible reconocer directamente los códigos dentro del texto comprimido. De hecho, la búsqueda (o la descompresión) puede comenzar en cualquier lugar del texto comprimido. Además, las búsquedas son muy eficientes ya que nunca existirán falsos emparejamientos. Basta con comprobar si el byte anterior está marcado o no, como se ha hecho en el ejemplo.

Además de búsquedas por palabra clave, como la presentada, es posible realizar otro tipo de consultas básicas (por frase o por proximidad, combinadas mediante expresiones booleanas) o consultas complejas formadas por palabras simples utilizando comodines y expresiones regulares, como sucede para la codificación Tagged Huffman (ver Sección 2.6.3).

3.5. Conclusiones

En este capítulo se ha presentado la principal aportación de este trabajo: un nuevo esquema de compresión para bases de datos textuales. La codificación, denominada Densa con Post-Etiquetado, se basa en otros trabajos previos en el campo de la compresión de textos en lenguaje natural, principalmente en la técnica Huffman basada en palabras orientada a byte propuesta por Moura *et al.* [37], y explicada en el capítulo anterior.

Sin embargo, la codificación Densa con Post-Etiquetado tiene importantes ventajas frente a las codificaciones Huffman basadas en palabras:

- El algoritmo de codificación es muy sencillo. Basta con ordenar las palabras del vocabulario por su frecuencia y asignar secuencialmente los códigos.
- La representación del vocabulario es más simple. No es necesario incluir los códigos en el vocabulario que se incorpora al documento comprimido ni ninguna información extra. Simplemente se deben almacenar las palabras ordenadas por frecuencia.
- Al disponer del vocabulario ordenado, la regeneración del código correspondiente a cada palabra se puede realizar rápidamente en el momento de las búsquedas o para descomprimir el texto.

A diferencia de las codificaciones Huffman, la codificación Densa con Post-Etiquetado no hace un uso inteligente de las distribuciones de frecuencia de las palabras del texto original. El algoritmo de Huffman balancea el árbol para la generación de códigos en base a la distribución asociada al vocabulario, mientras que la nueva codificación únicamente tiene en cuenta el orden de las palabras en base a sus frecuencias y no se ve afectada por las diferencias existentes entre las mismas. Sin embargo, a pesar de ello consigue un mayor rendimiento, tal como se demuestra en el siguiente capítulo. Esto se debe a que al utilizar la totalidad de los valores posibles de cada byte se reduce el tamaño medio de los códigos (si se representan los códigos en un árbol, sería equivalente a decir que se reduce su profundidad media) y, por tanto, la longitud media del texto comprimido.

Un sistema de Recuperación de Textos basado en la nueva técnica de codificación Densa con Post-Etiquetado se puede beneficiar además de otras ventajas, como son: (*i*) la integración al sistema de Recuperación de Textos se simplifica debido a que sólo es necesario almacenar el vocabulario, sin ninguna información adicional; (*ii*) no es necesario construir el árbol Huffman; la codificación y la decodificación se llevan a cabo rápidamente mediante un par de programas muy simples.

Una vez presentada la codificación Densa con Post-Etiquetado y analizada la eficiencia de los algoritmos asociados, damos paso al siguiente capítulo donde se muestran los estudios teóricos y empíricos realizados que confirman las ventajas de la compresión utilizando la nueva codificación frente a las codificaciones Huffman.

3. ESQUEMA DE CODIFICACIÓN DENSA CON POST-ETIQUETADO

4

Análisis del esquema de codificación Densa con Post-Etiquetado

Una vez presentada la nueva técnica de codificación Densa con Post-Etiquetado vamos a analizar su rendimiento mediante un conjunto de estudios analíticos y empíricos. Dado que nuestra idea es mantener la base de datos textual permanentemente comprimida, para mejorar así la eficiencia de almacenamiento, es fundamental que los costes asociados a la compresión/descompresión, así como el porcentaje de compresión, se mantengan en unos niveles razonables.

Para una mejor evaluación de nuestro esquema de codificación, los estudios se han extendido a diferentes técnicas de compresión basadas en palabras. En concreto, se han aplicado también a las codificaciones Plain y Tagged Huffman por ser las más próximas al nuevo método.

El estudio se centra principalmente en la estimación de la *longitud media de los códigos* para cada una de las codificaciones, empleando diferentes distribuciones de probabilidad. Recordemos que el porcentaje de compresión está directamente relacionado con el concepto de *redundancia* (ver Sección 2.2.2), y que la redundancia es una medida de la diferencia entre la longitud

media del código y la entropía. Cuanto menor sea la redundancia mejor será el porcentaje de compresión y, dado que la entropía es constante para una distribución determinada, la redundancia depende únicamente del valor de dicha longitud media.

La Sección 4.1 se dedica al estudio teórico realizado, mientras que en la Sección 4.2 se presentan las pruebas experimentales que se han llevado a cabo para contrastar los resultados teóricos anteriormente obtenidos. Los estudios se han realizado sobre dos distribuciones de palabras hipotéticas (uniforme y exponencial) y una distribución de palabras que teóricamente se ajusta a los textos en lenguaje natural. Para este estudio teórico basado en la distribución del lenguaje natural se ha empleado un modelo aproximado, la *Ley de Zipf*[7, 31, 42], que intenta representar la distribución de las frecuencias de las palabras en el texto y es ampliamente aceptado en el campo de la Recuperación de Textos. A nivel experimental se han utilizado un conjunto de *corpus* de textos en lenguaje natural de la colección TREC-4 [16].

4.1. Resultados Analíticos

En primer lugar vamos a estudiar la longitud media teórica de los códigos (es decir, número medio de símbolos de salida por código) para las codificaciones Plain Huffman, Tagged Huffman y Densa con Post-Etiquetado.

Para el estudio se considerará una distribución de la forma $\{p_i\}_{i=1...v}$, donde p_i es la probabilidad de la i-ésima palabra más frecuente y v es el tamaño del vocabulario. Además, asumimos que se emplean símbolos de b bits para la codificación, de modo que cada código es una secuencia de símbolos de b-bits. Dado que, en cada byte, el número de bits utilizados para la codificación Densa con Post-Etiquetado es $b-1$ (debido al bit reservado como marca), denominaremos $B = 2^{b-1}$ al número de combinaciones que es posible generar con $(b-1)$ bits. En la práctica se emplean bytes (es decir, $b = 8$).

Como ya se presentó en la Sección 2.3.2, el algoritmo de Huffman siempre

produce una codificación óptima de redundancia mínima, pero no alcanza la entropía debido a la imposibilidad de representar partes fraccionarias de símbolos de salida. Es decir, si un símbolo de entrada tiene probabilidad p_i se debe representar con exactamente $log_{2^b}\ (1/p_i)\ =\ -\ log_{2^b}\ p_i$ símbolos de salida (es decir, su entropía), lo cual no es posible si p_i no es potencia de $1/2^b$. Así, si la entropía de un símbolo de entrada es 3.7, el algoritmo de Huffman generará un código formado por 4 símbolos de salida.

Generalizando, si la entropía correspondiente a un símbolo de entrada i con probabilidad p_i es $(-log_{2^b}p_i)$, Huffman utilizará $\lceil\ -log_{2^b}p_i\ \rceil$ símbolos. Esto indica que, independientemente de la distribución de probabilidad, nunca se utilizará más de un símbolo extra por código debido a los ajustes por redondeo mencionados.

Por lo tanto si la entropía de un texto (en base b) es:

$$H_b\ =\ \sum_{i=1}^{v} -p_i \log_{2^b} p_i\ =\ \frac{1}{b}\sum_{i=1}^{v} -p_i \log_2 p_i$$

entonces el número medio de símbolos para codificar una palabra utilizando codificación Plain Huffman $\overline{long_{P_b}}$ es:

$$H_b\ \leq\ \overline{long_{P_b}}\ \leq\ H_b + 1$$

Por otro lado, la codificación Tagged Huffman es una codificación Huffman sobre $b-1$ bits, aunque utilizando b bits por símbolo (ya que se reserva un bit como marca). Entonces, en este caso, la longitud media de los símbolos $\overline{long_{T_b}}$ es:

$$H_{b-1}\ \leq\ \overline{long_{T_b}}\ \leq\ H_{b-1} + 1$$

Consideremos ahora la nueva codificación Densa con Post-Etiquetado, donde el número medio de símbolos por palabra es $\overline{long_{D_b}}$. En un análisis preliminar se tiene que $\overline{long_{P_b}}\ \leq\ \overline{long_{D_b}}$ ya que, aunque se trata de una codificación de prefijo libre, Huffman es la mejor codificación de prefijo libre (como ya se comentó en el capítulo anterior). Por otro lado, $\overline{long_{D_b}}\ \leq\ \overline{long_{T_b}}$ porque la codificación Densa emplea todas las combinaciones de los $(b-1)$

bits restantes (recordemos que se hace una asignación de códigos secuencial utilizando 7 bits) mientras que la codificación Tagged Huffman no. Por lo tanto,

$$\overline{long_{P_b}} \leq \overline{long_{D_b}} \leq \overline{long_{T_b}}$$

Dado que estas desigualdades presentan unas cotas muy amplias, vamos a precisar más las diferencias entre las tres codificaciones estudiando cómo varían sus longitudes medias cuando se aplican sobre diferentes distribuciones de palabras. En concreto, el estudio se centra en las distribuciones exponencial y uniforme (ya utilizadas en la Sección 2.5.1 para ilustrar las codificaciones Plain y Tagged Huffman) y en la distribución teórica para textos en lenguaje natural determinada por la *Ley de Zipf*.

4.1.1. CASO 1.- Distribución Exponencial

En el caso de las codificaciones Plain Huffman y Tagged Huffman, si el vocabulario de entrada sigue una distribución exponencial (es decir, la probabilidad de la i-ésima palabra del vocabulario es $p_i = 2^{-i}$) el árbol Huffman obtenido es totalmente degenerado (ver Figura 2.6). En concreto, en cada nivel del árbol habrá $(2^b - 1) = (2B - 1)$ nodos hoja en caso de la codificación Plain Huffman, o $(2^{b-1} - 1) = (B - 1)$ para la codificación Tagged Huffman, y un único nodo no terminal.

En la Tabla 4.1 se muestran los códigos generados por las codificaciones Plain y Tagged Huffman en caso de una distribución exponencial. En la primera columna se indican las posiciones de las palabras correspondientes en el vocabulario de entrada (ordenadas decrecientemente por frecuencia), y en la tercera se muestra el número de palabras codificadas para cada longitud de código (número de palabras codificadas con 1 byte, con 2 bytes, etc.).

La longitud media de los códigos se puede calcular como la suma de las porciones de la longitud media total que originan las palabras codificadas con 1 byte, 2 bytes, etc. Así, dado que la expresión para calcular la longitud media de un código es $\sum_{i=1}^{v} p_i * long_i$ (ver Sección 2.2.2), la porción de longitud media originada por las palabras codificadas con 1 byte para la

Posición	Cód. Plain Huffman	N°Pal.	$\overline{long}$
1	11111111		
2	11111110		
...	...		
$(2B-1)$	00000001	$2B-1$	$\overline{long_{P_b1}} = 1\sum_{i=1}^{(2B-1)} p_i$
$(2B-1)+1$	00000000:11111111		
$(2B-1)+2$	00000000:11111110		
...	...		
$2(2B-1)$	00000000:00000001	$2B-1$	$\overline{long_{P_b2}} = 2\sum_{i=(2B-1)+1}^{2(2B-1)} p_i$
$2(2B-1)+1$	00000000:00000000:11111111		
$2(2B-1)+2$	00000000:00000000:11111110		
...	...		
$3(2B-1)$	00000000:00000000:00000001	$2B-1$	$\overline{long_{P_b3}} = 3\sum_{i=2(2B-1)+1}^{3(2B-1)} p_i$
...	...		

Posición	Cód. Tagged Huffman	N°Pal.	$\overline{long}$
1	11111111		
2	11111110		
...	...		
$(B-1)$	10000001	$B-1$	$\overline{long_{T_b1}} = 1\sum_{i=1}^{(B-1)} p_i$
$(B-1)+1$	10000000:01111111		
$(B-1)+2$	10000000:01111110		
...	...		
$2(B-1)$	10000000:00000001	$B-1$	$\overline{long_{T_b2}} = 2\sum_{i=(B-1)+1}^{2(B-1)} p_i$
$2(B-1)+1$	10000000:00000000:01111111		
$2(B-1)+2$	10000000:00000000:01111110		
...	...		
$3(B-1)$	10000000:00000000:00000001	$B-1$	$\overline{long_{T_b3}} = 3\sum_{i=2(B-1)+1}^{3(B-1)} p_i$
...	...		

Tabla 4.1: Asignación de códigos mediante codificaciones Plain y Tagged Huffman para una distribución exponencial

codificación Plain Huffman se puede calcular como:

$$\overline{long_{P_b1}} = \sum_{i=1}^{(2B-1)} p_i * long_i = \sum_{i=1}^{(2B-1)} p_i * 1 = 1 \sum_{i=1}^{(2B-1)} p_i$$

Siguiendo el mismo razonamiento se pueden calcular las porciones de longitud media originadas por las palabras codificadas con 2 bytes, 3 bytes, etc., como se muestran en la Tabla 4.1.

Generalizando y sumando se puede obtener la expresión de la longitud media de los códigos para un texto cuyas palabras siguen una distribución de probabilidades exponencial y que se comprime mediante la codificación Plain Huffman ($\overline{long_{P_b}}$) como:

$$\begin{aligned}\overline{long_{P_b}} &= \overline{long_{P_b1}} + \overline{long_{P_b2}} + \overline{long_{P_b3}} + \ldots + \overline{long_{P_bS}} = \\ &= 1 \sum_{i=1}^{(2B-1)} p_i + 2 \sum_{i=(2B-1)+1}^{2(2B-1)} p_i + 3 \sum_{i=2(2B-1)+1}^{3(2B-1)} p_i + \ldots = \\ &= \sum_{i=1}^{S} i \sum_{j=(i-1)(2B-1)+1}^{i(2B-1)} p_j\end{aligned}$$

donde S es la longitud (el número de bytes) del código más largo.

Dado que cada grupo de códigos de igual longitud codifica el mismo número de palabras (en concreto $2B - 1 = 255$), es posible calcular el valor de S como:

$$S = \left\lceil \frac{v}{(2B-1)} \right\rceil$$

Para la codificación Tagged Huffman, la longitud media $\overline{long_{T_b}}$ es similar:

$$\overline{long_{T_b}} = \sum_{i=1}^{S} i \sum_{j=(i-1)(B-1)+1}^{i(B-1)} p_j$$

y el valor de S se calcula como:

$$S = \left\lceil \frac{v}{(B-1)} \right\rceil$$

Posición	Cod. Densa con Post-Etiquet	N°Pal	$\overline{long}$
1	10000000		
2	10000001		
...	...		
B	11111111	B	$\overline{long_{D_b1}} = 1 \sum_{i=1}^{B} p_i$
$B+1$	00000000:10000000		
$B+2$	00000000:10000001		
...	...		
$B+B^2$	01111111:11111111	B^2	$\overline{long_{D_b2}} = 2 \sum_{i=B+1}^{B+B^2} p_i$
$B+B^2+1$	00000000:00000000:10000000		
$B+B^2+2$	00000000:00000000:10000001		
...	...		
$B+B^2+B^3$	01111111:01111111:11111111	B^3	$\overline{long_{D_b3}} = 3 \sum_{i=B+B^2+1}^{B+B^2+B^3} p_i$
...	...		

Tabla 4.2: Asignación de códigos mediante codificación Densa con Post-Etiquetado para una distribución exponencial

Veamos ahora qué sucede para nuestra codificación. La codificación Densa con Post-Etiquetado es independiente de las diferencias entre las frecuencias de las distintas palabras, tal como se ha mencionado en el capítulo anterior. Realiza una asignación secuencial de códigos utilizando todas las combinaciones posibles de 7 bits, y reservando el octavo como bit de marca.

De este modo, tal como puede observarse en la Tabla 4.2, bajo codificación Densa con Post-Etiquetado el número de palabras que se codifican con códigos de 1 byte es $B = 2^7$. A partir de dos bytes el número de palabras codificadas crece exponencialmente. Esto es debido a que, en el árbol que representa la codificación, cada uno de los B nodos no terminales del primer nivel se expanden en B nodos terminales y otros B nodos no terminales. Consecuentemente, el número de palabras codificadas con 2 bytes es de $B^2 = 2^7 2^7$, las palabras codificadas con 3 bytes serán $B^3 = 2^7 2^7 2^7$, y así sucesivamente.

Por lo tanto, la porción de longitud media originada por las palabras codificadas con 1 byte ($\overline{long_{D_b1}}$) es

$$\overline{long_{D_b1}} = \sum_{i=1}^{B} p_i * long_i = \sum_{i=1}^{B} p_i * 1 = 1 \sum_{i=1}^{B} p_i$$

Igualmente, puede calcularse la longitud media originada por los códigos de 2 bytes, 3 bytes, etc. (ver Tabla 4.2).

Generalizando, la longitud media de los códigos correspondientes a la codificación Densa con Post-Etiquetado puede definirse como:

$$\begin{aligned}\overline{long_{D_b}} &= \overline{long_{D_b1}} + \overline{long_{D_b2}} + \overline{long_{D_b3}} + \ldots + \overline{long_{D_bS}} = \\ &= 1\sum_{i=1}^{B} p_i + 2\sum_{i=B+1}^{B+B^2} p_i + 3\sum_{i=B+B^2+1}^{B+B^2+B^3} p_i + \ldots + S\sum_{i=B+\ldots+B^{S-1}+1}^{B+\ldots+B^S} p_i\end{aligned}$$

Si denominamos $s_i = \sum_{j=1}^{i} B^j$ (donde $s_0 = 0$), podemos expresar el límite superior de los sumatorios como s_i y el límite inferior como $s_{i-1} + 1$. Por lo tanto,

$$\begin{aligned}\overline{long_{D_b}} &= 1\sum_{i=s_0+1}^{s_1} p_i + 2\sum_{i=s_1+1}^{s_2} p_i + 3\sum_{i=s_2+1}^{s_3} p_i + \ldots + S\sum_{i=s_{S-1}+1}^{s_S} p_i = \\ &= \sum_{i=1}^{S} i \sum_{j=s_{i-1}+1}^{s_i} p_j\end{aligned}$$

donde, recordemos, S representa la longitud del código más largo.

Podemos calcular S en función del tamaño del vocabulario de entrada v. Teniendo en cuenta que se codifican B palabras con códigos de 1 byte, B^2 palabras con 2 bytes, etc., entonces:

$$v = \sum_{j=1}^{S} B_j$$

La expresión $\sum_{j=1}^{S} B_j$ representa una progresión geométrica de razón B y término B, por lo que v puede definirse de la forma:

$$v = \sum_{j=1}^{S} B^j = \frac{B}{B-1}(B^S - 1)$$

Entonces:

$$v = \frac{B}{B-1}(B^S - 1)$$

$$\begin{aligned} B^S &= \frac{B-1}{B}v+1 \\ \log_B B^S &= \log_B\left(\frac{B-1}{B}v+1\right) \\ S &= \left\lceil \log_B\left(\frac{B-1}{B}v+1\right)\right\rceil \end{aligned}$$

En base a las longitudes medias presentadas para las tres codificaciones, se puede afirmar que, si el tamaño del vocabulario es pequeño, la codificación Densa con Post-Etiquetado da lugar a una compresión claramente mejor. Sin embargo, dado que se trata de una distribución exponencial (y, por lo tanto, muy sesgada), a medida que crece el tamaño del vocabulario la eficiencia de la compresión para las tres codificaciones se asemeja más. Debe tenerse en cuenta que, en este caso, las frecuencias de las palabras poco probables (precisamente las que se ven más desfavorecidas en las codificaciones Plain y Tagged Huffman) afectan cada vez menos al resultado. De este modo, cuando el tamaño del vocabulario es muy grande la longitud media de los códigos tiende a 1 para las tres codificaciones y, por lo tanto, podemos concluir que las tres son igualmente buenas.

4.1.2. CASO 2.- Distribución Uniforme

Supongamos ahora el caso en que el vocabulario sigue una distribución uniforme. Es decir, $p_i = p_j \ \forall i, j$.

En caso de las codificaciones Huffman, si todas las palabras del vocabulario de entrada tienen la misma probabilidad se codifican con códigos de igual longitud. En concreto, si utilizamos d símbolos para la codificación y el número de palabras diferentes v es potencia de d, todas las palabras del vocabulario se codifican con $\log_d v$ símbolos.

Ejemplo 4.1.1 Como se puede observar en el ejemplo de codificación Tagged Huffman mostrado en la Tabla 2.4 y Figura 2.5(B), dado que el número de símbolos para la codificación es $2^{3-1} = 4$ y el número de palabras a codificar es $v = 16$, todas las palabras del vocabulario se codifican con $\log_4 16 = 2$ bytes. □

De este modo, si el número de palabras diferentes en el texto v es potencia de B, la longitud media de los códigos generados por la codificación Tagged Huffman para una distribución uniforme es:

$$\overline{long_{T_b}} = \log_B v$$

Dado que todos los códigos tienen la misma longitud, $\log_B v$ es también la longitud máxima (es decir, S).

En caso de la codificación Plain Huffman, si el número de palabras diferentes en el texto v es potencia de $2B$, todas las palabras se codifican con $\log_{2B} v$. Por lo tanto, la longitud media (y máxima) es:

$$\overline{long_{P_b}} = \log_{2B} v$$

Respecto a la codificación Densa con Post-Etiquetado, la longitud media de los códigos (y máxima) es la misma que la obtenida para la distribución exponencial. Recordemos que la asignación de códigos se realiza secuencialmente atendiendo al orden de las palabras en el vocabulario y no a las diferencias entre sus frecuencias.

Si el vocabulario de entrada es muy grande, puede decirse que para una distribución uniforme las tres codificaciones se comportan de un modo similar. En los tres casos, la longitud media de los códigos se aproxima a $\log v$.

4.1.3. CASO 3.- Distribución (teórica) de textos en lenguaje natural: Ley de Zipf

Hemos presentado las dos distribuciones más extremas que podrían seguir las palabras de un vocabulario de entrada: la distribución exponencial ($p_i = 2^{-i}$) y la distribución uniforme ($p_i = p_j \ \forall i, j$). Sin embargo, ninguna de ellas se ajusta a las distribuciones reales de las palabras en un texto en lenguaje natural.

Existe un modelo aproximado, ampliamente aceptado en el campo de la Recuperación de Textos, para representar la distribución de las frecuencias de

las palabras, la *Ley de Zipf* [7, 31, 42]. Esta ley establece que si ordenamos las v palabras del vocabulario de un texto en orden decreciente de frecuencia, la probabilidad de la palabra más frecuente es i^θ veces la de la i-ésima palabra, para todo i. Esto significa que la probabilidad de la i-ésima palabra es $p_i = A/i^\theta$, donde $A = \frac{1}{\sum_{i\geq 1} 1/i^\theta} = \frac{1}{\zeta(\theta)}$, y el valor de θ depende del texto.

La Ley de Zipf tiene dos aproximaciones. En el primer caso se considera $\theta = 1$. Esta versión es muy sencilla pero también inexacta y no sigue bien la distribución real de los textos en lenguaje natural. La mayoría de los textos reales tienen un vocabulario más sesgado, de modo que los valores de θ mayores que 1 (más concretamente entre 1.4 y 1.8) se ajustan mejor [4, 7]. Esta segunda aproximación, donde $\theta > 1$, se denomina *Ley de Zipf generalizada.*

Una vez presentado el modelo a utilizar para representar las distribuciones de textos en lenguaje natural, vamos a calcular las longitudes medias originadas por cada una de las codificaciones. Previamente podemos analizar el valor de la entropía como:

$$
\begin{aligned}
H_b &= \frac{1}{b}\sum_{i}^{v} p_i \log_2(1/p_i) = \frac{1}{b}\sum_{i\geq 1} p_i \log_2(1/p_i) = \\
&= \frac{1}{b}\sum_{i\geq 1}\frac{A}{i^\theta}\log_2(i^\theta/A) = \frac{1}{b}\sum_{i\geq 1}\frac{A}{i^\theta}(\theta\log_2 i - \log_2 A) = \\
&= \frac{1}{b}\left(A\theta\sum_{i\geq 1}\frac{\log_2 i}{i^\theta} - A\log_2 A\sum_{i\geq 1}\frac{1}{i^\theta}\right) = \\
&= \frac{1}{b}\left(A\theta\sum_{i\geq 1}\frac{\log_2 i}{i^\theta} - \log_2 A\right) = \frac{1}{b}\left(A\theta\sum_{i\geq 1}\frac{\ln i}{\ln 2}\frac{1}{i^\theta} - \frac{\ln A}{\ln 2}\right) = \\
&= \frac{1}{b}\left(\frac{\theta}{\zeta(\theta)\ln 2}\sum_{i\geq 1}\frac{\ln i}{i^\theta} - \frac{\ln(1/\zeta(\theta))}{\ln 2}\right) = \frac{-\theta\zeta'(\theta)/\zeta(\theta) + \ln\zeta(\theta)}{b\ln 2}
\end{aligned}
$$

donde $\zeta'(x) = \partial\zeta(x)/\partial x$.

Considerando que, tal como se ha visto al comienzo de esta sección, el número medio de símbolos que utiliza la codificación Plain Huffman para

comprimir un texto oscila entre la entropía (H_b) y la entropía más un símbolo (H_b+1), podemos estimar la longitud media correspondiente a dicha codificación para la distribución de Zipf como:

$$\begin{array}{rcccl} H_b & \leq & \overline{long_{P_b}} & \leq & H_b+1 \\ \dfrac{-\theta\zeta'(\theta)/\zeta(\theta)+\ln\zeta(\theta)}{b\ln 2} & \leq & \overline{long_{P_b}} & \leq & \dfrac{-\theta\zeta'(\theta)/\zeta(\theta)+\ln\zeta(\theta)}{b\ln 2}+1 \end{array}$$

Respecto a la codificación Tagged Huffman, el número medio de símbolos se encuentra entre H_{b-1} y $H_{b-1}+1$. Por lo tanto,

$$\begin{array}{rcccl} H_{b-1} & \leq & \overline{long_{T_b}} & \leq & H_{b-1}+1 \\ \dfrac{-\theta\zeta'(\theta)/\zeta(\theta)+\ln\zeta(\theta)}{(b-1)\ln 2} & \leq & \overline{long_{P_b}} & \leq & \dfrac{-\theta\zeta'(\theta)/\zeta(\theta)+\ln\zeta(\theta)}{(b-1)\ln 2}+1 \end{array}$$

Veamos ahora cuál es el valor de la longitud media para la codificación Densa con Post-Etiquetado. En este caso,

$$\begin{aligned} \overline{long_{D_b}} &= \sum_{i=1}^{S} i \sum_{j=s_{i-1}+1}^{s_i} p_j = \sum_{i=1}^{S} i \sum_{j=s_{i-1}+1}^{s_i} \frac{A}{j^\theta} = \\ &= A\sum_{i\geq 1}((i-1)+1) \sum_{j=s_{i-1}+1}^{s_i} \frac{1}{j^\theta} = \\ &= A\sum_{i\geq 1}\sum_{j=s_{i-1}+1}^{s_i} \frac{1}{j^\theta} + A\sum_{i\geq 1}(i-1)\sum_{j=s_{i-1}+1}^{s_i} \frac{1}{j^\theta} = \\ &= A\sum_{j\geq 1}\frac{1}{j^\theta} + A\sum_{i\geq 1} i\sum_{j=s_i+1}^{s_{i+1}} \frac{1}{j^\theta} = \\ &= \frac{1}{\sum_{i\geq 1} 1/i^\theta}\sum_{j\geq 1}\frac{1}{j^\theta} + A\sum_{i\geq 1} i\sum_{j=s_i+1}^{s_{i+1}} \frac{1}{j^\theta} = \\ &= 1 + A\sum_{i\geq 1} i\sum_{j=s_i+1}^{s_{i+1}} \frac{1}{j^\theta} \end{aligned}$$

Es posible simplificar la expresión $\sum_{i\geq 1} i\sum_{j=s_i+1}^{s_{i+1}} 1/j^\theta$ teniendo en cuenta que si:

$$\begin{aligned}
&\text{i=1}\Rightarrow 1\textstyle\sum_{j=s_1+1}^{s_2}\frac{1}{j^\theta}= && \textstyle\sum_{j=s_1+1}^{s_2}\frac{1}{j^\theta}\\
&\text{i=2}\Rightarrow 2\textstyle\sum_{j=s_2+1}^{s_3}\frac{1}{j^\theta}= && \textstyle\sum_{j=s_2+1}^{s_3}\frac{1}{j^\theta}+\sum_{j=s_2+1}^{s_3}\frac{1}{j^\theta}\\
&\text{i=3}\Rightarrow 3\textstyle\sum_{j=s_3+1}^{s_4}\frac{1}{j^\theta}= && \textstyle\sum_{j=s_3+1}^{s_4}\frac{1}{j^\theta}+\sum_{j=s_3+1}^{s_4}\frac{1}{j^\theta}+\sum_{j=s_3+1}^{s_4}\frac{1}{j^\theta}\\
&&& \vdots\\
&\text{i=k}\Rightarrow k\textstyle\sum_{j=s_k+1}^{s_{k+1}}\frac{1}{j^\theta}= && \textstyle\sum_{j=s_k+1}^{s_{k+1}}\frac{1}{j^\theta}+\sum_{j=s_k+1}^{s_{k+1}}\frac{1}{j^\theta}+\ldots\ldots+\sum_{j=s_k+1}^{s_{k+1}}\frac{1}{j^\theta}\\
&&& \vdots\\
&\text{como i}\simeq\infty\Rightarrow && \textstyle\sum_{j=s_1+1}^{\infty}\frac{1}{j^\theta}+\sum_{j=s_2+1}^{\infty}\frac{1}{j^\theta}+\ldots\ldots+\infty\ =\\
&&& =\quad \textstyle\sum_{i\geq 1}\sum_{j\geq s_i+1}\frac{1}{j^\theta}
\end{aligned}$$

Por lo tanto, la longitud media de la codificación Densa con Post-Etiquetado para una distribución que se ajuste a la Ley de Zipf es:

$$\overline{long_{D_b}} = A\sum_{i\geq 1} i \sum_{j=s_{i-1}+1}^{s_i}\frac{1}{j^\theta} = 1+A\sum_{i\geq 1} i \sum_{j\geq s_i+1}^{s_{i+1}}\frac{1}{j^\theta} = 1+A\sum_{i\geq 1}\sum_{j\geq s_i+1}\frac{1}{j^\theta}$$

Dado que $f(x) = 1/j^\theta$ es una función no negativa y decreciente (ver Figura 4.1), podemos establecer una cota superior y una cota inferior de la expresión anterior aplicando el *criterio integral de Cauchy* tal como se explica en los siguientes apartados.

Cota superior

Puede obtenerse una cota superior de la longitud media de la codificación Densa con Post-Etiquetado para la distribución de Zipf aplicando el *criterio integral de Cauchy*:

$$\sum_{k=2}^{v} f(k) \leq \int_1^v f(x)\,dx$$

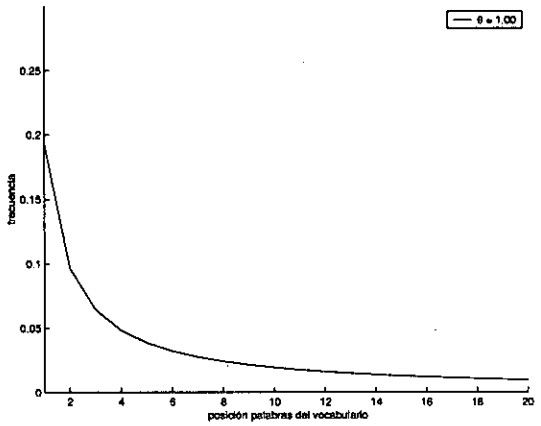


Figura 4.1: Ley de Zipf para un valor de θ =1.0

De este modo,

$$\overline{long_{D_b}} = 1 + A \sum_{i \geq 1} \sum_{j \geq s_i + 1} 1/j^{\theta} \leq 1 + A \sum_{i \geq 1} \int_{s_i}^{\infty} 1/x^{\theta} dx$$

Resolviendo la integral

$$\int_{s_i}^{\infty} 1/x^{\theta} dx = \int_{s_i}^{\infty} \frac{1}{1-\theta} x^{1-\theta} = \frac{1}{1-\theta} [\infty^{1-\theta} - s_i^{1-\theta}] =$$
$$= -\frac{1}{1-\theta} s_i^{1-\theta} = \frac{1}{\theta - 1} s_i^{1-\theta}$$

Por lo tanto,

$$\overline{long_{D_b}} \leq 1 + A \sum_{i \geq 1} \frac{1}{\theta - 1} s_i^{1-\theta} =$$
$$= 1 + A \sum_{i \geq 1} \frac{1}{\theta - 1} \left[\frac{B}{B-1} (B^i - 1) \right]^{1-\theta} =$$
$$= 1 + A \sum_{i \geq 1} \frac{1}{\theta - 1} \frac{B^{1-\theta}}{(B-1)^{1-\theta}} (B^i - 1)^{1-\theta} =$$
$$= 1 + A \frac{1}{\theta - 1} \frac{B^{1-\theta}}{(B-1)^{1-\theta}} \sum_{i \geq 1} \frac{1}{(B^i - 1)^{1-\theta}} =$$
$$= 1 + \frac{A(B-1)^{\theta-1}}{(\theta - 1) B^{\theta-1}} \sum_{i \geq 1} \frac{1}{(B^i - 1)^{\theta-1}}$$

El sumatorio puede simplificarse considerando que

$$\begin{aligned}\sum_{i\geq 1}\frac{1}{(B^i-1)^{\theta-1}} &= \sum_{i\geq 1}\frac{1}{[B^i(1-\frac{1}{B^i})]^{\theta-1}} = \sum_{i\geq 1}\frac{1}{B^{i(\theta-1)}}\left(1-\frac{1}{B^i}\right)^{1-\theta} = \\ &= \sum_{i\geq 1}\frac{1}{B^{i(\theta-1)}}e^{\ln(1-\frac{1}{B^i})^{1-\theta}} \leq \sum_{i\geq 1}\frac{1}{B^{i(\theta-1)}}e^{(1-\theta)\ln(1-\frac{1}{B})} = \\ &= \sum_{i\geq 1}\frac{1}{B^{i(\theta-1)}}\left(1-\frac{1}{B}\right)^{1-\theta} = \frac{B^{1-\theta}}{1-B^{1-\theta}}\left(1-\frac{1}{B}\right)^{1-\theta}\end{aligned}$$

De este modo la cota superior queda de la forma:

$$\begin{aligned}\overline{long_{D_b}} &\leq 1+\frac{A\,(B-1)^{\theta-1}}{(\theta-1)\,B^{\theta-1}}\frac{B^{1-\theta}}{1\,-\,B^{1-\theta}}\left(1\,-\,\frac{1}{B}\right)^{1-\theta} = \\ &= 1+\frac{1}{(\theta-1)\zeta(\theta)}\,\frac{(B-1)^{\theta-1}}{B^{\theta-1}}\,\frac{1}{B^{\theta-1}\,[1\,-\,B^{1-\theta}]}\left(1\,-\,\frac{1}{B}\right)^{1-\theta} = \\ &= 1+\frac{1}{(\theta-1)\zeta(\theta)}\,\frac{(B-1)^{\theta-1}\,(\frac{B-1}{B})^{1-\theta}}{B^{\theta-1}\,B^{\theta-1}\,[1\,-\,B^{1-\theta}]} = \\ &= 1+\frac{1}{(\theta-1)\zeta(\theta)}\,\frac{1}{B^{\theta-1}\,[1\,-\,B^{1-\theta}]} = \\ &= 1+\frac{1}{(\theta-1)\zeta(\theta)(B^{\theta-1}-1)}\end{aligned}$$

Cota inferior

De modo similar puede obtenerse una cota inferior para la codificación Densa con Post-Etiquetado, aplicando el *criterio integral de Cauchy.*

$$\int_2^v f(x)\,dx \;\leq\; \sum_{k=2}^{v} f(k)$$

La cota inferior de la longitud media de los códigos para una distribución de Zipf puede entonces establecerse como

$$\overline{long_{D_b}} \geq 1 \;+\; A\sum_{i\geq 1}\int_{s_i+1}^{\infty} 1/x^{\theta}dx$$

Resolviendo la integral:

$$\int_{s_i+1}^{\infty} 1/x^{\theta} dx = \int_{s_i+1}^{\infty} \frac{1}{1-\theta} x^{1-\theta} = \frac{1}{1-\theta}[\infty^{1-\theta} - (s_i+1)^{1-\theta}] =$$
$$= -\frac{1}{1-\theta}(s_i+1)^{1-\theta} = \frac{1}{\theta-1}(s_i+1)^{1-\theta}$$

Por lo tanto,

$$\overline{long_{D_b}} \geq 1 + A\sum_{i\geq 1} \frac{1}{\theta-1}(s_i+1)^{1-\theta} =$$
$$= 1 + A\sum_{i\geq 1} \frac{1}{\theta-1}\left[\frac{B}{B-1}(B^i-1)+1\right]^{1-\theta} =$$
$$= 1 + \frac{A}{\theta-1}\sum_{i\geq 1}\left[\frac{B^{i+1}-1}{B-1}\right]^{1-\theta} =$$
$$= 1 + \frac{A}{(\theta-1)(B-1)^{1-\theta}}\sum_{i\geq 1}(B^{i+1}-1)^{1-\theta} =$$
$$= 1 + \frac{A\ (B-1)^{\theta-1}}{\theta-1}\sum_{i\geq 1}\frac{1}{(B^{i+1}-1)^{\theta-1}}$$

El sumatorio puede simplificarse considerando que

$$\sum_{i\geq 1}\frac{1}{(B^{i+1}-1)^{\theta-1}} = \sum_{i\geq 1}\frac{1}{\left[B^{i+1}\left(1-\frac{1}{B^{i+1}}\right)\right]^{\theta-1}} =$$
$$= \sum_{i\geq 1}\frac{1}{B^{(i+1)(\theta-1)}}\frac{1}{\left(1-\frac{1}{B^{i+1}}\right)^{\theta-1}}$$

Como $\left[1/\left(1-\frac{1}{B^{i+1}}\right)^{\theta-1}\right]$ es siempre mayor de 1, entonces

$$\sum_{i\geq 1}\frac{1}{B^{(i+1)(\theta-1)}}\frac{1}{\left(1-\frac{1}{B^{i+1}}\right)^{\theta-1}} \geq \sum_{i\geq 1}\frac{1}{B^{(i+1)(\theta-1)}} = \sum_{i\geq 1} B^{(i+1)(\theta-1)} =$$
$$= \frac{B^{2(1-\theta)}(B^{(1-\theta)\infty}-1)}{B^{1-\theta}-1} =$$
$$= \frac{-B^{2(1-\theta)}}{B^{1-\theta}-1} = \frac{B^{2(1-\theta)}}{1-B^{1-\theta}}$$

Por lo tanto, la cota superior queda de la forma:

$$\begin{aligned}
\overline{long_{D_b}} &\geq 1 + \frac{A(B-1)^{\theta-1}B^{2(1-\theta)}}{(\theta-1)(1-B^{1-\theta})} = \\
&= 1 + \frac{1}{(\theta-1)\zeta(\theta)} \frac{(B-1)^{\theta-1}B^{2(1-\theta)}}{1-B^{1-\theta}} = \\
&= 1 + \frac{1}{(\theta-1)\zeta(\theta)} \frac{B^{\theta-1}(1-1/B)^{\theta-1}B^{1-\theta}B^{1-\theta}}{1-B^{1-\theta}} = \\
&= 1 + \frac{1}{(\theta-1)\zeta(\theta)} \frac{(1-1/B)^{\theta-1}}{B^{\theta-1}[1-B^{1-\theta}]} = \\
&= 1 + \frac{(1-1/B)^{\theta-1}}{(\theta-1)\zeta(\theta)(B^{\theta-1}-1)}
\end{aligned}$$

Como consecuencia, puede observarse como las cotas superior e inferior anteriores permiten conocer la longitud media de los códigos con un factor de precisión de $(1-1/B)^{\theta-1}$. Esto supone una precisión de alrededor del $0,5\,\%$, para $B = 128$ y θ entre 1.4 y 1.8 (los valores de θ que, teóricamente, más se ajustan a los textos en lenguaje natural).

Esto demuestra que la nueva codificación es más simple de analizar que la codificación Huffman pues los límites establecidos para las codificaciones Plain y Tagged Huffman:

$$H_b \leq \overline{long_{P_b}} \leq H_b + 1$$

y

$$H_{b-1} \leq \overline{long_{T_b}} \leq H_{b-1} + 1$$

respectivamente, son mucho más imprecisos.

De hecho, el límite inferior H_b para la codificación Plain Huffman resulta poco útil para nuestro caso: Para $b = 8$ bits, $H_b < 1$ para $\theta > 1,3$; sin embargo, el límite inferior obvio es un único símbolo. Lo mismo sucede cuando se utiliza H_{b-1} para limitar la codificación Tagged Huffman. Se han considerado cálculos más ajustados [13, 25] pero ninguno resultó útil en nuestro caso.

Por ello, para poder comparar los resultados obtenidos para la codificación Densa con Post-Etiquetado con los correspondientes a las

codificaciones Plain Huffman y Tagged Huffman, utilizamos un análisis propio. El razonamiento es el siguiente:

- Supongamos que $b = 9$ (con lo $B = 2^{b-1} = 256$). Si aplicamos la codificación Densa con Post-Etiquetado para este caso, la longitud media será necesariamente mejor que la obtenida para la codificación Plain Huffman con 8 bits. Esto es obvio, ya que la codificación Densa utiliza 8 bits de forma óptima, mientras que la codificación Plain Huffman no emplea todas las combinaciones para mantener un código de prefijo libre. Por lo tanto, se cumple que

$$\overline{long_{D_{b+1}}} \leq \overline{long_{P_b}}$$

- Por otro lado, si consideramos $b = 7$ entonces la codificación Densa con Post-Etiquetado es un código de prefijo libre de 7 bits, que utiliza 6 bits para la codificación y 1 bit como marca. Por lo tanto, la longitud media obtenida es necesariamente inferior a la correspondiente a la codificación Tagged Huffman de 8 bits. Es decir,

$$\overline{long_{T_b}} \leq \overline{long_{D_{b-1}}}$$

Añadiendo estas desigualdades a las presentadas al principio de la Sección 4.1 obtenemos:

$$\overline{long_{D_{b+1}}} \leq \overline{long_{P_b}} \leq \overline{long_{D_b}} \leq \overline{long_{T_b}} \leq \overline{long_{D_{b-1}}}$$

Estos resultados aportan límites interesantes para comparar el rendimiento de las diferentes codificaciones. De este modo, este análisis adicional de la codificación Densa con Post-Etiquetado ha generado nuevos límites (superior e inferior) para la redundancia de los códigos Huffman orientados a byte.

Este hecho también se puede observar en la Figura 4.2 que ilustra los cálculos analíticos realizados en función del parámetro θ de la Ley de Zipf. En dicha figura se ve claramente cómo los límites inferior y superior para la codificación Densa con Post-Etiquetado son bastante próximos.

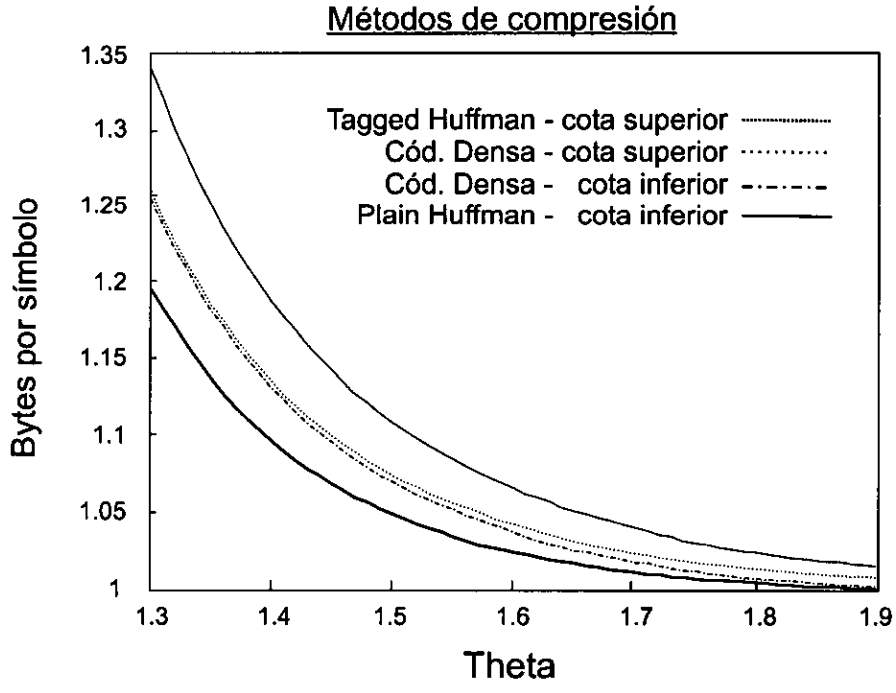


Figura 4.2: Límites teóricos para la longitud media de las codificaciones Plain Huffman, Tagged Huffman y Densa con Post-Etiquetado, considerando una distribución de Zipf cuyo parámetro θ se encuentra en el eje x (utilizando $b = 8$)

4.2. Estudios Experimentales

Además del estudio teórico presentado en la sección anterior, se ha realizado una comparativa de los resultados obtenidos al comprimir textos reales en lenguaje natural mediante la codificación Densa con Post-Etiquetado frente a las codificaciones Huffman orientadas a byte basadas en palabras.

Los *corpus* de textos en inglés que se han utilizado para realizar las pruebas de codificación, compresión y descompresión han sido los siguientes:

- Un conjunto de colecciones extraídas de entre los *corpus* utilizados en la conferencia $TREC-4$ [16]:

  AP AP Newswire 1988

  $ZIFF$ Ziff Data 1989-90: artículos de *Computer Selected*(Ziff-Davis Publishing)

  CR Congressional Record 1993

  $FT91$ Financial Times 1991

  $FT92$ Financial Times 1992

  $FT93$ Financial Times 1993

  $FT94$ Financial Times 1994

- Un subconjunto del *corpus* de Calgary/Canterbury [8]:

  TOT formado por la totalidad de los textos seleccionados del *corpus* Calgary (*Book*1, *Book*2, *Paper*1 al *Paper*6, *Bib* y *News*).

  *Book*1 y *Book*2, *Paper*1 al *Paper*6: escritos en inglés coloquial, combinan historias reales y de ficción.

  Bib: referencias bibliográficas escritas en inglés técnico.

  News: se trata de un conjunto de artículos con noticias en inglés no publicadas.

Sobre estos *corpus* se han calculado las longitudes medias para las codificaciones Plain Huffman, Tagged Huffman y nuestra codificación Densa con Post-Etiquetado, así como los porcentajes de compresión alcanzados.

Paralelamente se ha realizado un estudio para comprobar hasta qué punto la Ley de Zipf se ajusta a los textos en lenguaje natural. En concreto, el análisis se ha centrado en el parámetro θ de la Ley de Zipf por ser el que aporta información respecto a la variabilidad del texto. Así, teóricamente, cuando las frecuencias de las palabras son más similares entre sí, θ es más pequeño, mientras que si la distribución de frecuencias es más sesgada (es decir, hay un número de palabras con frecuencias muy altas mientras que el resto tienen frecuencias similares muy bajas) el valor de θ es más alto. Por otro lado, tanto para las codificaciones Huffman como para la codificación Densa con Post-Etiquetado, cuánto más sesgada sea la distribución de un texto mejor será el porcentaje de compresión, ya que eso significa que los códigos más cortos estarán codificando la mayoría de las palabras. Por lo tanto, un valor de θ alto debería dar lugar a una mejor compresión.

La Figura 4.3 ilustra las probabilidades teóricas de las palabras en un texto para distintos valores de θ. Se puede observar que cuanto más grande es el valor de θ, más sesgada es la distribución de frecuencias del *corpus* (mejor compresión), mientras que valores menores de θ se asocian a textos con distribuciones de frecuencia más uniformes.

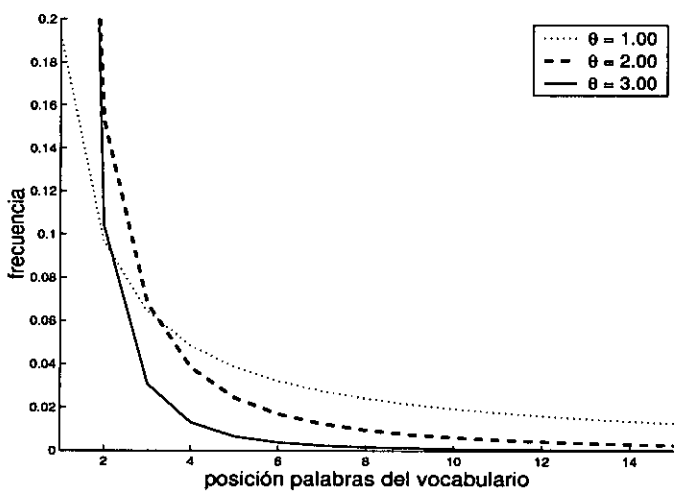


Figura 4.3: Distribución teórica de las frecuencias de las palabras en un texto para valores de θ= 1,0, 2,0 y 3,0

A lo largo del estudio que se presenta a continuación se han contrastado

los resultados teóricos anteriores con datos reales. Para ello, se ha calculado el valor del θ empírico de los diferentes *corpus* y se ha comparado con los porcentajes de compresión alcanzados para las diferentes codificaciones.

Los experimentos se han realizado utilizando dos alternativas diferentes para llevar a cabo el preprocesamiento del texto original y la extracción de las palabras a codificar:

- utilizando dos vocabularios de entrada diferentes [26]: uno con las palabras y otro con los separadores (conjunto de caracteres no alfanuméricos entre dos palabras consecutivas).
- mediante un único vocabulario que incluye tanto palabras como separadores. Además, se utiliza la aproximación *spaceless* [37], donde el espacio simple se considera como el separador por defecto por lo que se omite su almacenamiento.

Los experimentos llevados a cabo para ambas alternativas se presentan a continuación.

4.2.1. CASO 1.- Empleando DOS VOCABULARIOS

En este primer experimento se creó, como se ha mencionado, un vocabulario de entrada con las palabras del texto y otro con los separadores [26]. Dado que las palabras y los separadores siguen una alternancia estricta en el texto, no hay ninguna confusión sobre qué vocabulario utilizar en cada momento una vez se conoce si el texto comienza por una palabra o por un separador.

Ejemplo 4.2.1 Dado un texto de la forma: "*In the beginning there was the text: the text was words ... and there was too much of it, and it had to be compressed.*".

En este caso, el vocabulario de las palabras es {*In, the, beginning, there, was, text, words, and, too, much, of, it, had, to, be, compressed*}, y el vocabulario de separadores queda de la forma {" " (un espacio en blanco), " "(dos espacios en blanco), ": ", " ... ", ", ", "."}. □

En la Tabla 4.3 se muestra el número de palabras y separadores extraídos para cada uno de los *corpus*.

La Tabla 4.4 muestra las longitudes medias para las tres codificaciones. Puede observarse como, para todos los casos, la codificación Densa con Post-Etiquetado da lugar a una longitud media de códigos que se encuentra entre las correspondientes a la codificación Plain Huffman y Tagged Huffman (en la mayoría de los casos, los valores se aproximan más a la codificación Plain Huffman).

En la Tabla 4.5 se presentan los tamaños correspondientes a los textos comprimidos. En este caso no se ha considerado la inclusión del vocabulario en el fichero comprimido. En dicha tabla se puede observar como, para textos grandes, la codificación Densa con Post-Etiquetado consigue ratios de compresión entre un 2 % y un 3 % mejores que la codificación Tagged Huffman, y menos de un 2 % peores de la codificación Plain Huffman. Aunque los resultados cambian algo para las colecciones más pequeñas (correspondientes al *corpus* Calgary), realmente podemos llegar a las mismas conclusiones.

Cálculo del valor de θ empírico

A partir de los vocabularios ordenados por frecuencias, se obtuvieron los valores de θ correspondientes a las palabras y a los separadores para cada uno de los *corpus*. Para ello se ha calculado la función de regresión lineal entre la posición de las palabras en el vocabulario y su frecuencia de aparición en el texto, obteniéndose una expresión de la forma $f(x) = Ax^{-\theta}$ (donde, recordemos, $A = \frac{1}{\sum_{i \geq 1} 1/i^{\theta}}$).

Los valores de θ obtenidos se presentan en la última columna de la Tabla 4.3. Puede observarse cómo las distribuciones de los separadores están más sesgadas que las de las palabras (los valores de θ son más altos en la mayoría de los casos). De hecho están tan sesgadas (θ alcanza valores cercanos a 2,0) que dan lugar a una codificación casi óptima; así, las longitudes medias de los códigos son muy próximas a 1, lo que indica que se emplea un único byte por separador. Por el contrario, las palabras presentan un valor de θ similar (entre 1,6 y 1,8 para los *corpus* grandes) dando lugar a códigos más largos

corpus	Total Palabras Total Separad	Vocabulario Palabras Vocabulario Separad	θ Palabras θ Separad
AP	41 135 103	238 434	1,845506
	41 135 425	2 876	2,308981
Ziff	30 228 425	213 920	1,741795
	30 228 606	7 516	1,737952
CR	7 765 434	112 850	1,623903
	7 765 481	1 320	2,319811
FT91	2 426 418	74 351	1,438503
	2 426 433	1 245	2,003911
FT92	28 882 888	279 330	1,627602
	28 883 064	5 564	1,774848
FT93	31 948 457	285 485	1,643654
	31 948 655	5 830	1,808228
FT94	32 843 959	288 427	1,646028
	32 844 163	6 592	1,772367
TOT	372 957	26 849	1,227113
	372 965	588	1,677979
Book1	141 223	13 165	1,135144
	141 224	588	1,677979
Book2	105 962	7 907	1,271044
	105 963	1 114	1,521758
Paper1	9 157	1 791	1,004143
	9 158	316	1,426831
Paper2	14 266	2 468	1,035512
	14 267	184	1,632045
Paper3	7 364	2 087	0,835870
	7 365	125	1,667132
Paper4	2 219	751	0,809930
	2 218	75	1,582207
Paper5	2 207	613	0,914670
	2 208	190	1,134055
Paper6	7 246	1 163	1,098575
	7 247	373	1,263650
Bib	20 519	3 667	0,922653
	20 520	56	2,735627
News	62 794	9 912	1,025764
	62 795	2 060	1,107732

Tabla 4.3: Número de palabras/separadores y valores de θ para los *corpus* procesados empleando 2 vocabularios

corpus	Longitud media Código		
	Plain Palabras Separadores	Denso Palabras Separadores	Tagged Palabras Separadores
AP	1,580889	1,621690	1,752314
	1,001058	1,002637	1,002953
Ziff	1,552861	1,595390	1,718536
	1,002246	1,004602	1,005507
CR	1,543411	1,586550	1,709030
	1,000506	1,001422	1,001595
FT91	1,539468	1,581893	1,702489
	1,000843	1,001705	1,001870
FT92	1,552060	1,591722	1,721922
	1,001234	1,002280	1,002726
FT93	1,546462	1,588016	1,722232
	1,001589	1,002983	1,003587
FT94	1,546728	1,588283	1,722868
	1,001890	1,003563	1,004290
TOT	1,532575	1,582532	1,707515
	1,002776	1,006599	1,006812
Book1	1,393286	1,452787	1,566480
	1,002776	1,006599	1,006812
Book2	1,410222	1,491638	1,564929
	1,014467	1,027028	1,028444
Paper1	1,300426	1,415420	1,435514
	1,006661	1,028281	1,028936
Paper2	1,326090	1,427309	1,453175
	1	1,003925	1,003995
Paper3	1,381586	1,483568	1,502308
	1	1	1
Paper4	1,248422	1,385933	1,392696
	1	1	1
Paper5	1,166289	1,299048	1,304486
	0,999999	1,028080	1,028533
Paper6	1,213497	1,341844	1,355921
	1,016283	1,044708	1,045361
Bib	1,295287	1,365953	1,393391
	1	1	1
News	1,511116	1,581616	1,701739
	1,045242	1,062107	1,065690

Tabla 4.4: Longitudes medias para las codificaciones Plain Huffman, Densa con Post-Etiquetado y Tagged Huffman empleando 2 vocabularios

corpus	Tamaño en Bytes			
	Descompr	Plain Huffman	Denso con Post-Etiq	Tagged Huffman
AP	250 994 525	106 208 528 (42,32 %)	107 952 284 (43,00 %)	113 338 515 (45,16 %)
Ziff	185 417 980	77 237 041 (41,66 %)	78 593 845 (42,39 %)	82 343 712 (44,41 %)
CR	51 085 545	19 754 666 (38,67 %)	20 089 659 (39,33 %)	21 049 227 (41,20 %)
FT91	14 749 355	6 165 963 (41,80 %)	6 268 903 (42,50 %)	6 561 920 (44,49 %)
FT92	175 449 248	73 746 681 (42,03 %)	74 922 445 (42,70 %)	78 695 879 (44,85 %)
FT93	197 586 334	81 406 496 (41,20 %)	82 778 619 (41,89 %)	87 085 910 (44,07 %)
FT94	203 783 923	83 706 909 (41,08 %)	85 126 689 (41,77 %)	89 570 870 (43,95 %)
TOT	2 131 045	954 992 (44,81 %)	978 119 (45,90 %)	1 027 273 (48,21 %)
Book1	785 389	338 380 (43,14 %)	347 323 (44,22 %)	363 409 (46,27 %)
Book2	610 856	256 926 (42,06 %)	266 884 (43,69 %)	274 800 (44,99 %)
Paper1	54 360	21 127 (38,86 %)	22 378 (41,17 %)	22 568 (41,52 %)
Paper2	82 199	33 185 (40,37 %)	34 685 (42,20 %)	35 055 (42,65 %)
Paper3	46 526	17 539 (37,70 %)	18 290 (39,31 %)	18 428 (39,61 %)
Paper4	13 286	4 988 (37,54 %)	5 293 (39,84 %)	5 308 (39,95 %)
Paper5	11 954	4 782 (40,01 %)	5 136 (42,96 %)	5 150 (43,08 %)
Paper6	38 105	16 158 (42,40 %)	17 294 (45,39 %)	17 401 (45,67 %)
Bib	111 261	47 098 (42,33 %)	48 548 (43,63 %)	49 111 (44,14 %)
News	377 109	161 584 (42,85 %)	166 109 (44,05 %)	173779 (46,08 %)

Tabla 4.5: Ratios de compresión para las codificaciones Plain Huffman, Densa y Tagged Huffman empleando 2 vocabularios, sin incluir el vocabulario en el documento comprimido

(entre 1,5 y 1,7 bytes por palabra).

Obsérvese cómo los valores de θ para los ficheros más pequeños se desvían considerablemente de estos valores y de otros resultados previos correspondientes a colecciones grandes [4]. Con textos pequeños los valores de θ obtenidos (tanto para las palabras como para los separadores) son más bajos que los correspondientes a los *corpus* de mayor tamaño. Esto debería dar lugar a ratios de compresión peores, pero en la práctica no ha sucedido así, lo cual evidencia que este tipo de análisis sólo es aplicable a colecciones grandes.

4.2.2. CASO 2.- Aproximación *SPACELESS*

En un segundo experimento, la compresión se realizó mediante un único vocabulario que incluía tanto palabras como separadores. Además, se utilizó la aproximación *spaceless* ideada por Moura *et al.* [37], donde el espacio simple se considera como el separador por defecto por lo que se omite su almacenamiento. Así, si una palabra es seguida por un espacio se codifica únicamente la palabra. En caso contrario se codifica tanto la palabra como el separador. En el momento de la decodificación se decodifica la palabra y se asume que va seguida de un espacio en blanco excepto si el siguiente símbolo corresponde a un separador (es decir, si el siguiente carácter no es alfanumérico).

Ejemplo 4.2.2 Supongamos el mismo texto utilizado como ejemplo en el apartado anterior: *In the beginning there was the text: the text was words ... and there was too much of it, and it had to be compressed..*

En este caso, habrá un único vocabulario de la forma {*In, the, " "(dos espacios en blanco), beginning, there, was, text, ": ", words, " ... ", and, too, much, of, it, ", ", had, to, be, compressed, "."*}. □

En la Tabla 4.6 se puede observar el número de palabras extraídas para cada *corpus* y el nuevo valor de θ obtenido, mientras que en la Tabla 4.7 se presentan las longitudes medias para las codificaciones Plain Huffman, Densa con Post-Etiquetado y Tagged Huffman.

corpus	Total Palabras	Vocabulario Palabras	θ
AP	52 960 212	241 315	1,852045
Ziff	40 548 114	221 443	1,744346
CR	9 445 990	114 174	1,634076
FT91	3 059 634	75 597	1,449878
FT92	36 518 075	284 904	1,630996
FT93	41 772 135	291 322	1,647456
FT94	43 039 879	295 023	1,649428
TOT	497 607	30 898	1,235893
Book1	177 545	13 755	1,603719
Book2	141 355	9 023	1,299097
Paper1	12 878	2 107	1,066433
Paper2	17 607	2 652	1,071368
Paper3	9 342	2 212	0,893581
Paper4	2 874	825	0,873023
Paper5	3 048	802	0,932310
Paper6	10 352	1 535	1,116883
Bib	29 619	3 722	0,967341
News	92 987	11 973	1,040270

Tabla 4.6: Número de palabras y valores de θ para los *corpus* procesados empleando aproximación *spaceless*

corpus	Longitud media Código		
	Plain Huffman	Denso	Tagged Huffman
AP	1,477827	1,516603	1,638228
Ziff	1,449977	1,490903	1,606416
CR	1,475534	1,520167	1,638070
FT91	1,455481	1,497421	1,612230
FT92	1,465063	1,504248	1,627545
FT93	1,447384	1,489149	1,613115
FT94	1,447626	1,489669	1,613737
TOT	1,485337	1,534128	1,653926
Book1	1,348807	1,403774	1,508001
Book2	1,379697	1,451494	1,527700
Paper1	1,295698	1,405964	1,427551
Paper2	1,299881	1,395411	1,423184
Paper3	1,338043	1,434061	1,452366
Paper4	1,232429	1,354558	1,362909
Paper5	1,211614	1,338583	1,347113
Paper6	1,246233	1,366016	1,382631
Bib	1,213883	1,267902	1,290219
News	1,438995	1,492402	1,614247

Tabla 4.7: Longitudes medias para codificaciones Plain Huffman, Densa y Tagged Huffman empleando *spaceless*

corpus	Tamaño en Bytes			
	Descompr	Plain Huffman	Denso con Post-Etiq	Tagged Huffman
AP	250 994 525	78 266 031 (31,18 %)	80 319 616 (32,00 %)	86 760 902 (34,56 %)
Ziff	185 417 980	58 793 833 (31,70 %)	60 453 305 (32,60 %)	65 137 139 (35,13 %)
CR	51 085 545	13 937 879 (27,28 %)	14 359 482 (28,11 %)	15 473 193 (30,29 %)
FT91	14 749 355	4 453 239 (30,19 %)	4 581 560 (31,06 %)	4 932 834 (33,44 %)
FT92	175 449 248	53 501 281 (30,49 %)	54 932 241 (31,31 %)	59 434 810 (33,88 %)
FT93	197 586 334	60 460 320 (30,60 %)	62 204 933 (31,48 %)	67 383 258 (34,10 %)
FT94	203 783 923	62 305 648 (30,57 %)	64 115 174 (31,46 %)	69 455 045 (34,08 %)
TOT	2 131 045	739 114 (34,68 %)	763 393 (35,82 %)	823 005 (38,62 %)
Book1	768 771	239 474 (31,15 %)	249 233 (32,42 %)	267 738 (34,83 %)
Book2	610 856	195 027 (31,93 %)	205 176 (33,59 %)	215 241 (35,24 %)
Paper1	53 161	16 686 (31,38 %)	18 106 (34,06 %)	18 384 (34,58 %)
Paper2	82 199	22 887 (27,84 %)	24 569 (29,89 %)	25 058 (30,48 %)
Paper3	46 526	12 500 (26,87 %)	13 397 (28,79 %)	13 568 (29,16 %)
Paper4	13 286	3 542 (26,66 %)	3 893 (29,30 %)	3 917 (29,48 %)
Paper5	11 954	3 693 (30,89 %)	4 080 (34,13 %)	4 106 (34,35 %)
Paper6	38 105	12 901 (33,86 %)	14 141 (37,11 %)	14 313 (37,56 %)
Bib	111 261	35 954 (32,32 %)	37 554 (33,75 %)	38 215 (34,35 %)
News	377 109	133 808 (35,48 %)	138 774 (36,80 %)	150 104 (39,80 %)

Tabla 4.8: Ratios de compresión para codificaciones Plain Huffman, Densa y Tagged Huffman empleando *spaceless* sin incluir el vocabulario en el documento comprimido

En la Tabla 4.8 se indica el tamaño del fichero comprimido sin incluir el vocabulario. Como puede observarse, los porcentajes de compresión alcanzados son considerablemente mejores a los obtenidos bajo la aproximación con 2 vocabularios.

En este caso se ha estudiado también cómo afecta en el porcentaje de compresión final la inclusión del vocabulario en el fichero comprimido. Recuérdese que tanto para las codificaciones Plain y Tagged Huffman como para la codificación Densa con Post-Etiquetado es necesario almacenar en el documento final el vocabulario con las palabras ordenadas por frecuencia para poder regenerar los códigos en el momento de la descompresión o búsqueda. El tamaño del vocabulario es prácticamente igual para nuestra representación que para los métodos basados en Huffman, debido a que para representar los árboles Huffman se emplean códigos Huffman canónicos.

Para intentar que la sobrecarga derivada de la necesidad de almacenar el vocabulario en el texto comprimido sea mínima se ha comprimido usando

el método de Huffman binario basado en caracteres. Los resultados de la compresión del vocabulario pueden observarse en la Tabla 4.9. La necesidad de descomprimir el vocabulario en el momento de la búsqueda supone una sobrecarga de tiempo mínima que queda compensada con el ahorro en tiempo de entrada/salida.

En la Tabla 4.10 se muestran los porcentajes de compresión conseguidos finalmente una vez incluido el vocabulario comprimido. Como puede observarse, aunque para grandes documentos la inclusión del vocabulario apenas ha afectado en la compresión final (por ejemplo, el *corpus AP* ha pasado de un 32,0 % a un 32,53 %) cuando los textos son pequeños puede degradar significativamente el porcentaje de compresión final. En todos estos casos el porcentaje de compresión ha aumentado considerablemente (véase, por ejemplo, como *Book*1 ha reducido su compresión de un 32,42 % a un 40,50 %, o el caso más extremo de *News* donde se ha pasado de un 36,80 % a un 52,84 %).

Los resultados de la compresión empeoran, por tanto, de forma clara para las colecciones de texto pequeñas. Esto es un hecho habitual para todas las aproximaciones siempre que se emplean las palabras como los símbolos a comprimir, y es debido a esta sobrecarga producida por la necesidad de almacenar el vocabulario. En el campo de la Recuperación de Textos, sin embargo, es habitual trabajar con grandes colecciones de textos; en este caso, el espacio extra ocupado por el vocabulario llega a ser despreciable.

Comparando los resultados experimentales con los estudios analíticos anteriores, podemos concluir que, cuando se emplea un único vocabulario para palabras y separadores, las predicciones son más optimistas, como puede observarse en la Tabla 4.10.

Con los porcentajes de compresión para los 7 corpus de la colección $TREC-4$ (reflejados en la tabla anterior) se ha realizado un test estadístico para comprobar si hay diferencias significativas entre los porcentajes correspondientes a las codificaciones Densa con Post-Etiquetado y Tagged Huffman, respecto a la codificación Plain Huffman. Los resultados obtenidos permiten confirmar, por un lado, que el incremento de la codificación Tagged Huffman respecto a la codificación Plain Huffman es de aproximadamente un 11 %, tal como se indica en [37]. Por otro lado, podemos concluir que

corpus	Vocabulario Descomp	Vocabulario Comprimido
AP	2 145 325	1 332 511 (62,11 %)
Ziff	1 788 471	1 175 576 (65,73 %)
CR	914 223	594 106 (64,98 %)
FT91	644 496	421 470 (65,40 %)
FT92	2 651 981	1 792 843 (67,60 %)
FT93	2 716 564	1 845 142 (67,92 %)
FT94	2 789 199	1 890 554 (67,78 %)
TOT	243 754	156 443 (64,18 %)
Book1	107 159	62 095 (57,95 %)
Book2	69 356	42 559 (61,36 %)
Paper1	14 725	9 481 (64,39 %)
Paper2	20 011	11 855 (59,24 %)
Paper3	17 310	10 132 (58,53 %)
Paper4	5 710	3 619 (63,38 %)
Paper5	5 080	3 422 (67,36 %)
Paper6	9 796	6 490 (66,25 %)
Bib	27 525	17 109 (62,16 %)
News	90 839	60 483 (66,58 %)

Tabla 4.9: Ratios de compresión del vocabulario utilizando Huffman binario orientado a carácter

corpus	Tamaño en Bytes			
	Descompr	Plain Huffman	Denso con Post-Etiq	Tagged Huffman
AP	250 994 525	79 598 542 (31,71 %)	81 652 127 (32,53 %)	88 093 413 (35,10 %)
Ziff	185 417 980	59 969 409 (32,34 %)	61 628 881 (33,24 %)	66 312 715 (35,76 %)
CR	51 085 545	14 531 985 (28,45 %)	14 953 588 (29,27 %)	16 067 299 (31,45 %)
FT91	14 749 355	4 874 709 (33,05 %)	5 003 030 (33,92 %)	5 354 304 (36,30 %)
FT92	175 449 248	55 294 124 (31,52 %)	56 725 084 (32,33 %)	61 227 653 (34,90 %)
FT93	197 586 334	62 305 462 (31,53 %)	64 050 075 (32,42 %)	69 228 400 (35,04 %)
FT94	203 783 923	64 196 202 (31,50 %)	66 005 728 (32,39 %)	71 345 599 (35,01 %)
TOT	2 131 045	895 557 (42,02 %)	919 836 (43,16 %)	979 448 (45,96 %)
Book1	768 771	301 569 (39,23 %)	311 328 (40,50 %)	329 833 (42,90 %)
Book2	610 856	237 586 (38,89 %)	247 735 (40,56 %)	257 800 (42,20 %)
Paper1	53 161	26 167 (49,22 %)	27 587 (51,89 %)	27 865 (52,42 %)
Paper2	82 199	34 742 (42,27 %)	36 424 (44,31 %)	36 913 (44,91 %)
Paper3	46 526	22 632 (48,64 %)	23 529 (50,57 %)	23 700 (50,94 %)
Paper4	13 286	7 161 (53,90 %)	7 512 (56,54 %)	7 536 (56,72 %)
Paper5	11 954	7 115 (59,52 %)	7 502 (62,76 %)	7 528 (62,97 %)
Paper6	38 105	19 391 (50,89 %)	20 631 (54,14 %)	20 803 (54,59 %)
Bib	111 261	53 063 (47,69 %)	54 663 (49,13 %)	55 324 (49,72 %)
News	377 109	194 291 (51,52 %)	199 257 (52,84 %)	210 587 (55,84 %)

Tabla 4.10: Ratios de compresión para las codificaciones Plain Huffman, Densa y Tagged Huffman empleando *spaceless* e incluyendo el vocabulario comprimido en el documento final

para *corpus* grandes la codificación Densa con Post-Etiquetado mejora los porcentajes de compresión en aproximadamente un 8 % (concretamente, entre un 7,55 % y un 8,31 %) respecto a la codificación Tagged Huffman, y únicamente es superada por la codificación Plain Huffman en un 3 %.

Es decir, tal como se preveía, la codificación Densa con Post-Etiquetado ha alcanzado mejores valores para las longitudes medias de los códigos que la codificación Tagged Huffman, aún a pesar de reservar igualmente un bit de cada byte para poder delimitar los códigos.

4.3. Conclusiones

En este capítulo se han demostrado, analítica y experimentalmente, las ventajas de realizar la compresión de textos utilizando la nueva codificación Densa con Post-Etiquetado frente a las codificaciones Plain y Tagged Huffman.

El estudio se ha centrado principalmente en el cálculo de la redundancia correspondiente a las tres codificaciones, la cual viene determinada por la longitud media de sus códigos. Para la pruebas empíricas se ha utilizado un conocido conjunto de *corpus* en inglés, mientras que para las pruebas teóricas se han utilizado 3 distribuciones hipotéticas de diferente sesgo, entre las que se incluye el modelo de Zipf que representa la distribución de los textos en lenguaje natural. Para este último caso se ha obtenido una cota superior e inferior lo cual ha servido, adicionalmente, para establecer nuevas cotas de las codificaciones Huffman.

Los resultados obtenidos tras la comparativa entre las tres codificaciones tienen una interpretación clara. La longitud media de la codificación Plain Huffman es la más próxima a la entropía debido a que realiza un aprovechamiento completo de los posibles valores de los símbolos (excepto los valores no utilizados para mantener la restricción de prefijo libre). La codificación Tagged Huffman trabaja con un número menor de símbolos del vocabulario de salida, debido a la reserva del bit de marca de comienzo de código y a la necesidad de mantener igualmente la restricción de prefijo libre, lo cual afecta claramente a la longitud media de los códigos obtenidos y, por

tanto, al porcentaje de compresión final, que se ve incrementado en un 11 %.

La codificación Densa con Post-Etiquetado utiliza también un bit de marca, pero en este caso para señalar el fin de un código, de modo que se realiza un mayor aprovechamiento de los códigos lo que provoca que la longitud media de los códigos sea más reducida que para la codificación Tagged Huffman.

Otra de las conclusiones obtenidas del estudio es cómo afecta en los porcentajes de compresión la necesidad de incorporar el vocabulario al texto comprimido, sobre todo cuando los textos son pequeños. Como puede observarse en las Tablas 4.8 y 4.10, al igual que sucede con las codificaciones Plain y Tagged Huffman, la codificación Densa con Post-Etiquetado alcanza ratios de compresión aceptables cuando los textos originales tienen un tamaño de al menos 10 MB.

Por último, los experimentos demuestran que el modelo de Zipf no es lo suficientemente preciso para representar las distribuciones de textos en lenguaje natural. De hecho, en ocasiones se han alcanzado mejores ratios de compresión con un valor de θ más pequeño.

5

Compresión de textos en lenguas romances

5.1. Introducción

Las técnicas de compresión de textos basadas en palabras, como las codificaciones Plain y Tagged Huffman de Moura *et al.* o la nueva codificación Densa con Post-Etiquetado desarrollada en este trabajo, presentan un buen comportamiento cuando se emplean para la compresión del inglés. Sin embargo, no se adaptan bien a lenguas romances que presentan un mayor vocabulario y una distribución de las frecuencias de las palabras bastante diferente. La variabilidad gramatical de estas lenguas penaliza tanto la compresión como la eficiencia a la hora de aplicar técnicas de Recuperación de Textos sobre los documentos comprimidos.

Precisamente, la necesidad de implementar técnicas de Recuperación de Textos ha provocado el desarrollo de herramientas que facilitan el análisis de textos [7, 14, 17]. Entre estas herramientas se encuentran los *stemmers (lematizadores)* que representan todas las variaciones morfológicas de una palabra mediante un único término (o *lema*). Se trata de una herramienta importante para lenguas romances como el español, portugués o gallego pues, a diferencia del inglés, las palabras en estas lenguas tienen un amplio

número de formas gramaticales debido a variaciones de género y número, aumentativos, diminutivos, superlativos y paradigmas verbales.

El problema de la lematización es que, aunque mejora de forma considerable la eficiencia de las búsquedas, no permite volver a reproducir el texto original dado que las palabras se sustituyen por sus lemas. Por ello no resulta una solución viable para su aplicación sobre textos comprimidos.

En este trabajo se ha desarrollado una nueva herramienta lingüística, denominada *segmentación*, que es similar a la lematización pero a diferencia de ésta permite la recreación del texto original. En este caso, cada palabra se divide en dos partes: raíz y sufijo, que se almacenan en ficheros separados. La eficiencia de la recuperación es similar al caso de la lematización porque las búsquedas se aplican únicamente sobre las raíces, que forman un vocabulario mucho más reducido al original. La ventaja es que, a la hora de mostrar los resultados, las palabras originales pueden reconstruirse concatenando su raíz y sufijo.

Antes de presentar la nueva técnica de preprocesado, se explica con más detalle la problemática asociada a la compresión de textos en lenguas con un amplio vocabulario, como ocurre con las lenguas romances. Este estudio teórico se amplia con un estudio experimental, en la Sección 5.3, donde se presenta una comparativa sobre la compresión mediante las técnicas Huffman basadas en palabras y nuestra codificación Densa con Post-Etiquetado de textos escritos en lenguas romances (castellano, portugués y gallego, concretamente) frente a *corpus* en inglés. Los resultados obtenidos apoyan la necesidad de adecuar de estas técnicas de compresión cuando se aplican sobre lenguas con grandes vocabularios. Como consecuencia de ello, en la Sección 5.4 se plantea la adaptación de las técnicas basadas en palabras presentadas anteriormente mediante la utilización de algoritmos de segmentación para preprocesar los textos en lenguas romances antes de realizar la compresión. En la Sección 5.5 se muestran los resultados obtenidos tras la aplicación del algoritmo de segmentación sobre los *corpus* de lenguas romances anteriores. Finalmente, en la Sección 5.7 se presentan las conclusiones obtenidas.

5.2. Influencia de la variación morfológica en la compresión de documentos

La principal característica diferenciadora entre un *corpus* en inglés y otro en una lengua romance es la variación morfológica existente en este último. Esto da lugar a que una misma palabra en inglés se pueda traducir generalmente por una familia de palabras de raíz común con variación en la desinencia. Por ejemplo, las tres formas típicas de los verbos en inglés corresponden a más de sesenta formas en lenguas romances, los adjetivos tienen cuatro posibles terminaciones (corresponden a variaciones de género y número), y así sucesivamente.

Este hecho conlleva que nos encontremos con un tamaño de vocabulario y una distribución de frecuencias notablemente diferentes según el *corpus* tratado pertenezca a lenguas romances o al inglés. En el caso de las lenguas romances el vocabulario es mayor y posee una distribución de frecuencias más homogénea; es decir, menos sesgada. Esto provoca una reducción en la eficiencia de la compresión y de las búsquedas.

La disminución en el porcentaje de compresión viene dada porque, recordemos, las técnicas de compresión semiadaptativas (como las codificaciones Huffman o nuestra codificación Densa con Post-Etiquetado) obienen mejores resultados si la distribución de probabilidad del vocabulario a codificar está más sesgada (el caso extremo sería cuando los símbolos siguen una distribución exponencial, tal y como se vio en el Capítulo 4). En un vocabulario más sesgado habrá un pequeño número de palabras con muy alta frecuencia (que corresponden, por tanto, a la gran mayoría de las palabras existentes en el texto) a las cuales se les asignarán los códigos más cortos, mientras que las palabras asignadas a códigos más largos tendrán una probabilidad de aparición muy pequeña, por lo que afectarán menos al tamaño final del texto comprimido. En cambio, si el vocabulario es más uniforme habrá un gran número de palabras con frecuencia media a las que se les asignan códigos grandes. Es decir, los códigos cortos de las palabras más frecuentes no compensarán los códigos largos de las menos frecuentes si la diferencia entre las palabras más frecuentes y las menos frecuentes no es lo suficientemente grande. En consecuencia, la longitud media de los códigos

será mayor.

El hecho de que la longitud media de los códigos crezca para el caso de las lenguas romances significa que el porcentaje de compresión final se verá afectado negativamente. Recordemos que, como se ha visto en la Sección 2.2.2, cuánto mayor sea la longitud de los códigos mayor será también la redundancia de la codificación (la diferencia respecto a la entropía) y, consecuentemente, el porcentaje de compresión será peor.

Durante este capítulo se va a hacer uso de las leyes para capturar la variabilidad de los *corpus*. Una de ellas es la *Ley de Zipf*, que ya ha sido empleada en el Capítulo 4 para ilustrar el estudio de nuestra codificación Densa con Post-Etiquetado, y la segunda es la *Ley de Heaps* que se detalla a continuación.

5.2.1. Ley de Heaps

La *Ley de Heaps* es una ley empírica que relaciona el tamaño del vocabulario de entrada y el número total de palabras en el texto. La Ley de Heaps [17] indica que un texto de $O(n)$ palabras tiene un vocabulario de tamaño del orden $O(n^{\beta})$ para $0 < \beta < 1$. En estudios previos [4, 29, 37] se ha demostrado que, para textos en inglés, el valor de β es lo suficientemente bajo (entre $0,4$ y $0,6$) para asegurar la eficiencia en la compresión utilizando la técnica de compresión Huffman orientada a byte basada en palabras (Plain o Tagged Huffman) cuando el texto original tiene un tamaño superior a 10 MB. Sin embargo, para textos escritos en lenguas romances (con más variación gramatical), el valor de β es mayor (como se podrá observar en los estudios que se presentan en la Sección 5.3) y, por tanto, la compresión obtenida es peor debido a que el vocabulario que debe almacenarse con el texto comprimido es más grande. Como consecuencia, el tamaño mínimo del texto para obtener un compresión aceptable parece que, *a priori*, debe ser mayor al de un texto en inglés.

La Figura 5.1 ilustra los diferentes tamaños de vocabulario que se obtienen con textos de distintos tamaños tomando los valores de $\beta = 0,4$, $0,5$ y $0,6$. Se puede observar que para un tamaño del *corpus* dado, el tamaño del vocabulario aumenta a medida que lo hace el parámetro β.

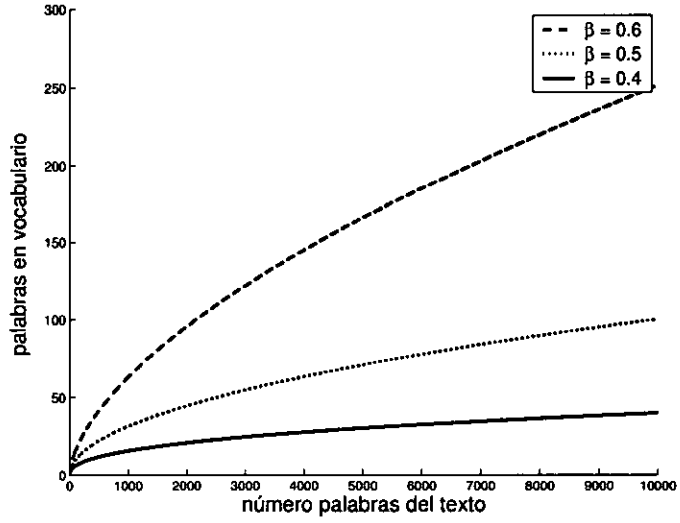


Figura 5.1: Tamaño del vocabulario para distintos valores de β

5.3. Estudios experimentales

Para los experimentos se ha utilizado un *corpus* para cada una de las tres lenguas romances (gallego, portugués y castellano), construidos *ad hoc* a partir de diferentes fuentes, como documentos Web, *e-books* y documentos de bibliotecas digitales, entre otros, la mayoría disponibles en Internet. Dentro de cada *corpus*, los textos se clasificaron en cinco categorías:

- *PER*: medios de comunicación (prensa, radio, televisión)
- *ECO*: economía y sociedad
- *JUR*: textos legales
- *LIT*: lengua y literatura
- *TEC*: ciencia y tecnología

Las abreviaturas *GALL*, *PORT* y *CAST* (para gallego, portugués y castellano, respectivamente) representan los resultados del procesamiento conjunto de todos los textos de cada lengua.

Por otro lado, a modo de comparativa, se ha seleccionado un subconjunto de las colecciones en inglés empleadas en el capítulo anterior para ilustrar nuestra técnica de compresión. En concreto, los *corpus* utilizados aquí corresponden a la colección $TREC - 4$ [16] y son: $ZIFF$ Ziff Data 1989-90, CR Congressional Record 1993, $FT91$ Financial Times 1991, $FT92$ Financial Times 1992, $FT93$ Financial Times 1993 y $FT94$ Financial Times 1994.

Todos los *corpus* anteriores se han comprimido aplicando las dos codificaciones propuestas por Moura *et al.* (Plain y Tagged Huffman) y la nueva codificación Densa con Post-Etiquetado. Además, hemos observado las variaciones en el tamaño del vocabulario calculando los valores correspondientes a las leyes de Heaps y Zipf.

5.3.1. Tamaño de los vocabularios

En primer lugar se ha realizado un estudio sobre el tamaño del vocabulario para los diferentes *corpus*, para demostrar así si el número de palabras diferentes para castellano, portugués y gallego es mayor que el correspondiente a un texto del mismo tamaño en inglés. Para ello, se han extraído las palabras de los diferentes *corpus*, utilizando la aproximación *spaceless* donde, recordemos, las palabras y los separadores (conjunto de caracteres no alfanuméricos entre dos palabras consecutivas) comparten un mismo vocabulario.

En la Tabla 5.1 se presenta el número total de palabras y el número de palabras que componen el vocabulario de cada uno de los *corpus*. En la última columna se puede observar claramente cómo se ha producido un incremento importante del vocabulario para las lenguas romances.

Los valores de β de la Ley de Heaps confirman los datos anteriores. Así, se ha aplicado la función de regresión lineal de la forma $f(x) = x^{\beta}$, obteniéndose los valores de β de $0,513143$ para inglés, seguido por un $0,585411$ para el gallego, $0,664178$ para castellano y $0,688383$ para el *corpus* de portugués.

Se han realizado pruebas estadísticas que demuestran la existencia de

corpus	Total Palabras	Vocabulario Palabras	%
Ziff	40 548 114	221 443	0,55 %
CR	9 445 990	114 174	1,21 %
FT91	3 059 634	75 597	2,47 %
FT92	36 518 075	284 904	0,78 %
FT93	41 772 135	291 322	0,70 %
FT94	43 039 879	295 023	0,69 %
		Media	**1,06 %**

corpus	Total Palabras	Vocabulario Palabras	%
JUR	788 180	38 067	4,83 %
PER	2 142 321	67 258	3,14 %
TEC	2 176 623	85 525	3,93 %
ECO	2 666 897	78 374	2,94 %
LIT	18 323 253	311 542	1,70 %
CAST	26 097 274	382 332	1,47 %
		Media	**3,00 %**

corpus	Total Palabras	Vocabulario Palabras	%
JUR	299 964	16 916	5,64 %
PER	6 228 650	137 543	2,21 %
TEC	418 564	22 751	5,44 %
ECO	10 829 005	232 092	2,14 %
LIT	2 967 862	110 790	3,73 %
PORT	20 744 045	298 276	1,44 %
		Media	**3,43 %**

corpus	Total Palabras	Vocabulario Palabras	%
JUR	3 751 606	87 168	2,32 %
PER	281 503	43 375	15,41 %
TEC	411 417	22 939	5,58 %
ECO	2 992 415	113 166	3,78 %
LIT	2 101 286	137 159	6,53 %
GALL	9 547 764	277 699	2,91 %
		Media	**6,09 %**

Tabla 5.1: Número total de palabras y tamaño de los vocabularios para los *corpus* procesados

una relación significativa entre el tamaño del vocabulario y el número de palabras del *corpus*, con una confianza del 99 % para el caso de los *corpus* en inglés, castellano y portugués, y del 95 % para el *corpus* gallego. Los tests estadísticos también concluyen que el modelo multiplicativo que propone la Ley de Heaps explica un gran porcentaje de la variabilidad de los datos (concretamente, el 96 %, 99 %, 99 % y 81 % para inglés, castellano, portugués y gallego, respectivamente). De ello se deduce que, en general, los modelos inferidos para las diferentes lenguas son muy representativos, aunque algo menos para el caso del gallego.

En base a los resultados anteriores, se puede concluir que el valor del parámetro β de la Ley de Heaps es asintóticamente mayor para todos los casos de las lenguas romances frente a los corpus en inglés. Por lo tanto, el número de palabras en los vocabularios romances será siempre mayor que para el inglés.

5.3.2. Ratio de compresión

Una vez realizado este primer análisis sobre el tamaño del vocabulario, donde se demuestra el importante incremento de tamaño para el caso de las lenguas romances, pasamos a estudiar ahora cómo afecta a la compresión. Para ello, se han aplicado las codificaciones Plain y Tagged Huffman y la nueva codificación Densa con Post-Etiquetado sobre los *corpus* anteriormente descritos.

En las tres últimas columnas de la Tabla 5.2 se muestran las longitudes medias de los códigos para las codificaciones Plain Huffman, Densa con Post-Etiquetado y Tagged Huffman, respectivamente. Como se puede observar, las longitudes medias para el caso de los *corpus* romances han crecido respecto a los *corpus* en inglés.

Los porcentajes de compresión alcanzados para cada una de las codificaciones se muestran en la Tabla 5.3. Para calcular los porcentajes de compresión se ha tenido en cuenta el tamaño del texto comprimido y del vocabulario que debe almacenarse junto con el texto. Para que el vocabulario afecte lo menos posible al porcentaje de compresión se ha comprimido utilizando la codificación Hufman binaria basada en caracteres.

corpus	Long. Media Palabras	Longitud Media Codificación		
		Plain	Densa	Tagged
Ziff	4,0682	1,449977	1,490903	1,606416
CR	4,4062	1,475534	1,520167	1,638070
FT91	4,2276	1,455481	1,497421	1,612230
FT92	4,2225	1,465063	1,504248	1,627545
FT93	4,1998	1,447384	1,489149	1,613115
FT94	4,2085	1,447626	1,489669	1,613737
Media	4,2221	1,456844	1,498593	1,618519
corpus	Long. Media Palabras	Longitud Media Codificación		
		Plain	Densa	Tagged
JUR	4,4683	1,440885	1,481127	1,578283
PER	4,6791	1,439057	1,497705	1,603345
TEC	4,6706	1,522333	1,564251	1,679987
ECO	4,8067	1,520462	1,560929	1,665515
LIT	4,3524	1,546468	1,586692	1,714396
CAST	4,4557	1,569430	1,608999	1,739183
Media	4,5721	1,506439	1,549951	1,663452
corpus	Long. Media Palabras	Longitud Media Codificación		
		Plain	Densa	Tagged
JUR	4,7881	1,416823	1,465282	1,567951
PER	4,3238	1,508331	1,545545	1,651269
TEC	3,9716	1,371859	1,410042	1,506379
ECO	4,3065	1,532751	1,568919	1,682186
LIT	4,0586	1,528904	1,574729	1,692698
PORT	4,2764	1,549008	1,586333	1,704661
Media	4.2875	1.484613	1.525142	1.634191
corpus	Long. Media Palabras	Longitud Media Codificación		
		Plain	Densa	Tagged
JUR	4,7425	1,456155	1,501989	1,603905
PER	4,7621	1,554644	1,618334	1,732802
TEC	4,6269	1,463532	1,510198	1,625421
ECO	4,7048	1,555748	1,595974	1,707939
LIT	4,3117	1,630578	1,675906	1,798625
GALL	4,6316	1,589519	1,629133	1,759678
Media	4,6299	1,541696	1,588589	1,704728

Tabla 5.2: Longitudes medias para las codificaciones Plain Huffman, Densa y Tagged Huffman

corpus	Tamaño	Plain	Denso	Tagged
ZIFF	185 417 980	32,34 %	33,24 %	35,76 %
CR	51 085 545	28,45 %	29,27 %	31,45 %
FT91	14 749 355	33,05 %	33,92 %	36,30 %
FT92	175 449 248	31,52 %	32,33 %	34,90 %
FT93	197 586 334	31,53 %	32,42 %	35,04 %
FT94	203 783 923	31,50 %	32,39 %	35,01 %

corpus	Tamaño sin comprimir	Plain	Denso	Tagged
JUR	4 034 740	33,34 %	34,13 %	36,02 %
PER	11 228 495	30,70 %	31,82 %	33,83 %
TEC	12 772 811	30,00 %	30,71 %	32,68 %
ECO	14 684 576	30,71 %	31,45 %	33,35 %
LIT	102 721 471	29,32 %	30,04 %	32,31 %
CAST	145 442 093	29,73 %	30,44 %	32,78 %

corpus	Tamaño sin comprimir	Plain	Denso	Tagged
JUR	1 869 104	28,19 %	28,96 %	30,61 %
PER	37 124 140	27,30 %	27,92 %	29,70 %
TEC	2 481 622	28,03 %	28,67 %	30,30 %
ECO	61 842 062	28,86 %	29,49 %	31,47 %
LIT	14 157 793	36,18 %	37,14 %	39,62 %
PORT	117 474 721	28,76 %	29,42 %	31,51 %

corpus	Tamaño sin comprimir	Plain	Denso	Tagged
JUR	20 094 747	30,00 %	30,84 %	32,75 %
PER	1 682 774	39,93 %	41,00 %	42,91 %
TEC	2 195 731	33,05 %	33,93 %	36,09 %
ECO	17 145 486	30,81 %	31,51 %	33,46 %
LIT	13 377 423	31,16 %	31,88 %	33,80 %
GALL	54 496 161	30,95 %	31,65 %	33,93 %

Tabla 5.3: Ratios de compresión para las codificaciones Plain Huffman, Densa con Post-Etiquetado y Tagged Huffman

Obsérvese cómo aunque las longitudes medias de los códigos generados por las tres codificaciones son mayores para los *corpus* romances, tal como se ha mostrado en la Tabla 5.2, sin embargo los porcentajes de compresión no han empeorado tanto como se esperaba. El motivo hay que buscarlo en la longitud media de las palabras.

Como se muestra en la segunda columna de la Tabla 5.2, la longitud media ponderada de las palabras es mayor para los textos romances que para los textos en inglés. Dado que las palabras son más largas, el número total de palabras que forman un texto romance de nuestro corpus es menor que el existente en un *corpus* inglés de igual tamaño. De este modo, lo que sucede es que, aunque los códigos empleados para la compresión son algo más largos para los textos romances, este incremento de longitud de los códigos se ve compensado porque hay menos palabras que codificar y, además, son más largas. Cada código, entonces, realmente está codificando (es decir, sustituyendo) un número de caracteres del texto original mayor que en caso de que el texto sea en inglés consiguiéndose, por tanto, una mejor compresión que la esperada.

Estos resultados no reflejan fielmente la realidad y están condicionados por el origen y la calidad de los textos procesados. Como ya se ha comentado en el momento de la descripción de los *corpus*, los textos romances han sido construidos *ad hoc*, ante la imposibilidad de disponer de *corpus* de prueba como sucede para el inglés. Estos textos romances, la mayoría de ellos obtenidos a través de Internet, disponen de un número más elevado de secuencias largas de caracteres no alfanuméricos (líneas en blanco, tabulaciones, tablas, etc.) que en caso de los *corpus* en inglés. Dado que cada separador corresponde al conjunto de caracteres no alfanuméricos existente entre dos palabras, el tamaño de los separadores es mucho mayor que en inglés lo que, consecuentemente, provoca ese aumento en la longitud media de las entradas del vocabulario mencionado.

Paralelamente al cálculo de los porcentajes de compresión, se ha realizado un estudio sobre la distribución de las frecuencias de las palabras en los diferentes *corpus* aplicando la *Ley de Zipf*. Para calcular el valor de θ se utilizó, al igual que en el capítulo anterior, la función de regresión de la forma $f(x) = Ax^{-\theta}$ entre la posición que ocupa cada palabra en el vocabulario y

corpus	Valor θ
ZIFF	1,744346
CR	1,634076
FT91	1,449878
FT92	1,630996
FT93	1,647456
FT94	1,649428
Media θ	1,626030

corpus	θ Castellano	θ Portugués	θ Gallego
JUR	1,278427	1,300601	1,434490
PER	1,302528	1,467588	0,900207
TEC	1,324321	1,226397	1,198054
ECO	1,395134	1,438089	1,334486
LIT	1,481170	1,298492	1,154273
TOTAL	1,510951	1,528964	1,332616
Media θ	1,382089	1,376689	1,225688

Tabla 5.4: Valor θ de la Ley de Zipf

su frecuencia de aparición en el *corpus* (recordemos que $A = \frac{1}{\sum_{i\geq 1} 1/i^{\theta}}$). Los resultados obtenidos, que se presentan en la Tabla 5.4, muestran que los valores de θ para las lenguas romances son menores que los obtenidos para los *corpus* en inglés o, lo que es lo mismo, que sus distribuciones son menos sesgadas. Es decir, en los textos romances el número de palabras con alta frecuencia es menor, y hay un elevado número de palabras con baja frecuencia a los que se les asignan códigos grandes.

Para confirmar estas afirmaciones, en la Tabla 5.5 y en la Figura 5.2 se muestra la reducción en el tamaño de los *corpus* romances (CAST, PORT y GALL) y el *corpus* inglés ZIFF provocada por la sustitución de las palabras más frecuentes por sus códigos. El cálculo se ha realizado teniendo en cuenta la diferencia de longitud entre cada palabra y el código Denso con Post-Etiquetado que se le asigna, y la frecuencia de aparición de dicha palabra en el texto original. Así, por ejemplo, en la primera entrada de la tabla correspondiente al corpus *ZIFF* se muestra el número de bytes (20508548, concretamente) en que se ha reducido el fichero original debido a la codificación de sus 50 palabras más frecuentes. Esto supone un 16,41 % del número total de bytes reducidos al comprimir el fichero original.

N° Pal.	ZIFF
50	20 508 548 (16,41 %)
100	29 015 950 (23,22 %)
200	37 681 964 (30,15 %)
500	50 799 956 (40,65 %)
1000	62 267 812 (49,83 %)

N° Pal.	CAST	PORT	GALL
50	11 792 263 (11,40 %)	7 891 984 (9,33 %)	3 915 936 (10,06 %)
100	15 373 323 (14,86 %)	10 348 263 (12,24 %)	5 509 175 (14,15 %)
200	19 034 974 (18,39 %)	13 257 844 (15,68 %)	6 878 544 (17,66 %)
500	25 646 458 (24,79 %)	17 630 072 (20,85 %)	9 666 986 (24,82 %)
1000	32 115 056 (31,04 %)	22 939 760 (27,13 %)	12 482 776 (32,05 %)

Tabla 5.5: Reducción del tamaño del texto a comprimir originado por las palabras más frecuentes

Como puede observarse en la tabla y figuras anteriores, la reducción es más importante para el caso del inglés, donde la codificación de las 1000 palabras más frecuentes da lugar a prácticamente el 50 % de la reducción total en el tamaño del fichero, mientras que para las lenguas romances apenas alcanza el 30 %. Esto demuestra que la distribución de las palabras en inglés está mucho más sesgada, coincidiendo con lo indicado por los valores de θ.

Es fácil deducir la razón de la diferencia entre las tres lenguas romances y el inglés. En lenguas romances, la misma raíz tiene más sufijos diferentes que para el inglés, e incluso verbos muy utilizados como es el caso de *to have* o *to be* distribuyen su frecuencia entre muchas más palabras diferentes en lenguas romances que en inglés.

5.4. Usando segmentación para reducir el vocabulario

En la sección anterior hemos estudiado cómo el aumento del vocabulario para los textos romances provoca una reducción en la compresión cuando se aplican técnicas basadas en palabras. Pero este no es el único problema que origina manejar vocabularios grandes. La eficiencia en la recuperación

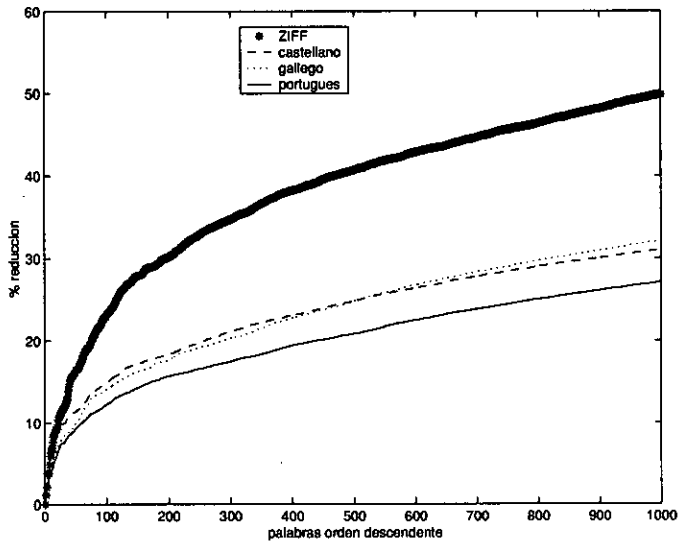


Figura 5.2: Porcentaje de reducción del tamaño del texto a comprimir originado por las 1000 palabras más frecuentes para *corpus* en inglés y en lenguas romances

también se ve claramente afectada.

La mejora de la eficiencia a la hora de realizar búsquedas sobre textos con amplios vocabularios es un problema abierto dentro del entorno de la Recuperación de Textos [7, 14]. Generalmente, la solución adoptada conlleva un preprocesamiento del texto donde se llevan a cabo operaciones tales como la eliminación de *stopwords* (palabras no relevantes), la construcción de *thesaurus* (palabras relacionadas con las existentes en el texto, tales como sinónimos o antónimos) o la *lematización(stemming)* (reducción de una palabra a su raíz), de la que hablaremos con más detalle a continuación. Con todo ello se pretende reducir el tamaño del vocabulario y, a la vez, ampliar el resultado de las consultas incorporando las apariciones de las variantes morfológicas (de género, número, tiempos verbales, etc.) de la palabra buscada.

El problema de las herramientas como los lematizadores es, como ya se ha mencionado, que no son aplicables sobre textos comprimidos, ya que imposibilitan la reproducción del texto original. Por ello en esta sección se

describe una nueva técnica para preprocesar el texto antes de comprimirlo que se basa en la utilización de un algoritmo denominado *de segmentación.* La segmentación conserva las ventajas de la lematización pero permite reconstruir de nuevo el texto original en el momento de la descompresión, como se verá más adelante.

5.4.1. Lematización

La *lematización* es el proceso de unir las diferentes variaciones morfológicas de una palabra bajo una representación común, denominada *lema*[7, 14]. El *lema* se obtiene como resultado de eliminar los sufijos (y en algunos casos los prefijos también) de la palabra. Por ejemplo, las palabras *cantar, cantante, cantaba y cantaría* pueden reducirse al lema *cantar.* En el proceso, algunos lematizadores utilizan las reglas gramaticales de las lenguas con las que trabajan. Habitualmente las reglas tienen la forma siguiente:

{*Sufijo, Tamaño de la raíz, Sustitución, Lista de excepciones*}

Una regla se aplica a una palabra si ésta termina en *Sufijo*, si no está en la *Lista de excepciones* y la longitud de la raíz es al menos *Tamaño de la raíz.* Cuando la cadena *Sustitución* no es una cadena vacía, se sustituye *Sufijo* por ella.

Ejemplo 5.4.1 Para eliminar los plurales de las palabras en inglés podría crearse un algoritmo de lematización que utilizase las siguientes reglas:

si la palabra acaba en *ies* pero no en *eies* o *aies*
entonces $ies \rightarrow y$
si la palabra acaba en *es* pero no en *aes*, *ees* o *oes*
entonces $es \rightarrow e$
si la palabra acaba en *s* pero no en *us* o *ss*
entonces $s \rightarrow \emptyset$

de modo que sólo se aplique la primera regla válida. □

La mayoría de los lematizadores actuales establecen un orden de análisis de las reglas, de modo que siempre se elimina en primer lugar el sufijo mayor. Así, de acuerdo con las reglas del ejemplo anterior y considerando que se analizan de un modo secuencial, la palabra *skies* da lugar al lema *sky* y no a *skie*.

Un algoritmo de lematización basado en la eliminación de sufijos es el *algoritmo de Porter* [32]. Debido a su sencillez es ampliamente utilizado y ha sido adaptado a diferentes lenguas como inglés, lenguas romances, germánicas y escandinavas [33].

El algoritmo de Porter se compone de reglas simples de la forma *condición/acción*. Las reglas se agrupan en conjuntos (reglas correspondientes a sufijos de género, de número, aumentativos, etc). Dentro de cada conjunto se evalúan en un determinado orden de modo que en cada iteracción sólo se pueda aplicar una regla y esta regla eliminará el sufijo más largo posible de la palabra analizada.

5.4.2. Segmentación

La ventaja principal de utilizar lematización sobre un texto es, como ya se ha comentado, que al representar todas las variaciones morfológicas de una misma palabra por un único lema se mejoran enormemente las búsquedas. El problema estriba en que no es una herramienta aplicable cuando los textos se almacenan en formato comprimido, ya que no permite la recreación del texto original.

En este trabajo se ha desarrollado una nueva herramienta lingüística, denominada *segmentación*, que posee unas ventajas similares a las ofrecidas por los algoritmos de lematización, pero que pueden utilizarse para preprocesar documentos antes de llevar a cabo la compresión de los mismos.

En términos generales, la *segmentación* es similar a la lematización salvo que en la segmentación las palabras se dividen en dos partes: la *raíz* y el *sufijo*. De este modo, palabras como *andaban* y *cantaban* se sustituyen, en el texto original, por *and-aban* y *cant-aban*, respectivamente.

Ejemplo 5.4.2 El texto *cantaban andaban jugaban cantaría andaría jugaría cantaré andaré jugaré*, después de la segmentación queda de la forma *cant-aban and-aban jug-aban cant-aría and-aría jug-aría cant-aré and-aré jug-aré.* □

Para segmentar las palabras se utilizan un conjunto de reglas similares a las empleadas por Porter en su algoritmo de lematización.

Una vez realizada la segmentación, el texto estará formado por dos tipos de palabras: raíces y sufijos, dando lugar a dos vocabularios diferentes. Como se verá en los resultados empíricos de la siguiente sección, el tamaño de cada uno de estos vocabularios (raíces y sufijos) es mucho menor que el correspondiente a las palabras del texto original.

Ejemplo 5.4.3 Para el ejemplo anterior, se puede observar que antes de la segmentación el vocabulario de entrada estaría formado por nueve palabras, mientras que tras la segmentación el vocabulario de raíces tendrá tres símbolos (*cant, and, jug*), al igual que el de sufijos (*aban, aría, aré*). □

Las raíces y los sufijos (junto con los separadores) se almacenan en ficheros separados. Para reconstruir el texto original basta con seguir una alternancia estricta entre ambos ficheros, una vez conocido si el documento comienza por una palabra (es decir, por una raíz) o por un separador.

Después de la segmentación ya se puede llevar a cabo el proceso de compresión de ambos ficheros. La compresión del fichero de raíces debe realizarse mediante una técnica basada en palabras, como las codificaciones Plain Huffman, Tagged Huffman o el nuevo esquema de codificación Densa con Post-Etiquetado, para poder posteriormente realizar búsquedas directas sobre el texto comprimido. La codificación de este fichero de raíces da lugar a mejores porcentajes de compresión que los alcanzados por el texto original, como se verá en la Sección 5.5, debido a que las raíces tienen una frecuencia de aparición más alta que las palabras completas.

Respecto a la compresión del fichero de sufijos puede optarse por utilizar la misma técnica de compresión que la empleada para el fichero de raíces u otra (incluso una basada en caracteres). La elección de la técnica de

compresión en este caso depende del tipo de búsquedas que se desee realizar sobre este fichero, tal como se verá más adelante.

Antes de explicar el proceso de recuperación sobre los ficheros segmentados, en la siguiente sección se presentan los resultados obtenidos tras la aplicación del algoritmo de segmentación sobre nuestros *corpus* romances, y su compresión posterior.

5.5. Resultados experimentales tras la segmentación

Para demostrar que el problema de la compresión de las lenguas romances es debido a su variedad de sufijos, se ha realizado la segmentación de los *corpus* originales para comprobar si los parámetros en las leyes de Heaps y Zipf resultaban más adecuados. Nuestro objetivo es que los textos en lenguas romances analizados alcancen valores más próximos a los obtenidos para los textos en inglés para los parámetros β y θ, de modo que la compresión sea igualmente efectiva.

Previamente, en la Tabla 5.6 se muestra el número de palabras del vocabulario antes y después de la segmentación para los *corpus* romances. En las columnas 2 y 3 se puede observar la diferencia entre el número total de palabras en el texto sin segmentar y el número total de raíces tras la segmentación. En las columnas 4 y 5 se muestra el número de palabras que forman el vocabulario sin segmentar y el número de raíces diferentes tras la segmentación. La sexta columna corresponde al porcentaje de reducción de tamaño del vocabulario de raíces frente al de las palabras, que supera de media el 40 %. Por último, se muestra el número de sufijos diferentes extraídos del texto. Obsérvese como, en la gran mayoría de los casos, incluso la suma de las raíces y sufijos sigue siendo menor al número de palabras que componen el vocabulario de entrada sin segmentar.

Tras el proceso de segmentación, se ha calculado de nuevo el valor de β y, como era de esperar, los valores se reducen sensiblemente como se puede apreciar en la Tabla 5.7.

corpus	Número Total		Tamaño Vocabulario			
	Palabras	Raíces	Palabras	Raíces	Dec	Sufijos
JUR	788 180	621 251	38 067	20 769	45 %	11 340
PER	2 142 321	1 553 194	67 258	37 713	44 %	13 167
TEC	2 176 623	1 878 808	85 525	52 911	38 %	39 547
ECO	2 666 897	2 145 514	78 374	41 924	47 %	23 399
LIT	18 323 253	16 694 053	311 542	173 604	44 %	106 215
CAST	26 097 274	22 892 820	382 332	215 606	44 %	153 066
				Media	44 %	

corpus	Número Total		Tamaño Vocabulario			
	Palabras	Raíces	Palabras	Raíces	Dec	Sufijos
JUR	299 964	263 464	16 916	10 271	39 %	10 461
PER	6 228 650	6 042 240	137 543	74 639	46 %	33 553
TEC	418 564	372 885	20 751	17 839	22 %	13 017
ECO	10 829 005	9 882 193	232 092	163 091	30 %	61 780
LIT	2 967 862	2 431 056	110 790	45 306	59 %	29 550
PORT	20 744 045	18 991 838	298 276	191 553	36 %	106 584
				Media	39 %	

corpus	Número Total		Tamaño Vocabulario			
	Palabras	Raíces	Palabras	Raíces	Dec	Sufijos
JUR	3 751 606	3 189 985	87 168	58 911	32 %	39 888
PER	281 503	246 518	43 375	29 765	31 %	13 843
TEC	411 417	341 209	22 939	12 469	46 %	6 867
ECO	2 992 415	2 665 421	113 166	65 408	42 %	32 921
LIT	2 101 286	2 129 007	137 159	76 923	44 %	47 493
GALL	9 547 764	8 572 140	277 699	168 204	39 %	113 301
				Media	39 %	

Tabla 5.6: Número total de palabras y tamaño de los vocabularios antes y después de segmentar

Lenguas	β sin segmentar	β raíces
Castellano	0,664178	0,638824
Portugués	0,688383	0,661193
Gallego	0,585411	0,581727

Tabla 5.7: Valores del parámetro β de la Ley de Heaps antes y después de la segmentación

corpus	θ sin segmentar	θ Raíces	θ Sufijos
JUR	1,278427	1,456140	1,336168
PER	1,302528	1,469344	1,607051
TEC	1,324321	1,459791	1,173129
ECO	1,395134	1,571330	1,365141
LIT	1,481170	1,641780	1,280512
CAST	1,510951	1,642198	1,273388
	Media θ	1,540097	1,339232
corpus	θ sin segmentar	θ Raíces	θ Sufijos
JUR	1,300601	1,488468	1,165305
PER	1,467588	1,674396	1,504084
TEC	1,226397	1,375461	1,184890
ECO	1,438089	1,443072	1,403563
LIT	1,298492	1,595871	1,293901
PORT	1,528964	1,598433	1,366827
	Media θ	1,529284	1,319762
corpus	θ sin segmentar	θ Raíces	θ Sufijos
JUR	1,434490	1,516985	1,386932
PER	0,900207	1,042315	1,021415
TEC	1,198054	1,378669	1,365721
ECO	1,334486	1,485456	1,275109
LIT	1,154273	1,367239	1,157330
GALL	1,332616	1,429557	1,226292
	Media θ	1,370037	1,238800

Tabla 5.8: Valores de θ para las raíces y los sufijos tras la segmentación de los *corpus* romances

De igual modo, tras el proceso de segmentación los valores de θ han aumentado respecto al valor obtenido para los textos sin segmentar, como se muestra en la Tabla 5.8. Obsérvese cómo los valores asociados a las raíces oscilan entre 1,4 y 1,6 en la mayoría de los casos, lo que indica que su distribución está más sesgada (ver Sección 4.1.3).

Para apoyar estos datos, en la Figura 5.3 se presenta la distribución seguida por las 50 palabras más frecuentes de nuestros *corpus* romances antes y después de la segmentación. Obsérvese cómo las distribuciones para las raíces (líneas discontinuas) están más sesgadas que para las palabras del vocabulario sin segmentar (líneas continuas).

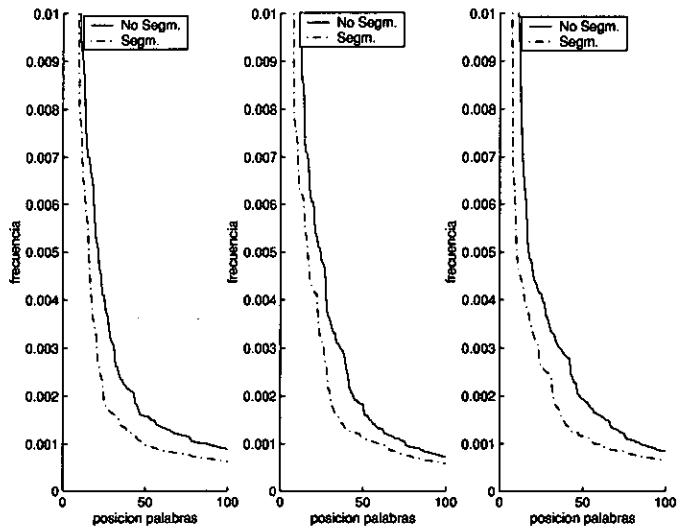


Figura 5.3: De izquierda a derecha, frecuencias de las 50 palabras más frecuentes para los *corpus* castellano, portugués y gallego antes y después de segmentar

Por último, en la Tabla 5.9 se muestran las longitudes medias de los códigos y el porcentaje de compresión (incluyendo el vocabulario en el documento comprimido) tras aplicar la codificación Densa con Post-Etiquetado sobre los ficheros de raíces y sufijos respecto al tamaño del texto original. A modo de comparativa, en la segunda columna de la tabla se muestra el porcentaje de compresión correspondiente al texto comprimido con la misma codificación sin realizar segmentación, y en la última aparece la suma de los porcentajes alcanzados por los ficheros de raíces y sufijos.

En base a los resultados presentados en la tabla anterior, podemos concluir que aunque el porcentaje de compresión final (la suma de los porcentajes para los ficheros de raíces y sufijos comprimidos) es mayor al obtenido para el texto sin segmentar, los porcentajes se han reducido de forma considerable para los ficheros por separado. Esto resulta especialmente interesante en el caso de fichero de raíces, pues la mayoría de las búsquedas, como veremos en el siguiente apartado, se aplican sobre este fichero, mucho más pequeño que la versión sin segmentar.

	NoSegm.	Codificación Raíces		Codificación Sufijos		
corpus	Ratio	Long.Media	Ratio	Long.Media	Ratio	Suma
JUR	34,13 %	1,463961	24,78 %	1,141185	19,14 %	43,93 %
PER	31,82 %	1,478194	21,94 %	1,193005	17,05 %	38,98 %
TEC	30,71 %	1,540453	24,62 %	1,182823	19,73 %	44,35 %
ECO	31,45 %	1,521647	23,55 %	1,161381	18,02 %	41,58 %
LIT	30,04 %	1,542873	25,89 %	1,128874	18,98 %	44,87 %
CAST	30,44 %	1,561531	25,31 %	1,156990	18,89 %	44,21 %
	Media	1,518110	Media	1,160710		

	NoSegm.	Codificación Raíces		Codificación Sufijos		
corpus	Ratio	Long.Media	Ratio	Long.Media	Ratio	Suma
JUR	28,96 %	1,486226	22,92 %	1,184955	20,99 %	43,91 %
PER	27,92 %	1,535627	25,74 %	1,140783	18,99 %	44,73 %
TEC	28,67 %	1,497180	25,04 %	1,188409	21,12 %	46,15 %
ECO	29,49 %	1,560897	26,08 %	1,157890	19,07 %	45,15 %
LIT	37,14 %	1,525602	27,31 %	1,140754	20,80 %	48,11 %
PORT	29,42 %	1,566131	26,04 %	1,166832	19,44 %	45,48 %
	Media	1,528611	Media	1,163271		

	NoSegm.	Codificación Raíces		Codificación Sufijos		
corpus	Ratio	Long.Media	Ratio	Long.Media	Ratio	Suma
JUR	30,84 %	1,530607	25,93 %	1,213079	20,73 %	46,66 %
PER	41,00 %	1,608272	31,92 %	1,209065	22,72 %	54,65 %
TEC	33,93 %	1,476922	25,39 %	1,139544	19,57 %	44,96 %
ECO	31,51 %	1,559270	25,97 %	1,152666	19,42 %	45,38 %
LIT	31,88 %	1,607739	28,19 %	1,204354	21,26 %	49,45 %
GALL	31,65 %	1,601952	26,84 %	1,216049	20,69 %	47,52 %
	Media	1,564127	Media	1,189126		

Tabla 5.9: Longitudes medias de códigos y ratios de compresión para raíces y sufijos aplicando codificación Densa con Post-Etiquetado

5.6. Búsquedas sobre texto segmentado

La aproximación propuesta aumenta considerablemente la eficiencia de la recuperación favoreciendo la realización de búsquedas aproximadas sobre las raíces. Dado que las raíces y los sufijos se almacenan en ficheros separados, las búsquedas se limitan al fichero de raíces, que es (como se demuestra en los resultados presentados en la sección anterior) considerablemente más pequeño al que se obtendría en caso de no hacer segmentación. Como resultado de una consulta, se obtienen todas las variaciones morfológicas de una determinada palabra en un *corpus*.

El proceso de búsqueda se puede ejecutar de tres modos diferentes, todos ellos relevantes para lenguas romances. Puede buscarse únicamente sobre raíces, localizando todas las ocurrencias de palabras con la misma raíz. Otra posibilidad es buscar por palabra exacta. Para ello, es necesario segmentar la palabra y posteriormente buscar sus dos componentes en el orden correcto, del mismo modo que se haría en la búsqueda por frase. La última posibilidad es buscar palabras con un sufijo determinado. Por ejemplo, para buscar todos los verbos en tiempo subjuntivo el usuario solo necesita elegir los sufijos de los tiempos correspondientes. Esta clase de búsqueda es útil para estudios lingüísticos del texto que no podrían ser llevados a cabo tan eficientemente si la compresión se realiza sin segmentación (ver la Sección 2.6.3 para una explicación detallada sobre cómo ejecutar diferentes tipos de búsquedas sobre texto comprimido).

Para los dos últimos tipos de búsqueda mencionados es necesario que los sufijos se hayan comprimido aplicando alguna técnica basada en palabras (como nuestra codificación o las codificaciones Huffman de Moura *et al.*) ya que es necesario acceder al sufijo directamente sobre el fichero comprimido.

Si el tipo de búsqueda a realizar va a ser siempre de tipo aproximado, podría optarse por aplicar otra técnica de compresión para los sufijos, incluso alguna basada en caracteres. Sin embargo, se han realizado diferentes pruebas de compresión empleando otros métodos y, en base a los resultados obtenidos, podemos concluir que la mejor compresión de los sufijos se alcanza aplicando una codificación basada en palabras y no una basada en caracteres.

5.7. Conclusiones

En este capítulo se ha realizado un estudio detallado, tanto a nivel teórico como empírico, de la problemática a la hora de comprimir textos con amplia variación gramatical, como es el caso de las lenguas romances. Los resultados obtenidos han permitido confirmar nuestras dos hipótesis de partida. La primera de ellas era la existencia de un importante crecimiento del tamaño del vocabulario respecto a los textos en inglés, lo cual ha sido confirmado con datos experimentales y teóricos (Ley de Heaps). La segunda hipótesis era que la distribución de las palabras en el texto también estaba más sesgada. Igual que en el caso anterior, se ha confirmado mediante resultados empíricos y teóricos (Ley de Zipf).

Una vez confirmadas ambas hipótesis, se ha presentado una alternativa para la compresión de textos con vocabularios grandes. Esta técnica se basa en preprocesar los documentos antes de llevar a cabo la compresión. Para ello, se ha definido un algoritmo de *segmentación* que separa las raíces de los sufijos de las palabras, obteniéndose así dos vocabularios diferenciados.

Las ventajas de la solución propuesta son las siguientes:

- Se obtienen mejores ratios de compresión que con la aproximación sin segmentar. Sobre los ficheros de raíces y sufijos se aplica la codificación Densa con Post-Etiquetado obteniendo buenos porcentajes gracias al incremento de las repeticiones de símbolos.
- Los tiempos de búsqueda son mejores. Las consultas son de tipo aproximado por lo que se realizan directamente sobre el fichero comprimido de raíces, que es considerablemente más pequeño al que se obtendría en caso de no hacer segmentación. Como resultado de una búsqueda se obtienen todas las variaciones morfológicas de la palabra o palabras solicitadas.

6

Conclusiones y trabajo futuro

6.1. Conclusiones

En el marco de este trabajo se ha desarrollado un nuevo esquema de compresión de especial interés en el campo de las bases de datos textuales. La nueva codificación, denominada *codificación Densa con Post-Etiquetado*, conserva muchas características de otros esquemas de compresión de textos similares existentes en la actualidad [26, 37, 45], en particular la codificación Tagged Huffman [36, 37], como son:

- Es un esquema de compresión basado en palabras; es decir, los símbolos a comprimir son palabras y no caracteres.
- Es una codificación de prefijo libre. Por lo tanto, se garantiza que ningún código es prefijo de otro.
- Soporta la descompresión de porciones arbitrarias de texto gracias a la incorporación de marcas que permiten distinguir los códigos dentro del texto comprimido.
- Es posible realizar búsquedas eficientes de palabras de un modo exacto,

aproximado o utilizando expresiones regulares, directamente sobre el texto comprimido.

Al mismo tiempo, la codificación Densa con Post-Etiquetado aporta importantes ventajas adicionales frente a la codificación Tagged Huffman:

- Se consiguen mejores ratios de compresión, dado que los códigos son alrededor de un 8 % más cortos que los generados por la codificación Tagged Huffman. Esto se consigue gracias a un mayor aprovechamiento de los posibles valores de los códigos de compresión.
- La codificación es mucho más sencilla y rápida, al no ser necesario construir el árbol Huffman. La sencillez a la hora de la codificación permite agilizar también los procesos de compresión y descompresión del texto. Lo importante es que, a pesar de estas destacables mejoras, mantiene las mismas posibilidades y eficiencia que la codificación Tagged Huffman a la hora de realizar búsquedas (exactas o aproximadas) directamente sobre el texto comprimido.

Un sistema de Recuperación de Textos basado en esta nueva técnica de compresión podría también beneficiarse de otras ventajas añadidas. Entre ellas destacan (*i*) basta con almacenar el vocabulario ordenado por frecuencia, sin ninguna información adicional, aspecto muy interesante sobre todo para colecciones pequeñas; (*ii*) no es necesario calcular los códigos Huffman, se puede codificar y decodificar en cualquier instante mediante un programa de unas pocas líneas.

Se han demostrado analítica y experimentalmente las ventajas de la codificación Densa con Post-Etiquetado en términos de tamaño final del fichero comprimido. Los estudios teóricos se han realizado considerando diferentes distribuciones de palabras en el texto, entre las que se incluye la propuesta por la *Ley de Zipf* [7, 31, 42]. Adicionalmente, derivado del análisis teórico se han establecido nuevas cotas (superior e inferior) para la longitud de los códigos cuando se emplea codificación Huffman no binaria. Estos límites son de distinta naturaleza a los empleados habitualmente, y pueden ser mejores para ciertas distribuciones. Esto es lo que sucede, como se ha demostrado, para las distribuciones que siguen la ley de Zipf.

Para el estudio empírico se han empleado *corpus* de textos escritos en inglés de la colección TREC-4 [16] y *corpus* de documentos escritos en lenguas romances (en concreto, en español, portugués y gallego) construidos *ad hoc*. Los resultados, coincidiendo con los presentados en otros trabajos como [26, 37, 45], demuestran que el espacio de almacenamiento se ve reducido cuando el tamaño del texto a comprimir es mayor de 10 MB.

Como resultado de los estudios experimentales anteriores, se han obtenido también una serie de conclusiones más detalladas asociadas al uso de la codificación Densa con Post-Etiquetado y las codificaciones Plain y Tagged Huffman cuando se aplican sobre textos escritos en lenguas romances. Así, los resultados obtenidos tras la compresión de los *corpus* han servido para poner de manifiesto que las técnicas basadas en palabras no resultan tan eficientes para la recuperación de textos romances como cuando se aplican sobre textos en inglés. El problema es originado por el importante incremento en el vocabulario para esas lenguas y el tipo de distribución que caracteriza estos textos.

Por otro lado, dado que las pruebas experimentales realizadas sobre *corpus* en lenguas romances han dado lugar a ratios de compresión poco eficientes, se ha desarrollado una adaptación del nuevo esquema de compresión para que produzca mejores resultados cuando se aplica sobre textos con amplia variación gramatical. La solución propuesta consiste en la utilización de una herramienta basada en ingeniería lingüística, denominada segmentador, para el preprocesamiento de los *corpus* antes de realizar la compresión, y así reducir el tamaño del vocabulario. De este modo, el fichero original a comprimir da lugar a dos archivos comprimidos: uno con las raíces y otro con los sufijos.

La ventaja principal de esta aproximación es que se consigue una reducción importante del tamaño del fichero de raíces. Concretamente, se alcanzan ratios de compresión entre un 5 % y un 10 % mejores respecto al fichero comprimido sin segmentar. Este hecho repercute favorablemente en la eficiencia de la recuperación debido a que, en la mayoría de los casos, las búsquedas se van a realizar sobre el fichero de raíces, mucho más pequeño (y con un vocabulario mucho más reducido) que el fichero sin segmentar.

6.2. Líneas de trabajo futuro

A raíz del trabajo presentado pueden considerarse dos líneas de trabajo abiertas, una originada tras el desarrollo de la nueva técnica de codificación y otra directamente relacionada con la compresión de textos romances.

Así, en primer lugar, se puede explorar la posibilidad de reducir la redundancia de la codificación Densa con Post-Etiquetado. Actualmente, a la hora de realizar la codificación no se tienen en cuenta las diferencias entre las frecuencias de las palabras a codificar; el código asignado a cada palabra depende únicamente de la posición que ésta ocupa en el vocabulario de entrada. Además, el número de valores utilizados como marca de fin de código está predeterminado a 128. Podría trabajarse hacia la generalización del modelo de compresión, de modo que la codificación se ajustase a la distribución de las frecuencias de las palabras de los *corpus* a comprimir, modificando el número de marcas para que la codificación resultante sea más óptima.

En segundo lugar, sería interesante también realizar pruebas de segmentación y compresión sobre *corpus* en lenguas romances más "depurados". Como ya se ha comentado durante los estudios experimentales, los *corpus* de los que disponemos actualmente han sido construidos *ad hoc*. Por este motivo, en ellos podemos encontrarnos errores tipográficos, palabras formadas por la concatenación de otras, separadores muy largos, etc., que han afectado a ciertos parámetros analizados, como es el caso del porcentaje de compresión. A pesar de ello, estos *corpus* nos han servido para confirmar las hipótesis de partida y presentar nuestro modelo de segmentación, ya que las palabras "erróneas" son poco frecuentes y, por lo tanto, poco representativas del texto.

Relacionada con esta línea de trabajo, es importante también mejorar las capacidades de búsqueda sobre los textos segmentados. Actualmente, las búsquedas aproximadas son más eficientes que cuando se realizan sobre un fichero comprimido sin segmentar, pero no sucede lo mismo en el caso de las búsquedas exactas o cuando se desea obtener todas las variaciones morfológicas de una palabra. Es necesario analizar en detalle el rendimiento alcanzado para los distintos tipos de recuperación sobre

nuevos corpus de textos y explorar la posibilidad de depurar la técnica de segmentación presentada aquí. Una vez realizado este análisis podría plantearse una variación de la codificación presentada en este trabajo, que mejore los resultados obtenidos para las lenguas romances y que no tenga una penalización excesiva en las búsquedas de palabras exactas.

Por último, los experimentos realizados han servido para demostrar que los textos en lenguaje natural que hemos procesado no se ajustan bien a la distribución de Zipf. De hecho, en ocasiones se han obtenido mejores ratios de compresión con un valor de θ menor. Hay diferentes modos de calcular el valor θ para una distribución y sería importante encontrar nuevas aproximaciones que permitan predecir la compresión obtenida de acuerdo al análisis teórico. Ahora mismo, un θ más grande no garantiza una compresión mejor como sucedería si la distribución real se ajustase perfectamente a la distribución de Zipf. En cualquier caso esto no altera el hecho de que la codificación Densa con Post-Etiquetado es estrictamente mejor que la codificación Tagged Huffman y que en la práctica se produce una mejora en el porcentaje de compresión en un 8 % respecto a la misma. Para realizar la predicción del porcentaje de compresión, podría optarse por la utilización de modelos de frecuencias de palabras más sofisticados como el propuesto por Mandelbrot [24].

Apéndice A

Fuentes principales del *corpus* de lenguas romances

A.1. CASTELLANO

A.1.1. TEC-TECNOLÓGICO: ciencias y tecnología

- Ministerio de Ciencia y Tecnología
  (www.mcyt.es)
- Manuales on-line de programas informáticos y dispositivos electrónicos
- *corpus* lingüístico de referencia de la lengua española en Argentina
  (www.lllf.uam.es/ fmarcos/informes/*corpus*/coarginl.html)
- *corpus* lingüístico de referencia de la lengua española en Chile
  (www.lllf.uam.es/ fmarcos/informes/*corpus*/cochile.html)

A.1.2. PER-PERIODÍSTICO: medios de comunicación (prensa, radio, televisión)

- Reportaje sobre el Sida: la agonía de África
  (bjcu.uca.edu.ni/biblioteca/)

- *corpus* lingüístico de referencia de la lengua española en Argentina (www.lllf.uam.es/ fmarcos/informes/*corpus*/coarginl.html)
- *corpus* lingüístico de referencia de la lengua española en Chile (www.lllf.uam.es/ fmarcos/informes/*corpus*/cochile.html)

A.1.3. LIT-LITERARIO: lengua y literatura

- Biblioteca Virtual Miguel de Cervantes (cervantesvirtual.com)
- Word by word multilingual library (www.wordtheque.com)
- Proyecto Gutemberg (promo.net/pg)
- Aaron Chio - Métodos de pensamiento (www.geocities.com/Pentagon/7889/)
- Estudios de Lingüística Española (ELiEs) (elies.rediris.es)
- *corpus* lingüístico de referencia de la lengua española en Argentina (www.lllf.uam.es/ fmarcos/informes/*corpus*/coarginl.html)
- *corpus* lingüístico de referencia de la lengua española en Chile (www.lllf.uam.es/ fmarcos/informes/*corpus*/cochile.html)
- Seminario de Lingüística Informática (webs.uvigo.es/sli/)
- Centro Telemático de Filosofía (www.blues.uab.es/filosofia/centrotelematico.html)
- Proyecto Filosofía en Español. Joaquín Abreu Orta (1782-1851) (www.filosofia.org)
- El sitio de Kryon en español (www.kryon.8m.com)

- Relatos religiosos
  (www.visi.com/ contra_m/castellano/Reino,
  portalespiritual.cyberxcel.com/varios/chopra/7leyes.htm,
  www.gnosisonline.org/download2/Iglesia%20Gnostica.pdf)

- Web para descarga de libros
  (www.elmistico.com.ar/descarga/index.htm)

A.1.4. JUR-JURÍDICO

- Ministerio de Interior
  (www.mir.es)

- Ministerio de Educación, Cultura y Deporte
  (www.mec.es)

- Serie Directrices Técnicas sobre agua
  (www.unicef.org/programme/wes/pubs/glines/Wat_s.pdf)

- Acuerdo Interconfederal para la negociación colectiva 2002
  (www.cnt.es)

- Izquierda Unida
  (www.izquierda-unida.es)

- Xunta de Galicia
  (www.xunta.es)

- Manual de Legislación Pesquera
  (www.eco-index.org/)

- Ingeniería Civil y Medio Ambiente
  (www.miliarium.com/)

- UGT Galicia
  (www.ugtgalicia.org)

- *corpus* lingüístico de referencia de la lengua española en Argentina
  (www.lllf.uam.es/ fmarcos/informes/*corpus*/coarginl.html)

- *corpus* lingüístico de referencia de la lengua española en Chile (www.lllf.uam.es/ fmarcos/informes/*corpus*/cochile.html)

A.1.5. ECO-ECONÓMICO: economía y sociedad

- Ministerio de Economía (www.mineco.com)
- Ministerio de Hacienda (www.minhac.com)
- Publicación Xurídica Xeral do ilustre Colexio de Avogados de Ourense (controversia.avogacia.org)
- Instituto superior de estudios védicos (www.vrindavan.org/isev/)
- Biblioteca Virtual Miguel de Cervantes (cervantesvirtual.com)
- *corpus* lingüístico de referencia de la lengua española en Argentina (www.lllf.uam.es/ fmarcos/informes/corpus/coarginl.html)
- *corpus* lingüístico de referencia de la lengua española en Chile (www.lllf.uam.es/ fmarcos/informes/corpus/cochile.html)
- UGT Galicia (www.ugtgalicia.org)

A.2. GALLEGO

A.2.1. TEC-TECNOLÓGICO: ciencias y tecnología

- Tesis Doctorales de la Universidade da Coruña
- CORGA - (*corpus* de Referencia do Galego Actual) (corpus.cirp.es)
- Manuales on-line de programas informáticos

A.2.2. PER-PERIODÍSTICO: medios de comunicación (prensa, radio, televisión)

- A Nosa Terra
  (www.anosaterra.com)

A.2.3. LIT-LITERARIO: lengua y literatura

- Biblioteca Virtual Galega
  (www.bvg.udc.es)
- Biblioteca Virtual de literatura universal en galego BiVir
  (www.bivir.com)
- Instituto da Lingua Galega
  (www.usc.es/ilgas)
- Word by word multilingual library
  (www.wordtheque.com)
- Páxina da CRTVG
  (www.crtvg.es)
- Dirección Xeral de Política Lingüística
  (edu.xunta.es/dxpl)
- Literatura Galega (web USC)
- Seminario de Lingüística Informática
  (webs.uvigo.es/sli/)
- Biblioteca virtual eonaviega
  (www.terra.es/personal/cvalledor/biblioteca_virtual_eonaviega.htm)

A.2.4. JUR-JURÍDICO

- Diario Oficial de Galicia (DOGA)
  (www.xunta.es/dog/dog.nsf)

- Dirección General de Política Lingüística
  (edu.xunta.es/dxpl)
- Normativas de la UDC
  (www.udc.es)

A.2.5. ECO-ECONÓMICO: economía y sociedad

- Urban Vigo
  (www.urban-vigo.com/)
- Bloque Nacionalista Galego
  (www.bng-galiza.org)
- Consellerías de Medio Ambiente, Educación y Emigración
  (www.xunta.es/conselle/)
- Instituto galego de Estatística
  (www.ige.xunta.es)
- Universidade da Coruña
  (www.udc.es)
- CORGA - (*corpus* de Referencia do Galego Actual)
  (corpus.cirp.es/)
- Manual legislativo de la enseñanza no universitaria gallega
  (www.xente.mundo-r.com/LexislacionEducativa/)
- Coordinadora de traballadores de Normalización da Lingua
  (http://www.ctnl.org/)
- Concello de Vigo
  (www.vieiros.com)
- Universidade de Vigo
  (www.uvigo.es)
- PSOE
  (www.psoe.es)

- UGT Galicia
  (www.ugtgalicia.org)
- Revista Galega do Ensino
  (www.xunta.es/conselle/ceoug/dxpl/rge/)

A.3. PORTUGUÉS

A.3.1. TEC-TECNOLÓGICO: ciencias y tecnología

- Núcleo Interinstitucional de Lingüística Computacional
  (www.nilc.icmsc.sc.usp.br)
- Manuales on-line de programas informáticos

A.3.2. PER-PERIODÍSTICO: medios de comunicación (prensa, radio, televisión)

- Gabinete de planeamento e de coordenação do combate a droga
  (www.terravista.pt/meco/1185/psptgpcd.html)

A.3.3. LIT-LITERARIO: lengua y literatura

- Projecto Vercial
  (www.ipn.pt/literatura/)
- Biblioteca On-Line de Ciencias da Comunicacao
  (bocc.ubi.pt/index2.html)
- Textos portugueses
  (www.wordtheque.com)
- Núcleo Interinstitucional de Lingüística Computacional
  (www.nilc.icmsc.sc.usp.br)
- Universidade Católica Portuguesa - Faculdade De Filosofia De Braga
  (www.ucp.pt/)

A.3.4. JUR-JURÍDICO

- Jornal Oficial de Madeira
  (www.gov-madeira.pt/joram)

A.3.5. ECO-ECONÓMICO: economía y sociedad

- Biblioteca On-Line de Ciencias da Comunicação
  (bocc.ubi.pt/index2.html)
- Corpora de lengua portuguesa
  (natura.di.uminho.pt/jjbin/corpora)
- Gabinete de planeamento e de coordenação do combate a droga
  (www.terravista.pt/meco/1185/psptgpcd.html)

Bibliografía

[1] N. Abramson. *Information Theory and Coding.* McGraw-Hill, New York, 1963.

[2] Amir and Benson. Efficient two-dimensional compressed matching. In *DCC: Data Compression Conference.* IEEE Computer Society TCC, 1992.

[3] A. Amir, G. Benson, and M. Farach. Let sleeping files lie: Pattern matching in Z-compressed files. *Journal of Computer and System Sciences*, 52(2):299–307, Apr. 1996.

[4] M. D. Araújo, G. Navarro, and N. Ziviani. Large text searching allowing errors. In R. Baeza-Yates, editor, *Proceedings of the 4th South American Workshop on String Processing*, pages 2–20, Valparaiso, Chile, 1997. Carleton University Press.

[5] R. A. Baeza-Yates and G. H. Gonnet. A new approach to text searching. In *Proceedings of the Twelfth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Information Access Methods, pages 168–175, 1989.

[6] R. A. Baeza-Yates and G.Ñavarro. Faster approximate string matching. *Algorithmica*, 23(2):127–158, 1999.

[7] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley Longman, May 1999.

[8] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression.* Prentice Hall, 1990.

[9] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, Oct. 1977. See also [20] and [38].

[10] M. Buro. On the maximum length of huffman codes. *Information Processing Letters*, 45(5):219–223, 1993.

[11] R. M. Capocelli, R. Giancarlo, and I. J. Taneja. Bounds on the redundancy of Huffman codes. *IEEE Transactions on Information Theory*, 32:854–857, 1986.

[12] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[13] R. De Prisco and A. De Santis. On lower bounds for the redundancy of optimal codes. *Designs, Codes, and Cryptography*, 15(1):29–45, Oct. 1998.

[14] W. B. Frakes and R. A. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1992.

[15] R. G. Gallager. Variations on a theme by Huffman. *IEEE Transactions on Information Theory*, 24(6):668–674, 1978.

[16] D. Harman. The fourth text retrieval conference, 1996.

[17] H. S. Heaps. *Information Retrieval: Computational and Theoretical Aspects*. Academic Press, New York, 1978.

[18] D. S. Hirschberg and D. A. Lelewer. Efficient decoding of prefix codes. Technical Report ICS-TR-89-09, University of California, Irvine, Department of Information and Computer Science, Mar. 1989.

[19] D. A. Huffman. A method for the construction of minimum-redundancy codes. In *Proc. Inst. Radio Eng.*, pages 1098–1101, Sept. 1952. Published as Proc. Inst. Radio Eng., volume 40, number 9.

[20] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6:323–350, 1977.

[21] D. A. Lelewer and D. S. Hirschberg. Data compression. *ACM Computing Surveys*, 19(3):261–296, Sept. 1987.

[22] M. J. L. López. Criptografía y seguridad en computadores. 3ª edición. Libro electrónico. http://wwwdi.ujaen.es/ mlucena/lcripto.html.

[23] U. Manber. A text compression scheme that allows fast searching directly in the compressed file. *Lecture Notes in Computer Science*, 807:113–124, 1994.

[24] B. Mandelbrot. An informational theory of the statistical structure of language. In *Proc. Symp. on Applications of Communication Theory*, pages 486–500, 1952.

[25] D. Manstetten. Tight bounds on the redundancy of Huffman codes. *IEEETIT: IEEE Transactions on Information Theory*, 38, 1992.

[26] A. Moffat. Word-based text compression. *Software - Practice and Experience*, 19(2):185–198, 1989.

[27] A. Moffat and J. Zobel. Compression and fast indexing for multi-gigabyte text databases. *Australian Computer Journal*, 26(1):1–9, 1994.

[28] Navarro and M. Raffinot. A general practical approach to pattern matching over Ziv-Lempel compressed text. *Lecture Notes in Computer Science*, 1645:14–??, 1999.

[29] Navarro, E. Silva de Moura, M. Neubert, N. Ziviani, and R. Baeza-Yates. Adding compression to block addressing inverted indexes. *Information Retrieval*, 3(1):49–77, 2000.

[30] Navarro and J. Tarhio. Boyer-Moore string matching over Ziv-Lempel compressed text. In R. Giancarlo and D. Sankoff, editors, *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, number 1848 in Lecture Notes in Computer Science, pages 166–180, Montréal, Canada, 2000. Springer-Verlag, Berlin.

[31] V. Poosala. Zipf's law.

[32] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980.

[33] S. Project. Página Web. http://snowball.sourceforge.net.

[34] D. Salomon. *Data Compression: The Complete Reference.* Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., second edition, 2000.

[35] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication.* University of Illinois Press, Urbana, Illinois, 1949.

[36] E. Silva de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Fast searching on compressed text allowing errors. In W. B. Croft, A. Moffat, C. J. van Rijsbergen, R. Wilkinson, and J. Zobel, editors, *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-98)*, pages 298–306, New York City, Aug. 24–28 1998. ACM Press.

[37] E. Silva de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Fast and flexible word searching on compressed text. *ACM Transactions on Information Systems*, 18(2):113–139, Apr. 2000.

[38] D. M. Sunday. A very fast substring search algorithm. *Communications of the ACM*, 33(8):132–142, Aug. 1990. See also [9] and [20].

[39] A. Turpin and A. Moffat. Fast file search using text compression. In *Proceedings of the 20th Australian Computer Science Conference*, pages 1–8, 1997.

[40] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Morgan Kaufmann Publishers, USA, 1999.

[41] S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, Oct. 1992.

[42] G. K. Zipf. *Human Behavior and the Principle of Least Effort.* Addison-Wesley (Reading MA), 1949.

[43] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[44] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

[45] N. Ziviani, E. Silva de Moura, G. Navarro, and R. Baeza-Yates. Compression: A key for next-generation text retrieval systems. *IEEE Computer*, 33(11):37–44, 2000.

Universidade da Coruña

TESIS DOCTORAL