Facultad de Ingeniería
Universidad de Concepción

Departamento Ingeniería Informática y Ciencias de la Computación

Ingeniería Civil Informática

503220-1 ESTRUCTURA DE DATOS

Proyecto semestral: Codificar vs Comprimir

Integrantes	Matriculas
Diego Alejandro Gutiérrez Mendoza	2020405271
Matías Ignacio Urrea Moya	2020448591
Victor Hugo Andre Fernandez Veloso	2020433500

Prof.: José Fuentes Sepúlveda
Junio 2024

Facultad de Ingeniería
Universidad de Concepción

Introducción:

La codificación de Huffman es un algoritmo de compresión sin pérdida que asigna códigos de longitud variable a los símbolos basándose en sus frecuencias de aparición. Los símbolos más frecuentes reciben códigos más cortos, mientras que los menos frecuentes reciben códigos más largos. Este método se basa en la construcción de un árbol binario de Huffman, donde cada hoja representa un símbolo y su frecuencia. La codificación de Huffman garantiza que ningún código sea prefijo de otro, lo que permite una decodificación única y eficiente. La eficacia de este método radica en su capacidad para minimizar la longitud media de los códigos, logrando una representación más compacta del texto original.

Por otro lado, la compresión Lempel-Ziv (LZ) es un algoritmo de compresión sin pérdida que elimina redundancias mediante la identificación de patrones repetidos en el texto. En lugar de almacenar cada aparición de una subcadena repetida, LZ la reemplaza con un par de valores que indican la posición y longitud de la subcadena en una ventana de búsqueda. Esta técnica es la base de muchos formatos de compresión modernos, incluyendo ZIP y PNG. A diferencia de Huffman, LZ no requiere un análisis previo de frecuencias, lo que lo hace adecuado para compresiones en tiempo real.

El objetivo de este proyecto es doble. Primero, se implementarán ambos algoritmos en C++ y se verificará su correcto funcionamiento a través de métodos específicos para codificar y decodificar (en el caso de Huffman) y para comprimir y descomprimir (en el caso de LZ). Segundo, se realizará un estudio experimental para comparar el desempeño de ambos métodos. Se evaluará el tiempo requerido para codificar/comprimir y el tamaño resultante del texto procesado. Este análisis permitirá entender las ventajas y limitaciones de cada técnica en distintos escenarios.

Prof.: José Fuentes Sepúlveda
Junio 2024

Descripción de las técnicas usada e implementación propuesta:

Huffman Coding

La implementación del algoritmo de codificación y decodificación Huffman en los códigos proporcionados se centra en la compresión de datos basada en la frecuencia de los caracteres.

1. Codificación Huffman

Técnica Utilizada:

- **Árbol de Huffman**: Un árbol binario en el cual los caracteres con menor frecuencia tienen una mayor profundidad. Se utiliza una cola de prioridad para construir el árbol de Huffman.

Implementación Propuesta:

- Contar la frecuencia de cada carácter en la cadena de entrada.
- Construir una cola de prioridad (heap) con nodos que contienen caracteres y sus frecuencias.
- Combinar los nodos con menor frecuencia para construir el árbol de Huffman.
- Generar los códigos binarios para cada carácter mediante un recorrido del árbol.
- Codificar la cadena de entrada utilizando los códigos binarios generados.

2. Decodificación Huffman

Técnica Utilizada:

- **Árbol de Huffman**: Utilizar el árbol de Huffman construido durante la codificación para decodificar la cadena comprimida.

Implementación Propuesta:

- Leer la cadena codificada.

Prof.: José Fuentes Sepúlveda
Junio 2024

- Recorrer el árbol de Huffman siguiendo los bits de la cadena codificada (0 para ir a la izquierda, 1 para ir a la derecha).
- Cuando se llega a una hoja del árbol, se agrega el carácter correspondiente a la salida y se reinicia el recorrido desde la raíz del árbol.

Lempel-Ziv-Welch (LZW)

La implementación del algoritmo de compresión y descompresión Lempel-Ziv-Welch (LZW) en los códigos proporcionados se centra en la construcción de un diccionario dinámico de subcadenas durante la compresión y descompresión de datos.

1. Compresión LZW

Técnica Utilizada:

- **Trie**: Se utiliza un Trie (árbol de prefijos) para la búsqueda y almacenamiento eficiente de subcadenas durante la compresión.

Implementación Propuesta:

- **Inicialización**:
  - Crear un Trie vacío e insertar las 256 entradas iniciales correspondientes a los caracteres ASCII.
  - Inicializar variables auxiliares para manejar las subcadenas (p y c) y el código inicial para nuevas subcadenas (code = 256).
- **Proceso de Compresión**:
  - Iterar sobre cada carácter de la cadena de entrada.
  - Formar la subcadena pc a partir de p y c.
  - Buscar pc en el Trie.
    - Si pc se encuentra, actualizar p a pc.
    - Si pc no se encuentra, insertar pc en el Trie con un nuevo código, agregar el código de p a la salida, y actualizar p a c.
  - Al final, agregar el código de la subcadena restante p a la salida.

2. Descompresión LZW

Técnica Utilizada:

- **Trie y Unordered Map**: Utilizar el Trie o unordered_map construido durante la compresión para descomprimir los datos.

Implementación Propuesta:

- **Inicialización**:
  - Crear un Trie o unordered_map vacío e insertar las 256 entradas iniciales correspondientes a los caracteres ASCII.
  - Inicializar variables auxiliares para manejar la descompresión (old, s, c, code).
- **Proceso de Descompresión**:
  - Iniciar la descompresión con el primer código de la entrada.
  - Iterar sobre los códigos de entrada.
  - Reconstruir las subcadenas a partir de los códigos y actualizar el Trie o unordered_map con nuevas entradas generadas durante la descompresión.

Análisis teórico:

1. Algoritmo Lempel-Ziv-Welch (LZW)

LZW es un algoritmo de compresión sin pérdida que construye un diccionario de subcadenas vistas previamente durante el proceso de compresión. Cada subcadena en el diccionario está asociada con un código único. Durante la compresión, se identifican las subcadenas más largas que ya están en el diccionario y se emite su código correspondiente. Durante la descompresión, se utiliza el diccionario para reconstruir la cadena original a partir de los códigos.

Comprensión:

Complejidad Temporal:

- **Inserción y Búsqueda en Trie**: En el peor caso, insertar y buscar en un Trie tienen una complejidad de $O(m)$, donde m es la longitud de la subcadena. Sin embargo, en promedio, la búsqueda y la inserción son eficientes debido a la naturaleza estructurada del Trie.
- **Compresión**: Dado que cada carácter de la cadena de entrada se procesa una vez, la complejidad temporal general de la compresión es $O(n \cdot m)$, donde n es la longitud de la cadena de entrada y m es la longitud promedio de las subcadenas almacenadas en el Trie.

Complejidad Espacial:

- **Espacio del Trie**: El espacio utilizado por el Trie depende del número de subcadenas almacenadas y la longitud de estas subcadenas. En el peor caso, esto puede ser $O(n^2)$ pero en la práctica es más eficiente debido a la estructura compartida del Trie.

Prof.: José Fuentes Sepúlveda
Junio 2024

Descompresión:

Complejidad Temporal:

- **Búsqueda en Trie o Unordered Map**: La búsqueda en un Trie tiene una complejidad de $O(m)$, mientras que en un unordered_map es $O(1)$ en promedio.
- **Descompresión**: Similar a la compresión, la complejidad temporal es $O(n \cdot m)$.

Complejidad Espacial:

- **Espacio del Trie o Unordered Map**: La complejidad espacial es similar a la de la compresión.

2. Codificación Huffman

Codificación

Descripción del Algoritmo: El algoritmo de codificación Huffman construye un árbol binario basado en las frecuencias de los caracteres en la cadena de entrada. Los caracteres más frecuentes se asignan a códigos binarios más cortos, mientras que los menos frecuentes se asignan a códigos más largos.

Complejidad Temporal:

- **Construcción del Árbol**: La construcción del árbol de Huffman implica combinar nodos en una cola de prioridad, lo que tiene una complejidad de $O(d\ logd)$, donde d es el número de diferentes caracteres en la cadena de entrada.
- **Generación de Códigos**: Una vez construido el árbol, generar los códigos binarios para cada carácter tiene una complejidad de $O(d)$.

Complejidad Espacial:

- **Espacio del Árbol**: El espacio utilizado por el árbol de Huffman es $O(d)$, ya que cada nodo representa un carácter o una combinación de caracteres.

Decodificación

Descripción del Algoritmo: Durante la decodificación, el árbol de Huffman se utiliza para reconstruir la cadena original a partir de la secuencia de bits comprimida. A medida que se procesan los bits, se recorre el árbol hasta llegar a una hoja, que representa un carácter original.

Complejidad Temporal:

- **Decodificación**: La decodificación de cada bit implica un recorrido en el árbol, que tiene una complejidad de $O(1)$ por bit en promedio, ya que la profundidad del árbol es logarítmica en función del número de caracteres. La complejidad temporal total es $O(b)$, donde b es el número de bits en la cadena codificada.

Complejidad Espacial:

- **Espacio del Árbol**: Similar a la codificación, el espacio utilizado es $O(d)$.

Comparación Teórica

LZW vs Huffman:

Compresión:

- LZW es más adecuado para datos con patrones repetitivos largos, sin necesidad de una etapa previa para analizar las frecuencias de los caracteres.
- Huffman es más eficiente para datos con distribución de frecuencia desigual, proporcionando compresión óptima basada en las frecuencias de los caracteres.

Descompresión:

- LZW puede ser más complejo y menos predecible debido a la reconstrucción dinámica del diccionario.

- Huffman ofrece un proceso de descompresión más rápido y directo gracias a su estructura de árbol fija.

Complejidad Espacial:

- LZW puede requerir más espacio debido al crecimiento del diccionario, especialmente con datos variados.
- Huffman tiene una complejidad espacial más predecible y generalmente menor, acotada por el número de caracteres distintos en los datos de entrada.

Descripción de conjuntos de datos usados:

- Para probar el algoritmo de Huffman se usaron los siguientes archivos:
  - Un archivo de prueba de solo 819KB que contiene una concatenación de las letras de diversas canciones.
  - El archivo *english.50MB* encontrado en la página Pizza&Chili sugerida para el proyecto.
- Para probar el algoritmo de Lempel-Ziv se usaron los siguientes archivos:
  - El archivo *universities_followers.csv* proporcionado para el segundo entregable del curso, con un tamaño de 1.56MB
  - Nuevamente el archivo *english.50MB*

Maquina Utilizada:
Procesador: AMD Ryzen 5 5600 6-Core Processor 3.50 GHz
Memoria (RAM): 16 GB
Sistema operativo: Windows 10
Plataforma de trabajo: Visual Studio Code

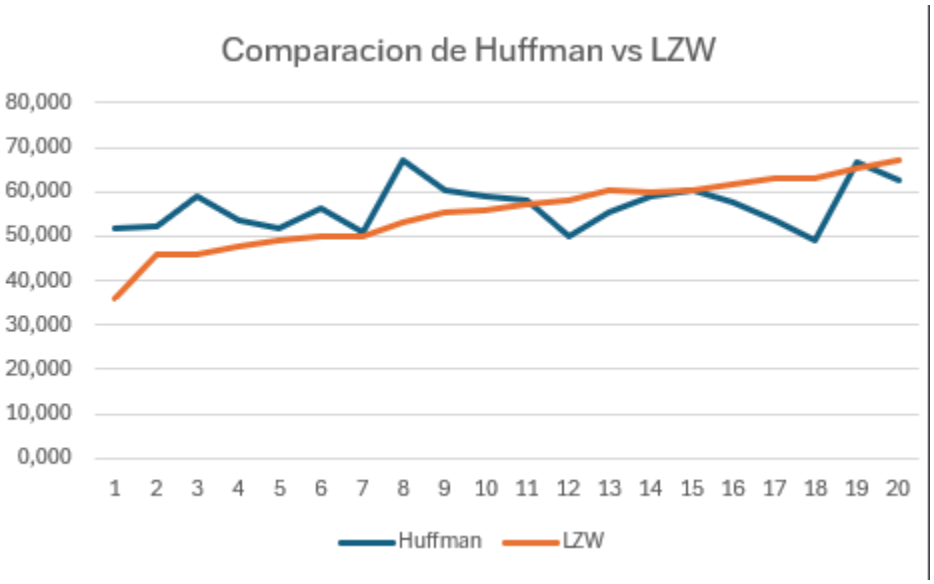
Prof.: José Fuentes Sepúlveda
Junio 2024

Análisis de los resultados:

Tabla de Pruebas

Prueba	Huffman	LZW
1	51,957	35,867
2	52,217	45,870
3	58,91	46,063
4	53,690	47,744
5	51,898	48,908
6	56,243	49,933
7	50,860	50,181
8	66,986	52,964
9	60,528	55,350
10	58,943	55,700
11	58,055	57,234
12	50,070	57,926
13	55,191	60,566
14	59,190	60,015
15	60,478	60,409
16	57,461	61,841
17	53,458	62,892
18	49,083	63,199
19	66,522	65,412
20	62,517	66,956

Prof.: José Fuentes Sepúlveda
Junio 2024



Tamaños Resultantes de cada uno:
Huffman: 24.9 MB
LZW: 24.76 MB

Prof.: José Fuentes Sepúlveda
Junio 2024

Conclusiones:

Eficiencia de Compresión

- **LZW**:
  - Eficaz para comprimir datos con patrones repetitivos largos.
  - No requiere análisis previo de las frecuencias de los caracteres.
- **Huffman**:
  - Eficiente para datos con una distribución de frecuencia desigual entre caracteres.
  - Proporciona compresión óptima basada en las frecuencias de los caracteres.
- **Elección del Algoritmo**:
  - Depende del tipo de datos a comprimir y los requisitos específicos de espacio y tiempo.

Complejidad Computacional

- **LZW**:
  - **Compresión y Descompresión**: Complejidad temporal de $O(n)$, donde n es la longitud de la cadena de entrada.
- **Huffman**:
  - **Construcción del Árbol**: Complejidad de $O(d\ logd)$, donde d es el número de caracteres distintos.
  - **Decodificación**: Complejidad de $O(b)$, donde b es el número de bits en la cadena codificada.

Manejo de Memoria

- **LZW**:
  - Requiere manejo adecuado del diccionario dinámico para evitar fugas de memoria y problemas de rendimiento.
- **Huffman**:
  - Es importante manejar adecuadamente la memoria dinámica asignada para los nodos del árbol de Huffman.
  - La complejidad espacial es más predecible y generalmente menor.

Prof.: José Fuentes Sepúlveda
Junio 2024

Conclusión Final

Basado en los experimentos realizados se llegó a la conclusión de que en general, la elección entre Huffman y LZW dependerá de la naturaleza especifica de los datos a comprimir y los requisitos de consistencia en el rendimiento de la comprensión. Puesto que, ambos algoritmos de compresión tienen rendimientos comparables, pero con cierta diferencia en sus comportamientos:

- Huffman: Tiene un rendimiento más variable con picos más altos en algunos casos, lo q puede ser beneficioso para datos que se comprimen bien con este algoritmo

- LZW: ofrece una mayor consistencia en sus resultados y podría ser preferible en escenarios donde se requiere una compresión más uniforme

Prof.: José Fuentes Sepúlveda
Junio 2024

Facultad de Ingeniería
Universidad de Concepción

Repositorio:

- https://github.com/MuddLoser/semestral

Referencias:

- https://www.geeksforgeeks.org/huffman-coding-using-priority-queue/
- https://edaa.inf.udec.cl/Preludio/huffman/
- https://www.geeksforgeeks.org/trie-insert-and-search/
- https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/
- https://www.youtube.com/watch?v=MbxvZiuFO2U

Prof.: José Fuentes Sepúlveda

Junio 2024