CMSC423: Chapter 9
Suffix tree, suffix arrays, Burrows Wheeler Transform

Class so far...

- Deterministic searching (counting, clumps)
- Exact matching (KMP, Z algorithm)
- Randomized searching (Gibbs sampling)
- Branch and bound search (Proteomics)
- Dynamic programming for inexact matching

- This week: exact matching again, for indexing

Stop and think

- Given a text T and pattern P
- Find the longest prefix of P that matches somewhere in T

- Note: KMP solves this for the prefix that is the whole P
- What if the whole of P does not match?

Stop and think...part 2

- Given text T and pattern P
- Find the longest substring of P that matches somewhere in T
- in O(n) time

- Substring – the characters are adjacent (unlike subsequence discussed last week)

- Note: dynamic programming solves the above in $O(n^2)$ time (pick the right weights and use local alignment)

Solution...

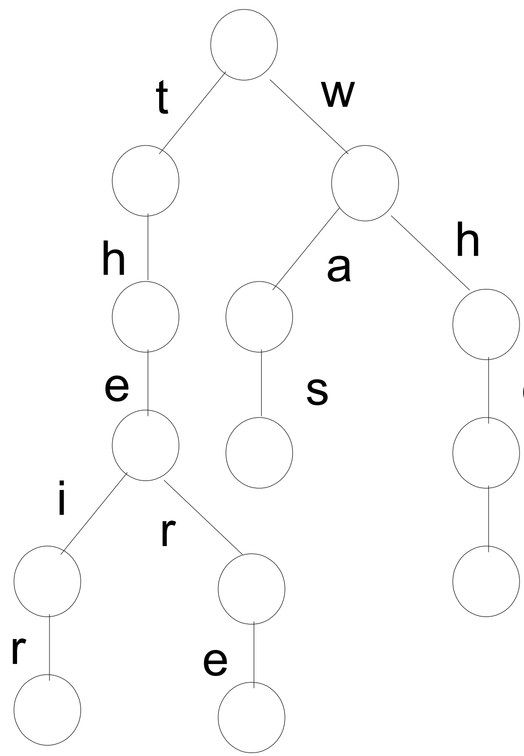
- Note: Donald Knuth did not think O(n) was possible

- Solution:
  - Think of suffixes
  - Each substring is a prefix of a suffix
  - But we know how to solve longest prefix

- How do we organize suffixes?

Many strings: trie

- Basic idea: if many strings share a same sequence only represent it once in the tree



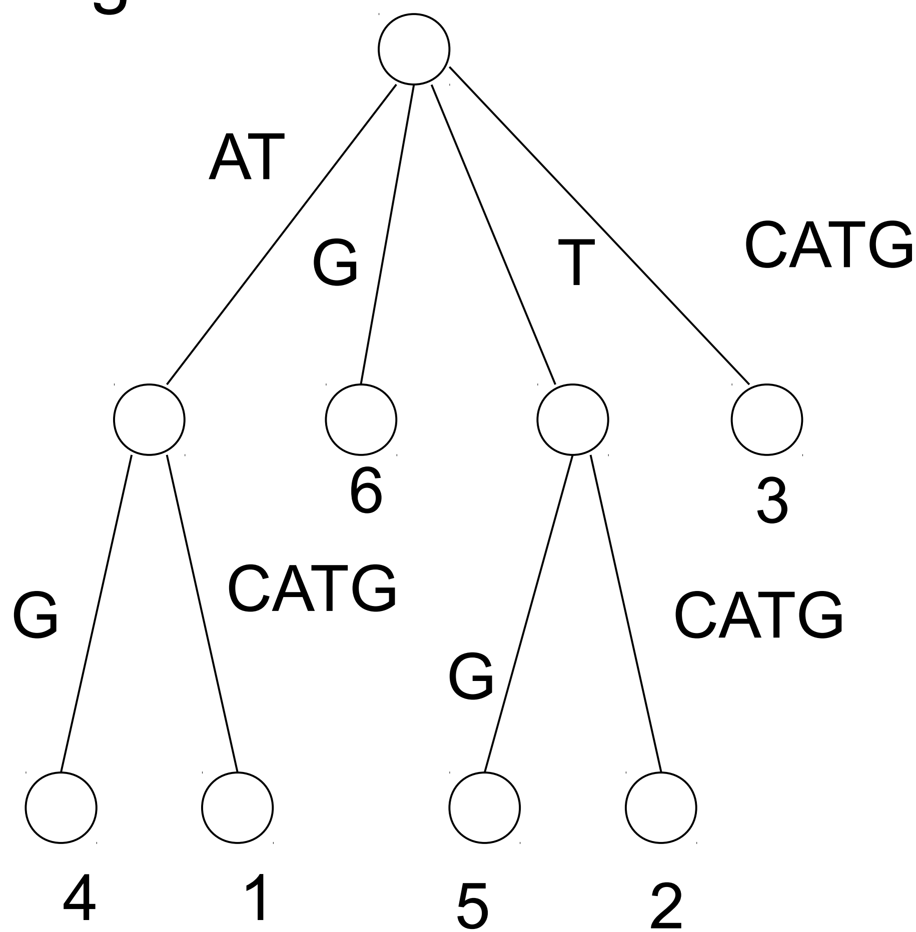
their
there
was
when

Stop and think: How many nodes are in the suffix trie for a string of length N?

Suffix tree

- Extends trie of all suffixes of a string
- Collapses non-branching nodes

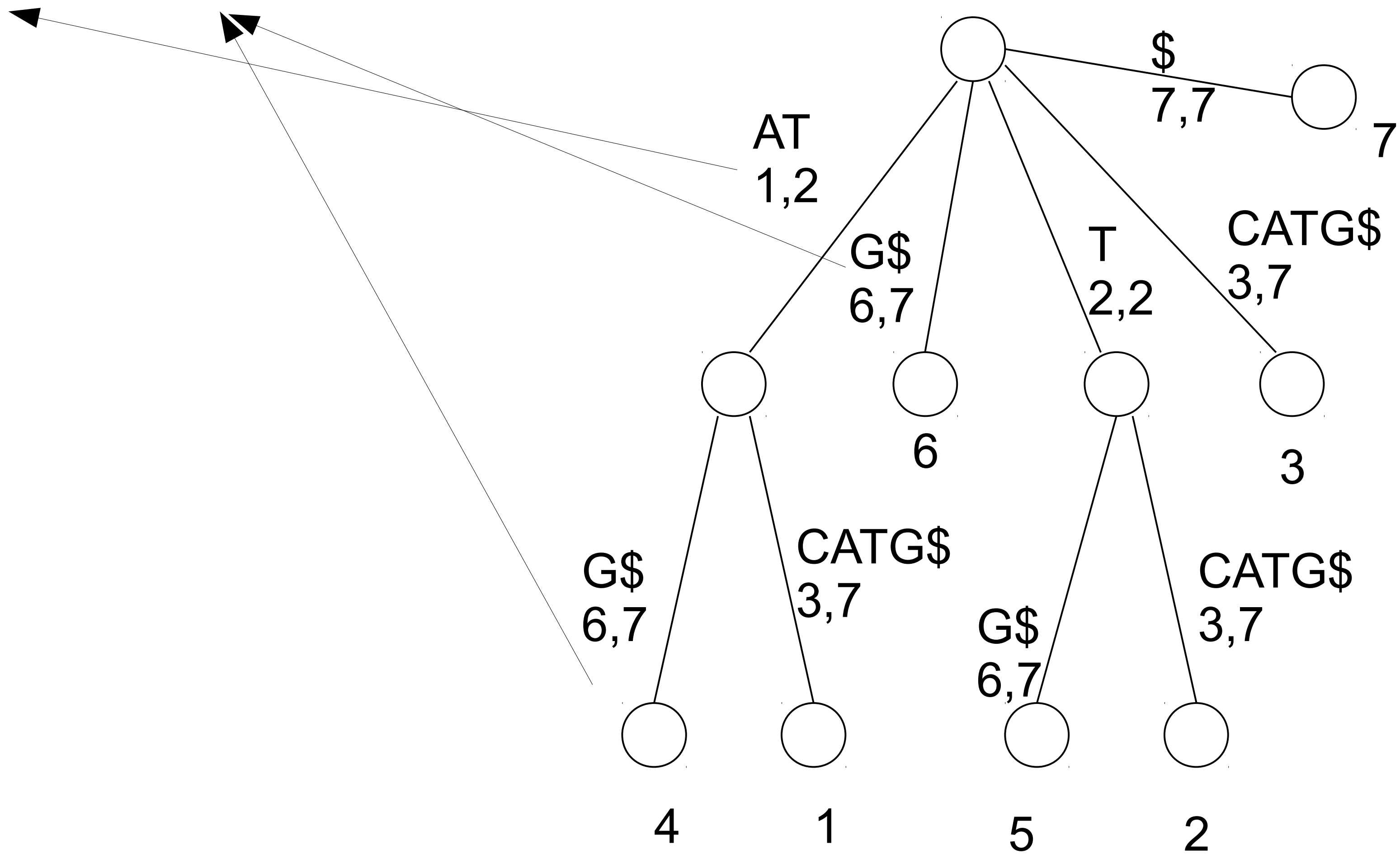
1	ATCATG
2	TCATG
3	CATG
4	ATG
5	TG
6	G



Stop and think:
How many nodes are in the suffix tree for a string of length N?
How much memory do you need to store the suffix tree?

Suffix tree ...cont

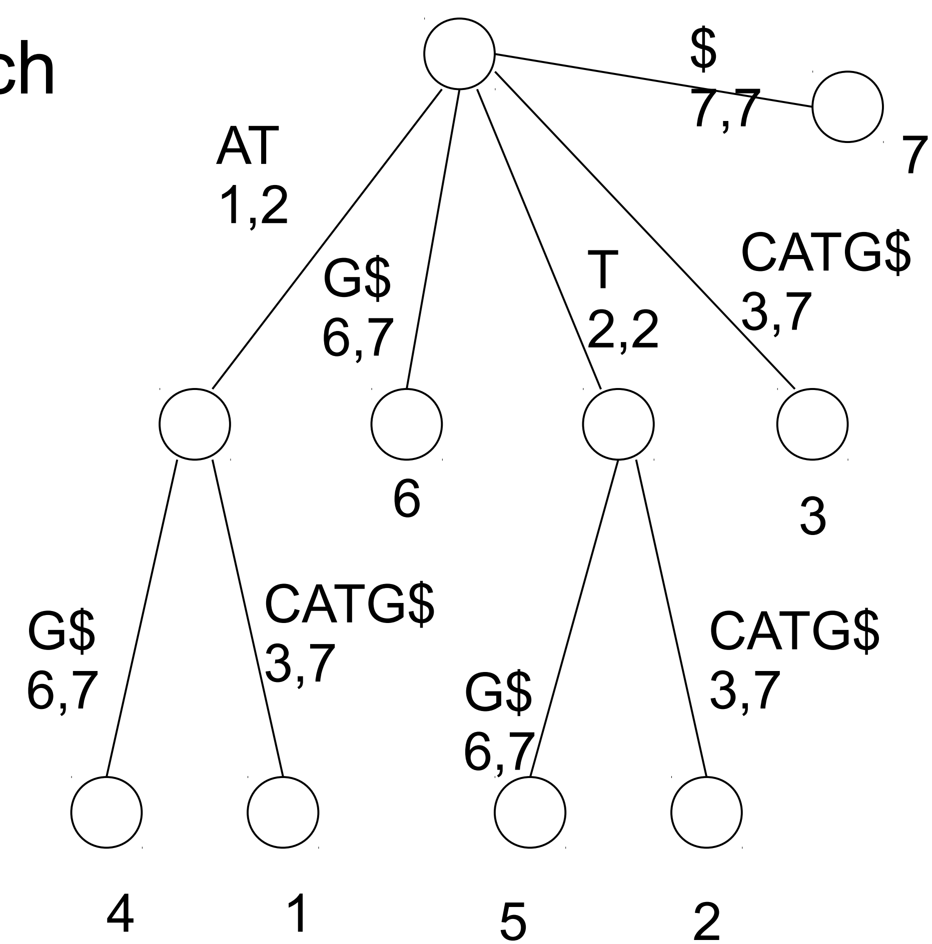
- To store in linear space – just store range in sequence instead of string
- To ensure suffixes end at leaves, add $ char at end of string
- ATCATG$



Suffix trees for matching

- Suffix trees use O(n) space
- Suffix trees can be constructed in O(n) time
- Is CAT part of ATCATG ?
- Match from root, char by char
- If run out of query – found match
- otherwise, there is no match

- intuition: CAT is the prefix of some suffix



Other uses

- Finding repeats
  - internal nodes with multiple children – DNA that occurs in multiple places in the genome
- Longest common substring of two strings
  - build suffix tree of both strings. Find lowest internal node that has leaves from both strings
  - or: build suffix tree on one string and use suffix links to find longest match
- Note: running time for matching is O(|Pattern|),
  not O(|Pattern| + |Text|)
  (though O(|Text|) was spent in pre-processing)
- In KMP,runtime is O(|Text|) with O(|Pattern|) preprocesssing

Suffix arrays

- Suffix trees are expensive > 20 bytes / base
- Suffix arrays: lexicographically sort all suffixes

ATG	4
ATCATG	1
CATG	3
G	6
TCATG	2
TG	5

- Can quickly find the correct suffix through binary search

- Stop and think: How long does it take to sort N strings of length L?

Suffix arrays and compression

- Burrows-Wheeler transform

BANANA

BANANA$
ANANA$B
NANA$BA
ANA$BAN
NA$BANA
A$BANAN
$BANANA

sort →

character before the suffix

$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA

→ ANNB$AA (BWT) → compress

Note: characters in last column occur in same order as in first column
Useful for matching within BWT

BWT – string matching

- Look for “BANA”
- Start at end (match right to left)
- Find character in rightmost column
- Identify corresponding range in first column
- Switch back to last column
- ...
- How do we know the first A in the pattern is the 2nd/3rd from the top of the matrix?
- Note: add'l data needed: # of times each letter appears before every pos'n
- Running time?

<u>$</u>BANANA
<u>A$</u>BANAN
N
A
<u>ANA$</u>BAN
A
<u>ANANA$</u>B
B
<u>BANANA$</u>
<u>NA$</u>BANA
A
<u>NANA$</u>BA

ABN$
0000
1000
1010
1020
1120
1121
2121

O(len(P)) operations. Each may cost O(log(len(T)))