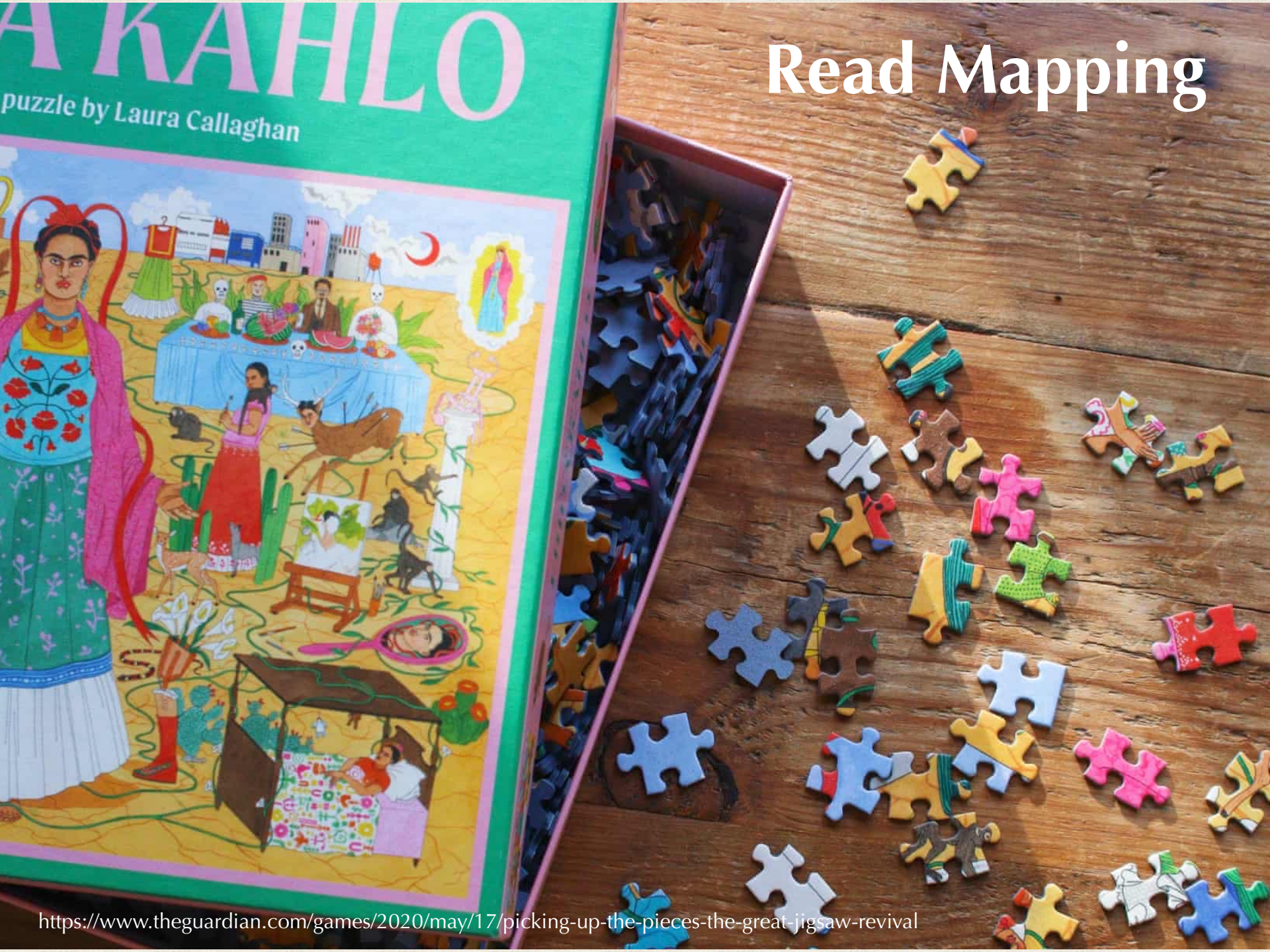
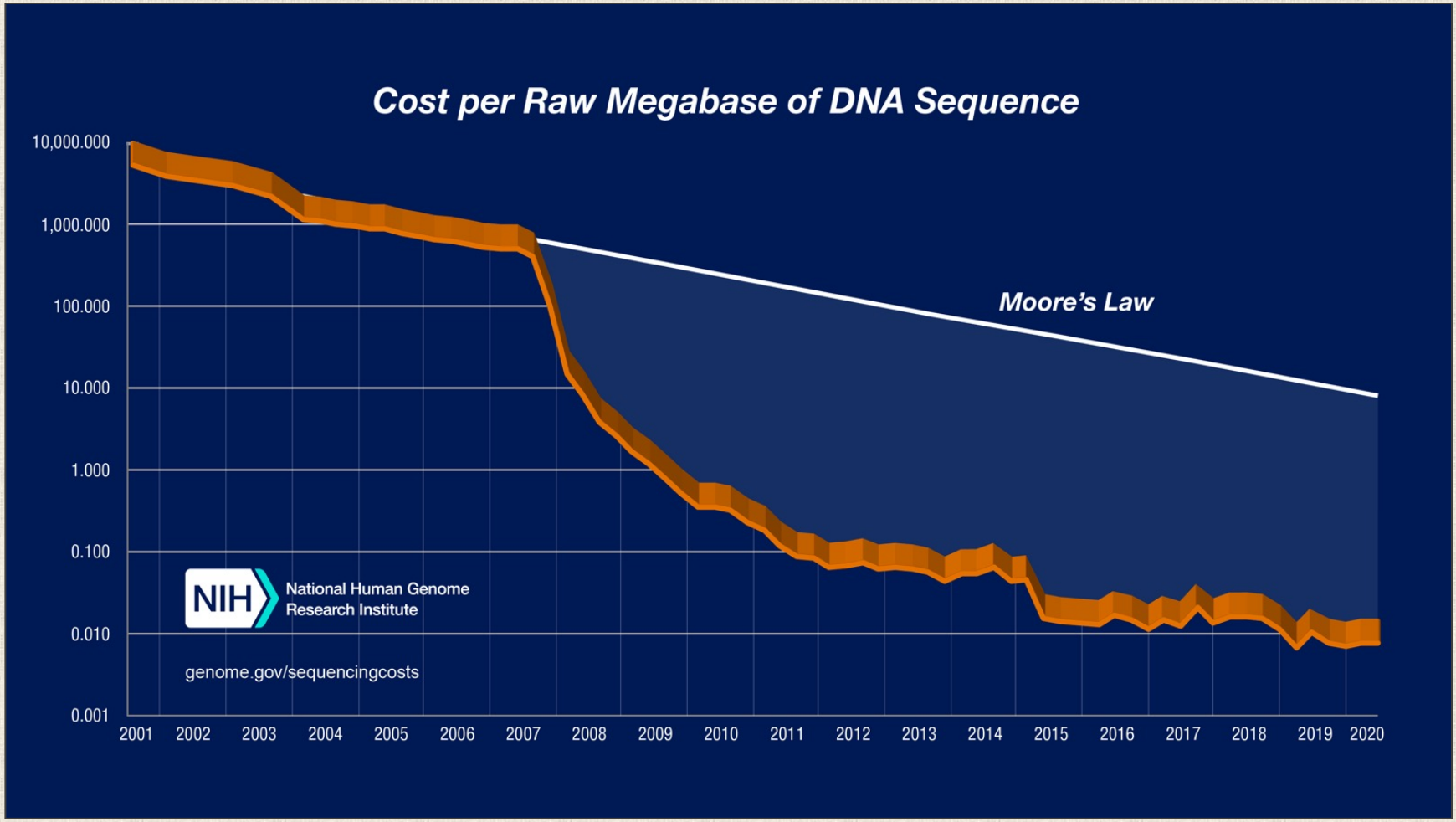
Read Mapping



https://www.theguardian.com/games/2020/may/17/picking-up-the-pieces-the-great-jigsaw-revival

Plummeting Sequencing Costs Mean Millions of Human Genomes



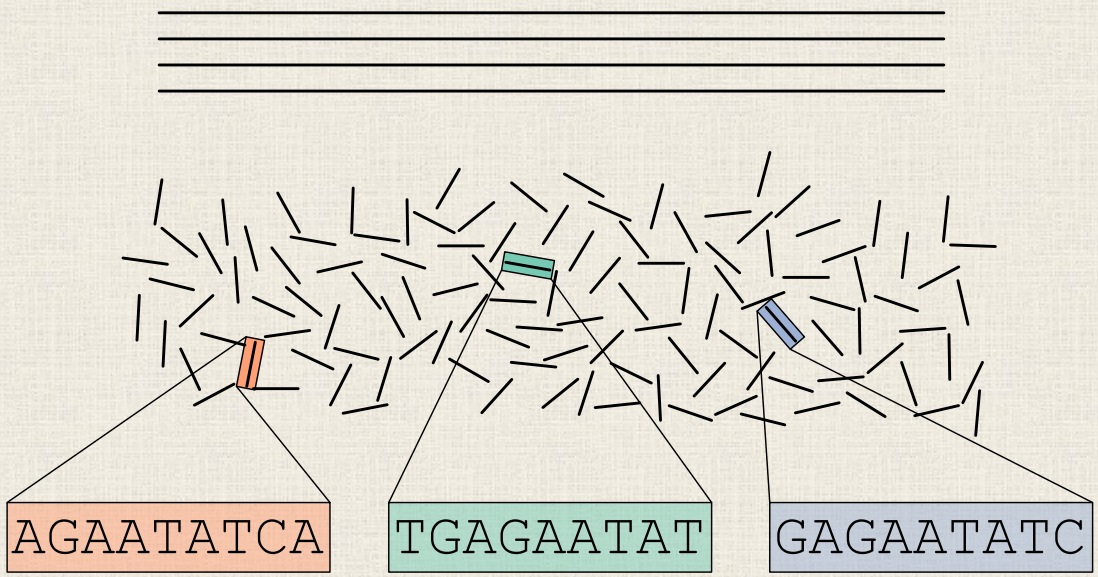
Recall: Overview of Genome Sequencing

Multiple identical copies of a genome

Shatter the genome into reads

Sequence the reads (Lab)

Assemble the genome using overlapping reads (Computational)



AGAATATCA
GAGAATATC
TGAGAATAT
...TGAGAATATCA...

Recall: Overview of Genome Sequencing

But why would we do this if we already have a good sense of the human genome?

Assemble the genome using overlapping reads (Computational)

Recall: Overview of Genome Sequencing

But why would we do this if we already have a good sense of the human genome?

Analogy: Would you assemble a jigsaw puzzle without looking at the photo on the box?

Assemble the genome using overlapping reads (Computational)

As different as we may seem, we are genetically very similar

Two humans share ~99.9% of their genomes. The rest is typically single-nucleotide variations.

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAG
CTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGATCGAT
CGATCGATTATCTACGATCGATCGATCGATCACTATACGAGCTA
CTACGTACGTACGATCGCGGGACTATTATCGACTACAGATAAAA
CATGCTAGTACAACAGTATACATAGCTGCGGGATACGATTAGCT
AATAGCTGACGATATCCGAT

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAG
CTACAACATCGTAGCTACGATGCATTAGCAAGCTATCGATCGAT
CGATCGATTATCTACGATCGATCGATCGATCACTATACGAGCTA
CTACGTACGTACGATCGCGTGACTATTATCGACTACAGATGAAA
CATGCTAGTACAACAGTATACATAGCTGCGGGATACGATTAGCT
AATAGCTGACGATATCCGAT

As different as we may seem, we are genetically very similar

Single-nucleotide polymorphism (SNP): One-nucleotide mutation present in at least 1% of humans.

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAG
CTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGATCGAT
CGATCGATTATCTACGATCGATCGATCGATCACTATACGAGCTA
CTACGTACGTACGATCGCGGGACTATTATCGACTACAGATAAAA
CATGCTAGTACAACAGTATACATAGCTGCGGGATACGATTAGCT
AATAGCTGACGATATCCGAT

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAG
CTACAACATCGTAGCTACGATGCATTAGCAAGCTATCGATCGAT
CGATCGATTATCTACGATCGATCGATCGATCACTATACGAGCTA
CTACGTACGTACGATCGCGTGACTATTATCGACTACAGATGAAA
CATGCTAGTACAACAGTATACATAGCTGCGGGATACGATTAGCT
AATAGCTGACGATATCCGAT

A First Attempt at Read Mapping

Read-mapping: Comparison of sequencing reads *Patterns* from an individual against a **reference human genome** *Text* stored in a database.

```
CTGAGGATGGACTACGCTACTACTGATAGCTGTTT      Text
  GAGGA     CCACG          TGA-A         Patterns
```

A First Attempt at Read Mapping

Read-mapping: Comparison of sequencing reads *Patterns* from an individual against a **reference human genome** *Text* stored in a database.

```
CTGAGGATGGACTACGCTACTACTGATAGCTGTTT        Text
  GAGGA      CCACG          TGA-A          Patterns
```

STOP: How could we map reads against a reference genome?

A First Attempt at Read Mapping

Read-mapping: Comparison of sequencing reads *Patterns* from an individual against a **reference human genome** *Text* stored in a database.

```
CTGAGGATGGACTACGCTACTACTGATAGCTGTTT        Text
  GAGGA      CCACG          TGA-A          Patterns
```

Idea: We could *align* each read against the reference genome. But what problem are we solving?

Recall these Two Problems from Text

A **fitting alignment** of *v* and *w* is an alignment of a substring of *v* against all of *w*.

Fitting Alignment Problem:

- **Input:** Two strings and a scoring matrix.
- **Output:** A fitting alignment of the strings with maximum alignment score according to the scoring matrix.

Global	Local	Fitting
CGTAGGCTTAAGGTTA	CGTAGGCTTAAGGTTA	CGTAGGCTTAAGGTTA
A-TAG----A---T-A	ATAGATA	ATAGA--TA

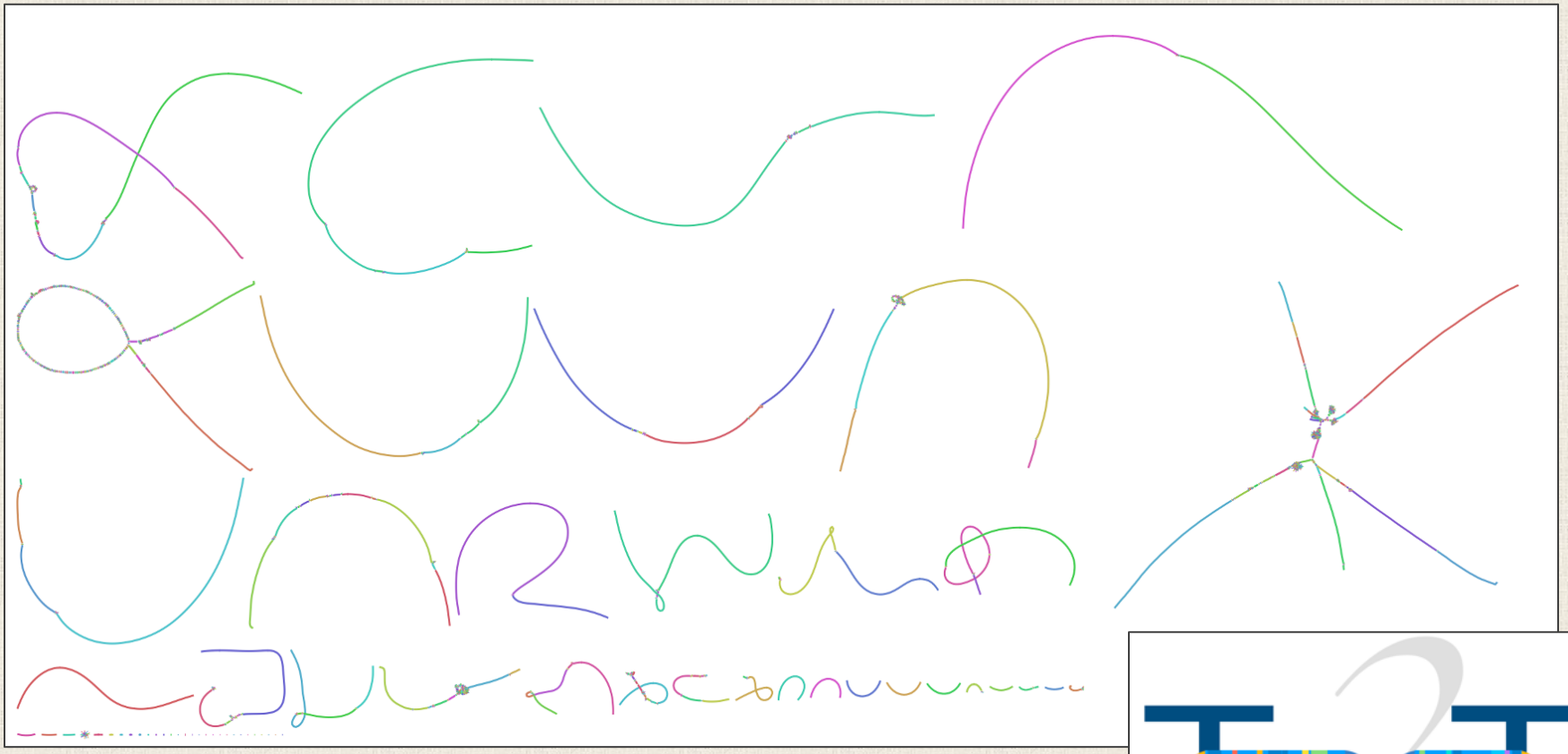
A First Attempt at Read Mapping

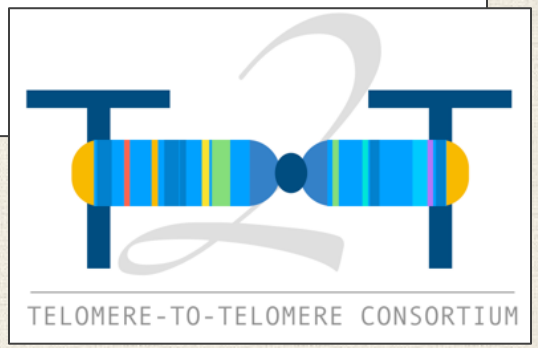
Read-mapping: Comparison of sequencing reads *Patterns* from an individual against a **reference human genome** *Text* stored in a database.

```
CTGAGGATGGACTACGCTACTACTGATAGCTGTTT      Text
  GAGGA      CCACG          TGA-A        Patterns
```

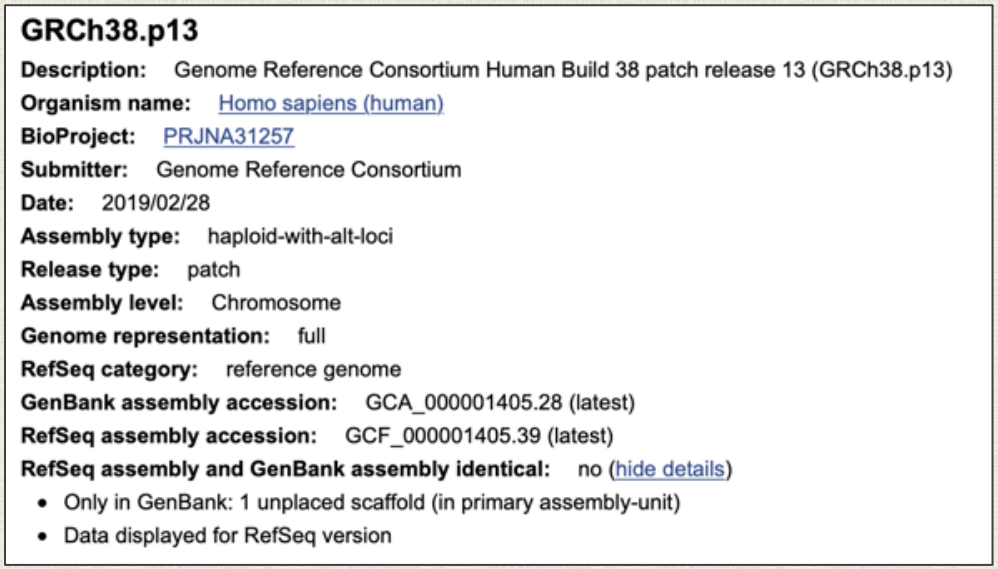
Problem: Fitting alignment has runtime O(|*Pattern*| * |*Text*|) for each string *Pattern* in our reads – too long!

Remember: The First *Full* Human Genome Wasn't Sequenced Until 2020!





Reference Genomes are Not Diverse



GRCh38.p13

Description: Genome Reference Consortium Human Build 38 patch release 13 (GRCh38.p13)
Organism name: Homo sapiens (human)
BioProject: PRJNA31257
Submitter: Genome Reference Consortium
Date: 2019/02/28
Assembly type: haploid-with-alt-loci
Release type: patch
Assembly level: Chromosome
Genome representation: full
RefSeq category: reference genome
GenBank assembly accession: GCA_000001405.28 (latest)
RefSeq assembly accession: GCF_000001405.39 (latest)
RefSeq assembly and GenBank assembly identical: no (hide details)

- Only in GenBank: 1 unplaced scaffold (in primary assembly-unit)
- Data displayed for RefSeq version

https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.39

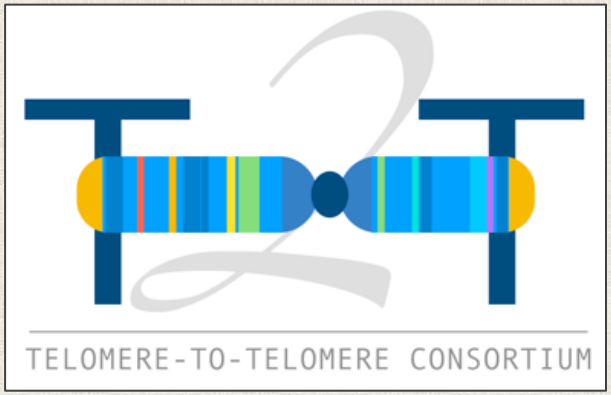
When we ″map″ reads against a reference human genome, the most commonly used reference is 70% made up of RP11, or “some guy from Buffalo”.

The first complete human reference is taken from a single sperm

This reference is quickly being replaced by "CHM13", the first ever complete human genome.

CHM13 is a cell line grown from a "molar pregnancy", an empty egg fertilized by a sperm. Both copies of chromosomes come from this sperm.

MAGEE-WOMENS RESEARCH INSTITUTE



Long, Cheap Reads Give Hope of a "Pan Genome" Era

TOWARDS A COMPLETE REFERENCE OF HUMAN GENOME DIVERSITY

STOP: Why might mapping reads from one ethnicity against a genome from a different ethnicity lead to biased conclusions?

INTRODUCTION TO MULTIPLE PATTERN MATCHING

Simplifying Assumption: No Mutations

We will first find all the *exact* matches of reads in the reference. If we are looking for variants, then we can eliminate from consideration areas of the reference genome where exact matches occur.

Simplifying Assumption: No Mutations

We will first find all the *exact* matches of reads in the reference. If we are looking for variants, then we can eliminate from consideration areas of the reference genome where exact matches occur.

Key Point: This lecture will assume that we are working with highly accurate reads rather than messy reads.

Simplifying Assumption: No Mutations

We will first find all the *exact* matches of reads in the reference. If we are looking for variants, then we can eliminate from consideration areas of the reference genome where exact matches occur.

Multiple Pattern Matching Problem: *Find all occurrences of a collection of patterns in a text.*

- **Input:** A string *Text* and a collection *Patterns* containing (shorter) strings.
- **Output:** All starting positions in *Text* where a string from *Patterns* appears as a substring.

Wait: Isn't This Too Simple?

STOP: What is the probability of a randomly selected 250-mer matching in two genomes with 99.9% similarity? (Assume mutations are uniform.)

Multiple Pattern Matching Problem: *Find all occurrences of a collection of patterns in a text.*

- **Input:** A string *Text* and a collection *Patterns* containing (shorter) strings.
- **Output:** All starting positions in *Text* where a string from *Patterns* appears as a substring.

Wait: Isn't This Too Simple?

Answer: We need all 250 positions to match, so the probability is $(1\text{-}0.001)^{250} = 0.779$. So nearly 80% of 250-mers will match in two 99.9% similar genomes.

Multiple Pattern Matching Problem: *Find all occurrences of a collection of patterns in a text.*

- **Input:** A string *Text* and a collection *Patterns* containing (shorter) strings.
- **Output:** All starting positions in *Text* where a string from *Patterns* appears as a substring.

Wait: Isn't This Too Simple?

BruteForcePatternMatching: slide each string down *Text,* one at a time.

Multiple Pattern Matching Problem: *Find all occurrences of a collection of patterns in a text.*

- **Input:** A string *Text* and a collection *Patterns* containing (shorter) strings.
- **Output:** All starting positions in *Text* where a string from *Patterns* appears as a substring.

Wait: Isn't This Too Simple?

STOP: What is the Big-O runtime of **BruteForcePatternMatching**?

Multiple Pattern Matching Problem: *Find all occurrences of a collection of patterns in a text.*

- **Input:** A string *Text* and a collection *Patterns* containing (shorter) strings.
- **Output:** All starting positions in *Text* where a string from *Patterns* appears as a substring.

So Things Are Much Harder Than They Seem …

Answer: For one read, it is O(|*Text*| * |*Pattern*|). For multiple reads, it is O(|*Text*| * |*Patterns*|), where |*Patterns*| is the total length of *Patterns*.

Multiple Pattern Matching Problem: *Find all occurrences of a collection of patterns in a text.*

- **Input:** A string *Text* and a collection *Patterns* containing (shorter) strings.
- **Output:** All starting positions in *Text* where a string from *Patterns* appears as a substring.

HERDING PATTERNS INTO A TRIE

In Brute Force, Patterns Travel One at a Time

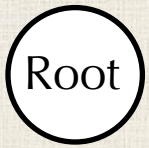
Text

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTAC

Idea: Organize Patterns and Make a Single Pass Down the Reference Genome

Text

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTAC

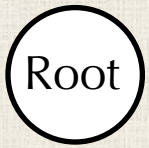


Trie: Graph constructed from *Patterns*.

- There is a single root node with indegree 0, denoted *root*; all other nodes have indegree 1.
- Each edge of *Trie*(*Patterns*) is labeled with a letter.
- Edges leading out of a given node have distinct labels.
- Every string in *Patterns* is spelled out by concatenating the letters along some path from the root downward.
- Every path from the root to a **leaf** (node with outdegree 0) spells a string from *Patterns*.

Patterns

```
banana
pan
and
nab
antenna
bandana
ananas
nana
```



Patterns

banana

pan

and

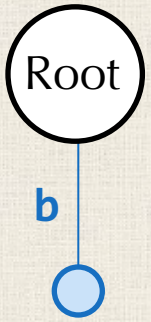
nab

antenna

bandana

ananas

nana



Patterns

banana

pan

and

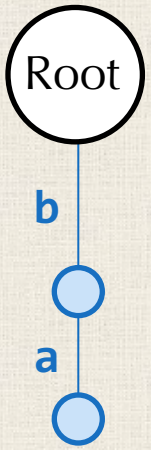
nab

antenna

bandana

ananas

nana



Patterns

banana

pan

and

nab

antenna

bandana

ananas

nana

Patterns

banana

pan

and

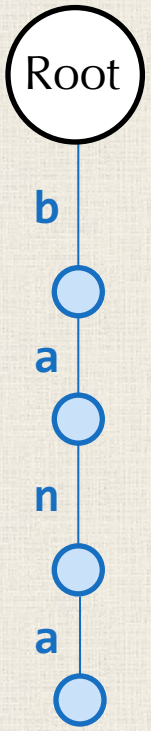
nab

antenna

bandana

ananas

nana



Patterns

banana

pan

and

nab

antenna

bandana

ananas

nana

Patterns

banana

pan

and

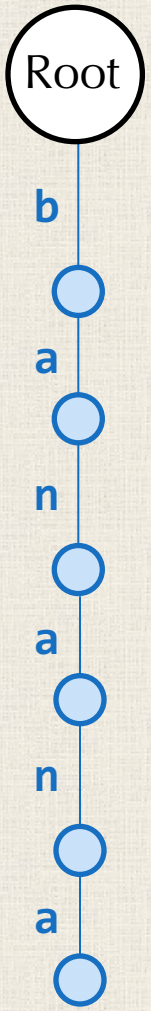
nab

antenna

bandana

ananas

nana



Patterns

banana

pan

and

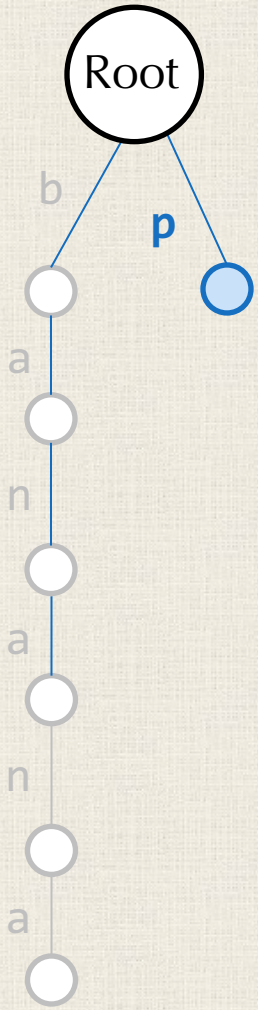
nab

antenna

bandana

ananas

nana



Patterns

banana

pan

and

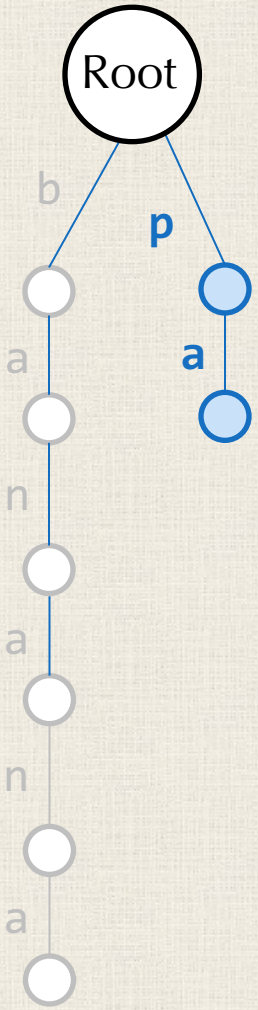
nab

antenna

bandana

ananas

nana



Patterns

banana

pan

and

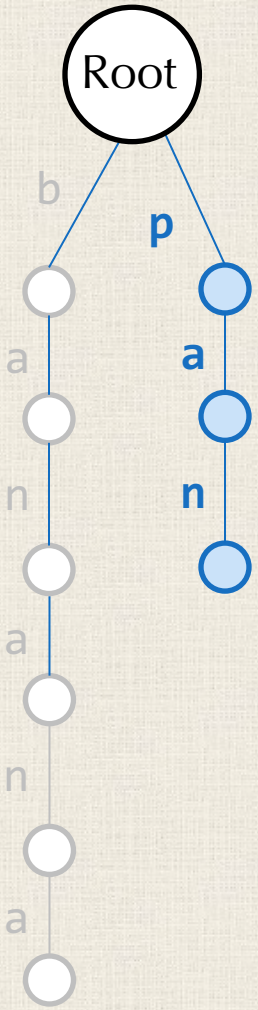
nab

antenna

bandana

ananas

nana



Patterns

banana

pan

and

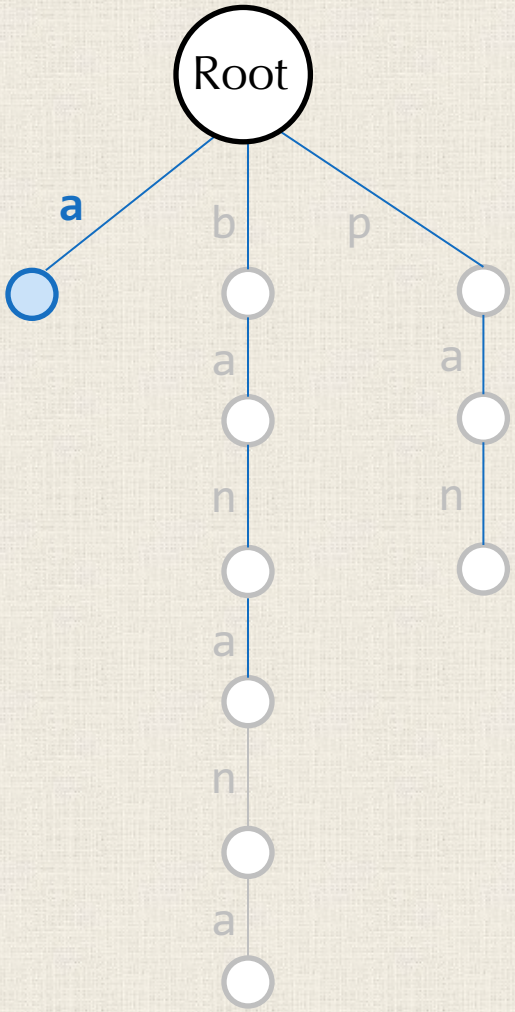
nab

antenna

bandana

ananas

nana



Patterns

banana

pan

and

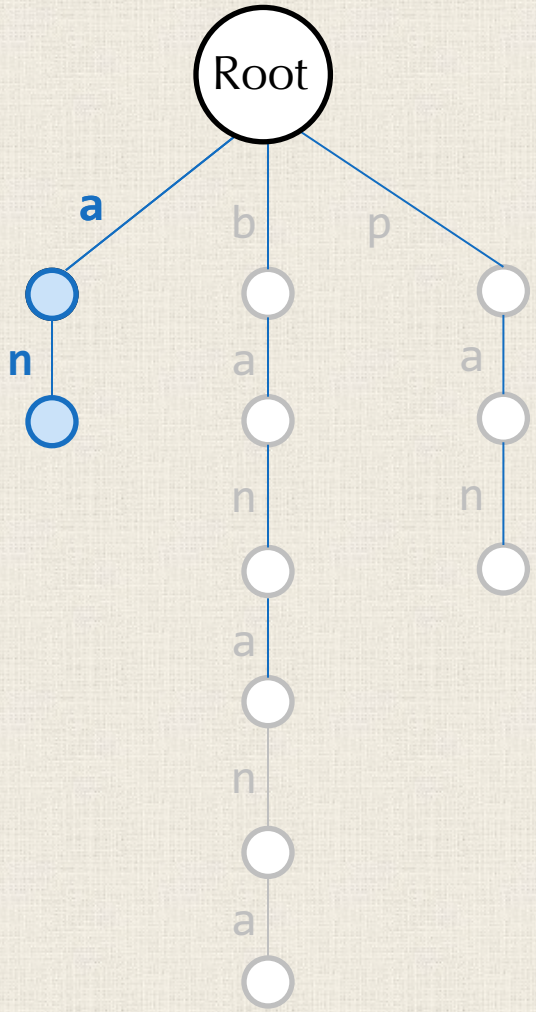
nab

antenna

bandana

ananas

nana



Patterns

banana

pan

and

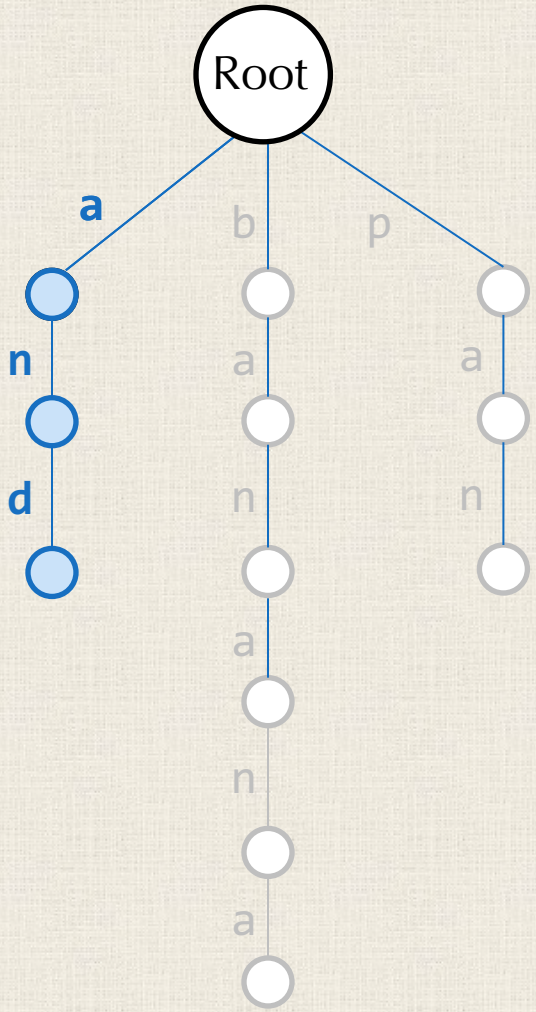
nab

antenna

bandana

ananas

nana



Patterns

banana

pan

and

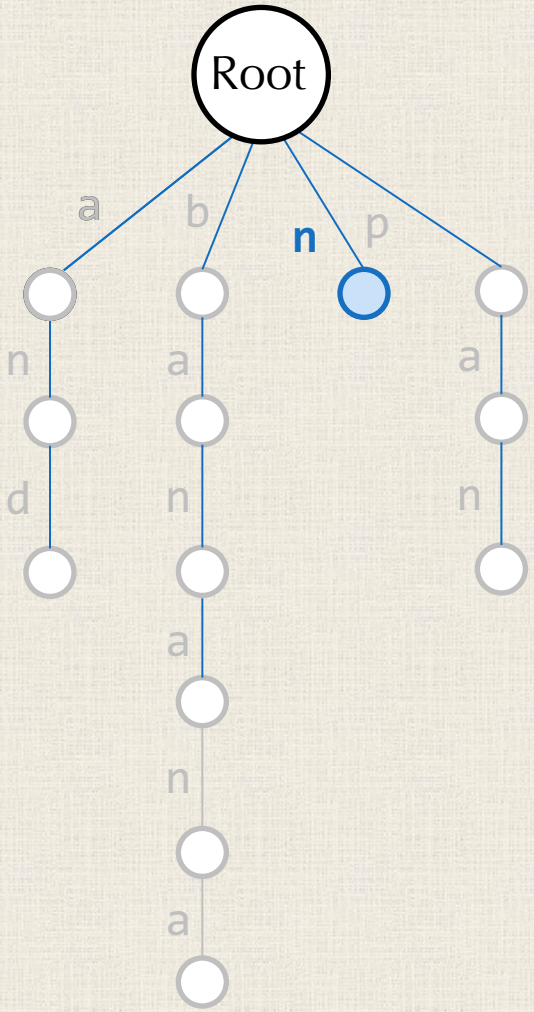
nab

antenna

bandana

ananas

nana



Patterns

banana

pan

and

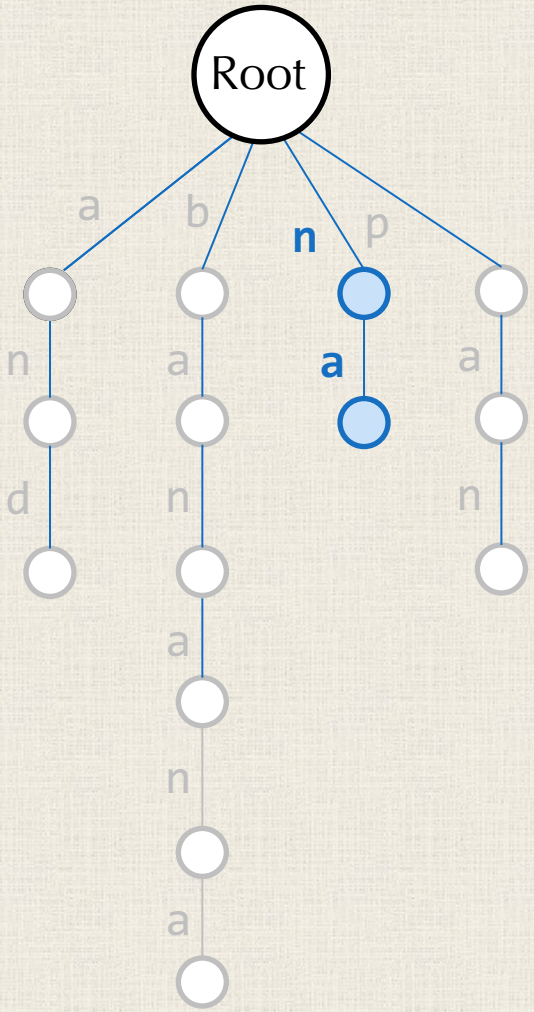
nab

antenna

bandana

ananas

nana



Patterns

banana

pan

and

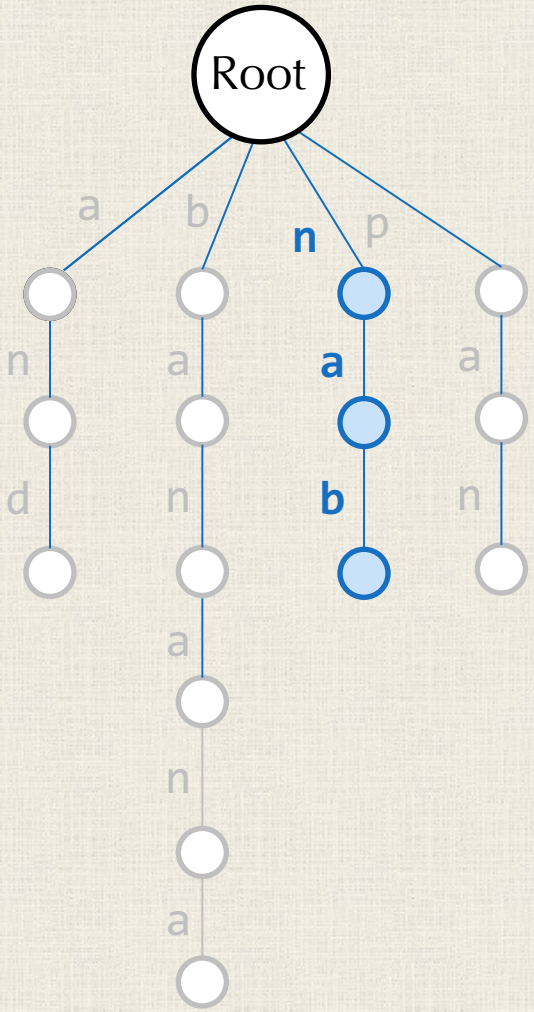
nab

antenna

bandana

ananas

nana



Patterns

banana

pan

and

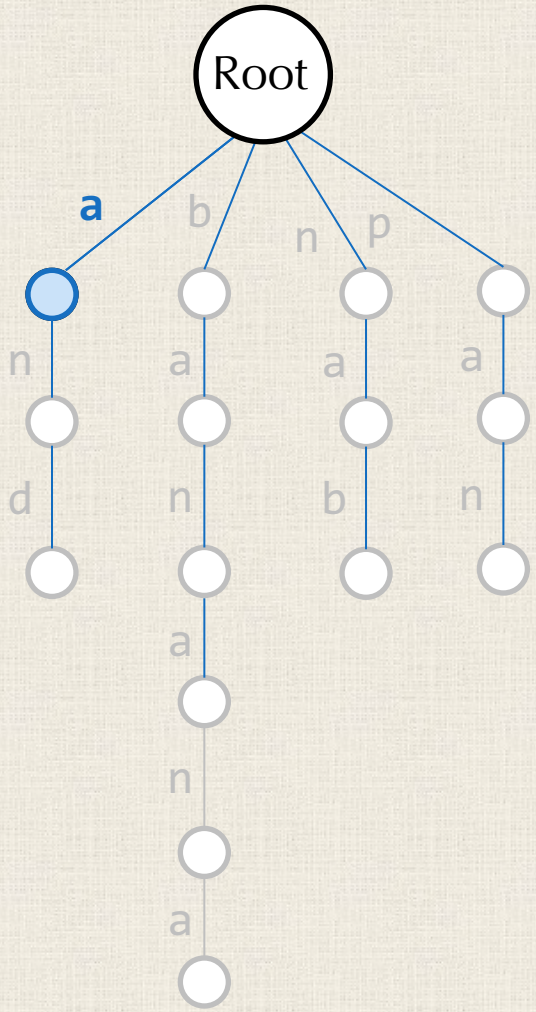
nab

antenna

bandana

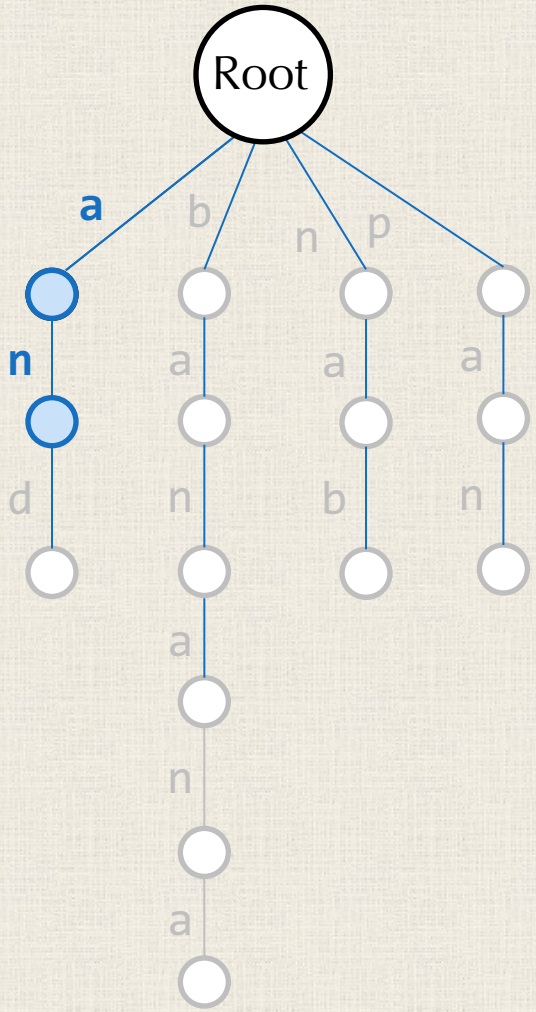
ananas

nana



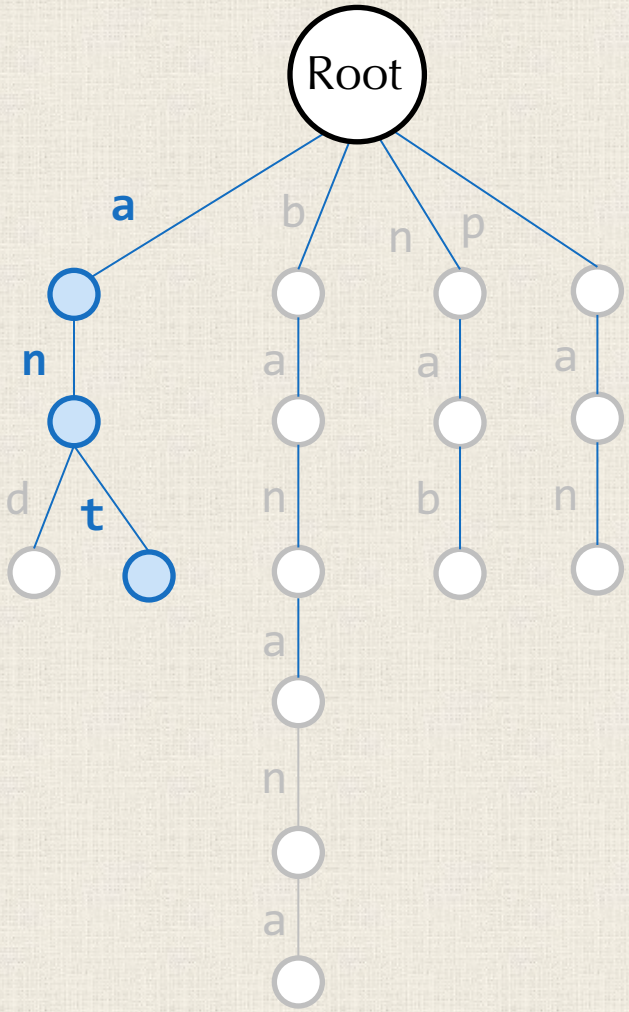
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



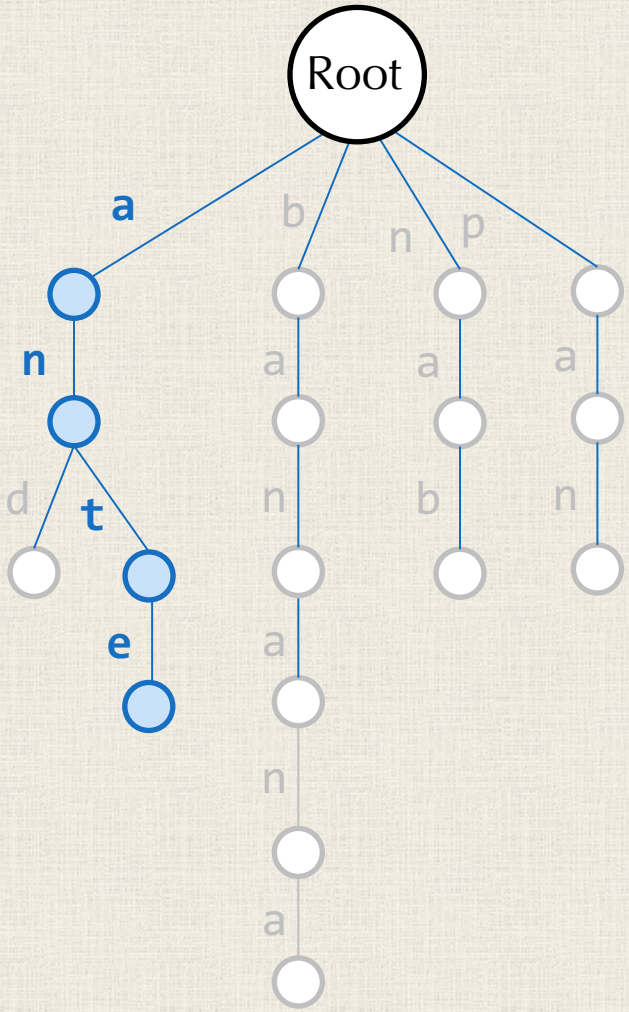
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



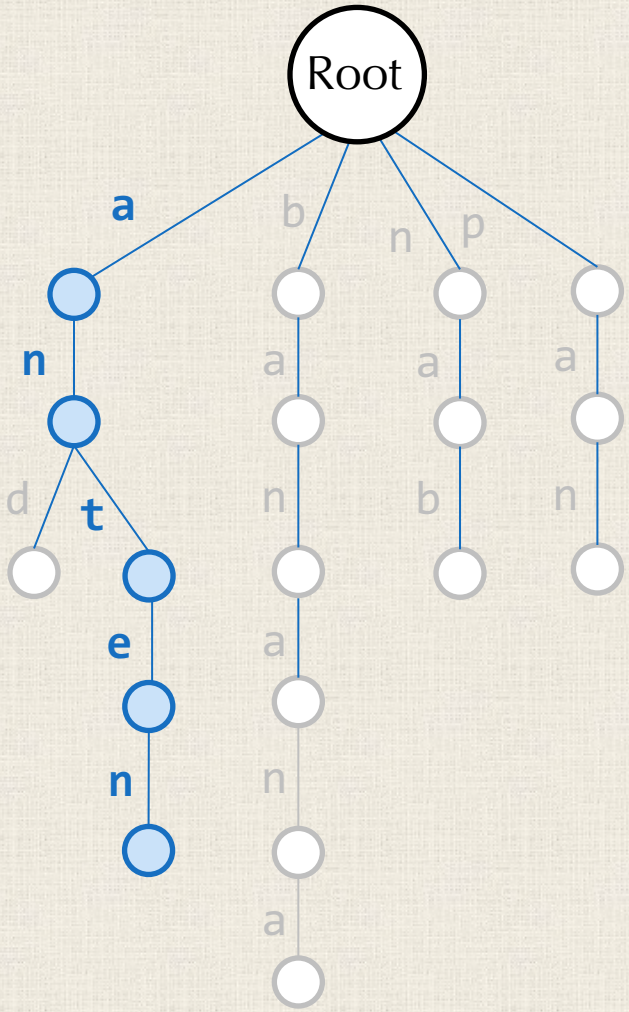
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



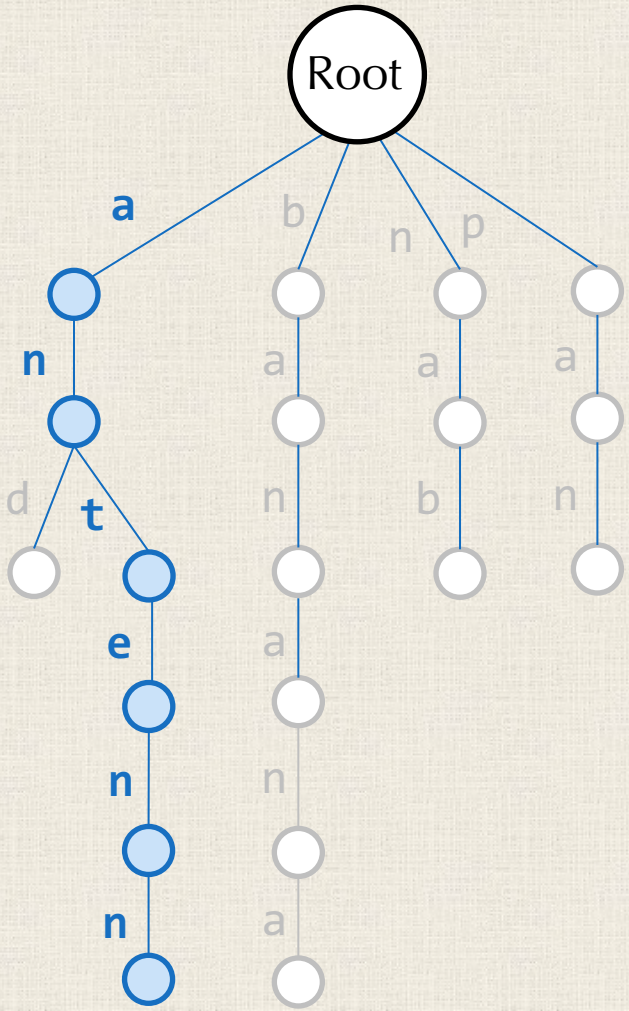
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



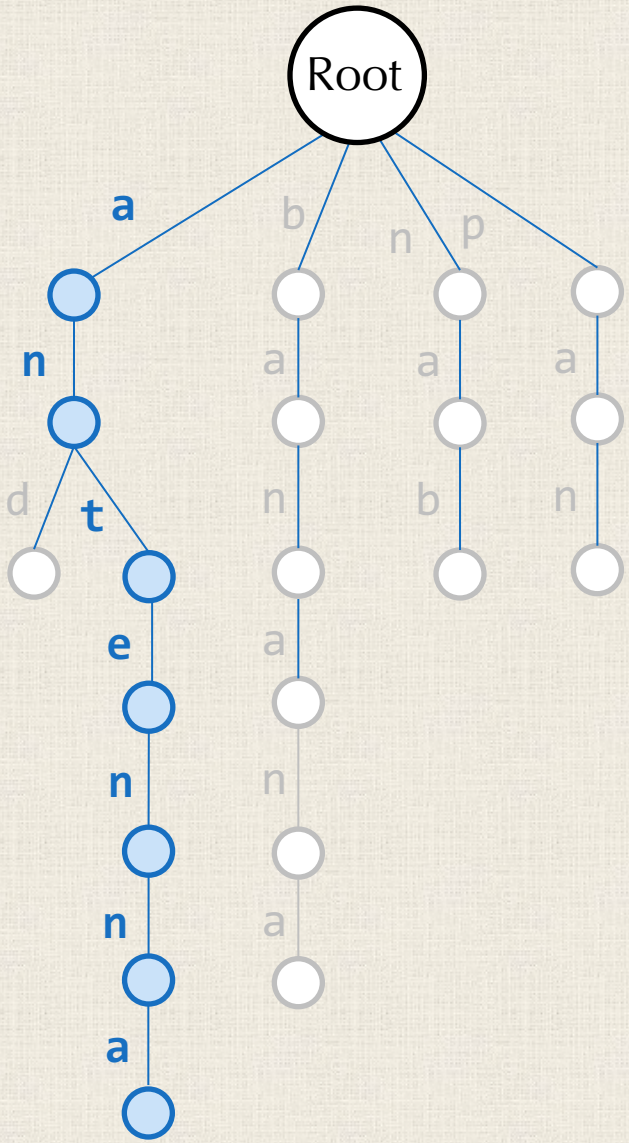
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



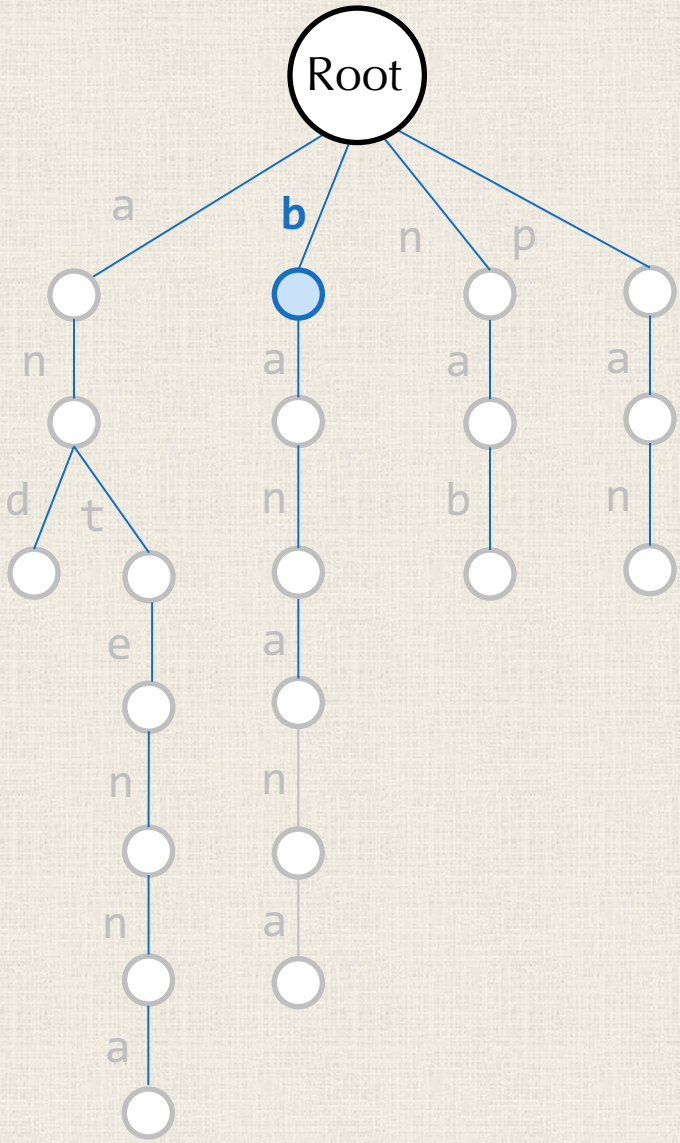
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



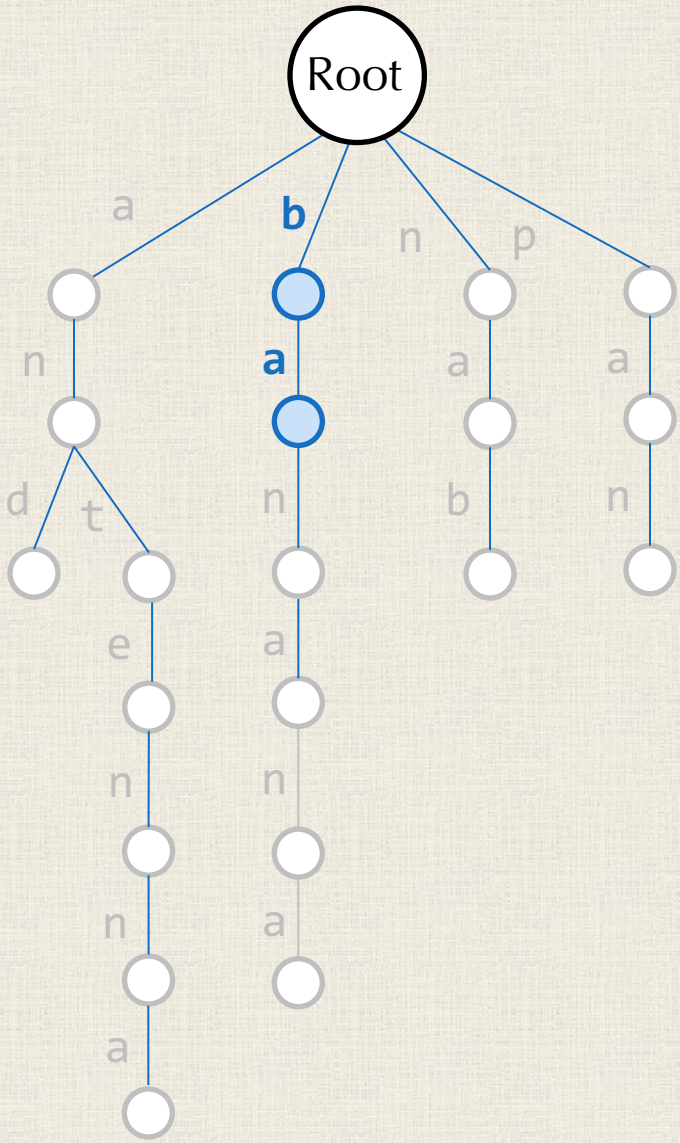
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



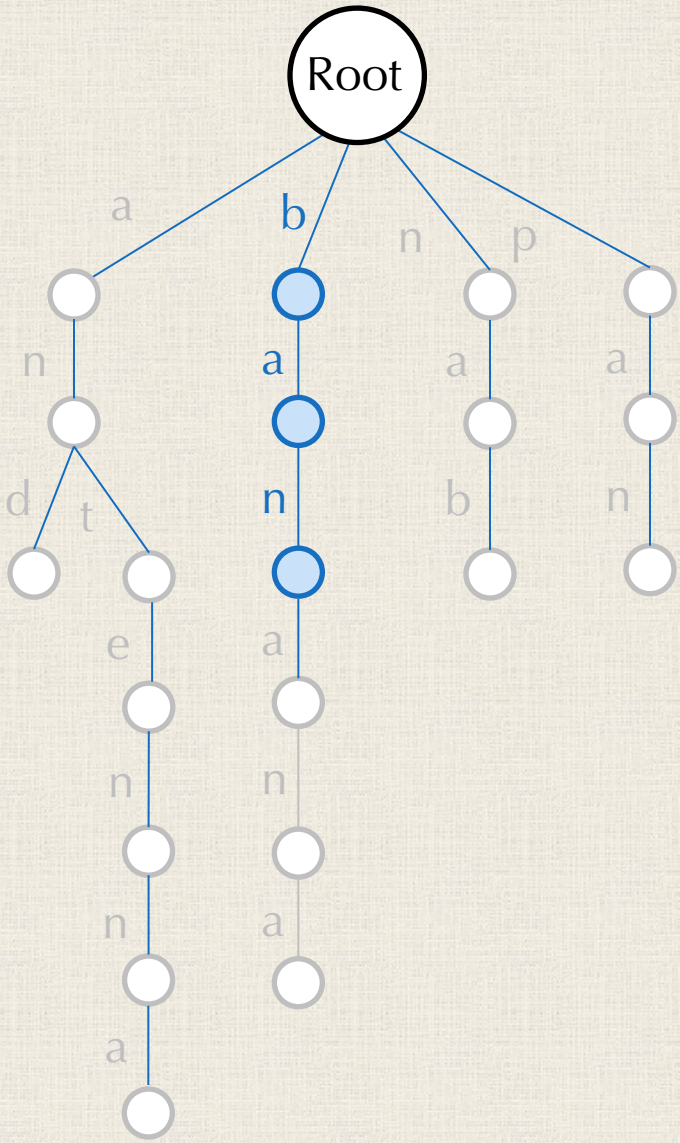
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



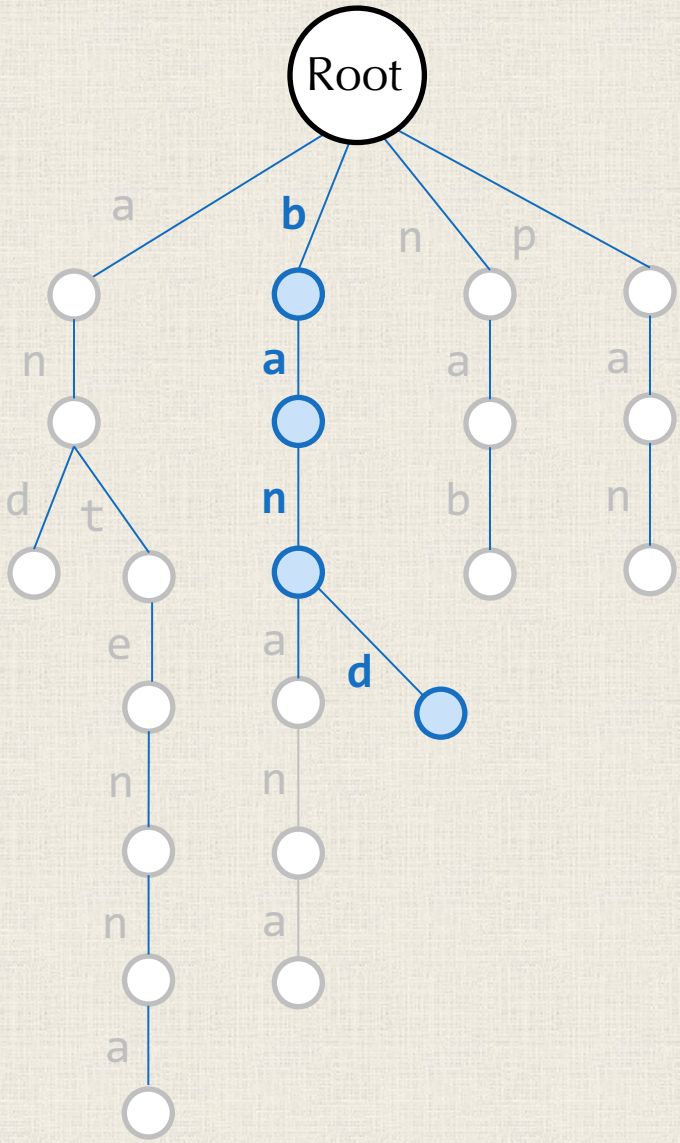
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



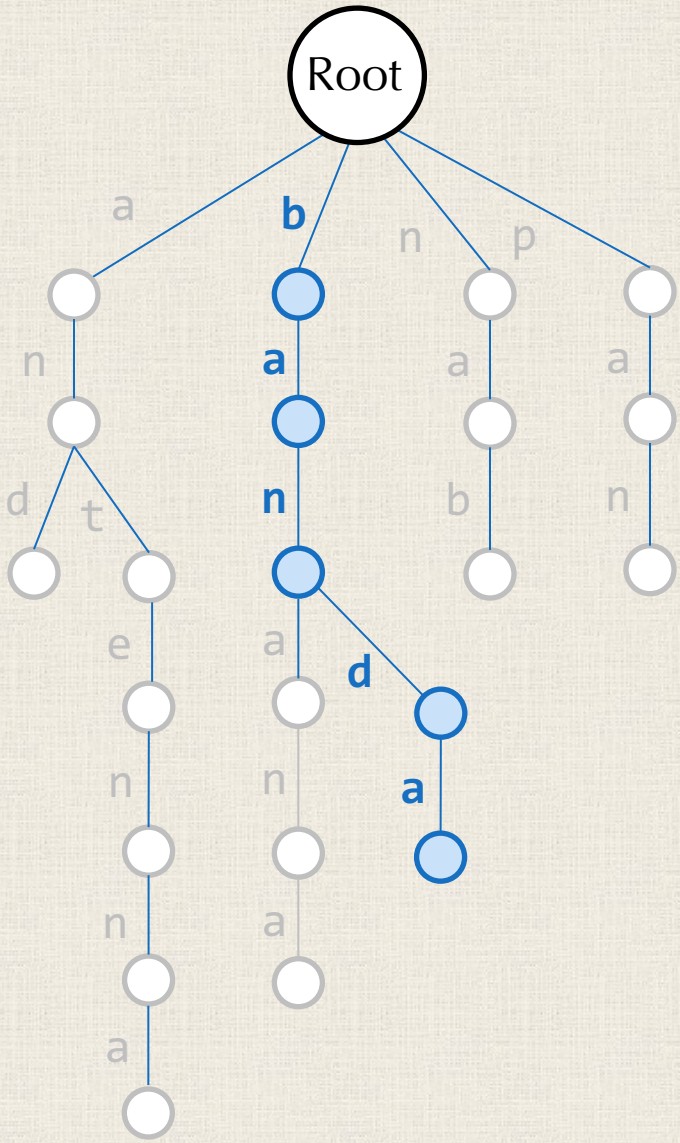
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



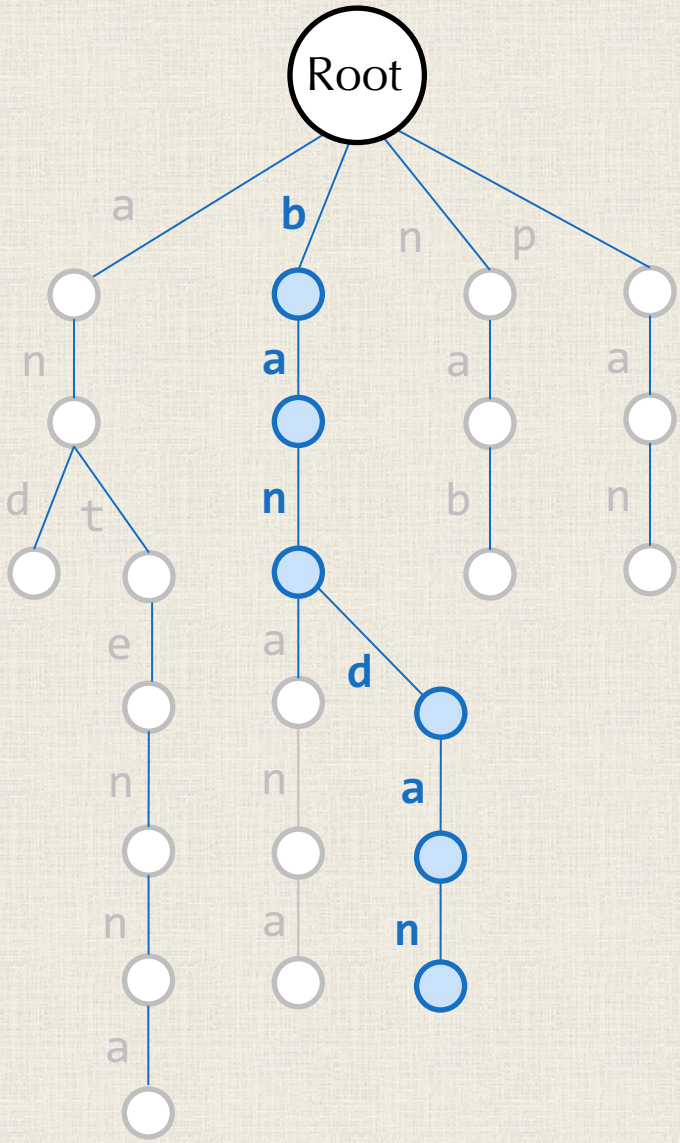
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



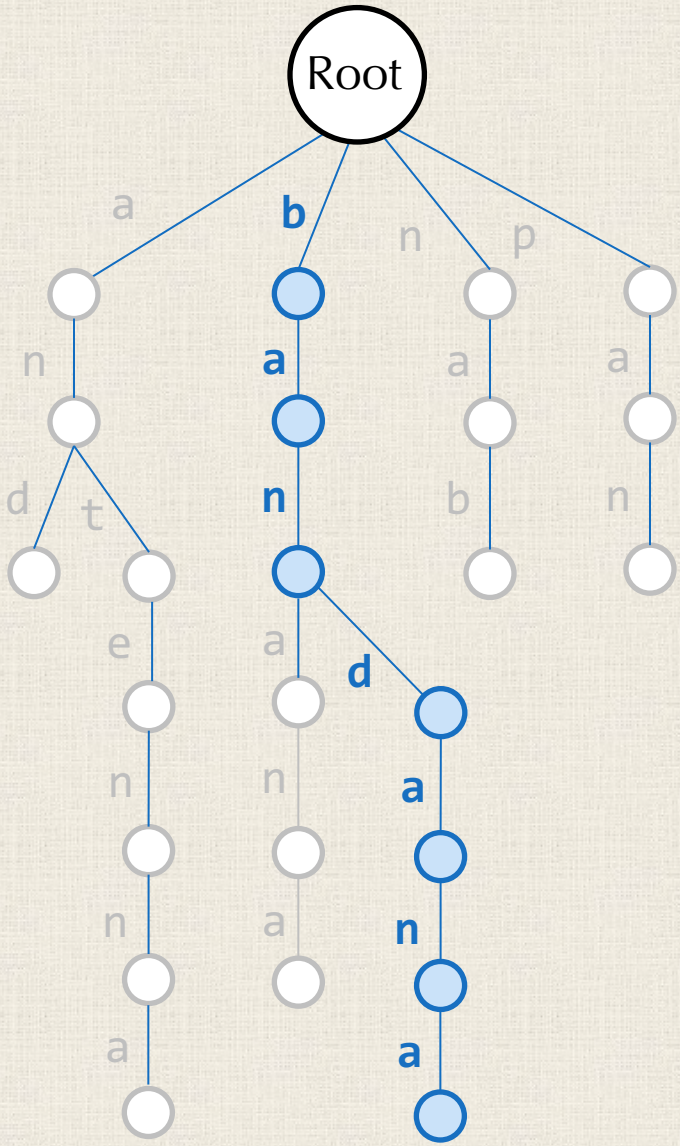
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



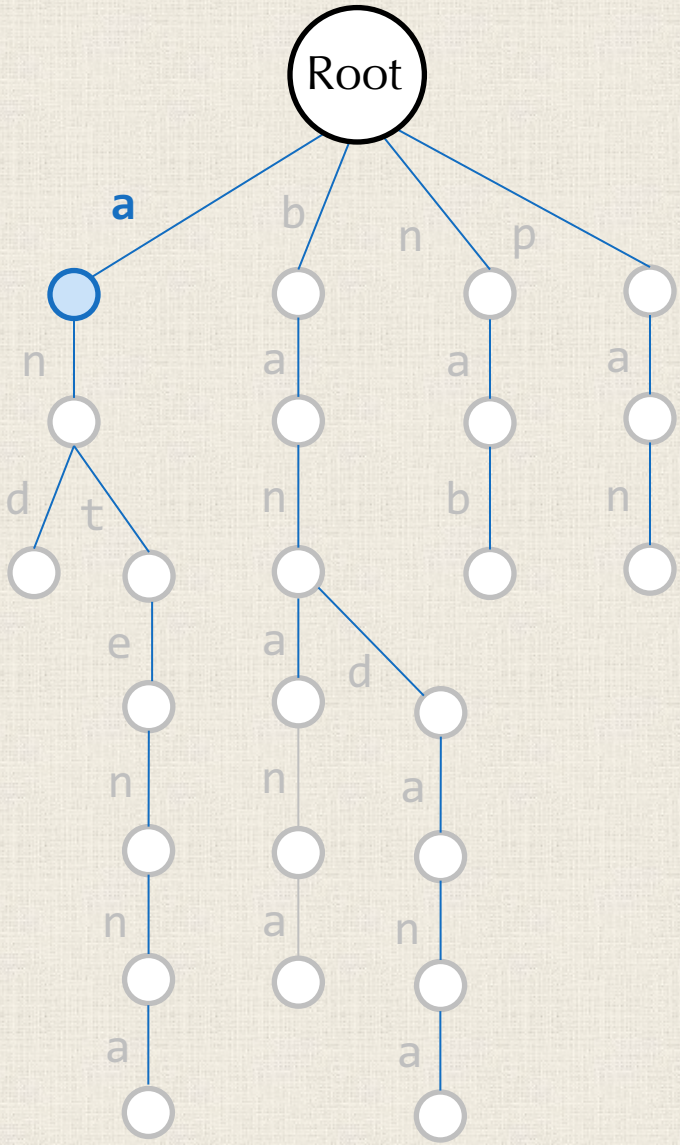
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



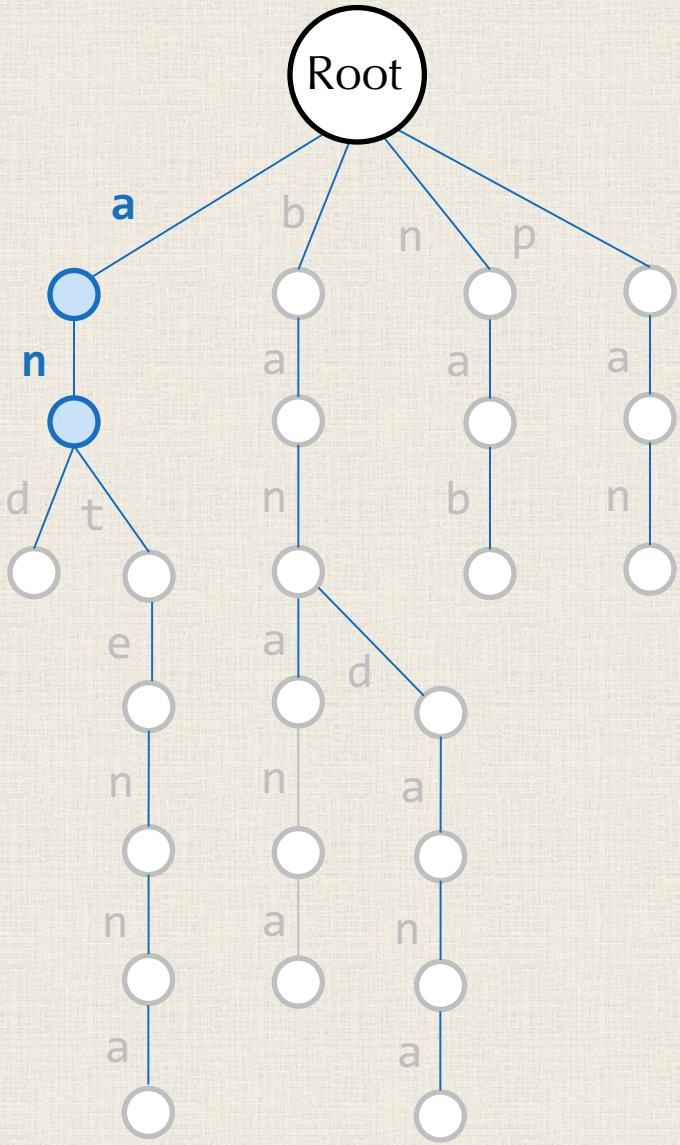
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



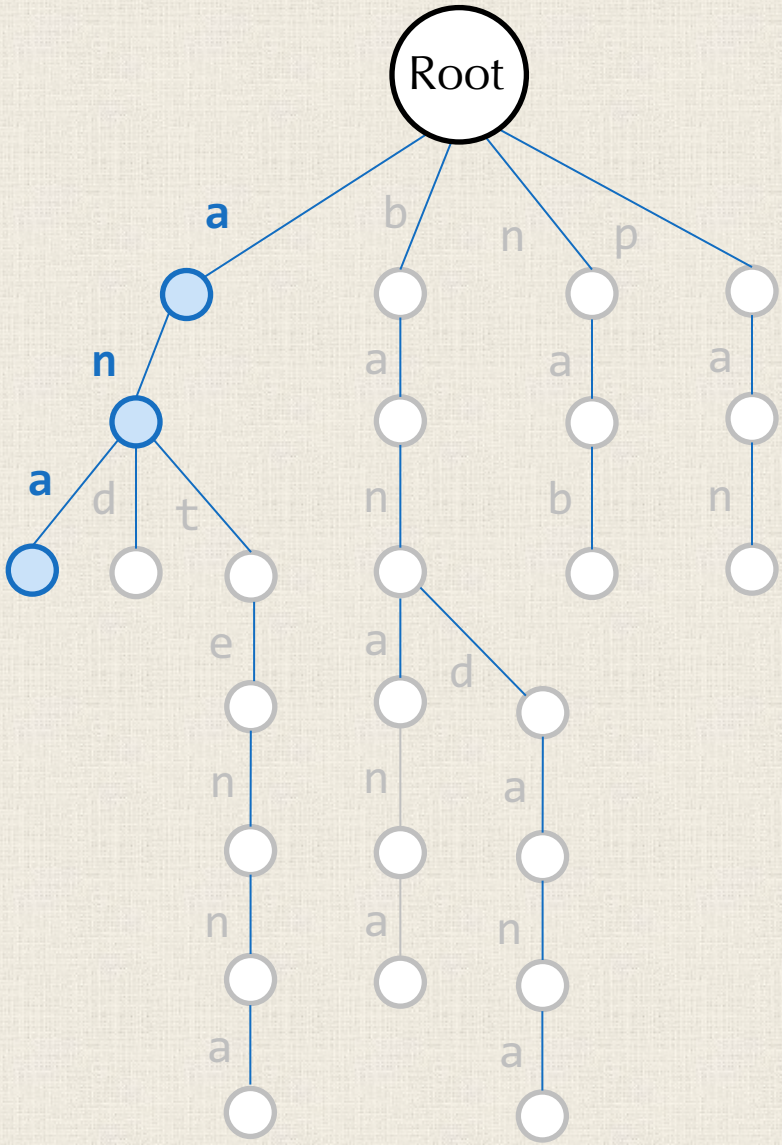
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



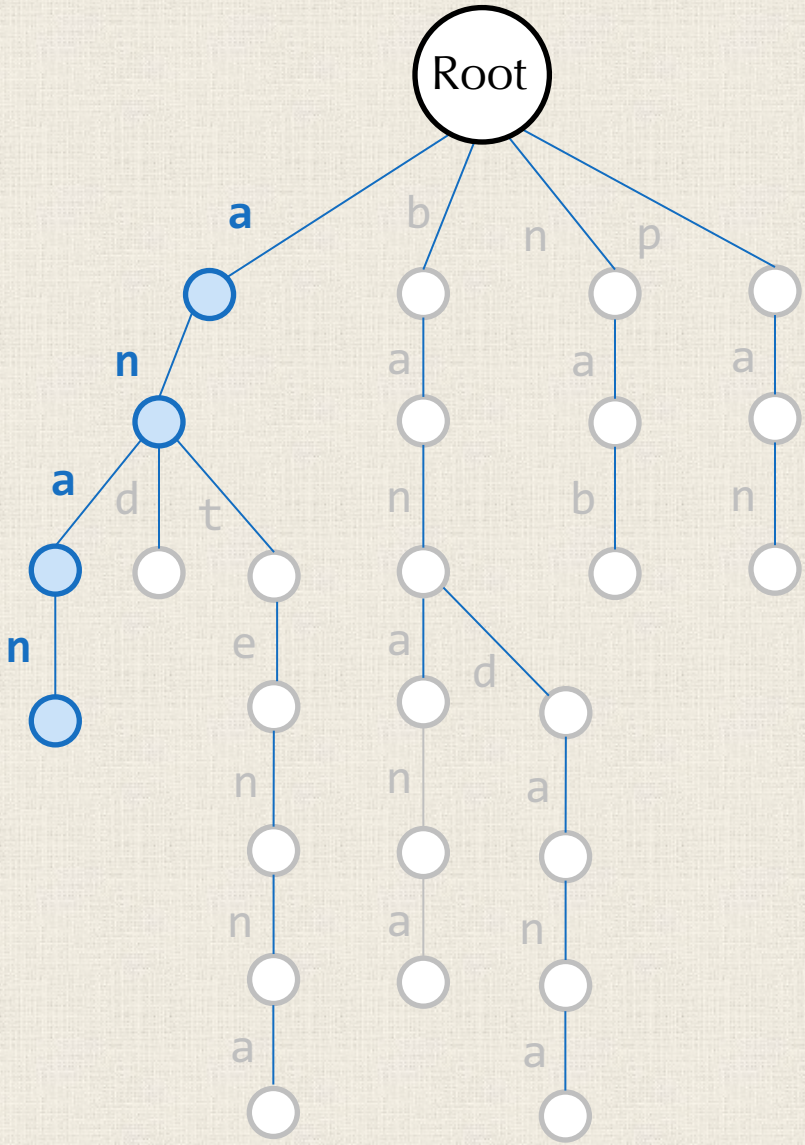
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



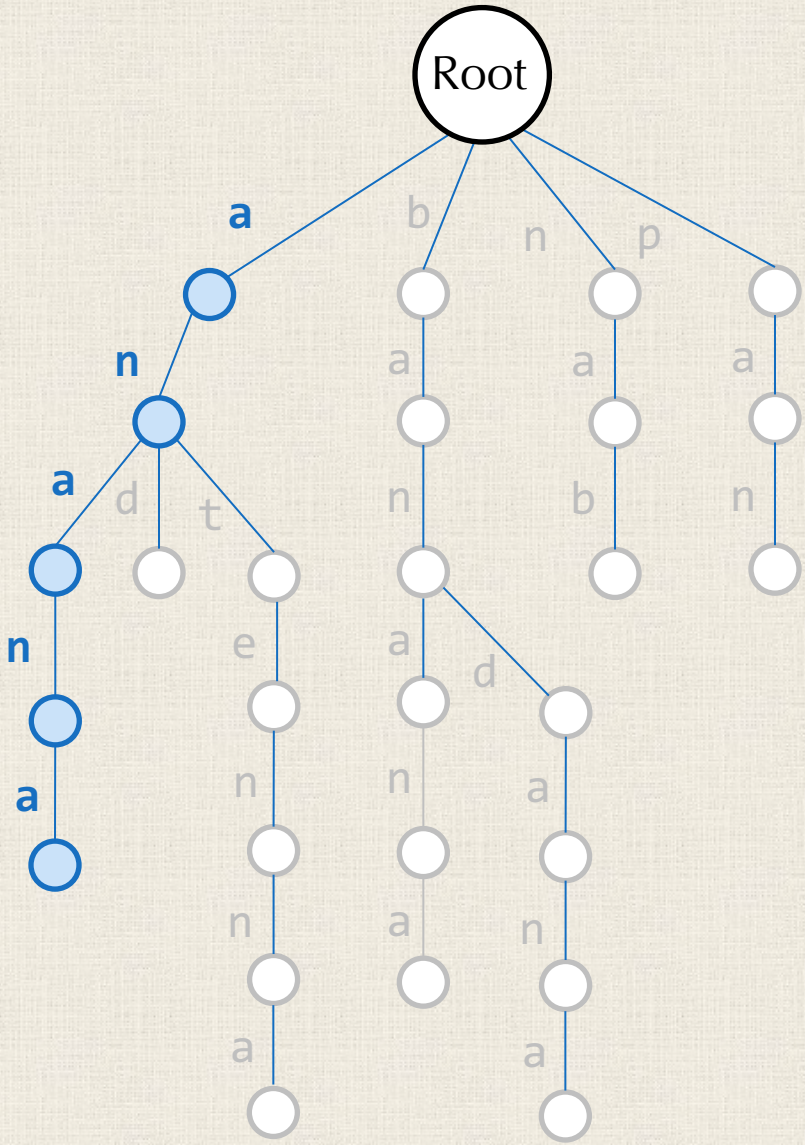
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



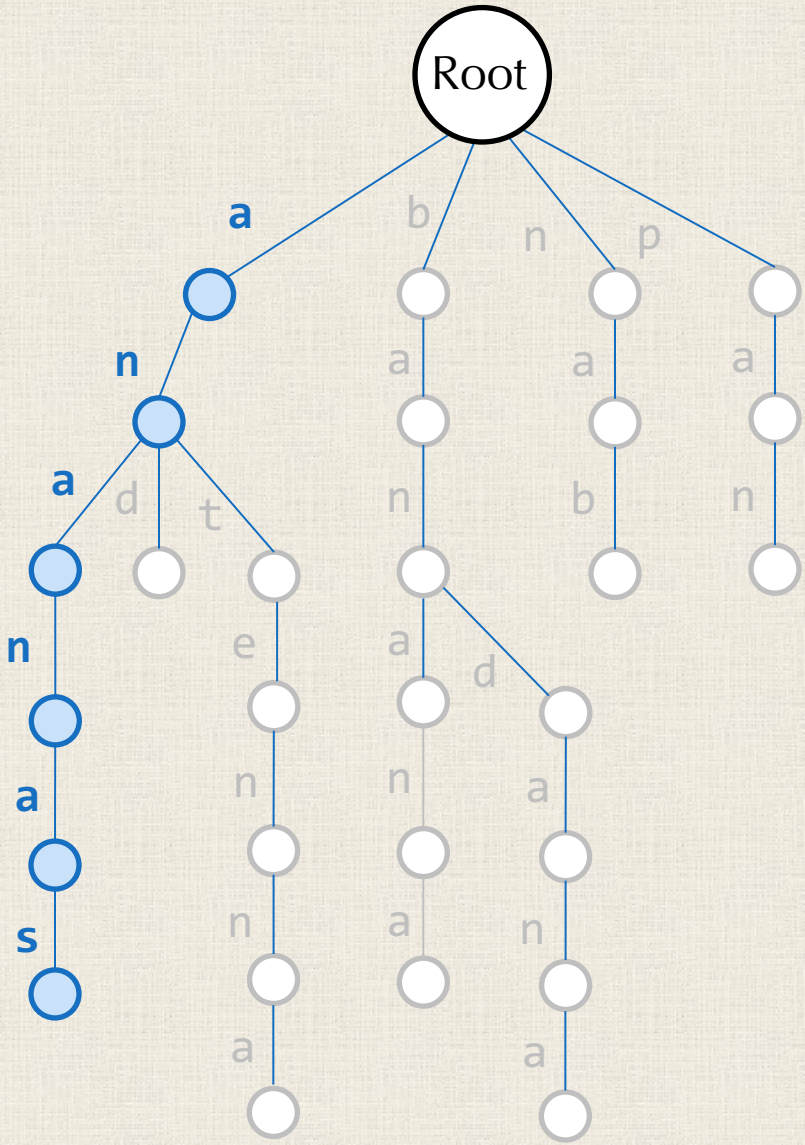
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



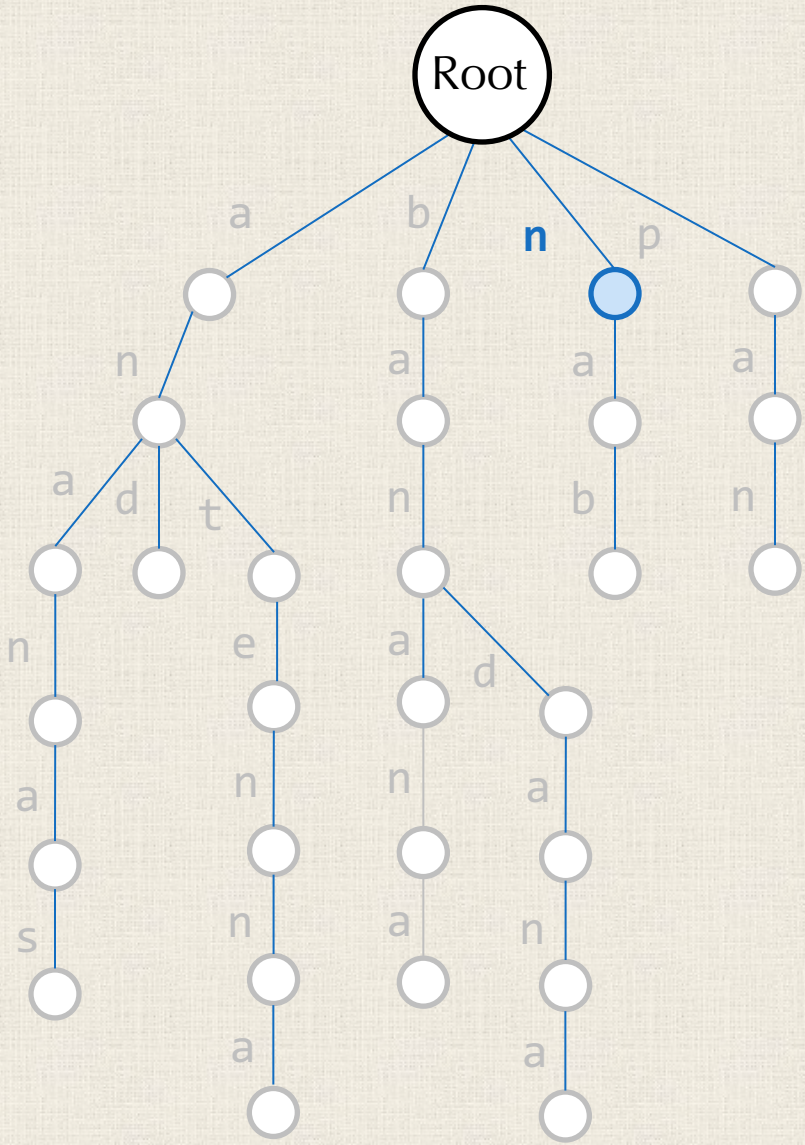
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



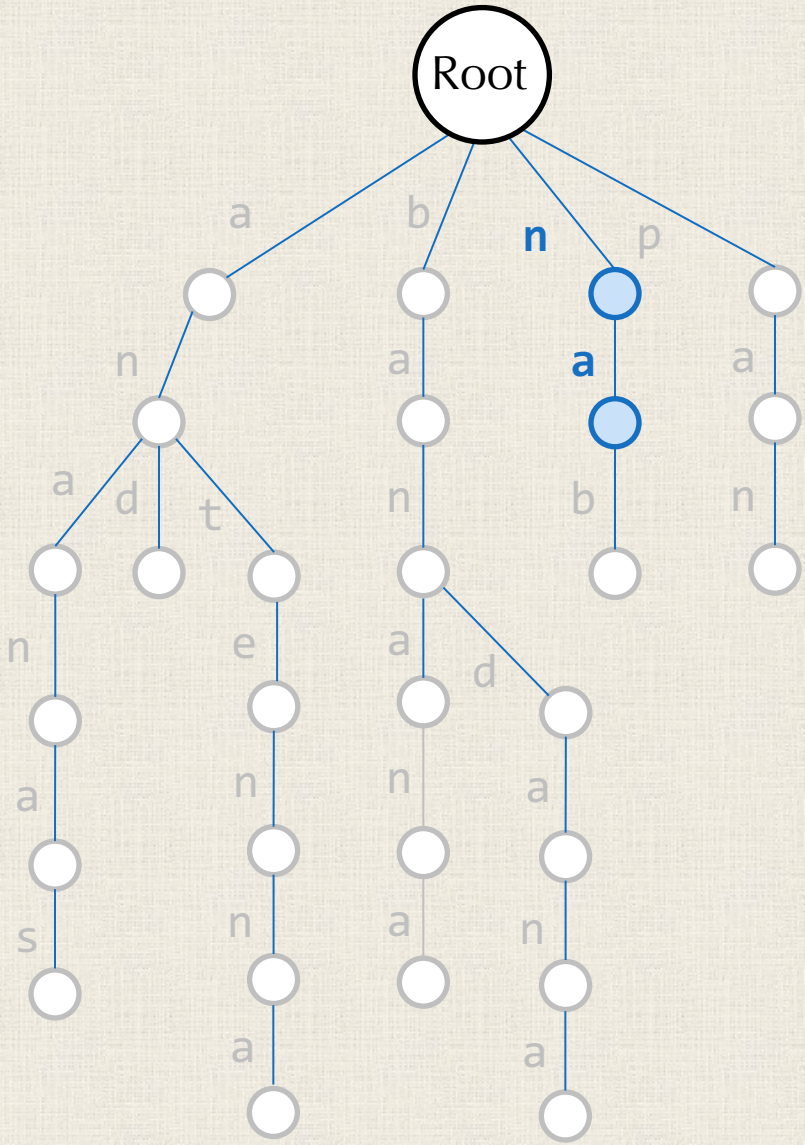
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



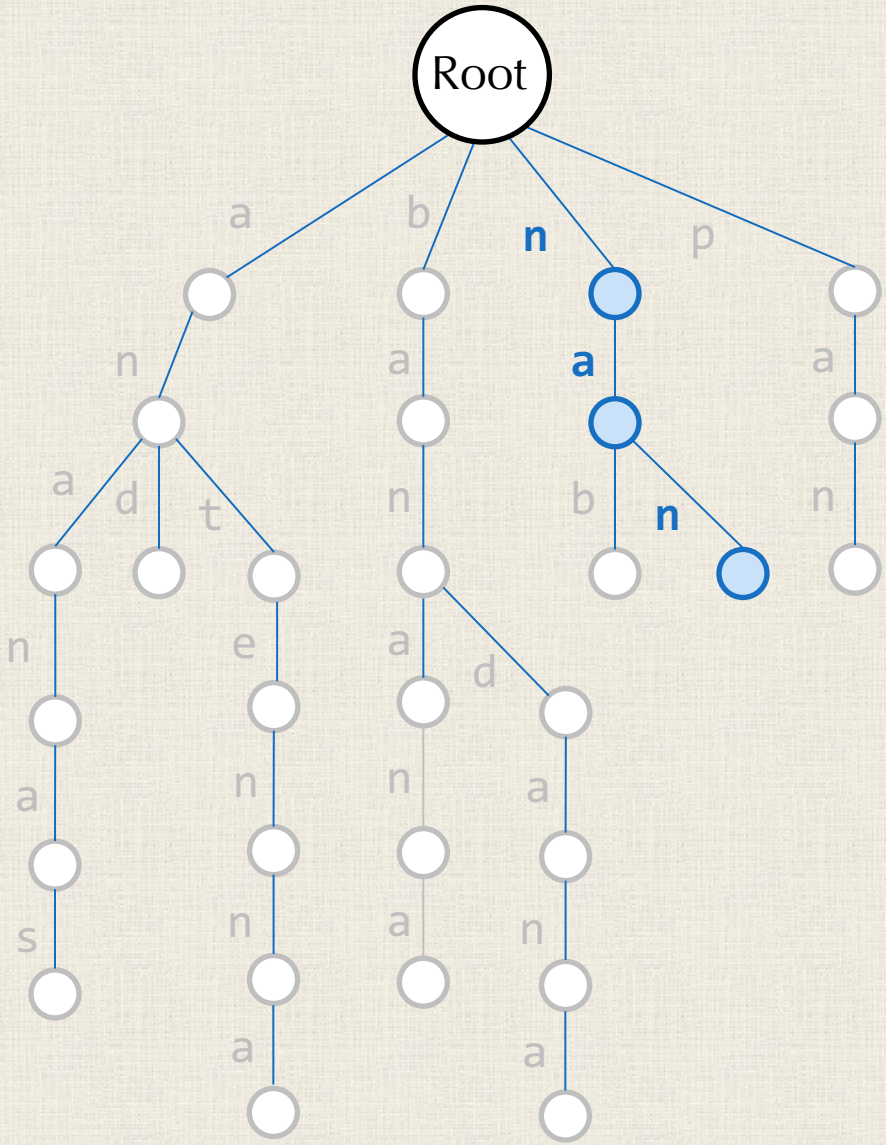
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



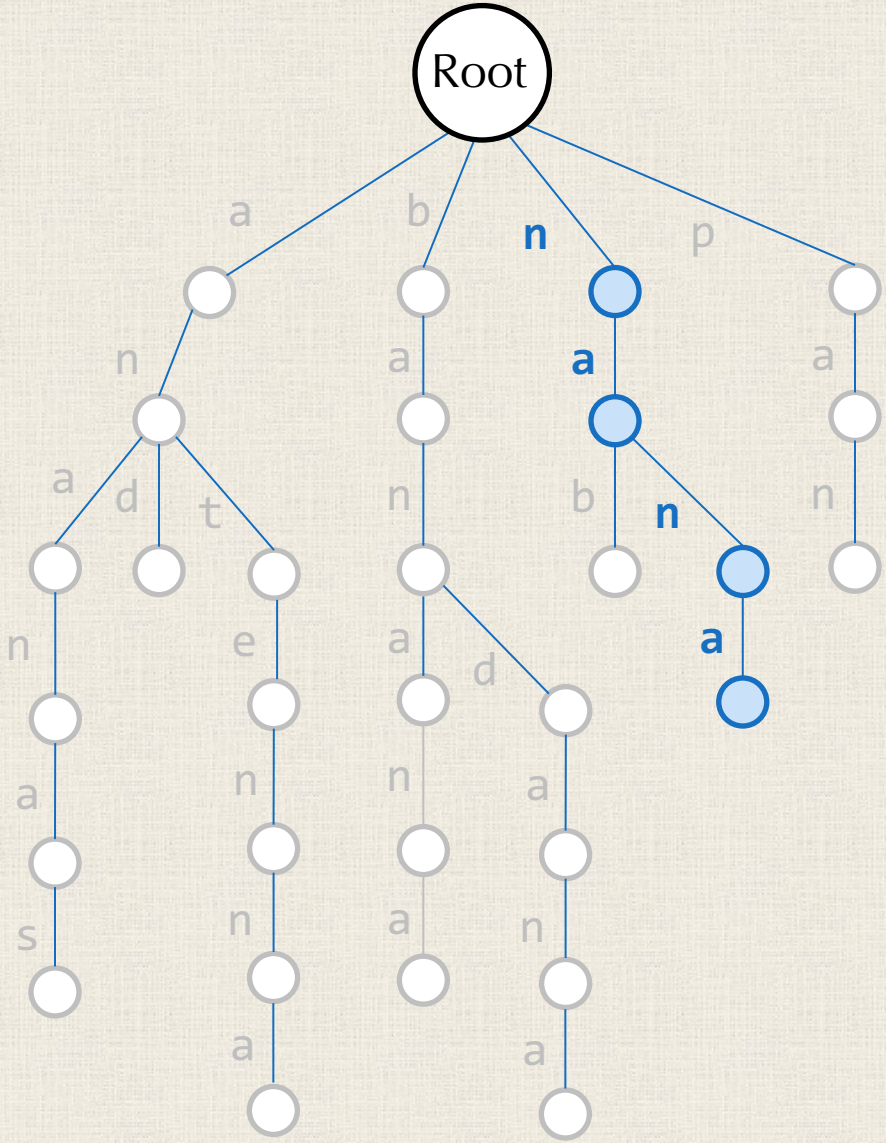
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana



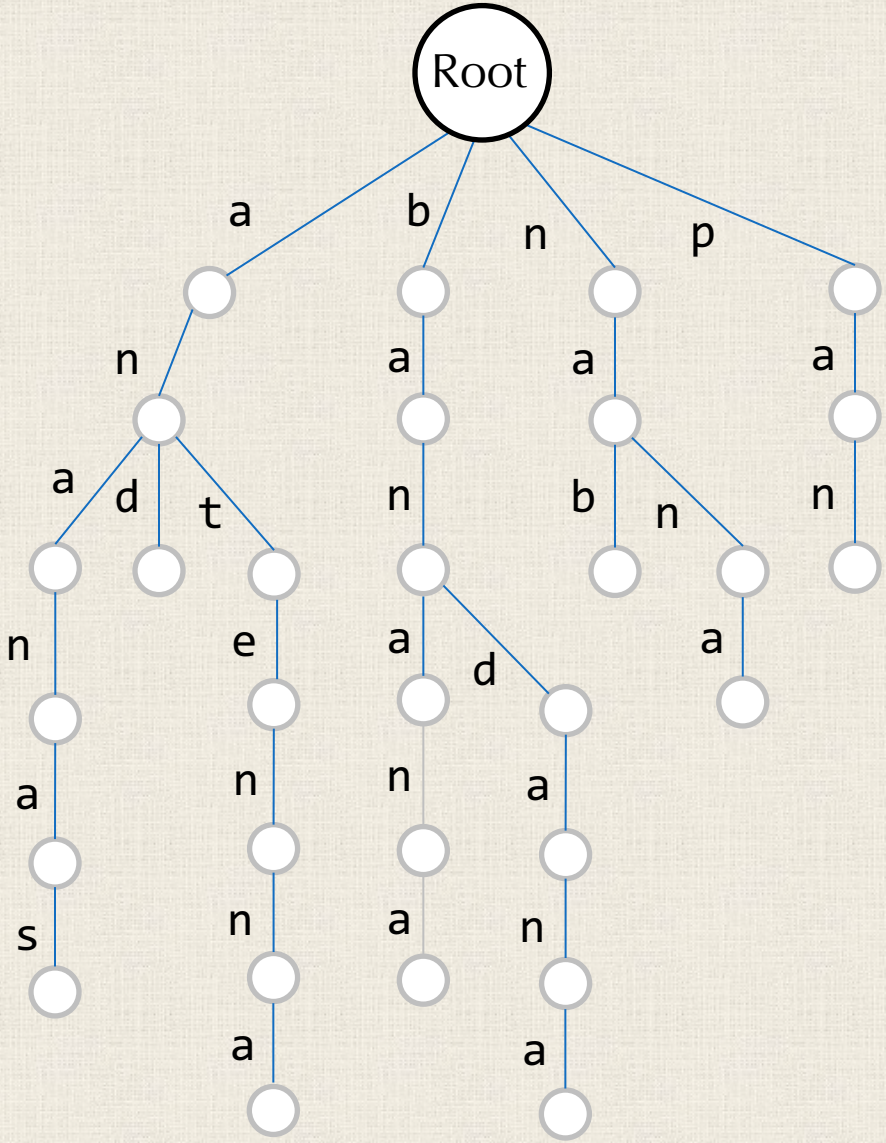
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana

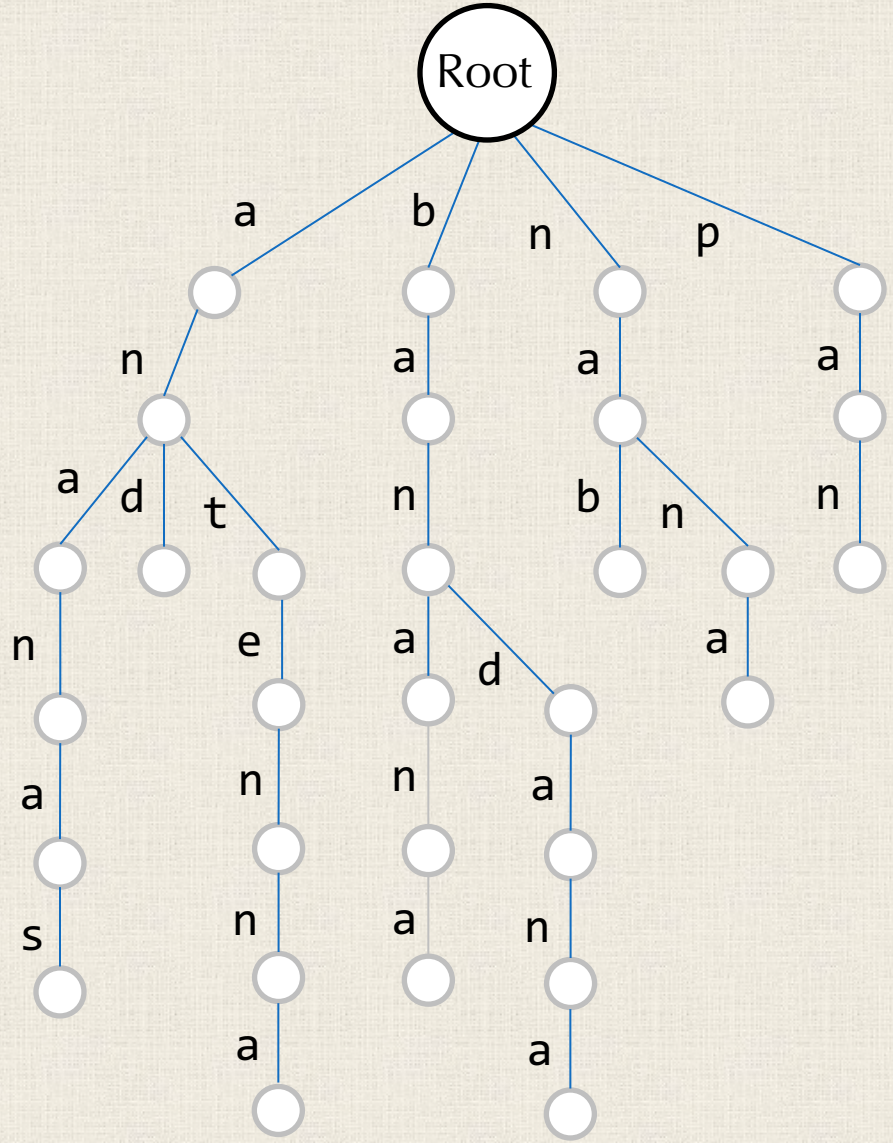


Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana

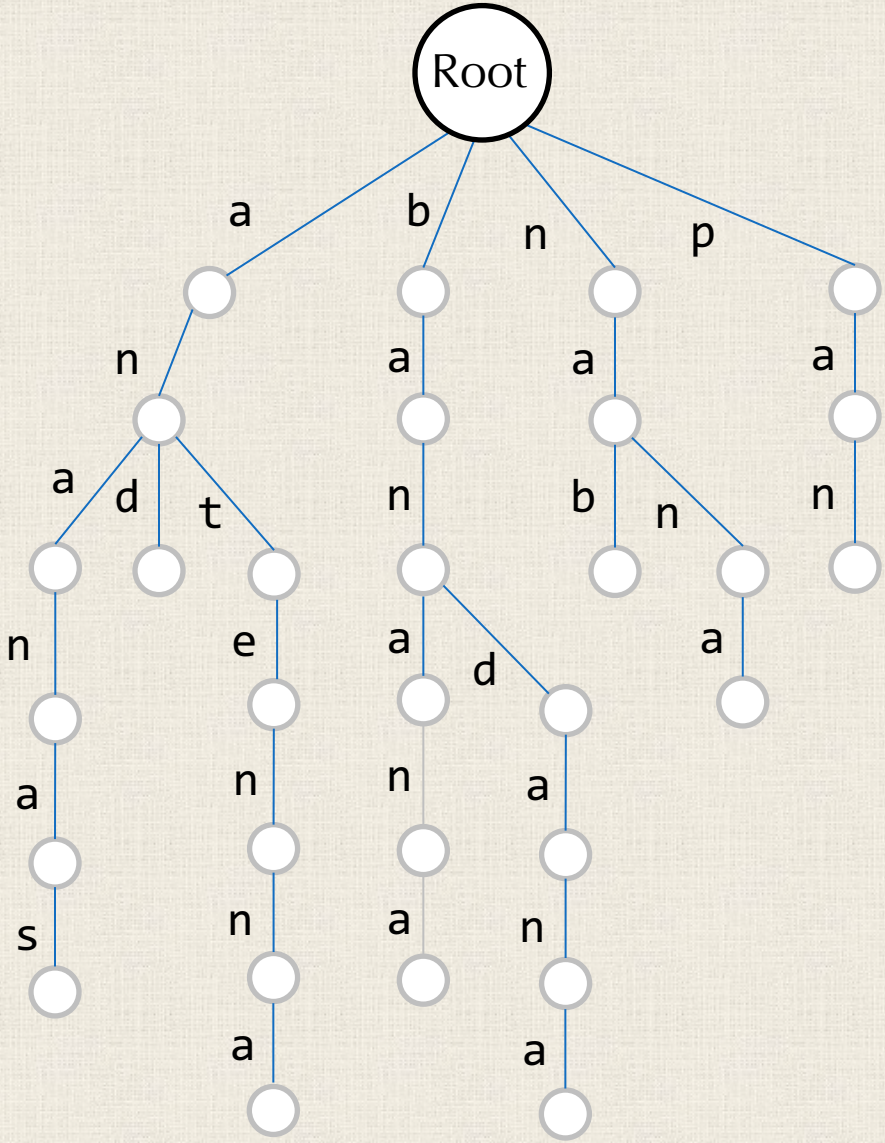


STOP: How can we use the trie for pattern matching? (e.g, say *Text* = "panamabananas").



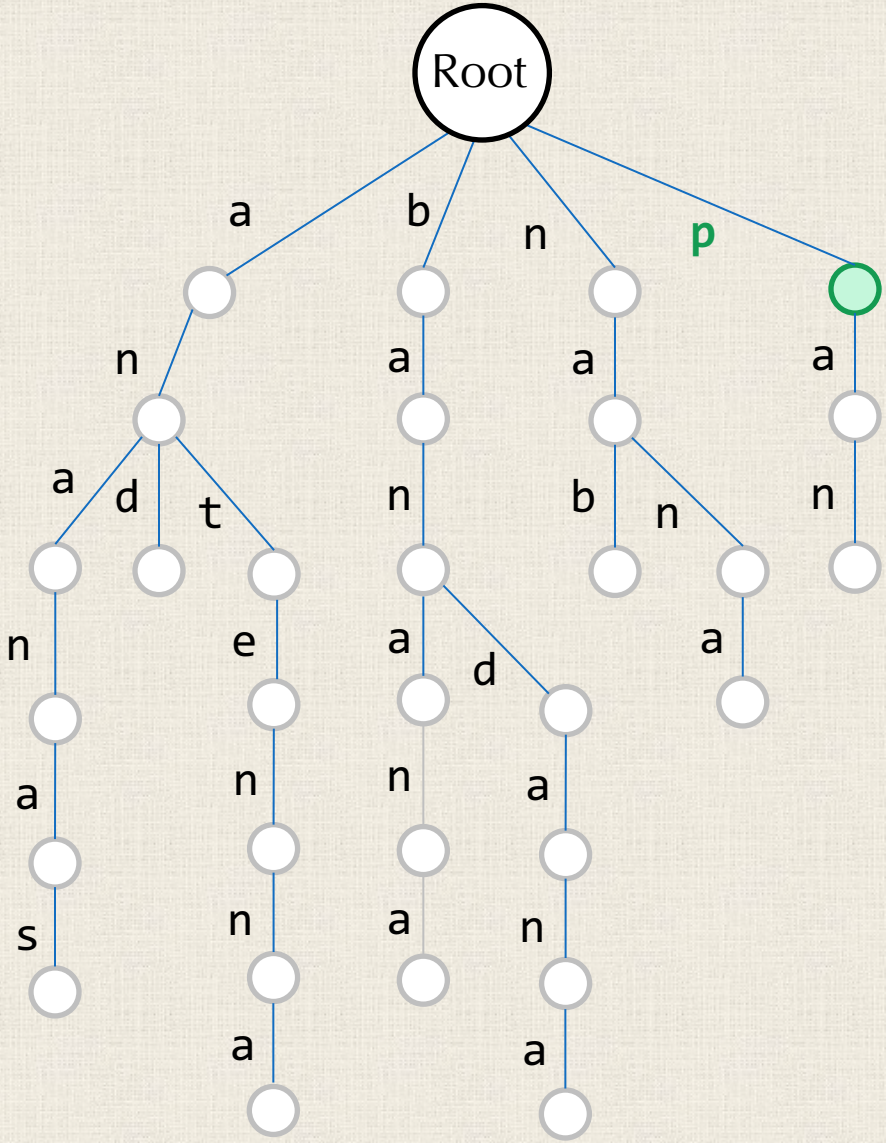
Answer: We can test whether any string in *Patterns* matches a prefix of *Text* by "sliding it" down the *Text*.

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



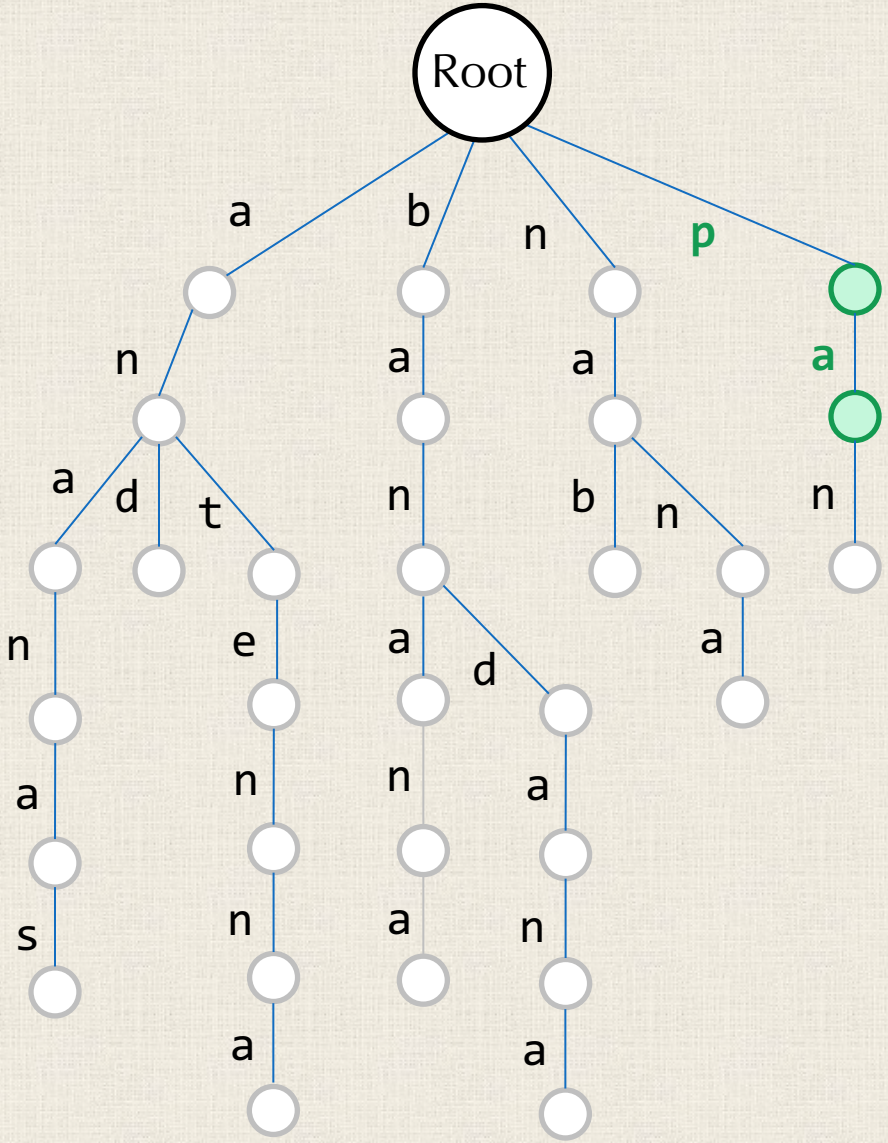
Pattern	Positions

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



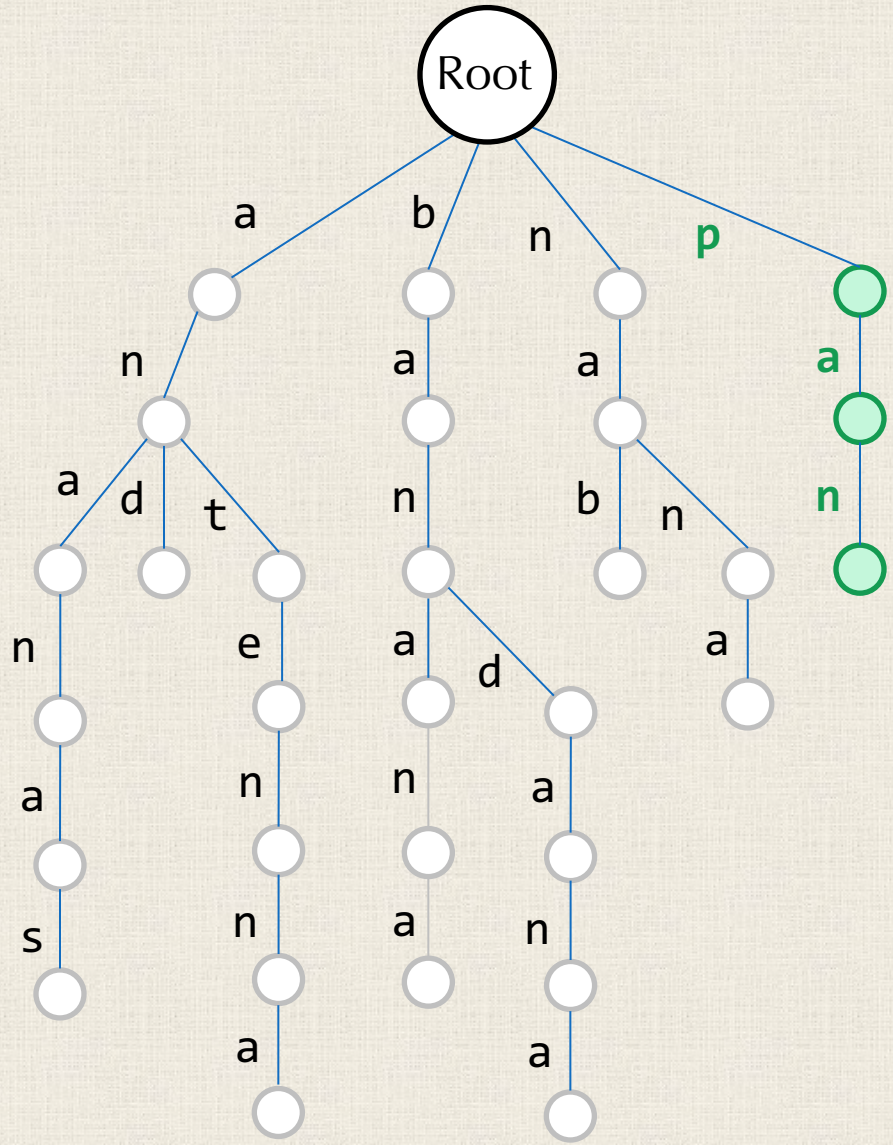
Pattern	Positions

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



Pattern	Positions

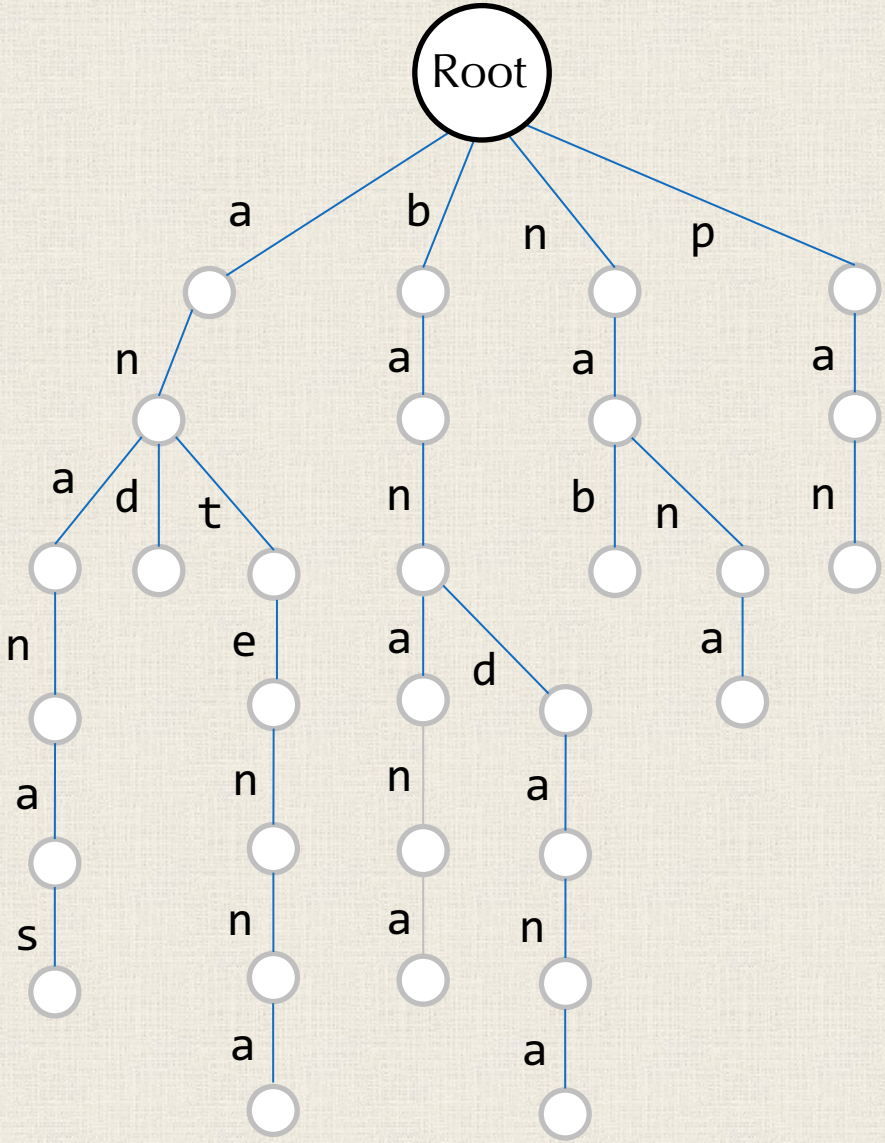
panamabananas



We have found a pattern match – now we just need to "slide down" and start the search at the next symbol of *Text*.

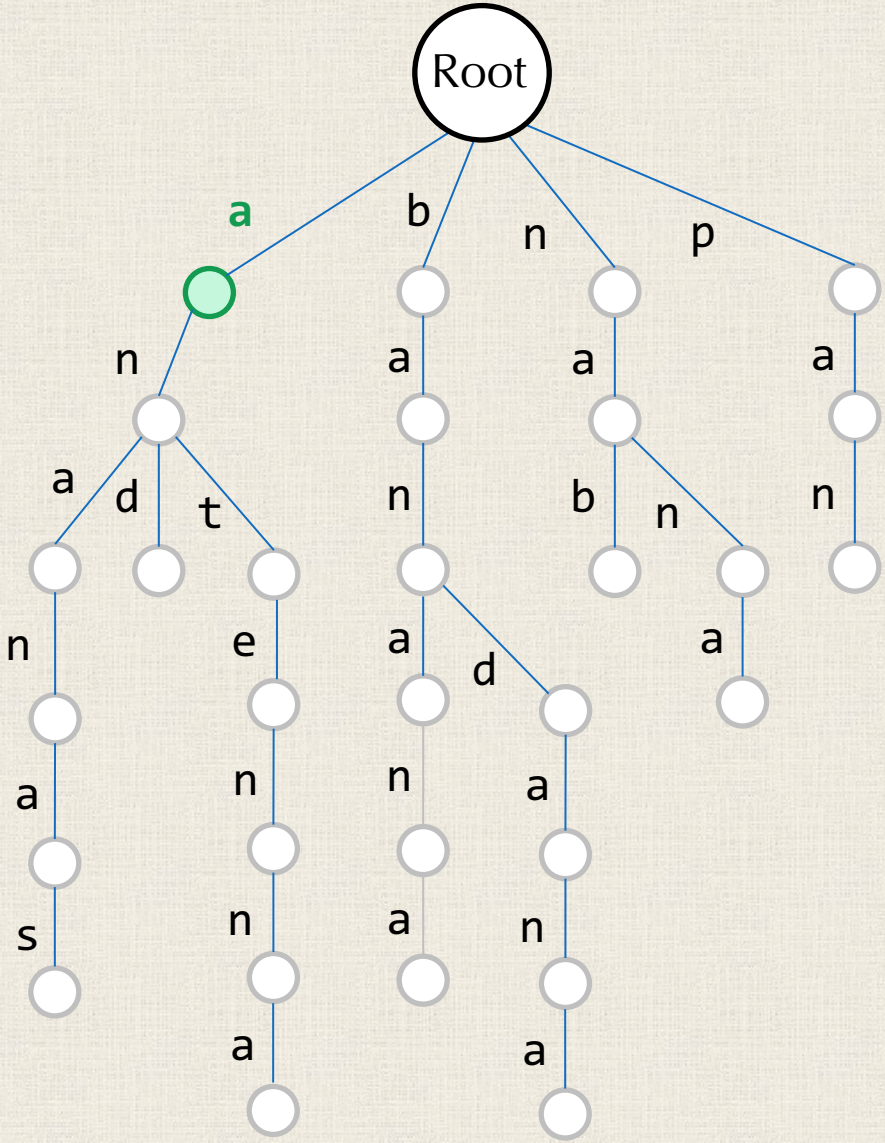
Pattern	Positions
pan	0

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



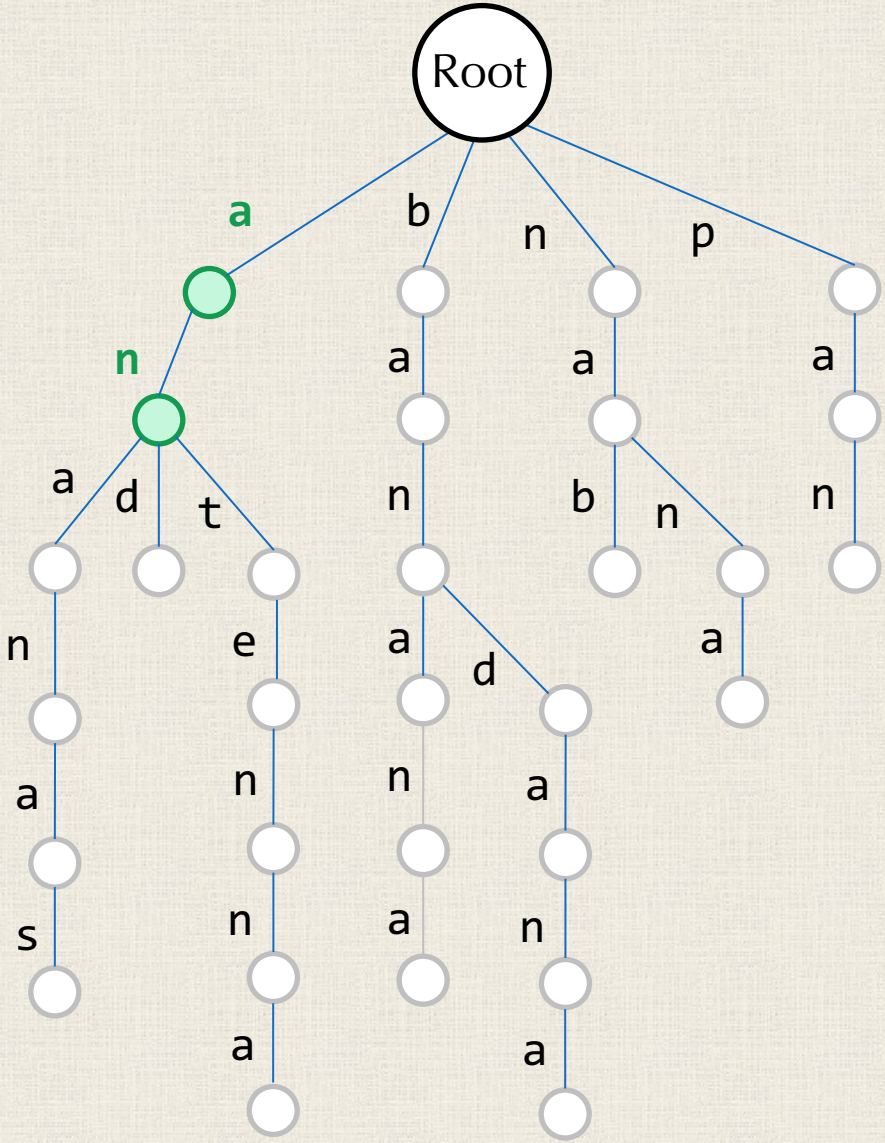
Pattern	Positions
pan	0

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



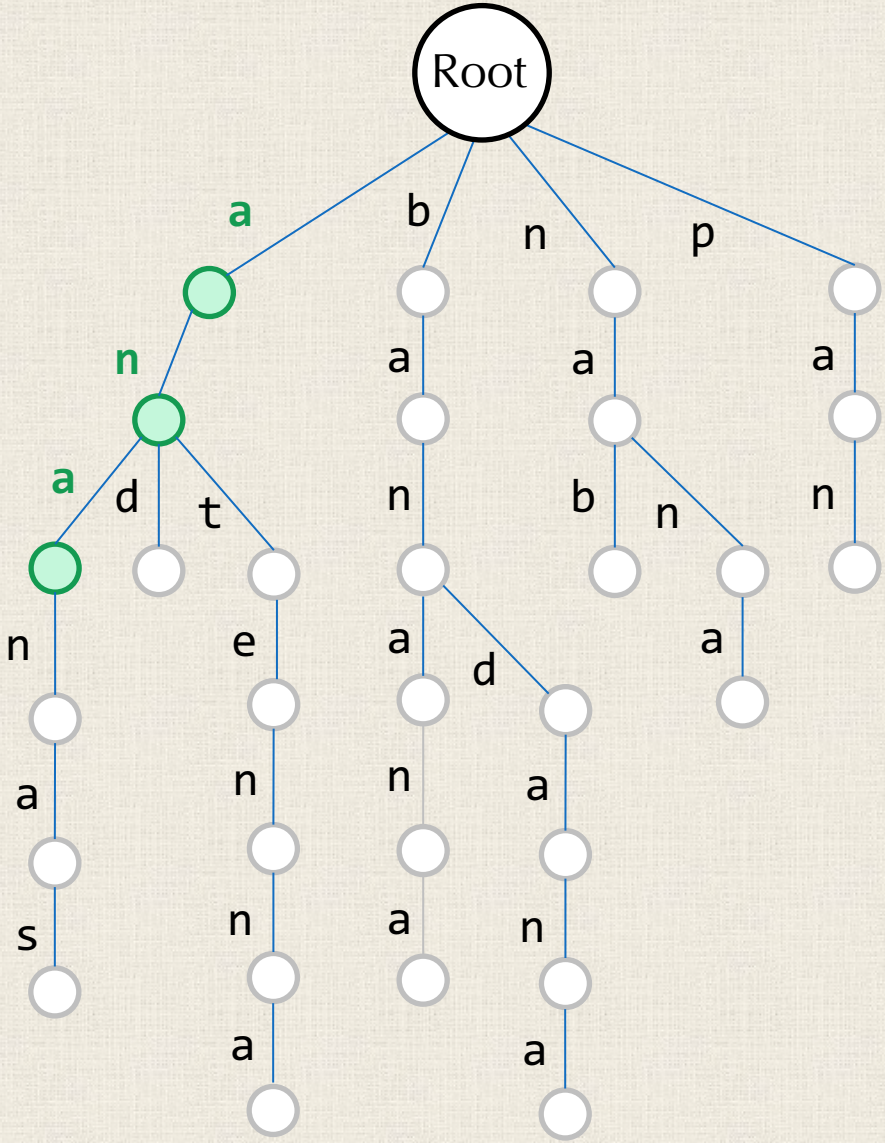
Pattern	Positions
pan	0

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



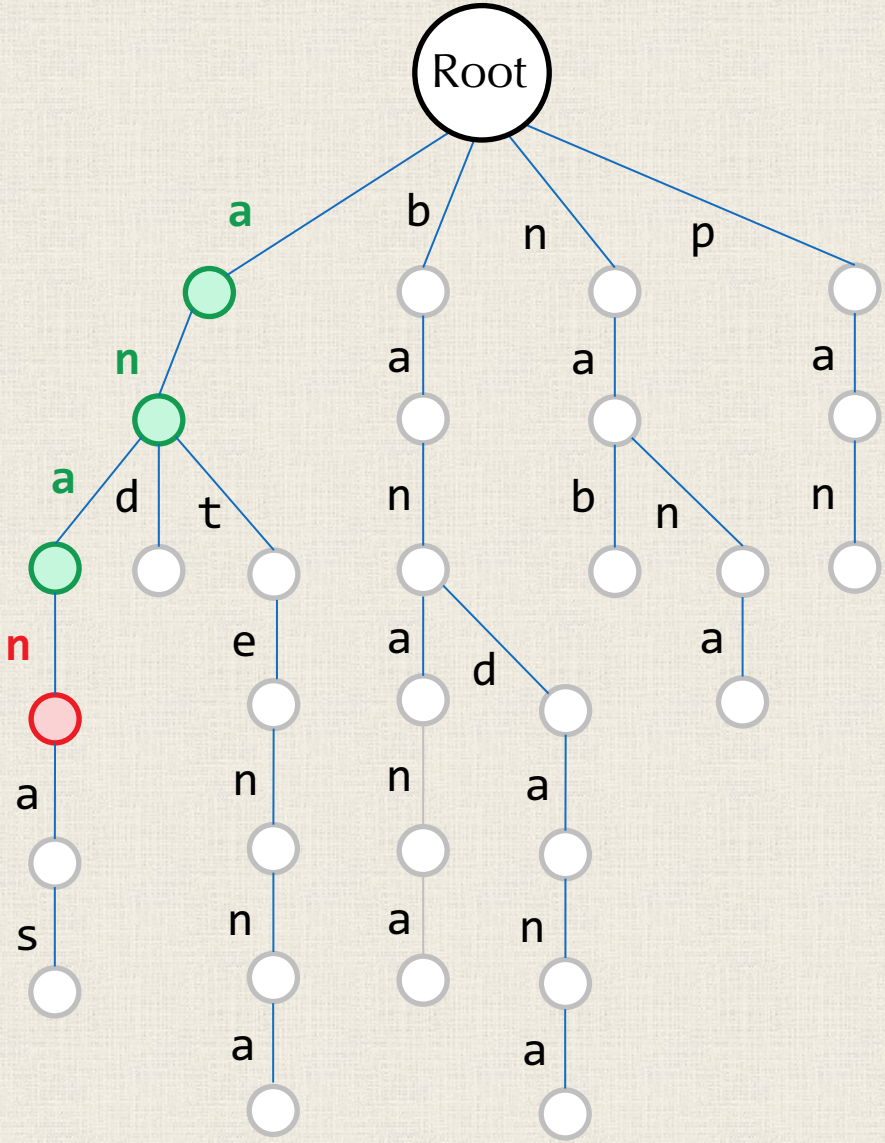
Pattern	Positions
pan	0

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



Pattern	Positions
pan	0

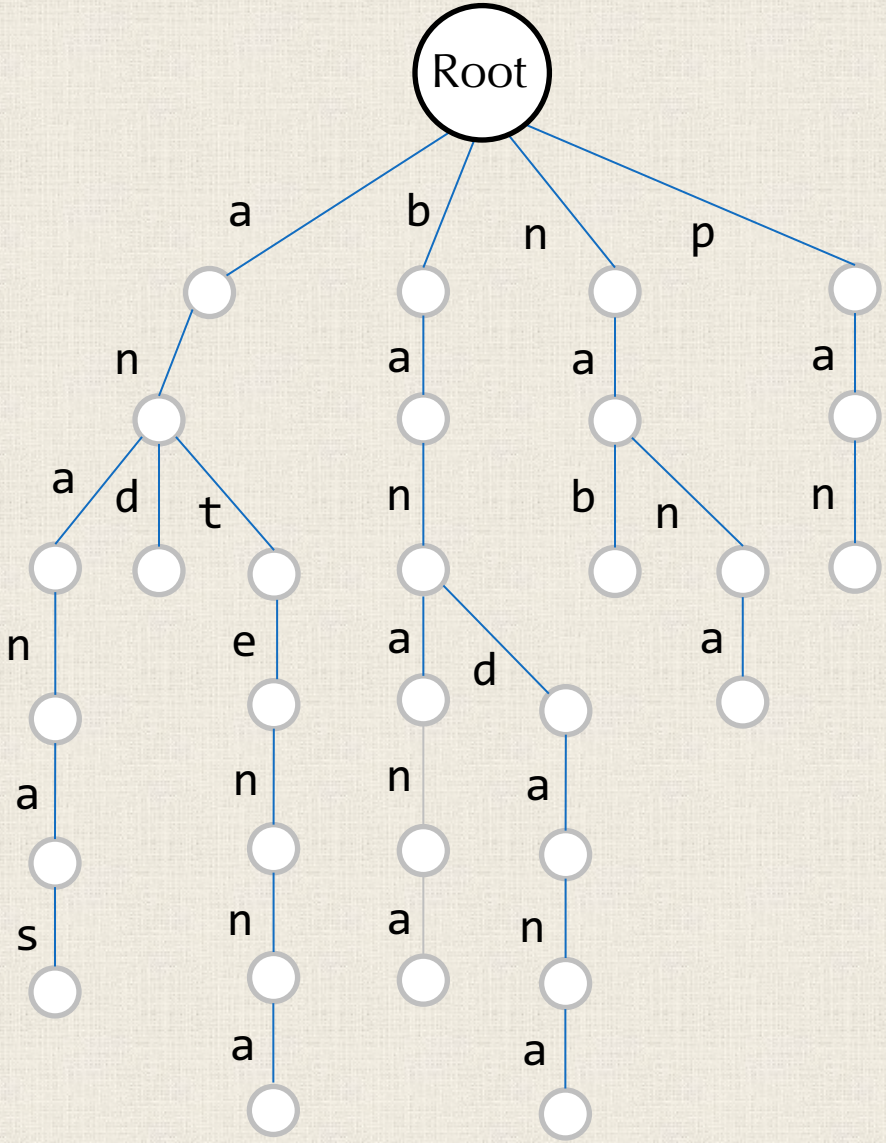
p a n a m a b a n a n a s



By how the trie is organized, we can be sure that there are no patterns matching this position of *Text*.

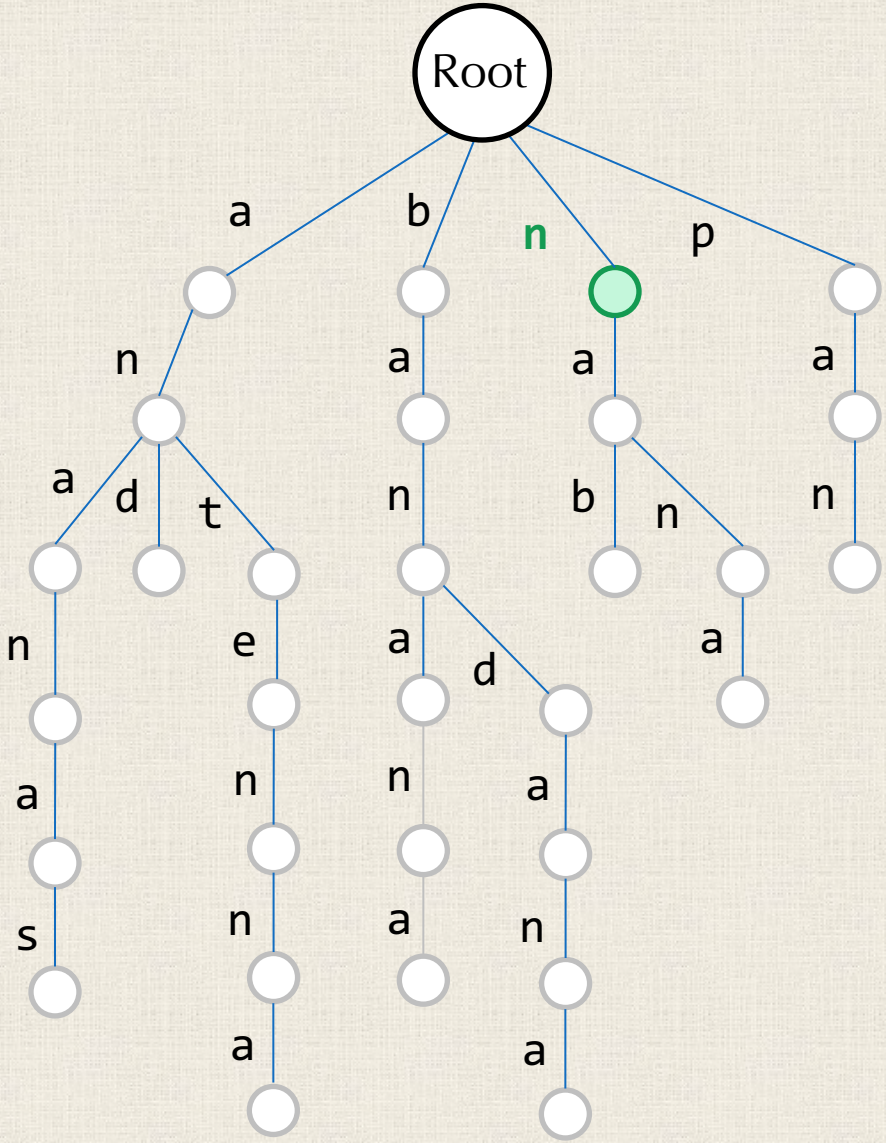
Pattern	Positions
pan	0

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



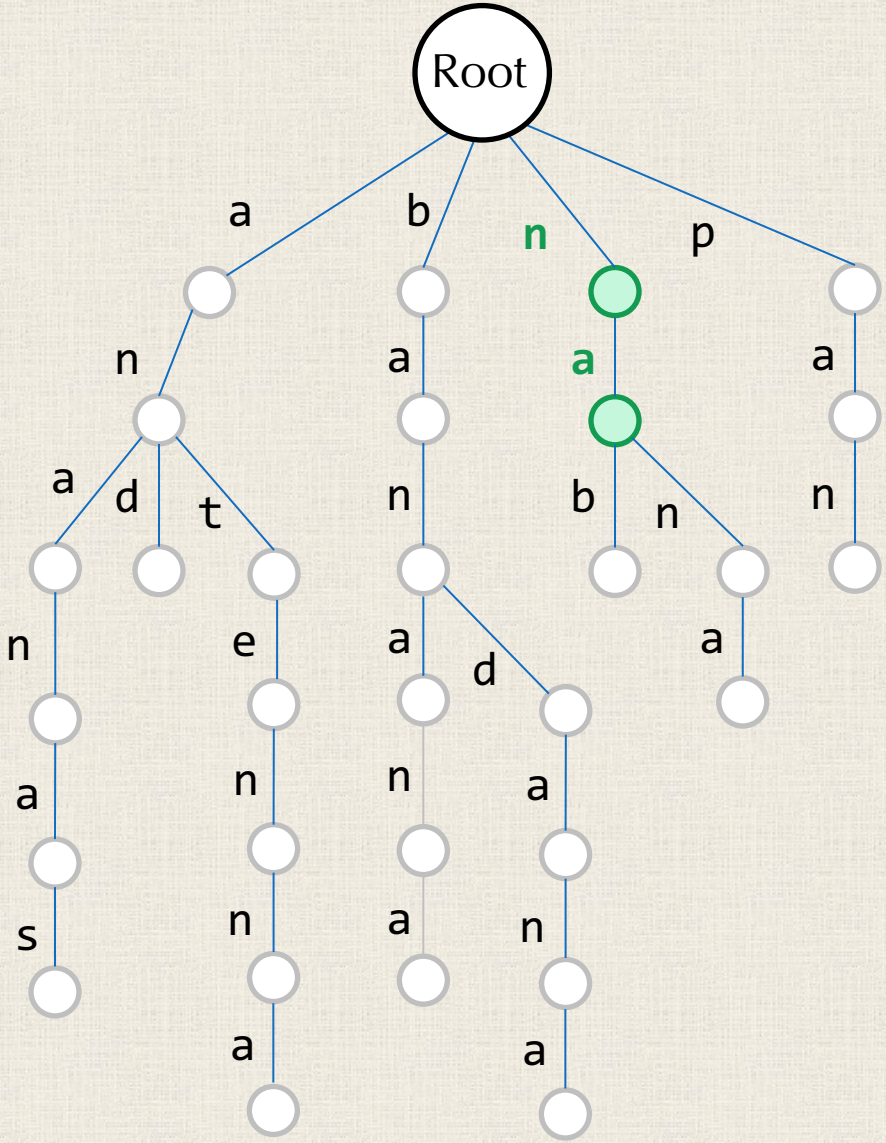
Pattern	Positions
pan	0

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



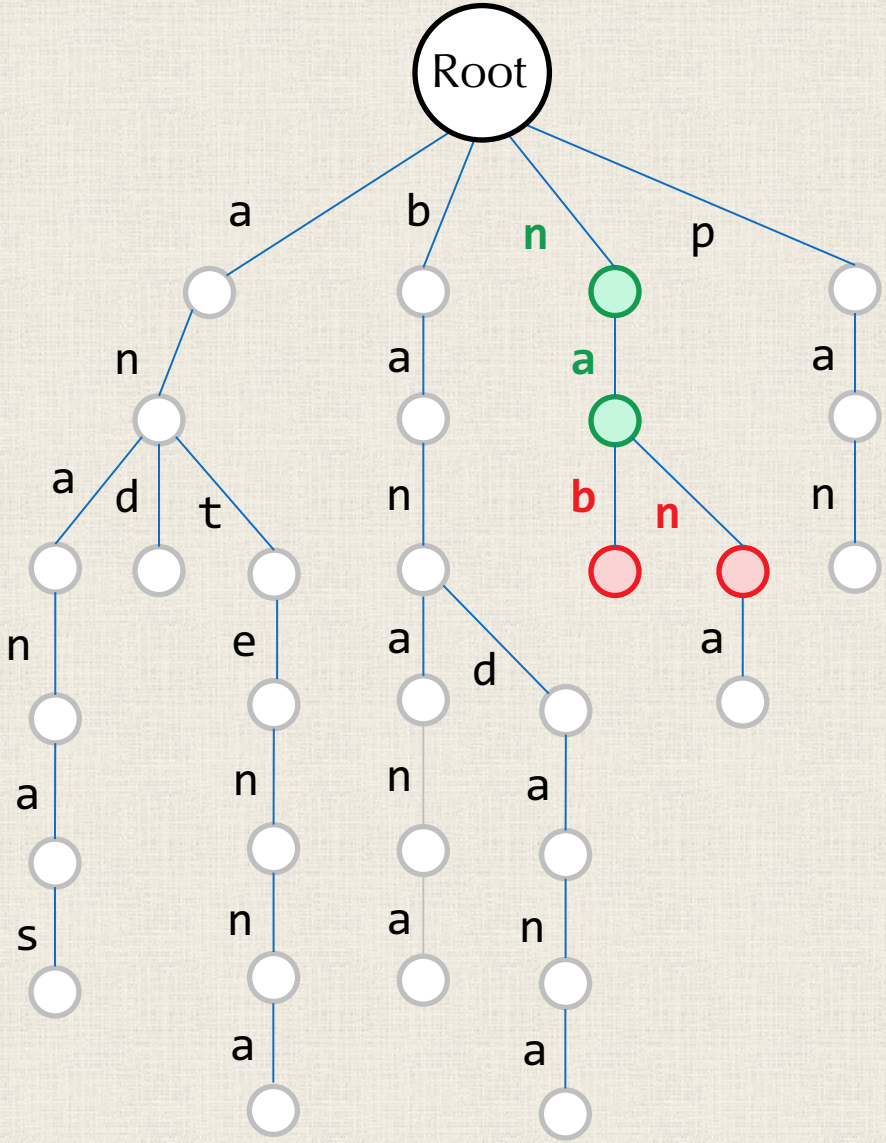
Pattern	Positions
pan	0

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



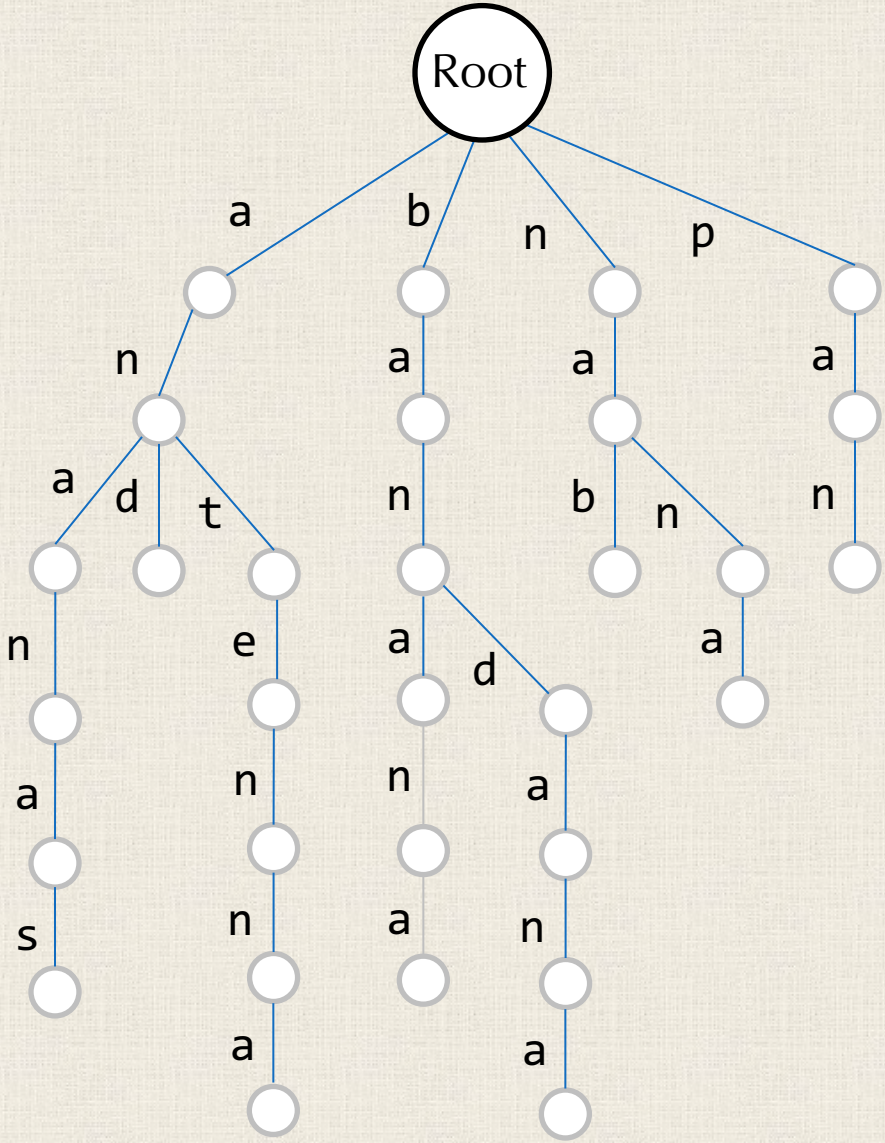
Pattern	Positions
pan	0

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



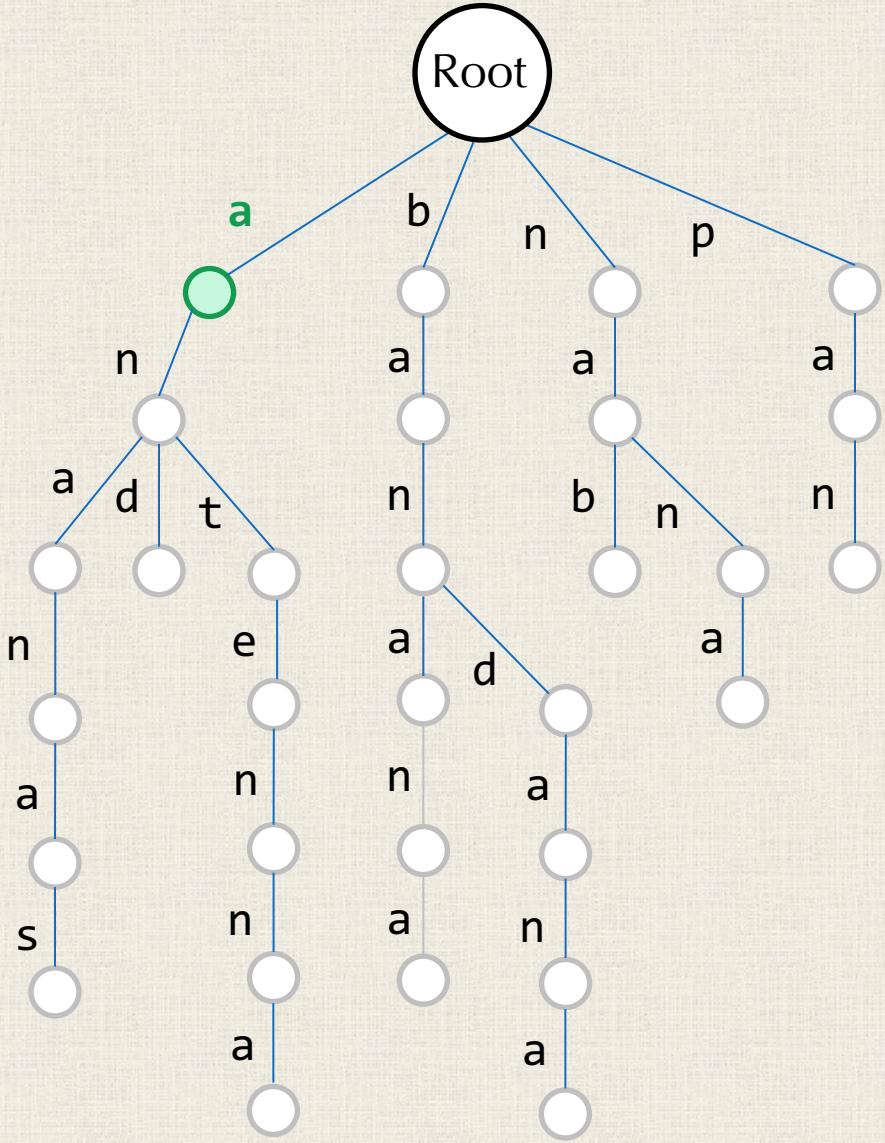
Pattern	Positions
pan	0

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



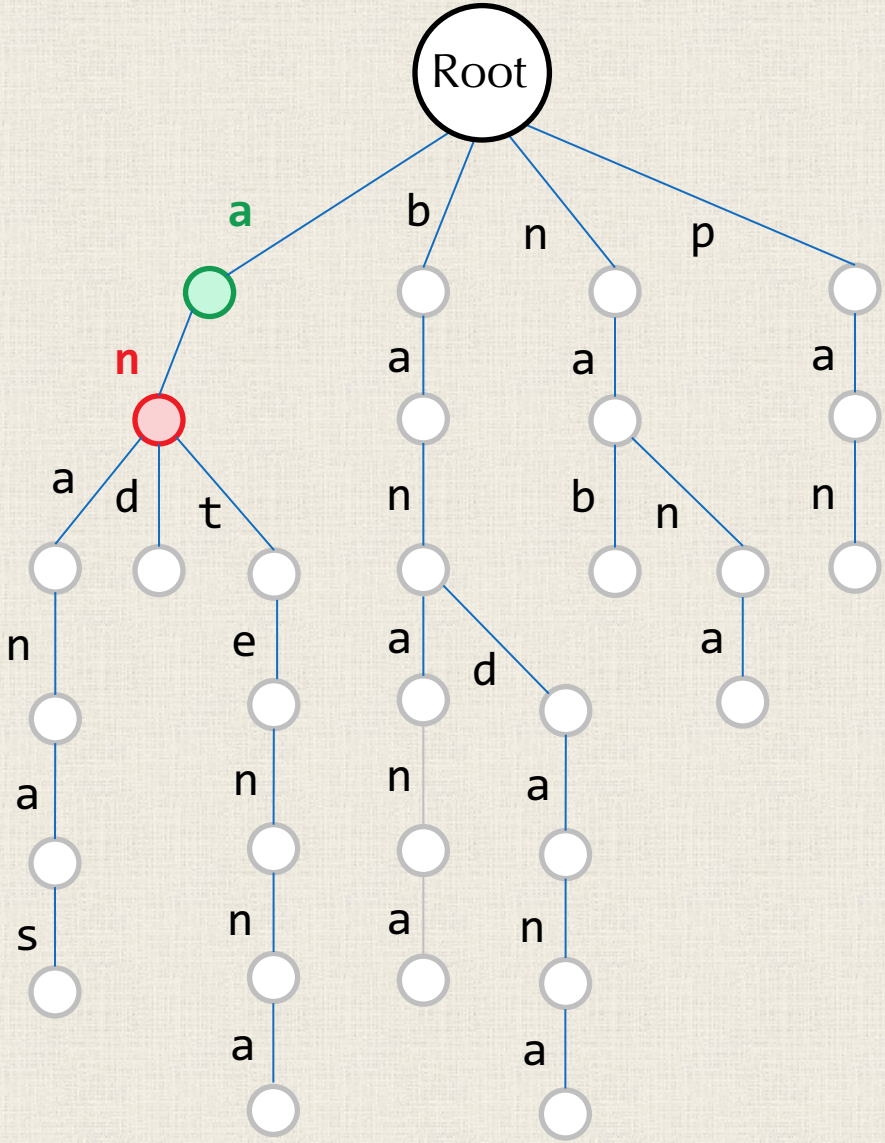
Pattern	Positions
pan	0

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



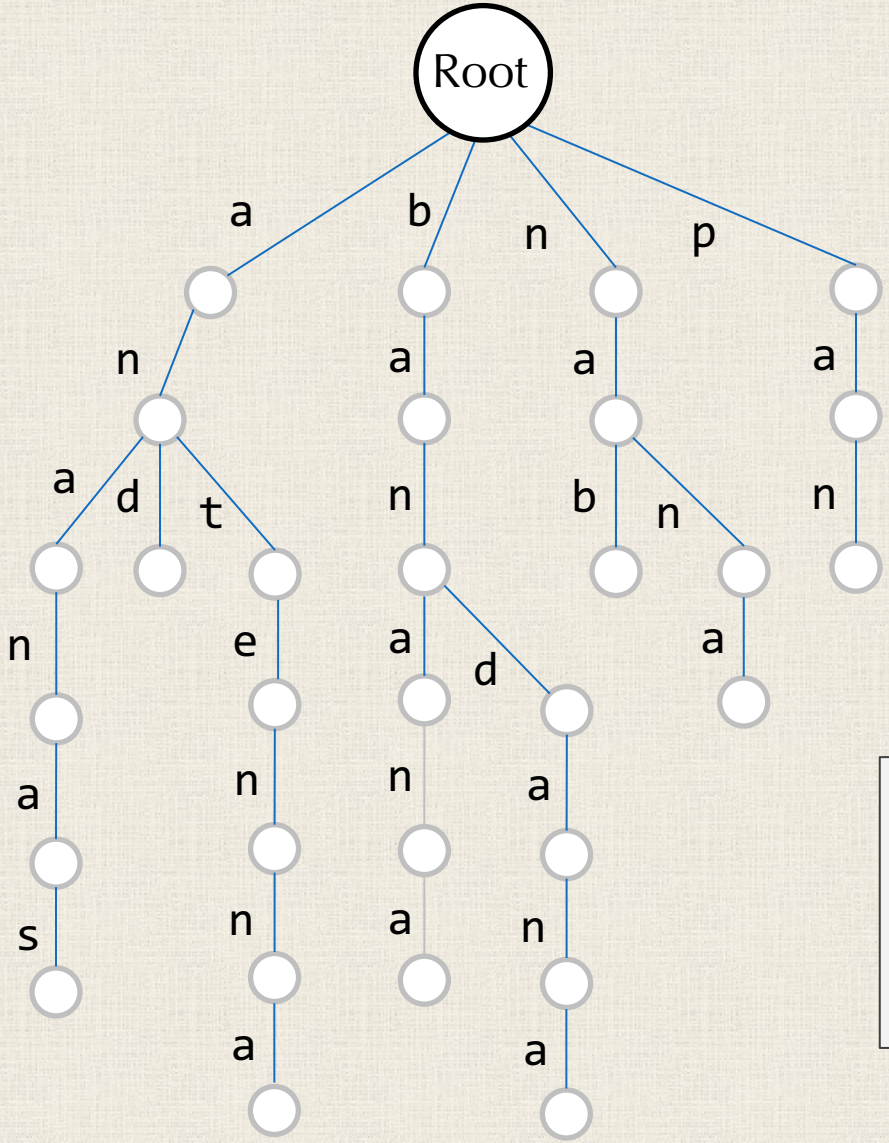
Pattern	Positions
pan	0

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



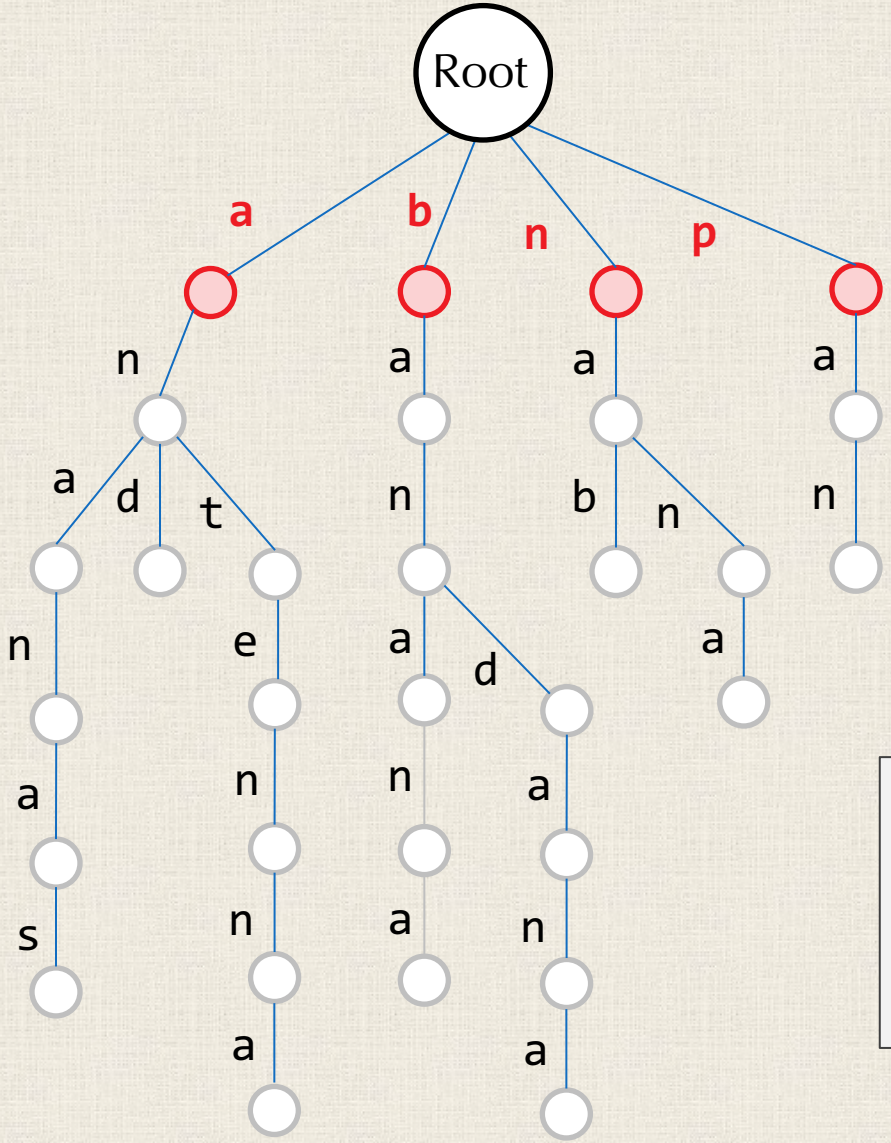
Pattern	Positions
pan	0

panamabananas



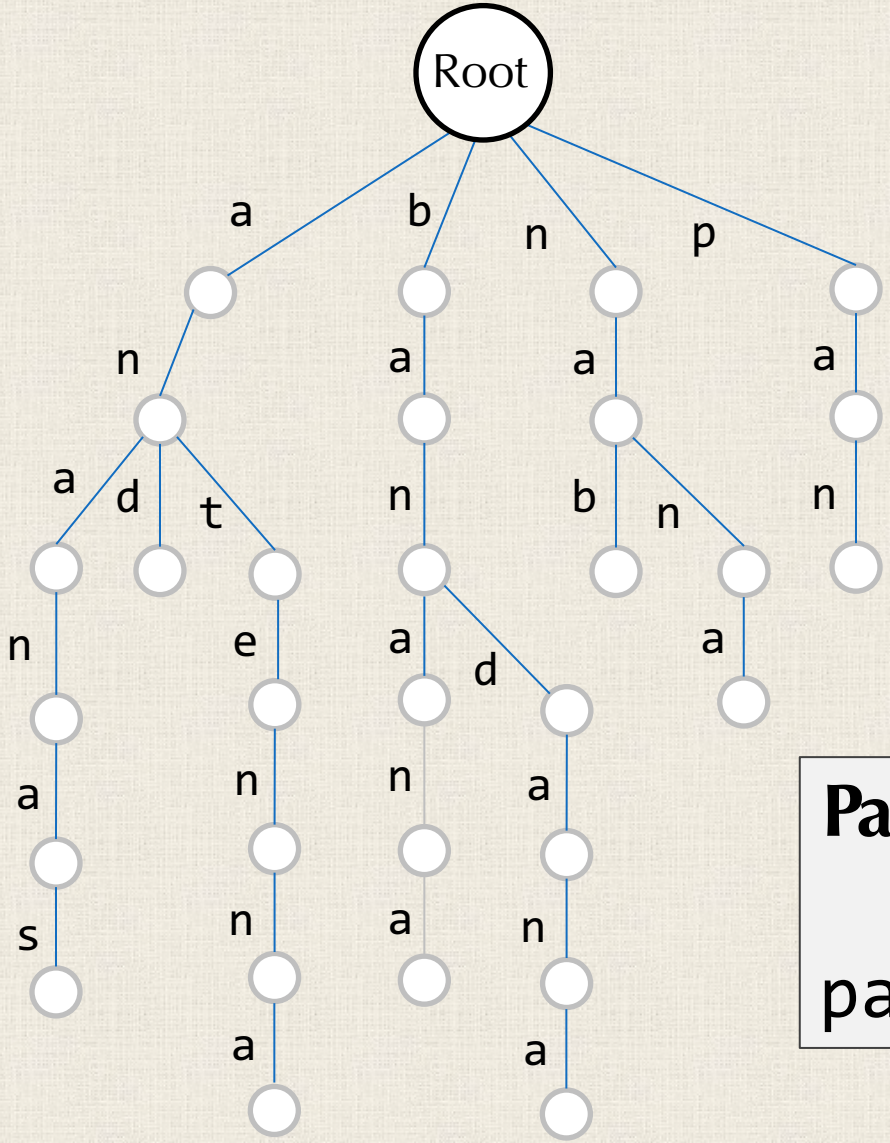
Pattern	Positions
pan	0

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



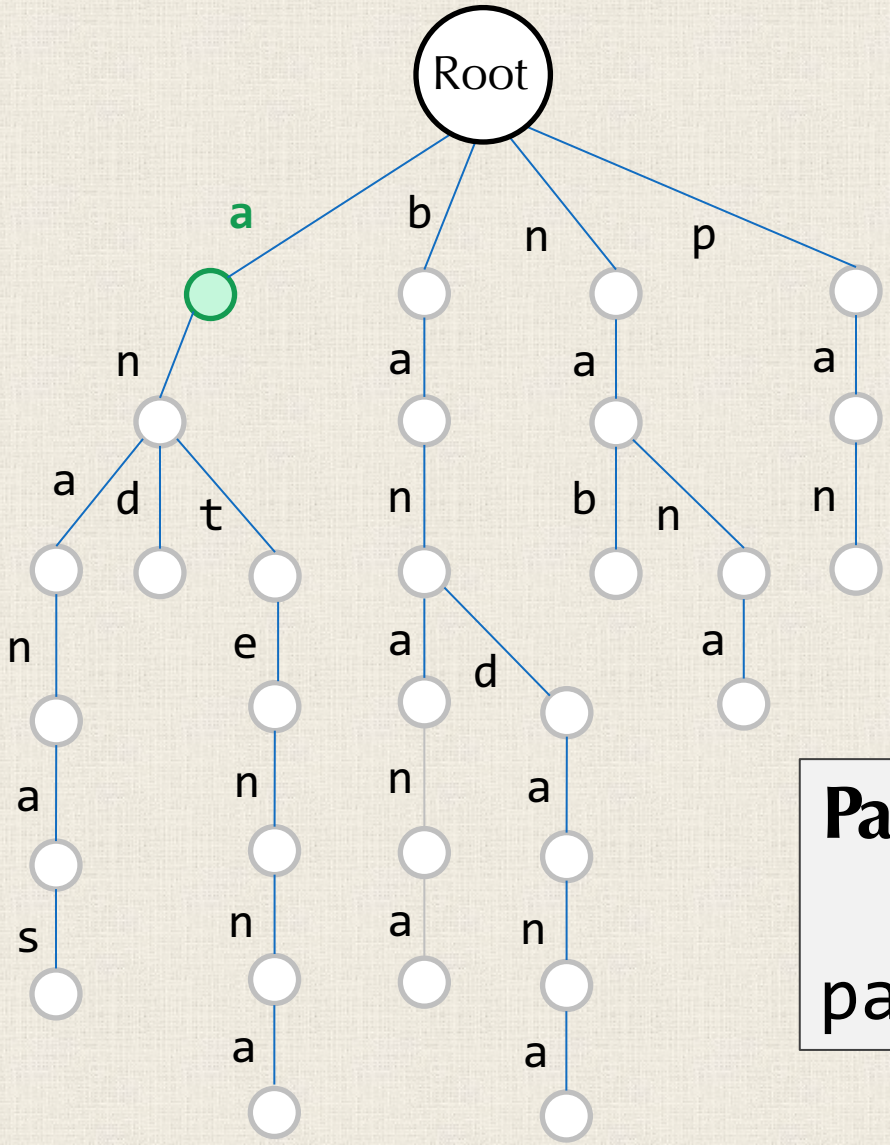
Pattern	Positions
pan	0

panamabananas



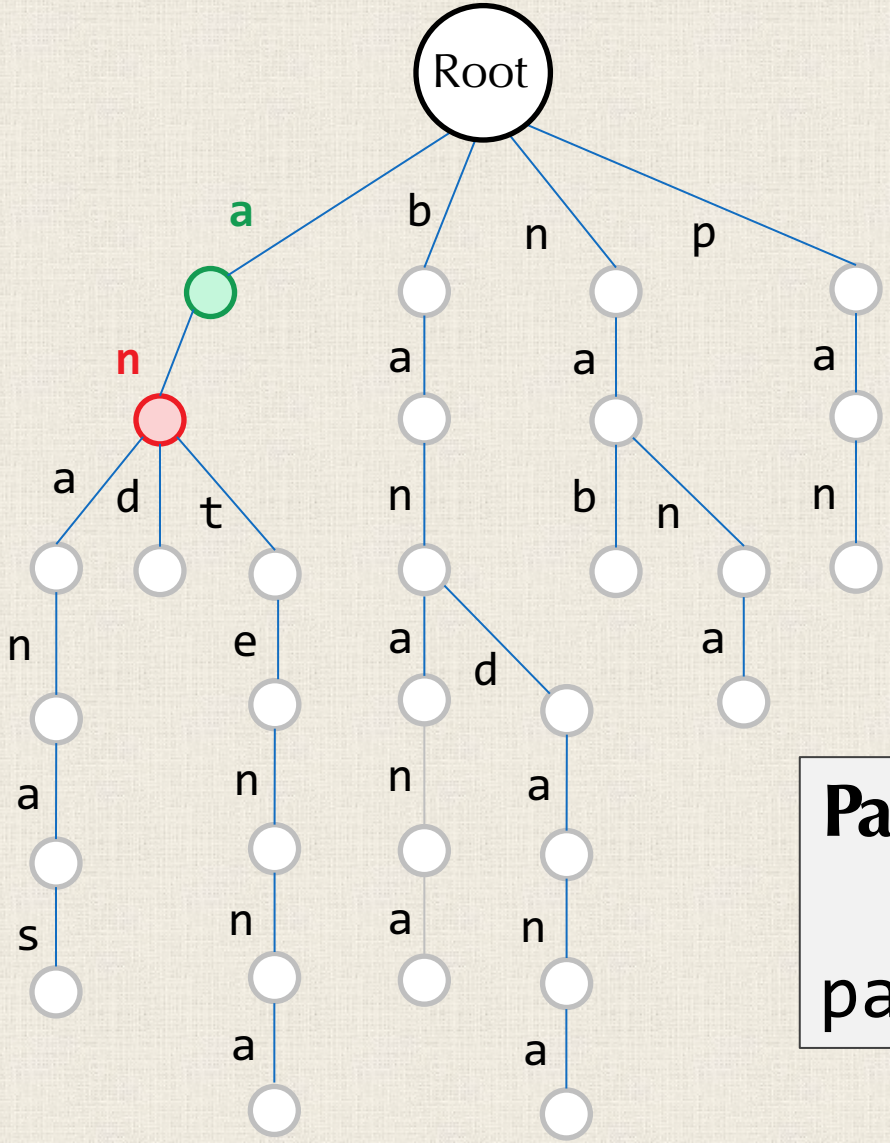
Pattern	Positions
pan	0

panamabananas



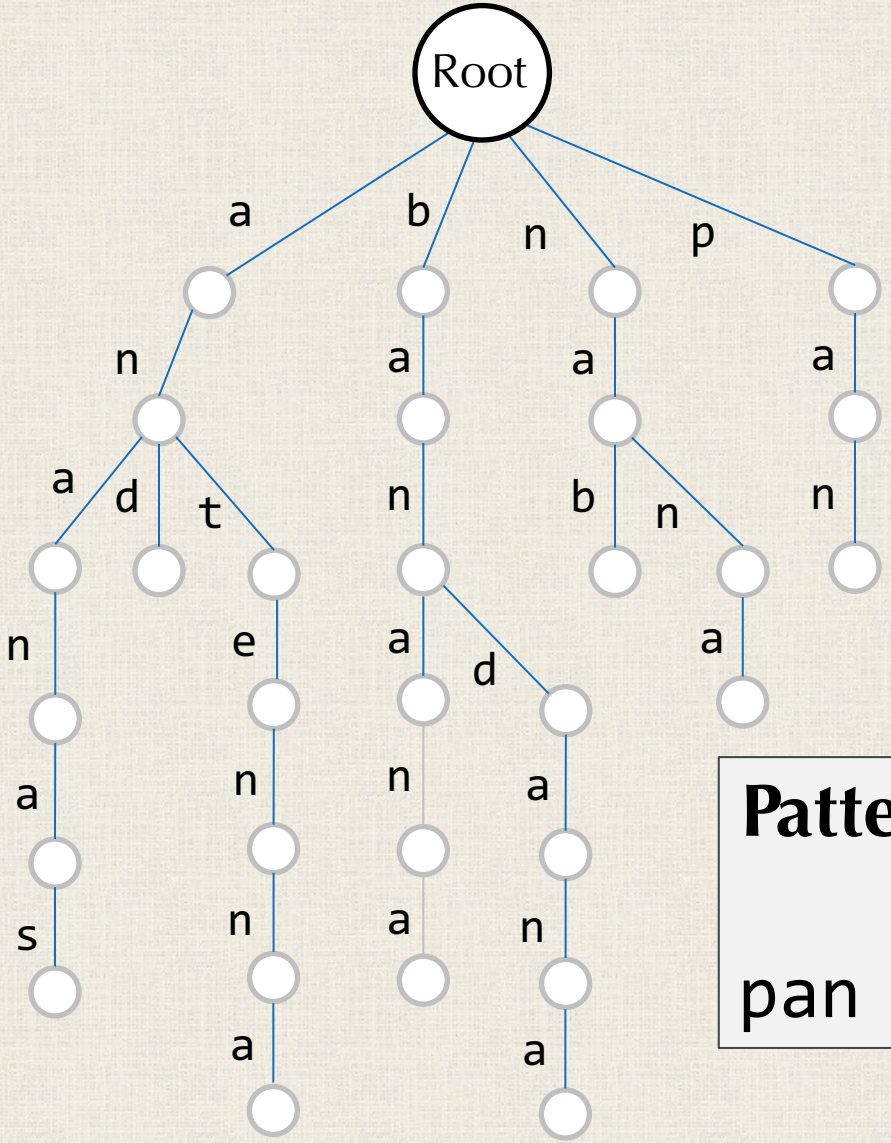
Pattern	Positions
pan	0

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



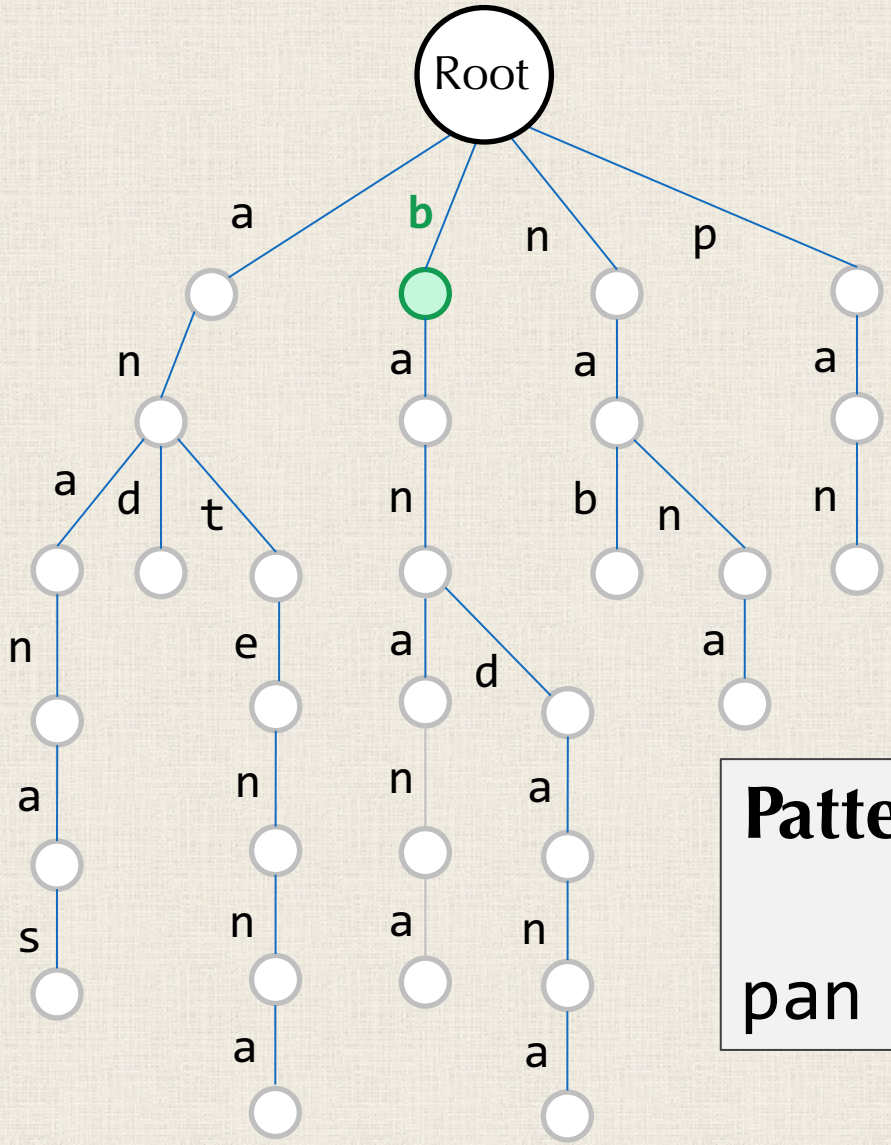
Pattern	Positions
pan	0

panamabananas



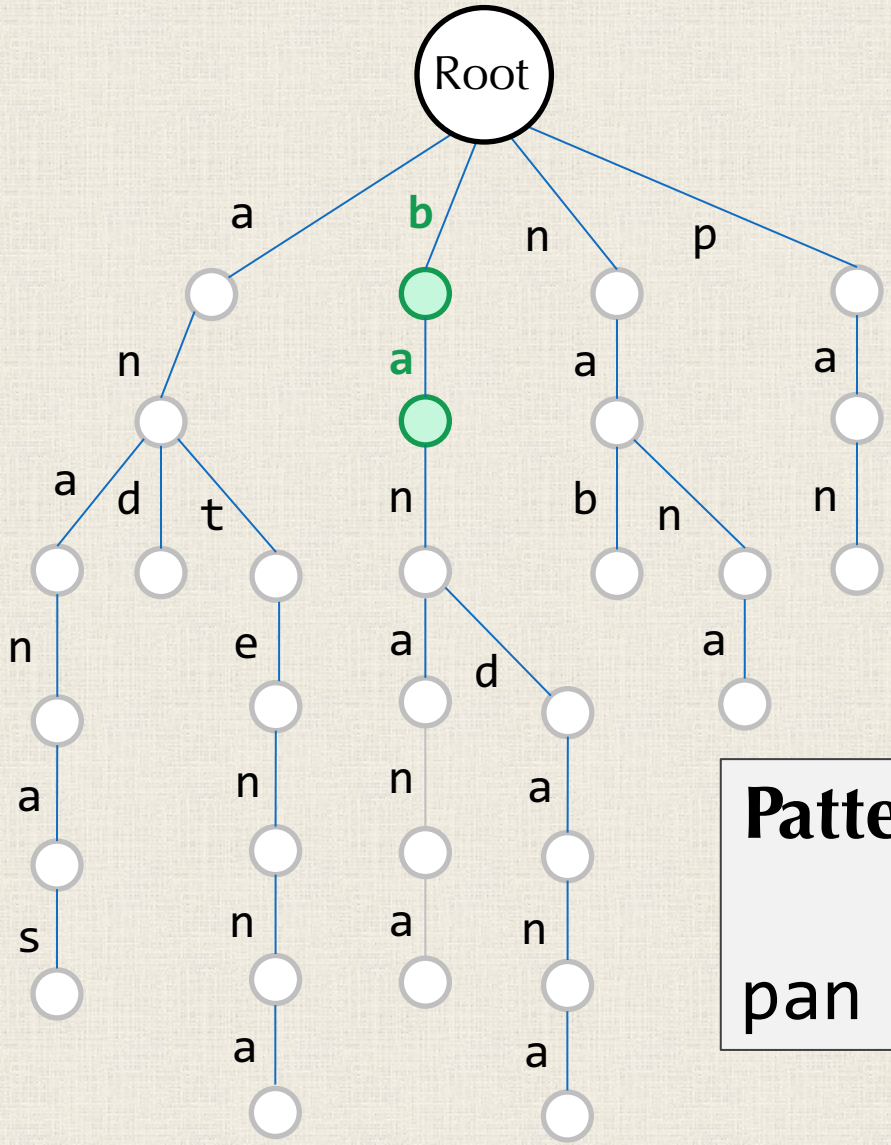
Pattern	Positions
pan	0

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



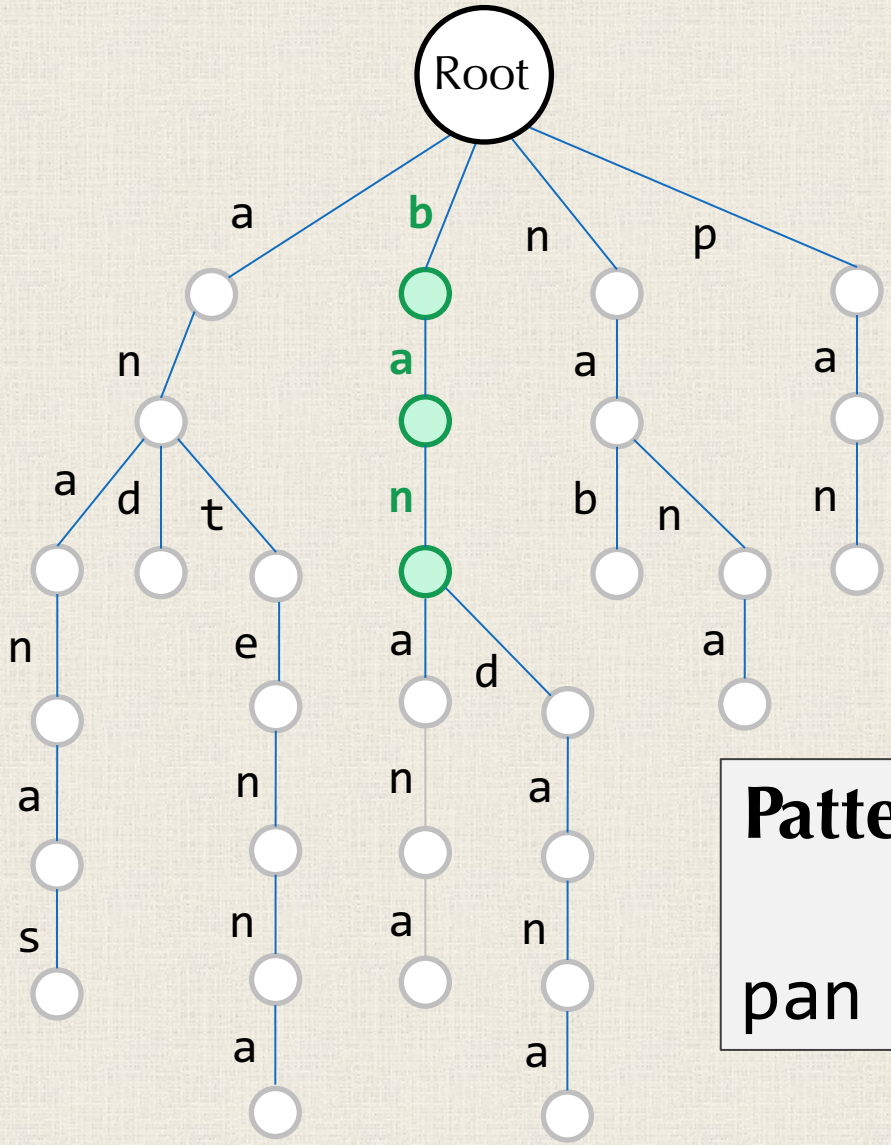
Pattern	Positions
pan	0

panama**ba**nanas



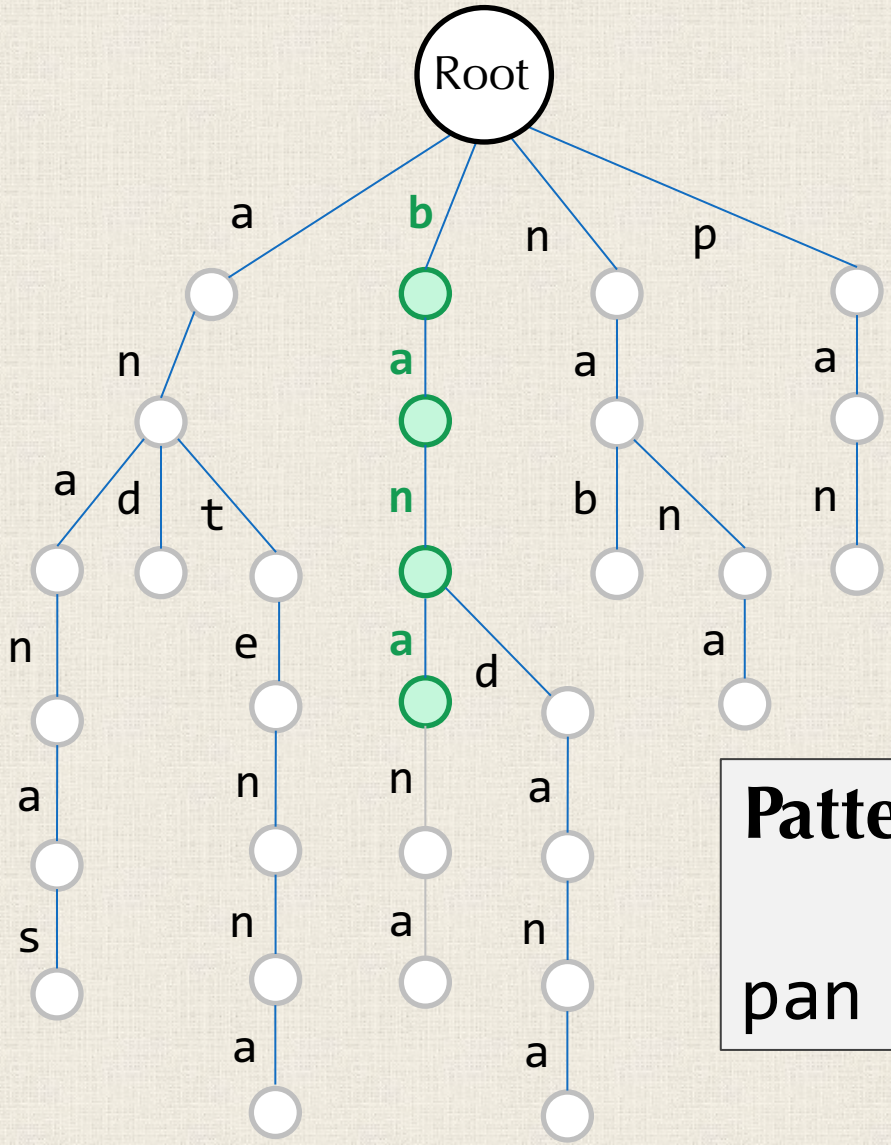
Pattern	Positions
pan	0

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



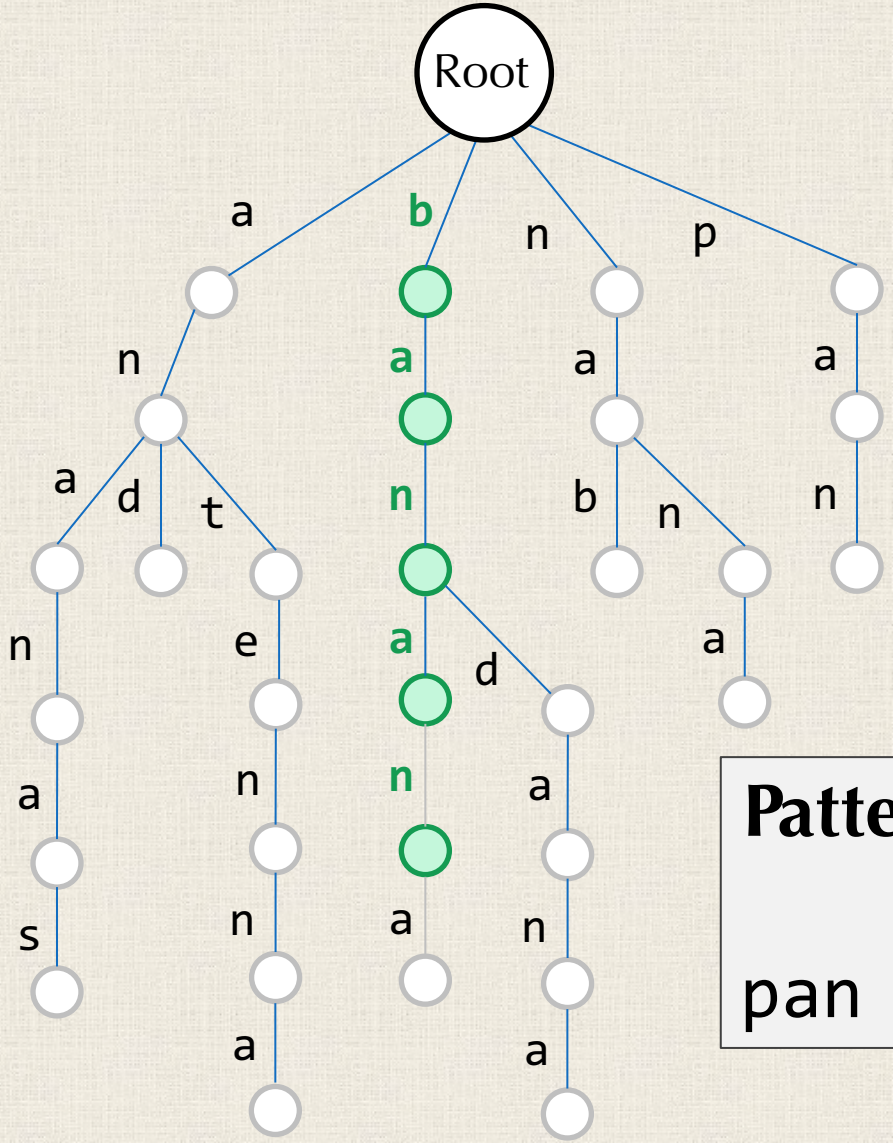
Pattern	Positions
pan	0

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



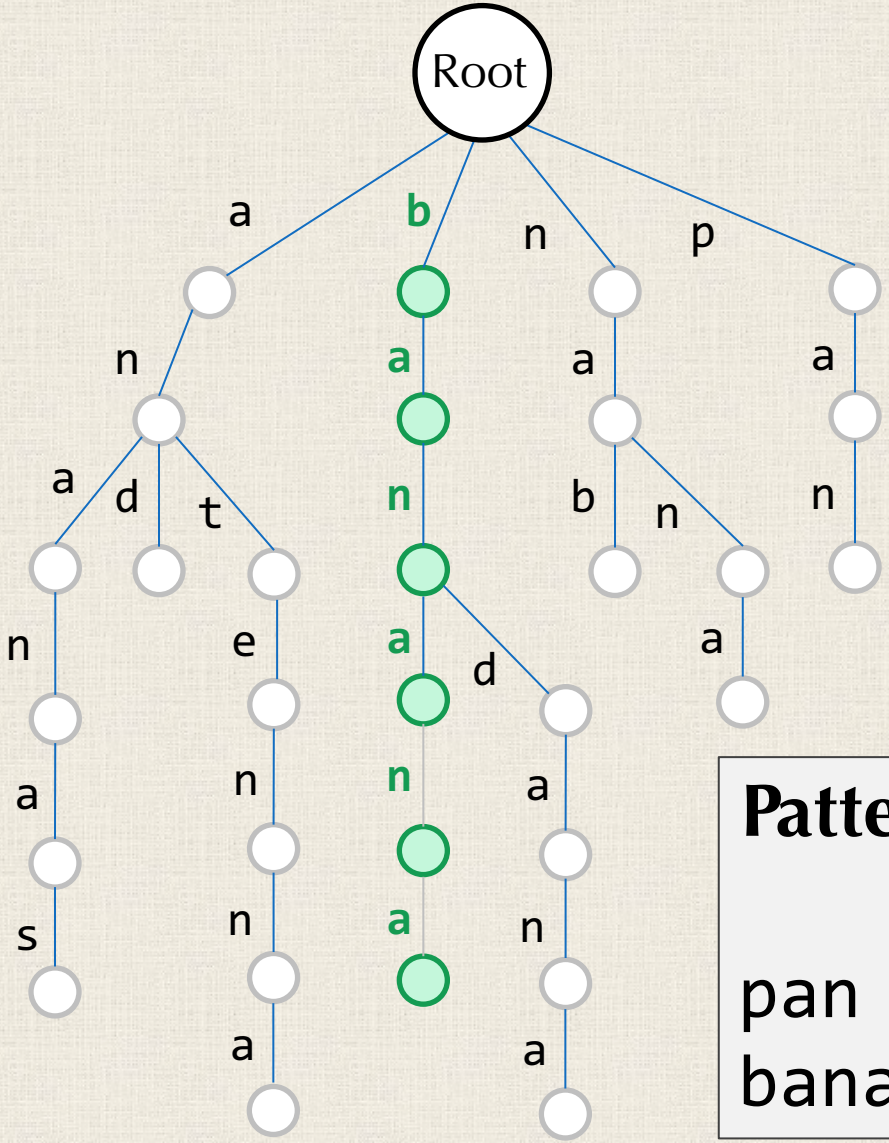
Pattern	Positions
pan	0

panama**banan**as



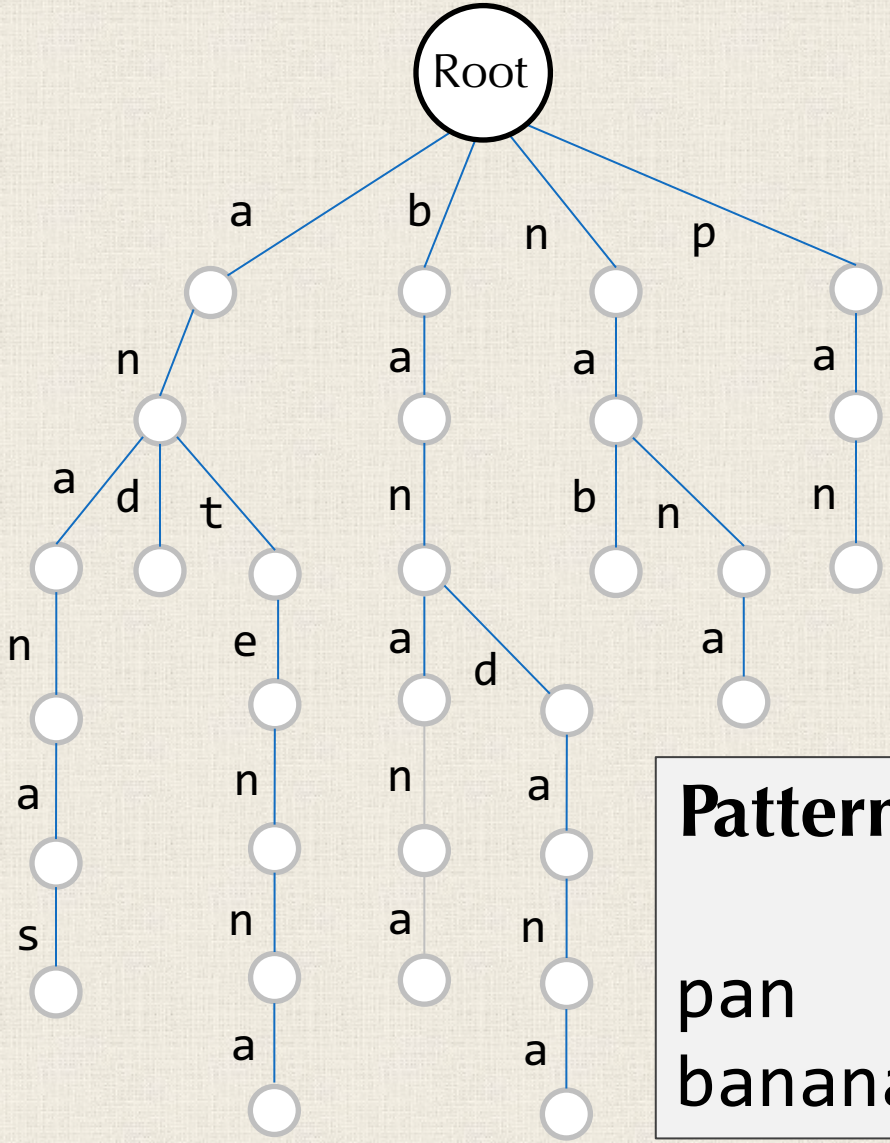
Pattern	Positions
pan	0

panama**banana**s



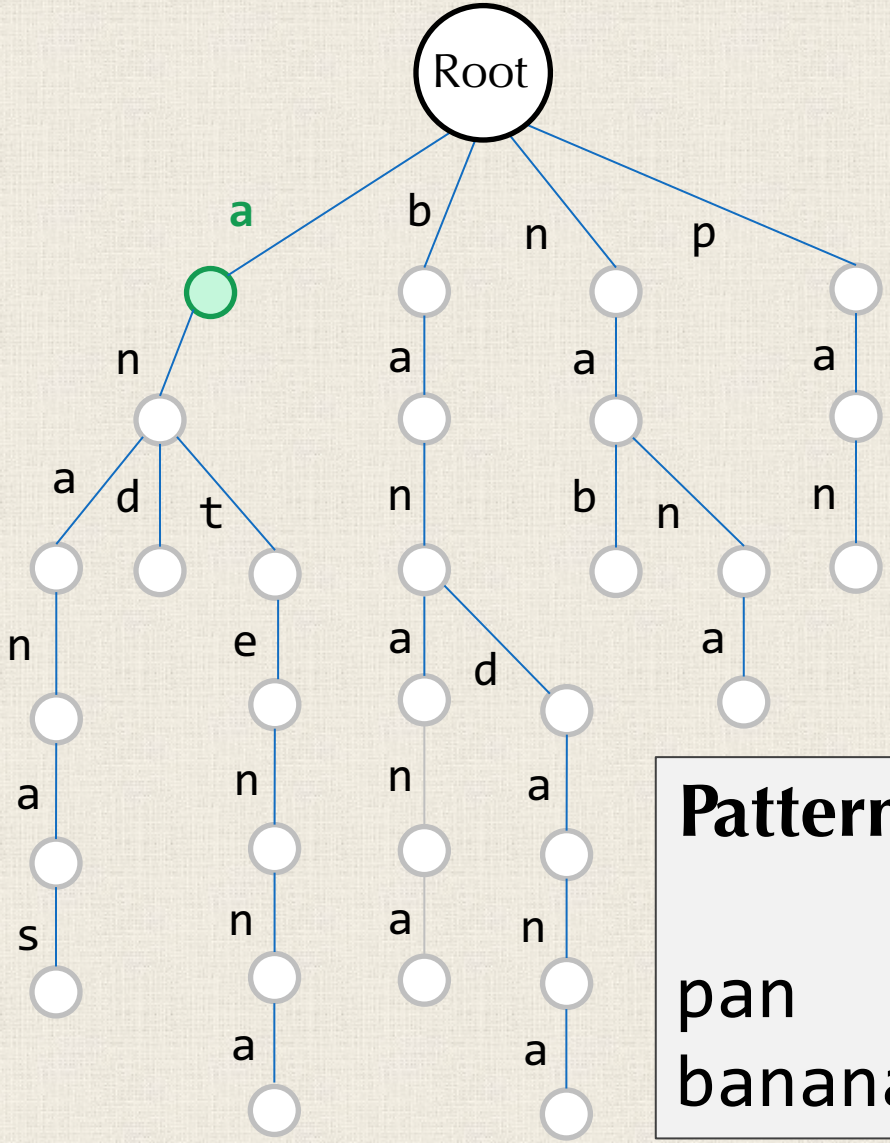
Pattern	Positions
pan	0
banana	6

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



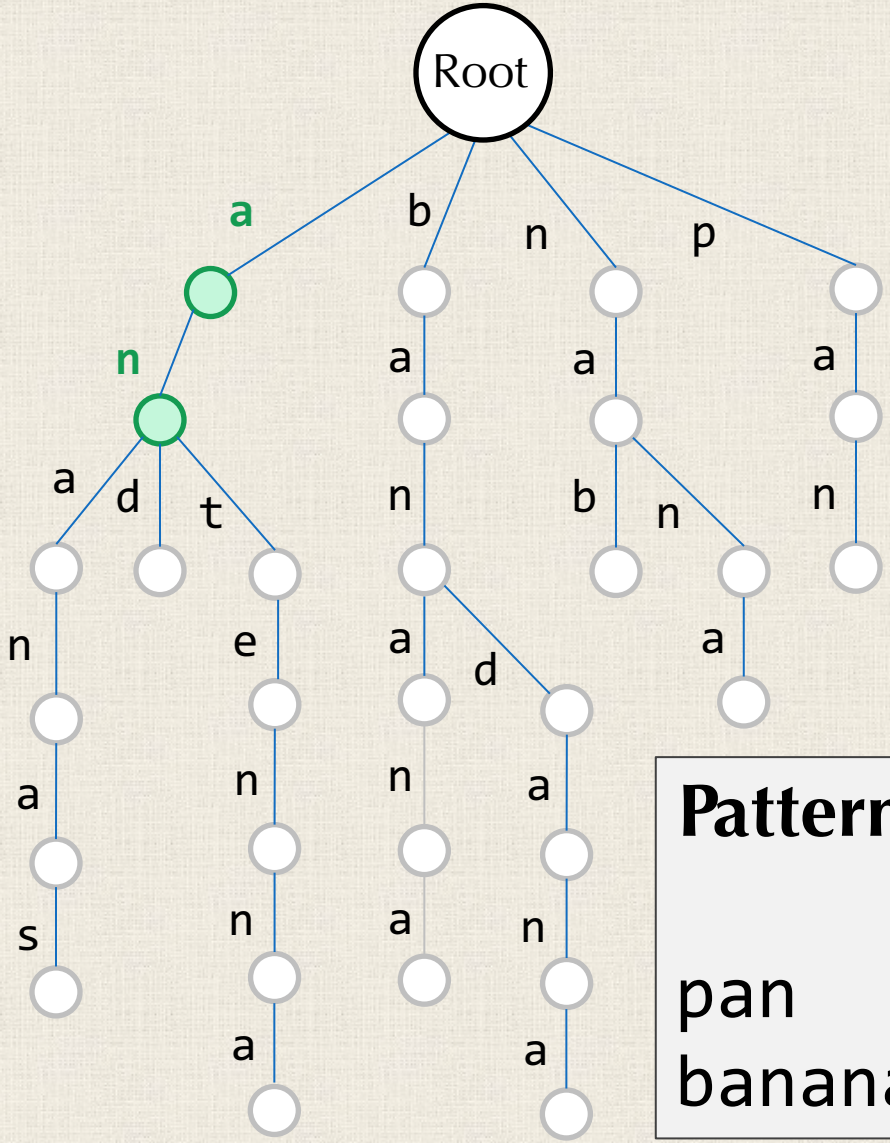
Pattern	Positions
pan	0
banana	6

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



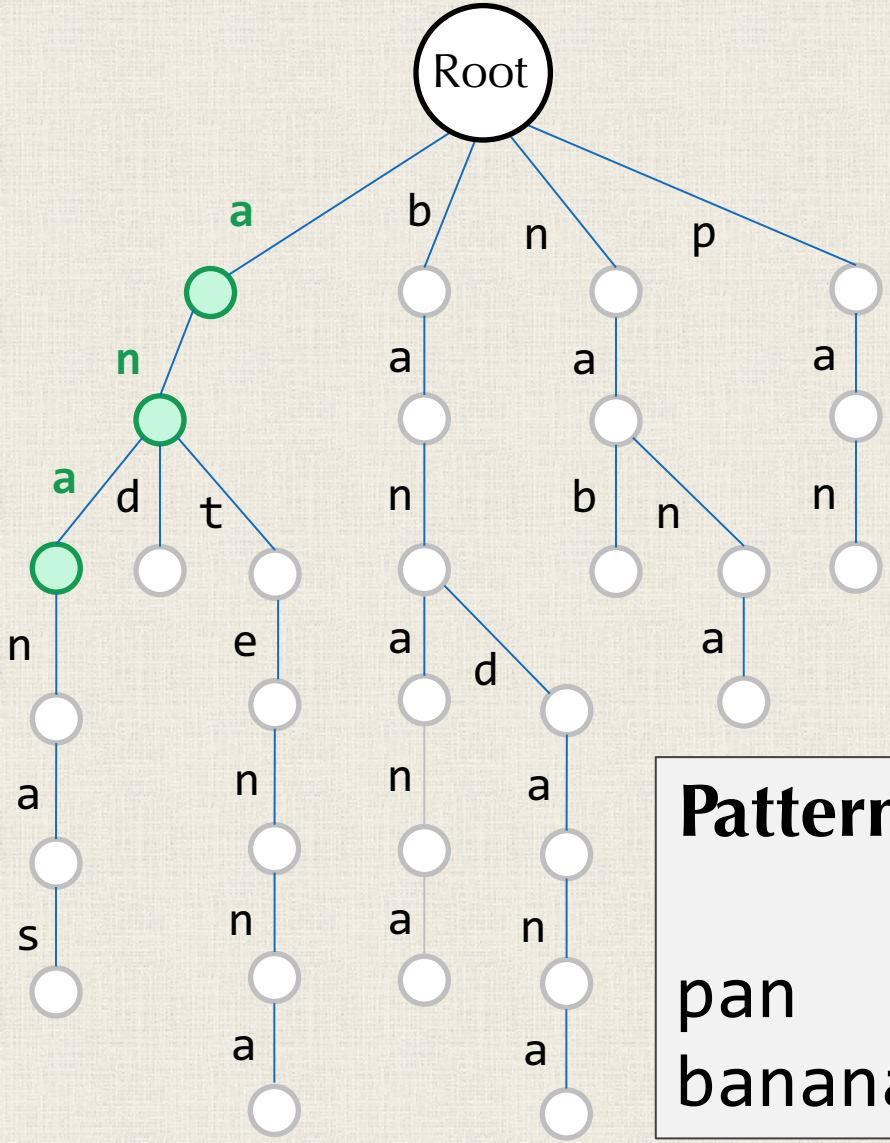
Pattern	Positions
pan	0
banana	6

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



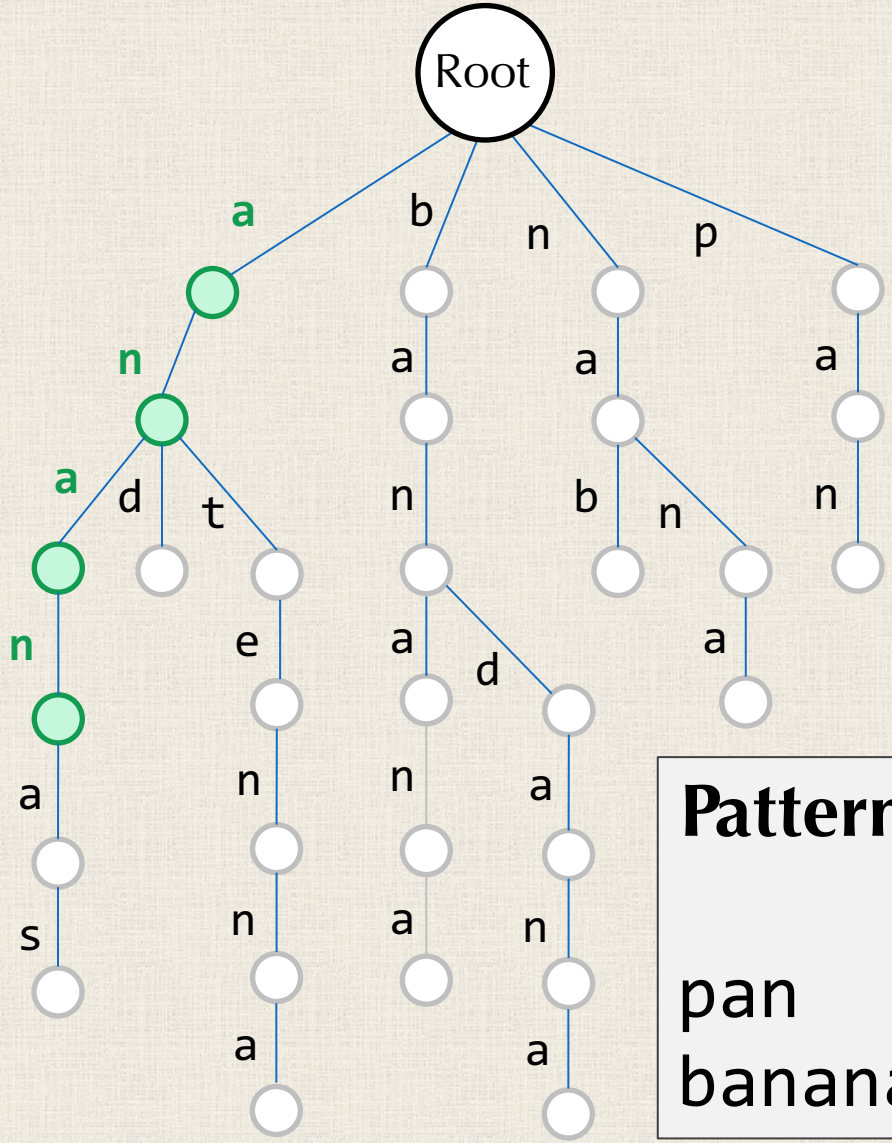
Pattern	Positions
pan	0
banana	6

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



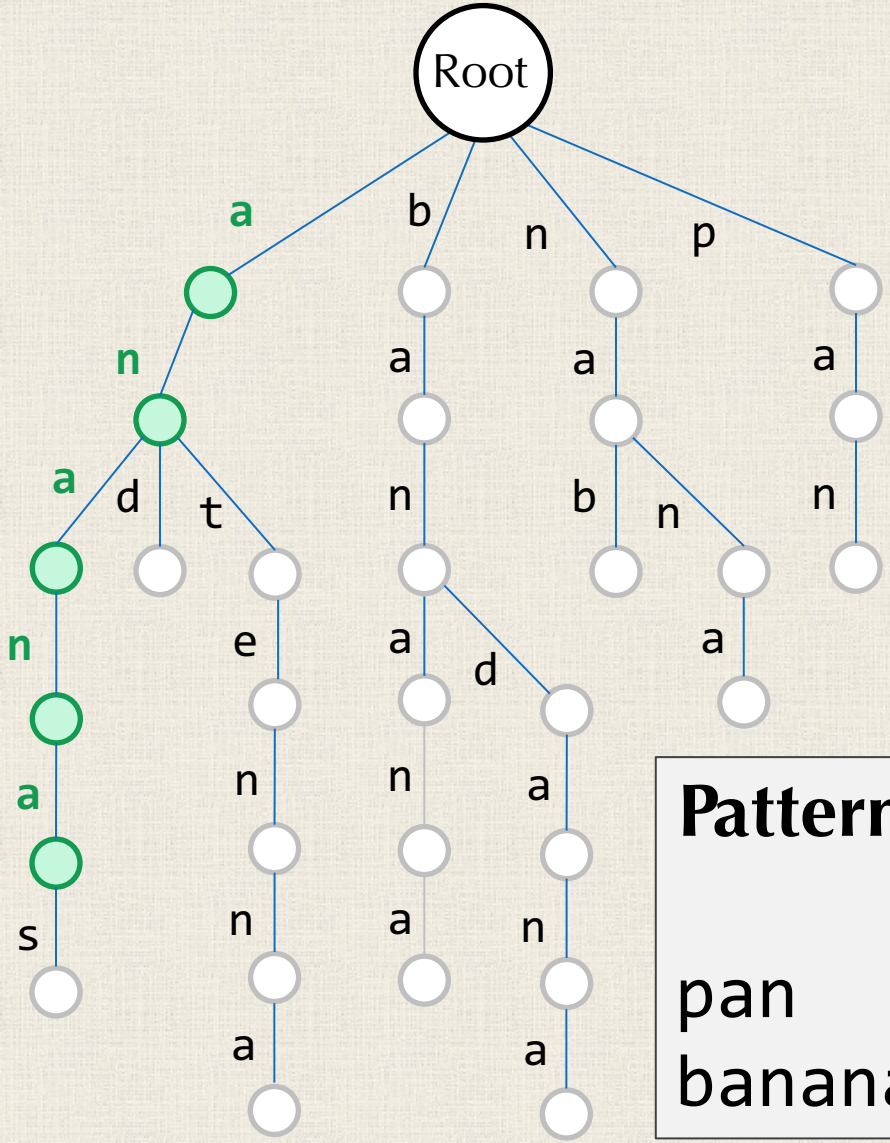
Pattern	Positions
pan	0
banana	6

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



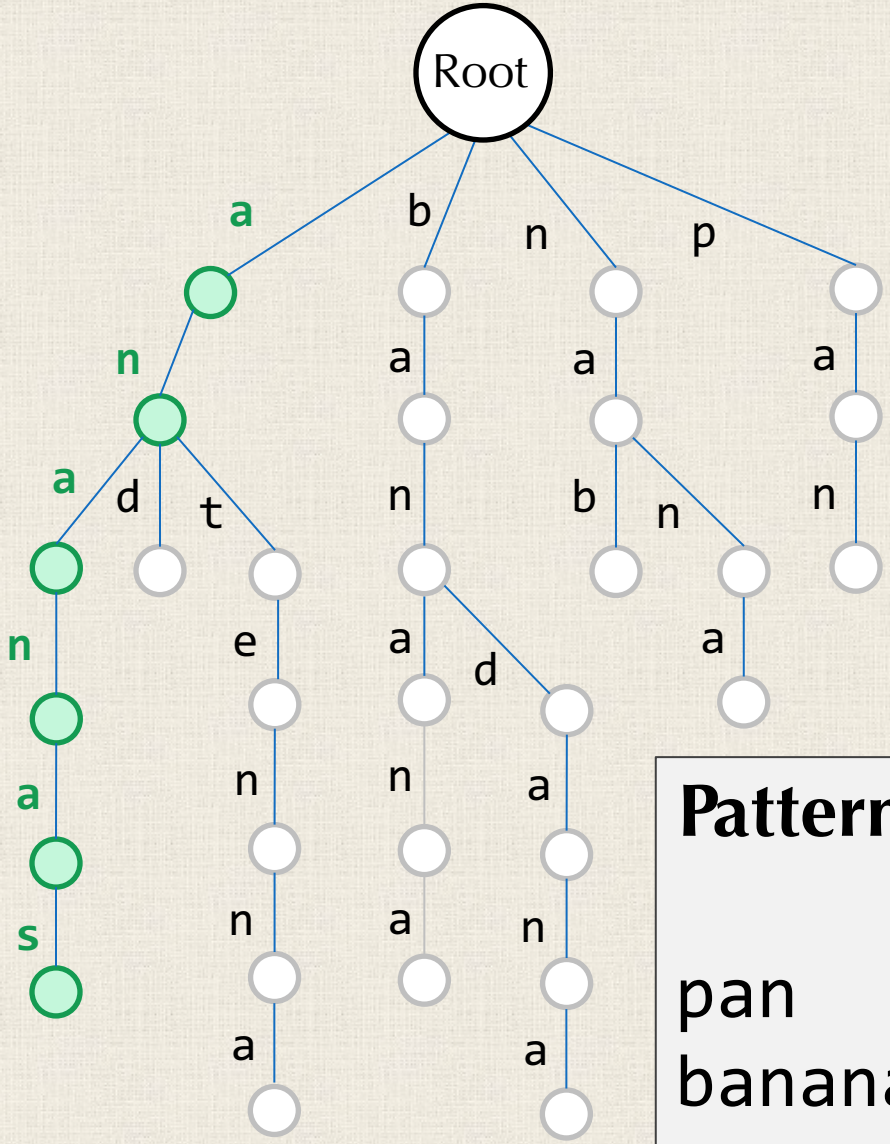
Pattern	Positions
pan	0
banana	6

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



Pattern	Positions
pan	0
banana	6

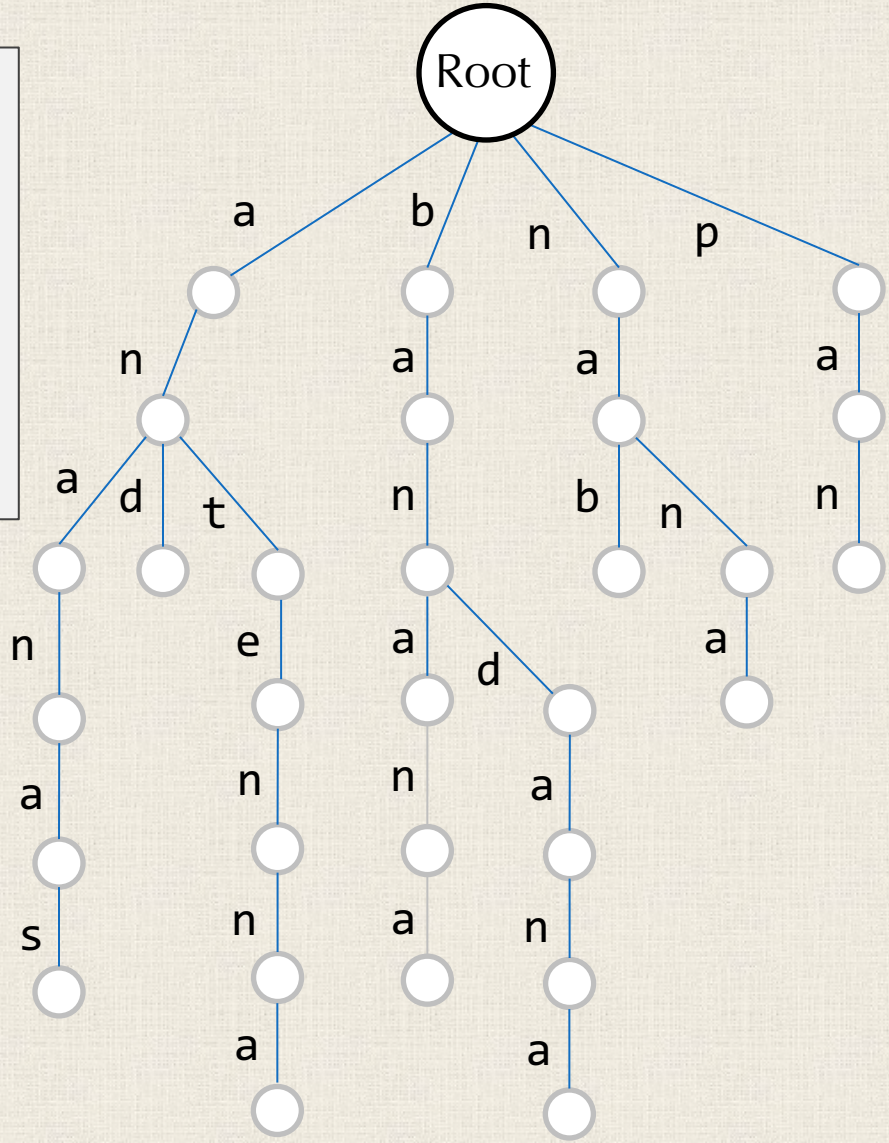
p a n a m a b **a n a n a s**



Pattern	Positions
pan	0
banana	6
ananas	7

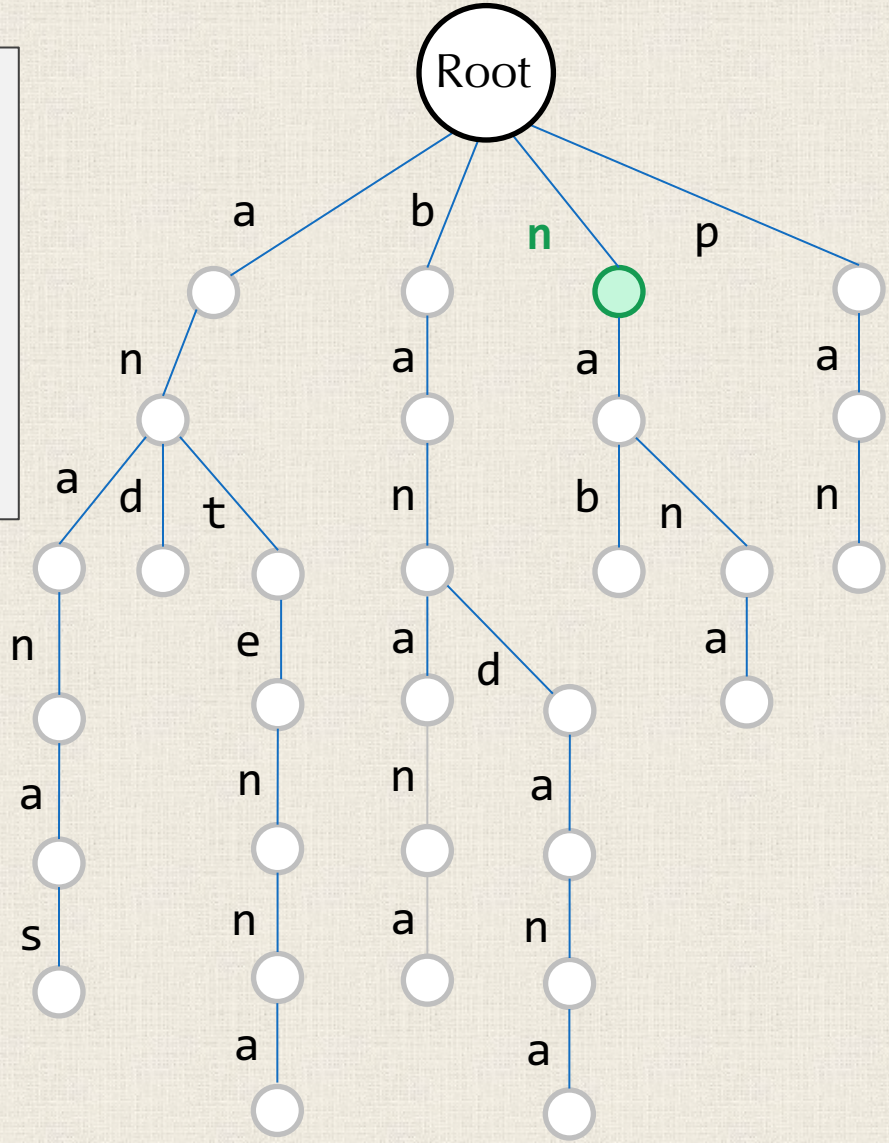
p a n a m a b a n a n a s

Pattern	Positions
pan	0
banana	6
ananas	7



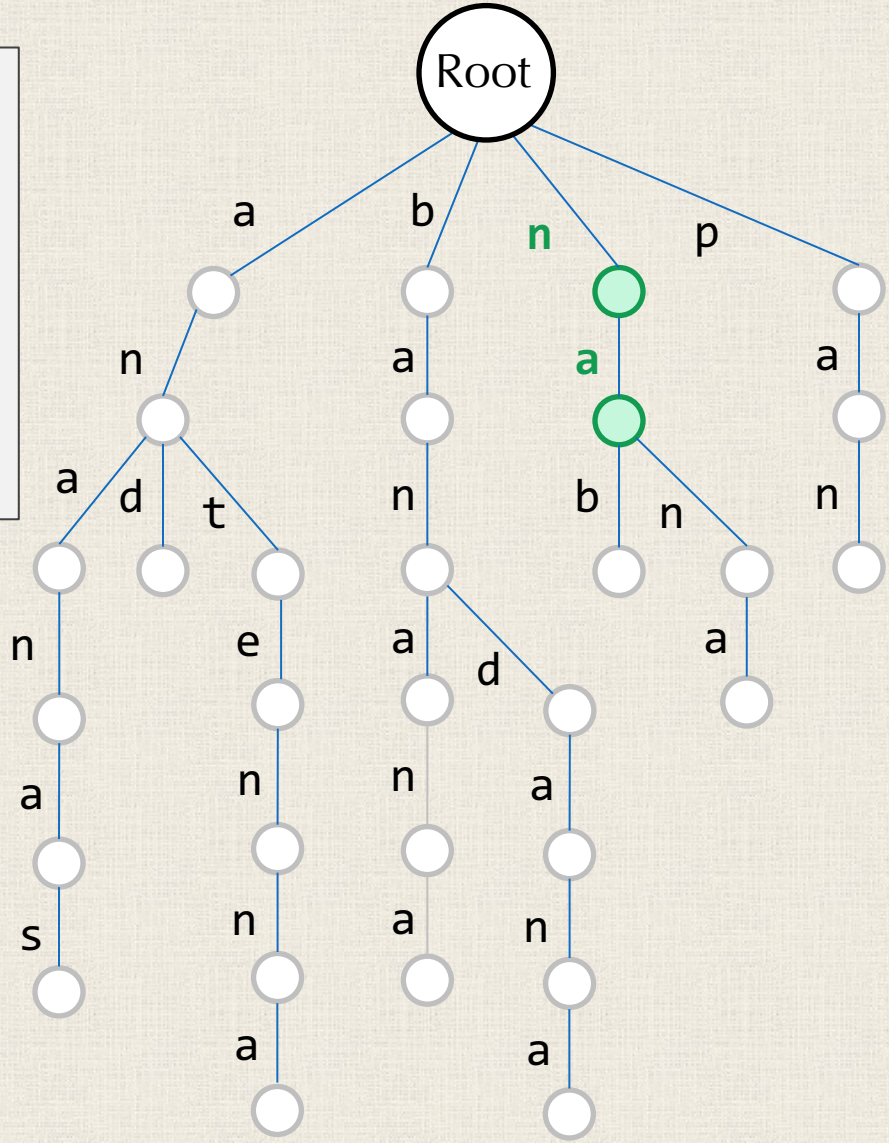
p a n a m a b a n a n a s

Pattern	Positions
pan	0
banana	6
ananas	7



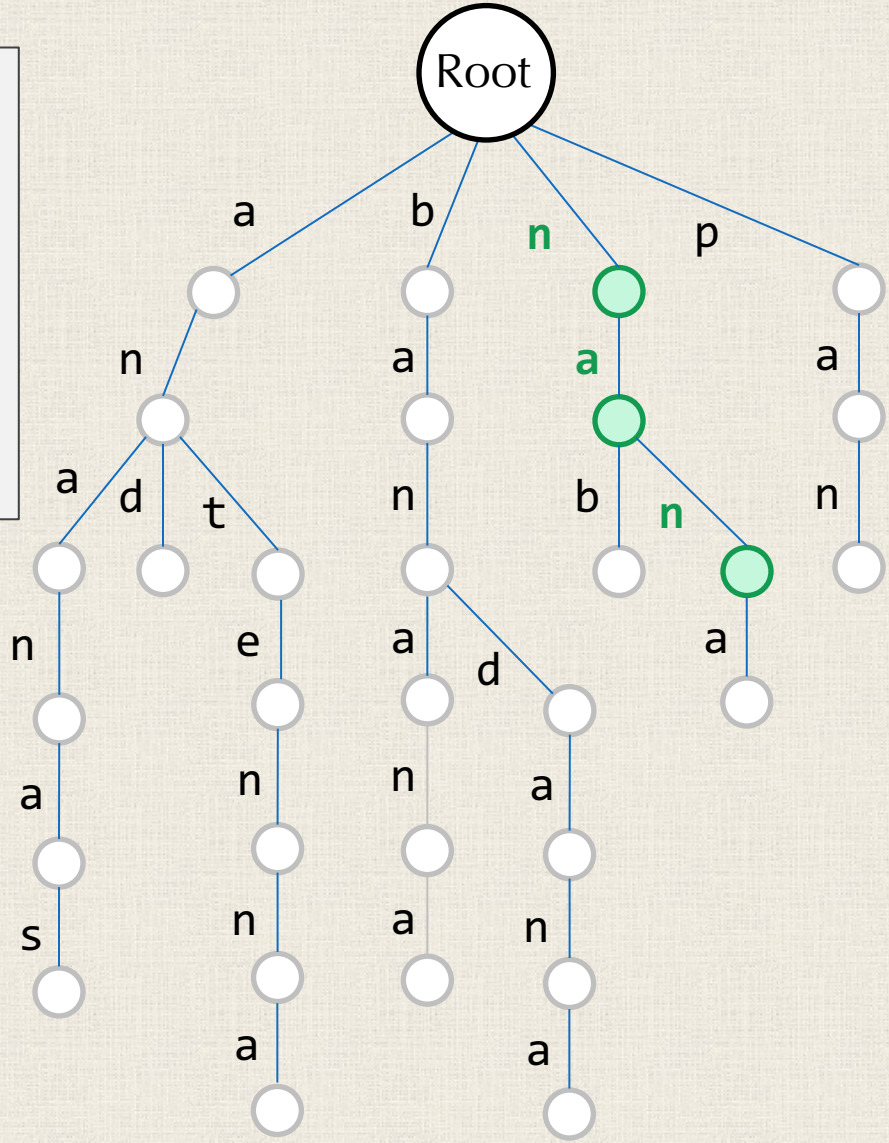
p a n a m a b a **n a** n a s

Pattern	Positions
pan	0
banana	6
ananas	7



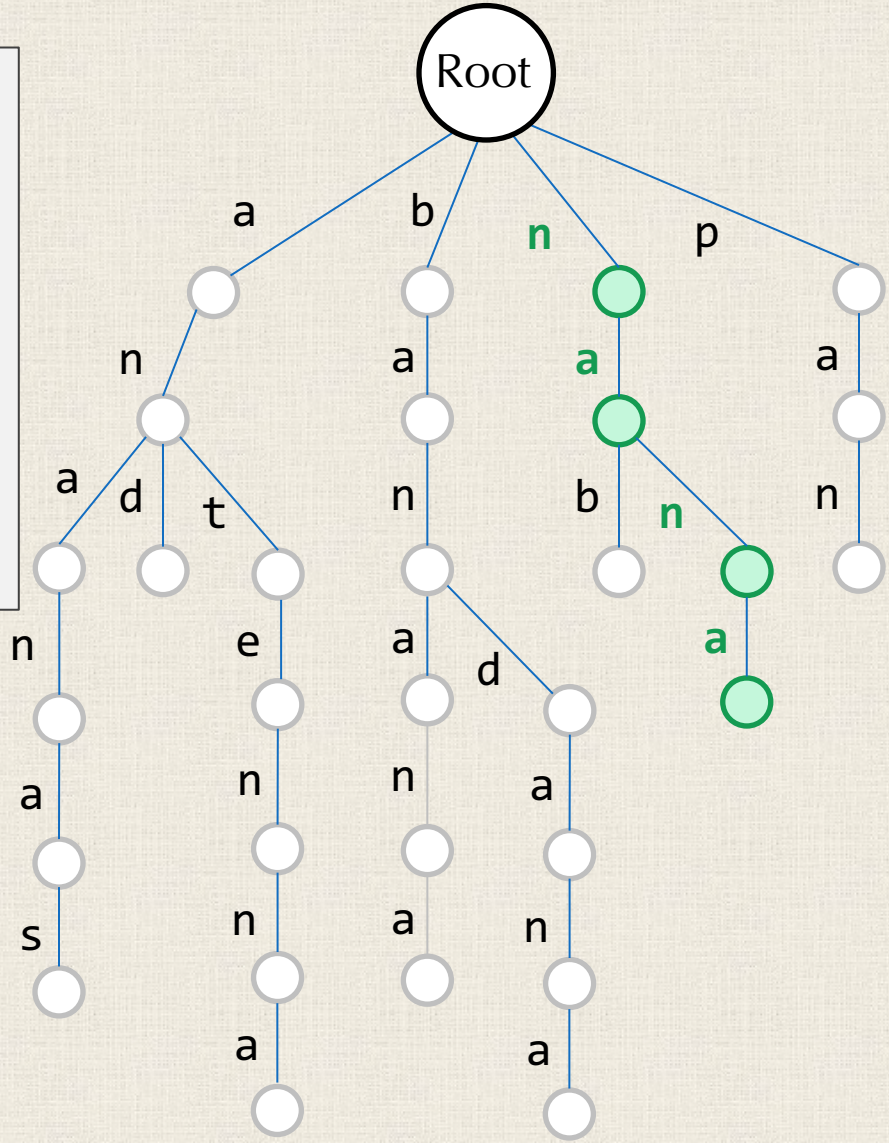
p a n a m a b a **n a n** a s

Pattern	Positions
pan	0
banana	6
ananas	7



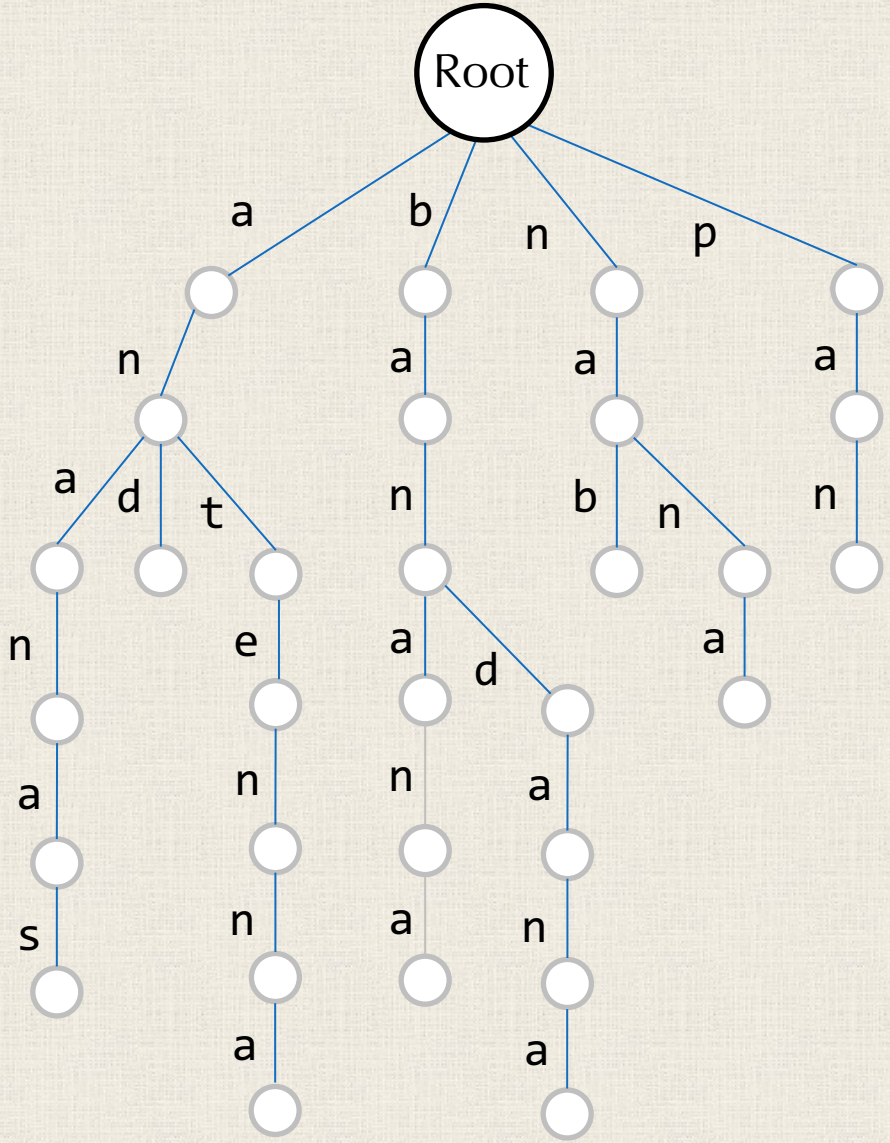
p a n a m a b a **n a n a** s

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8



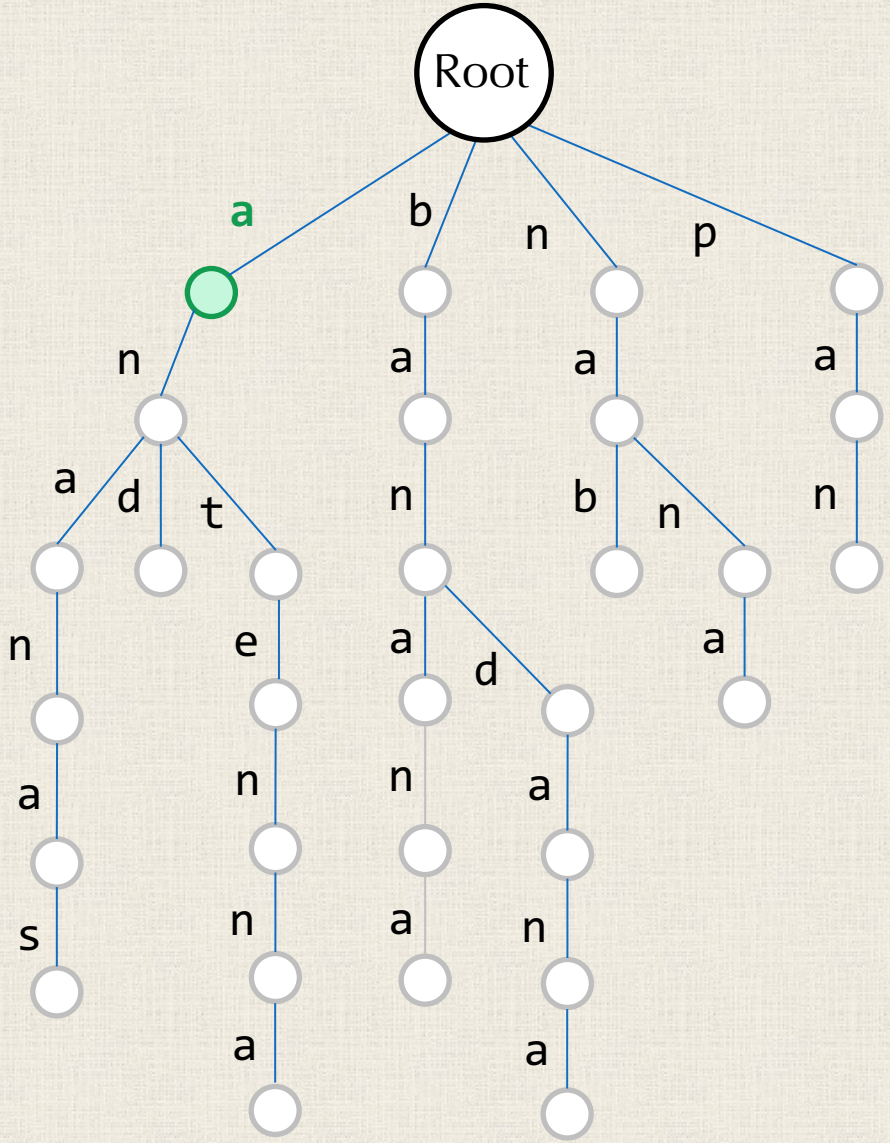
p a n a m a b a n a n a s

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8



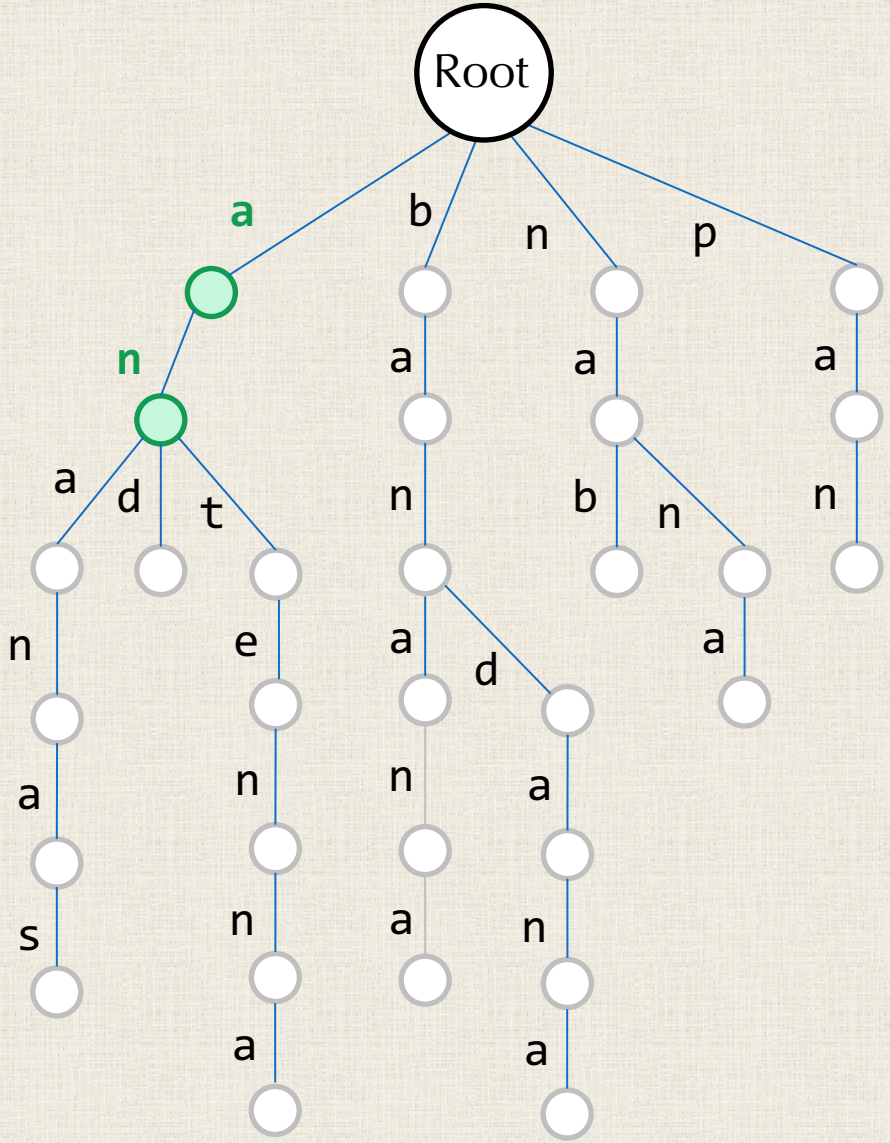
p a n a m a b a n a n a s

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8



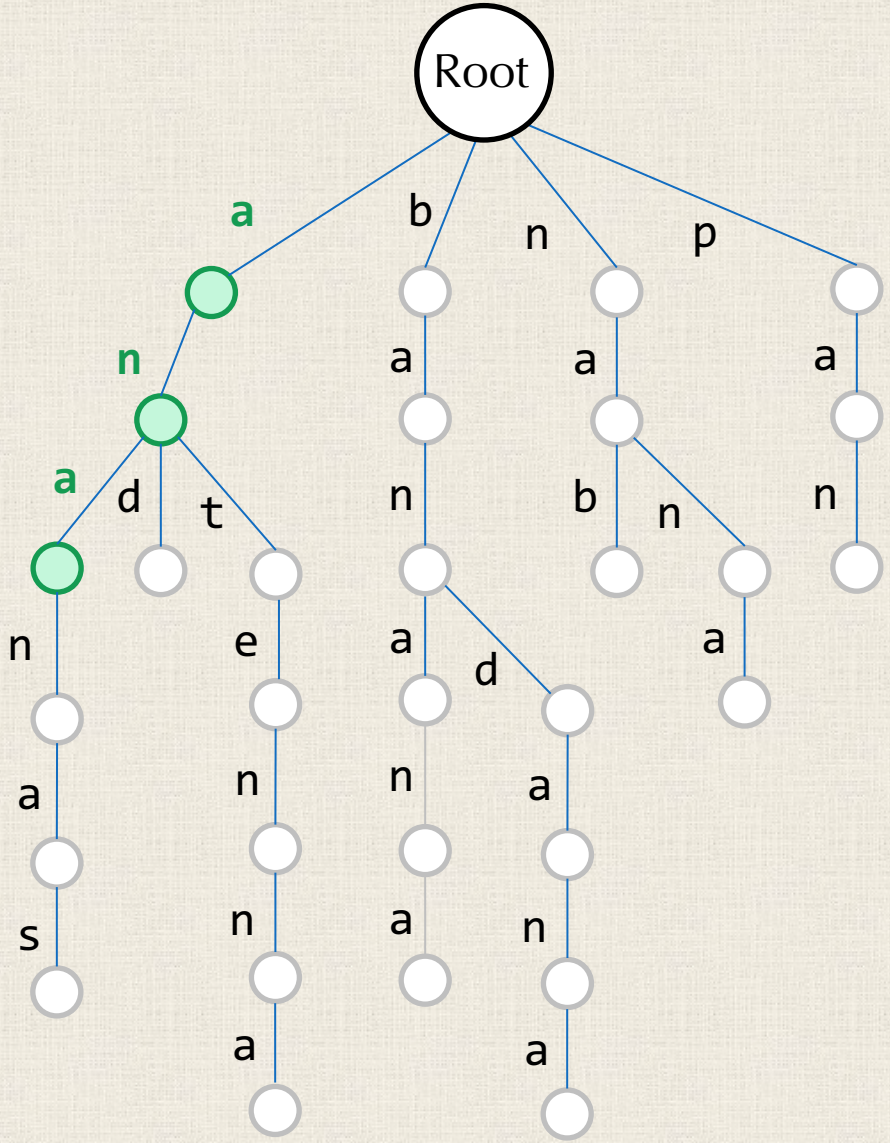
p a n a m a b a n a n a s

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8



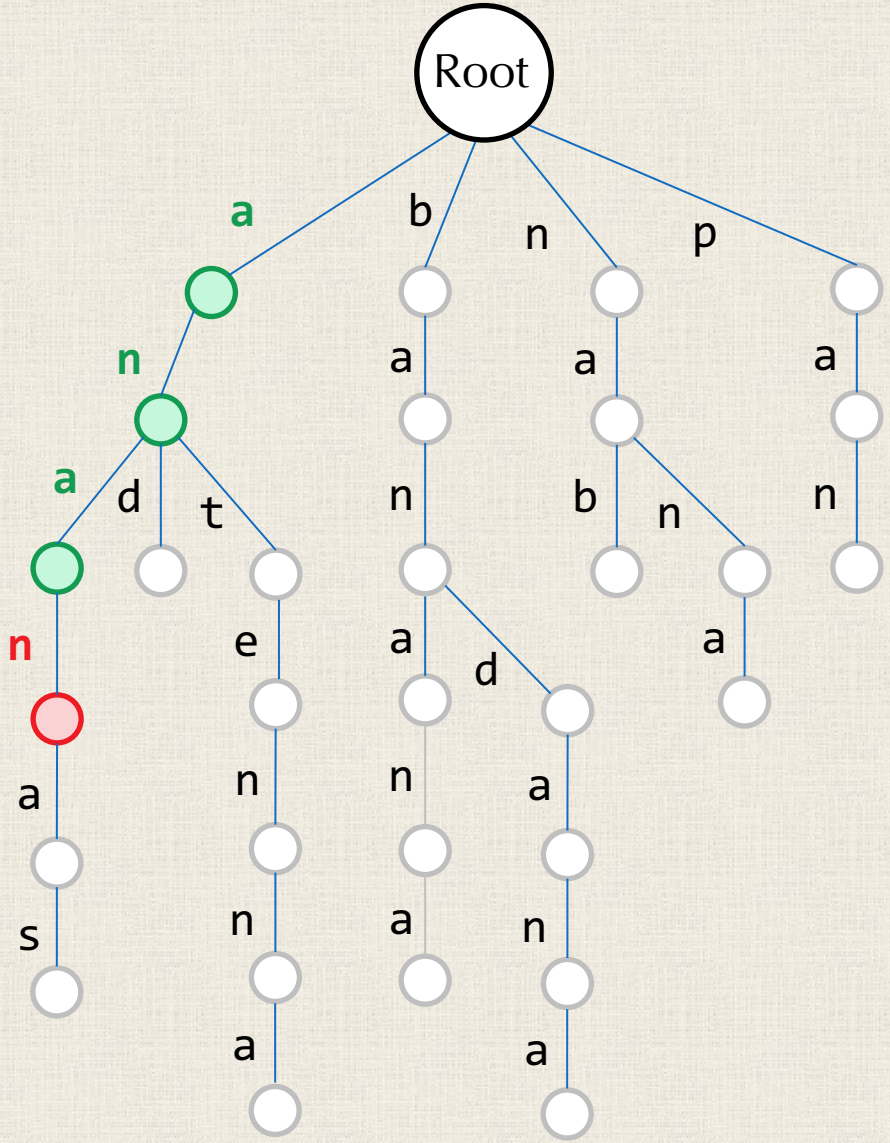
p a n a m a b a n a n a s

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8



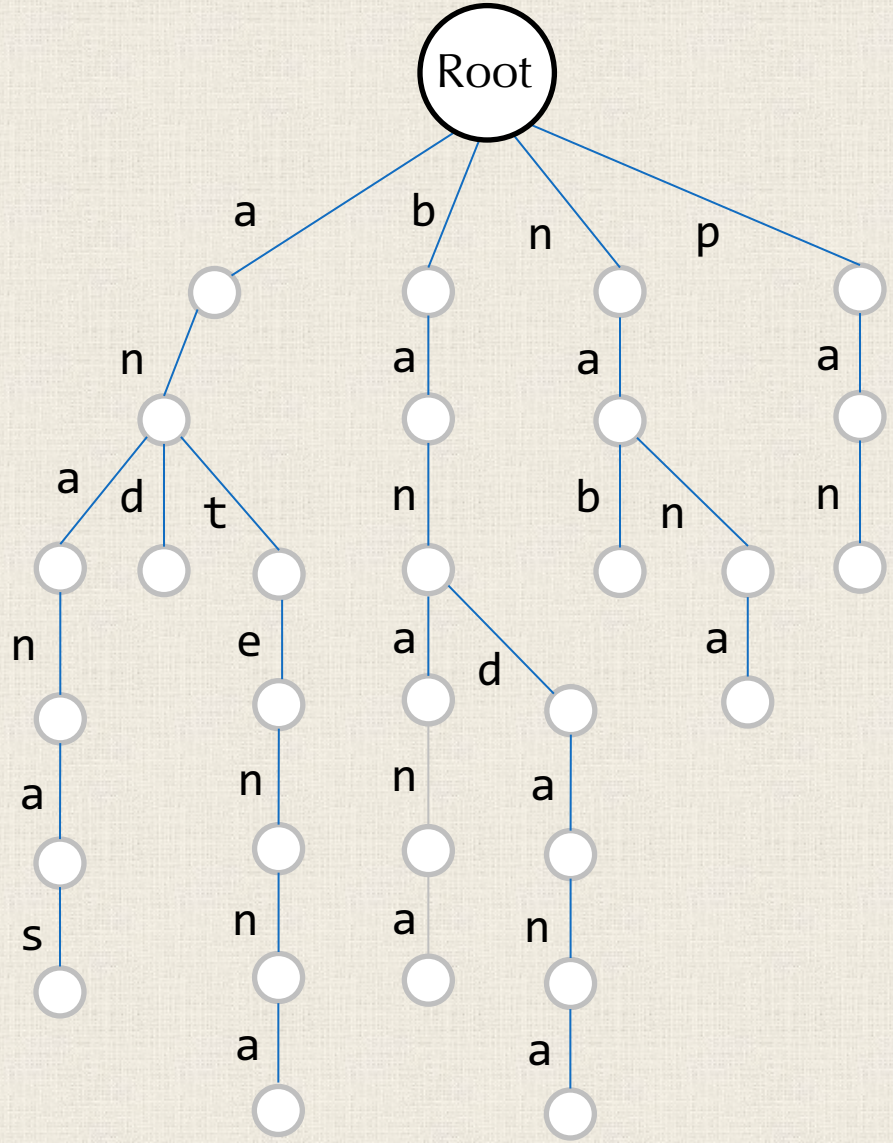
p a n a m a b a n a n a s

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8



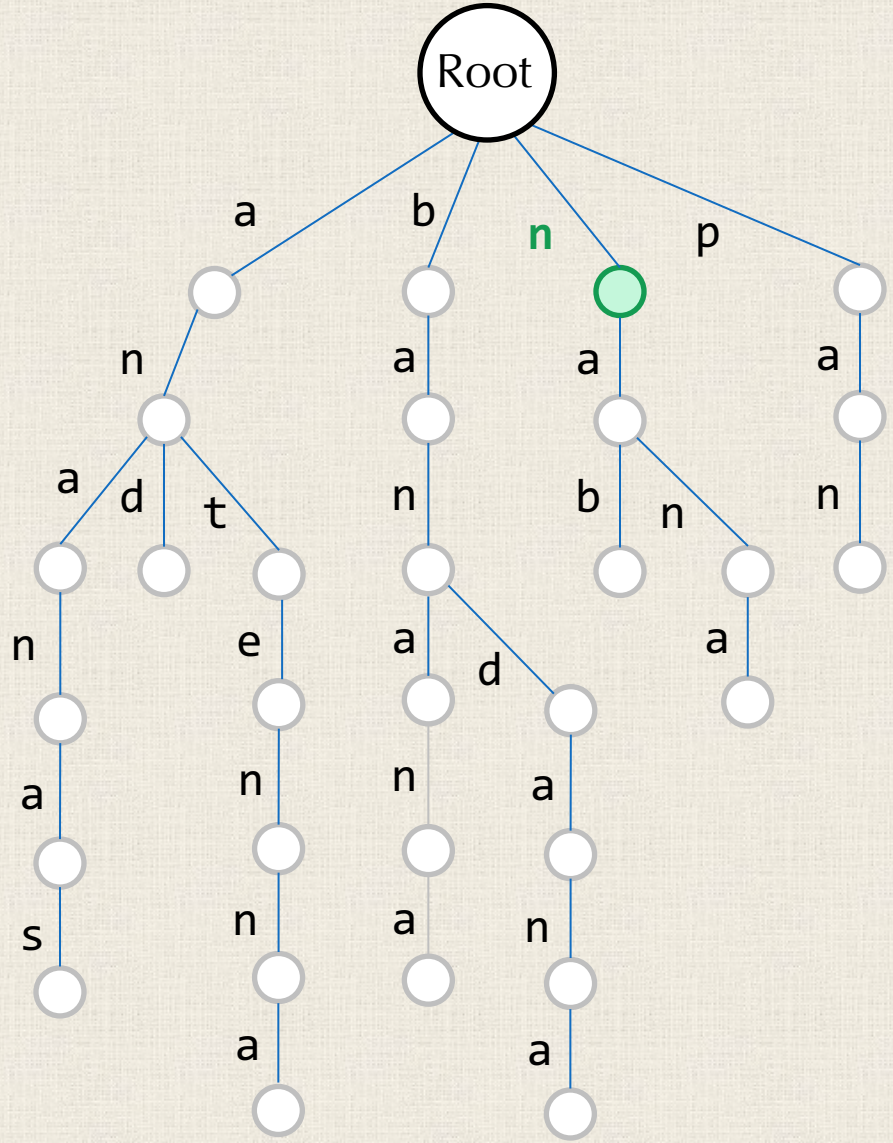
p a n a m a b a n a n a s

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8



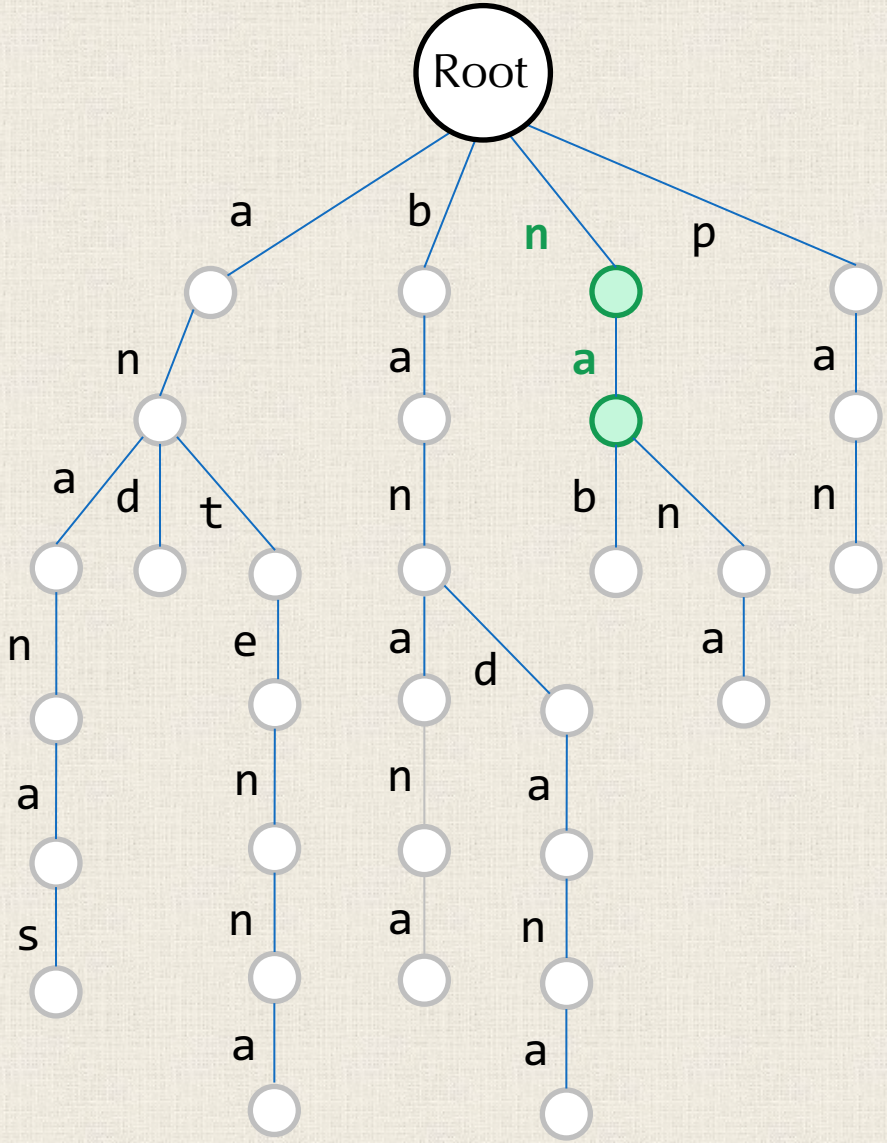
p a n a m a b a n a **n** a s

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8



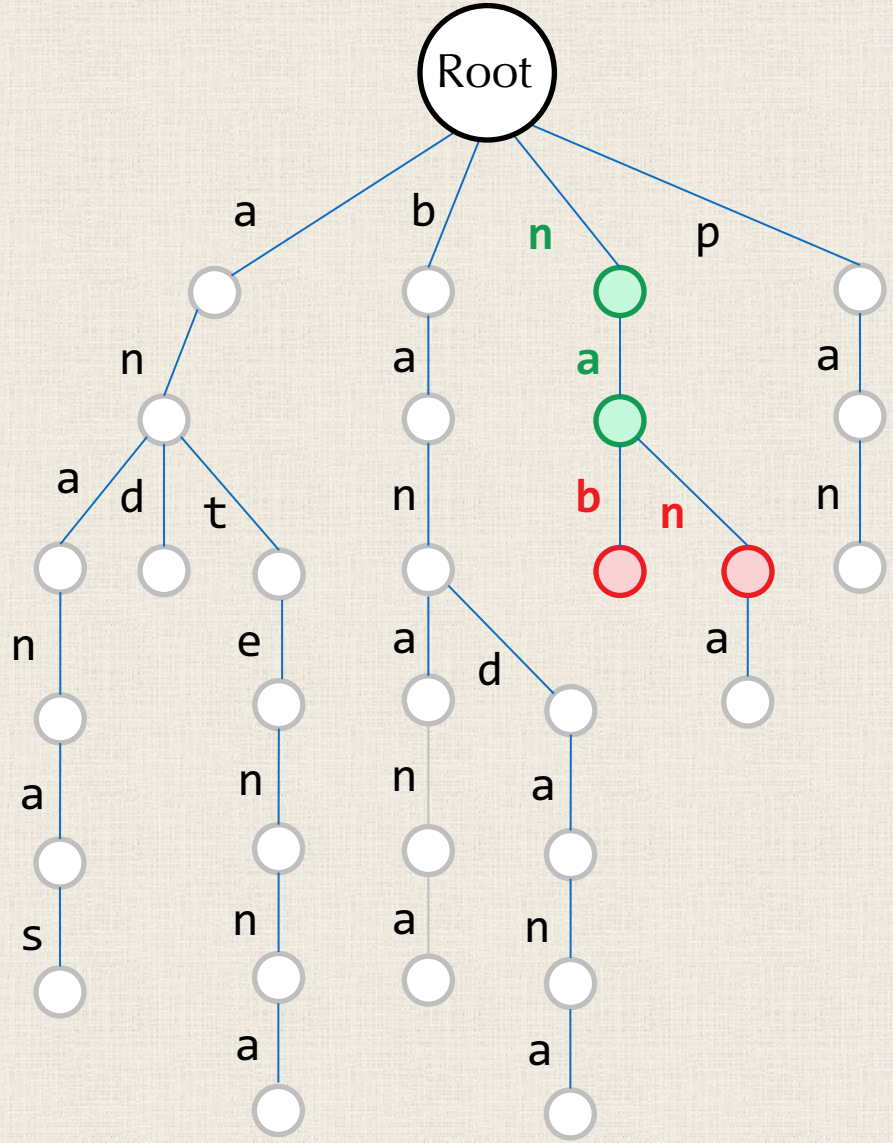
p a n a m a b a n a **n a** s

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8



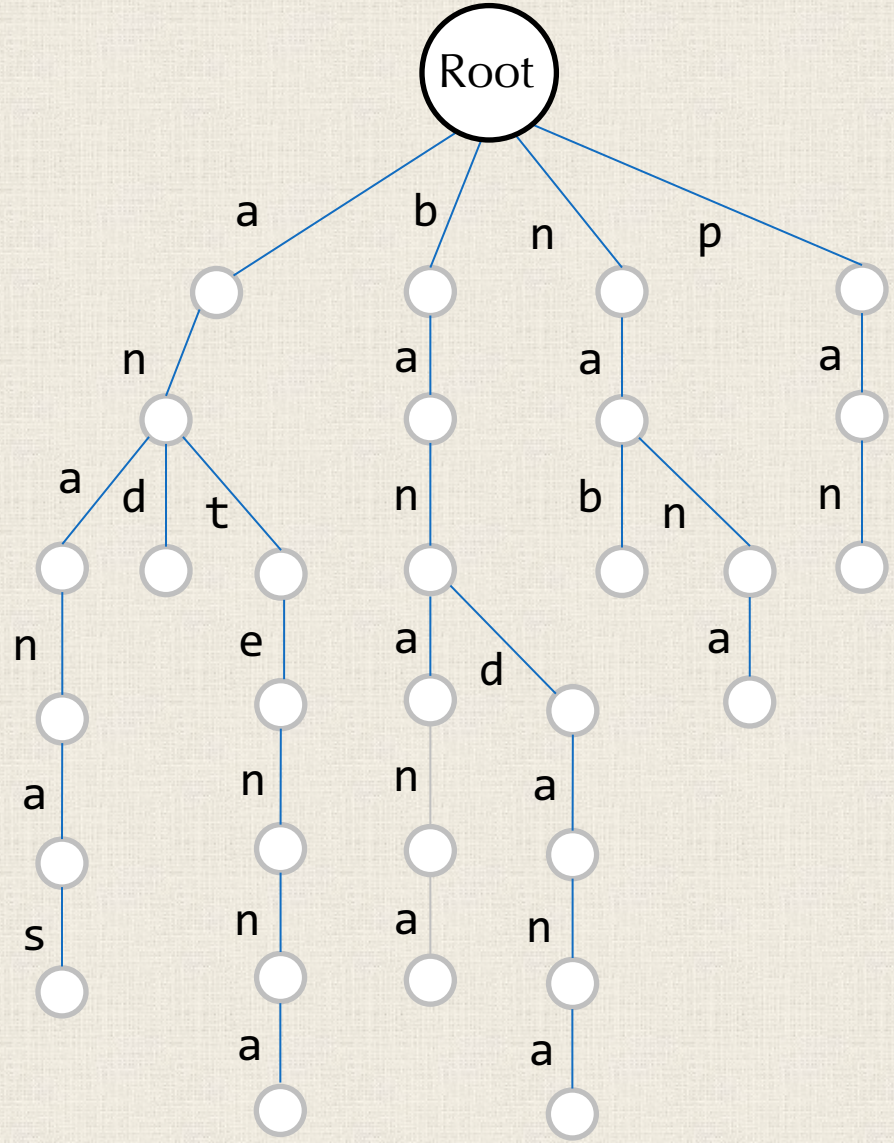
p a n a m a b a n a **n a s**

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8



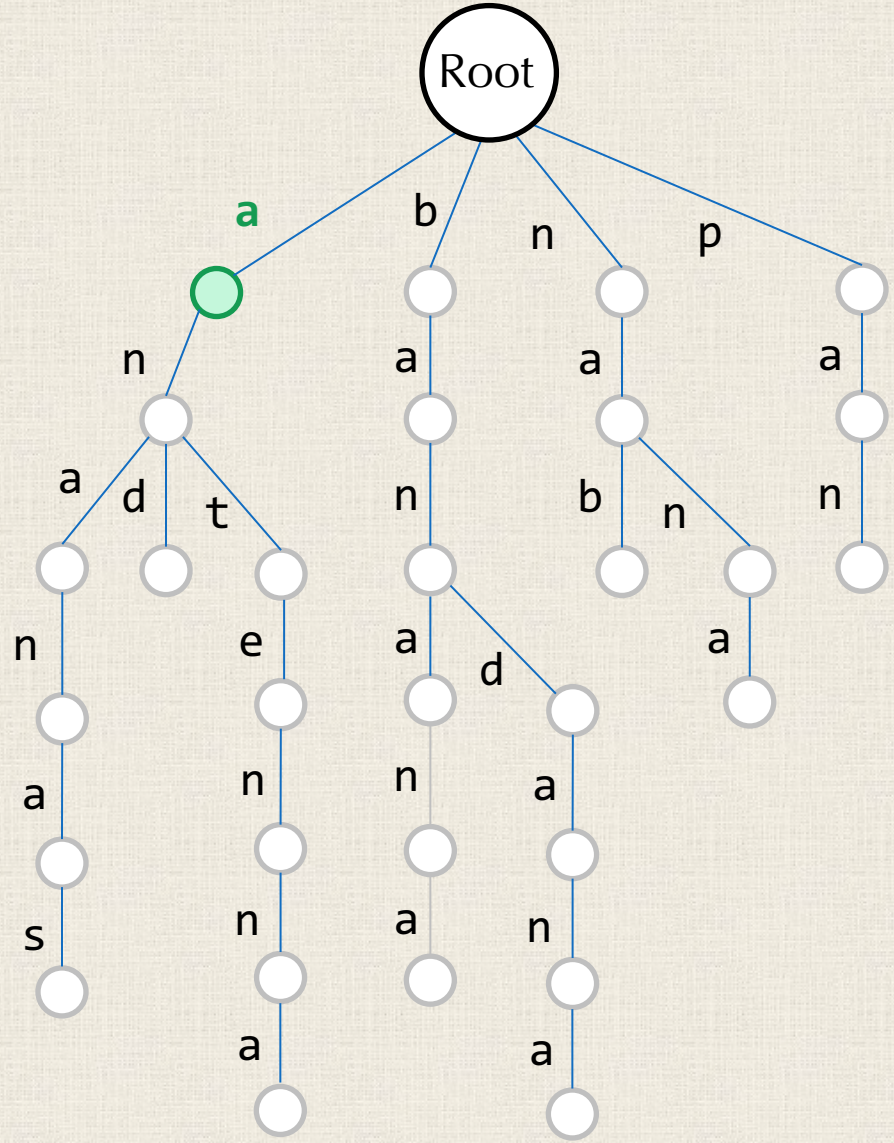
p a n a m a b a n a n a s

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8



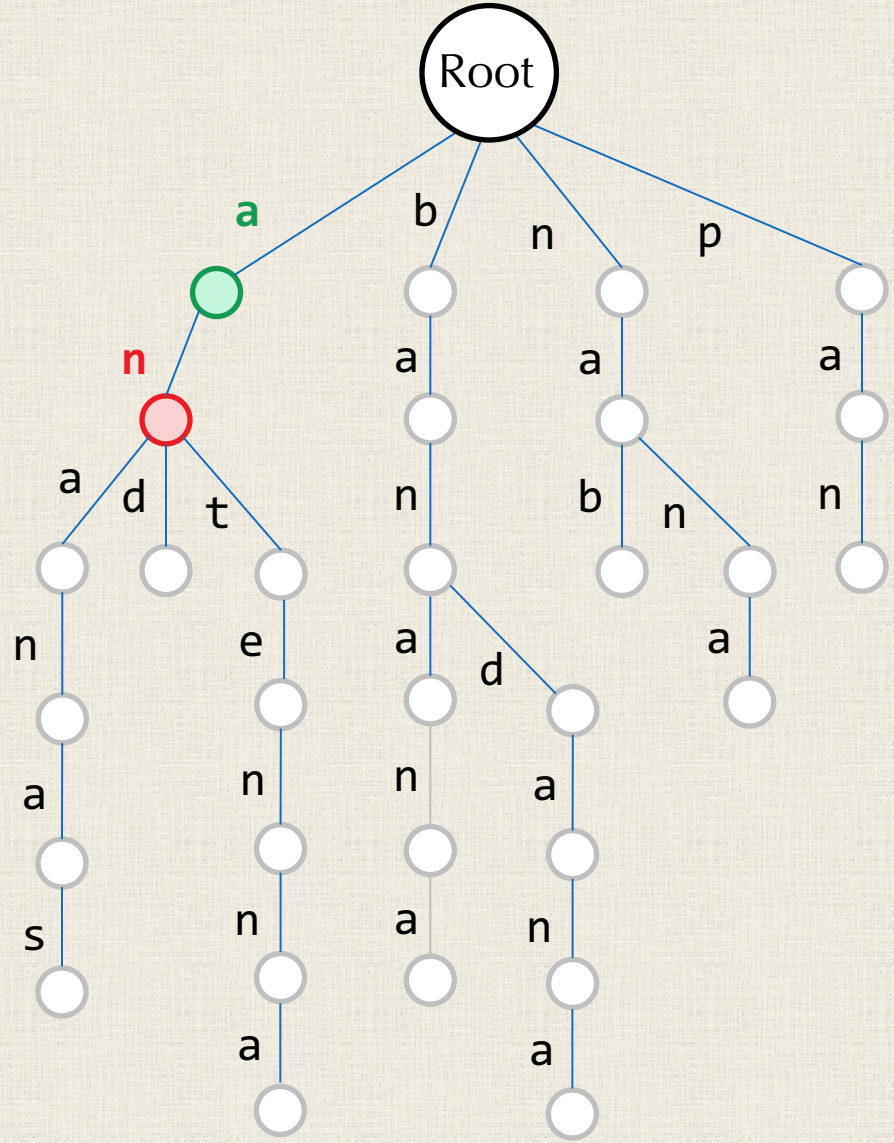
p a n a m a b a n a n a s

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8



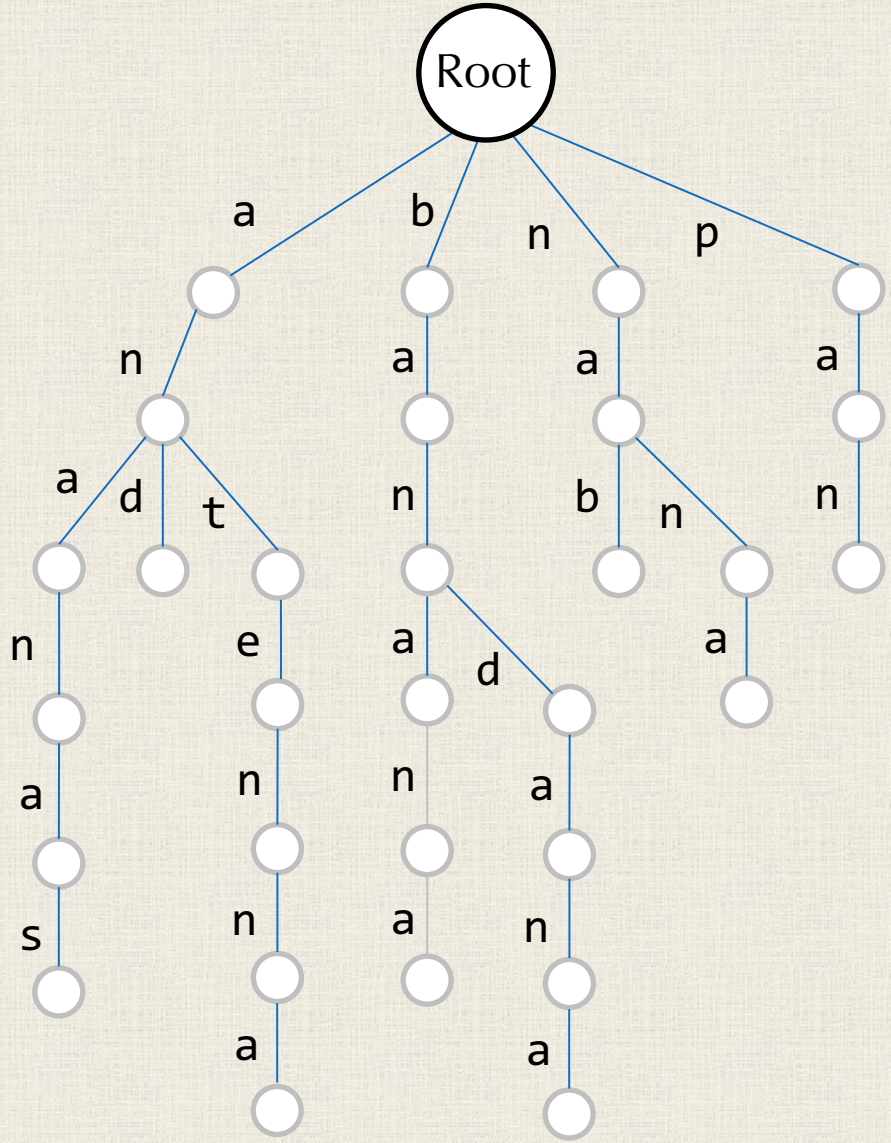
p a n a m a b a n a n a s

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8



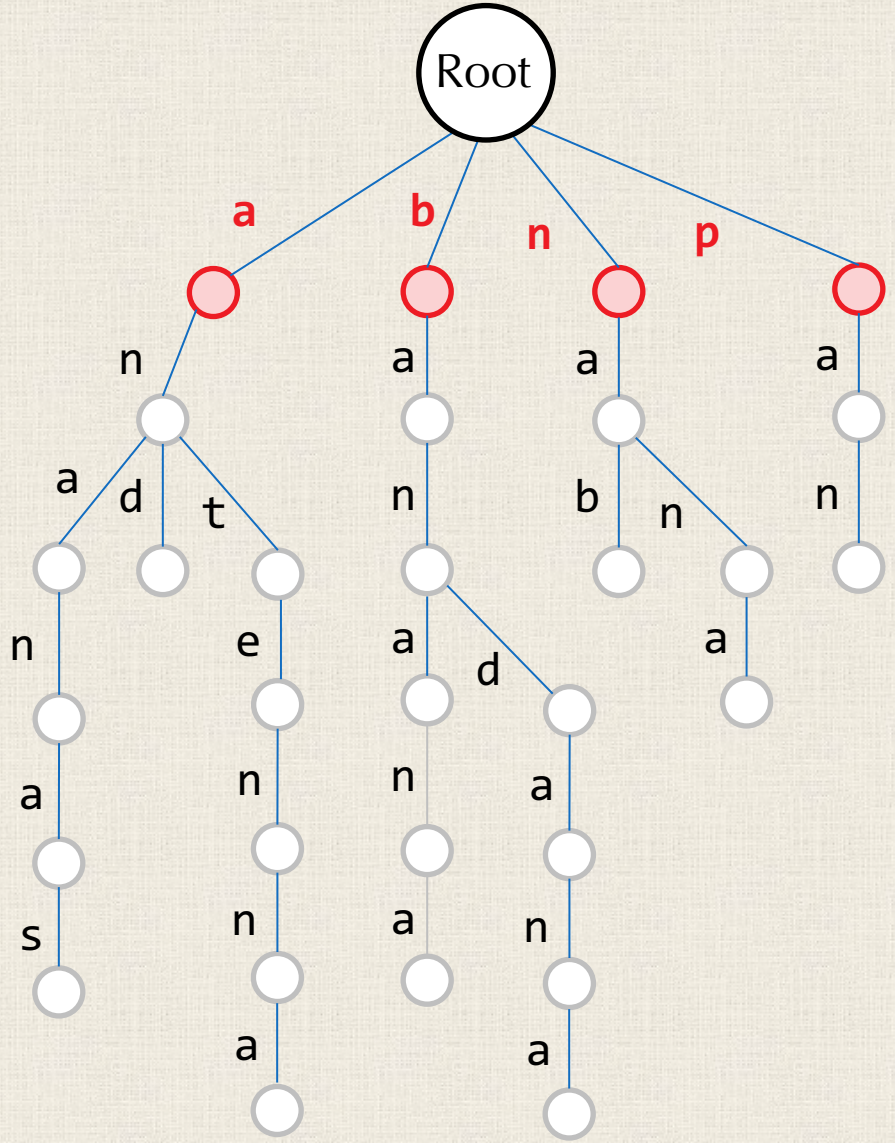
p a n a m a b a n a n a $

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8

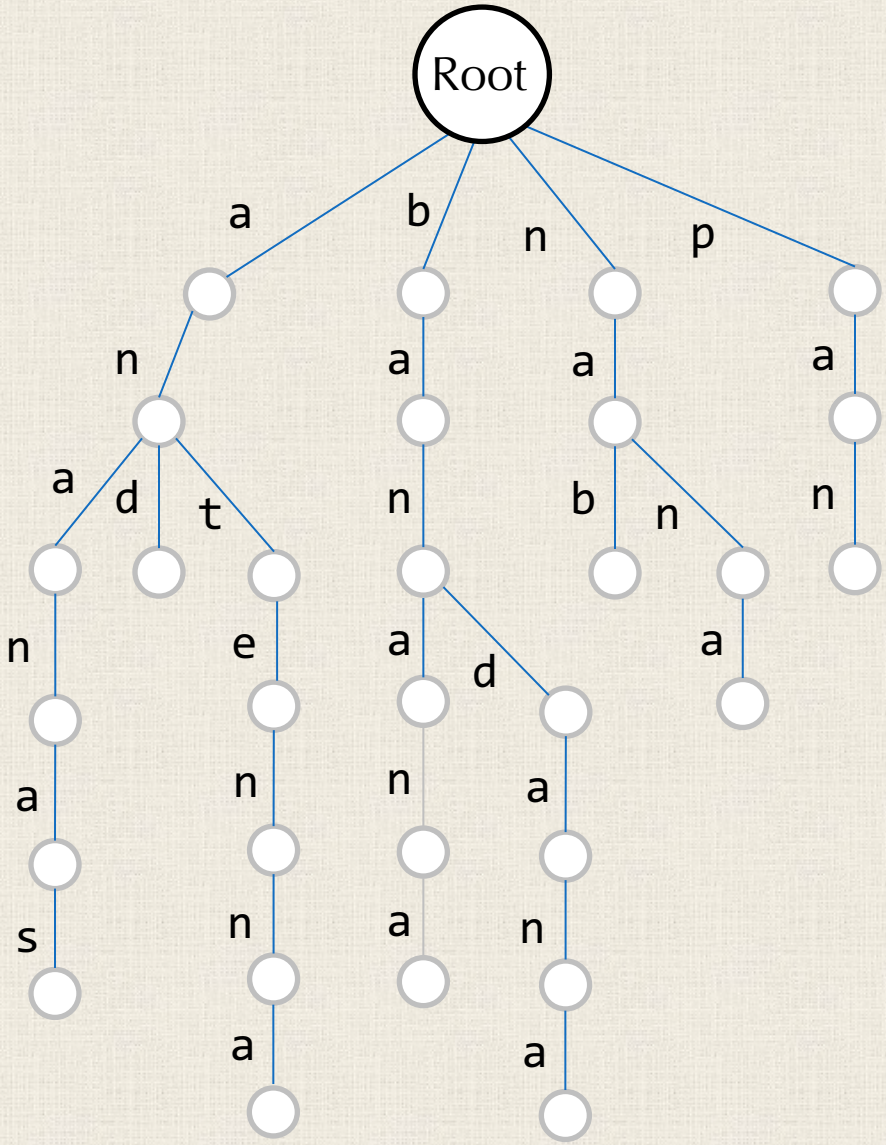


p a n a m a b a n a n a **s**

Pattern	Positions
pan	0
banana	6
ananas	7
nana	8

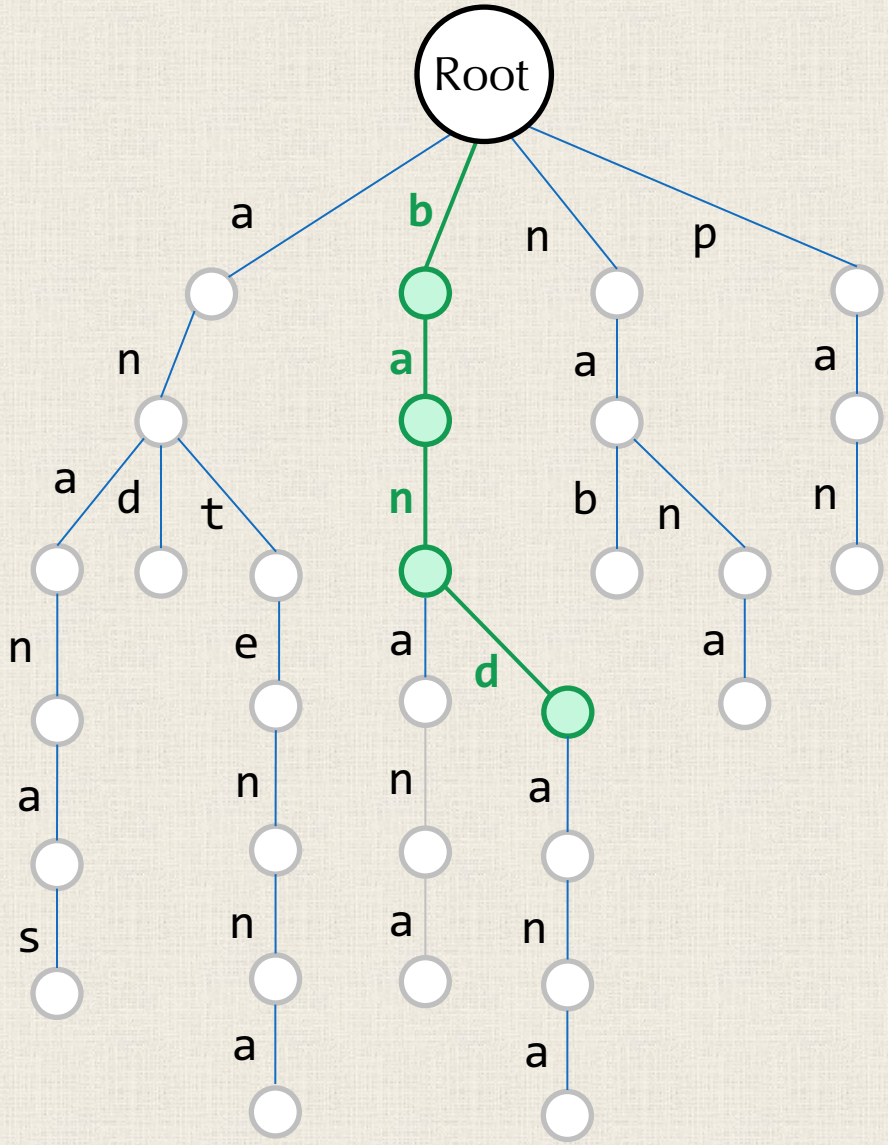


What if one patttern is a prefix of another?



STOP: If we add the pattern "band" to our dataset, why would it cause problems?

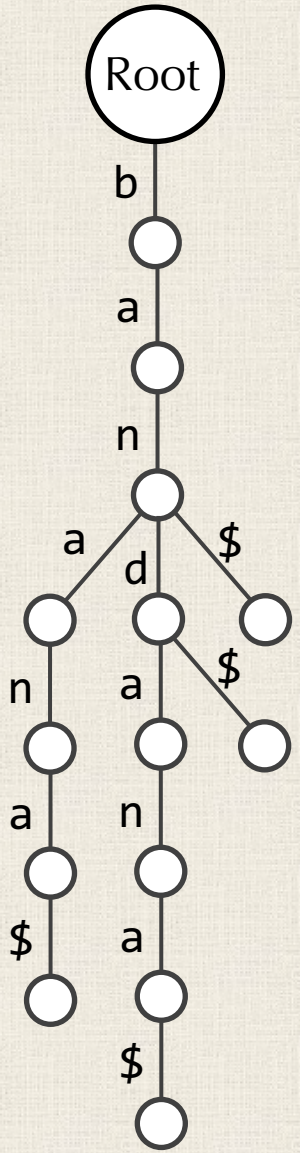
What if one patttern is a prefix of another?



STOP: If we add the pattern "band" to our dataset, why would it cause problems?

Answer: "band" is a *prefix* of "bandana" and would not end at a leaf node …

Adding an "End" Symbol Fixes the Issue



Patterns

ban$
banana$
band$
bandana$

Success?

Runtime of Brute Force: O(|*Text*|*|*Patterns*|)

Success?

Runtime of Brute Force: $O(|Text|*|Patterns|)$

STOP: What is the running time of multiple pattern matching with a trie?

Success?

Runtime of Brute Force: O(|*Text*|*|*Patterns*|)

STOP: What is the running time of multiple pattern matching with a trie?

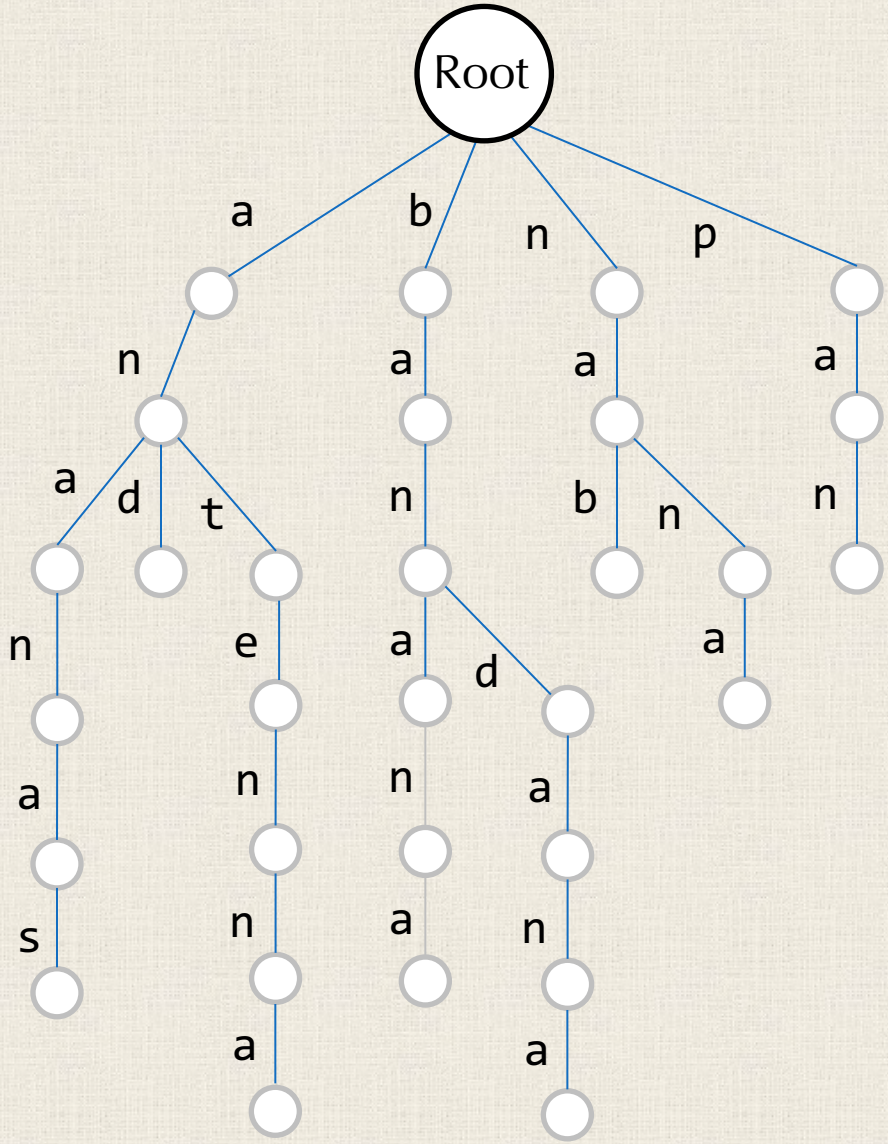
Runtime of Pattern Matching with Trie:

- Trie Construction: O(|*Patterns*|)
- Pattern Matching: O(|*Text*| * |*LongestPattern*|)

But We Neglected Memory …

Our trie has 30 edges (38 when we add the $ symbol), and |*Patterns*| = 39.

Worst case: Trie takes up O(|*Patterns*|), which for a human sequencing project may be ~1 TB!



PREPROCESSING THE GENOME INSTEAD

Preprocessing the Genome

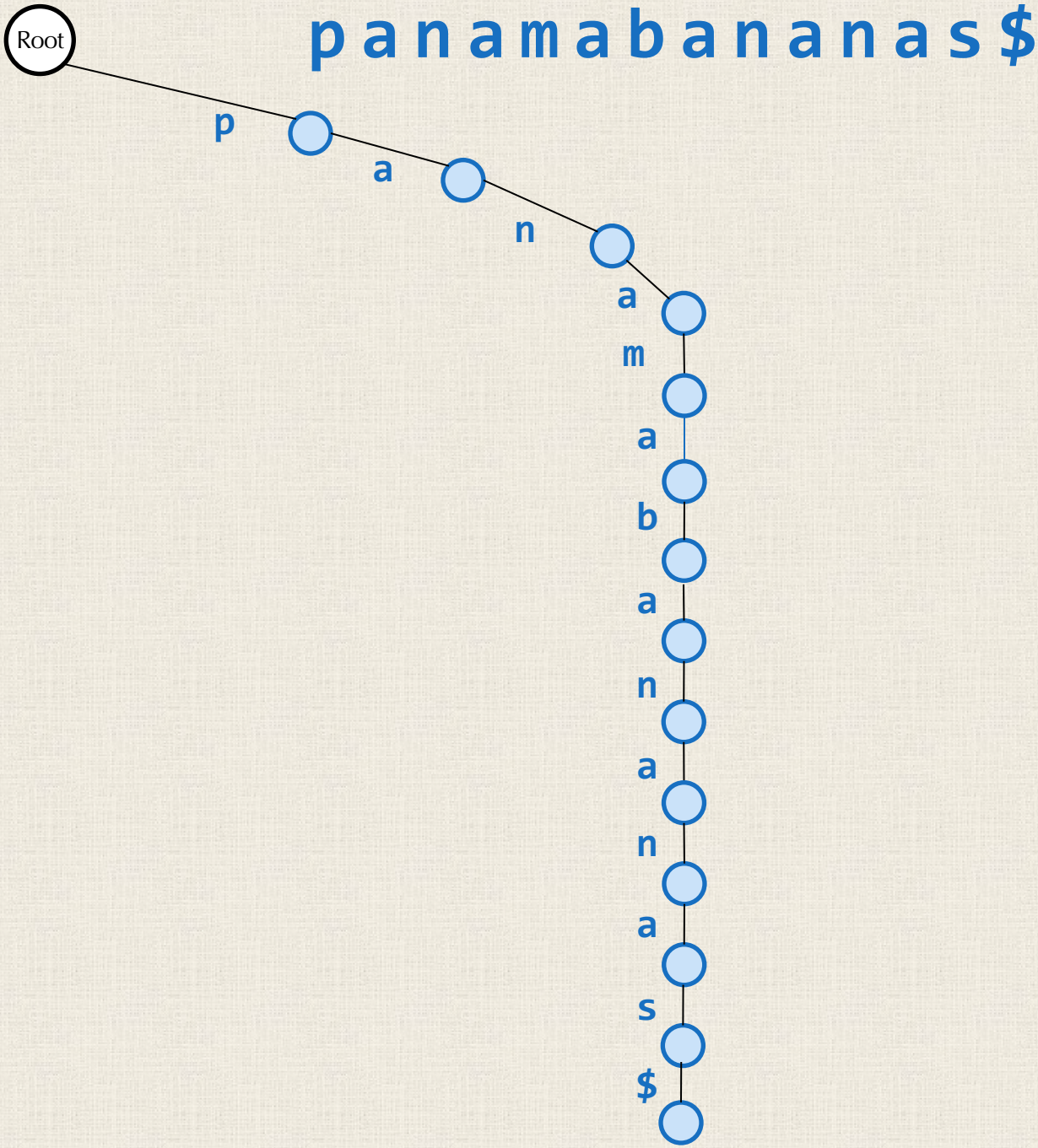
If |*Patterns*| > |*Text*|, then why not try to construct a data structure from *Text*?

This would take less memory and allow us to "teleport" each string *Pattern* to its occurrences in *Text*.

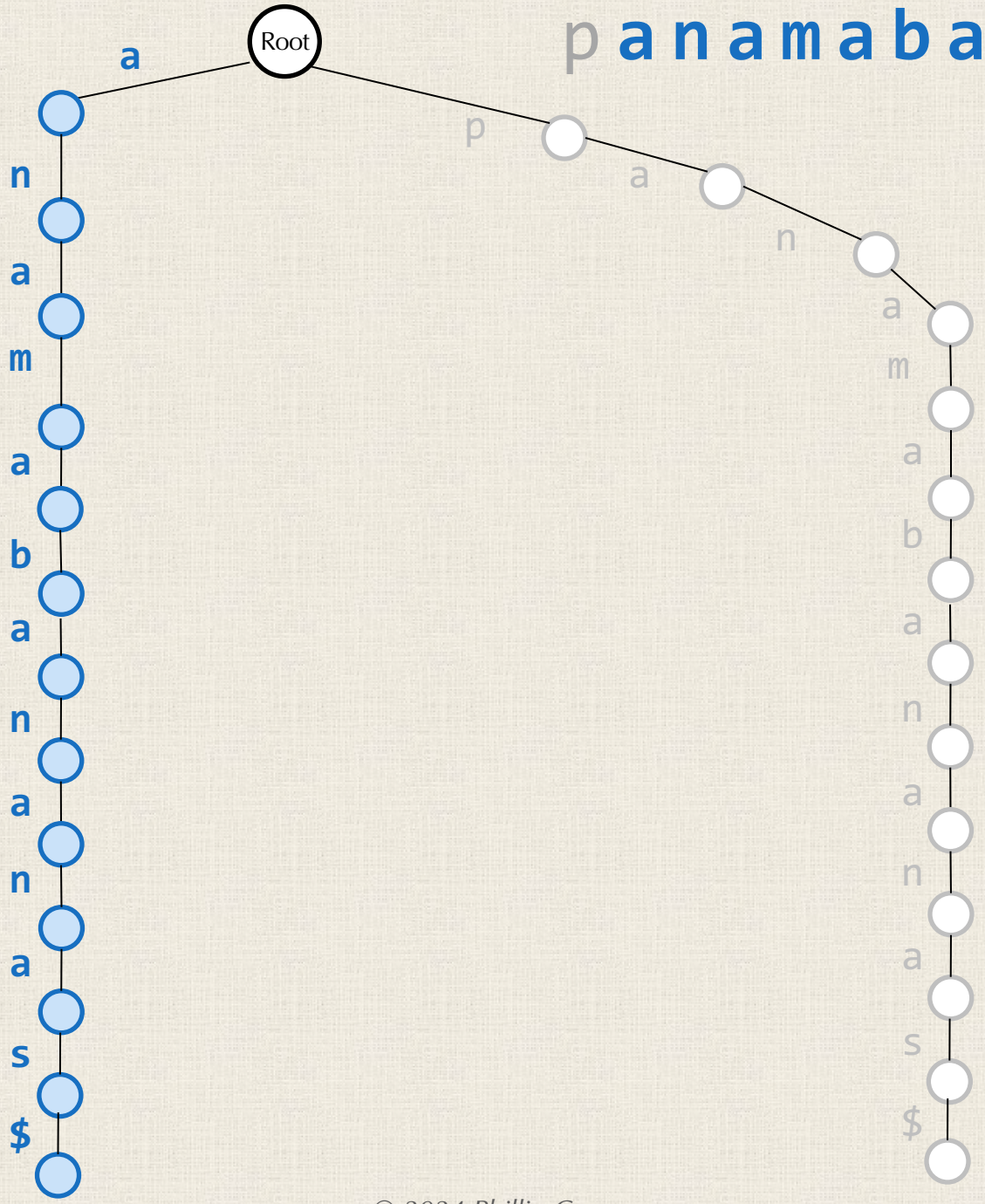
panamabananas$

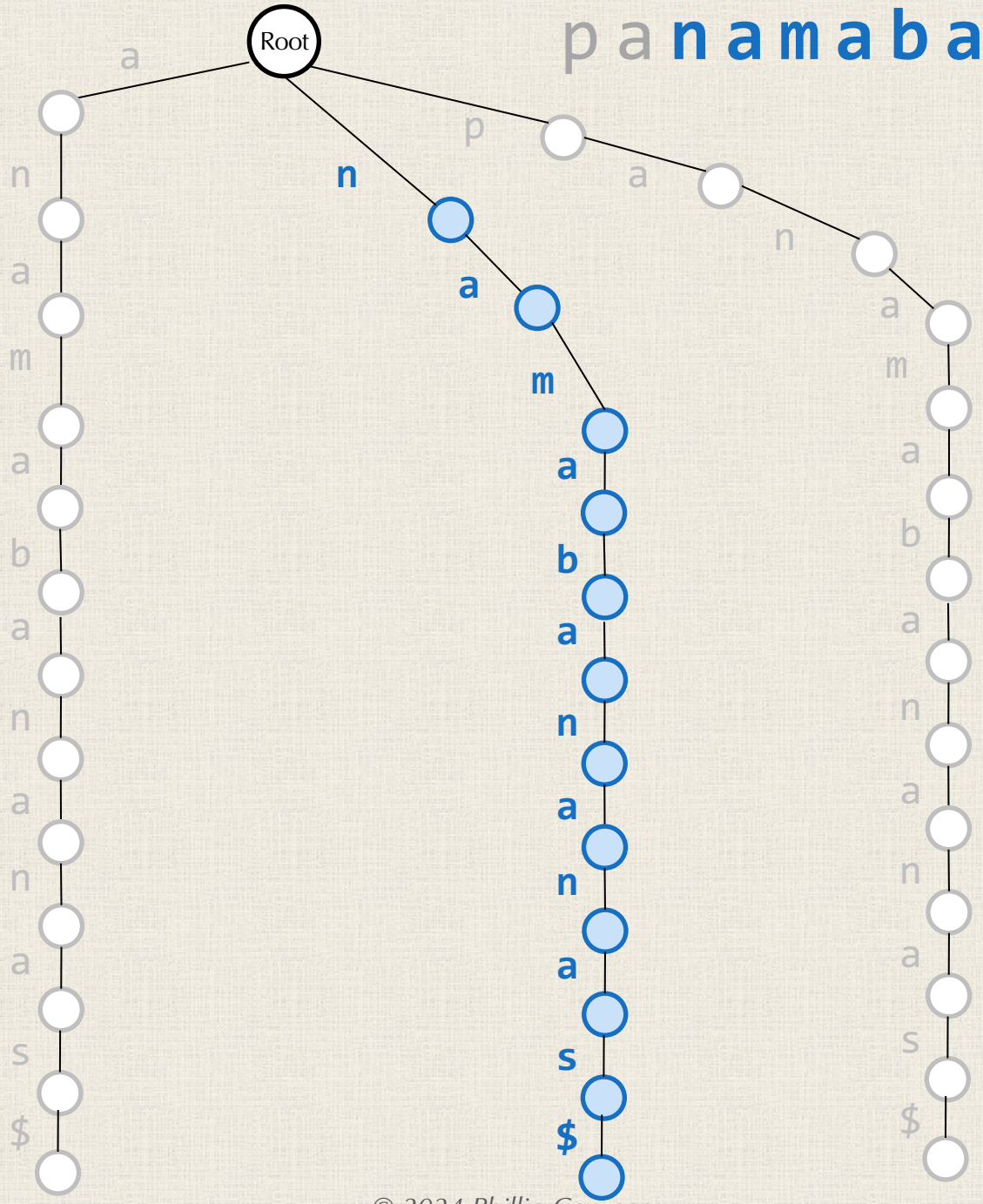
Suffix trie: Trie formed from the *suffixes* of *Text*.

Note: We augment *Text* with a $ to mark the end of the string (why?)

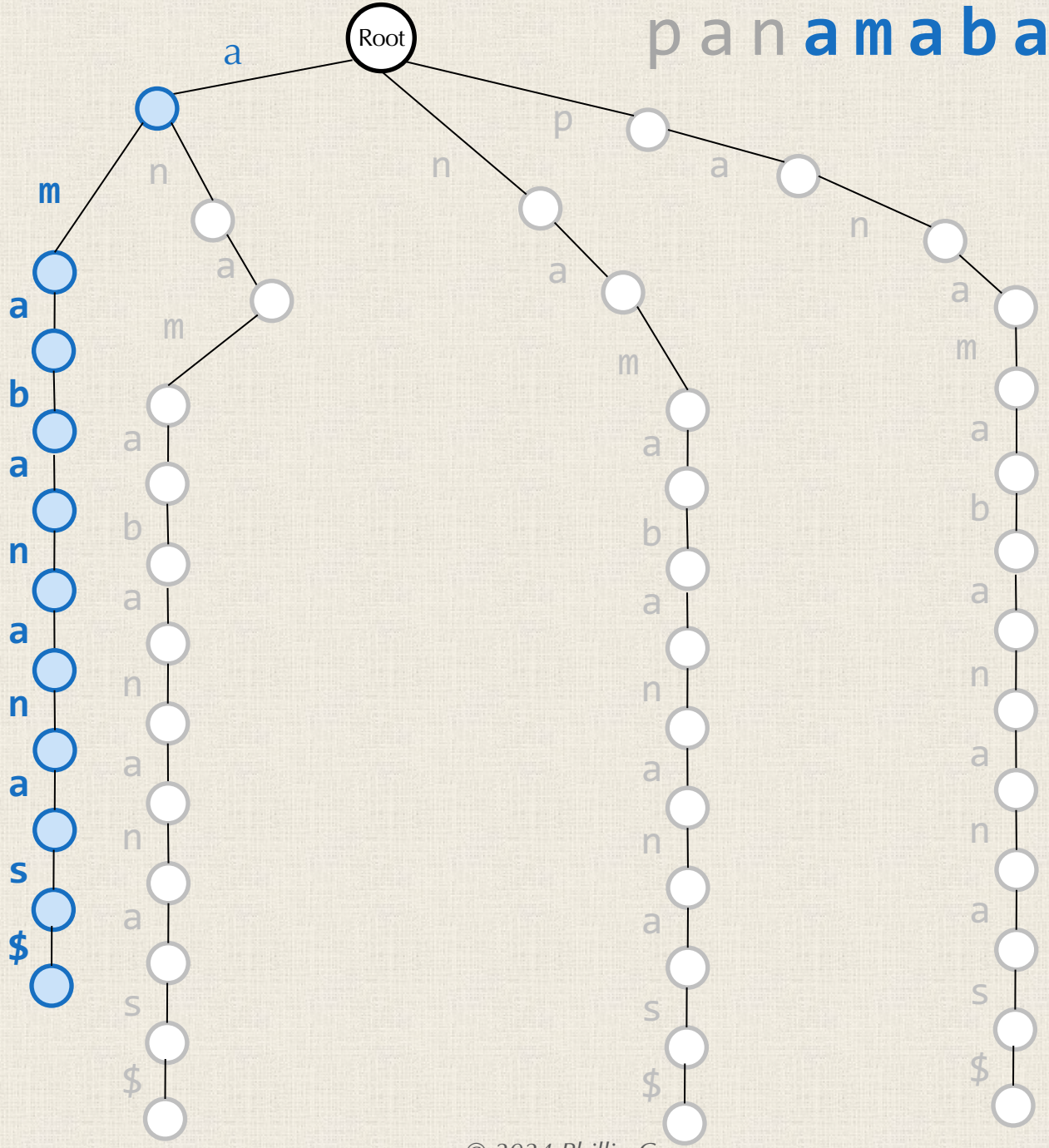


panamabananas$

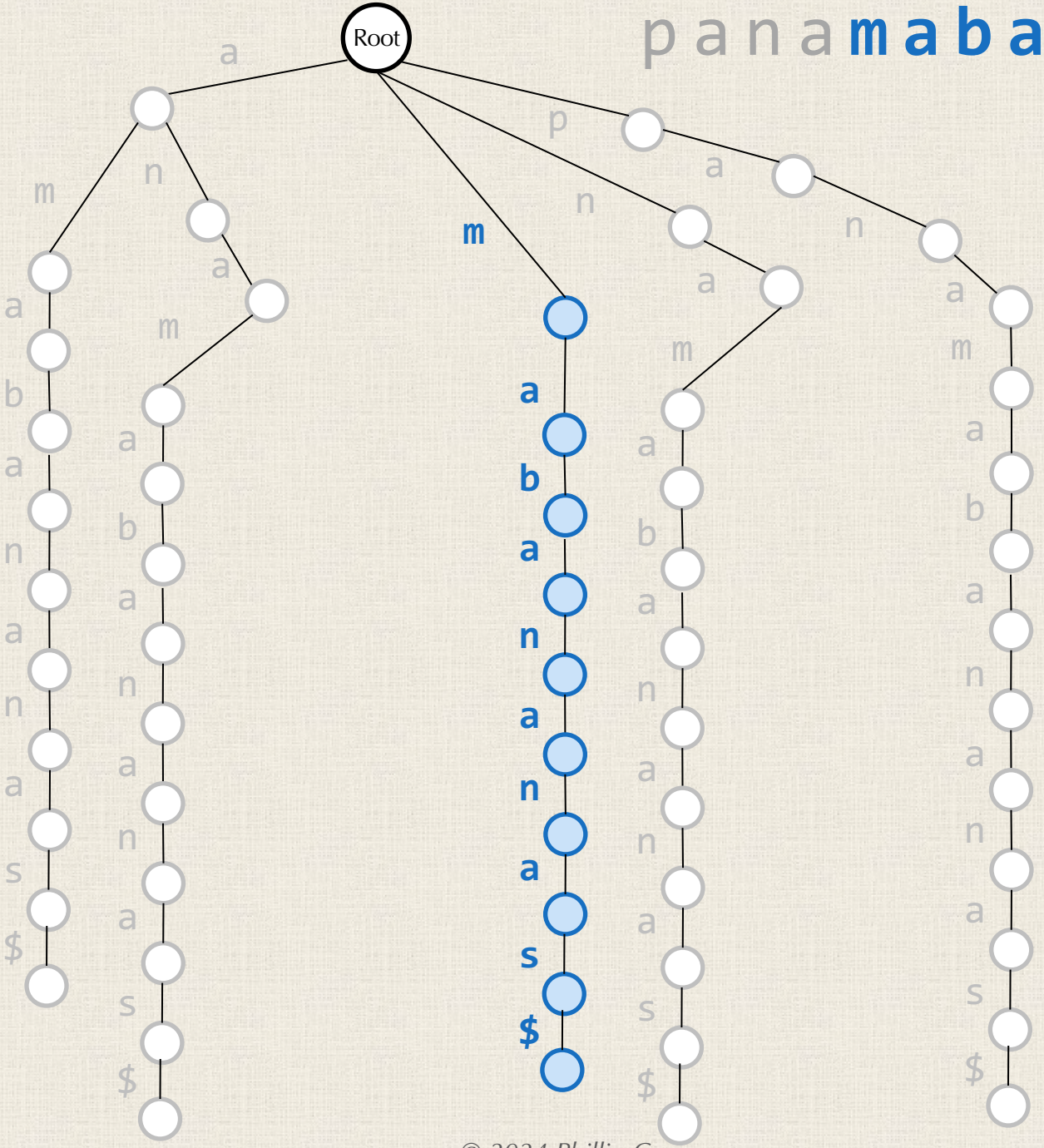




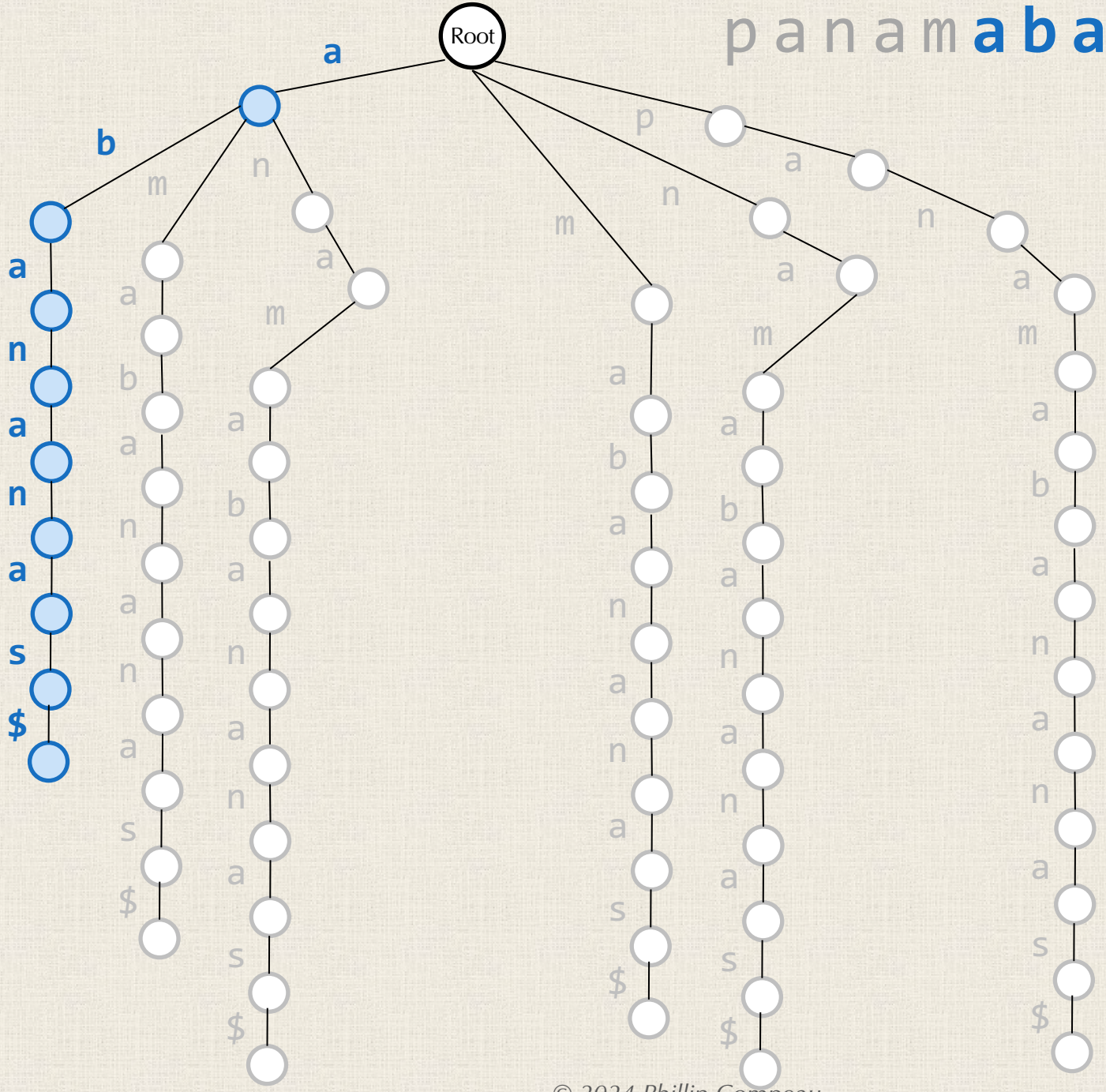
panamabananas$

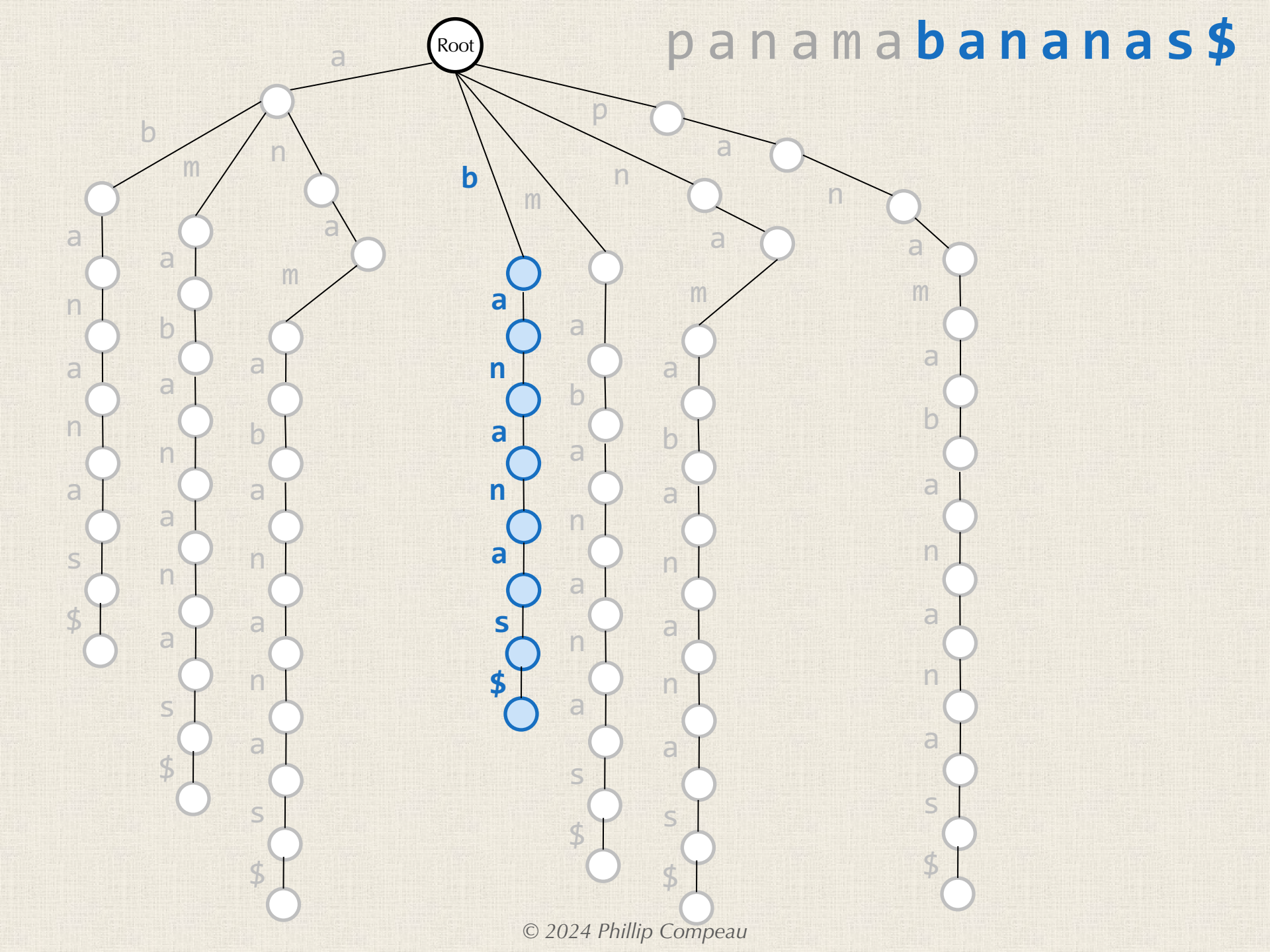


panama**bananas$**

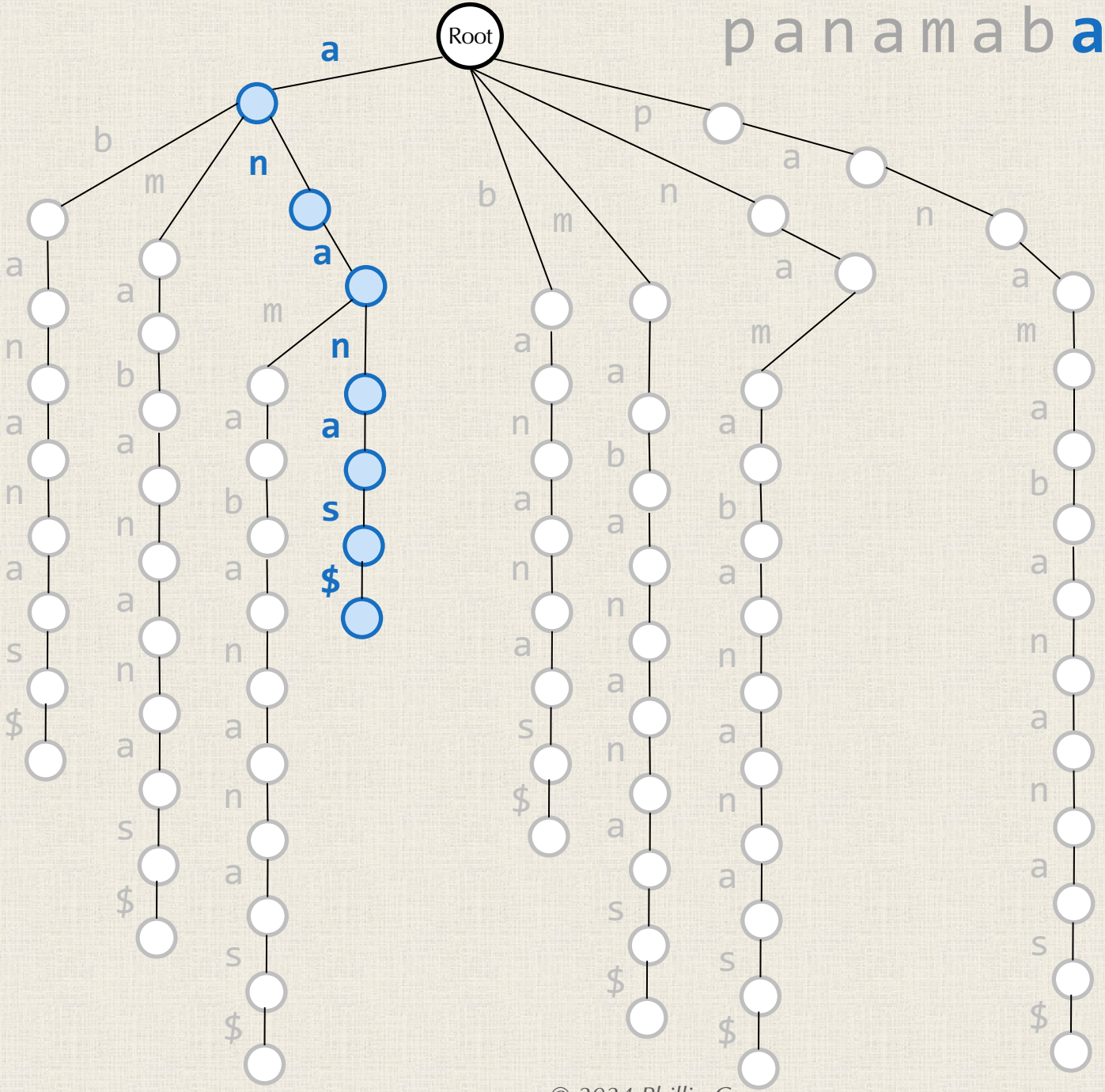


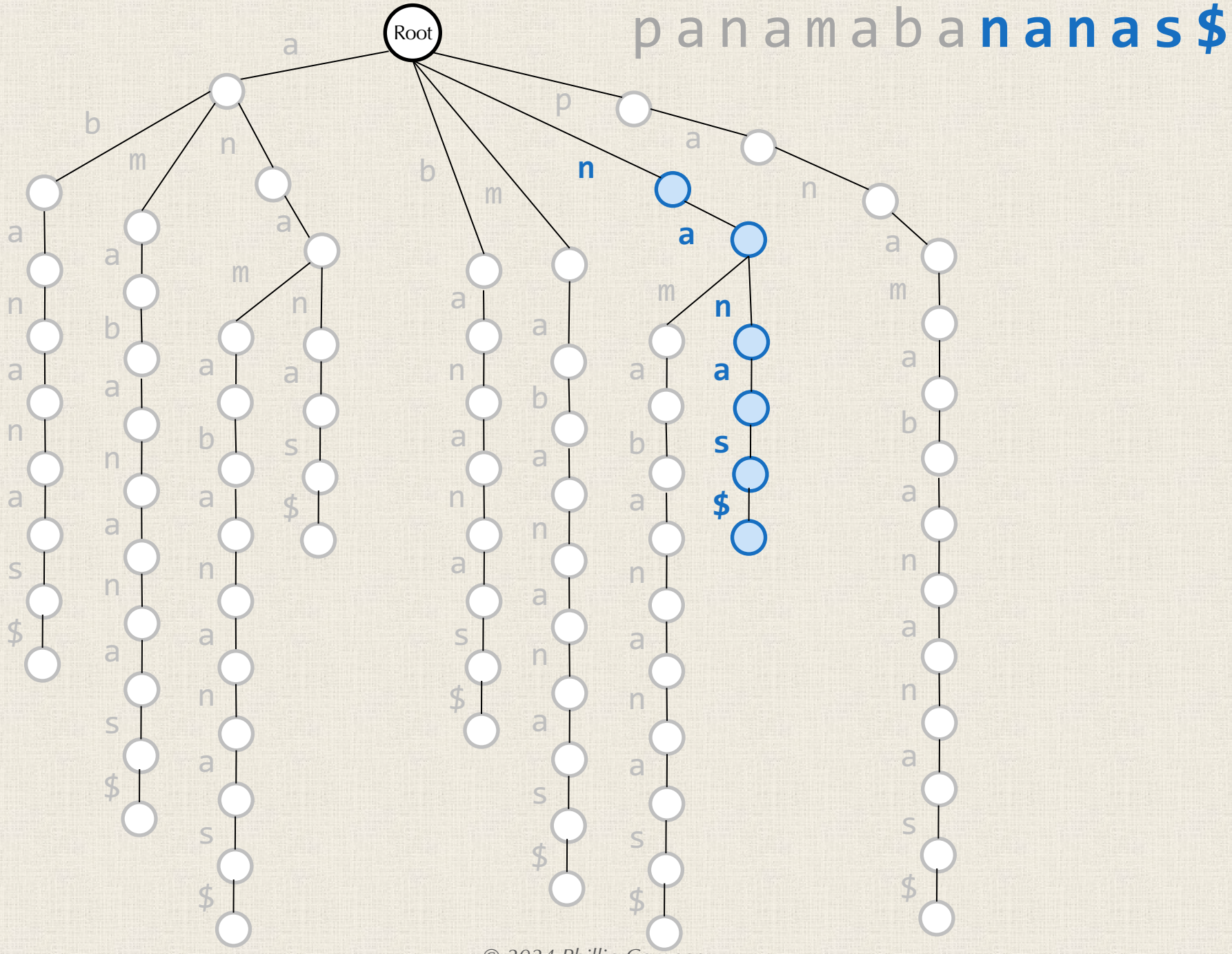
panam**abananas$**

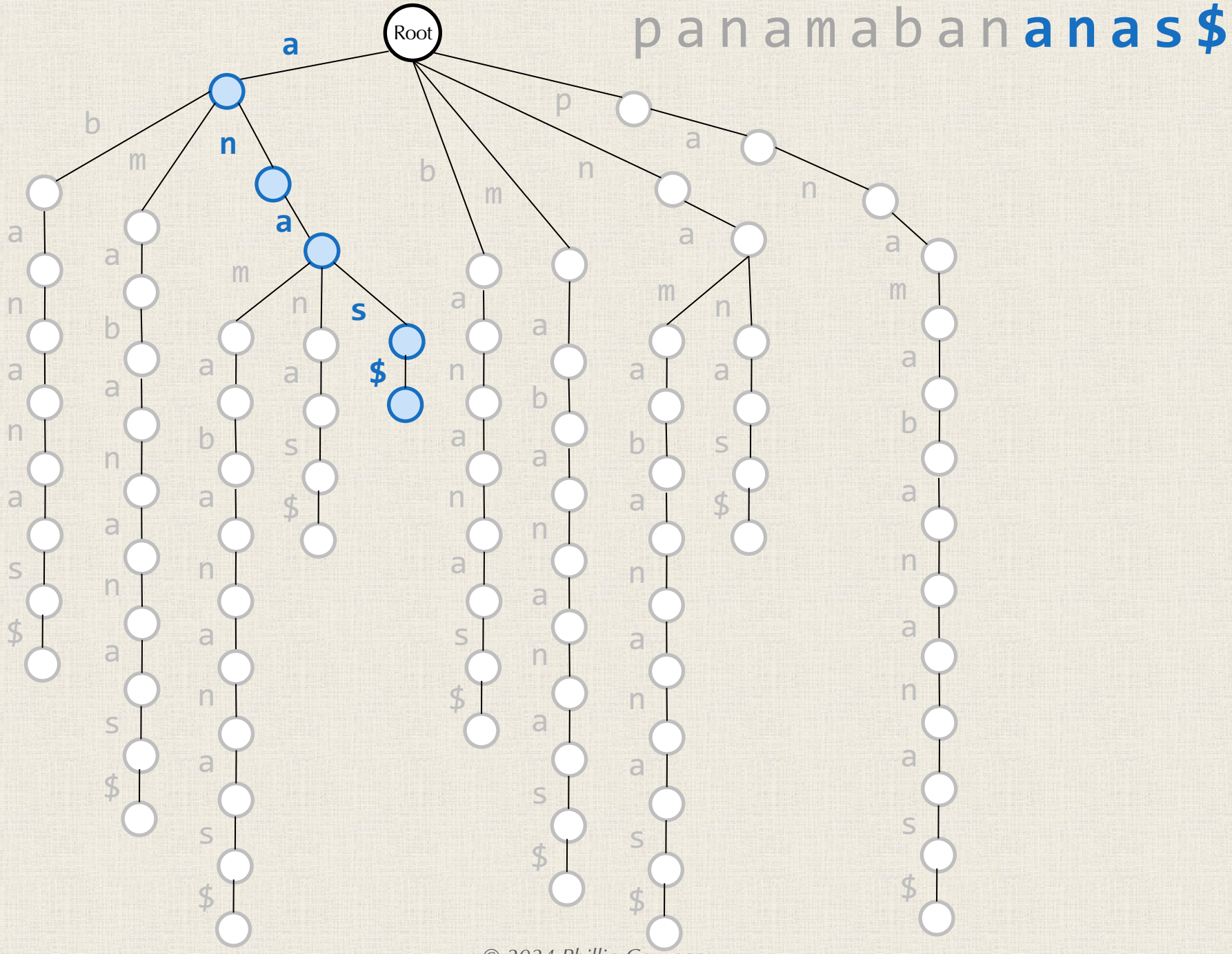


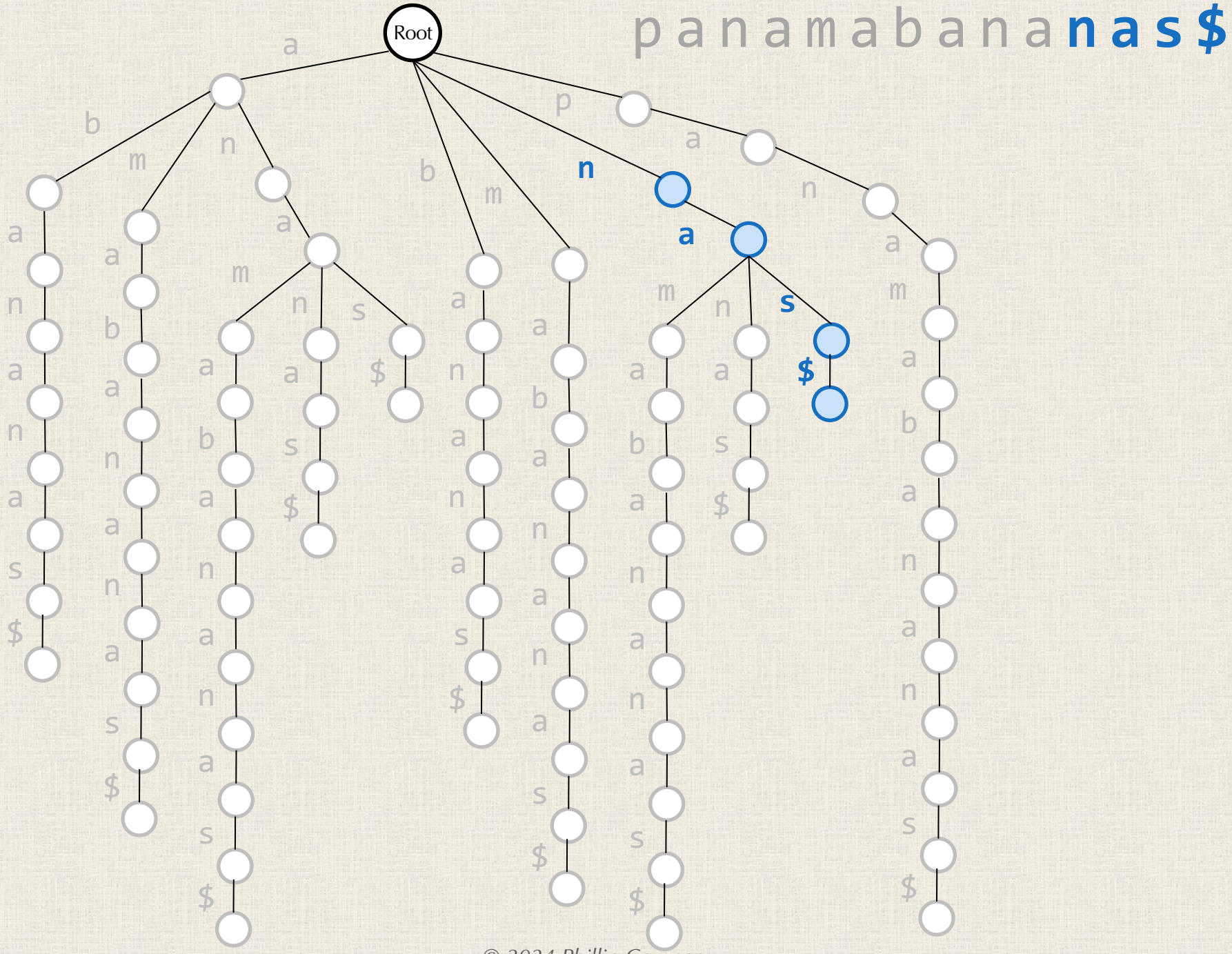


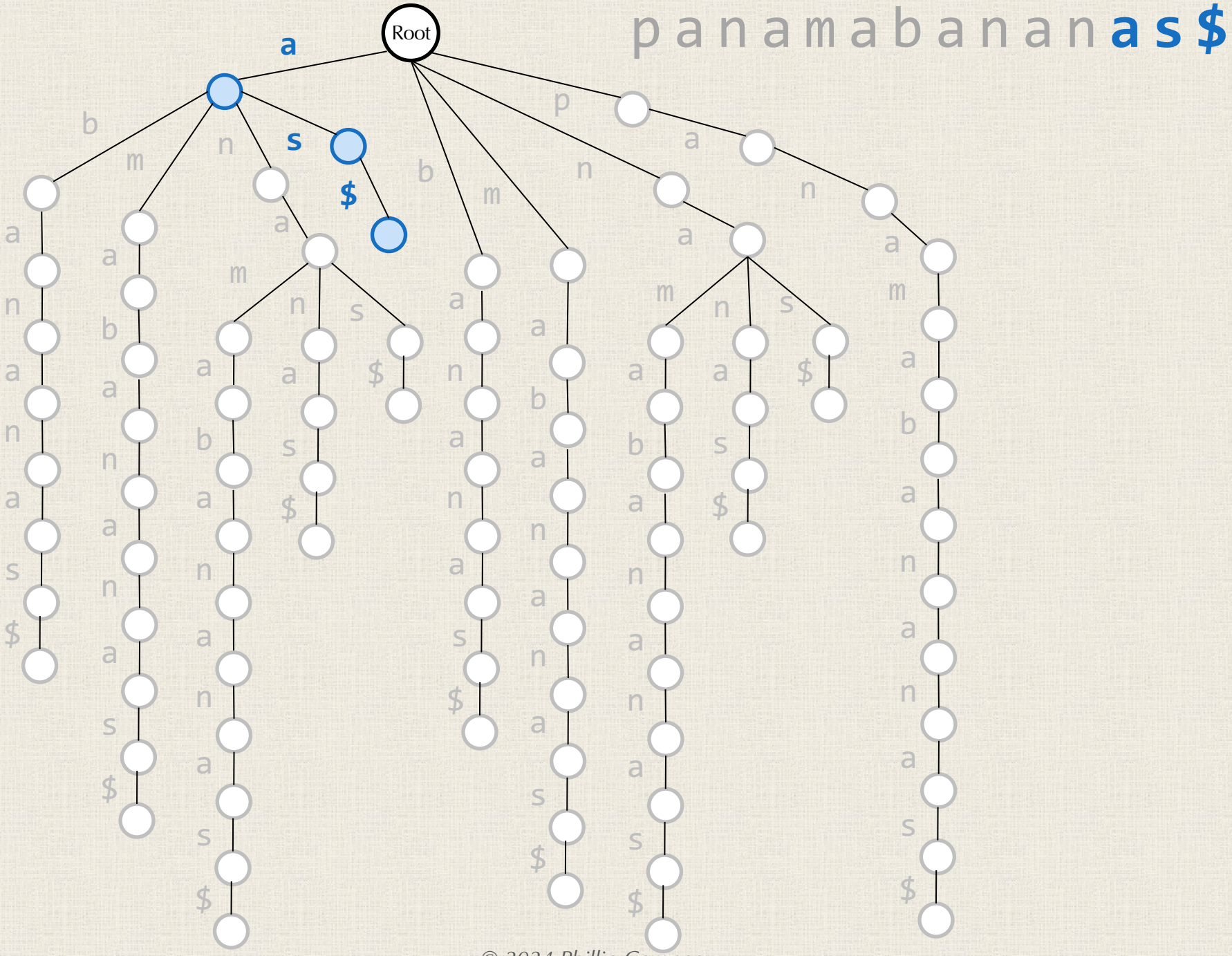
panamab**ananas$**

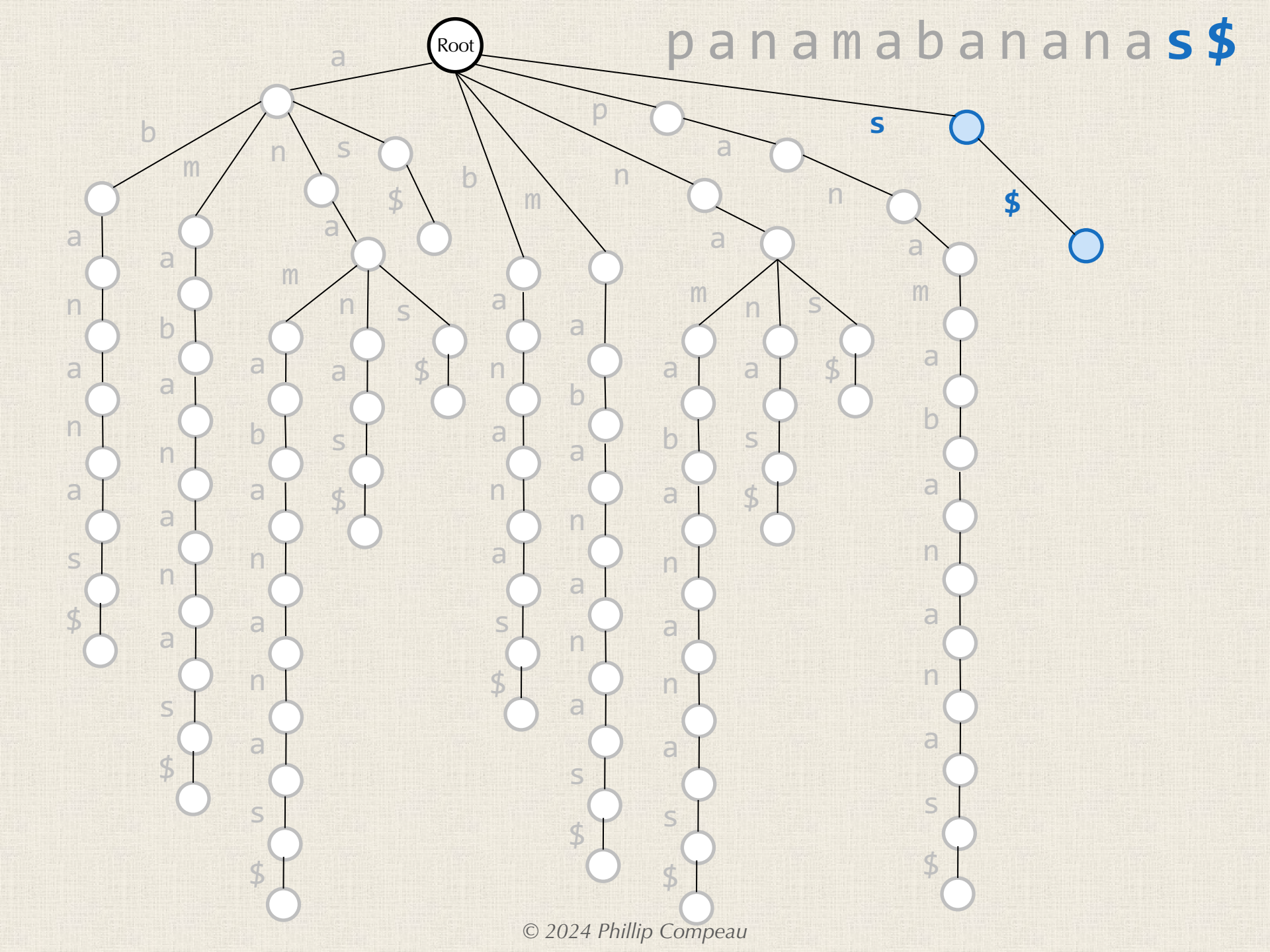


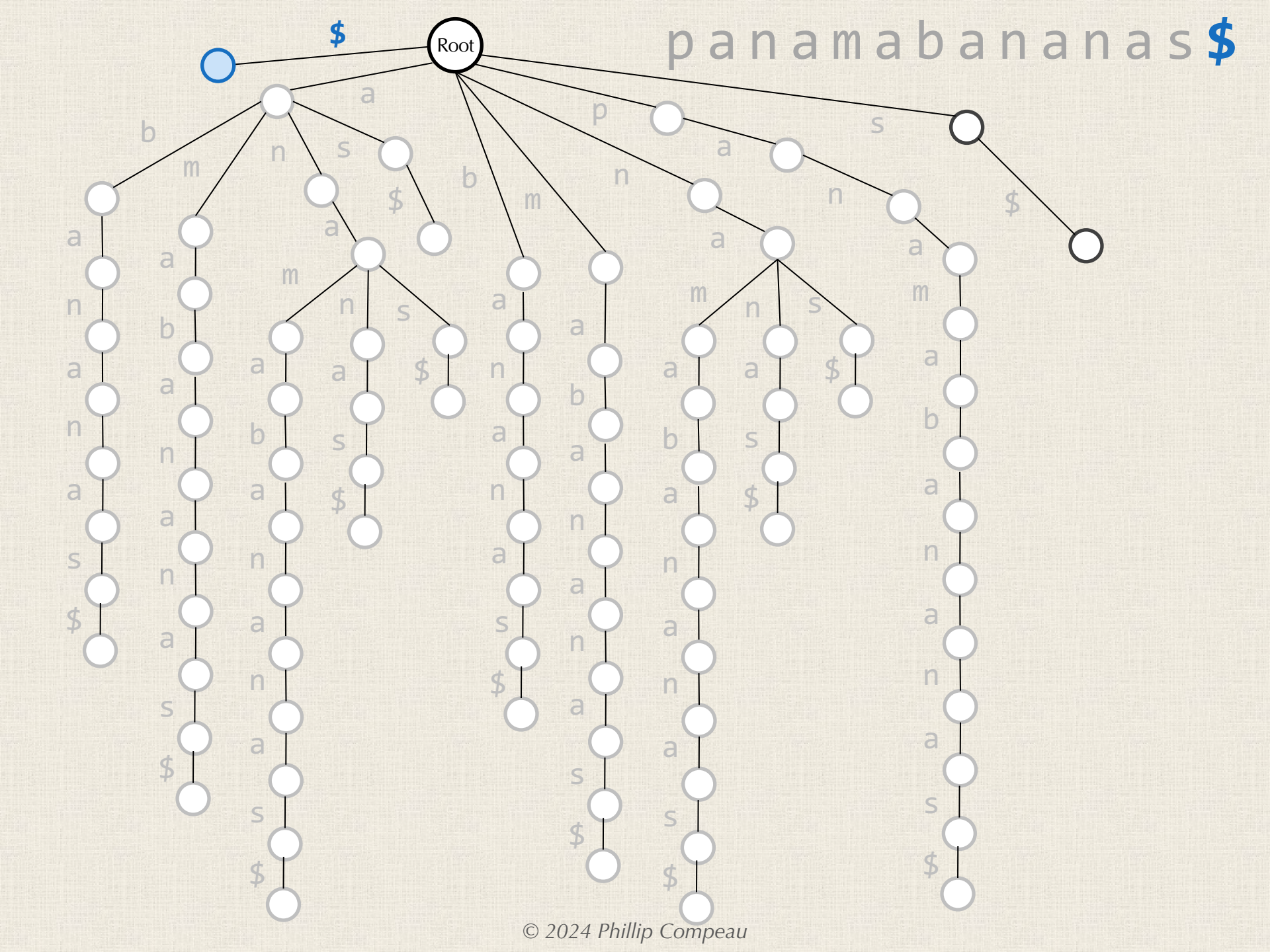


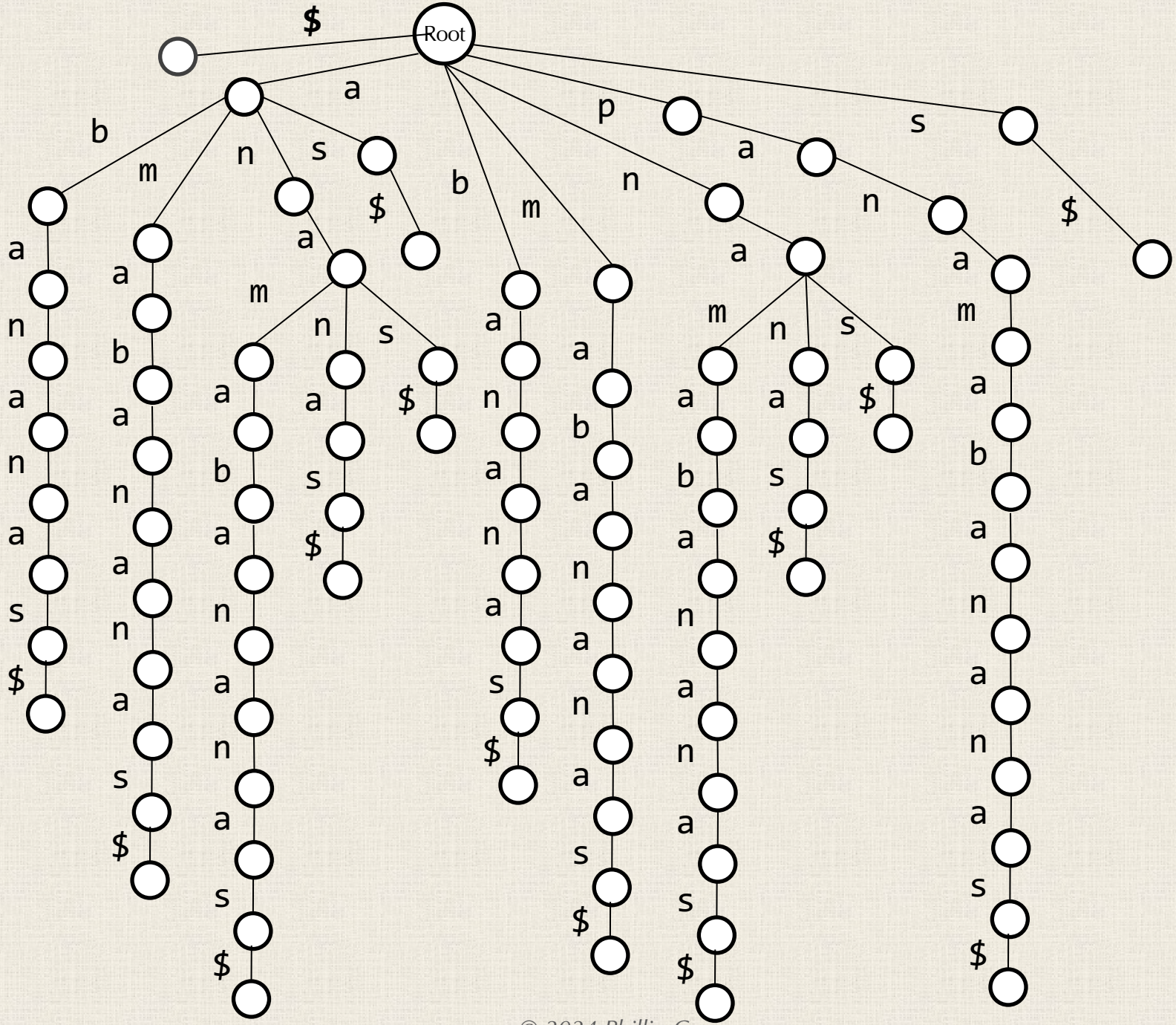


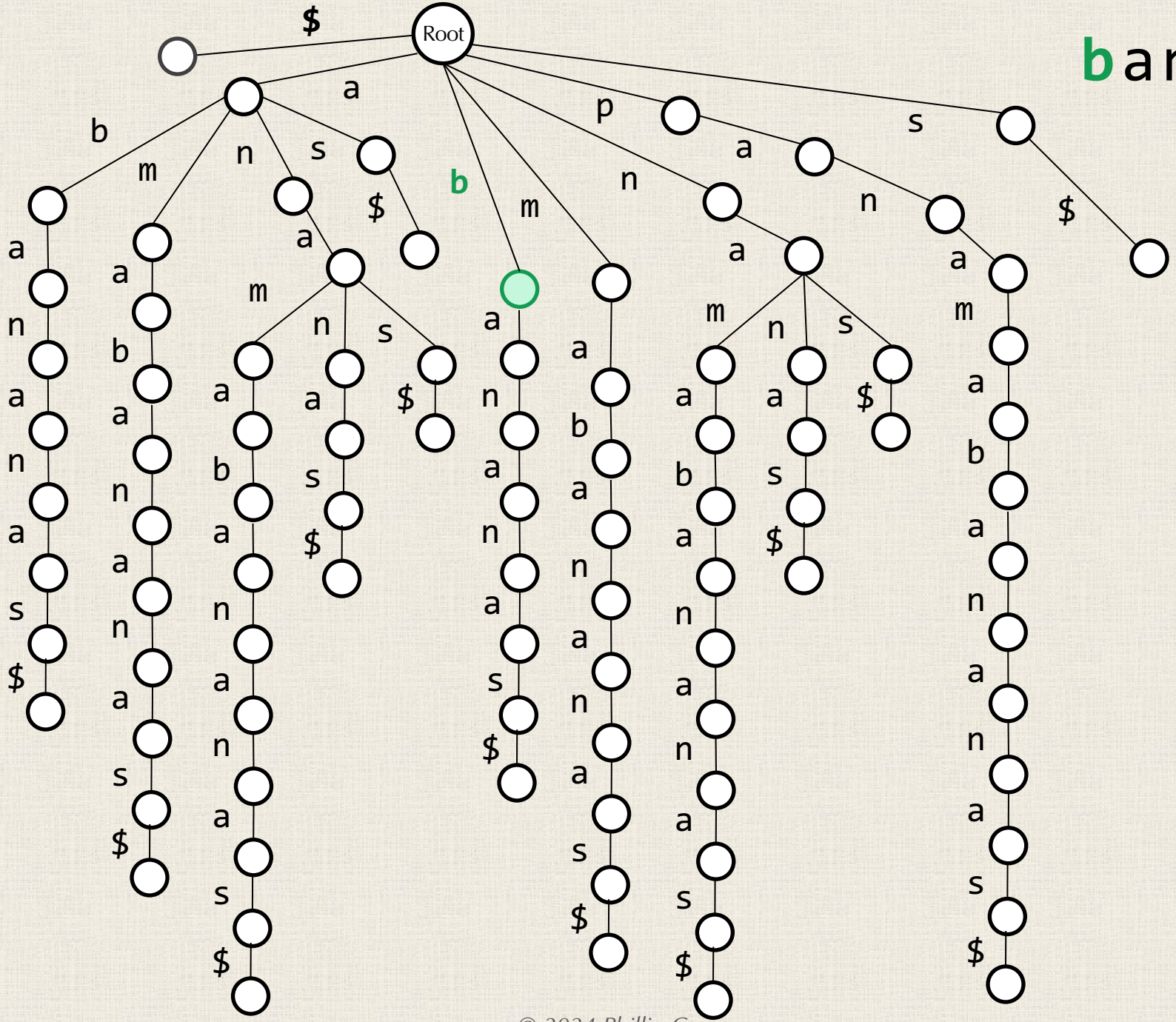


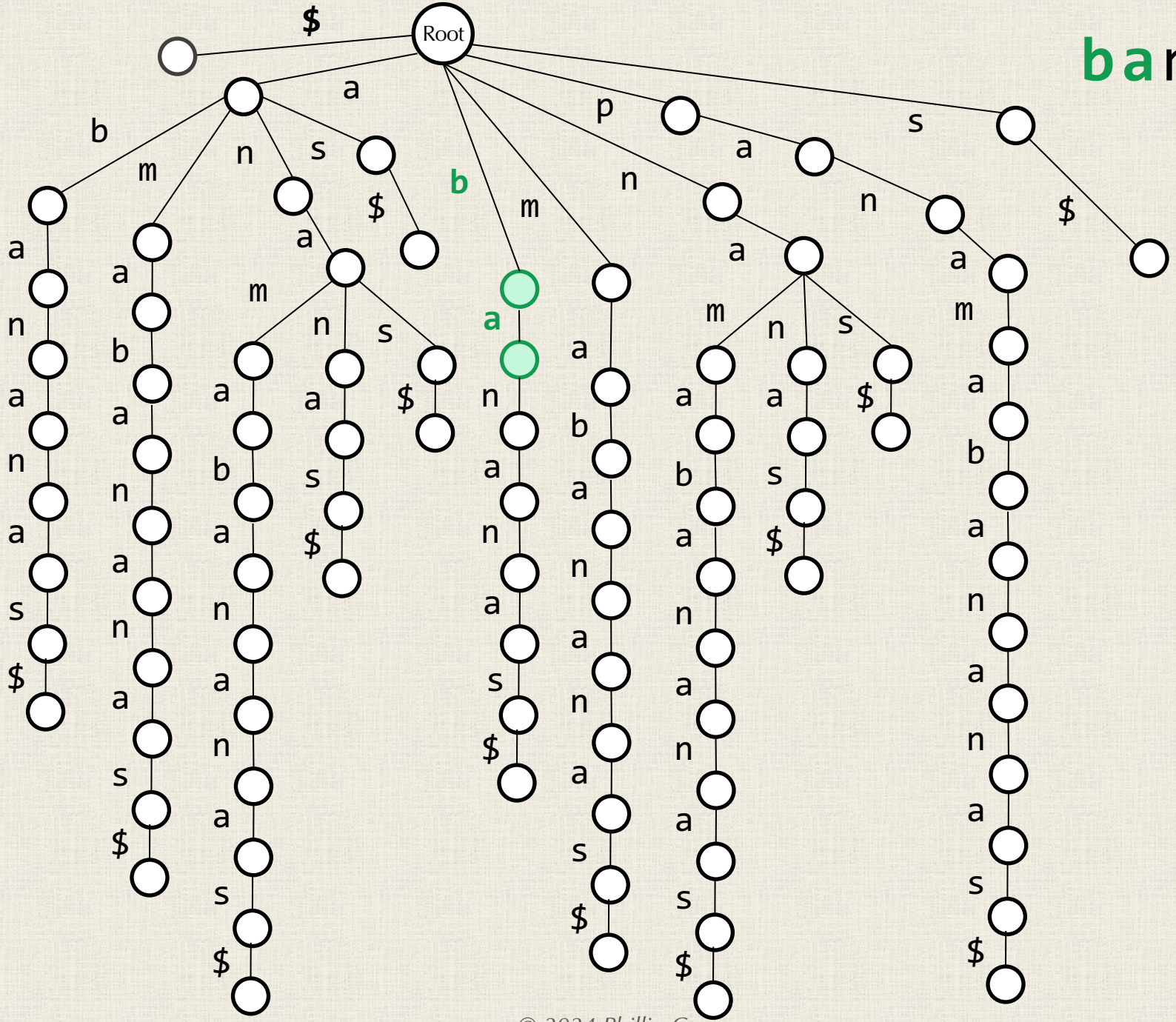


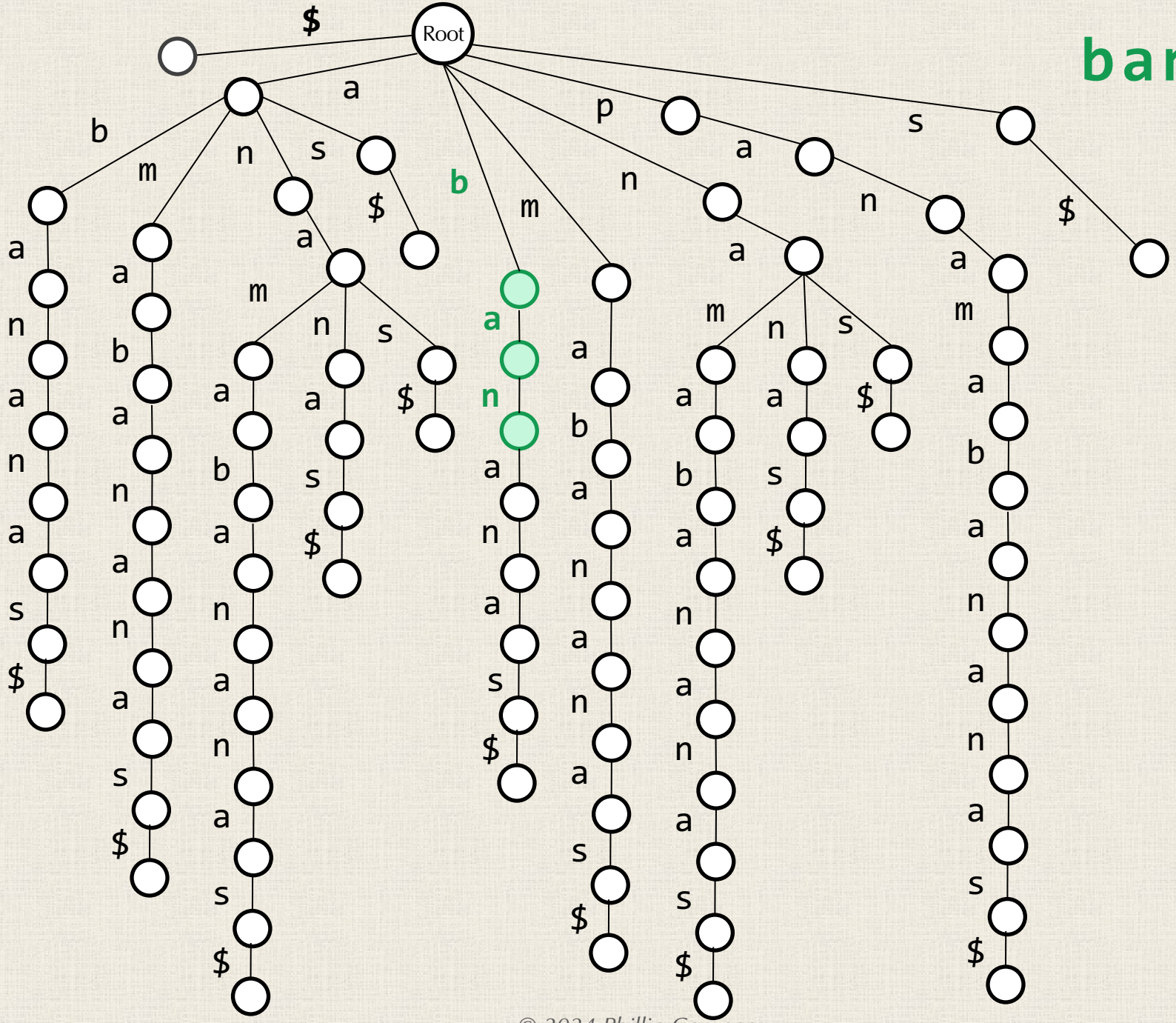


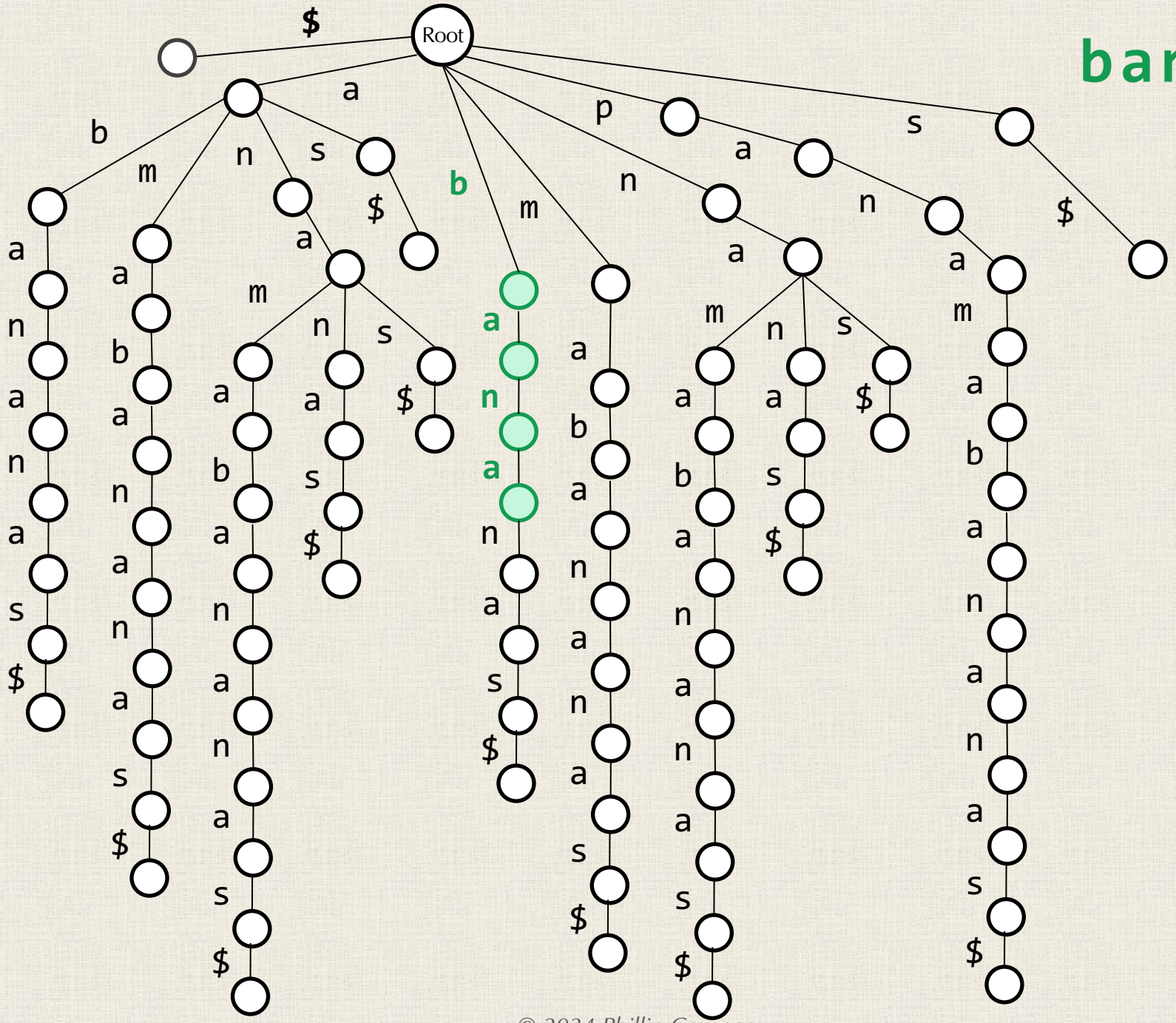


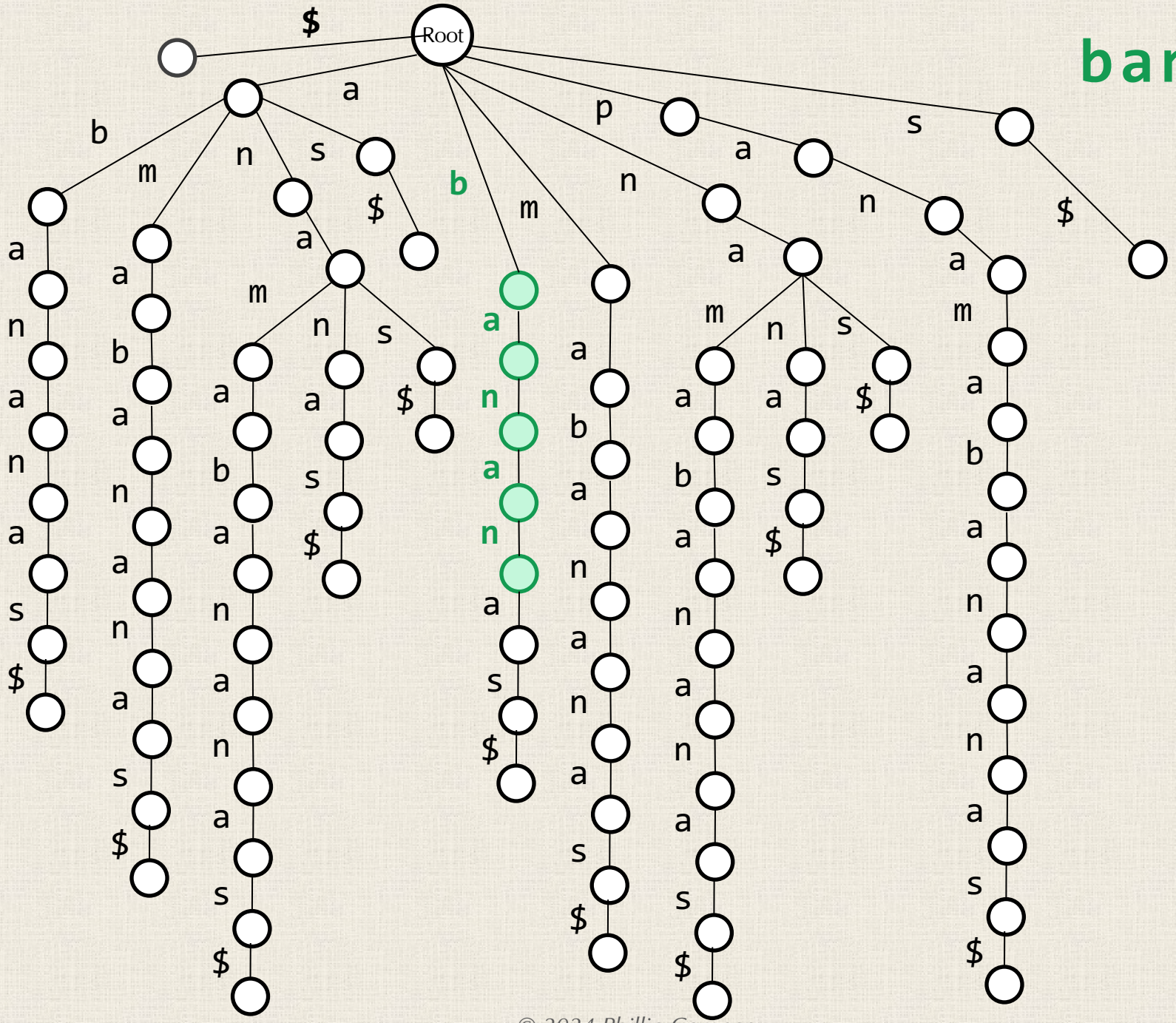


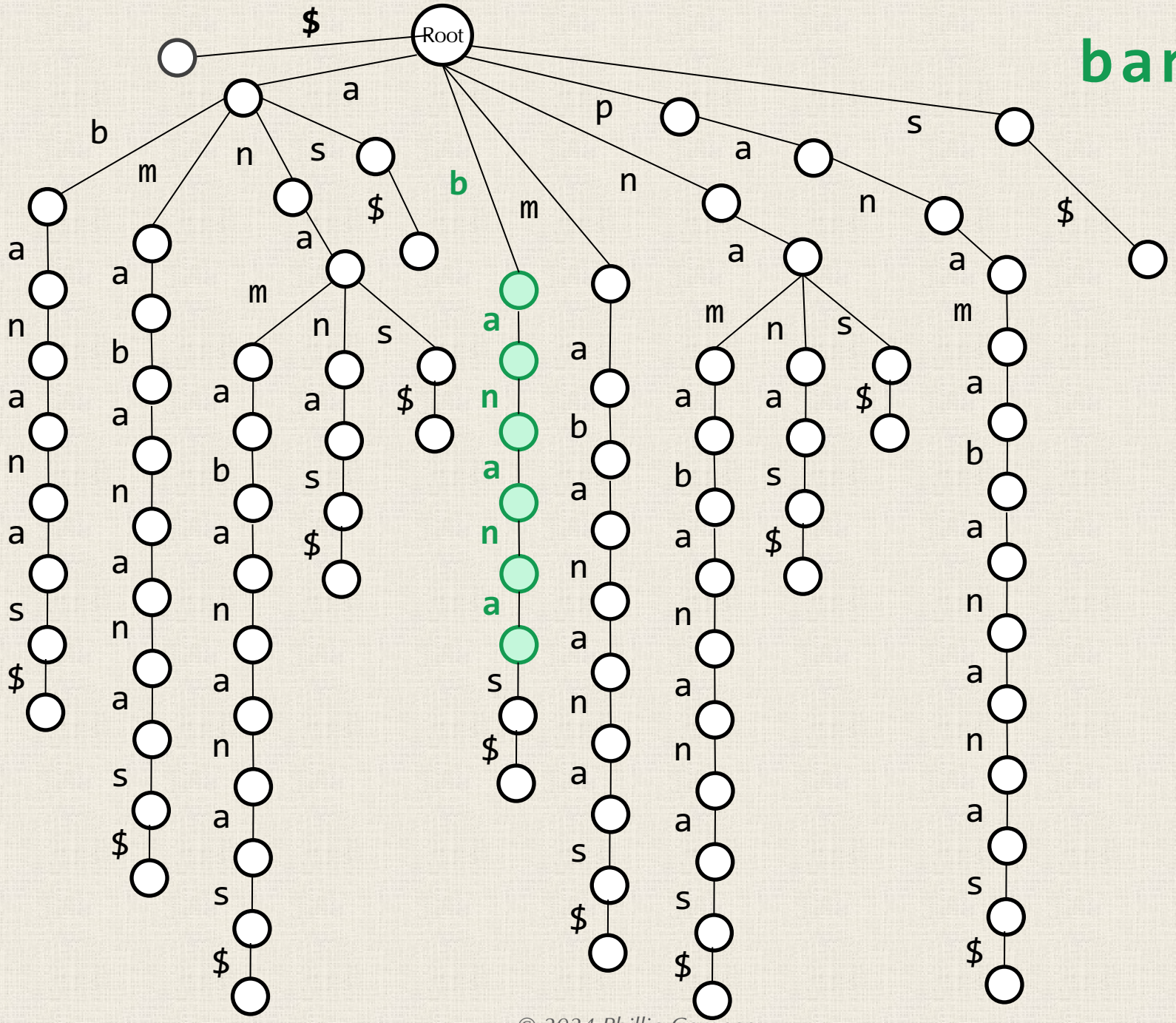


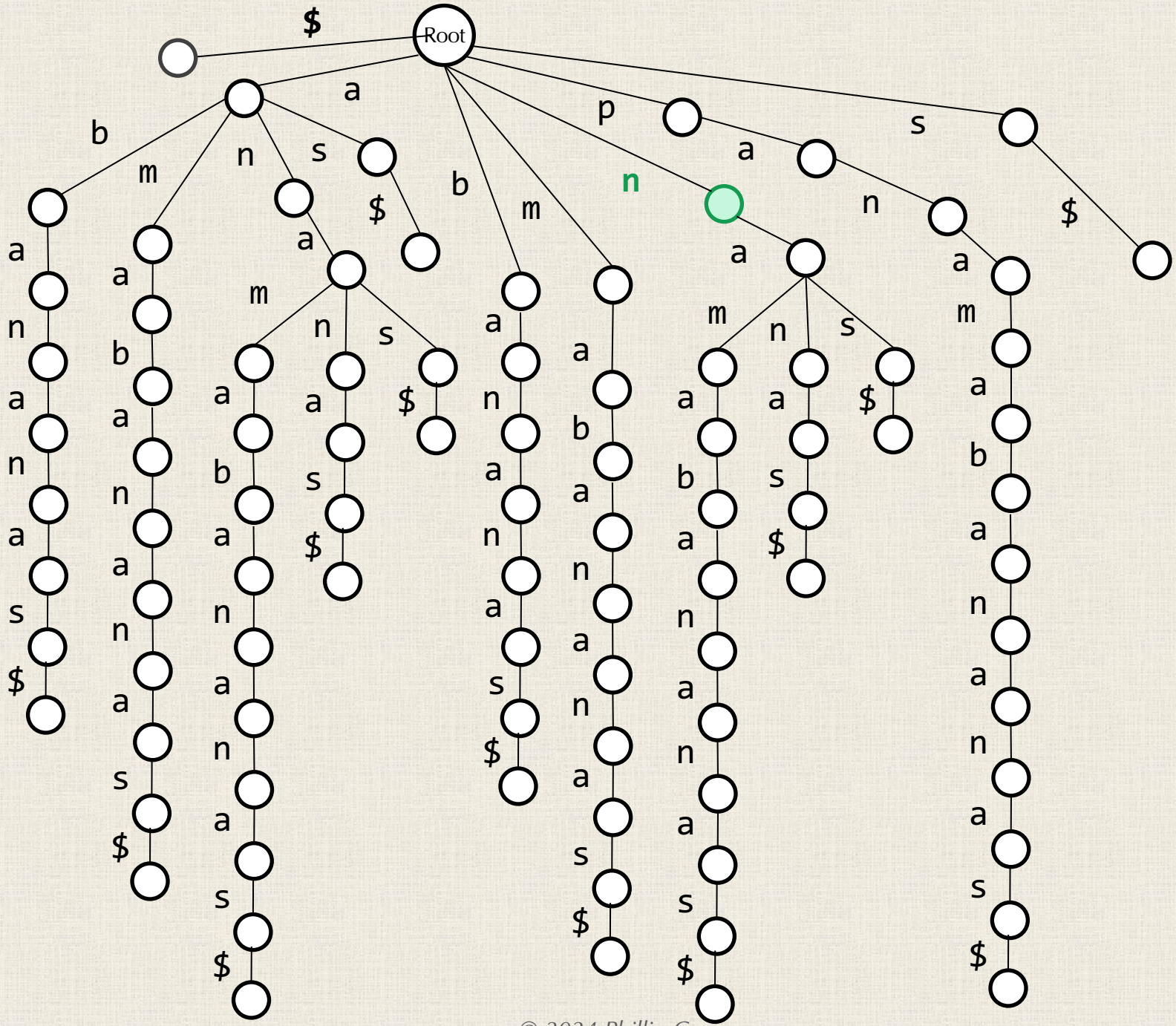


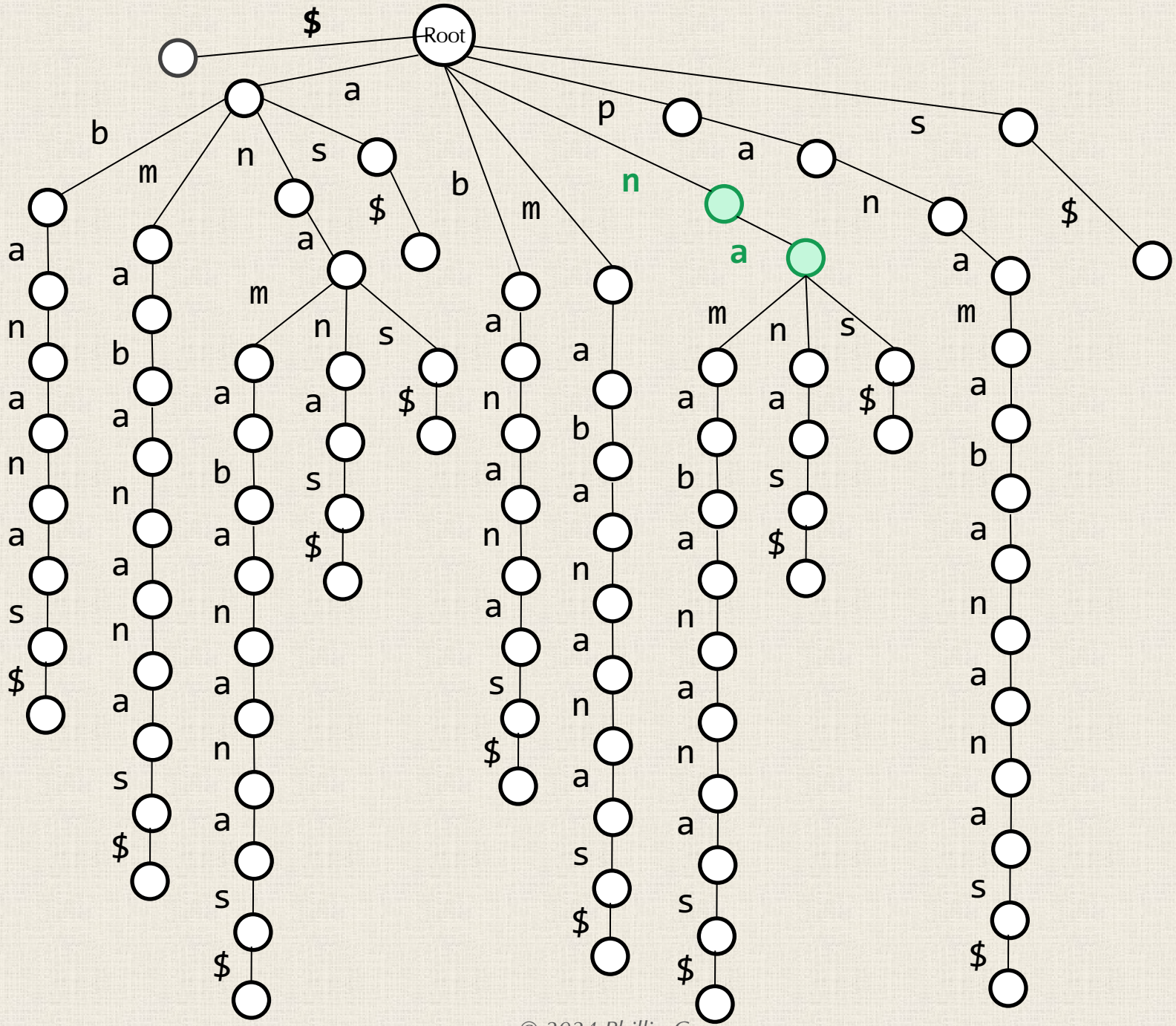




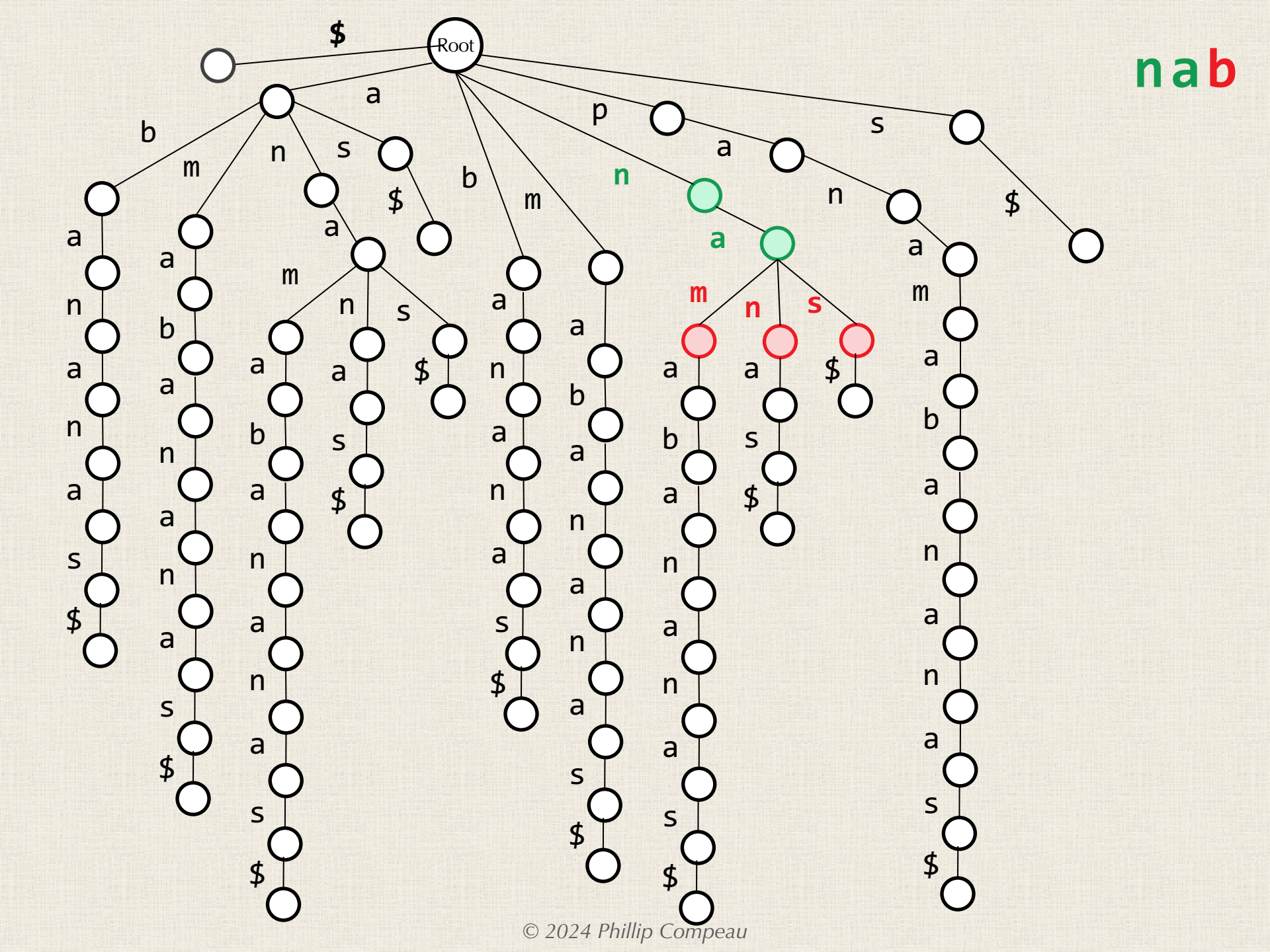

banana
Root
$
a
b
m
n
s
p
banana$
anana$
nana$
ana$
na$
a$
$
© 2024 Phillip Compeau

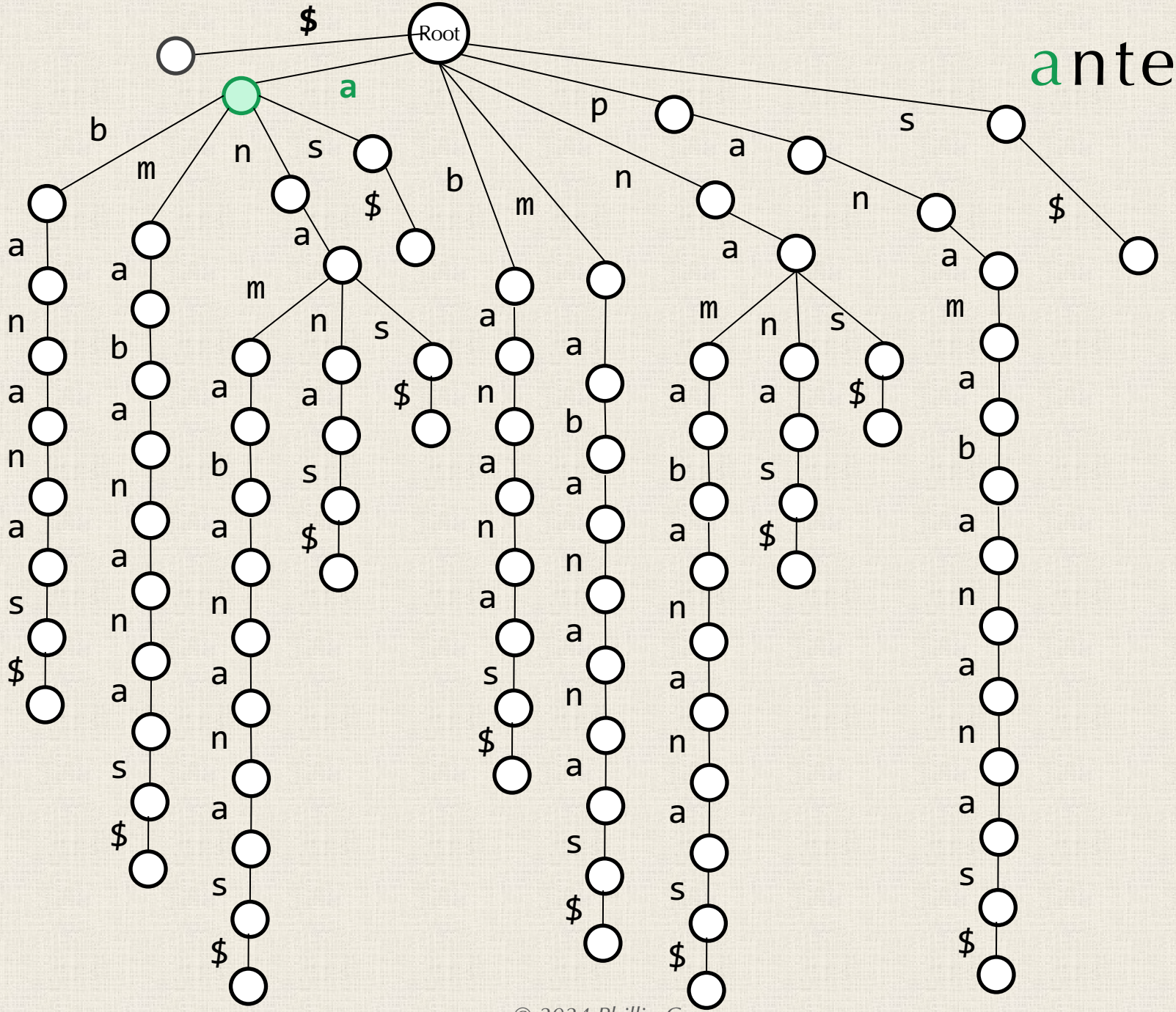


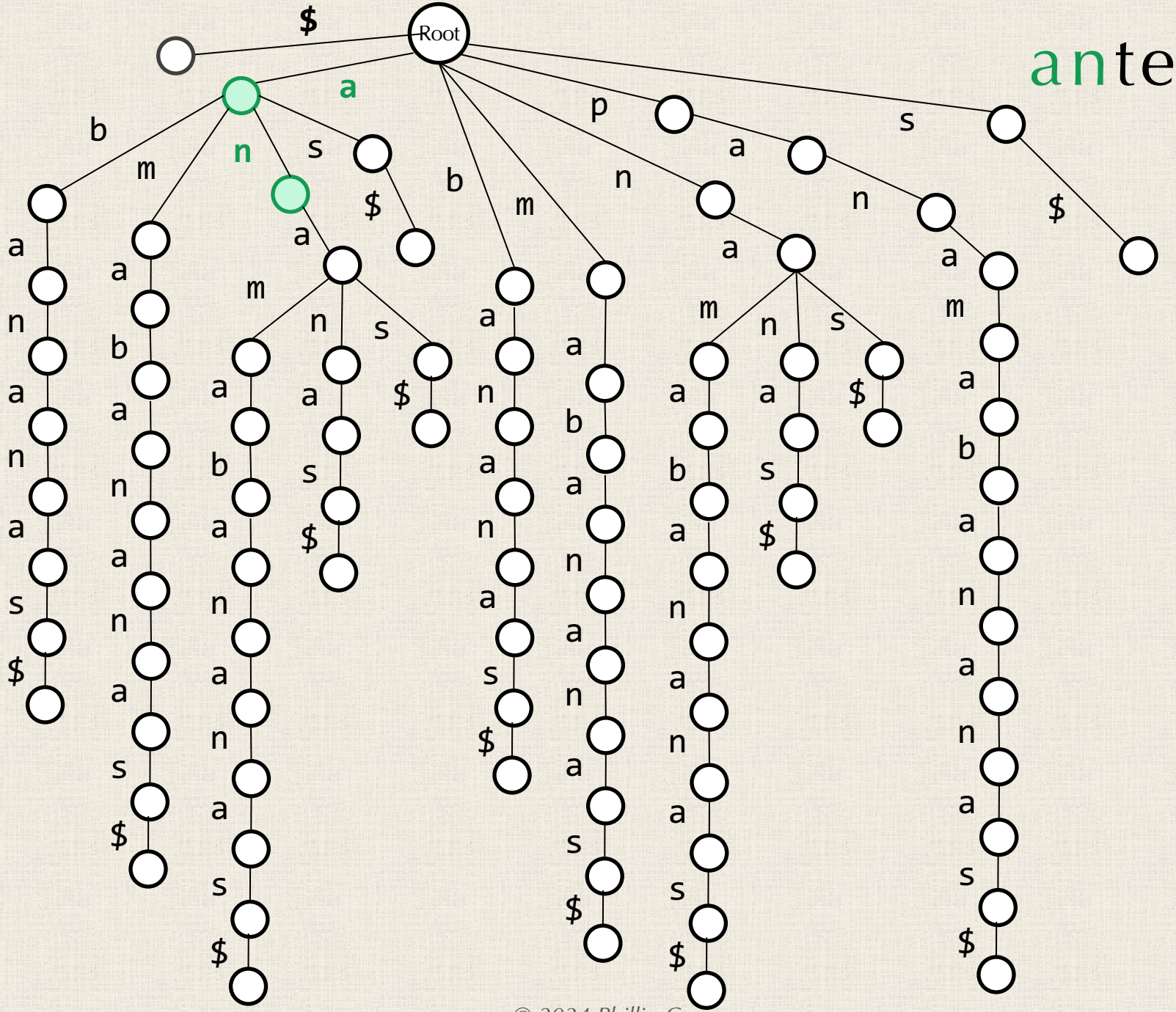


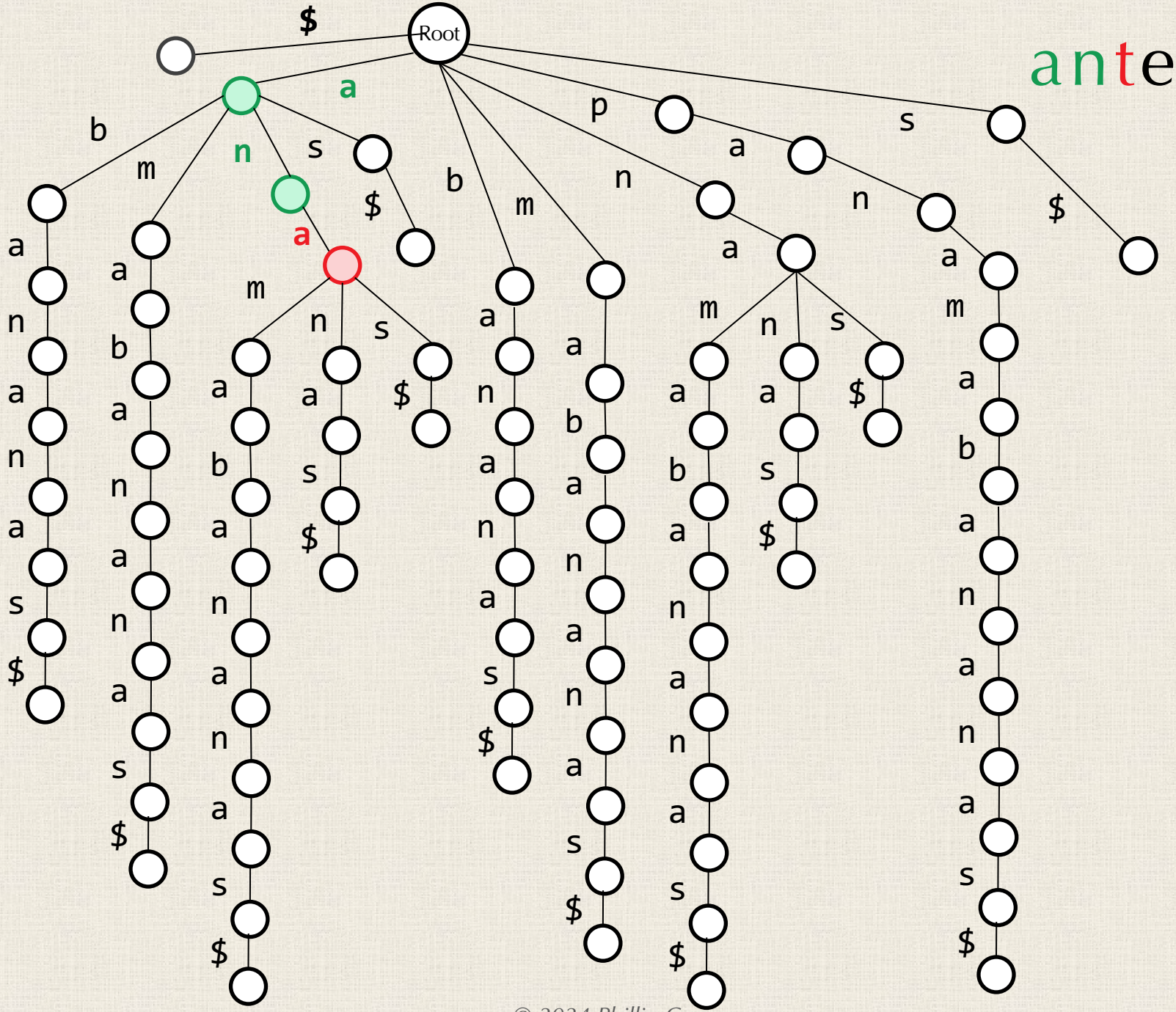


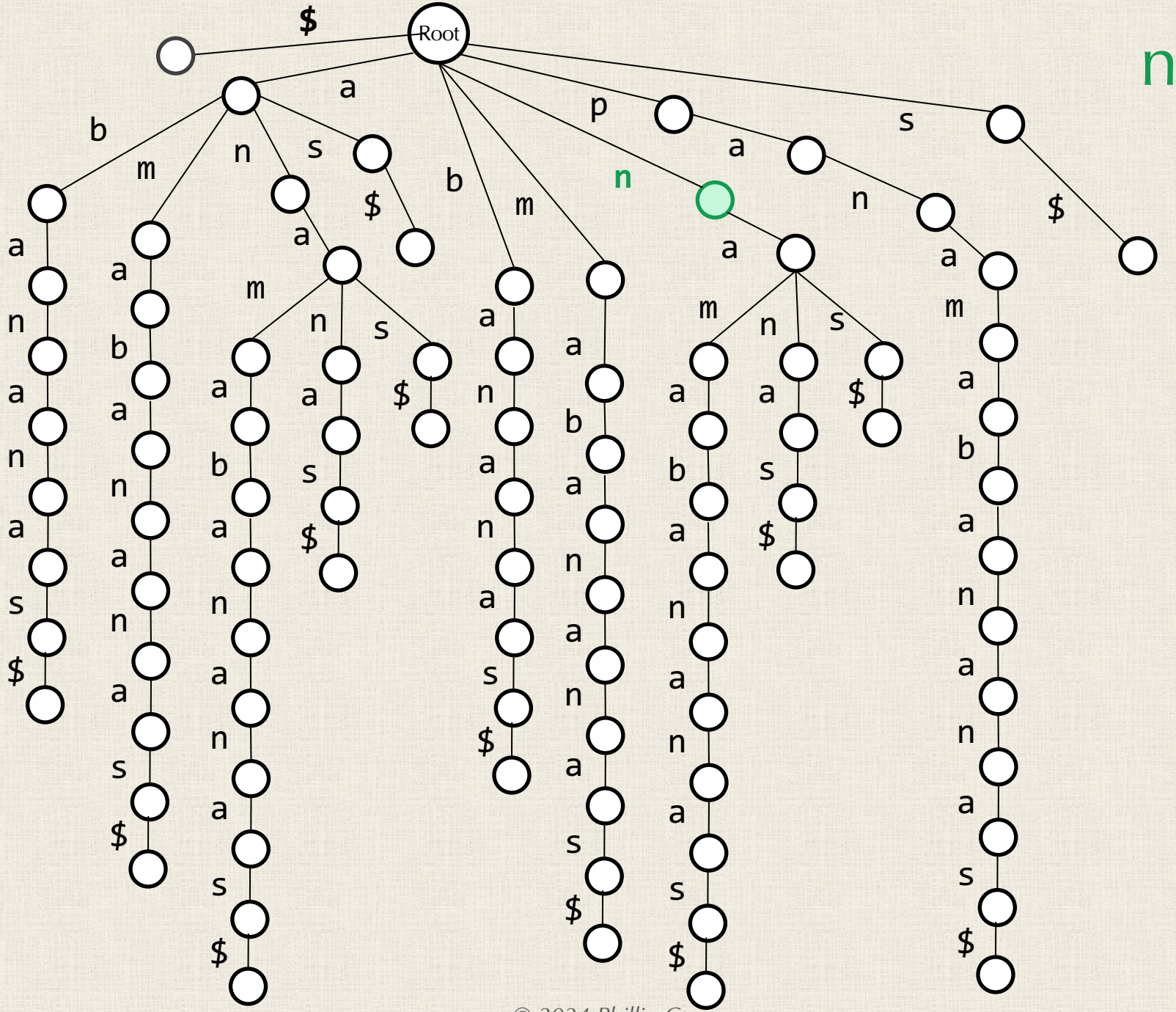
nab



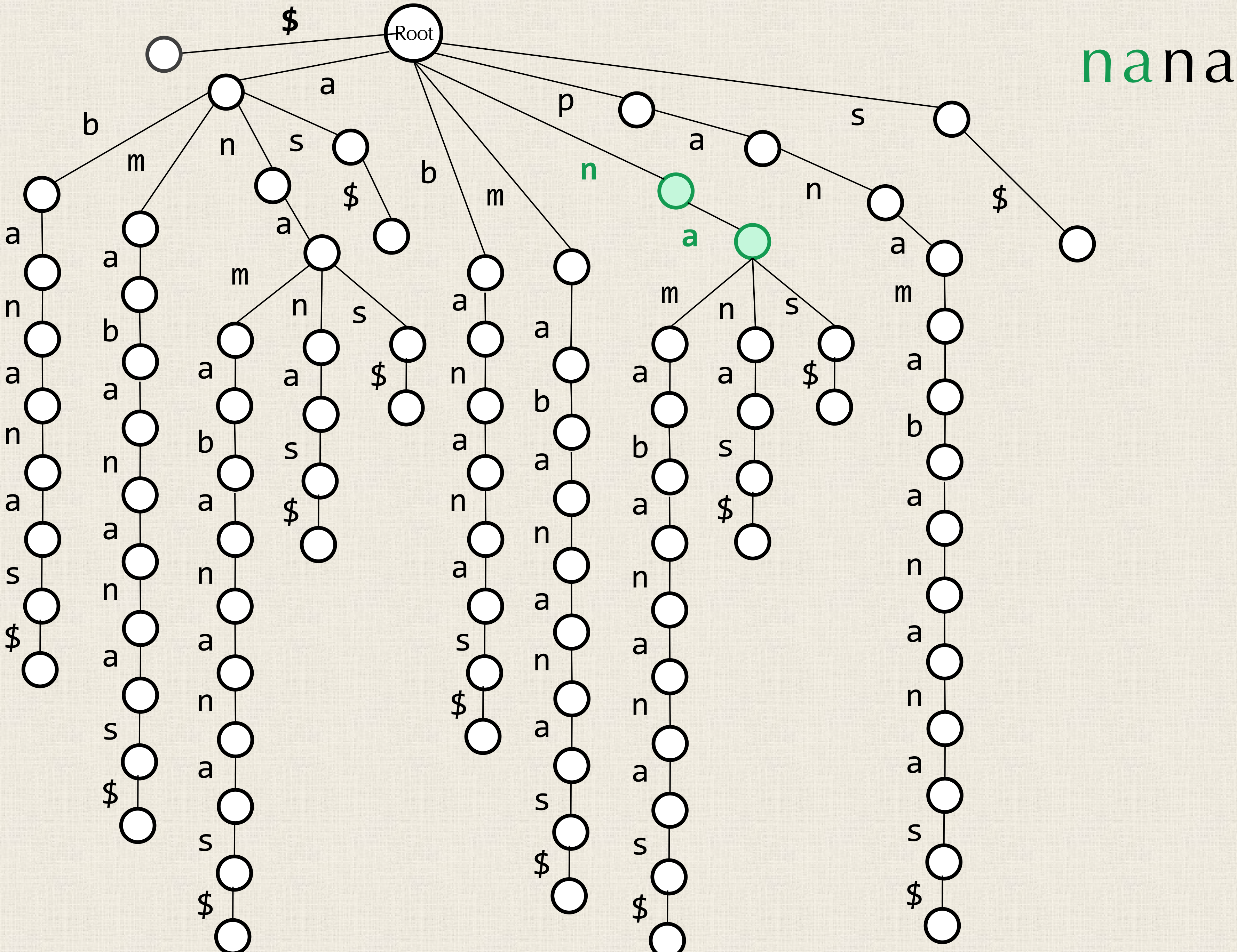


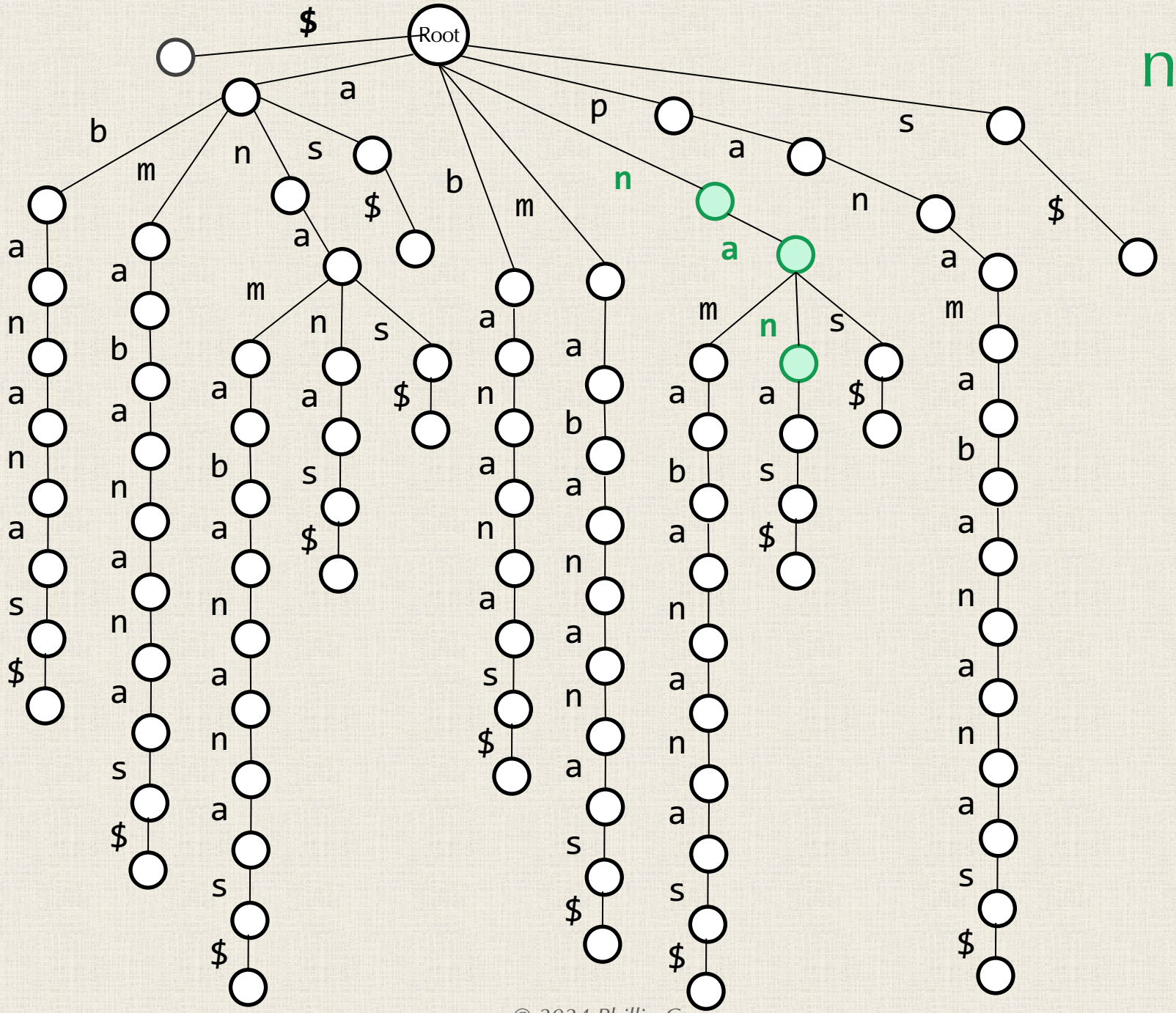


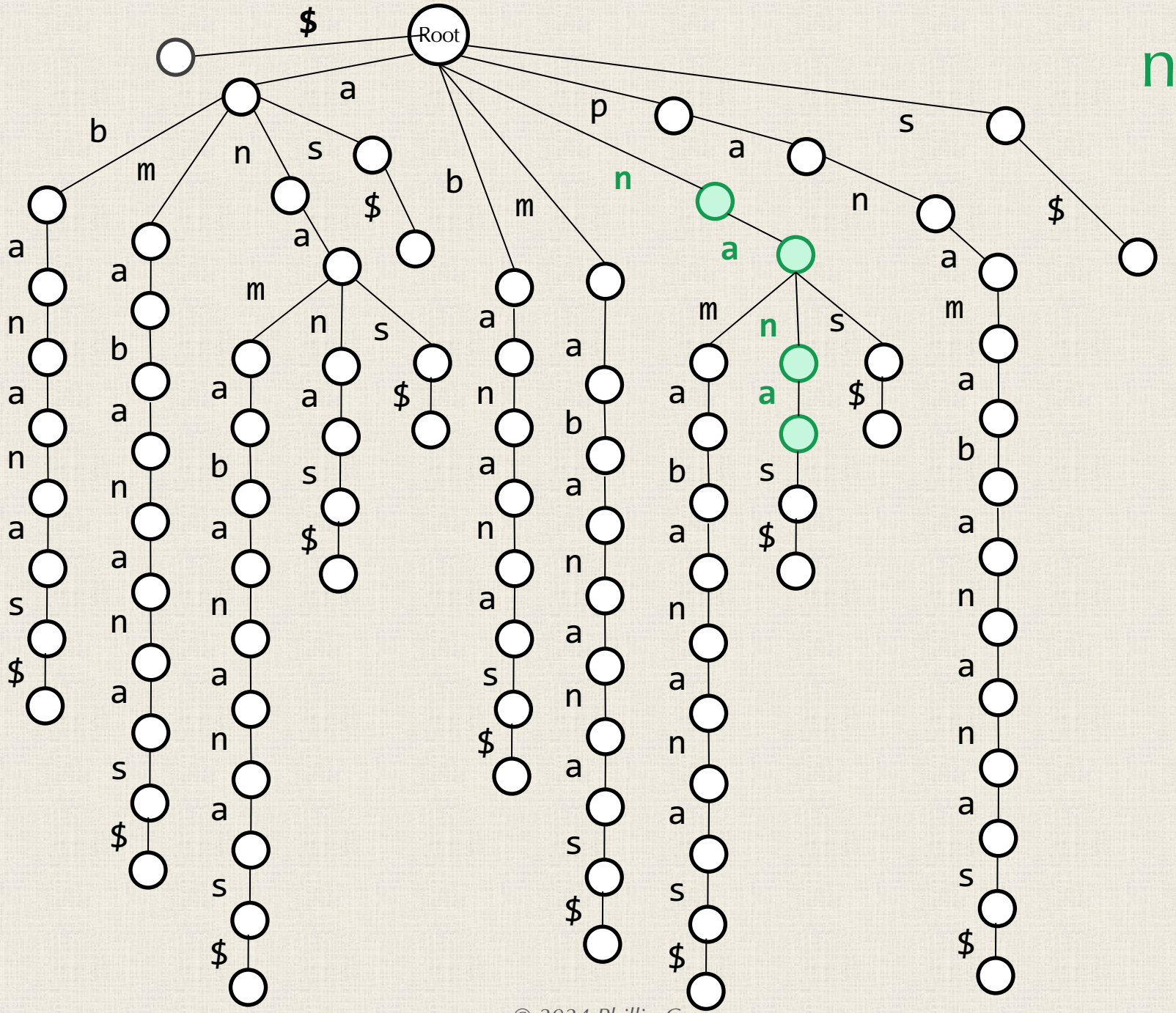


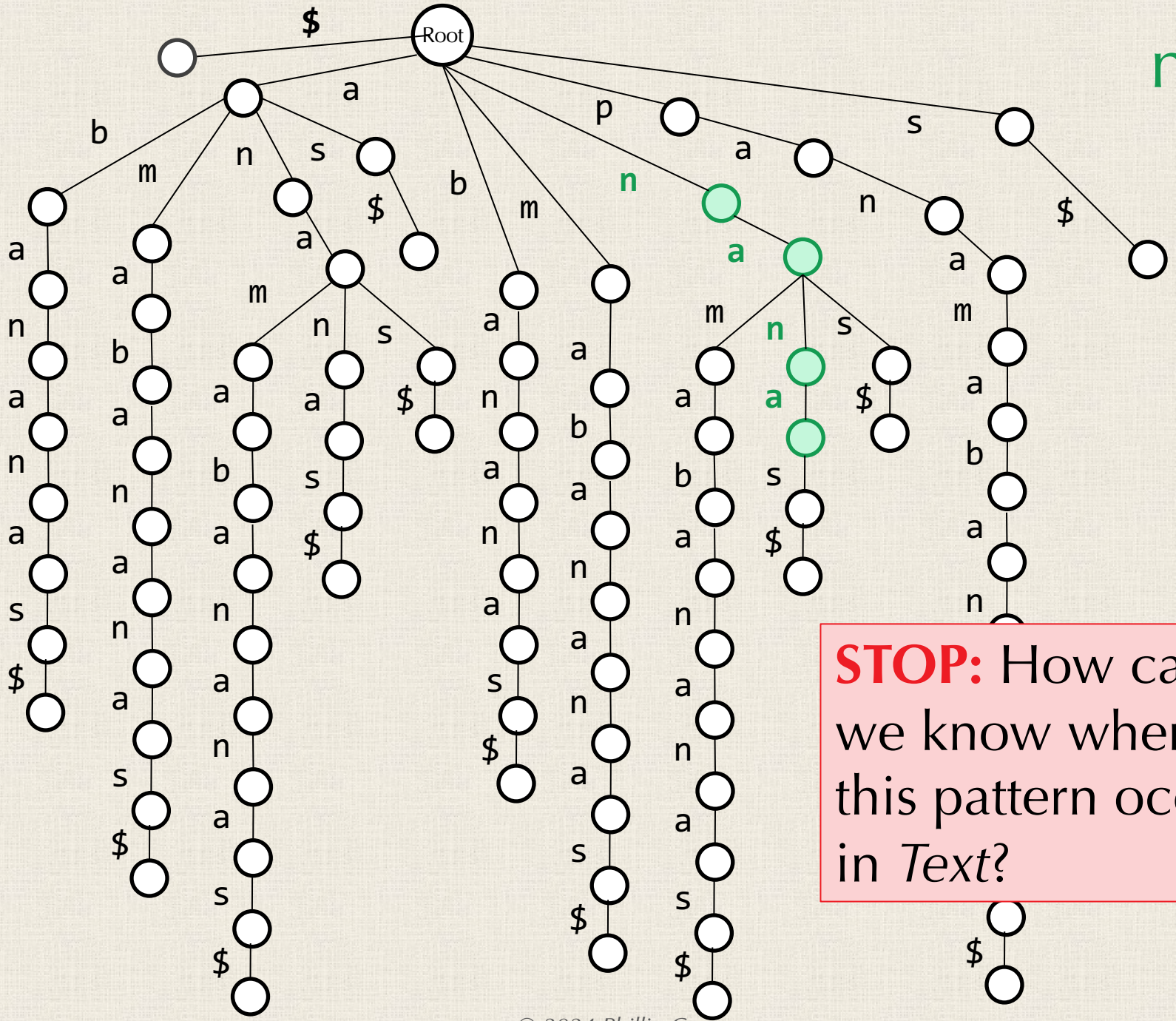


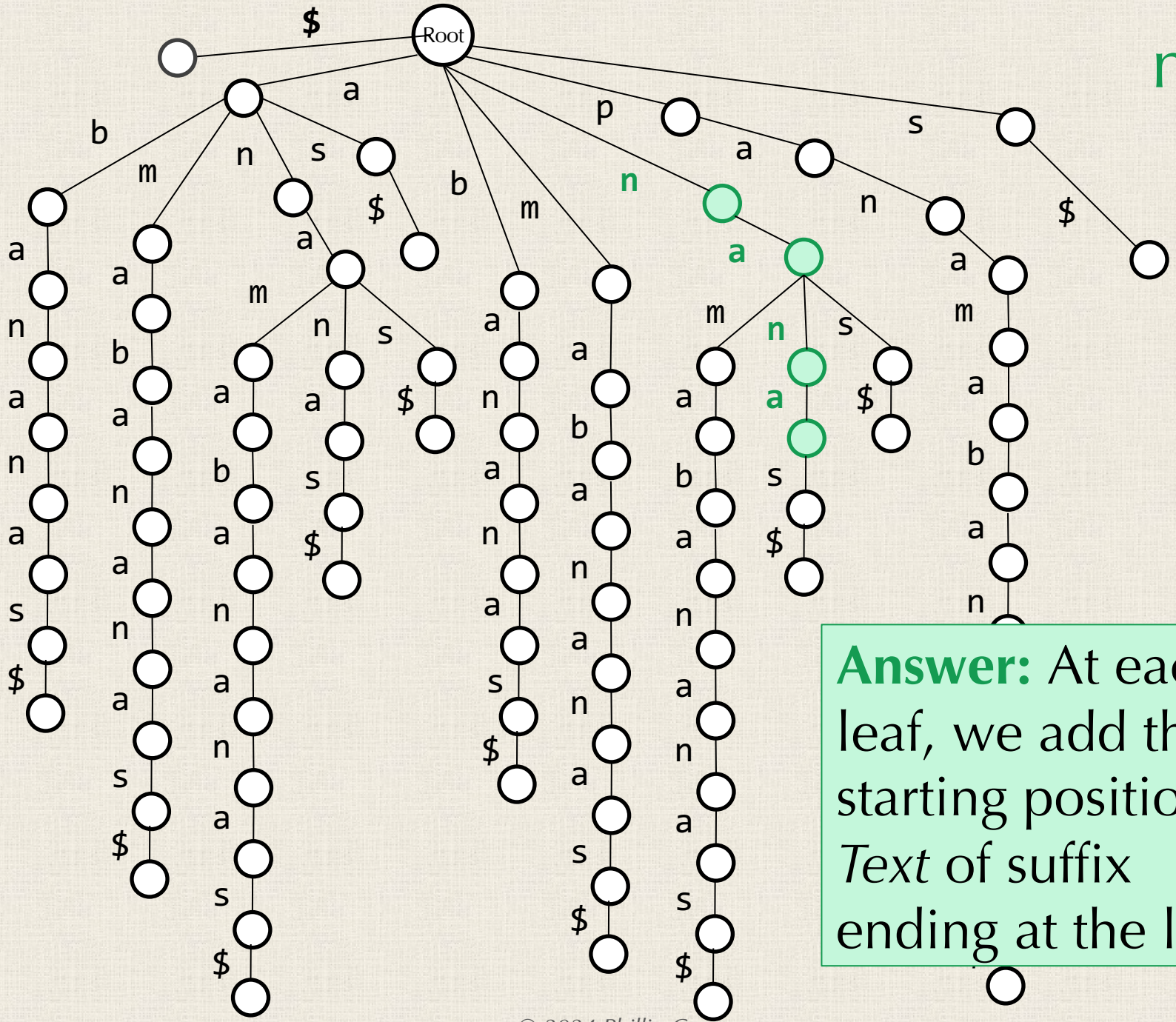
nana

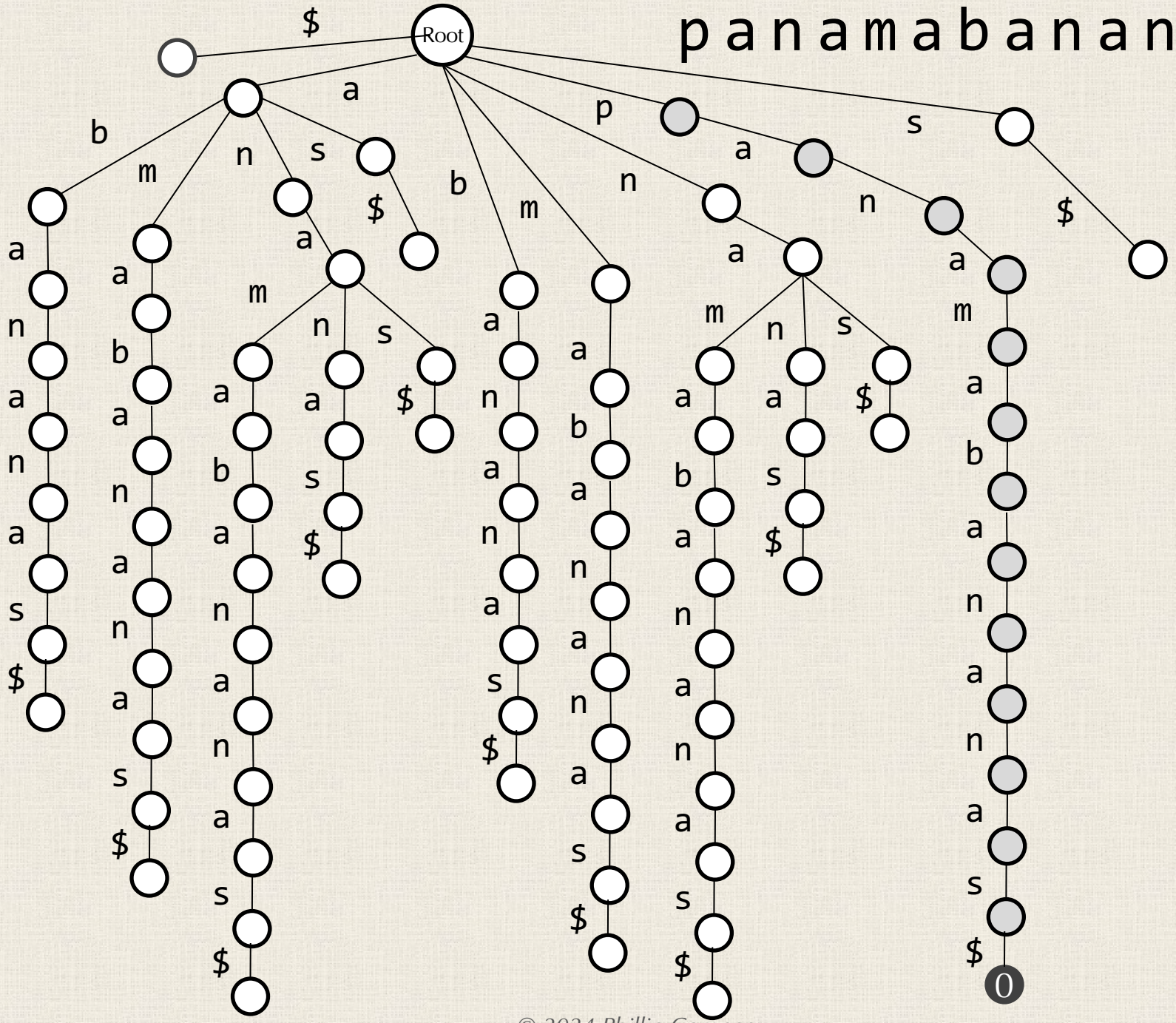


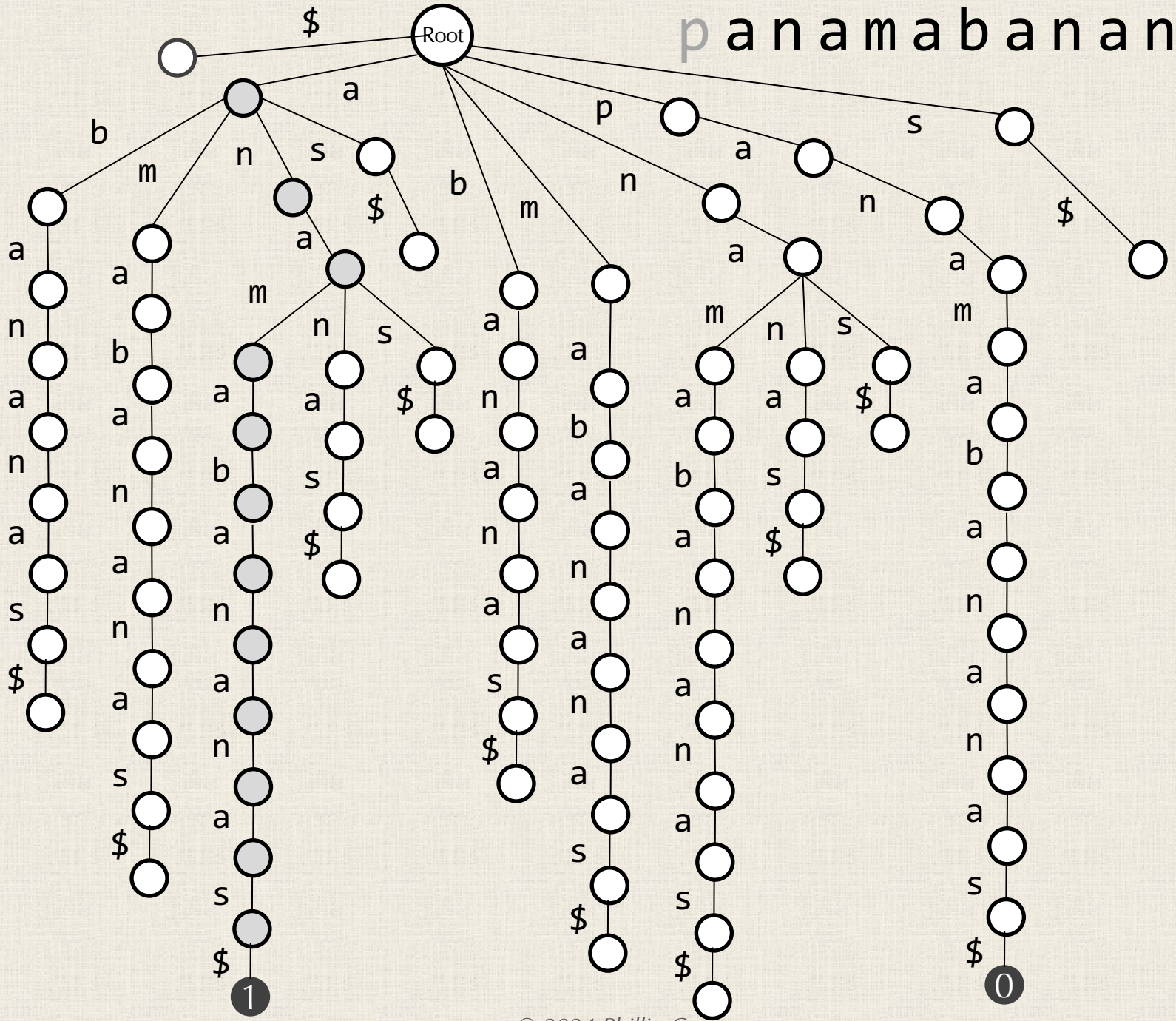


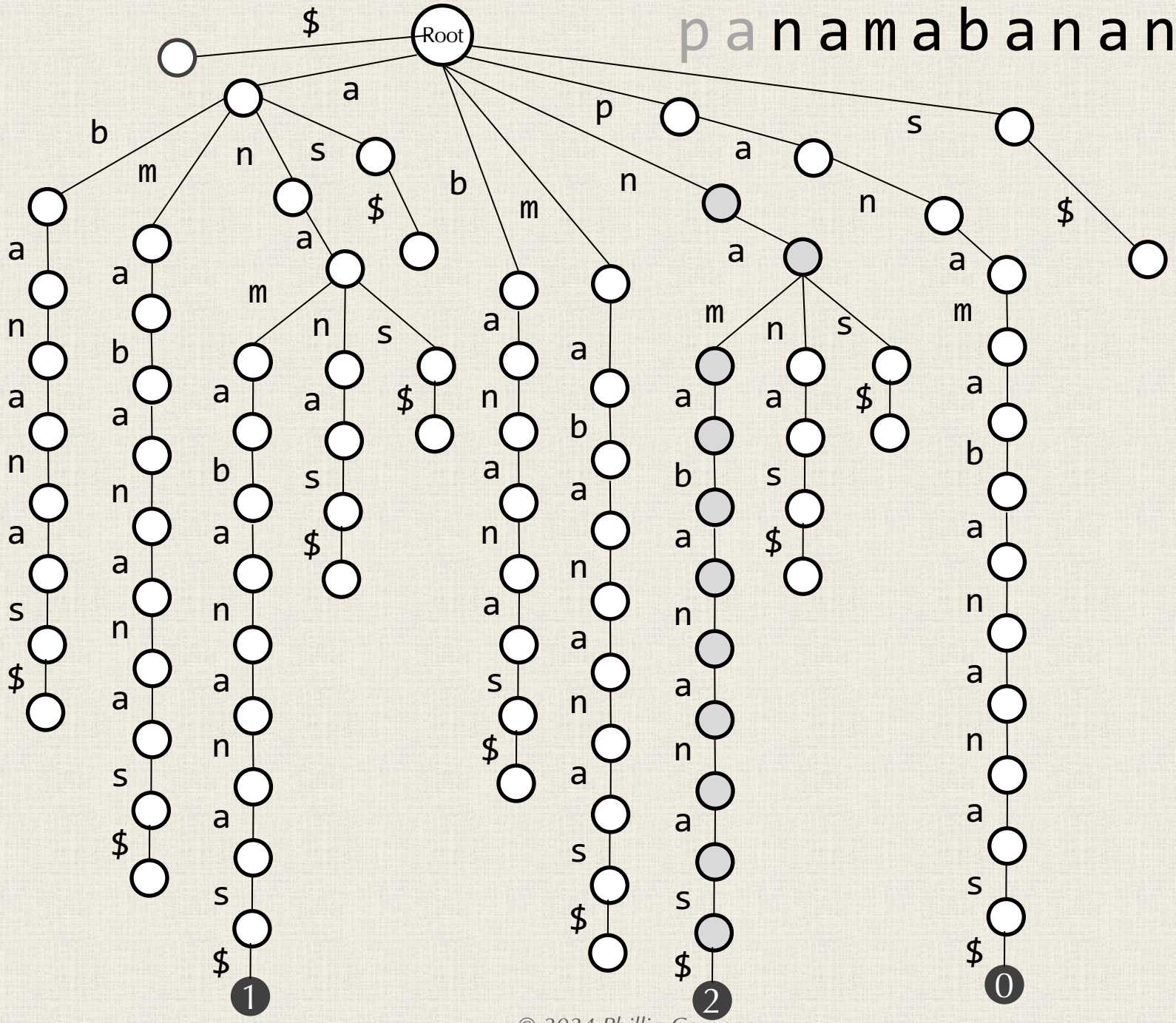


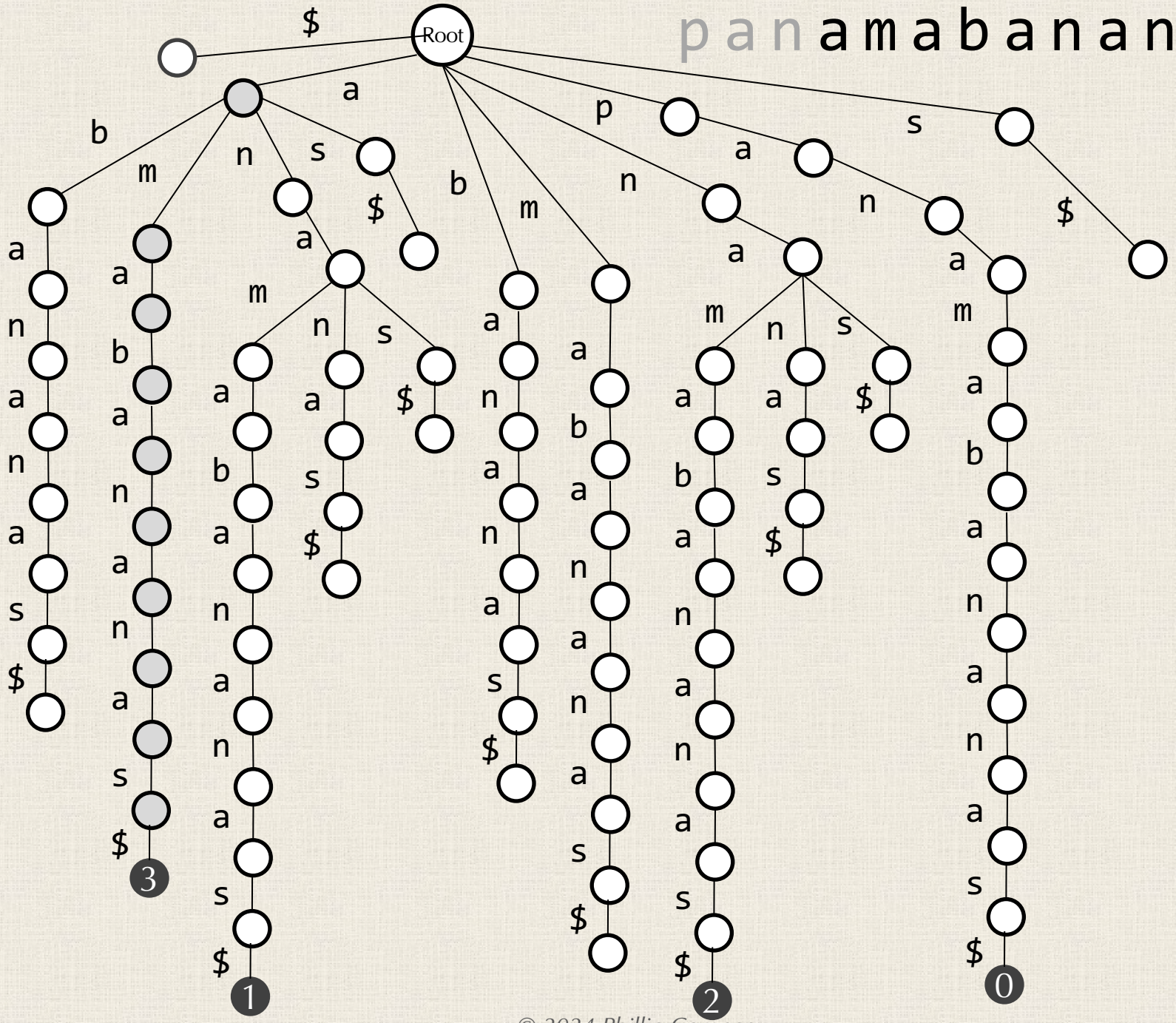


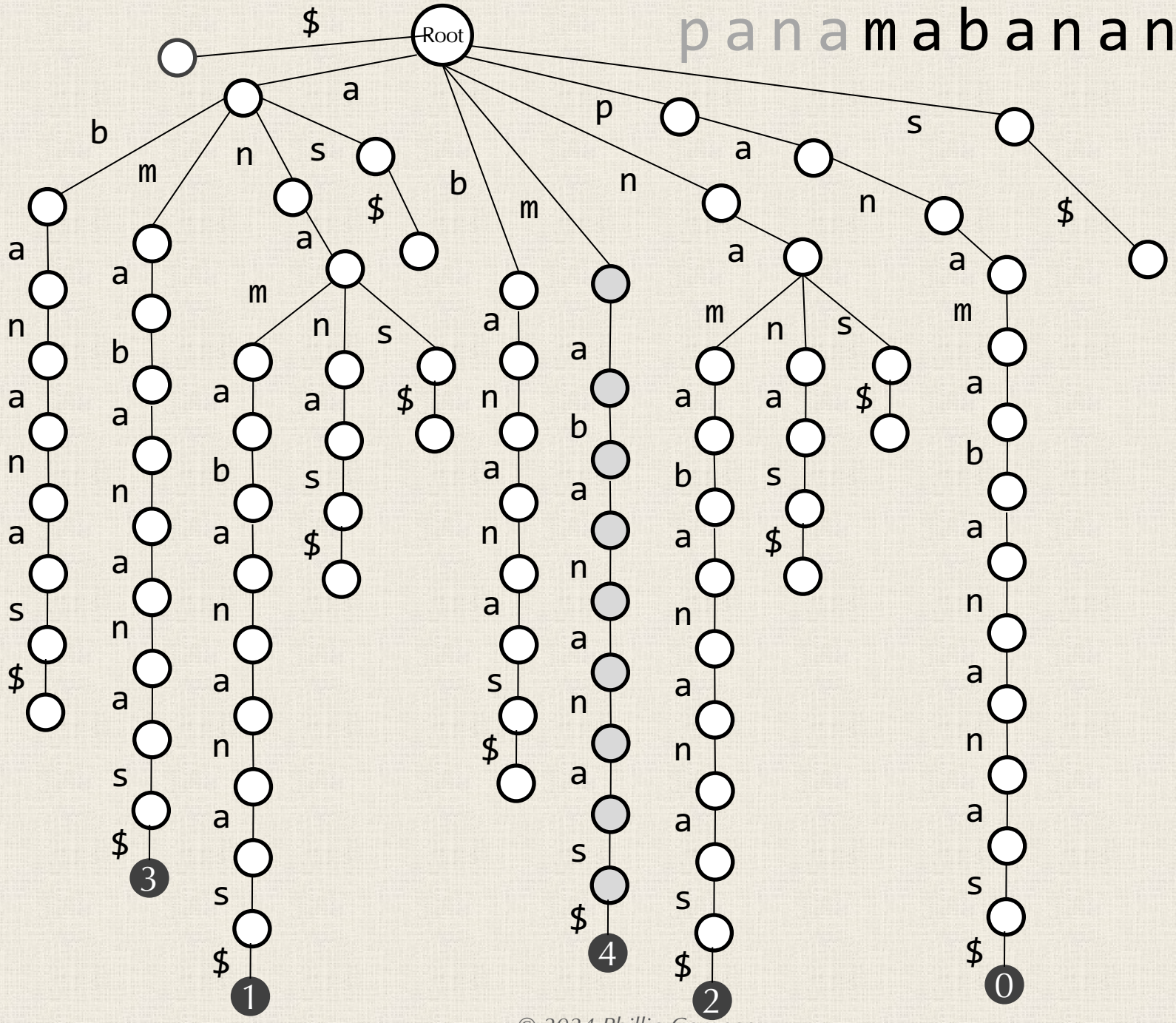


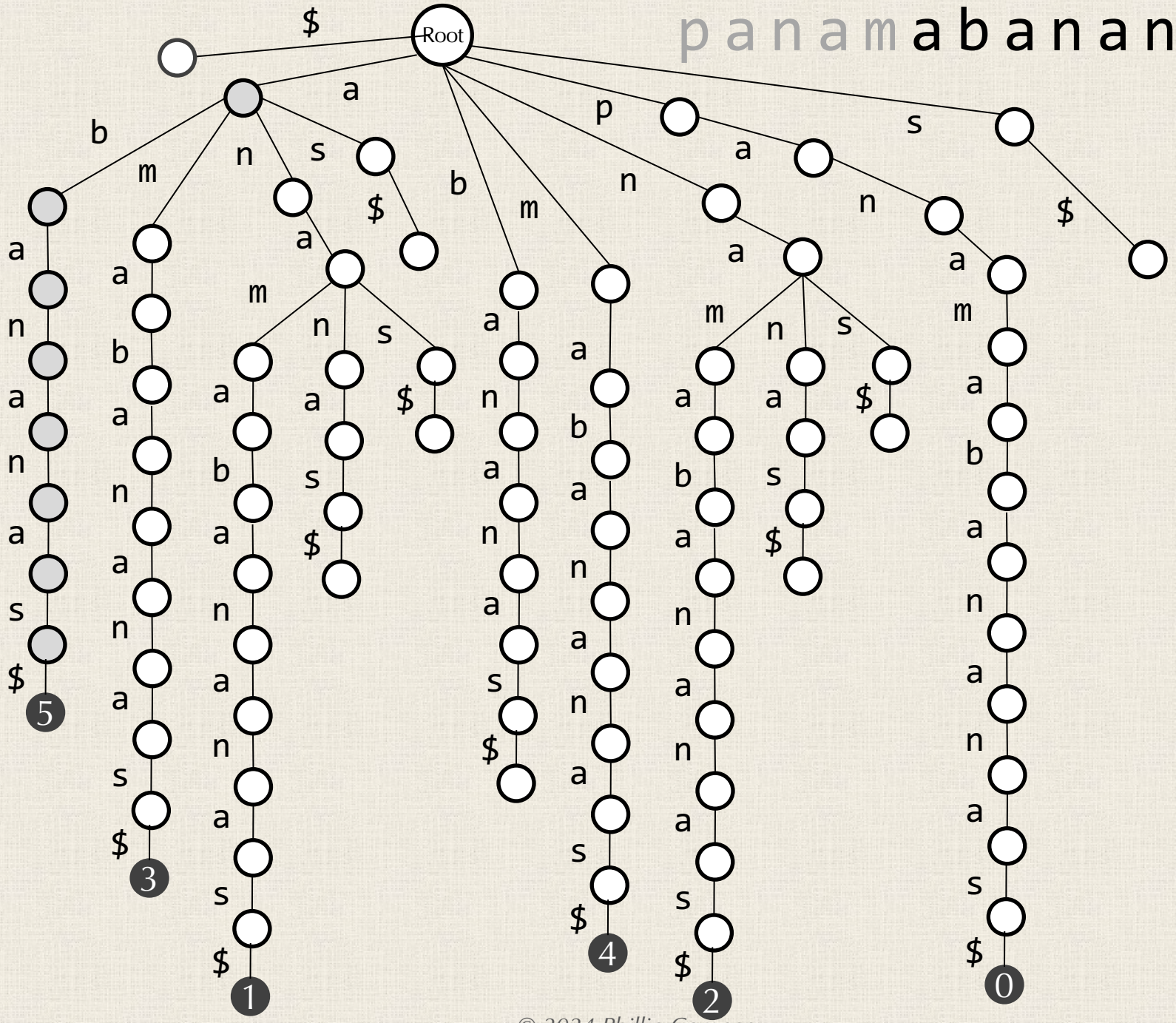


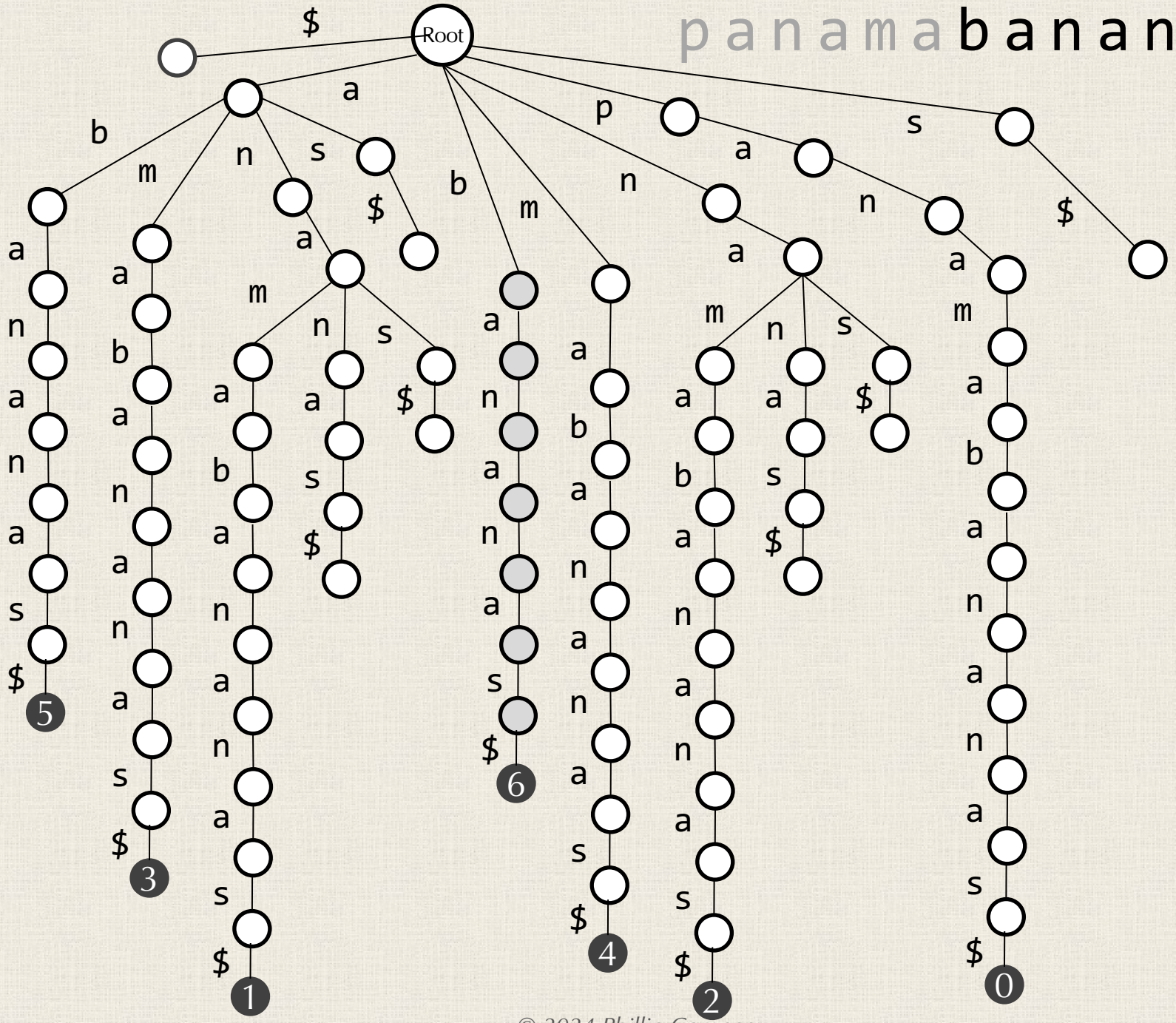


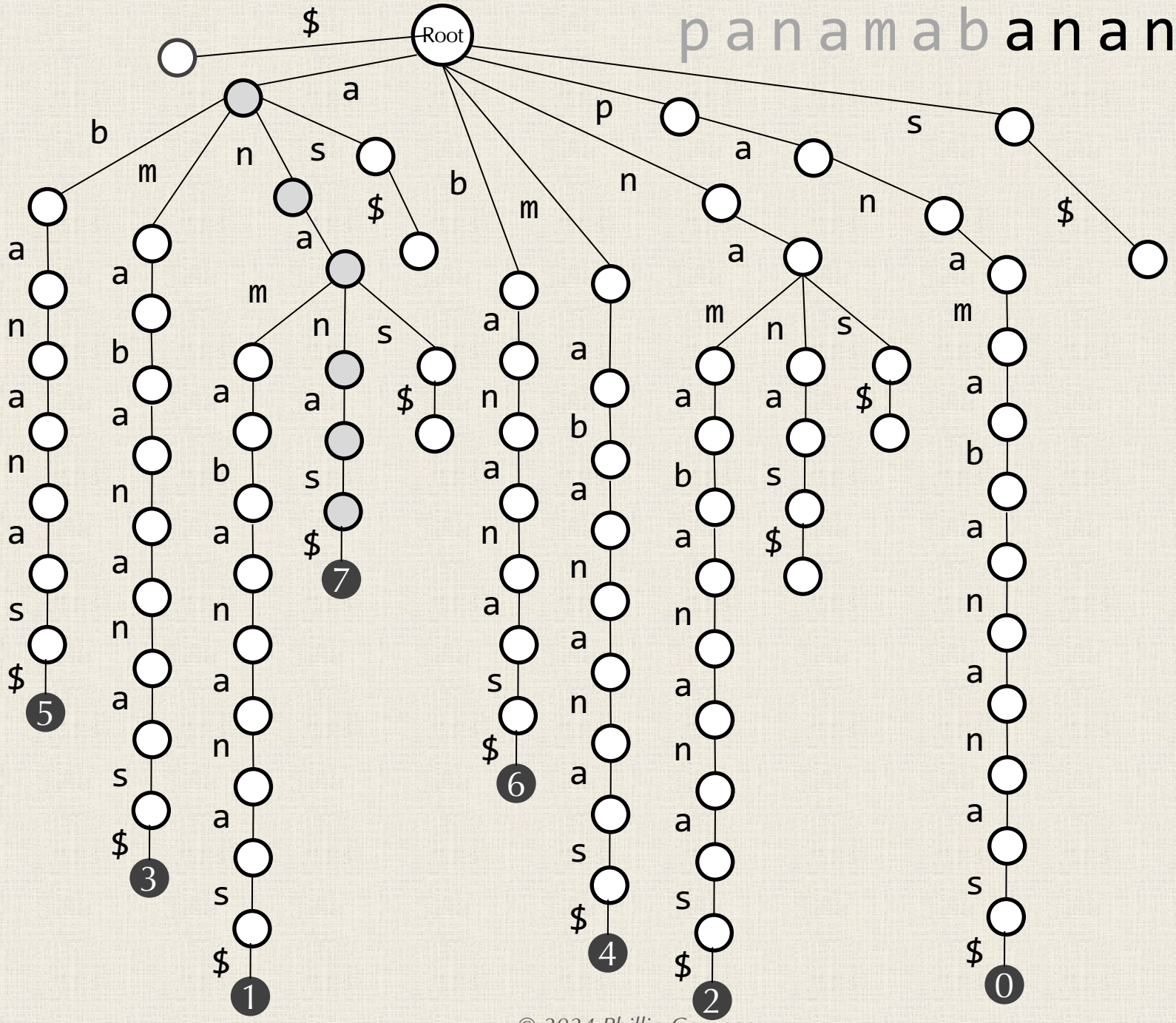


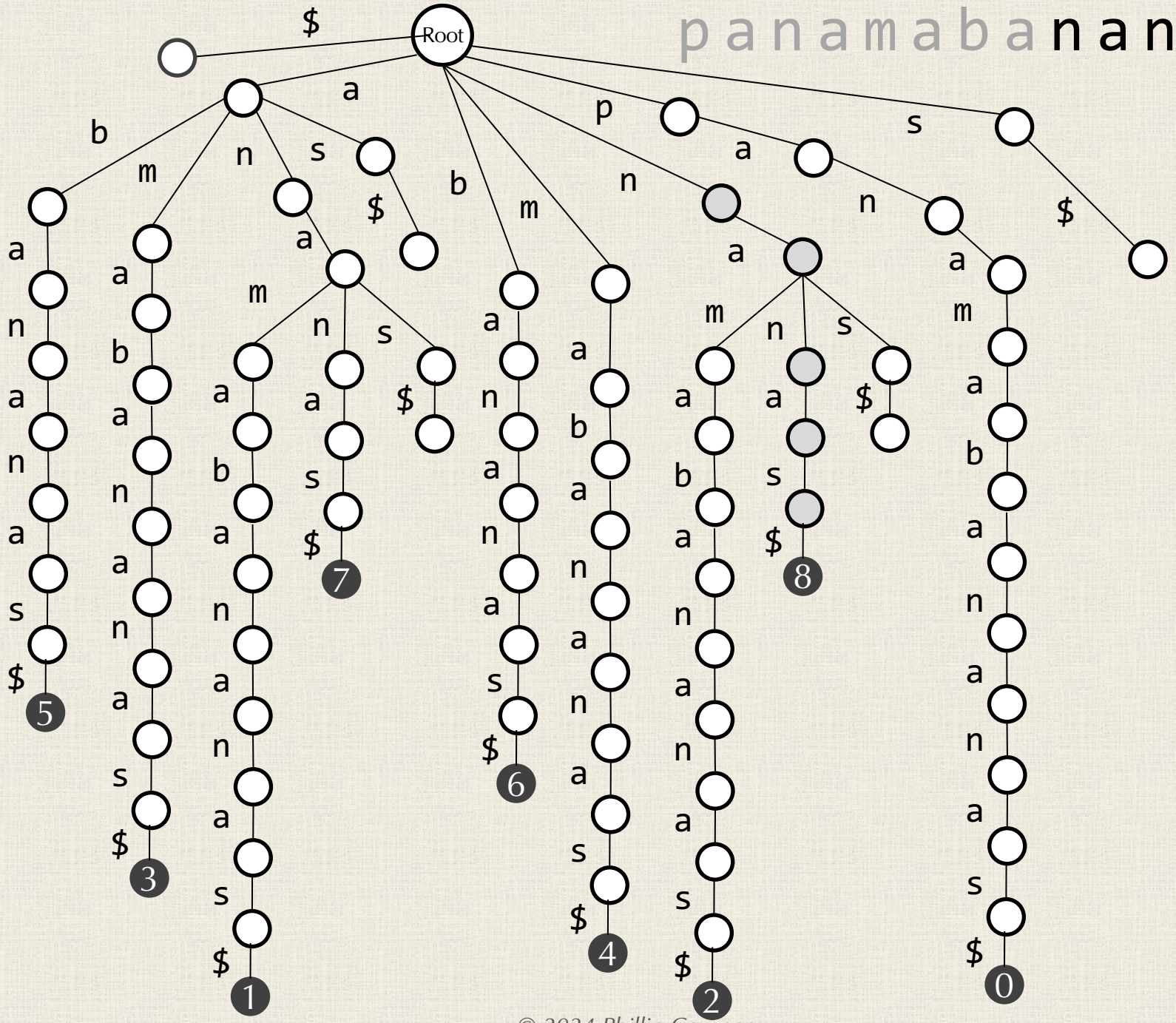


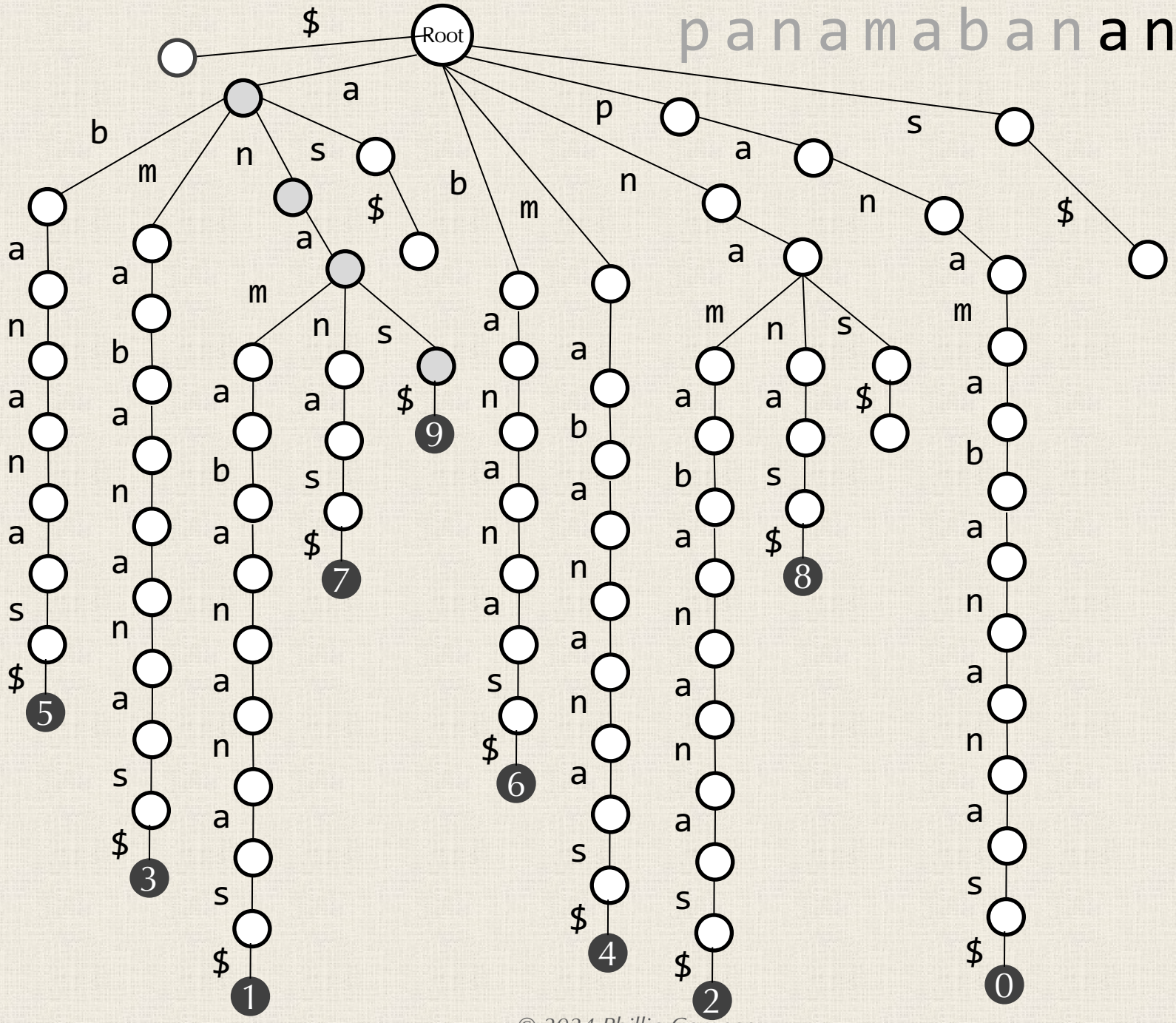


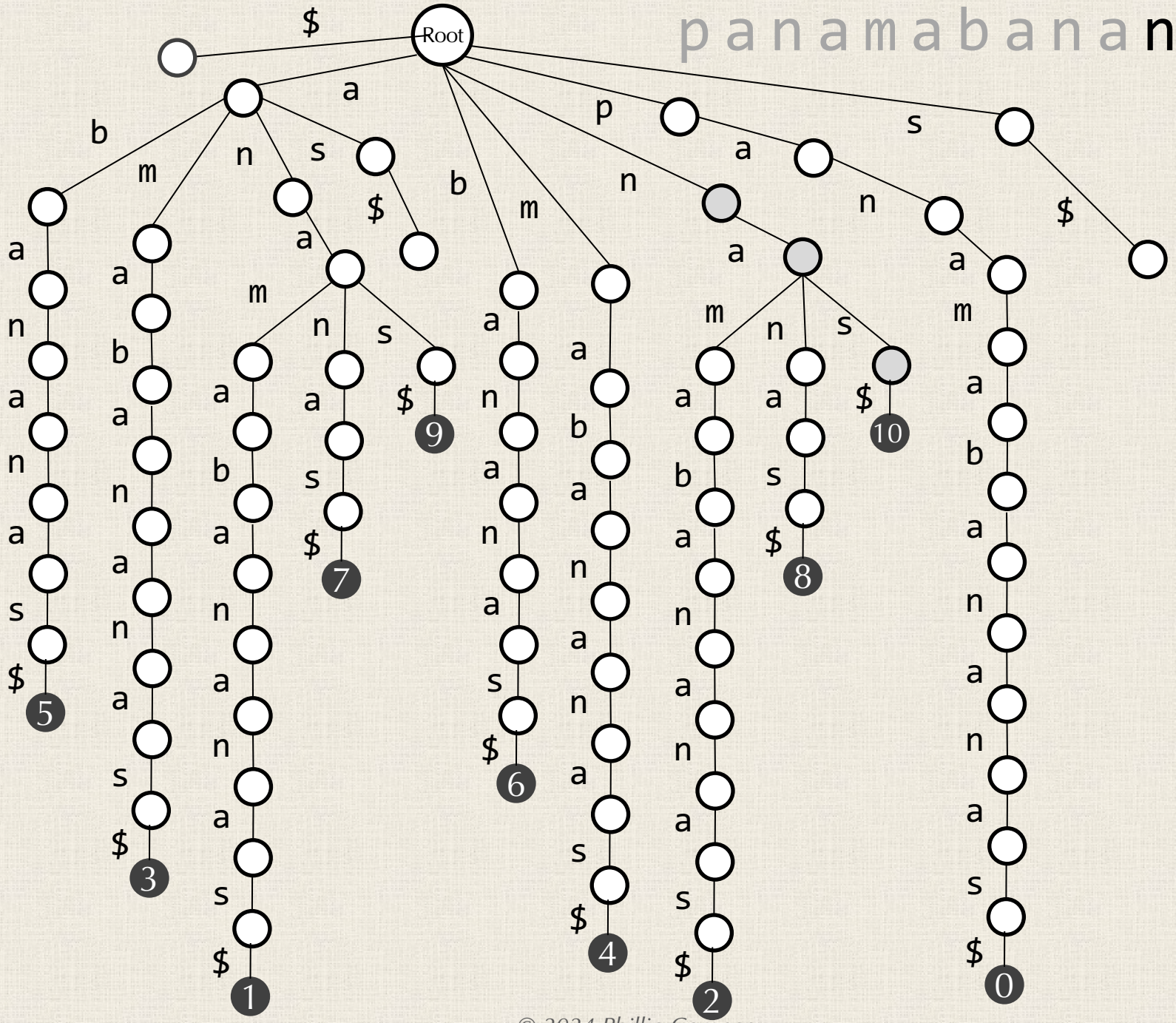


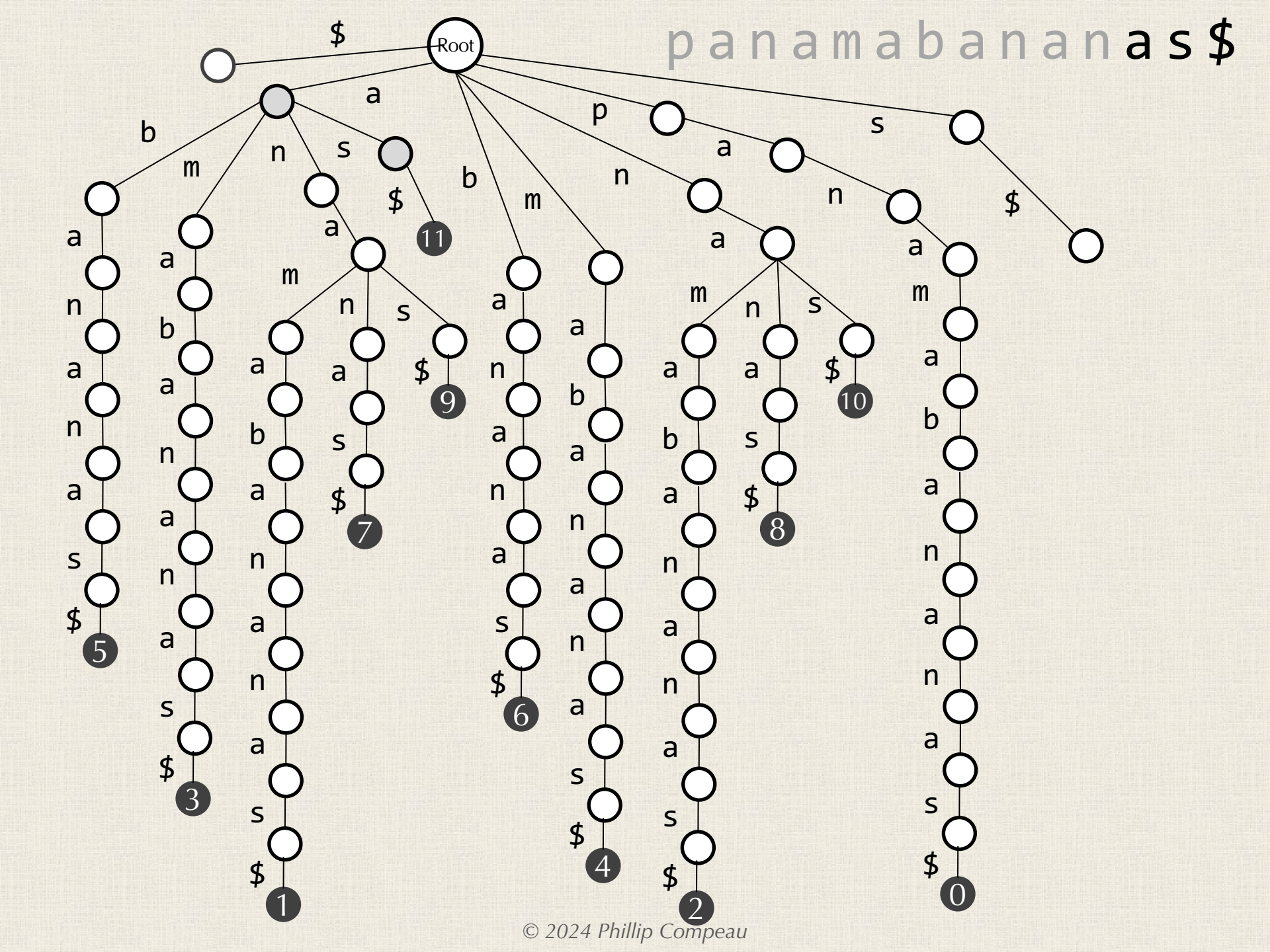


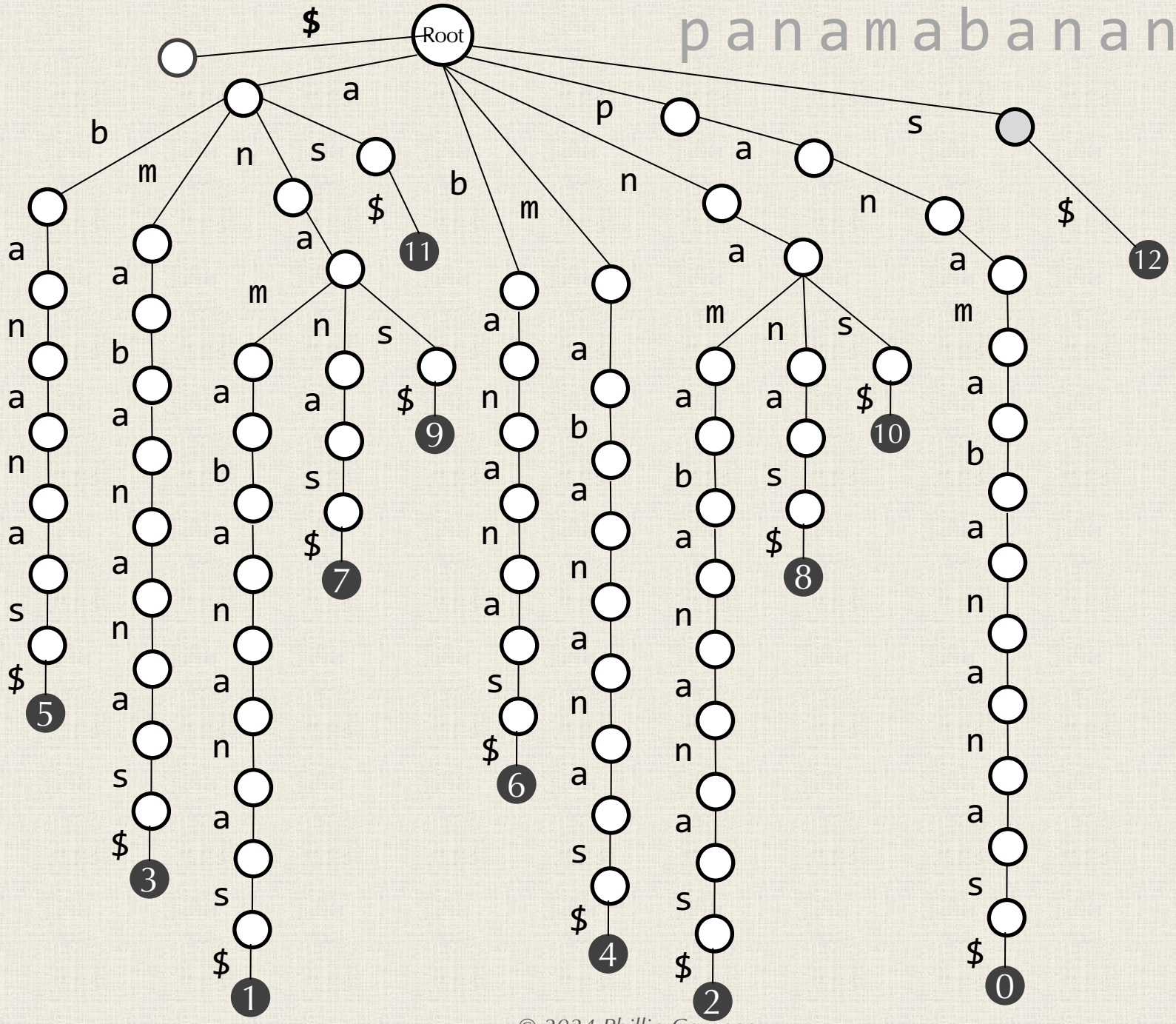


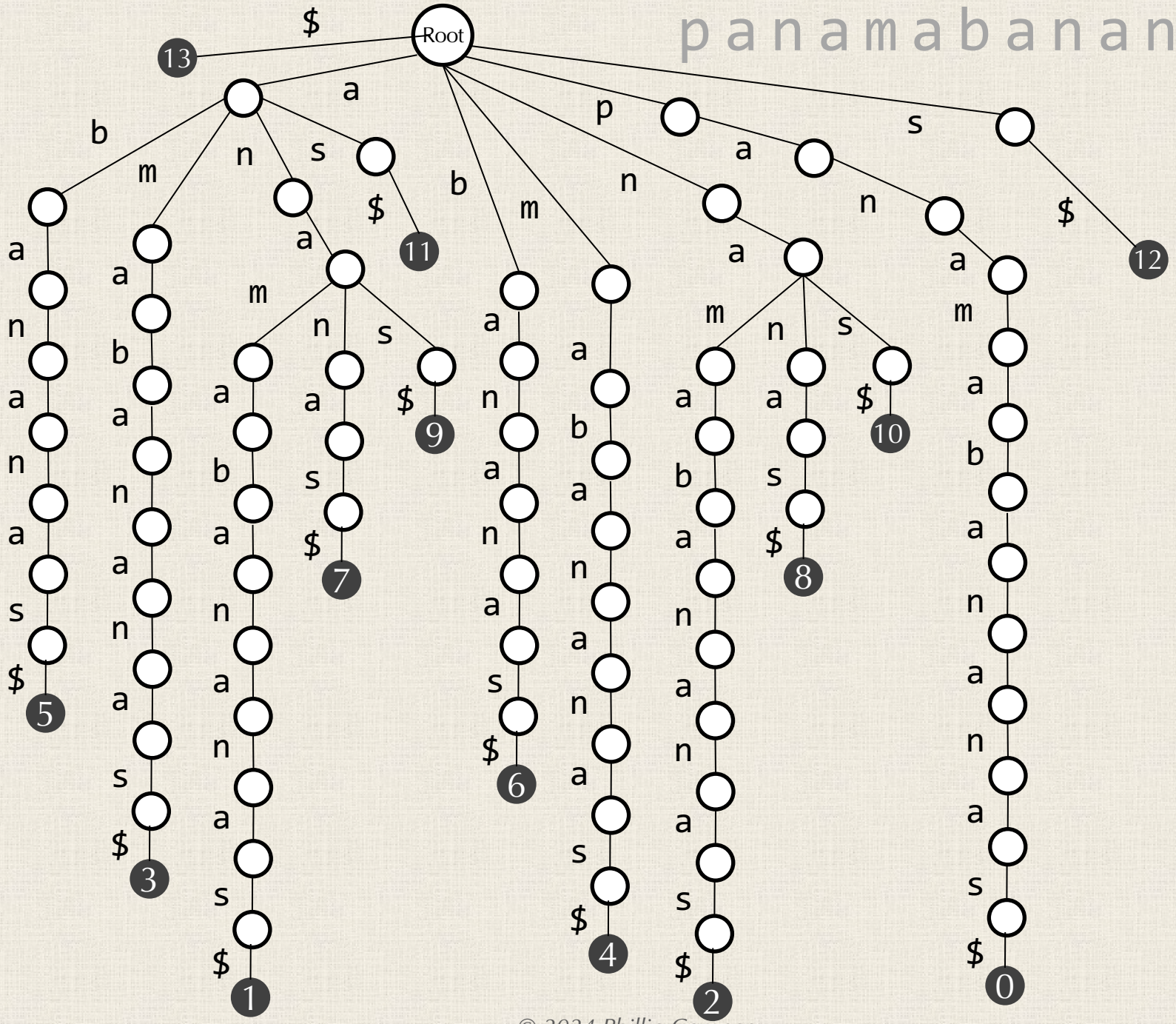


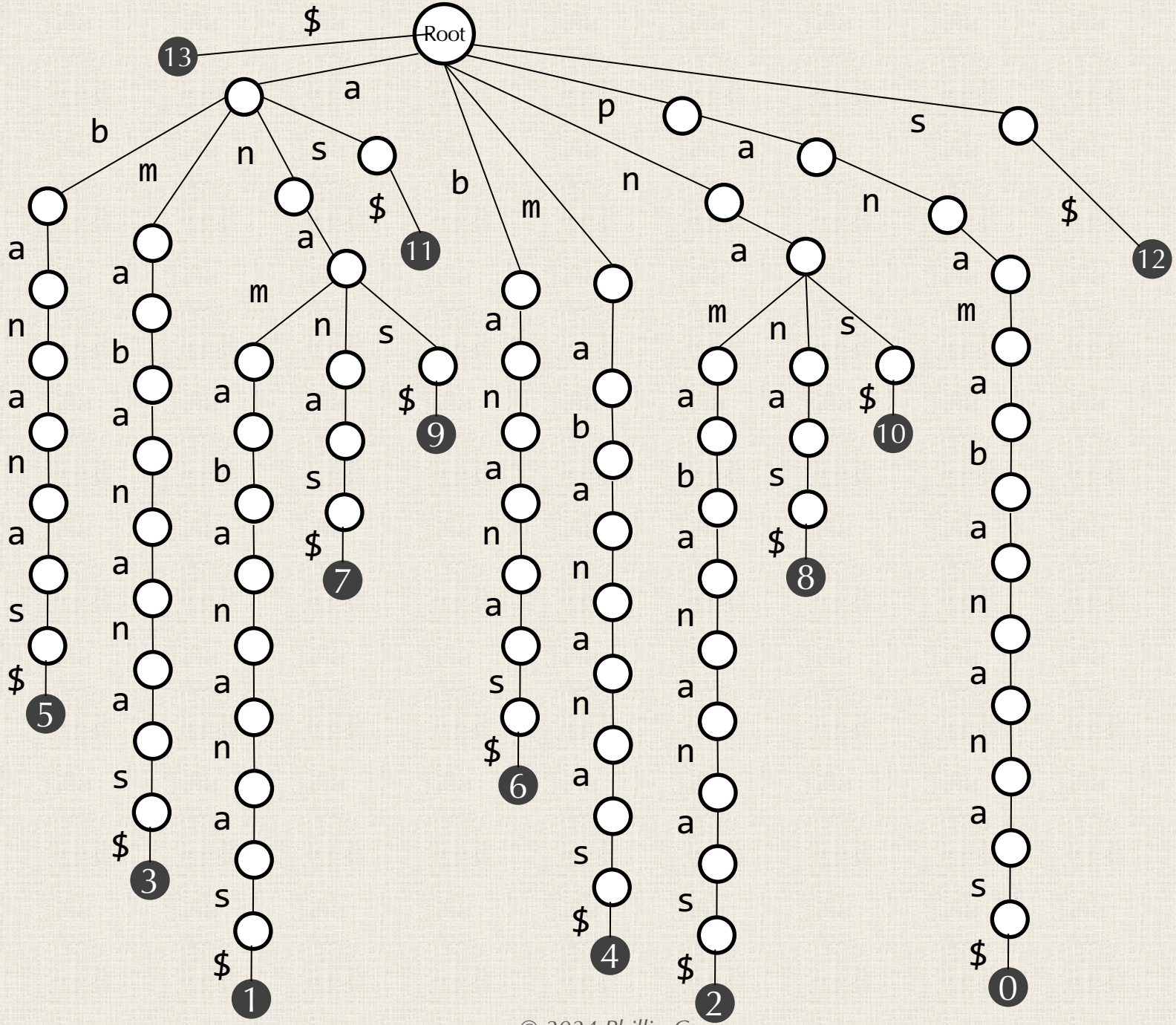


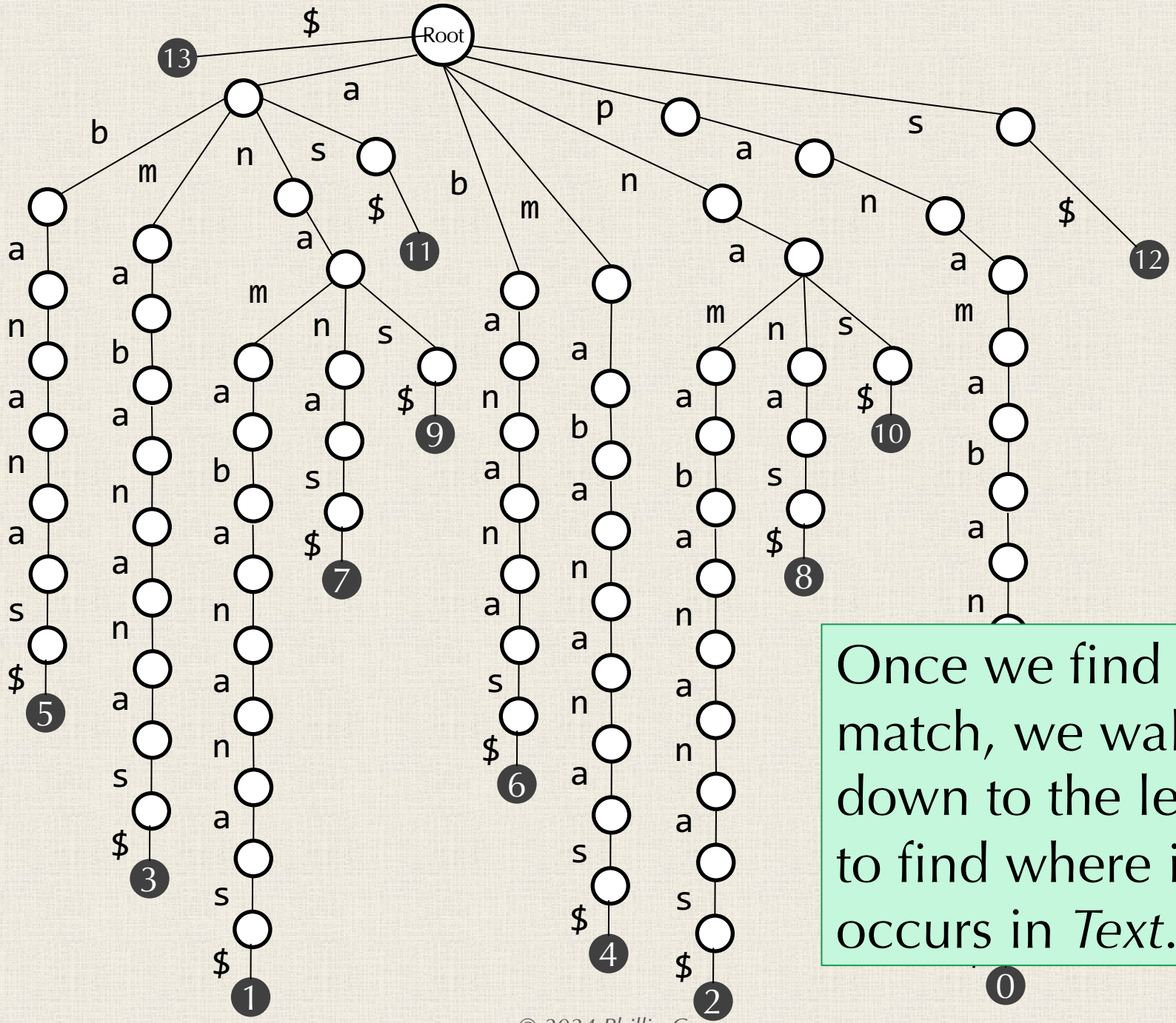






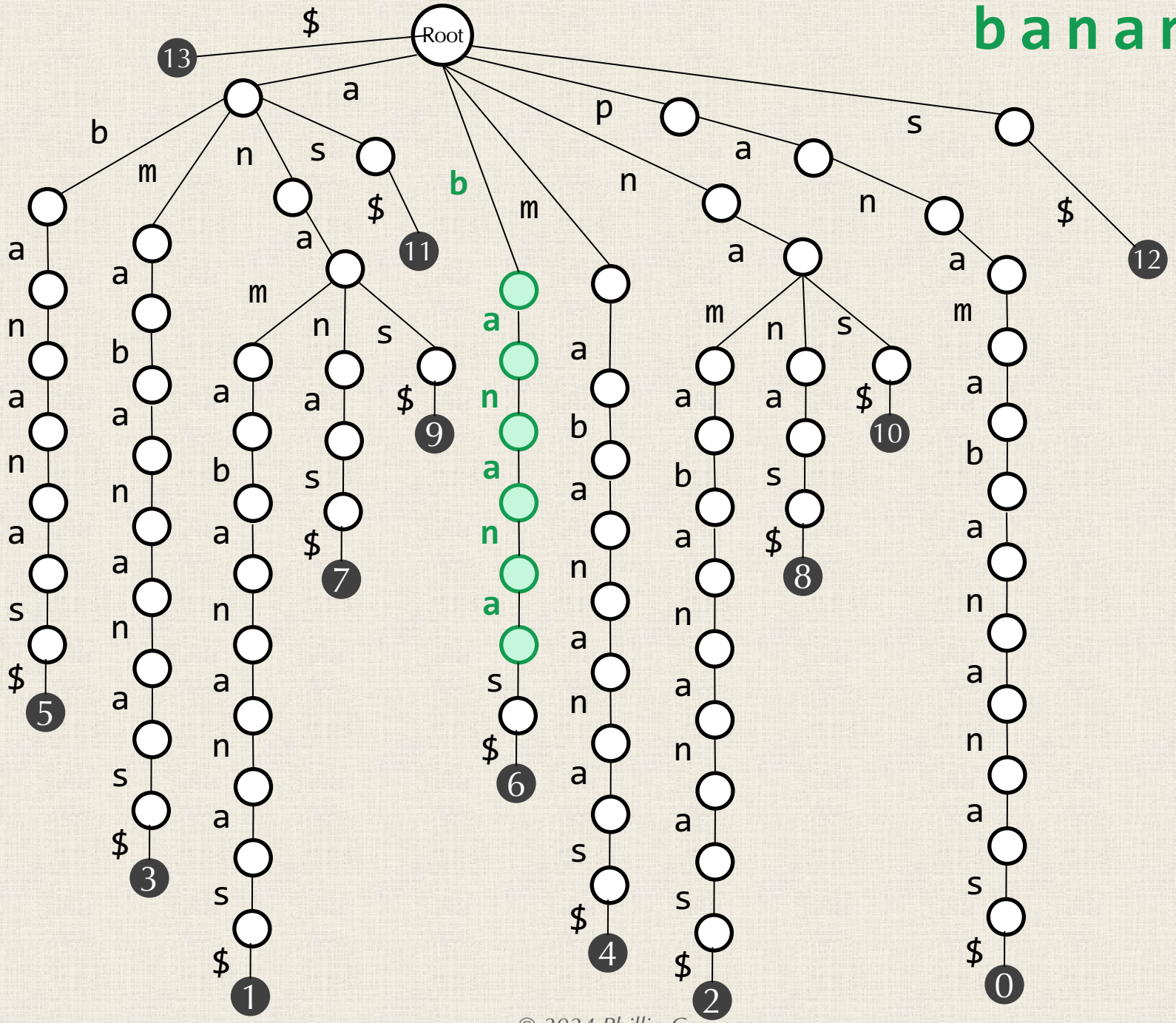




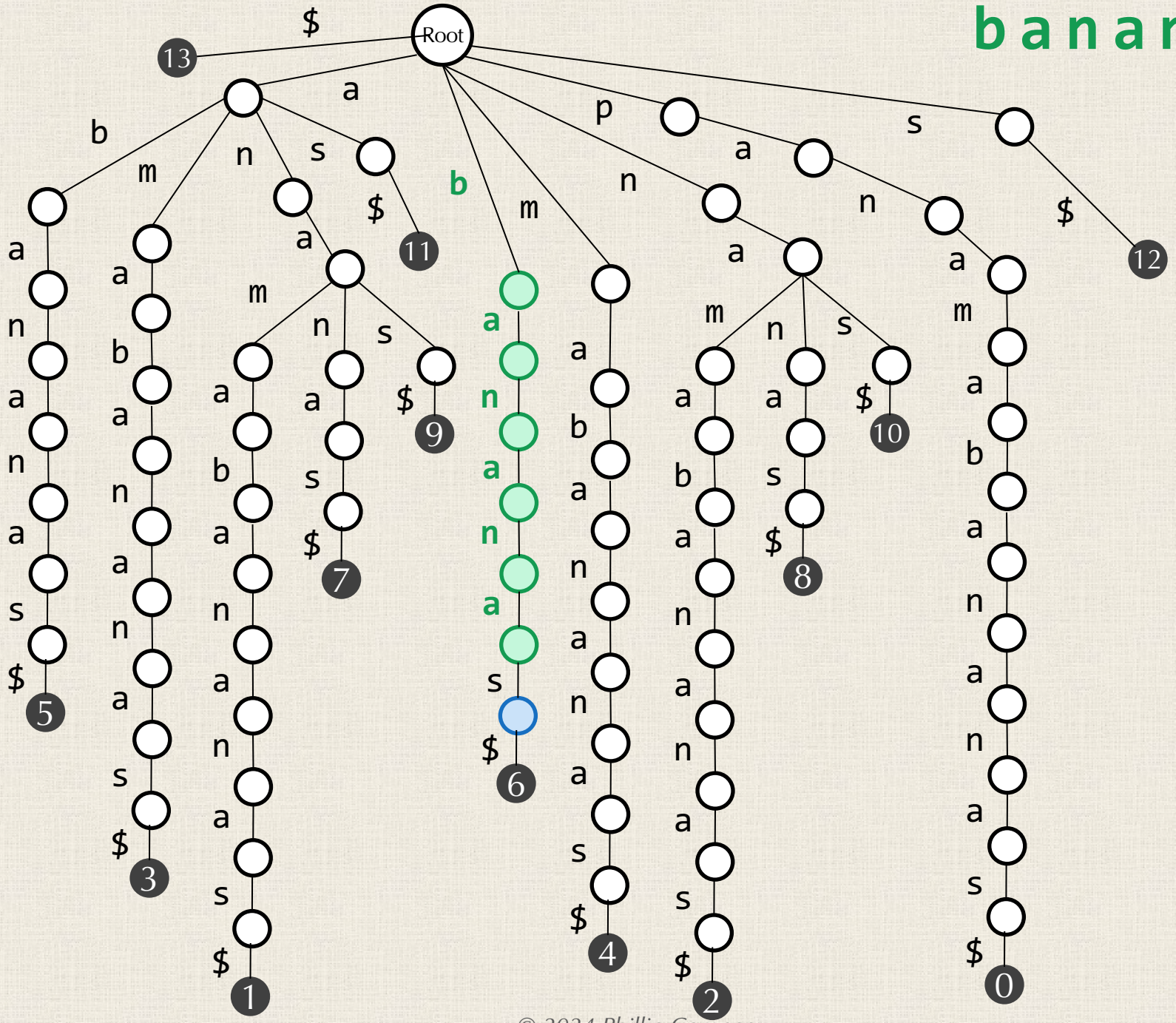


Once we find a match, we walk down to the leaves to find where it occurs in *Text*.

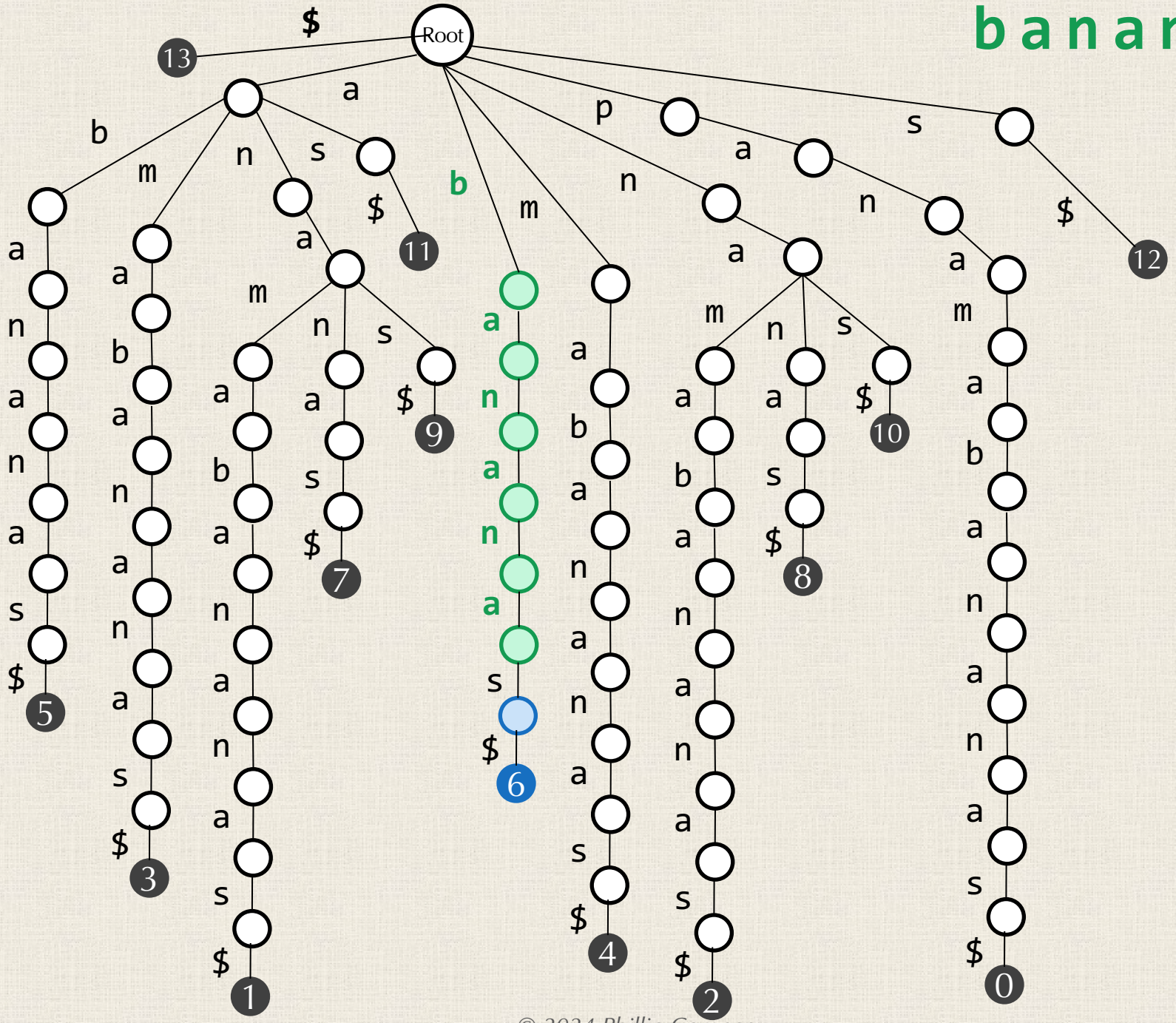
banana

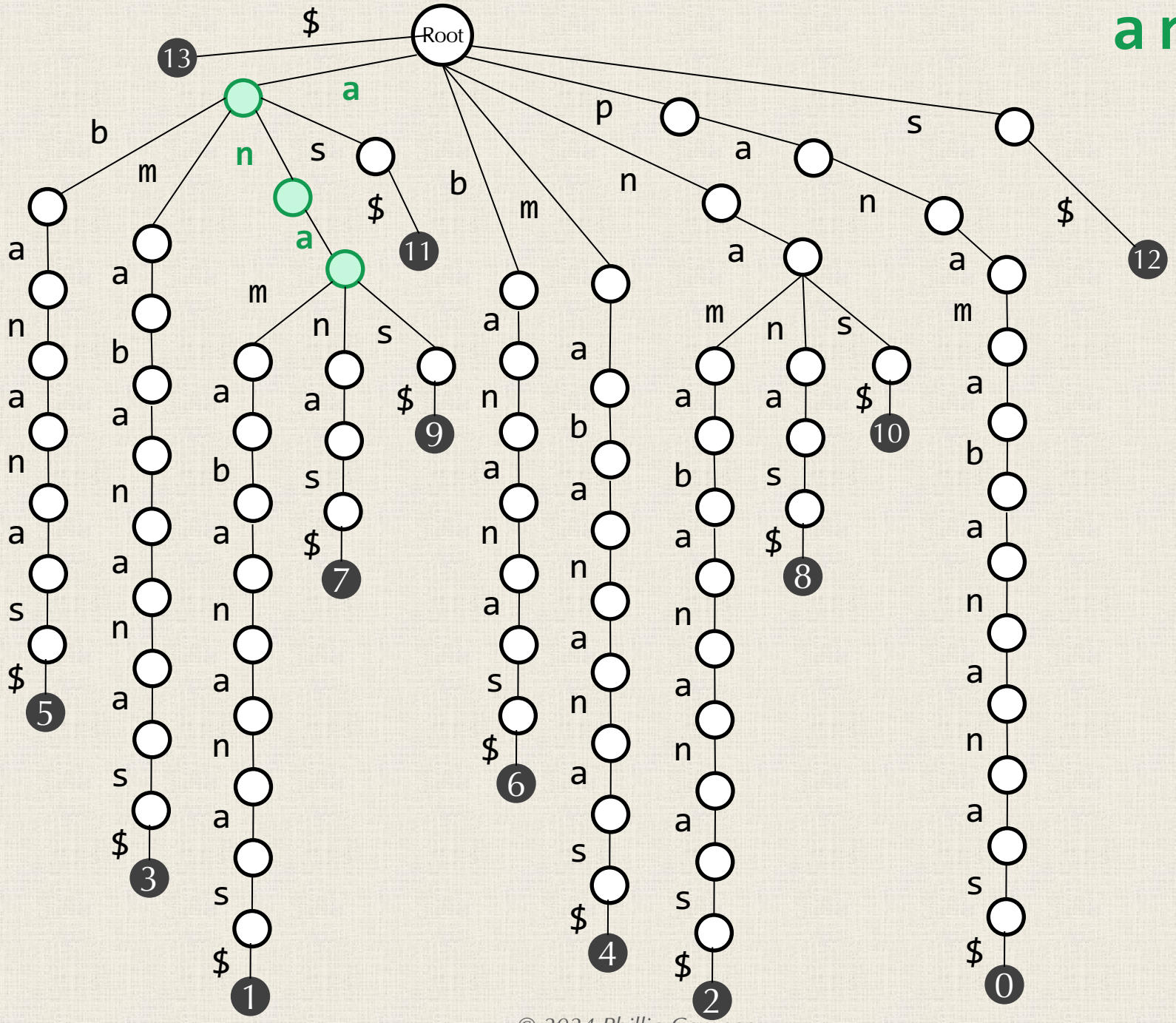


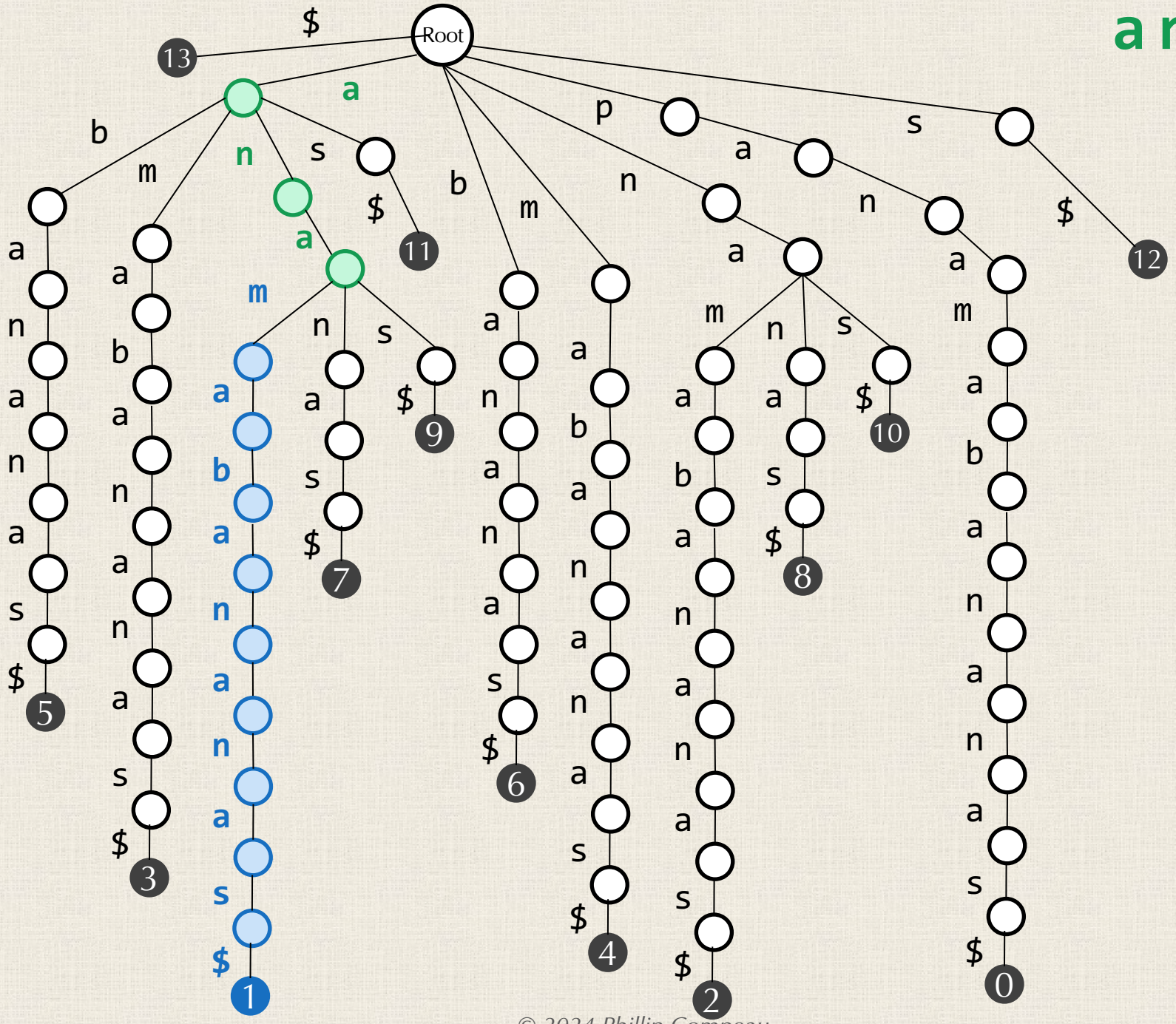
banana

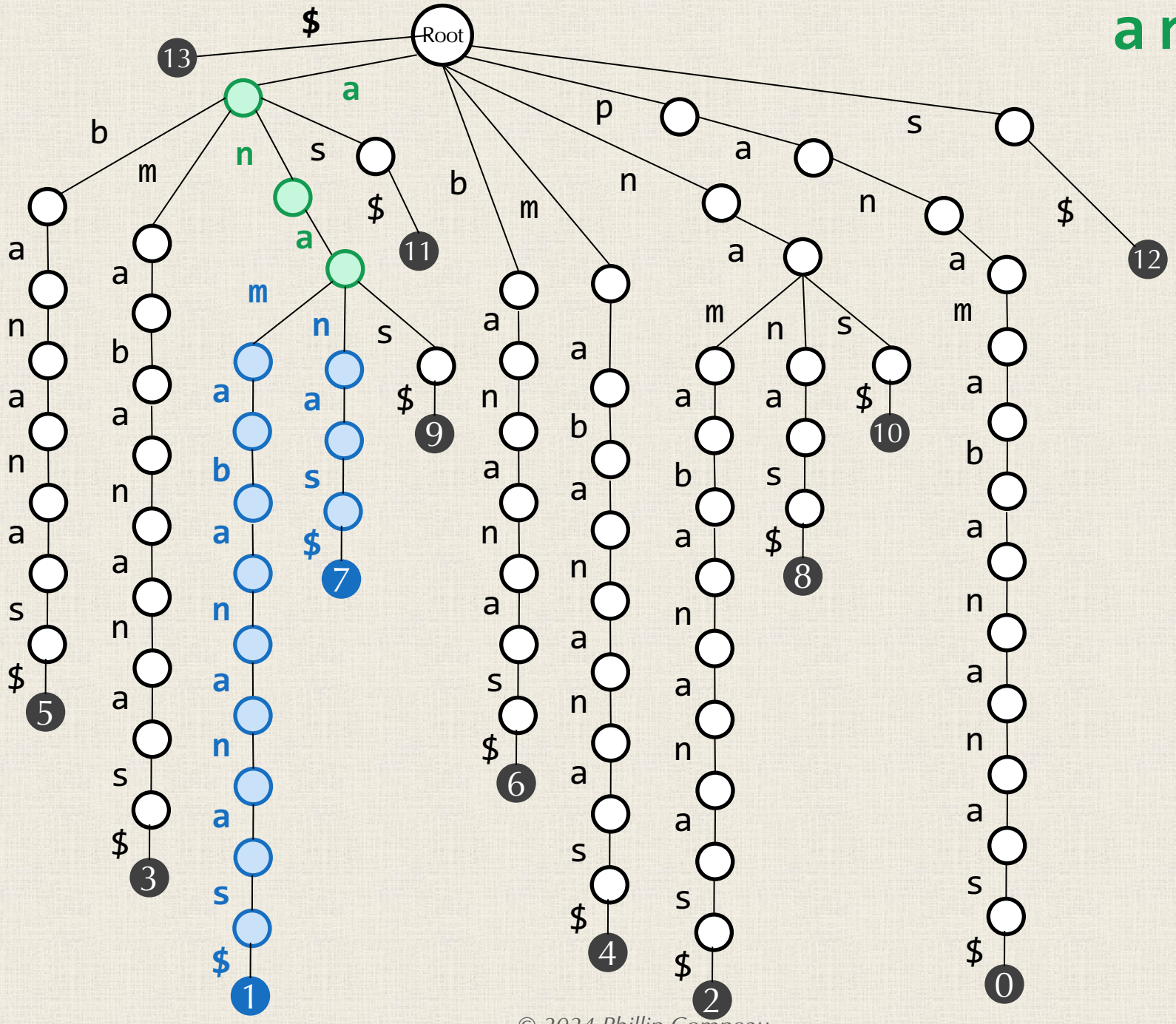


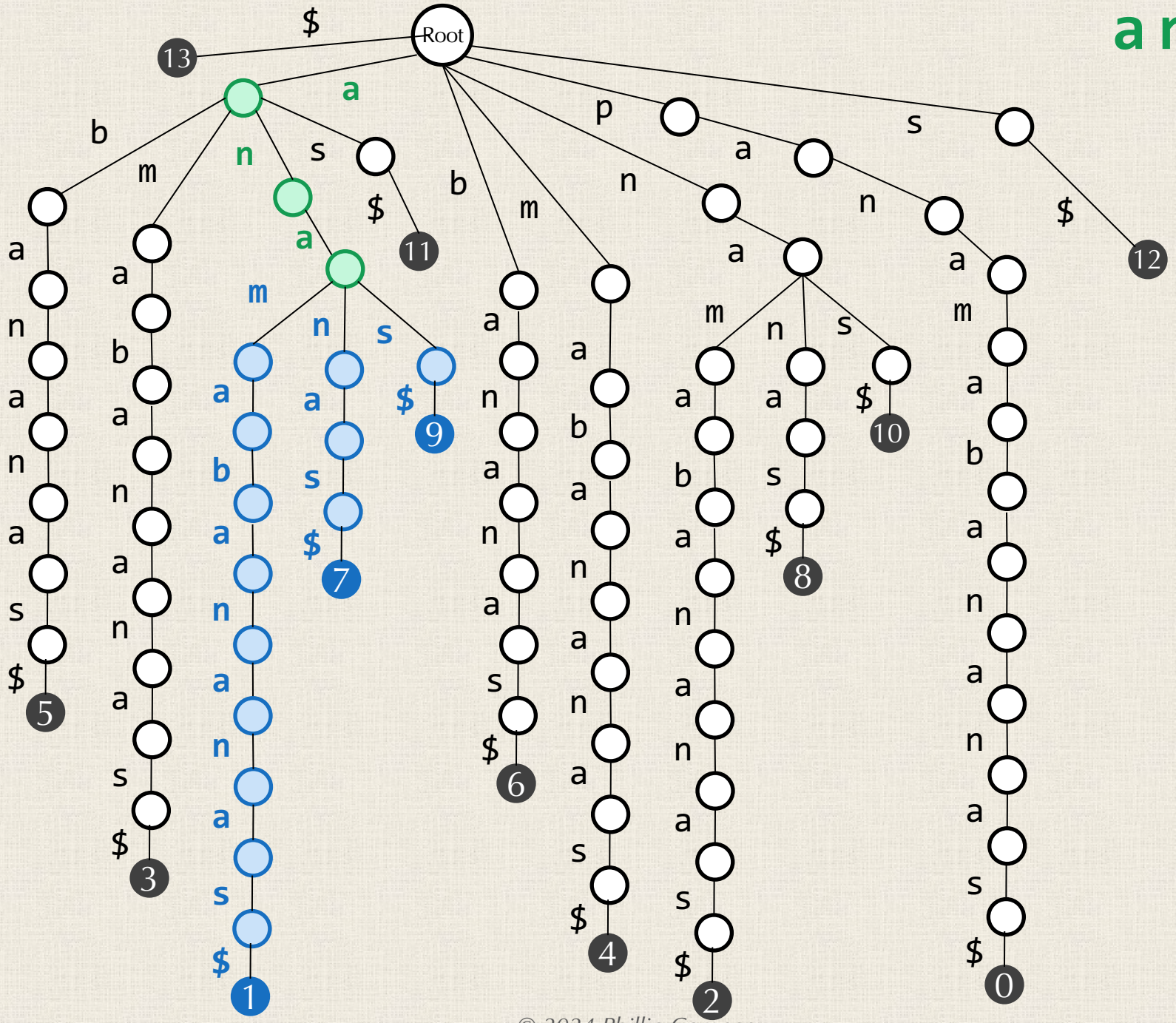
banana

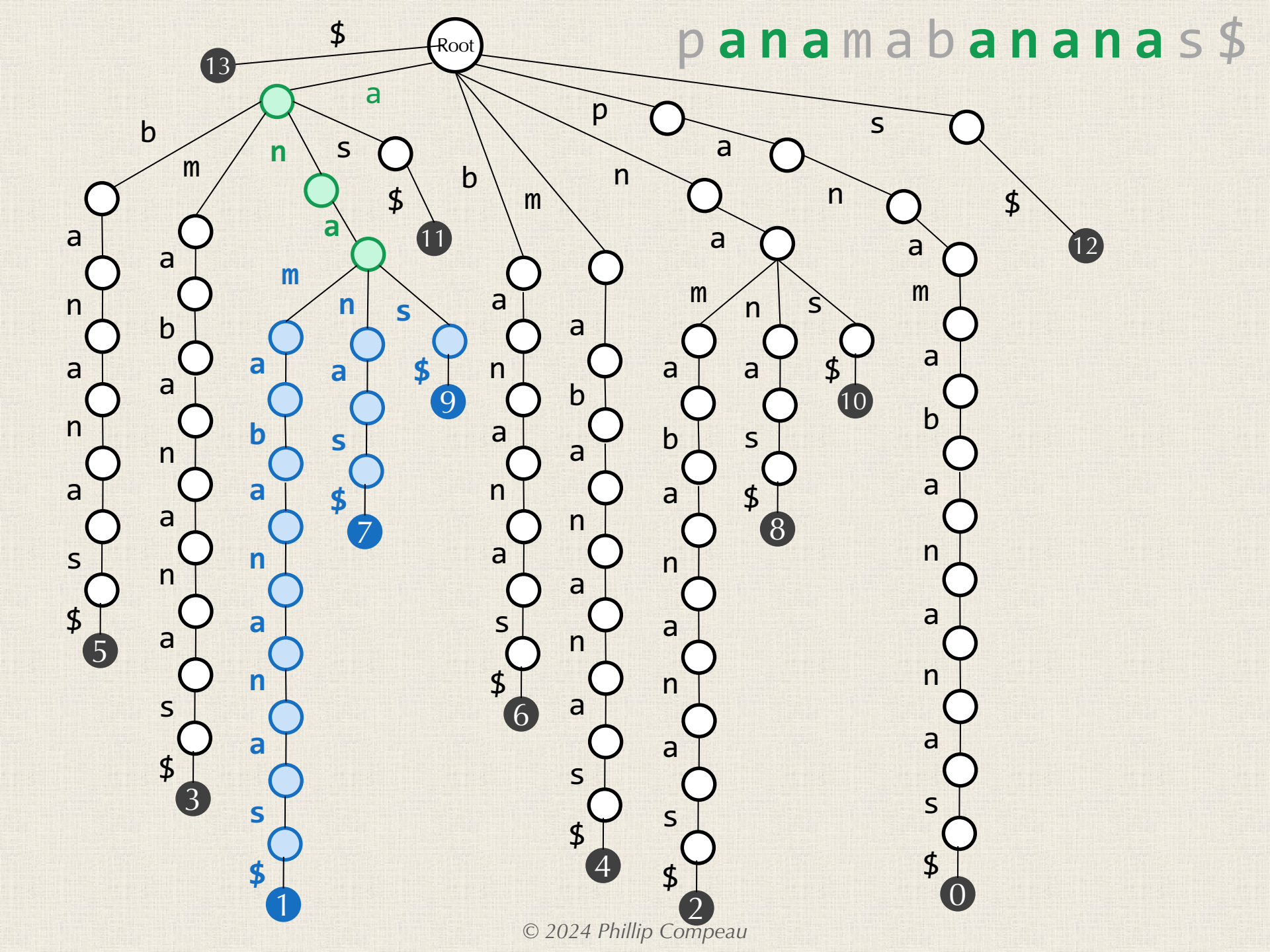


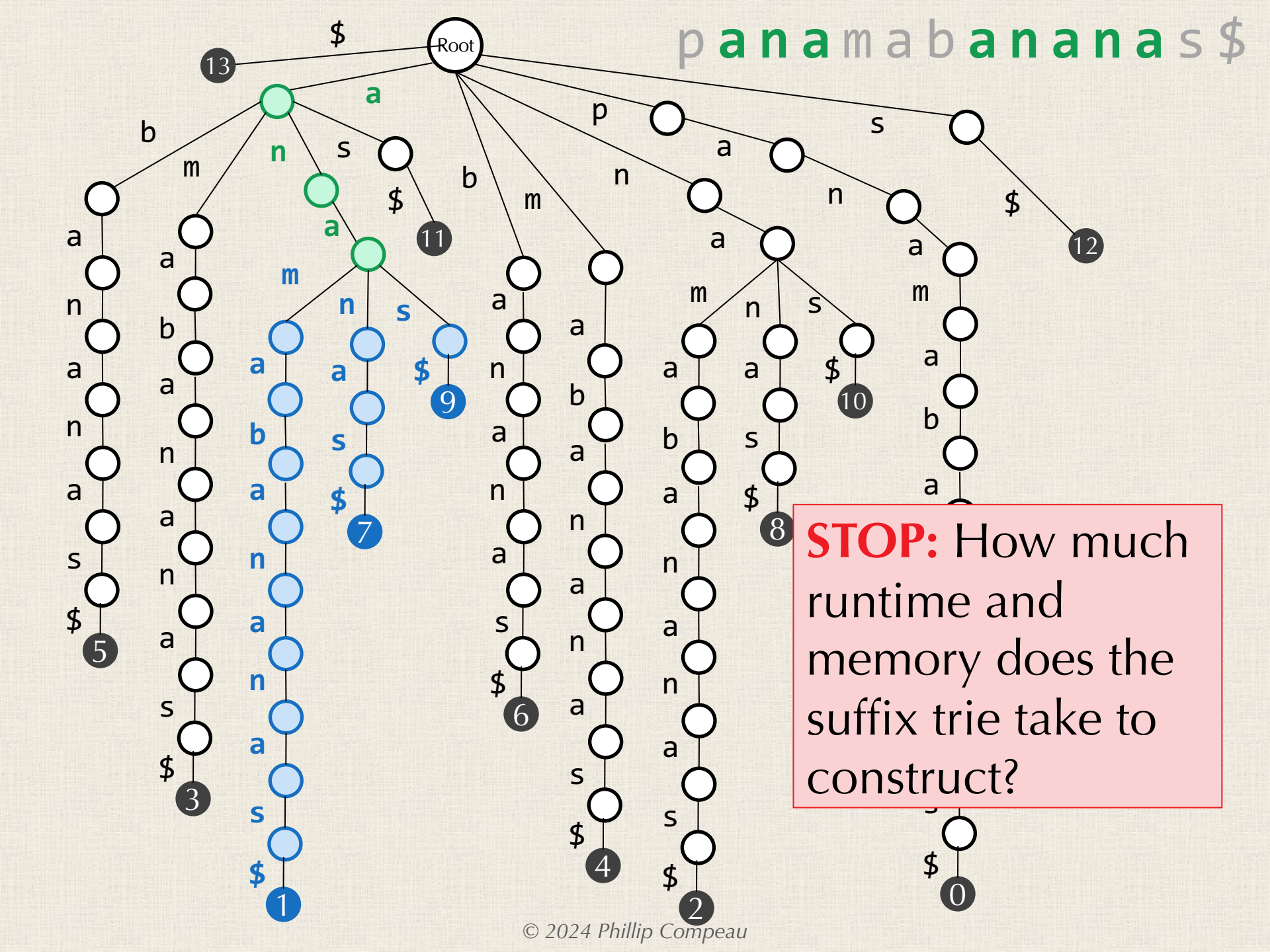












We Have Memory Troubles Again

Constructing a trie for a collection of *Patterns* takes $O(|Patterns|)$ runtime and memory.

Constructing a suffix trie takes $O(Suffixes(Text)) = O(|Text|^2)$ runtime and memory.

Suffixes

```
panamabananas$
anamabananas$
namabananas$
amabananas$
mabananas$
abananas$
bananas$
ananas$
nanas$
anas$
nas$
as$
s$
$
```

SUFFIX TREES

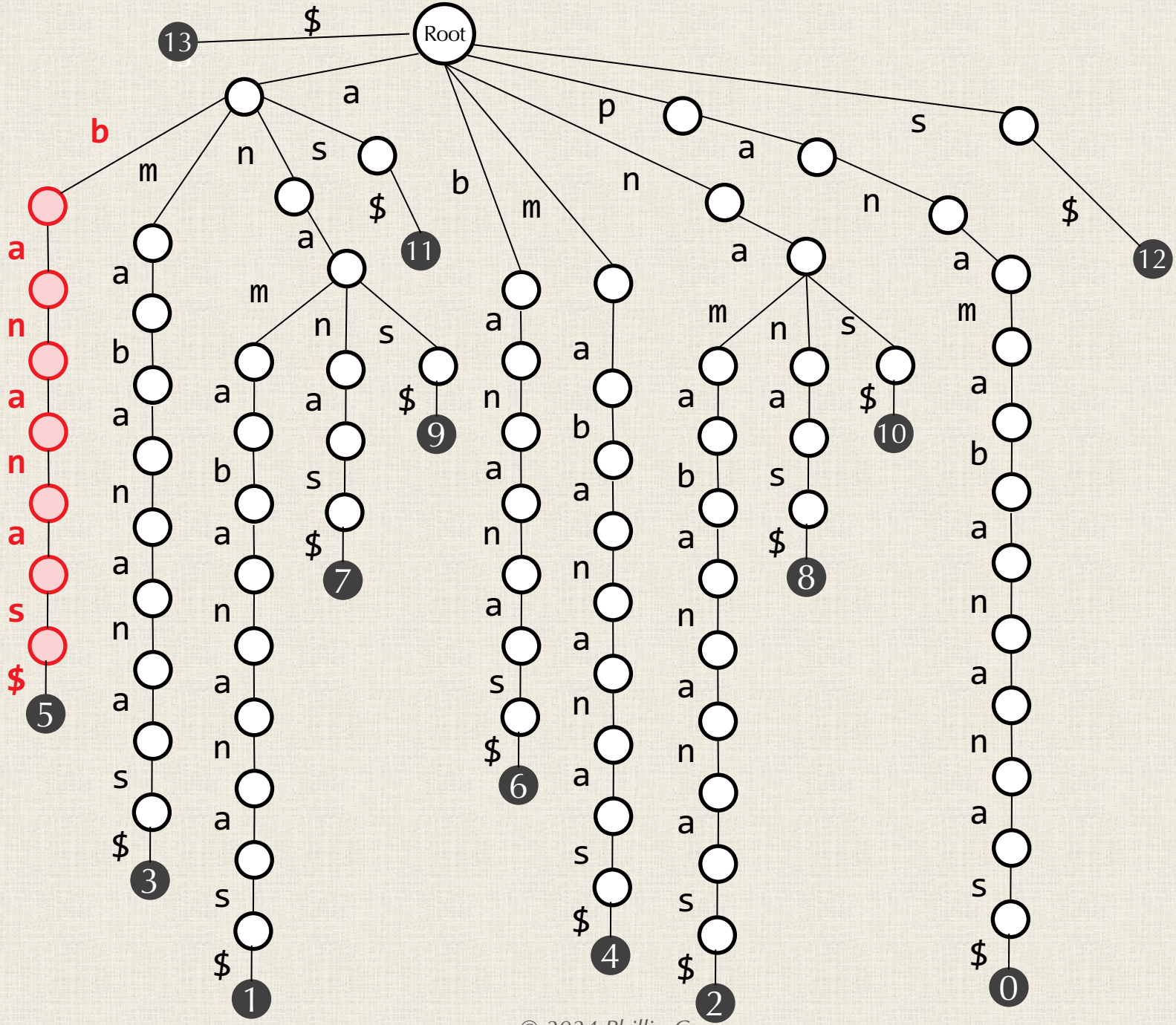
Compressing the Suffix Trie

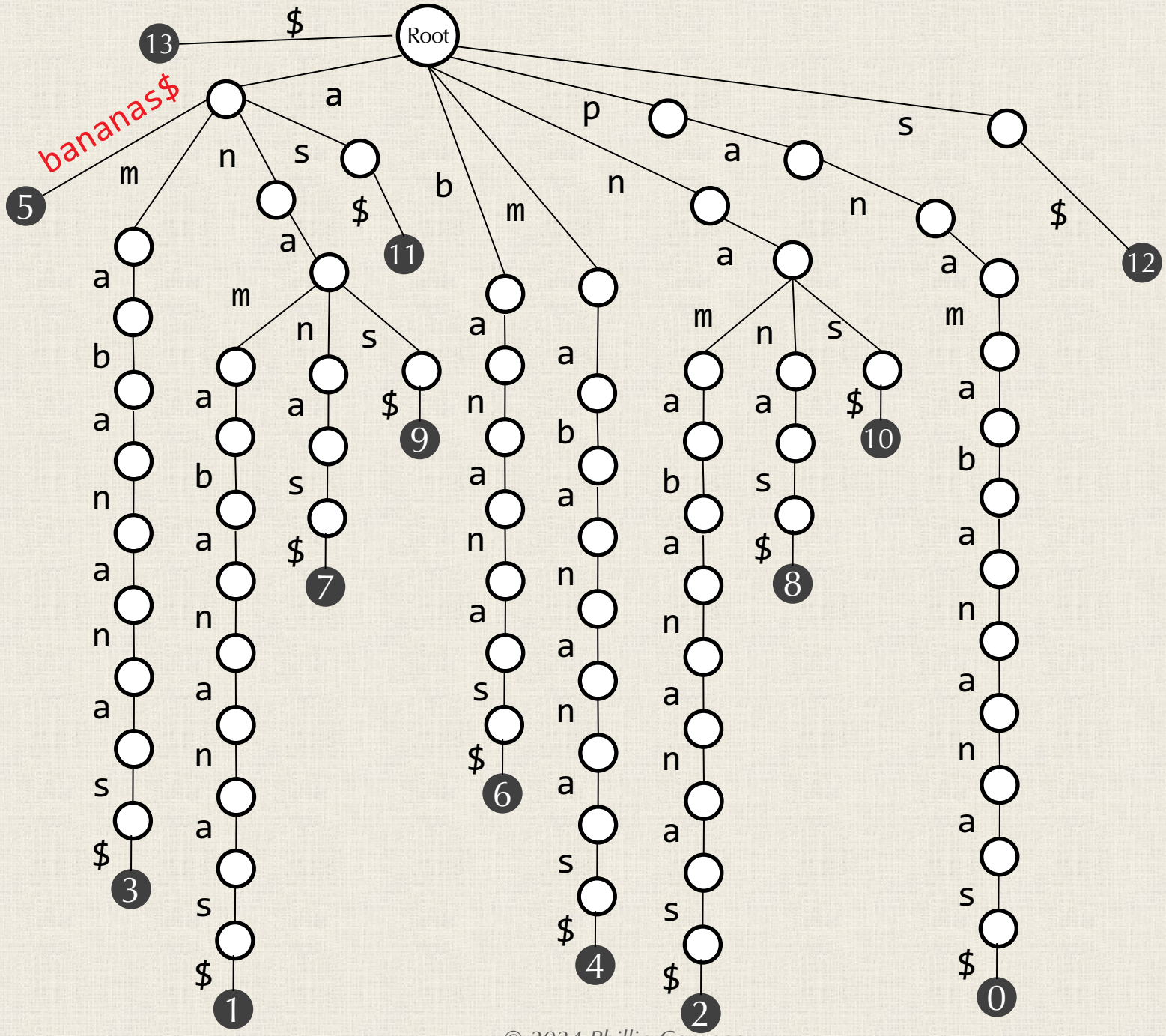
The suffix trie will only offer a memory improvement if $|Text|^2 < |Patterns|$ (i.e., for very short genomes and many reads). But we can compress the suffix trie into a smaller structure.

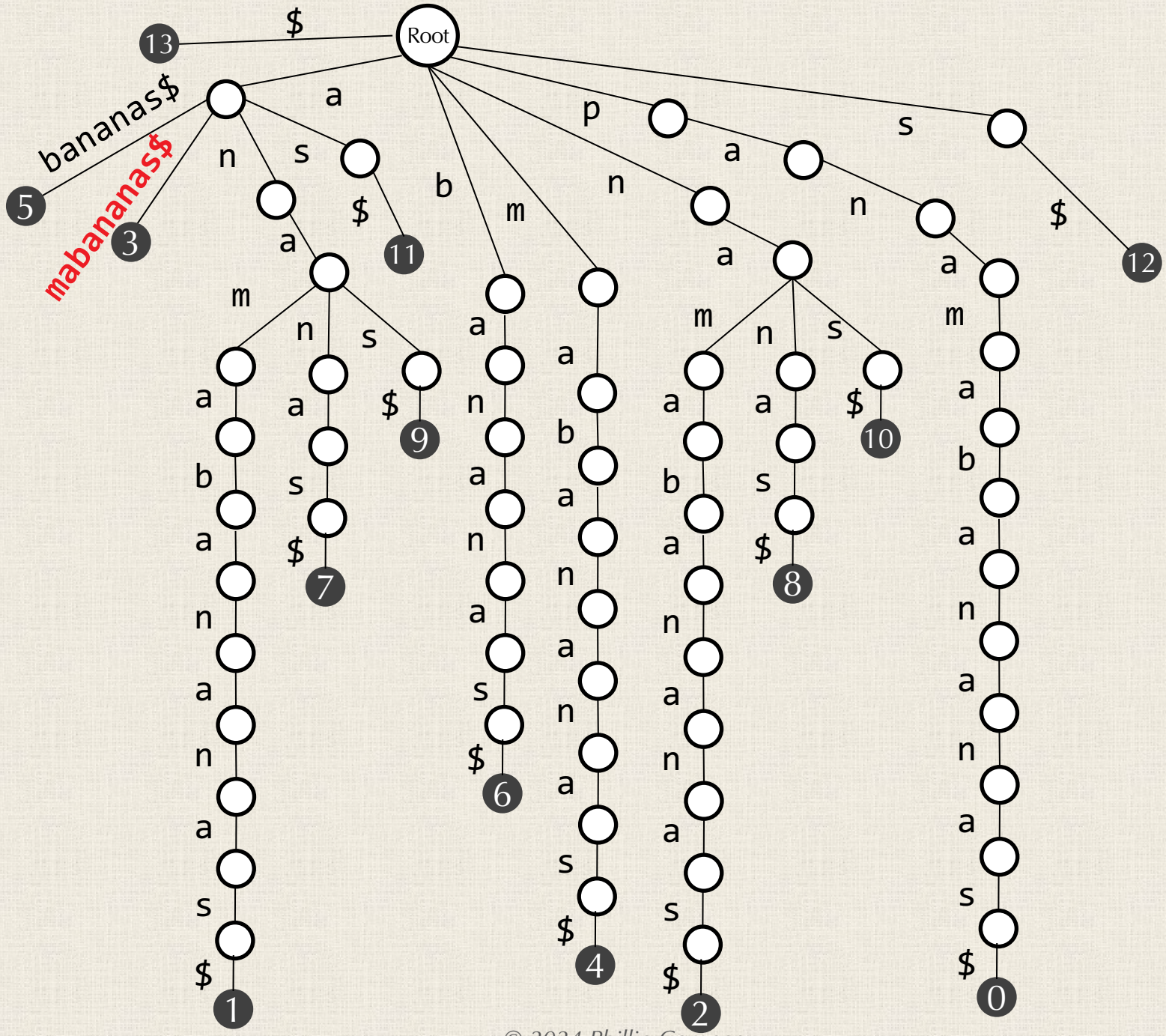
Compressing the Suffix Trie

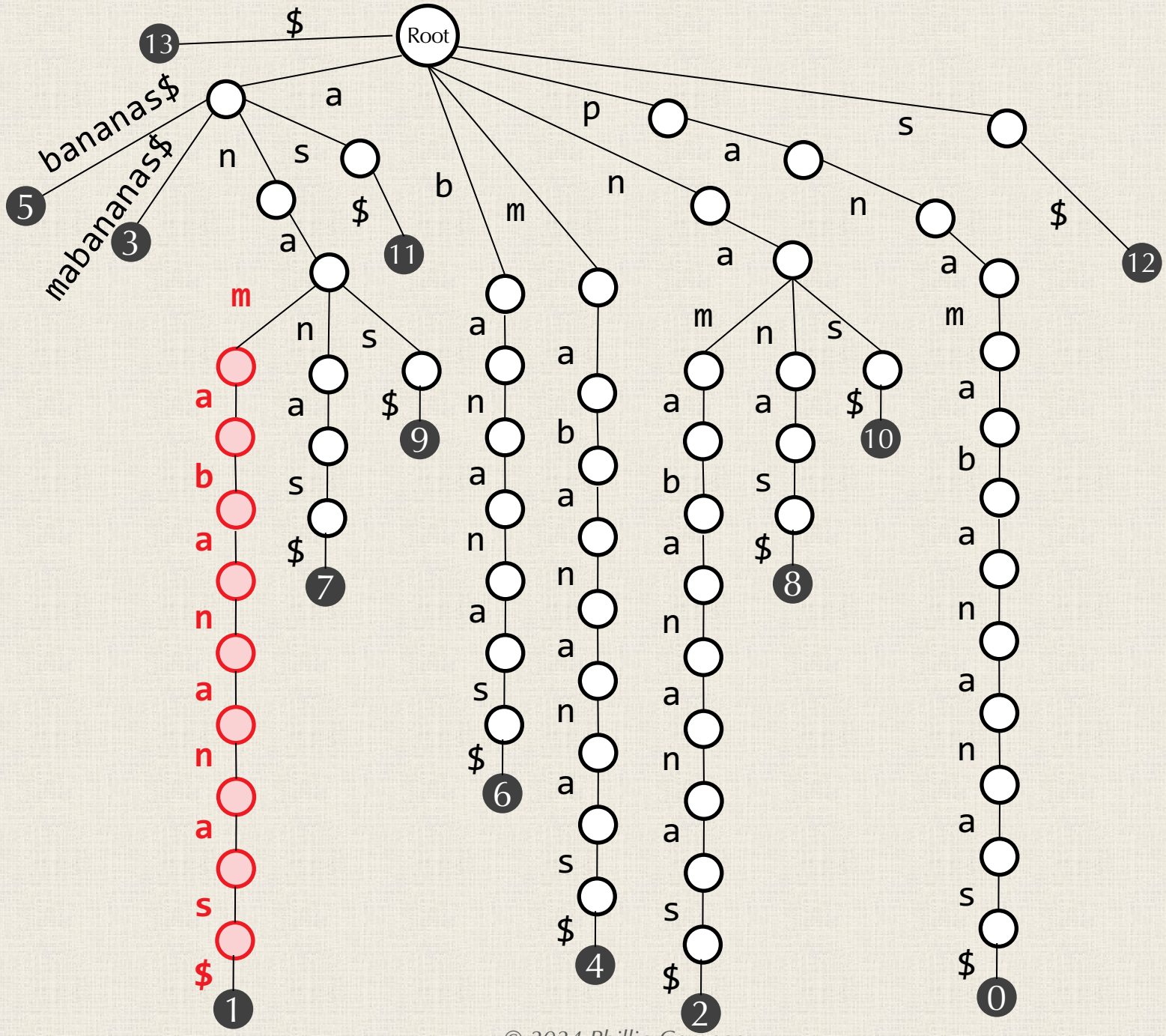
The suffix trie will only offer a memory improvement if $|Text|^2 < |Patterns|$ (i.e., for very short genomes and many reads). But we can compress the suffix trie into a smaller structure.

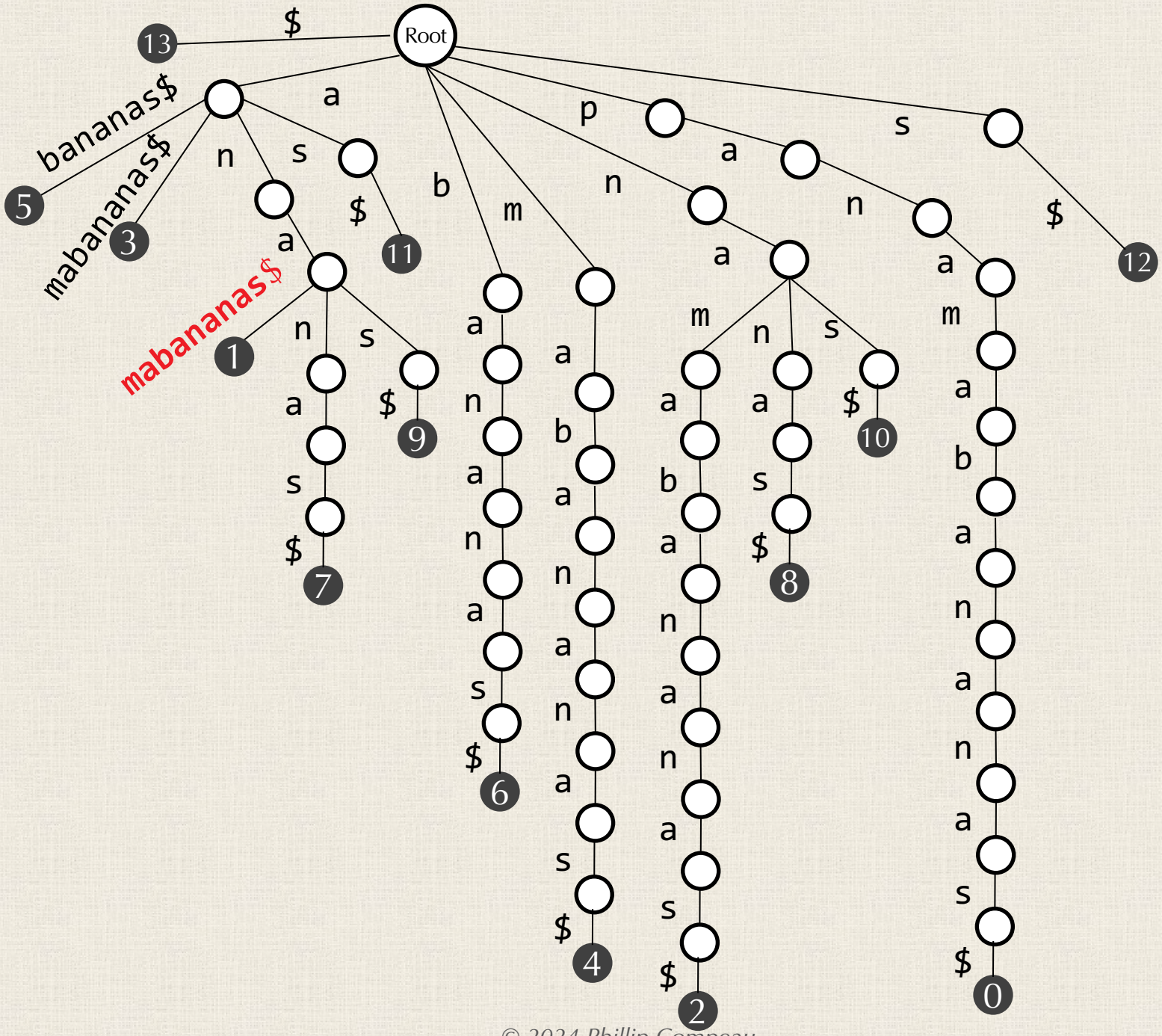
Suffix Tree: Formed by combining the edges on any *maximal non-branching path* of the suffix trie into a single edge.

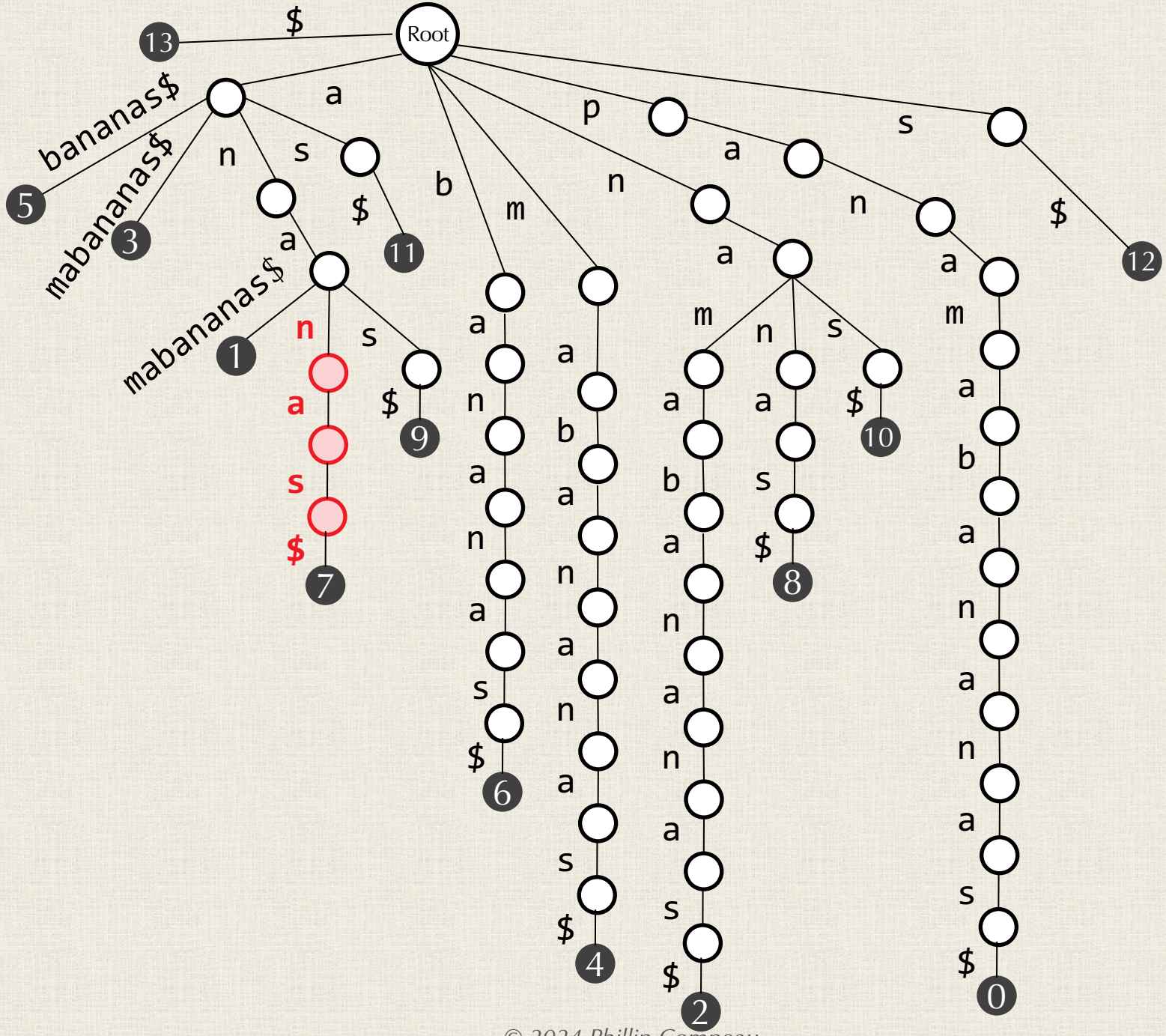


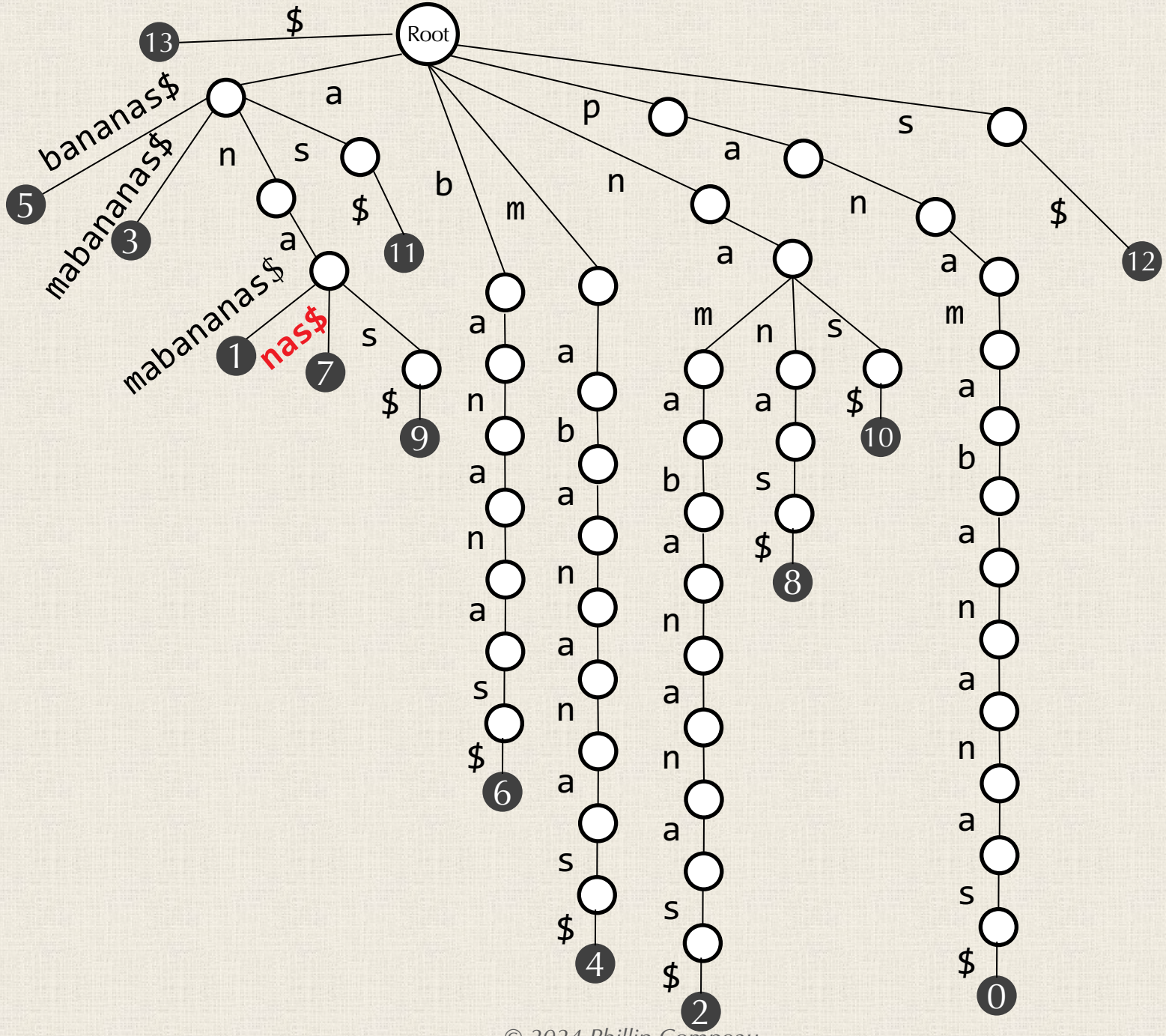


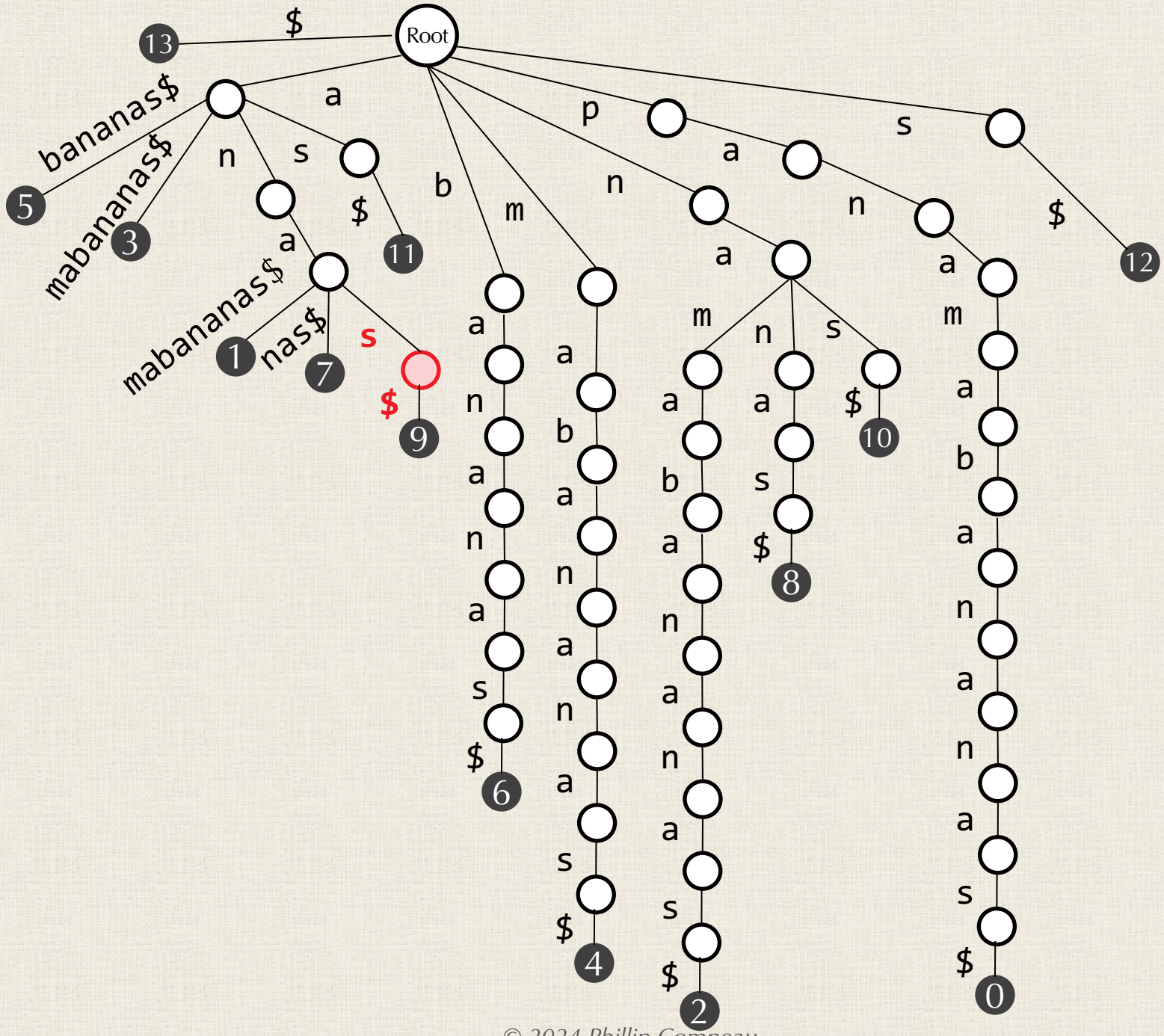


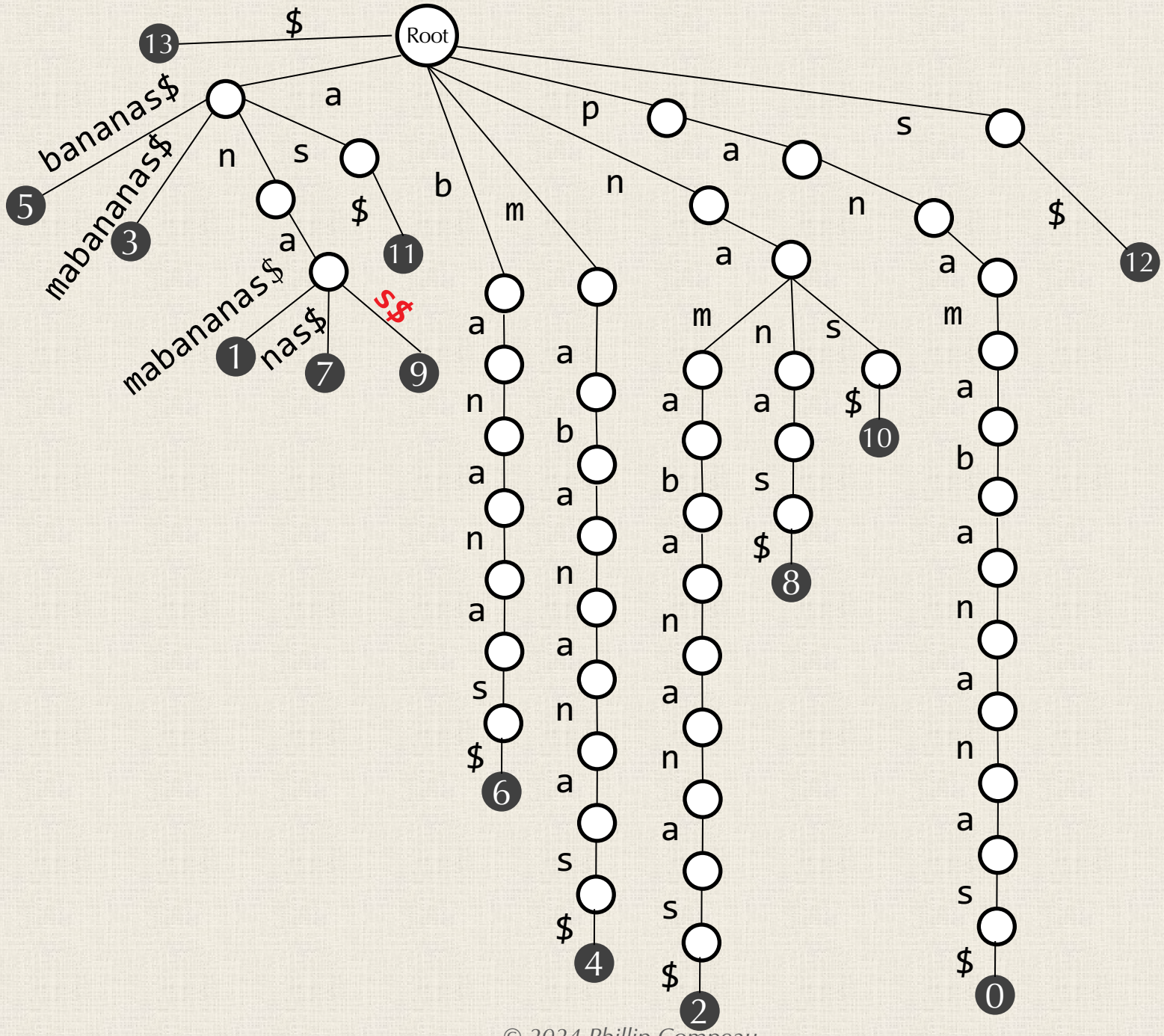


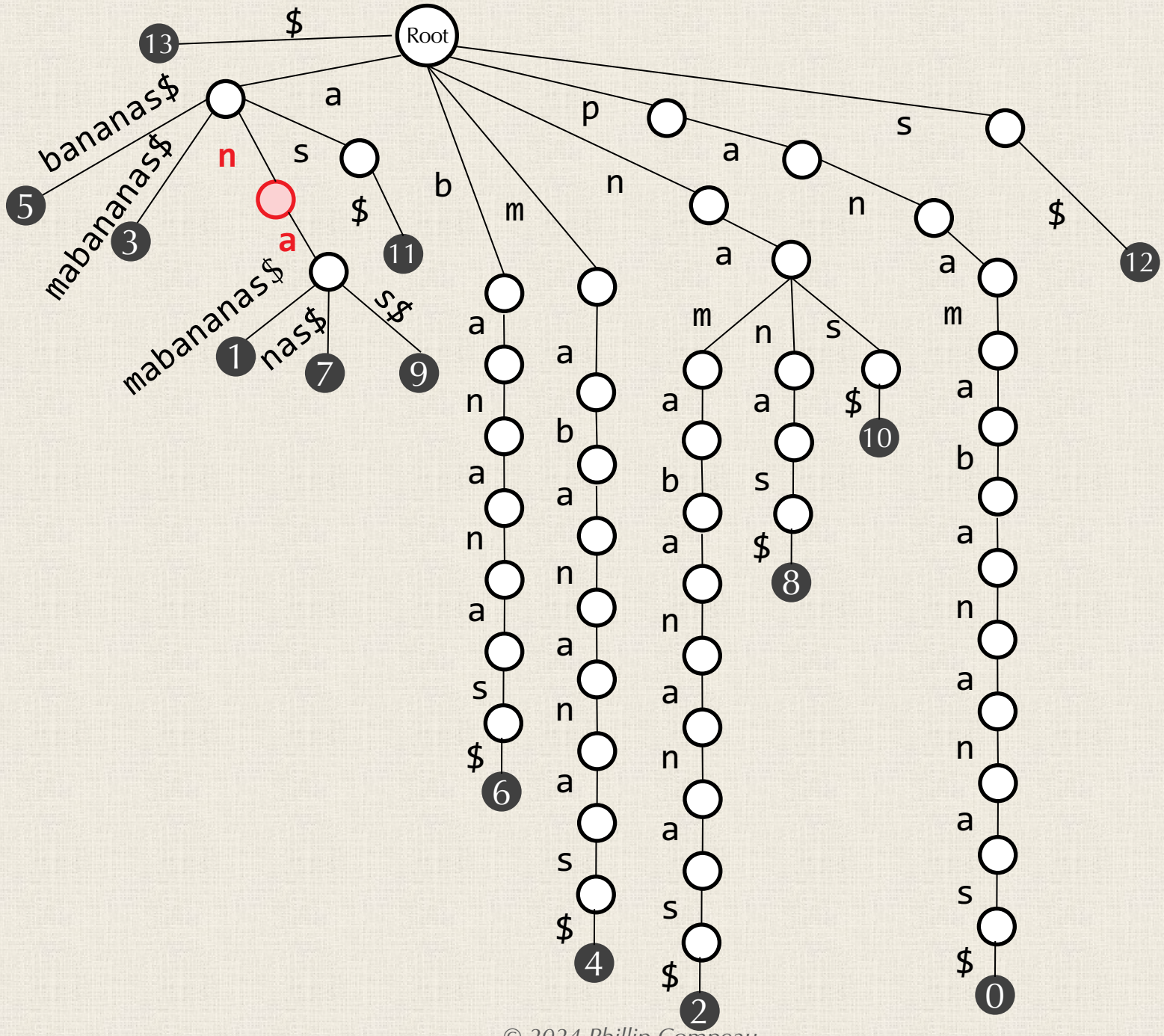


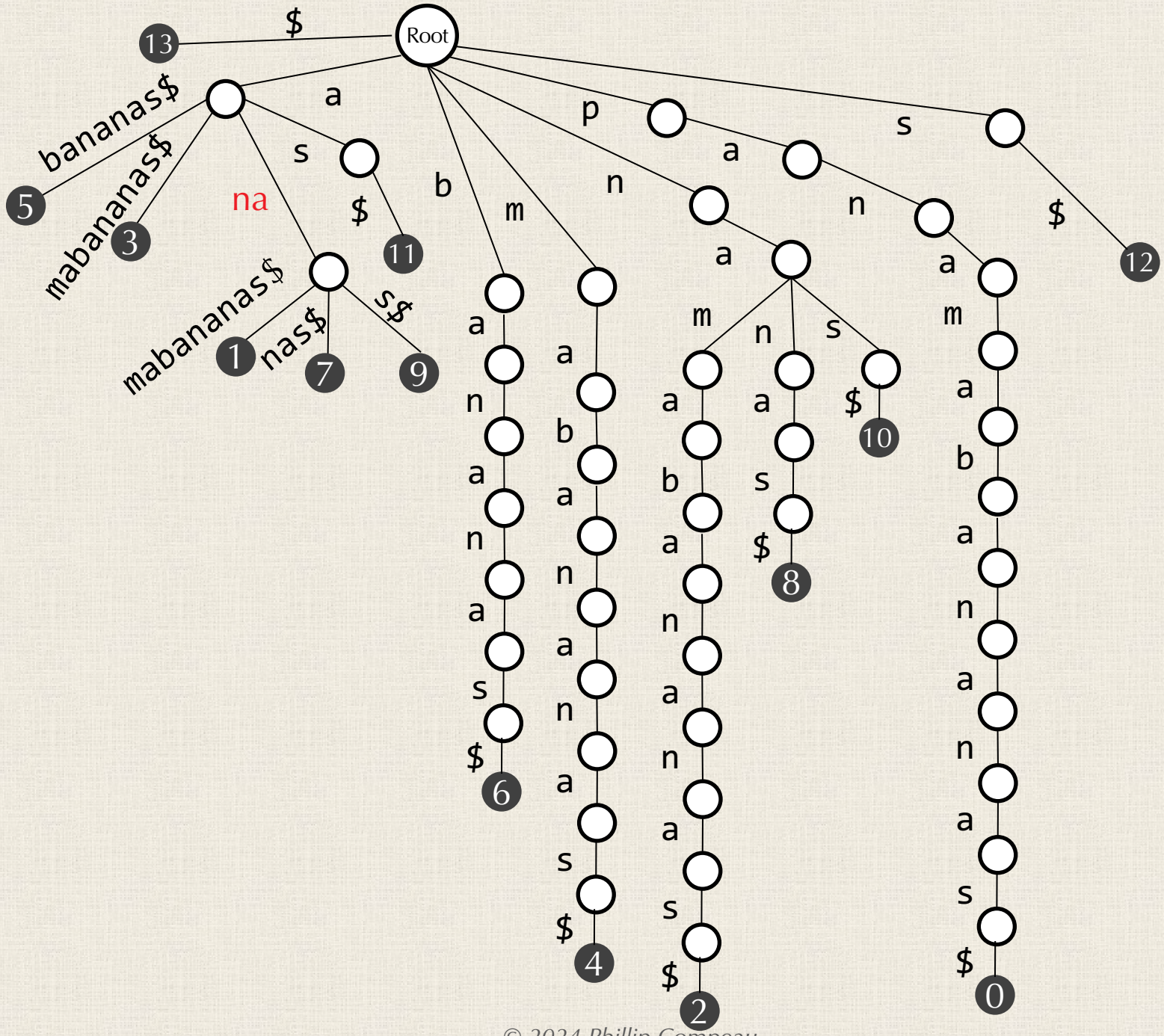


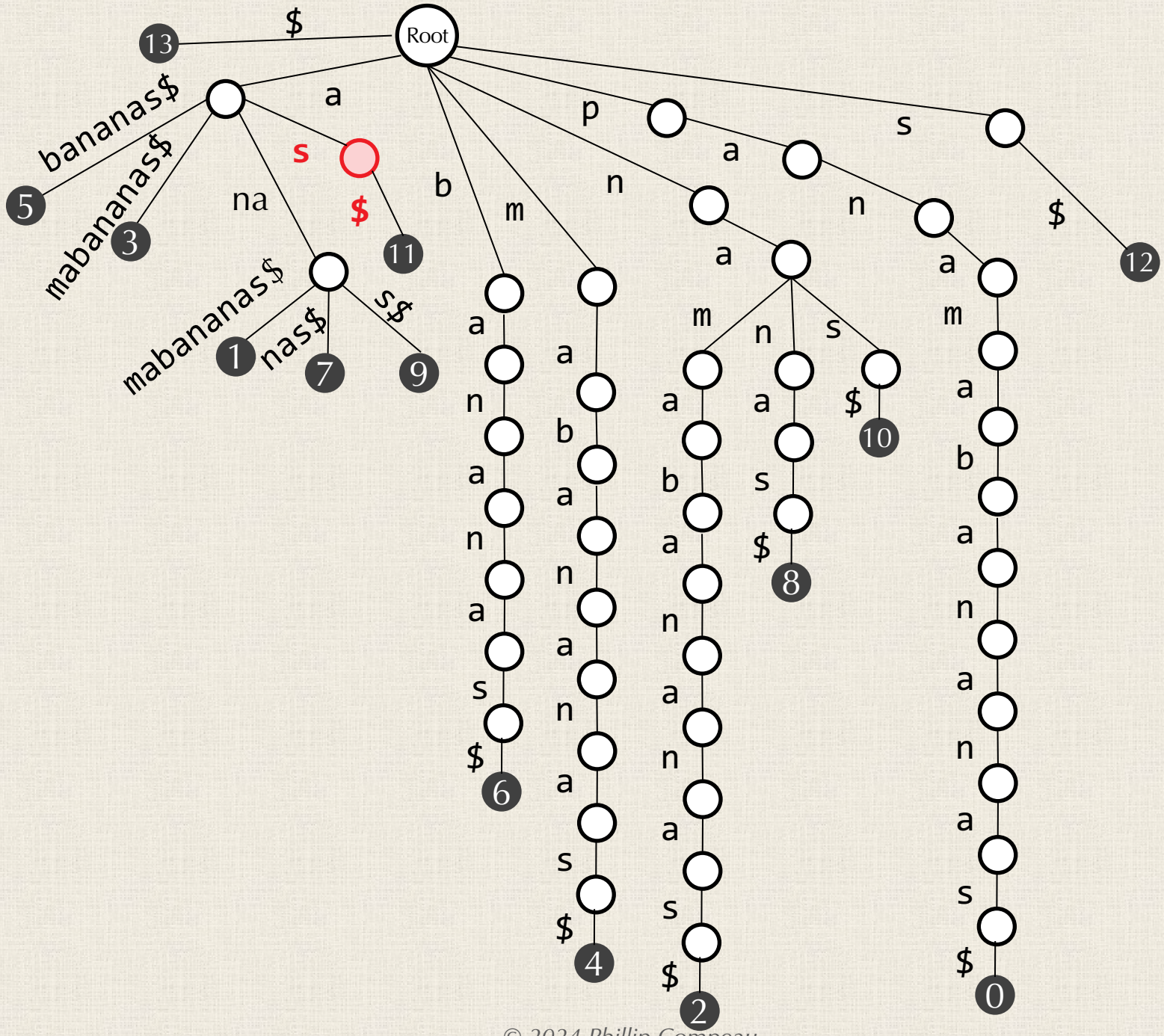


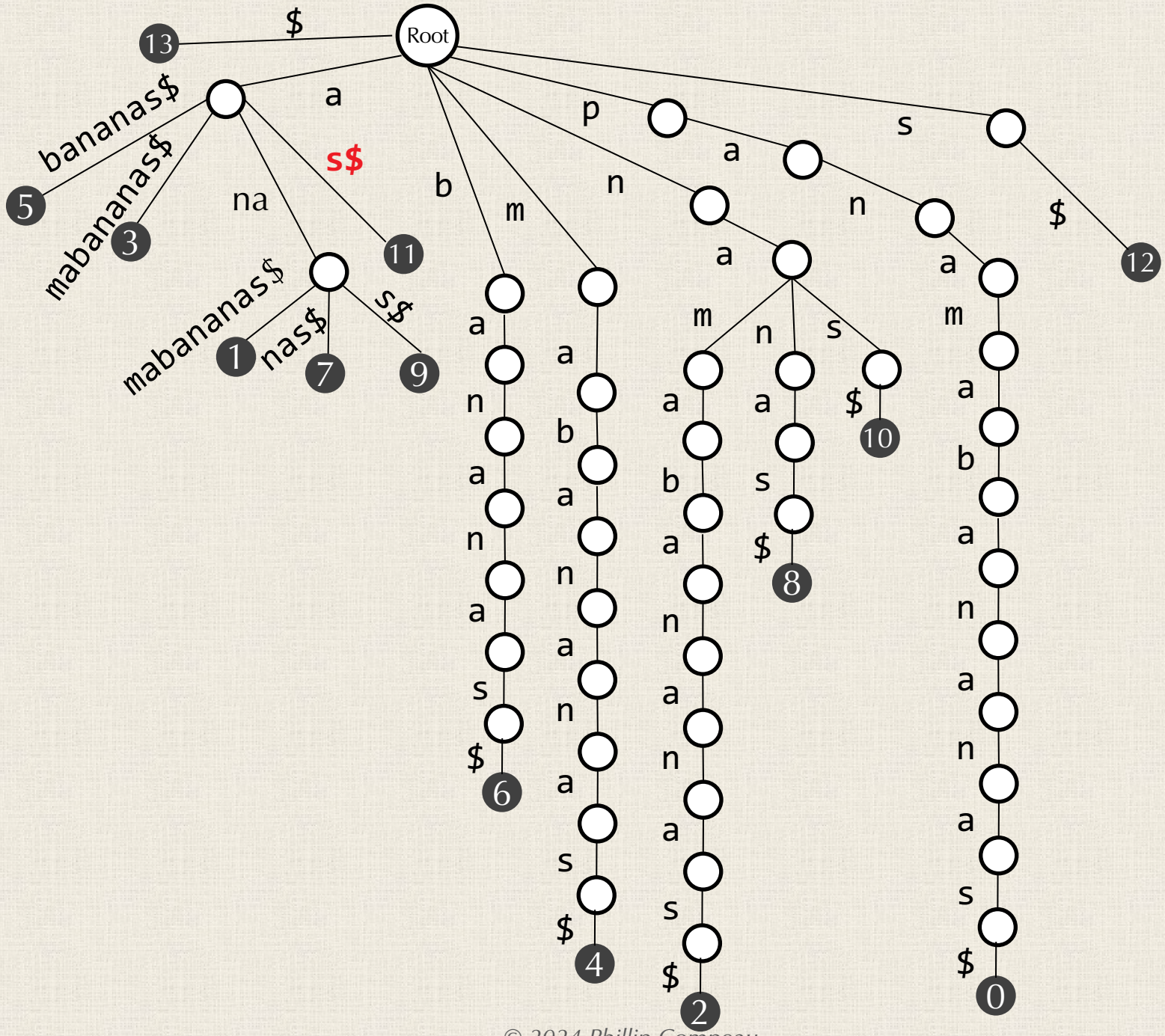


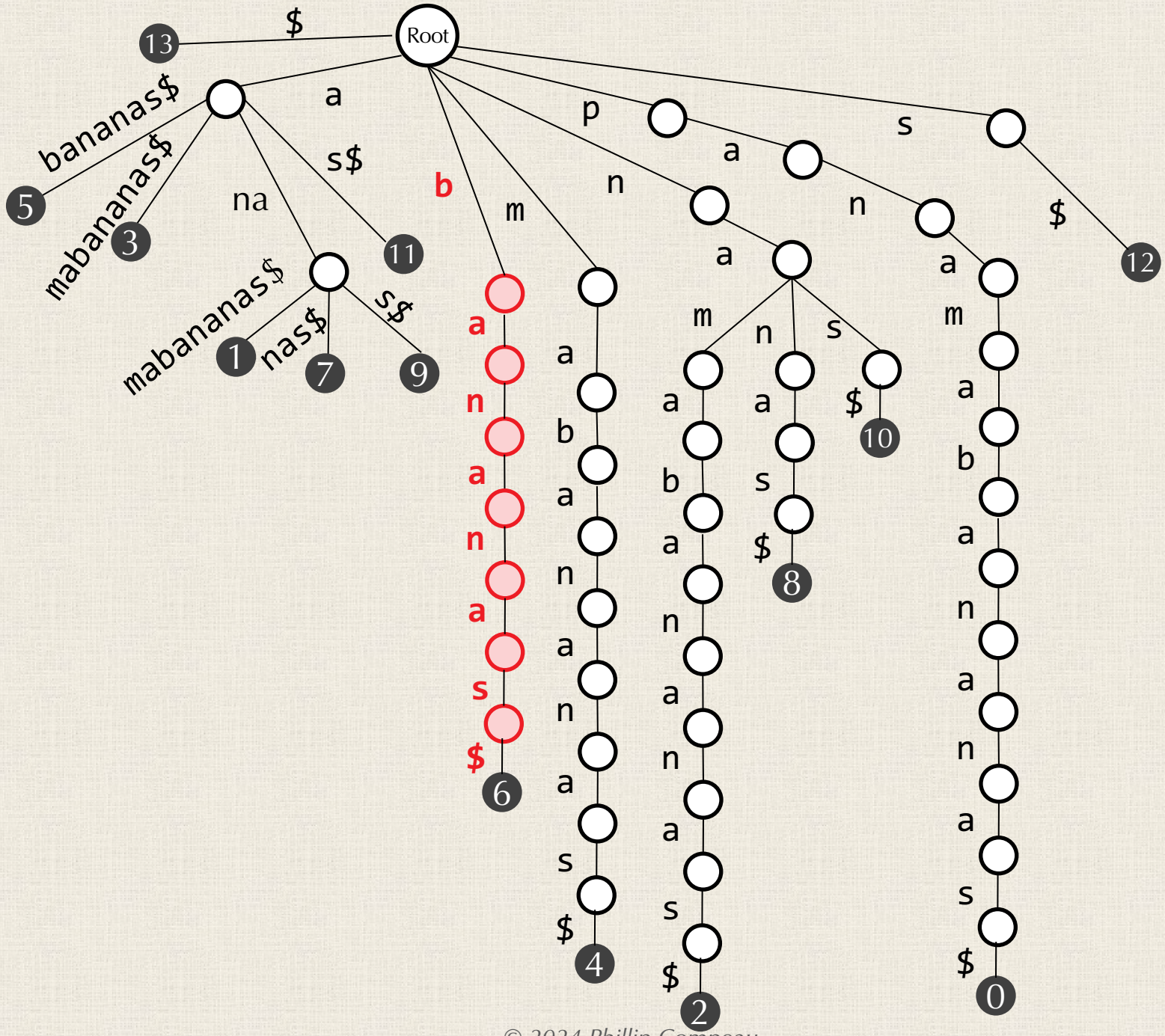


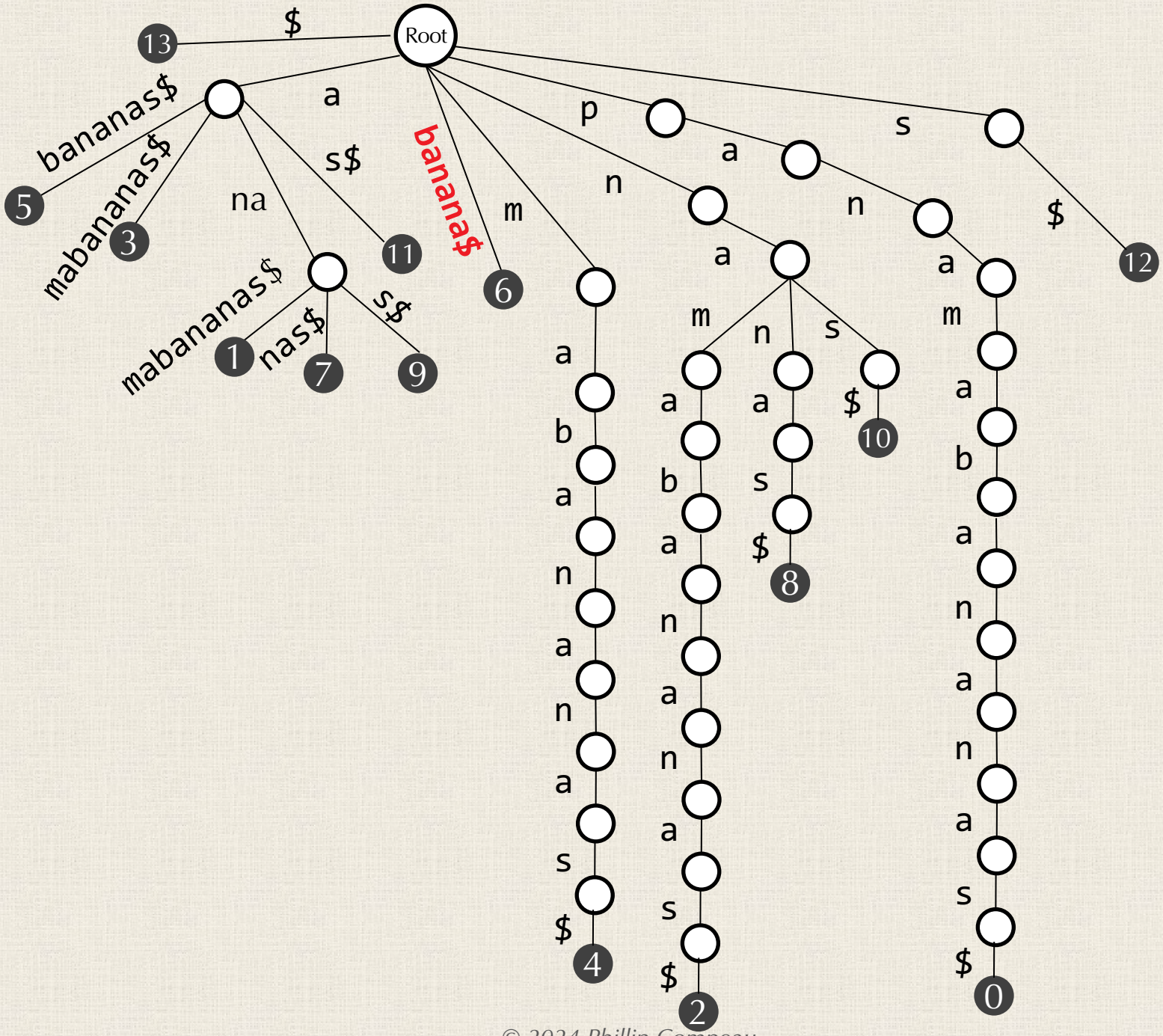


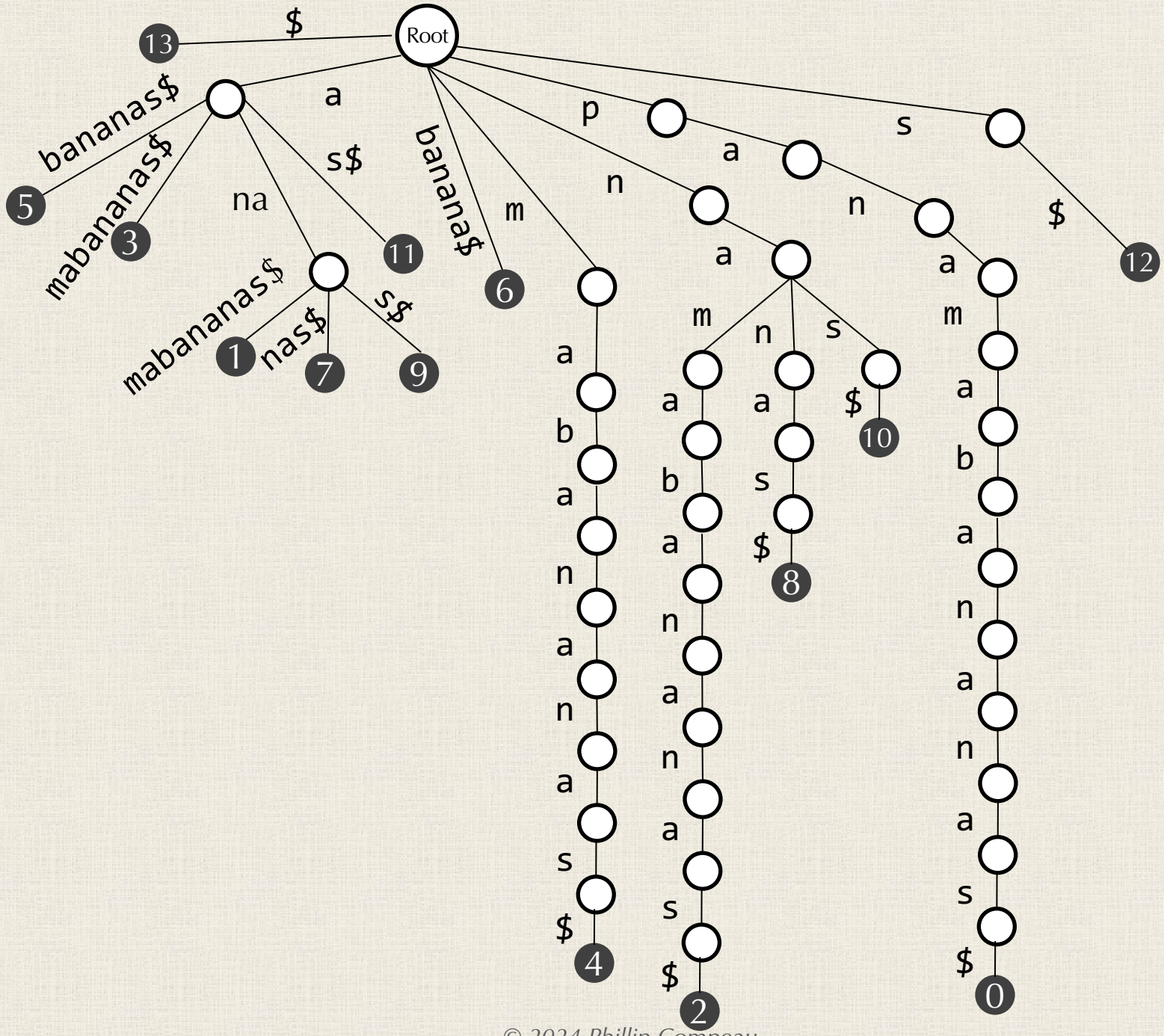


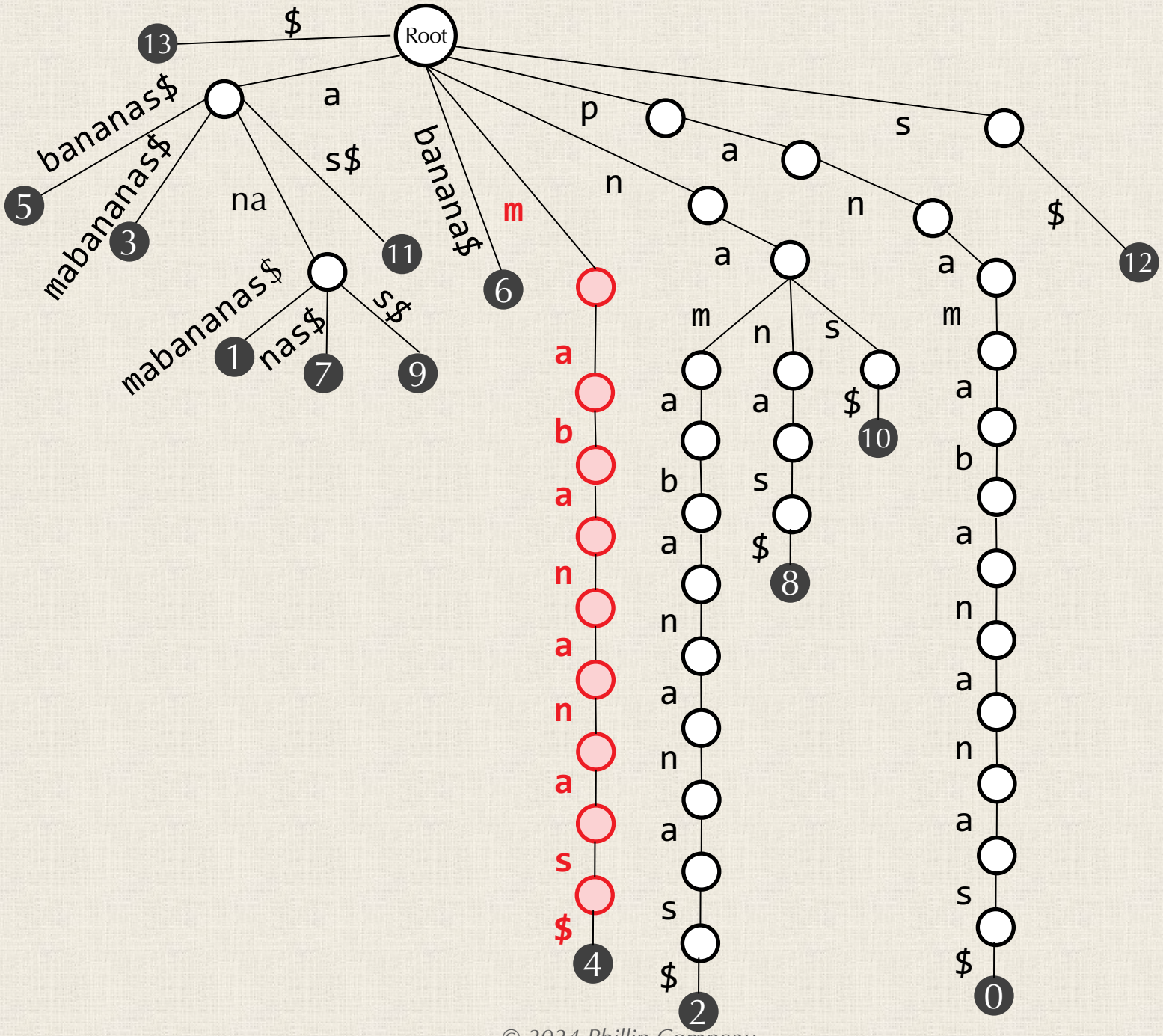


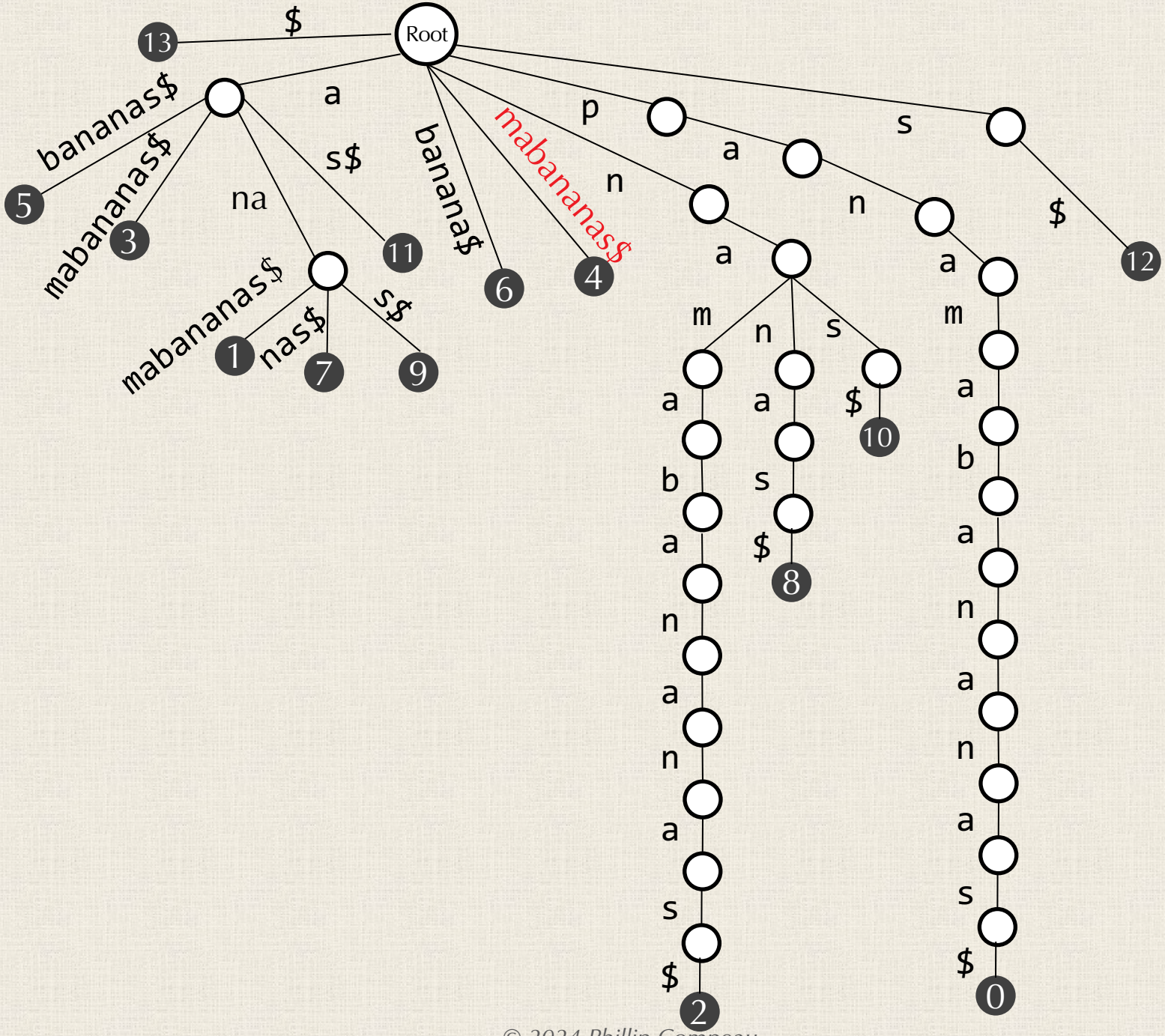


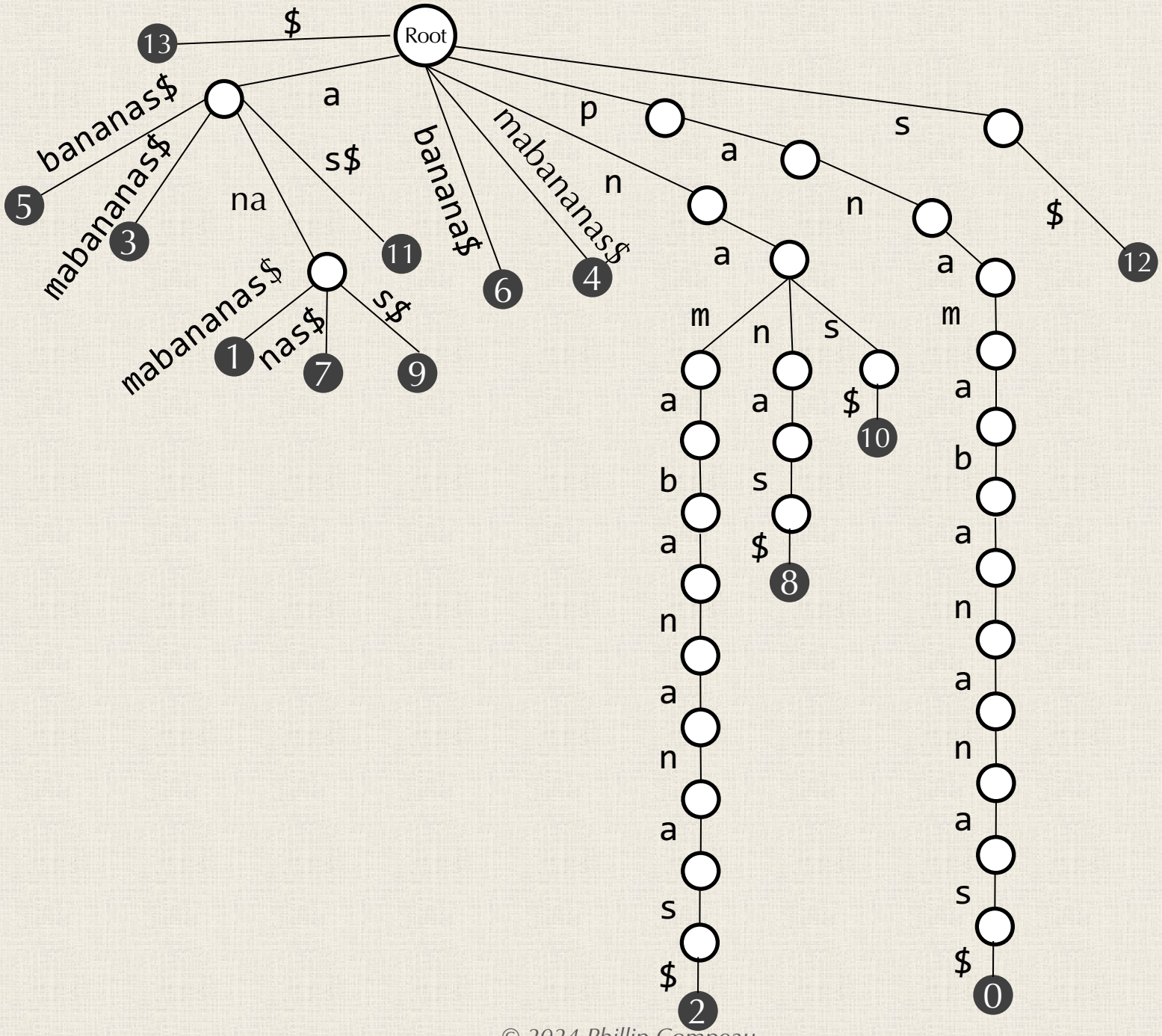


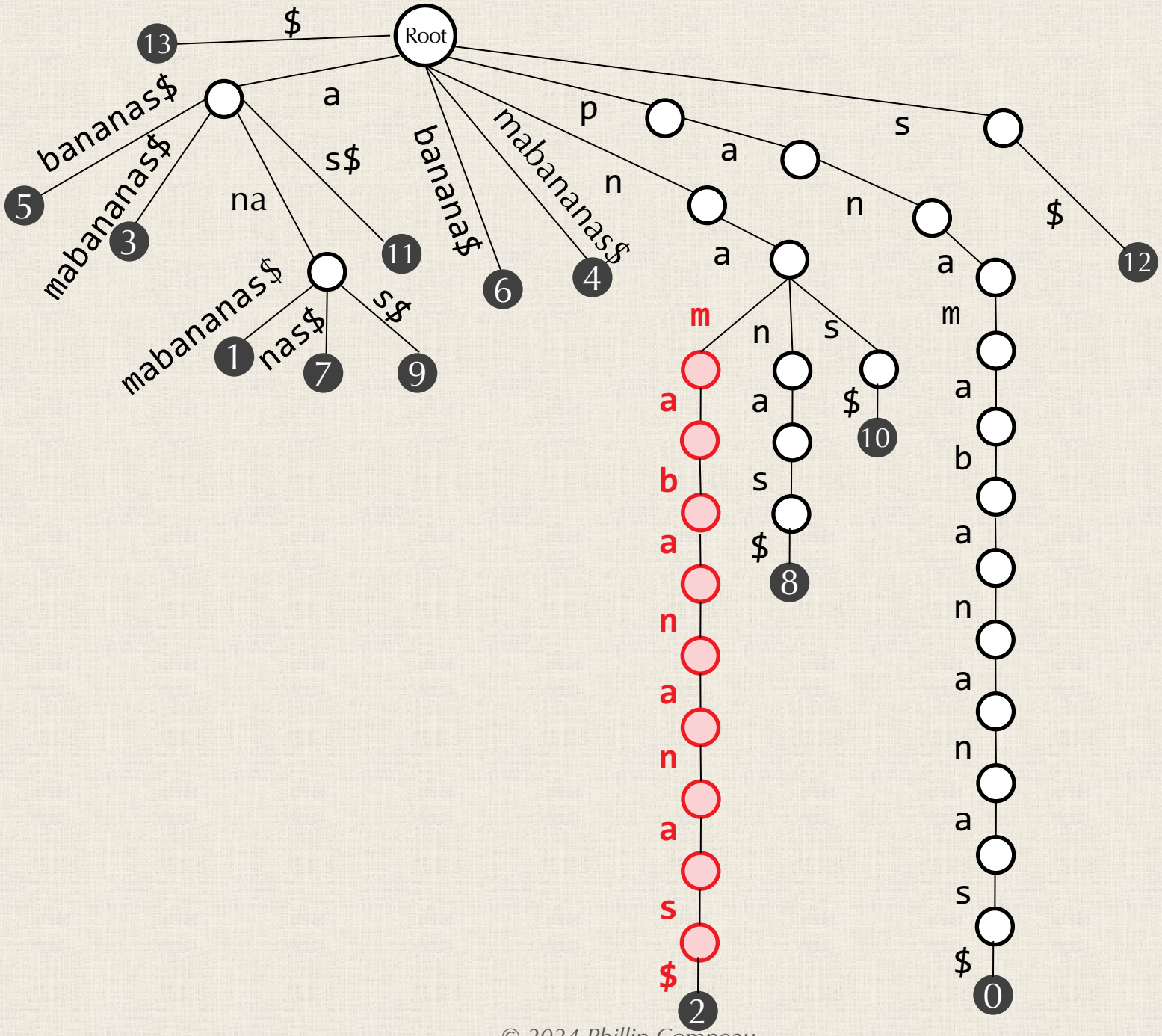


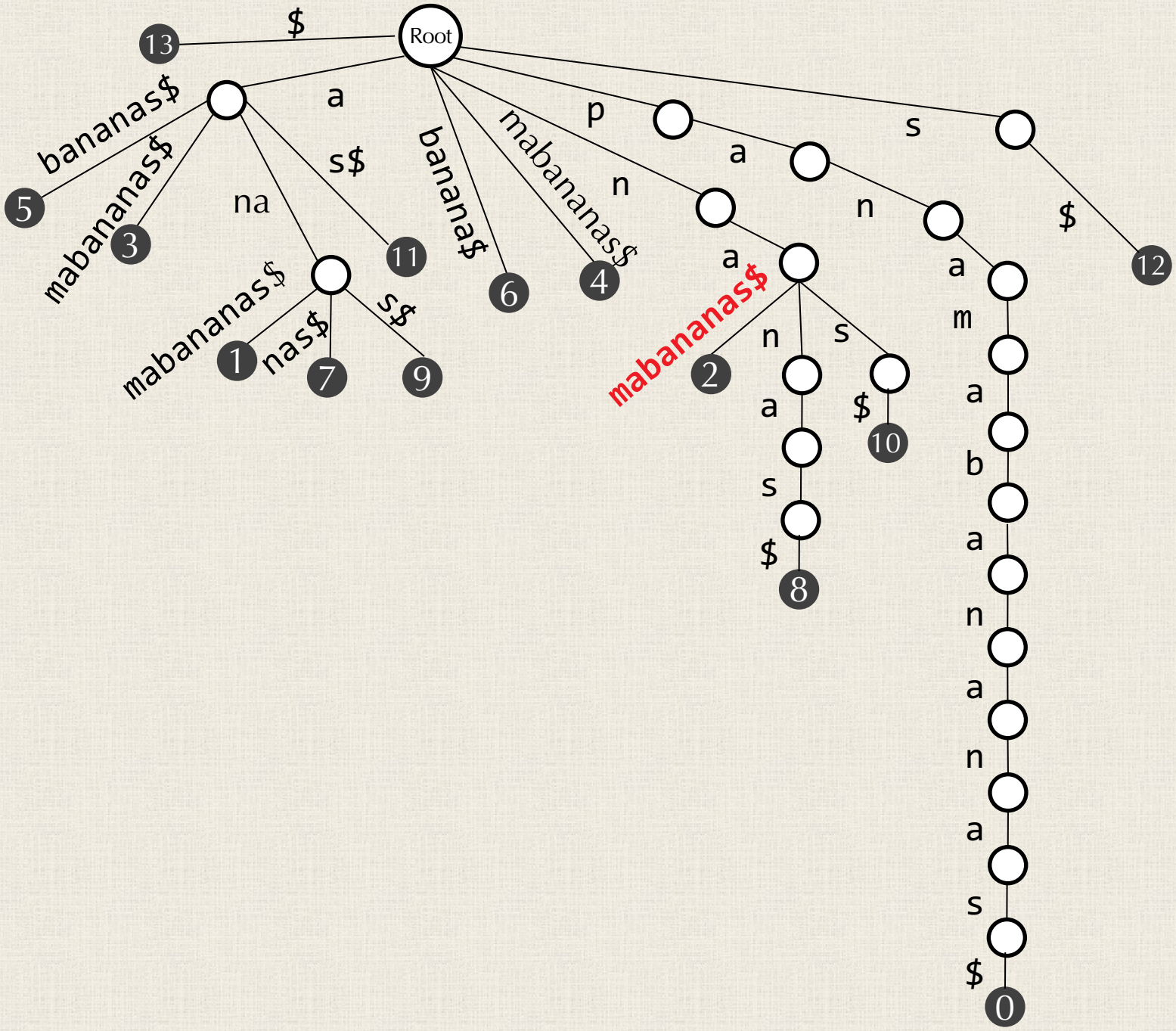


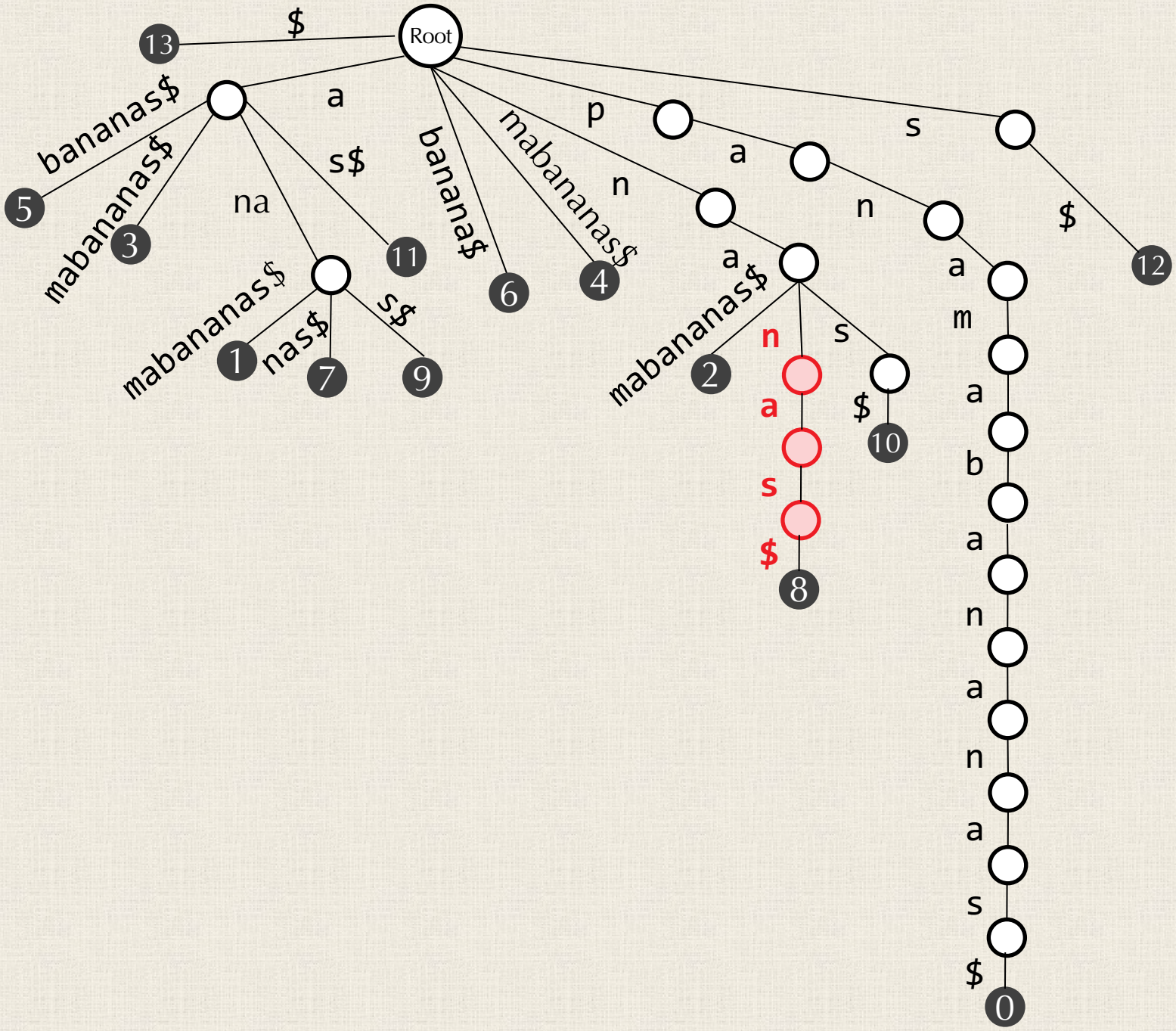


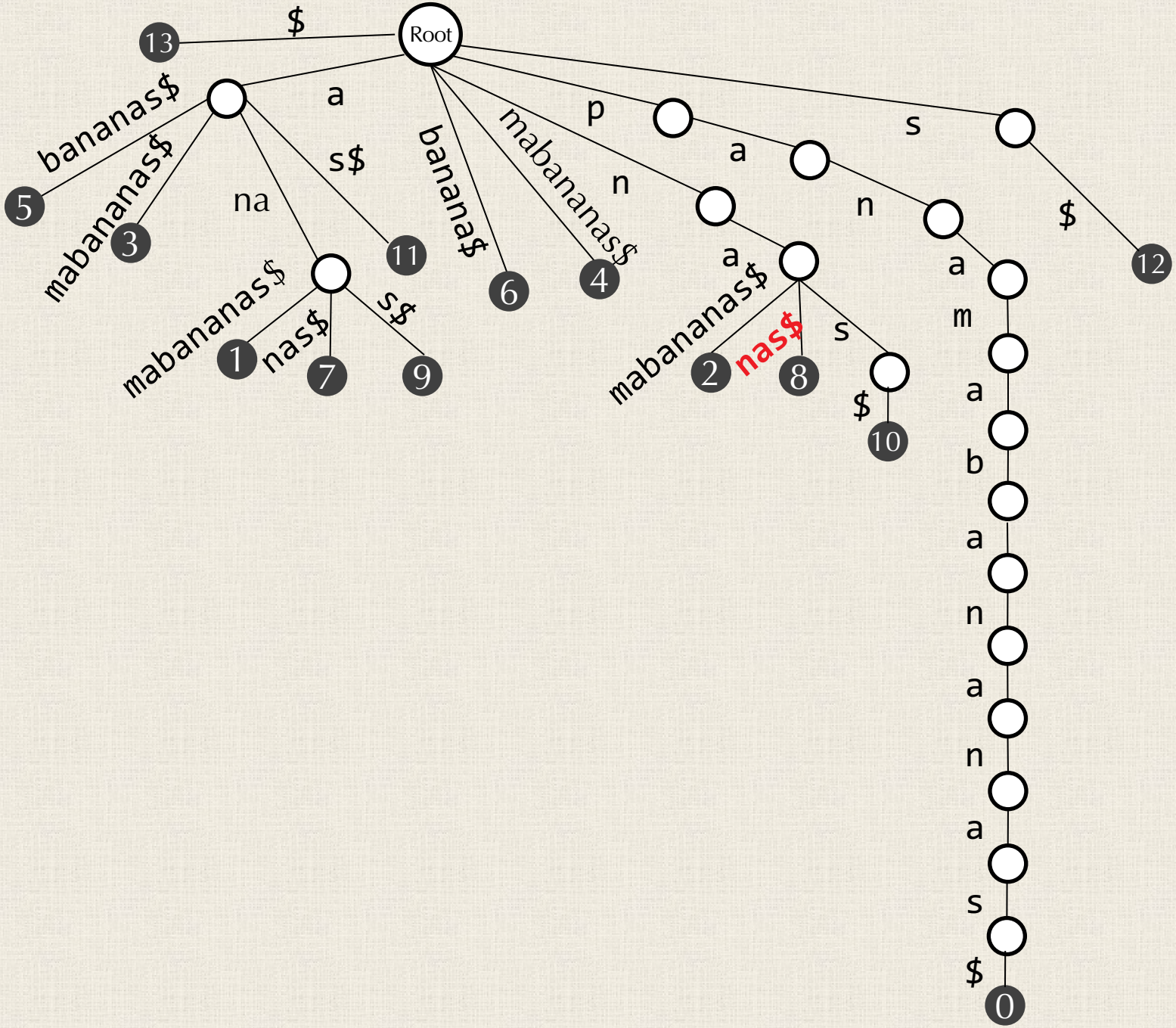


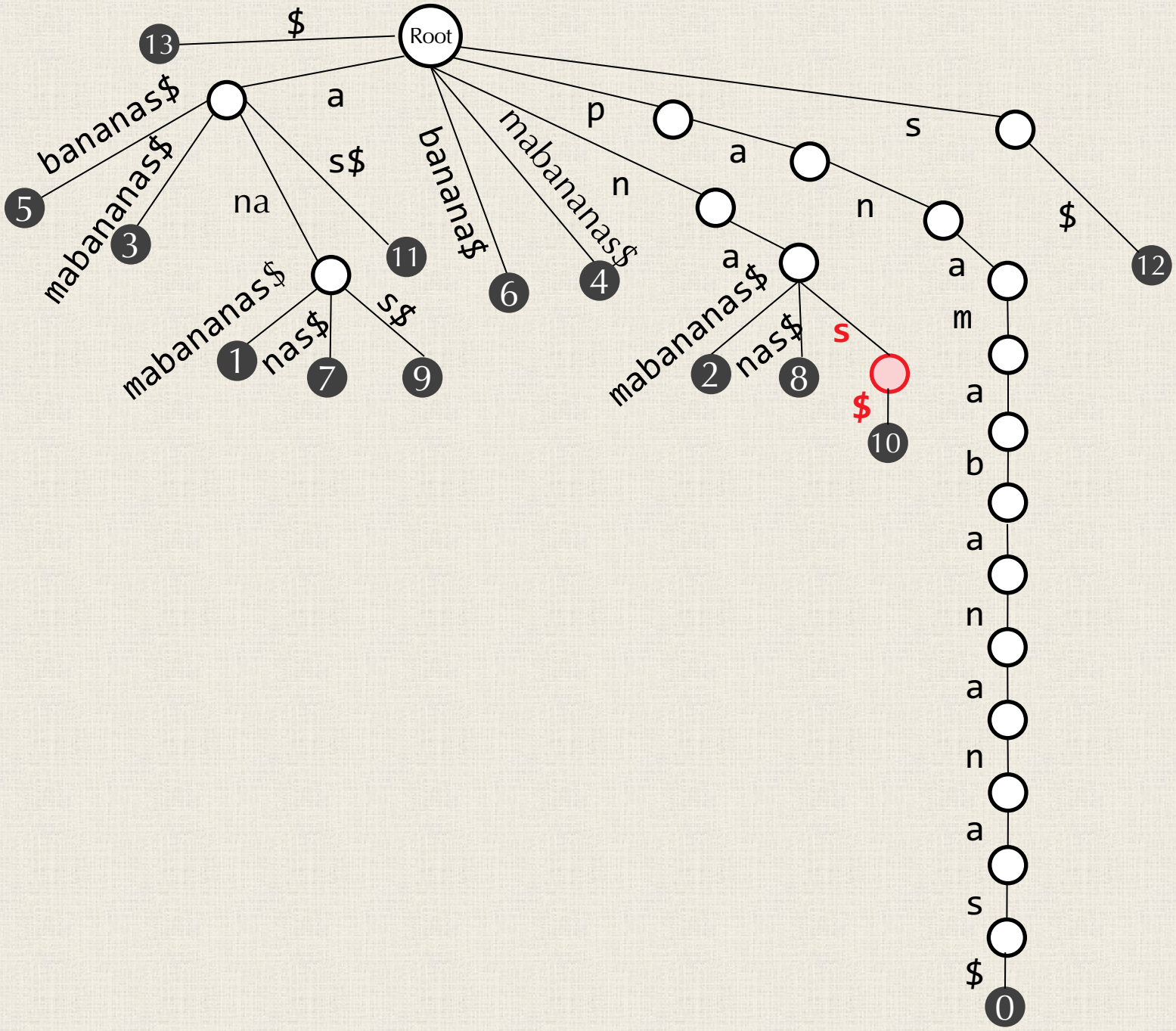


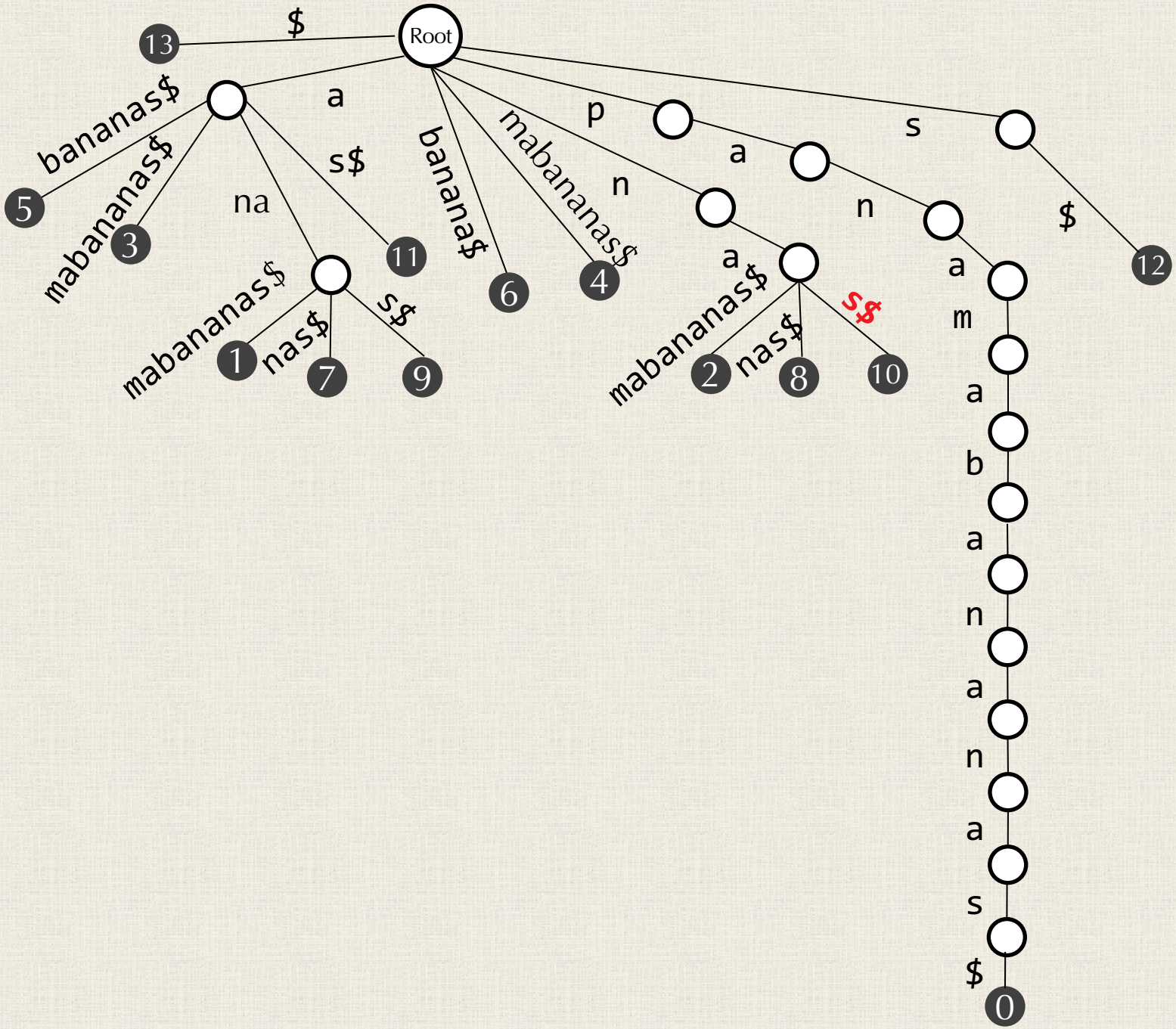


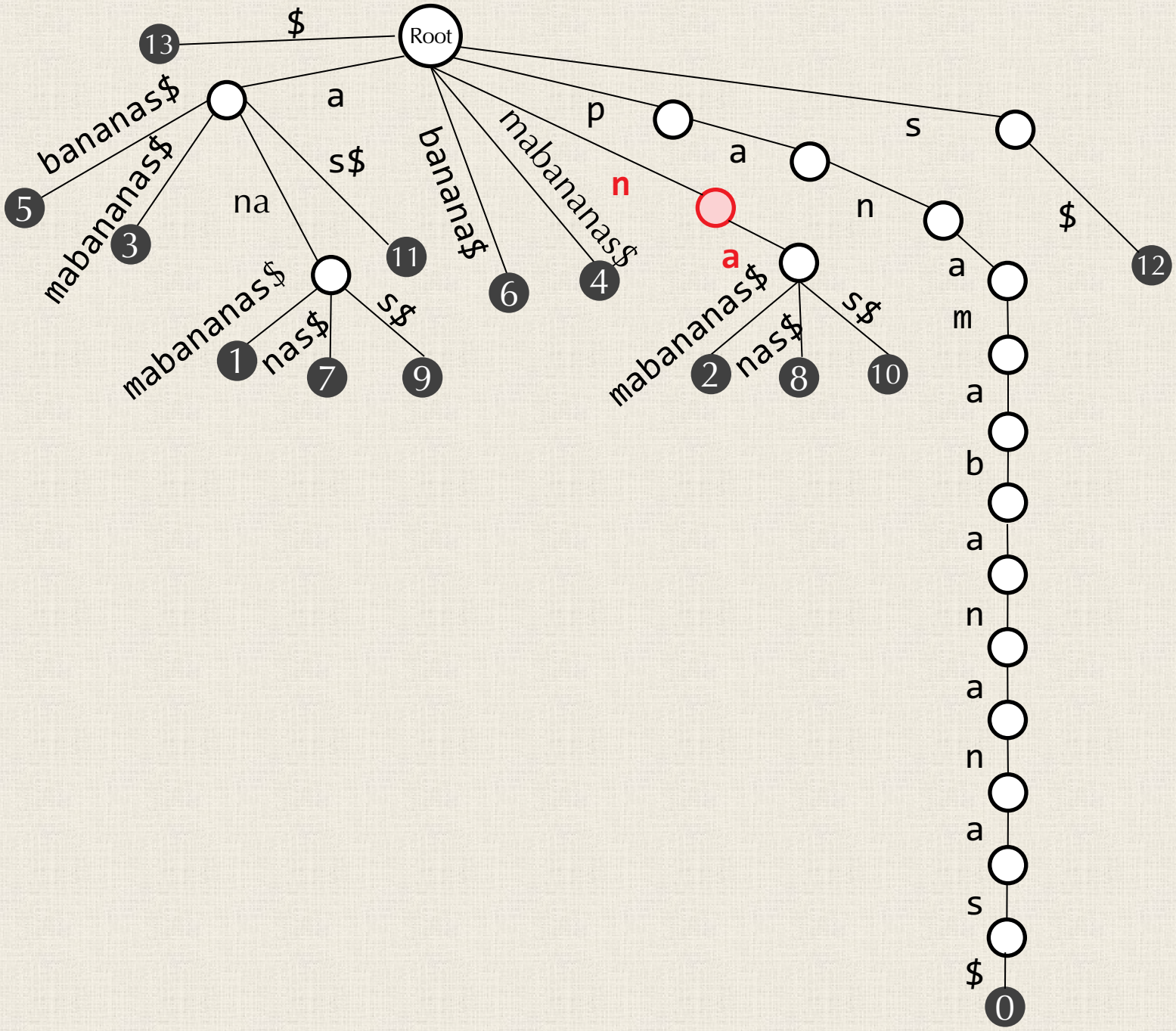


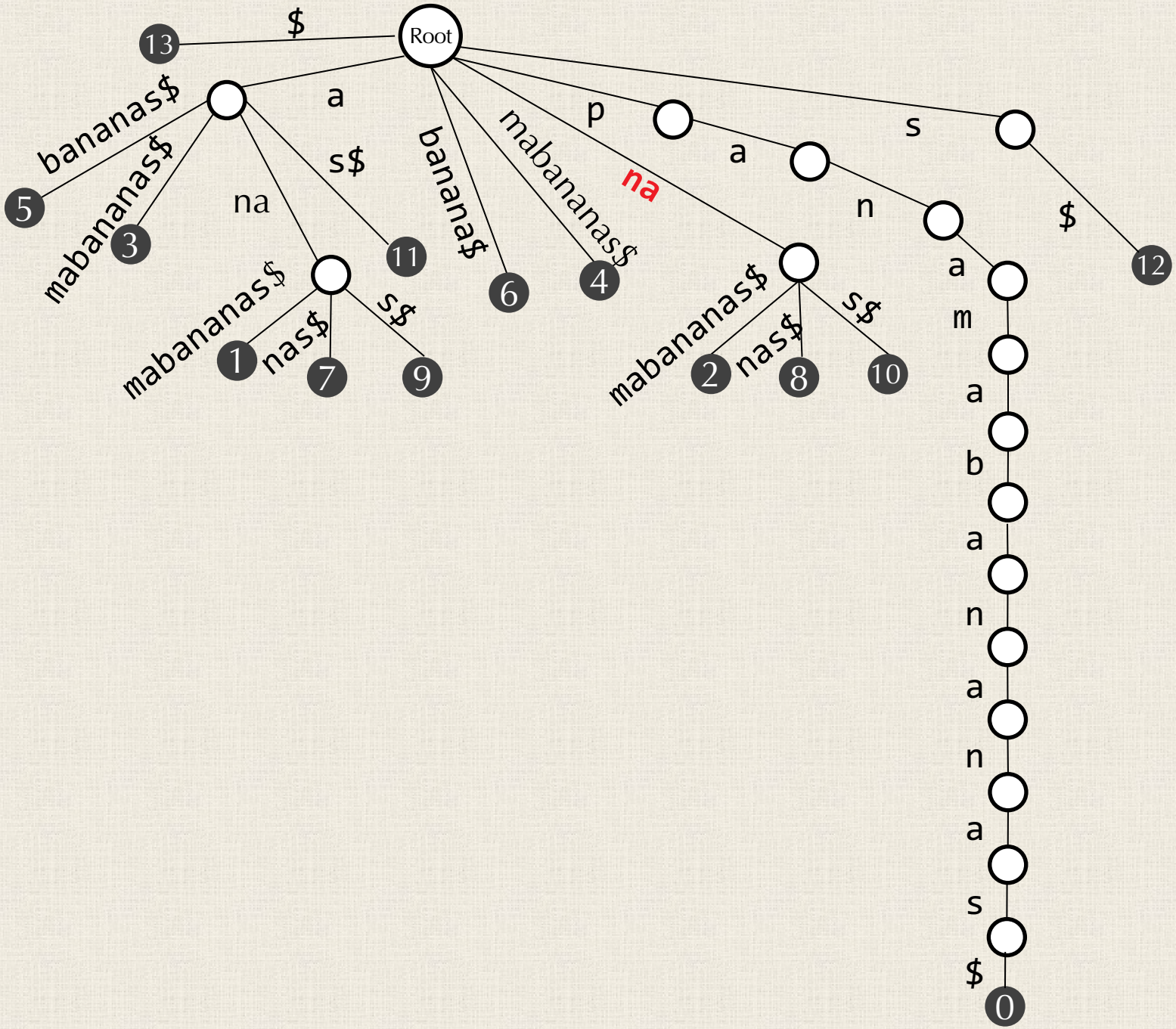


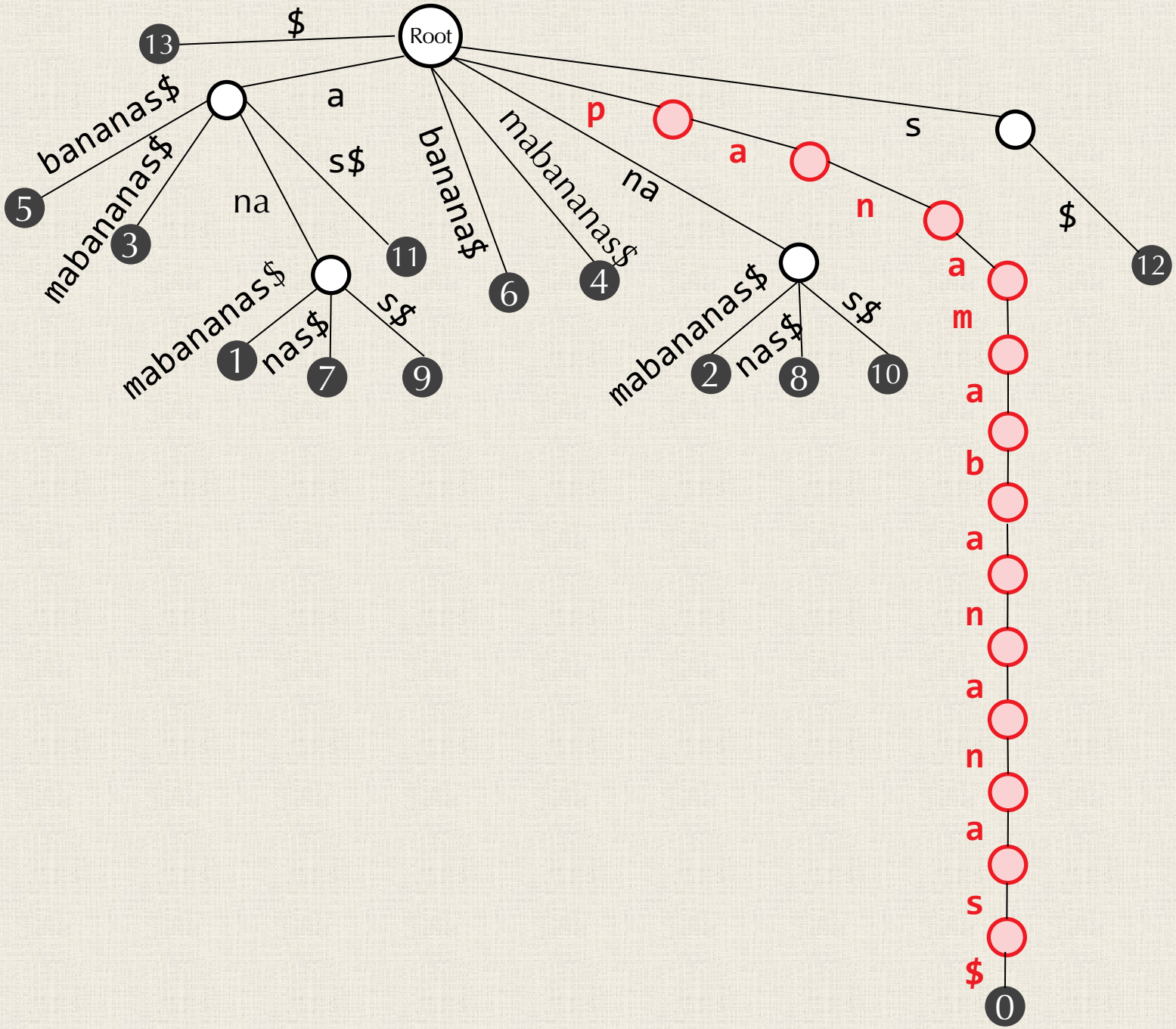


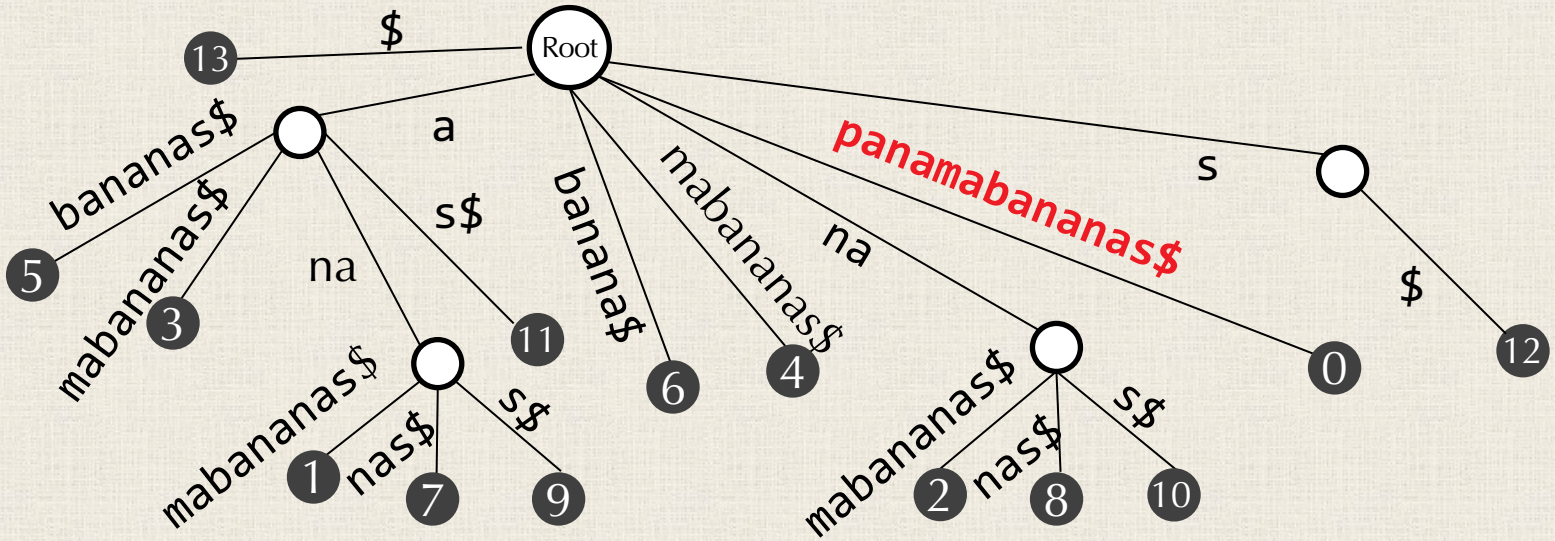


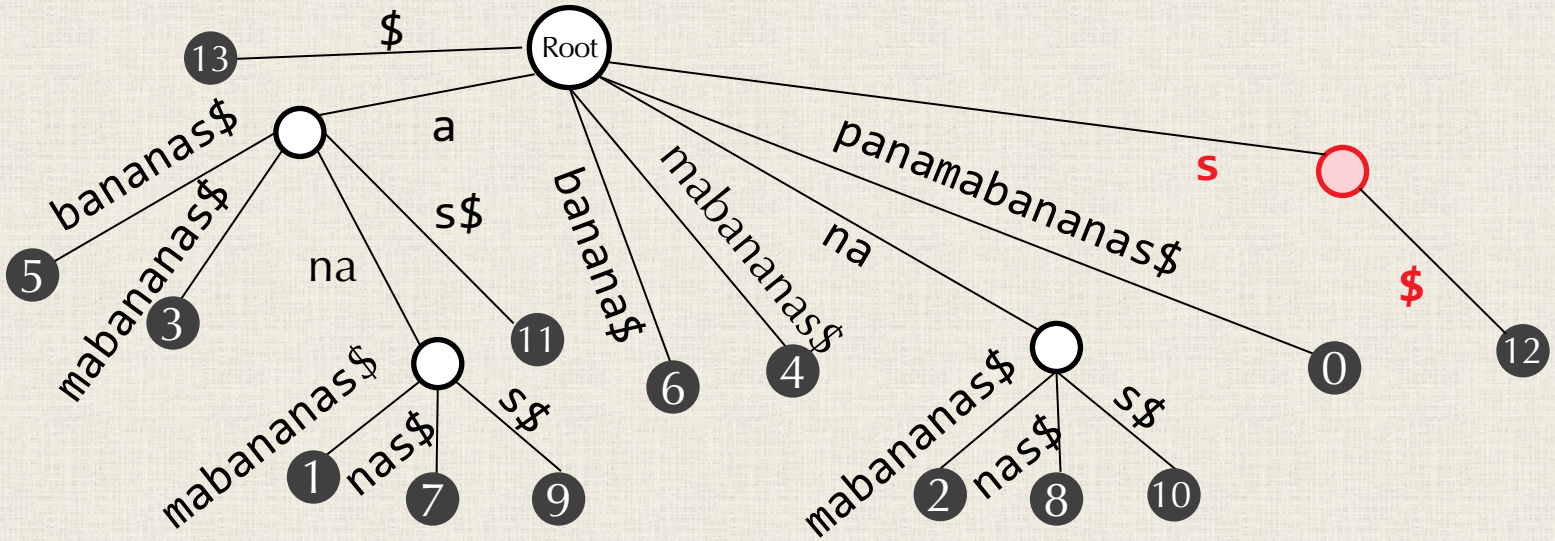


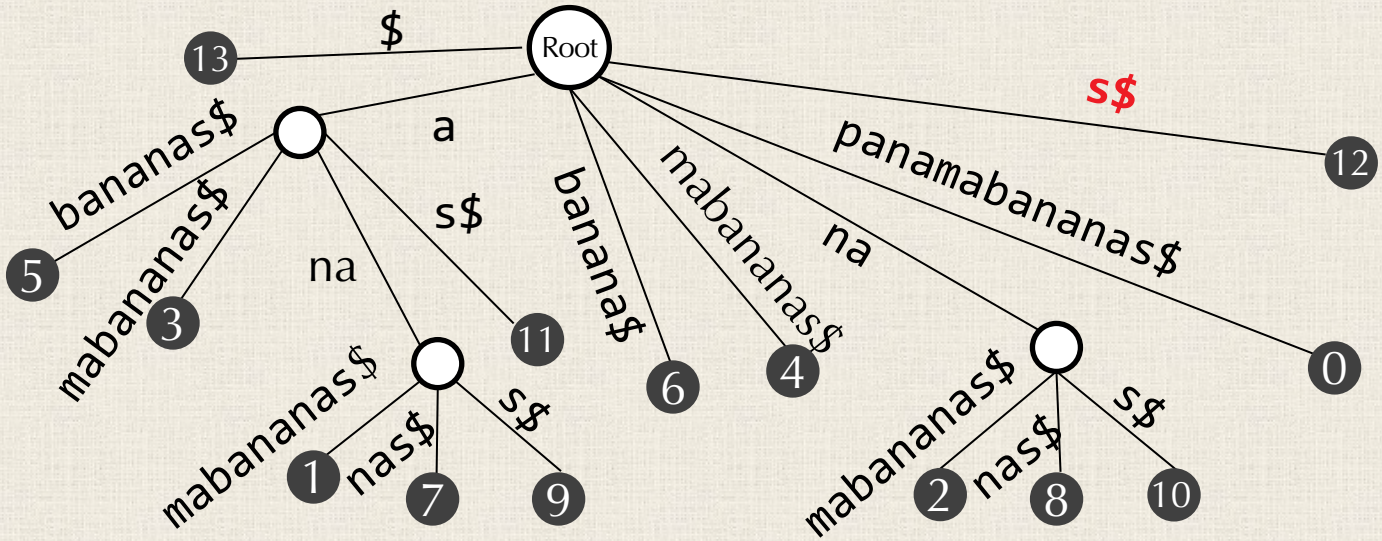


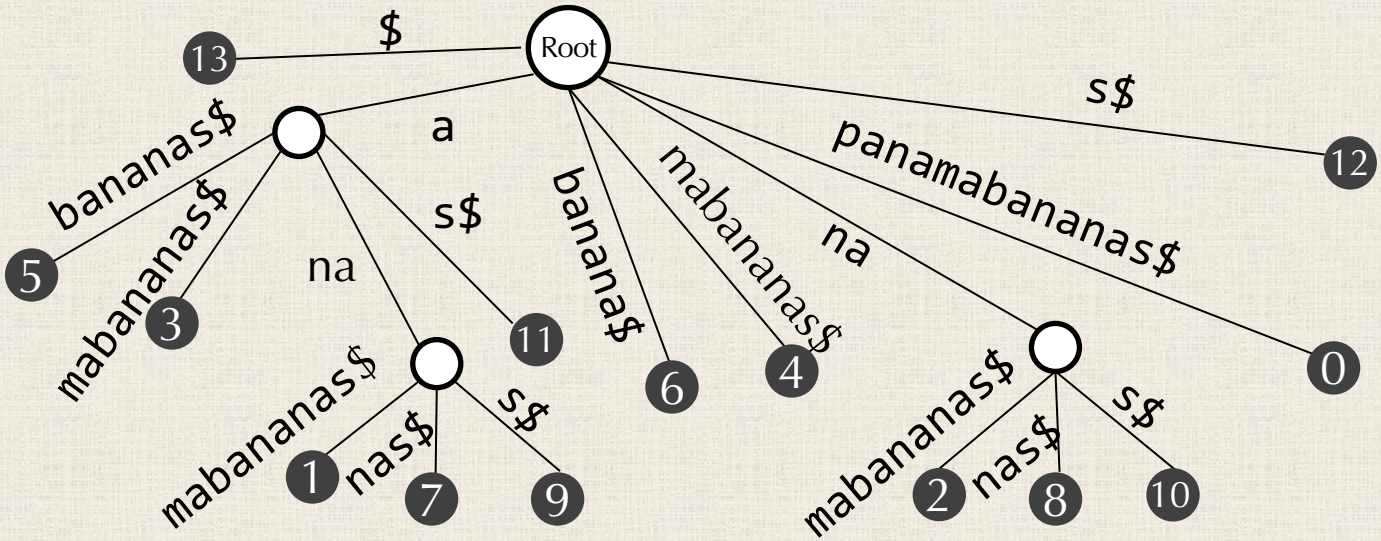








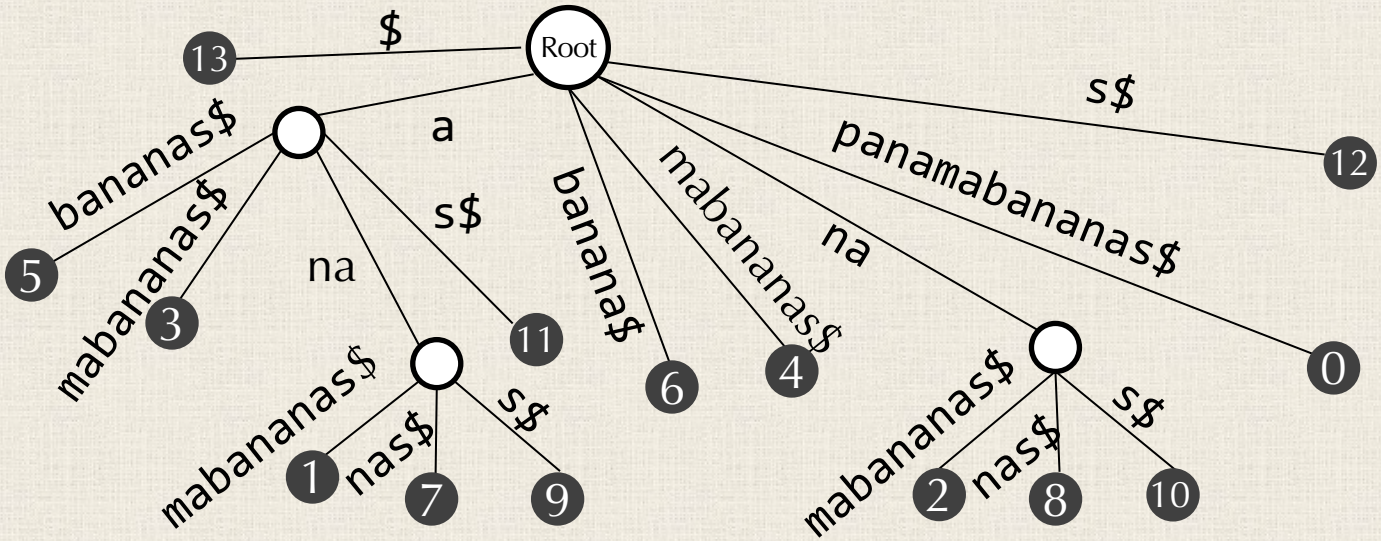




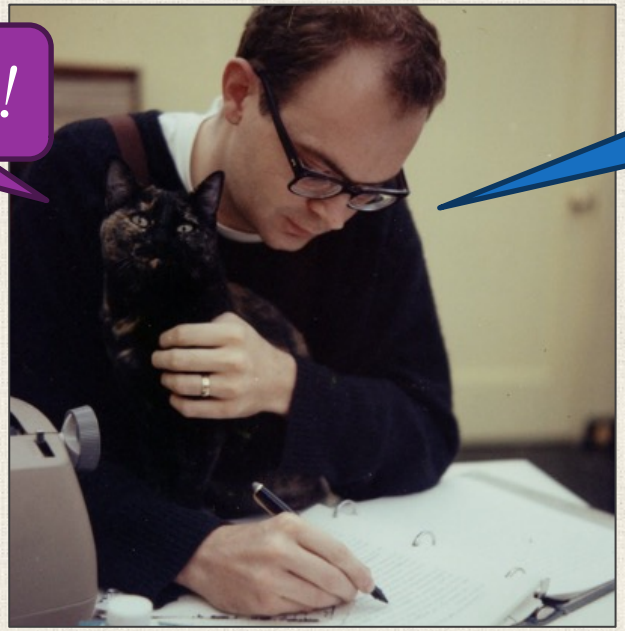
Linear pattern matching algorithms - IEEE Conference ...

by P Weiner · 1973 · Cited by 2270 · Related articles

A **linear** time **algorithm** for obtaining a compacted version of a bi-tree associated with a given **string** is presented. ... With this construction as the basic tool, we indicate how to solve several **pattern matching** problems, including some from [4] in **linear** time.

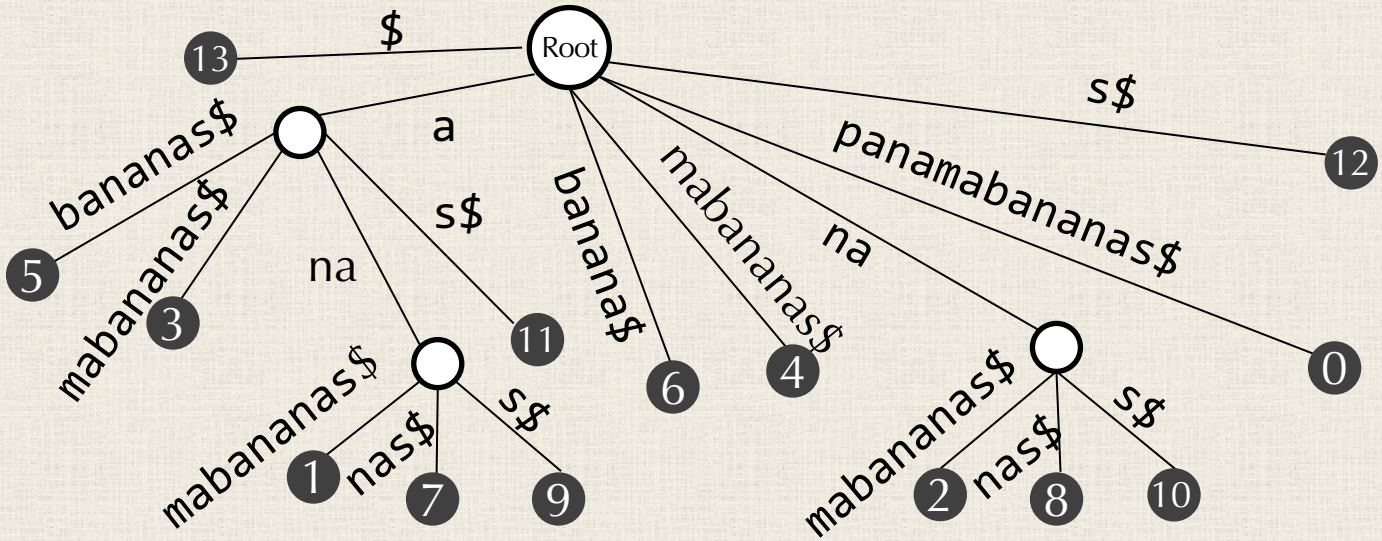


No Kitten!



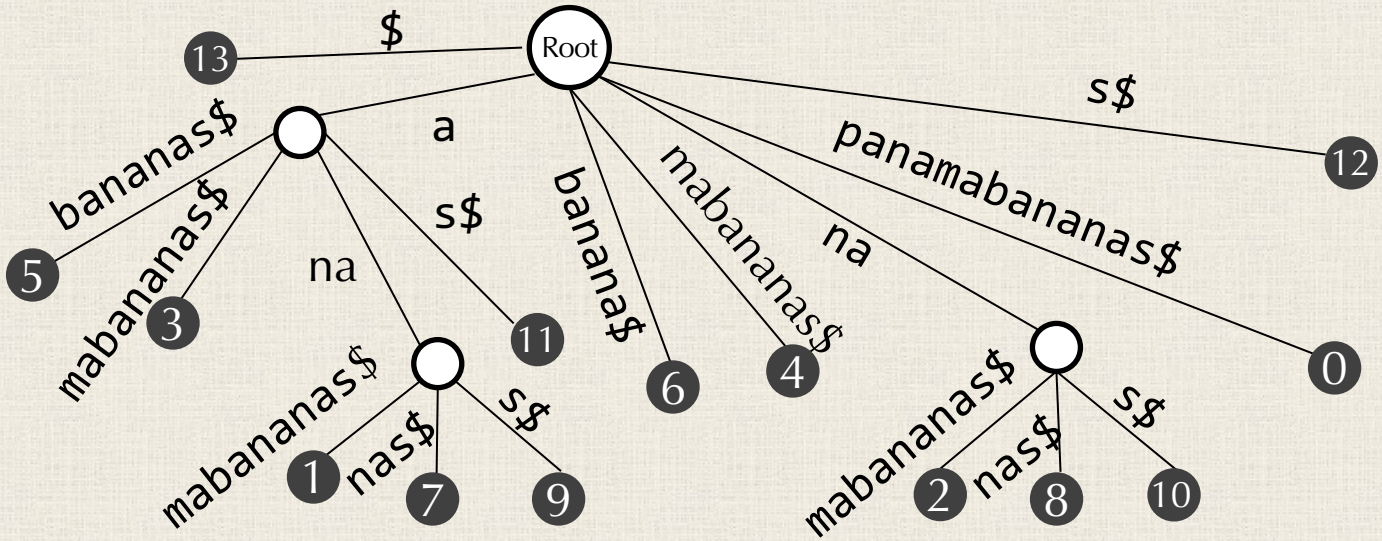
Algorithm of the Year! (1973)

https://www.nytimes.com/2018/12/17/science/donald-knuth-computers-algorithms-programming.html



Theorem: The suffix tree of *Text* has:

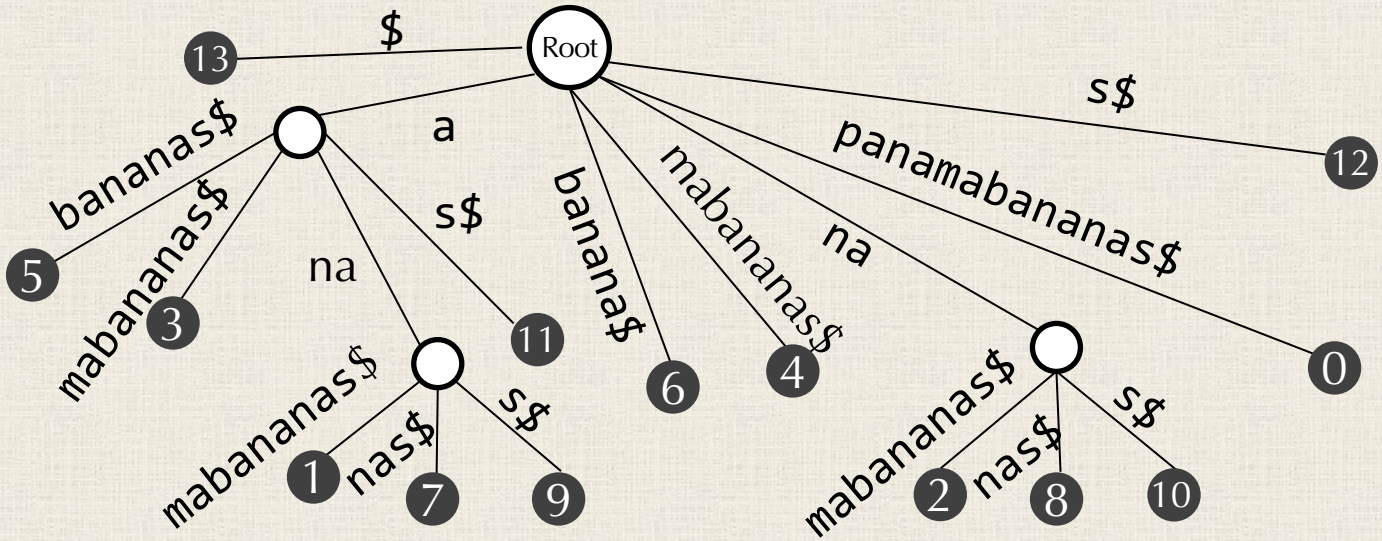
1. exactly |*Text*| + 1 leaves (degree 1)
2. at most |*Text*| internal nodes (larger degree).



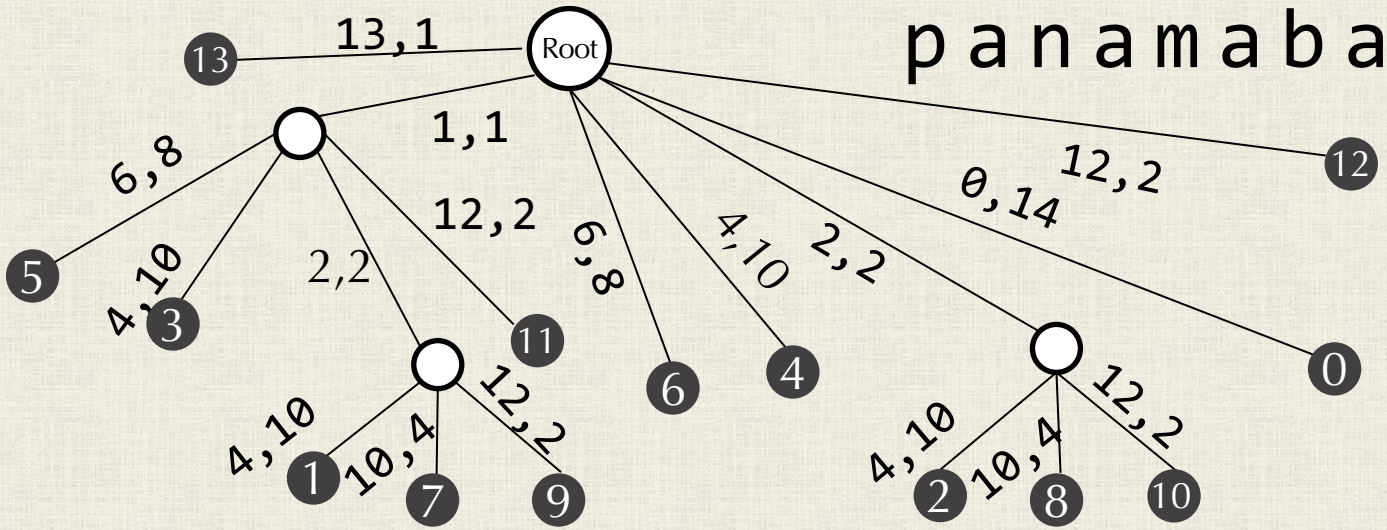
Theorem: The suffix tree of *Text* has:

1. exactly |*Text*| + 1 leaves (degree 1)
2. at most |*Text*| internal nodes (larger degree).

Thus, the suffix tree only takes O(|*Text*|) space. (It is also possible to construct the suffix tree in just O(|*Text*|) time; that's a story for another day.)



STOP: But wait ... we are still storing all the suffixes! So why are suffix trees more memory efficient than suffix tries?



STOP: But wait ... we are still storing all the suffixes! So why are suffix trees more memory efficient than suffix tries?

Answer: We can replace each edge with two integers: the starting position of the string, and its length (constant total memory).

Runtime and Memory Analysis

Runtime:

- $O(|Text|)$ to construct the suffix tree *directly*.
- $O(|Patterns|)$ to find pattern matches.
- Total: $O(|Text| + |Patterns|)$

Memory:

- $O(|Text|)$ to construct the suffix tree *directly*.
- $O(|Text|)$ to store the suffix tree.
- Total: $O(|Text|)$

From Theory to Practice

We are trained to see an algorithm with $O(|Text|)$ runtime and memory and celebrate.

From Theory to Practice

We are trained to see an algorithm with O(|*Text*|) runtime and memory and celebrate.

But a data structure with 2*|*Text*| memory is very different from one with 1000*|*Text*| memory.

From Theory to Practice

We are trained to see an algorithm with O(|*Text*|) runtime and memory and celebrate.

But a data structure with 2*|*Text*| memory is very different from one with 1000*|*Text*| memory.

It takes ~60 GB to store a suffix tree for a 3 GB human genome (4 bytes for each edge of tree). Can we do better?

SUFFIX ARRAYS

Fun Time! *The Clock Game*

https://www.youtube.com/watch?v=oc9H8bo8yg0

Suffix Array of "panamabananas$"

20 years go by...

Suffix array: Starting positions of sorted suffixes of *Text*.

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

20 years go by...

Suffix array: Starting positions of sorted suffixes of *Text*.

Can be constructed in linear time, and takes only about 12 GB for human genome.

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

STOP: How can we use the suffix array for pattern matching?

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

STOP: How can we use the suffix array for pattern matching?

Answer: Like the Clock Game, we can pick a value, obtain "higher" or "lower", and move the next value to the middle of the unsearched suffixes.

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

Answer: Like the Clock Game, we can pick a value, obtain "higher" or "lower", and move the next value to the middle of the unsearched suffixes.

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

First, we find the *first* occurrence of "ana".

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

First, we find the *first* occurrence of "ana".

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

First, we find the *first* occurrence of "ana".

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

First, we find the *first* occurrence of "ana".

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

First, we find the *first* occurrence of "ana".

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

First, we find the *first* occurrence of "ana".

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	**3**
anamabananas$	**1**
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

First, we find the *first* occurrence of "ana".

Next, we find the *last* occurrence of "ana".

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	**1**
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	**4**
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	**12**

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

First, we find the *first* occurrence of "ana".

Next, we find the *last* occurrence of "ana".

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

First, we find the *first* occurrence of "ana".

Next, we find the *last* occurrence of "ana".

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	**9**
as$	11
bananas$	6
mabananas$	**4**
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

First, we find the *first* occurrence of "ana".

Next, we find the *last* occurrence of "ana".

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	**9**
as$	**11**
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

STOP: How do we know the starting locations of these three occurrences?

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	**1**
ananas$	**7**
anas$	**9**
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

Answer: The suffix array tells us!

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	**1**
ananas$	**7**
anas$	**9**
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

STOP: What is the runtime of the binary search approach using the suffix array?

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	**1**
ananas$	**7**
anas$	**9**
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Suffix Array of "panamabananas$"

This approach, shown for *Pattern* = "ana", is called **binary search**.

Answer: Each pattern takes O(log(|*Text*|)).

Sorted Suffixes	Suffix Array
$	13
abananas$	5
amabananas$	3
anamabananas$	**1**
ananas$	**7**
anas$	**9**
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

The Binary Search Pseudocode that 90% of Programmers Get Wrong

```
PatternMatchingSuffixArray(Text, Pattern)
        s ← SuffixArray(Text)
        minIndex ← 0
        maxIndex ← |Text| - 1
        while minIndex <= maxIndex
                midIndex ← floor((minIndex + maxIndex)/2)
                if Pattern > suffix of Text starting at s(midIndex)
                        minIndex ← midIndex + 1
                else
                        maxIndex ← midIndex - 1
        if Pattern matches suffix of Text starting at s(midIndex)
                first ← minIndex
        else
                return no occurrences
        minIndex ← first
        maxIndex ← |Text| - 1
        while minIndex <= maxIndex
                midIndex ← floor((minIndex + maxIndex)/2)
                if Pattern matches suffix of Text starting at s(midIndex)
                        minIndex ← midIndex + 1
                else
                        maxIndex ← midIndex – 1
        last ← maxIndex
        return (first, last)
```

Suffix Arrays are Another Example of Computational Biologists Influencing CS

Suffix arrays: a new method for on-line string searches

U Manber, G Myers - siam Journal on Computing, 1993 - SIAM

A new and conceptually simple data structure, called a suffix array, for on-line string searches is introduced in this paper. Constructing and querying suffix arrays is reduced to a sort and search paradigm that employs novel algorithms. The main advantage of suffix arrays over suffix trees is that, in practice, they use three to five times less space. From a complexity standpoint, suffix arrays permit on-line string searches of the type,"Is W a substring of A?" to be answered in time O(P+\logN), where P is the length of W and N is the …

☆ 99 Cited by 2870 Related articles All 43 versions »

Note: This is a paper produced by a computational biologist (and future head of search for Google) in a non-biological context seven years before the publication of the first draft human genome.

THE BURROWS-WHEELER TRANSFORM

Idea to Reduce Memory: Compress the Genome

Run-length encoding: compresses every run of *n* identical symbols X into the substring *"n*X*"*.

Text

GGGGGGGGGGCCCCCCCCCCCAAAAAAATTTTTTTTTTTTTTTTCCCCCG

10G11C7A15T5C1G

Run-length encoding

Idea to Reduce Memory: Compress the Genome

Run-length encoding: compresses every run of *n* identical symbols X into the substring "*n*X".

Text

GGGGGGGGGGCCCCCCCCCCCAAAAAAATTTTTTTTTTTTTTTTCCCCCG

10G11C7A15T5C1G

Run-length encoding

Problem: Genomes don't have lots of runs…

Idea to Reduce Memory: Compress the Genome

Run-length encoding: compresses every run of *n* identical symbols X into the substring *"nX"*.

Text

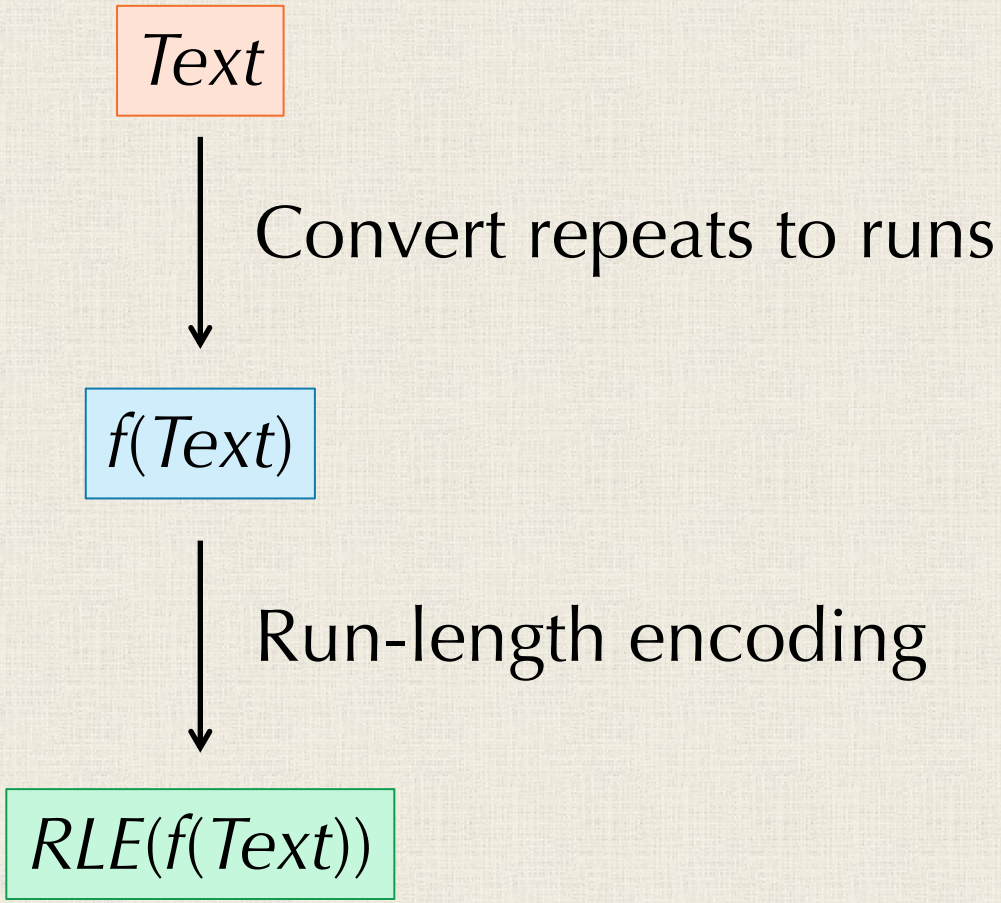
GGGGGGGGGGCCCCCCCCCCCAAAAAAATTTTTTTTTTTTTTTTCCCCCG

10G11C7A15T5C1G

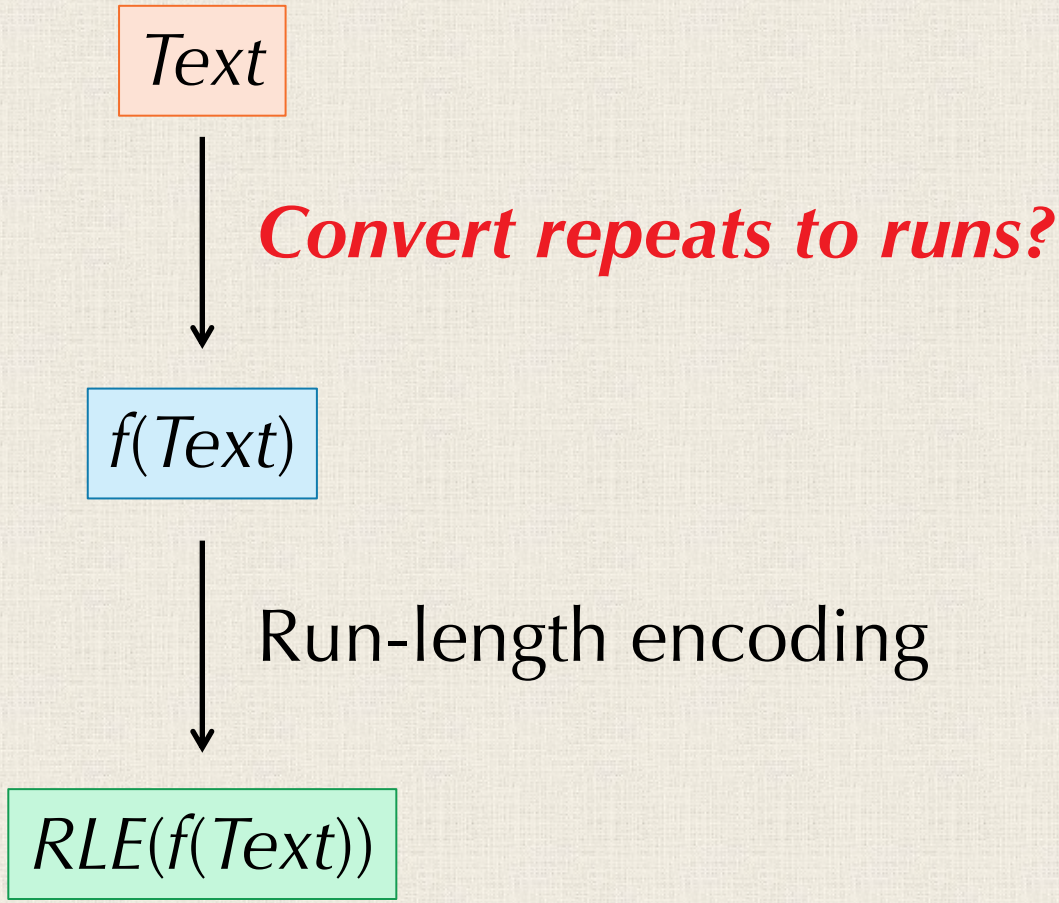
Run-length encoding

…but they do have a lot of repeats!

Converting Repeats to Runs?



Converting Repeats to Runs?



Converting Repeats to Runs?

One way we could convert repeats into runs would be to simply sort *Text* lexicographically.

Converting Repeats to Runs?

STOP: What is wrong with sorting the string?

Converting Repeats to Runs?

STOP: What is wrong with sorting the string?

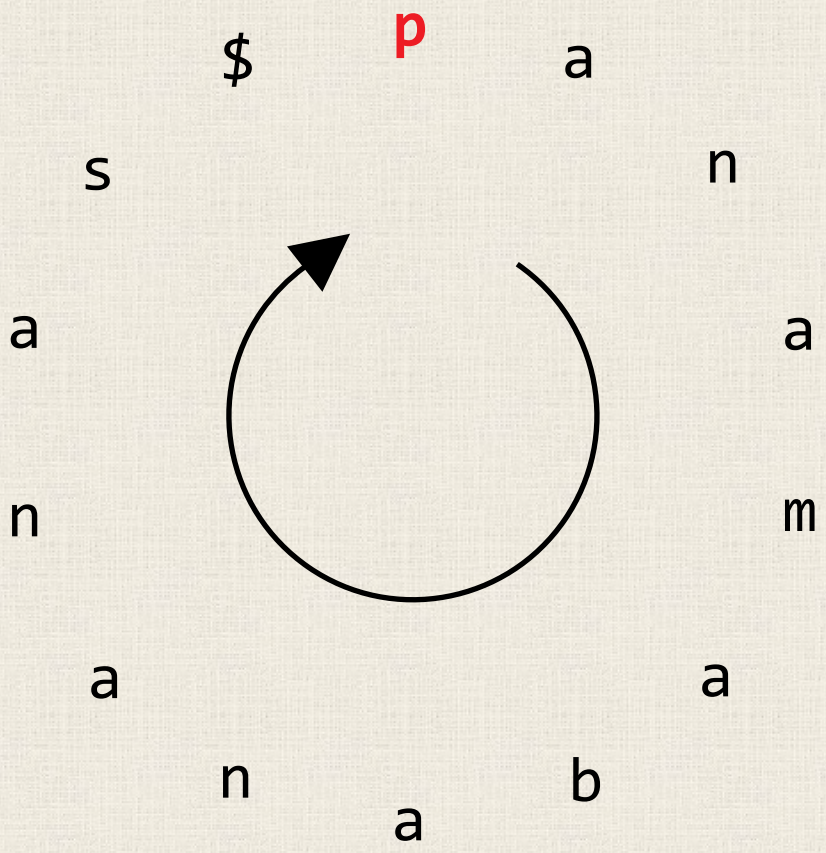
Answer: There is no way to "invert" the compression to recover *Text*. That is, *f* should be an injection!

Converting Repeats to Runs?

Formally, define $S_{A,\,n}$ as the set of all strings of length n+1, with a "$" and n symbols taken from an alphabet A. We want a function f: $S_{A,\,n} \rightarrow S_{A,\,n}$ that is **invertible:** if $f(x) = f(y)$, then $x = y$.

The **B**urrows-**W**heeler **T**ransform

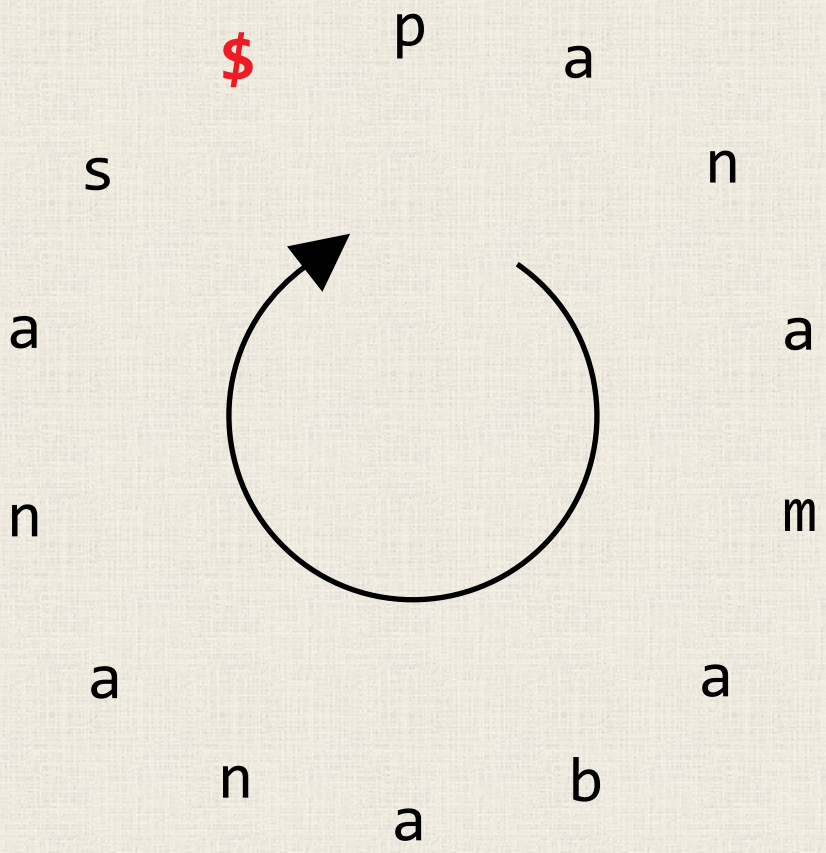
panamabananas$



Form all cyclic rotations of
“panamabananas$”

The Burrows-Wheeler Transform

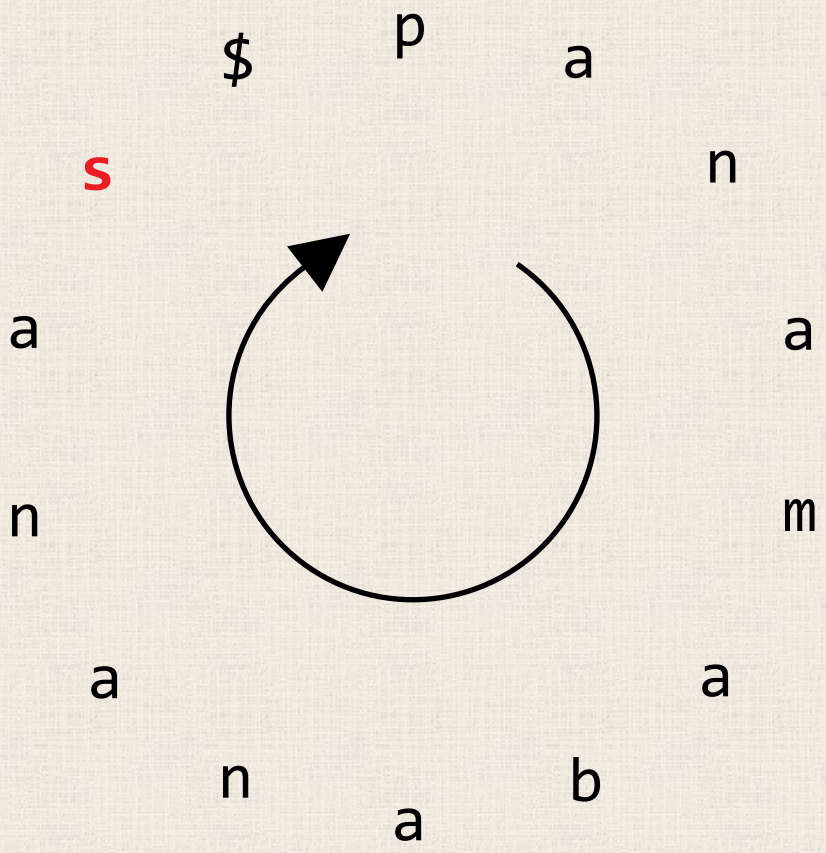
```
panamabananas$
$panamabananas
```



Form all cyclic rotations of
“panamabananas$”

The Burrows-Wheeler Transform

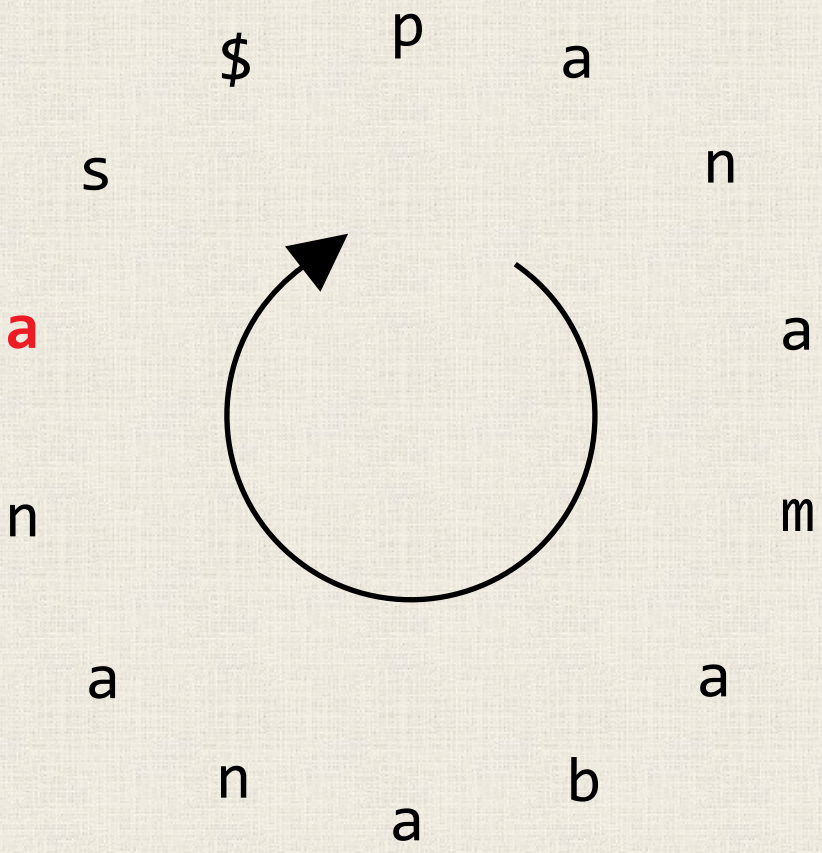
```
panamabananas$
$panamabananas
s$panamabanana
```



Form all cyclic rotations of
“panamabananas$”

The Burrows-Wheeler Transform

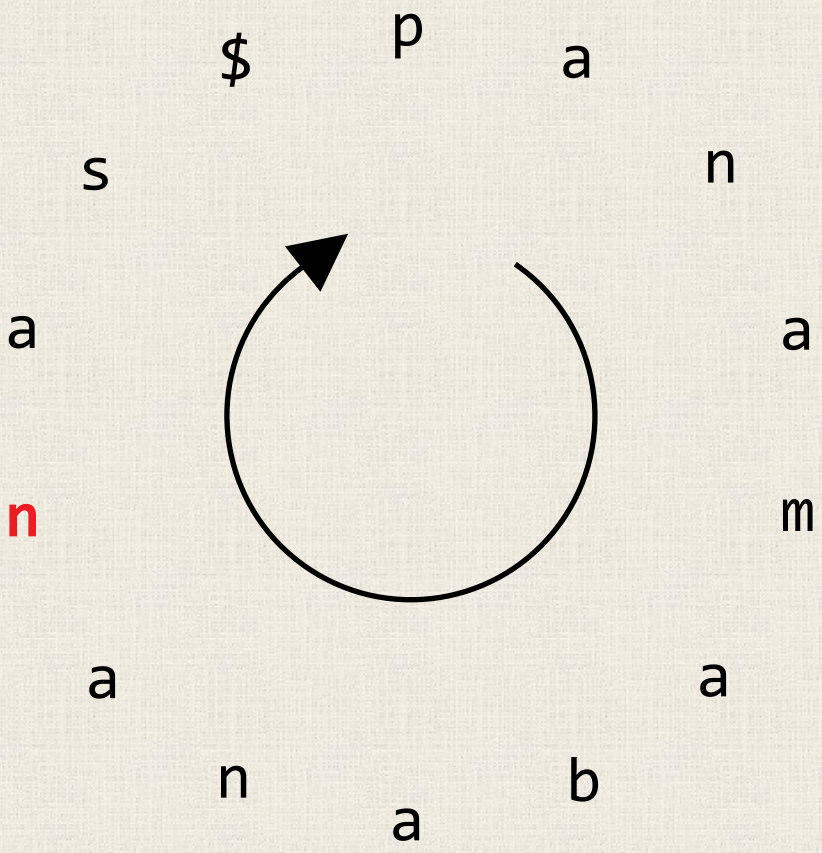
```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
```



Form all cyclic rotations of
“panamabananas$”

The Burrows-Wheeler Transform

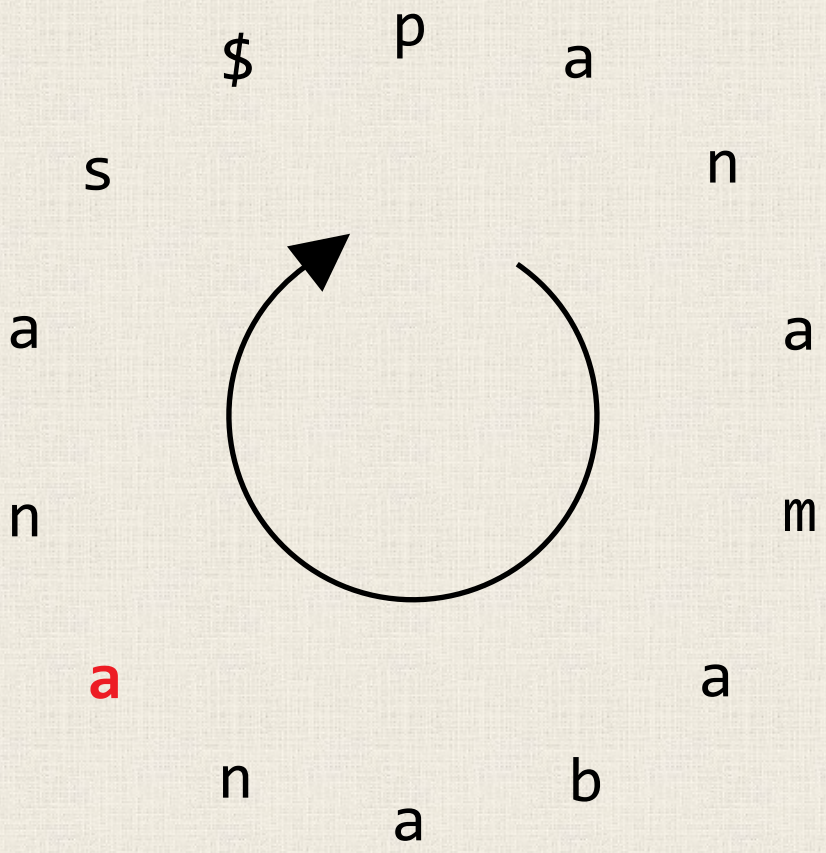
```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
```



Form all cyclic rotations of
“panamabananas$”

The Burrows-Wheeler Transform

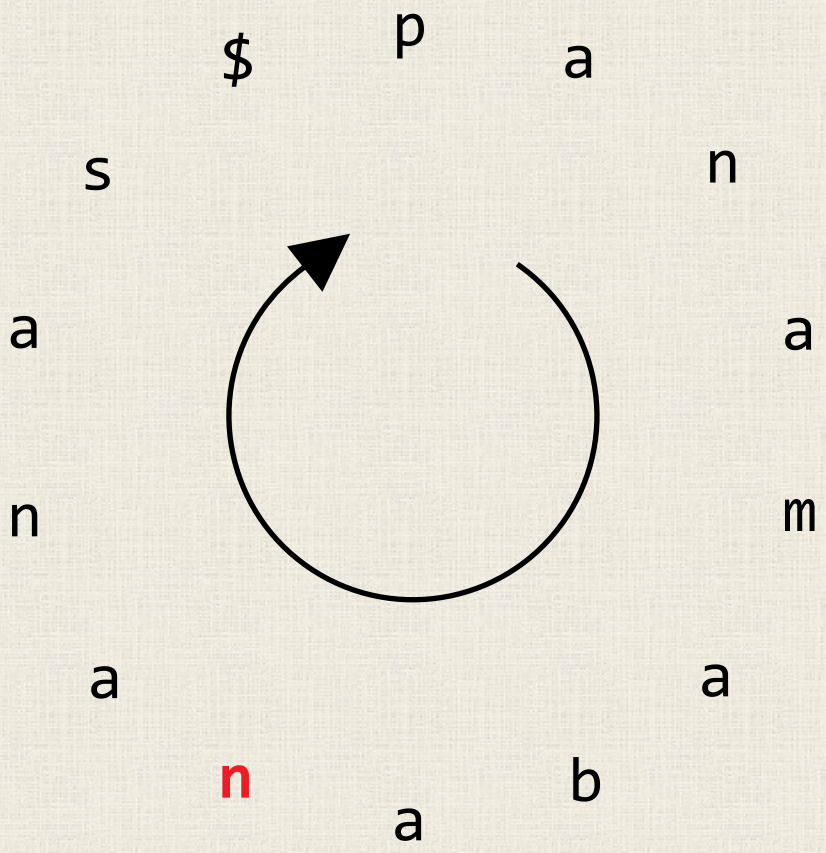
```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
```



Form all cyclic rotations of
“panamabananas$”

The Burrows-Wheeler Transform

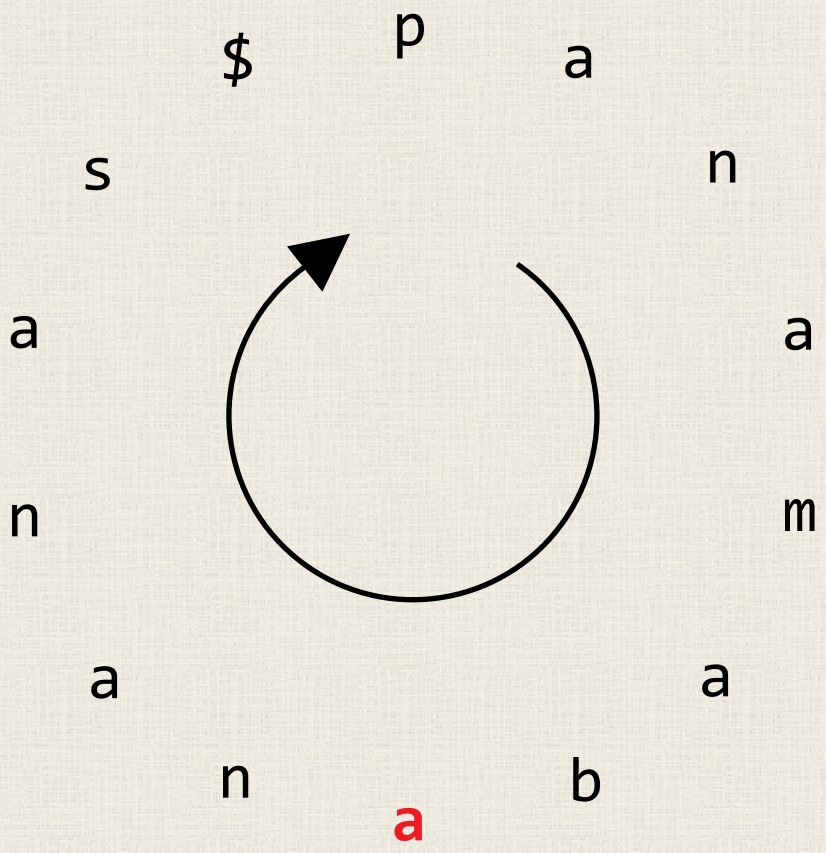
```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
```



Form all cyclic rotations of
“panamabananas$”

The Burrows-Wheeler Transform

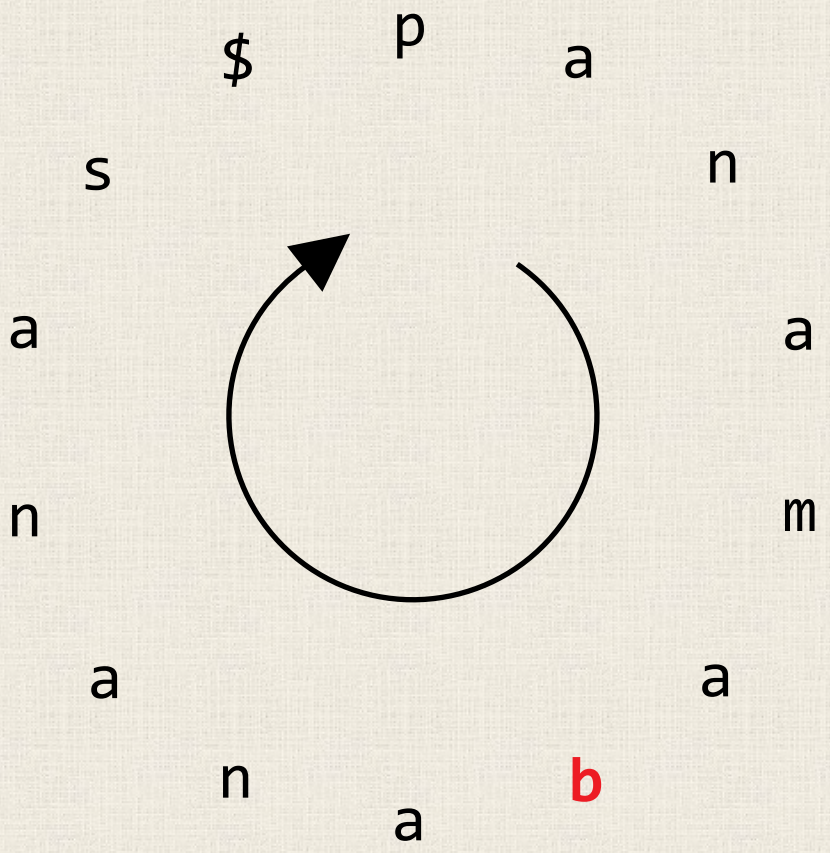
```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
ananas$panamab
```



Form all cyclic rotations of
“panamabananas$”

The Burrows-Wheeler Transform

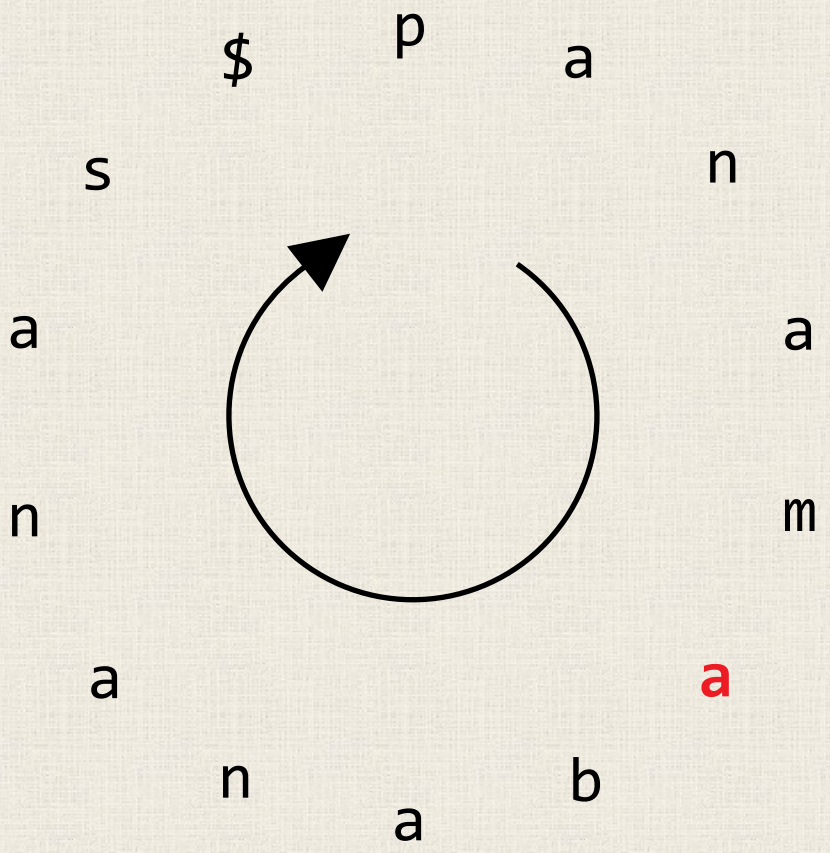
```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
ananas$panamab
bananas$panama
```



Form all cyclic rotations of
“panamabananas$”

The Burrows-Wheeler Transform

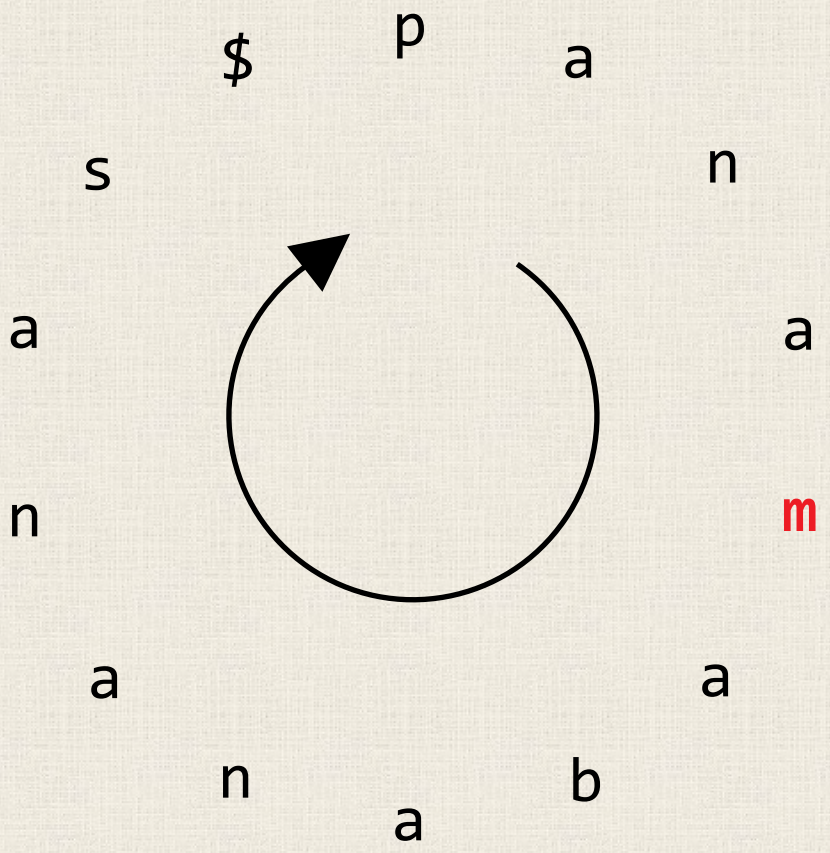
```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
ananas$panamab
bananas$panama
abananas$panam
```



Form all cyclic rotations of
“panamabananas$”

The Burrows-Wheeler Transform

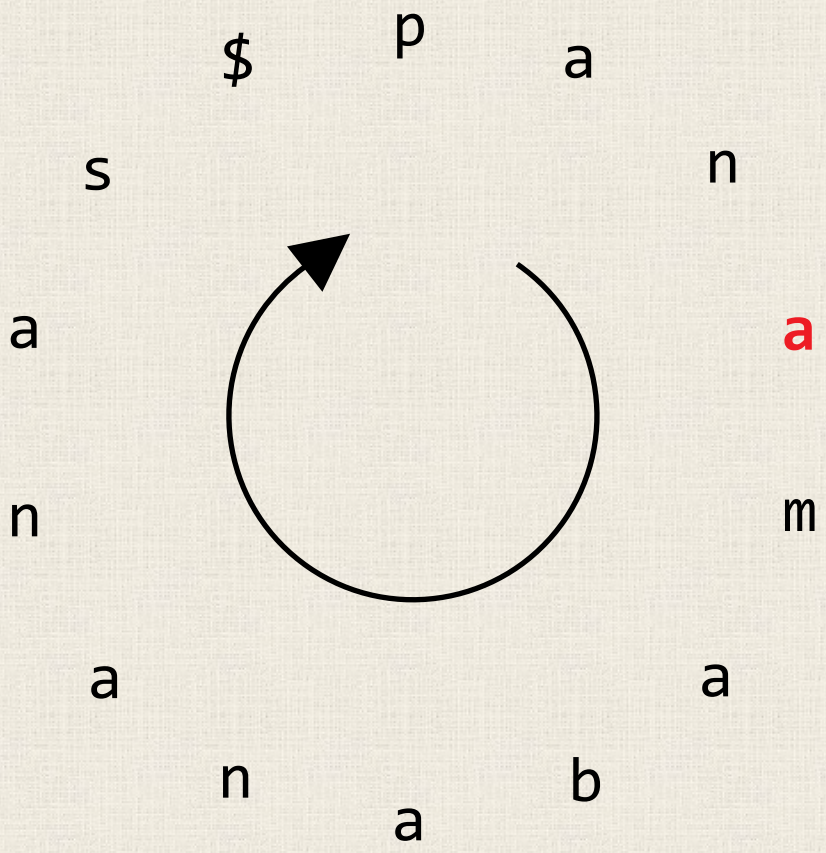
```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
ananas$panamab
bananas$panama
abananas$panam
mabananas$pana
```



Form all cyclic rotations of
“panamabananas$”

The Burrows-Wheeler Transform

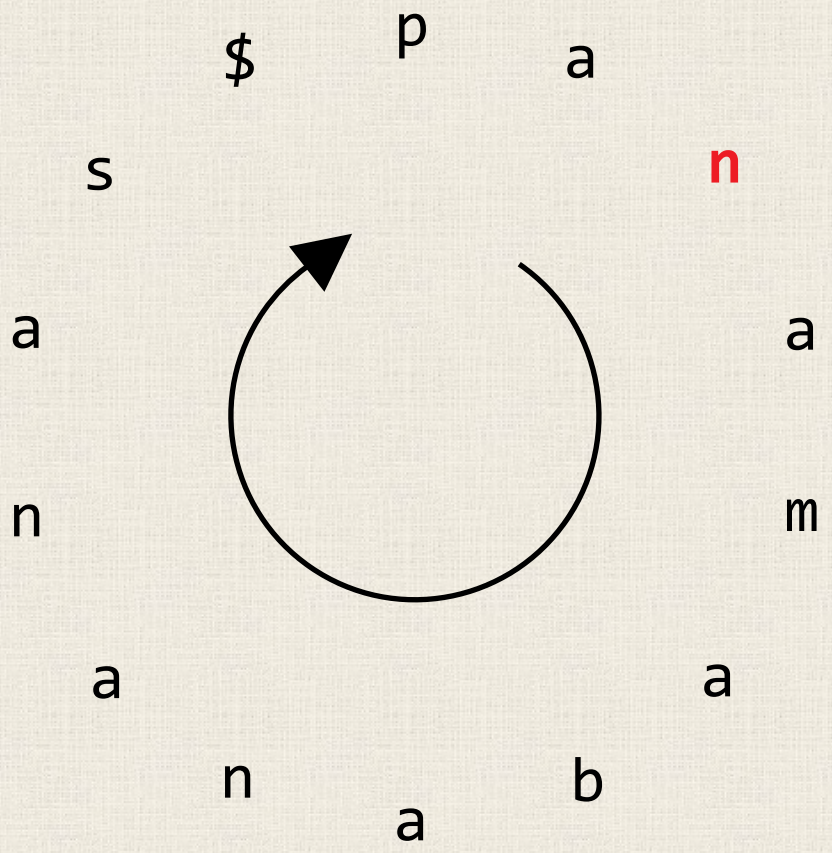
```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
ananas$panamab
bananas$panama
abananas$panam
mabananas$pana
amabananas$pan
```



Form all cyclic rotations of
“panamabananas$”

The Burrows-Wheeler Transform

```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
ananas$panamab
bananas$panama
abananas$panam
mabananas$pana
amabananas$pan
namabananas$pa
```

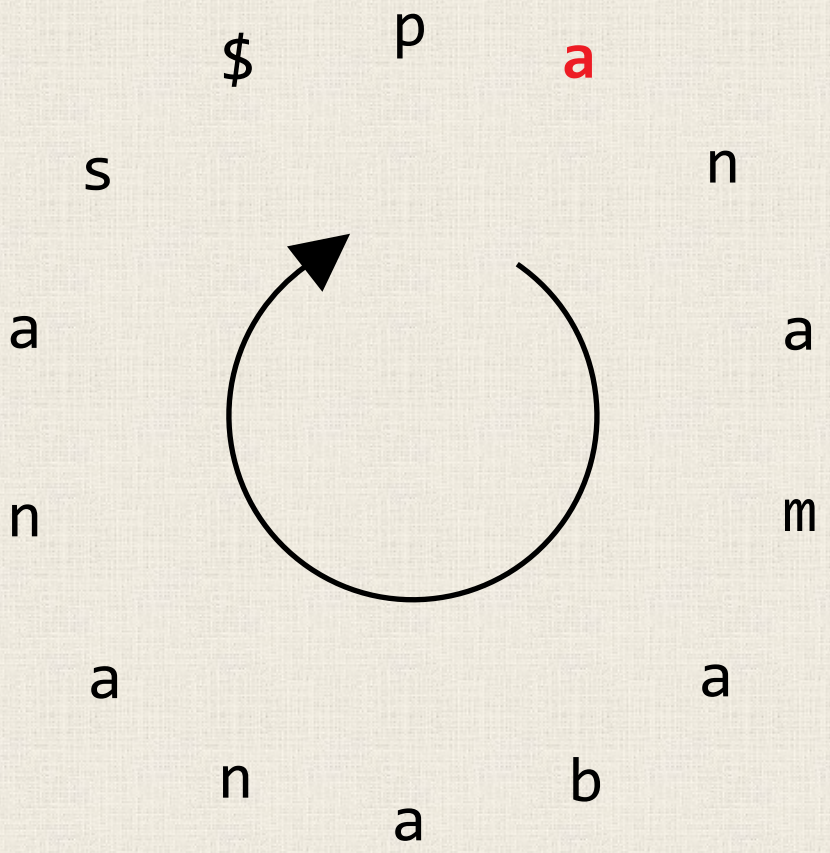


Form all cyclic rotations of
"panamabananas$"

The Burrows-Wheeler Transform

```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
ananas$panamab
bananas$panama
abananas$panam
mabananas$pana
amabananas$pan
namabananas$pa
anamabananas$p
```

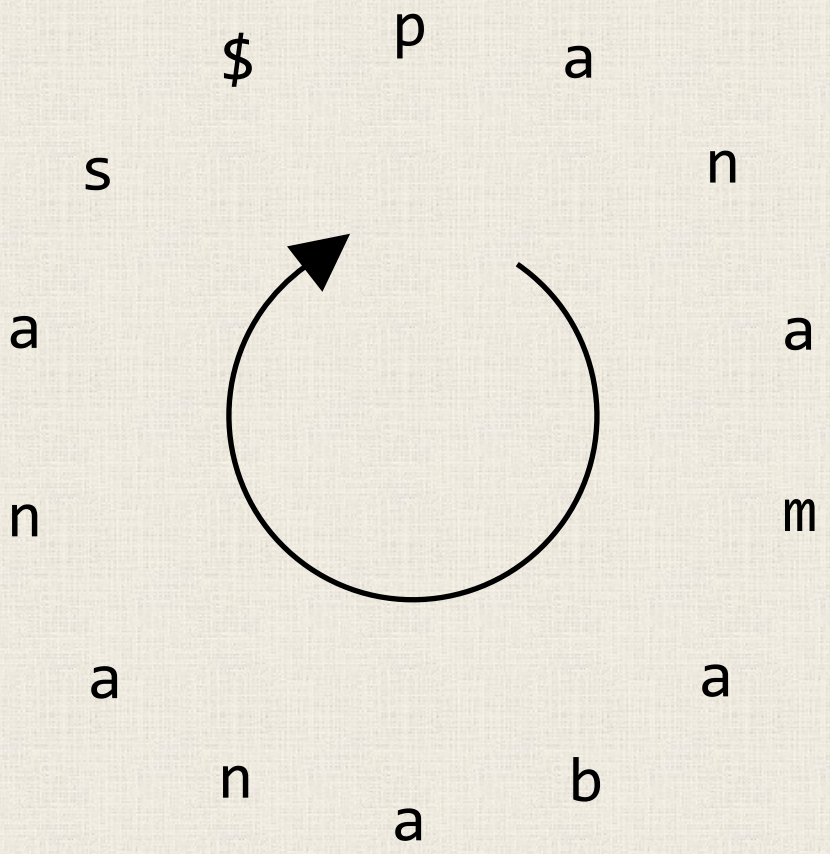
Form all cyclic rotations of "panamabananas$"



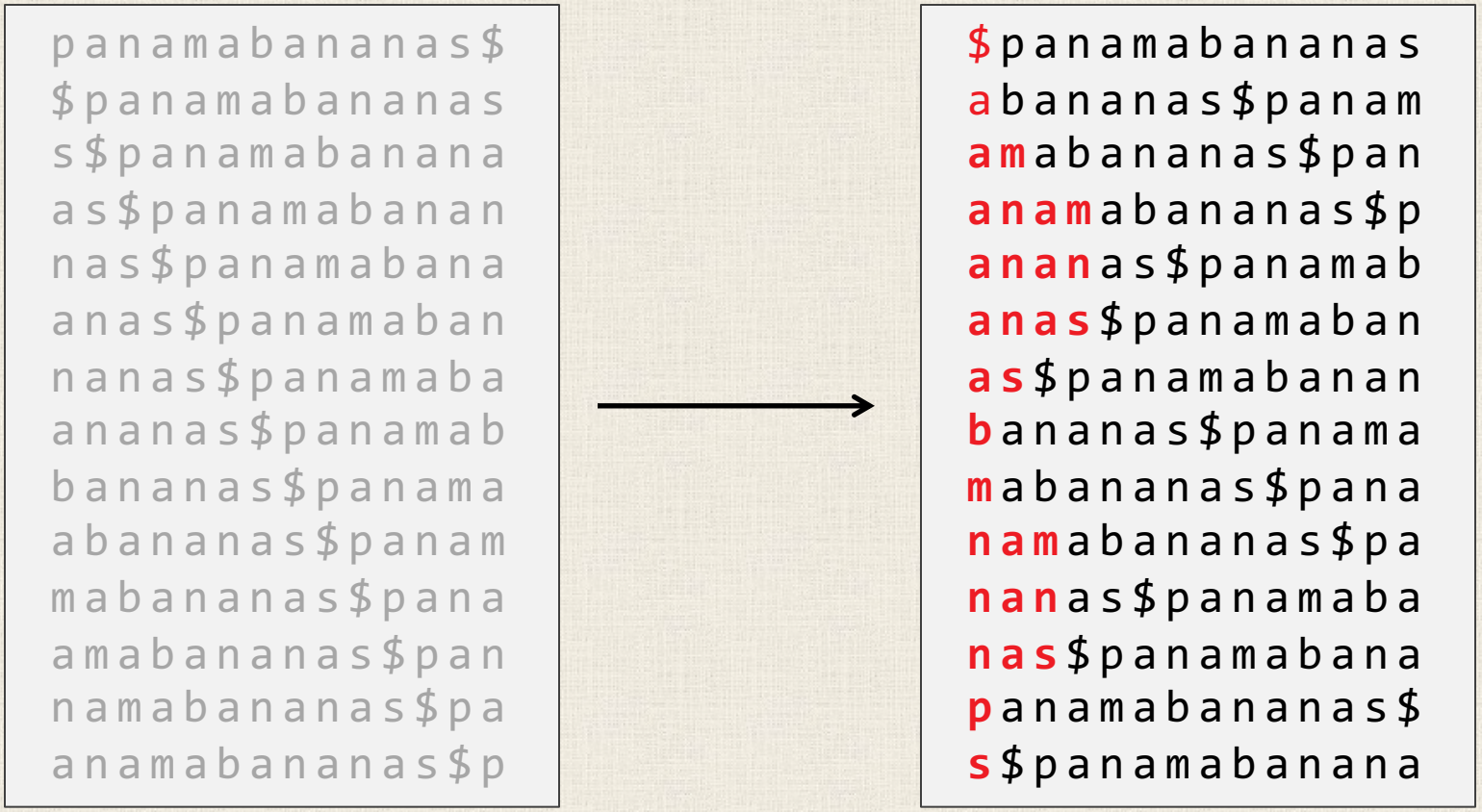
The Burrows-Wheeler Transform

```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
ananas$panamab
bananas$panama
abananas$panam
mabananas$pana
amabananas$pan
namabananas$pa
anamabananas$p
```

Form all cyclic rotations of "panamabananas$"



The Burrows-Wheeler Transform

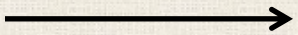


Form all cyclic rotations of "panamabananas$"

Sort the strings lexicographically ($ comes first)

The Burrows-Wheeler Transform

```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
ananas$panamab
bananas$panama
abananas$panam
mabananas$pana
amabananas$pan
namabananas$pa
anamabananas$p
```

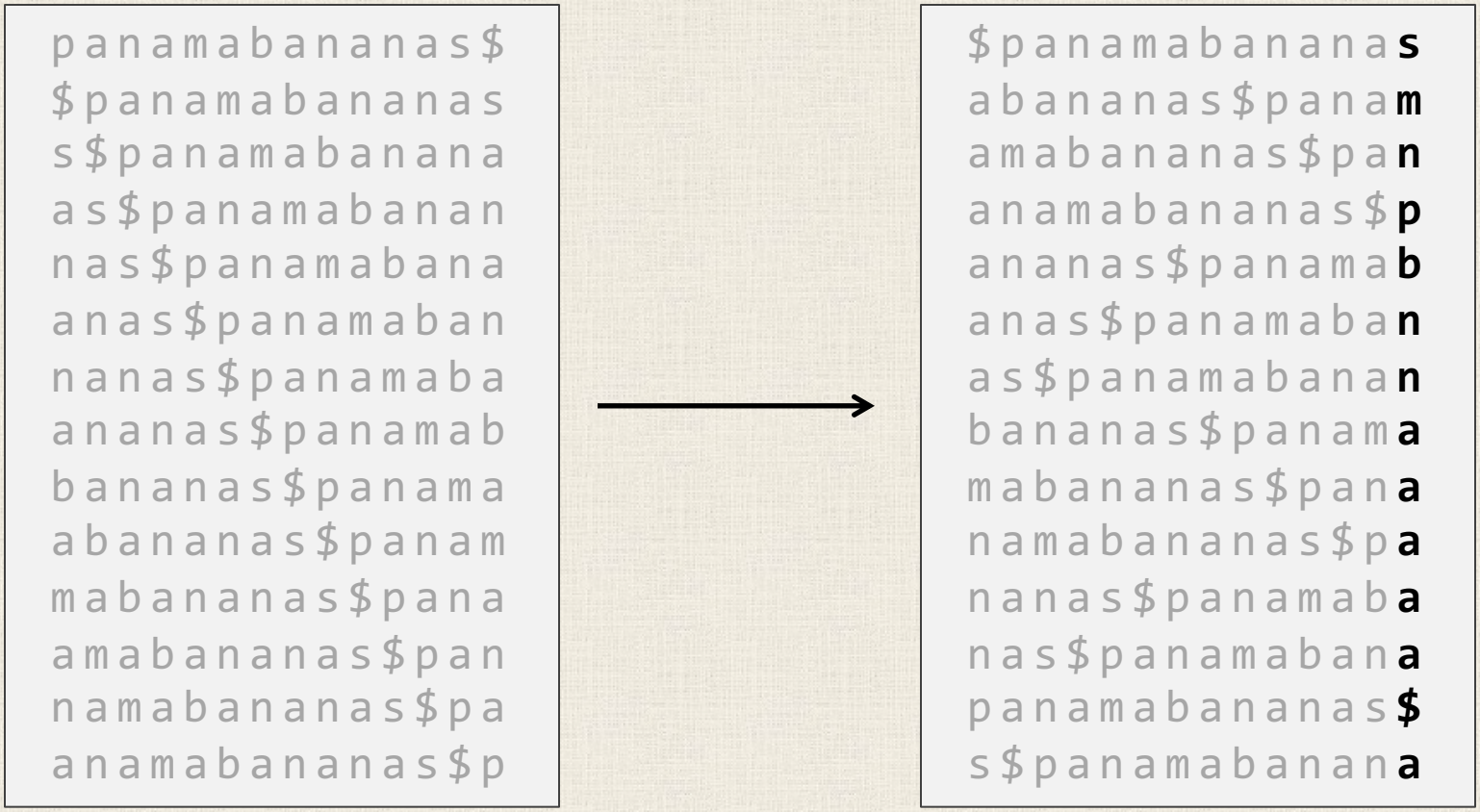


```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```

Form all cyclic rotations of "panamabananas$"

We call this matrix of symbols *M*(*Text*)

The Burrows-Wheeler Transform



Form all cyclic rotations of *Text* = "panamabananas$"

BWT(*Text*) = last column of *M*(*text*)= "smnpbnnaaaaa$a".

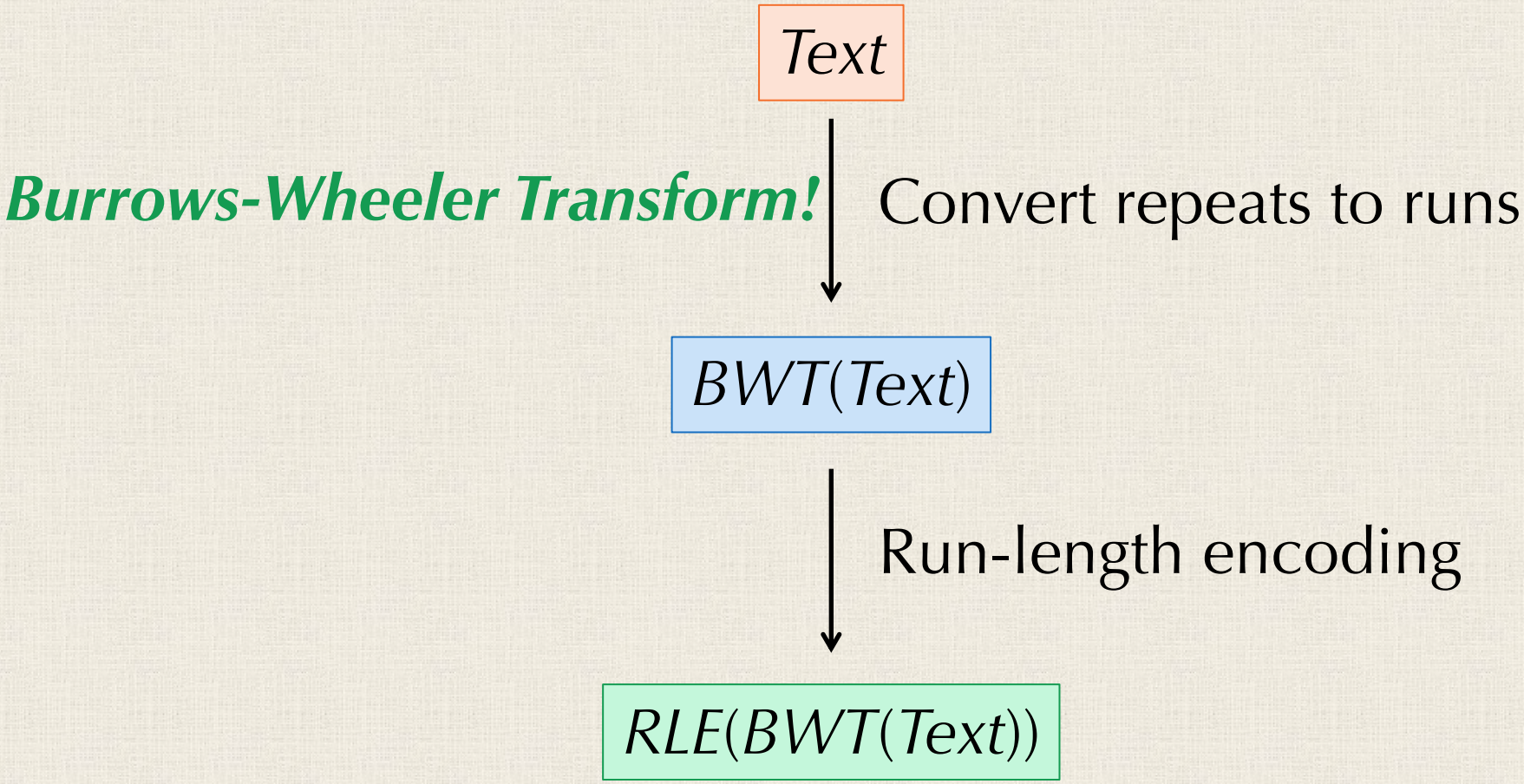
Let's Examine BWT(*Text*) for *Text* = Watson & Crick, 1953

```
nd Corey (1).  They kindly made their manuscript availa ..... a
nd criticism, especially on interatomic distances.  We  ..... a
nd cytosine.  The sequence of bases on a single chain d ..... a
nd experimentally (3,4) that the ratio of the amounts o ..... u
nd for this reason we shall not comment on it.  We wish ..... a
nd guanine (purine) with cytosine (pyrimidine).  In oth ..... a
nd ideas of Dr.  M. H. F. Wilkins, Dr.  R. E. Franklin  ..... a
nd its water content is rather high.  At lower water co ..... a
nd pyrimidine bases.  The planes of the bases are perpe ..... a
nd stereochemical arguments.  It has not escaped our no ..... a
nd that only specific pairs of bases can bond together  ..... u
nd the atoms near it is close to Furberg's 'standard co ..... a
nd the bases on the inside, linked together by hydrogen ..... a
nd the bases on the outside.  In our opinion, this stru ..... a
nd the other a pyrimidine for bonding to occur.  The hy ..... a
nd the phosphates on the outside.  The configuration of ..... a
nd the ration of guanine to cytosine, are always very c ..... a
nd the same axis (see diagram).  We have made the usual ..... u
nd their co-workers at King's College, London.  One of  ..... a
```

Where do you think the “u”s come from?

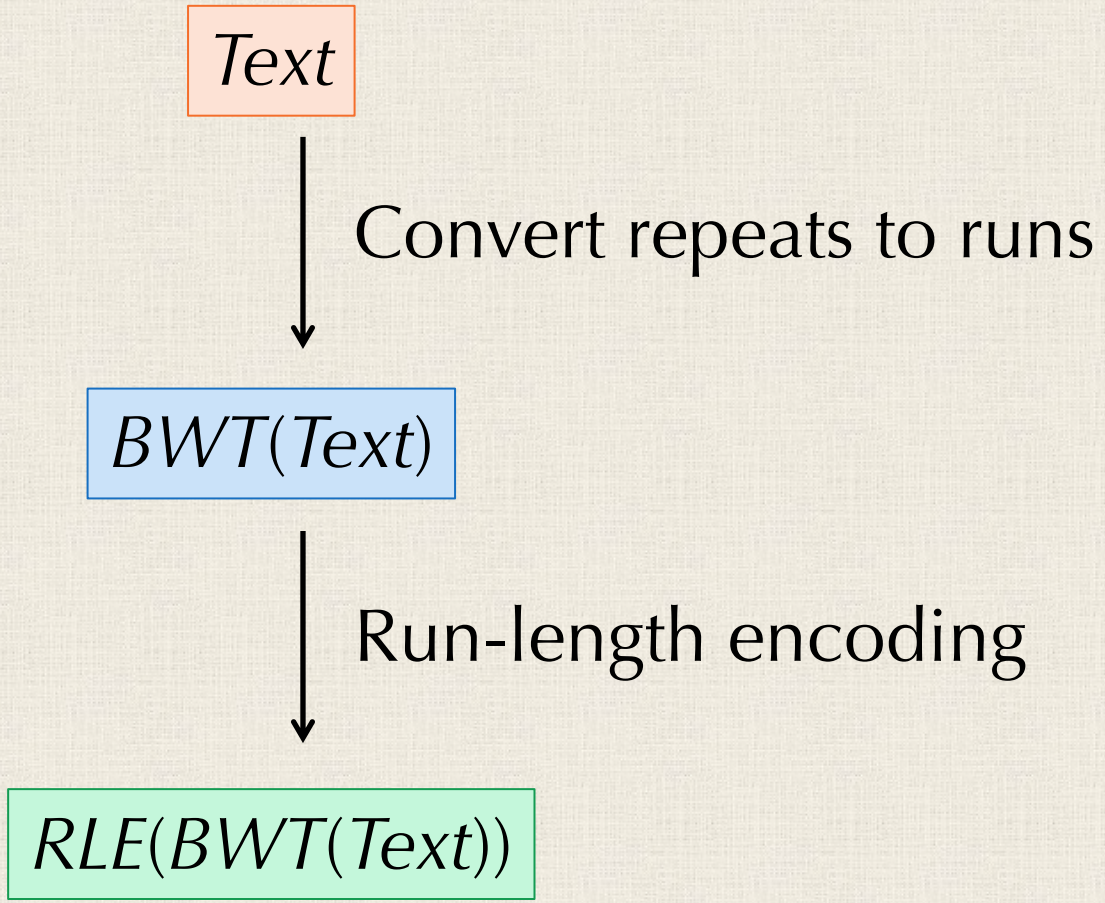
```
nd Corey (1).  They kindly made their manuscript availa ..... a
nd criticism, especially on interatomic distances.  We  ..... a
nd cytosine.  The sequence of bases on a single chain d ..... a
nd experimentally (3,4) that the ratio of the amounts o ..... u
nd for this reason we shall not comment on it.  We wish ..... a
nd guanine (purine) with cytosine (pyrimidine).  In oth ..... a
nd ideas of Dr.  M. H. F. Wilkins, Dr.  R. E. Franklin  ..... a
nd its water content is rather high.  At lower water co ..... a
nd pyrimidine bases.  The planes of the bases are perpe ..... a
nd stereochemical arguments.  It has not escaped our no ..... a
nd that only specific pairs of bases can bond together  ..... u
nd the atoms near it is close to Furberg's 'standard co ..... a
nd the bases on the inside, linked together by hydrogen ..... a
nd the bases on the outside.  In our opinion, this stru ..... a
nd the other a pyrimidine for bonding to occur.  The hy ..... a
nd the phosphates on the outside.  The configuration of ..... a
nd the ration of guanine to cytosine, are always very c ..... a
nd the same axis (see diagram).  We have made the usual ..... u
nd their co-workers at King's College, London.  One of  ..... a
```

BWT Converts Repeats to Runs

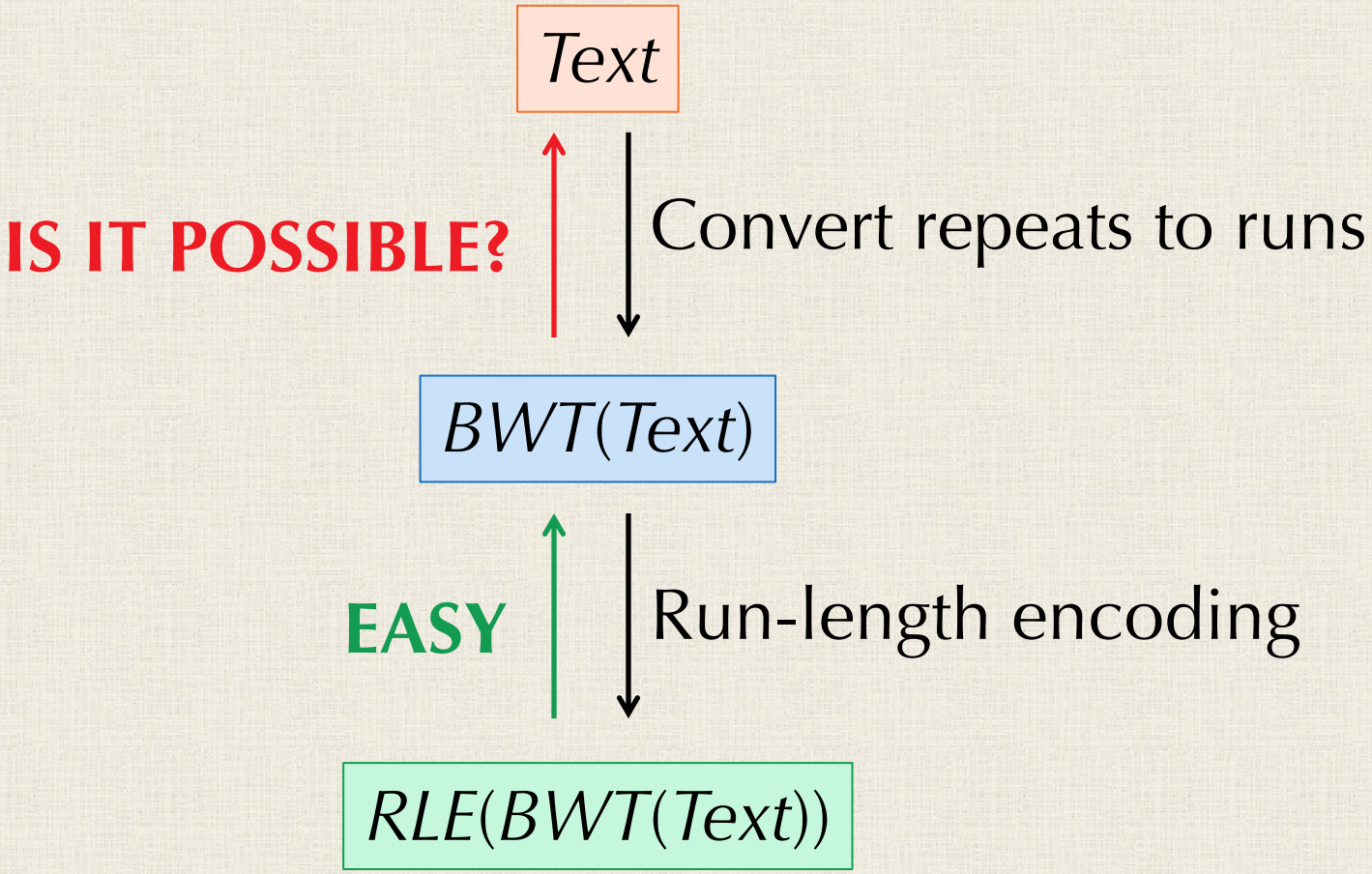


INVERTING THE BURROWS-WHEELER TRANSFORM

How Can We Decompress?

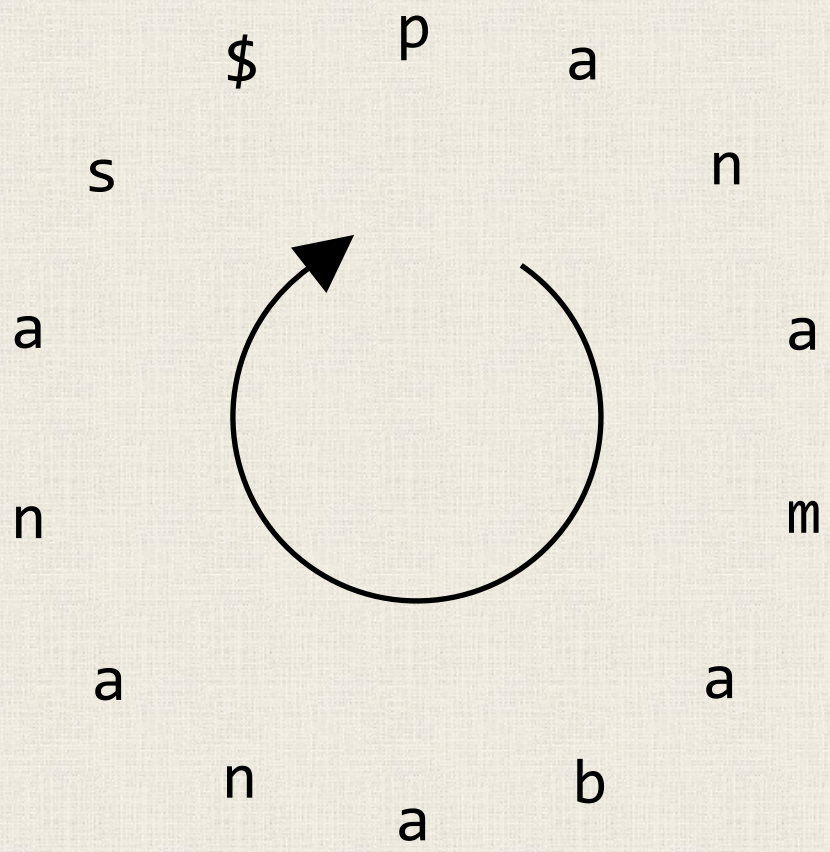


How Can We Decompress?



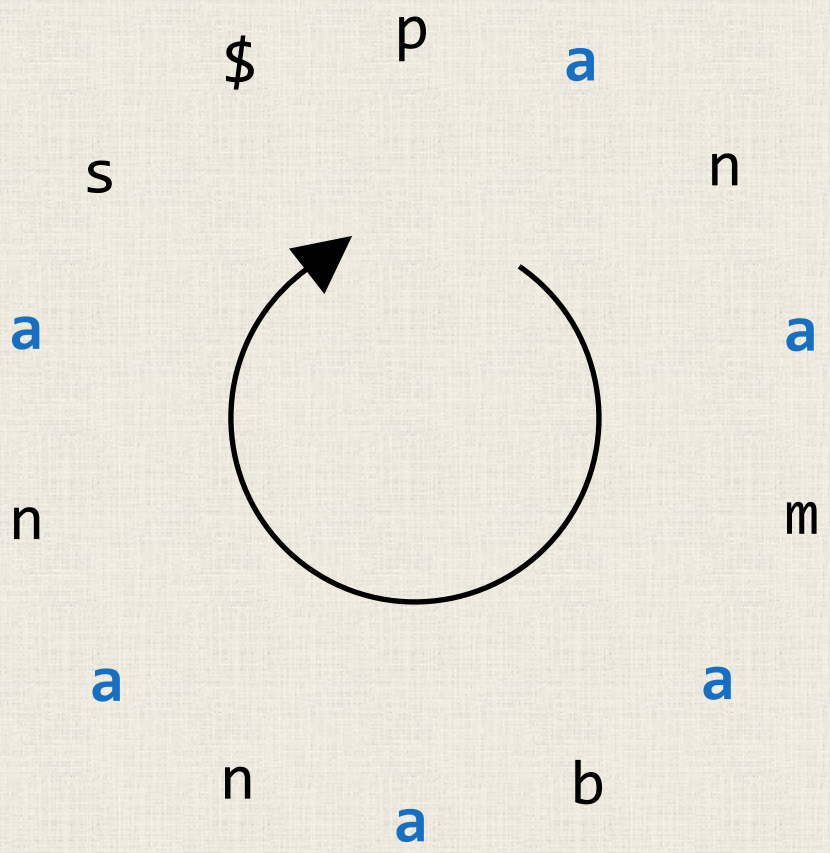
A Strange Observation

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```



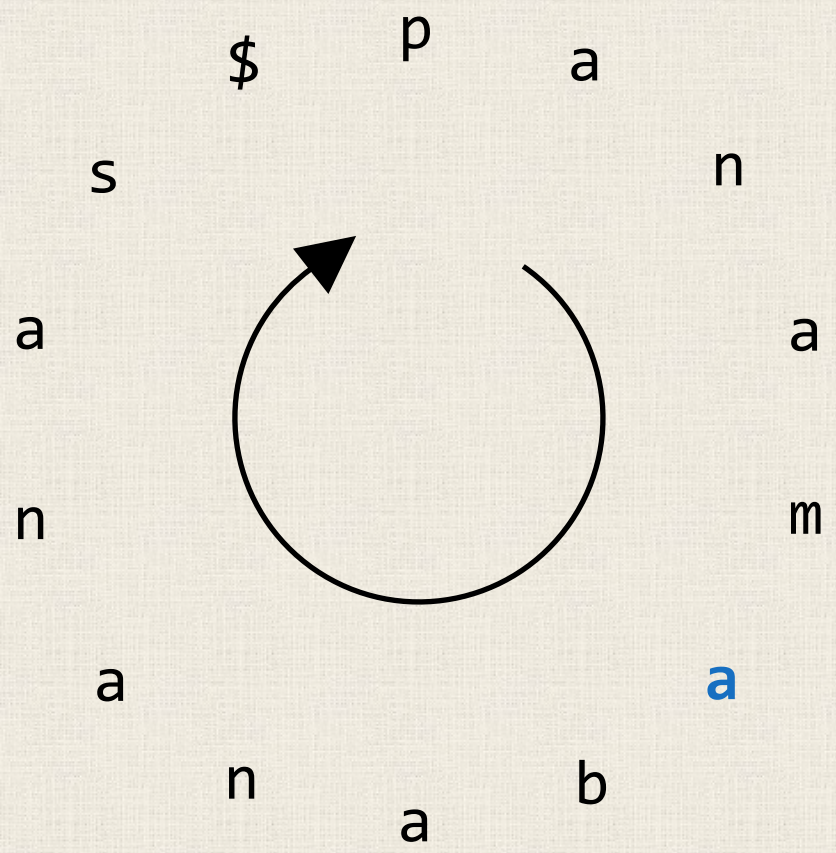
A Strange Observation

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```



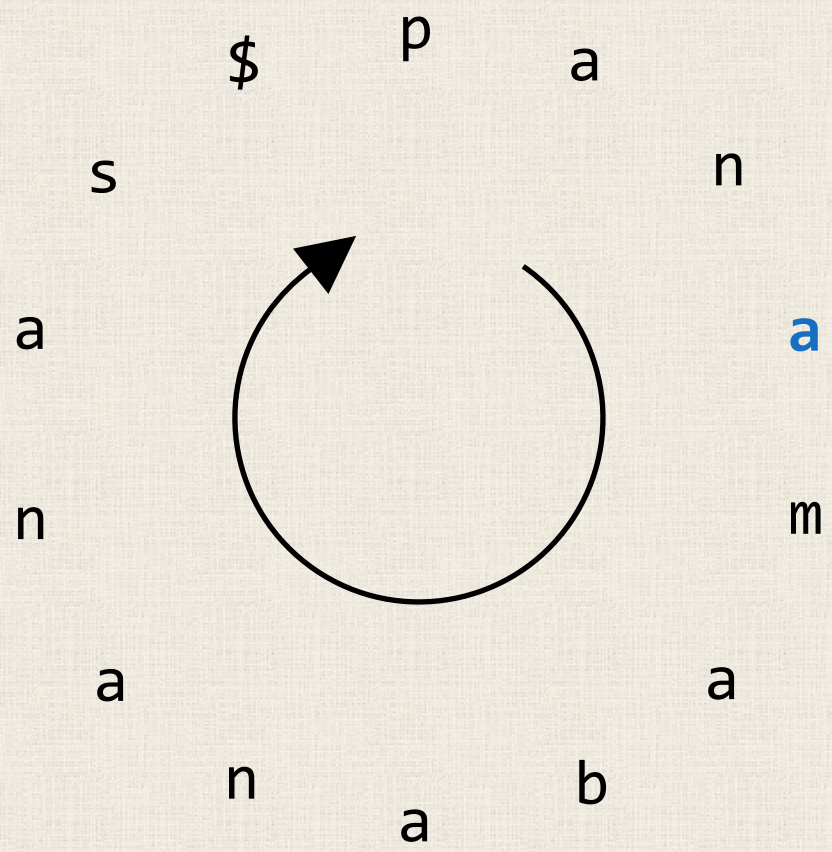
A Strange Observation

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```



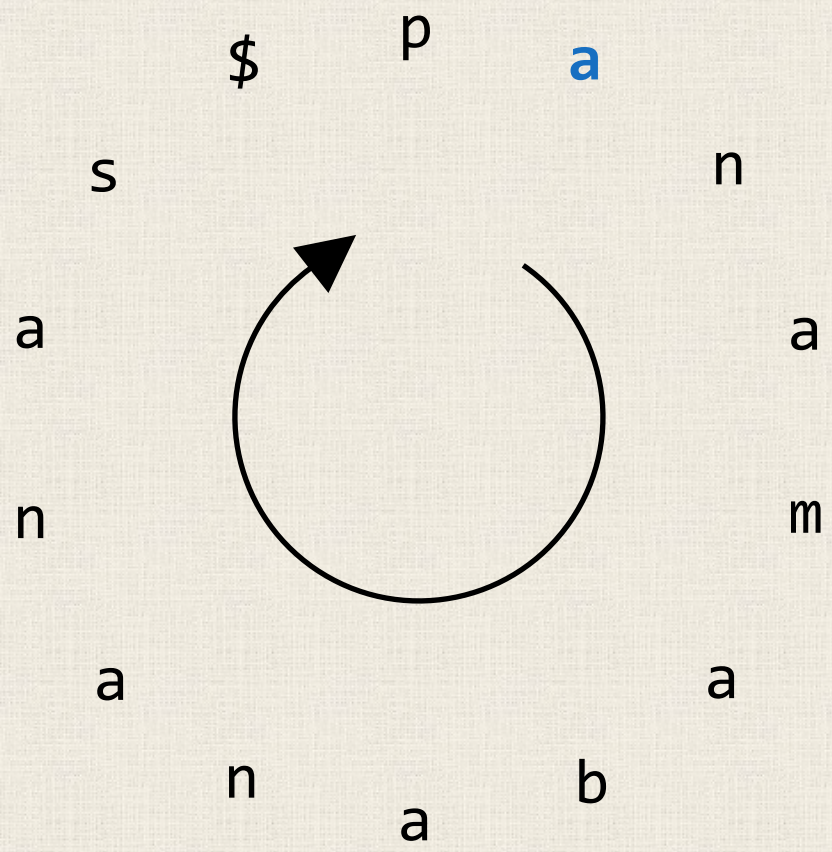
A Strange Observation

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```



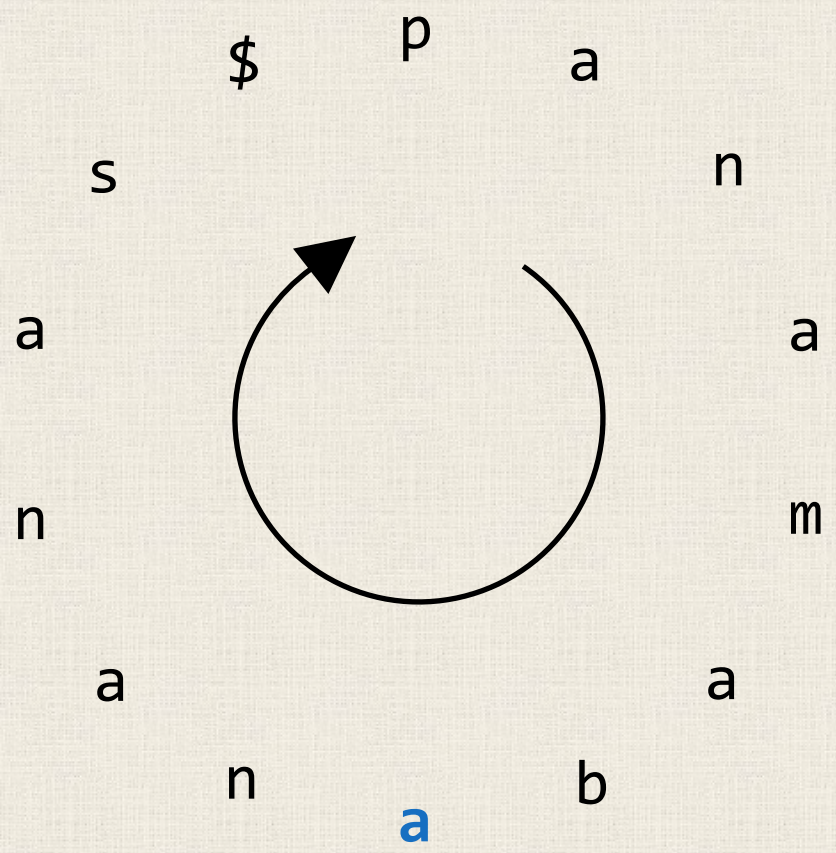
A Strange Observation

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```



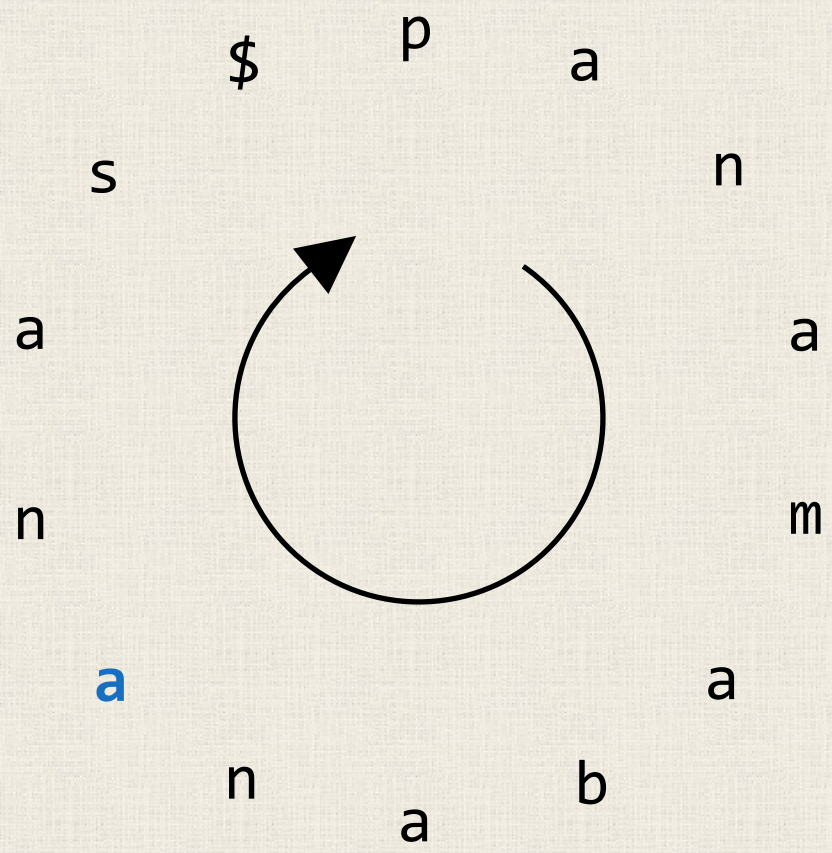
A Strange Observation

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```



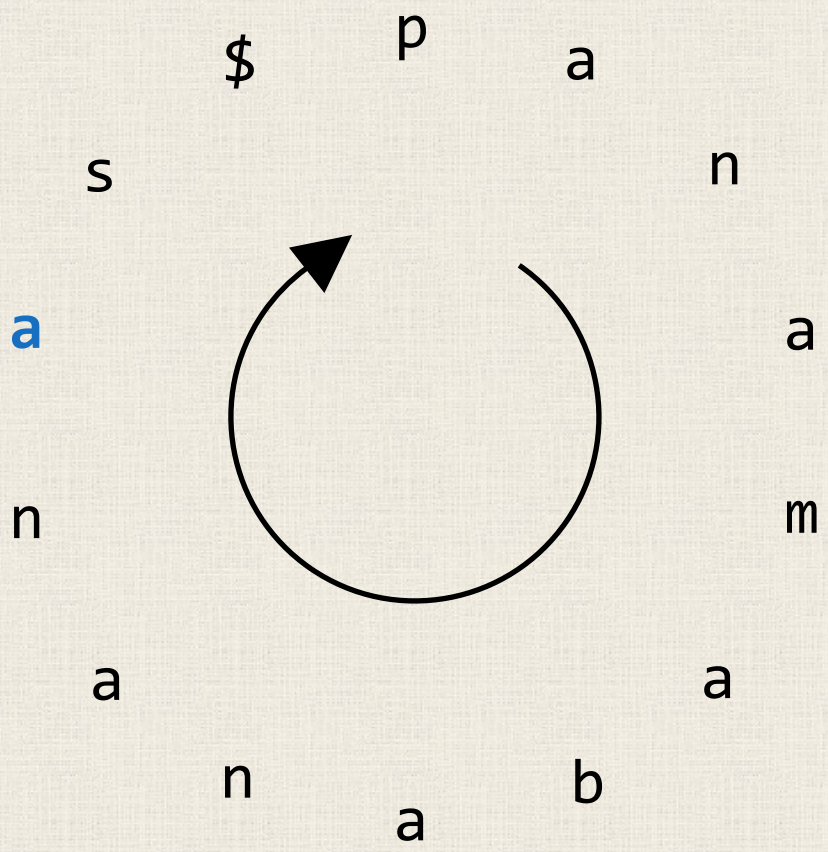
A Strange Observation

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```



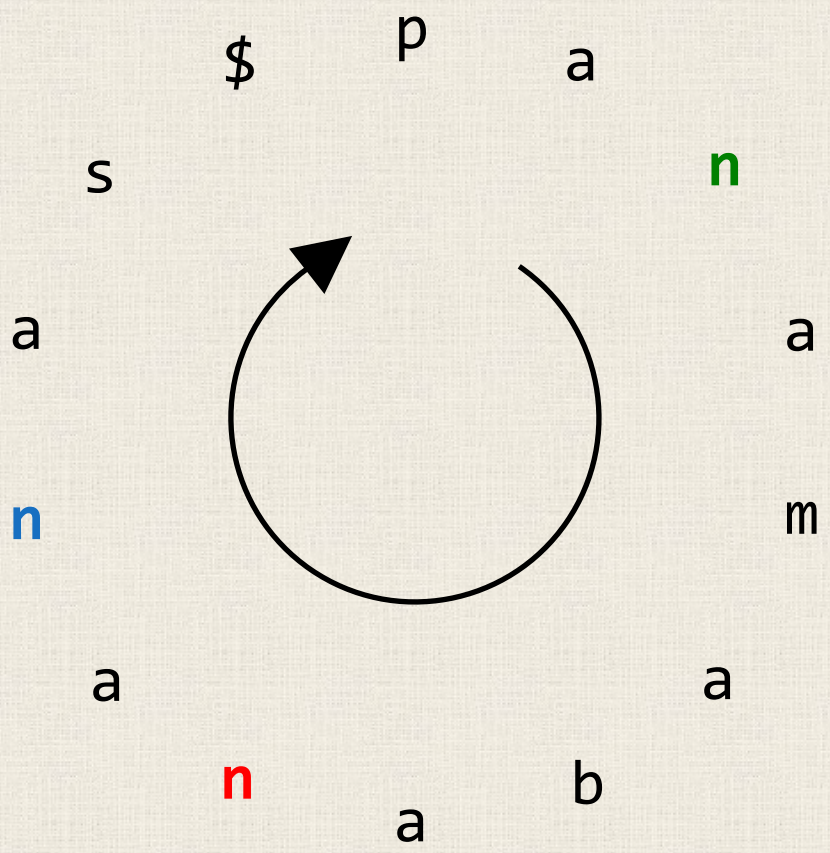
A Strange Observation

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```



A Strange Observation

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```



Is It True in General?

	$panamabananas
1	abananas$panam
2	amabananas$pan
3	anamabananas$p
4	ananas$panamab
5	anas$panamaban
6	as$panamabanan
	bananas$panama
	mabananas$pana
	namabananas$pa
	nanas$panamaba
	nas$panamabana
	panamabananas$
	s$panamabanana

Is It True in General?

	$panamabananas
1	abananas$panam
2	amabananas$pan
3	anamabananas$p
4	ananas$panamab
5	anas$panamaban
6	as$panamabanan
	bananas$panama
	mabananas$pana
	namabananas$pa
	nanas$panamaba
	nas$panamabana
	panamabananas$
	s$panamabanana

These strings are sorted

Is It True in General?

	$panamabananas
1	abananas$panam
2	amabananas$pan
3	anamabananas$p
4	ananas$panamab
5	anas$panamaban
6	as$panamabanan
	bananas$panama
	mabananas$pana
	namabananas$pa
	nanas$panamaba
	nas$panamabana
	panamabananas$
	s$panamabanana


```
bananas$panam
mabananas$pan
namabananas$p
nanas$panamab
nas$panamaban
s$panamabanan
```

These strings are sorted

Is It True in General?

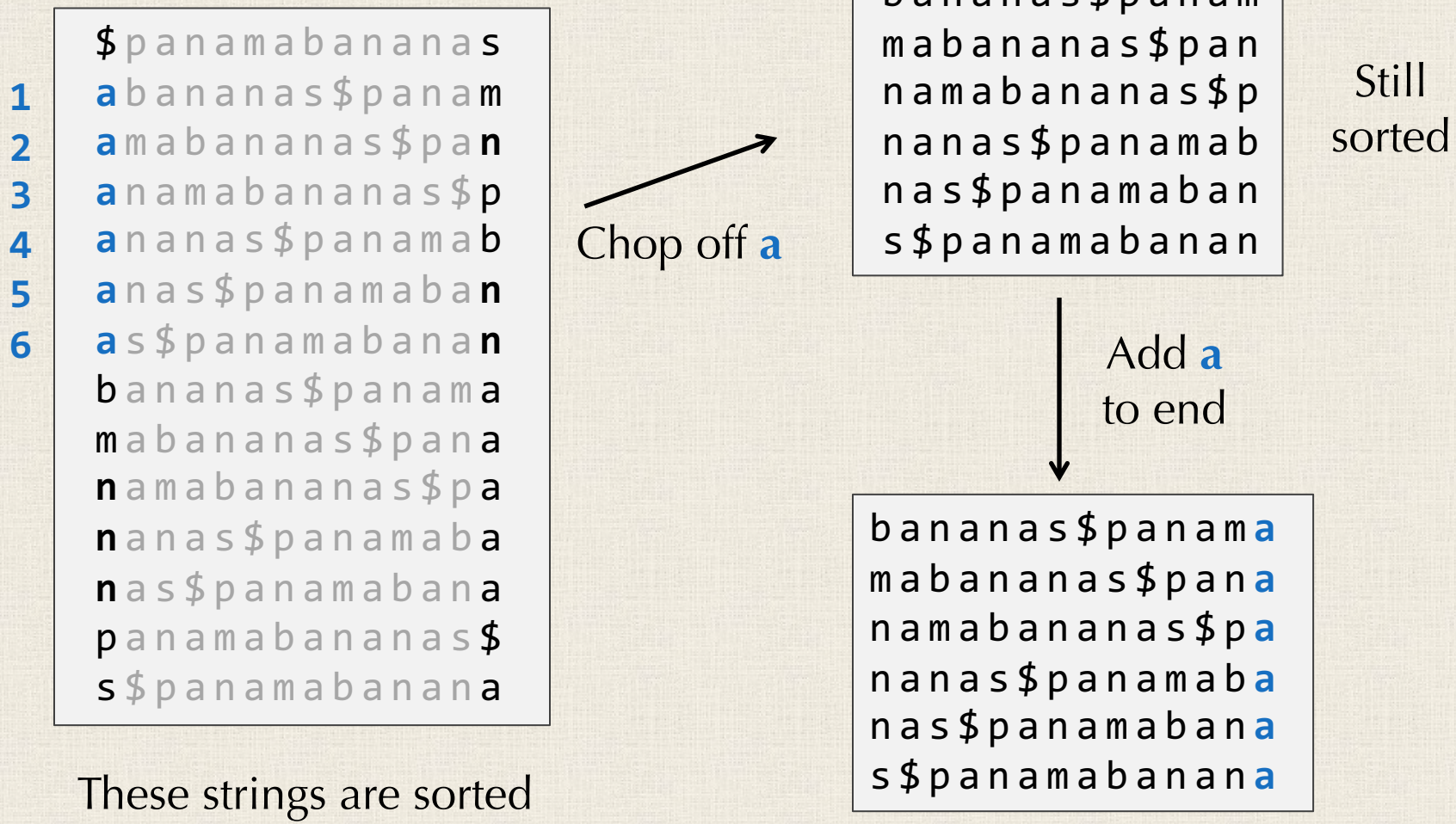
	$panamabananas
1	abananas$panam
2	amabananas$pan
3	anamabananas$p
4	ananas$panamab
5	anas$panamaban
6	as$panamabanan
	bananas$panama
	mabananas$pana
	namabananas$pa
	nanas$panamaba
	nas$panamabana
	panamabananas$
	s$panamabanana

These strings are sorted

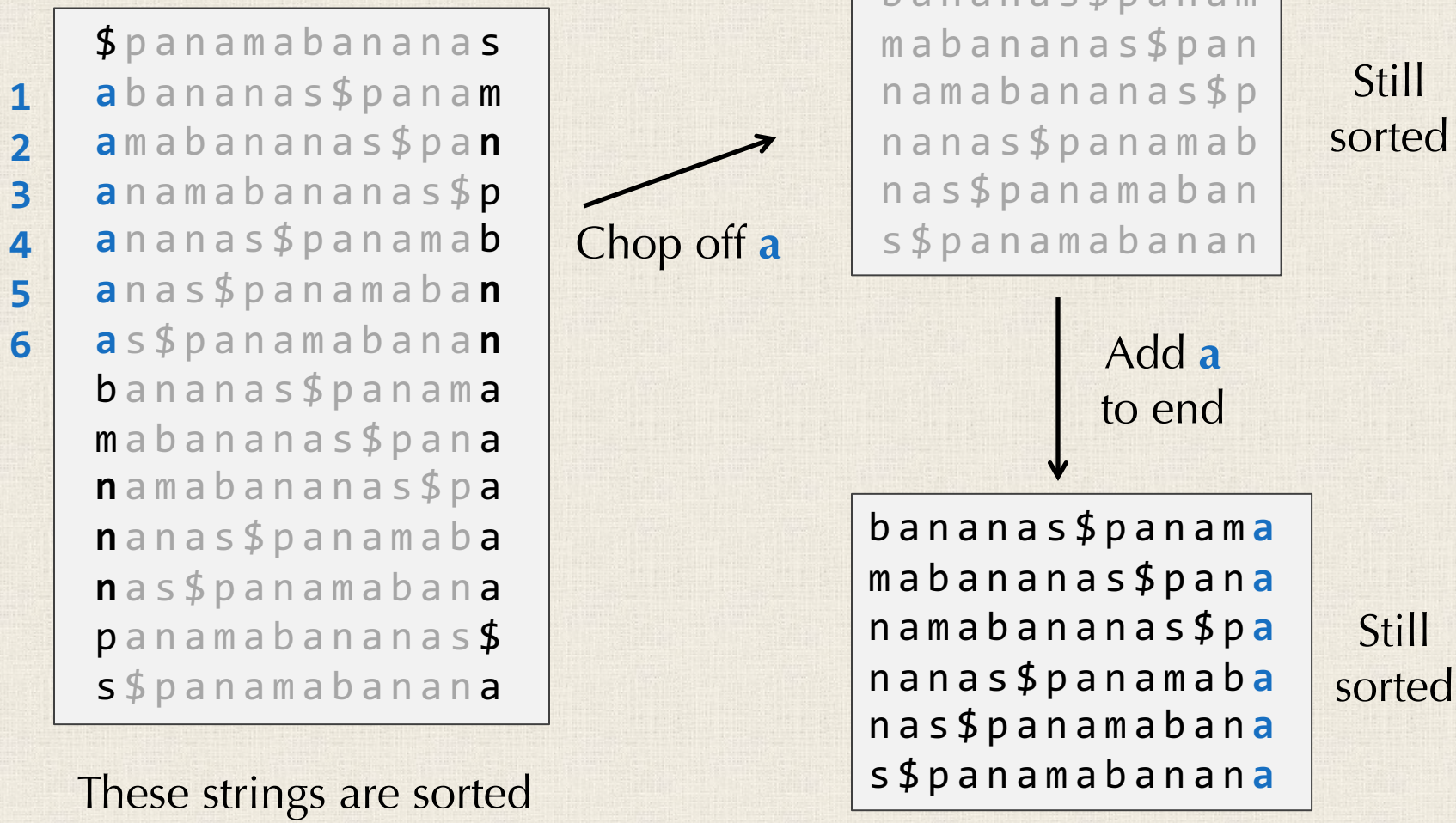
bananas$panam
mabananas$pan
namabananas$p
nanas$panamab
nas$panamaban
s$panamabanan

Still sorted

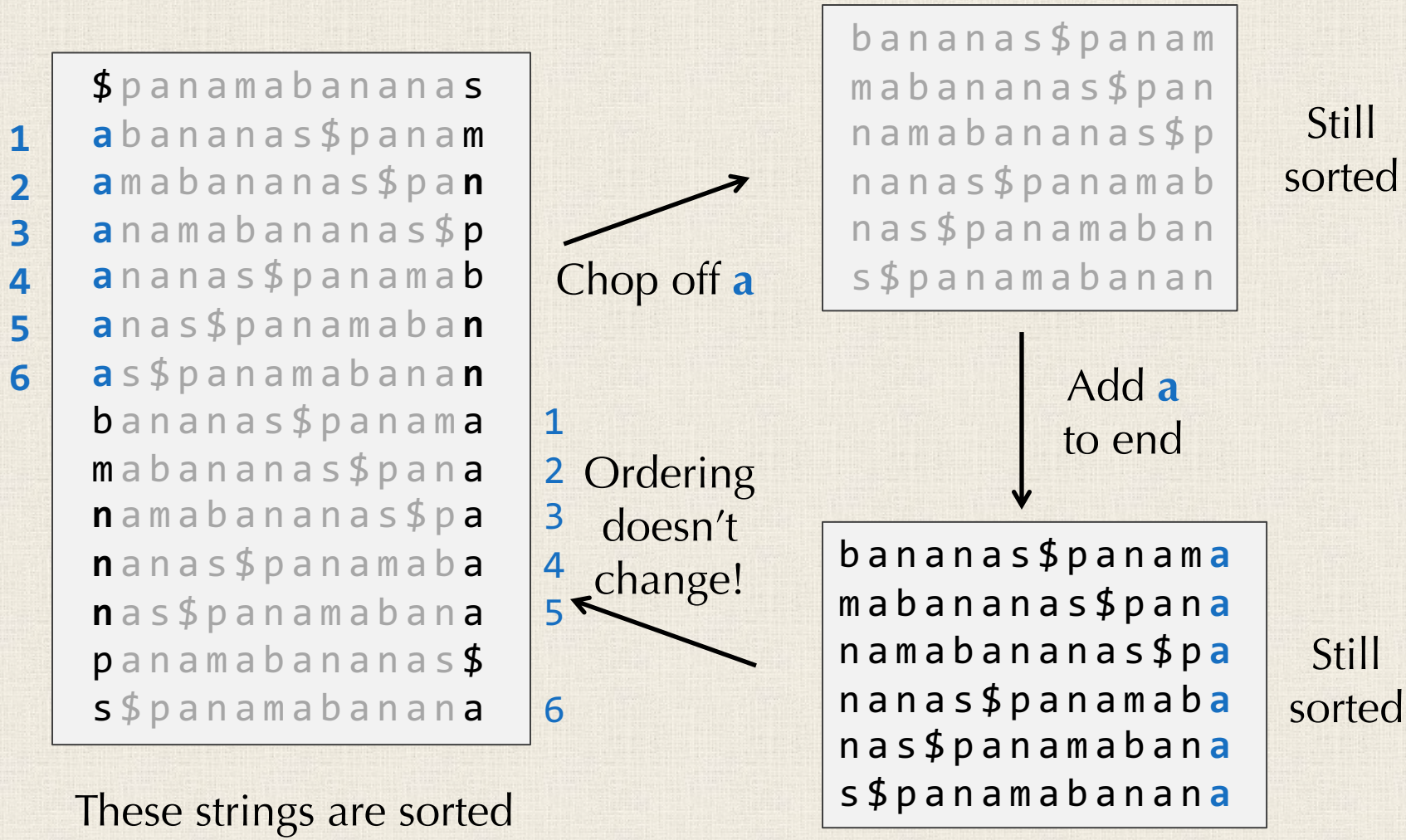
Is It True in General?



Is It True in General?



Is It True in General?



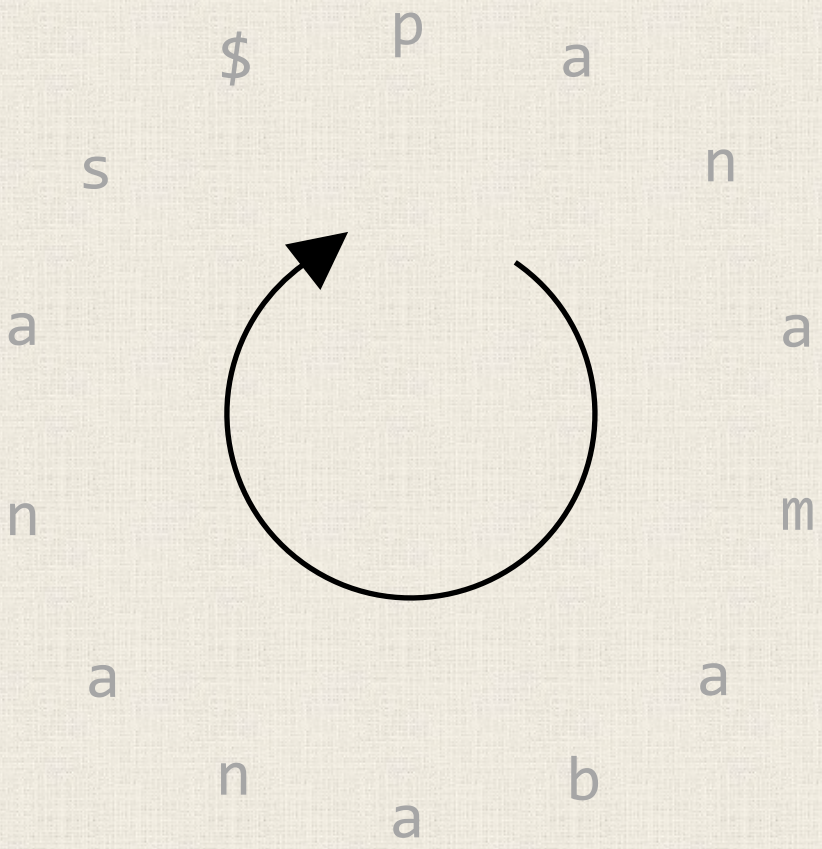
Is It True in General?

$ 1	panamabanana s 1
a 1	bananas$pana m 1
a 2	mabananas$pa n 1
a 3	namabananas$ p 1
a 4	nanas$panama b 1
a 5	nas$panamaba n 2
a 6	s$panamabana n 3
b 1	ananas$panam a 1
m 1	abananas$pan a 2
n 1	amabananas$p a 3
n 2	anas$panamab a 4
n 3	as$panamaban a 5
p 1	anamabananas $ 1
s 1	$panamabanan a 6

First-Last Property: The *k*-th occurrence of *symbol* in *FirstColumn* and the *k*-th occurrence of *symbol* in *LastColumn* correspond to the same position of *symbol* in *Text*.

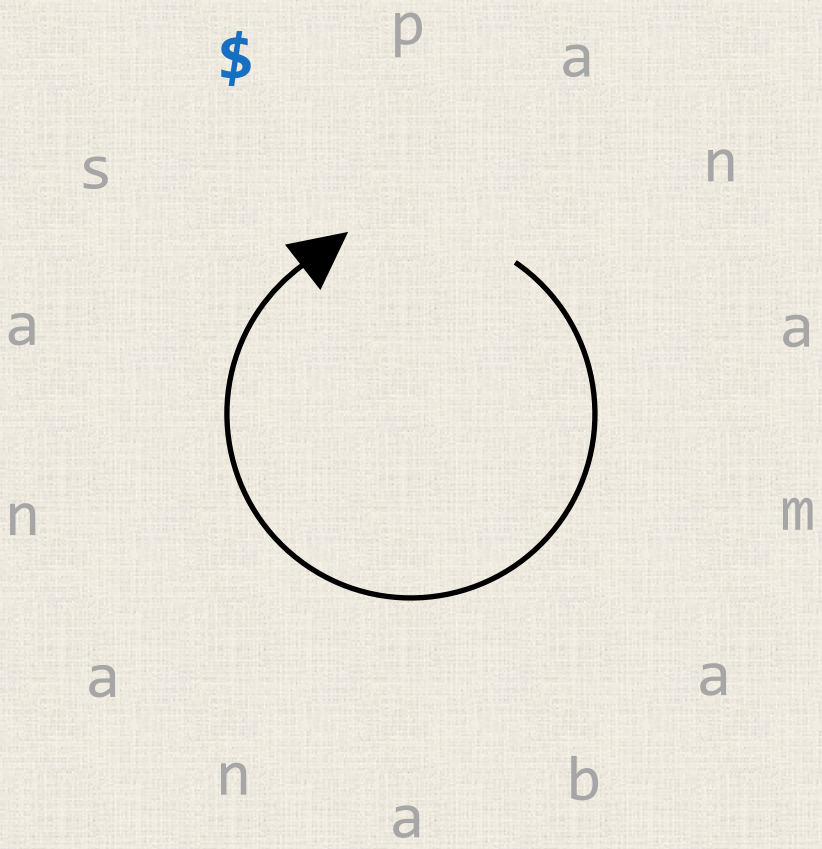
Efficient BWT Decompression

$_1	panamabananas_1
a_1	bananas$panam_1
a_2	mabananas$pan_1
a_3	namabananas$p_1
a_4	nanas$panamab_1
a_5	nas$panamaban_2
a_6	s$panamabanan_3
b_1	ananas$panama_1
m_1	abananas$pana_2
n_1	amabananas$pa_3
n_2	anas$panamaba_4
n_3	as$panamaban_5
p_1	anamabananas$_1
s_1	$panamabanana_6



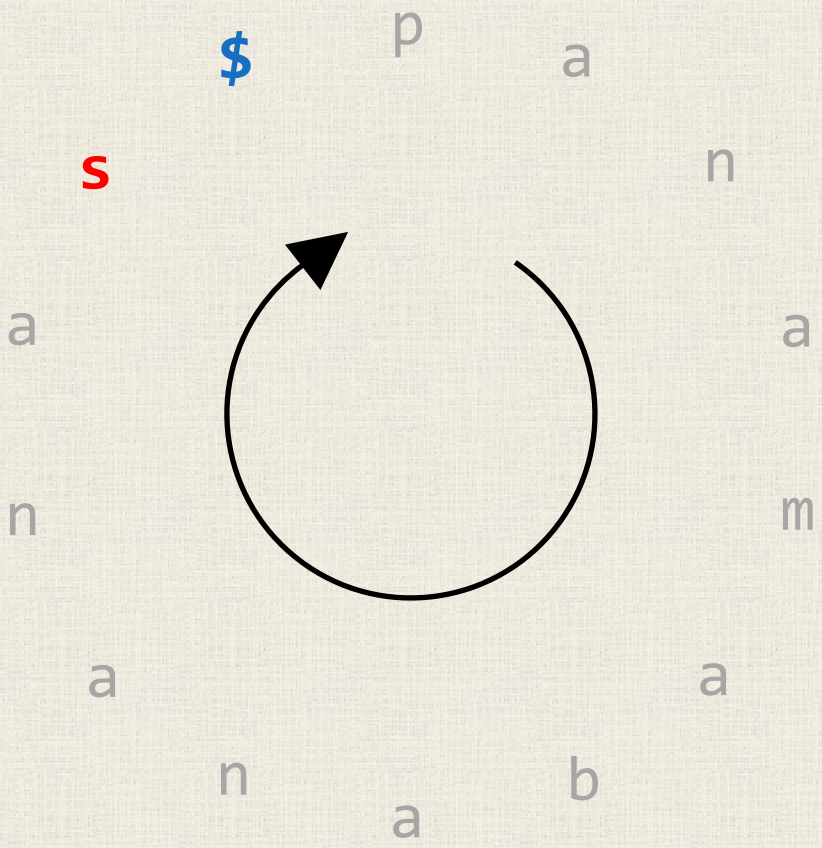
Efficient BWT Decompression

$	panamabananas	$_1$
a	bananas$panam	a$_1$
a	mabananas$pan	a$_2$
a	namabananas$p	a$_3$
a	nanas$panamab	a$_4$
a	nas$panamaban	a$_5$
a	s$panamabanan	a$_6$
b	ananas$panama	b$_1$
m	abananas$pana	m$_1$
n	amabananas$pa	n$_1$
n	anas$panamaba	n$_2$
n	as$panamabana	n$_3$
p	anamabananas$	p$_1$
s	$panamabanana	s$_1$



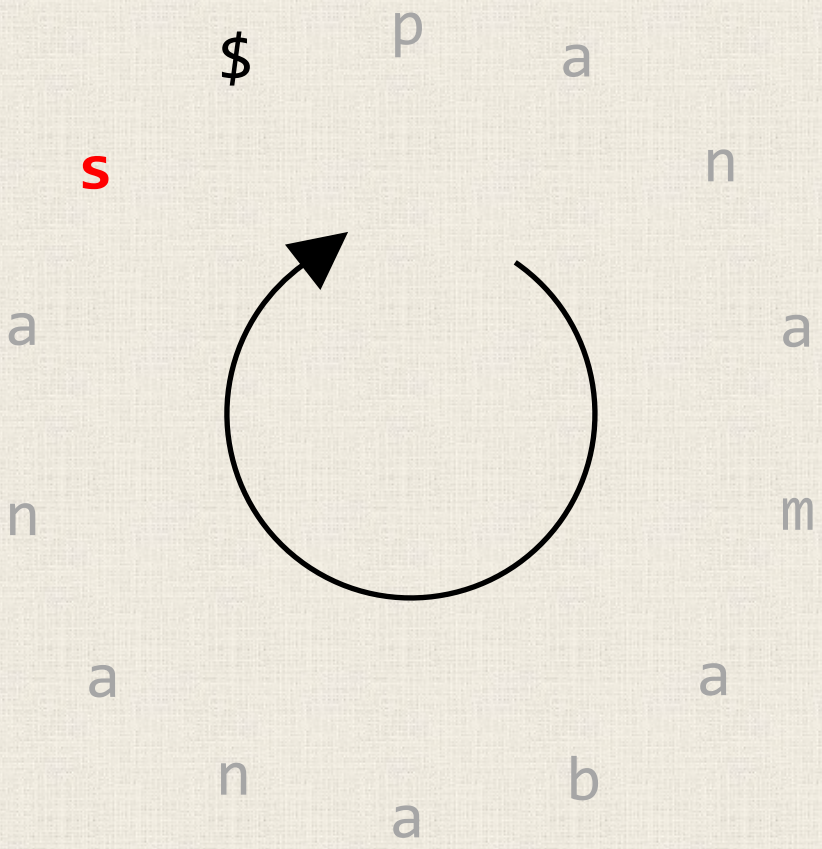
Efficient BWT Decompression

$ 1	panamabananas	s 1
a 1	bananas$panam	a 1
a 2	mabananas$pan	n 1
a 3	namabananas$p	a 4
a 4	nanas$panamab	n 2
a 5	nas$panamaban	a 6
a 6	s$panamabanan	n 3
b 1	ananas$panama	a 1
m 1	abananas$pana	a 2
n 1	amabananas$pa	a 3
n 2	anas$panamaba	a 5
n 3	as$panamabana	a 4
p 1	anamabananas$	$ 1
s 1	$panamabanana	a 6



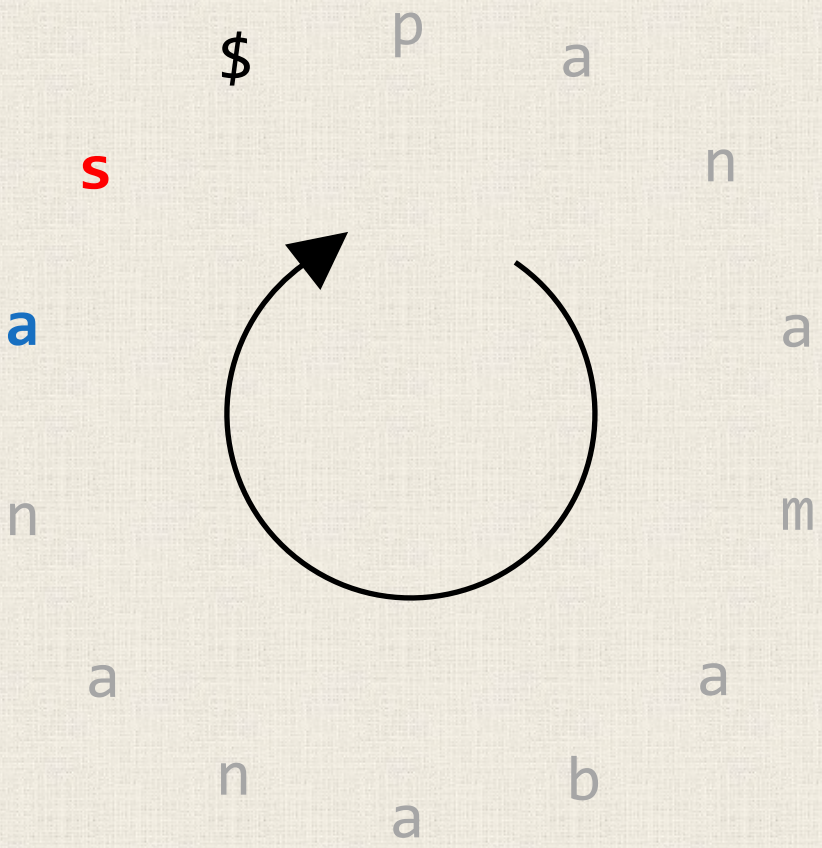
Efficient BWT Decompression

$	1	p	a	n	a	m	a	b	a	n	a	s	1
a	1	b	a	n	a	n	a	s	$	p	a	n	a	m	1
a	2	m	a	b	a	n	a	n	a	s	$	p	a	n	1
a	3	n	a	m	a	b	a	n	a	n	a	s	$	p	1
a	4	n	a	n	a	s	$	p	a	n	a	m	a	b	1
a	5	n	a	s	$	p	a	n	a	m	a	b	a	n	2
a	6	s	$	p	a	n	a	m	a	b	a	n	a	n	3
b	1	a	n	a	n	a	s	$	p	a	n	a	m	a	1
m	1	a	b	a	n	a	n	a	s	$	p	a	n	a	2
n	1	a	m	a	b	a	n	a	n	a	s	$	p	a	3
n	2	a	n	a	s	$	p	a	n	a	m	a	b	a	4
n	3	a	s	$	p	a	n	a	m	a	b	a	n	a	5
p	1	a	n	a	m	a	b	a	n	a	n	a	s	$	1
s	1	$	p	a	n	a	m	a	b	a	n	a	n	a	6



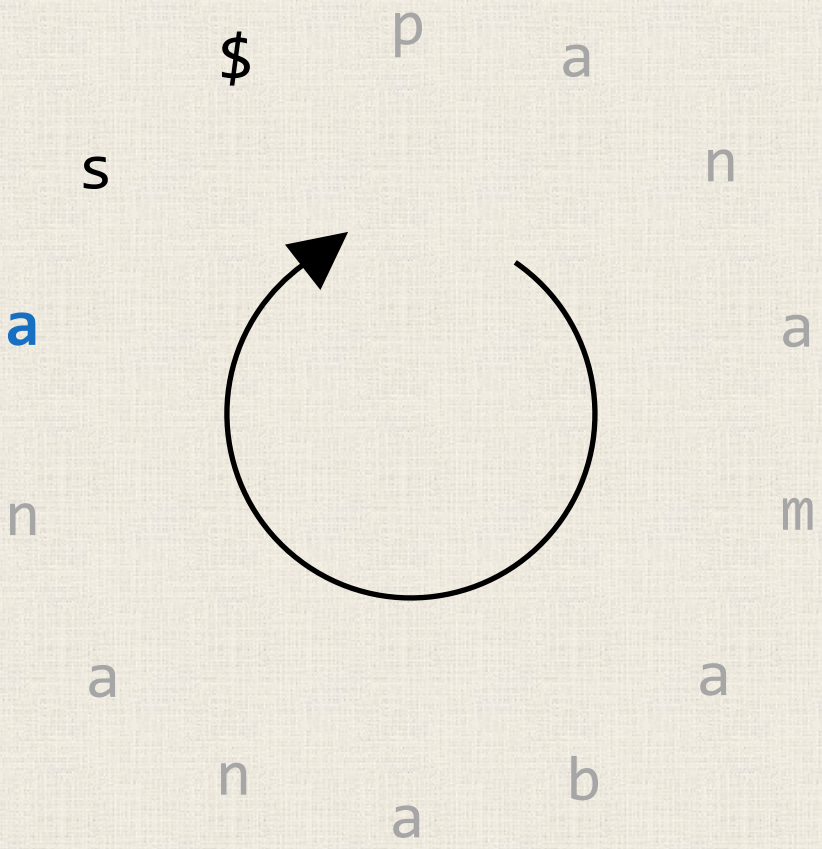
Efficient BWT Decompression

$	p	a	n	a	m	a	b	a	n	a	s	1
a	b	a	n	a	n	a	s	$	p	a	n	a	1
a	m	a	b	a	n	a	n	a	s	$	p	a	2
a	n	a	m	a	b	a	n	a	n	a	s	$	3
a	n	a	s	$	p	a	n	a	m	a	b	a	4
a	s	$	p	a	n	a	m	a	b	a	n	a	5
a	n	a	n	a	s	$	p	a	n	a	m	a	6
b	a	n	a	n	a	s	$	p	a	n	a	m	1
m	a	b	a	n	a	n	a	s	$	p	a	n	2
n	a	m	a	b	a	n	a	n	a	s	$	p	3
n	a	n	a	s	$	p	a	n	a	m	a	b	4
n	a	s	$	p	a	n	a	m	a	b	a	n	5
p	a	n	a	m	a	b	a	n	a	n	a	s	6
s	$	p	a	n	a	m	a	b	a	n	a	a	1



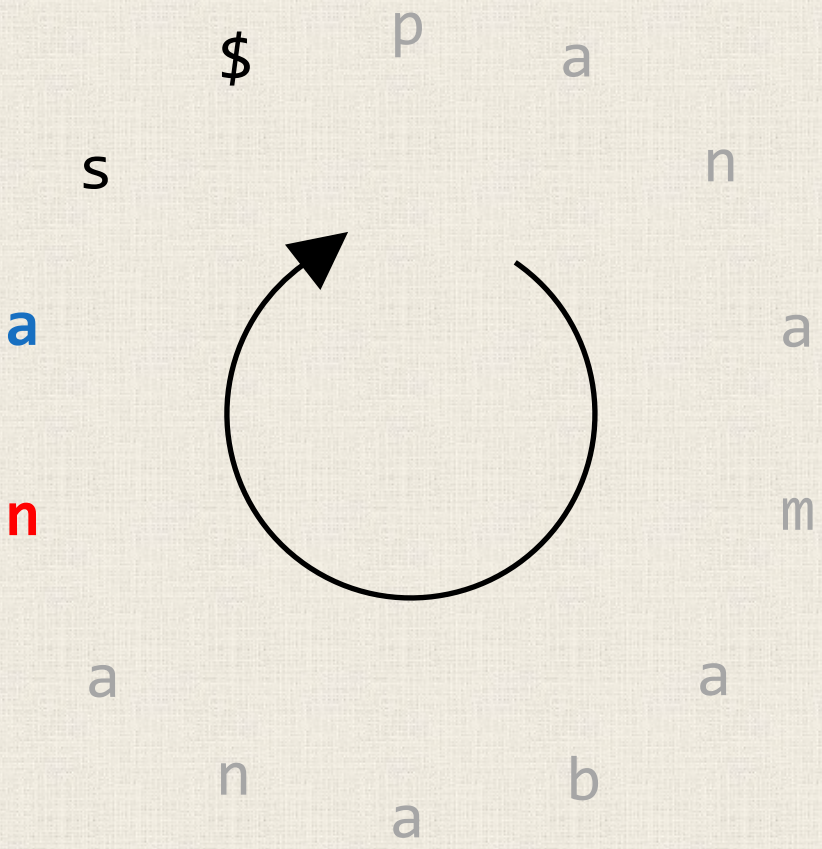
Efficient BWT Decompression

$ 1	panamabananas	s 1
a 1	bananas$panam	m 1
a 2	mabananas$pan	n 1
a 3	namabananas$p	p 1
a 4	nanas$panamab	b 1
a 5	nas$panamaban	n 2
a 6	s$panamabanan	n 3
b 1	ananas$panama	a 1
m 1	abananas$pana	a 2
n 1	amabananas$pa	a 3
n 2	anas$panamaba	a 4
n 3	as$panamaban	a 5
p 1	anamabananas$	$ 1
s 1	$panamabanan	**a 6**



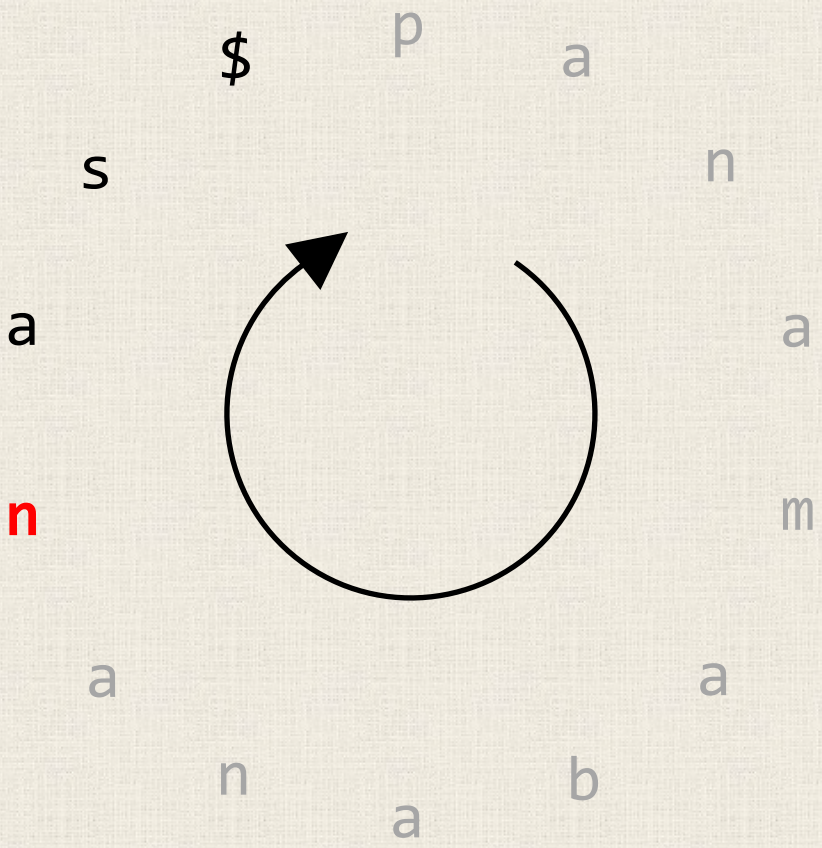
Efficient BWT Decompression

$ 1	panamabananas	s 1
a 1	bananas$panam	m 1
a 2	mabananas$pan	n 1
a 3	namabananas$p	p 1
a 4	nanas$panamab	b 1
a 5	nas$panamaban	n 2
a 6	s$panamabana	**n 3**
b 1	ananas$panama	a 1
m 1	abananas$pana	a 2
n 1	amabananas$pa	a 3
n 2	anas$panamaba	a 4
n 3	as$panamaban	a 5
p 1	anamabananas$	$ 1
s 1	$panamabanana	a 6



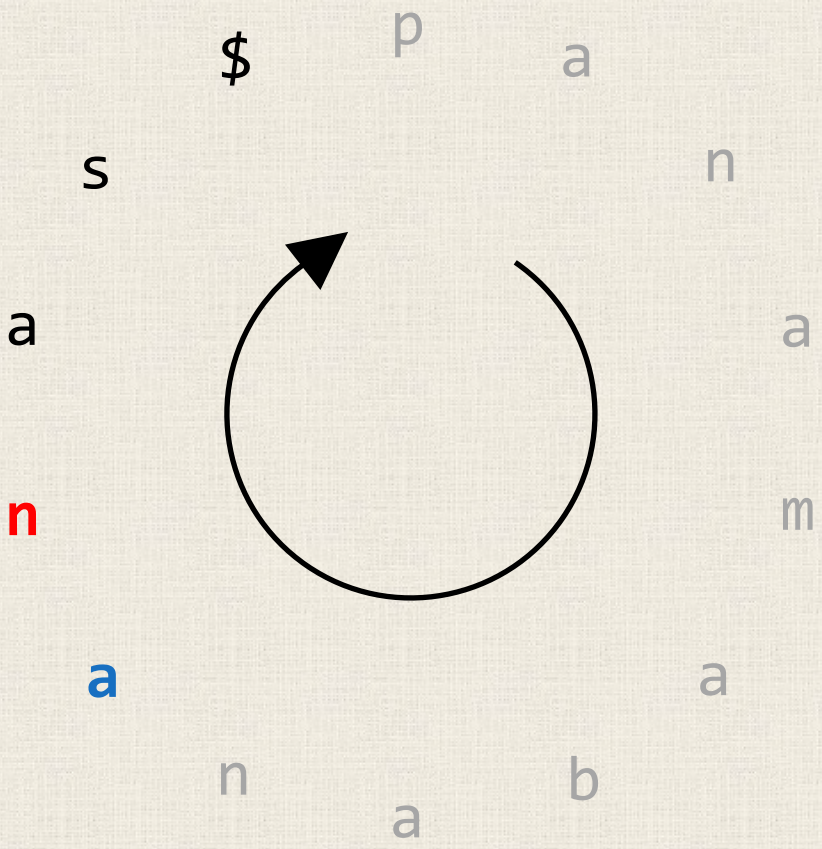
Efficient BWT Decompression

$_1	panamabananas	s_1
a_1	bananas$panam	m_1
a_2	mabananas$pan	n_1
a_3	namabananas$p	p_1
a_4	nanas$panamab	b_1
a_5	nas$panamaban	n_2
a_6	s$panamabanan	**n_3**
b_1	ananas$panama	a_1
m_1	abananas$pana	a_2
n_1	amabananas$pa	a_3
n_2	anas$panamaba	a_4
n_3	as$panamaban	a_5
p_1	anamabananas$	$_1
s_1	$panamabanana	a_6



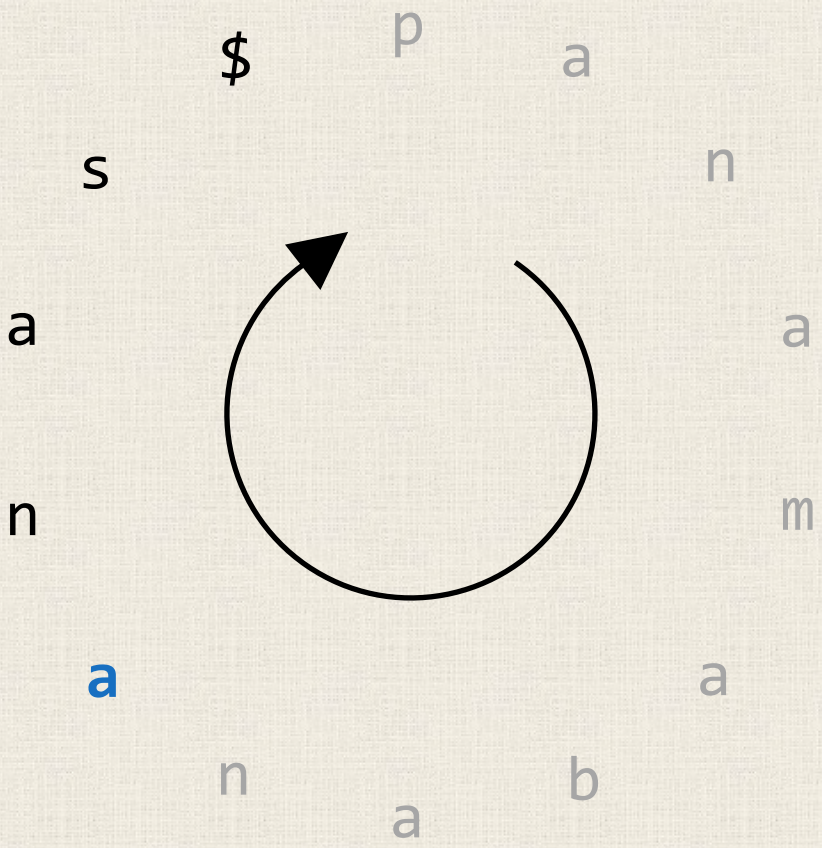
Efficient BWT Decompression

$ 1	panamabananas	s 1
a 1	bananas$panam	m 1
a 2	mabananas$pan	n 1
a 3	namabananas$p	p 1
a 4	nanas$panamab	b 1
a 5	nas$panamaban	n 2
a 6	s$panamabanan	n 3
b 1	ananas$panama	a 1
m 1	abananas$pana	a 2
n 1	amabananas$pa	a 3
n 2	anas$panamaba	a 4
n 3	as$panamabana	a 5
p 1	anamabananas$	$ 1
s 1	$panamabanana	a 6



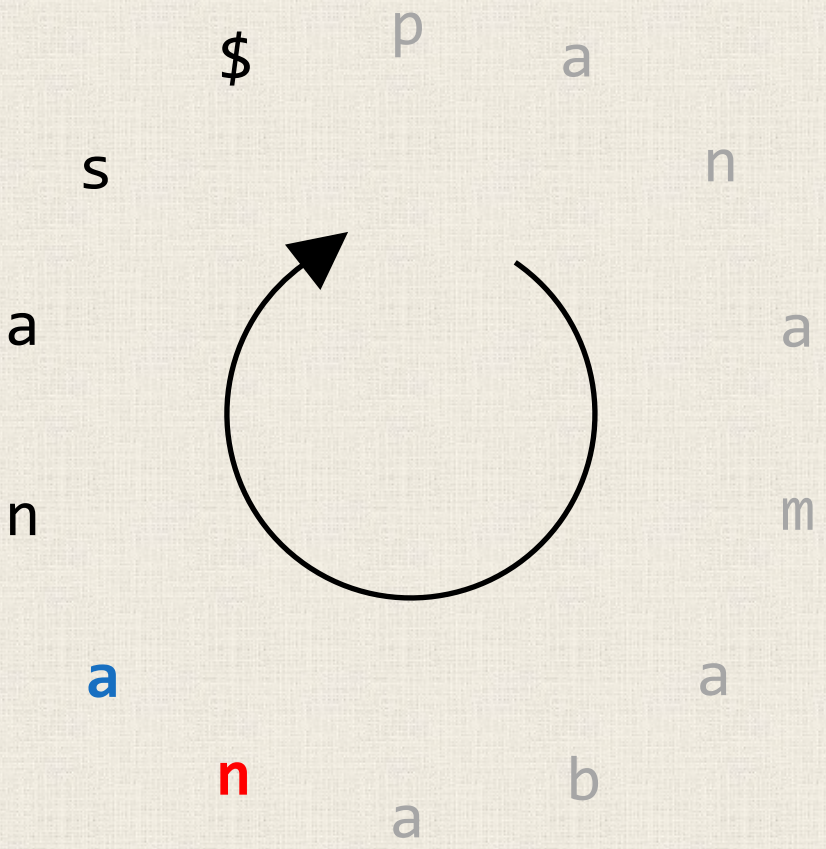
Efficient BWT Decompression

$	p	a	n	a	m	a	b	a	n	a	s	1
a	b	a	n	a	n	a	s	$	p	a	n	a	m	1
a	m	a	b	a	n	a	n	a	s	$	p	a	n	1
a	n	a	m	a	b	a	n	a	n	a	s	$	p	1
a	n	a	n	a	s	$	p	a	n	a	m	a	b	1
a	n	a	s	$	p	a	n	a	m	a	b	a	n	2
a	s	$	p	a	n	a	m	a	b	a	n	a	n	3
b	a	n	a	n	a	s	$	p	a	n	a	m	a	1
m	a	b	a	n	a	n	a	s	$	p	a	n	a	2
n	a	m	a	b	a	n	a	n	a	s	$	p	a	3
n	a	n	a	s	$	p	a	n	a	m	a	b	a	4
n	a	s	$	p	a	n	a	m	a	b	a	n	**a**	5
p	a	n	a	m	a	b	a	n	a	n	a	s	$	1
s	$	p	a	n	a	m	a	b	a	n	a	n	a	6



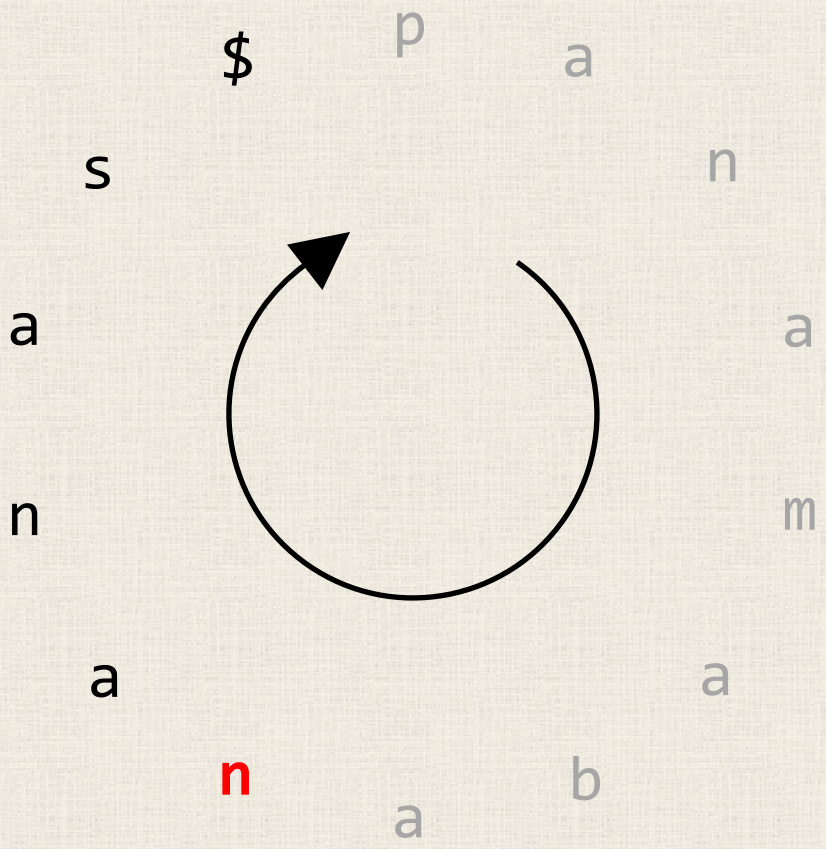
Efficient BWT Decompression

$	p	a	n	a	m	a	b	a	n	a	n	a	s	1
a	b	a	n	a	n	a	s	$	p	a	n	a	m	1
a	m	a	b	a	n	a	n	a	s	$	p	a	n	1
a	n	a	m	a	b	a	n	a	n	a	s	$	p	1
a	n	a	n	a	s	$	p	a	n	a	m	a	b	1
a	n	a	s	$	p	a	n	a	m	a	b	a	**n**	**2**
a	s	$	p	a	n	a	m	a	b	a	n	a	n	3
b	a	n	a	n	a	s	$	p	a	n	a	m	a	1
m	a	b	a	n	a	n	a	s	$	p	a	n	a	2
n	a	m	a	b	a	n	a	n	a	s	$	p	a	3
n	a	n	a	s	$	p	a	n	a	m	a	b	a	4
n	a	s	$	p	a	n	a	m	a	b	a	n	a	5
p	a	n	a	m	a	b	a	n	a	n	a	s	$	1
s	$	p	a	n	a	m	a	b	a	n	a	n	a	6



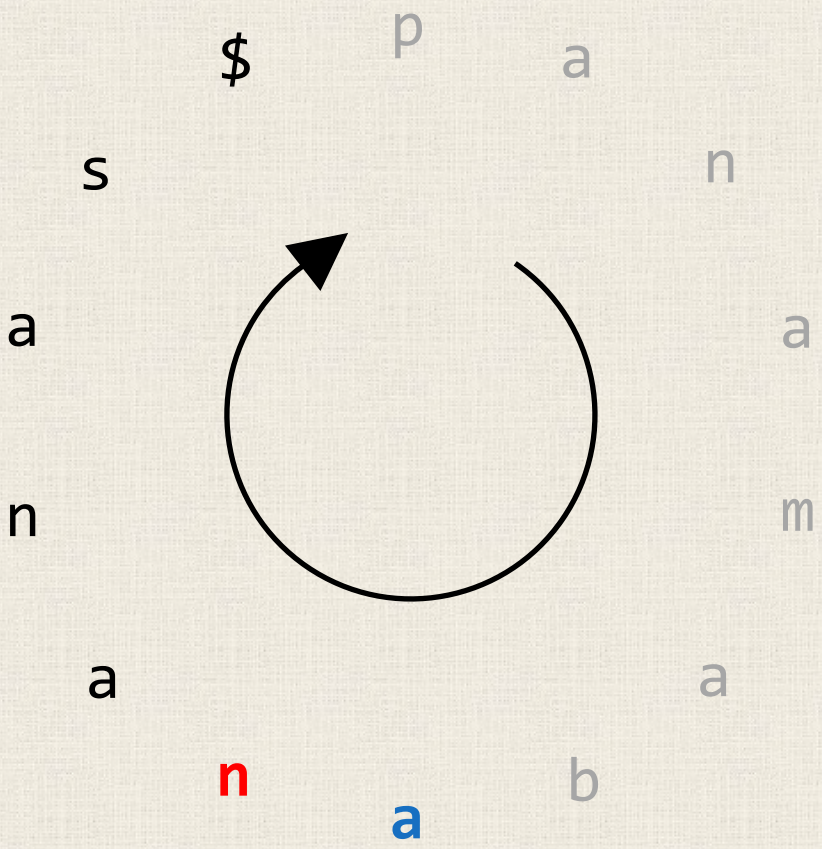
Efficient BWT Decompression

$_1$	panamabananas$_1$
a$_1$	bananas$panam$_1$
a$_2$	mabananaspan_1$
a$_3$	namabananas$p$_1$
a$_4$	nanas$panamab$_1$
a$_5$	nas$panamaban$_2$
a$_6$	s$panamabanan$_3$
b$_1$	ananas$panama$_1$
m$_1$	abananas$pana$_2$
n$_1$	amabananaspa_3$
n$_2$	anas$panamaba$_4$
n$_3$	as$panamaban$_5$
p$_1$	anamabananas$$_1$
s$_1$	$panamabanana$_6$



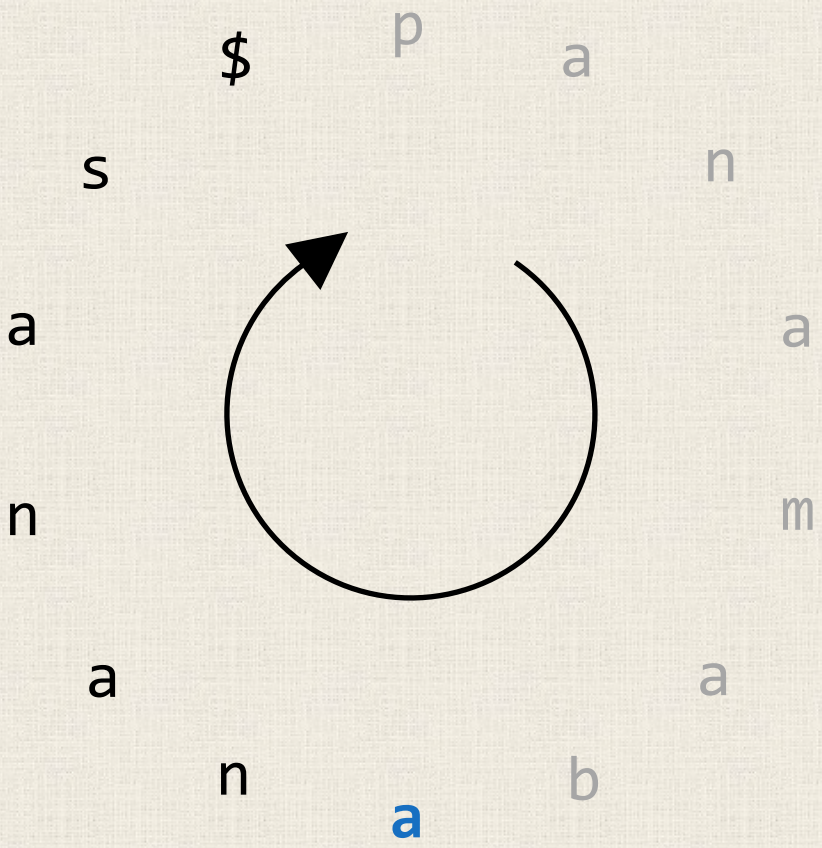
Efficient BWT Decompression

$	panamabananas	$_1
a	bananas$panam	m_1
a	mabananas$pan	n_1
a	namabananas$p	p_1
a	nanas$panamab	b_1
a	nas$panamaban	n_2
a	s$panamabanan	n_3
b	ananas$panama	a_1
m	abananas$pana	a_2
n	amabananas$pa	a_3
n	**anas$panamab**	**a_4**
n	as$panamaban	a_5
p	anamabananas$	$_1
s	$panamabanan	a_6



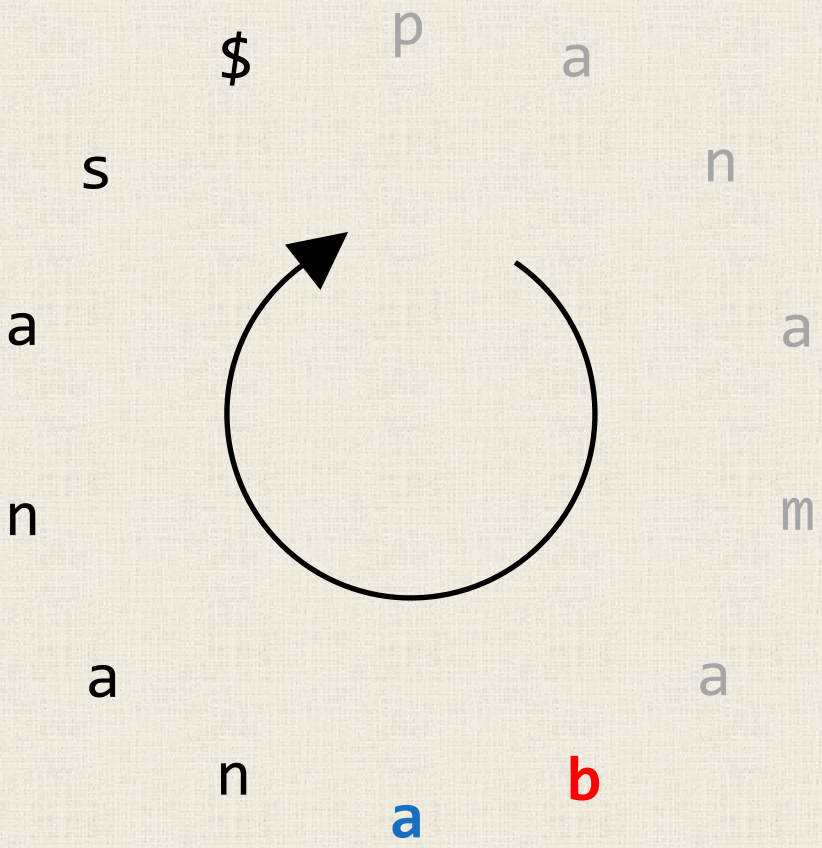
Efficient BWT Decompression

$_1$panamabananas$_1$	
a$_1$bananas$panam$_1$	
a$_2$mabananaspan_1$	
a$_3$namabananas$p$_1$	
a$_4$nanas$panamab$_1$	
a$_5$nas$panamaban$_2$	
a$_6$s$panamabanan$_3$	
b$_1$ananas$panama$_1$	
m$_1$abananas$pana$_2$	
n$_1$amabananaspa_3$	
n$_2$anas$panamab**a$_4$**	
n$_3$as$panamaban$a$_5$	
p$_1$anamabananas$$_1$	
s$_1$$panamabanana$_6$	



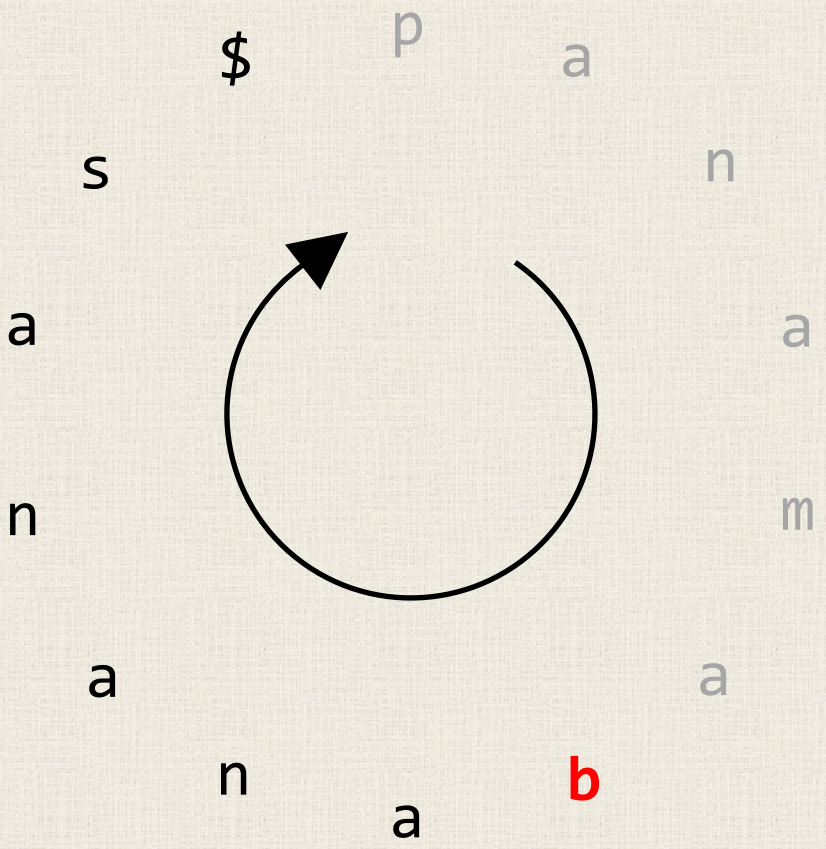
Efficient BWT Decompression

$ 1	panamabananas	s 1
a 1	bananas$panam	m 1
a 2	mabananas$pan	n 1
a 3	namabananas$p	p 1
a 4	nanas$panama	**b 1**
a 5	nas$panamaban	n 2
a 6	s$panamabanan	n 3
b 1	ananas$panama	a 1
m 1	abananas$pana	a 2
n 1	amabananas$pa	a 3
n 2	anas$panamaba	a 4
n 3	as$panamabana	a 5
p 1	anamabananas$	$ 1
s 1	$panamabanana	a 6



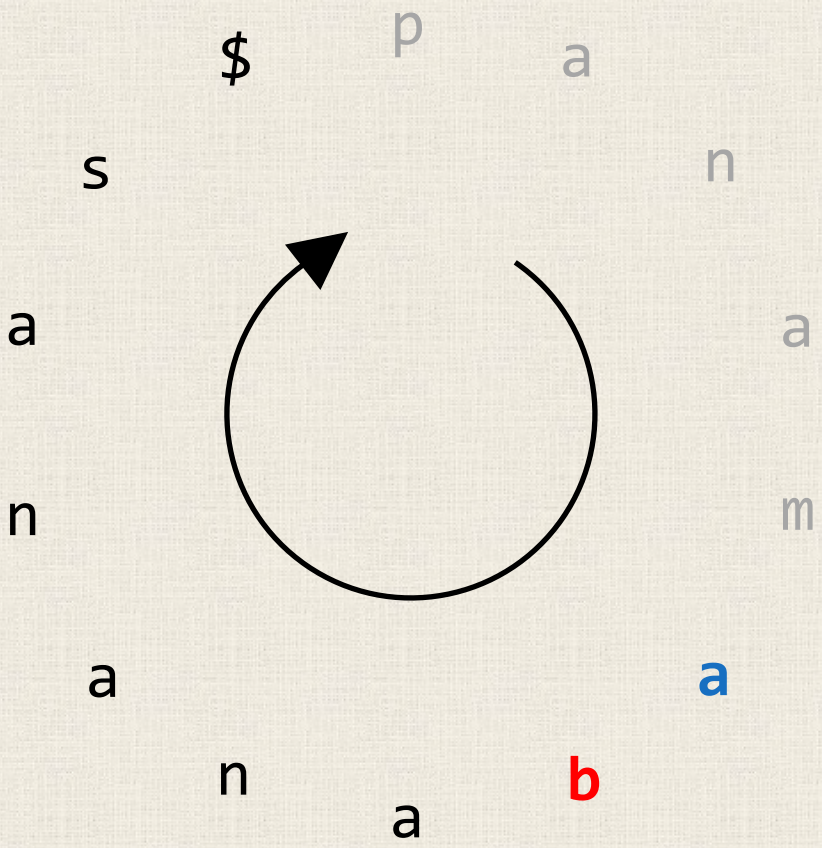
Efficient BWT Decompression

$_1	panamabananas	s_1
a_1	bananas$panam	m_1
a_2	mabananas$pan	n_1
a_3	namabananas$p	p_1
a_4	nanas$panamab	**b_1**
a_5	nas$panamaban	n_2
a_6	s$panamabanan	n_3
b_1	ananas$panama	a_1
m_1	abananas$pana	a_2
n_1	amabananas$pa	a_3
n_2	anas$panamaba	a_4
n_3	as$panamaban	a_5
p_1	anamabananas$	$_1
s_1	$panamabanana	a_6



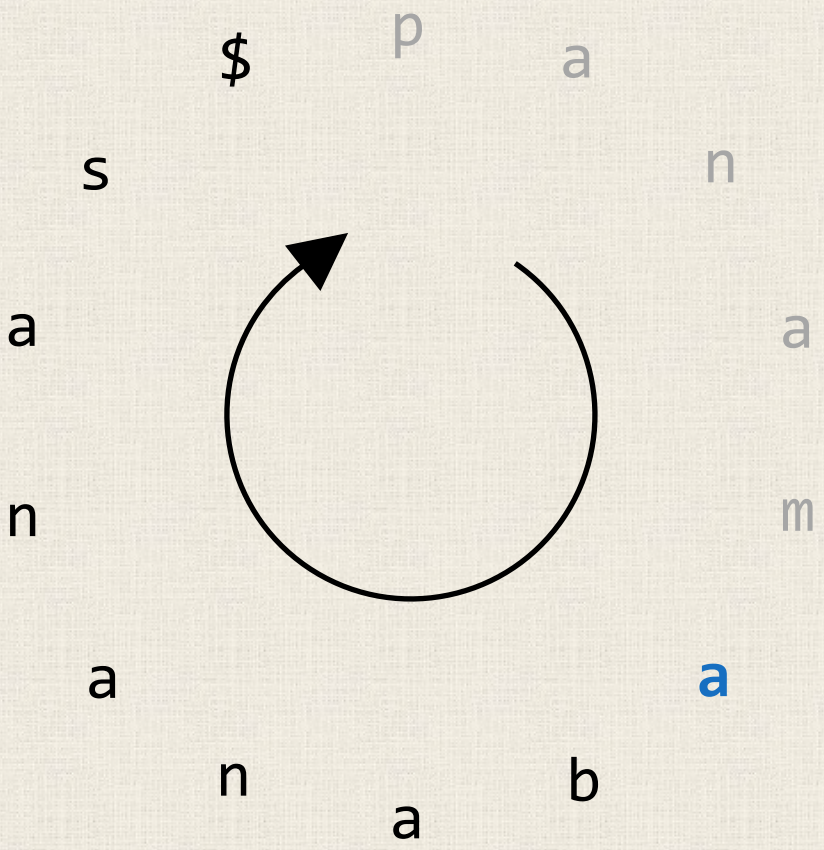
Efficient BWT Decompression

$$_1$$	panamabananas$$_1$$
a$$_1$$	bananas$panam$$_1$$
a$$_2$$	mabananas$pan$$_1$$
a$$_3$$	namabananas$p$$_1$$
a$$_4$$	nanas$panamab$$_1$$
a$$_5$$	nas$panamaban$$_2$$
a$$_6$$	s$panamabanan$$_3$$
b$$_1$$	ananas$panam**a$$_1$$**
m$$_1$$	abananas$pana$$_2$$
n$$_1$$	amabananas$pa$$_3$$
n$$_2$$	anas$panamaba$$_4$$
n$$_3$$	as$panamaban$$_5$$
p$$_1$$	anamabananas$$_1$$
s$$_1$$	$panamabanana$$_6$$



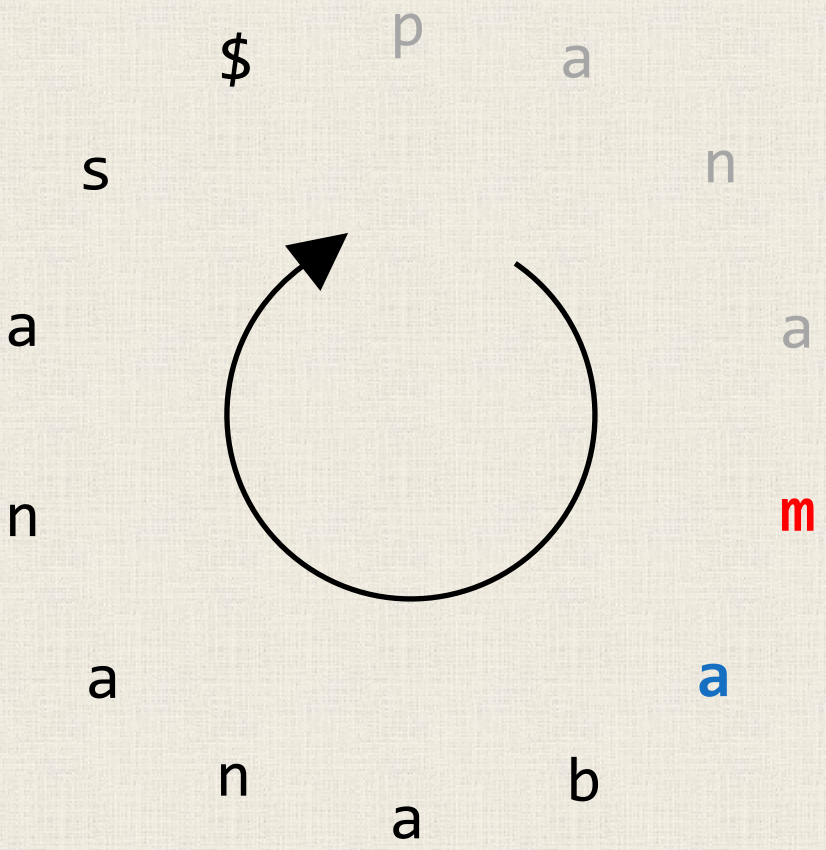
Efficient BWT Decompression

$_1$panamabananas$_1$	
a$_1$bananas$panam**m$_1$**	
a$_2$mabananas$pan**n$_1$**	
a$_3$namabananas$p**p$_1$**	
a$_4$nanas$panama**b$_1$**	
a$_5$nas$panamaba**n$_2$**	
a$_6$s$panamaban**n$_3$**	
b$_1$ananas$panam**a$_1$**	
m$_1$abananas$pan**a$_2$**	
n$_1$amabananas$p**a$_3$**	
n$_2$anas$panamab**a$_4$**	
n$_3$as$panamaban**a$_5$**	
p$_1$anamabananas**$$_1$**	
s$_1$$panamabanan**a$_6$**	



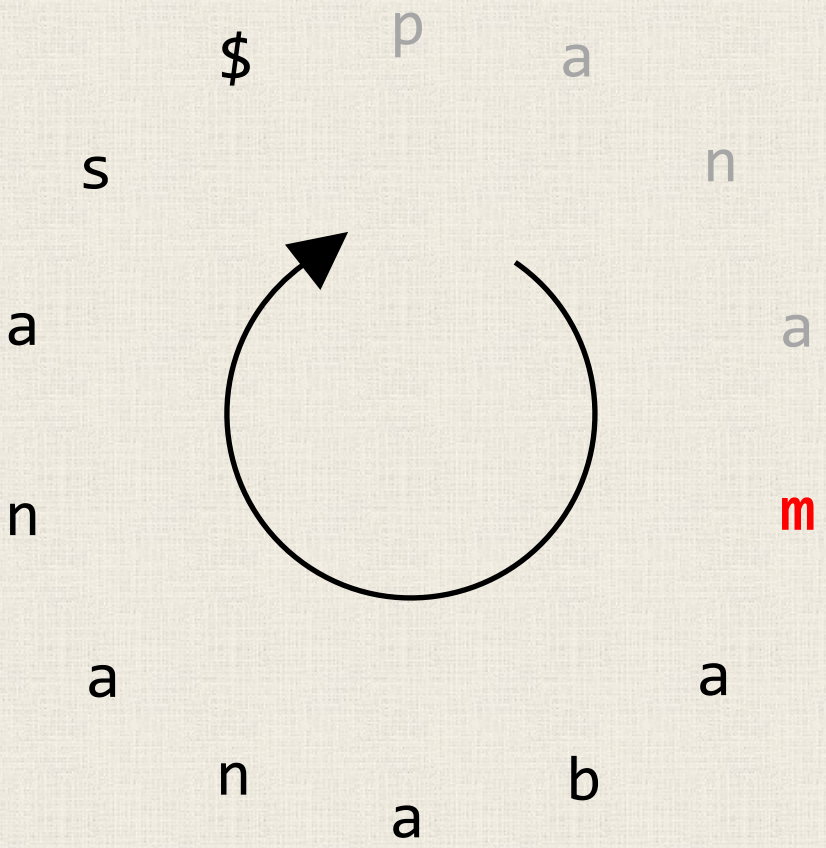
Efficient BWT Decompression

$$_1$$	panamabananas$$_1$$
a$$_1$$	bananas$panam$_1$$
a$$_2$$	mabananaspan_1$$
a$$_3$$	namabananas$p$_1$$
a$$_4$$	nanas$panamab$_1$$
a$$_5$$	nas$panamaban$_2$$
a$$_6$$	s$panamabanan$_3$$
b$$_1$$	ananas$panama$_1$$
m$$_1$$	abananas$pana$_2$$
n$$_1$$	amabananaspa_3$$
n$$_2$$	anas$panamaba$_4$$
n$$_3$$	as$panamaban$_5$$
p$$_1$$	anamabananas$$_1$$
s$$_1$$	$panamabanana$_6$$



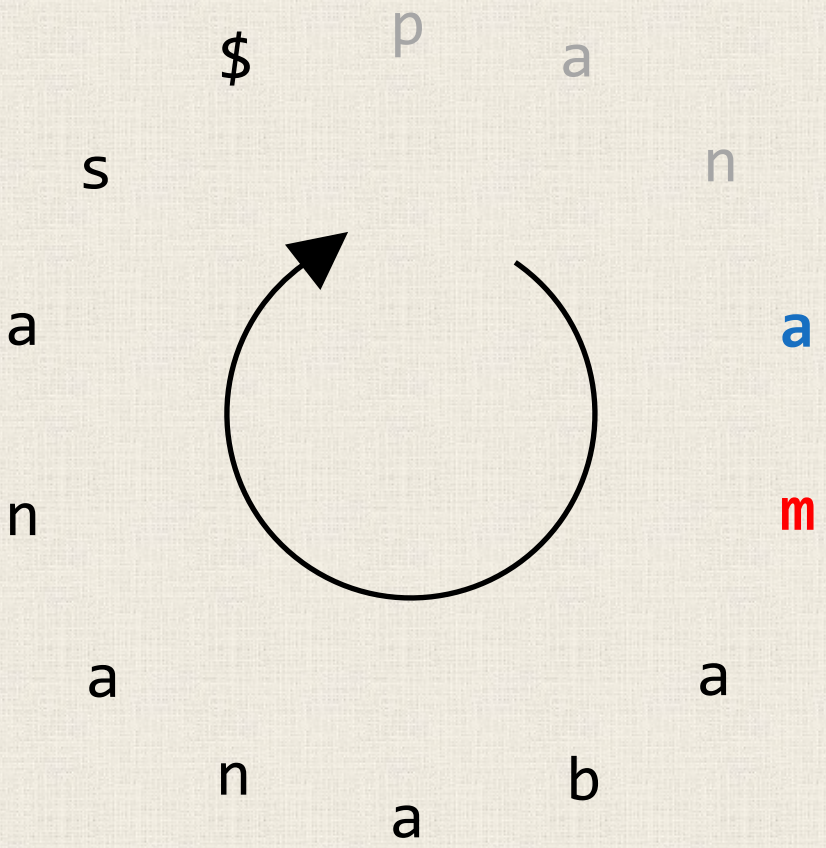
Efficient BWT Decompression

$ 1	panamabananas	s 1
a 1	bananas$panam	m 1
a 2	mabananas$pan	n 1
a 3	namabananas$p	p 1
a 4	nanas$panamab	b 1
a 5	nas$panamaban	n 2
a 6	s$panamabanan	n 3
b 1	ananas$panama	a 1
m 1	abananas$pana	a 2
n 1	amabananas$pa	a 3
n 2	anas$panamaba	a 4
n 3	as$panamabana	a 5
p 1	anamabananas$	$ 1
s 1	$panamabanana	a 6



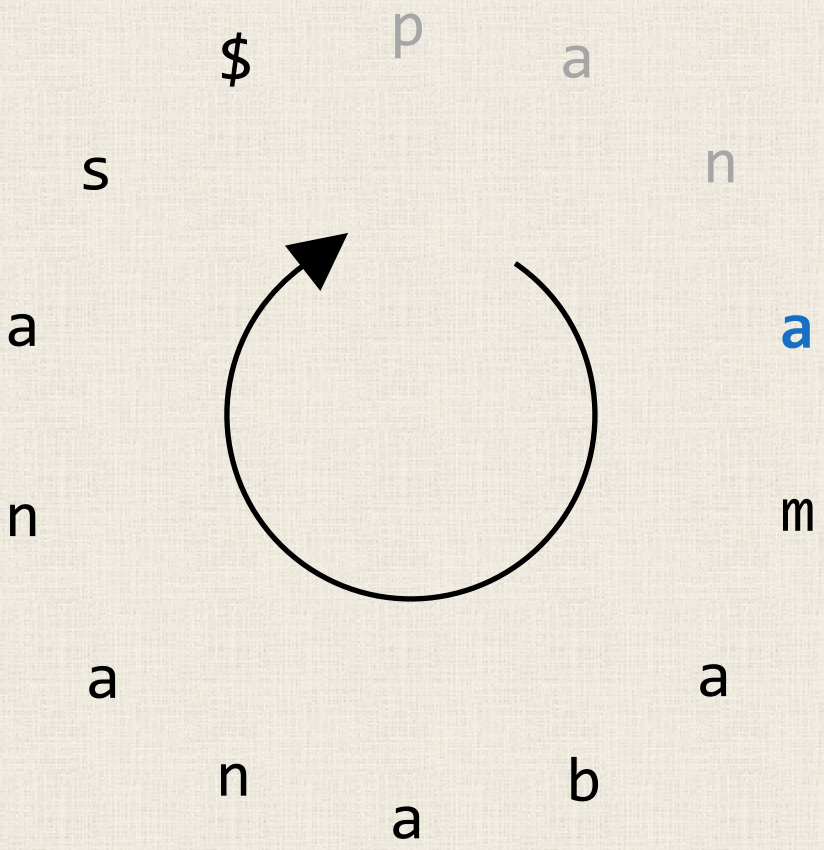
Efficient BWT Decompression

$	panamabananas	$
a	bananas$panam	m
a	mabananas$pan	n
a	namabananas$p	p
a	nanas$panamab	b
a	nas$panamaban	n
a	s$panamabanan	n
b	ananas$panama	a
m	abananas$pana	a
n	amabananas$pa	a
n	anas$panamaba	a
n	as$panamabana	a
p	anamabananas$	$
s	$panamabanana	a



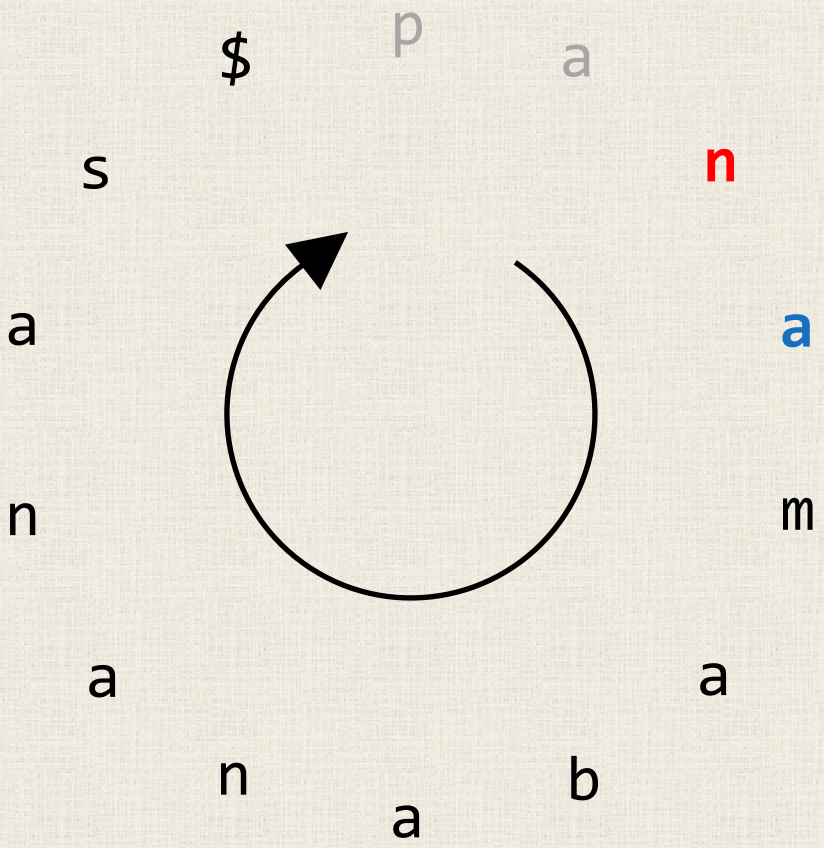
Efficient BWT Decompression

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



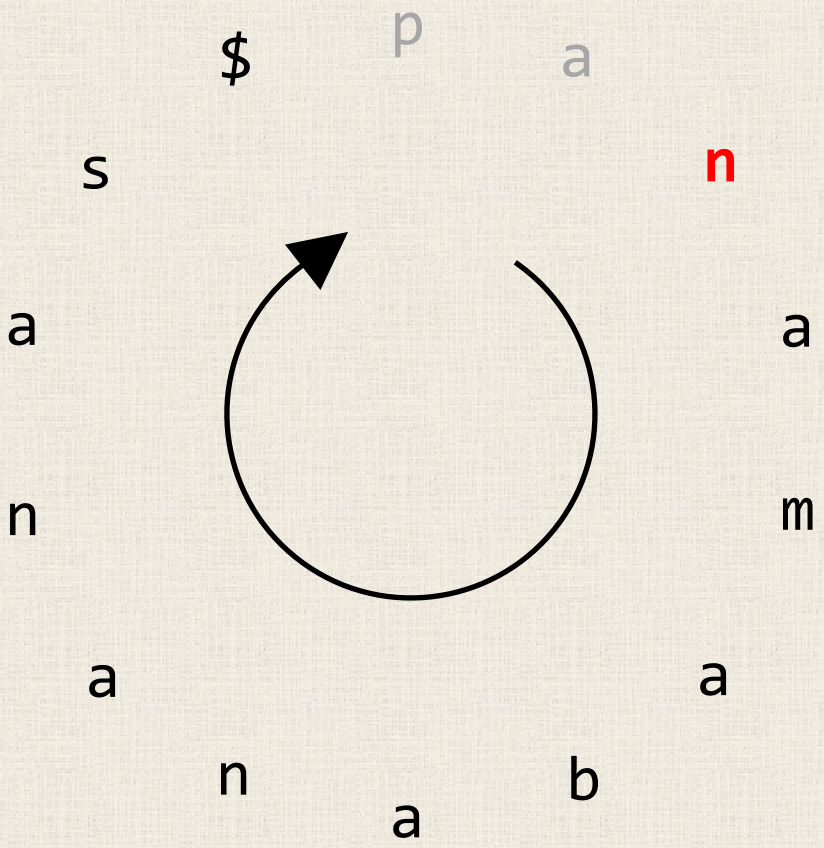
Efficient BWT Decompression

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



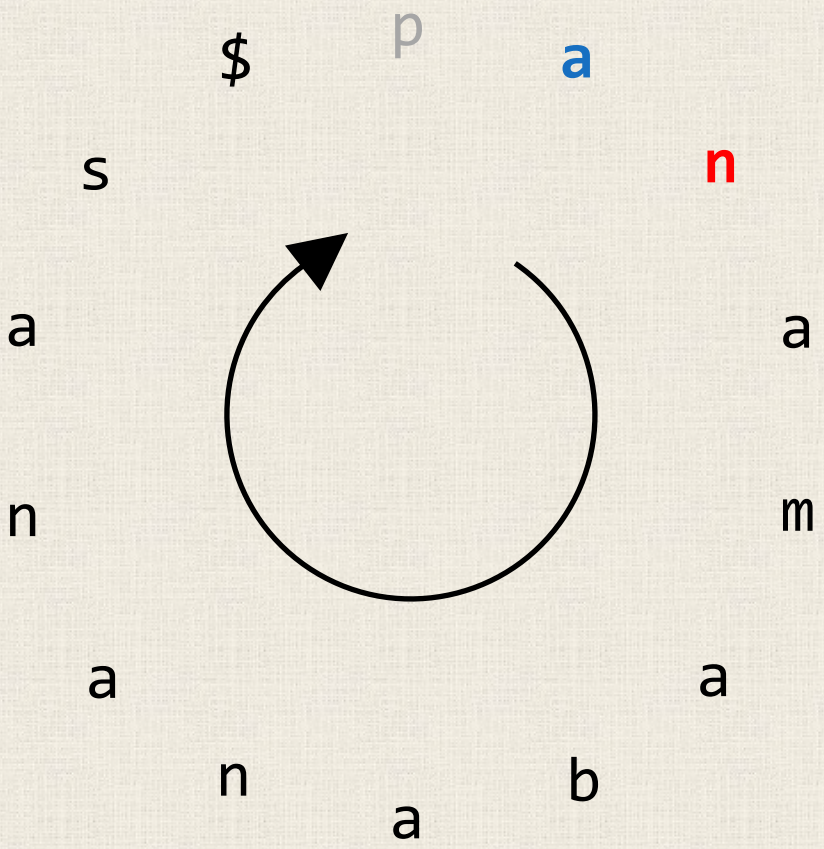
Efficient BWT Decompression

$ 1	panamabananas	s 1
a 1	bananas$panam	m 1
a 2	mabananas$pan	n 1
a 3	namabananas$p	p 1
a 4	nanas$panamab	b 1
a 5	nas$panamaban	n 2
a 6	s$panamabanan	n 3
b 1	ananas$panama	a 1
m 1	abananas$pana	a 2
n 1	amabananas$pa	a 3
n 2	anas$panamaba	a 4
n 3	as$panamaban	a 5
p 1	anamabananas$	$ 1
s 1	$panamabanan	a 6



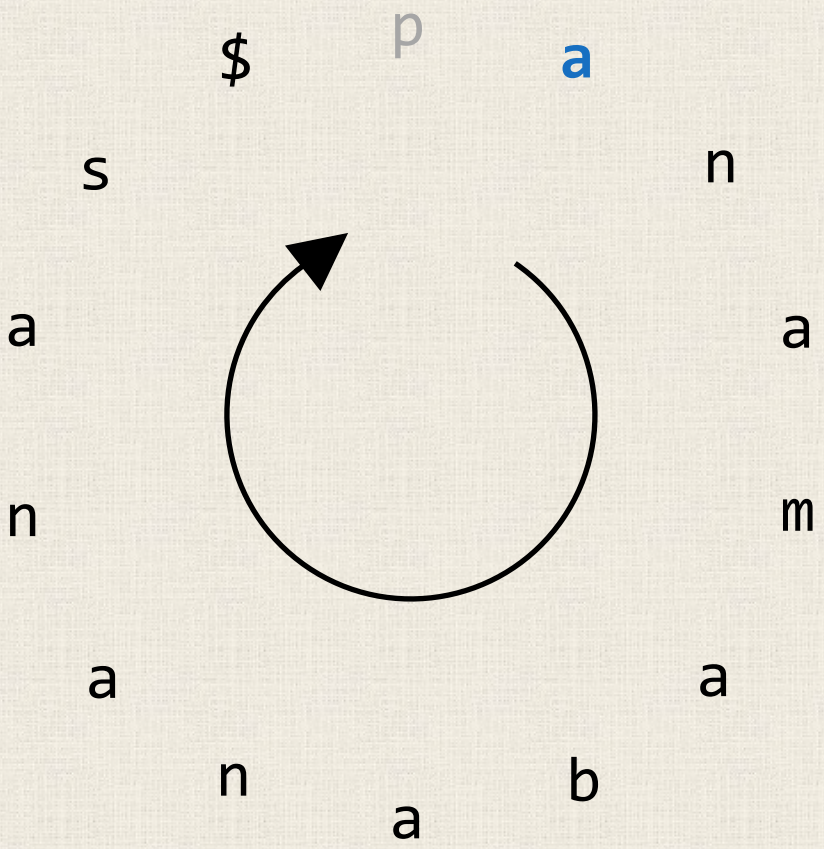
Efficient BWT Decompression

$ 1	panamabananas	s 1
a 1	bananas$panam	m 1
a 2	mabananas$pan	n 1
a 3	namabananas$p	p 1
a 4	nanas$panamab	b 1
a 5	nas$panamaban	n 2
a 6	s$panamabanan	n 3
b 1	ananas$panama	a 1
m 1	abananas$pana	a 2
n 1	amabananas$pa	a 3
n 2	anas$panamaba	a 4
n 3	as$panamaban	a 5
p 1	anamabananas$	$ 1
s 1	$panamabanan	a 6



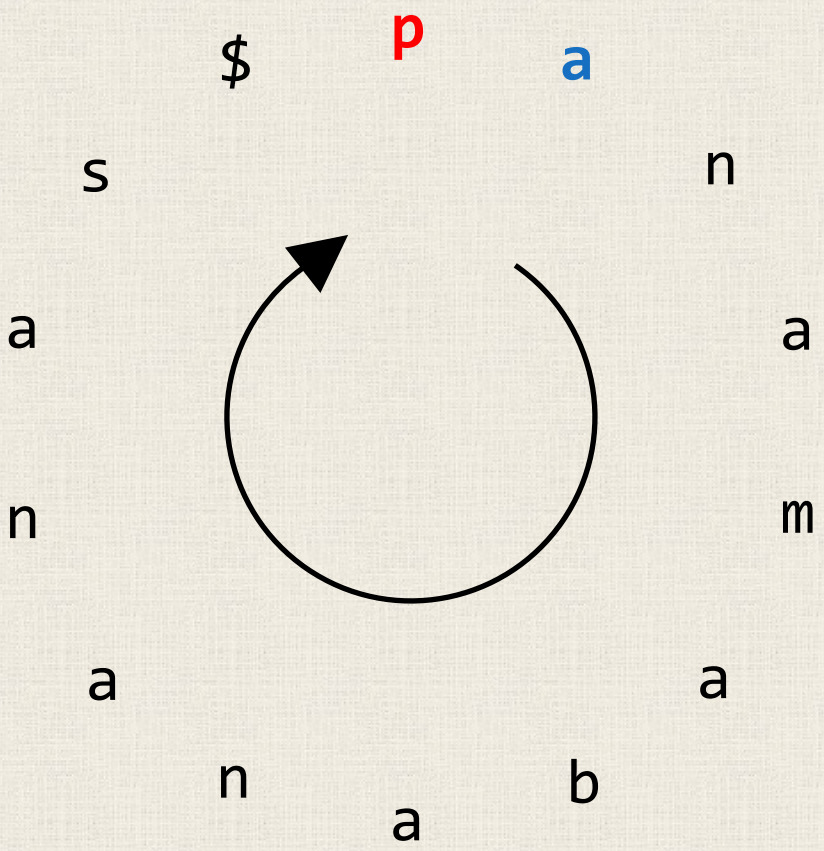
Efficient BWT Decompression

$$_1$$	panamabananas$$_1$$
a$$_1$$	bananas$panam$$_1$$
a$$_2$$	mabananas$pan$$_1$$
a$$_3$$	namabananas$p$$_1$$
a$$_4$$	nanas$panamab$$_1$$
a$$_5$$	nas$panamaban$$_2$$
a$$_6$$	s$panamabanan$$_3$$
b$$_1$$	ananas$panama$$_1$$
m$$_1$$	abananas$pana$$_2$$
n$$_1$$	amabananas$p**a$$_3$$**
n$$_2$$	anas$panamaba$$_4$$
n$$_3$$	as$panamaban$$_5$$
p$$_1$$	anamabananas$$_1$$
s$$_1$$	$panamabanana$$_6$$



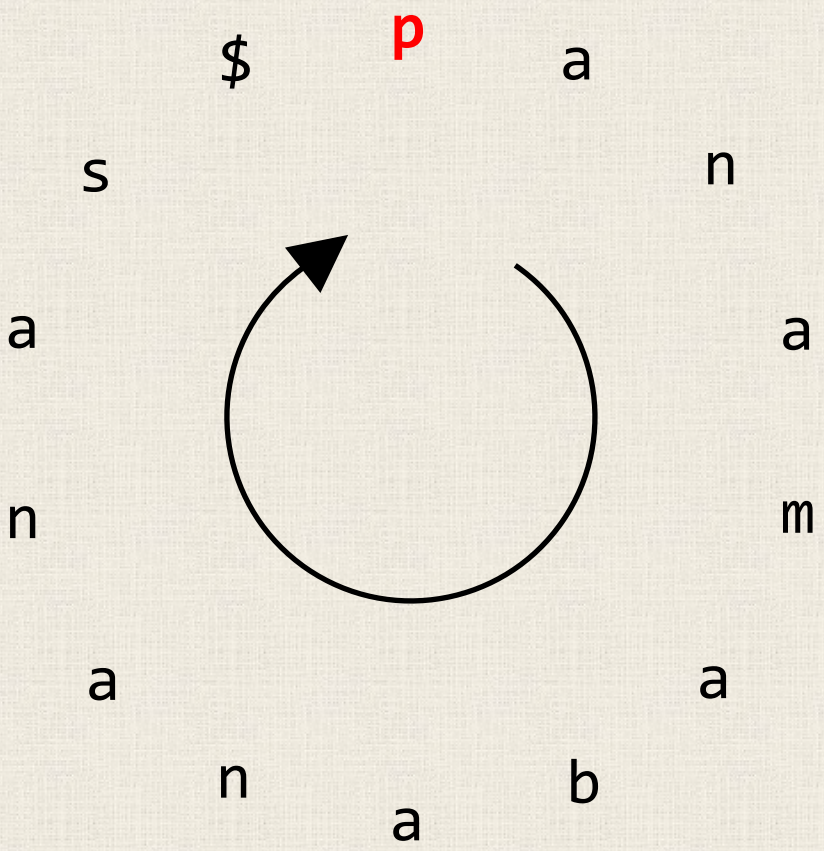
Efficient BWT Decompression

$ 1	panamabananas	s 1
a 1	bananas$panam	m 1
a 2	mabananas$pan	n 1
a 3	namabananas$p	p 1
a 4	nanas$panamab	b 1
a 5	nas$panamaban	n 2
a 6	s$panamabanan	n 3
b 1	ananas$panama	a 1
m 1	abananas$pana	a 2
n 1	amabananas$pa	a 3
n 2	anas$panamaba	a 4
n 3	as$panamaban	a 5
p 1	anamabananas$	$ 1
s 1	$panamabanan	a 6



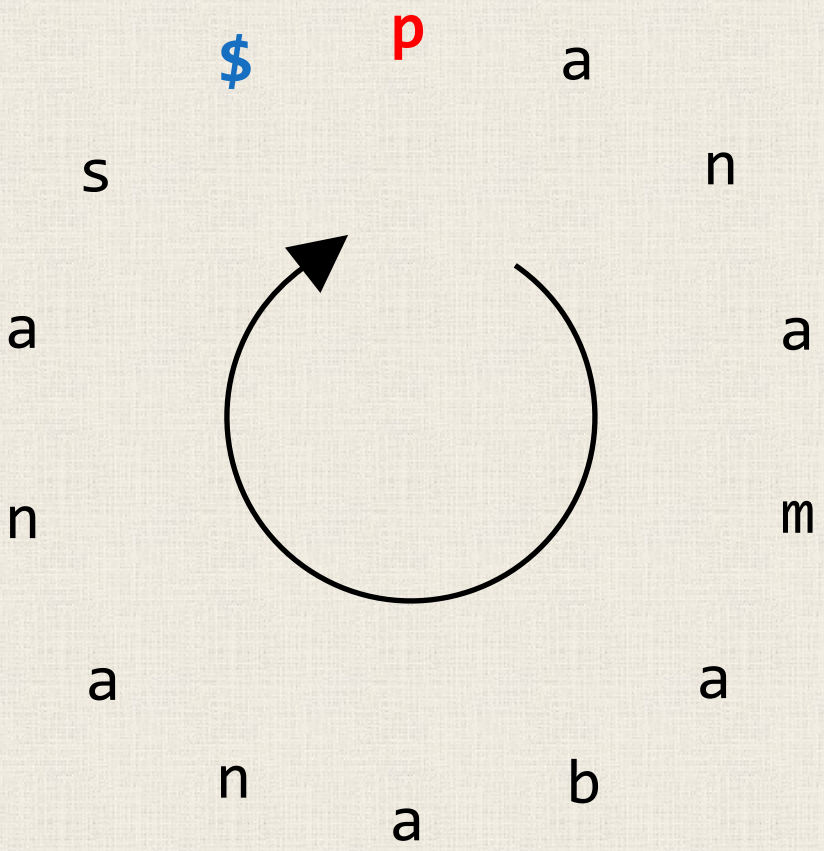
Efficient BWT Decompression

$	p	a	n	a	m	a	b	a	n	a	s	1
a	b	a	n	a	n	a	s	$	p	a	n	a	m	1
a	m	a	b	a	n	a	n	a	s	$	p	a	n	1
a	n	a	m	a	b	a	n	a	n	a	s	$	**p**	**1**
a	n	a	n	a	s	$	p	a	n	a	m	a	b	1
a	n	a	s	$	p	a	n	a	m	a	b	a	n	2
a	s	$	p	a	n	a	m	a	b	a	n	a	n	3
b	a	n	a	n	a	s	$	p	a	n	a	m	a	1
m	a	b	a	n	a	n	a	s	$	p	a	n	a	2
n	a	m	a	b	a	n	a	n	a	s	$	p	a	3
n	a	n	a	s	$	p	a	n	a	m	a	b	a	4
n	a	s	$	p	a	n	a	m	a	b	a	n	a	5
p	**1**	a	n	a	m	a	b	a	n	a	n	a	s	$	1
s	1	$	p	a	n	a	m	a	b	a	n	a	n	a	6



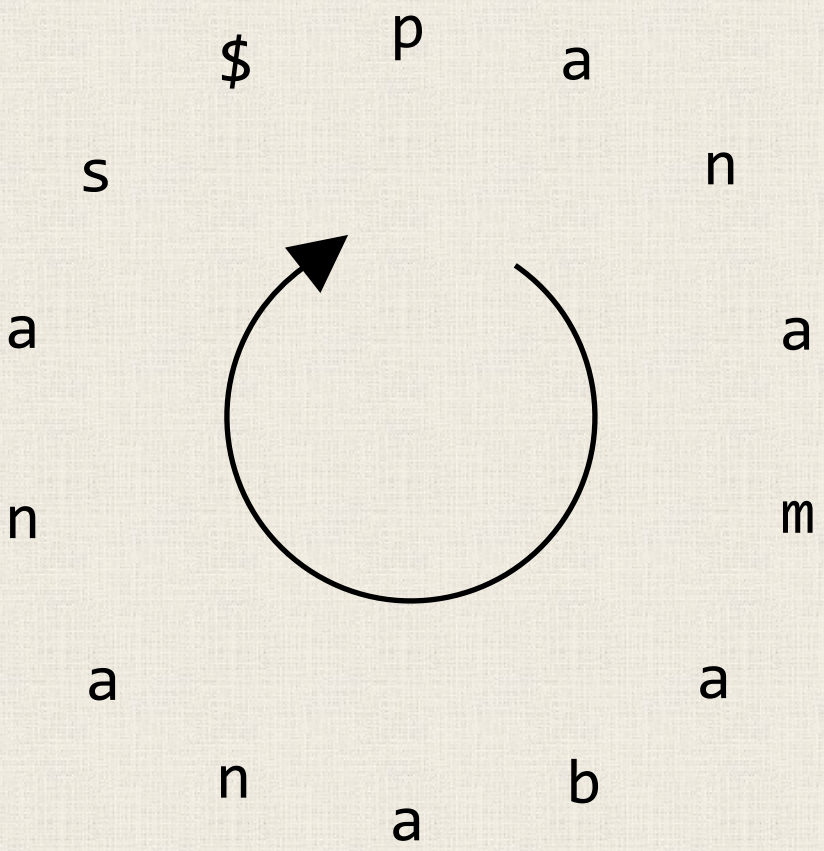
Efficient BWT Decompression

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



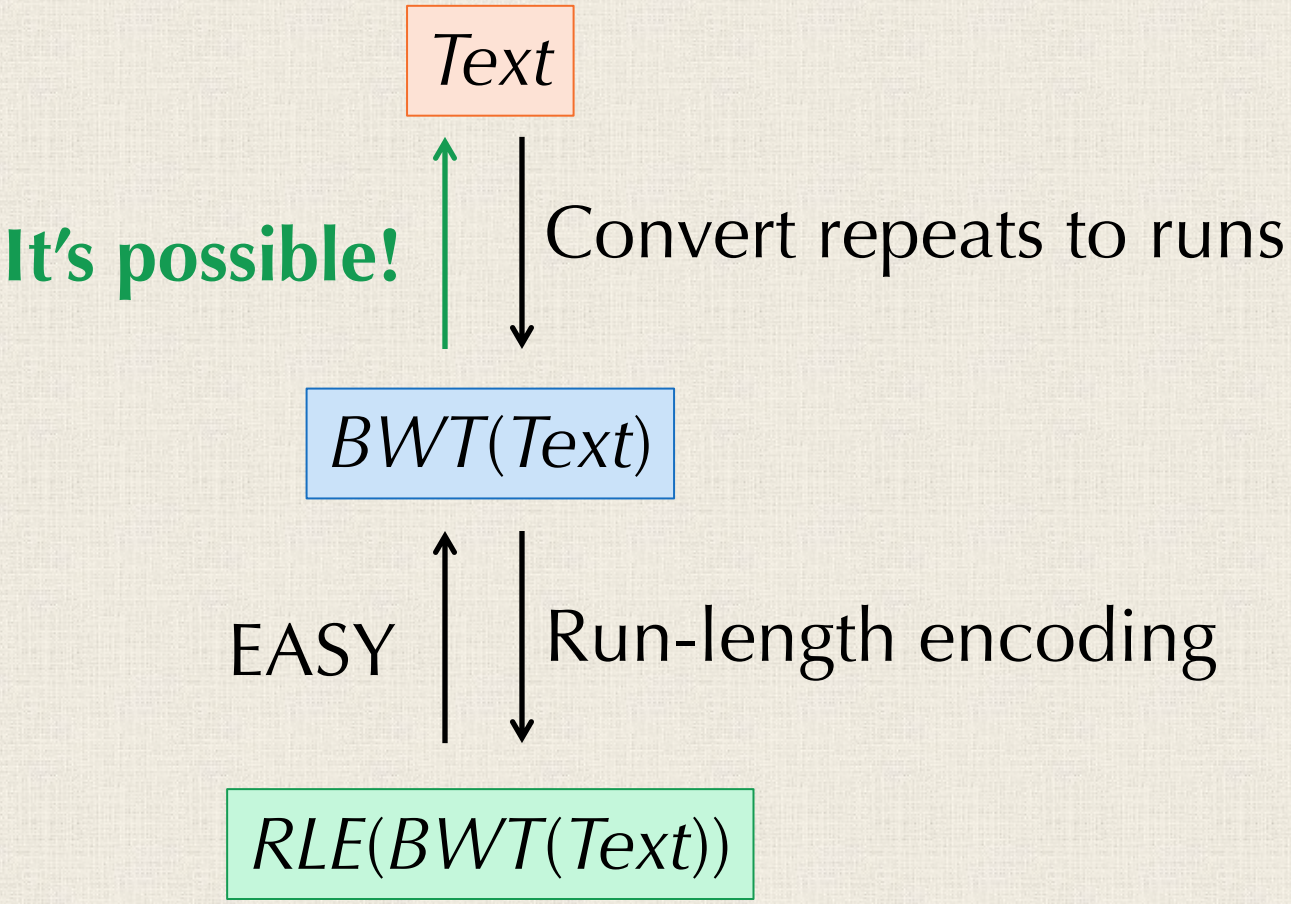
Efficient BWT Decompression

$_1	panamabananas	s_1
a_1	bananas$panam	m_1
a_2	mabananas$pan	n_1
a_3	namabananas$p	p_1
a_4	nanas$panamab	b_1
a_5	nas$panamaban	n_2
a_6	s$panamabanan	n_3
b_1	ananas$panama	a_1
m_1	abananas$pana	a_2
n_1	amabananas$pa	a_3
n_2	anas$panamaba	a_4
n_3	as$panamaban	a_5
p_1	anamabananas$	$_1
s_1	$panamabanan	a_6



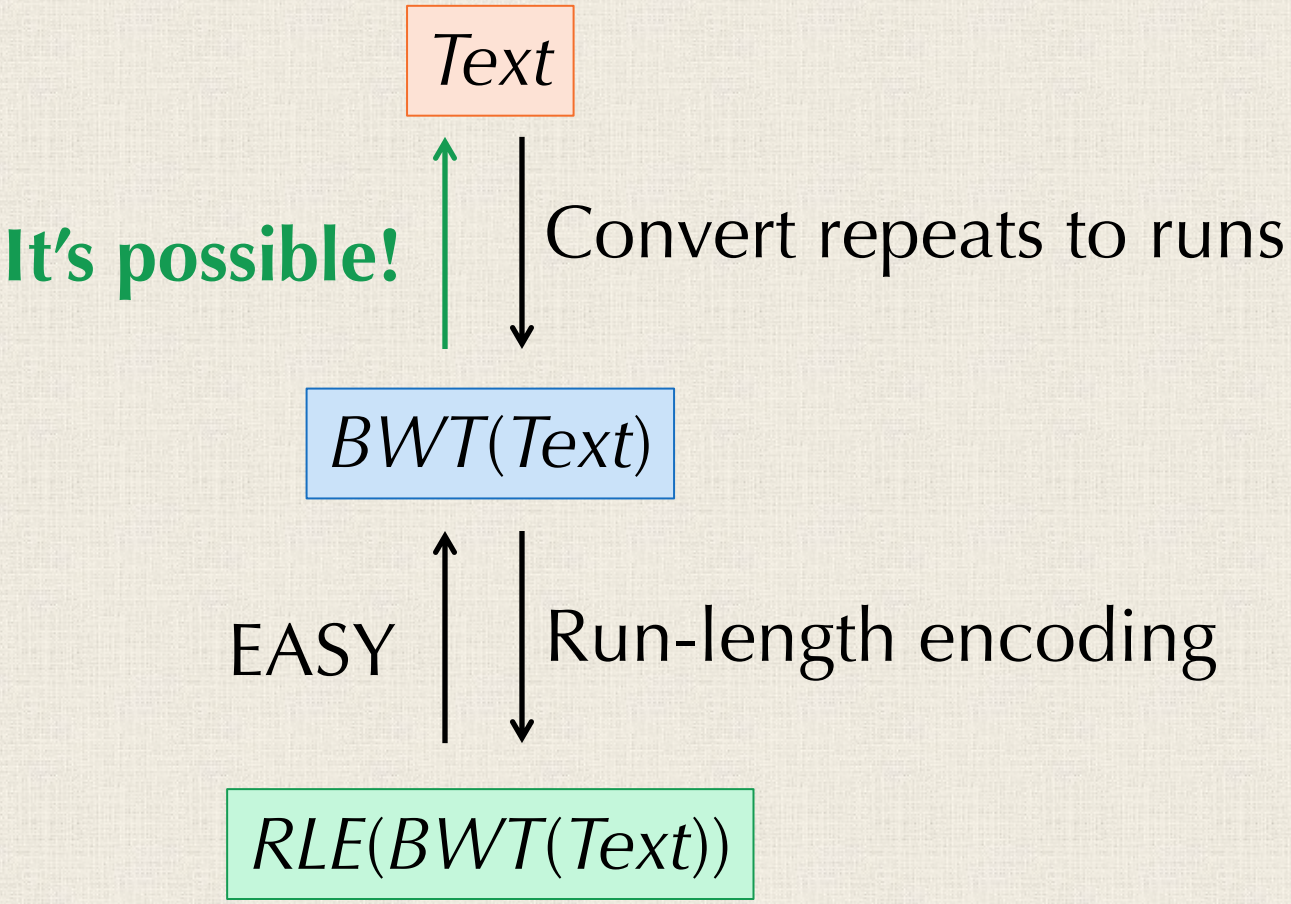
Memory: $2|Text| = O(|Text|)$.

So We Can Decompress BWT ...



So We Can Decompress BWT ...

But What About Pattern Matching?



PATTERN MATCHING WITH BURROWS-WHEELER

Recalling Our Goal

Pattern Matching with Suffix Array:

- Runtime: O(|*Text*| + |*Patterns*|)
- Memory: O(|*Text*|)
- Problem: suffix tree takes ~20 x |*Text*| space, but suffix array takes ~4 x |*Text*| space.

Recalling Our Goal

Pattern Matching with Suffix Array:

- Runtime: O(|*Text*| + |*Patterns*|)
- Memory: O(|*Text*|)
- Problem: suffix tree takes ~20 x |*Text*| space, but suffix array takes ~4 x |*Text*| space.

Can we use BWT(*Text*) as our data structure instead?

Pattern Matches "Clump" at Start of *M*(*Text*)

$_1$panamabananas$_1$
a$_1$bananas$panam$_1$
a$_2$mabananaspan_1$
a$_3$namabananas$p$_1$
a$_4$nanas$panamab$_1$
a$_5$nas$panamaban$_2$
a$_6$s$panamabanan$_3$
b$_1$ananas$panama$_1$
m$_1$abananas$pana$_2$
n$_1$amabananaspa_3$
n$_2$anas$panamaba$_4$
n$_3$as$panamabana$_5$
p$_1$anamabananas$_1$
s$_1$$panamabanana$_6$

Connecting *M*(*Text*) to Suffix Array

$_1$panamabananas$_1$
a$_1$bananas$panam$_1$
a$_2$mabananaspan_1$
a$_3$namabananas$p$_1$
a$_4$nanas$panamab$_1$
a$_5$nas$panamaban$_2$
a$_6$s$panamabanan$_3$
b$_1$ananas$panama$_1$
m$_1$abananas$pana$_2$
n$_1$amabananaspa_3$
n$_2$anas$panamaba$_4$
n$_3$as$panamabana$_5$
p$_1$anamabananas$$_1$
s$_1$$panamabanana$_6$

Sorted Suffixes	Suffix Array
$	13
abananas	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

Connecting *M*(*Text*) to Suffix Array

$_1$panamabananas$_1$
a$_1$bananas$panam$_1$
a$_2$mabananaspan_1$
a$_3$namabananas$p$_1$
a$_4$nanas$panamab$_1$
a$_5$nas$panamaban$_2$
a$_6$s$panamabanan$_3$
b$_1$ananas$panama$_1$
m$_1$abananas$pana$_2$
n$_1$amabananaspa_3$
n$_2$anas$panamaba$_4$
n$_3$as$panamabana$_5$
p$_1$anamabananas$$_1$
s$_1$$panamabanana$_6$

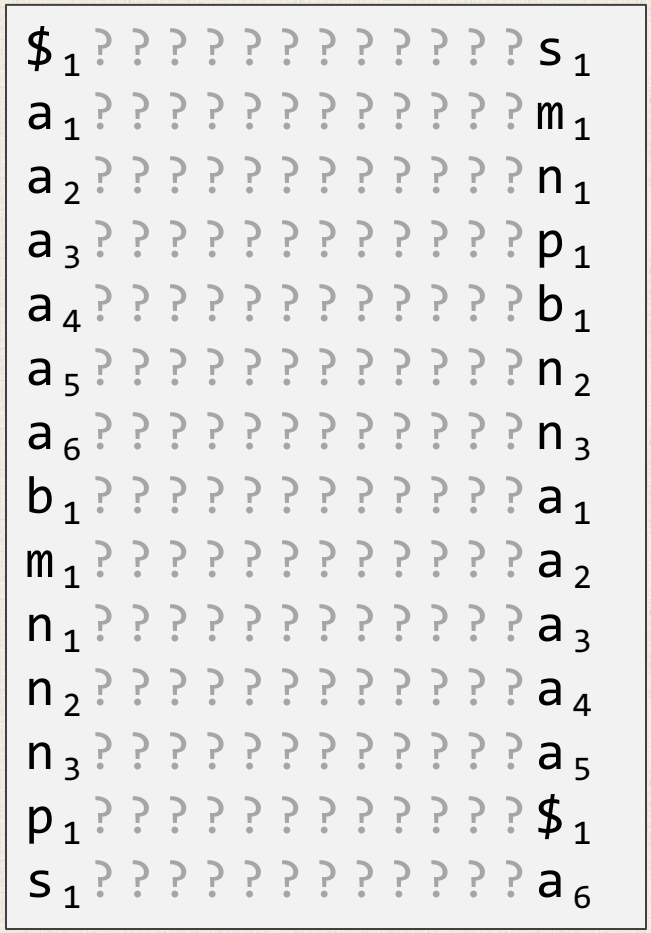
We could find pattern matches easily if we had all the suffixes, but we would need $O(|Text|^2)$ space…

Connecting *M*(*Text*) to Suffix Array

We could find pattern matches easily if we had all the suffixes, but we would need $O(|Text|^2)$ space…

We are going to pattern match using just two columns of *M*(*Text*): *FirstColumn* and *LastColumn*

We Match Patterns *Backward*



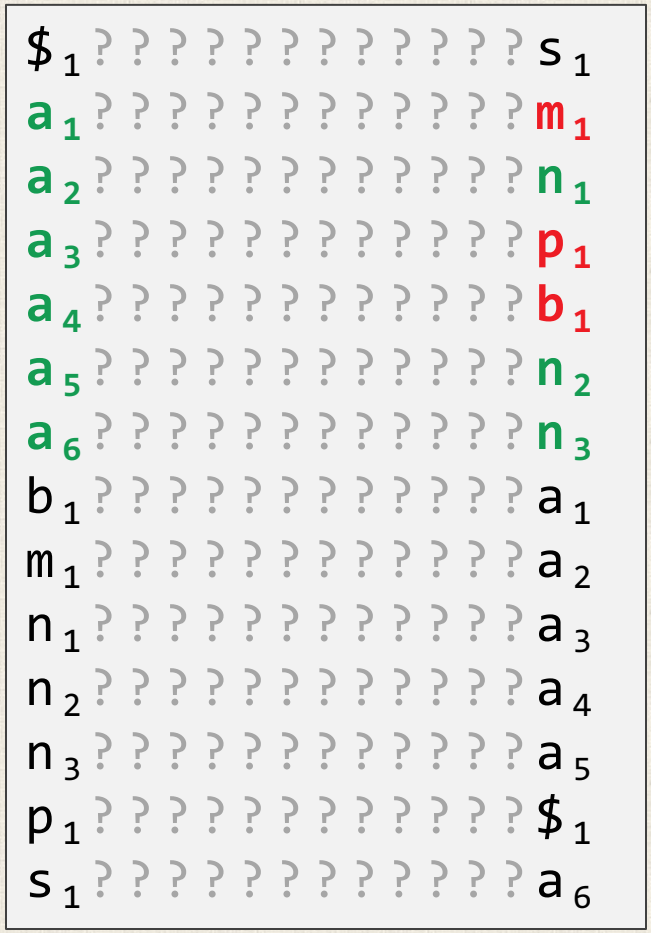
Searching for ana in *Text* = panamabananas

We Match Patterns *Backward*

$1	?	?	?	?	?	?	?	?	?	?	?	s1
a1	?	?	?	?	?	?	?	?	?	?	?	m1
a2	?	?	?	?	?	?	?	?	?	?	?	n1
a3	?	?	?	?	?	?	?	?	?	?	?	p1
a4	?	?	?	?	?	?	?	?	?	?	?	b1
a5	?	?	?	?	?	?	?	?	?	?	?	n2
a6	?	?	?	?	?	?	?	?	?	?	?	n3
b1	?	?	?	?	?	?	?	?	?	?	?	a1
m1	?	?	?	?	?	?	?	?	?	?	?	a2
n1	?	?	?	?	?	?	?	?	?	?	?	a3
n2	?	?	?	?	?	?	?	?	?	?	?	a4
n3	?	?	?	?	?	?	?	?	?	?	?	a5
p1	?	?	?	?	?	?	?	?	?	?	?	$1
s1	?	?	?	?	?	?	?	?	?	?	?	a6

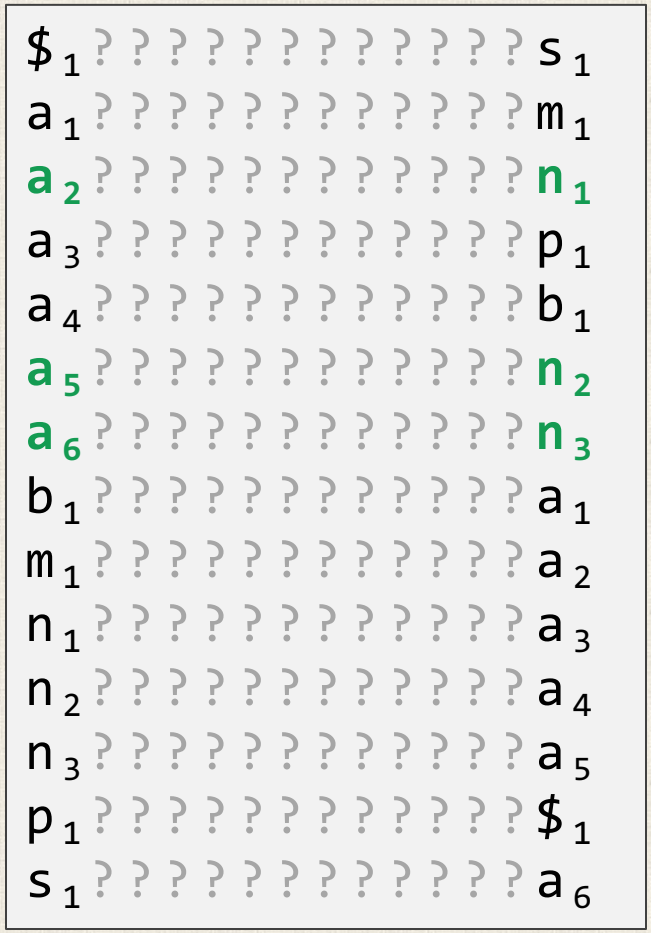
Searching for ana in *Text* = panamabananas

We Match Patterns *Backward*



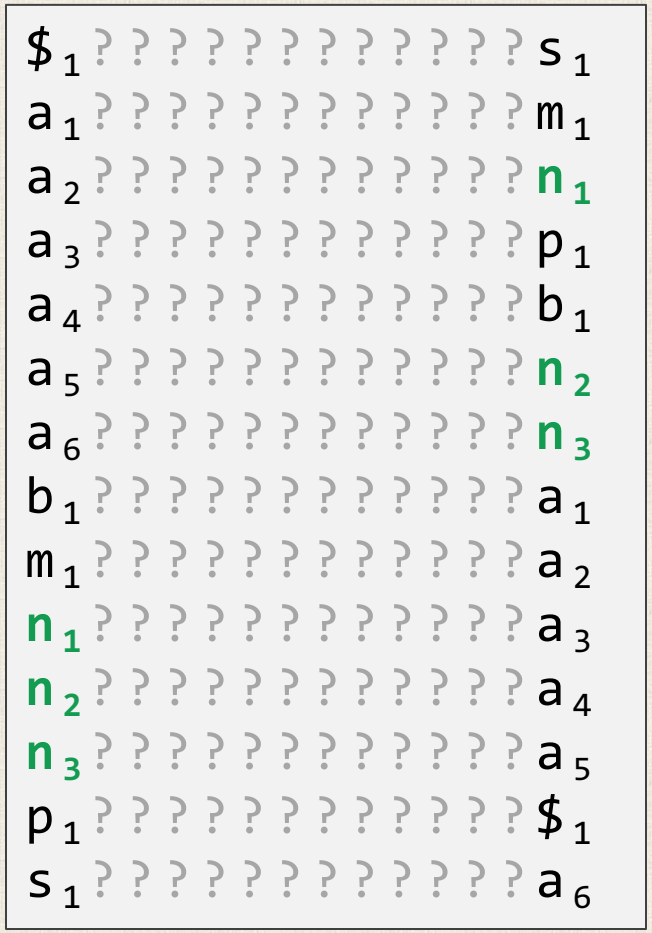
Searching for **ana** in *Text* = panamabananas

We Match Patterns *Backward*



Searching for **ana** in *Text* = panamabananas

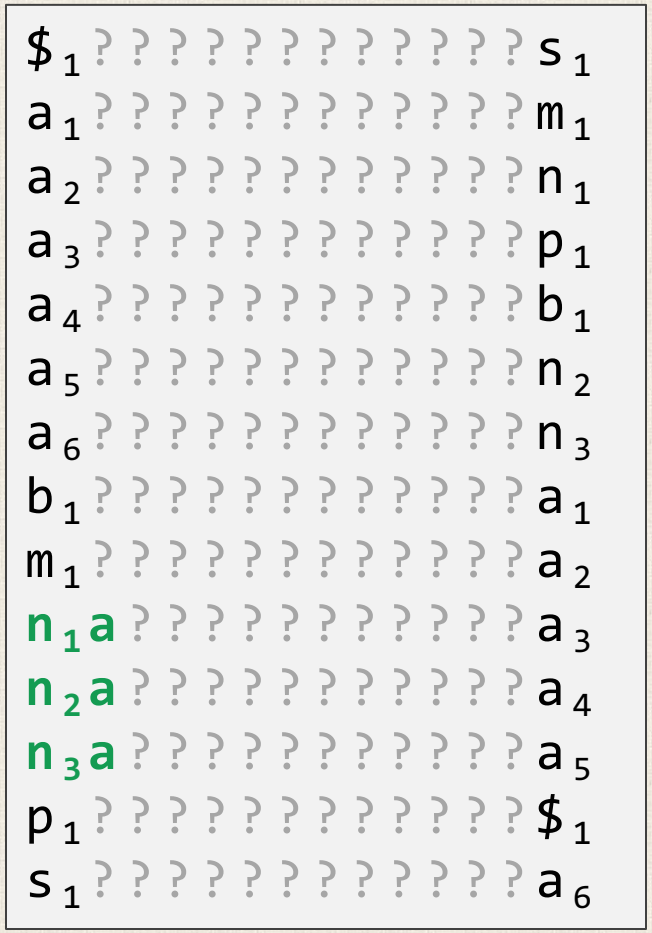
We Match Patterns *Backward*



Searching for **ana** in *Text* = panamabananas

Now we can apply the First-Last Property and find where these three "n" are hiding in *FirstColumn*.

We Match Patterns *Backward*

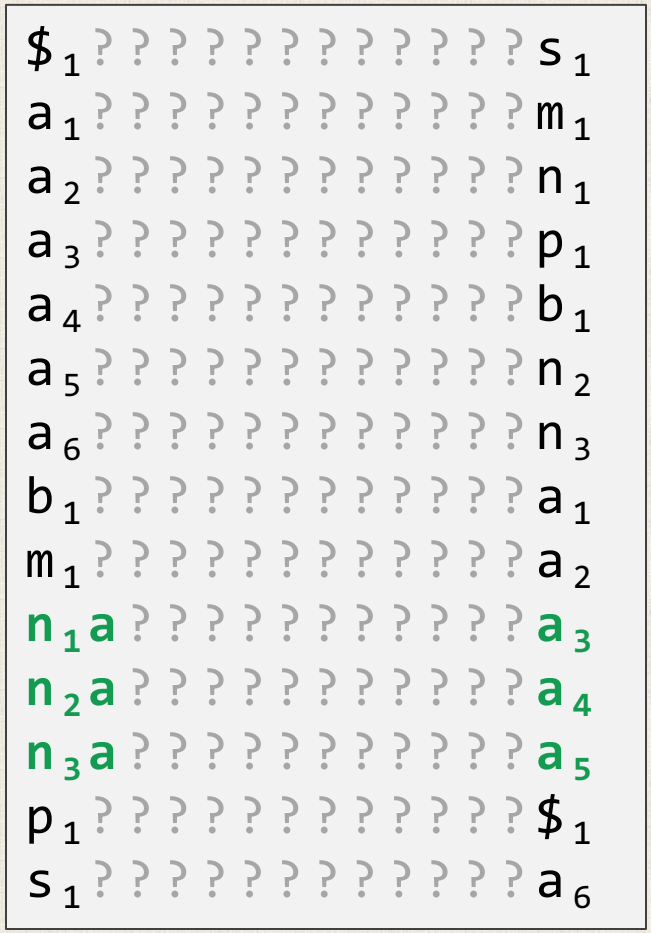


Searching for **ana** in *Text* = panamabananas

Now we can apply the First-Last Property and find where these three "n" are hiding in *FirstColumn*.

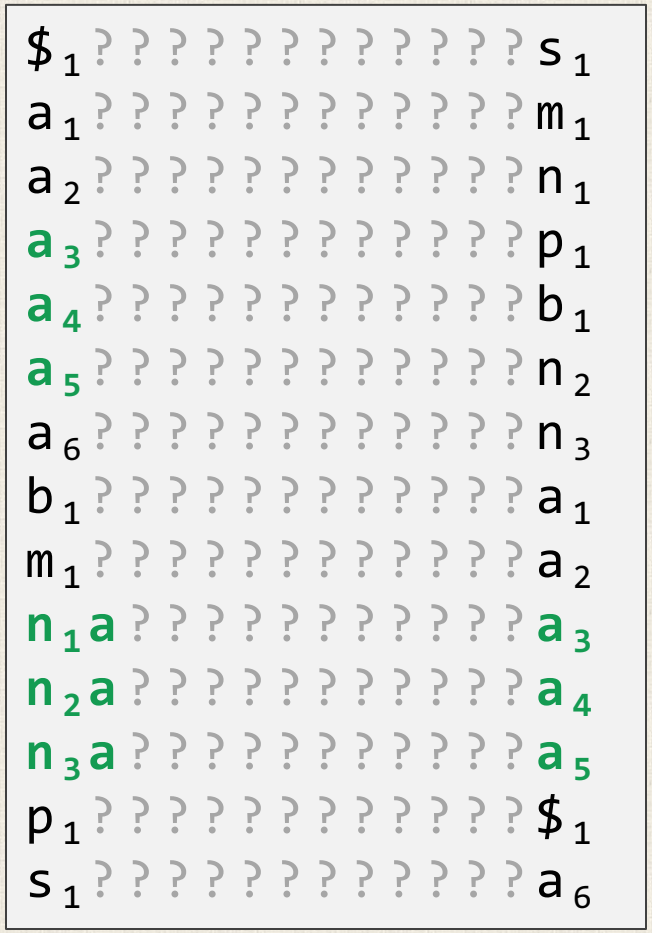
We can infer that these three rows start with "na".

We Match Patterns *Backward*



Searching for **ana** in *Text* = panamabananas

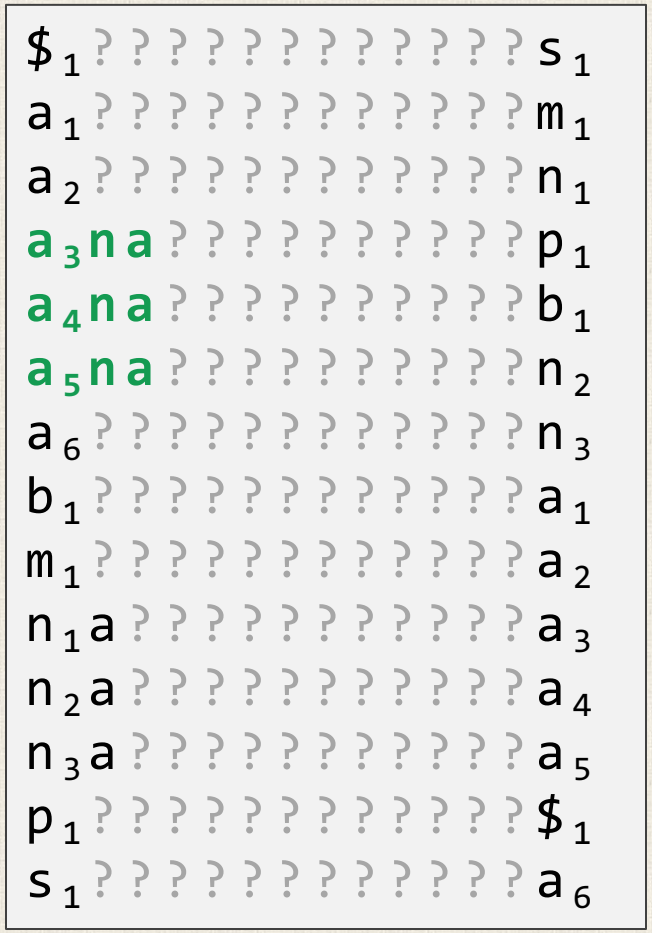
We Match Patterns *Backward*



Searching for **ana** in *Text* = panamabananas

All three match, and again we apply the First-Last Property.

We Match Patterns *Backward*

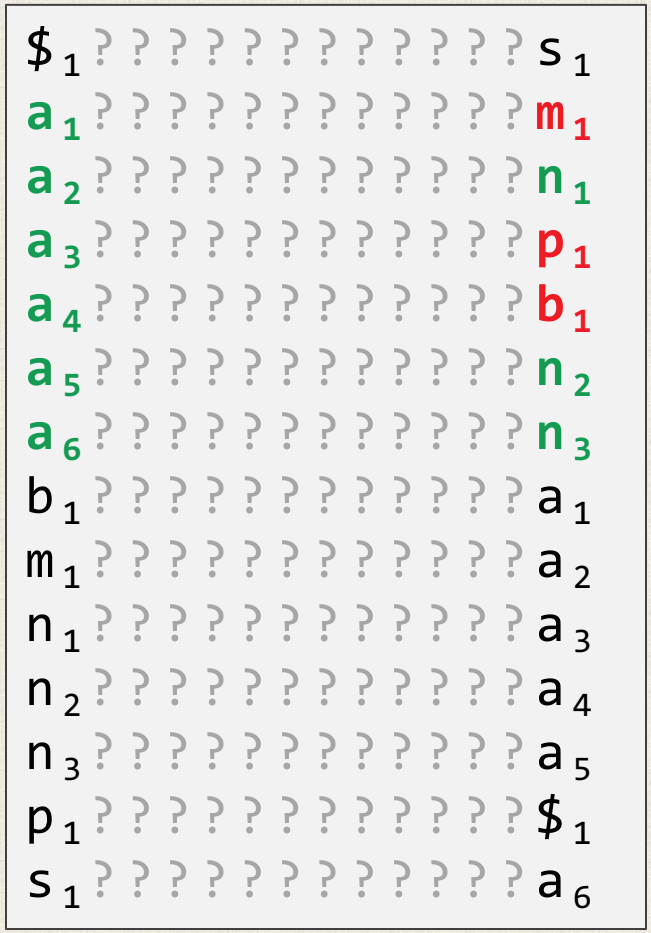


Searching for **ana** in *Text* = panamabananas

All three match, and again we apply the First-Last Property.

We have found the occurrences of "ana"!

Additional Information Needed



Problem: For a DNA alphabet, it will take $(1/4)|Text|$ runtime just to cycle through all these!

Matching "ana" backward with count arrays

$	?	?	?	?	?	?	?	?	?	?	?	s
a	?	?	?	?	?	?	?	?	?	?	?	m
a	?	?	?	?	?	?	?	?	?	?	?	n
a	?	?	?	?	?	?	?	?	?	?	?	p
a	?	?	?	?	?	?	?	?	?	?	?	b
a	?	?	?	?	?	?	?	?	?	?	?	n
a	?	?	?	?	?	?	?	?	?	?	?	n
b	?	?	?	?	?	?	?	?	?	?	?	a
m	?	?	?	?	?	?	?	?	?	?	?	a
n	?	?	?	?	?	?	?	?	?	?	?	a
n	?	?	?	?	?	?	?	?	?	?	?	a
n	?	?	?	?	?	?	?	?	?	?	?	a
p	?	?	?	?	?	?	?	?	?	?	?	$
s	?	?	?	?	?	?	?	?	?	?	?	a

$	a	b	m	n	p	s
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	0	1	1	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	0	1	1	2	1	1
0	0	1	1	3	1	1
0	1	1	1	3	1	1
0	2	1	1	3	1	1
0	3	1	1	3	1	1
0	4	1	1	3	1	1
0	5	1	1	3	1	1
1	5	1	1	3	1	1
1	6	1	1	3	1	1

"Count" arrays allow us to know what we've encountered at every point, so the subscripts become unnecessary.

Matching "ana" backward with count arrays

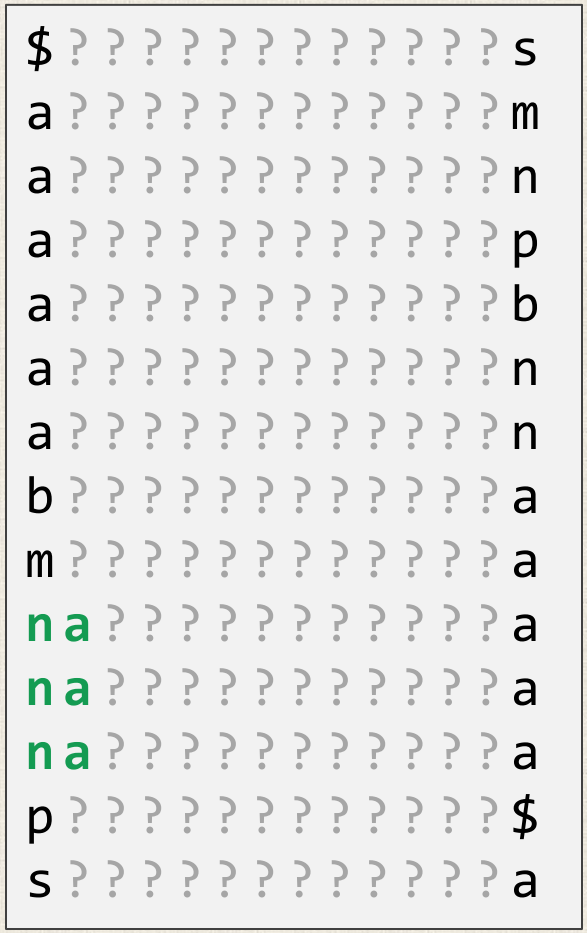
$	?	?	?	?	?	?	?	?	?	?	?	s
a	?	?	?	?	?	?	?	?	?	?	?	m
a	?	?	?	?	?	?	?	?	?	?	?	n
a	?	?	?	?	?	?	?	?	?	?	?	p
a	?	?	?	?	?	?	?	?	?	?	?	b
a	?	?	?	?	?	?	?	?	?	?	?	n
a	?	?	?	?	?	?	?	?	?	?	?	n
b	?	?	?	?	?	?	?	?	?	?	?	a
m	?	?	?	?	?	?	?	?	?	?	?	a
n	?	?	?	?	?	?	?	?	?	?	?	a
n	?	?	?	?	?	?	?	?	?	?	?	a
n	?	?	?	?	?	?	?	?	?	?	?	a
p	?	?	?	?	?	?	?	?	?	?	?	$
s	?	?	?	?	?	?	?	?	?	?	?	a

$	a	b	m	n	p	s
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	0	1	1	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	0	1	1	2	1	1
0	0	1	1	3	1	1
0	1	1	1	3	1	1
0	2	1	1	3	1	1
0	3	1	1	3	1	1
0	4	1	1	3	1	1
0	5	1	1	3	1	1
1	5	1	1	3	1	1
1	6	1	1	3	1	1

Here: we know that n_1 through n_3 are here by checking the two green values.

Of course, these arrays are a memory waste…

Matching "ana" backward with count arrays



$	a	b	m	n	p	s
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	0	1	1	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	0	1	1	2	1	1
0	0	1	1	3	1	1
0	1	1	1	3	1	1
0	**2**	1	1	3	1	1
0	3	1	1	3	1	1
0	4	1	1	3	1	1
0	**5**	1	1	3	1	1
1	5	1	1	3	1	1
1	6	1	1	3	1	1

We repeat this process and see that we must be looking for $\mathtt{a}_3$ through $\mathtt{a}_5$.

Matching "ana" backward with count arrays

$	a	b	m	n	p	s
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	0	1	1	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	0	1	1	2	1	1
0	0	1	1	3	1	1
0	1	1	1	3	1	1
0	2	1	1	3	1	1
0	3	1	1	3	1	1
0	4	1	1	3	1	1
0	5	1	1	3	1	1
1	5	1	1	3	1	1
1	6	1	1	3	1	1

Once we find a_3 through a_5, we have have now found that there are three occurrences of "ana".

But where are they in *text*?

WHERE ARE THE MATCHED PATTERNS?

Where are the Matches?

Multiple Pattern Matching Problem: *Find all occurrences of a collection of patterns in a text.*

- **Input:** A string *Text* and a collection *Patterns* containing (shorter) strings.
- **Output:** All ***starting positions*** in *Text* where a string from *Patterns* appears as a substring.

Where are the Matches?

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```

Example: We know that **ana** occurs 3 times, but where?

The Suffix Array Holds the Key

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```

The Suffix Array Holds the Key

	Suffix Array
$panamabananas	13
abananas$panam	5
amabananas$pan	3
anamabananas$p	1
ananas$panamab	7
anas$panamaban	9
as$panamabanan	11
bananas$panama	6
mabananas$pana	4
namabananas$pa	2
nanas$panamaba	8
nas$panamabana	10
panamabananas$	0
s$panamabanana	12

The Suffix Array Holds the Key

$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana

Suffix Array
13
5
3
1
7
9
11
6
4
2
8
10
0
12

"But we've already used the suffix array for pattern matching! How is this useful?"

How Can BWT Possibly Be Useful if we need all this information?

$	? ? ? ? ? ? ? ? ? ? ? s
a	? ? ? ? ? ? ? ? ? ? ? m
a	? ? ? ? ? ? ? ? ? ? ? n
a	? ? ? ? ? ? ? ? ? ? ? p
a	? ? ? ? ? ? ? ? ? ? ? b
a	? ? ? ? ? ? ? ? ? ? ? n
a	? ? ? ? ? ? ? ? ? ? ? n
b	? ? ? ? ? ? ? ? ? ? ? a
m	? ? ? ? ? ? ? ? ? ? ? a
n	? ? ? ? ? ? ? ? ? ? ? a
n	? ? ? ? ? ? ? ? ? ? ? a
n	? ? ? ? ? ? ? ? ? ? ? a
p	? ? ? ? ? ? ? ? ? ? ? $
s	? ? ? ? ? ? ? ? ? ? ? a

Suffix Array
13
5
3
1
7
9
11
6
4
2
8
10
0
12

$	a	b	m	n	p	s
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	0	1	1	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	0	1	1	2	1	1
0	0	1	1	3	1	1
0	1	1	1	3	1	1
0	2	1	1	3	1	1
0	3	1	1	3	1	1
0	4	1	1	3	1	1
0	5	1	1	3	1	1
1	5	1	1	3	1	1
1	6	1	1	3	1	1

BURROWS AND WHEELER SET UP CHECKPOINTS

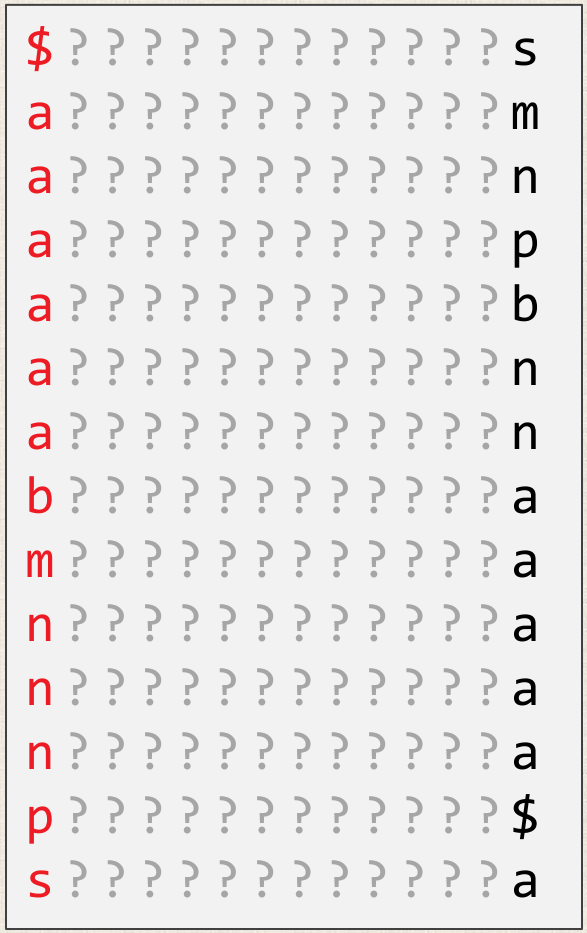
Key Point: We Don't Need to Store All This Data

$	? ? ? ? ? ? ? ? ? ? ? s
a	? ? ? ? ? ? ? ? ? ? ? m
a	? ? ? ? ? ? ? ? ? ? ? n
a	? ? ? ? ? ? ? ? ? ? ? p
a	? ? ? ? ? ? ? ? ? ? ? b
a	? ? ? ? ? ? ? ? ? ? ? n
a	? ? ? ? ? ? ? ? ? ? ? n
b	? ? ? ? ? ? ? ? ? ? ? a
m	? ? ? ? ? ? ? ? ? ? ? a
n	? ? ? ? ? ? ? ? ? ? ? a
n	? ? ? ? ? ? ? ? ? ? ? a
n	? ? ? ? ? ? ? ? ? ? ? a
p	? ? ? ? ? ? ? ? ? ? ? $
s	? ? ? ? ? ? ? ? ? ? ? a

Suffix Array
13
5
3
1
7
9
11
6
4
2
8
10
0
12

$	a	b	m	n	p	s
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	0	1	1	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	0	1	1	2	1	1
0	0	1	1	3	1	1
0	1	1	1	3	1	1
0	2	1	1	3	1	1
0	3	1	1	3	1	1
0	4	1	1	3	1	1
0	5	1	1	3	1	1
1	5	1	1	3	1	1
1	6	1	1	3	1	1

First: Dealing with the First Column



STOP: Is there any way of representing the information in the first column using less data?

First: Dealing with the First Column

	$? ? ? ? ? ? ? ? ? ? s
1	a ? ? ? ? ? ? ? ? ? ? m
	a ? ? ? ? ? ? ? ? ? ? n
	a ? ? ? ? ? ? ? ? ? ? p
	a ? ? ? ? ? ? ? ? ? ? b
	a ? ? ? ? ? ? ? ? ? ? n
	a ? ? ? ? ? ? ? ? ? ? n
7	b ? ? ? ? ? ? ? ? ? ? a
8	m ? ? ? ? ? ? ? ? ? ? a
9	n ? ? ? ? ? ? ? ? ? ? a
	n ? ? ? ? ? ? ? ? ? ? a
	n ? ? ? ? ? ? ? ? ? ? a
12	p ? ? ? ? ? ? ? ? ? ? $
13	s ? ? ? ? ? ? ? ? ? ? a

STOP: Is there any way of representing the information in the first column using less data?

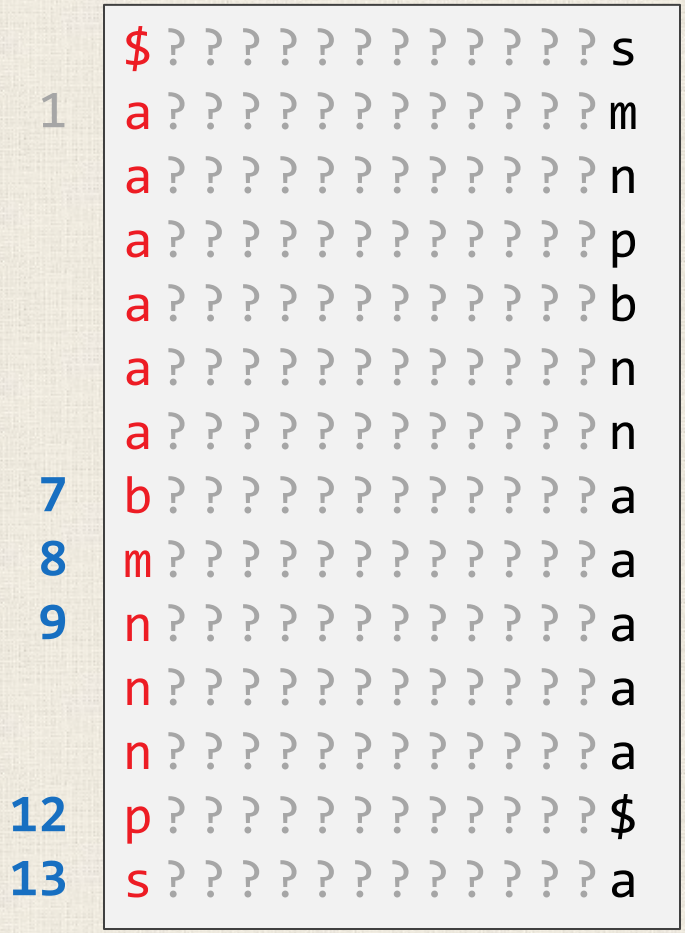
Answer: We only need to store the *first occurrence* of each symbol.

First: Dealing with the First Column

	$	? ? ? ? ? ? ? ? ? ? s
1	a	? ? ? ? ? ? ? ? ? ? m
	a	? ? ? ? ? ? ? ? ? ? n
	a	? ? ? ? ? ? ? ? ? ? p
	a	? ? ? ? ? ? ? ? ? ? b
	a	? ? ? ? ? ? ? ? ? ? n
	a	? ? ? ? ? ? ? ? ? ? n
7	b	? ? ? ? ? ? ? ? ? ? a
8	m	? ? ? ? ? ? ? ? ? ? a
9	n	? ? ? ? ? ? ? ? ? ? a
	n	? ? ? ? ? ? ? ? ? ? a
	n	? ? ? ? ? ? ? ? ? ? a
12	p	? ? ? ? ? ? ? ? ? ? $
13	s	? ? ? ? ? ? ? ? ? ? a

Note: we don't even need to store that the first symbol occurs at position 1 of the first column.

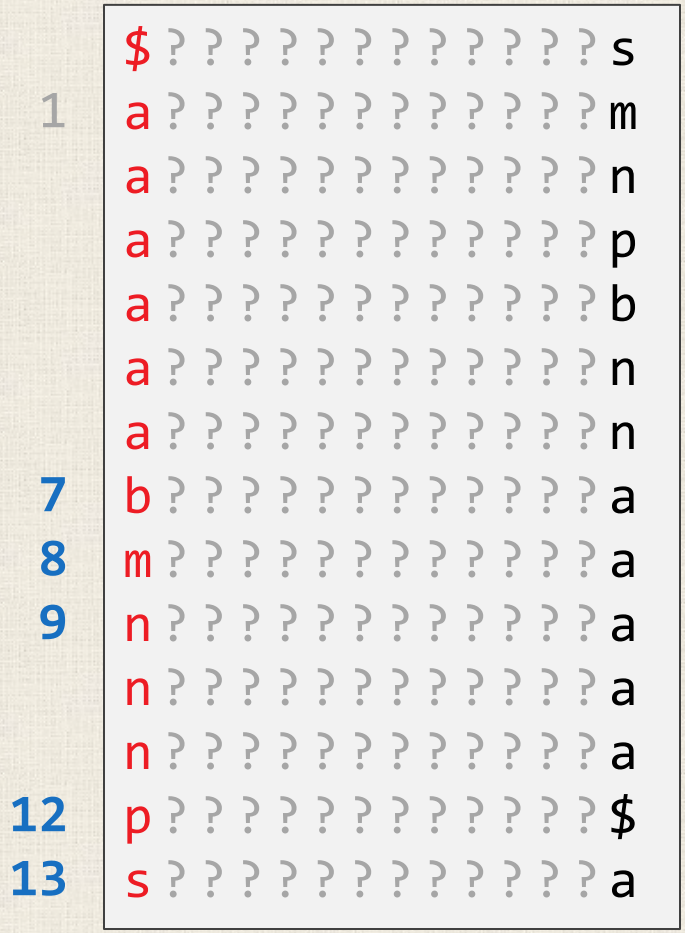
First: Dealing with the First Column



Note: we don't even need to store that the first symbol occurs at position 1 of the first column.

STOP: For a genome, how many integers will we need to store to hold the first occurrence of each symbol?

First: Dealing with the First Column



Note: we don't even need to store that the first symbol occurs at position 1 of the first column.

Answer: Only three! The position of the first occurrence of `C`, `G`, and `T`. (`$` occurs at position 0; `A` occurs at position 1.)

Second: A "Partial" Suffix Array

Partial suffix array: only stores values that are divisible by K for some integer K.

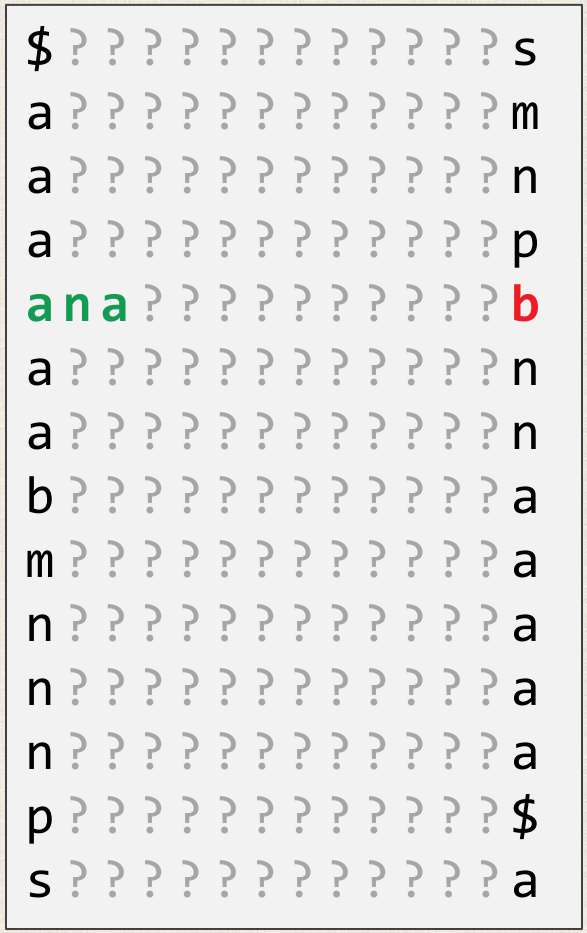
$K = 5$ at left.

Second: A "Partial" Suffix Array

Suffix Array
13
5
3
1
7
9
11
6
4
2
8
10
0
12

We can make at most $K - 1$ additional backtrack steps to determine where our pattern match is.

Second: A "Partial" Suffix Array



Suffix Array
13
5
3
1
7
9
11
6
4
2
8
10
0
12

We can make at most $K - 1$ additional backtrack steps to determine where our pattern match is.

Second: A "Partial" Suffix Array

Suffix Array
13
5
3
1
7
9
11
6
4
2
8
10
0
12

We can make at most $K - 1$ additional backtrack steps to determine where our pattern match is.

Second: A "Partial" Suffix Array

Suffix Array
13
5
3
1
7
9
11
6
4
2
8
10
0
12

We can make at most $K - 1$ additional backtrack steps to determine where our pattern match is.

Second: A "Partial" Suffix Array

Suffix Array
13
5
3
1
7
9
11
6
4
2
8
10
0
12

We can make at most $K - 1$ additional backtrack steps to determine where our pattern match is.

Second: A "Partial" Suffix Array

Suffix Array
13
5
3
1
7
9
11
6
4
2
8
10
0
12

We have a match! We know "abana" occurs at position 5, and we took two steps back, so this "ana" occurs at position 7.

Third: "Checkpoint" Count Arrays

$	?	?	?	?	?	?	?	?	?	?	?	s
a	?	?	?	?	?	?	?	?	?	?	?	m
a	?	?	?	?	?	?	?	?	?	?	?	n
a	?	?	?	?	?	?	?	?	?	?	?	p
a	?	?	?	?	?	?	?	?	?	?	?	b
a	?	?	?	?	?	?	?	?	?	?	?	n
a	?	?	?	?	?	?	?	?	?	?	?	n
b	?	?	?	?	?	?	?	?	?	?	?	a
m	?	?	?	?	?	?	?	?	?	?	?	a
n	?	?	?	?	?	?	?	?	?	?	?	a
n	?	?	?	?	?	?	?	?	?	?	?	a
n	?	?	?	?	?	?	?	?	?	?	?	a
p	?	?	?	?	?	?	?	?	?	?	?	$
s	?	?	?	?	?	?	?	?	?	?	?	a

$	a	b	m	n	p	s
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	0	1	1	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	0	1	1	2	1	1
0	0	1	1	3	1	1
0	1	1	1	3	1	1
0	2	1	1	3	1	1
0	3	1	1	3	1	1
0	4	1	1	3	1	1
0	5	1	1	3	1	1
1	5	1	1	3	1	1
1	6	1	1	3	1	1

"Count" arrays help us know what we've encountered up to given row.

Third: "Checkpoint" Count Arrays

$	?	?	?	?	?	?	?	?	?	?	?	s
a	?	?	?	?	?	?	?	?	?	?	?	m
a	?	?	?	?	?	?	?	?	?	?	?	n
a	?	?	?	?	?	?	?	?	?	?	?	p
a	?	?	?	?	?	?	?	?	?	?	?	b
a	?	?	?	?	?	?	?	?	?	?	?	n
a	?	?	?	?	?	?	?	?	?	?	?	n
b	?	?	?	?	?	?	?	?	?	?	?	a
m	?	?	?	?	?	?	?	?	?	?	?	a
n	?	?	?	?	?	?	?	?	?	?	?	a
n	?	?	?	?	?	?	?	?	?	?	?	a
n	?	?	?	?	?	?	?	?	?	?	?	a
p	?	?	?	?	?	?	?	?	?	?	?	$
s	?	?	?	?	?	?	?	?	?	?	?	a

$	a	b	m	n	p	s
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	0	1	1	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	0	1	1	2	1	1
0	0	1	1	3	1	1
0	1	1	1	3	1	1
0	2	1	1	3	1	1
0	3	1	1	3	1	1
0	4	1	1	3	1	1
0	5	1	1	3	1	1
1	5	1	1	3	1	1
1	6	1	1	3	1	1

"Count" arrays help us know what we've encountered up to given row.

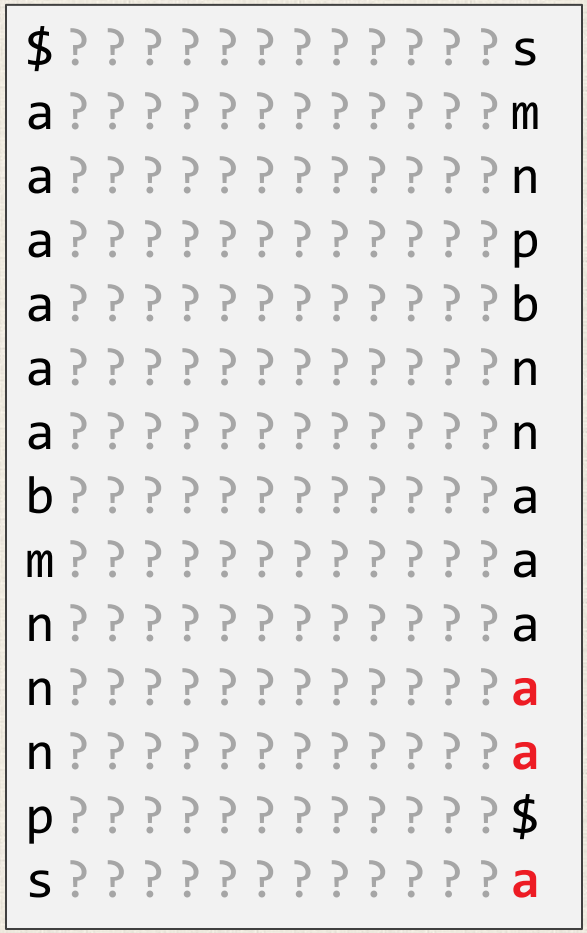
Checkpoint arrays: only store every C arrays.

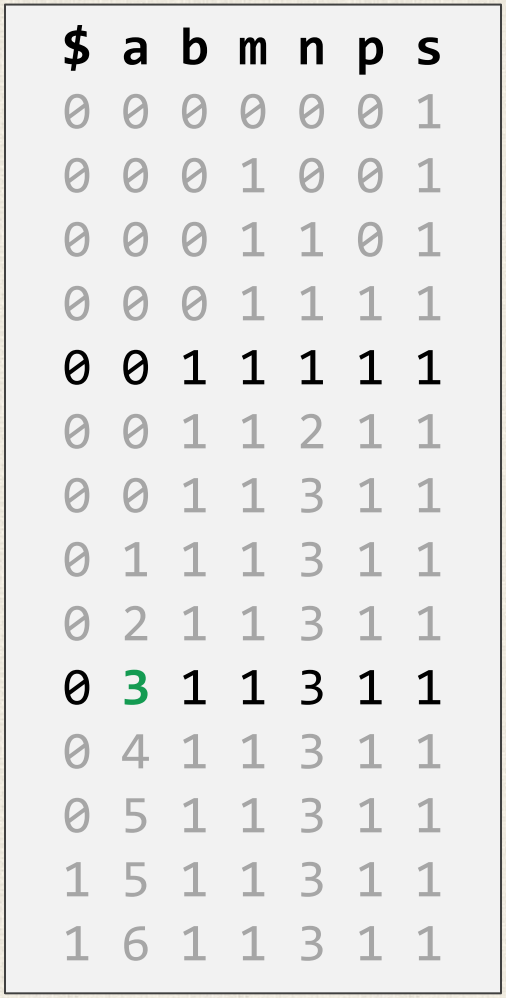
Third: "Checkpoint" Count Arrays

$	a	b	m	n	p	s
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	0	1	1	0	1
0	0	0	1	1	1	1
0	**0**	**1**	**1**	**1**	**1**	**1**
0	0	1	1	2	1	1
0	0	1	1	3	1	1
0	1	1	1	3	1	1
0	2	1	1	3	1	1
0	**3**	**1**	**1**	**3**	**1**	**1**
0	4	1	1	3	1	1
0	5	1	1	3	1	1
1	5	1	1	3	1	1
1	6	1	1	3	1	1

STOP: Say we are at position 13 and see "a". How can we know how many "a"s have occurred so far?

Third: "Checkpoint" Count Arrays





Answer: Go to nearest checkpoint array, which has counted 3 "a"s up to position 10, and then count three more (total of 6).

"How are *K* and *C* chosen in practice?"

$? ? ? ? ? ? ? ? ? ? s
a ? ? ? ? ? ? ? ? ? ? m
a ? ? ? ? ? ? ? ? ? ? n
a ? ? ? ? ? ? ? ? ? ? p
a ? ? ? ? ? ? ? ? ? ? b
a ? ? ? ? ? ? ? ? ? ? n
a ? ? ? ? ? ? ? ? ? ? n
b ? ? ? ? ? ? ? ? ? ? a
m ? ? ? ? ? ? ? ? ? ? a
n ? ? ? ? ? ? ? ? ? ? a
n ? ? ? ? ? ? ? ? ? ? a
n ? ? ? ? ? ? ? ? ? ? a
p ? ? ? ? ? ? ? ? ? ? $
s ? ? ? ? ? ? ? ? ? ? a

$	a	b	m	n	p	s
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	0	1	1	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	0	1	1	2	1	1
0	0	1	1	3	1	1
0	1	1	1	3	1	1
0	2	1	1	3	1	1
0	3	1	1	3	1	1
0	4	1	1	3	1	1
0	5	1	1	3	1	1
1	5	1	1	3	1	1
1	6	1	1	3	1	1

(Partial) Suffix Array

13
5
3
1
7
9
11
6
4
2
8
10
0
12

So, We Need Just Four Items … Total Memory ($K = C = 100$): ~1.5|*Text*|!

First Occurrence	BWT
	s
1	m
	n
	p
	b
	n
	n
7	a
8	a
9	a
	a
	a
12	$
13	a

Checkpoint Arrays

$	a	b	m	n	p	s
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	0	1	1	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	0	1	1	2	1	1
0	0	1	1	3	1	1
0	1	1	1	3	1	1
0	2	1	1	3	1	1
0	3	1	1	3	1	1
0	4	1	1	3	1	1
0	5	1	1	3	1	1
1	5	1	1	3	1	1
1	6	1	1	3	1	1

(Partial) Suffix Array

13
5
3
1
7
9
11
6
4
2
8
10
0
12

EPILOGUE: MISMATCH-TOLERANT READ MAPPING

Returning to Our Original Problem

Multiple Pattern Matching Problem: *Find all occurrences of a collection of patterns in a text.*

- **Input:** A string *Text* and a collection *Patterns* containing (shorter) strings.
- **Output:** All starting positions in *Text* where a string from *Patterns* appears as a substring.

Returning to Our Original Problem

Multiple Approximate Pattern Matching Problem:
Find all occurrences of a collection of patterns in a text.

- **Input:** A string *Text,* a collection *Patterns* containing (shorter) strings, **and an integer *d***.
- **Output:** All starting positions in *Text* where a string from *Patterns* appears as a substring **with at most *d* mismatches**.

Method 1: Seeding

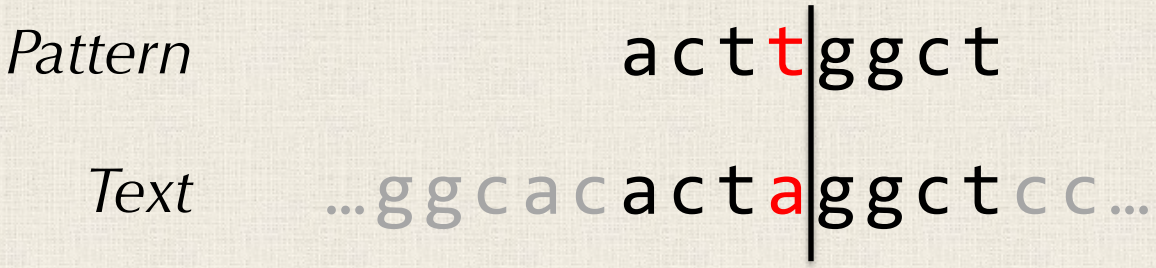
Say that *Pattern* appears in *Text* with just one mismatch.

Pattern actt**t**ggct

Text …ggcacact**a**ggctcc…

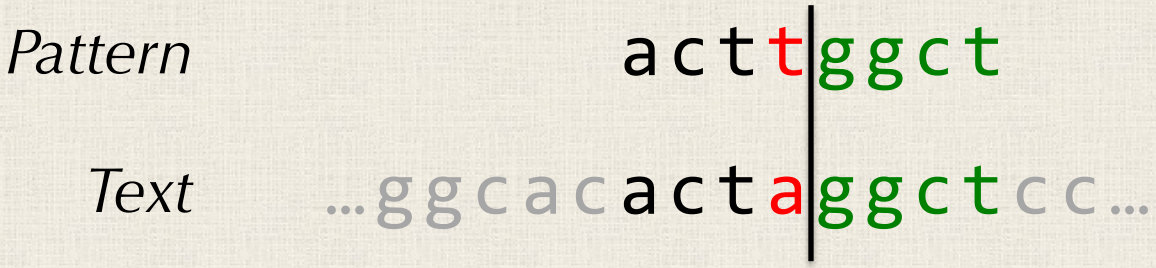
Method 1: Seeding

Say that *Pattern* appears in *Text* with just one mismatch.



Method 1: Seeding

Say that *Pattern* appears in *Text* with just one mismatch.



If we divide the strings in half, then one must match exactly!

Method 1: Seeding

Let's take another example of strings that have 3 mismatches.

Pattern acttaggctcgggataatcc

Text …ggcacactaagtctcgggataagcccc…

Method 1: Seeding

Let's take another example of strings that have 3 mismatches.

Now we can divide strings into four equal pieces and find at least one that matches.

Method 1: Seeding

Theorem: If *Pattern* occurs in *Text* with *d* mismatches, then we can divide *Pattern* into *d* + 1 "equal" pieces and find at least one exact match.

https://en.wikipedia.org/wiki/Pigeonhole_principle#/media/File:TooManyPigeons.jpg

Method 1: Seeding

Theorem: If *Pattern* occurs in *Text* with *d* mismatches, then we can divide *Pattern* into *d* + 1 "equal" pieces and find at least one exact match.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

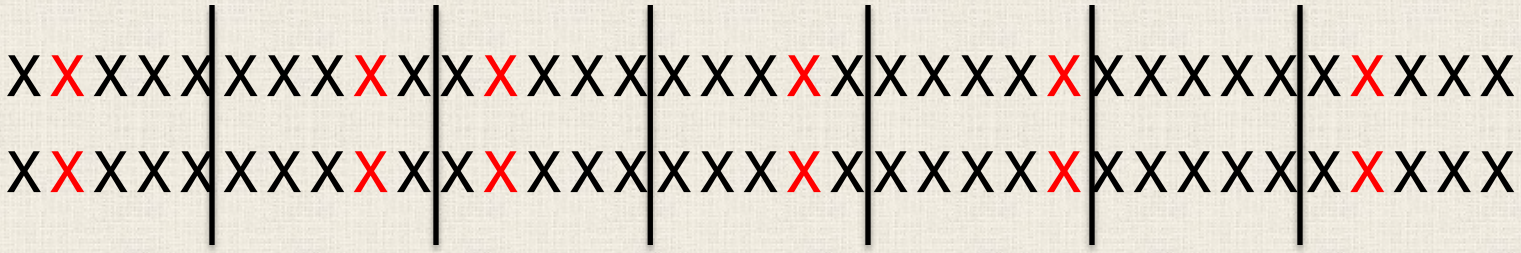
Method 1: Seeding

Theorem: If *Pattern* occurs in *Text* with *d* mismatches, then we can divide *Pattern* into *d* + 1 "equal" pieces and find at least one exact match.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

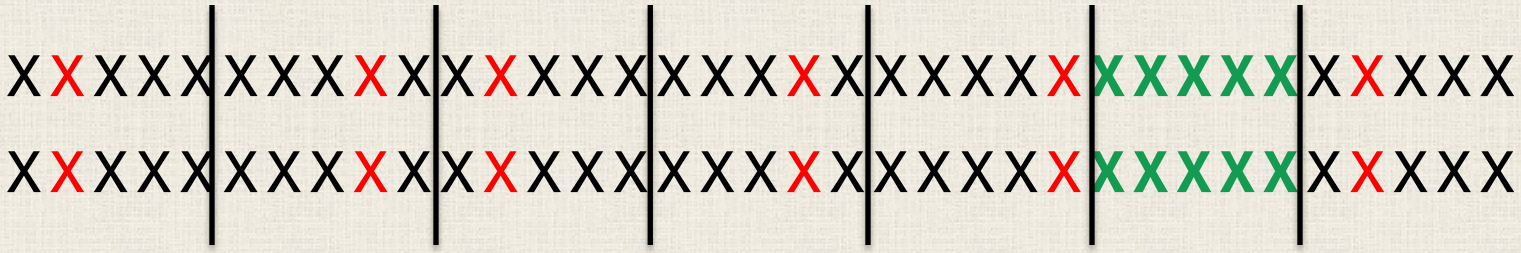
Method 1: Seeding

Theorem: If *Pattern* occurs in *Text* with *d* mismatches, then we can divide *Pattern* into *d* + 1 "equal" pieces and find at least one exact match.



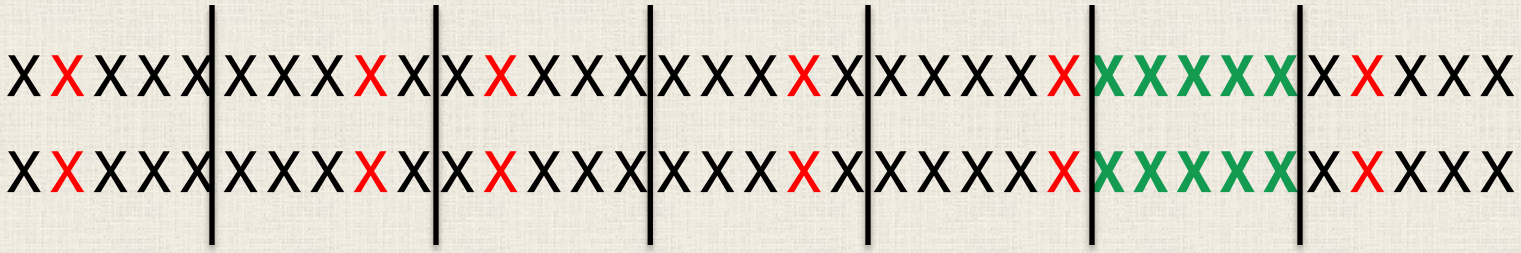
Method 1: Seeding

Theorem: If *Pattern* occurs in *Text* with *d* mismatches, then we can divide *Pattern* into *d* + 1 "equal" pieces and find at least one exact match.



Method 1: Seeding

Theorem (Formally): If two strings of length n match with at most d mismatches, then they must share a k-mer of length $k = \lfloor n/(d+1) \rfloor$.



Method 1: Seeding

Seed-and-extend for inexact pattern matching:

1. Divide *Pattern* into d+1 "equal" segments (called **seeds**).
2. Find which seeds match *Text* exactly, using an approach like BWT (**seed detection**).
3. Extend all seeds in both directions and align *Pattern* against this region of *Text* (**seed extension**) – can be generalized to include gaps.

Method 2: BWT Saves the Day Again

Recall: searching for ana in panamabananas

$	1	p	a	n	a	m	a	b	a	n	a	n	a	s	1
a	1	b	a	n	a	n	a	s	$	p	a	n	a	m	1
a	2	m	a	b	a	n	a	n	a	s	$	p	a	n	1
a	3	n	a	m	a	b	a	n	a	n	a	s	$	p	1
a	4	n	a	n	a	s	$	p	a	n	a	m	a	b	1
a	5	n	a	s	$	p	a	n	a	m	a	b	a	n	2
a	6	s	$	p	a	n	a	m	a	b	a	n	a	n	3
b	1	a	n	a	n	a	s	$	p	a	n	a	m	a	1
m	1	a	b	a	n	a	n	a	s	$	p	a	n	a	2
n	1	a	m	a	b	a	n	a	n	a	s	$	p	a	3
n	2	a	n	a	s	$	p	a	n	a	m	a	b	a	4
n	3	a	s	$	p	a	n	a	m	a	b	a	n	a	5
p	1	a	n	a	m	a	b	a	n	a	n	a	s	$	1
s	1	$	p	a	n	a	m	a	b	a	n	a	n	a	6

Method 2: BWT Saves the Day Again

Recall: searching for **ana** in **panamabananas**

$ 1	panamabananas 1
a 1	bananas$panam 1
a 2	mabananas$pan 1
a 3	namabananas$p 1
a 4	nanas$panamab 1
a 5	nas$panamaban 2
a 6	s$panamabanan 3
b 1	ananas$panama 1
m 1	abananas$pana 2
n 1	amabananas$pa 3
n 2	anas$panamaba 4
n 3	as$panamabana 5
p 1	anamabananas$ 1
s 1	$panamabanana 6

Method 2: BWT Saves the Day Again

Recall: searching for **ana** in panamabananas

Method 2: BWT Saves the Day Again

Recall: searching for **ana** in panamabananas

If we allow 1 mismatch, then we need to keep the red letters around.

Method 2: BWT Saves the Day Again

Recall: searching for **ana** in panamabananas

If we allow 1 mismatch, then we need to keep the red letters around.

Mismatches
1
0
1
1
0
0

Method 2: BWT Saves the Day Again

Recall: searching for **ana** in panamabananas

Now we extend only strings that have at most 1 mismatch.

	# Mismatches
$$_1$panamabananas$_1$	1
a$_1$bananas$panam**m**$_1$	0
a$_2$mabananas$pan**n**$_1$	1
a$_3$namabananas$**p**$_1$	1
a$_4$nanas$panama**b**$_1$	0
a$_5$nas$panamaba**n**$_2$	0
a$_6$s$panamaban**n**$_3$	
b$_1$**a**nanas$panama$_1$	
m$_1$**a**bananas$pana$_2$	
n$_1$**a**mabananaspa_3$	
n$_2$**a**nas$panamaba$_4$	
n$_3$**a**s$panamaban$_5$	
p$_1$anamabananas$$_1$	
s$_1$$panamabanan**a**$_6$	

Method 2: BWT Saves the Day Again

Recall: searching for **ana** in panamabananas

Now we extend only strings that have at most 1 mismatch.

	# Mismatches
$$_1$panamabananas$_1$	
a$_1$bananas$panam$_1$	
a$_2$mabananaspan_1$	
a$_3$namabananas$p$_1$	
a$_4$nanas$panamab$_1$	
a$_5$nas$panamaban$_2$	
a$_6$s$panamabanan$_3$	
b$_1$ananas$panama$_1$	1
m$_1$abananas$pana$_2$	1
n$_1$amabananaspa_3$	0
n$_2$anas$panamaba$_4$	0
n$_3$as$panamaban$_5$	0
p$_1$anamabananas$$_1$	1
s$_1$$panamabanana$_6$	

Method 2: BWT Saves the Day Again

Recall: searching for **ana** in panamabananas

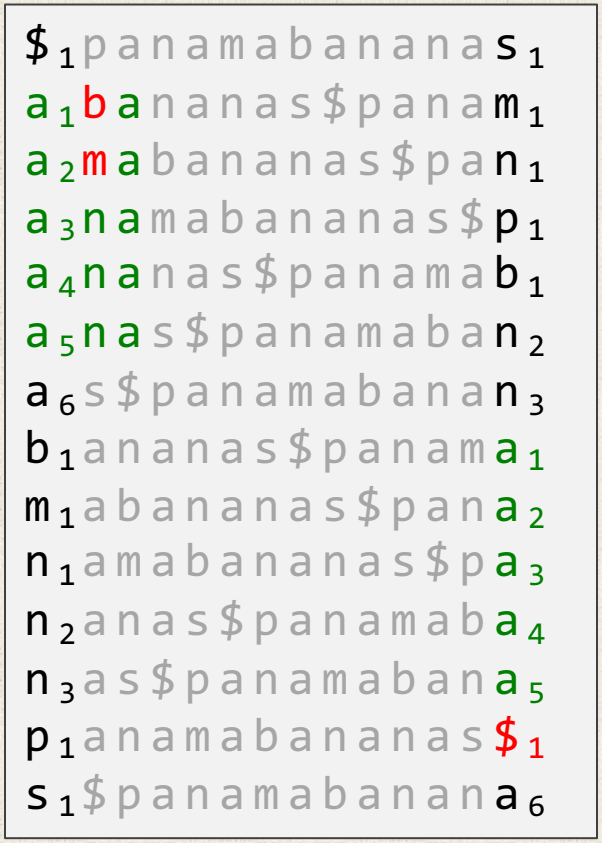
One string produces a second mismatch (the $), so we discard it.

	# Mismatches
$$_1$panamabananas$_1$	
a$_1$bananas$panam$_1$	
a$_2$mabananaspan_1$	
a$_3$namabananas$p$_1$	
a$_4$nanas$panamab$_1$	
a$_5$nas$panamaban$_2$	
a$_6$s$panamabanan$_3$	
b$_1$ananas$panama$_1$	1
m$_1$abananas$pana$_2$	1
n$_1$amabananaspa_3$	0
n$_2$anas$panamaba$_4$	0
n$_3$as$panamaban$_5$	0
p$_1$anamabananas$$_1$	2
s$_1$$panamabanana$_6$	

Method 2: BWT Saves the Day Again

Recall: searching for ana in panamabananas

In the end, we have five 3-mers with at most 1 mismatch.



 # Mismatches

1
1
0
0
0
2

Method 2: BWT Saves the Day Again

Note: this approach faces complications in practice. We will have to consider a huge number of strings at the beginning, most of which may be frivolous.

In practice, we may require that a suffix of length *m* matches the string exactly to reduce the number of strings we have to consider initially.

Citations

Compression

[PDF] A block-sorting lossless data compression algorithm
www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf ▾
by AB Lossless - 1994 - Related articles
May 10, 1994 - **A Block-sorting Lossless**. **Data Compression Algorithm**. M. Burrows and D.J. Wheeler digital. Systems Research Center. 130 Lytton Avenue.

Pattern Matching

Opportunistic data structures with applications - IEEE Conference ...
https://ieeexplore.ieee.org/document/892127
by P Ferragina - 2000 - Cited by 1016 - Related articles
Opportunistic data structures with applications. Abstract: We address the issue of compressing and **indexing** data. We devise a data structure whose space occupancy is a function of the entropy of the underlying data set.

Read Mapping

Fast and accurate short read alignment with Burrows–Wheeler transform
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2705234/ ▾
by H Li - 2009 - Cited by 17996 - Related articles
May 18, 2009 - Results: We implemented Burrows-Wheeler Alignment tool (**BWA**), a new read alignment package that is based on backward search with ...

Ultrafast and memory-efficient alignment of short DNA sequences to ...
https://genomebiology.biomedcentral.com/articles/10.1186/gb-2009-10-3-r25 ▾
by B Langmead - 2009 - Cited by 13797 - Related articles
Mar 4, 2009 - **Bowtie** is an ultrafast, memory-efficient alignment program for aligning short DNA sequence reads to large genomes. For the human genome, ...