Bidirectional Burrows-Wheeler Transform and (Approximate) Pattern Matching

StringBio 2018

University of Central Florida, USA

Arnab Ganguly
Department of Computer Science, UW - Whitewater

October 25, 2018

Pattern Matching

The Problem

Input: A text $T[1,n]$ and a pattern $P[1,p]$
Output: All positions in T where P appears as a substring

Text Indexing – Suffix Tree and Suffix Array

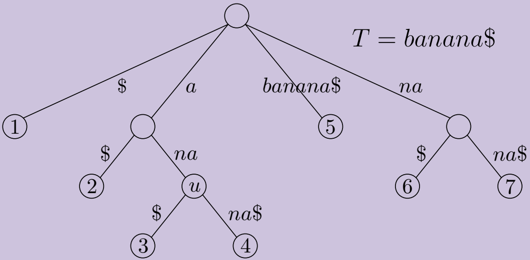
- Pre-process the text and create a data structure
- Answer queries using the data structure efficiently – avoid reading the text every time
- Suffix Trees and Suffix Arrays are the ubiquitous data structures for this purpose

We can report all occurrences in time $O(p + occ)$ after a one-time $O(n)$-time pre-processing

$$occ = \# \text{ of occurrences of } P \text{ in } T$$

Suffix Tree and Suffix Array

Pattern P appears at position i iff P is a prefix of the suffix $T[i, n]$



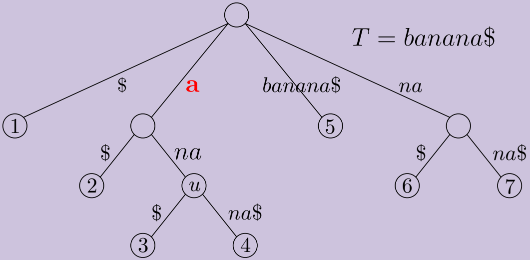
Leaves are arranged in lexicographic order of the corresponding suffixes

One leaf per suffix: number of suffixes = n = length of T

Number of nodes $< 2n$

Suffix Tree and Suffix Array

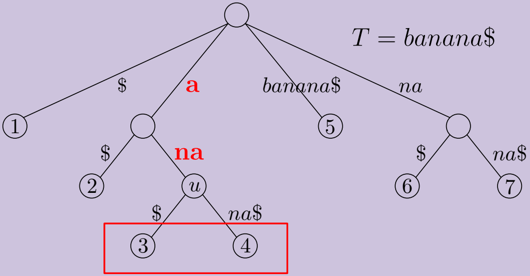
Searching with $P = ana$



i	1	2	3	4	5	6	7
SA[i]	7	6	4	2	1	5	3

Suffix Tree and Suffix Array

Searching with $P = ana$



i	1	2	3	4	5	6	7
SA[i]	7	6	4	2	1	5	3

Compressed Text Indexing

The Huge Space Problem

- The space occupied by suffix tree is $\Theta(n \log n)$ bits
- T occupies $n\lceil \log \sigma \rceil$ bits, where σ is the alphabet size

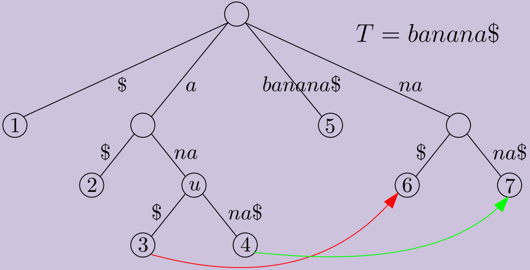
Too large for most practical purposes, such as for Human Genome ($\sigma = 4$ and $n \approx 3$ billion)
The Human Genome occupies space $\approx\ 1GB$
Suffix tree occupies space $\approx\ 40GB$

Answer

- Compressed Suffix Array [Grossi and Vitter, STOC' 00]
- FM Index [Ferragina and Manzini, FOCS' 00]

Space: $n \log \sigma + o(n)$ bits – close to the text
Time: $O((p + occ)\,\mathrm{poly}(\log n))$ – close to the suffix tree

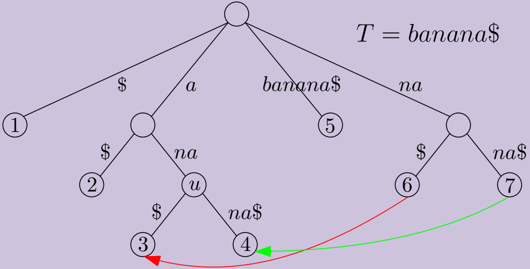
Suffix Links – Key Concept behind Succinct Indexing



Rank-preserving property

- Consider the two suffixes under u.
- Chop off the first character a.
- The suffixes preserve their relative rank: $[3, 4] \rightarrow [6, 7]$

Reverse Suffix Links



Rank-preserving property

- Consider any two suffixes with ranks i and j, and previous characters $\mathsf{BWT}[i]$ and $\mathsf{BWT}[j]$
- Let the rank of the suffixes obtained by prepending the previous characters be i' and j'.
- Then, $i' < j'$ iff $\mathsf{BWT}[i] < \mathsf{BWT}[j]$ or $\mathsf{BWT}[i] = \mathsf{BWT}[j]$ and $i < j$.

LF Mapping → Suffix Array

Definition

$\mathsf{LF}(i)$ is the lexicographic rank of the suffix starting at $\mathsf{SA}[i] - 1$

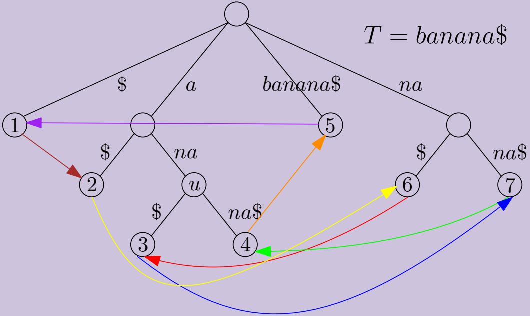
Sampled Suffix Array

- Explicitly store $\langle i, \mathsf{SA}[i] \rangle$ iff $\mathsf{SA}[i] \in \{1, 1 + \lceil \log n \rceil, 1 + 2\lceil \log n \rceil, \ldots, n\}$.
- The space needed is $O(n)$ bits

Computing $\mathsf{SA}[i]$

- If $i \in D$, retrieve $\mathsf{SA}[i]$
- Otherwise, let $i_1 = \mathsf{LF}(i), i_2 = \mathsf{LF}(i_1), \ldots, i_k = \mathsf{LF}(i_{k-1})$, where $i_k \in D$
- Then, $\mathsf{SA}[i_k] = \mathsf{SA}[i] - k \implies \mathsf{SA}[i] = \mathsf{SA}[i_k] + k$
- Since $k \leq \lceil \log n \rceil$, time needed is $O(t_{\mathsf{LF}} \cdot \log n)$

BWT → LF Mapping



BWT[i] =	1	2	3	4	5	6	7
	a	n	n	b	\$	a	a

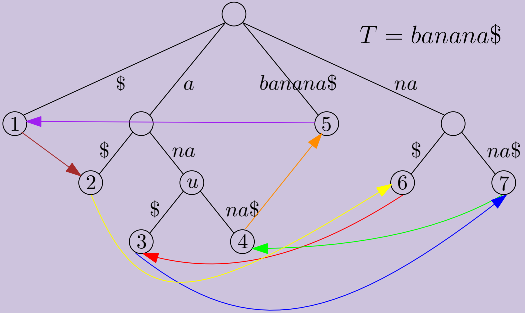
$$\text{LF}(i) = \text{num of } j \text{ with } \text{BWT}[j] \prec \text{BWT}[i] + \text{num of } j \text{ with } \text{BWT}[j] = \text{BWT}[i] \text{ and } j \leq i$$

$$= \text{count}(1, n, < \text{BWT}[i]) + \text{count}(1, i, = \text{BWT}[i])$$

For e.g., $\text{LF}(3) = 5 + 2 = 7$ and $\text{LF}(6) = 1 + 2 = 3$

BWT → Suffix Range (Backward Search)

Main Idea



$\mathsf{BWT}[i]$ =

1	2	3	4	5	6	7
a	n	n	b	$	a	a

If the suffix range of P is $[L, R]$, then

- the size of the suffix range of xP is the number of $i \in [L, R]$ such that $\mathsf{BWT}[i] = x$.
- the suffix range of xP STARTS at $1 + \mathsf{count}(1, n, < P[i]) + \mathsf{count}(1, L-1, = P[i])$
- the suffix range of xP ENDS at $1 + \mathsf{count}(1, n, < P[i]) + \mathsf{count}(1, R, = P[i])$

Approximate Pattern Matching

The Problem

Input: A text $T[1,n]$ and a pattern $P[1,p]$
Output: All positions in T where P appears as a substring with at most k mismatches (i.e., Hamming distance $\leq k$)

The Obvious Approach

- Try every position i in T and check whether the number of mismatches at this position is at most k. If yes, then report i, else do not report i.
- Complexity is $O(pn)$, which is too high for most practical purposes.
- Landau and Vishkin [Journal of Algorithms' 89] gives an $O(nk)$ time algorithm

k-errata Suffix Tree

Cole, Gottlieb, and Lewenstein [STOC' 04] presents an $O(n \log^k n)$-space data structure with a query time of $O(p + \log^k n + occ)$ query time, assuming $k = \Theta(1)$.

What about BWT based approaches?

The Overall Idea for 1-mismatch

- Split the pattern $P[1, p]$ into two equal parts $P[1, p/2]$ and $P[p/2 + 1]$
- Mismatch can be either in first part or second part
- To find mismatch in first part, do the following:
  - Backward search the second part to find suffix range of $P[p/2 + 1, p]$
  - Now, for $i = p/2, p/2 - 1, \dots, 1$
    - find the suffix range of $P[i] \circ P[i + 1, p]$
    - find the suffix range of $P[1, i - 1] \circ x \circ P[i + 1, p]$ for every $x \in \Sigma \setminus P[i]$ and report occurrences from the non-empty suffix ranges.
- To find mismatch in second part, do the following:
  - Forward search the first part to find suffix range of $P[1, p/2]$
  - Now, for $i = p/2 + 1, p/2 + 2, \dots, p$
    - find the suffix range of $P[1, i - 1] \circ P[i]$
    - find the suffix range of $P[1, i - 1] \circ x \circ P[i + 1, p]$ for every $x \in \Sigma \setminus P[i]$ and report occurrences from the non-empty suffix ranges.

The Bidirectional BWT [Lam et al., BIBM' 09]

- Maintain two separate BWTs – BWT for T and BWT^r for T^r
- Store the sampled suffix array for T

Given the suffix range of P and some $x \in \Sigma$, our task is the following:

- compute the suffix range of xP – backward search using BWT – EASY!
- compute the suffix range of Px

Computing the suffix range of Px – Main Idea

- Let suffix range of P be $[L, R]$. Note that the suffix range of Px is a sub range of $[L, R]$
- Let α be the number of suffixes that are prefixed by Pw, where $w \in \Sigma$ is lexicographically smaller than x
- Let β be the number of suffixes that are prefixed by Px
- Then, the suffix range of Px is $[L + \alpha, L + \alpha + \beta - 1]$

How to compute α and β?

The Bidirectional BWT [Lam et al., BIBM' 09]

- Maintain two separate BWTs – BWT for T and BWT^r for T^r
- Store the sampled suffix array for T

Given the suffix range of P and some $x \in \Sigma$, our task is the following:

- compute the suffix range of xP – backward search using BWT – EASY!
- compute the suffix range of Px

Computing the suffix range of Px – Main Idea

- Let suffix range of P be $[L, R]$. Note that the suffix range of Px is a sub range of $[L, R]$
- Let α be the number of suffixes that are prefixed by Pw, where $w \in \Sigma$ is lexicographically smaller than x
- Let β be the number of suffixes that are prefixed by Px
- Then, the suffix range of Px is $[L + \alpha, L + \alpha + \beta - 1]$

How to compute α and β?

The Bidirectional BWT [Lam et al., BIBM' 09]

- Maintain two separate BWTs – BWT for T and BWT^r for T^r
- Store the sampled suffix array for T

Given the suffix range of P and some $x \in \Sigma$, our task is the following:

- compute the suffix range of xP – backward search using BWT – EASY!
- compute the suffix range of Px

Computing the suffix range of Px – Main Idea

- Let suffix range of P be $[L, R]$. Note that the suffix range of Px is a sub range of $[L, R]$
- Let α be the number of suffixes that are prefixed by Pw, where $w \in \Sigma$ is lexicographically smaller than x
- Let β be the number of suffixes that are prefixed by Px
- Then, the suffix range of Px is $[L + \alpha, L + \alpha + \beta - 1]$

How to compute α and β?

Computing α and β

- Note that the suffix range of any string Y w.r.t T has the same size as that of Y^r w.r.t T^r
- Therefore, given the suffix range $[L, R]$ of P^r w.r.t T^r

To compute α

Compute the total size of the suffix ranges of $(Pw)^r$ using a backward search via BWT^r for every w lexicographically smaller than x

$$\mathsf{count}^r(L, R, < x)$$

To compute β

Compute the size of the suffix range of $(Px)^r$ using a backward search via BWT^r

$$\mathsf{count}^r(L, R, = x)$$

Closing Remarks

- For 2-mismatches, split pattern into roughly 3 equal parts.
- Consider 6 cases – 101, 011, 110, 200, 020, 002.
- Works for higher number of mismatches.
- Can be made to work for edit-distance with slight modifications.
- Can be used to interleave backward and forward searches.

Thank you! Questions?