Lecture 19: Suffix Tries/Trees

BT 3051 – Data Structures and Algorithms for Biology

Karthik Raman
Department of Biotechnology
Bhupat and Jyoti Mehta School of Biosciences
Indian Institute of Technology Madras

Exact Pattern Matching: Multiple Patterns

- If we are going to search the same text over and over ...
- ...*why not pre-process the text?!*
- Initial cost of pre-processing the text is compensated by a speedup in each subsequent pattern queried

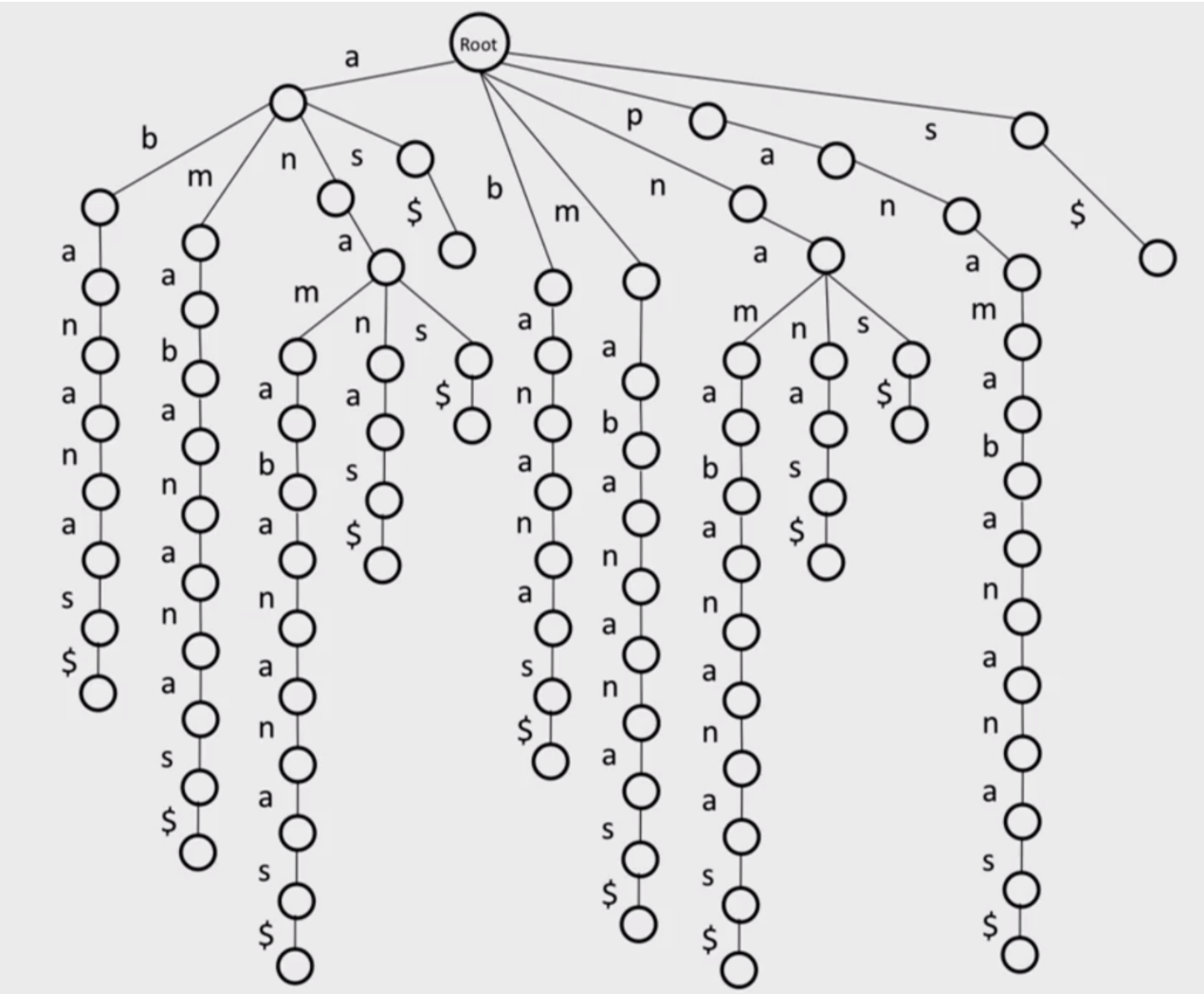
Can we create a new data structure for the genome itself?

Pre-processing the Genome

- Form *all suffixes* of the *genome text*
- How do we combine these suffixes?
- Let's use a trie ...

An Example Suffix Trie

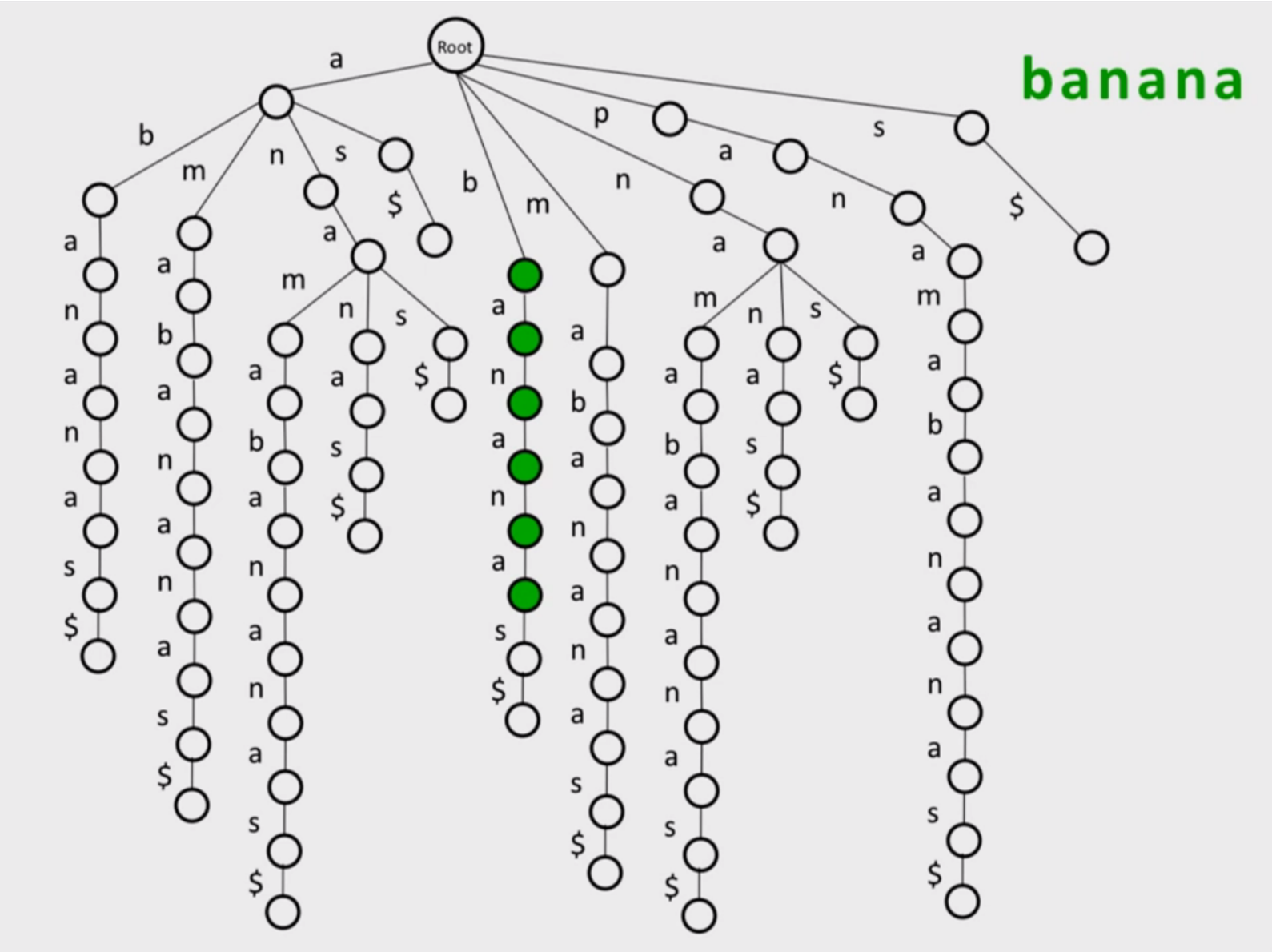
Genome: panamabananas$



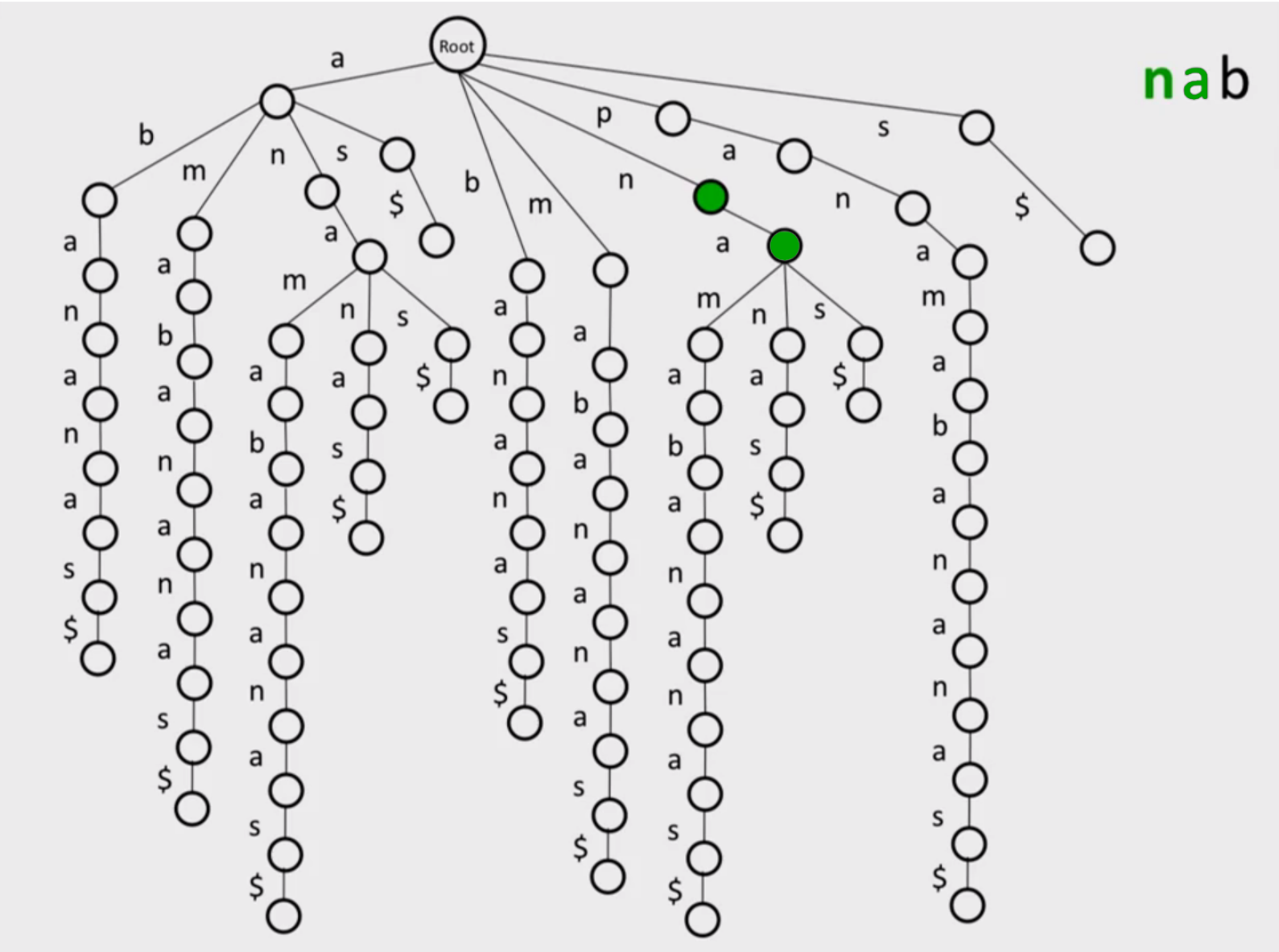
How to use a Suffix Trie for matching a *Pattern*?

- For every *Pattern*, check if it can be spelt out from the root in the trie

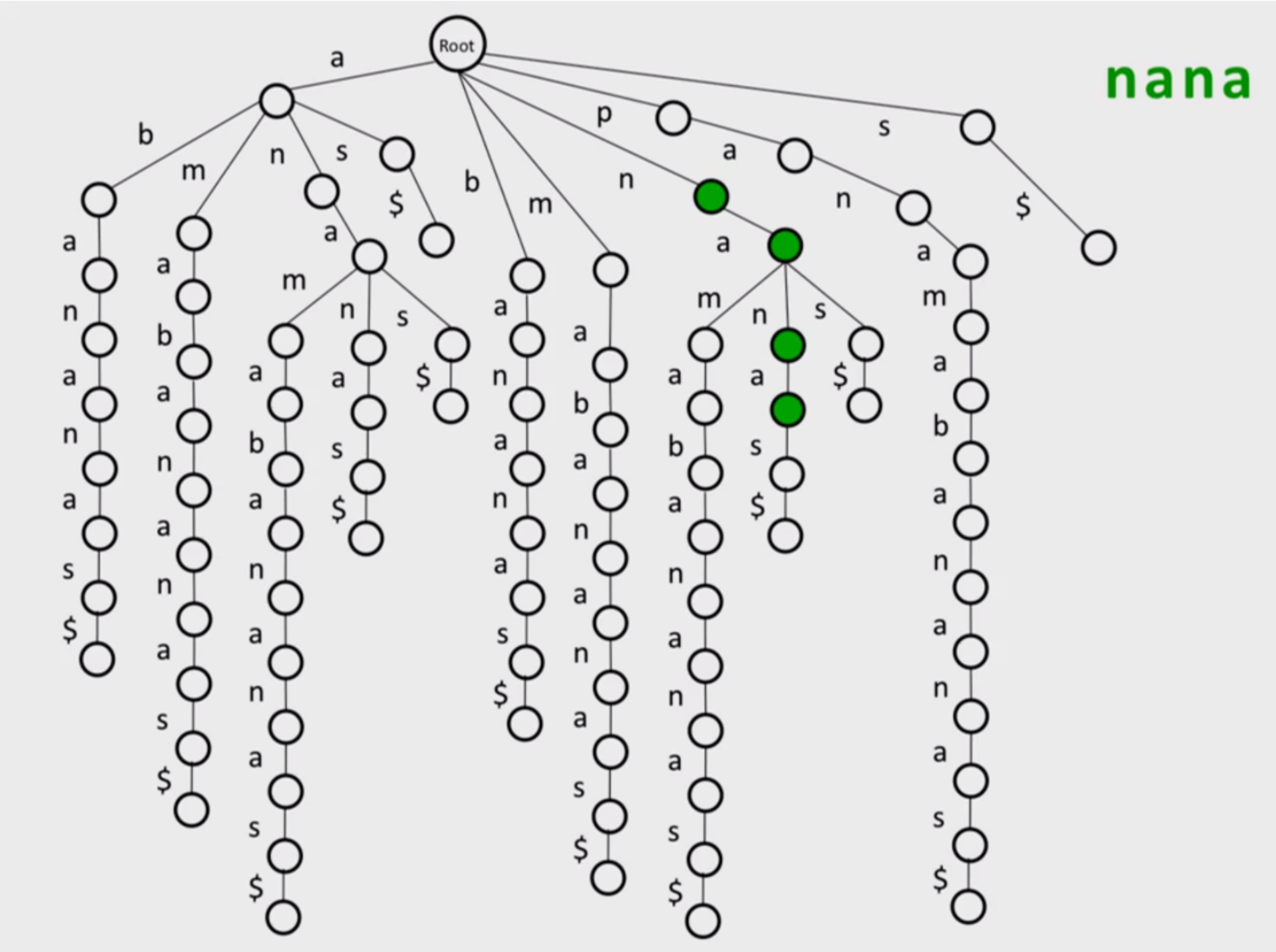
How to use a Suffix Trie for matching a *Pattern*?



How to use a Suffix Trie for matching a *Pattern*?



How to use a Suffix Trie for matching a *Pattern*?



Where are the matches in the *Text*?

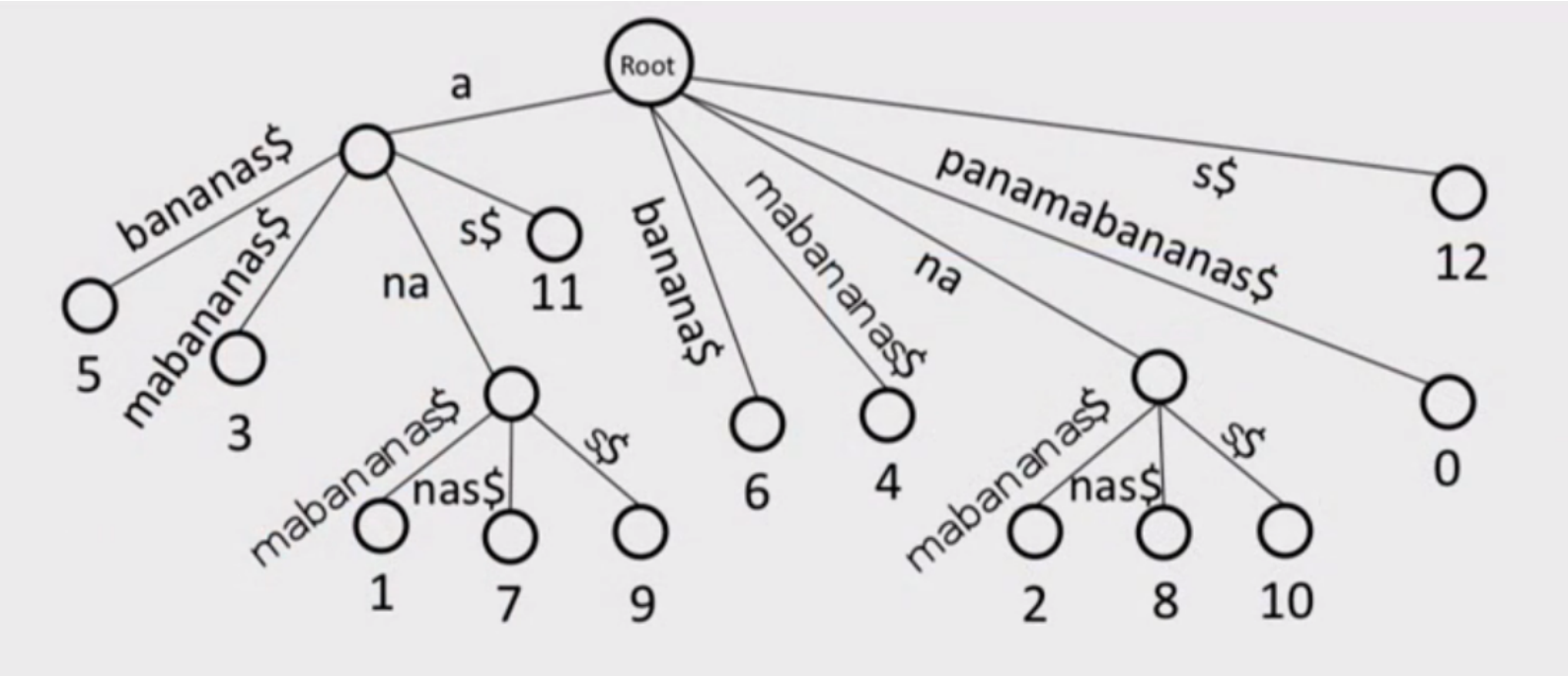
Suffix Tries: Memory Issues

- For a Genome of length n, how many characters do we need to store?
- $n(n+1)/2$, or $\Theta(n^2)$!
- Infeasible!

How do we salvage this (seemingly) nice data structure?

Enter Suffix 'Trees'

- Trim/compress every non-branching path into an edge!



Number of leaves = |Genome|

Suffix Tree Complexity

- Smart algorithms exist!
- No need to build trie and then compress
- Ukkonen's algorithm is *linear*, and also on-line!
- Memory, Runtime: both $\Theta(n)$!
- Query time: $O(n + \sum_i m_i)$

Must watch video: https://www.youtube.com/watch?v=LB-ANFydv30