ST0254 – Organización de computadores
Práctica 2 y 3: Compresión de archivos

MBA, I.S. José Luis Montoya Pareja
Universidad EAFIT
Medellín, Colombia, Suramérica

RESUMEN
El paralelismo es clave a la hora de generar un mejor rendimiento en los programas. Como todo, tiene sus límites. Sin embargo, la computación de alto rendimiento (HPC) representa la capacidad de procesar datos y realizar cálculos complejos a velocidades muy altas; pudiendo realizar cuadrillones de cálculos por segundo.

PALABRAS CLAVE
Intel, Compresión, Paralelismo.

CONTEXTO

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". (Wikipedia, 2022)

Gracias a la continuidad de las imágenes y al hecho anteriormente mencionado con respecto a que la intensidad individual de los píxeles es muy difícil de ver, o distinguir para el ojo humano. Hay información sobreabundante, que puede comprimirse o eliminarse, reduciendo así de manera considerable la cantidad de espacio necesario de almacenamiento.

En otras palabras, la compresión de imágenes trata de minimizar el número de bits necesarios para representar una imagen. La idea de la compresión es muy sencilla - simplemente cambiamos la representación de bits de cada byte con la esperanza de que el nuevo diccionario produce una cadena más corta de bits necesarios para almacenar la imagen. A diferencia de la representación ASCII de cada byte, nuestro nuevo diccionario podría utilizar representaciones de longitud de bits variables. De hecho, en 1952, David Huffman (Huffman, 1952) hizo la siguiente observación:
En lugar de utilizar el mismo número de bits para representar cada carácter, ¿Por qué no utilizar una cadena de bits corta para los caracteres que aparecen con mayor frecuencia en una imagen y una cadena de bits más larga para los caracteres que aparecen con menor frecuencia en la imagen? (Lezama, 2017)

Huffman adaptativo
La codificación de Huffman adolece del hecho de que el descompresor necesita tener cierto conocimiento de las probabilidades de los caracteres en los archivos comprimidos. Esto no solo puede agregar algo a los bits necesarios para codificar el archivo, sino que, si este conocimiento crucial no está disponible, comprimir el archivo requiere dos pasadas: una para encontrar la frecuencia de cada carácter y construir el árbol Huffman y una segunda pasada para comprimir realmente el archivo. Ampliando el algoritmo de Huffman, Faller y Gallagher, y más tarde Knuth y Vitter, desarrollaron una forma de realizar el algoritmo de Huffman como un procedimiento de un solo paso. (Low, 2008)

En su concepción original, el código de Huffman necesita conocer la probabilidad que cada símbolo tiene de aparecer en un texto. La idea detrás del adaptativo es ir construyendo y recalculando el árbol de frecuencia de cada símbolo binario en función de los valores que van apareciendo en el texto a medida que son leídos. Esto hace que los códigos generados para cada símbolo se vayan adaptando a medida que se va leyendo el archivo.

Tanto la ejecución del programa de compresión como la de descompresión deben tener en cuenta que no conocen las probabilidades de cada símbolo ni el código a generar. Ambos deberán

construir el árbol a medida que se vayan ejecutando e irán reconstruyendo la información.

Algoritmo de Huffman
Sea $A = \{a_1, a_2, \dots, a_n\}$ un alfabeto de n símbolos que pueden estar en un texto.

El procedimiento original de Huffman requiere que se recorra el archivo a comprimir contando las ocurrencias de cada carácter del alfabeto definido. Una vez encontrada la frecuencia, se calcula la probabilidad de ocurrencia de cada carácter en función del total de caracteres que tiene el archivo a comprimir. Este procedimiento implicaba que se recorriera dos veces el archivo a comprimir, lo cual es altamente ineficiente (Orden $O(n^2)$).

Optimización de Huffman Adaptativo
Teniendo el mismo alfabeto A, se crea un árbol binario dinámicamente; donde cada nodo tiene tres elementos:

1. El símbolo que se debe codificar.
2. El peso (número de veces que cada elemento está en el texto). Se va llenando a medida que se encuentre una ocurrencia de cada símbolo del alfabeto.
3. Un identificador único por cada posible símbolo que se puede encontrar.

Características:

a. Si el alfabeto es de n elementos, la cantidad máxima de nodos que puede tener el árbol es $2n-1$
b. El primer nodo del árbol (la raíz) tiene un identificador igual a $2n-1$. Cada que se añade un nodo al árbol, se le asigna un valor desde $2n-1, 2n-2$ ... hasta que se acaban los símbolos encontrados en el texto o se llega a 1.
c. Sea x El peso (cantidad de símbolos encontrados) de cada nodo. Cada x_i de la serie cumple con la propiedad:
$$x_1 \leq x_2 \leq x_{2n-1}$$
d. Cada nodo del árbol cumple con la propiedad de *siblings* (hermanos):
Los nodos y_{2j-1} y y_{2j} son *siblings* para cada j en $1 \leq j < n$
e. Se establece un sistema de codificación en bits basado en la cantidad de elementos del alfabeto A. Se parte del supuesto que el otro extremo (que llamaremos en este caso, el descompresor), no sabe cuáles símbolos y en qué orden se han encontrado en el texto, por lo que la codificación deberá cubrir todas las posibilidades y se deberá colocar en el archivo comprimido para que el descompresor sepa como descomprimir el carácter encontrado. Se utiliza el esquema del *Not Yet Transmitted* (NYT).

   Sea e el número bits a usar en la codificación. Si siempre se diera el caso que $n = 2^e$, la codificación sería simple, se usan los n bits para codificar. Cuando no se cumple, se debe buscar un resto (residuo) que llamaremos r tal que cumpla que:
$$n = 2^e + r$$
$$0 \leq r < 2^e$$

   Se incluye un archivo en Excel llamado HuffmanAdaptativo.xlsx donde se puede calcular el esquema de codificación para n número de bits.

Para la práctica, se usará el siguiente alfabeto:
A ={Símbolos ASCI del 32 al 127, las vocales tildadas en mayúscula y minúscula y la letras Ñ y ñ}

En total son 107 símbolos.

PRÁCTICA 2

OBJETIVOS
1. Objetivo General
Construir el algoritmo y la implementación del Árbol Binario del Huffman Adaptativo.

2. Objetivos Específicos

a. Para el alfabeto especificado, genere un árbol binario de Huffman con las características que explicará el profesor en clase; para una entrada de texto a codificar escrita por teclado hasta que se presione la tecla [Enter].
b. El árbol que se genere se recorrer en infijo y debe mostrar en pantalla de la siguiente manera:
   [símbolo,peso,2n-1]
   [símbolo,peso,2n-2]
   …
   [símbolo,peso,m]

   Donde m es el número de símbolos encontrados tal que
$$1 \leq m < 2n-1$$
   Si m=1, es porque se encontraron todos los 107 símbolos.

c. Si hay un símbolo que no pertenece al alfabeto, se descarta el proceso completo y se genera un mensaje de error en pantalla diciendo “Símbolo encontrado genera error de compresión”.

CONSIDERACIONES GENERALES

1. La práctica se desarrollará en cualquier lenguaje de programación de su predilección, se sugiere usar alguno de la siguiente lista: C++, Java, Python, Haskell. Si usa otro lenguaje de programación, infórmeselo al profesor.
2. La aplicación debe correr en entorno Windows o Linux.
3. Se entregan los fuentes en un proyecto de GitHub. Indicar el nombre del proyecto y compartirlo con el usuario del profesor (jlmontoy).
4. El desarrollo de la práctica puede ser individual o en grupos de máximo tres personas.
5. Enviar al profesor a más tardar el jueves 27 de octubre a las 10:00 p.m. el nombre de las personas que conforman el grupo. Se debe informar vía TEAMS.
6. Criterios de evaluación:
    a. Entrega de los integrantes de la práctica a tiempo (5%)
    b. Entrega a tiempo de la práctica (5%): Se considera entregada cuando reporte en la tarea de Teams el nombre del repositorio que debe buscar el profesor y la versión de código definitiva.
    c. Cumplimiento del objetivo 2a (35%)
    d. Cumplimiento del objetivo 2b (30%)
    e. Cumplimiento del objetivo 2c (5%)
    f. Funcionamiento general de la práctica (10%)
    g. Video de sustentación de la práctica (10%): La sustentación será mediante un video de máximo 15 minutos (se citan los integrantes a una reunión de Teams y se graban explicando la práctica). Deben invitar al profesor a la sesión así el no asista a la grabación del video. Este debe incluir:
        i. Cada integrante del equipo presentará un pedazo del código (no hay que explicarlo todo).
        ii. Cada integrante lanzará una de las pruebas que dejará el profesor y mostrará los resultados en la pantalla.

FECHA DE ENTREGA

Viernes 4 de noviembre, hasta las 12 del día.