Apuntes

Tema ***Compresión***

"Neo, tarde o temprano te darás cuenta, como yo, que es diferente conocer el camino que recorrerlo."
Morfeo en la película "The Matrix"

Compresión de Datos

Introducción

La compresión de datos se utiliza en gran medida en una gran variedad de contextos de las ciencias informáticas. Todos los sistemas operativos populares y los lenguajes de programación tienen varias herramientas y bibliotecas para tratar la compresión de datos de varios tipos. La elección correcta de herramientas de compresión y de bibliotecas para una aplicación determinada depende de las características de los datos y de la aplicación en cuestión: transmisión y archivo; patrones esperados y regularidades en los datos; importancia relativa del uso del CPU, uso de la memoria, demandas de canales y requisitos de almacenamiento; y otros factores.

Y, entonces... **¿Qué es la compresión de datos?** La respuesta corta es que la compresión de datos elimina la redundancia de los datos; en términos teóricos de información, la compresión aumenta la ***entropía*** del texto comprimido. Pero estas afirmaciones son esencialmente verdades por definición. La redundancia se presenta de muchas formas. Las secuencias de bit repetidas (11111111) son un tipo. Las secuencias de byte repetidas son otro (XXXXXXXX). Sin embargo, más a menudo, las redundancias tienden a aparecer a mayor escala, ya sea que las regularidades del conjunto de datos se consideran como un todo o las secuencias de variación de las longitudes que son relativamente comunes. Poniéndolo en palabras simples, el propósito de la compresión de datos ***consiste en encontrar transformaciones algorítmicas de representaciones de datos que producirán representaciones más compactas debido a los conjuntos de datos "típicos".***

Introducción - Entropía

En el ámbito de la teoría de la información la **entropía**, también llamada **entropía de la información** y **entropía de Shannon** (en honor a Claude E. Shannon), **mide la incertidumbre de una fuente de información**.

La entropía también **se puede considerar como la cantidad de información promedio que contienen los símbolos usados**. ***Los símbolos con menor probabilidad son los que aportan***

mayor información; por ejemplo, si se considera como sistema de símbolos a las palabras en un texto, palabras frecuentes como «que», «el», «a» aportan poca información, mientras que palabras menos frecuentes como «corren», «niño», «perro» aportan más información. Si de un texto dado borramos un «que», seguramente no afectará a la comprensión y se sobreentenderá, no siendo así si borramos la palabra «niño» del mismo texto original. Cuando todos los símbolos son igualmente probables (distribución de probabilidad plana), todos aportan información relevante y la entropía es máxima.
El concepto entropía es usado en termodinámica, mecánica estadística y teoría de la información. En todos los casos la entropía se concibe como una **«medida del desorden»** o la **«peculiaridad de ciertas combinaciones»**.
La entropía puede ser considerada como una medida de la incertidumbre y de la información necesaria para, en cualquier proceso, poder acotar, reducir o eliminar la incertidumbre. Resulta que el concepto de información y el de entropía están básicamente relacionados entre sí, aunque se necesitaron años de desarrollo de la mecánica estadística y de la teoría de la información antes de que esto fuera percibido.


Claude Elwood Shannon (30 de abril de 1916 – 24 de febrero de 2001)

La Teoría de la Información es la disciplina que se encarga del estudio y cuantificación de los procesos que se realizan sobre la información. La cantidad de información que nos proporciona cierto dato es menor cuanto más esperamos ese dato. La idea de la probabilidad de ocurrencia de cierto evento que nos da información que puede modelarse por una variable aleatoria y su conjunto de mensajes y probabilidades asociadas. Al recibir un mensaje de esa variable aleatoria *(fuente de información),* se obtiene una cantidad de información, que depende sólo de la probabilidad de emisión de ese mensaje. La cantidad de información es una función creciente e inversa a la probabilidad, es decir, un menaje con menor probabilidad proporciona mayor cantidad de información. En 1948, **Claude Shannon** propuso una medida para la información. Si se supone una fuente F de información de n mensajes Fi, cada uno con probabilidad pi, la información al recibir un mensaje esta dado por la ecuación:

$$\mathrm{I}(F = F_i) = -log(p_i)$$

La medida media de información o entropía de F esta dada por la esperanza matemática de la información de cada símbolo

$$\mathrm{H}(F) = \sum_{i=1}^{n} p_i * (F = F_i) = -\sum_{i=1}^{n} p_i * log(p_i)$$

Al utilizar logaritmo en base 2, la medida estará en bits. Esta medida da una cota superior teórica de la cota de compresión que puede obtenerse de la información, no se puede tener ninguna codificación que consiga una longitud en bits media por símbolo emitido menor que la entropía de la fuente sobre la que se realiza la codificación. Los compresores actuales no llegan a esta cota pero quedan muy cerca.

Definición de Compresión

La **compresión de datos** es la reducción del volumen de datos tratables para representar una determinada información empleando una menor cantidad de espacio. Al acto de comprimir datos se denomina **compresión**, y al contrario **descompresión**.

El espacio que ocupa una información codificada *(datos, señal digital, etc.)* sin compresión es el cociente entre la frecuencia de muestreo y la resolución. Por tanto, cuantos más bits se empleen mayor será el tamaño del archivo. No obstante, la resolución viene impuesta por el sistema digital con que se trabaja y no se puede alterar el número de bits a voluntad; por ello, se utiliza la compresión, para transmitir la misma cantidad de información que ocuparía una gran resolución en un número inferior de bits.

La compresión ***es un caso particular de la codificación***, cuya característica principal es que el código resultante tiene menor tamaño que el original.

La compresión de datos se basa fundamentalmente en buscar repeticiones en series de datos para después almacenar solo el dato junto al número de veces que se repite. Así, por ejemplo, si en un fichero aparece una secuencia como "AAAAAA", ocupando 6 bytes se podría almacenar simplemente "6A" que ocupa solo 2 bytes, en algoritmo RLE.

El desempeño de los métodos de compresión se mide en base a dos criterios: la **razón de compresión** y el **factor de compresión**, siendo el segundo el inverso del primero. Entre mayor redundancia exista en los datos, mejor razón (factor) de compresión será obtenido.

$$Razón\ de\ compresión = \frac{Nro.\ Bytes\ Archivo\ comprimido}{Nro.\ Bytes\ Archivo\ original}$$

$$Factor\ de\ compresión = \frac{Nro.\ Bytes\ Archivo\ original}{Nro.\ Bytes\ Archivo\ comprimido}$$

Conceptos generales

A la hora de hablar de compresión hay que tener presentes dos conceptos:

Redundancia: Datos que son repetitivos o previsibles

Entropía: La información nueva o esencial que se define como la diferencia entre la cantidad total de datos de un mensaje y su redundancia.

La información que transmiten los datos puede ser de tres tipos:

Redundante: información repetitiva o predecible.

Irrelevante: información que no podemos apreciar y cuya eliminación por tanto no afecta al contenido del mensaje. Por ejemplo, si las frecuencias que es capaz de captar el oído humano están entre 16/20 Hz y 16.000/20.000 Hz, serían irrelevantes aquellas frecuencias que estuvieran por debajo o por encima de estos valores.

Básica: la relevante. La que no es ni redundante ni irrelevante. La que debe ser transmitida para que se pueda reconstruir la señal.

Teniendo en cuenta estos tres tipos de información, se establecen tres tipologías de compresión de la información:

Sin pérdidas reales: es decir, transmitiendo toda la entropía del mensaje (toda la información básica e irrelevante, pero eliminando la redundante).

Subjetivamente sin pérdidas: es decir, además de eliminar la información redundante se elimina también la irrelevante.

Subjetivamente con pérdidas: se elimina cierta cantidad de información básica, por lo que el mensaje se reconstruirá con errores perceptibles pero tolerables (por ejemplo: la videoconferencia).
Desafíos para la compresión de Datos:

- Que el proceso de compresión-transmisión-descompresión sea más rápido que el proceso de transmisión sin compresión.
- Que sea comprimir la mayor cantidad de información. Cota teórica de Shannon.
- Que el espacio de memoria sea lo mínimo posible.
- Que la perdida de información sea nula.

Compresión de datos sin perdida

Se denomina **algoritmo de compresión sin pérdida** a cualquier procedimiento de codificación que tenga como objetivo representar cierta cantidad de información utilizando u ocupando un espacio menor, **siendo posible una reconstrucción exacta de los datos originales**. Es decir, la compresión sin pérdidas engloba a aquellas técnicas que garanticen generar un duplicado exacto del flujo de datos de entrada después de un ciclo de compresión / expansión. Por esta razón es utilizada para comprimir archivos que contienen datos que no pueden ser degradados o perdidos, como pueden ser documentos de texto, imágenes y sonido.
Se fundamenta en conceptos de la Teoría de la Información, como la **Redundancia** y **Entropía** de los datos y es generalmente implementada usando uno o dos tipos de modelos diferentes: el estadístico y aquel basado en diccionario.

- Estadísticos
    - Codificación RLE
    - Shanon-Fano
    - Huffman
    - Aritméticos
    - Predictivos
- Basados en Diccionarios o sustitucionales.

Estadísticos:

Los métodos de compresión estadísticos usan las propiedades estadísticas de los datos que van a comprimirse para asignar códigos de longitud variable a los símbolos individuales en los datos. La diferencia entre los distintos métodos es la forma en que se obtienen las probabilidades de los símbolos.
Dadas las probabilidades de los símbolos, la codificación se encarga de convertir dichas probabilidades en una cadena de bits para obtener los datos en forma comprimida.

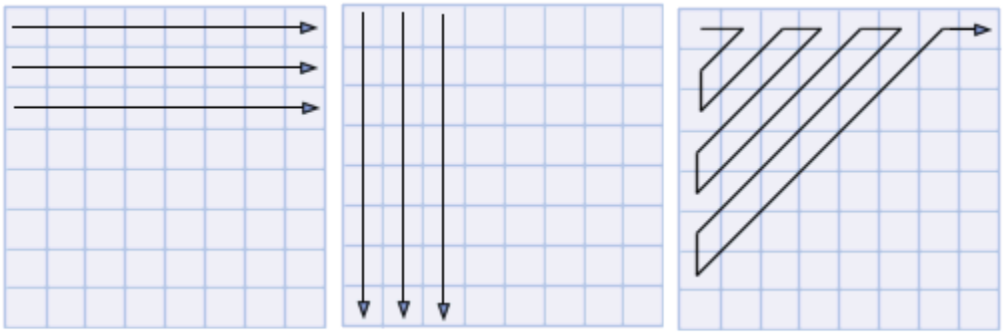
Codificación RLE

La compresión RLE o **Run-length encoding** es una forma muy simple de compresión de datos en la que secuencias de datos con el mismo valor consecutivas son almacenadas como un único valor más su recuento. Esto es más útil en datos que contienen muchas de estas "secuencias"; por ejemplo, gráficos sencillos con áreas de color plano, como iconos y logotipos.
Por ejemplo, considera una pantalla que contiene texto en negro sobre un fondo blanco. Habría muchas secuencias de este tipo con píxeles blancos en los márgenes vacíos, y otras secuencias

de píxeles negros en la zona del texto. Supongamos una única línea, con N representando las zonas en negro y B las de blanco:
BBBBBBBBBBBNBBBBBBBBBBBNNNBBBBBBBBBBBBBBBBBBBBBNBBBBBBBBBBBBBB
Si aplicamos la codificación run-length a esta línea, obtendríamos lo siguiente:
12B1N12B3N24B1N14B
Interpretado esto como 12 letras B, 1 letra N , 12 letras B, 3 letras N, etc. El código run-length representa el original de 67 caracteres en tan sólo 16.
Algunos formatos que utilizan esta codificación incluyen Packbits, PCX e ILBM.

En realidad, la compresión RLE está regida por reglas particulares que permiten que se ejecute la compresión cuando sea necesario y que se deje la cadena como está cuando la compresión genere pérdida. Las reglas son las siguientes:
* Si se repiten tres o más elementos consecutivamente, se utiliza el método de compresión RLE.
* De lo contrario, se inserta un carácter de control (00) seguido del número de elementos de la cadena no comprimida y después la última.
* Si el número de elementos de la cadena es extraño, se agrega el carácter de control (00) al final.
* Finalmente, se definen los caracteres de control específicos según el código:
    un final de línea (00 01)
    el final de la imagen (00 00)
    un desplazamiento de puntero sobre la imagen de XX columnas e YY filas en la dirección de lectura (00 02 XX YY).

Por lo tanto, no tiene sentido utilizar la compresión RLE excepto para datos con diversos elementos repetidos de forma consecutiva, en imágenes particulares con áreas grandes y uniformes. Sin embargo, la ventaja de este método es que es de fácil implementación. Existen alternativas en las que la imagen está codificada en bloques de píxeles, en filas o incluso en zigzag.



Algoritmo de Shannon Fano


Robert Fano

El algoritmo de Shannon-Fano construye un código sin prefijo basado en un conjunto de símbolos y sus probabilidades. En los códigos sin prefijos, los códigos de los caracteres pueden tener longitudes diferentes ya que ningún código de carácter es prefijo de otro código. Estos algoritmos generalmente usan árboles para su construcción, y almacenan caracteres del alfabeto solo en las hojas, ya que de lo contrario no se puede garantizar una decodificación que no sea ambigua.
El algoritmo de Shannon-Fano es un algoritmo anterior al de Huffman, que veremos a continuación. Su autor es **Claude Elwood Shannon**, y lo publico en 1947 en "A Mathematical Theory of Communication".
El método de compresión se atribuye a **Robert Fano**. En la codificación Shannon-Fano los símbolos se ordenan por probabilidad, del más probable al menos probable. Se dividen en 2 conjuntos cuya suma de probabilidades sean tan iguales como sea posible. A los símbolos del primer conjunto se les asigna 0 como primer dígito, mientras que a los del segundo se les asigna 1. Este proceso continúa hasta que cada conjunto tiene solo 1 elemento. Este algoritmo produce codificaciones de longitud variable bastante eficientes.
El caso óptimo se da cuando los 2 subconjuntos producidos al particionar tienen la misma probabilidad. Sin embargo, en la mayoría de los casos no produce códigos óptimos. Por ejemplo, para el conjunto {0.35, 0.17, 0.17, 0.16, 0.15} se asignan códigos no óptimos. Al no garantizar códigos óptimos, este algoritmo no se usa casi nunca, aunque el método de compresión IMPLODE del formato ZIP lo usa.
Asumiendo que tenemos una lista de símbolos con sus frecuencias de aparición, el algoritmo de Shannon Fano consiste en:
1 Ordenar los símbolos de acuerdo a su frecuencia
2 Dividir la lista en 2 mitades
3 Suma de frecuencias de cada mitad debe ser lo mas cercanas posibles
4 Asignar el digito 0 a la mitad izquierda, y el dígito 1 a la mitad derecha
5 Aplicar recursivamente los pasos 2 y 3 hasta que cada símbolo es una hoja del árbol

Codificación de Huffman

La codificación de Huffman es un método propuesto en 1952 por David Huffman. La idea de la codificación de Huffman es asignar códigos binarios lo más cortos posibles a los símbolos que ocurren con mayor frecuencia en los datos. Los símbolos con poca frecuencia tendrán asignado códigos binarios de longitud más grande. El óptimo desempeño del algoritmo se consigue cuando el número de bits asignado a cada carácter es proporcional al logaritmo de la probabilidad de mismo, se diferencia de Shannon-Fano en como construye el árbol.

Estadísticos-Aritméticos

La codificación aritmética es una forma de codificación entrópica utilizado en compresión sin pérdidas. Normalmente, una cadena de caracteres como las palabras "hola allí" está representada utilizando un número fijo de bits por carácter, como en el código ASCII. Cuándo una cadena es convertida a codificación aritmética, los caracteres frecuentemente usados serán

almacenados con menos bits y los no-tan-frecuentemente utilizados caracteres serán almacenados con más bits, resultando en menos bits utilizados en total. La codificación aritmética difiere de otras formas de codificación entrópica, como la codificación de Huffman, en que más que separar la entrada a símbolos componentes y reemplazar cada uno con un código, la codificación aritmética codifica el mensaje entero a un solo número, una fracción n dónde [0.0 ≤ "n" < 1.0).
En el caso más simple, la probabilidad de aparición de cada símbolo es igual. Por ejemplo, considere un conjunto de tres símbolos A, B y C, cada uno con la misma probabilidad de ocurrir. Un simple código de bloque requeriría 2 bits por símbolo, lo que es un desperdicio: una de las variaciones de bits nunca es usada. Es decir, A=00, B=01 y C=10, pero 11 no es usado. Una solución más eficiente es representar una secuencia de estos tres símbolos como un número racional en base 3 donde cada dígito representa un símbolo. Por ejemplo, la sequencia "ABBCAB" podría convertirse en 0.011201_3 (en la codificación aritmética los números están entre 0 y 1). El paso siguiente es codificar este número ternario usando un número binario de punto fijo con la suficiente precisión para recuperarlo, tal como 0.0010110010_2 — esto es sólo 10 bits; 2 bits son salvados en comparación con la codificación por bloque. Esto es factible para secuencias largas porque hay algoritmos eficientes para convertir la base de números precisos arbitrariamente.
Para decodificar el valor, conociendo que la cadena original tenía longitud 6, uno puede simplemente convertir de vuelta a base 3, redondear a 6 dígitos y recuperar la cadena.

Predictivos

Procuran predecir el siguiente mensaje de la entrada tomando como base de conocimiento la entrada procesada hasta ese momento. Si el mensaje que se encuentra a la entrada coincide con el predicho, su codificación se podrá hacer con menos bits. Sino, su codificación se hará con mas bits.

Basados en diccionario

Su idea es construir un diccionario tomando como referencia la entrada procesada hasta ese momento. El diccionario contiene cadenas de mensajes. Estas cadenas están identificadas por un índice, de manera que índice y cadena son de correspondencia.
Si una cadena dentro de la información se encuentra en el diccionario el compresor solo coloca el índice de la cadena dentro del diccionario y así en vez de poner toda la cadena solo se coloca un índice, esto significa un ahorro, tomando en cuenta que la cadena puede ser muy larga.

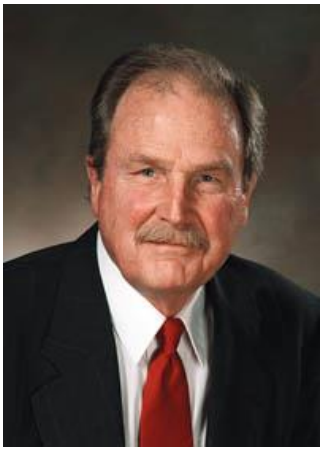
Compresión LZW o Lempel-Ziv-Welch

Los métodos de compresión Lempel-Ziv o LZ son los algoritmos más populares, ampliamente usados en la actualidad. Utilizan un modelo de compresión basado en tablas, donde las entradas de la tabla son remplazadas por cadenas de datos repetidos. Dicha tabla se genera dinámicamente.

De izq. a der.: Abraham Lempel, Jacob Ziv & Terry Welch

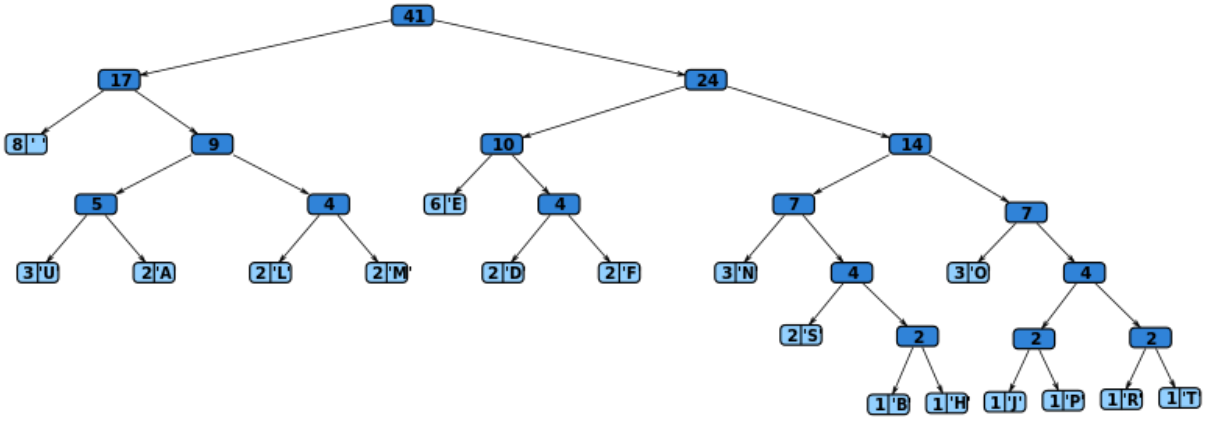
DEFLATE es una variación de LZ optimizada para velocidad de descompresión y radio de compresión, usada en PKZIP, gzip, PNG. LZW (LZ-Welch) se usa en imágenes GIF, LZR (LZ-Renau) es la base del método ZIP, y LZX se usa en el formato CAB de Microsoft.

Codificación Huffman



En ciencias de la computación y teoría de la información, la **codificación Huffman** es un algoritmo usado para compresión de datos. El término se refiere al uso de una tabla de códigos de longitud variable para codificar un determinado símbolo *(como puede ser un carácter en un archivo)*, donde la tabla ha sido rellenada de una manera específica basándose en la probabilidad estimada de aparición de cada posible valor de dicho símbolo. Fue desarrollado por **David A. Huffman** mientras era estudiante de doctorado en el MIT, y publicado en ***"A Method for the Construction of Minimum-Redundancy Codes"***.

La **codificación Huffman** usa un método específico para elegir la representación de cada símbolo, que da lugar a un código prefijo *(es decir, la cadena de bits que representa a un símbolo en particular nunca es prefijo de la cadena de bits de un símbolo distinto)* que representa los caracteres más comunes usando las cadenas de bits más cortas, y viceversa. Huffman fue capaz de diseñar el método de compresión más eficiente de este tipo: **ninguna representación alternativa de un conjunto de símbolos de entrada produce una salida media más pequeña** cuando las frecuencias de los símbolos coinciden con las usadas para crear el código. Posteriormente se encontró un método para llevar esto a cabo en un tiempo lineal si las probabilidades de los símbolos de entrada *(también conocidas como "pesos")* están ordenadas.



Para un grupo de símbolos con una distribución de **probabilidad uniforme** y un número de miembros que es potencia de dos, la **codificación Huffman** es equivalente a una codificación en bloque binaria, por ejemplo, la codificación ASCII. Esta codificación es un método para crear códigos prefijo tan extendido que el término **"codificación Huffman"** es ampliamente usado como sinónimo de **"código prefijo",** incluso cuando dicho código no se ha producido con el algoritmo de Huffman.
Aunque la codificación de Huffman es óptima para una codificación símbolo a símbolo dada una distribución de probabilidad, su optimalidad a veces puede verse accidentalmente exagerada.

Por ejemplo, la **codificación aritmética** y la codificación **LZW** normalmente ofrecen mayor capacidad de compresión. Estos dos métodos pueden agrupar un número arbitrario de símbolos para una codificación más eficiente, y en general se adaptan a las estadísticas de entrada reales. Este último es útil cuando las probabilidades no se conocen de forma precisa o varían significativamente dentro del flujo de datos.

Definición informal

Dados: Un **conjunto de símbolos** y **sus pesos** *(normalmente proporcionales a probabilidades)*.
Encontrar: Un código binario prefijo *(un conjunto de elementos del código)* con longitud de palabra esperada mínima *(de forma equivalente, un árbol con longitud del camino mínima)*.

Definición Formal

Entradas:

- El Alfabeto $A = \{a_1, a_2, \dots, a_n\}$, que es un alfabeto de tamaño n.
- El conjunto W= $\{w_1, w_2, \dots, w_n\}$, que es el conjunto de pesos(positivos) de los símbolos(normalmente proporcionales a probabilidades), es decir $w_i = peso(a_i), 1 \leq i \leq n$

Salida:

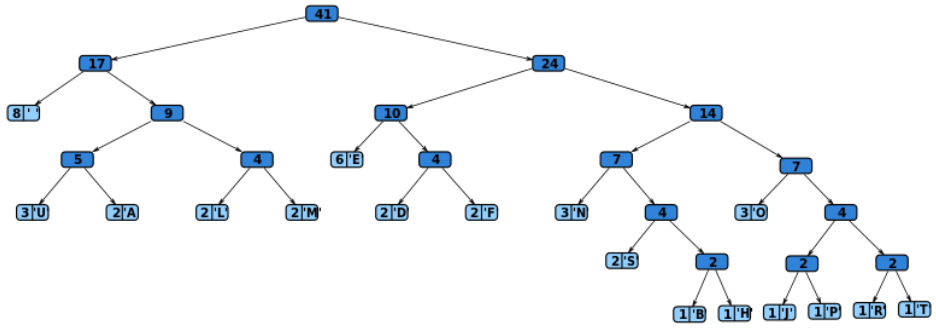
- El código $\mathrm{C}(A, W) = \{c_1, c_2, \dots, c_n\}$, que es el conjunto de elementos del código (binario) donde c_i es la palabra del código para$a_i, 1 \leq i \leq n$.

Objetivo:

- Sea $\mathrm{L}(C) = \sum_{i=1}^{n} \left(w_i . \ longitud(c_i) \right)$ la longitud del camino ponderado del código C. Condición $\mathrm{L}(C) \leq \mathrm{L}(T)$ para cualquier código $\mathrm{T}(A, W)$.

Técnica

La técnica utilizada es el propio **algoritmo de Huffman**. Consiste en la creación de un árbol binario en el que se etiquetan los nodos hoja con los caracteres, junto a sus frecuencias, y de forma consecutiva se van uniendo cada pareja de nodos que menos frecuencia sumen, pasando a crear un nuevo nodo intermedio etiquetado con dicha suma. Se procede a realizar esta acción hasta que no quedan nodos hoja por unir a ningún nodo superior , y se ha formado el árbol binario.
Posteriormente se etiquetan las aristas que unen cada uno de los nodos con ceros y unos (hijo derecho e izquierdo, respectivamente, por ejemplo). El código resultante para cada carácter es la lectura, siguiendo la rama, desde la raíz hacia cada carácter (o viceversa) de cada una de las etiquetas de las aristas.



Carácter	Frecuencia	Código
Espacio	8	00
E	6	100
N	3	1100
O	3	1110
U	3	0100
A	2	0101
D	2	1010
F	2	1011
L	2	0110
M	2	0111
S	2	11010
B	1	110110
H	1	110111
J	1	111100
P	1	111101
R	1	111110
T	1	111111

Árbol de Huffman generado para las frecuencias de apariciones exactas del texto "ESTO ES UN EJEMPLO DE UN ARBOL DE HUFFMAN". las frecuencias y códigos de cada carácter se muestran abajo. Codificar esta frase usando este código requiere 156 bits, sin contar con el espacio para el árbol.

Construcción del árbol

Para obtener los códigos de Huffman hay que construir un árbol binario de nodos, a partir de una lista de nodos, cuyo tamaño depende del número de símbolos, **n**. Los nodos contienen dos campos, el **símbolo** y el **peso** *(frecuencia de aparición)*.
Cada nodo del árbol puede ser o bien un nodo hoja o un nodo interno. Inicialmente se considera que todos los nodos de la lista inicial son nodos hoja del árbol. Al ir construyendo el árbol, los nodos internos tendrán un peso y dos nodos hijos *(opcionalmente un enlace al nodo padre que puede servir para recorrer el árbol en ambas direcciones)*. Por convención el bit '0' se asocia a la rama izquierda y el bit '1' a la derecha. Una vez finalizado el árbol contendrá **n** nodos hijo y **n-1** nodos internos.
El proceso de construcción del árbol comienza formando un nodo intermedio que agrupa a los dos nodos hoja que tienen menor peso *(frecuencia de aparición)*. El nuevo nodo intermedio tendrá como nodos hijos a éstos dos nodos hoja y su campo peso será igual a la suma de los pesos de los nodos hijos. Los dos nodos hijos se eliminan de la lista de nodos, sustituyéndolos por el nuevo nodo intermedio. El proceso se repite hasta que sólo quede un nodo en la lista. Éste último nodo se convierte en el **nodo raíz** del **árbol de Huffman**.

El algoritmo de construcción del árbol puede resumirse así:

- Crear un nodo hoja para cada símbolo, asociando un peso según su frecuencia de aparición e insertarlo en la lista ***ordenada ascendentemente***.
- Mientras haya más de un nodo en la lista:
  - Eliminar los dos nodos con menor probabilidad de la lista
  - Crear un nuevo nodo interno que enlace a los nodos anteriores, asignándole como peso la suma de los pesos de los nodos hijos.
  - Insertar el nuevo nodo en la lista, (en el lugar que le corresponda según el peso).

- El nodo que quede es el nodo raíz del árbol.

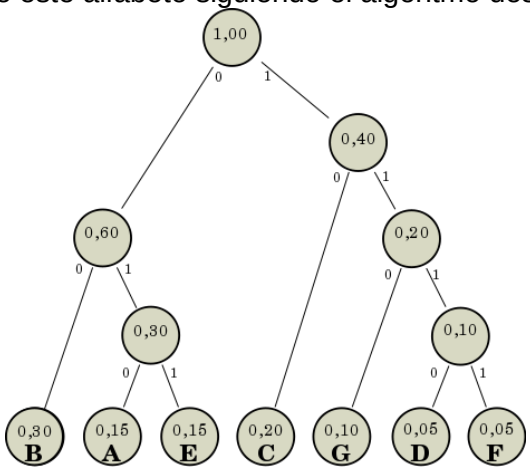
Propiedades

Las probabilidades usadas pueden ser genéricas para el dominio de la aplicación, que están basadas en el caso promedio, o pueden ser las frecuencias reales encontradas en el texto que se está comprimiendo. *(Esta variación requiere que la tabla de frecuencias u otra estructura utilizada para la codificación deben ser almacenadas con el texto comprimido; las implementaciones emplean varios mecanismos para almacenar tablas de manera eficiente).*
La **codificación Huffman** es óptima cuando la probabilidad de cada símbolo de entrada es una potencia negativa de dos. Los códigos prefijos tienden a ser ligeramente ineficientes en alfabetos pequeños, donde las probabilidades normalmente se encuentran entre esos puntos óptimos. El "empaquetado", o expansión del tamaño del alfabeto concatenando múltiples símbolos en "palabras" de tamaño fijo o variable antes de la codificación Huffman, normalmente ayuda, especialmente cuando símbolos adyacentes están correlacionados *(como en el caso de un texto en lenguaje natural)*. El peor caso para una codificación Huffman puede darse cuando la probabilidad de un símbolo excede 2-1 = 0.5, haciendo el límite superior de ineficiencia ilimitado. Estas situaciones a menudo responden bien a una forma de paquete llamada **codificación run-length**.
La codificación aritmética produce una ligera ganancia sobre la codificación Huffman, pero en la práctica esta ganancia raramente ha sido lo bastante grande como para utilizar la codificación aritmética que posee una complejidad computacional más elevada y además requiere el pago de royalties*. (A julio de 2006, IBM posee patentes de muchos métodos de codificación aritmética en varias jurisdicciones).*

Ejemplo

La tabla describe el alfabeto a codificar, junto con las frecuencias de sus símbolos. En el gráfico se muestra el árbol construido a partir de este alfabeto siguiendo el algoritmo descrito.

Símbolo	Frecuencia
A	0,15
B	0,30
C	0,20
D	0,05
E	0,15
F	0,05
G	0,10

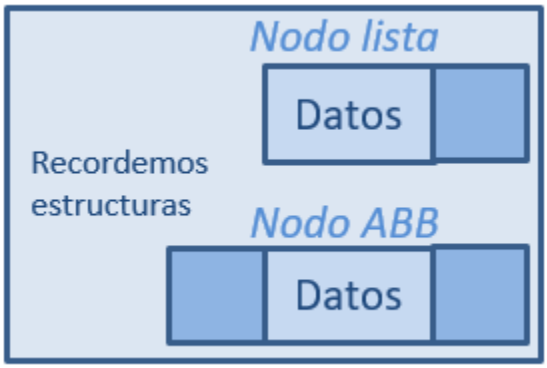


Se puede ver con facilidad cuál es el código del símbolo E: subiendo por el árbol se recorren ramas etiquetadas con 1, 1 y 0; por lo tanto, el código es 011. Para obtener el código de D se recorren las ramas 0, 1, 1 y 1, por lo que el código es 1110.

La operación inversa también es fácil de realizar: dado el código 10 se recorren desde la raíz las ramas 1 y 0, obteniéndose el símbolo C. Para descodificar 010 se recorren las ramas 0, 1 y 0, obteniéndose el símbolo A.

Ejemplo de codificación *(paso a paso)*:

Usamos una estructura mixta, una lista de árboles.

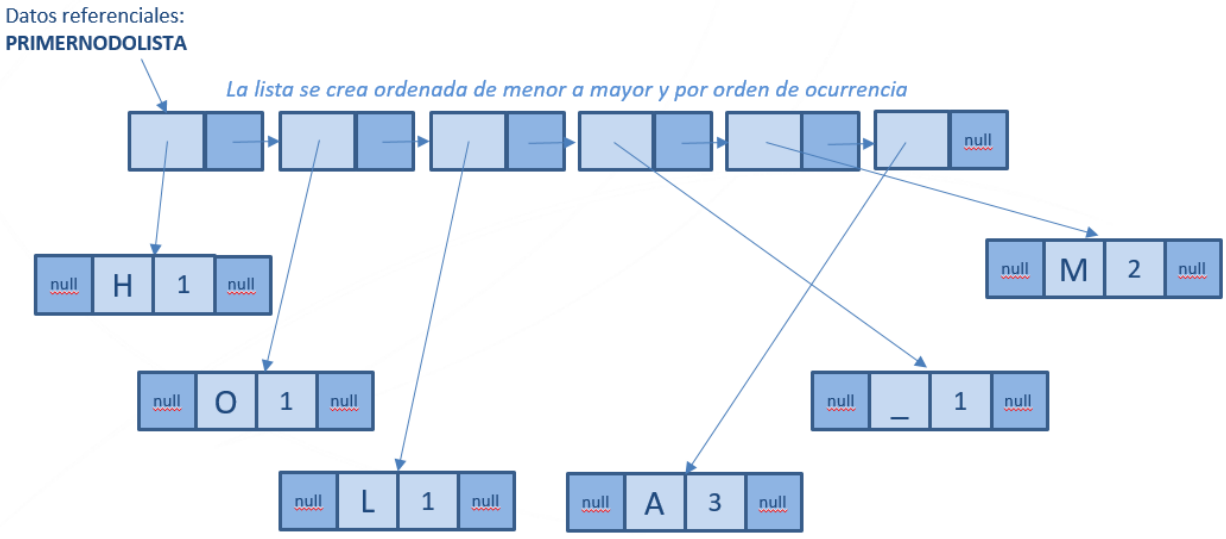


Mensaje: **HOLA MAMA**

Espacio

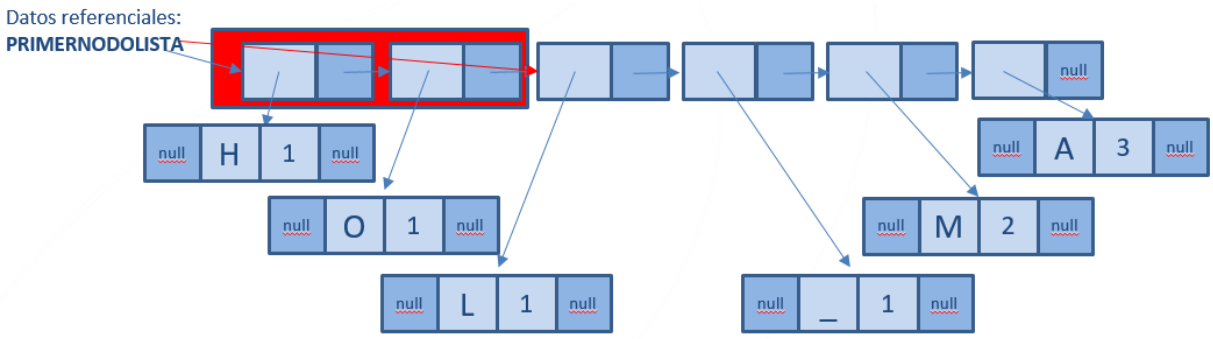
9 caracteres a codificar

1) Armamos una lista de nodos de árbol binario *(ordenada por ocurrencias de menor a mayor)*

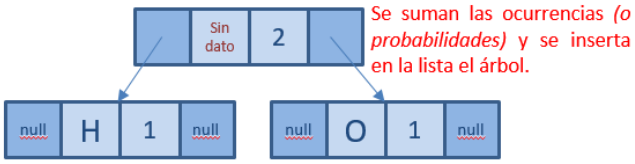


2) Iniciamos el proceso de armado de sub arboles de 2 en 2 *(primera iteración)*

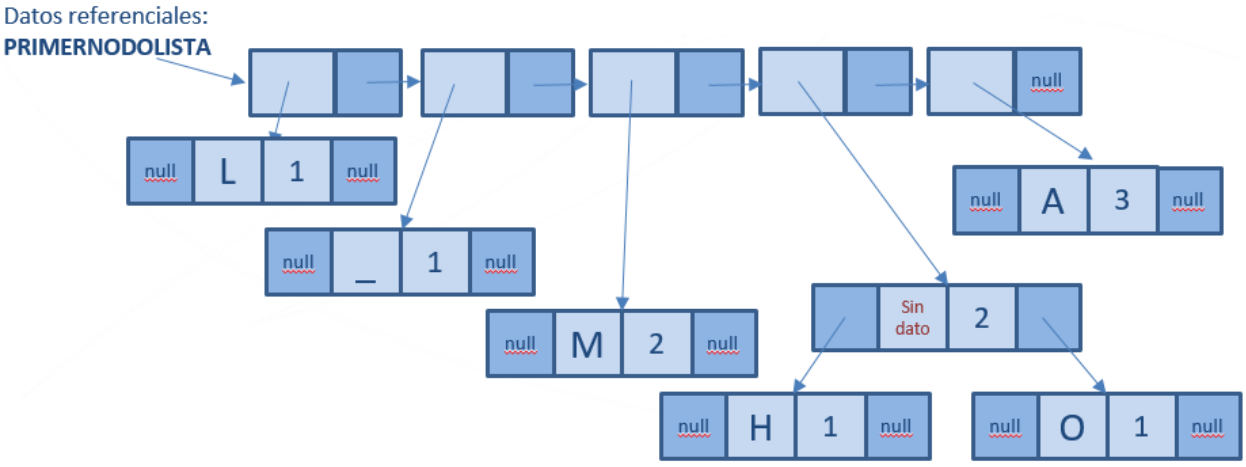
Eliminamos “momentáneamente” los 2 primeros nodos de la lista.



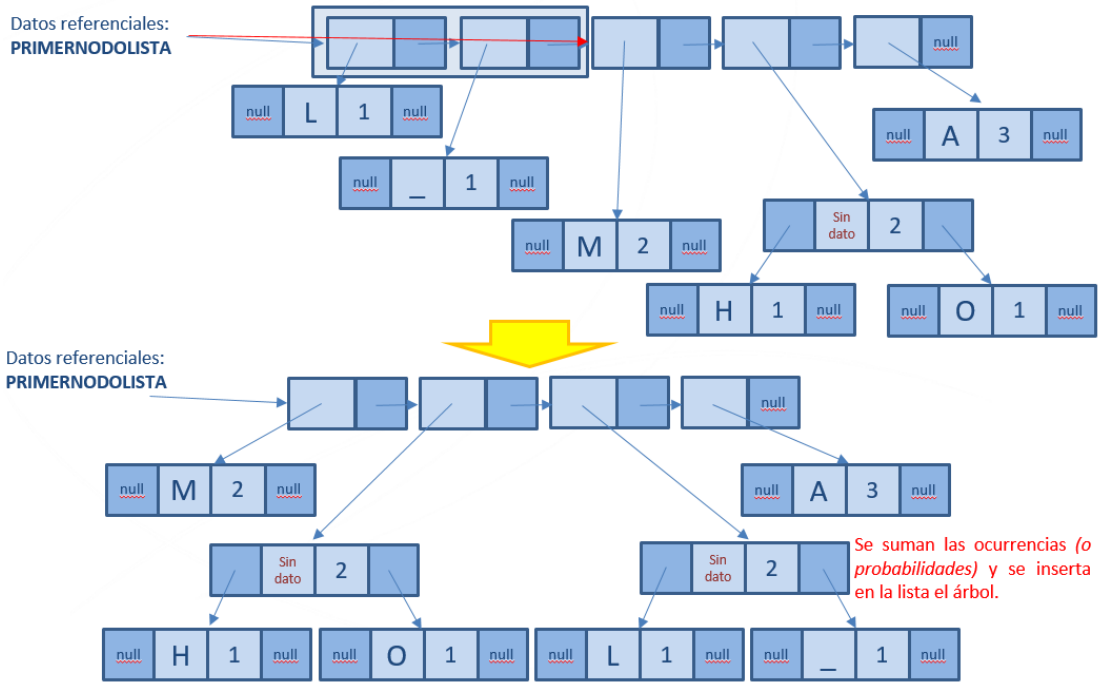
Decimos momentáneamente, porque le creamos un nodo padre a ambos sumándole las ocurrencias de los hijos



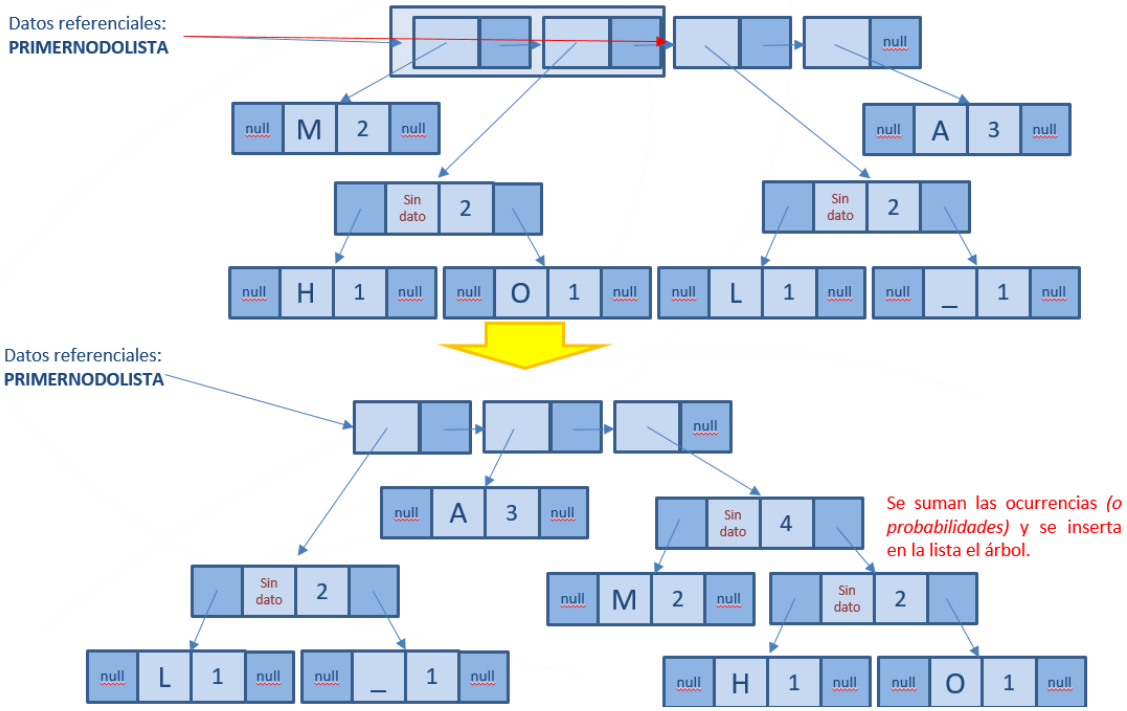
Y luego a ese nodo padre lo volvemos a insertar en la lista



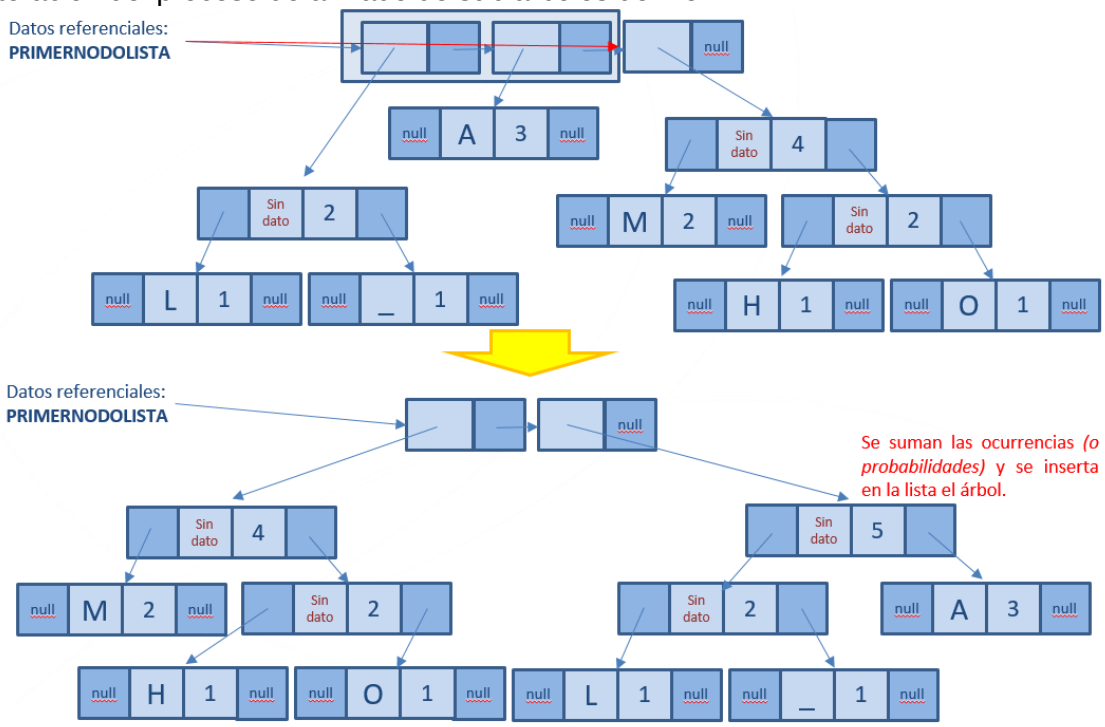
2da iteración del proceso de armado de sub arboles de 2 en 2
Volvemos a eliminar “momentáneamente” los 2 primeros nodos de la lista.



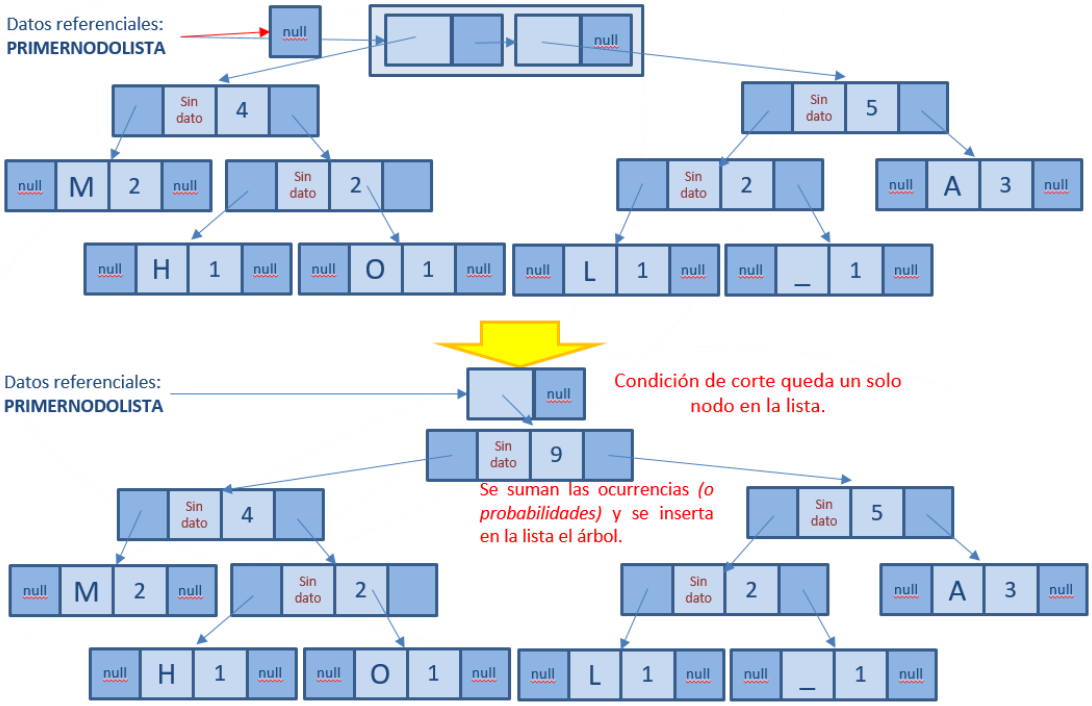
3er iteración del proceso de armado de sub arboles de 2 en 2



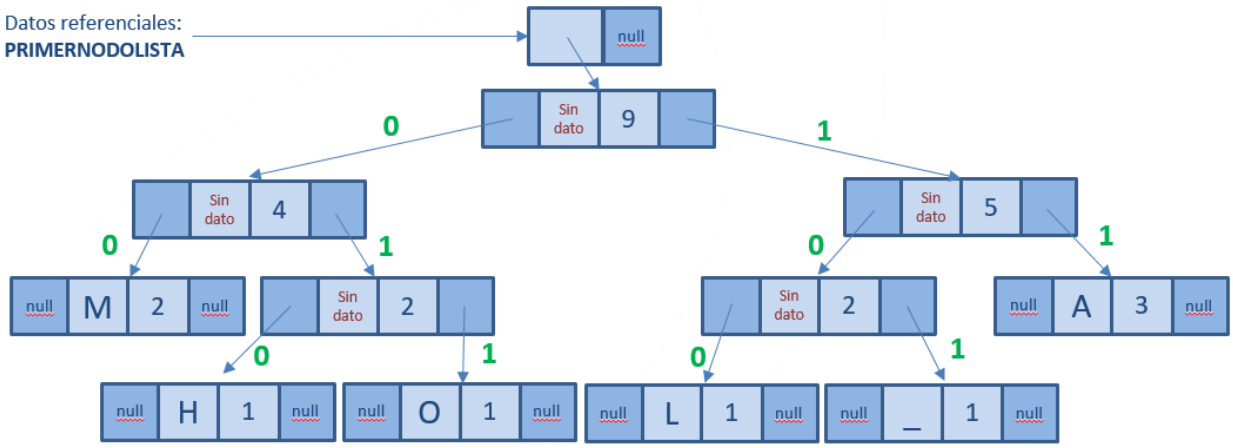
4ta iteración del proceso de armado de sub arboles de 2 en 2



5ta iteración del proceso de armado de sub arboles de 2 en 2



3) Etiquetamos aristas y armamos tabla



H	010	72	01001000
O	011	79	01001111
L	100	76	01001100
_	101	32	00100000
M	00	77	01001101
A	11	65	01000001

Mensaje ascii original: **HOLA MAMA**

Mensaje binario original
01001000 01001111 01001100 01000001 00100000 01001101 01000001 01001101 01000001

Mensaje binario compactado
010011100111010011 0

4) Generamos el archivo codificado *(comprimido)* a partir de la tabla. **COMPRESIÓN**

Mensaje ascii original: HOLA MAMA

Mensaje binario original

0100100001001111010011000100000100100000010011010100000101001101 01000001

Mensaje binario compactado Teórico

01001110011101001100 11

Mensaje binario compactado en forma práctica

01001110 01110100 11001100

BITS de relleno para completar el byte

5) Generamos el archivo original *(descomprimido)* a partir del archivo codificado *(comprimido)* usando la tabla. **DESCOMPRESIÓN**

NOTAS: ** No se puede descomprimir el archivo si no poseemos la tabla y/o el árbol.*
** Precisamos saber cuantos símbolos había en el archivo original (cuestiones de implementación).*
Se concatenan bits en una variable auxiliar hasta que sea una de las entradas de la tabla

Mensaje binario compactado

01001110011101001100 1100

Mensaje ascii original:

Mensaje binario original

0100100001001111010011000100000100100000010011010100000101001101010000 01

Si no hubiésemos contado los simbolos del archivo original, confundiríamos los bits de relleno con información a decodificar.

*Para implementar descompresión por **Huffman**, deberemos haber guardado previamente, tanto la información de la tabla con cantidad de ocurrencias y/o árbol, como la cantidad de símbolos que poseía el archivo sin comprimir. De lo contrario nos sería imposible descomprimir el mensaje.*

Huffman adaptable

El método original propuesto por Huffman, requiere **dos pasadas** sobre los datos para realizar la compresión, en la primera, se obtienen las estadísticas de los símbolos, en la segunda se realiza la compresión.
Entre ambos pasos se crea el árbol binario. El método suele ser lento debido a estas dos pasadas y por tanto*, poco atractivo en aplicaciones de compresión de datos en tiempo real*.
El método usado en la práctica es **Codificación de Huffman Adaptable o Dinámica**. En este método, el compresor y el descompresor inician con un árbol vacío y conforme se leen y procesan (comprimir/descomprimir) los símbolos, ambos modifica el árbol de la misma forma.
Inicialmente, el compresor lee cada símbolo de entrada y si es la primera vez que aparece, lo escribe tal cual al archivo de salida, lo agrega al árbol y le asigna un código de salida. Si el símbolo vuelve a ocurrir, el actual código asignado a él se escribe en el archivo de salida, su frecuencia se incrementa en uno y se actualiza el árbol para mantener la mejor codificación de acuerdo a las frecuencias de los símbolos. El descompresor refleja los mismos pasos que el compresor, cuando lee un símbolo no-comprimido, lo agrega al árbol y le asigna un código. Cuando lee un código comprimido, utiliza el árbol que ha formado hasta el momento para obtener el símbolo asociado. El descompresor también reorganiza el árbol de la misma forma que el compresor. El descompresor necesita saber distinguir entre códigos comprimidos y códigos no-comprimidos. Para ello, utiliza un símbolo especial de escape que puede variar a lo largo del proceso de descompresión.

Generalmente, el símbolo de escape corresponde al valor codificado de una hoja en el árbol con ocurrencia cero.
El árbol es verificado por cada símbolo de entrada. El símbolo con mayor frecuencia de ocurrencia debe parecer en los primeros niveles del árbol mientras que los símbolos menos frecuentes se encuentran el los últimos niveles. Sea **X** el nodo del símbolo actual de entrada, **F** su frecuencia de ocurrencia. Las operaciones para modificar el árbol se realizan en un ciclo comenzando en el nodo correspondiente al símbolo de entrada actual. Cada iteración se compone de tres operaciones:

1) Comparar **X** con sus sucesores, de izquierda a derecha y de abajo hacia arriba. Si el sucesor inmediato tiene frecuencia **F+1** o mayor, los nodos están aun en orden y no se realiza ningún cambio. De lo contrario, algunos sucesores tienen frecuencia igual o más pequeña que **X**. En este caso, **X** debe intercambiarse con el último de sus sucesores que probabilidad igual o menor excepto con su padre.
2) La frecuencia de **X** se incrementa en **1** y también la frecuencia de todos sus padres.
3) Si **X** es la raíz del árbol, el ciclo termina, de lo contrario, los pasos 1-3 se repiten ahora con el padre del nodo **X**.

Compresión de Datos con pérdida

Algoritmo de compresión con pérdida se refiere a cualquier procedimiento de codificación que tenga como objetivo representar cierta cantidad de información utilizando una menor cantidad de la misma, **siendo imposible una reconstrucción exacta de los datos originales**. Esto es porque, en lugar de guardar una copia exacta, sólo se guarda una **aproximación**. Esta aproximación se aprovecha de las limitaciones de la percepción humana para esconder la distorsión introducida.
Estos algoritmos son de gran utilidad para guardar imágenes fotográficas que de otra manera ocuparían mucho espacio dificultando su transmisión y almacenamiento. 1 Un ejemplo de algoritmo con pérdida de calidad es JPEG.

La compresión con pérdida sólo es útil cuando la reconstrucción exacta no es indispensable para que la información tenga sentido. La información reconstruida es solo una aproximación de la información original. Suele restringirse a información analógica que ha sido digitalizada (imágenes, audio, video, etc.), donde la información puede ser "parecida" y, al mismo tiempo, ser subjetivamente la misma. Su mayor ventaja reside en las altas razones de compresión que ofrece en contraposición a un algoritmo de compresión sin pérdida.
La compresión con pérdida acepta una pérdida de datos para poder mejorar el factor de compresión. Se aplica generalmente al almacenamiento digital de datos analógicos como archivos de gráficos y de sonidos. La gran ventaja de comprensión con pérdida es alcanzar una tasa de compresión más elevadas a costa de sufrir una pérdida de información sobre la imagen original.