Burrows-Wheeler Transform, part 1

Ben Langmead

Johns Hopkins
Whiting School of Engineering

Department of Computer Science

Burrows-Wheeler Transform

Rotations of a string:

b o n b o n

o n b o n b

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

(etc)

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

(not necessarily distinct)

Burrows-Wheeler Transform

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

For cases where no character "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

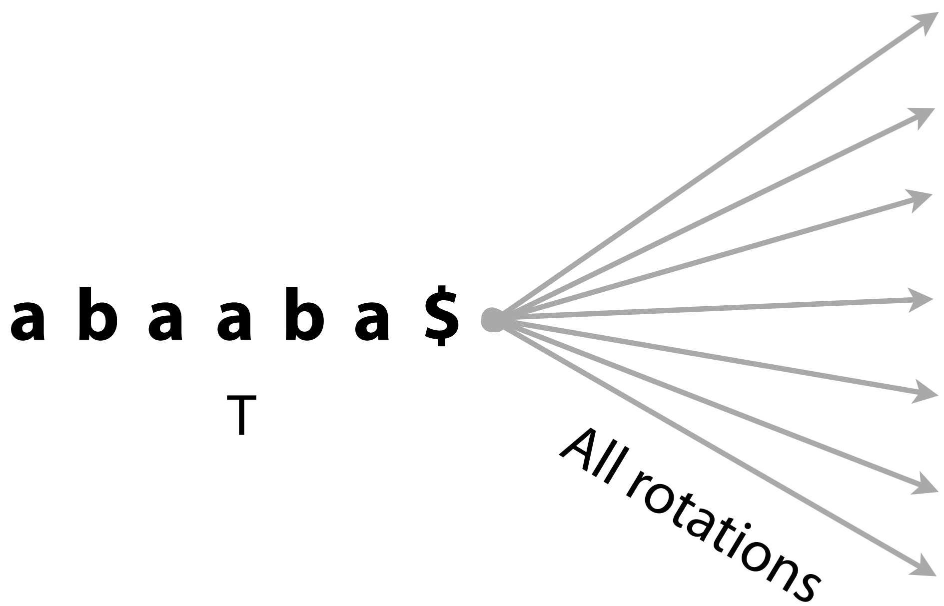
Define new symbol **$** ("terminator"), to be alphabetically less than others:

bonbon$

Enforces dictionary order and:

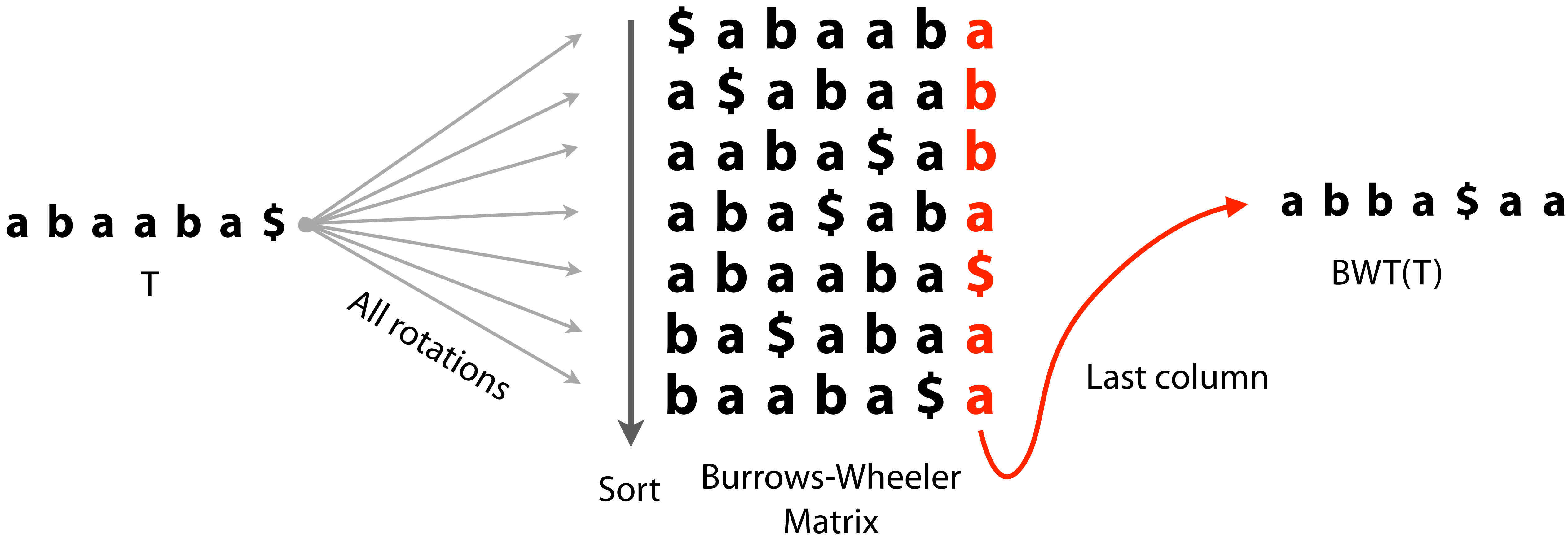
No suffix is a prefix of another suffix:	bonbon$ onbon$ nbon$ bon$ on$ n$ $	No two rotations are the same:	bonbon$ $bonbon n$bonbo on$bonb bon$bon nbon$bo onbon$b

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



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

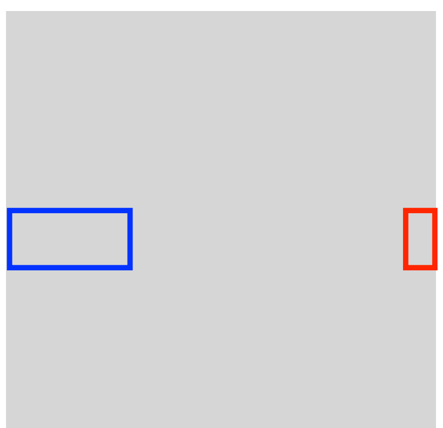
→

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.

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

Compression strategy:

(a) Get BWT(T)

(b) Partition by k-context

(c) H_0 encode partitions

Space: H_0 code for each partition

Decompression strategy:

(a) H_0 decode partitions

(b) Concatenate partitions

(c) Reverse BWT(T) to get T

TODO

BWT is a "compression booster"

Burrows-Wheeler Transform

Consider building a H_1 compressor for `mississippi`

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

$$H_1(T) = (4/11)\, H_0(\texttt{pssm}) + (1/11)\, H_0(\texttt{i}) +$$
$$= (2/11)\, H_0(\texttt{pi}) + (4/11)\, H_0(\texttt{ssii})$$

Burrows-Wheeler Transform

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

Overall: H_2

Obtain H_k code by applying H_0 code in each k-context chunk

Or just take chunks of fixed # rows. Either way, ***order is key***.