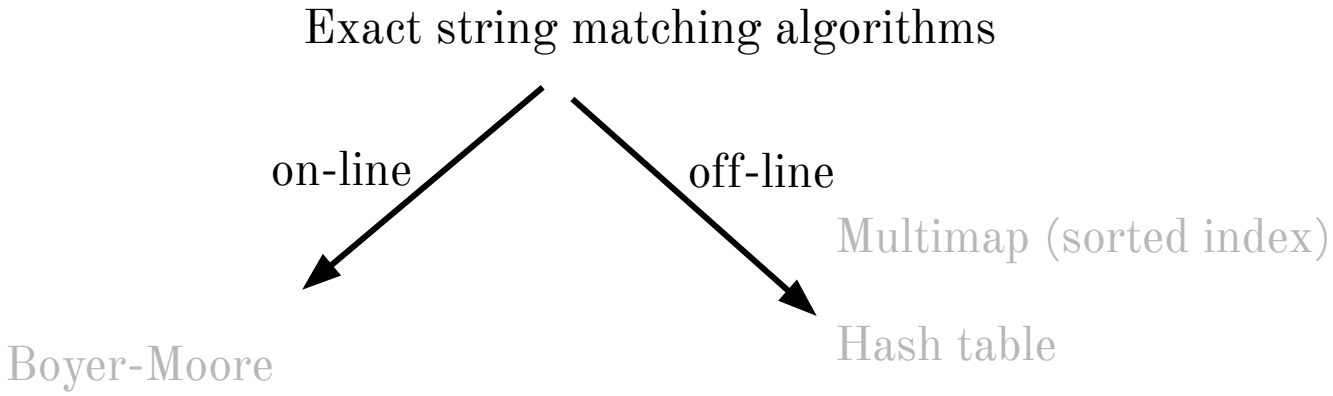
Lesson 06 Genome Informatics

Suffix trie
Suffix Tree
Unix commands in bioinformatics

Recapitulation



Boyer-Moore

Multimap (sorted index)

Hash table

Suffix array

Suffix trie

Suffix tree

Burrows Wheeler FM index

Tries

A trie (pronounced “try”) is a tree representing a collection of strings with one node per common prefix

Smallest tree such that:

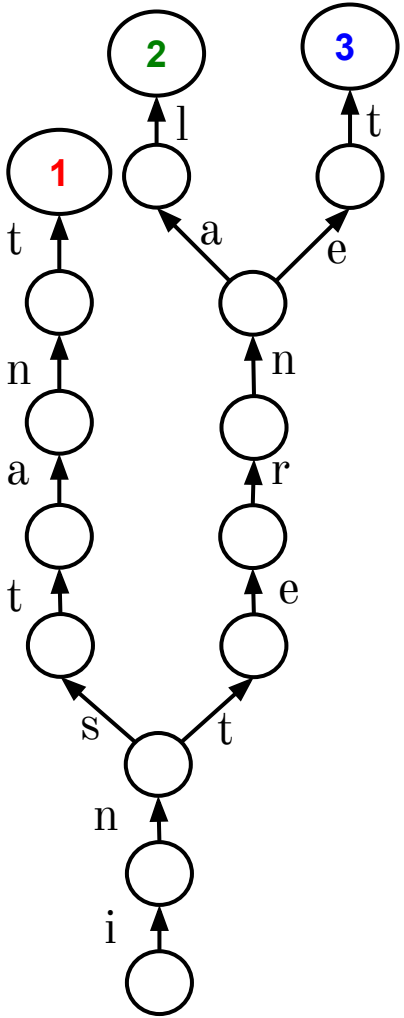
- Each **edge** is labeled with a character $c \in \Sigma$
- A **node** has at most one outgoing edge labeled c, for $c \in \Sigma$
- Each key is “spelled out” along some path starting at the root

Natural way to represent either a set or a map where keys are strings

Tries: example

Represent this map with a trie:

key	value
instant	1
internal	2
internet	3

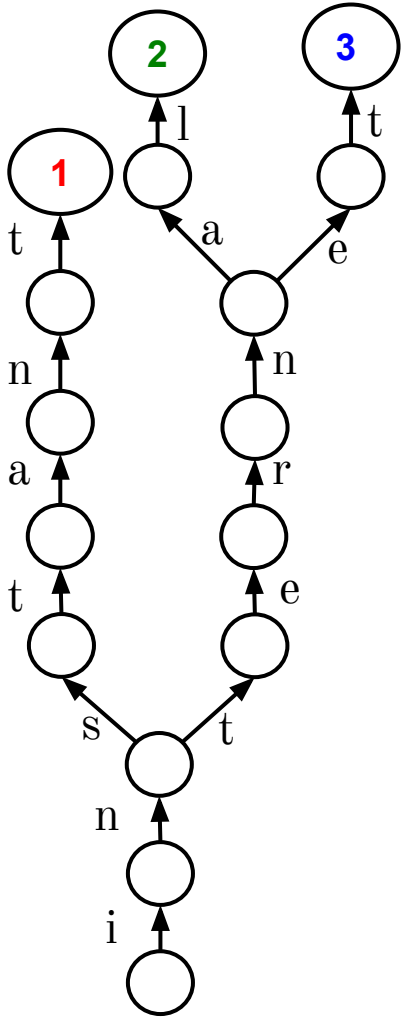


- Each **edge** is labeled with a character c ∈ Σ
- A **node** has at most one outgoing edge labeled c, for c ∈ Σ
- Each key is “spelled out” along some path starting at the root

Tries: example

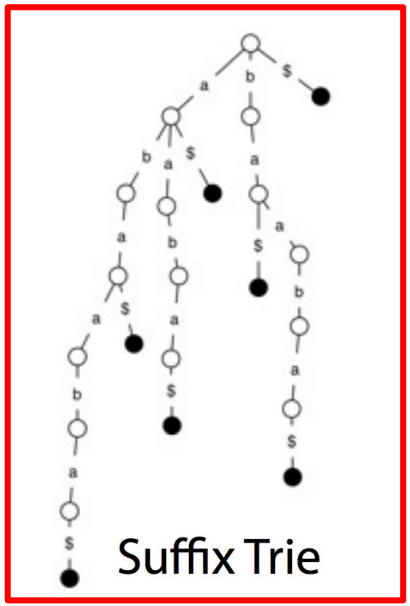
- Checking for presence of a key P, where n = | P |, is O(n) time
- If total length of all keys is N, trie has O(N) nodes

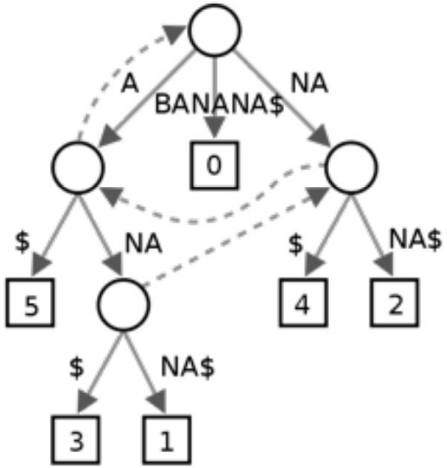
Tries
implementation



Indexing with suffixes

- Until now, our indexes have been based on extracting substrings from T
  A very different approach is to extract suffixes from T. This will lead us to some interesting and practical index data structures





6	$
5	A$
3	ANA$
1	ANANA$
0	BANANA$
4	NA$
2	NANA$

Suffix Array

$ B A N A N A
A $ B A N A N
A N A $ B A N
A N A N A $ B
B A N A N A $
N A $ B A N A
N A N A $ B A

FM Index

Suffix Trie

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

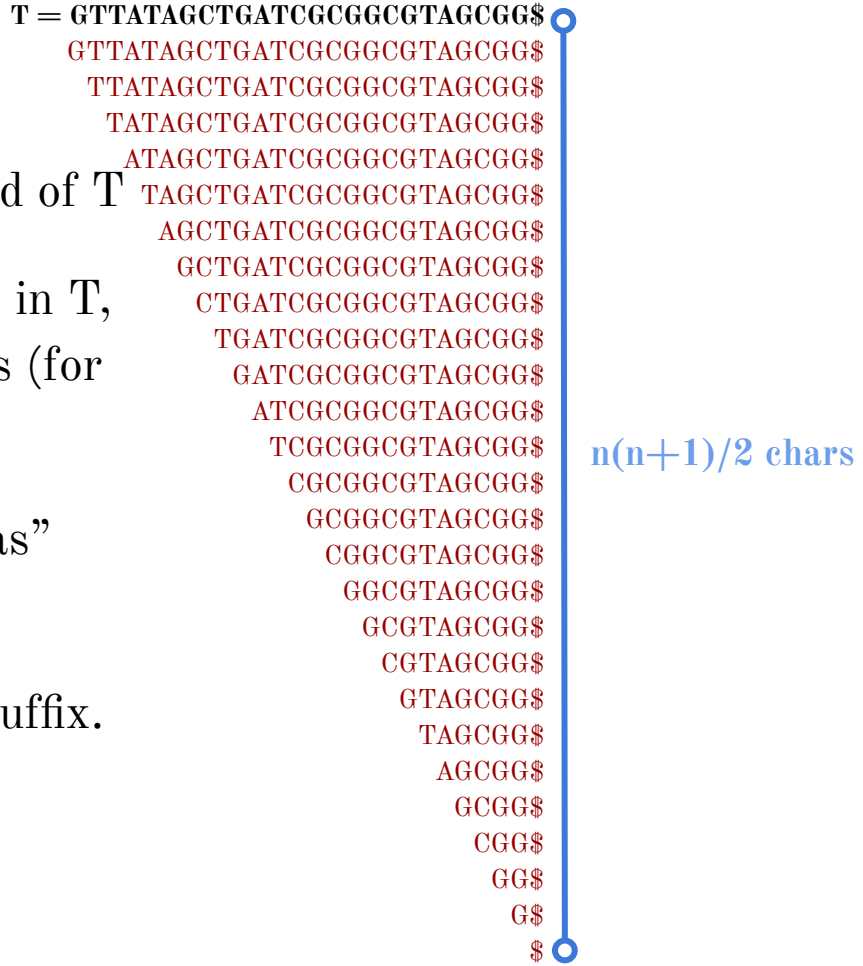
Suffix Trie

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 (for DNA: $ < A < C < G < T)

$ enforces a rule we're all used to using: e.g. "as" comes before "ash" in the dictionary.

$ guarantees no suffix is a prefix of any other suffix.



Tries

Smallest tree such that:

Each **edge** is labeled with a character from Σ

A **node** has at most one outgoing edge

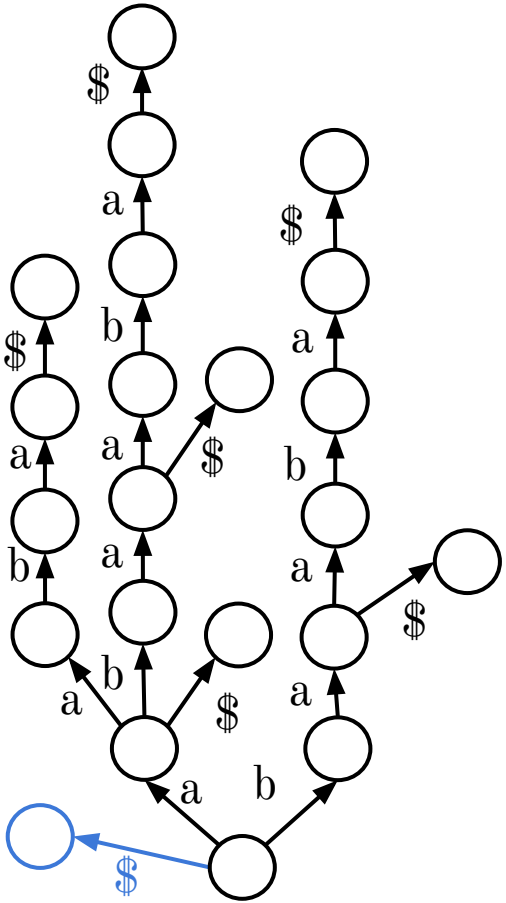
Each key is “spelled out” along some path starting at the root

Suffix trie

T: abaaba 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 $?



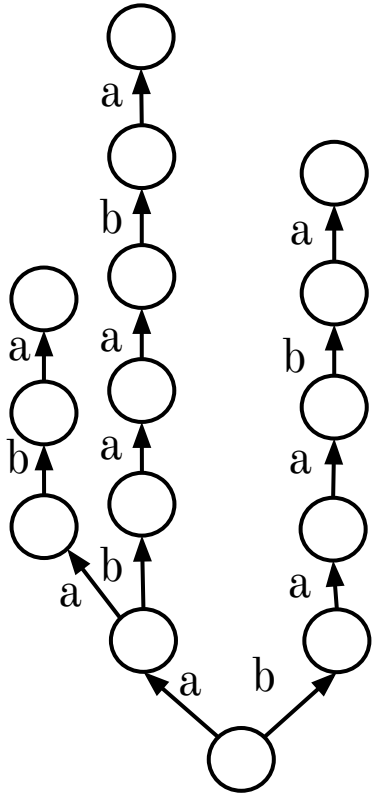
Suffix trie

T: abaaba T$: abaaba$

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

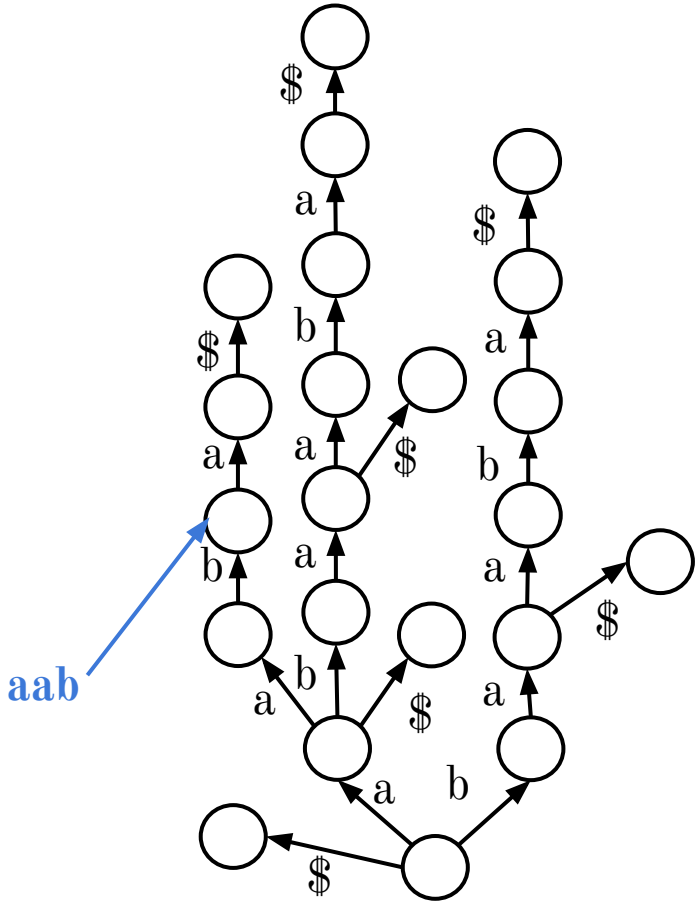
Would this still be the case if we hadn't added $?

No. Lost suffixes: "aba", "ba", etc.



Suffix trie

We can think of nodes as having labels, where the label spells out characters on the path from the root to the node



Suffix trie

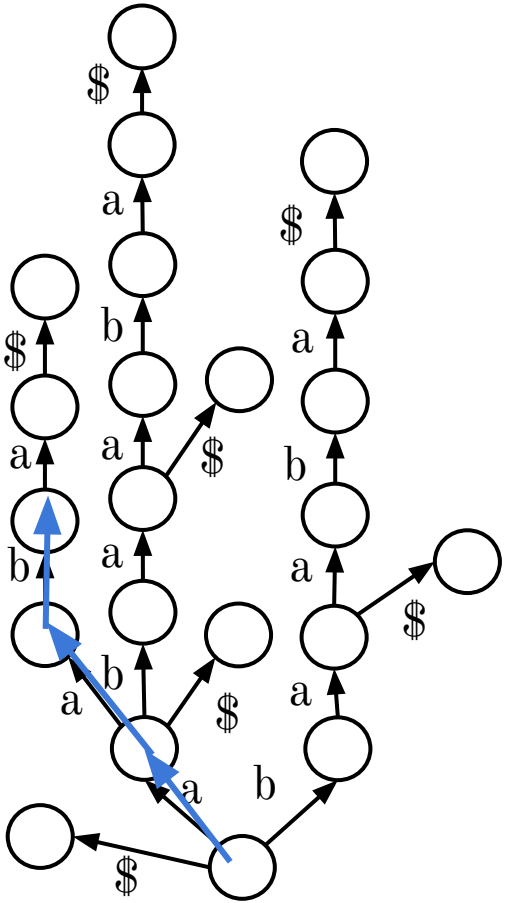
How do we check whether a string S is a substring of T?

Note: Each of T's substrings is spelled out along a path from the root. I.e., every substring is a prefix of some suffix of T.

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

1. If we "fall off" the trie (there is no outgoing edge for next character of S) then S is not a substring of T
2. If we exhaust S without falling off, S is a substring of T

YES! "aab" is a substring of T



Suffix trie

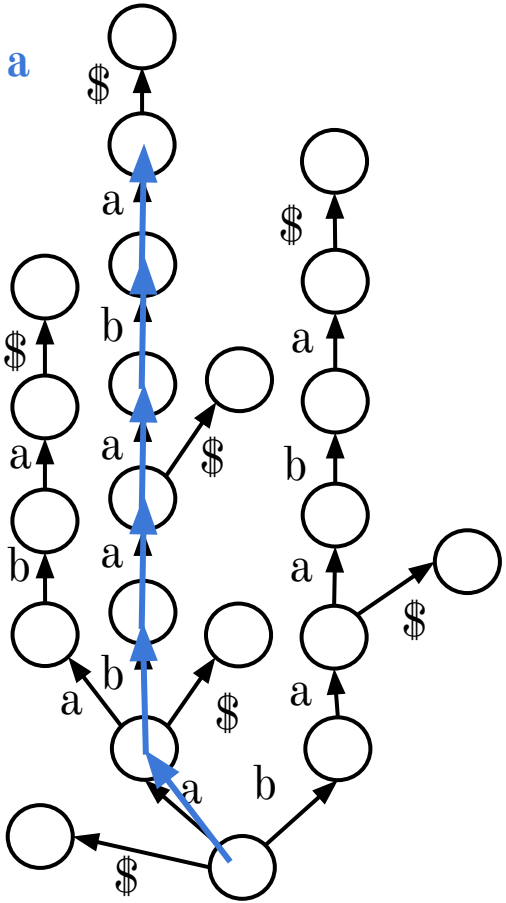
YES! “abaaba” is a substring of T

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

Note: Each of T’s substrings is spelled out along a path from the root. I.e., every substring is a prefix of some suffix of T.

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

1. If we “fall off” the trie (there is no outgoing edge for next character of S) then S is not a substring of T
2. If we exhaust S without falling off, S is a substring of T



Suffix trie

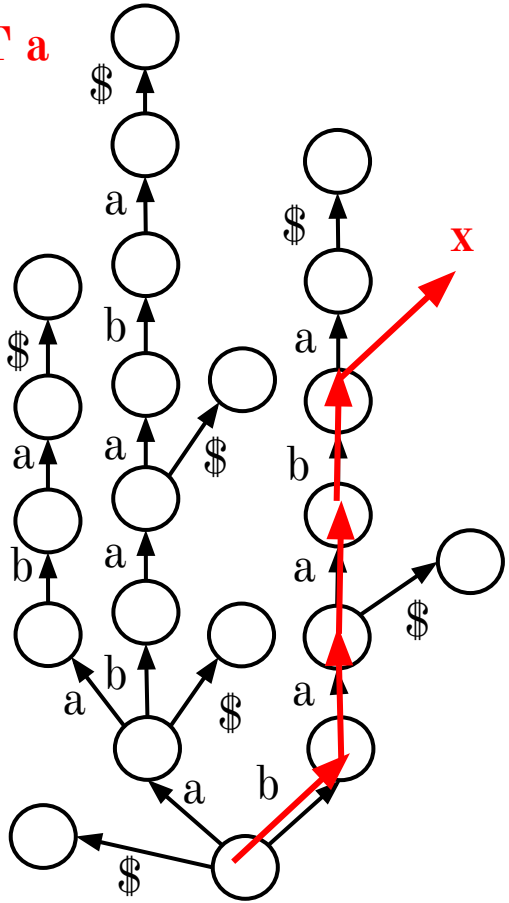
“baabb” is NOT a substring of T

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

Note: Each of T’s substrings is spelled out along a path from the root. I.e., every substring is a prefix of some suffix of T.

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

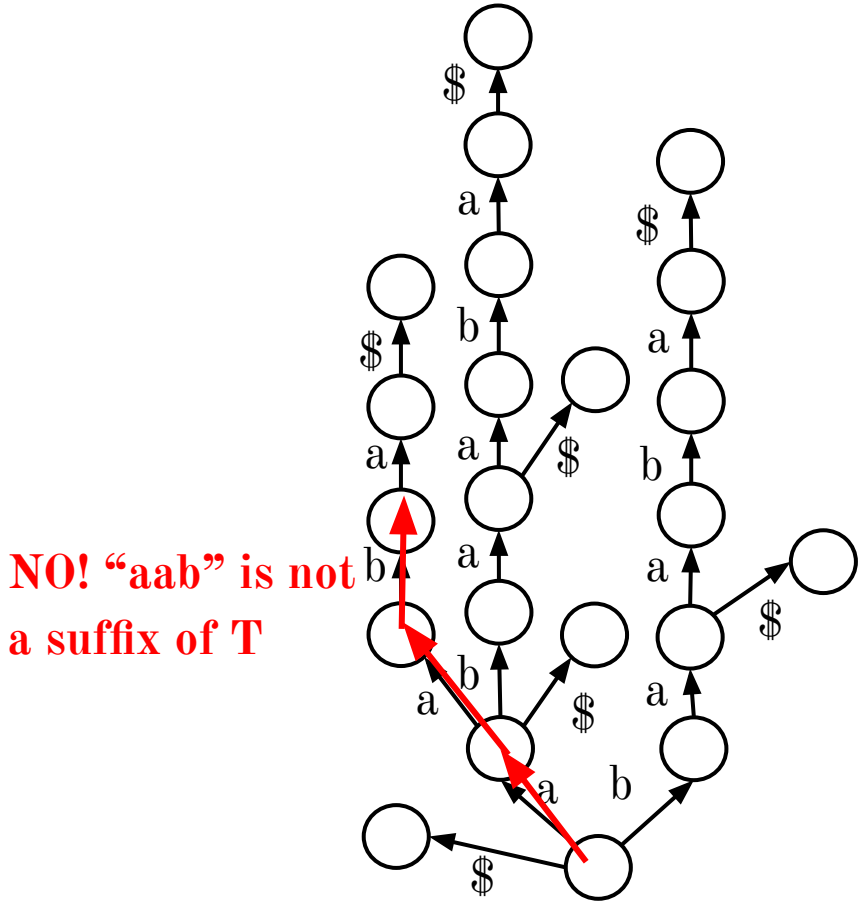
1. If we “fall off” the trie (there is no outgoing edge for next character of S) then S is not a substring of T
2. If we exhaust S without falling off, S is a substring of T



Suffix trie

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

Same procedure as for substring, but additionally check whether the **final node** in the walk has an outgoing edge labeled $



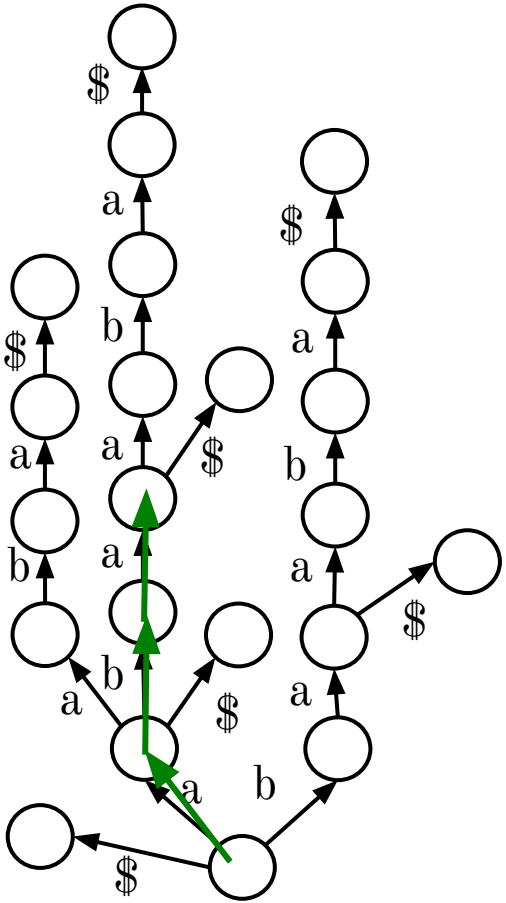
NO! “aab” is not a suffix of T

Suffix trie

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

Same procedure as for substring, but additionally check whether the **final node** in the walk has an outgoing edge labeled $

YES! “aba” is a suffix of T



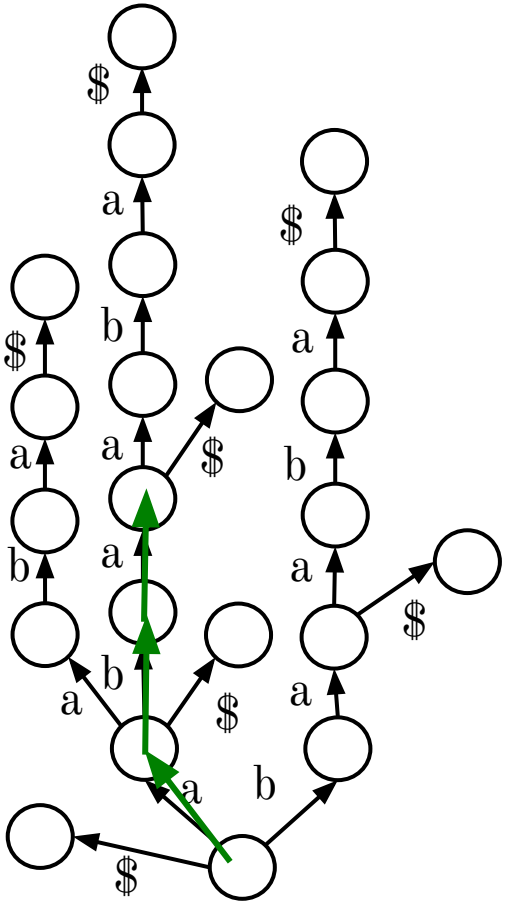
Suffix trie

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

Follow path corresponding to S. Either we fall off, in which case answer is 0, or we end up at node n and the answer = # of leaf nodes in the subtree rooted at n.

Leaves can be counted with depth-first traversal.

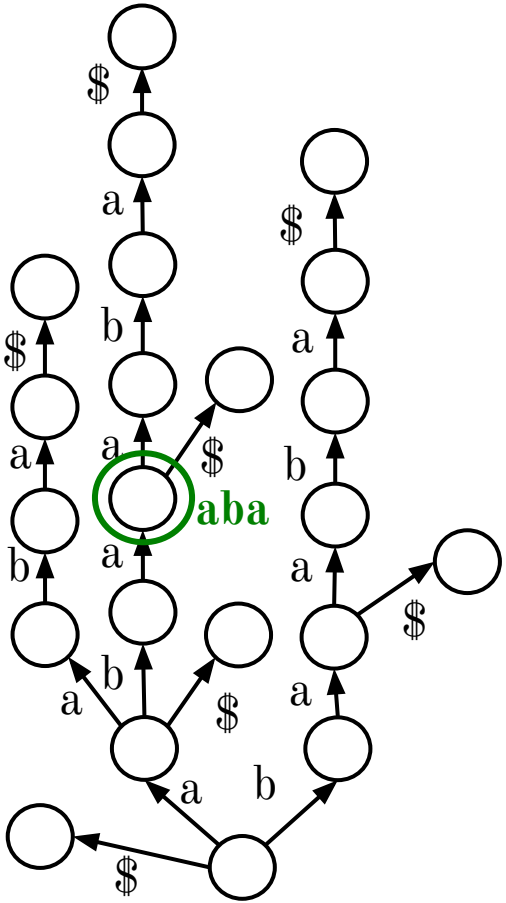
“aba” - 2 occurrences



Suffix trie

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

Find the deepest node with more than one child

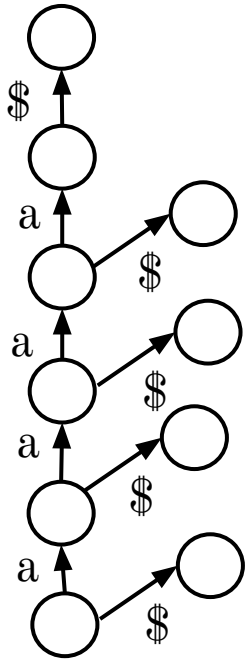


Suffix trie

How many nodes does the suffix trie have?

Is there a class of string where the number of suffix trie nodes grows linearly with m?
Yes: e.g. a string of m a's in a row

T = aaaa



Total nodes:
1 Root
m nodes with incoming **a** edge
m + 1 nodes with incoming $ edge
Total: 2m + 2 nodes

Suffix trie

Is there a class of string where the number of suffix trie nodes grows with $O(m^2)$?
Yes: $a^n b^n$

Total nodes:
- 1 root
- n nodes along “b chain,” right
- n nodes along “a chain,” middle
- n chains of n “b” nodes hanging off each“a chain” node
- $2n + 1$ $ leaves (not shown)

Total: $n^2 + 4n + 2$ nodes, where $m = 2n$

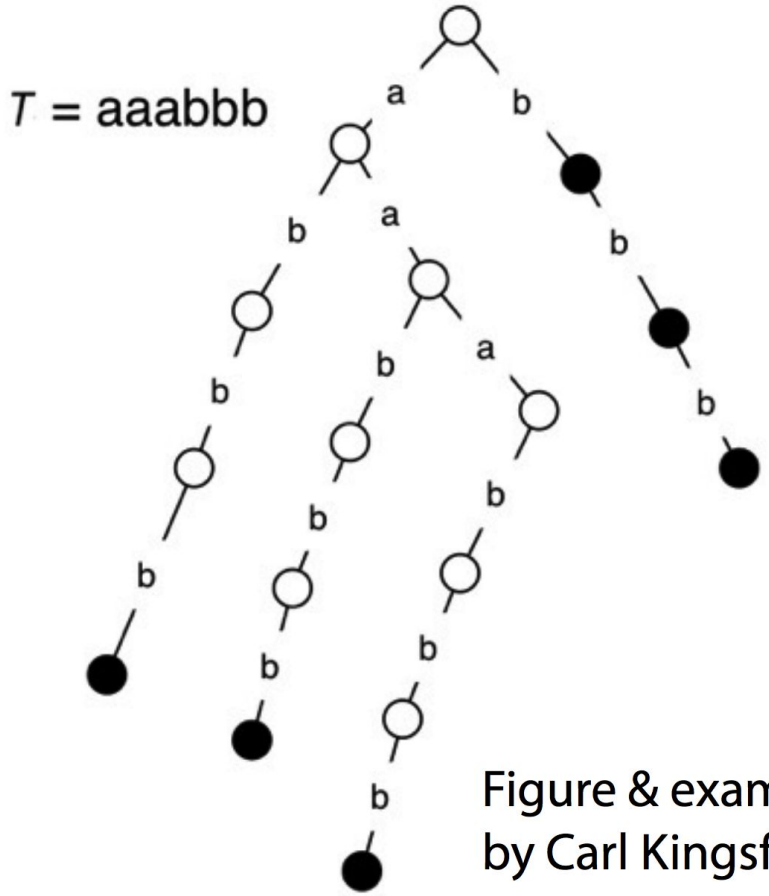


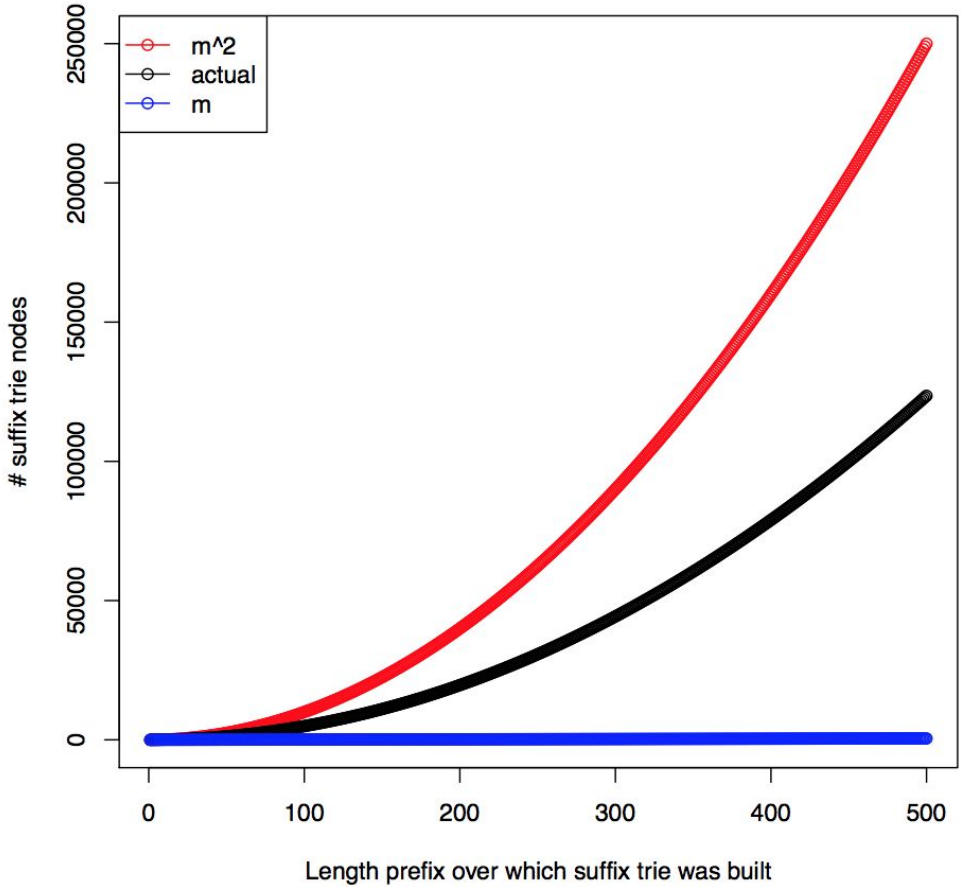
Figure & example by Carl Kingsford

Suffix trie

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

Black curve shows how # nodes increases with prefix length

Suffix trie implementation

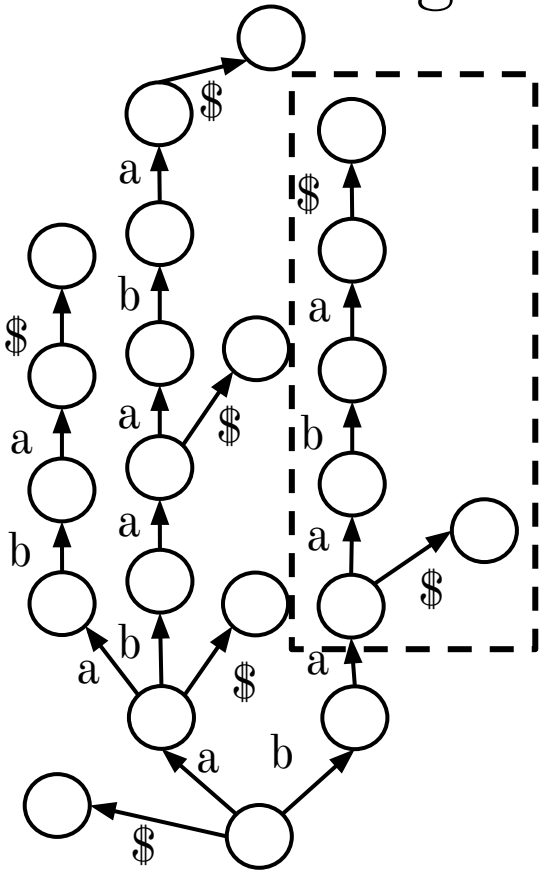


Suffix Tree

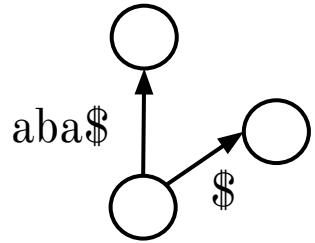
Lesson 6.2

Suffix trie: making it smaller

T: abaaba$

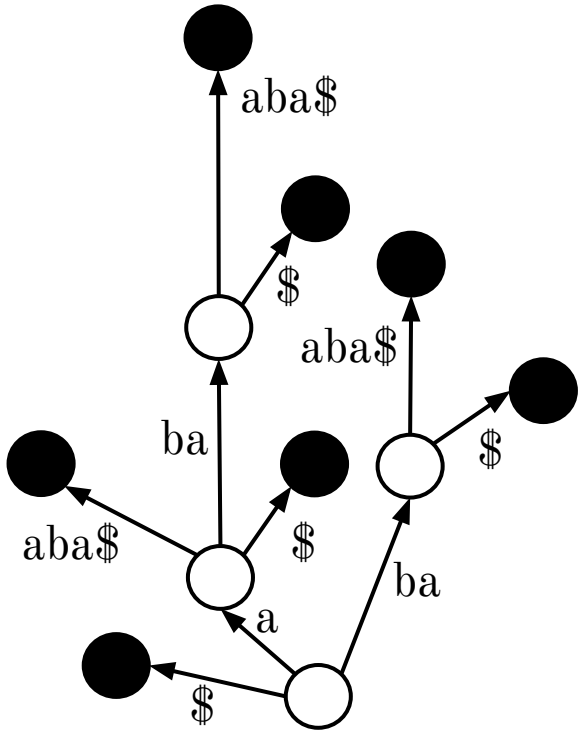


Idea 1: Coalesce non-branching paths into a single edge with a string label



Reduces # nodes, edges, guarantees internal nodes have >1 child

Suffix trie: making it smaller

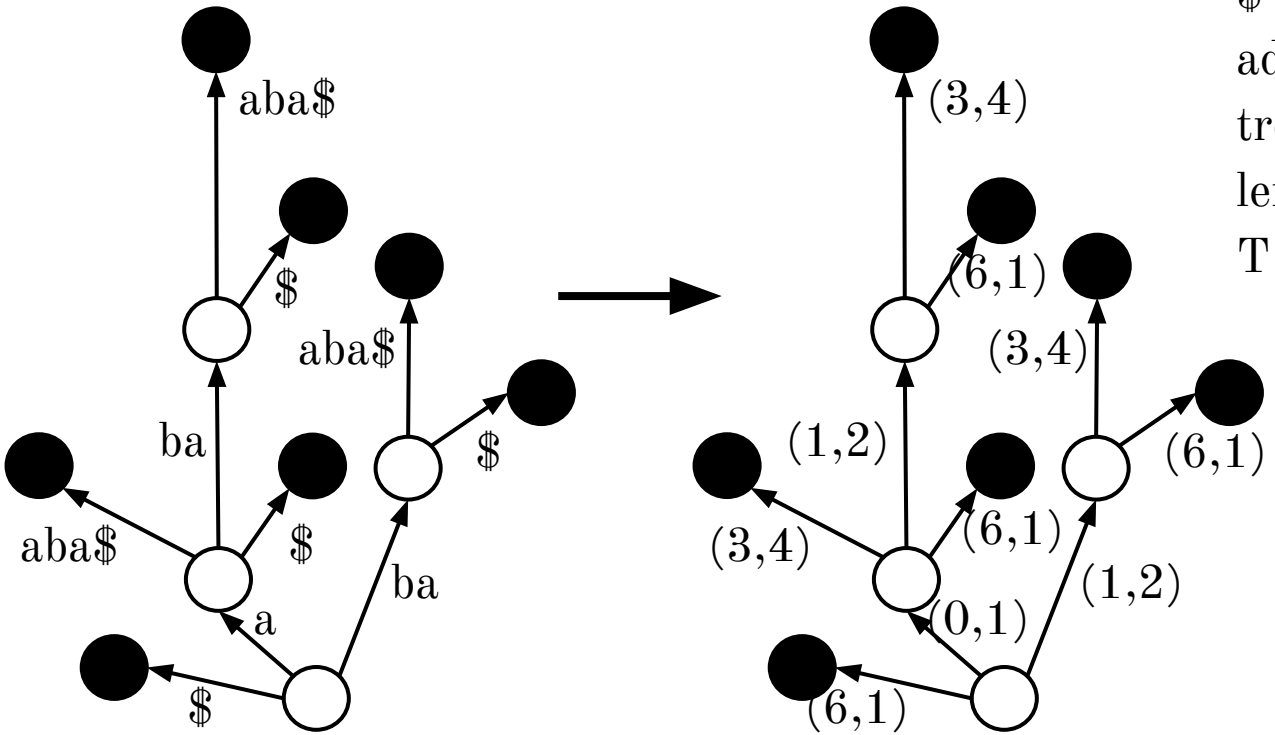


How many leaves? M

How many non-leaf nodes? $\leq$ m - 1

$\leq$ 2m -1 nodes total - O(m) nodes

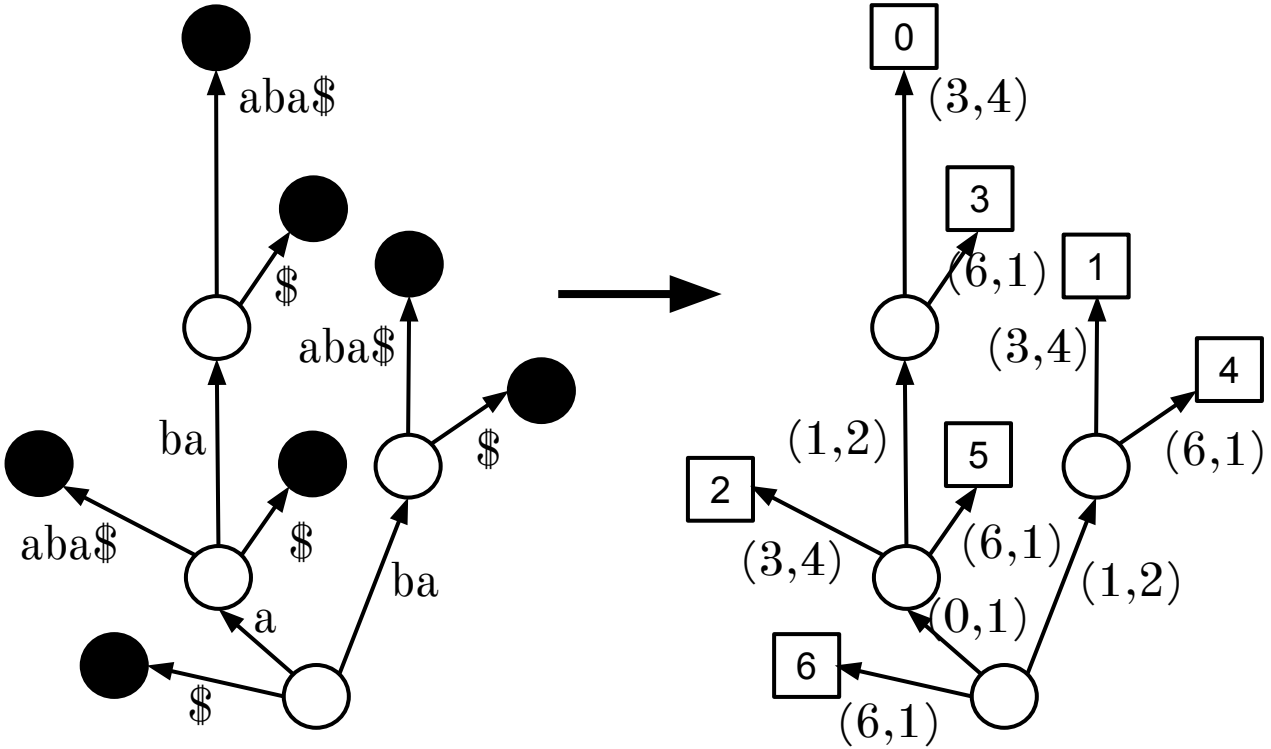
Suffix trie: making it smaller



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

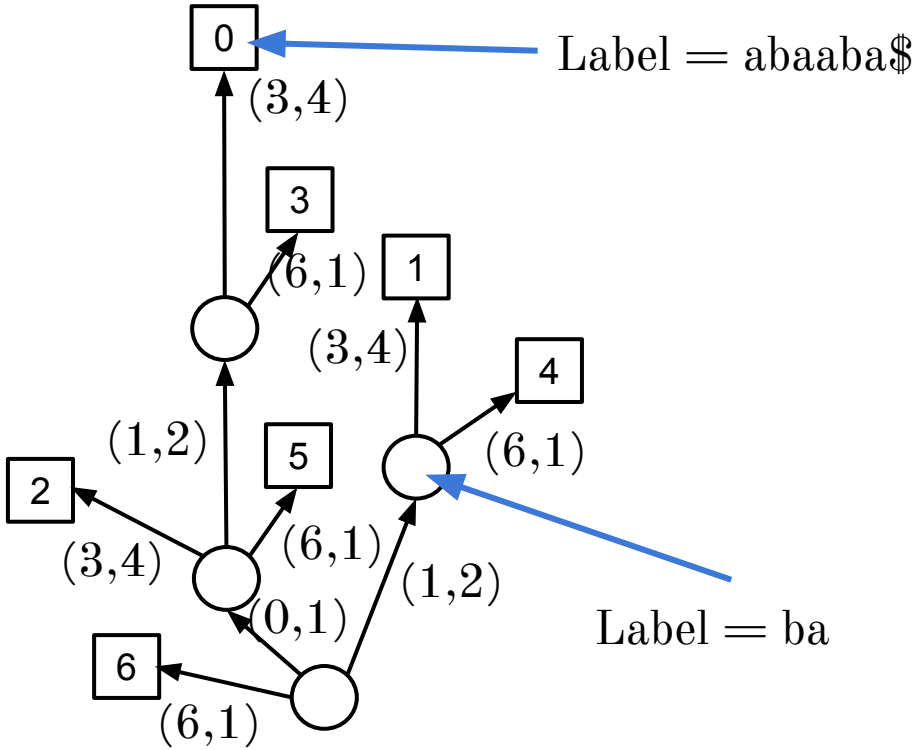
T = abaaba$

Suffix trie: leaves hold offsets of suffixes



$T = abaaba\$$

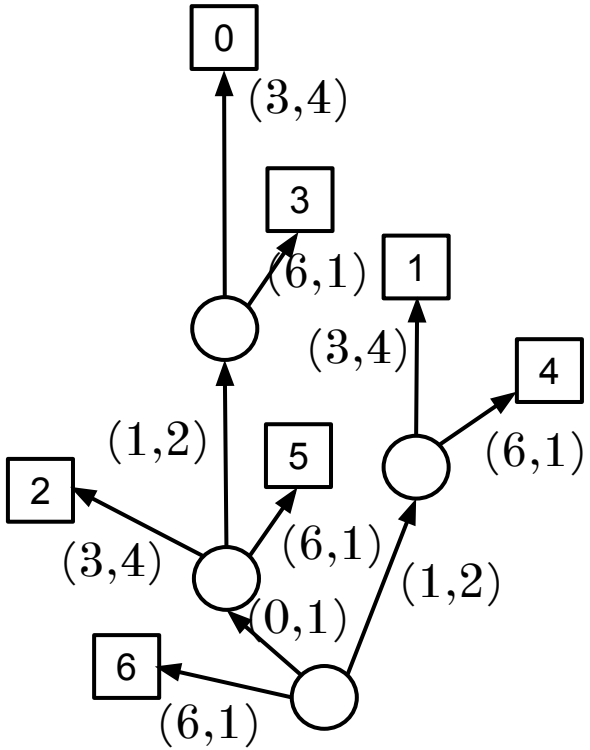
Suffix trie: labels



T = abaaba$

Again, each node's label equals the concatenated edge labels from the root to the node. These aren't stored explicitly

...and we get: Suffix tree



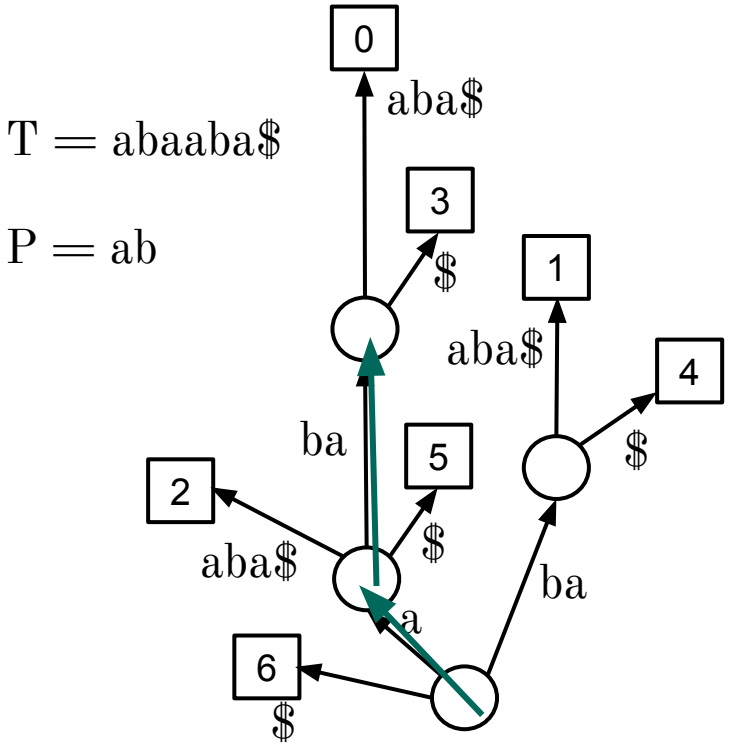
$T = abaaba\$$

Because edges can have string labels, we must distinguish two notions of "depth"

- **Node depth**: how many edges we must follow from the root to reach the node
- **Label depth**: total length of edge labels for edges on path from root to node

Suffix tree

- How do we check whether a string S is a **substring** of T?
- Essentially same procedure as for suffix trie, except we have to deal with coalesced edges



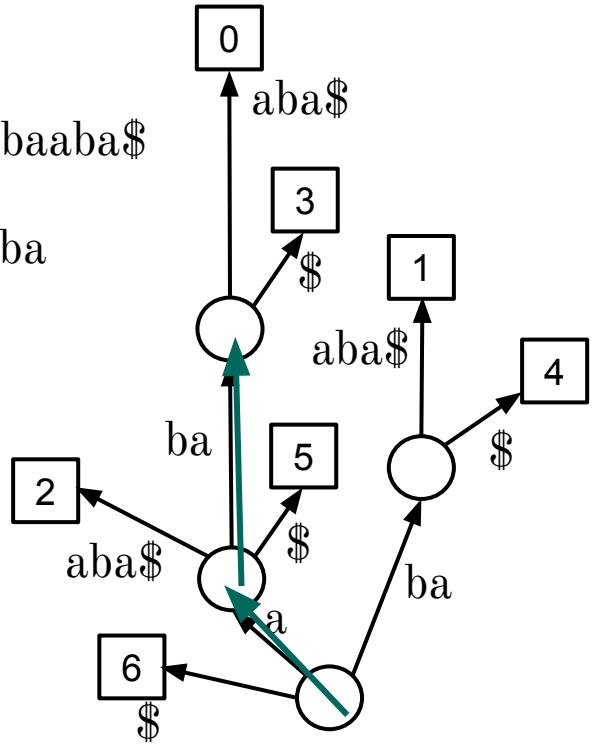
Suffix tree

- How do we check whether a string S is a **suffix** of T?
- Essentially same procedure as for suffix trie, except we have to deal with coalesced edges

```
fall_off, ended_in_node = climb_the_tree()
if fall_off:
    substring = False
    suffix = false
else:
    substring = True
    if ended_in_node:
        suffix = True
    else:
        suffix = False
```

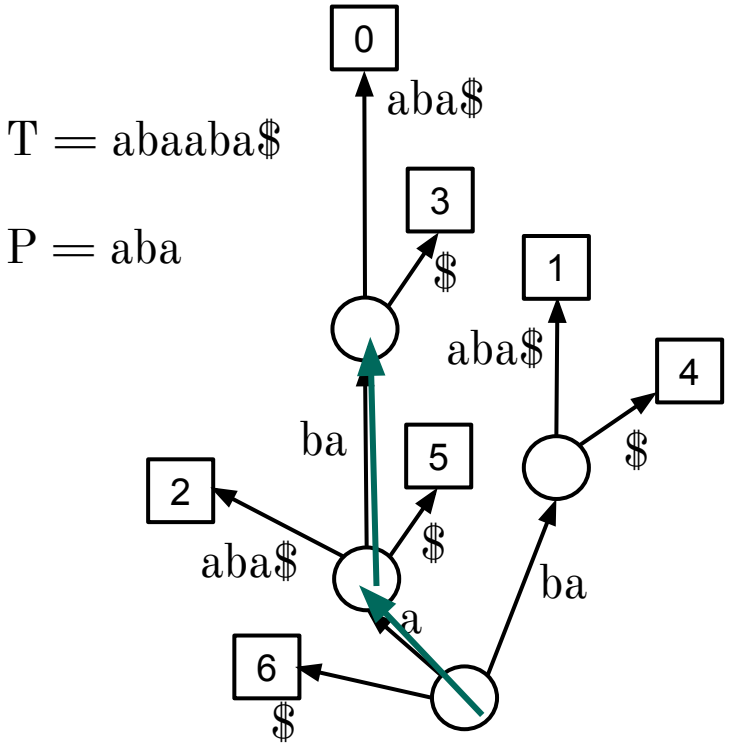
T = abaaba$

P = aba



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: Count the number of branches going out of the node in which we ended following the path
- This is common application of suffix tree: Find all matches of P in T

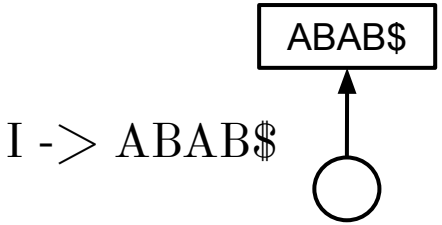


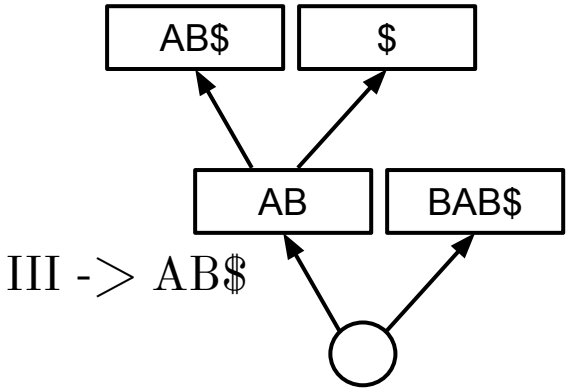
Suffix tree: building

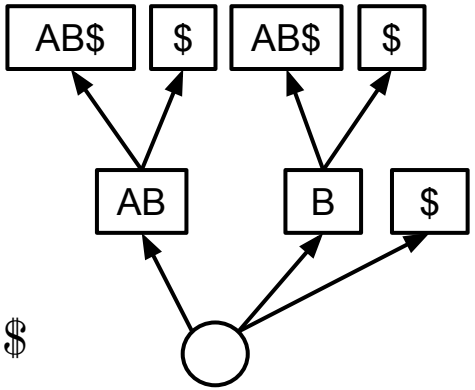
- Naive method 1: build a suffix trie, then coalesce non-branching paths and relabel edges
- Naive method 2: build a single-edge tree representing only the longest suffix, then augment to include the 2nd-longest, then augment to include 3rd-longest, etc
- Both are O(m2) time, but first uses $O(m^2)$ space while second uses O(m)

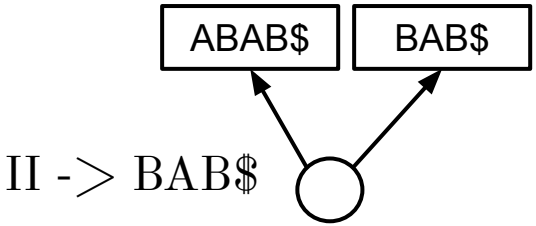
Suffix tree: building Naive method 2

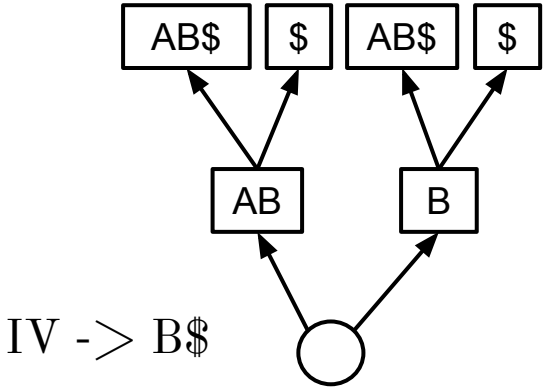
- T = ABAB$











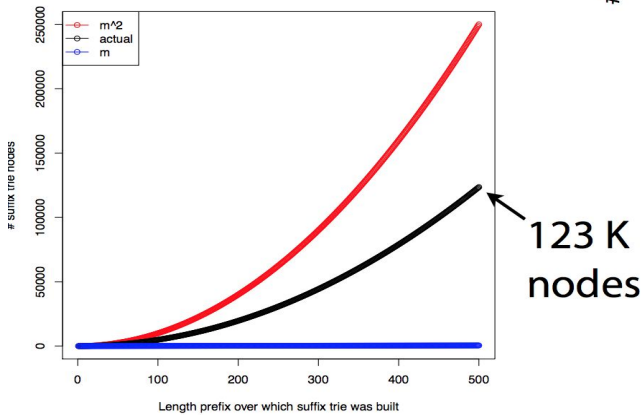
Suffix tree implementation

Suffix tree: building - performance

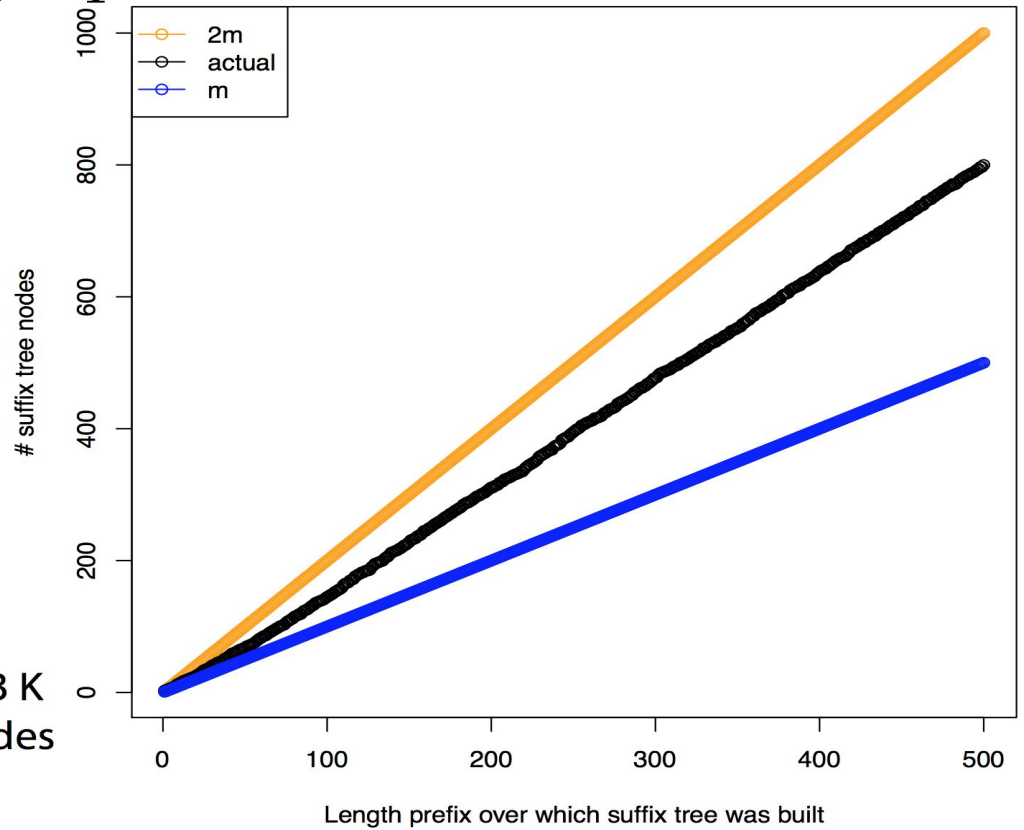
Built suffix trees for the first 500 prefixes of the lambda phage virus genome

Black curve shows # nodes increasing with prefix length

Compare with suffix trie:



123 K nodes



Suffix tree: building

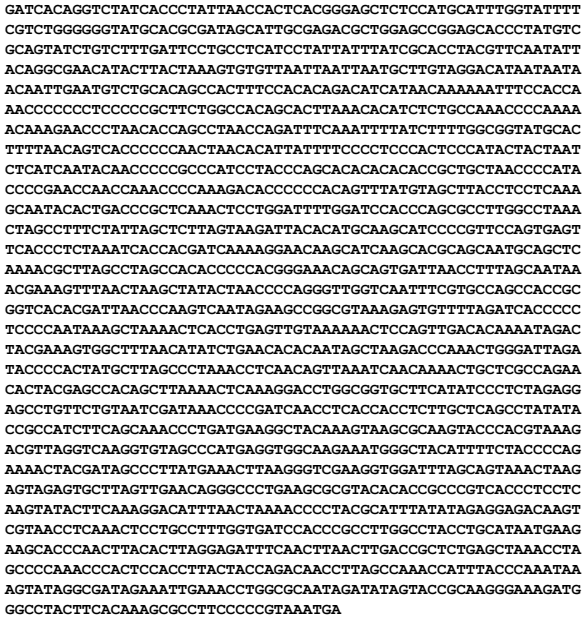
- **Ukkonen's** algorithm - O(m) time and space:
  Ukkonen, Esko. "On-line construction of suffix trees."
  Algorithmica 14.3 (1995): 249-260
- Has online property: if T arrives one character at a time, algorithm efficiently updates suffix tree upon each arrival
- We won't cover it here; see Gusfield Ch. 6 for details

Genome sequence alignment requires approximate

Read

CTCAAACTCCTGACCTTTGGTGATCCACCCGCCTNGGCCTTC

Reference



Sequence differences occur because of...

1. Sequencing errors
2. Natural variation

... 3 billion nucleotides long reference genome...
...and we have to deal with repetitive sequences

Approximate string matching

Looking for places where a P matches T with up to a certain number of mismatches or edits. Each such place is an approximate match.

A mismatch is a single-character substitution (variation) - SNV:

T: GGAAAAAGAGGTAGCGGCGTTTAACAGTAG

P: GTAACGGCG

An edit is a single-character substitution or gap (**insertion** or **deletion**):

T: GGAAAAAGAGGTAGC-GCGTTTAACAGTAG (**insertion)**

P: GTAGCGGCG

T: GGAAAAAGAGGTAGCGGCGTTTAACAGTAG (**deletion)**

P: GTAGC-GCG

Hamming and edit distance

For two same-length strings X and Y, **hamming** distance is the minimum number of single-character substitutions needed to turn one into the other:

X: GAGGTAGCGGCGTTTAAC
Y: GTGGTAACGGGGTTTAAC

Hamming distance = 3

Edit distance (Levenshtein distance): minimum number of edits required to turn one into the other:

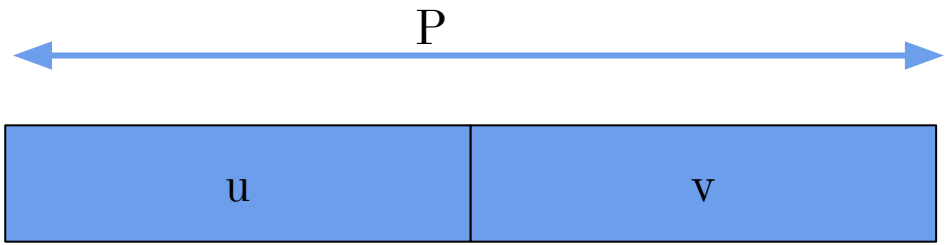
X: TGGCCGCGCAAAAACAGC
Y: TGACCGCGCAAAA- CAGC

Edit distance = 2

What would be the Hamming distance here?

Approximate matching

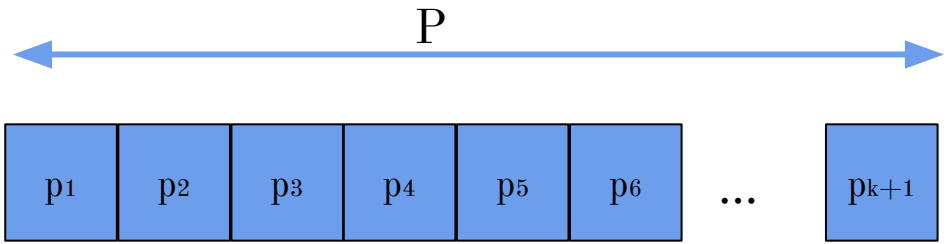
Mission: apply exact matching algorithms to approximate matching problems!



If P occurs in T with 1 edit, then u or v appears with no edits
(u and v are two non-overlapping substrings of P)

Approximate matching

Mission: apply exact matching algorithms to approximate matching problems!

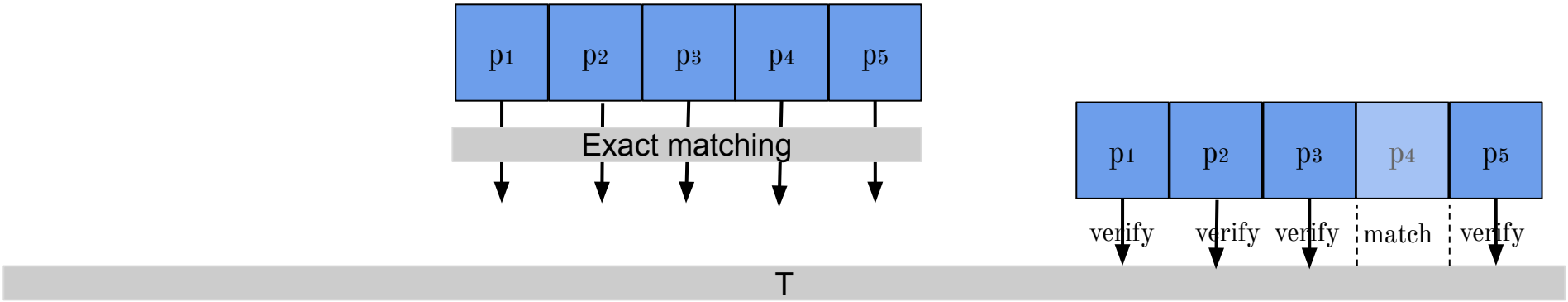


If P occurs in T with up to k edits, at least one of
p1, p2, ..., pk+1 must appear with 0 edits

Pigeonhole principle

If n items are put into m containers, with n $>$ m, then at least one container must contain more than one item (Dirichlet's principle).

Pigeonhole principle



Unix commands in bioinformatics

—

Lesson 6.3

Unix in bioinformatics

- Majority of bioinformatics software is written for Unix (Linux)
- Unix terminal commands could be powerful tool for some simpler analysis
- Faster that Python!

Some basic Unix commands

Command	What it does
ls	Lists the contents of the current directory
mkdir	Creates a new directory
mv	Moves or renames a file
cp	Copies a file
rm	Removes a file
cat	Print or concatenates files
less	Displays the contents of a file one page at a time
head	Displays the first ten lines of a file
tail	Displays the last ten lines of a file
cd	Changes current working directory
pwd	Prints working directory
find	Finds files matching an expression
grep	Searches a file for patterns
wc	Counts the lines, words, characters, and bytes in a file
history	DIsplay previously executed commands

Cut - through the file

- The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output
- It can be used to cut parts of a line by **byte position**, **character** and **field**
- Useful for slicing columns from TSV/CSV

```
$ cut -c 3,6,8 example.txt  # Extracts character 3,6 and 8 from each line (one based)

$ cut -f 2-4 example.tsv  # Extracts columns 2,3 and 4 from file (TAB is default delimiter)

$ cut -f -3 -d ‘,’example.csv  # Extracts columns 1,2 and 3 from file
```

awk - dig into the file

- Search files for lines that contain certain patterns
- awk refers to a program, and to the language used by program
- When a line matches the patterns, awk performs defined actions on that line
- awk keeps processing input lines until the end of the input file is reached
- Options:
  - -F fs To specify a file separator.
  - -f file To specify a file that contains awk script.
  - -v var=value To declare a variable.

```
$ awk -F: '{print $1}' /etc/passwd  # same as cut -f 1 -d ‘:’ /etc/passwd

$ echo "Hello Tom" > hello
$ awk '{$2="Adam"; print $0}' hello  # Outputs Hello Adam. $0 prins the entire line

$ awk 'BEGIN {print "The File Contents:"} {print $0} END {print "File footer"}' myfile
# Print contents of the file and add a sentence to the start and end of it
```

awk - dig into the file (2)

```
$ awk 'BEGIN{FS=":"; OFS="-"} {print $1,$6,$7}' /etc/passwd
# OFS  Specifies the Output separator, DEFAULT IS " "

$ awk '{if ($1 > 30) print $1}' test.tsv  # Output first column if its value is > 30

$ awk '{if ($1 > 30){x = $1 * 3; print x} else{x = $1 / 2; print x }}' testfile

# awk supports mathematical functions: sin(x) | cos(x) | sqrt(x) | exp(x) | log(x) | rand()

$ $ awk 'BEGIN{x = "likegeeks"; printf "The output is: %e\n", toupper(x)}'
```

Source

sed command - non-interactive stream text editor

- Modifying the input as specified by a list of commands.
- A single command may be specified as the first argument to sed

```
$ echo "ATATATAGAATGATGA" | sed 's/TA/CG/' # s command replaces the first text with the second text pattern

$ sed 's/test/another test/2' myfile  # specifying the occurrence number that should be replaced like this

$ sed -n 's/test/another test/p' myfile # The p flag prints each line with a matching pattern, -n option to prints the modified lines only.

$ sed '2,3s/test/another test/' myfile # Only lines 2 and 3 are modified

$ sed '2,$s/test/another test/' myfile # Modify starting from line 2 to the end
```

Source

sed command - non-interactive stream text editor

```
$ sed '2d' myfile # Deletes 2nd line from the stream, not the original file

$ sed '3,$d' myfile # Keeps only first two lines from myfile

$ sed '2a\This is the appended line.' myfile # Appends line after second line

$ sed '3c\This is a modified line.' myfile # Replaces 3rd line

$ $ sed 'y/123/567/' myfile  # Replaces character 1->5, 2->6, 3->7
```

Source

Grep this!

- Searches for the pattern inside the file: **`grep pattern file_name`**

```
$ grep '^\.Pp' myfile # Find all occurrences of ’.Pp' at the beginning of a line

$ grep -v -e 'foo' -e 'bar' myfile  # To find all lines in a file which do not contain the words `foo' or `bar'

$ grep -B 1 -A 1 'aagtagggttca' hg38.fasta  # Search for a nucleotide sequence and print 1 line before and after any match. It won’t find the pattern if it spans more than 1 line.

$ grep -i "is" demo_file  # Key upper/lower case insensitive

$ grep -iw "is" demo_file  # “is” must be a word, surrounded by spaces
```

Grep this!

```
$grep -r "GATTACA" *  # Searching in all files recursively using grep

$ grep -v "go" demo_text # Invert search (include if pattern is not found)

$ grep -c "go" demo_text  # count how many lines matches the given pattern

$ grep -n "go" demo_text  # Show line number while displaying the output

$ grep -m 1 pattern file # Stops search after first match

$ grep -E 'pattern1|pattern2' filename  # Look for appearance of any of two

$ grep -E 'Dev.*Tech' employee.txt # Look for Dev and Tech. No AND in grep.
```

How to grep for “Dev” or “Tech” but not “PM”?

Unix pipe (|)

- Passed stdout of one command to stdin of the other
- Very useful for fast file manipulation
- Saves time for writing/reading to hard drive

```
$ grep -E 'Dev|Tech' employee.txt | grep -v PM

$ cat state.txt | head -n 3 | cut -d ' ' -f 1 > list.txt

$ ls -al | grep '^d'  # List only directories

$ ps auxwww | grep jupyter # List all processes containing string jupyter in the name

$ ps aux | wc -l # Count the number of currently running processes

$ sort record.txt | uniq  # Sort file and remove duplicate lines

$ cat result.txt | grep "Rajat Dua" | tee file2.txt | wc -l
# read the particular entry from user and store in a file and print line count.
```

Source
Source2

Unix pipe to file > and >>

- Pass command output to file
- “>” write
- “>>” append

```
$ grep John names.txt > john.txt
```

Command chaining operators

- Executing set of commands within one command
- Every Unix commands returns a value
- Zero, by default, if everything went well
- && || and ;
- Useful for executing more commands within same Docker container

```
$ untar seq.fasta.tar && bwa mem fq1 fq2 seq.fasta > out.sam
```

How to...?

- Concatenate two tables with the same columns?

```
Name ID   Available
John 332323    Yes
Mike 343434    No
Steven    323421    YES
```

```
Name ID   Available
Bin  336323    Yes
Vera 373434    Yes
Sara 324441    YES
```

```
$ cat table1.tsv | sed 1d table2.tsv
```

Vim editor

- Interactive ultra fast, keyboard-only text manipulation
- Insert, command and visual mode
- More to know than just: How can I exit?

Resources and additional reads

Presentation available at: github.com/vladimirkovacevic/gi-2020-etf

- Vince Buffalo: Bioinformatics Data Skills
- Dan Gusfield: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge
- Pavel Pevzner, Neils Jones: An Introduction to Bioinformatics Algorithms (Computational Molecular Biology), MIT
- R. Durbin, S. Eddy, A. Krogh, G. Mitchinson: Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids , Cambridge University Press
- Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, Alexandru I. Tomescu: Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing, Cambridge University press
- Introduction to Unix