Suffix Tree

(Adapted from *An Introduction to Bioinformatics Algorithms* by *Neil C. Jones* and *Pavel A. Pevzner*)

A suffix tree is a data structure that allows preprocessing of text in such a way that for any pattern of length *n*, we can find out whether or not it occurs in the text using only O(*n*) time.

Inputs: A piece of text **X** of length *m* and a pattern **Y** of length *n*
For example: **X** = ATCTAATG, **Y** = AT
Output: All positions within **X** where **Y** occurs.
Positions i = 1, 6 (x_1x_2 = AT, x_6x_7 = AT)

If **X** is of length *m*, it will have *m* suffixes varying in length from *1* to *m*.
Suffixes of **X** here are: G, TG, ATG, AATG, TAATG, CTAATG, TCTAATG, ATCTAATG
Total length of all the suffixes = 1 + 2 + 3 + ... + *m* = *m*(*m*+1)/2 = O(m^2)
It is possible to easily build a suffix tree in O(m^2).[1]

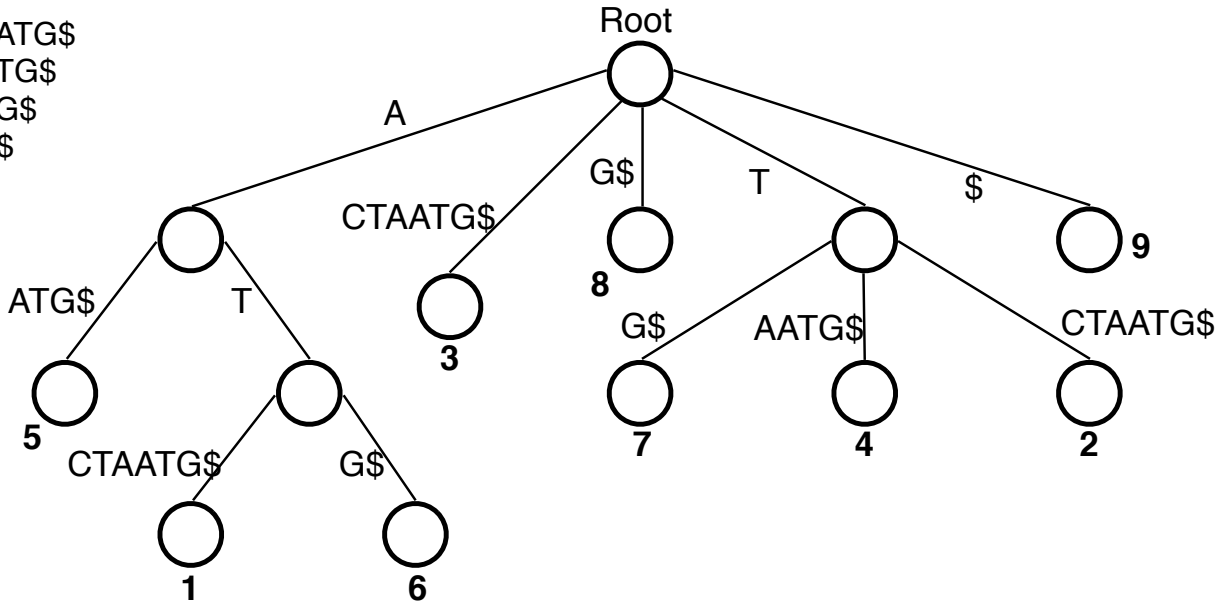
The suffix tree for text **X** = $x_1....x_m$ is a rooted labeled tree with m leaves (numbered from 1 to *m*, one for each of the m suffixes) satisfying the following conditions:

- Each edge is labeled with a substring of the text **X**
- Each internal vertex has at least 2 children
- Any two edges out of the same vertex start with a different letter
- Every suffix of the text **X** is spelled out on a path from the root to some leaf

One requirement of the suffix tree is that no suffix string is the prefix of another suffix string. Since it is unlikely that the last letter of the text will not occur anywhere else in the text, we add a unique special character (like $) to the end of the text to ensure this requirement is met.

X = ATCTAATG$
The suffixes (numbered by their position in the string **X**) and the suffix tree are shown below.

1) ATCTAATG$
2) TCTAATG$
3) CTAATG$
4) TAATG$
5) AATG$
6) ATG$
7) TG$
8) G$
9) $



[1] It is also possible to create the suffix tree in O(*m*). Check Weiner's algorithm for a slightly more complicated algorithm that achieves this.

To search for the pattern **Y** in the suffix tree for text **X**, we need to ***thread*** the pattern in the tree. Threading involves matching the characters from **Y** on a unique path from the root of the suffix tree.

- Complete threading: When all the characters from **Y** can be matched along a path
- Incomplete threading: When it is not possible to match all the characters of **Y** along any path in the tree.

When threading is complete, it will end at some vertex of the tree. We define the **Y**-matching leaves to be all the descendant leaves of the vertex where the threading ends. The positions that are stored in the **Y**-matching leaves are thus the positions in the text **X** where **Y** occurs.

When threading is incomplete we can say that the pattern **Y** does not occur in **X**.

In the following diagram, the black vertices show the path of threading pattern AT.
The grey vertices are thus the **Y**-matching leaves, showing AT occurs at positions 1 and 6 in **X**.