# Weekplan: Suffix Trees I

Philip Bille and Inge Li Gørtz

## References and Reading

[1] Tries and Suffix Trees. Inge Li Gørtz.

[2] Algorithms on Strings, Trees, and Sequences, Chap. 5-9, D. Gusfield

We recommend reading [1] in detail. [2] provides an extensive list of applications of suffix trees.

## Exercises

**1** [$w$] **Suffix Trees** Draw the suffix tree $T$ for the string `mississippi$`. Write edge labels (substrings) and leaf labels (suffix number). Illustrate how a search for "issi"works.

**2** [$w$] **Substring Counting** Let $S = s_0 s_1 \cdots s_{n-1}$ be a string of length $n$ over an alphabet $\Sigma$. We are interested in a data structure for $S$ that supports the following query.

- count($P$): return the number of occurrences of $P$ in $S$.

Give a data structure that supports count($P$) queries efficiently.

**3** **Number of nodes in a compact trie** Let $T$ be a tree where every internal node has a least 2 children. Let $\ell$ be the number of leaves in $T$ and let $i$ be the number of internal nodes. Use induction to prove that $i \leq \ell - 1$. Give an example showing that this is a tight bound.

**4** **Repeats** Solve the following exercises. Assume you have an efficient black-box algorithm for computing the suffix tree of a string.

**4.1** A *repeat* in a string $S$ is a substring $R$ that occurs at least twice in $S$. Show how to efficiently compute the length of a longest substring of $S$ that is a repeat.

**4.2** Given a string $S$ of length $n$ and an integer $k$, show how to efficiently find the smallest substring of $S$ occurring *exactly* $k$ times. Analyze the time and space consumption of your algorithm.

**5** **Longest Common Extensions** Let $S$ be a string of length $n$ over alphabet $\Sigma$. The *longest common extension* problem is to preprocess S into data structure to support queries of the following form:

- LCE(i,j): Return the length of the longest common prefix of $S[i, n]$ and $S[j, n]$.

**6** **Restricted Suffix Search** Let $S$ be a string of length $n$ over alphabet $\Sigma$. Give an efficient data structure for $S$ that supports the following query:

- rsearch($P, i, j$): report the starting positions of occurrences of string $P$ in $S[i, j]$.

**7 DNA contamination [2]** Various laboratory processes used to isolate, purity, clone, copy, maintain, probe, or sequence a DNA string can course unwanted DNA to become inserted into the string of interest or mixed together with a collection of strings. Often, the DNA sequences from many of the possible contaminants are known. This motivates the following computational problem:

Given a string $S_1$ (the newly isolated and sequenced string of DNA) and a string $S_2$ (the combined sources of possible contamination), find all substrings of $S_2$ that occur in $S_1$ and that are longer than some given length $\ell$. These substrings are candidates for unwanted pieces of $S_2$ that have contaminated the desired DNA string. Give an efficient algorithm to solve the problem.

**8 Reversible substrings** Let $S = s_1 s_2 \cdots s_n$ be a string of length $n$ over a constant size alphabet $\Sigma$. A *reversible substring* of $S$ is a substring of odd length that reads the same from left-to-right and right-to-left. Give an efficient algorithm that computes the length of the longest reversible substring of $S$.

**9 Lexicographically smallest shift** In chemical databases for circular molecules, each molecule is represented by a circular string of chemical characters. To allow faster lookup and comparisons of molecules, one wants to store each circular string by a canonical linear string. A natural choice for a canonical linear strings the one that is lexicographically smallest. That gives the following computational problem.

Assume we are given a string $T = x_1 \ldots x_n$ of length $n$. A *shift* of $T$ by $s$, $0 \leq s < n$, is the string $T^s = x_{s+1} x_{s+2} \ldots x_n x_1 x_2 \ldots x_s$. In this problem we want to find the *lexicographically smallest shift*, i.e. the shift $s$ where $T^s$ is lexicographically smallest among $T^0, \ldots, T^{n-1}$. Eg. $T^2 = T^7 = \texttt{aababaabab}$ are the lexicographically smallest shifts of the string

$$T = \texttt{abaababaab}$$

**9.1** State all $s$ where $T^s$ is a lexicographically smallest shift of the string

$$T = \texttt{bcabaabcabaabcabaa}$$

**9.2** Describe an algorithm that given a string $T$ of length $n$ over an alphabet of size $O(1)$ computes all $s$ where $T^s$ is a lexicographically smallest shift of $T$. State the algorithms running time.