

Session 9 –Theory and Exercises

Read Alignment I: The FM Index



Date: 12/02/2024, 15:00-17:00

Teacher: **Fernando Cruz** (CNAG)

fernando.cruz@prof.esci.upf.edu

Bachelor's Degree in Bioinformatics

Course 2023-2024

52115 - Algorithms for sequence analysis in Bioinformatics (**ASAB**)

Read

CTCAAACCTCTGACCTTTGGTGATCCACCCGCTNNGGCTTC

Reference

GATCACAGGTCTATCACCTATTAACCACCTCACGGGAGCTCTCCATGCATTTGGTATTTT
CGTCTGGGGGGTATGCACGCATAGCATTGCGAGACGCTGGAGCCGGAGCACCTATGTC
GCAGTATCTGCTTTTGATTCTGCCTCATCCTATTATTTATCGCACCTACGTTCATATTT
ACAGGCGAACATACTTACTAAAGTGTTAATTAATTAATGCTTGAGGACATAATAATA
ACAATTGAATGTCTGCACAGCCACTTTCCACACAGACATCATAACAAAAAATTTCCACCA
AACCCCCCTCCCCGCTTCTGGCCACAGCACTTCTGCCCCAACCCAAAA
ACAAAGAACCCTAACACAGCCTAACCAATTTTCAAAATTTTATCTTTGGCGGTATGCAC
TTTTAACAGTCACCCCCACTAACCAATTTATTTTCCCCCTCCCACTTCCATACTACTAAT
CTCATCAATACAACCCCGCCCATCTTACCCAGCACACACACACCCCTCTAACCCCAT
CCCCGAACCAACCAACCCCAAAACCCACCCCAAGTCTTATGTAGCTTACCTCTCTCAAA
GCAATACACTGACCCCGCTCAAAACCTGGATTTTGGATCCACCCAGCCCTTTGGCCTAAA
CTAGCCTTTCTATTAGCTCTTAGAAGATTACACATGCAAGCATCCCCCTCCAGTGAGT
TCACCCCTCTAAATCACCAGGATCAAAAGGAACAAGCATCAAGCAAGCAGCAATGCAGCTC
AAAACGCTTAGCCTAGCCACACCCCTCACGGGAAAACAGCAGTGATTAACCTTTAGCAATAA
ACGAAAGTTAACTAAGCTATACTTACCCAGGGTTGGTCAATTTCTGCTCCAGCCACCGC
GGTCACACGATTAAACCAAGTCAATGAAGCCGGCGTAAAGAGTGTCTAGATCACCCCC
TCCCCAATAAAGCTAAAACTCACCTGCTTGTAAAAAACTCCAGTCAACAAAAATAGAC
TACGAAAGTGGCTTTAACATATCTGAACCACTAAGCTAAGCTTGGGATTAGA
TACCCCACTATGCTTAGCCCTAAACCTCAACCACTTACCAAACTGGCCAGAA
CACTACGAGCCACAGCTTAAAACTCAAGGACCTGGCGGTGCTTCATTTAGAGG
AGCCTGTTCTGTAAATCGATAAACCCTGATCAACCTCACCACCTCTTGCTTATATA
CCGCCATTTCAACCAACCTGATGAAGGCTACAAAGTAAGCGCAAGTACCTTACAG
ACGTTAGGTCAAGGTGTAGCCATGAGGTGGCAAGAAATGGGCTACATTTTC
AAAACCTACGATAGCCCTTATGAACTTAAAGGTCGAAGGTGGATTTAGCAGTAA
AGTAGAGTGCTTAGTTGAACAGGGCCCTGAAGCGCGTACACACCGCCCGTCAACCT
AAGTATACTTCAAGGACATTTAACTAAACCCCTACGCATTTATATAGAGGAGACA
CGTAACCTCAAACTCCTGCTTTGGTGATCCACCCGCTTGGCCTACCTGCATAATGAAG
AAGCACCCAACTTACACTTAGGAGATTTCAACTTAACTTGACCGCTCTGAGCTAAACCTA
GCCCAAACCCACTCCACCTTACTACAGACAACCTTAGCCAAACCAATTTACCCAAATAA
AGTATAGGCGATAGAAATTGAAACCTGGCGCAATAGATATAGTACCGCAAGGGAAAGATG
AAAAATTATAACCAAGCATAATATAGCAAGGACTAACCCCTATACCTTCTGCATAATGAA
TTAACTAGAAATAACTTTGCAAGGAGAGCCAAAGCTAAGACCCCGAAACCAAGACGAGCT
ACCTAAGAACAGCTAAAGAGCACAACCGCTCTATGTAGCAAAATAGTGGGAAGATTATA
GGTAGAGGCGACAAACCTACCGAGCCTGGTGATAGCTGGTTGTCCAAGATAGAATCTTAG
TTCAACTTTAAATTTGCCACAGAACCTCTAAATCCCCCTGTAAATTTAACTGTTAGTC
CAAAGAGGAACAGCTCTTTGGACACTAGGAAAAAACCTGTAGAGAGAGTAAAAATTTA
ACACCAATAGTAGGCCTAAGAGCAGCCACCAATTAAAGAAAGCGTTCAAGCTCAACACCA
CTACCTAAAAAATCCCAACATATAACTGAACCTCTCACACCAATTTGGACCAATCTATC
ACCTTATAGAAGAACTAATGTTAGTATAAGTAACATGAAACATTTCTCTCCGCATAAAGC
CTGCGTCAGATTAAAACTGAACCTGACAATTAACAGCCCAATATCTACAATCAACCAAC
AAGTCATTATTAACCTCACTGTCAACCAACACAGGCAATGCTCATAAGGAAAGGTTAAAA
AAAGTAAAGGAAGCTCGGCAATCTTACCCCGCTGTTTACCAAAAAATCAACCTCTAGC
ATCACCAAGTATTAGAGGCAACCGCTGCCAGTGACACATGTTTAAACGGCCGCGGTACCTT
AAGCCTCAAACTTACCTAGCATAATGCTTCTTAAATACCAACCTCTATCAATGCTCTC

Sequence differences occur because of...

1. Sequencing error
2. Genetic variation

What do we need to determine the original genomic location of these reads?

Global Alignment with Respect to the Reads

Local Alignment

Target Sequence

5' ACTACTAGATTACTTACGGATCAGGTACTTTAGAGGCTTGCAACCA 3'

||||| ||||| ||||| ||||| ||||| ||||| ||||| |||||

Query Sequence

5' TACTCACGGATGAGGTACTTTAGAGGC 3'

Global Alignment

Target Sequence

Reference

5' ACTACTAGATTACTTACGGATCAGGTACTTTAGAGGCTTGCAACCA 3'

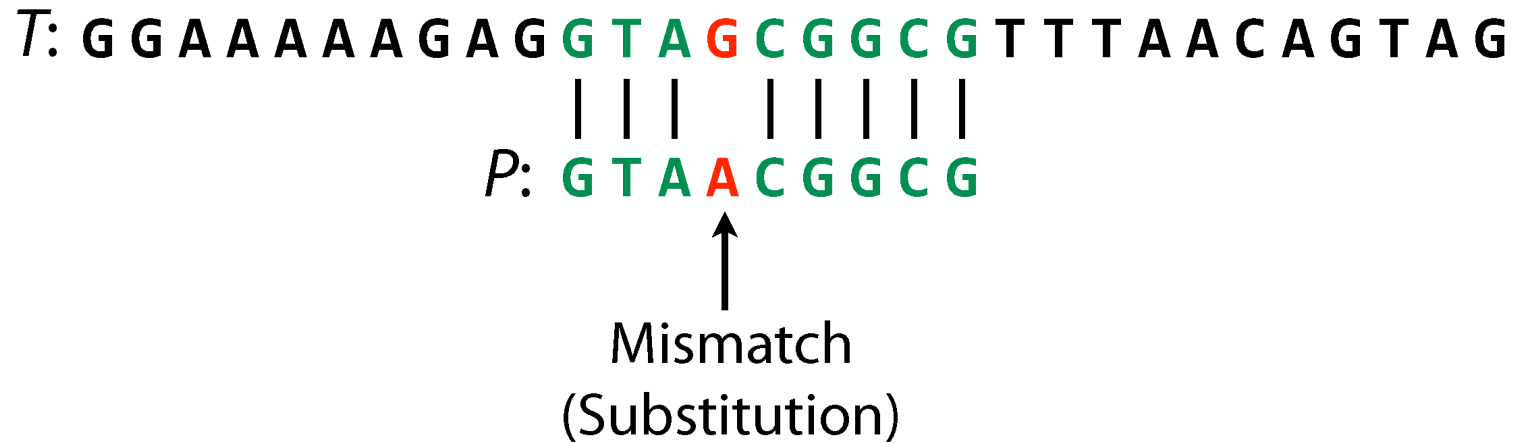
||||| ||||| ||||| ||||| ||||| ||||| ||||| |||||

5' ACTACTAGATT- - - -ACGGATC- -GTACTTTAGAGGCTAGCAACCA 3'

Query Sequence

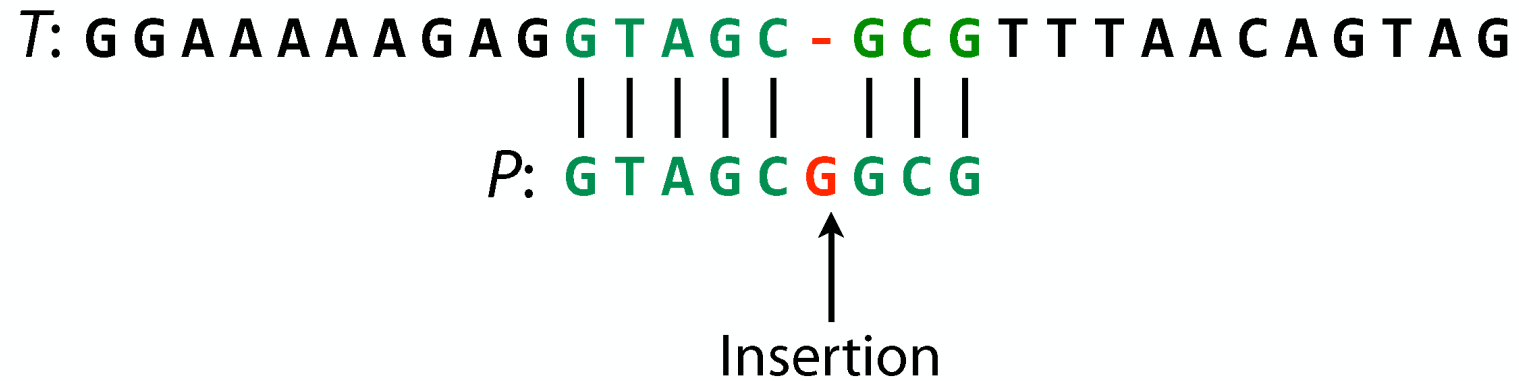
Reads

Approximate matching



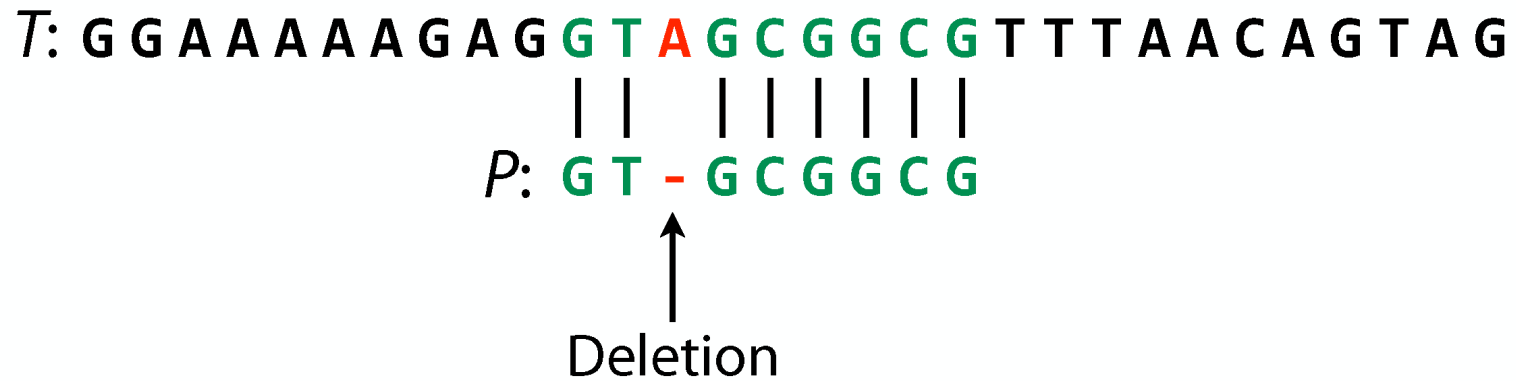
Tolerate more mismatches to accomodate sequencing *errors* and *SNPs*

Approximate matching



allow more gaps

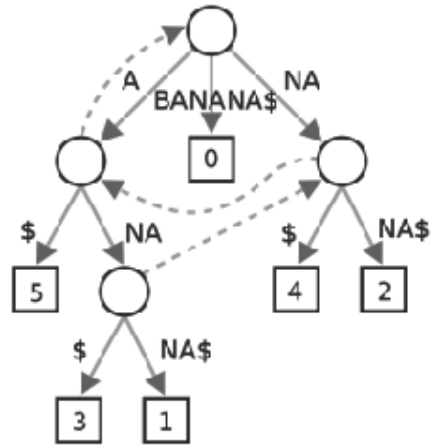
Approximate matching



allow more gaps

Burrows-Wheeler Transform (BWT)

Indexing with suffixes



Suffix Tree

6	\$
5	A\$
3	ANA\$
1	ANANA\$
0	BANANA\$
4	NA\$
2	NANA\$

Suffix Array

\$ B A N A N A
A \$ B A N A N
A N A \$ B A N
A N A N A \$ B
B A N A N A \$
N A \$ B A N A
N A N A \$ B A

FM Index

Suffix arrays are space efficient data structures...

Manber, U., and G. Myers, **1990** Suffix arrays: a new method for on-line string searches, pp. 319–327 in *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, San Francisco, California, USA.

Burrows-Wheeler Transform

Reversible permutation of the characters of a string, used originally for compression

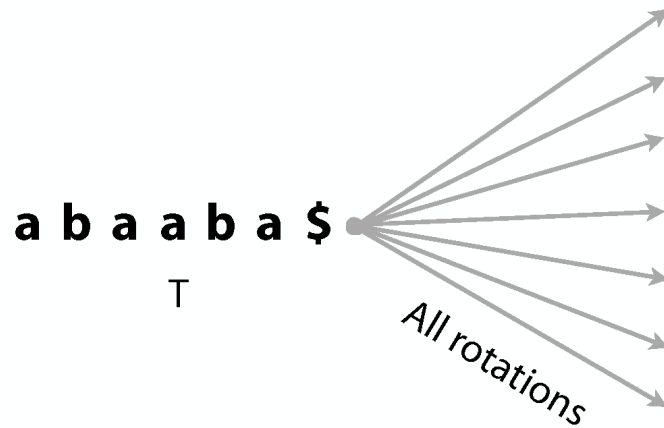
a b a a b a \$

T

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

Reversible permutation of the characters of a string, used originally for compression



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

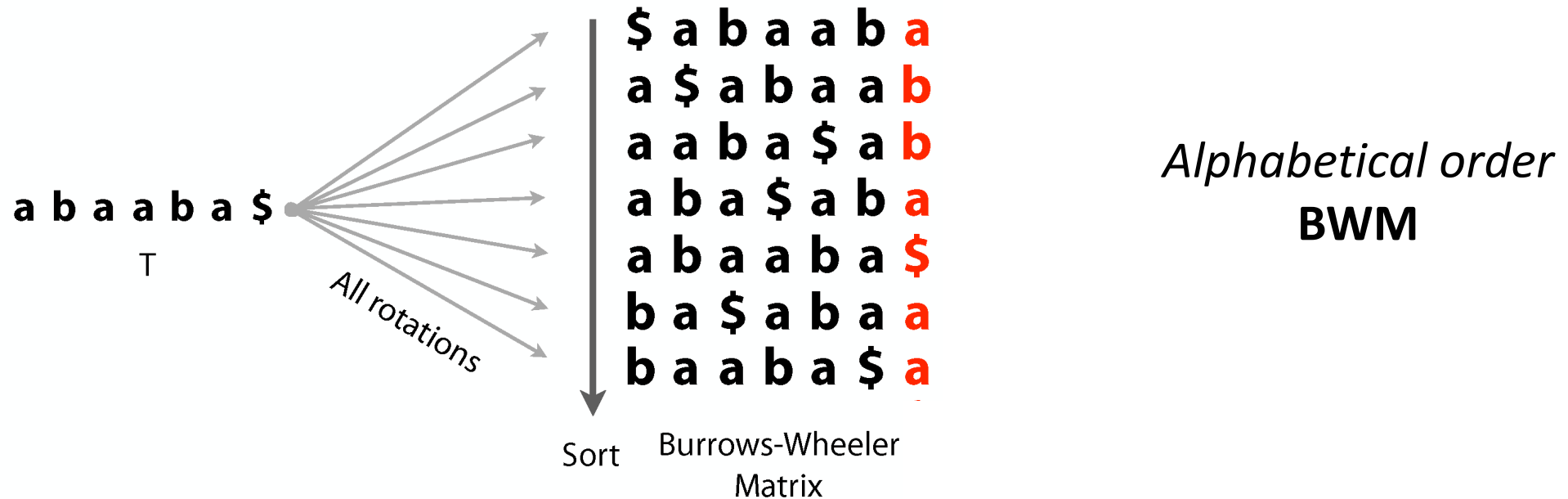
Burrows-Wheeler Transform

All rotations

a b a a b a \$
b a a b a \$ a
a a b a \$ a b
a b a \$ a b a
b a \$ a b a a
a \$ a b a a b
\$ a b a a b a
(then they repeat)

Burrows-Wheeler Transform

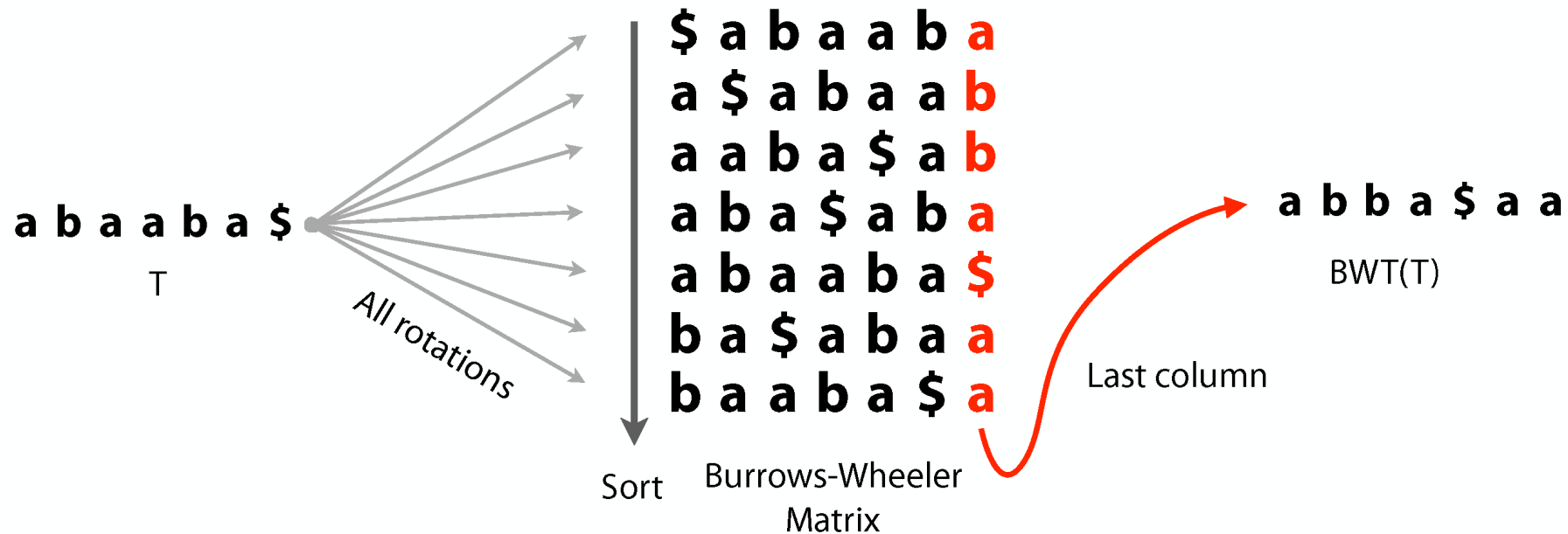
Reversible permutation of the characters of a string, used originally for compression



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

Reversible permutation of the characters of a string, used originally for compression



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

BWT in Python Code

```
def rotations(t):  
    # Return list of rotations of input string t  
    tt = t*2 # this duplicates string t to ensures being able of reading first character after last one  
    return [ tt[i:i+len(t)] for i in range(len(t)) ]
```

```
rotations('cat')
```

```
['cat', 'atc', 'tca']
```

```
def bwm(t):  
    # Return lexicographically sorted list of t's rotations  
    return sorted(rotations(t))
```

```
bwm('abaaba$')
```

```
['$abaaba', 'a$abaab', 'aaba$ab', 'aba$aba', 'abaaba$', 'ba$abaa', 'baaba$a']
```

```
print('\n'.join(bwm('abaaba$')))
```

```
$abaaba  
a$abaab  
aaba$ab  
aba$aba  
abaaba$  
ba$abaa  
baaba$a
```

```
def bwtViaBwm(t):  
    # Given T, returns BWT(T) by way of the BWM  
    return ''.join(map(lambda x: x[-1], bwm(t)))
```

```
bwtViaBwm('abaaba$') # we can see the result equals the last column of the matrix above
```

```
'abba$aa'
```

BWT features – Repetitive Runs

BWT often shows repetitive patterns

```
# Tendency to show repetitive runs
```

```
bwtViaBwm('I_cant_get_no_satisfaction_and_i_try_and_i_try$')
```

```
'y$ynItddtoiif__cs_anngs__ttoa_a_nitt_ineca__rr'
```

```
bwtViaBwm('We_all_live_in_a_yellow_submarine_yellow_submarine$')
```

```
'e$neelwwea__mmuunWvnyy_rrll_aeellbbiiiillaa__ssioo__'
```

```
bwtViaBwm('You_gotta_run_run_run_run_run_take_a_drag_or_two$')
```

```
'o$eaugannnnnr_trt_ka_auuuuuw_gYod_____t_o_orrrrrt'
```

```
# Repetitive Runs
```

```
bwtViaBwm('Tomorrow_and_tomorrow_and_tomorrow$')
```

```
'w$wwdd__nnoooaattTmmrrrrrrrooo__ooo'
```

Allows using Run-Length Encoding

```
bwtViaBwm('You_gotta_run_run_run_run_run_take_a_drag_or_two$')
```

```
'o$eaugannnnnr_trt_ka_auuuuuw_gYod_____t_o_orrrrrt'
```

```
return_encoding_list(bwtViaBwm('You_gotta_run_run_run_run_run_take_a_drag_or_two$'))
```

```
o1$1e1a1u1g1a1n5r1_1t1r1t1_1k1a1_1a1u5w1_1g1Y1o1d1_5t1_1o1_1o1r5t1
```

```
bwtViaBwm('AAAG$GAGAAAGTTTTGTTGCTA')
```

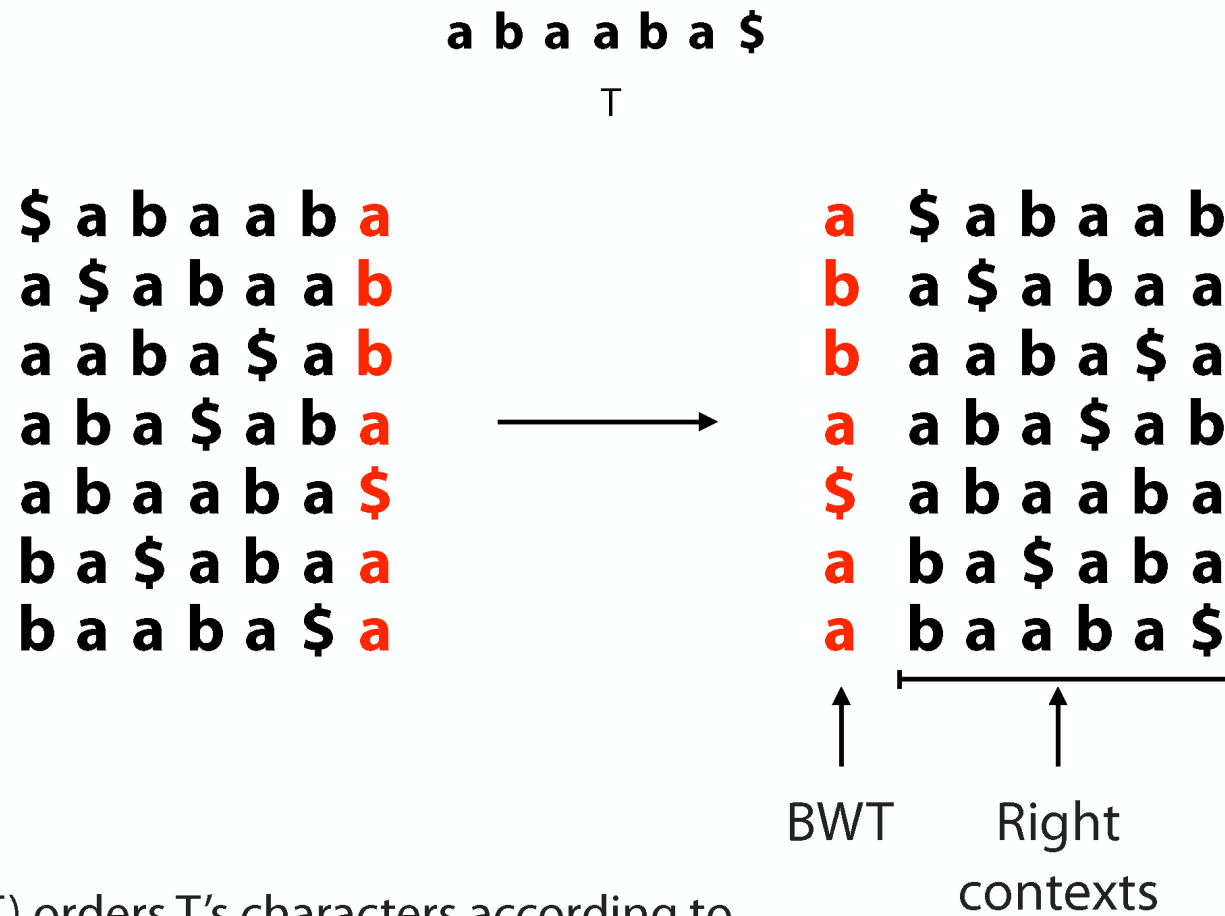
```
'GTAGAAAGAGAA$TTACTTGTTTG'
```

```
return_encoding_list(bwtViaBwm (Read))
```

```
A3G1$1G1A1G1A3G1T5G1T2G1C1T1A1
```

BWT features - The right contexts

Burrows-Wheeler Transform



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

Right context

The right context of a position in T consists of everything that comes after it with "wrap around"

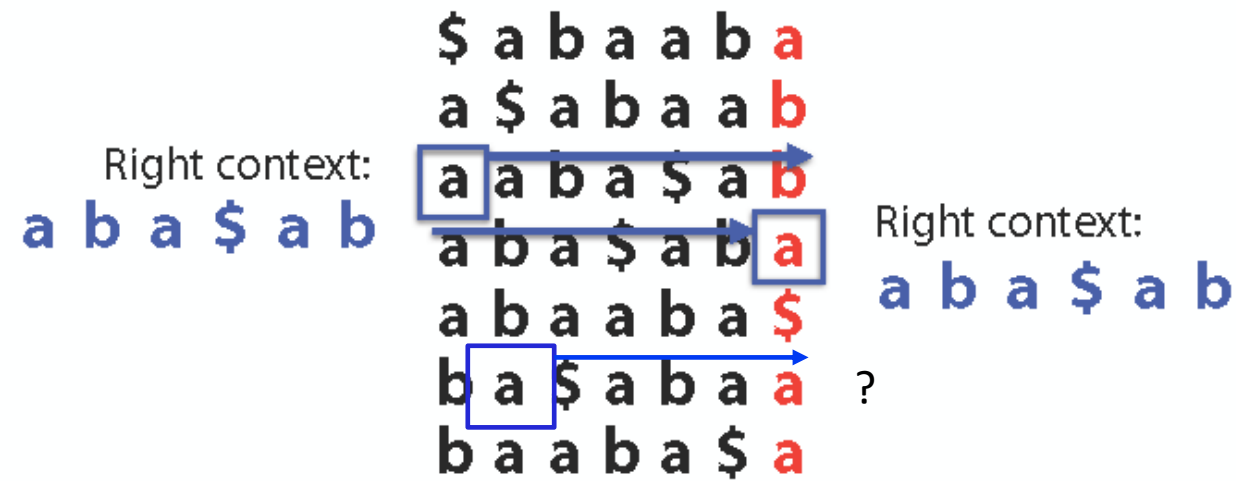
T : **a** b a a b a \$

Right context: **b a a b a \$**

T : a b a **a** b a \$

Right context: **b a \$ a b a**

Burrows-Wheeler Transform



BWT can be obtained from the Suffix Array

Burrows-Wheeler Transform

BWM is related to the suffix array

\$ 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

BWM(T)

6	\$
5	a \$
2	a a b a \$
3	a b a \$
0	a b a a b a \$
4	b a \$
1	b a a b a \$

SA(T)

Same order whether rows are rotations or suffixes

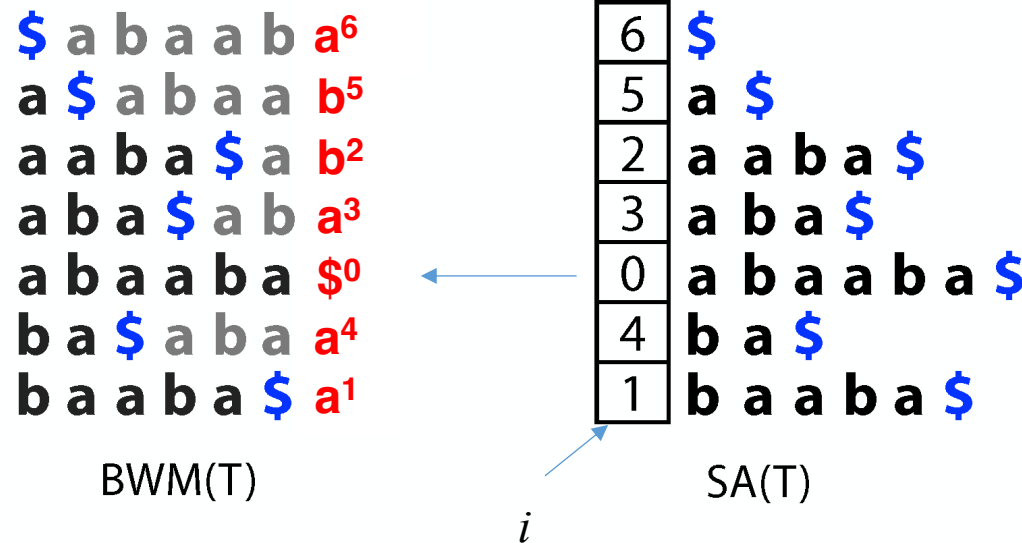
The suffix is the right-context

Burrows-Wheeler Transform

In fact, this gives us a new definition / way to construct BWT(T):

$$BWT[i] = \begin{cases} T[SA[i] - 1] & \text{if } SA[i] > 0 \\ \$ & \text{if } SA[i] = 0 \end{cases}$$

“BWT = characters just to the left of the suffixes in the suffix array”



i determines the position of the character *before* the suffix

FM Index

	F	T^{bwt}
mississippi#	# mississipp	i
ississippi#m	i #mississip	p
ssissippi#mi	i ppi#missis	s
sissippi#mis	i ssippi#mis	s
issippi#miss	i ssissippi#	m
ssippi#missi	m ississippi	#
sippi#missis	p i#mississi	p
ippi#mississ	p pi#mississ	i
ppi#mississi	s ippi#missi	s
pi#mississip	s issippi#mi	s
i#mississipp	s sippi#miss	i
#mississippi	s sissippi#m	i

Fig. 1. Example of Burrows-Wheeler transform for the string $T = \text{mississippi}$. The matrix on the right has the rows sorted in lexicographic order. The output of the BWT is the last column; in this example the string ipssm\#pissii .

Why is enough to store the First and Last Column
of Burrows Wheeler Matrix (BWM)?

Burrows-Wheeler Transform: T-ranking

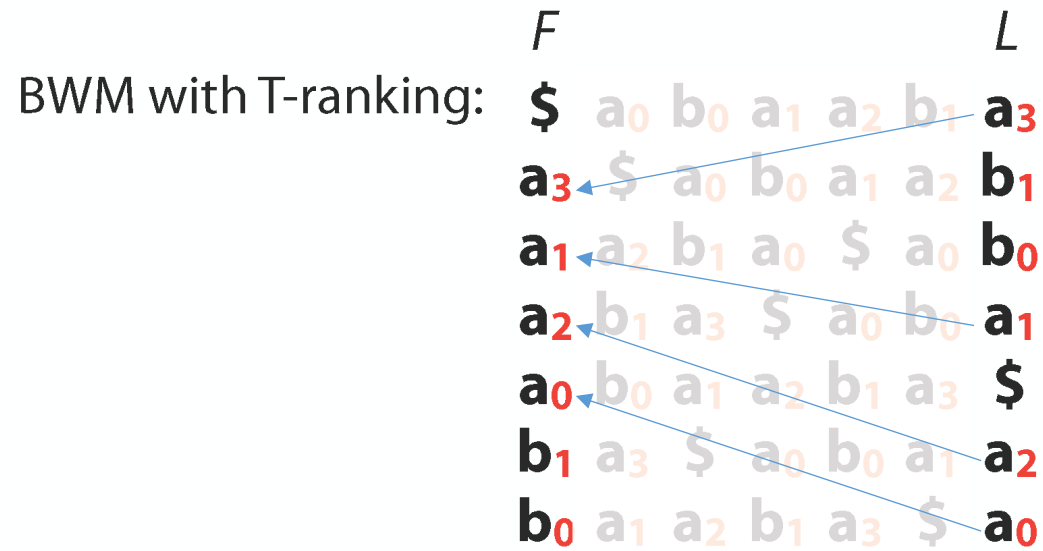
Give each character in T a rank, equal to # times the character occurred previously in T . Call this the T -ranking.

a₀ b₀ a₁ a₂ b₁ a₃ \$

Ranks aren't explicitly stored; they are just for illustration

Now let's re-write the BWM including ranks...

Burrows-Wheeler Transform: LF Mapping

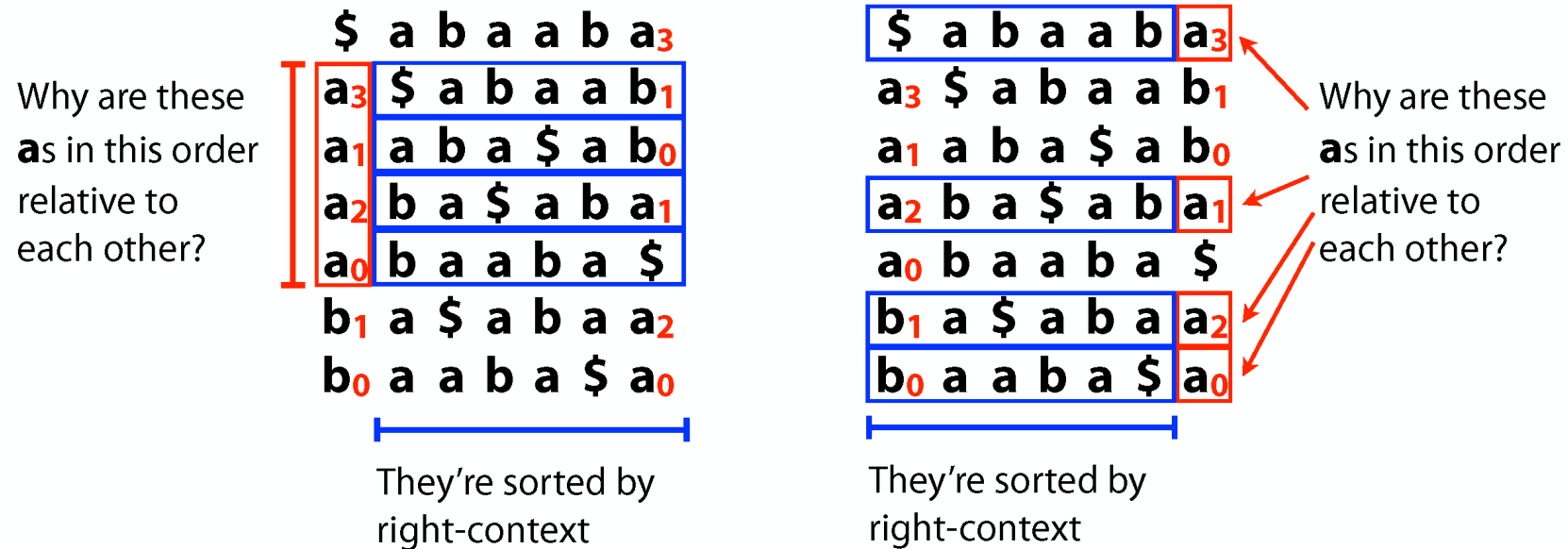


LF Mapping: The i^{th} occurrence of a character c in L and the i^{th} occurrence of c in F correspond to the *same* occurrence in T (i.e. have same rank)

However we rank occurrences of c , ranks appear in the same order in F & L

Burrows-Wheeler Transform: LF Mapping

Why does the LF Mapping hold?



Occurrences of c in F are sorted by right-context. Same for L !

Whatever ranking we give to characters in T , rank orders in F and L will match

FM Index

FM Index: an index combining the BWT *with a few small auxiliary data structures*

e.g. ranks

Core of index is ***F*** and ***L*** from BWM:

L is the same size as *T*

F can be represented as array of $|\Sigma|$ integers

L is compressible (but even uncompressed, it's small compared to suffix array)

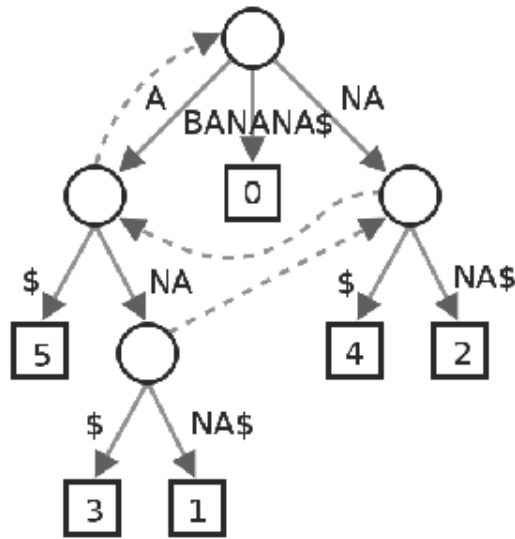
We can recreate the original string from right to left (LF-mapping property)

<i>F</i>		<i>L</i>	← L is BWT			
\$	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

└──────────┘
Not stored in index

Paolo Ferragina, and Giovanni Manzini. "Opportunistic data structures with applications." *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*. IEEE, 2000.

FM Index: small memory footprint



Suffix tree
 ≥ 45 GB

6	\$
5	A\$
3	ANA\$
1	ANANA\$
0	BANANA\$
4	NA\$
2	NANA\$

Suffix array
 ≥ 12 GB

\$ BANANA
A \$ BANAN
ANA \$ BAN
ANANA \$ B
BANANA \$
NA \$ BANA
NANA \$ BA

FM Index
~ 1.5 GB

Hands-on Exercises in Python

Let's put in practice what we have been learning Ex. 9.1:

- 1) Build Matrix BWM
- 2) Burrows-Wheeler BWT
- 3) Obtain FM Index from BWT

In addition, Ex. 9.2:

- 4) Algorithm for Run-Length Encoding of DNA Sequences