# Computing Matching Statistics on Wheeler DFAs

Alessio Conte[1] Nicola Cotumaccio[2,3] Travis Gagie[3]
Giovanni Manzini[1] Nicola Prezza[4] Marinella Sciortino[5]

[1]University of Pisa, Italy

[2]GSSI, L'Aquila, Italy

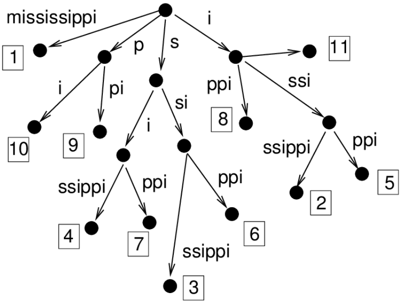
[3]Dalhousie University, Halifax, Canada

[4]Ca' Foscari University, Venice, Italy

[5]University of Palermo, Italy

# Suffix trees

The suffix tree[1] of a string is a general-purpose data structure which is able to efficiently handle a variety of problems (pattern matching, approximate pattern matching, shortest/longest substrings with some desired properties, palindromes, and so on).




[1] P. Weiner, *Linear Pattern Matching Algorithms*, FOCS (SWAT) 1973.

# Suffix trees

- The suffix tree of a string can be compressed by exploiting the repetitiveness of the string[2].
- It is also possible to build (variants of) suffix trees which are able to solve pattern matching queries where some variables are fixed up to a permutation[3].

---

[2] T. Gagie, G. Navarro, N. Prezza, *Fully Functional Suffix Trees and Optimal Text Searching in BWT-Runs Bounded Space*, JACM 2020.

[3] B. S. Baker, *A Theory of Parametrized Pattern Matching: Algorithms and Applications*, STOC 1993.

# Suffix trees

- The main limitation of suffix trees is their space consumption.
- As a consequence, suffix trees have been replaced with *suffix arrays*.
- Suffix arrays allow to solve pattern matching queries, but they do not have the whole functionality of suffix trees.

| $i$ | Sorted suffixes | $SA[i]$ |
|---|---|---|
| 1 | $ | 12 |
| 2 | i$ | 11 |
| 3 | ippi$ | 8 |
| 4 | issippi$ | 5 |
| 5 | ississippi$ | 2 |
| 6 | mississippi$ | 1 |
| 7 | pi$ | 10 |
| 8 | ppi$ | 9 |
| 9 | sippi$ | 7 |
| 10 | sissippi$ | 4 |
| 11 | ssippi$ | 6 |
| 12 | ssissippi$ | 3 |

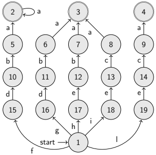
# Suffix trees

- Nonetheless, if a suffix array is augmented with the tree topology and the *longest common prefix (LCP) array*, then one retrieves the whole functionality of a suffix tree.
- All these data structures can be compressed, thus leading to *compressed suffix trees*.

| $i$ | Sorted suffixes | $LCP[i]$ |
|---|---|---|
| 1 | $ | |
| 2 | i$ | 0 |
| 3 | ippi$ | 1 |
| 4 | issippi$ | 1 |
| 5 | ississippi$ | 4 |
| 6 | mississippi$ | 0 |
| 7 | pi$ | 0 |
| 8 | ppi$ | 1 |
| 9 | sippi$ | 0 |
| 10 | sissippi$ | 2 |
| 11 | ssippi$ | 1 |
| 12 | ssissippi$ | 3 |

# From strings to graphs

- In the meanwhile, the notion of suffix array has been generalized from strings to trees[4], Wheeler graphs[5] and arbitrary labeled graphs [6].
- Since the functionality of a suffix tree can be simulated starting from a suffix array, the natural question is whether it is possible to design suffix trees of graphs.




---

[4] P. Ferragina, F. Luccio, G. Manzini, S. Muthukrishnan, *Structuring labeled trees for optimal succinctness, and beyond*, FOCS 2005.

[5] T. Gagie, G. Manzini, J. Sirén, *Wheeler graphs: A framework for BWT-based data structures*, TCS 2017.

[6] N. Cotumaccio, N. Prezza, *On indexing and compressing finite automata*, SODA 2021.

# Suffix trees of graphs

- In the absence of suffix trees for graphs, some authors have proposed alternative approaches (such as the direct product of graphs), but none of them provides the same flexibility and generality of suffix trees.
- Suffix trees of strings were designed well before the modern development of the data compression field, and the invention of suffix trees of graphs, while elicited by the research on suffix arrays, would have an impact which would go far beyond the applications to compression.
- At the same time, the journey leading to suffix trees of graphs is expected to provide a deeper insight into data compression concepts and techniques (entropy, Lempel-Ziv factorization,...) and their interpretation and application in graph theory.

# Our contribution

- The main contribution of our paper is to provide a first clear step towards extending suffix tree functionality to labeled graphs.
- Since in order to simulate the suffix tree of a string we need not only a suffix array, but also an LCP array, we show how to define the LCP array *of a graph*.
- We actually focus on (deterministic) Wheeler graphs (or equivalently, automata), because Wheeler graphs best capture the intuition behind suffix arrays in a graph setting, thus they are expected to be the crucial class of graphs one should work with.

# Our contribution

- A classical and useful problem that can be solved using the suffix tree of a string (and in particular the LCP array of a string) is the problem of determining the *matching statistics of a given pattern w.r.t the string*.
- We test our definition of the LCP array for a Wheeler graph by showing that the LCP array can be effectively and efficiently used to determine the *matching statistics of a given pattern w.r.t the Wheeler graph*, which is the natural generalization of the matching statistics problem from a string setting to a graph setting.
- More precisely, we generalize an algorithm[7] for determining matching statistics from a string setting to a graph setting.

[7] E. Ohlebusch, S. Gog, A. Kügell, *Computing matching statistics and maximal exact matches on compressed full-text indexes*, SPIRE 2010.

# Matching statistics of a string $\pi$ w.r.t a string $T$

$$T = mississippi\$$$
$$\pi = stpissi$$

1. $\ell_1 = 1$, $[l_1, r_1] = [9, 12]$.
2. $\ell_2 = 0$, $[l_2, r_2] = [1, 12]$.
3. $\ell_3 = 2$, $[l_3, r_3] = [7, 7]$.
4. $\ell_4 = 4$, $[l_4, r_4] = [4, 5]$.
5. $\ell_5 = 3$, $[l_5, r_5] = [11, 12]$.
6. $\ell_6 = 2$, $[l_6, r_6] = [9, 10]$.
7. $\ell_7 = 1$, $[l_7, r_7] = [2, 5]$.

| $i$ | Sorted suffixes |
|---|---|
| 1 | $ |
| 2 | i$ |
| 3 | ippi$ |
| 4 | issippi$ |
| 5 | ississippi$ |
| 6 | mississippi$ |
| 7 | pi$ |
| 8 | ppi$ |
| 9 | sippi$ |
| 10 | sissippi$ |
| 11 | ssippi$ |
| 12 | ssissippi$ |

# $T = mississippi\$$, $\pi = stpissi$

We start from the end of $\pi$, and we repeatedly apply a backward search step (using the FM-index, which relies on the Burrows-Wheeler transform of the string).

1. $\ell_1 = 1$, $[l_1, r_1] = [9, 12]$.
2. $\ell_2 = 0$, $[l_2, r_2] = [1, 12]$.
3. $\ell_3 = 2$, $[l_3, r_3] = [7, 7]$.
4. $\ell_4 = 4$, $[l_4, r_4] = [4, 5]$.
5. $\ell_5 = 3$, $[l_5, r_5] = [11, 12]$.
6. $\ell_6 = 2$, $[l_6, r_6] = [9, 10]$.
7. $\ell_7 = 1$, $[l_7, r_7] = [2, 5]$.

| $i$ | Sorted suffixes |
|---|---|
| 1 | $ |
| 2 | i$ |
| 3 | ippi$ |
| 4 | issippi$ |
| 5 | ississippi$ |
| 6 | mississippi$ |
| 7 | pi$ |
| 8 | ppi$ |
| 9 | sippi$ |
| 10 | sissippi$ |
| 11 | ssippi$ |
| 12 | ssissippi$ |

## $T = mississippi\$$, $\pi = stpissi$

$\pi = stpissi$

1. $\ell_1 = 1$, $[l_1, r_1] = [9, 12]$.
2. $\ell_2 = 0$, $[l_2, r_2] = [1, 12]$.
3. $\ell_3 = 2$, $[l_3, r_3] = [7, 7]$.
4. $\ell_4 = 4$, $[l_4, r_4] = [4, 5]$.
5. $\ell_5 = 3$, $[l_5, r_5] = [11, 12]$.
6. $\ell_6 = 2$, $[l_6, r_6] = [9, 10]$.
7. $\ell_7 = 1$, $[l_7, r_7] = [2, 5]$.

| $i$ | Sorted suffixes | $LCP[i]$ | $SA[i]$ | $BWT[i]$ |
|---|---|---|---|---|
| 1 | $ | | 12 | i |
| 2 | i$ | 0 | 11 | p |
| 3 | ippi$ | 1 | 8 | s |
| 4 | issippi$ | 1 | 5 | s |
| 5 | ississippi$ | 4 | 2 | m |
| 6 | mississippi$ | 0 | 1 | $ |
| 7 | pi$ | 0 | 10 | p |
| 8 | ppi$ | 1 | 9 | i |
| 9 | sippi$ | 0 | 7 | s |
| 10 | sissippi$ | 2 | 4 | s |
| 11 | ssippi$ | 1 | 6 | i |
| 12 | ssissippi$ | 3 | 3 | i |

# $T = mississippi\$$, $\pi = stpissi$

$\pi = stpissi$

1. $\ell_1 = 1$, $[l_1, r_1] = [9, 12]$.
2. $\ell_2 = 0$, $[l_2, r_2] = [1, 12]$.
3. $\ell_3 = 2$, $[l_3, r_3] = [7, 7]$.
4. $\ell_4 = 4$, $[l_4, r_4] = [4, 5]$.
5. $\ell_5 = 3$, $[l_5, r_5] = [11, 12]$.
6. $\ell_6 = 2$, $[l_6, r_6] = [9, 10]$.
7. $\ell_7 = 1$, $[l_7, r_7] = [2, 5]$.

| $i$ | Sorted suffixes | $LCP[i]$ | $SA[i]$ | $BWT[i]$ |
|---|---|---|---|---|
| 1 | $ | | 12 | i |
| 2 | i$ | 0 | 11 | p |
| 3 | ippi$ | 1 | 8 | s |
| 4 | issippi$ | 1 | 5 | s |
| 5 | ississippi$ | 4 | 2 | m |
| 6 | mississippi$ | 0 | 1 | $ |
| 7 | pi$ | 0 | 10 | p |
| 8 | ppi$ | 1 | 9 | i |
| 9 | sippi$ | 0 | 7 | s |
| 10 | sissippi$ | 2 | 4 | s |
| 11 | ssippi$ | 1 | 6 | i |
| 12 | ssissippi$ | 3 | 3 | i |

# $T = mississippi\$$, $\pi = stpissi$

$\pi = stpi\,ssi$

1. $\ell_1 = 1$, $[l_1, r_1] = [9, 12]$.
2. $\ell_2 = 0$, $[l_2, r_2] = [1, 12]$.
3. $\ell_3 = 2$, $[l_3, r_3] = [7, 7]$.
4. $\ell_4 = 4$, $[l_4, r_4] = [4, 5]$.
5. $\ell_5 = 3$, $[l_5, r_5] = [11, 12]$.
6. $\ell_6 = 2$, $[l_6, r_6] = [9, 10]$.
7. $\ell_7 = 1$, $[l_7, r_7] = [2, 5]$.

| $i$ | Sorted suffixes | $LCP[i]$ | $SA[i]$ | $BWT[i]$ |
|---|---|---|---|---|
| 1 | \$ | | 12 | i |
| 2 | i\$ | 0 | 11 | p |
| 3 | ippi\$ | 1 | 8 | s |
| 4 | issippi\$ | 1 | 5 | s |
| 5 | ississippi\$ | 4 | 2 | m |
| 6 | mississippi\$ | 0 | 1 | \$ |
| 7 | pi\$ | 0 | 10 | p |
| 8 | ppi\$ | 1 | 9 | i |
| 9 | sippi\$ | 0 | 7 | s |
| 10 | sissippi\$ | 2 | 4 | s |
| 11 | ssippi\$ | 1 | 6 | i |
| 12 | ssissippi\$ | 3 | 3 | i |

# $T = mississippi\$,\ \pi = stpissi$

$\pi = stpissi$

1. $\ell_1 = 1,\ [l_1, r_1] = [9, 12].$
2. $\ell_2 = 0,\ [l_2, r_2] = [1, 12].$
3. $\ell_3 = 2,\ [l_3, r_3] = [7, 7].$
4. $\ell_4 = 4,\ [l_4, r_4] = [4, 5].$
5. $\ell_5 = 3,\ [l_5, r_5] = [11, 12].$
6. $\ell_6 = 2,\ [l_6, r_6] = [9, 10].$
7. $\ell_7 = 1,\ [l_7, r_7] = [2, 5].$

| $i$ | Sorted suffixes | $LCP[i]$ | $SA[i]$ | $BWT[i]$ |
|---|---|---|---|---|
| 1 | $ |  | 12 | i |
| 2 | i$ | 0 | 11 | p |
| 3 | ippi$ | 1 | 8 | s |
| 4 | issippi$ | 1 | 5 | s |
| 5 | ississippi$ | 4 | 2 | m |
| 6 | mississippi$ | 0 | 1 | $ |
| 7 | pi$ | 0 | 10 | p |
| 8 | ppi$ | 1 | 9 | i |
| 9 | sippi$ | 0 | 7 | s |
| 10 | sissippi$ | 2 | 4 | s |
| 11 | ssippi$ | 1 | 6 | i |
| 12 | ssissippi$ | 3 | 3 | i |

# $T = mississippi\$$, $\pi = stpissi$

$\pi = stpissi$

1. $\ell_1 = 1$, $[l_1, r_1] = [9, 12]$.
2. $\ell_2 = 0$, $[l_2, r_2] = [1, 12]$.
3. $\ell_3 = 2$, $[l_3, r_3] = [7, 7]$.
4. $\ell_4 = 4$, $[l_4, r_4] = [4, 5]$.
5. $\ell_5 = 3$, $[l_5, r_5] = [11, 12]$.
6. $\ell_6 = 2$, $[l_6, r_6] = [9, 10]$.
7. $\ell_7 = 1$, $[l_7, r_7] = [2, 5]$.

| $i$ | Sorted suffixes | $LCP[i]$ | $SA[i]$ | $BWT[i]$ |
|---|---|---|---|---|
| 1 | \$ | | 12 | i |
| 2 | i\$ | 0 | 11 | p |
| 3 | ippi\$ | 1 | 8 | s |
| 4 | issippi\$ | 1 | 5 | s |
| 5 | ississippi\$ | 4 | 2 | m |
| 6 | mississippi\$ | 0 | 1 | \$ |
| 7 | pi\$ | 0 | 10 | p |
| 8 | ppi\$ | 1 | 9 | i |
| 9 | sippi\$ | 0 | 7 | s |
| 10 | sissippi\$ | 2 | 4 | s |
| 11 | ssippi\$ | 1 | 6 | i |
| 12 | ssissippi\$ | 3 | 3 | i |

The backward search fails: "pissi" does not occur in $T = mississippi\$$.

# $T = mississippi\$$, $\pi = stpissi$

We use the LCP array to determine the longest prefix of "issi" with *more* occurrences than "issi" in $T$.

$\pi = stpissi$

1. $\ell_1 = 1$, $[l_1, r_1] = [9, 12]$.
2. $\ell_2 = 0$, $[l_2, r_2] = [1, 12]$.
3. $\ell_3 = 2$, $[l_3, r_3] = [7, 7]$.
4. $\ell_4 = 4$, $[l_4, r_4] = [4, 5]$.
5. $\ell_5 = 3$, $[l_5, r_5] = [11, 12]$.
6. $\ell_6 = 2$, $[l_6, r_6] = [9, 10]$.
7. $\ell_7 = 1$, $[l_7, r_7] = [2, 5]$.

| $i$ | Sorted suffixes | $LCP[i]$ | $SA[i]$ | $BWT[i]$ |
|---|---|---|---|---|
| 1 | $ |  | 12 | i |
| 2 | i$ | 0 | 11 | p |
| 3 | ippi$ | 1 | 8 | s |
| 4 | issippi$ | 1 | 5 | s |
| 5 | ississippi$ | 4 | 2 | m |
| 6 | mississippi$ | 0 | 1 | $ |
| 7 | pi$ | 0 | 10 | p |
| 8 | ppi$ | 1 | 9 | i |
| 9 | sippi$ | 0 | 7 | s |
| 10 | sissippi$ | 2 | 4 | s |
| 11 | ssippi$ | 1 | 6 | i |
| 12 | ssissippi$ | 3 | 3 | i |

The desired prefix is the one of length $\max\{1, 0\} = 1$, that is, "i".

# $T = mississippi\$$, $\pi = stpissi$

We now extend[8] the interval $[4, 5]$ (suffixes starting with "issi") to $[2, 5]$ (suffixes starting with "i").

$\pi = st\,pi\cancel{ssi}$

1. $\ell_1 = 1$, $[l_1, r_1] = [9, 12]$.
2. $\ell_2 = 0$, $[l_2, r_2] = [1, 12]$.
3. $\ell_3 = 2$, $[l_3, r_3] = [7, 7]$.
4. $\ell_4 = 4$, $[l_4, r_4] = [4, 5]$.
5. $\ell_5 = 3$, $[l_5, r_5] = [11, 12]$.
6. $\ell_6 = 2$, $[l_6, r_6] = [9, 10]$.
7. $\ell_7 = 1$, $[l_7, r_7] = [2, 5]$.

| $i$ | Sorted suffixes | $LCP[i]$ | $SA[i]$ | $BWT[i]$ |
|---|---|---|---|---|
| 1 | $ | | 12 | i |
| 2 | i$ | 0 | 11 | p |
| 3 | ippi$ | 1 | 8 | s |
| 4 | issippi$ | 1 | 5 | s |
| 5 | ississippi$ | 4 | 2 | m |
| 6 | mississippi$ | 0 | 1 | $ |
| 7 | pi$ | 0 | 10 | p |
| 8 | ppi$ | 1 | 9 | i |
| 9 | sippi$ | 0 | 7 | s |
| 10 | sissippi$ | 2 | 4 | s |
| 11 | ssippi$ | 1 | 6 | i |
| 12 | ssissippi$ | 3 | 3 | i |

[8] There exists a data structure efficiently supporting such an extension.

# $T = mississippi\$$, $\pi = stpissi$

We try again to perform a backward search step.

$\pi = st\,pi\cancel{ssi}$

1. $\ell_1 = 1$, $[l_1, r_1] = [9, 12]$.
2. $\ell_2 = 0$, $[l_2, r_2] = [1, 12]$.
3. $\ell_3 = 2$, $[l_3, r_3] = [7, 7]$.
4. $\ell_4 = 4$, $[l_4, r_4] = [4, 5]$.
5. $\ell_5 = 3$, $[l_5, r_5] = [11, 12]$.
6. $\ell_6 = 2$, $[l_6, r_6] = [9, 10]$.
7. $\ell_7 = 1$, $[l_7, r_7] = [2, 5]$.

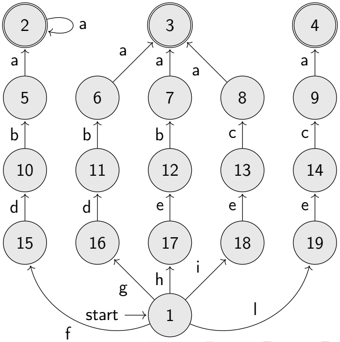
| $i$ | Sorted suffixes | $LCP[i]$ | $SA[i]$ | $BWT[i]$ |
|---|---|---|---|---|
| 1 | $ | | 12 | i |
| 2 | i$ | 0 | 11 | p |
| 3 | ippi$ | 1 | 8 | s |
| 4 | issippi$ | 1 | 5 | s |
| 5 | ississippi$ | 4 | 2 | m |
| 6 | mississippi$ | 0 | 1 | $ |
| 7 | pi$ | 0 | 10 | p |
| 8 | ppi$ | 1 | 9 | i |
| 9 | sippi$ | 0 | 7 | s |
| 10 | sissippi$ | 2 | 4 | s |
| 11 | ssippi$ | 1 | 6 | i |
| 12 | ssissippi$ | 3 | 3 | i |

This time the backward search step is successful, and we go on with the algorithm.

# Wheeler graphs

- Wheeler DFAs have a total order on the set of states.
- The set $T(\pi)$ of all states *reached* by a pattern $\pi$ is always an interval.
- Intuively, the ordering plays the role of the suffix array.

$$T(e) = [12, 14]$$
$$T(ca) = [3, 4]$$
$$T(dba) = [2, 3].$$

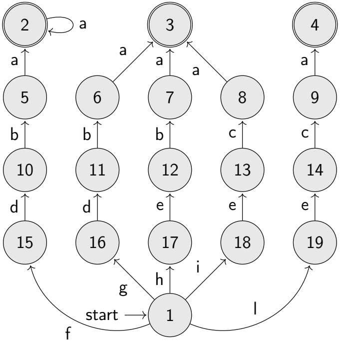


# Matching statistics of a string $\pi$ w.r.t a Wheeler DFA $\mathcal{A}$

$\mathcal{A} =$ the Wheeler DFA in the figure

$\pi = caa$

1. $\ell_1 = 1$, $[l_1, r_1] = [8, 9]$.
2. $\ell_2 = 2$, $[l_1, r_1] = [3, 4]$.
3. $\ell_3 = 2$, $[l_1, r_1] = [2, 2]$.

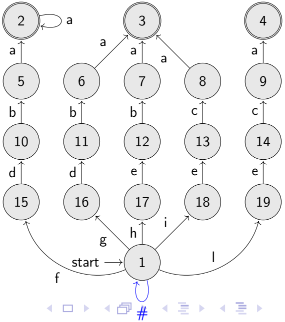



The definition is symmetrical for historical reasons.

# LCP array of a Wheeler DFA

- Since in the algorithm on strings we had an LCP array for $T = mississippi\$$, now we need to somehow define an LCP array for the Wheeler DFA $\mathcal{A}$.
- Even though there may be infinitely many strings reaching a state, we show that it suffices to consider the *minimum* and *maximum* such strings.

| State | $i$ | Minima and maxima | $LCP[i]$ |
|---|---|---|---|
| 1 | 1 | ######... | |
| | 2 | ######... | $\infty$ |
| 2 | 3 | aaaaaa... | 0 |
| | 4 | abdf##... | 1 |
| 3 | 5 | abdg##... | 3 |
| | 6 | acei##... | 1 |
| 4 | 7 | acel##... | 3 |
| | 8 | acel##... | $\infty$ |
| ... | ... | ... | ... |



# It is not that easy...

- Analogously to the algorithm on strings, given a string $\pi$, we will sometimes need to determine the longest suffix $\pi'$ of $\pi$ reaching more states than $\pi$.
- If $T(\pi) = [r, s]$, we can determine $\pi'$ by the following formula:

$$|\pi'| = \max\left\{ \min\{\mathsf{lcp}(\max_{r-1}, \min_r), \mathsf{lcp}(\min_r, \pi^R)\}, \min\{\mathsf{lcp}(\pi^R, \max_s), \mathsf{lcp}(\max_s, \min_{s+1})\} \right\}.$$

- Orange values are stored in the LCP array (just like in the algorithm on strings).
- Blue values were not needed in the algorithm on strings, but they are now needed because some strings reaching $T(\pi)$ may not have $\pi$ as a suffix. We maintain the blue values, and we show that they can be efficiently updated during the algorithm.

# Computing Matching Statistics on Wheeler DFAs

Alessio Conte[1] Nicola Cotumaccio[2,3] Travis Gagie[3]
Giovanni Manzini[1] Nicola Prezza[4] Marinella Sciortino[5]

[1]University of Pisa, Italy

[2]GSSI, L'Aquila, Italy

[3]Dalhousie University, Halifax, Canada

[4]Ca' Foscari University, Venice, Italy

[5]University of Palermo, Italy