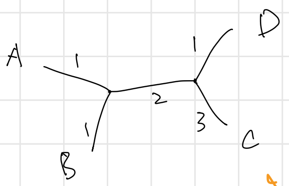
Review

1. Distance-based phylogeny.



	A	B	C	D
A	0	2	5	3
B	2	0	6	4
C	5	6	0	4
D	3	4	4	0

$\approx$

	A	B	C	D
A	0	2	6	4
B	2	0	6	4
C	6	6	0	4
D	4	4	4	0

2. How to get a good distance.

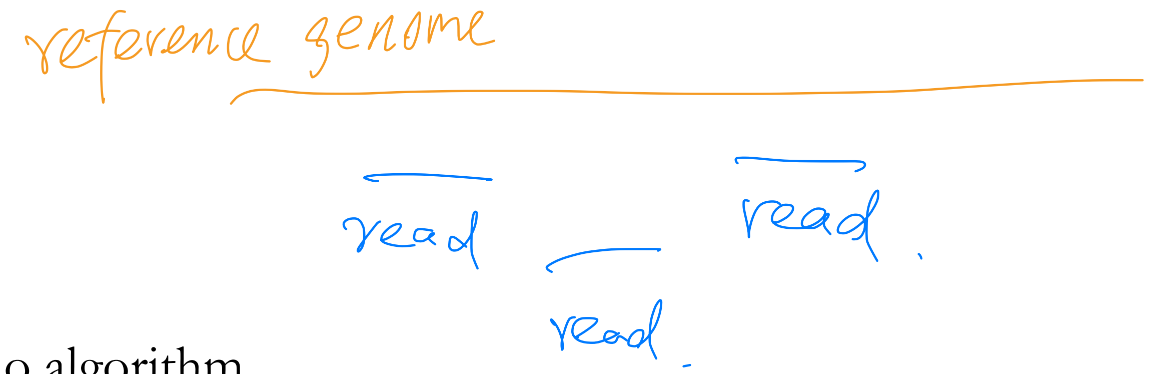
- # of mutations
- edit distance
- Information distance.

3. phylogeny v.s. classification.

Suffix Tree and Array

String Matching

- So far we learned how to find "approximate" matches – the alignments. And they are difficult. Finding exact matches are much easier.
- To search for a short string P of length m in a large text T of length n.
- Applications:
  - Keyword searching
  - DNA reads mapping
- Type I: Match only once.
  - E.g. KMP algorithm and Apostolico-Giancarlo algorithm.
  - O(m) to preprocess, and O(n) to match.
- Type II: Match multiple patterns multiple times.
  - Better index T first to speed up the matching time.



Things To Study

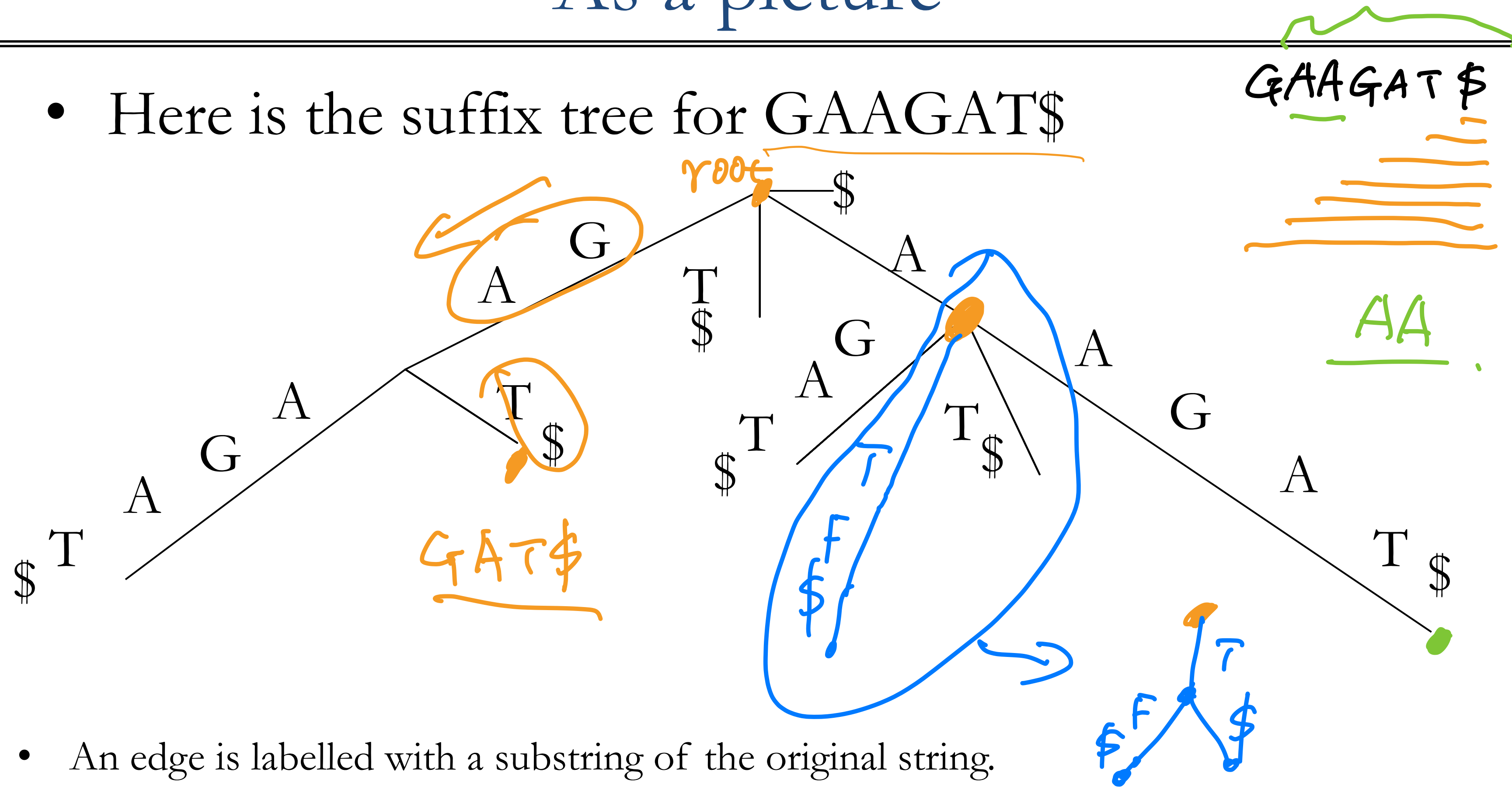
- Suffix tree and array are two data structures for this purpose.
- Suffix Tree
  - Data structure
  - A few examples of using suffix tree to solve practical problems.
- Suffix Array
  - Data structure
  - The skew algorithm for constructing suffix array.

A Little History

- 1973, Weiner introduced the concept of suffix tree (position tree), which Donald Knuth subsequently characterized as "Algorithm of the Year 1973".
- 1990, Gene Myers and Udi Manber proposed suffix array.
  - Gene Myers: former VP Informatics Research at Celera Genomics
  - Udi Manber: VP engineering, Google.
- 1992, Gonnet, Baeza-Yates & Snider independently discovered suffix array (called PAT array).
  - Gaston Gonnet: cofounders Maplesoft and OpenText.
  - Baeza-Yates: VP for Yahoo! Europe and Latin America.

As a picture

- Here is the suffix tree for GAAGAT$



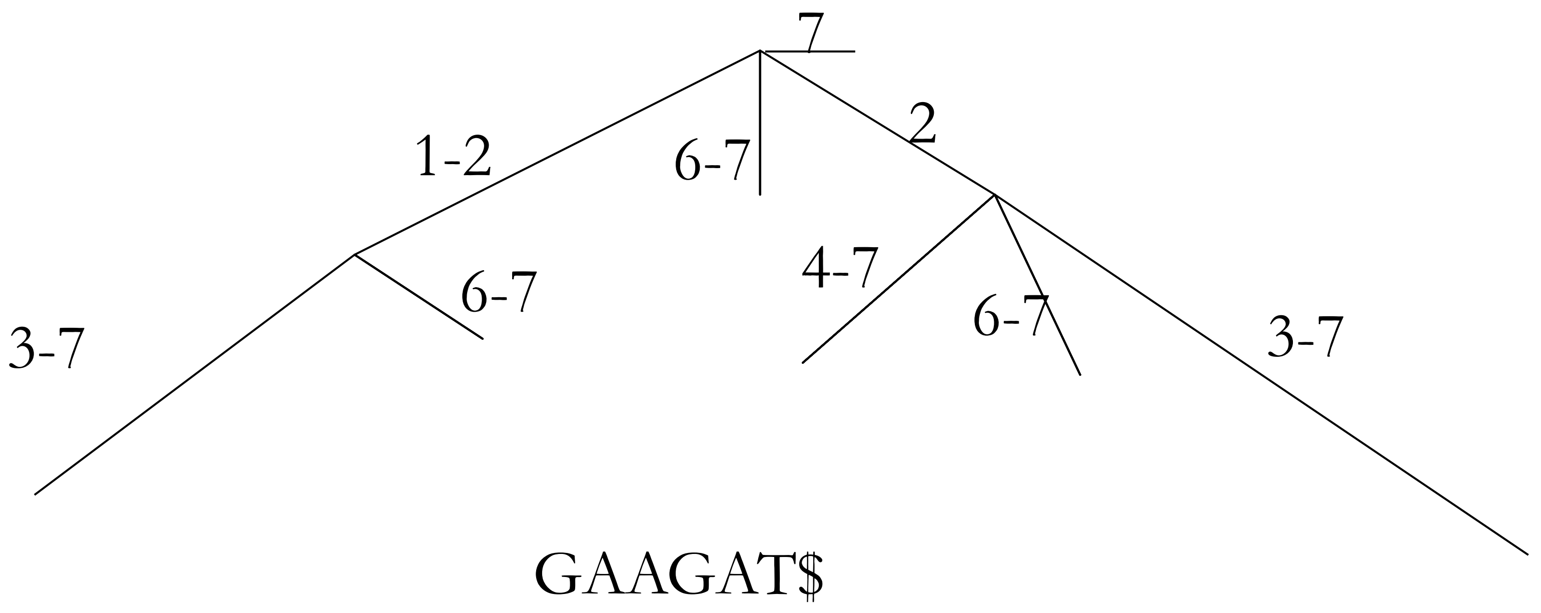
- An edge is labelled with a substring of the original string.
- A **node**'s label is the concatenation of all edge labels for the path leading to that node.
- The path from the root, *r*, to any leaf *x* is a suffix of the string S.
- Suppose there is a special "end-of-string" character, each suffix will end at the leaf.
- Each internal node has at least 2 children.
- Edge labels to the child nodes of an internal node start with different letters.

Application I. Search for a substring.

- Any substring of S is a **prefix** of a **suffix**.

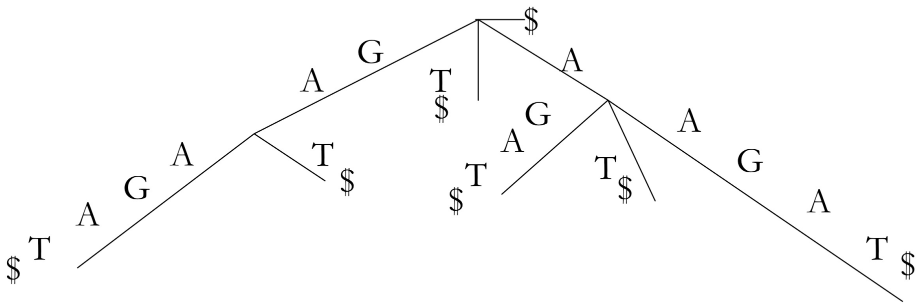
- Example of using this: Is the string x a substring of S?
  - Start at the root, and follow paths labelled by the characters of x. If you can get to the end of x, then yes, it is.

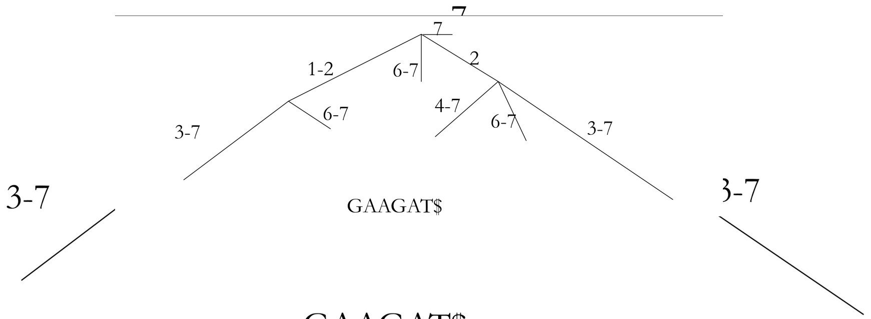
Linear Space Structure



- Each edge doesn't need to be labelled with a string, but just with starting and ending in the sequence.
- This is the same suffix tree as before, but in **linear space**.

- Here is the suffix tree for GAAGAT$

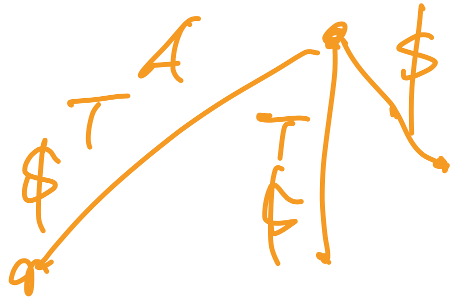




GAAGAT$

How to construct a suffix tree?

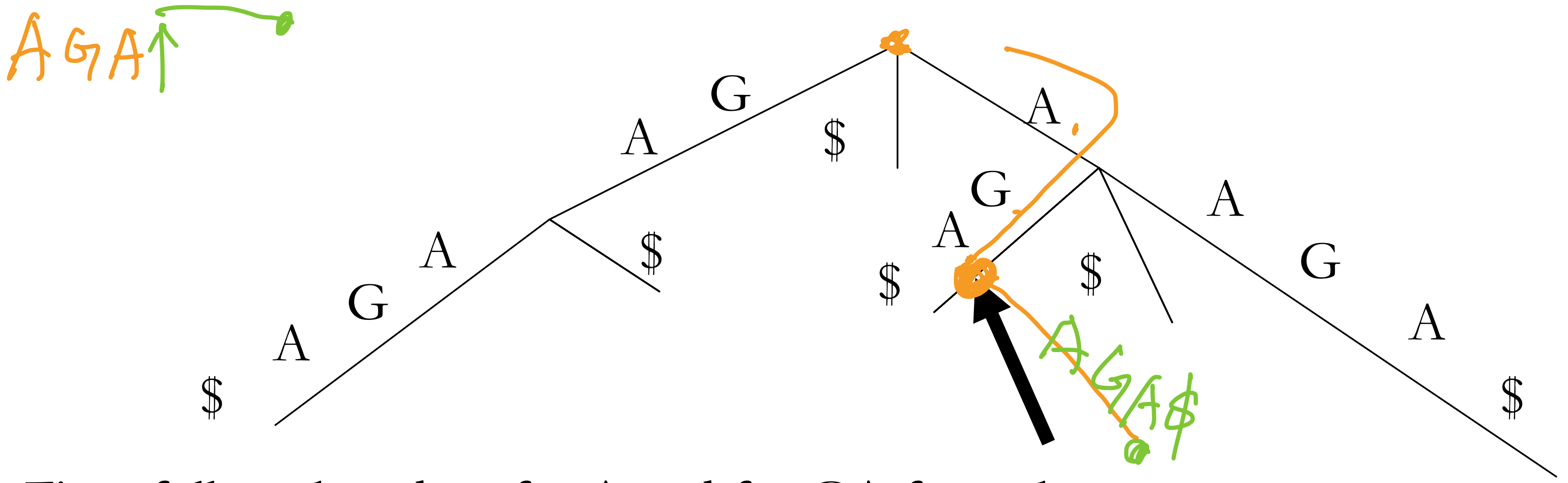
- There is a linear time algorithm to construct a suffix tree. (We will not study it.)
- We'll examine a quadratic-time algorithm (quite intuitive).
- The idea is to
  - Start with an empty tree.
  - Iteratively add more suffices into the tree (from shortest to longest).



A T $

One round

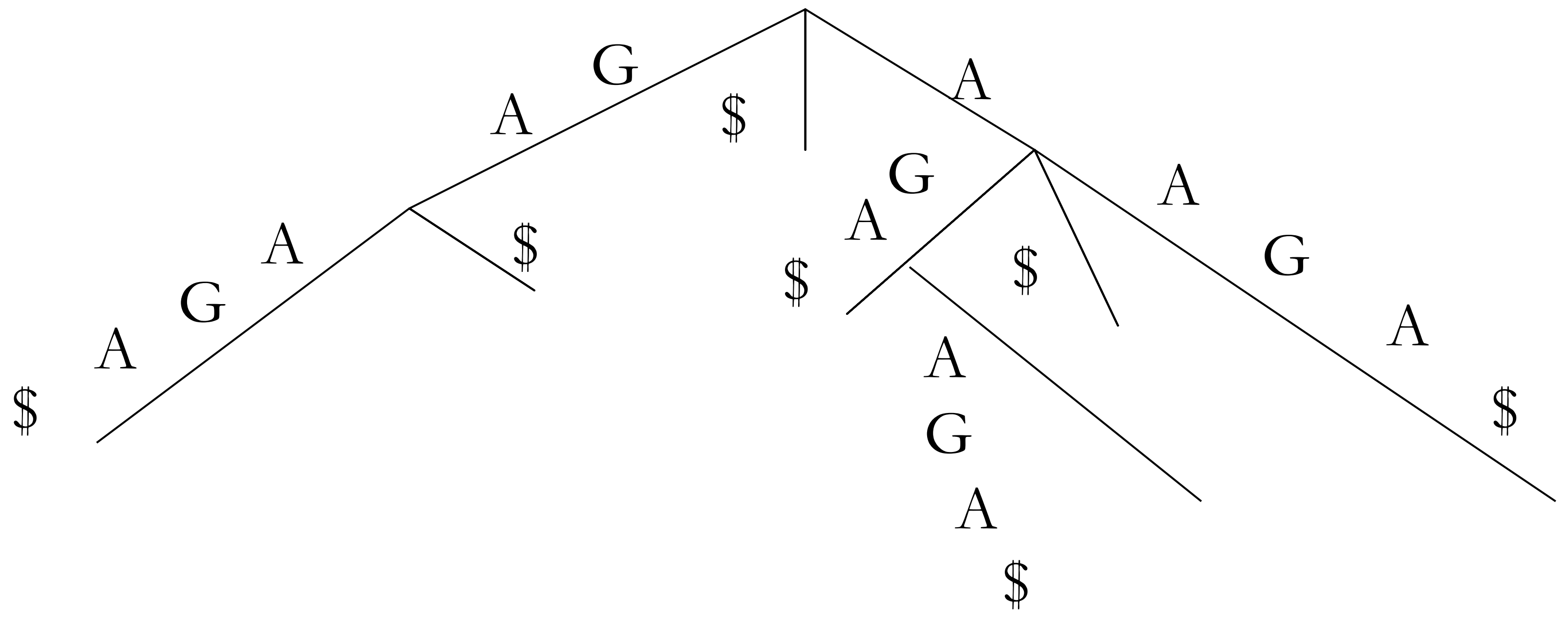
- Suppose the following is the suffix tree for A GAAGA$, add another suffix AGAAGA$.



- First, follow the edges for A and for GA from the root.
- Then split after the A since the only path in the tree is for $, and we have an A, instead.
- Add a new edge for AGA$.

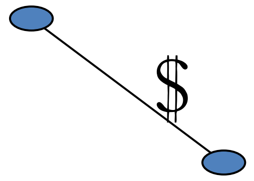
New tree

- This yields this new tree for AGAAGA$



Quadratic Time Construction

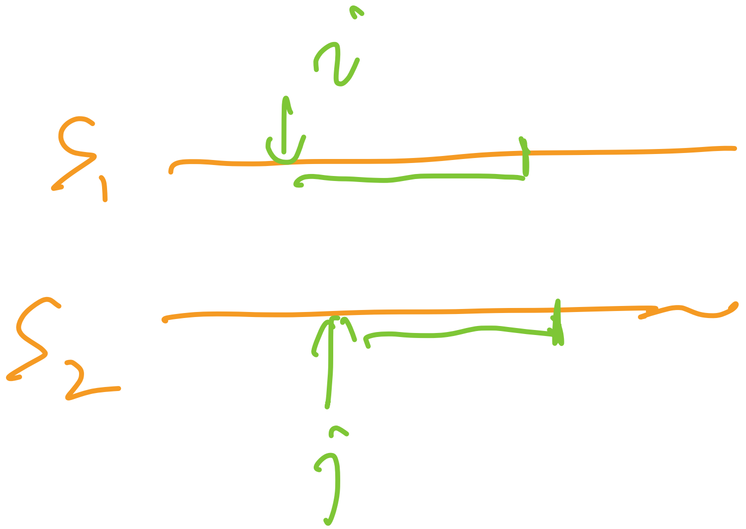
- Given: A string S of length *m* over a finite alphabet. The last character of S is a unique $ character.
- We'll build the suffix tree from right to left.
    - S[m..m], S[m-1..m], S[m-2..m], ……
- Begin with this tree:



- Then, for $i = m$ downto 1:
- Follow the letters of S[$i \ldots m$] along the edges of the tree T.
- When we reach a point where no path exists, break the current edge and add a new edge for what is left.
- Time complexity: $O(m^2)$. (Remember: The best algorithm has linear time.)

Application II: Longest Common Substring

- What's the longest substring common to both S_1 and S_2?
- Straightforward algorithm will try to compare all substrings of equal length. This takes cubic time.
- Can we do better?



match $= 1$

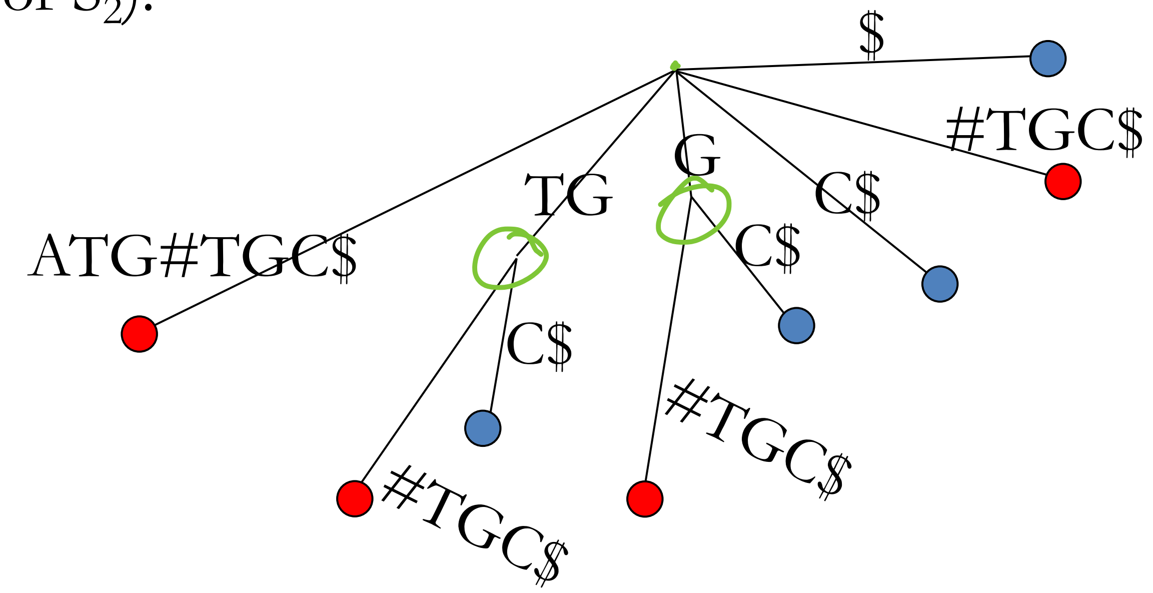
mismatch $= -\infty$

indel $= -\infty$

local alignment.

Longest Common Substring with Suffix Tree

- Build a suffix tree for $S=S_1\#S_2\$$, where # and $ are unique characters.
- All suffixes of S_1 end with an edge including $\#S_2\$$. So we can label whether a leaf belongs to S_1 or S_2
- Substrings are prefixes of suffixes, i.e. internal and leaf nodes of the tree.
- Each common substring is the prefix of at least two suffixes, each from an input string (S_1 or S_2).
- Longest?

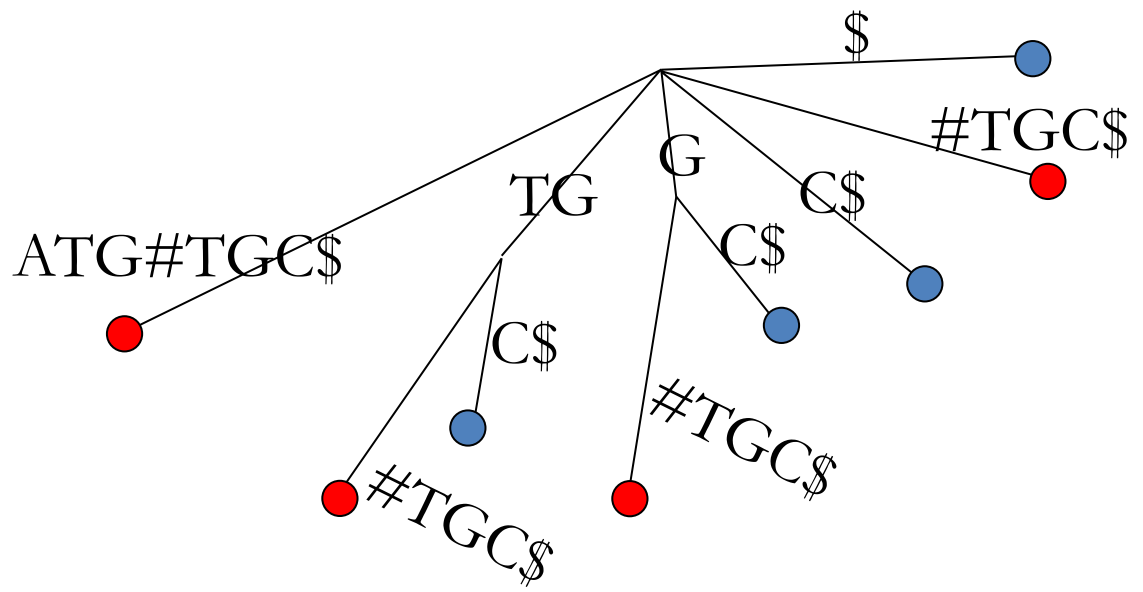


internal node
mixed color leaves.
common substring.

Example

ATG#TGC$

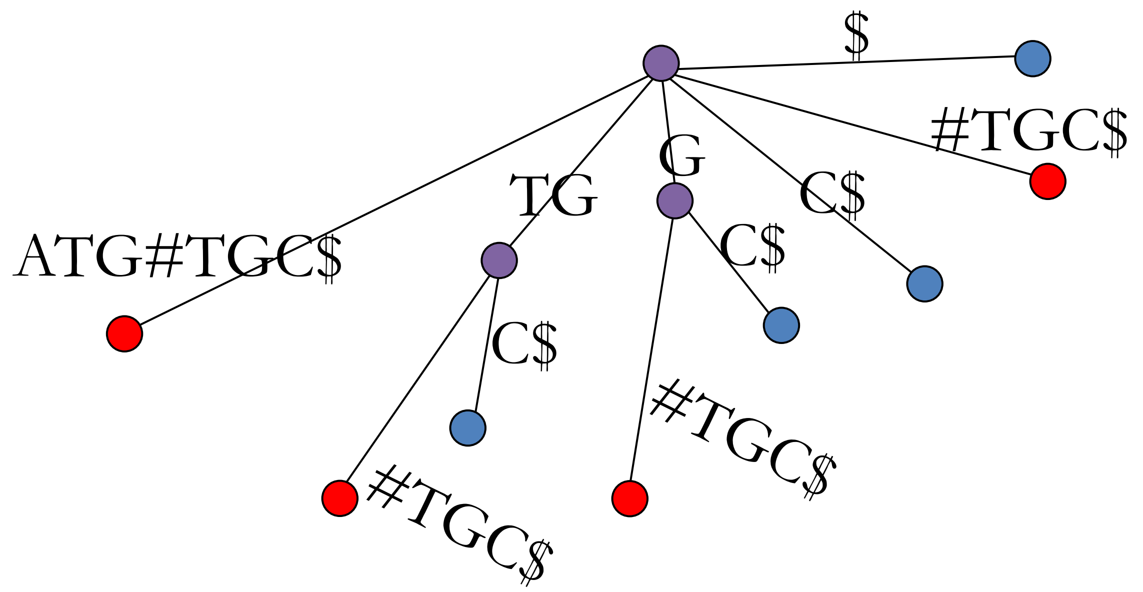
Step 1. Label leaves as red or blue, depending on whether it is a suffix starting in first or second string.



Example

ATG#TGC$

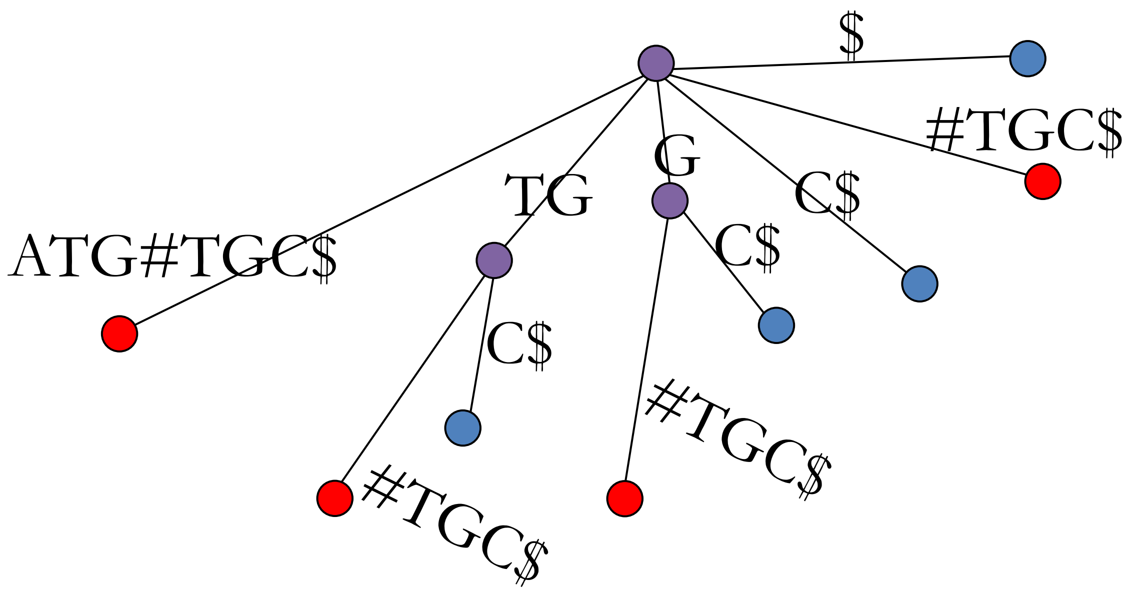
Step 2. In a bottom up order, label internal nodes. If all child nodes have the same color, label it with the same color; If not, label it with purple.



Example

ATG#TGC$

Step 3. Find the purple node with the longest path to the root.



Algorithm Summary

- 1. Build suffix tree of $S_1\#S_2\$$
- 2. Color all leaf nodes
  - red if *v*'s label is a substring of S_1
  - blue if it's a substring of S_2
- 3. Color all internal nodes from bottom up
  - red (or blue) if all child nodes are red (or blue)
  - purple if otherwise
- 4. Find the purple node with longest path label.
- Complexity: Linear time, linear space.
- Sketch proof of correctness:
  - Let t be the longest common substring. Follow the path label t starting from the root. The path can't stop in the middle of the edge – otherwise t is not the longest. Then the path has to stop at an internal node. And it has to be purple.

Application III: Maximal Unique Match

Maximal Unique Matches

- Given two strings, a MUM (Maximal Unique Match) is a string that occurs exactly once in each string, and is maximal (can't be extended either way and still be a match).
- E.g. ATGAATC vs. AGATC
    - AT is not. not unique
    - G is not. not maximal: GA is longer.
    - GA is a MUM.
    - ATC is a mum.

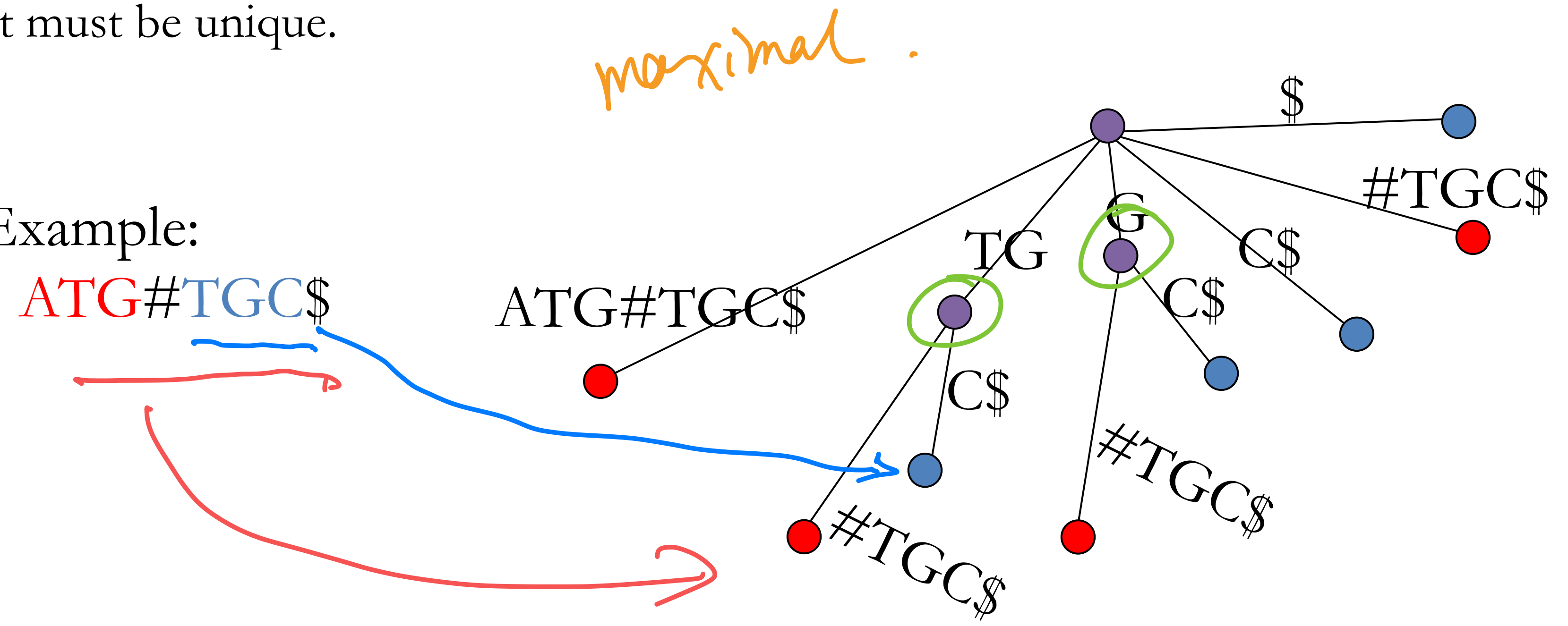
How to find mums?

- Build a suffix tree for $S_1\#S_2\$$
- Color the nodes as in the longest common substring algorithm.
- Each MUM must be a purple internal node that has exactly two leaf children: one red and one blue. ⇒ unique.
  - It is shared by the two strings.
  - It can't extend to the right by an additional letter and still be shared.
  - It must be unique.

maximal.

Example:

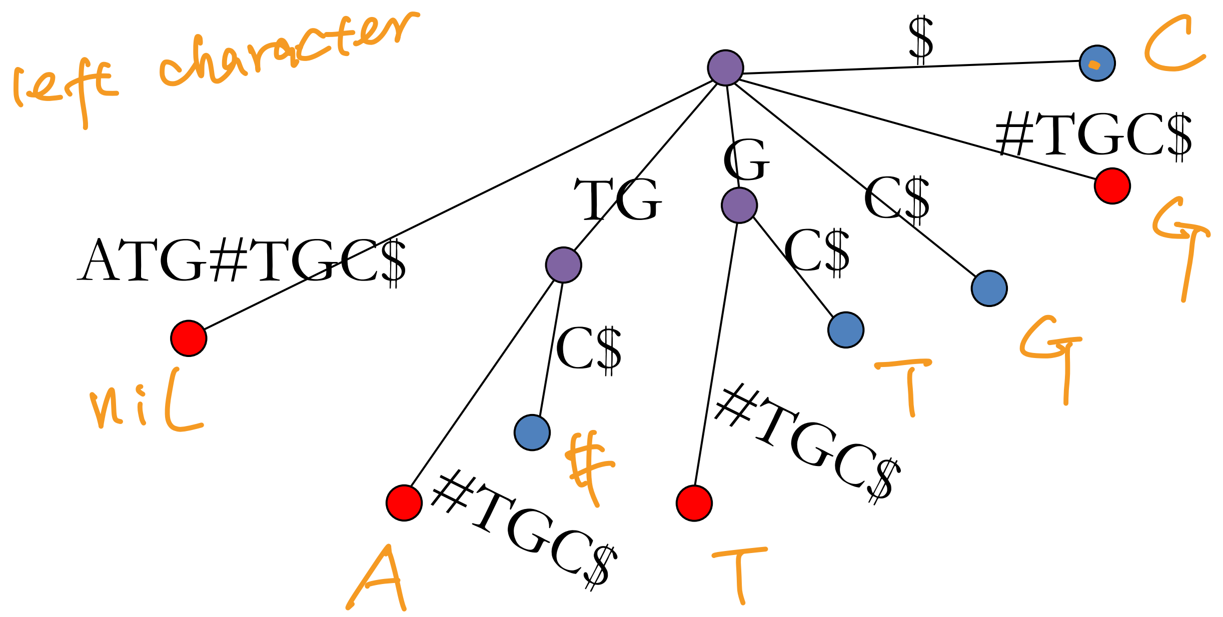
ATG#TGC$



How to find mums?

- But a purple internal node may not be a MUM: only because the two occurrences may still extend to the left.
  - Node G is not: For G's two occurrences, the left character are both T.
  - Node TG is: For TG's two occurrences, the left characters are A and #, respectively.
- But it is easy to compute the left character of each leaf
  - It is a suffix, and we know its path's starting position in the original string.

Example:
ATG#TGC$

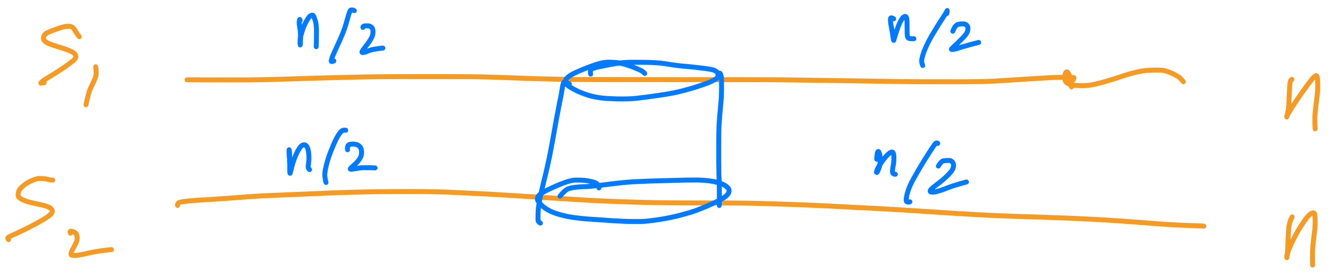


Summary

- Build a suffix tree for $S_1\#S_2\$$.
- For each leaf v, define left(v) be the letter at left of suffix v.
- Find the internal nodes that
  - Have exactly two child leaves
  - The two child leaves are two suffixes from S1 and from S2, respectively.
  - The two child leaves must have two different left characters.
- Linear time.
- After find all MUMs, use them as anchor to speed up global alignment.

MUMMER: Large-scale Global Alignment

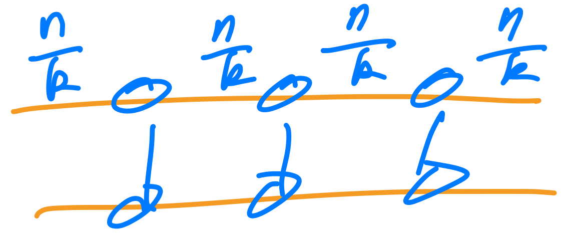
- Large-scale global alignment



- Idea:
- Pick some “anchors” through which the true alignment is very likely to fall.
- Align the regions between the anchors either recursively or just using classical global alignment tools.

$$\left(\frac{n}{2}\right)^2 + \left(\frac{n}{2}\right)^2 = \frac{n^2}{2} \quad O(n^2).$$

- MUMs are good anchors: maximal, unique, match.
- First program that does so: MUMMER by Delcher et al.



$$k \cdot \left(\frac{n}{k}\right)^2 = \frac{n^2}{k}$$

Quick Note on Suffix Array

- Suffix tree is not a compact data structure.
  - A lot of pointers
- Gene Myers and Udi Manber (VP enginnering, Google) proposed suffix array.
- A suffix array stores the positions in a string. Each position is an integer so this is a length n integer array.
- Each position corresponds to a suffix starting at this position.
- The suffix array is sorted according to the string order of the corresponding suffixes.

Suffix Array

$O(n^2 \cdot \log n)$ for sorting.

$O(m \cdot \log n)$ for query.

- AGAAGAT

all suffixes

1 = AGAAGAT
2 = GAAGAT
3 = AAGAT
4 = AGAT
5 = GAT
6 = AT
7 = T

3 = AAGAT
1 = AGAAGAT
4 = AGAT
6 = AT
2 = GAAGAT
5 = GAT
7 = T

AG.

3, 1, 4, 6, 2, 5, 7

String Matching

- Binary search to find substring of length m.
  - O(m log n) if implemented straightforwardly
  - O(m + log n) if with an auxiliary data structure called longest common prefix (LCP) array. We do not study this but you should be aware of this fact.