

# **Suffix Trees**

Pavel Pevzner

Department of Computer Science and Engineering  
University of California at San Diego

**Algorithms on Strings  
Data Structures and Algorithms**

This slide desk is incomplete.  
For the complete set of frames,  
please see our videos in the  
[Algorithms on Strings](#) course on [Coursera](#)  
[\(Algorithms and Data Structures Specialization\)](#)

# Outline

- **From Genome Sequencing to Pattern Matching**
- Brute Force Approach to Pattern Matching
- Herding Patterns into Trie
- Herding Text into Suffix Trie
- From Suffix Tries to Suffix Trees

# The Newspaper Problem

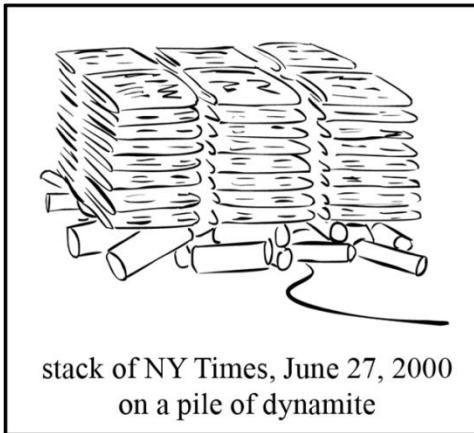


stack of NY Times, June 27, 2000

# The Newspaper Problem



stack of NY Times, June 27, 2000



stack of NY Times, June 27, 2000  
on a pile of dynamite

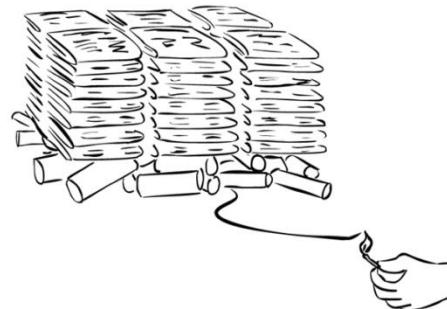
# The Newspaper Problem



stack of NY Times, June 27, 2000

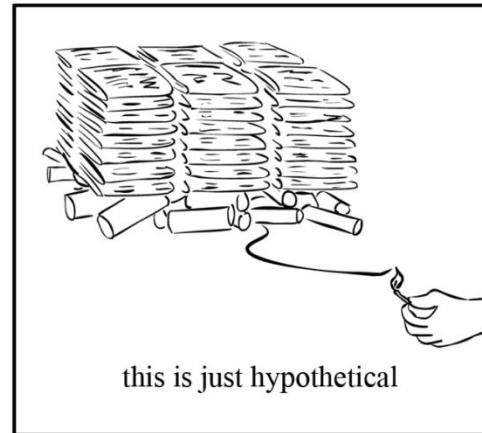
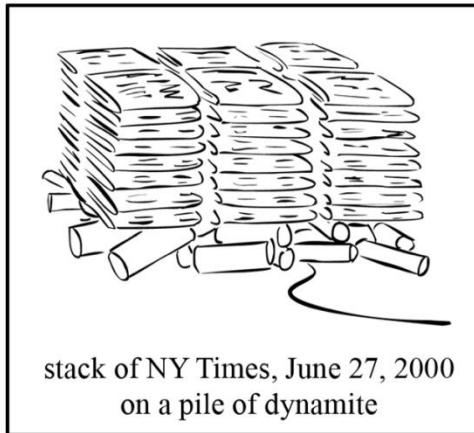


stack of NY Times, June 27, 2000  
on a pile of dynamite

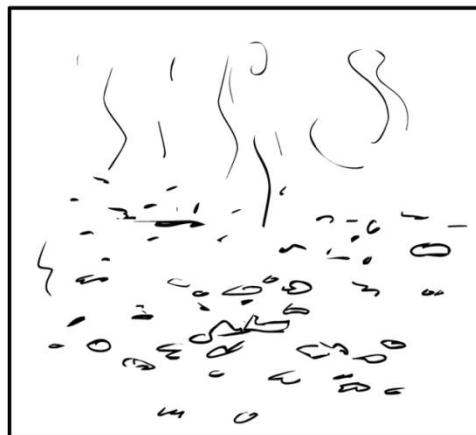
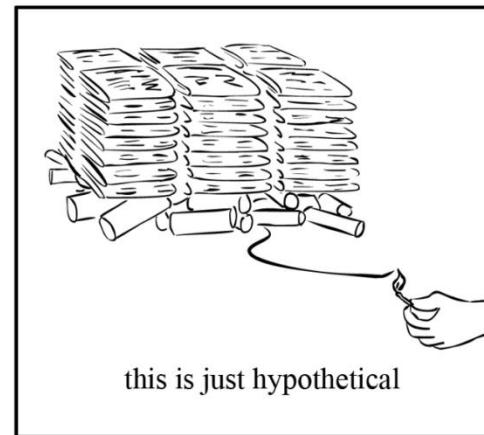
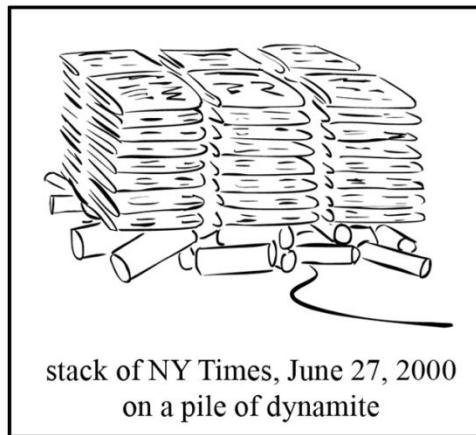
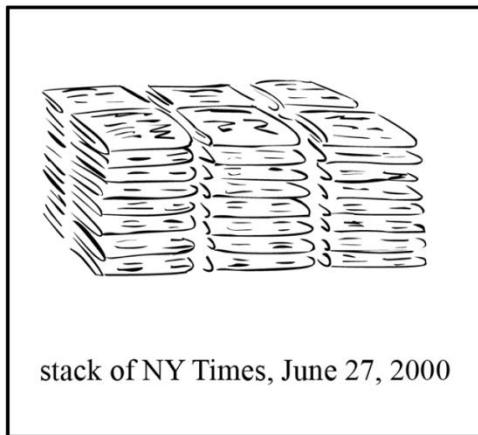


this is just hypothetical

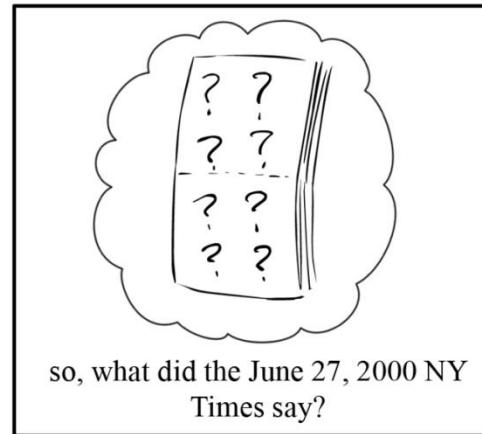
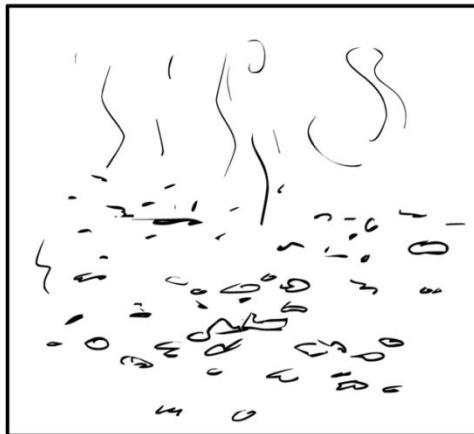
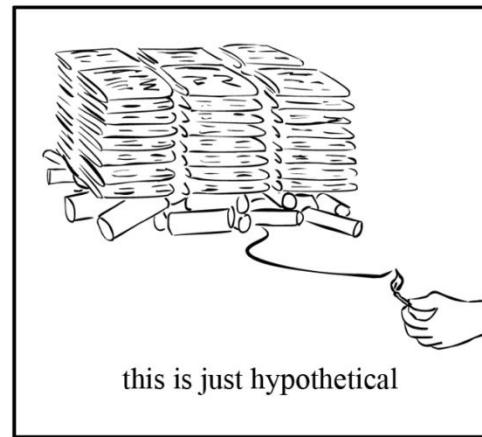
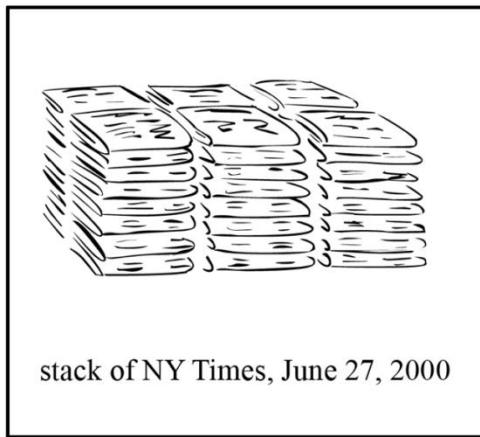
# The Newspaper Problem



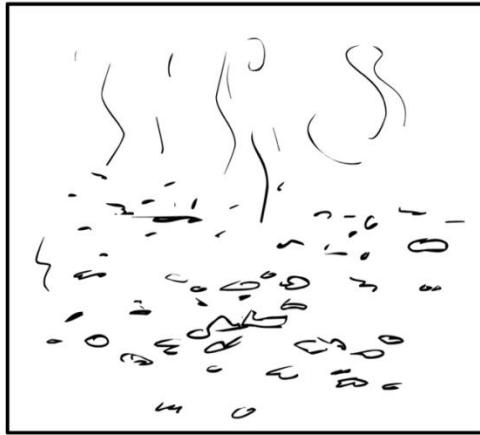
# The Newspaper Problem



# The Newspaper Problem



# The Newspaper Problem as an Overlapping Puzzle

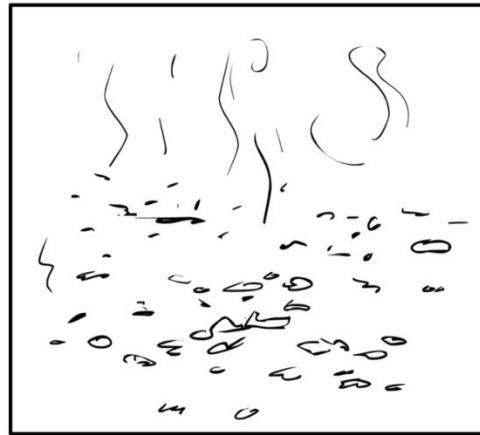


hoodie, app...  
we have not yet named  
information is welc...

lie, app...  
yet named any suspects, alt...  
is welc...

o'2'  
ce ca...

# The Newspaper Problem as an Overlapping Puzzle



hoodie, appre-  
we have not yet named  
information is welc

die, appre-  
yet named any suspects, alt  
is welc

# Millions of Copies of a Genome



stack of NY Times, June 27, 2000

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC  
CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC  
CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC  
CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC

# Breaking the Genomes at Random Positions



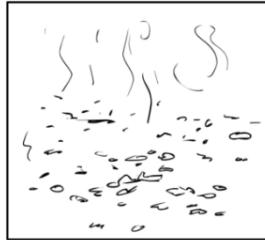
CTGATGATGGACTACGC ACTACTGCT GCTGTATTAGTCAGCTACC ATCGTAGCTA GATGCATTAGC AGCTATCGTCAGCTACCATCGTAGC  
CTGA GATGGACTCGCTACTACTAGCTGTATACGATCAGC ACCACATCGTAGCTACGATGCA TAGCAAGCTCGGATCAC TACCACATGTAGC  
CTGATGATGGACTACGC ACTACTGCTA CTGTATTACATCGCTAACATCGTAGCAGATGCATTAGCAAGCTATGGATCAGCTACACATCGTAGC  
CTGATGATGCTACGCTACACTGCTAGCTATTAGCATGCTACACAGATGCAAGCTATGGATCAGCTACACATCGTAGC

# Generating Short Substring (Reads)



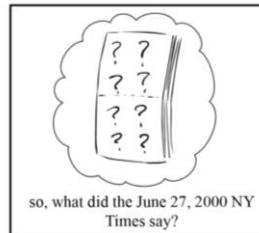
CTGATGA TGGACTACGCTAC TACTGCTAG CTGTATTACG ATCAGCTACCACA TCGTAGCTACG ATGCATTAGCAA GCTATCGGA TCAGCTACCA CATCGTAGC  
CTGATGATG GACTACGCT ACTACTGCTA GCTGTATTACG ATCAGCTACC ACATCGTAGCT ACGATGCATTA GCAAGCTATC GGATCAGCTAC CACATCGTAGC  
CTGATGATGG ACTACGCTAC TACTGCTAGCT GTATTACGATC AGCTACCAC ATCGTAGCTACG ATGCATTAGCA AGCTATCGG A TCAGCTACCA CATCGTAGC  
CTGATGATGGACT ACGCTACTACT GCTAGCTGTAT TACGATCAGC TACCACATCGT AGCTACGATGCA TTAGCAAGCT ATCGGATCA GCTACCACATC GTAGC

# Burning Some Reads

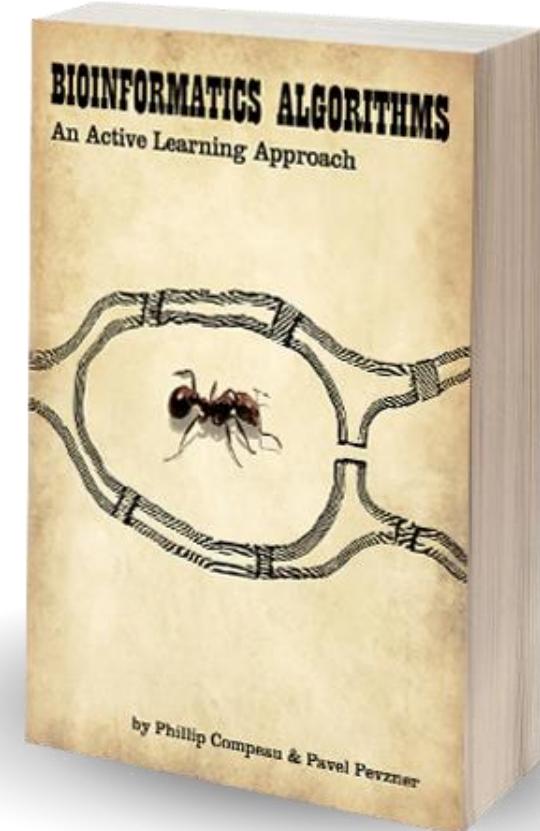


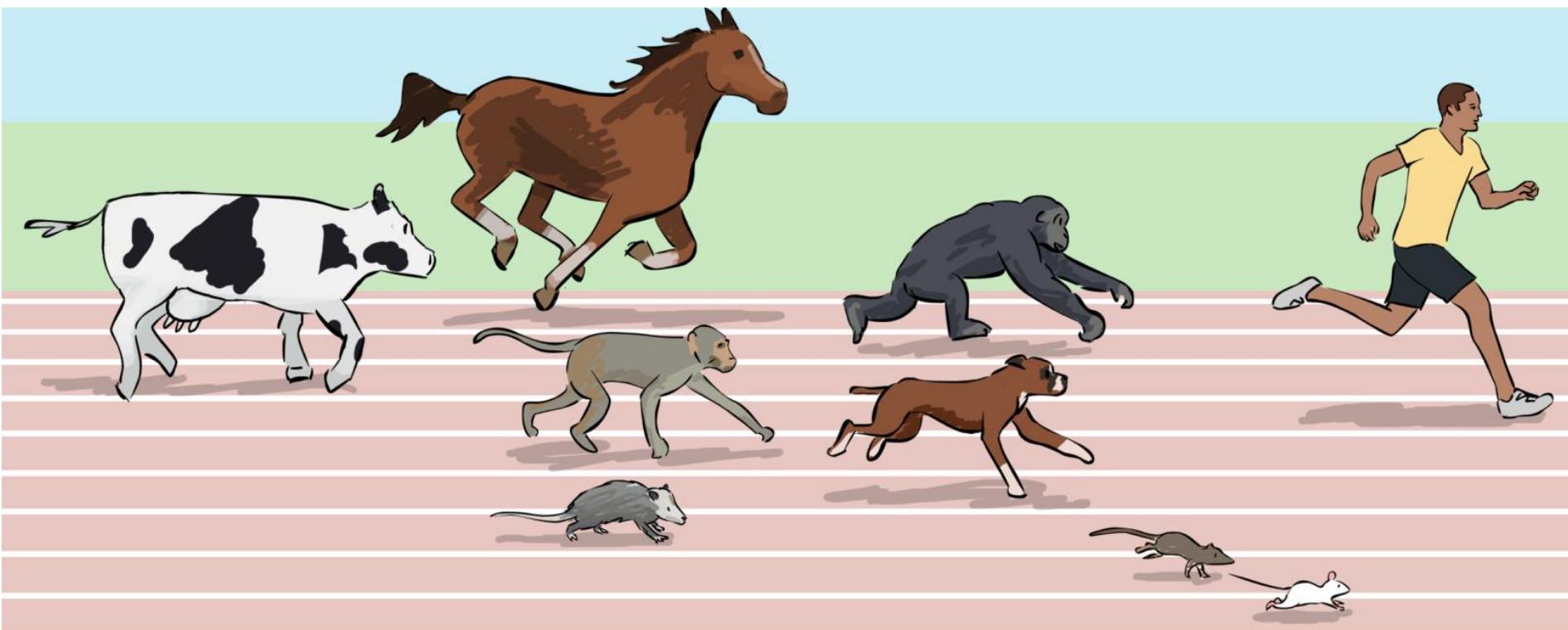
CTGATGA TGGACTACGCTAC TACTGCTAG CTGTATTACG ATCAGCTACCACA TCGTAGCTACG ATGCATTAGCAA GCTATCGGA TCAGCTACCA CATCGTAGC  
CTGATGATG GACTACGCT ACTACTGCTA GCTGTATTACG ATCAGCTACC ACATCGTAGCT ACGATGCTTA GCAAGCTATC GGATCAGCTAC CACATCGTAGC  
CTGATGATGG ACTACGCTAC TACTGCTAGCT GTATTACGATC AGCTACCAC ATCGTAGCTACG ATGCATTAGCA AGCTATCGG A TCAGCTACCA CATCGTAGC  
CTGATGATGGACT ACGCTACTACT GCTAGCTGTAT TACGATCAGC TACCACATCGT AGCTACGATGCA TTAGCAAGCT ATCGGATCA GCTACCACATC GTAGC

# Assembling Genome



CTGATGA TGGACTACGCTAC TACTGCTAG CTGTATTACG ATCAGCTACCACA TCGTAGCTACG ATGCATTAGCAA GCTATCGGA TCAGCTACCA CATCGTAGC  
CTGATGATG GACTACGCT ACTACTGCTA GCTGTATTACG ATCAGCTACC ACATCGTAGCT ACGATGCTTAA GCAAGCTATC GGATCAGCTAC CACATCGTAGC  
CTGATGATGG ACTACGCTAC TACTGCTAGCT GTATTACGATC AGCTACCAC ATCGTAGCTACG ATGCATTAGCA AGCTATCGG A TCAGCTACCA CATCGTAGC  
CTGATGATGGACT ACGCTACTACT GCTAGCTGTAT TACGATCAGC TACCACATCGT AGCTACGATGCA TTAGCAAGCT ATCGGATCA GCTACCACATC GTAGC





cow  
2009

horse  
2007

opossum  
2007

macaque  
2006

dog  
2005

chimpanzee  
2005

rat  
2004

mouse  
2002

human  
2001

# Why Do We Sequence 1000s of Species?



## Applications in

- **medicine** (mouse genome)
- **agriculture** (rice genome)
- **biotechnology** (genomes of energy-producing bacteria),
- etc., etc., etc.

# Few Mutations Can Make a Big Difference...

- Different people have slightly different genomes: on average, roughly 1 mutation in 1000 nucleotides.
- The 1 in 1000 nucleotides difference accounts for height, high cholesterol susceptibility, and 7000+ genetic diseases.

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGA  
TCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTAT  
CGATCGATCGATCGATTATCTACGATCGATCGATCGATCA  
CTATACGAGCTACTACGTACGTACGATCGCGGGACTATT  
TCGACTACAGATAAAACATGCTAGTACAACAGTATACATA  
GCTGCGGGATACGATTAGCTAATAGCTGACGATATCCGAT

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGA  
TCAGCTACAACATCGTAGCTACGATGCATTAGCAAGCTAT  
CGATCGATCGATCGATTATCTACGATCGATCGATCGATCA  
CTATACGAGCTACTACGTACGTACGATCGCGTGACTATT  
TCGACTACAGATGAAACATGCTAGTACAACAGTATACATA  
GCTGCGGGATACGATTAGCTAATAGCTGACGATATCCGAT

# Emergence of Personalized Medicine

- **2010:** Nicholas Volker became the first child to be saved by genome sequencing.
  - Doctors could not diagnose his condition; he went through dozens of surgeries.
  - Sequencing revealed a mutation in a gene linked to a defect in his immune system.
  - This led doctors to use immunotherapy, which saved the child.

# From Reference Genome to Personal Genomes

- Reference human genome assembled in 2000.

CTGAGGATGGACTACGCTACTACTGATAGCTGTTT

reference  
genome

GAGGA

CCACG

TGA-AG

CTGA

GGACC

ACTAC A-AGCT

reads

GATGG

ACGCT

TGTTT



CTGAGGATGGAC**C**ACGCTACTACTGA-AGCTGTTT

personal  
genome

# Exact Pattern Matching

- Where does a read match the reference genome *exactly*?
- **Pattern Matching Problem:**
  - **Input:** A string *Pattern* (read) and a string *Text* (genome).
  - **Output:** All positions in *Text* where *Pattern* appears as a substring.

# Approximate Pattern Matching

- Where does a read match the reference genome *approximately*?
- **Approximate Pattern Matching Problem:**
  - **Input:** A string *Pattern*, a string *Text*, and an integer  $d$
  - **Output:** All positions in *Text* where *Pattern* appears as a substring **with at most  $d$  mismatches**.

# Multiple Pattern Matching

- Where do **billions** of reads match the reference genome?
- **Multiple Pattern Matching Problem:**
  - **Input:** A **set of strings** *Patterns* and a string *Text*.
  - **Output:** All positions in *Text* where a string from *Patterns* appears as a substring.

# Outline

- From Genome Sequencing to Pattern Matching
- **Brute Force Approach to Pattern Matching**
- Herding Patterns into Trie
- Herding Text into Suffix Trie
- From Suffix Tries to Suffix Trees

# A Brute Force Approach to Exact Pattern Matching

*Genome*

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTAC



# *Pattern* drives along *Text*

p a n a m a b a n a n a s

*Text*

n a n a

*Pattern*



**p** a n a m a b a n a n a s

*Text*

**n** a n a

*Pattern*



p **a** n a m a b a n a n a s

*Text*

**n** a n a

*Pattern*



p a **n** a m a b a n a n a s

*Text*

**n** a n a

*Pattern*



p a **n a** m a b a n a n a s

*Text*

**n a** n a

*Pattern*



p a **n a m** a b a n a n a s  
**n a n** a

*Text*  
*Pattern*



p a n **a** m a b a n a n a s

*Text*

**n** a n a

*Pattern*



p a n a **m** a b a n a n a s

*Text*

**n** a n a

*Pattern*



p a n a m **a** b a n a n a s

*Text*

**n** a n a

*Pattern*



p a n a m a **b** a n a n a s

*Text*

**n** a n a

*Pattern*



p a n a m a b **a** n a n a s

*Text*

**n** a n a

*Pattern*



p a n a m a b a **n** a n a s  
**n** a n a

*Text*  
*Pattern*



p a n a m a b a **n a** n a s  
**n a** n a

*Text*  
*Pattern*



p a n a m a b a **n a n** a s  
**n a n** a

*Text*  
*Pattern*



# Pattern Found!

p a n a m a b a **n a n a** s *Text*  
**n a n a** *Pattern*



p a n a m a b a n a n a s  
n a n a

*Text*  
*Pattern*



# Brute Force Approach Is Fast!

- **single Pattern:**  $O(|Text| \cdot |Pattern|)$



The runtime of the **Knuth-Morris-Pratt** algorithm:  $O(|Text|)$   
(wait for the lecture on algorithmic challenges in pattern matching)



# Brute Force Approach is Slow for **Billions** of Patterns

- **single Pattern:**  $O(|Text| \cdot |Pattern|)$
- **multiple Patterns:**

$$O\left(\sum_{\substack{\text{all strings } Pattern \\ \text{In } Patterns}} |Text| \cdot |Pattern|\right)$$

# Brute Force Approach is Slow for Billions of Patterns

- **single Pattern:**  $O(|Text| \cdot |Pattern|)$
- **multiple Patterns:**

$$O\left(\sum_{\substack{\text{all strings } Pattern \\ \text{In } Patterns}} |Text| \cdot |Pattern|\right) =$$

$$O(|Text| \cdot |Patterns|)$$

For human genome:

- $|Text| \approx 3 \cdot 10^9$
- $|Patterns| \approx 10^{12}$



# Outline

- From Genome Sequencing to Pattern Matching
- Brute Force Approach to Pattern Matching
- **Hherding Patterns into Trie**
- Herding Text into Suffix Trie
- From Suffix Tries to Suffix Trees

# Patterns Travel One at a Time

Genome

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTAC

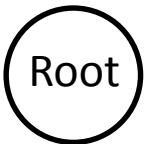


# Hherding Patterns onto a Bus

Genome

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTAC





Root

# *Patterns*

banana

pan

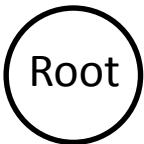
nab

antenna

bandana

ananas

nana



Root

# *Patterns*

**banana**

pan

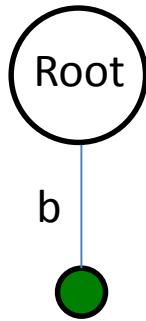
nab

antenna

bandana

ananas

nana



# *Patterns*

**banana**

pan

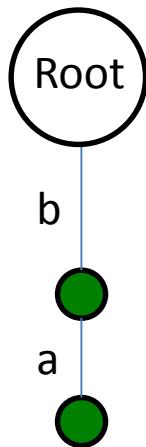
nab

antenna

bandana

ananas

nana



# *Patterns*

**banana**

pan

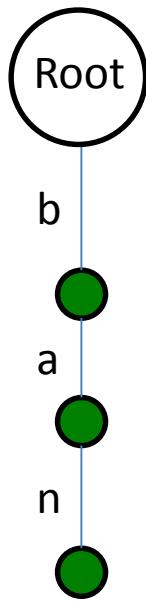
nab

antenna

bandana

ananas

nana



# ***Patterns***

**banana**

pan

nab

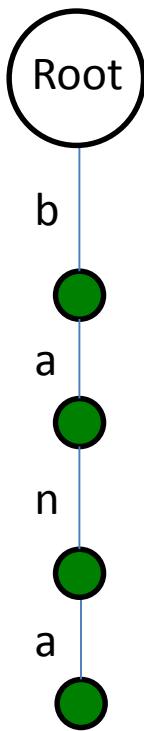
antenna

bandana

ananas

nana

# *Patterns*



**banana**

pan

nab

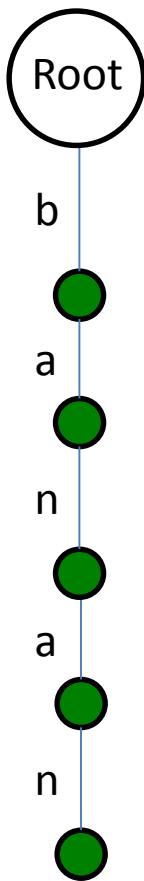
antenna

bandana

ananas

nana

# *Patterns*



**banana**

pan

nab

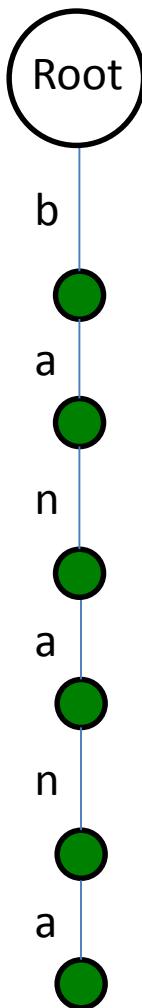
antenna

bandana

ananas

nana

# *Patterns*



**banana**

pan

nab

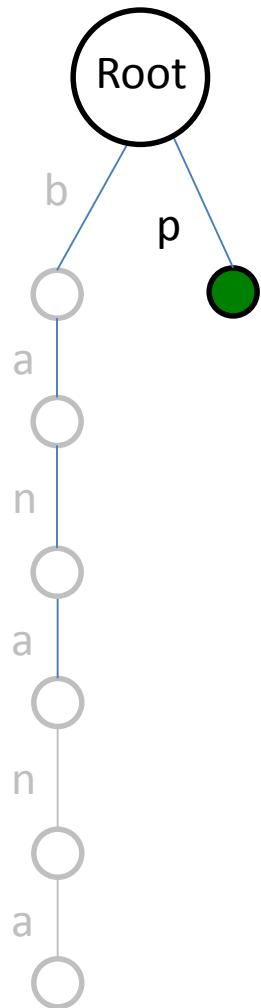
antenna

bandana

ananas

nana

# *Patterns*



banana

pan

and

nab

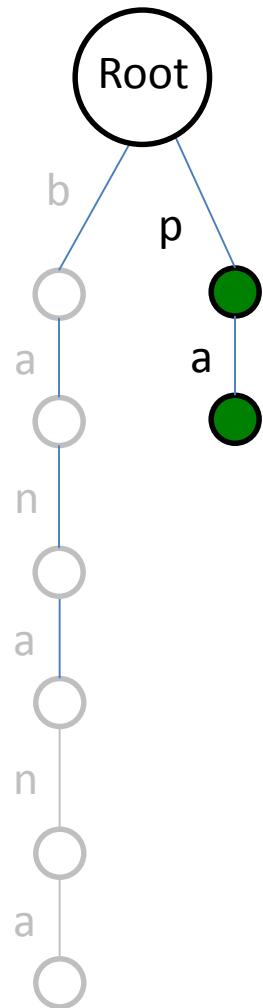
antenna

bandana

ananas

nana

# *Patterns*



# banana

**pan**

and

nab

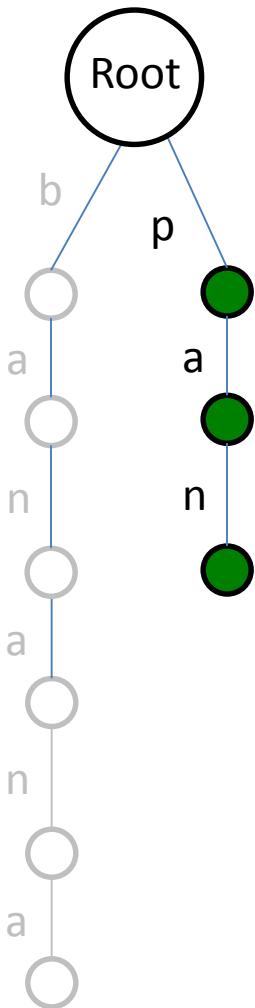
# antenna

# bandana

# ananas

nana

# *Patterns*



banana

pan

and

nab

antenna

bandana

ananas

nana

# *Patterns*

# banana

pan

and

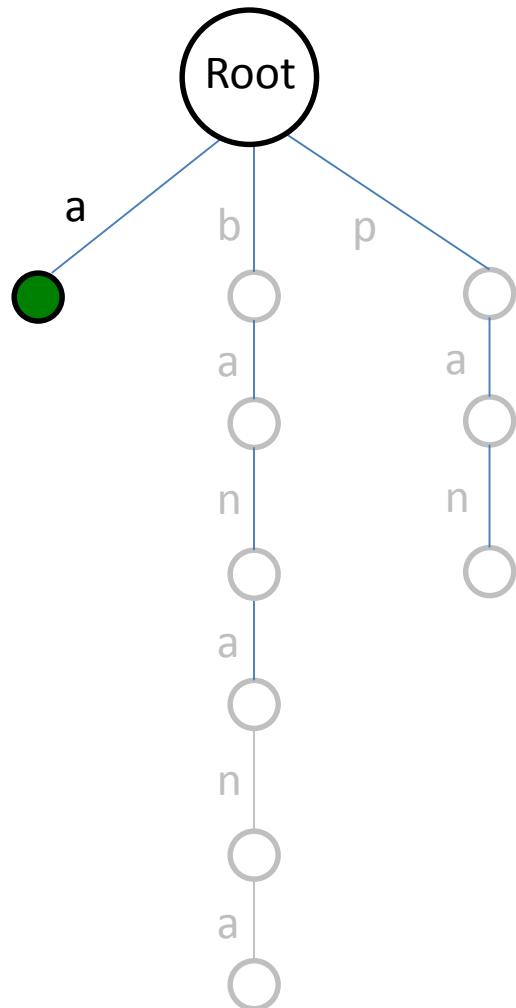
nab

## antenna

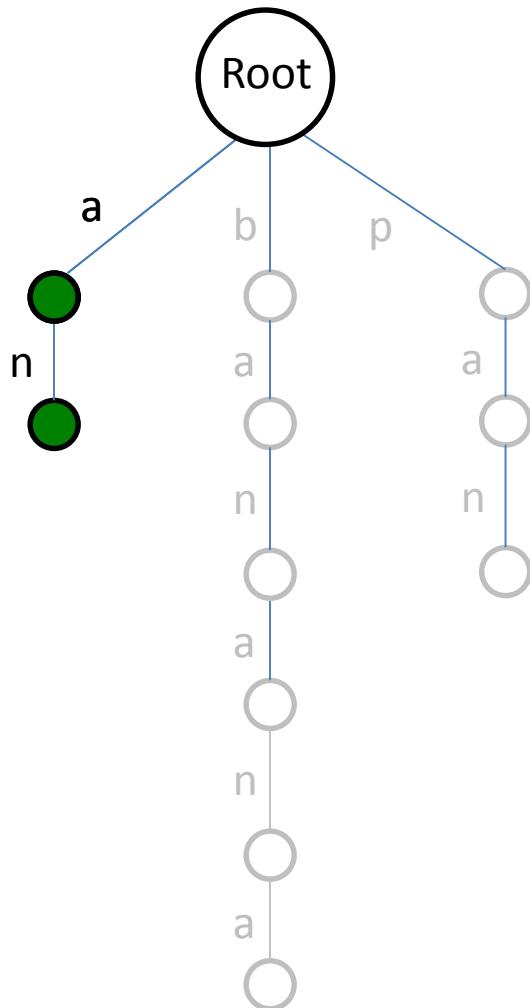
# bandana

# ananas

nana



# *Patterns*



banana

pan

**and**

nab

antenna

bandana

ananas

nana

# *Patterns*

# banana

pan

and

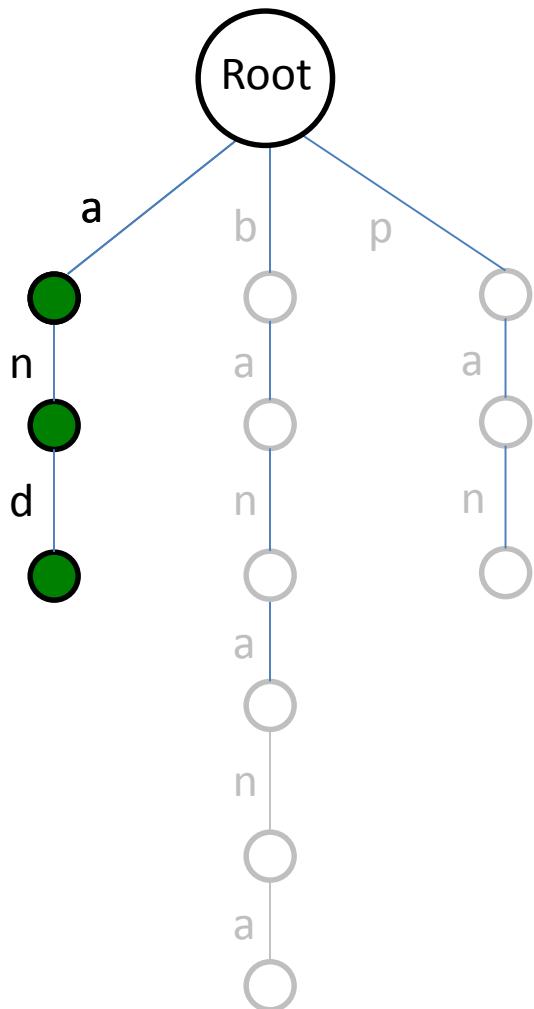
nab

# antenna

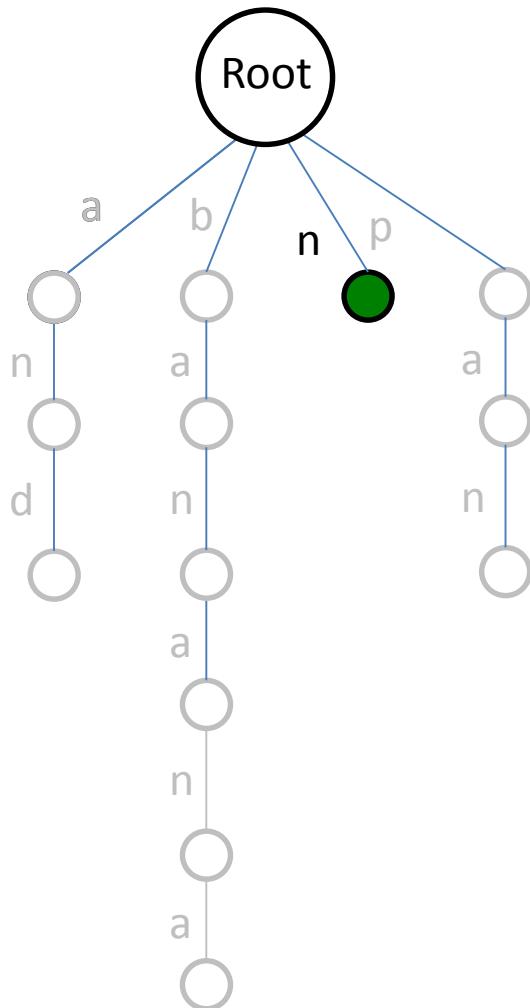
# bandana

# ananas

nana



# *Patterns*



banana

pan

and

**nab**

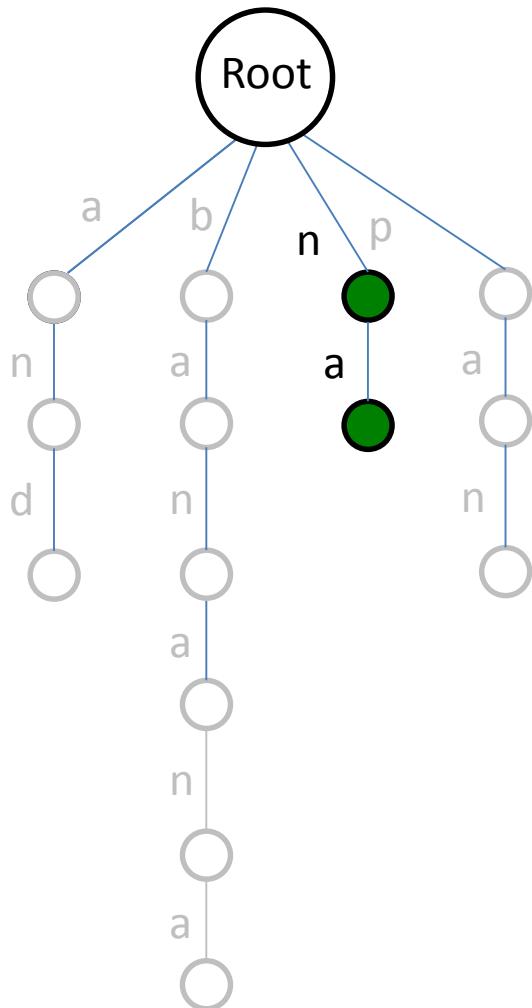
antenna

bandana

ananas

nana

# *Patterns*



banana

pan

and

**nab**

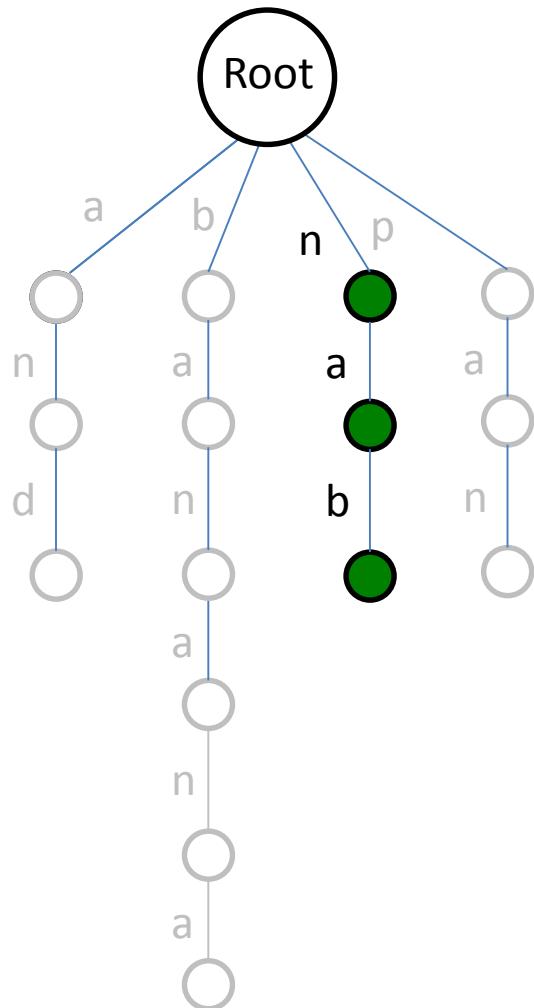
antenna

bandana

ananas

nana

# *Patterns*



banana

pan

and

**nab**

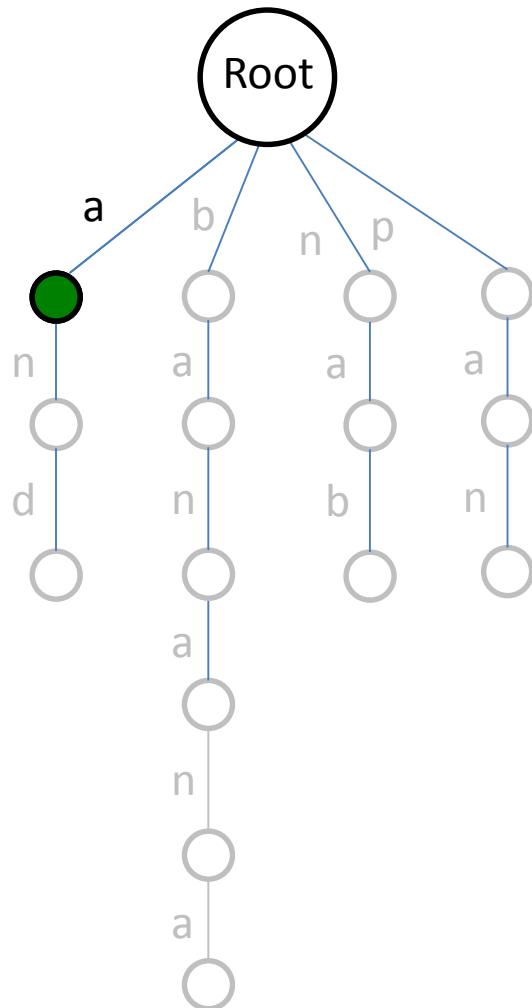
antenna

bandana

ananas

nana

# *Patterns*



banana

pan

and

nab

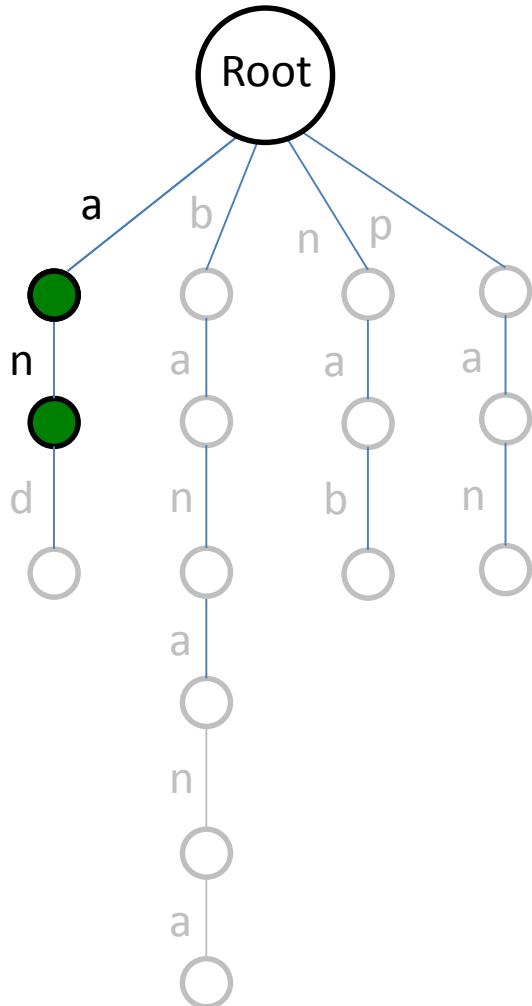
**antenna**

bandana

ananas

nana

# *Patterns*



banana

pan

and

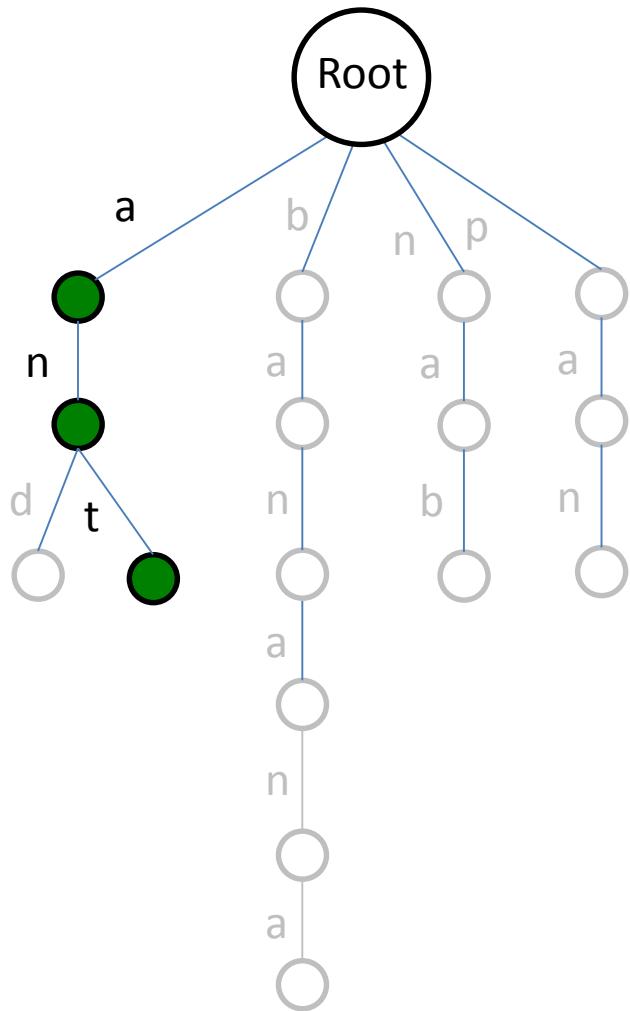
nab

**antenna**

bandana

ananas

nana



# *Patterns*

banana

pan

and

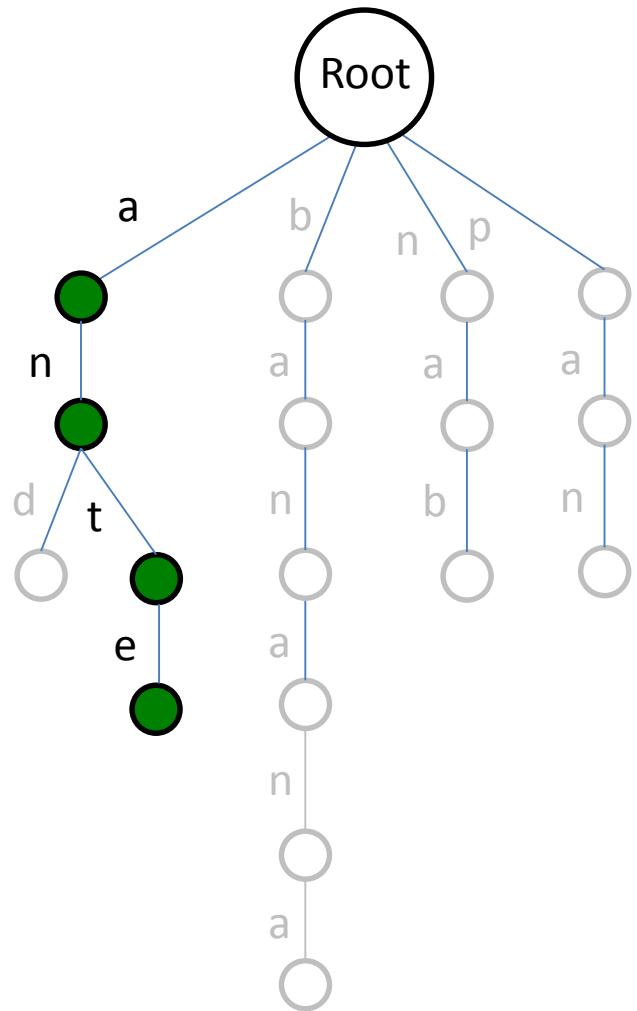
nab

**antenna**

bandana

ananas

nana



# *Patterns*

banana

pan

and

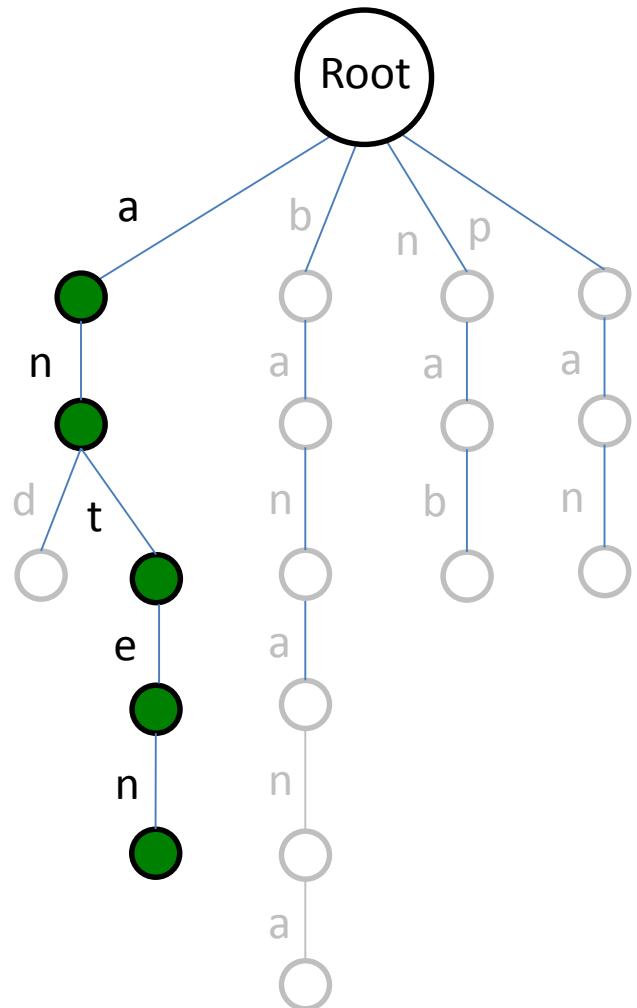
nab

**antenna**

bandana

ananas

nana



# *Patterns*

banana

pan

and

nab

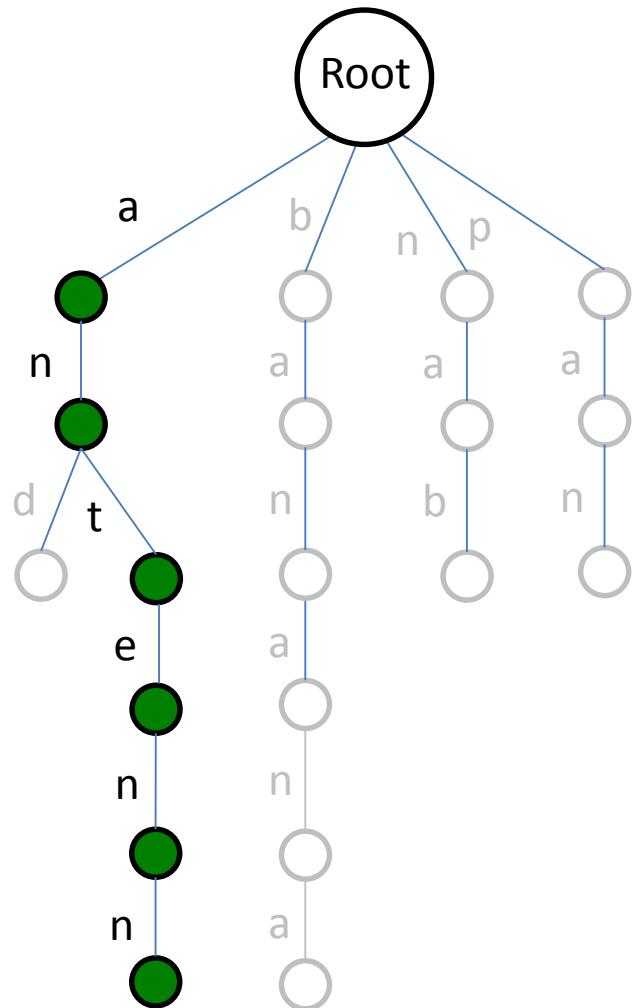
**antenna**

bandana

ananas

nana

# *Patterns*



banana

pan

and

nab

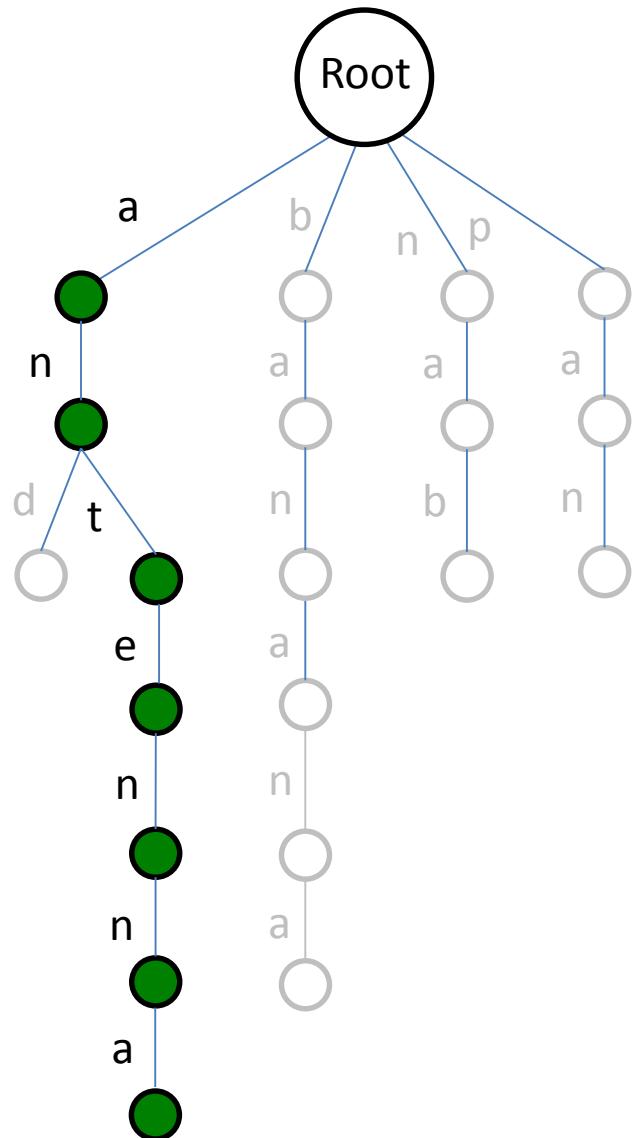
**antenna**

bandana

ananas

nana

# *Patterns*



banana

pan

and

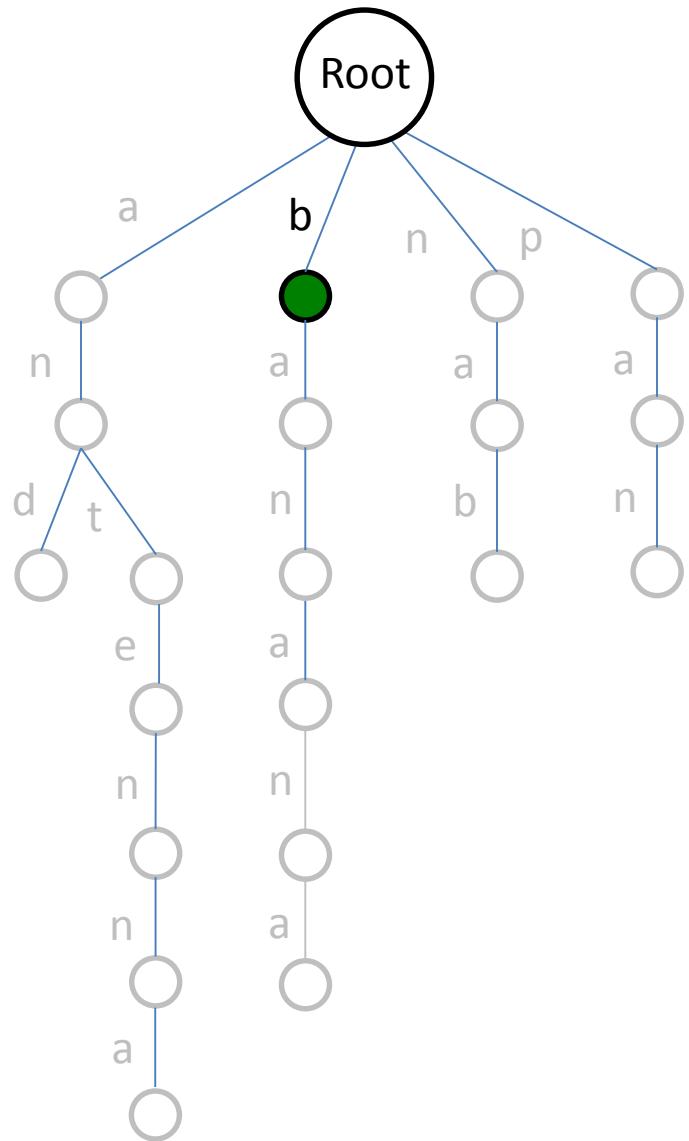
nab

**antenna**

bandana

ananas

nana



# *Patterns*

banana

pan

and

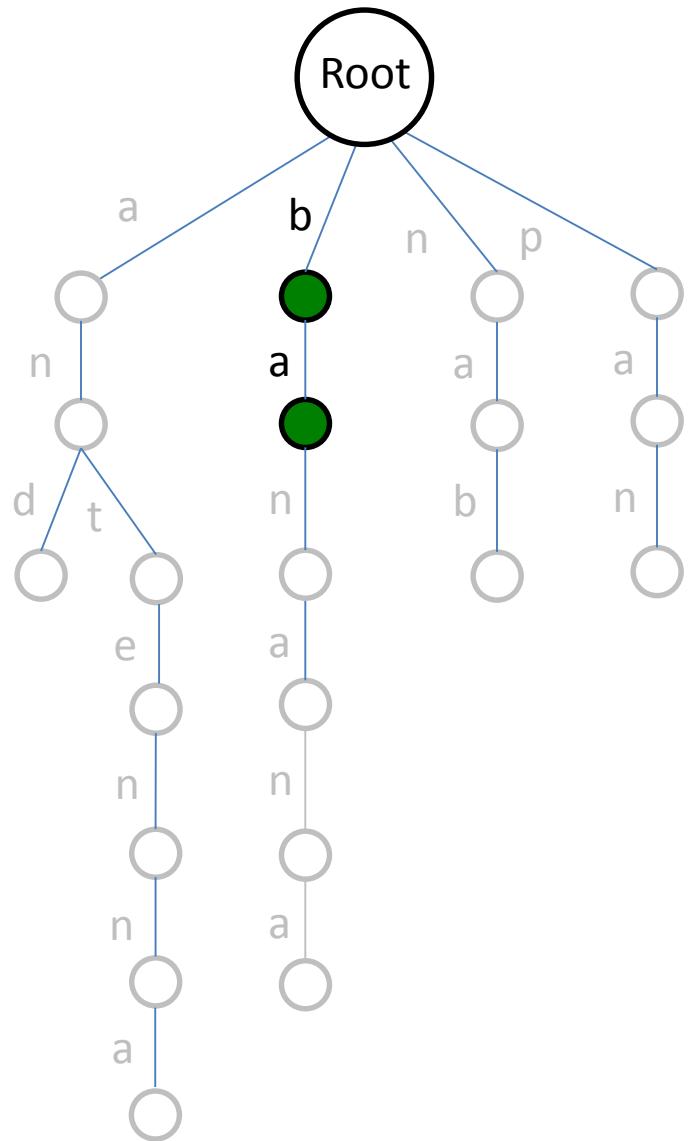
nab

antenna

**bandana**

ananas

nana



# *Patterns*

banana

pan

and

nab

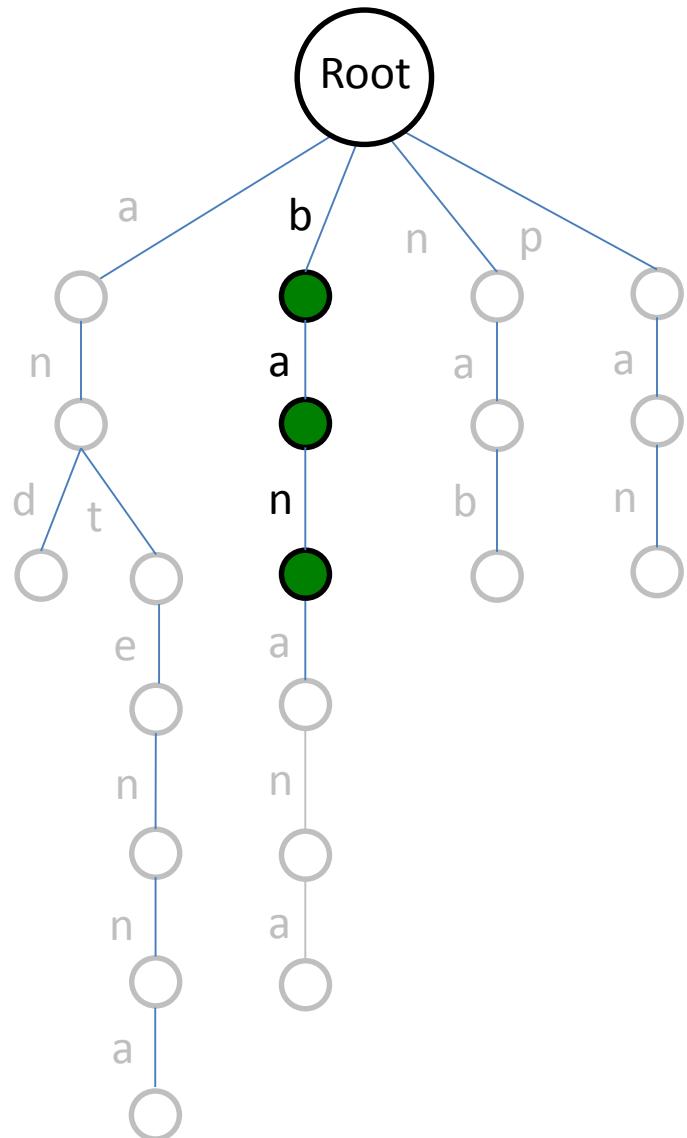
antenna

**bandana**

ananas

nana

# *Patterns*



banana

pan

and

nab

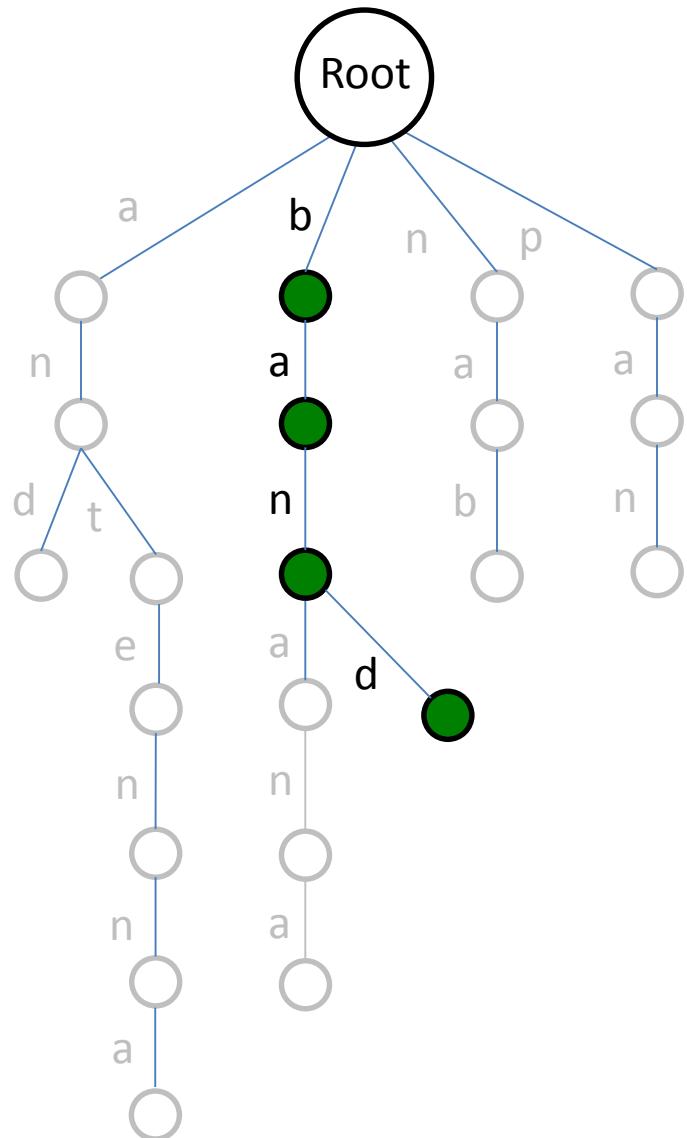
antenna

**bandana**

ananas

nana

# *Patterns*



banana

pan

and

nab

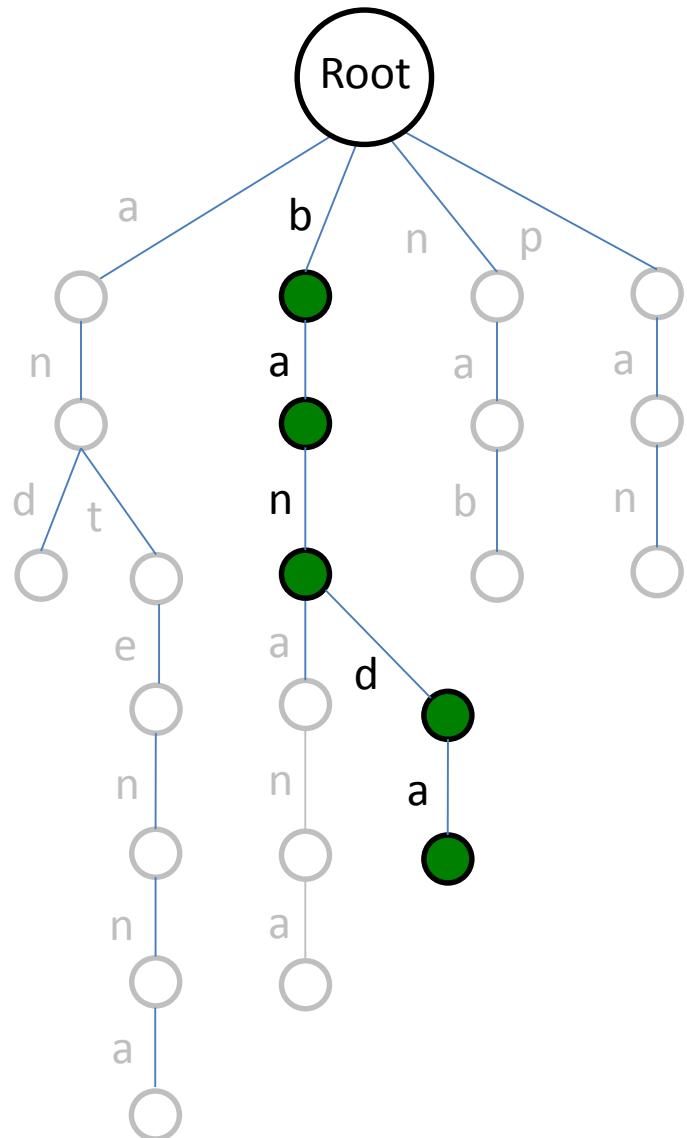
antenna

**bandana**

ananas

nana

# *Patterns*



banana

pan

and

nab

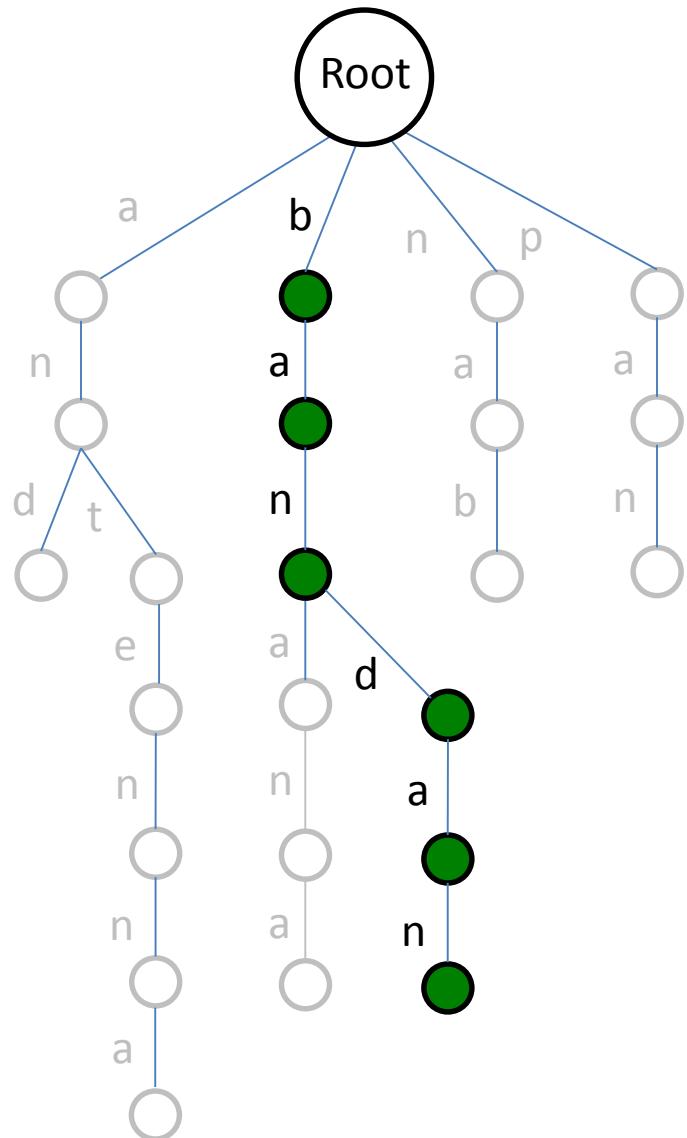
antenna

**bandana**

ananas

nana

# *Patterns*



banana

pan

and

nab

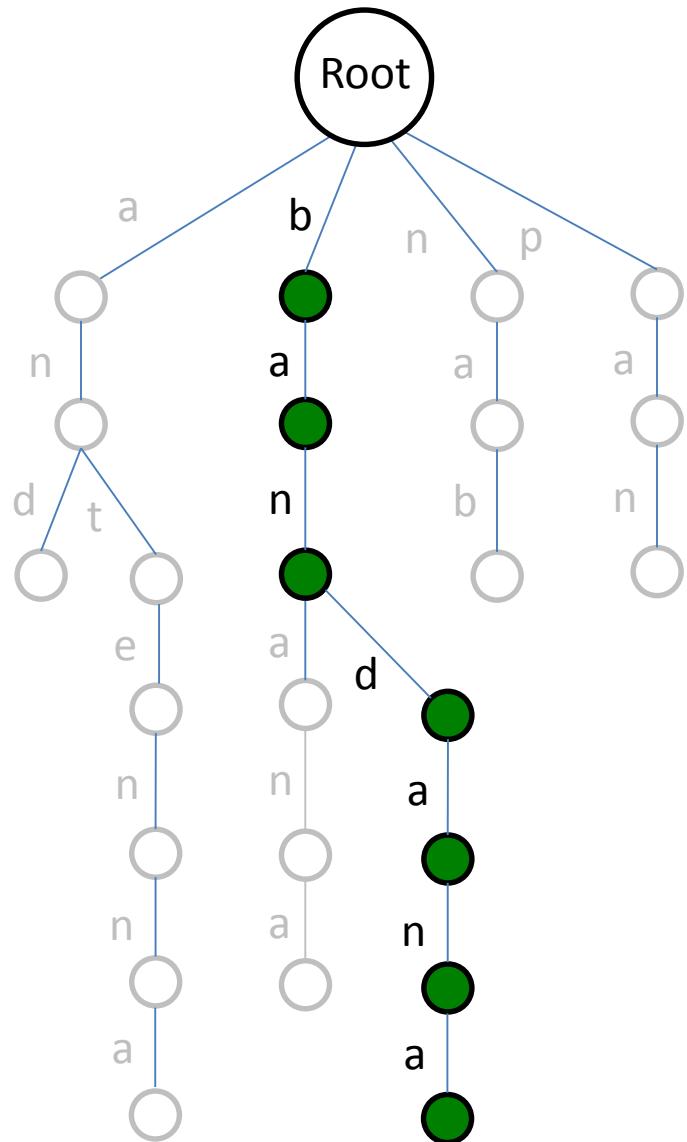
antenna

**bandana**

ananas

nana

# *Patterns*



banana

pan

and

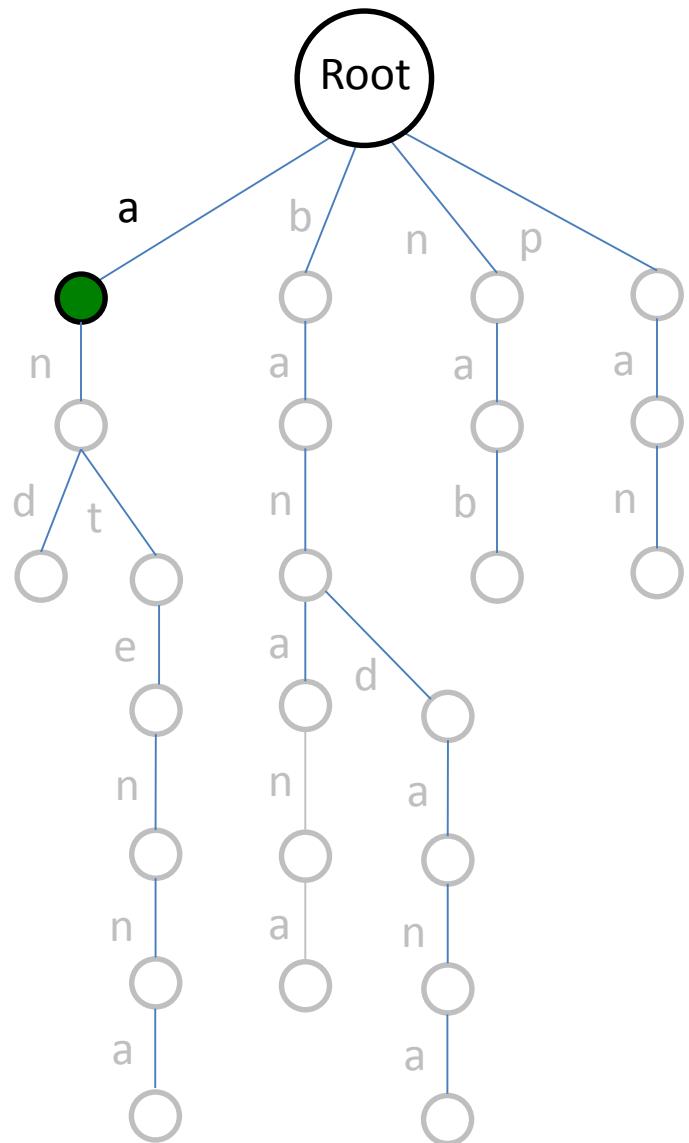
nab

antenna

**bandana**

ananas

nana



# *Patterns*

banana

pan

and

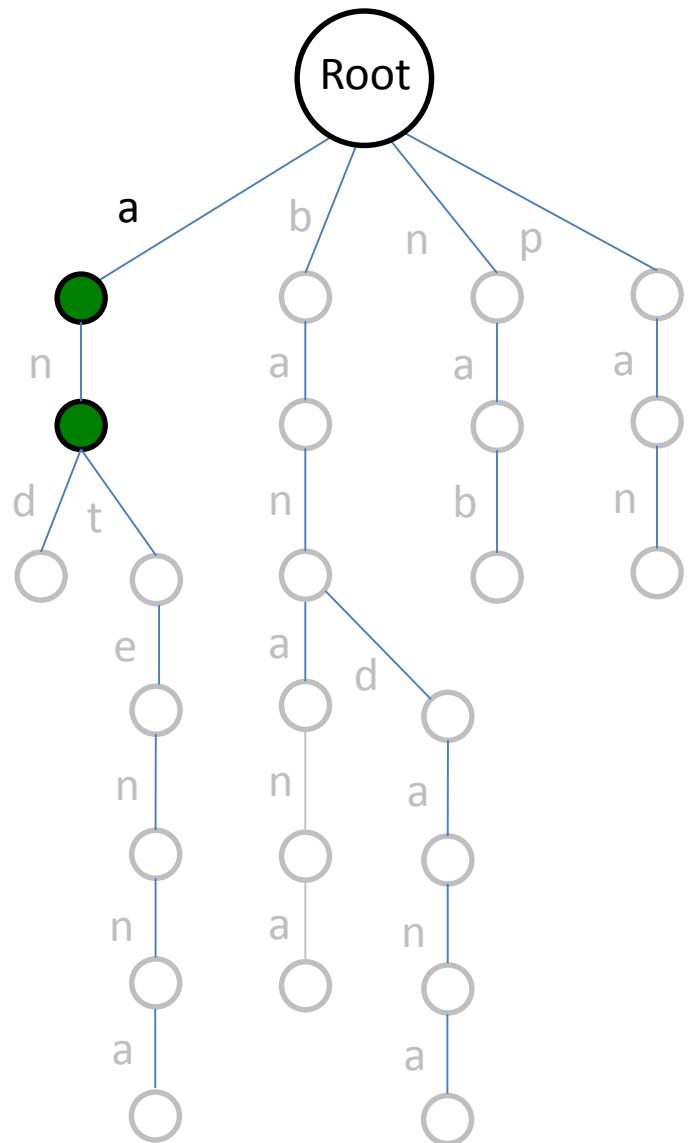
nab

antenna

bandana

**ananas**

nana



# *Patterns*

banana

pan

and

nab

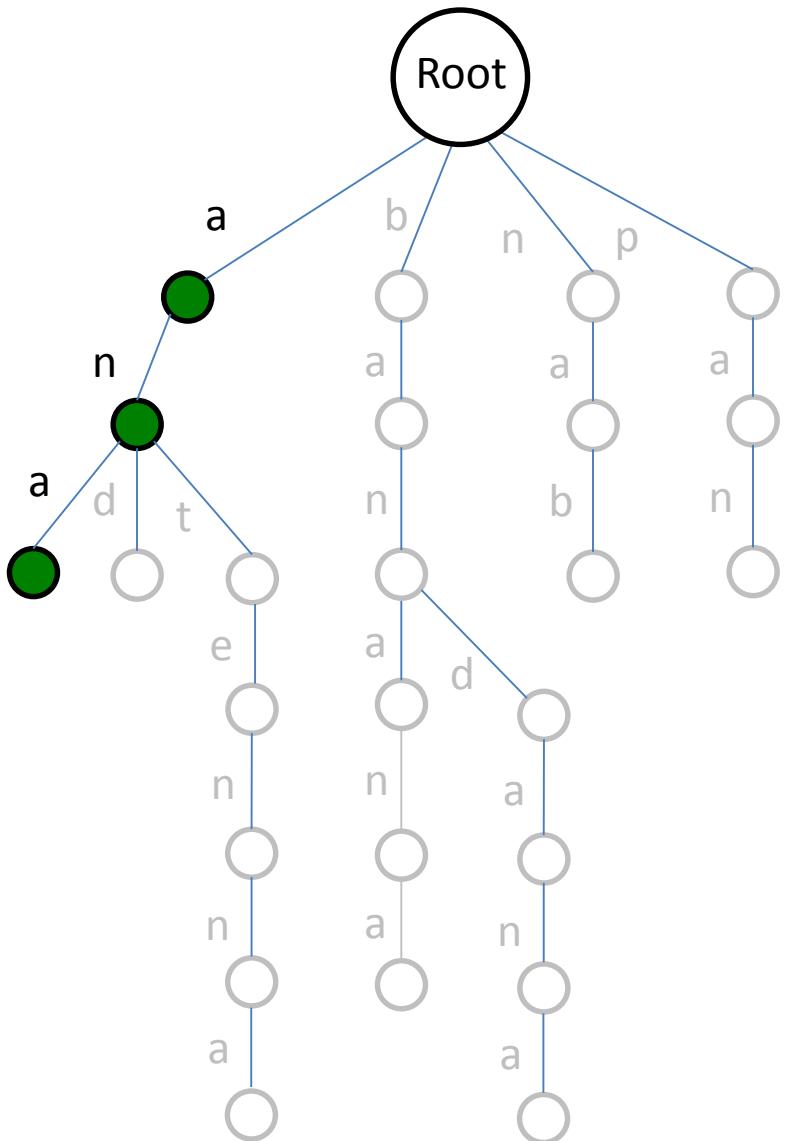
antenna

bandana

**ananas**

nana

# *Patterns*



banana

pan

and

nab

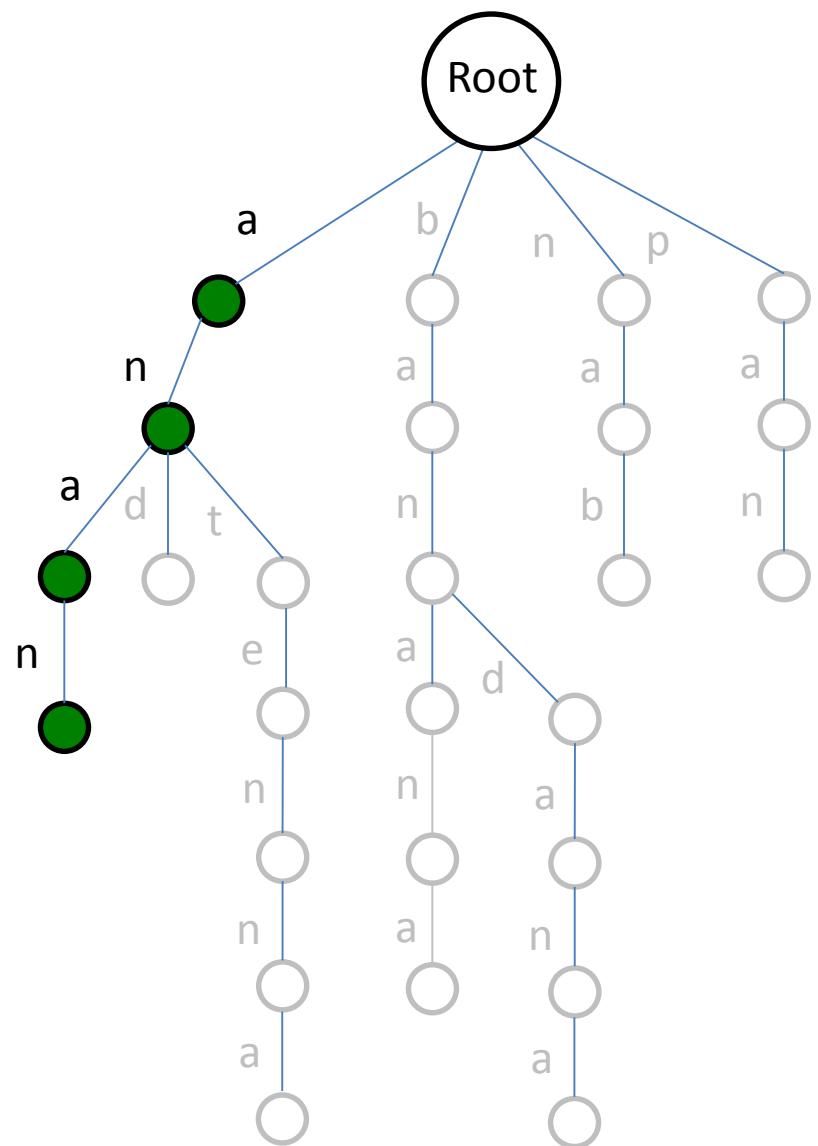
antenna

bandana

**ananas**

nana

# *Patterns*



banana

pan

and

nab

antenna

bandana

**ananas**

nana

# *Patterns*

# banana

pan

and

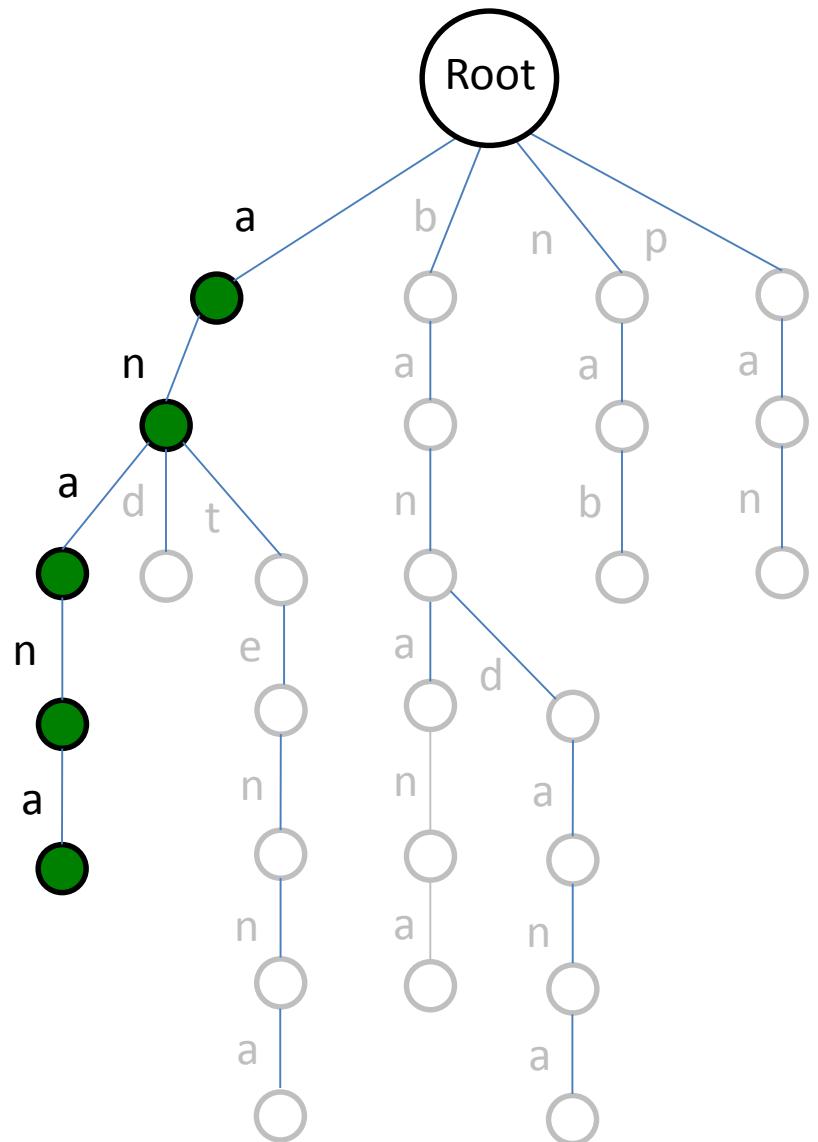
nab

## antenna

# bandana

# ananas

nana



# *Patterns*

# banana

pan

and

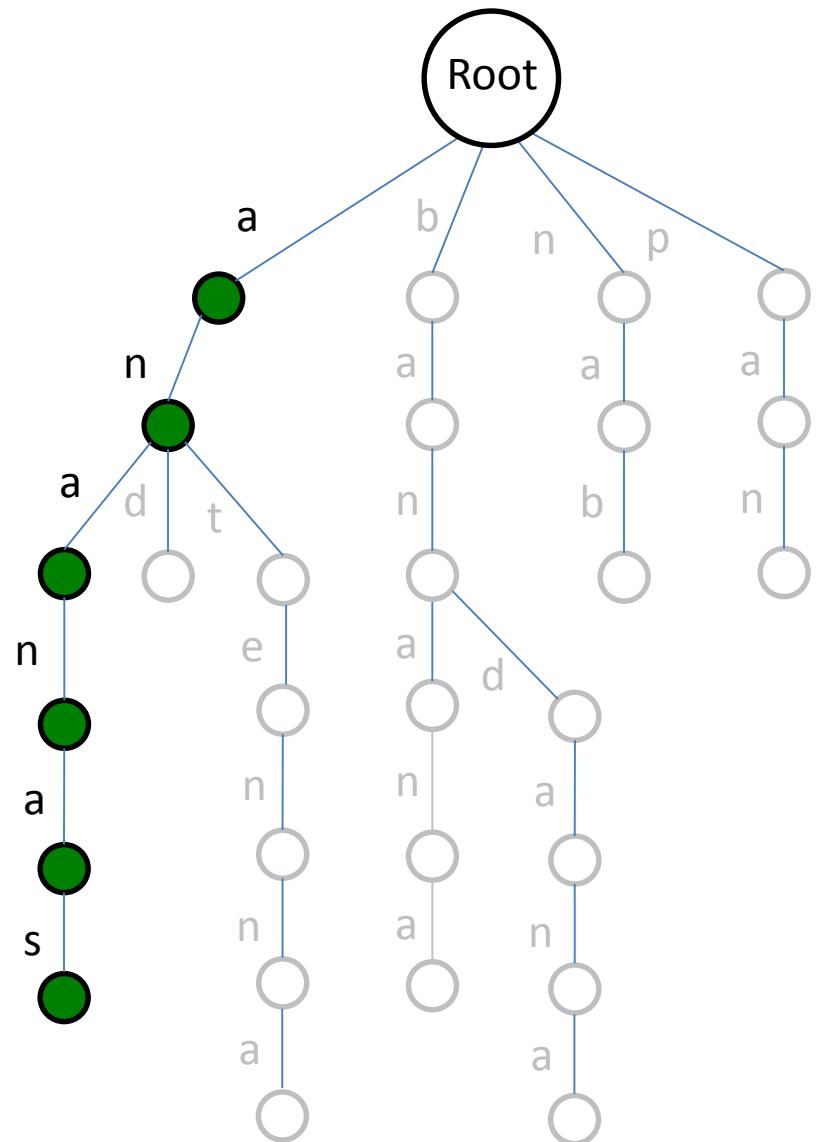
nab

## antenna

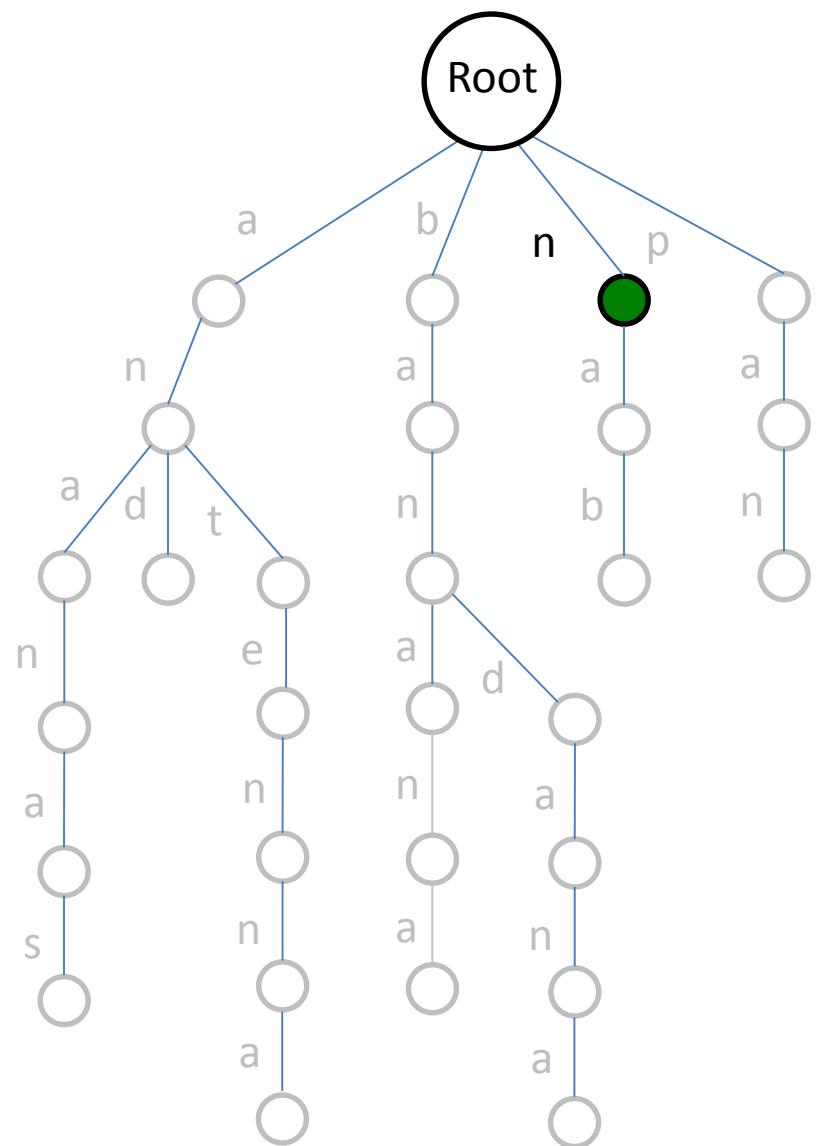
# bandana

# ananas

nana



# *Patterns*



banana

pan

and

nab

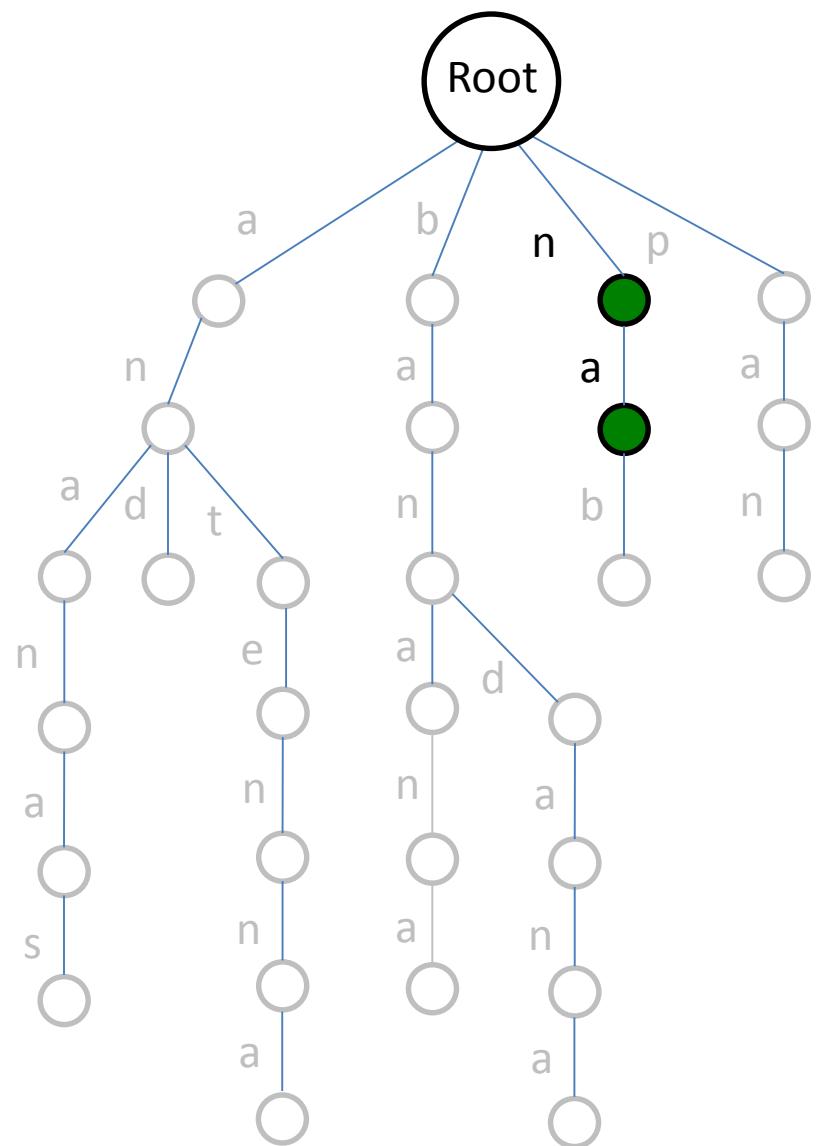
antenna

bandana

ananas

**nana**

# *Patterns*



banana

pan

and

nab

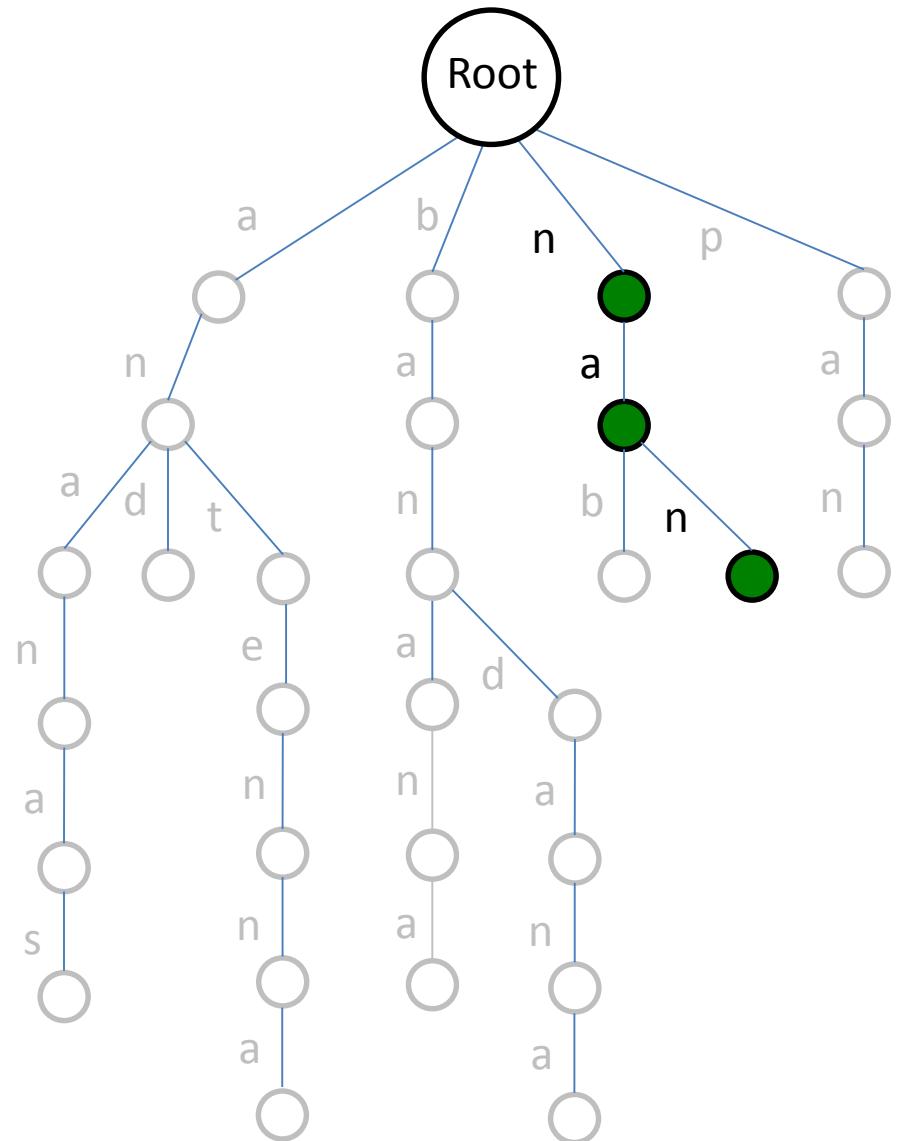
antenna

bandana

ananas

**nana**

# *Patterns*



banana

pan

and

nab

antenna

bandana

ananas

**nana**

# Patterns

banana

pan

and

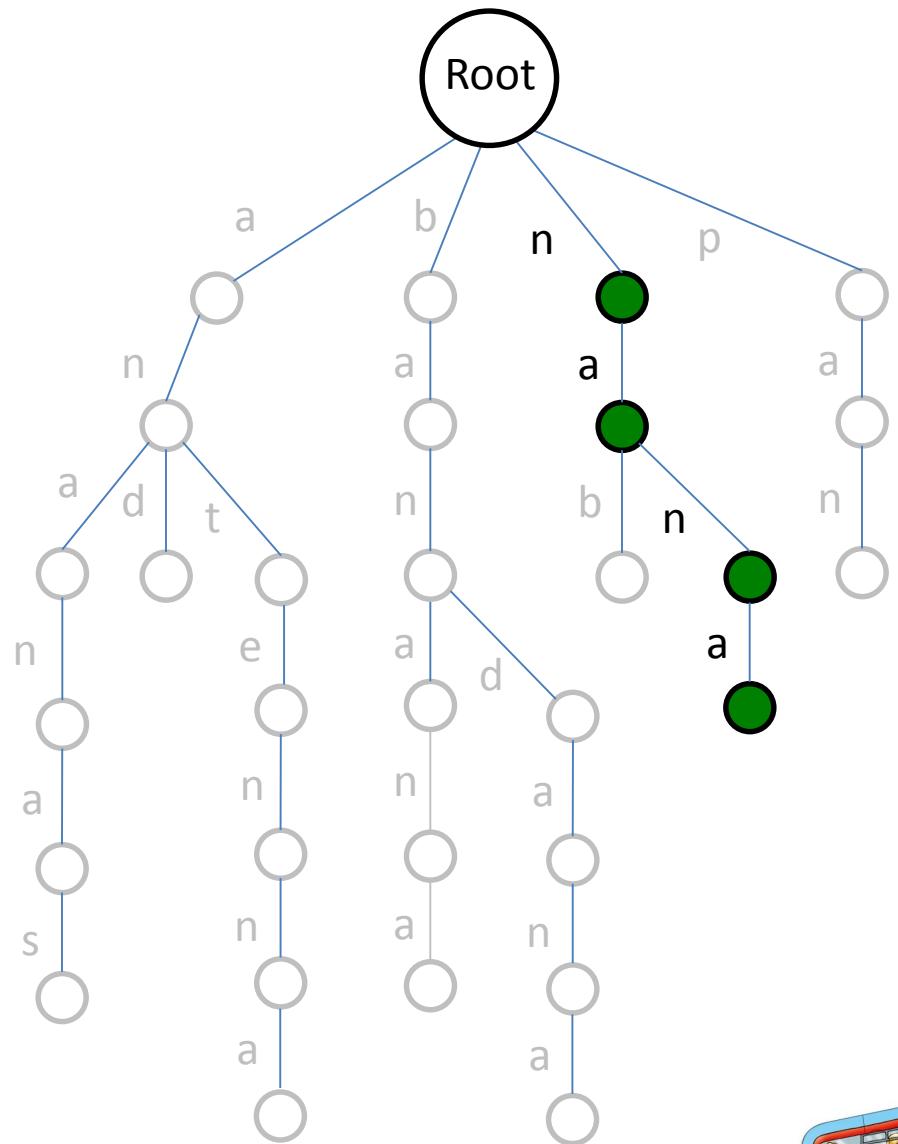
nab

## antenna

# bandana

ananas

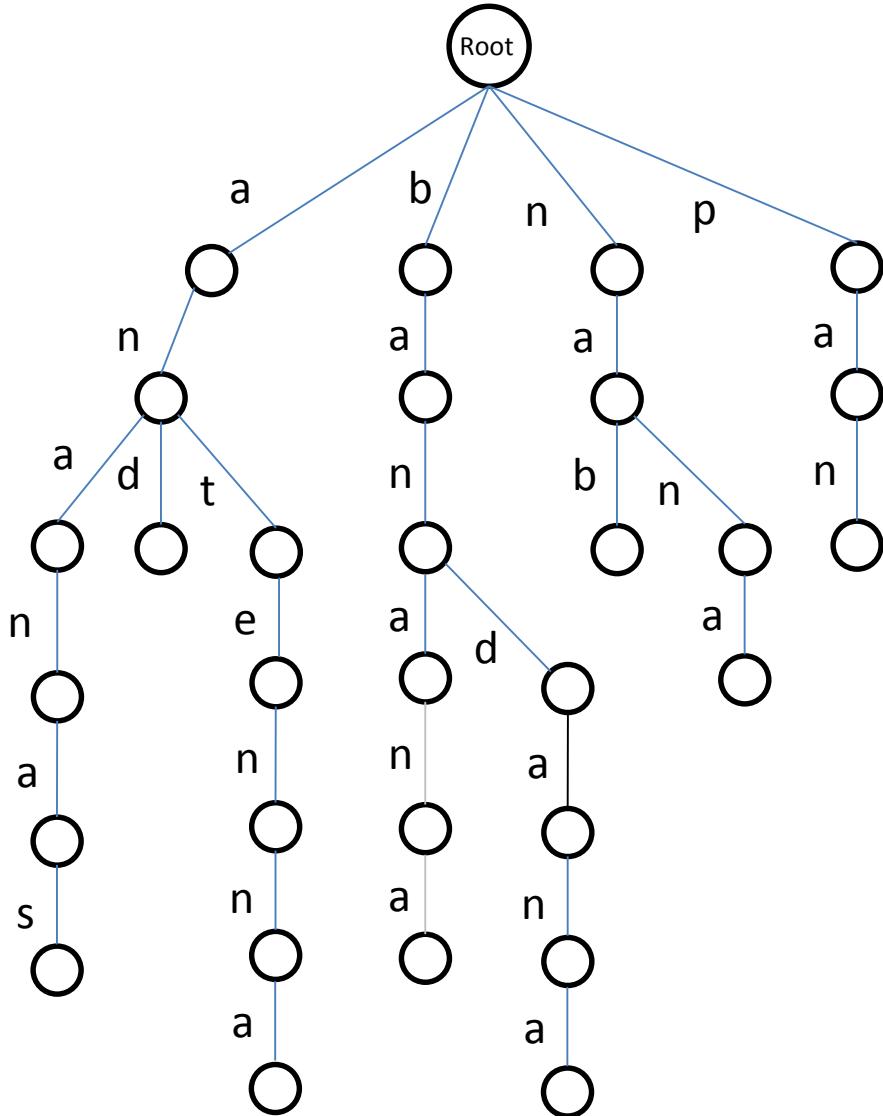
nana



# Trie(*Patterns*)



panamabananas

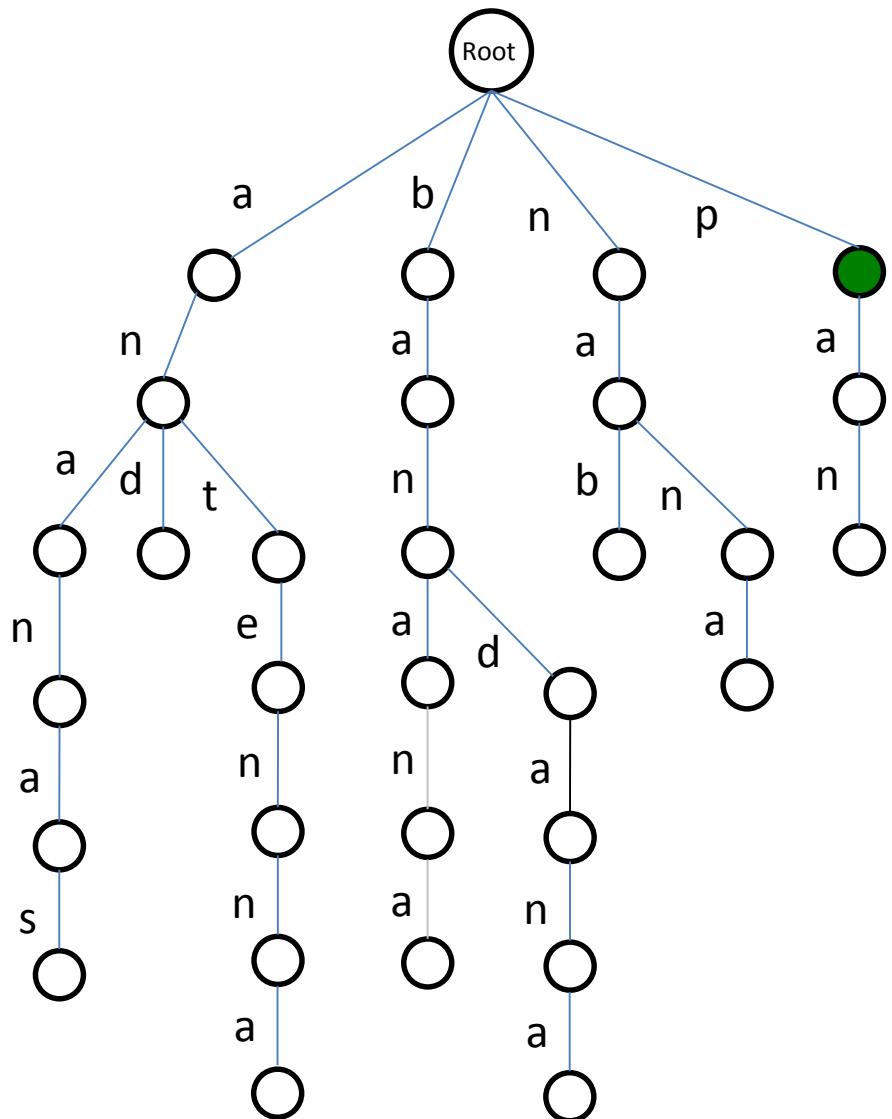


**TrieMatching(*Text*, *Patterns*):**  
drive  $\text{Trie}(\text{Patterns})$  along *Text*  
at each position of *Text*

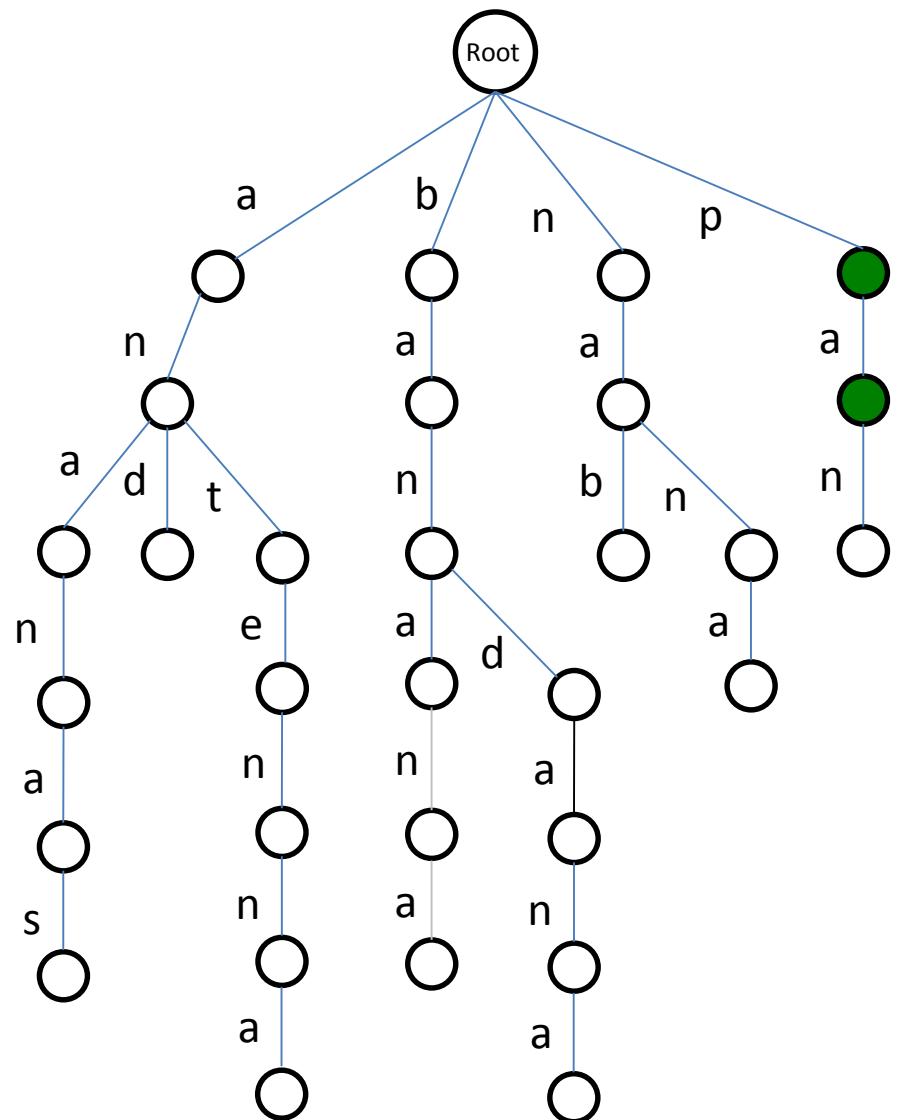
- walk down  $\text{Trie}(\text{Patterns})$   
by spelling symbols of *Text*
- a pattern from *Patterns*  
matches *Text* each time  
you reach a leaf!

For simplicity, we assume that no pattern is a substring of another pattern

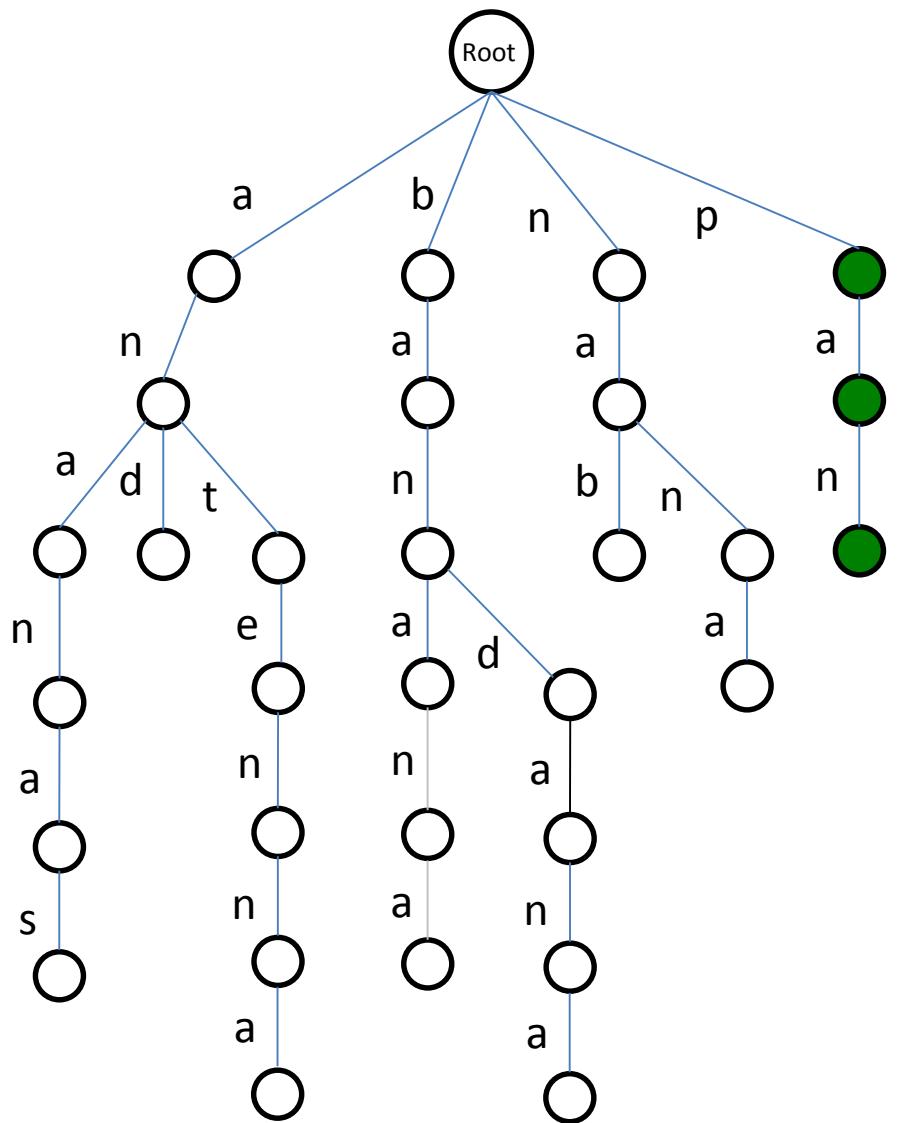
**p**anamabanas



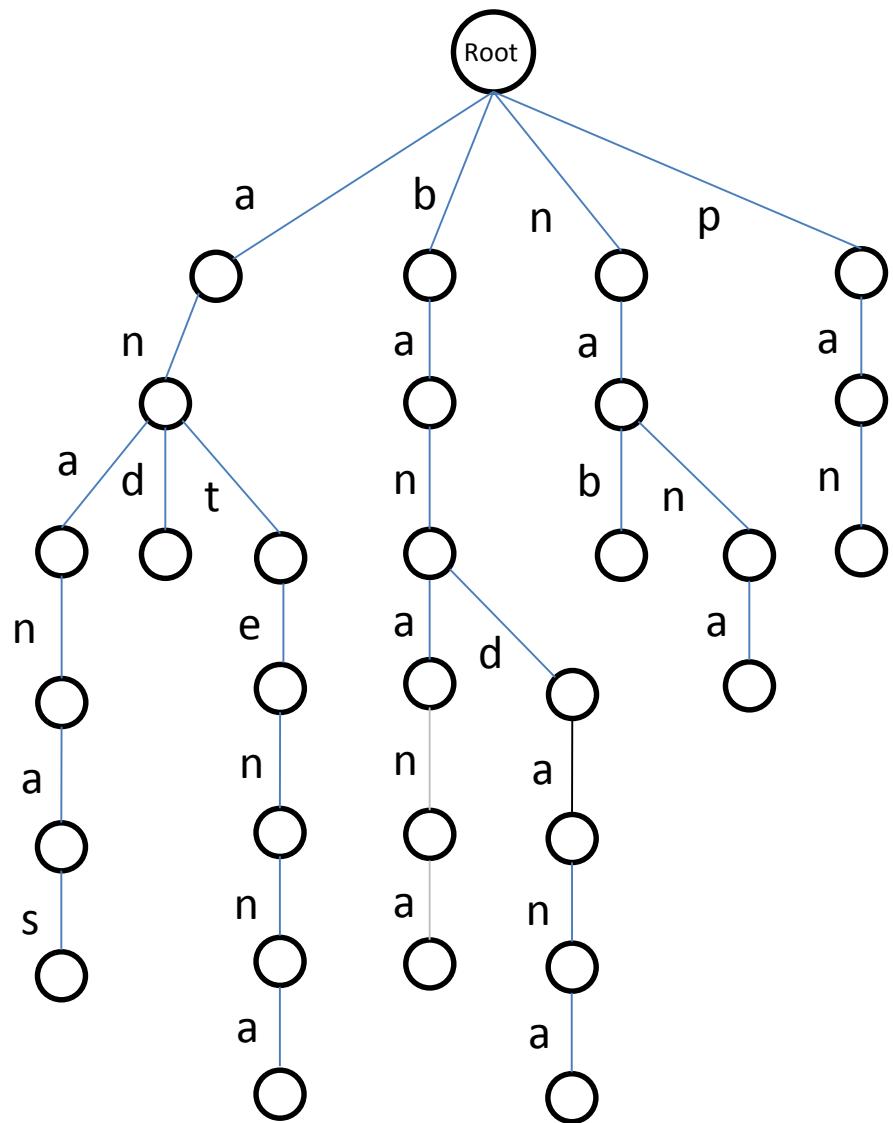
# p a n a m a b a n a n a s



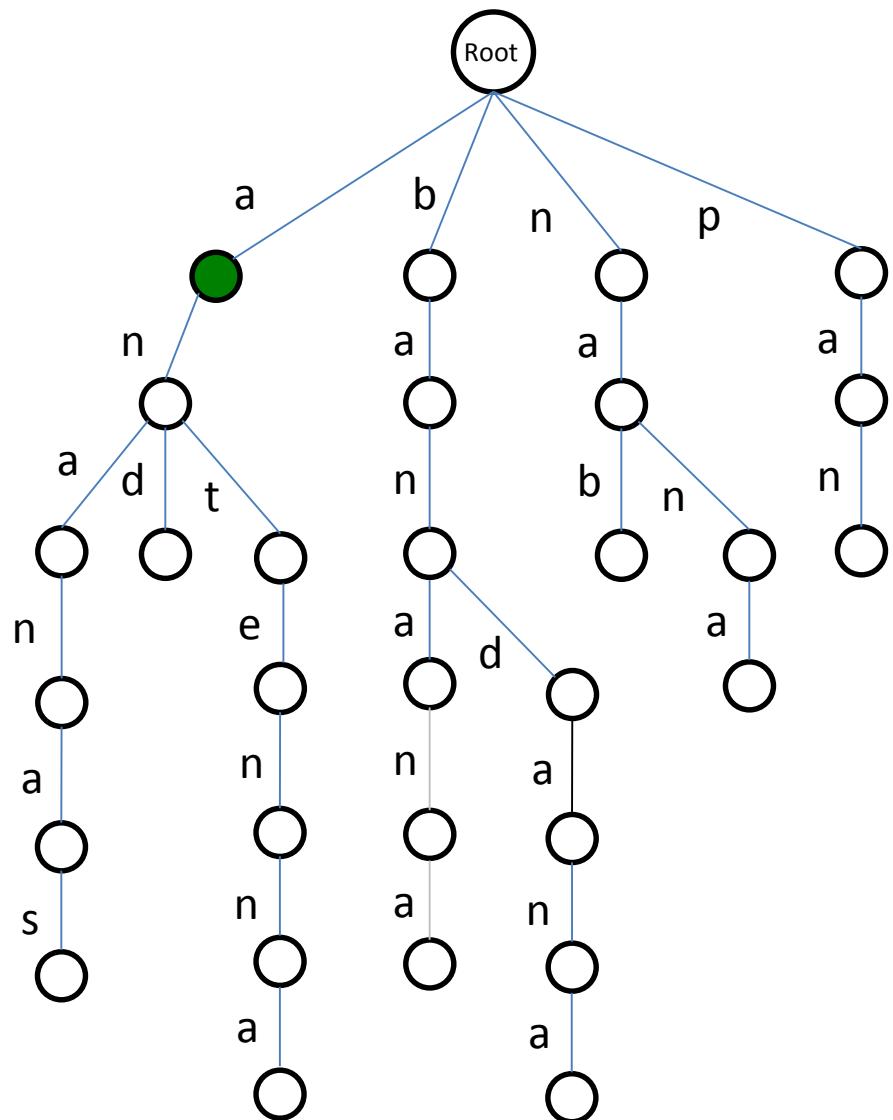
**p a n a m a b a n a n a s**



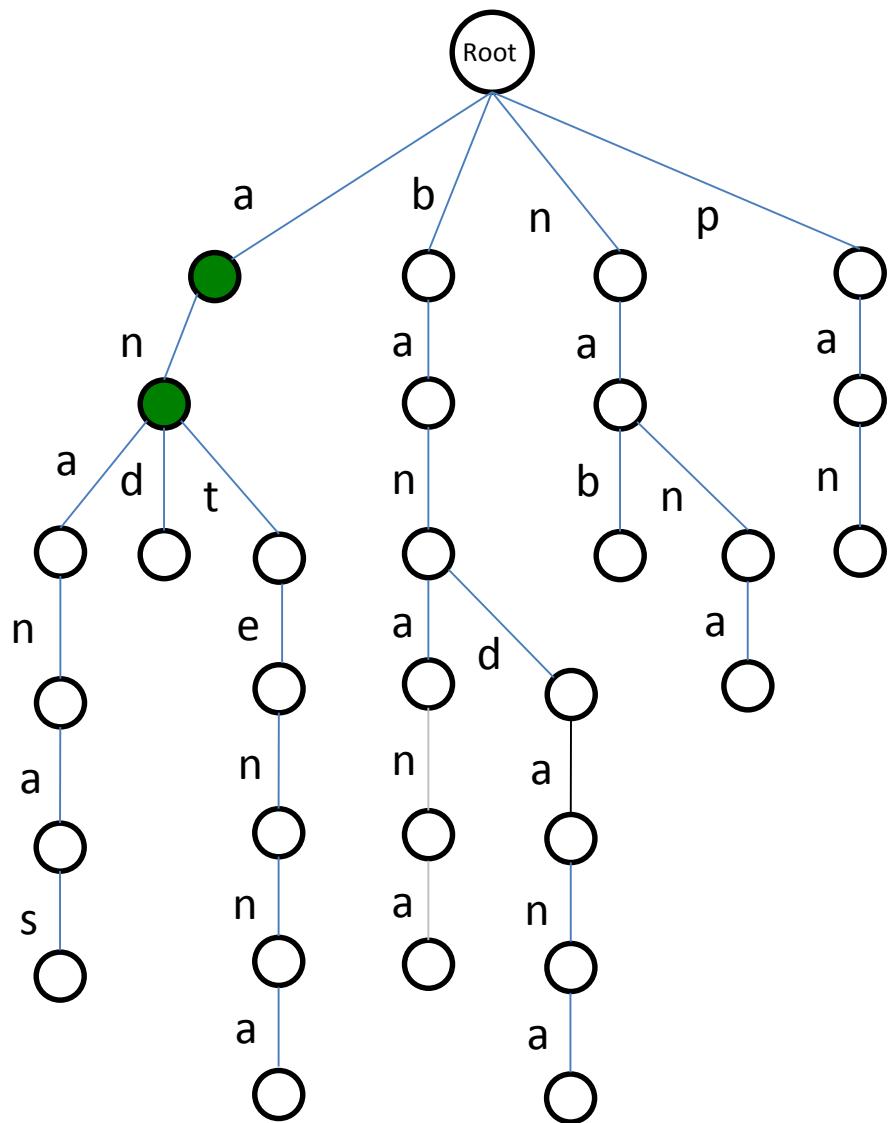
p a n a m a b a n a n a s



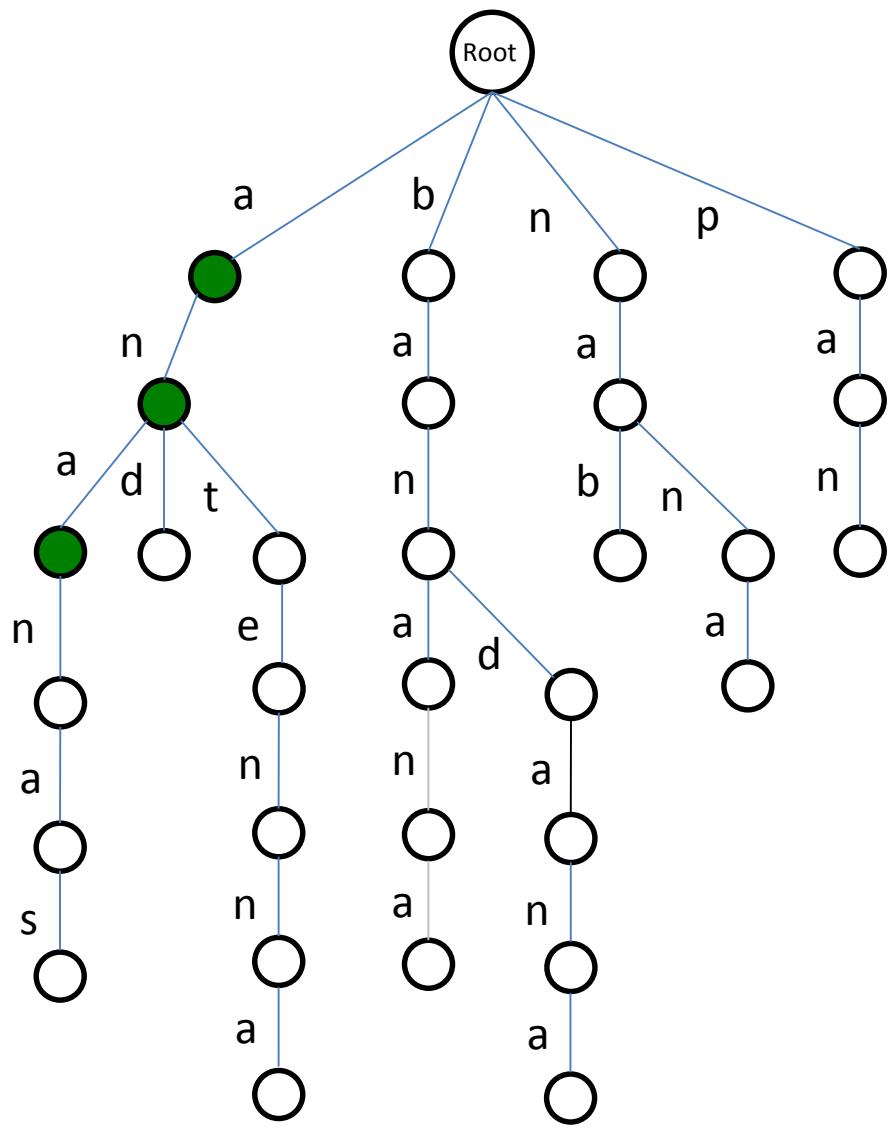
p a n a m a b a n a n a s



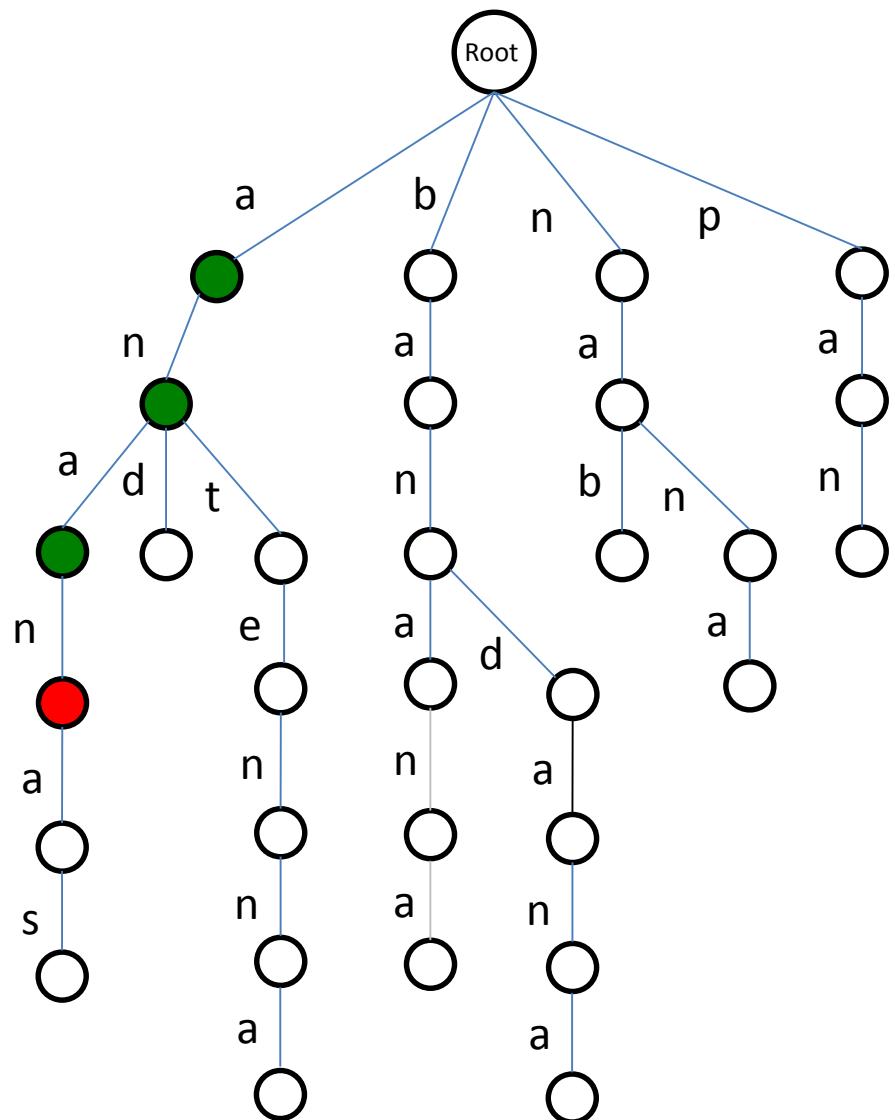
p **a**n a m a b a n a n a s



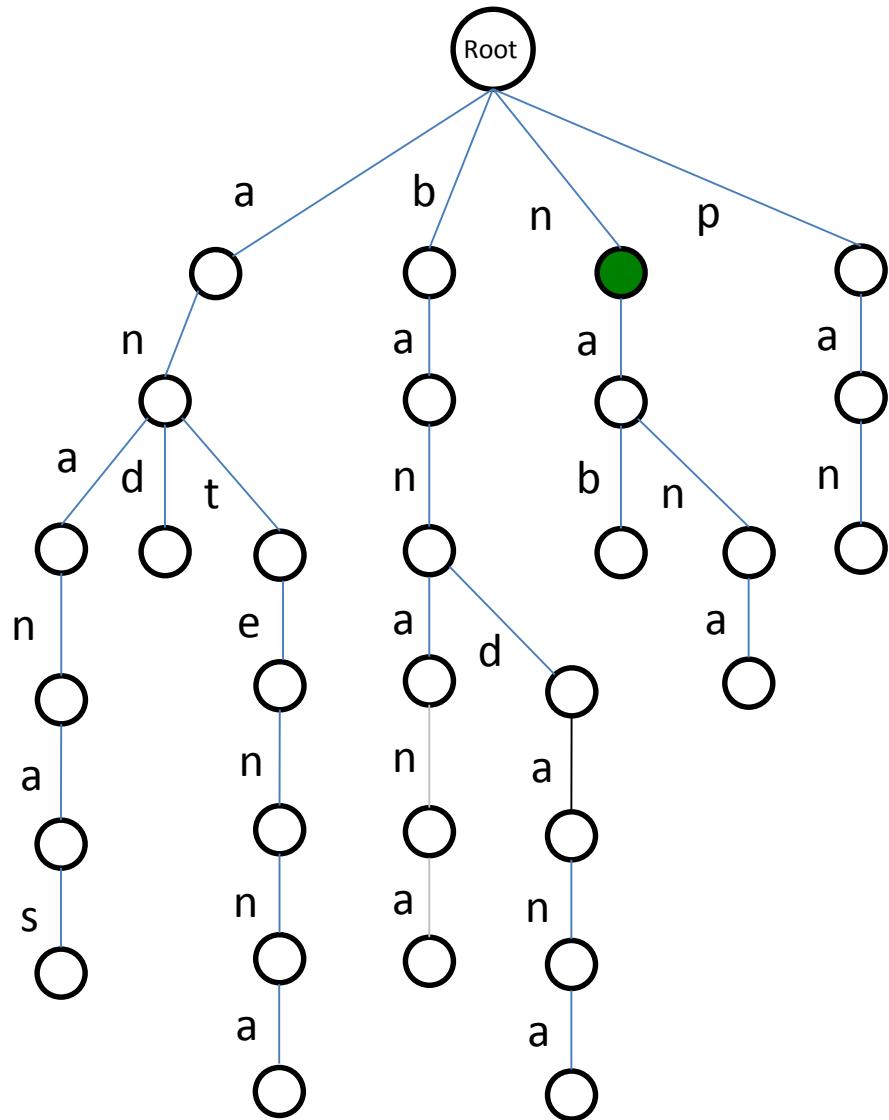
p a n a m a b a n a n a s



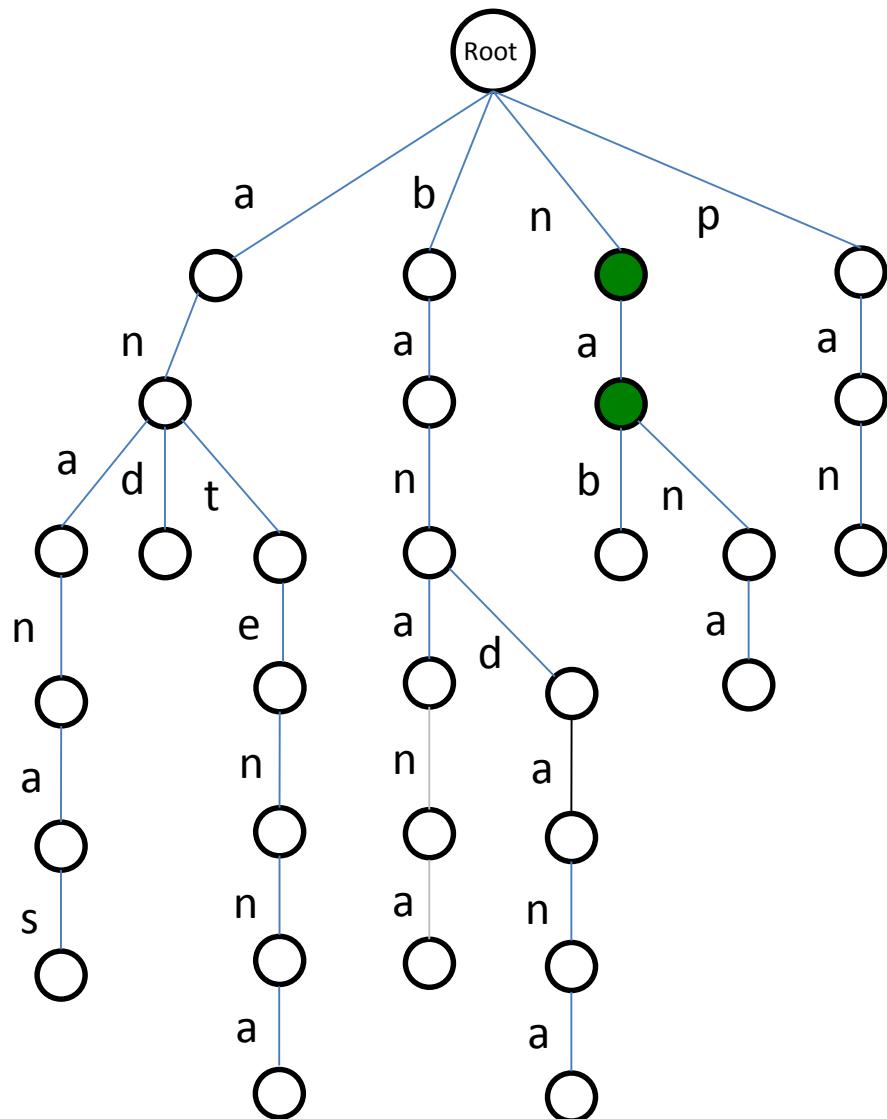
p a n a m a b a n a n a s



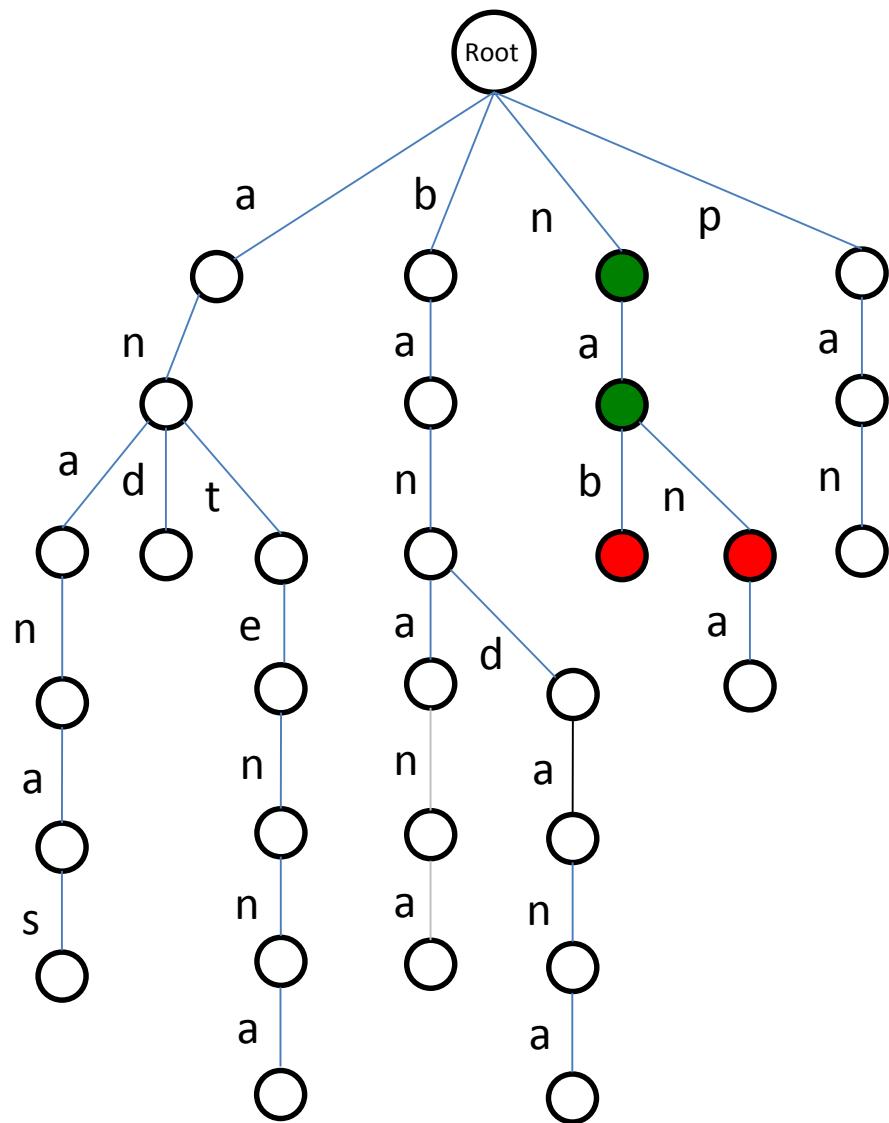
p a n a m a b a n a n a s



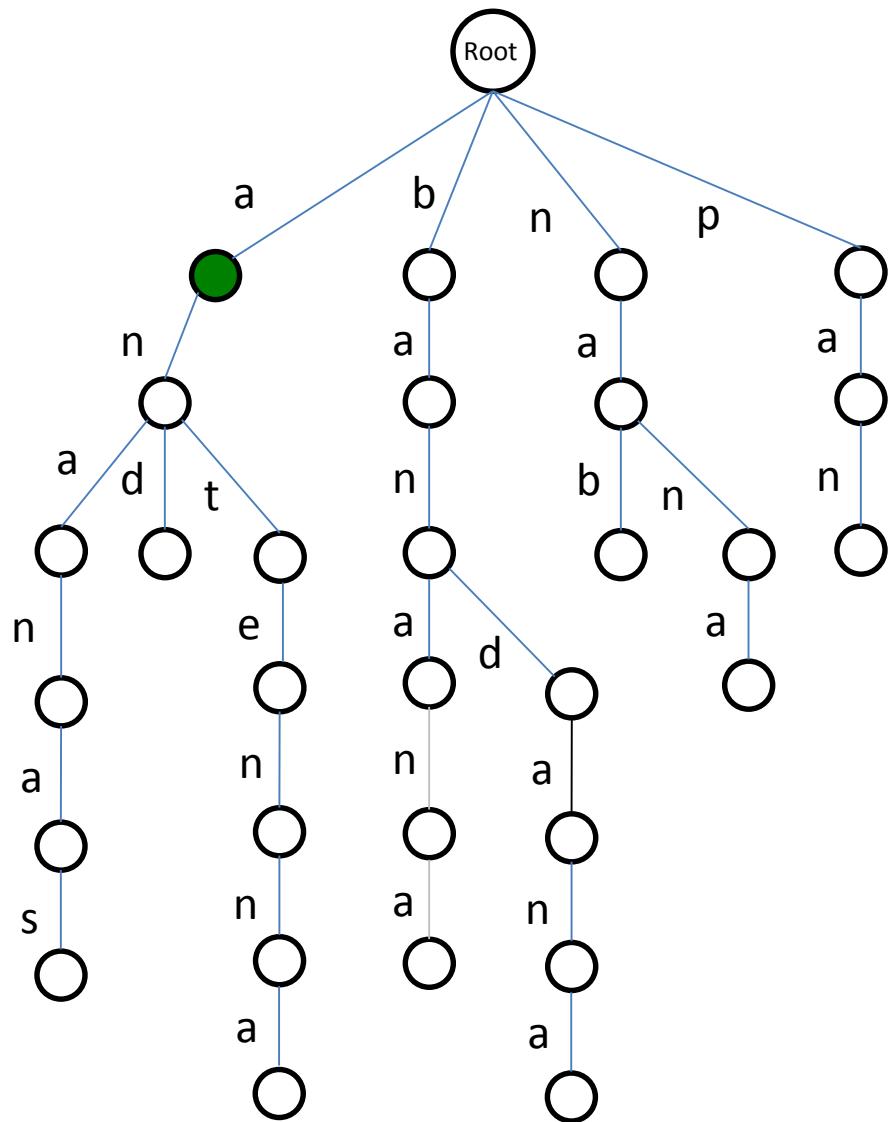
p a n a m a b a n a n a s



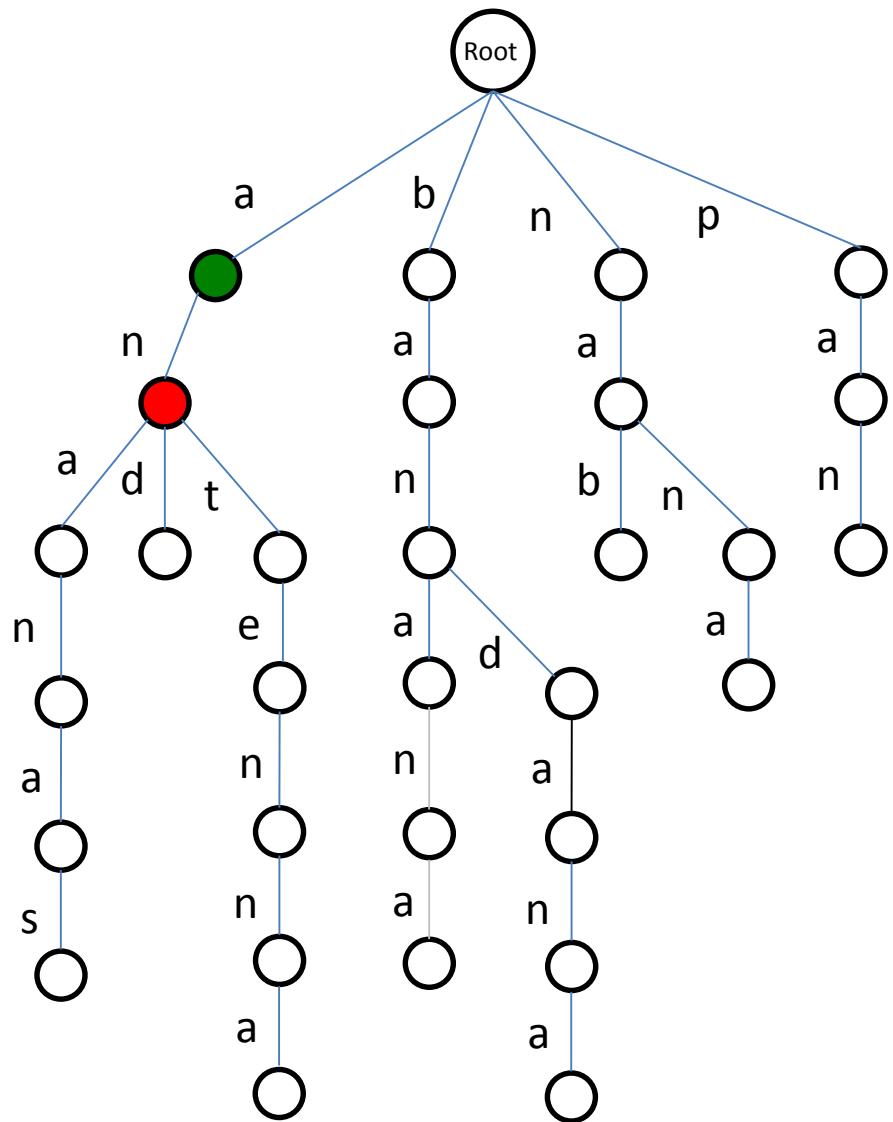
p a n a m a b a n a n a s



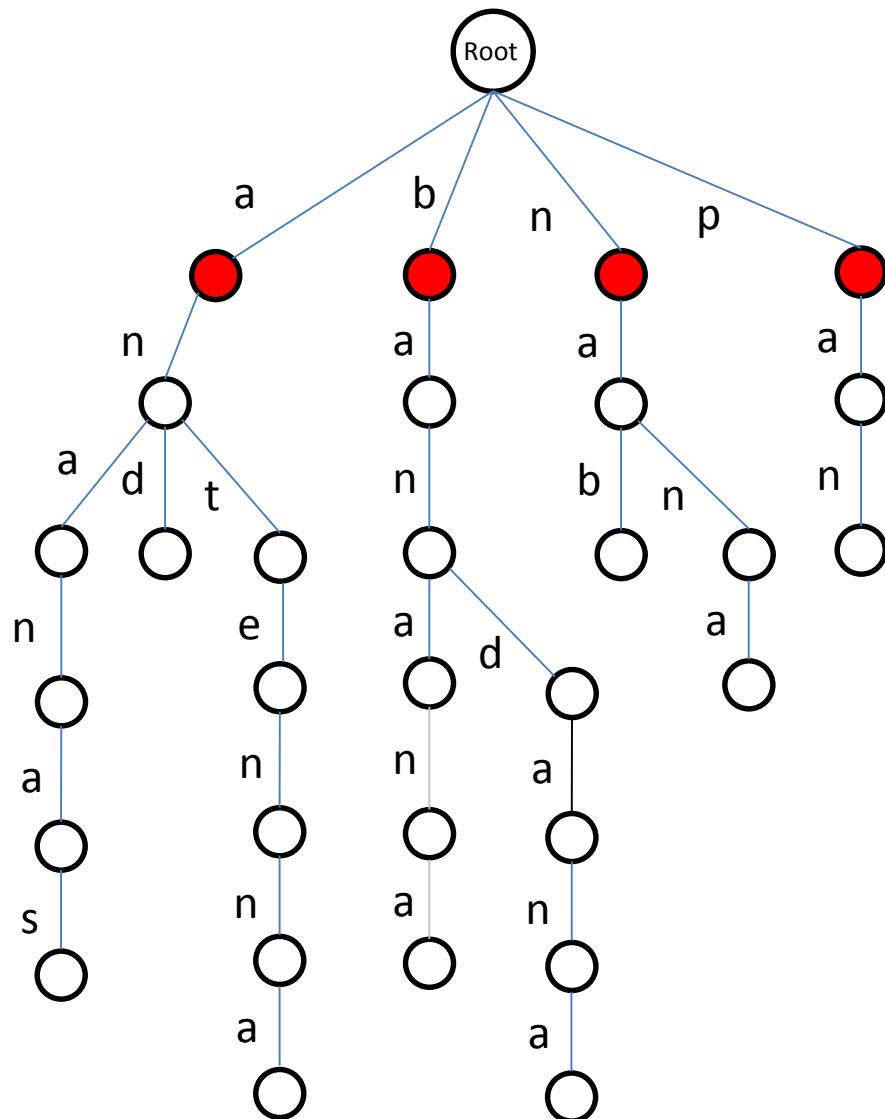
p a n a m a b a n a n a s



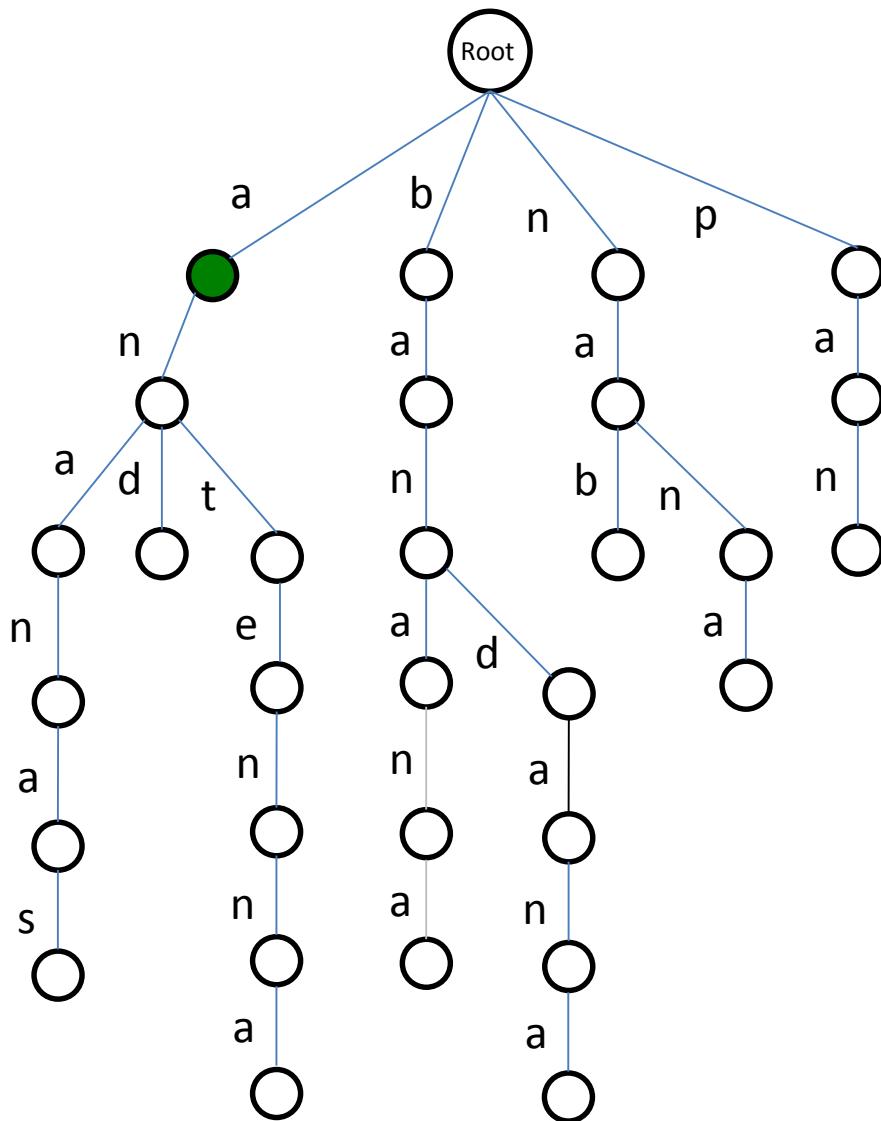
p a n a m a b a n a n a s



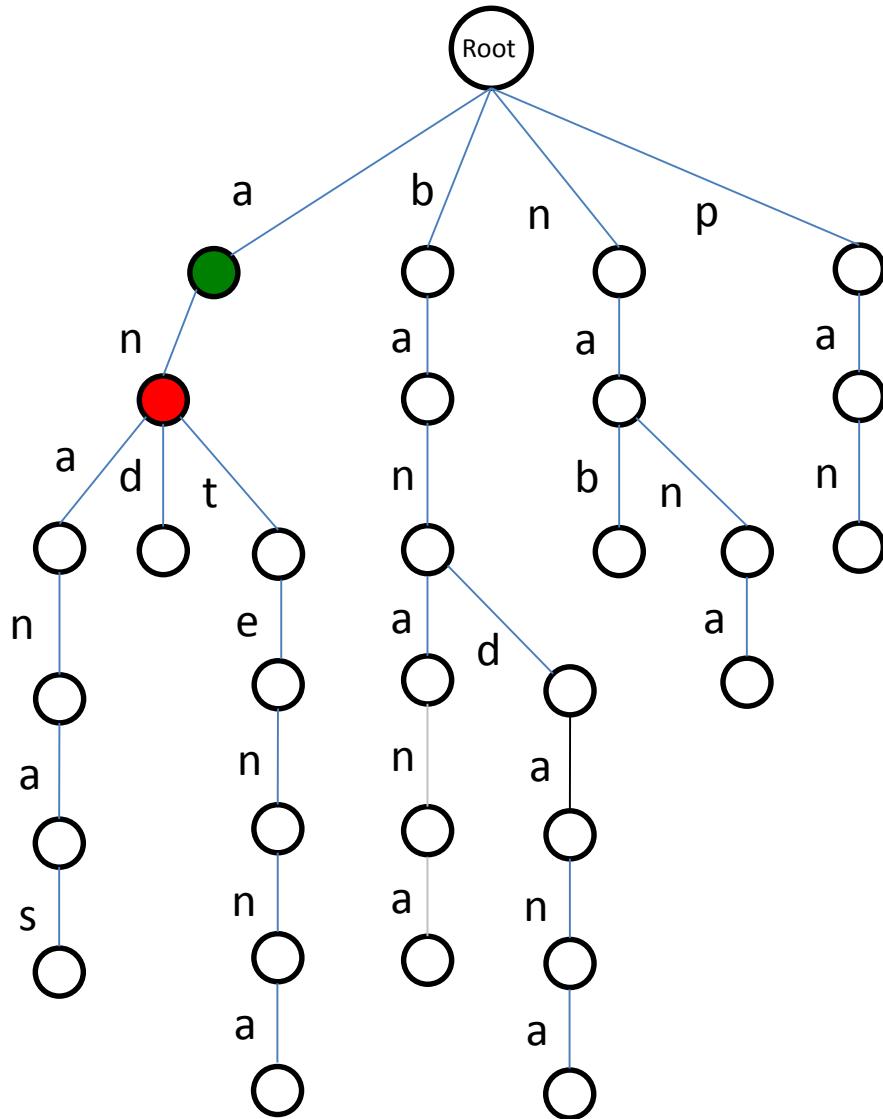
p a n a m a b a n a n a s



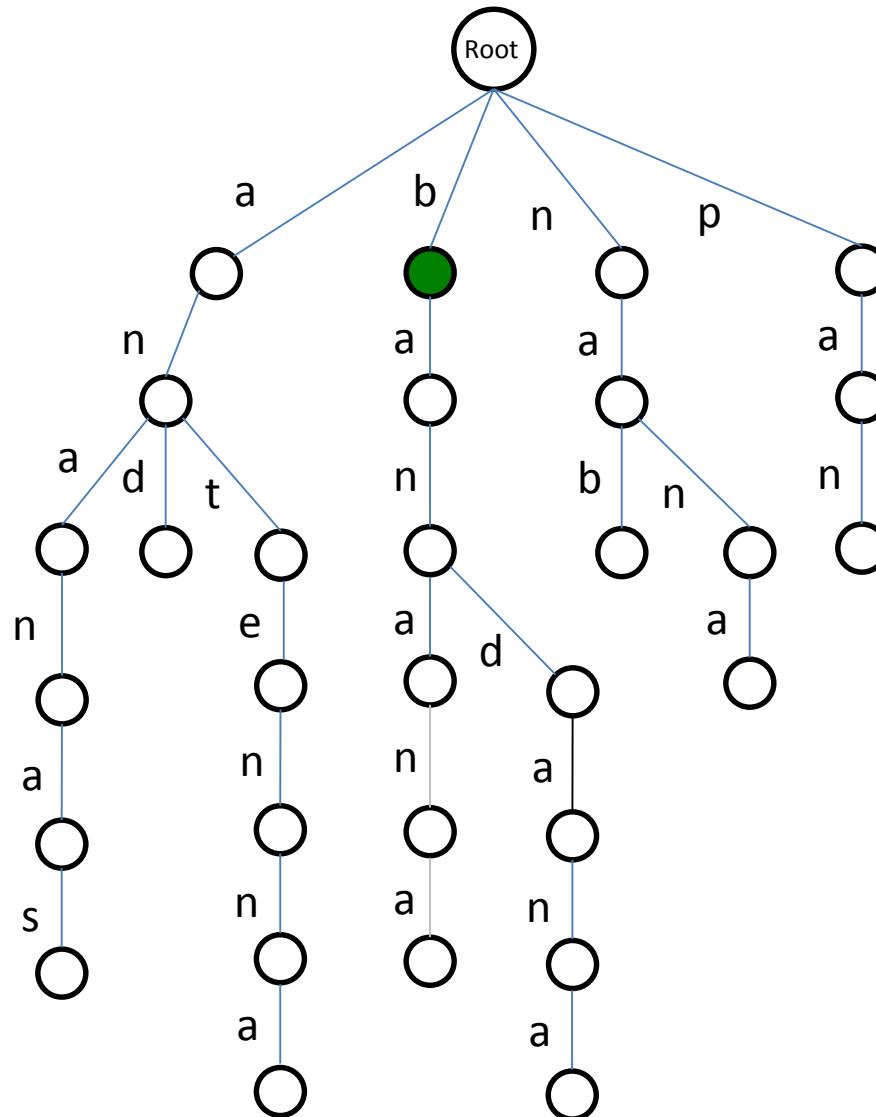
panam**a**bananas



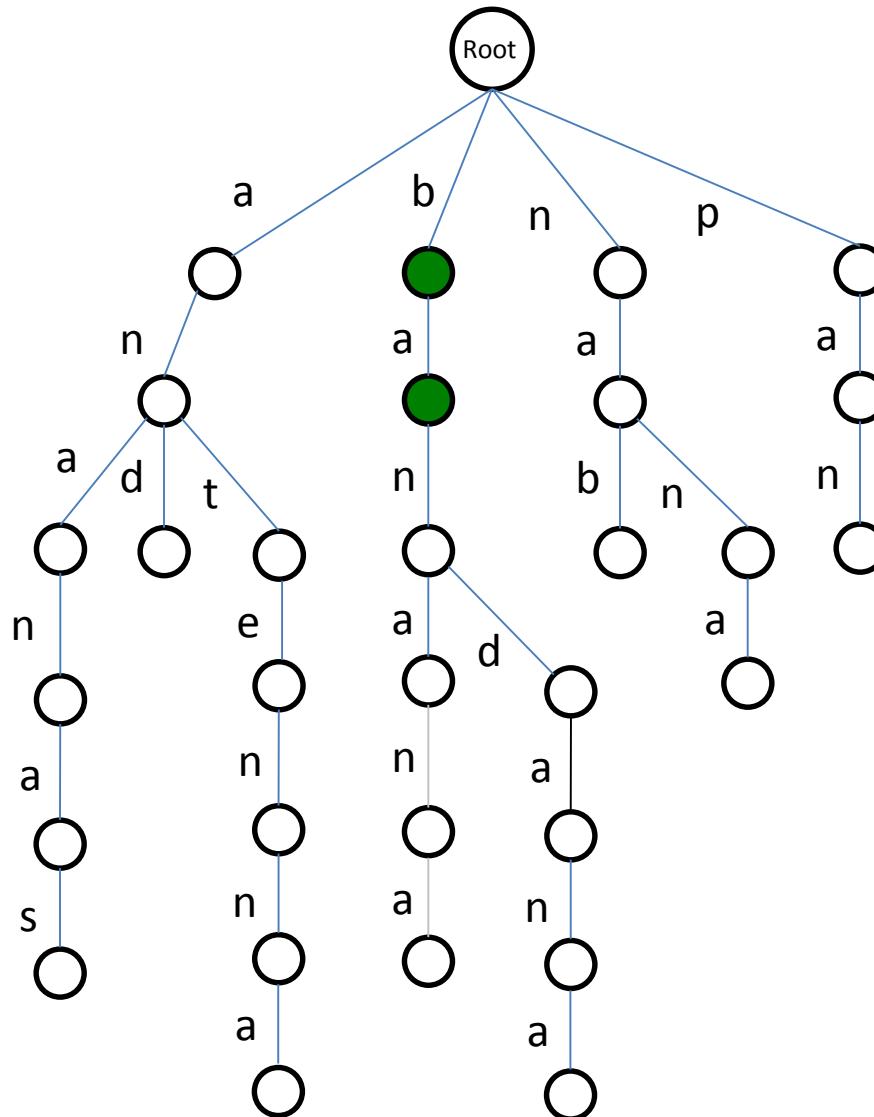
panam**a****b**ananas



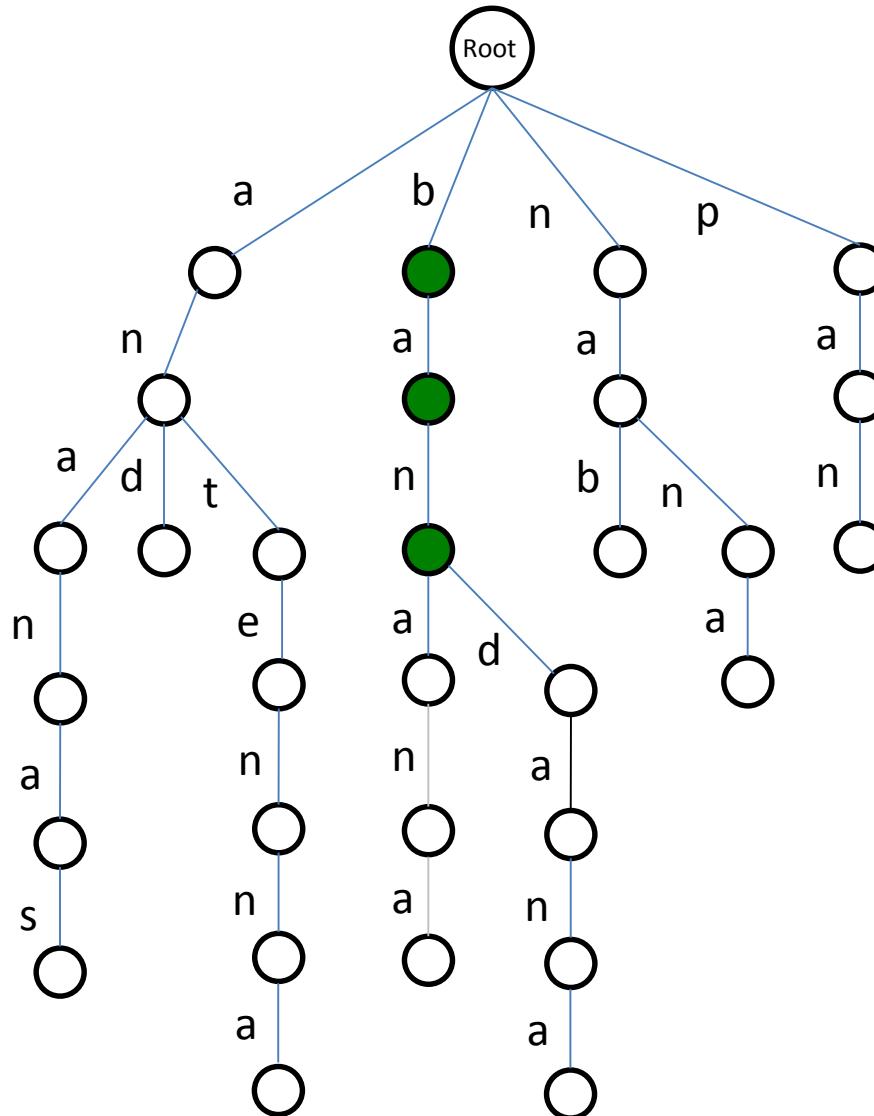
panama**ba**nanas



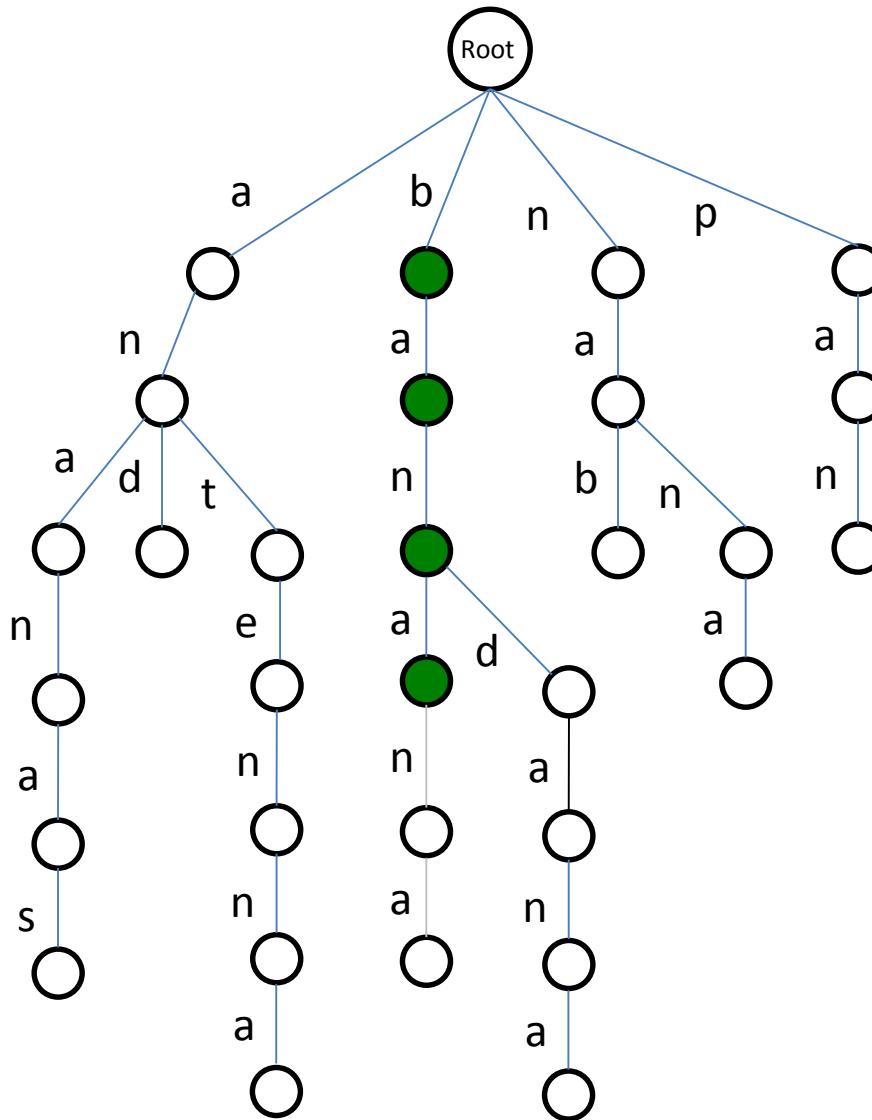
panama**ba**nanas



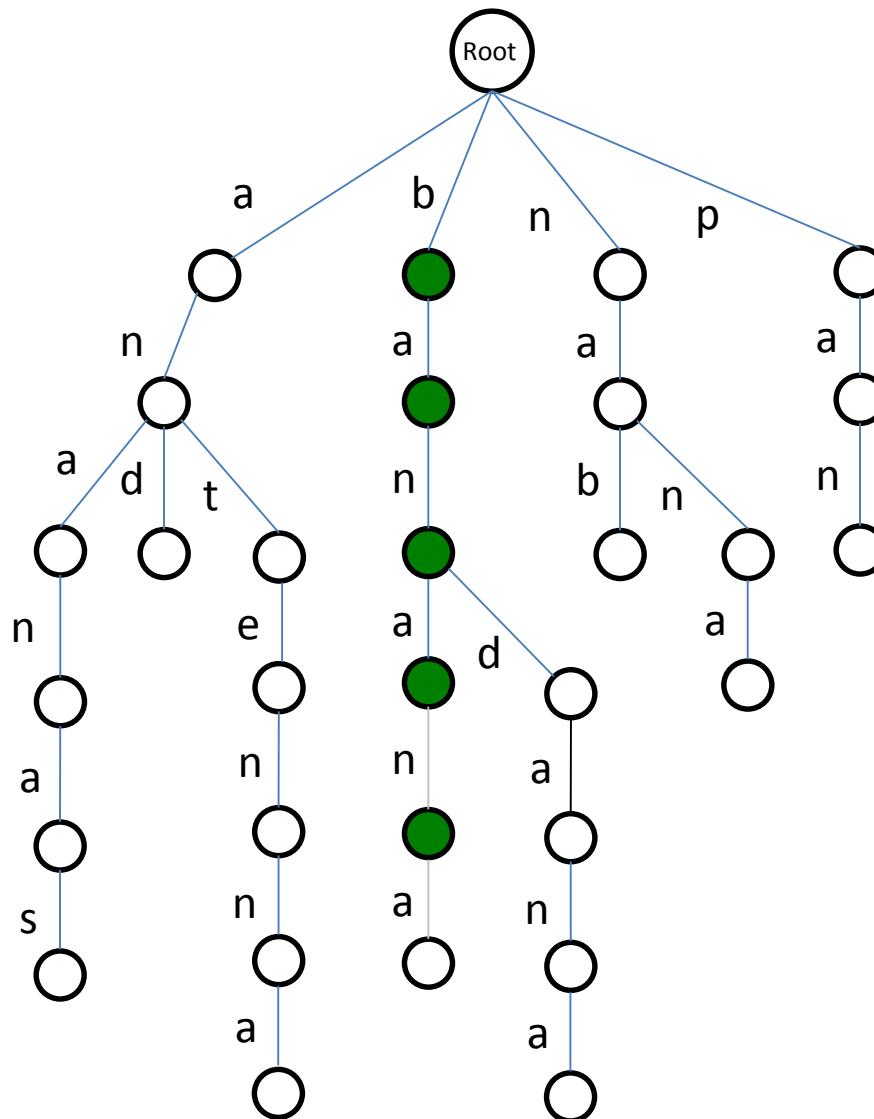
panama**bananas**



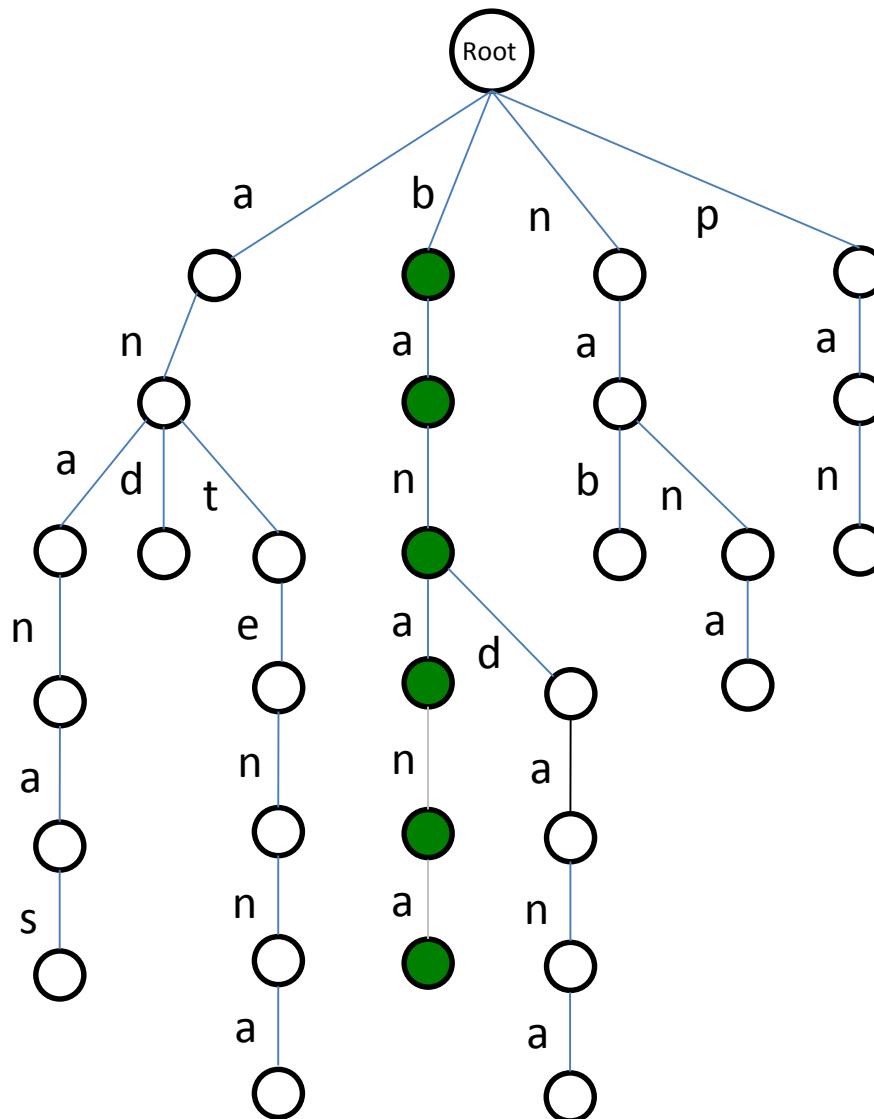
panama **bananas**



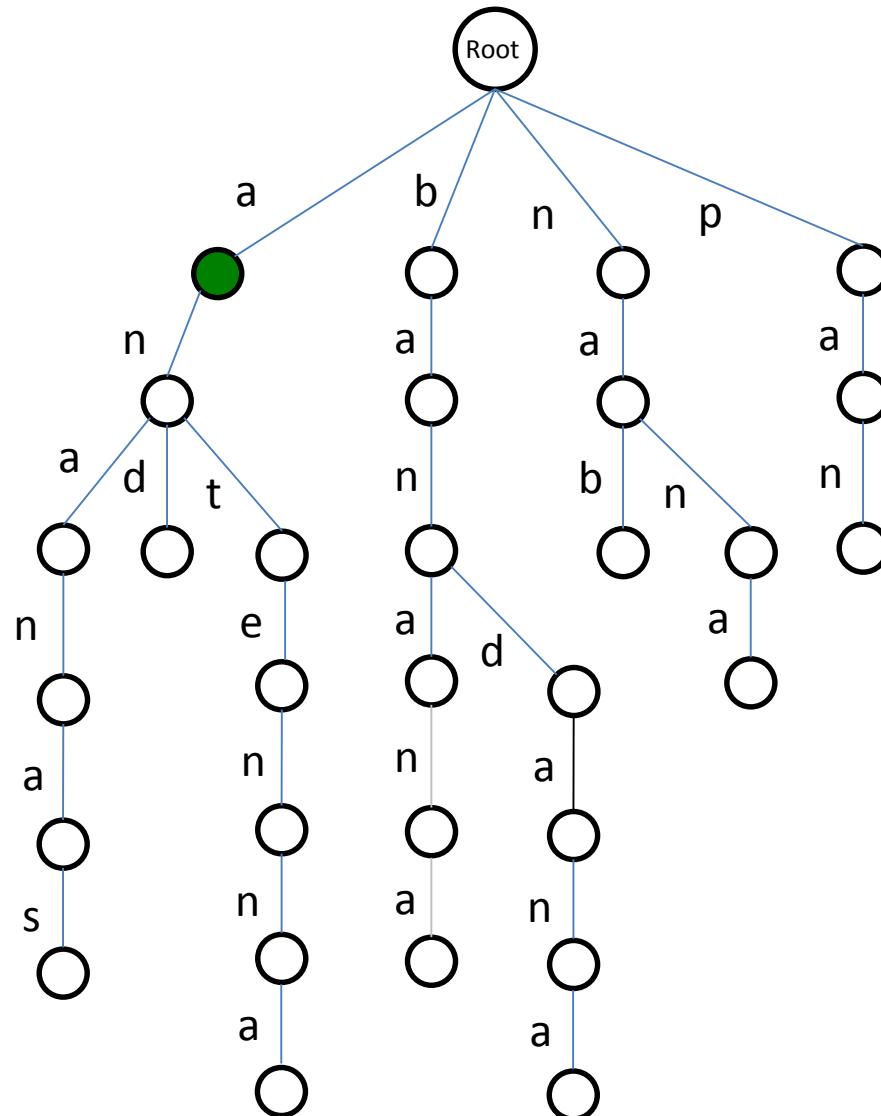
panama **bananas**



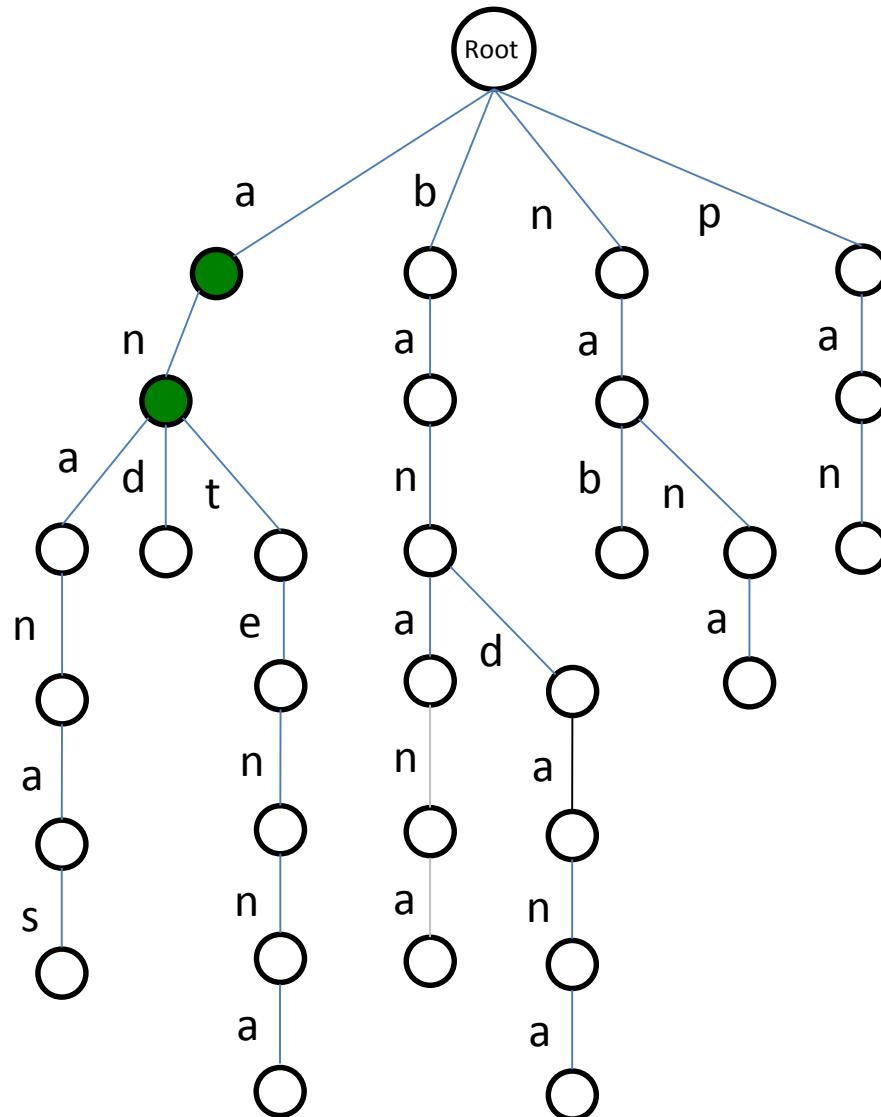
panama **bananas**



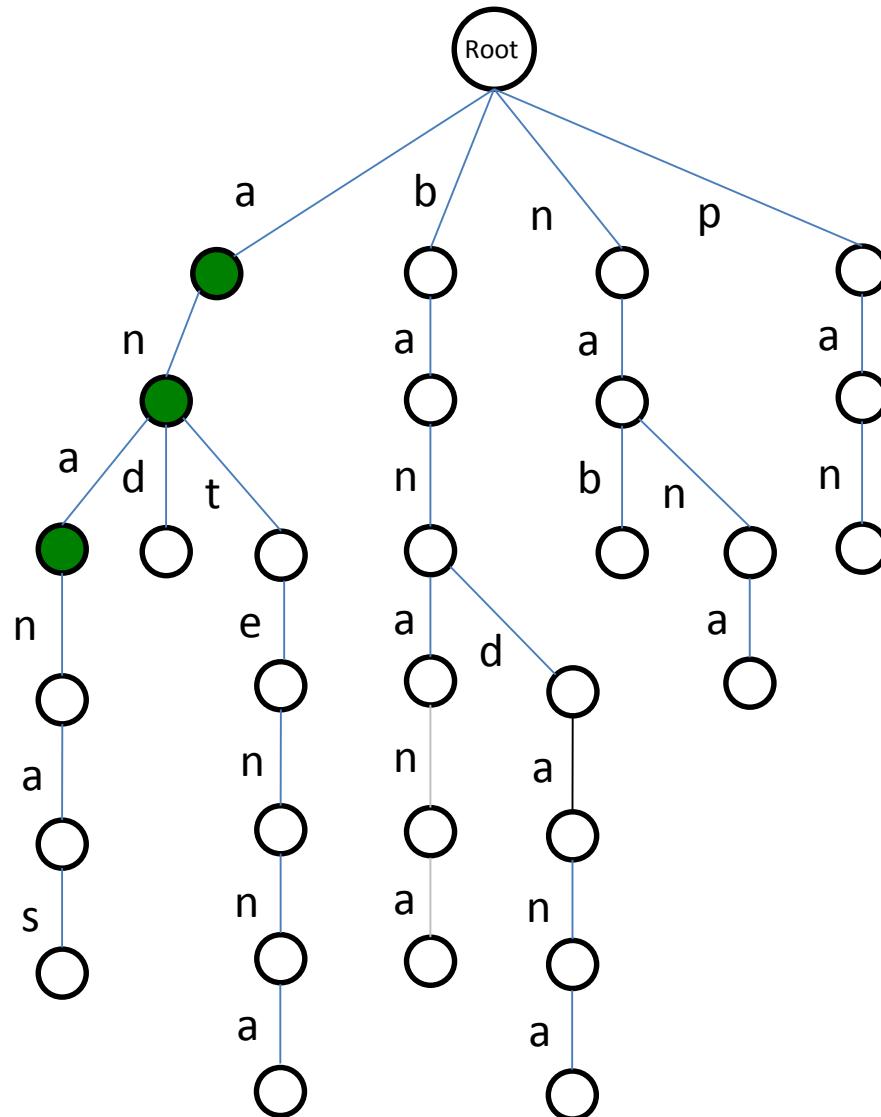
panamab[ananas](#)



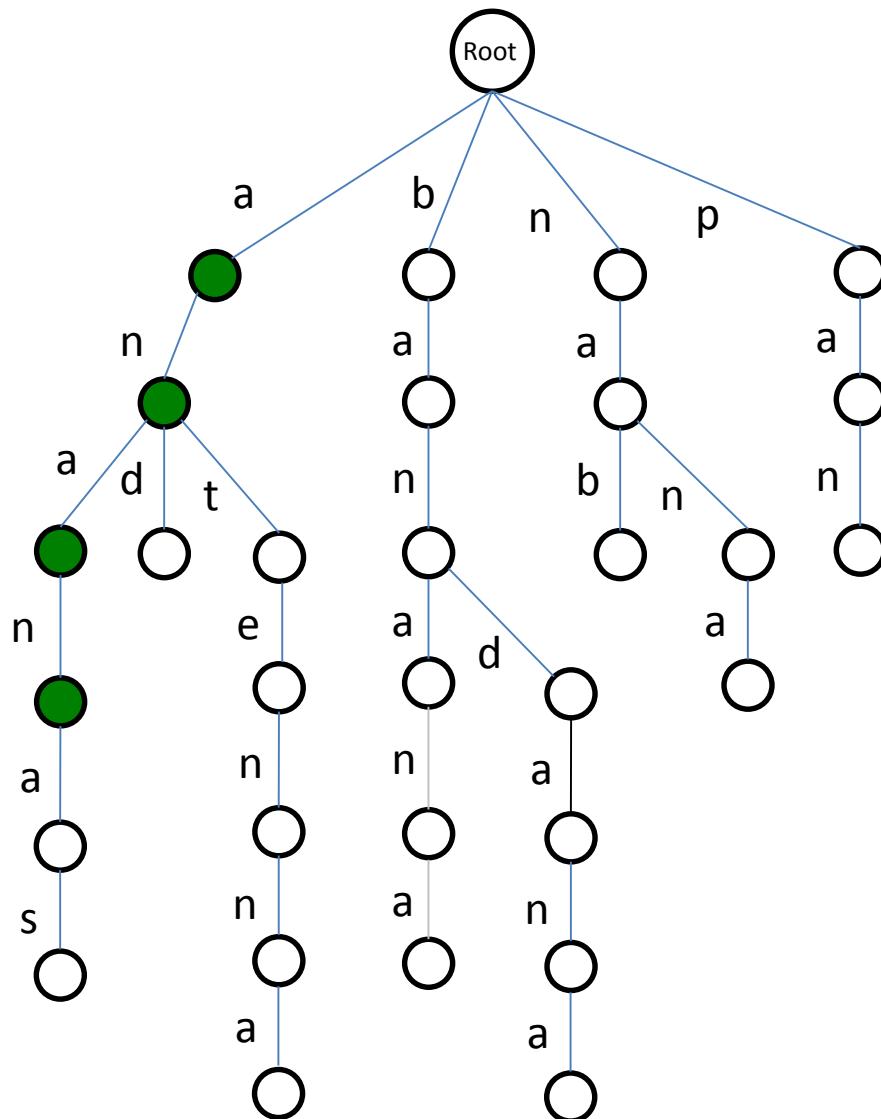
panamab **a**nanas



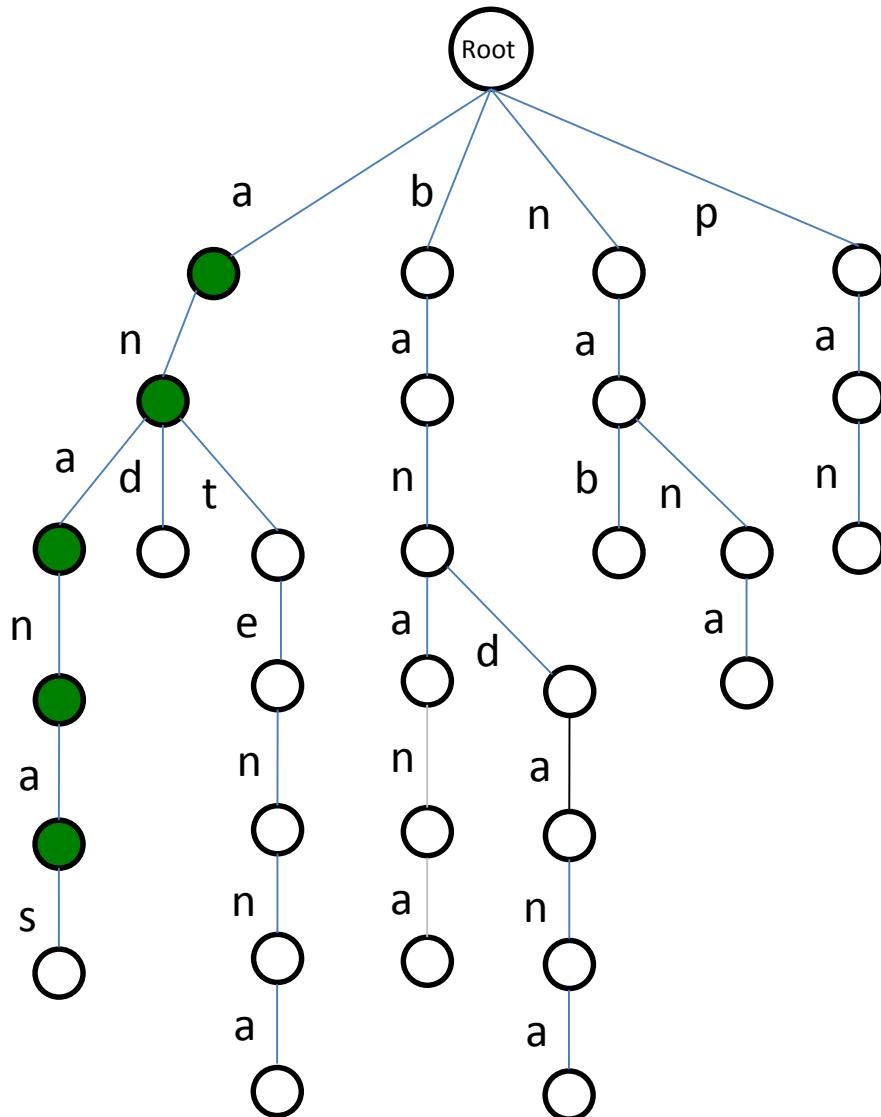
panamab **a**nanas



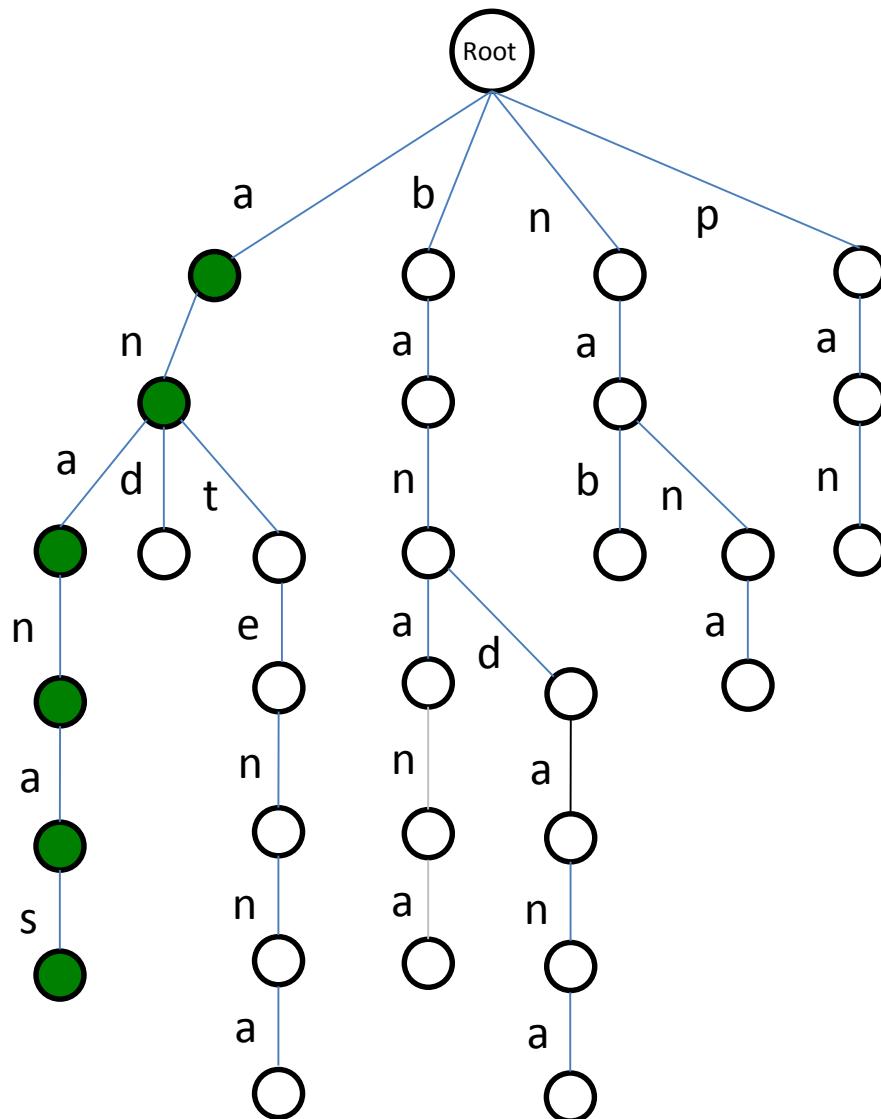
panama bananas



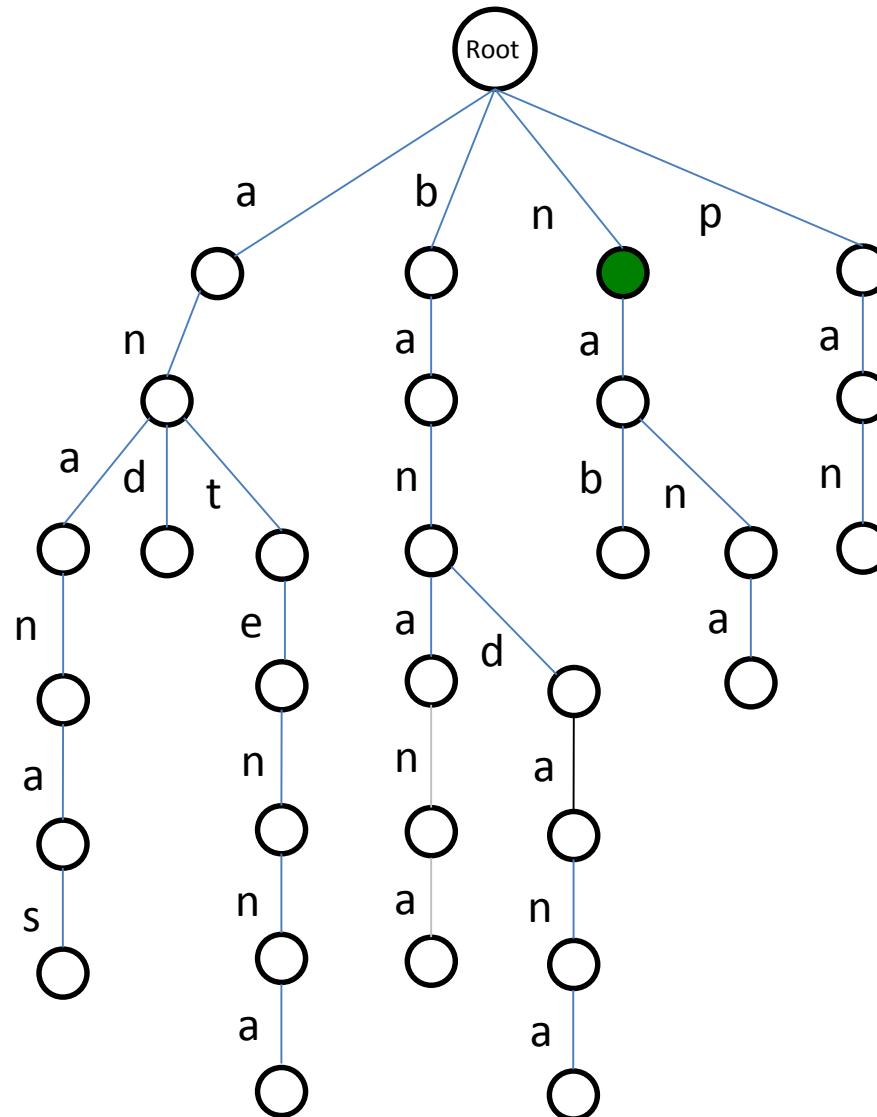
panamab **a**nanas



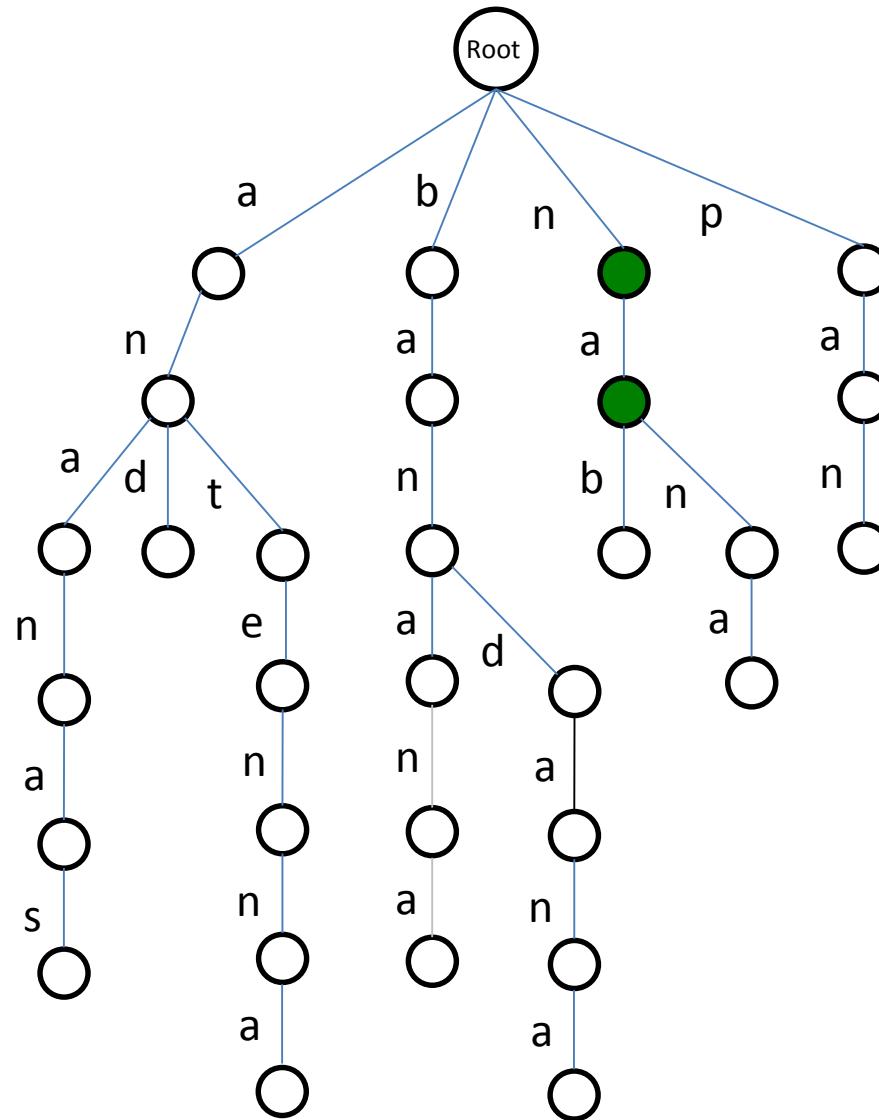
panama bananas



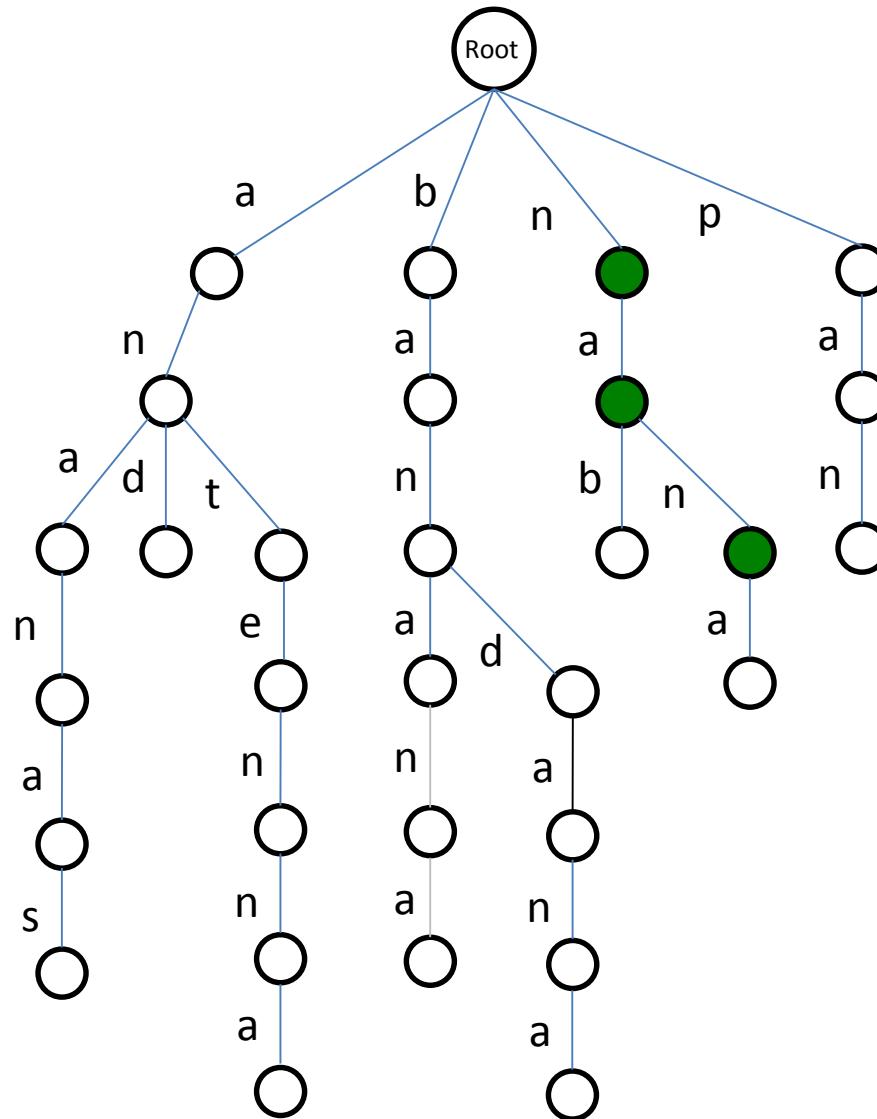
panamabananas



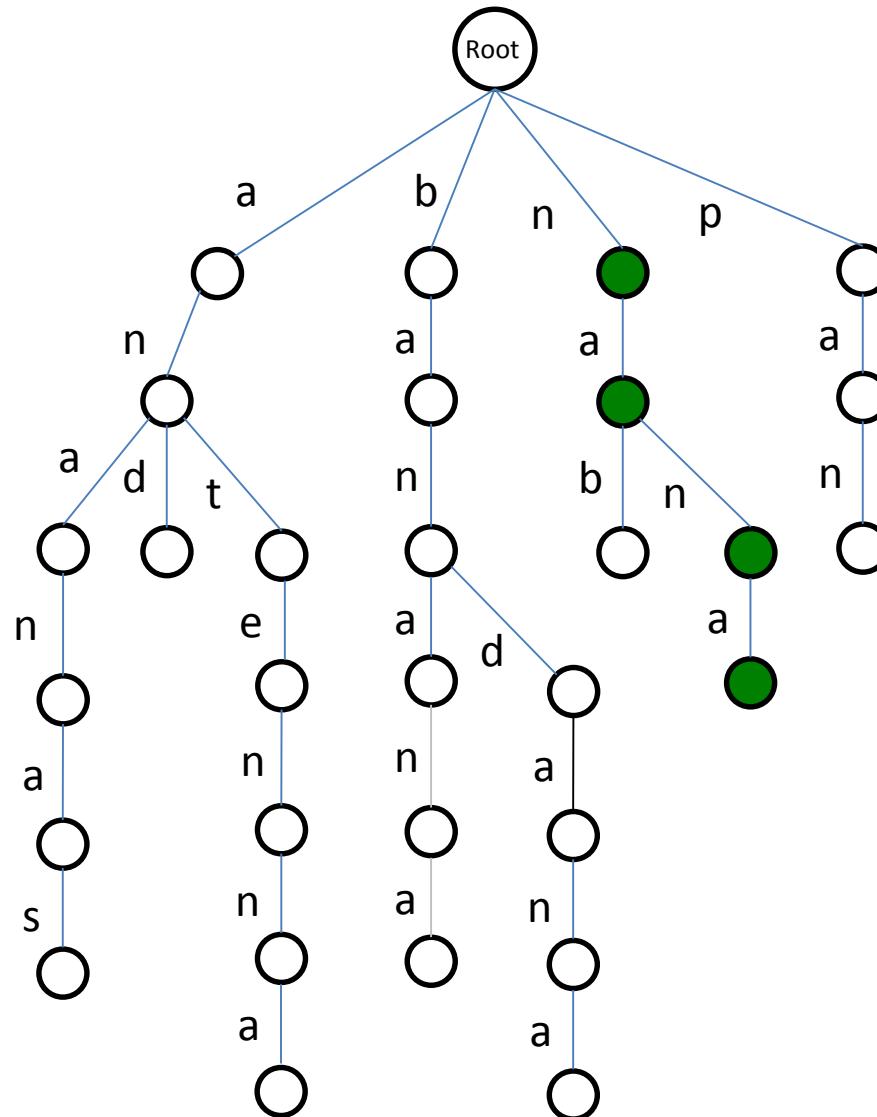
panamaba **n**anas



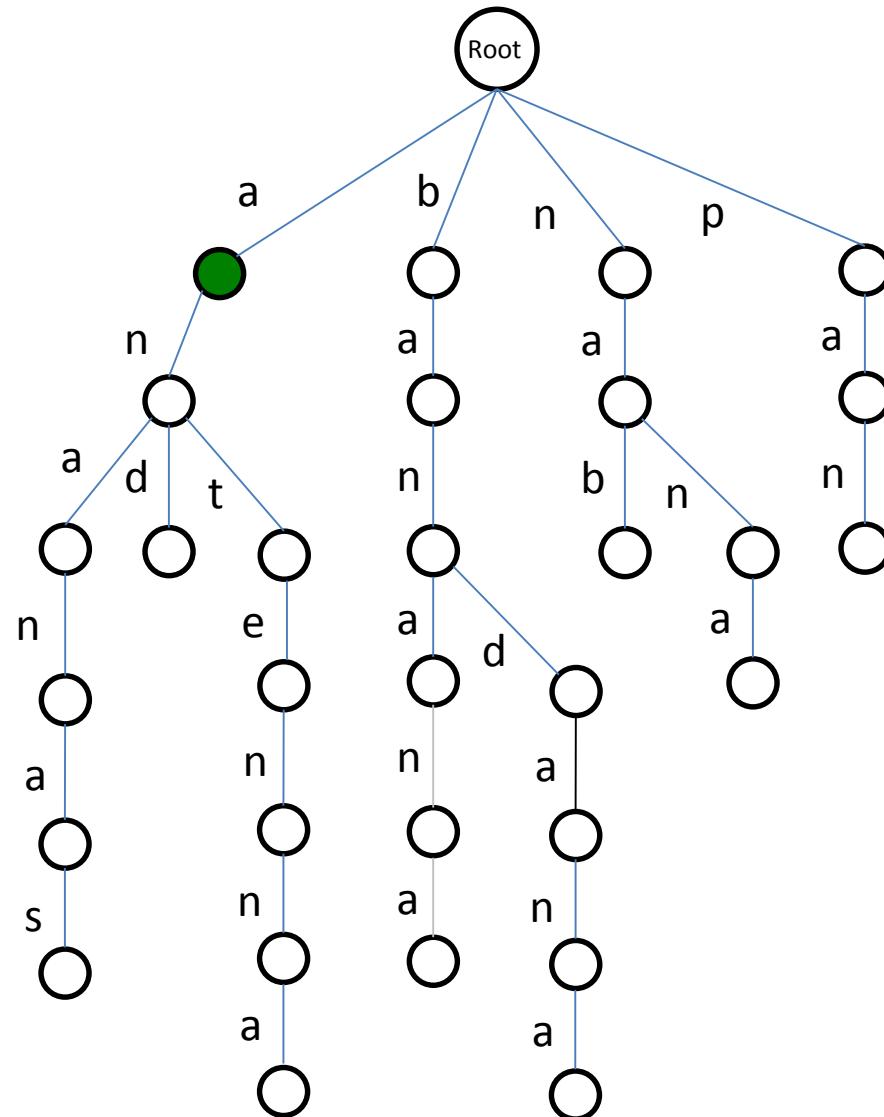
panamabananas



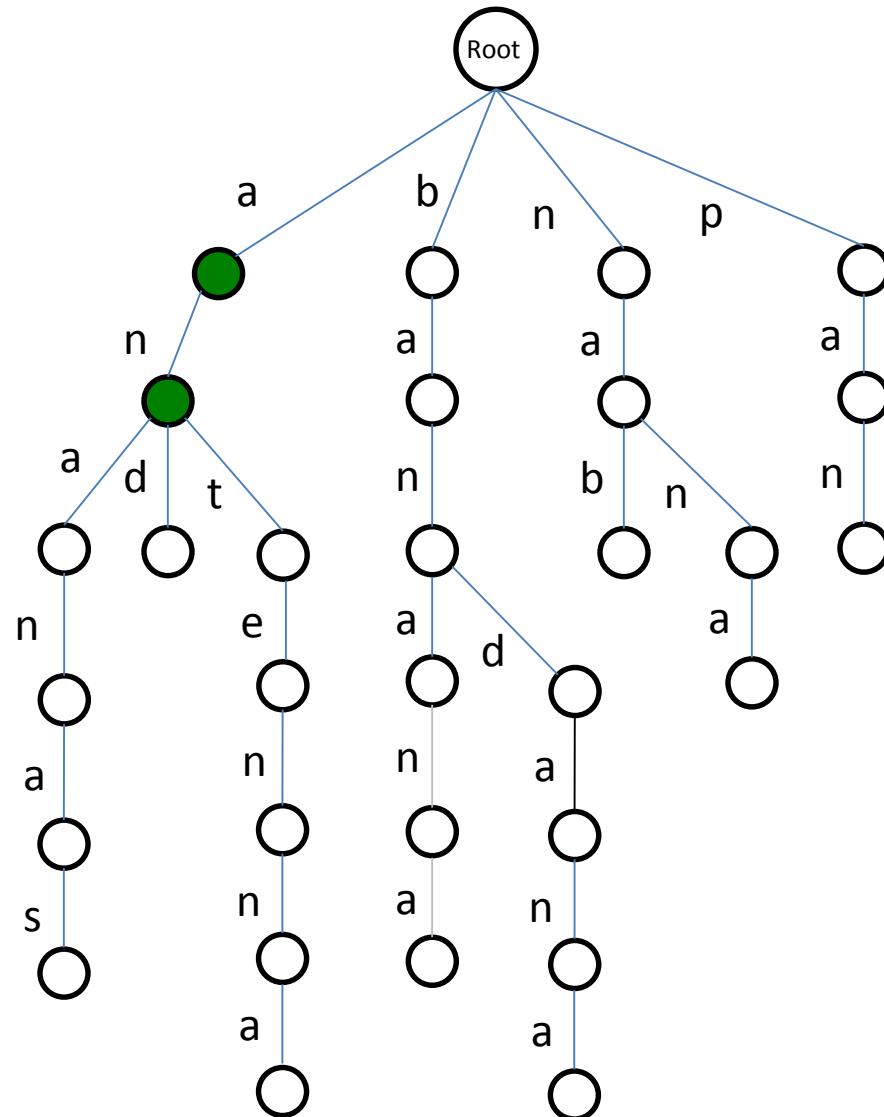
panamabananas



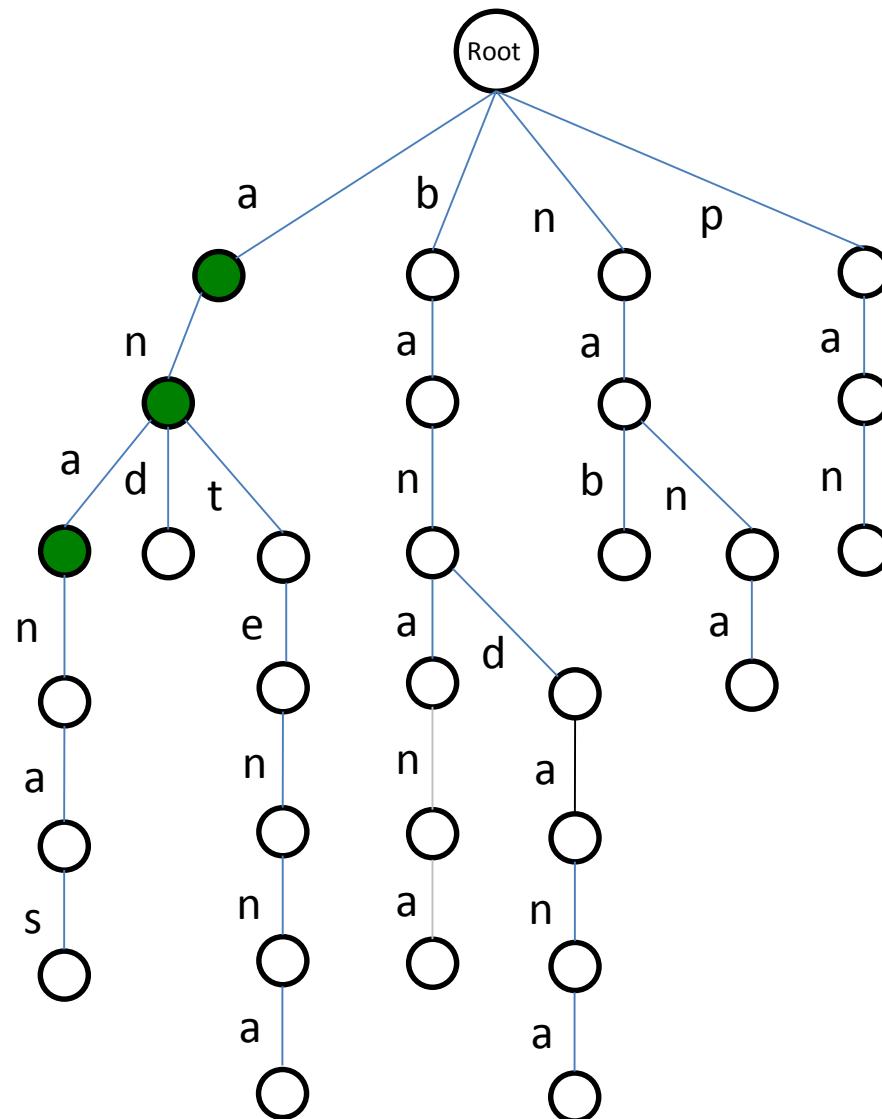
panamabananas



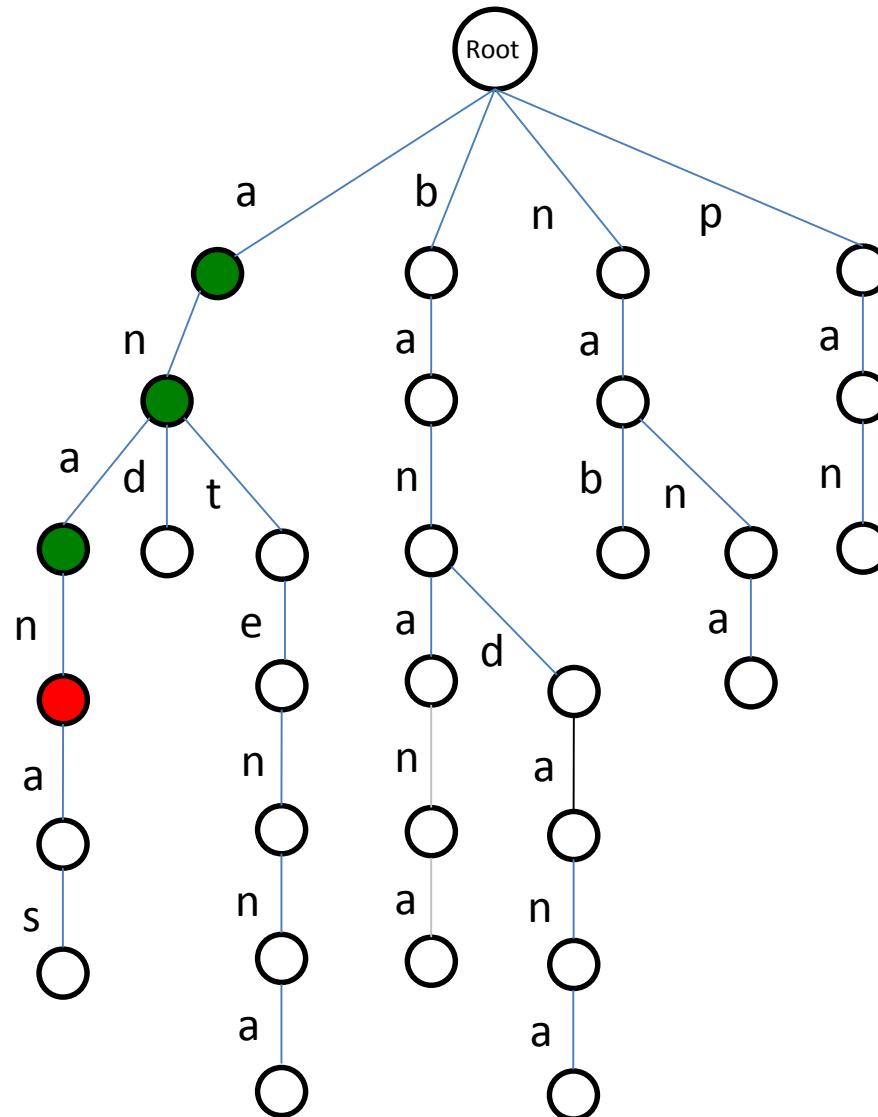
panamabananas



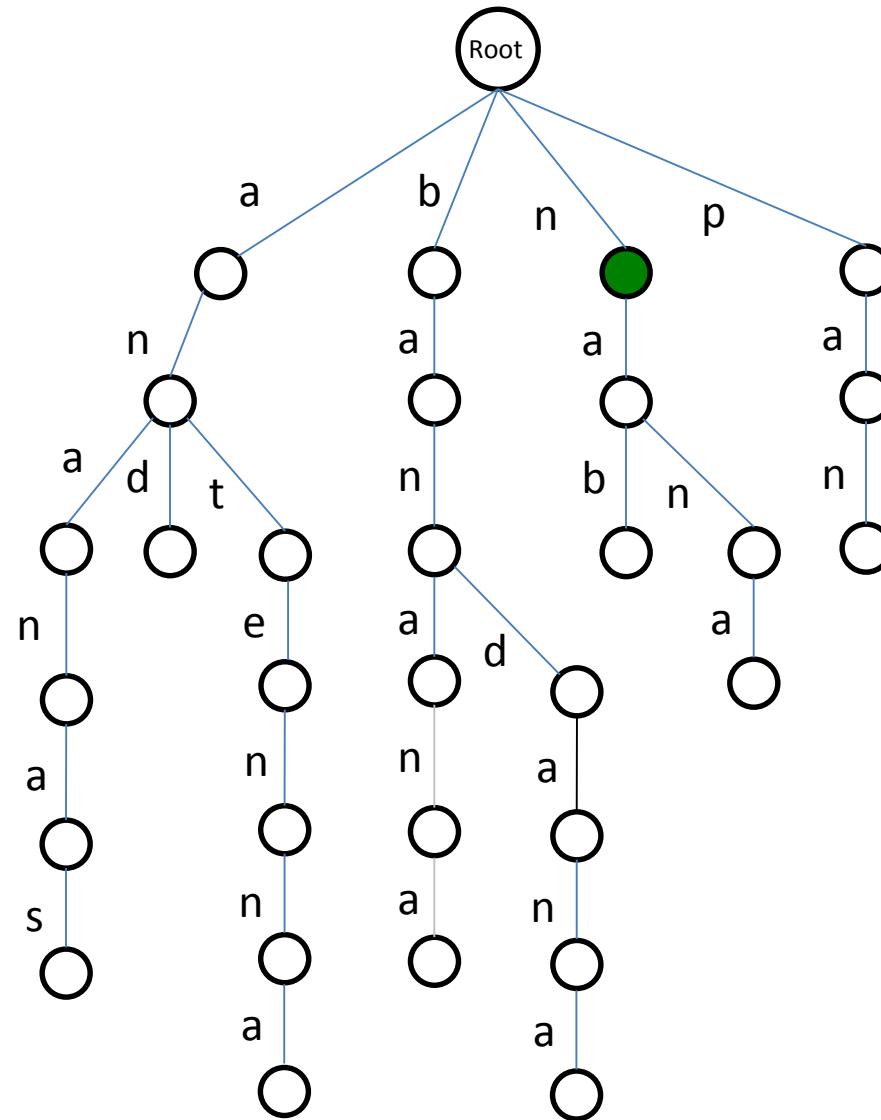
panamabananas



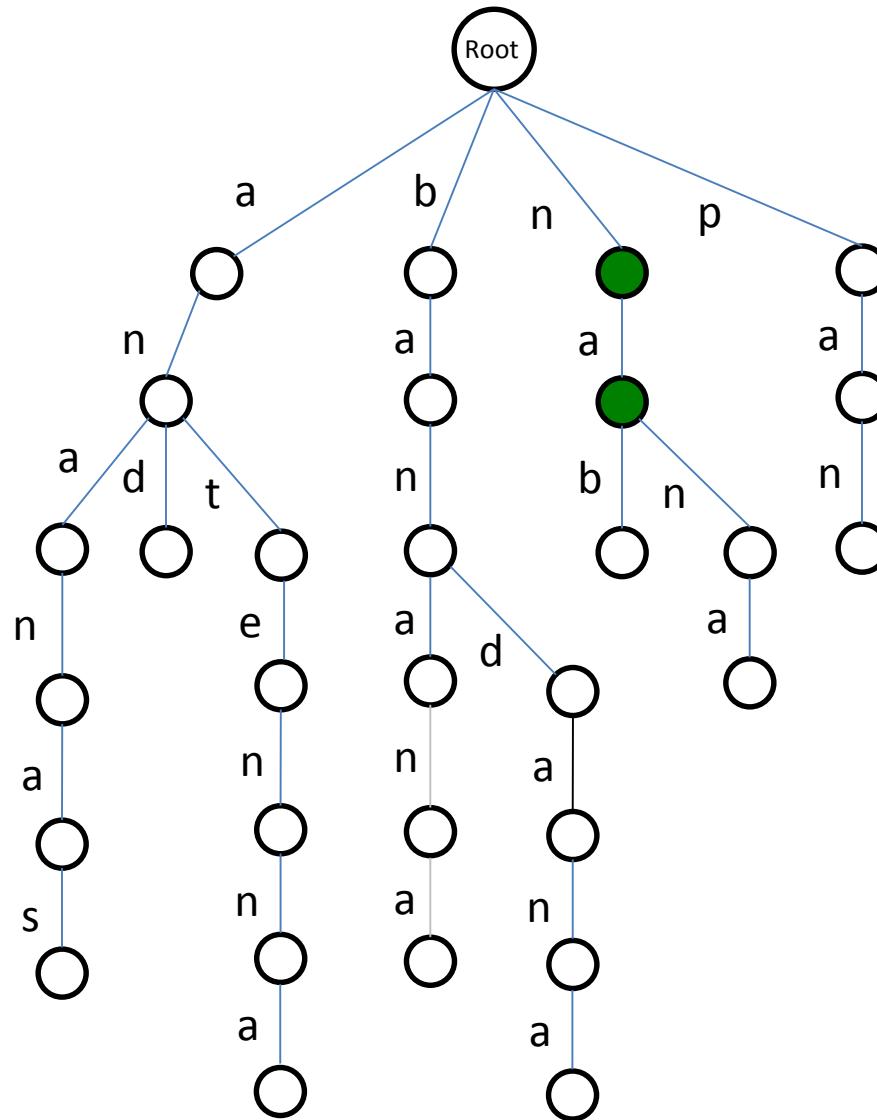
panamabananas



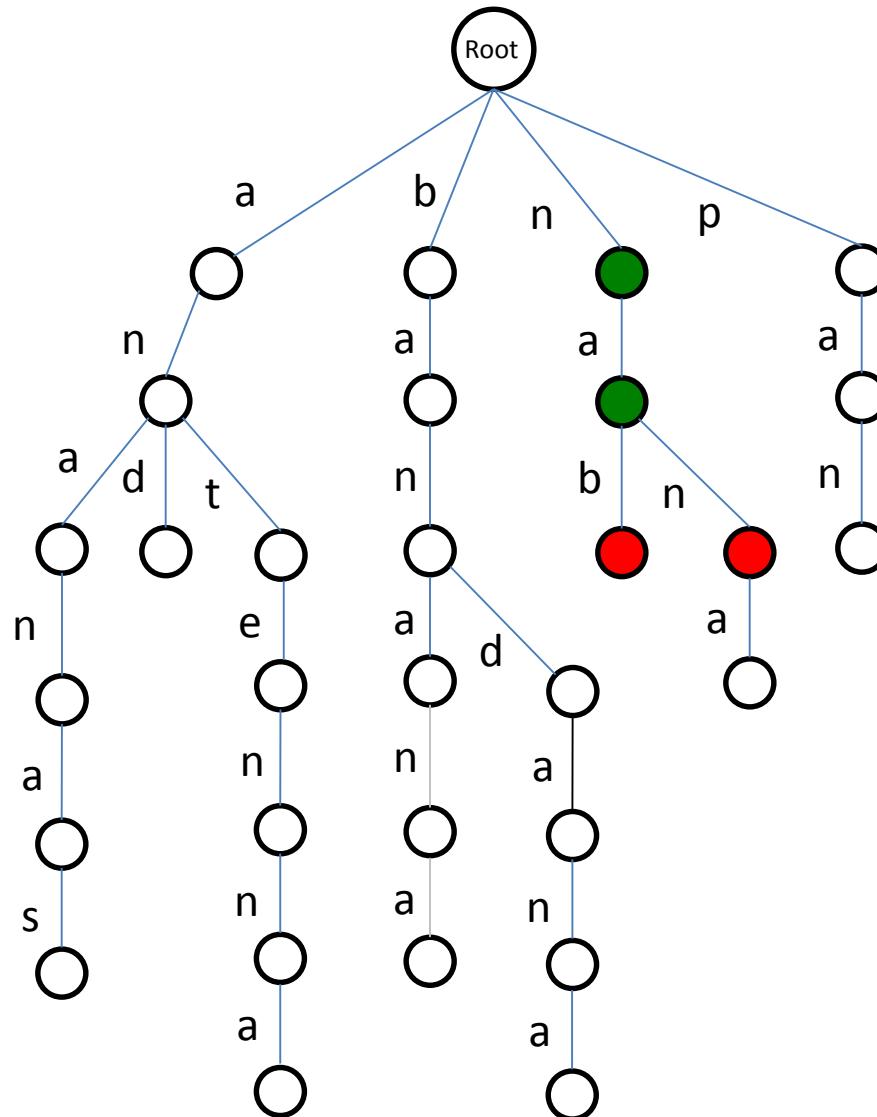
panamabananas



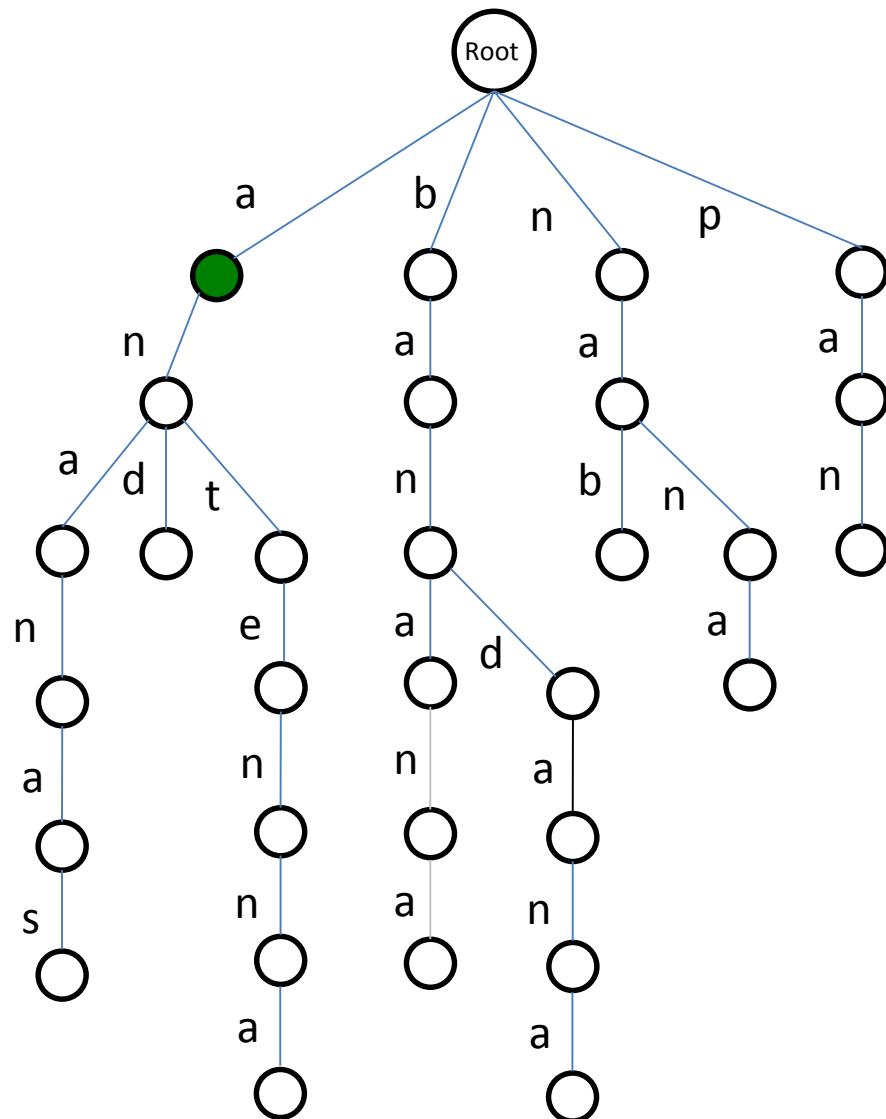
panamabananas



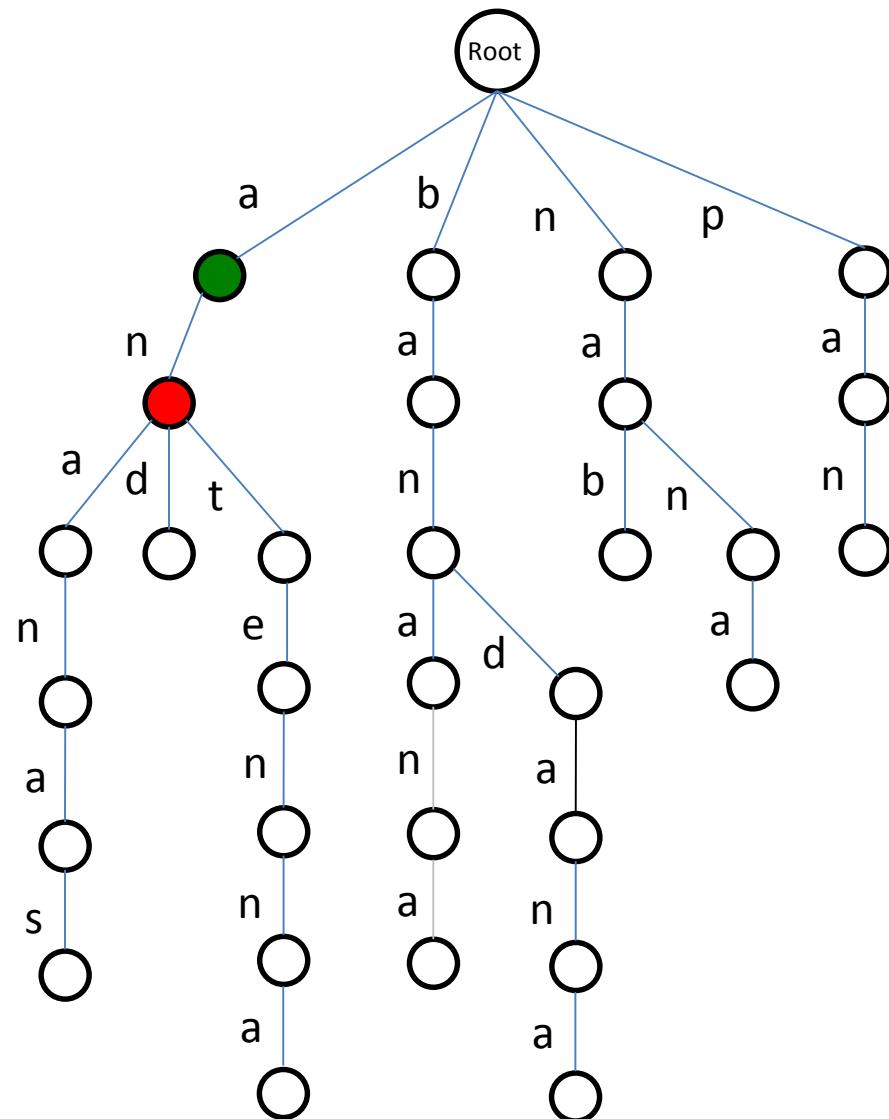
panamabananas



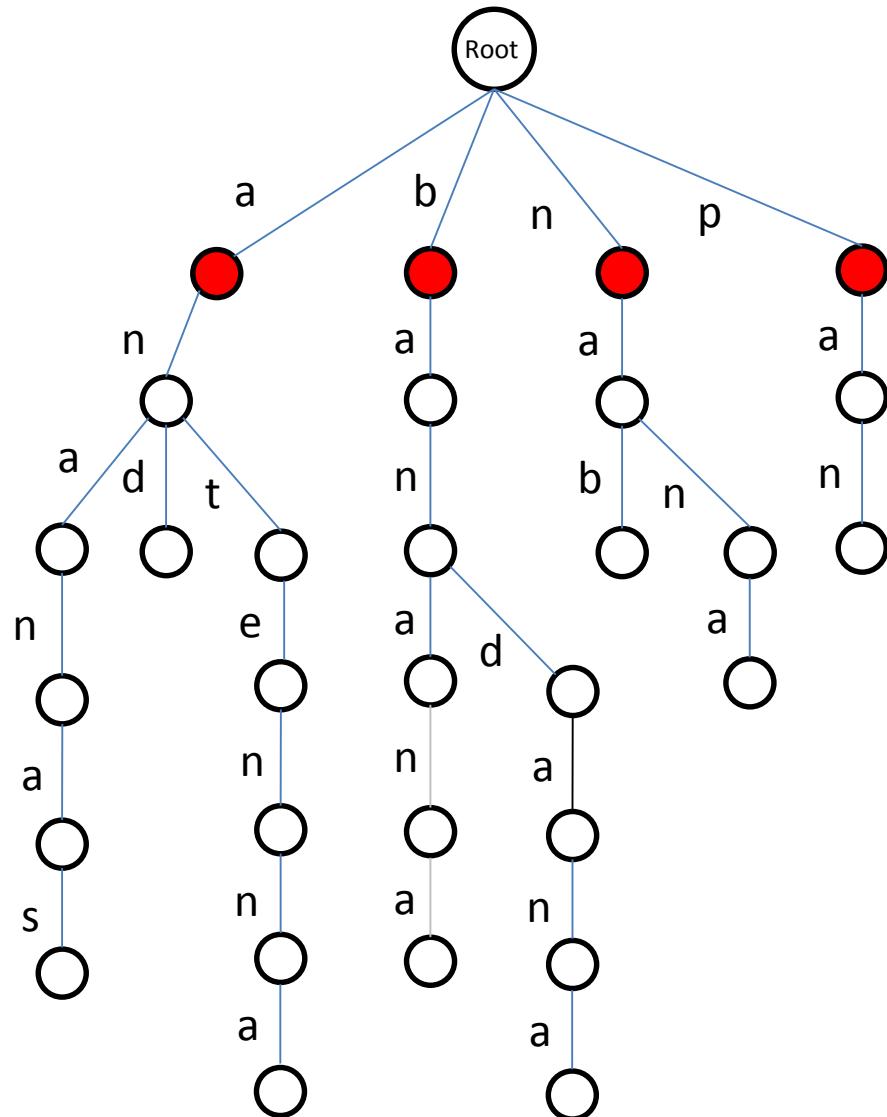
panamabananas



panamabananas



panamabananas



# Our Bus Is Fast!

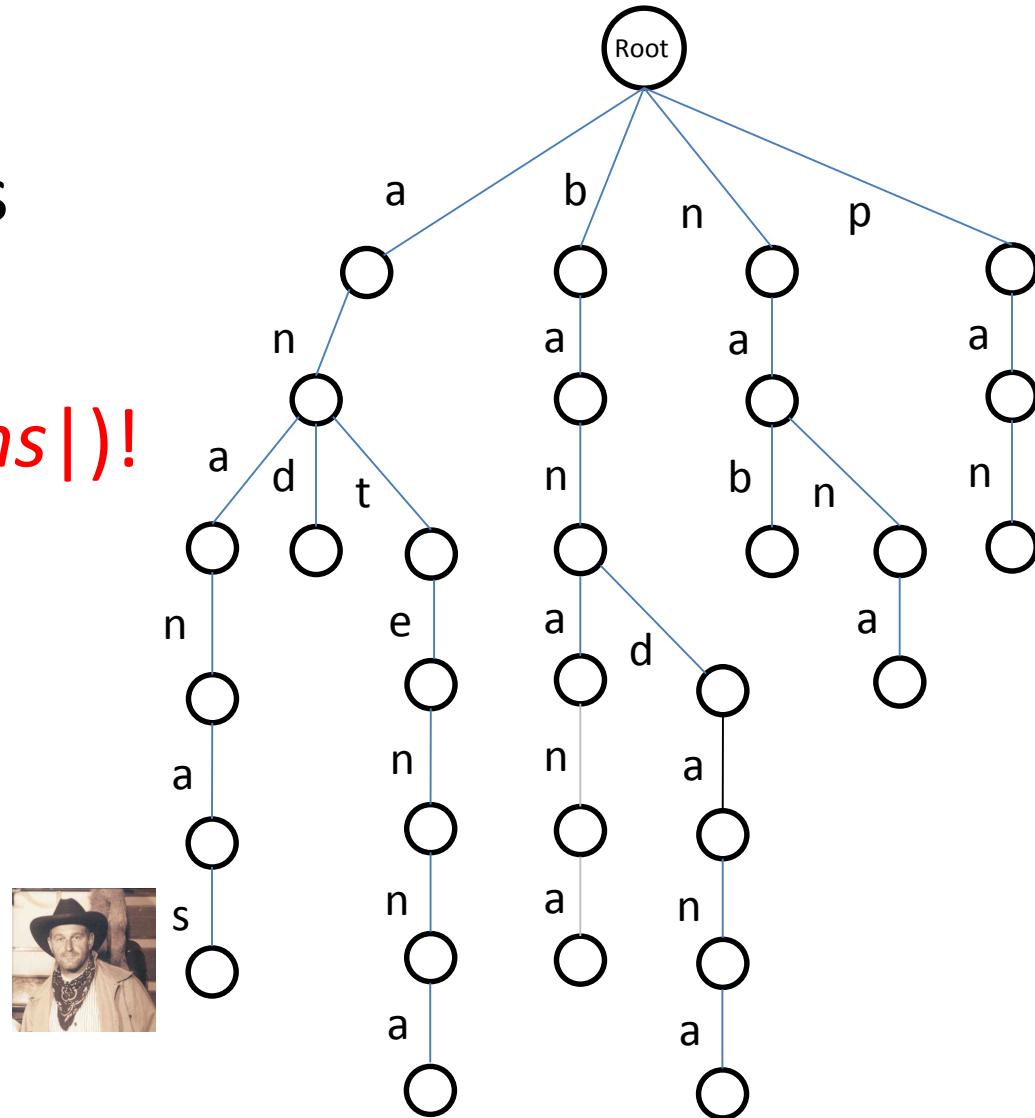
- Runtime of brute force approach:
  - $O(|Text| \cdot |Patterns|)$
- Runtime of **TrieMatching**:
  - $O(|Text| * |LongestPattern|)$



Trie construction takes  $O(|Patterns|)$  time

# Memory Footprint of TrieMatching

- Our trie has 30 edges
- # edges =  $O(|Patterns|)$ !
- For human genome:  
 $|Patterns| \approx 10^{12}$



# Outline

- From Genome Sequencing to Pattern Matching
- Brute Force Approach to Pattern Matching
- Herding Patterns into Trie
- **Herd~~ing~~ Text into Suffix Trie**
- From Suffix Tries to Suffix Trees

# New Idea: Packing *Text* onto a Bus

- Generate all suffixes of *Text*
- Form a trie out of these suffixes (**suffix trie**)
- For each *Pattern*, check if it can be spelled out from the root downward in the suffix trie

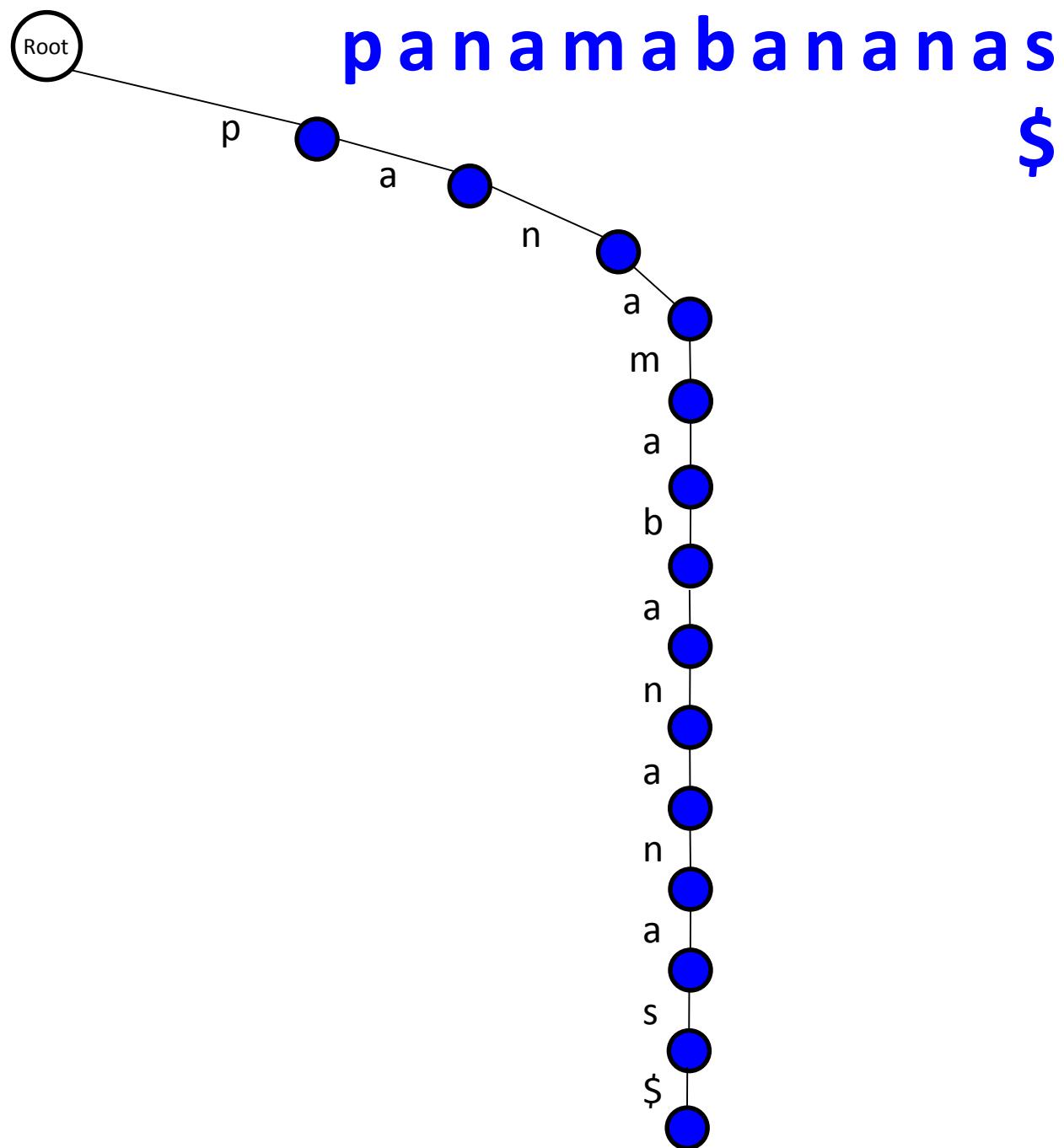


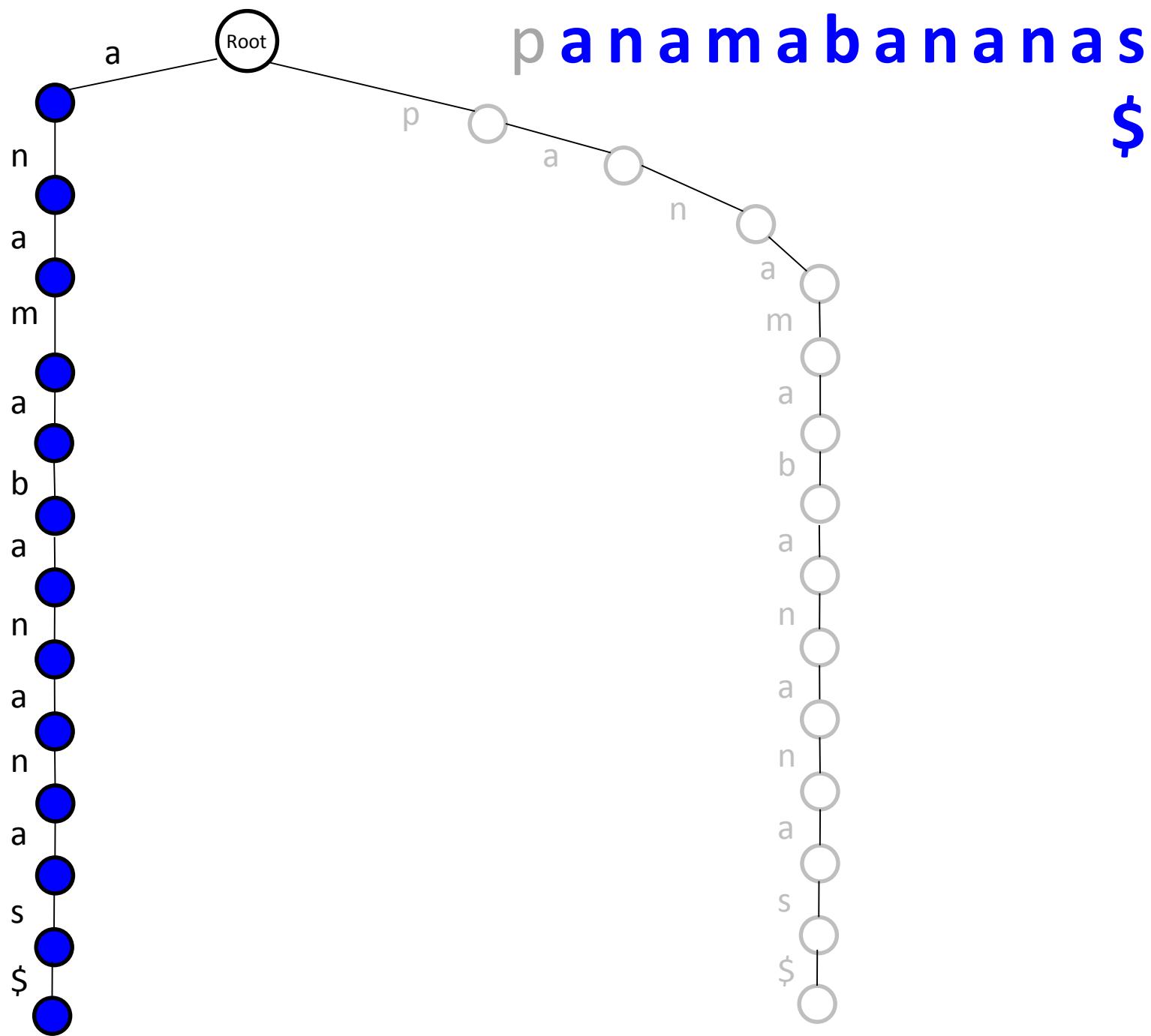
panama bananas

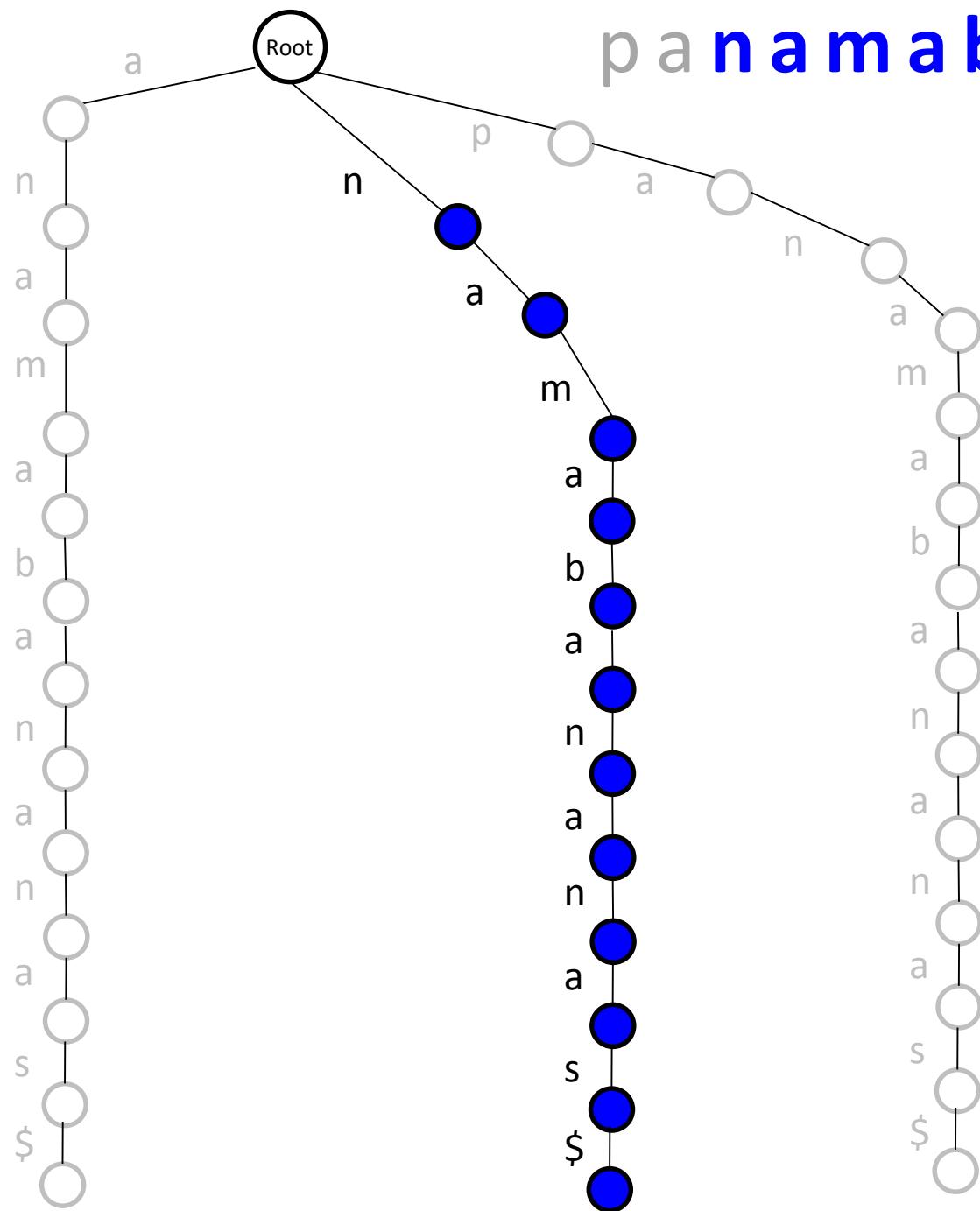


p a n a m a b a n a n a s \$

Adding “\$” sign in the end (we’ll explain later why)

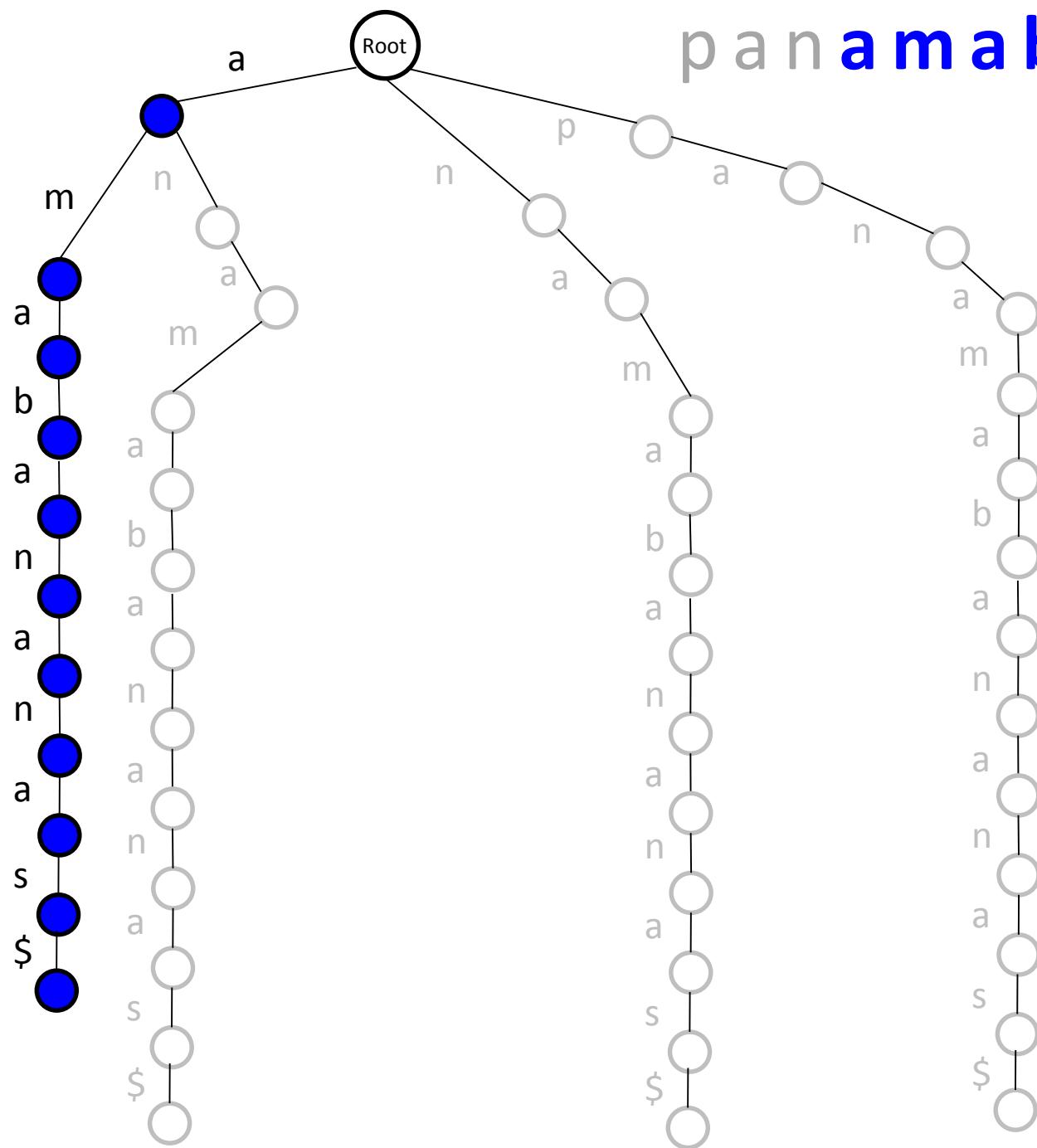






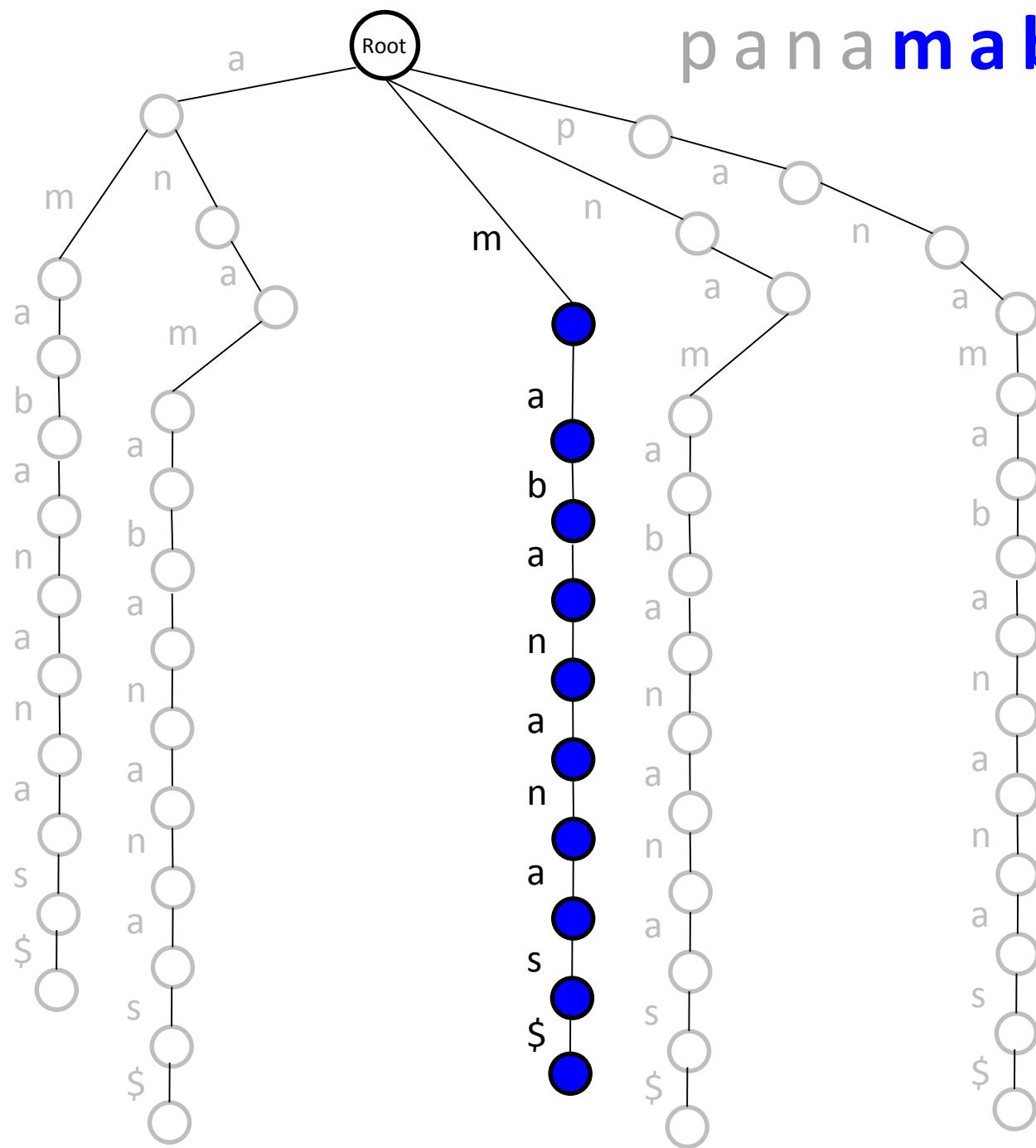
p a n a m a b a n a n a s

\$



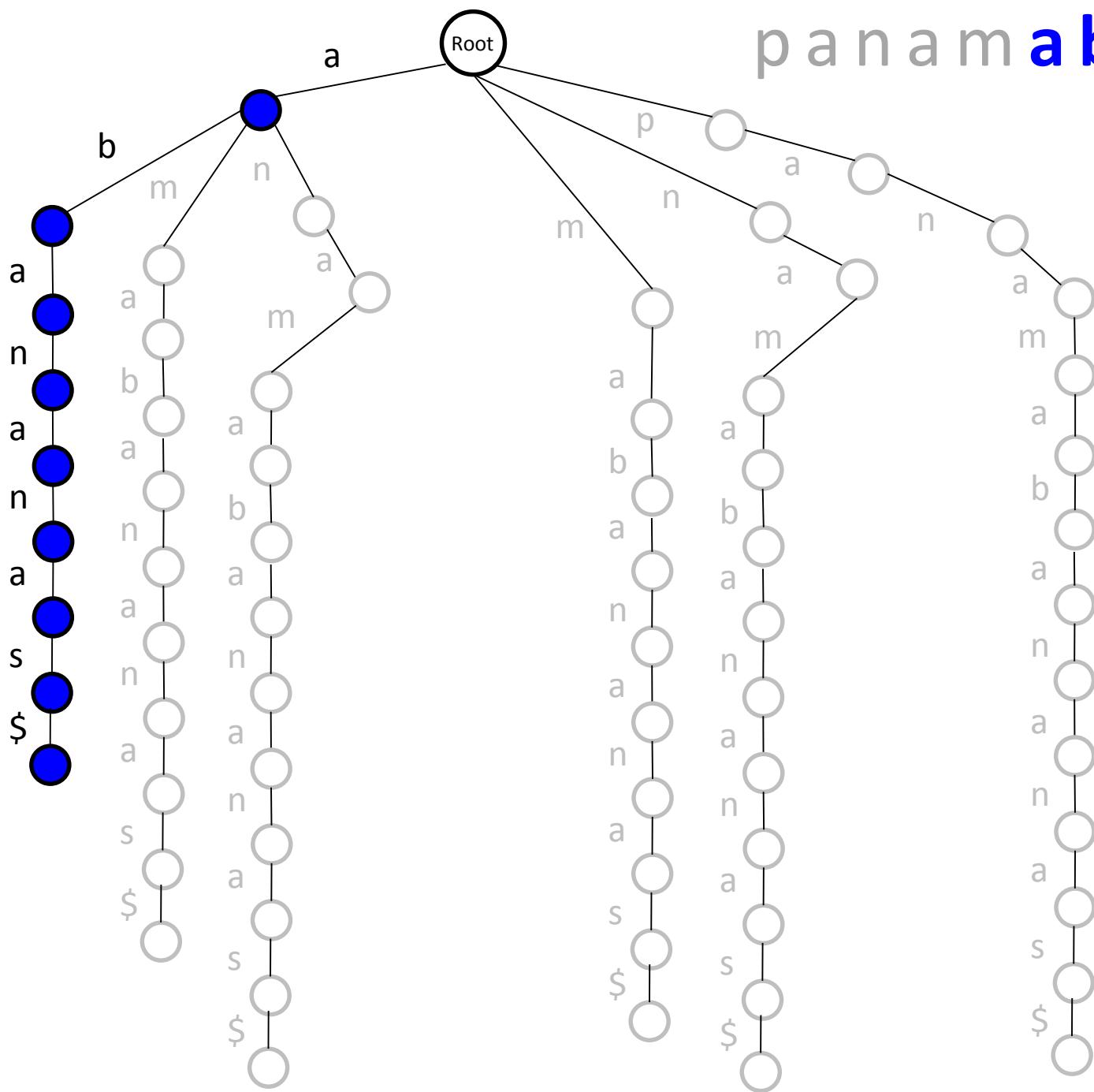
panamabananas

\$



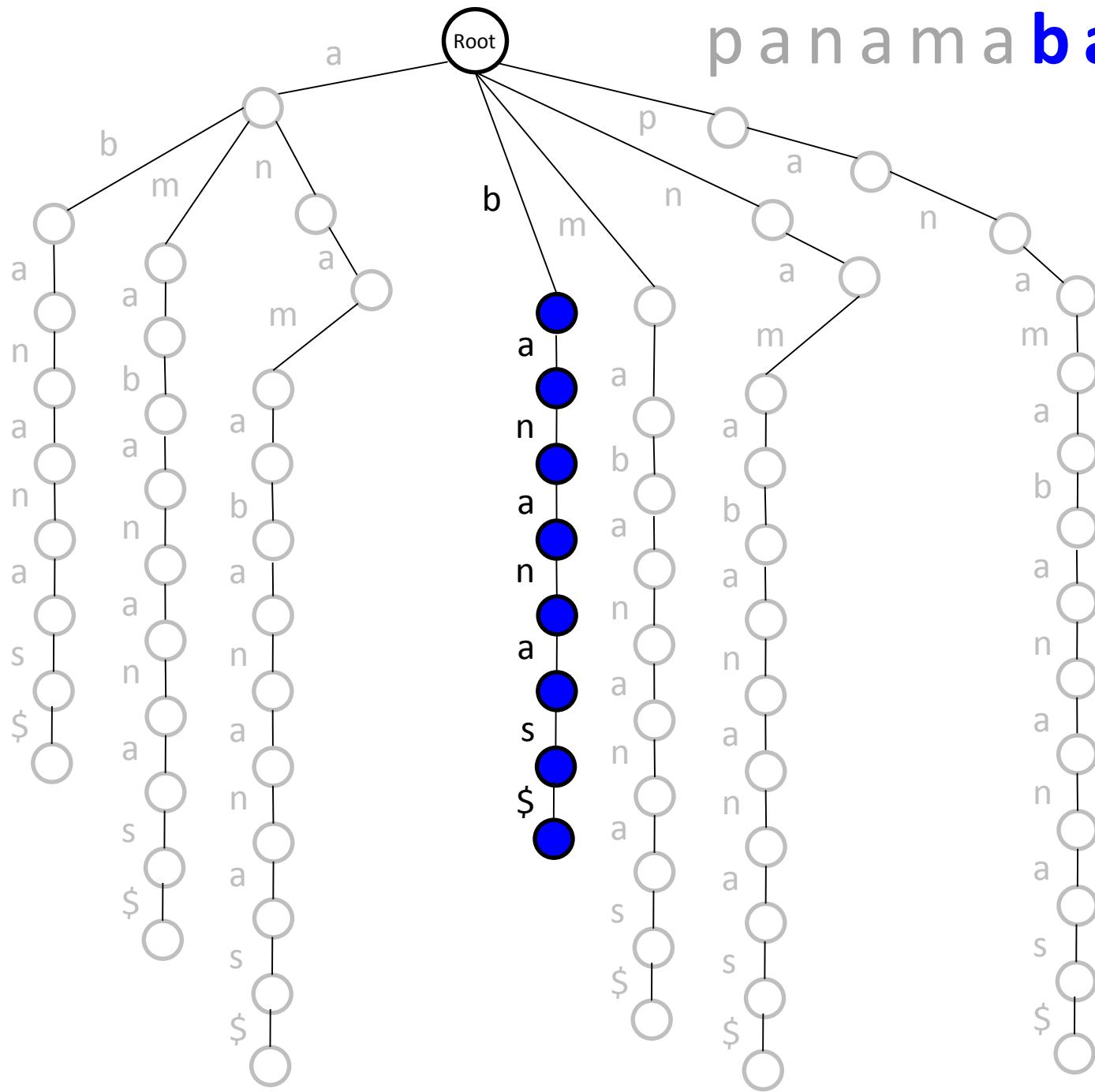
p a n m a b a n a n a s

\$

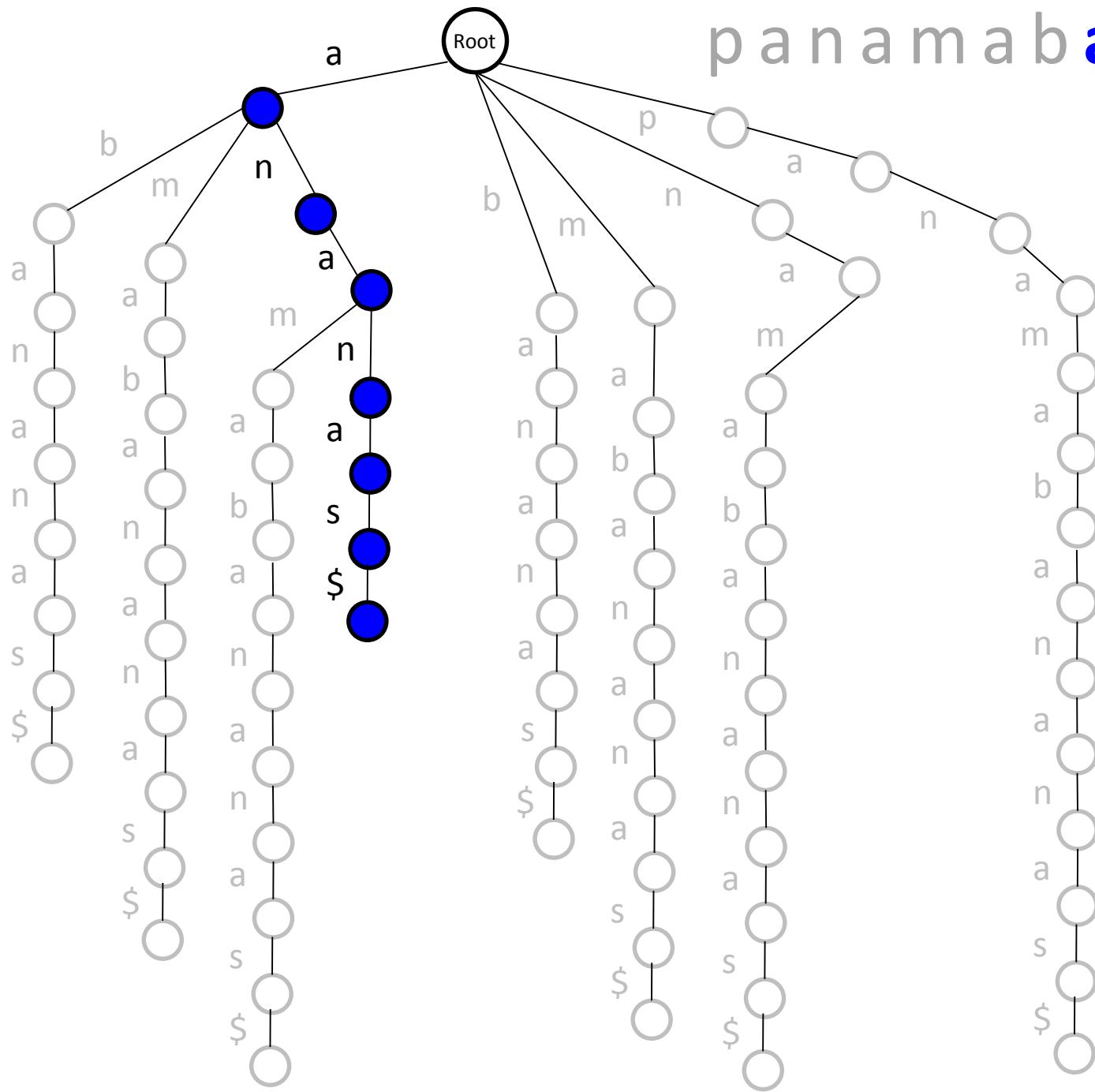


panam**a**bananas

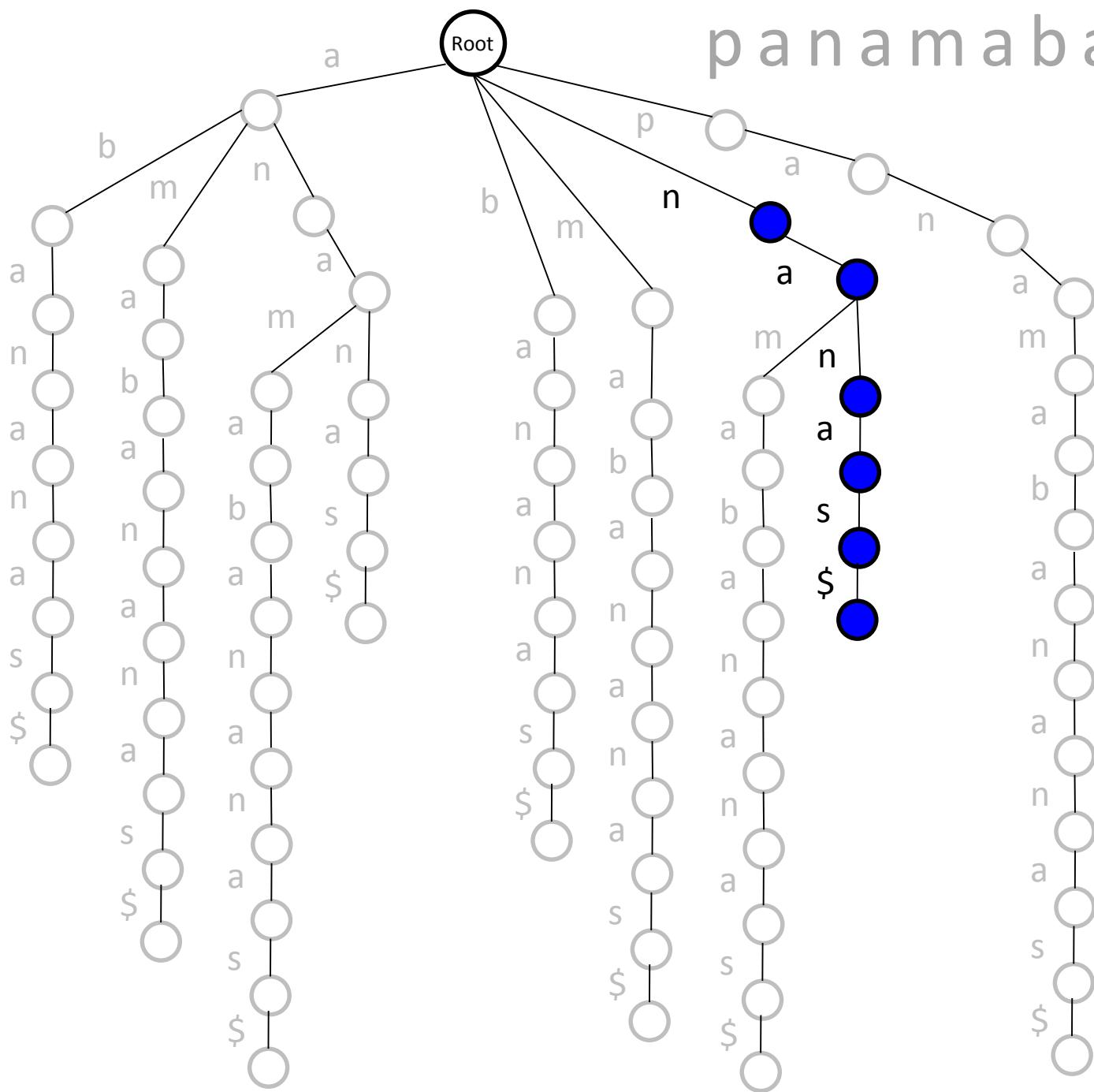
\$



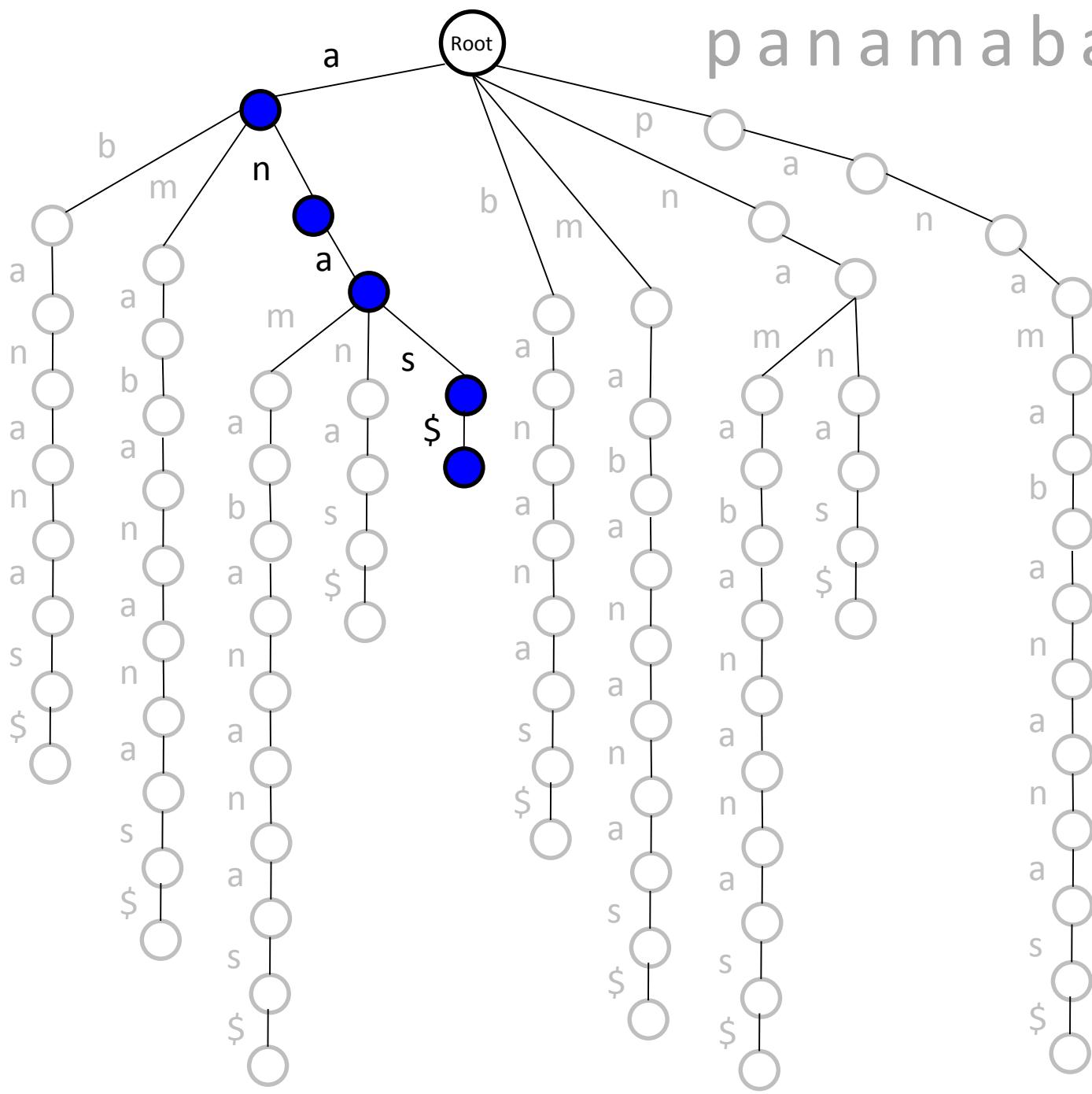
panama bananas\$



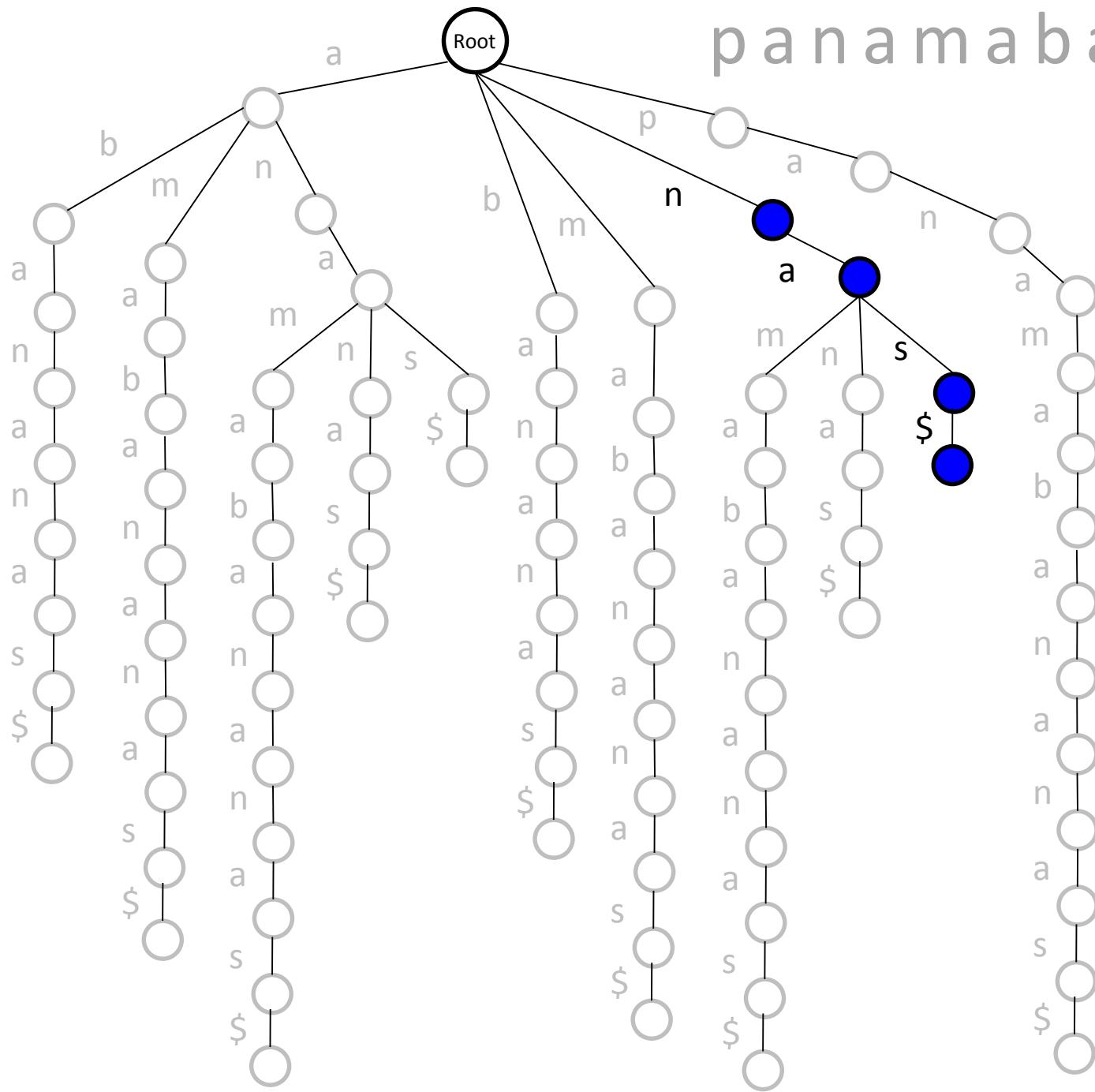
panamabananas\$



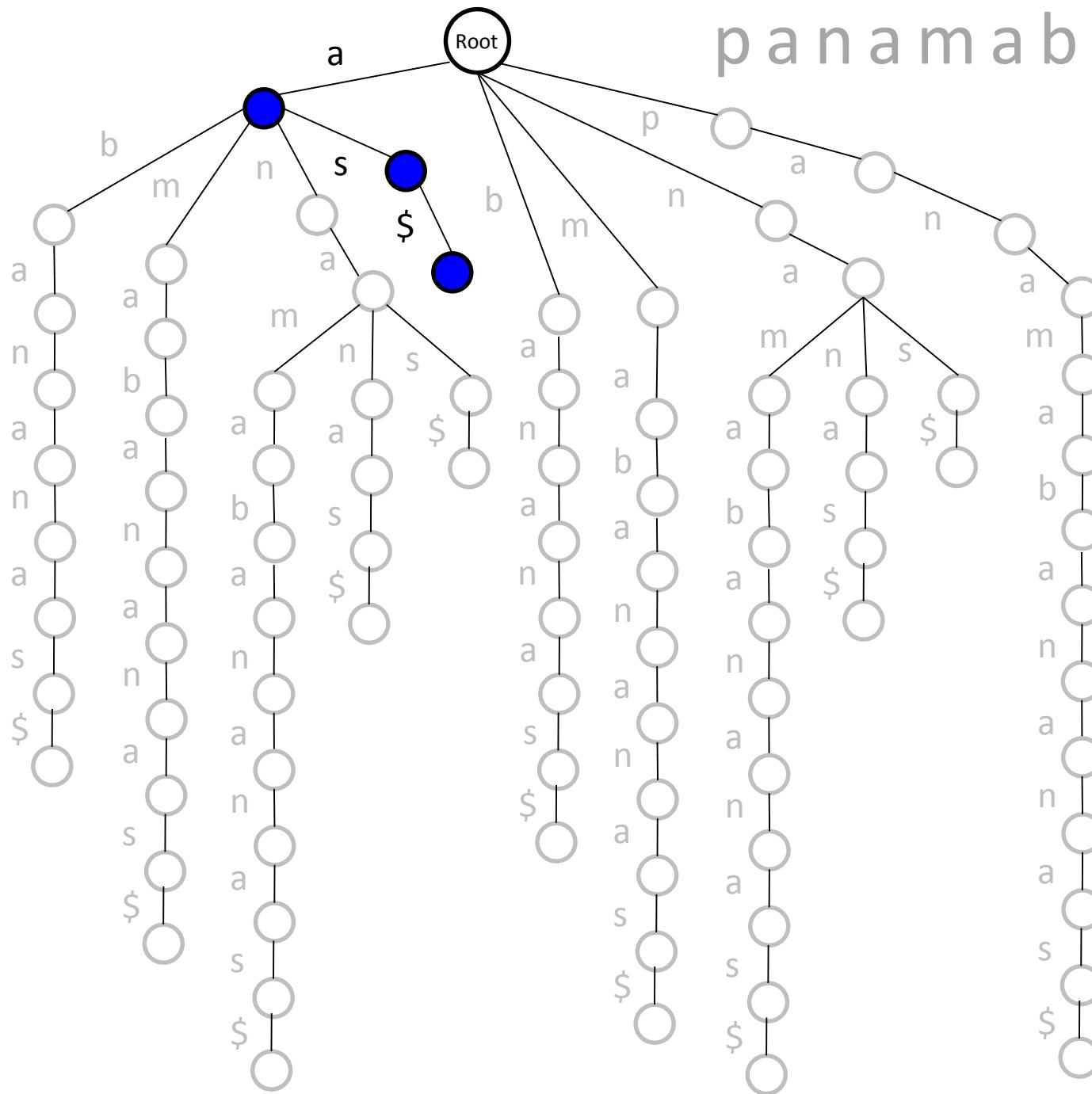
panamabananas\$



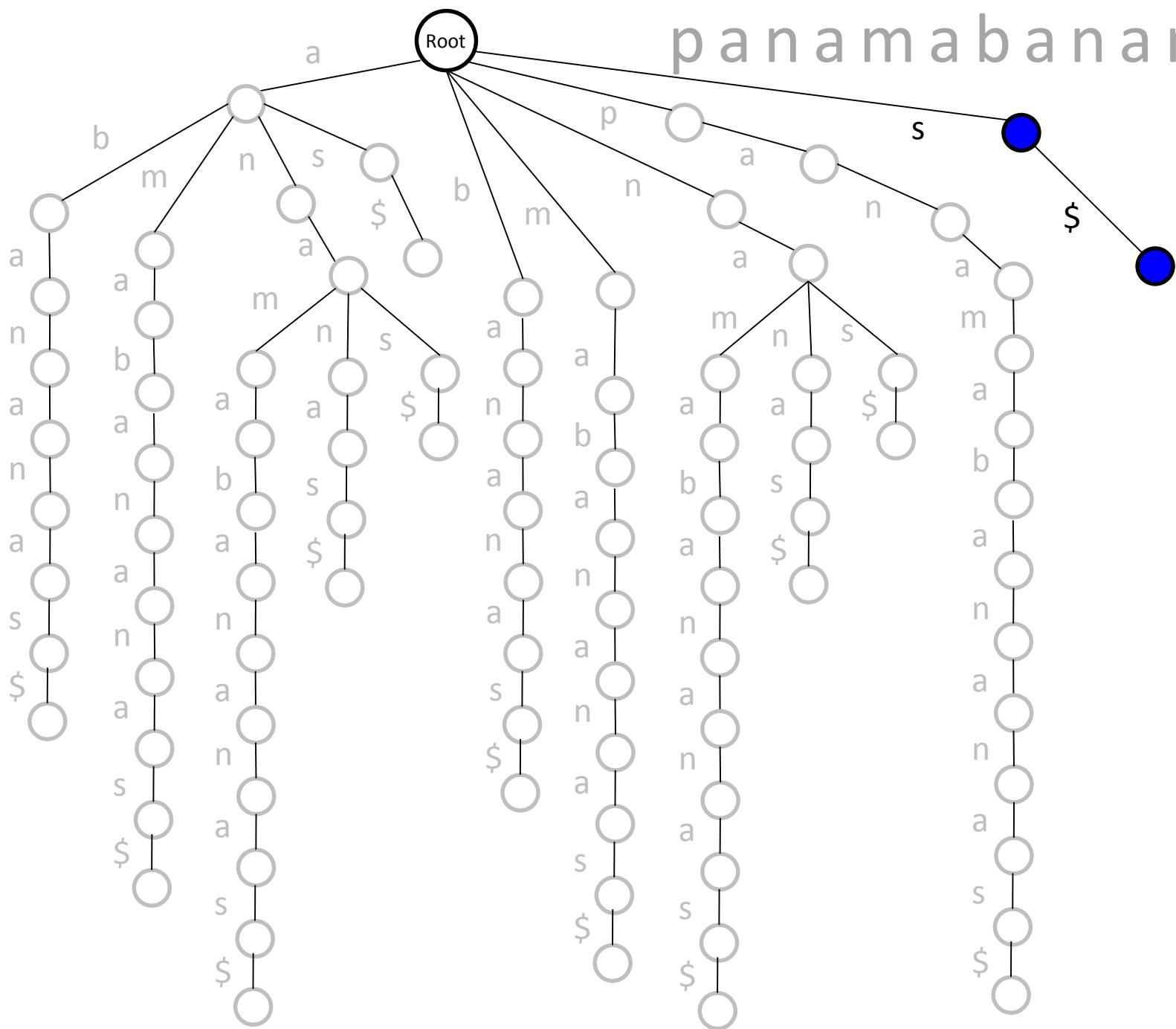
panamabananas\$



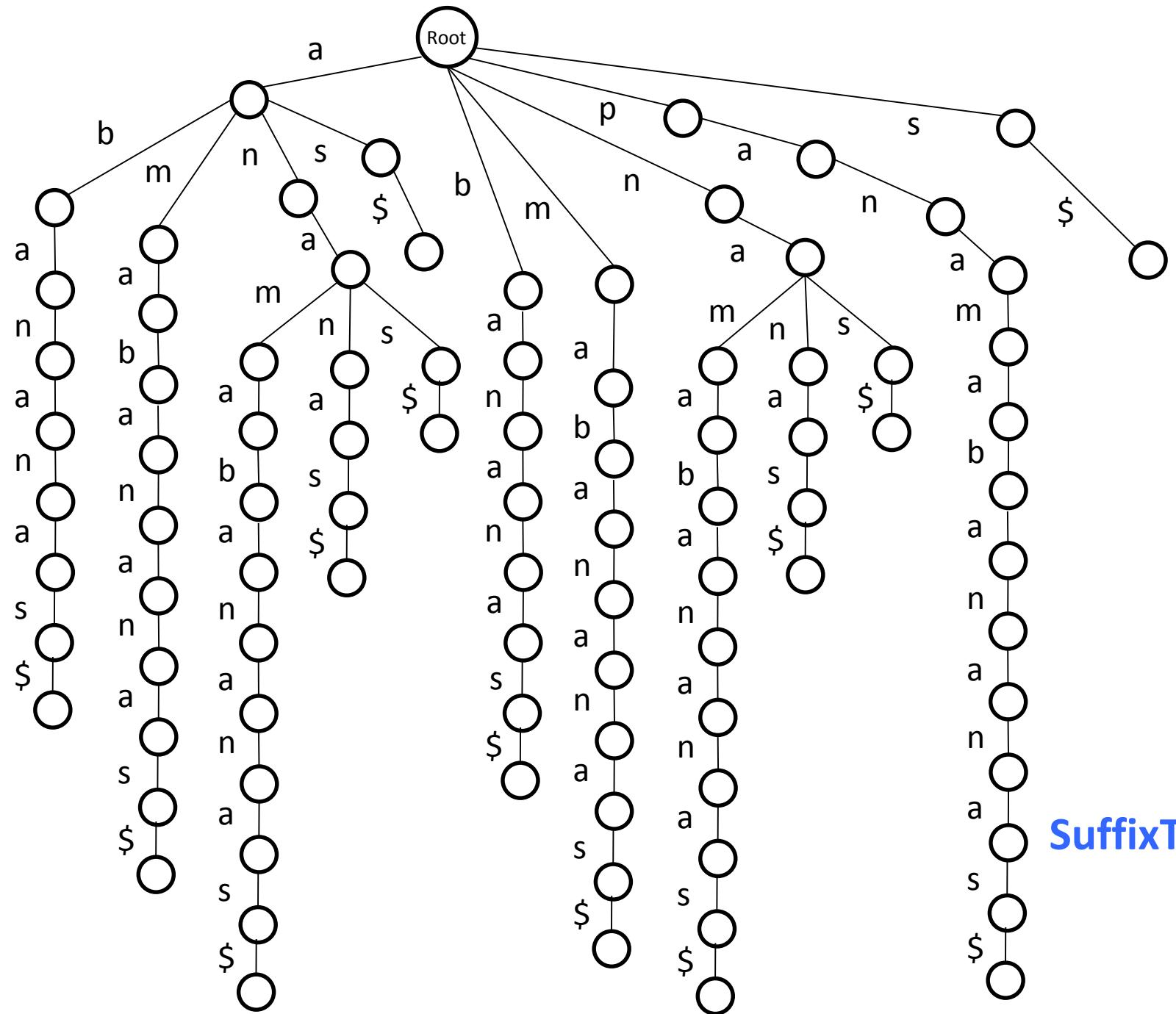
panamabananas\$



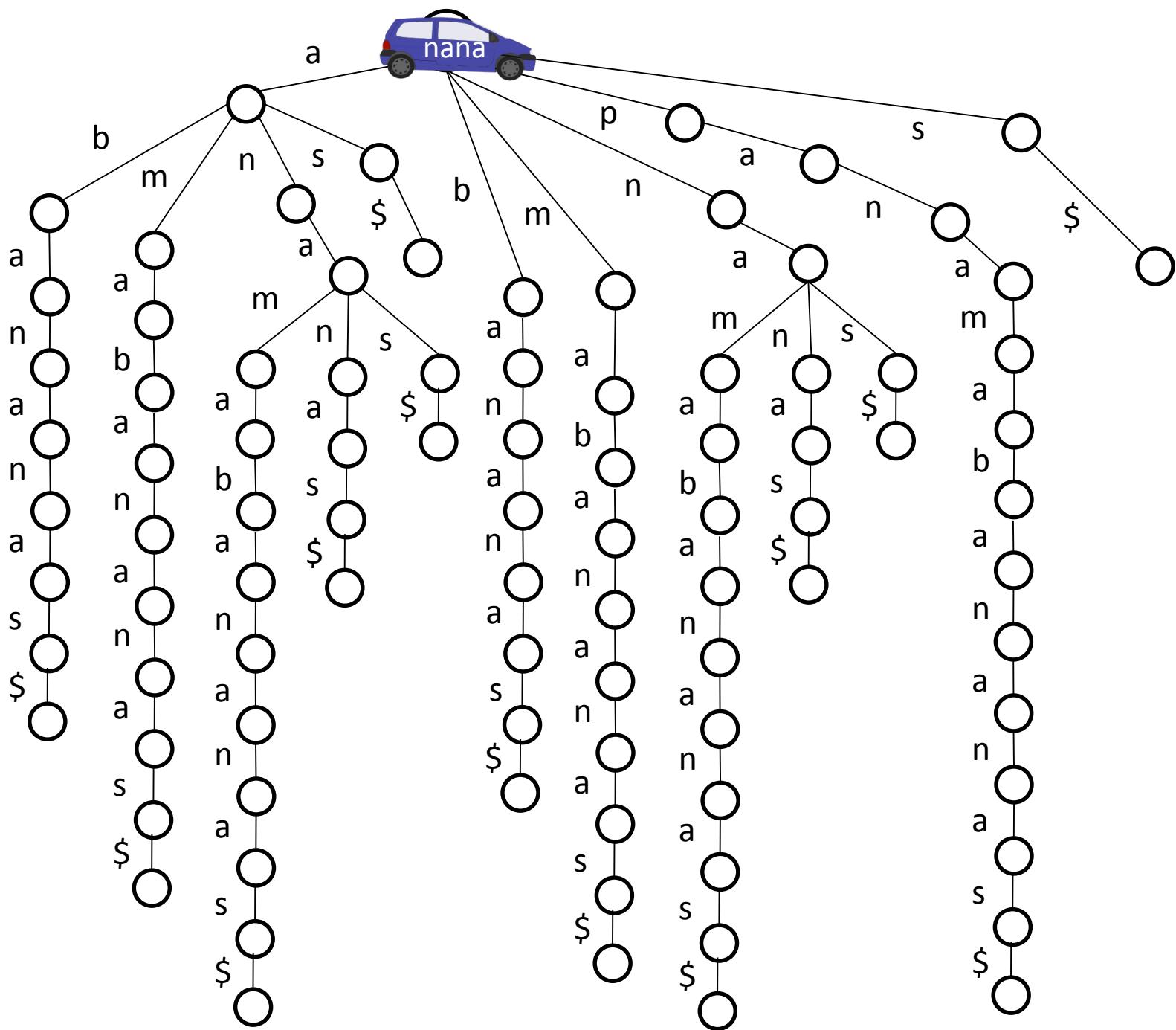
# panamabananas\$

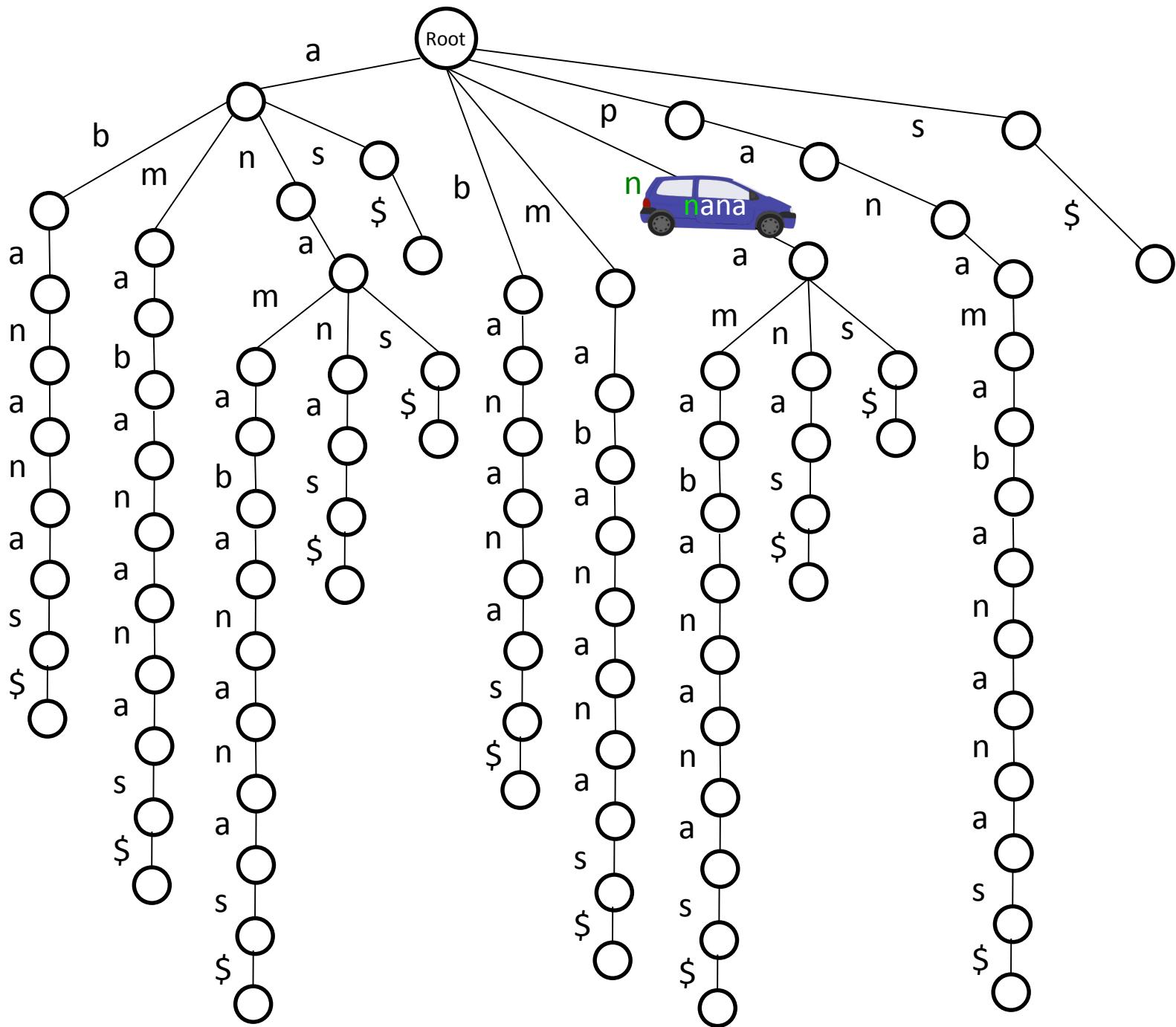


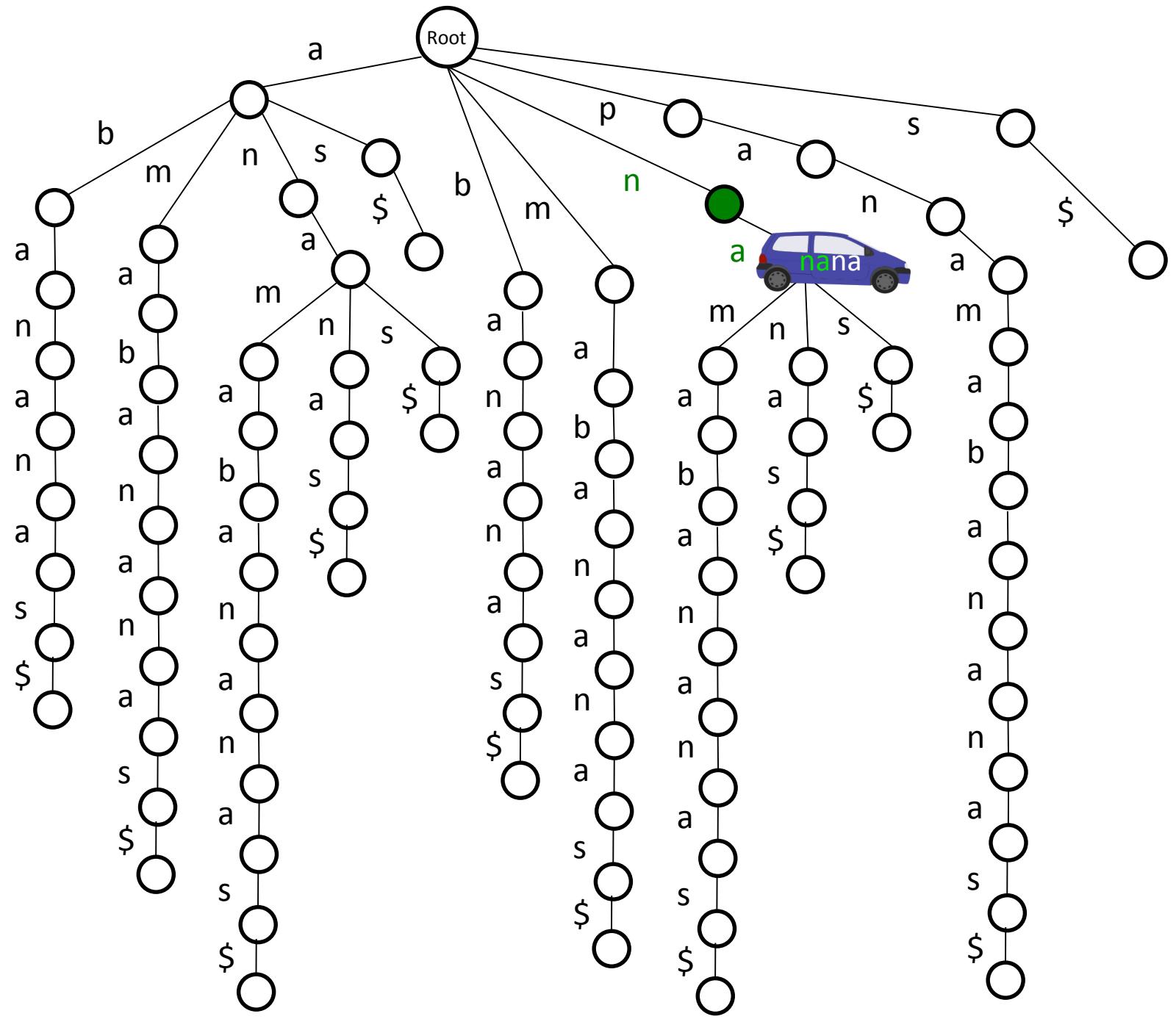
panamabanas\$

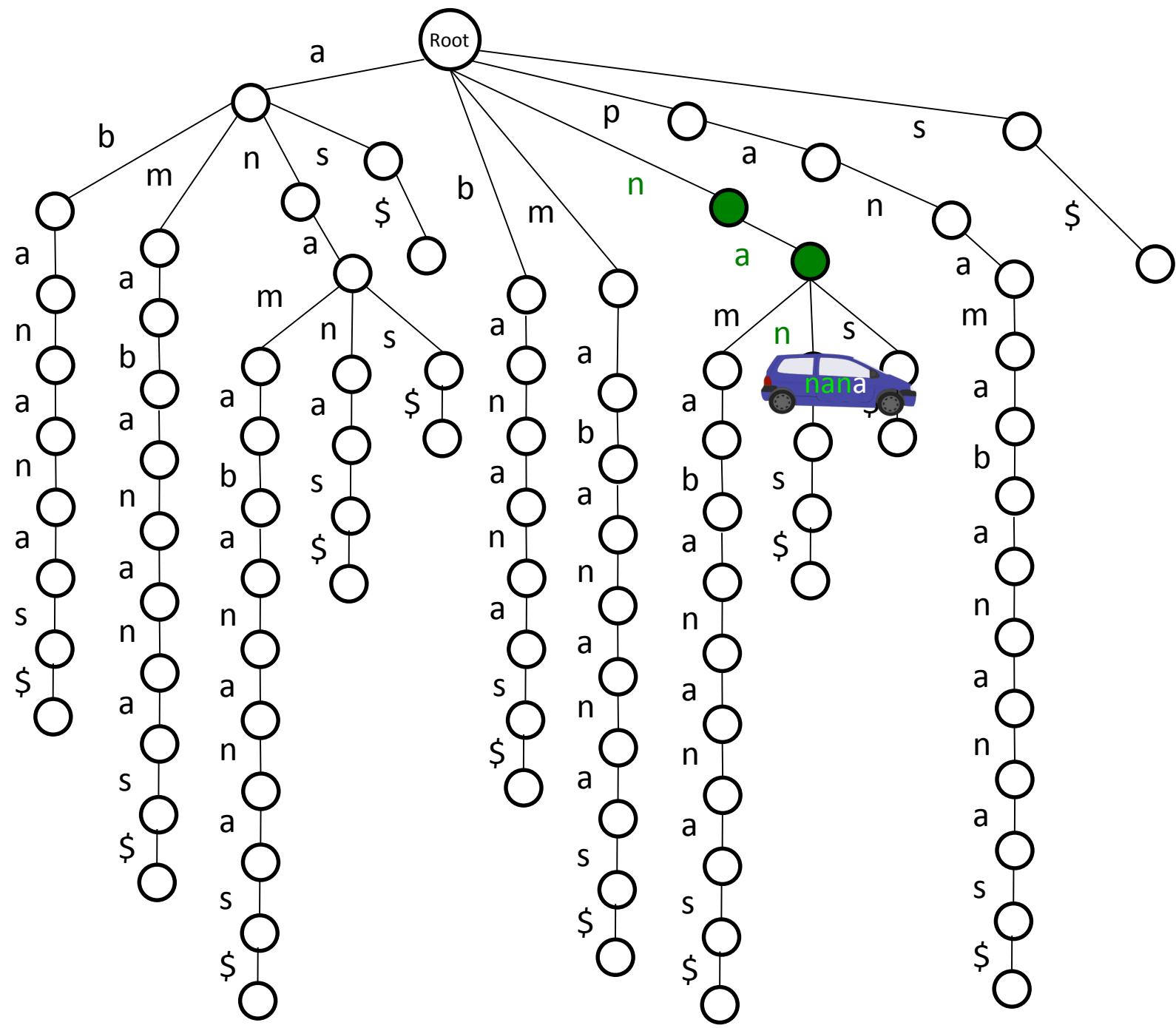


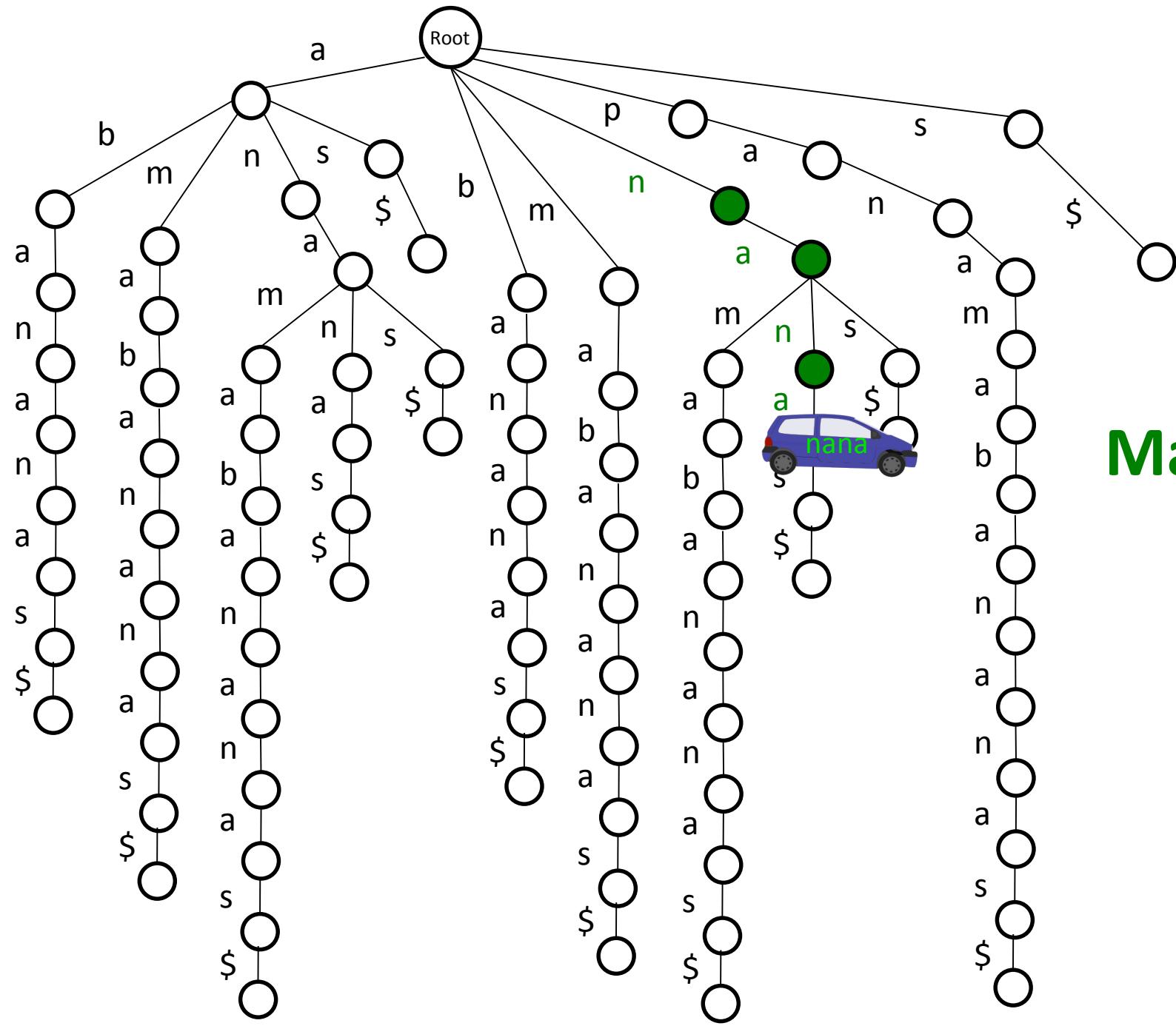
## SuffixTrie(*Text*)



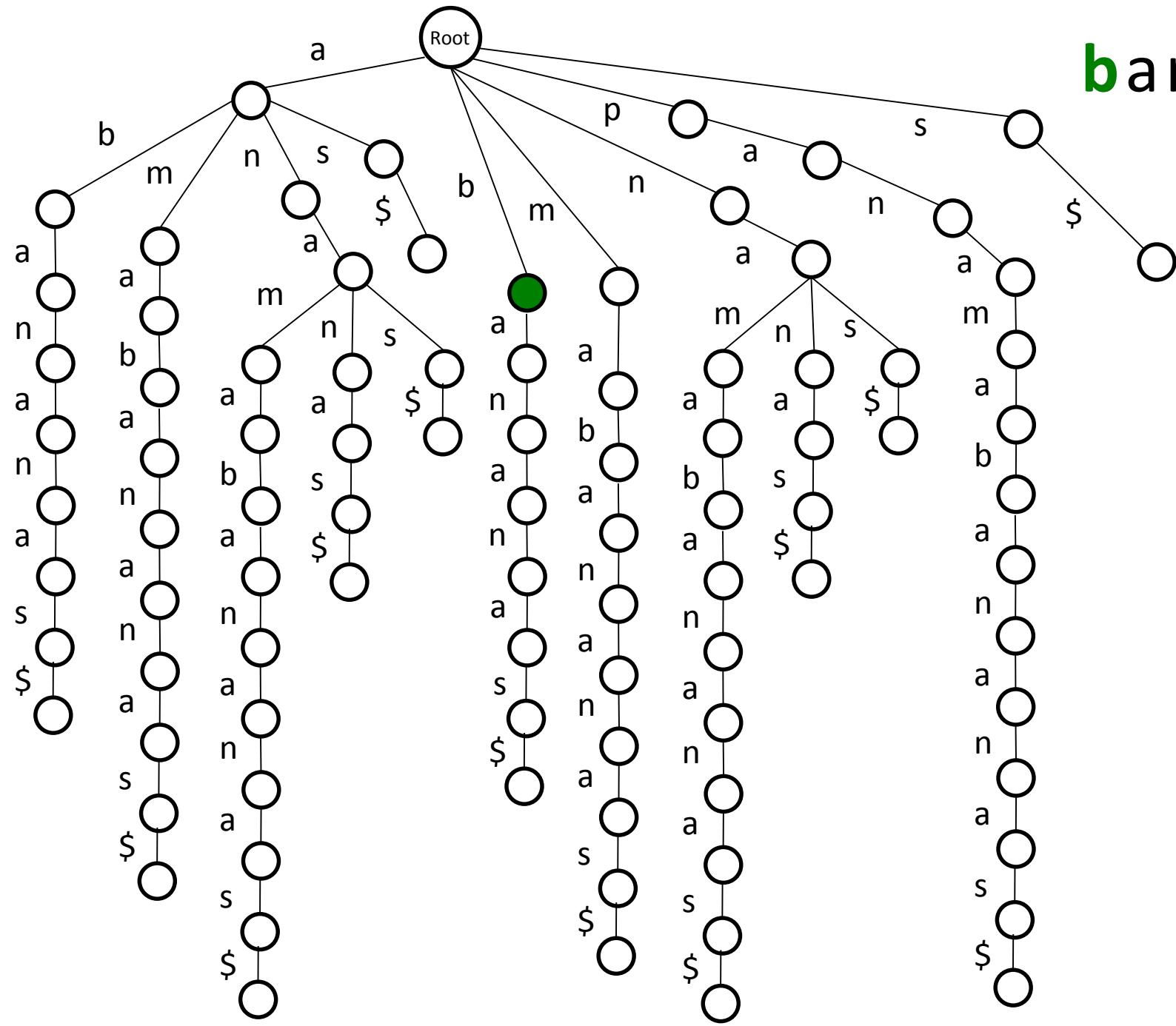




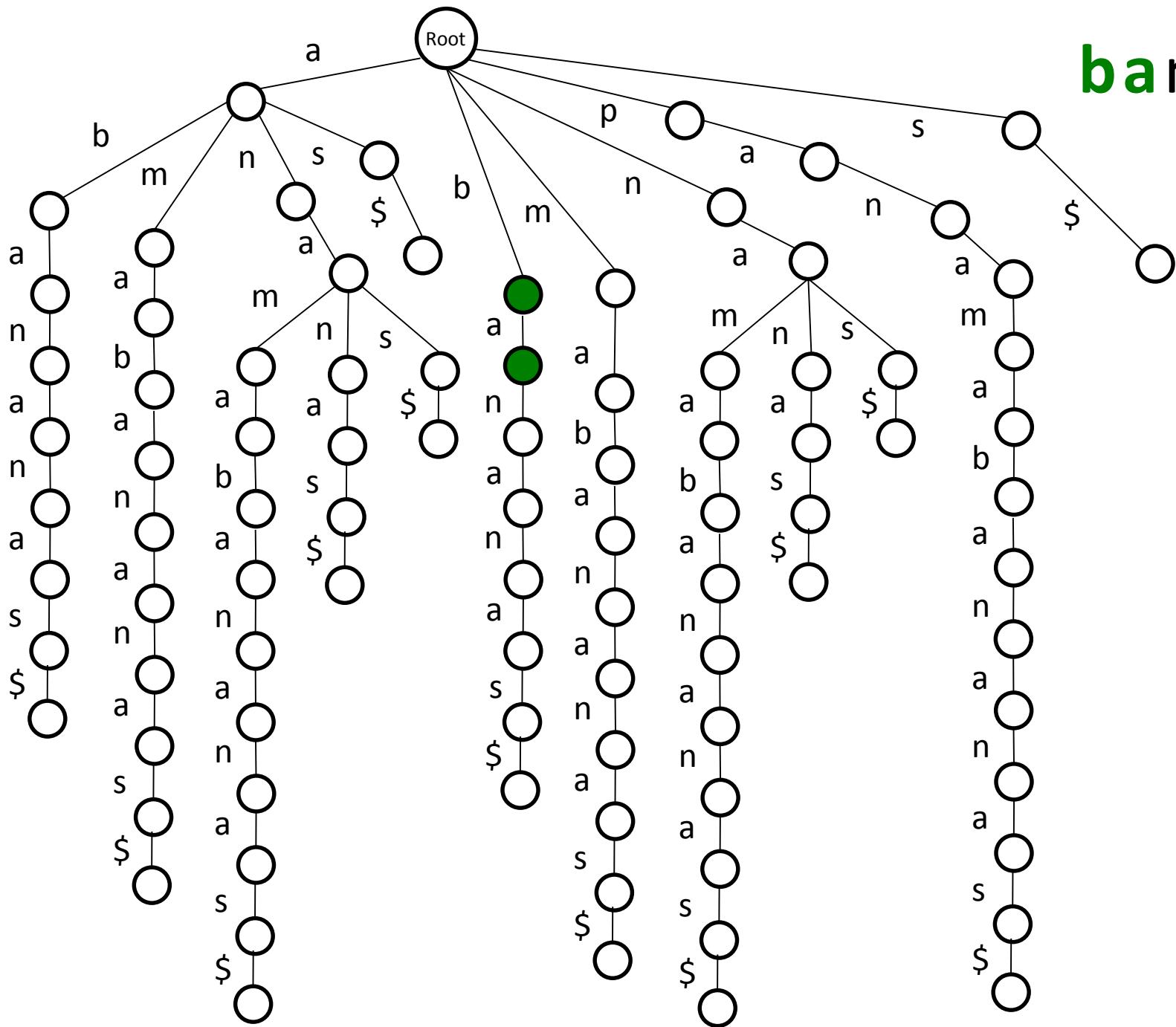




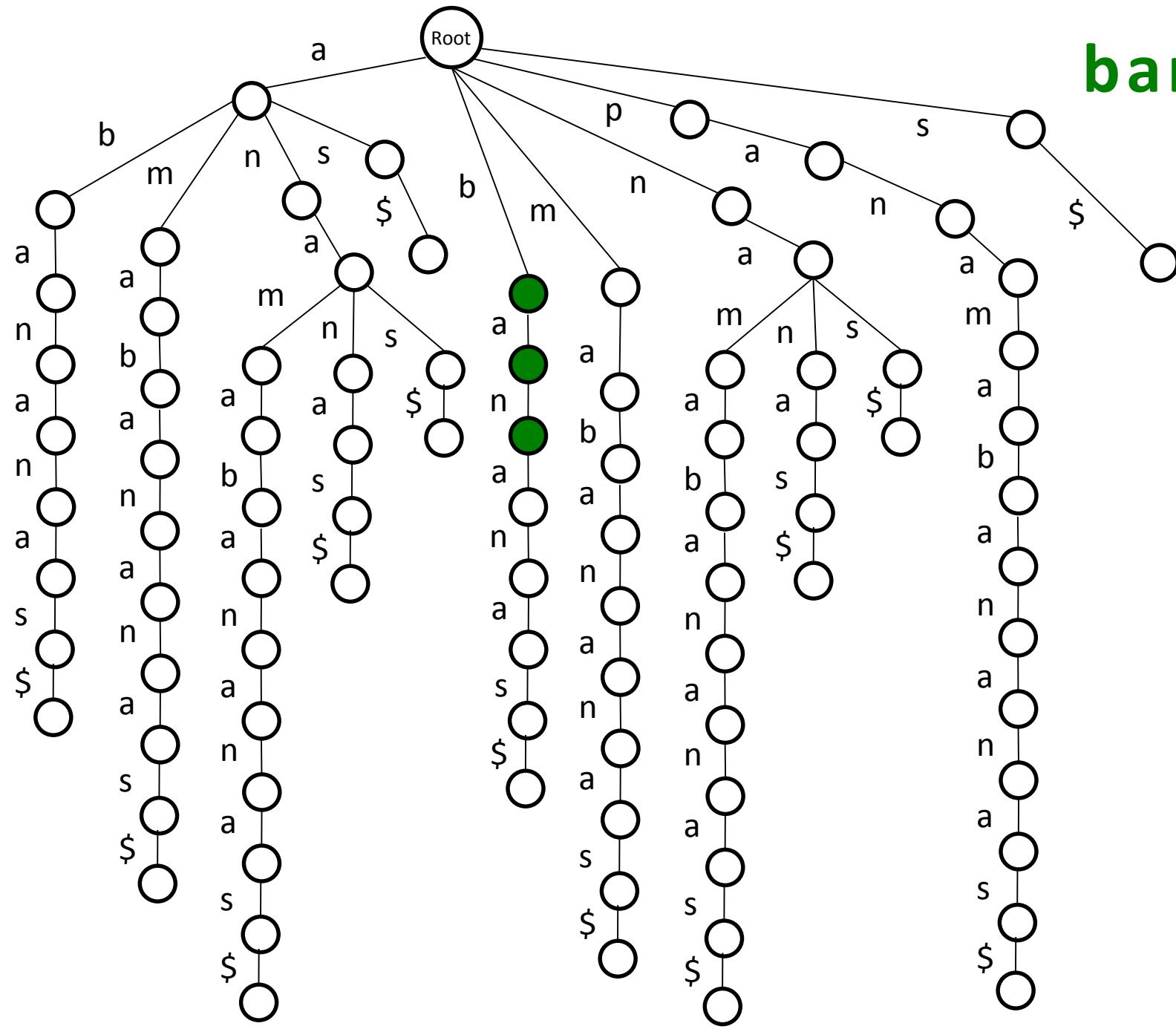
# Match!



**b**anana

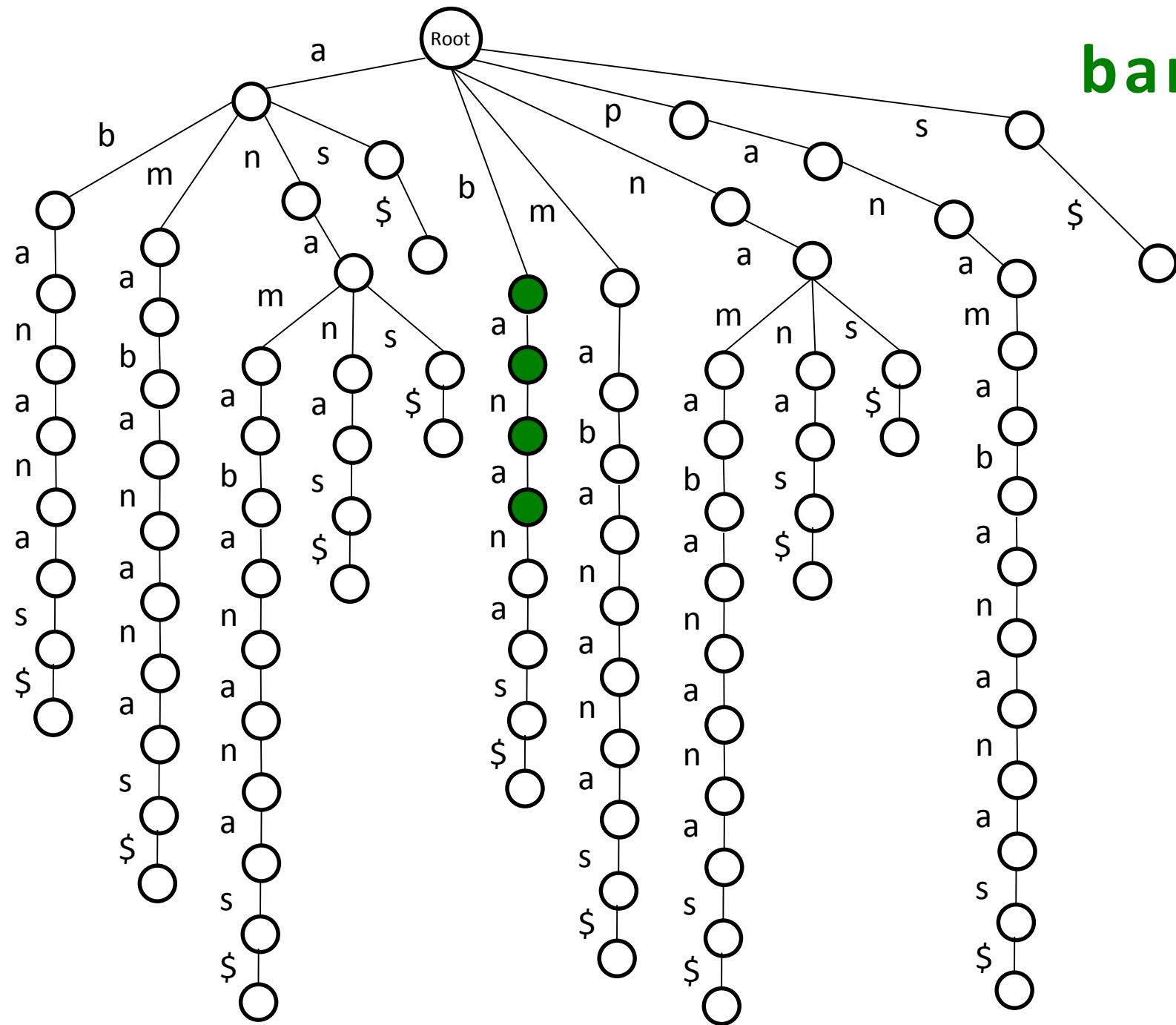


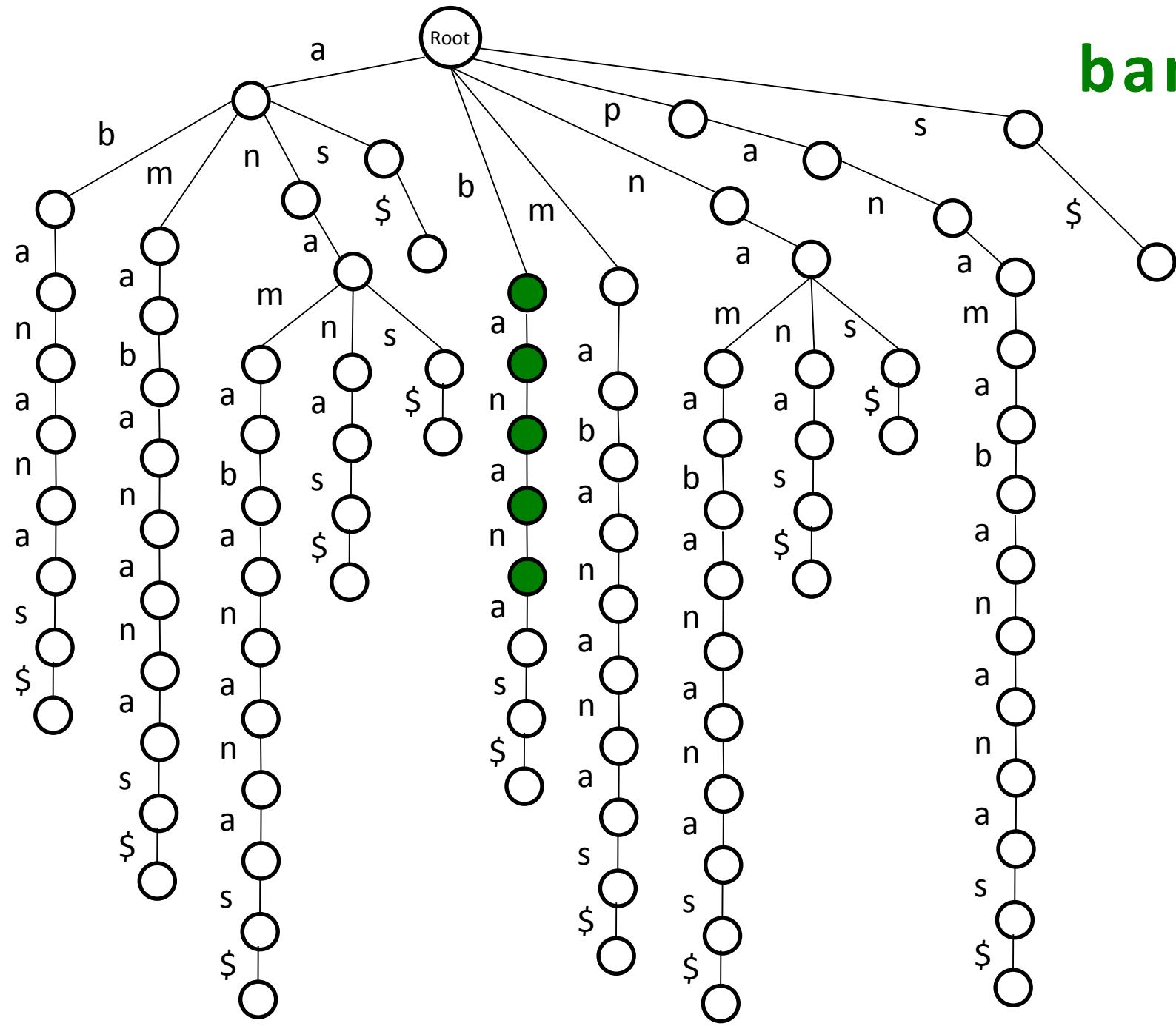
**banana**



# banana

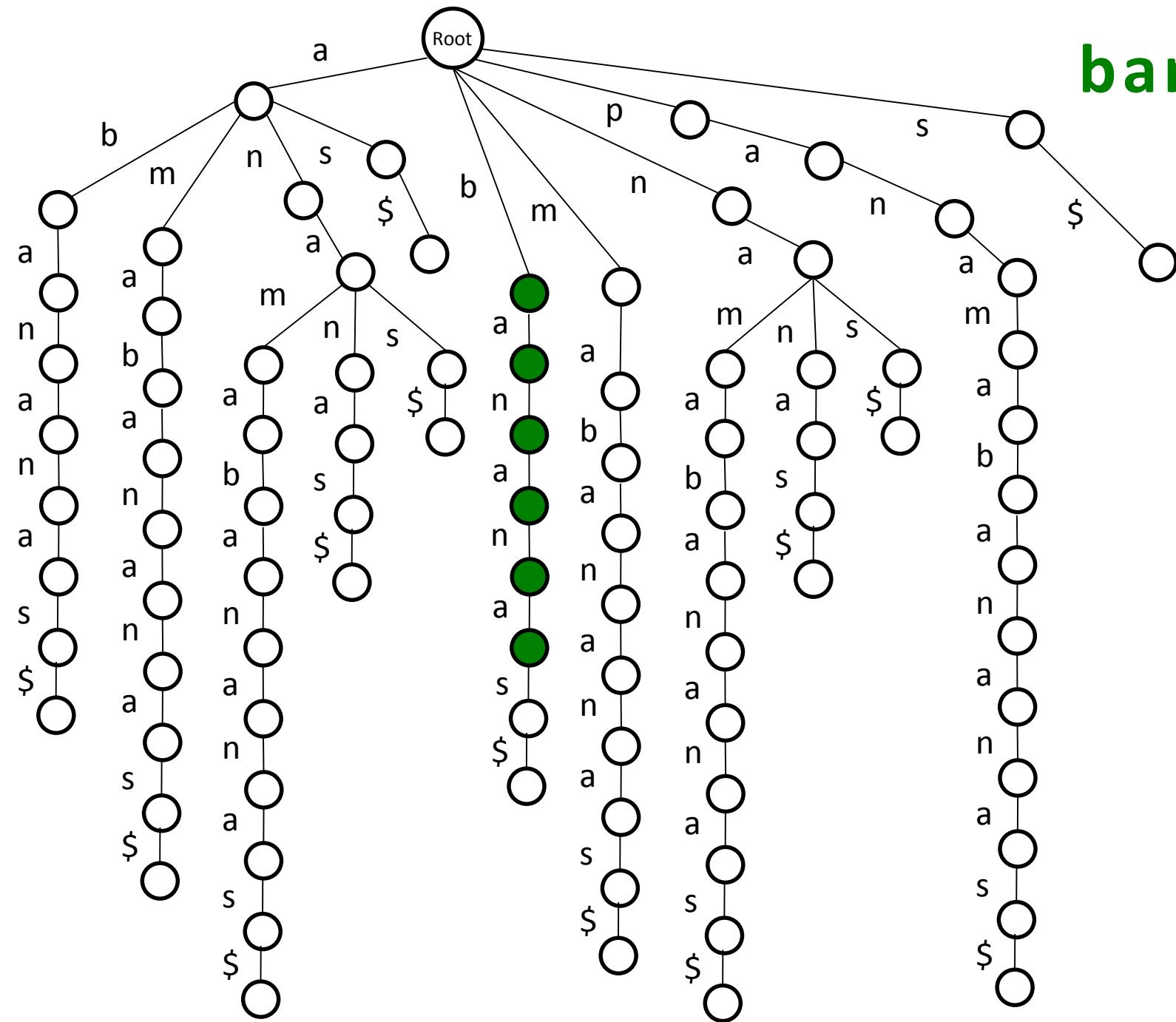
**banana**



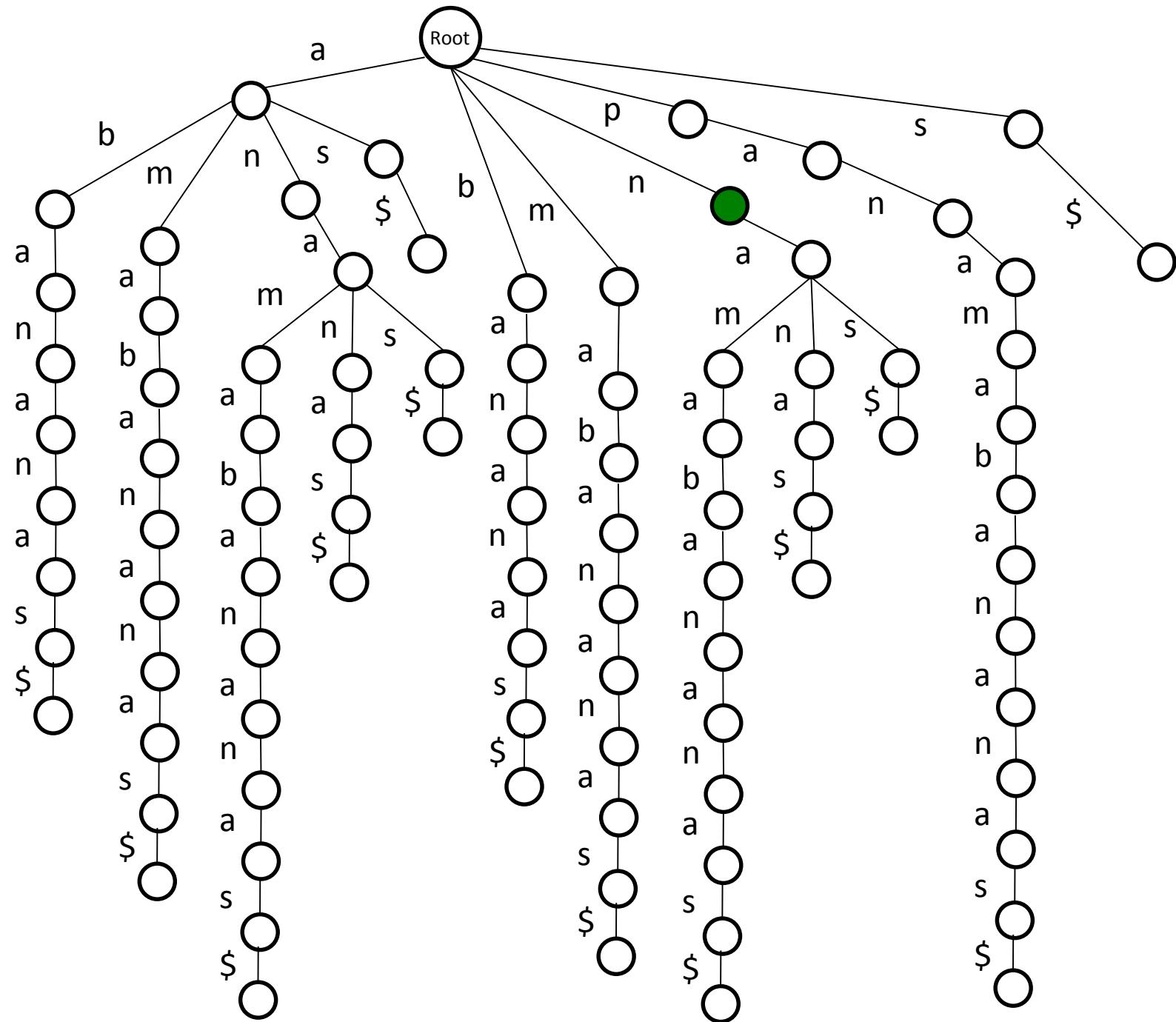


**banana**

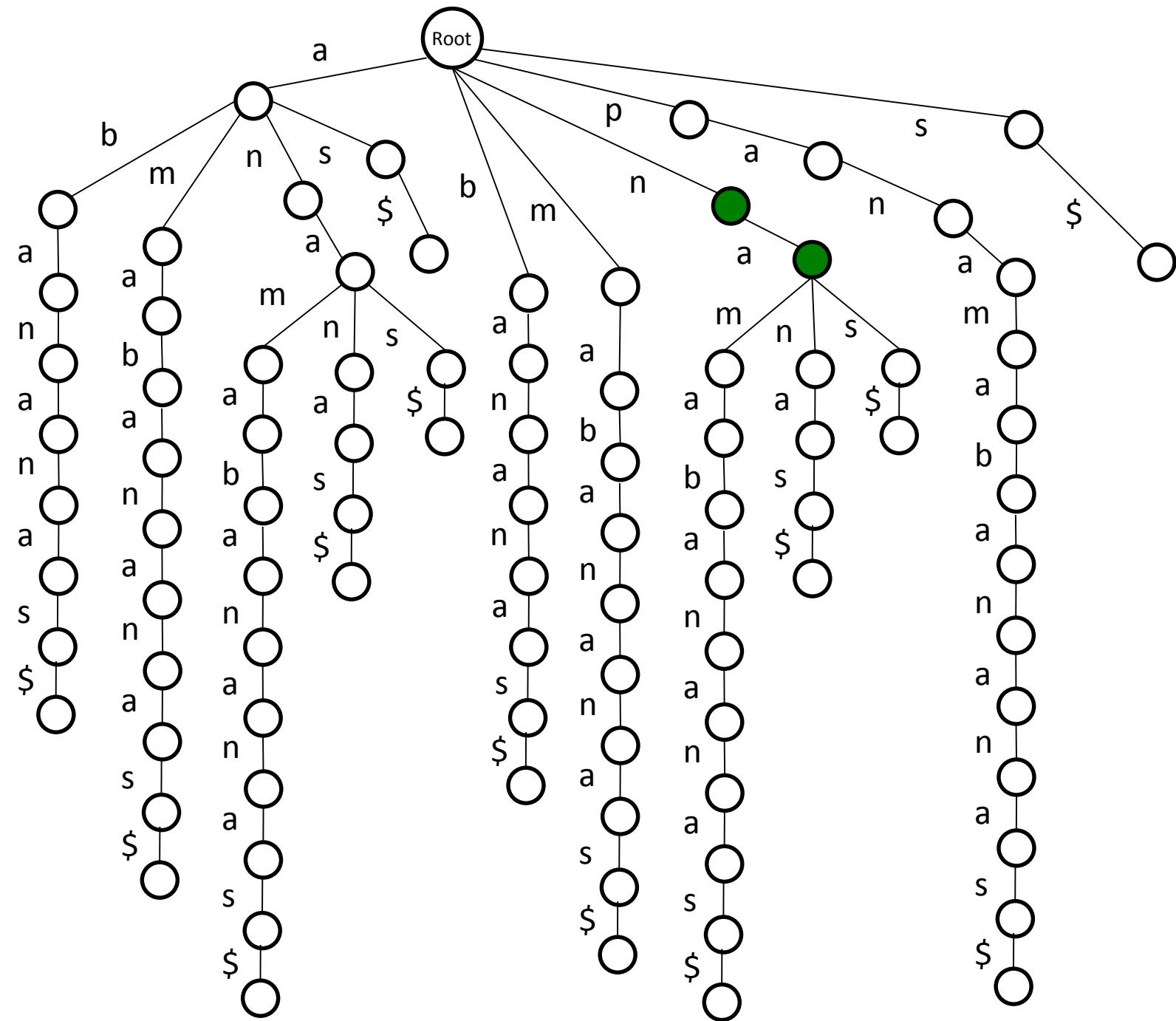
**banana**



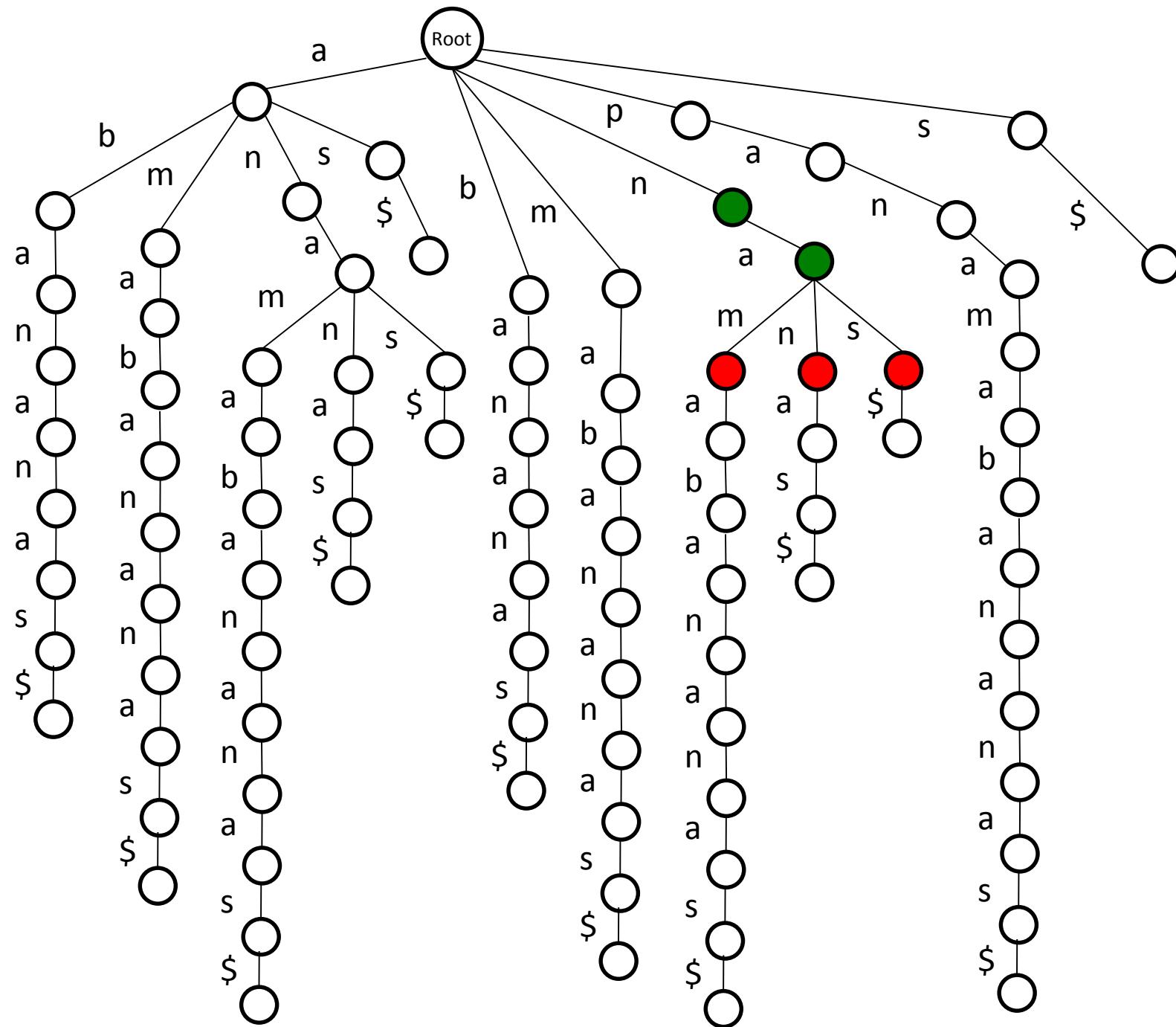
n a b

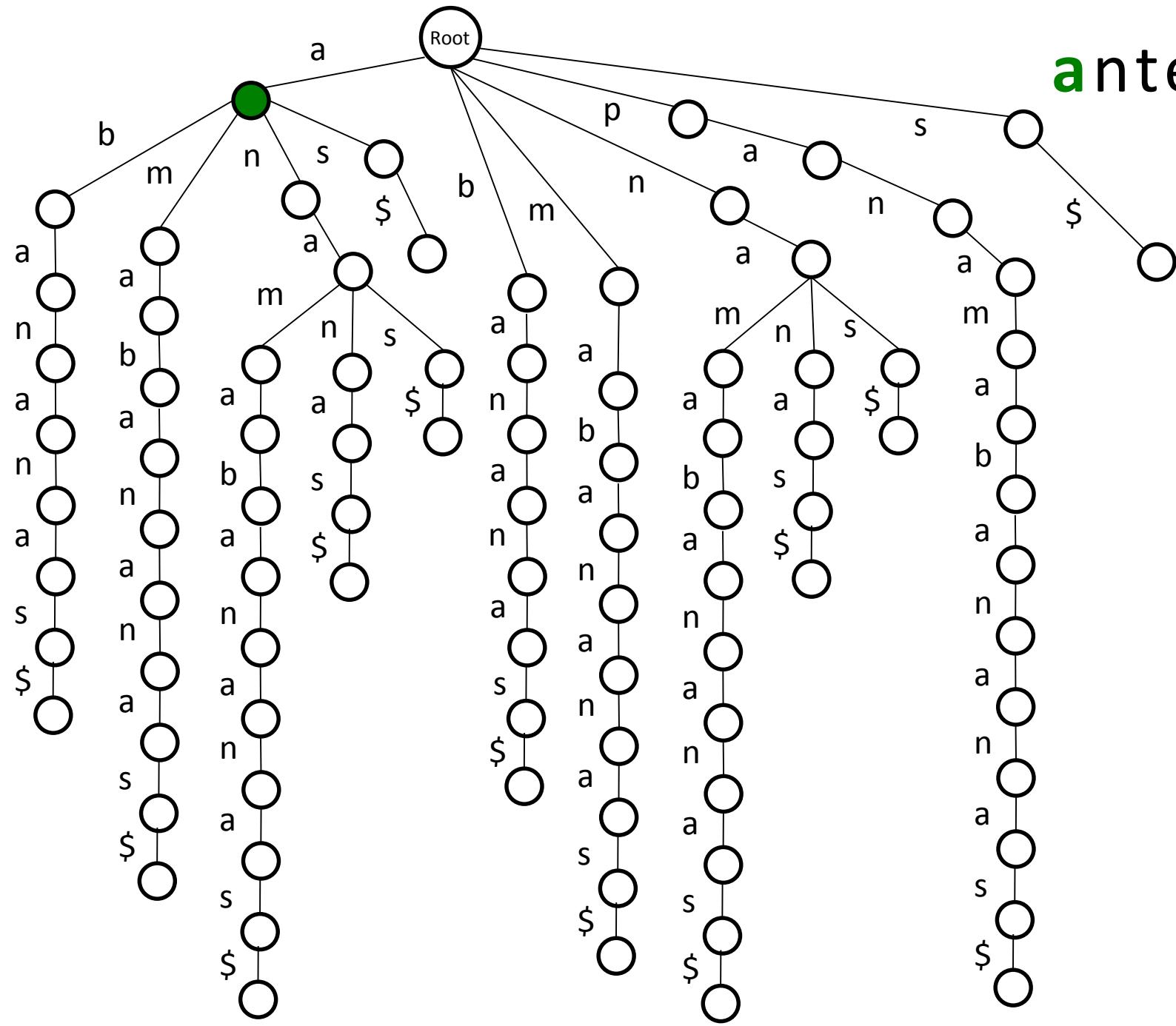


**nab**

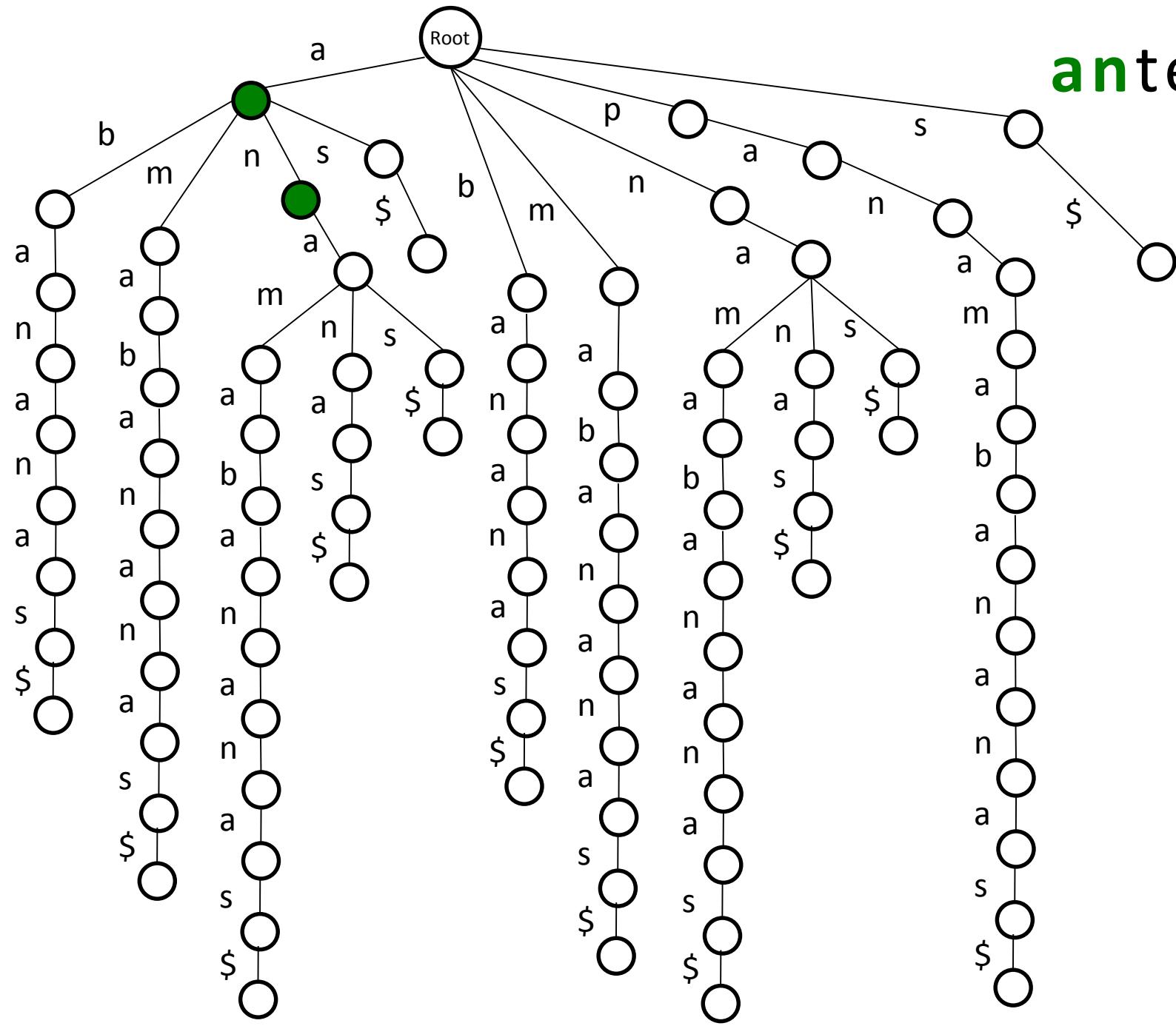


# nab

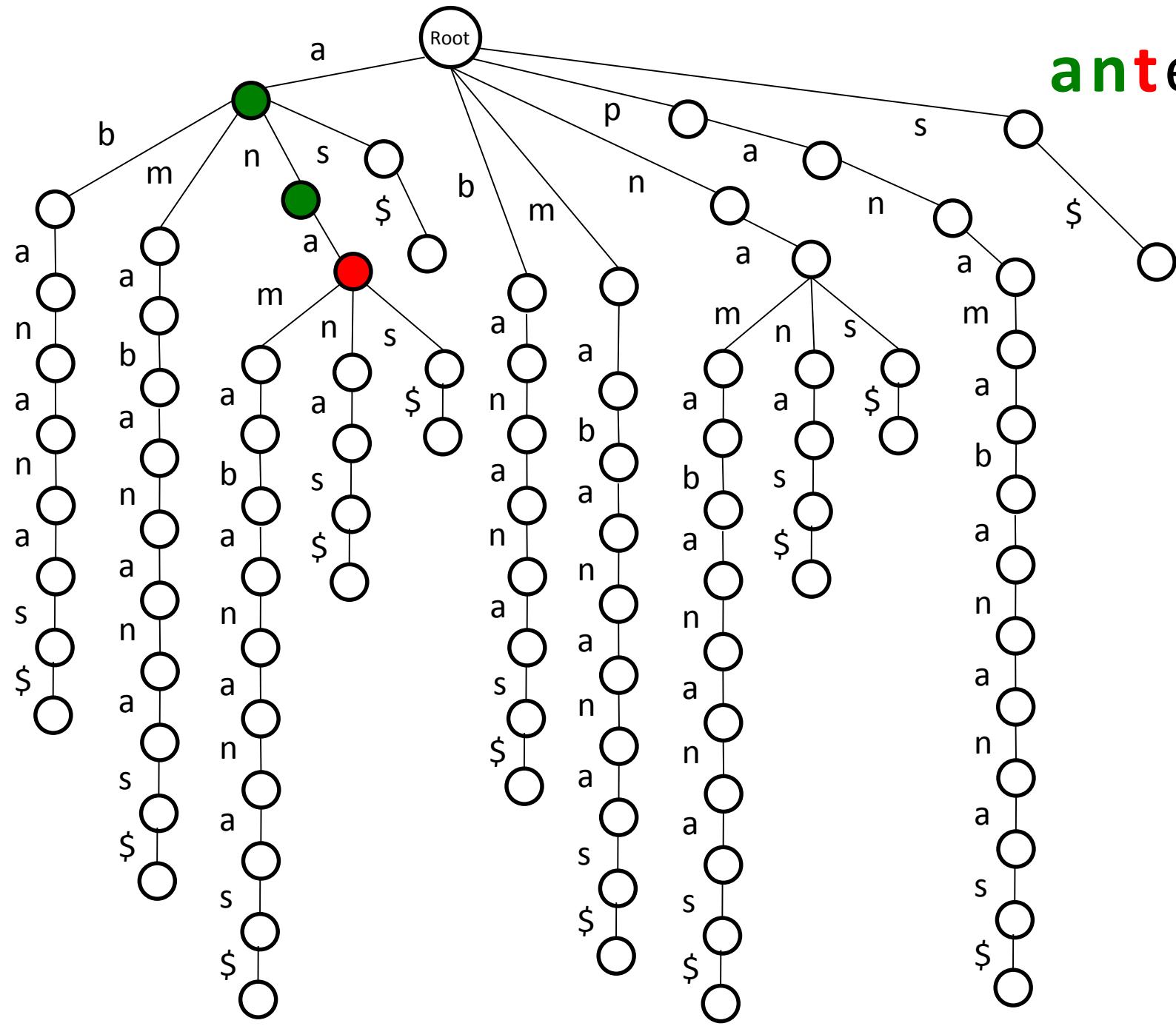




# antenna

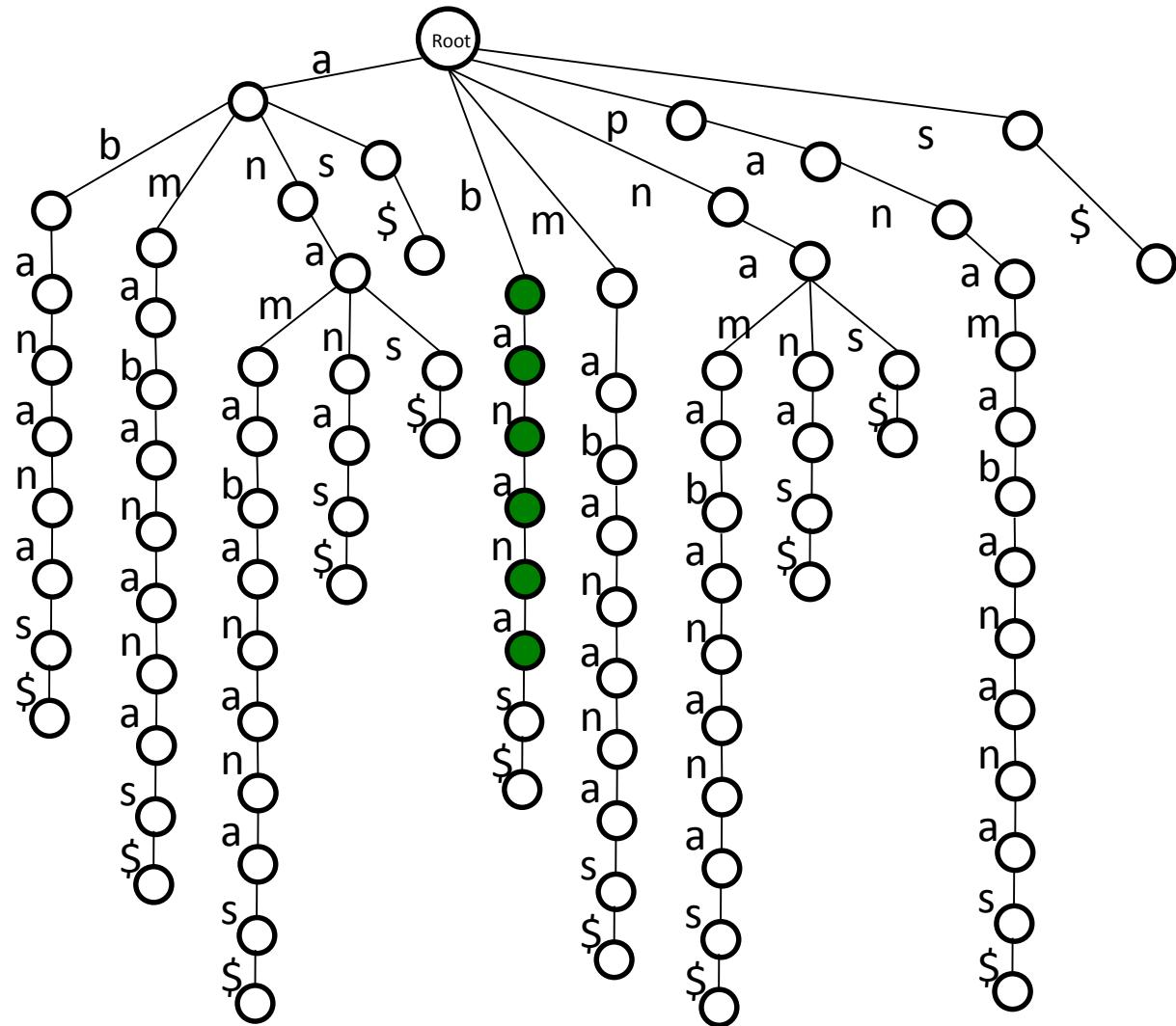


# antenna



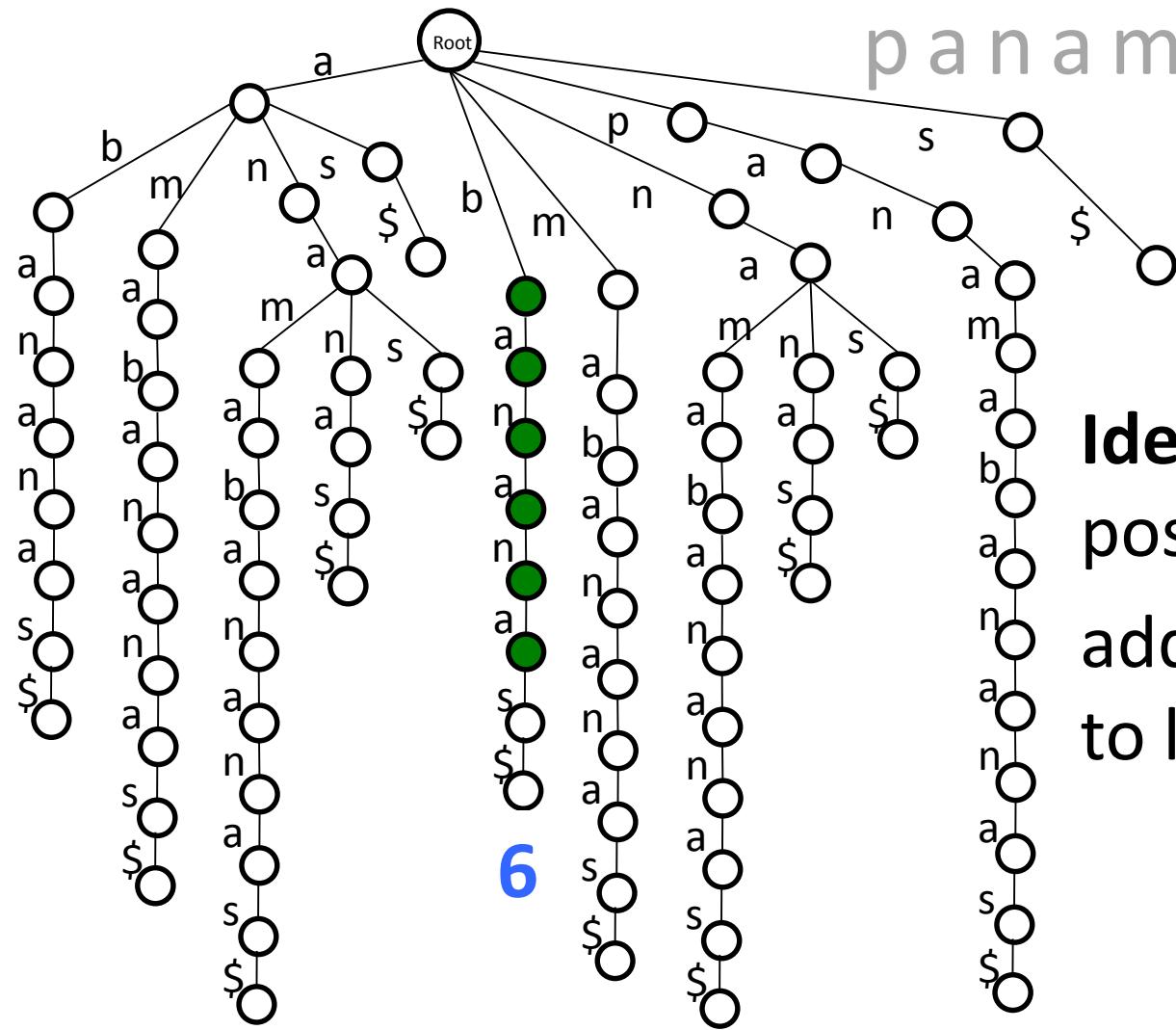
# antenna

# Where Are the Matches???

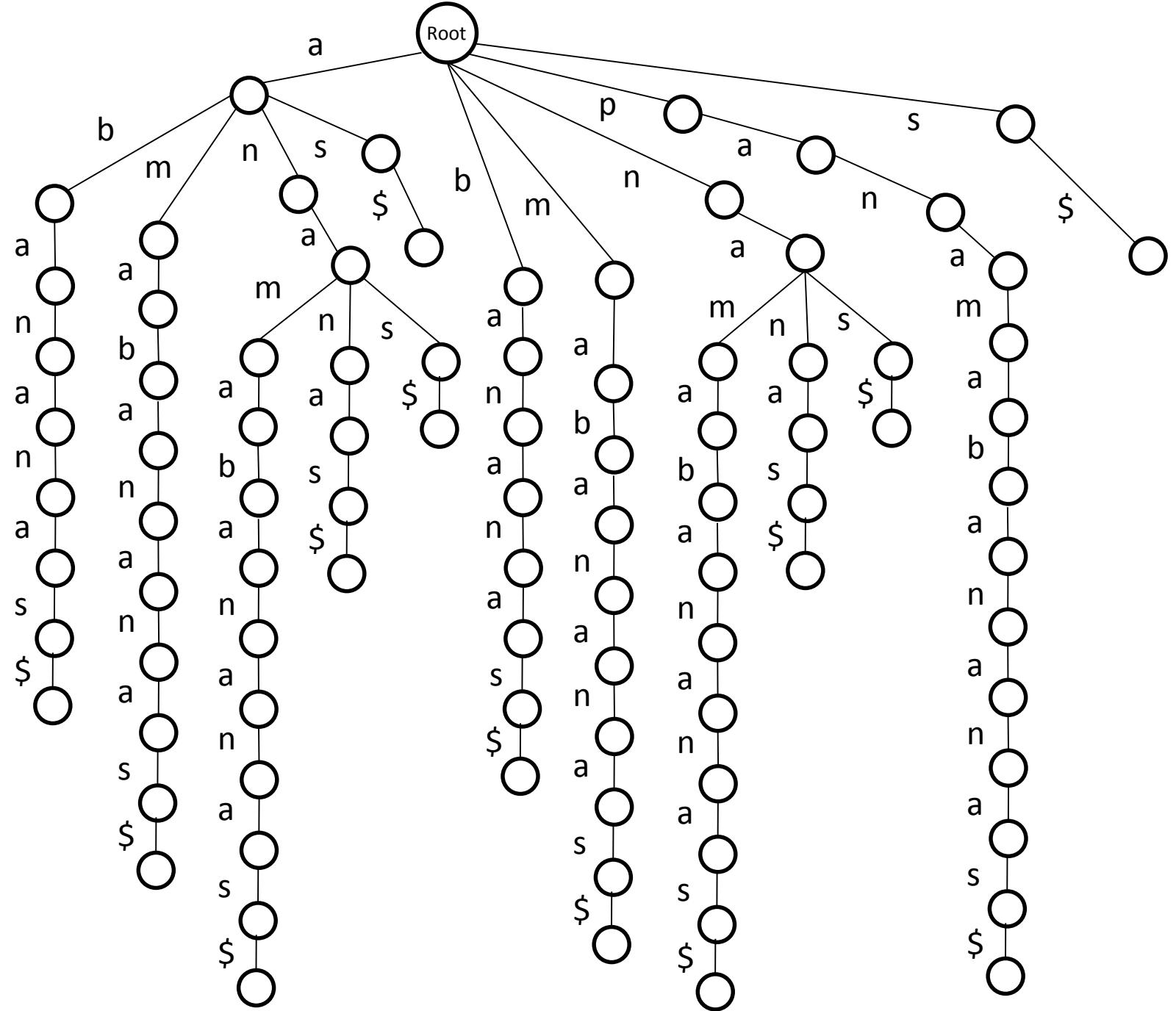


# bananas\$

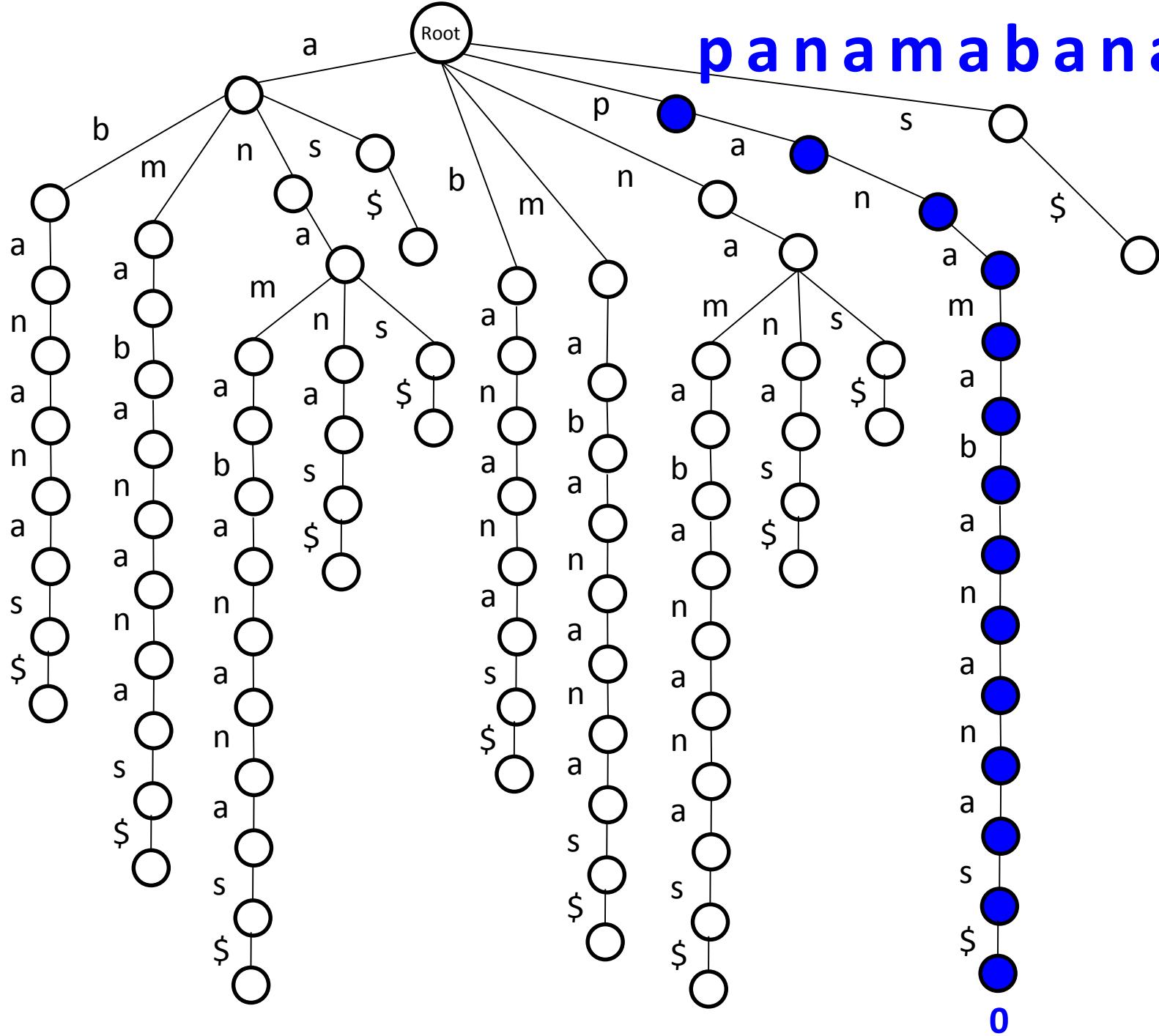
# Where Are the Matches???



**Idea:** to find the positions of matches, add some information to leaves

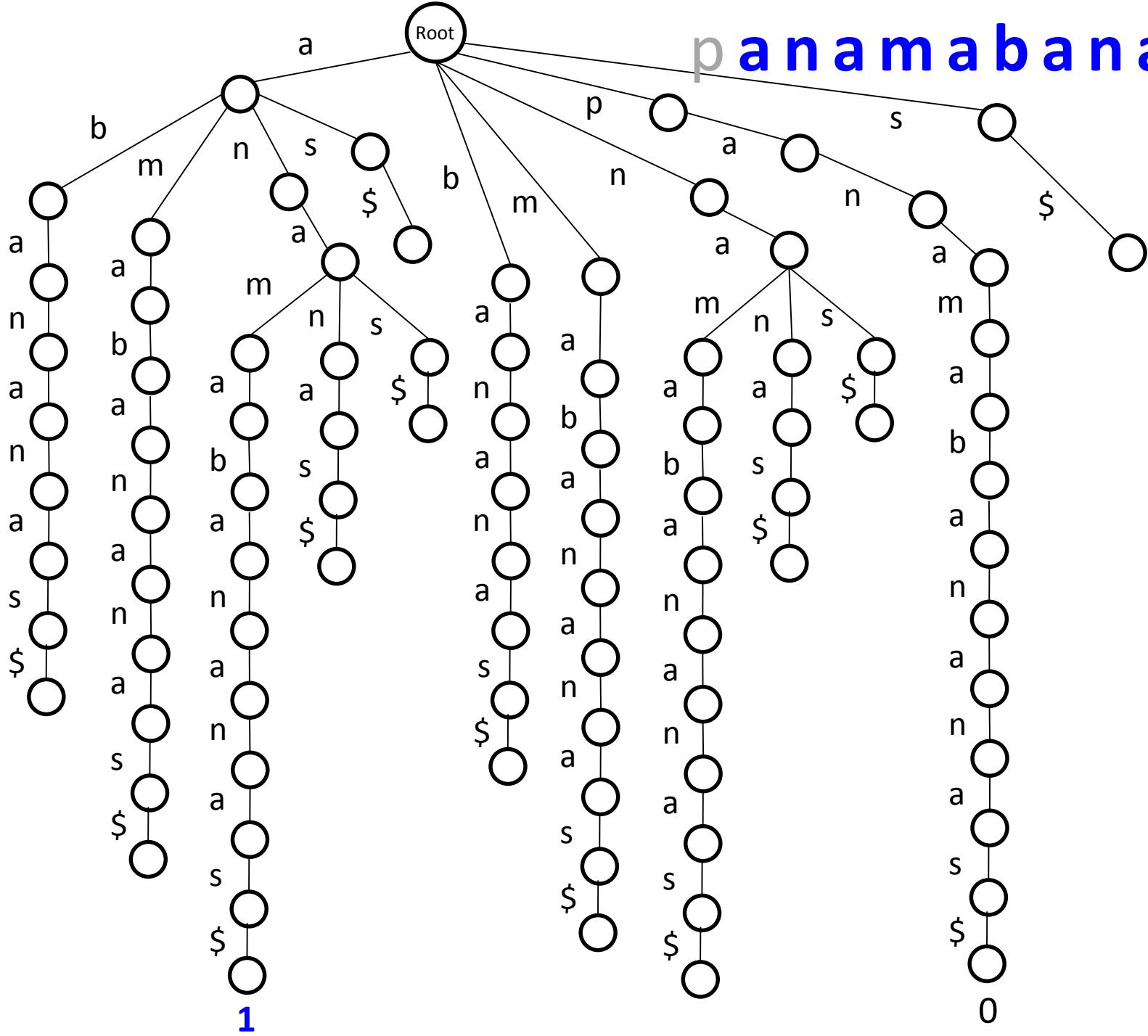


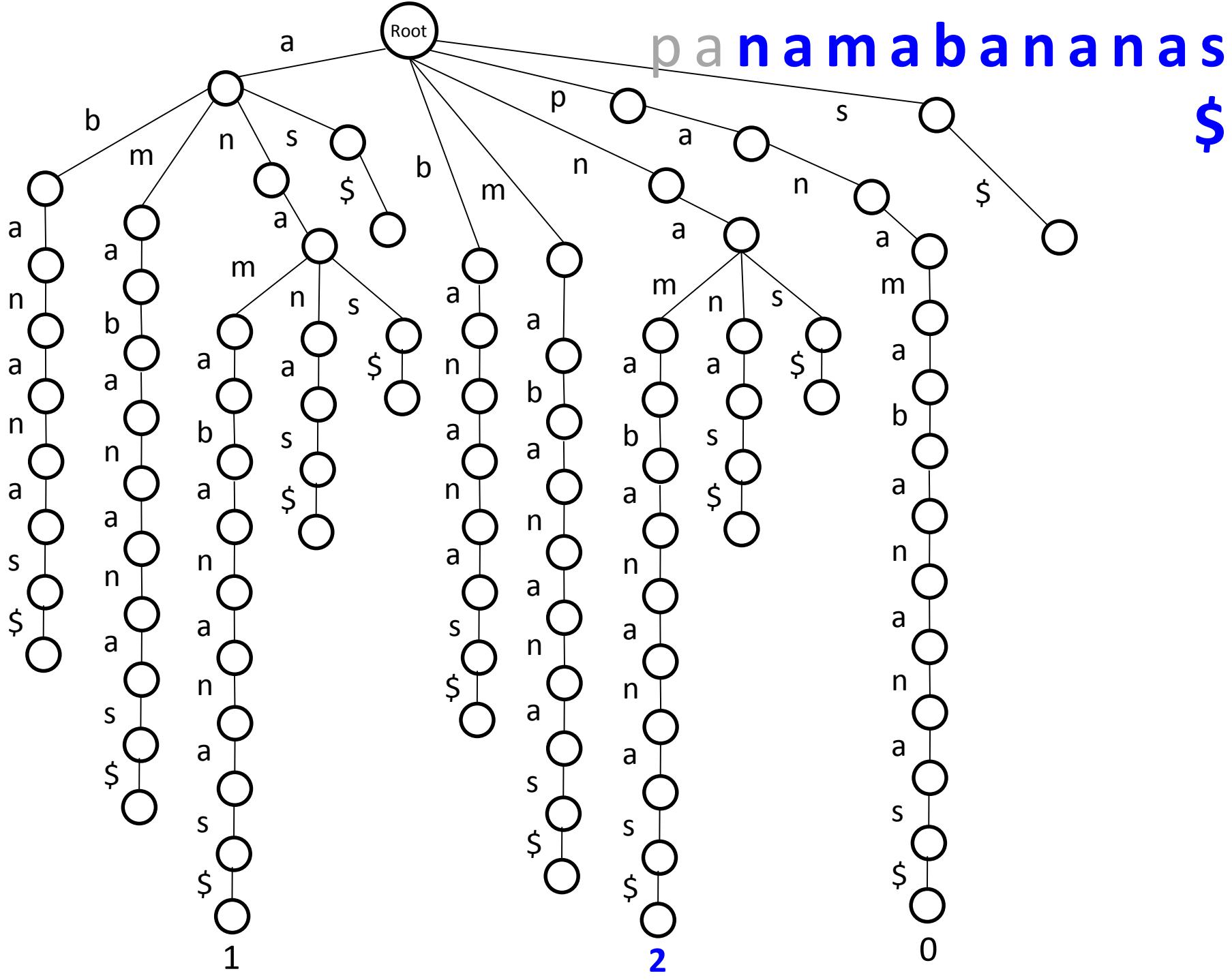
panamabananas

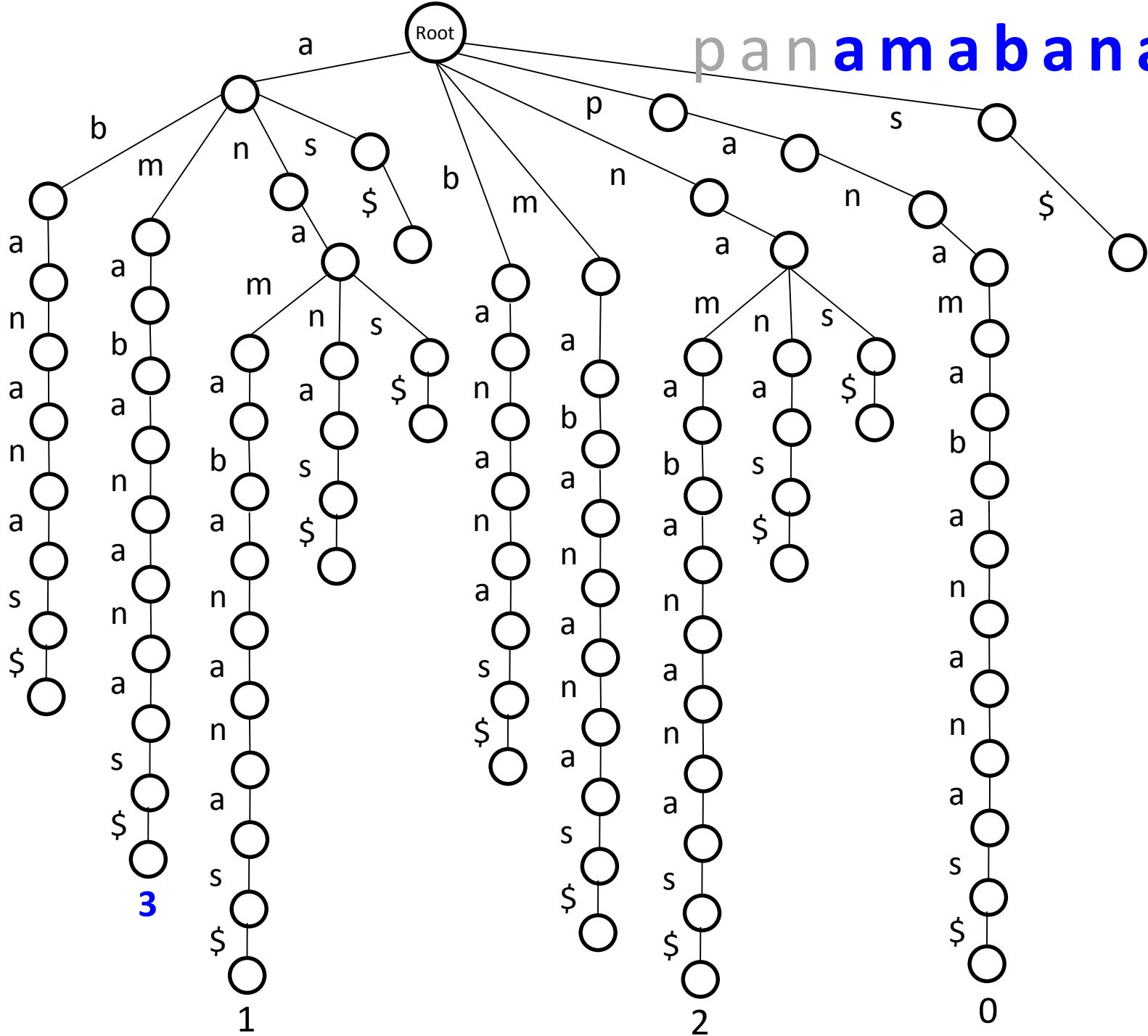


\$

0







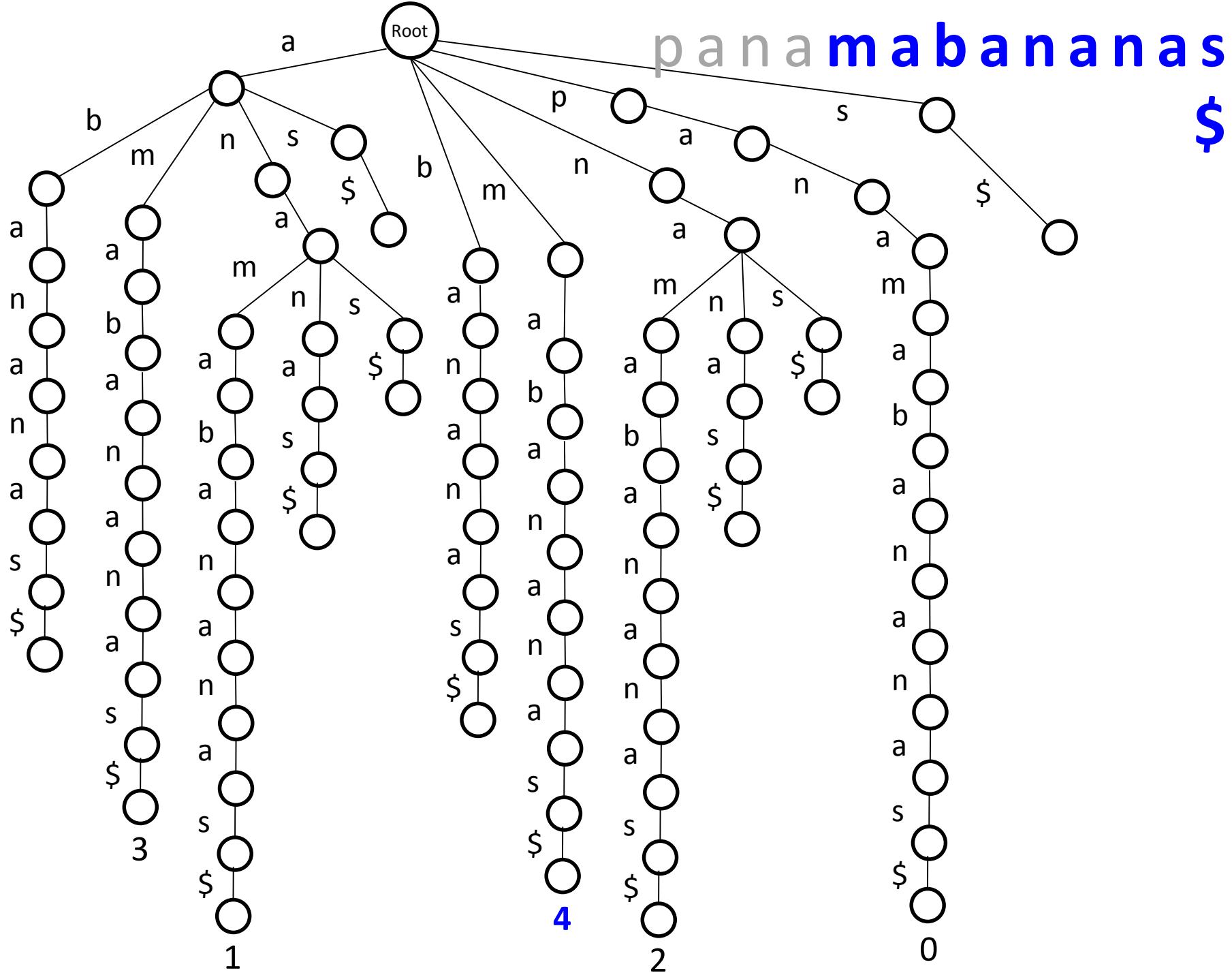
panamabananas  
\$

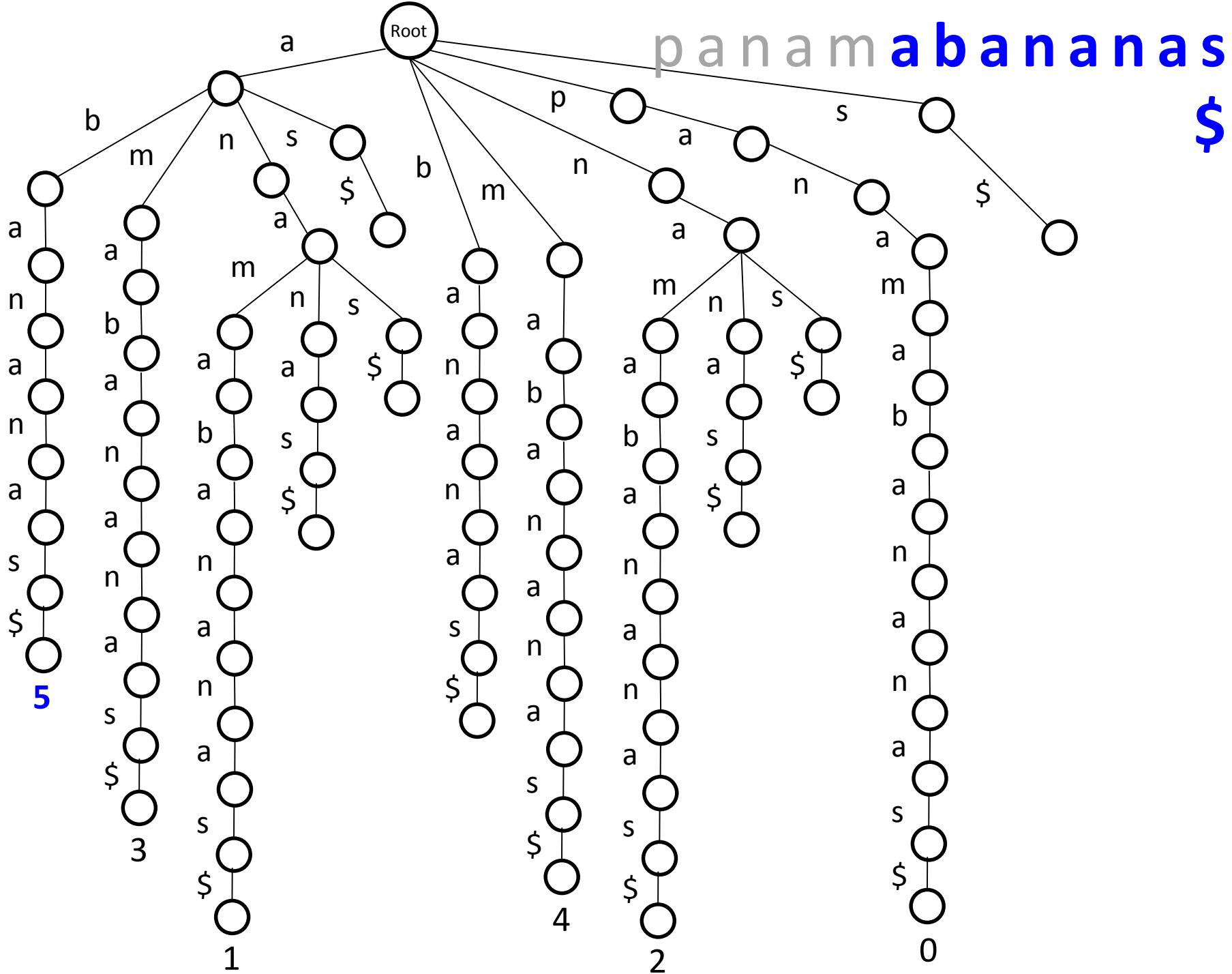
3

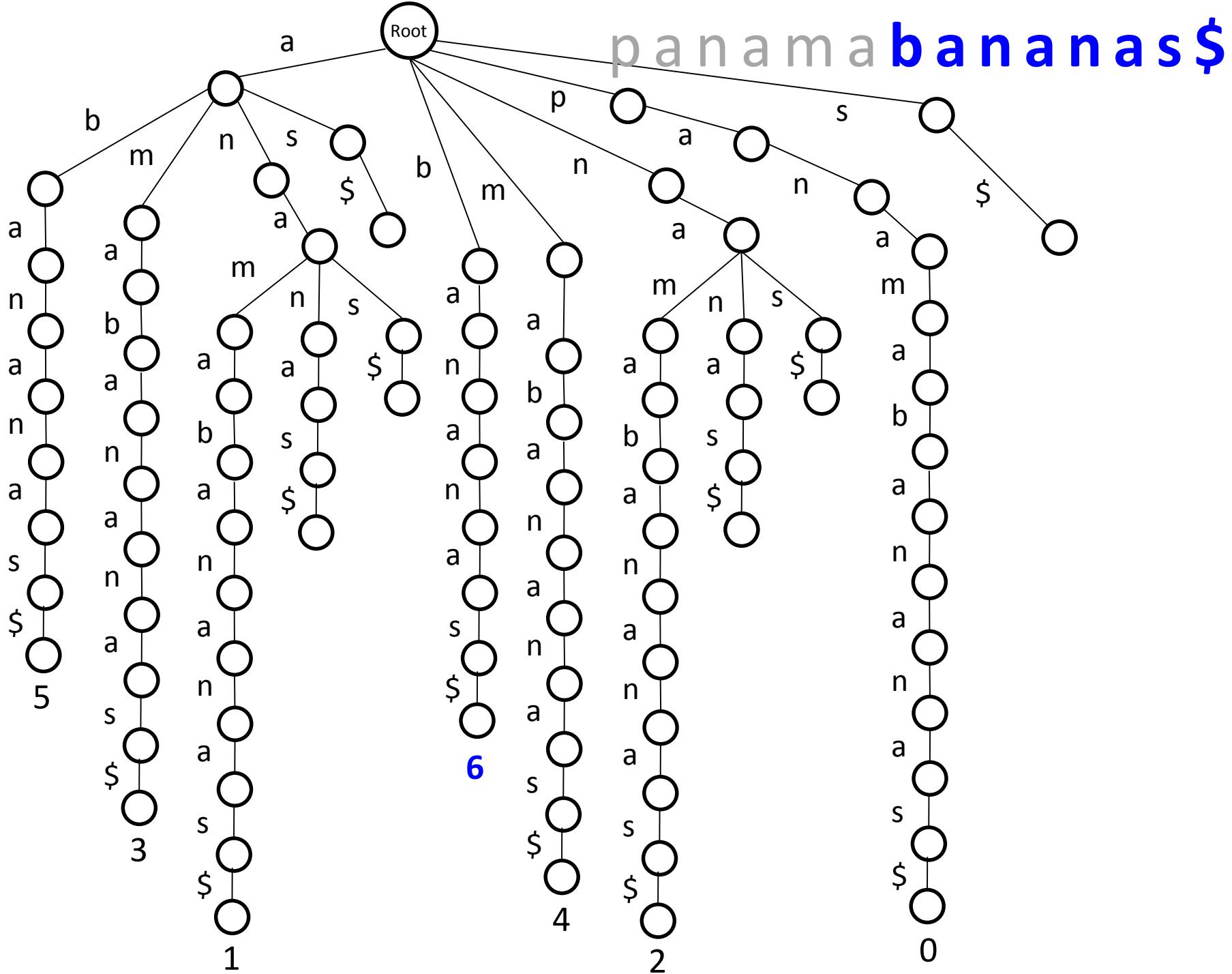
1

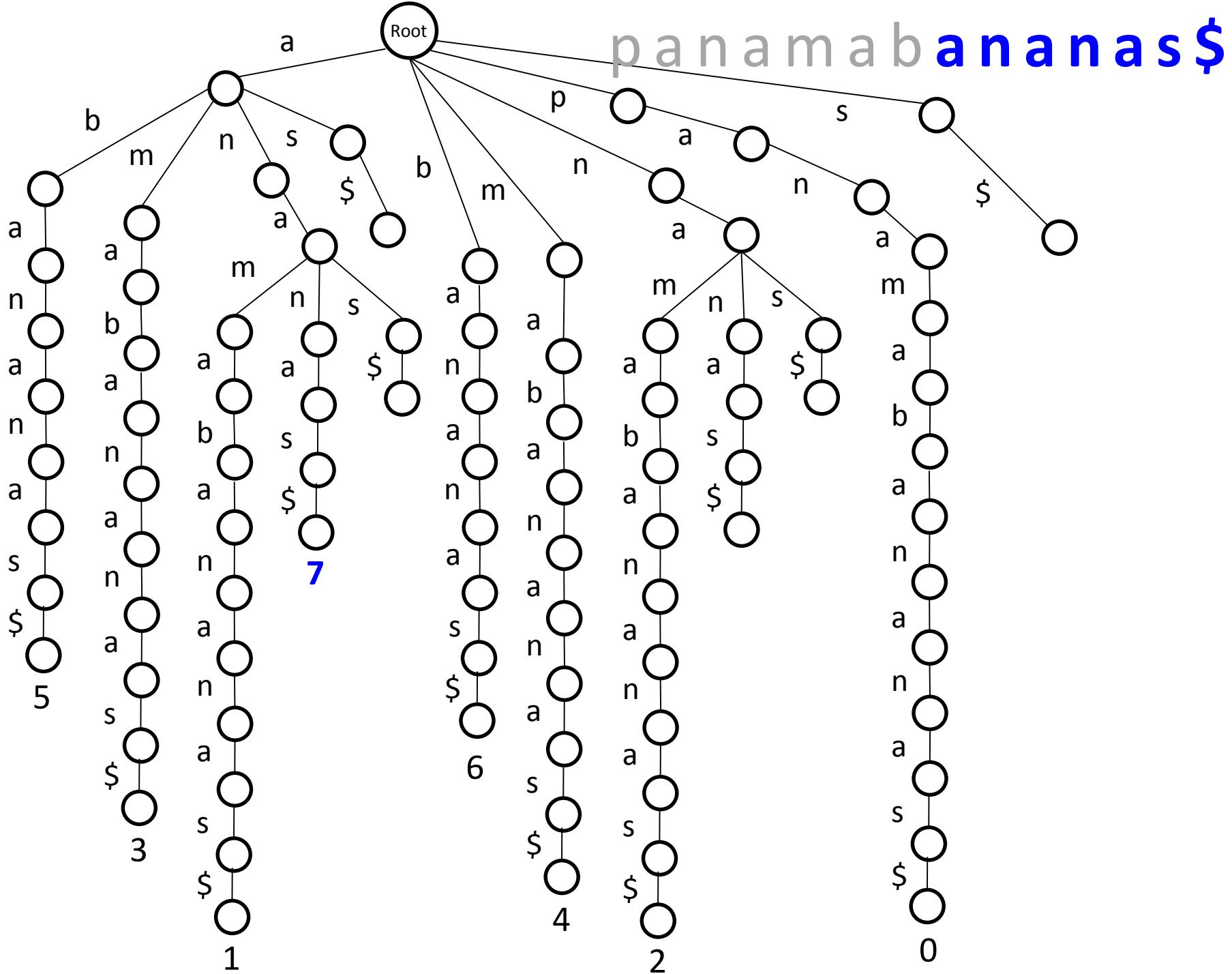
2

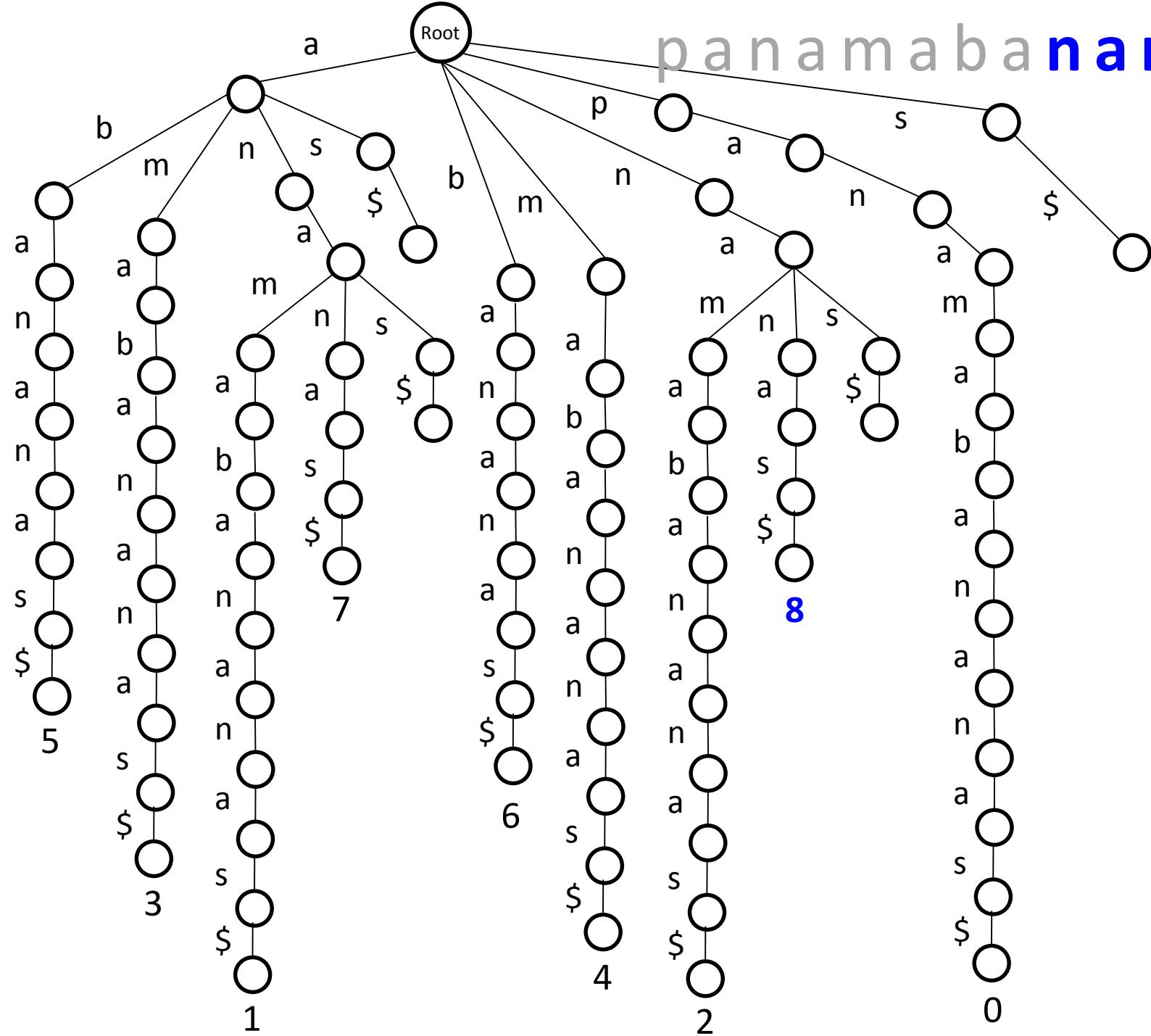
0



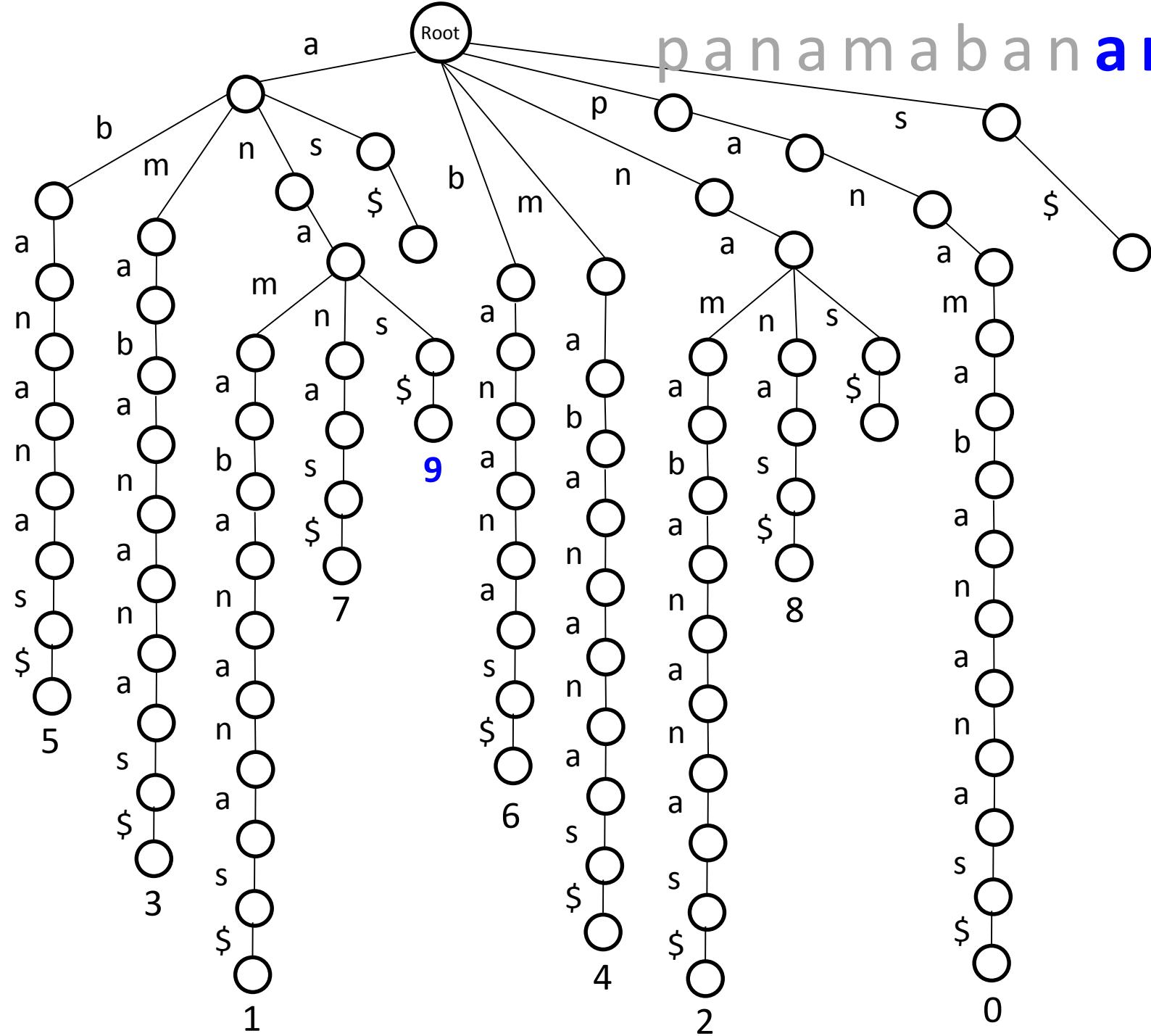




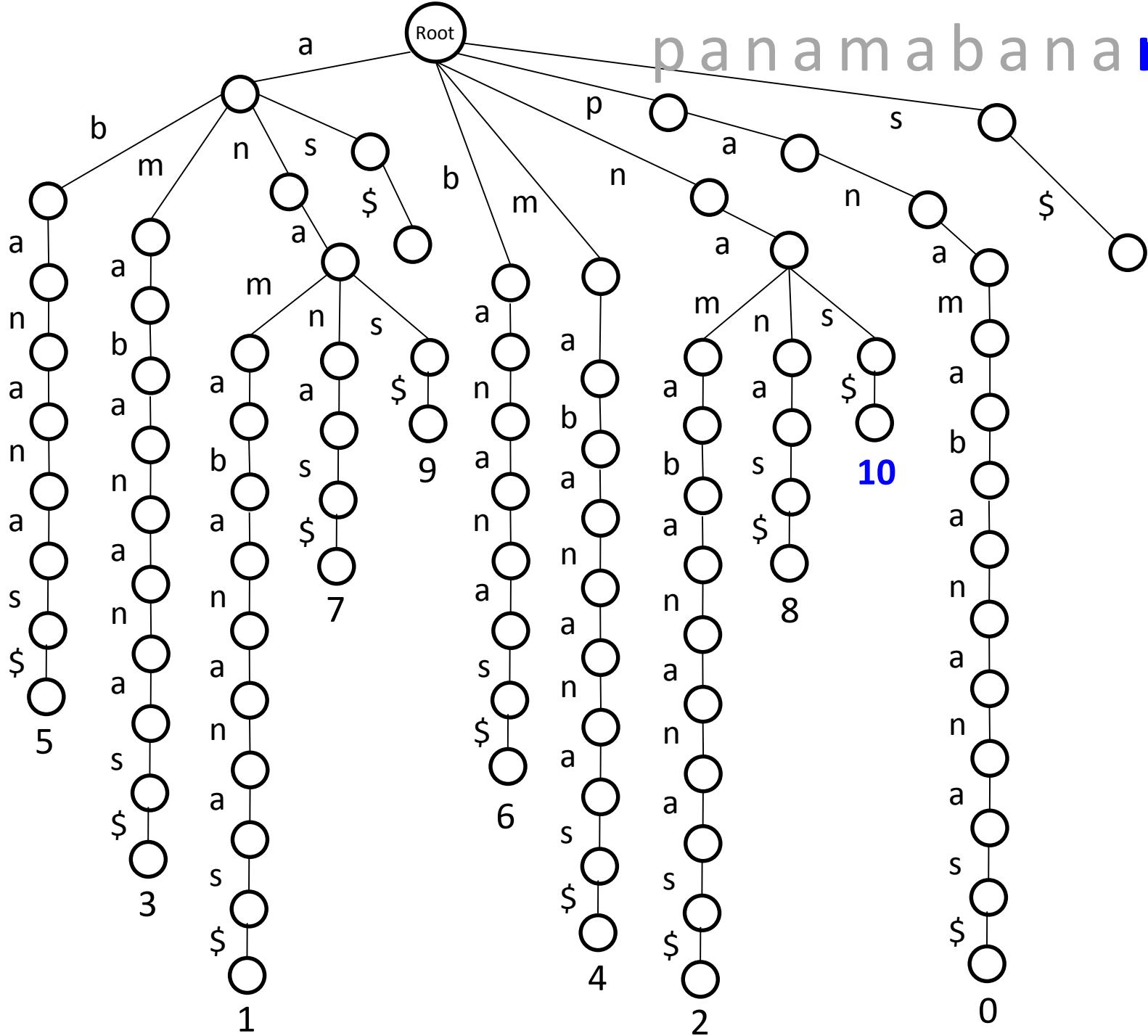




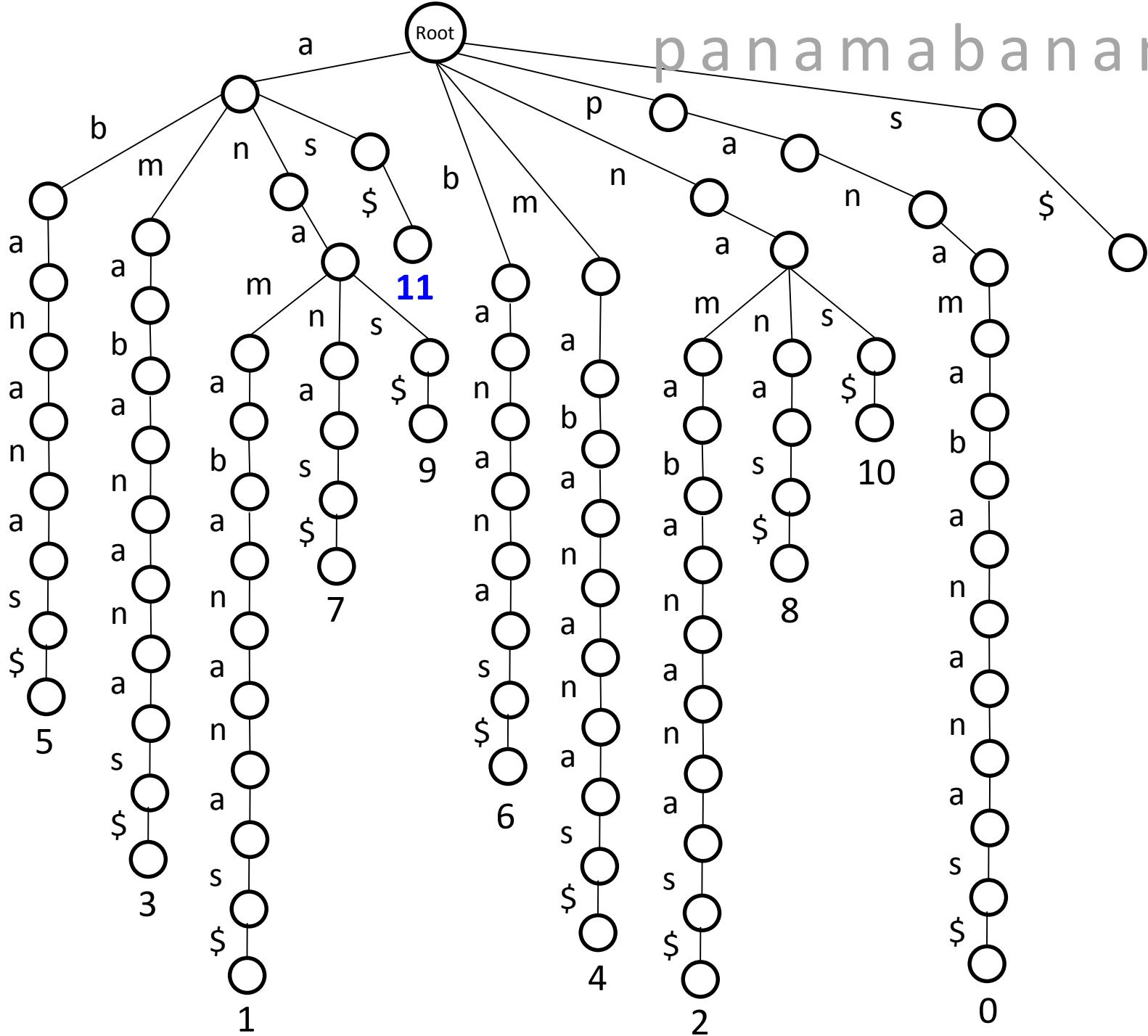
# panamabananas\$



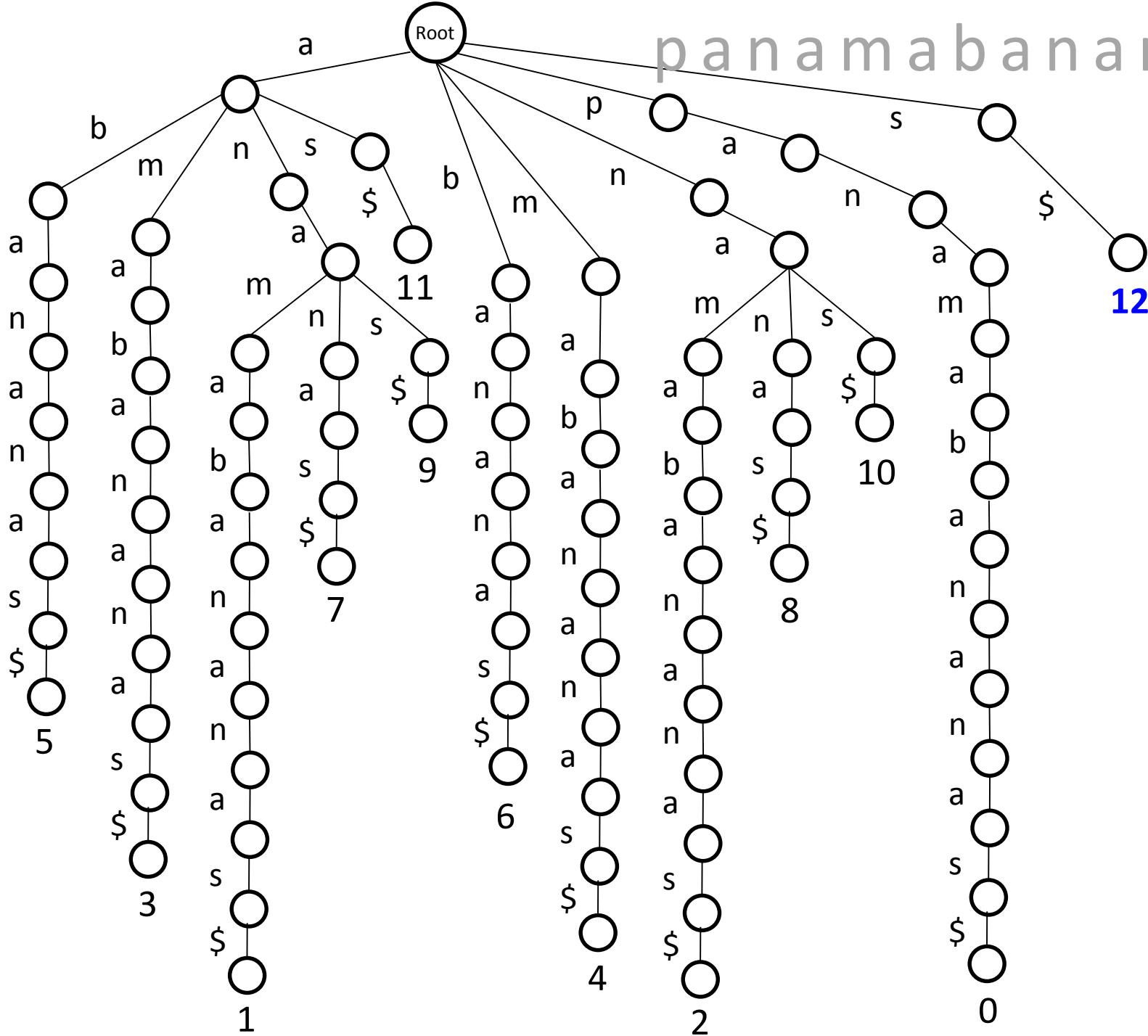
# panamabananas\$



panamabana**nas\$**

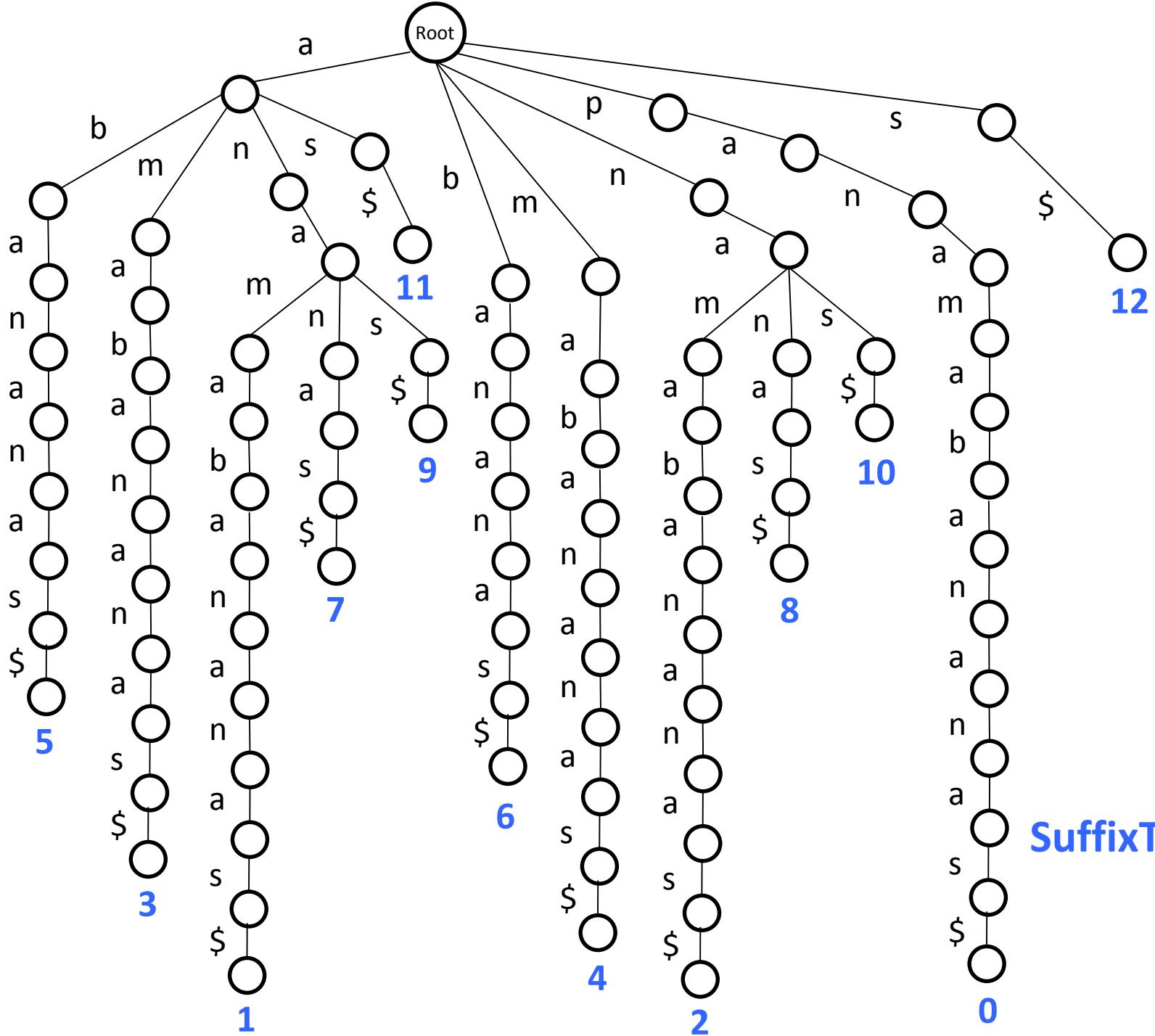


panamabananas\$



panamabanas\$

12

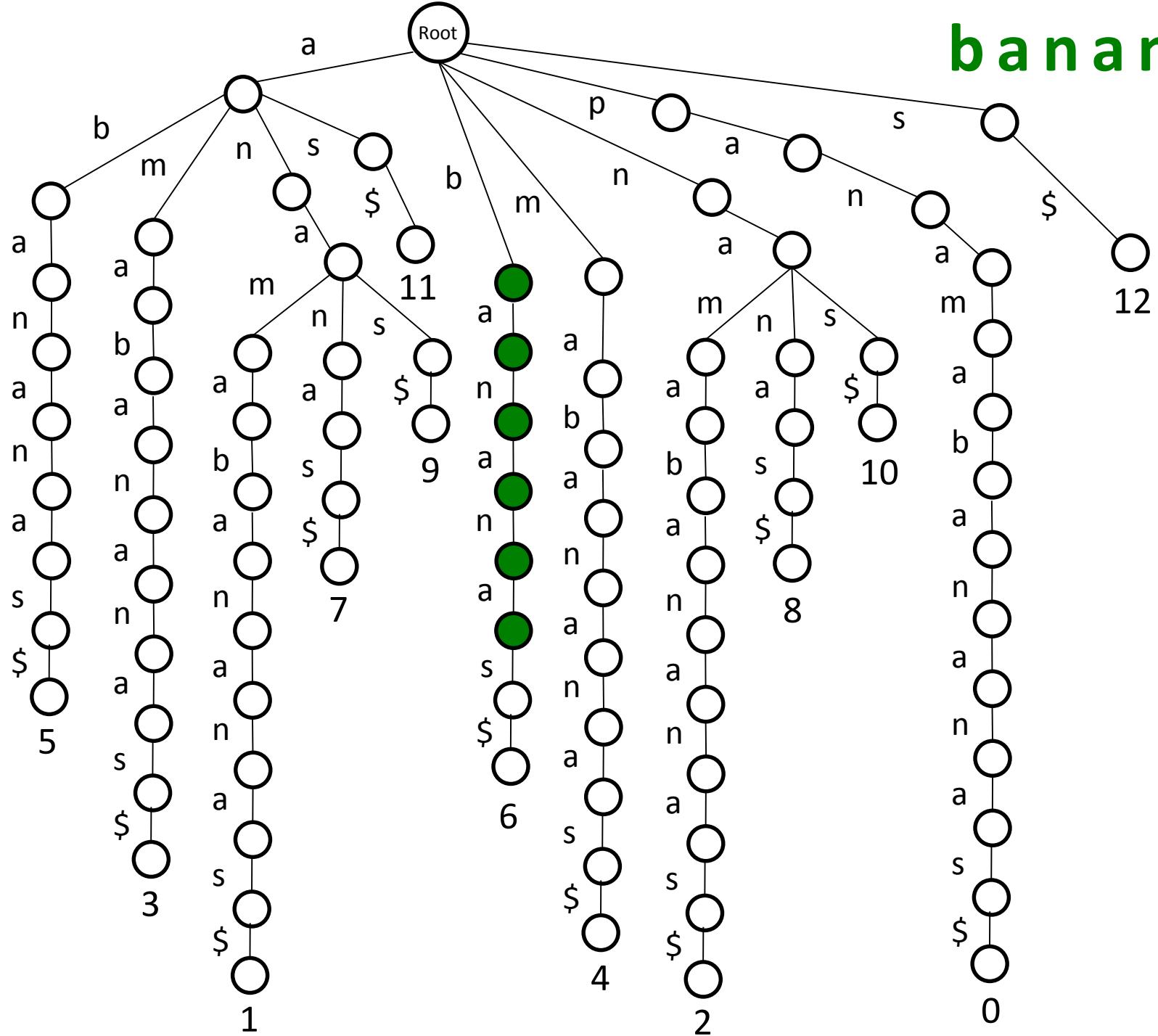


SuffixTrie(*Text*)

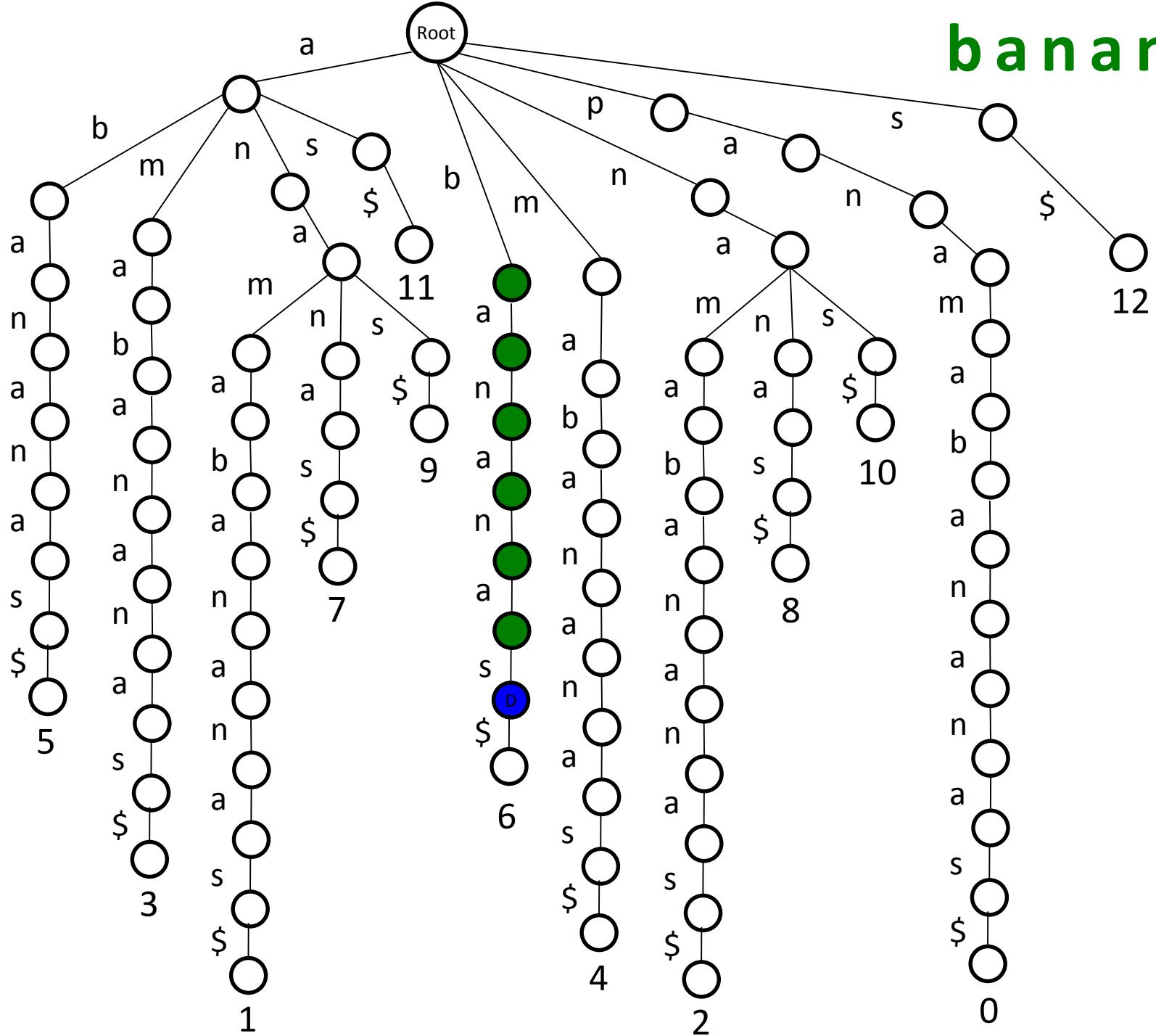
# Walking Down to the Leaves to Find Matches

- Once we find a match, we “walk down” to the leaf (or leaves) in order to find the starting position of the match.

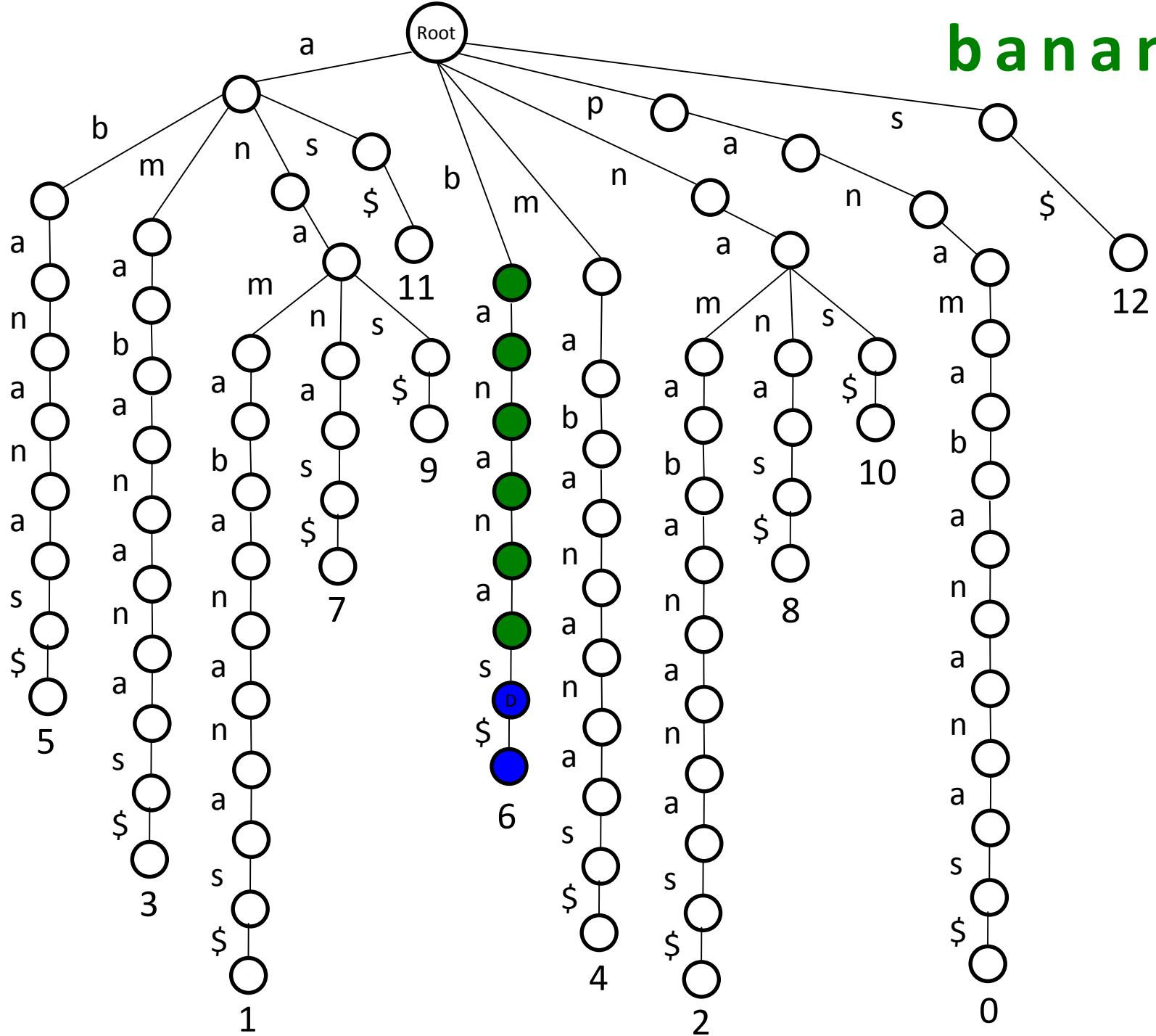
**banana**

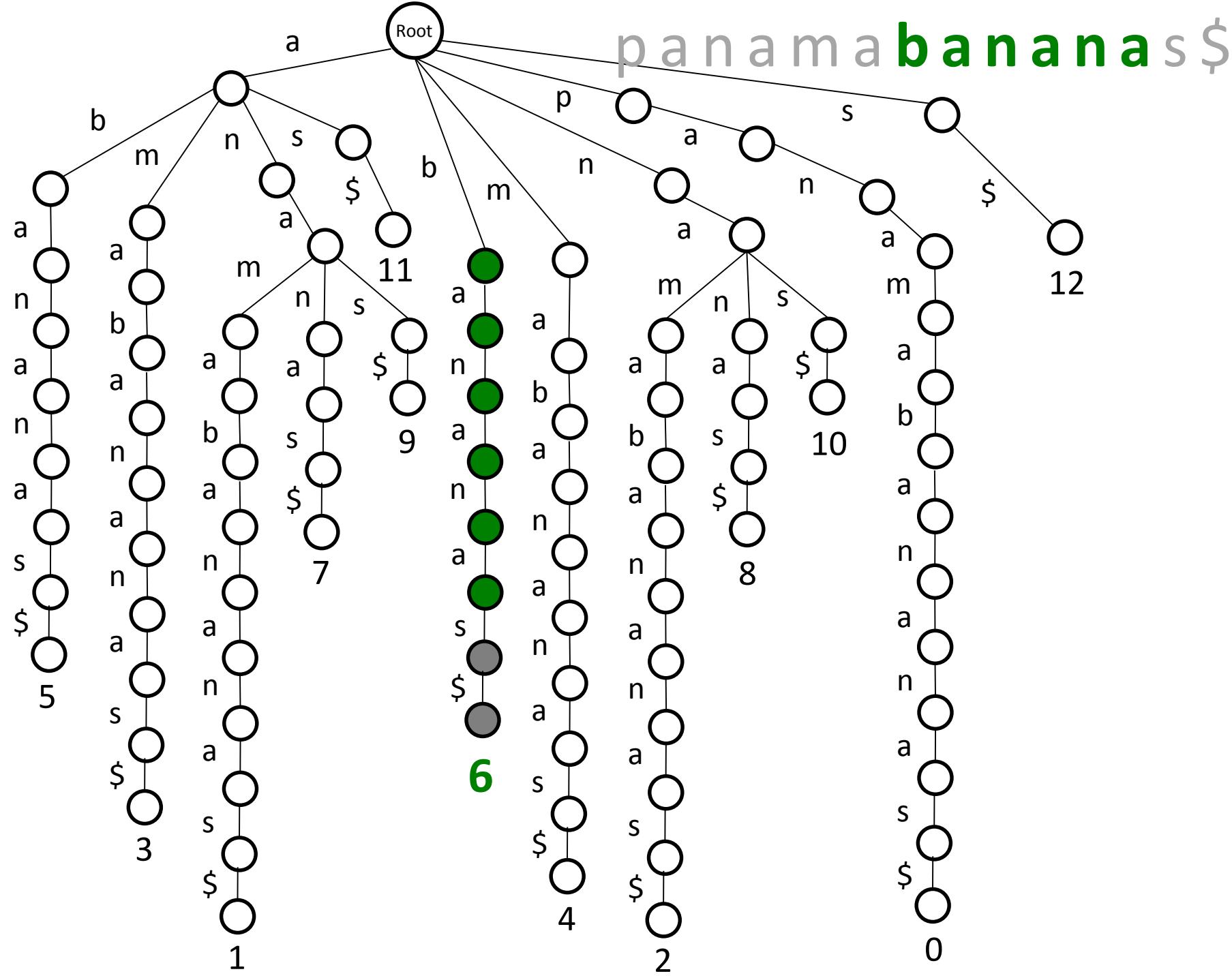


# banana

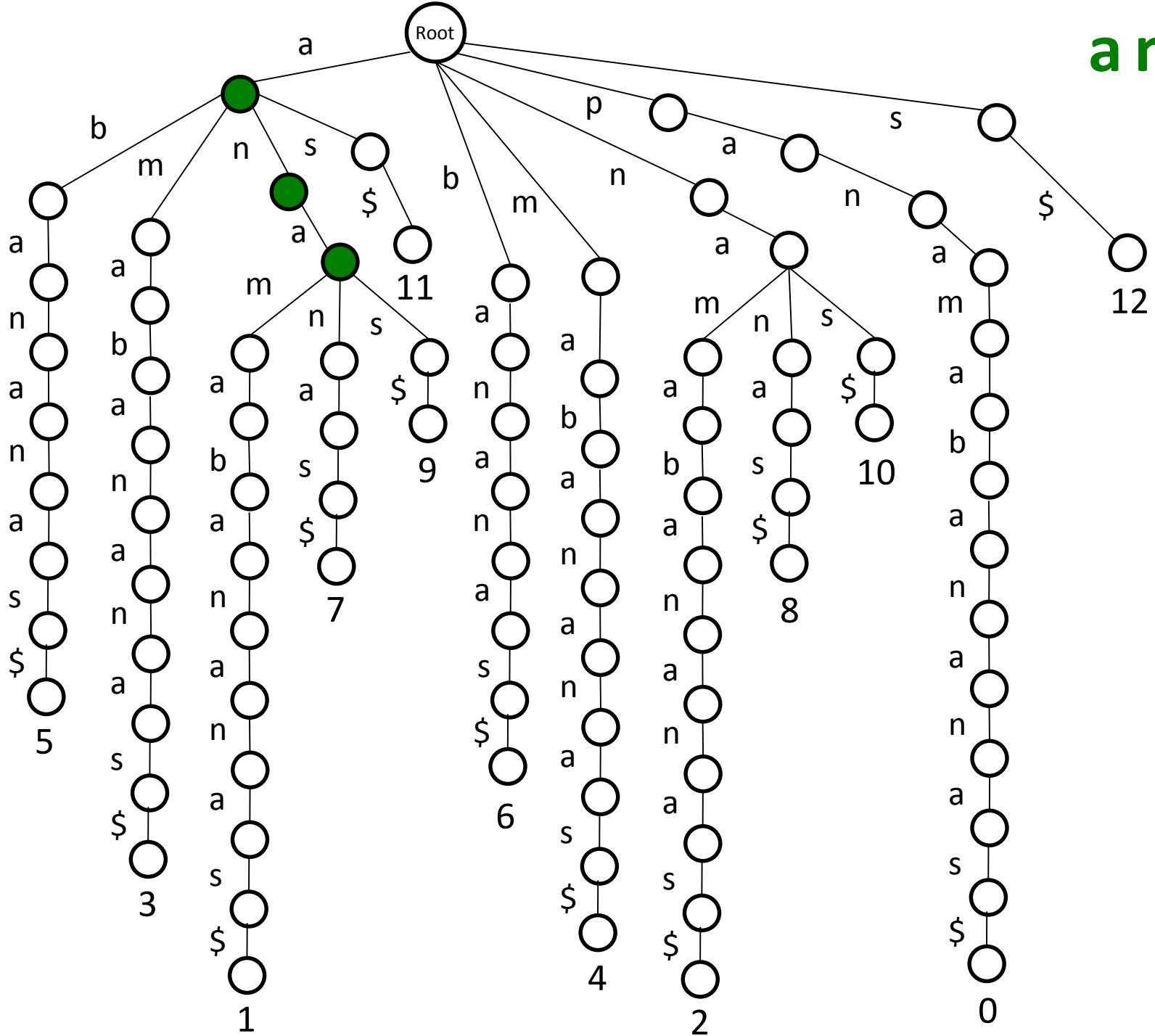


# banana

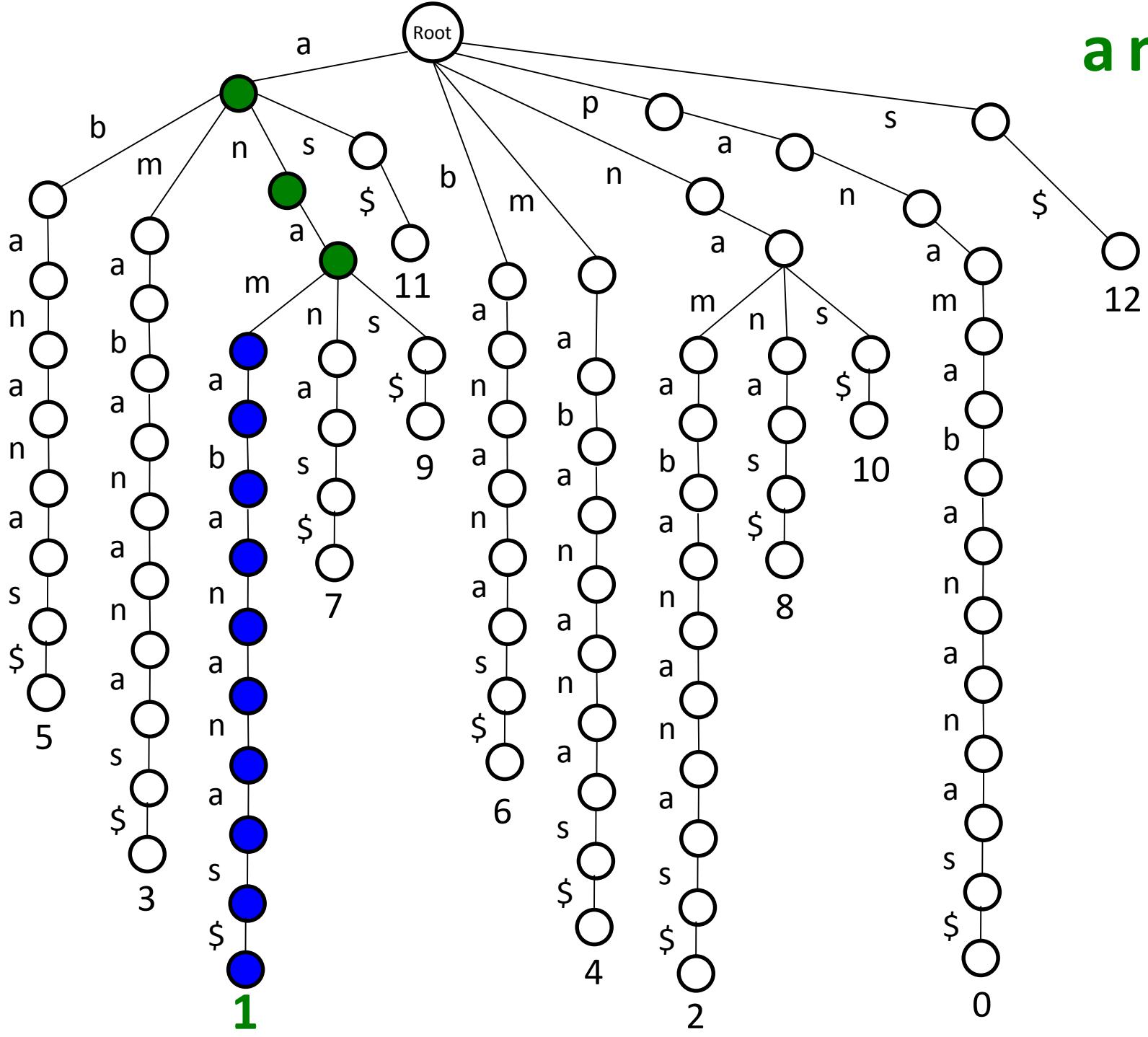




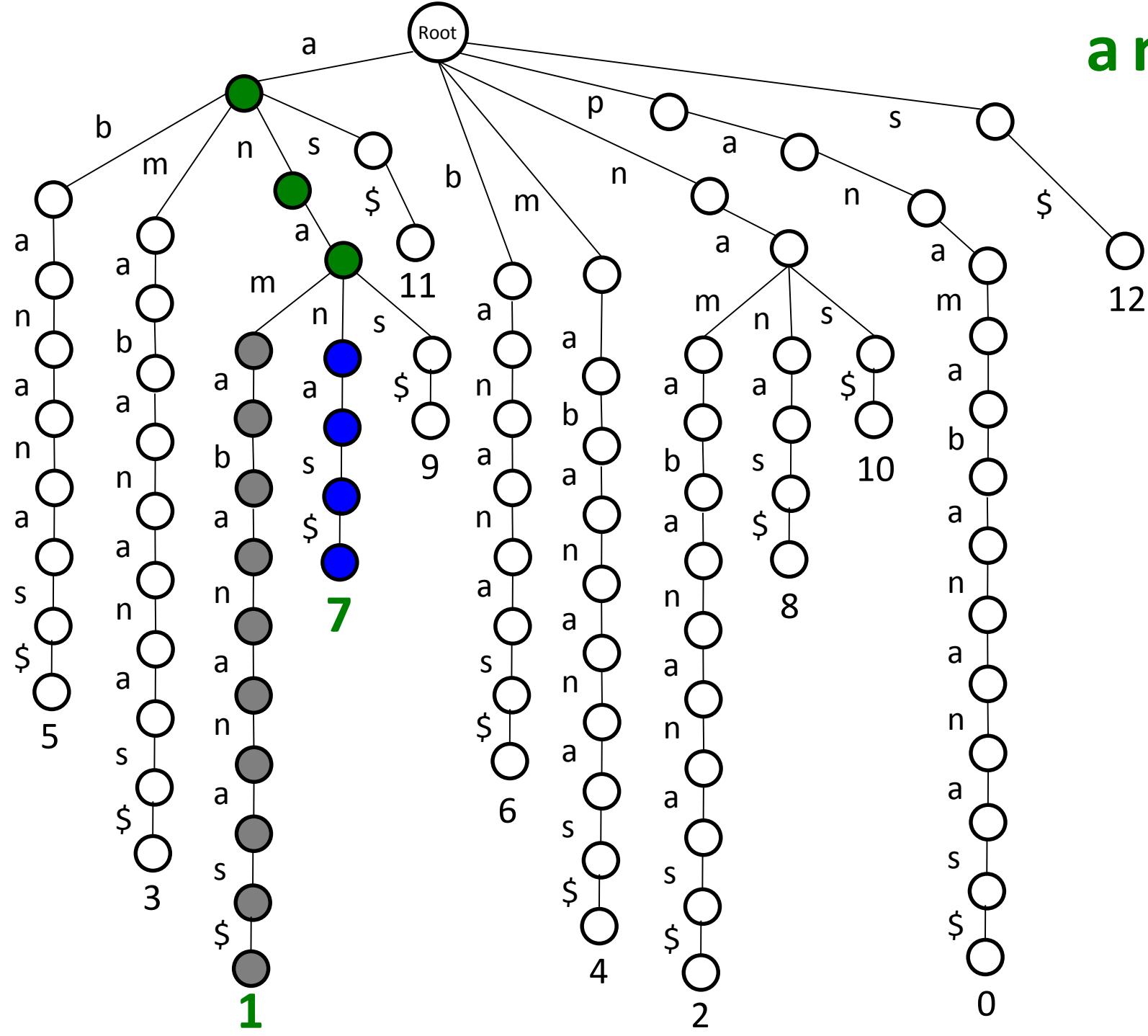
a n a



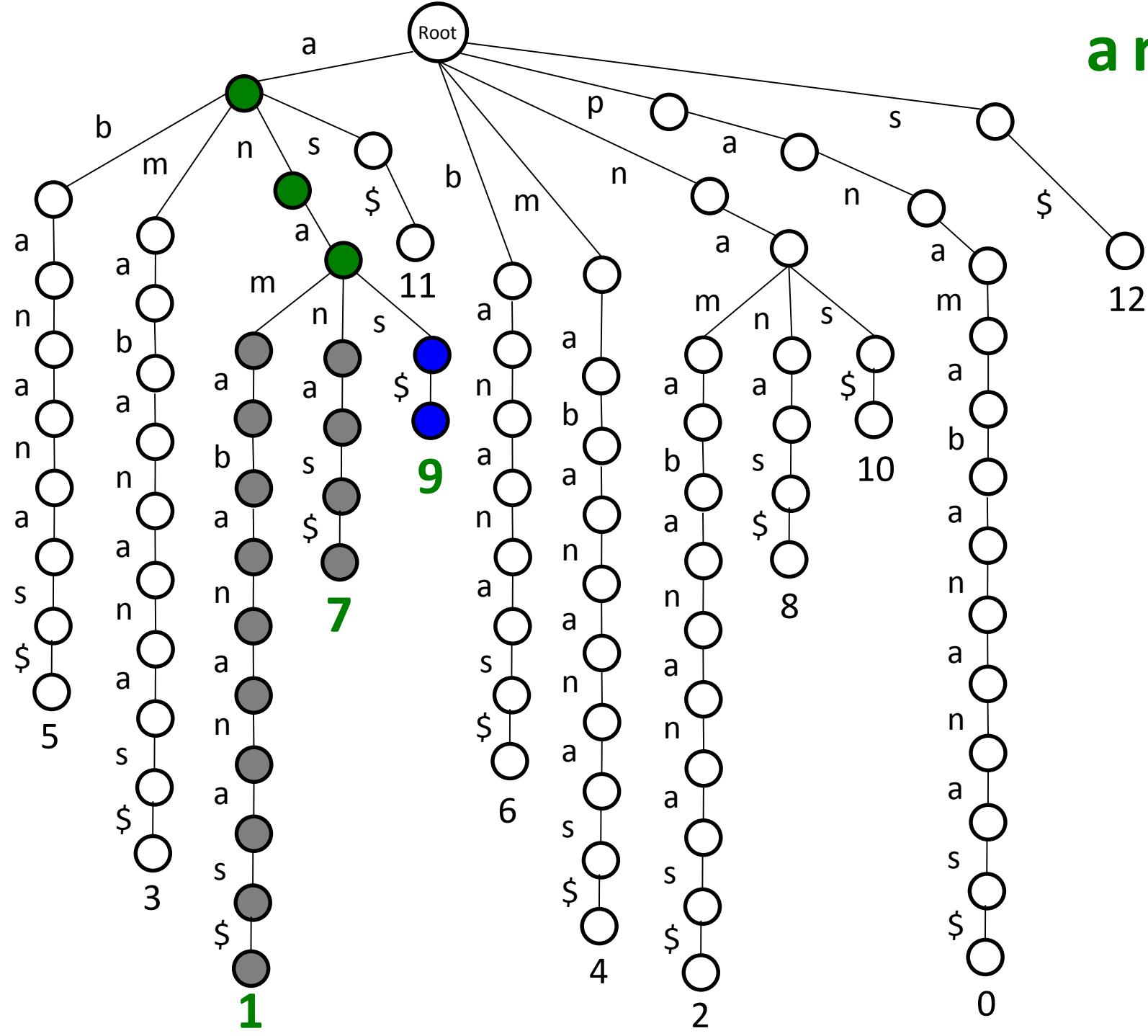
a n a

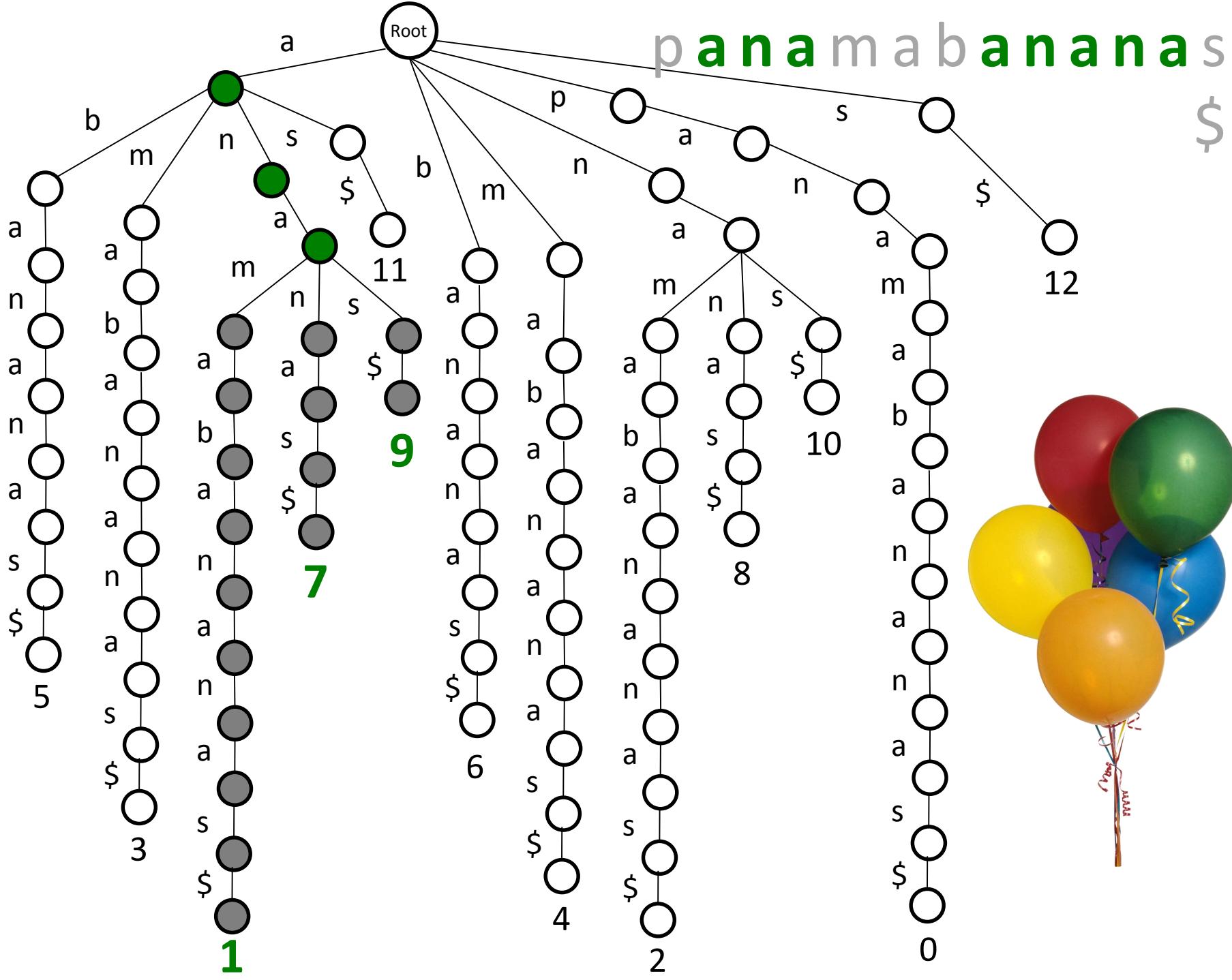


a n a



a n a





# Memory Footprint of Suffix Trie

The suffix trie is formed from  $|Text|$  suffixes with total length:

$$|Text|^*(|Text|-1)/2$$

## For human genome:

- $|Text| \approx 3 * 10^9$

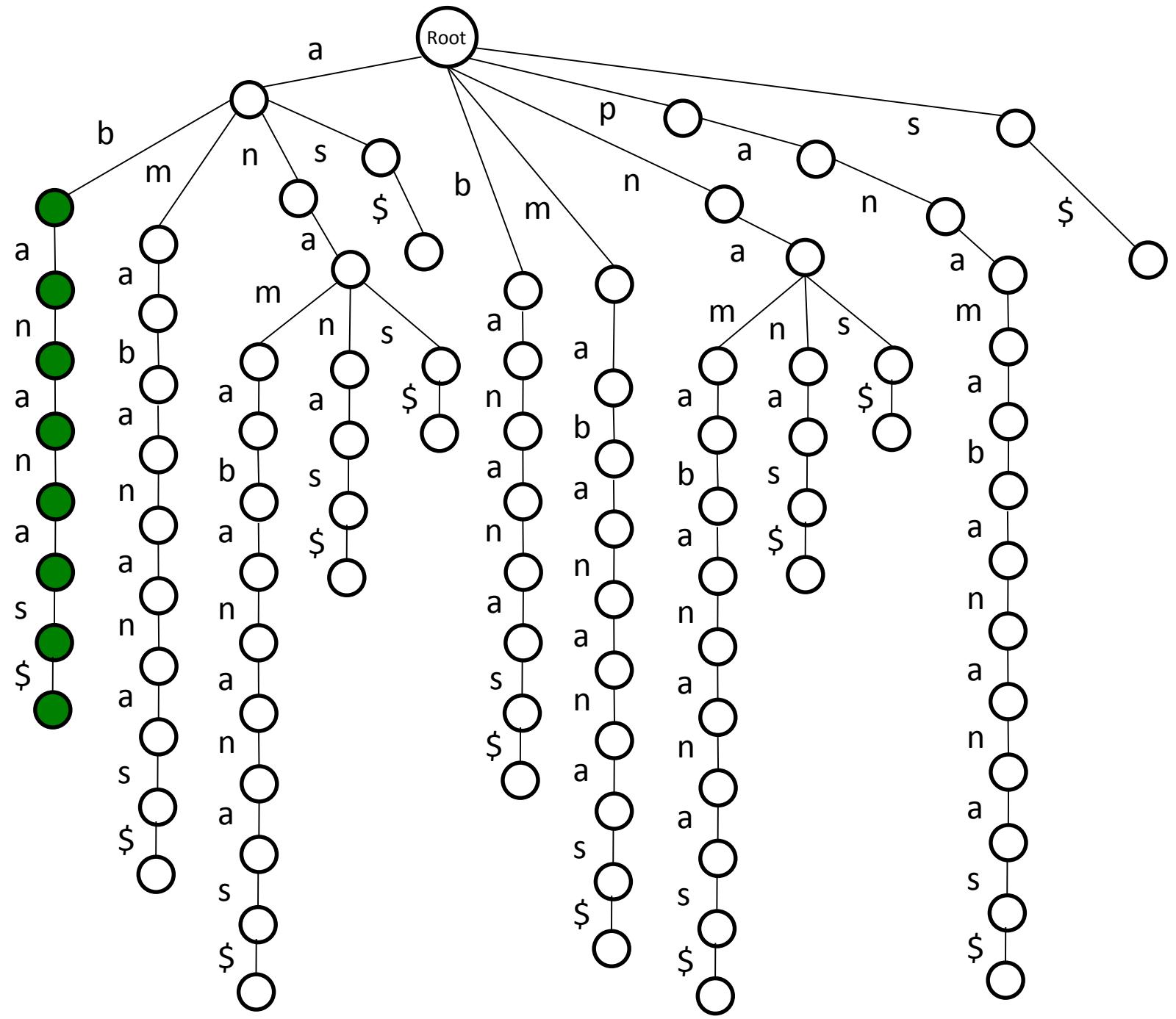
## | *Text* | symbols

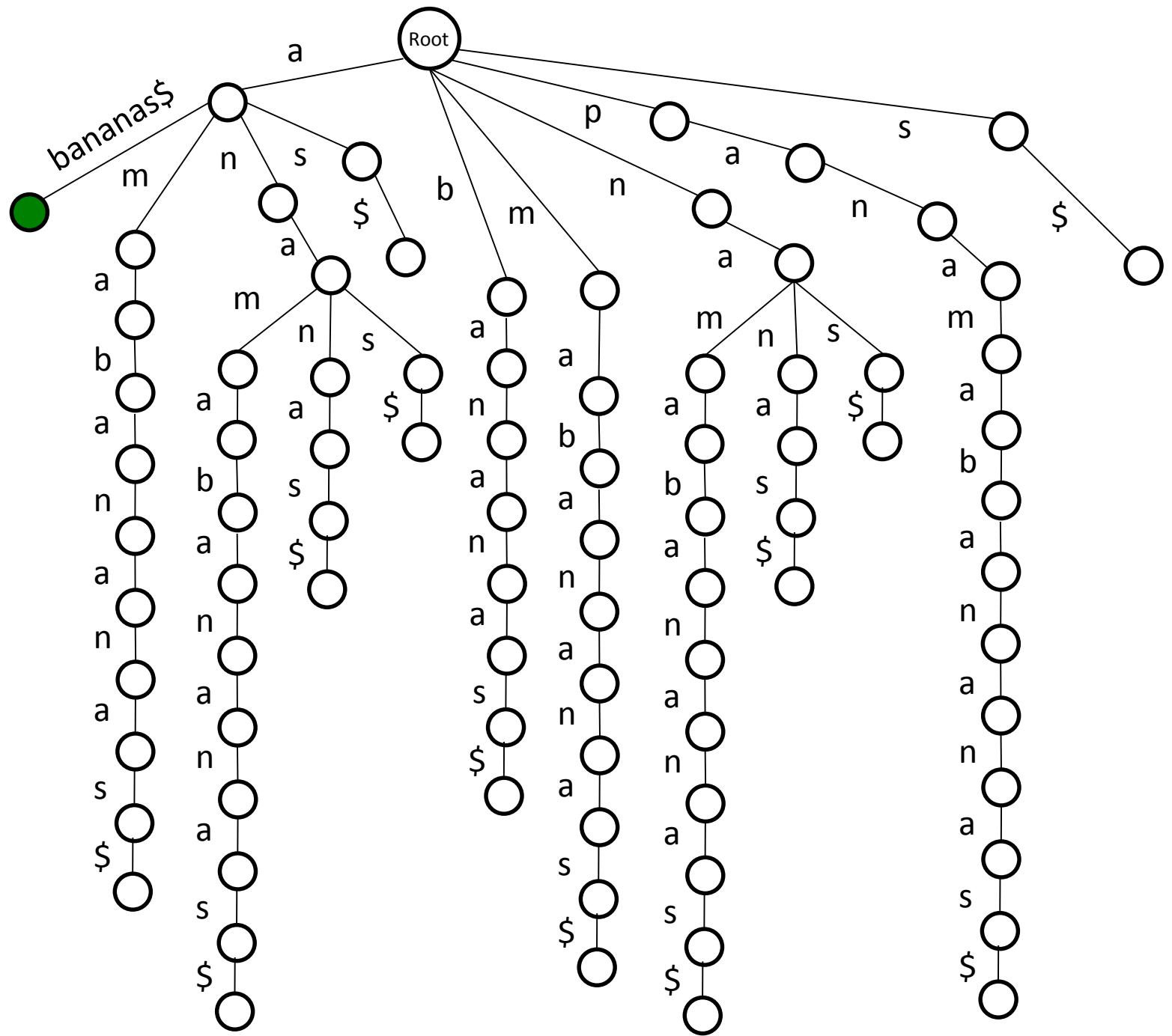
| *Text* | suffixes

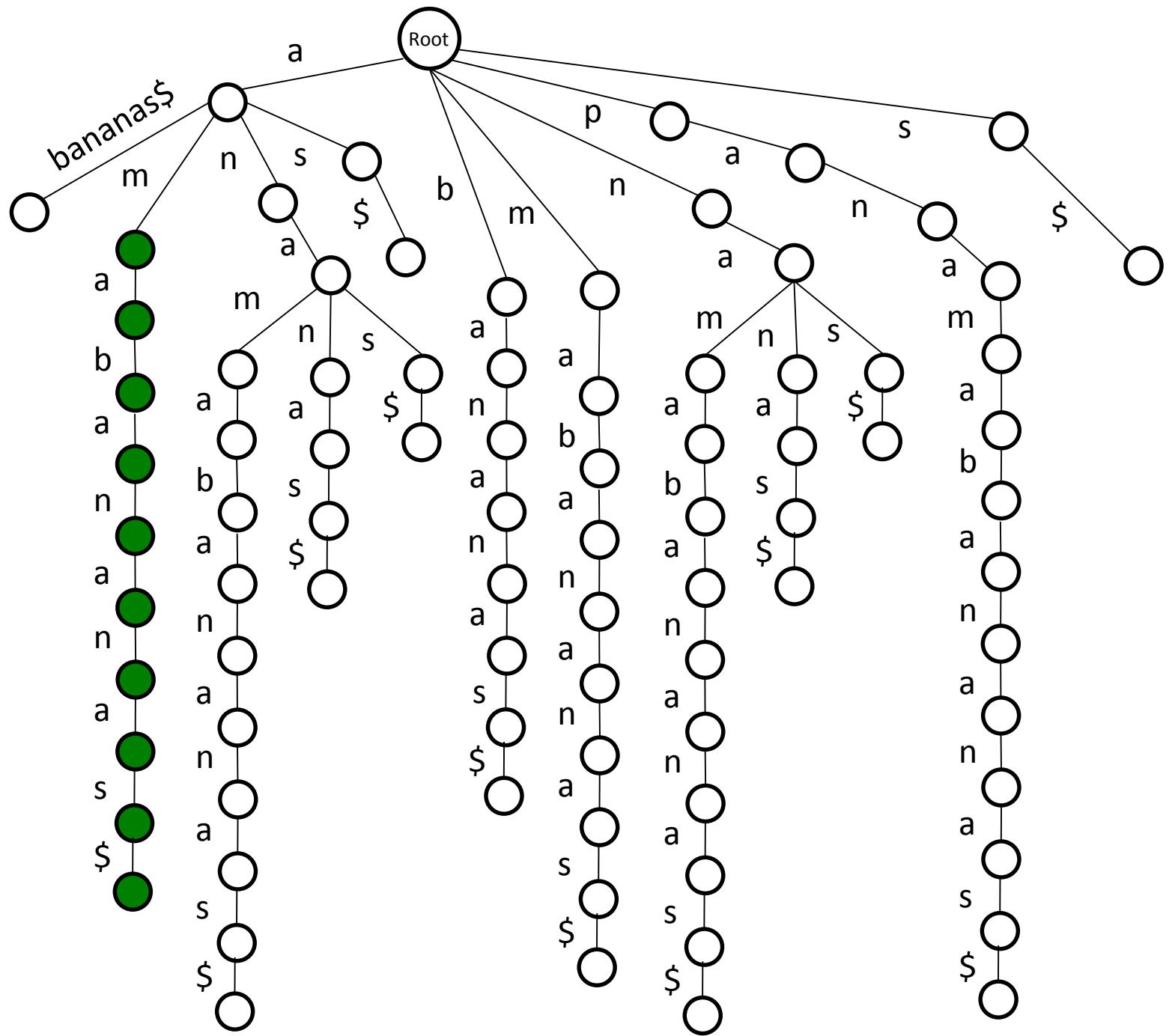


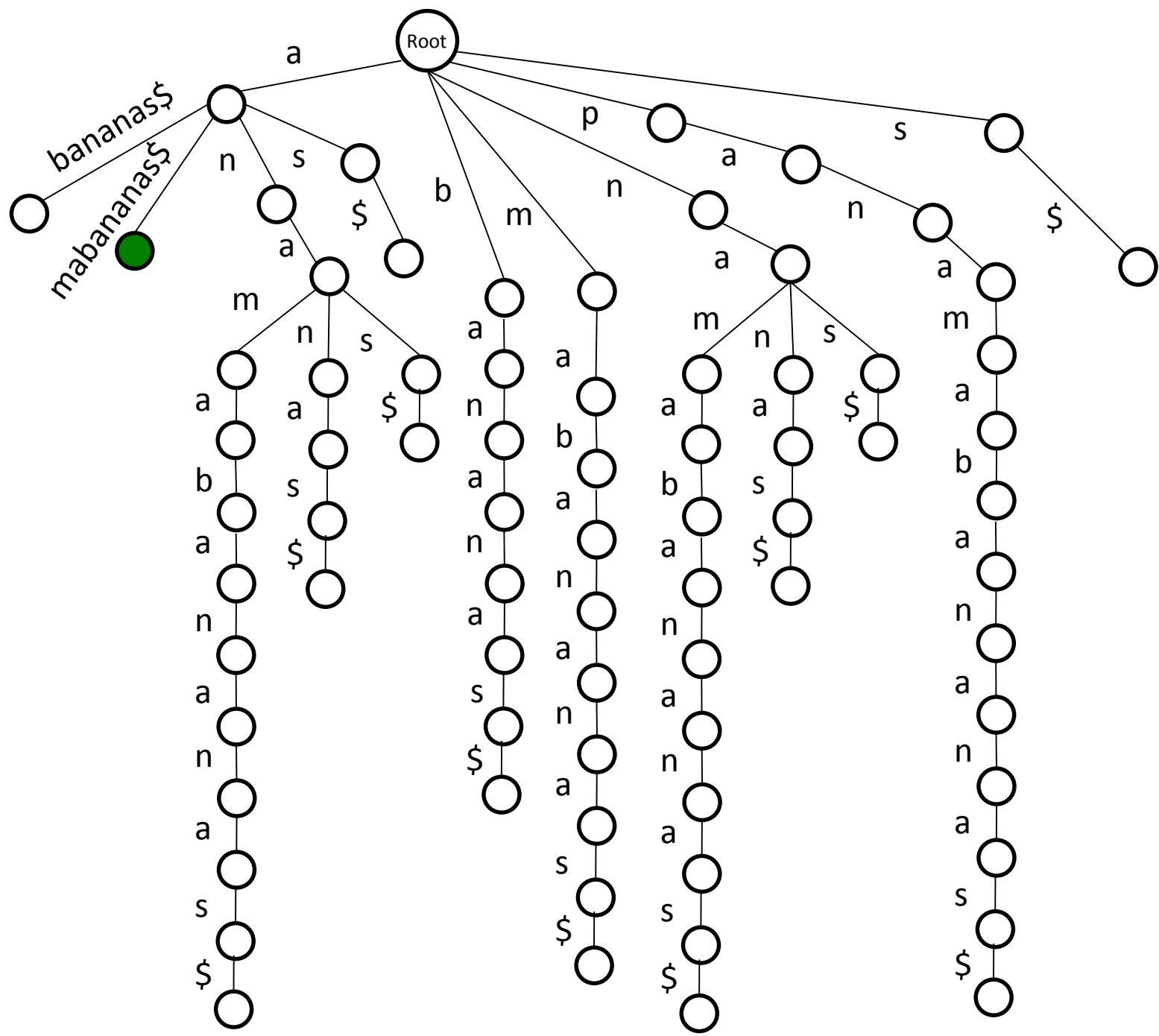
# Outline

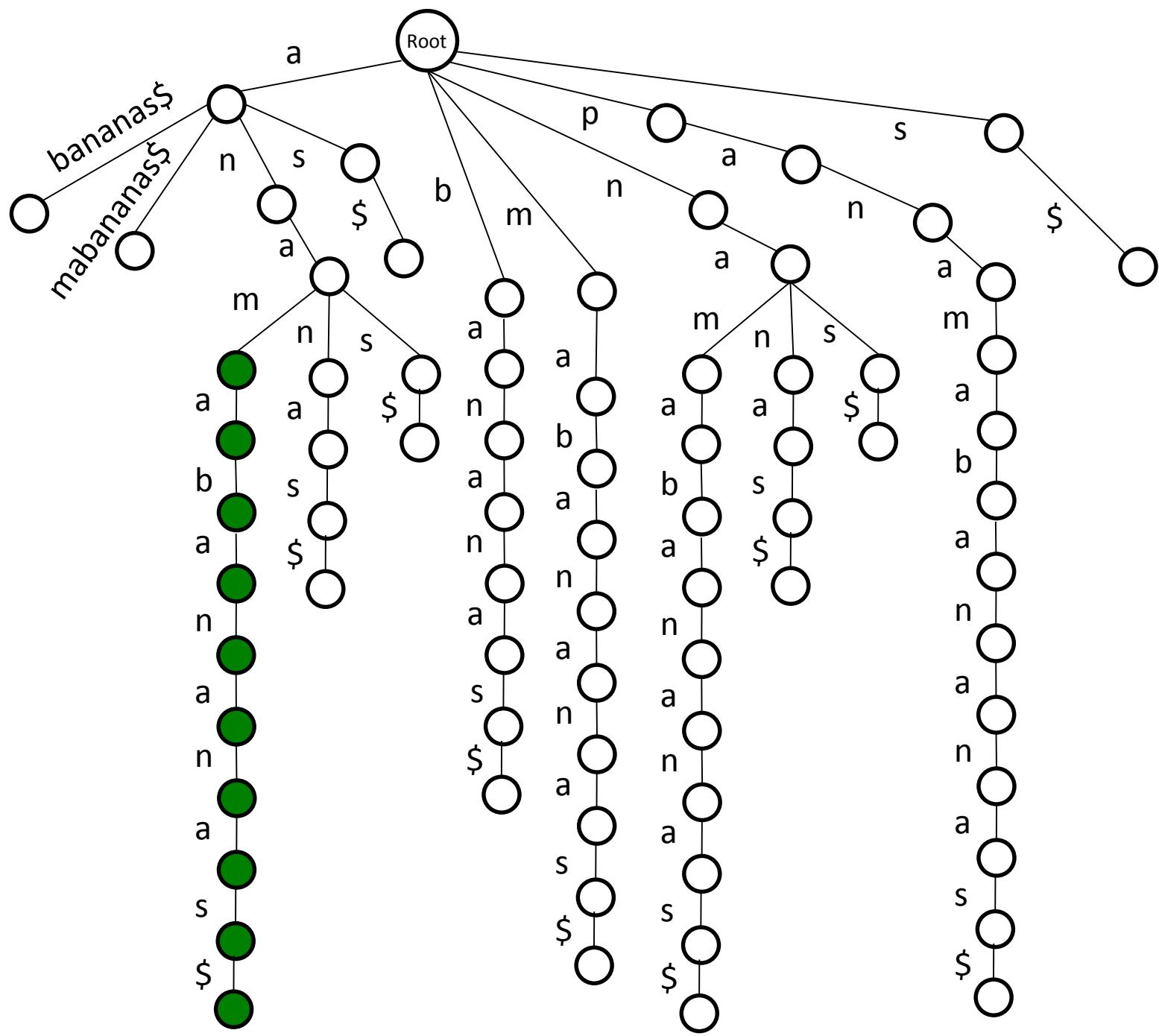
- From Genome Sequencing to Pattern Matching
- Brute Force Approach to Pattern Matching
- Herding Patterns into Trie
- Herding Text into Suffix Trie
- **From Suffix Tries to Suffix Trees**

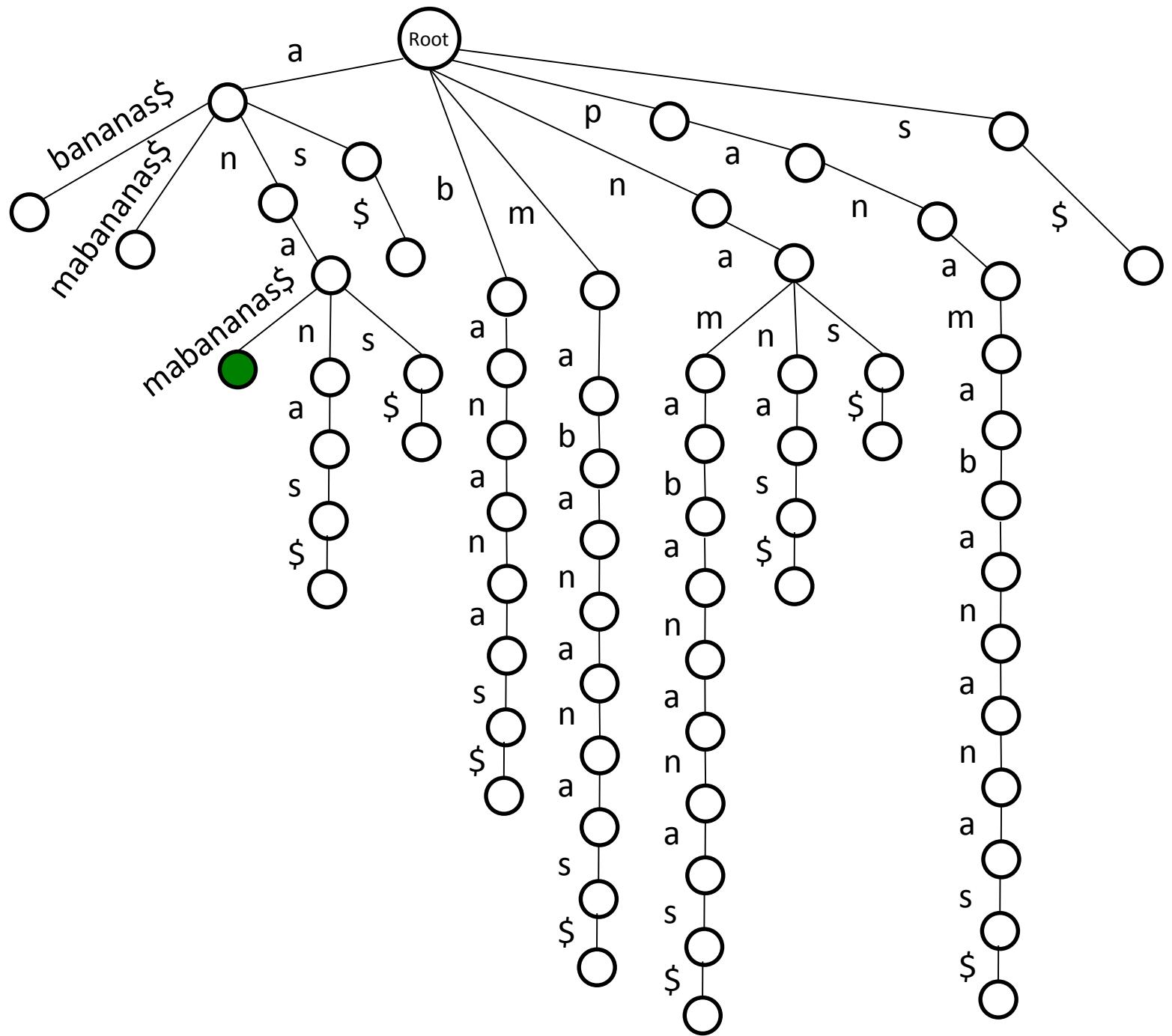


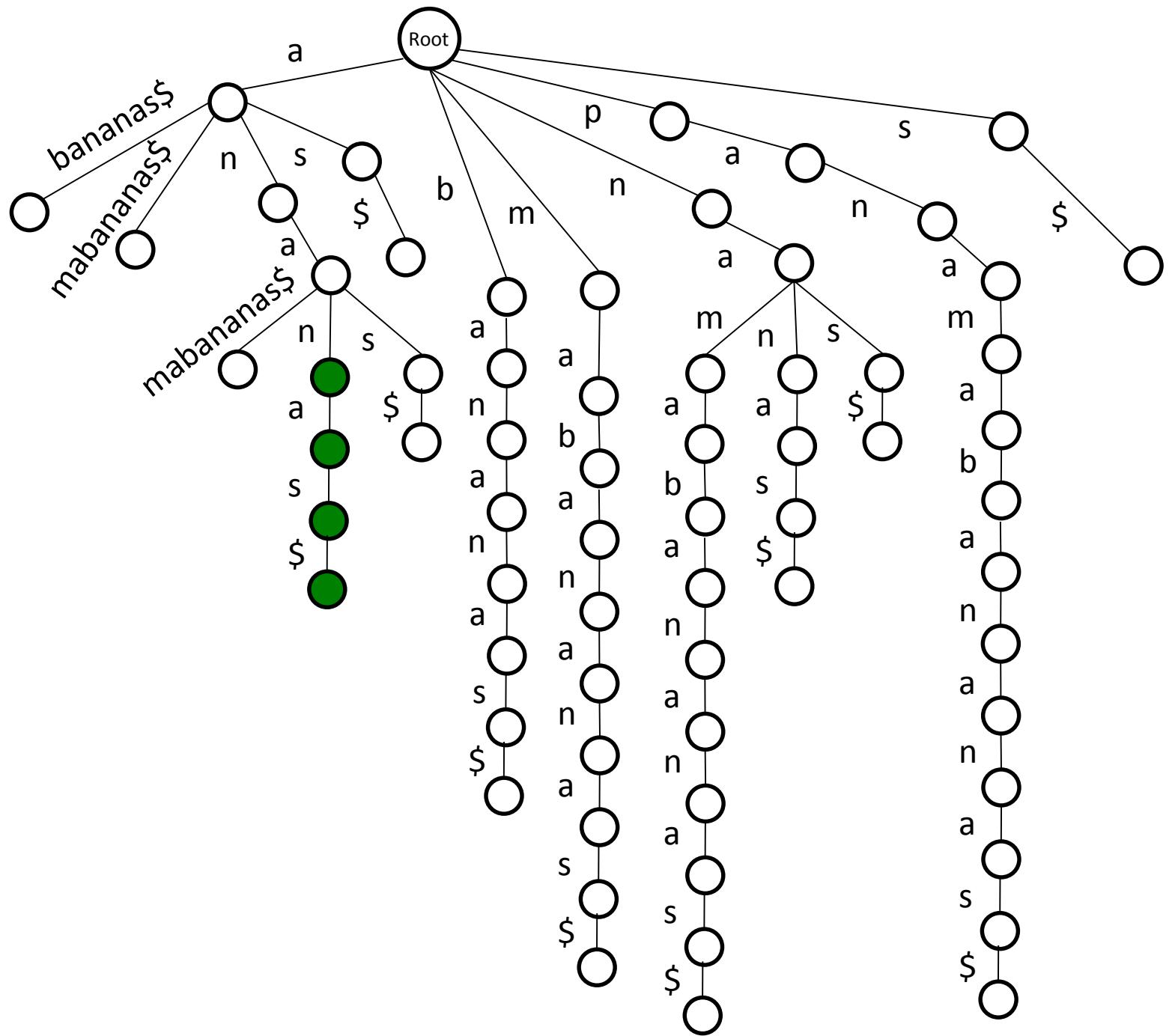


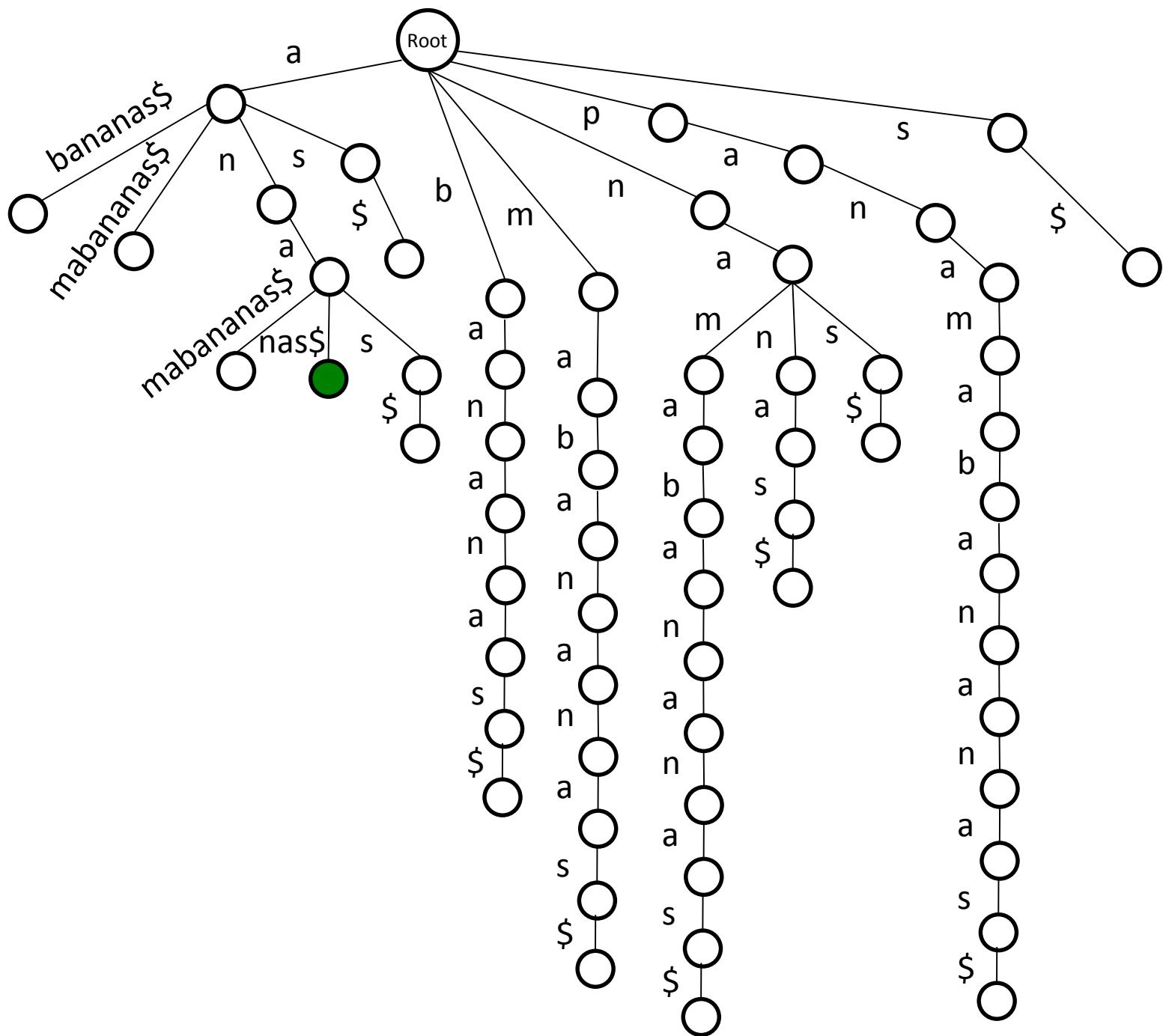


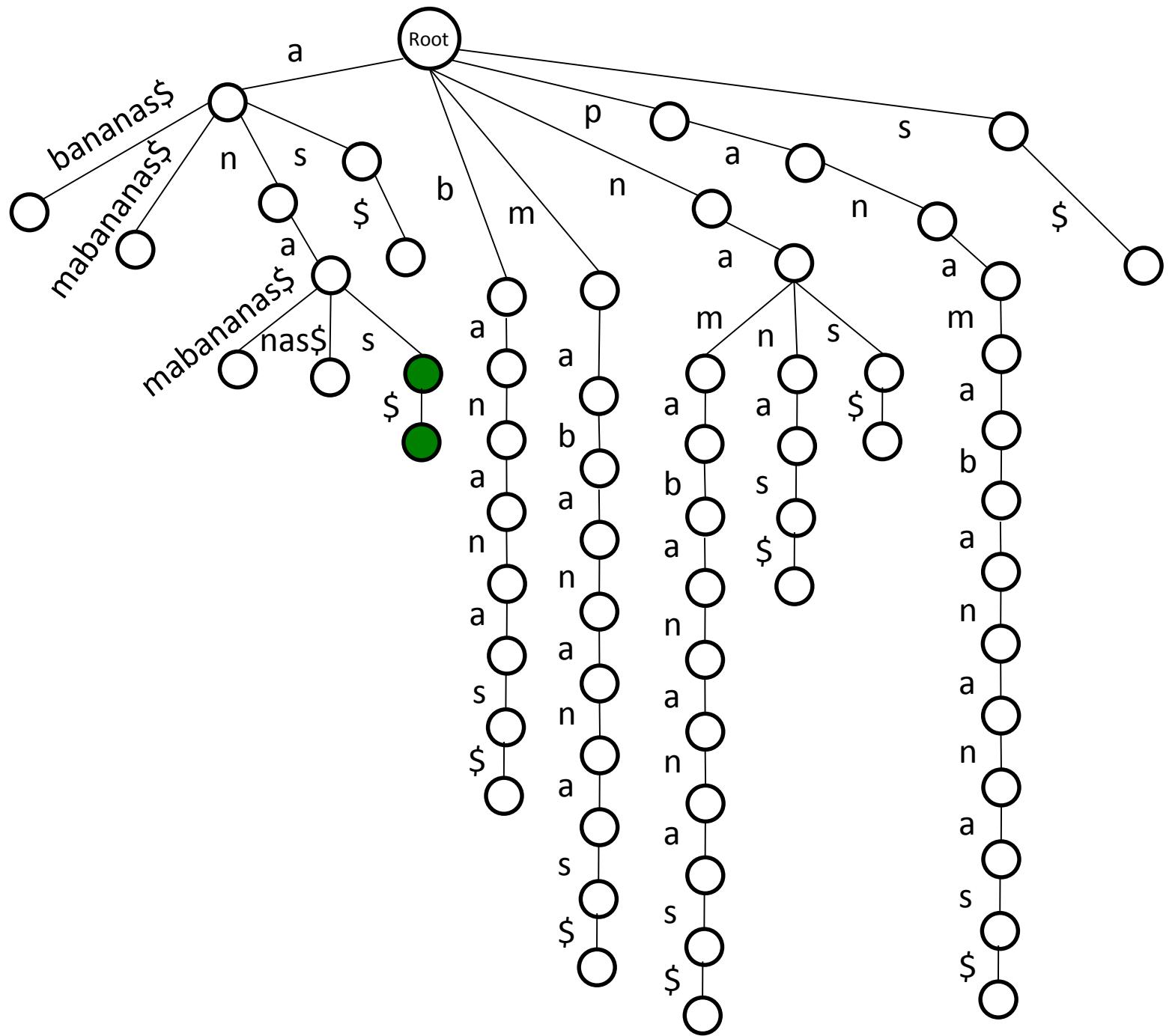


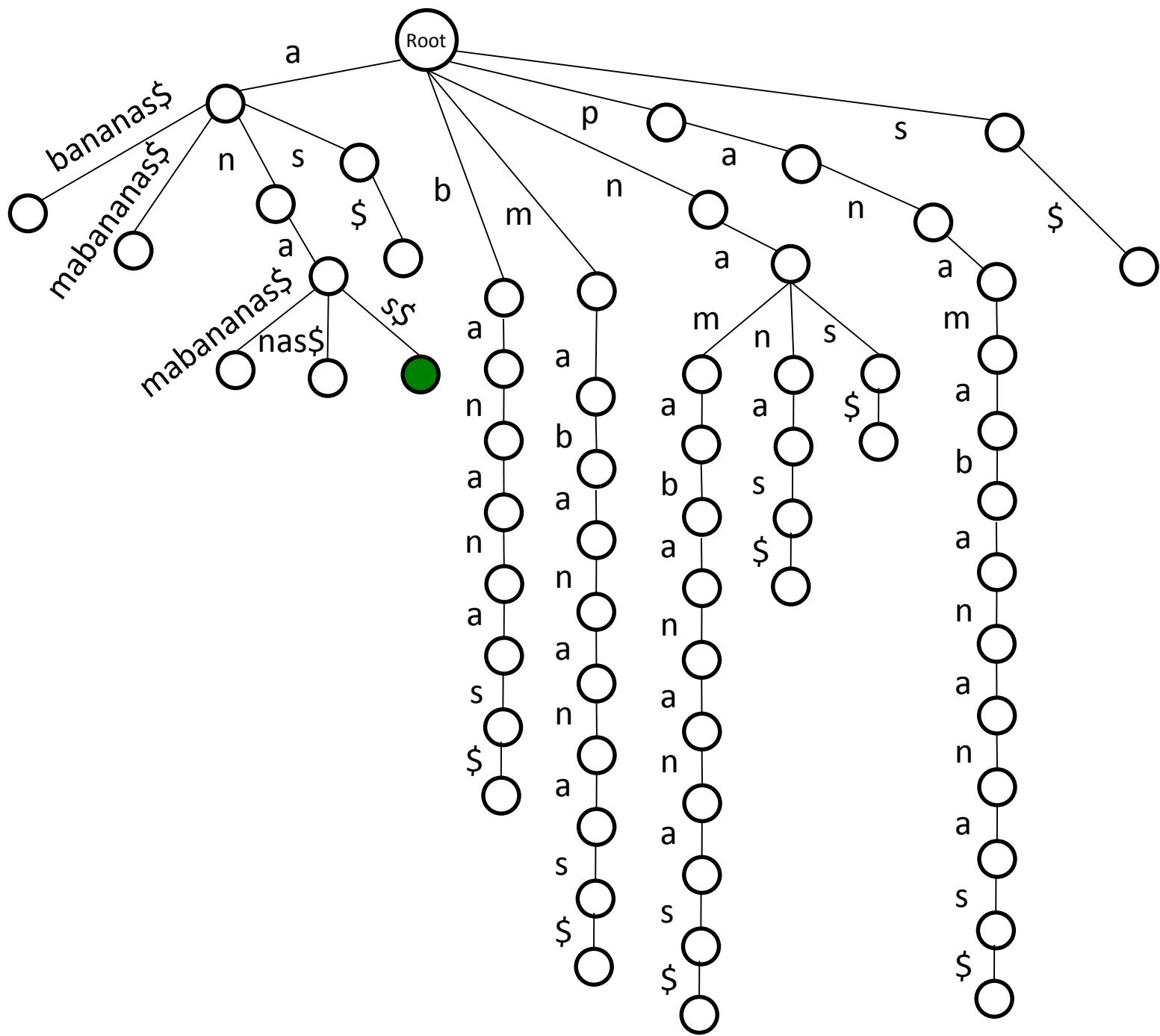


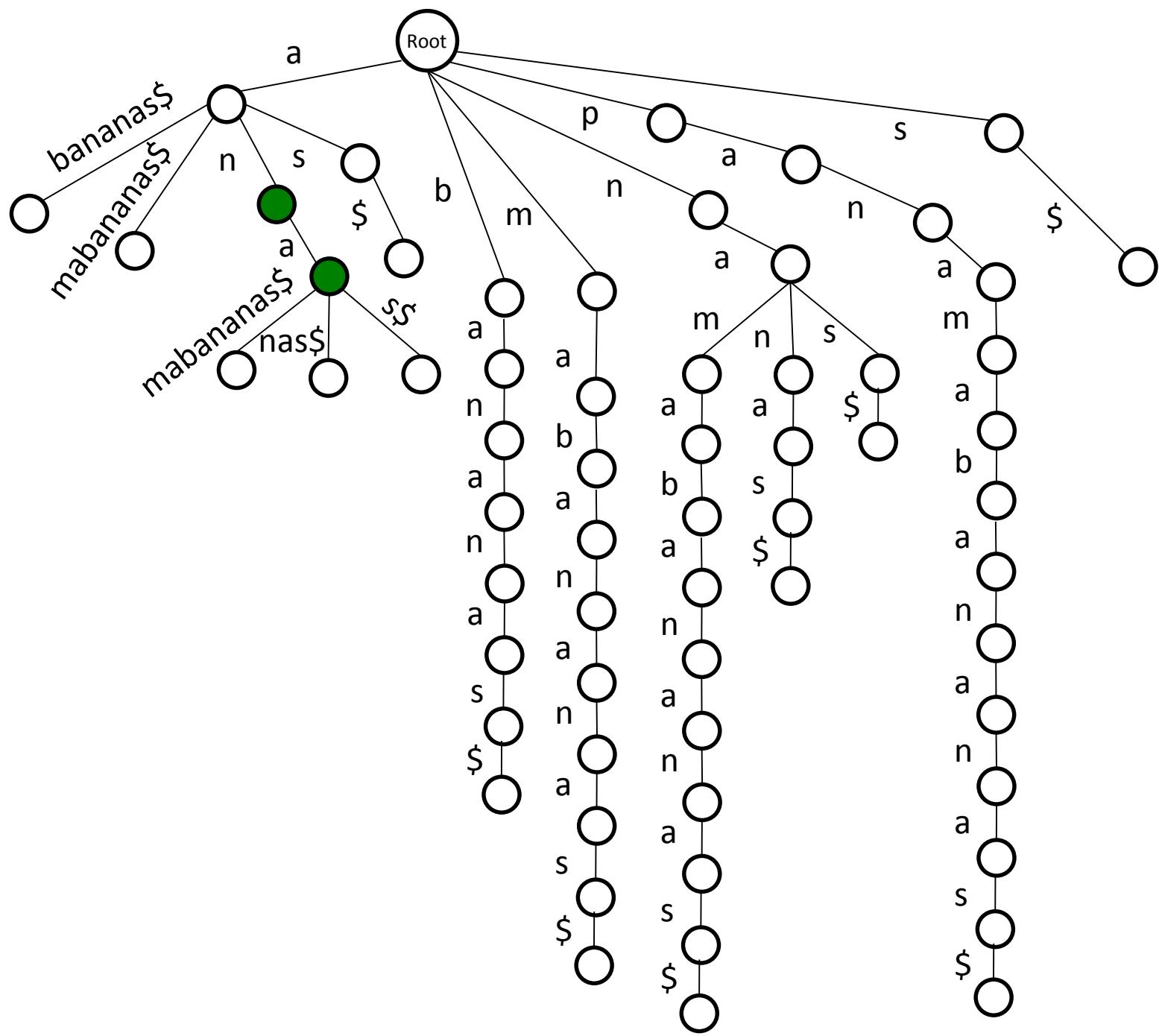


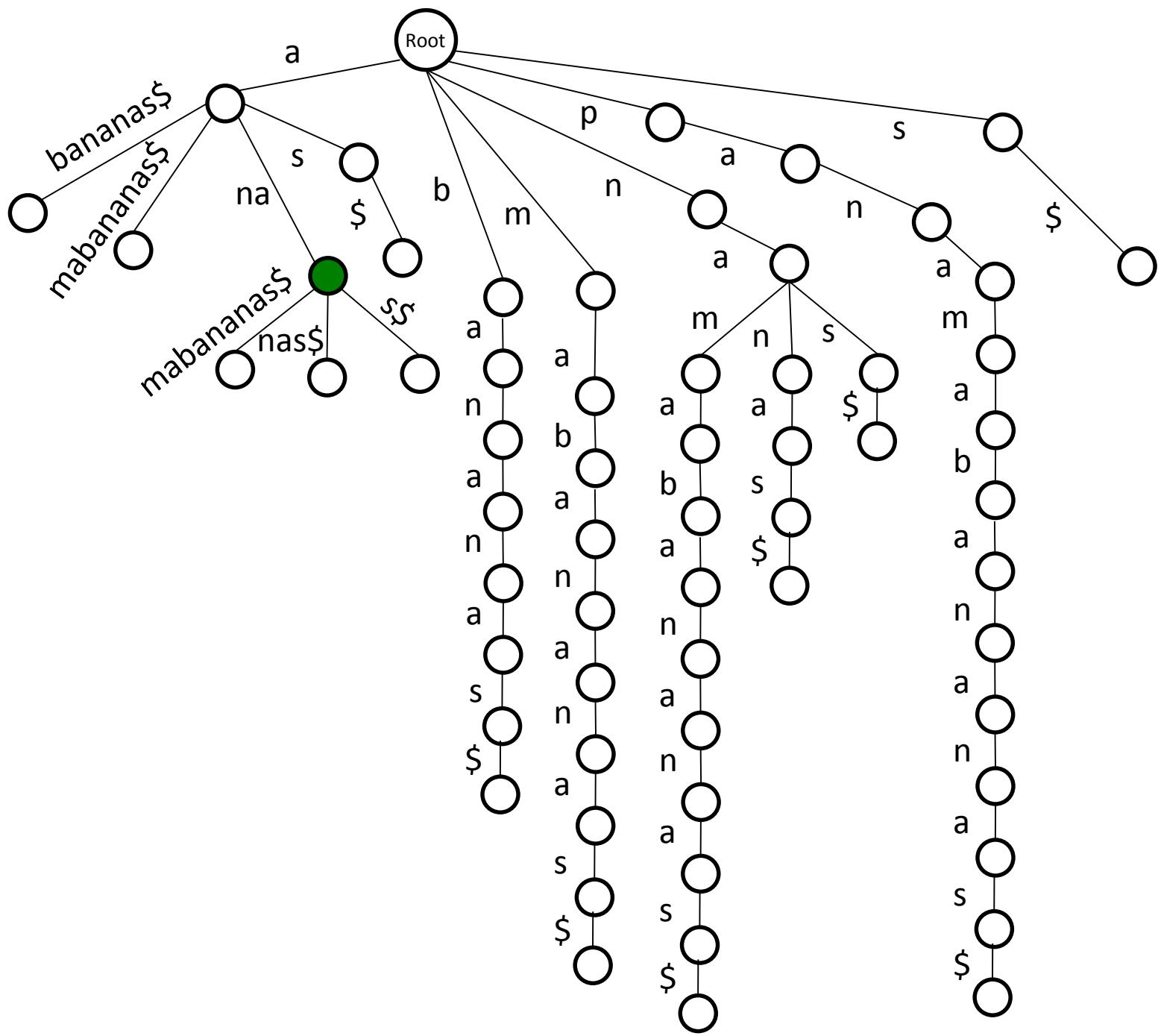


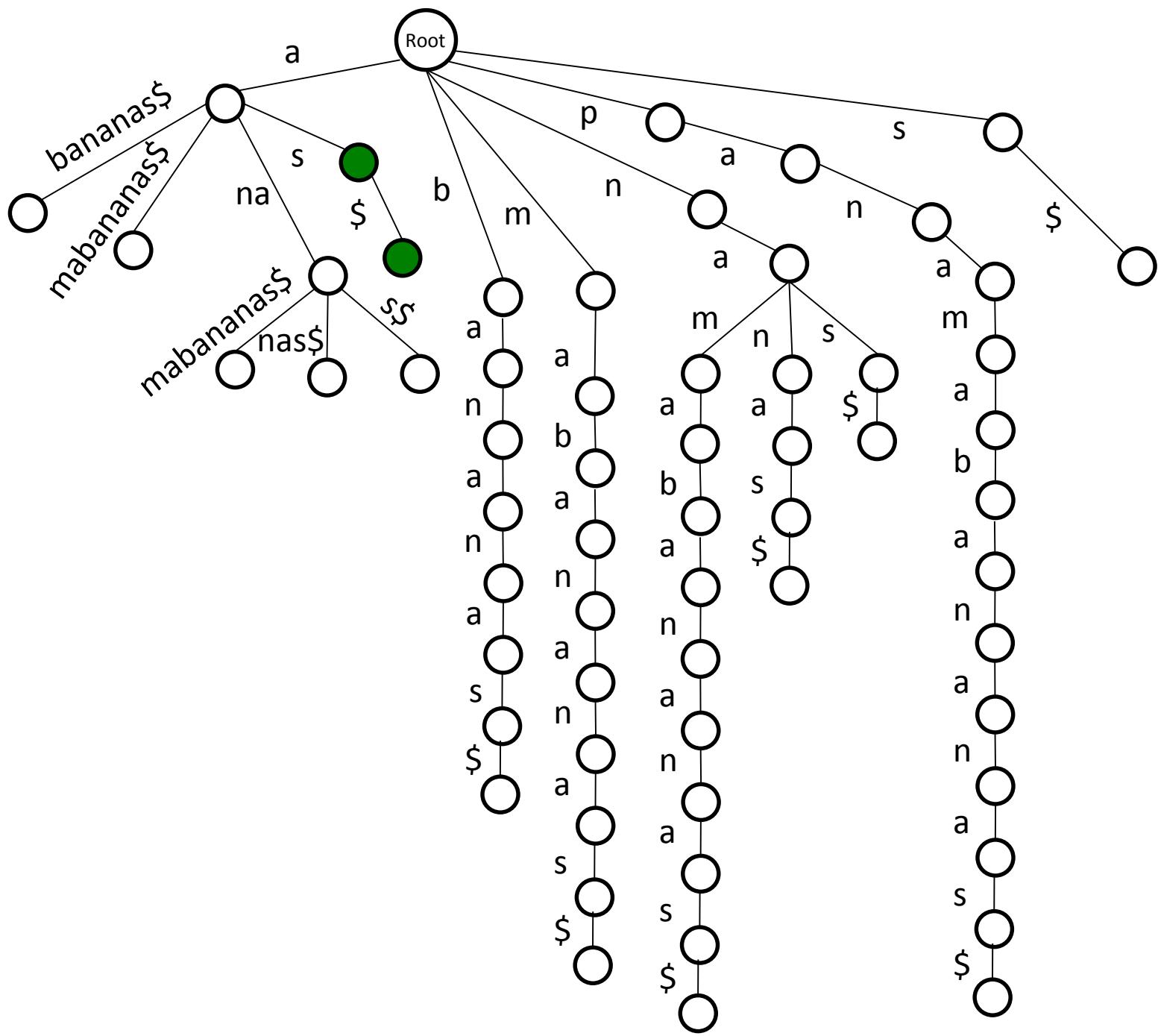


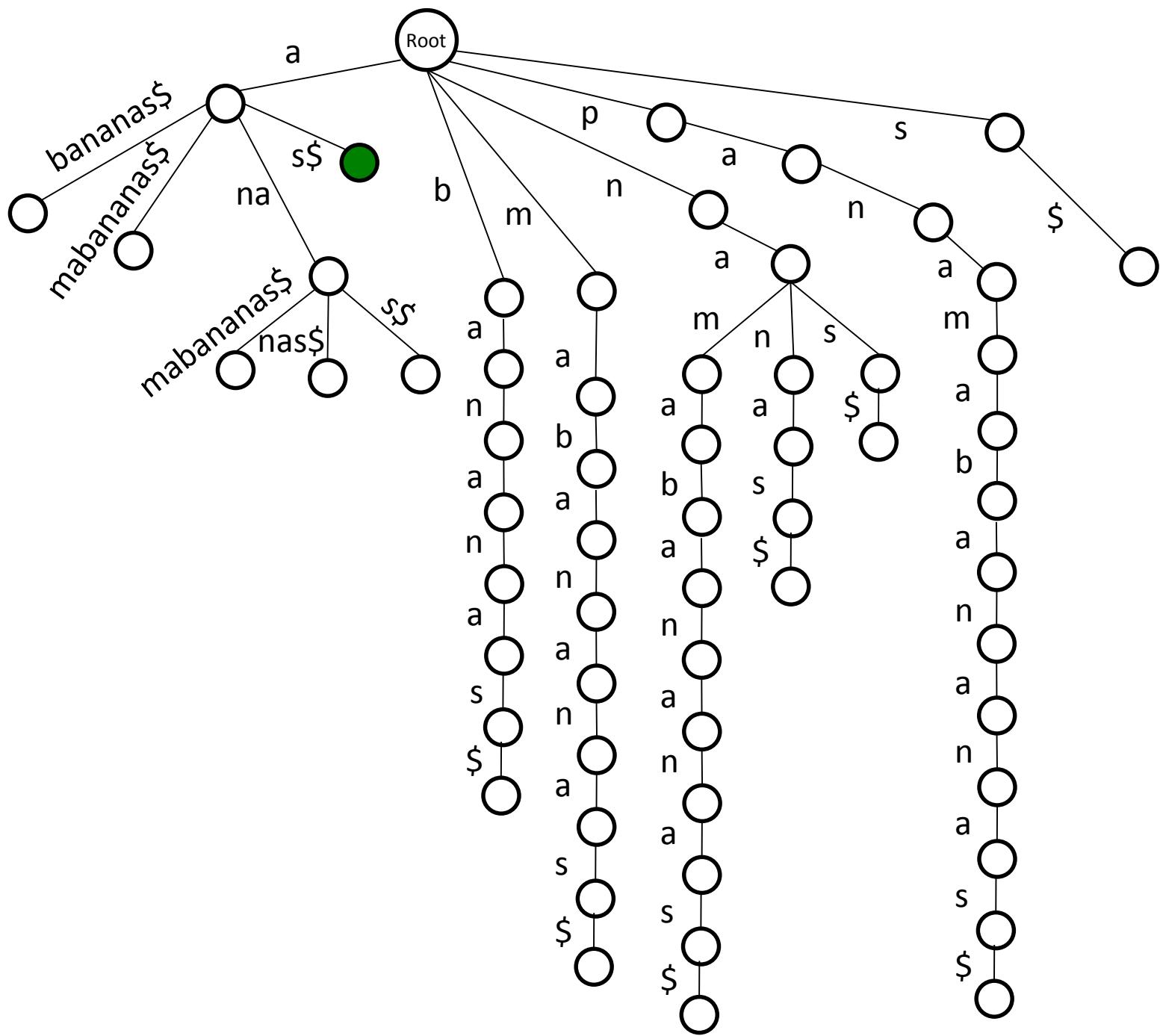


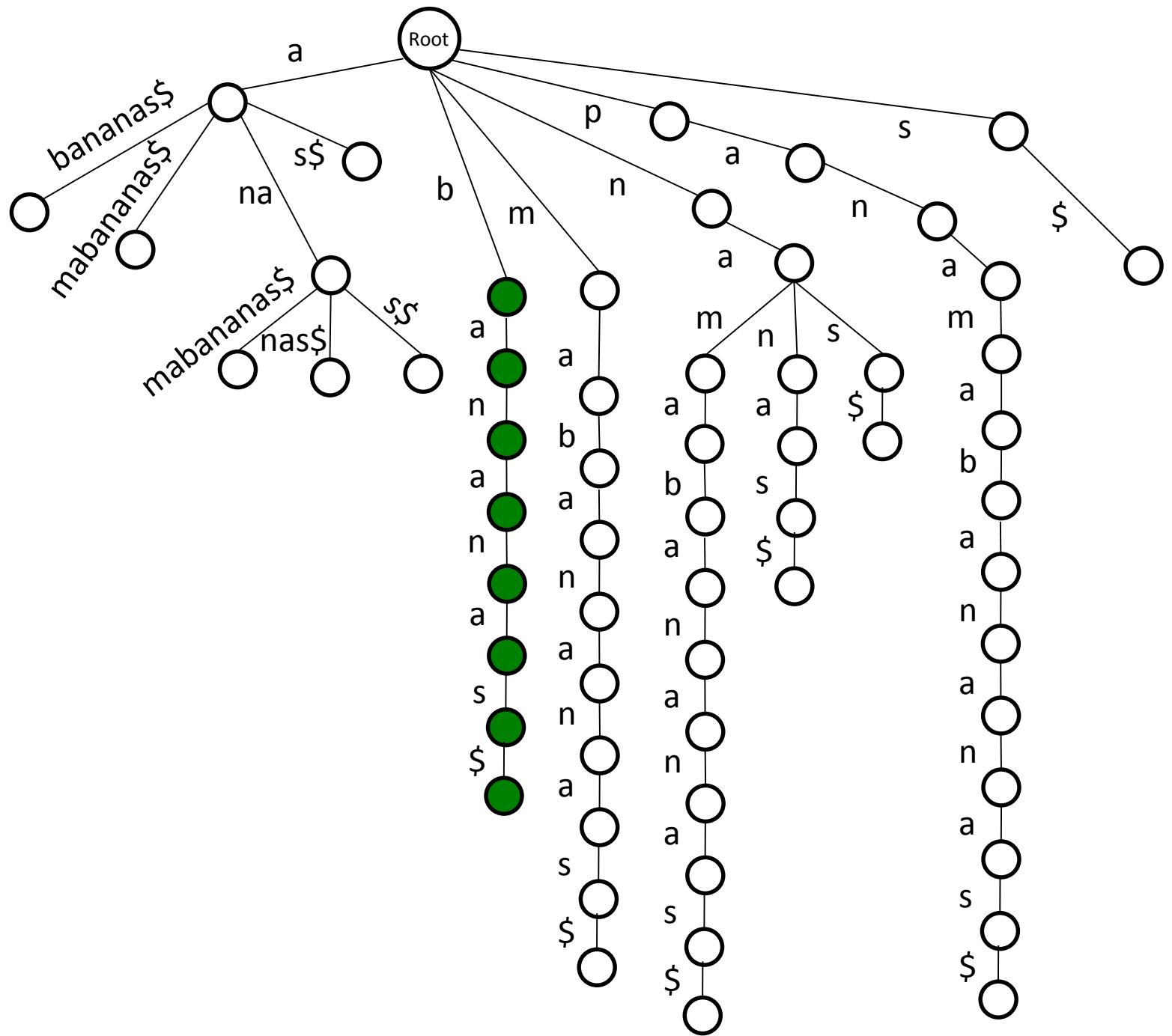


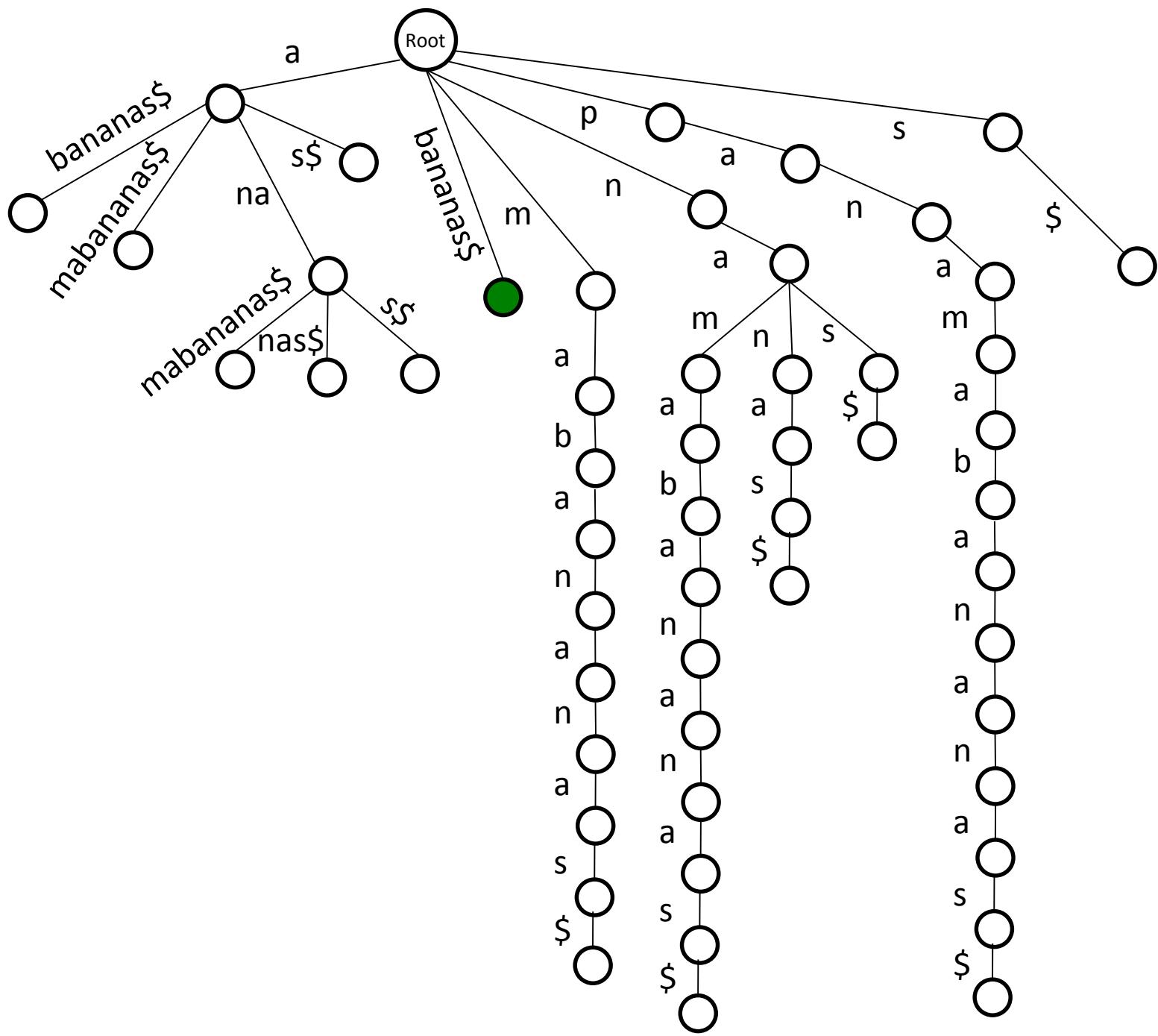


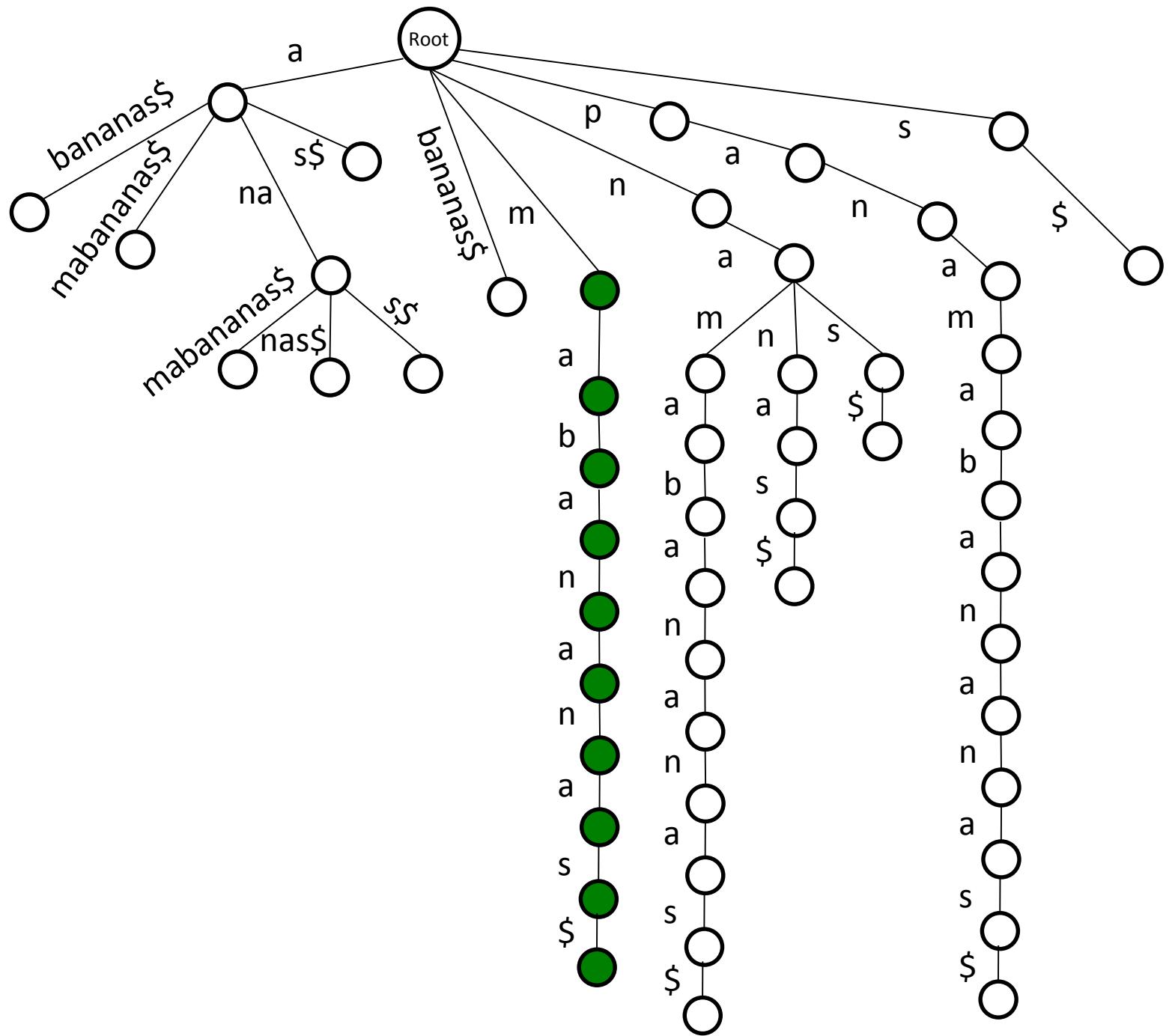


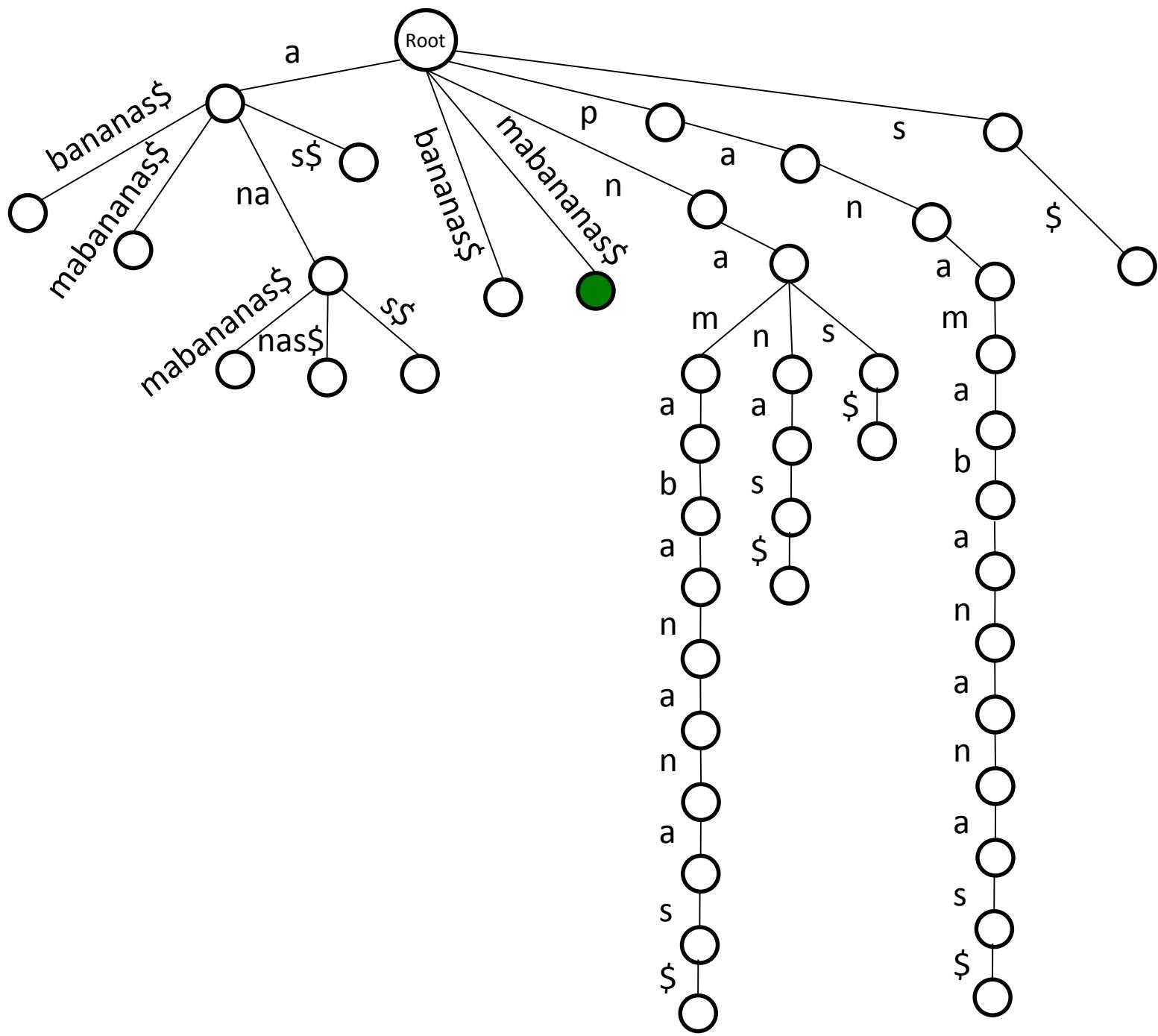


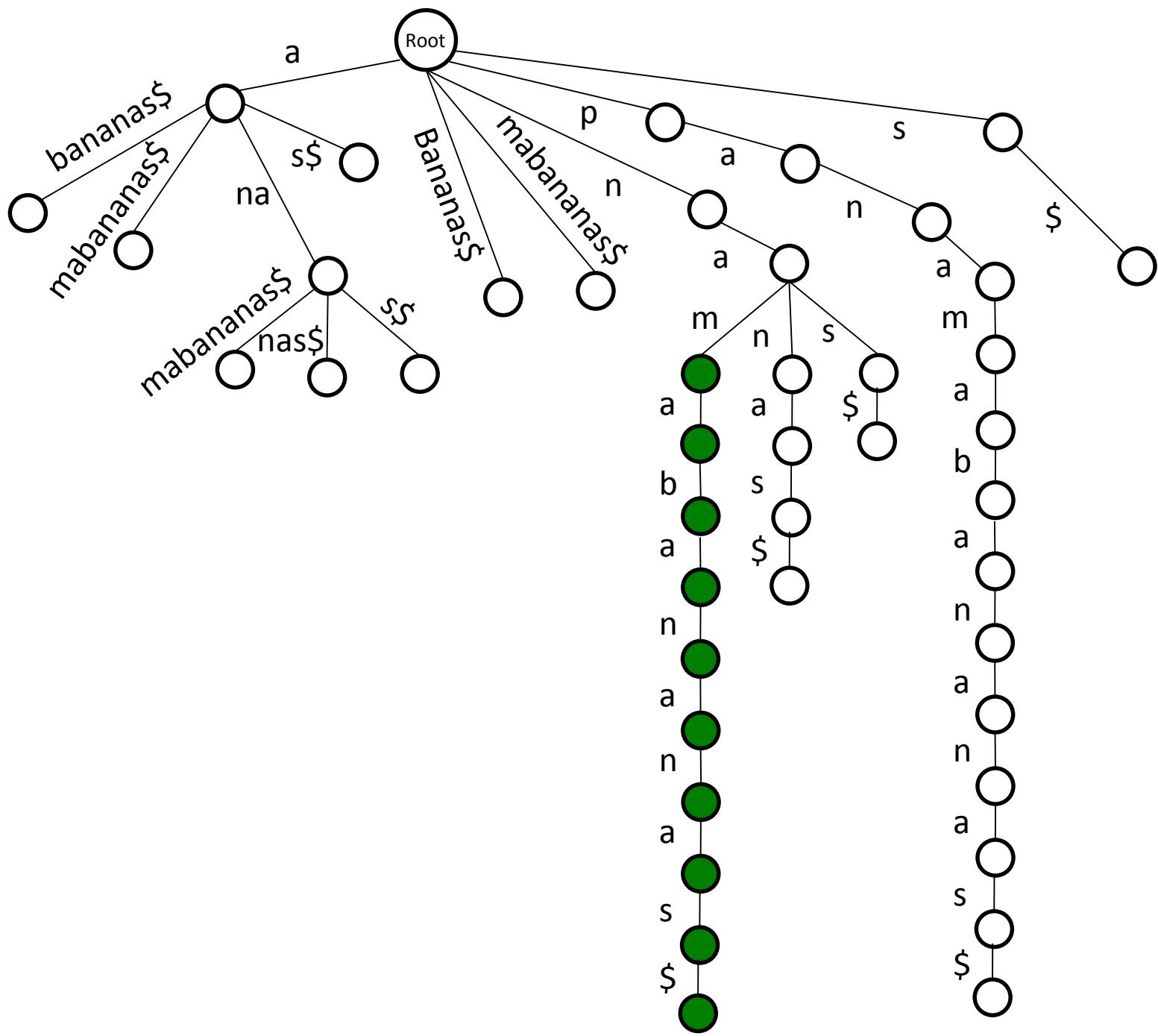


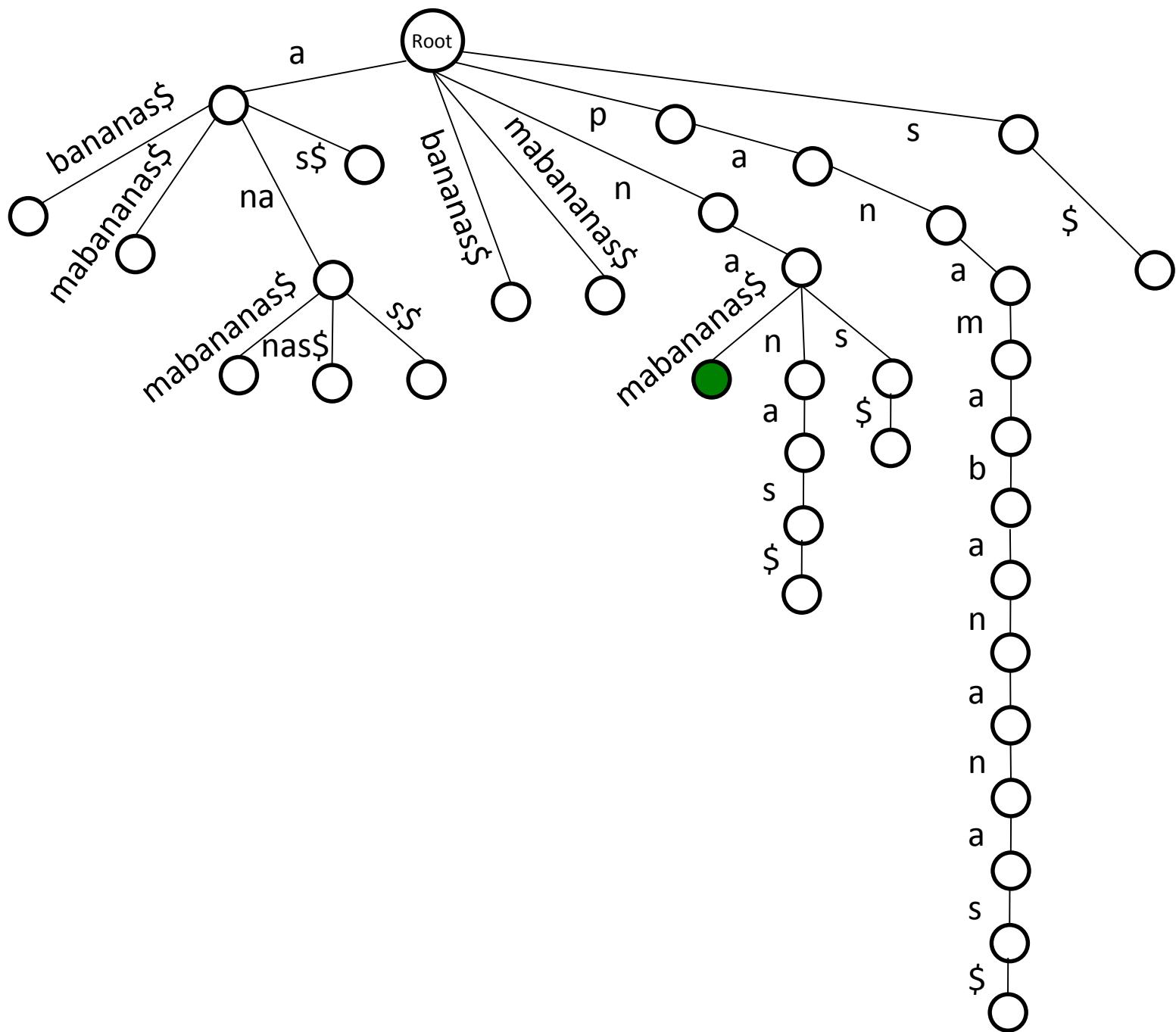


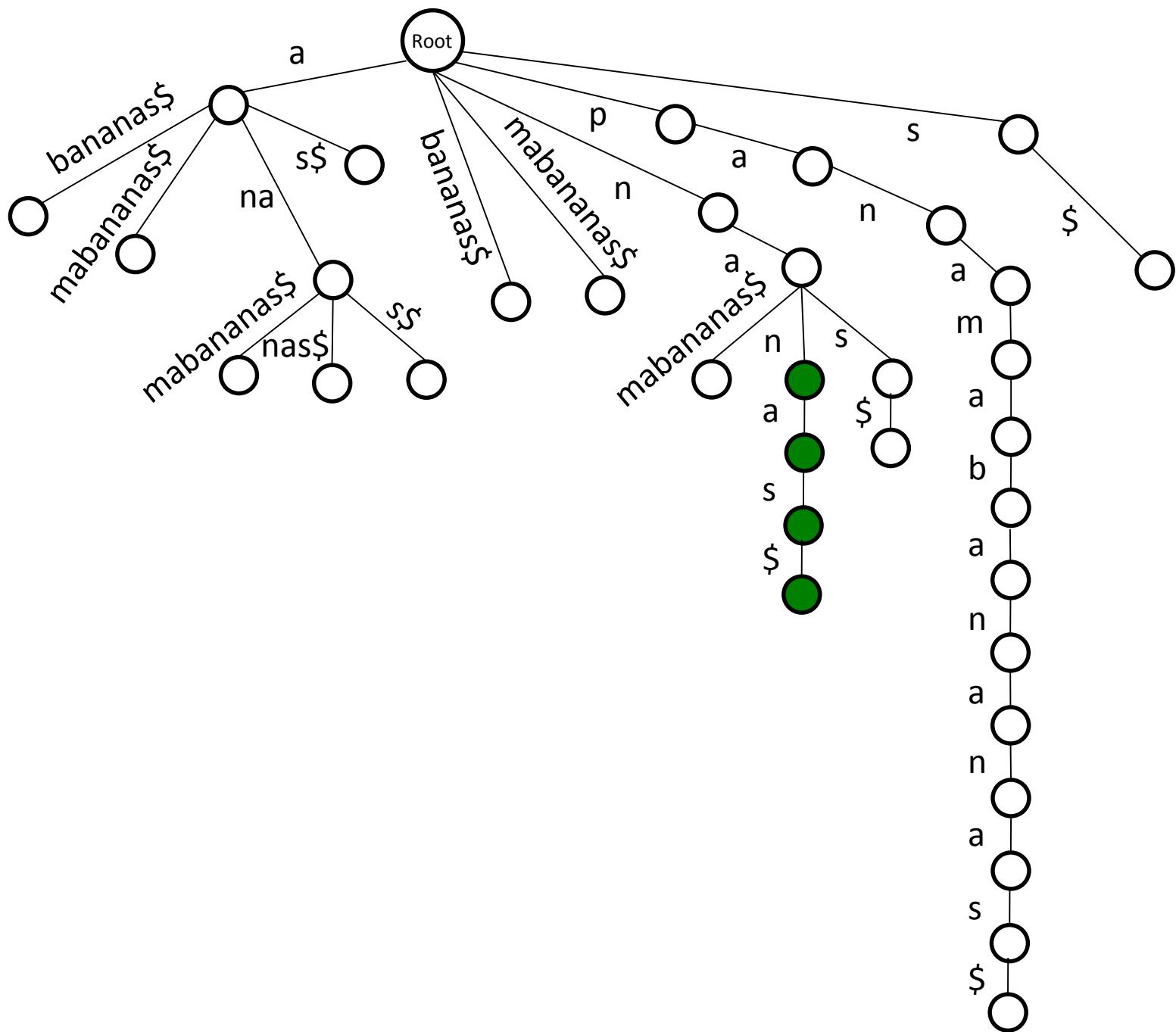


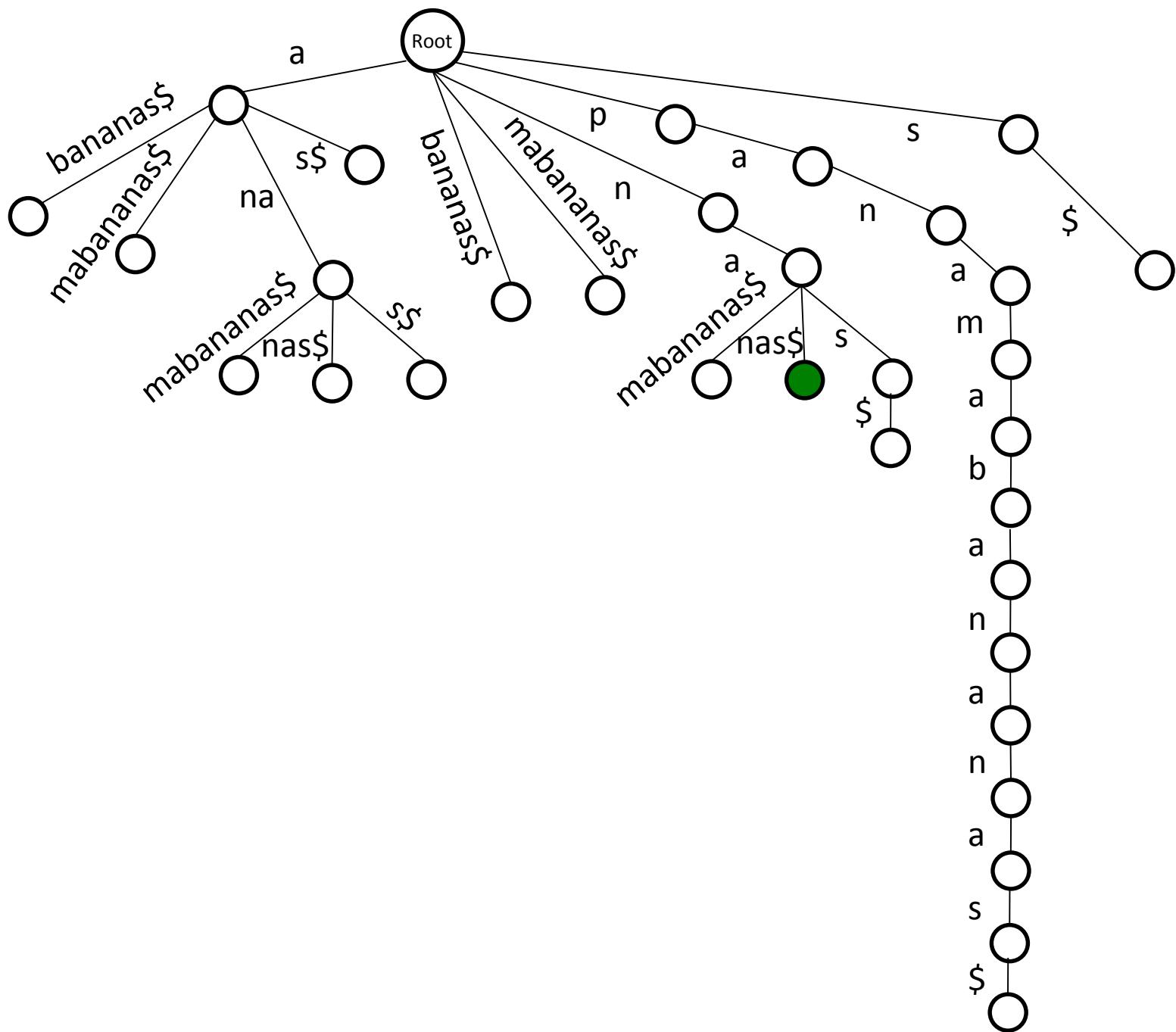


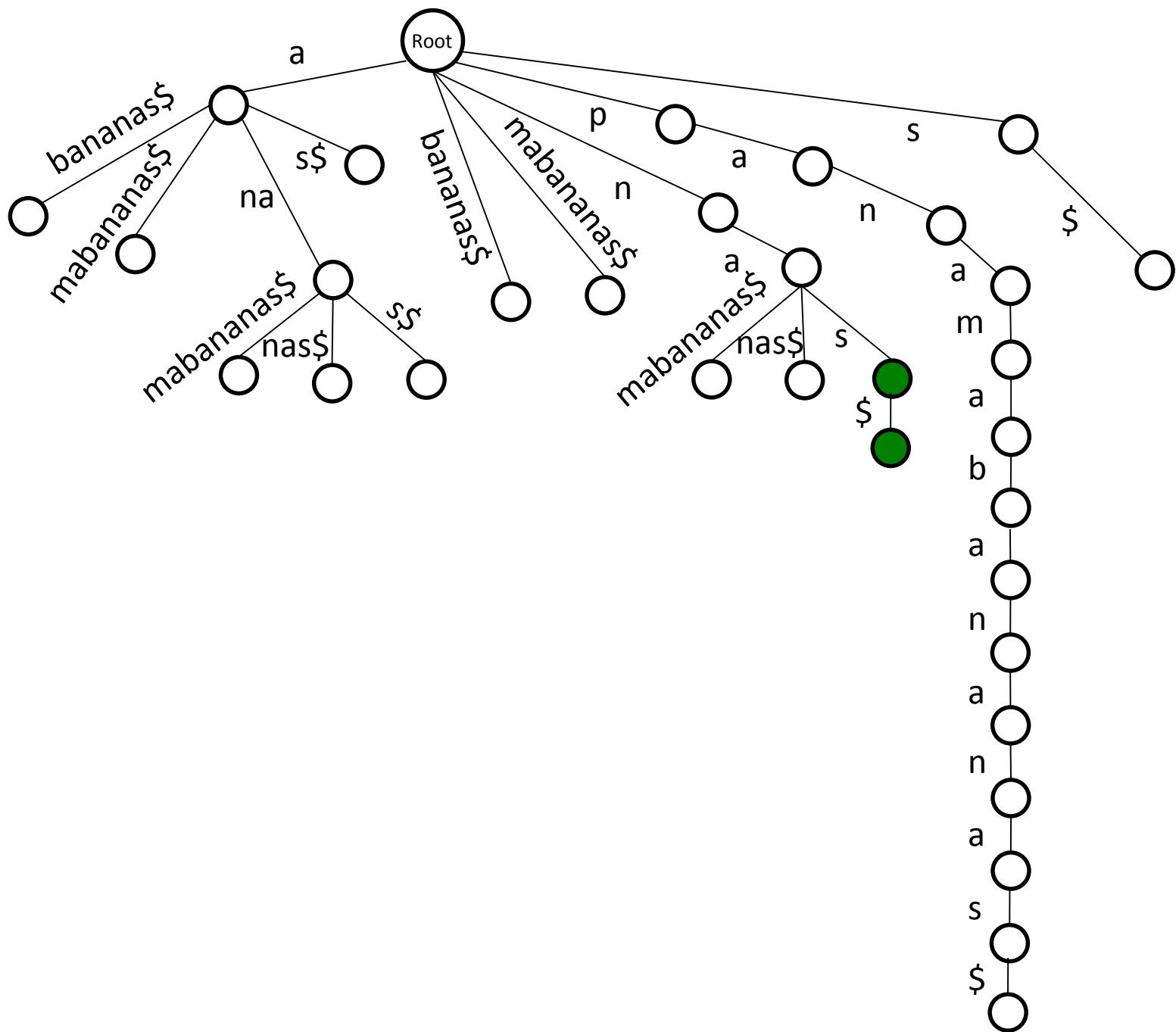


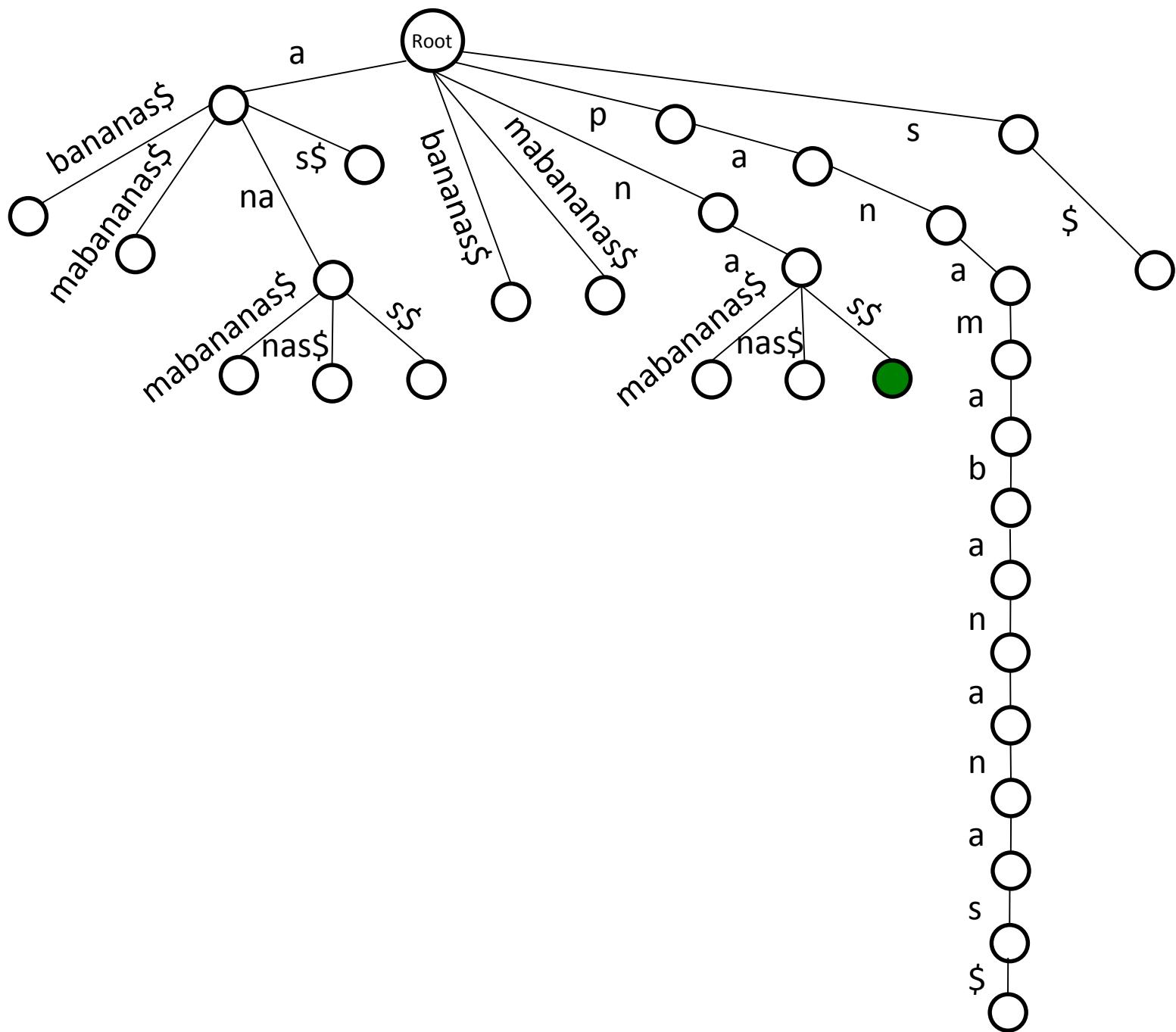


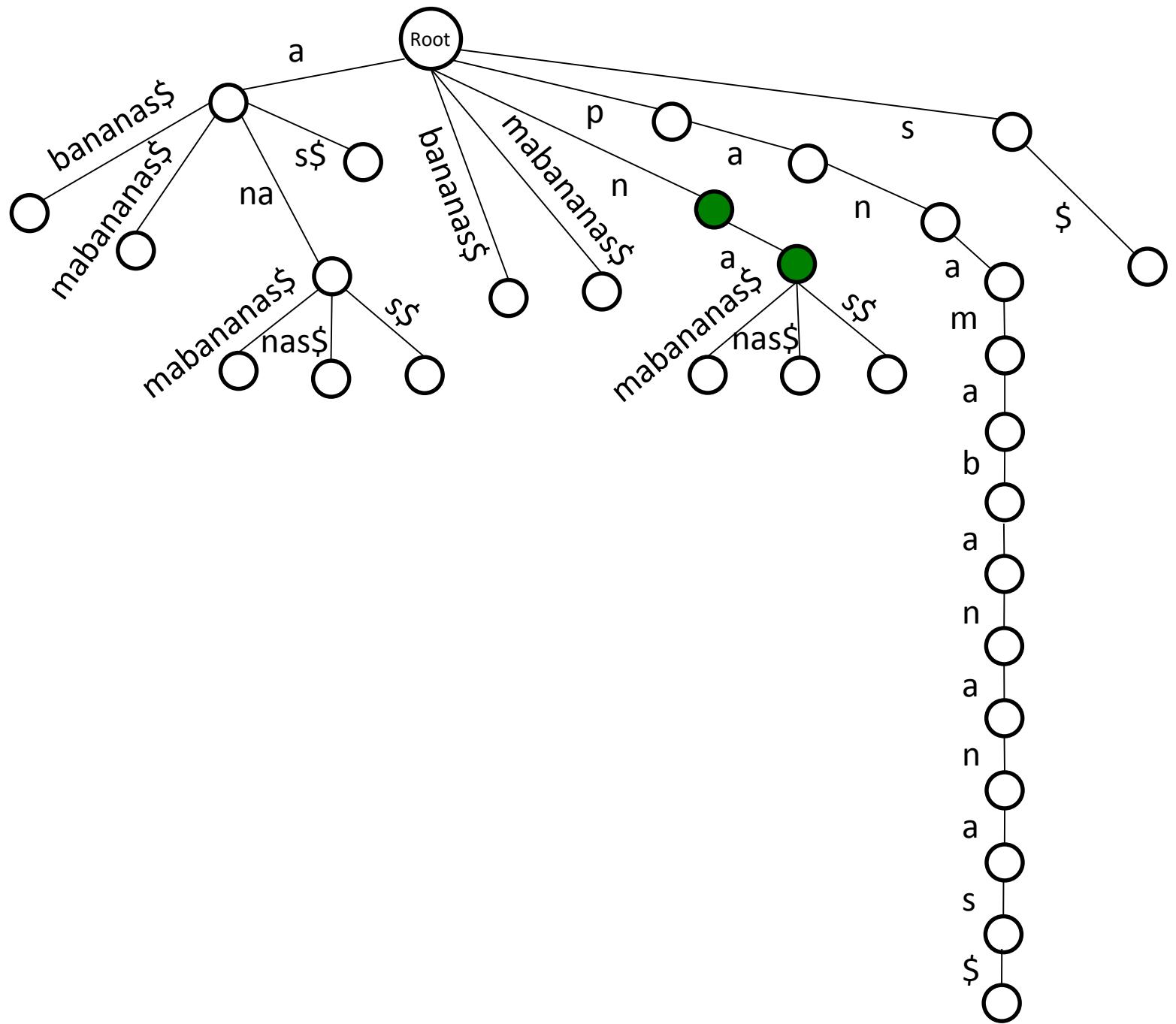


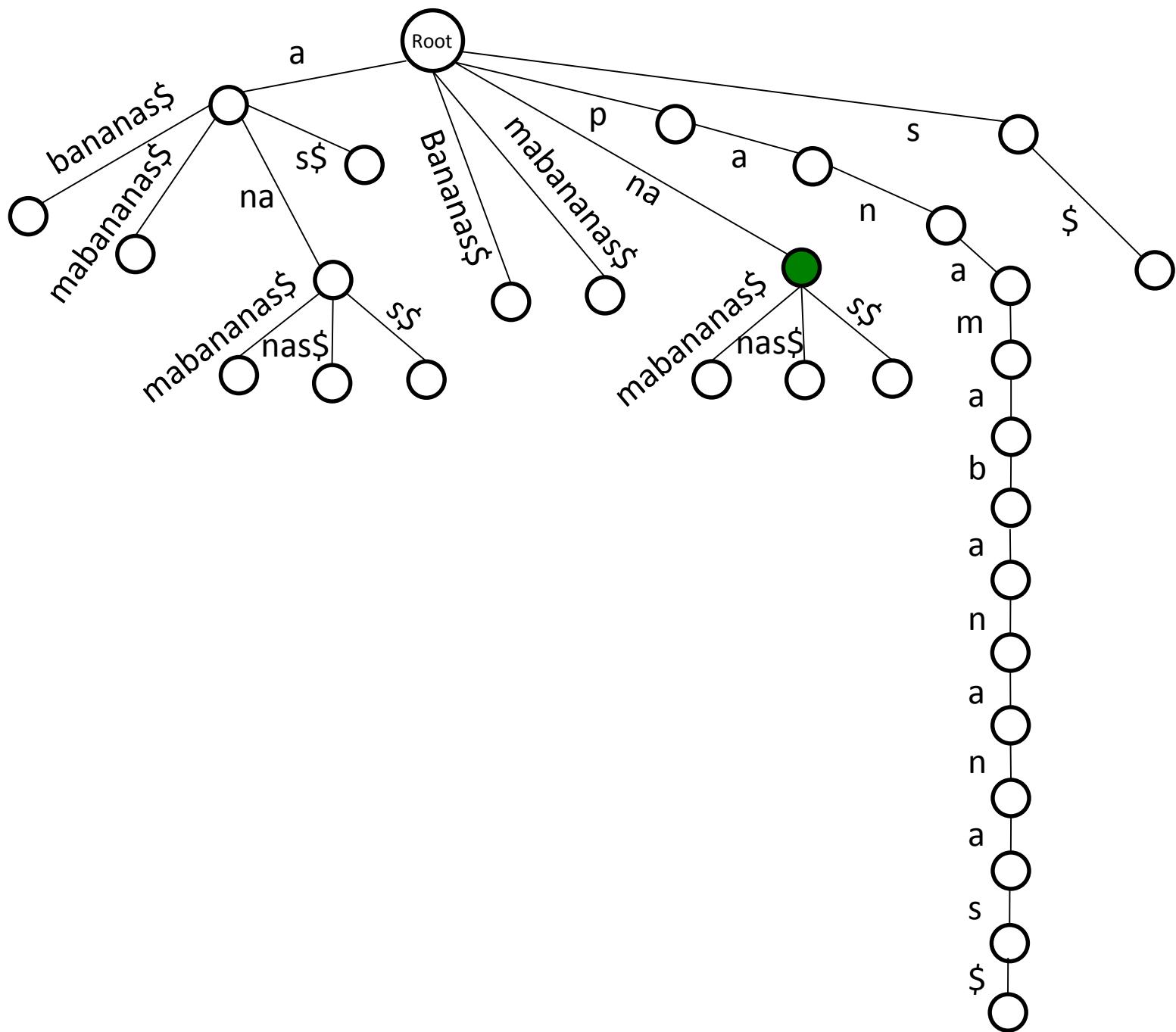


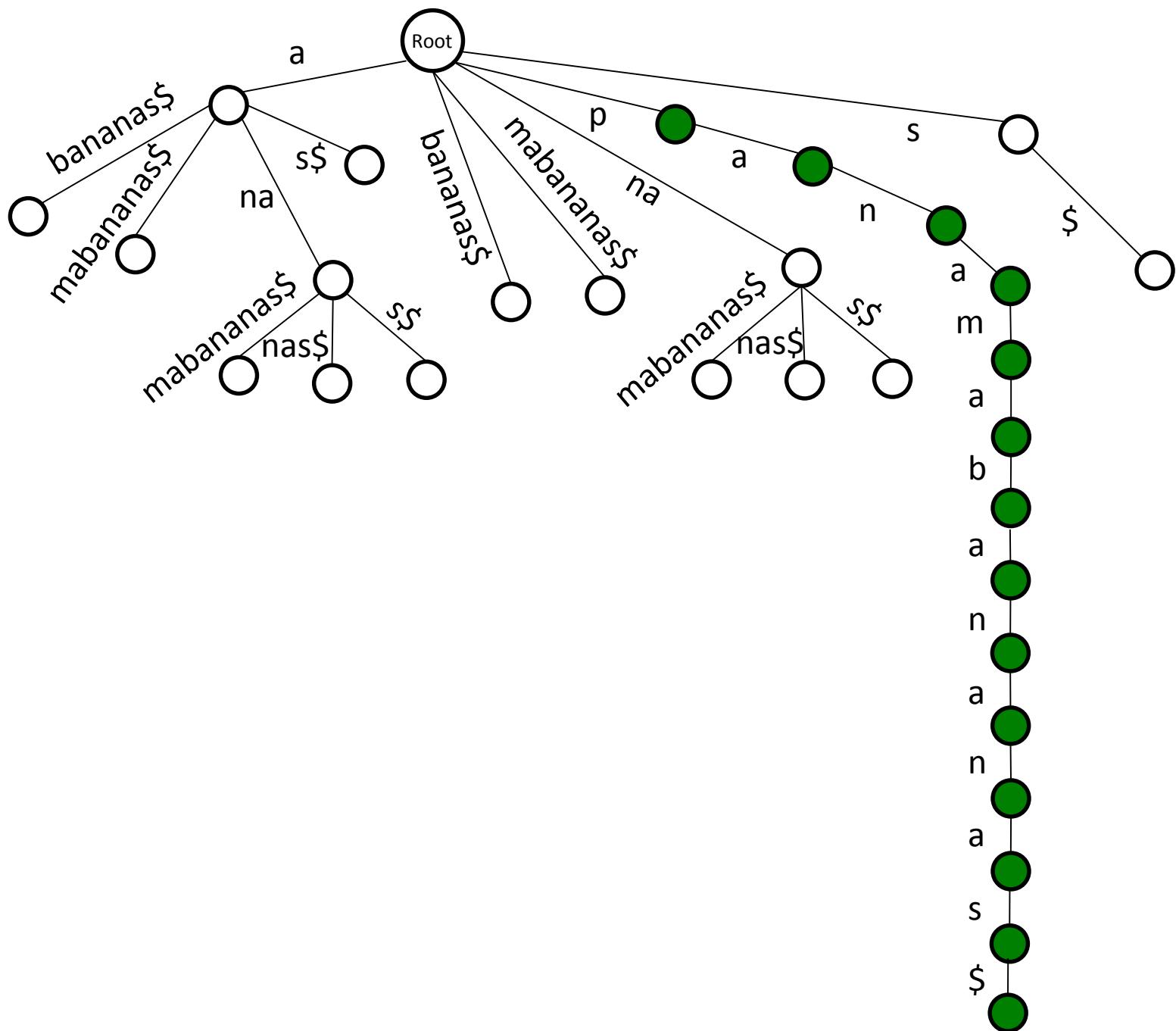


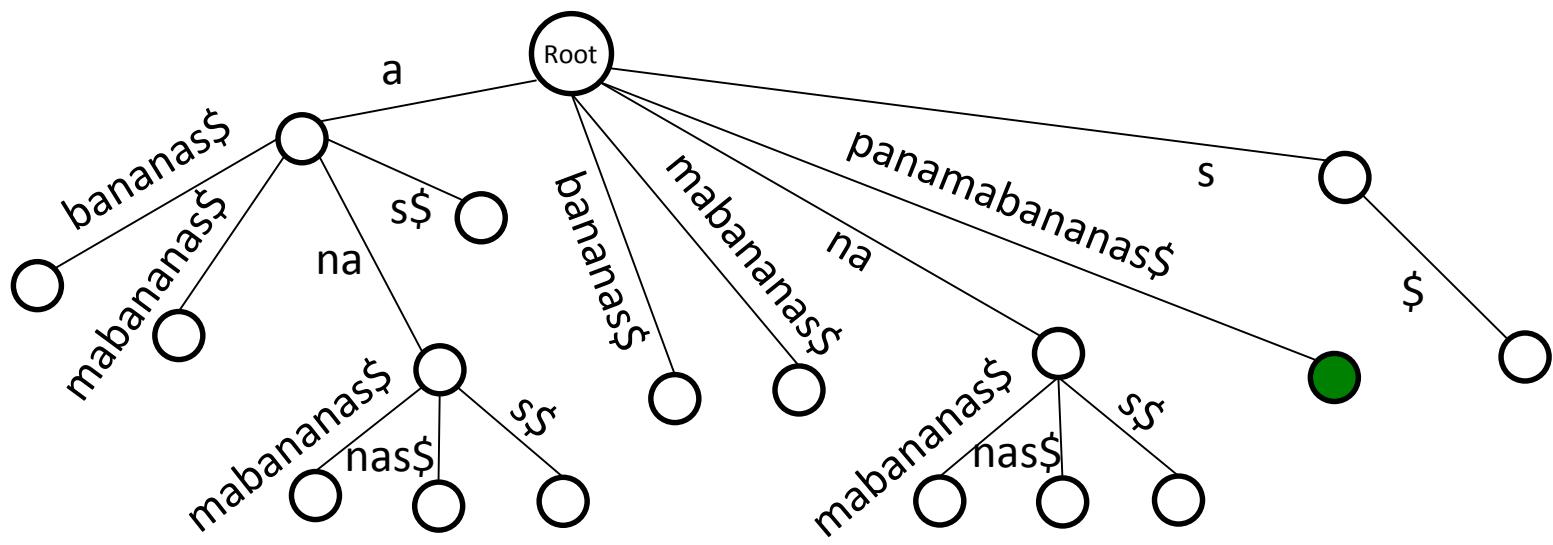


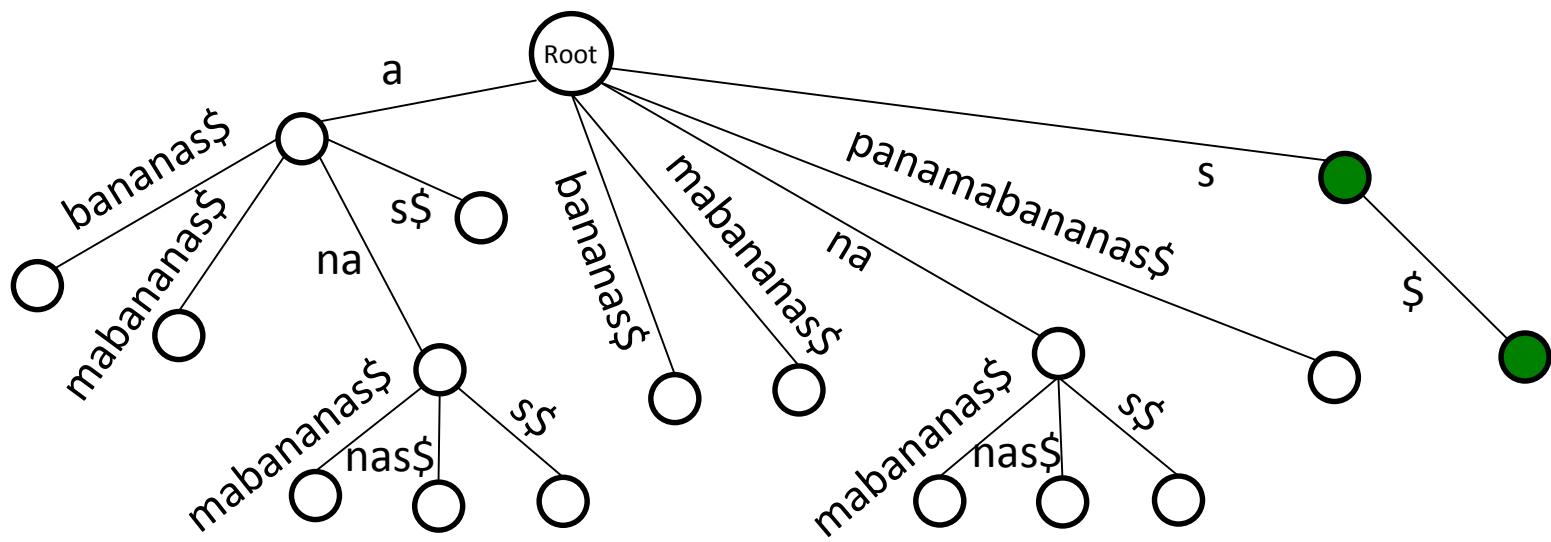


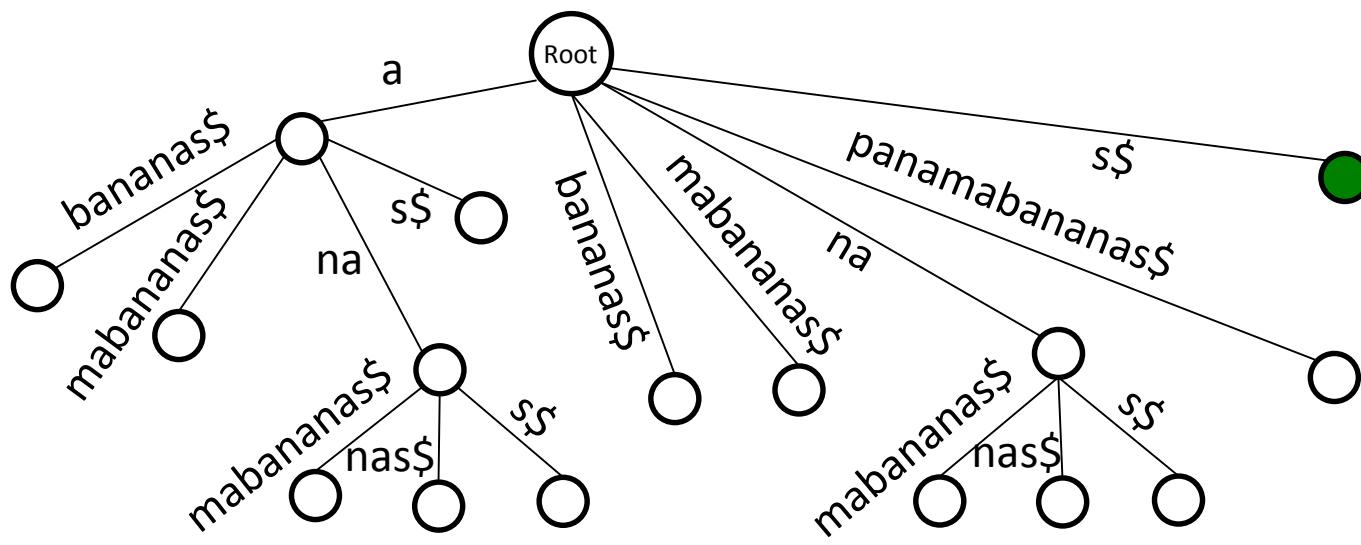




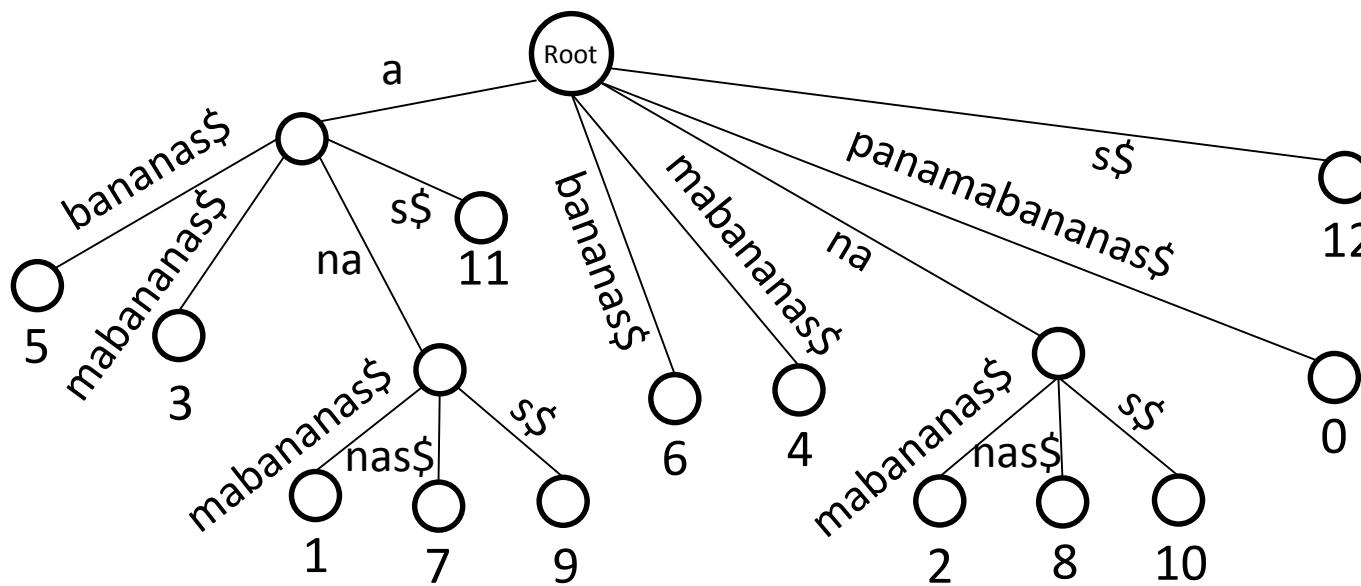






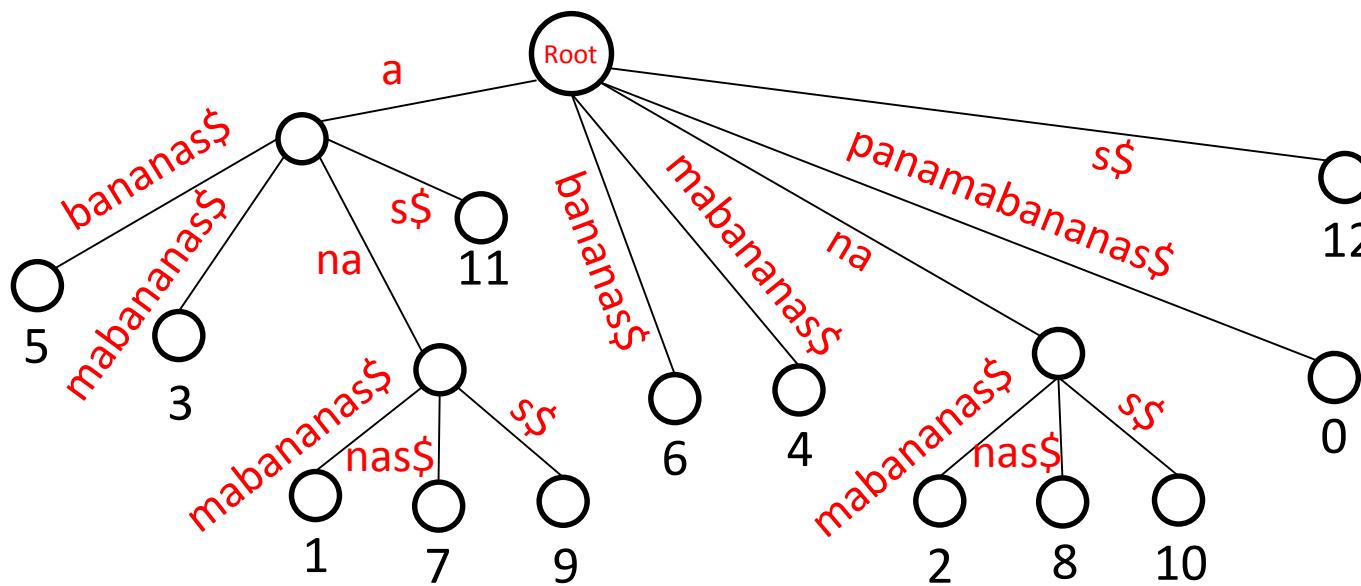


**SuffixTree(*Text*)**



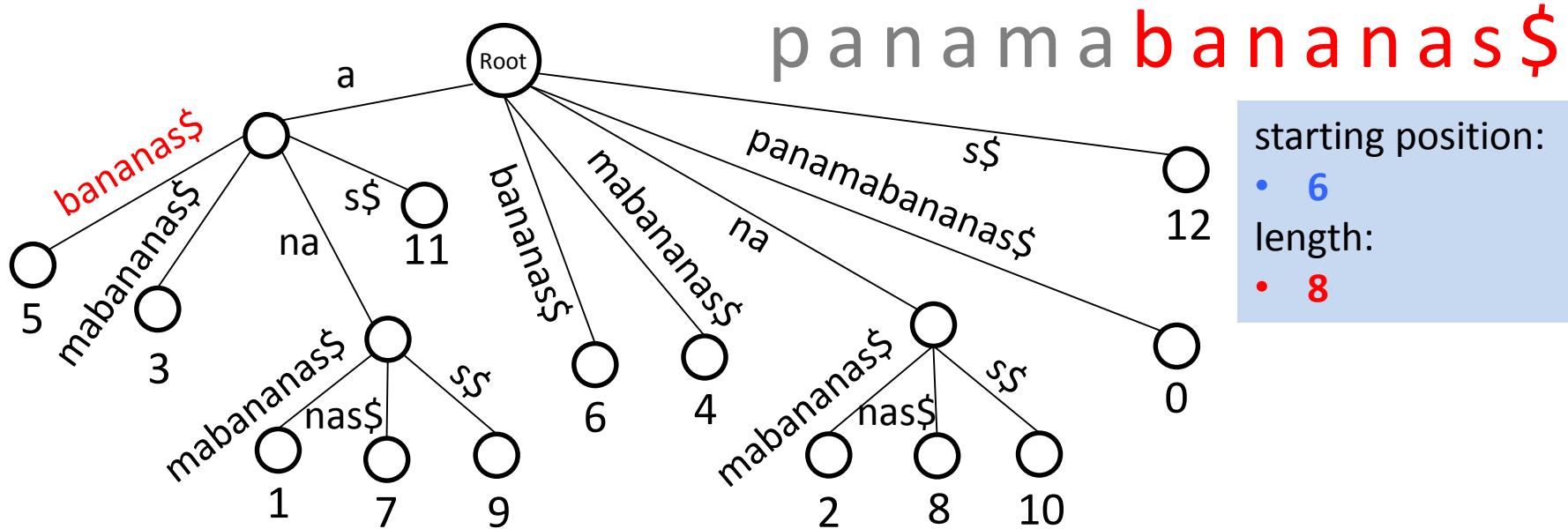
Since each suffix adds one leaf and at most one internal vertex to the suffix tree:

- # vertices  $< 2|Text|$
- memory footprint of the suffix tree:  $O(|Text|)$



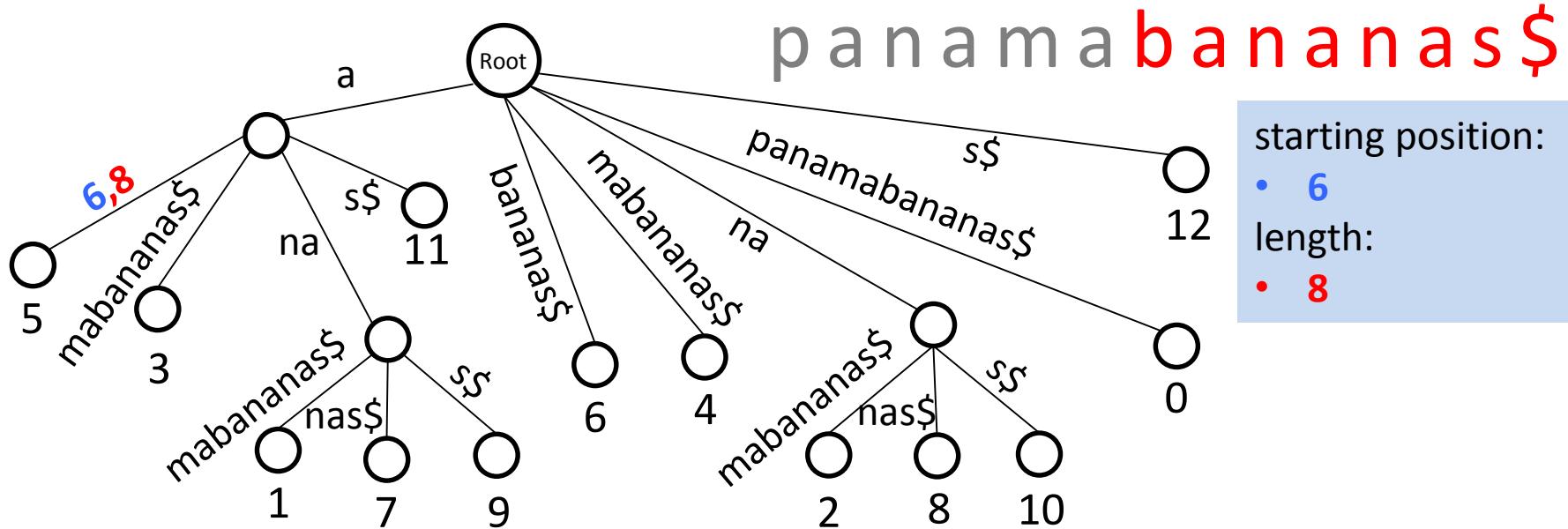
Since each suffix adds one leaf and at most one internal vertex to the suffix tree:

- # vertices  $< 2|Text|$
- memory footprint of the suffix tree:  $O(|Text|)$
- **Cheating!!!** - how do we store all edge labels?



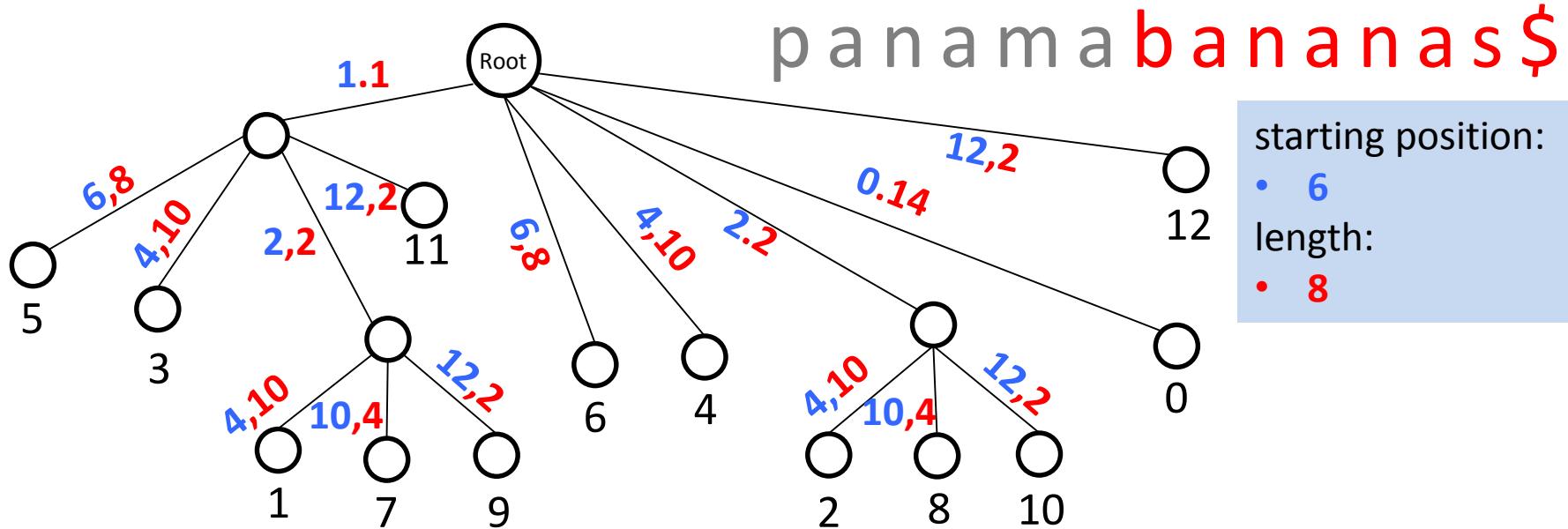
Since each suffix adds one leaf and at most one internal vertex to the suffix tree:

- # vertices  $< 2|\text{Text}|$
- memory footprint of the suffix tree:  $O(|\text{Text}|)$
- **Cheating!!!** - how do we store all edge labels?



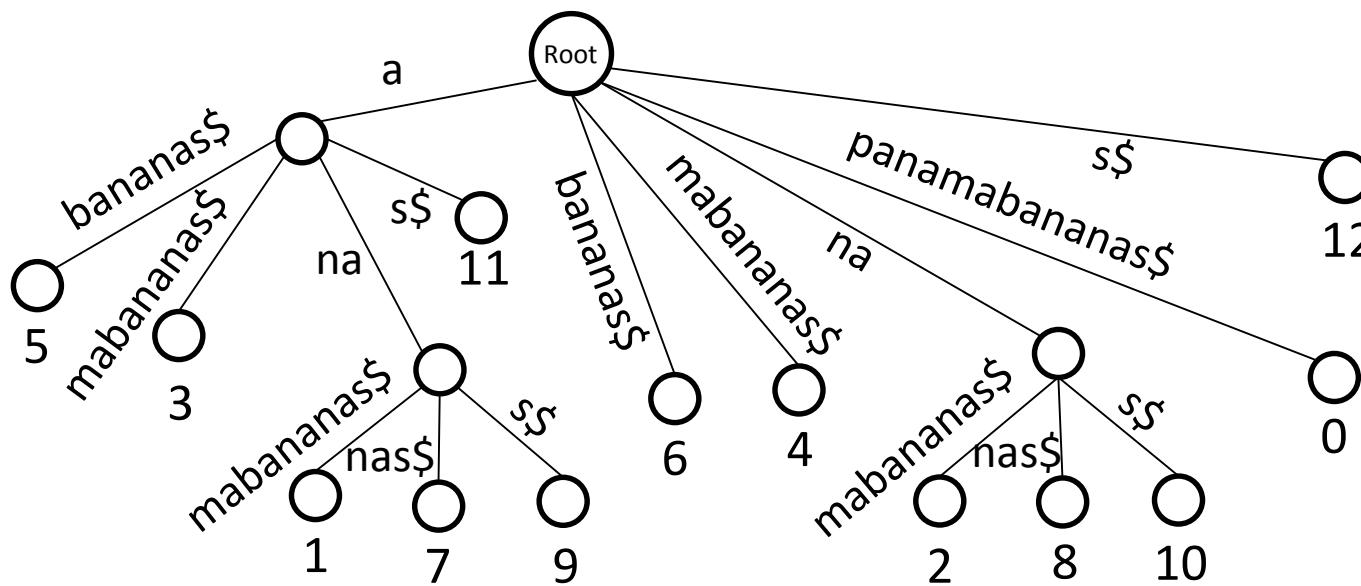
Since each suffix adds one leaf and at most one internal vertex to the suffix tree:

- # vertices  $< 2|\text{Text}|$
- memory footprint of the suffix tree:  $O(|\text{Text}|)$
- **storing edge labels**



Since each suffix adds one leaf and at most one internal vertex to the suffix tree:

- # vertices  $< 2|\text{Text}|$
- memory footprint of the suffix tree:  $O(|\text{Text}|)$
- storing edge labels



Why did we bother to add “\$” to “panamabananass”?

- to make sure that each suffix corresponds to a leaf

Why do we want to make sure that each suffix correspond to a leaf?

- construct suffix tree for “**papa**”(without adding “\$”) and compare it with the suffix tree for “**papa\$**”

# Constructing Suffix Tree: Naive Approach

- **Quadratic** runtime:
  - $O(|Text|^2)$

$O(|Genome| + |Patterns|)$  to find pattern matches

# Constructing Suffix Tree: Linear-Time Algorithm

- **Linear** runtime (for a constant-size alphabet):
  - $O(|Text|)$



Linear-time algorithm (Weiner, 1973) was simplified by Ukkonen (1995)  
but it is still too complex to cover in this course

# Big Secret of the Big O Notation

- Suffix trees enable fast **Exact** Multiple Pattern Matching:
  - Runtime:  $O(|Text| + |Patterns|)$
  - Memory:  $O(|Text|)$
- However, big-O notation hides constants!
  - suffix tree algorithms has large memory footprint  
 $\sim 20 \cdot |Text|$  for long texts like human genome
- We want to find mutations!
  - it is unclear how to develop fast **Approximate** Multiple Pattern Matching using suffix trees