

Suffix Trees

Ben Langmead



JOHNS HOPKINS

WHITING SCHOOL
of ENGINEERING

Department of Computer Science



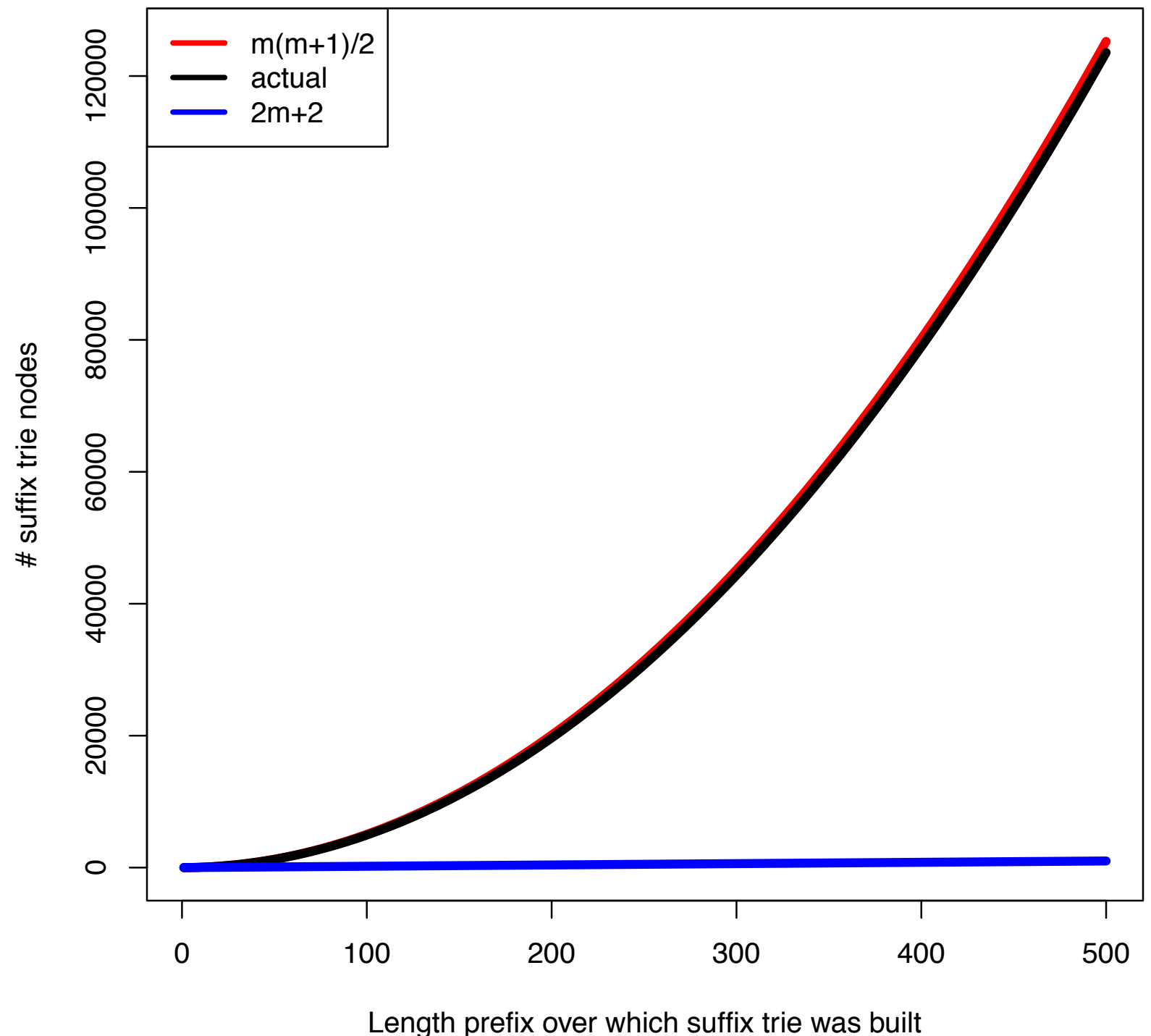
Please sign guestbook (www.langmead-lab.org/teaching-materials) to tell me briefly how you are using the slides. For original Keynote files, email me (ben.langmead@gmail.com).

Suffix trie

We saw the suffix trie, but we also saw its size grows *quadratically* with the length of the string

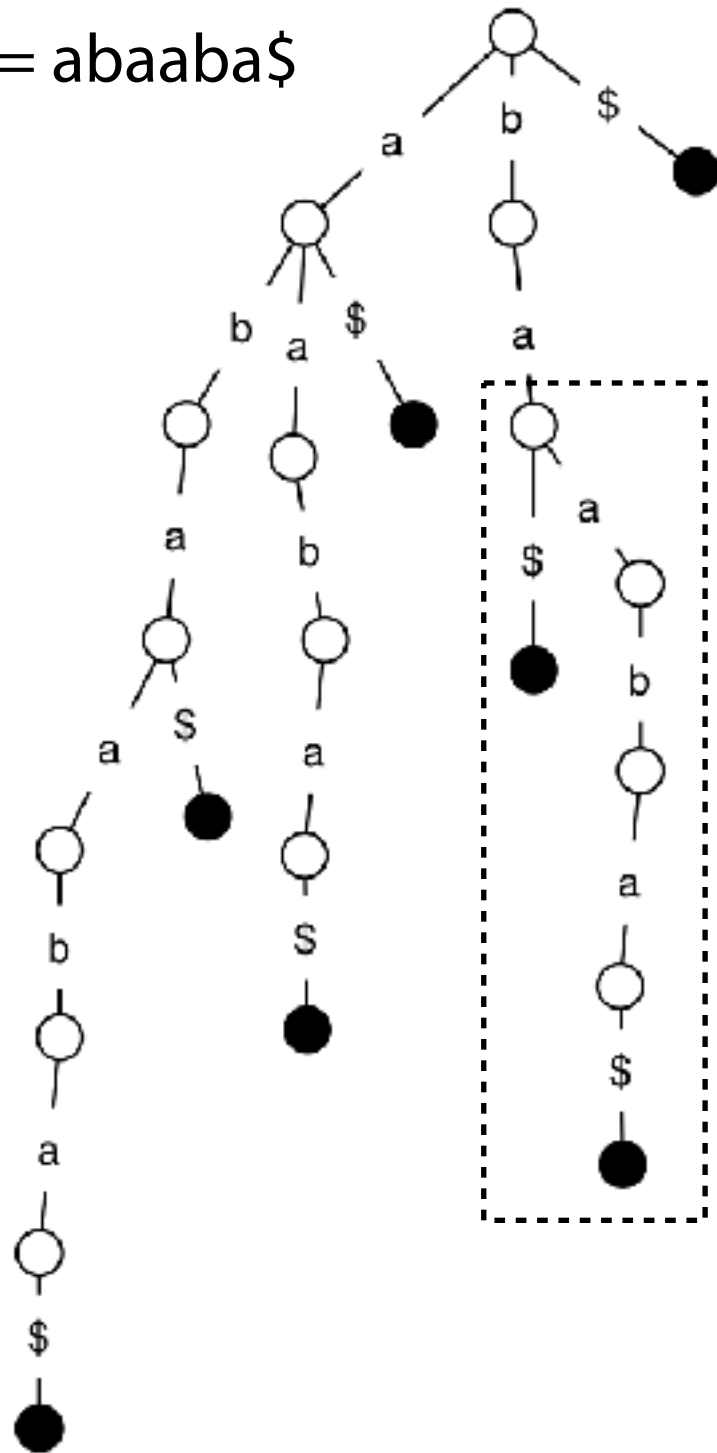
Human genome is $3 \cdot 10^9$ bases long.

If $m = 3 \cdot 10^9$, m^2 is way huge, far beyond what we can store in memory



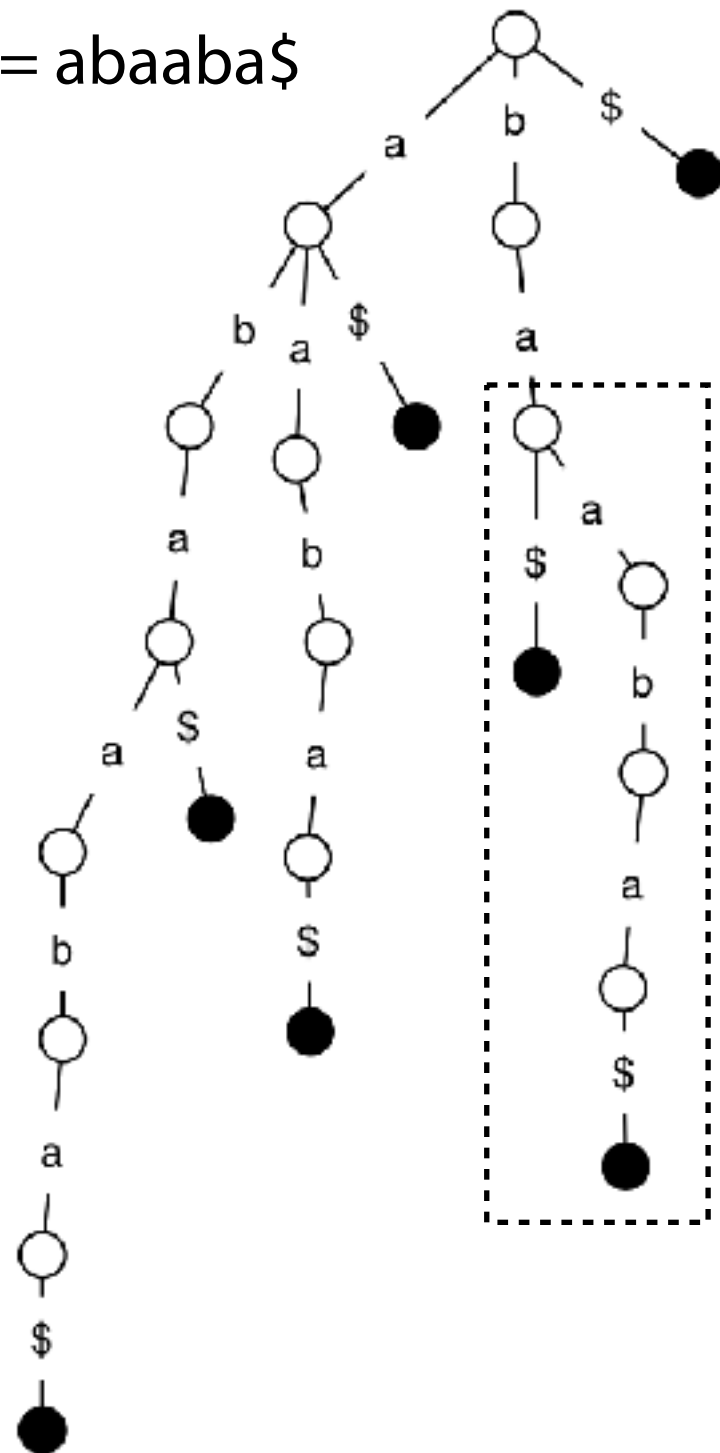
Suffix trie: making it smaller

$T = \text{abaaba\$}$

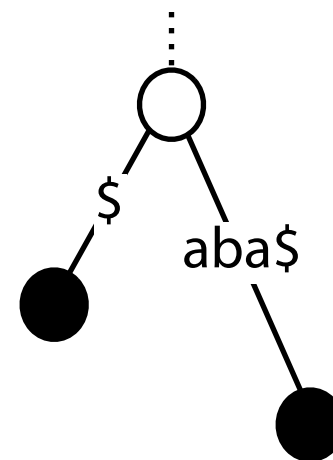


Suffix trie: making it smaller

$T = \text{abaaba}\$$



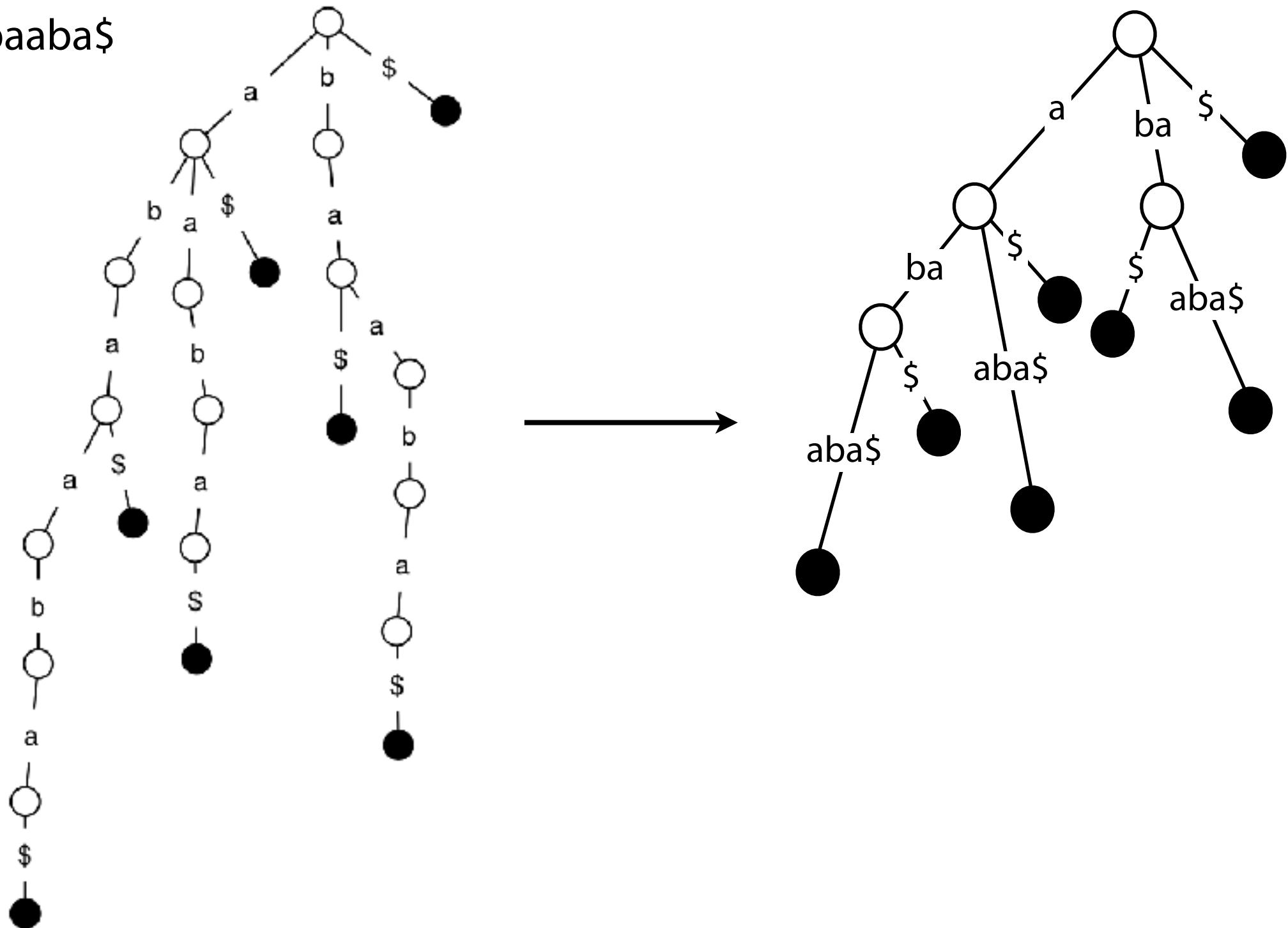
Idea 1: Coalesce non-branching paths into a *single edge* with a *string* label



Reduces # nodes, edges,
guarantees non-leaf nodes have >1 child

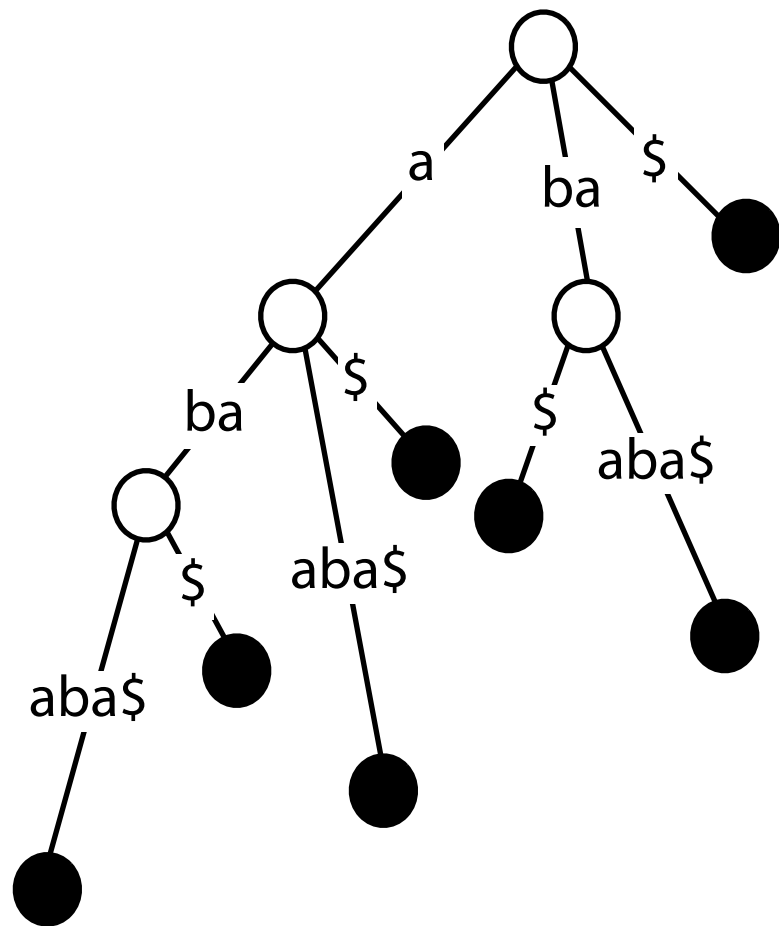
Suffix trie: making it smaller

$T = \text{abaaba\$}$



Suffix tree

$T = \text{abaaba\$}$ $|T| = m$



leaves?

m

non-leaf nodes (bound)? $\leq m - 1$

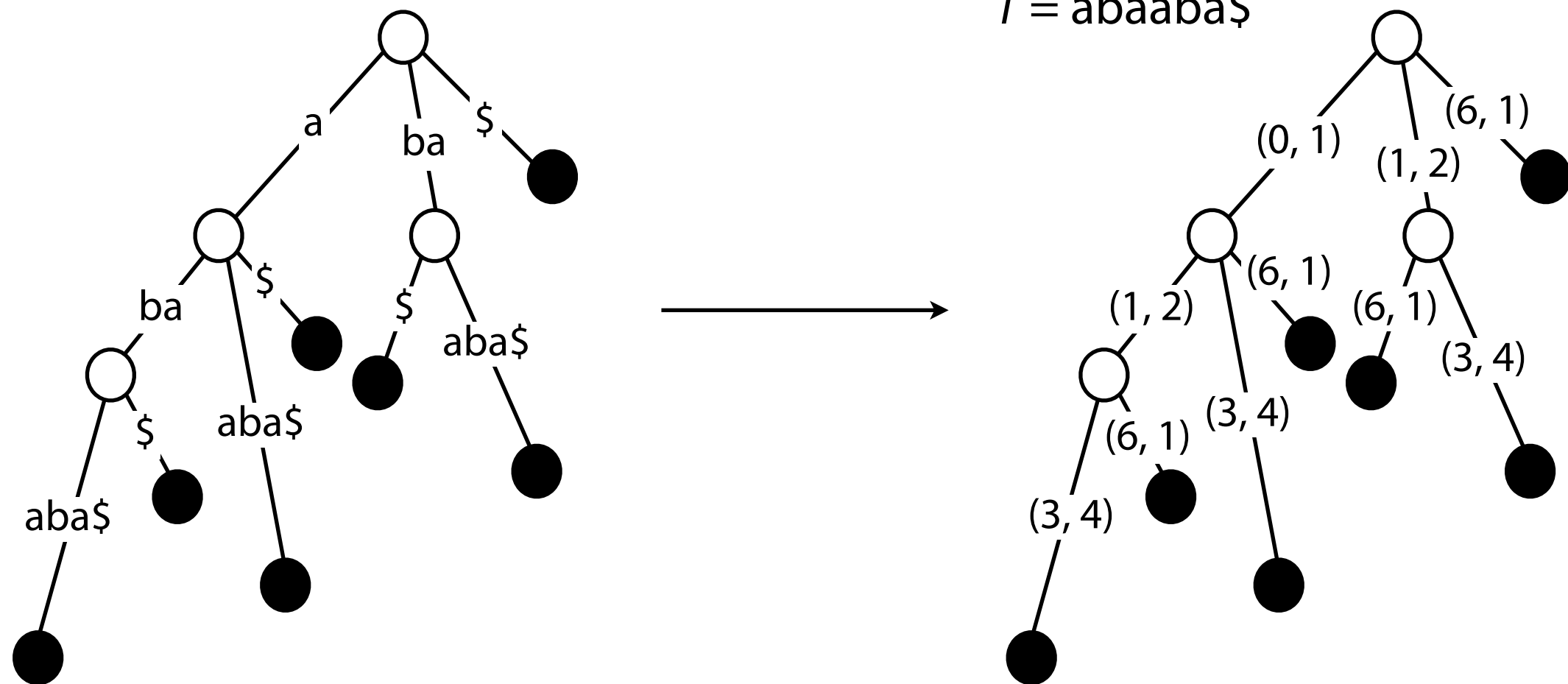
$\leq 2m - 1$ nodes total — $O(m)$

Is *total size* $O(m)$ now?

No: total length of edge labels grows with m^2

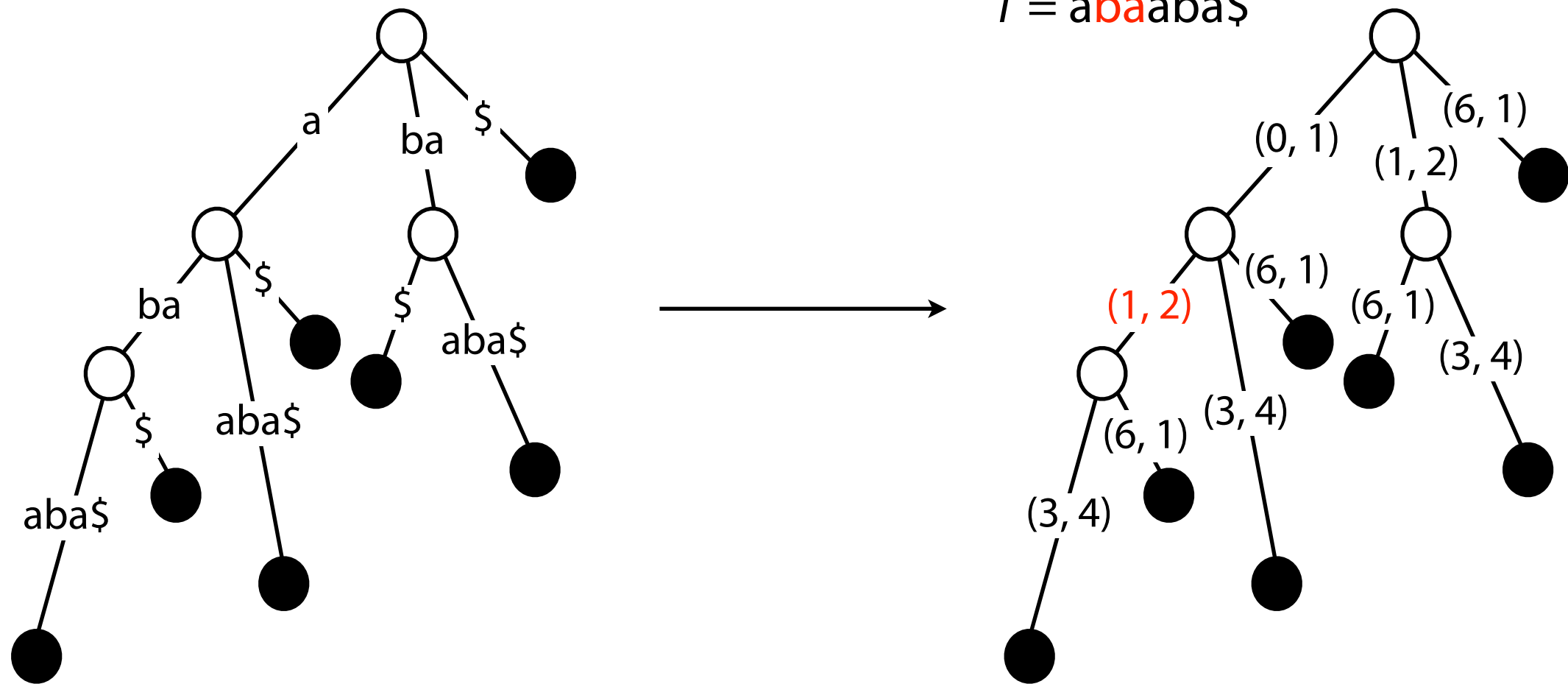
Suffix tree

Idea 2: Store T itself in addition to the tree. Convert tree's edge labels to (offset, length) pairs with respect to T .



Suffix tree

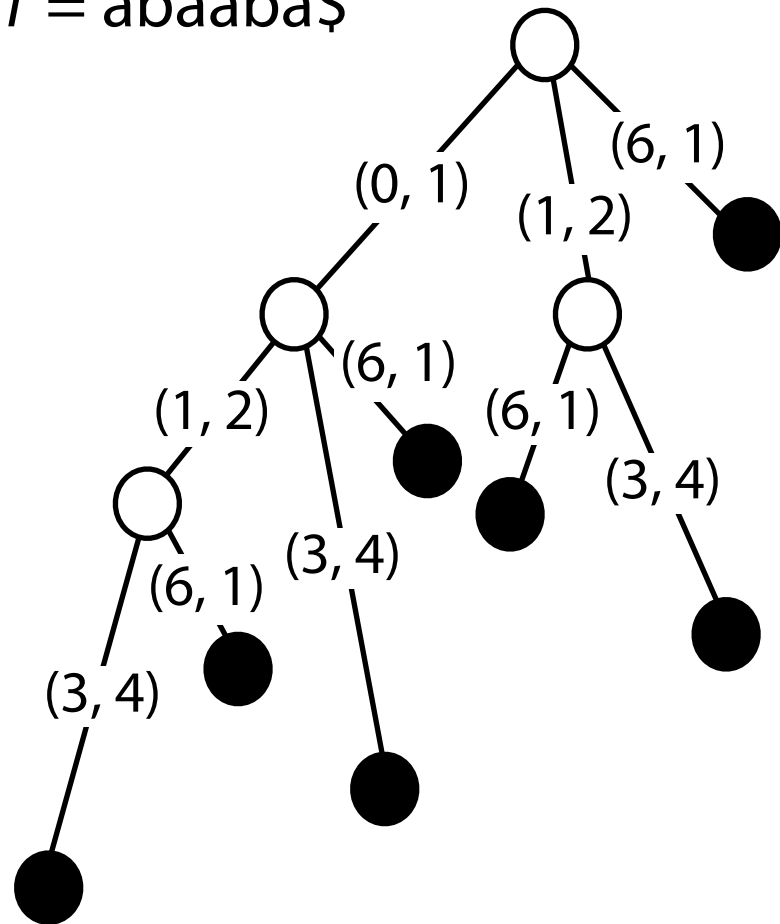
Idea 2: Store T itself in addition to the tree. Convert tree's edge labels to (offset, length) pairs with respect to T .



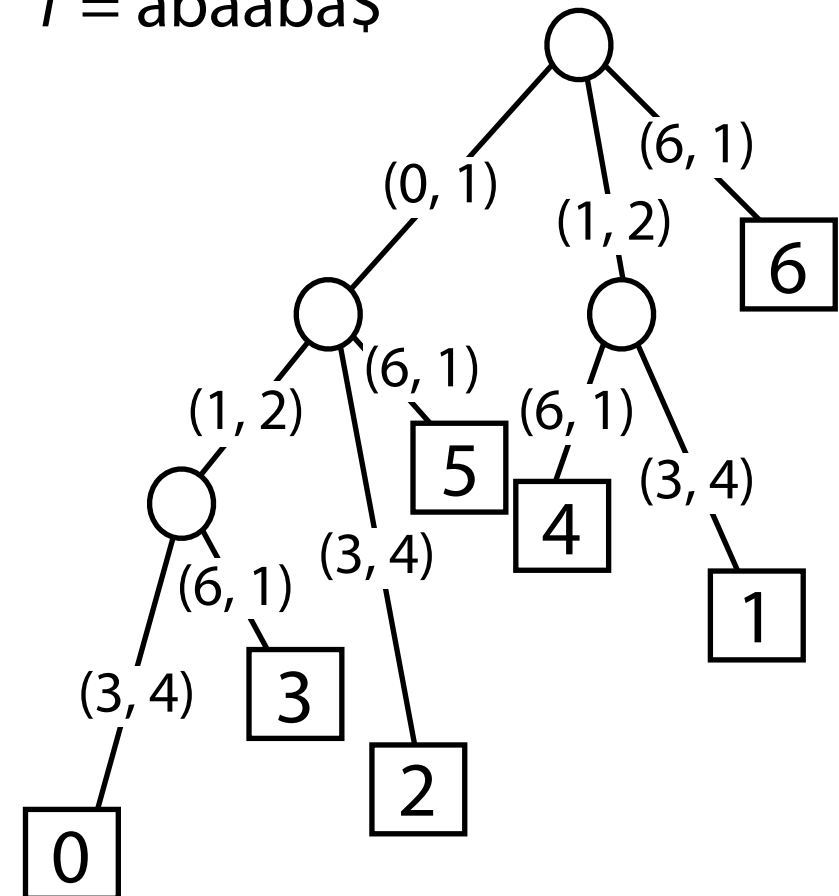
Space is now $O(m)$ Suffix trie was $O(m^2)$!

Suffix tree: leaves hold offsets

$T = \text{abaaba}\$$

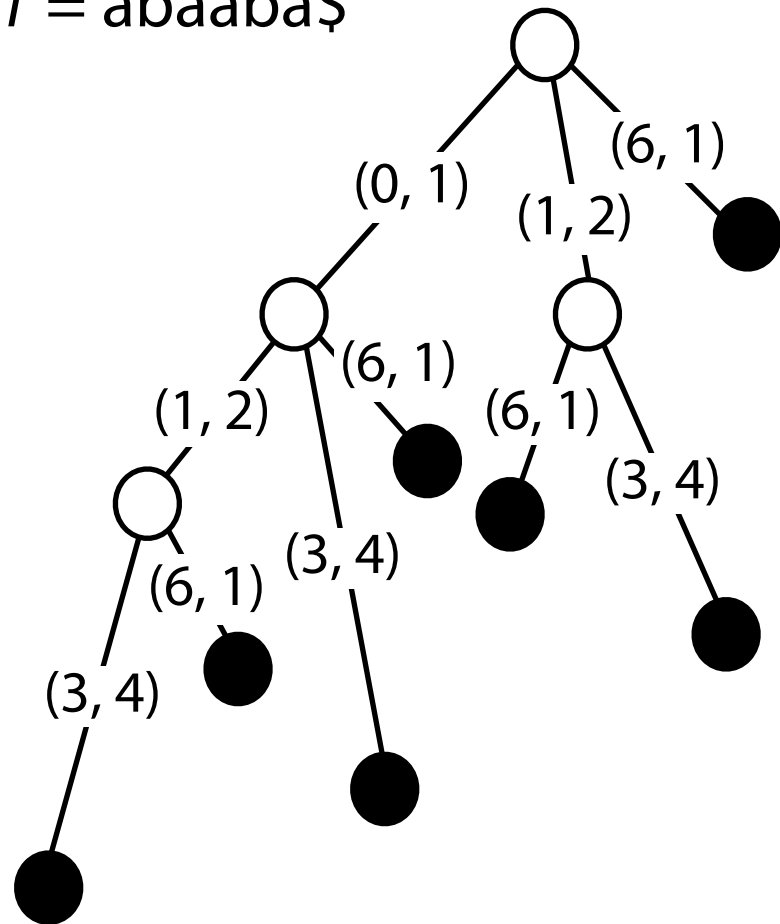


$T = \text{abaaba}\$$

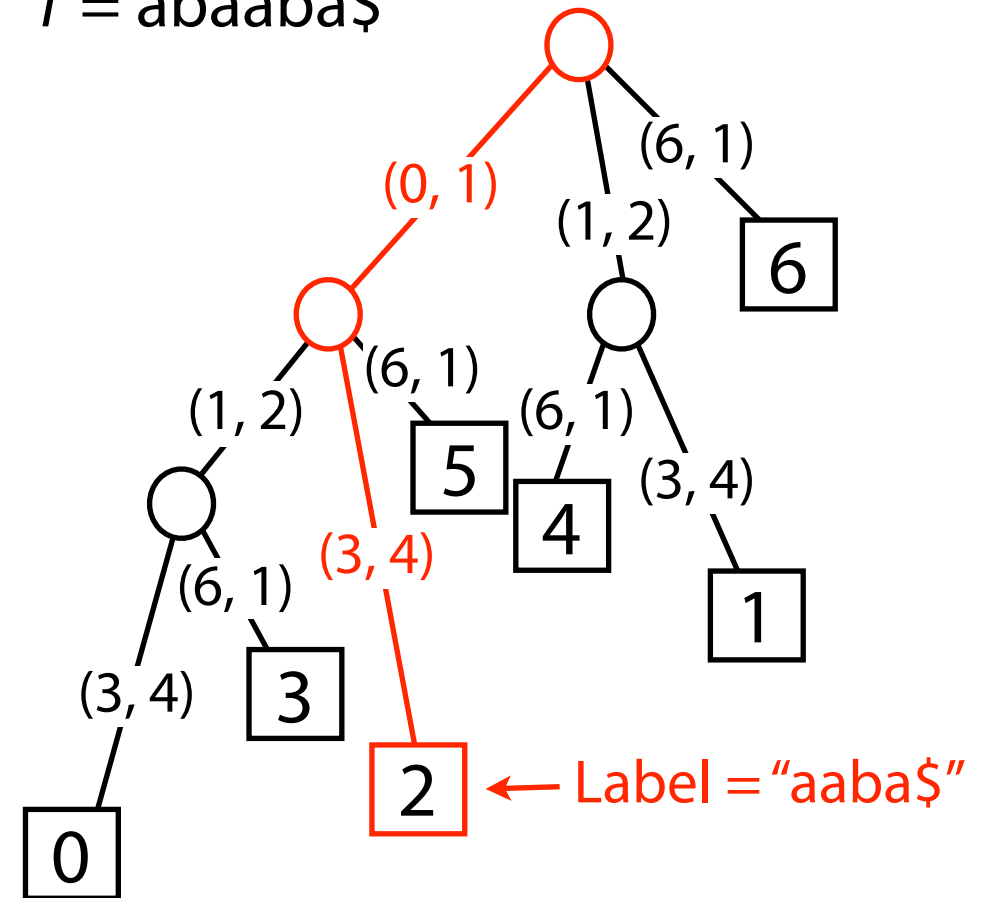


Suffix tree: leaves hold offsets

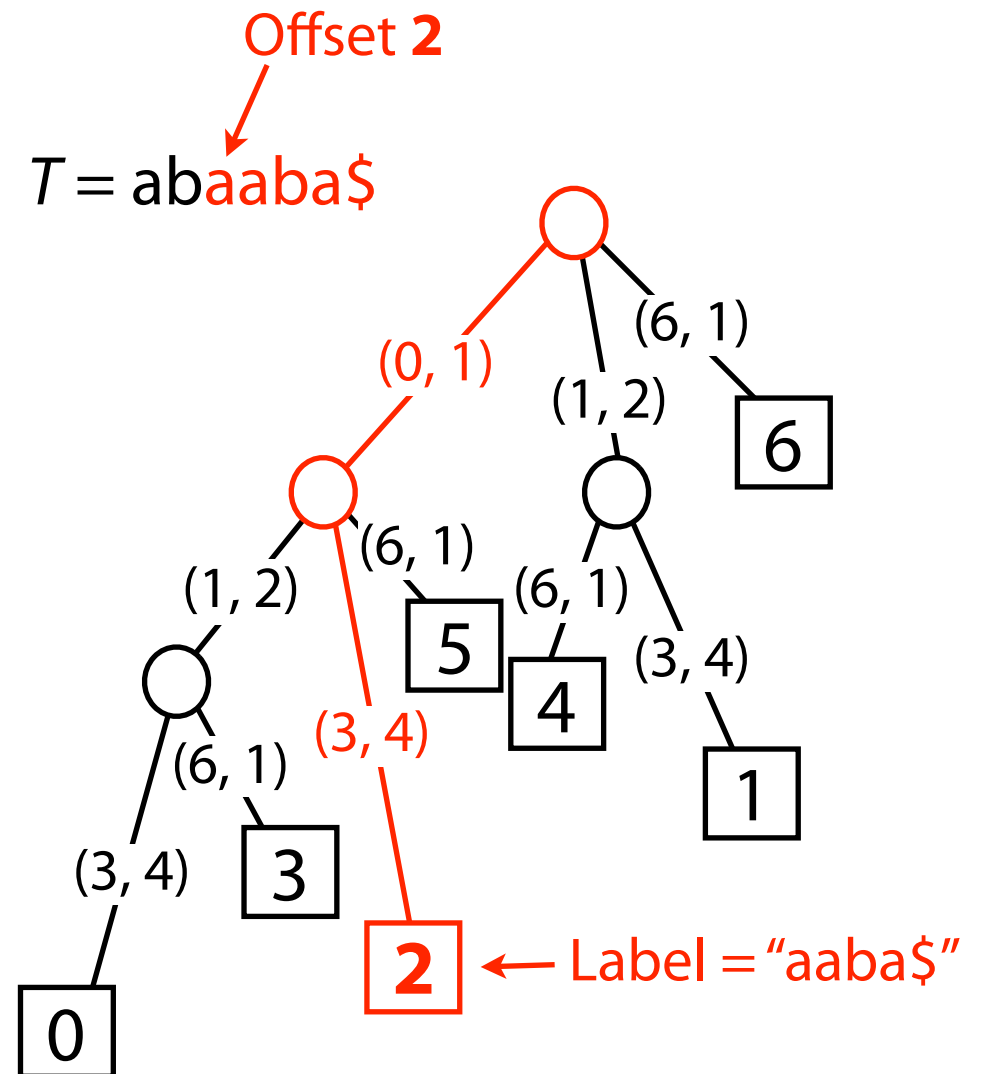
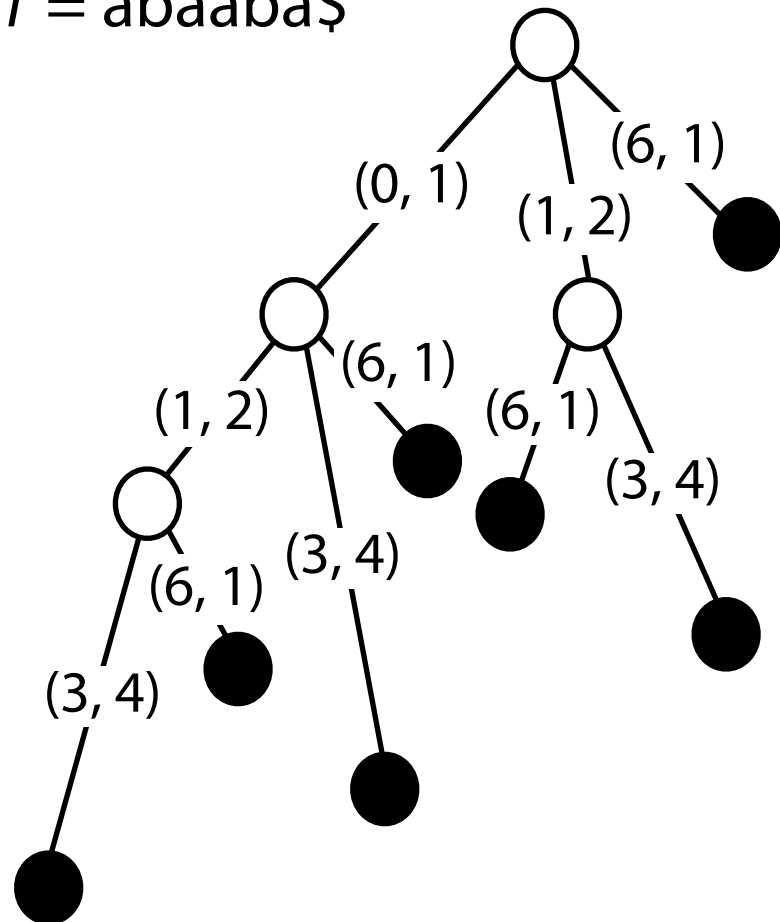
$T = \text{abaaba}\$$



$T = \text{abaaba}\$$

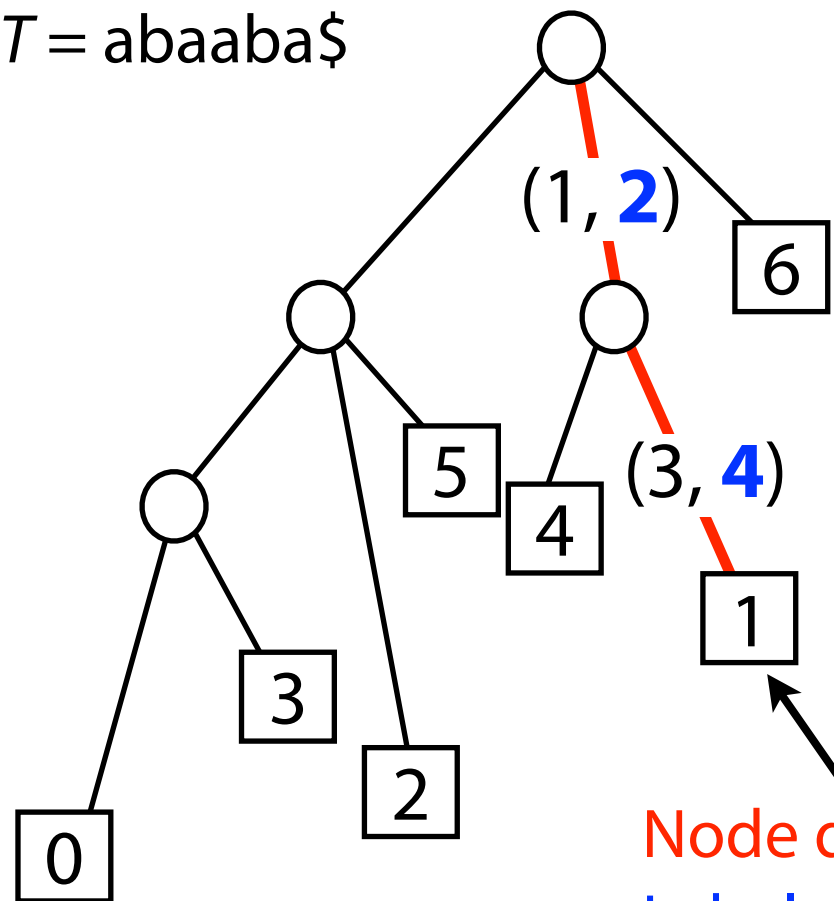


Suffix tree: leaves hold offsets



Suffix tree: labels

$T = \text{abaaba}\$$



Node depth = 2

Label depth = 2 + 4 = 6

Two notions of depth:

- **Node** depth: # edges from root to node
- **Label** depth: total length of edge labels from root to node

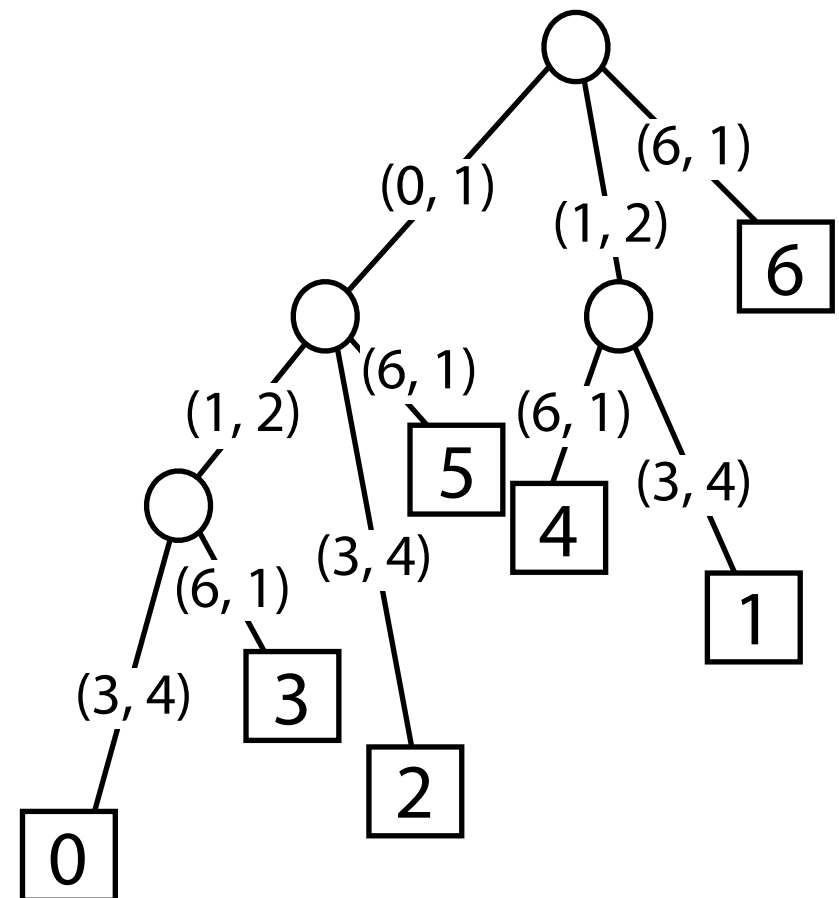
Suffix tree: building

Method 1: build suffix trie, coalesce non-branching paths, relabel edges

$O(m^2)$ time, $O(m^2)$ space

Method 2: build single-edge tree representing longest suffix, augment to include the 2nd-longest, augment to include 3rd-longest, etc (Gusfield 5.4)

$O(m^2)$ time, $O(m)$ space



Suffix tree: building

Canonical method: Ukkonen's algorithm

Ukkonen, Esko. "On-line construction of suffix trees."
Algorithmica 14.3 (1995): 249-260.

$O(m)$ time and space!

Won't cover it in class; see Gusfield Ch. 6 for details

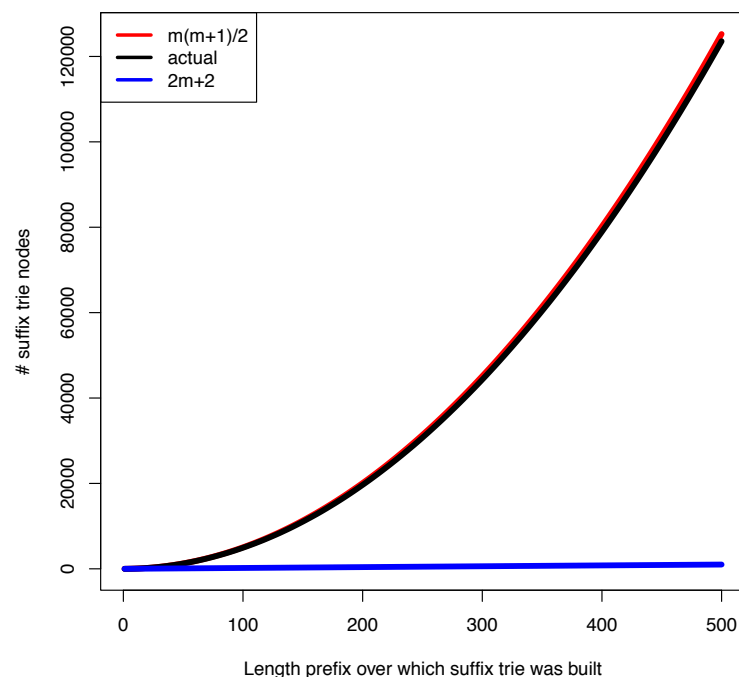
Also: McCreight's $O(m)$ time and space
algorithm, see Sec. 10.3.3 in the textbook.

Suffix tree: actual growth

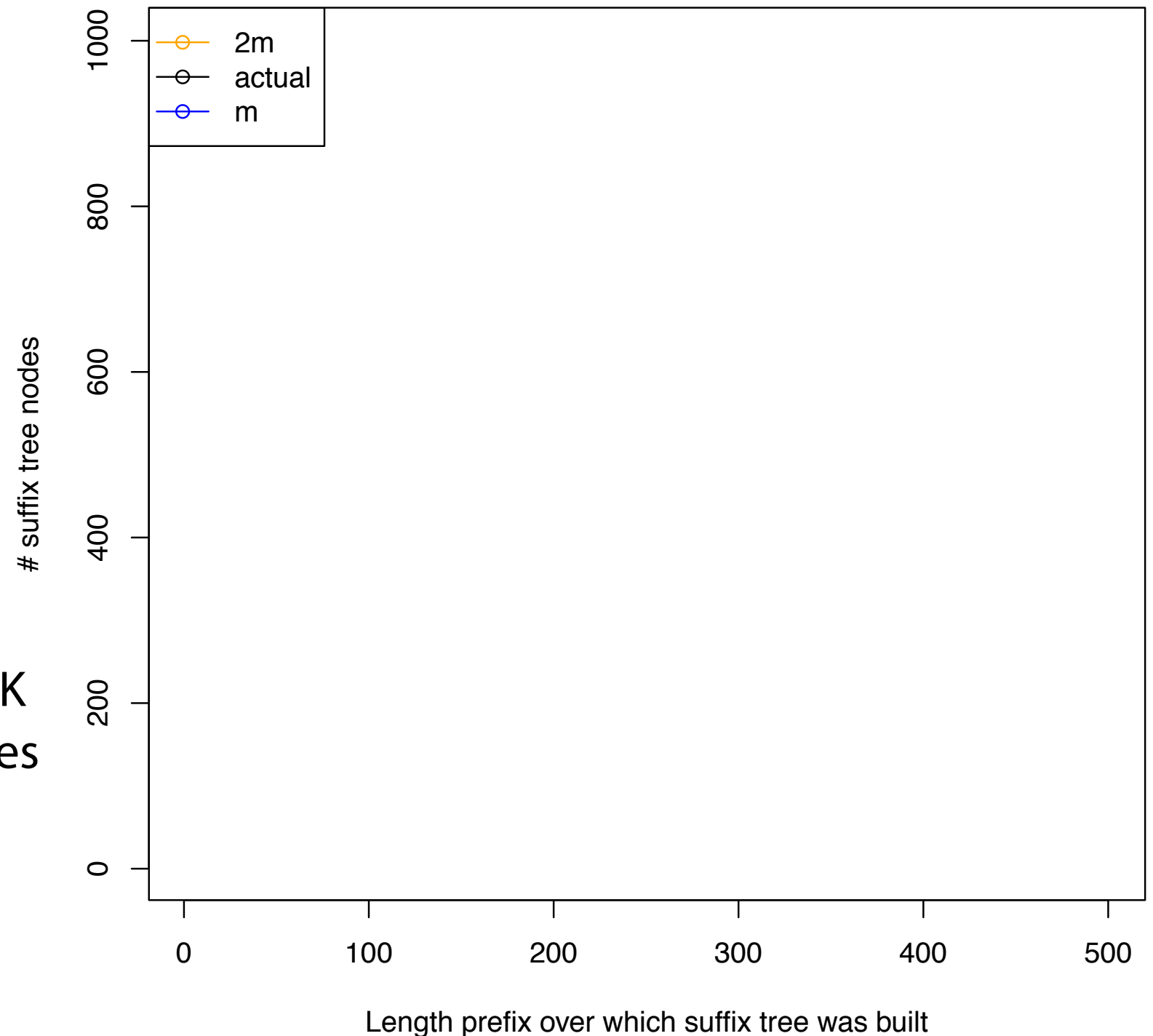
Built suffix trees for the first 500 prefixes of the lambda phage virus genome

Black curve shows # nodes increasing with prefix length

Remember suffix trie plot:



123 K
nodes

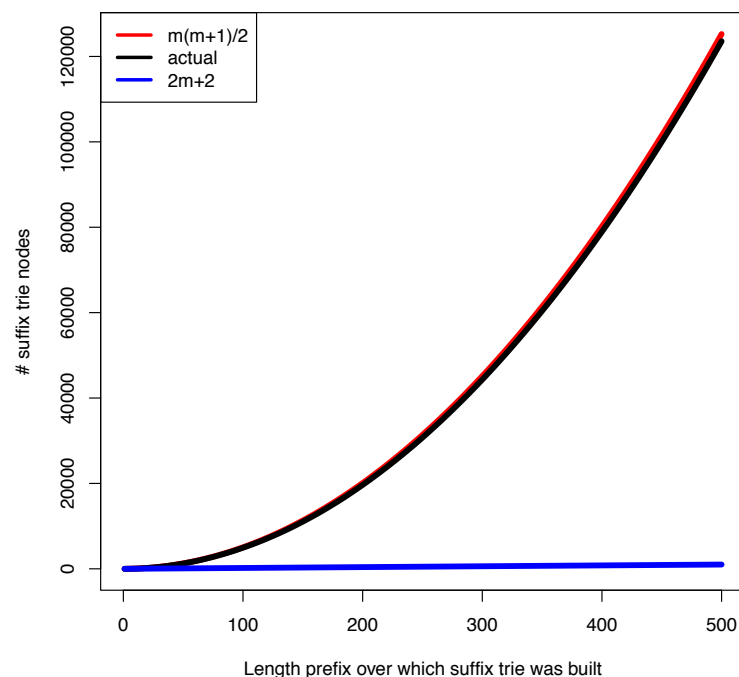


Suffix tree: actual growth

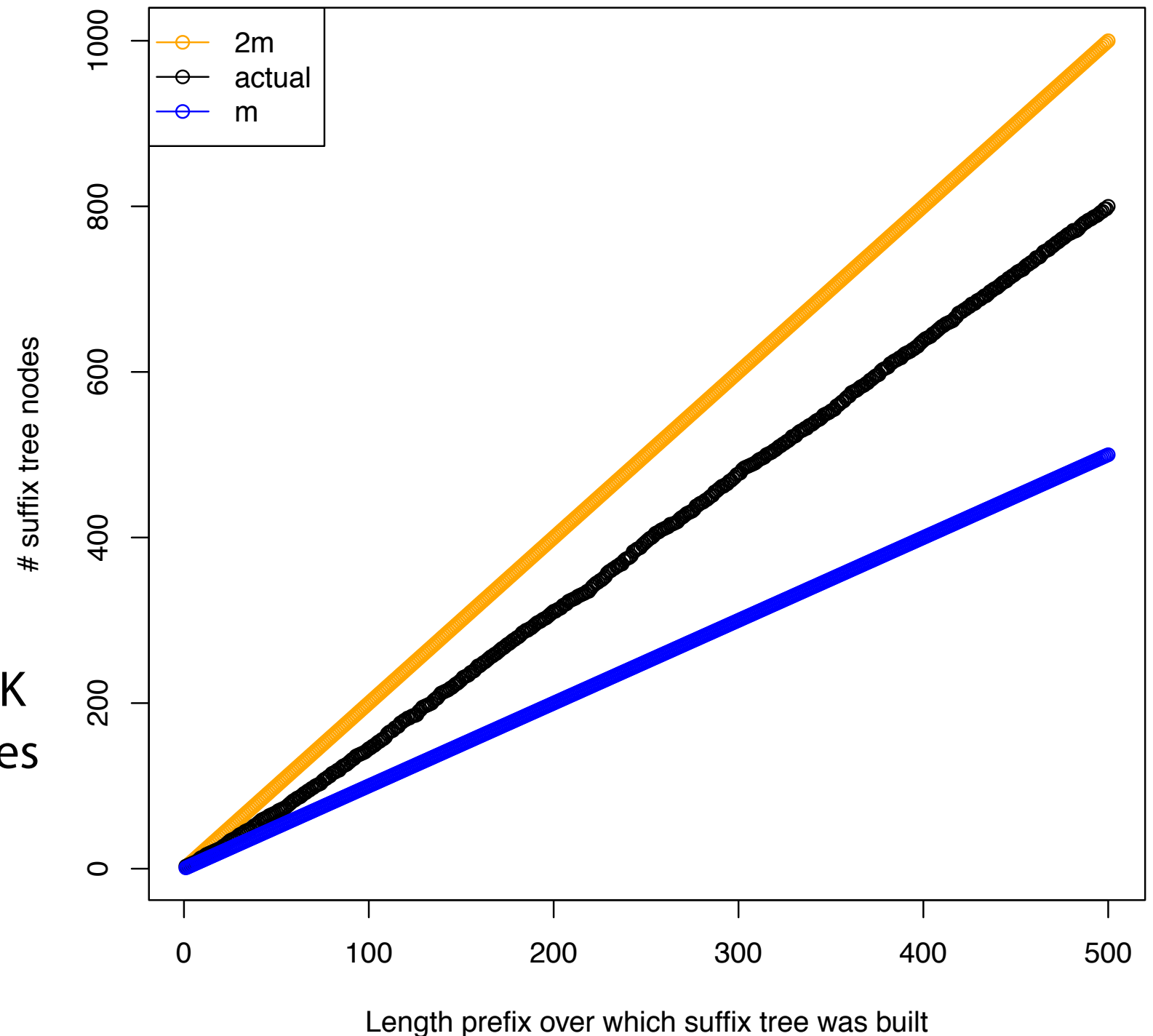
Built suffix trees for the first 500 prefixes of the lambda phage virus genome

Black curve shows # nodes increasing with prefix length

Remember suffix trie plot:

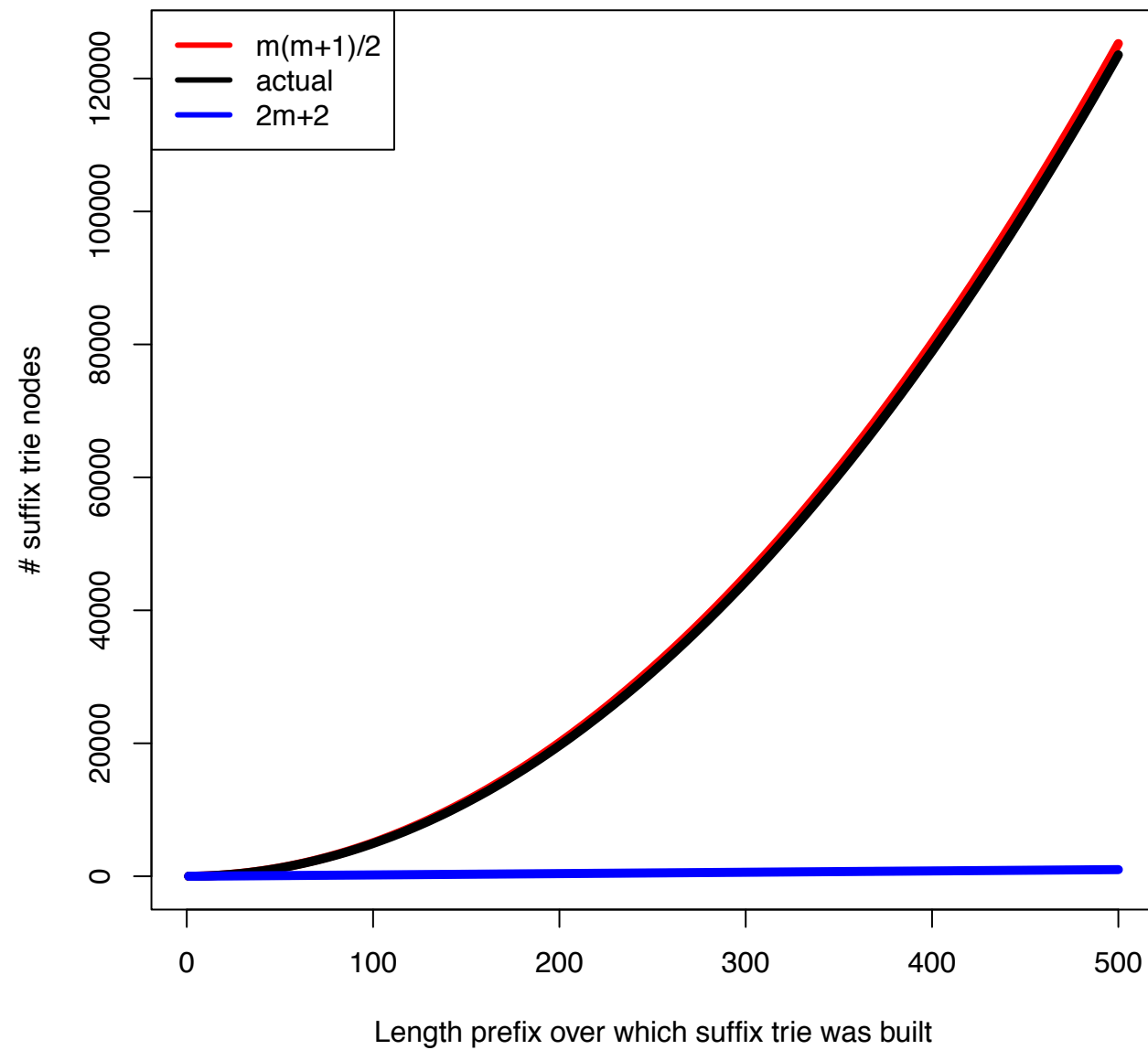


123 K nodes



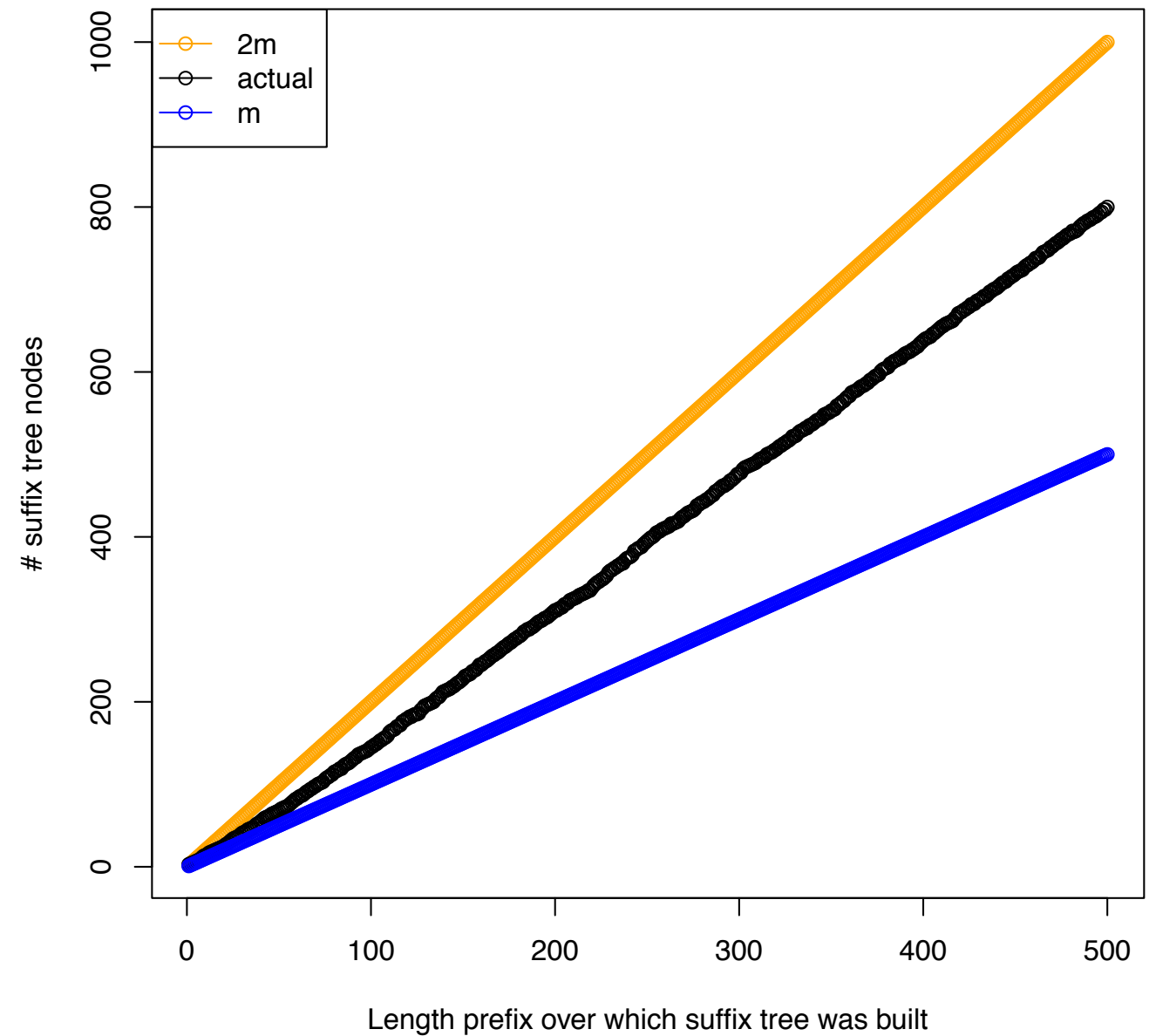
Suffix trie

>100K nodes



Suffix tree

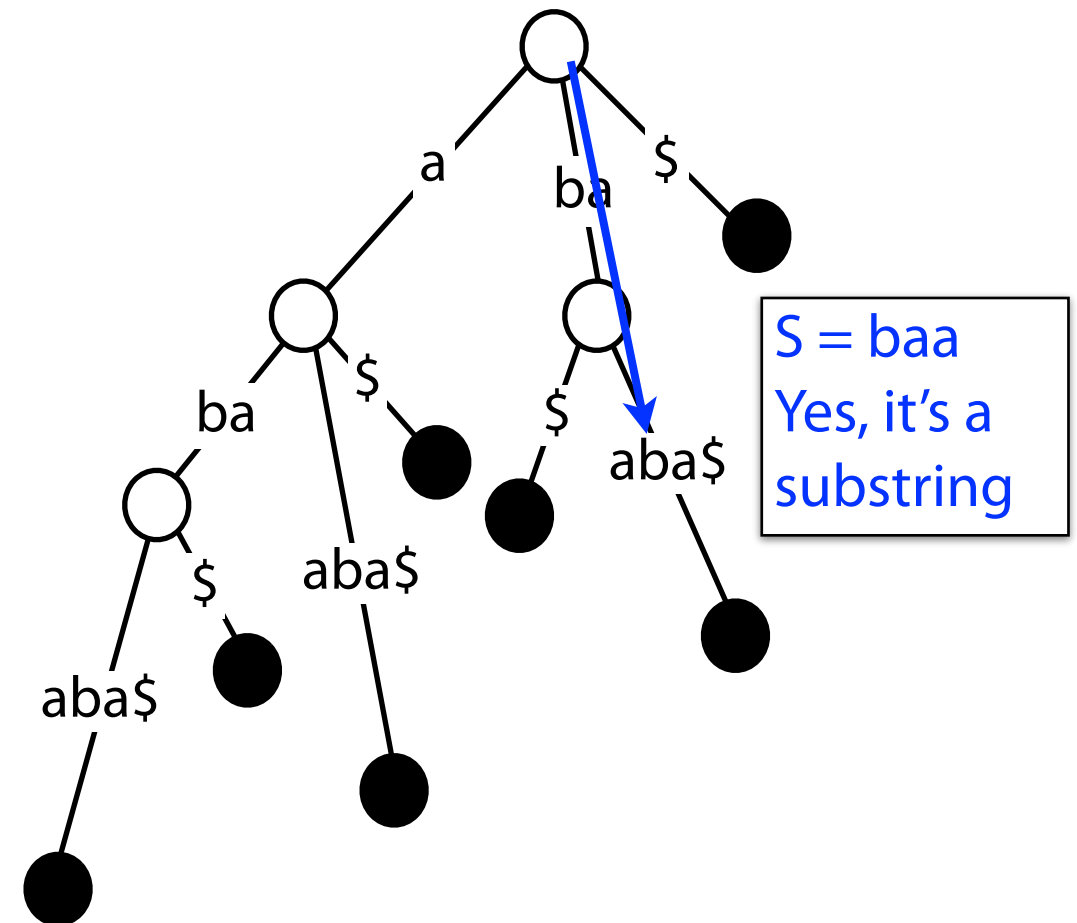
<1K nodes



Suffix tree

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

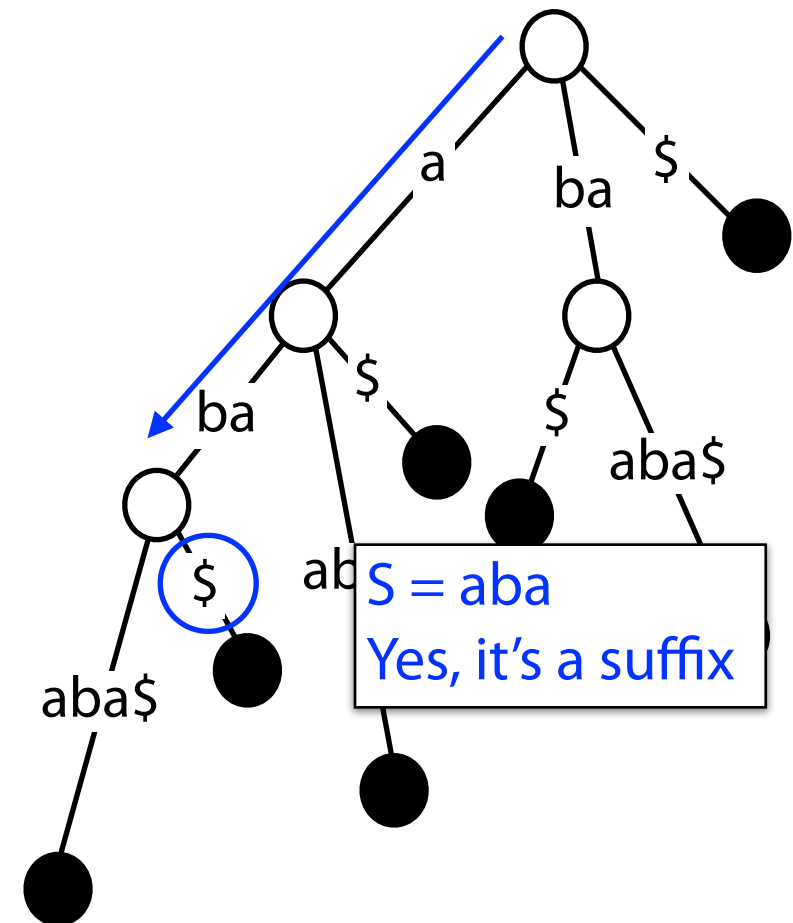
Same procedure as for suffix trie, but we have to deal with coalesced edges



Suffix tree

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

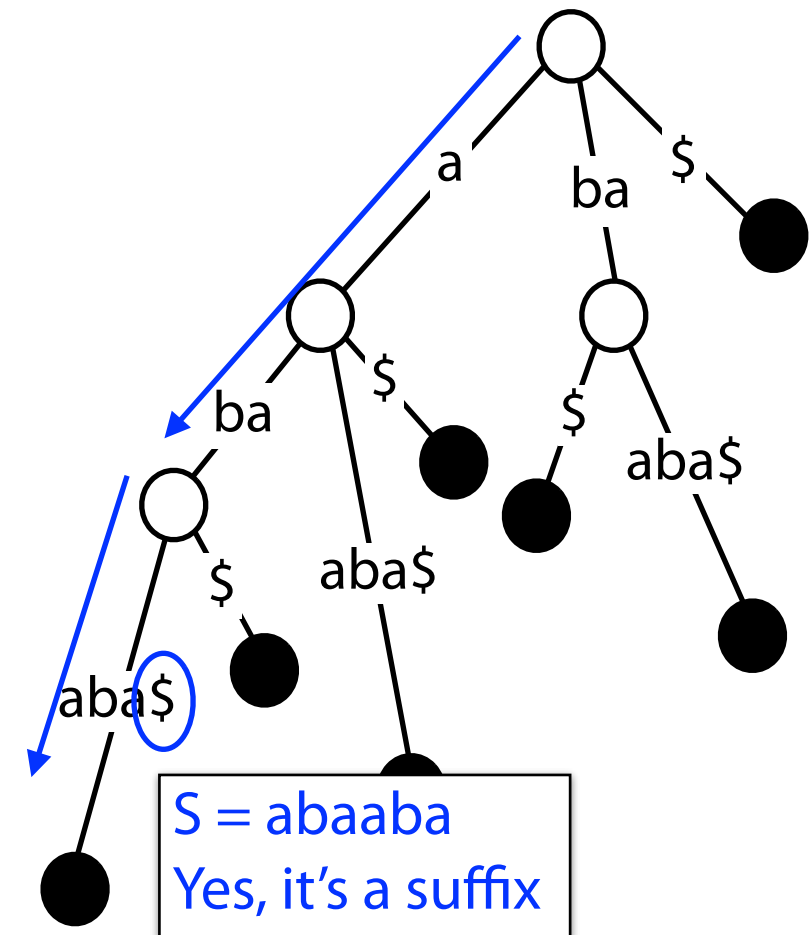
Same procedure as for suffix trie, but we have to deal with coalesced edges



Suffix tree

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

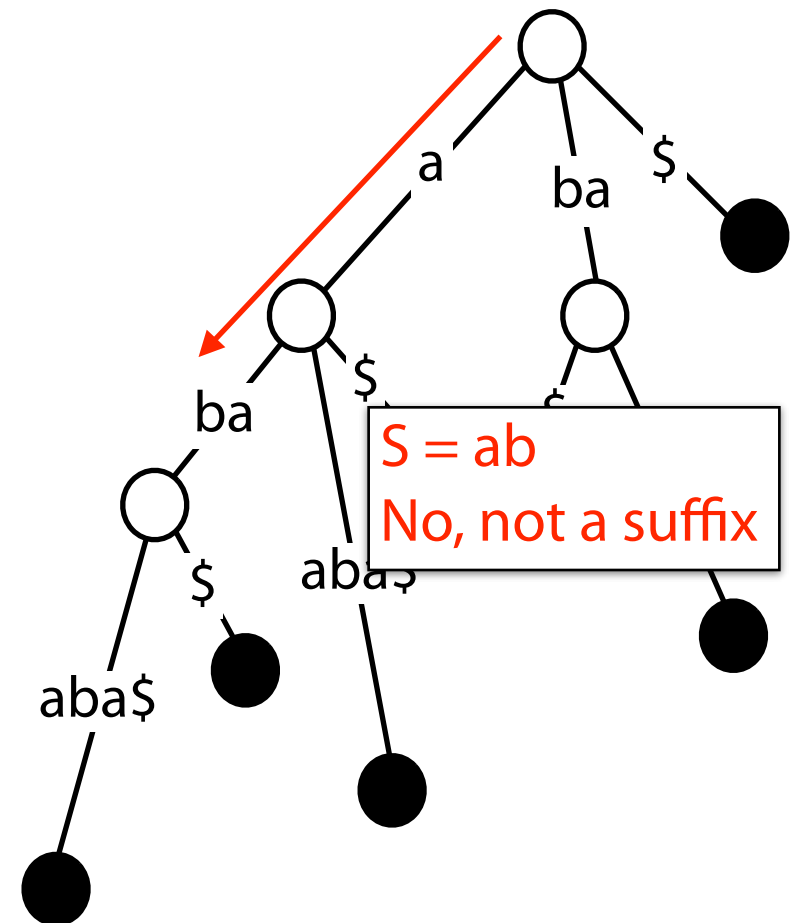
Same procedure as for suffix trie, but we have to deal with coalesced edges



Suffix tree

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

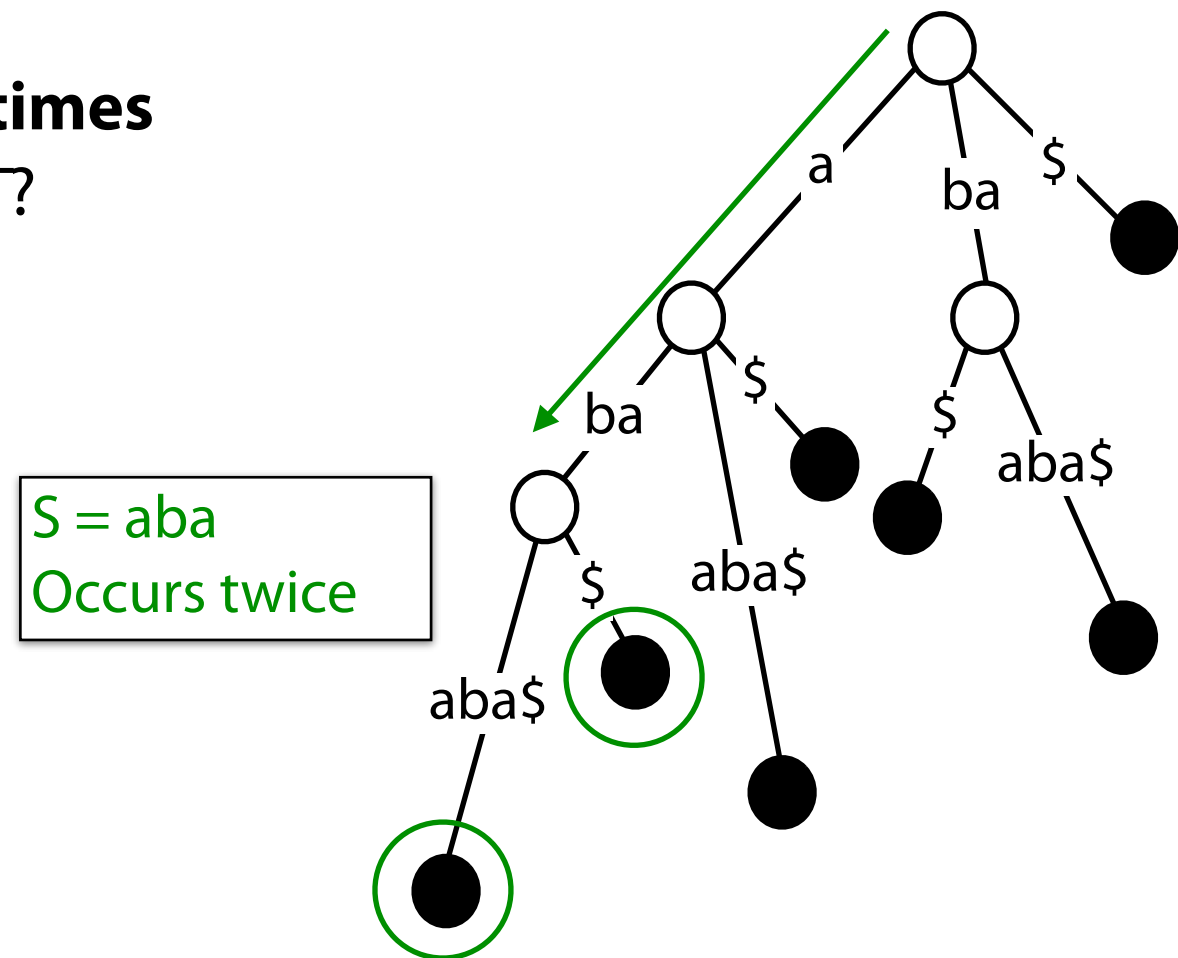
Same procedure as for suffix trie, but we have to deal with coalesced edges



Suffix tree

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

Same procedure as for suffix trie



Suffix tree

We can also **count or find** all the matches of P to T . Let $k = \#$ matches.

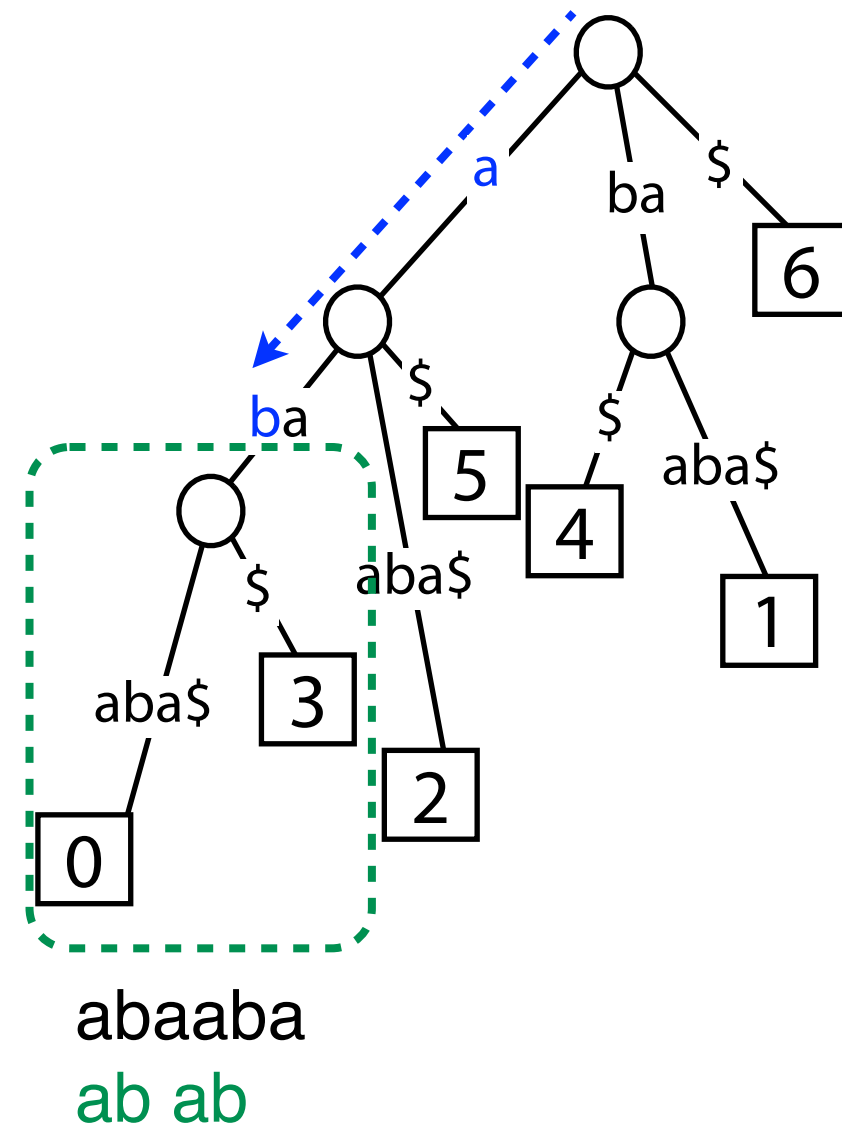
E.g., $P = ab$, $T = abaaba\$$

$O(n)$ Step 1: walk down ab path
If we “fall off” there are no matches

$O(k)$ Step 2: visit all leaf nodes below
Report each leaf offset as match offset

leaves in subtree is k ,
non-leaves is $\leq k-1$

$O(n + k)$ time overall



Suffix tree: some bounds

Suffix tree	
Time: Does P occur?	$O(n)$
Time: Count k occurrences of P	$O(n + k)$
Time: Report k locations of P	$O(n + k)$
Space	$O(m)$

$m = |T|$, $n = |P|$, $k = \#$ occurrences of P in T

Suffix trees in the real world: the constant factor

$O(m)$ is desirable, but “constant factor” is significant, sometimes making the suffix tree inconvenient

Constant factor varies depending on implementation:

MUMmer constant factor = 3.94 GB / 250 million nt $\approx$ **15.76 bytes per nt**

Kurtz, Stefan. "Reducing the space requirement of suffix trees." *Software Practice and Experience* 29.13 (1999): 1149-1171.

Suffix tree of human genome will be >45GB, perhaps much larger depending on exact data structures underlying suffix tree nodes/edges

Exercise:

Given strings s and t use a suffix tree to compute their longest common substring

Hint: for $s=\text{banana}$ $t=\text{ananas}$

consider their concatenation $s\#t\$ \rightarrow \text{banana\#ananas\$}$