83GIIN - Compresión y Recuperación de Información Multimedia

Actividad 2 - Portafolio

Gagliardo Miguel Angel

28 de Marzo de 2025

Introducción

La compresión sin pérdida es una técnica fundamental en el procesamiento, transporte y almacenamiento de datos, ya que permite reducir el tamaño de los archivos sin sacrificar información. En esta actividad, se utilizaron 3 implementaciones distintas a partir de códigos de compresión en repositorios mantenidos por Michael Dipperstein en GitHub: **Huffman, codificación aritmética y LZSS**. Cada uno de estos algoritmos utiliza diferentes enfoques para representar los datos de manera más eficiente, lo que impacta directamente en la tasa de compresión obtenida como veremos más adelante.

Para analizar el desempeño de cada método, se aplicaron los algoritmos antes mencionados a distintos tipos de archivos, incluyendo texto y datos binarios en formato RAW. A partir de la Actividad 1 donde se calcularon métricas como la entropía, mostraremos en una tabla comparativa la longitud promedio de los códigos canónicos, el tamaño del archivo comprimido y la tasa de compresión. Los resultados obtenidos permiten comparar la eficiencia de cada técnica según el tipo de dato procesado.

Aclaraciones

1. Cálculo de la Longitud Promedio

Para calcular la longitud promedio de la codificación, utilizamos la fórmula:

$$L = \sum p_i * l_i$$

Donde:

- $\mathbf{p_i}$ es la probabilidad de ocurrencia de cada símbolo
- $\mathbf{l_i}$ es la longitud del código asignado a cada símbolo

Para calcular la cantidad total de símbolos utilizaremos los outputs para cada archivo incluidos en el directorio **OutputData**, sumaremos las ocurrencias de cada carácter en un sólo número y finalmente luego calcularemos las probabilidades de cada símbolo. Con esto obtendremos la longitud promedio.

2. Calculo de Tasa de Compresión

Para calcular la tasa de compresión (en porcentaje) utilizaremos la siguiente fórmula:

$$T = 100 * (1 - (T_{comprimido} / T_{original})$$

Donde:

- $T_{comprimido}$ es el tamaño del archivo luego de ser comprimido por el algoritmo
- $T_{original}$ es el tamaño original del archivo, o sea sin comprimir

Compresión de archivos

Para cada tipo de compresión (Aritmética, LZSS y Huffman) se ejecutaron los archivos **test_${compresion}.sh** respectivamente y a continuación se muestran los resultados de dichas ejecuciones.

Compresión aritmética

```
root@459c7d1c7c67:/app# ./test_arcode.sh
checking ./InputData/don-quixote.txt
uncompressed size:              2166786
static model compressed size:   1199888
adaptive model compressed size: 1202452

checking ./InputData/ecoli.fa
uncompressed size:              4089209
static model compressed size:   1101735
adaptive model compressed size: 1115152

checking ./InputData/imagen_1.raw
uncompressed size:              4096
static model compressed size:   4778
adaptive model compressed size: 4141

checking ./InputData/imagen_2.raw
uncompressed size:              4096
static model compressed size:   523
adaptive model compressed size: 688

checking ./InputData/imagen_3.raw
uncompressed size:              4096
static model compressed size:   652
adaptive model compressed size: 814

checking ./InputData/imagen_4.raw
uncompressed size:              4096
static model compressed size:   919
adaptive model compressed size: 1078

checking ./InputData/imagen_5.raw
uncompressed size:              4096
static model compressed size:   1207
adaptive model compressed size: 1362

checking ./InputData/imagen_6.raw
uncompressed size:              4096
static model compressed size:   2836
adaptive model compressed size: 2876
```

LZSS

```
root@459c7d1c7c67:/app# ./test_lzss.sh
checking ./InputData/don-quixote.txt
uncompressed size:   2166786
compressed size:     1120504

checking ./InputData/ecoli.fa
uncompressed size:   4089209
compressed size:     1487492

checking ./InputData/imagen_1.raw
uncompressed size:   4096
compressed size:     4608

checking ./InputData/imagen_2.raw
uncompressed size:   4096
compressed size:     500

checking ./InputData/imagen_3.raw
uncompressed size:   4096
compressed size:     500

checking ./InputData/imagen_4.raw
uncompressed size:   4096
compressed size:     508

checking ./InputData/imagen_5.raw
uncompressed size:   4096
compressed size:     2015

checking ./InputData/imagen_6.raw
uncompressed size:   4096
compressed size:     4432
```

Huffman

```
root@459c7d1c7c67:/app# ./test_huffman.sh
checking ./InputData/don-quixote.txt
uncompressed size:   2166786
traditional size:    1209135
canonical size:      1209032

checking ./InputData/ecoli.fa
uncompressed size:   4089209
traditional size:    1177553
canonical size:      1177675

checking ./InputData/imagen_1.raw
uncompressed size:   4096
traditional size:    5370
canonical size:      4342

checking ./InputData/imagen_2.raw
uncompressed size:   4096
traditional size:    784
canonical size:      1026

checking ./InputData/imagen_3.raw
uncompressed size:   4096
traditional size:    789
canonical size:      1026

checking ./InputData/imagen_4.raw
uncompressed size:   4096
traditional size:    1018
canonical size:      1250

checking ./InputData/imagen_5.raw
uncompressed size:   4096
traditional size:    1349
canonical size:      1576

checking ./InputData/imagen_6.raw
uncompressed size:   4096
traditional size:    2962
canonical size:      3014
```

Tabla de resultados

Nombre Archivo	Tamaño Original (bytes)	Entropía Calculada (bits)	Longitud promedio a partir de los códigos canónicos (bits)	Huffman		Aritmético		Lzss	
				Bytes	Tasa de Compresión	Bytes	Tasa de Compresión	Bytes	Tasa de Compresión
don-quixote.txt	2166786	4.4286 bits/symbol	4.46 bits/symbol	1209032	44.2%	1199888	44.6%	1120504	48.3%
ecoli.fa	4089209	2.1545 bits/symbol	4.46 bits/symbol	1177553	71.2%	1101735	73.1%	1487492	63.6%
imagen_1.raw	4096	7.9474 bits/symbol	1.5007 bits/symbol	5370	-31%	4778	-16.6%	4608	-12.5%
imagen_2.raw	4096	1.1 bits/symbol	1.5001 bits/symbol	784	80.8%	523	87.2%	500	87.8%
imagen_3.raw	4096	1.2476 bits/symbol	1.5004 bits/symbol	789	80.7%	652	84.1%	500	87.8%
imagen_4.raw	4096	1.7641 bits/symbol	1.9380 bits/symbol	1018	75.1%	919	77.6%	508	87.6%
imagen_5.raw	4096	2.321 bits/symbol	2.5748 bits/symbol	1349	67%	1207	70.5%	2015	50.8%
imagen_6.raw	4096	5.3152 bits/symbol	5.3830 bits/symbol	2962	27.6%	2836	30.8%	4432	-8.2%

Comparación de los resultados

Los resultados muestran diferencias notables en la eficiencia de cada algoritmo según el tipo de archivo.

Analizando los archivos de texto (**don-quixote.txt**, **ecoli.fa**) para los 3 tipos de codificación no se observan grandes diferencias (no más de un 9.5% punta contra punta en **Aritmético** vs **LZSS** para el archivo **ecoli.fa**). Aún así vemos que para el archivo **don-quixote.txt** LZSS logra una mejor tasa de compresión aunque no muy alejadas de Huffman y Aritmético, lo que puede indicar que dado que el Quijote es un texto con frases muy variadas, no encuentra suficientes

repeticiones largas que pueda utilizar en su diccionario interno. Aún así, vemos que en **ecoli.fa**, ocurre lo contrario donde se destacan Aritmético y Huffman, recordar que en este caso es un archivo que contiene secuencias protéicas por lo que la estructura del texto parece favorecer más a modelos estadísticos que a LZSS, basado en diccionarios.

En cuanto a los archivos de imágen en formato RAW, los resultados en contraste son más bien variados:

- Para el archivo **imagen_1.raw**, **todos los métodos tuvieron un rendimiento negativo, con tamaños comprimidos mayores al original**. Es de suponer que se debe a que los algoritmos de compresión funcionan bien cuando hay patrones o redundancia en los datos, pero si los datos son prácticamente aleatorios (todos los pixeles aparecen una cantidad muy similar de veces) como en este caso y como hemos visto en la actividad 1, no hay nada que "comprimir" realmente y por lo tanto solo queda el overhead que agregan los propios algoritmos (razón por la cual los archivos “comprimidos” terminan pesando más que el original).
- **imagen_6.raw** es el otro caso particular, donde LZSS mostró tasas de compresión negativas (-8.2%) mientras que Huffman y Aritmética mostraron tasas positivas de 27.6% y 30.8% respectivamente. Esto puede deberse a que **LZSS** busca patrones repetitivos y largas secuencias iguales, en este caso si vamos a la actividad 1 vemos que sólo hay 4 píxeles en dicho archivo (0, 50, 150 y 250) por lo cual el aumento del peso del archivo “comprimido” al utilizar LZSS sugiere que los píxeles no parecen estar organizados en secuencias largas que se puedan reemplazar eficientemente, con lo cual al igual que en el caso de **imagen_1.raw** termina introduciendo más overhead del

esperado.

- No se observan grandes dispersiones o diferencias (aprox un 8-9% de diferencia entre Huffman y Aritmético vs LZSS) en el resto de archivos de imagen que valga la pena comentar, en mi visión.

Conclusión

La efectividad de un algoritmo de compresión depende en gran medida de la distribución de los datos y la redundancia en la fuente. En los experimentos realizados, observamos que no existe un algoritmo universalmente superior, sino que su desempeño varía según el tipo de archivo procesado.

Los resultados analizados resaltan la importancia del análisis previo de los datos antes de seleccionar un método de compresión. Evaluar métricas como la entropía permite determinar si un archivo es un buen candidato para compresión y cuál método puede ser más adecuado. Además, esto demuestra que, en la práctica, muchas aplicaciones utilizan esquemas híbridos o técnicas adaptativas que combinan distintos algoritmos para maximizar la eficiencia según la naturaleza de los datos.

En definitiva, la compresión sin pérdida es una herramienta fundamental para la optimización del almacenamiento y transmisión de información, pero su éxito depende de una correcta elección del algoritmo en función de la estructura y características del archivo a comprimir.