

APD problema compresión

Darío Marcos Casalé

18 diciembre 2022

1. Implementación

Se ha elegido como lenguaje de implementación Rust, un lenguaje similar a C y C++ que permite gestionar la memoria de manera eficiente y segura. Además, se ha utilizado la librería `bitvec`¹ para la manipulación de bits.

Se ha implementado el algoritmo LZ77 de manera incremental, probando todos los fragmentos de código para asegurar que el algoritmo es correcto, realizando una serie de tests que se pueden encontrar en los distintos fuentes. Para asegurar además que el algoritmo completo funciona tanto comprimiendo como descomprimiendo, se ha verificado tras la conversión de cualquier cadena, si se descomprime se obtiene la original (en caso contrario el programa lanzará una excepción).

En cuanto al propio algoritmo, se ha implementado la versión original de LZ77, sin ningún tipo de diccionario ni estructura tipo árbol para guardar las subcadenas previas. Esto implica que el coste en tiempo es $O(n^2)$, por lo que para ficheros grandes el coste temporal se puede disparar. Por último, cabe destacar que esta implementación permite comprimir cualquier tipo de fichero (no sólo texto).

2. Análisis

2.1. Texto

Se han comprimido varios ficheros obtenidos de la librería online del Proyecto Gutenberg². Por ejemplo, consideremos una obra del célebre escritor Charles Dickens llamada

¹<https://docs.rs/bitvec/latest/bitvec/>

²<https://www.gutenberg.org>

'Some Christmas Stories'. Su tamaño en memoria es de unos 130 KB, mientras que el fichero comprimido ocupa finalmente en torno a

```
54209 Dec 19 02:20 christmas-stories.gz
138894 Dec 19 01:25 christmas-stories.txt
100079 Dec 19 02:20 christmas-stories.txt.lz77
```

Figura 1: Tamaños del texto comprimido y sin comprimir (en bytes).

La ratio de compresión alcanzada con la implementación propia es de 1.38, mientras que 'gzip' alcanza una razón en torno a 2.56.

2.2. Imágenes

Se ha probado también con una imagen para comprobar cómo se comporta el algoritmo ante ficheros de este tipo. Es esperable que su rendimiento no sea razonable en la práctica y sea peor que con ficheros convencionales, ya que los ficheros de texto generalmente contienen palabras repetidas y un subconjunto reducido de los 256 caracteres que permite representar un byte.

Se ha ejecutado el algoritmo LZ77 con la siguiente imagen:

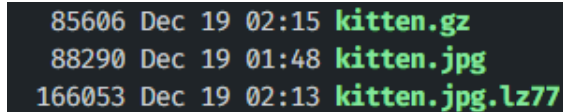


Figura 2: Imagen a comprimir

La implementación propia alcanza un 0.53 de ratio de compresión (es decir, ¡el fichero comprimido ocupa más!), y gzip apenas alcanza un 1.03 de razón de compresión.

3. Conclusión

Se ha observado que el algoritmo LZ77 original comprime razonablemente bien pero existen muchas variantes que lo mejoran y/o modifican su comportamiento (por ejemplo gzip usa DEFLATE, que combina LZ77 y codificación de Huffman). Además, se ha observado que no se comporta especialmente bien con ficheros distintos de texto, ya que no hay tantos patrones de repetición.



```
85606 Dec 19 02:15 kitten.gz
88290 Dec 19 01:48 kitten.jpg
166053 Dec 19 02:13 kitten.jpg.lz77
```

Figura 3: Tamaños de la imagen comprimida y sin comprimir (en bytes).