WIKIPEDIA

Compresión de Burrows-Wheeler

La **transformación de Burrows–Wheeler** (**BWT** del inglés **Burrows–Wheeler transform**, también conocida como **compresión por ordenación de bloques**), es un algoritmo usado en técnicas de compresión de datos como en bzip2. Fue inventado por Michael Burrows y David Wheeler en 1994 mientras trabajaban en el DEC Systems Research Center en Palo Alto, California.[1] Se basa en una transformación previamente descubierta por Wheeler que no se encuentra publicada.

Cuando se transforma una cadena de caracteres mediante la BWT, ninguno de sus caracteres cambia de valor. La transformación permuta el orden de los caracteres. Si la cadena original contiene muchas subcadenas que aparecen a menudo, entonces la cadena transformada contendrá múltiples posiciones en las que un mismo carácter esté repetido varias veces en una fila. Esto es útil para la compresión, ya que tiende a ser fácil comprimir una cadena que contiene secuencias de caracteres repetidos con técnicas como move-to-front transform y run-length encoding.

Por ejemplo:

Entrada	`SIX.MIXED.PIXIES.SIFT.SIXTY.PIXIE.DUST.BOXES`
Salida	`TEXYDST.E.IXIXIXXSSMPPS.B..E.S.EUSFXDIIOIIIT`

La salida es más fácil de comprimir ya que tiene muchos caracteres repetidos. De hecho, en la cadena transformada, aparece un total de seis secuencias de caracteres idénticos:

`XX`, `SS`, `PP`, `..`, `II`, y `III`, que juntos representan 13 de los 44 caracteres.

Índice

Ejemplo
Optimización
Variante Biyectiva
Transformación dinámica de Burrows–Wheeler
Ejemplo de implementación
BWT en bioinformática
Referencias
Enlaces externos

Ejemplo

La transformación se realiza ordenando todas las rotaciones del texto en orden lexicográfico, y seleccionando la última columna. Por ejemplo, el texto "^BANANA@" se transforma en "BNN^AA@A" a través de estos

pasos (el símbolo rojo @ indica el 'EOF (final de fichero)'):

Transformación			
Entrada	**Todas las Rotaciones**	**Ordenar Filas**	**Salida**
^BANANA@	^BANANA@ @^BANANA A@^BANAN NA@^BANA ANA@^BAN NANA@^BA ANANA@^B BANANA@^	ANANA@^**B** ANA@^BA**N** A@^BANA**N** BANANA@**^** NANA@^B**A** NA@^BAN**A** ^BANANA**@** @^BANAN**A**	BNN^AA@A

El siguiente pseudocódigo ofrece una forma simple pero ineficiente de calcular el BWT y su inversa. Se asume que la cadena de entrada `s` contiene un carácter especial 'EOF' que será el último carácter, que sólo aparece una vez en el texto, y será ignorado durante la ordenación.

```
  function BWT (string s)
    crear una tabla donde las filas son todas las rotaciones posibles de s
    ordenar las filas alfabéticamente
    return (última columna de la tabla)
```

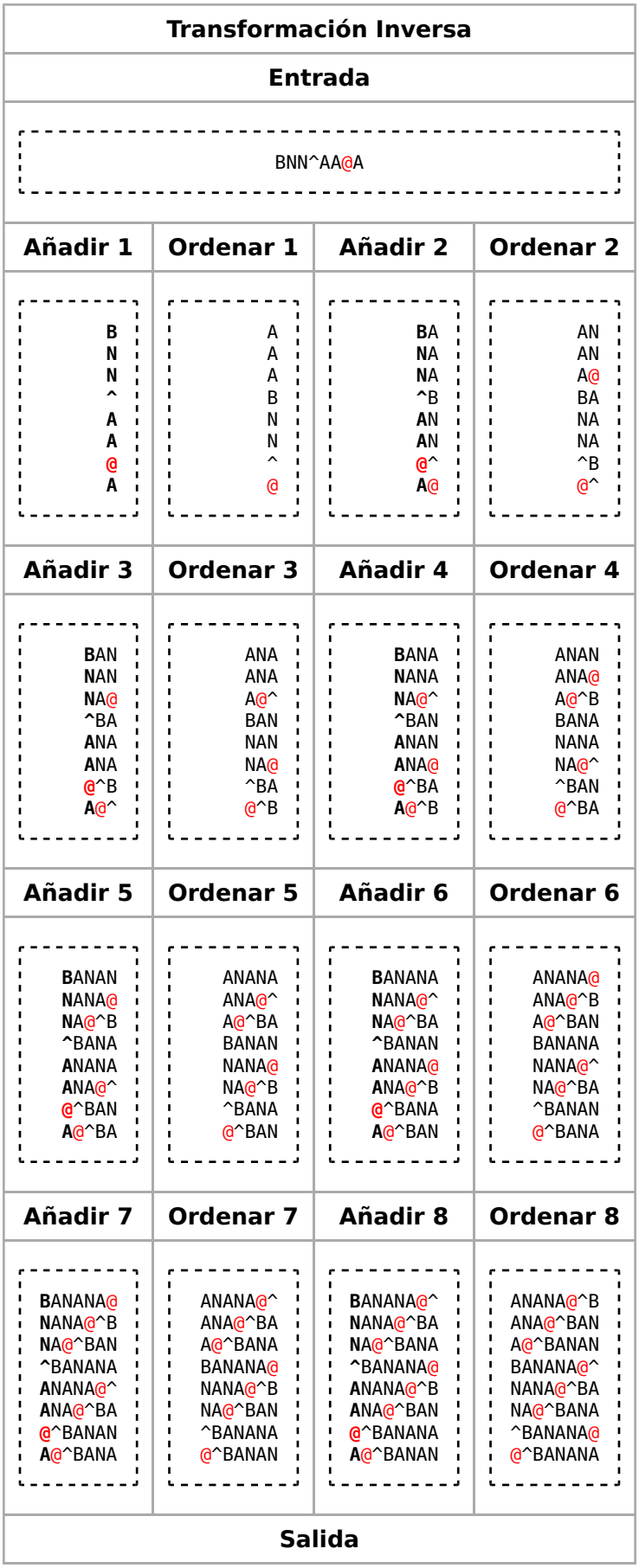
```
  function invertirBWT (string s)
    crear una tabla vacía
    repeat length(s) times
        Insertar s como una columna de la tabla antes de la primera columna de la tabla   // la primera
 inserción crea la primera columna
        ordenar las filas de la tabla alfabéticamente
    return (la fila que acabe en el carácter 'EOF')
```

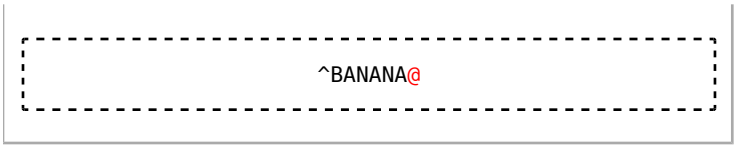
Para entender por qué se crean datos más fáciles de comprimir, se puede considerar la transformación de un texto largo en inglés que contenga frecuentemente la palabra "the". Ordenando las rotaciones de este texto a menudo agrupará rotaciones que empiecen por "he ", y el último carácter de esa rotación (que también será el carácter anterior a "he ") normalmente será "t", por lo que el resultado de la transformación contendrá un número de caracteres "t" junto con algunas excepciones menos comunes (por ejemplo, si contuviese la cadena "Brahe "). Por lo tanto, se puede ver que el éxito de esta transformación depende de un único valor con una probabilidad muy alta de ocurrencia antes de una secuencia, por lo que en general necesita muestras muy largas (de al menos unos pocos kilobytes) de datos apropiados (como texto).

Lo más interesante sobre la BWT no es que genere una salida más fácil de codificar—una ordenación ordinaria podría hacerlo igualmente—sino que es un proceso *reversible*, permitiendo re-generar el documento original a partir de la última columna de datos.

La inversa puede entenderse de la siguiente manera. Se toma la tabla final del algoritmo BWT y se eliminan todas las columnas salvo la última. Con esta información, se puede reconstruir fácilmente la primera columna. La última columna indica todos los caracteres del texto, así que se ordena, para obtener la primera columna. Entonces, la primera y la última columna juntas indican todos los *pares* de caracteres sucesivos en el documento, donde los pares están tomados cíclicamente de manera que el último y el primer carácter forman un par. Ordenando la lista de pares se obtendrán la primera *y la sengunda* columna. Continuando de esta manera, se puede recontruir la lista entera. Por último, la fila con el carácter "EOF" al final será el texto

original. La inversa del ejemplo anterior se realiza como sigue:


Transformación Inversa
Entrada
BNN^AA@A
Añadir 1
B
N
N
^
A
A
@
A
Ordenar 1
A
A
A
B
N
N
^
@
Añadir 2
BA
NA
NA
^B
AN
AN
@^
A@
Ordenar 2
AN
AN
A@
BA
NA
NA
^B
@^
Añadir 3
BAN
NAN
NA@
^BA
ANA
ANA
@^B
A@^
Ordenar 3
ANA
ANA
A@^
BAN
NAN
NA@
^BA
@^B
Añadir 4
BANA
NANA
NA@^
^BAN
ANAN
ANA@
@^BA
A@^B
Ordenar 4
ANAN
ANA@
A@^B
BANA
NANA
NA@^
^BAN
@^BA
Añadir 5
BANAN
NANA@
NA@^B
^BANA
ANANA
ANA@^
@^BAN
A@^BA
Ordenar 5
ANANA
ANA@^
A@^BA
BANAN
NANA@
NA@^B
^BANA
@^BAN
Añadir 6
BANANA
NANA@^
NA@^BA
^BANAN
ANANA@
ANA@^B
@^BANA
A@^BAN
Ordenar 6
ANANA@
ANA@^B
A@^BAN
BANANA
NANA@^
NA@^BA
^BANAN
@^BANA
Añadir 7
BANANA@
NANA@^B
NA@^BAN
^BANANA
ANANA@^
ANA@^BA
@^BANAN
A@^BANA
Ordenar 7
ANANA@^
ANA@^BA
A@^BANA
BANANA@
NANA@^B
NA@^BAN
^BANANA
@^BANAN
Añadir 8
BANANA@^
NANA@^BA
NA@^BANA
^BANANA@
ANANA@^B
ANA@^BAN
@^BANANA
A@^BANAN
Ordenar 8
ANANA@^B
ANA@^BAN
A@^BANAN
BANANA@^
NANA@^BA
NA@^BANA
^BANANA@
@^BANANA
Salida



Optimización

Se puede realizar una serie de optimizaciones para que estos algoritmos se ejecuten de manera más eficiente sin cambiar la salida. En BWT, no hay necesidad de representar la tabla ni en el codificador ni en el decodificador. En el codificador, cada fila de la tabla puede ser representada por un único puntero entre las cadenas, realizar la ordenación mediante los índices. Se debe llevar cuidado para asegurar que la ordenación no presente un mal comportamiento en el peor caso: las librerías estándar de ordenación probablemente no resulten adecuadas. En el decodificador, tampoco hay necesidad de almacenar la tabla o realizar la ordenación. En tiempo proporcional al tamaño del alfabeto y a la longitud de la cadena, la cadena decodificada puede generarse carácter a carácter de derecha a izquierda. Un "carácter" en el algoritmo puede ser un byte, un bit o cualquier otro tamaño pertinente.

No es necesario disponer de un carácter 'EOF' real. A cambio, se puede usar un puntero que recuerde dónde debería aparecer el 'EOF' si existiese. En esta aproximación, la salida del BWT debe incluir tanto la cadena transformada como el valor del puntero final. Esto significa que el BWT expande ligeramente su entrada. Por lo que la transformación inversa se reduce al tamaño original: se proporciona una cadena y un puntero, y se devuelve sólo una cadena.

Se puede encontrar una descripción completa de los algoritmos en el paper de Burrows y Wheeler o varias fuentes en línea.

Variante Biyectiva

Puesto que cualquier rotación de la cadena de entrada derivará en la misma cadena tranformada, la BWT no puede invertirse sin añadir una marca de 'EOF' a la entrada, o aumentando la salida con información, como un índice, que haga posible identificar la cadena de entrada del conjunto de todas sus rotaciones.

Existe una versión biyectiva de la transformación, por la cual la cadena transformada identifica inequívocamente a la original. En esta versión, cada cadena tiene una inversa única de la misma longitud.

La transformación biyectiva se procesa, primero factorizando la entrada como palabras de Lyndon, y luego ordenando todas las rotaciones de esas palabras. La cadena transformada se obtiene recogiendo el último carácter de las cadenas de la lista ordenada.

Por ejemplo, una aplicación de la transformación biyectiva:

Input	`SIX.MIXED.PIXIES.SIFT.SIXTY.PIXIE.DUST.BOXES`
Output	`STEYDST.E.IXXIIXXSMPPXS.B..EE..SUSFXDIOIIIIT`

La transformación biyectiva proporciona ocho secuencias de caracteres idénticos. Estas secuencias son, en orden: `XX`, `II`, `XX`, `PP`, `..`, `EE`, `..`, y `IIII`. En total, 18 caracteres forman parte de estas secuencias.

Se puede encontrar una descripción completa del algoritmo en el artículo de Gil y Scott.

Transformación dinámica de Burrows–Wheeler

En lugar de reconstruir la transformación de Burrows–Wheeler de un texto editado, Salson *et al.*[2] proponen un algoritmo que deduce la nueva transformación de Burrows–Wheeler desde la original, realizando un número limitado reordenaciones locales sobre la transformación de Burrows–Wheeler original.

Ejemplo de implementación

Esta implementación en Python sacrifica velocidad frente a simplicidad: el programa es corto, pero tiene un coste temporal mayor que lineal, que el deseado para una implementación práctica.

Usando un carácter nulo como marca de final de fichero y usando `s[i:] + s[:i]` para construir la *i*-ésima rotación de `s`, la transformación final recoge el último carácter de cada una de las filas ordenadas.

```
def bwt(s):
    """Applica la transformación de Burrows–Wheeler a la cadena de entrada."""
    assert "\0" not in s, "La cadena de entrada no puede contener el carácter nulo ('\0')"
    s += "\0"  # Se añade la marca de final de fichero
    table = sorted(s[i:] + s[:i] for i in range(len(s)))  # Tabla de rotaciones de la cadena
    last_column = [row[-1] for row in table]  # Últimos caracteres de cada fila
    return "".join(last_column)  # Convierte la lista de caracteres en una cadena
```

La transformada inversa inserta `r` repetidas veces como columna izquierda de la tabla y ordena la tabla. Tras completar la tabla entera, la función devuelve la fila que acaba en carácter nulo, salvo el carácter nulo.

```
def ibwt(r):
    """Aplica la transformación inversa de Burrows–Wheeler."""
    table = [""] * len(r)  # Crea una tabla vacía
    for i in range(len(r)):
        table = sorted(r[i] + table[i] for i in range(len(r)))  # Añade una columna de r
    s = [row for row in table if row.endswith("\0")][0]  # Encuentra la fila correcta (acabada en "\0")
    return s.rstrip("\0")  # Deshacerse del carácter nulo
```

BWT en bioinformática

La llegada de técnicas de secuenciación de alto rendimiento (HTS, del inglés High-Throughput Sequencing) en la década del 2000 lleva a otra aplicación de la transformación de Burrows–Wheeler. En la HTS, se fragmenta el ADN en pequeñas piezas, de las cuales las primeras bases son secuenciadas, arrojando millones de "lecturas", de una longitud de 30 a 500 pares de bases ("caracteres del ADN"). En numerosos experimentos, e.g., en la ChIP-Seq, la tarea es alinear estas lecturas a un genoma de referencia, i.e., a la secuencia conocida y casi completa del organismo en cuestión (el cual puede tener una longitud de varios miles de millones de pares). Se publicaron numerosos programas de alineamiento especializados en esta tarea, que inicialmente confiaban en el hashing (e.g., SOAP (https://web.archive.org/web/20181224221234/http://soap.genomics.org.cn/),[3] o Maq[4]). En un esfuerzo por reducir las necesidades de memoria para el alineamiento de secuencias, se desarrollaron muchos programas de alineamiento (Bowtie,[5] BWA,[6] and SOAP2[7]), que usan la transformación de Burrows–Wheeler para reducir la memoria necesaria para indexar el genoma humano.

Referencias

1. Burrows M and Wheeler D (1994), *A block sorting lossless data compression*

algorithm, Technical Report 124, Digital Equipment Corporation
2. Salson M, Lecroq T, Léonard M and Mouchard L (2009). «A Four-Stage Algorithm for Updating a Burrows–Wheeler Transform». *Theoretical Computer Science* **410**: 4350. doi:10.1016/j.tcs.2009.07.016 (https://dx.doi.org/10.1016%2Fj.tcs.2009.07.016).
3. Li R, *et al.* (2008). «SOAP: short oligonucleotide alignment program». *Bioinformatics* **24** (5): 713–714. PMID 18227114 (https://www.ncbi.nlm.nih.gov/pubmed/18227114). doi:10.1093/bioinformatics/btn025 (https://dx.doi.org/10.1093%2Fbioinformatics%2Fbtn025).
4. Li H, Ruan J, Durbin R (19 de agosto de 2008). «Mapping short DNA sequencing reads and calling variants using mapping quality scores» (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2577856). *Genome Research* **18** (11): 1851–1858. PMC 2577856 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2577856). PMID 18714091 (https://www.ncbi.nlm.nih.gov/pubmed/18714091). doi:10.1101/gr.078212.108 (https://dx.doi.org/10.1101%2Fgr.078212.108).
5. Langmead B, Trapnell C, Pop M, Salzberg SL (2009). «Ultrafast and memory-efficient alignment of short DNA sequences to the human genome» (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2690996). *Genome Biology* **10** (3): R25. PMC 2690996 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2690996). PMID 19261174 (https://www.ncbi.nlm.nih.gov/pubmed/19261174). doi:10.1186/gb-2009-10-3-r25 (https://dx.doi.org/10.1186%2Fgb-2009-10-3-r25).
6. Li H, Durbin R (2009). «Fast and accurate short read alignment with Burrows–Wheeler Transform» (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2705234). *Bioinformatics* **25** (14): 1754–1760. PMC 2705234 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2705234). PMID 19451168 (https://www.ncbi.nlm.nih.gov/pubmed/19451168). doi:10.1093/bioinformatics/btp324 (https://dx.doi.org/10.1093%2Fbioinformatics%2Fbtp324).
7. Li R, *et al.* (2009). «SOAP2: an improved ultrafast tool for short read alignment». *Bioinformatics* **25** (15): 1966–1967. PMID 19497933 (https://www.ncbi.nlm.nih.gov/pubmed/19497933). doi:10.1093/bioinformatics/btp336 (https://dx.doi.org/10.1093%2Fbioinformatics%2Fbtp336).

Enlaces externos

- Página de ResearchIndex para el paper sobre la BWT en Penn State (http://citeseer.ist.psu.edu/76182.html)
- Paper sobre la BWT hospedado en HP (http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.html)
- Artículo de Mark Nelson sobre la BWT (http://marknelson.us/1996/09/01/bwt/)
- A Bijective String-Sorting Transform, por Gil y Scott (http://bijective.dogma.net/00yyy.pdf)
- On Bijective Variants of the Burrows–Wheeler Transform, por Kufleitner (http://arxiv.org/abs/0908.0239)
- Post en blog (http://google-opensource.blogspot.com/2008/06/debuting-dcs-bwt-experimental-burrows.html) y página de proyect (https://code.google.com/p/dcs-bwt-compressor/) para un programa open-source de compresión y una librería basados en el algoritmo de Burrows–Wheeler

---

Esta página se editó por última vez el 16 abr 2020 a las 23:24.