Suffix Trees

Ben Langmead

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Department of Computer Science

CC BY

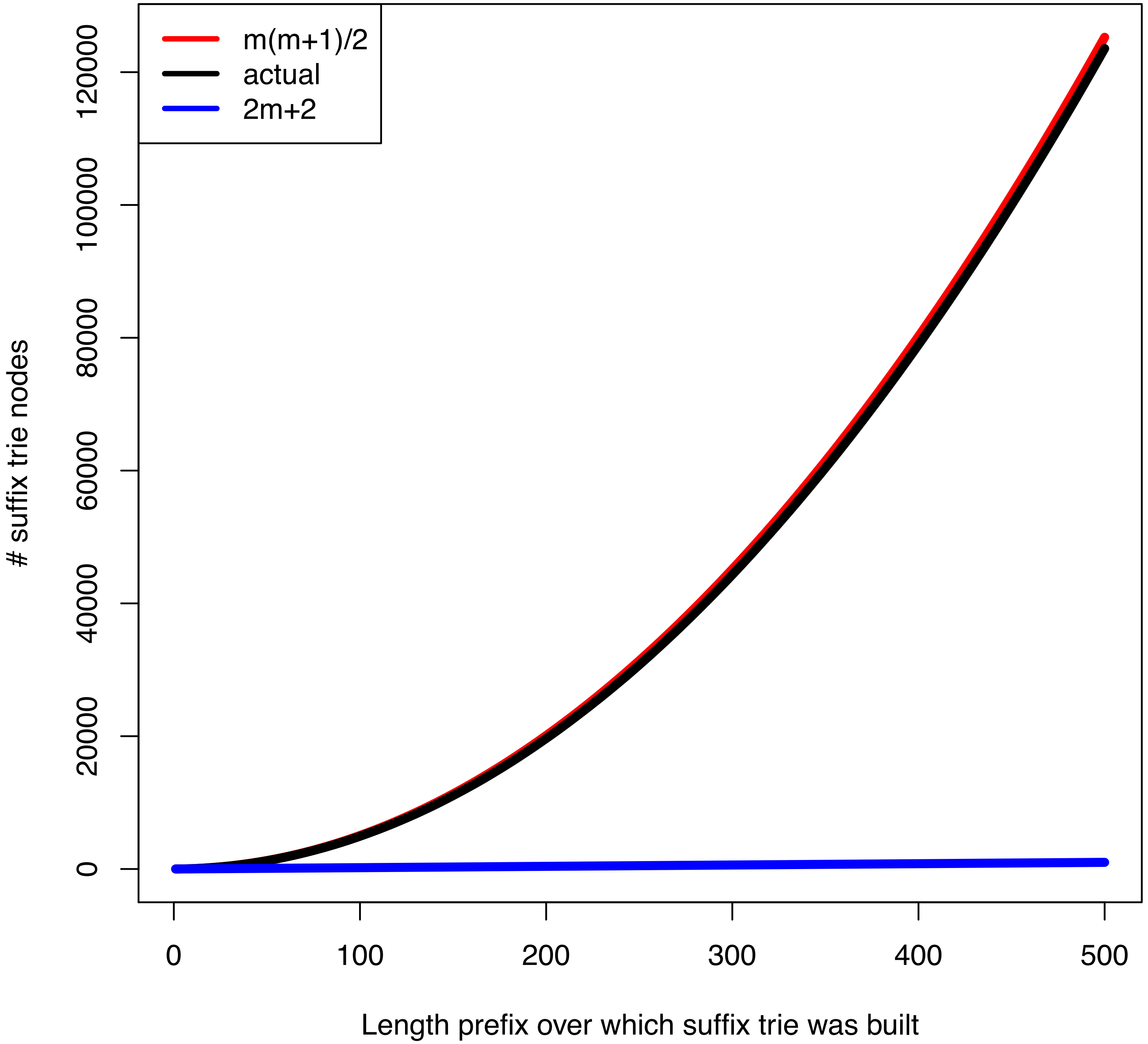
Please sign guestbook (www.langmead-lab.org/teaching-materials) to tell me briefly how you are using the slides. For original Keynote files, email me (ben.langmead@gmail.com).

Suffix trie

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

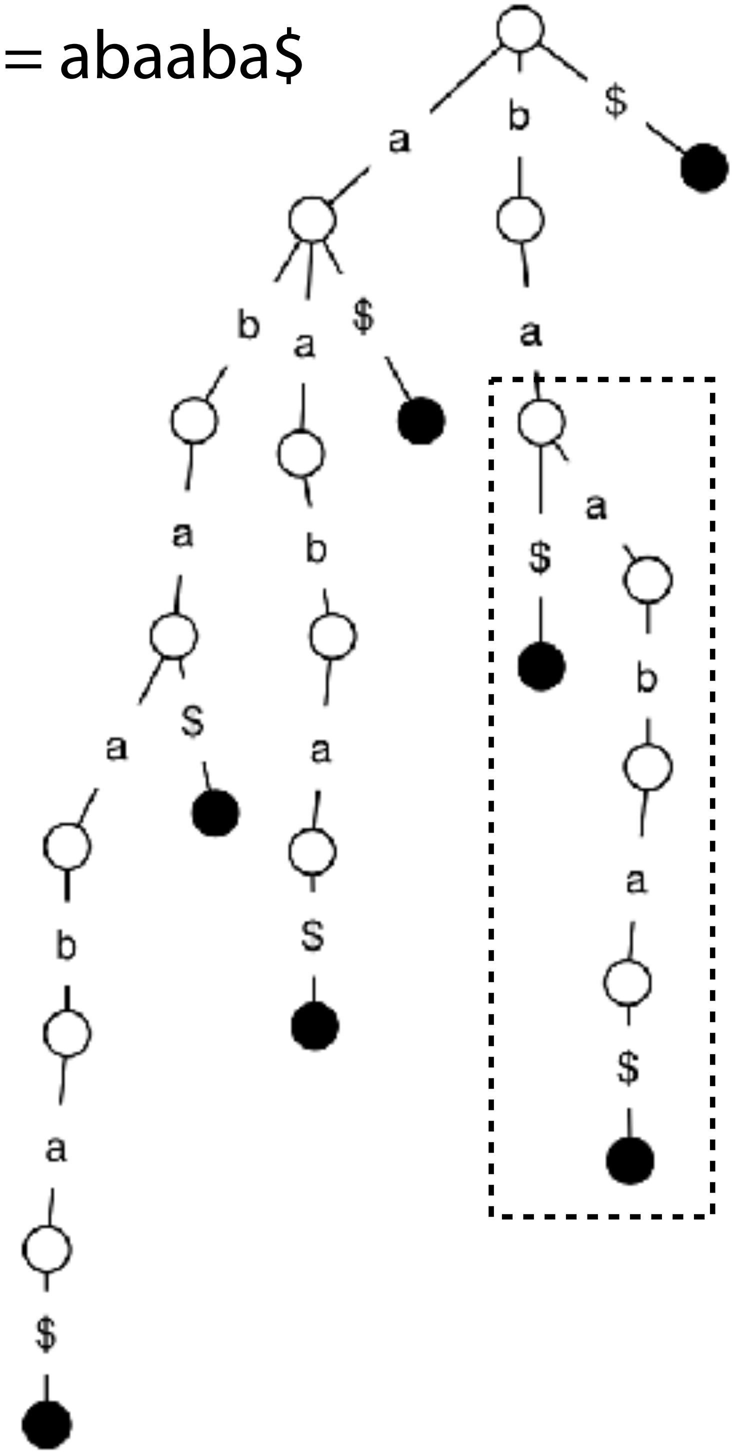
Human genome is $3 \cdot 10^9$ bases long.

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



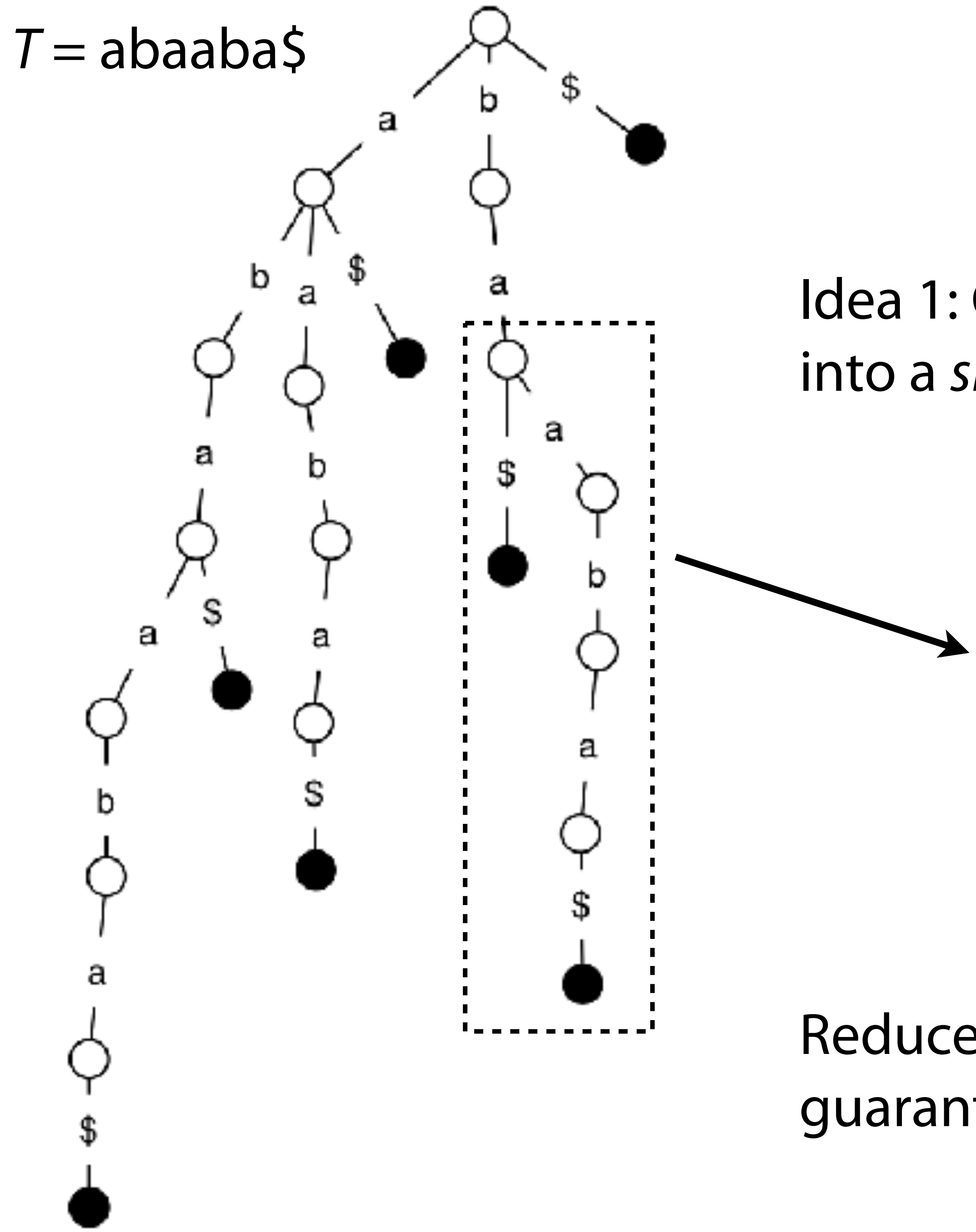
Suffix trie: making it smaller

T = abaaba$

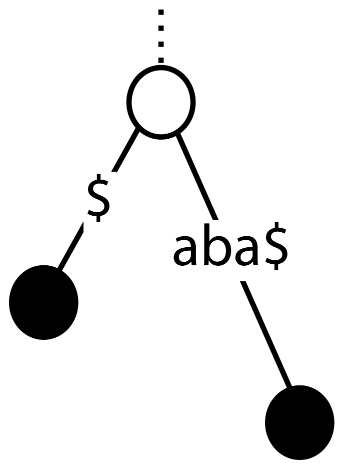


Suffix trie: making it smaller

T = abaaba$



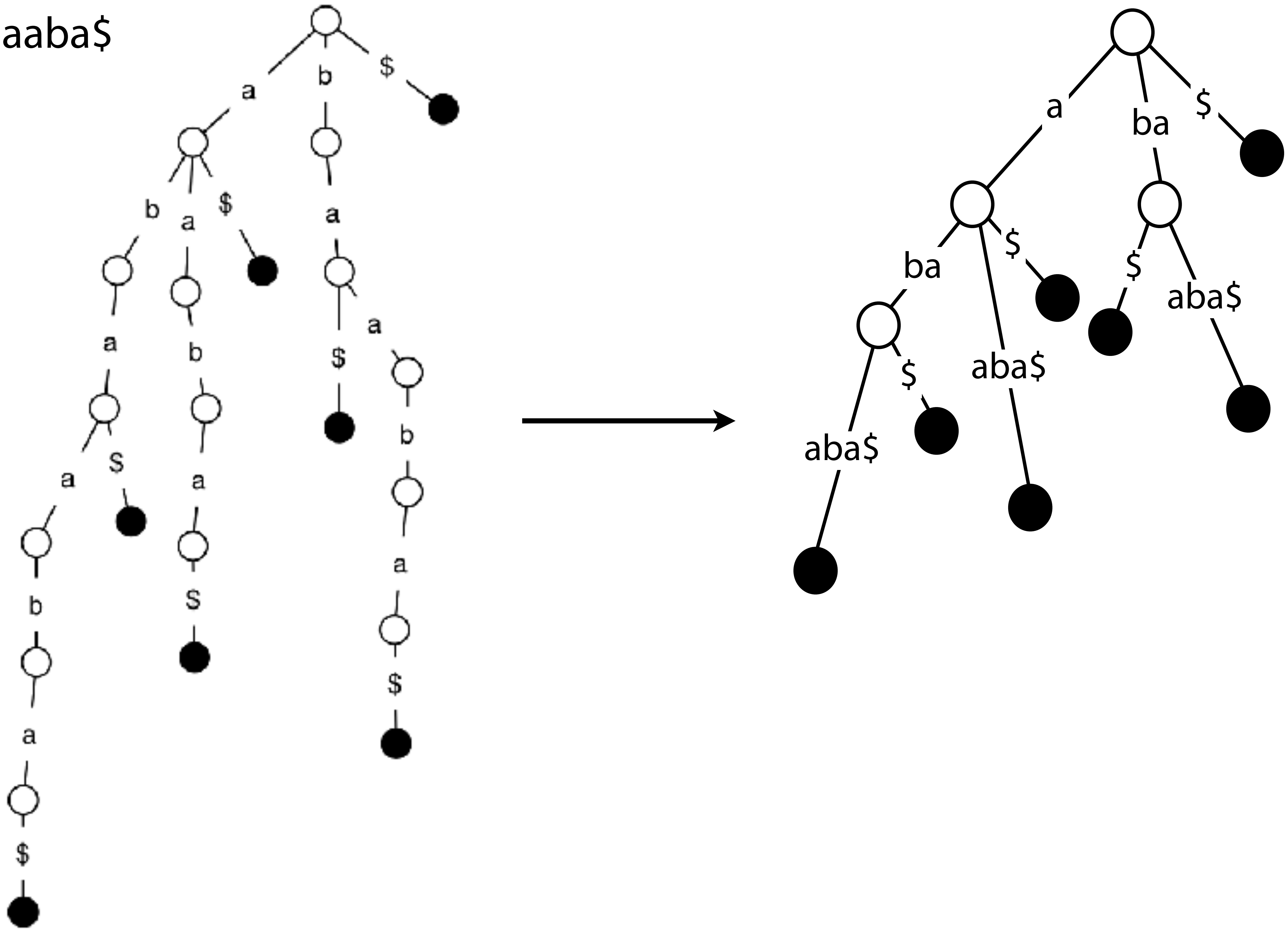
Idea 1: Coalesce non-branching paths into a *single edge* with a *string* label



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

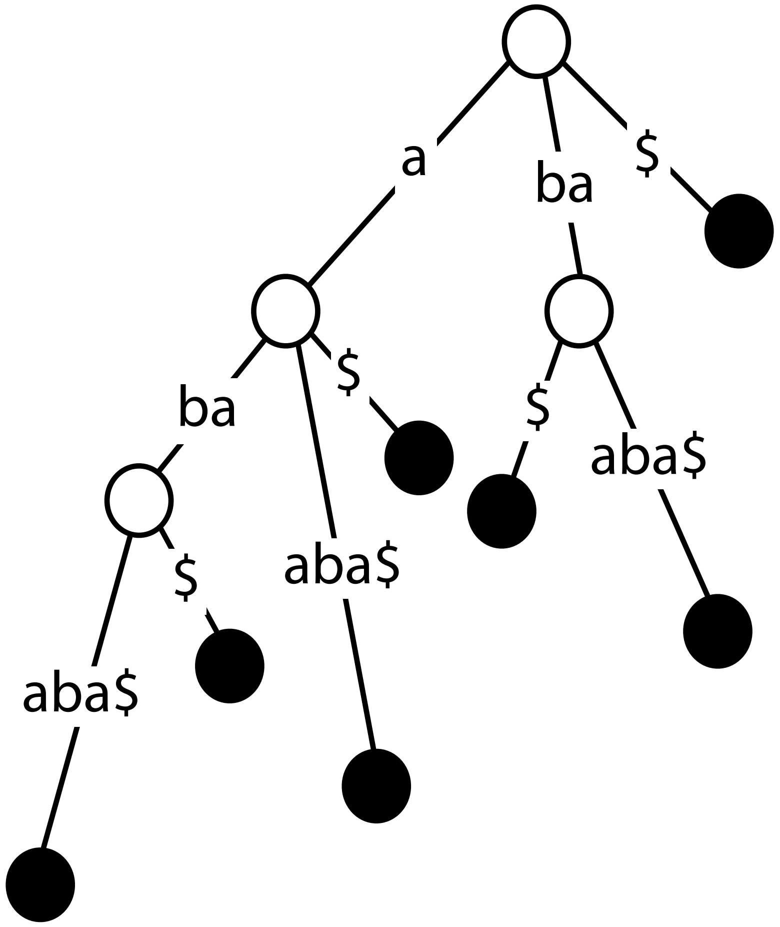
Suffix trie: making it smaller

T = abaaba$



Suffix tree

T = abaaba$ | *T* | = *m*



# leaves? ***m***

# non-leaf nodes (bound)? **≤ *m* - 1**

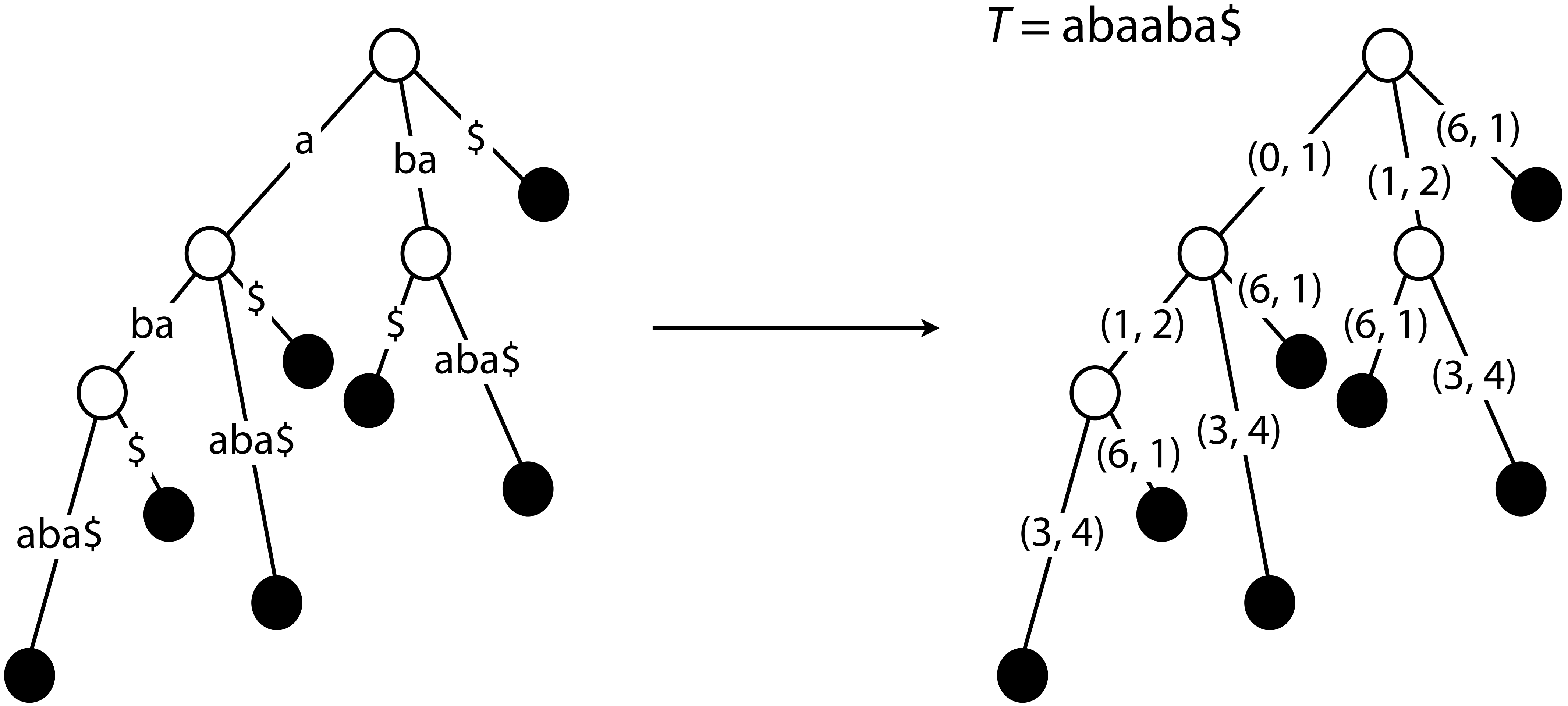
≤ 2*m* -1 nodes total — *O*(*m*)

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

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

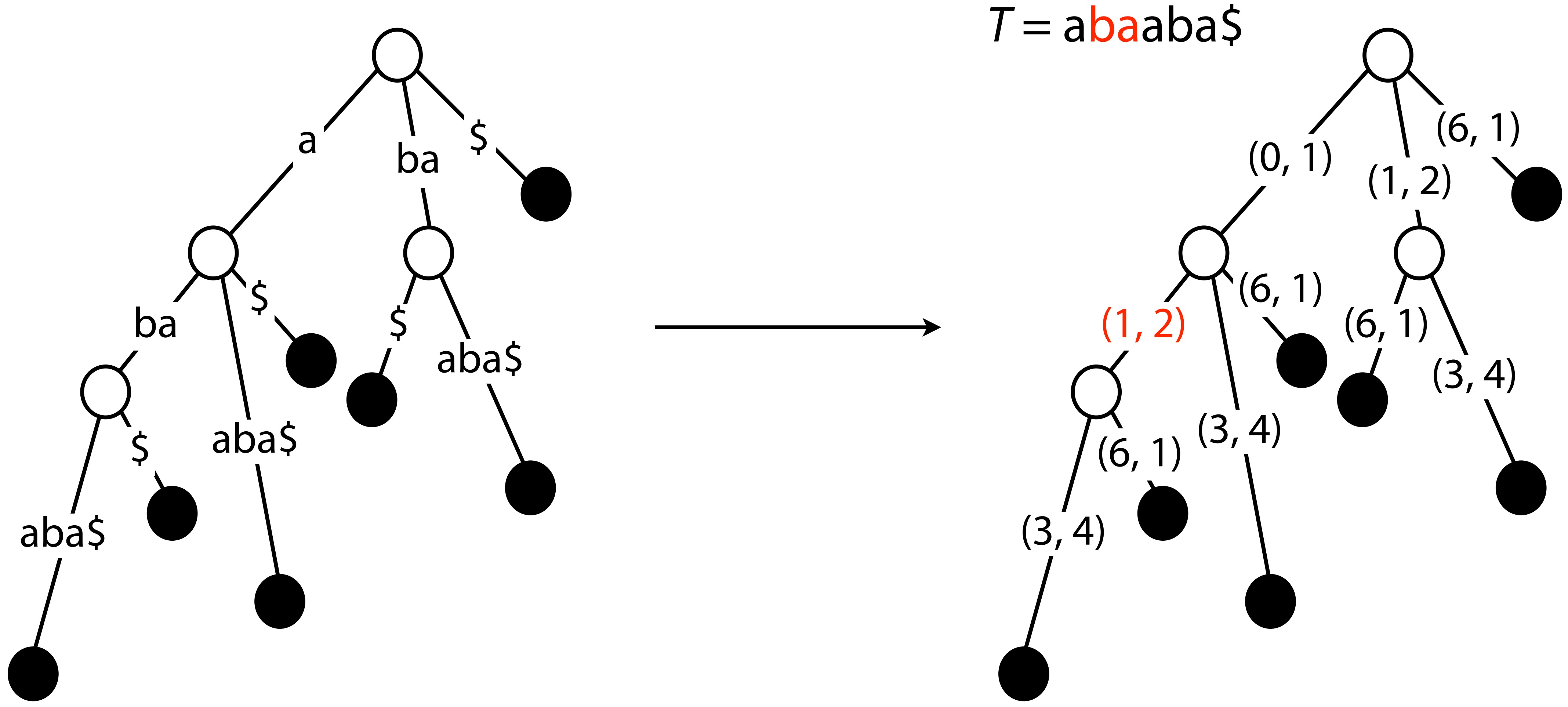
Suffix tree

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



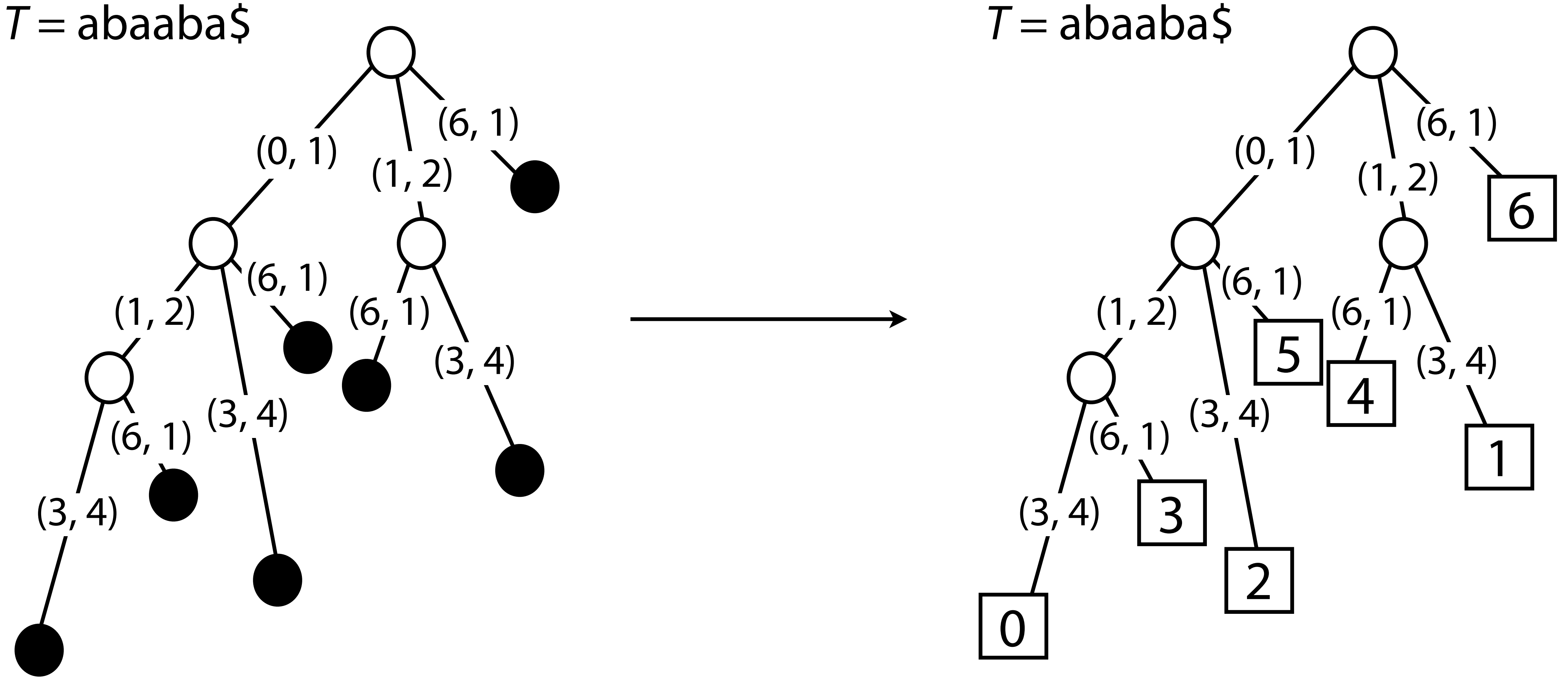
Suffix tree

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

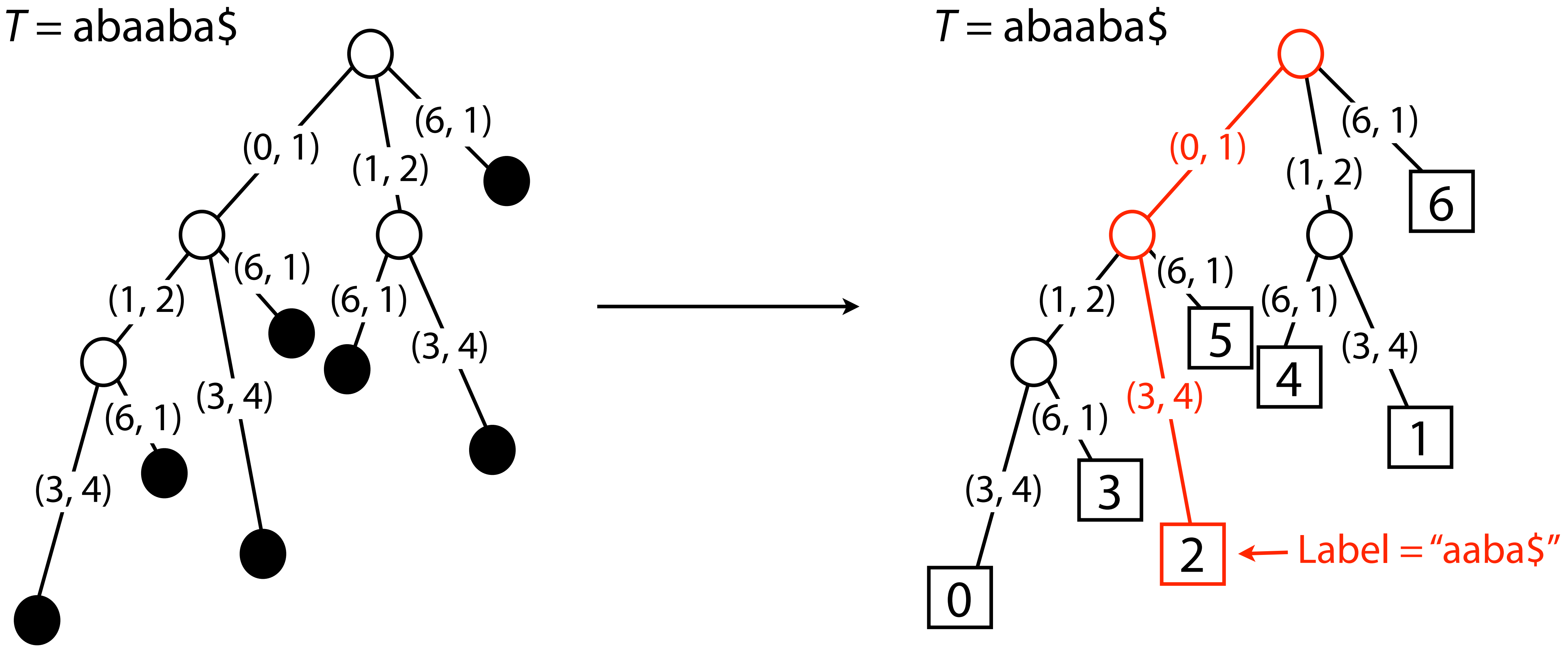


Space is now $O(m)$ Suffix trie was $O(m^2)$!

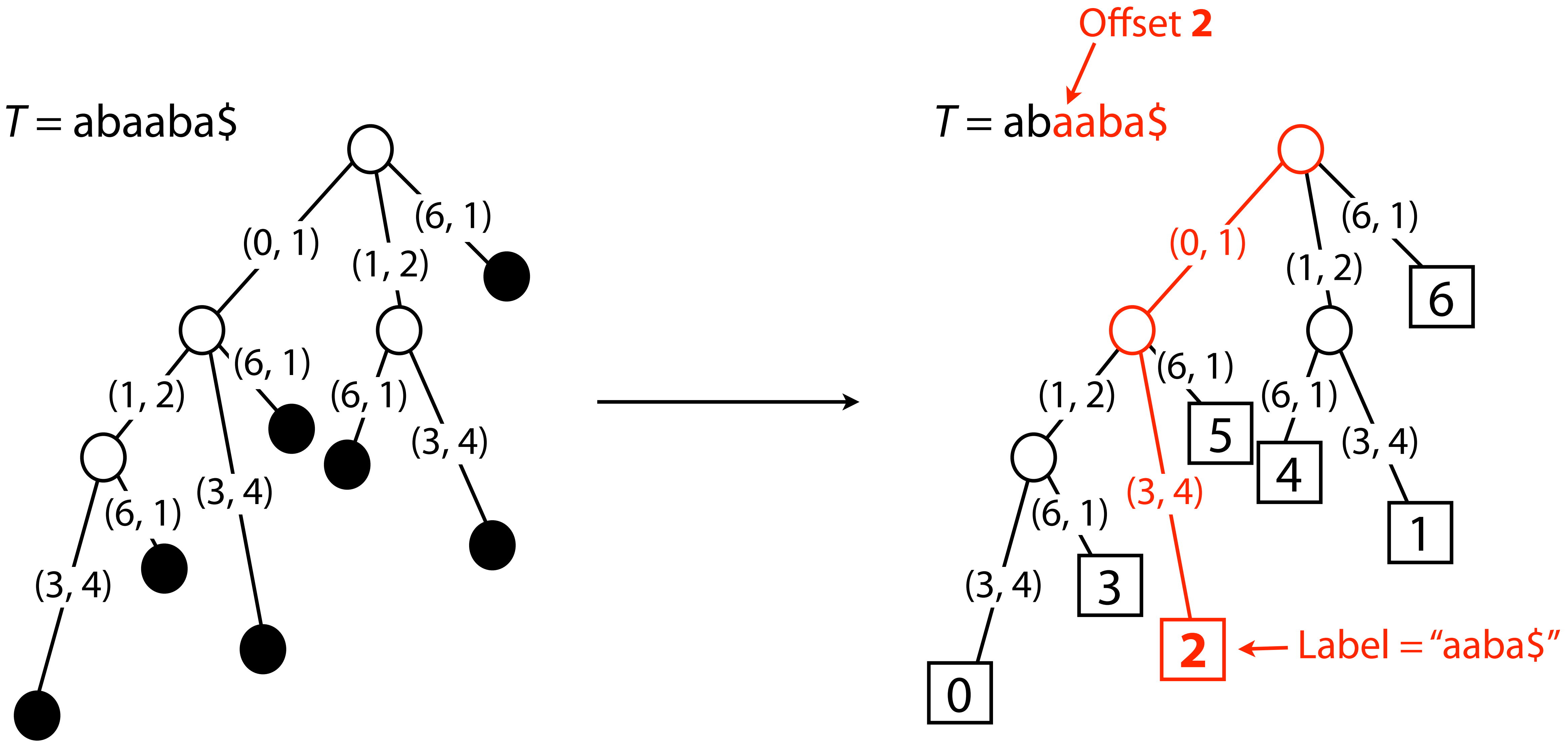
Suffix tree: leaves hold offsets



Suffix tree: leaves hold offsets

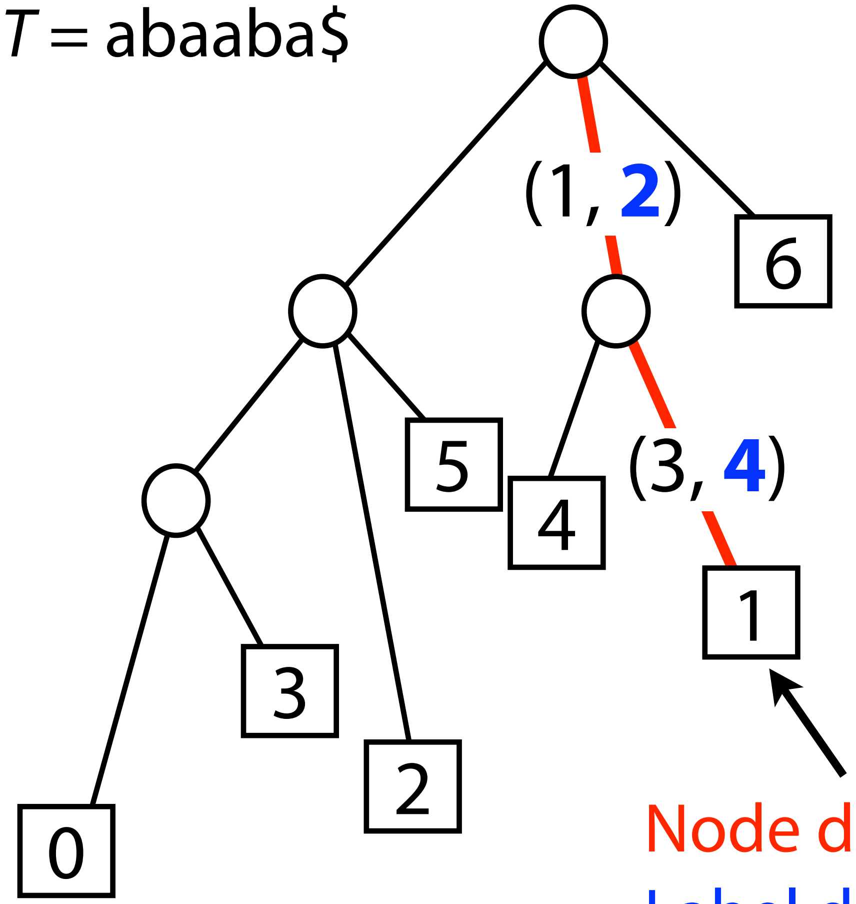


Suffix tree: leaves hold offsets



Suffix tree: labels

T = 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

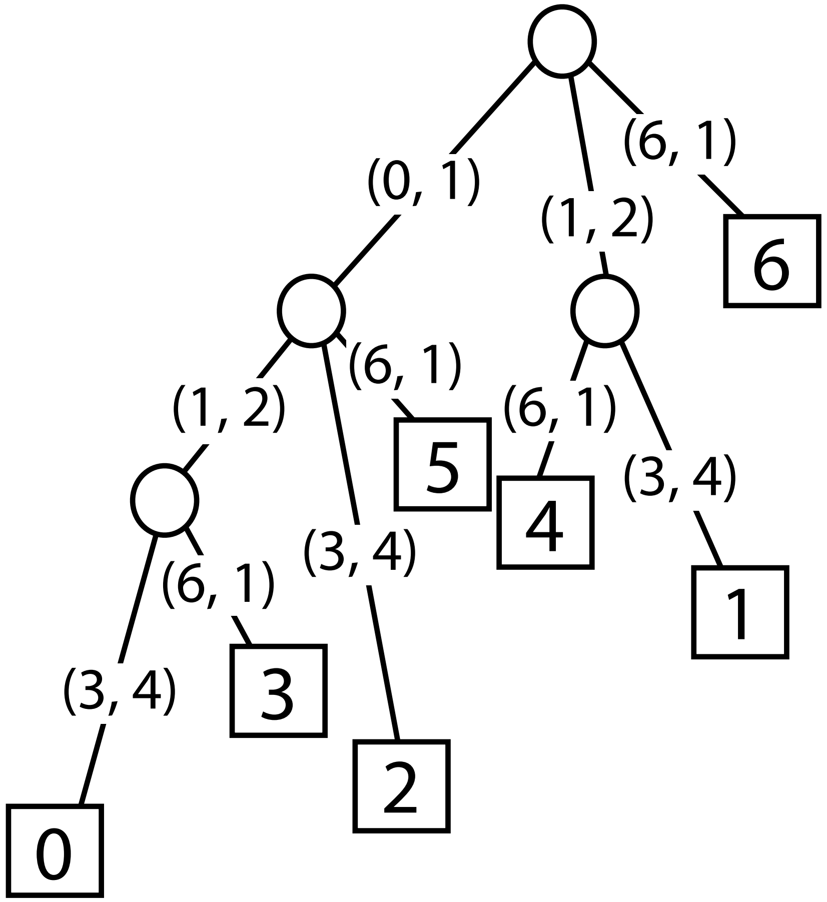
Suffix tree: building

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: implementation

http://bit.ly/CG_SuffixTree

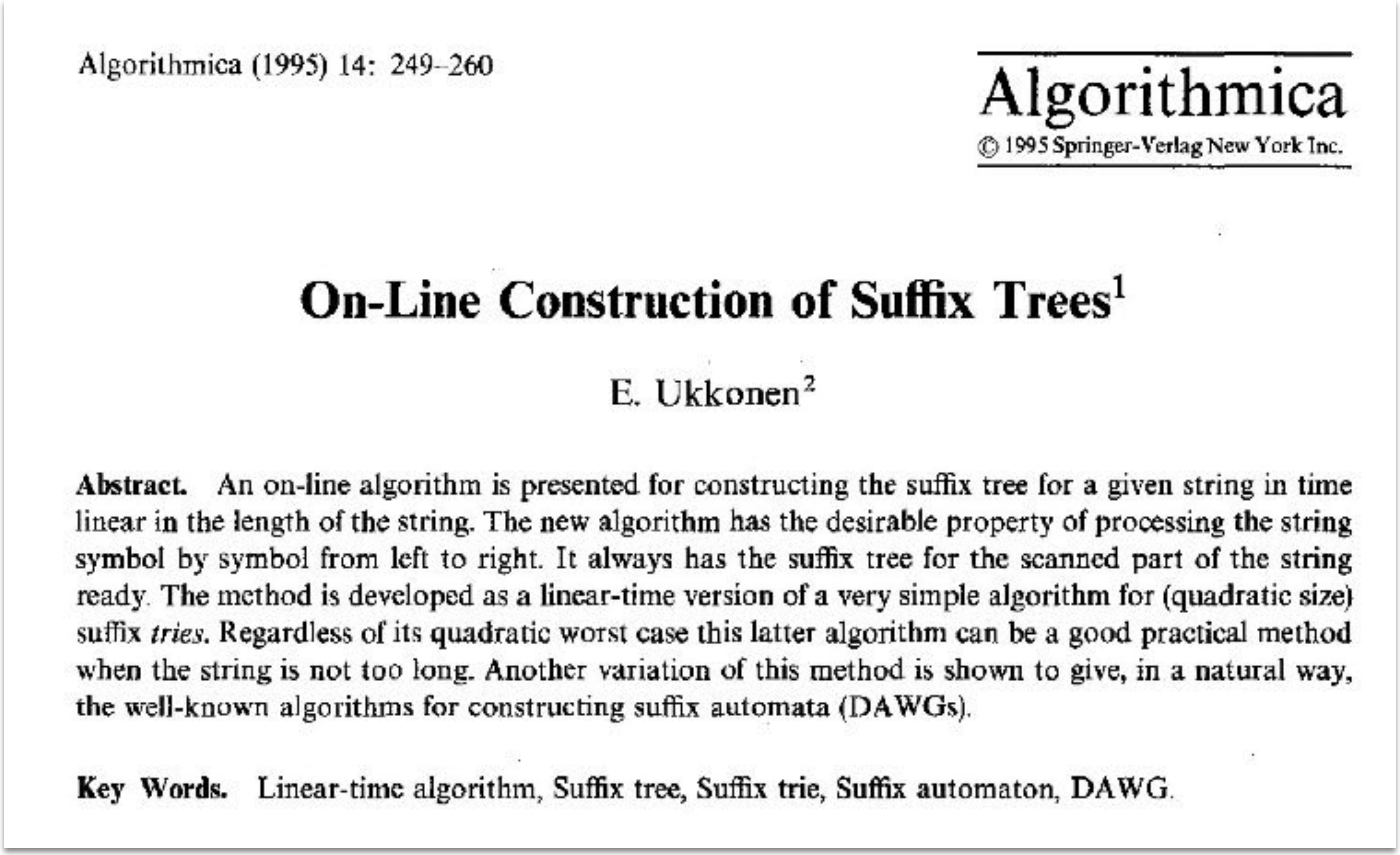
Suffix tree: building

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



Algorithmica (1995) 14: 249–260

Algorithmica
© 1995 Springer-Verlag New York Inc.

On-Line Construction of Suffix Trees[1]

E. Ukkonen[2]

Abstract. An on-line algorithm is presented for constructing the suffix tree for a given string in time linear in the length of the string. The new algorithm has the desirable property of processing the string symbol by symbol from left to right. It always has the suffix tree for the scanned part of the string ready. The method is developed as a linear-time version of a very simple algorithm for (quadratic size) suffix *tries*. Regardless of its quadratic worst case this latter algorithm can be a good practical method when the string is not too long. Another variation of this method is shown to give, in a natural way, the well-known algorithms for constructing suffix automata (DAWGs).

Key Words. Linear-time algorithm, Suffix tree, Suffix trie, Suffix automaton, DAWG.

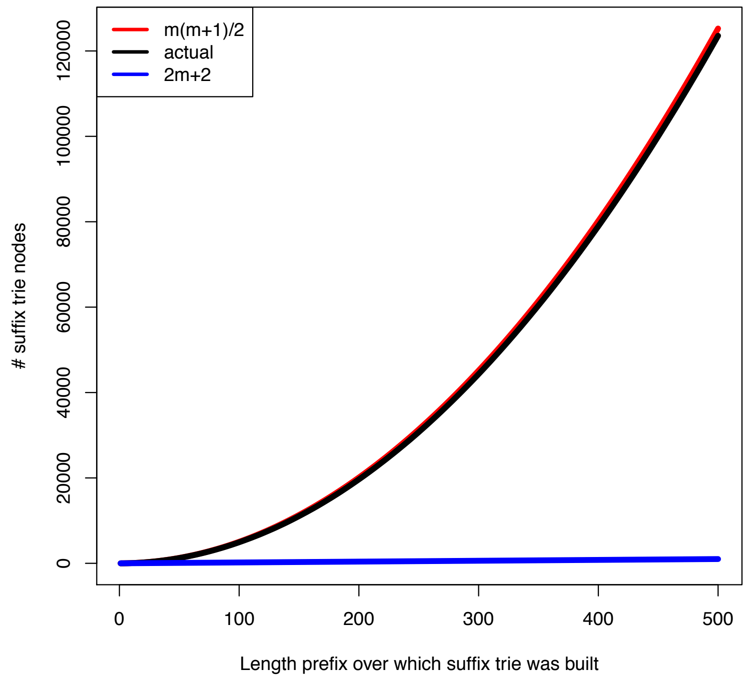
Canonical algorithm for $O(m)$ time & space suffix tree construction

Suffix tree: actual growth

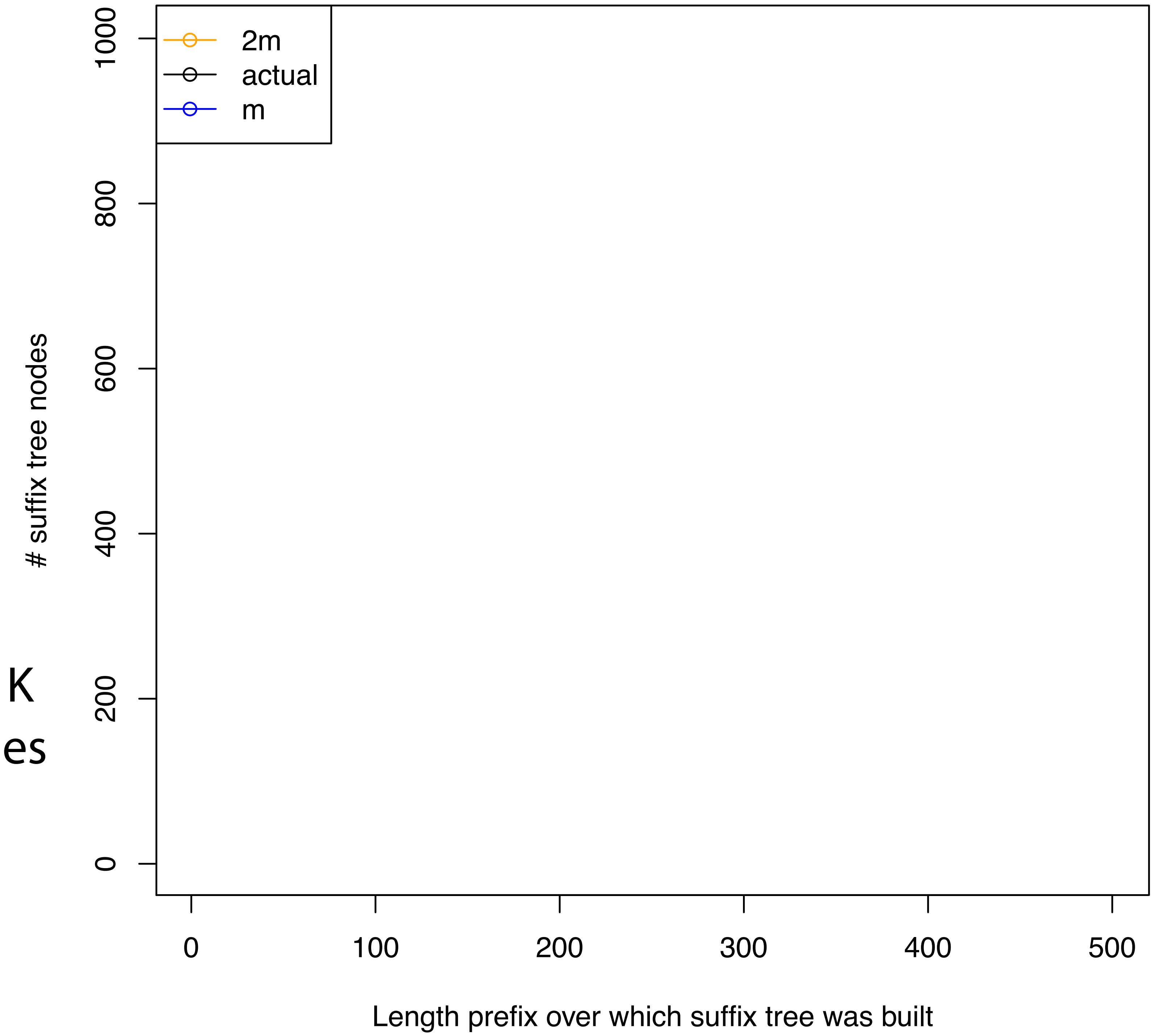
Built suffix trees for the first 500 prefixes of the lambda phage virus genome

Black curve shows # nodes increasing with prefix length

Remember suffix trie plot:



123 K nodes

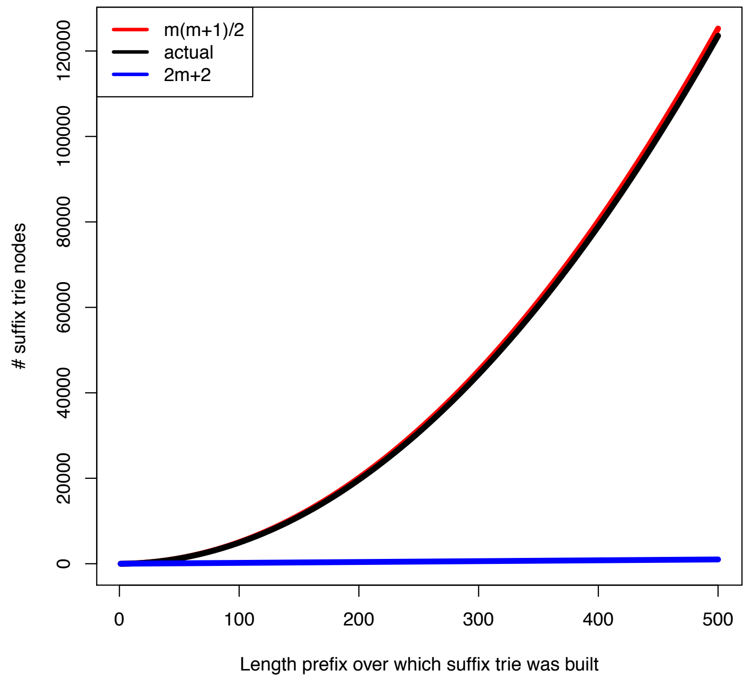


Suffix tree: actual growth

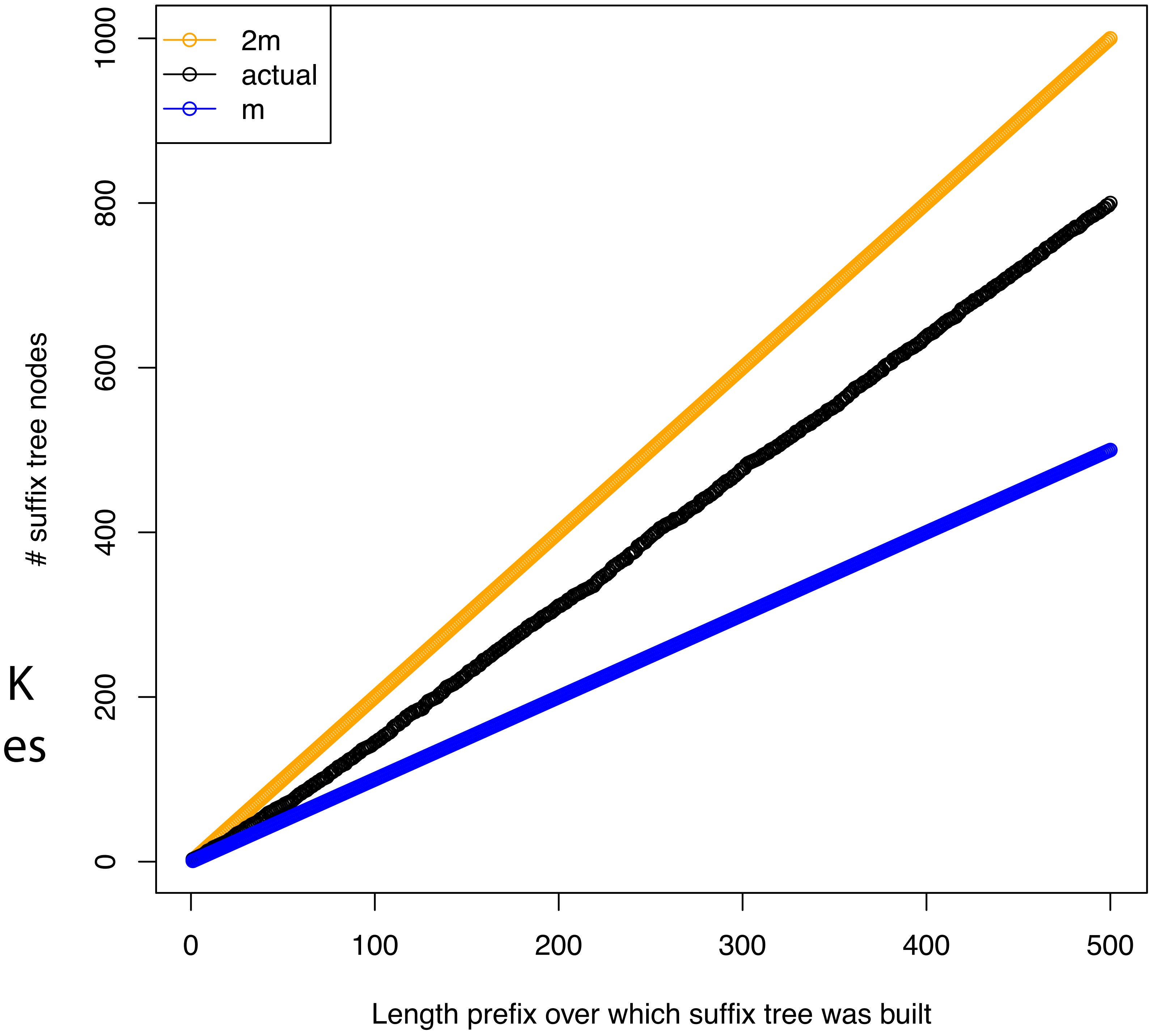
Built suffix trees for the first 500 prefixes of the lambda phage virus genome

Black curve shows # nodes increasing with prefix length

Remember suffix trie plot:

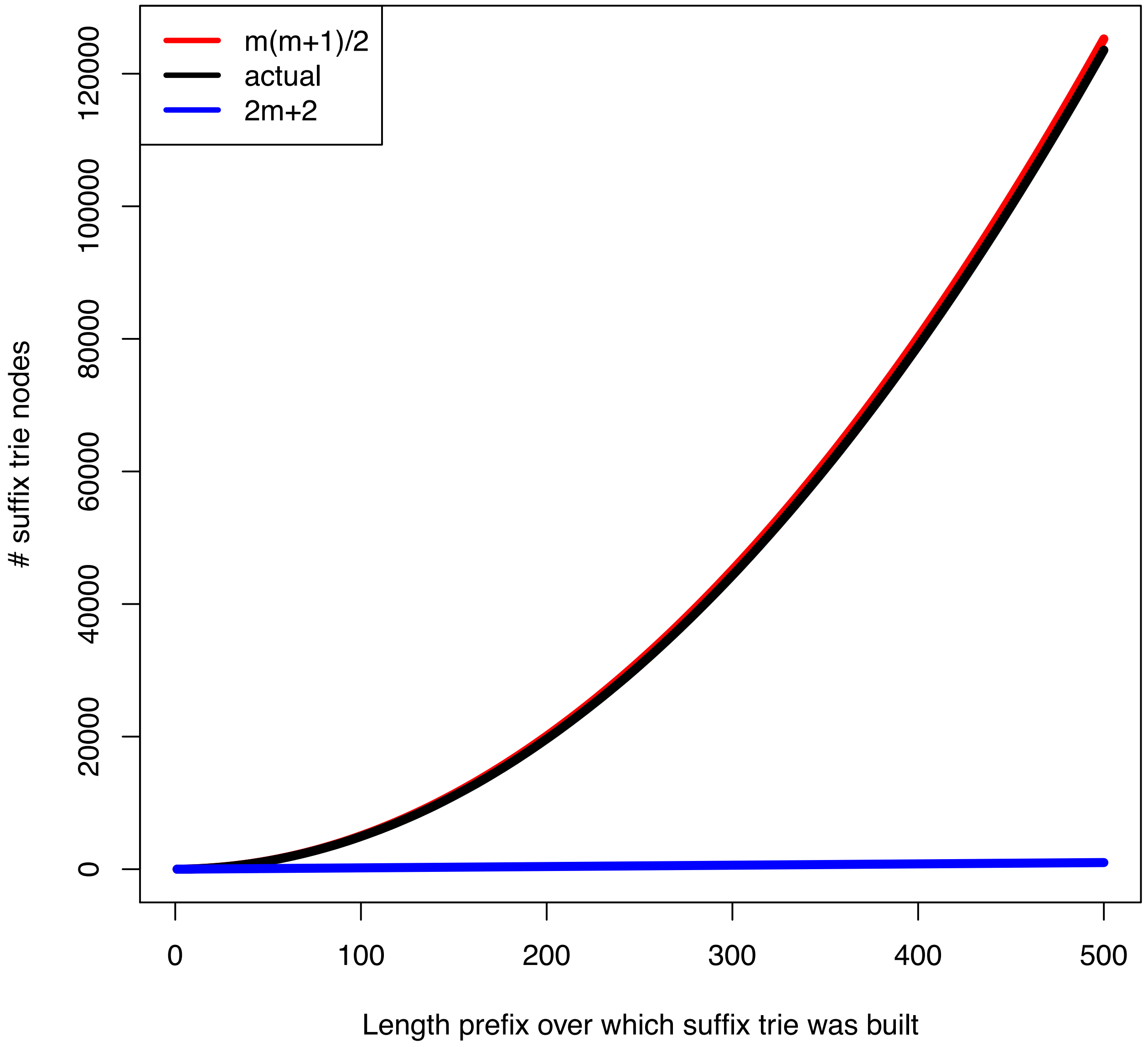


123 K nodes



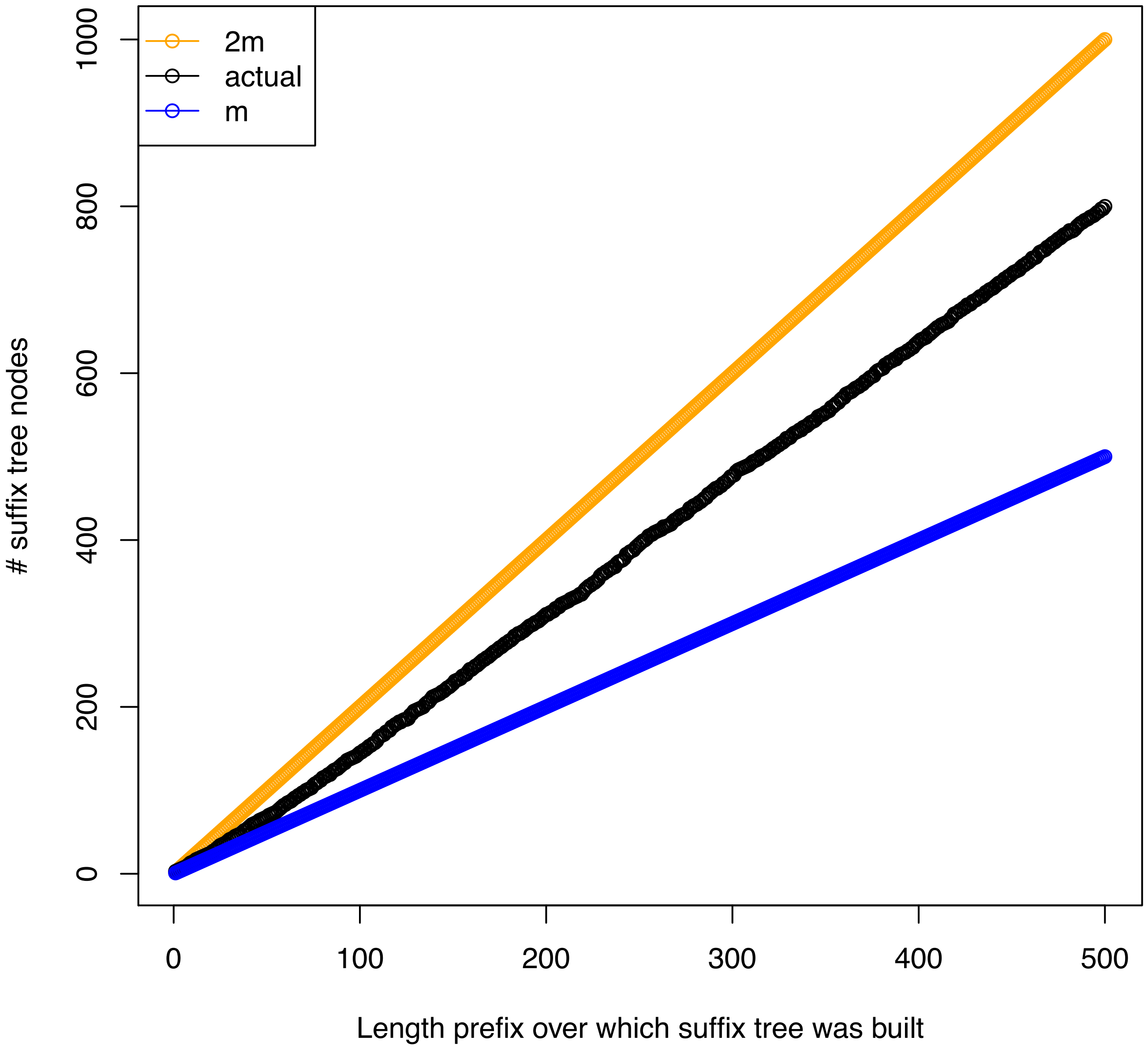
Suffix trie

>100K nodes



Suffix tree

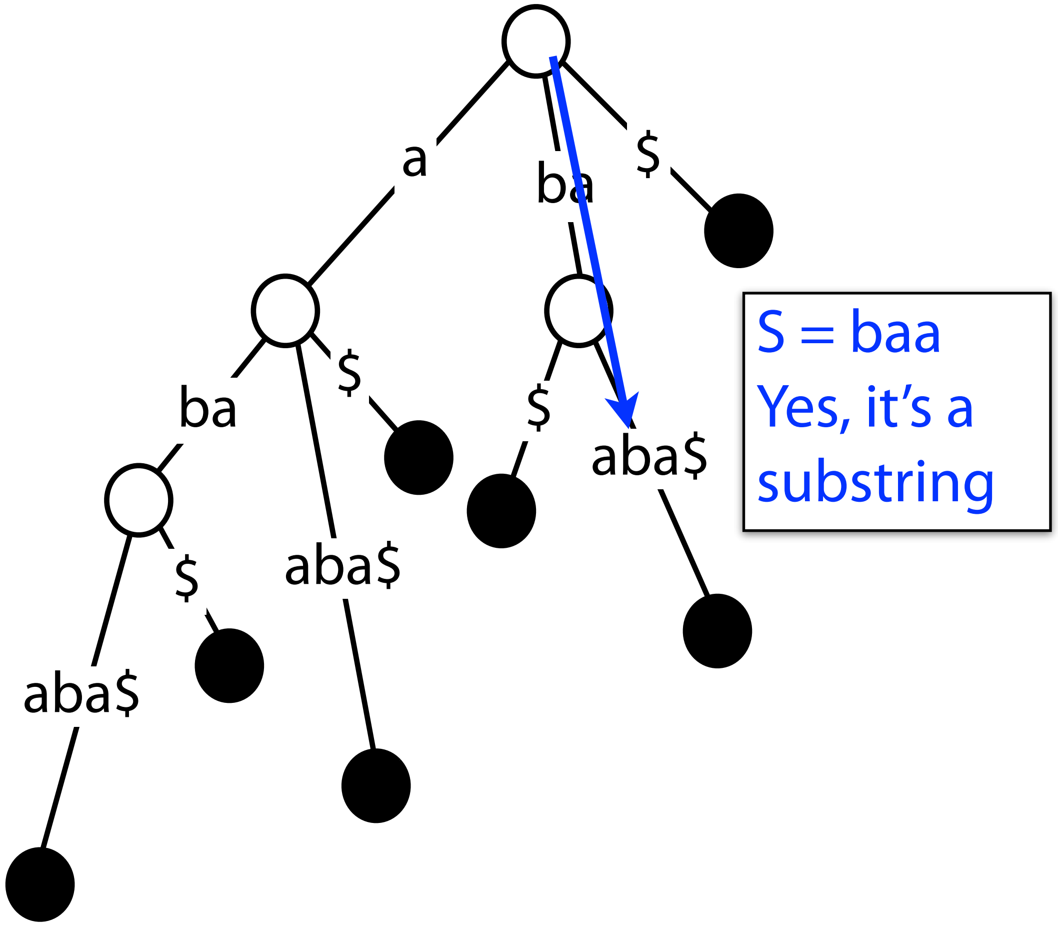
<1K nodes



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

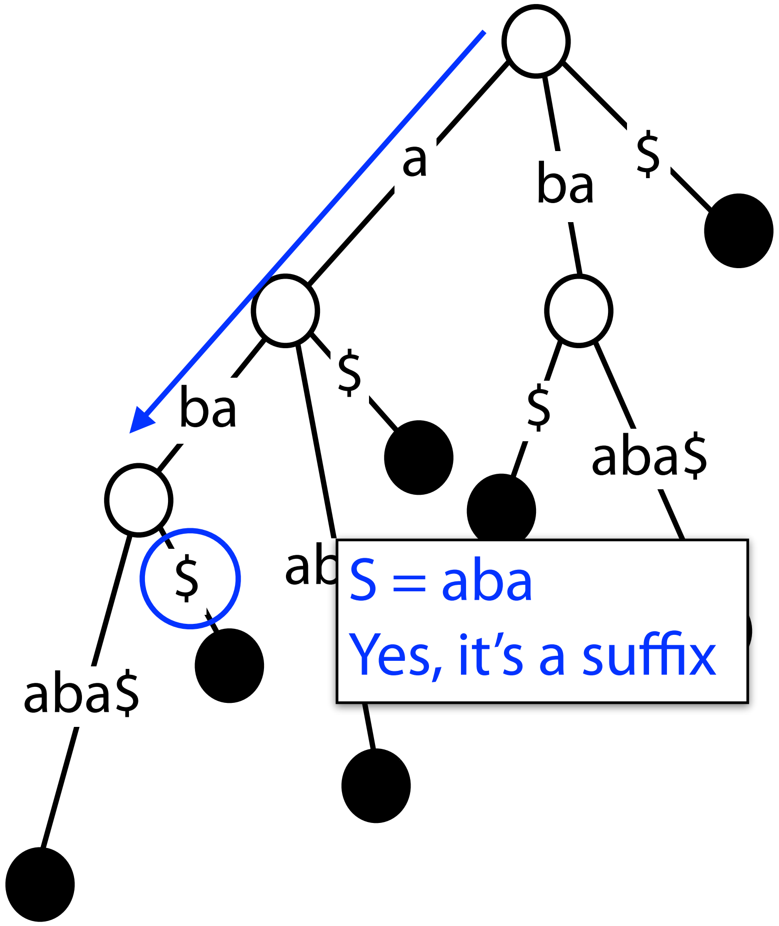


S = baa
Yes, it's a substring

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

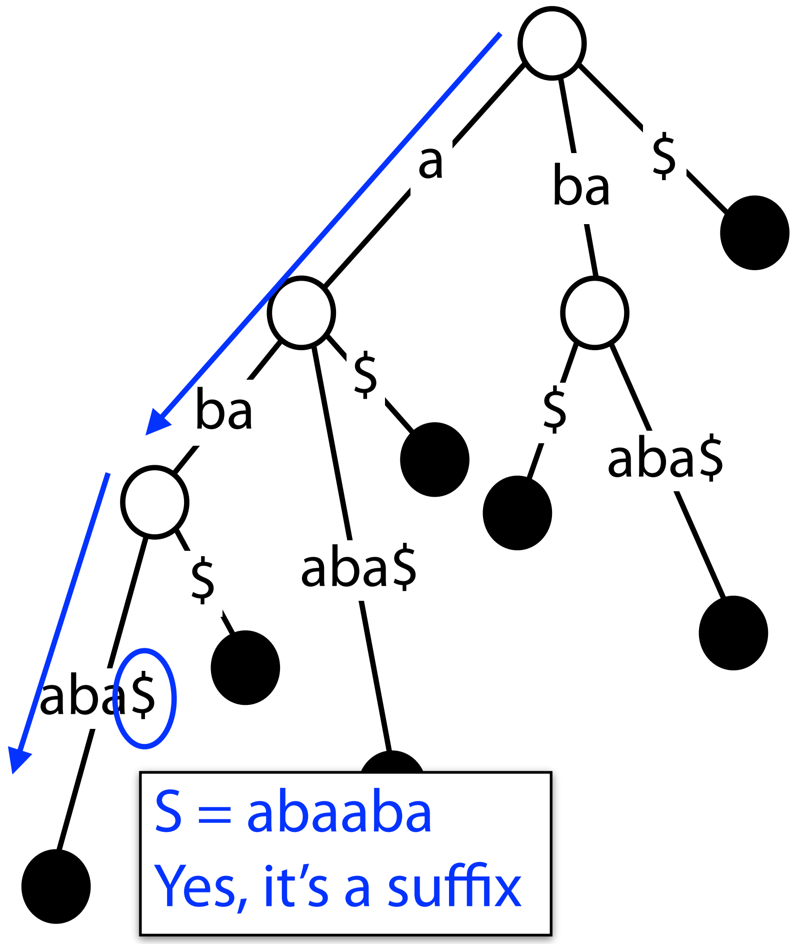


S = aba
Yes, it's a suffix

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

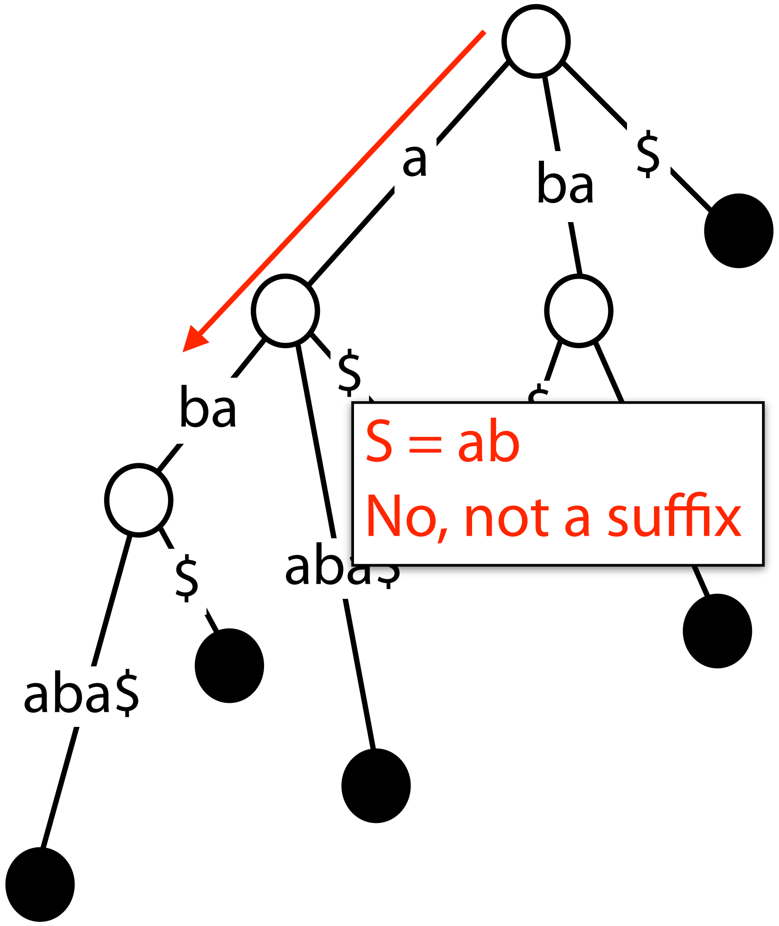


S = abaaba
Yes, it's a suffix

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

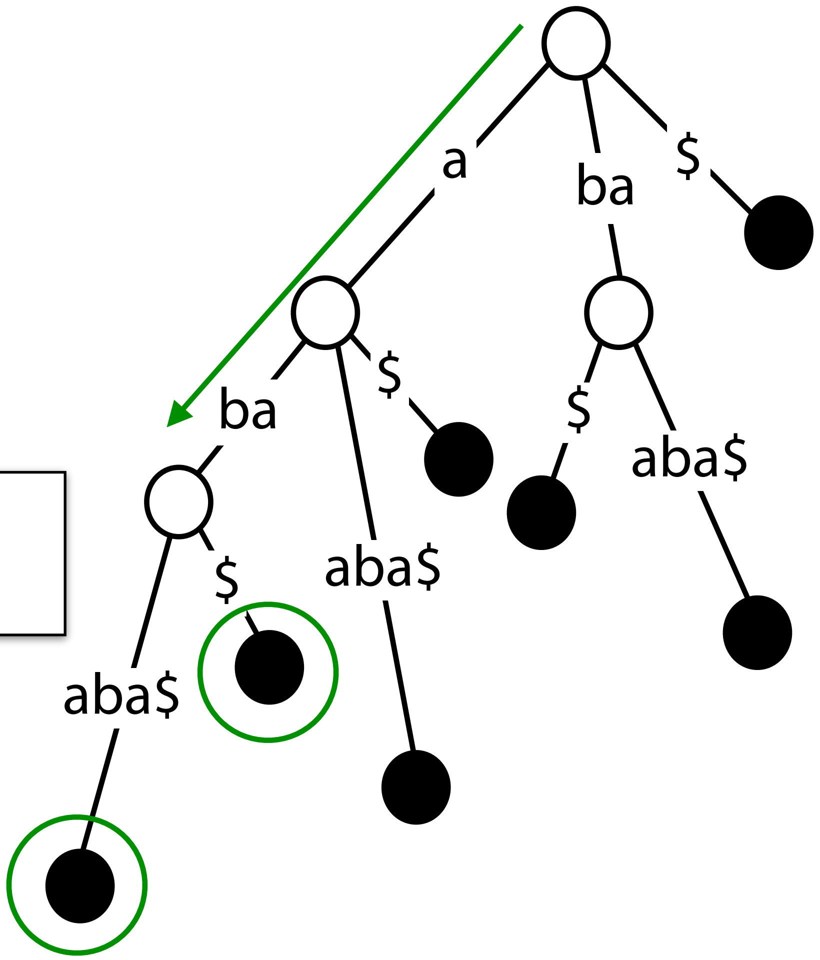


S = ab
No, not a suffix

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



S = aba
Occurs twice

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

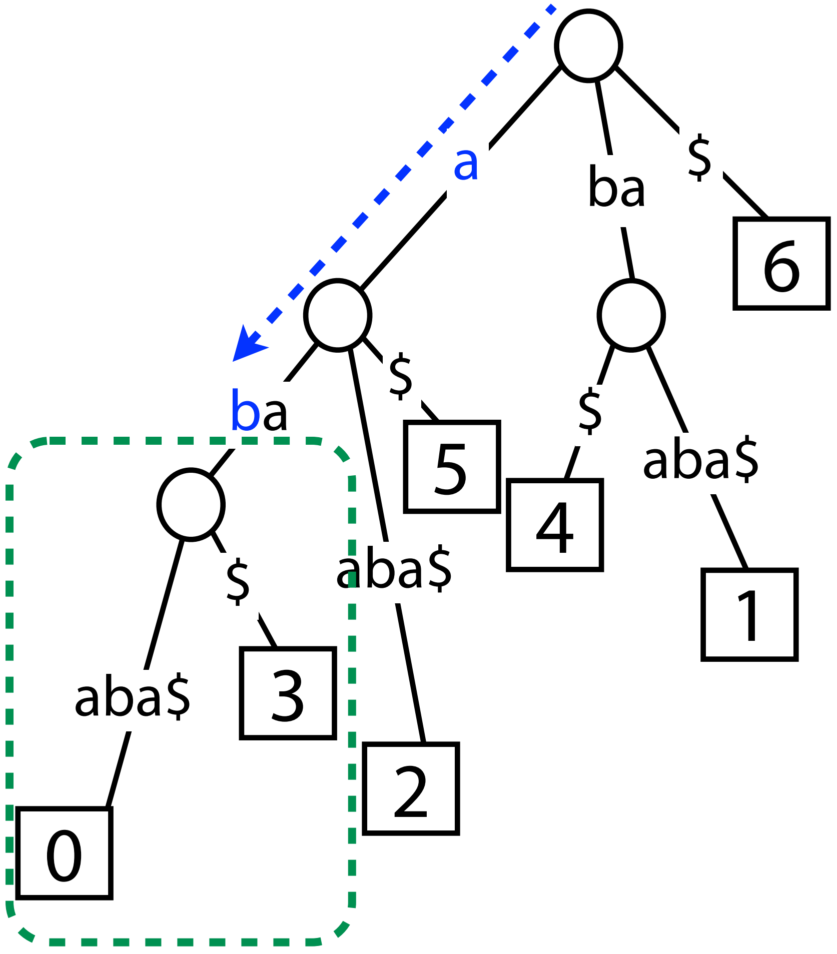
O(*k*) Step 2: visit all leaf nodes below

Report each leaf offset as match offset

# leaves in subtree is is k,

# non-leaves is ≤ k-1

O(*n* + *k*) time overall



abaaba

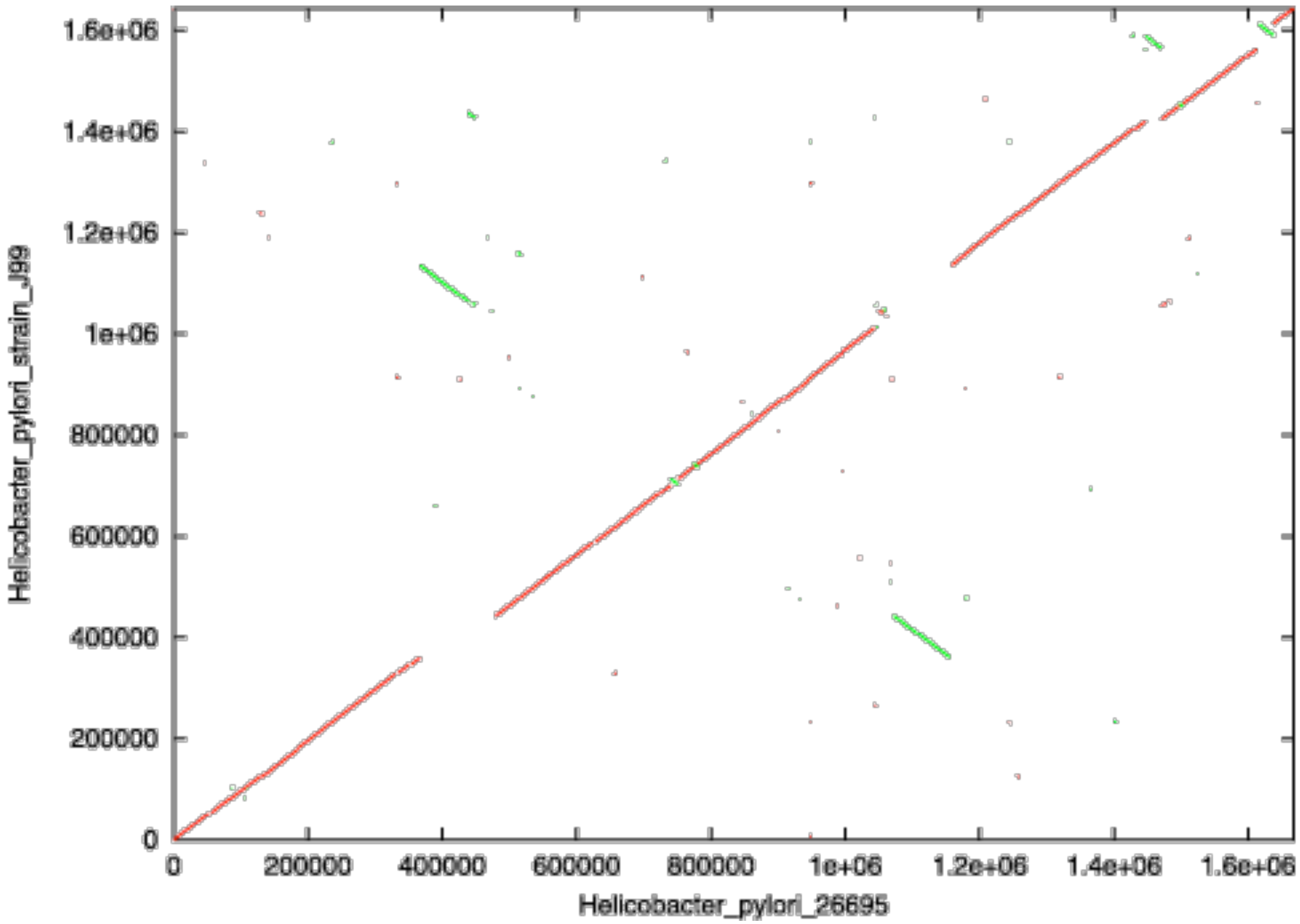
ab ab

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	$O(m)$

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

Suffix tree: long common substrings



Dots are *maximal unique matches* (*MUMs*), a kind of long substring shared by two sequences

Red = match between like strands
green = different strands

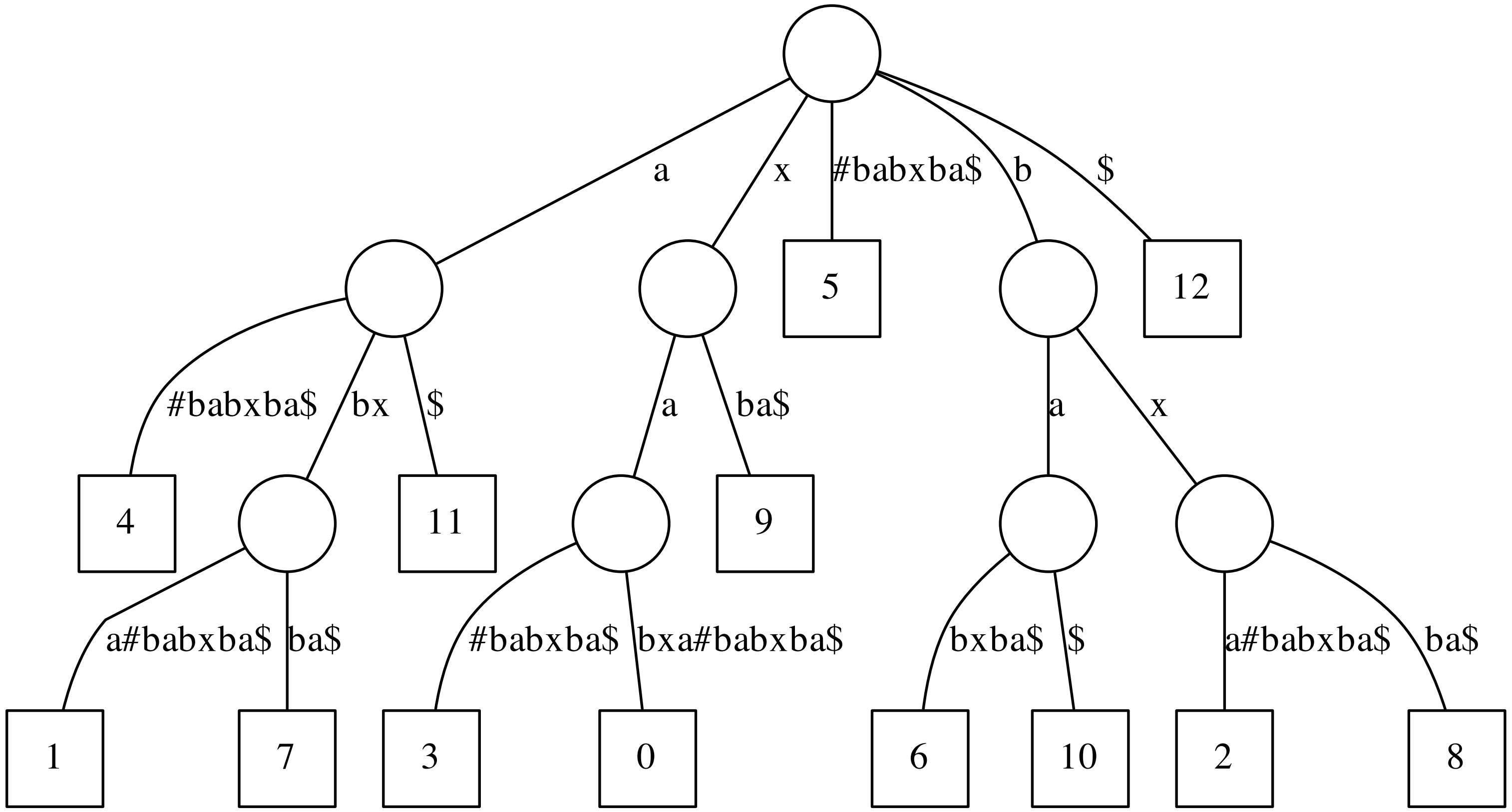
Axes are strains of Helicobacter pylori, bacterium found in stomach & associated with ulcers

Suffix tree application: find longest common substring

Find longest common substring (LCS) of *X* and *Y*, make a new string ***X#Y$*** where #, $ are both terminal symbols. Build a suffix tree for ***X#Y$***.

X = xabxa ***Y*** = babxba

X#***Y***$ = xabxa#babxba$



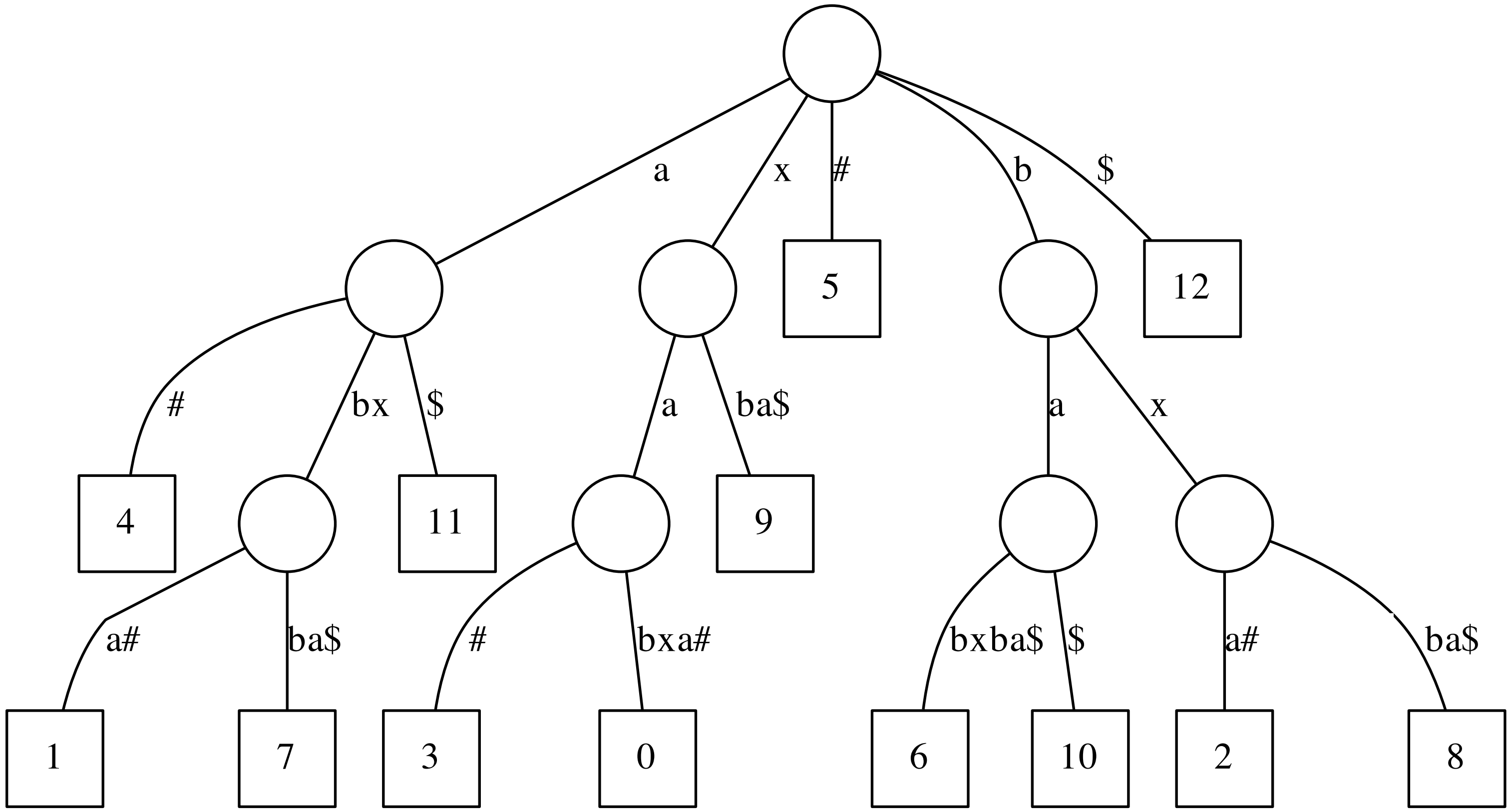
For clarity, if a suffix includes part of both strings, let's hide the portion after the #

Suffix tree application: find longest common substring

Find longest common substring (LCS) of *X* and *Y*, make a new string ***X#Y$*** where #, $ are both terminal symbols. Build a suffix tree for ***X#Y$***.

X = xabxa ***Y*** = babxba

X#Y*$* = xabxa#babxba$



For clarity, if a suffix includes part of both strings, let's hide the portion after the #

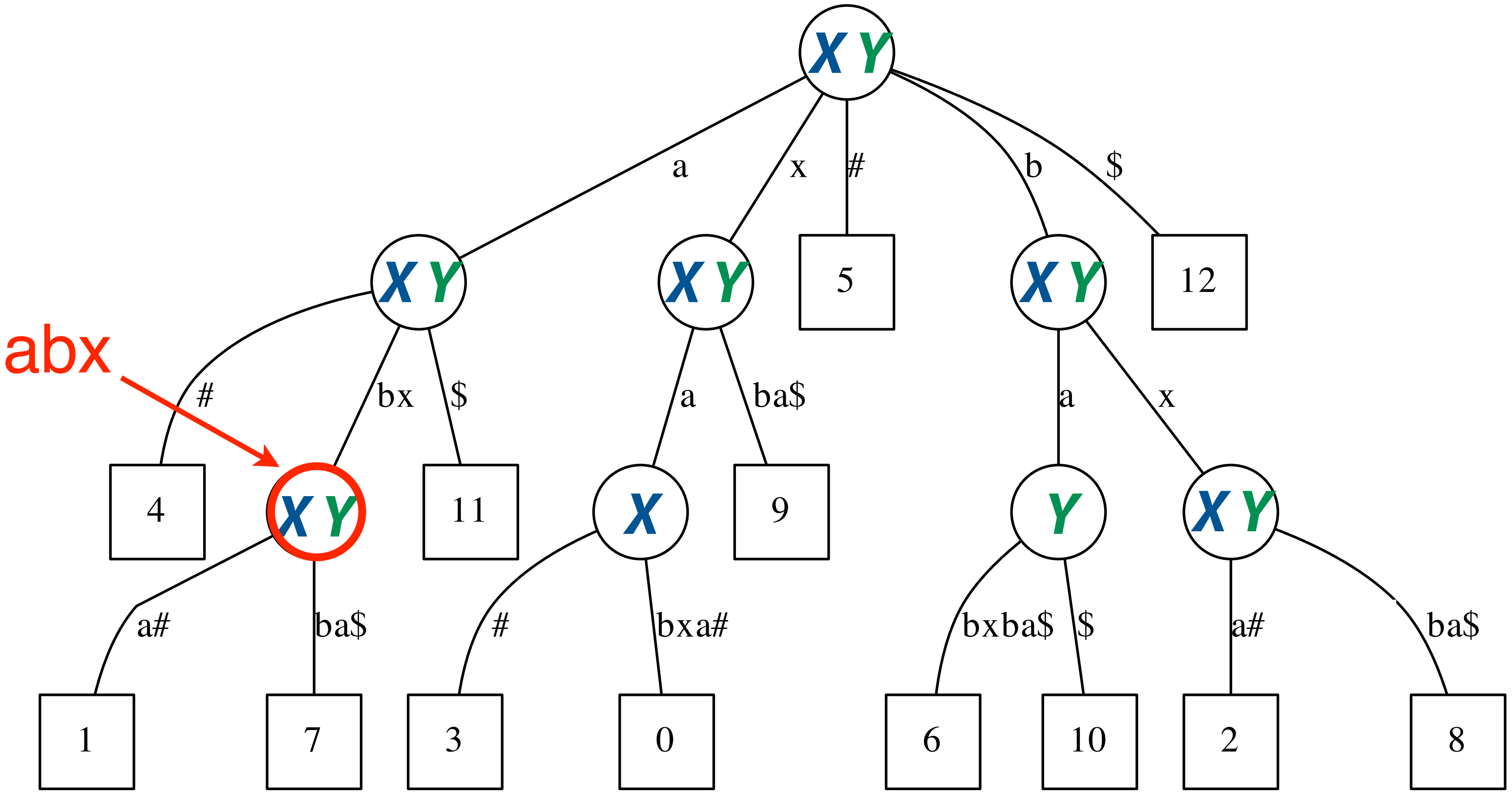
Now suffixes of ***X*** end in # and suffixes of ***Y*** end in $

Suffix tree application: find longest common substring

Find longest common substring (LCS) of *X* and *Y*, make a new string ***X#Y$*** where #, $ are both terminal symbols. Build a suffix tree for ***X#Y$***.

$\boldsymbol{X} = \text{xabxa} \quad \boldsymbol{Y} = \text{babxba}$

$X\#Y\$ = \text{xabxa\#babxba\$}$



Leaves with labels in [0, 5] are suffixes of ***X#***, labels of [6, 12] are suffixes of ***Y$***

Traverse tree, annotating each node according to whether leaves below include suffixes of ***X***, ***Y*** or both

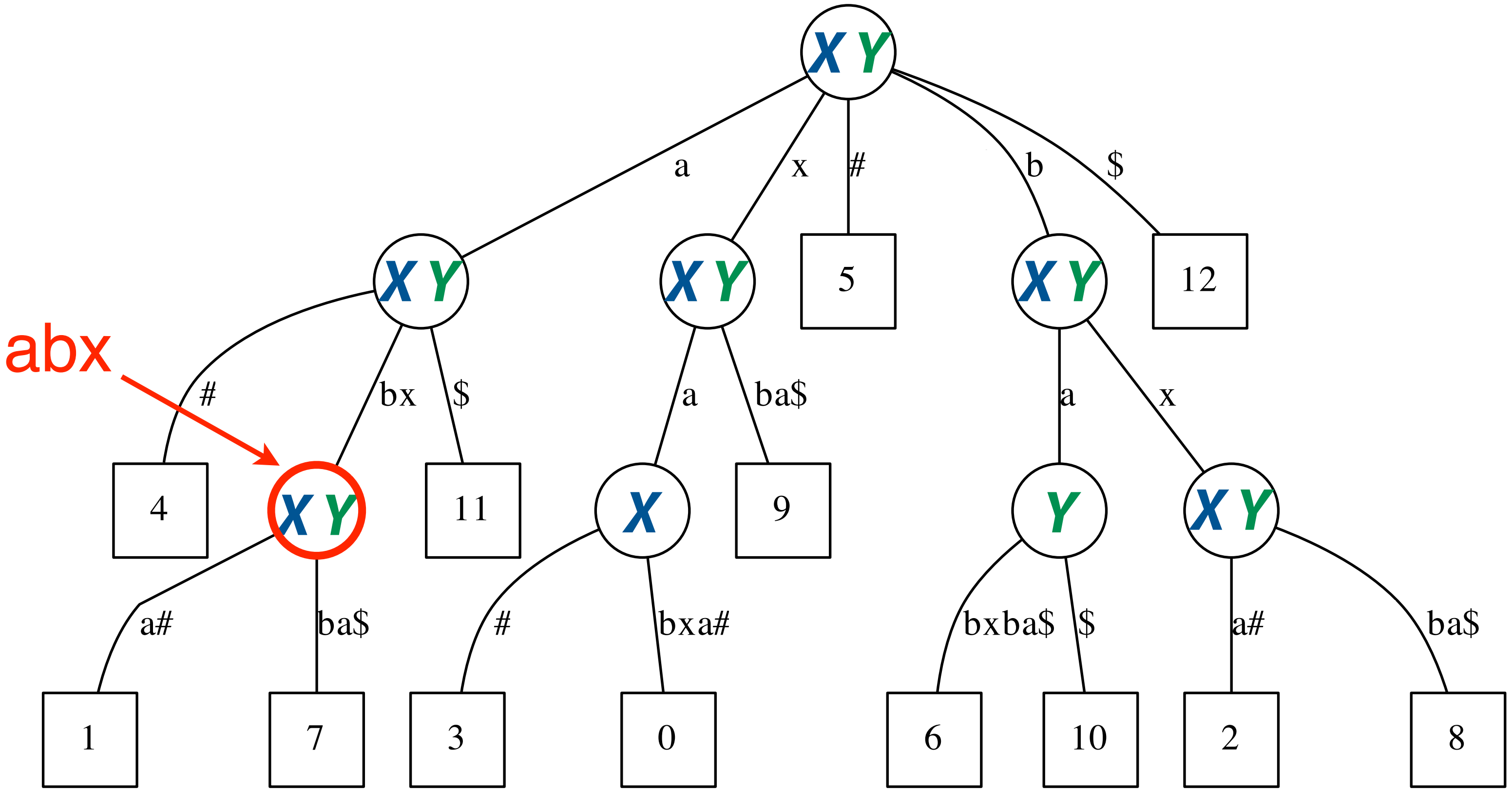
Node w/ greatest label depth annotated ***XY*** corresponds to LCS

$O(|\boldsymbol{X}| + |\boldsymbol{Y}|)$ time and space!

Suffix tree application: generalized suffix trees

It's often useful to build a suffix tree of many strings at once

This is a *generalized suffix tree.* See *Gusfield* 6.4.



Suffix trees in the real world

Alignment of whole genomes (MUMmer):

Delcher, Arthur L., et al. "Alignment of whole genomes." *Nucleic Acids Research* 27.11 (1999): 2369-2376.

Delcher, Arthur L., et al. "Fast algorithms for large-scale genome alignment and comparison." *Nucleic Acids Research* 30.11 (2002): 2478-2483.

Kurtz, Stefan, et al. "Versatile and open software for comparing large genomes." *Genome Biol* 5.2 (2004): R12.

~ 4,000 citations http://mummer.sourceforge.net

Computing and visualizing repeats in whole genomes (REPuter):

Kurtz, Stefan, and Chris Schleiermacher. "REPuter: Fast computation of maximal repeats in complete genomes." *Bioinformatics* 15.5 (1999): 426-427.

Kurtz, Stefan, et al. "REPuter: the manifold applications of repeat analysis on a genomic scale." *Nucleic acids research* 29.22 (2001): 4633-4642.

> 1,000 citations http://bibiserv.techfak.uni-bielefeld.de/reputer

Identifying sequence motifs

Marsan, Laurent, and Marie-France Sagot. "Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification." *Journal of Computational Biology* 7.3-4 (2000): 345-362.

Sagot, Marie. "Spelling approximate repeated or common motifs using a suffix tree." *LATIN'98: Theoretical Informatics* (1998): 374-390.

~ 600 citations

Also used in: multiple alignment

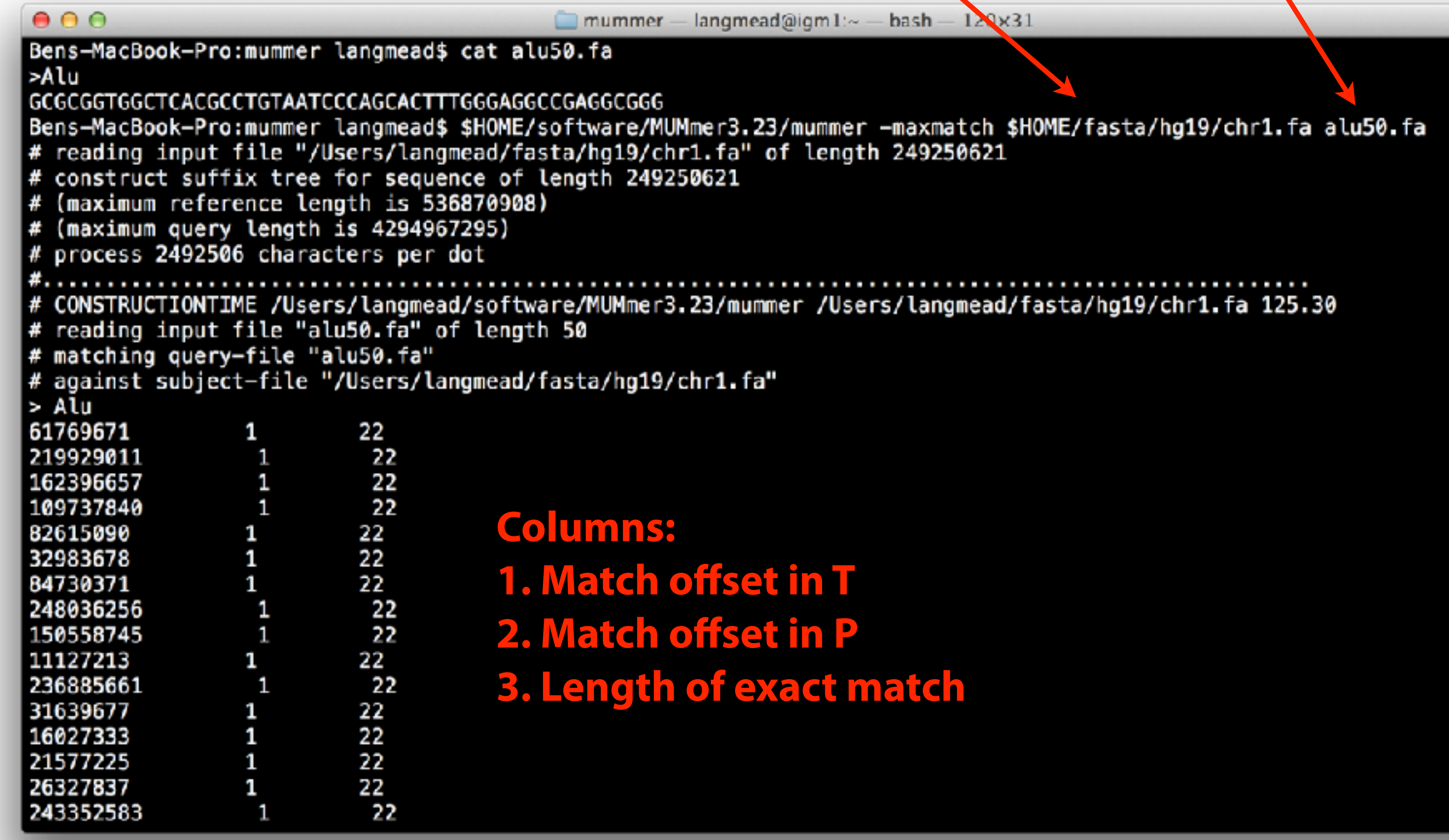
Suffix trees in the real world: MUMmer

FASTA file containing "reference" ("text")

FASTA file containing ALU string

Indexing phase: ~2 minutes

Matching phase: very fast



```
Bens-MacBook-Pro:mummer langmead$ cat alu50.fa
>Alu
GCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGG
Bens-MacBook-Pro:mummer langmead$ $HOME/software/MUMmer3.23/mummer -maxmatch $HOME/fasta/hg19/chr1.fa alu50.fa
# reading input file "/Users/langmead/fasta/hg19/chr1.fa" of length 249250621
# construct suffix tree for sequence of length 249250621
# (maximum reference length is 536870908)
# (maximum query length is 4294967295)
# process 2492506 characters per dot
#....................................................................................................
# CONSTRUCTIONTIME /Users/langmead/software/MUMmer3.23/mummer /Users/langmead/fasta/hg19/chr1.fa 125.30
# reading input file "alu50.fa" of length 50
# matching query-file "alu50.fa"
# against subject-file "/Users/langmead/fasta/hg19/chr1.fa"
> Alu
61769671         1        22
219929011         1        22
162396657         1        22
109737840         1        22
82615090         1        22
32983678         1        22
84730371         1        22
248036256         1        22
150558745         1        22
11127213         1        22
236885661         1        22
31639677         1        22
16027333         1        22
21577225         1        22
26327837         1        22
243352583         1        22
```

Columns:

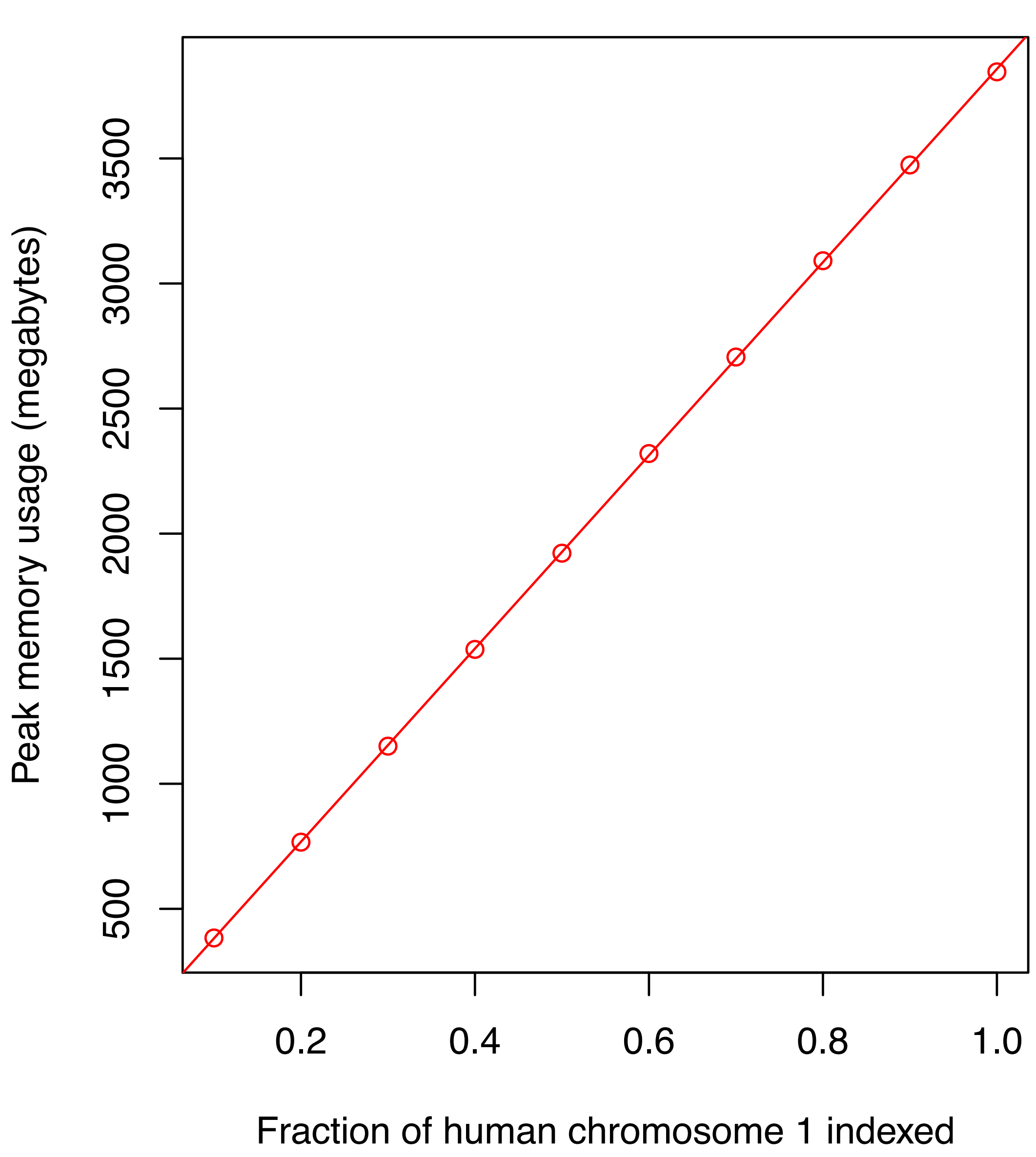
1. Match offset in T

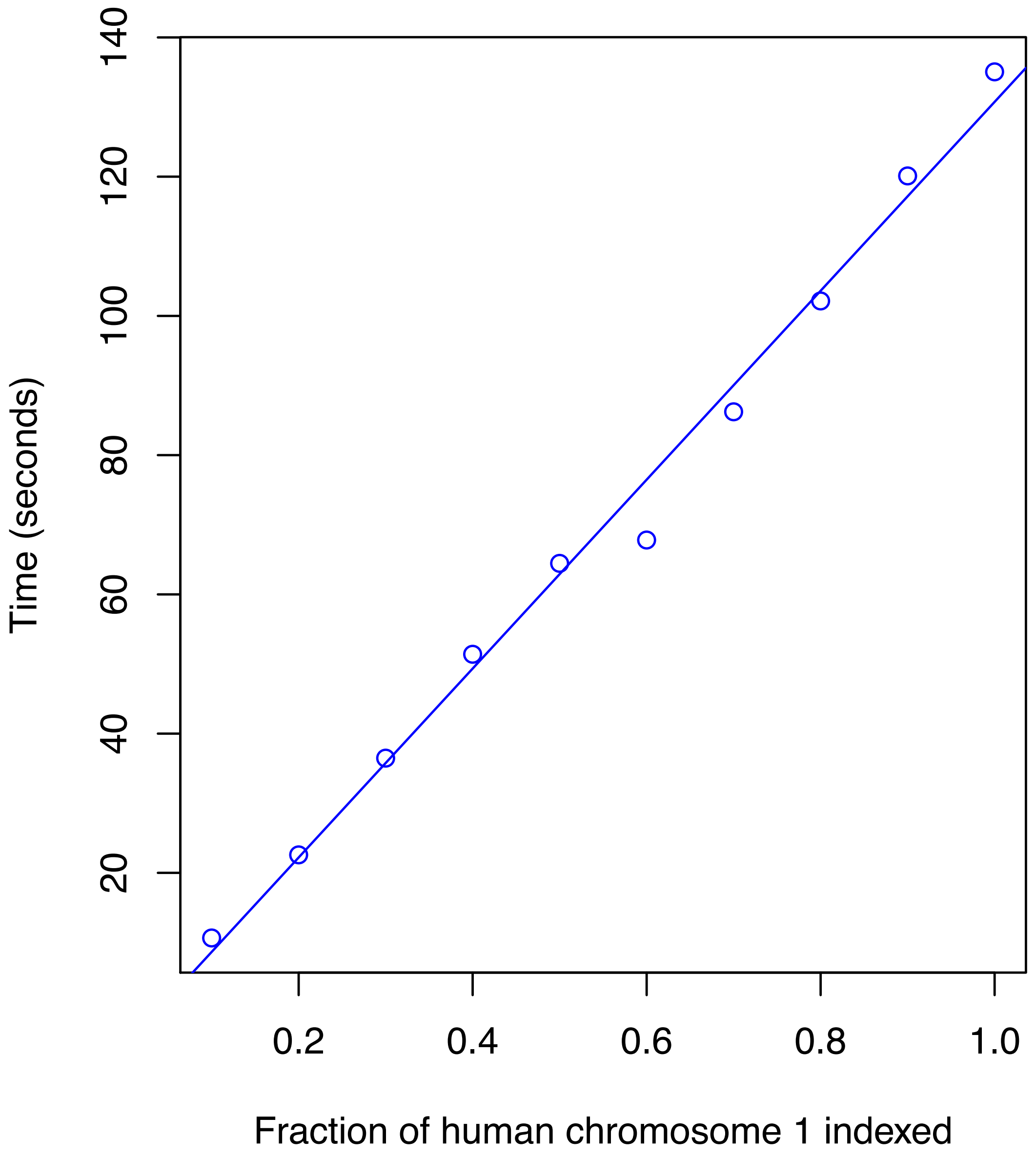
2. Match offset in P

3. Length of exact match

Suffix trees in the real world: MUMmer

MUMmer v3.32 time and memory scaling when indexing increasingly larger fractions of human chromosome 1





For whole chromosome 1, took 2m:14s and used 3.94 GB memory

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 ≈ **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