Suffix Tries

Ben Langmead

Suffix trie

Build a **trie** containing all **suffixes** of a text *T*

```
T: GTTATAGCTGATCGCGGCGTAGCGG$
   GTTATAGCTGATCGCGGCGTAGCGG$
    TTATAGCTGATCGCGGCGTAGCGG$
     TATAGCTGATCGCGGCGTAGCGG$
      ATAGCTGATCGCGGCGTAGCGG$
       TAGCTGATCGCGGCGTAGCGG$
        AGCTGATCGCGGCGTAGCGG$
         GCTGATCGCGGCGTAGCGG$
          CTGATCGCGGCGTAGCGG$
           TGATCGCGGCGTAGCGG$
            GATCGCGGCGTAGCGG$
             ATCGCGGCGTAGCGG$
              TCGCGGCGTAGCGG$
               CGCGGCGTAGCGG$
                GCGGCGTAGCGG$
                 CGGCGTAGCGG$
                  GGCGTAGCGG$
                   GCGTAGCGG$
                    CGTAGCGG$
                     GTAGCGG$
                      TAGCGG$
                       AGCGG$
                        GCGG$
                         CGG$
                          GG$
                           G$
                            $
```

m(m+1)/2 chars

Suffix trie

First add special *terminal character* **$** to the end of *T*

$ is a character that does not appear elsewhere in *T*, and we define it to be less than other characters (**$** < **A** < **C** < **G** < **T**)

$ enforces a familiar rule: e.g. "as" comes before "ash" in the dictionary.
$ also guarantees no suffix is a prefix of any other suffix.

```
T: G T T A T A G C T G A T C G C G G C G T A G C G G $
   G T T A T A G C T G A T C G C G G C G T A G C G G $
     T T A T A G C T G A T C G C G G C G T A G C G G $
       T A T A G C T G A T C G C G G C G T A G C G G $
         A T A G C T G A T C G C G G C G T A G C G G $
           T A G C T G A T C G C G G C G T A G C G G $
             A G C T G A T C G C G G C G T A G C G G $
               G C T G A T C G C G G C G T A G C G G $
                 C T G A T C G C G G C G T A G C G G $
                   T G A T C G C G G C G T A G C G G $
                     G A T C G C G G C G T A G C G G $
                       A T C G C G G C G T A G C G G $
                         T C G C G G C G T A G C G G $
                           C G C G G C G T A G C G G $
                             G C G G C G T A G C G G $
                               C G G C G T A G C G G $
```

Suffix trie

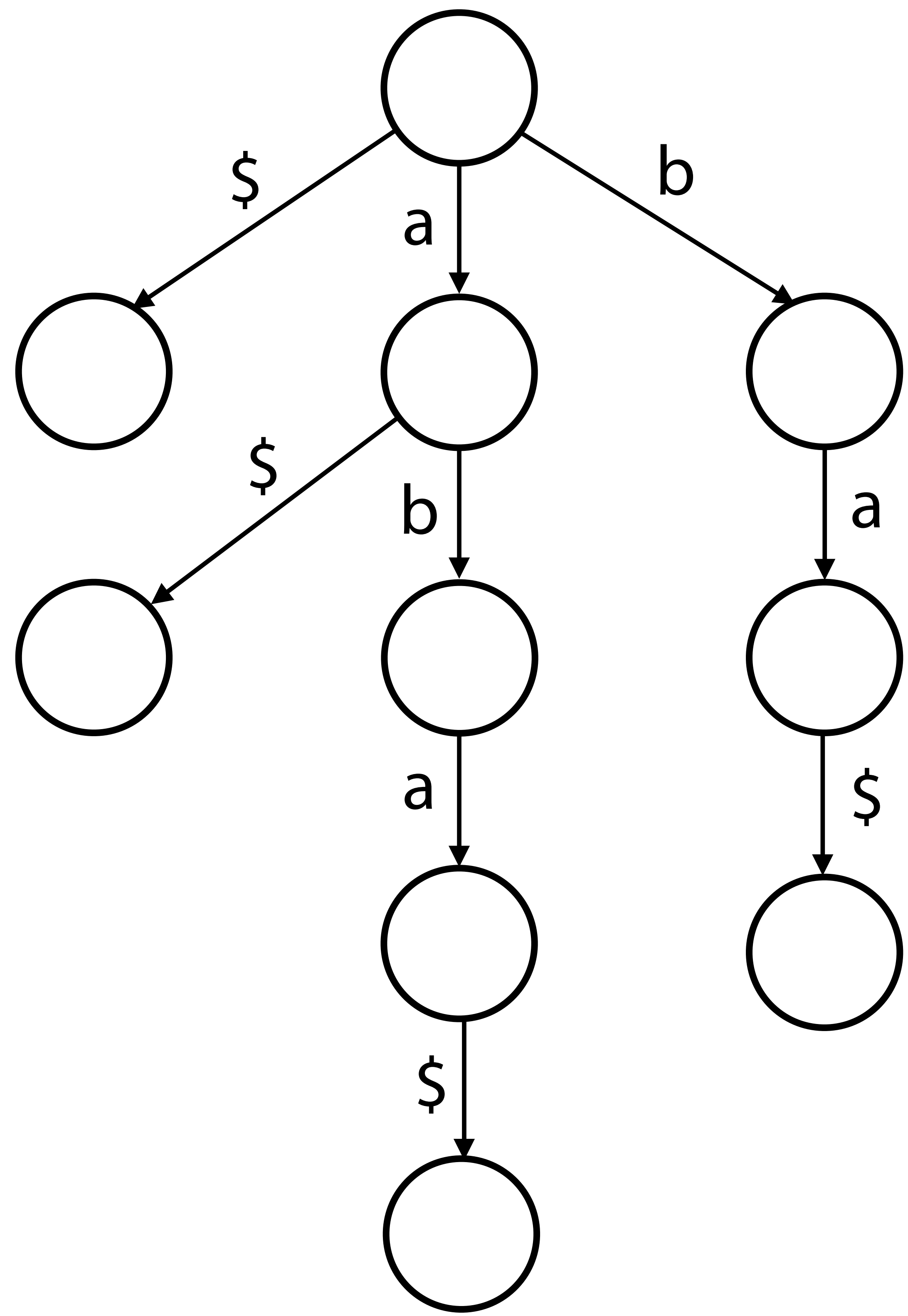
T: aba**$**

Suffix trie:

Suffix trie

T: aba**$**

Suffix trie:

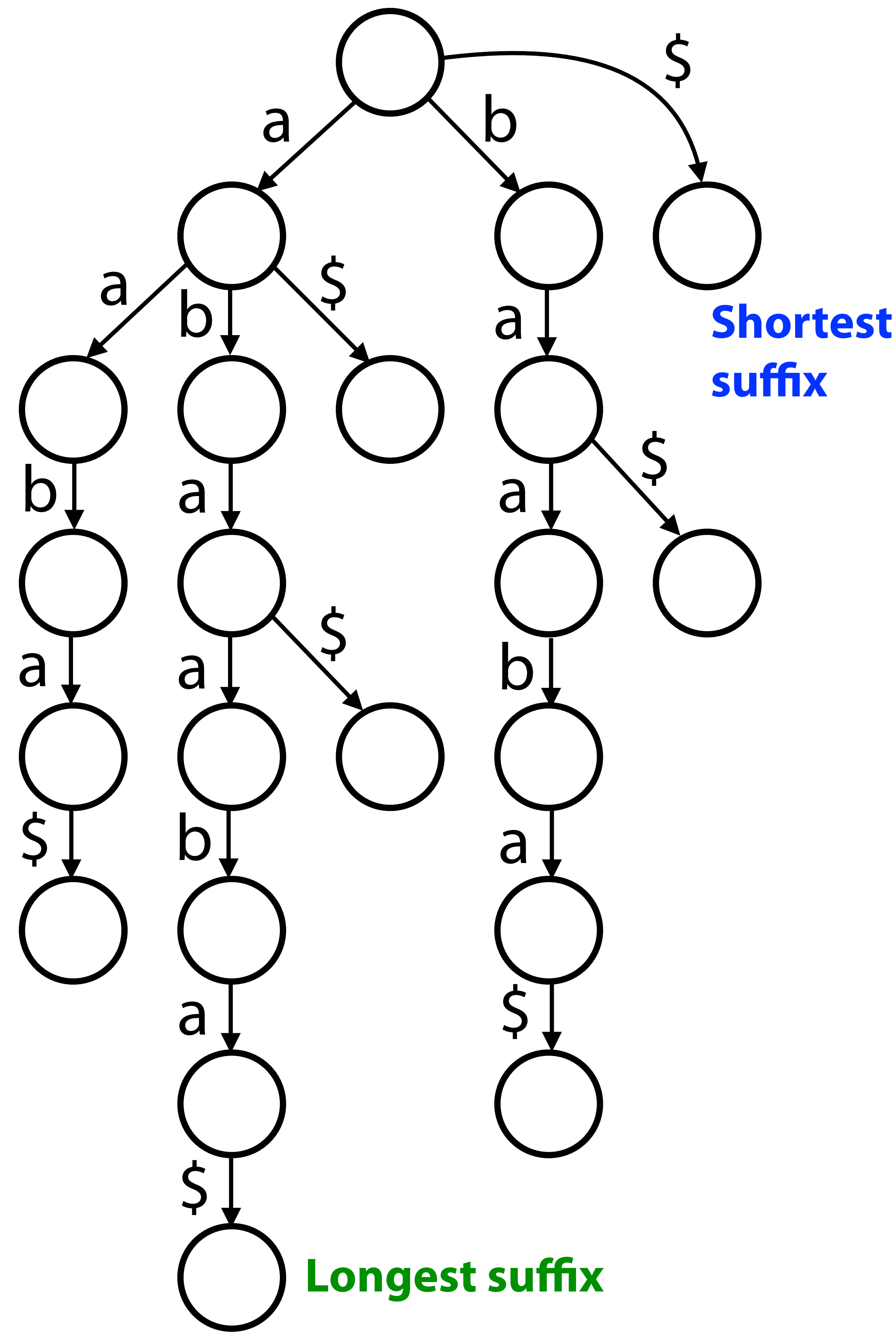


Suffix trie

T: abaaba**$**

Each path from root to leaf represents a suffix; each suffix is represented by some path from root to leaf

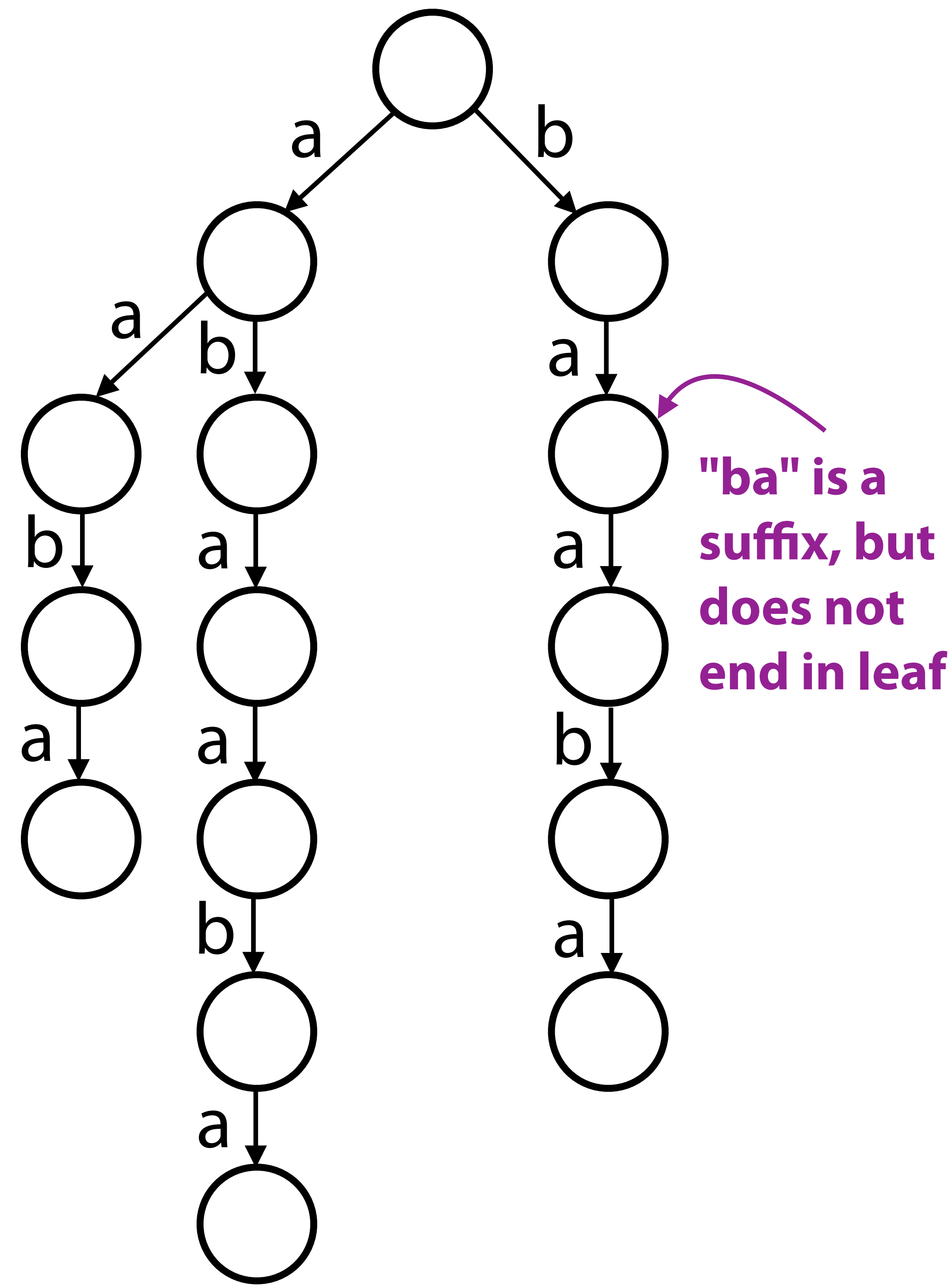
Would this still be the case if we hadn't added **$**?



Suffix trie

T: abaaba

no $



Each path from root to leaf represents a suffix; each suffix is represented by some path from root to leaf

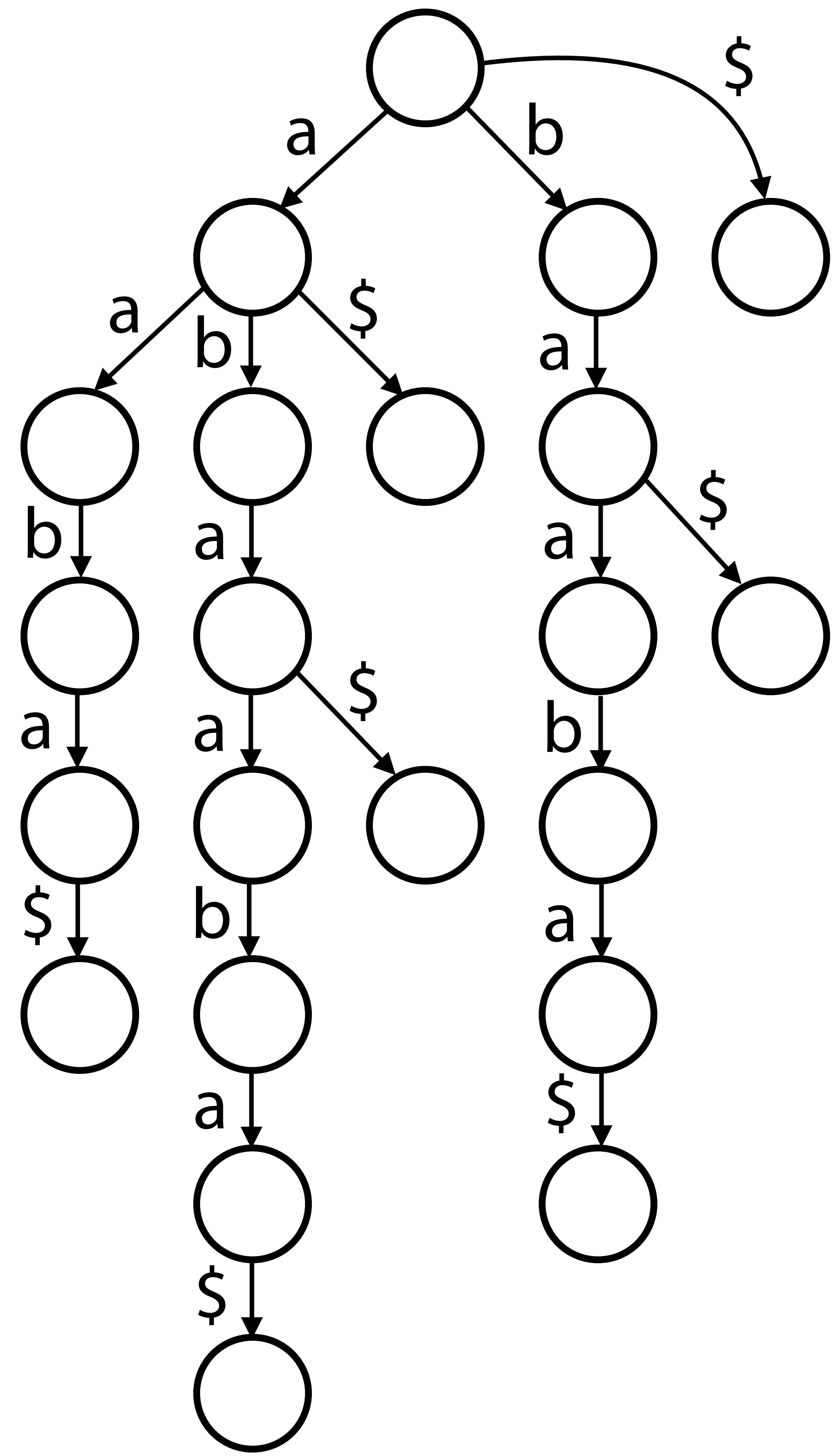
Would this still be the case if we hadn't added **$**? **No**

Suffix trie

T: abaaba**$**

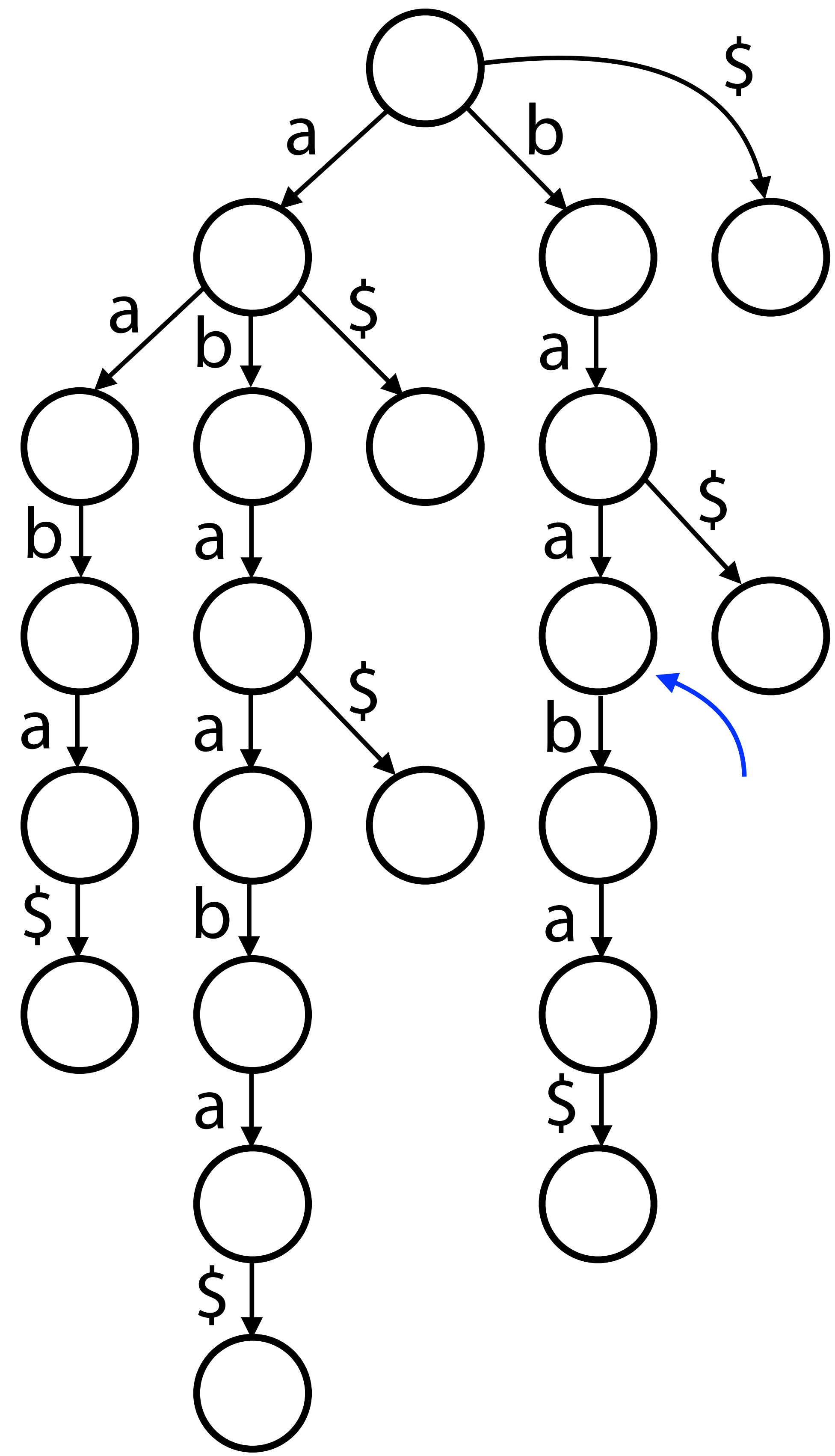
We **need** the **$** for this property:

Each path from root to leaf represents a suffix; each suffix is represented by some path from root to leaf



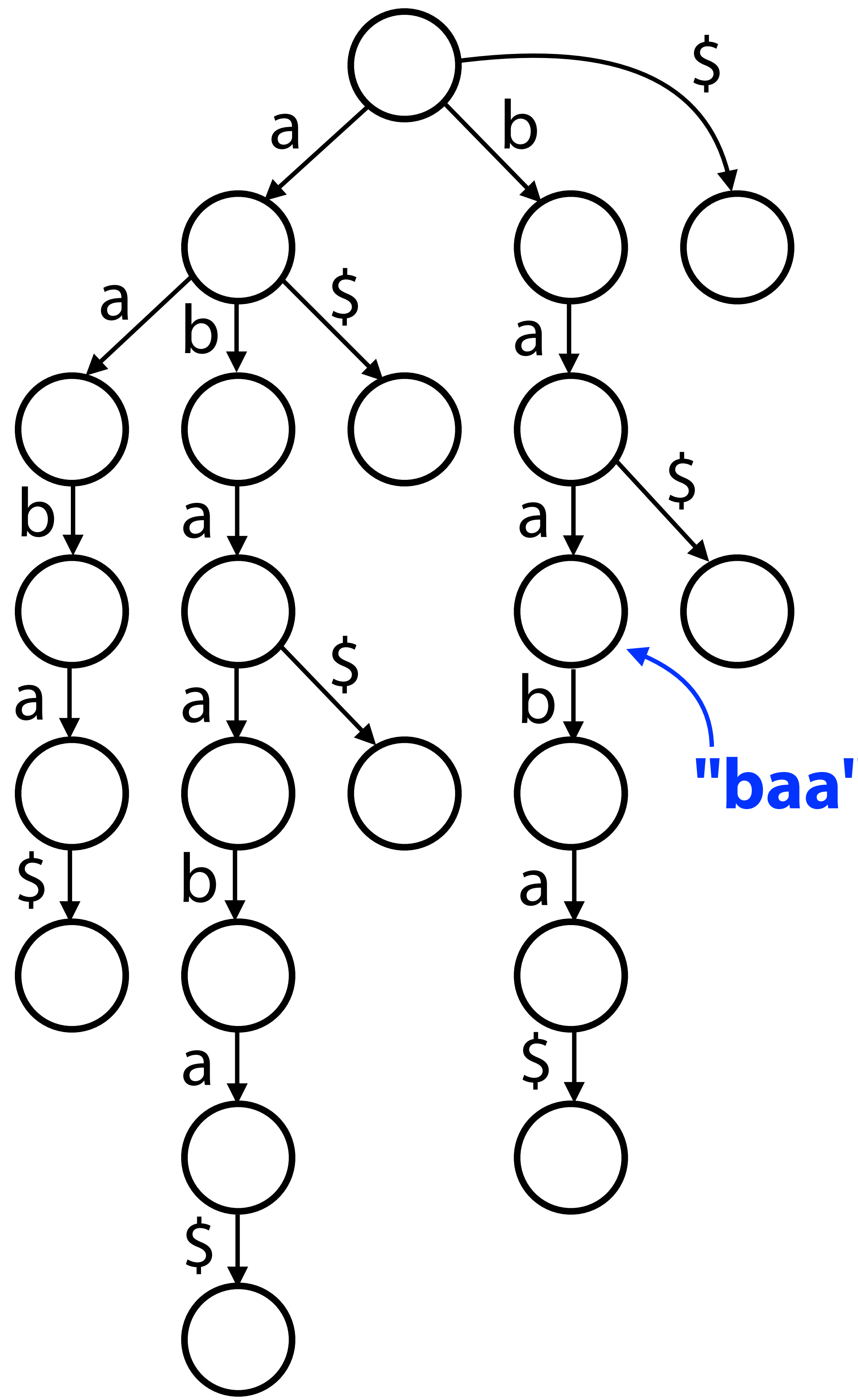
Suffix trie

Think of each node as having a **label**, spelling out characters on path from root to node



Suffix trie

Think of each node as having a **label**, spelling out characters on path from root to node



Suffix trie

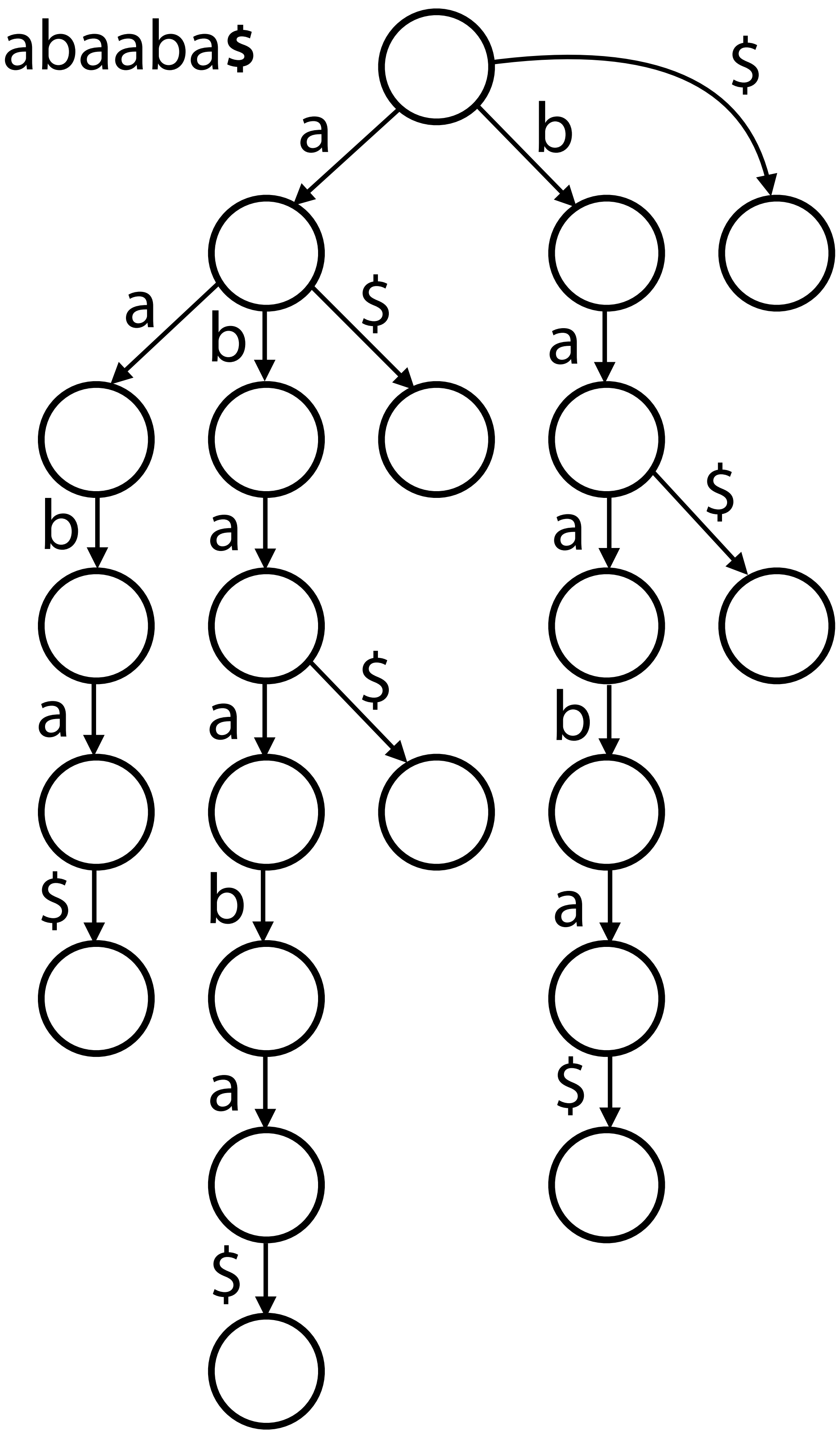
How do we check whether a string *S* is a substring of *T*?

A *substring* is a *prefix* of a *suffix*

T:

Each of *T*'s substrings is a prefix of a suffix, and so is spelled out along a path from the root.

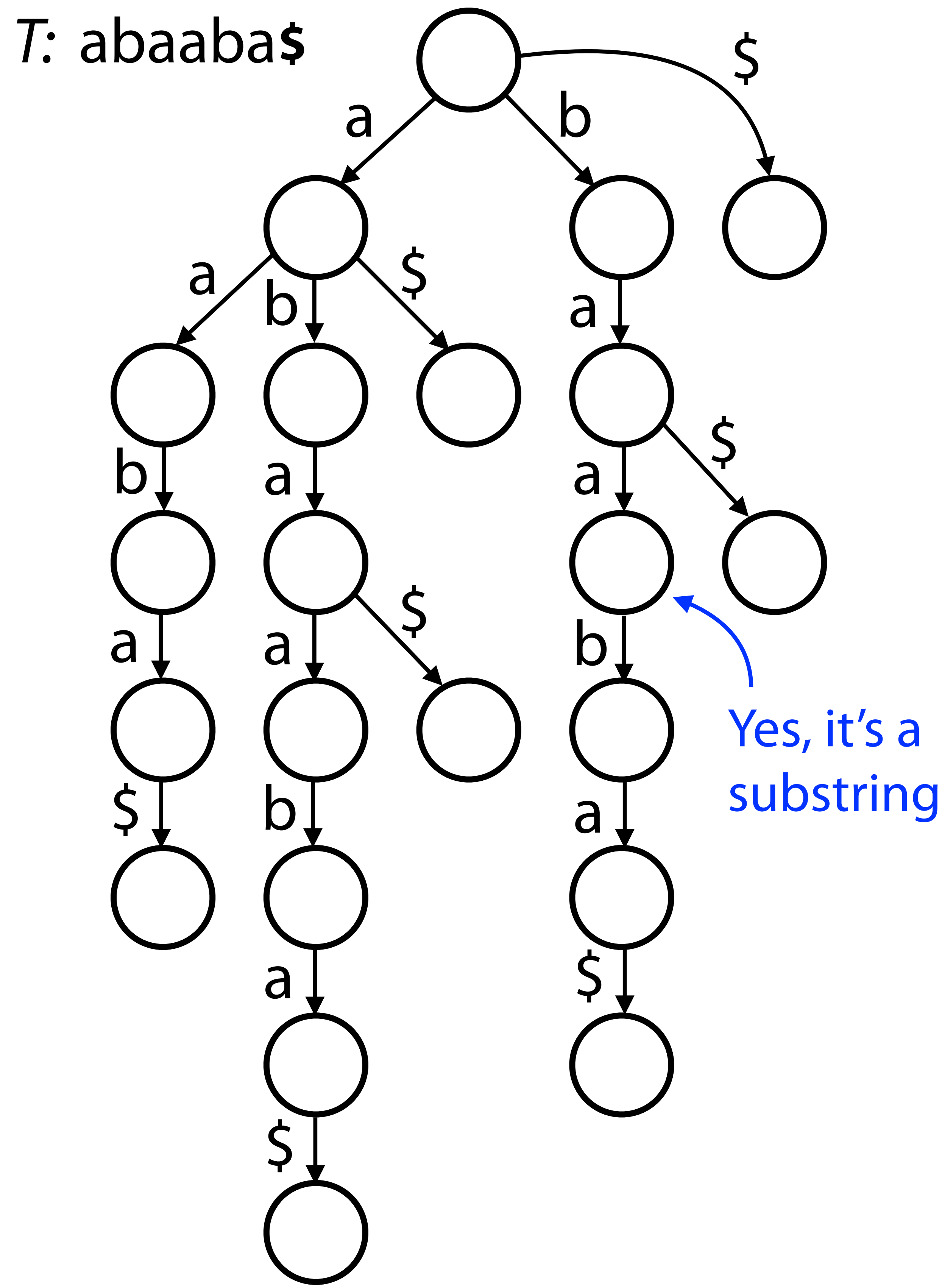
T: abaaba**$**



Suffix trie

How do we check whether a string *S* is a substring of *T*?

S = baa



Suffix trie

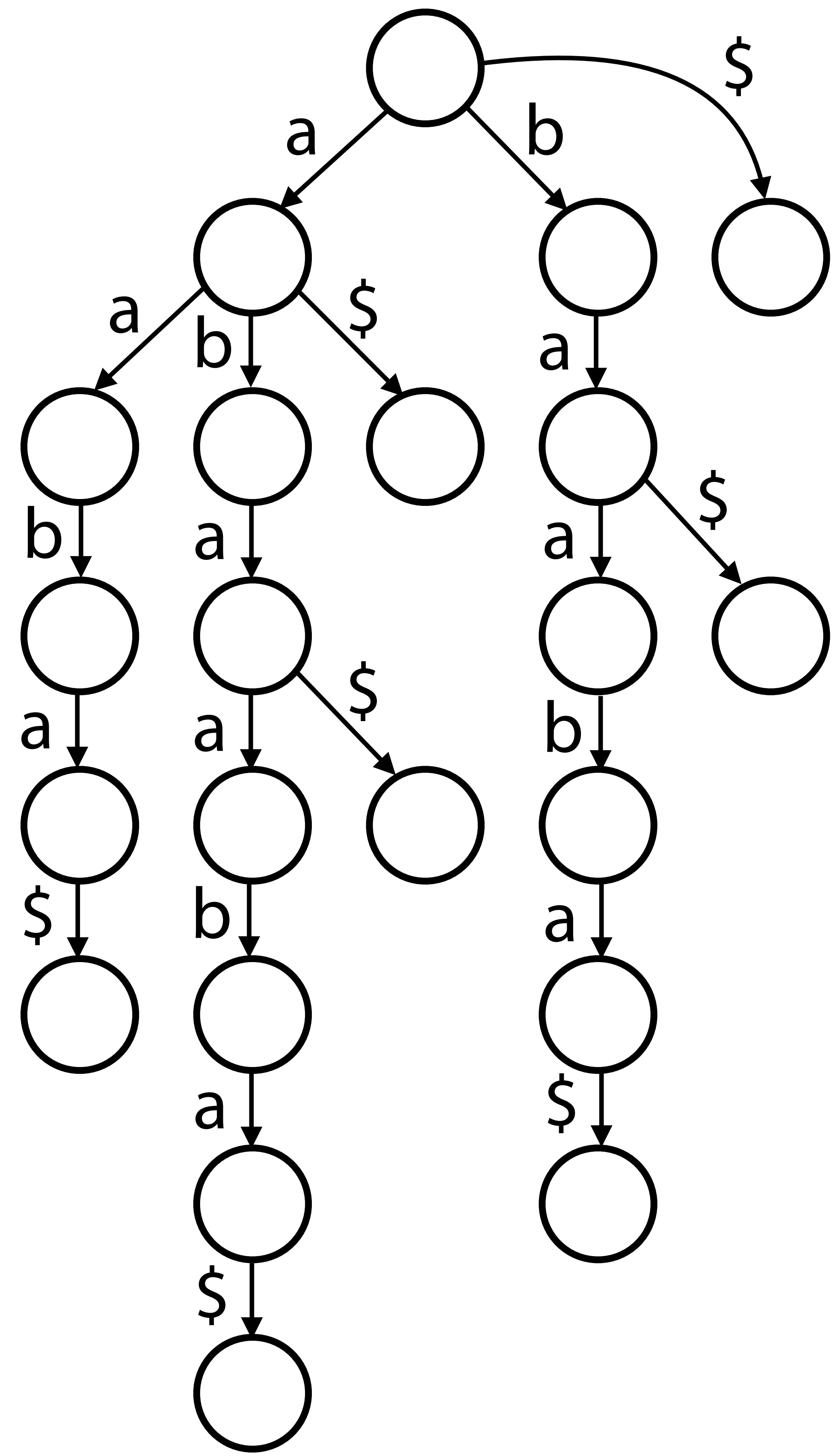
To check whether a string *S* is a substring of *T*:

Start at root and follow edges labeled with the characters of *S*

If we "fall off," *S* is not a substring

If we exhaust *S* without falling off, *S* is a substring of *T*

Reasonable to assume $O(n)$ time where $|S| = n$



Suffix trie

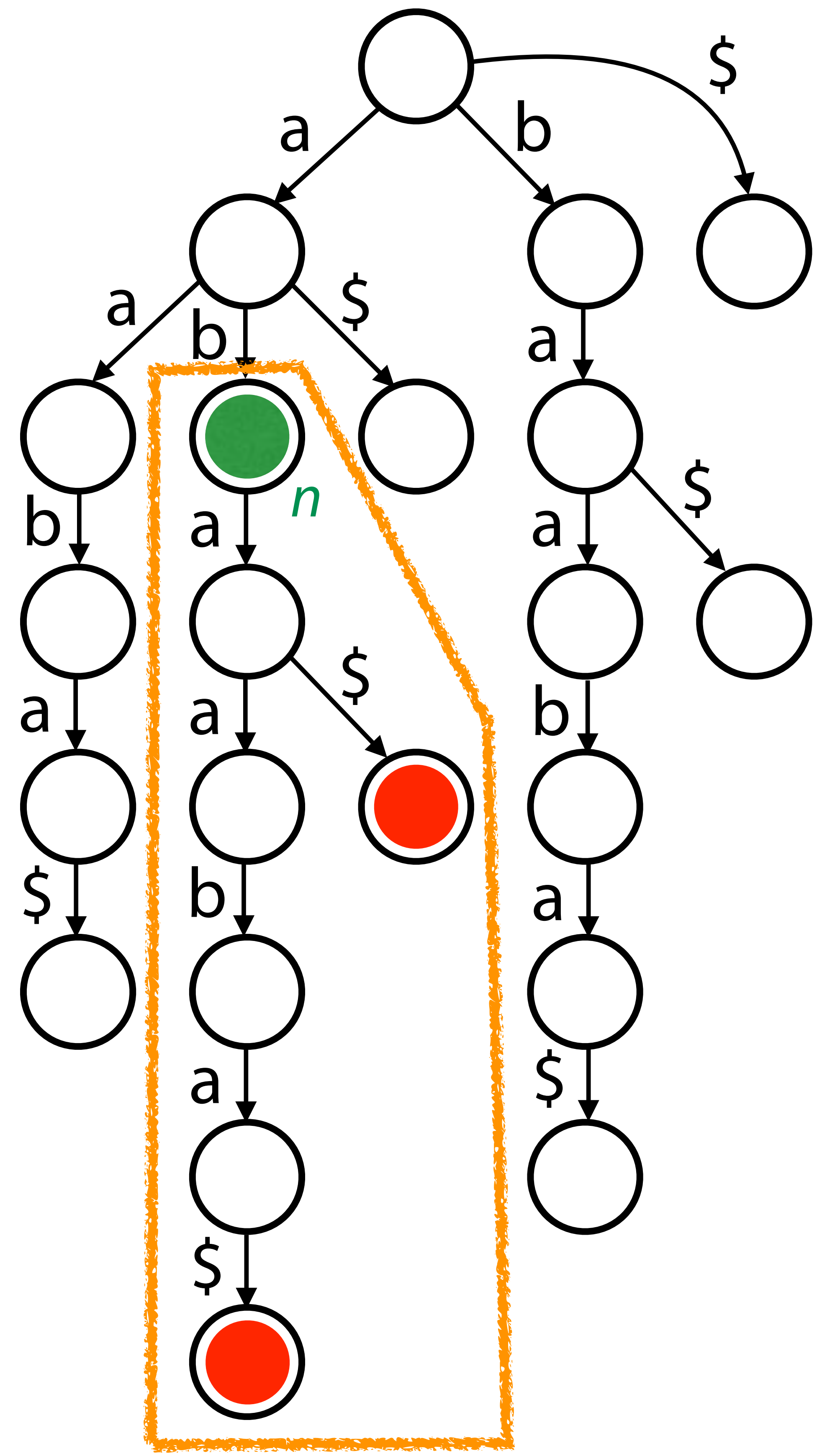
How do we count the **number of times** a string *S* occurs as a substring of *T*?

Say *S* = ab

Let *n* be the **node** we reach after "walking down" according to S

The **subtree** rooted at *n* holds suffixes for which *S* is a prefix

2 leaves in the subtree, so 2 suffixes for which S is a prefix = 2 occurrences!



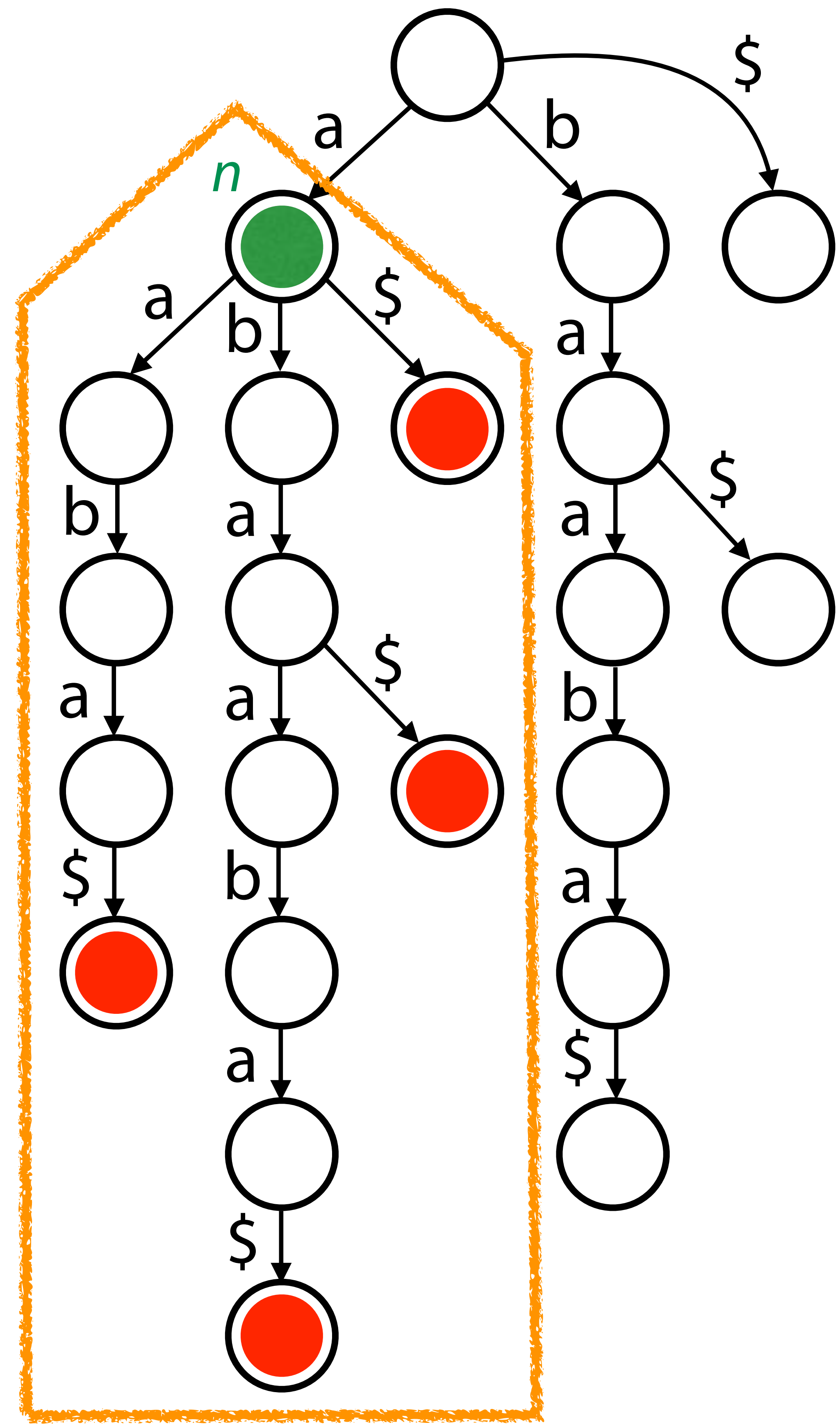
Suffix trie

How do we count the **number of times** a string *S* occurs as a substring of *T*?

Walk down according to *S*. If we fall off, answer is 0.

Otherwise, if we ended at node ***n***, answer = **# of leaves in subtree rooted at *n***.

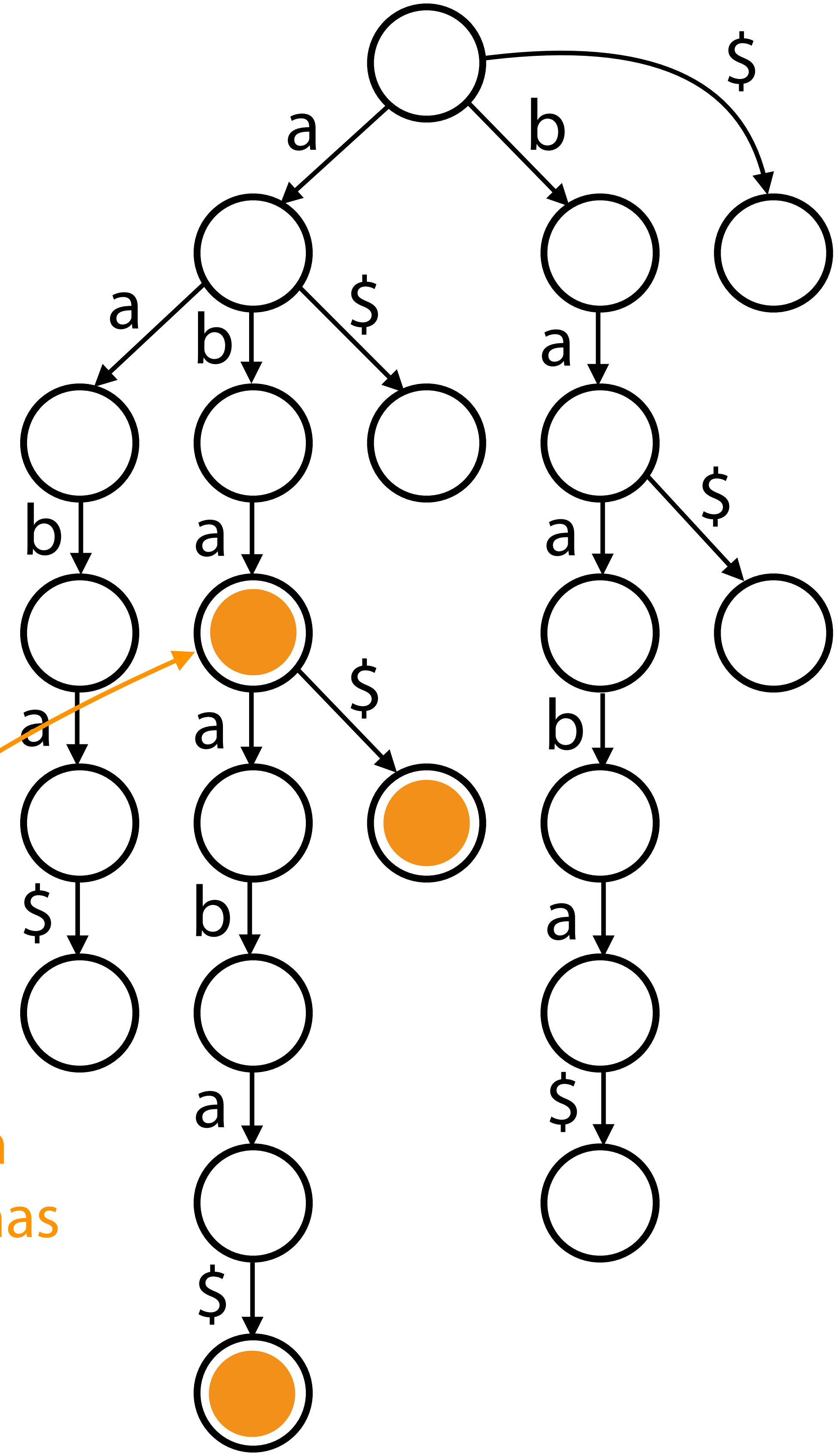
Leaves can be counted with depth-first traversal.



Suffix trie

How do we find the **longest repeated substring** of *T*?

Find the **deepest node with more than one child**



Deepest node with multiple children has label "aba"