University of
BRISTOL

# Advanced Algorithms – COMS31900

## Pattern Matching part one

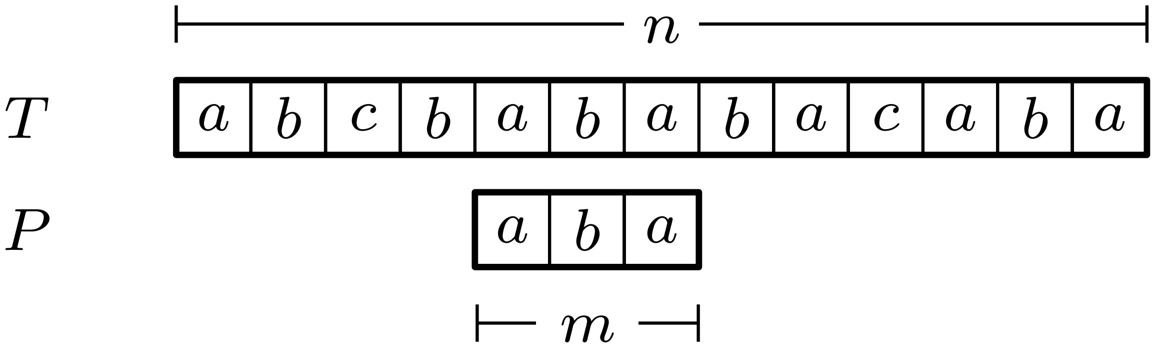
### Suffix Trees

Raphaël Clifford

Slides by Benjamin Sach

# Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)



**Goal:** Find all the locations where $P$ **matches** in $T$

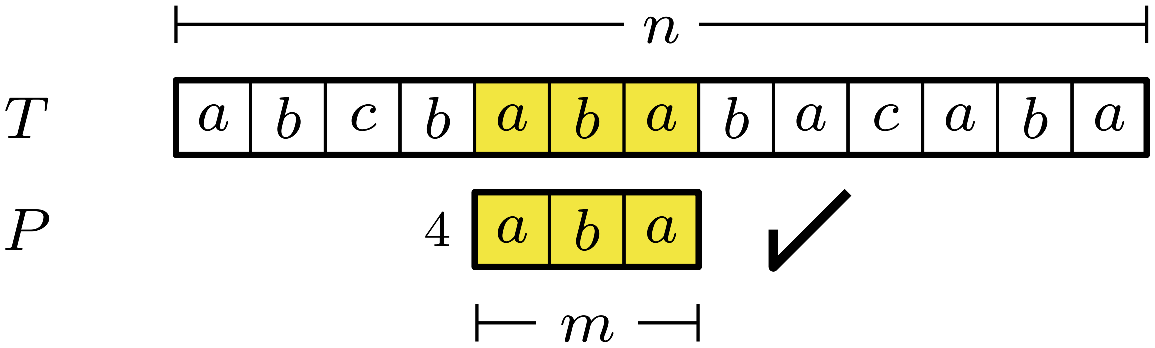
$P$ matches at location $i$ iff

for all $0 \leqslant j \leqslant m$ we have that $P[j] = T[i+j]$

*(our strings are zero-indexed)*

# Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)



**Goal:** Find all the locations where $P$ **matches** in $T$

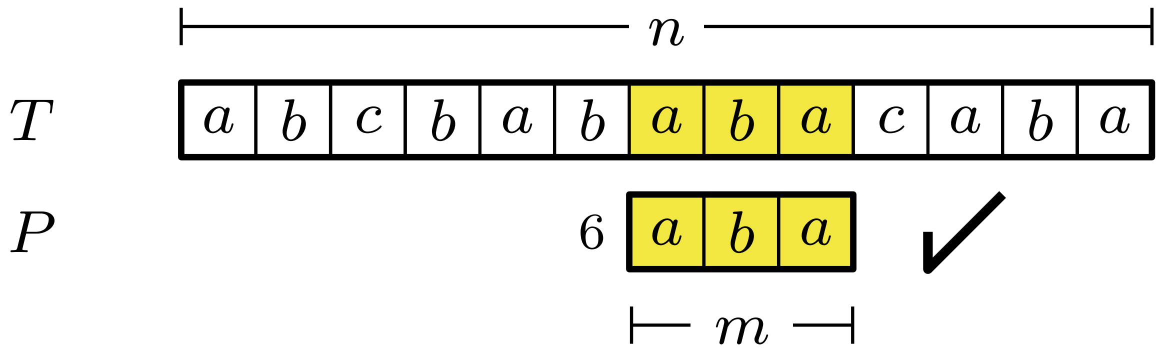
$P$ matches at location $i$ iff

for all $0 \leqslant j \leqslant m$ we have that $P[j] = T[i+j]$

*(our strings are zero-indexed)*

# Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)



**Goal:** Find all the locations where $P$ **matches** in $T$

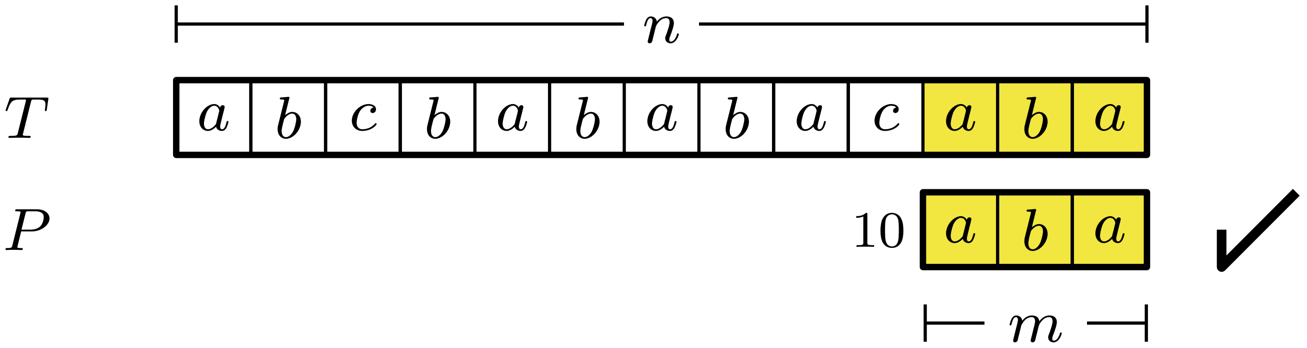
$P$ matches at location $i$ iff

for all $0 \leqslant j \leqslant m$ we have that $P[j] = T[i+j]$

*(our strings are zero-indexed)*

# Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)



**Goal:** Find all the locations where $P$ **matches** in $T$

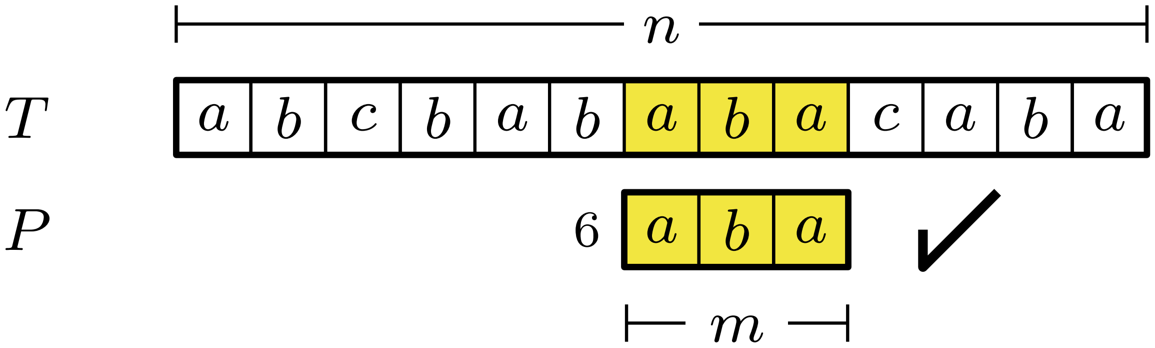
$P$ matches at location $i$ iff

for all $0 \leqslant j \leqslant m$ we have that $P[j] = T[i+j]$

*(our strings are zero-indexed)*

# Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)



**Goal:** Find all the locations where $P$ **matches** in $T$

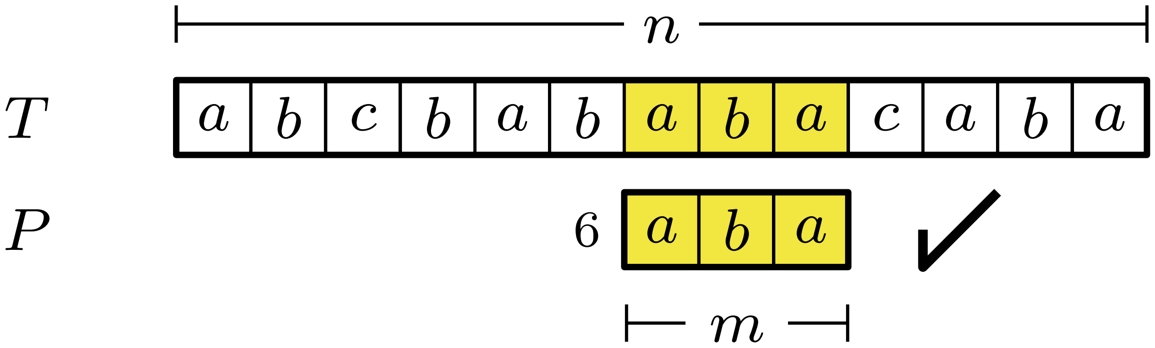
$P$ matches at location $i$ iff

for all $0 \leqslant j \leqslant m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*

# Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)



**Goal:** Find all the locations where $P$ **matches** in $T$

$P$ matches at location $i$ iff

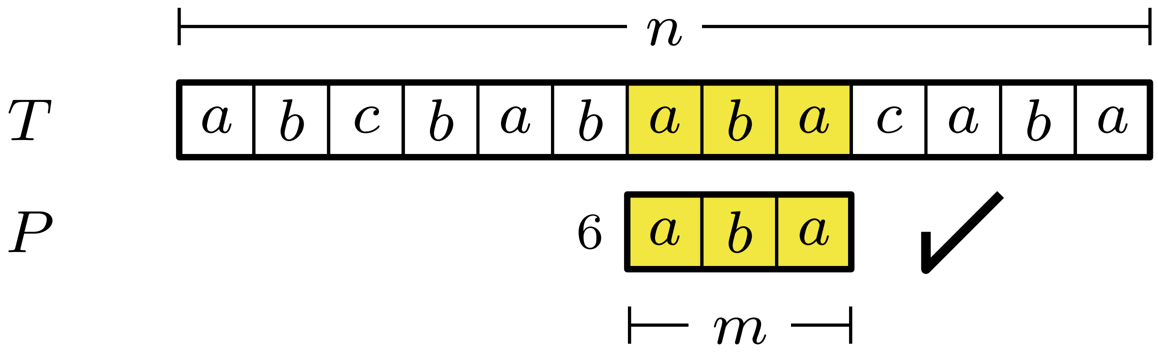
for all $0 \leqslant j \leqslant m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*

- A naive algorithm takes $O(nm)$ time

# Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)



**Goal:** Find all the locations where $P$ **matches** in $T$

$P$ matches at location $i$ iff

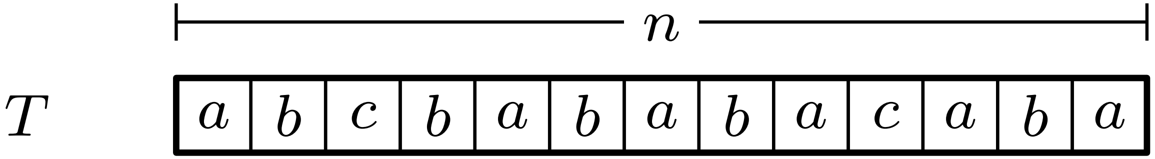
for all $0 \leqslant j \leqslant m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*

- A naive algorithm takes $O(nm)$ time
- Many $O(n)$ time algorithms are known (for example KMP)

# Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries. . .

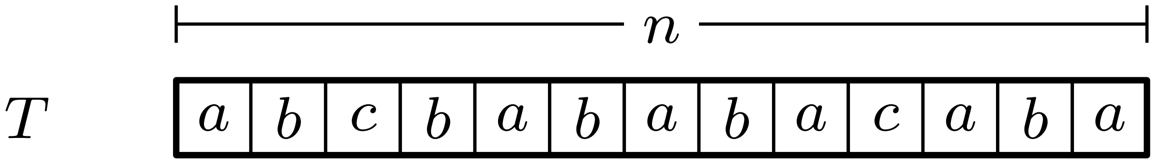


$n$

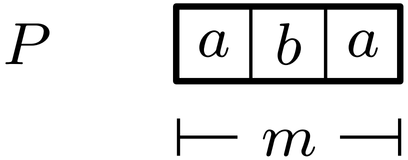
| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $T$ | $a$ | $b$ | $c$ | $b$ | $a$ | $b$ | $a$ | $b$ | $a$ | $c$ | $a$ | $b$ | $a$ |

# Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...

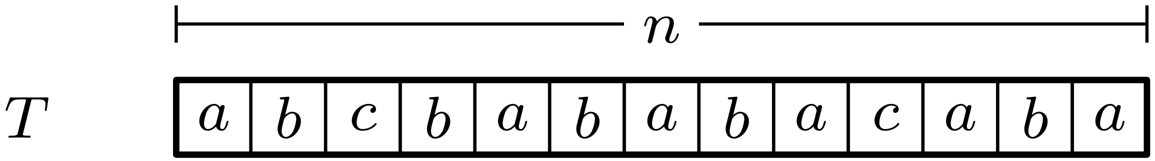


After preprocessing, a **query** is a pattern $P$ (length $m$),

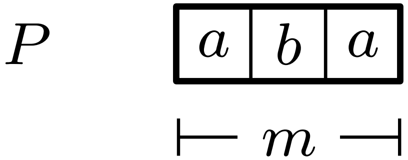


# Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries…



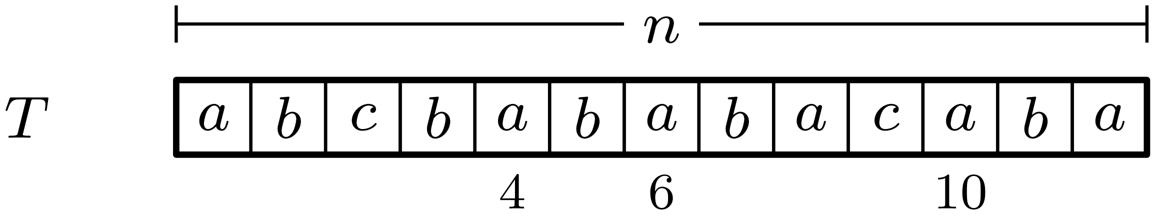
After preprocessing, a **query** is a pattern $P$ (length $m$),



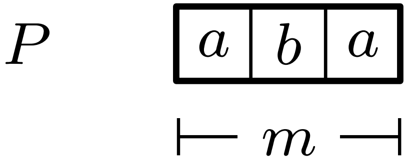
the output is a list of all matches in $T$.

# Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries. . .



After preprocessing, a **query** is a pattern $P$ (length $m$),

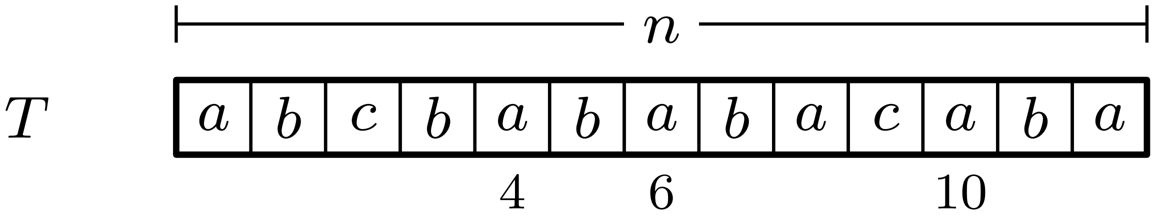


the output is a list of all matches in $T$.

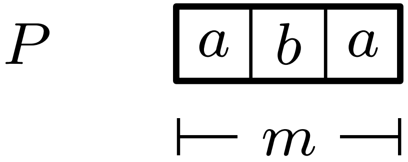
e.g. $4, 6, 10$

# Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...



After preprocessing, a **query** is a pattern $P$ (length $m$),



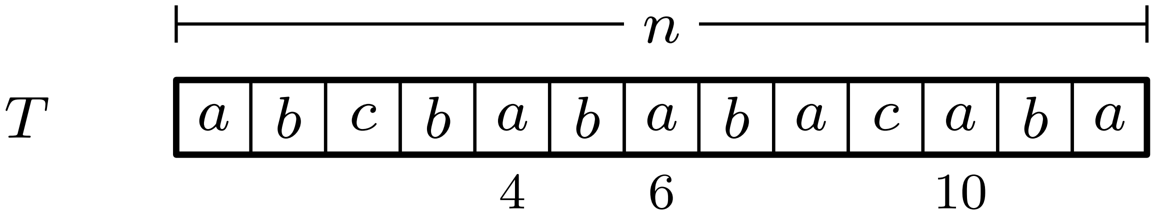
the output is a list of all matches in $T$.

e.g. $4, 6, 10$

- A naive algorithm takes $O(n)$ query time (using KMP)

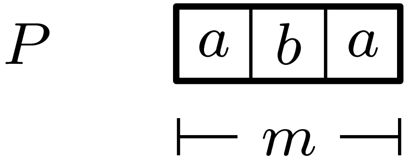
# Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...



After preprocessing, a **query** is a pattern $P$ (length $m$),

the output is a list of all matches in $T$.



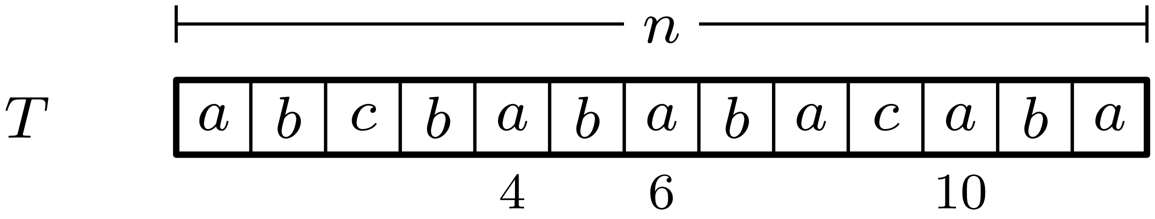
e.g. $4, 6, 10$

- A naive algorithm takes $O(n)$ query time (using KMP)
- We want a query time which depends only on $m$ and occ

- occ is the number of occurences (matches)

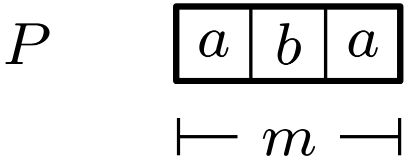
# Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries. . .



After preprocessing, a **query** is a pattern $P$ (length $m$),

the output is a list of all matches in $T$.



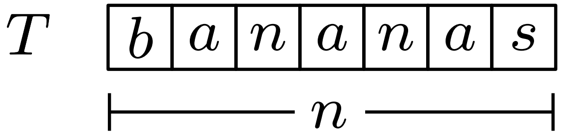
e.g. $4, 6, 10$

- A naive algorithm takes $O(n)$ query time (using KMP)
- We want a query time which depends only on $m$ and occ

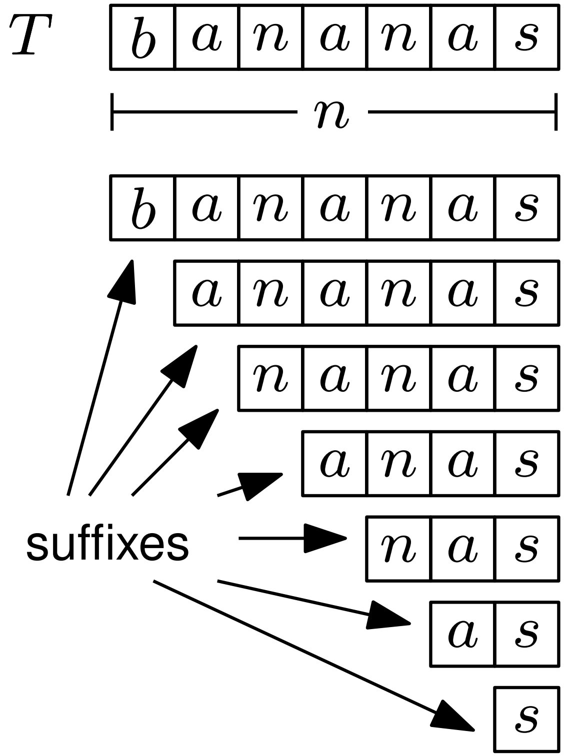
  \- occ is the number of occurences (matches)

- We also want $O(n)$ space and fast preprocessing (prep.) time

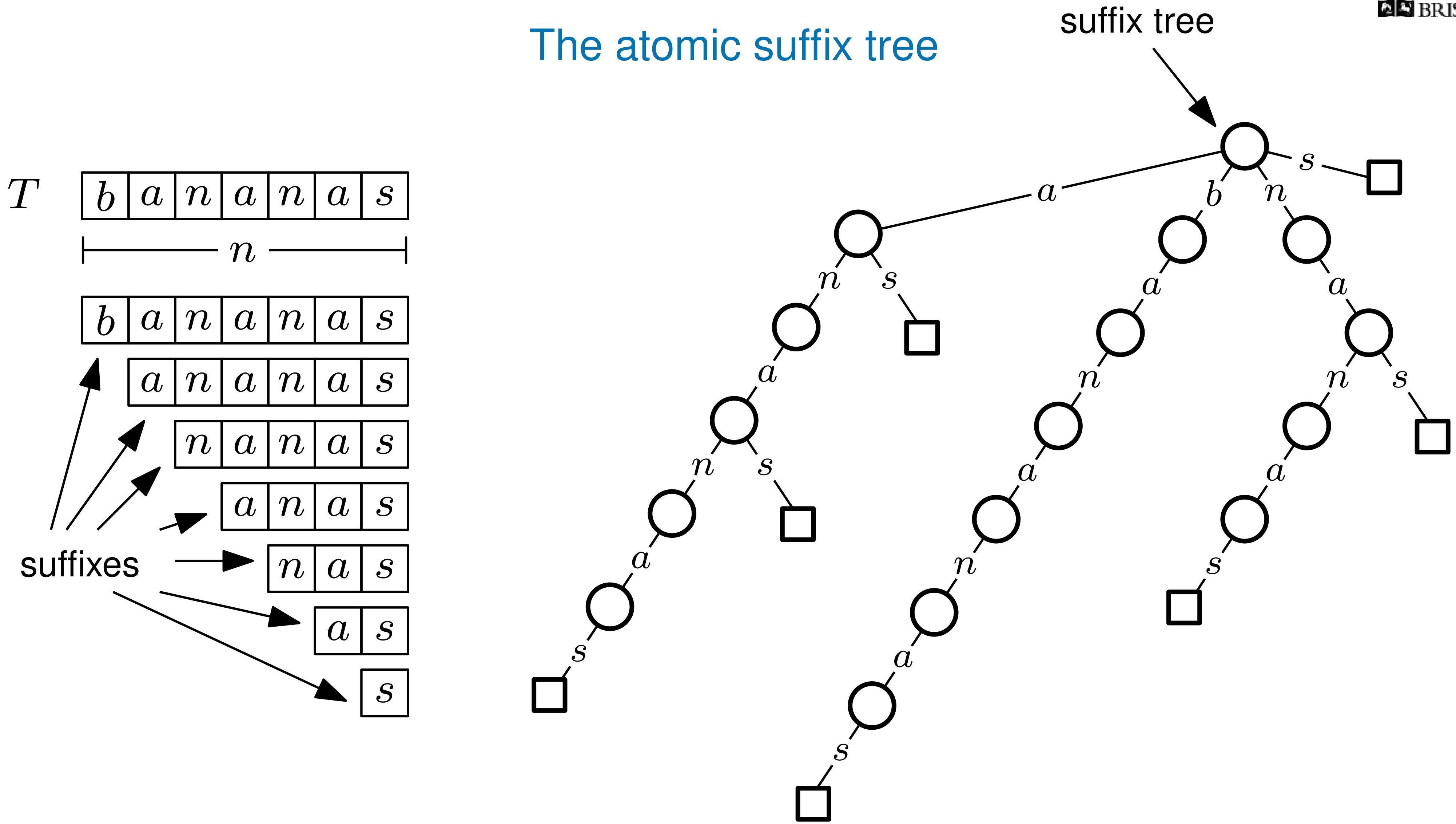
# The atomic suffix tree



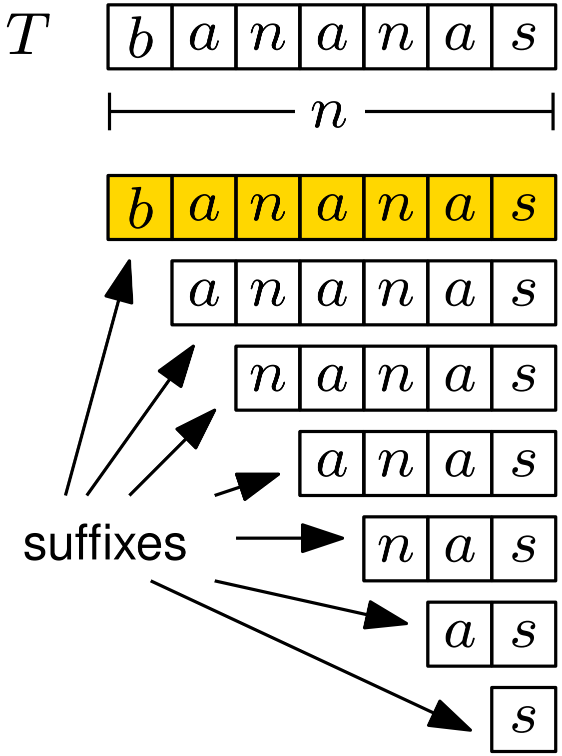
# The atomic suffix tree

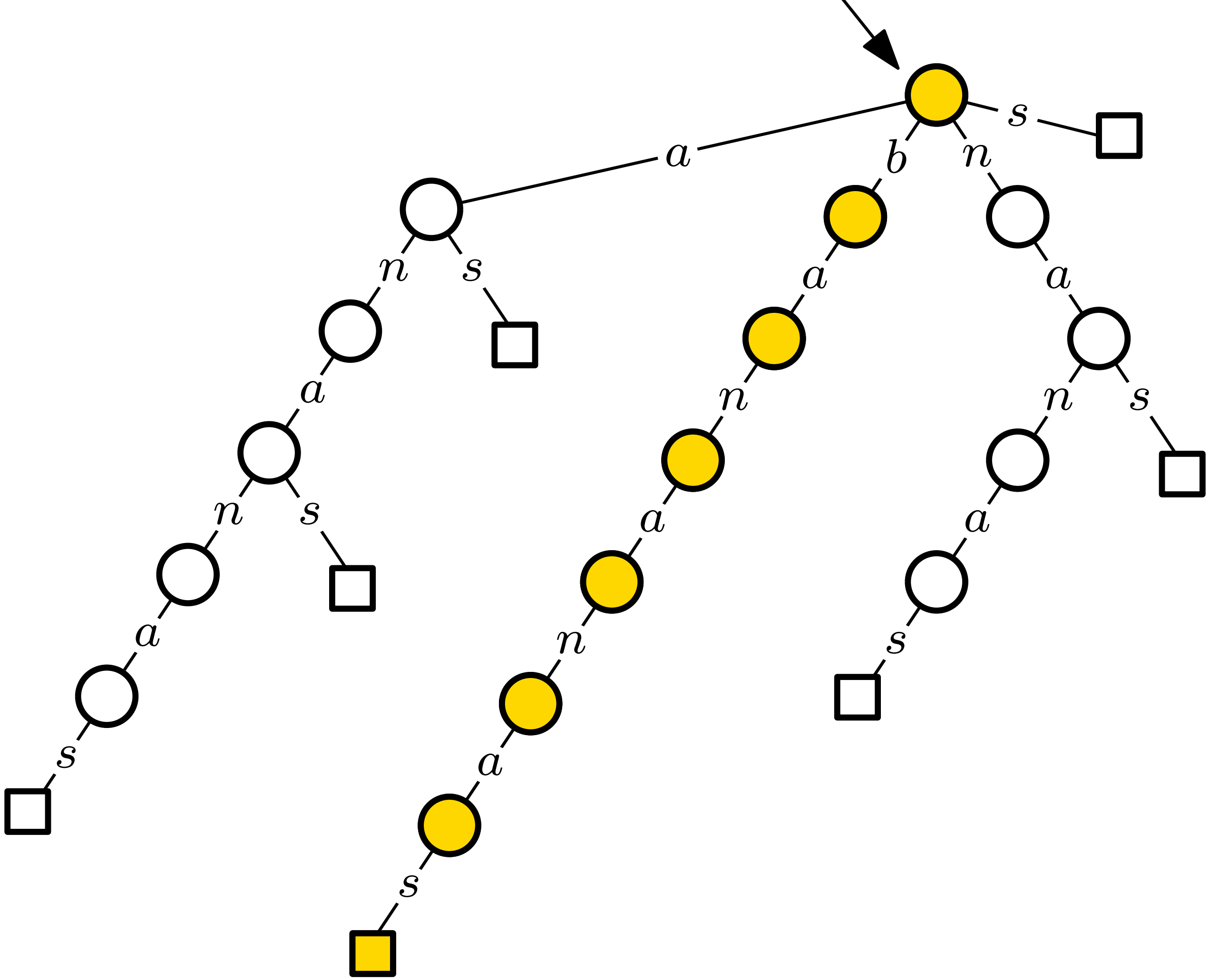


# The atomic suffix tree

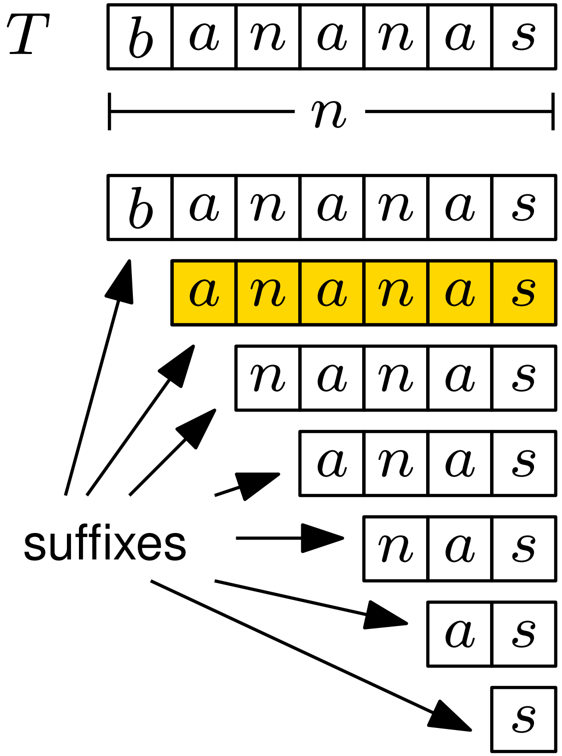


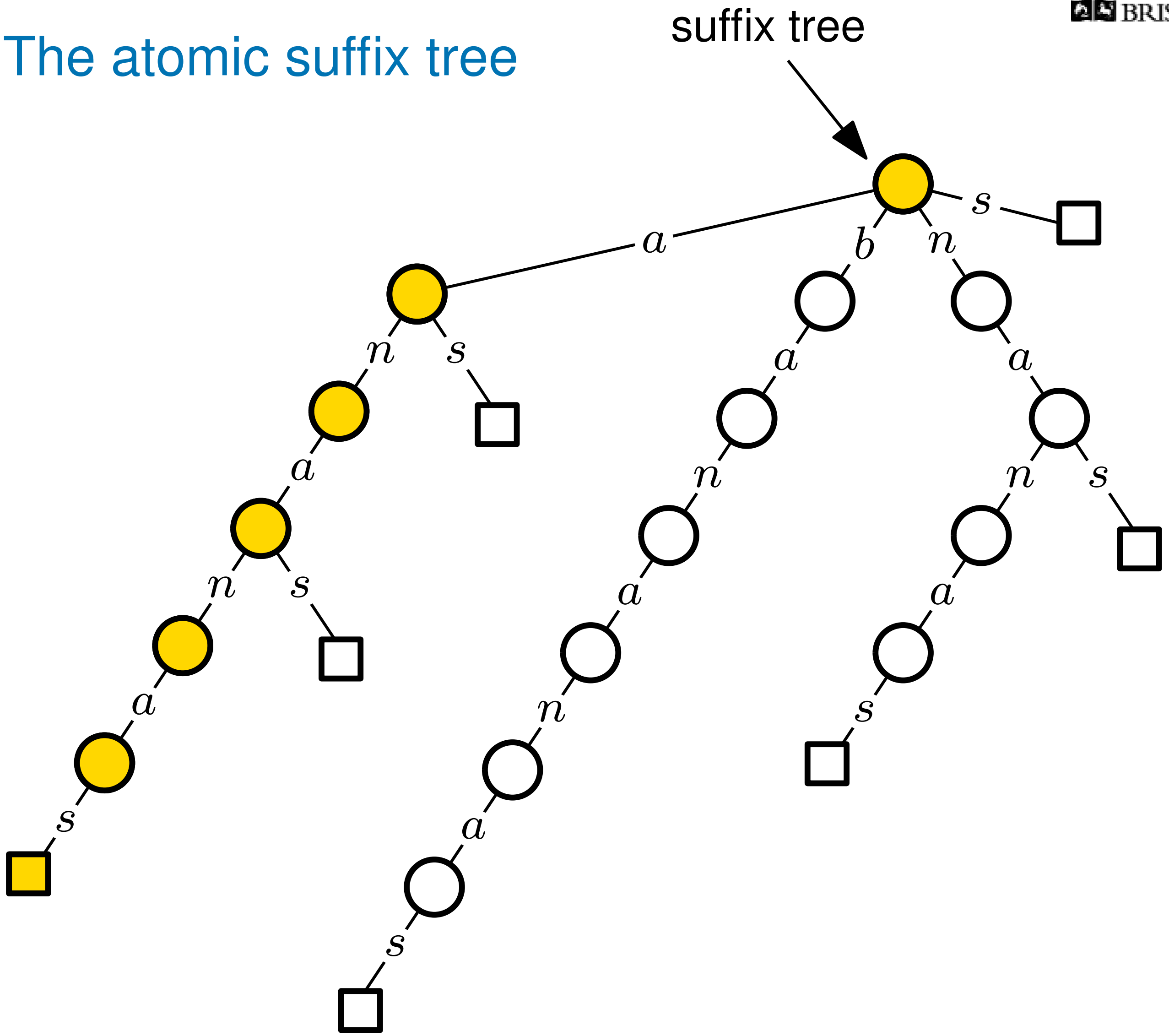
# The atomic suffix tree





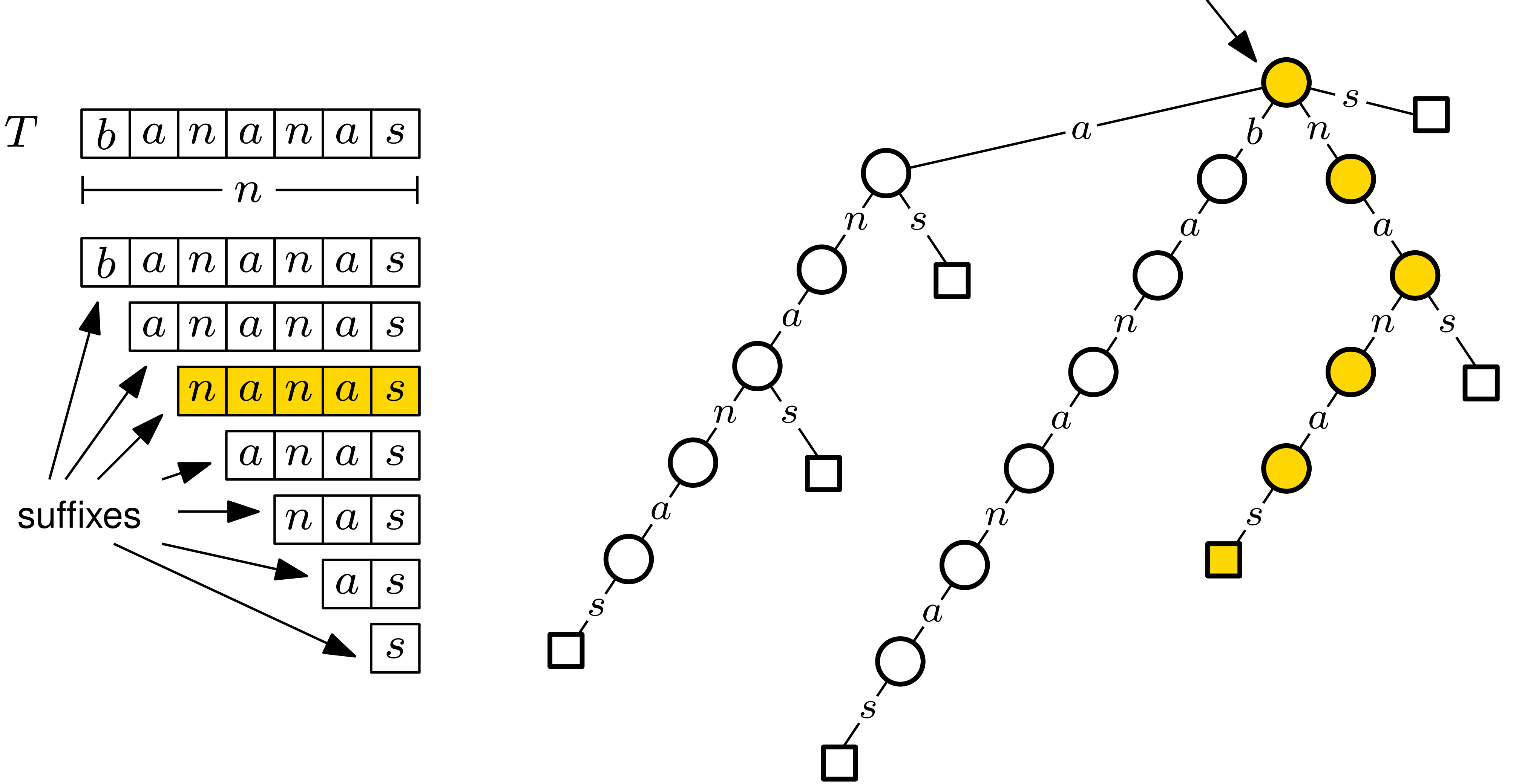
# The atomic suffix tree



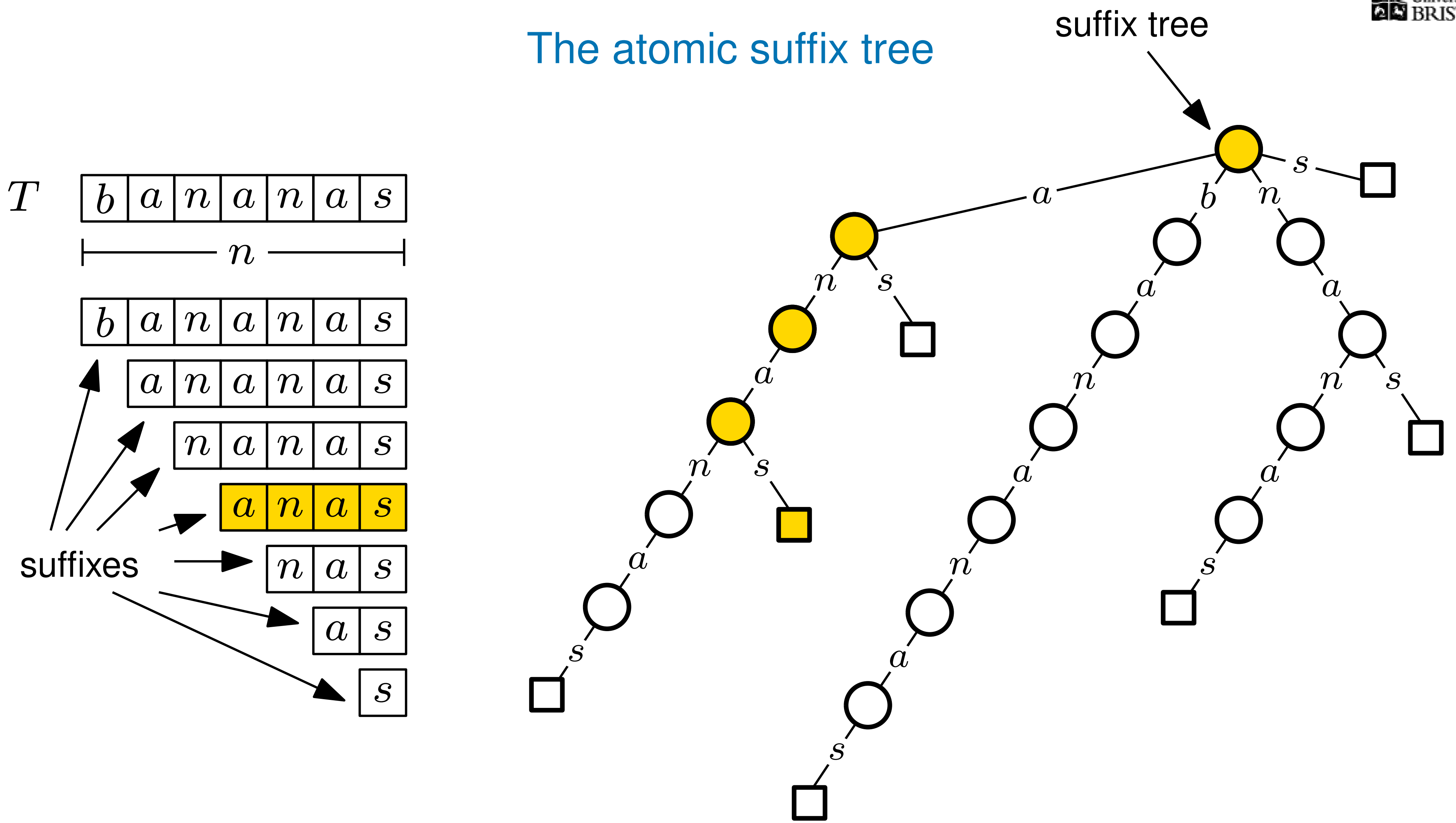


# The atomic suffix tree

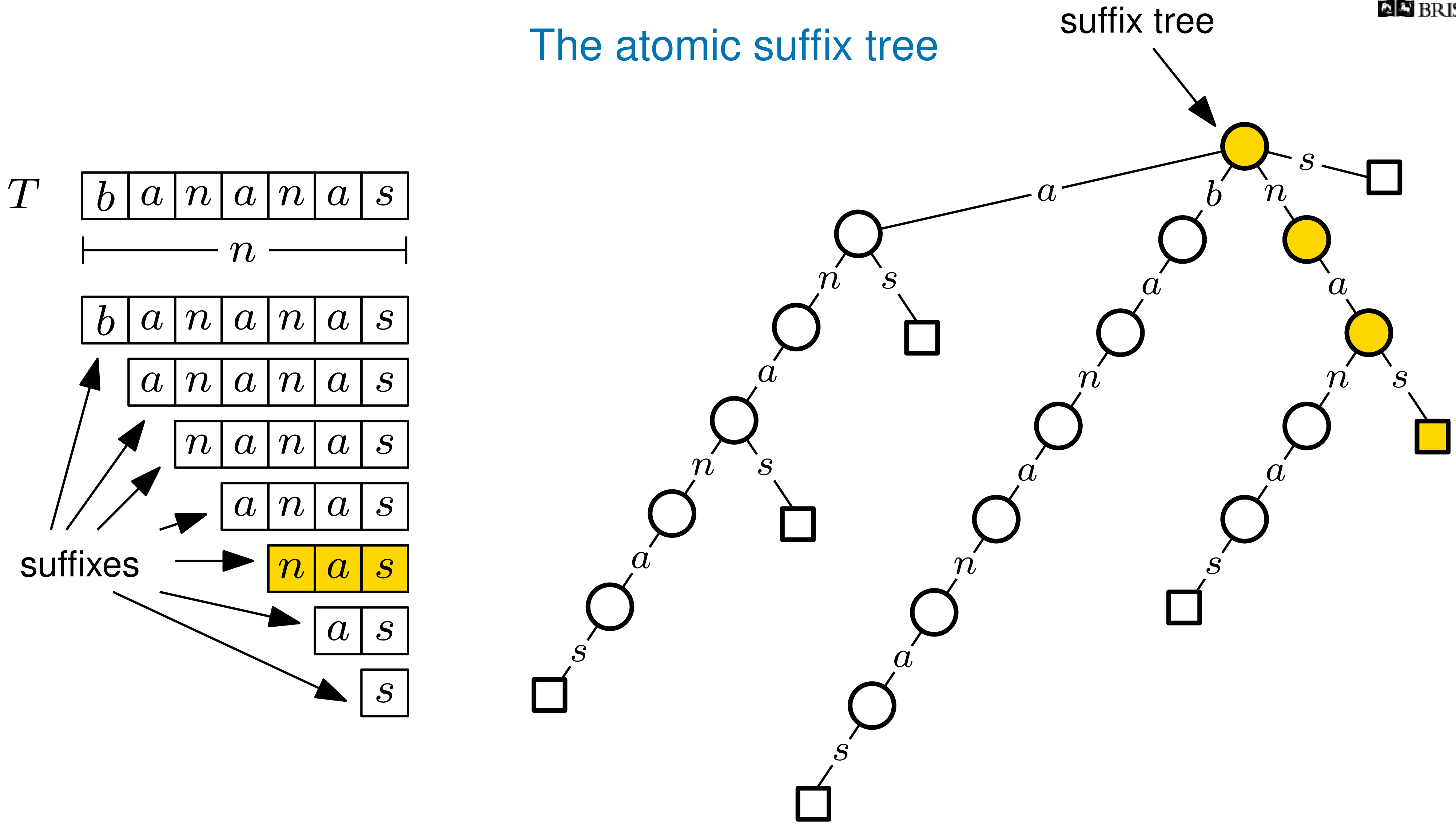
suffix tree



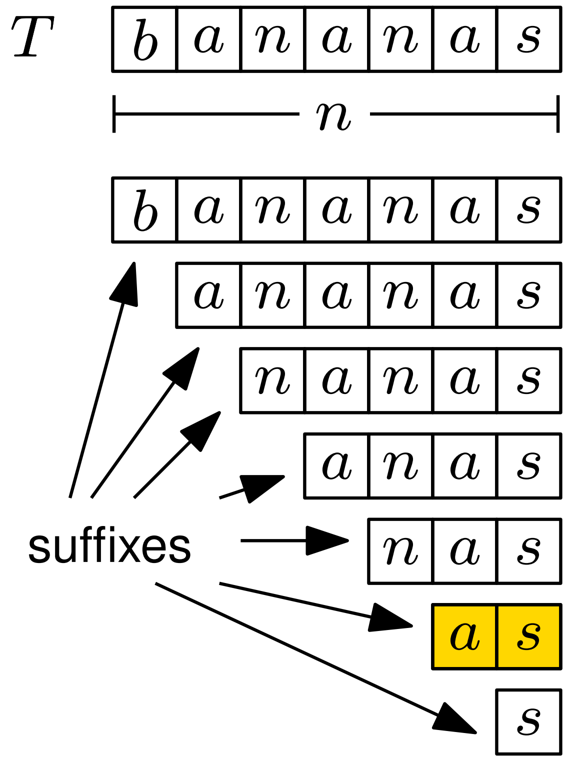
# The atomic suffix tree



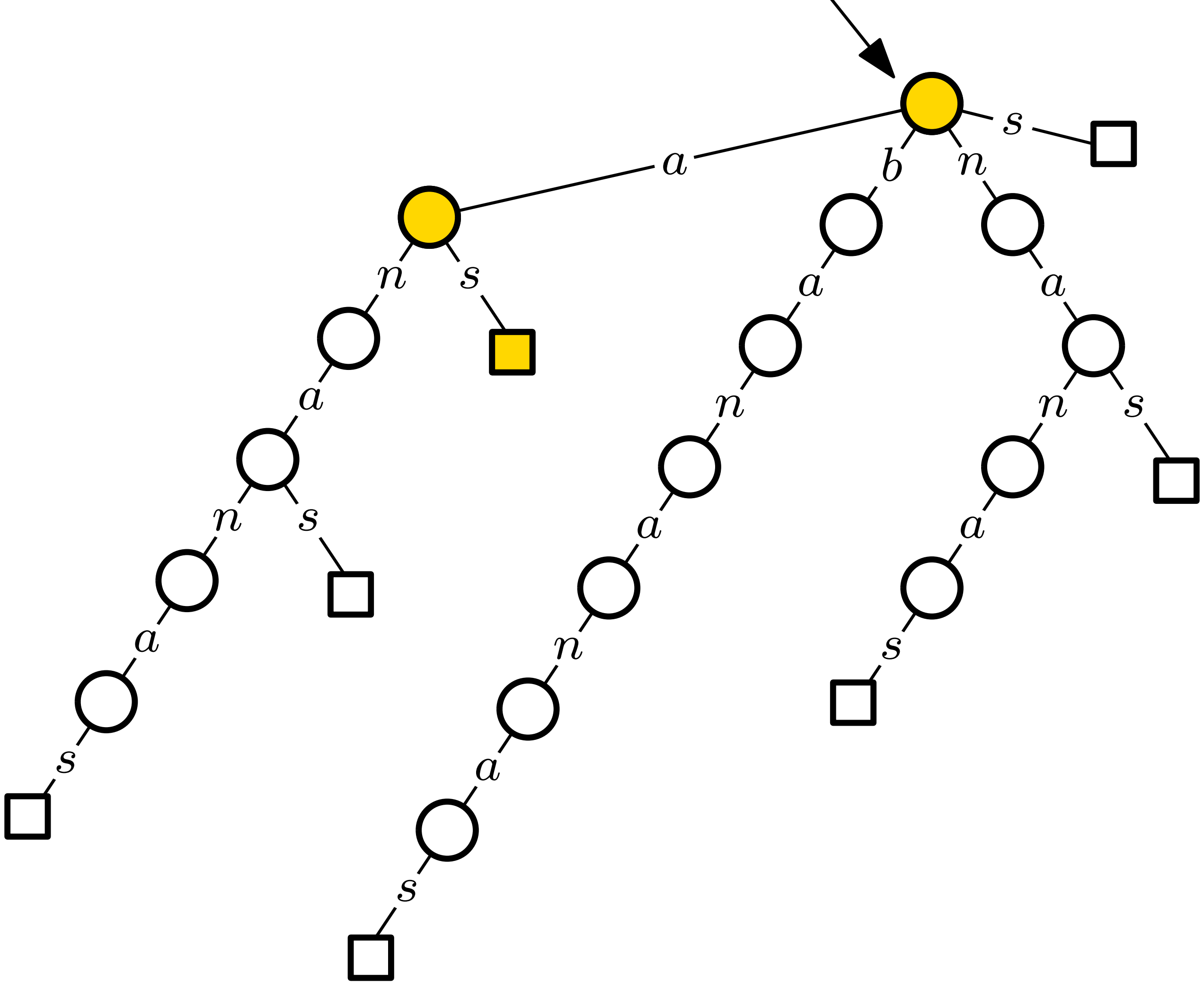
# The atomic suffix tree



# The atomic suffix tree

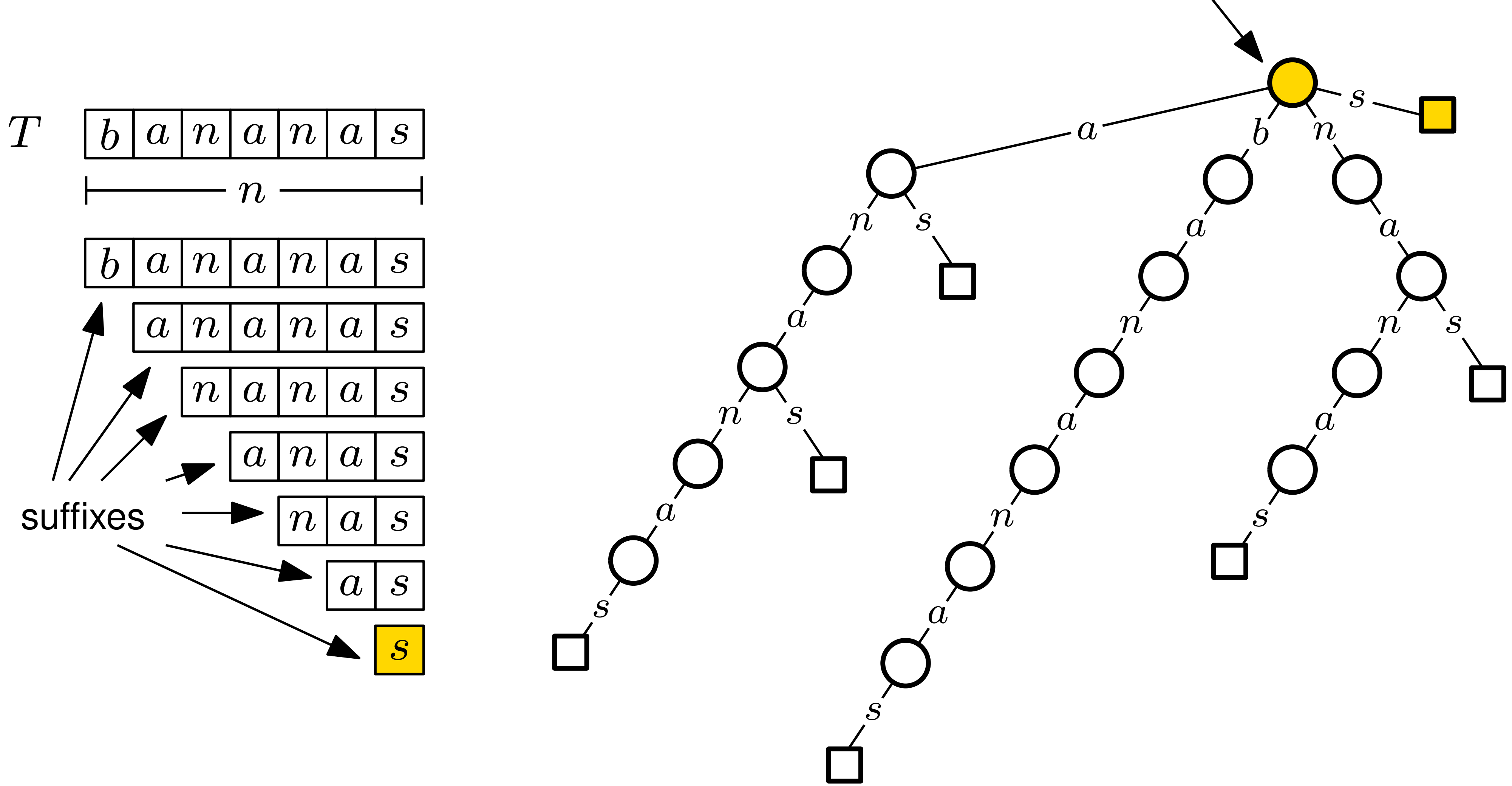


suffix tree



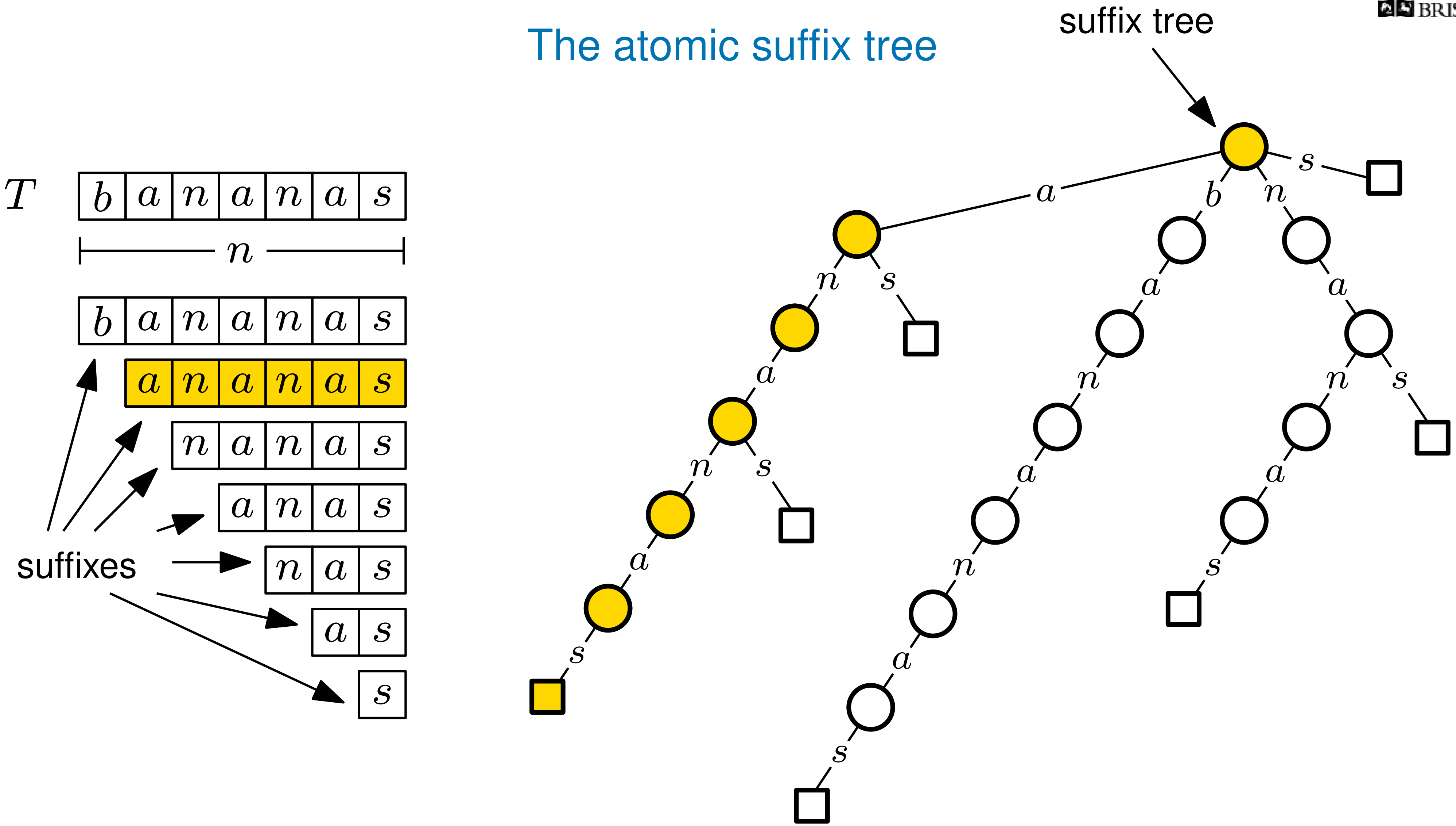
# The atomic suffix tree

suffix tree

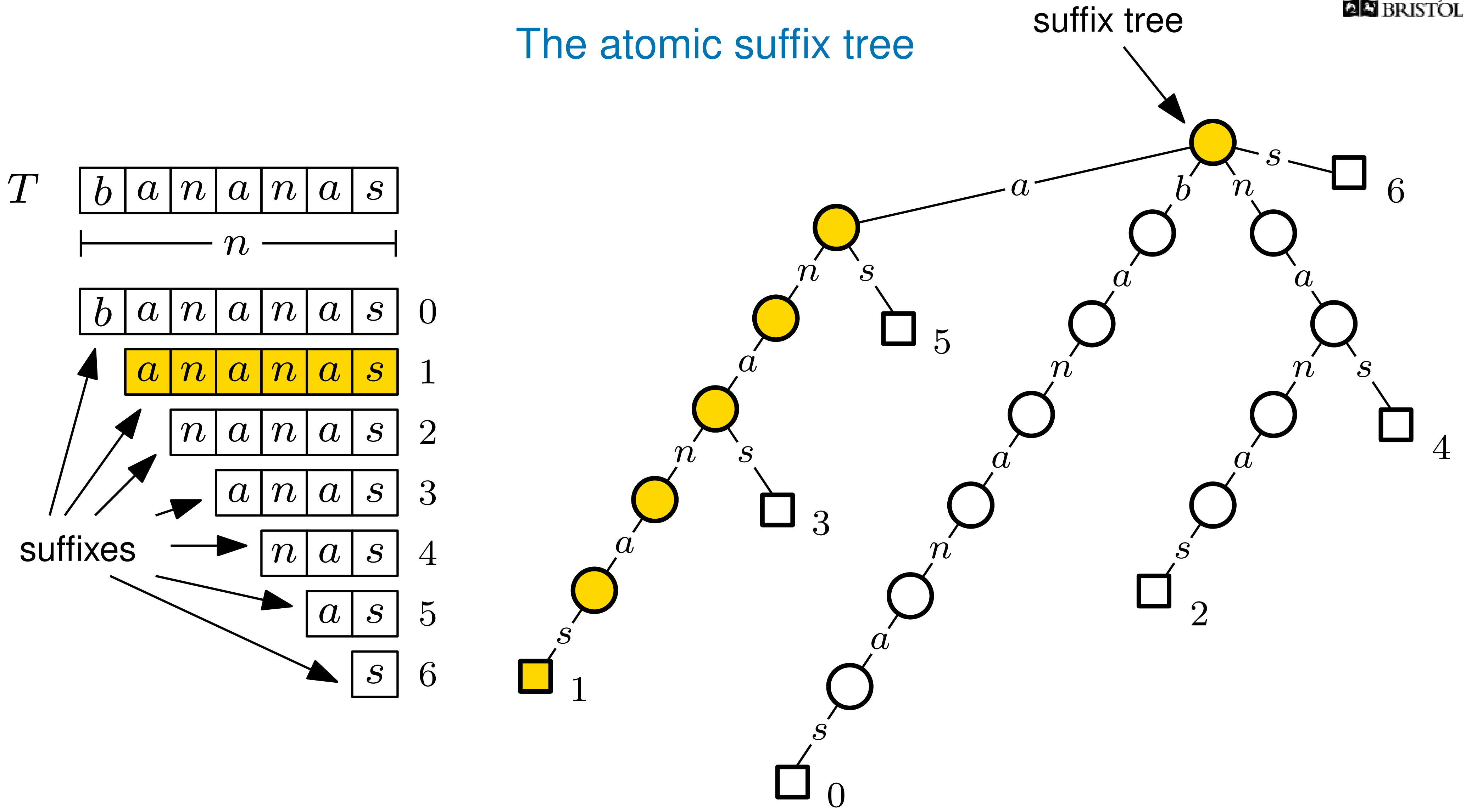


suffixes

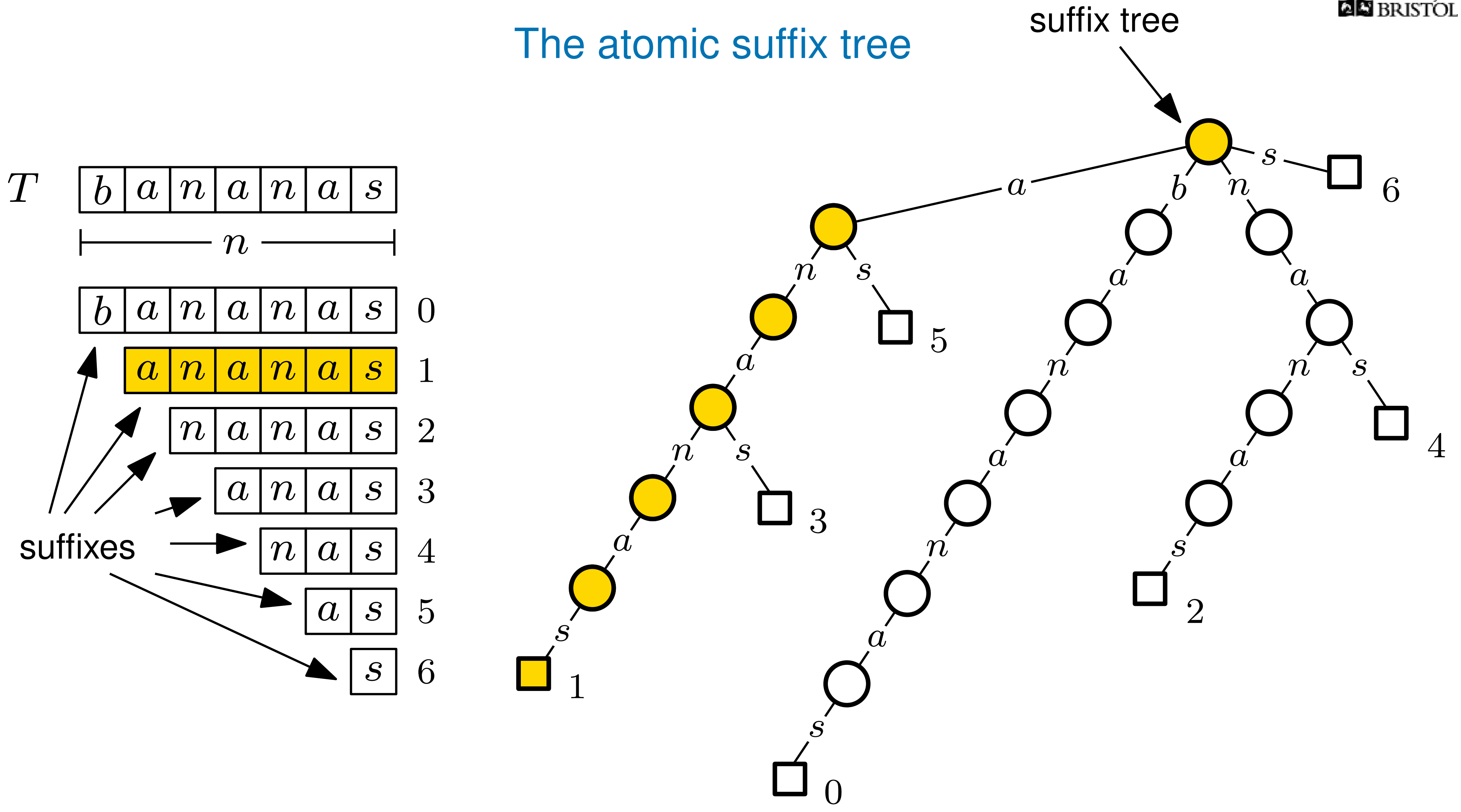
# The atomic suffix tree



# The atomic suffix tree

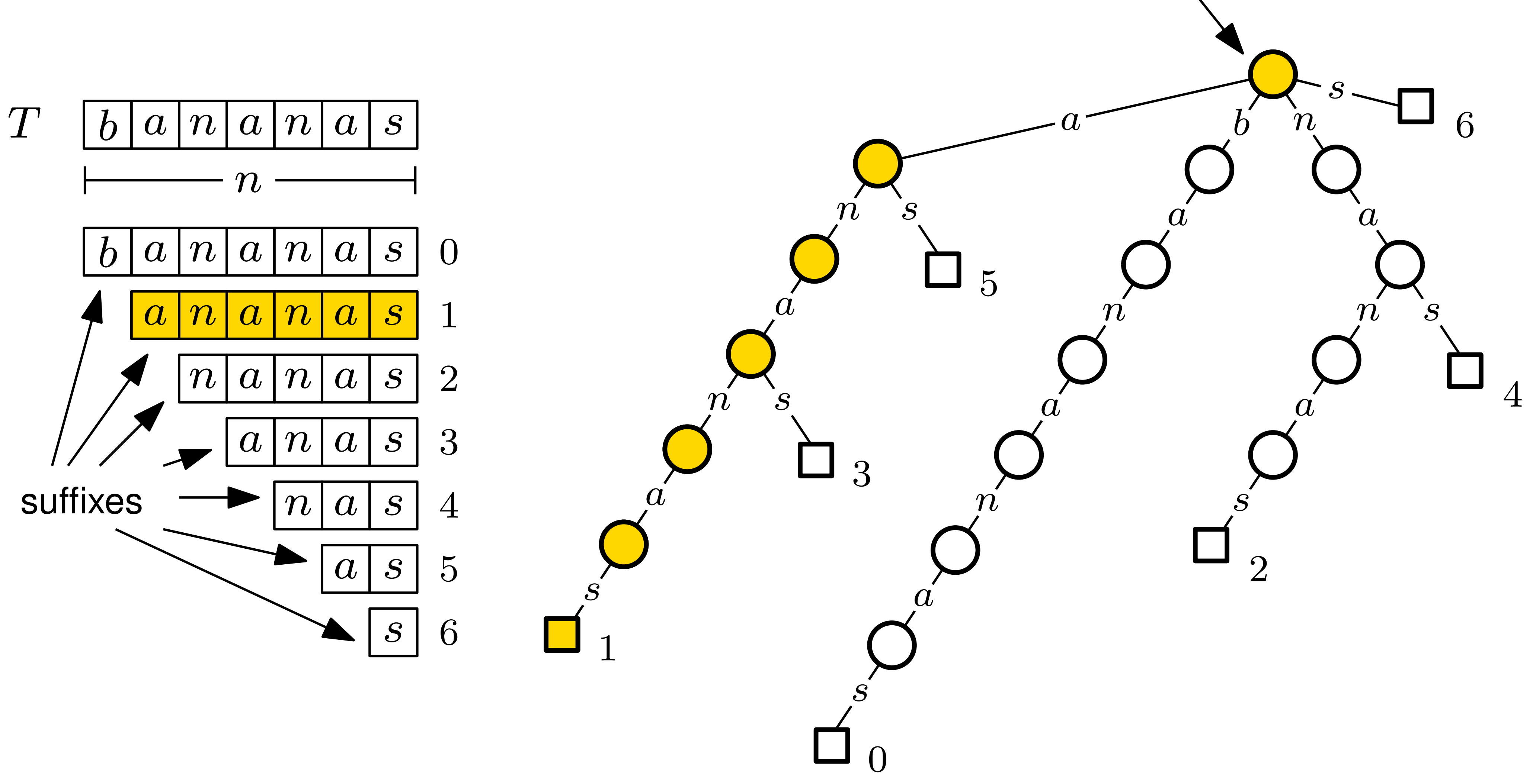


# The atomic suffix tree



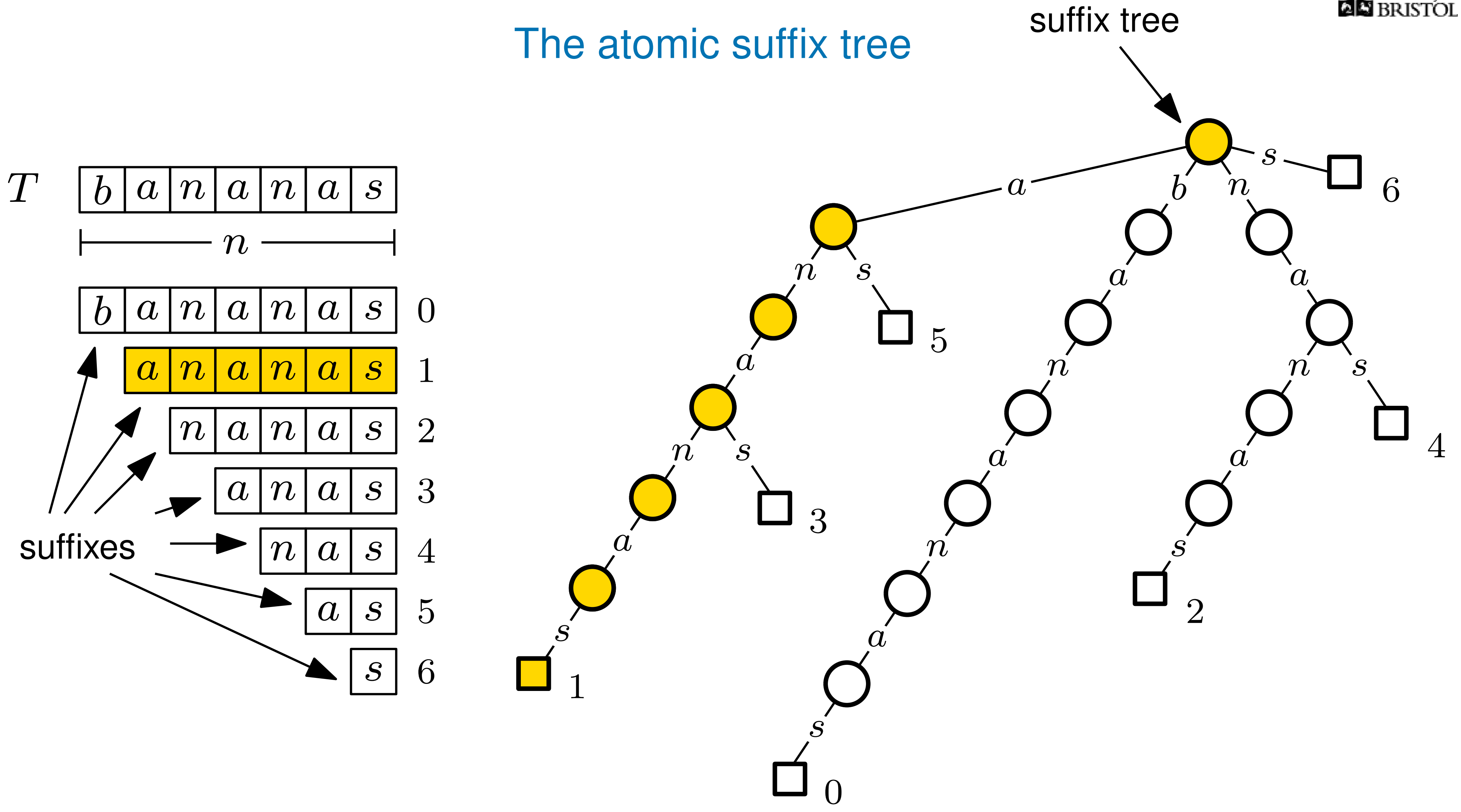
- The suffix tree contains every suffix of $T$ as a root to leaf path

# The atomic suffix tree



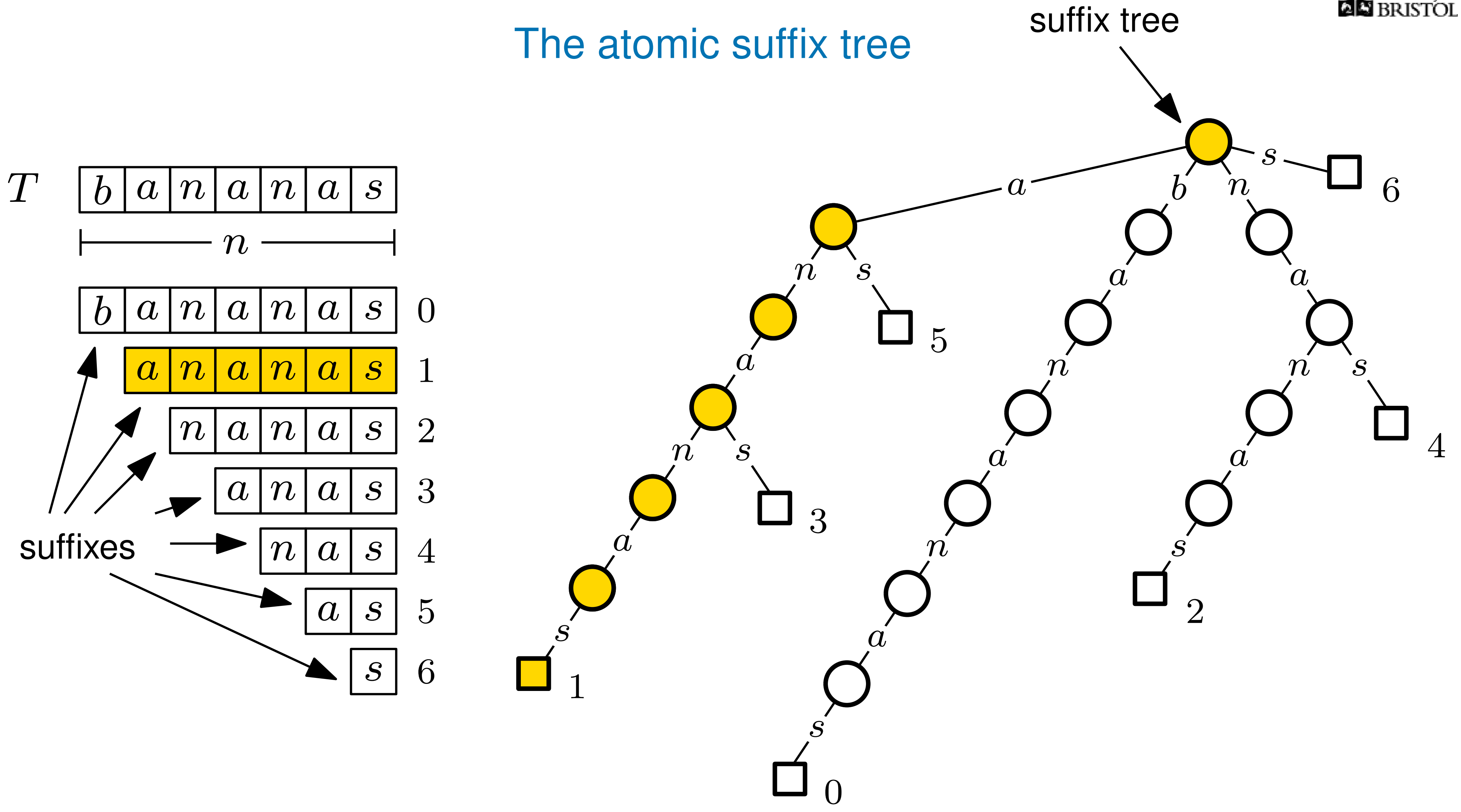
- The suffix tree contains every suffix of $T$ as a root to leaf path
- Every edge is labelled with a character from $T$

# The atomic suffix tree



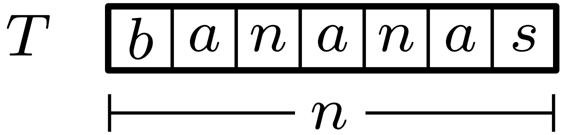
- The suffix tree contains every suffix of $T$ as a root to leaf path
- Every edge is labelled with a character from $T$
- No two edges leaving the same node have the same label

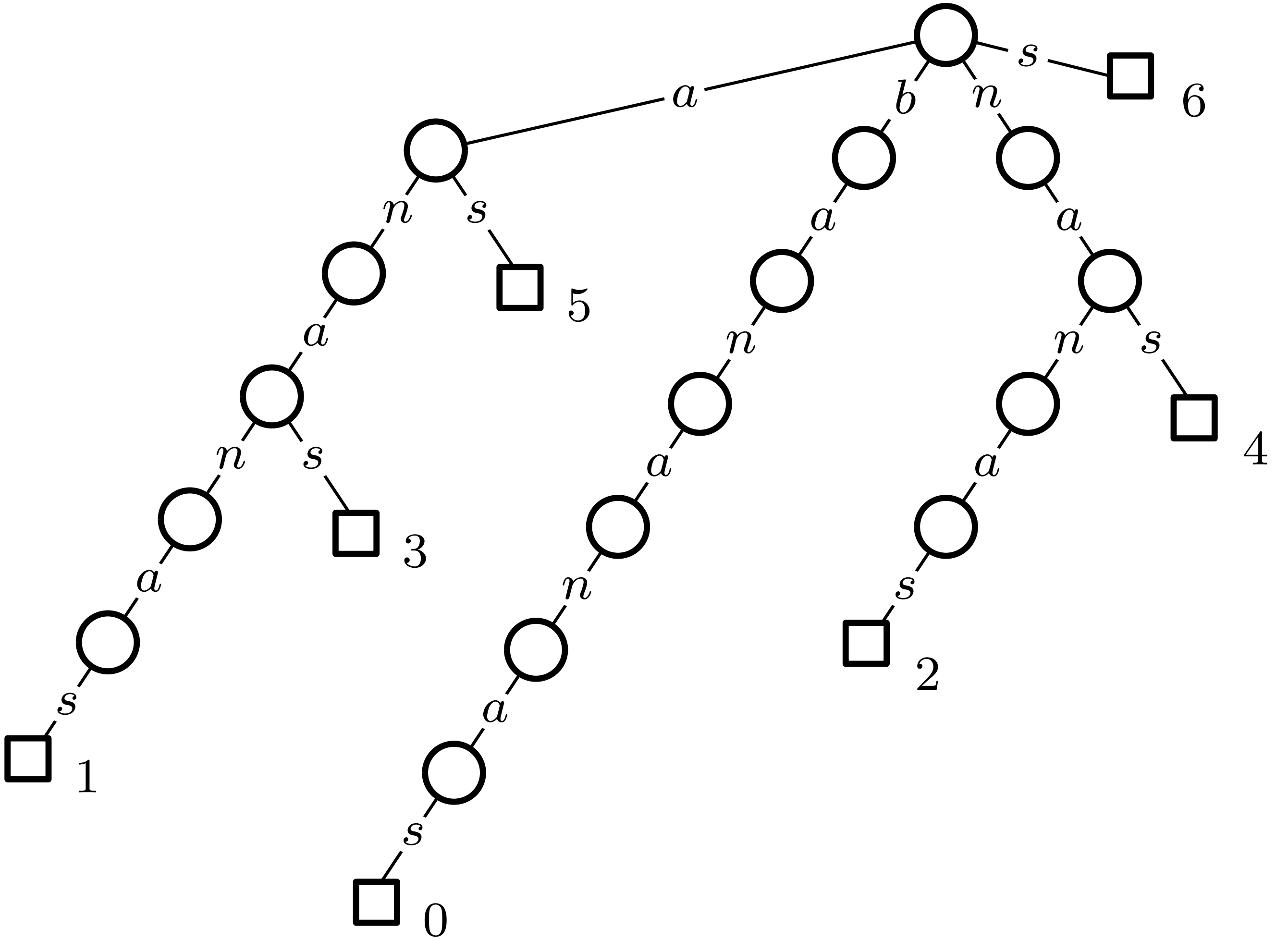
# The atomic suffix tree



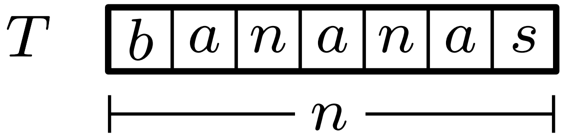
- The suffix tree contains every suffix of $T$ as a root to leaf path
- Every edge is labelled with a character from $T$
- No two edges leaving the same node have the same label
- Each leaf corresponds to a suffix (so there are $n$ leaves)

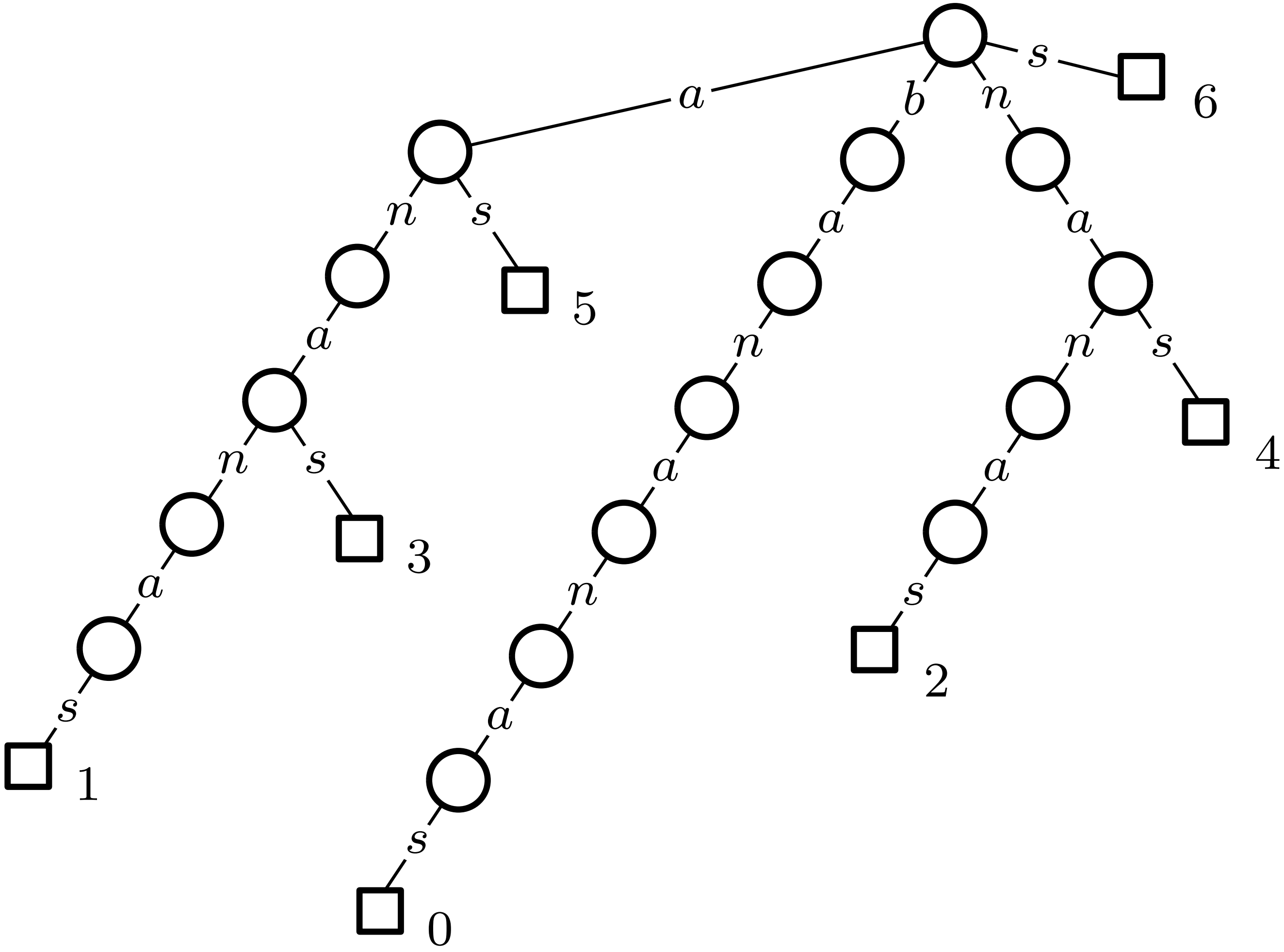
# Searching in an atomic suffix tree





# Searching in an atomic suffix tree





*How do you find a pattern?*

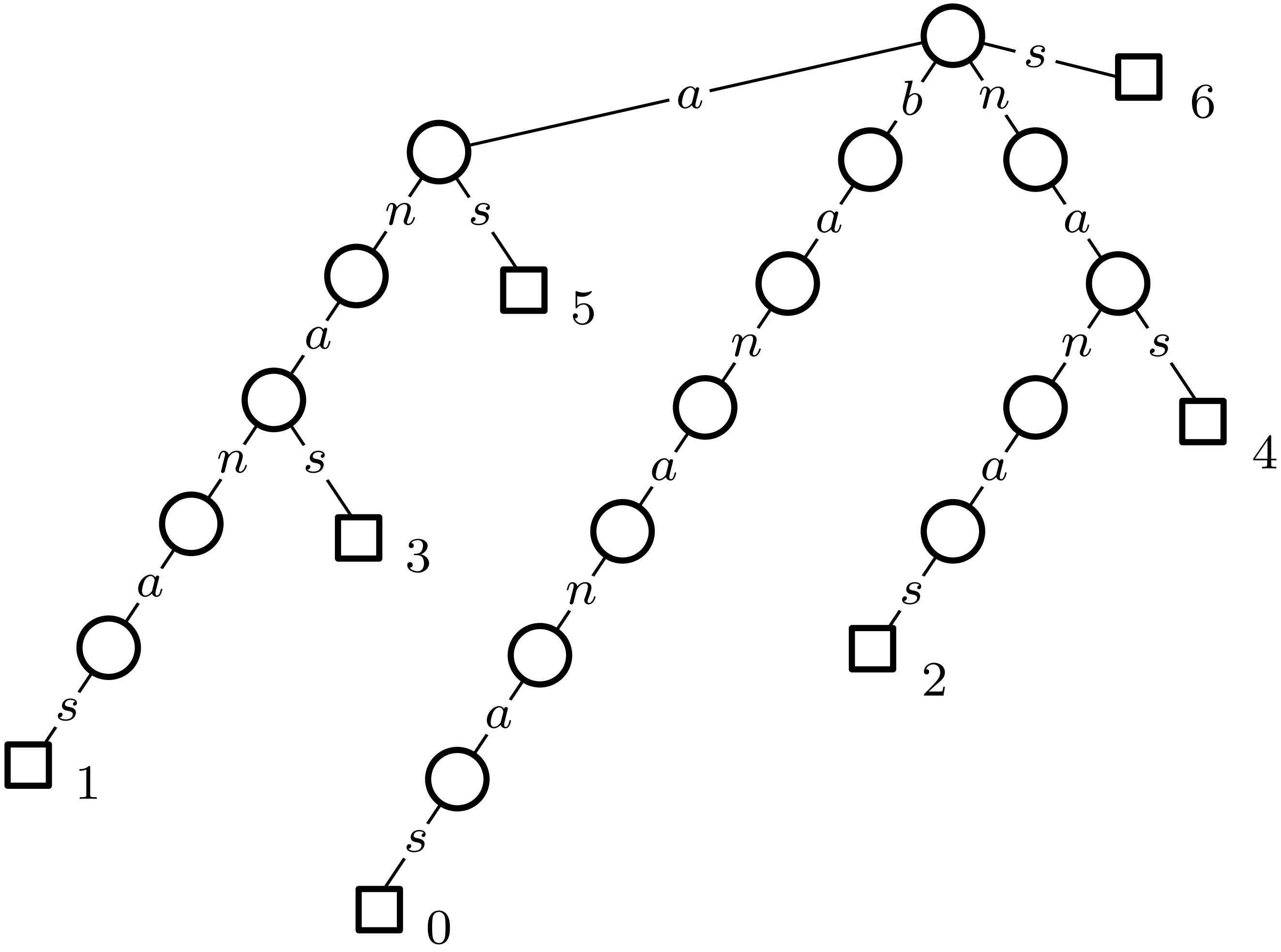
# Searching in an atomic suffix tree

| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
|---|---|---|---|---|---|---|---|

$\vdash n \dashv$

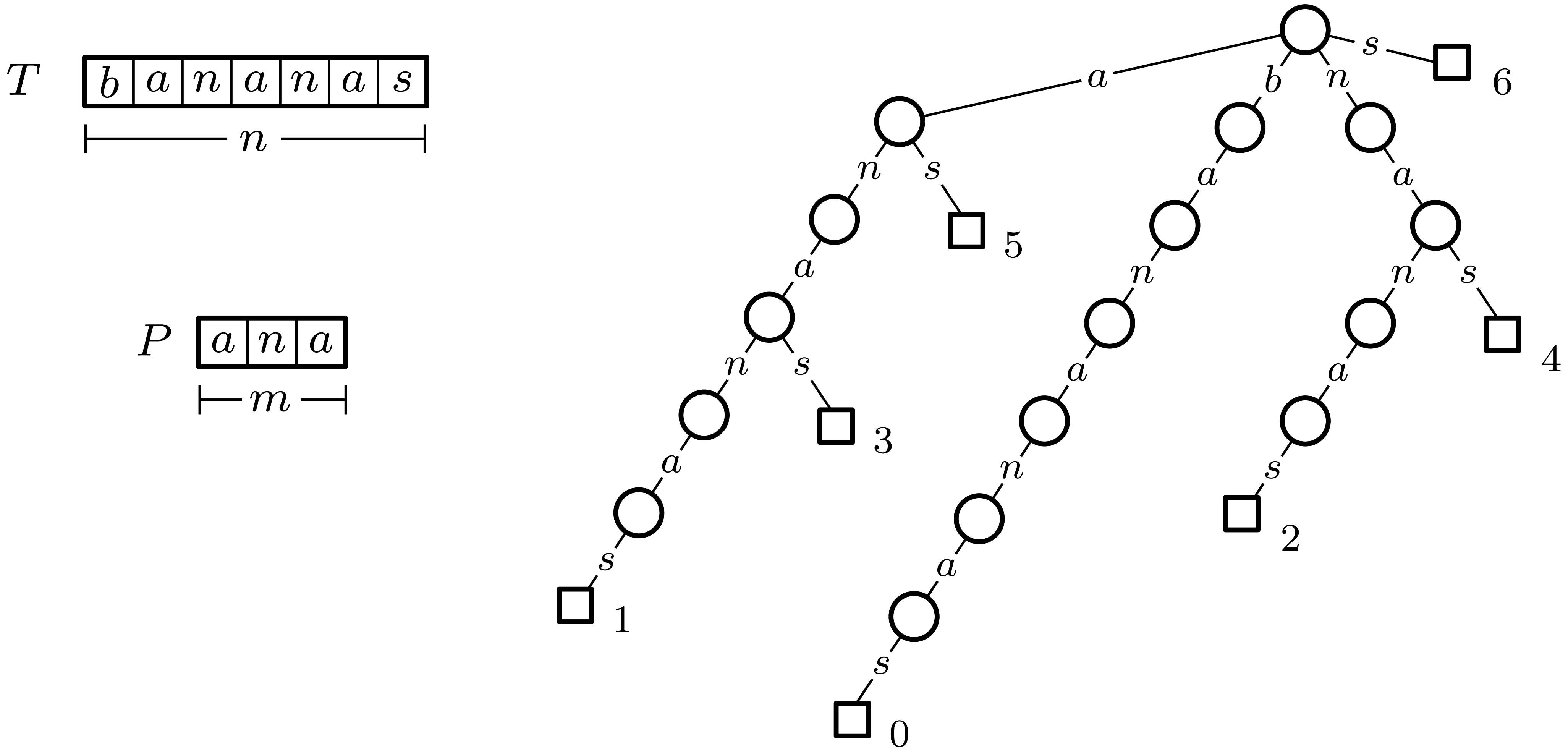
| $P$ | $a$ | $n$ | $a$ |
|---|---|---|---|

$\vdash m \dashv$



*How do you find a pattern?*

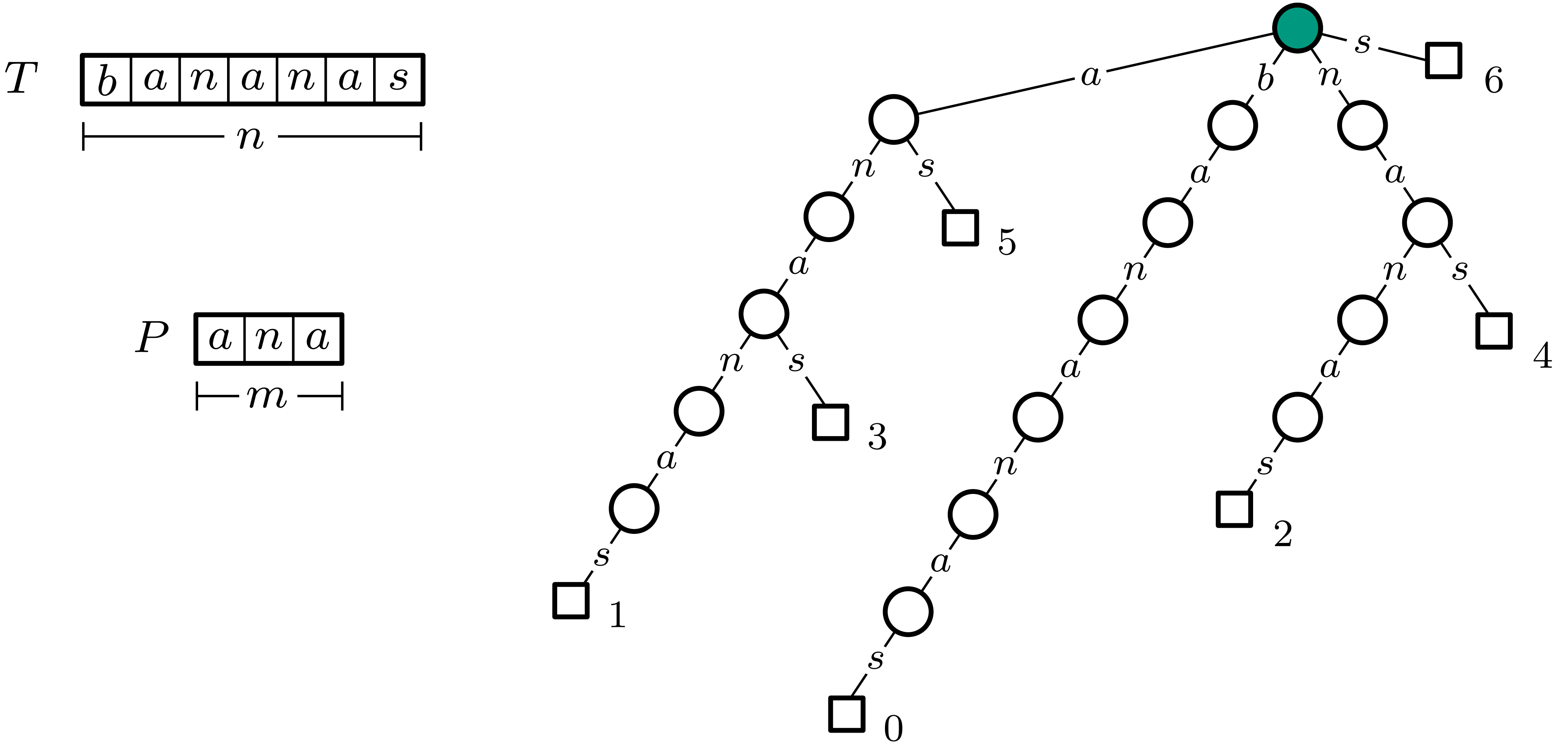
# Searching in an atomic suffix tree



*How do you find a pattern?*

start at the root and walk down the tree

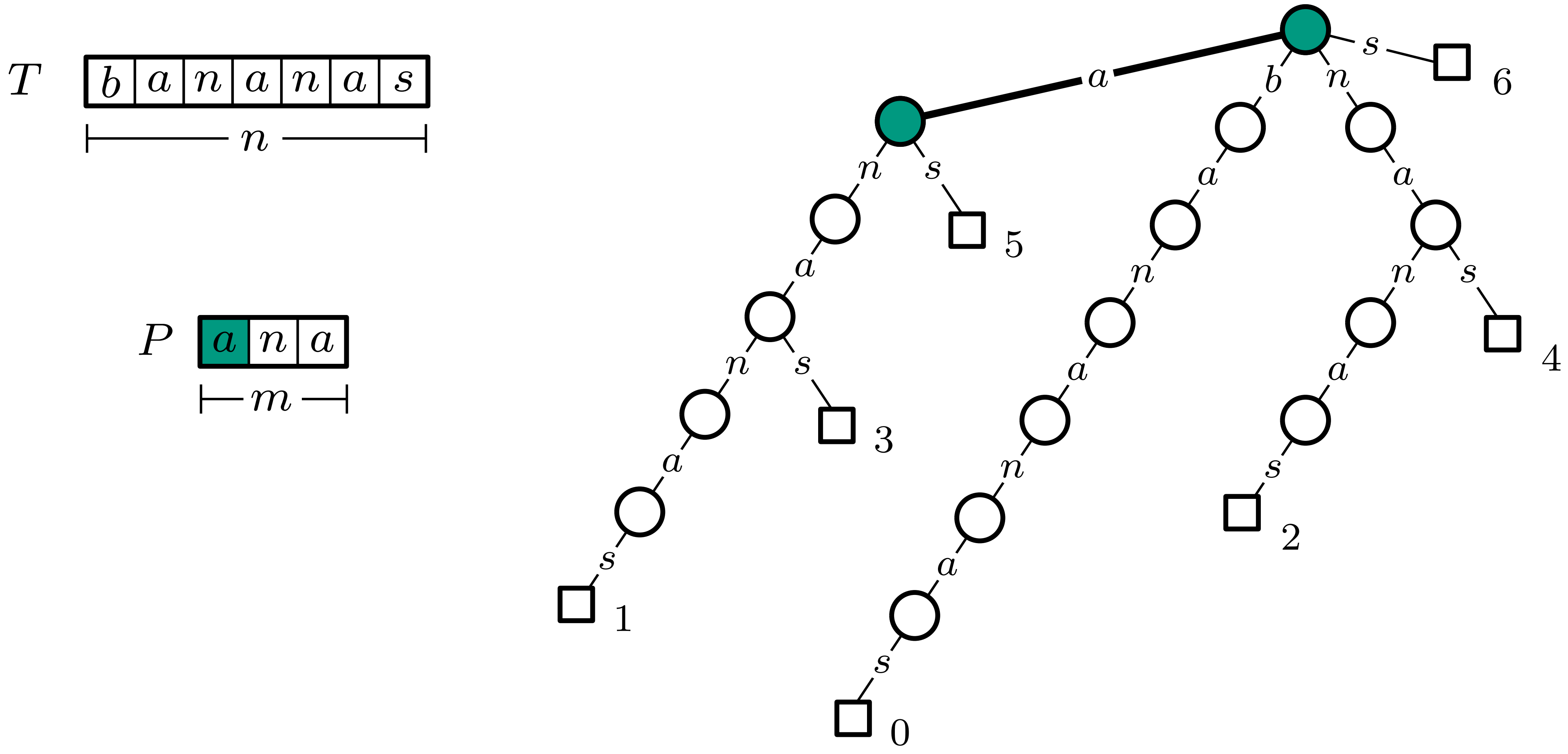
# Searching in an atomic suffix tree



*How do you find a pattern?*

start at the root and walk down the tree

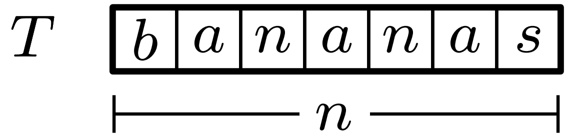
# Searching in an atomic suffix tree

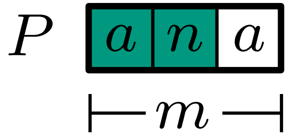


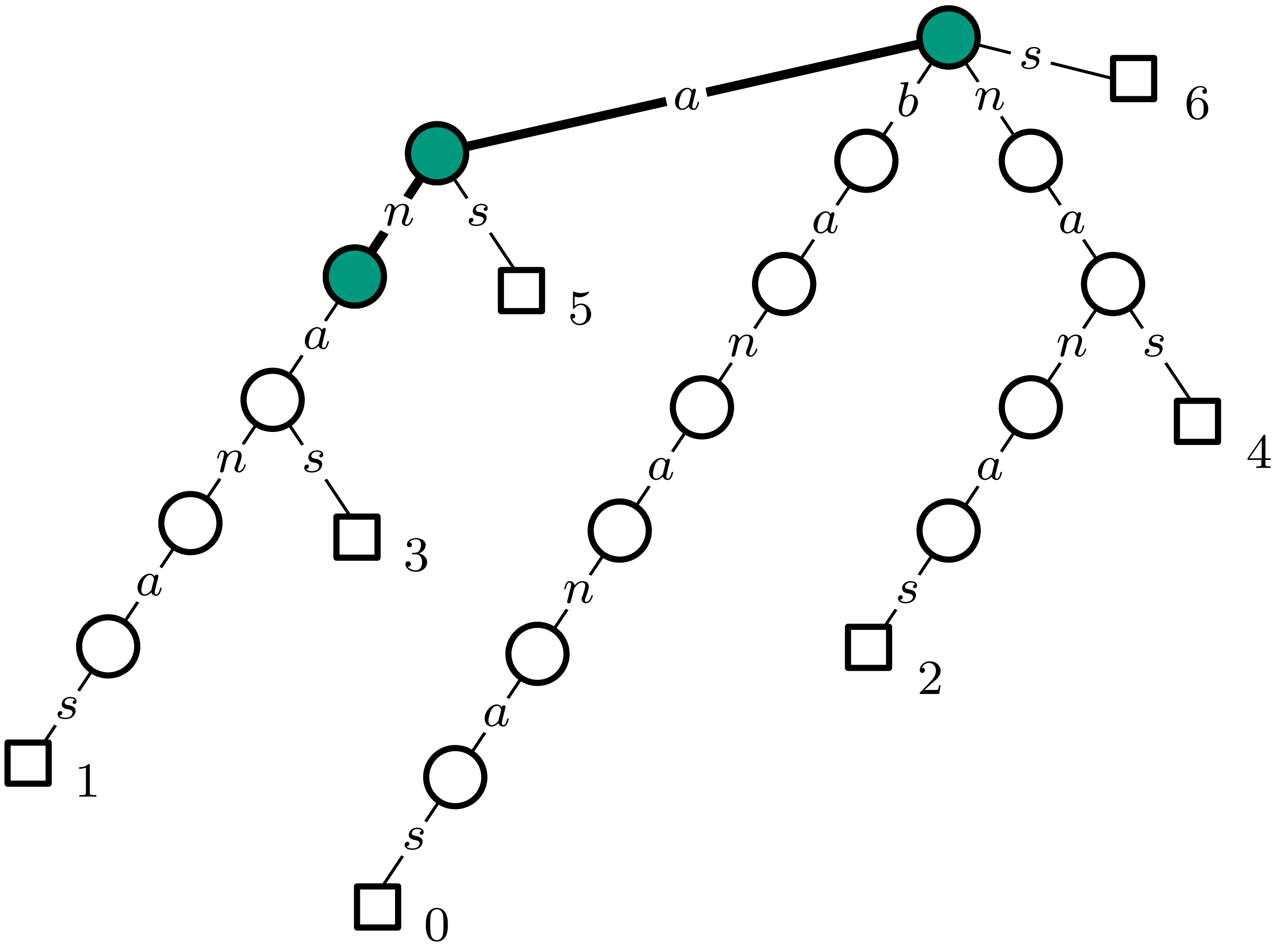
*How do you find a pattern?*

start at the root and walk down the tree

# Searching in an atomic suffix tree



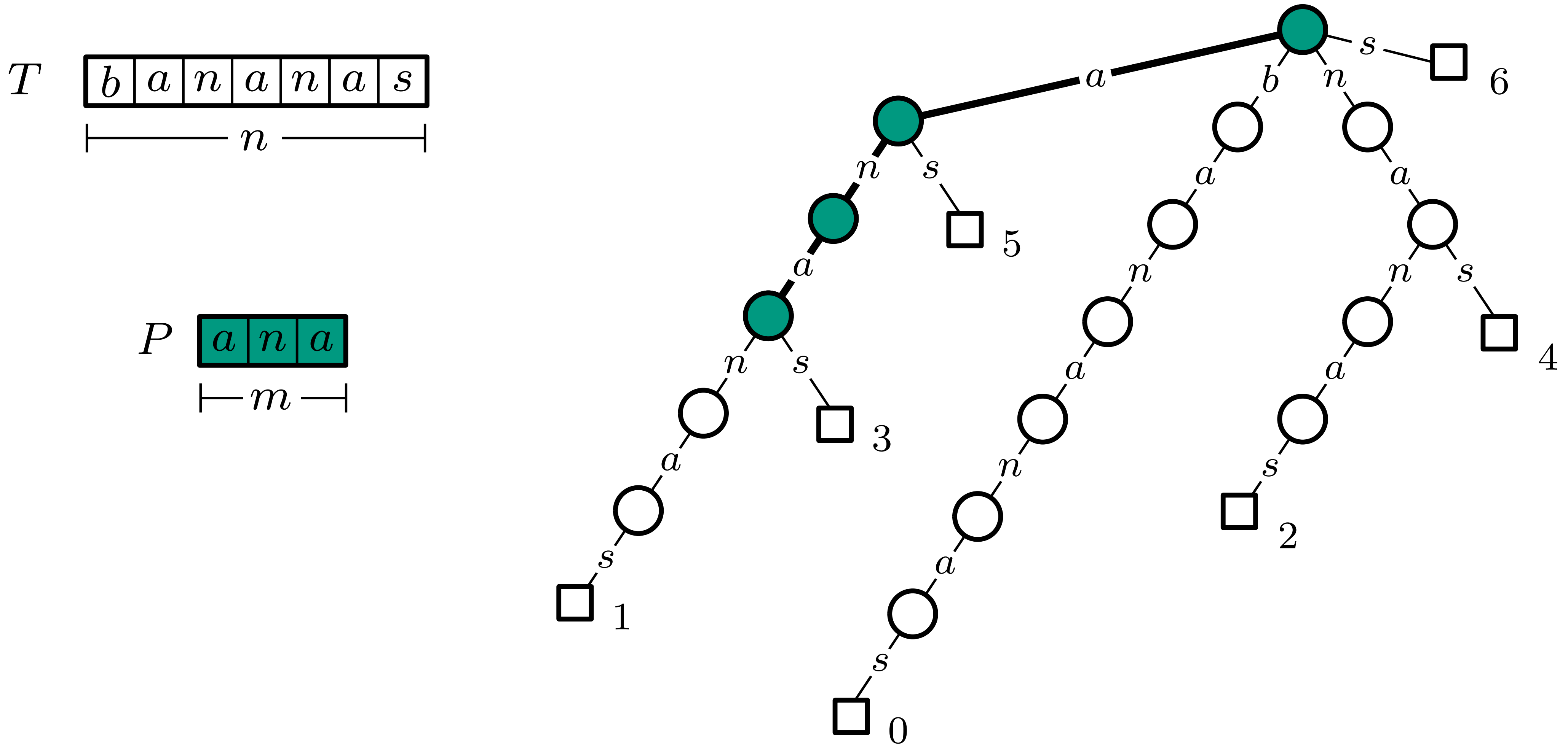




*How do you find a pattern?*

start at the root and walk down the tree

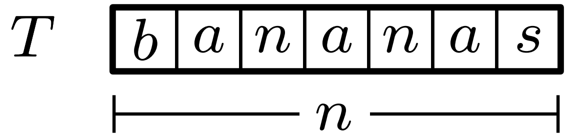
# Searching in an atomic suffix tree

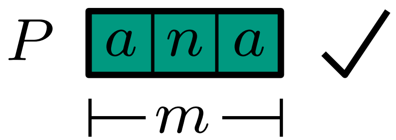


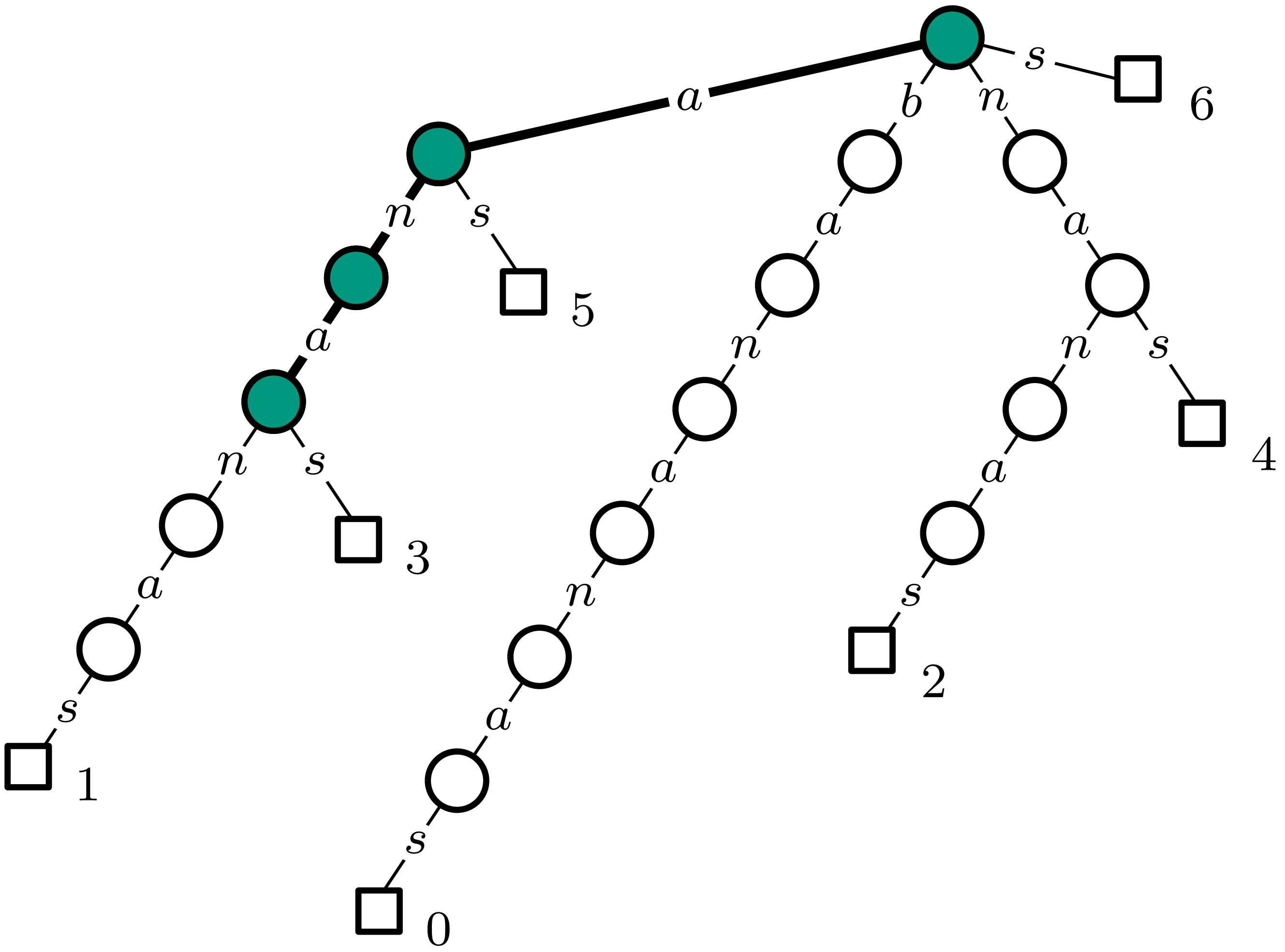
*How do you find a pattern?*

start at the root and walk down the tree

# Searching in an atomic suffix tree





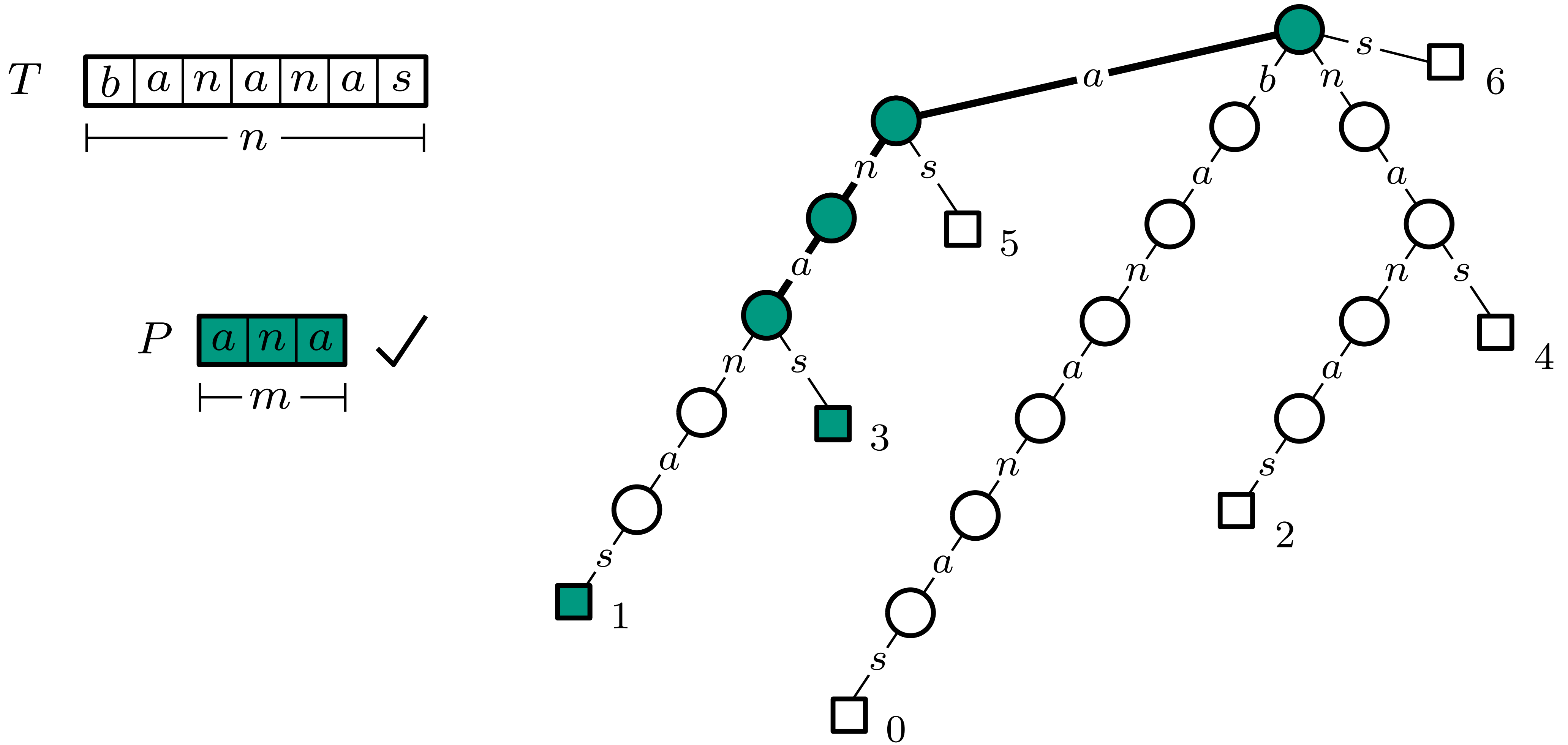


*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in an atomic suffix tree

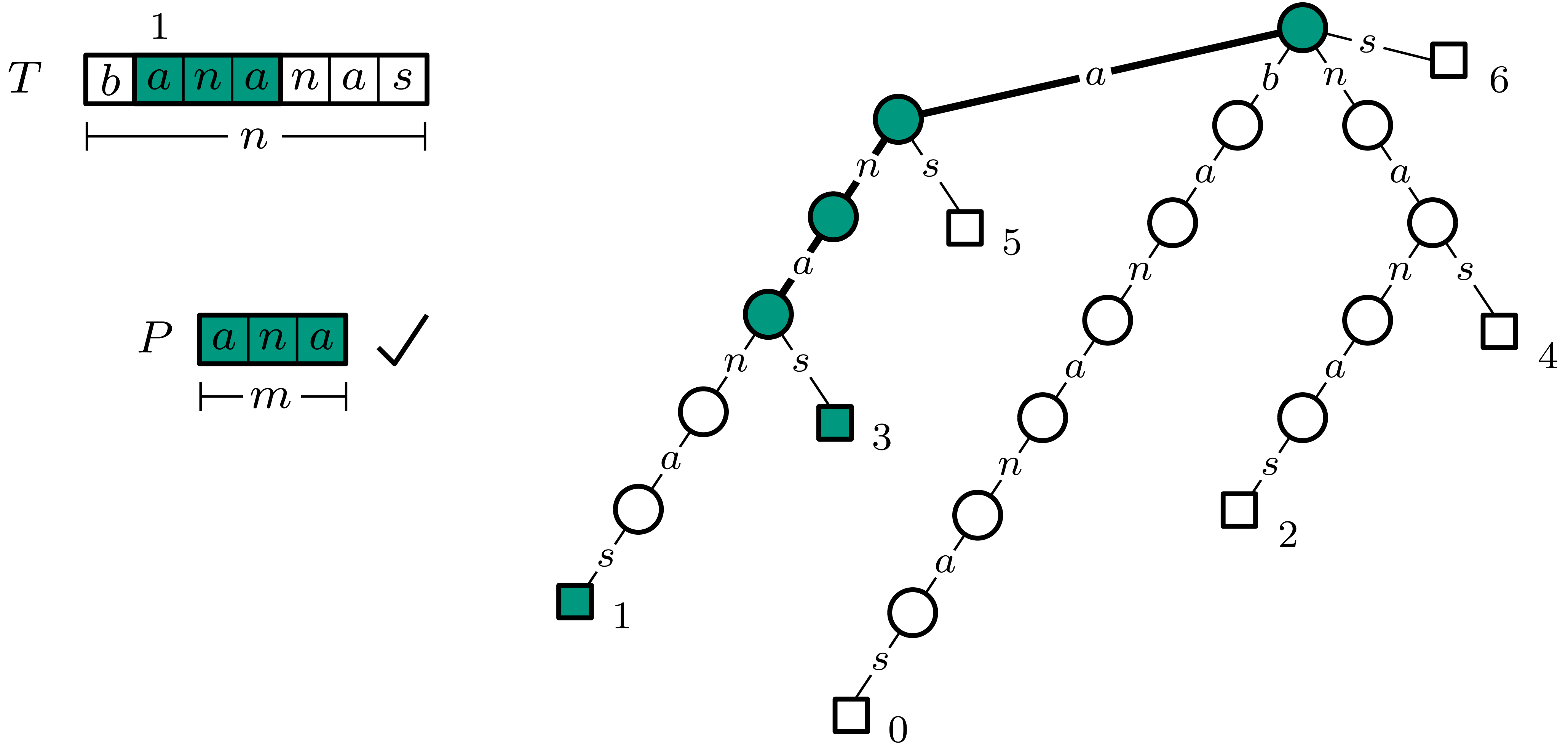


*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in an atomic suffix tree

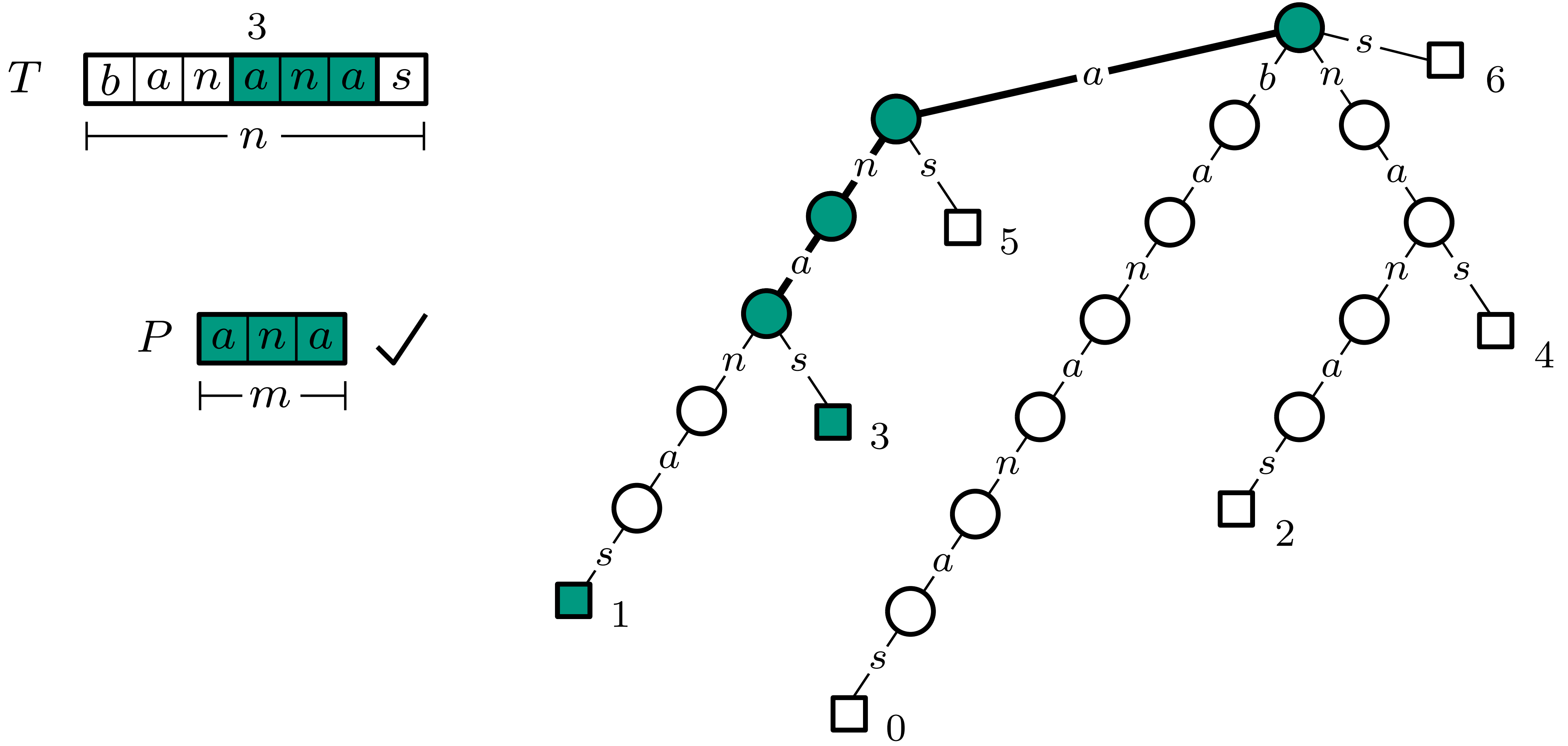


*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in an atomic suffix tree

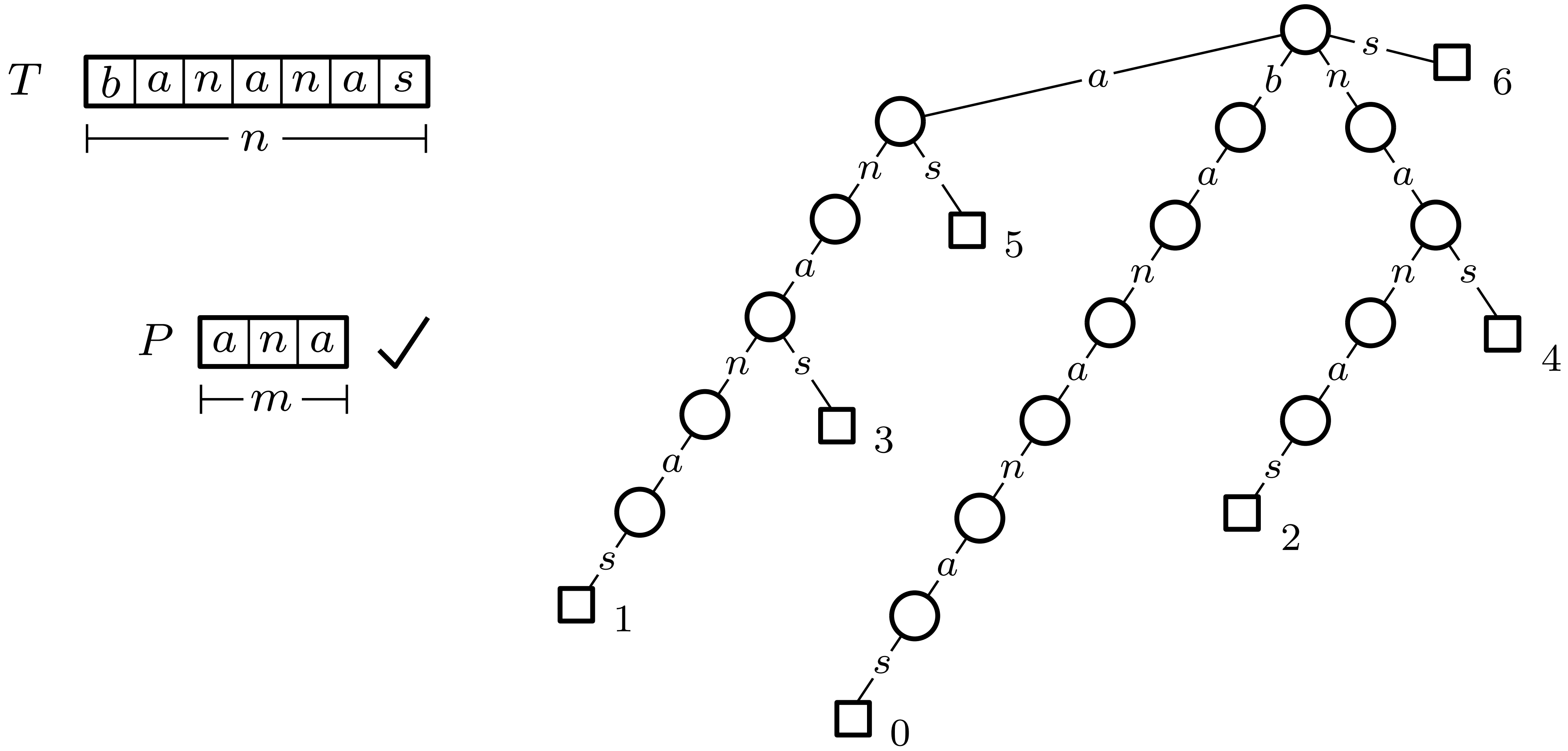


*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in an atomic suffix tree

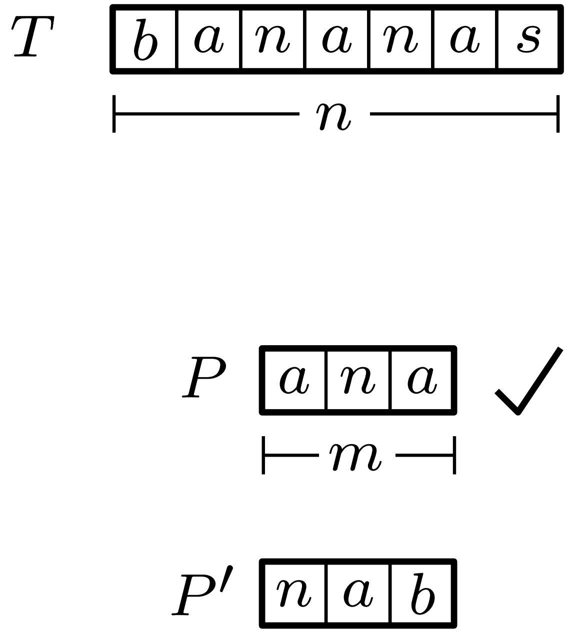


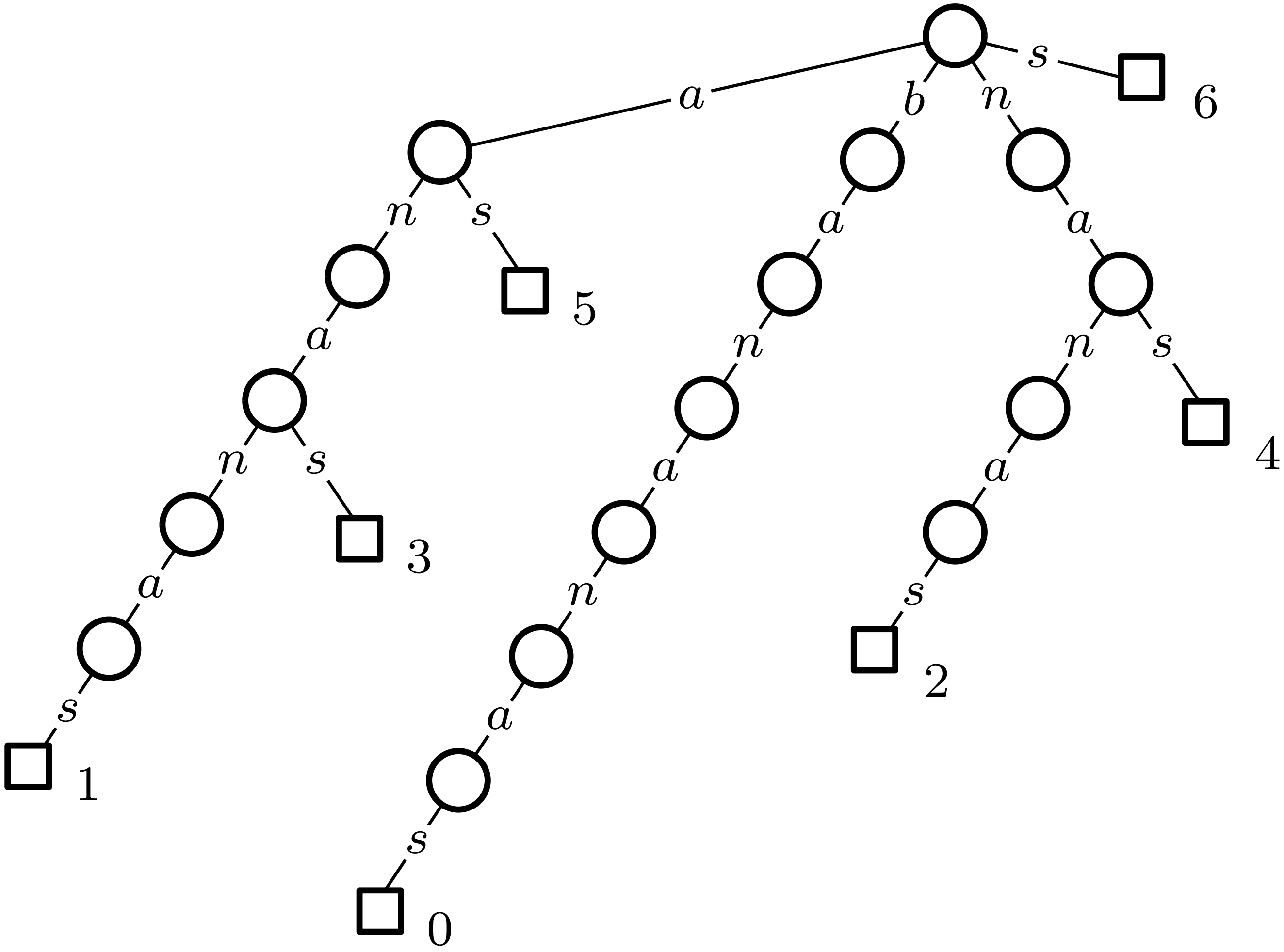
*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in an atomic suffix tree



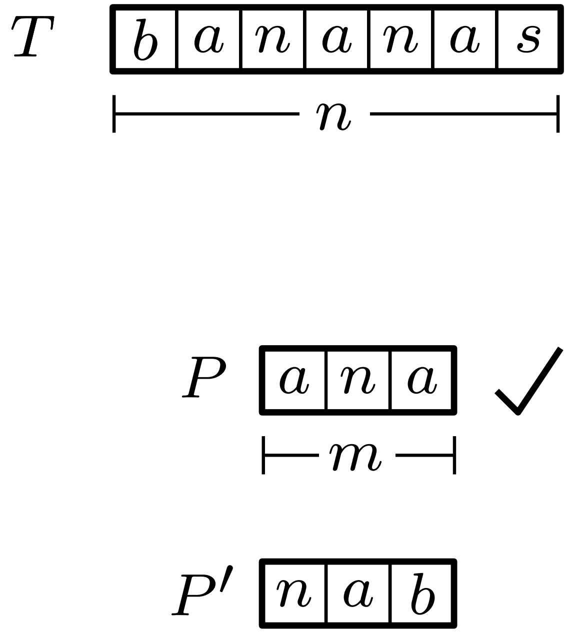


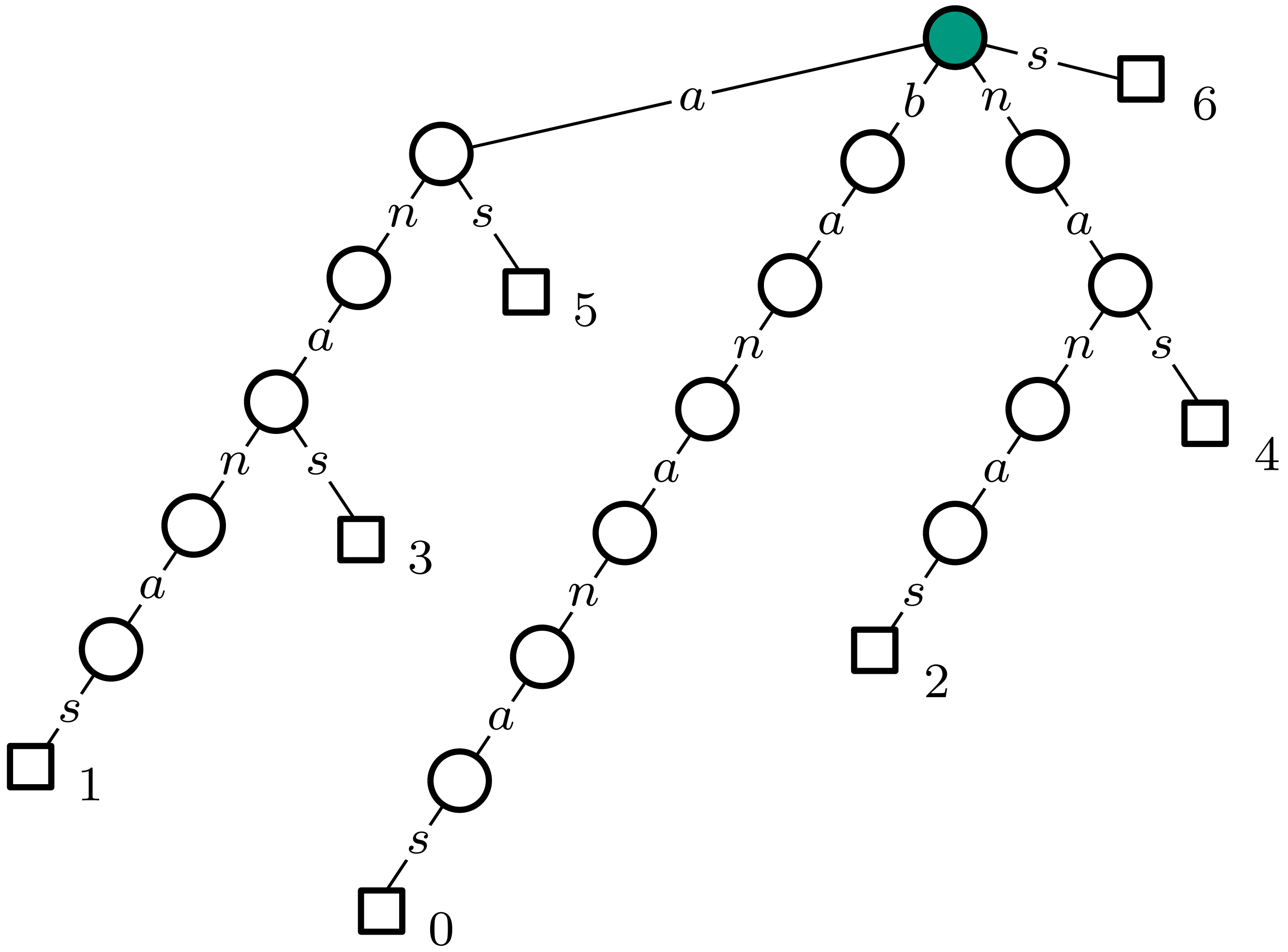
*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in an atomic suffix tree



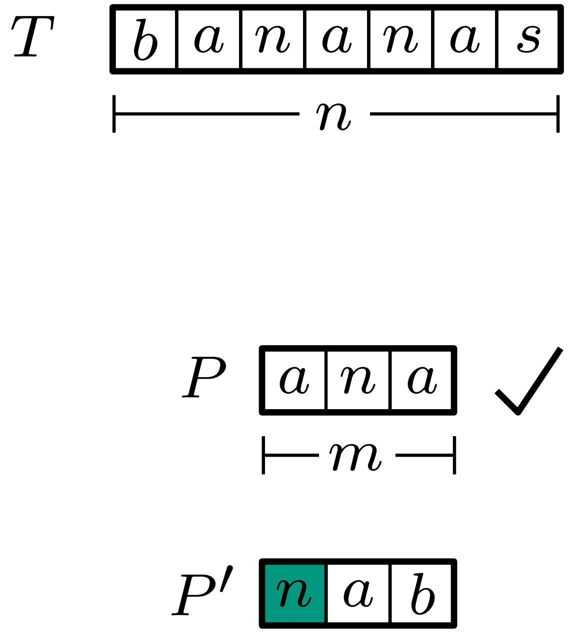


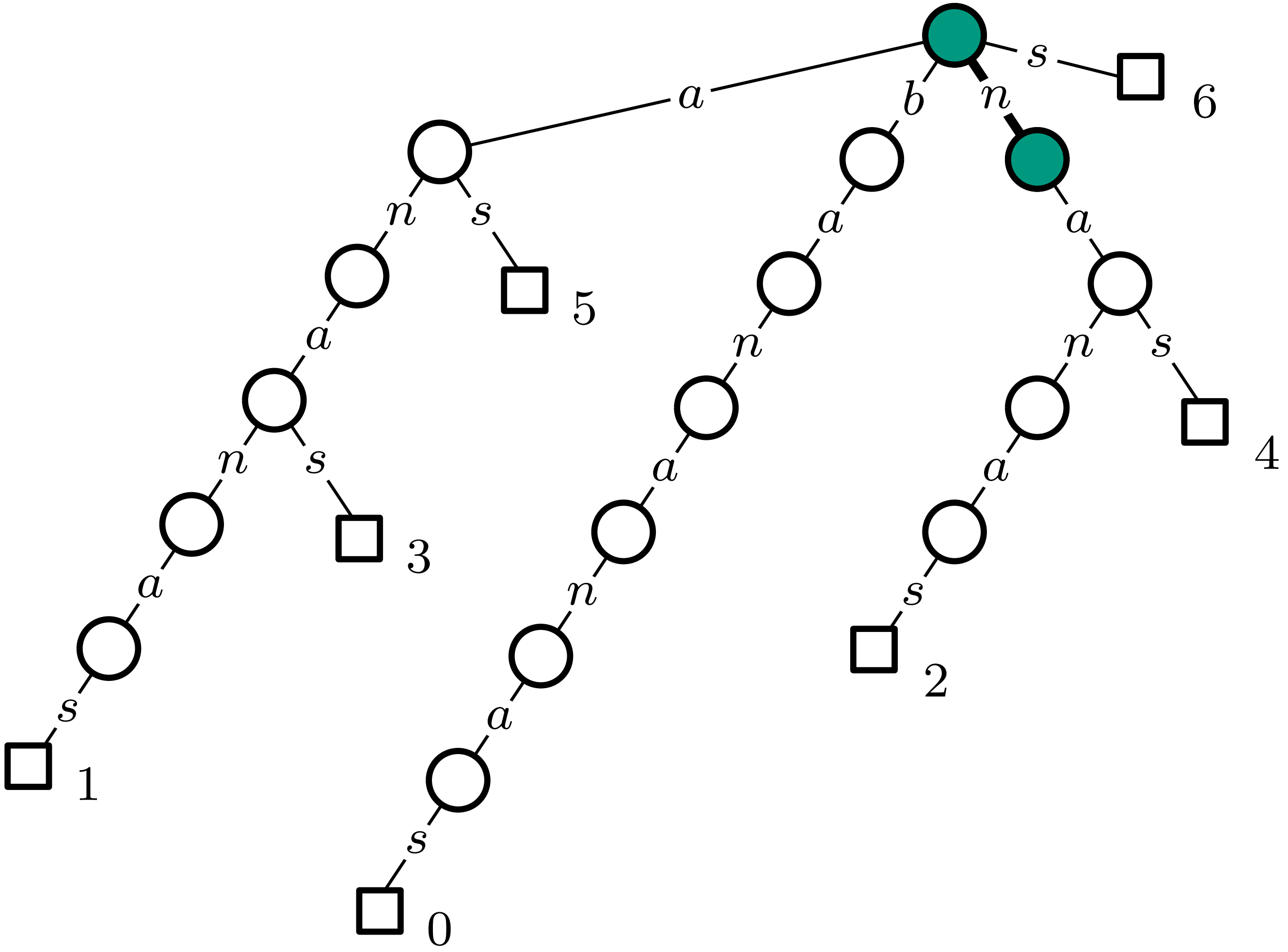
*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in an atomic suffix tree



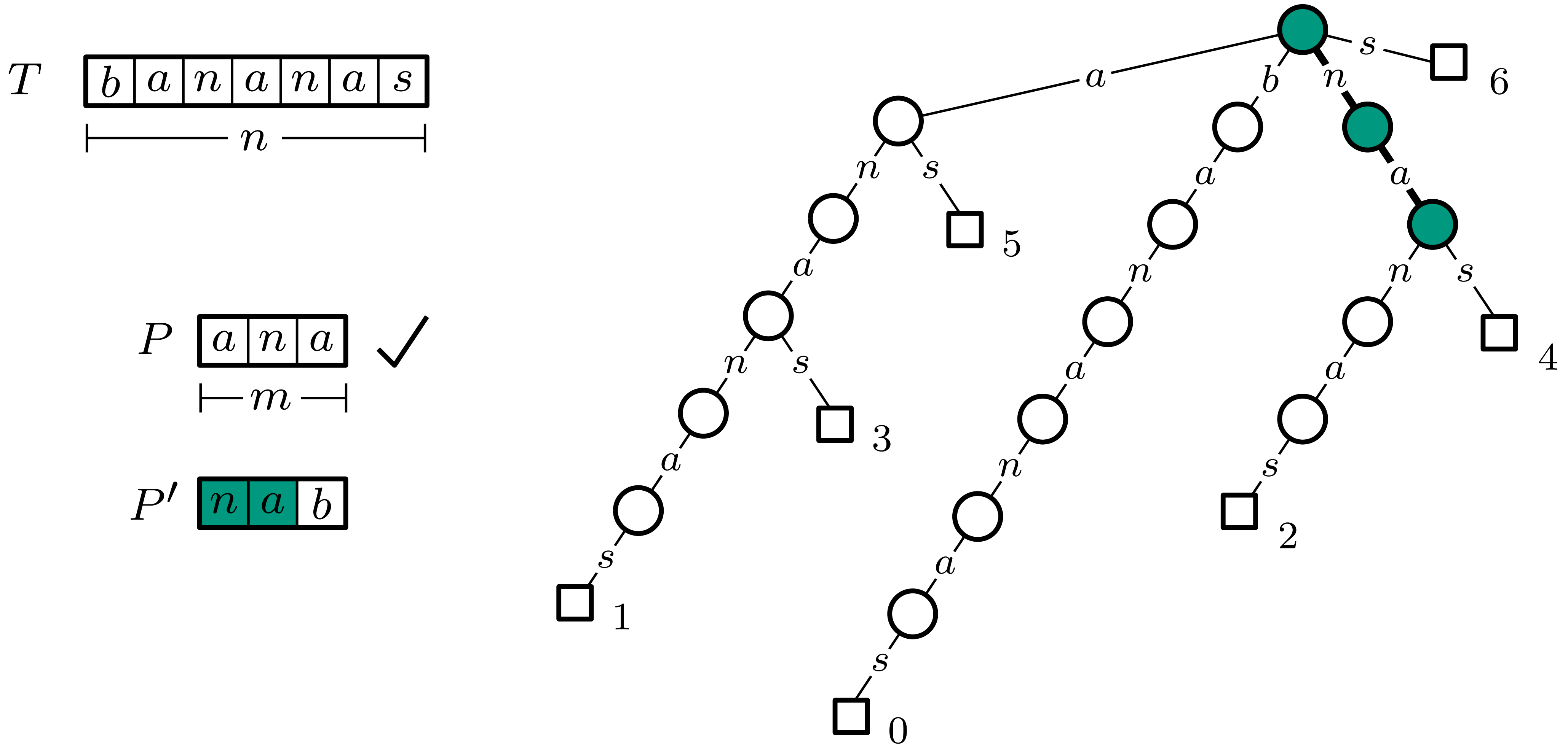


*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in an atomic suffix tree

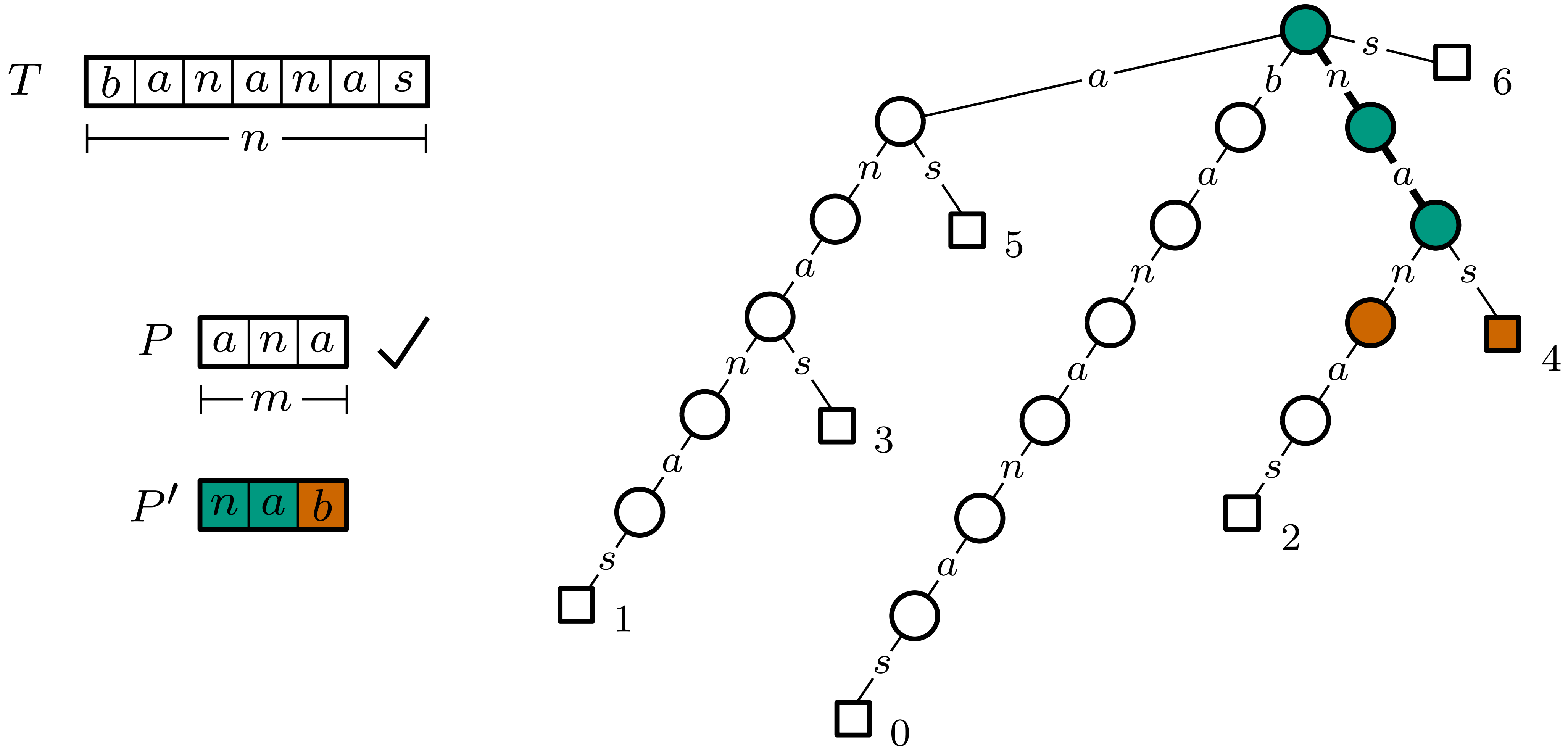


*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in an atomic suffix tree

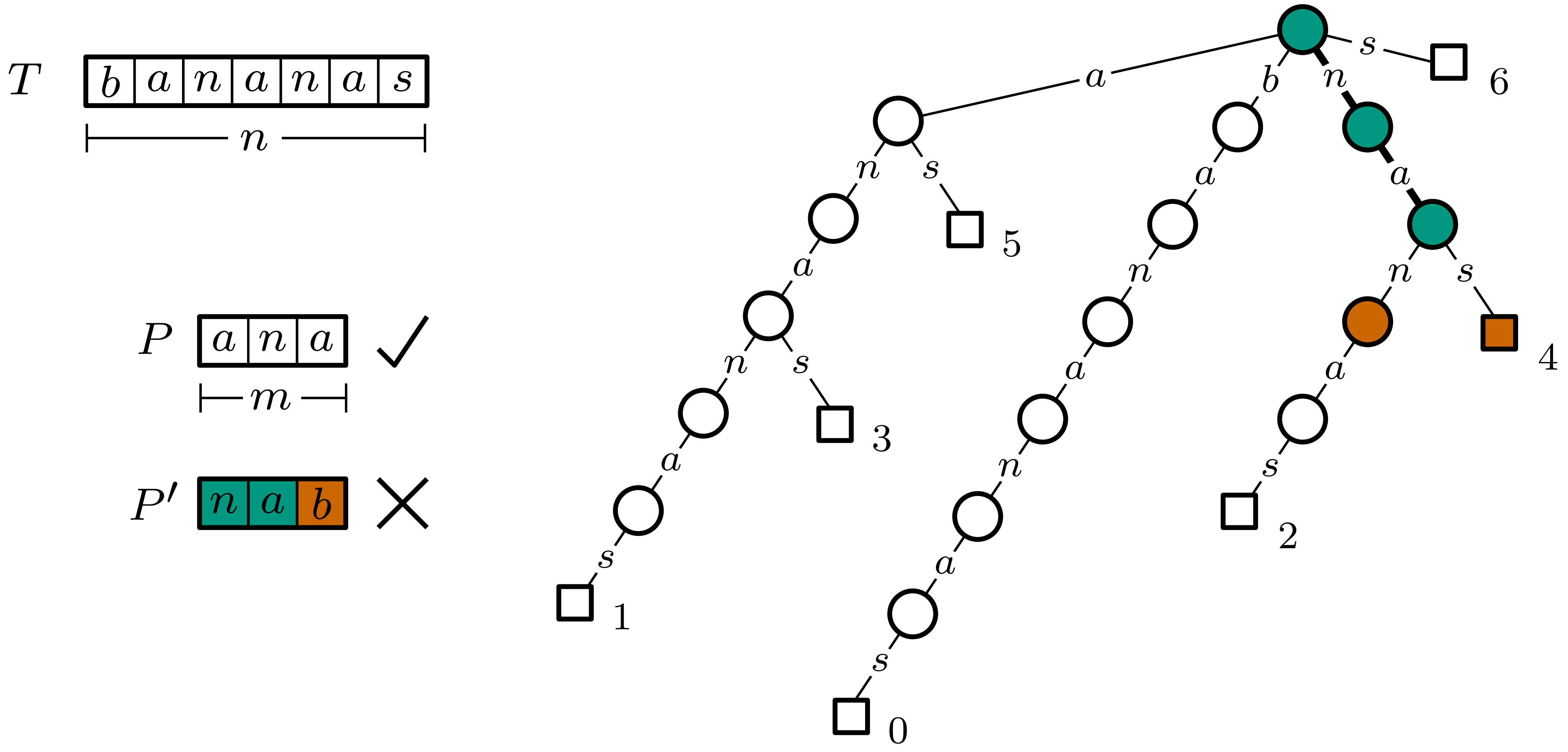


*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in an atomic suffix tree

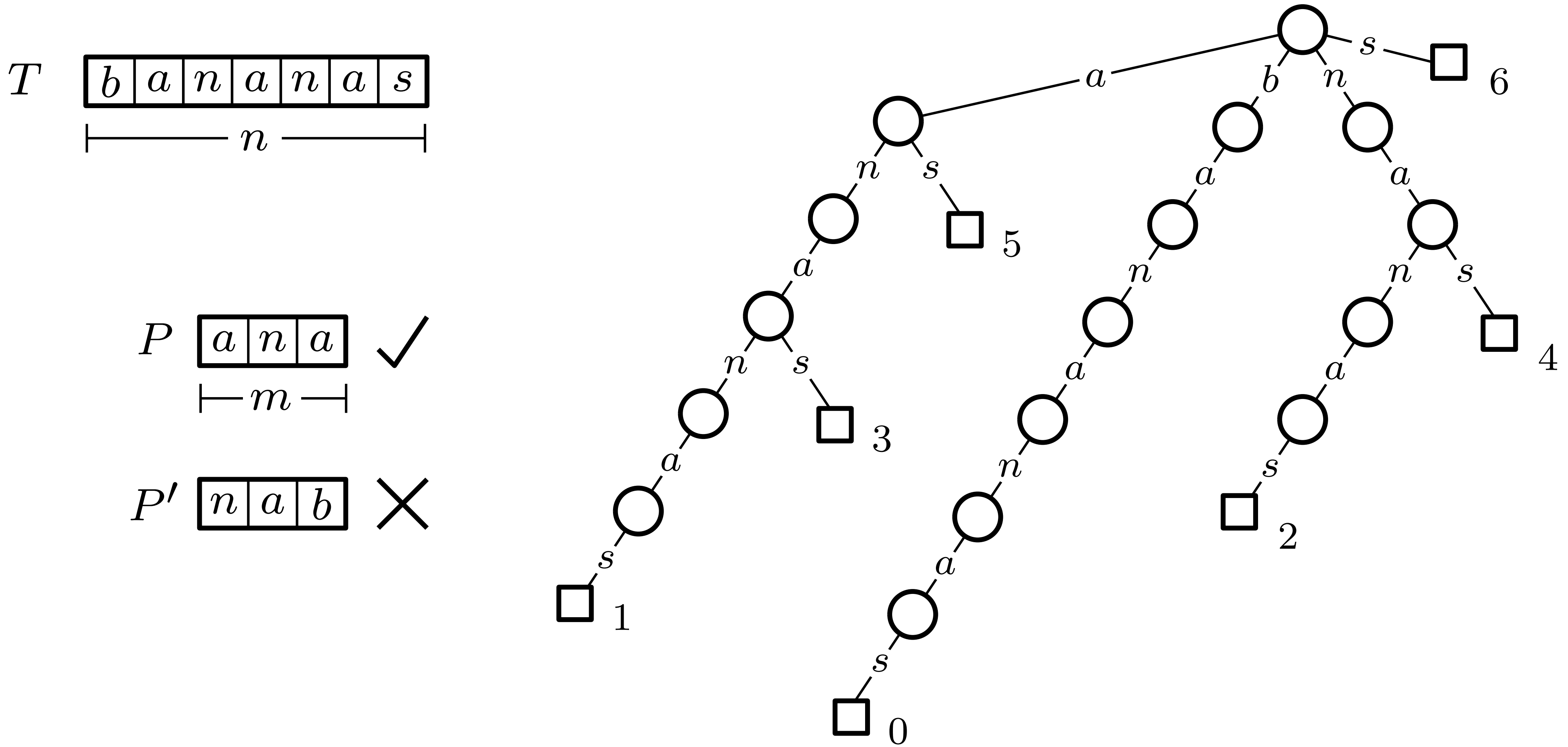


*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in an atomic suffix tree

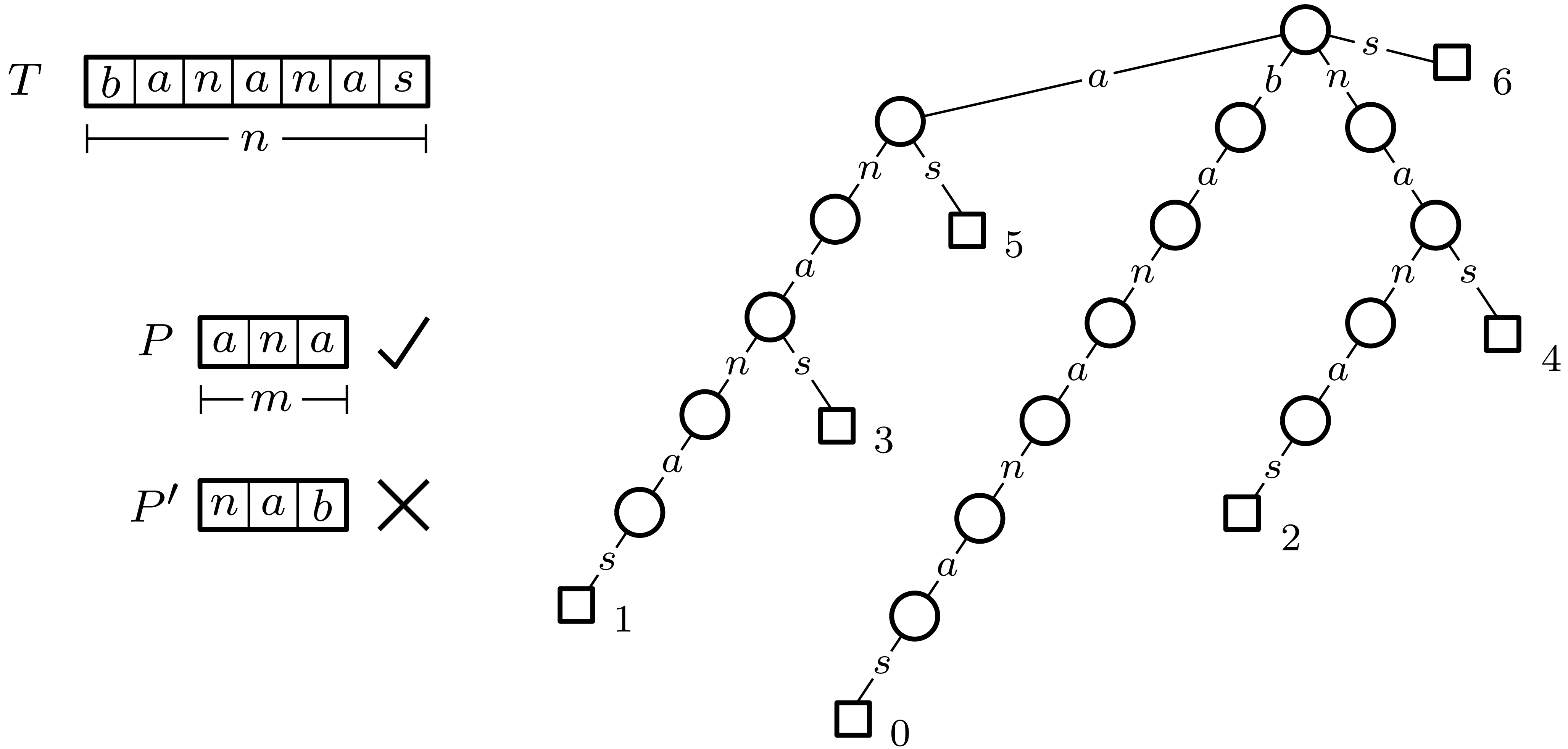


*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in an atomic suffix tree



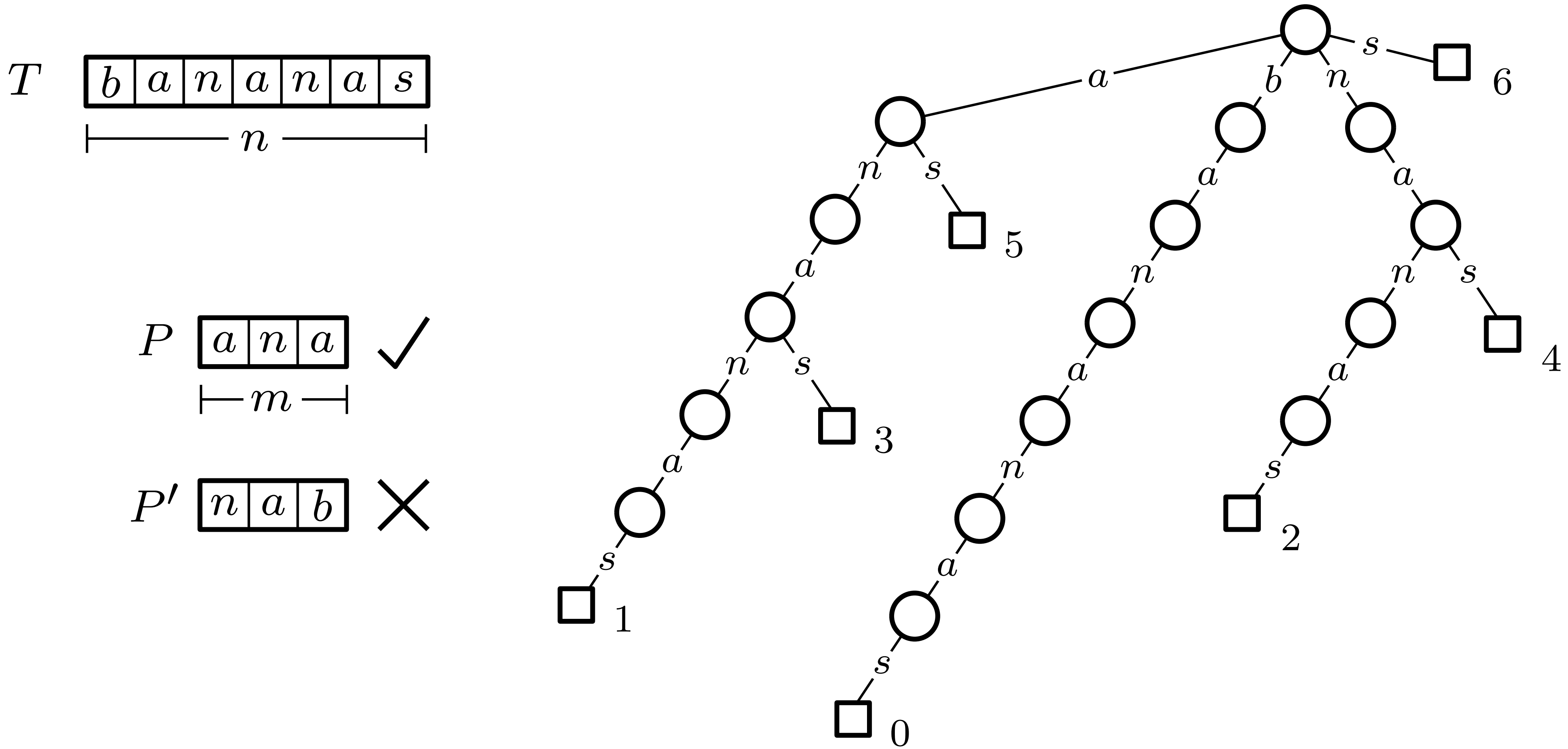
*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

We can decide whether $P$ matches *somewhere* in $O(m)$ time

# Searching in an atomic suffix tree



*How do you find a pattern?*

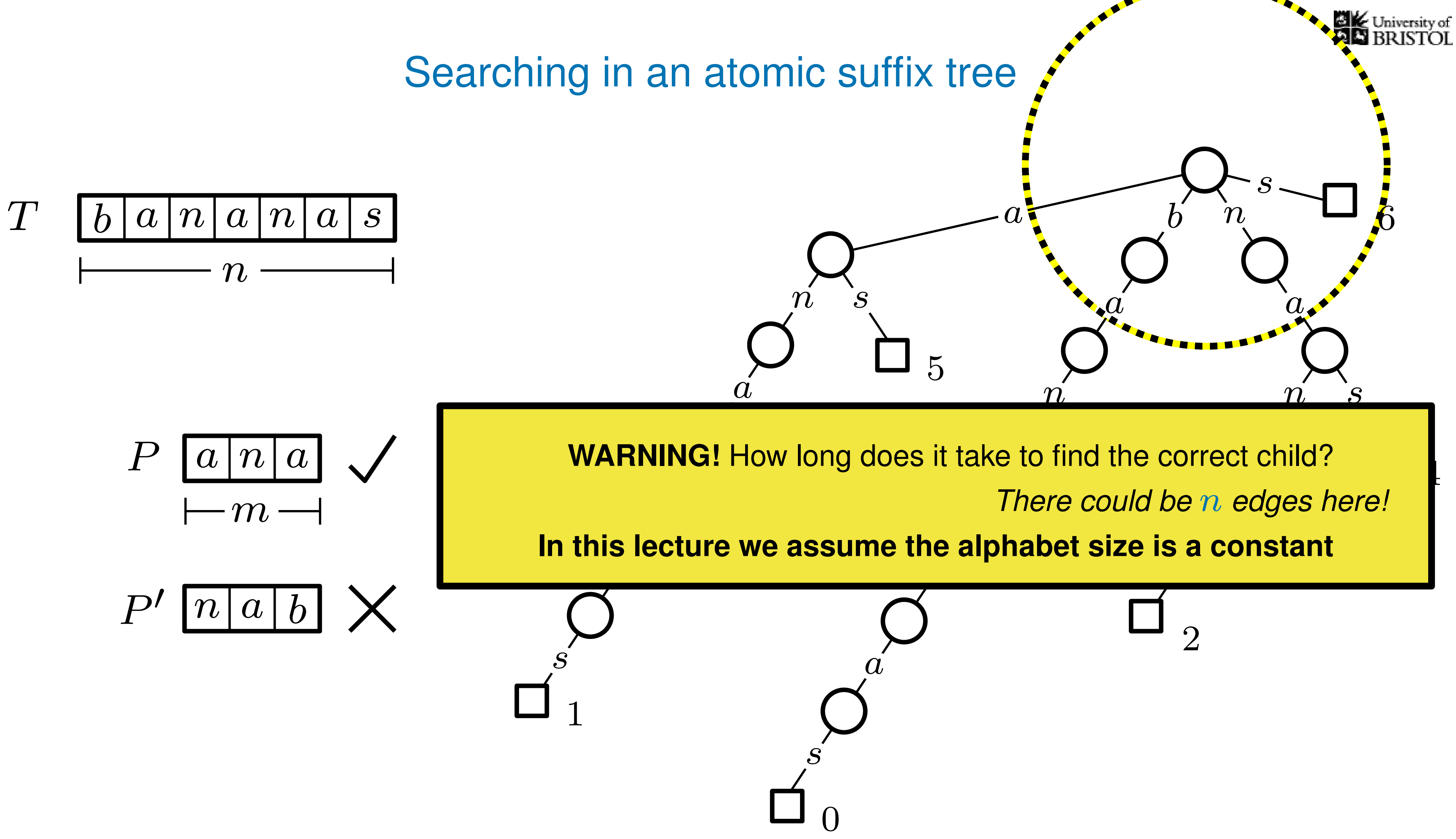
start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

We can decide whether $P$ matches *somewhere* in $O(m)$ time

*(we'll worry about outputting the matches later)*

# Searching in an atomic suffix tree



**WARNING!** How long does it take to find the correct child?

*There could be $n$ edges here!*

**In this lecture we assume the alphabet size is a constant**

*How do you find a pattern?*

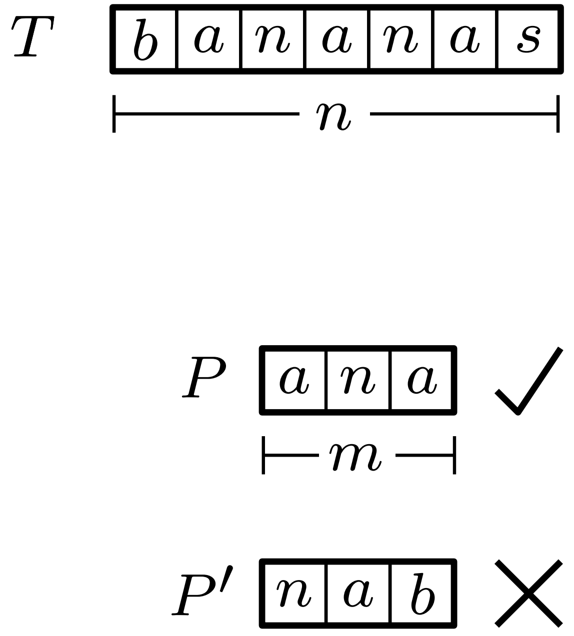
start at the root and walk down the tree

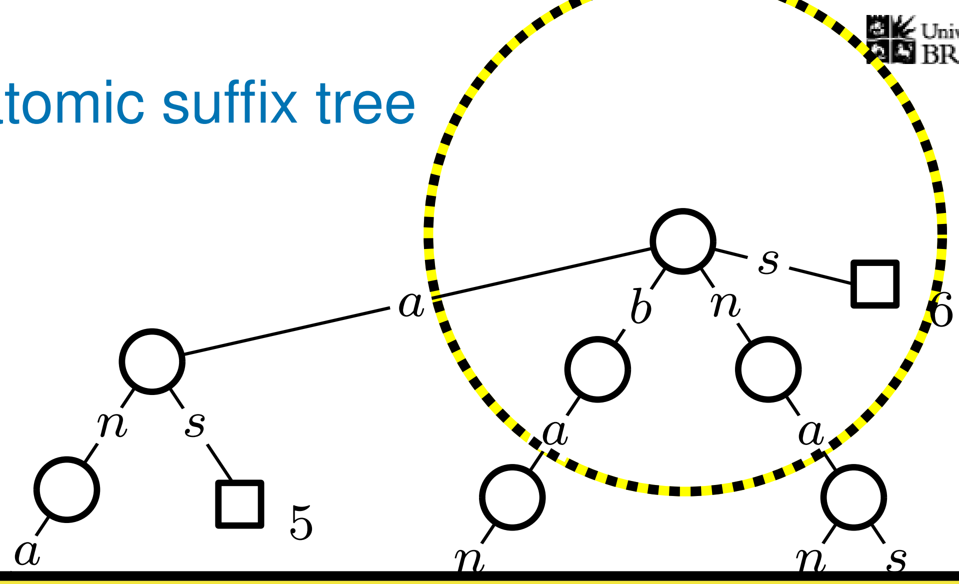
. . . matches occur at the leaves of the subtree

We can decide whether $P$ matches *somewhere* in $O(m)$ time

*(we'll worry about outputting the matches later)*

# Searching in an atomic suffix tree





**WARNING!** How long does it take to find the correct child?

*There could be $n$ edges here!*

**In this lecture we assume the alphabet size is a constant**

This may be fine in some applications

(English text or DNA for example)

*We can remove the assumption via the magic of hashing*

*How do you find a pattern?*

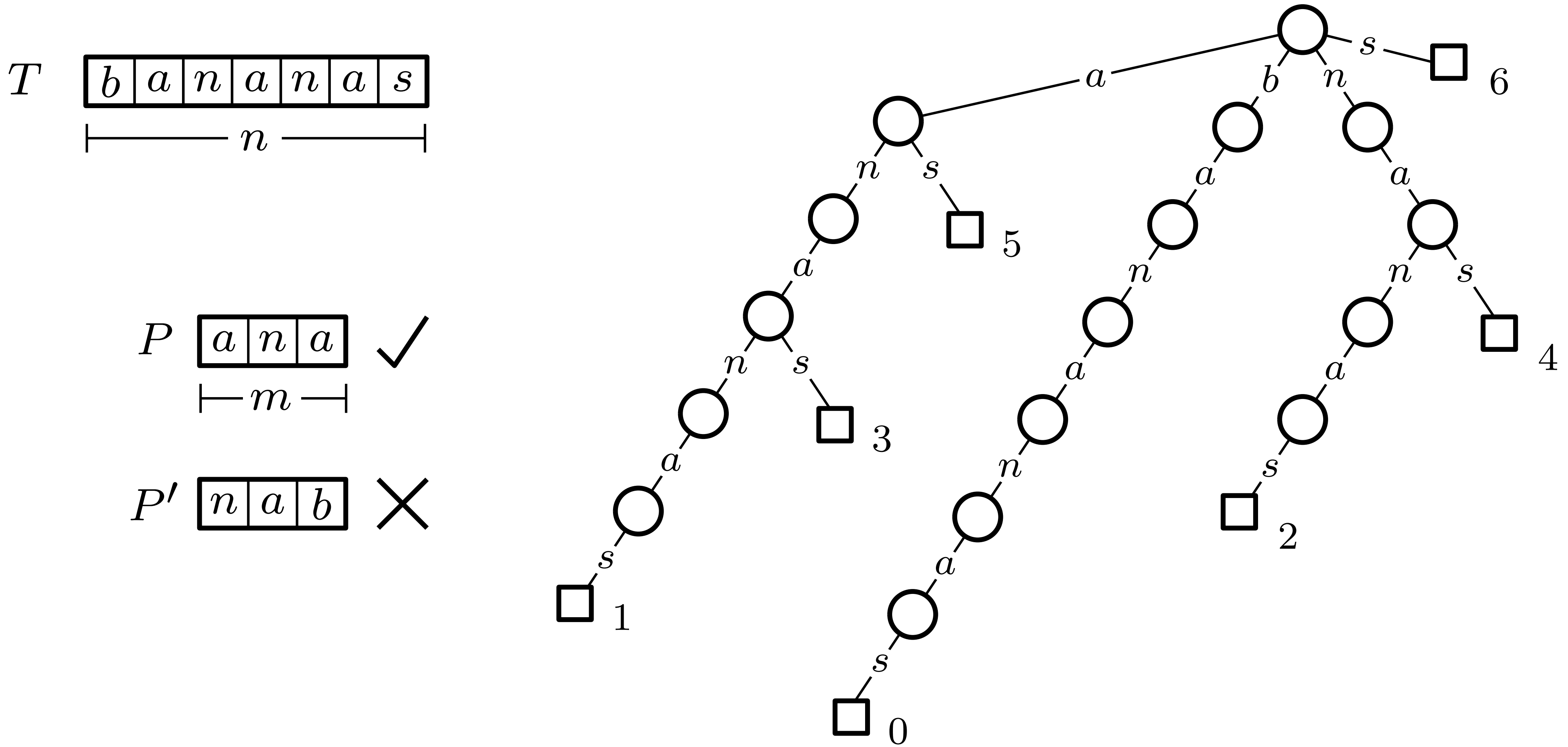
start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

We can decide whether $P$ matches *somewhere* in $O(m)$ time

*(we'll worry about outputting the matches later)*

# Searching in an atomic suffix tree



*How do you find a pattern?*

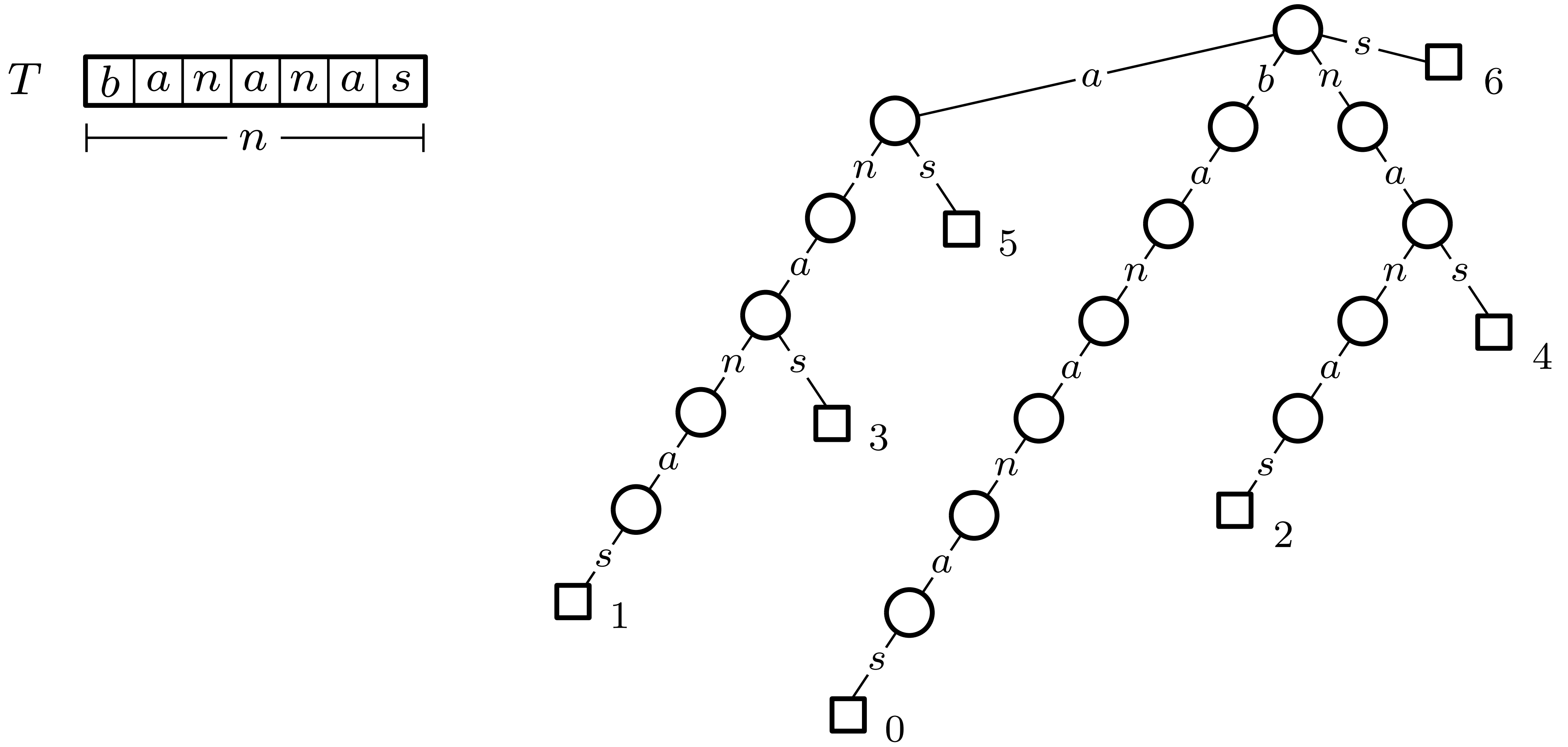
start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

We can decide whether $P$ matches *somewhere* in $O(m)$ time

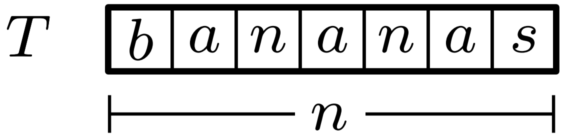
*(we'll worry about outputting the matches later)*

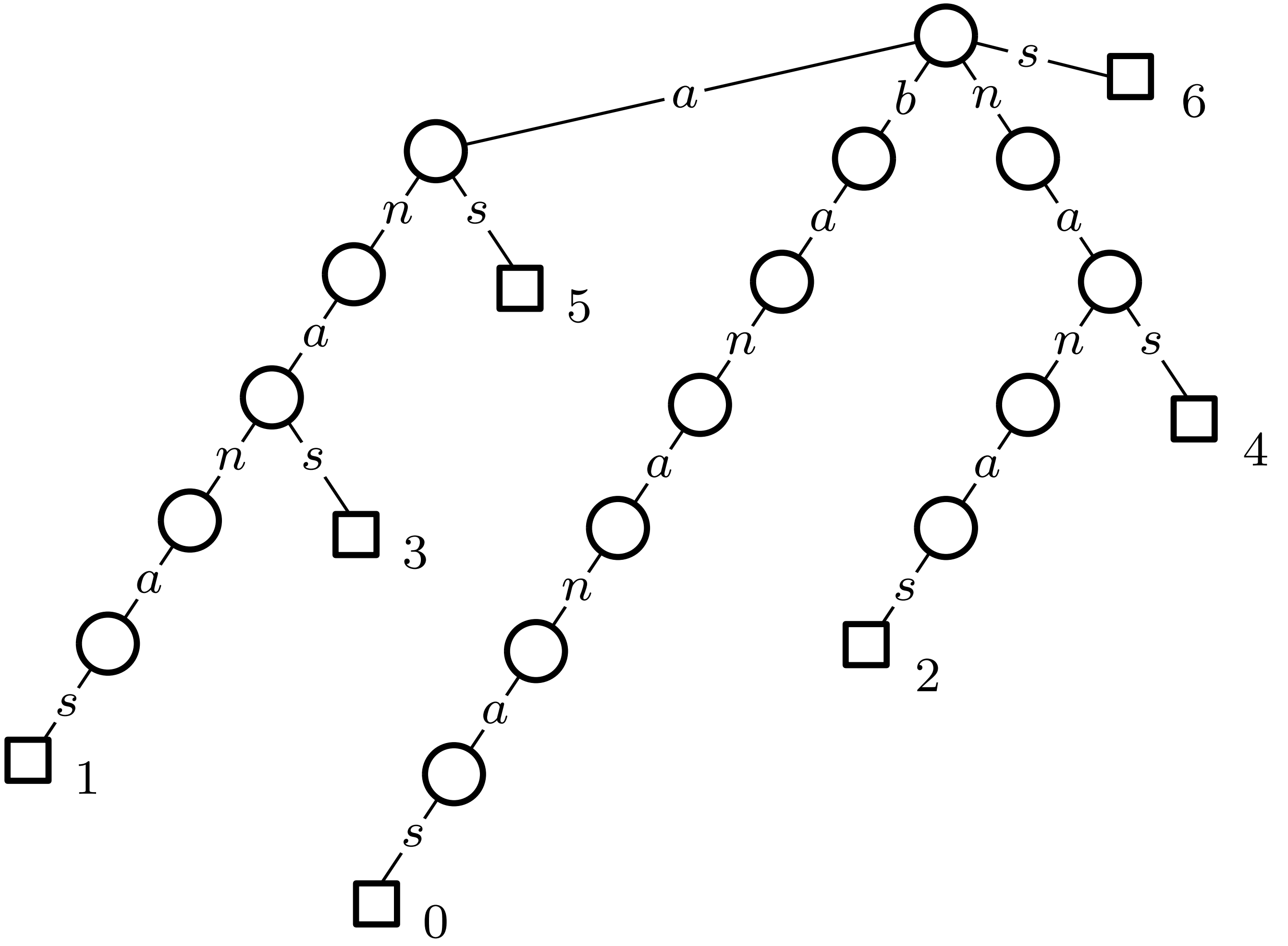
# how large is the atomic suffix tree?



*There are at most $n$ leaves*

# how large is the atomic suffix tree?

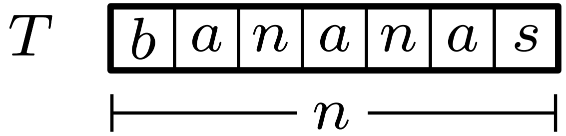


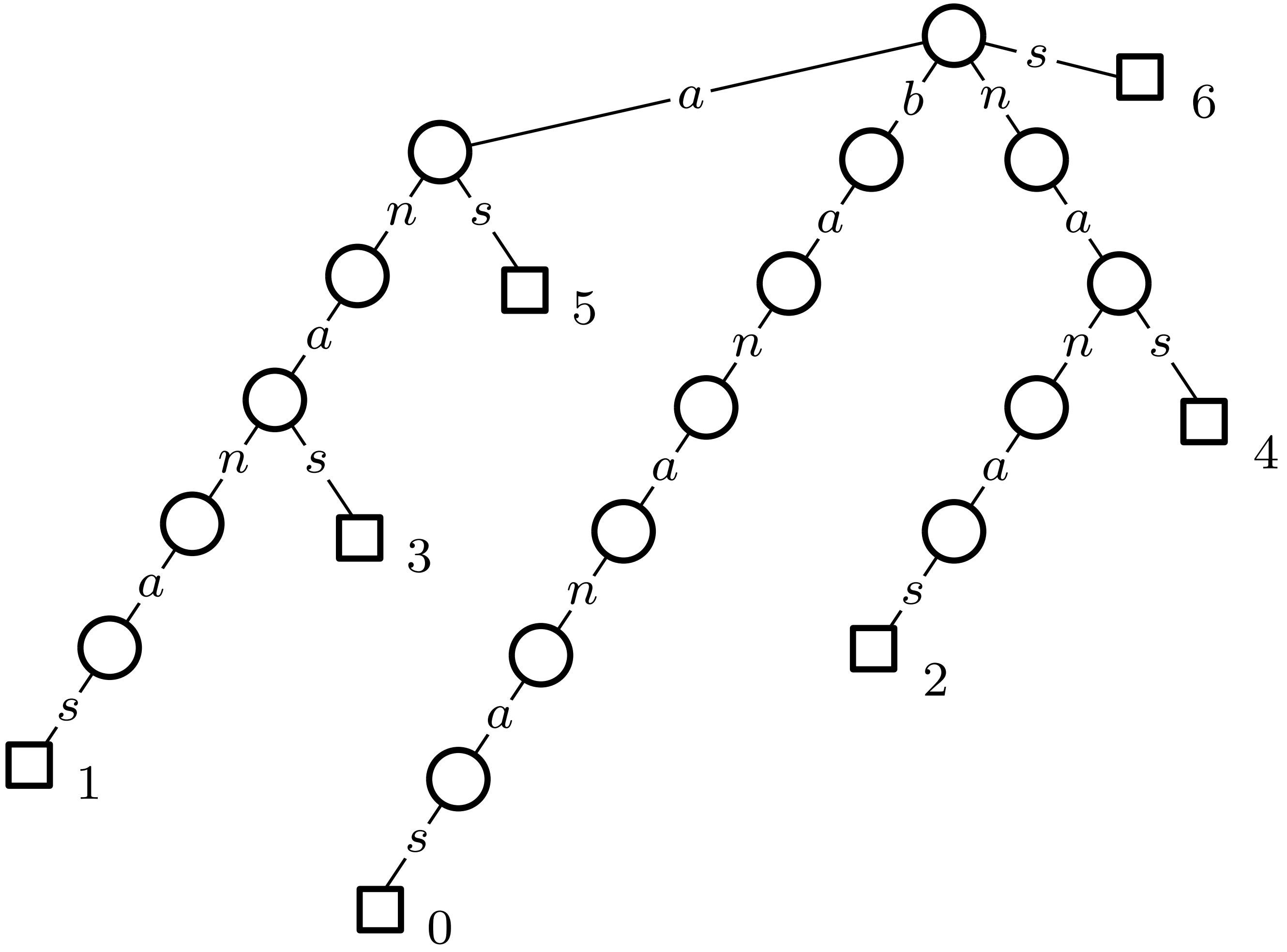


*There are at most $n$ leaves*

*that's good right?*

# how large is the atomic suffix tree?



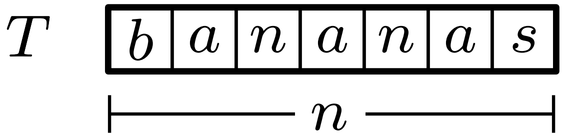


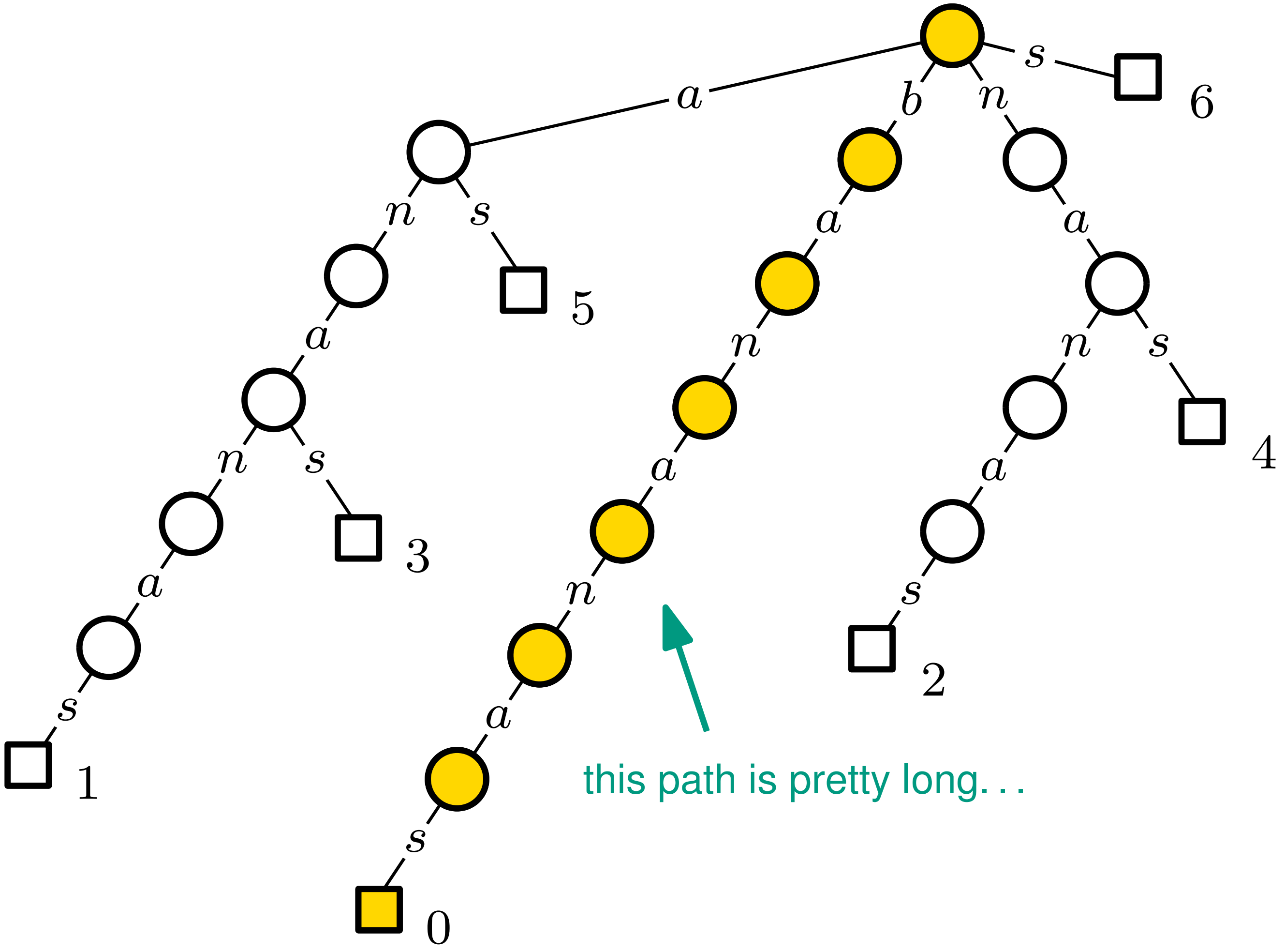
*There are at most* $n$ *leaves*

*that's good right?*

Unfortunately there can be *lots* of internal nodes

# how large is the atomic suffix tree?





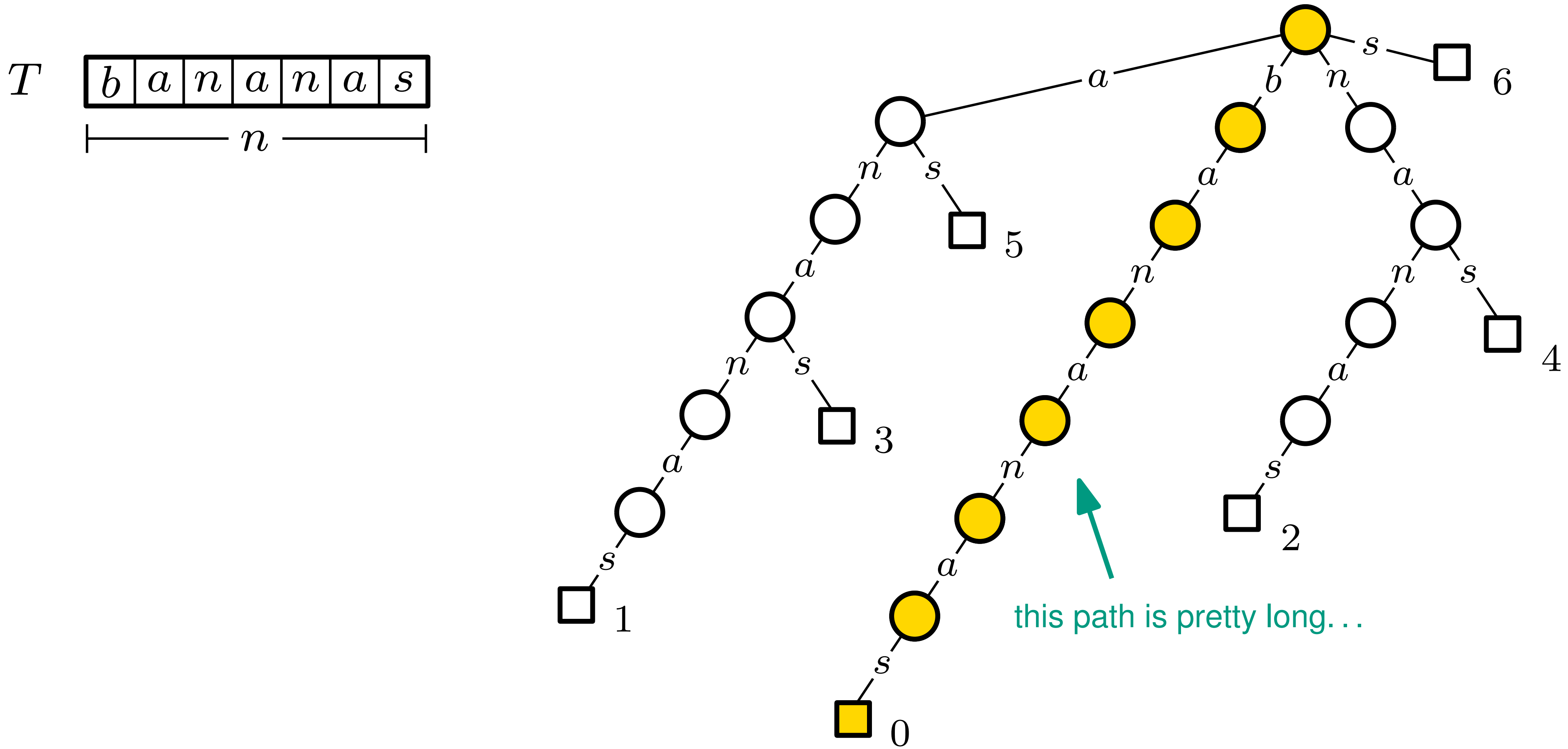
this path is pretty long. . .

*There are at most* $n$ *leaves*

*that's good right?*

Unfortunately there can be *lots* of internal nodes

# how large is the atomic suffix tree?



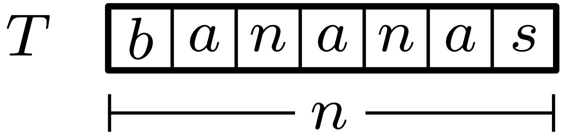
this path is pretty long. . .

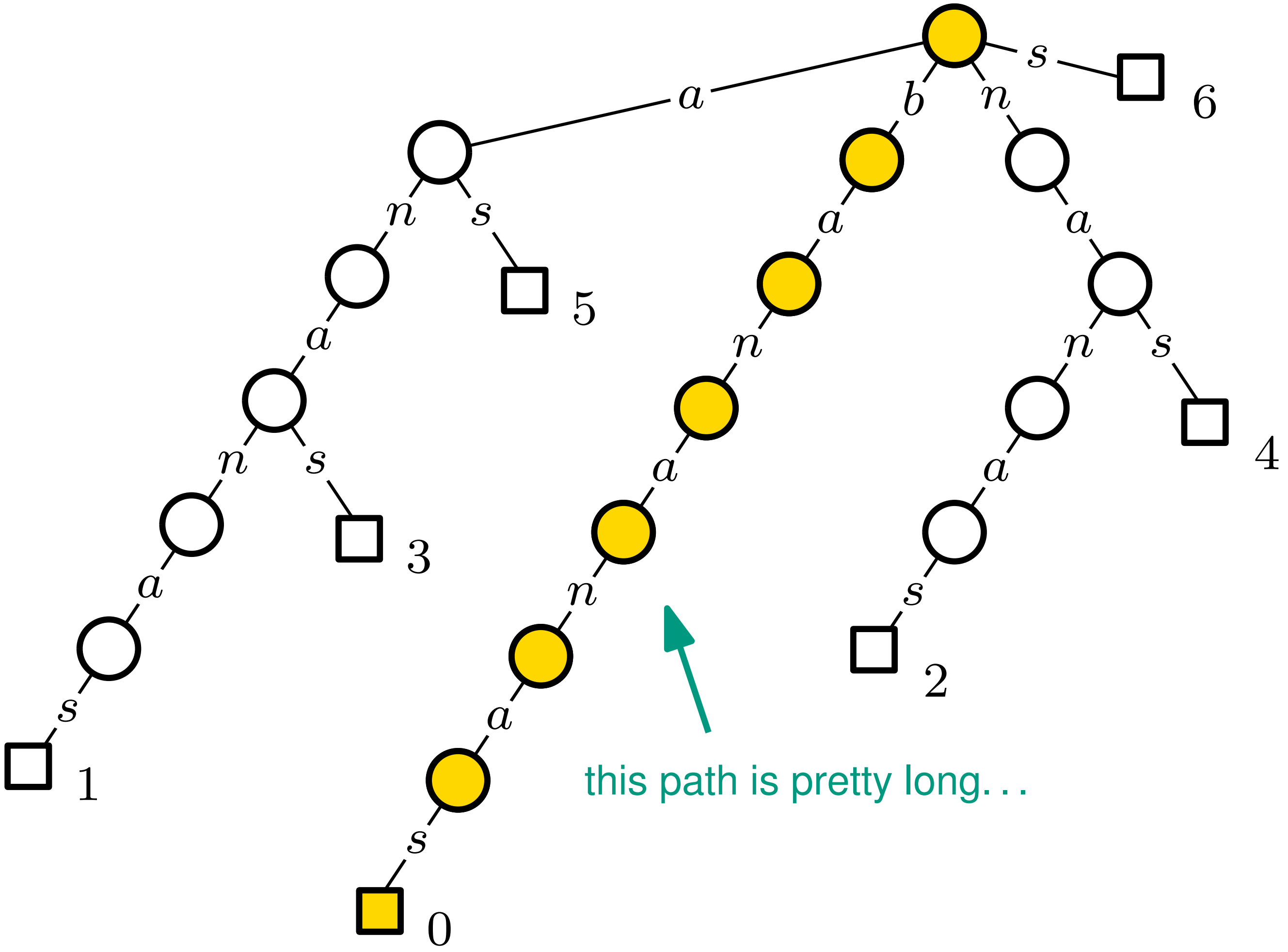
*There are at most $n$ leaves*

*that's good right?*

Unfortunately there can be *lots* of internal nodes

# how large is the atomic suffix tree?





this path is pretty long. . .

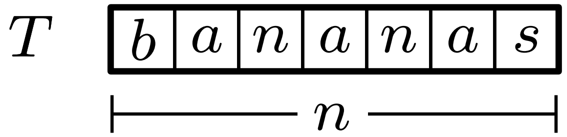
*There are at most $n$ leaves*

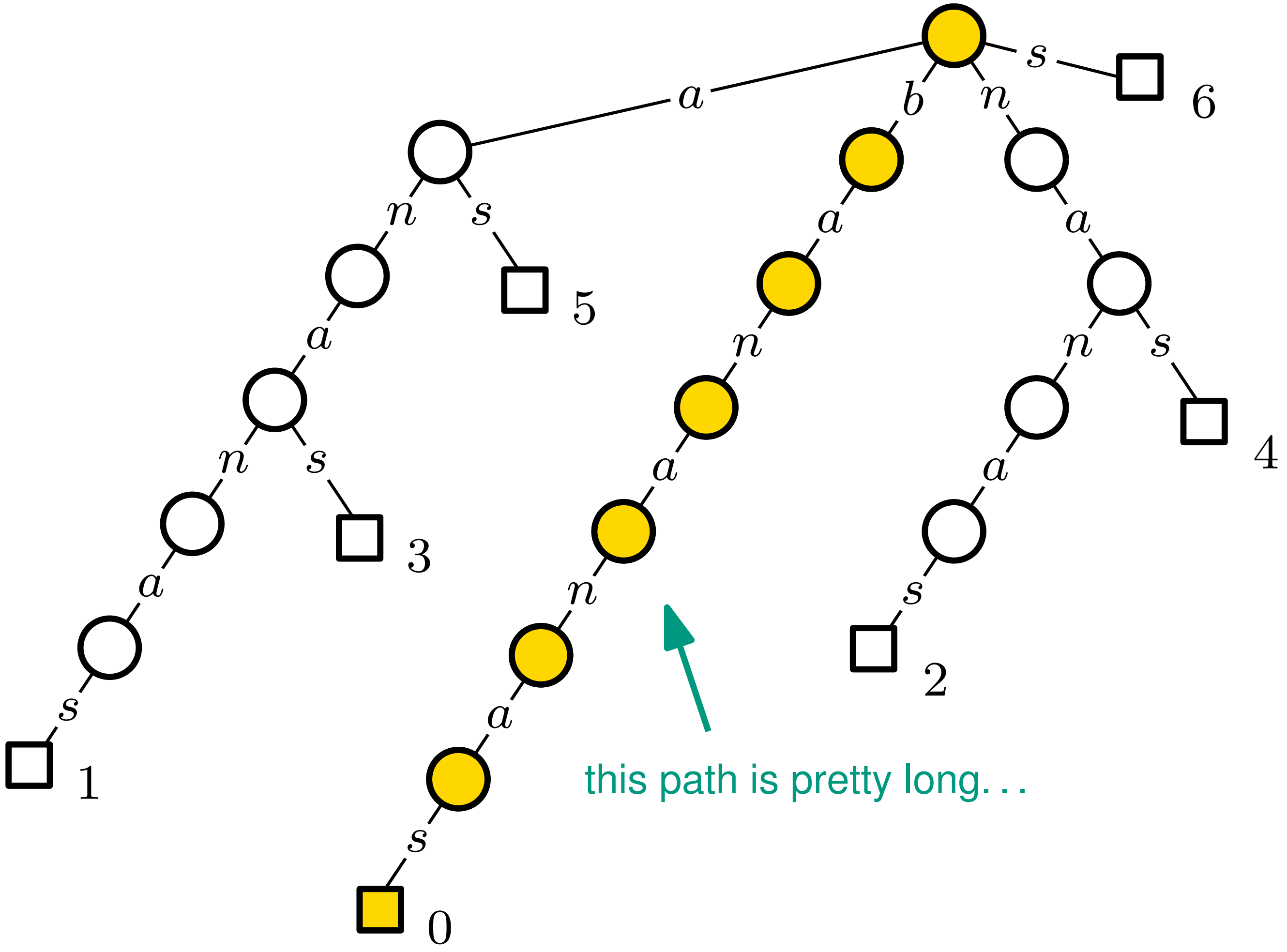
*that's good right?*

Unfortunately there can be *lots* of internal nodes

7 characters

# how large is the atomic suffix tree?





this path is pretty long. . .

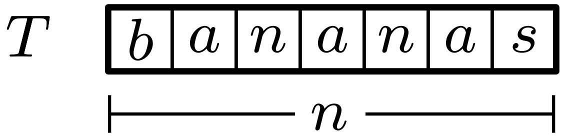
*There are at most* $n$ *leaves*

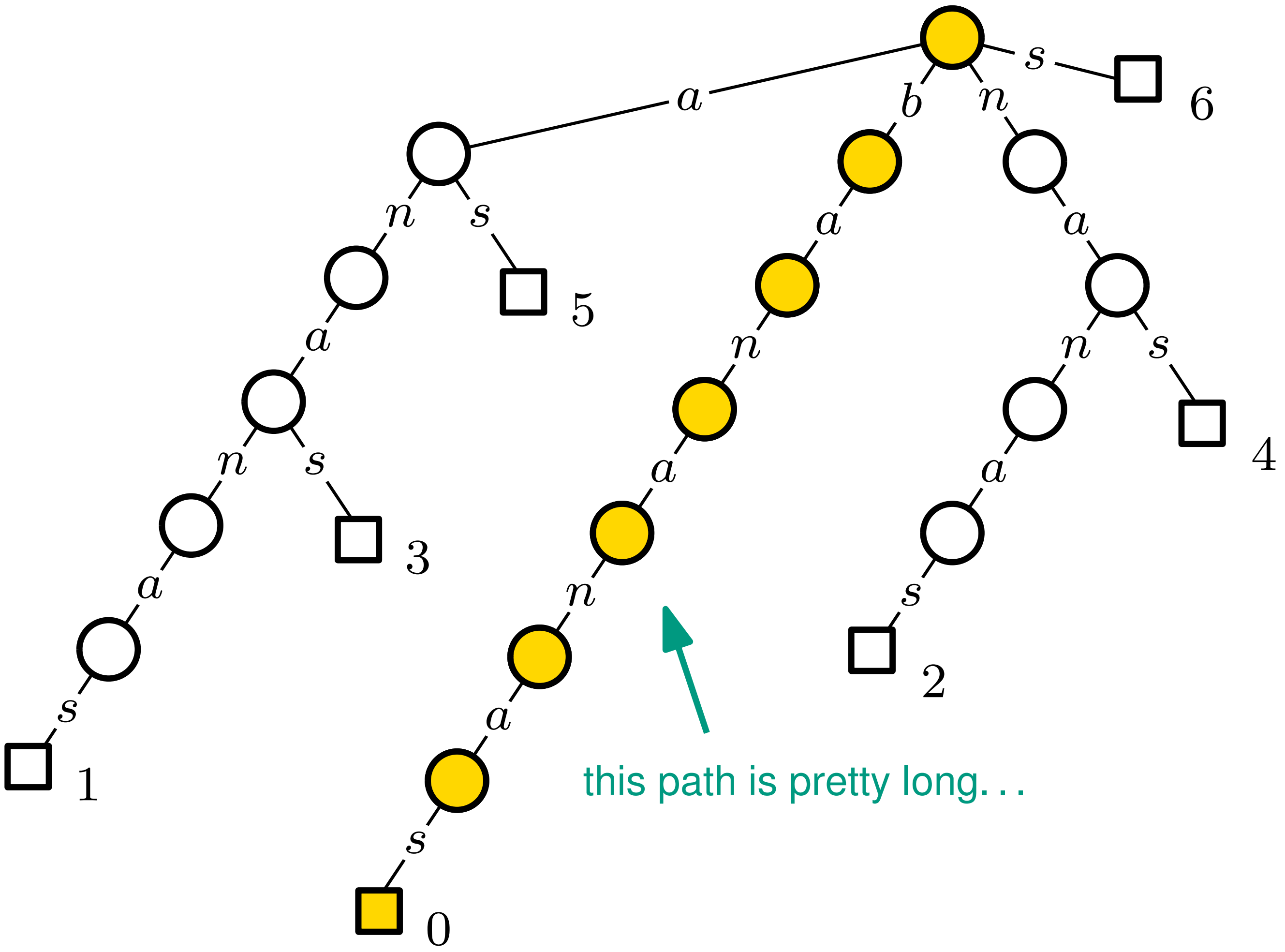
*that's good right?*

Unfortunately there can be *lots* of internal nodes

$7$ characters $23$ nodes

# how large is the atomic suffix tree?





this path is pretty long. . .

*There are at most* $n$ *leaves*

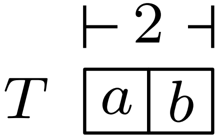
*that's good right?*

Unfortunately there can be *lots* of internal nodes

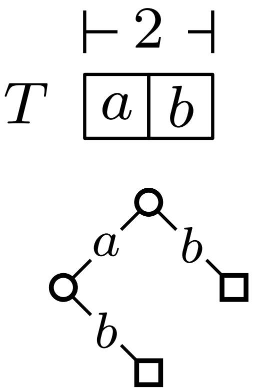
$7$ characters $23$ nodes that's not so bad, right?

# how large is the atomic suffix tree?

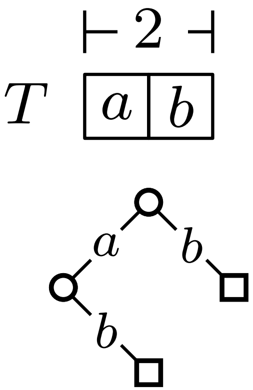
# how large is the atomic suffix tree?



# how large is the atomic suffix tree?

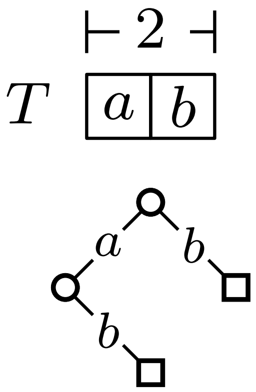


# how large is the atomic suffix tree?

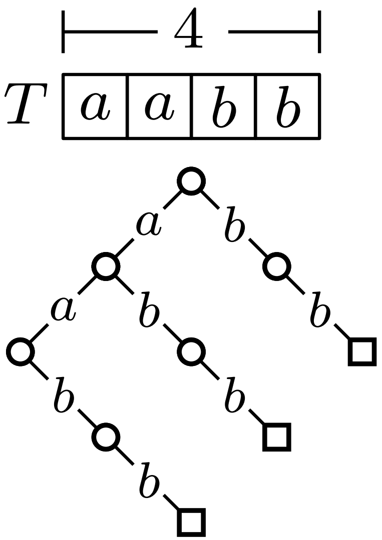


$4$ nodes

# how large is the atomic suffix tree?

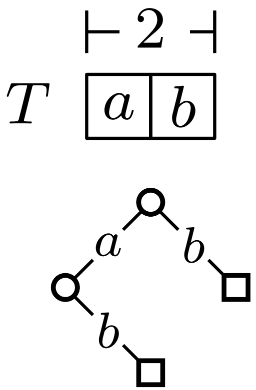


$4$ nodes

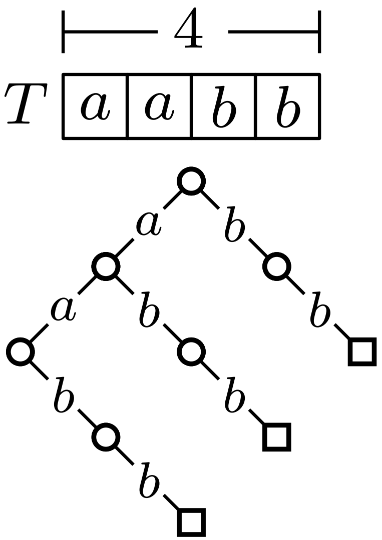


$9$ nodes

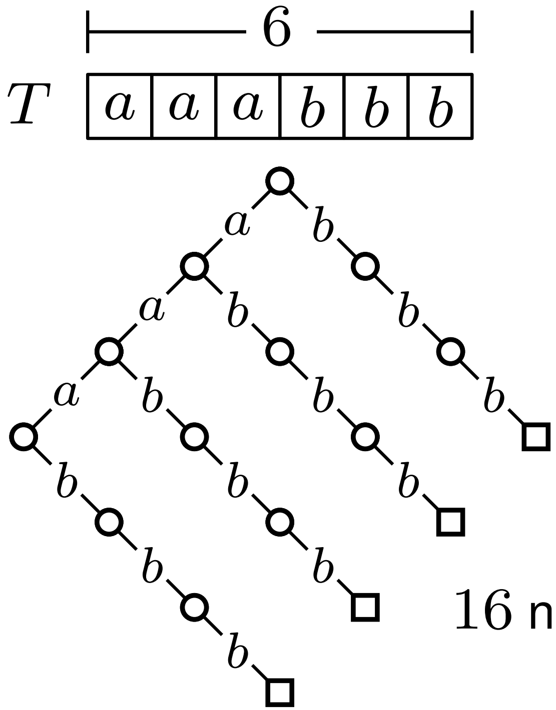
# how large is the atomic suffix tree?



$4$ nodes

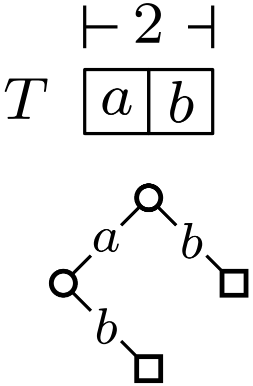


$9$ nodes

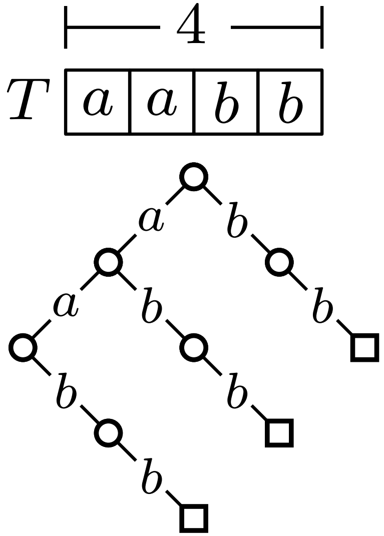


$16$ nodes

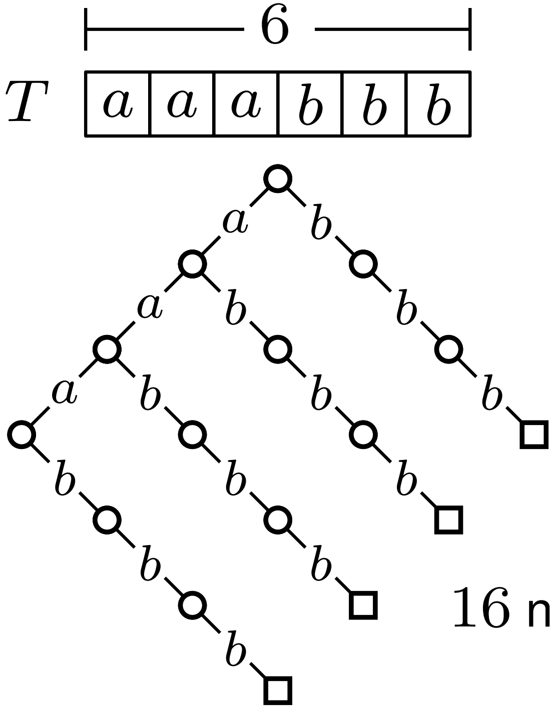
# how large is the atomic suffix tree?



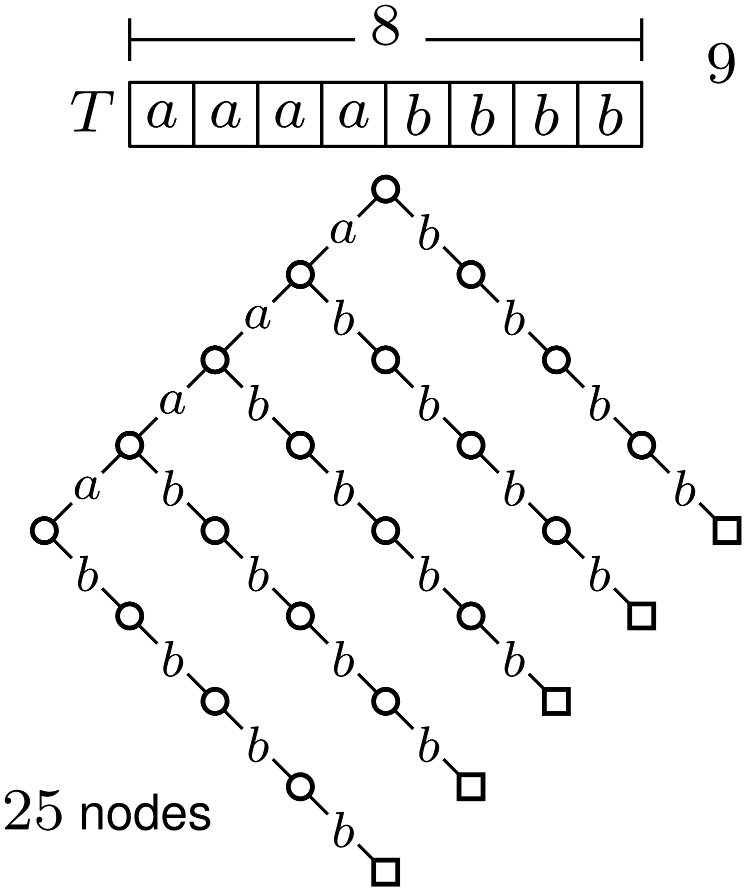
4 nodes



9 nodes

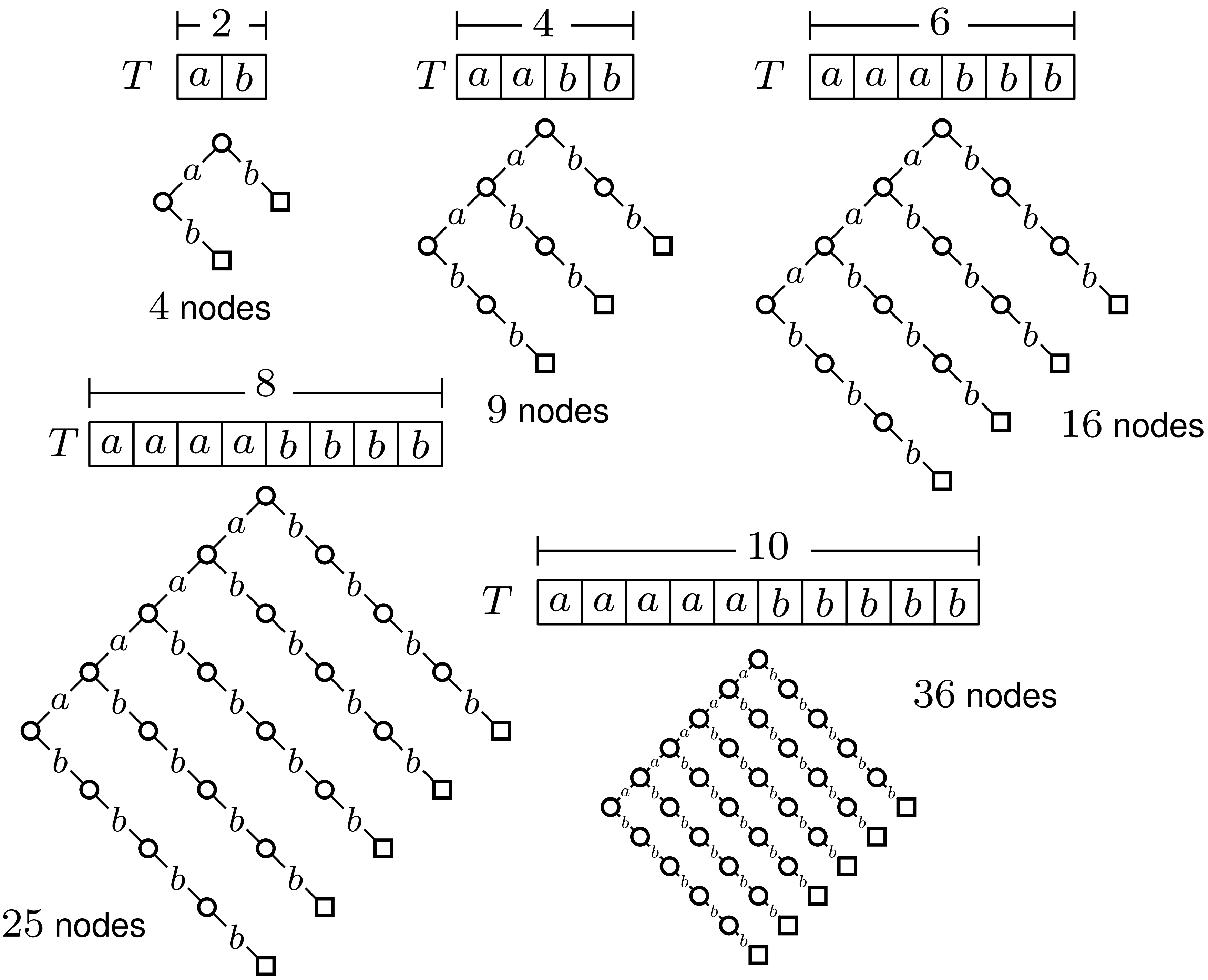


16 nodes



25 nodes

# how large is the atomic suffix tree?



4 nodes

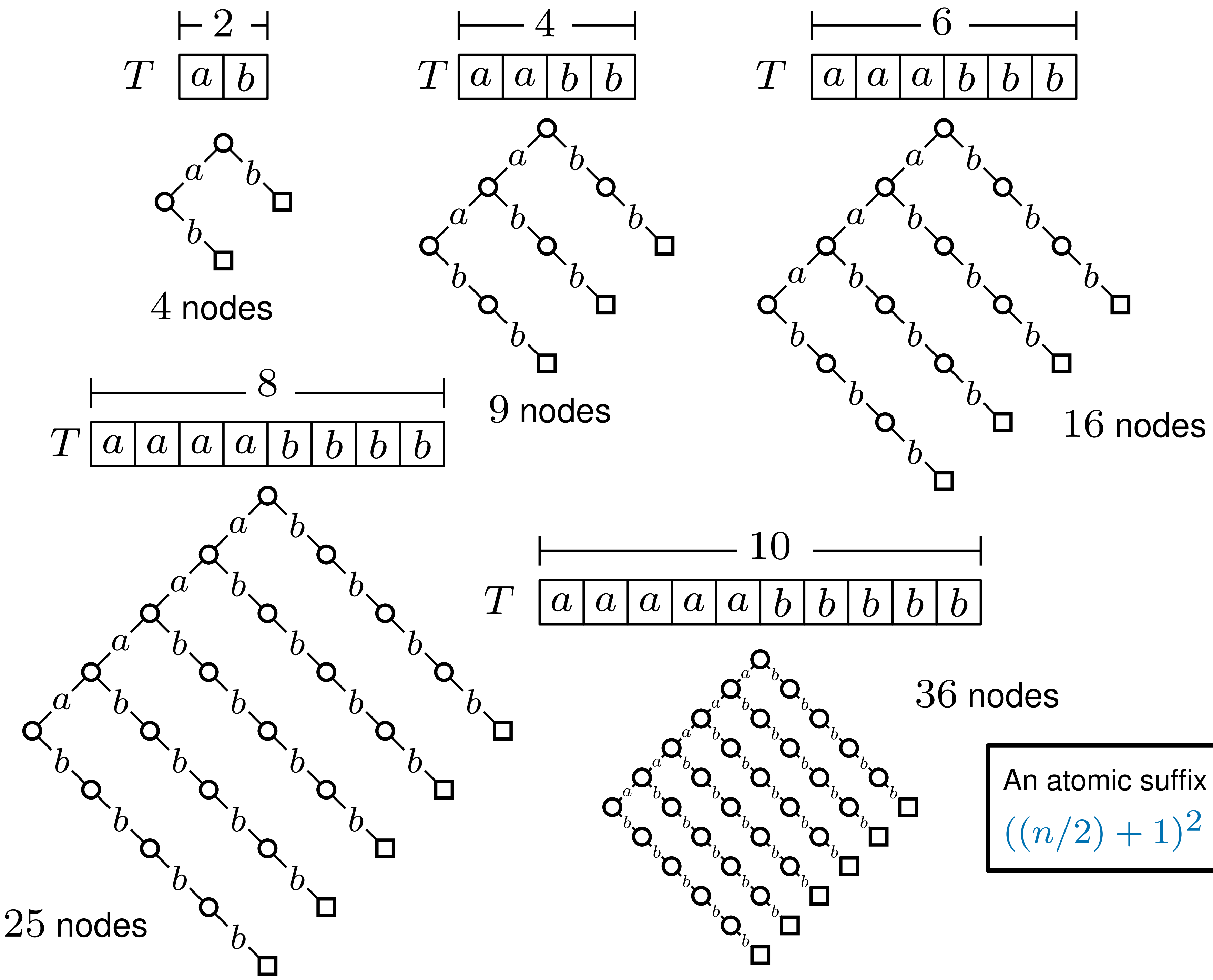
9 nodes

16 nodes

25 nodes

36 nodes

# how large is the atomic suffix tree?



4 nodes

9 nodes

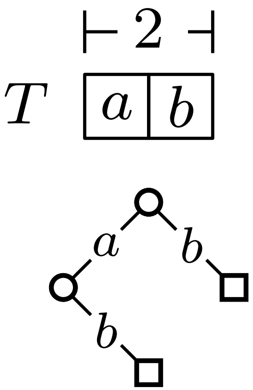
16 nodes

25 nodes

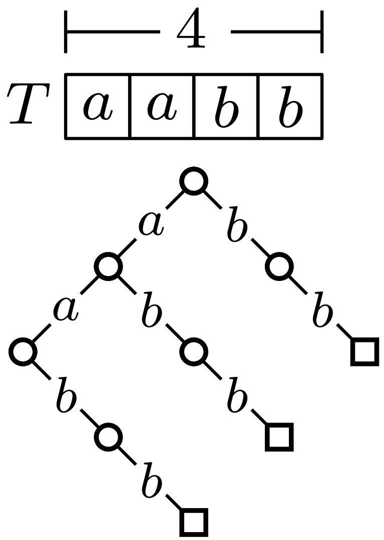
36 nodes

An atomic suffix tree can have $((n/2)+1)^2$ nodes

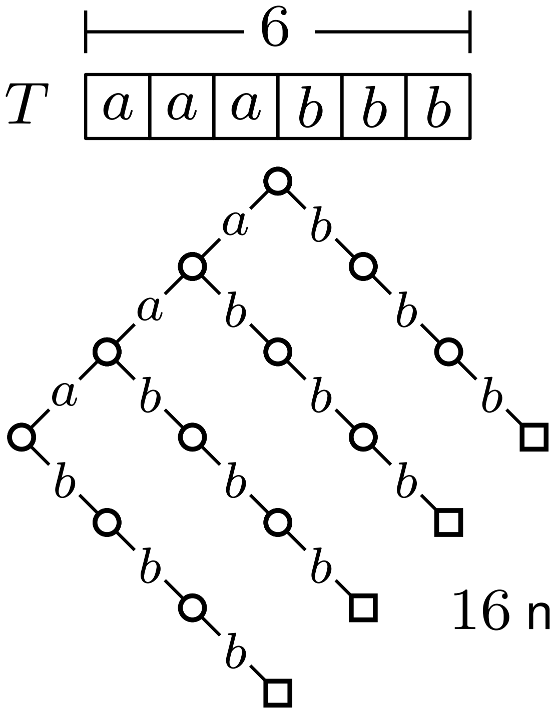
# how large is the atomic suffix tree?



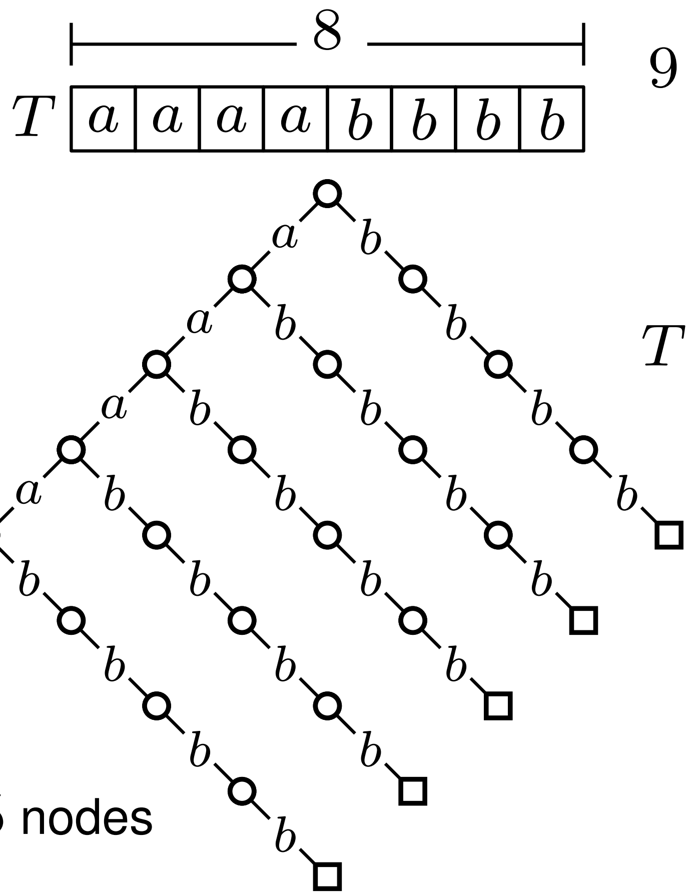
4 nodes



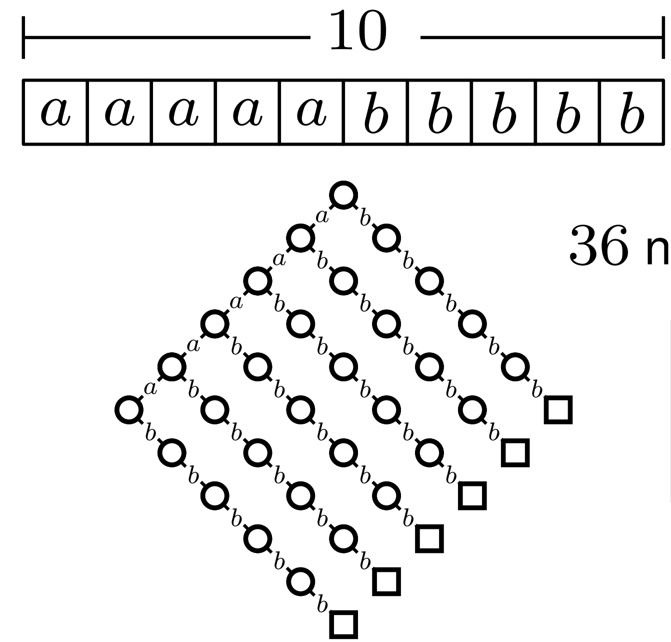
9 nodes



16 nodes



25 nodes

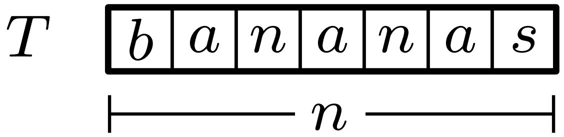


36 nodes

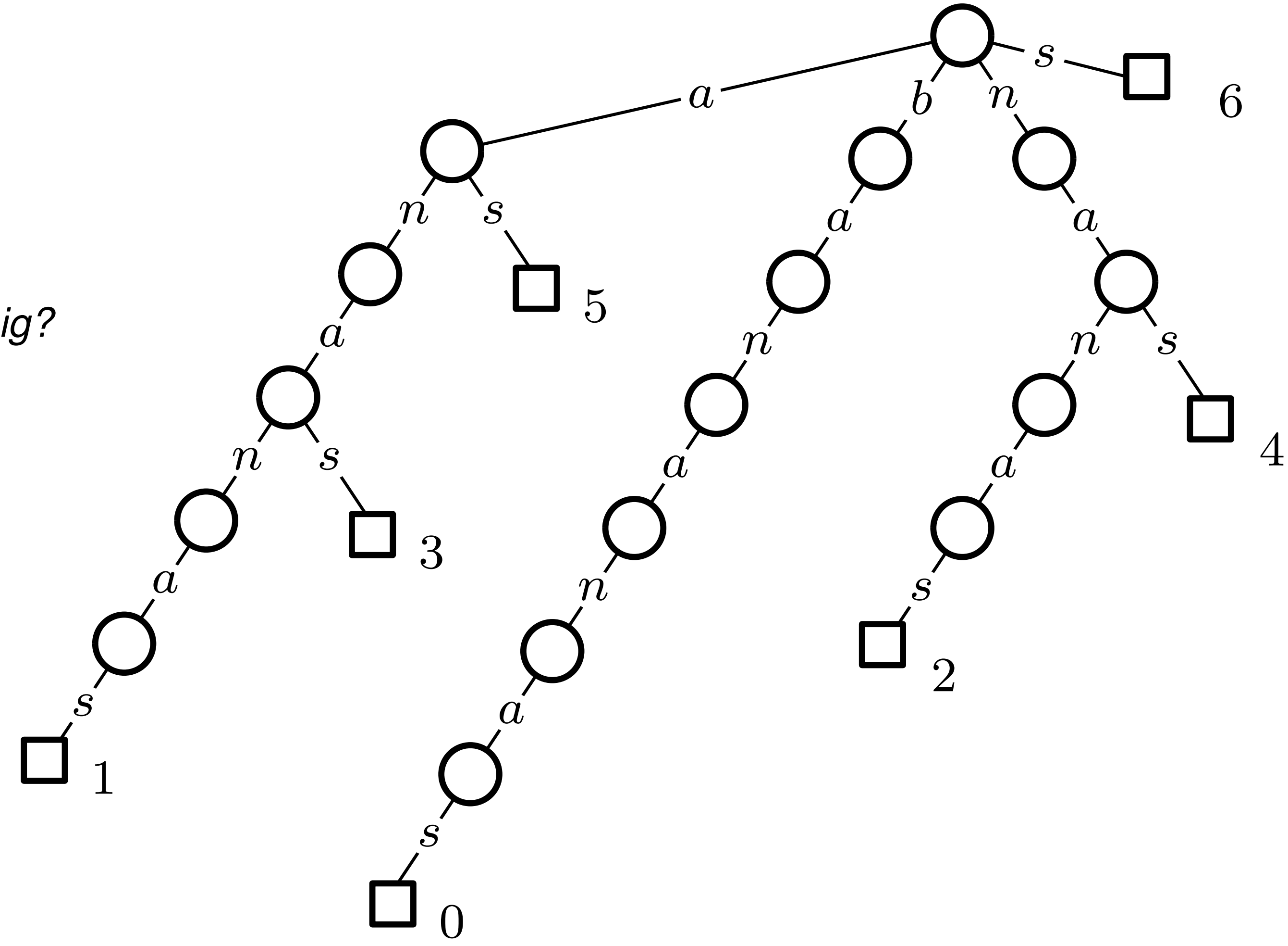
An atomic suffix tree can have $((n/2)+1)^2$ nodes

*this is **far** too big!*

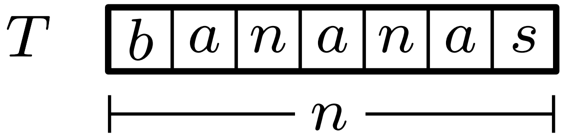
# Compacted suffix trees



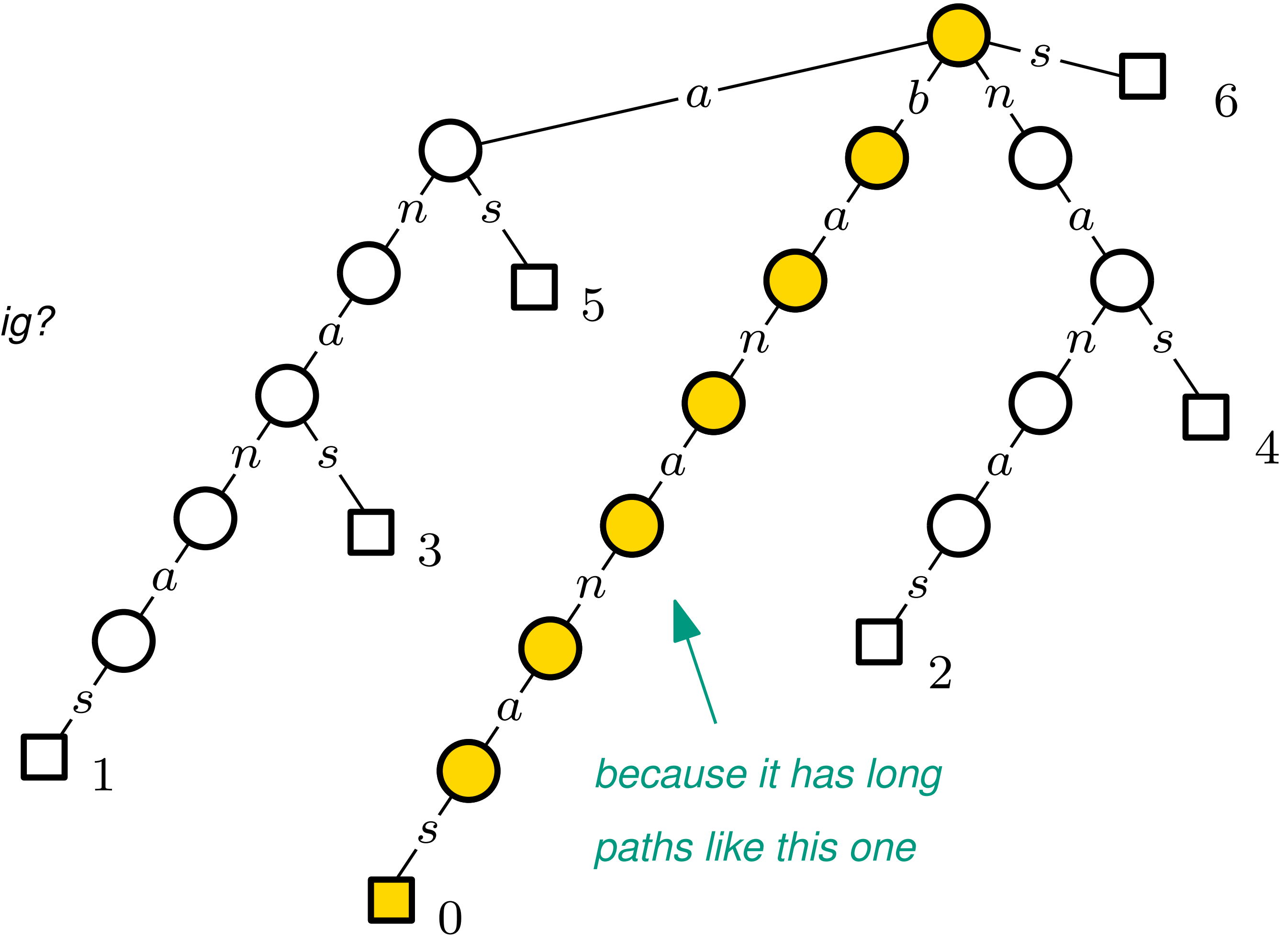
*Why is the atomic suffix tree so big?*



# Compacted suffix trees

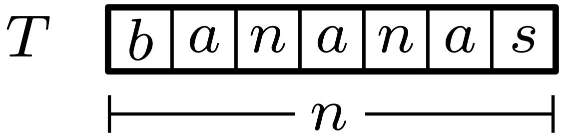


*Why is the atomic suffix tree so big?*

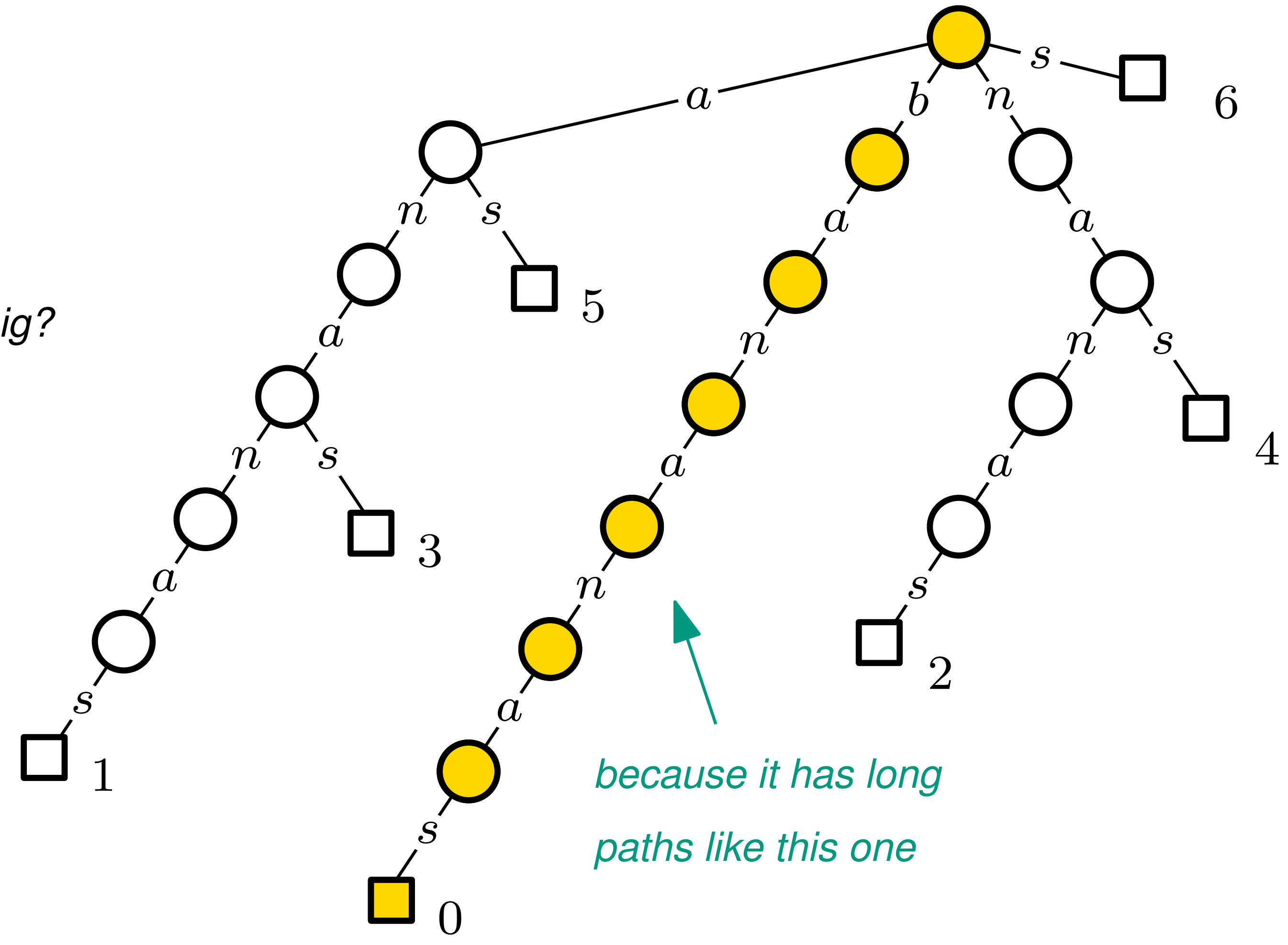


*because it has long paths like this one*

# Compacted suffix trees



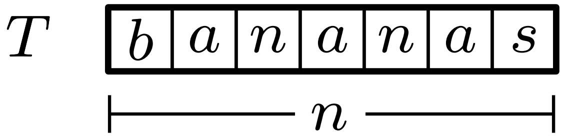
*Why is the atomic suffix tree so big?*



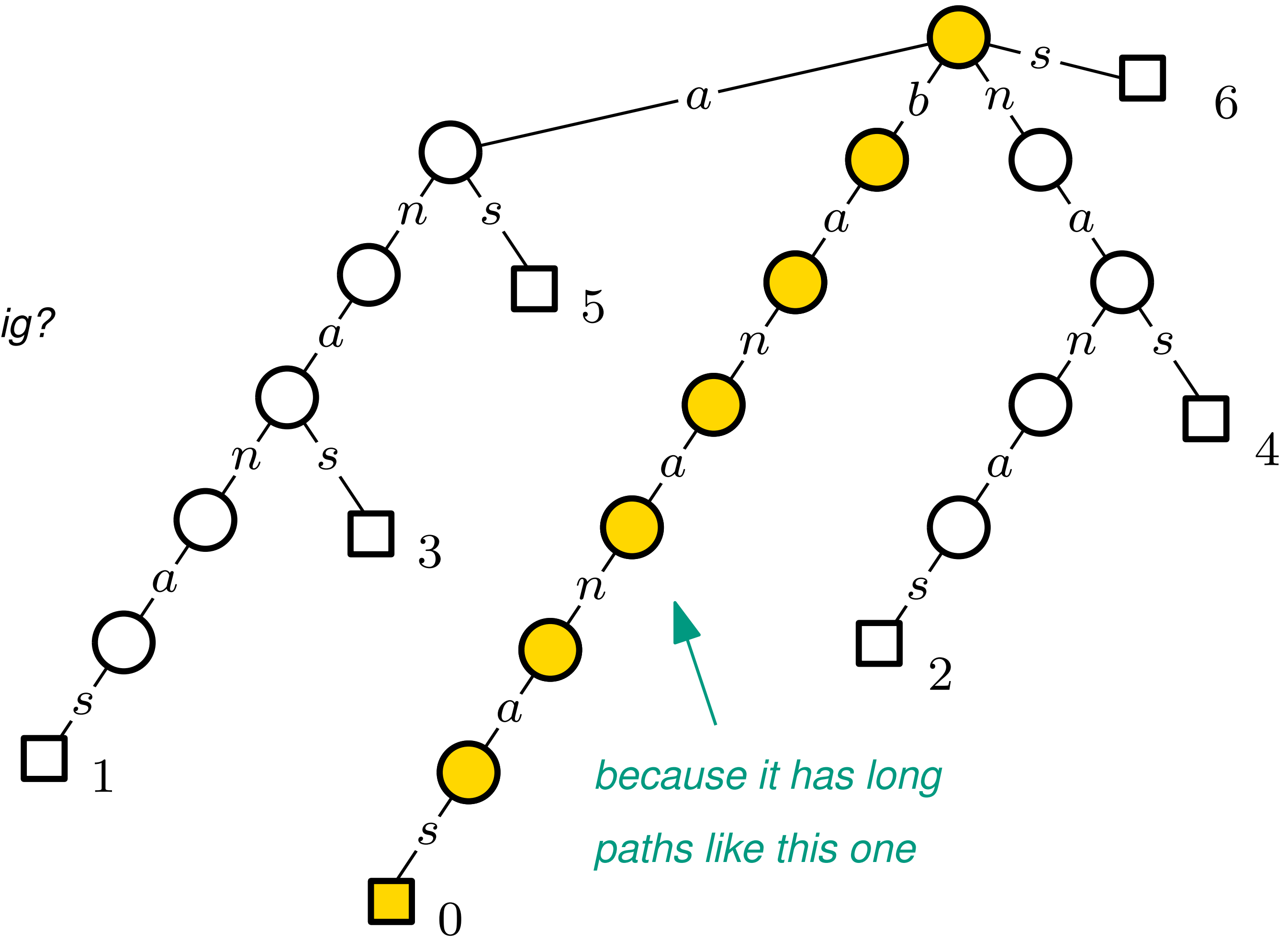
*because it has long paths like this one*

**Main Idea** replace each non-branching path with a single edge

# Compacted suffix trees



*Why is the atomic suffix tree so big?*

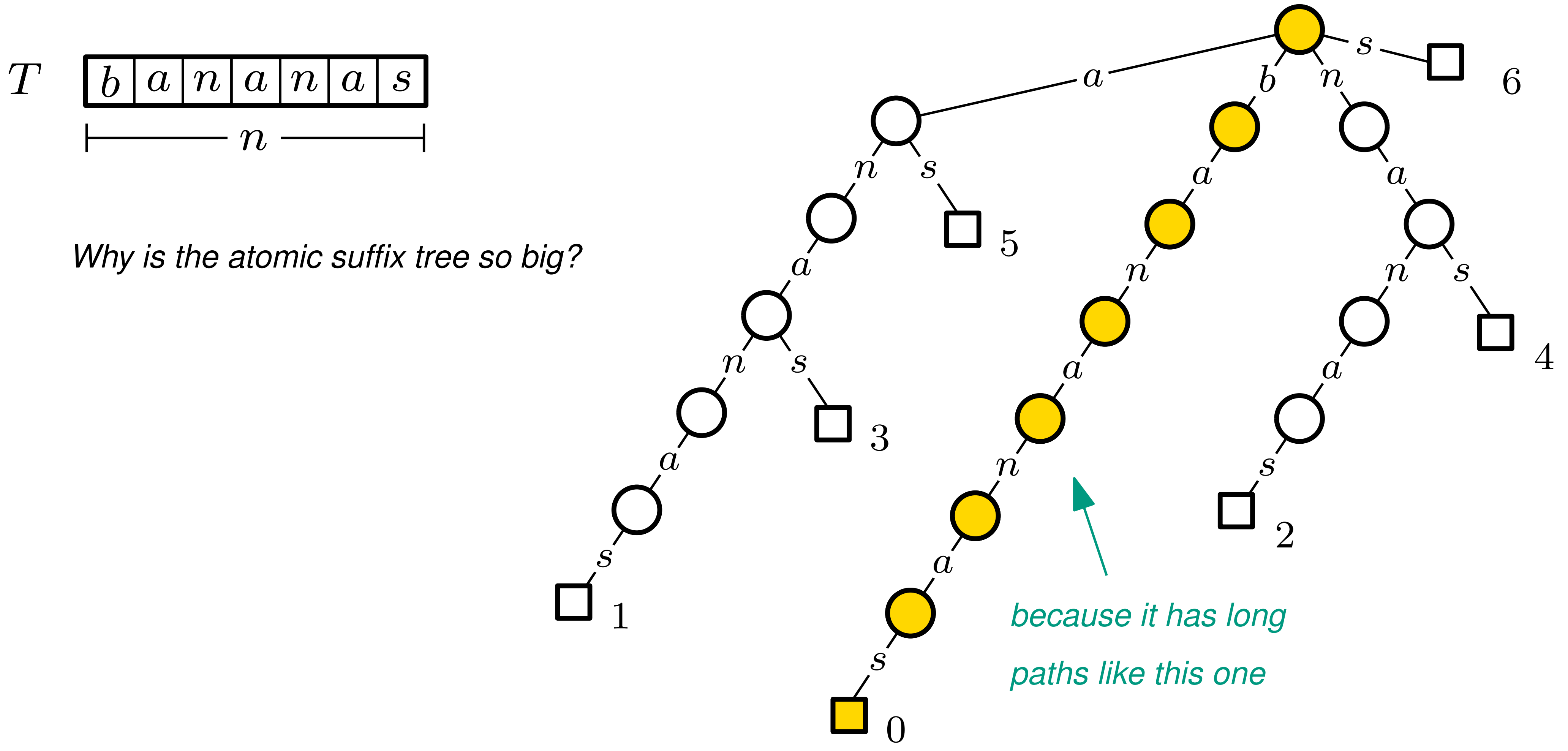


*because it has long paths like this one*

**Main Idea** replace each non-branching path with a single edge

- edges are now labelled with substrings

# Compacted suffix trees



*Why is the atomic suffix tree so big?*

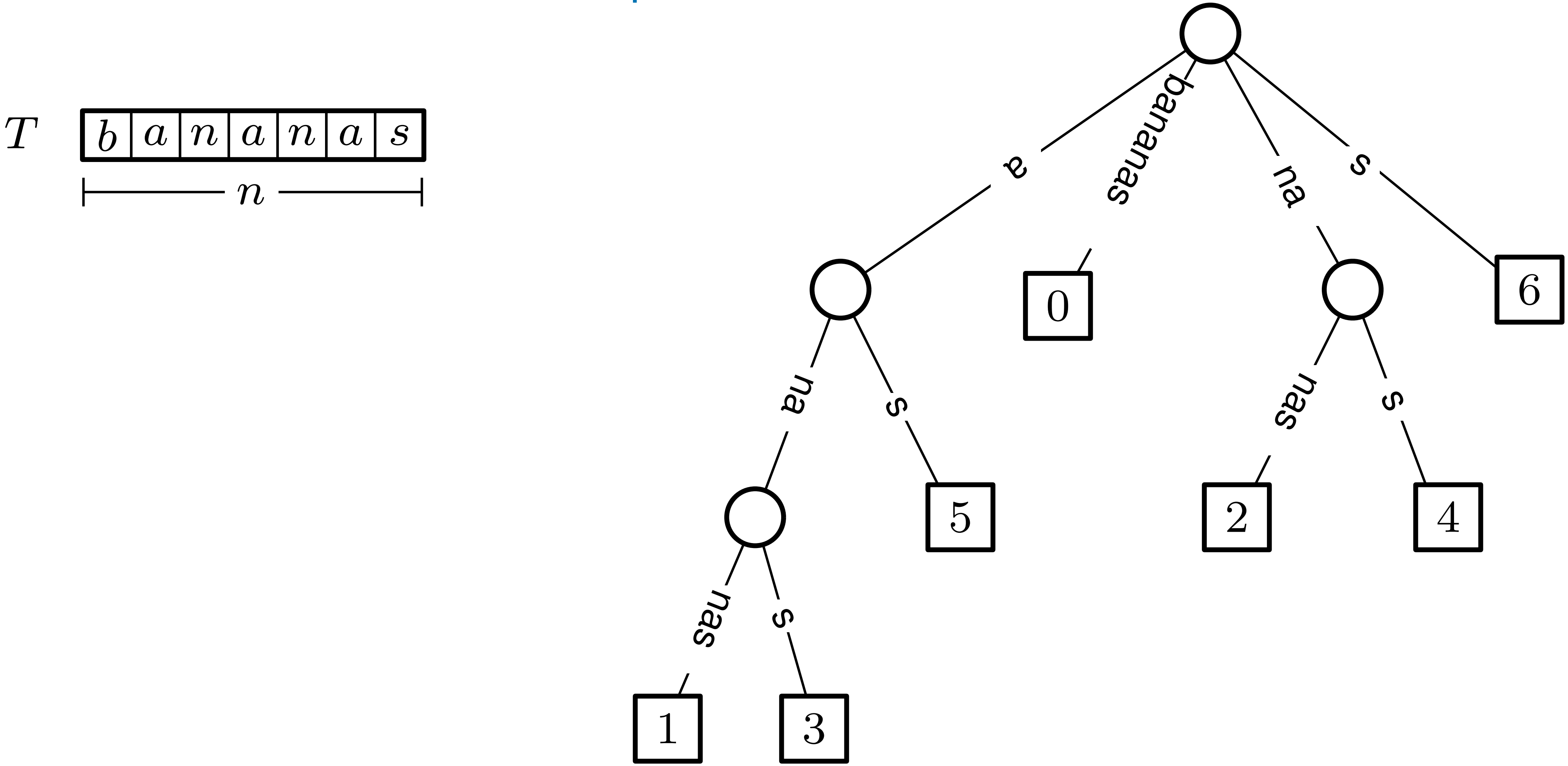
*because it has long paths like this one*

**Main Idea** replace each non-branching path with a single edge

- edges are now labelled with substrings

*(instead of single characters)*

# Compacted suffix trees

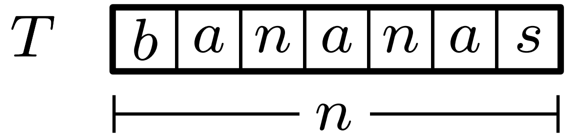


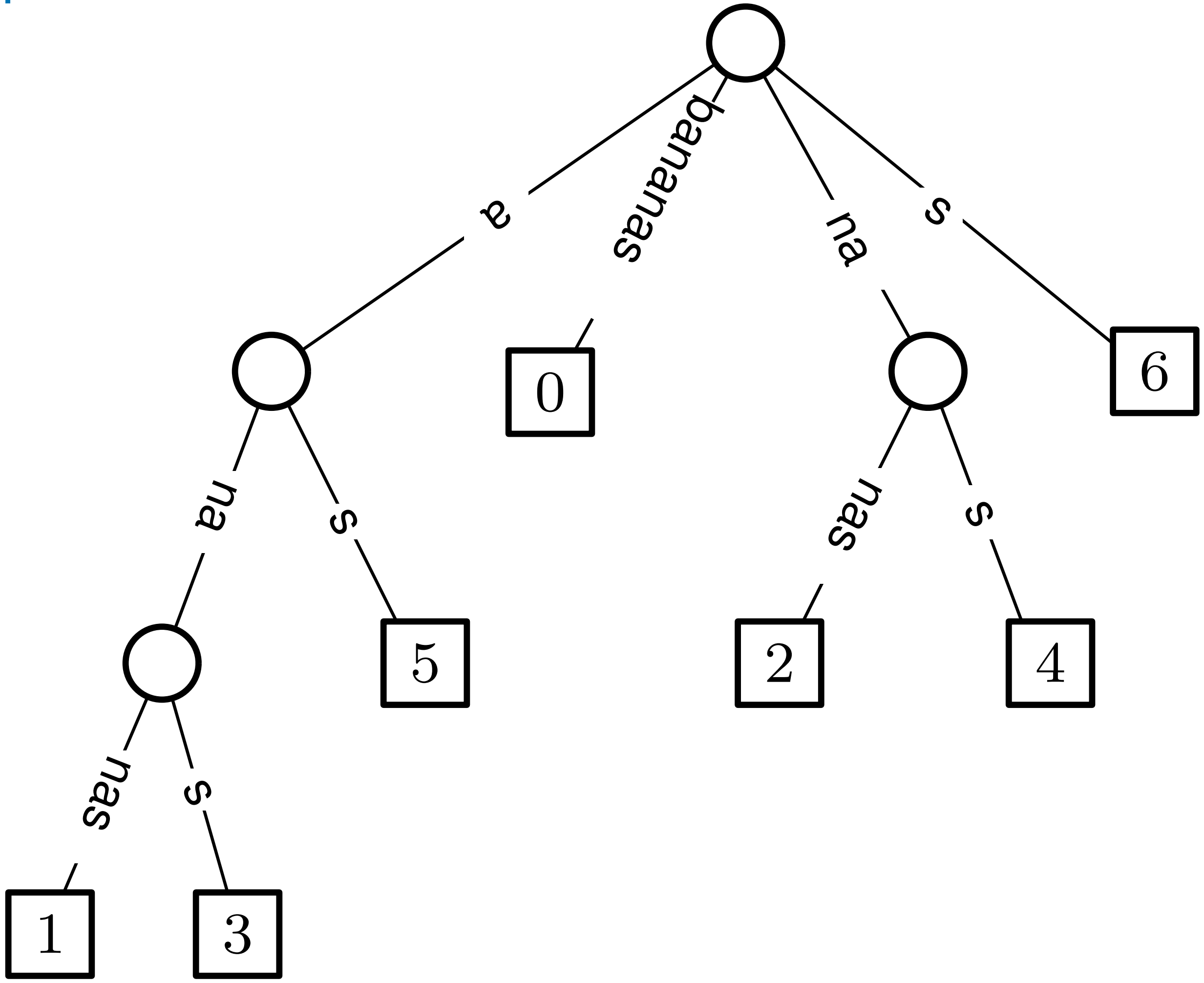
**Main Idea** replace each non-branching path with a single edge

- edges are now labelled with substrings

*(instead of single characters)*

# Compacted suffix trees





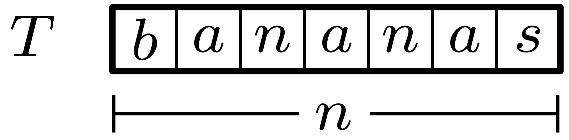
- There are at most $n$ leaves

**Main Idea** replace each non-branching path with a single edge

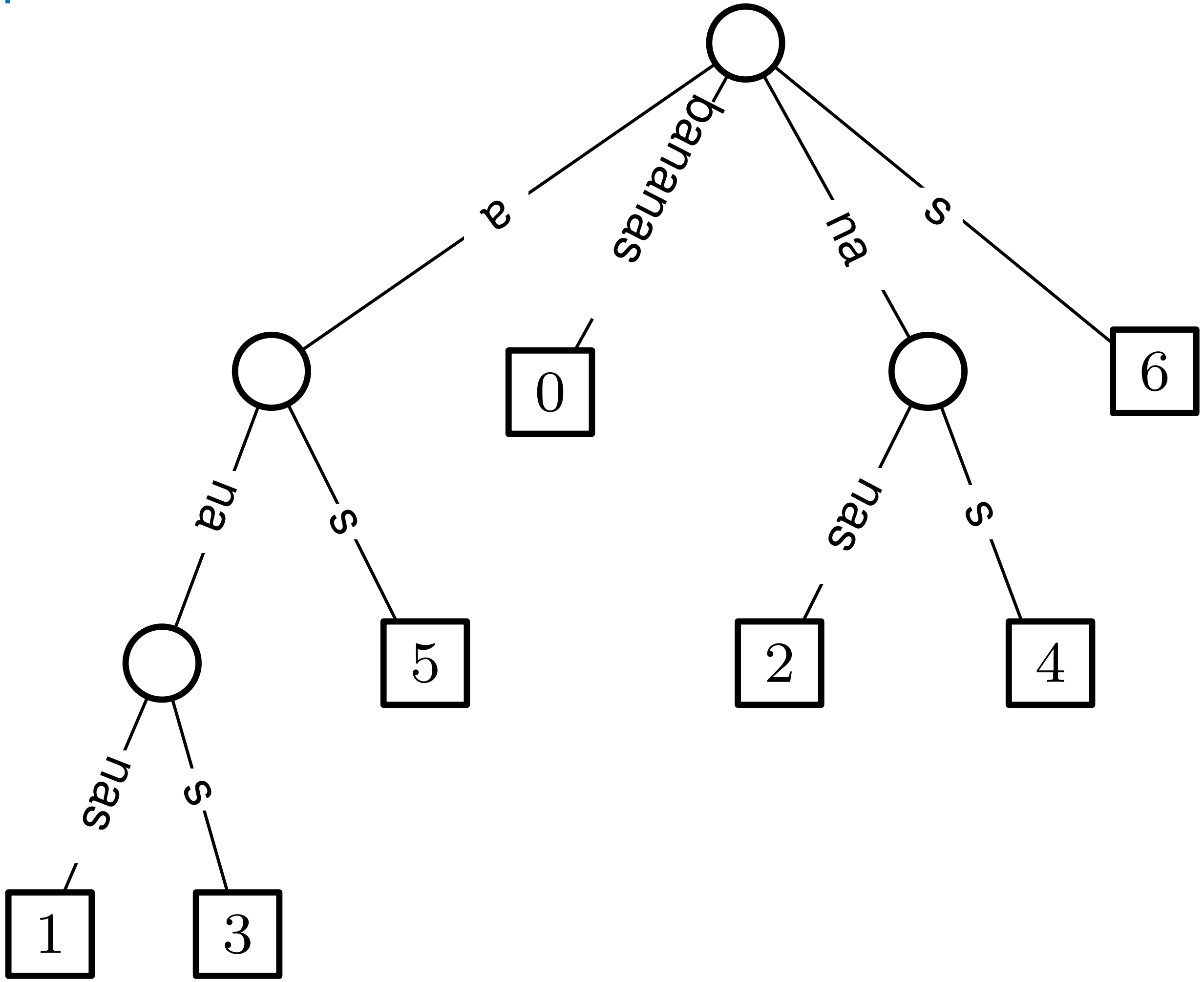
- edges are now labelled with substrings

*(instead of single characters)*

# Compacted suffix trees



- There are at most $n$ leaves
- Every internal node has two or more children

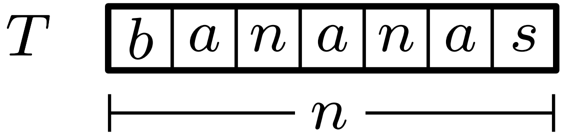


**Main Idea** replace each non-branching path with a single edge

- edges are now labelled with substrings

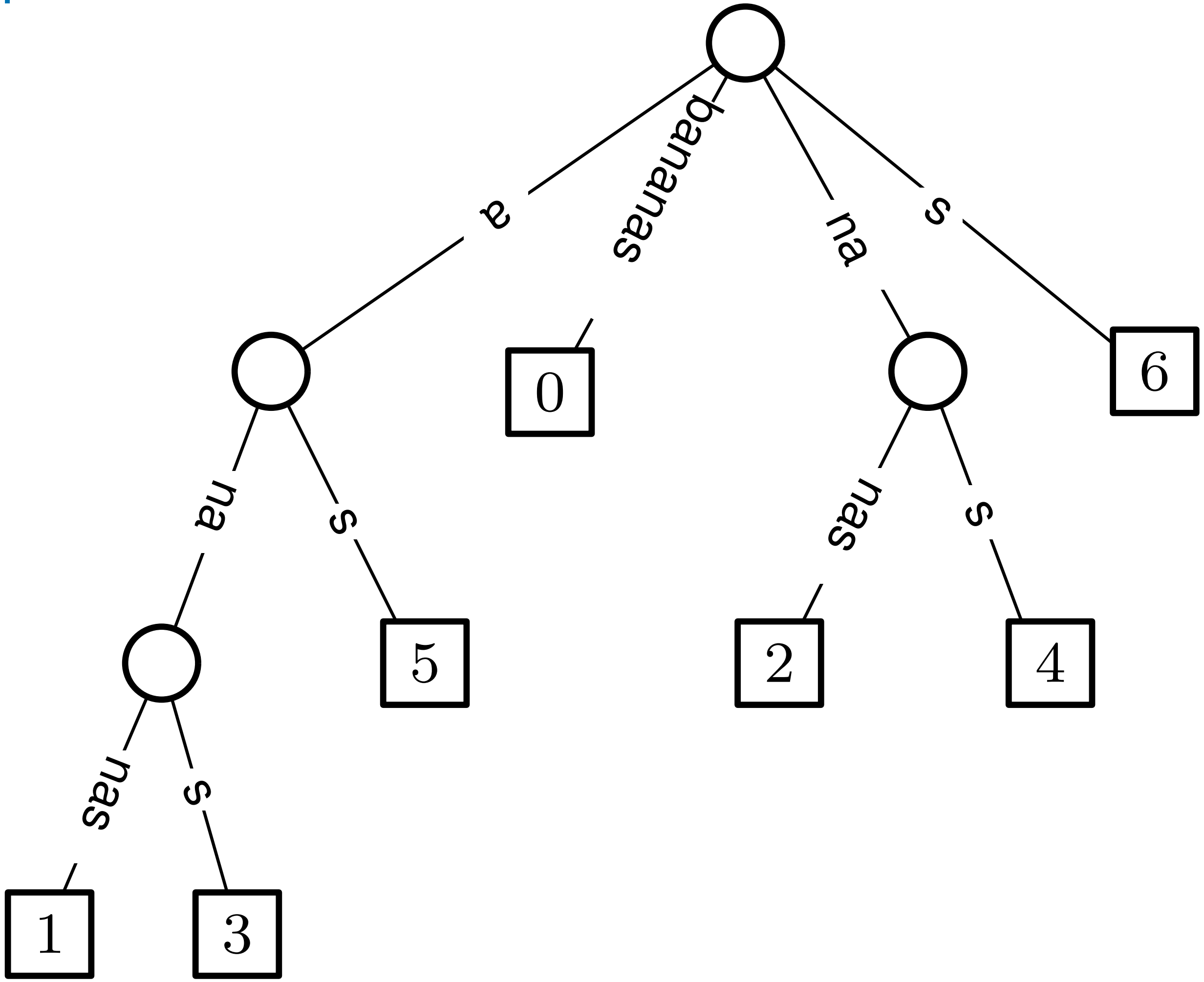
*(instead of single characters)*

# Compacted suffix trees



- There are at most $n$ leaves
- Every internal node has two or more children

*so there are* $O(n)$ *edges*

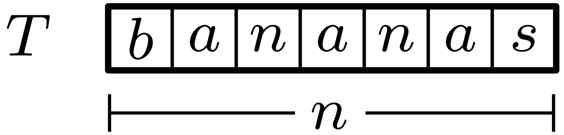


**Main Idea** replace each non-branching path with a single edge

- edges are now labelled with substrings

*(instead of single characters)*

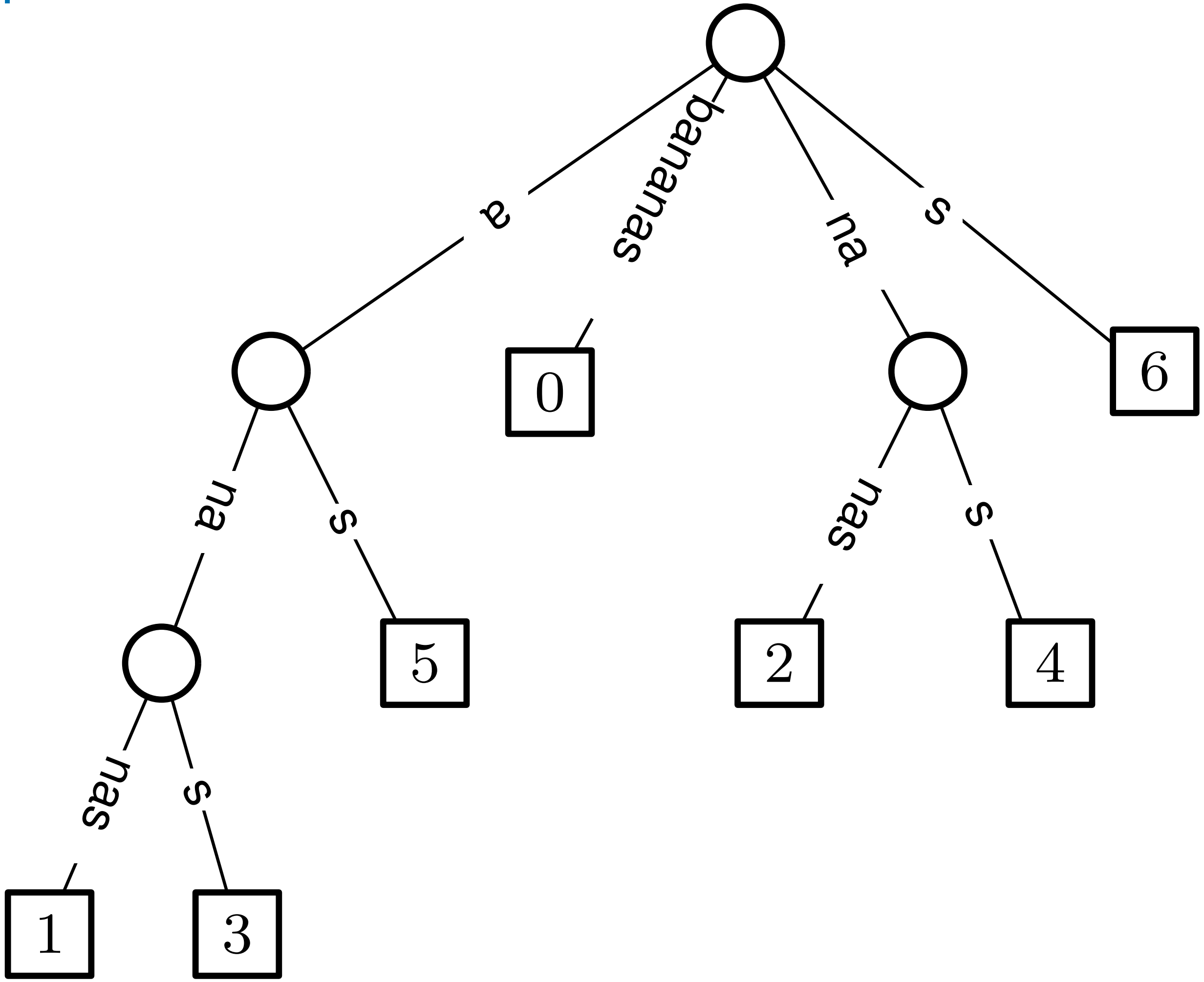
# Compacted suffix trees



- There are at most $n$ leaves
- Every internal node has two or more children

*so there are $O(n)$ edges*

don't the edges take up lots of space?



**Main Idea** replace each non-branching path with a single edge

- edges are now labelled with substrings

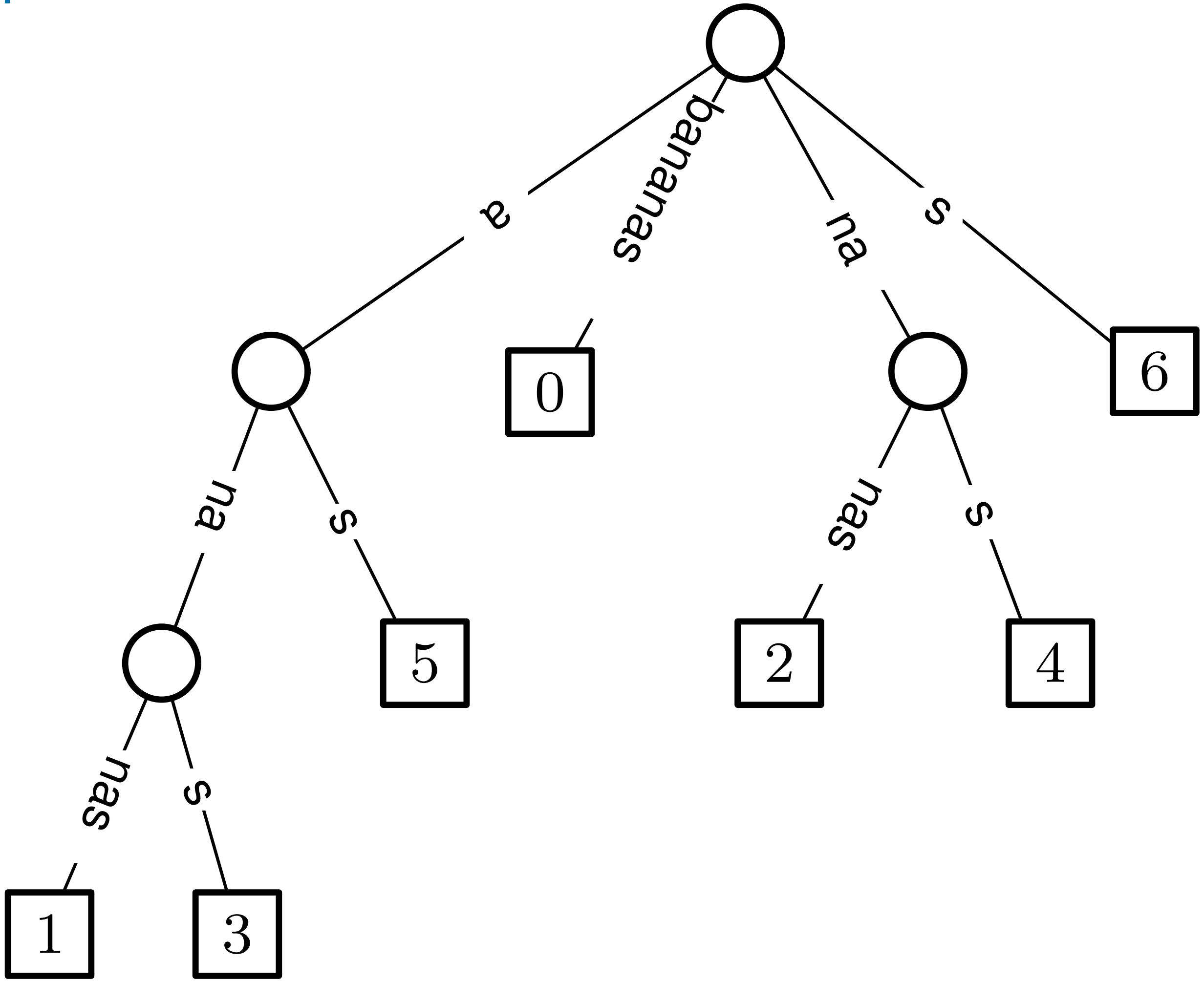
*(instead of single characters)*

# Compacted suffix trees

$T$

| | | | | | | |
|---|---|---|---|---|---|---|
| $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |

$\longmapsfrom n \longmapsto$



- There are at most $n$ leaves
- Every internal node has two or more children

*so there are $O(n)$ edges*

don't the edges take up lots of space?

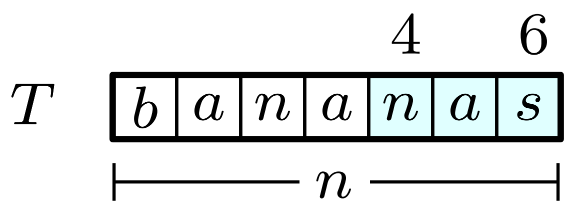
*we only store the end points*

**Main Idea** replace each non-branching path with a single edge

- edges are now labelled with substrings

*(instead of single characters)*

# Compacted suffix trees

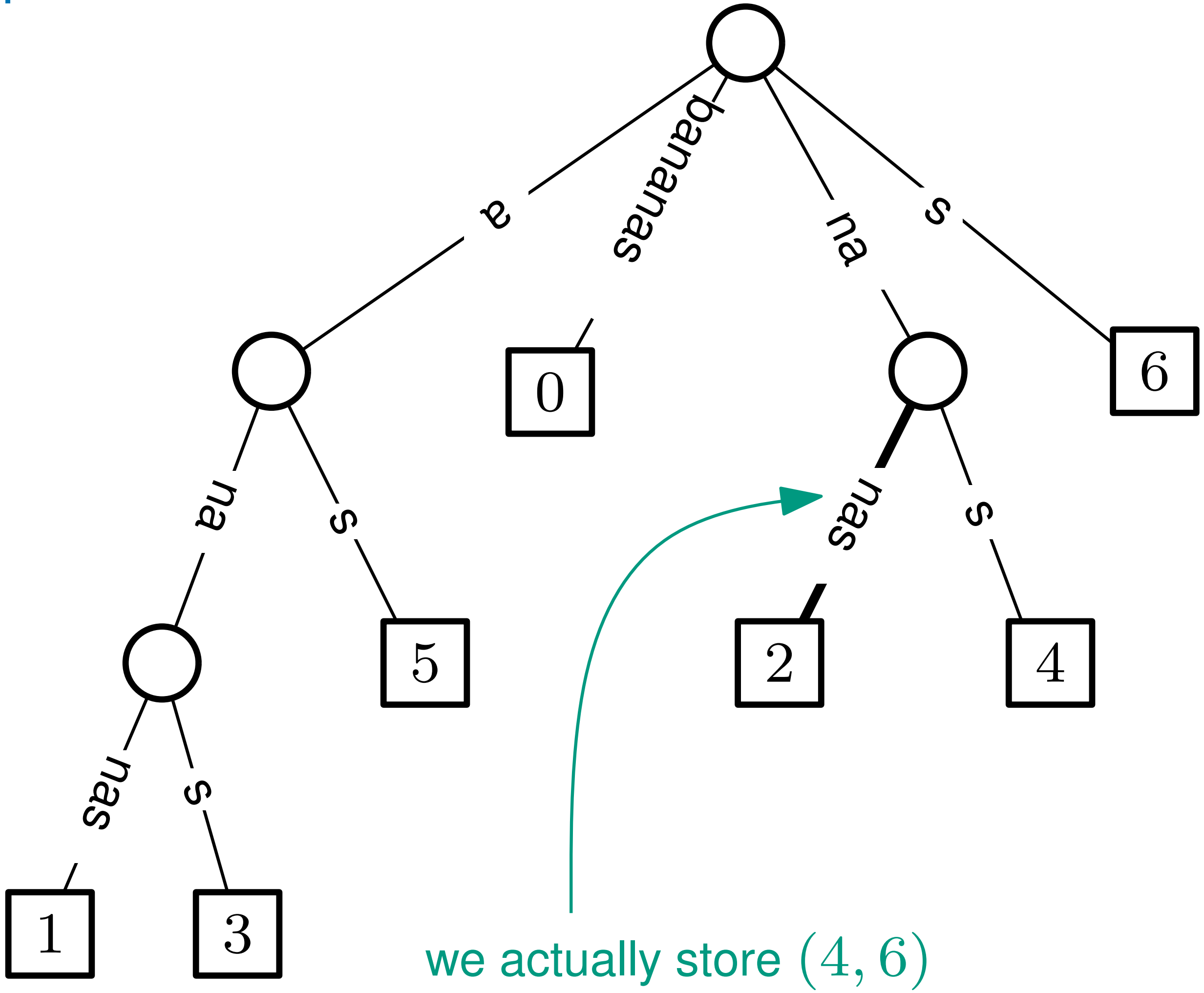


- There are at most $n$ leaves
- Every internal node has two or more children

*so there are $O(n)$ edges*

don't the edges take up lots of space?

*we only store the end points*



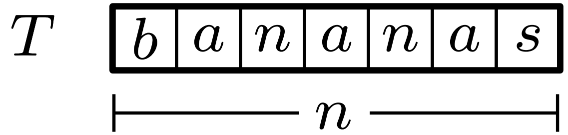
we actually store $(4,6)$

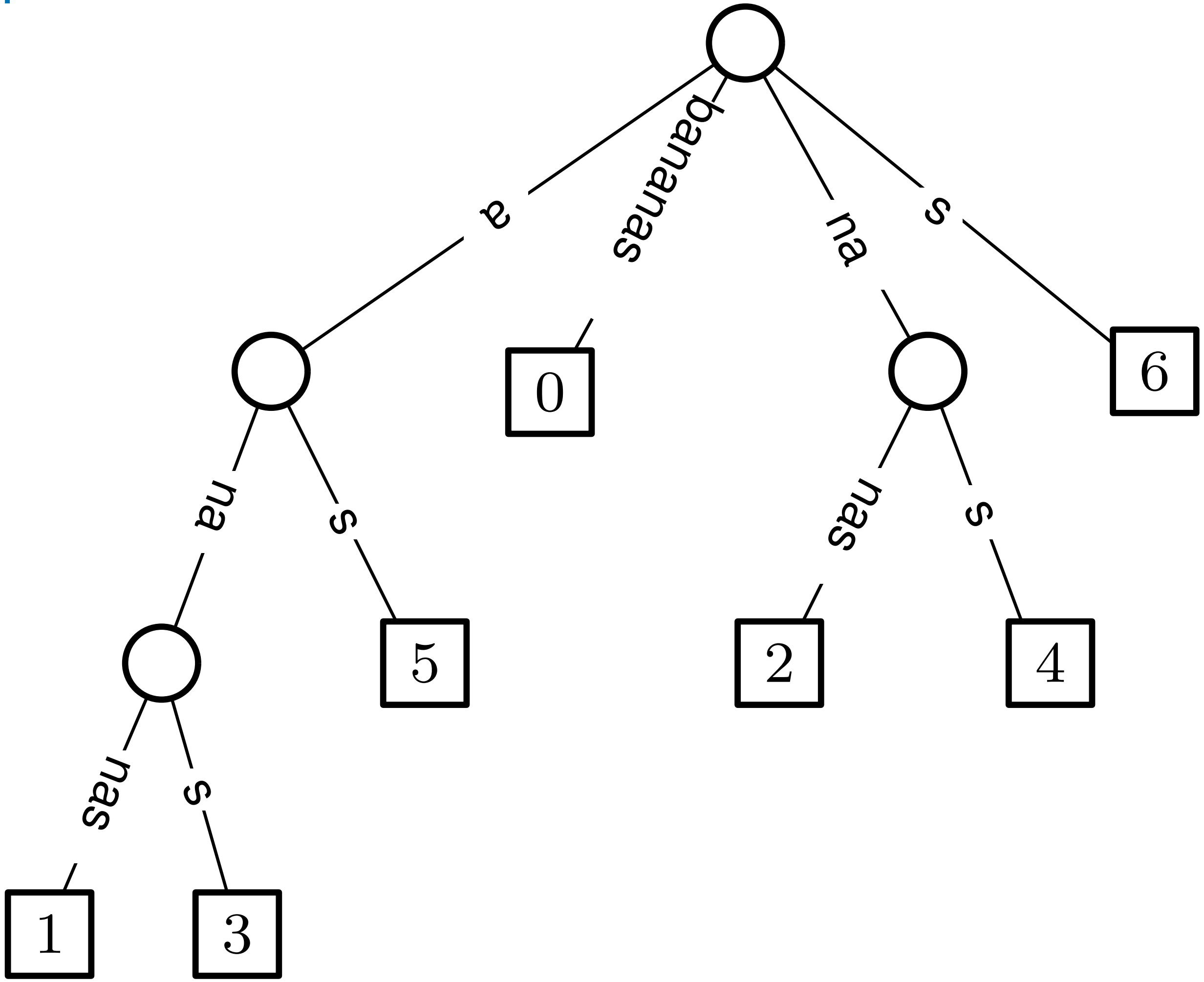
**Main Idea** replace each non-branching path with a single edge

- edges are now labelled with substrings

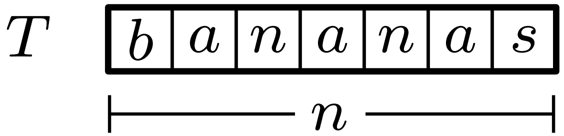
*(instead of single characters)*

# Compacted suffix trees

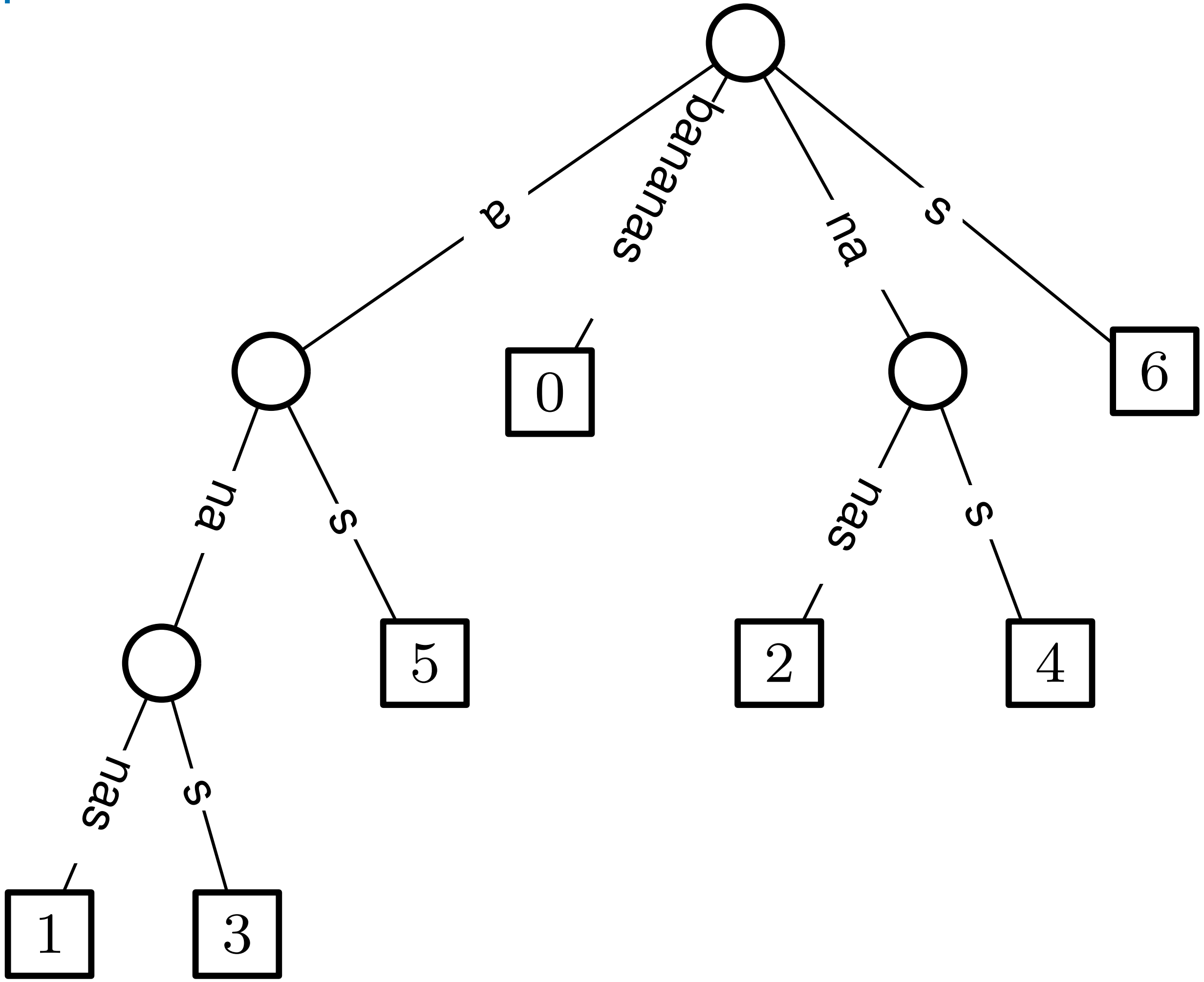




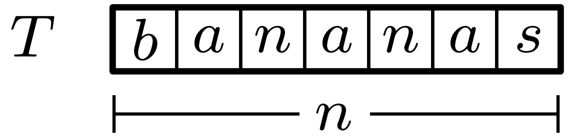
# Compacted suffix trees

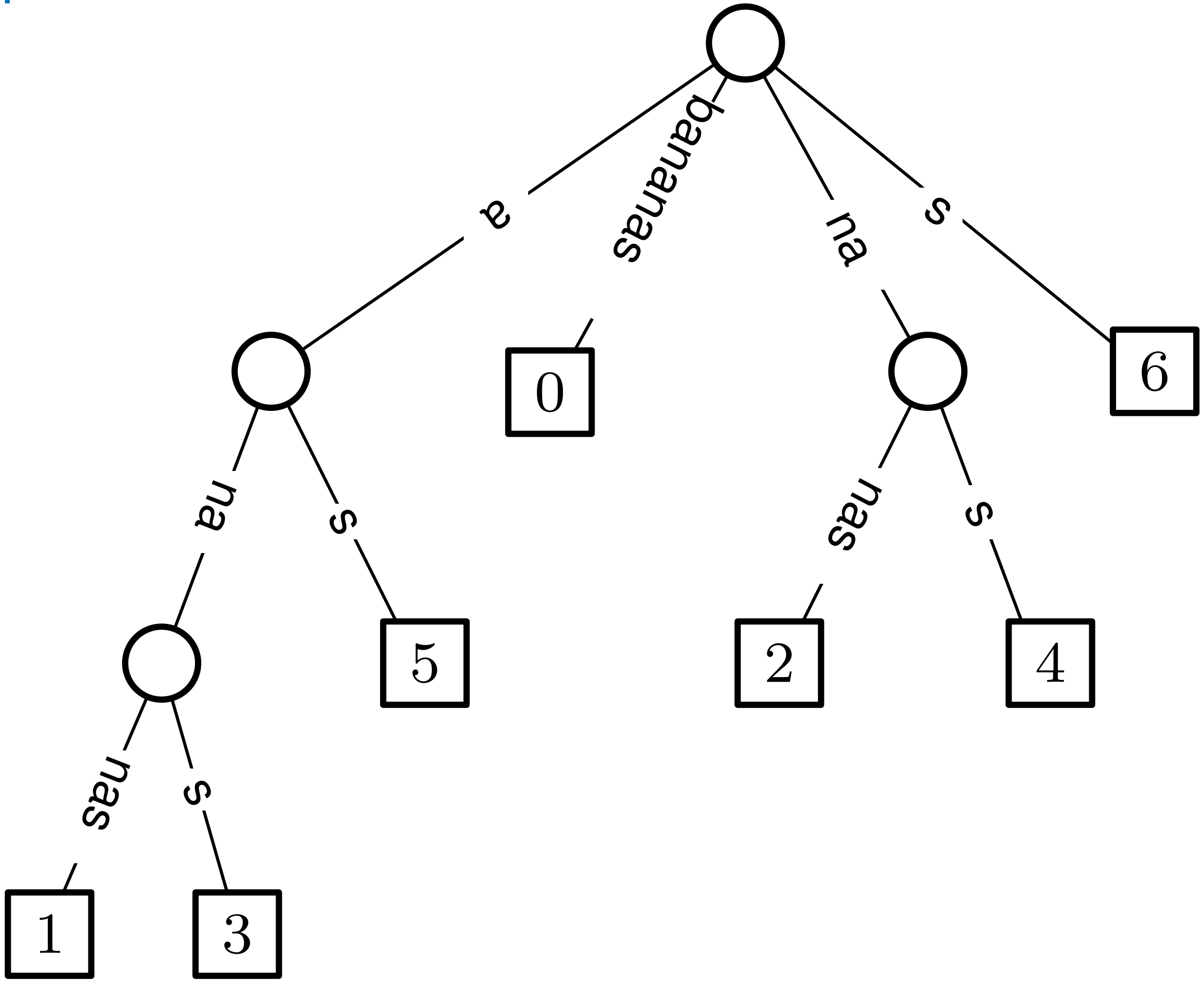


**Compacted Suffix Tree** of $T$



# Compacted suffix trees

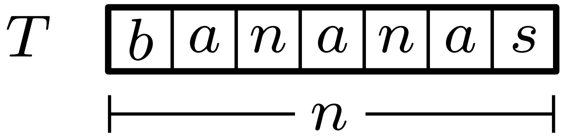




**Compacted Suffix Tree** of $T$

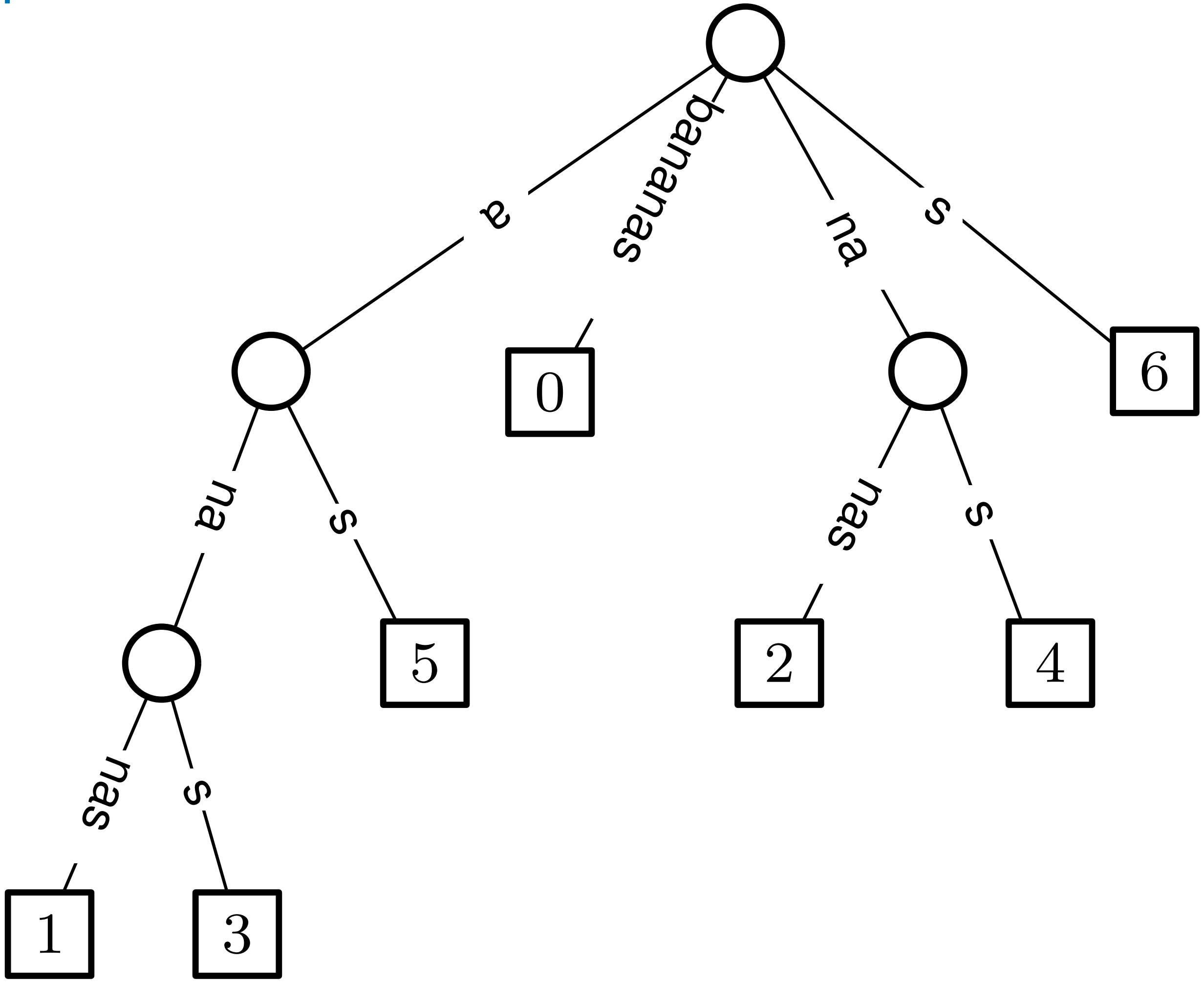
- A rooted tree with $n$ leaves

# Compacted suffix trees

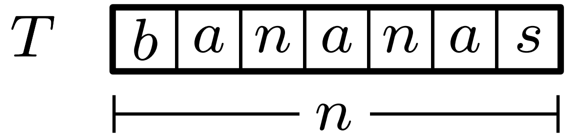


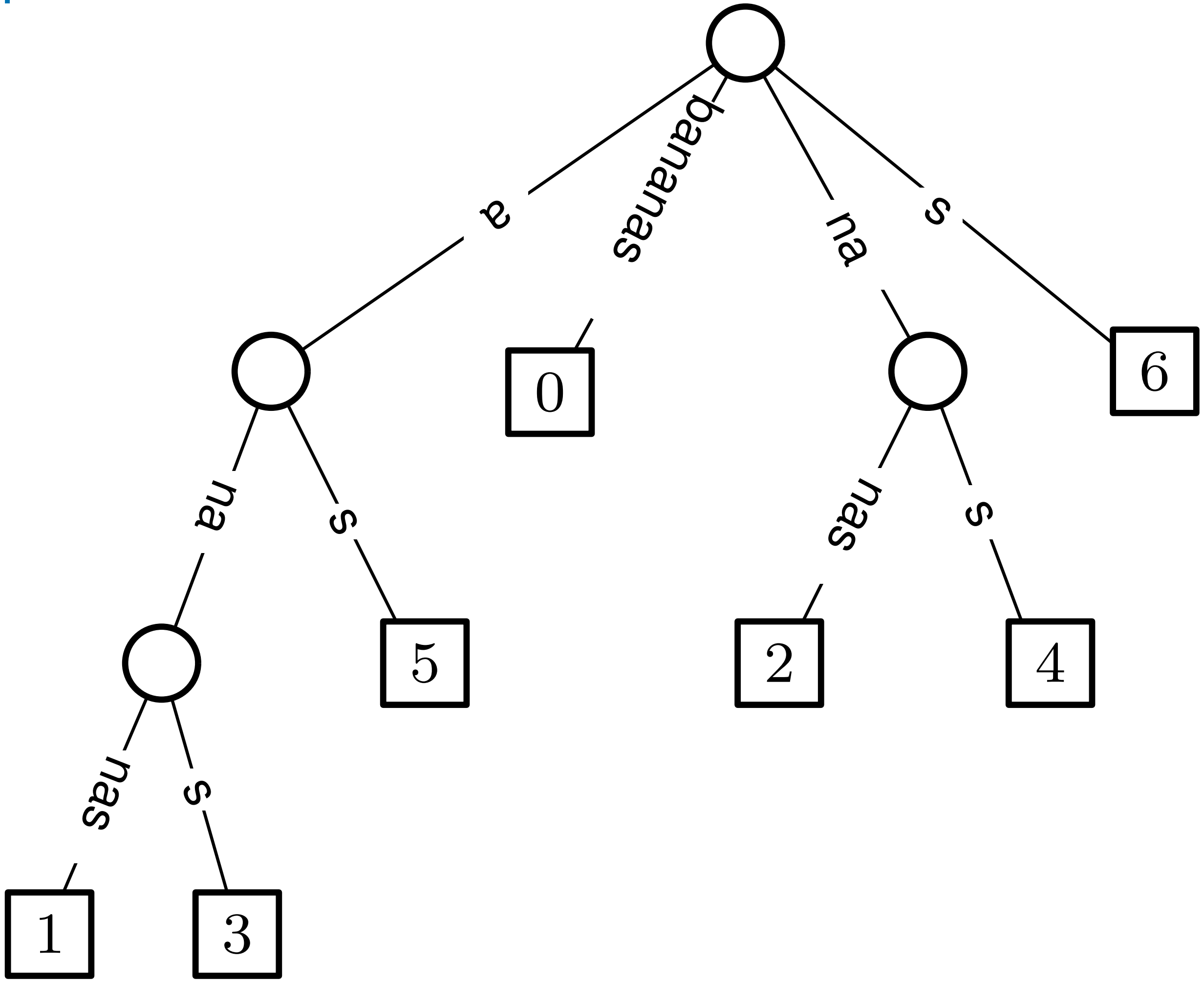
**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children



# Compacted suffix trees

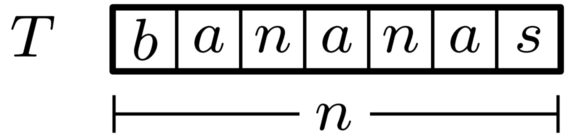


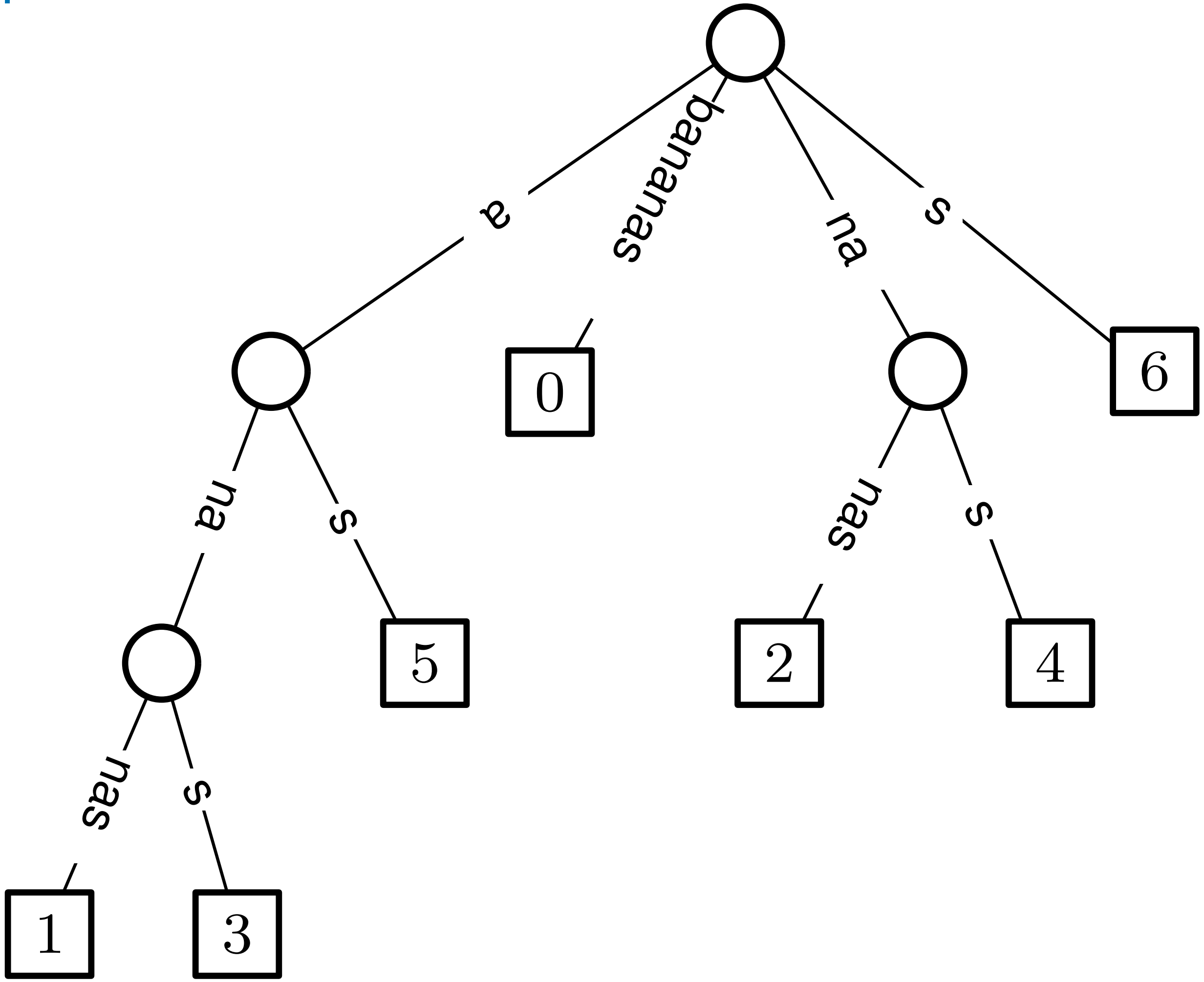


**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring

# Compacted suffix trees

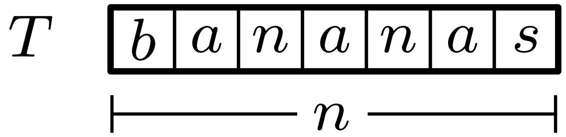


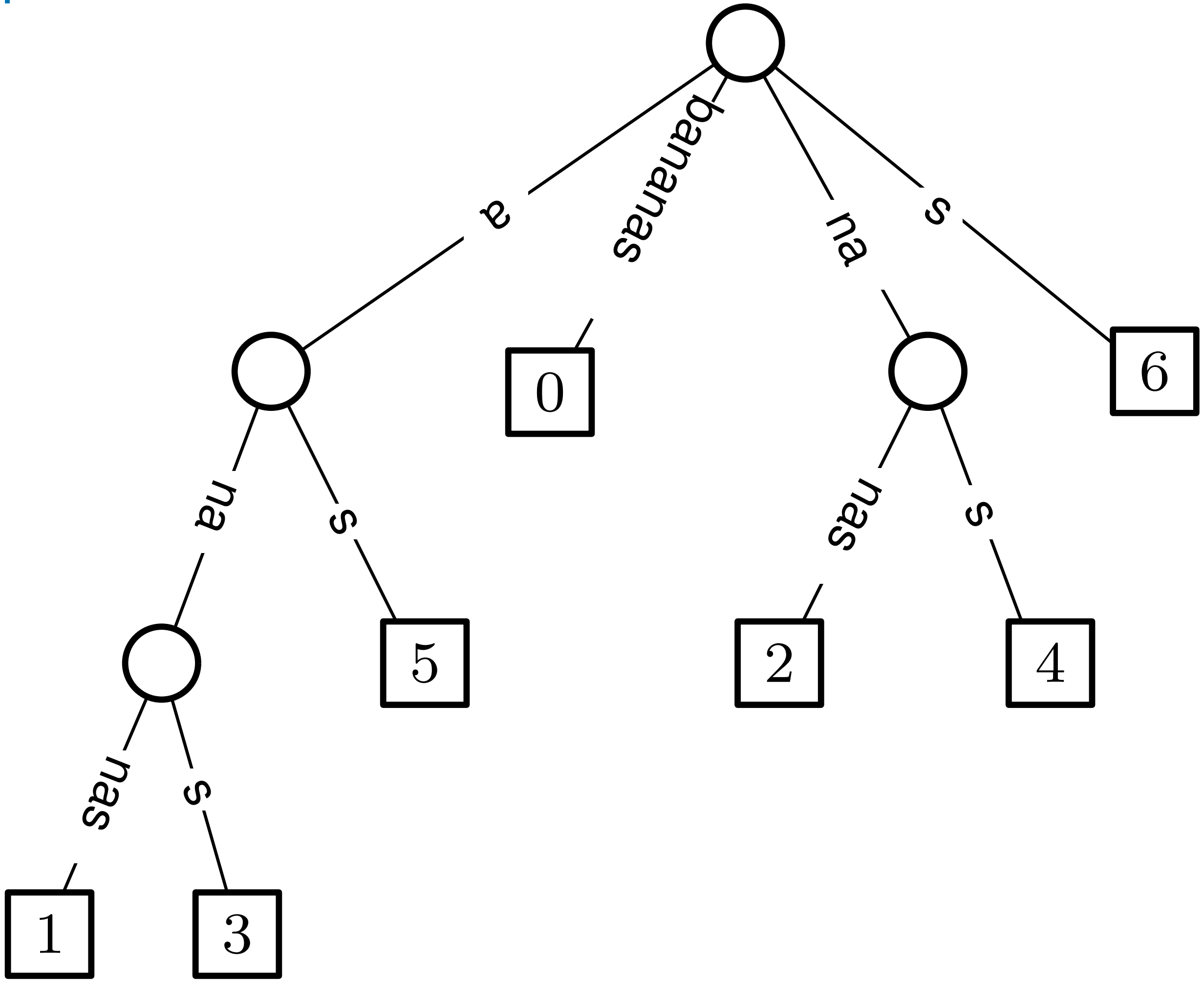


**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character

# Compacted suffix trees

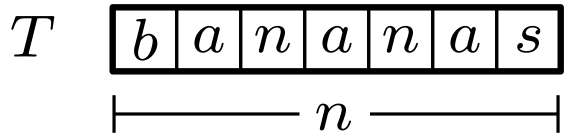


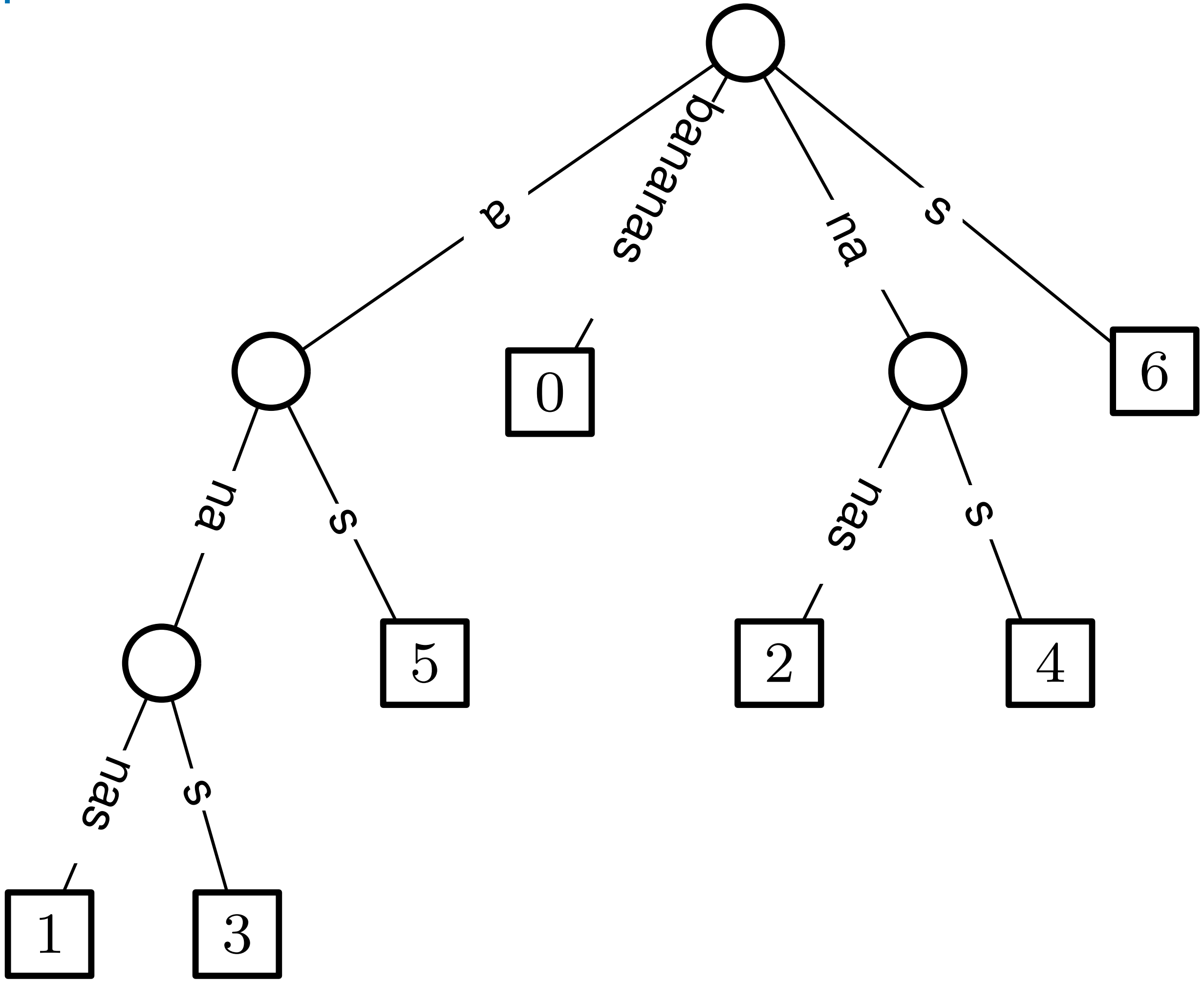


**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$

# Compacted suffix trees

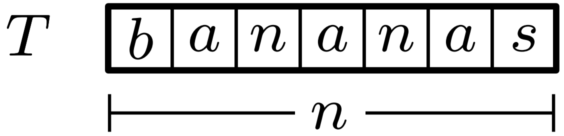


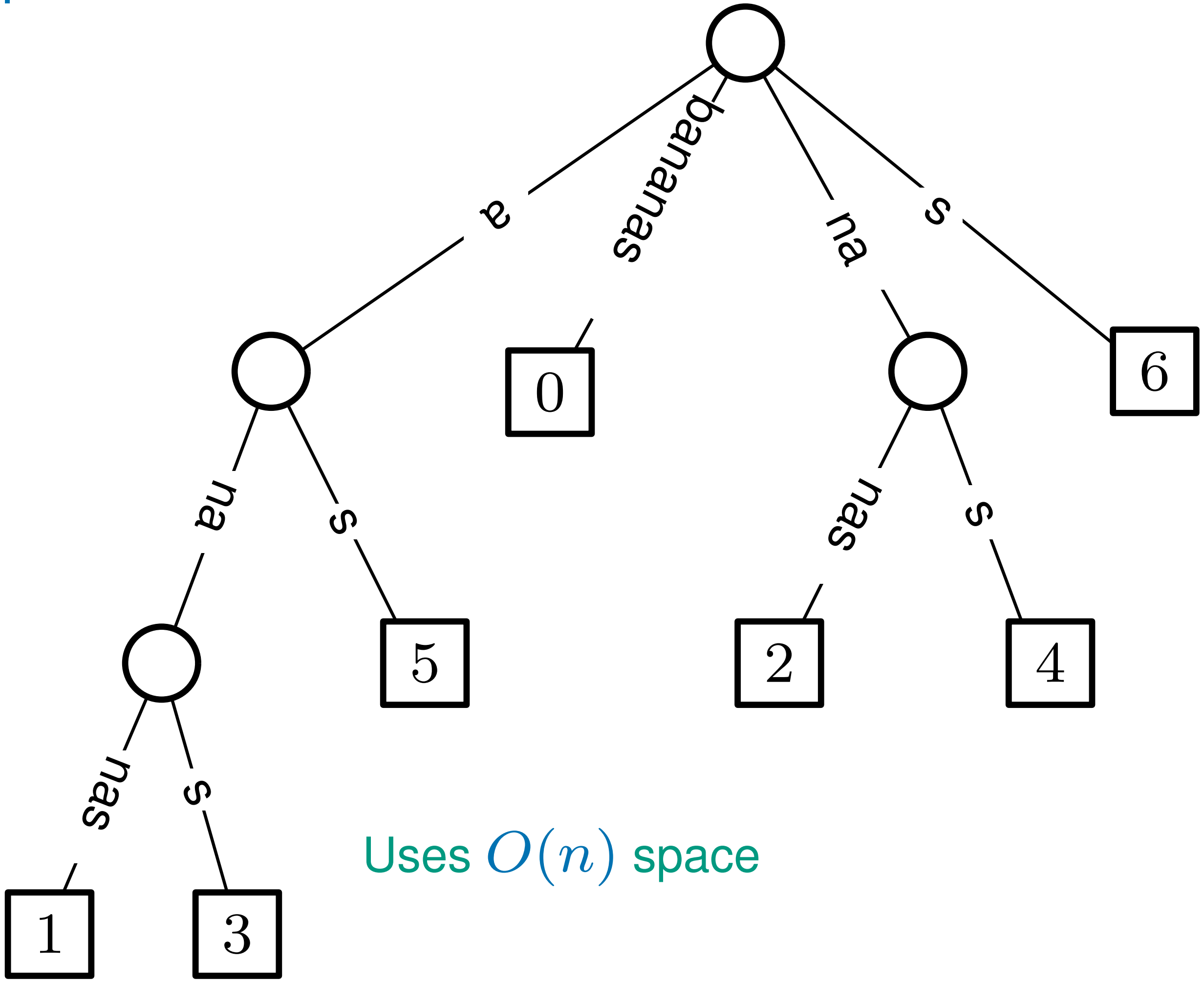


**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

# Compacted suffix trees





**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring

Uses $O(n)$ space

- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

# Compacted suffix trees

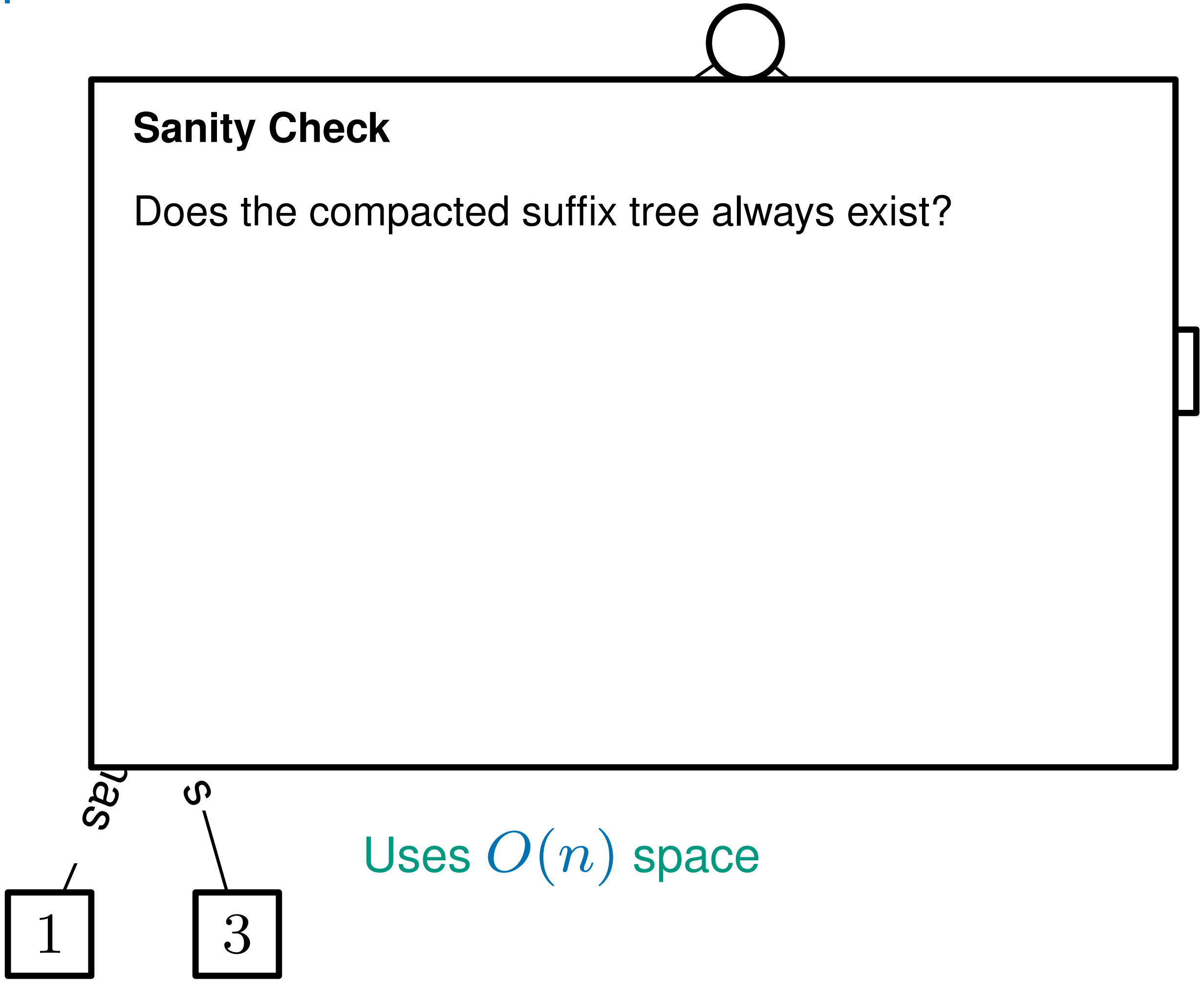
$T$

| $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
|---|---|---|---|---|---|---|

$\longleftarrow n \longrightarrow$

**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring



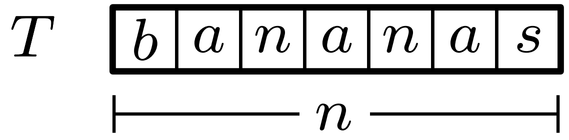
**Sanity Check**

Does the compacted suffix tree always exist?

Uses $O(n)$ space

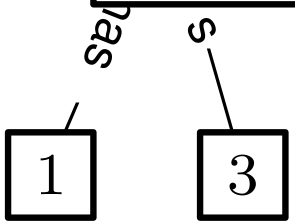
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

# Compacted suffix trees



**Compacted Suffix Tree** of $T$

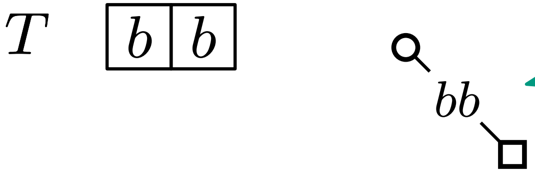
- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix



Uses $O(n)$ space

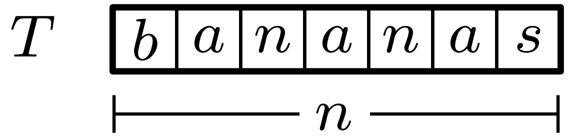
**Sanity Check**

Does the compacted suffix tree always exist?



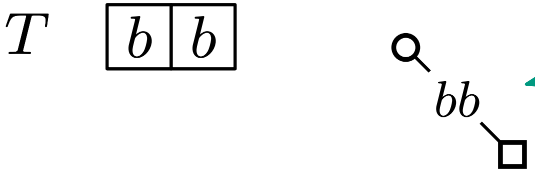
this doesn't have $n$ leaves

# Compacted suffix trees

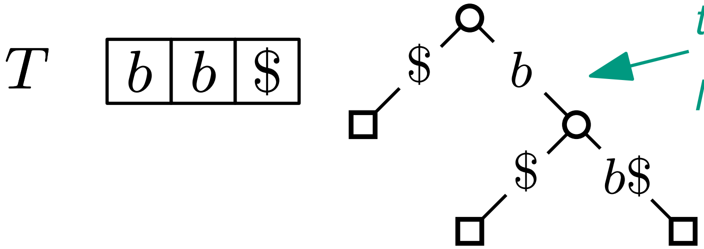


**Sanity Check**

Does the compacted suffix tree always exist?



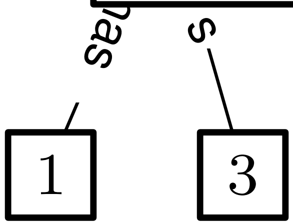
this doesn't have $n$ leaves



this has $n$ leaves

**Compacted Suffix Tree** of $T$

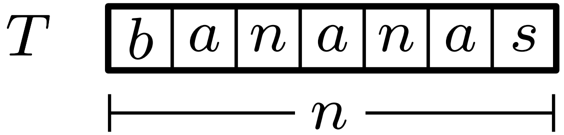
- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring

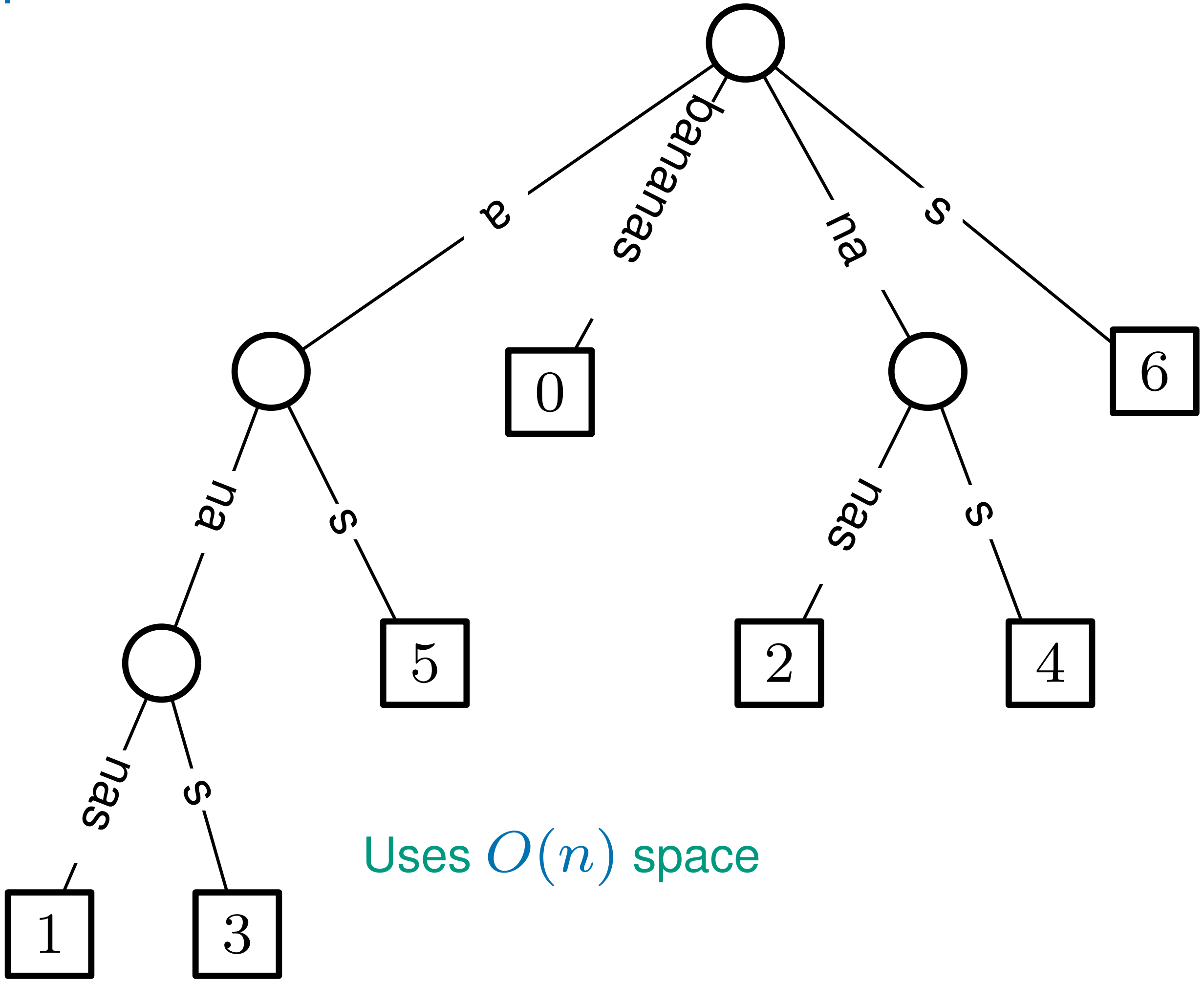


Uses $O(n)$ space

- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

# Compacted suffix trees





**Compacted Suffix Tree** of $T$

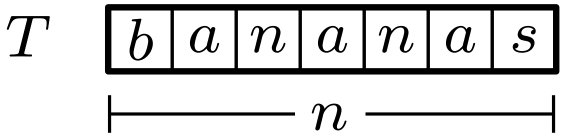
- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring

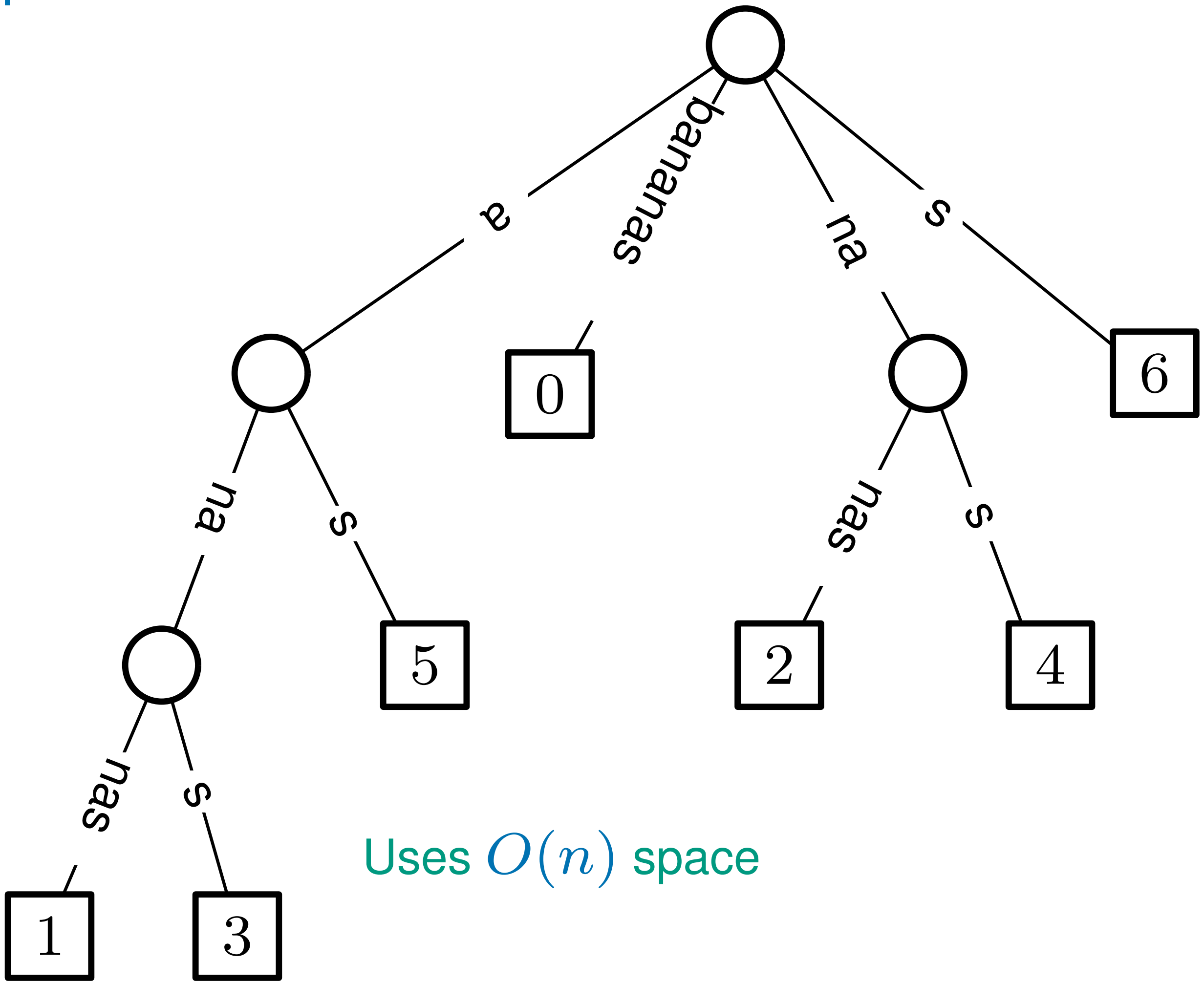
Uses $O(n)$ space

- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

# Compacted suffix trees

**Step one:** Add a $\$$ (unique symbol) to $T$





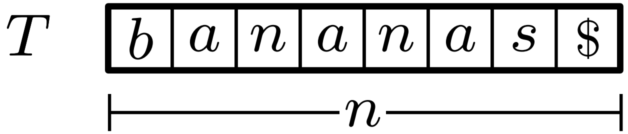
Uses $O(n)$ space

**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

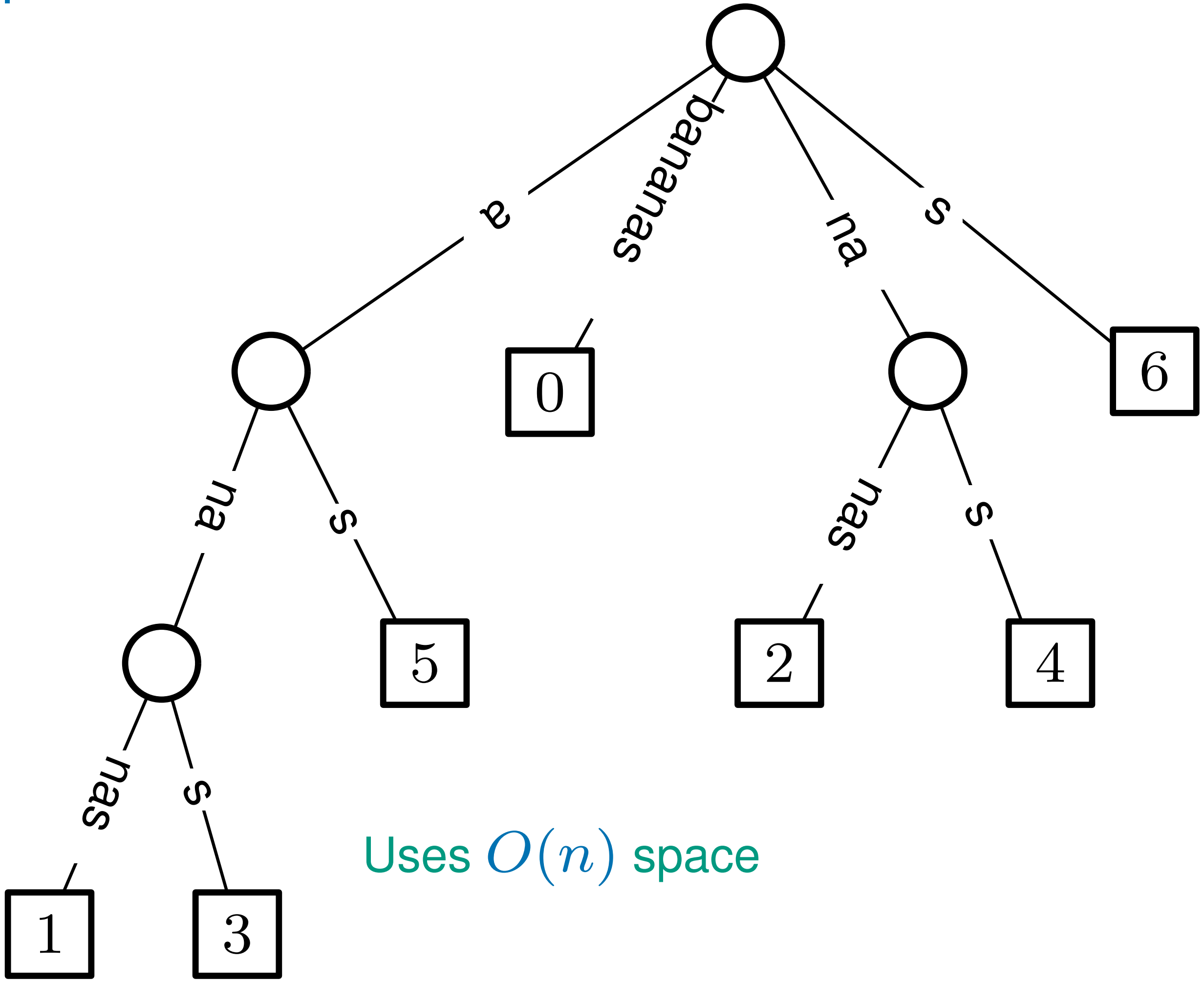
# Compacted suffix trees

**Step one:** Add a $\$$ (unique symbol) to $T$



**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring

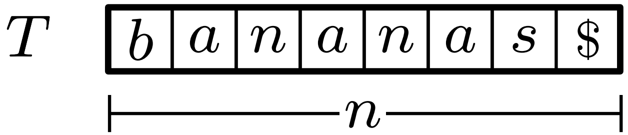


Uses $O(n)$ space

- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

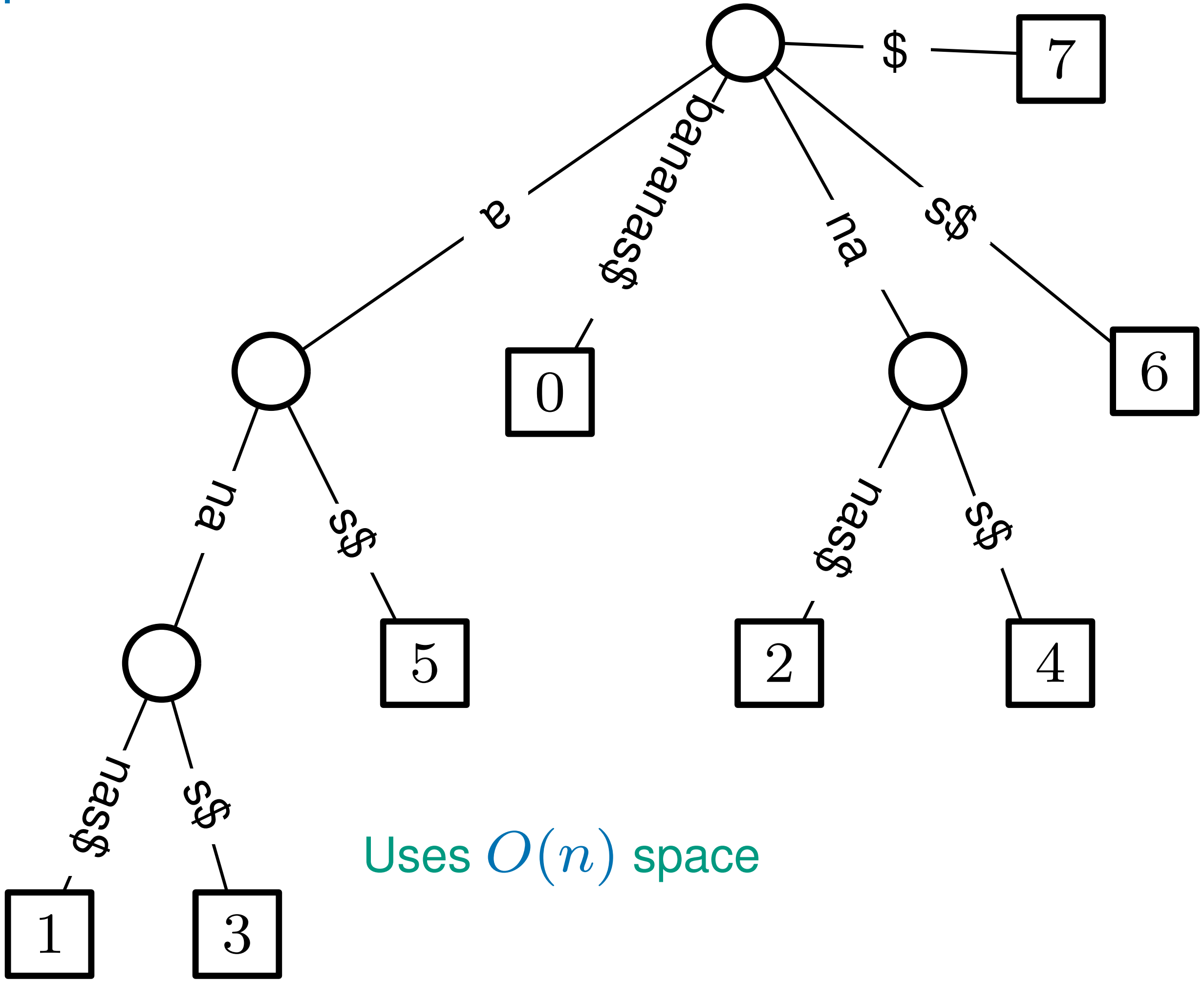
# Compacted suffix trees

**Step one:** Add a $\$$ (unique symbol) to $T$



**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring



Uses $O(n)$ space

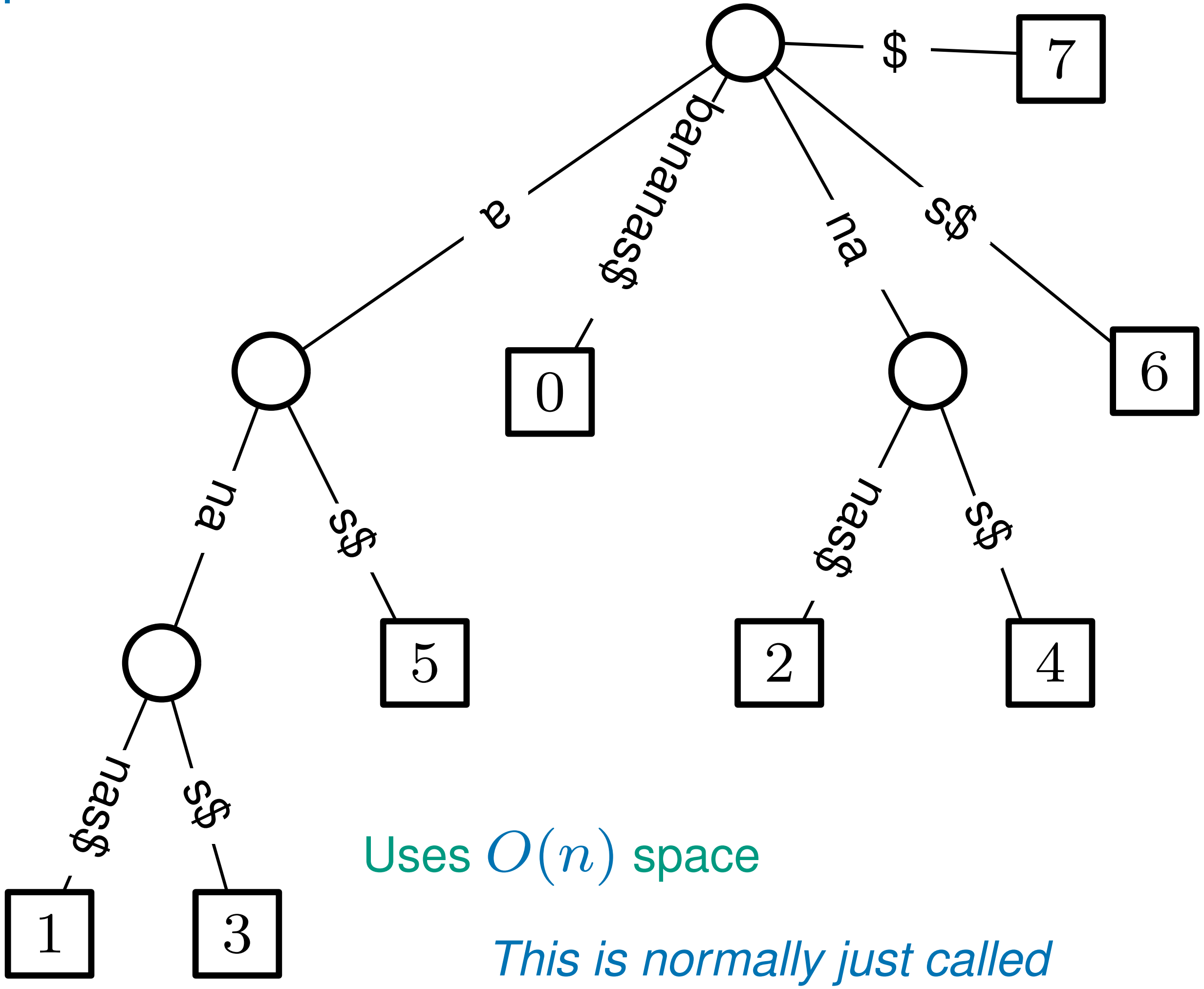
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

# Compacted suffix trees

**Step one:** Add a $\$$ (unique symbol) to $T$

$T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ | $\$$ |

$n$



**Compacted Suffix Tree** of $T$

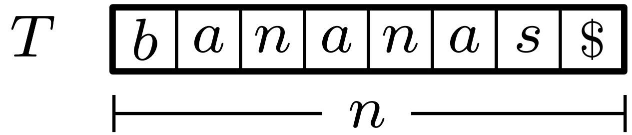
- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring

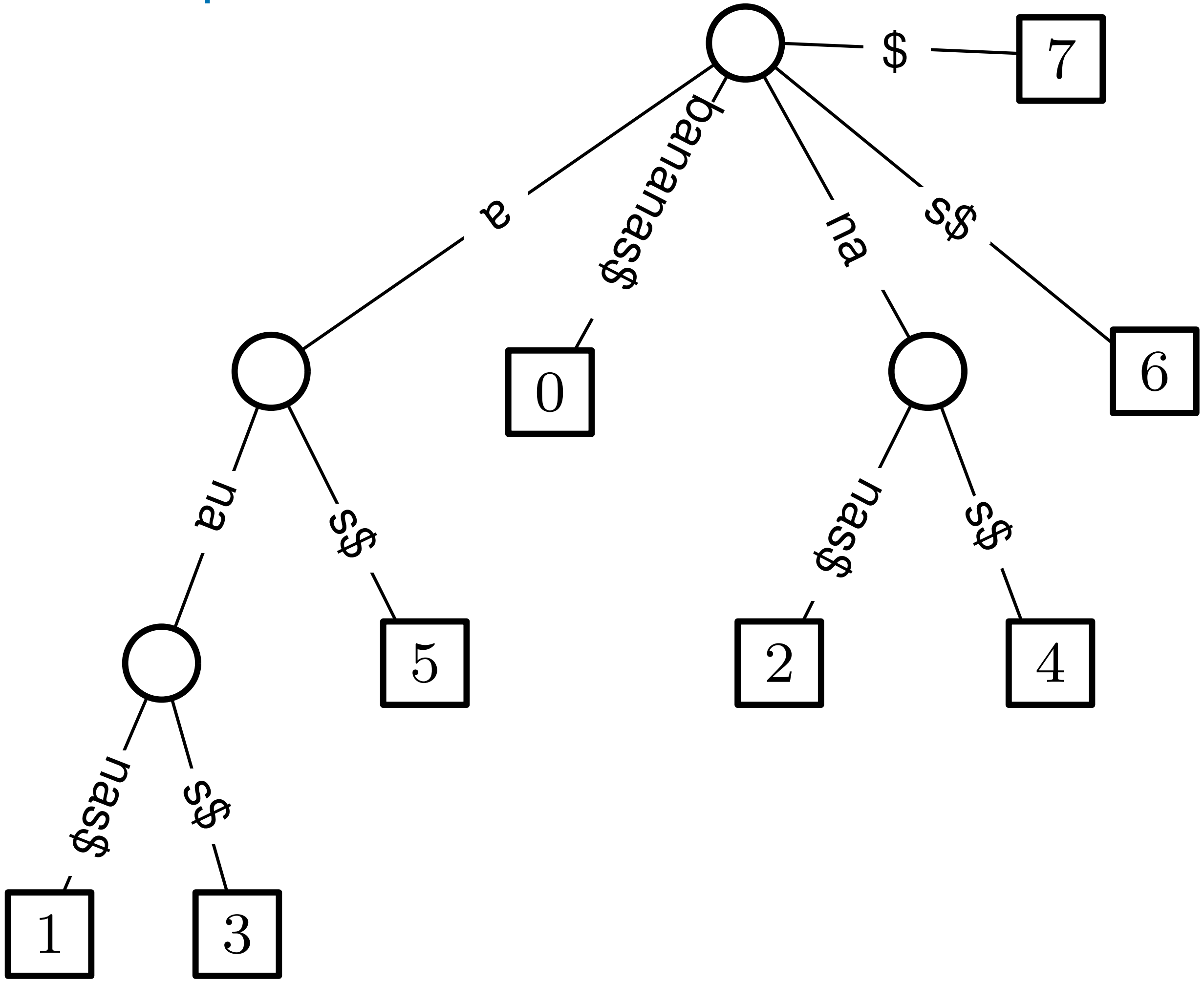
Uses $O(n)$ space

*This is normally just called a suffix tree*

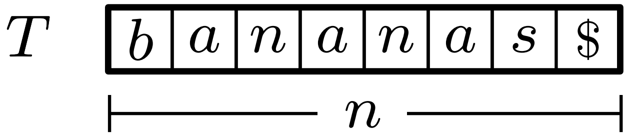
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

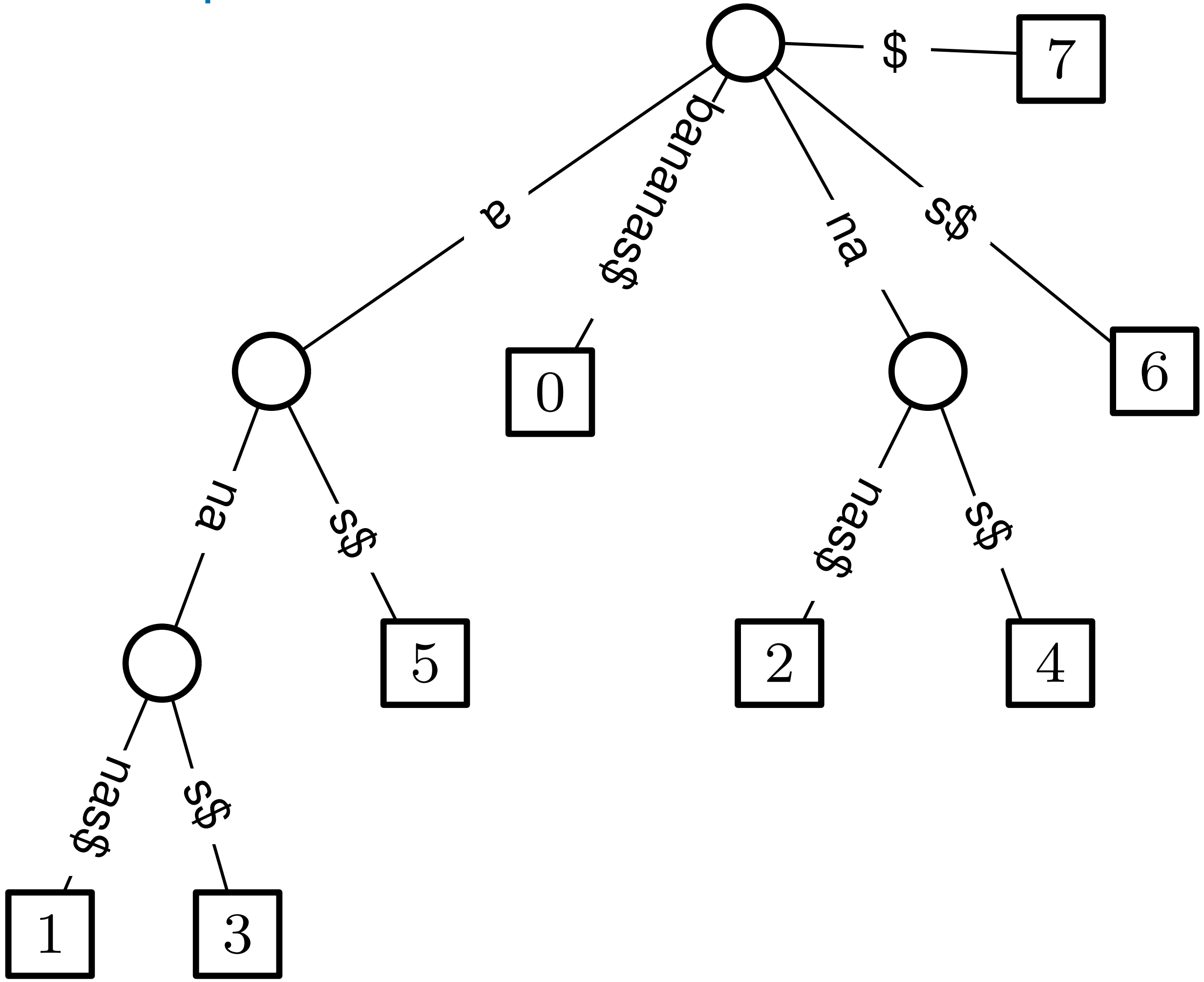
# Searching in a compacted suffix tree





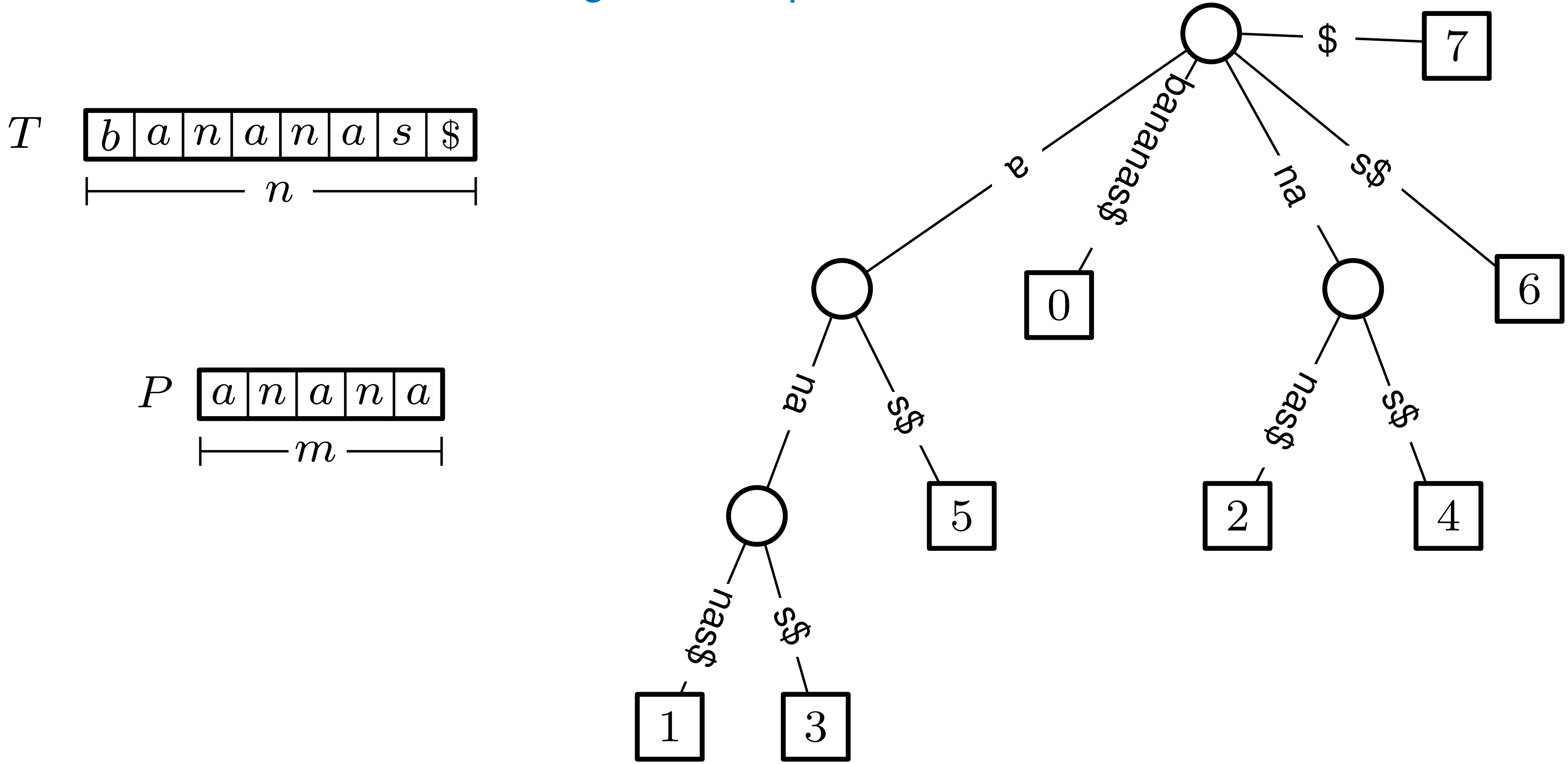
# Searching in a compacted suffix tree





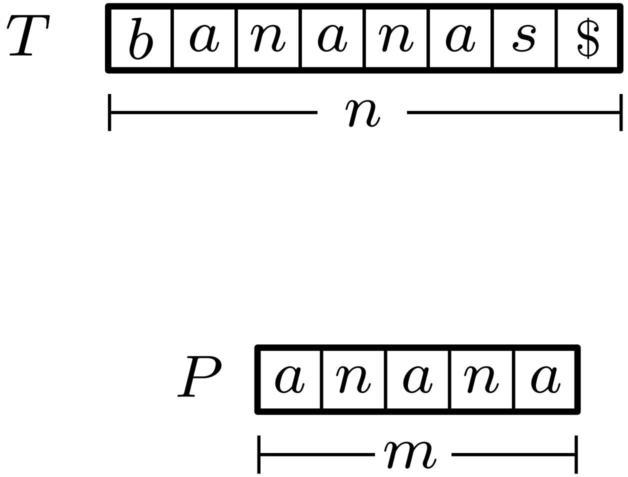
*How do you find a pattern?*

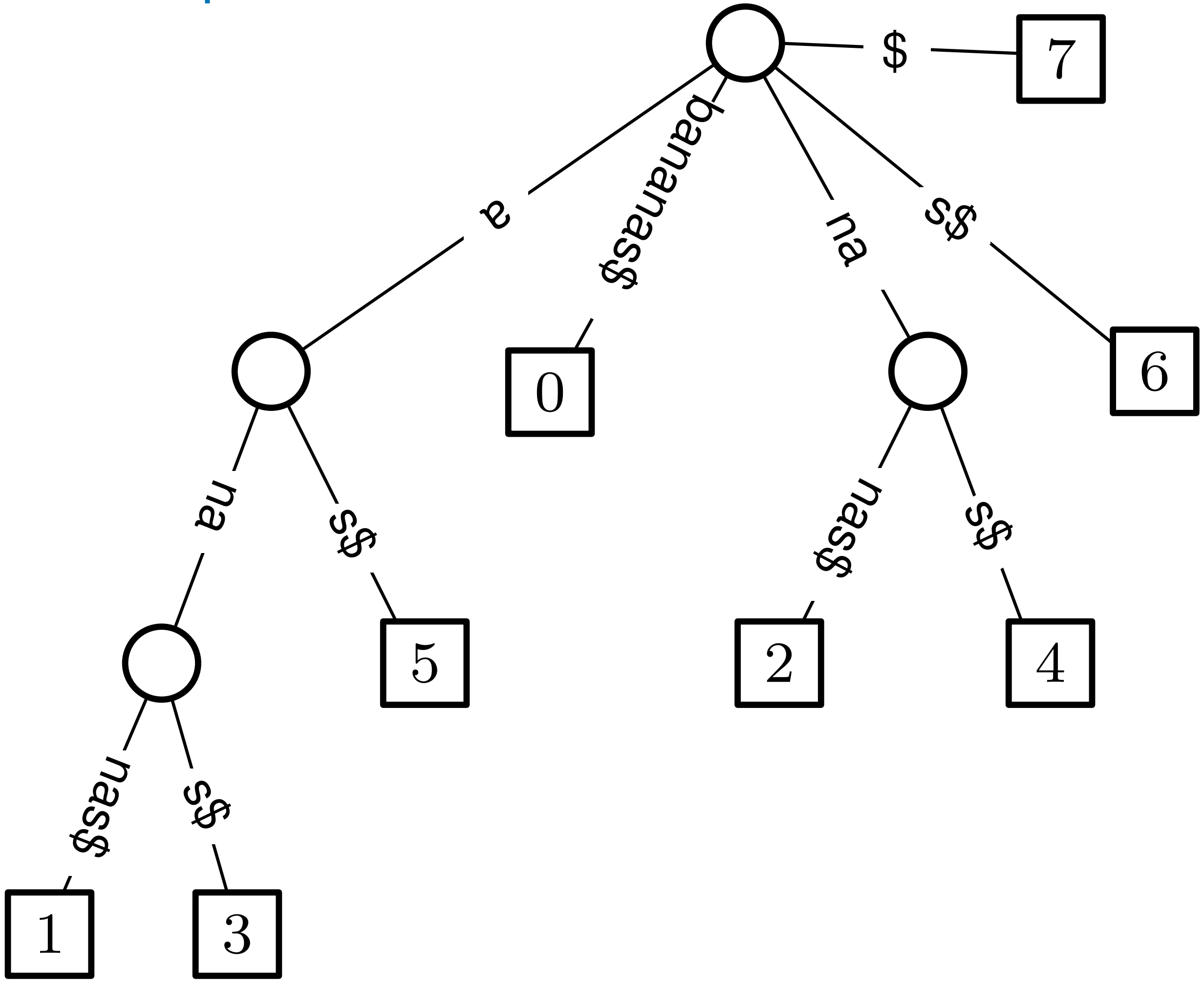
# Searching in a compacted suffix tree



*How do you find a pattern?*

# Searching in a compacted suffix tree

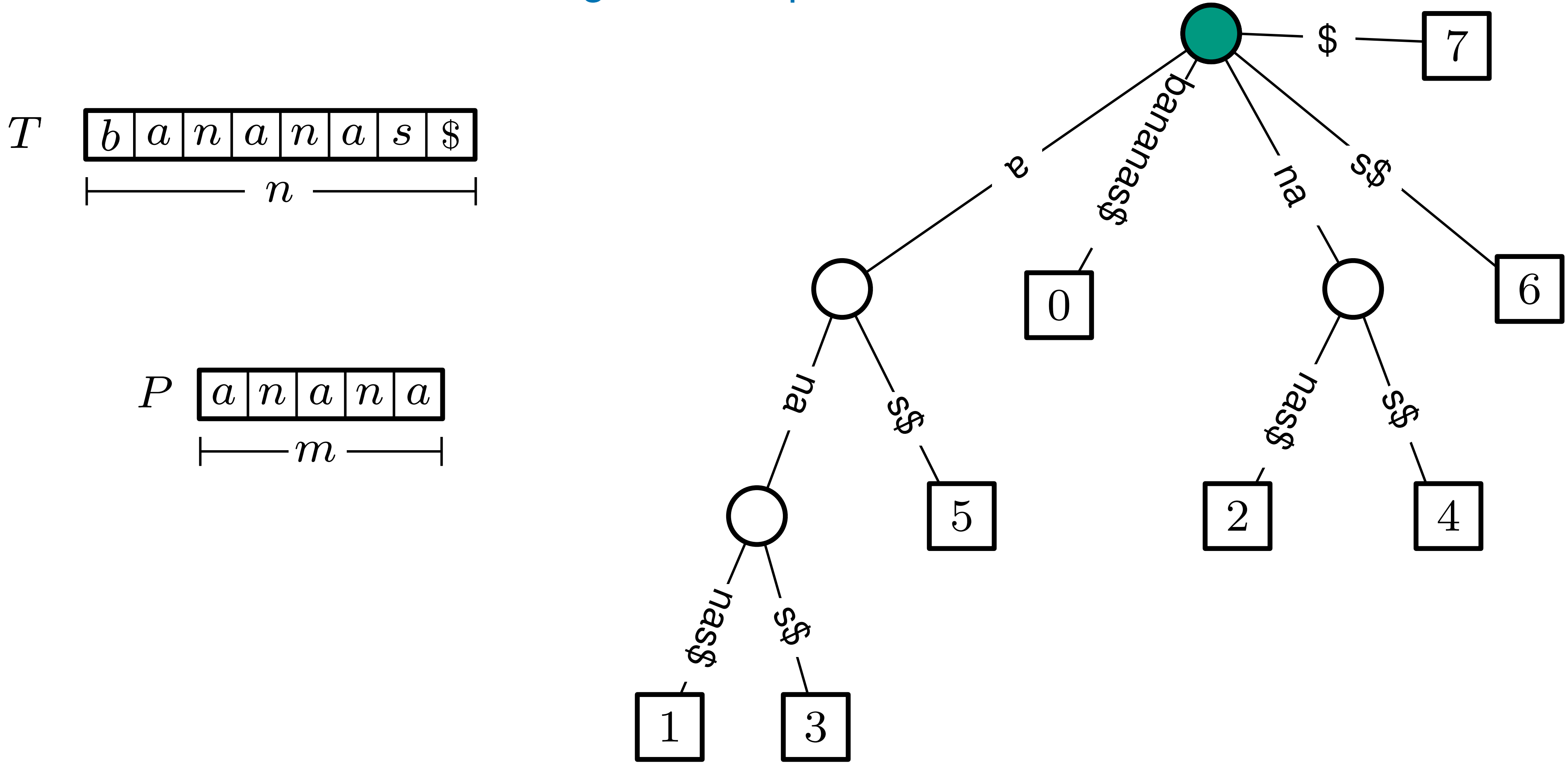




*How do you find a pattern?*

start at the root and walk down the tree

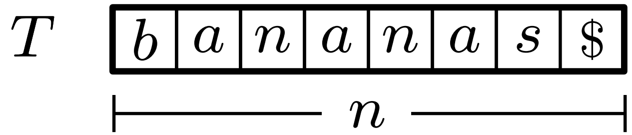
# Searching in a compacted suffix tree

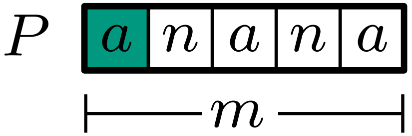


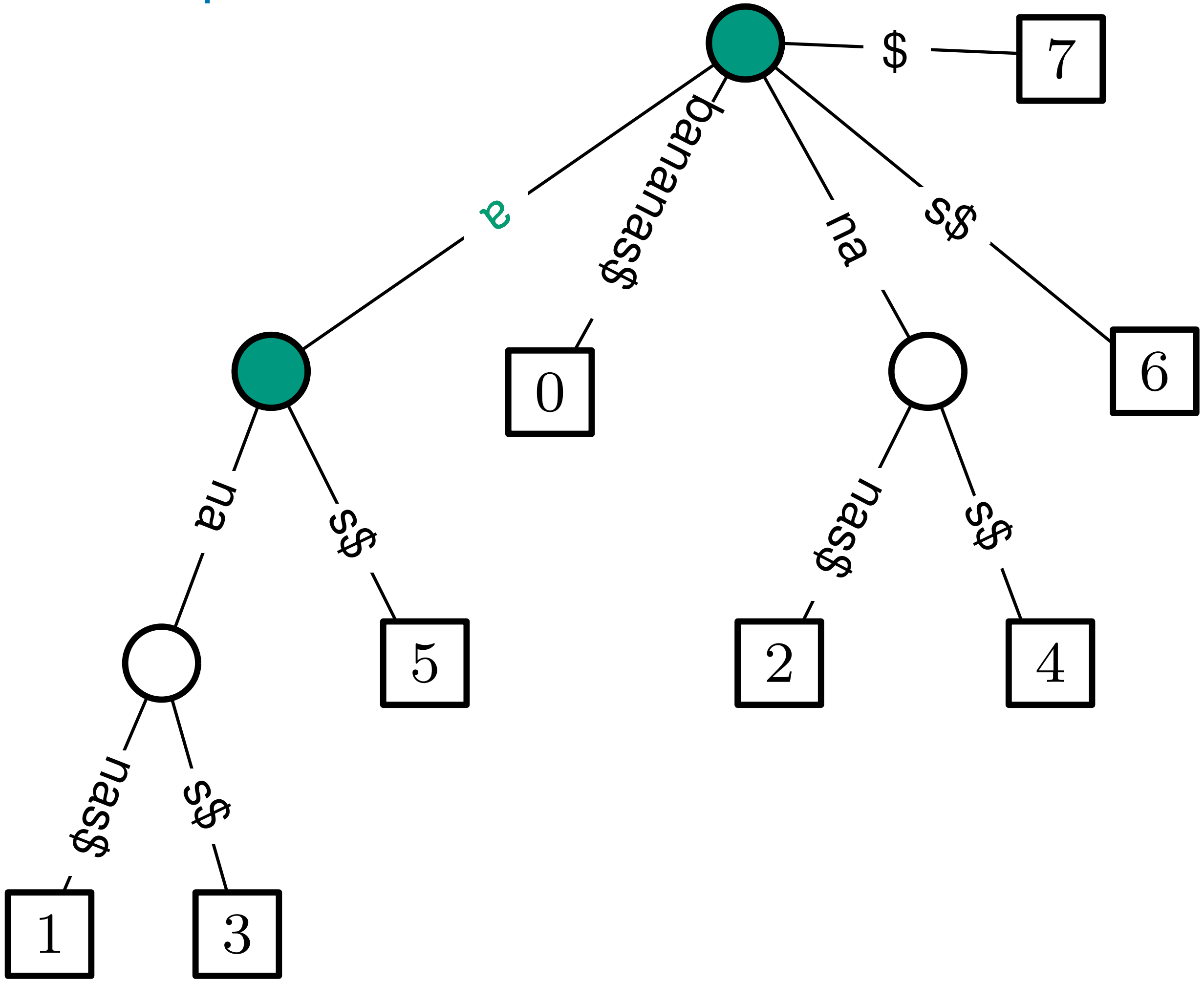
*How do you find a pattern?*

start at the root and walk down the tree

# Searching in a compacted suffix tree



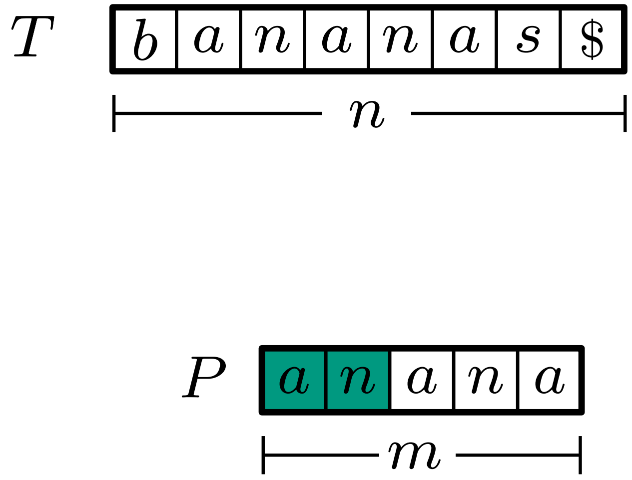


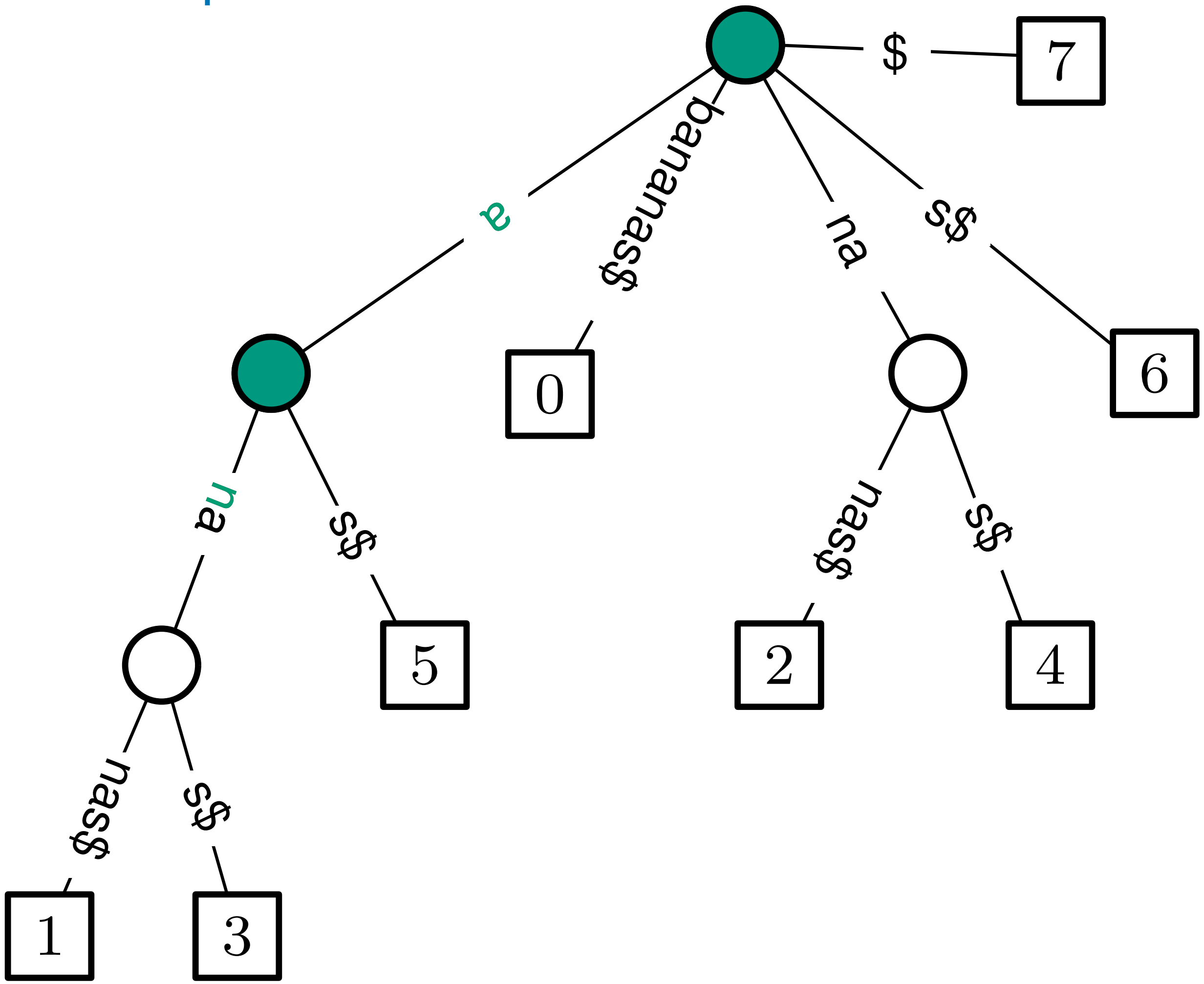


*How do you find a pattern?*

start at the root and walk down the tree

# Searching in a compacted suffix tree

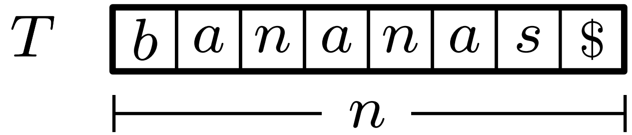


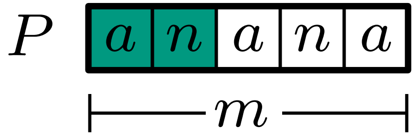


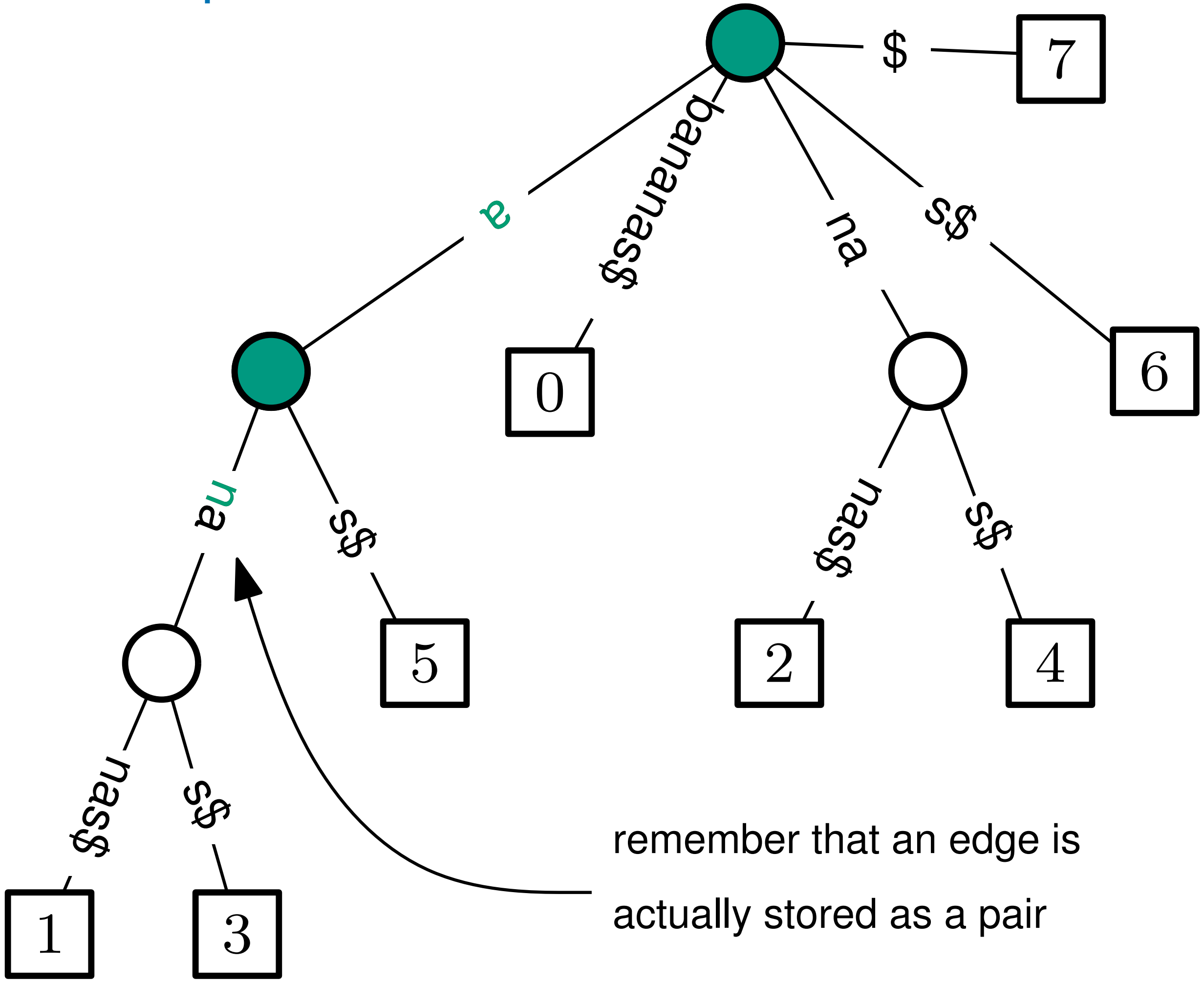
*How do you find a pattern?*

start at the root and walk down the tree

# Searching in a compacted suffix tree







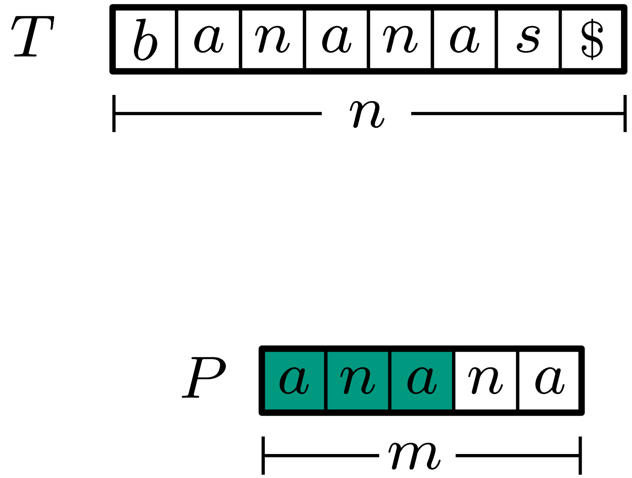
remember that an edge is actually stored as a pair

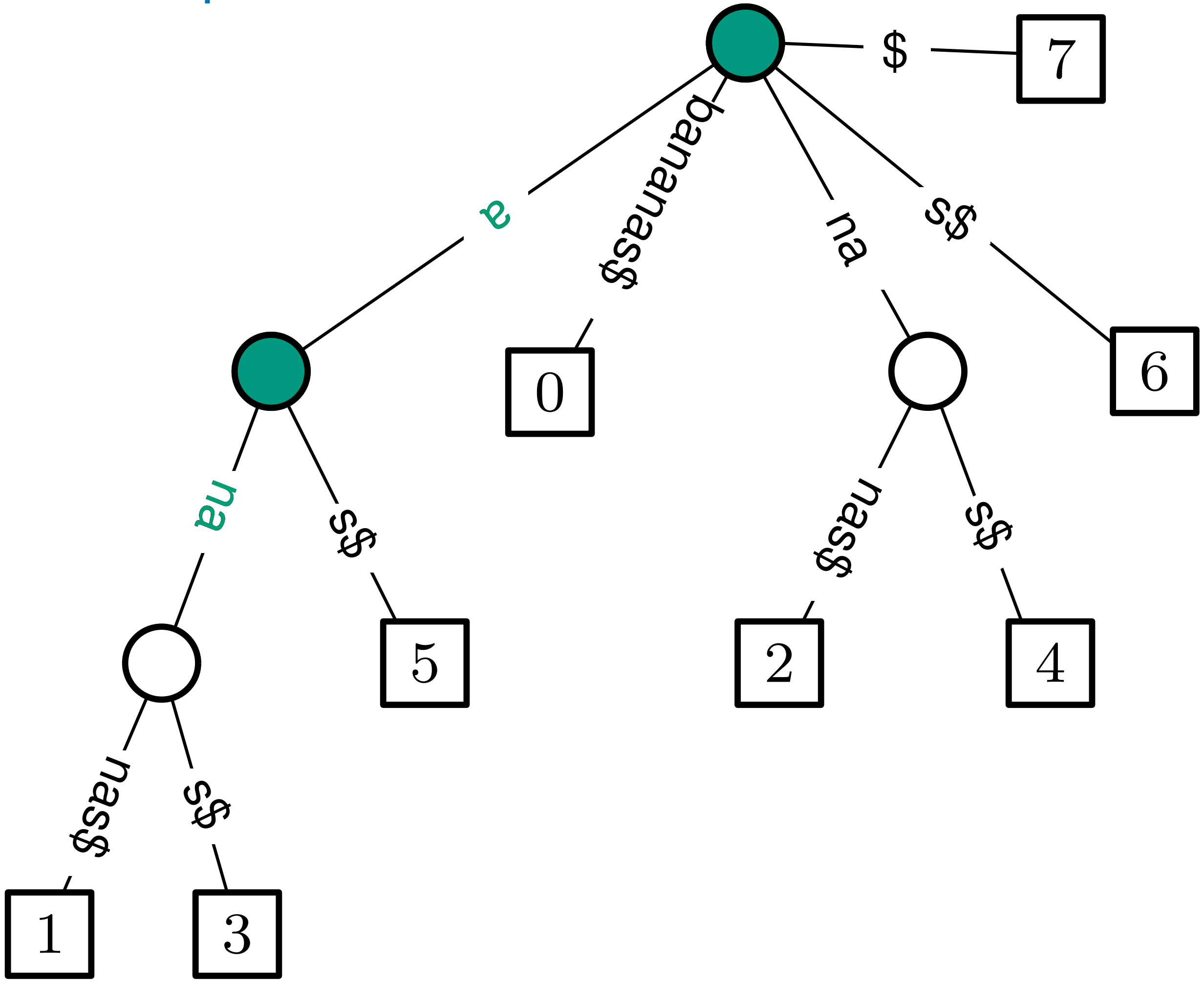
*we're actually looking in $T$*

*How do you find a pattern?*

start at the root and walk down the tree

# Searching in a compacted suffix tree

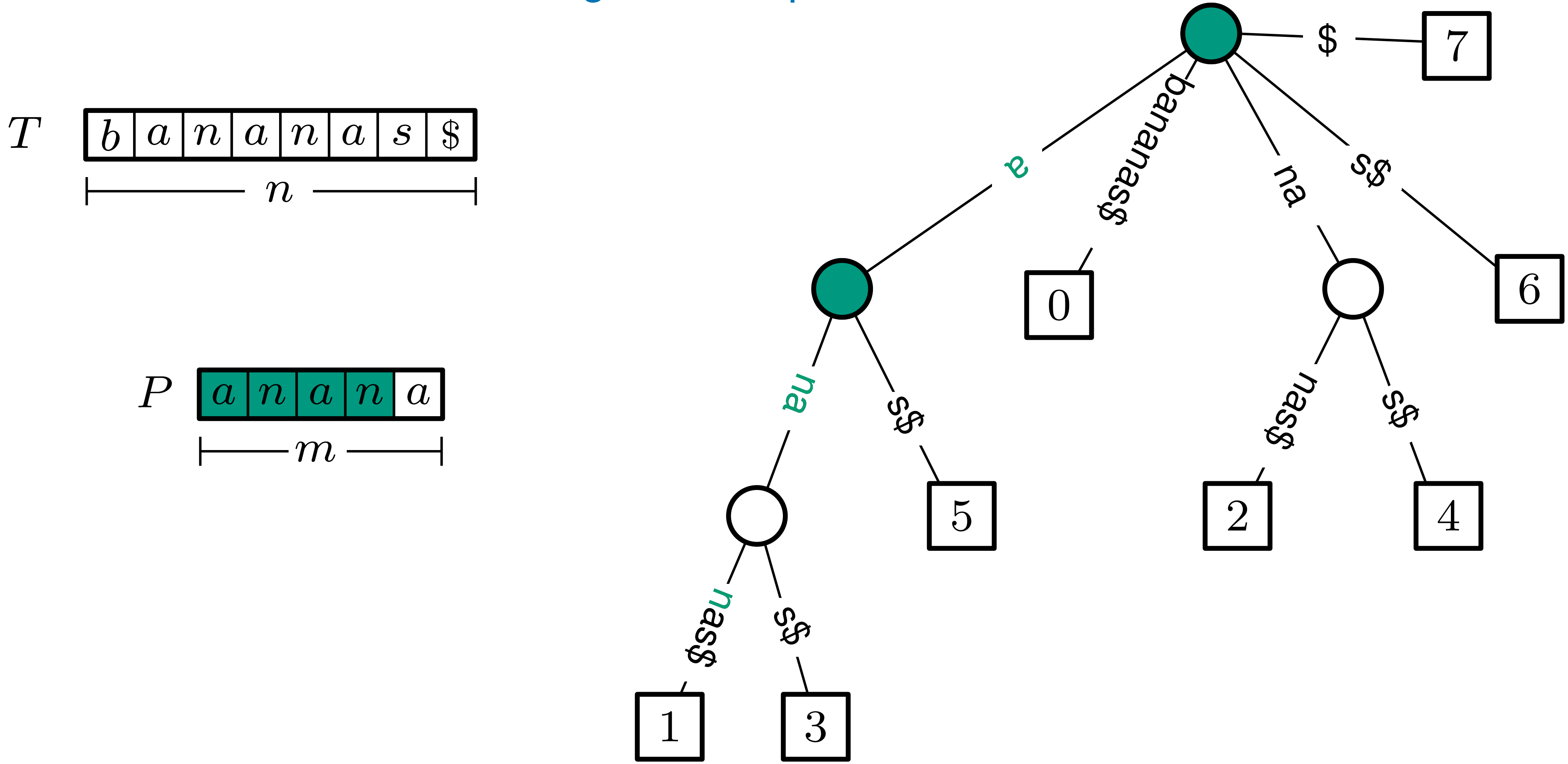




*How do you find a pattern?*

start at the root and walk down the tree

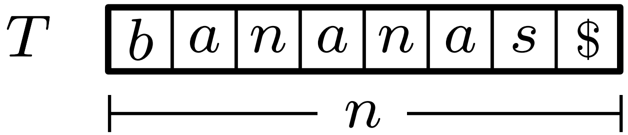
# Searching in a compacted suffix tree

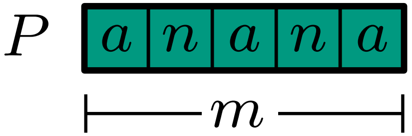


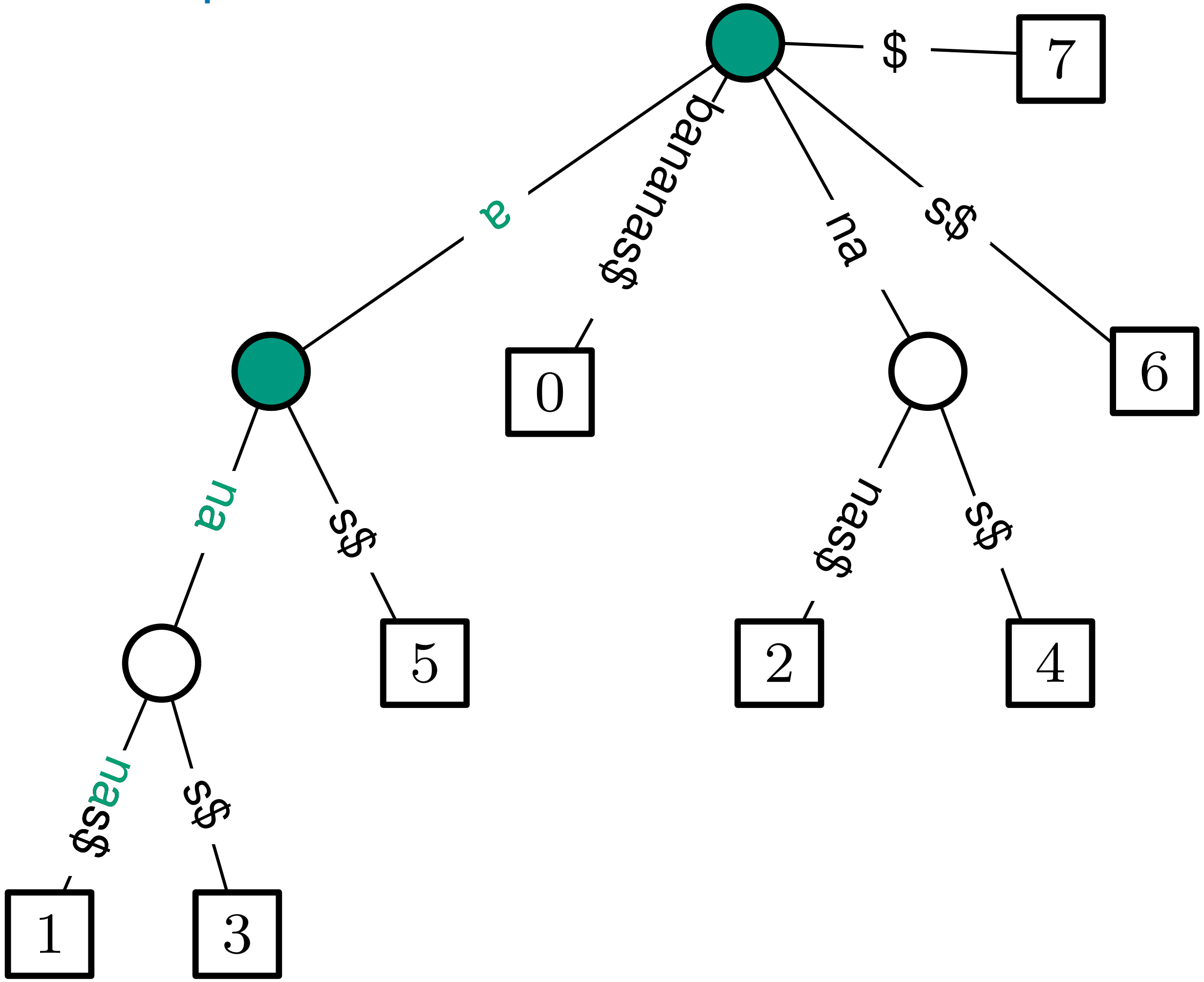
*How do you find a pattern?*

start at the root and walk down the tree

# Searching in a compacted suffix tree



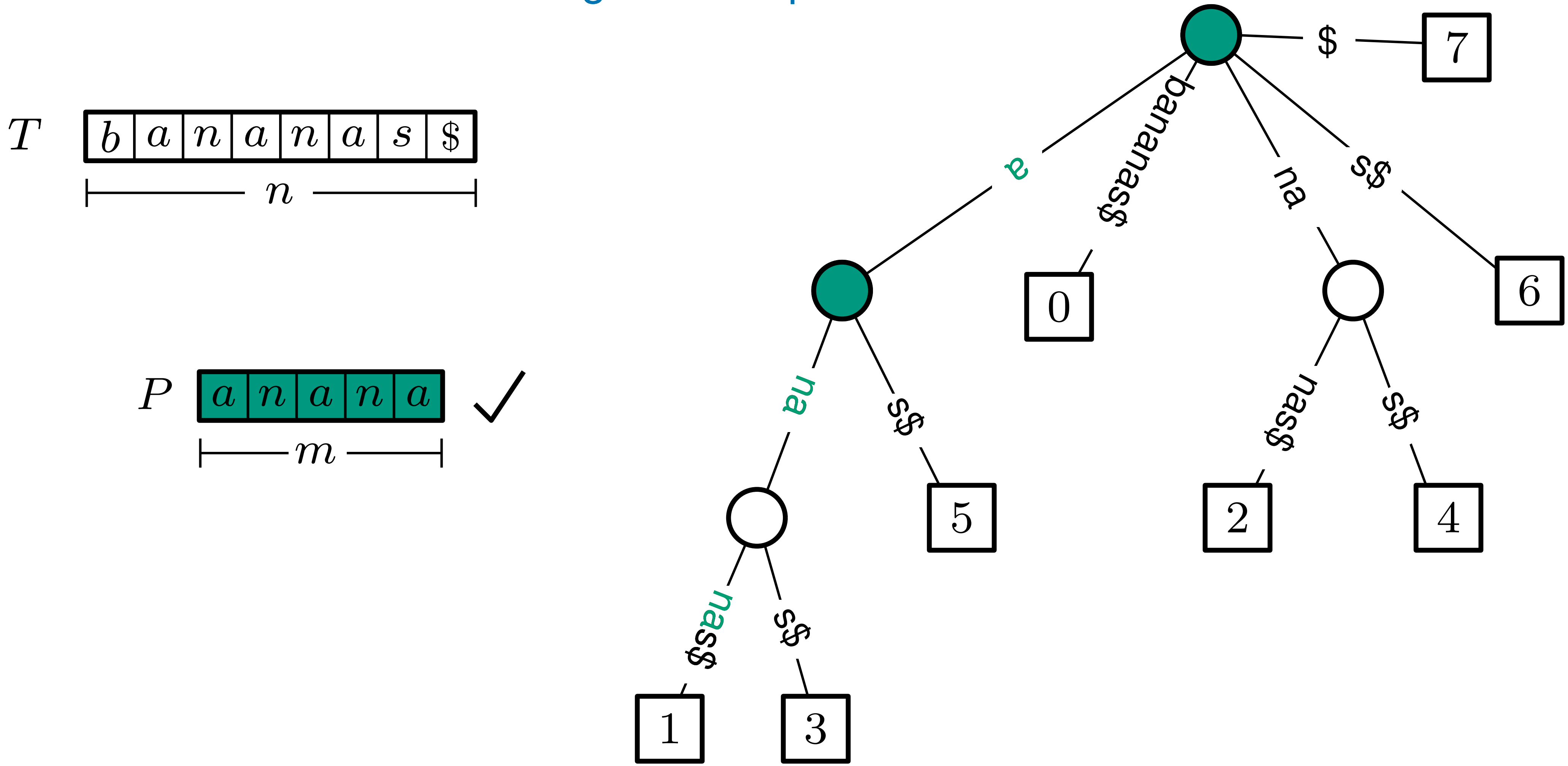




*How do you find a pattern?*

start at the root and walk down the tree

# Searching in a compacted suffix tree

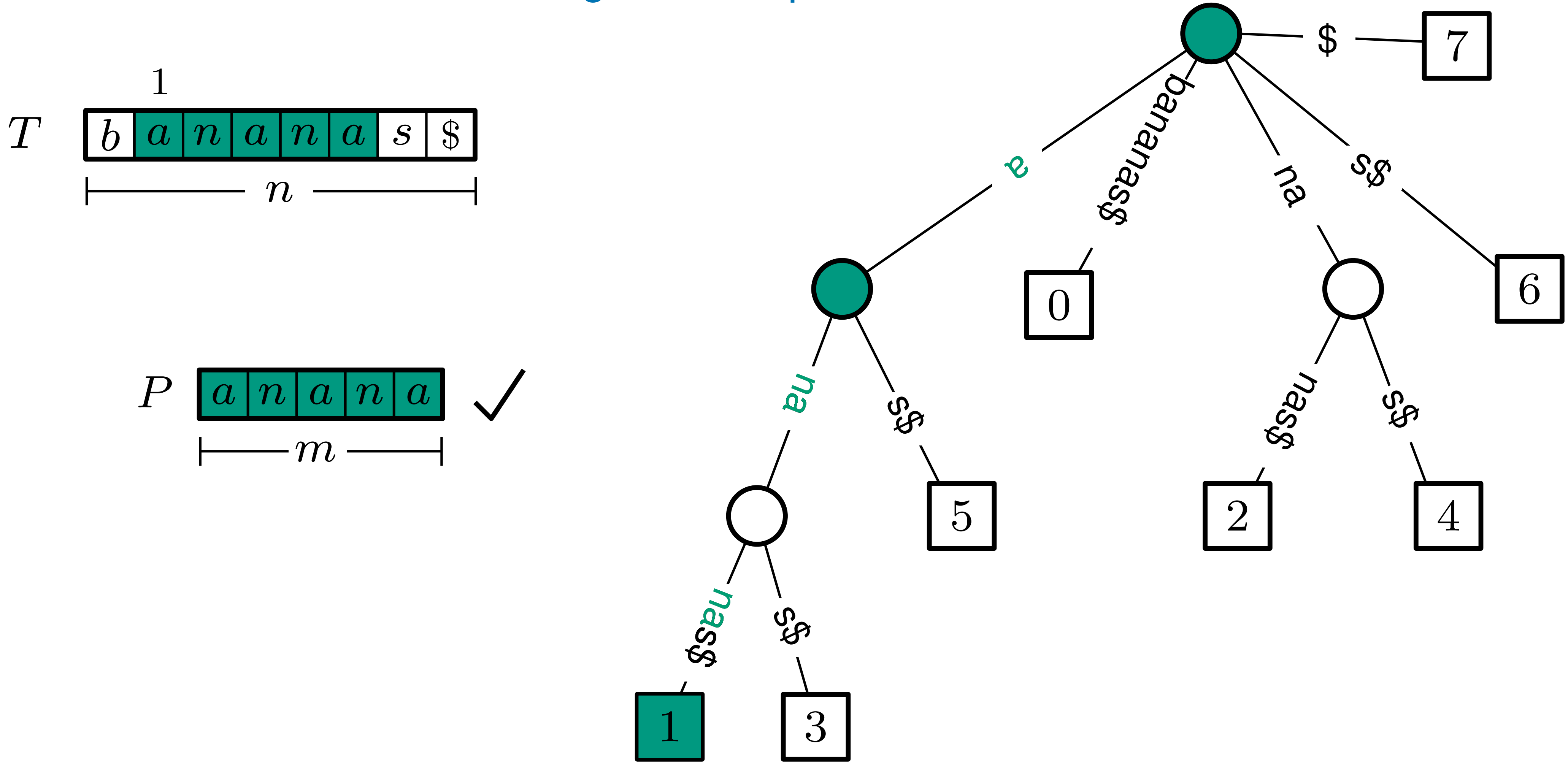


*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in a compacted suffix tree

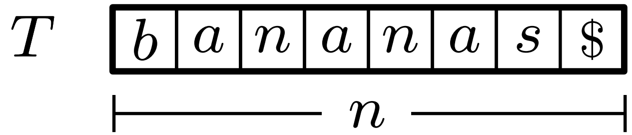


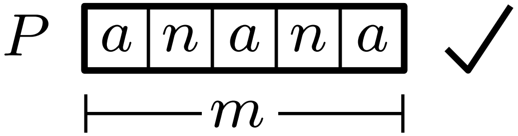
*How do you find a pattern?*

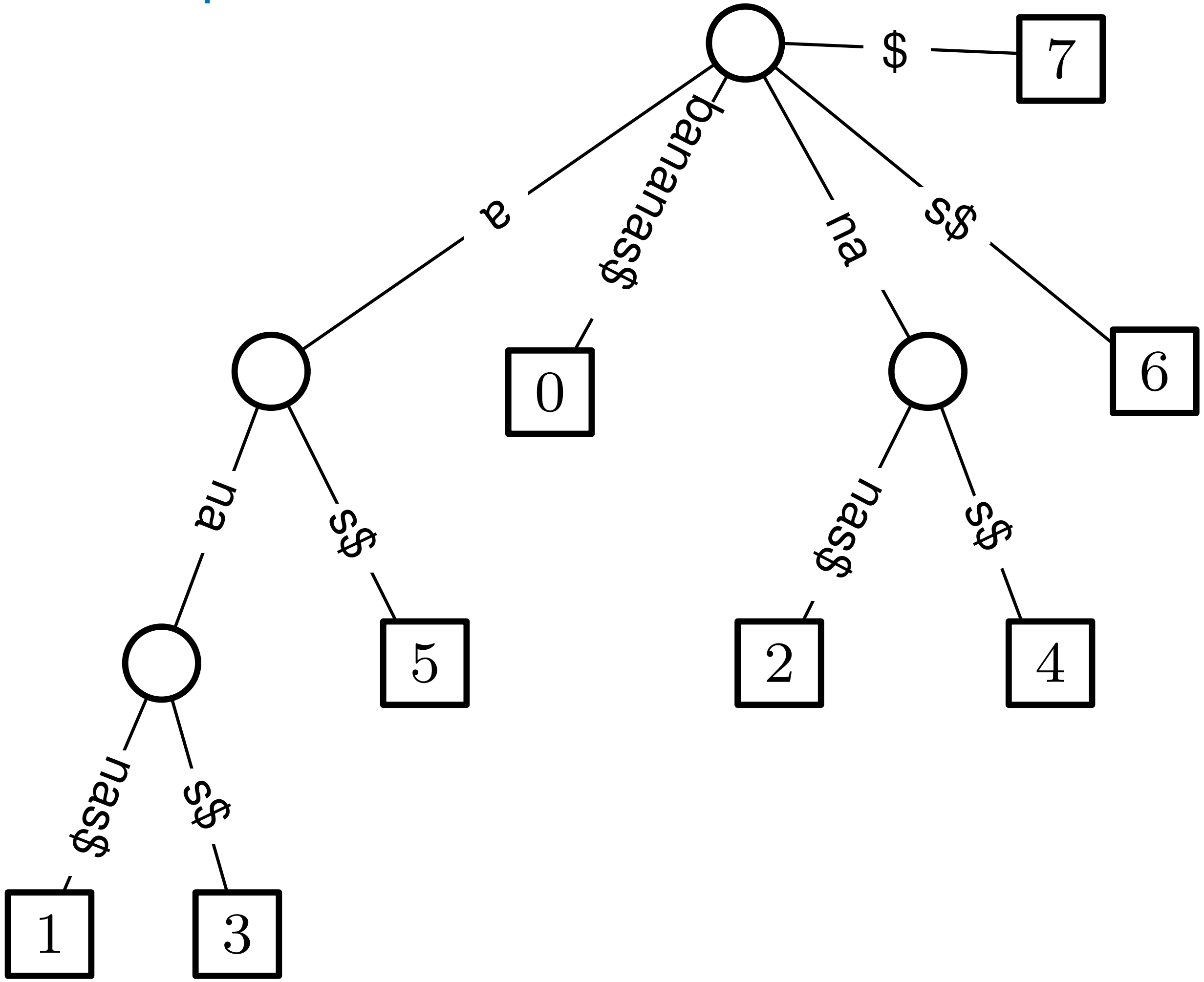
start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in a compacted suffix tree







*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in a compacted suffix tree

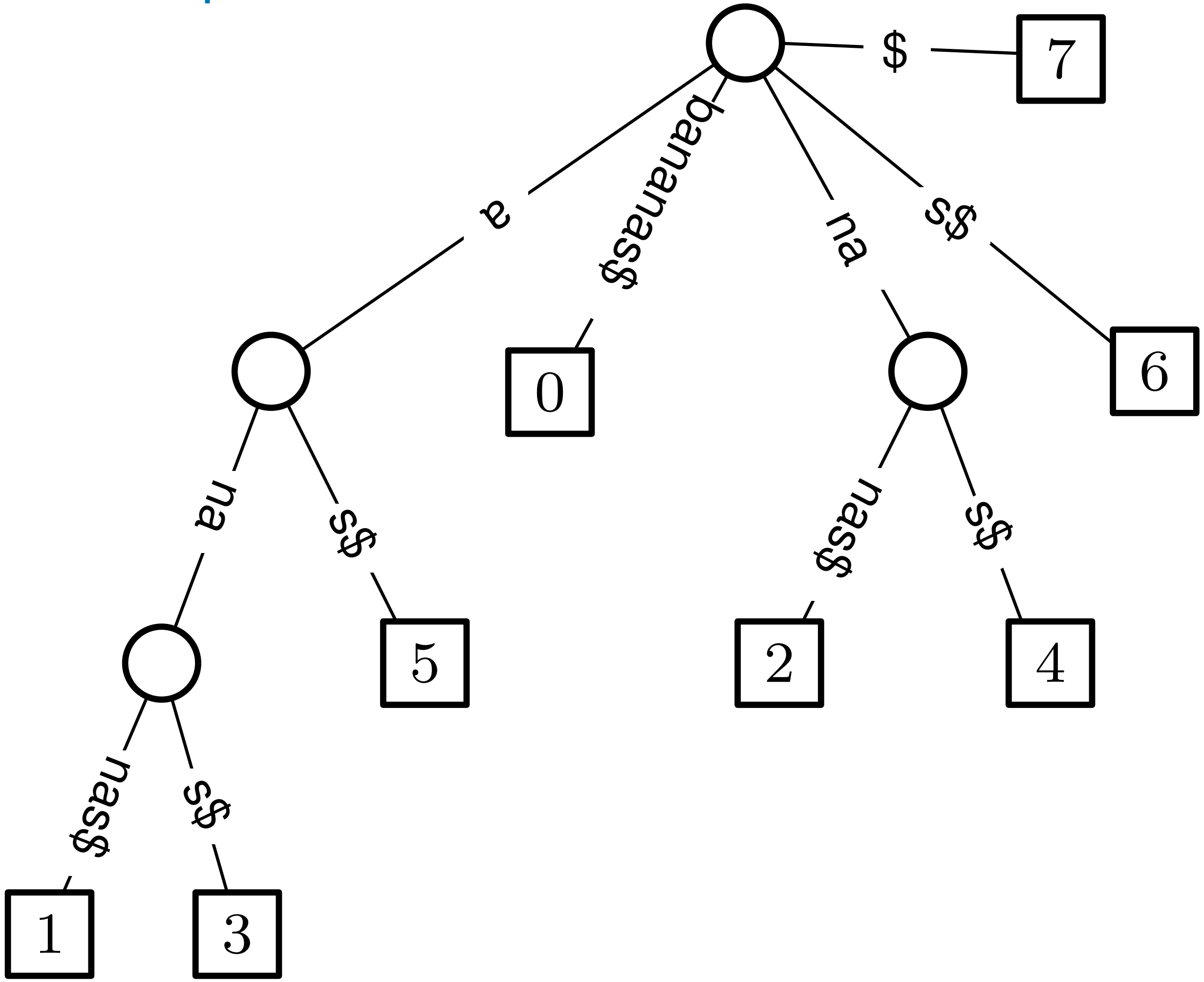
$T$ $\boxed{b\,|\,a\,|\,n\,|\,a\,|\,n\,|\,a\,|\,s\,|\,\$}$

$\vdash\!\!-\!\!-\; n \;-\!\!-\!\!\dashv$

$P$ $\boxed{a\,|\,n\,|\,a\,|\,n\,|\,a}$ ✓

$\vdash\!\!-\!\!-\; m \;-\!\!-\!\!\dashv$

$P'$ $\boxed{n\,|\,a}$



*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in a compacted suffix tree

$T$

|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ | $\$$ |

$\vdash n \dashv$

$P$

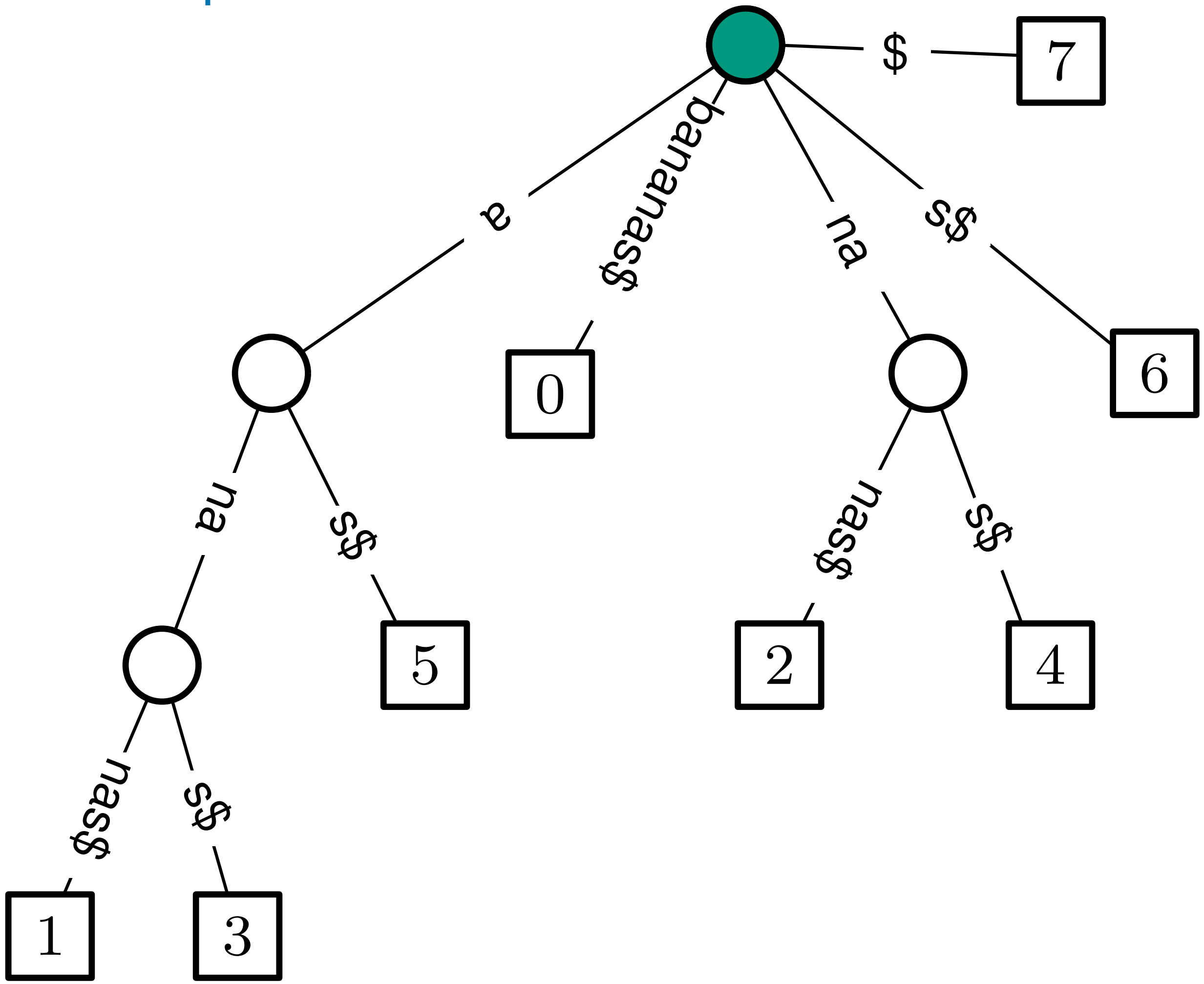
|  |  |  |  |  |
|---|---|---|---|---|
| $a$ | $n$ | $a$ | $n$ | $a$ |

✓

$\vdash m \dashv$

$P'$

|  |  |
|---|---|
| $n$ | $a$ |

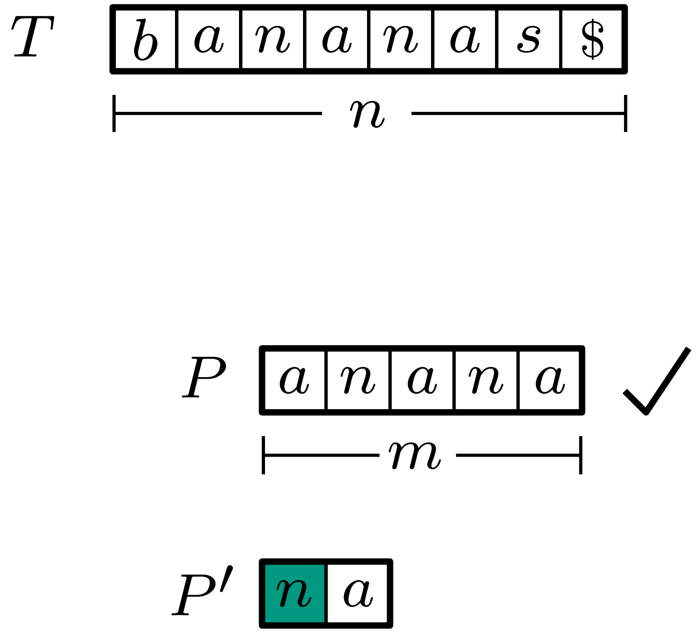


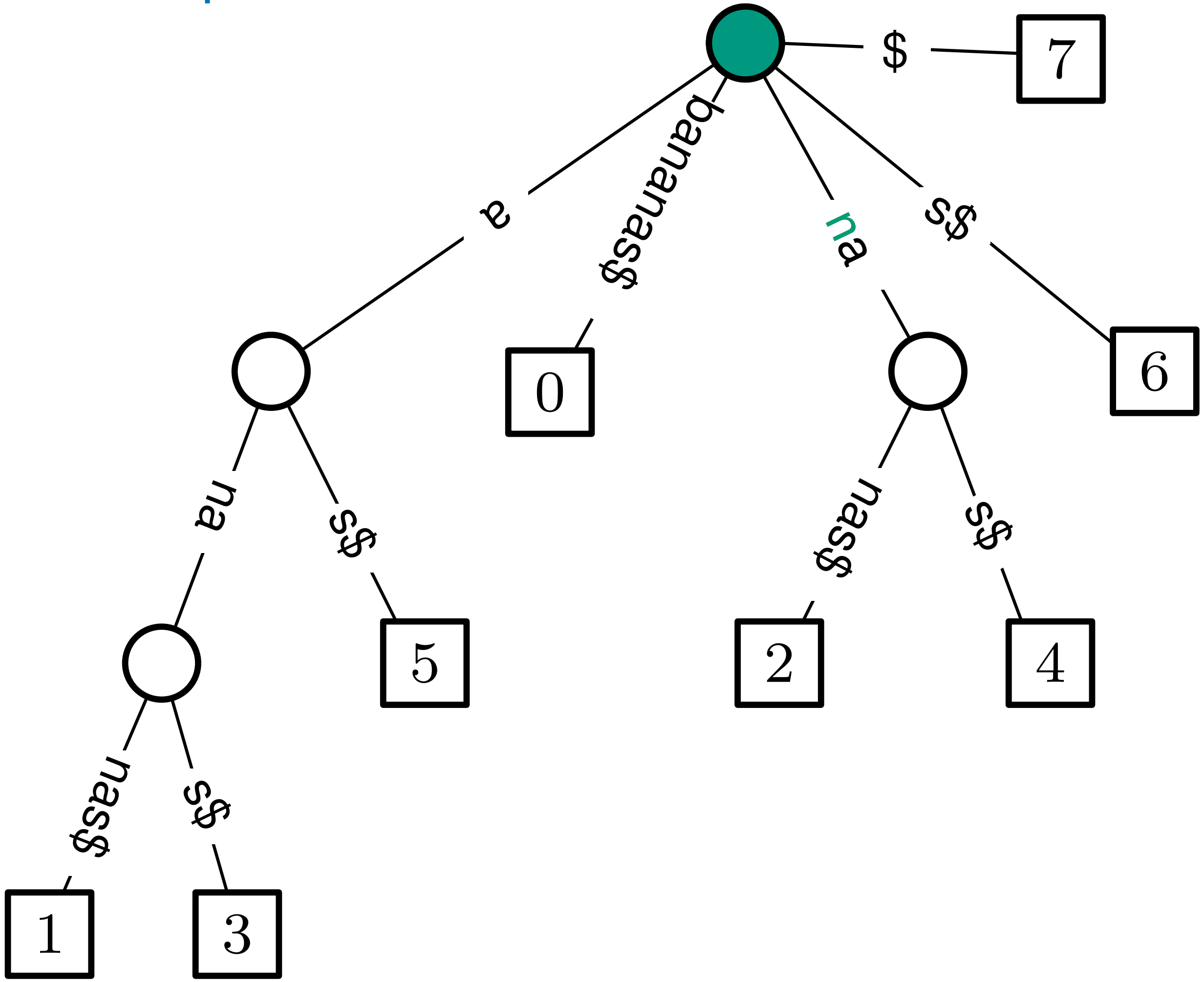
*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in a compacted suffix tree



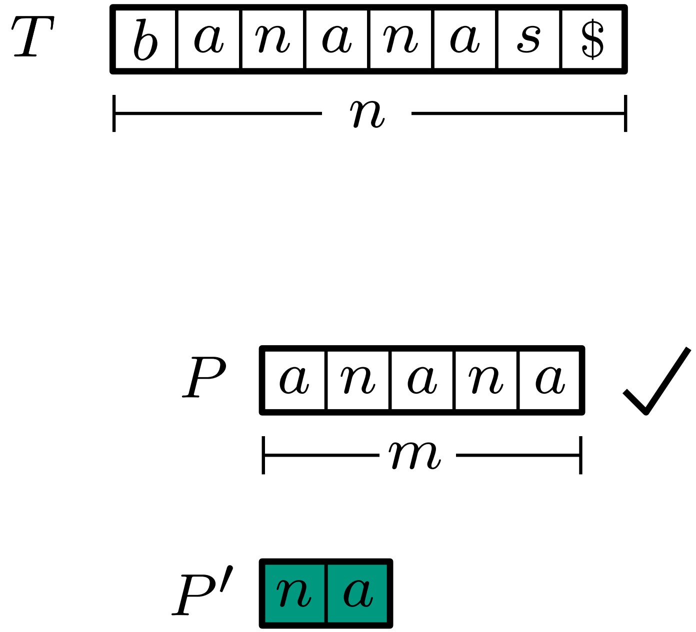


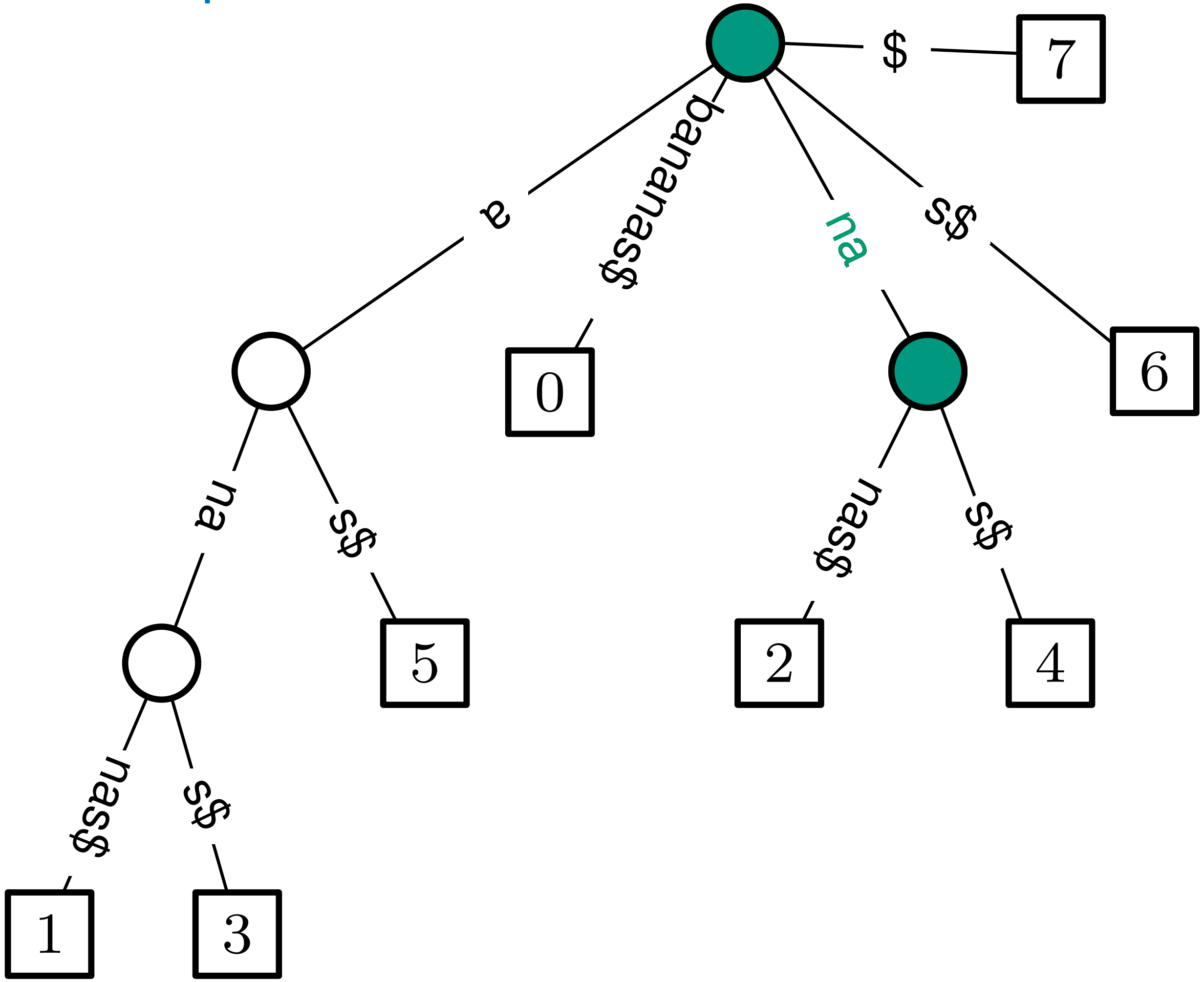
*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in a compacted suffix tree





*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in a compacted suffix tree

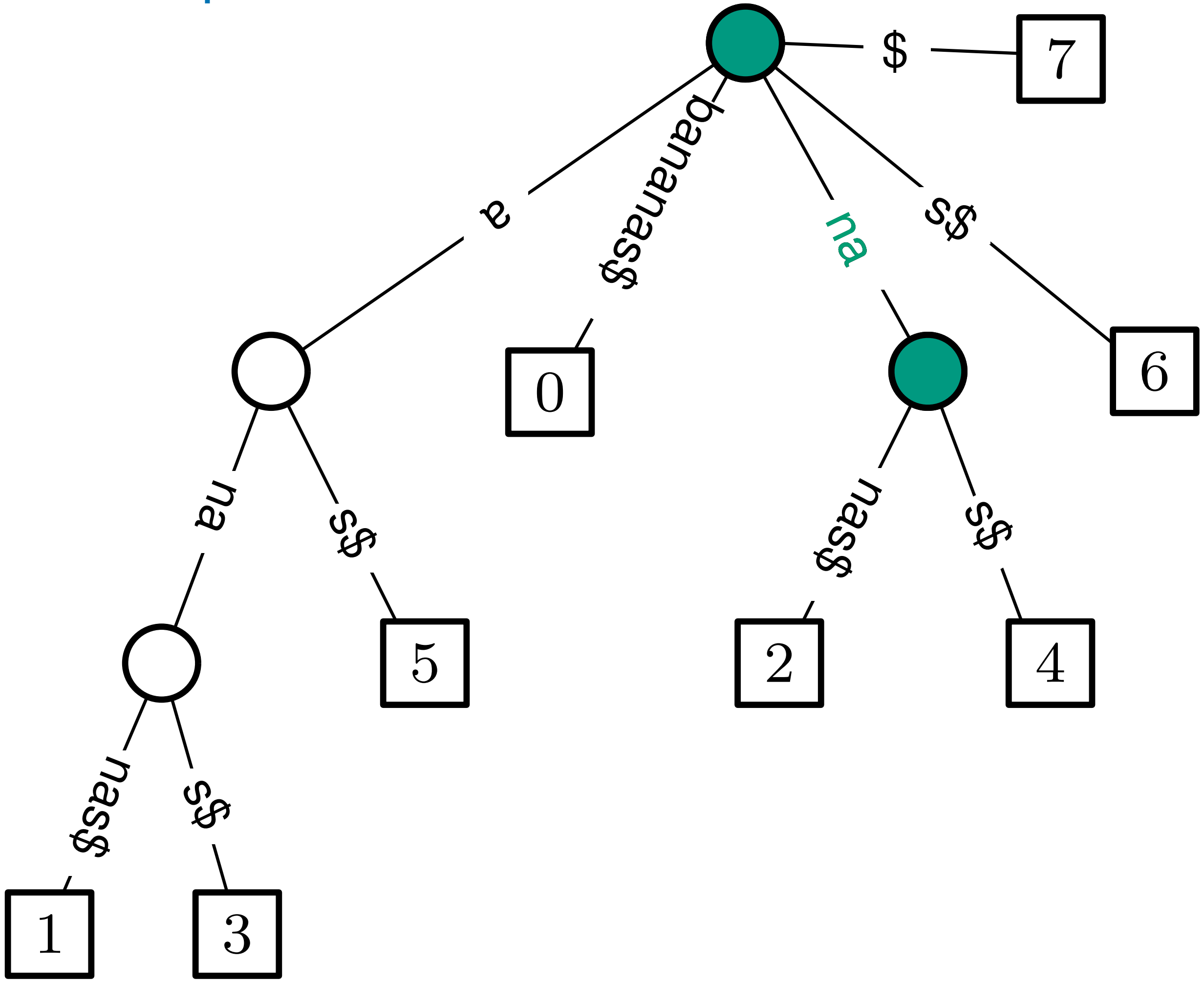
|  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ | $\$$ |

$\vdash n \dashv$

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| $P$ | $a$ | $n$ | $a$ | $n$ | $a$ | ✓ |

$\vdash m \dashv$

|  |  |  |  |
|---|---|---|---|
| $P'$ | $n$ | $a$ | ✓ |



*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in a compacted suffix tree

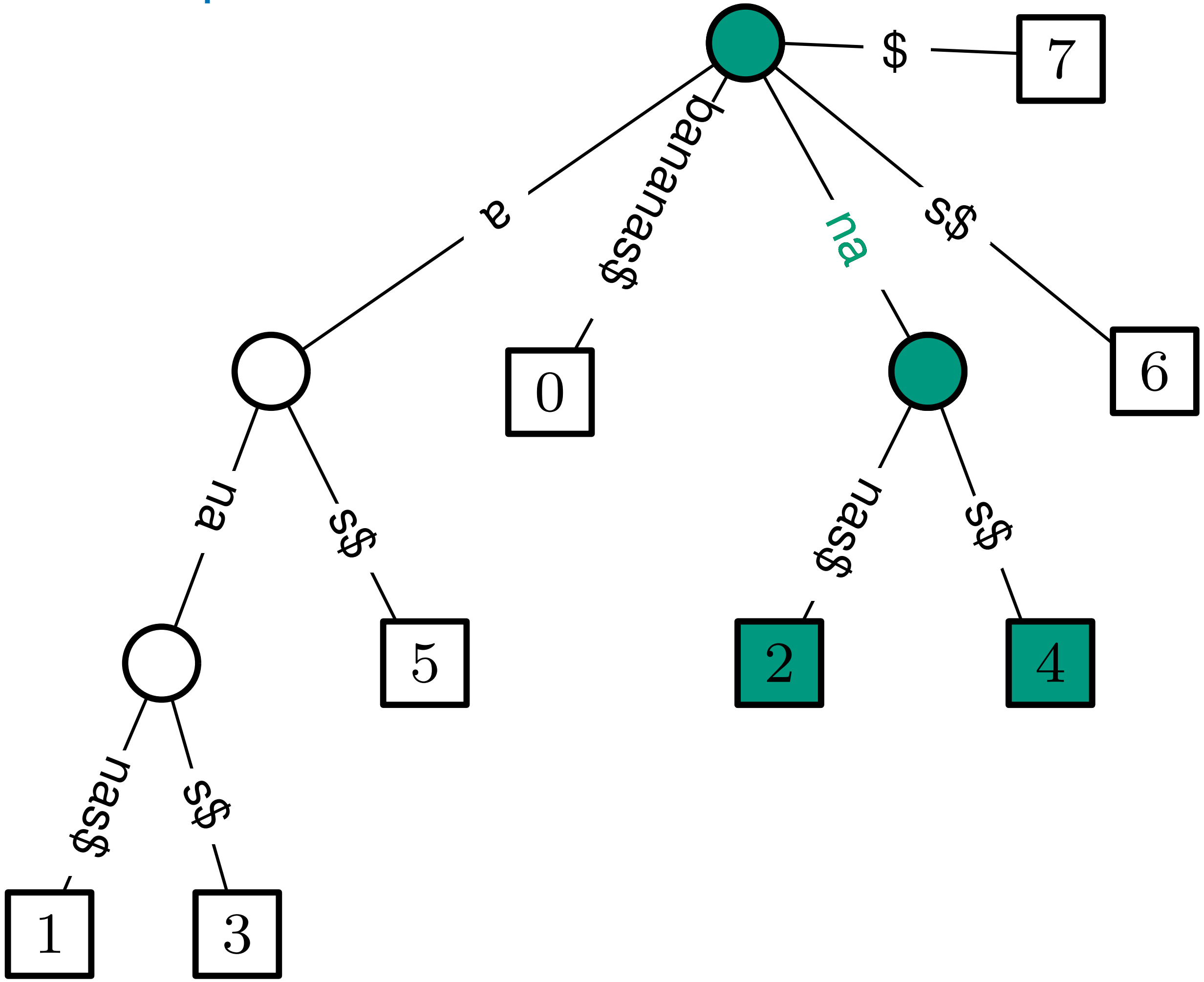
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ | $\$$ |

$\longleftarrow n \longrightarrow$

| | | | | | | |
|---|---|---|---|---|---|---|
| $P$ | $a$ | $n$ | $a$ | $n$ | $a$ | ✓ |

$\longleftarrow m \longrightarrow$

| | | | |
|---|---|---|---|
| $P'$ | $n$ | $a$ | ✓ |

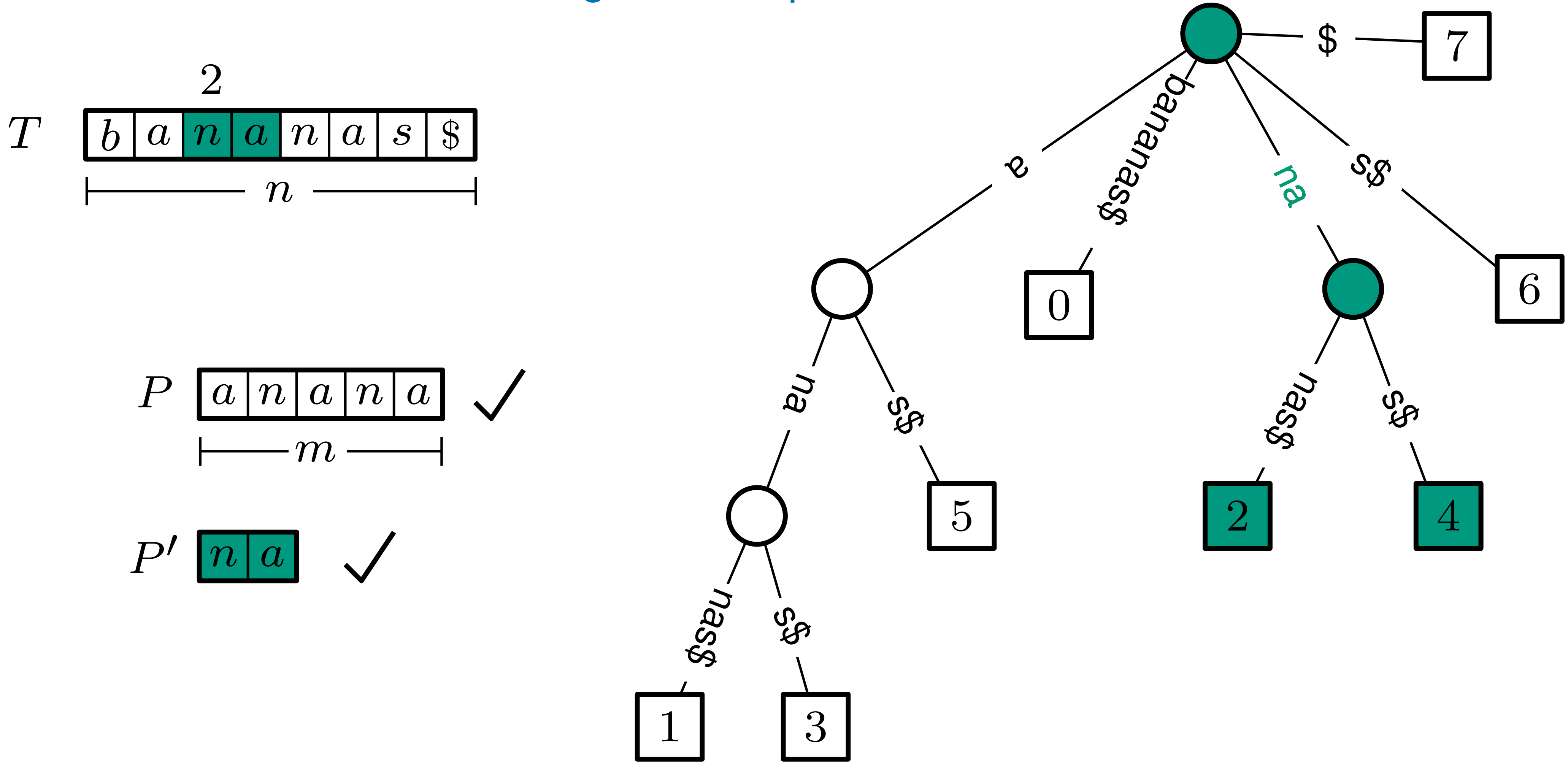


*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in a compacted suffix tree

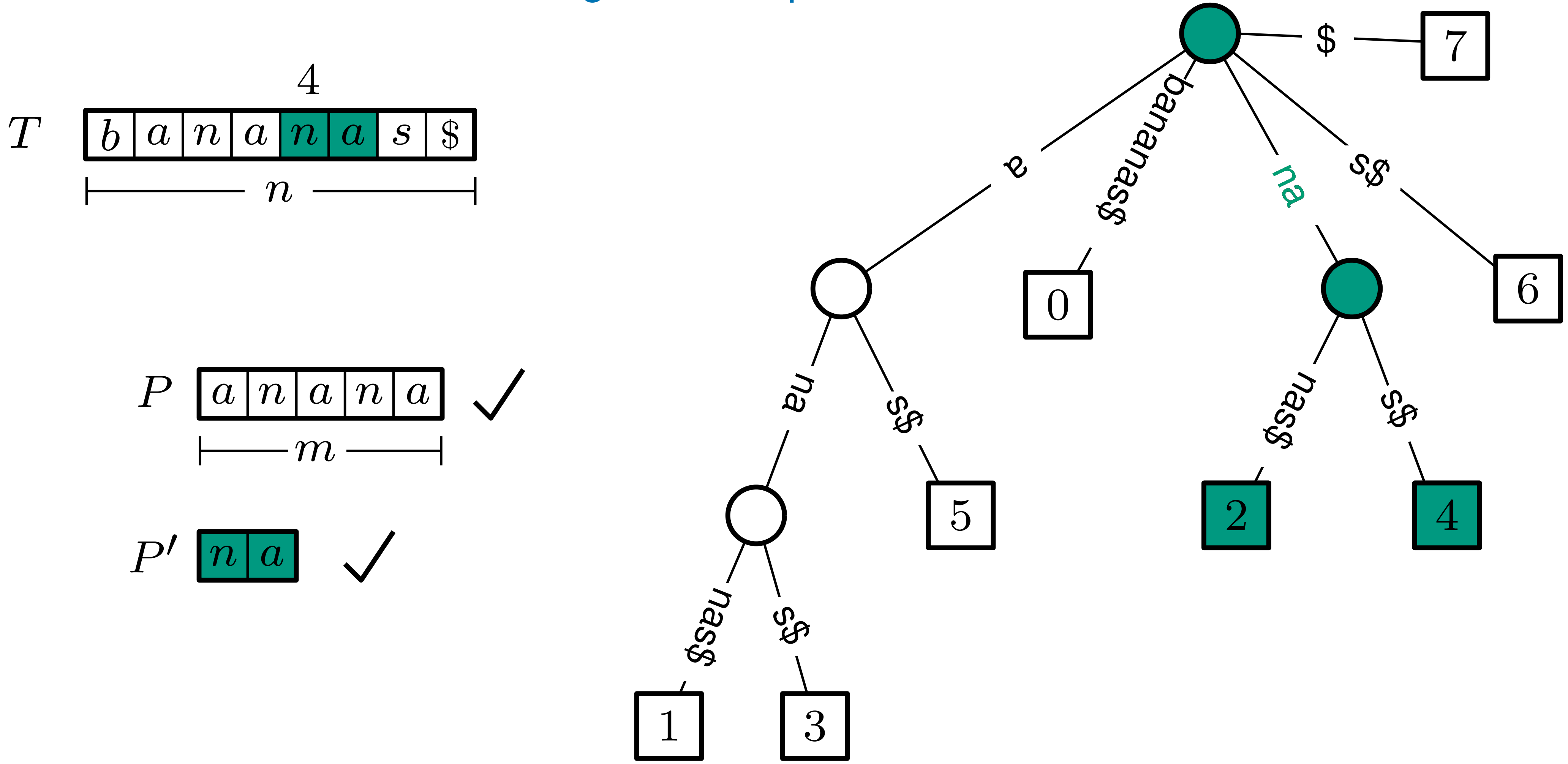


*How do you find a pattern?*

start at the root and walk down the tree

…matches occur at the leaves of the subtree

# Searching in a compacted suffix tree

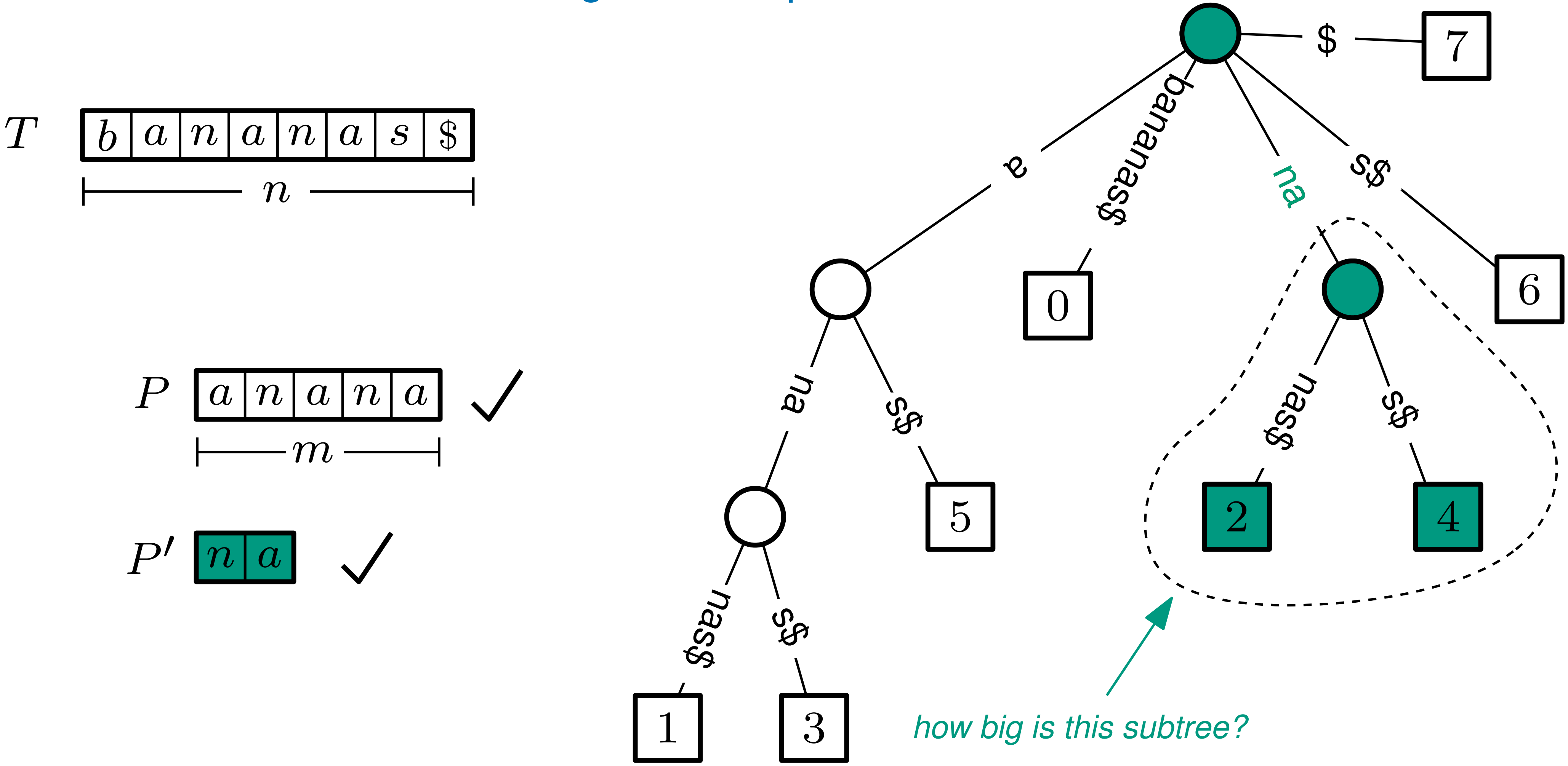


*How do you find a pattern?*

start at the root and walk down the tree

…matches occur at the leaves of the subtree

# Searching in a compacted suffix tree



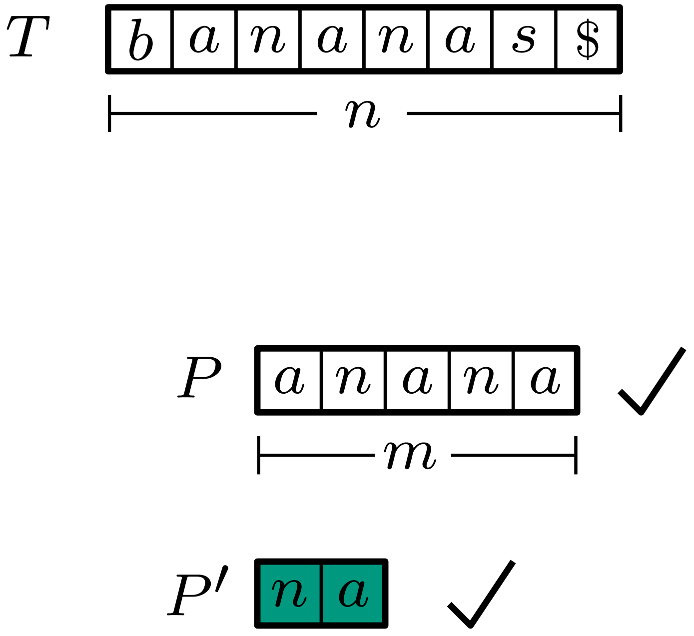
*how big is this subtree?*

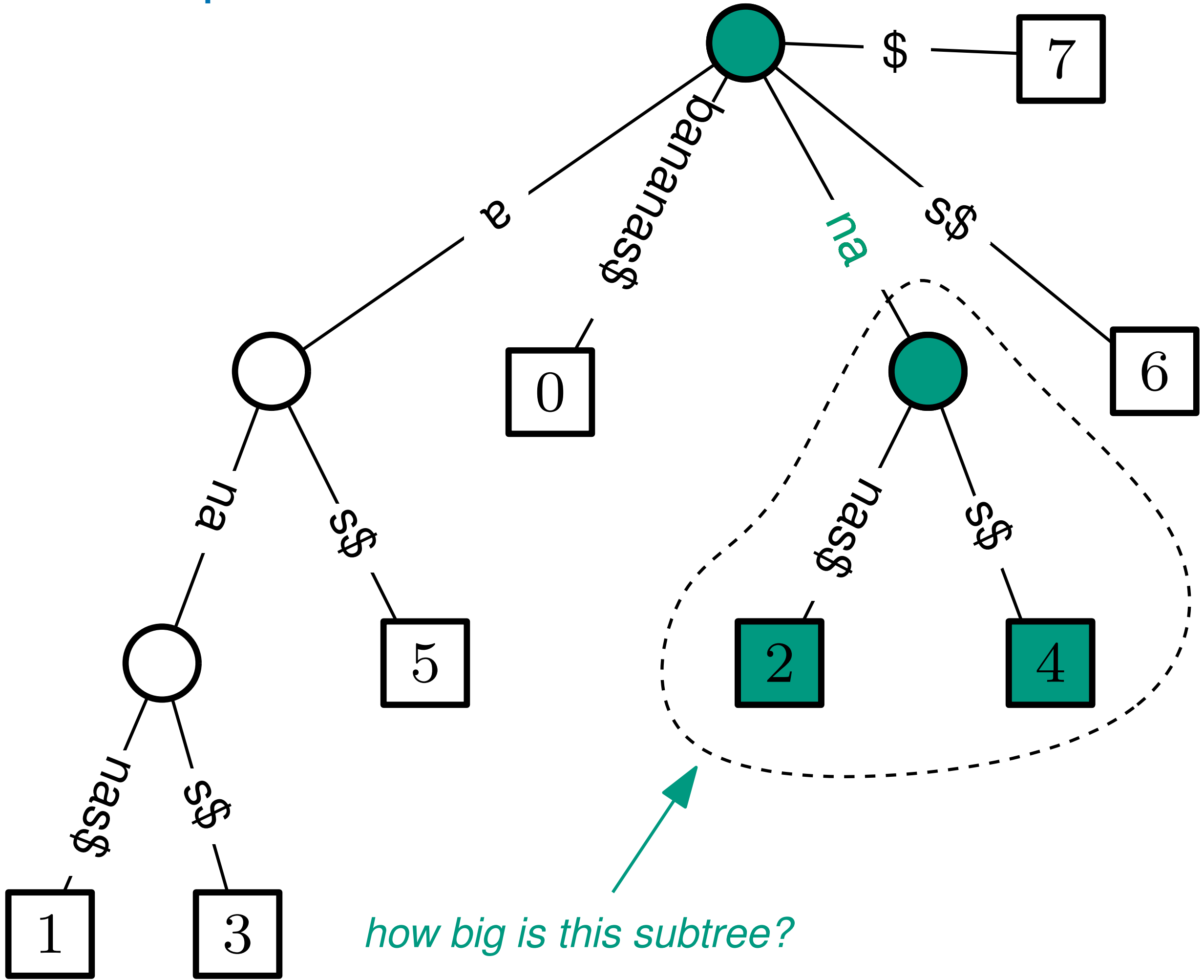
*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in a compacted suffix tree





*how big is this subtree?*

$O(\text{occ})$ because it has occ leaves

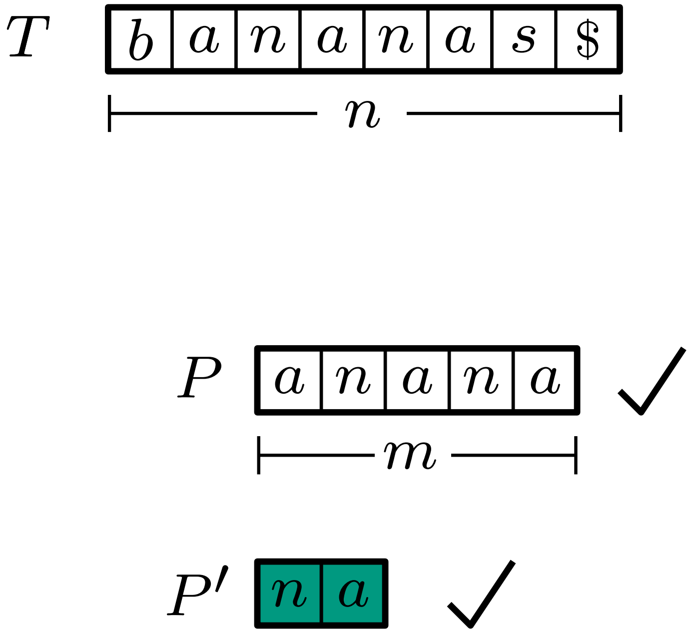
(and each internal node has
at least two children)

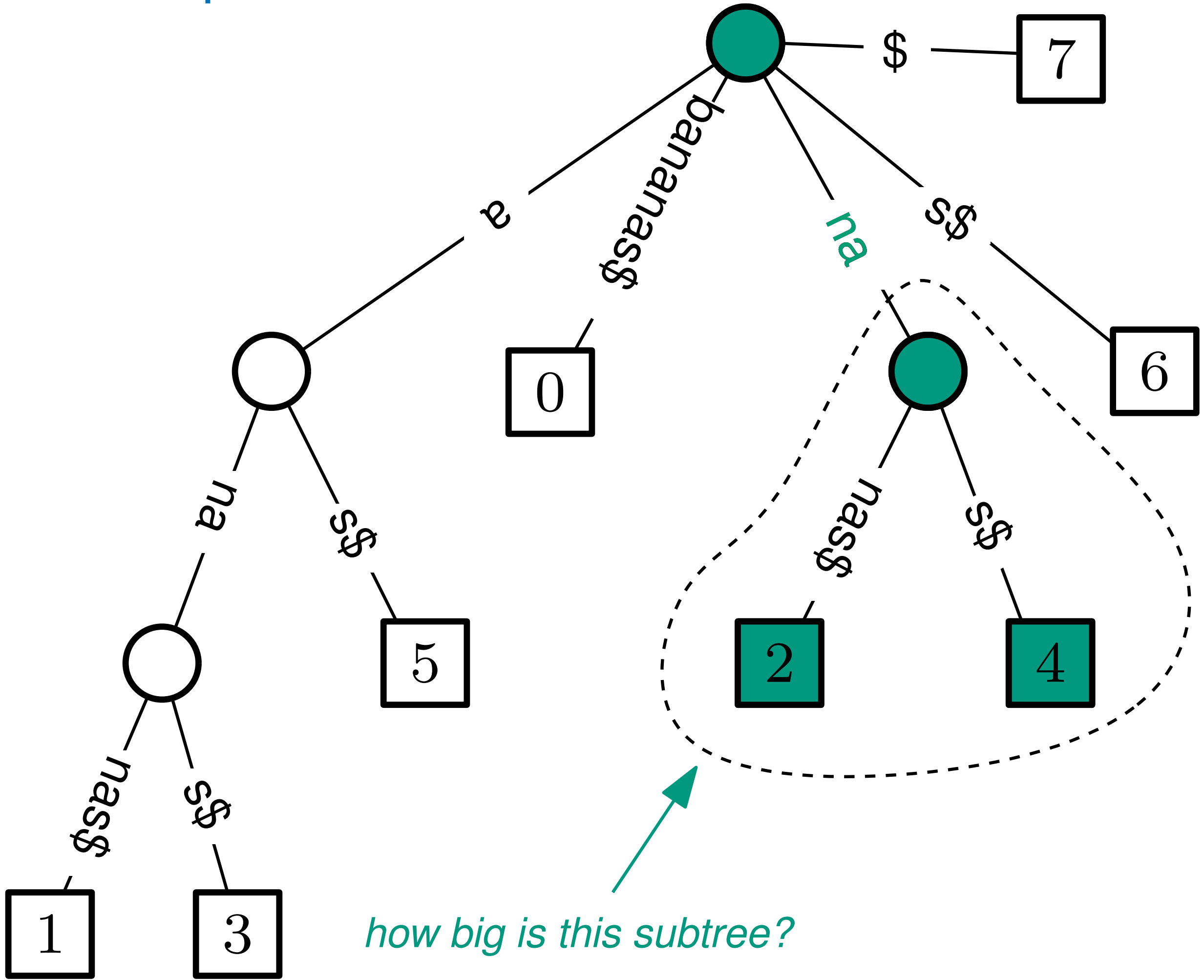
*How do you find a pattern?*

start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

# Searching in a compacted suffix tree





*how big is this subtree?*

$O(\text{occ})$ because it has occ leaves

(and each internal node has at least two children)

*How do you find a pattern?*

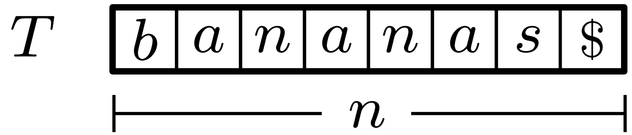
start at the root and walk down the tree

. . . matches occur at the leaves of the subtree

We can find all the matches in $O(m + \text{occ})$ time (by looking at the whole subtree)

you should
*never* actually
do it like this

# Naively constructing a compacted suffix tree



---

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

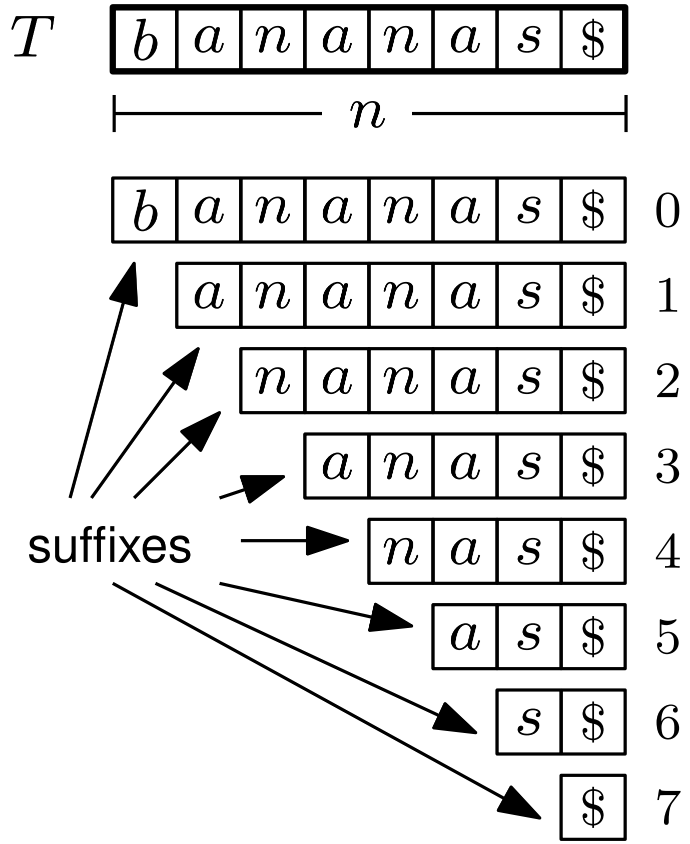
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

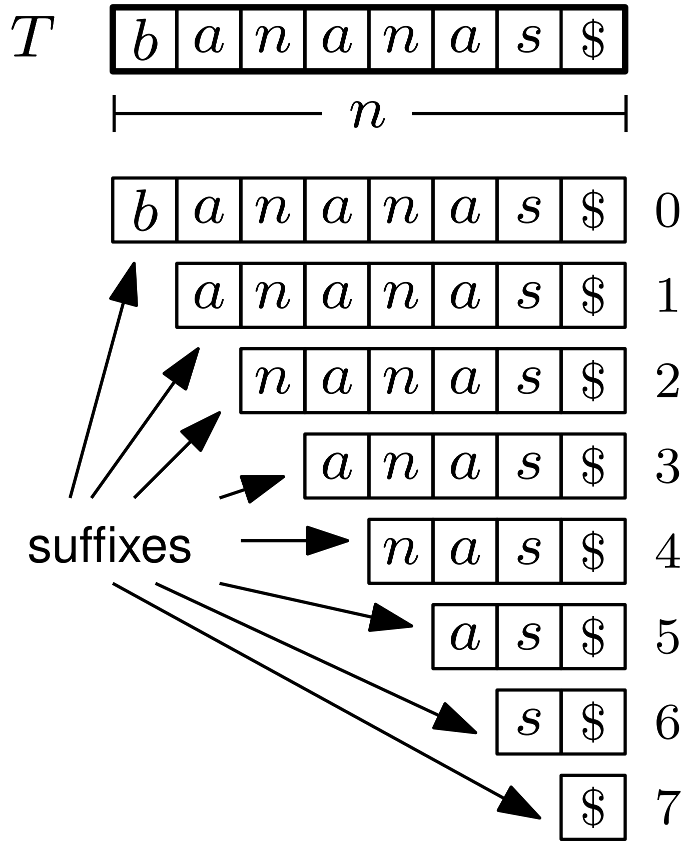
  *(as if you were matching a pattern)*

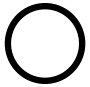
- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should *never* actually do it like this



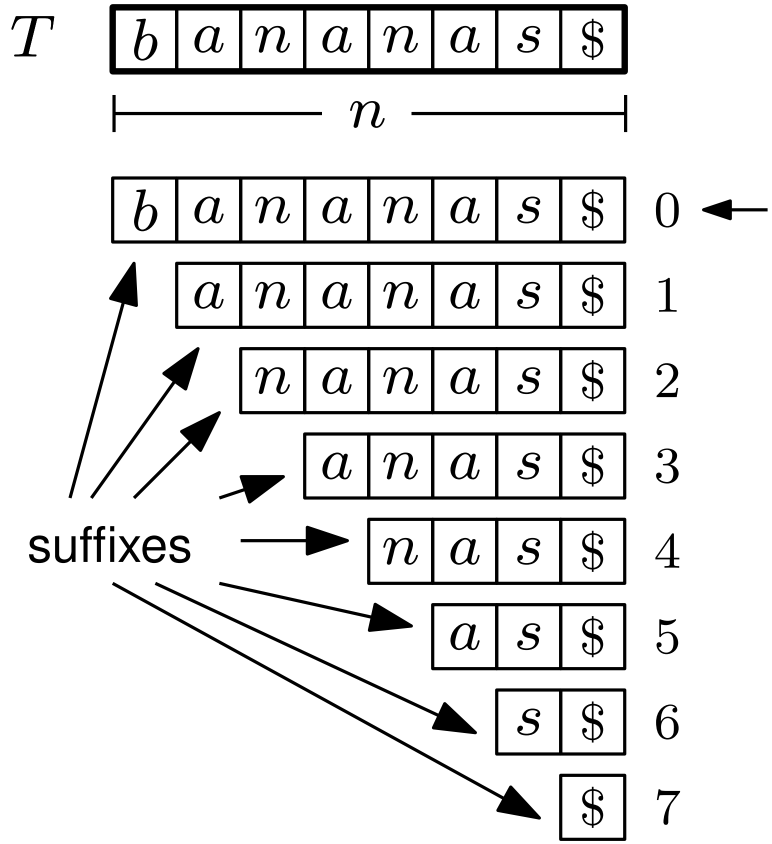


Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

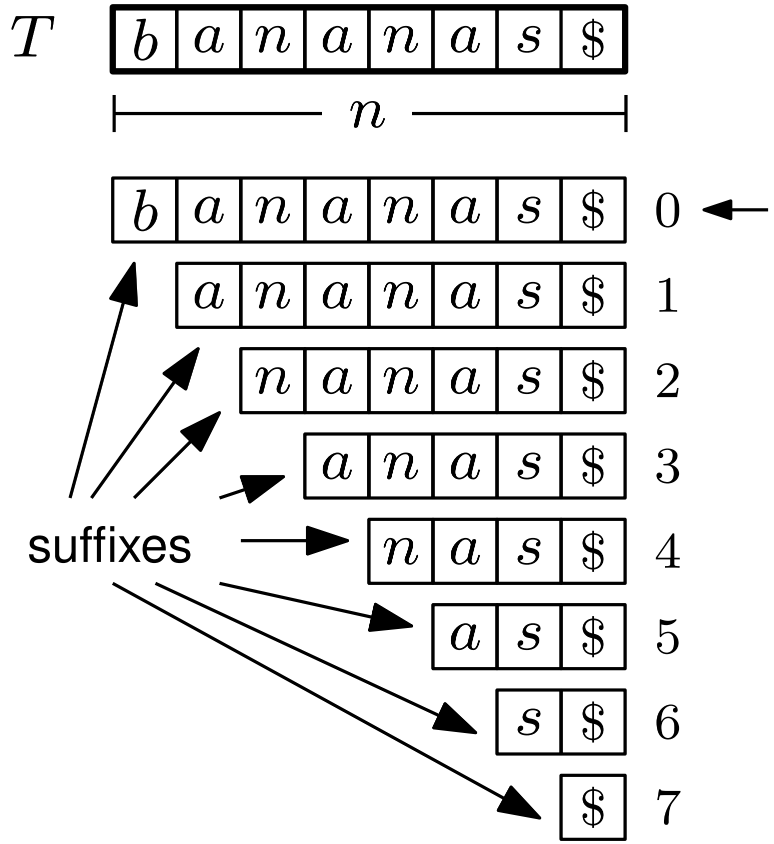
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

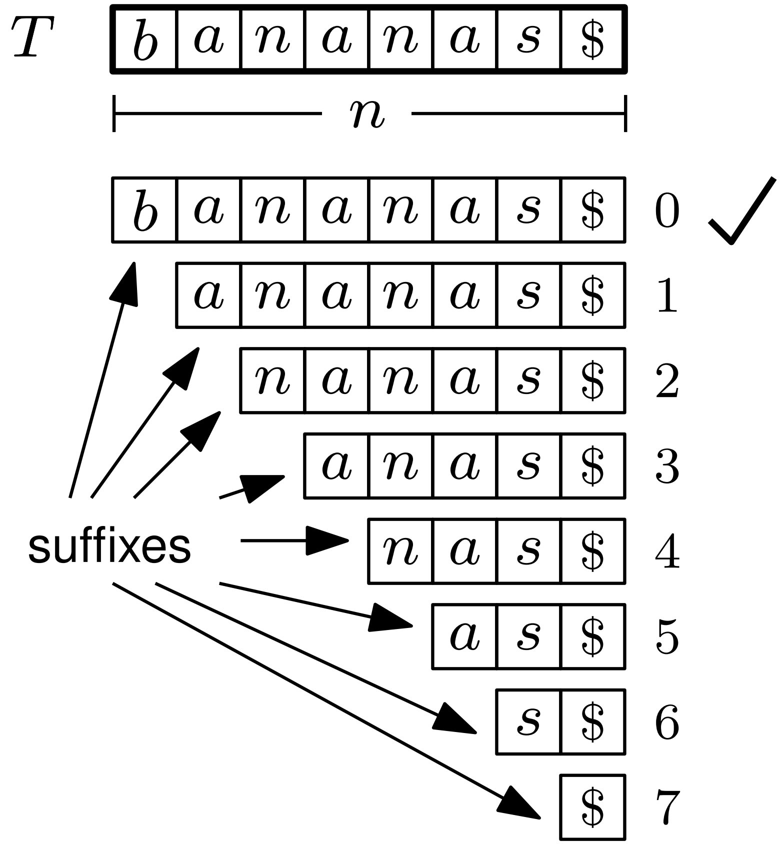
  *(as if you were matching a pattern)*

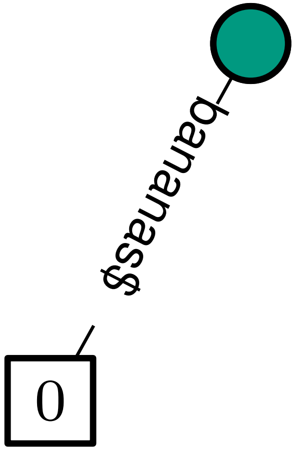
- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



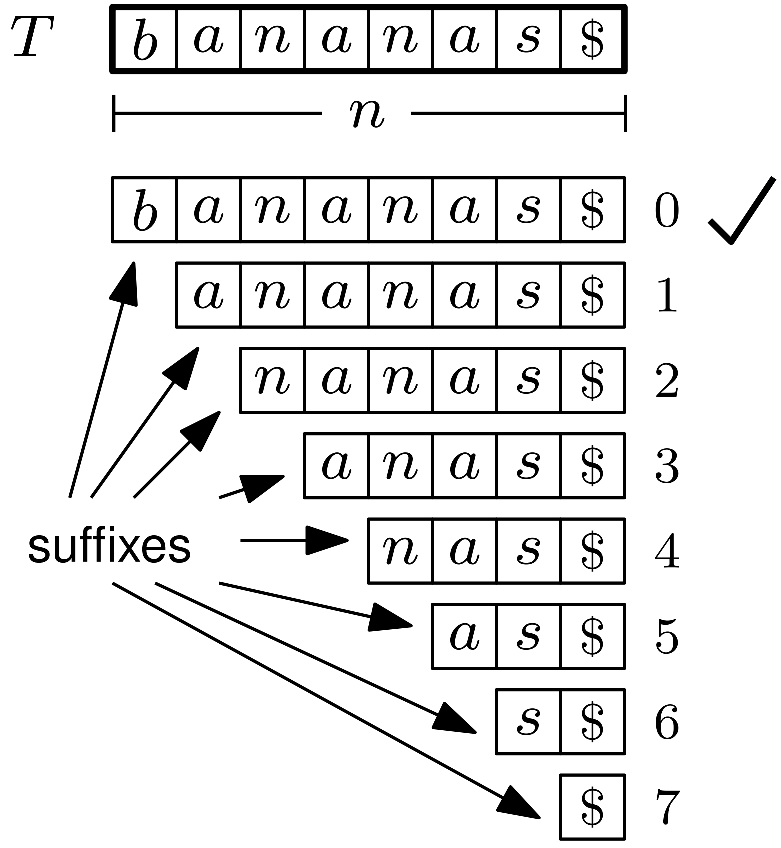


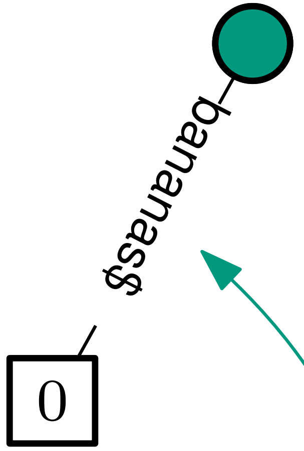
Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree





we actually store this as $(0, 7)$

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

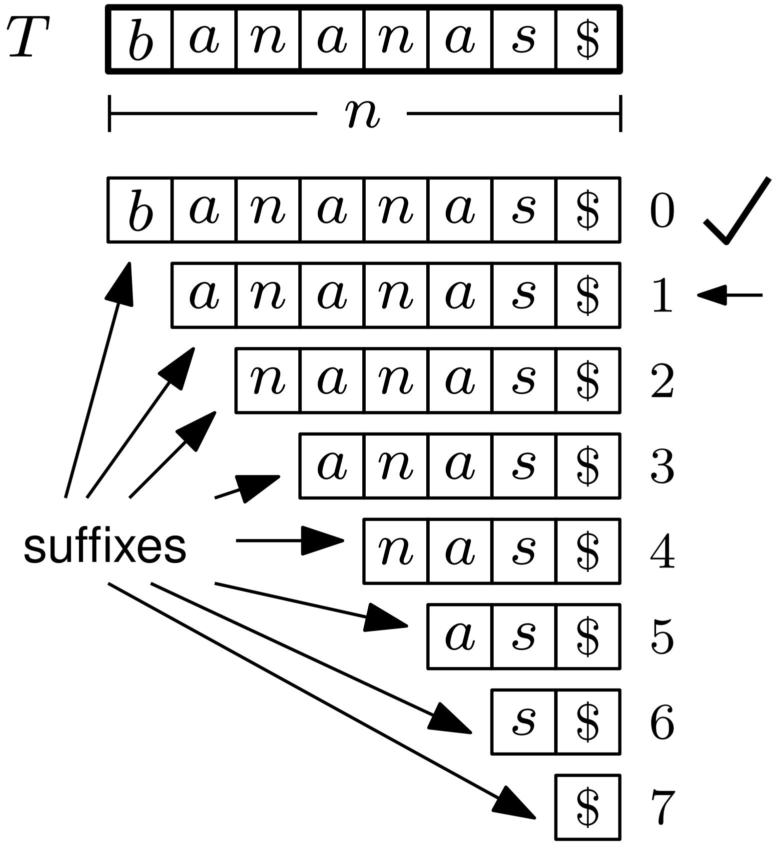
  *(as if you were matching a pattern)*

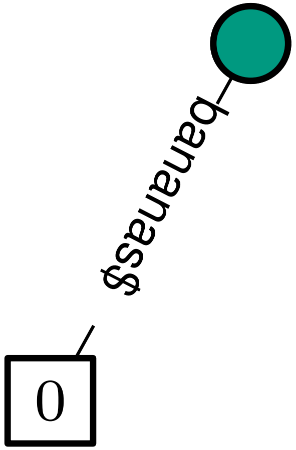
- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree





Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

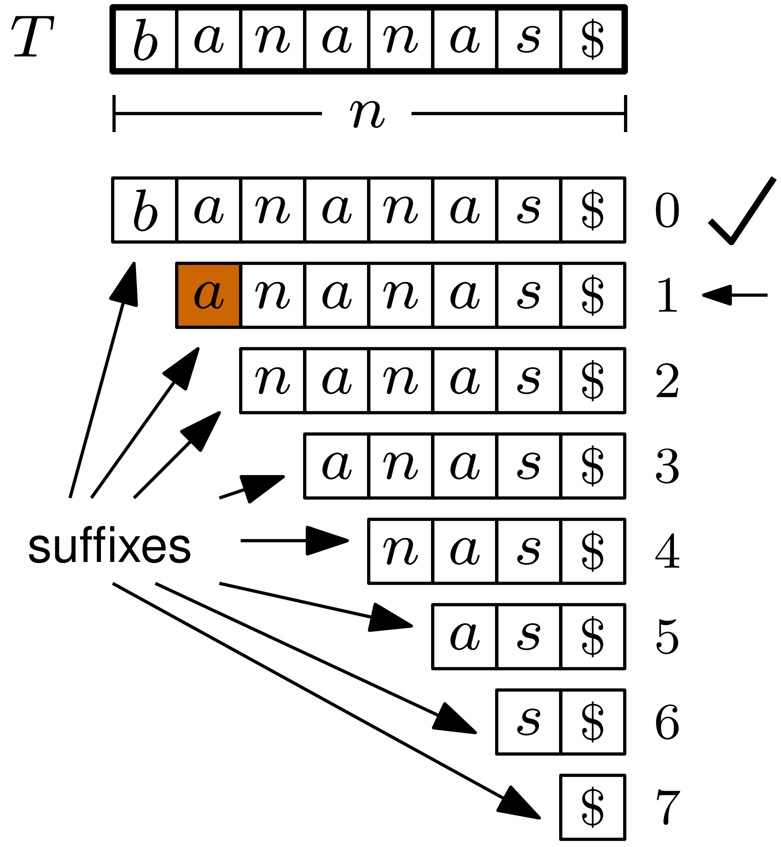
  *(as if you were matching a pattern)*

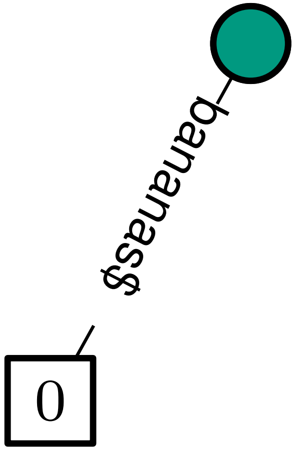
- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree





Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

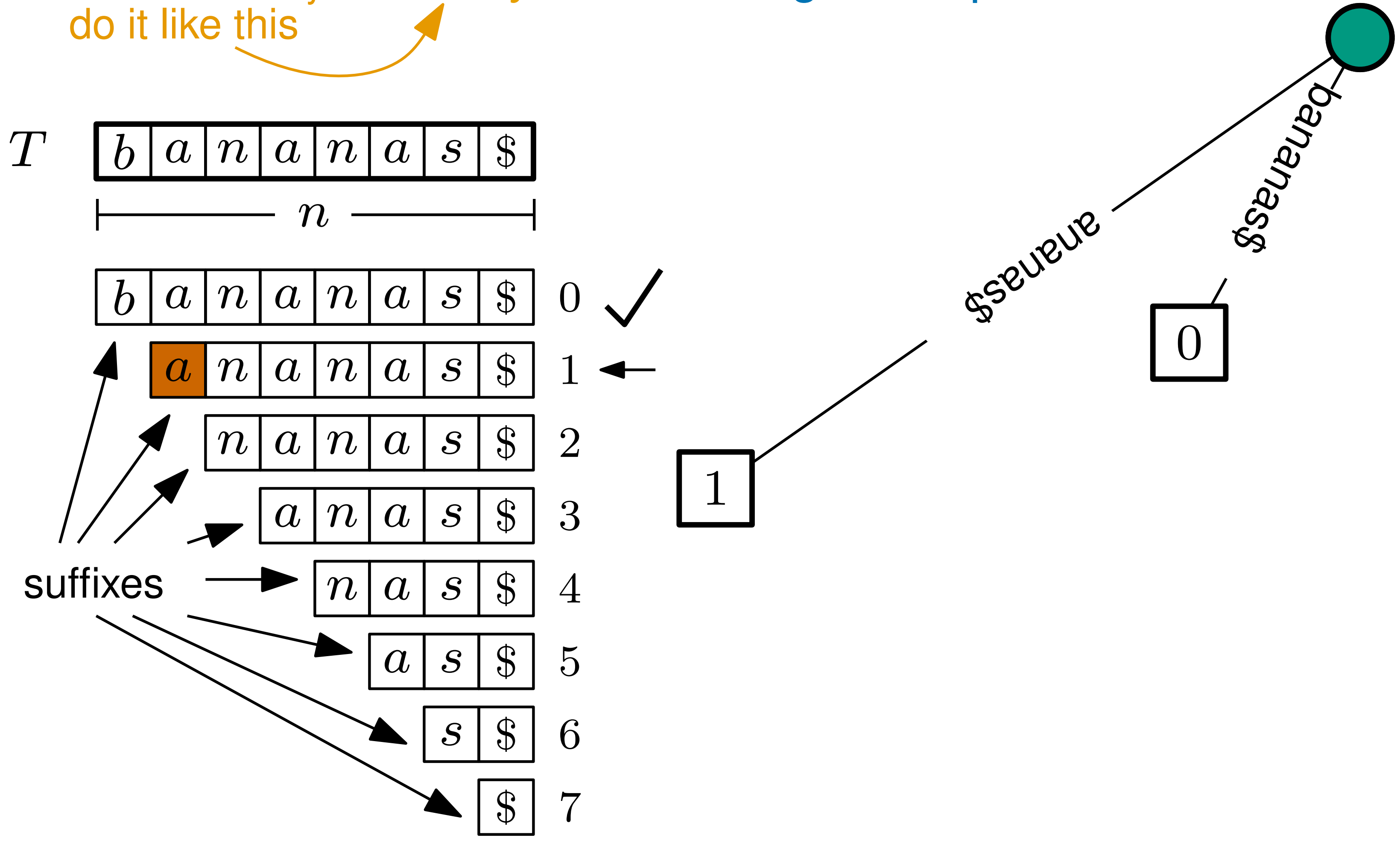
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should *never* actually do it like this



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

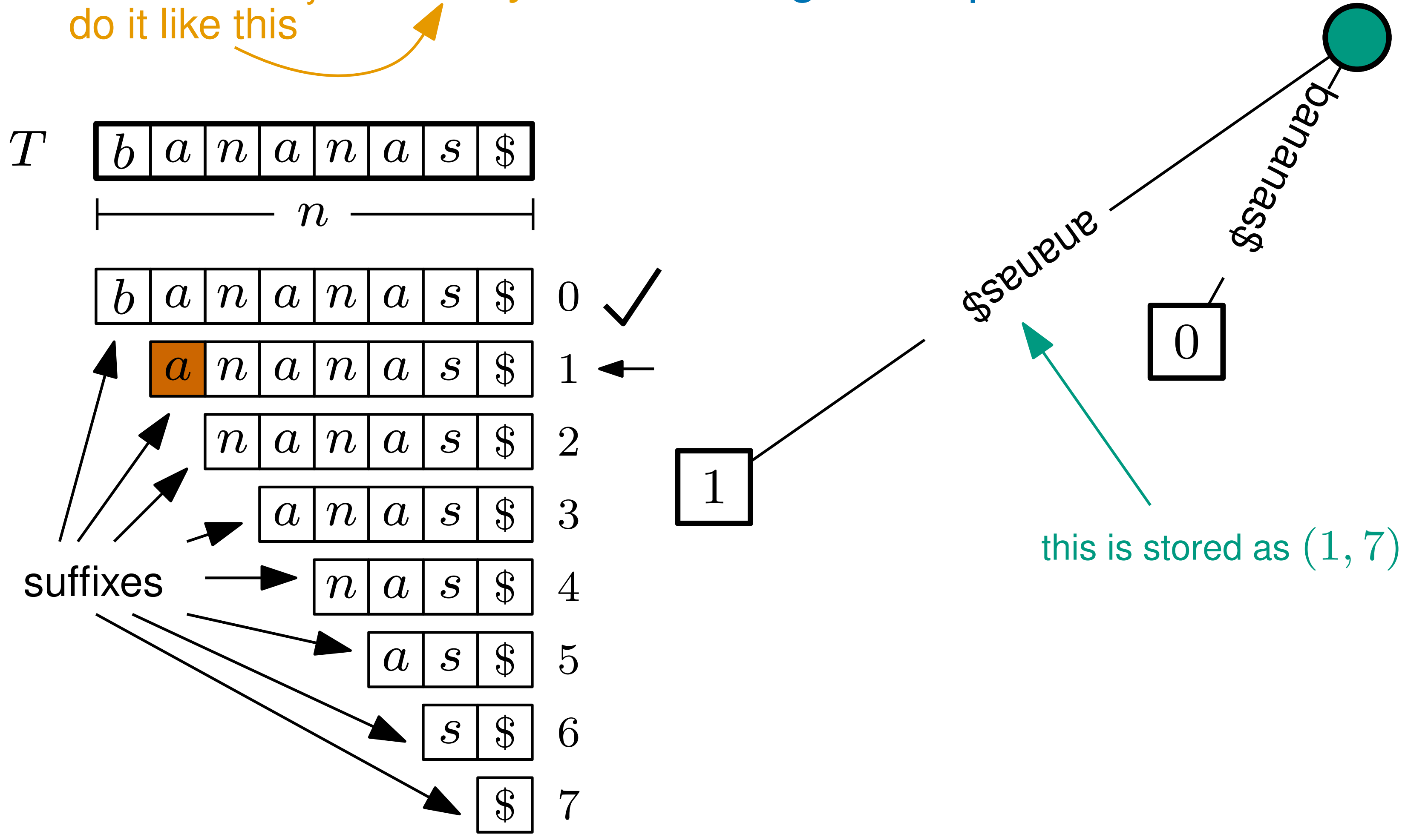
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should *never* actually do it like this



this is stored as $(1, 7)$

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

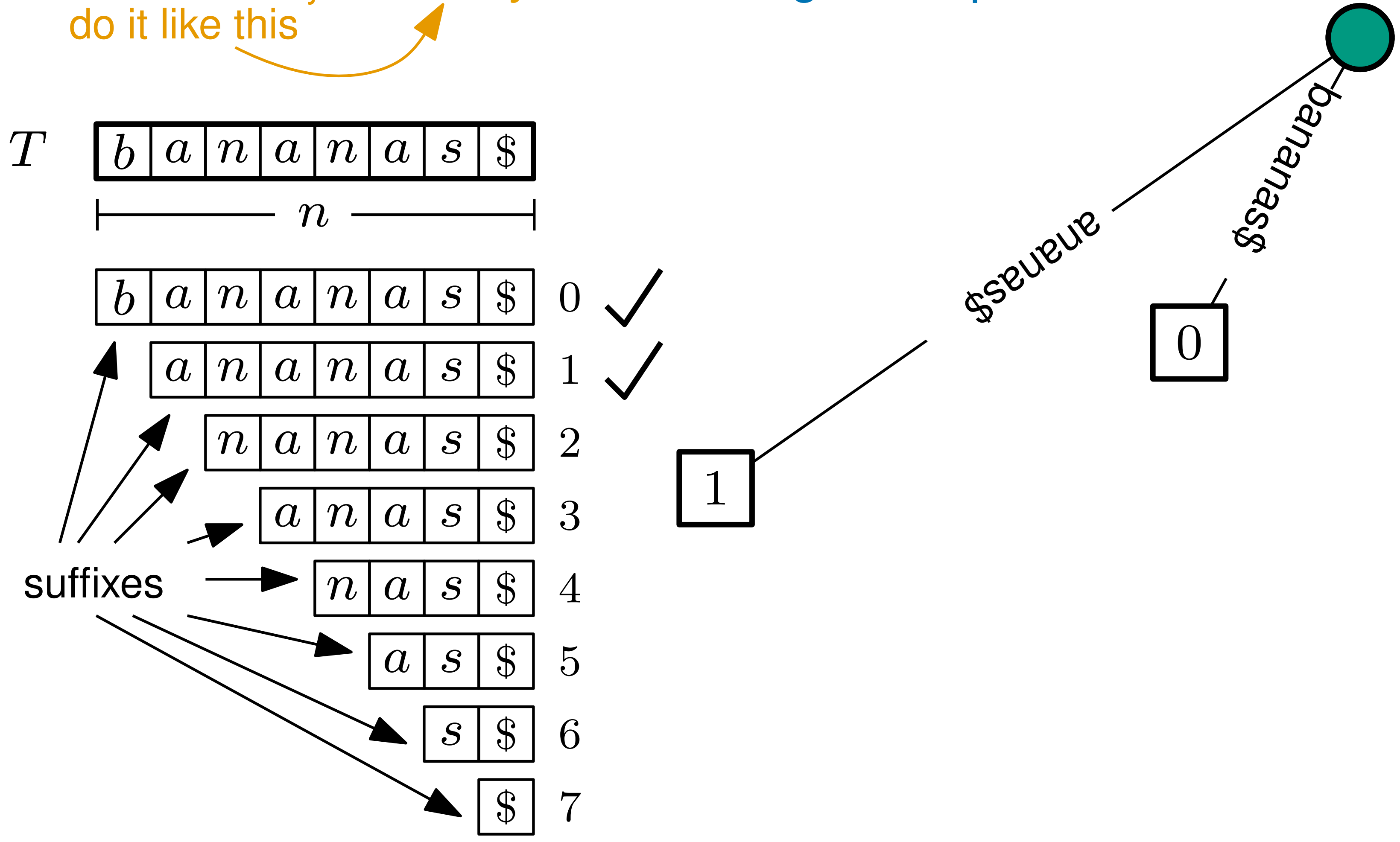
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

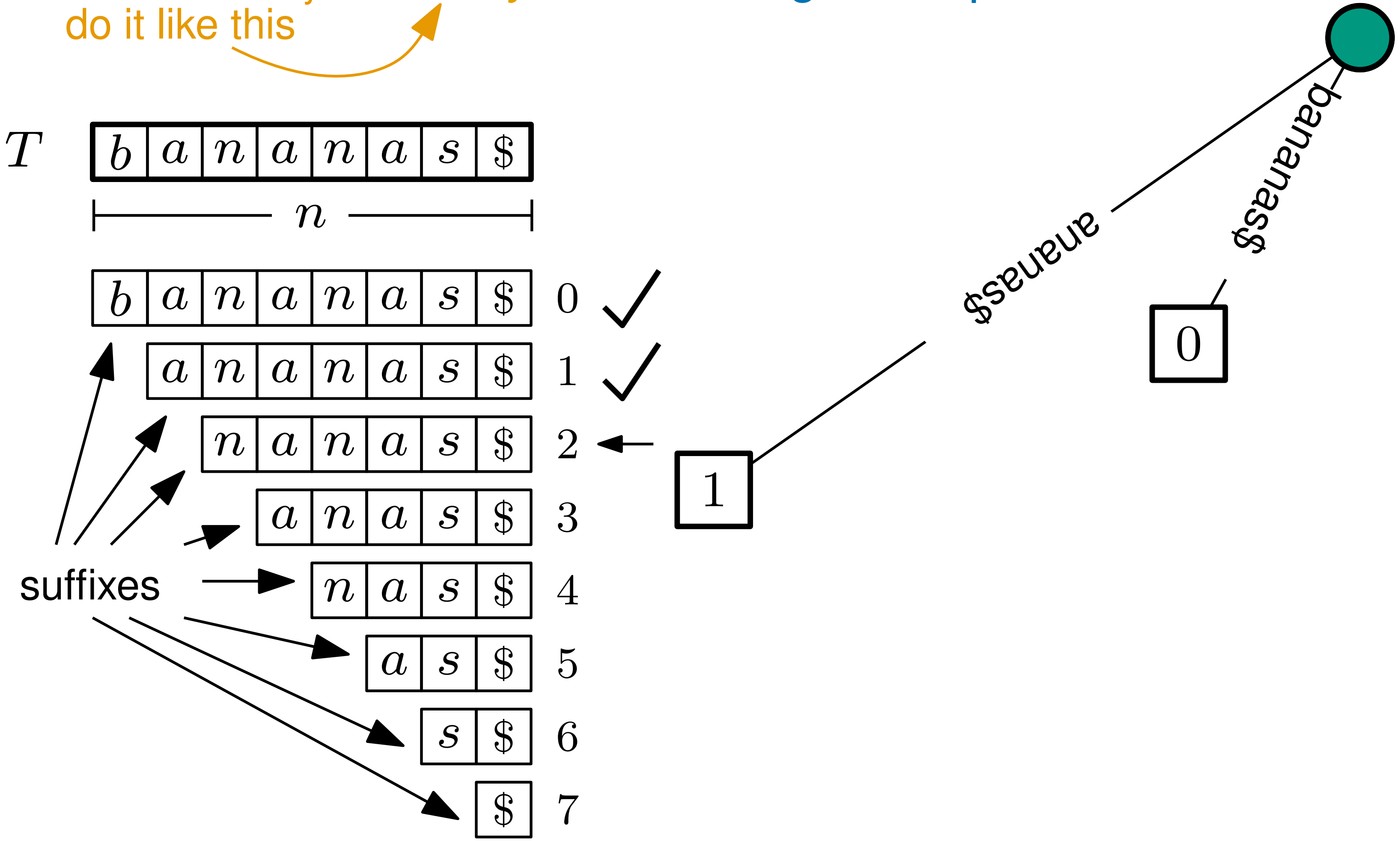
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should
*never* actually
do it like this

# Naively constructing a compacted suffix tree

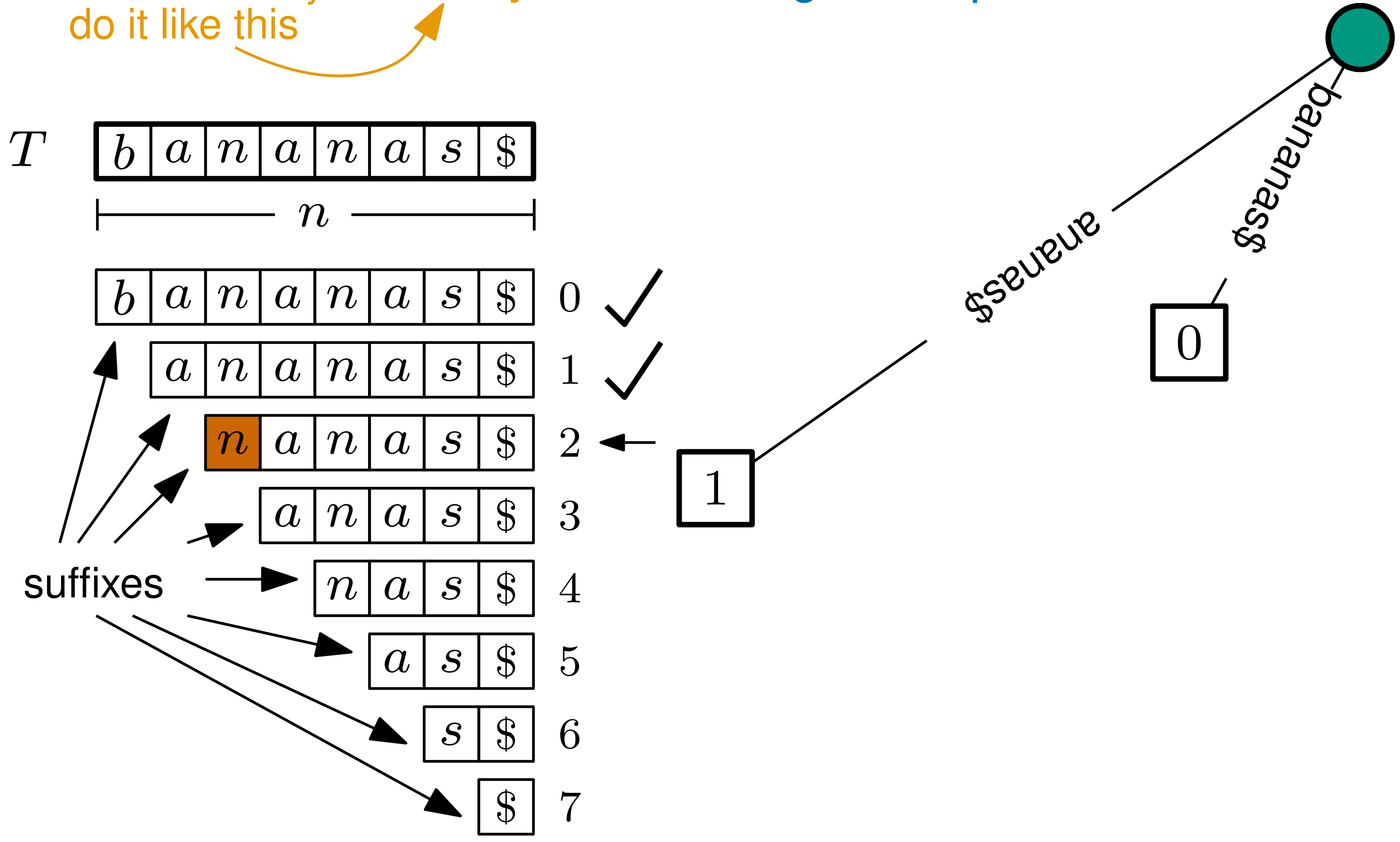


Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should *never* actually do it like this

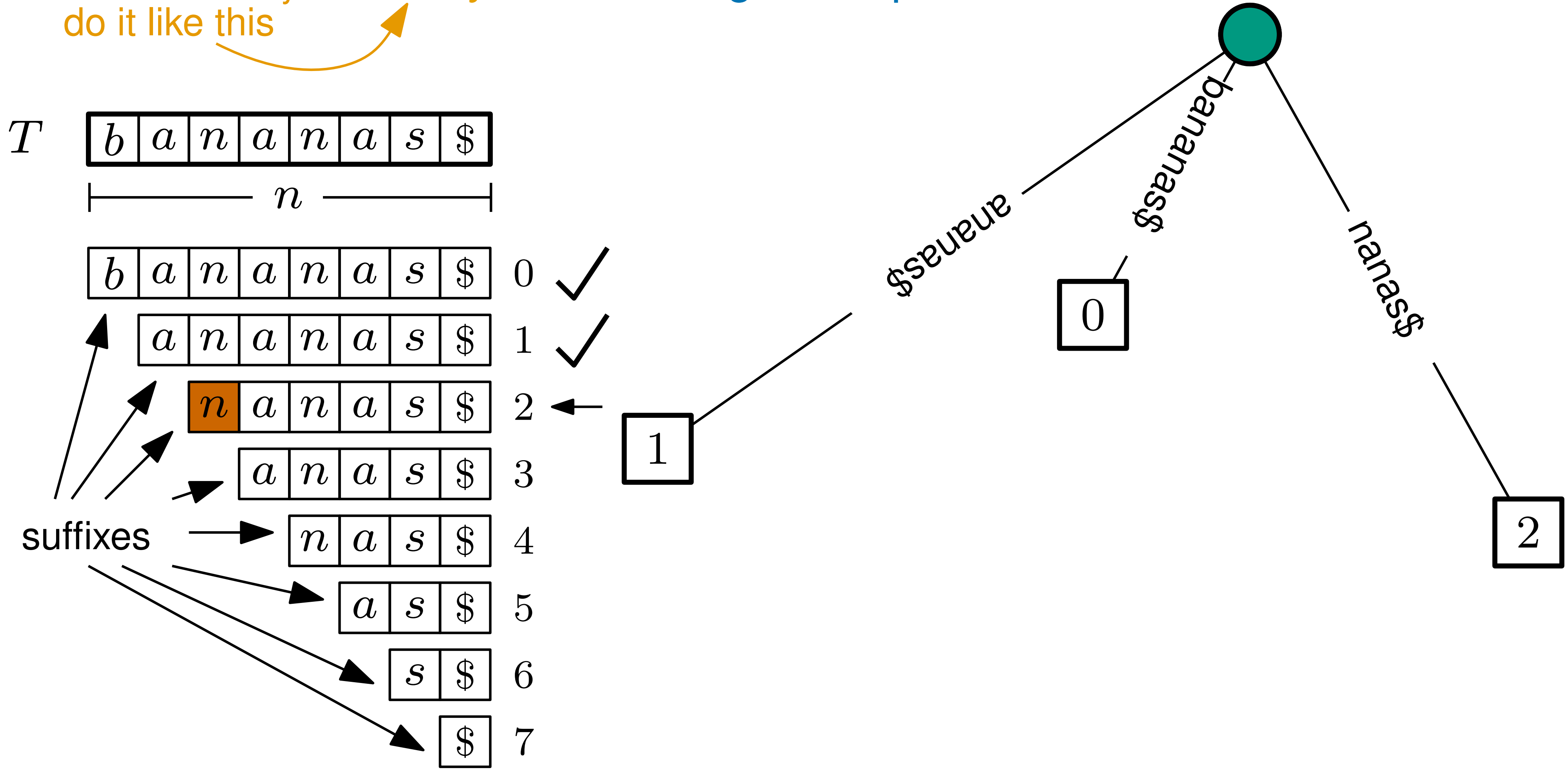


Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

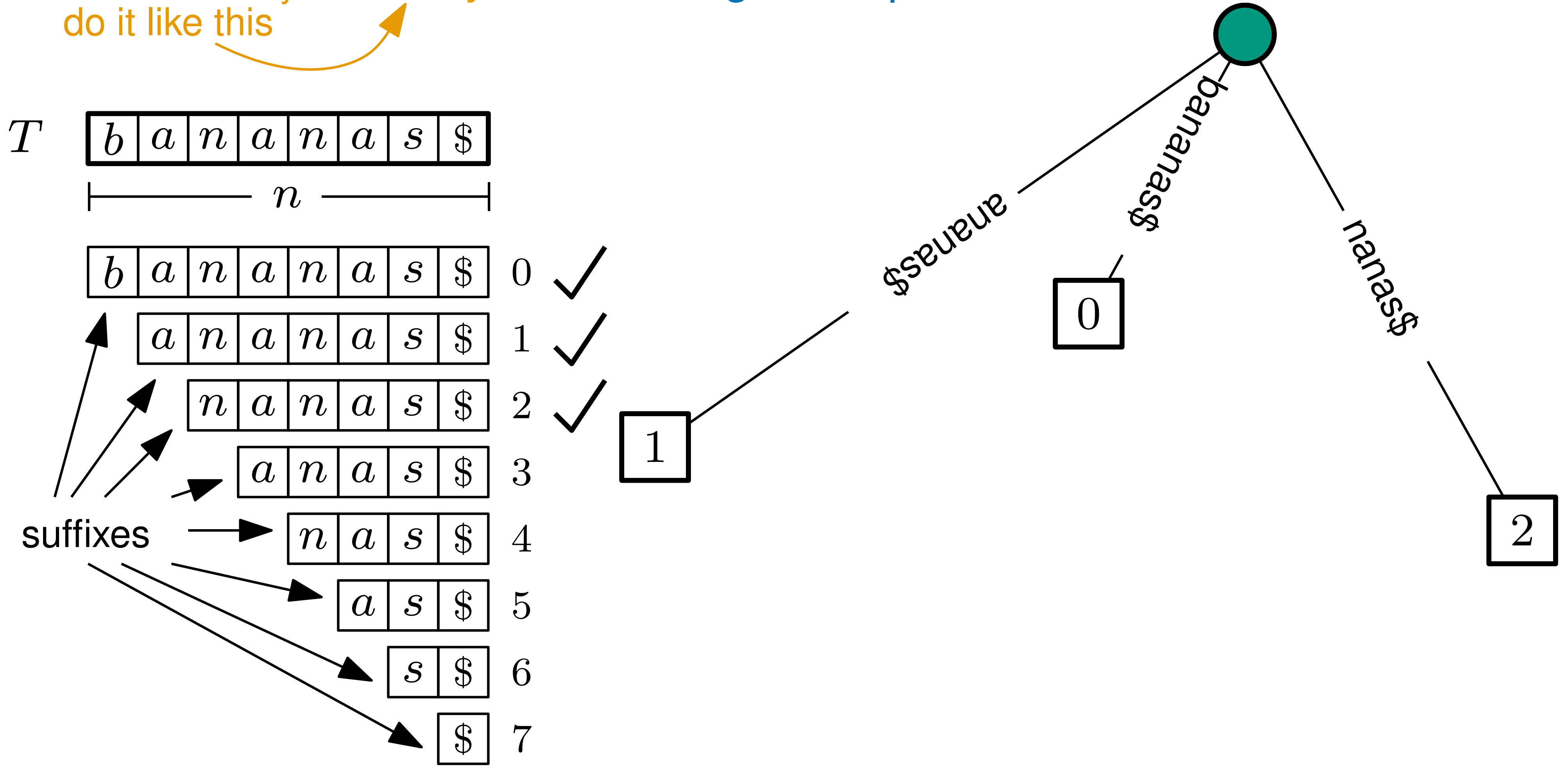
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should *never* actually do it like this

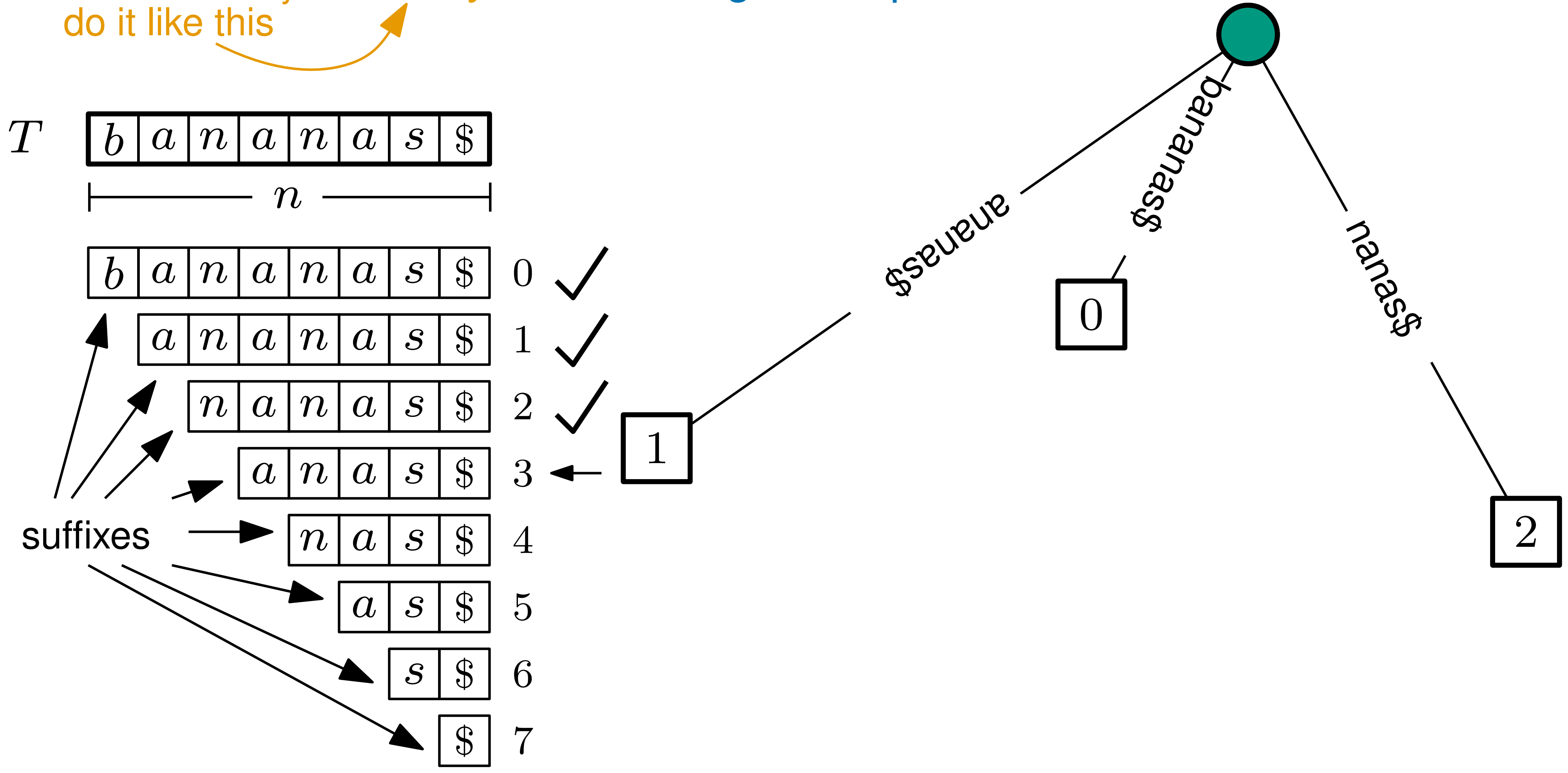


Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

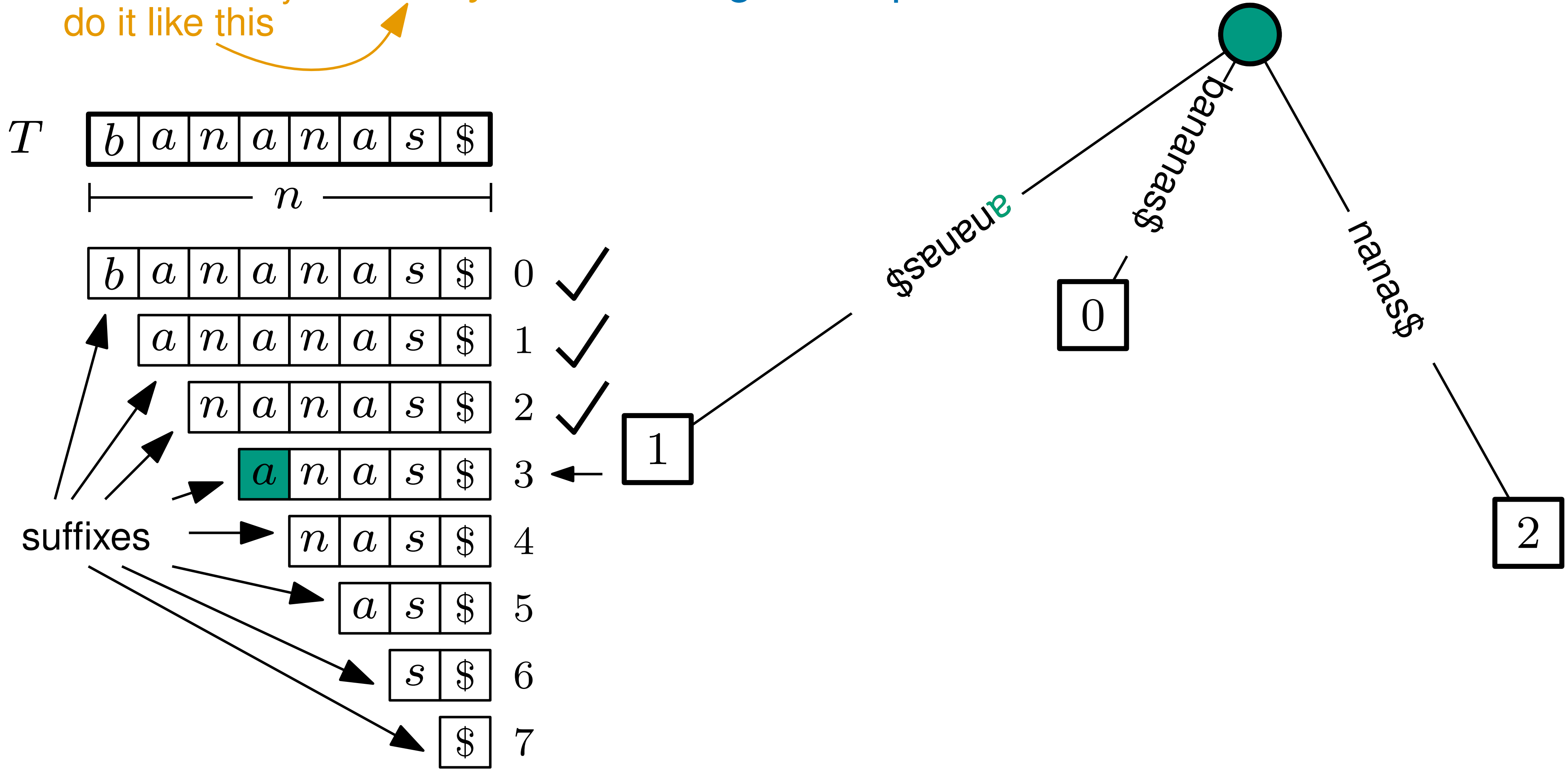
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

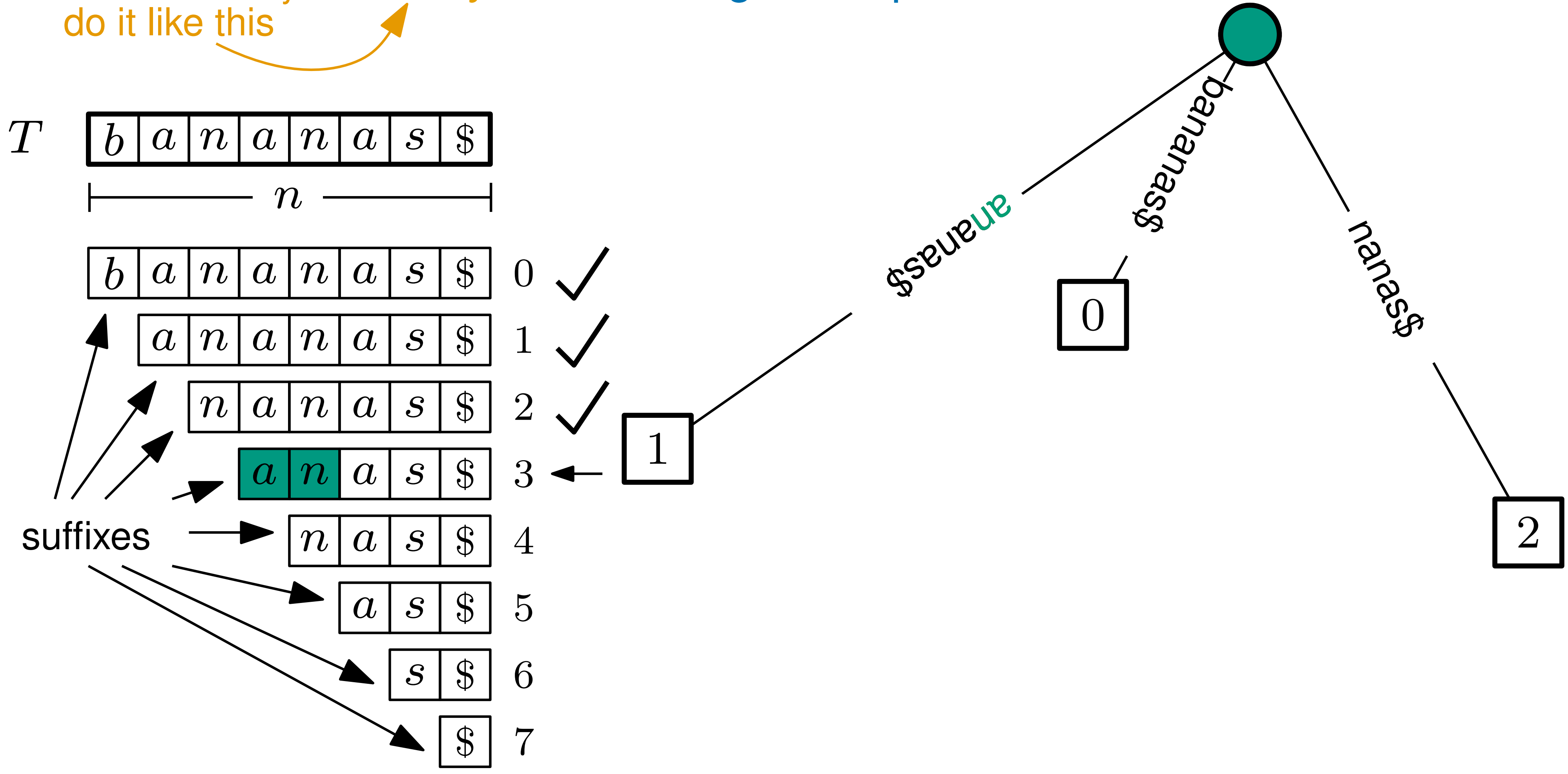
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree

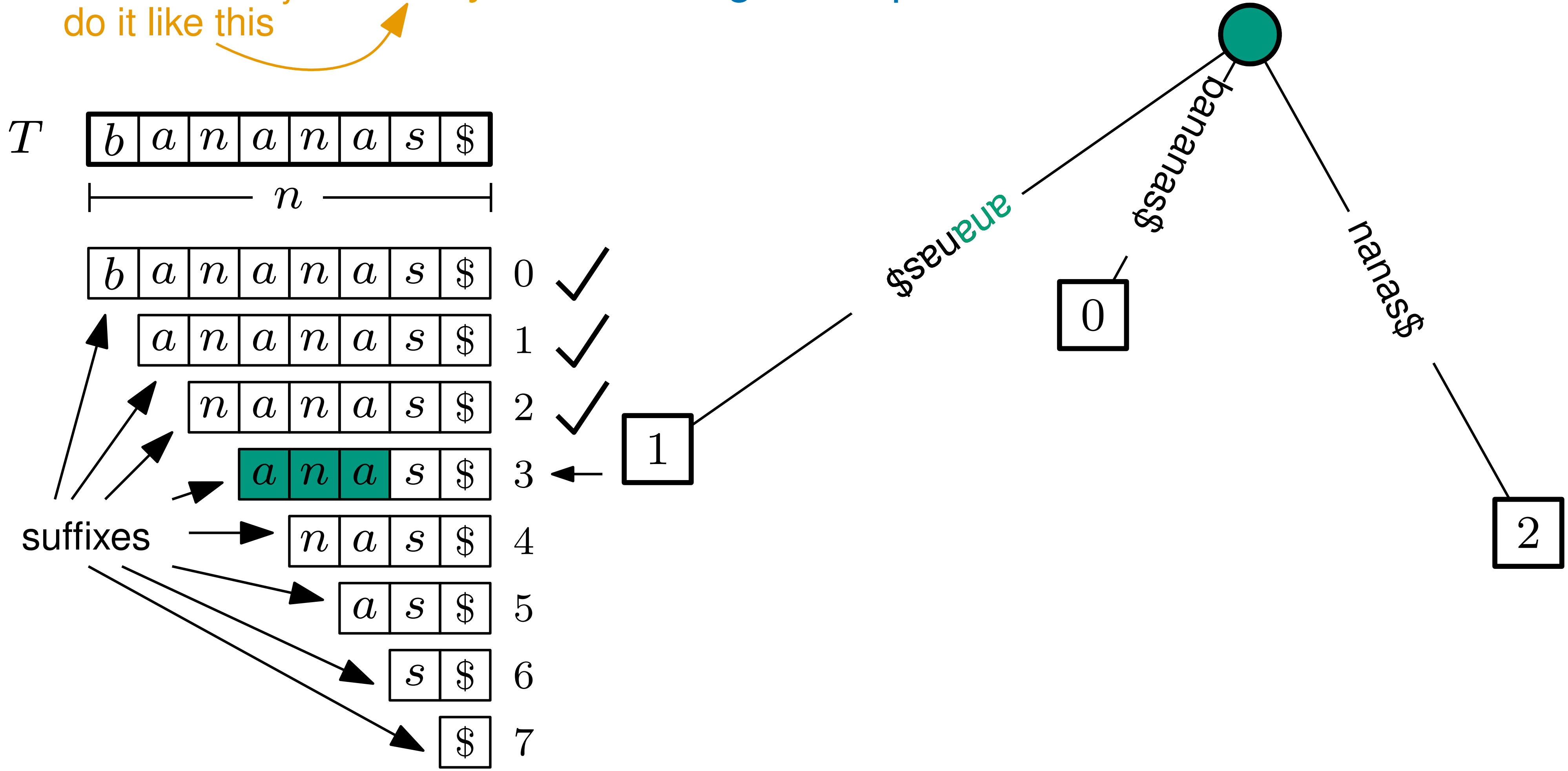


Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should
*never* actually
do it like this



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

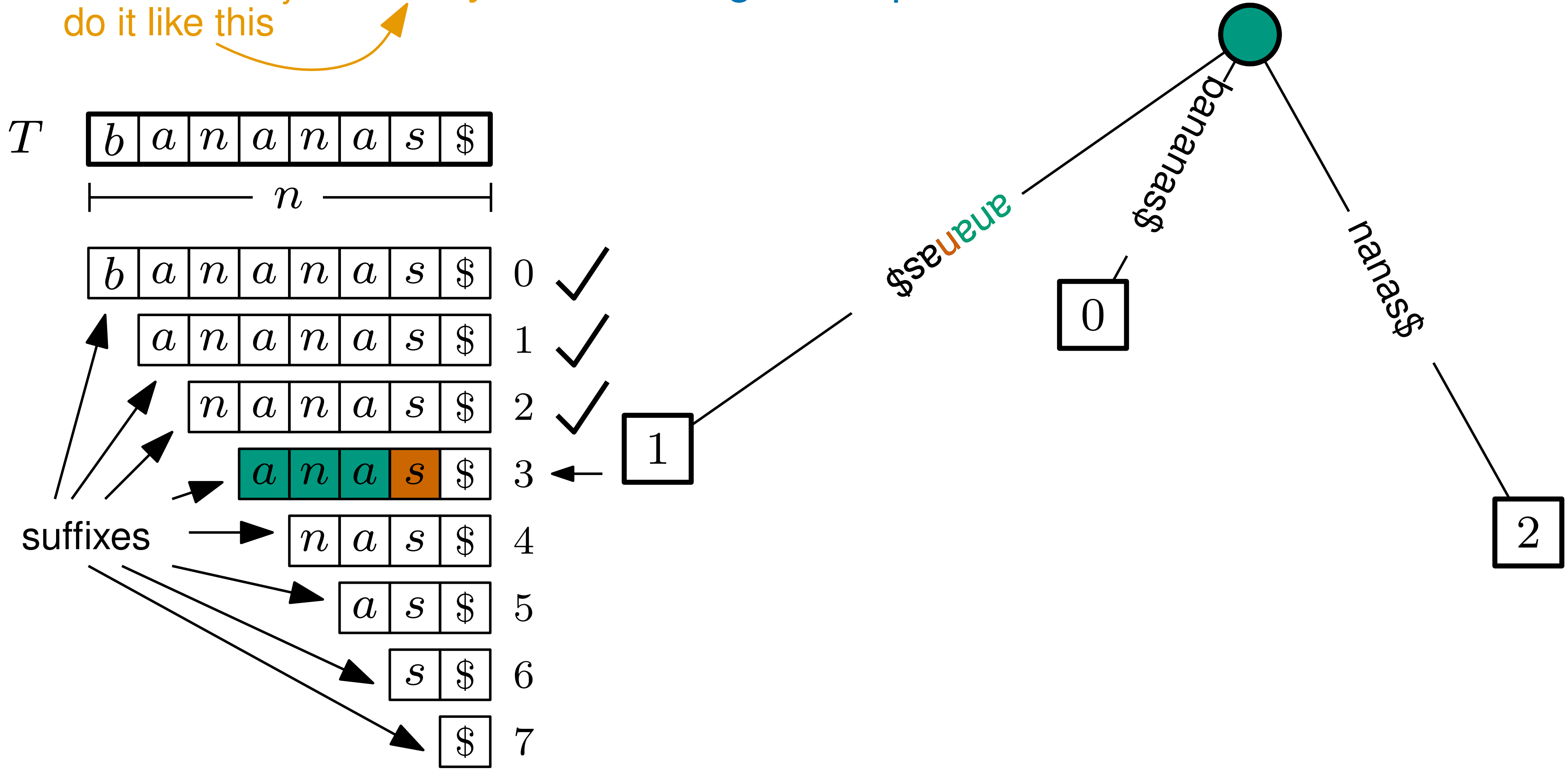
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



---

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

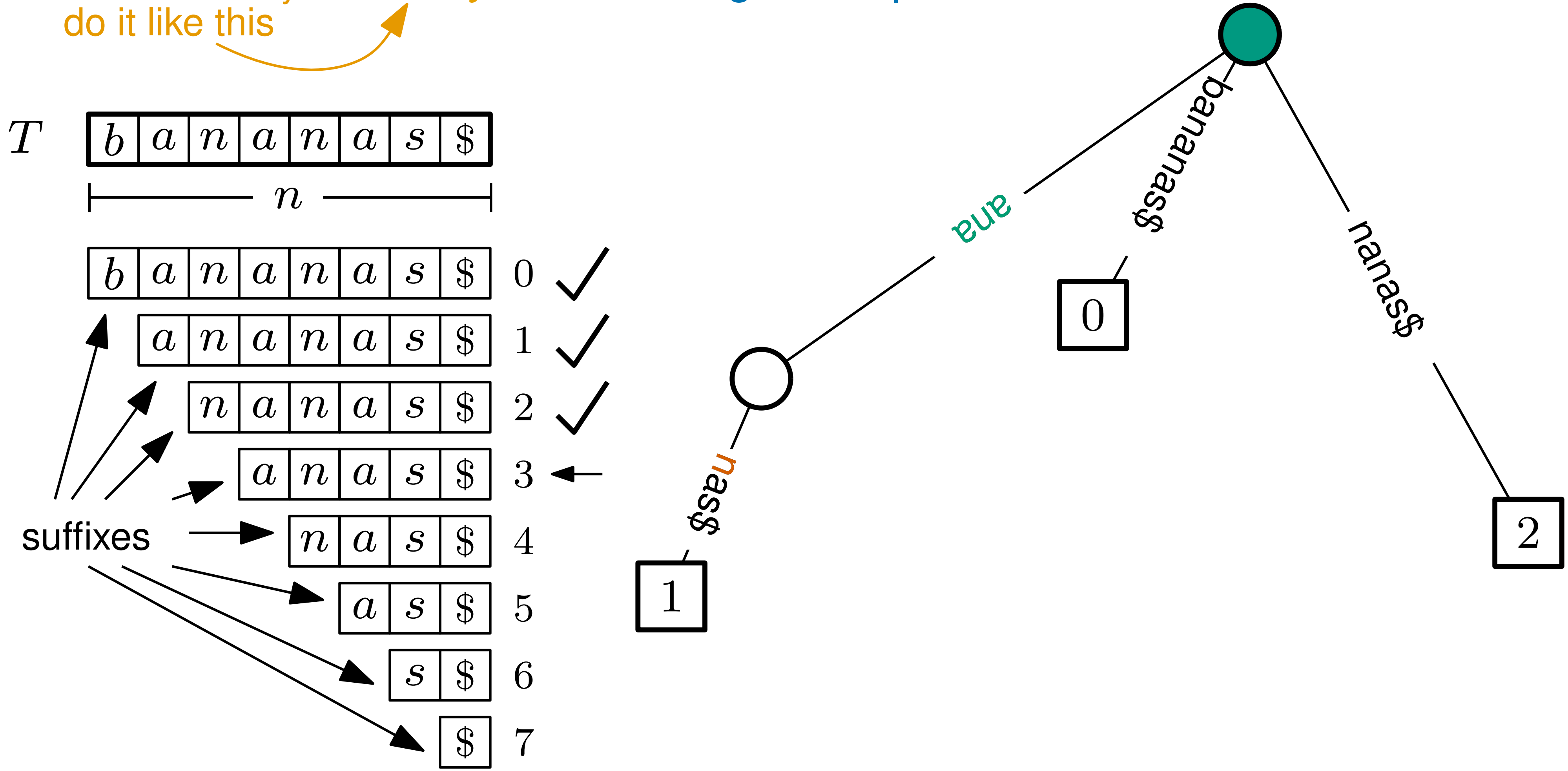
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should
*never* actually
do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

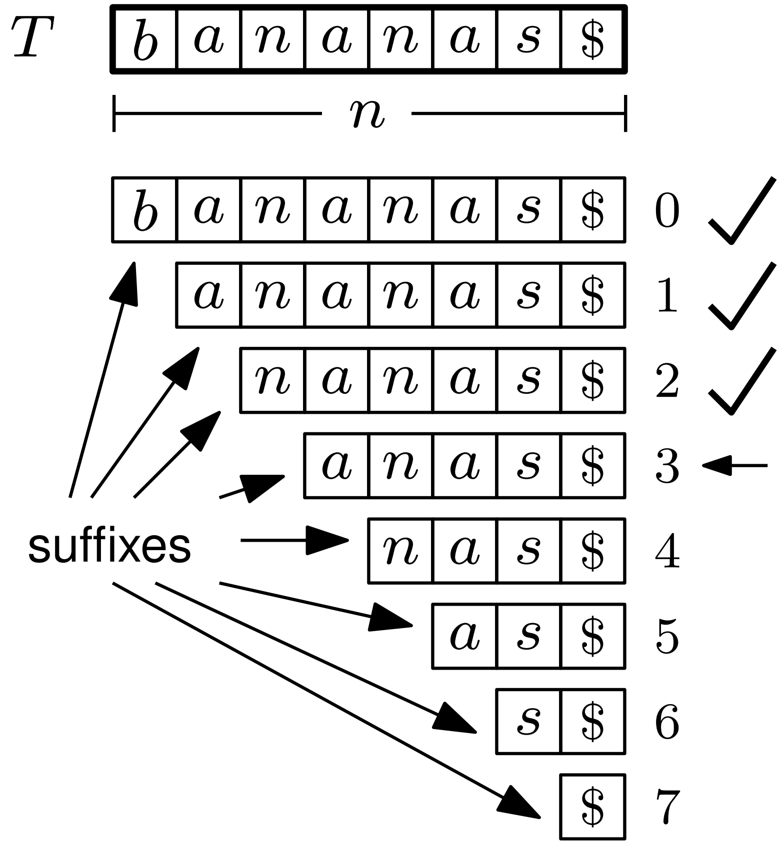
  *(as if you were matching a pattern)*

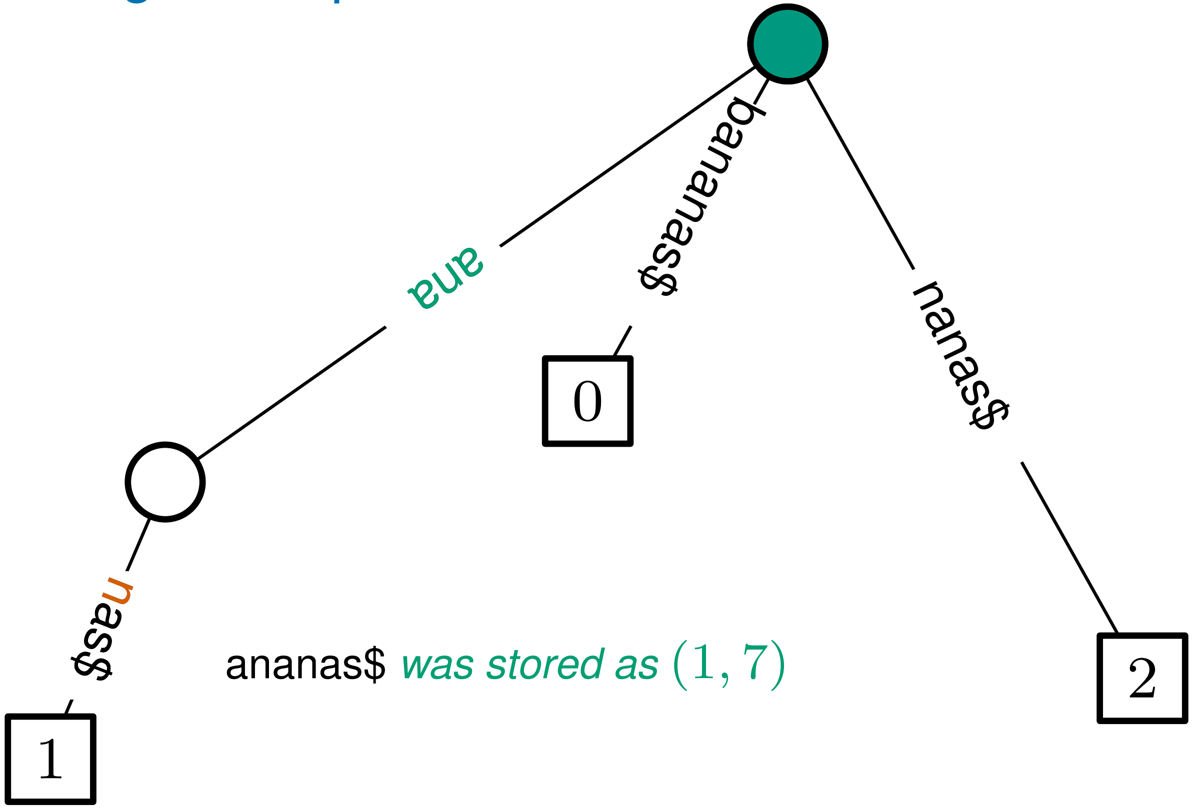
- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree





ananas$ *was stored as* $(1, 7)$

---

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

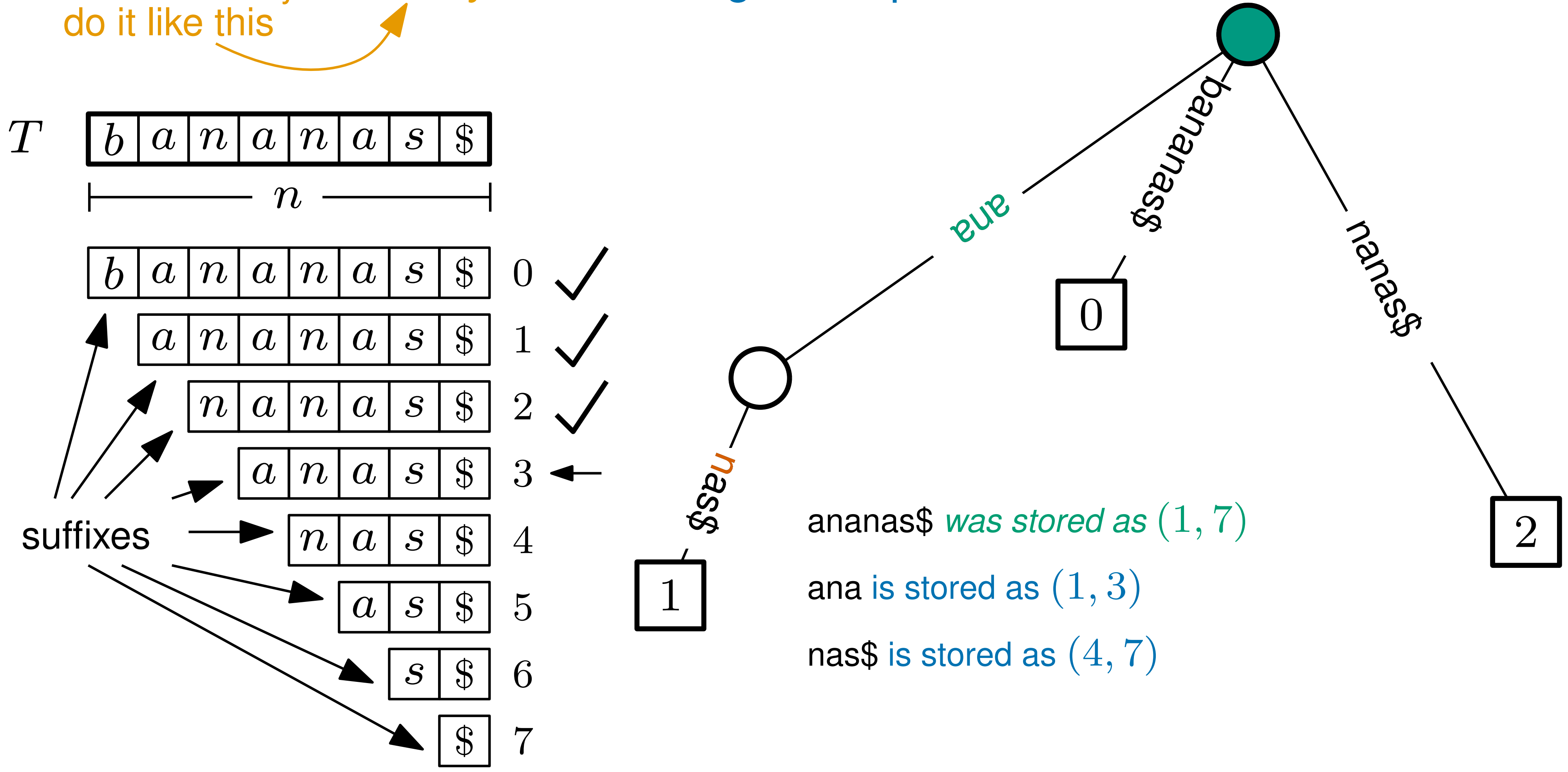
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



ananas\$ *was stored as* $(1, 7)$

ana is stored as $(1, 3)$

nas\$ is stored as $(4, 7)$

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

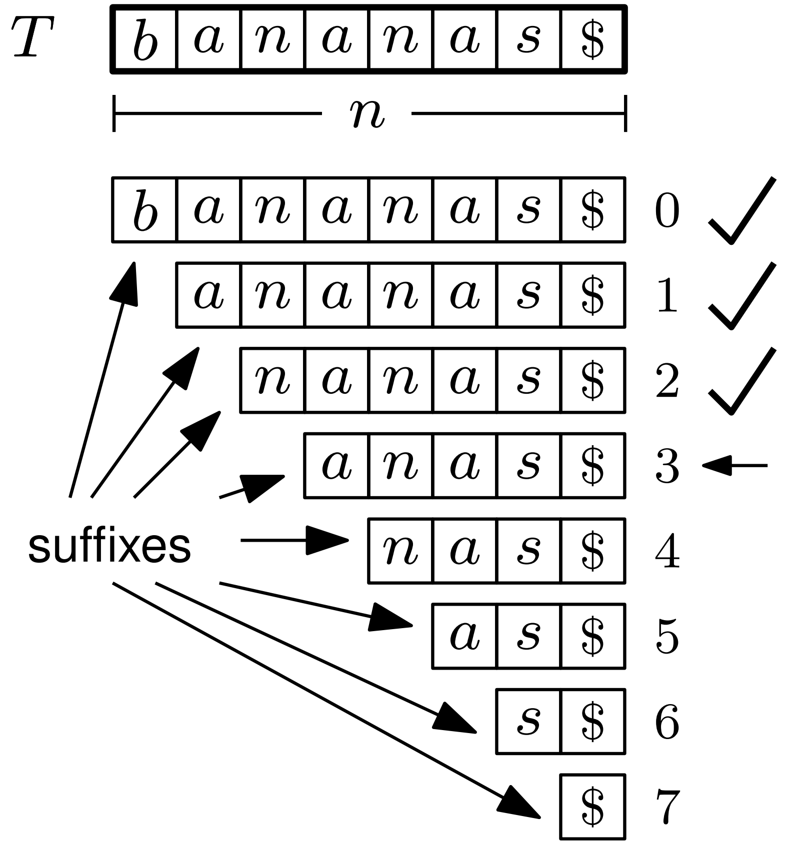
  *(as if you were matching a pattern)*

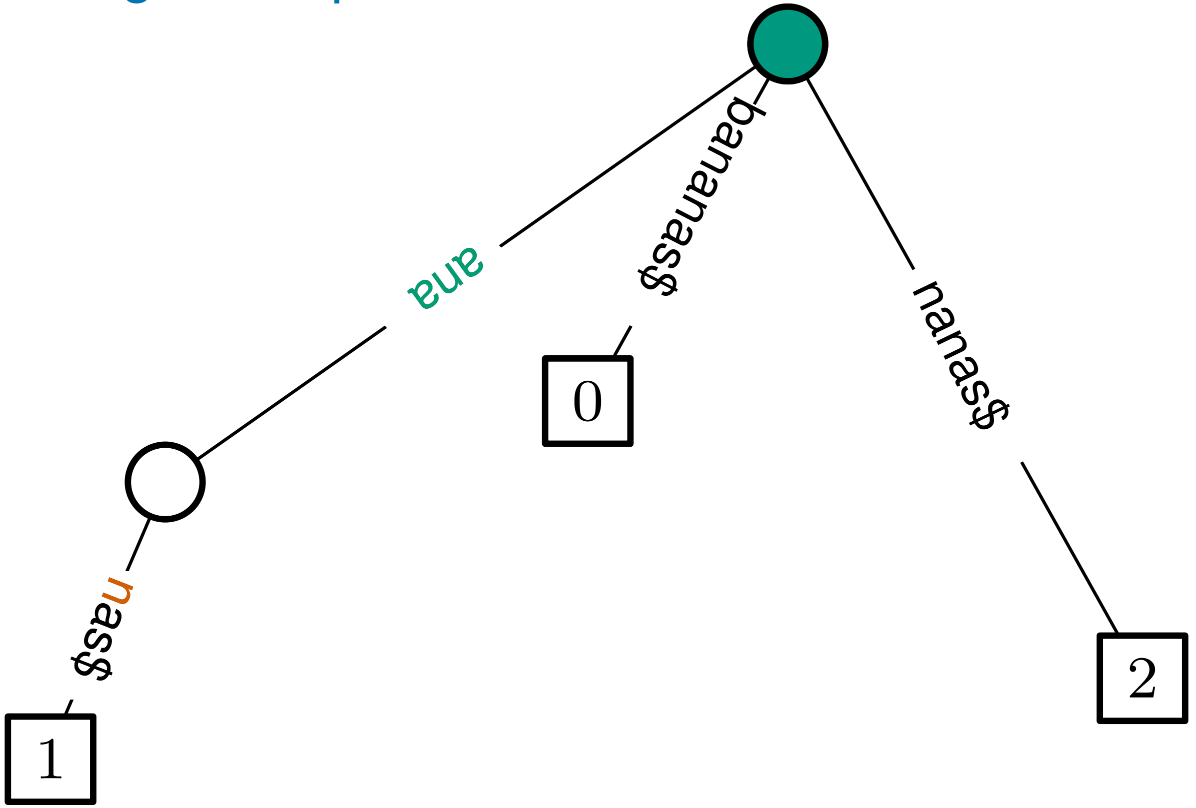
- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should
*never* actually
do it like this

# Naively constructing a compacted suffix tree



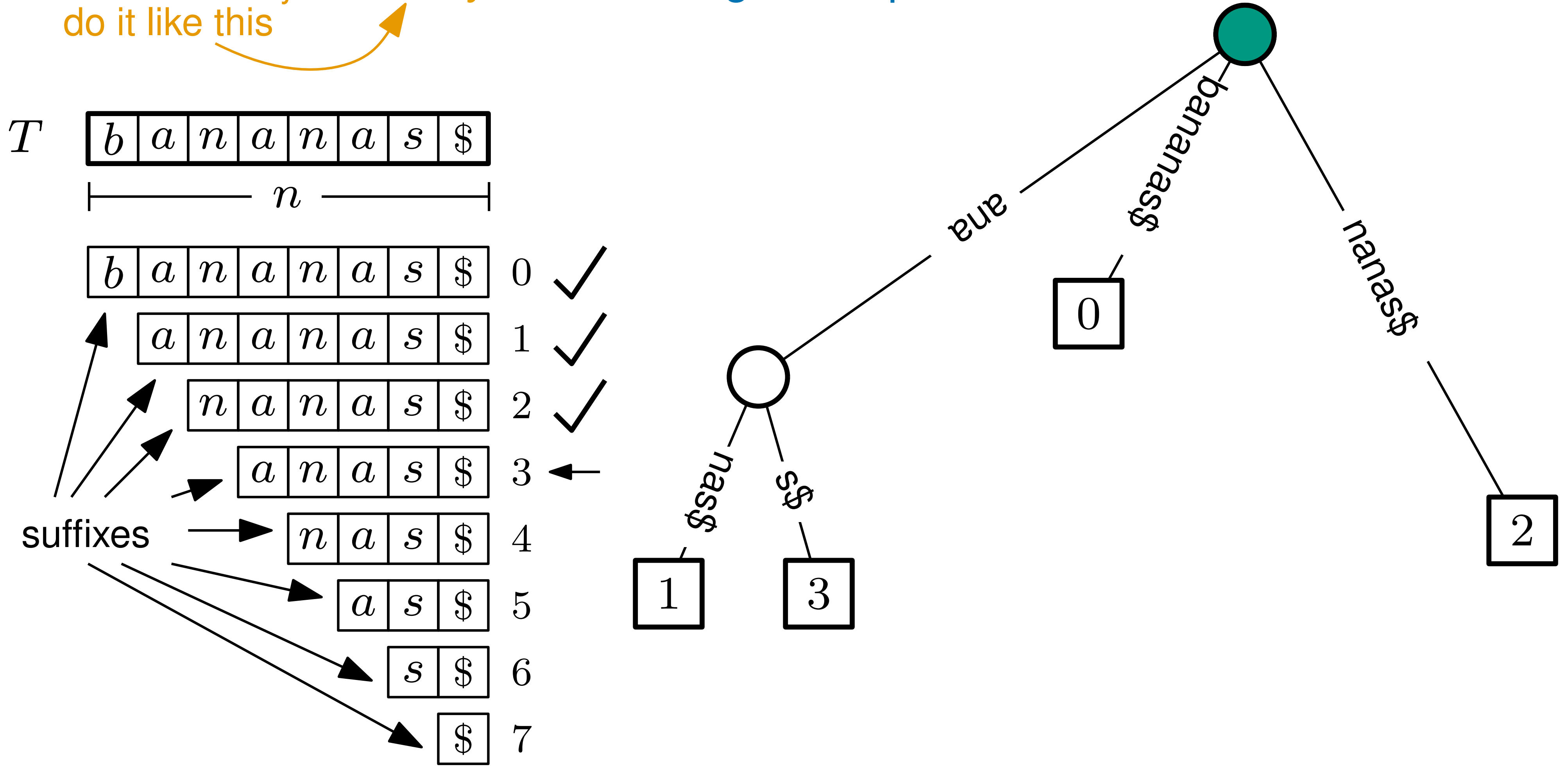


Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*

you should
*never* actually
do it like this

# Naively constructing a compacted suffix tree

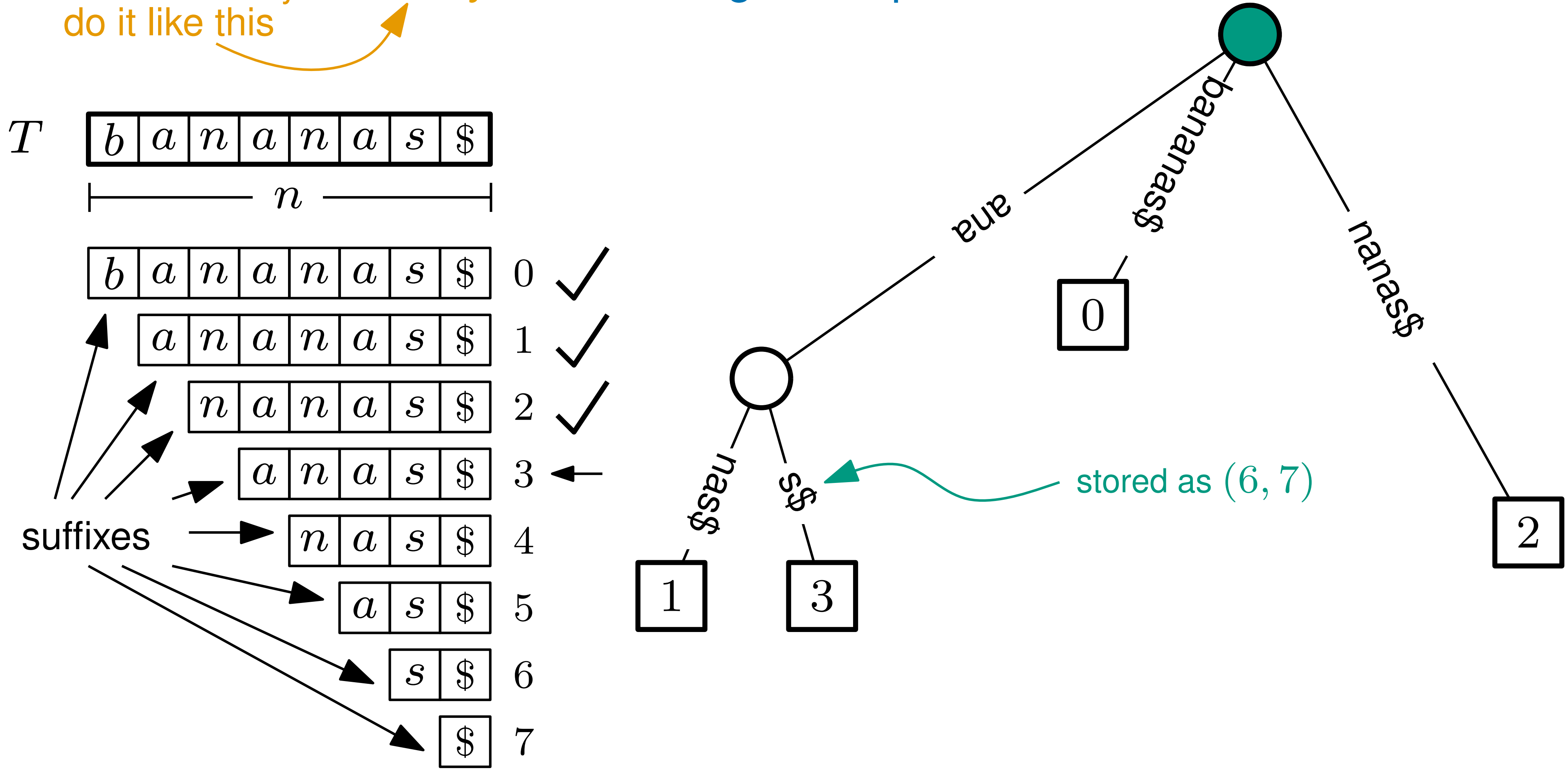


Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree

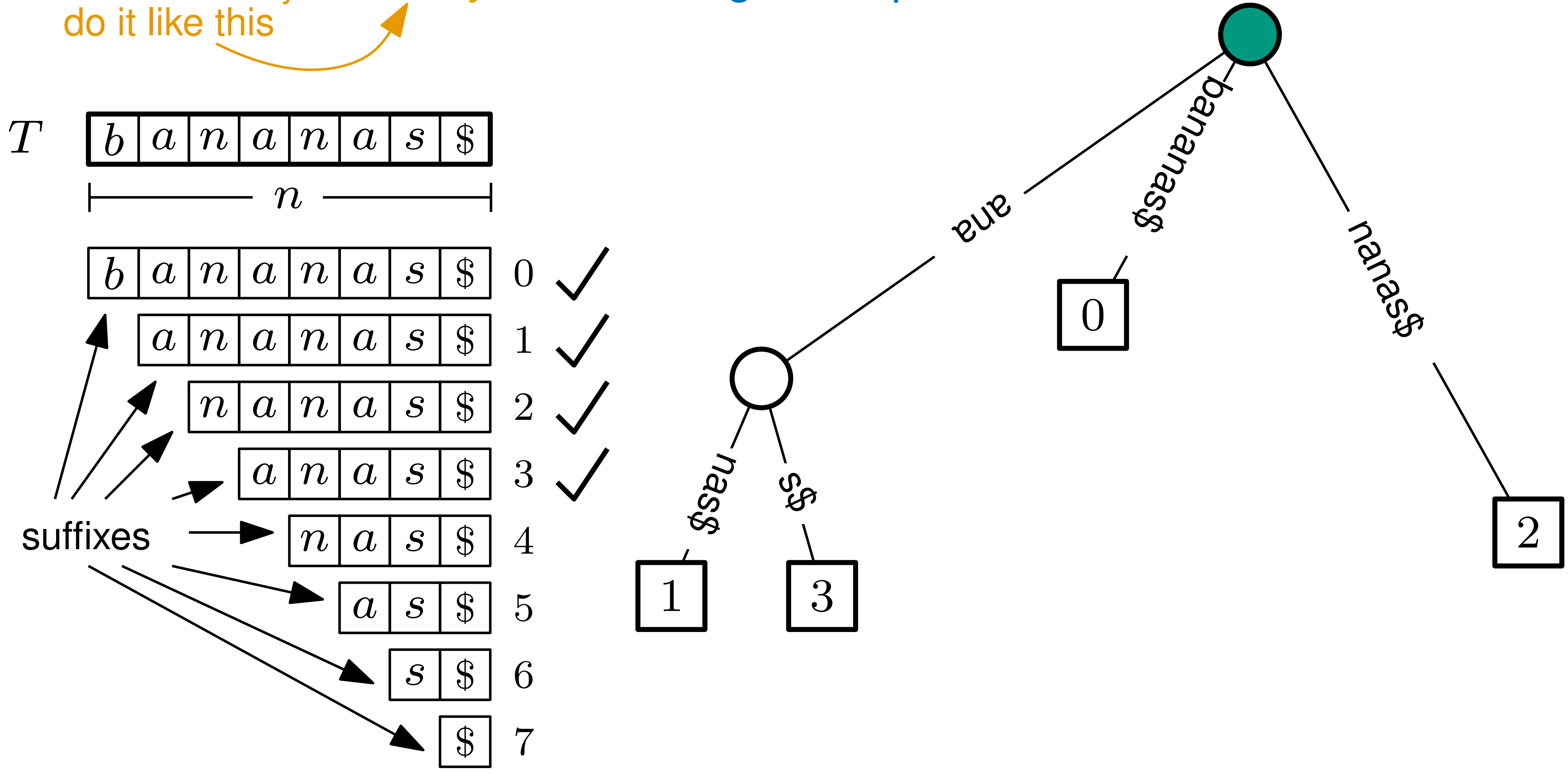


Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

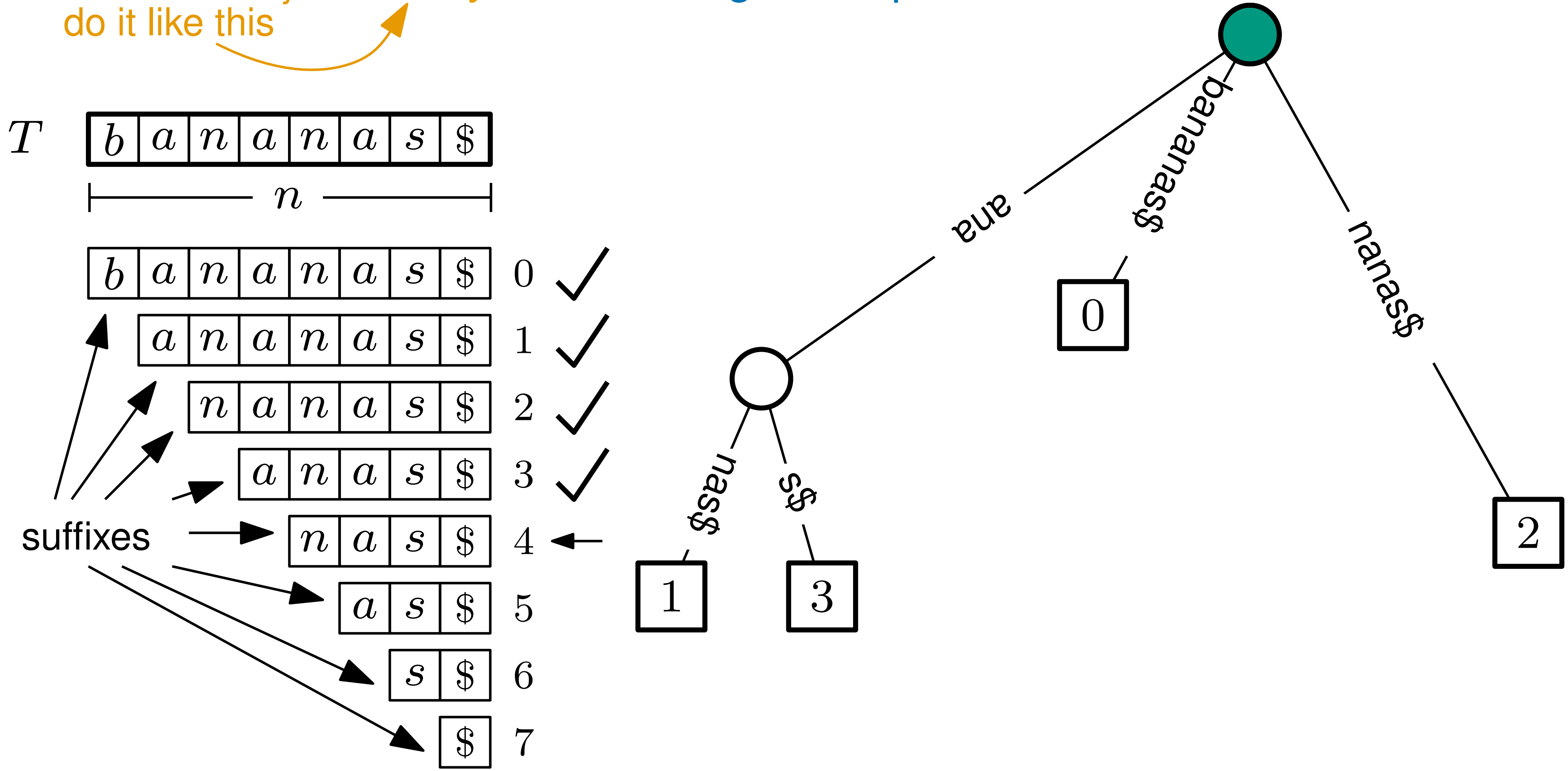
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should *never* actually do it like this



---

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

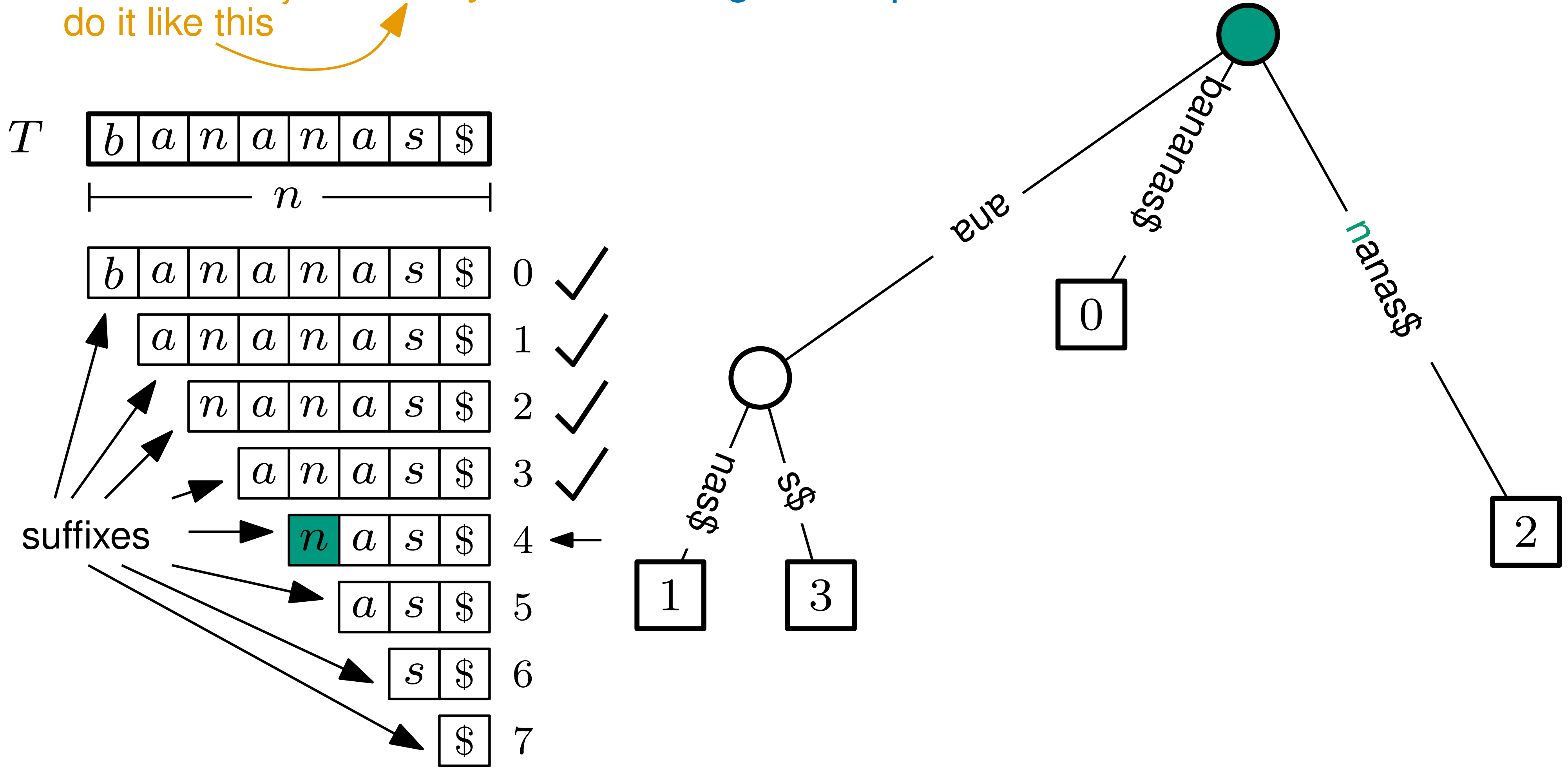
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should
*never* actually
do it like this



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

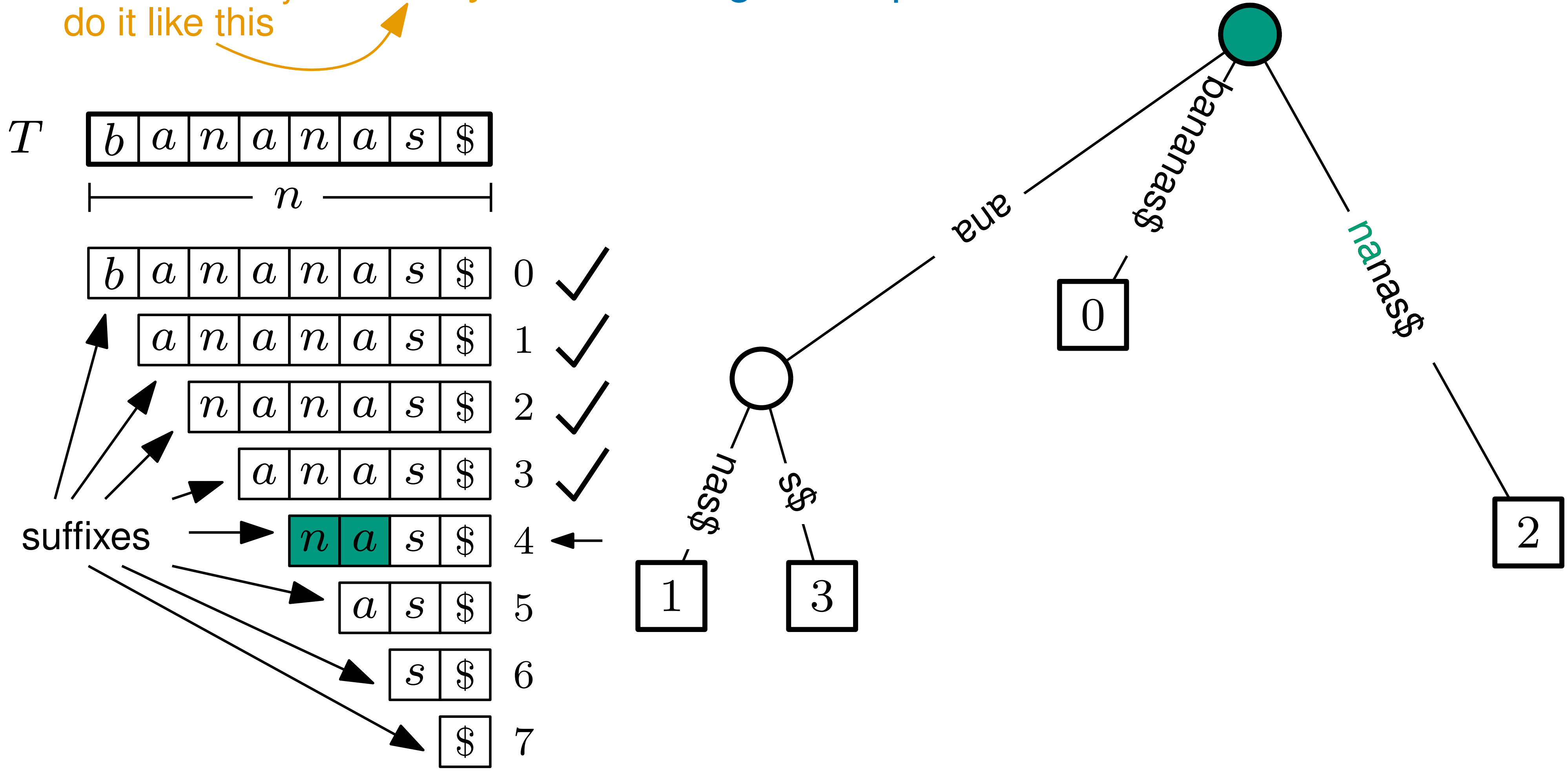
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should
*never* actually
do it like this

# Naively constructing a compacted suffix tree

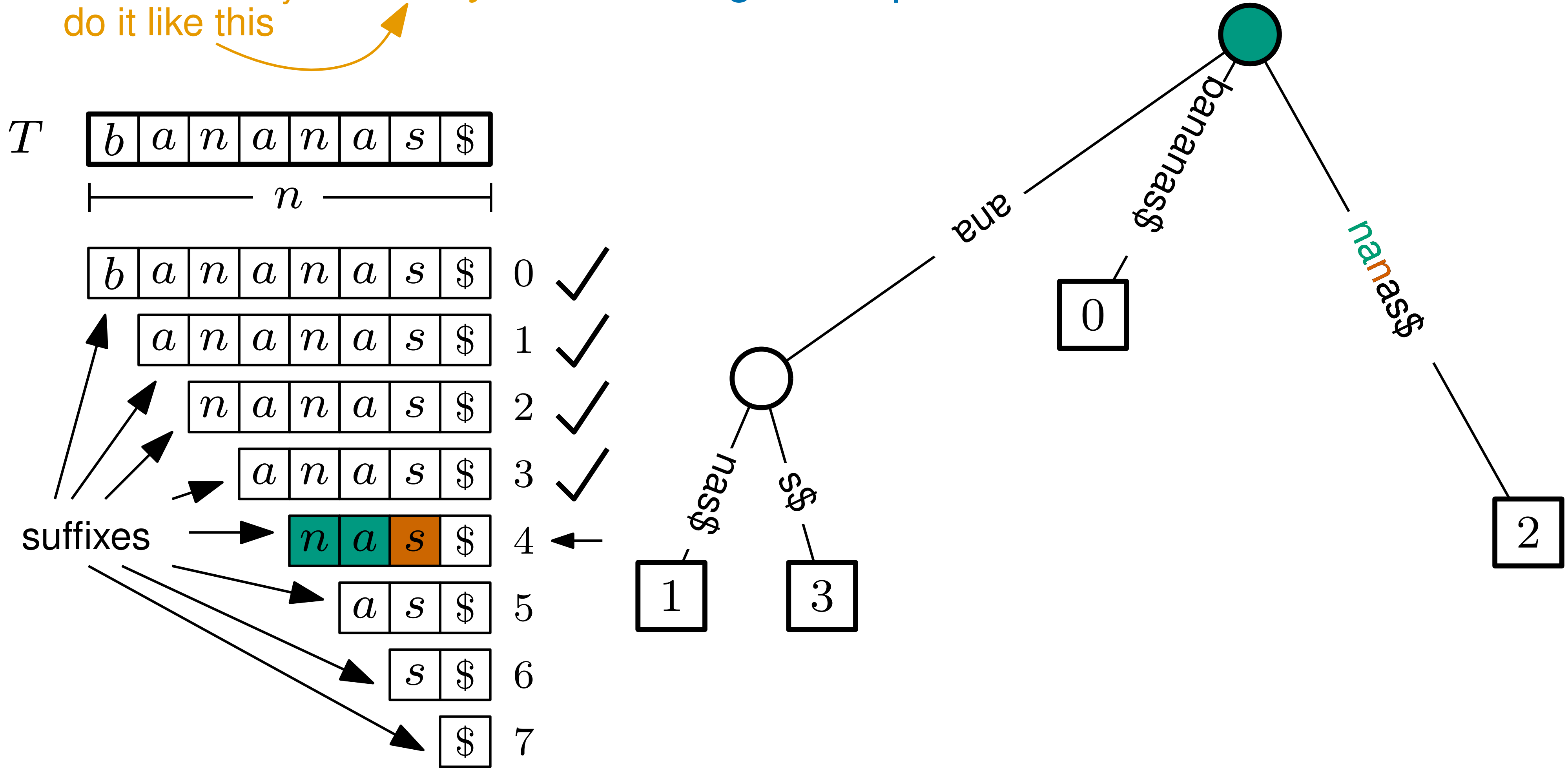


Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should *never* actually do it like this

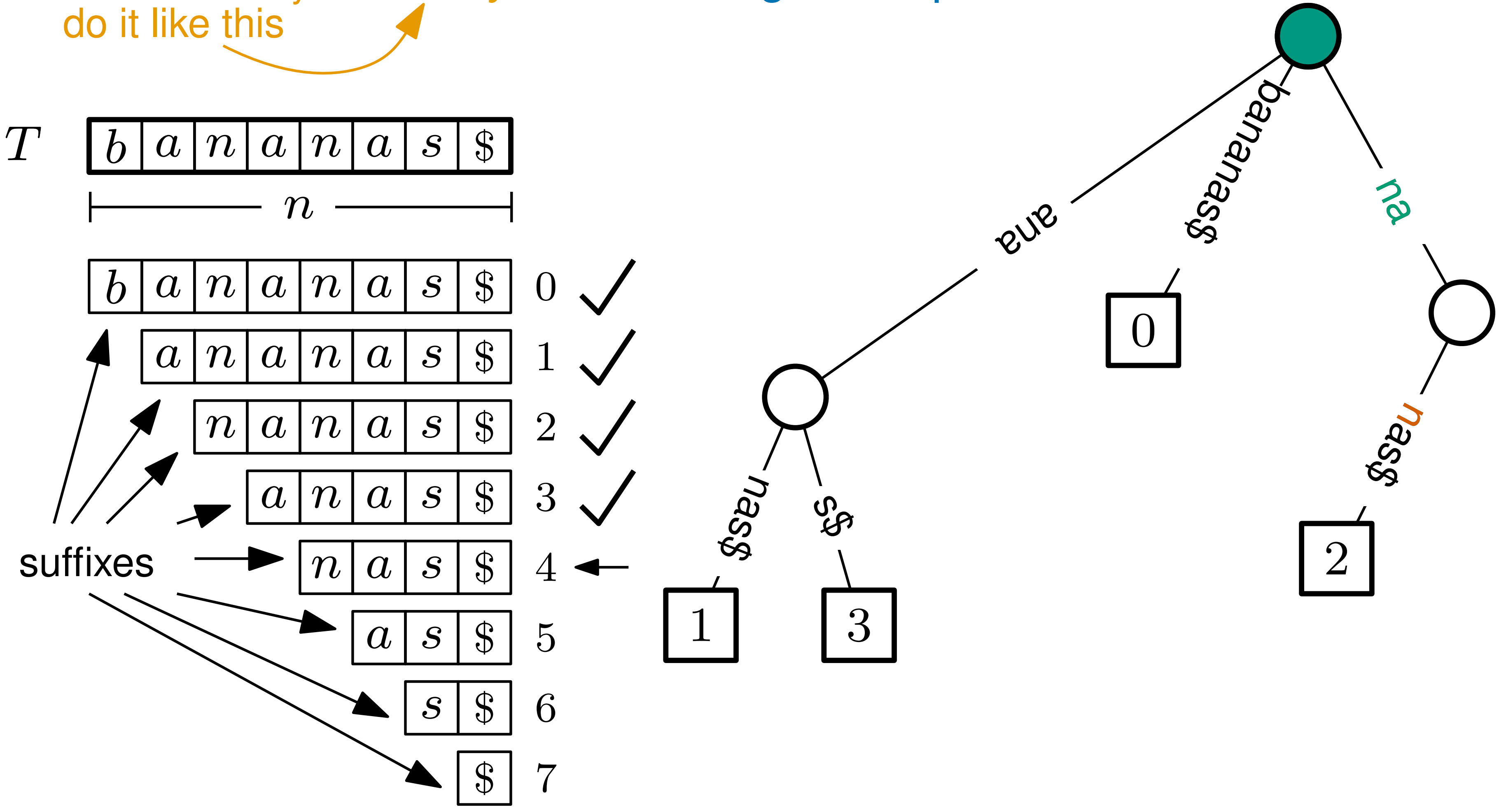


Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*

you should
*never* actually
do it like this

# Naively constructing a compacted suffix tree

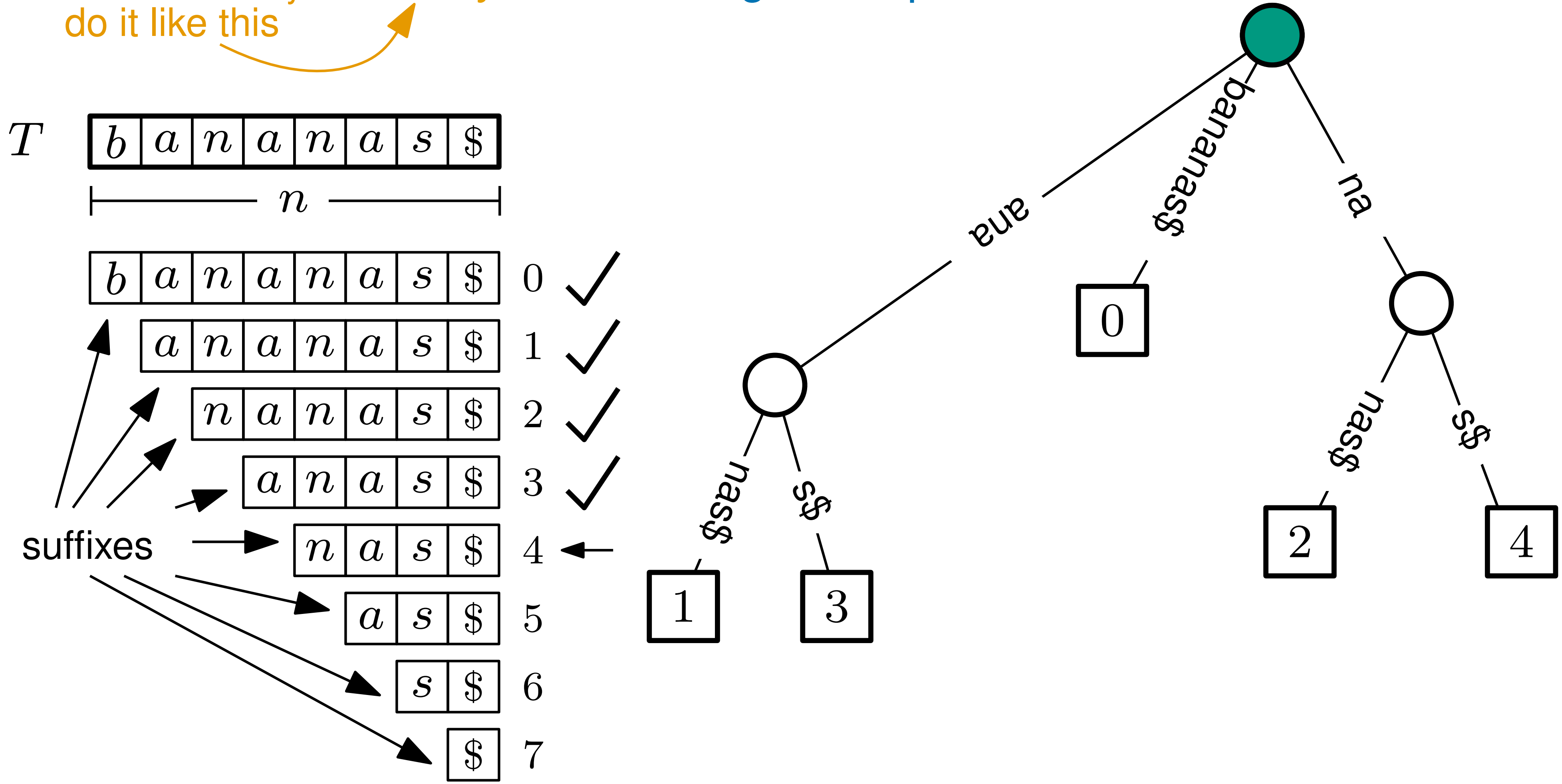


Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

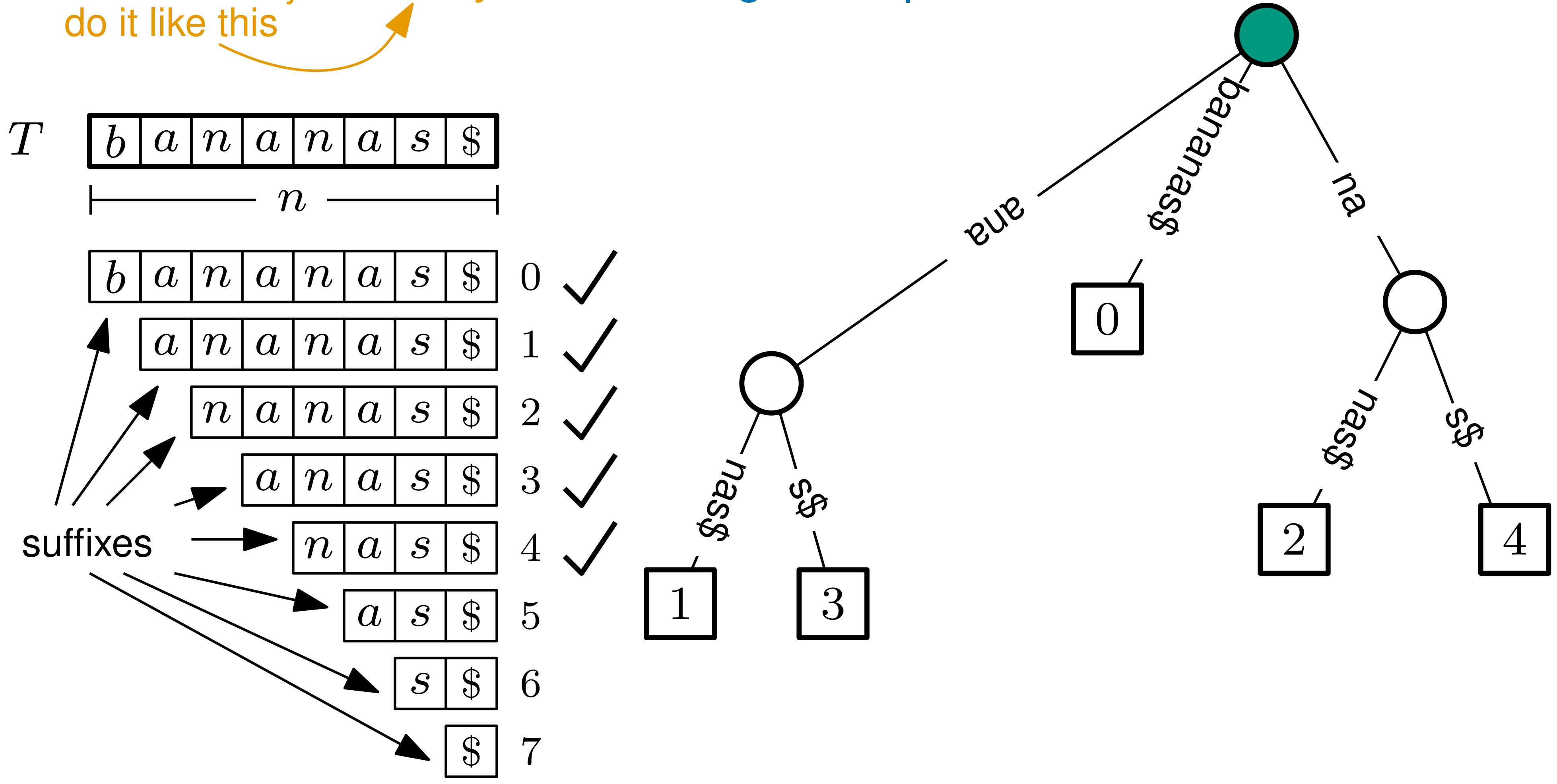
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should *never* actually do it like this



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

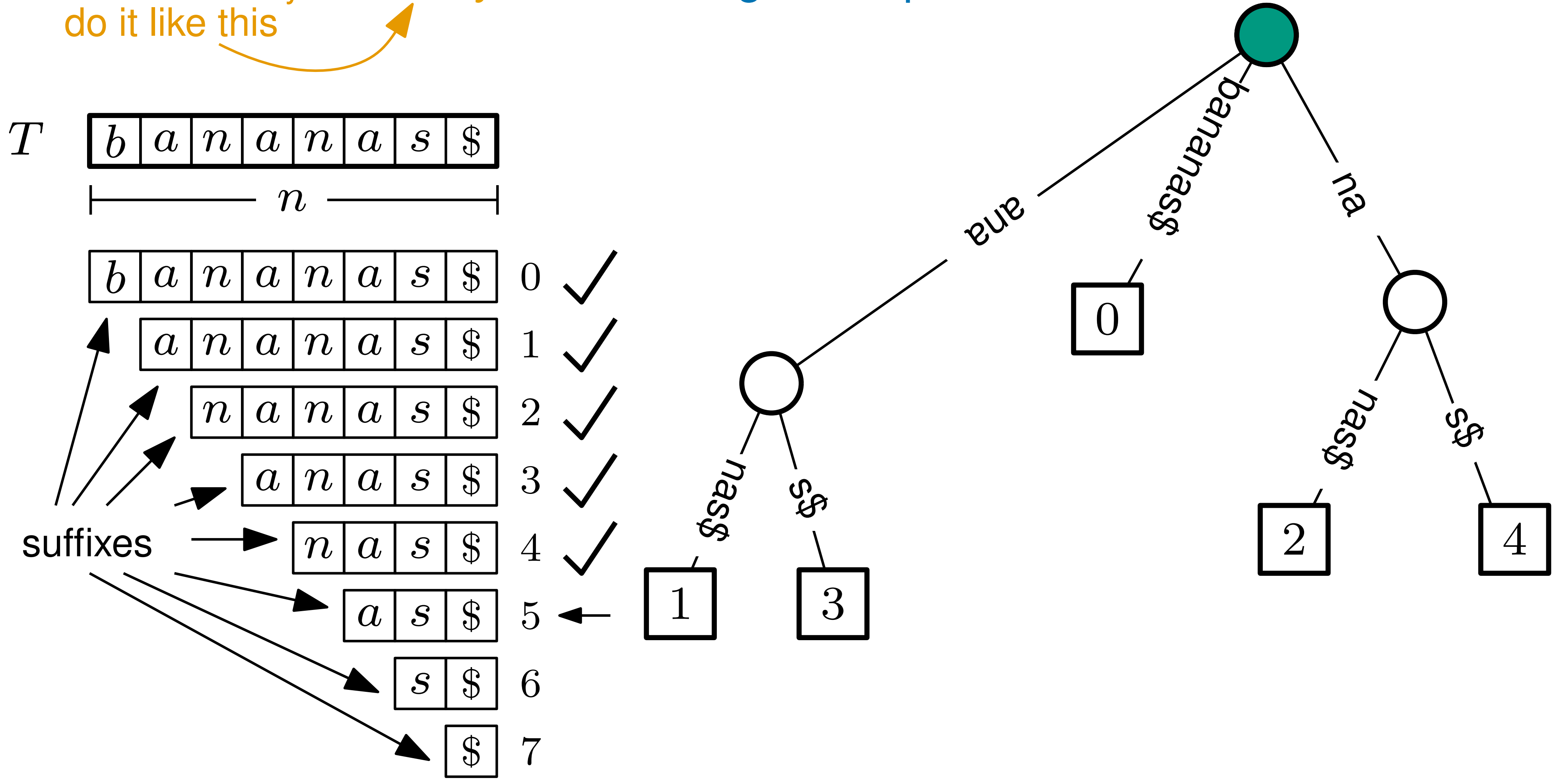
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree

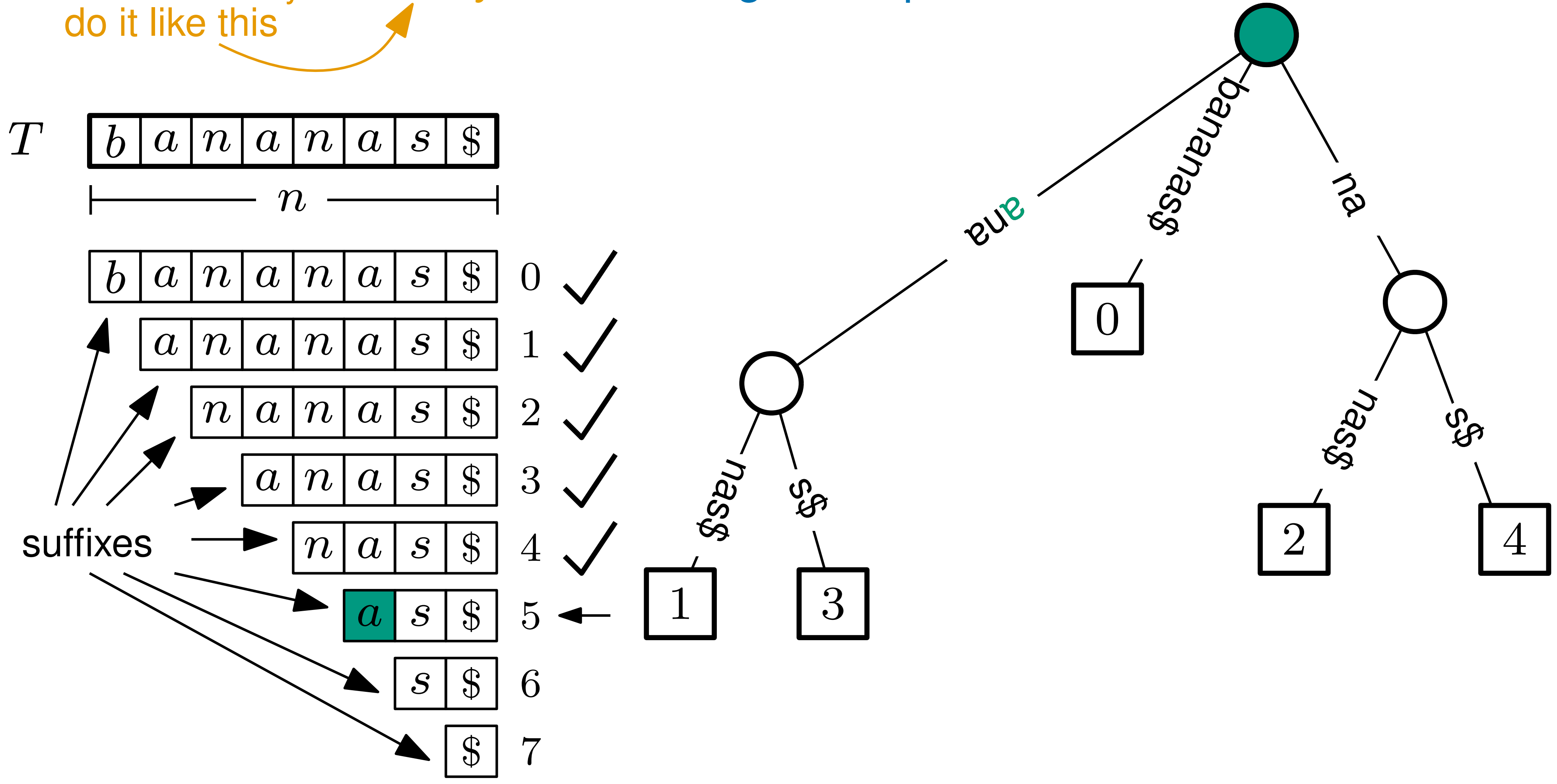


Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

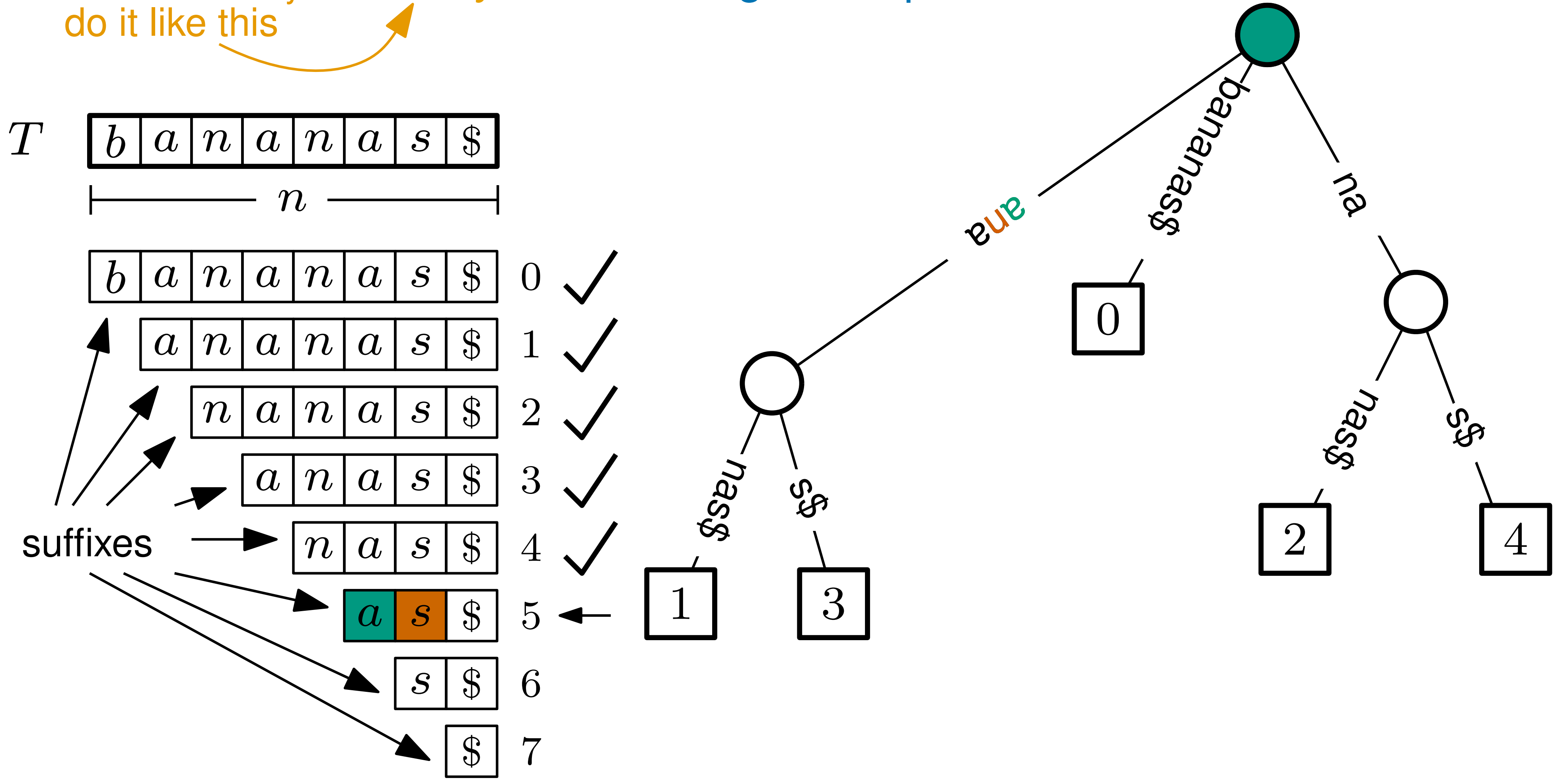
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should
*never* actually
do it like this

# Naively constructing a compacted suffix tree



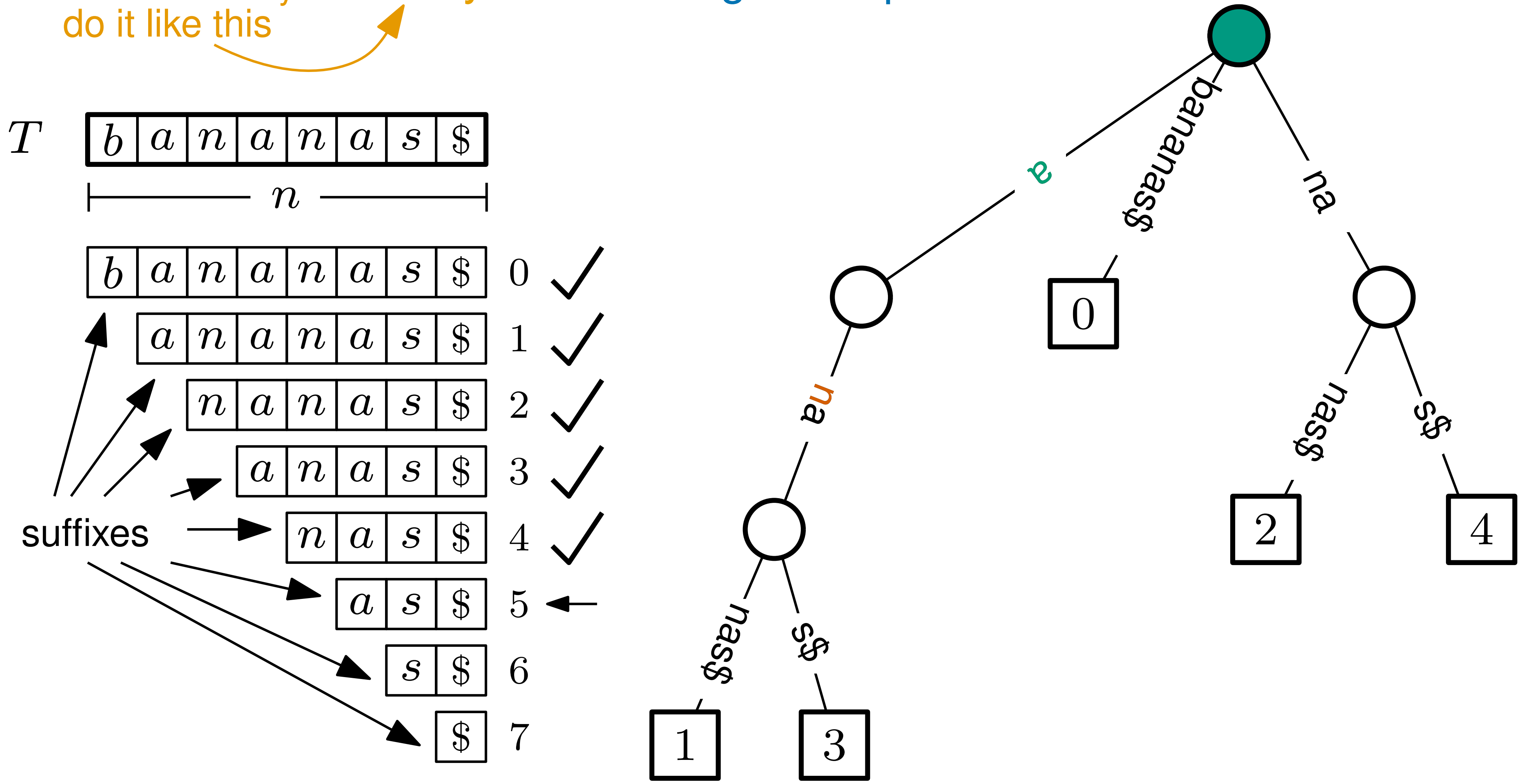
---

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

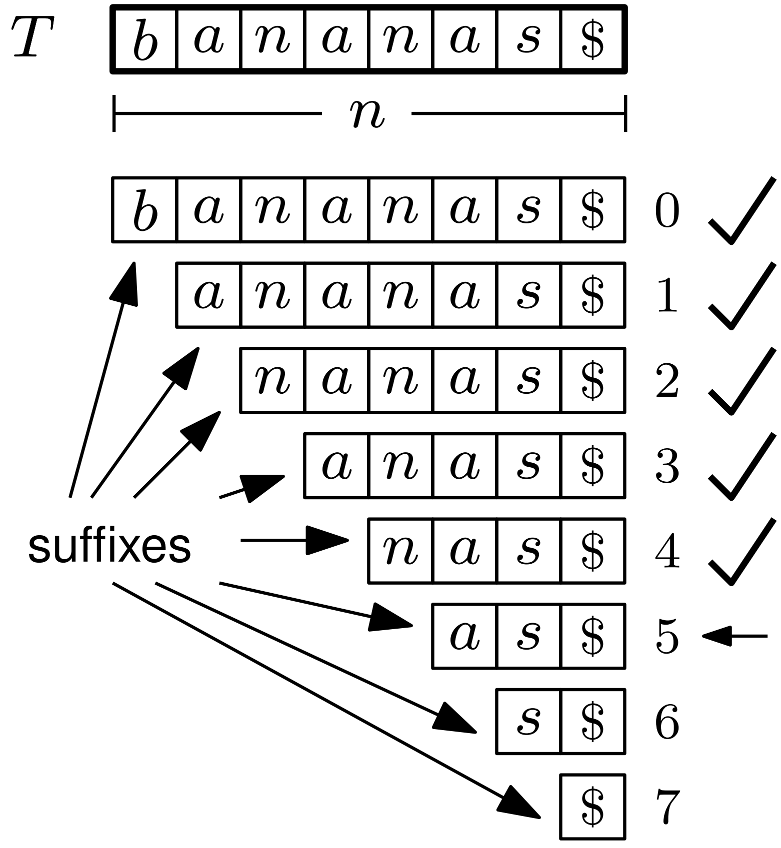
  *(as if you were matching a pattern)*

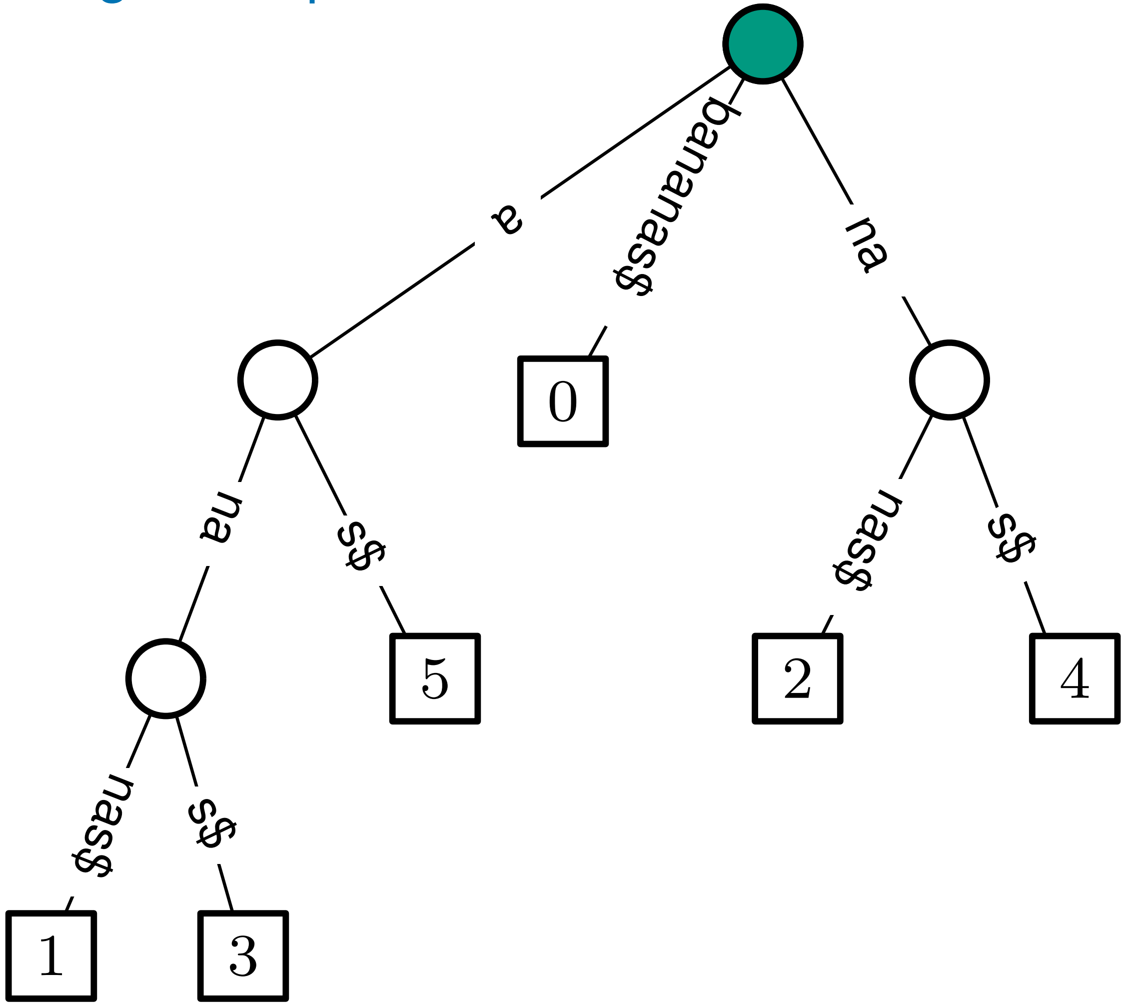
- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should
*never* actually
do it like this

# Naively constructing a compacted suffix tree





Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

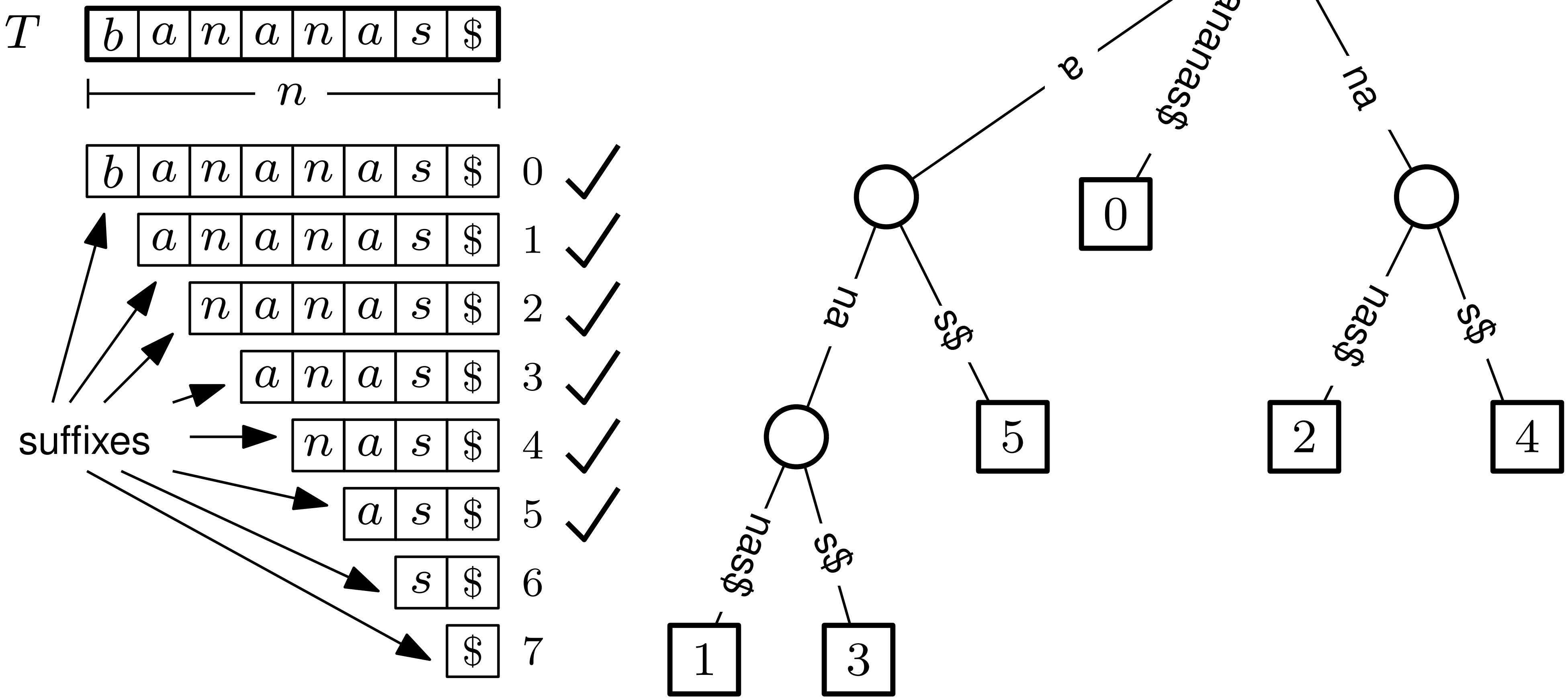
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should *never* actually do it like this



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

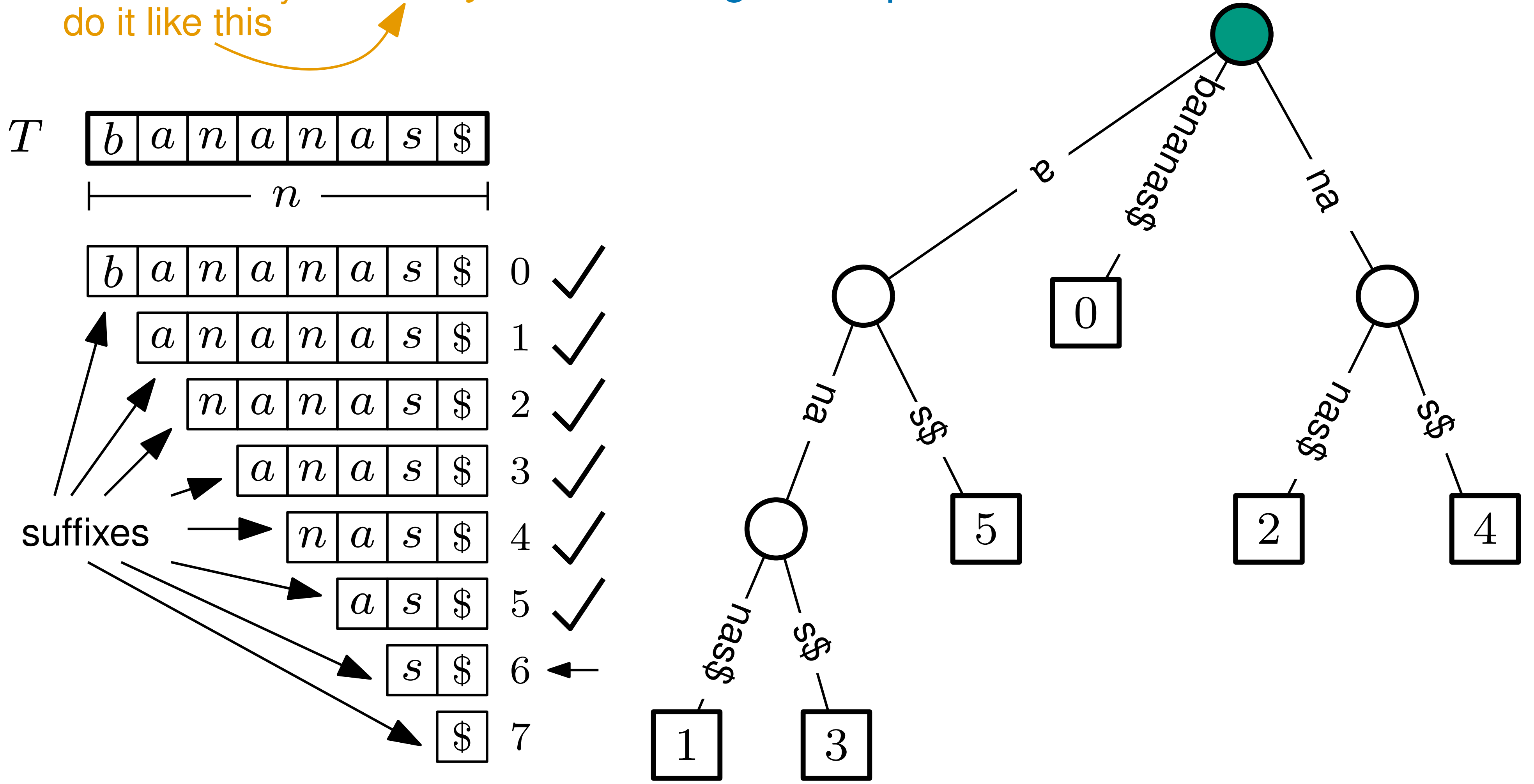
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should
*never* actually
do it like this



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

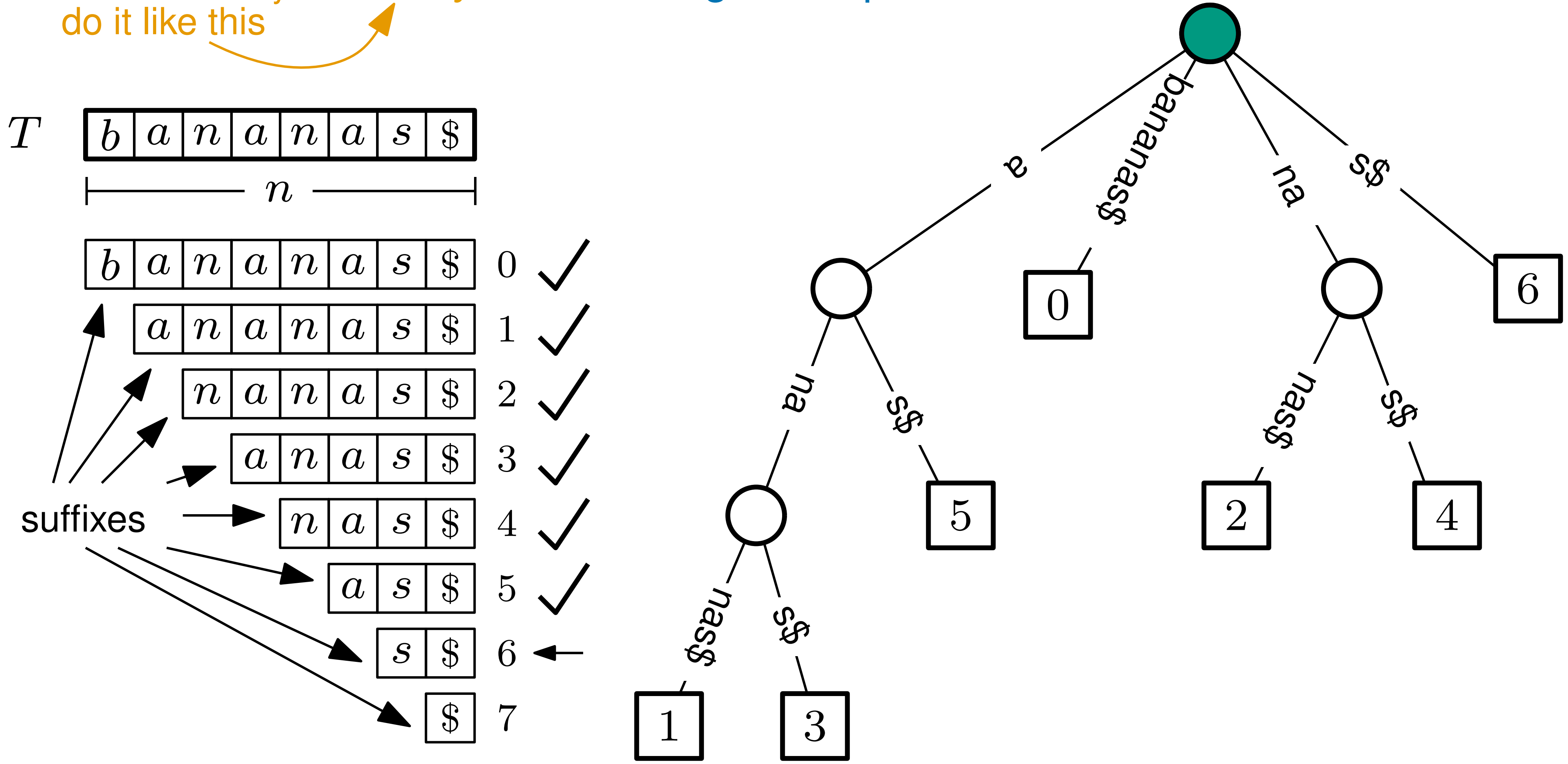
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree

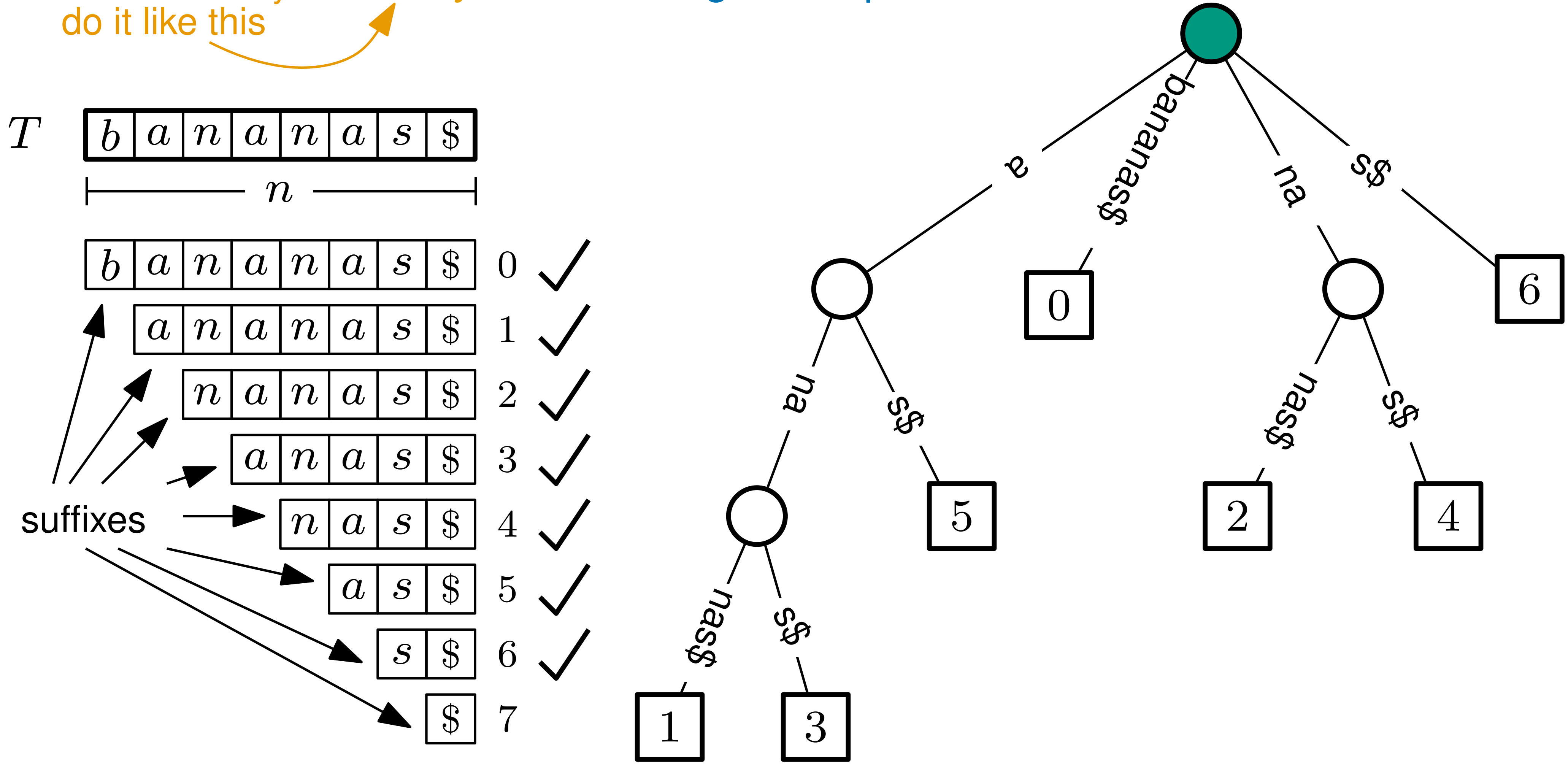


Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should *never* actually do it like this



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

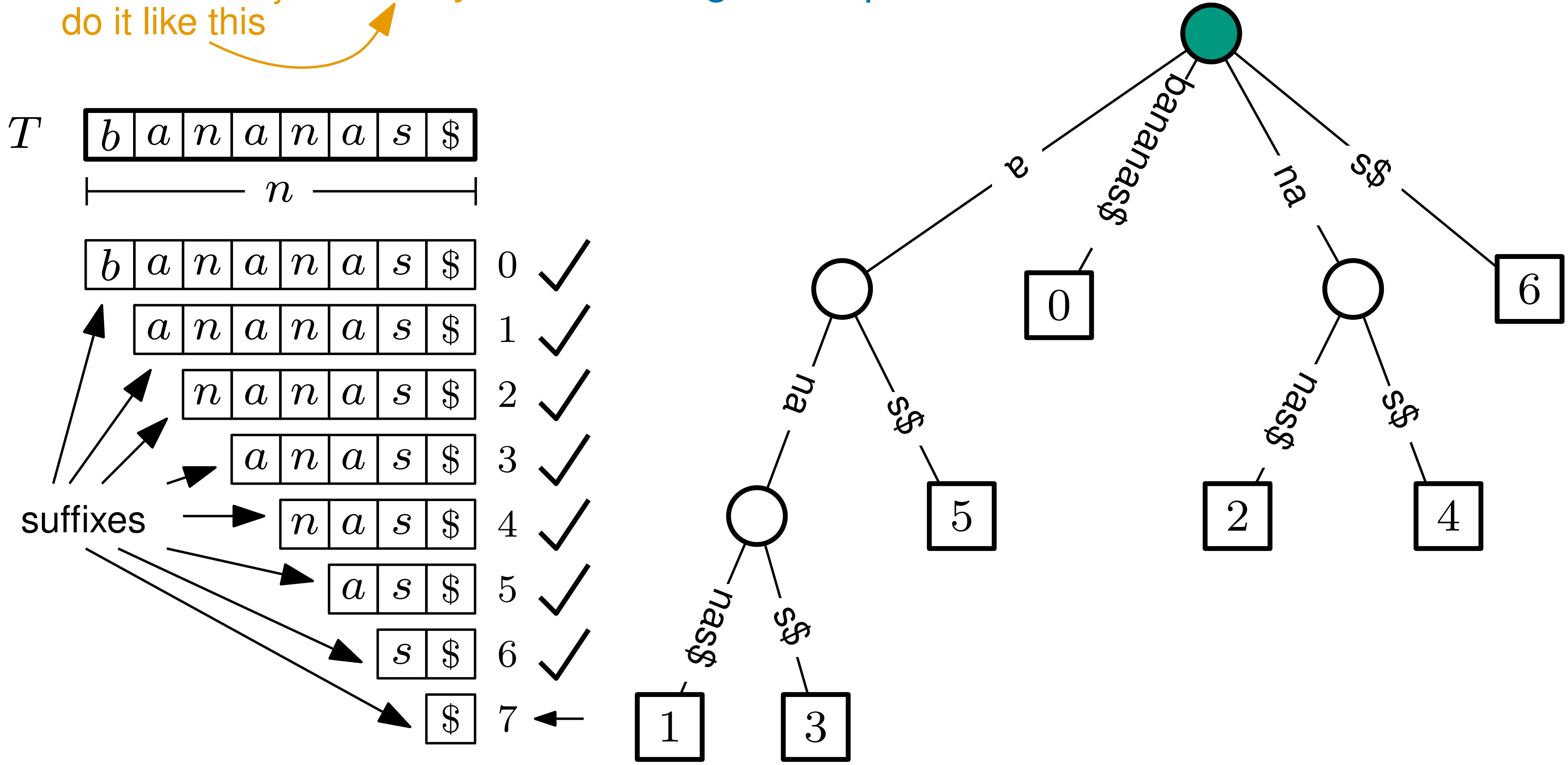
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

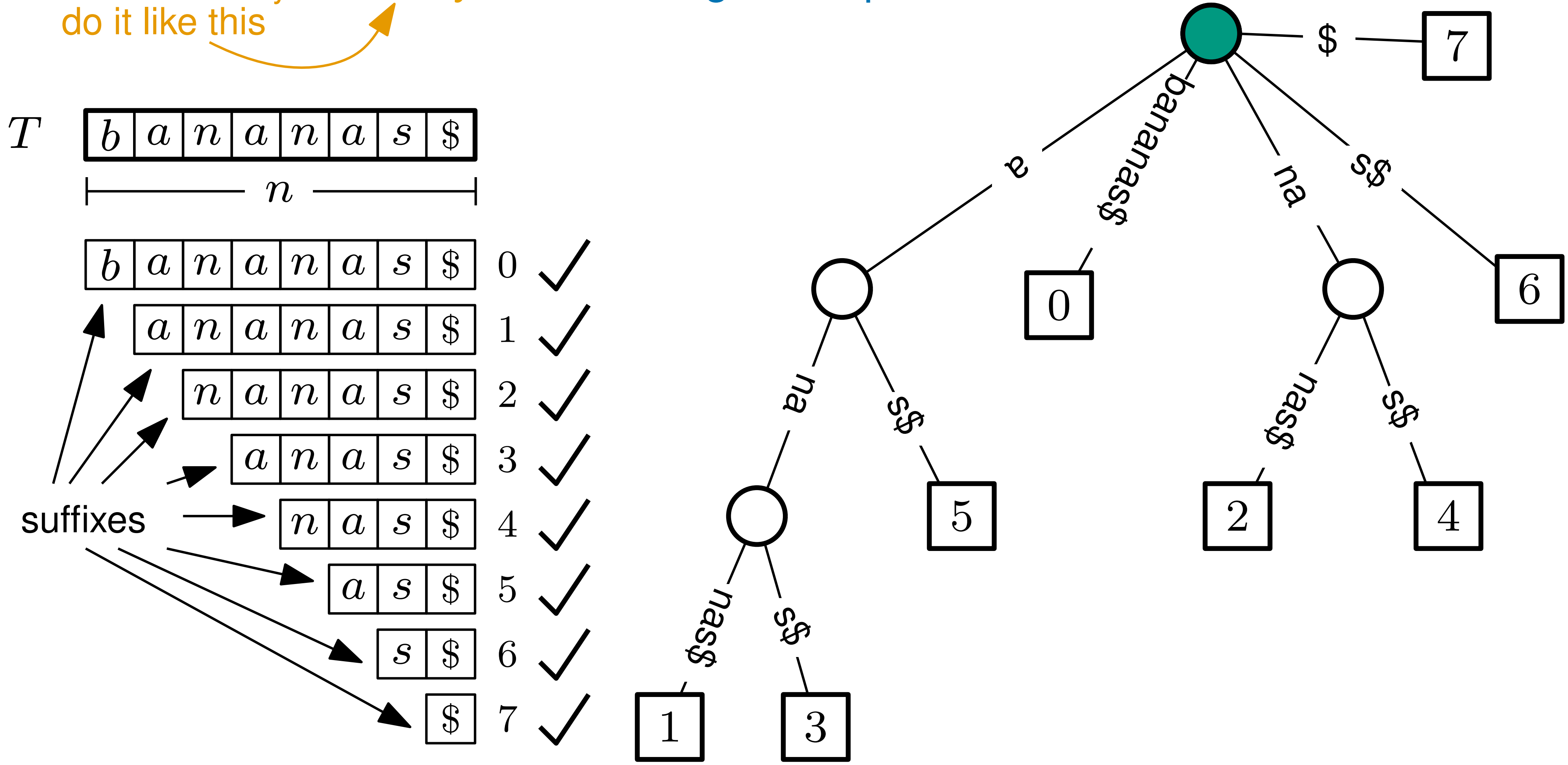
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

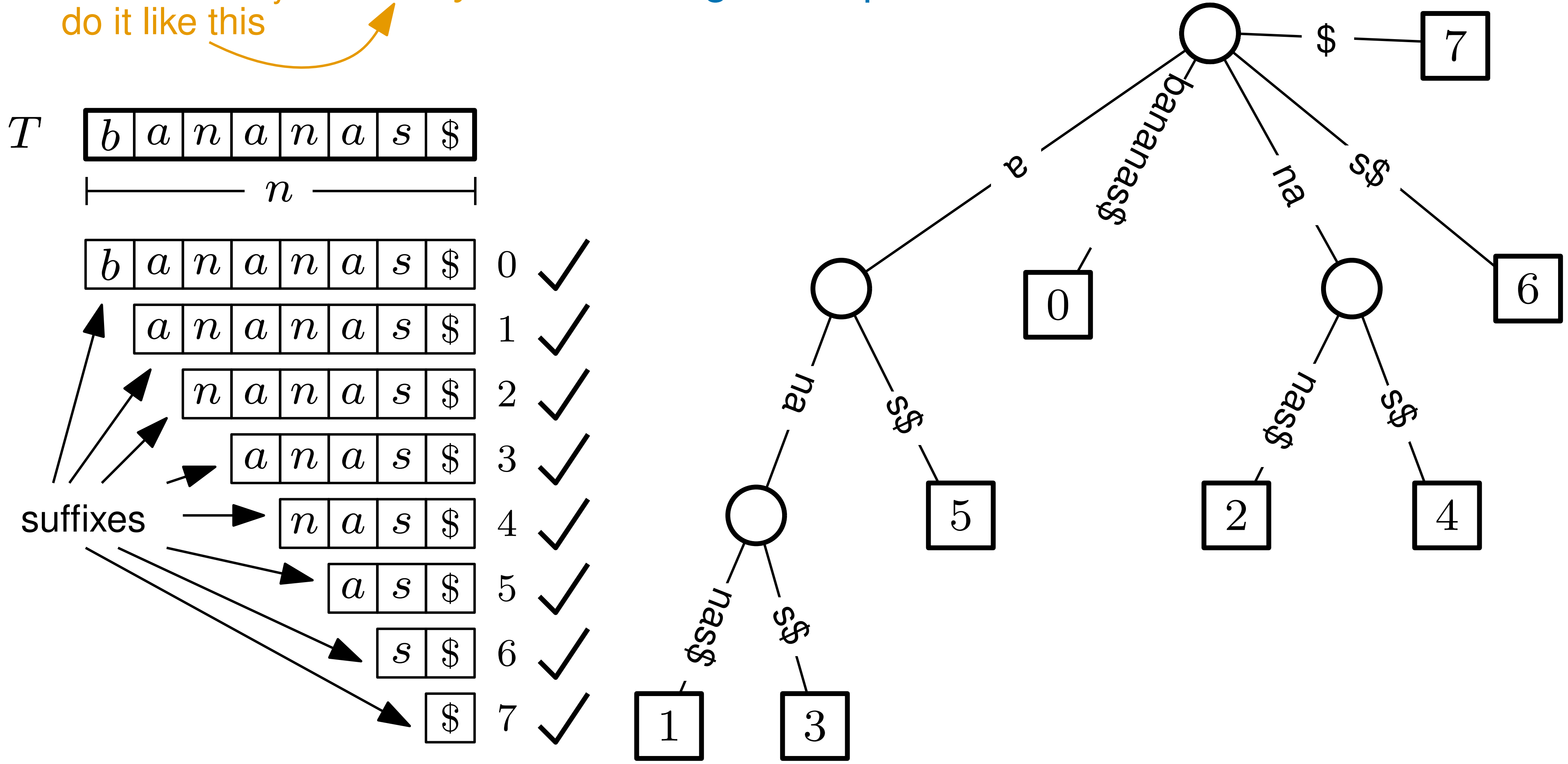
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

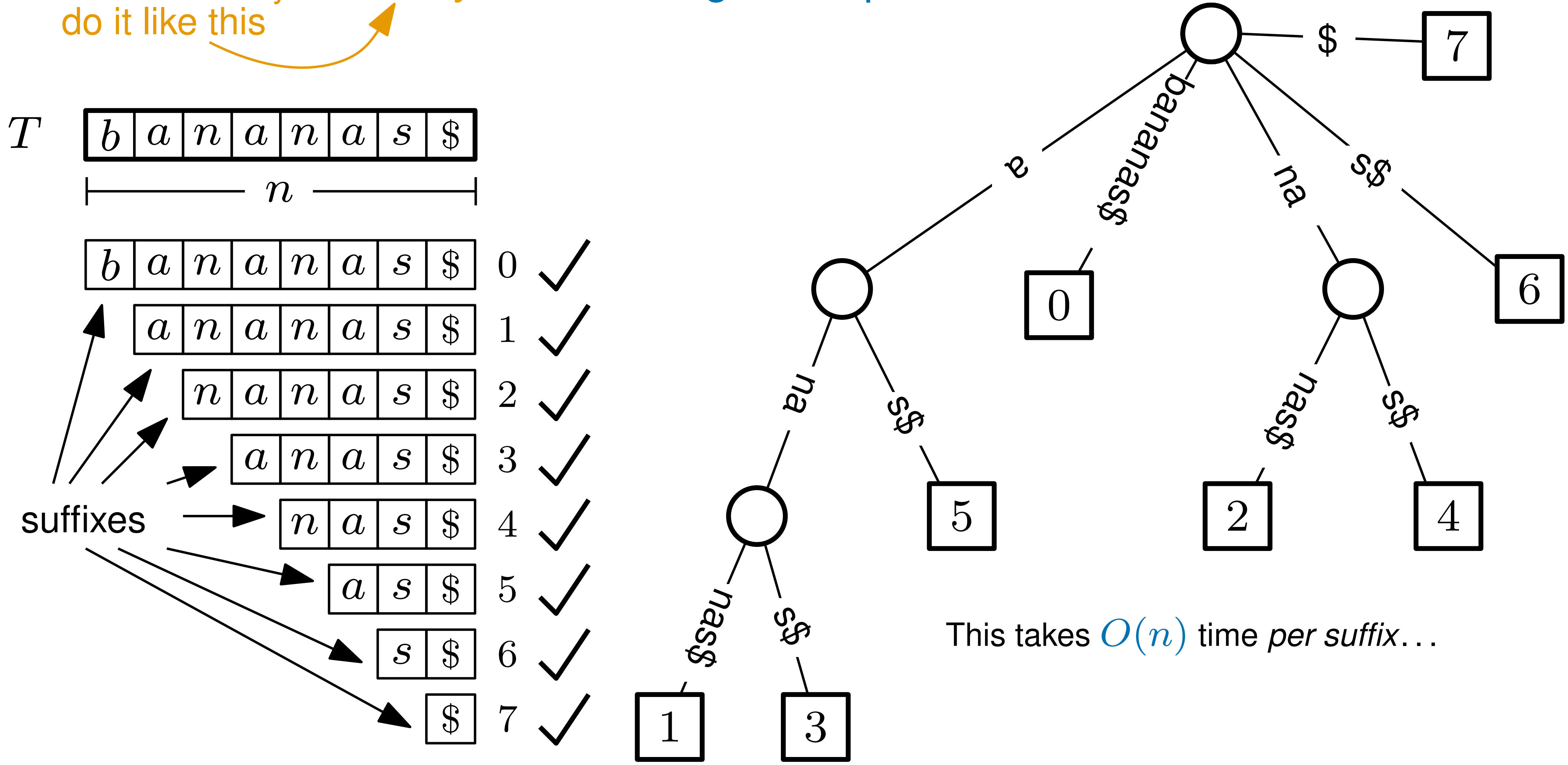
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

you should *never* actually do it like this

# Naively constructing a compacted suffix tree



This takes $O(n)$ time *per suffix*...

---

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree

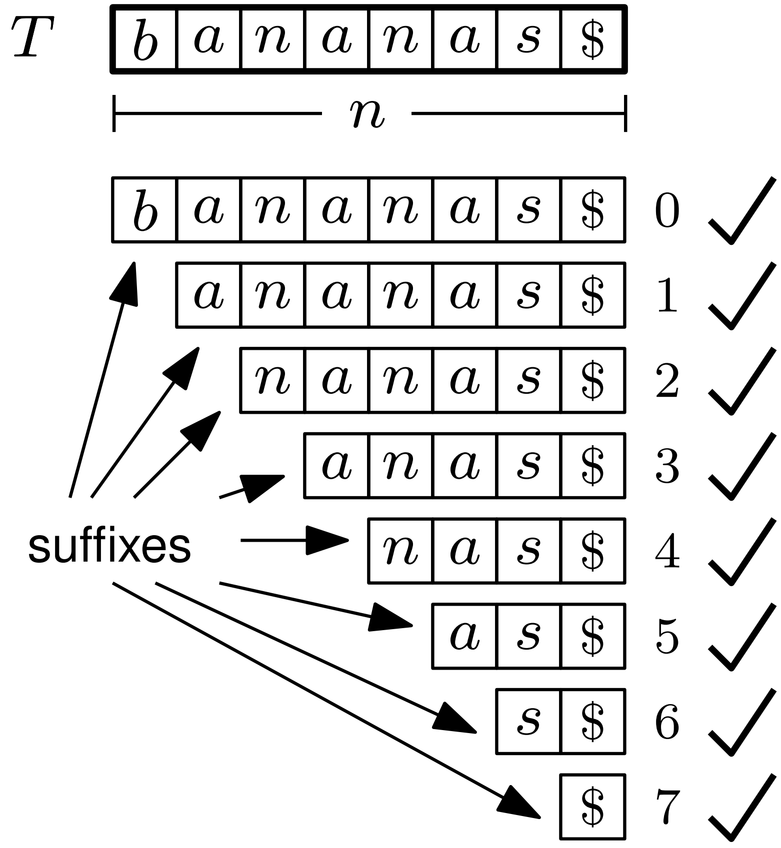
  *(as if you were matching a pattern)*

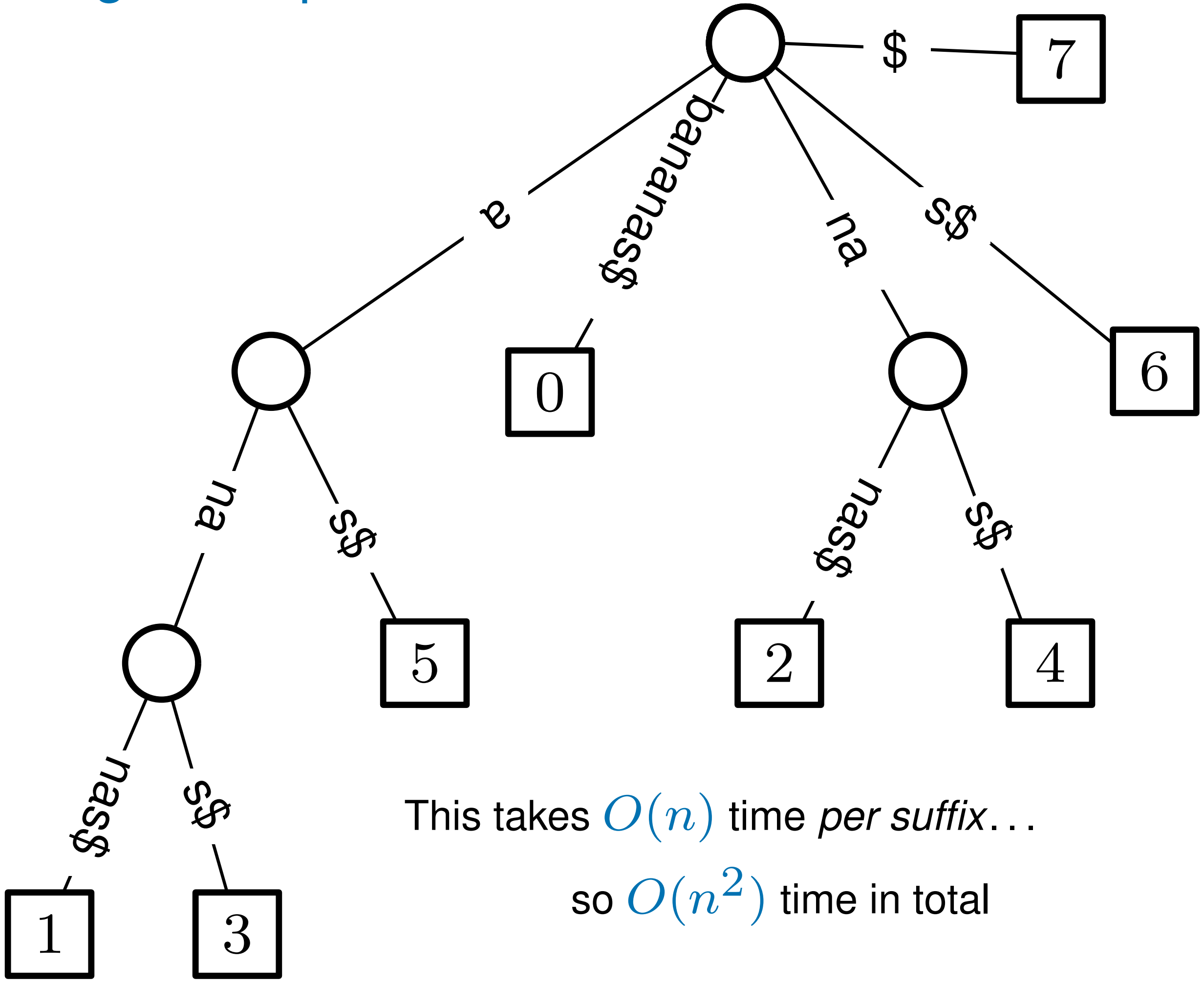
- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

# Naively constructing a compacted suffix tree

you should *never* actually do it like this





This takes $O(n)$ time *per suffix*...

so $O(n^2)$ time in total

Insert the suffixes one at a time (longest first)

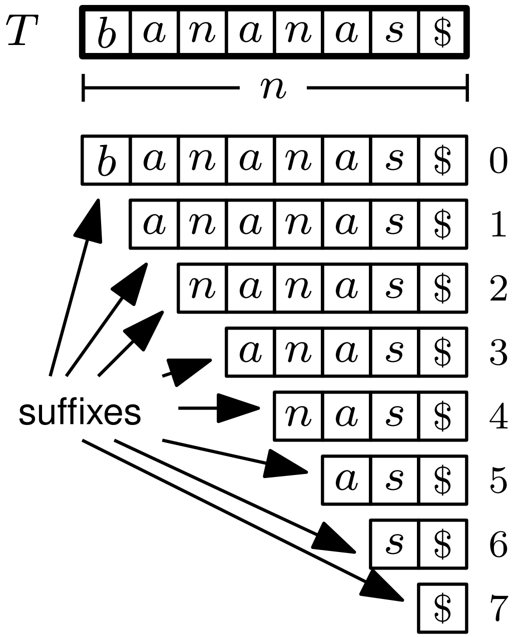
- Search for the new suffix in the partial suffix tree

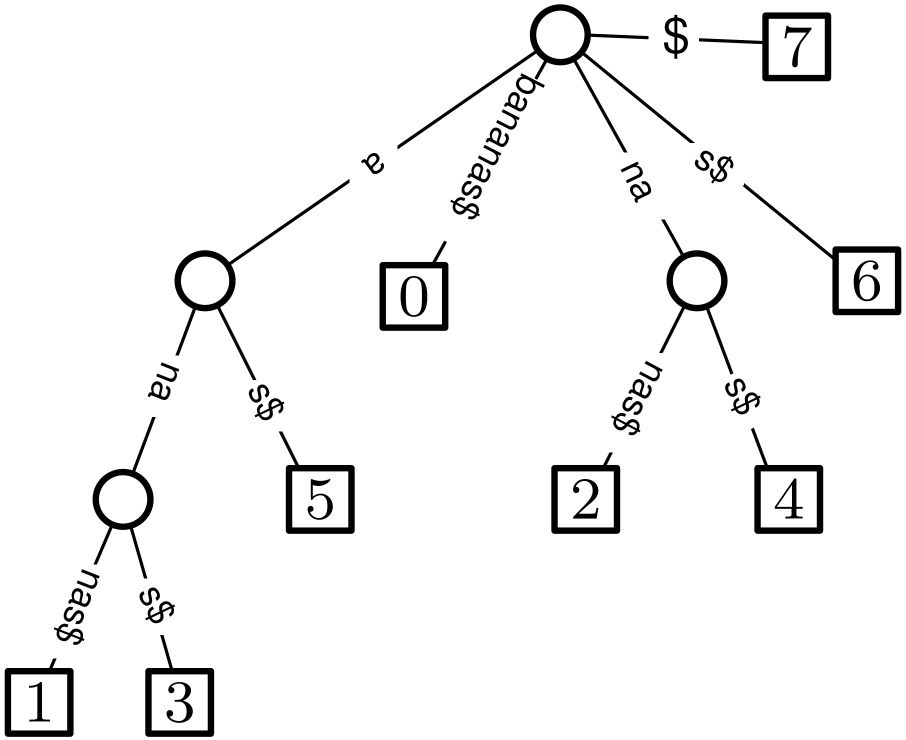
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*

# Suffix tree summary





- The (compacted) suffix tree of a (length $n$) text uses $O(n)$ space
- Finding all matches of a pattern $P$ of length $m$ takes $O(m + \text{occ})$

*where occ is the number of matches*

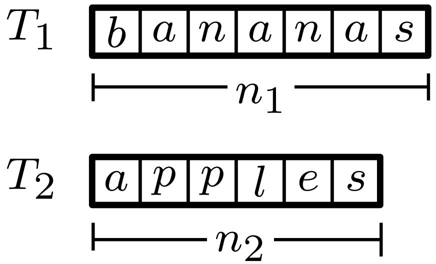
- Suffix trees can be built in $O(n)$ time

*but we have only seen the $O(n^2)$ time method*

you should actually do it like this (or build a *suffix array* instead)

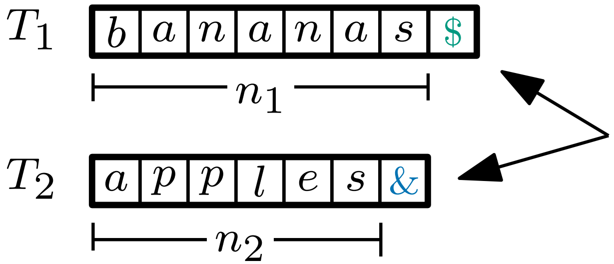
*we assumed that the alphabet contained a constant number of symbols*

# Multiple text indexing



How can we index multiple texts?

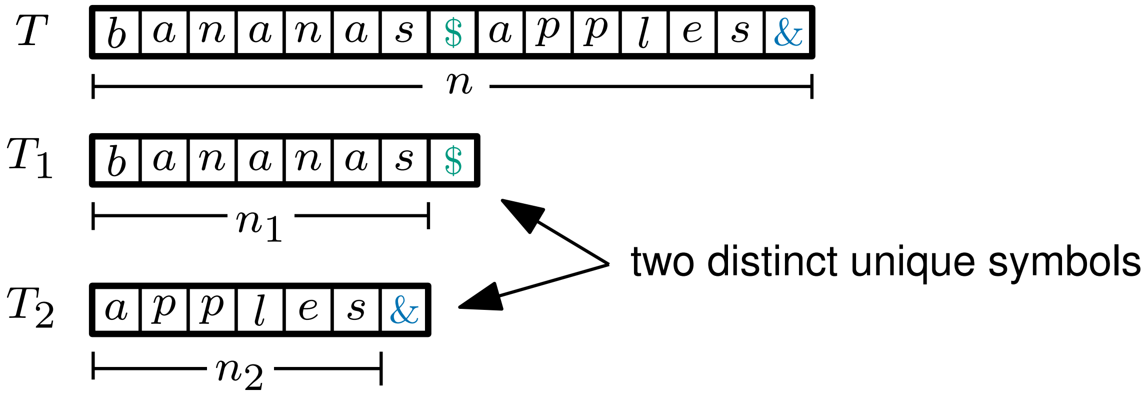
# Multiple text indexing



two distinct unique symbols

How can we index multiple texts?

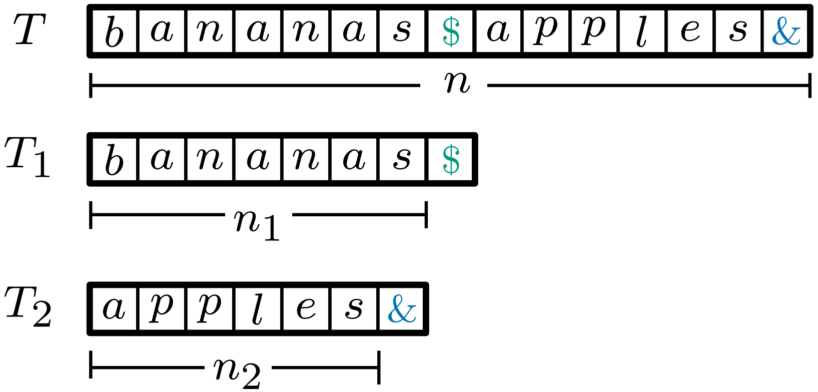
# Multiple text indexing



two distinct unique symbols

How can we index multiple texts?

# Multiple text indexing



$T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ | $\$$ | $a$ | $p$ | $p$ | $l$ | $e$ | $s$ | $\&$

$n$

$T_1$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ | $\$$

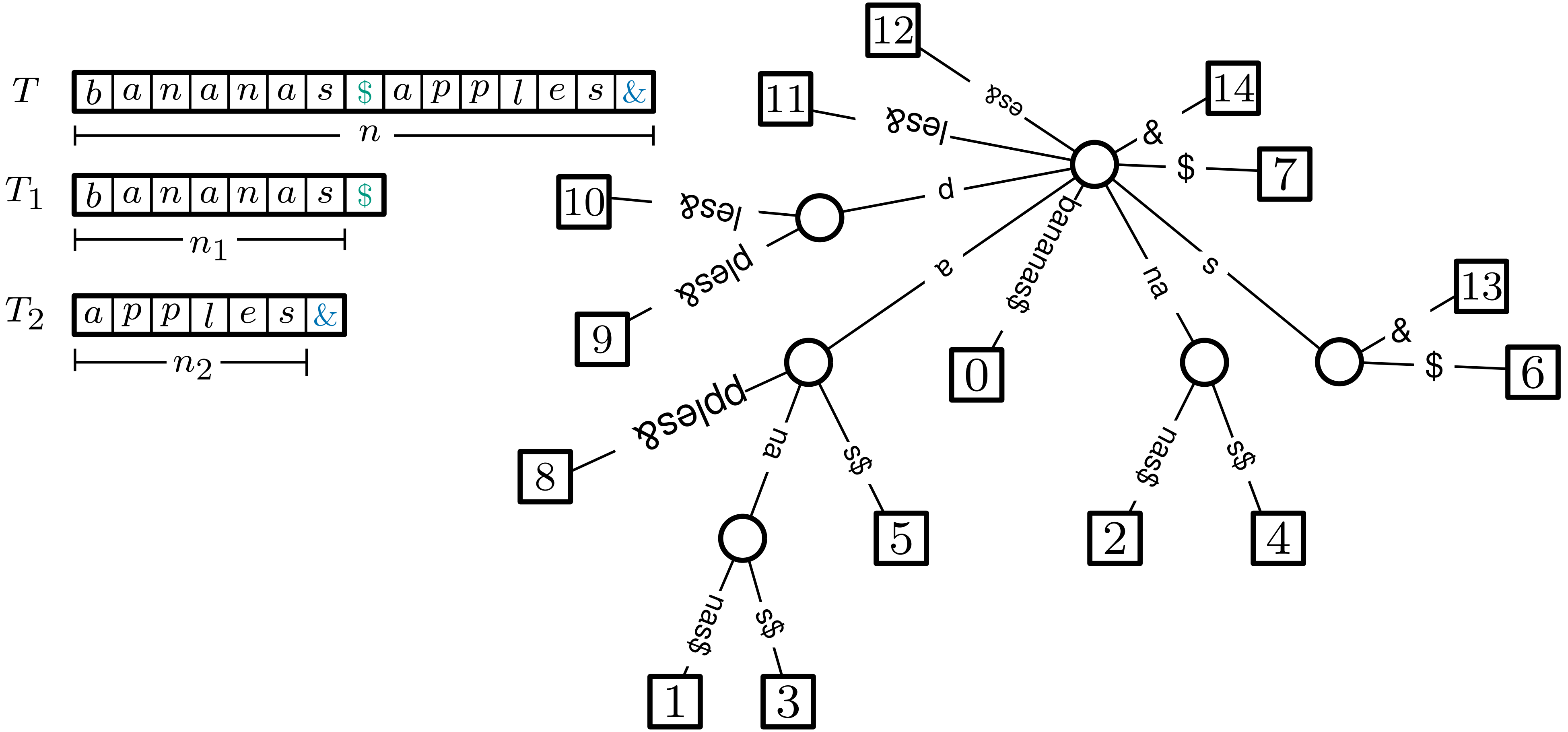
$n_1$

$T_2$ | $a$ | $p$ | $p$ | $l$ | $e$ | $s$ | $\&$

$n_2$

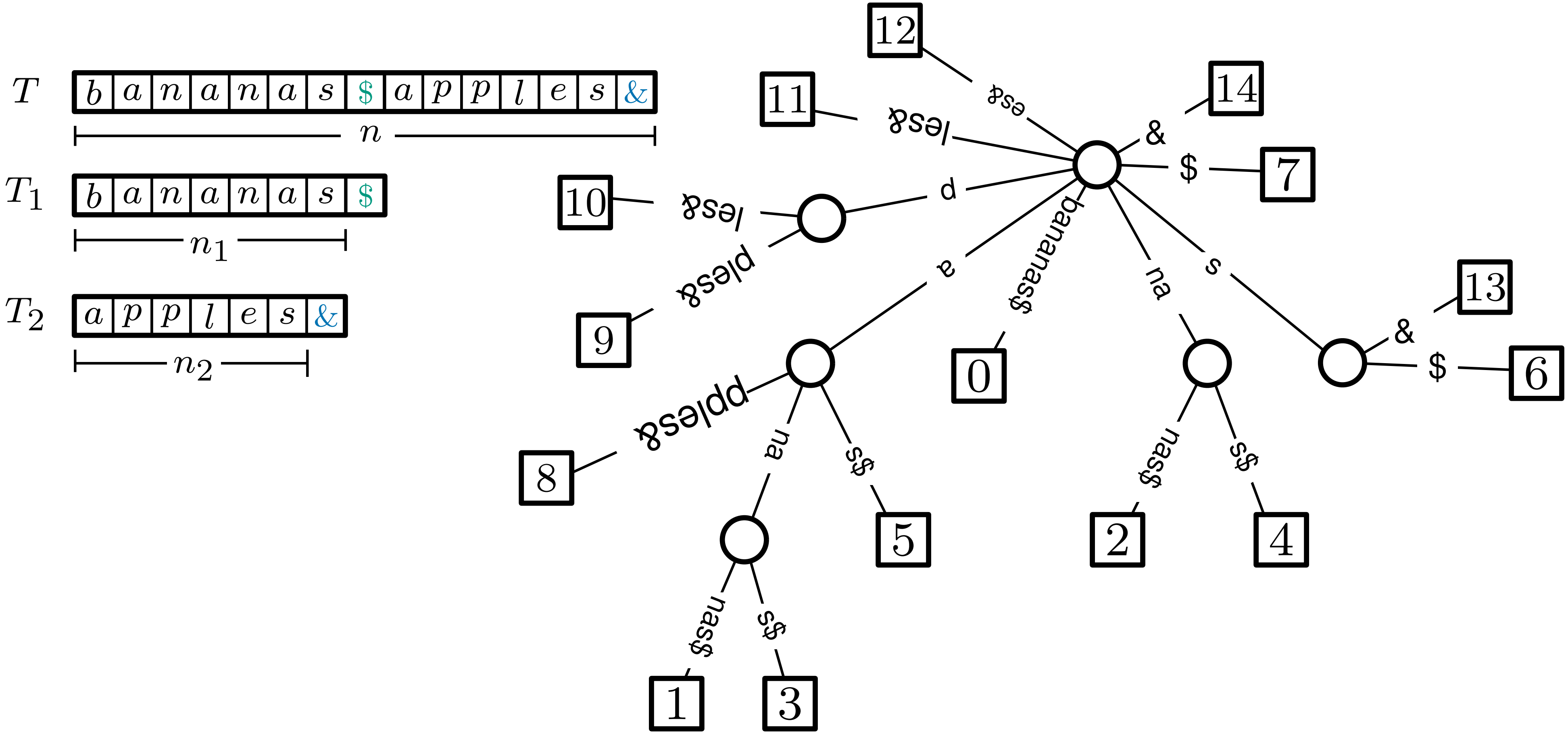
How can we index multiple texts?

# Multiple text indexing



How can we index multiple texts?

# Multiple text indexing

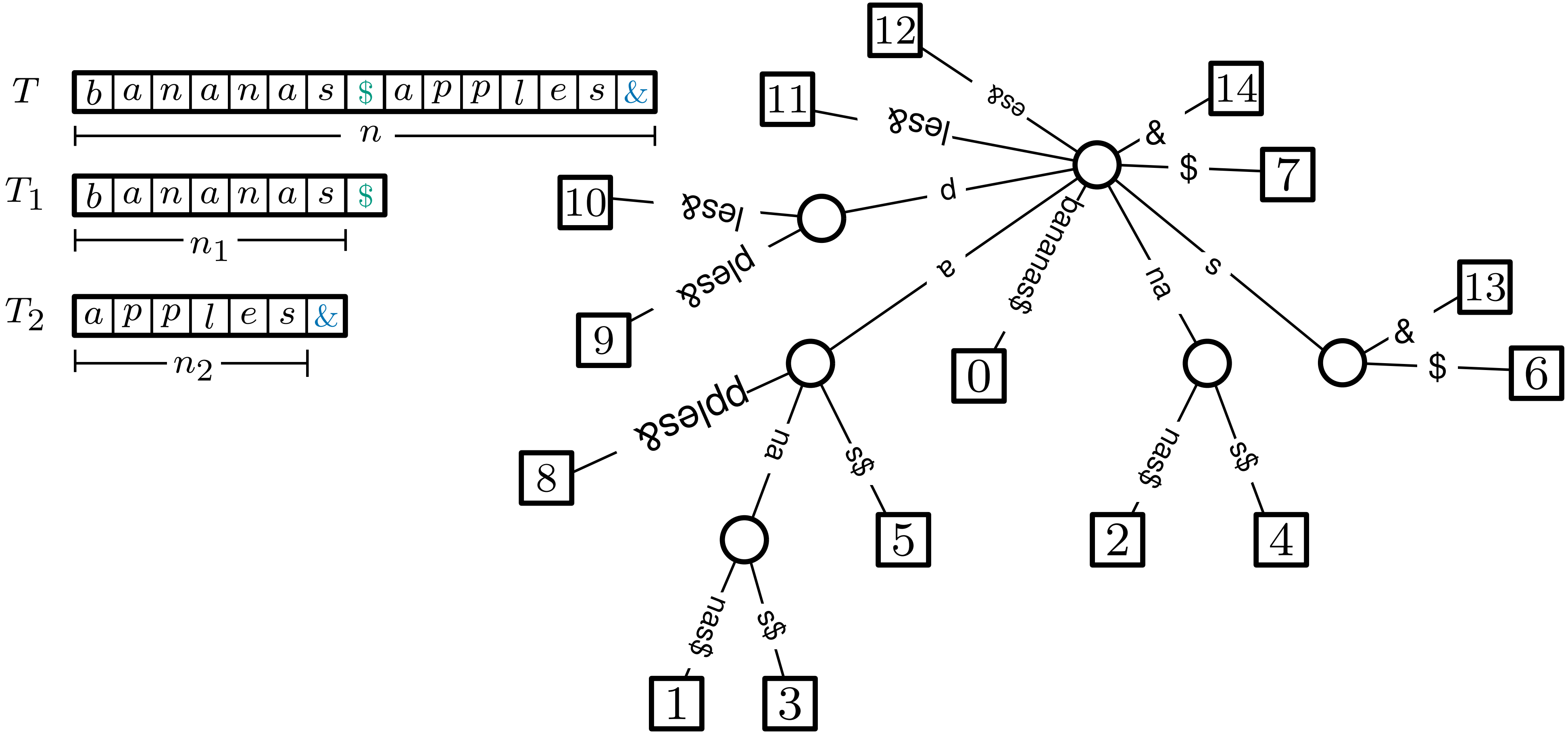


---

How can we index multiple texts?

- *Build a generalised suffix tree in* $O(n_1 + n_2)$ *space*

# Multiple text indexing

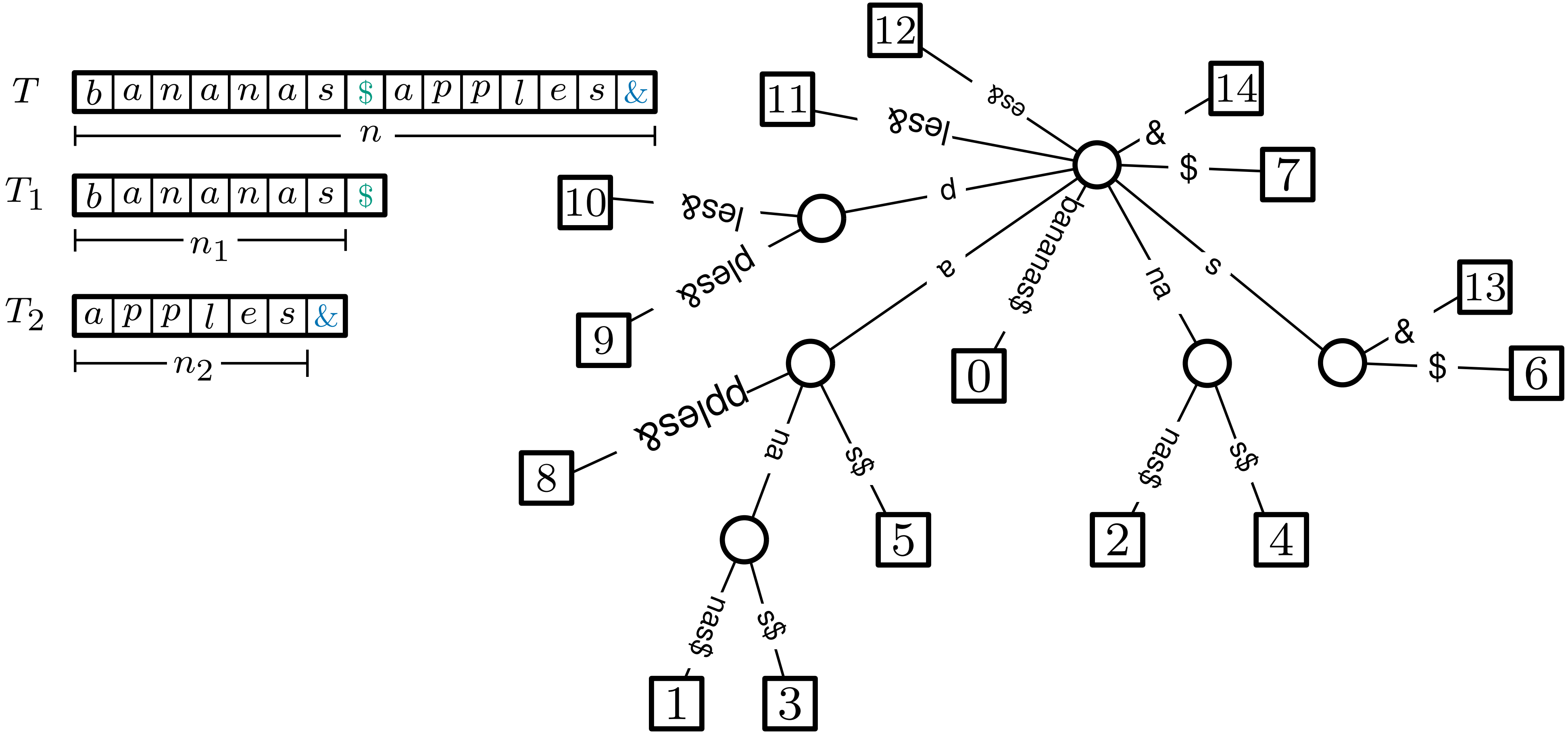


---

How can we index multiple texts?

- *Build a generalised suffix tree in* $O(n_1 + n_2)$ *space*
- *Using the linear time method (which we omitted), this takes* $O(n_1 + n_2)$ *time*

# Multiple text indexing

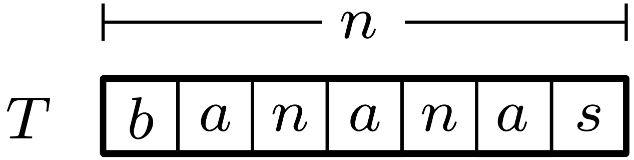


How can we index multiple texts?

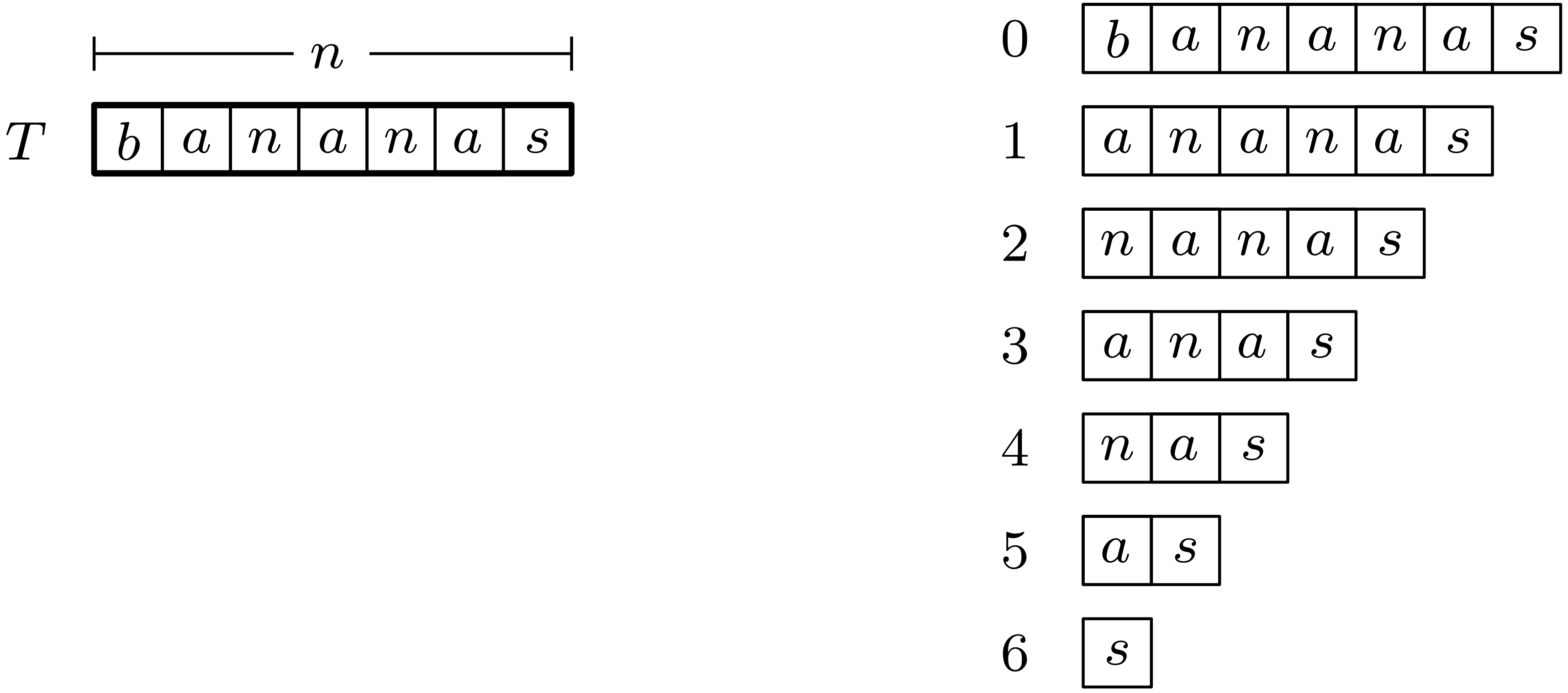
- *Build a generalised suffix tree in* $O(n_1 + n_2)$ *space*
- *Using the linear time method (which we omitted), this takes* $O(n_1 + n_2)$ *time*
- *Finding all matches of a pattern* $P$ *of length* $m$ *still takes* $O(m + occ)$ *time*

*where occ is the number of matches*

# The suffix array - a sneak preview

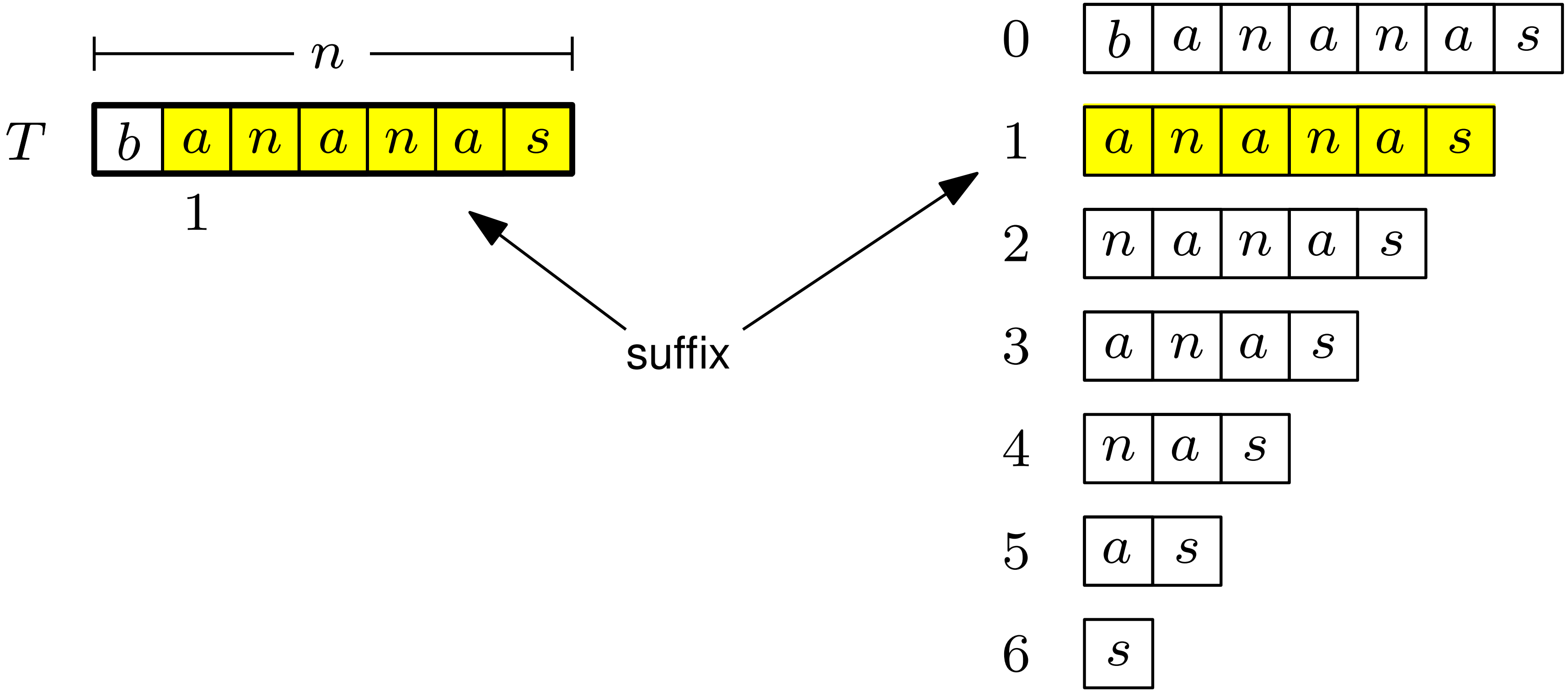


# The suffix array - a sneak preview

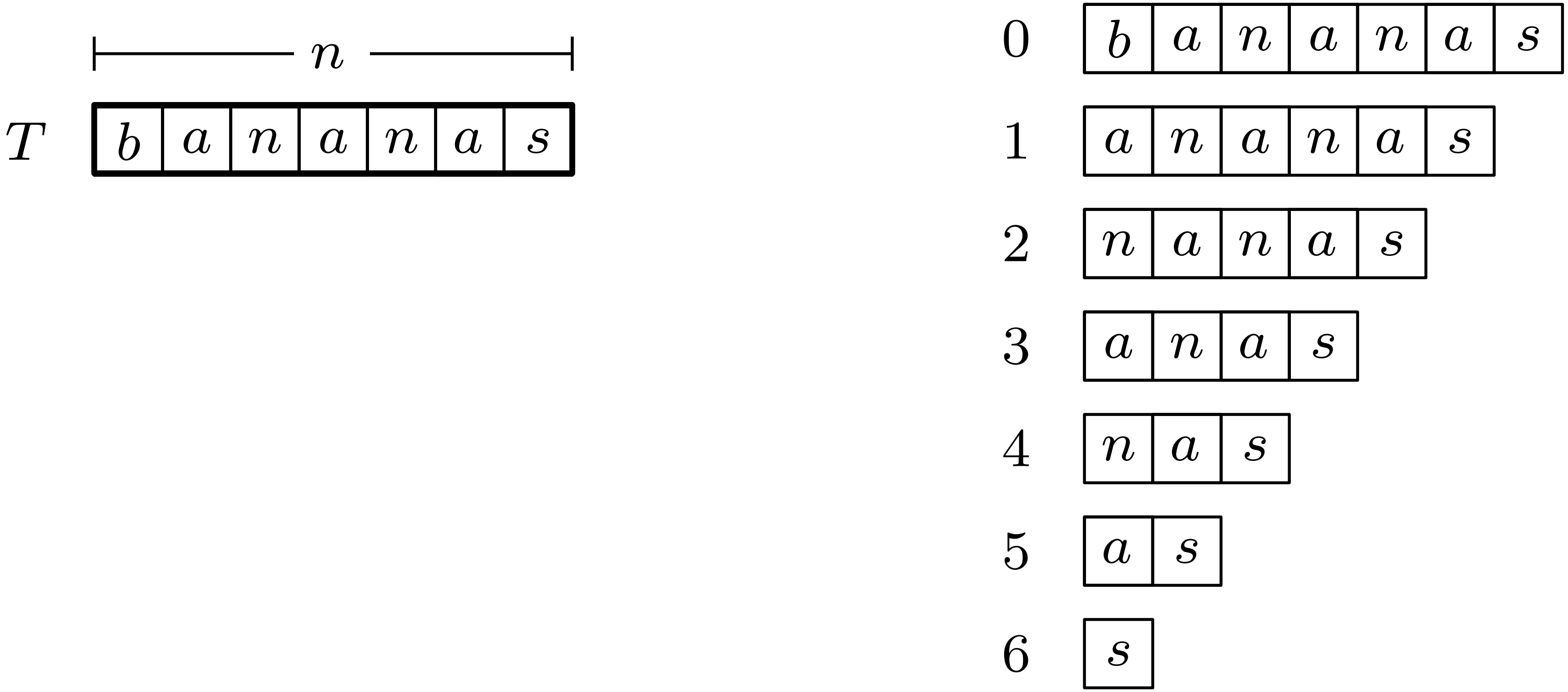


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
| 1 | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ | |
| 2 | $n$ | $a$ | $n$ | $a$ | $s$ | | |
| 3 | $a$ | $n$ | $a$ | $s$ | | | |
| 4 | $n$ | $a$ | $s$ | | | | |
| 5 | $a$ | $s$ | | | | | |
| 6 | $s$ | | | | | | |

# The suffix array - a sneak preview



# The suffix array - a sneak preview

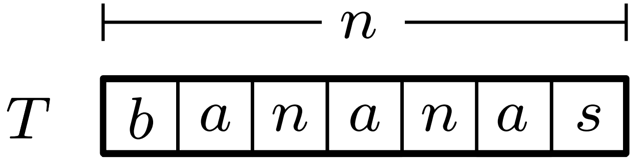


$n$

$T$: $b$ $a$ $n$ $a$ $n$ $a$ $s$

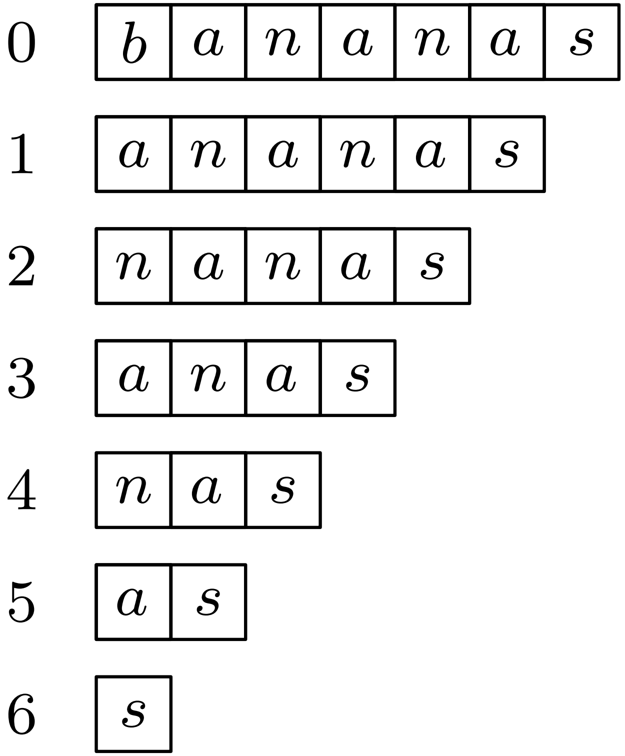
0 $b$ $a$ $n$ $a$ $n$ $a$ $s$
1 $a$ $n$ $a$ $n$ $a$ $s$
2 $n$ $a$ $n$ $a$ $s$
3 $a$ $n$ $a$ $s$
4 $n$ $a$ $s$
5 $a$ $s$
6 $s$

# The suffix array - a sneak preview

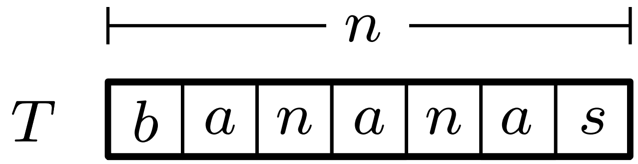


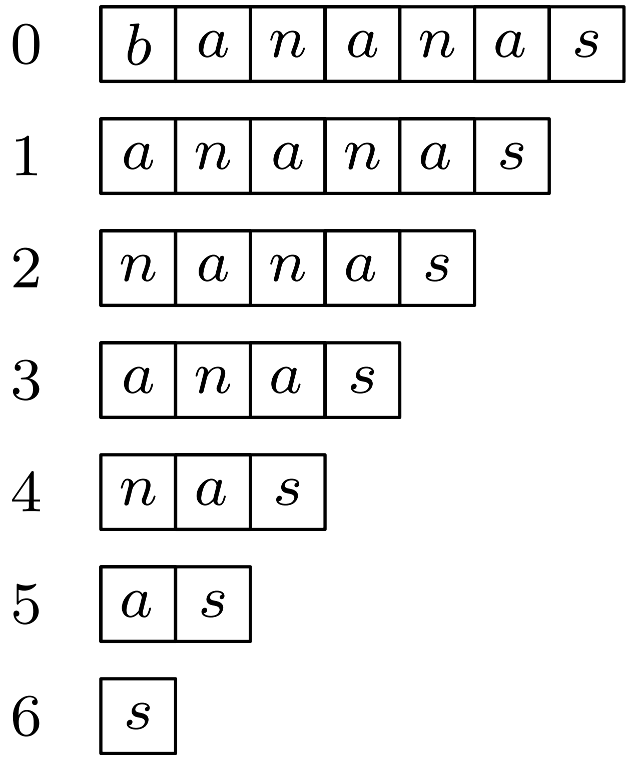
*Sort the suffixes*

*lexicographically*



# The suffix array - a sneak preview





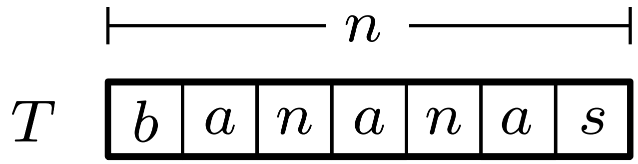
*Sort the suffixes*

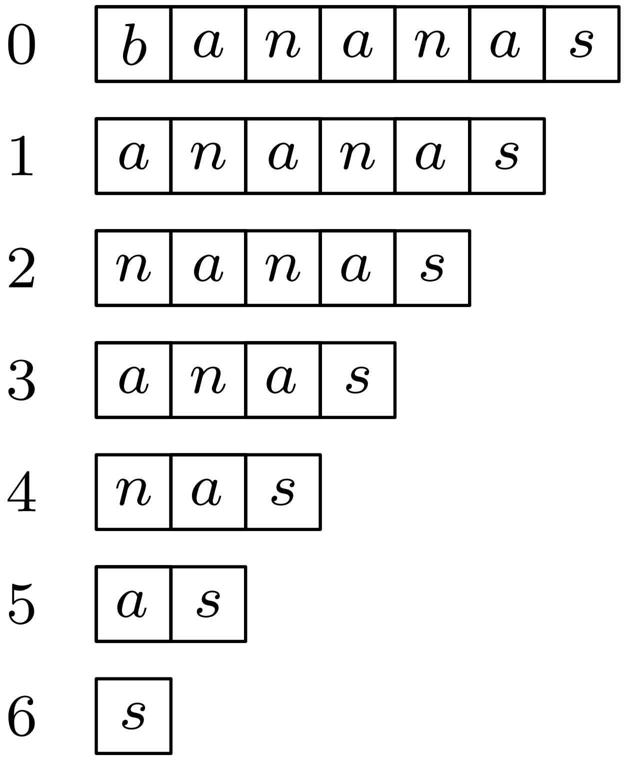
*lexicographically*

- The symbols themselves must have an order

  *throughout we will use alphabetical order*

# The suffix array - a sneak preview





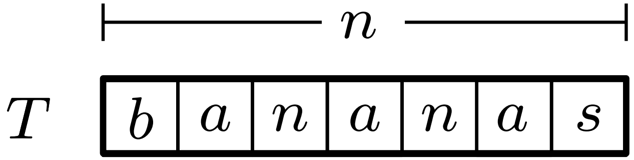
*Sort the suffixes lexicographically*

- The symbols themselves must have an order

  *throughout we will use alphabetical order*

In lexicographical ordering we sort strings based on the first symbol that differs:

# The suffix array - a sneak preview



|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
| 1 | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |   |
| 2 | $n$ | $a$ | $n$ | $a$ | $s$ |   |   |
| 3 | $a$ | $n$ | $a$ | $s$ |   |   |   |
| 4 | $n$ | $a$ | $s$ |   |   |   |   |
| 5 | $a$ | $s$ |   |   |   |   |   |
| 6 | $s$ |   |   |   |   |   |   |

*Sort the suffixes*

*lexicographically*

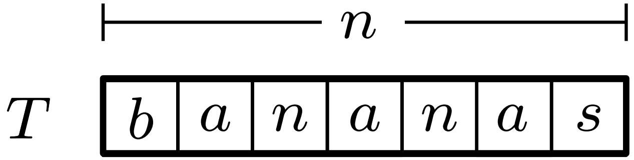
- The symbols themselves must have an order

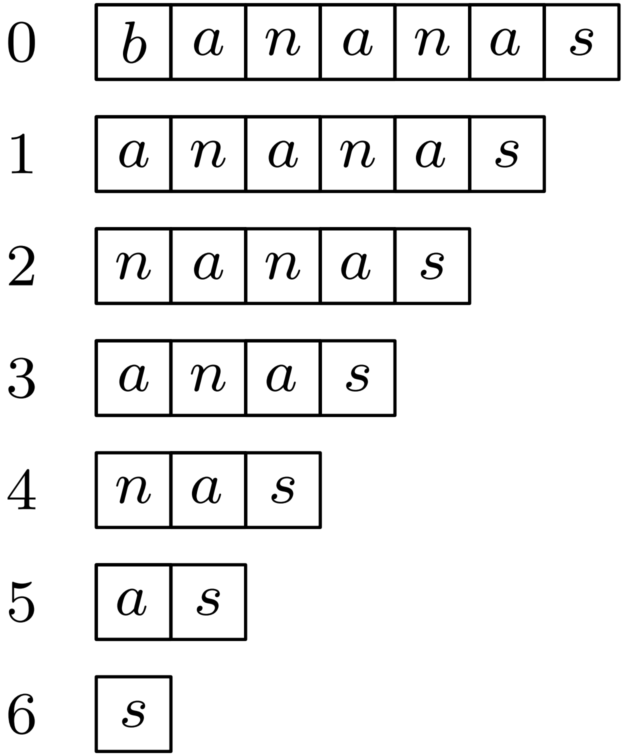
  *throughout we will use alphabetical order*

In lexicographical ordering we sort strings based on the first symbol that differs:

$$\boxed{a}\boxed{a} < \boxed{b}\boxed{a}$$

# The suffix array - a sneak preview





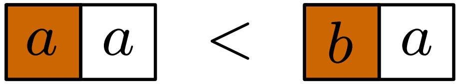
*Sort the suffixes*

*lexicographically*

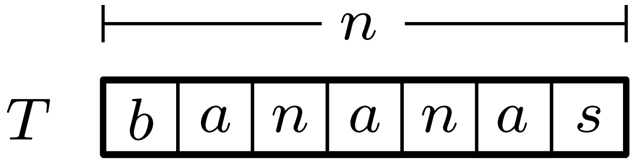
- The symbols themselves must have an order

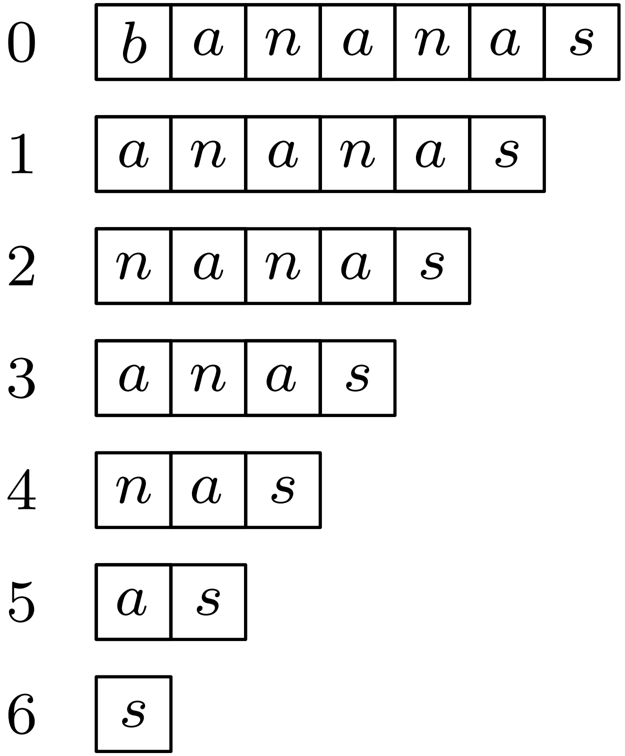
  *throughout we will use alphabetical order*

In lexicographical ordering we sort strings based on the first symbol that differs:



# The suffix array - a sneak preview





*Sort the suffixes*

*lexicographically*

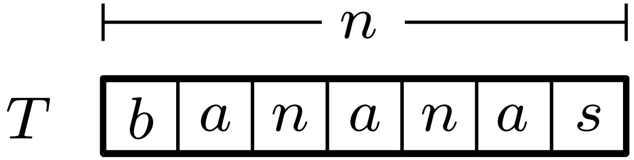
- The symbols themselves must have an order

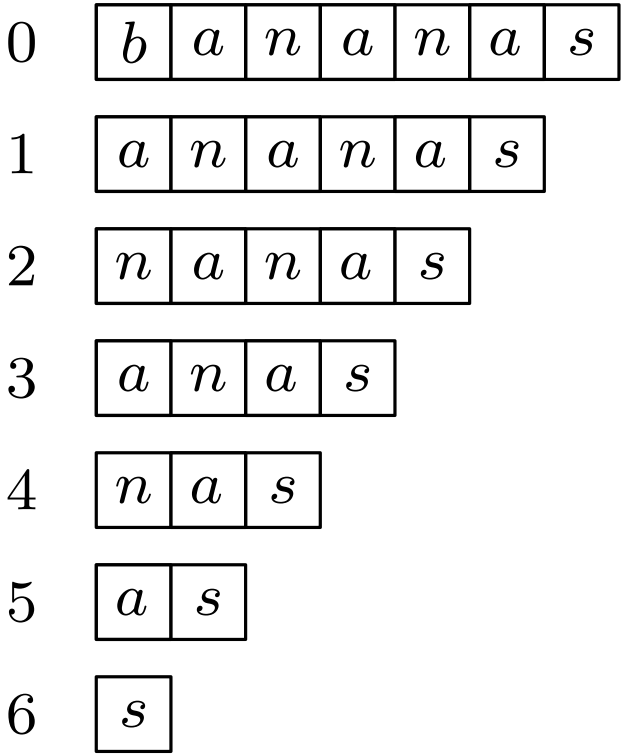
  *throughout we will use alphabetical order*

In lexicographical ordering we sort strings based on the first symbol that differs:

$$\boxed{a}\boxed{a} < \boxed{b}\boxed{a}$$

# The suffix array - a sneak preview





*Sort the suffixes lexicographically*

- The symbols themselves must have an order

  *throughout we will use alphabetical order*

In lexicographical ordering we sort strings based on the first symbol that differs:

$$\boxed{a \mid a} < \boxed{b \mid a} < \boxed{b \mid c}$$

# The suffix array - a sneak preview

$n$

$T$ $\boxed{b}\boxed{a}\boxed{n}\boxed{a}\boxed{n}\boxed{a}\boxed{s}$

0 $\boxed{b}\boxed{a}\boxed{n}\boxed{a}\boxed{n}\boxed{a}\boxed{s}$
1 $\boxed{a}\boxed{n}\boxed{a}\boxed{n}\boxed{a}\boxed{s}$
2 $\boxed{n}\boxed{a}\boxed{n}\boxed{a}\boxed{s}$
3 $\boxed{a}\boxed{n}\boxed{a}\boxed{s}$
4 $\boxed{n}\boxed{a}\boxed{s}$
5 $\boxed{a}\boxed{s}$
6 $\boxed{s}$

*Sort the suffixes*

*lexicographically*

- The symbols themselves must have an order

  *throughout we will use alphabetical order*

In lexicographical ordering we sort strings based on the first symbol that differs:

$\boxed{a}\boxed{a} < \boxed{b}\boxed{a} < \boxed{b}\boxed{c}$

# The suffix array - a sneak preview

$n$

| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
|---|---|---|---|---|---|---|---|

*Sort the suffixes lexicographically*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
| 1 | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ | |
| 2 | $n$ | $a$ | $n$ | $a$ | $s$ | | |
| 3 | $a$ | $n$ | $a$ | $s$ | | | |
| 4 | $n$ | $a$ | $s$ | | | | |
| 5 | $a$ | $s$ | | | | | |
| 6 | $s$ | | | | | | |

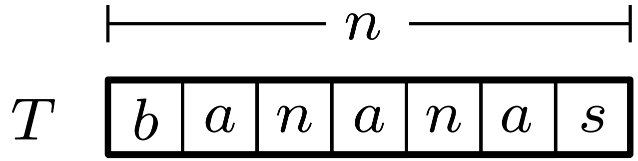
- The symbols themselves must have an order

  *throughout we will use alphabetical order*

In lexicographical ordering we sort strings based on the first symbol that differs:

$$\boxed{a\,|\,a} < \boxed{b\,|\,a} < \boxed{b\,|\,c}$$

# The suffix array - a sneak preview

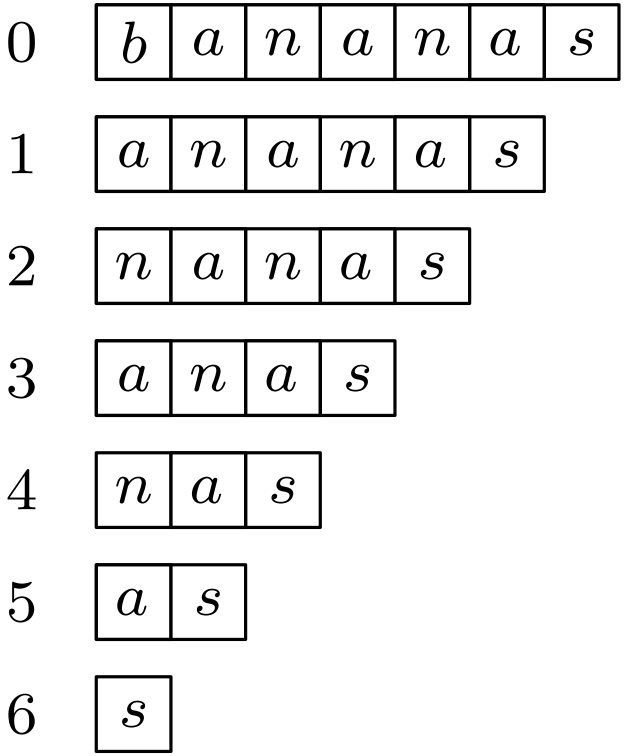


*Sort the suffixes*

*lexicographically*

- The symbols themselves must have an order

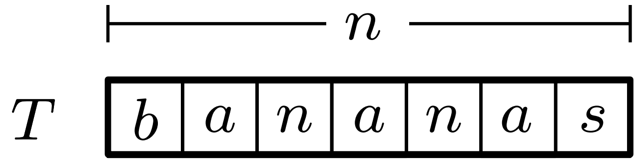
  *throughout we will use alphabetical order*



In lexicographical ordering we sort strings based on the first symbol that differs:

$$aa < ba < bc < bca$$

# The suffix array - a sneak preview



| | |
|---|---|
| 0 | $b\,a\,n\,a\,n\,a\,s$ |
| 1 | $a\,n\,a\,n\,a\,s$ |
| 2 | $n\,a\,n\,a\,s$ |
| 3 | $a\,n\,a\,s$ |
| 4 | $n\,a\,s$ |
| 5 | $a\,s$ |
| 6 | $s$ |

*Sort the suffixes*

*lexicographically*

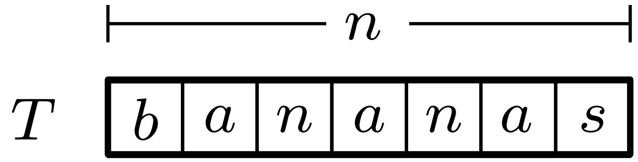
- The symbols themselves must have an order

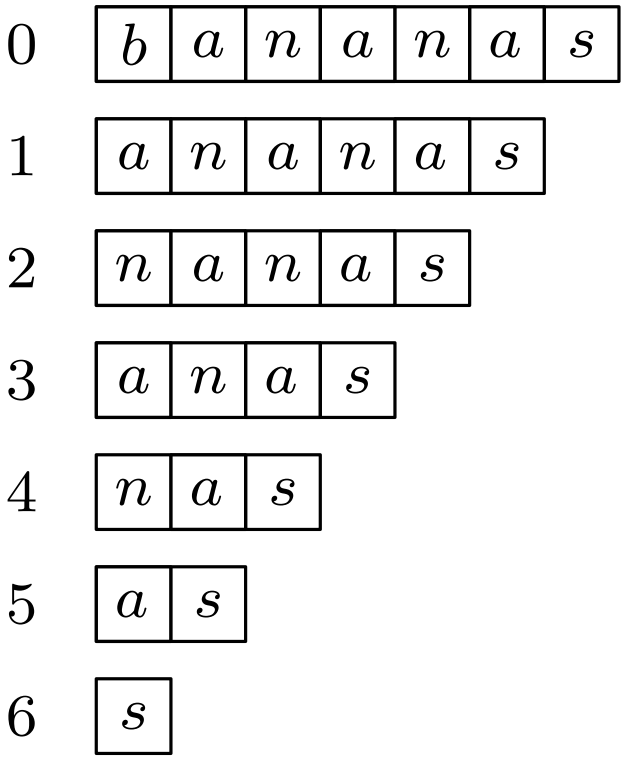
  *throughout we will use alphabetical order*

In lexicographical ordering we sort strings based on the first symbol that differs:

$$aa < ba < bc < bca$$

# The suffix array - a sneak preview





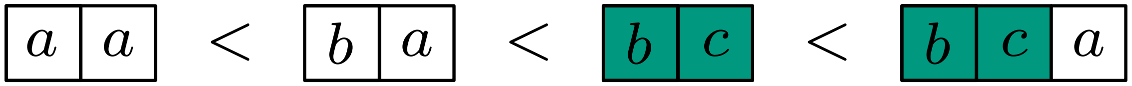
*Sort the suffixes*

*lexicographically*

- The symbols themselves must have an order

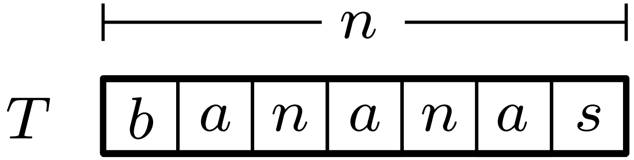
  *throughout we will use alphabetical order*

In lexicographical ordering we sort strings based on the first symbol that differs:



*(in a ‘tie’, the shorter string is smaller)*

# The suffix array - a sneak preview



*Sort the suffixes*

*lexicographically*

- The symbols themselves must have an order

  *throughout we will use alphabetical order*

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

In lexicographical ordering we sort strings based on the first symbol that differs:

$$aa < ba < bc < bca$$

*(in a ‘tie’, the shorter string is smaller)*

# The suffix array - a sneak preview

$\longleftarrow n \longrightarrow$

| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
|---|---|---|---|---|---|---|---|

*Sort the suffixes*

*lexicographically*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ | |
| 3 | $a$ | $n$ | $a$ | $s$ | | | |
| 5 | $a$ | $s$ | | | | | |
| 0 | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
| 2 | $n$ | $a$ | $n$ | $a$ | $s$ | | |
| 4 | $n$ | $a$ | $s$ | | | | |
| 6 | $s$ | | | | | | |

- The symbols themselves must have an order

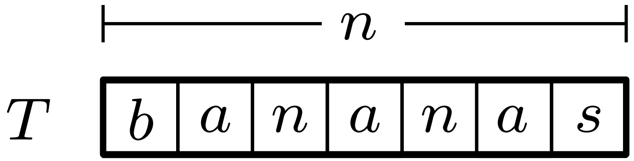
  *throughout we will use alphabetical order*

In lexicographical ordering we sort strings based on the first symbol that differs:

$$\boxed{a\,a} < \boxed{b\,a} < \boxed{b\,c} < \boxed{b\,c\,a}$$

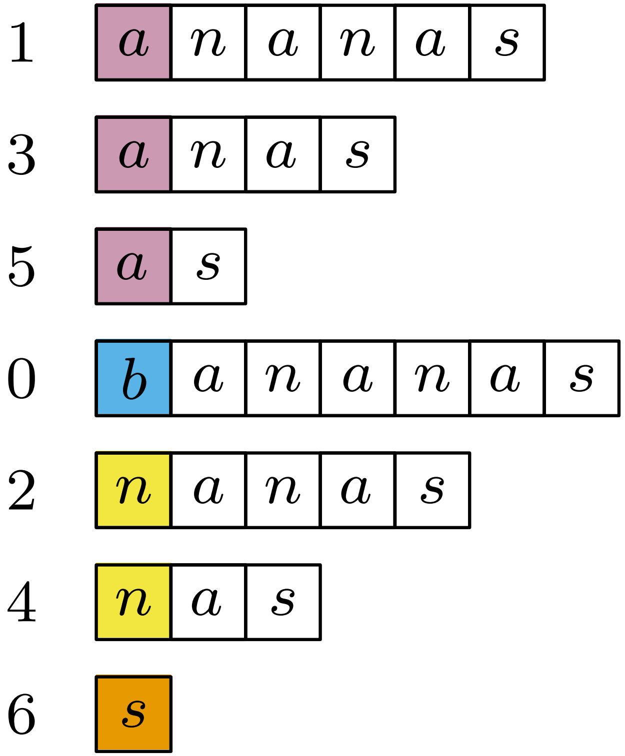
*(in a ‘tie’, the shorter string is smaller)*

# The suffix array - a sneak preview



*Sort the suffixes*

*lexicographically*



- The symbols themselves must have an order

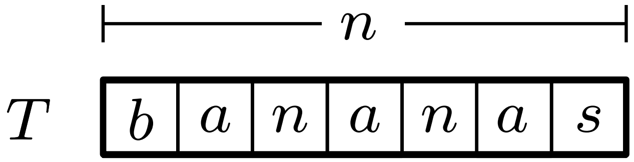
  *throughout we will use alphabetical order*

In lexicographical ordering we sort strings based on the first symbol that differs:

$$aa < ba < bc < bca$$

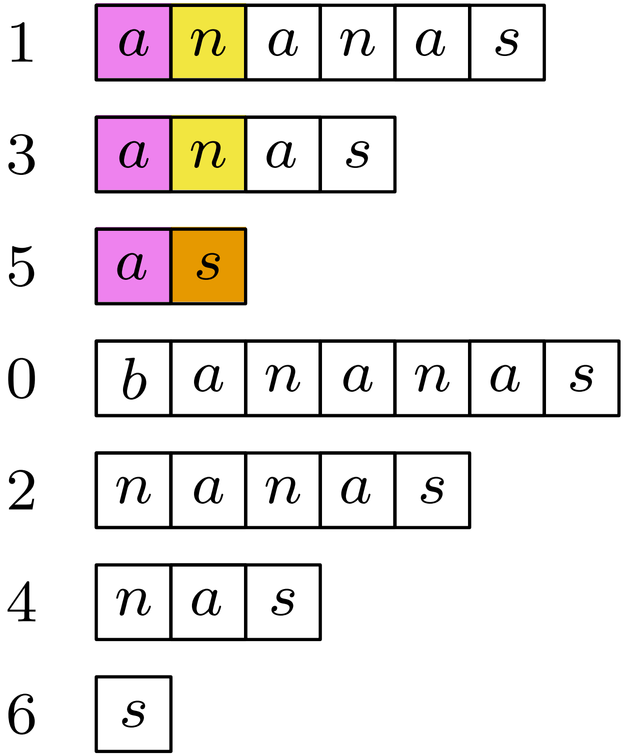
*(in a 'tie', the shorter string is smaller)*

# The suffix array - a sneak preview



*Sort the suffixes*

*lexicographically*



- The symbols themselves must have an order

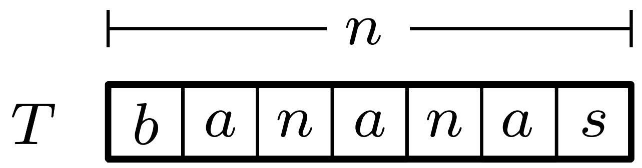
  *throughout we will use alphabetical order*

In lexicographical ordering we sort strings based on the first symbol that differs:

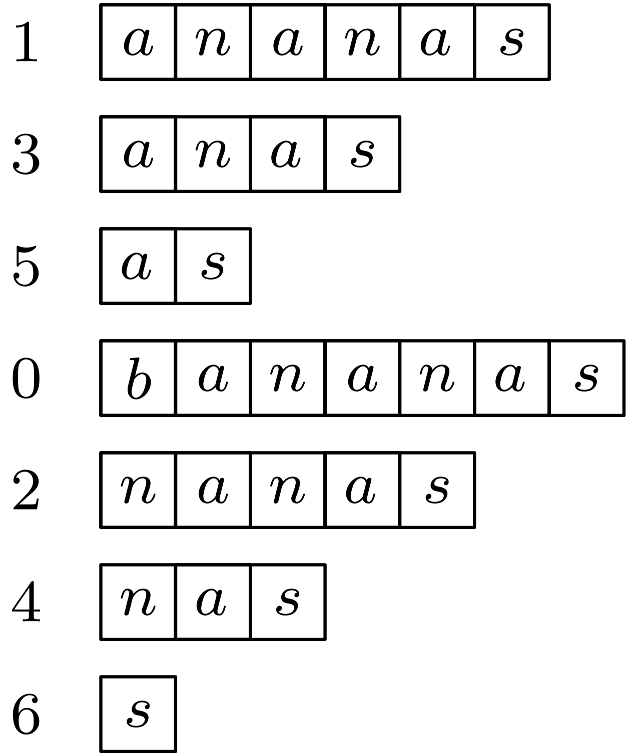
$$\boxed{a\,|\,a} < \boxed{b\,|\,a} < \boxed{b\,|\,c} < \boxed{b\,|\,c\,|\,a}$$

*(in a 'tie', the shorter string is smaller)*

# The suffix array - a sneak preview



*Sort the suffixes lexicographically*


1 a n a n a s
3 a n a s
5 a s
0 b a n a n a s
2 n a n a s
4 n a s
6 s

- The symbols themselves must have an order

  *throughout we will use alphabetical order*

In lexicographical ordering we sort strings based on the first symbol that differs:

$$\boxed{a\,|\,a} < \boxed{b\,|\,a} < \boxed{b\,|\,c} < \boxed{b\,|\,c\,|\,a}$$

*(in a ‘tie’, the shorter string is smaller)*

# The suffix array - a sneak preview

$n$

$T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |

*Sort the suffixes lexicographically*

- The symbols themselves must have an order

  *throughout we will use alphabetical order*

1 | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |

3 | $a$ | $n$ | $a$ | $s$ |

5 | $a$ | $s$ |

0 | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |

2 | $n$ | $a$ | $n$ | $a$ | $s$ |

4 | $n$ | $a$ | $s$ |

6 | $s$ |

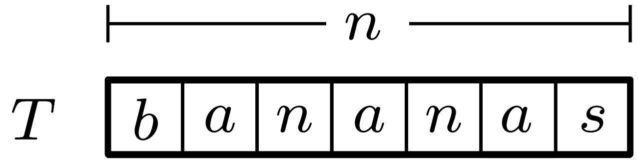
just a fancy name for the order the strings would appear in a dictionary

In lexicographical ordering we sort strings based on the first symbol that differs:

$aa < ba < bc < bca$

*(in a 'tie', the shorter string is smaller)*

# The suffix array - a sneak preview



| | |
|---|---|
| 1 | a n a n a s |
| 3 | a n a s |
| 5 | a s |
| 0 | b a n a n a s |
| 2 | n a n a s |
| 4 | n a s |
| 6 | s |

*Sort the suffixes*

*lexicographically*

- The symbols themselves must have an order

*throughout we will use alphabetical order*

just a fancy name for the order the strings would appear in a dictionary

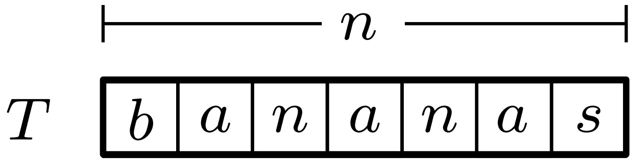
In lexicographical ordering we sort strings based on the first symbol that differs:

$$\boxed{a\,|\,a} < \boxed{b\,|\,a} < \boxed{b\,|\,c} < \boxed{b\,|\,c\,|\,a}$$

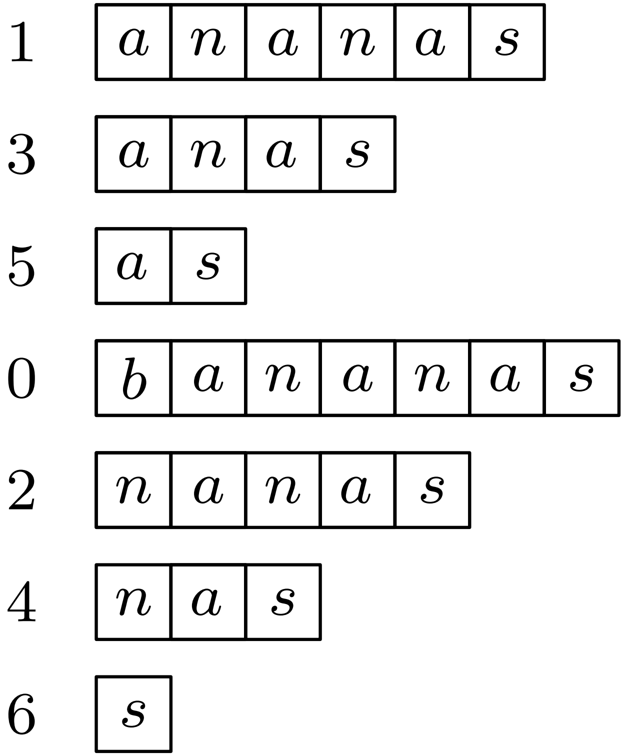
*(in a ‘tie’, the shorter string is smaller)*

If the symbols don’t have a natural order, we use their binary representation in memory

# The suffix array - a sneak preview

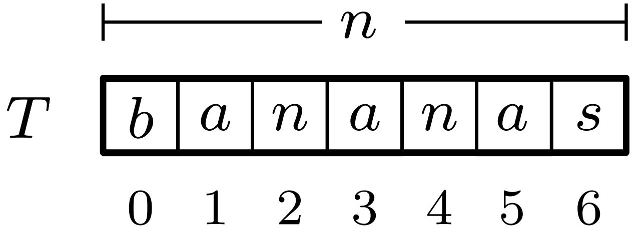


*Sort the suffixes lexicographically*



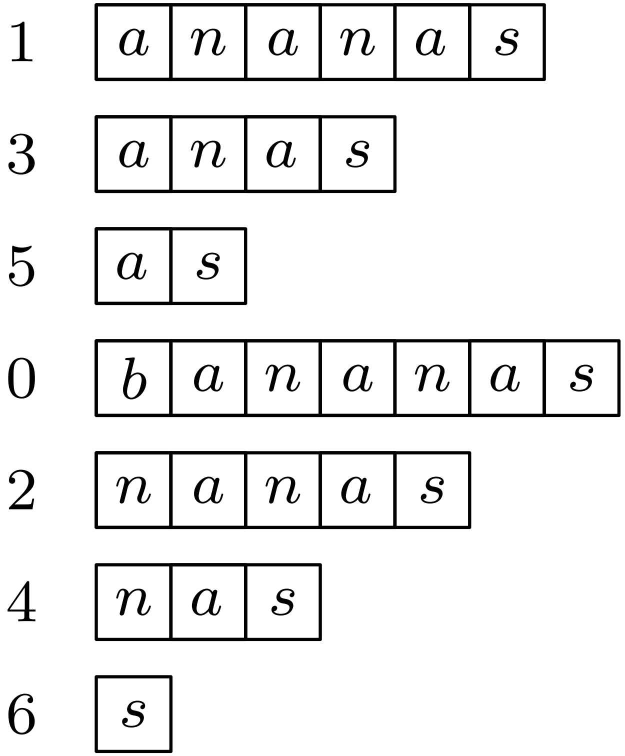
| | |
|---|---|
| 1 | $a\ n\ a\ n\ a\ s$ |
| 3 | $a\ n\ a\ s$ |
| 5 | $a\ s$ |
| 0 | $b\ a\ n\ a\ n\ a\ s$ |
| 2 | $n\ a\ n\ a\ s$ |
| 4 | $n\ a\ s$ |
| 6 | $s$ |

# The suffix array - a sneak preview

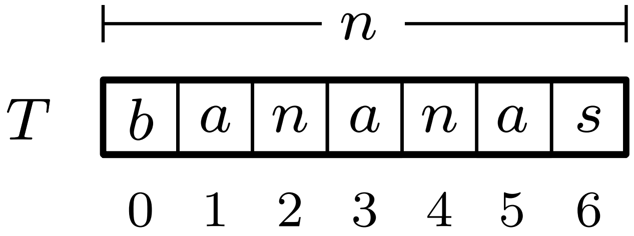


*Sort the suffixes*

*lexicographically*



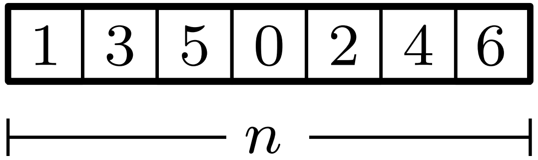
# The suffix array - a sneak preview



|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

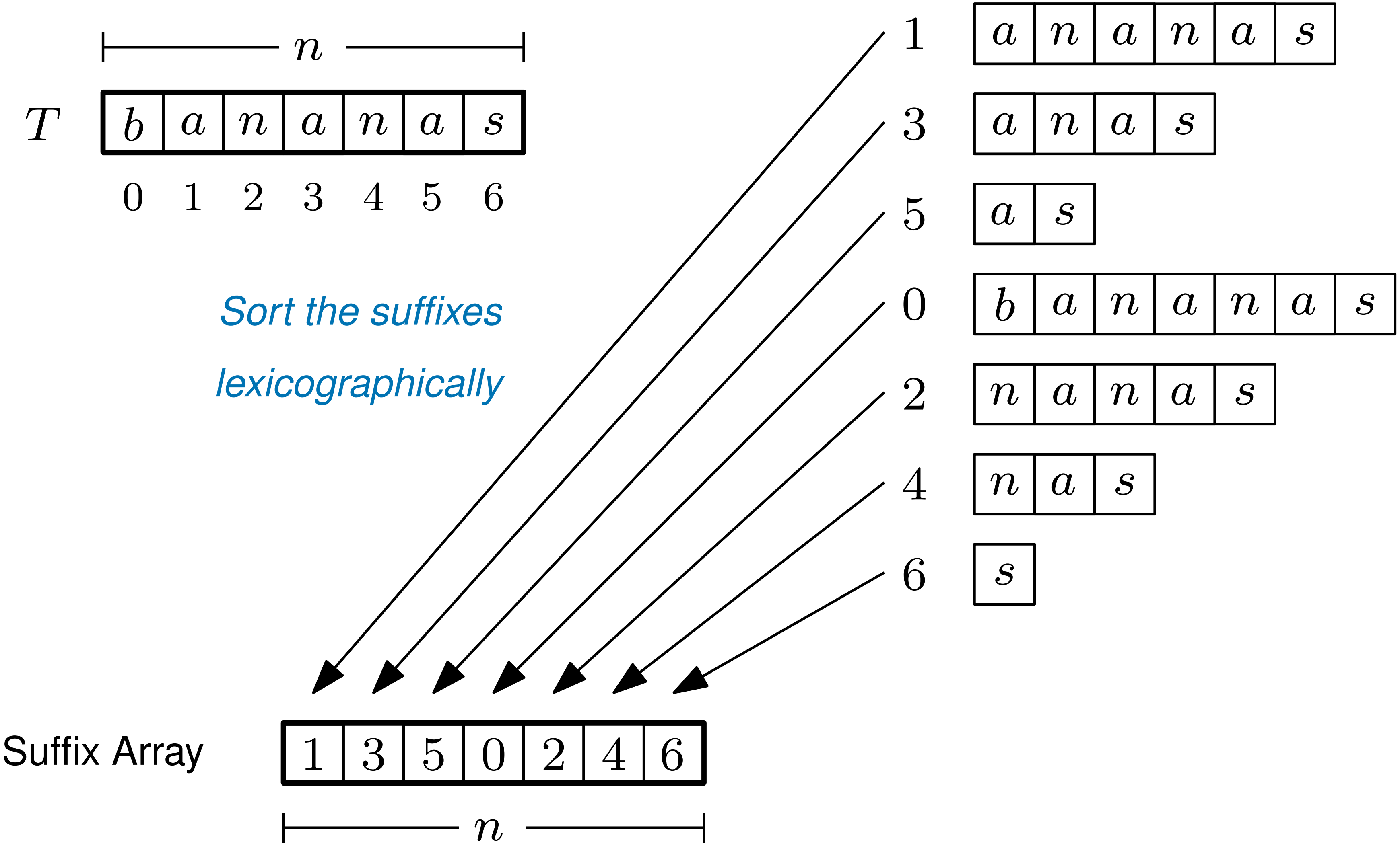
*Sort the suffixes lexicographically*

|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 1 | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |  |
| 3 | $a$ | $n$ | $a$ | $s$ |  |  |  |
| 5 | $a$ | $s$ |  |  |  |  |  |
| 0 | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
| 2 | $n$ | $a$ | $n$ | $a$ | $s$ |  |  |
| 4 | $n$ | $a$ | $s$ |  |  |  |  |
| 6 | $s$ |  |  |  |  |  |  |



|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| Suffix Array | 1 | 3 | 5 | 0 | 2 | 4 | 6 |

# The suffix array - a sneak preview



$n$

$T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |

0 1 2 3 4 5 6

*Sort the suffixes lexicographically*

1 $a\ n\ a\ n\ a\ s$

3 $a\ n\ a\ s$

5 $a\ s$

0 $b\ a\ n\ a\ n\ a\ s$

2 $n\ a\ n\ a\ s$

4 $n\ a\ s$

6 $s$

Suffix Array | 1 | 3 | 5 | 0 | 2 | 4 | 6 |

$n$

# The suffix array - a sneak preview

$n$

| $T$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

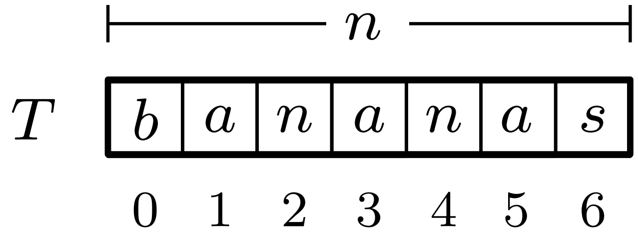
*Sort the suffixes lexicographically*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ | |
| 3 | $a$ | $n$ | $a$ | $s$ | | | |
| 5 | $a$ | $s$ | | | | | |
| 0 | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
| 2 | $n$ | $a$ | $n$ | $a$ | $s$ | | |
| 4 | $n$ | $a$ | $s$ | | | | |
| 6 | $s$ | | | | | | |

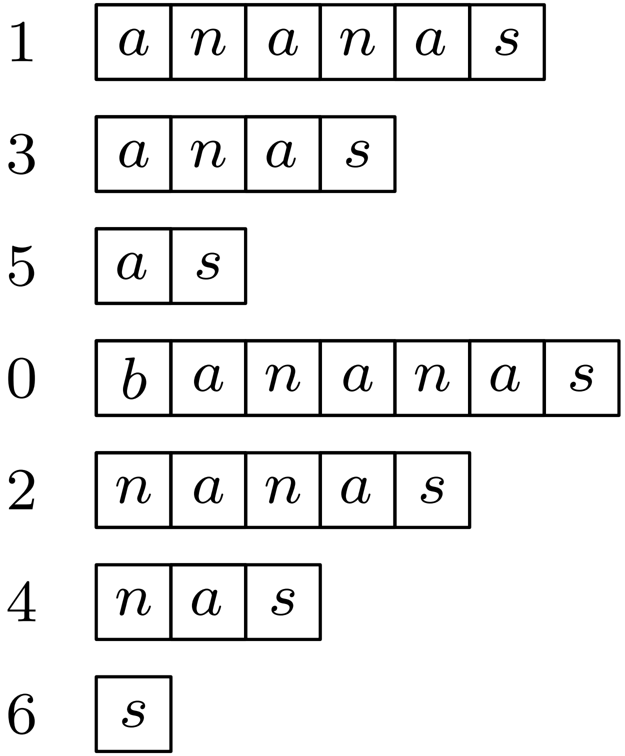
| Suffix Array | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 3 | 5 | 0 | 2 | 4 | 6 |

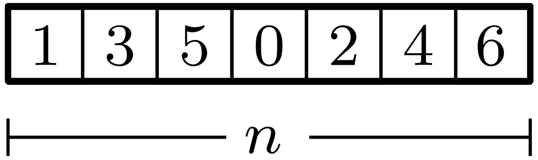
$n$

# The suffix array - a sneak preview



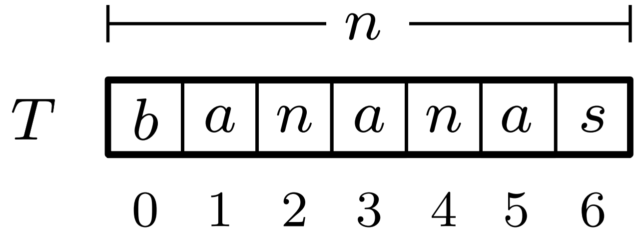
*Sort the suffixes lexicographically*



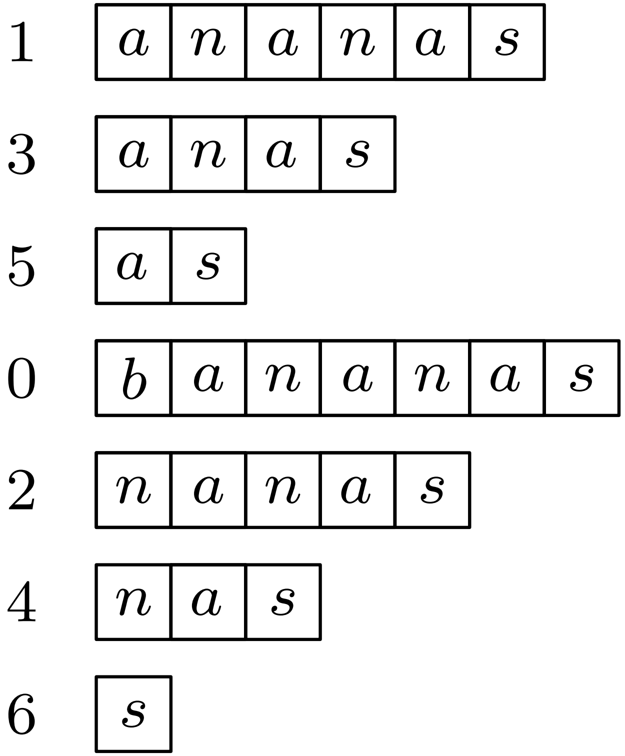


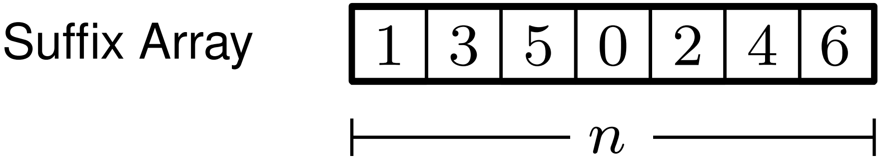
*The suffix array is much smaller than the suffix tree (in terms of constants)*

# The suffix array - a sneak preview



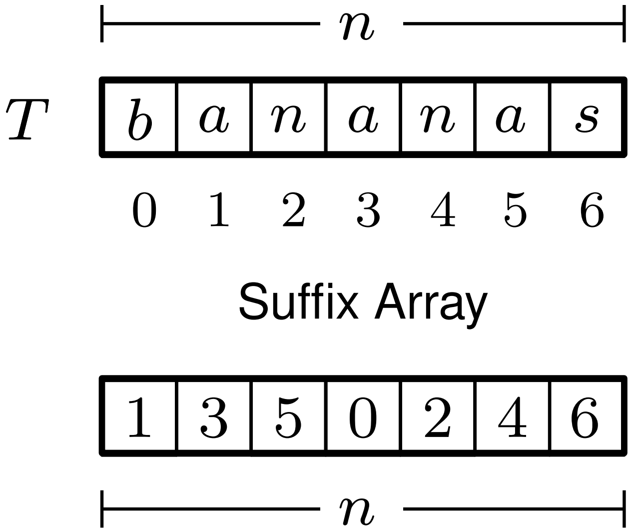
*Sort the suffixes lexicographically*



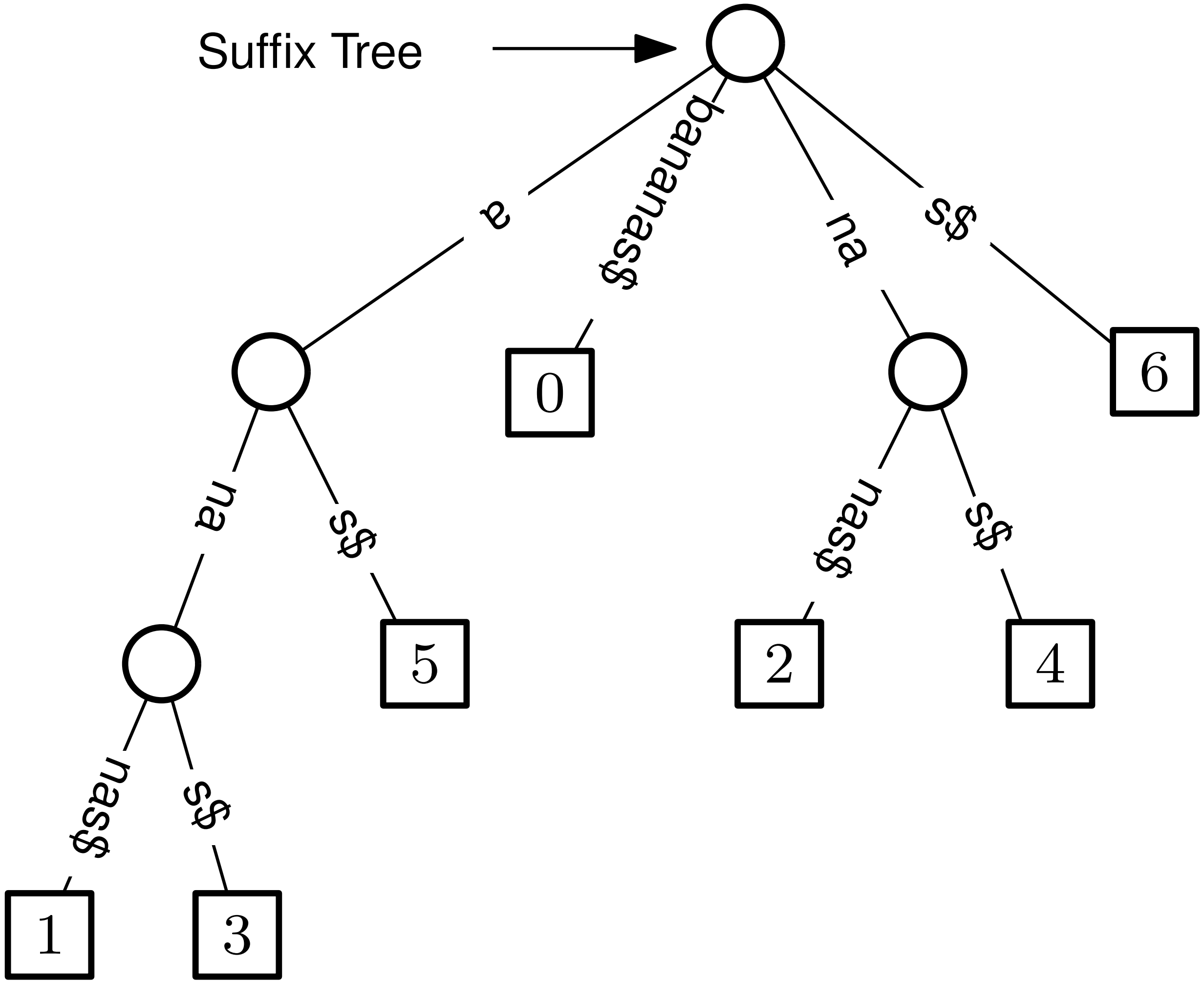


*The suffix array is much smaller than the suffix tree (in terms of constants)*

# Constructing the Suffix Array from the Suffix Tree



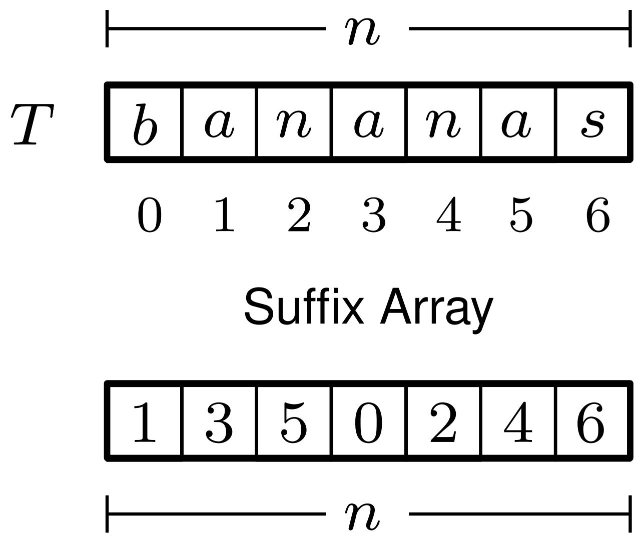
Suffix Tree

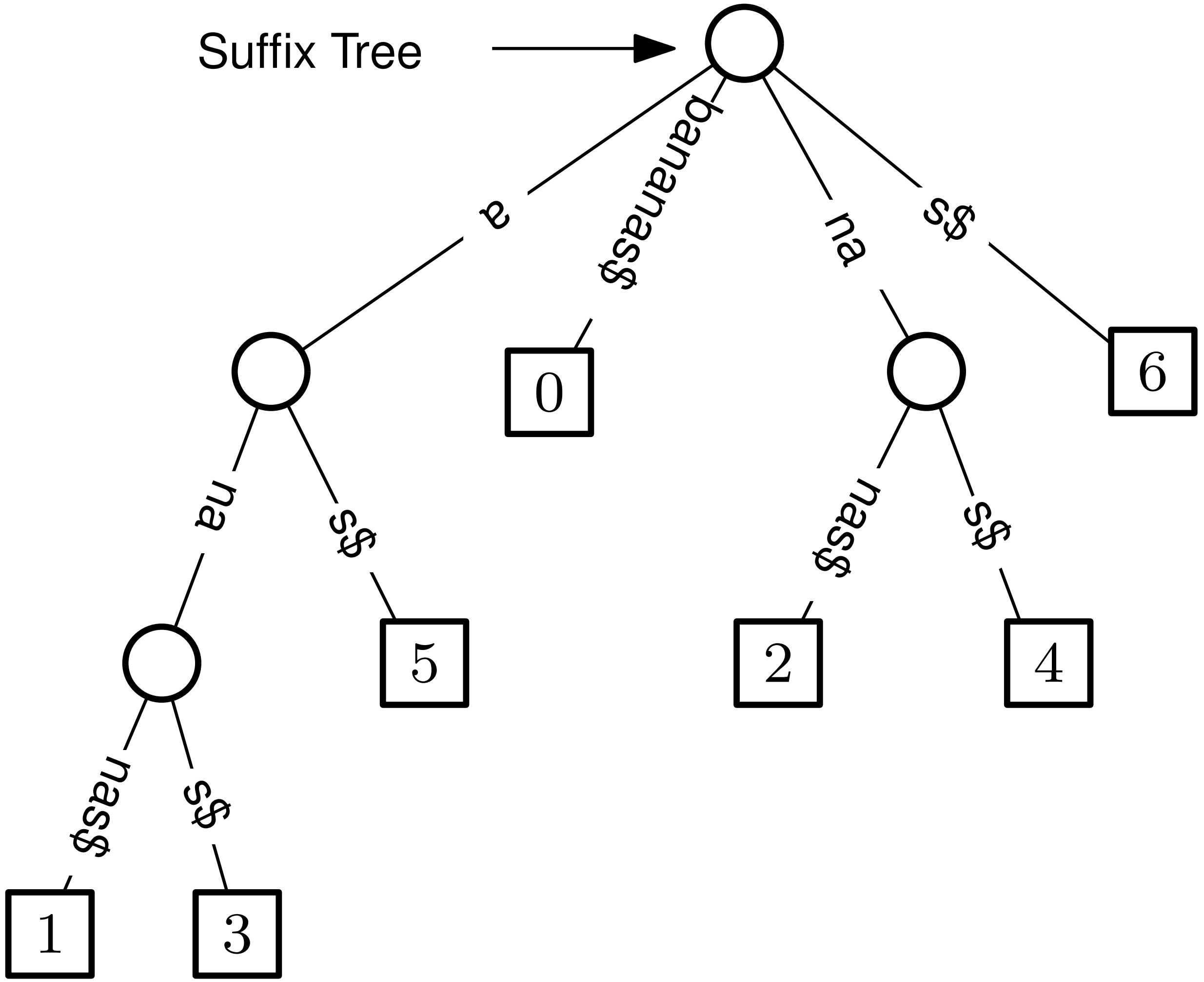


*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

# Constructing the Suffix Array from the Suffix Tree



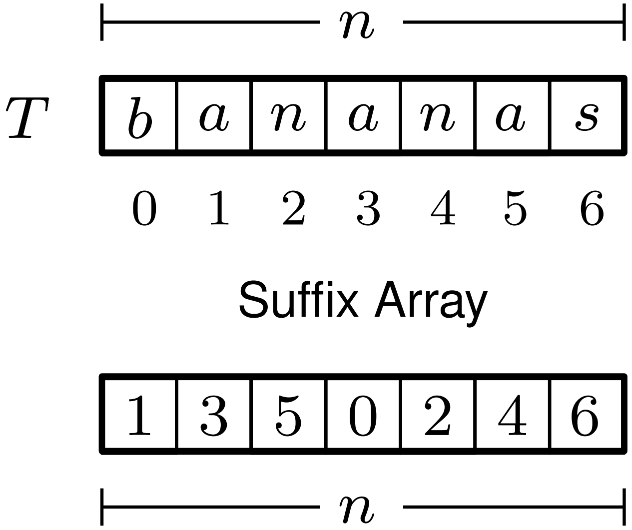


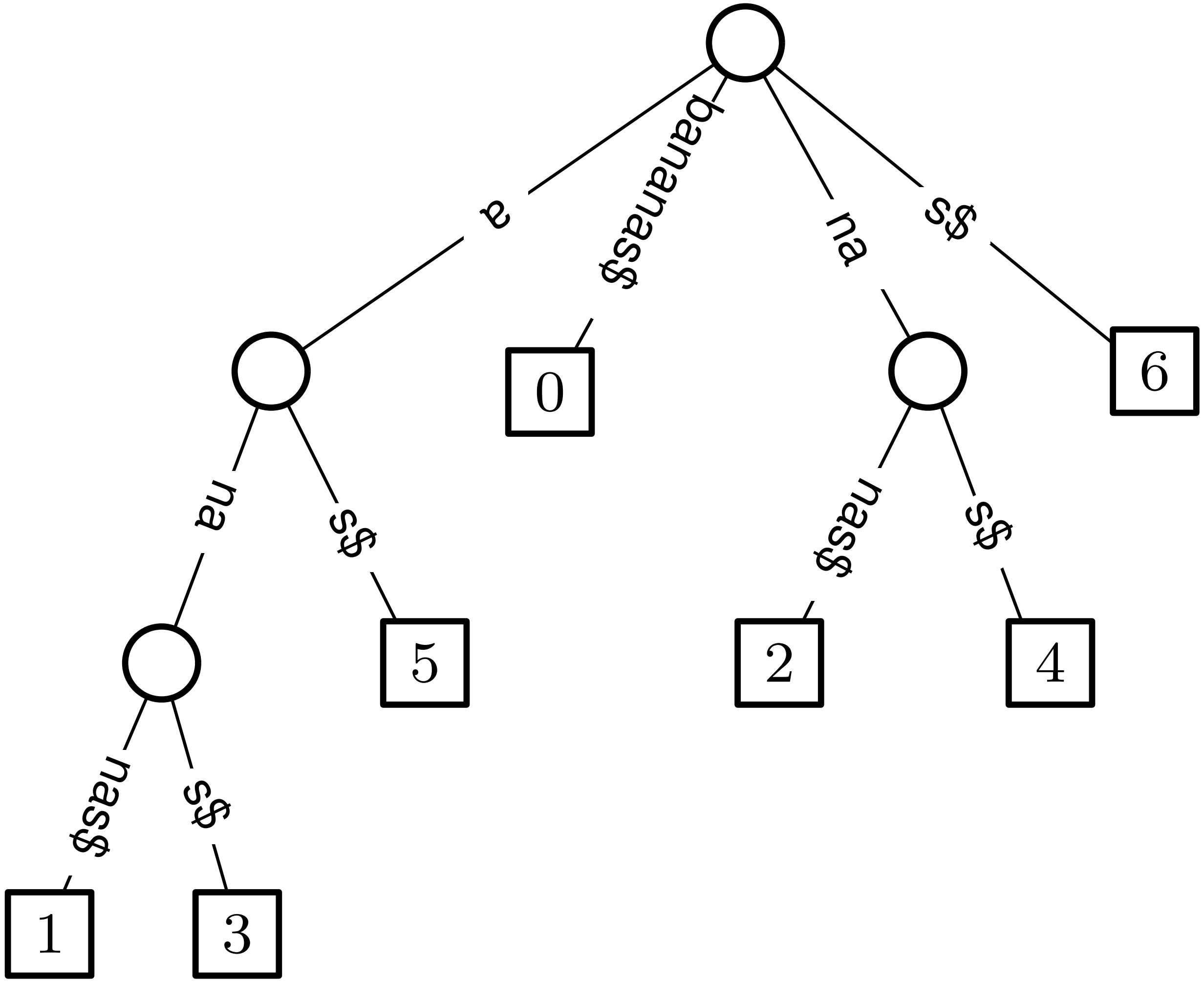
*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree



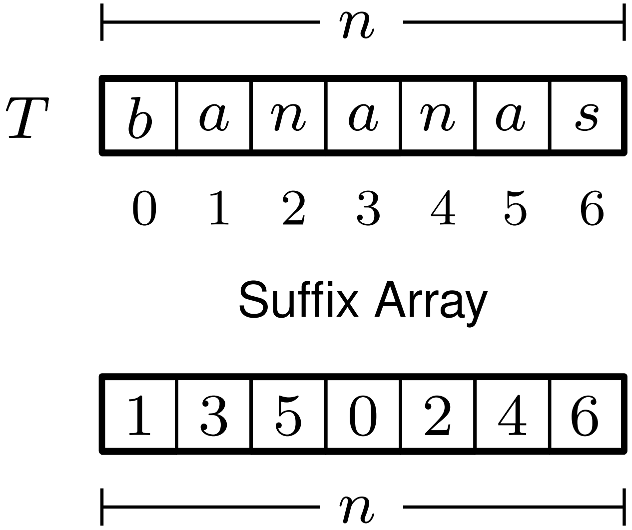


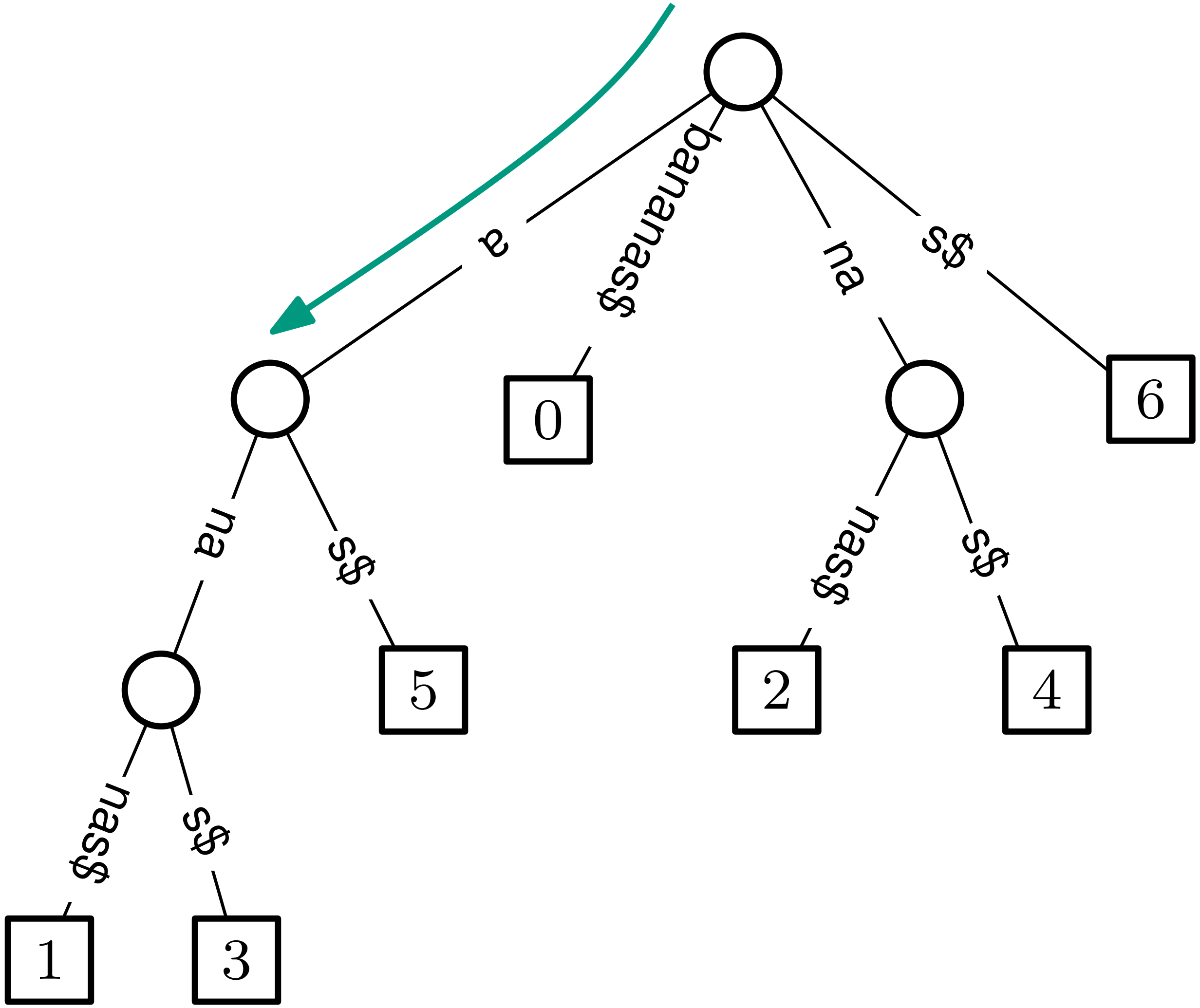
*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree





*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree

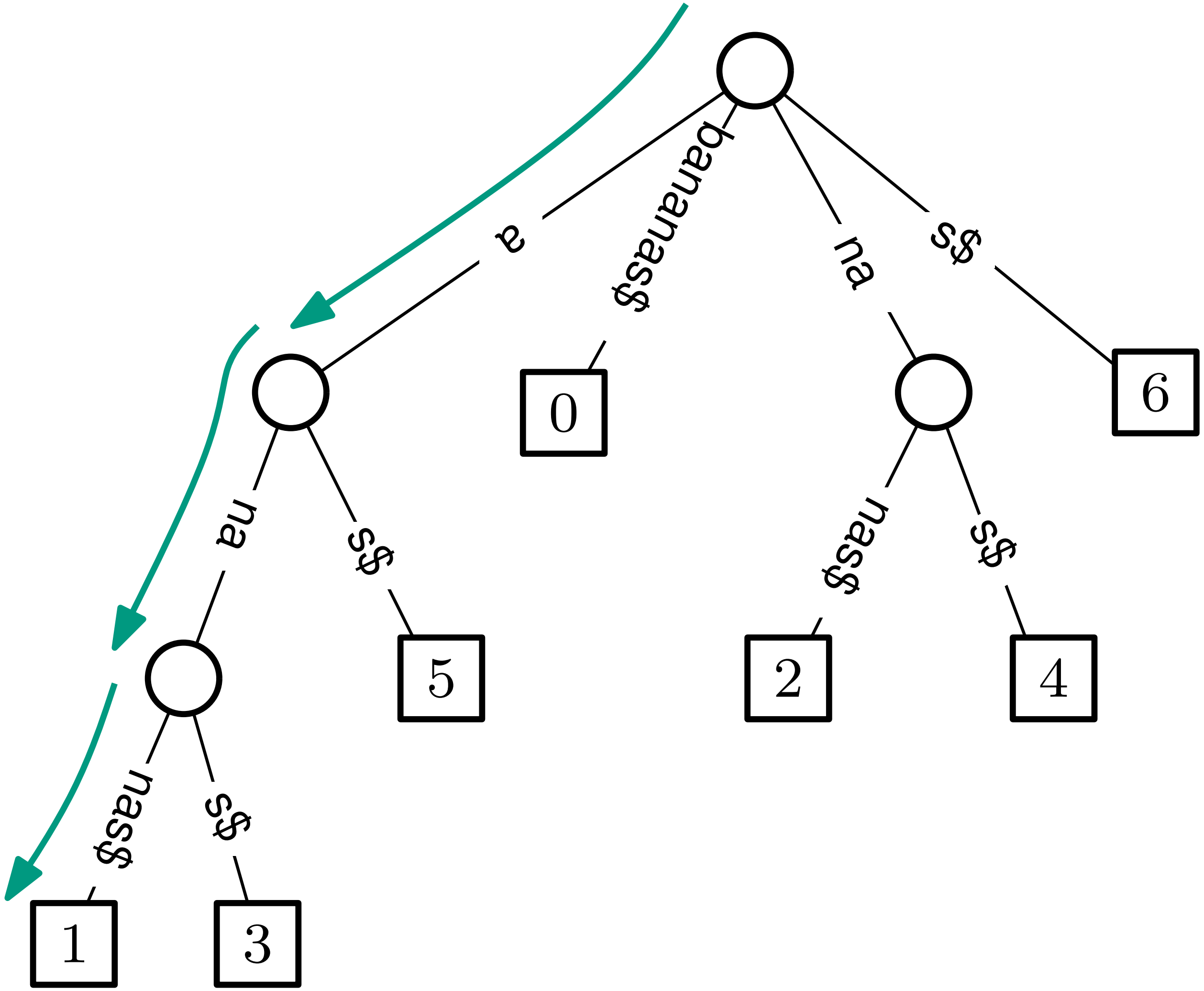
$n$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Suffix Array

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 0 | 2 | 4 | 6 |

$n$

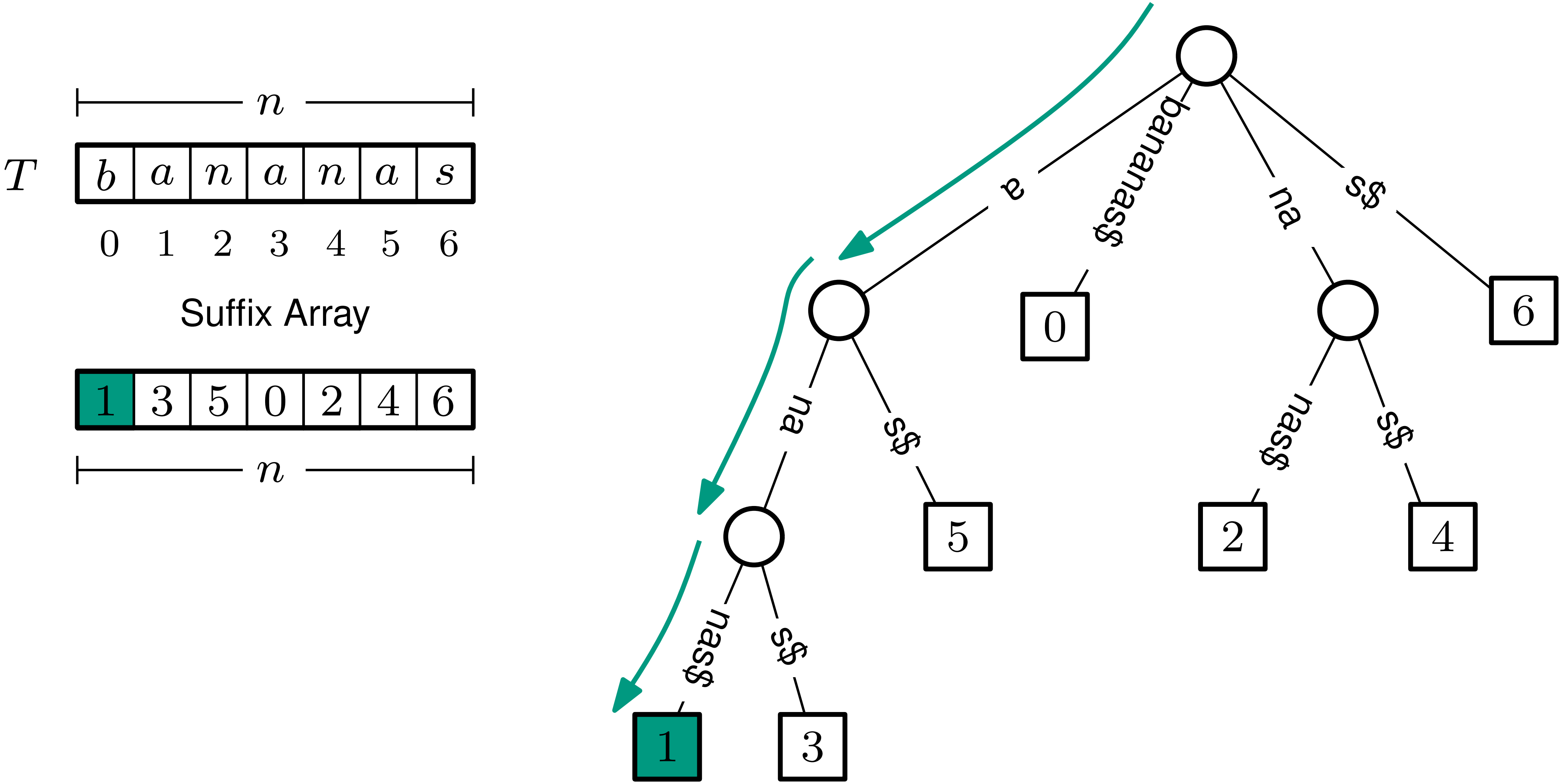


*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree



| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Suffix Array

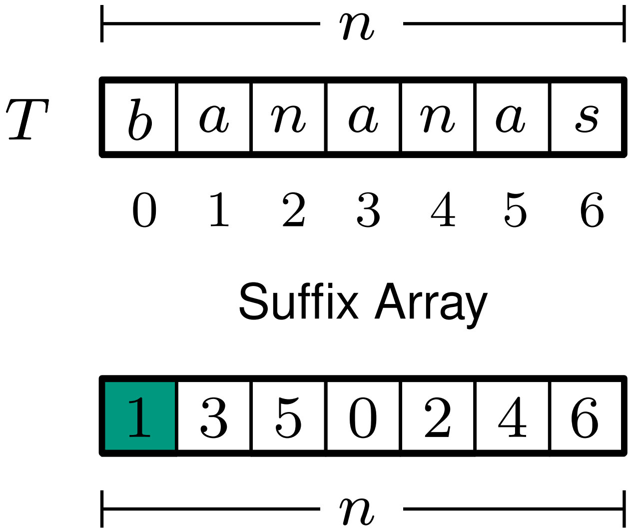
| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 0 | 2 | 4 | 6 |

*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

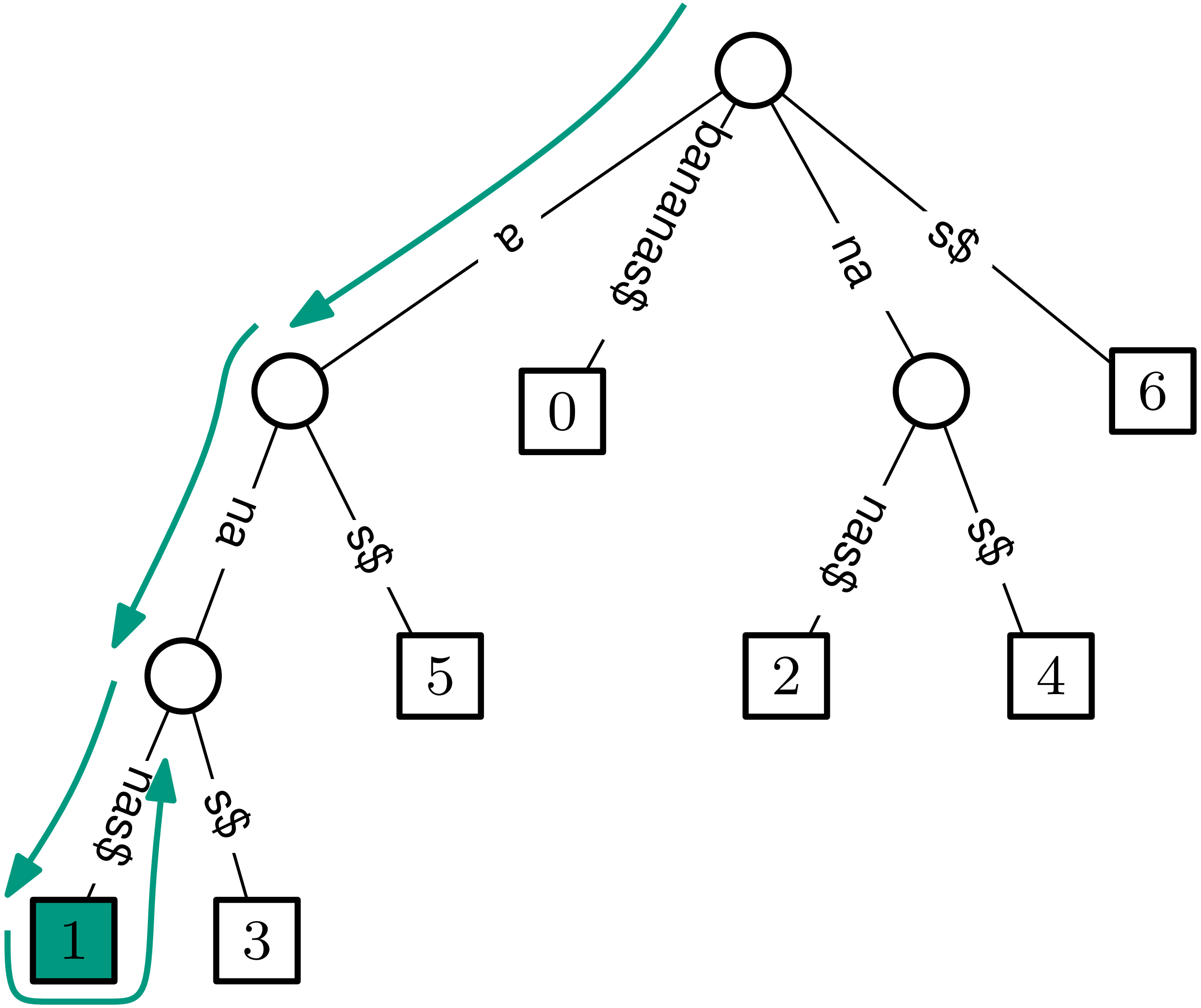
# Constructing the Suffix Array from the Suffix Tree



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Suffix Array

| | | | | | | |
|---|---|---|---|---|---|---|
| 🟩 1 | 3 | 5 | 0 | 2 | 4 | 6 |

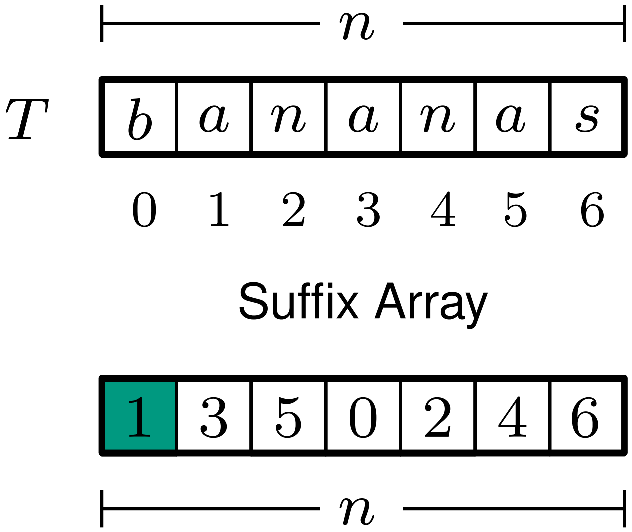


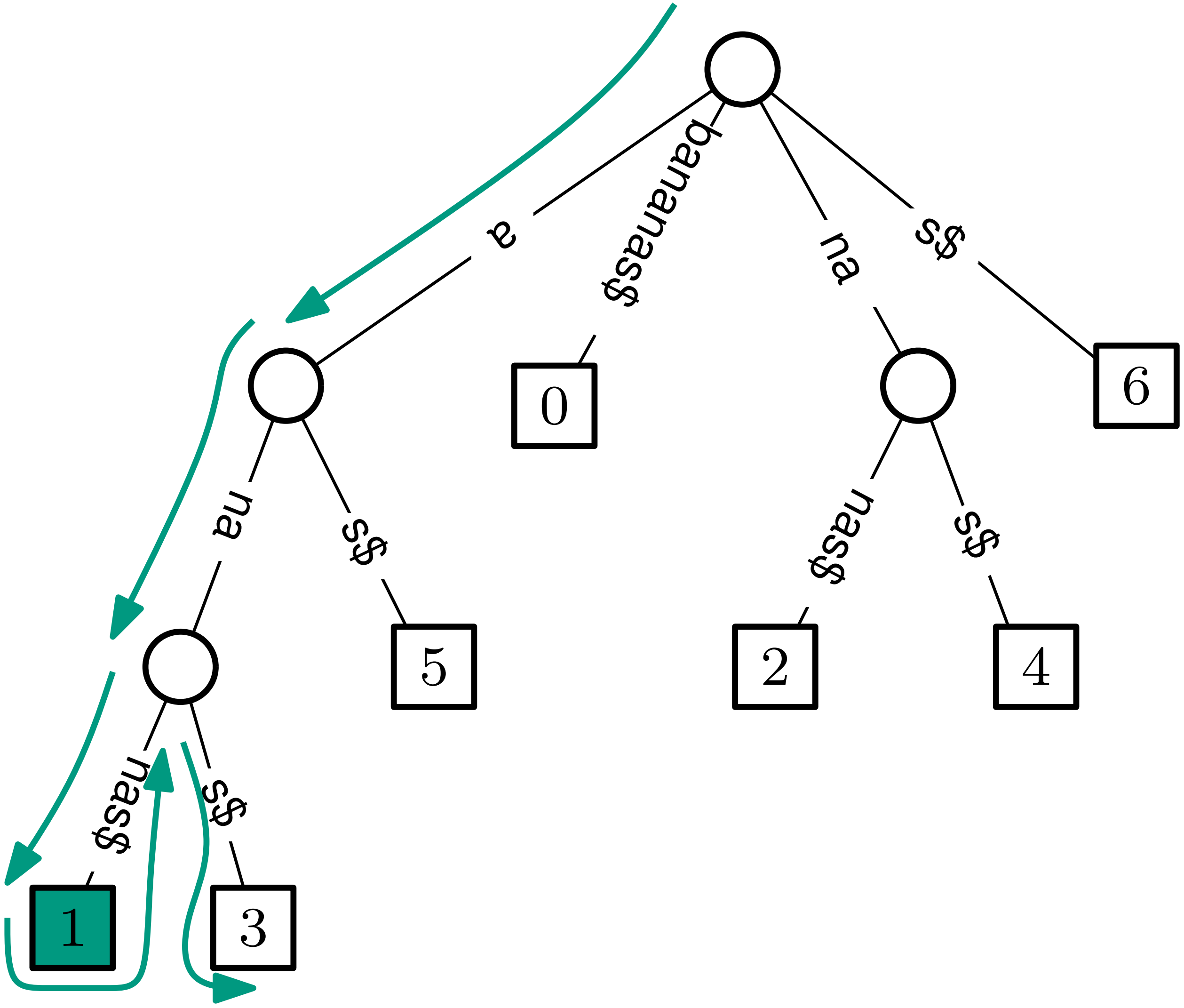
*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree





*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree

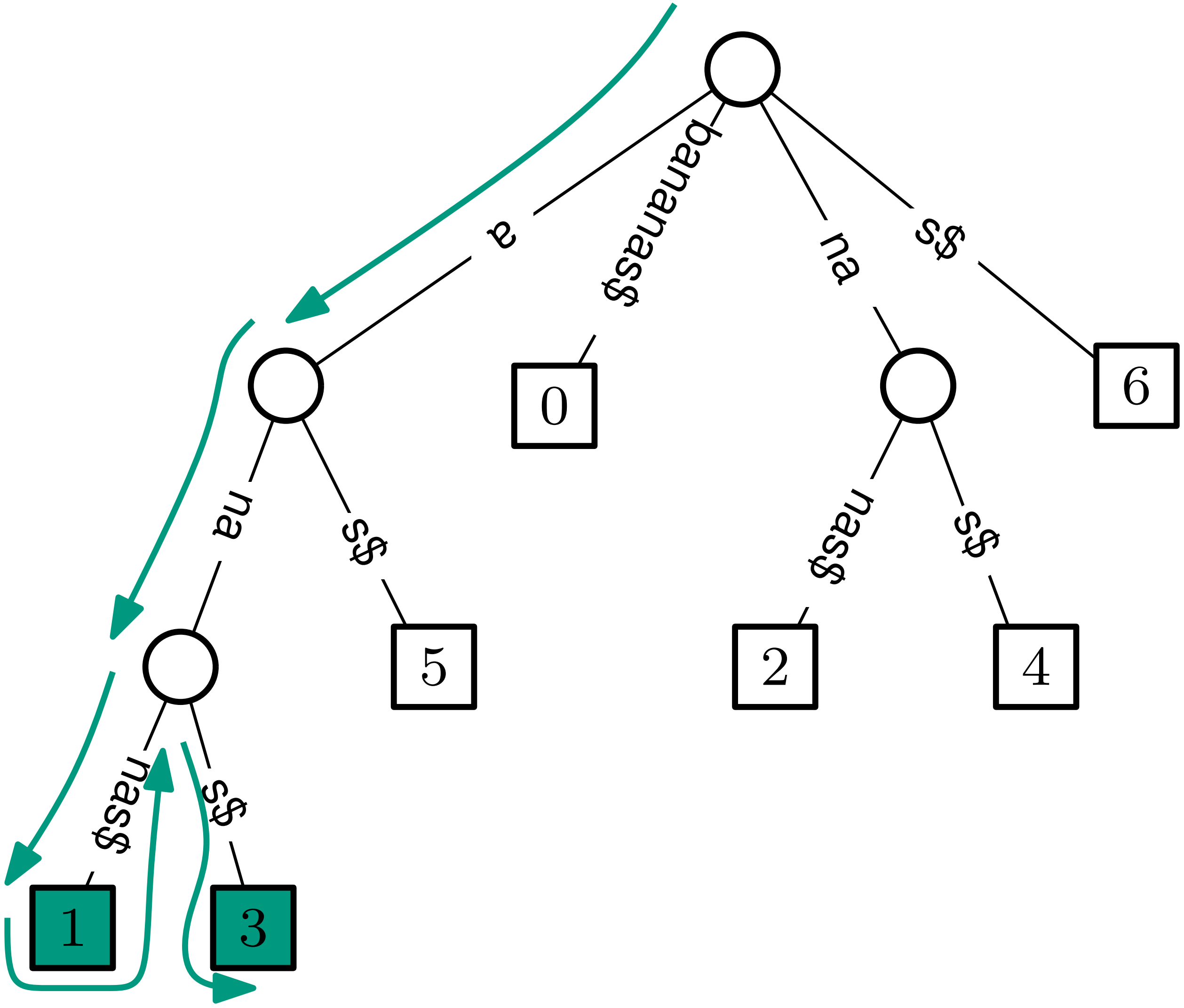
$n$

| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Suffix Array

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 0 | 2 | 4 | 6 |

$n$

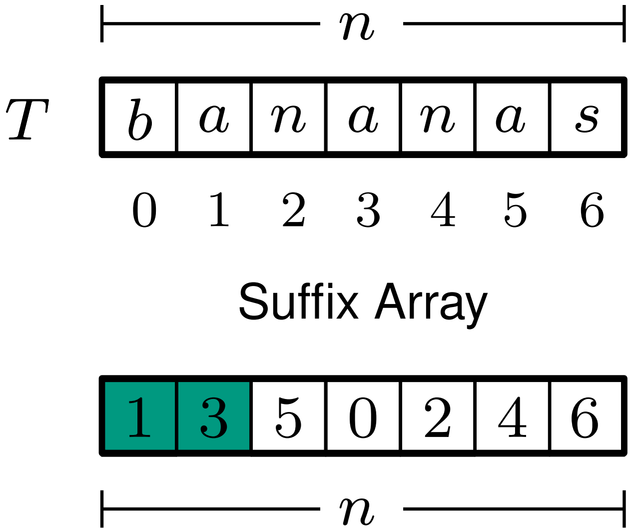


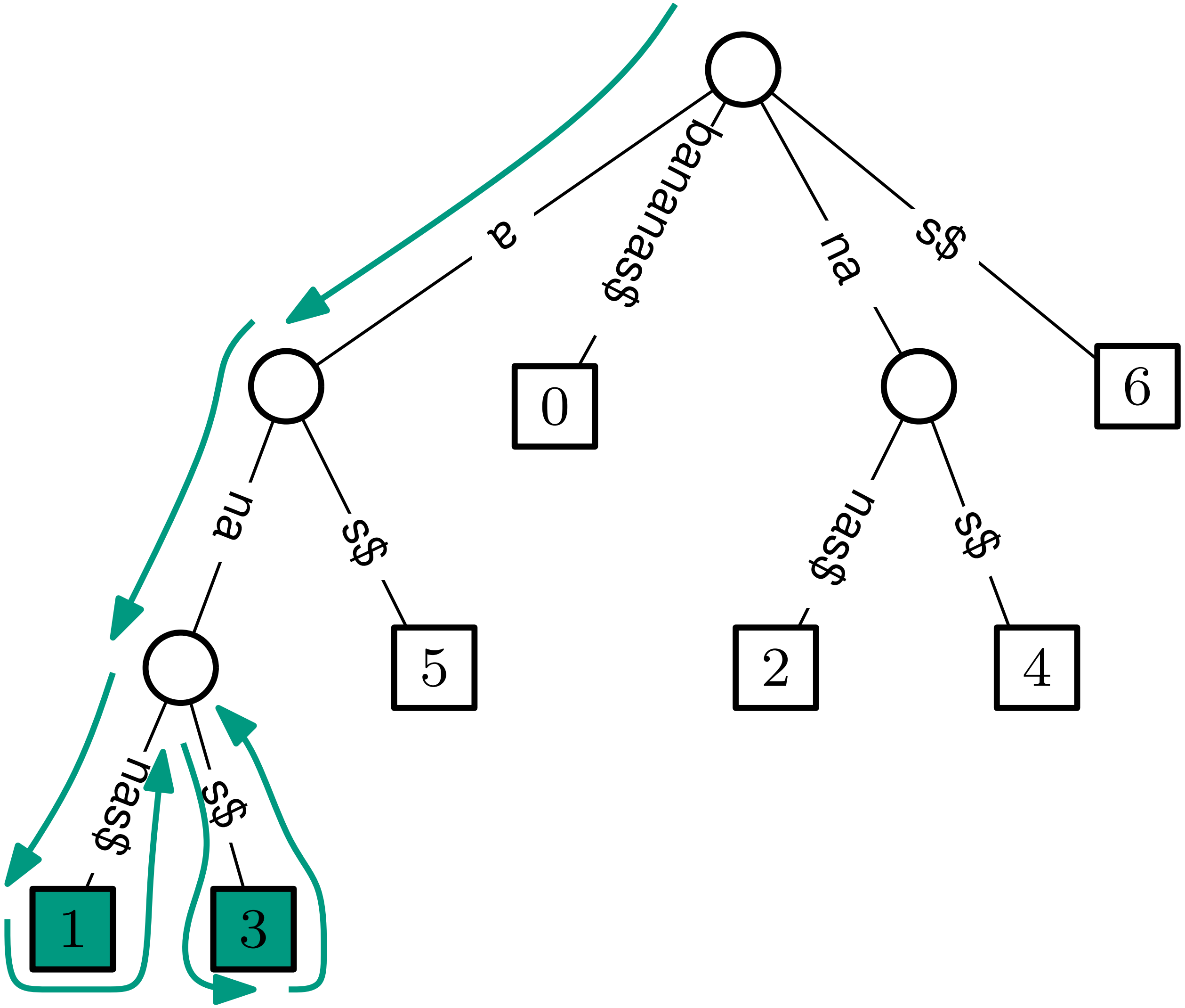
*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree





*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree

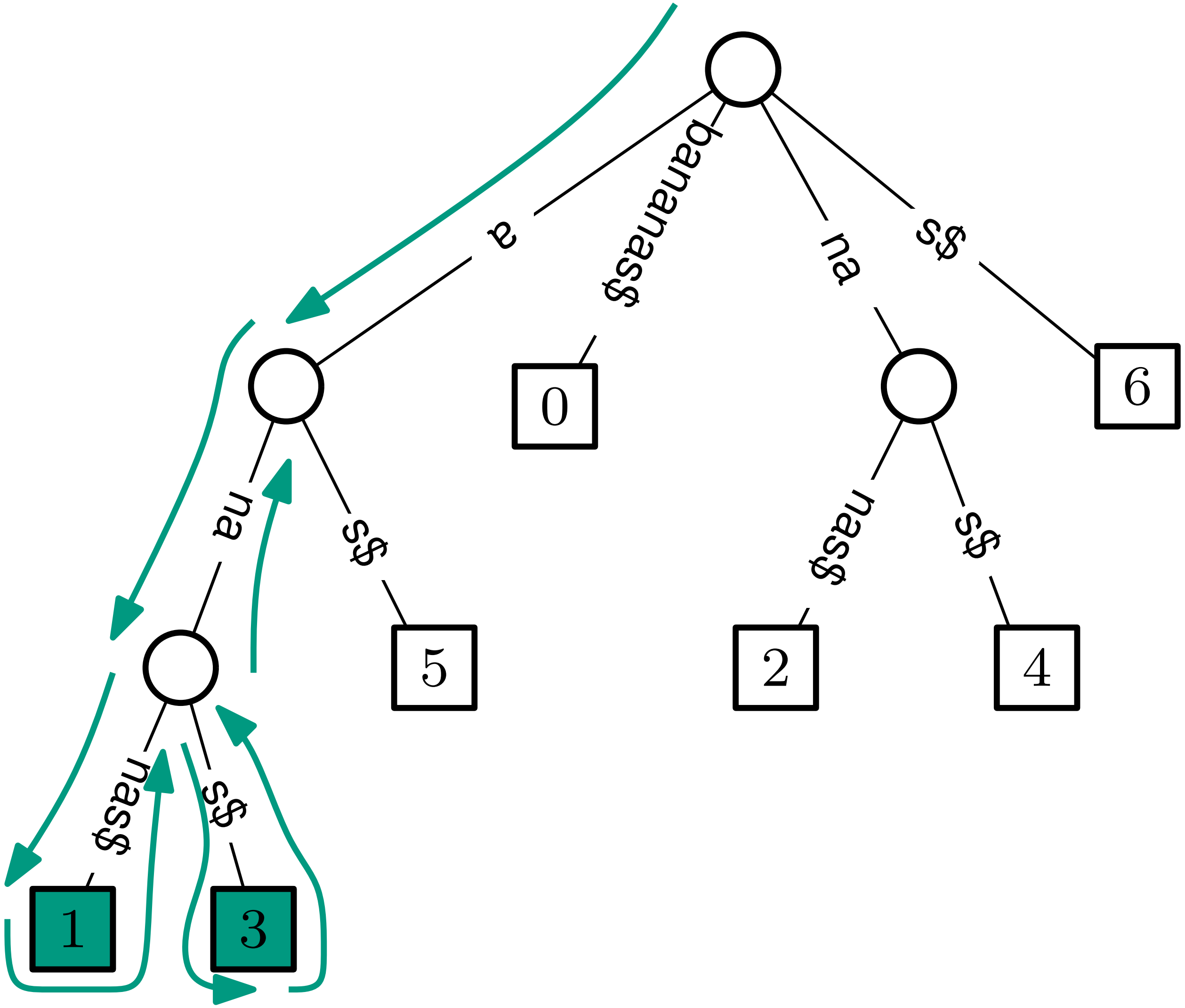
$n$

| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
|---|---|---|---|---|---|---|---|
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Suffix Array

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 0 | 2 | 4 | 6 |

$n$

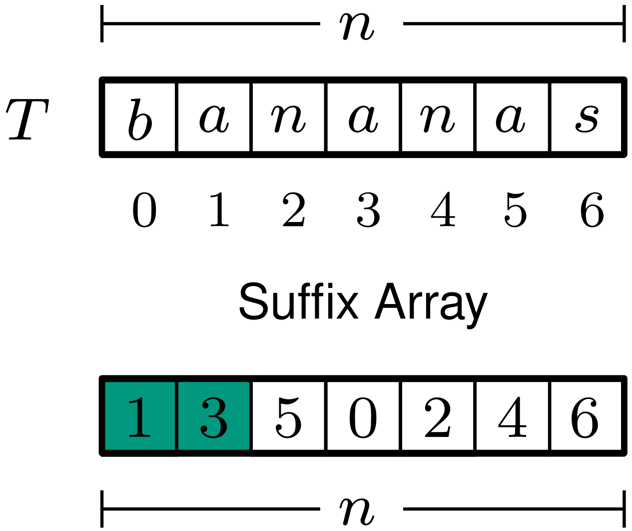


*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

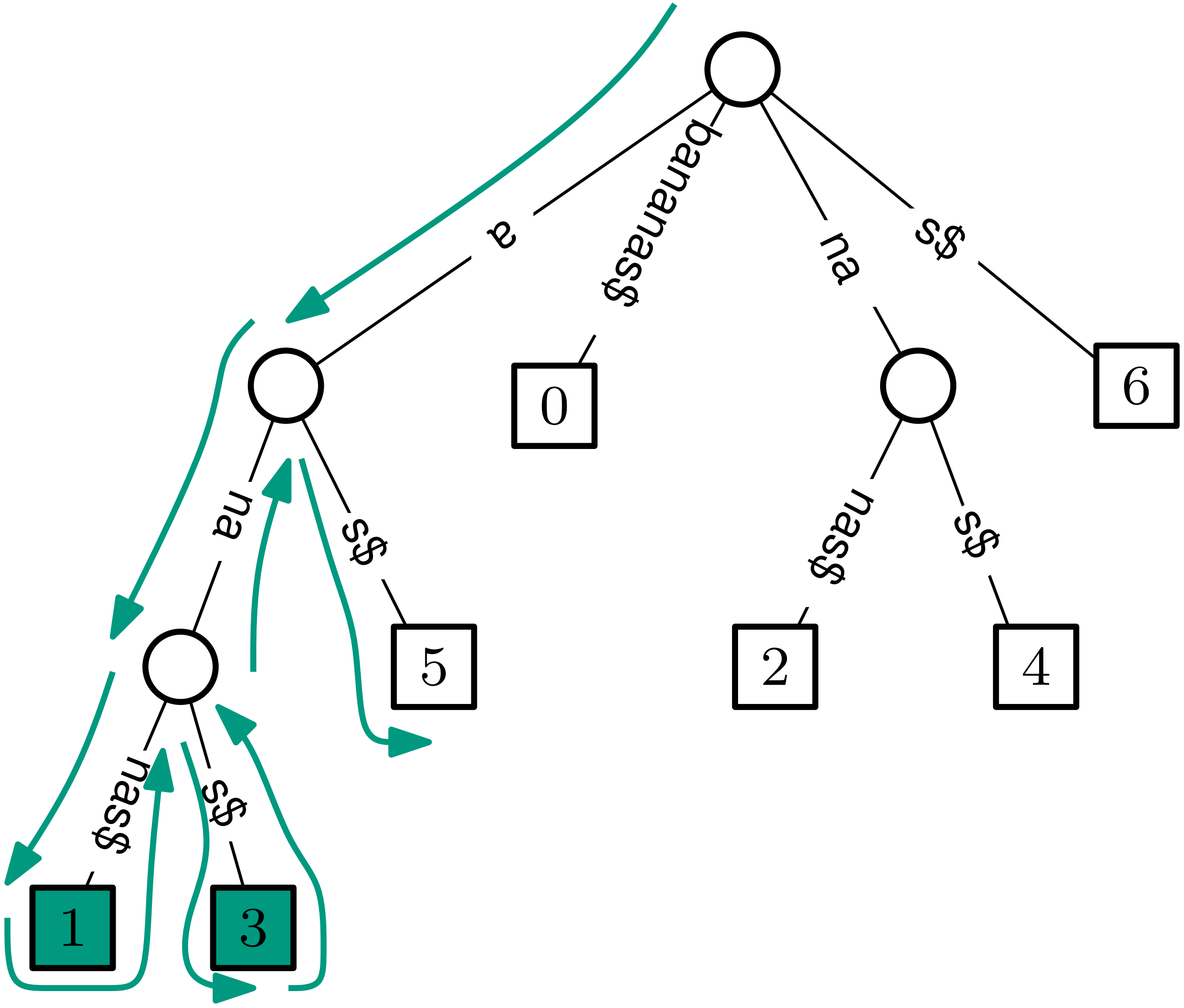
To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree



Suffix Array

| 1 | 3 | 5 | 0 | 2 | 4 | 6 |
|---|---|---|---|---|---|---|

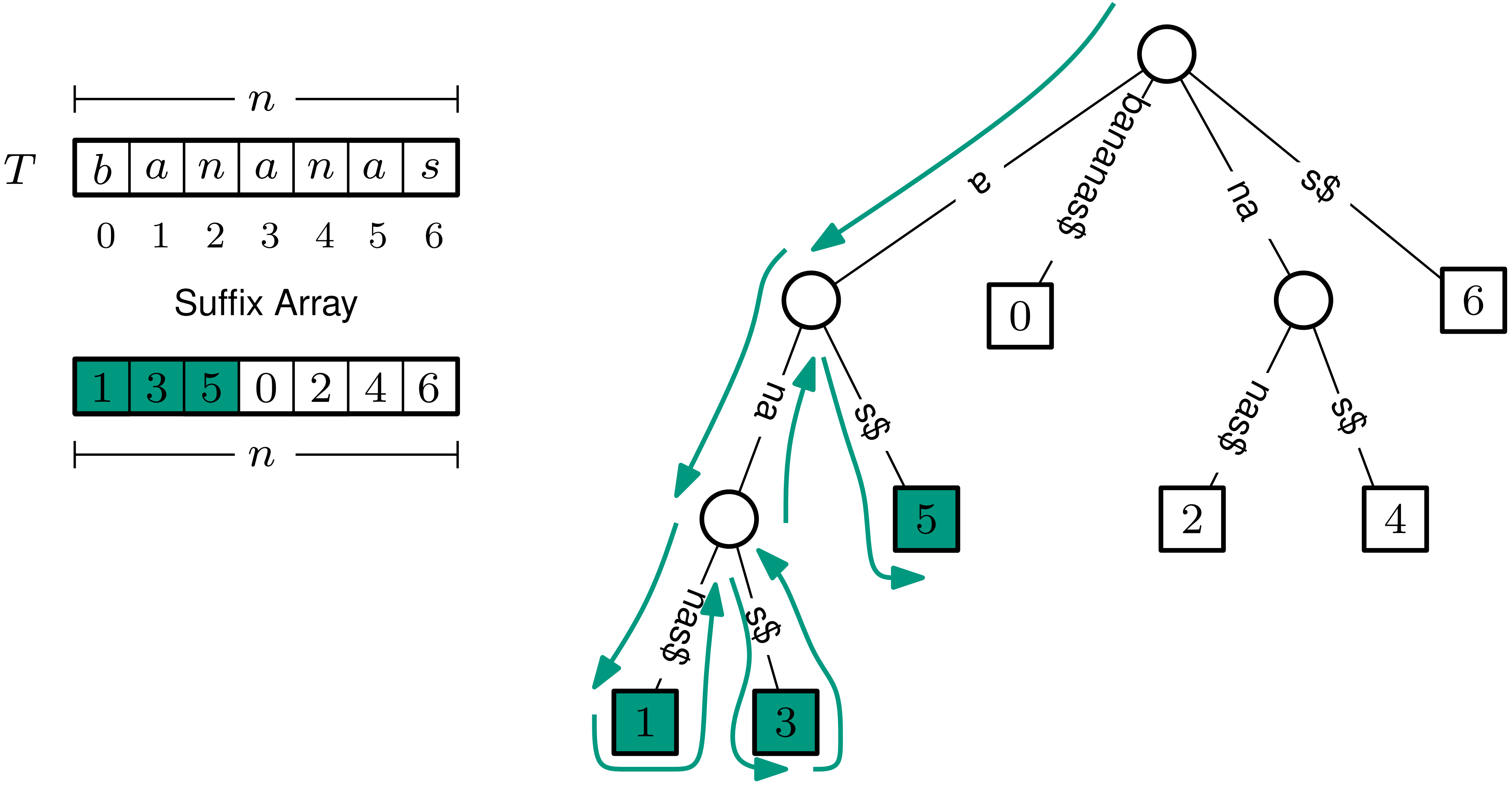


*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree



*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree

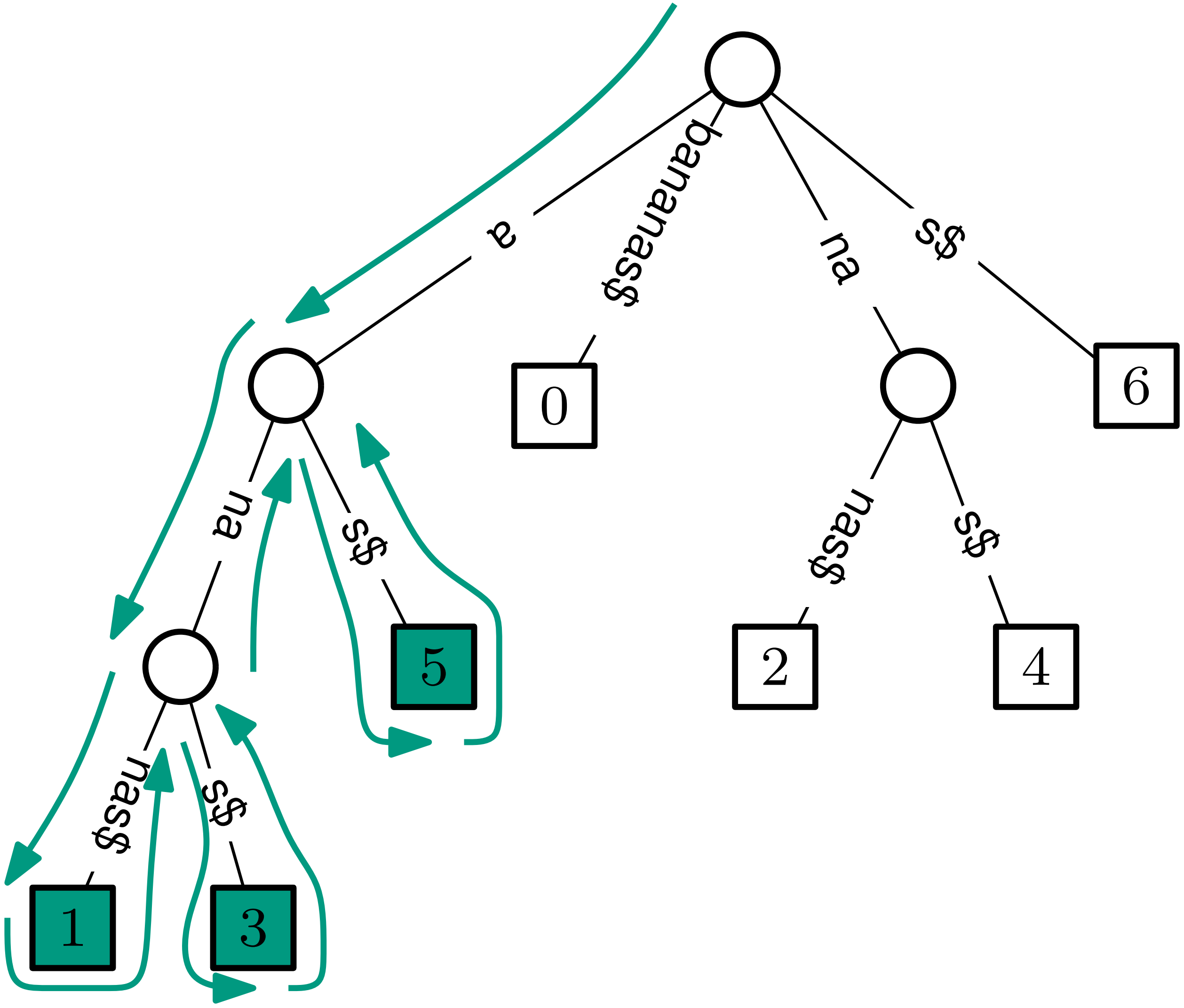
$n$

| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Suffix Array

| 1 | 3 | 5 | 0 | 2 | 4 | 6 |
|---|---|---|---|---|---|---|

$n$

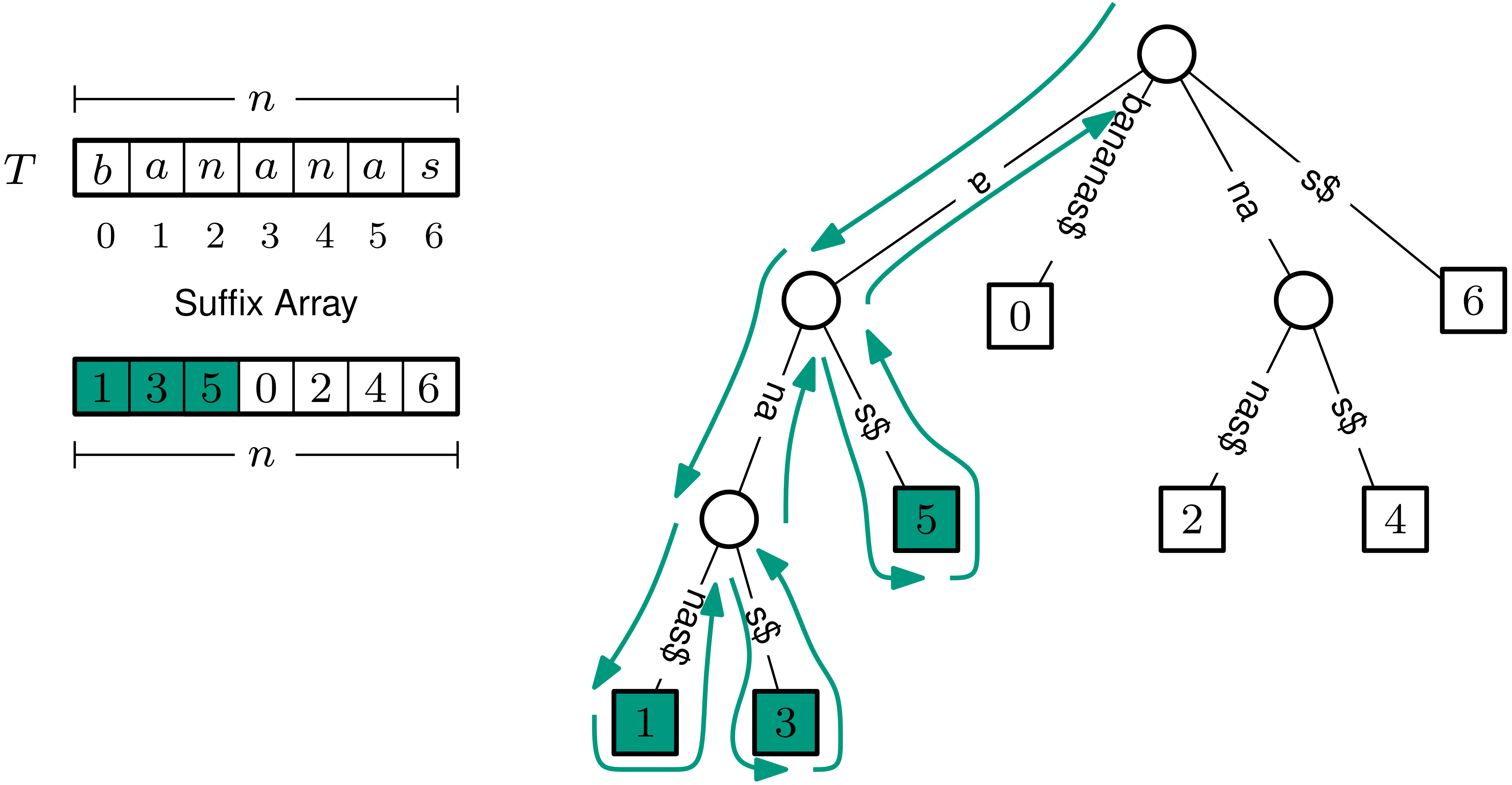


*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree



*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree

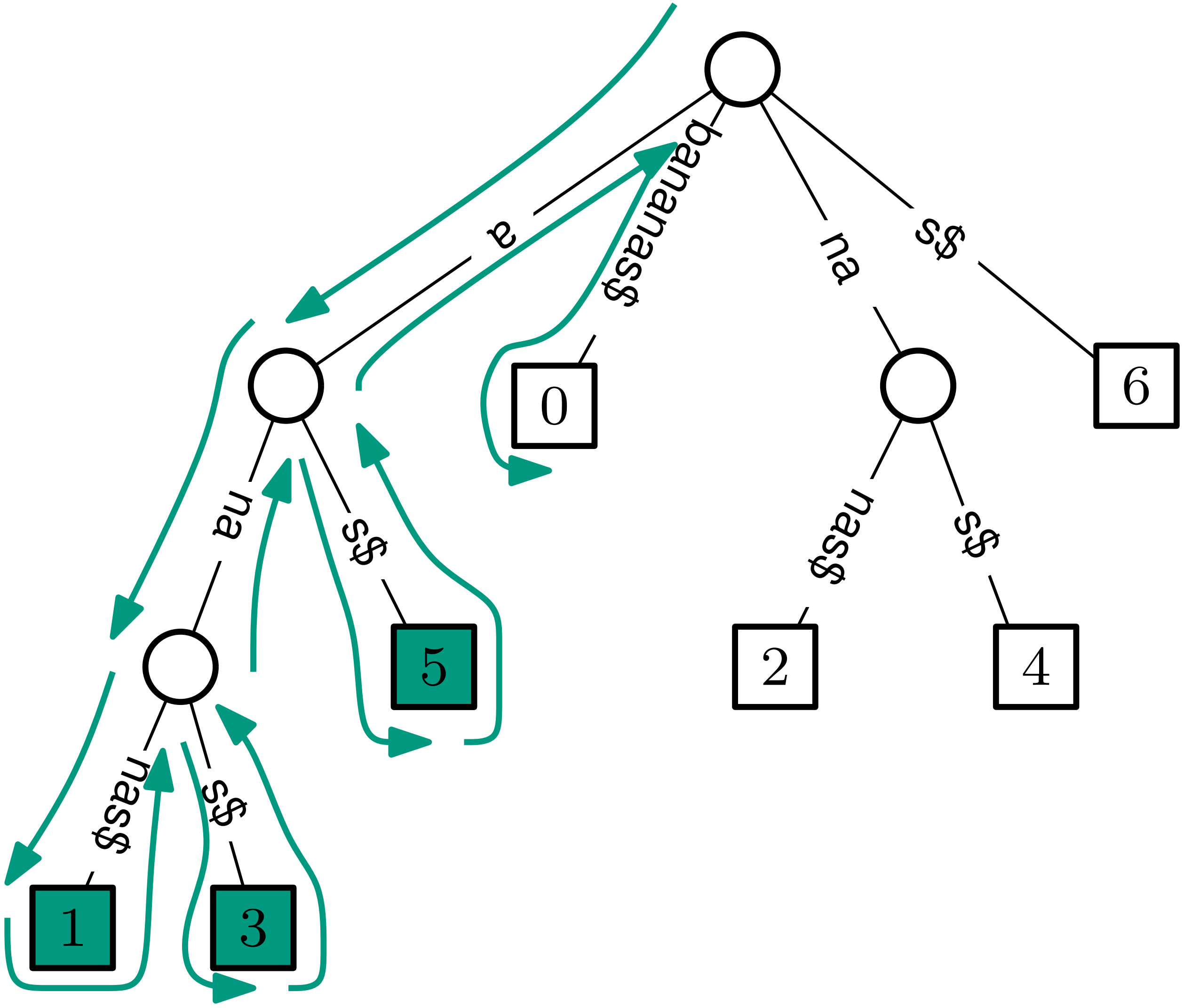
$n$

| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Suffix Array

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 0 | 2 | 4 | 6 |

$n$



*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree

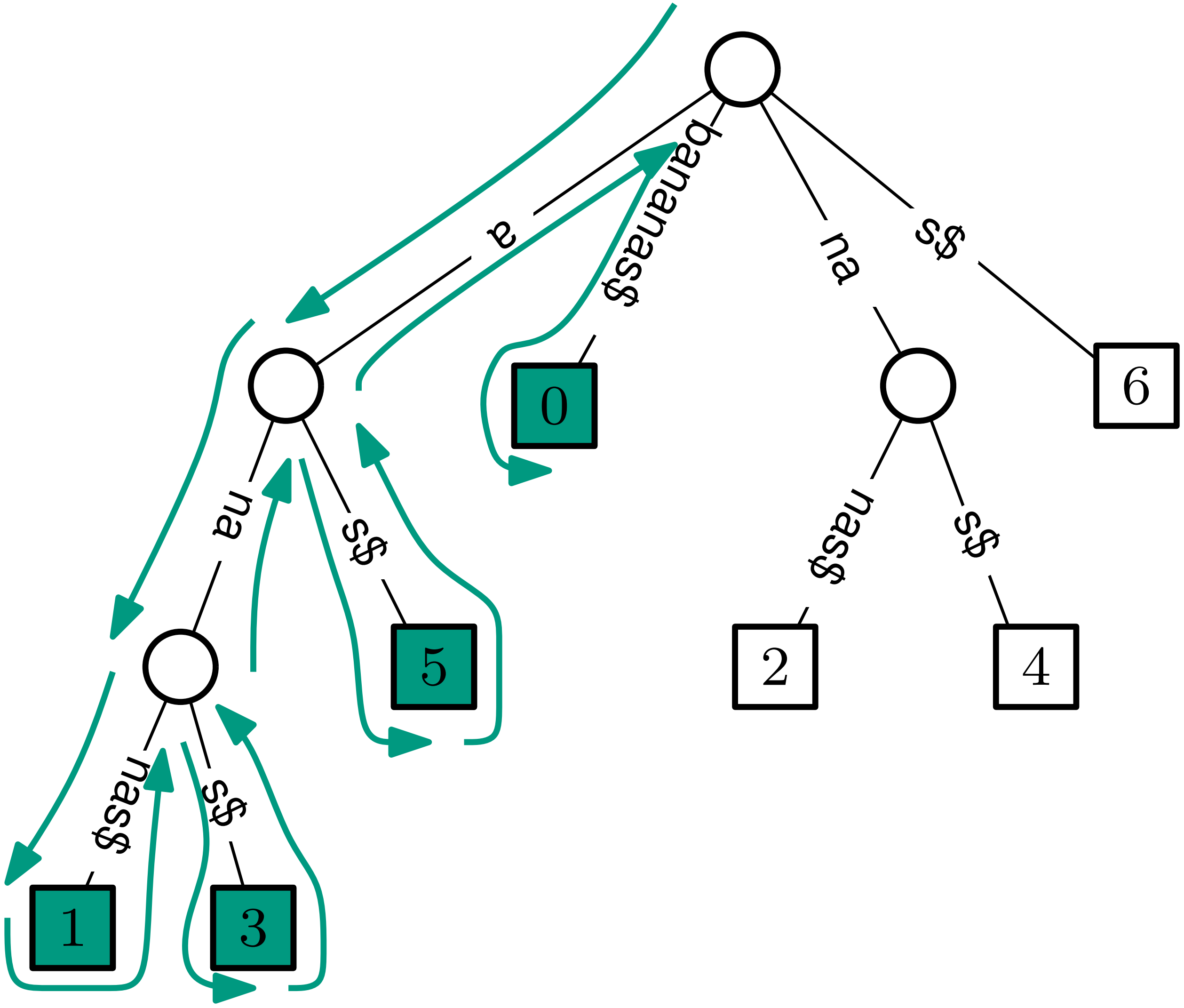
$n$

| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Suffix Array

| 1 | 3 | 5 | 0 | 2 | 4 | 6 |
|---|---|---|---|---|---|---|

$n$

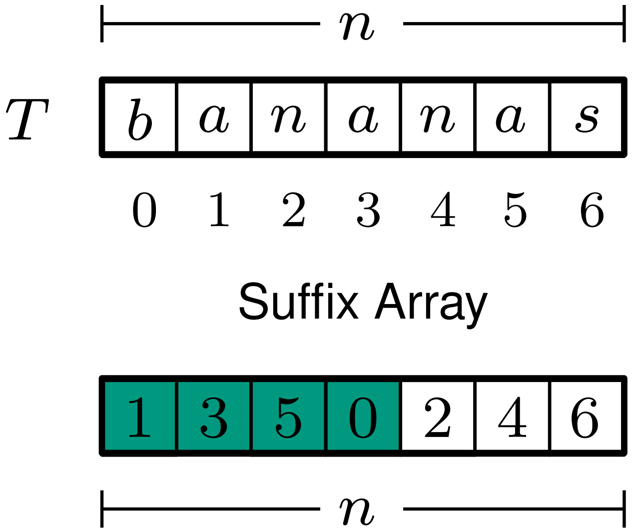


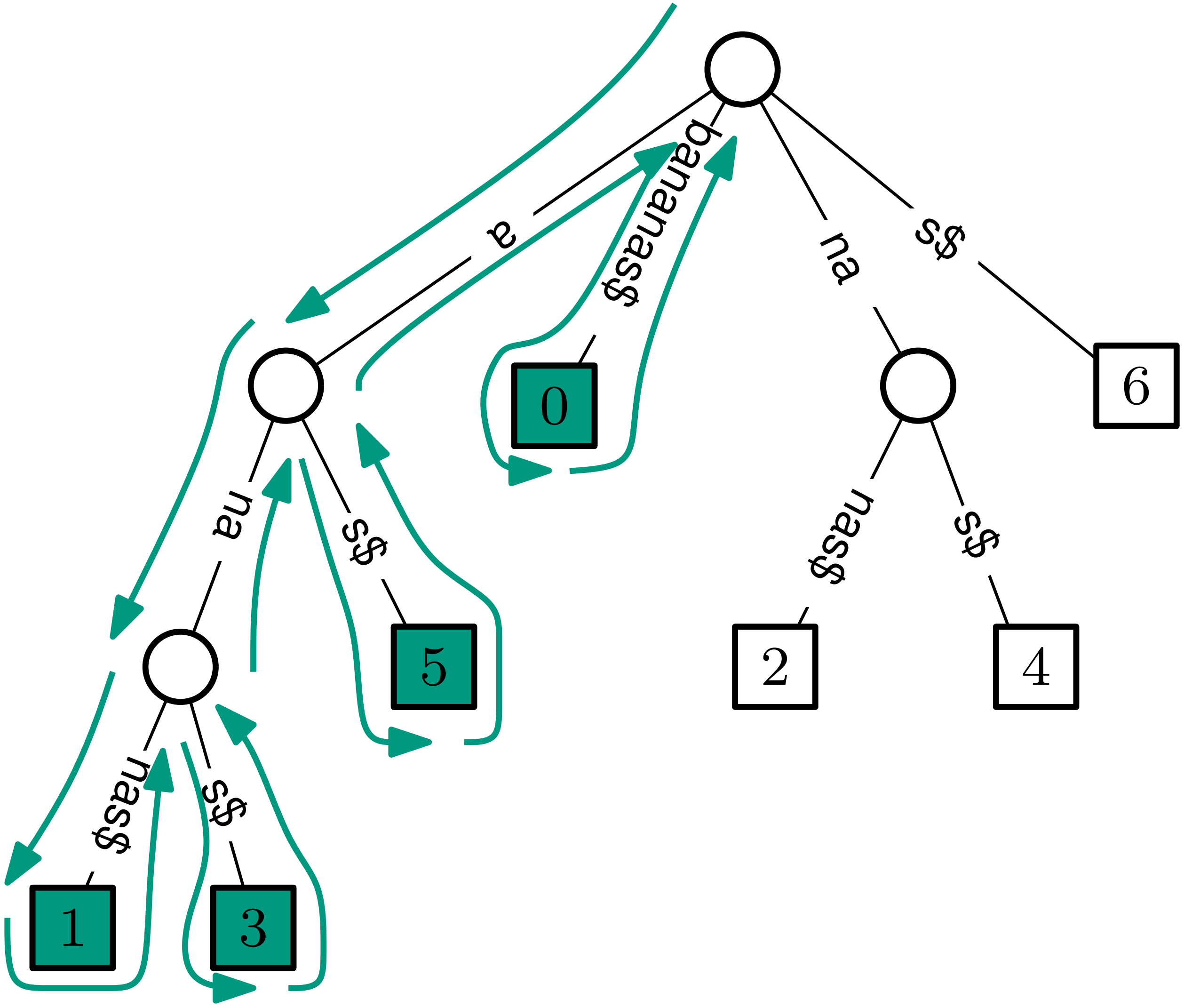
*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree



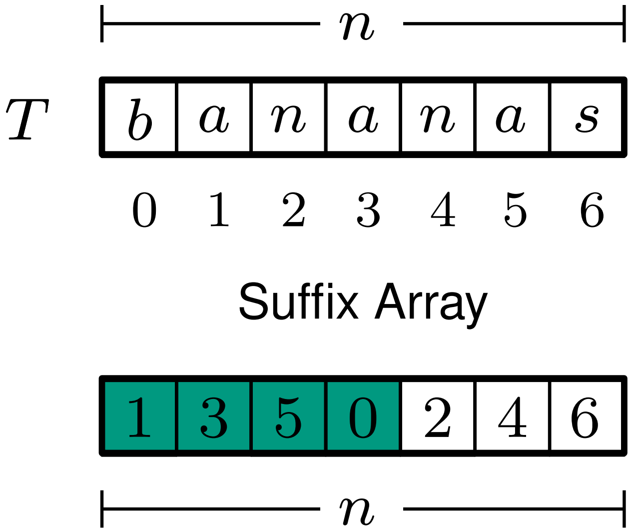


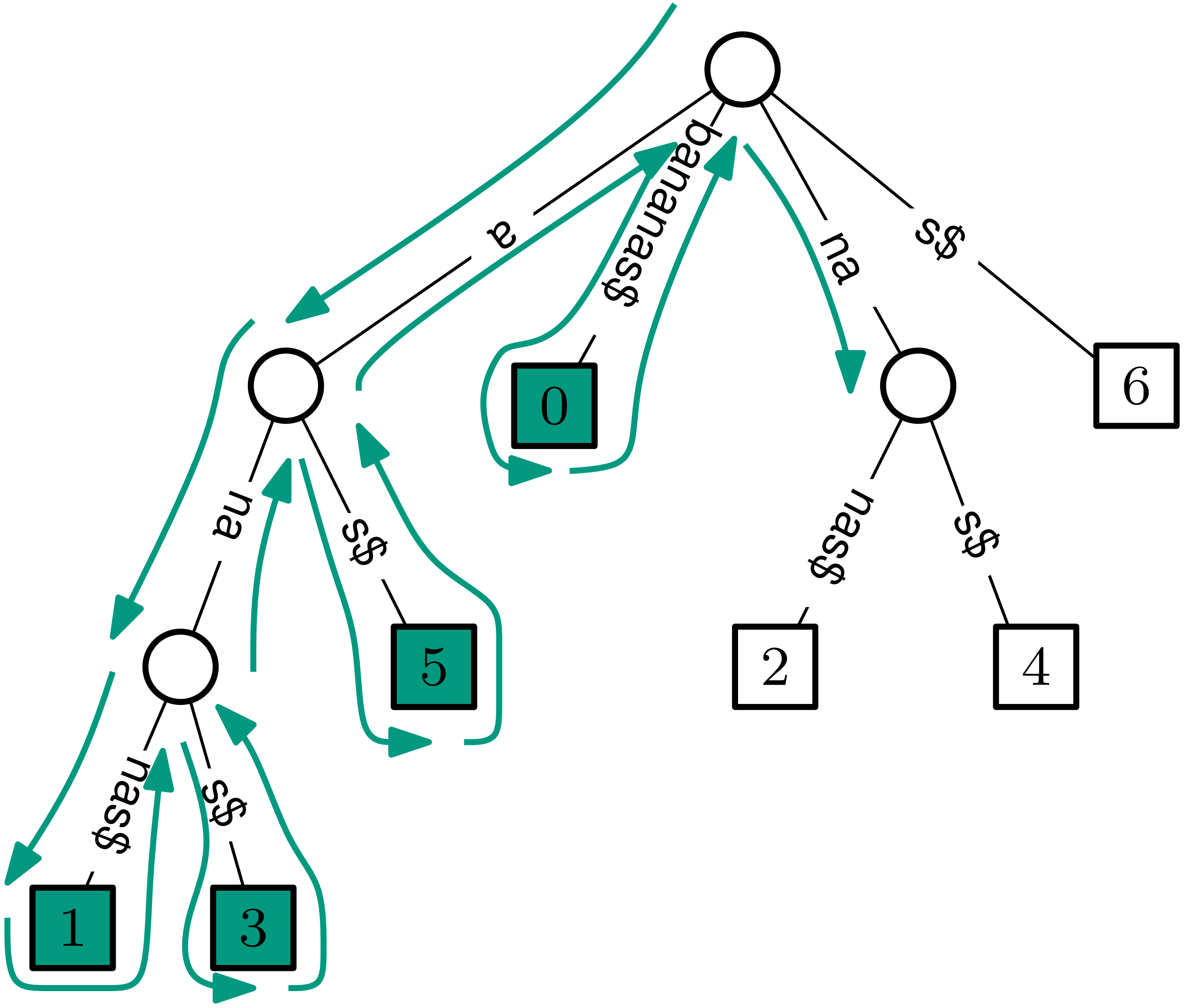
*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree



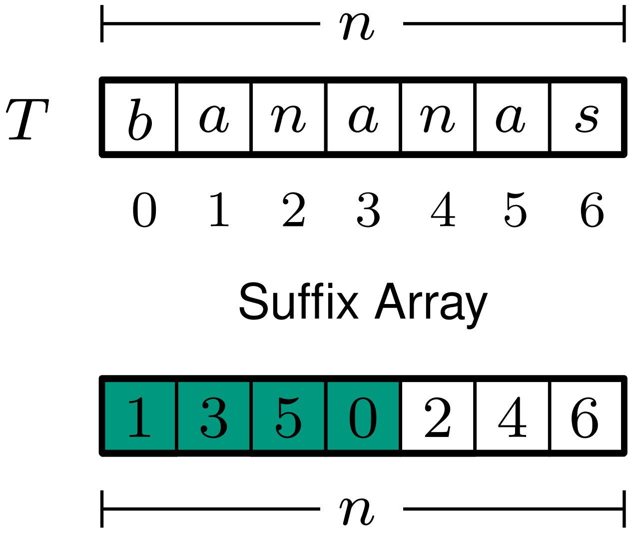


*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree



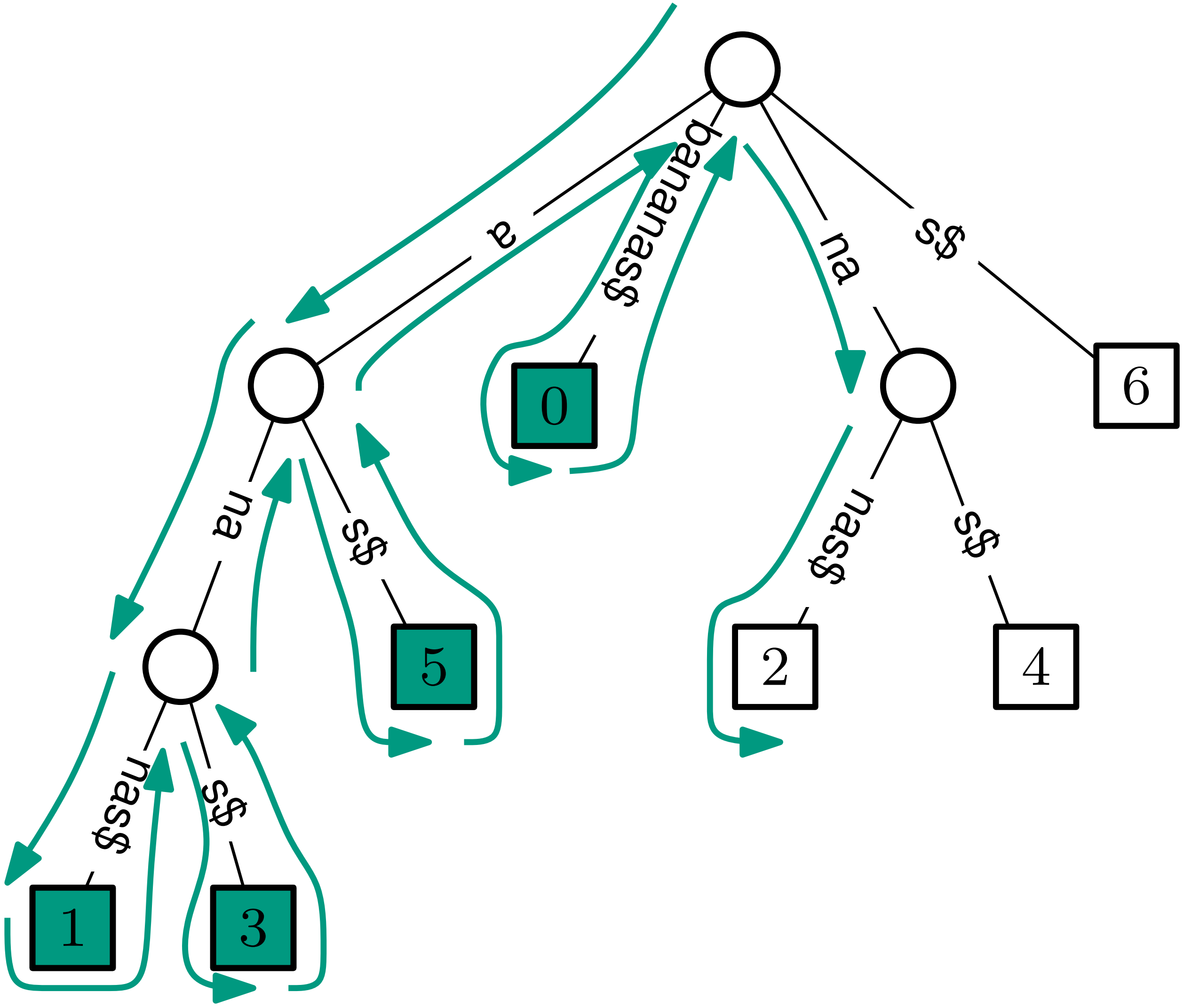
$n$

| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Suffix Array

| 1 | 3 | 5 | 0 | 2 | 4 | 6 |
|---|---|---|---|---|---|---|

$n$

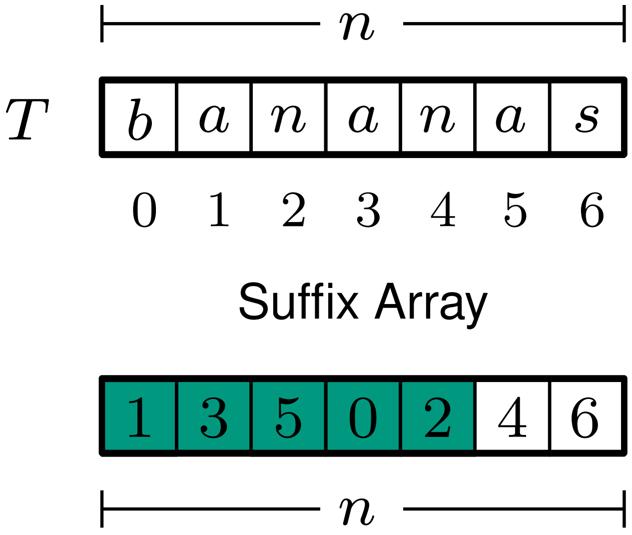


*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

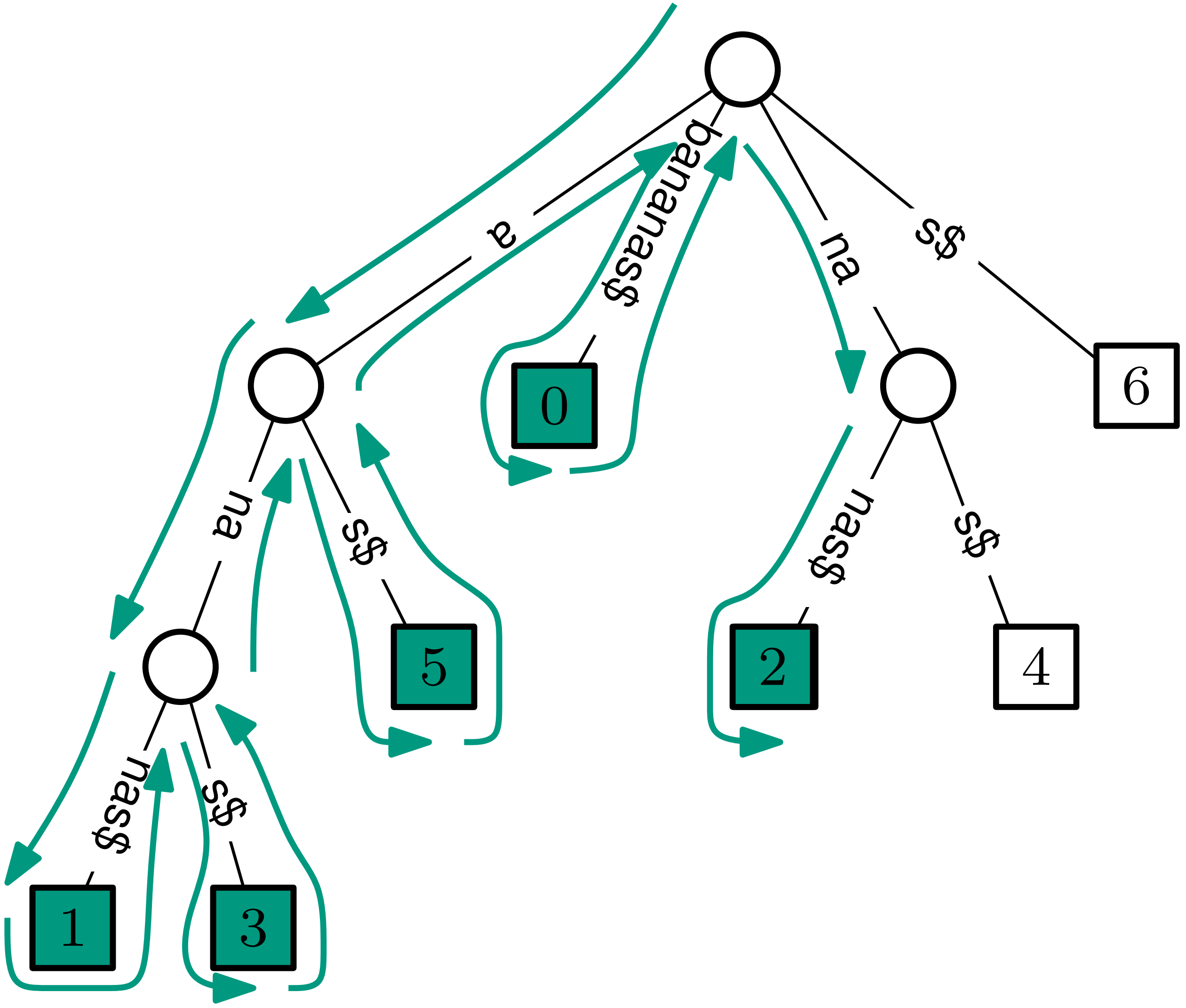
# Constructing the Suffix Array from the Suffix Tree



| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |

Suffix Array

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 0 | 2 | 4 | 6 |

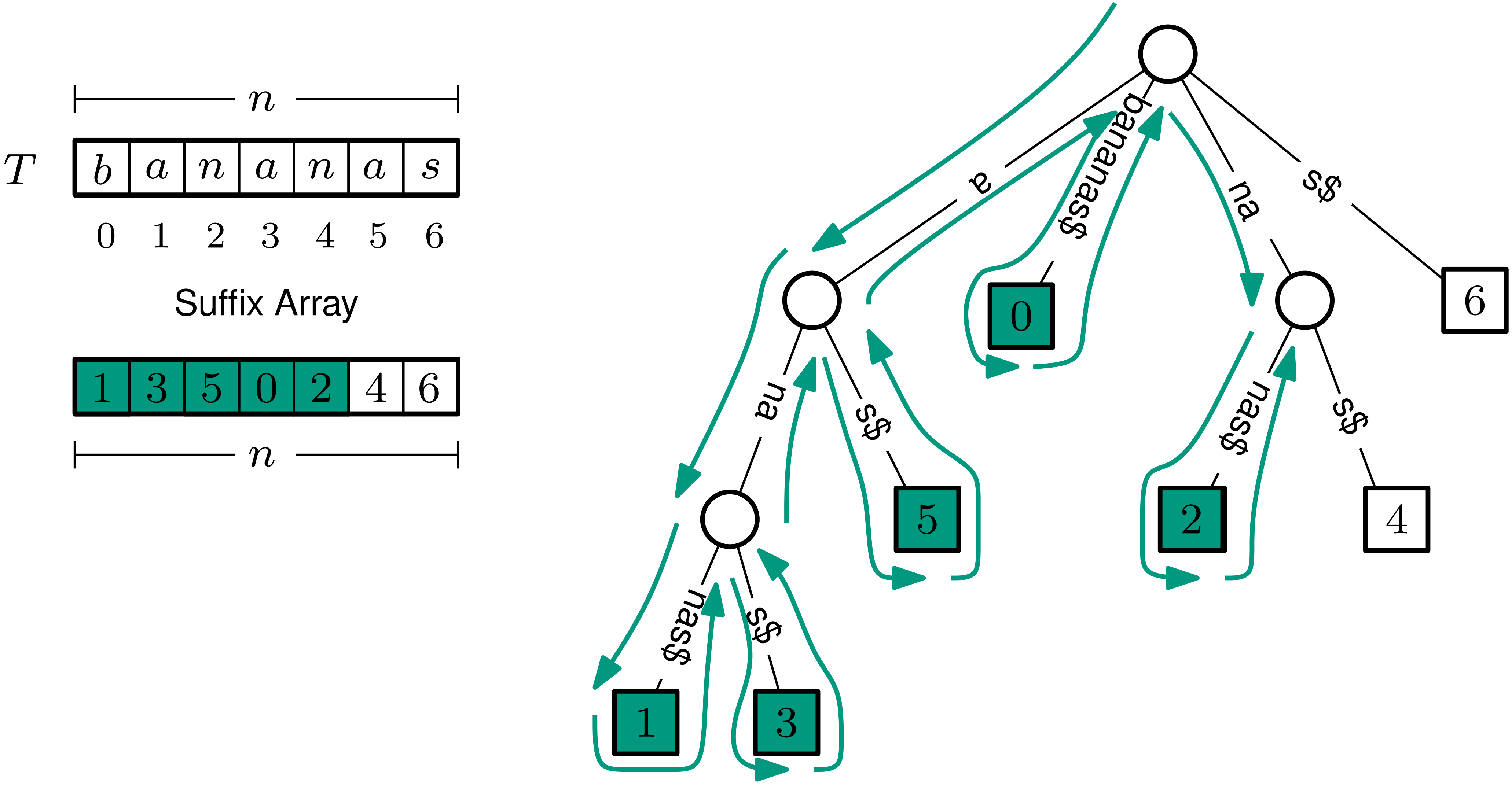


*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree

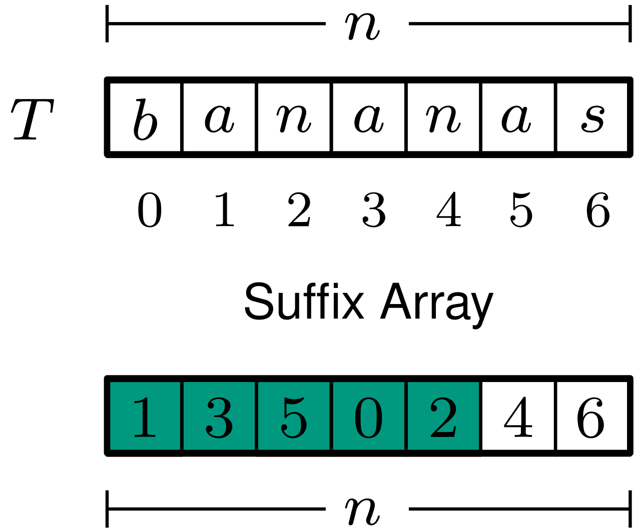


*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

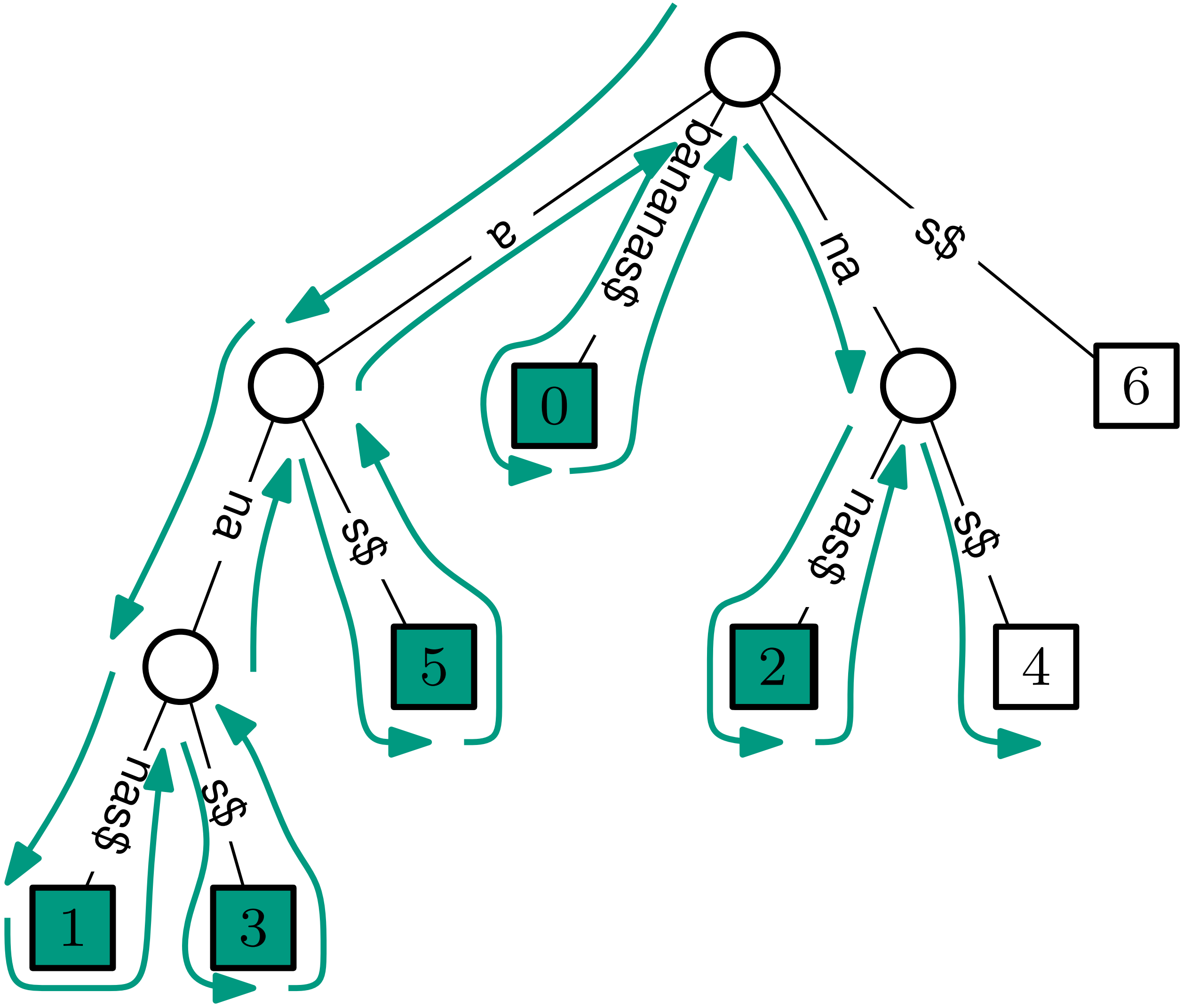
# Constructing the Suffix Array from the Suffix Tree



| | | | | | | |
|---|---|---|---|---|---|---|
| $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Suffix Array

| | | | | | | |
|---|---|---|---|---|---|---|
| 🟩 1 | 🟩 3 | 🟩 5 | 🟩 0 | 🟩 2 | 4 | 6 |



*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree

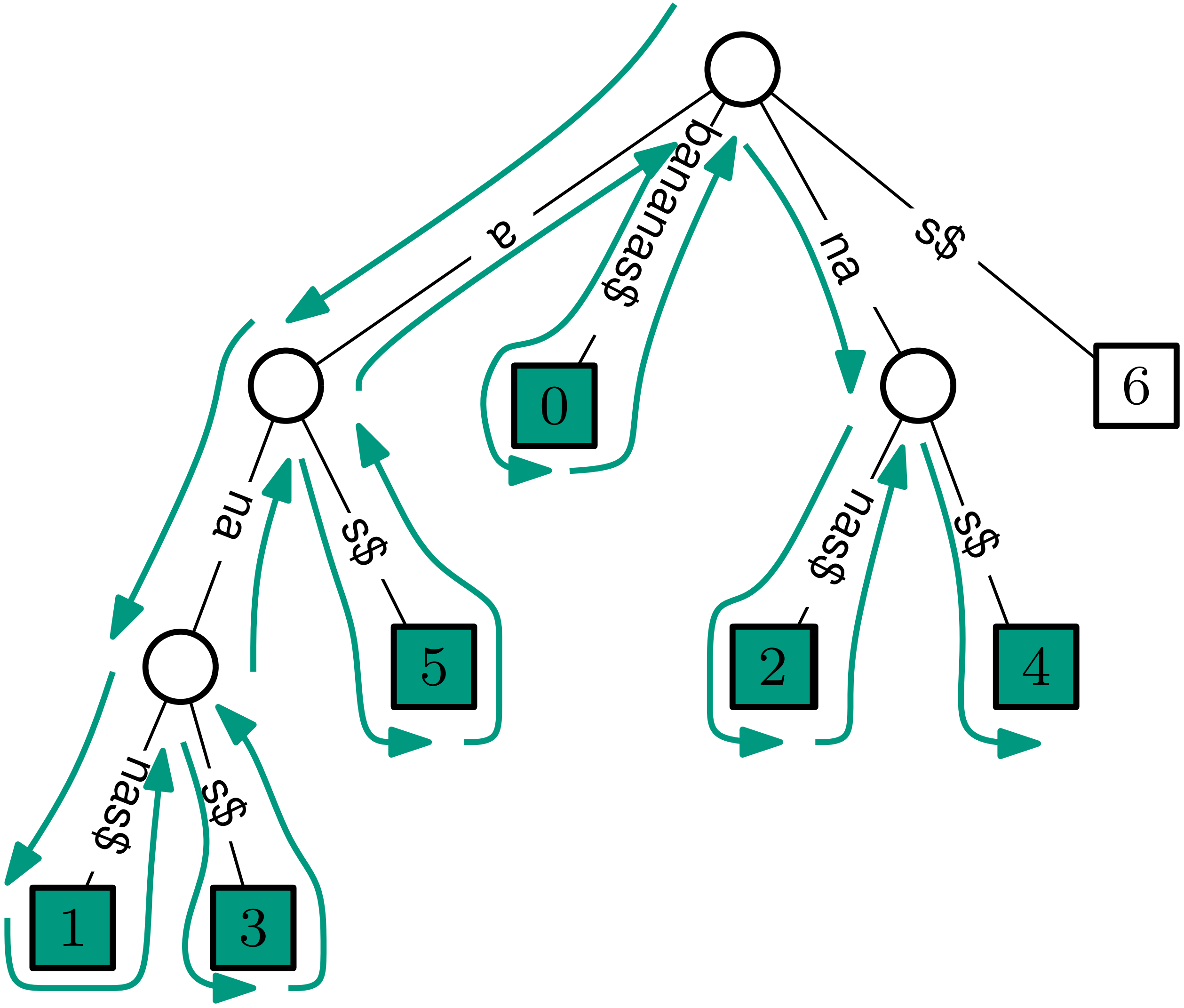
$n$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $T$ | $b$ | $a$ | $n$ | $a$ | $n$ | $a$ | $s$ |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Suffix Array

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 0 | 2 | 4 | 6 |

$n$

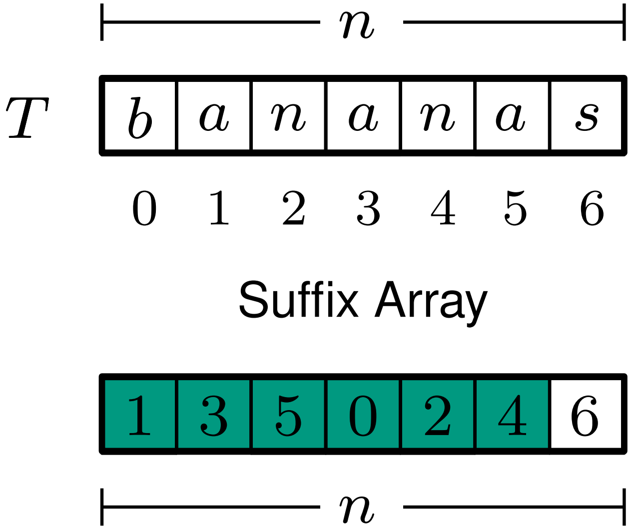


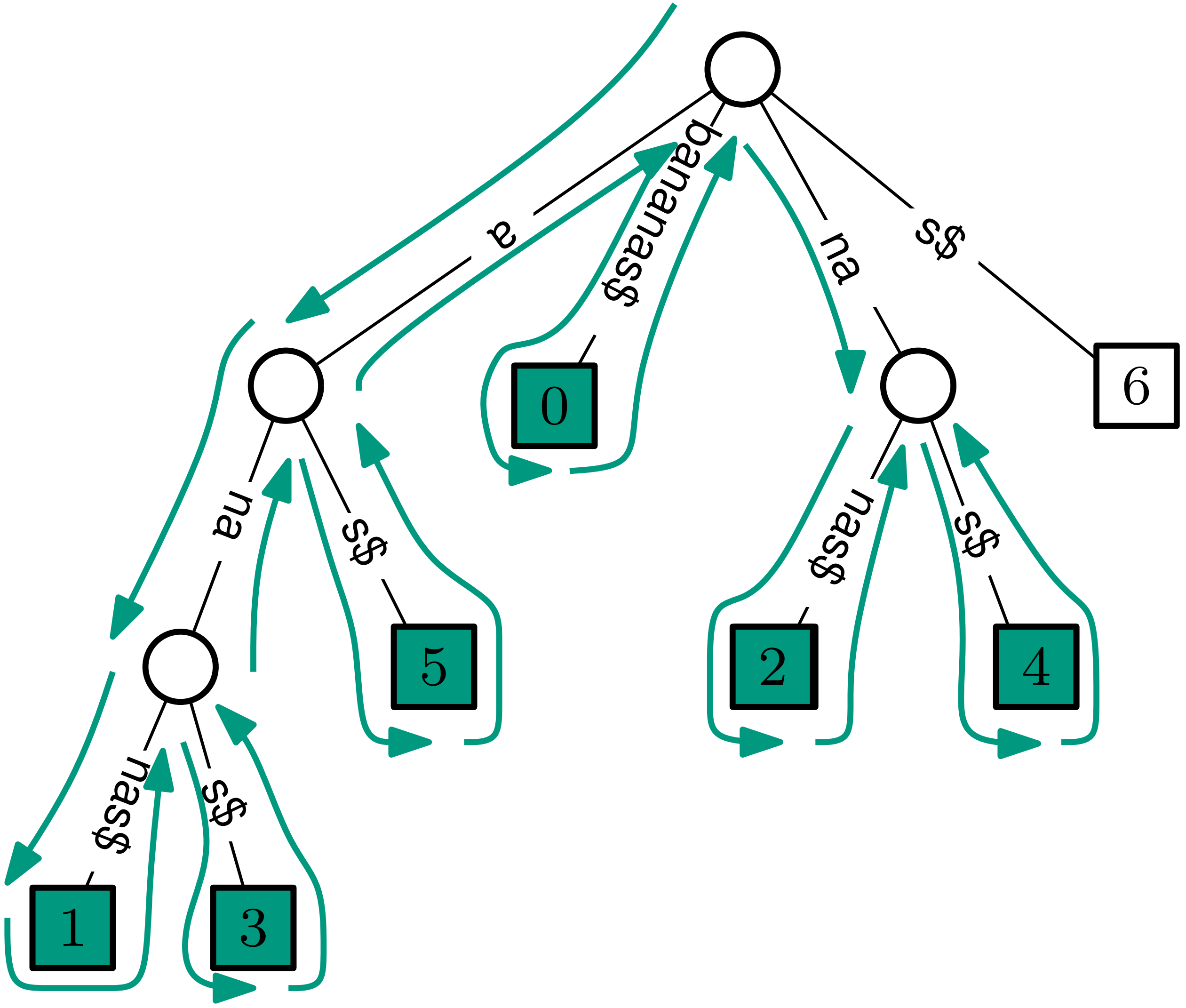
*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree



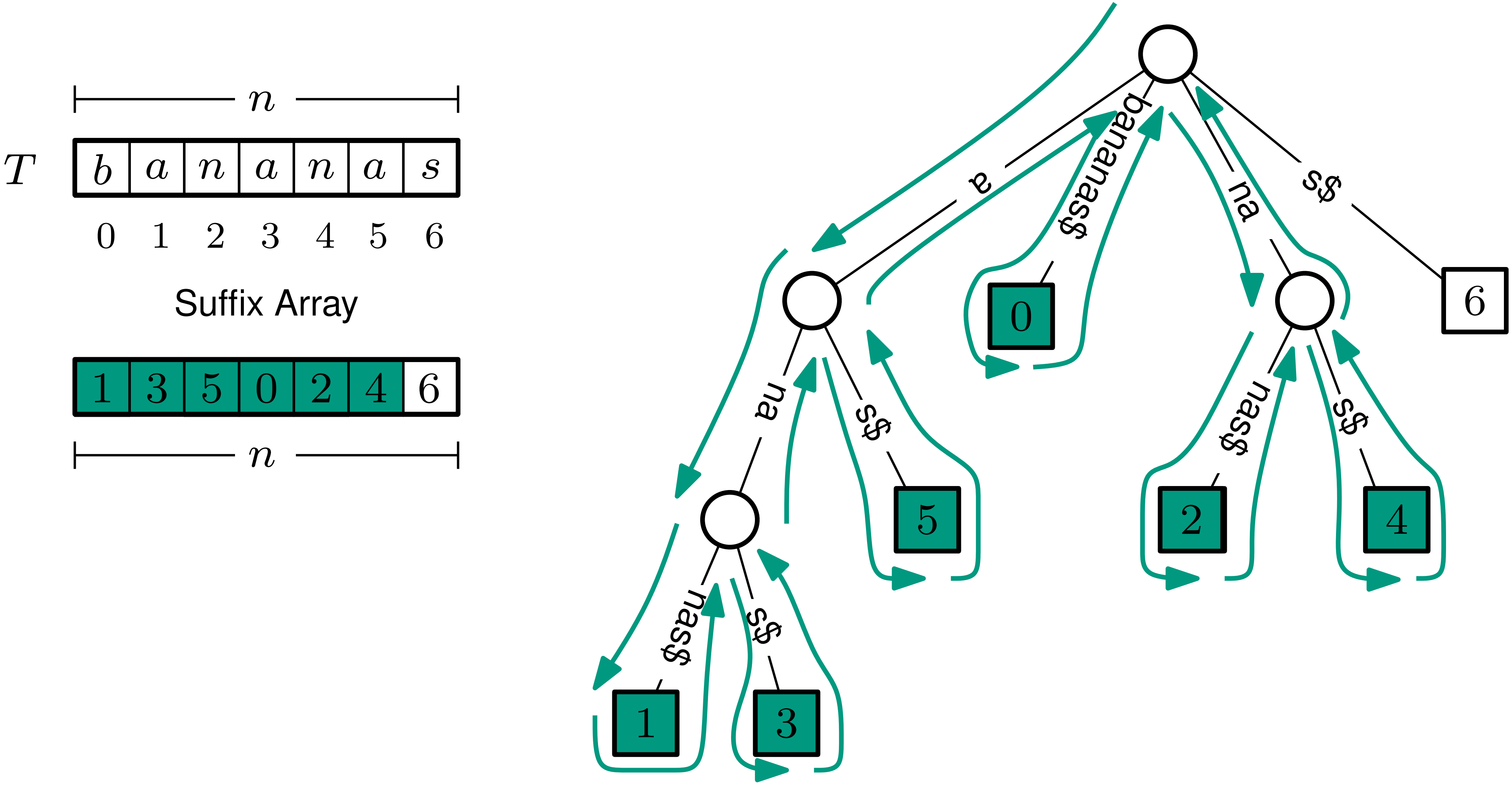


*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree

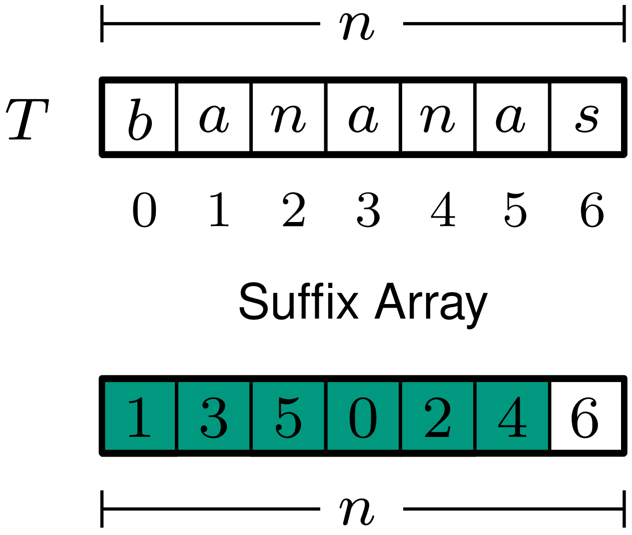


*recall that we added a unique symbol $ to make sure the tree exists*

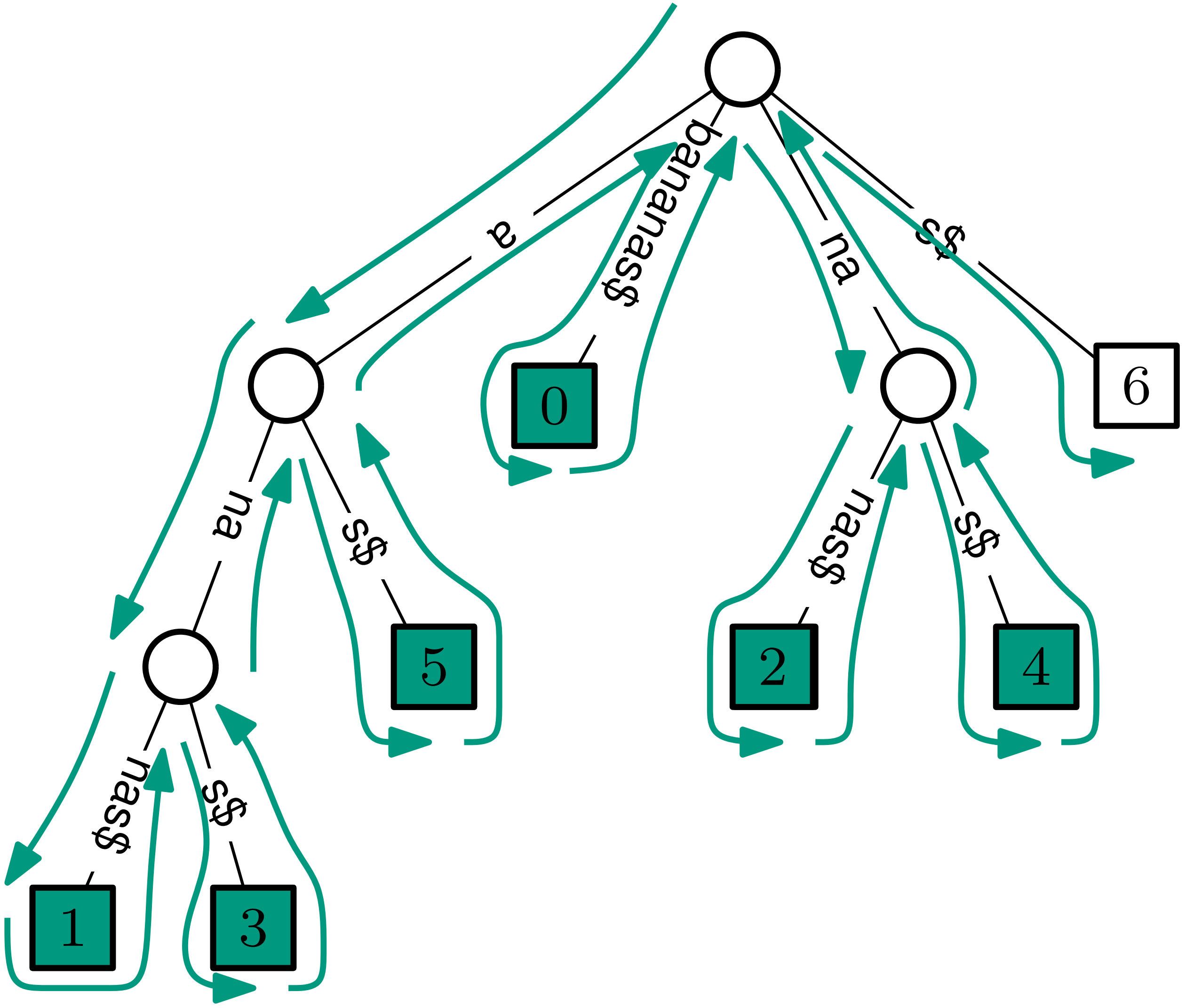
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree



Suffix Array

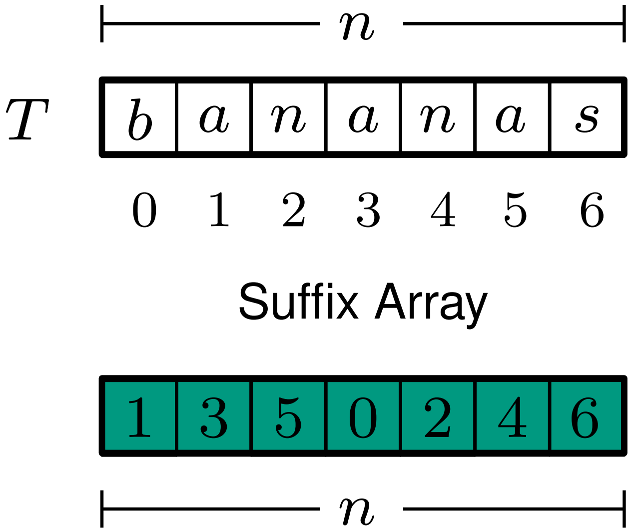


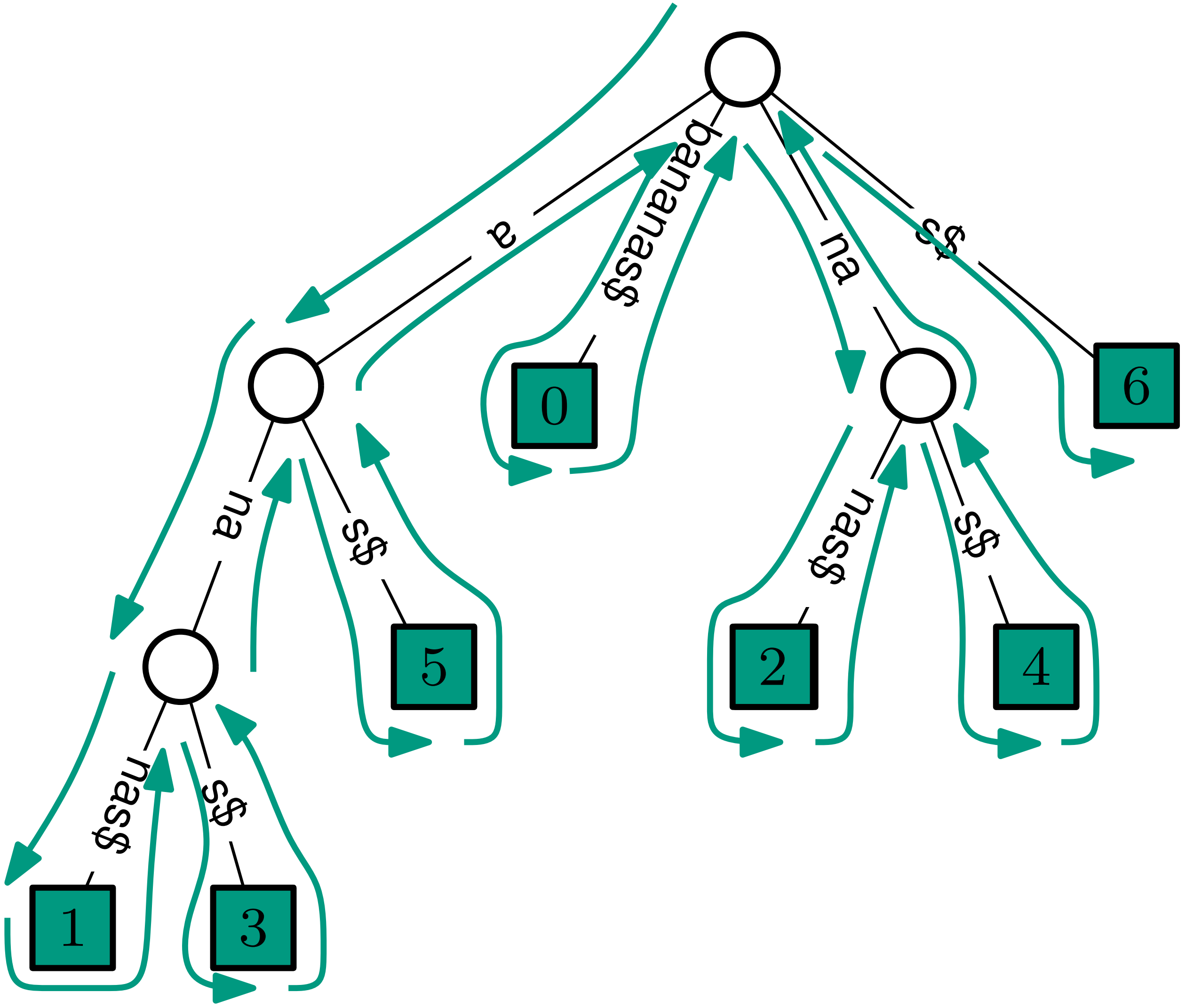
*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree



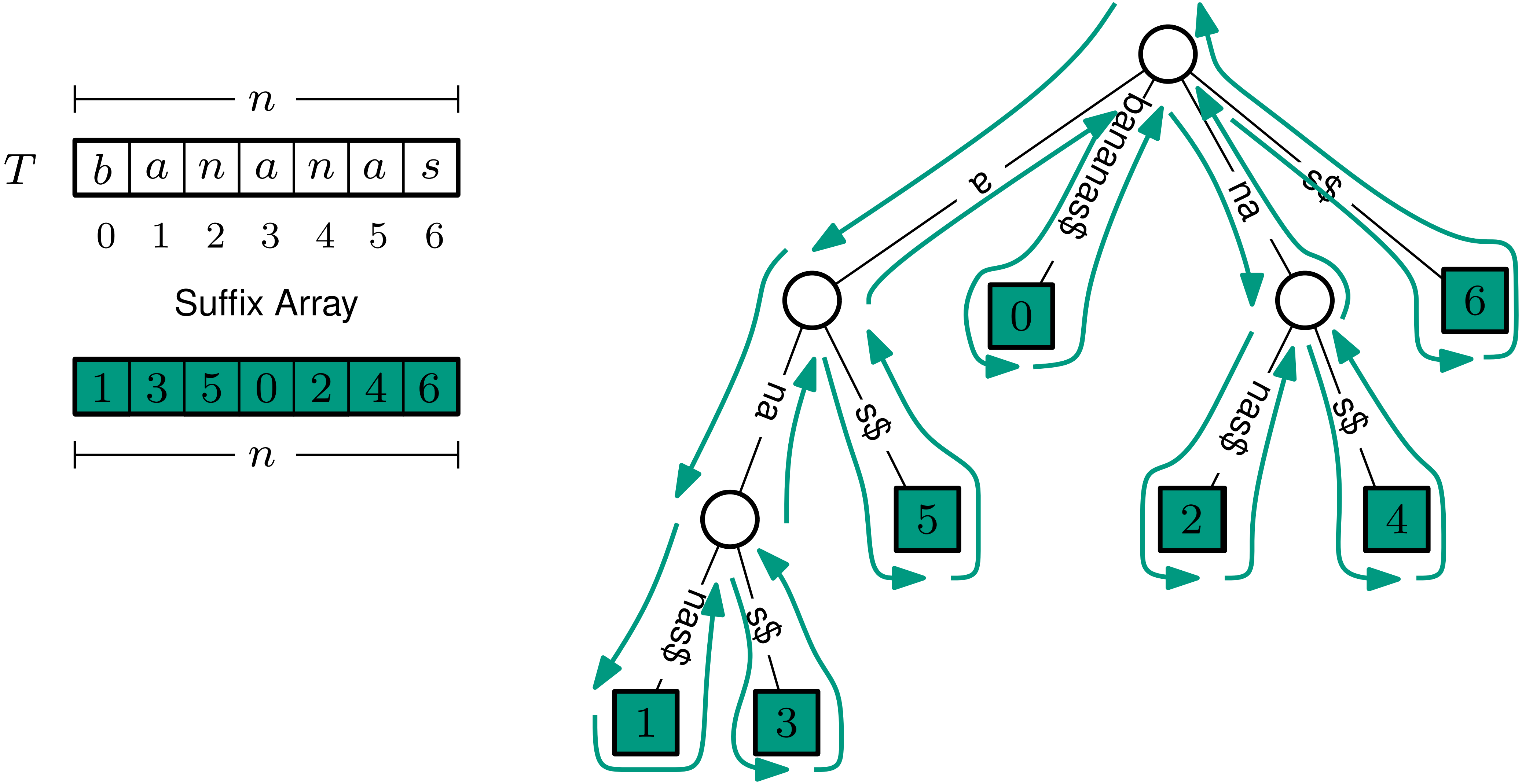


*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree

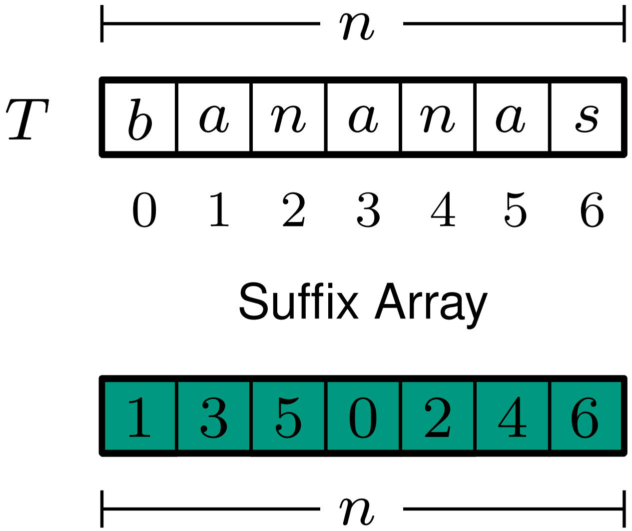


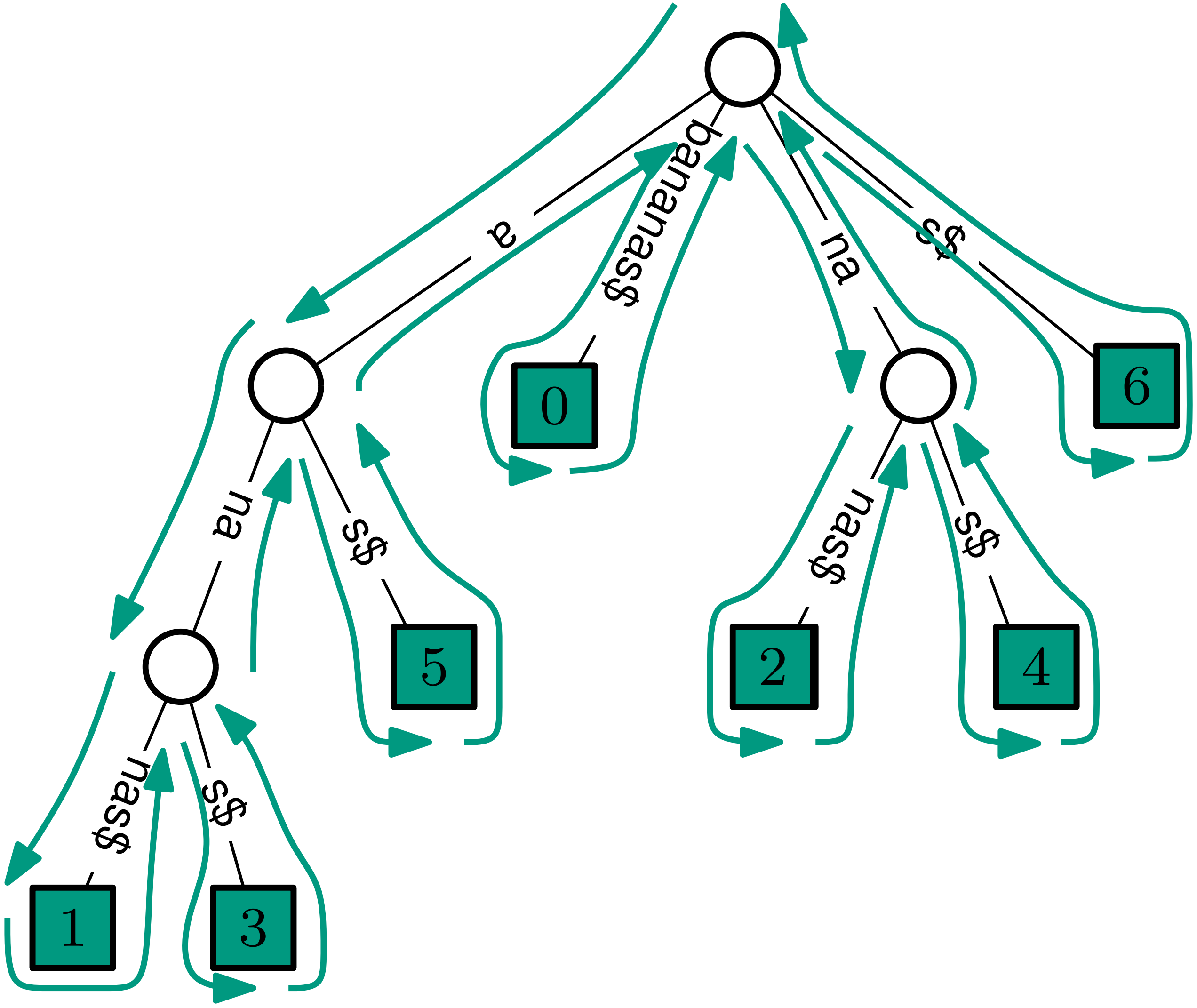
*recall that we added a unique symbol $ to make sure the tree exists*

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

# Constructing the Suffix Array from the Suffix Tree





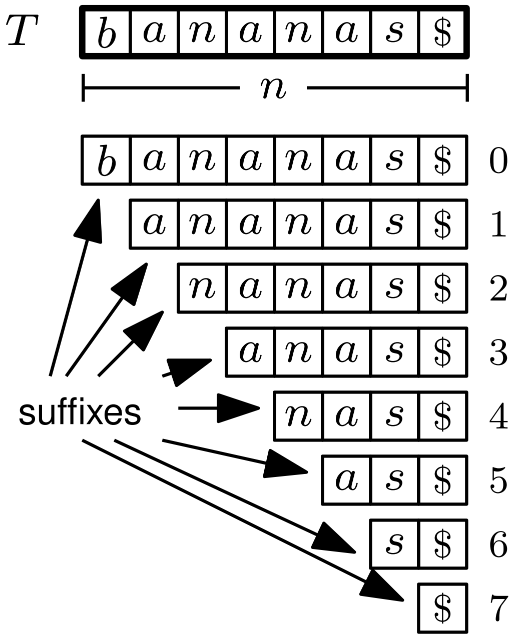
*recall that we added a unique symbol $ to make sure the tree exists*

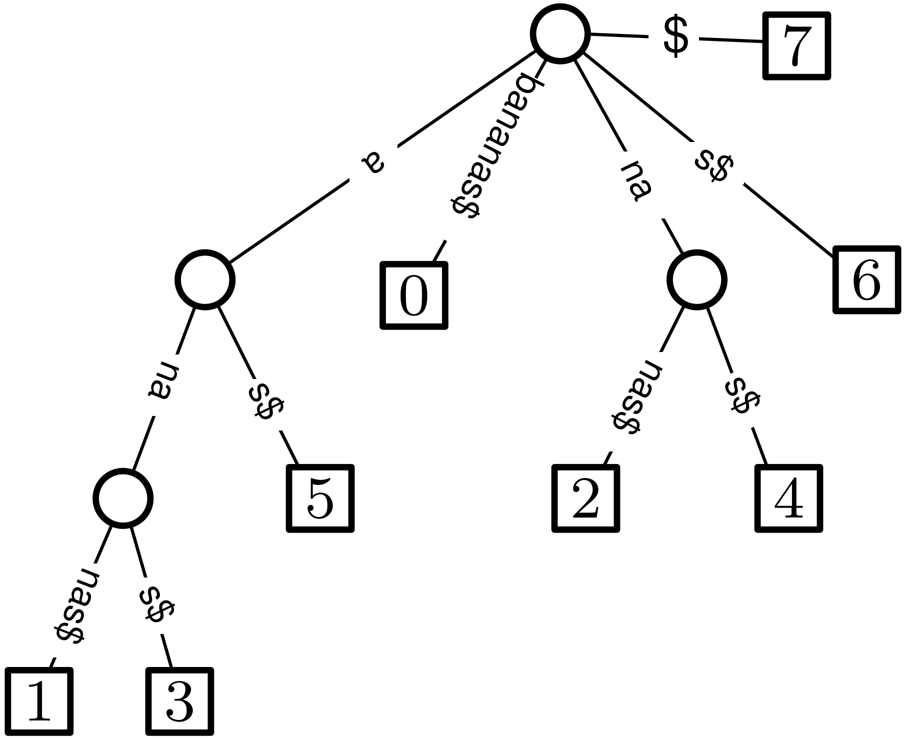
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

*this takes $O(n)$ time*

# Suffix tree summary





- The (compacted) suffix tree of a (length $n$) text uses $O(n)$ space

- Finding all matches of a pattern $P$ of length $m$ takes $O(m + \text{occ})$

  *where occ is the number of matches*

- Suffix trees can be built in $O(n)$ time

  *but we have only seen the $O(n^2)$ time method*

*we assumed that the alphabet contains a constant number of symbols*