Burrows-Wheeler Transform, part 1

Ben Langmead

JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

Department of Computer Science

CC BY

Please sign guestbook (www.langmead-lab.org/teaching-materials) to tell me briefly how you are using the slides. For original Keynote files, email me (ben.langmead@gmail.com).

Burrows-Wheeler Transform

Rotations of a string:

b o n b o n

o n b o n b

n b o n b o

b o n b o n

o n b o n b

n b o n b o

(after this they repeat)

bonbon
nbonbo
onbonb
bonbon
nbonbo
onbonb

Burrows-Wheeler Transform

We know dictionary order:
`as < ash` and `flower < flowers`

This is a *convention* for cases where no character comparison "breaks the tie," i.e. where one string is a prefix of the other

We could have said `ash < as` and `flowers < flower`; still a total order

Burrows-Wheeler Transform

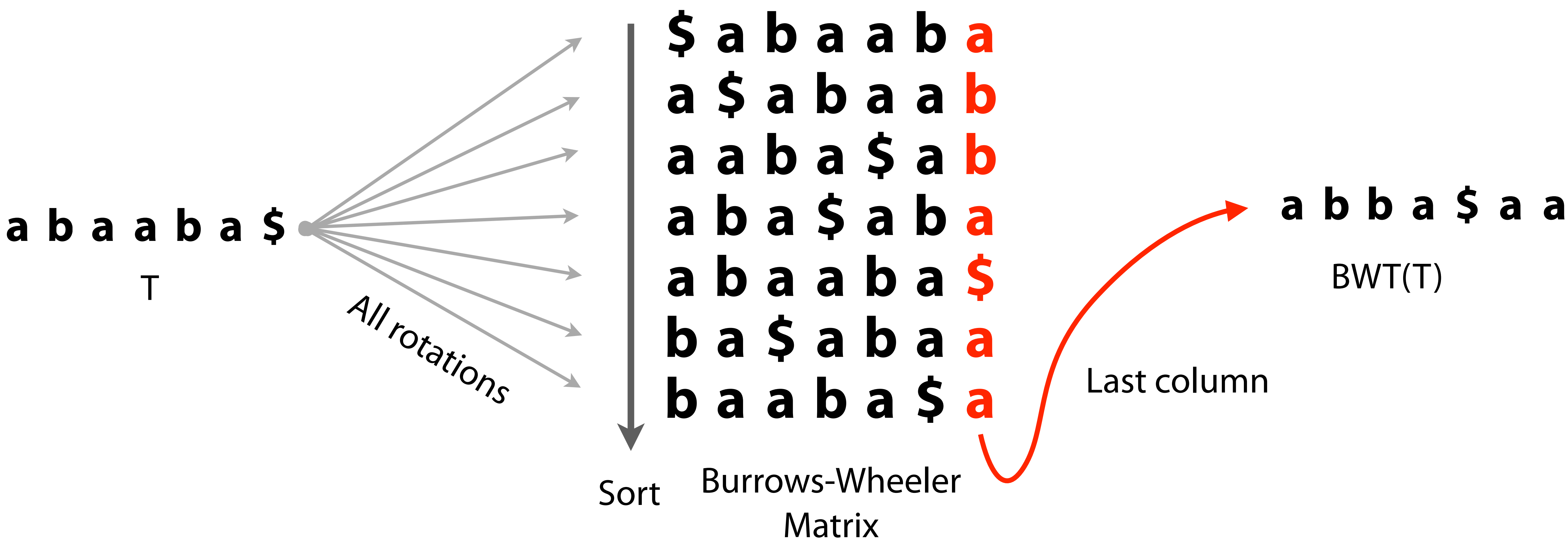
We invent a new symbol **$** ("terminator"), defined to be alphabetically less than all others:

bonbon$

Enforces dictionary order and:

No suffix is a prefix of another suffix:	bonbon$ onbon$ nbon$ bon$ on$ n$ $	And no two rotations are the same:	bonbon$ $bonbon n$bonbo on$bonb bon$bon nbon$bo onbon$b

Burrows-Wheeler Transform



Burrows M, Wheeler DJ: A block sorting lossless data compression algorithm.
Digital Equipment Corporation, Palo Alto, CA 1994, Technical Report 124; 1994

Burrows-Wheeler Transform

a b a a b a $

T

$	a	b	a	a	b	a
a	$	a	b	a	a	b
a	a	b	a	$	a	b
a	b	a	$	a	b	a
a	b	a	a	b	a	$
b	a	$	a	b	a	a
b	a	a	b	a	$	a

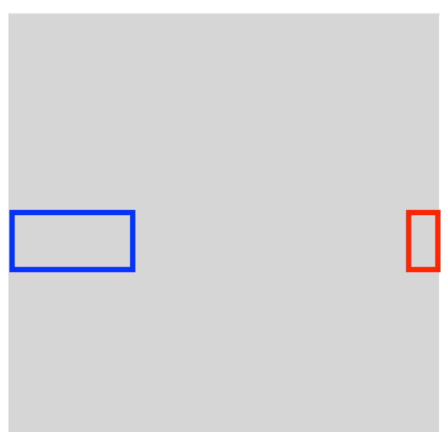
→

BWT	Right contexts
a	$ a b a a b
b	a $ a b a a
b	a a b a $ a
a	a b a $ a b
$	a b a a b a
a	b a $ a b a
a	b a a b a $

BWT(T) orders T's characters according to alphabetical order of right contexts in T

Burrows-Wheeler Transform

Ordered by *right-context*



Colors show what parts of matrix are shown on right

final char (L)	sorted rotations
a	n to decompress. It achieves compression
o	n to perform only comparisons to a depth
o	n transformation} This section describes
o	n transformation} We use the example and
o	n treats the right-hand side as the most
a	n tree for each 16 kbyte input block, enc
a	n tree in the output stream, then encodes
i	n turn, set $L[i]$ to be the
i	n turn, set $R[i]$ to the
o	n unusual data. Like the algorithm of Man
a	n use a single set of probabilities table
e	n using the positions of the suffixes in
i	n value at a given point in the vector $R
e	n we present modifications that improve t
e	n when the block size is quite large. Ho
i	n which codes that have not been seen in
i	n with ch appear in the {\em same order
i	n with ch. In our exam
o	n with Huffman or arithmetic coding. Bri
o	n with figures given by Bell~\cite{bell}.

Figure 1: Example of sorted rotations. Twenty consecutive rotations from the sorted list of rotations of a version of this paper are shown, together with the final character of each rotation.

Figure: Burrows M, Wheeler DJ: A block sorting lossless data compression algorithm. *Digital Equipment Corporation, Palo Alto, CA* 1994, Technical Report 124; 1994

Burrows-Wheeler Transform

Consider building a high-order compressor for `mississippi`

```
$mississippi
------------
i$mississipp
ippi$mississ
issippi$miss
ississippi$m
------------
mississippi$
------------
pi$mississip
ppi$mississi
------------
sippi$missis
sissippi$mis
ssippi$missi
ssissippi$mi
```

Say we're using **right** context, $k = 1$

$$H_1(T) = (1/12)\, H_0(\texttt{i}) + (4/12)\, H_0(\texttt{pssm}) +$$
$$= (1/12)\, H_0(\$) + (2/12)\, H_0(\texttt{pi}) + (4/12)\, H_0(\texttt{ssii})$$

Burrows-Wheeler Transform

$	mississipp	i	H_0
i	$mississip	p	
i	ppi$missis	s	H_0
i	ssippi$mis	s	
i	ssissippi$	m	
m	ississippi	$	H_0
p	i$mississi	p	H_0
p	pi$mississ	i	
s	ippi$missi	s	
s	issippi$mi	s	H_0
s	sippi$miss	i	
s	sissippi$m	i	

Overall: H_1

We obtain a H_k compressor by using a H_0 compressor in each length-k-context chunk. k can vary.

Burrows-Wheeler Transform

$m	ississipp	i	H_0
i$	mississip	p	H_0
ip	pi$missis	s	H_0
is	sippi$mis	s	H_0
is	sissippi$	m	
mi	ssissippi	$	H_0
pi	$mississi	p	H_0
pp	i$mississ	i	H_0
si	ppi$missi	s	H_0
si	ssippi$mi	s	
ss	ippi$miss	i	H_0
ss	issippi$m	i	

Overall: H_2

We obtain a H_k compressor by using a H_0 compressor in each length-k-context chunk. k can vary.

Burrows-Wheeler Transform

Compression strategy:

(a) Find BWT(T)

(b) Partition BWT by k-context

(c) Apply H_0 encoding in each

Glossing over details here

Space required:

H_0 codebook for each partition

Decompression strategy:

(a) Decode H_0 in each partition

(b) Concatenate partitions

(c) Find T by reversing BWT(T)

TODO

BWT is a "compression booster"

bzip2 uses this method