CMSC423: Bioinformatic Algorithms, Databases and Tools

Exact string matching:

Suffix trees

Suffix arrays

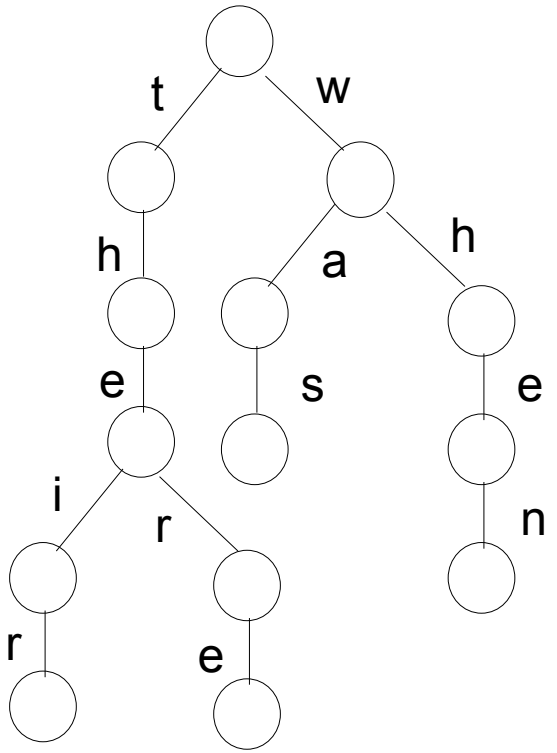
Searching multiple strings

- Can we search multiple strings at the same time?

- Would it help if we broke a single string into multiple strings?

Many strings: trie

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



their
there
was
when

- Aho-Corasick algorithm extends sp(i) values to a trie

From one string to many

- Break up a string into all its suffixes

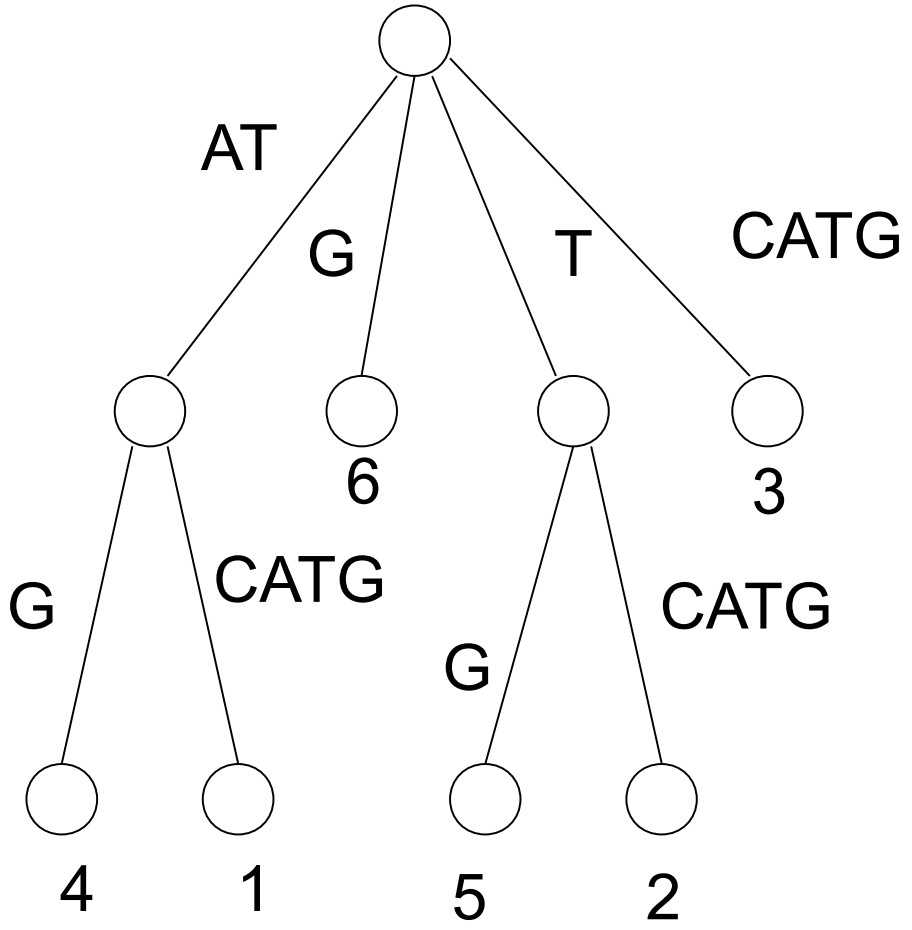
- Build a search structure from the suffixes (assume it's easy to search a collection of strings even if I haven't shown you how to do it)

- What questions could you answer?

Suffix tree

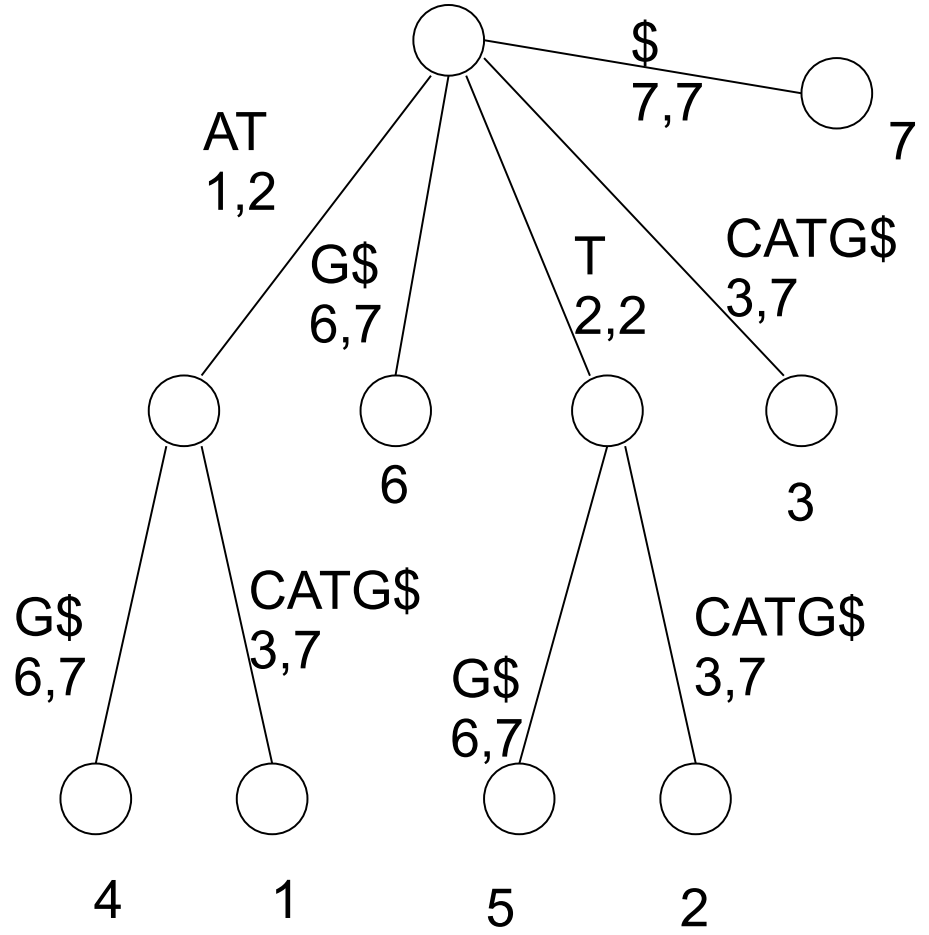
- Extends trie of all suffixes of a string

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



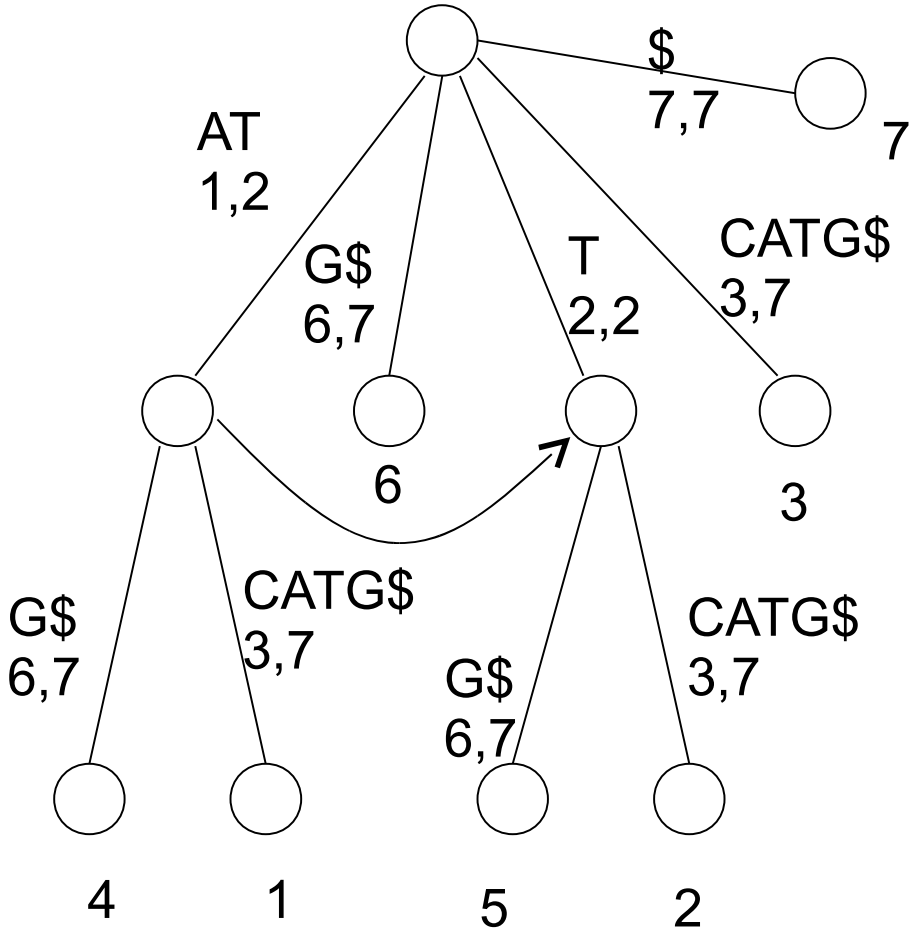
Suffix tree ...cont

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



Suffix links

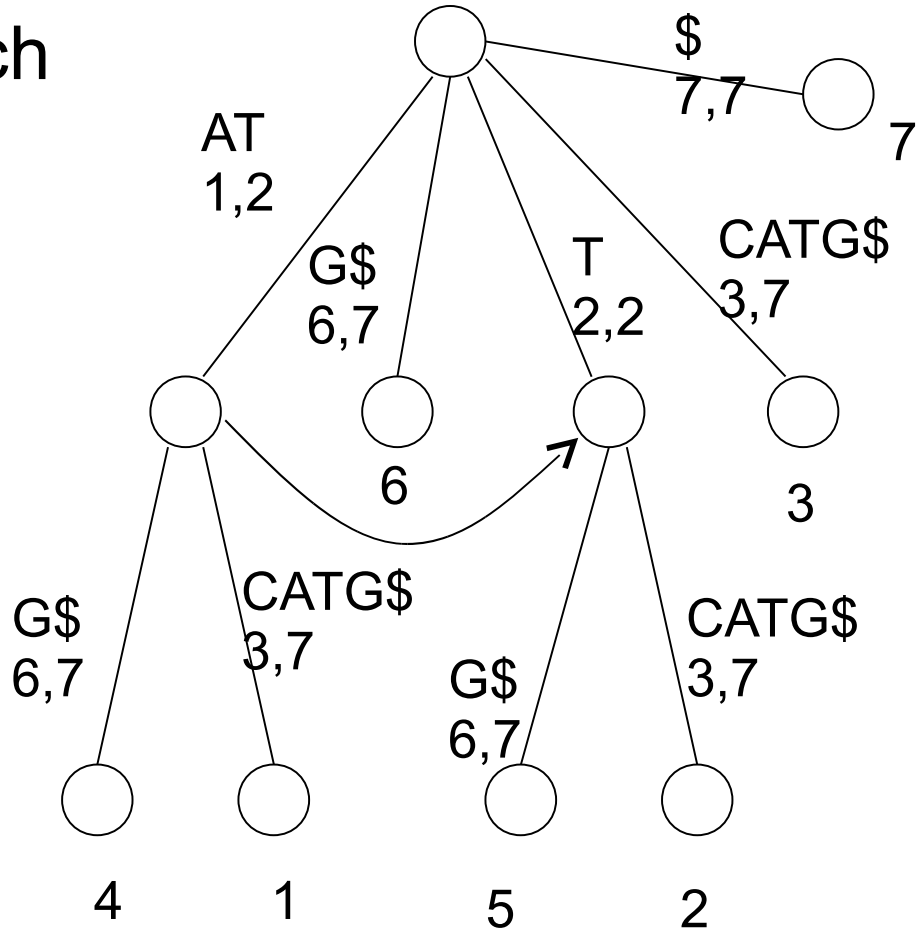
- Link every node labeled aS for some string S to node labeled S (note – it always exists)



Suffix trees for matching

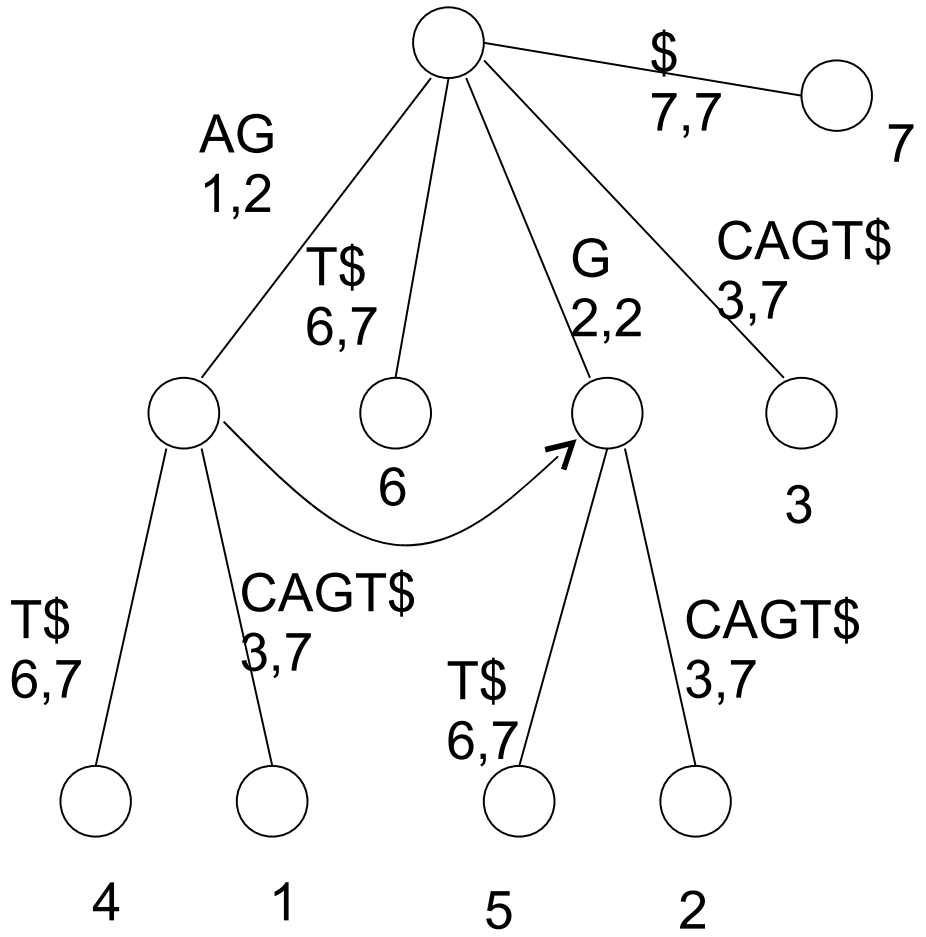
- Suffix trees use O(n) space
- Suffix trees can be constructed in O(n) time
- Is CAT part of ATCATG ?
- Match from root, char by char
- If run out of query – found match
- otherwise, there is no match

- 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

Suffix arrays

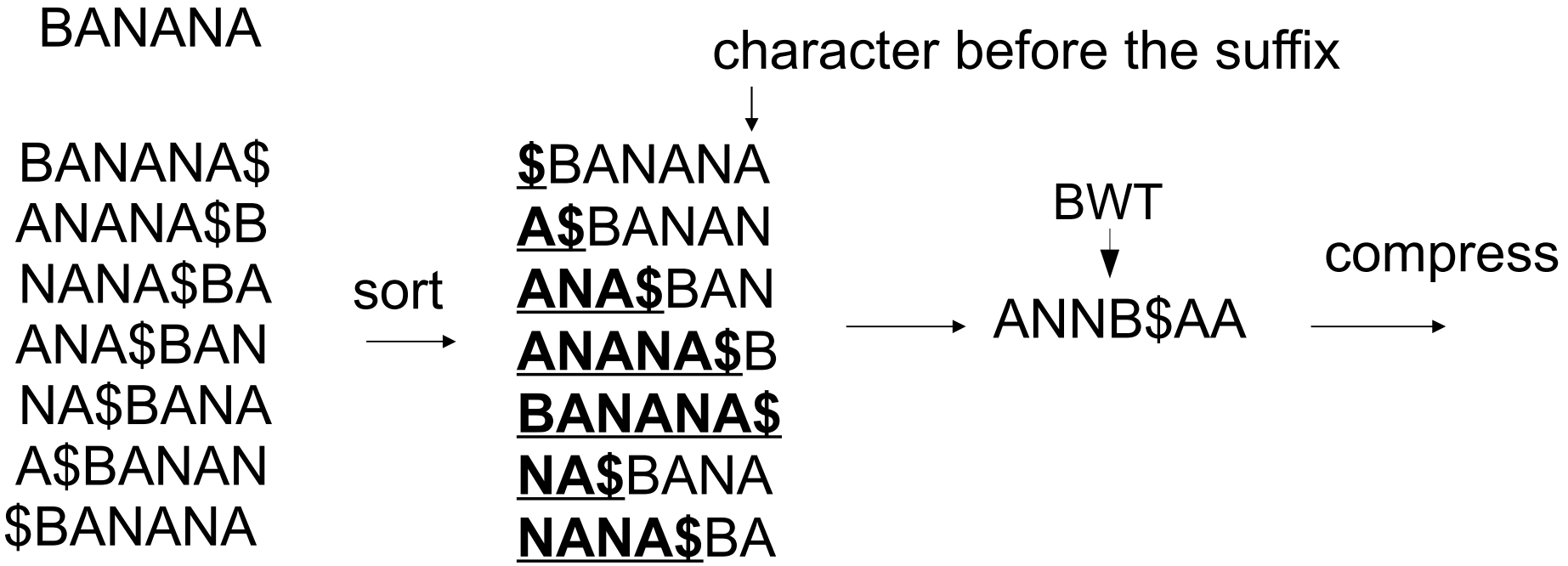
- Suffix trees are expensive > 20 bytes / base
- Suffix arrays: lexicographically sort all suffixes

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

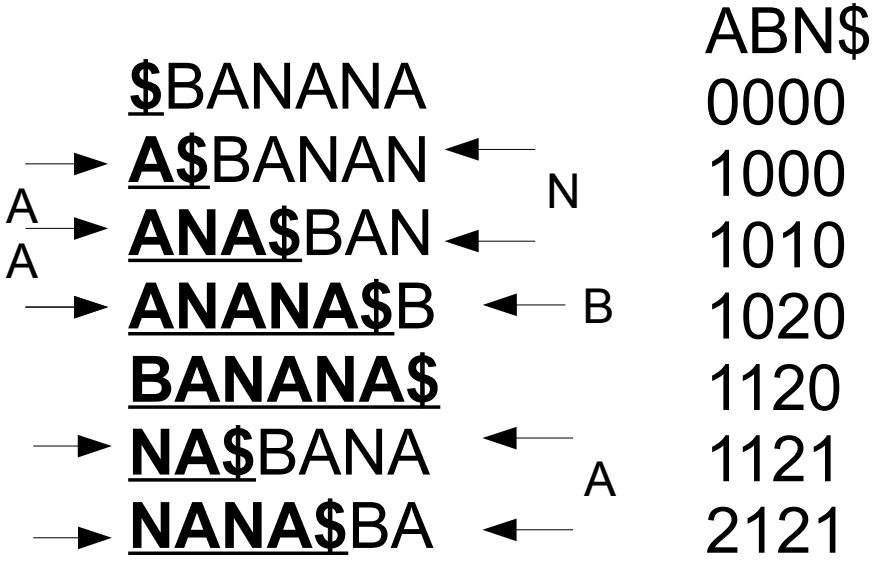
- Burrows-Wheeler transform



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)))

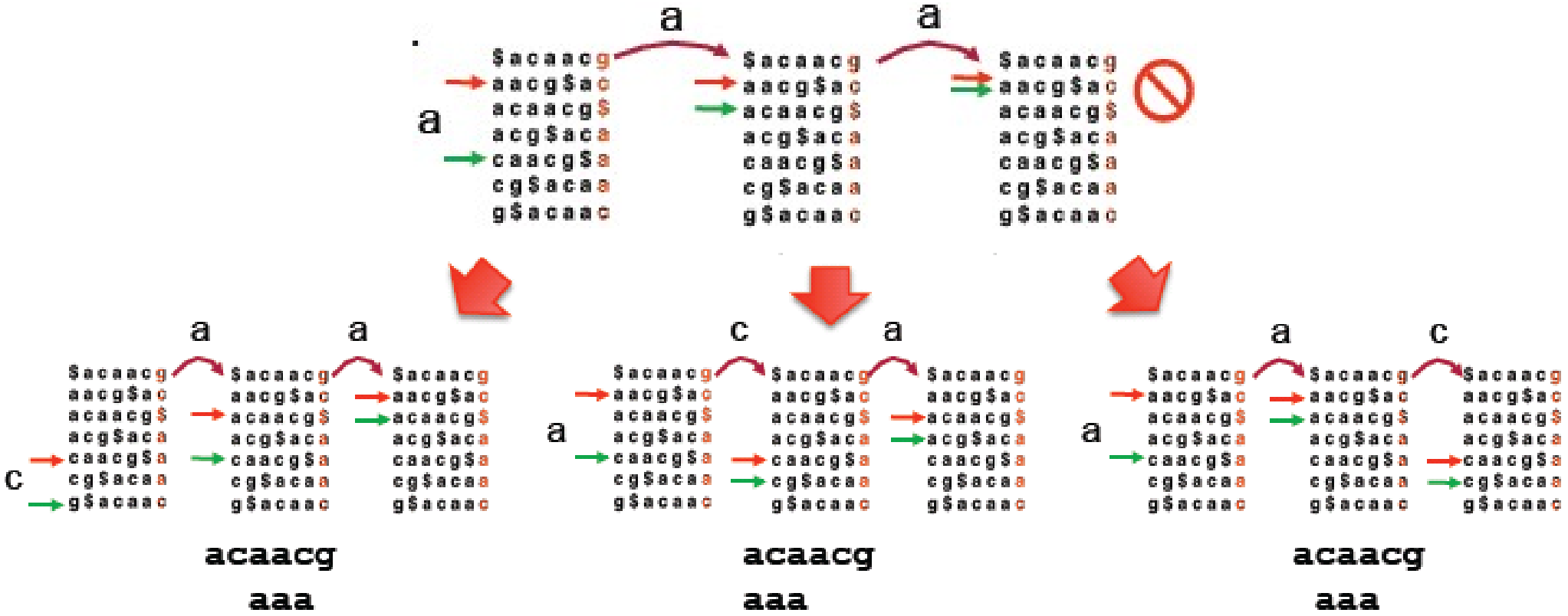
Application of BWT

- Alignment of short reads to human genome sequence
- Needs to be done fast
- Needs to be done in low memory
- Needs to handle errors (sequencing errors or single nucleotide polymorphisms)

- Trick – backtracking

  Langmead, Trapnell, Pop, Salzberg. *Ultrafast and memory-efficient alignment of short DNA sequences to the human genome*. Genome Biol. 10(3):R25. 2009
  http://genomebiology.com/2009/10/3/R25

Bowtie backtracking



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