An Introduction of Burrows-Wheeler Transform (BWT)

and Its Variants

Wing-Kai Hon

39th Symposium on Combinatorial Mathematics and Computational Theory (2022/6/24)

Outline

- Pattern Matching & Text Indexing
- Suffix Tree and Suffix Array
- BWT
- 2BWT, Permuterm, XBW
- pBWT
- GBWT

Pattern Matching Problem

Basic Pattern Matching Problem

- Input:

  (1) a text T

  (2) a pattern P
- Output:

  (I) # of times P occurs in T

  (II) locations in T of where P occurs

Pattern Matching Problem

Basic Pattern Matching Problem

Example:

- Input:

  T banana

  P an

- Output:

  P occurs 2 times in T

  P occurs at positions 2 and 4

Text Indexing Problem

How good can we solve basic pattern matching?

- Denote $|T| = t$ and $|P| = p$
- KMP [Knuth & Pratt 70; Morris 70] [Knuth, Morris, Pratt 77]

processing: $O(t+p)$ time

Text Indexing Problem

Basic Text Indexing Problem

- Input:

  a text T

- Output:

  an index structure Δ to represent T

  such that

  given any query pattern P,

  we can solve pattern matching quickly

Text Indexing Problem

Basic Text Indexing Problem

- Key Observation:

  Each time P occurs in T, P occurs as the prefix of a distinct suffix of T

T	banana		T	banana
P	an		P	an

Text Indexing Problem

How good can we solve text indexing?

- Denote $|T| = t$ and $|P| = p$
- Suffix Tree [McCreight 76; Weiner 73]

  space: $O(t)$

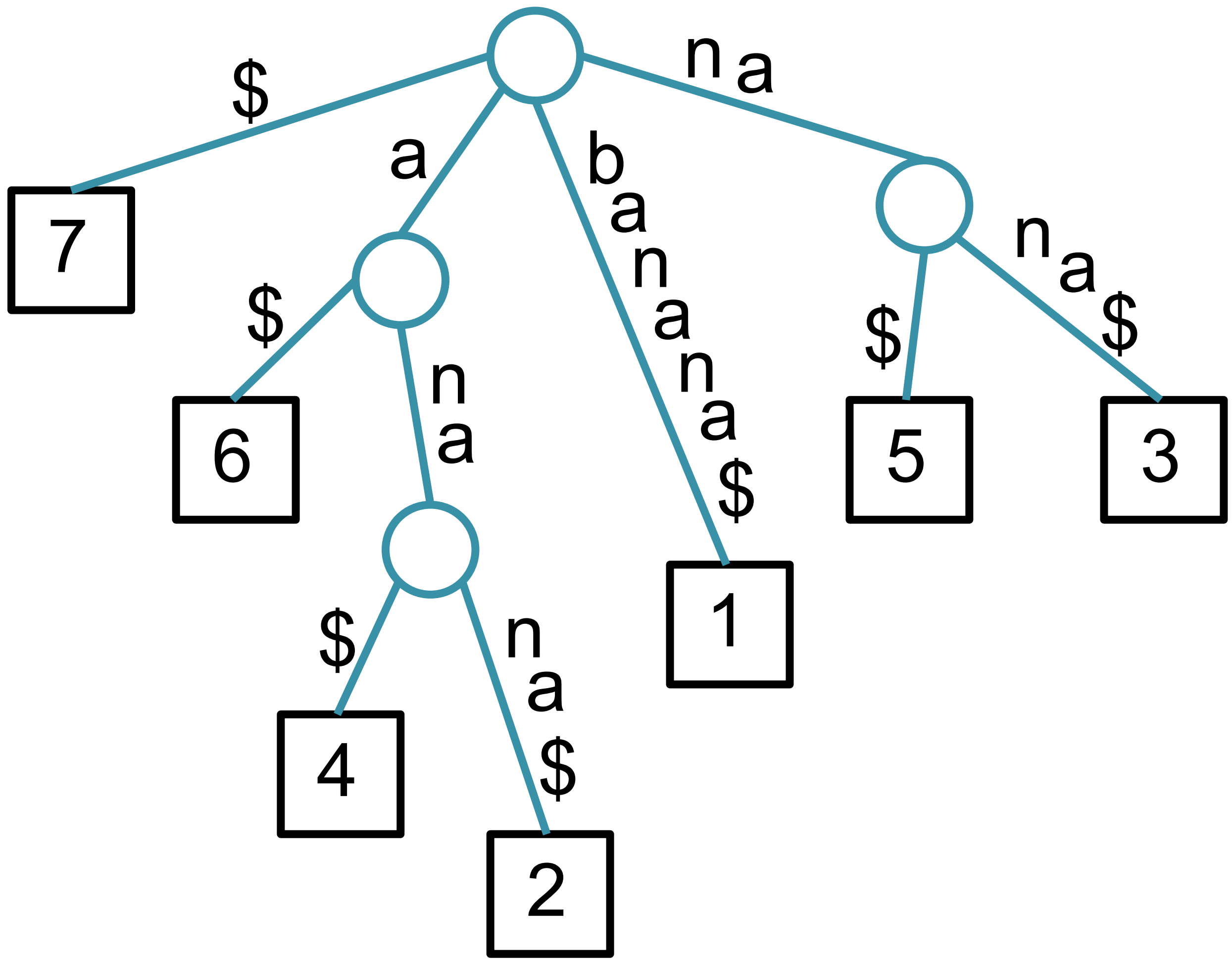
  query: $O(p+occ)$ time

- Suffix Array [Manber & Myers 93]

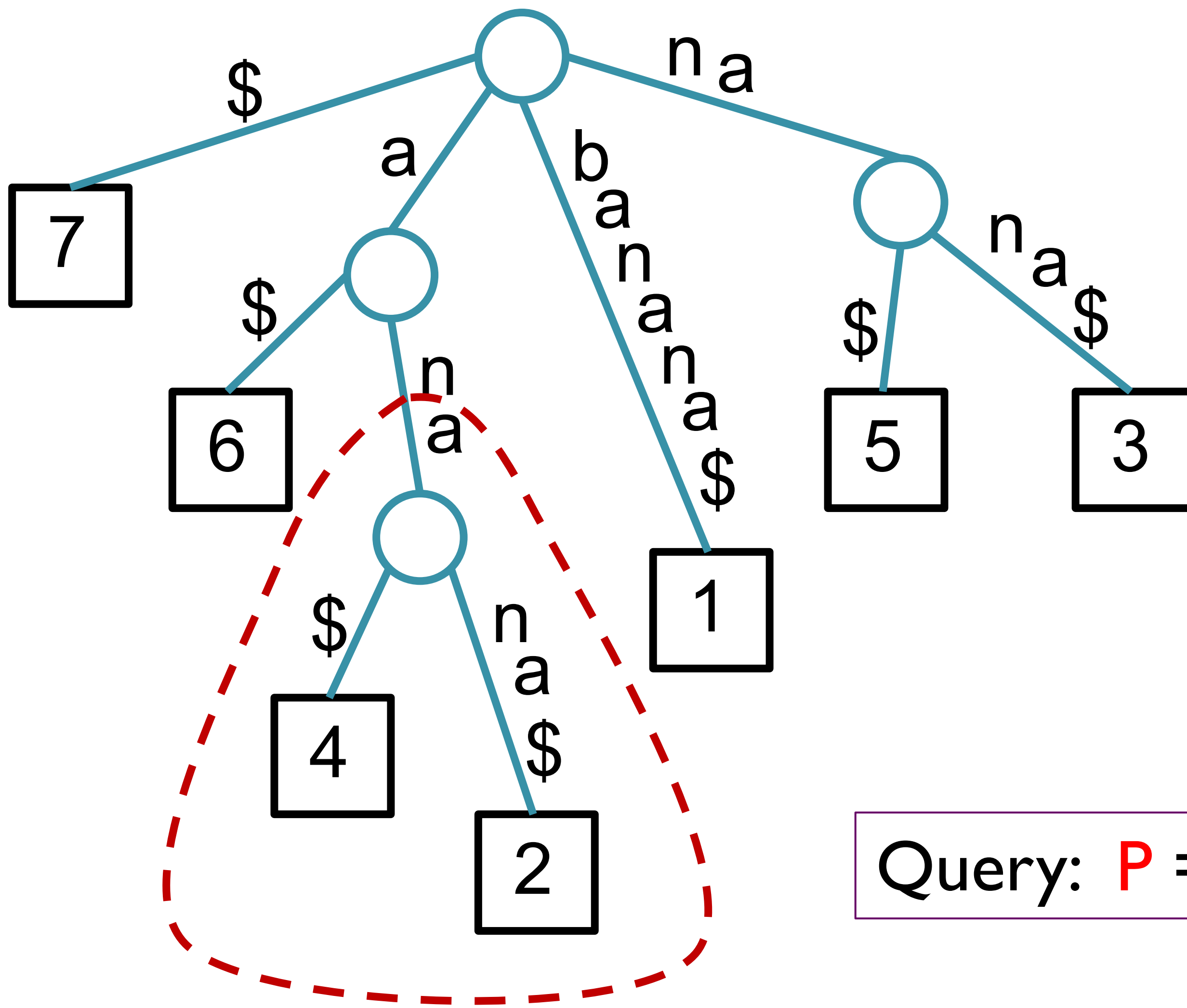
  space: $O(t)$

  query: $O(p + \log t + occ)$ time

Suffix Tree of banana$



Suffix Tree of banana$



Query: P = an

Suffix Array of banana$

j	SA[j]	suffixes of banana$						
1	7	$						
2	6	a	$					
3	4	a	n	a	$			
4	2	a	n	a	n	a	$	
5	1	b	a	n	a	n	a	$
6	5	n	a	$				
7	3	n	a	n	a	$		

Suffix Array of banana$

j	SA[j]	suffixes of banana$						
1	7	$						
2	6	a	$					
3	4	a	n	a	$			
4	2	a	n	a	n	a	$	
5	1	b	a	n	a	n	a	$
6	5	n	a	$				
7	3	n	a	n	a	$		

occurrences of P occupy a contiguous range in SA

➔ We call this the suffix range of P

Text Indexing Problem

Is Suffix Tree optimal?

- Σ = alphabet; $|\Sigma| = \sigma$
- Minimal space to represent T

  = t characters = $O(t \log \sigma)$ bits
- Suffix Tree of T

  = t integers = $O(t \log t)$ bits

Text Indexing Problem

Can we achieve optimal space?

- BWT [Burrows & Wheeler 94]

  space: $O(t \log \sigma)$ bits

  query: not supported

- BWT + ⓘ [Ferragina & Manzini 00]

  space: $O(t \log \sigma)$ bits

  query: $O(p \log \sigma + occ \log^{\varepsilon} t)$ time

BWT of banana$

j	BWT[j]	cyclic shifts of banana$						
1	a	$	b	a	n	a	n	a
2	n	a	$	b	a	n	a	n
3	n	a	n	a	$	b	a	n
4	b	a	n	a	n	a	$	b
5	$	b	a	n	a	n	a	$
6	a	n	a	$	b	a	n	a
7	a	n	a	n	a	$	b	a

BWT of banana$

j	BWT[j]	suffixes of banana$						
1	a	$	b	a	n	a	n	a
2	n	a	$	b	a	n	a	n
3	n	a	n	a	$	b	a	n
4	b	a	n	a	n	a	$	b
5	$	b	a	n	a	n	a	$
6	a	n	a	$	b	a	n	a
7	a	n	a	n	a	$	b	a

Some Properties of BWT

- a permutation of T
- Last-to-Front Mapping
  - reversible [Burrows & Wheeler 94]
  - searchable [Ferragina & Manzini 00]
- compressible [Manzini 01]

BWT is a permutation of T

j	BWT[j]	suffixes of banana$						
1	a	$						
2	n	a	$					
3	n	a	n	a	$			
4	b	a	n	a	n	a	$	
5	$	b	a	n	a	n	a	$
6	a	n	a	$				
7	a	n	a	n	a	$		

T = banana$

Last-to-Front Mapping

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

T = banana$

BWT is Reversible

j	BWT[j]	first character of suffixes						
1	a							
2	n							
3	n							
4	b							
5	$							
6	a							
7	a							

T = ???????

BWT is Reversible
(1. Get First Characters)

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

T = ???????

BWT is Reversible

(2. Get LF Mapping)

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

T = ???????

BWT is Reversible
(3. Retrieve Characters in Backward Manner)

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

T = ??????$

BWT is Reversible

(3. Retrieve Characters in Backward Manner)

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

T = ?????a$

BWT is Reversible

(3. Retrieve Characters in Backward Manner)

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

T = ????na$

BWT is Reversible

(3. Retrieve Characters in Backward Manner)

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

T = ???ana$

BWT is Reversible

(3. Retrieve Characters in Backward Manner)

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

T = ??nana$

BWT is Reversible

(3. Retrieve Characters in Backward Manner)

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

T = ?anana$

BWT is Reversible

(3. Retrieve Characters in Backward Manner)

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

T = banana$

BWT is Searchable

j	BWT[j]	first character of suffixes						
1	a							
2	n							
3	n							
4	b							
5	$							
6	a							
7	a							

P = nana

BWT is Searchable

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

P = ???a

BWT is Searchable

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

P = ??na

BWT is Searchable

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

P = ?ana

BWT is Searchable

j	BWT[j]	first character of suffixes						
1	a	$						
2	n	a						
3	n	a						
4	b	a						
5	$	b						
6	a	n						
7	a	n						

P = nana

BWT is Searchable

- Main Idea:
  If we know the suffix range of a pattern P, then we can obtain the suffix range of cP for any char c

- We call this backward search

BWT Real Applications

- Short Read Alignment Problem
  - Need to locate occurrences of numerous patterns in a very long genome
- Suffix Tree or Suffix Array take huge space (64G for Human DNA)
- BWT saves space (1G for Human DNA)

➔ Core index in BWA, Bowtie, SOAP2

Bi-directional BWT

- BWT searches backwardly
- Can it support forward search?
  - That is, given the suffix range of P, and a character c, can we get the suffix range of Pc?

Bi-directional BWT

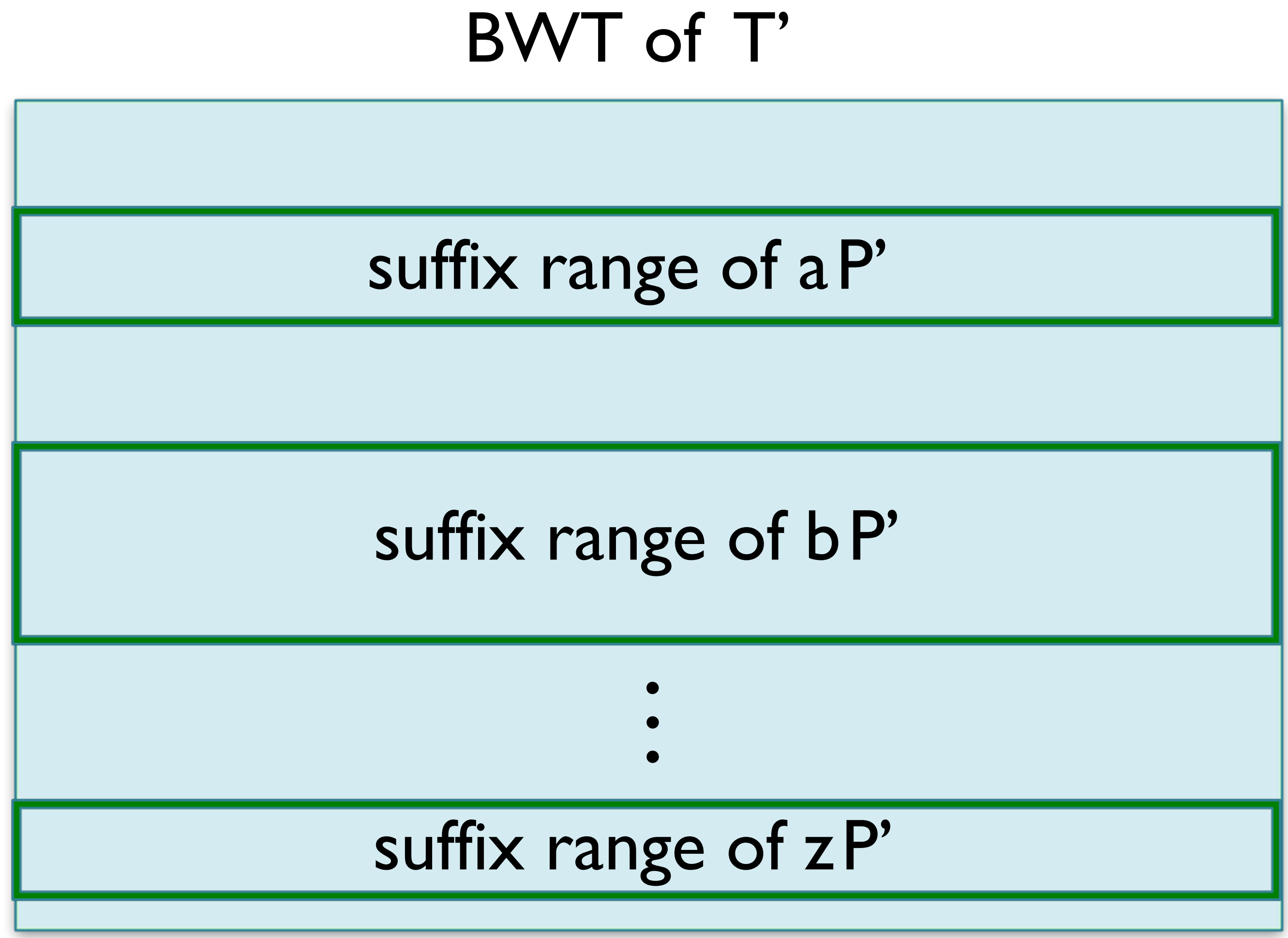
- If SA is provided, we can solve this with O(log t) accesses to SA
- Lam et al. (2009) suggested a simple but elegant solution:

  maintain two BWTs, one for T and the other for T' (the reverse of T)

Bi-directional BWT

- At any time, we keep track of the suffix range of P in T, and the suffix range of P' in T'
- Next, perform backward search in BWT of xP' for every character x

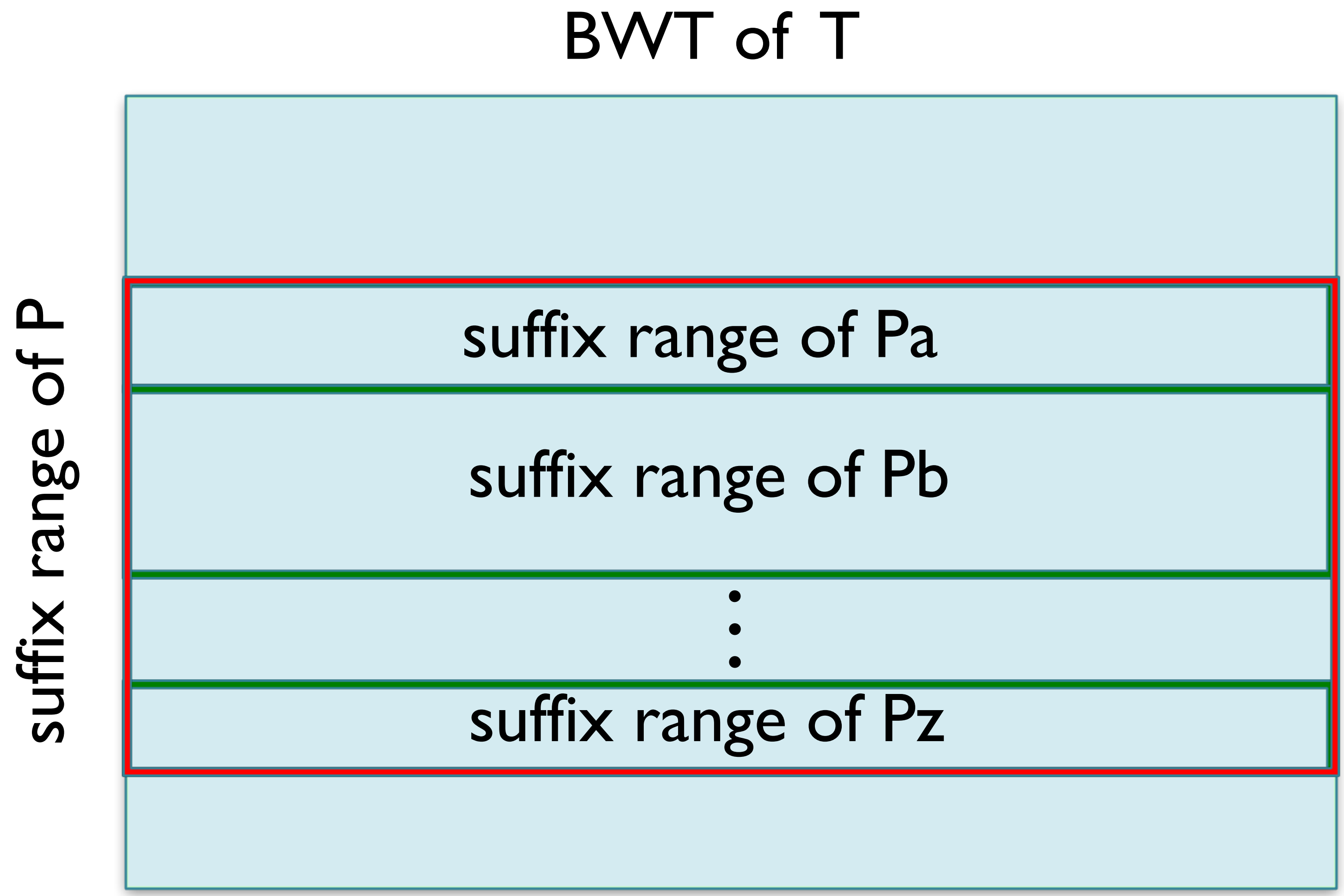
Bi-directional BWT



Bi-directional BWT

- After that, we get the number of times x P' occurring in T'
  ➔ same as number of times P x occurring in T
- Use the above to refine the suffix range of P in T to get the suffix range of P c in T

Bi-directional BWT



Bi-directional BWT

- Each forward search step takes O(σ) time
  - Recently improved to O(1) time by Belazzougui et al. (2014)
- Lam et al. implemented this, called 2BWT (a part of SOAP2), for locating short reads with small errors

Tolerant Retrieval Problem

- Input: A list L of m strings
- Query:

Given any query pattern of the form

P, P*, *P, P*Q, or *P*

we can locate the query pattern

in the strings of L (*= wildcard string)

Tolerant Retrieval Problem

- Ferragina and Venturini (2007) used a single BWT to index L so that all the queries can be supported
  - Only 1 line change in search method
- This is called Compressed Permuterm Index

XPath Query in XML Tree

- Input: A rooted tree X with labeled nodes
- Query:
  Given a query pattern of P, find all sub-paths in X such that the concatenation of the labels in the sub-paths matches P

XPath Query in XML Tree

- Naïve method: Maintain a separate BWT for the concatenated labels of each root-to-leaf path in X
- If each node v of X is represented by the lexicographical order of the `reverse' of the corresponding path labels, the BWTs can be merged and also searchable [Ferragina et al. 05]

XPath Query in XML Tree

- This is called XBW transform
- Can be applied to compress

  Aho-Corasick automaton

  for dictionary matching problem

  without any slowdown

  [Belazzougui 10; Hon et al. 10]

When Problems are Harder

- Parameterized Matching [Baker 93]
  - abba can match with yxxy
- Structural Matching [Shibuya 04] with focus on RNA strings
  - AUGCAA can match with GCAUGG
  - AUGCAA not match with GACUGG

Structural Match

= Parameterized Match + Complement Constraint

When Problems are Harder

- pBWT [Ganguly, Shah, Thankachan 17]
  - Based on Baker's encoding to transform each suffix of T into another string (so searching is efficient)
  - LF mapping of encoded suffixes becomes non-trivial

Text Indexing Problem (revisited)

Can we achieve optimal space?

- CSA [Grossi & Vitter 00; Sadakane 00]
- Many Improvements

ACM Comp Survey [Navarro & Makinen 07]

Open Problem

Can we achieve optimal space and optimal time simultaneously ?

Geometric BWT

- Hon et al. (2008) observed that one can reduce 2D orthogonal range searching into a text indexing
- This is called GBWT
- Leads to some lower bound result in compressed text indexing

Thanks for Listening

Questions?