Implementación del algoritmo de compresión LZ78

Fernando Peña Bes

4 de enero de 2020, Universidad de Zaragoza

Algoritmo LZ78 El algoritmo LZ78, publicado por Abraham Lempel y Jacob Ziv en 1978, es un algoritmo de compresión sin pérdida. Comprime reemplazando ocurrencias repetidas de datos por referencias a un diccionario, que se va construyendo conforme se va comprimiendo la entrada. Dada una cadena de entrada, el algoritmo la va partiendo en "frases", de forma que cada frase es una de las anteriores más un símbolo. Por ejemplo, la cadena *abacbca* se comprimiría de la siguiente forma:

a	b	ac	bc	a
$(0,a)$	$(0,b)$	$(1,c)$	$(2,c)$	$(1,\varepsilon)$
1	2	3	4	5

El resultado del algoritmo de compresión es la lista ordenada de N pares $(i_n, s_n), \quad n \in \{1 \dots N\}$ (en el orden de generación de los mismos). Los pares se numeran empezando por el 1, y vamos a denotar en número asignado a cada uno con n. Cada i_n es una referencia a un par anterior, por lo que siempre se debe cumplir que $i_n < n$, y s_n es el símbolo que se añade a la frase que representa el par i_n. El par 0 representa la cadena vacía, ε.

Para obtener la frase asociada a un par (i_n, s_n), se concatena recursivamente el símbolo s al final de la cadena asociada al par i_n, hasta llegar al par 0. En el ejemplo anterior, el par $(1, c)$ se expandiría de la siguiente manera:

$$\begin{array}{ccccc} (1,c) & \xrightarrow{1} & (0,a) & \xrightarrow{0} & \varepsilon \\ c & \rightarrow & ac & \rightarrow & \boxed{\text{ac}} \end{array}$$

Para recuperar la cadena original, basta con concatenar las frases representadas por cada uno de los pares.

Compresión Con el objetivo de facilitar la implementación del proceso de compresión, se utiliza un diccionario D con entradas de la forma $(i_n, s_n) : n$, en las que el par (i_n, s_n) es la clave y n, el valor. Como clave se usa el propio par, ya que este diccionario se utiliza para comprobar si ya se ha creado un par determinado, y en ese caso se necesita obtener su n.

El diccionario que se crearía para el ejemplo planteado al principio sería el siguiente:

$\{$	a	b	ac	bc	a	$\}$
	$(0,a) : 1$	$(0,b) : 2$	$(1,c) : 3$	$(2,c) : 4$	$(1,\varepsilon) : 5$	

El algoritmo de compresión completo se incluye en el Algoritmo 1.

Algoritmo 1 Algoritmo de compresión LZ78.

```
function COMPRESS(S)
    D = {}                                          ▷ Diccionario
    C = []                                          ▷ Lista de pares con la compresión
    i = 0
    n = 1
    for s in S do
        if (i, s) ∈ D then
            i = D[(i, s)]
            L += (i, s)
        else
            D[(i, s)] = n
            i = 0
            n += 1
        end if
    end for
    return C
end function
```

La variable i va almacenando la referencia de el último par que ha encajado con la frase que se está procesando. Cuando se da la situación de que $(i, s) \notin D$, significa que se ha encontrado una nueva frase, una de las frase de las ya había en el diccionario más un carácter nuevo, y hay que añadirla al diccionario. Por otro lado, n lleva la cuenta de cuál será la referencia que se asignará a la siguiente entrada que se cree en el diccionario. Se incrementa en uno cada vez que se añade una entrada al diccionario. Cada par que se crea se añade también a la lista C, que es lo que se devuelve al final del algoritmo.

Descompresión Para descomprimir también se usa un diccionario, W. Este diccionario también contendrá una entrada por cada par de la compresión, pero para facilitar la descompresión, ahora tienen la forma $n : w$. El valor n sigue siendo el índice asociado al par, y w es la cadena de símbolos completa (la frase) que representa el par.

El algoritmo de descompresión se puede ver en el Algoritmo 2.

Algoritmo 2 Algoritmo de descompresión LZ78.

```
function DECOMPRESS(C)
    W = {}                                          ▷ Diccionario
    n = 1
    S = ε                                           ▷ Cadena vacía
    for (i, s) in C do
        if i ≠ 0 then
            w' = W[i]
            w = w' + s                              ▷ Se concatena el símbolo s a la frase recuperada, w'
        else
            w = s                                   ▷ Es como si se concatenara s a ε
        end if
        W[n] = w
        S += w
        n += 1
    end for
    return S
end function
```

Se van recorriendo los pares de la lista desde el principio hasta el final. Para cada uno, primero se obtiene la frase que representa concatenando el símbolo s a la frase que representa el par con índice i, que estará ya guardada en el diccionario (por la propiedad de que $i < n$). Una vez hecho esto, la frase que corresponde al par que se está procesando se introduce en el diccionario con n como clave, se concatena a la cadena descomprimida S, y n se incrementa en uno. Al final del algoritmo se devuelve S, que contendrá la cadena de símbolos original.

Nota: Por motivos de eficiencia, el diccionario W se podría sustituir por un vector adaptando ligeramente el algoritmo, ya que las claves son números de 1 a N.

Implementación Los algoritmos de compresión y descompresión se ha implementado en Python, y se encuentran en el fichero `lz78.py`. Los ficheros se comprimen por bytes, por lo que el programa es capaz de comprimir cualquier tipo de fichero y descomprimirlo correctamente sin ningún tipo de pérdida.

El resultado de la compresión es un fichero binario que contiene la lista de pares C, una detrás de otra. Cada par se codifica usando $\lceil \log_2(n) \rceil$ bits para i y 8 bits para s.

Para tratar el caso en el que la última porción de la entrada es igual a una de las frases que ya se ha encontrado, que provoca que el último par sea el tipo (i, ε), el primer bit del fichero es un 1 si el último byte tiene información y 0 si no. Al almacenar el par (i, ε) en el fichero se escribe el carácter nulo (`0x00`) como byte para representar ε.

Continuando con el ejemplo del inicio, la cadena `abacbca` se comprimiría de la siguiente forma:

`a`	`b`	`ac`	`bc`	`a`
$(0, \text{`a'})$	$(0, \text{`b'})$	$(1, \text{`c'})$	$(2, \text{`c'})$	$(1, \text{`}\varepsilon\text{'})$

0	1	2 … 9	10	11 … 18	19 20	21 … 28	29 30	31 … 38	39 … 41	42 … 49
0	0	01100001 ‘a’	0	01100010 ‘b’	1	01100011 ‘c’	2	01100011 ‘c’	1	00000000 ‘ε’

Como se puede ver, el 0 al inicio indica que el último byte no es válido.

La forma de invocar al programa es la siguiente:

```
Uso:
    lz78 -{c, d} [opciones]

Opciones:
    -f fichero_entrada    Indicar el fichero de entrada
    -o fichero_salida     Indicar el fichero de salida
    -h                    Mostrar ayuda

La opción -c comprime un fichero. Si no se indica un fichero de salida, se
guarda en un nuevo fichero con el mismo nombre que el de entrada más la
extensión .lz.

La opción -d restaura un fichero comprimido a su estado inicial y lo renombra
eliminado la extensión .lz. Si el fichero de entrada no tiene la extensión
.lz, el programa termina con un error y sin realizar ningún cambio.

Si no se especifican ficheros o el nombre de un fichero es -, la entrada
estándar se comprime o descomprime en la salida estándar.
```

Pruebas Para comprobar las prestaciones del compresor, se ha probado a comprimir dos libros del proyecto Gutenberg [1]: Don Quijote (en español) y Gulliver's Travels into Several Remote Nations of the World (en inglés). Los libros se han descargado en texto plano con codificación UTF-8.

Adicionalmente, se ha intentado comprimir un fichero generado de forma uniforme aleatoria de la siguiente manera:

```
cat /dev/random | head -c 1000000 > random.txt
```

y otro formado únicamente por caracteres `'a'`:

```
for i in {1..1000000}; do echo -n a >> repetido.txt; done
```

Las medidas de rendimiento han sido el factor de compresión ($\frac{|\text{comprimido}|}{|\text{sin comprimir}|}$) y el tiempo de compresión. Los tamaños de los ficheros se han calculado en KB, de la siguiente forma:

```
du -k <fichero>
```

Además de realizar las medidas para el programa implementado, también se han tomado para los compresores "reales" `gzip` y `bzip2`, con la intención tener una idea de cómo de bueno es su rendimiento.

Los resultados se muestran en la Tabla 1. Para el algoritmo LZ78 se incluye también el número de frases que se han generado en cada compresión.

Fichero	Medidas	LZ78	`gzip`	`bzip2`
Quijote `pg2000.txt`	Tamaño sin comprimir (KB)		2148	
	Tamaño símbolos (bits)		8	
	Tamaño comprimido (KB)	1024	796	592
	Factor compresión	0,4767	0,3705	0,2756
	Número de frases	314569	-	-
	Tiempo (s)	0,353	0,039	0,054
Gulliver `829-0.txt`	Tamaño sin comprimir (KB)		600	
	Tamaño símbolos (bits)		8	
	Tamaño comprimido (KB)	300	224	164
	Factor compresión	0,5	0,3733	0,2733
	Número de frases	102535	-	-
	Tiempo (s)	1,151	0,152	0,175
Uniforme Aleatorio `random.txt`	Tamaño sin comprimir (KB)		980	
	Tamaño símbolos (bits)		8	
	Tamaño comprimido (KB)	1108	980	984
	Factor compresión	1,1306	1,0000	1,0040
	Número de frases	354416	-	-
	Tiempo (s)	0,852	0,035	0,116
`'a'` repetidas `repetido.txt`	Tamaño sin comprimir (KB)		980	
	Tamaño símbolos (bits)		8	
	Tamaño comprimido (KB)	4	4	4
	Factor compresión	0,0040	0,0040	0,0040
	Número de frases	1414	-	-
	Tiempo (s)	0,270	0,009	0,010

Tabla 1: Resultados de las pruebas.

Conclusiones Al analizar las prestaciones del programa implementado, se ve que la clave para que comprima bien es la presencia de patrones en la entrada. En los libros consigue reducir su tamaño a la mitad, ya que en el lenguaje natural aparecen bastante patrones. El fichero formado por caracteres `'a'` repetidos, que es la mejor entrada para el compresor, se comprime al 0,4 % de su tamaño original. Por el contrario, el fichero obtenido de manera aleatoria uniforme con un generador de calidad no contiene patrones aparentes, y su tamaño comprimido es incluso mayor que el inicial.

Los otros dos compresores presentan fortalezas y debilidades similares, aunque ambos tienen un rendimiento mejor que el implementado.

El compresor `gzip` utiliza el algoritmo LZ77, que suele producir resultados similares a los de LZ78. Esta implementación de LZ77 seguramente utilice optimizaciones para mejorar la compresión.

Por otro lado, `bzip2` comprime utilizando el algoritmo Burrows-Wheeler y codificación Huffman. Generalmente, la compresión con este tipo de algoritmos es mejor que los que se consiguen con los algoritmos LZ77 y LZ78 tradicionales. En todas las pruebas este compresor es el que mayor factor de compresión alcanza, aunque su tiempo de compresión es ligeramente superior al de `gzip`.

El programa que se ha creado es considerablemente más lento que ambos compresores, pero hay que tener en cuenta que la implementación es bastante básica, y no está compilado sino que se ejecuta sobre el intérprete de Python.

Referencias

[1] *Gutenberg Project*. URL: `https://www.gutenberg.org` (visitado 05-01-2021).

[2] Elvira Mayordomo. *Material de la asignatura Algoritmia para problemas difíciles*. Curso 2020–21.

[3] *The LZ78 algorithm*. URL: `https://archive.is/20130107200800/http://oldwww.rasip.fer.hr/research/compress/algorithms/fund/lz/lz78.html` (visitado 05-01-2021).

[4] *Wikipedia - LZ77 and LZ78*. URL: `https://en.wikipedia.org/wiki/LZ77_and_LZ78` (visitado 05-01-2021).