Universidad de Concepción

´

Proyecto Estructura de Datos
2024-1

Nombres: Felipe Mendoza Ojeda
Vicente Wiederhold Andrade
Franco Ponce Chacana
Profesor: Jose Sebastian Fuentes Sepulveda

Universidad de Concepción

Introducción:

En este proyecto el problema planteado es la implementación de estructuras de datos más complicadas con el objetivo de trabajar con el espacio de manera más eficiente y luego comparar el tiempo que tarda en codificar un texto contra el el tiempo que tarda comprimirlo y esto realizando al menos 20 repeticiones del experimento.

Usando la codificación de Huffman tendremos que implementar métodos de codificar y decodificar de un texto y de la misma manera con la compresión Lempel-Ziv tendremos que implementar la compresión y descompresión.

Universidad de Concepción

Algoritmo de Huffman:

El algoritmo de compresión de Huffman es un método de compresión sin pérdida utilizado para reducir el tamaño de datos para archivos de texto, mediante la asignación de códigos binarios de largo variable donde los símbolos más frecuentes se le asignan un código más corto y los menos frecuentes códigos más largos.

El archivo comprimido consiste en los códigos binarios, que generalmente ocupan menos espacio que los símbolos originales, especialmente cuando los símbolos frecuentes tienen códigos más cortos.

- **Frecuencia:** Se calcula la frecuencia de cada símbolo en el archivo que se desea comprimir los símbolos son caracteres individuales como letras, número o caracteres especiales.

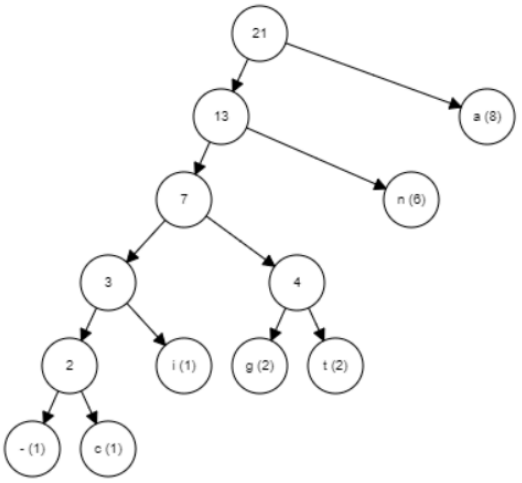
- **Construcción del árbol:** Creación de nodos iniciales donde cada símbolo se representa como un nodo inicial con su frecuencia de aparición como valor estos nodos hojas del árbol. Los nodos de menor frecuencia se combinan para formar los nodos internos cuya frecuencia es la suma de las frecuencias de los dos nodos combinados. Este proceso se repite hasta que todos los nodos se combinan en un único árbol binario completo.

- **Asignación de códigos binarios:** Recorriendo el árbol de Huffman desde la raíz hasta cada nodo hoja, se asignan códigos binarios donde cada movimiento a la izquierda del árbol representa un 0 mientras que un movimiento a la derecha del árbol representa un 1.(Cada símbolo termina con un código único basado en el camino desde la raíz hasta ese símbolo en el árbol)

- **Generación del archivo comprimido**: Se reemplazan los símbolos originales del archivo con los códigos binarios generados según la asignación de código del paso anterior.

- **Descompresión**: Utilizando el árbol de Huffman construido durante la compresión, se puede recuperar el archivo original a partir de los códigos binarios del archivo comprimido.

Caracteres	Frecuencia	Codigo
a	8	0
n	6	10
t	2	1100
g	2	1101
i	1	1110
c	1	11110
-	1	11111



Universidad de Concepción

Algoritmo de LZ (Lempel-Ziv):

El algoritmo de compresión LZ (Lempel-Ziv) es un método de compresión de datos sin pérdida. Funciona identificando y reemplazando secuencias repetitivas de caracteres por referencia a texto visto anteriormente.

- **Diccionario:** Se comienza con un diccionario vacío que se utilizará para almacenar los caracteres y las secuencias de caracteres encontradas durante la compresión.

- **Lectura del texto:** El algoritmo lee el texto de entrada que se desea comprimir. donde se agregara al diccionario los caracteres y las secuencias de caracteres a medida que se vaya leyendo.

- **Codificación:** Se busca la secuencia más larga en el diccionario que coincida con la parte actual de la lectura.Cuando se encuentra una coincidencia, se codifica utilizando su posición en el diccionario.

- **Finalización**: El proceso termina cuando se ha procesado todo el texto. El resultado es una secuencia de datos comprimidos que contiene las referencias a las secuencias repetitivas en lugar de las secuencias completas.

- **Implementación**: se buscó la implementación clásica del Lempel Ziv Welch utilizando “unordered_map” llamado “dictionary” para almacenar las cadenas de caracteres ASCII.

- **Compresión:**Se busca la secuencia ya vista en el texto y se reemplaza por un código que representa la posición y longitud de la secuencia.

- **Descompresión:** Se utiliza la información codificada para reconstruir el texto original.

- **Altura máxima:** La altura máxima es limitada por el tamaño de la ventana de búsqueda

- **Referencias:** se usó el link (3) como referencia principal para guiarnos, aparte de las otras referencias dadas.

Universidad de Concepción

Descripción de la experimentación

La función en C++ lee los archivo input.txt, input2.txt, input3.txt, input4.txt y input5.txt donde verifica que se pueda encontrar el archivo. el texto se almacena en string donde en un ciclo que se repita 20 veces se realizan la medición del tiempo de la compresión y descompresión del algoritmos de huffmann y LZ (Lempel-Ziv) además de entregar el peso del mensaje codificado.

Especificaciones de la máquina utilizada:

Tarjeta de video:	Nvidia RTX 3080
Procesador:	Amd ryzen 5800x (8cores)
Memoria RAM:	16gb @ 3600 mhz
Memoria caché:	4MB L2-32 MB L3
Windows:	10

Universidad de Concepción

Resultados experimentales

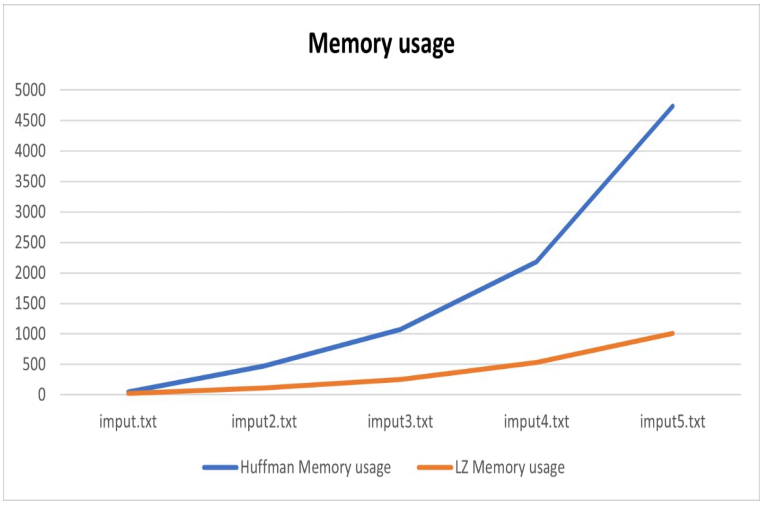
Para realizar el experimento se realizaron 20 pruebas para cada una con textos de aproximadamente 20, 100, 250, 500,1000 caracteres.

El peso de los archivo comprimido se encuentra en bits mientras que el tiempo fué medido en microsegundos para cada una de las pruebas, en las que se midió el tiempo de compresión y descompresión.

Caracteres	Huffman Encoding time	Huffman Decoding time	Huffman Memory usage	LZ Encoding time	LZ Decoding time	LZ Memory usage
imput.txt	38	4	50	18	1	21
imput2.txt	119	25	475	61	3	113
imput3.txt	200	74	1075	122	7	256
imput4.txt	242	111	2180	221	12	531
imput5.txt	546	243	4740	417	33	1010

Peso de los archivos (en bits)

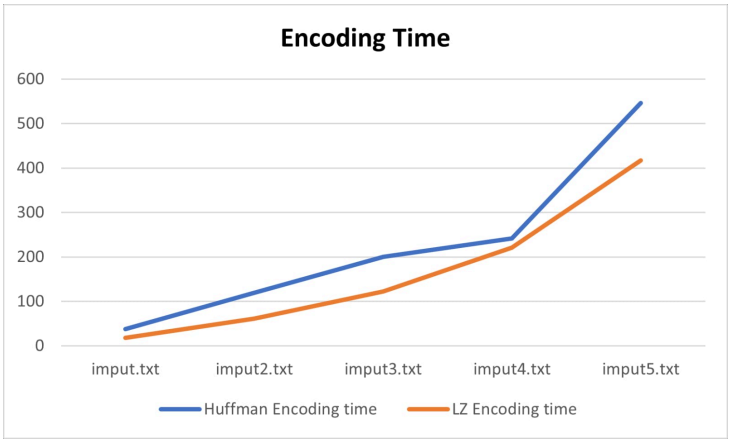
	Huffman Memory usage	LZ Memory usage
imput.txt	50	21
imput2.txt	475	113
imput3.txt	1075	256
imput4.txt	2180	531
imput5.txt	4740	1010



Universidad de Concepción

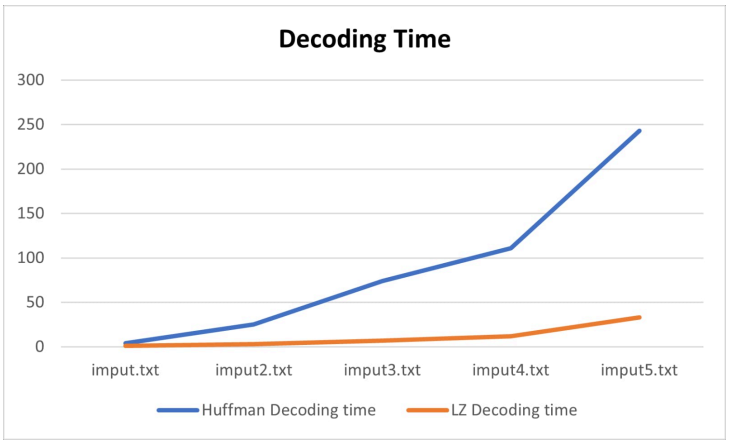
Tiempo de compresión (en microsegundos)

	Huffman Encoding time	LZ Encoding time
imput.txt	38	18
imput2.txt	119	61
imput3.txt	200	122
imput4.txt	242	221
imput5.txt	546	417



Tiempo de descompresión (en microsegundos)

	Huffman Decoding time	LZ Decoding time
imput.txt	4	1
imput2.txt	25	3
imput3.txt	74	7
imput4.txt	111	12
imput5.txt	243	33



Universidad de Concepción

Resultados experimentales (Corregidos)

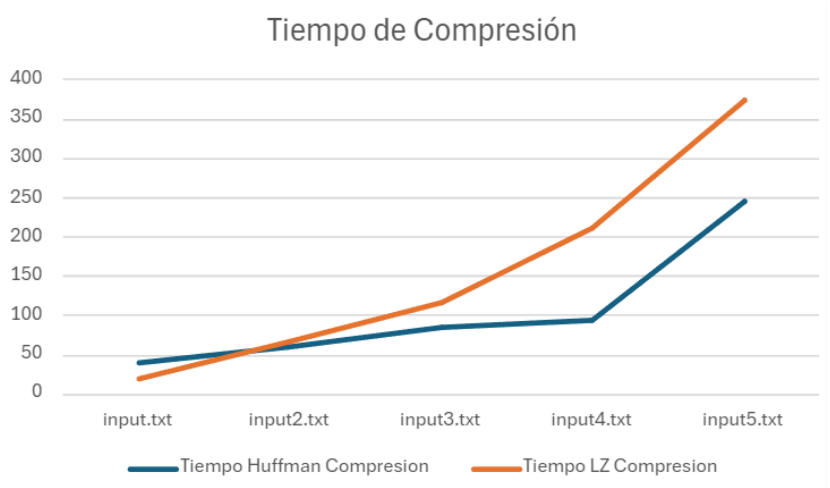
Archivo	Tiempo Huffman Compresion	Tiempo Huffman Descompresion	Memoria Huffman	Tiempo LZ Compresion	Tiempo LZ Descompresion	Memoria LZ
input.txt	39	6	5464064	20	0	5468160
input2.txt	61	29	5525504	66	2	5537792
input3.txt	84	58	5537792	117	5	5554176
input4.txt	94	117	5554176	212	9	5627904
input5.txt	246	254	5627904	373	16	5681152

En la imagen anterior están representados los resultados experimentales pero esta vez con el código funcionando correctamente (con los mismos inputs), por lo que podemos identificar ciertos cambios ya que en los primeros resultados experimentales (los que están en las páginas anteriores) los obtuvimos con unos fallos que vamos a abarcar detalladamente, compararlos con los correctos y extraer las razones de estas diferencias.

Universidad de Concepción

Tiempo de compresión (en microsegundos)

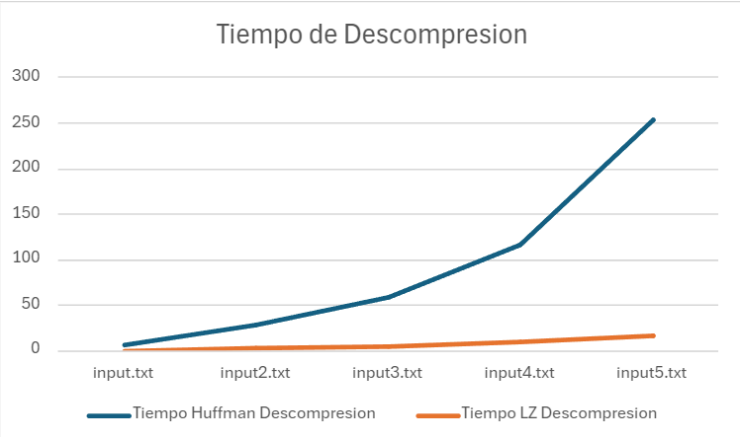
Archivo	Tiempo Huffman Compresion	Tiempo LZ Compresion
input.txt	39	20
input2.txt	61	66
input3.txt	84	117
input4.txt	94	212
input5.txt	246	373



Aca vemos una clara diferencia ya que para comprimir el LZ toma más tiempo, al principio parten parejos pero a mayor cantidad de caracteres este se dispara del Huffman, esto porque al LZ le toma más tiempo guardar cadenas. En nuestro experimento anterior obtuvimos que el LZ era mucho más rápido ya que al comprimir mal, había una pérdida de información. Nuestro Huffman pasado tenía un límite de bits por lo que le tomaba más tiempo comprimir. Nuestro Huffman actual no tiene esa limitación.

Tiempo de descompresión (en microsegundos)

Archivo	Tiempo Huffman Descompresion	Tiempo LZ Descompresion
input.txt	6	0
input2.txt	29	2
input3.txt	58	5
input4.txt	117	9
input5.txt	254	16

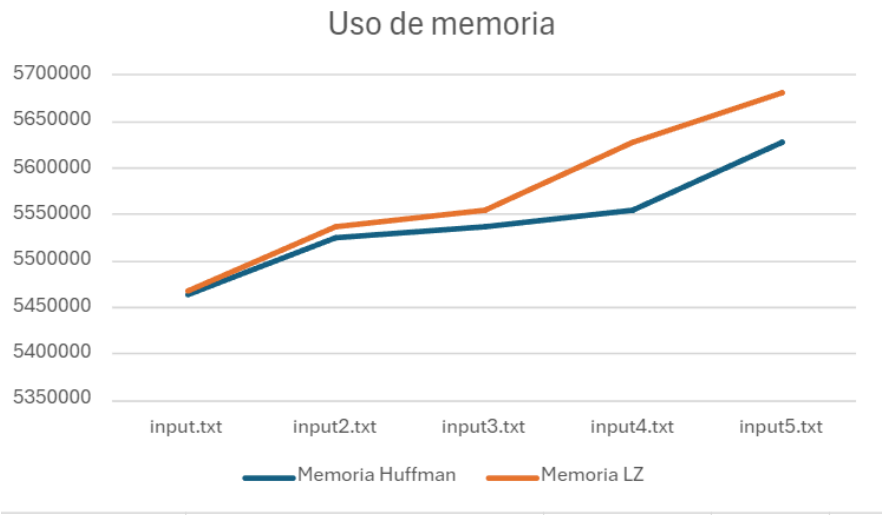


Aquí es donde hubo más similitud con nuestro experimento pasado sin embargo hay una diferencia entre los tiempos. El LZ actual al tener cadenas repetida guardadas le toma menos tiempo descomprimir pero nuestro LZ pasado le tomaba poco tiempo ya que al no guardar nada descomprimir mal y había una falta/pérdida de datos.

Universidad de Concepción

Peso de los archivos (bytes)

Archivo	Memoria Huffman	Memoria LZ
input.txt	5464064	5468160
input2.txt	5525504	5537792
input3.txt	5537792	5554176
input4.txt	5554176	5627904
input5.txt	5627904	5681152



La diferencia con nuestro gráfico del código anterior es más notoria, ya que el Huffman ocupaba más memoria porque tenía un límite de bits por caracter y no consideraba las repeticiones de ese caracter. Por otro lado nuestro LZ anterior ocupaba mucha menos memoria, pero no porque había más o menos cadenas repetidas sino que ya que como hemos mencionado anteriormente había pérdida de información por lo que codificaba mal.

Universidad de Concepción

Conclusiones:

Sobre este trabajo podemos concluir muchas cosas partiendo por la eficiencia, ya que corroboramos que efectivamente tanto la implementación de Huffman como el LZ transforman el tamaño de los archivos a unos más pequeños cuando se trata de comprimir y descomprimir, ahora bien, en términos de rendimiento es donde resalta la diferencia ya que si nos fijamos bien mientras más se va aumentando el tamaño de caracteres el LZ va siendo más efectivo y se va alejando de Huffman en lo que a tiempo se refiere, esto es lógico ya que en el algoritmo de Huffman tiene que ir recorriendo el árbol, mientras que el LZ usa su diccionario y si se repite una parte de caracteres se ahorra tiempo.
También tenemos que por obvias razones al tener más caracteres, aumenta la tasa de compresión. Con las diversas pruebas que hicimos se puede decir que en términos de tiempo es más confiable el LZ.

Con tiempo pudimos arreglar ciertos problemas que tuvimos. Al estudiar más a fondo el código huffman nos dimos cuenta de su construcción y de cómo es aplicada, ya que en si los errores que tuvimos fueron más de comprensión que de codificación.

En resumen, tanto Huffman como LZ son algoritmos de compresión valiosos con sus propias ventajas y desventajas. Huffman es más sencillo y eficiente para textos pequeños, mientras que LZ ofrece mejor rendimiento temporal y tasa de compresión para textos más grandes a expensas de un mayor consumo de memoria. La elección del algoritmo adecuado depende del contexto específico y de las prioridades de almacenamiento y rendimiento de la aplicación.

Universidad de Concepción

Linkografia:

1. https://github.com/BisenteWiederhold/Proyecto_Semestral_ED.git
2. https://youtu.be/DNICSLYRAY0?si=3tPylLUcD6fNi8gO
3. https://www.youtube.com/watch?v=woWY9LUnqtg
4. https://riunet.upv.es/bitstream/handle/10251/80465/GASTALDO%20-%20ESTUDIO%20DE%20IMPLEMENTACI%C3%93N%20DEL%20ALGORITMO%20LZW%20%28Lempel-Ziv-Welch%29%20EN%20UNA%20PLATAFORMA%20ZYNQ%20....pdf?sequence=1&isAllowed=y
5. https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-techniqu