CMSC423: Bioinformatic Algorithms, Databases and Tools Lecture 8

Sequence alignment:

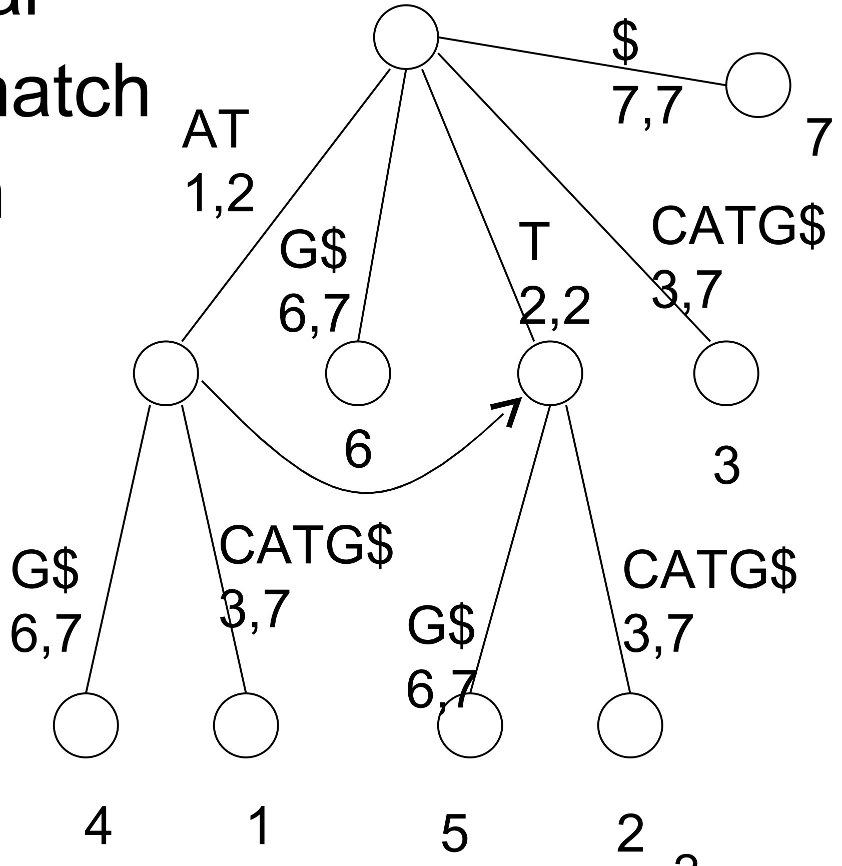
exact alignment

inexact alignment

dynamic programming, gapped alignment

Suffix trees for matching

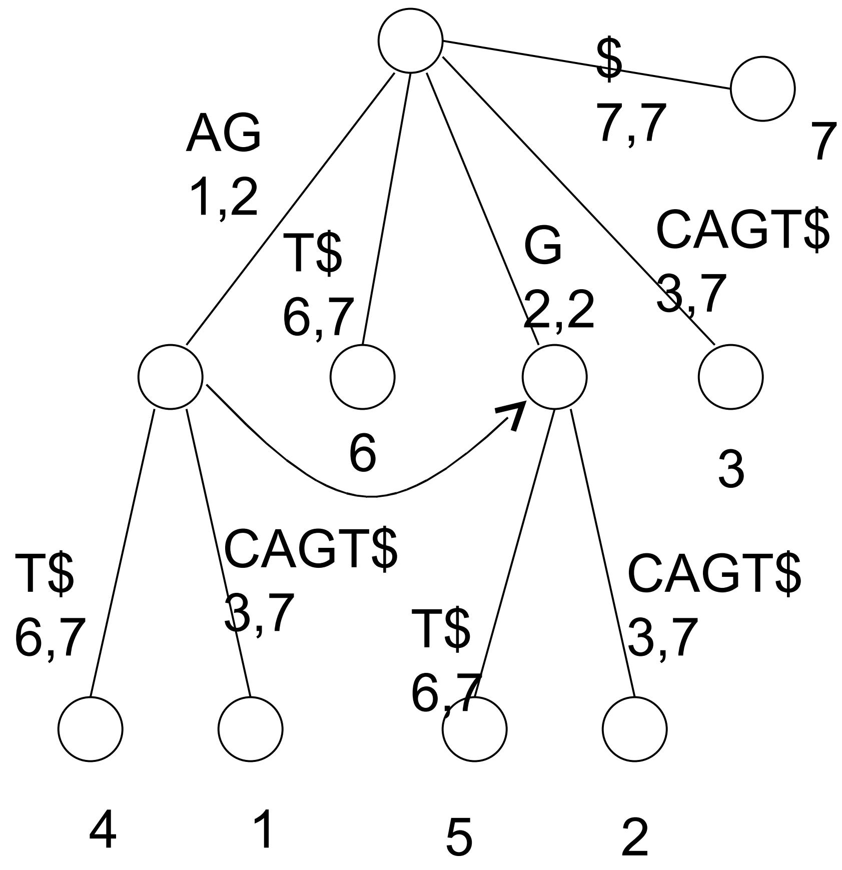
- Suffix trees use O(n) space
- Suffix trees can be constructed in O(n) time
- Is CAT part of ATCATG ?
- Match from root, char by char
- If run out of query – found match
- otherwise, there is no match



- intuition: CAT is the prefix of some suffix

Suffix links – useful for substring matches

- Does any part of AGATG match string AGCAGT?



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

Why do we care?

- Suffix trees are used for
  - mapping reads to a genome (e.g. personal genomics)
  - comparing genomes (comparative genomics)
  - finding repeats
  - identifying genome signatures
- Exact matching – what to expect on exams
  - build a suffix tree for a string
  - answer some questions about one of the algorithms, e.g. for Z algorithm – is it necessary j be the farthest reaching Z-value or just any Z value extending past i?
  - do something with the help of some of the algorithms (e.g. look for repeats that occur exactly twice, etc.)

Suffix arrays

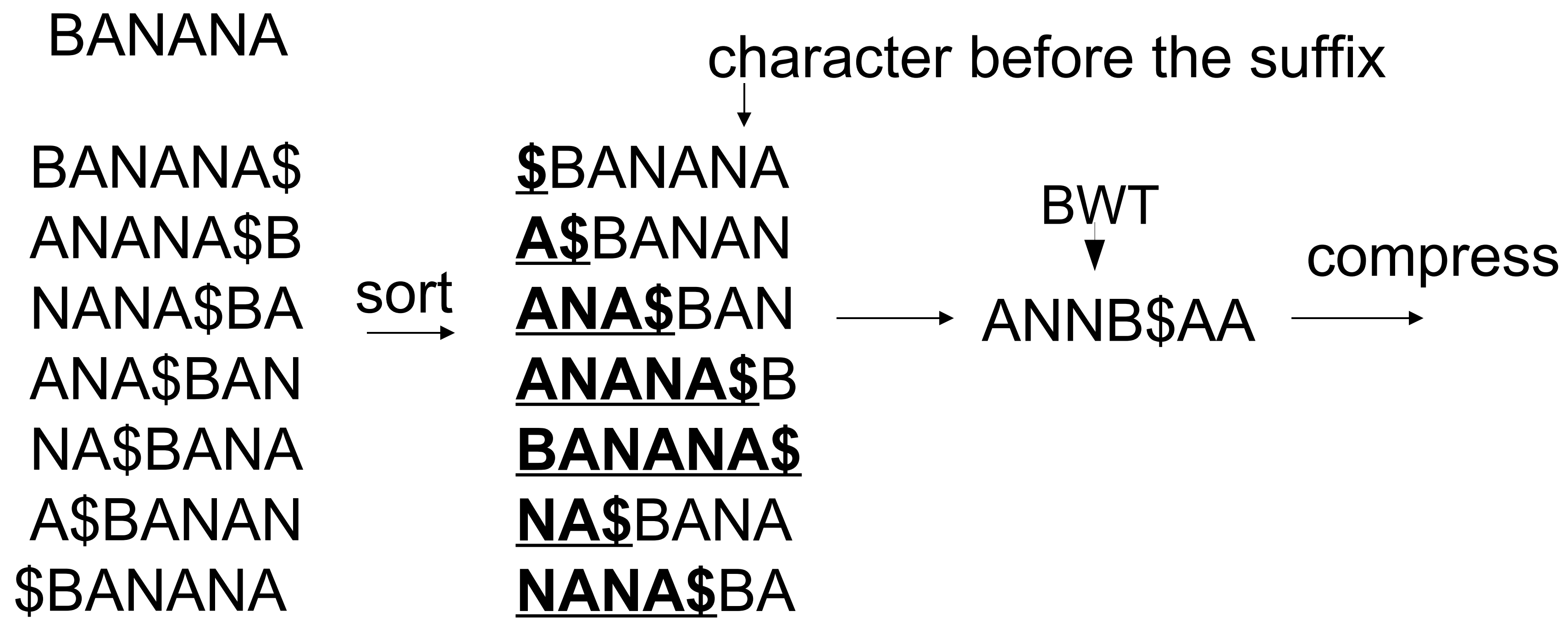
- Suffix trees are expensive > 20 bytes / base
- Suffix arrays: lexicographically sort all suffixes

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

- Can quickly find the correct suffix through binary search
- Note: much less space, but longer running time (incur a log(n) term)

Suffix arrays and compression

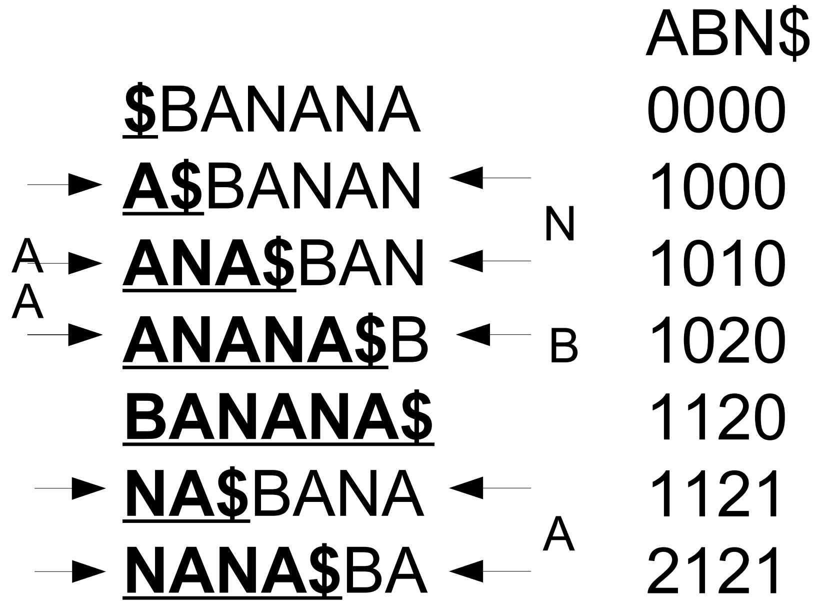
- Burrows-Wheeler transform



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?



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

Exact alignment recap

- Exact matching can be done efficiently:
  O(|Text| + |Pattern|)

- Key idea: preprocess data to keep track of similar regions, then use information to "jump" over places where no match can occur