UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DISEÑO Y REALIZACION DE ALGORITMOS DE COMPRESION DE DATOS

T E S I S

QUE PARA OBTENER EL TITULO DE
INGENIERO EN COMPUTACION
P R E S E N T A N :

DIANA RAMIREZ FIGUEROA
EDUARDO VIVANCO RODRIGUEZ

MEXICO, D. F. 1993

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Contenido.

Introducción

En estos últimos años, la compresión de información ha sido uno de los factores determinantes que impulsaron a las industrias de la computación y de las comunicaciones digitales, a innovar la mayor parte de sus productos. Estos productos, tanto de software como de hardware, han contribuido en la formación de una sólida plataforma para el desarrollo de sofisticados sistemas de comunicación y procesamiento digital.

Ha causa de esto, varios sistemas de comunicación se han vuelto lentos, e incluso, han llegado a ser obsoletos. Esto puede deberse: al gran volumen de información a transmitir; a que no se cuenta con el hardware de compresión en los equipos en que se opera; a la falta de capital para reemplazar a dicho equipo. Como única solución a estos problemas, se propone utilizar un software de compresión de datos que permita optimizar el canal de transmisión.

Supongamos que se tiene un archivo a transmitir por un canal de comunicaciones cuyo tamaño es de 64 Kbytes y se transmite a una velocidad de 1200 bits por segundo. El tiempo de transmisión es de 425 segundos. Al comprimir el archivo, el número de bytes a transmitir se reduce y, por lo tanto, disminuye el tiempo de transmisión. Por otro lado, eso implica también un ahorro de ancho de banda del canal de comunicaciones.

La gran demanda de equipos con mayor capacidad de almacenamiento y el elevado costo de estos, ha generado que los usuarios de computadoras inicien la búsqueda de herramientas que les permitan optimizar sus actuales recursos de almacenamiento digital.

Por todo lo anterior hemos resuelto que el propósito de esta tesis es desarrollar un algoritmo universal de compresión de datos, para solucionar los problemas ya mencionados. También se ofrece a los lectores relacionados con la programación de sistemas, la oportunidad de desarrollar su propio software de compresión de datos, para proteger sus programas o para aprovechar mejor los recursos de almacenamiento de sus equipos de cómputo.

Para desarrollar nuestro algoritmo de compresión universal de datos, se analizaron varios algoritmos existentes. Sin embargo, la mayoría de ellos no pueden satisfacer todas las necesidades de los usuarios de PC's. Por ejemplo, algunos son muy buenos para compresiones de textos, pero no se aplican para archivos binarios o archivos generales.

A continuación se describe, en forma breve, el contenido de cada uno de los capítulos que integran esta tesis.

En el **Capítulo 1** se explica la teoría necesaria para entender adecuadamente la compresión de datos, así como los conceptos que ayudan a evaluar el algoritmo desarrollado en esta tesis.

El **Capítulo 2** da a conocer y analiza algunos ejemplos de algoritmos de compresión de datos ya existentes, con el fin de introducirnos al diseño y desarrollo de estos. Dentro de estos algoritmos o esquemas se encuentra el Código de Huffman, reconocido a nivel mundial, y algunos otros basados en este mismo esquema.

En el **Capítulo 3** se describe el desarrollo y realización (implementación) del algoritmo de compresión universal de datos de esta tesis. También se explica y analiza los algoritmos que sirvieron de base para su desarrollo.

El **Capítulo 4** proporciona las bases necesarias para demostrar que el desempeño del algoritmo desarrollado es eficiente. Estas bases se fundamentan en una serie de experimentos, que a su vez hacen uso de modelos probabilísticos bien conocidos. La comparación con otros algoritmos de compresión, como son : los comerciales, los institucionales, etc., es otra de las herramientas que utiliza este capítulo para argumentar lo anterior.

Por último, el **Capítulo 5** señala, en forma por demás breve, las posibles áreas en que encuentra aplicación nuestro algoritmo.

CAPITULO 1

Introducción a la Teoría de la Información.

El objetivo principal del presente capítulo es exponer las bases teóricas en las que se basa la compresión de datos. Definiciones fundamentales, tales como: información, medición de información, entropía, códigos y fuentes de datos, dan esencia y significado a este tema.

1.1 Antecedentes

Antes de iniciar cualquier clase de análisis acerca de la compresión de datos, es necesario conocer los fundamentos de la teoría de la información.

La teoría de información es una disciplina que utiliza métodos matemáticos formales para el estudio de la colección y manipulación de la información. Esta disciplina proporciona las bases teóricas para: la observación, la medición, la compresión y el almacenamiento de datos, así como: comunicaciones, estimación, toma de decisiones y reconocimiento de patrones, entre otras [2].

La teoría de información tuvo sus orígenes en los años 20's, en los cuales famosos teóricos e investigadores en el área de las comunicaciones (Nyquist y Hartley) y la estadística (Fisher) trataron de dar una definición de lo que es información. Sin embargo, sus esfuerzos no fueron tomados en cuenta sino hasta que en 1948 a través de Claude Shannon, quien desarrolló en su mayor parte la

teoría de información y publicó un artículo titulado "A Mathematical Theory of Communication". En él dió a conocer la teoría básica sobre la comunicación a través de canales ruidosos y mostró una extraordinaria perceptividad, que se basa principalmente en razonamientos de análisis heurísticos. Antes de la publicación de dicho artículo, se creía que las comunicaciones no estaban libres de error en presencia de ruido, es decir, que el incremento de ruido limitaba el flujo de información a través de un canal, aumentando la probabilidad de error (Pe). Sin embargo, era posible mejorar la exactitud en las señales digitales disminuyendo la velocidad de transmisión. Shannon demostró que esto era falso, ya que, para un canal dado, mientras la tasa de información a transmitir se mantenga dentro de un límite (Capacidad del Canal), es posible realizar una comunicación libre de errores. La esencia del artículo de Shannon consiste en que, la presencia de alguna perturbación aleatoria en un canal no establece por sí misma ningún límite sobre la exactitud de la transmisión.

El método que se considera para obtener una pequeña probabilidad de error en la transmisión digital, consiste en codificar la información transmitida en bloques. En general, grandes bloques son necesarios para obtener un buen desempeño.

Después de la publicación del artículo de Shannon, los investigadores vincularon sus estudios a la interrogante de ¿ Cómo transmitir datos a velocidades por encima de la Capacidad del Canal ?. Shannon demostró la imposibilidad de obtener una comunicación libre de errores a velocidades por encima de la Capacidad del Canal. El concepto de compresión de datos fué el principal fundamento en esta declaración. *La Compresión de Datos* se define como el estudio de la reducción de información , es decir, quitar redundancia o cometer distorsión deliberadamente [1].

Las teorías de Shannon dieron origen a un esquema que generalizo la representación de los sistemas de comunicación (figura 1.1)

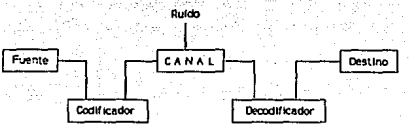


Figura 1.1 Sistema de comunicación.

La salida de la fuente puede ser: una sucesión de dígitos binarios proveniente de algún medio de almacenamiento magnético, un blanco en un sistema de radar o la salida de un conjunto de sensores en pruebas espaciales. El canal puede ser, por ejemplo: una línea telefónica, un enlace de microondas ó un enlace satelital. El codificador representa cualquier tipo de procesamiento de la señal de salida de la fuente. Tal procesamiento puede ser una combinación de: modulación, compresión de datos y la inserción de redundancia para combatir el ruido de canal. El decodificador tiene como objetivo obtener, para el destino, una réplica, lo más confiable posible, de la salida de la fuente a partir de la salida del canal [1].

El diseño del sistema codificador-decodificador se lleva a cabo mediante un modelo matemático, el cual depende en su mayor parte del tipo de información que contenga la fuente. Casi todos los modelos matemáticos de las fuentes, en la teoría de la información, se modelan por medio de procesos aleatorios. Las fuentes discretas sin memoria constituyen la clase más simple de los modelos de fuente. La salida de estas fuentes es una sucesión de símbolos, cuya selección, a partir de algún alfabeto fijo, es completamente aleatoria.

1.2 Medidas de la Información

Hay tres funciones matemáticas que se usan para medir la información. Estas son: la entropía, la información mutua y la discriminación, de las cuales solo se habla en el presente documento de la entropía, por ser de interés en lo que sigue.

El concepto de *información* es básico para el desarrollo de este capítulo.

Sea E un evento que puede presentarse con probabilidad P(E). Cuando ocurre E, se dice que se han recibido

$$I(E) = \log \frac{1}{P(E)} \qquad (1.1)$$

unidades de información.

Lo anterior se puede interpretar como la cantidad de información necesaria para representar la ocurrencia de E.

La elección de la base del logaritmo que interviene en la definición determina la unidad. Para los fines de esta tesis se elige la base 2, cuya unidad es el bit.

Suponiendo que una fuente de información emite una sucesión de símbolos que pertenecen a un alfabeto finito y fijo, $S = \{ s_1, s_2, \ldots, s_q \}$. Los símbolos emitidos sucesivamente se eligen de acuerdo a una determinada distribución de probabilidad. Una fuente de información que genera símbolos que son estadísticamente independientes, se conoce como *fuente sin memoria* y se describe completamente mediante el alfabeto fuente S y las probabilidades con que esos símbolos ocurren, esto es:

$$P(s_1),\ P(s_2),\ \ldots,\ P(s_q).$$

La información media que suministra una fuente de información sin memoria, se calcula de la siguiente forma: la presencia de un símbolo s_i corresponde a una cantidad de información igual a

$$I(s_i) = -\log P(s_i) \text{ bits.}$$

La probabilidad de que aparezca s_i es precisamente $P(s_i)$, de modo que la cantidad *promedio* de información por símbolo de la fuente, está dada por el promedio

$$\sum_i P(s_i)\, I(s_i) \quad bits \qquad (1.2)$$

donde el símbolo de *sumatoria* indica la suma extendida a los q símbolos posibles de la fuente S. La cantidad promedio de información por símbolo de la fuente recibe el nombre de *entropía* H(S) de la fuente y está dada por la siguiente ecuación:

$$H(S) = -\sum_i P(s_i) \log P(s_i) \quad bits/símbolo \qquad (1.3)$$

El valor de esta expresión es menor o igual al log q [3].

Una propiedad importante de la entropía de una fuente de información, como la descrita, es que su valor máximo se alcanza cuando todos sus símbolos son equiprobables.

Un ejemplo particularmente importante de una fuente de información sin memoria, corresponde a una fuente binaria, es decir, q=2. En tal fuente, el alfabeto se reduce a dos símbolos {0,1}. La probabilidad de "0" está dada por *p* y la de "1" por *1-p*. La entropía de dicha fuente está descrita por la siguiente ecuación:

$$H_b(p) = -\,(\, p \log_2 p + (1-p) \log_2 (1-p)\,),$$

la cual se conoce como *función entropía binaria* [2], y es función sólo del parámetro *p*. El comportamiento de la función entropía binaria se muestra en la siguiente figura.

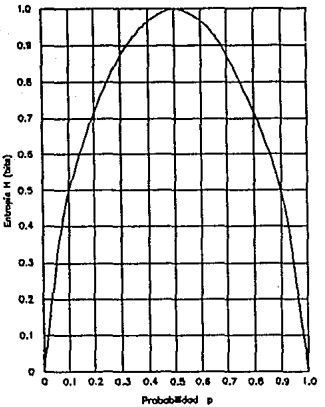


Figura 1.2 Función entropía binaria

Esta figura muestra que, en promedio, la cantidad de información proporcionada por una fuente binaria es siempre igual o menor que 1 bit (un bit de información por símbolo de la fuente). La información aportada por cada símbolo de la fuente binaria, alcanza su máximo valor solamente cuando los dos símbolos son equiprobables.

Otro punto a observar de la función entropía, es que la cantidad máxima de información dada por una fuente sin memoria de q símbolos, crece lentamente al aumentar q. De hecho, la cantidad máxima de información crece con el logaritmo del número de símbolos de la fuente, de modo que para duplicar la cantidad máxima de información por símbolo en una fuente de q símbolos, es necesaria una fuente de q^2 símbolos.

Lo anterior da la pauta para iniciar el tema referente al manejo de grupos de símbolos o sucesiones. Por ejemplo, una fuente binaria en la cual los bits son

emitidos en parejas, es equivalente a una fuente de 4 símbolos, 00, 01, 10, 11.

En general, si se tiene una fuente de información sin memoria, S, con un alfabeto $\{ s_1, s_2, ..., s_q \}$, las salidas se pueden agrupar en bloques de n símbolos, obteniendo q^n sucesiones distintas de salida. La nueva fuente de q^n símbolos generada por estas agrupaciones, se conoce como la **extensión de una fuente de información sin memoria**. Formalizando el concepto, se tiene lo siguiente [3]:

> Sea S una fuente de información sin memoria, con un alfabeto $\{ s_1, s_2, ..., s_q \}$. Sea P_i la probabilidad correspondiente a s_i. La extensión de orden n de S, S^n, es una fuente sin memoria de q^n símbolos, $\{ \mu_1, \mu_2, ..., \mu_{q^n} \}$. El símbolo $P(\mu_i)$, es precisamente la probabilidad de la sucesión correspondiente. Es decir, si μ_i representa la sucesión $(s_{i_1}, s_{i_2}, ..., s_{i_n})$, $P(\mu_i) = p_{i_1} \cdot p_{i_2} ... p_{i_n}$.

Puesto que un símbolo de la extensión de orden n, S^n, de la fuente sin memoria S corresponde a n símbolos de S, la entropía por símbolo de S^n es n veces mayor que la de S. Esto es,

$$H(S^n) = nH(S).$$

De todo lo anterior se concluye que la entropía es una característica inherente a las fuentes de información y el valor de ésta depende del tipo de información que generan dichas fuentes.

1.3 Propiedades de los Códigos

Sea $S = \{ s_1, s_2, ..., s_q \}$ el conjunto de símbolos de un alfabeto dado. Se define un código como la correspondencia de todas las sucesiones de símbolos de S a sucesiones de símbolos de algún otro alfabeto $X = \{ x_1, x_2, ..., x_r \}$. S recibe el nombre de alfabeto fuente y X de alfabeto código [1].

La primera propiedad exigida es que el código constituya un *código bloque*.

Un **código bloque** es aquel que asigna cada uno de los símbolos del alfabeto fuente S a una sucesión fija de símbolos del alfabeto código X. Esas sucesiones fijas (sucesiones de x_i) reciben el nombre de **palabras código**. Se denomina X_i a la palabra código que corresponde al símbolo s_i. X_i constituye una sucesión de $x'_i s$.

La tabla 1.1 muestra un ejemplo de código bloque binario.

Símbolo de la Fuente	Código
s_1	1
s_2	00
s_3	11
s_4	00

Tabla 1.1 Código bloque binario.

Es evidente que no basta hacer una correspondencia uno a uno entre el alfabeto fuente y el alfabeto código, por lo que hay que imponer ciertas restricciones al usar códigos de bloque. Una restricción es que todas las palabras código X_i sean distintas. A los códigos de bloque que cumplan con esta restricción, se les denominan **códigos no singulares.**

La siguiente tabla muestra un ejemplo de código bloque no singular.

Símbolo de la Fuente	Código
s_1	0
s_2	00
s_3	01
s_4	11

Tabla 1.2 Código bloque no singular.

A pesar de que todas las palabras código son diferentes, es posible encontrar algún caso en que una sucesión dada puede tener un origen indefinido. Por ejemplo, la sucesión 0011 puede corresponder a s_2s_4 ó $s_1s_1s_4$. Esto hace necesario imponer una nueva condición al hacer uso de estos códigos; dicha condición es que el código sea **unívocamente decodificable**.

> Un código bloque se dice *unívocamente decodificable* si y solamente si, su extensión de orden n es *no singular* para cualquier valor finito de n.

En otras palabras, un código es unívocamente decodificable si, y solamente si, para cada una de las sucesiones de salida de la fuente de longitud finita, la correspondiente sucesión de palabras código es diferente a la de otra sucesión de salida de la fuente de la misma longitud.

Una característica de este tipo de códigos, es que tiene la capacidad de reconocer cuando una palabra código, inmersa en una sucesión finita de símbolos, llega a su final. Un ejemplo de este código es el siguiente:

Símbolo de la Fuente	Código
s_1	0
s_2	01
s_3	011
s_4	0111

Tabla 1.3 Código unívocamente decodificable.

Este ejemplo da pie para describir otra propiedad que da origen a una nueva clase de código, **el código instantáneo**. Esto se desprende del hecho de que si se recibe una sucesión binaria del código anterior, no es posible decodificarla conforme se va recibiendo, ya que al recibir 01, por ejemplo, no se puede asegurar que es el símbolo s_2, en tanto no se haya recibido el símbolo siguiente. Este tipo

de análisis provoca un retraso en el proceso de decodificación, el cual es inherente a los códigos no instantáneos.

> Un código unívocamente decodificable se denomina *instantáneo*, cuando es posible decodificar las palabras de una sucesión sin precisar el conocimiento de los símbolos que siguen.

La condición necesaria y suficiente para que un código sea instantáneo, es que ninguna palabra de código coincida con el prefijo de otra, donde el prefijo se define de la siguiente manera:

> Sea $X_i = x_{i1}, x_{i2}, \ldots, x_{im}$ una palabra de un código. Se denomina *prefijo* de esta palabra a la sucesión de símbolos $(x_{i1}, x_{i2}, \ldots, x_{ij})$, donde $j \leq m$.

Resumiendo las distintas clases de códigos tratadas hasta el momento, se tiene la siguiente figura, en la cual se muestra la ramificación seguida en el árbol de subclases de códigos para llegar finalmente a los códigos instantáneos.

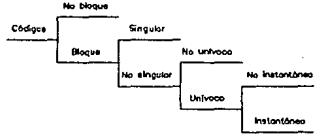


Figura 1.3 Subclases de códigos.

Por otra parte, resulta interesante obtener el código instantáneo cuyas longitudes de palabra de código sean lo más cortas posible. El siguiente teorema proporciona una condición necesaria y suficiente, que deben satisfacer las longitudes de las palabras de un código instantáneo [3].

Teorema Un código instantáneo con K símbolos en el alfabeto código y con q palabras código de longitudes $l_1, l_2, \ldots, l_q$, debe satisfacer la *desigualdad de Kraft*

$$\sum_{i=0}^{q} K^{-l_i} \leq 1 \qquad (1.4)$$

Hay que resaltar que esta desigualdad no garantiza que un código sea instantáneo. La desigualdad condiciona las longitudes de las palabras y no las palabras mismas. Por ejemplo, considérese el siguiente código.

Símbolo de la Fuente	Código
s_1	0
s_2	100
s_3	110
s_4	11

Tabla 1.4 Código no instantáneo.

Desarrollando la expresión (1.4), se tiene

$$\sum_{i=1}^{4} 2^{-l_i} = 2^{-1} + 2^{-3} + 2^{-3} + 2^{-2}$$
$$= 1$$

En este caso se observa que las longitudes del código satisfacen la desigualdad y puede constituir un código instantáneo. Sin embargo, como puede apreciarse, la cuarta palabra es un prefijo de la tercera, por lo que dicho código no es instantáneo. Puesto que los códigos instantáneos son un subconjunto de los unívocos, la desigualdad de Kraft se aplica a ellos, es decir, si las longitudes $l_1, l_2, \ldots, l_q$ satisfacen esta desigualdad, puede construirse con ellas un código unívocamente decodificable.

1.4 Codificación de Fuentes de Información

Una vez definida la entropía y los tipos de códigos que existen, se establece la relación entre ambos.

Al hablar de la codificación de una fuente de información discreta, se esta hablando de la conversión de la información de dicha fuente a una sucesión de símbolos de un alfabeto dado (alfabeto código). Es decir, se hace un mapeo del conjunto de símbolos de la fuente a otro conjunto de símbolos previamente seleccionado, conservando una relación unívoca (uno a uno). Las reglas para la codificación (compresión), se deben seleccionar de tal manera que la fuente original se recupere a partir de esta sucesión codificada, procurando que el número de símbolos de ésta, sea el menor posible para representar a la fuente original.

Para construir un código instantáneo que asocie los símbolos de éste con los de una fuente, se debe tomar en cuenta la longitud media del código, la cual se define como:

Sea un código de bloque que asocia los símbolos de una fuente $s_1, s_2, \ldots, s_q$ con las palabras $X_1, X_2, \ldots, X_q$. Suponiendo que las probabilidades de los símbolos de la fuente son $P_1, P_2, \ldots, P_q$ y las longitudes de las palabras $l_1, l_2, \ldots, l_q$. La longitud media del código, L, se define por la ecuación

$$L = \sum_{i=1}^{q} P_i l_i \qquad (1.5)$$

El problema fundamental de la codificación de fuentes de información, es el de la búsqueda de códigos de longitud media mínima (códigos compactos).

Anteriormente se comprobó que la entropía proporciona el número mínimo de símbolos, en promedio, de un alfabeto código, para representar a un símbolo de

un alfabeto fuente. Con base en esto, resulta interesante encontrar una relación que involucre a la entropía de una fuente de información, con la longitud promedio de las palabras código (L). El siguiente teorema proporciona dicha relación.

Teorema Para todo código instantáneo, con K símbolos, la longitud promedio de las palabras código L por símbolo de la fuente S satisface la desigualdad

$$H(S) \leq L \log K \tag{1.6}$$

Puesto que el valor de la entropía puede ser un número racional, es necesario aproximar el valor de la longitud de las palabras código a un número entero. Este entero es el inmediato superior del valor de la entropía. Por lo tanto, se hará a L igual al número entero que satisfaga la siguiente relación.

$$H_k(S) \leq L < H_k(S) + 1 \tag{1.7}$$

Ahora, aplicando ésta relación a una fuente de información sin memoria con extensión de orden n, se obtiene que

$$H_k(S^n) \leq L_n < H_k(S^n) + 1 \tag{1.8}$$

donde L_n representa la longitud media de las palabras correspondientes a los símbolos de la extensión de orden n de la fuente S. Por lo tanto L_n/n es el número medio de símbolos empleados en cada símbolo de S. La ecuación anterior puede escribirse de la forma

$$H_k(S) \leq \frac{L_n}{n} < H_k(S) + \frac{1}{n} \tag{1.9}$$

de modo que siempre es posible encontrar un valor de L_n/n tan cercano a $H_r(S)$ como se desee.

Esta ecuación se conoce como **teorema de Shannon para la codificación de fuentes de información** [1]. En éste cual se establece que el número de símbolos k-arios correspondientes a un símbolo de la fuente, puede hacerse tan pequeño pero no menor a la entropía de la fuente, expresada en unidades k-arias. Esto se logra al aumentar n y la siguiente expresión lo ilustra.

$$\lim_{n \to \infty} \frac{L_n}{n} = H_k(S) \tag{1.10}$$

Para evaluar la eficiencia de un código unívoco ó instantáneo, se cuenta con dos patrones de evaluación: **el rendimiento y la redundancia de un código.**

El rendimiento se define por la ecuación

$$\eta = \frac{H_k(S)}{L} \tag{1.11}$$

y la redundancia por

$$\begin{aligned} Redundancia &= 1 - \eta \\ &= \frac{L - H_k(S)}{L} \end{aligned} \tag{1.12}$$

El rendimiento es mejor mientras su valor esté más cercano a 1. Consecuentemente la redundancia es menor.

CAPITULO 2

Esquemas de Compresión de Datos

El estudio de algoritmos de compresión de datos es una de las actividades importantes en el área de la computación, especialmente en la transmisión y almacenamiento de información. Varios esquemas se utilizan para la compresión de datos y estos esquemas dependen del tipo de información que se maneje.

En este documento se clasifica a tales esquemas en dos clases: esquemas universales y esquemas particulares. Un esquema de compresión de datos se dice que es universal cuando el diseño del codificador y del decodificador se realiza sin conocer la fuente de información. Por el contrario, el esquema particular necesita conocer la fuente de información para el diseño de estos y utiliza ese conocimiento para mayor eficiencia en la compresión.

En el presente capítulo se analizan algunos esquemas universales de compresión de datos. El análisis incluye la metodología para construir un código instantáneo, así como la teoría básica necesaria para entender los diferentes esquemas de compresión.

2.1 Antecedentes.

Antes de iniciar el análisis de estos algoritmos, se mencionan algunas técnicas para comprimir fuentes de información.

Una de estas técnicas se aplica en aquellos documentos en los cuales se manejan dos tipos de información : fija y variable. Dentro de estos documentos se tiene, por mencionar algunos, contratos de arrendamiento, solicitudes de trabajo, facturas, ordenes de pago, etc. A continuación, se presenta un ejemplo para indicar cual es la información fija y cual información variable de una solicitud de trabajo:

NOMBRE : Carlos Vivanco Zavala

Carlos Vivanco Zavala constituye la información variable y NOMBRE la información fija. A continuación se manejará el término **formato** para hacer referencia a cualquier tipo de documento que cumpla con las características ya mencionadas.

La técnica de compresión consiste en borrar la información fija de un *formato*, transmitiendo solamente la información variable junto con un identificador que señale el tipo de formato que se usó. Los tipos de formato a usar se deben identificar previamente tanto en el receptor como en el transmisor. Una vez detectada la información, el receptor reconstruirá el formato original.

Otra técnica es la de codificar caracteres redundantes, esto es, una sucesión larga de 'x' caracter se puede codificar mediante unos caracteres de control. Por ejemplo, si se requiere enviar 5 caracteres ASCII iguales, la transmisión se puede sustituir por un carácter de control que indique el inicio de una sucesión comprimida, en seguida el ASCII 5 que indica el número de repetición y, por último el propio caracter ASCII. Tres caracteres (control, cantidad, caracter) representan a los originales cinco.

Uno de los métodos más prácticos para la compresión de datos es el de la representación binaria de los caracteres ASCII. Si hay transmisiones que comprenden varios números de grandes magnitudes, es mucho más eficiente transmitir estos en forma binaria, ya que, el número de bits a transmitir se reduce considerablemente. Por ejemplo, para una sucesión de 16 bits, se puede representar un valor numérico de hasta 65,535; así en vez de enviar 8 bits por

cada caracter numérico decimal (40 bits en total-5x8) se envian solamente 16 bits. Esta técnica se utiliza especialmente en sistemas cuya información, en su mayor parte, son números.

Hay otro tipo de técnicas para la compresión de datos que se basan en algoritmos matemáticos, diseñados para trabajar con base en la probabilidad de ocurrencia de una determinada sucesión de bits. De esta manera se puede codificar a dicha sucesión en otra sucesión más corta. En otras palabras, las sucesiones de bits de longitud fija que se utilizan se pueden representar por otra sucesión de bits más corta. Las menos utilizadas, una sucesión en particular, se representan con una sucesión más grande de bits, pero la cantidad total de bits transmitidos se reduce considerablemente.

Una de las técnicas que asignan sucesisones más cortas de bits a los símbolos que más frecuentemente ocurren, y sucesiones grandes de bits a los menos frecuentes se conoce como *Código de Huffman*, el cual se analiza a continuación.

2.2 Código de Huffman

Esta técnica es similar a la usada por Samuel Morse en el diseño del llamado código Morse, en el cual los caracteres más comúnmente utilizados se representan con menos cantidad de *puntos y rayas* (símbolos del alfabeto), y los menos utilizados con mayor cantidad de puntos y rayas. Esta idea genera mensajes codificados cuyas longitudes, en promedio, sean más cortos.

Este código, inventado por D.A.Huffman (1952), tiene como objetivo principal encontrar un conjunto óptimo de palabras código para representar un conjunto dado de mensajes (sucesiones de símbolos). Por óptimo se entiende a que no exista otro código únicamente decodificable cuya longitud promedio de sus palabras código sea menor al de éste.

El código compacto (código óptimo) de una fuente S es el de menor longitud media que se obtiene al codificar los símbolos de la fuente de *uno en uno*. A continuación se describe el proceso utilizado por Huffman para generar un código compacto en el caso de un alfabeto binario.

Considérese una fuente S, de símbolos $s_1, s_2, \ldots, s_q$ con probabilidades $P_1, P_2, \ldots, P_q$. Un ordenamiento de los símbolos por orden de probabilidad (de mayor a menor) constituye el primer paso de este método. El segundo paso consiste en efectuar reducciones sucesivas del conjunto de símbolos original. Lo anterior se realiza de la siguiente manera:

a) De la columna de probabilidades de la tabla 2.1, se toman las dos últimas probabilidades, a las cuales se les conoce como pareja generatriz, y se suman. La pareja generatriz se sustituye por el valor de la suma, obteniedose una nueva fuente denominada *fuente reducida*, la cual tendrá q-1 símbolos.

b) Aplicar el primer paso (ordenar símbolos) a la fuente reducida y continuar con el segundo paso, hasta obtener una fuente reducida con 2 símbolos.

Como puede observarse, cada fuente reducida tendrá un elemento menos que la anterior.

El tercer y último paso consiste simplemente en la asignación de código para cada una de las fuentes reducidas, empezando con la de menor número de elementos (de derecha a izquierda). Con esto se obtiene la tabla 2.2. El proceso de asignación de código se puede describir con los tres siguientes pasos:

a) Sea S_j la fuente reducida con menos elementos (en este caso 2), se le asignarán los símbolos 0 y 1 a cada uno de ellos (respectivamente).

b) Se localiza la pareja generatriz de la fuente S_{j-1} y se asignan los símbolos 0 y 1 a cada elemento de esta pareja (respectivamente), un prefijo se añade como parte del código, tal prefijo es el símbolo asignado al resultado de la suma de dicha pareja generatriz en la fuente S_j.

c) Se toma la siguiente fuente, la cual será S_{j-1}, con $j=j-1$, y se repiten los pasos b y c hasta llegar a la fuente original (la más a la izquierda).

Las siguientes tablas muestran, de una manera más clara, la construcción del Código de Huffman, para la fuente S_a.

En la tabla 2.1 aparece una fuente de cinco símbolos, junto con sus reducciones sucesivas.

Fuente Original		Fuentes Reducidas		
Símbolos	Probabilidades	S_1	S_2	S_3
s_1	0.4	0.4	0.4	0.6
s_2	0.2	0.2	0.4	0.4
s_3	0.2	0.2	0.2	
s_4	0.1	0.2		
s_5	0.1			

Tabla 2.1 Reducciones Sucesivas.

La tabla 2.2 representa el proceso seguido en la construcción del código compacto binario de la fuente S_o.

Fuente Original			Fuentes Reducidas		
Símbolos	Probabilidades	Código	S_1	S_2	S_3
s_1	0.4	00	0.4 00	0.4 1	0.6 0
s_2	0.2	01	0.2 01	0.4 00	0.4 1
s_3	0.2	11	0.2 10	0.2 01	
s_4	0.1	100	0.2 11		
s_5	0.1	101			

Tabla 2.2 Síntesis del Código de Huffman.

El cálculo de la entropía de dicha fuente y la longitud media de su código generado, se muestra a continuación.

La entropía de la fuente está dada por,

$$H(S) = \sum_{i=1}^{5} P(s_i) \log_2 \frac{1}{P(s_i)}$$

$$= 2.1219 \quad \textit{bits/símbolo}$$

y la longitud promedio de este código es,

$$L = \sum_{i=1}^{5} P(s_i)\, l_i$$

$$= 2.2 \quad \textit{dígitos binarios/símbolo}$$

Al comparar la longitud media del código con la entropía se observa que se puede lograr una mejora de cerca del 4%, en la construcción del código. La tasa de compresión puede aproximarse a la entropía tanto como se desee, aplicando el código de Huffman en bloques (sucesiones). Por ejemplo, sea una fuente con tres símbolos de alfabeto (A, B y C) con probabilidades 3/4, 3/16 y 1/16. La longitud promedio de la palabra código es de 1.25 bits por símbolo de la fuente, y su

entropía de 1.012 bits. Si se realizan combinaciones de estos tres símbolos y se codifica un bloque de longitud 2, la longitud media de la palabra código será de 2.09 bits, que es la longitud de dos símbolos de la fuente. Por lo tanto, la longitud media del código por símbolo de la fuente es de 1.045 bits, obteniéndose una mejor aproximación a la entropía. Como se vio en el capítulo anterior la longitud promedio de las palabras código tiende a la entropía mientras la longitud de la secuencia de salida de la fuente a codificar tienda al infinito.

Otro método de asignación de código a una fuente de información mediante el método de Huffman es el de tomar los dos elementos con menor probabilidad de cada fuente, asignar un 0 y un 1 a cada uno de ellos, obtener la fuente reducida y proceder con esto hasta obtener una fuente con dos elementos. El código se deduce fácilmente del árbol que se genera en tal proceso [2]. La figura 2.1 muestra el proceso de construcción del código.

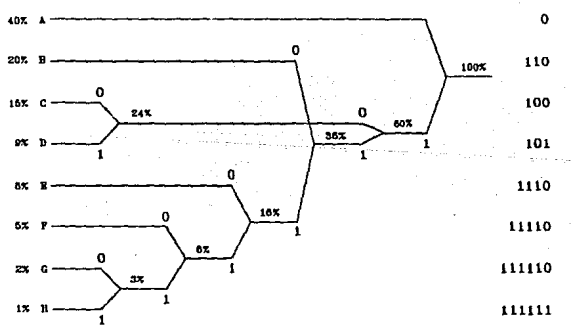


Figura 2.1 Proceso de construcción del código.

La implementación de este algoritmo resulta un tanto compleja, debido a que

previamente se deben conocer las probabilidades de ocurrencia de los símbolos de la fuente de información, además de que la construcción de su código se vuelve impráctica al codificar sucesiones largas de símbolos. Por lo que, a continuación se analizará un algoritmo de compresión de datos universal para fuentes binarias, cuya implementación resulta más práctica. Inclusive cumple con las reglas establecidas para los códigos de compresión, vistas en el capítulo anterior.

2.3 Algoritmo Universal de Compresión de Datos basado en la Teoría de Tiempos de Repetición.

En esta sección se describe tanto la base teórica como la implementación de este algoritmo desarrollado por Frans M.J. Willems (6), que tiene como base la teoría de tiempos de repetición.

Antes de describir la forma en la que operan tanto el codificador como el decodificador se menciona el método de tiempos de repetición.

Este método encuentra un valor M_t (tiempo de repetición) el cual señala si una sucesión X_t (sucesión aleatoria de la fuente), se ha repetido anteriormente. El valor de m fija el número máximo de comparaciones que se realizaron entre esta sucesión aleatoria X_t y el contenido del buffer del codificador para encontrar una igualdad.

El algoritmo, en general, funciona de la siguiente manera : Se tiene una fuente binaria que produce sucesiones de salida con valores en el alfabeto { 0,1 }, las cuales el codificador fragmenta en palabras fuente de longitud L. Posteriormente, el codificador usa un buffer para determinar el tiempo de repetición de la palabra fuente a ser transmitida. Este tiempo de repetición se transforma en una palabra código de longitud variable, la cual se transmite al decodificador. La palabra código consta de un prefijo y un sufijo, cuyos símbolos toman valores en el alfabeto { 0,1 }.

Conociendo este código, el decodificador podrá reconstruir el tiempo de repetición, con el cual podrá recobrar la palabra fuente de su buffer.

Formalizando lo anterior, se tiene que el tamaño del buffer, tanto del codificador como del decodificador, esta dado por $B = 2^L - 1$, la secuencia a codificar por $w_t = u(k) = (u_{t-L}, u_{t-L+1}, \ldots, u_{t-1})$ y, el contenido del buffer del codificador por $\Phi_e(k) = (u_{t-L-B}, u_{t-L-B+1}, \ldots, u_{t-L-1})$.

El codificador puede asignar a cada m_t (tiempo de repetición) un índice p tal que $m_t \in \mu_p$. El conjunto μ_p se define como:

$$\mu_p = \begin{cases} \{\, m: 2^p \le m \le 2^{p+1}-1 \,\}, & \text{para } p = 0, 1, \ldots, L-1, \\ \{\, m: m \ge 2^L \,\}, & \text{para } p = L\,. \end{cases}$$

El conjunto de índices p se transmite al codificador por medio de un prefijo de longitud fija de $\log(L+1)$ dígitos binarios. Este prefijo es la primera parte de la palabra código c(k). La construcción de la segunda parte de c(k), el sufijo, depende de los valores que tome p. Si $p < L$ entonces el sufijo se calcula como: $q = m_t - 2^p$, donde $q \in \{0, 1, \ldots, 2^p - 1\}$. Para mandar esta segunda parte de c(k), se necesita p dígitos binarios.

Cuando la palabra fuente w_t no se encuentre en el buffer $\Phi_e(k)$, p será igual a L y, el codificador mandará la palabra fuente de entrada u(k) al decodificador. Esto requiere un sufijo de L dígitos binarios.

El siguiente ejemplo sirve de base para aclarar lo anterior. Sea $L=3$ y $t=0$. Se asume que el buffer contiene

$$\Phi_e(0) = (u_{-10}, u_{-9}, \ldots u_{-4}) = (0,1,0,0,1,0,0)$$

y que

$$(u_{-3}, u_{-2}, \ldots) = 100|000|011|111|011|101|\ldots$$

es la futura sucesión de información a codificar. Los valores de tiempos de repetición m_t son respectivamente |3|1|8|1|6|4|8|. Se puede observar que cuando $(u_{t-L}, u_{t-L+1}, \ldots, u_{t-1}) = (u_{t-L-1}, u_{t-L}, \ldots, u_{t-2})$, es decir, se traslapa u(k) con el buffer, se obtiene un tiempo de repetición igual a uno.

Con ayuda de la tabla 2.3, se tienen los siguientes códigos:

|01, 1|00|11, 011|00|10, 10|10, 00|...

(el prefijo y el sufijo están separadas por una coma).

Tabla 2.3 Tabla de codificación para L = 3.

m	p	q	Prefijo	Sufijo	longitud
1	0	0	00	-	2
2	1	0	01	0	3
3	1	1	01	1	3
4	2	0	10	00	4
5	2	1	10	01	4
6	2	2	10	10	4
7	2	3	10	11	4
=8	3	-	11	u(k)	5

la tasa de compresión esta dada por la expresión.

$$R = H(U_0, U_1, \ldots, U_{L+1}) + \log (L+1)$$

Una ligera modificación al algoritmo podría proporcionar un mejor desempeño del mismo, es decir, se obtiene una mejor tasa de compresión. Dicha modificación consiste en incrementar la longitud de las secuencias de la fuente a codificar de $L = \mathcal{L}$ a $L = \mathcal{L} + \log(\mathcal{L})$, para un buffer de tamaño $B = 2^{\mathcal{L}} - 1$. El hacer uso de esta modificación implica redefinir al conjunto de μ_p como :

$$\mu_p = \begin{cases} \{ m: 2^p \le m \le 2^{p+1}-1 \}, & \text{para } p = 0, 1, \ldots, \mathcal{L}-1. \\ \{ m: m \ge 2^{\mathcal{L}} \}, & \text{para } p = \mathcal{L}, \end{cases}$$

lo que resta modificar es el manejo de los prefijos. Los prefijos tendrán ahora una longitud variable. Para distinguir entre el conjunto de $\mu_0, \mu_1, \ldots, \mu_{\mathcal{L}-1}$, el codificador manda un cero y posteriormente log(£) dígitos binarios, el sufijo se mantiene igual que antes. Para $\mu_{\mathcal{L}}$, el prefijo es uno y el sufijo es la palabra fuente u(k), la cual tiene una longitud de $L = \mathcal{L} + \log(\mathcal{L})$. La tabla correspondiente a $\mathcal{L} = 4$ es la siguiente.

Tabla 2.4 Tabla de codificación para $\mathcal{L} = 4$ ($L = 6$).

m	p	q	Prefijo	Sufijo	Longitud
1	0	0	0 0 0	-	3
2	1	0	0 0 1	0	4
3	1	1	0 0 1	1	4
4	2	0	0 1 0	0 0	5
5	2	1	0 1 0	0 1	5
6	2	2	0 1 0	1 0	5
7	2	3	0 1 0	1 1	5
8	3	0	0 1 1	0 0 0	6
9	3	1	0 1 1	0 0 1	6
...	...	...	...	...	...
15	3	7	0 1 1	1 1 1	6
= 16	4	-	1	u(k)	7

Los parámetros de este algoritmo para £, están dados por,

$$L = \mathcal{L} + \log(\mathcal{L})$$

$$B = 2^{\mathcal{L}} - 1$$

$$R \leq (H(U_0, U_1, \ldots, U_{L-1}) + \log(\mathcal{L}) + 1) / L$$

Para una implementación práctica se recomiendan valores de $\mathcal{L} \leq 24$.

CAPITULO 3

Desarrollo de un Algoritmo de Compresión Universal.

3.1 Introducción.

El propósito de este capítulo es presentar el desarrollo e implementación de un algoritmo de compresión universal de datos, lo cual es el tema de esta tesis. Por lo anterior y como antecedente, se mencionan los esquemas que sirvieron de base a dicho algoritmo, el esquema LZ77 y el LZSS.

El algoritmo LZ77, sugerido por Ziv y Lempel [5], se aplica solo para compresión de textos. Esta versión es modificada por Storer y Szymanski, obteniendose el esquema LZSS, el cual alcanza mejores tasas de compresión.

La decodificación en el esquema LZSS es muy rápida y no se requiere mucha memoria para la codificación y decodificación. Pero existe el inconveniente de que la codificación es muy lenta. Por esta razón Timothy C. Bell [5] propuso el algoritmo de árboles binarios, con el cual la velocidad de compresión aumentó considerablemente.

Experimentos realizados durante el desarrollo de esta tesis demuestran que al emplear un árbol binario, eventualmente provoca una degeneración en el mismo al codificar fuentes cuya salidas son secuencias largas. Tal degeneración acarrea un lento desempeño en el codificador.

Este problema se resuelve, utilizando los esquemas de árboles balanceados, lo cual es una de las aportaciones de este trabajo.

3.2 Trabajos Previos

Esquema LZ77

El esquema LZ77 se basa en el esquema OPM/L (original pointer macro restricted to left pointers), el cual consiste en reemplazar una subcadena que se ha repetido anteriormente en el texto, por un apuntador que representa la posición y el tamaño de dicha subcadena.

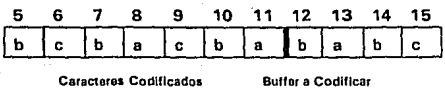
El esquema LZ77 restringe el alcance del apuntador a N caracteres previos, creando una ventana de N caracteres. Dicha ventana se usa como un diccionario deslizante. El uso de la ventana tiene las siguientes ventajas:

1. La cantidad de memoria requerida para codificar y decodificar se restringe al tamaño de la ventana.

2. La ventana es un buen diccionario para las subcadenas que siguen, ya que usualmente contiene el mismo lenguaje y estilo.

3. Todos los apuntadores pueden tener campos de tamaño fijo.

En cada paso de la codificación, una sección del texto de entrada está disponible en una ventana de N caracteres. De estos, los primeros N - F caracteres se han codificado y los F restantes caracteres forman el *buffer a codificar*, el cual almacena la siguiente cadena a codificar. Por ejemplo, si la sucesión S = abcabcbacbababcabc... se codifica con los parámetros N = 11 y F = 4, y el caracter 12 es el próximo a codificar, la ventana es la siguiente:



Inicialmente los primeros N-F caracteres de la ventana son (arbitrariamente) nulos, y los primeros F caracteres de el texto se cargan en el buffer a codificar.

En la parte codificada de la ventana se busca la subcadena más larga que sea igual al buffer de avance, esta subcadena se le denomina *la pareja del buffer*. La pareja se puede traslapar con el buffer, sin llegar a cubrir al mismo. En el ejemplo la pareja más larga para 'babc' es 'bab', la cual comienza en el caracter 10.

Una vez obtenida la pareja, se codifica en una tripleta (i, j, a), donde i es la distancia entre la pareja y el buffer, j es la longitud de la pareja, a es el primer caracter que no hace pareja con la subcadena de la ventana. Para este ejemplo la tripleta es (2, 3, 'c'). La ventana se recorre hacia la derecha j + 1 caracteres, y está lista para seguir codificando.

El número de bits que necesitan la distancia (i) y el número de caracteres (j) para representarse es $\lceil \log_2 (N-F) \rceil$, y $\lceil \log_2 F \rceil$, respectivamente.

La decodificación es muy simple y rápida. El decodificador mantiene una ventana de la misma forma que el codificador pero, en lugar de buscar una pareja en la ventana, este copia la pareja de la ventana usando la tripleta proporcionada por el codificador. La ventana se actualiza por cada pareja decodificada.

La principal desventaja del LZ77 es la lentitud con la que codifica los textos, ya que una implementación real puede requerir hasta (N-F) * F comparaciones de caracter por buffer a codificar.

Esquema LZSS

La salida del codificador del esquema LZ77 es una serie de tripletas (i,j,a). En el esquema LZSS, estas tripletas se representan por un apuntador (i,j), y el caracter a.

El caracter se usa solo cuando el apuntador ocupe más bits para su representación que el caracter a codificar. El siguiente pseudocódigo describe el algoritmo LZSS:

WHILE (buffer a codificar) not vacío **DO**
    OBTEN un apuntador (distancia, longitud) de la pareja
    más larga en la ventana para el buffer a codificar.
    IF (longitud > p) then
        sale el apuntador (distancia, longitud)
        recorre la ventana longitud caracteres.
    ELSE
        sale el primer caracter del buffer,
        recorre la ventana un caracter.

Un bit extra se agrega al apuntador o al caracter para distinguirlos en el momento de decodificar.

La implementación del LZ77 y LZSS se puede simplificar numerando módulo N, los caracteres del texto de entrada. La ventana será un arreglo con N caracteres. De esta forma se puede sustituir la longitud en el apuntador por la posición en el arreglo (0 .. N-1).

Al momento de codificar se observa que el primer elemento del apuntador puede representarse por $\lceil \log_2 N \rceil$ bits y el segundo elemento por $\lceil \log_2 (F-p) \rceil$ bits. Incluyendo el bit de bandera, un apuntador requiere para su representación un total de

$$1 + [\log_2 N] + [\log_2 (F-p)] \text{ bits.}$$

Para el caracter de salida se requiere de 8 bits: un bit de bandera y 7 bits para representar el caracter ASCII.

Al número de bits que necesitan los parámetros N y F para ser codificados se les denomina *n* y *f* respectivamente. Los parámetros de codificación se definen como:

$$p = (1 + n + f) / 8$$
$$N = 2^n$$
$$F = 2^f + p$$

A pesar de que se ha logrado una mejoría al momento de codificar, persiste el problema de la velocidad de codificación, ya que para N = 8k la codificación de un texto es del orden de 18 caracteres por segundo. A continuación se describe un esquema incorporado para solucionar este problema.

Algoritmo de árboles binarios.

En esta sección se explica el algoritmo propuesto por Timothy C. Bell, el cual se auxilia de árboles binarios para encontrar la pareja más larga dentro de la ventana W.

Se tiene un conjunto X U {/}, donde X es el conjunto de todas las cadenas de longitud F, de la ventana y / es la cadena a codificar. Estas cadenas se ordenan en forma lexicográfica. La cadena / será adyacente a la cadena x_a y a la cadena x_b en la lista ordenada, es decir, $x_a \leq / \leq x_b$. La pareja más larga para / en W se encuentra al principio de x_a ó x_b.

El siguiente ejemplo tiene por objetivo aclarar la forma en que el esquema LZSS utiliza los árboles binarios.

Si el conjunto X

x_5 = bcba, x_6 = cbac, x_7 = bacb, x_8 = acba,
x_9 = cbab, x_{10} = baba, x_0 = abab, x_1 = / = babc.

se ordena en forma lexicográfica, se obtiene la siguiente lista:

x_0	x_8	x_{10}	/	x_7	x_5	x_9	x_6
abab	acba	baba	babc	bacb	bcba	cbab	cbac

La pareja más larga / se puede encontrar al principio de la cadena x_{10} ó x_7. Si este conjunto X se inserta en un árbol binario vacío, se obtiene el siguiente árbol:

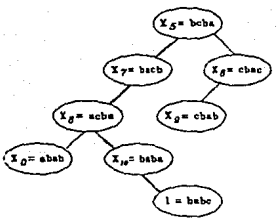


Figura 3.1 Arbol binario

Se observa que el orden simétrico en el árbol corresponde al orden lexicográfico de las subcadenas, esto es, para un nodo x_i, todos los nodos del subárbol izquierdo son lexicográficamente menores que x_i, y todos los nodos en el subárbol derecho son mayores que x_i. Los nodos del árbol, para este ejemplo, contienen las subcadenas del conjunto X. En la práctica, el nodo x_i solo necesita almacenar el índice i, ya que, x_i puede obtenerse de la ventana W.

También se observa que x_{10} y x_7 se encuentran en la trayectoria del nodo raiz al nodo *i*. De hecho, se puede demostrar fácilmente que tanto x_a como x_b serán los padres de *i* y posibles candidatos. El otro candidato para la pareja más larga se encuentra usando las siguientes reglas:

1. Si x_a es el padre de *i* entonces x_b es el nodo donde la trayectoria de inserción cambió a la izquierda, por última vez.

2. Si x_b es el padre de *i* entonces x_a es el nodo donde la trayectoria de inserción cambió a la derecha, por última vez.

Existe un caso especial, cuando ya existe un nodo $x_j = x_i$ en el árbol binario. En este caso se sustituye el nodo x_j por x_i y se escoge a x_j como la pareja más larga dentro del árbol para x_i.

Usando el esquema LZSS con este algoritmo de búsqueda basado en árboles binarios, el tiempo de codificación para cada caracter será el tiempo que tome la inserción y borrado de nodos en el árbol.

Arboles binarios balanceados.

A continuación se describe un método para mantener equilibrado un árbol binario. Los matemáticos Rusos, G.M. Adel'son-Vel'skii y E.M. Landis [7] inventaron este método en el año de 1962.

Primeramente, se presentarán los conceptos básicos utilizados en este algoritmo.

> **La altura del árbol** se define como la longitud más larga desde el nodo raíz hasta un nodo externo.
>
> **El factor de balance** es la diferencia de la altura del subárbol derecho de un nodo menos la altura de su subárbol izquierdo. Para que el árbol esté balanceado, los factores de cada nodo deben estar en el rango de [-1,1].

En la Figura 3.2 se muestra un árbol balanceado con 17 nodos internos y una altura de 5; el factor de balance dentro de cada nodo es mostrado como +, ', ó - de acuerdo a la diferencia obtenida entre la altura del subárbol derecho menos la altura del subárbol izquierdo, esto es, +1, 0, ó -1 respectivamente.

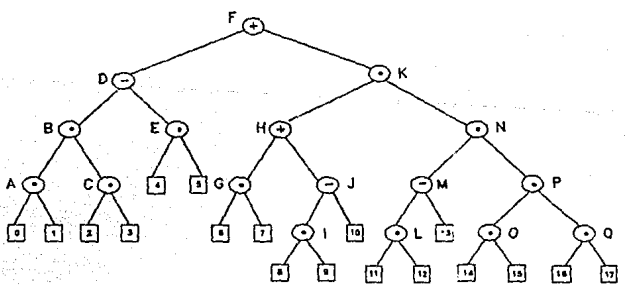


Figura 3.2 Arbol binário balanceado

Cabe señalar que la altura de un árbol balanceado es un factor determinante

para ser considerado óptimo. El siguiente teorema, propuesto por Adel'son-Vel'skii y Landis [7], expone cuando un árbol balanceado se considera óptimo.

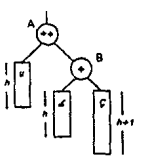
Teorema La altura de un árbol balanceado, *h*, con N nodos internos oscila entre $\log_2 (N + 1)$ y $1.4404 \log_2 (N + 2) - 0.328$.

De lo anterior se deduce que para insertar un nodo se requieren, para un árbol balanceado con N nodos, un máximo de *h* comparaciones.

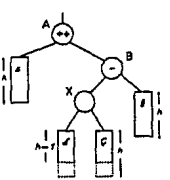
Por otra parte, considérese lo que pasa cuando un nuevo nodo se inserta en el árbol de la Fig. 3.2; solo se podrán insertar nodos en los lugares 4, 5, 6, 7, 10 ó 13, de otra forma el árbol se desbalanceará. Existen sólo dos casos en los cuales el árbol se desbalancea al insertar un nuevo nodo, estos son :

Caso 1 Ocurre cuando un nodo incrementa la altura del subárbol derecho de B, de h a h+1.

Caso 2 Ocurre cuando el nuevo nodo incrementa la altura del subárbol izquierdo de B.



Caso 1

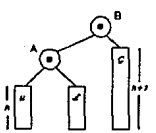


Caso 2

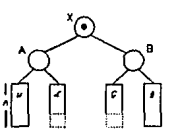
Los rectángulos de los diagramas representan subárboles con sus respectivas alturas.

Una transformación sencilla reestablece el balance en ambos casos, y se mantiene la simetría de los nodos del árbol; dichas transformaciones se muestran para ambos casos.

En el Caso 1, el árbol sólo necesita una rotación simple a la izquierda, insertando £ en A en vez de B. En el Caso 2 se usa una doble rotación, primero rotamos (X,B) a la derecha, y luego (A,X) a la izquierda. En ambos casos cambian en realidad pocas ligas del árbol. Los Factores de Balance se tienen que modificar en los dos casos.



Caso 1



Caso 2

En el procedimiento de borrado existen tres casos, los dos casos que se dan en el proceso de inserción y un nuevo **Caso 3** que se parece al Caso 1 excepto que £ tiene altura h + 1. En el Caso 3 se hace una rotación simple, y esto deja los factores de balance de A y B diferentes de cero sin cambiar la altura total.

La principal diferencia entre el borrado y la inserción es que el proceso de borrado puede requerir más de *log N* rotaciones, mientras la inserción nunca necesita más de una.

3.3 El Algoritmo Universal

A partir de los algoritmos de compresión de texto, LZ77 y LZSS, y de la implementación de los árboles binarios, se puede desarrollar un algoritmo de compresión universal. El método de la ventana móvil, del esquema OPM/L, será una de nuestras herramientas para tal desarrollo, al igual que la codificación usada por el esquema LZSS.

La implementación de un algoritmo de búsqueda es la principal aportación de este documento, el cual está basado en los árboles binarios balanceados. Una ventaja de utilizar árboles binarios balanceados es la uniformidad de tiempo que se logra para codificar bloques de longitud N.

Igual que el esquema LZSS, este algoritmo se enfoca en el manejo del código ASCII, con una sola diferencia: éste utiliza los 8 bits para la representación del caracter además del bit de bandera, sumando un total de 9 bits. De esta forma, podrá comprimir tanto archivos de texto como archivos binarios.

Uno de los mecanismos que se modificaron fue el proceso de obtención de la pareja más grande para el buffer a codificar, mientras que el algoritmo de Thimoty C. Bell va registrando los nodos de la ruta de inserción del buffer a codificar que, están envueltos en los más recientes cambios izquierda y derecha, para determinar los mejores candidatos para la pareja más larga del buffer de codificación. En esta versión se van registrando todos los nodos que se encuentran en la trayectoria del nodo raiz al nodo que se insertó.

Implementación.

Durante la codificación, el árbol se actualiza continuamente con los cambios en la ventana móvil. Cada vez que la ventana se mueve a lo largo del texto, un caracter sale de la ventana. Esto ocasiona que el nodo asociado a este caracter,

sea borrado del árbol, así como el nuevo caracter que entra en la ventana, es insertado al árbol. Mediante el proceso de inserción se escogerá el par más largo.

Después de un borrado o una inserción se procede a ajustar los factores de balanceo. Si el árbol se ha desbalanceado, se efectuará una rotación simple o una rotación doble para rebalancear el árbol binario.

En la figura 3.3 se muestra un diagrama de bloques general para este algoritmo.

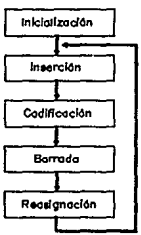


Figura 3.3 Diagrama de bloques del Algoritmo Universal de Compresión.

Análisis Detallado del Algoritmo.

Se tiene un arreglo W de N caractes, el cual representa la ventana. El árbol binario se representa con un arreglo matricial X. En él cual se almacena la información concerniente a las ligas de los nodos. Por último, el arreglo B contiene los factores de balanceo de cada nodo.

A continuación se describen los apuntadores que se utilizan en este algoritmo:

NODO	este apuntador se moverá hacia abajo del árbol. Inclusive, a partir de este se puede obtenter la posición, en la ventana, de un caracter.
S	apunta al lugar donde se necesite un rebalanceo.
T	apunta al padre de S.
BUFFER	apunta a los caracteres que constituyen el buffer a codificar.
LADO	apunta a la liga derecha o izquierda del NODO.

Por conveniencia, el algoritmo usa la notación 0, 1 y -1 para representar las ligas izquierda, derecha y superior, respectivamente.

Los pasos a seguir para la inserción de un nodo son los siguientes.

A1. **[Inicialización]** Asígnese a X[raiz,0] = raiz, X[raiz,1] = X[raiz,-1] = nil, B[raiz] = 0.

A2. **[Comparación]** Si W[BUFFER] < W[NODO] entonces lado = -1, sino lado = 1.

A3. **[Inspección.]** Asignar Q = X[NODO,LADO]. Si Q = nil, entoncer ir al paso A4. De otra forma si B(Q) <> 0, asignar T = NODO y S = Q. Finalmente asignar NODO = Q y regresar al paso A2.

A4. **[Insertar]** (Cuando se inserta un nuevo nodo en el árbol, sus campos necesitan inicializarse). Asignar X[NODO,LADO] = BUFFER, X[BUFFER,0] = NODO, X[BUFFER,-1] = X[BUFFER,1] = NIL, B(BUFFER) = 0.

A5. **[Ajuste de los factores de balance]** (Ahora los factores de balance de los nodos que se encuentran entre S y BUFFER necesitan ser cambiados de cero a ± 1). Si W[BUFFER] < W[S], asignar R = NODO = X[S,-1], de otra forma asignar R = NODO = X[S,1]. Luego repetidamente hacer la siguiente operación cero o más veces hasta que BUFFER = NODO; Si W[BUFFER] < W[NODO], asignar B(NOOD) = -1 y NODO = X[NODO,-1]; si W[BUFFER] > W[NODO], asignar B(NODO) = +1 y NODO = X[NODO,1]. (Si W[BUFFER] = W[NODO], entonces NODO = BUFFER y se continua con el siguiente paso).

A6. **[Balanceo]** Si W[BUFFER] < W[S] asignar a = -1, sino asignar a = 1. Hay diversos casos:

i) Si B(S) = 0, asignar B(S) = a, y termina el algoritmo.

ii) Si B(S) = -a, asignar B(S) = 0 y termina el algoritmo.

iii) Si B(S) = a, ir al paso A8 si B(R) = a, ir a A9 si B(R) = -a.

A7. **[Rotación simple]** Asignar NODO = R, X[S,a] = X[R,-a], X[R,-a] = S, B(S) = B(R) = 0. Ir a A9.

A8. **[Rotación doble]** Asignar NODO = X[R,-a], X[R,-a] = X[P,a], X[P,a] = R, X[S,a] = X[P,-a], X[P,-a] = S. Ahora asignar

$$(B(S),B(R)) \quad \begin{cases} (-a,0), & \text{si } B(NODO) = a; \\ (0,0), & \text{si } B(NODO) = 0; \\ (0,a), & \text{si } B(NODO) = -a; \end{cases}$$

y luego asignar B(NODO) = 0.

A9. **[Finalización]** Si S = raiz asignar raiz = nodo, x[raiz,0] = raiz. De otra forma si S = X[T,1] asignar X[T,1] = NODO sino X[T,-1] = NODO.

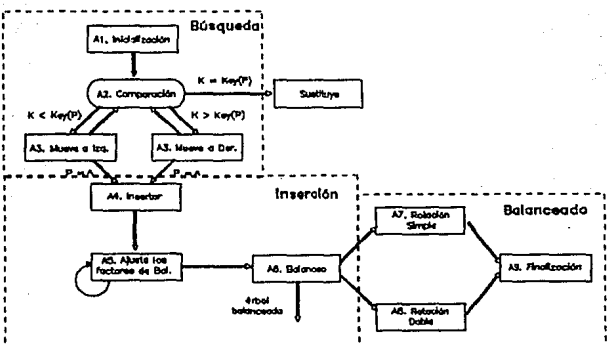


Figura 3.4 Diagrama de bloques del algoritmo de árboles balanceados

Con lo anterior se garantiza que la rutina de búsqueda de la pareja del *buffer de comparación* no requiera más de $\log_2 N + 2$ comparaciones, para un árbol con N nodos internos.

Resta señalar el procedimiento de borrado de uno de los nodos del árbol balanceado. Este proceso no es tan sencillo como el de los árboles ordinarios, ya que el árbol se puede desbalancear.

En primer lugar se analiza el proceso del borrado de un nodo P para el cual X[NODO,-1] ó X[NODO,1] ó ambos estan vacíos. El proceso consiste esencialmente de los siguientes pasos :

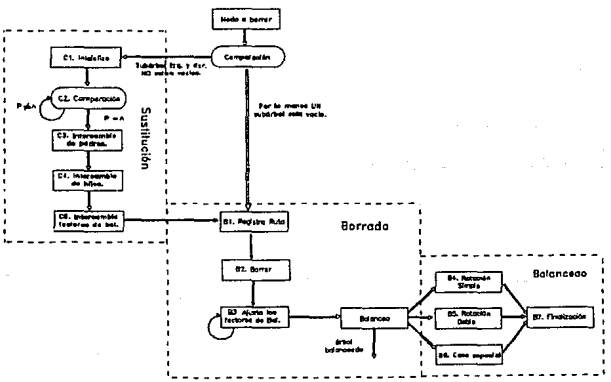


Figura 3.5 Diagrama de bloques detallado del Algoritmo Universal de Compresión

B1. **[Registro de Trayectoria]** (Se almacena en un stack la ruta desde el nodo raiz al nodo a Borrar). Asignar KEY=IV, SP=0 y repetir lo siguiente hasta que

NODO = X[NODO,0]. Establecer NODO = X[KEY,0], SP = SP + 1, PILANODO[SP] = NODO, KEY = NODO. Si X[NODO,1] = KEY entonces PILALADO[SP] = 1 sino PILALADO[SP] = -1.

B2. [Borrado] Asignar H1 = X[IV,-1], H2 = X[IV,1], PADRE = [IV,0]. Si X[PADRE,1] = IV asignar LADO = 1 sino LADO = -1. Si H1 = H2 = nil asignar X[PADRE,LADO] = nil de otra forma si H1 < > nil entonces X[H1,0] = PADRE Y X[PADRE,LADO] = H1, sino X[H2,0] = PADRE Y X[PADRE,LADO] = H2.

B3. [Reajuste de Factores] Inicializar CONT = 0 y repetir lo siguiente hasta que SP = CONT: asignar CONT = CONT + 1, NODO = PILANODO[CONT], a = PILALADO[CONT].

i) Si B(NODO) = 0 establecer B(NODO) = -a y CONT = SP
ii) Si B(NODO) = a establecer B(NODO) = 0
iii) Si B(NODO) = -a establecer T = X[NODO,0], S = NODO, R = X[S,-A]. Si B(R) = -a ir al paso B4, si B(R) = a ir al paso B5 y, si B(R) = 0 ir al paso B6.

B4. [Simple Rotación] Establecer NODO = R, X[S,-a] = X[R,a], X[R,a] = S, X[S,0] = R, B(S) = B(R) = 0. Si X[S,-a] < > nil asignar X[X[S,-a],0] = S.

B5. [Doble Rotación] Asignar NODO = X[R,a], X[R,a] = X[NODO,-a], X[NODO,-a] = R, X[S,-a] = X[NODO,a], X[NODO,a] = S. Ahora, establecer:

$$(B(S),B(R)) \begin{cases} (a,0), & \text{si} \quad B(NODO) = -a; \\ (0,0), & \text{si} \quad B(NODO) = 0; \\ (0,-a), & \text{si} \quad B(NODO) = a; \end{cases}$$

y asignar, B(NODO) = 0.

B6. **[Caso Especial]** Asignar NODO = R, X[S,-a] = X[R,A], X[R,a] = S, X[S,0] = R, B(R) = a. Si X[S,-a] < > nil entonces X[X[S,-a],0] = S.

El borrado de un nodo cuyos subárboles, izquierdo y derecho, no están vacíos consiste en: Buscar un nodo candidato que pueda sustituir al nodo a borrar. Este candidato debe ser el nodo más a la izquierda del subárbol derecho del nodo a borrar. Finalmente el nodo candidato es borrado con el algoritmo anterior. A continuación se muestra el proceso de sustitución:

C1. **[Inicialización]** Si B[iv] = -1 asignar LADO = -1 sino LADO = 1. Luego asignar NODO = iv.

C2. **[Comparación]** Repetir lo siguiente hasta que X[NODO,LADO] = nil: Asignar NODO = X[NODO,LADO]. Si W[iv] < W[NODO] entonces LADO = -1 sino LADO = 1.

C3. **[Intercambio de padres]** Si iv = raiz asignar raiz = NODO, X[raiz,0] = raiz de otra forma si X[X[iv,0],1] = iv asignar X[X[iv,0],1] = NODO sino X[X[iv,0],-1] = NODO. Luego asignar X[iv,0] = X[NODO,0]. Si X[X[NODO,0],1] = NODO asignar X[X[NODO,0],1] = iv sino X[X[NODO,0],-1] = iv.

C4. **[Intercambio de Hijos]** Asignar HIJO = X[NODO,-LADO], X[NODO,1] = X[iv,1], X[NODO,-1] = X[iv,-1], X[iv,-LADO] = HIJO, X[iv,LADO] = nil.

C5. [Asigna padre] Asignar X[X[NODO,-1],0] = X[X[NODO,1],0] = NODO. Si HIJO < > nil asignar X[HIJO,0] = iv.

C6. [Intercambio de Factores de Balance] B(NODO) = B(iv).

CAPITULO 4

Análisis y Comparación

Una vez descrita la implementación del algoritmo universal de compresión de datos, desarrollado en esta tesis, resta analizar el desempeño del mismo. Para fines prácticos se denominará al algoritmo anterior como ACUT. Como se mencionó anteriormente la velocidad y la tasa de compresión determinan el desempeño de un algoritmo de compresión de datos, por lo tanto dichos parámetros serán los temas a cubrir en éste análisis.

Los procesos que involucran a estos parámetros son los siguientes:

a) Inicialización de la estructura de datos (árboles balanceados).
b) Esquema de búsqueda (inserción de nodos).
c) Esquema de eliminación de nodos (borrado).
d) Obtención de los parámetros de distancia y longitud en la codificación.

Así que estos procesos formarán parte del análisis.

4.1 Velocidad de Compresión.

Una de las principales metas que se pretende alcanzar durante el desarrollo y realización de un algoritmo de compresión de datos es obtener una alta velocidad

de compresión, la cual no es más que el número de símbolos codificados por segundo, ya que de esta depende el retraso que sufre el flujo de información en los sistemas de comunicación.

En el capítulo 3 se mencionó la importancia de mejorar el esquema de búsqueda, es decir, incorporar una innovación a lo ya establecido o cambiarlo en su totalidad. En un principio, se tomó e implementó el esquema de búsqueda de los algoritmos LZ77 y LZSS, el cual requería de hasta (N-F)*F comparaciones para obtener los parámetros de longitud y distancia, cosa que resultaba lenta e impráctica. Sin embargo, con el pasar de los años surgieron técnicas novedosas para el manejo de datos y con ellas también surgió la idea de implementar el esquema de búsqueda con alguna de ellas y para esto se pensó en los árboles binarios.

Esta técnica, explicada en el capítulo anterior, disminuye considerablemente el número de comparaciones. Sin embargo, existe un problema: los procesos de inserción y borrado de nodos para esta estructura generan eventualmente un desequilibrio en ésta, llegando incluso a distorsionar completamente la forma binaria de los árboles. Lo anterior trae como consecuencia que el número de comparaciones se incremente al grado de alcanzar la cifra de (N-F)*F.

Puesto que el número de comparaciones y la actualización de nodos en el árbol binario determinan la velocidad de compresión, las rutinas de inicialización, inserción y borrado de nodos se modificaron; estas modificaciones se plantean a continuación.

Inicialización del árbol : la ventana, w, se inicializa con (F+1) símbolos de información, insertando en el árbol el primero de estos. Posteriormente se inicializan los parámetros del árbol. Inmediatamente después se ejecutan los procesos que integran al compresor.

Es obvio que mediante esta inicialización se disminuye el tiempo de compresión, ya que el número de inserciones se reduce a solo una en vez de N-F.

Inserción : la incorporación del esquema de árboles balanceados trajo consigo una importante reducción en el número de comparaciones, así como una estandarización en el tiempo de ejecución de ellas. El valor máximo de comparaciones oscila entre $(\log_2 N)*F$ y $(\log_2 N+1)*F$.

La forma de manejar estos árboles (arreglos), hace más comprensible su realización, sin embargo, para un mejor desempeño de estos se recomienda su realización mediante asignación de memoria dinámica, utilizando *apuntadores.*

La rutina de borrado : es sin duda el proceso donde se integraron los cambios más significativos para obtener una mayor velocidad de compresión, ya que originalmente este proceso degeneraba la forma del árbol. El esquema para borrar un nodo interno con ambos hijos, el cual es uno de los cambios, se diseñó tomando como base el algoritmo de borrado de un nodo frontera con un hijo como máximo. De esta manera no se tuvo que recurrir al esquema de concatenación de árboles propuesto por Clark A.Crane [8].

Mientras que con el esquema ACUT se necesitan un máximo de $(\log_2 N)*F$, comparaciones, el propuesto por Crane puede necesitar hasta el doble de lo anterior, es decir, $2*((\log 2\ N)*F)$ comparaciones, más el número de comparaciones que necesite el proceso de concatenación.

Analizando la figura 4.1, se corrobora la credibilidad del esquema ACUT.

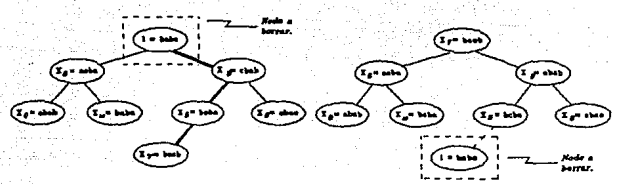


Figura (4.1.a) Árbol original. *Figura (4.1.b) Árbol después del borrado.*

De lo anterior se observa que para borrar el nodo *I*, primero se recorre el subárbol derecho de este árbol en busca del nodo más a la izquierda, inmediatamente después intercambian de lugar ambos nodos, de tal forma que aparezca como un nodo terminal, finalmente se ejecuta el algoritmo de borrado.

El proceso de codificación tal vez es el único proceso del ACUT que aumenta el tiempo de ejecución debido a su alcance. Ya que, originalmente estaba diseñado para comprimir textos de lengua inglesa, en el código ASCII del carácter 0 al 127, utilizando códigos cuyas longitudes eran múltiplos de 8 bits (byte), con lo cual su almacenamiento se facilitaba. Con la incorporación de un bit más a este código, su manejo se complicó, a tal grado; que su almacenamiento se tuvo que manejar a nivel bit. Tal manipulación consume cierto tiempo de ejecución que, sin embargo, es mínimo comparado con lo ganado anteriormente.

4.2 Tasa de Compresión.

Al inicio de esta tesis se mencionó que un algoritmo de compresión de datos es óptimo cuando no existe otro algoritmo que genere un código con una longitud menor a la del código de éste para representar a cada uno de los elementos de una fuente de información. Actualmente el algoritmo que se considera óptimo es el

Código de Huffman, no obstante, es posible demostrar a través de un modelo estocástico que el presente algoritmo, si bien no es óptimo, es muy práctico y eficiente.

Cabe mencionar que si bien el Código de Huffman, en teoría, es el mejor método de compresión, su desempeño no es muy práctico, ya que la base de su eficiencia depende en su totalidad del conocimiento previo de la probabilidad de ocurrencia de cada símbolo de la fuente, para posteriormente obtener un código para cada uno de ellos. Para obtener un código óptimo es necesario que la longitud de las sucesiones a codificar tienda al infinito o por lo menos sea bastante grande, lo cual resulta complejo e impráctico, por lo que generalmente se recurre a una longitud de sucesión igual a la unidad.

Es importante señalar que el desempeño de los algoritmos de compresión depende esencialmente de las frecuencias de repetición de patrones, por lo tanto la tasa de compresión varía de una fuente de información a otra.

La complejidad teórica y la falta de un modelo matemático que realice una predicción del comportamiento y desempeño del algoritmo ACUT, deja como única alternativa el uso de una técnica numérica, la simulación [12].

La simulación utiliza técnicas estadísticas para realizar una serie de serie de experimentos a través de una computadora digital, que muestren de alguna forma el comportamiento de un sistema. Para fines de esta tesis, la técnica estadística que se usará para determinar el comportamiento del algoritmo ACUT son los modelos de Markov.

La elección de estos modelos estocásticos radica en el hecho de que el mundo real es completamente probabilístico y la mayor parte de los sucesos depende de unas condiciones iniciales y una serie de excitaciones para determinar un estado futuro, el cual a su vez será la base para determinar los siguientes

estados (ver apéndice A).

A través de los modelos de Markov se generará una fuente de información (fuente de Markov) tal que las probabilidades de ocurrencia de los símbolos sean completamente conocidas, y con estas probabilidades se determinará el valor de su entropía. Conociendo éste valor junto con la tasa de compresión generada por el algoritmo ACUT al comprimir la fuente anterior es posible determinar la eficiencia de éste algoritmo.

Antes de iniciar el proceso de la simulación, es necesario definir y mencionar las características de las fuentes de Markov [12].

Fuente de Markov de Orden *m*

Es aquella en la cual la presencia de un determinado símbolo s_i depende de un número finito *m* de símbolos precedentes. Se define por su alfabeto fuente, S, y por un conjunto de probabilidades condicionales.

$P(s_i/s_{j1}, s_{j2}, ..., s_{jm})$ para $i = 1, 2, ...,q$; $j_p = 1, 2,...$

En una fuente de Markov de orden *m*, la probabilidad de un símbolo cualquiera se determina por la ocurrencia de los *m* símbolos que lo preceden. Por lo tanto, en cualquier momento, la probabilidad *del estado* de la fuente de Markov de orden *m* se define por los *m* símbolos precedentes. Puesto que existen q símbolos distintos, una cadena de Markov de orden *m* admitirá q^m estados posibles. Al emitir la fuente nuevos símbolos, el estado cambia.

Como último punto resta definir el concepto de la información media suministrada por una fuente de Markov de orden *m*. La información obtenida, si el símbolo s_i se presenta cuando el proceso se encuentra en el estado $(s_{j1}, s_{j2}, ..., s_{jm})$,

de acuerdo con la definición de información establecida en el capítulo 1, es

$$I(s_i | s_{j1}, s_{j2}, \ldots, s_{jm}) = \log \frac{1}{P(s_i | s_{j1}, s_{j2}, \ldots, s_{jm})} \qquad (4.1)$$

Por lo tanto, la cantidad media de información suministrada por símbolo cuando el proceso se encuentra en el estado $(s_{i1}, s_{i2}, \ldots, s_{im})$ está dada por la ecuación,

$$H(S | s_{j1}, s_{j2}, \ldots, s_{jm}) = \sum_{S} P(s_i | s_{j1}, s_{j2}, \ldots, s_{jm}) I(s_i | s_{j1}, s_{j2}, \ldots, s_{jm}) \qquad (4.2)$$

La cantidad media de información (entropía) de una fuente de Markov de orden *m*, se obtiene calculando el valor medio de esta cantidad, extendida a los q^m estados posibles, esto es,

$$H(S) = \sum_{S^m} P(s_{j1}, s_{j2}, \ldots, s_{jm}) H(S | s_{j1}, s_{j2}, \ldots, s_{jm}) \qquad (4.3)$$

donde las $P(s_{i1}, s_{i2}, \ldots, s_{im})$ representan las distribuciones estacionarias de la fuente, es decir, las probabilidades de equilibrio (ver apéndice A).

En este momento ya se cuenta con las herramientas necesarias para proceder a implementar la simulación antes mencionada, la cual consistirá de los siguientes pasos.

a) Se generará una matriz de transición de orden m, con una distribución de probabilidad previamente seleccionada. El valor de m que se elegirá es de 5, es decir, el número de elementos del alfabeto.

b) Se calculan las probabilidades de equilibrio, mediante el método de Gauss-Jordan.

c) Se calcula el valor de las entropías condicionales.

d) Se obtiene la entropía total a partir de las entropías condicionales.

e) Se genera una fuente de información con símbolos en el alfabeto q, qye cumplan con las probabilidades de la matriz de transición.

f) Se Comprime la fuente anterior con el algoritmo ACUT, y se obtiene la tasa de compresión generada por éste.

g) Se Compara el valor anterior con la tasa de compresión calculada a partir de la entropía.

Los resultados de tal simulación se exponen en la sección 4.3.

4.3 Resultados.

La presente sección ilustra la forma en la cual se realiza la simulación siguiendo la secuencia de pasos anteriormente indicada.

Tres experimentos son generados para determinar el desempeño del algoritmo ACUT. El primero de ellos genera algunas fuentes de información cuya probabilidad de ocurrencia de uno de sus símbolos es superior a la de los demás. El segundo experimento genera otras fuentes en las cuales la probabilidad de uno de sus símbolos es solamente un poco mayor a la de los demás símbolos. Por último, se generan fuentes de información cuyos símbolos son equiprobables.

El número de elementos de las fuentes de información es necesariamente el mismo, siendo indistinto su valor. Sin embargo, para lograr buenos resultados se utilizan fuentes de información con 10,000 símbolos, los cuales pertenecen a un alfabeto previamente seleccionado, S = { a, b, c, d, e }. Esto implica que un mínimo de 3 bits son necesarios para representar a cada uno de los símbolos.

- Experimento #1

Antes que nada se determinan las probabilidades de transición, las cuales se muestran en la matriz de la izquierda, posteriormente se calculan las probabilidades de equilibrio y se representan en forma matricial (matriz de la derecha). Con la matriz de transición se generan diferentes fuentes de información.

0.9	0.02	0.03	0.01	0.04
0.8	0.05	0.13	0.01	0.01
0.7	0.12	0.05	0.03	0.10
0.6	0.04	0.06	0.20	0.10
0.6	0.12	0.20	0.06	0.02

a) Matriz de transición

0.87	0.03	0.04	0.02	0.04
0.87	0.03	0.04	0.02	0.04
0.87	0.03	0.04	0.02	0.04
0.87	0.03	0.04	0.02	0.04
0.87	0.03	0.04	0.02	0.04

b) Probabilidades de equilibrio.

Una muestra de la fuente de información que generan las probabilidades de los estados en la matriz de transición es la siguiente:

eaaaaaaaaaaaaaaaaaadaabaaaaaaaaaaaaaaaaaaaaaaaaaacaaaacaaa
eaabcaaaaaaaaaaadaaaaaaabcadaaaaaaaaaaaaaaeaadaaaacaaaaaaaa
aaaaaaaaaaaabcaaaaaadaaaaaaaaaaaaaaaaaaaacaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaeaaaaaaaaaaaaaaaa

Al comprimir esta fuente de información con el algoritmo ACUT, se obtiene una fuente con 4,481 símbolos. Esto indica que la tasa de compresión es de 2.23 (# de símbolos de la fuente original / # de símbolos de la fuente comprimida). Se puede observar que, la fuente se comprimió más de la mitad. La entropía generada por el algoritmo es H(X) = **1.345** bits/símbolo.

Otro de los aspectos interesantes, que se observó, radica en que las

demás fuentes generadas con diferente semilla, en la generación de los números aleatorios, eventualmente se comportan en forma análoga a la fuente antes comprimida.

Por otro lado, se muestra la forma en la que se calcula la entropía de la fuente usando las expresiones (4.1), (4.2) y (4.3).

A partir de la ecuación (4.2) se obtienen las entropías condicionales.

$$H(S \mid a_k) = \sum_{i=1}^{m} P_{i|k} \log_2 1/p_{i|k}$$

donde m es igual al número de símbolos del alfabeto, es decir, m=5. Para k = 1, se tiene,

$H(S|a_1) = 0.9*\log_2 1/0.9 + 0.02*\log_2 1/0.02 + 0.03*\log_2 1/0.03 + 0.01*\log_2 1/0.01 + 0.04*\log_2 1/0.04$

$= 0.653640$

y para k = 2, 3, 4 y 5,

$H(S|a_2) = 0.989160$

$H(S|a_3) = 1.427324$

$H(S|a_4) = 1.668046$

$H(S|a_5) = 1.630043$

Ahora se ilustra el cálculo de la entropía total de esta fuente utilizando las probabilidades de equilibrio, las entropías condicionales y la ecuación (4.3), es decir,

$$H(S) = \sum_{k=1}^{m} H(S \mid a_k)\, \Pi_k$$

donde π es la probabilidad de equilibrio. Finalmente se tiene que la entropía total es,

$$H(S) = 0.653640*0.87 + 0.989160*0.03 + 1.427324*0.04 + 1.668046*0.02 + 1.630043*0.04 \text{ bits/símbolo}$$
$$= \mathbf{0.753997} \text{ bits/símbolo.}$$

A continuación solo se presentan los resultados de la entropía calculada para una fuente de Markov, la tasa de compresión y la entropía obtenida por el algoritmo ACUT en los siguientes 2 experimentos. Estos esxperimentos son iguales al anterior, con la excepción de que las probabilidades de transición cambian.

- Experimento #2

0.70	0.10	0.08	0.05	0.07
0.60	0.20	0.08	0.05	0.07
0.50	0.31	0.08	0.05	0.06
0.40	0.21	0.18	0.14	0.07
0.40	0.10	0.18	0.15	0.17

a) Matriz de transición

0.63	0.14	0.09	0.06	0.08
0.63	0.14	0.09	0.06	0.08
0.63	0.14	0.09	0.06	0.08
0.63	0.14	0.09	0.06	0.08
0.63	0.14	0.09	0.06	0.08

b) Probabilidades de equilibrio.

Esta es una muestra de la fuente de información generada por la matriz de transición,

acaaabbabaaaaeabcaabbabaaacbaeacacbaaaadabbdaabaabaababae
abaabadaaaaaaaabaacdacaaeaaaaaccaabbadaaaaaaacaacacdaacaea
ebbaaaabaaaabbaaacecaaaabbabacadcaaaaaaaaabbabaaaabaebdaa
daadabcabaaaaaadecaabaaaaaacabaaa

Tasa de compresión	:	1.25
Entropía, H(X)	:	2.4039 bits/símbolo
Entropía, H(S)	:	1.61938 bits/símbolo

- Experimento #3

0.20	0.20	0.20	0.20	0.20
0.20	0.20	0.20	0.20	0.20
0.20	0.20	0.20	0.20	0.20
0.20	0.20	0.20	0.20	0.20
0.20	0.20	0.20	0.20	0.20

a) Matriz de transición

0.20	0.20	0.20	0.20	0.20
0.20	0.20	0.20	0.20	0.20
0.20	0.20	0.20	0.20	0.20
0.20	0.20	0.20	0.20	0.20
0.20	0.20	0.20	0.20	0.20

b) Probabilidades de equilibrio.

Esta es una muestra de la fuente de información generada por la matriz de transición,

eabcbebaebdbedbcbeaeaebaceeccabddcacecdeacdbeaadecccaccae
dcedcbecacdccdaedbebcbaccebbcddceeddcccbcddddcbcdeeccbbdb
baeaccebbddecbbedabdddadcbcadecbddaaeeeabeccbdadccacedada
debebcaddbbddbecacbedebccccccceaa

Tasa de compresión	:	0.79
Entropía, H(X)	:	3.7882 bits/símbolo
Entropía, H(S)	:	2.3219 bits/símbolo

La siguiente gráfica ilustra el comportamiento del algoritmo ACUT al comprimir las fuentes de informnación ya citadas.

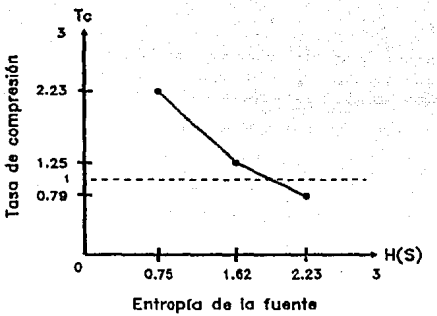


Figura 4.2 Comportamiento del algoritmo ACUT.

La gráfica muestra claramente que este tipo de fuentes markovianas se comporta de la siguiente manera: a mayores entropías el desempeño de algoritmo ACUT no estan eficiente como para aquellas fuentes en las cuales sus entropías son menores. Es obvio que el desempeño del algoritmo esté relacionado directamente con la incertidumbre de las fuentes, ya que su funcionamiento se basa en comparaciones de sucesiones de sus símbolos.

Para demostrar que en la práctica el desempeño del algoritmo ACUT es eficiente, a continuación se presenta una relación de diferentes tipos de fuentes de información y fuentes comprimidas, así como sus tasa de compresión.

Fuente de Información	Nombre de la Fuente	Tamaño Original (Bytes)	Fuente comprimida	Tasa de compresión
Texto	prologo.txt	15, 745	3, 938	25%
Código	prueba.pas	10,578	3, 804	35%
Binario	unzip.exe	23, 044	16, 631	72%
Gráfica	imagen.bmp	355, 400	71, 100	20%

Tabla 4.1 Tasa de compresión para diferentes archivos.

Otro parámetro importante de considerar, es la comparación de la tasa de compresión generada por el algoritmo ACUT y la generada por otros algoritmos comerciales tales como el **compress** de UNIX y el **pkzip** del DOS.

Fuente de Información	Nombre de la Fuente	Tamaño Original (Bytes)	Compresión ACUT	Compresión COMPRESS	Compresión PKZIP
Texto	prologo.txt	15, 745	3, 938	4, 125	3, 825
Código	prueba.pas	10,578	3, 804	4, 064	3, 740
Binario	unzip.exe	23, 044	16, 631	17, 124	15, 897
Gráfica	imagen.bmp	355, 400	71, 100	75, 572	68, 761

Tabla 4.2 Funcionalidad de diferentes esquemas de compresión.

Como se puede observar el algoritmo ACUT si bien no es el mejor puede competir con algunos esquemas comerciales.

CAPITULO 5

Aplicaciones Prácticas

Hoy en día el uso de los esquemas de compresión de datos es una de las actividades más comunes dentro de las áreas de transmisión y almacenamiento de datos. Algunos sistemas de comunicación los han incorporado con el fin de obtener "mayores" tasas de transmisión y proporcionar la opción de introducir distorsión positiva (código corrector de errores), esto se hace con el fin de disminuir la tasa error en la transmisión, como es el caso de los sistemas unidireccionales. Generalmente los esquemas de compresión de datos generan una fuente codificada (comprimida) a partir de un fuente de información, cuyo número de símbolos es menor a la original. Esto muestra claramente que un sistema de comunicación transmite más rápidamente una fuente comprimida que la fuente original, permitiendo de esta forma, utilizar mejor el canal de comunicación.

En lo que se refiere al almacenamiento de datos en los equipos de cómputo, se ha observado que la cantidad de información que manejan se ha incrementado exponencialmente llegando incluso a volver obsoletos a algunos modelos de computadoras fabricadas apenas un par de años atrás. Por otro parte, la gran cantidad de memoria requerida para el desarrollo e instalación de software y la limitada capacidad de manejo de los dispositivos de almacenamiento, ha iniciado una nueva etapa dentro de los sistemas computacionales, la etapa de **compactación de software.**

El objetivo principal del presente capítulo es la de exponer algunas aplicaciones, como las ya mencionadas, de los esquemas de compresión de datos

en sistemas de comunicación y de almacenamiento de información digital, los cuales de alguna manera presentan un serio problema para manejar adecuadamente sus recursos y poder obtener un buen desempeño del mismo.

5.1 Sistemas de Transmisión de Datos.

Actualmente existen diversos sistemas de comunicación que buscan obtener un buen aprovechamiento de sus recursos, principalmente del medio de comunicación (radio, satélite, líneas privadas, etc.). Una posible solución es comprimir las fuentes de información que se van a manejar.

El problema anterior se presentó en el sistema de comunicación por subportadora de audio FM denominado SEDI (Servicio Electrónico de Distribución de Información), el cual se desarrolló en la empresa MIL TEL XXI. A continuación se da una descripción general de su operación.

SEDI es un sistema de transmisión de datos que utiliza una subportadora de audio FM, específicamente la frecuencia de 100.9 mhz. (Rock 101, estación de radio de Núcleo Radio Mil), el cual permite enviar información desde un punto central hacia varios puntos diseminados en el Area Metropolitana de la Ciudad de México. Por la naturaleza del sistema, no hay límite en el número de receptores.

La operación general de dicho sistema es básicamente la siguiente:

1. Los usuarios transmisores envían a la estación central de SEDI sus mensajes a través de una computadora PC y un modem de 1200 bps. Dichos mensajes son seleccionados, paquetizados y transmitidos a través de un programa de aplicación proporcionado por SEDI. El destino de los mensajes es previamente elegido por el usuario con ayuda del programa.

2. Para enlazar el modem, el propio programa marca un número telefónico correspondiente al Centro de Operaciones de SEDI, en el cual se reciben y registran los mensajes.

3. Una vez que se reciben y registran los mensajes, estos se preparan para ser transmitidos a la estación de radio FM, ubicada en las instalaciones de Radio Mil, en Santa Fe. El proceso de preparación constituye la incorporación de un código corrector de errores y opcionalmente encripción.

4. La transmisión a la estación de radio de FM, se lleva a cabo a través de un enlace de microondas o de radio punto a punto.

5. Los mensajes que se reciben en la estación de radio se mezclan con la señal de audio y se transmiten al aire junto con ésta, a una tasa de 4800 bps.

6. Los usuarios receptores que se encuentren dentro del área de cobertura de la estación FM, recibirán la señal de radio, y por medio de equipos especializados, se extrae de ella los datos que les corresponden. Esto lo realizan por medio de un previo direccionamiento que se incorpora a los mensajes al momento de ser transmitidos.

La figura 5.1, muestra la operación del sistema SEDI.

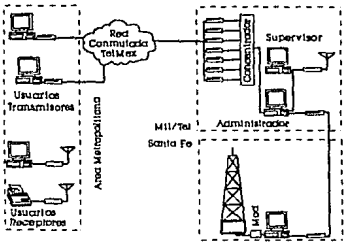


Figura 5.1 Configuración de la red de Mil/Tel.

Por otra parte, en lo que se refiere a la información a transmitir, ésta se paquetiza previamente, por el mismo programa proporcionado por SEDI. La paquetización consiste en dividir la información en fragmentos de igual tamaño, estos fragmentos son conocidos como *paquetes*. El manejo de la información mediante paquetes hace más eficiente la transmisión, y permite controlar en forma óptima el destino de ésta. Ya que, a estos paquetes se puede incorporar la dirección destino y la dirección origen. Además de establecer un esquema de reconocimiento y análisis de la información transmitida.

Los paquetes se dividen básicamente en dos tipos, **Ruteo** y **Datos**. El primero de ellos lleva la información necesaria para que las unidades receptoras, a las cuales se envía la información, se preparen para recibir los mensajes exitosamente. El segundo paquete lleva la información enviada por el transmisor. El formato de ambos paquetes se describe a continuación.

PAQUETE DE RUTEO

Tipo	Familia	Dirección	Aplicación	NTFrms	Nombre	IDA	REP	SUSO	Crc16

Los campos del paquete se interpretan de la siguiente manera:

Tipo	Identificador de paquete (Ruteo ó Datos).
Familia y Dirección	Seleccionan el destino de la información (unidades receptoras).
Aplicación	Indica el dispositivo al cual, la unidad receptora enviara la información recibida (PC, impresora, etc).
NTFrms	Señala el número de paquetes de datos que vienen en camino.
Nombre	Proporciona el nombre del archivo que se esta transmitiendo
IDA	Señala el identificador del archivo a enviar.
REP	Marca el número de veces que los paquetes se transmitirán.
SUSO	Son bytes libres que se mantienen para uso futuro.
Crc16	Diagnostica si los paquetes llegarón sin error.

PAQUETE DE DATOS

Tipo	NFrame	IDB	INFORMACION	Crc16

Los campos de Tipo, IDB y Crc16 funcionan exactamente igual que en el paquete de Ruteo. El NFrame indica el número de secuencia que le corresponde a cada uno estos paquetes. El campo de información contiene los datos a transmitir.

Anteriormente se mencionó que la incorporación de un código corrector de errores es uno de los recursos que utiliza SEDI para disminuir la tasa de error de los mensajes recibidos. Sin embargo, el uso de esta herramienta decrementa notoriamente el desempeño del sistema, ya que el número de símbolos (bytes) a

transmitir se incrementa. Este incremento depende del código corrector que se utilice y para el caso de SEDI, el número de símbolos por paquete se duplica. Lo anterior hace que los mensajes tarden más en llegar a sus destinos, pero a cambio se logra una mejor integridad en los datos recibidos.

Por otro lado, debido a que la propia naturaleza del sistema (unidireccional) impide garantizar que la información llegue libre de error, en SEDI se implementó un esquema que permite disminuir la tasa de error. Este esquema es el denominado **repetidor de paquetes** y se encarga de enviar cada paquete N veces (número de repetición). El número de repeticiones incrementa el número de símbolos a transmitir (Nt), el cual esta dado por la siguiente expresión:

$$Nt = (\text{longitud del paquete}) * 2 * N.$$

El número de repeticiones, aunado con el código corrector de errores, decrementa considerablemente el desempeño del sistema. La aplicación de una técnica de compresión de datos puede ser la solución del problema para que éste recupere en gran parte su desempeño original. El algoritmo se implementaría como una rutina más del programa proporcionado por SEDI. El algoritmo de compresión de datos se aplica de la siguiente manera:

a) El programa transmisor de mensajes, que SEDI proporciona, llama a una rutina que comprimirá la información del archivo a enviar. Una vez que el archivo se ha comprimido, éste se paquetiza para su posterior transmisión.

b) La rutina expansora residirá en las unidades receptoras, también proporcionadas por SEDI, las cuales irán expandiendo los paquetes en línea, esto es, conforme vayan llegando los paquetes, el expansor obtendrá el archivo original y lo enviará ya sea a la impresora, computadora ó demás equipos electrónicos (display, monitores, etc.) que cumplan con las características de SEDI.

La figura 5.2 ilustra de una manera más clara lo anterior.

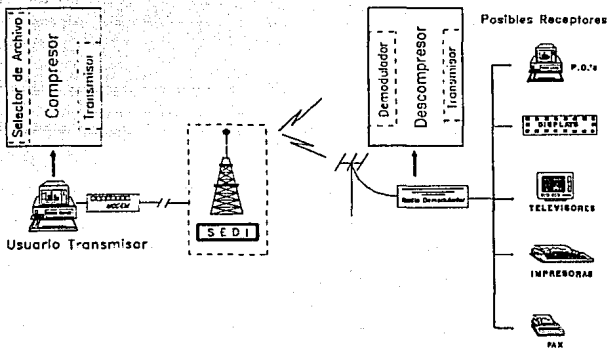


Figura 5.2 Incorporación del algoritmo compresor.

El algoritmo de compresión a implementar es el propuesto en esta tesis, ya que por su universalidad y buen desempeño se adapta a las necesidades de SEDI.

Las posibles desventajas de la incorporación de un esquema de compresión de datos con características similares al presente, se presentan al momento de recibir los datos, ya que el arribo de un solo símbolo (byte) erróneo en un punto clave del formato, iniciará una expansión errónea del mensaje , corrompiendo en su totalidad todos los datos restantes del mensaje.

5.2 Sistemas de Control de Tarjetas de Crédito.

Con el uso de las tarjetas de crédito, las empresas bancarias han innovado el proceso de adquisición de bienes de millones de personas, reduciendo la circulación de la moneda. Sin embargo, el control de dichas tarjetas constituye, en

estos momentos, un gran problema para la industria bancaria por el mal empleo que se le da a las tarjetas de crédito extraviadas y a la deficiencia de los sistemas de actualización de las mismas.

A partir del surgimiento de las tarjetas de crédito, la industria bancaria se ha esmerado en desarrollar grandes y sofisticados sistemas para su actualización. En un principio, se contaba con pequeños libros de registro, en los cuales los usuarios podían consultar el número de las tarjetas dadas de alta, así como el de las tarjetas extraviadas. Sin embargo, con la gran demanda que ha tenido este servicio, este sistema se volvió por demás obsoleto. En los últimos años, los sistemas de comunicación han tomado un papel importante para mantener actualizados los bancos de datos de las tarjetas de crédito en todos los puntos remotos en los cuales se maneja este sistema. Esta medida ha resultado muy acertada, no obstante, el costo de esta operación resulta bastante elevado, sobre todo cuando se manejan medios de comunicación con tasas de transmisión bajas.

Ante tal problema se ha considerado la posibilidad de implementar algún esquema de compresión de datos con el fin de reducir el número de símbolos a transmitir. Los algoritmos universales de compresión de datos, son una buena elección para reducir el tamaño de los archivos que contienen la información de las tarjetas de crédito. Sin embargo, el diseño de un algoritmo de compresión hecho especialmente para este tipo de fuente, es lo más recomendable, ya que se pueden alcanzar grandes tasas de compresión, claro que sin llegar a ser menores que la propia entropía de la fuente.

Ahora, una vez formulado el problema del diseño de tal algoritmo, es necesario analizar la naturaleza de la información contenida en los archivos de control de tarjetas de crédito. Para este caso se usará un archivo denominado archivo negativo, el cual se usa para registrar el número de las tarjetas extraviadas del Banco Nacional de México (BANAMEX). Generalmente estos archivos están ordenados en forma ascendente. Los números de las tarjetas que manejan se

componen de 16 dígitos, los cuales se almacenan por su correspondiente código ASCII. Por ejemplo, el número '1345267845678934' se almacena en código ASCII de la siguiente manera : #49, #51, #52, #53, #50, #54, #55, #56, #52, #53, #54, #55, #56, #57, #51, #52. El símbolo '#' indica el correspondiente código ASCII.

Con las características ya mencionadas, se observa que el diseño del algoritmo se torna cada vez más factible. De hecho, este algoritmo se basará esencialmente en obtener las diferencias entre los números de tarjetas sucesivas.

La implementación del algoritmo consiste de los siguientes pasos.

1) Obtener un número de tarjeta como patrón, esto es, tener alguna base a partir de la cual se irán obteniendo las diferencias; por la naturaleza de archivo se tomará el primer número. Sea N_p el número patrón.

2) Obtener la diferencia entre N_p y el siguiente número, este será denominado N_{p+1}.

3) Codificar esta diferencia.

4) Asignar a N_p el valor de N_{p+1} y repetir el paso 2 hasta comprimir todo el archivo.

En lo que respecta al expansor, éste realiza el proceso inverso, es decir, se toma el número patrón N_p y la diferencia, para posteriormente sumarlos y obtener el número N_{p+1}.

El formato de la codificación es el siguiente:

N_p	Flag	NbD	Diferencias	...	Flag	NbD	Diferencias	...

FORMATO DE CODIFICACION

Los campos se interpretan de la siguiente forma:

N_p	Número patrón del archivo.
Flag	Bandera de inicio de bloque de diferencias, es decir, marca el inicio de una serie de diferencias que tendrán características similares.
NbD	Marca el número de bits que necesitan las diferencias que están a continuación, todo el bloque de diferencias que sigue tiene el mismo número de bits.

Los resultados obtenidos al aplicar este esquema (particular) superaron considerablemente a los obtenidos en el algoritmo universal. La tabla 5.1 muestra las estadísticas obtenidas al comprimir un archivo de 10 Mbytes con ambos esquemas.

Tabla 5.1 Tabla de Resultados.

Algoritmo	Tasa de Compresión	Velocida de Compresión	Velocidad de Expansión
Universal	75%	512 bytes/s	8 kbytes/s
Particular	94%	16 kbytes/s	16 kbytes/s

Al combinar este método de compresión con un sistema de transmisión como el implementado por SEDI, los sistemas bancarios de actualización de números de tarjetas de crédito extraviadas, pueden lograr un buen desempeño en cuanto a la actualización de éstas.

5.3 Sistemas de Almacenamiento de Datos.

La demanda de dispositivos de mayor capacidad de almacenamiento digital en el mercado, los elevados precios de aquellos que cubren éstas necesidades y los bajos recursos económicos de la mayoría de los usuarios de PC's, ha generado un gran interés de estos últimos, en poseer una herramienta que les permita utilizar al máximo sus recursos de almacenamiento de

información. Actualmente hay programas capaces de almacenar diversos archivos en uno sólo, manteniendo íntegra toda la información de los archivos. Con la ayuda de un buen algoritmo de compresión de datos y el diseño de un formato adecuado para unir los diferentes archivos, se puede implementar una técnica que realice lo antes estipulado.

Antes de iniciar el análisis de la implementación de un esquema de compresión masiva, es necesario cubrir el tema de la compactación simple, es decir, la compresión de un solo archivo. Para hacer esto, basta con invocar un programa de compresión, basado en el algoritmo expuesto en esta tesis. Indicar el archivo a comprimir, y opcionalmente el nombre donde radicará el archivo comprimido; en caso de no proporcionar el archivo destino, el programa lo asignará con el nombre del archivo fuente más una extensión por default. El formato es el siguiente:

Encoder <archivo a comprimir> [archivo destino]

donde Encoder es el programa compresor de archivos.

En lo que se refiere al descompresor, el formato es muy similar al del compresor, esto es, se invoca al programa descompresor, se indica el archivo a descomprimir, y opcionalmente se asigna un nombre de archivo destino. El formato del descompresor es el siguiente:

Decoder <archivo a descomprimir> [archivo destino]

en donde Decoder es el programa expansor.

Una vez descrita la forma de compresión de un solo archivo, se procederá a describir el método de compresión colectiva.

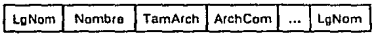
El formato de este método es muy similar al anteriormente descrito, con una pequeña variante, el nombre del archivo puede ser uno de los siguientes formatos:

- <nombre1, nombre2, ...,nombreN>
- <nombre1.*, nombre2.*, ...nombreN.*>

- <*.ext1, *.ext2, ..., *.extN>

- <*.*>

Todos estos archivos, una vez comprimidos, serán almacenados en un único archivo destino, con el siguiente formato.



La descripción de cada uno de campos se muestra acontinuación.

LgNom	Señala el número de caracteres que constituye el nombre del archivo.
Nombre	Indica el nombre del archivo comprimido.
TamArch	Indica el tamaño del archivo comprimido.
ArchCom	Constituye en sí el archivo comprimido.

Esta técnica de comprimir varios archivos en uno ofrece las siguientes ventajas:

- Fácil selección de los archivos a comprimir.

- Óptima portabilidad.

- Ahorro de memoria en disco.

Con este formato el expansor obtendrá los archivos originales así como sus respectivos nombres, sin error alguno.

Esta aplicación puede competir con algunos programas de compresión que se han desarrollado en la industria del software. Adicionalmente, ofrece a los usuarios de PC's la opción de desarrollar su propio software, basándose en toda la teoría mencionada en esta tesis.

CAPITULO 6

Conclusiones y Recomendaciones

En esta tesis se presenta los elementos matemáticos, de ingeniería y de computación necesarios para entender el desarrollo y la aplicación de algoritmos de compresión de datos.

Se presentó también el desarrollo completo de un algoritmo nuevo, derivado de una serie de esquemas de compresión de datos. Este algoritmo proporciona a todo aquel usuario, involucrado en el desarrollo de sistemas de cómputo, la oportunidad de aprovechar eficientemente los recursos de almacenamiento digital de sus equipos. También ofrece a los diseñadores de sistemas digitales de comunicaciones una herramienta que les permita utilizar más eficientemente el ancho de banda del canal de que disponen.

Un caso concreto de un sistema digital de comunicaciones, al cual se aplica el algoritmo de compresión desarrollado en esta tesis, es el sistema de radiodistribución punto a multipunto, en una subportadora de una estación comercial FM, SEDI.

Independientemente de que el algoritmo funciona en la actualidad de manera satisfactoria, se analizó su comportamiento por medio de simulaciones que permiten verificar condiciones bajo las cuales el algoritmo tiene un buen desempeño, coincidiendo éstos resultados con lo esperado teórica e intuitivamente.

Adicionalmente, más no como uno de los objetivos principales de la tesis, cabe mencionar que las herramientas que aquí se exponen tiene como consecuencia un aumento en la seguridad de la información, ya que los archivos, al ser comprimidos, mantienen un código completamente desconocido para aquellos usuarios que en un determinado momento los copiaran o quisieran hacer mal uso de ellos.

La simplicidad y el buen desempeño del algoritmo ACUT, en la práctica, hacen factible que sus usuarios puedan no solo emplearlo, sino incluso modificarlo para sus propias necesidades, sin mayor esfuerzo.

Por otra parte, el lector puede cuestionar (con cierta razón) el punto que se refiere a la velocidad de compresión, ya que esta puede no satisfacer sus necesidades de procesamiento.

La velocidad de compresión depende esencialmente del esquema de búsqueda que se utilice. Por lo tanto se recomienda, para trabajos futuros, explorar y posiblemente mejorar el esquema de búsqueda utilizando, por ejemplo, la técnica de asignación de memoria dinámica.

Como último punto se garantiza que este algoritmo se puede aplicar a cualquier tipo de información: gráficas, números de cuenta de bancos, textos, archivos binarios, etc. Sin embargo, si el usuario necesita comprimir un solo tipo de información, se recomienda desarrollar un algoritmo particular basándose en el algoritmo universal, de esta forma se obtiene una mejor tasa de compresión.

APENDICE

Cadenas de Markov.

La teoría de colas y los procesos estocásticos son las herramientas fundamentales para modelar sistemas de comunicación. A pesar de que se hacen algunas suposiciones para simplificar los modelos que representan un sistema, es indiscutible que estas áreas del conocimiento han permitido realizar diseños muy sofisticados de sistemas de comunicación.

Los procesos markovianos son considerados como los más importantes y fundamentales para la teoría de colas. Un proceso markoviano de estados discretos se conoce como **Cadena de Markov.**

Un conjunto de variables aleatorias (V.A.) $\{X_n\}$ forma una cadena de Markov si la probabilidad de que el siguiente valor o estado X_{n+1} (denotada por x_{n+1}) depende únicamente del estado presente X_n y no de cualquier estado anterior. Esto es, la única forma en que la historia afecta a su futuro está resumido en el estado actual de éste. A lo anterior se le conoce como **Propiedad Markoviana** [10]. Matemáticamente :

$$Pr\{ X_{n+1}=x_{n+1} \mid X_n=x_n, \ldots, X_1=x_1 \} = Pr\{ X_{n+1}=x_{n+1} \mid X_n=x_n \}.$$

Por otra parte, las probabilidades condicionales $Pr\{ X_{n+1}=j \mid X_n=i \}$ se conocen como **probabilidades de transición.** Si para cada i y j, pertenecientes al espacio de estados

$$Pr\{ X_{n+1}=j \mid X_n=i \} = Pr\{ X_1=j \mid X_0=i \}$$

entonces se dice que las probabilidades de transición (de un paso) son estacionarias y por lo general se denota p_{ij}. El tener probabilidades de transición estacionarias implica que las probabilidades de transición no cambian con el tiempo. La existencia de probabilidades de transición (de un paso) estacionarias también implica que, para cada i, j y n (n = 0, 1, 2, ...),

$$Pr\{ X_{t+n}=j \mid X_t=i \} = Pr\{ X_n=j \mid X_0=i \},$$

para toda t = 0, 1, ... Estas probabilidades condicionales se denotan por $p_{ij}^{(n)}$ y son conocidas como probabilidades de transición de n pasos. El término $p_{ij}^{(n)}$ es simplemente la probabilidad condicional de que la variable X, comenzando en el estado i, se encuentre en el estado j después de n pasos (unidades de tiempo).

Definiendo formalmente a las cadenas de Markov, se tiene que, un proceso estocástico $\{X_t\}$ (t = 0, 1, ...) es una *Cadena de estado finito* si cumple con lo siguiente:

1. Existe un número finito de estados.
2. Se cumple la propiedad markoviana.
3. Las probabilidades de transición son estacionarias.
4. Un conjunto de probabilidades iniciales $Pr\{X_0 = i\}$ para toda i.

A continuación se presentán los conceptos más importantes que se manejan en las cadenas de Markov [11].

Homogeneidad Una cadena de Markov discreta es homogénea si las probabilidades de transición no dependen del tiempo. Esto es,

$$Pr\{ X_n=j \mid X_{n-1}=i \} = P_{ij}.$$

La P_{ij} es la probabilidad de transición del estado E_i al estado E_j de la cadena en una etapa. Generalizando a k etapas, tenemos

$$P_{ij}^{(n)} = P(X_{n+k}=j \mid X_n=i).$$

Aplicando el método de Chapman-Kolmogorov [10] llegamos a la ecuación,

$$P_{ij}^{(n)}=\sum_{k=0}^{m} P_{ik}^{(n-1)}P_{kj} ,$$

de la cual resulta que las probabilidades de transición de n pasos se pueden obtener a partir de las probabilidades de transición de un paso de manera recursiva.

Irreducibilidad Una cadena de Markov se dice que es irreducible si todos los estados son alcanzables de cualquier otro estado. Es decir, para cada par de estados E_i y E_j, existe un número entero m tal que: $P_{ij}^{(m)} > 0$.

Tiempo de Primera Pasada Es el número de transiciones que hace el proceso al ir de un estado i a un estado j por primera vez.

En general los tiempos de primera pasada son variables aleatorias y por lo tanto, tienen una distribución de probabilidad asociada a ellos. Estas distribuciones de probabilidad dependen de las probabilidades de transición del proceso. En particular $f_{ij}^{(n)}$ denota la probabilidad de que el tiempo de primera pasada del estado i al estado j sea igual a n. Para fines prácticos seas j=i y definase a $f_{ii}^{(n)}$ como,

$$f_i \triangleq \sum_{n=1}^{m} f_{ii}^{(n)}$$

De lo anterior se tiene que un estado E_i de una cadena de Markov se define como:

Recurrente, si $f_i = 1$.
Transitorio, si $f_i < 1$.

Considerando a los estados recurrentes ($f_i = 1$), se define el tiempo promedio de recurrencia como,

$$\mu_{ii} = \sum_{n=1}^{\infty} n f_{ii}^{(n)}$$

con esto se pueden exponer otros dos conceptos, estado **recurrente nulo** , si $\mu_{ii} = \infty$, y estado **recurrente positivo**, si $\mu_{ii} < \infty$ [11].

Una última propiedad de las cadenas de Markov que debe tomarse en cuenta es la de *periodicidad*. Se dice que un estado i tiene un período t (t > 1) si $P_{ii}^{(n)} = 0$ siempre que n no sea divisible entre t, y t es el entero más grande con esta propiedad, es decir, el máximo común divisor. Si t = 1 entonces el estado se conoce como *estado aperiódico*. Los estados recurrentes positivos que son aperiódicos se llaman **estados ergódicos**.

Se puede demostrar que para una cadena de Markov irreducible ergódica el $\lim p_{ij}^{(n)}$, con n tendiendo al infinito, existe y es independiente de i, el estado inicial. Esto es,

$$\lim_{n \to \infty} P_{ij}^{(n)} = \pi_j$$

en donde las π_j's satisfacen de manera única el siguiente sistema de ecuaciones:

$$\pi_j > 0$$

$$\pi_j = \sum_{i=0}^{M} \pi_i P_{ij} \text{ , para } j = 0, 1, \ldots, M$$

$$\sum_{j=0}^{M} \pi_j = 1 .$$

Las π_j's se llaman probabilidades de **estado estable ó de equilibrio** de la cadena de Markov y son iguales al inverso del tiempo esperado de recurrencia, es decir

$$\pi_j = \frac{1}{\mu_{jj}} \text{ , para } j = 0, 1, \ldots, M.$$

El término probabilidad de *estado estable* significa que la probabilidad de encontrar el proceso en un cierto estado, por ejemplo j después de un número grande de transiciones, tiende al valor π_j, y es independiente de la distribución de probabilidad inicial definida para los estados. Representando dichas probabilidades en términos matriciales, se tiene

$$\Pi = \Pi P.$$

donde P es la matriz de transición, y π es el vector de las probabilidades de equilibrio.

Una vez expuesta la teoría de las cadenas de Markov, resta explicar brevemente la forma en que se utilizan estos conceptos en la presente tesis.

La matriz de transición es sin lugar a dudas el principal concepto a usar, ya que, de esta serie de probabilidades p_{ij} es posible generar una fuente de información cuyos símbolos dependen de un alfabeto previamente seleccionado. Esta fuente es de particular interés, ya que, a través de ella es posible determinar la eficiencia del algoritmo desarrollado en esta tesis. Esta determinación se realiza por medio de una simulación la cual se basa en los modelos de Markov.

Las probabilidades estacionarias son utilizadas para determinar teóricamente el mínimo número de símbolos binarios, bits, necesarios para representar a cada uno de los símbolos de la fuente antes generada.

Referencias

[1] GALLAGER, Robert G.,
"Information Theory y Reliable Communication",
John Wiley and Sons,
New York, 1968.

[2] BLAHUT, Richard E.,
"Principles and Practice of Information Theory",
Addison-Wesley,
Owego, New York, 1987.

[3] ABRAMSON, N.M,
"Information Theory and Coding",
Mc Graw-Hill,
New York, 1963.

[4] LATHI, B.P.,
"Sistemas de Comunicación",
Interamericana,
México, 1986.

[5] BELL, Timothy C.,
"Better OPM/L Text Compression",
IEEE Transactions on Conmunications,
vol. COM-34, no. 12, pp. 1176-1182,
December, 1986.

[6] WILLEMS, Frans M.J.,
"Universal Data Compression and Repetition Times",
IEEE Transaction on Information,
vol. 35, no. 1, pp. 54-58,
January 1989.

[7] KNUTH, Donald E.,
"The Art of Computer Programming",
Addison-Wesley,
1973.

[8] TANEMBAUM, Aaron M.,
"Data Structures using Pascal",
Prentice-Hall,
1981.

[9] CINLAR, Erhan,
"Introduction to Stochastic Processes",
Prentice-Hall,
New Jersey, 1975.

[10] HILLER, Federic, LIEDERMAN, Gerald,
"Introducción a la Investigación de Operaciones",
Mc Graw-Hill,
México, 1982.

[11] ZENG, Chaoming,
"Análisis de Enlaces de Comunicación con Conmutación Híbrida",
Tésis de Maestría,
México, 1985.

[12] KOBAYASHI, Hisashi,
Modeling and Analysis: An Introduction to System Performance
Evaluation Methodology,
Addison-Wesley,
New York, 1978.