CoNaIISI 2018
6to Congreso Nacional de Ingeniería Informática - Sistema de Información

Categoría
Docentes - Investigadores - Graduados

29 y 30 de Noviembre
Mar del Plata - Argentina

Trabajo N°: 598

Compactación y Codificación mediante códigos Huffman y Hamming

El presente trabajo detalla el funcionamiento del programa Huffman-Hamming junto con una pequeña introducción teórica al tema, el mismo fue desarrollado como práctico final de máquina de la materia Teoría de la Información y Comunicación perteneciente al 4° año de la carrera Ingeniería en informática. La dirección del trabajo fue efectuada por el Dr. German Montejano y el Lic. Mario A. Silvestri.

Introducción

En el esquema del envió de información existen tres componentes básicas, un emisor un receptor y el canal por el cual son enviados. El canal por diversos factores (cables en mal estado, distancias muy largas para la capacidad del cable, etc.) puede presentar errores en la información enviada, por otro lado, el tiempo que se tarda en enviar a información es otro problema presente en el esquema de comunicación. Existen varias soluciones para estos problemas, pero las explicadas en el siguiente informe son el código Hamming (para solucionar errores en la información por causa del canal) y código Huffman (para disminuir el tiempo de envió de la información).

Código Hamming

El código de Hamming llamado así por su inventor Richard Hamming es un código detector y corrector de errores. El funcionamiento básico de esta codificación es el siguiente: dado una cadena de bits (parte de la información) se le agregan ciertos bits en posiciones específicas (posiciones 2^n) que luego cuando la información llega al receptor mediante la operación de **sumas de paridad** es posible detectar si la cadena presenta un error y qué bit posee el error, de esta manera se puede corregir. En la siguiente tabla se muestra la representación binaria de los números del 1 al 9, cada número representa una posición que ocupa un bit en la cadena de información.

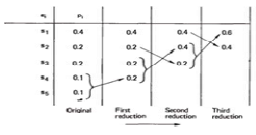
Posición	Representación Binaria
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010

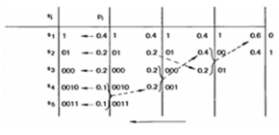
Todo número que tenga un 1 en su última posición de su representación binaria deberá ir en la primera suma de paridad, de forma similar los que tengan un 1 en su penúltima posición de su representación binaria deberá ir en la segunda suma de paridad y así siguiendo. Por lo tanto, podemos ver que la primera suma de paridad cubre las posiciones 1,3,5,7,9,11,13,15...; la segunda, 2,3,6,7,10,11,14,15,18,19,22,23...; la tercera 4,5,6,7,12,13,14,15,20,21,22,23...; la cuarta 8,9,10,11,12,13,14,15,24,25,26,27,28,29,30,31...; etc. Tomando los resultados de la suma de paridad generamos un numero llamado síndrome (cara resultado de una suma conforma un digito del síndrome) el cual si tiene ceros en cada uno de sus dígitos indica que no existen errores en la cadena de bits, en otro caso indica la posición en representación binaria del error.

Para genera varios bits de control sobre la cadena de información se debe buscar una forma general, esto se consigue a través de una matriz generadora G, la cual es multiplicada con la cadena de información para así obtener la cadena codificada con Hamming. Luego, para decodificar, multiplicamos el vector codificado por una segunda matriz decodificadora H, obteniendo el vector síndrome, el que nos permite finalmente corregir la cadena y recuperar la original.

Código Huffman

Es un algoritmo usado para la compresión de datos, desarrollado por David A. Huffman. Esta codificación utiliza una tabla en donde se tiene los códigos que representa a los distintos símbolos de un texto (información), estos mismos varían en longitud dependiendo de la probabilidad de aparición de dicho símbolo. Utiliza un método que consiste en ordenar las frecuencias y luego tomar las dos (usamos radix 2 ya que trabajamos en binario) probabilidades más chicas y sumarlas y volvemos a ordenar las probabilidades y se vuelve a tomar las dos más chicas hasta que nos quedan solo dos como se puede ver en la figura 1 y luego se asigna un símbolo a cada una (0 ó 1), de esta forma se sigue el camino por el cual se llegó a dicha probabilidad como se aprecia en la figura 2. Utilizando este método se logra que el código que representa un símbolo nunca sea prefijo del código de otro.





Para la implementación de esta codificación utilizamos un árbol binario en donde cada uno de sus nodos que son hoja del árbol tiene un símbolo asociado, una frecuencia. Para crear este árbol primero se debe recorrer el archivo completo y calcular la frecuencia de aparición de cada carácter en el archivo lo que incurre un mayor tiempo de ejecución, pero genera una tabla de frecuencias óptima para cada texto.

Conclusiones

La herramienta desarrollada permite la correcta codificación y compactación lo que nos permite reducir los errores en el canal de envió de datos como el tiempo en que estos van a tardar en enviarse. Mejorar de forma general el envió de información Codificación Hamming: Para recolectar los siguientes datos se realizaron 30 pasadas de cada ejecución y luego se calculó un promedio de tiempo.

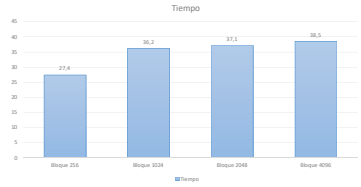
Funcionalidad de proteger información :

Bloque 256
- Tiempo de ejecución: 27.4 segundos
- Tamaño final del archivo codificado: 4.225.174 bytes

Bloque 1024
- Tiempo de ejecución: 36.2 segundos
- Tamaño final del archivo codificado: 4.120.901 bytes

Bloque 2048
- Tiempo de ejecución: 37.1 segundos
- Tamaño final del archivo codificado: 4.100.661 bytes

Bloque 4096
- Tiempo de ejecución: 38.5 segundos
- Tamaño final del archivo codificado: 4.089.613 bytes



Lo que podemos observar después de analizar los siguientes gráficos es que a medida que se aumenta el tamaño del bloque el tiempo de ejecución de la codificación aumenta, mientras que el tamaño del archivo codificado disminuye.

Funcionalidad decodificación de la información:

Bloque 256
- Tiempo de ejecución: 12.4 segundos

Bloque 1024
- Tiempo de ejecución: 18.5 segundos

Bloque 2048:
- Tiempo de ejecución: 20.2 segundos

Bloque 4096:
- Tiempo de ejecución: 22.9 segundos

Se observa que mientras el tamaño de bloque aumente el tiempo de ejecución para su decodificación también lo hace.

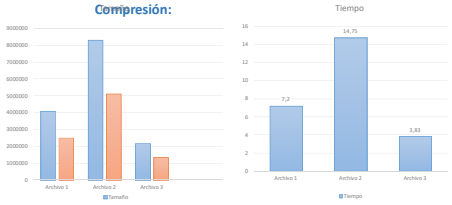
Compactación con Huffman:

Para realizar las ejecuciones se toman tres archivos de textos distintos, el primer archivo tiene un tamaño de 4.1 Megabytes (4.076.632 bytes), el segundo archivo tiene un tamaño de 8.3 Megabytes (8.292.953 bytes) y el ultimo archivo tiene un tamaño de 2.1 Megabytes (2.149.446 bytes).

Archivo 1 (4.076.632 bytes)
- Tiempo de ejecución: 7.20 segundos
- Tamaño 2.489.438 bytes
- Cantidad de caracteres distintos: 97

Archivo 2 (8.292.953 bytes)
- Tiempo de ejecución: 14,75 segundos
- Tamaño: 5.100.939 bytes
- Cantidad de caracteres distintos: 97

Archivo 3 (2.149.446 bytes)
- Tiempo de ejecución: 3.83 segundos
- Tamaño: 1.320.621 bytes
- Cantidad de caracteres distintos: 95



Se puede observar que en el archivo 1 la compresión total fue aproximadamente de un 39%, en el archivo 2 es de 38.5 % y con el archivo 3 la compresión fue de 38.6% por lo que no se puede apreciar una relación entre el tamaño del archivo y el porcentaje de compresión. El porcentaje de compresión está más relacionado con la cantidad de caracteres distintos que existen en el texto.

El archivo 2 es 50.9% más pesado que el archivo 1 y su tiempo de compresión es aproximadamente un 51,2% mayor. Por otro lado, comparando el archivo 1 contra el archivo 3 vemos que este último representa el 52.7 % del tamaño del archivo 1 y su tiempo de compresión es 53% más rápido respecto al archivo 1. Como se observa de estos resultados a medida que aumenta el tamaño de los archivos también lo hace el tiempo de compresión.

Universidad Nacional de San Luis, Facultad de Ciencias Físico-Matemáticas y Naturales

Autor/es:

Gonzalo Barroso Email: gonzalo.barroso94@gmail.com

Sebastian Riccardo Email: s.riccardo.barcia@gmail.com

Santiago Bordarampe Email: santiago.1422@gmail.com

ESCANEAR PARA CONTACTAR A LOS AUTORES