String Algorithm

Jeong-Hyeon Choi

School of Informatics,
Indiana University, USA
jeochoi@indiana.edu

Outline

Notation

- Alphabet is a set of characters,
  e.g., $\Sigma = \{A, C, G, T\}$.
- String is a
  consecutive sequence of characters over the alphabet, e.g., `ACGTACG`
- The length of stirng x, denoted by $|x|$, is the number of characters in x.
- xy is a concatenation of x and y.
- y (z) is a prefix (suffix) of x if $x = yz$ for some y and z.
- z is a suffix of x if $x = yz$ for some y and z.
- x_i^j or $x[i..j]$ is a substring of x that starts at position i and ens at position j of x.
- x^j (x_i) is a prefix (suffix) of x that ends (starts) at position j (i).

String Matching Problem

Definition

Given a text string $T[1..m]$ and a pattern string $P[1..n]$, find all occurrences of P in T

Example

P=`GACTTA`, T=`ATCGACTTACTGTTA`

Applications:

- Text editor (vi, emacs, word perfect)
- database search, etc.

Variations:

- Finding Period, Test Tandem Repeat, Finding Longest Common Substrings
- Approximate string matching (spelling correction),
- Multiple keyword search(library)

Pattern Matching Algorithms

- Naïve algorithm
- KMP (Knuth-Morris-Pratt)
- BM (Boyer-Moore)
- KR (Karp-Rabin)

Online resource:

- Exact String Matching Algorithms [1]
- SAGAReP [2]
- Demonstration of Naive, KMP, and BM pattern matching algorithms, and their variations [3]

Naïve Algorithm

Example: ▸ Click

Complexity

	worst	best
T	AAAAAAAT	AAAAAAAA
P	AAAT	TAAA
	$O(mn)$	$O(m)$

Average case ?

Average Case

Assumption

- Let $q = |\Sigma|$
- For pattern, q^n strings are equally likely.
- For text, q^m strings are equally likely.

Probability of a match:

- There are q text characters with each probability $1/q$.
- For each text char, the probability that pattern character matches it is $(1/q)^2$.
- So the probability of a match is $q(1/q)^2 = 1/q$.

Average Case

Let X_i be the random variable that represents the number of comparisons for position i of the text, i.e., $X_i = 1, 2, \cdots, n$
The expected number of comparisons at position i of the text is

$$\begin{aligned} E[X_i] &= \sum_{x=1}^{n} xP(X_i = x) = \sum_{x=1}^{n} x(\frac{1}{q})^{x-1}(1 - \frac{1}{q}) \\ &= (1 - \frac{1}{q}) \sum_{x=1}^{n} x(\frac{1}{q})^{x-1} < (1 - \frac{1}{q})/(1 - \frac{1}{q})^2 \\ &= \frac{q}{q-1} < 2 \end{aligned}$$

Average Case

The expected number of comparisons for all position is

$$\begin{aligned} E[\sum_{i=1}^{m-n+1} Xi] &= (m-n+1)\ E[X_i] \\ &< 2(m-n+1) = O(m-n) \end{aligned}$$

Efficient Pattern Matching

Key idea:

- Preprocess pattern P, and produce something (finite automata)
- Search text T

Recall the brute force algorithm.

- KMP (Knuth-Morris-Pratt)
- BM (Boyer-Moore)
- KR (Karp-Rabin)
- ...

Pattern Retrieval Problem

Idea:

- Preprocess the text : $O(m)$ time
- Searching : $O(n + occ)$ time

Index data structure

- Suffix Trie
- Suffix Tree
- Suffix Array
- String B-tree

Suffix

Definition

Suffix x_i is a substring of x that starts at position i and ends at position $|x|$.

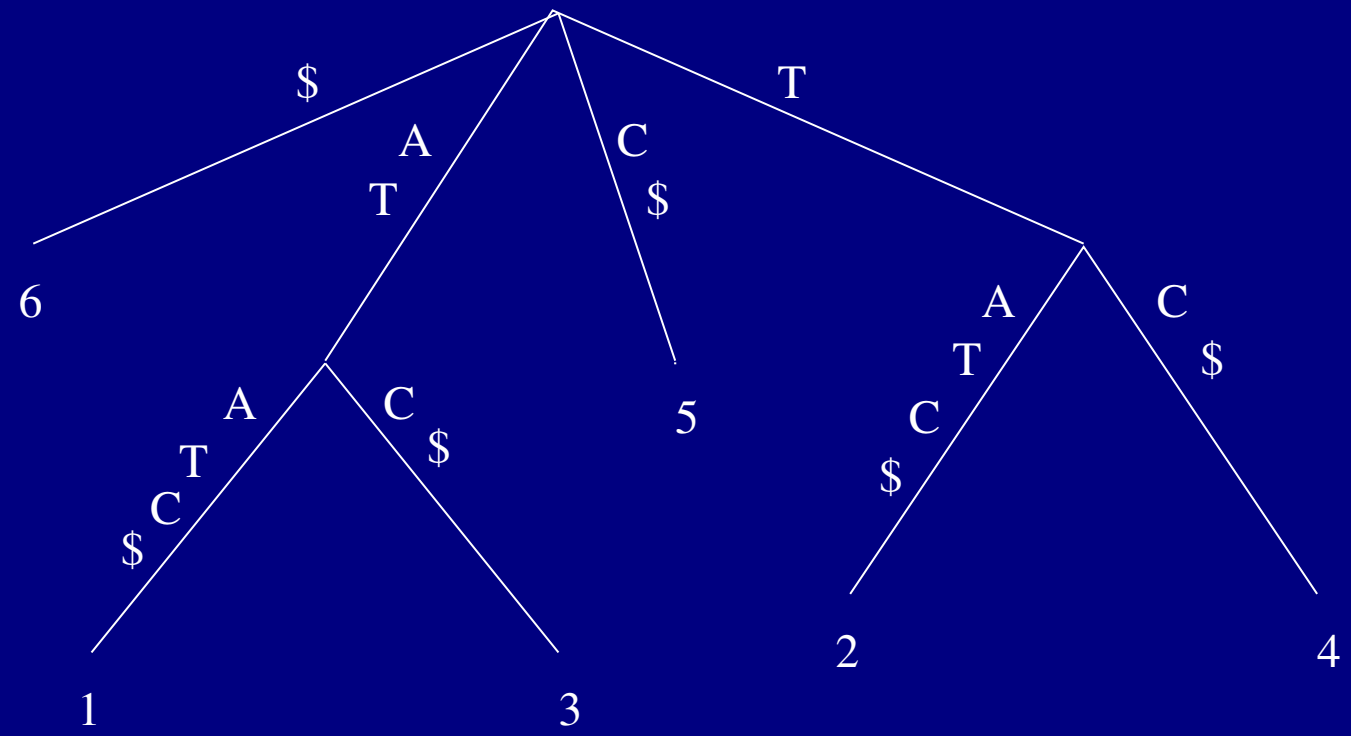
Example

Given $x =$ ATATC,

$$
\begin{aligned}
x_1 &= \texttt{ATATC\#} \\
x_2 &= \texttt{TATC\#} \\
x_3 &= \texttt{ATC\#} \\
x_4 &= \texttt{TC\#} \\
x_5 &= \texttt{C\#} \\
x_6 &= \texttt{\#}
\end{aligned}
$$

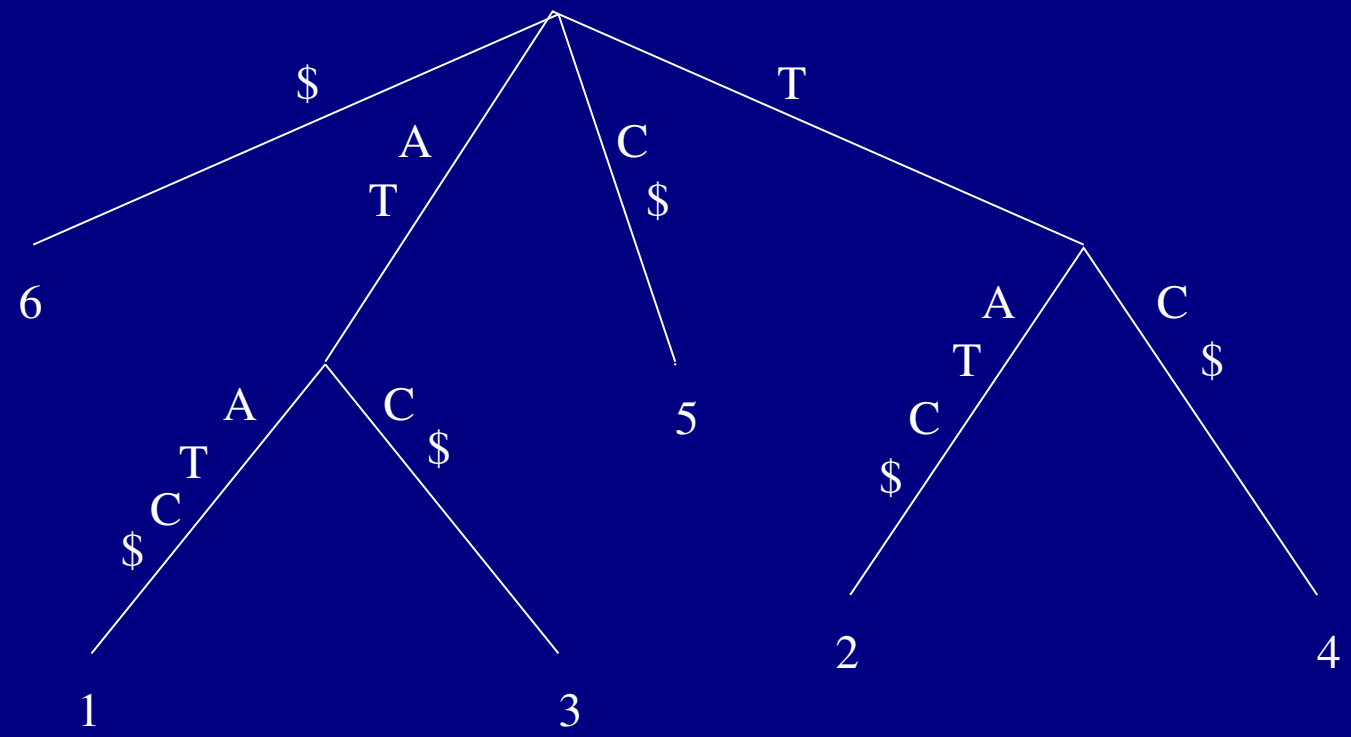
Suffix Tree

The compacted trie of all suffixes of a string, e.g., `ATATC`



Suffix Tree

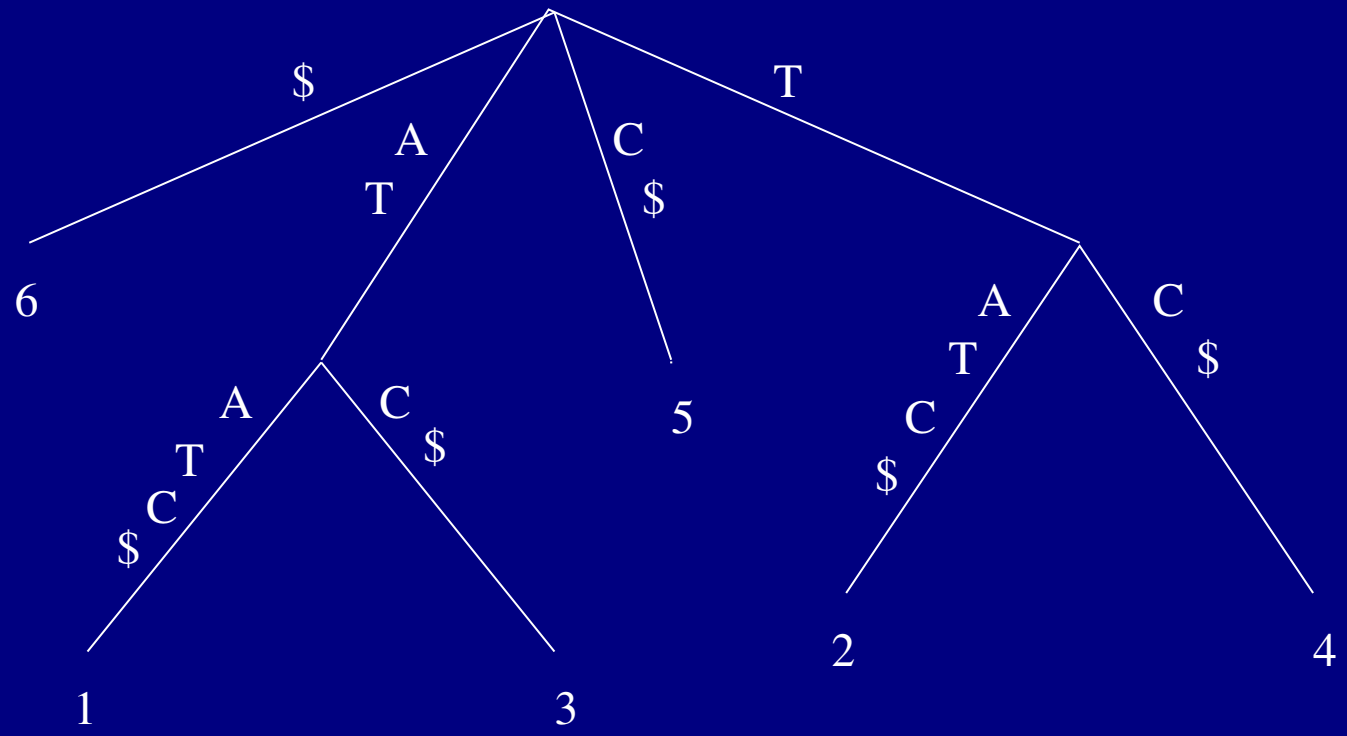
The compacted trie of all suffixes of a string, e.g., `ATATC`



- Each leaf node is numbered to suffix number.

Suffix Tree

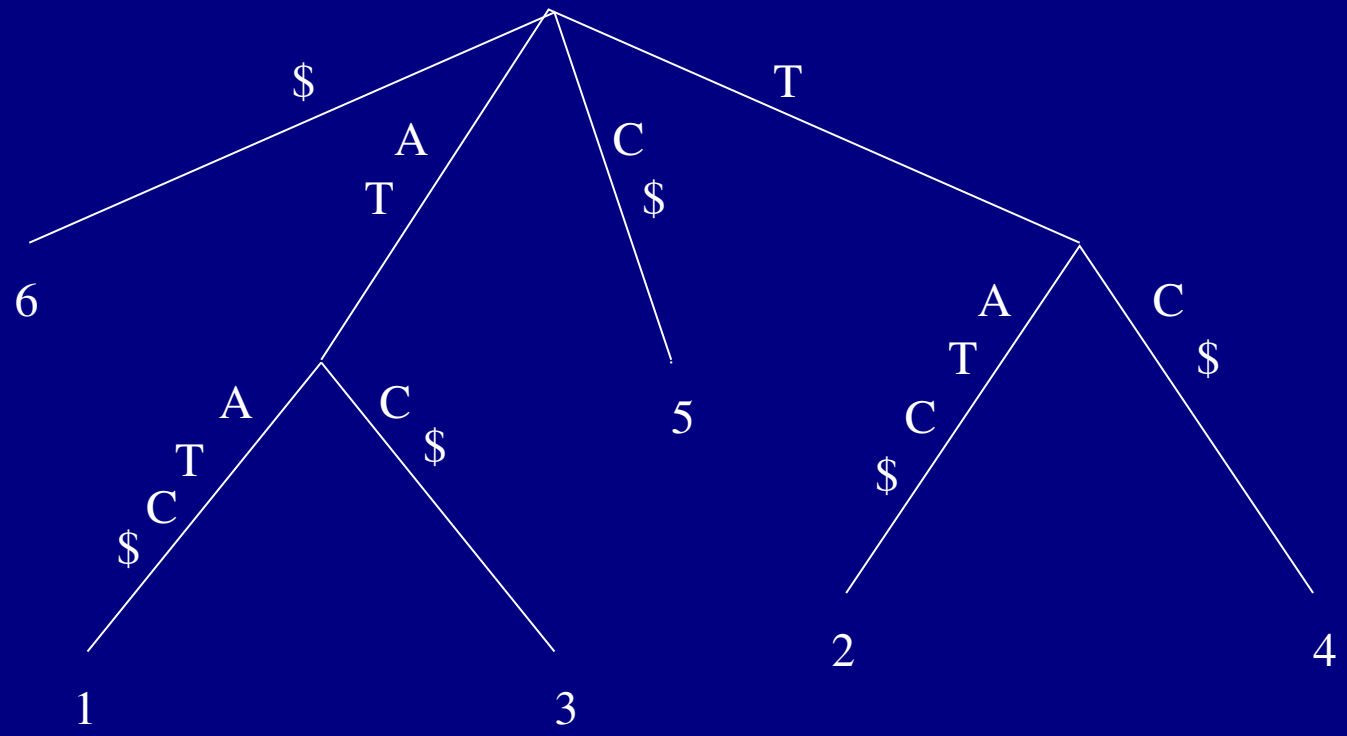
The compacted trie of all suffixes of a string, e.g., ATATC



- Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty substring of T.

Suffix Tree

The compacted trie of all suffixes of a string, e.g., `ATATC`

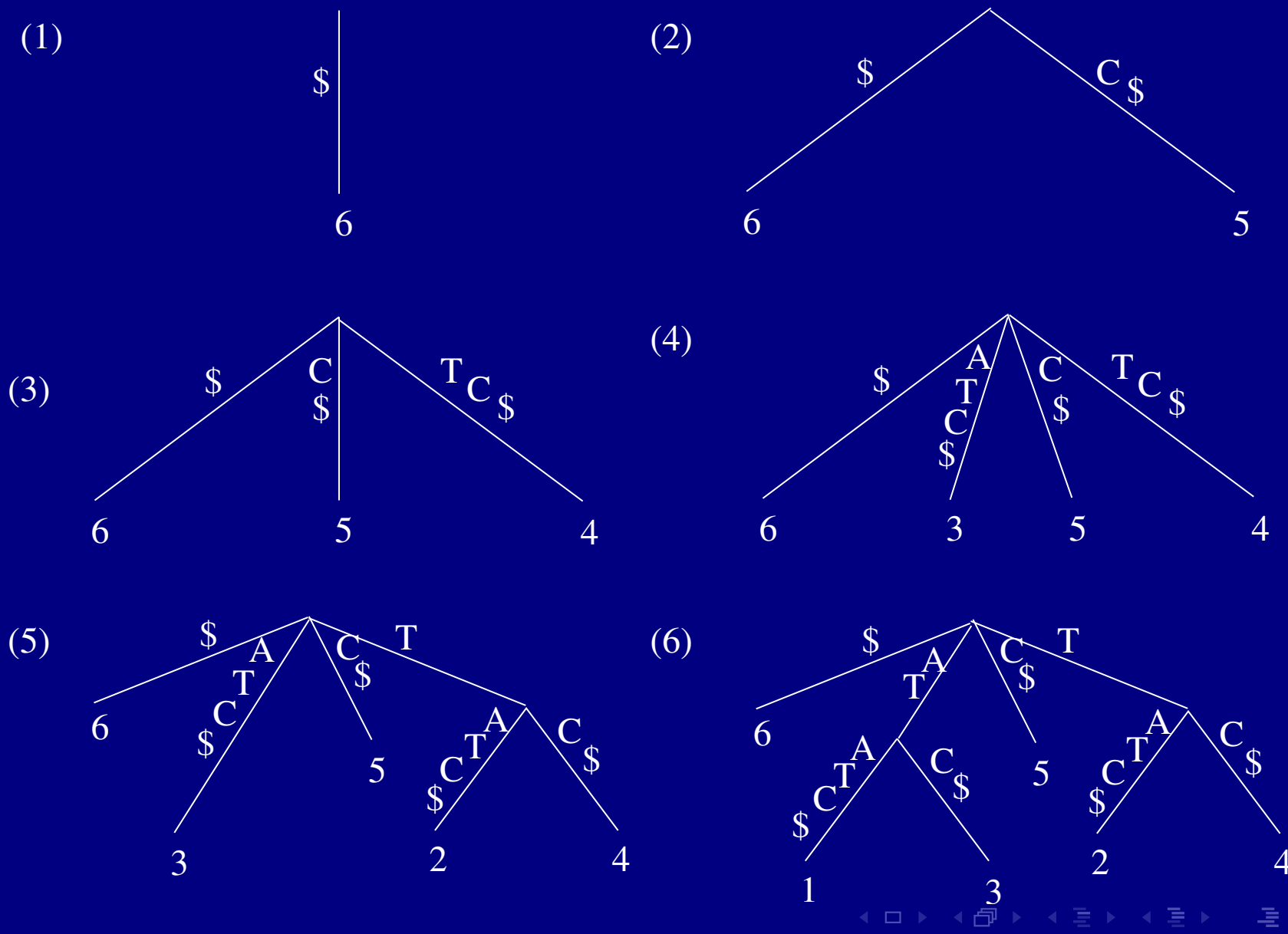


- The concatenation of the edge-labels on the path from the root to leaf i exactly spell out the suffix of T that starts at position i

Suffix Tree: Strategy

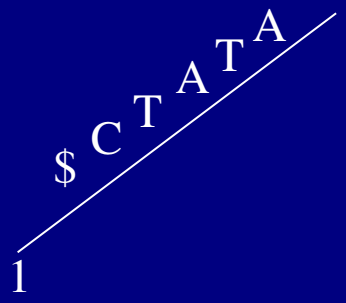
1. Preprocessing
   - Constant size : $O(m)\log|\Sigma|$ time and $O(m)$ space
     - Weiner (1973) [4]
     - McCreight (1976) [5]
     - Ukkonen (1995) [6]
   - Integer alphabet : $O(m)$ time
     - Farach (1997)
2. Searching: $O(n)\log|\Sigma| + occ$ time

Weiner's Algorithm

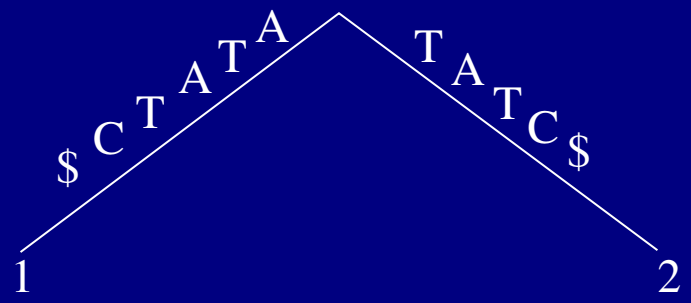


McCreight's Algorithm

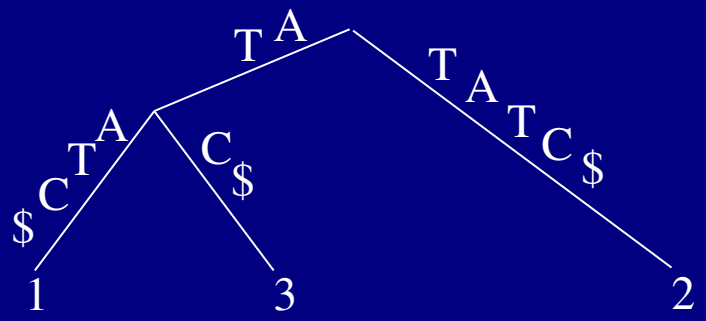
(1)



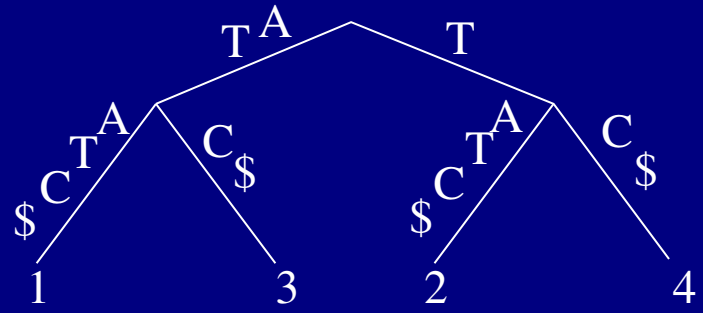
(2)



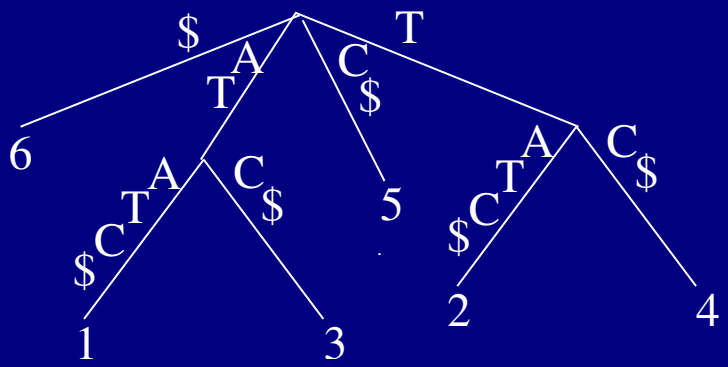
(3)



(4)



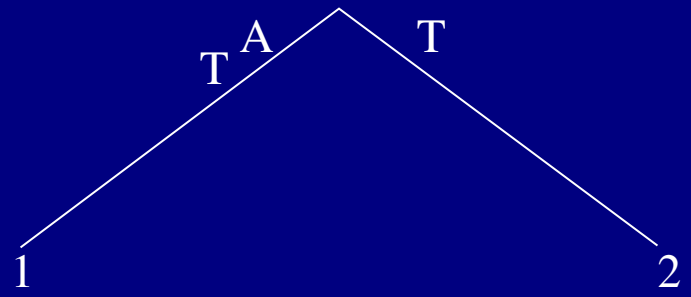
(5)



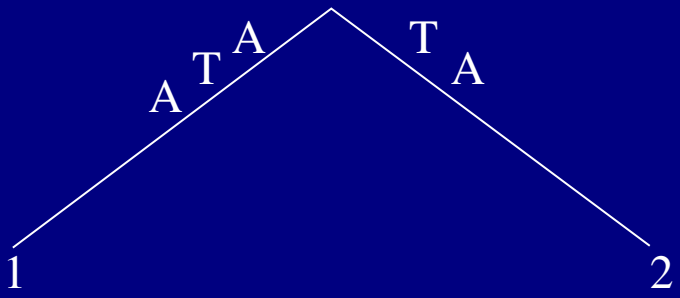
Ukkonen's Algorithm

(1)

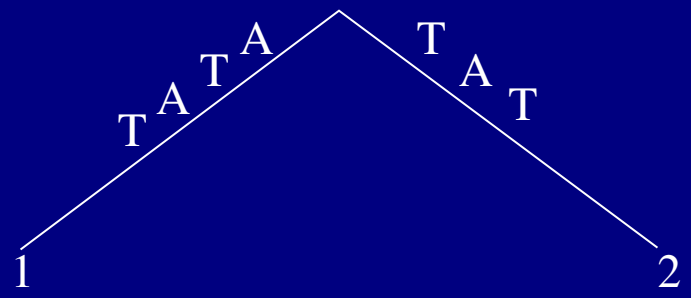
(2)



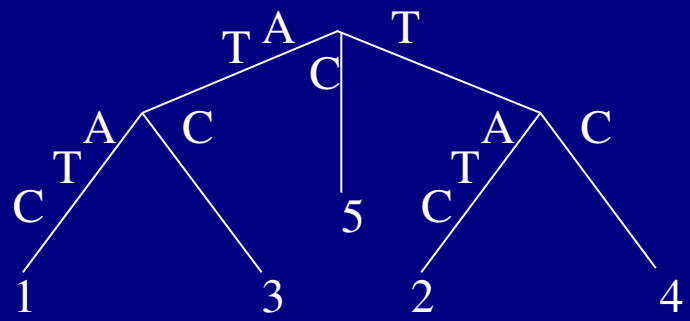
(3)



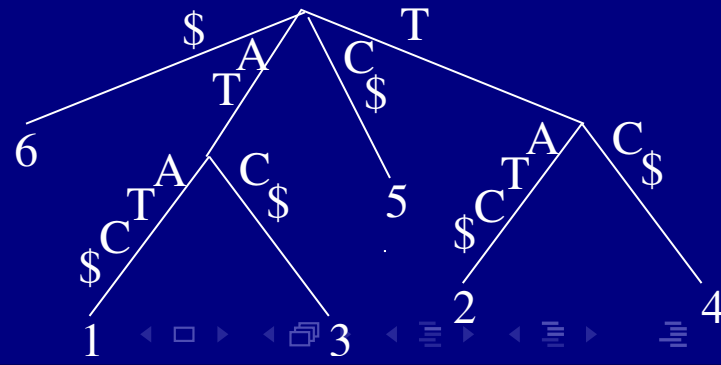
(4)



(5)



(6)

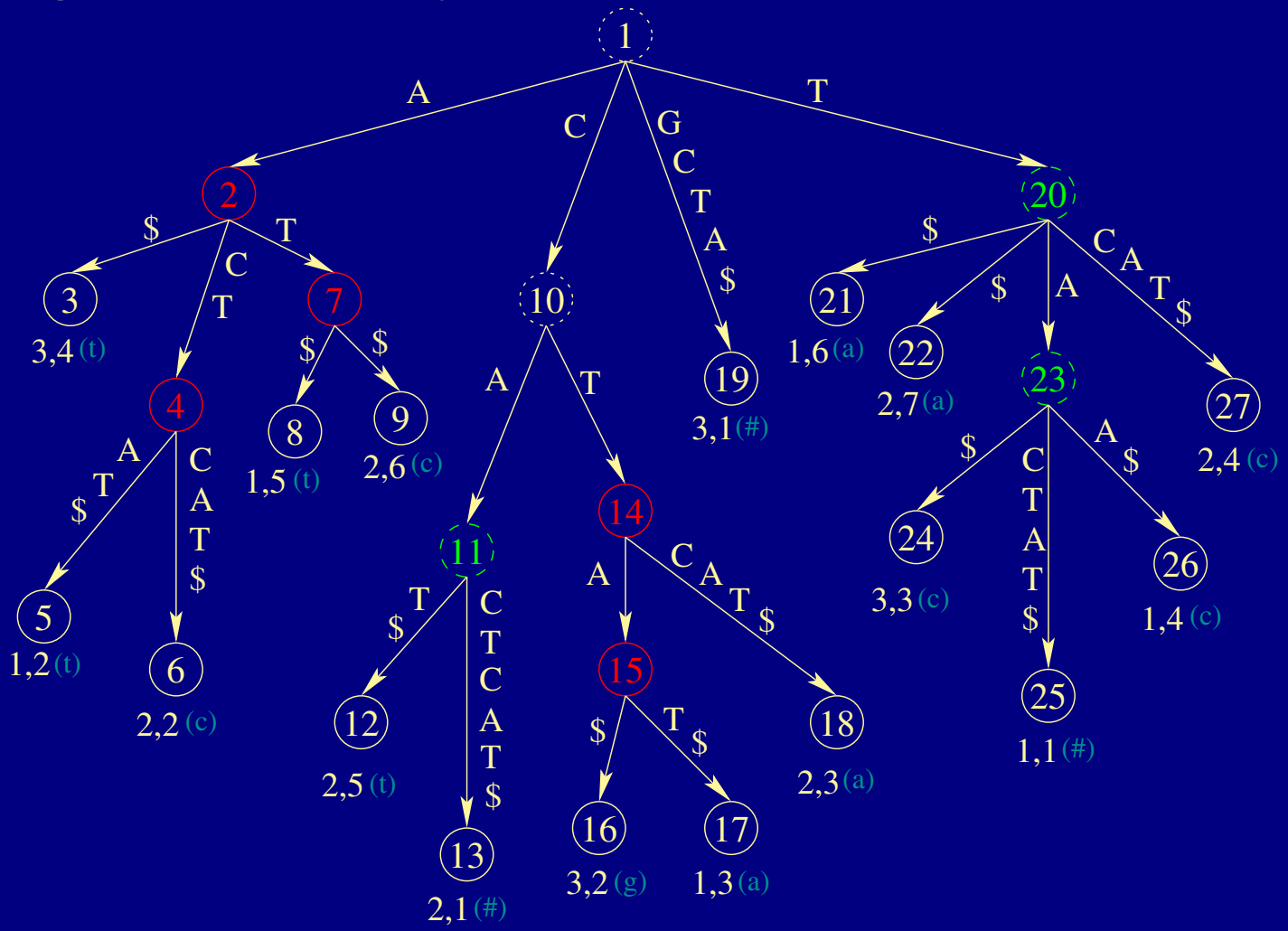


Online Resource for Suffix Tree

- Suffix Trees in Computational Biology [7]
- Growing A Suffix Tree [8]
- Rutgers Univ. [9]
- Suffix Tree Construction Applet [10]

Generalized Suffix Tree

The suffix tree of concatenated string of two or more strings
e.g., $x =$ TACTAT, $y =$ CACTCAT, $z =$ GCTA



Application of Suffix Tree [11]

- Exact pattern matching
- Exact pattern set matchging
- Database search (Substring)
- Longest common prefix
- Longest common substring
- Common substrings of length at least k
- Longest k-common substring
- Repeated substring
  - Maximal repeats
  - Supermaximal repeats
  - Maximal pairs
- Maximal Exact Matches

Exact Pattern Matching

▸ Example

Input

- Pattern P of length n
- Text T of length m

Output

- Position of all occurrences of P in T

Solution

- Preprocess to create suffix tree for T
  - $O(m)$ time, $O(m)$ space
- Maximally match P in suffix tree
- Output all leaf positions below match point
  - $O(n + occ)$ time where occ is number of matches

Exact Set Matching

Example

Input

- Set of patterns $\{P_i\}$ of total length n
- Text T of length m

Output

- Position of all occurrences of each pattern P_i in T

Solution

- Preprocess to create suffix tree for T
  - $O(m)$ time, $O(m)$ space
- Maximally match each P_i in suffix tree
- Output all leaf positions below match point
  - $O(n + occ)$ time where occ is number of total matches

Comparison with Aho-Corasick

Aho-Corasick

- $O(n)$ preprocess time and space
  - to build keyword tree of set of patterns P
- $O(m + occ)$ search time

Suffix Tree Approach

- $O(m)$ preprocess time and space
  - to build suffix tree of T
- $O(n + occ)$ search time
- Using matching statistics to be defined, can make this tradeoff similar to that of Aho-Corasick

Database Search (Substring)

▸ Example

Input

- Set of texts $\{T_i\}$ of total length m
- Pattern P of length n

Output

- Position of all occurrences of P in each text T_i

Solution

- Preprocess to create generalized suffix tree for $\{T_i\}$
  - $O(m)$ time, $O(m)$ space
- Maximally match P in generalized suffix tree
- Output all leaf positions below match point
  - $O(n + occ)$ time where occ is number of total matches

Longest Common Prefix (LCP) ▸ Example

Input

- Suffix S_i of string S and suffix T_j of string T

Output

- Longest common prefix of S_i and T_j

Solution

- Simple method: compare S_i to T_j
  - $O(\ell)$ where ℓ is the length of LCP
- Preprocess to create generalized suffix tree for $\{S, T\}$
  - $O(m)$ time, $O(m)$ space
- Preprocess to build efficient data structure to query lowest common ancestor (LCA) between two leaf nodes
  - $O(m)$ time, $O(m)$ space
- Query lowest common ancestor (LCA) between two leaf nodes for S_i and T_j
  - $O(1)$ time

Longest Common Substring (LCS) ▸ Example

Input

- Strings S and T of total length m

Output

- Longest common substring of S and T (and position in S and T)

Solution

- Preprocess to create generalized suffix tree for $\{S, T\}$
- Mark each node by whether or not its subtree contains a leaf node of S, T, or both
  - Simple tree traversal algorithm to do this: $O(m)$
- Path label of node with greatest string depth is the longest common substring of S and T

Common Substrings of length at least k ▸ Example

Input

- Strings S and T
- Integer k

Output

- All substrings of S and T (and position in S and T) of length at least k

Solution

- Same as previous problem
- Look for all nodes with 2 leaf labels of string depth at least k

Longest k-Common Substring ▸ Example

Definition

For a given set of K strings, $l(j)$ for $2 \leq j \leq K$ is the length of the longest substring common to at least j of the K strings.

Example

{`sanddollar`, `sandlot`, `handler`, `grand`, `pantry`}

j	$l(j)$	one string
2	4	sand
3	3	and
4	3	and
5	2	an

Longest k-Common Substring ▸ Example (cont.)

Input

- Strings $S_1, \cdots, S_K$ (total length m)

Output

- $l(j)$ (and pointers to substrings) for $2 \leq j \leq K$

Solution

- Build a generalized suffix tree for the K strings.
- $C(v)$: the number of distinct leaf string identifiers in the subtree of node v.
- $V(j)$: string-depth of the deepest node v encountered with $|C(v)| = j$.
- Do depth-first traversal of tree to find $K - 1$ values of V. For a node v,
  - calculate $C(v)$,

Longest k-Common Substring ▸ Example (cont.)

  - change $V(C(v))$ to the string-depth of v if $V(C(v))$ is less than that of v.
- Scan V from the largest to smallest index, setting $V(j)$ to $V(j+1)$ if $V(k)$ is empty or $V(k) < V(k+1)$.
  - $O(K)$ time
- Totally $O(Km)$ time and $O(K+m)$ space if $C(v)$ is calculated in $O(K)$.

How to calculate $C(v)$.

- $B(v)$: a set of distinct leaf string identifiers in the subtree of node v, i.e., $C(v) = |B(v)|$.
- leaf node: $B(v) = \{i\}$ where i is the string identifier of v.
- branch node: $B(v) = \cup_w B(w)$ where w is child node of v.
- $O(K)$ time, $O(K)$ space.

Longest k-Common Substring (Linear Time) ▸ Example

Observation

- $S(v)$: the total number of leaves in subtree of v, and $S(v) \geq C(v)$.
- $U(v)$: a correlation factor which counts how many duplicate suffixes from the same string occur in v's subtree.
- Calculate $S(v)$ and $U(v)$ since it is difficult to calculate $C(v)$ and $C(v) = S(v) - U(v)$.
  e.g., $C(20) = 6 - 3$.
- $n_i(v)$: the number of leaves with identifier i in v's subtree, so $U(v) = \sum_{i:n_i(v)>0}(n_i(v) - 1)$.
  $U(20) = (3 - 1) + (2 - 1)$ for $i = 1, 2$.

Longest k-Common Substring (Linear Time) ▸ Example (cont.)

- L_i: the list of leaves with identifier i, in increasing order of depth-first travsal.
  e.g.,
  $L_1 = \{5, 8, 17, 21, 25, 26\}$,
  $L_2 = \{6, 9, 12, 13, 18, 22, 27\}$,
  $L_3 = \{3, 16, 19, 24\}$.
- If we compute the LCA for each consecutive pair of leaves in L_i, then for any node v, exactly $n_i(v) - 1$ of the computed LCAs will lie in the subtree of v.
  e.g., for $v = 20$,
  $L_1 : LCA(21, 25), LCA(25, 26)$,
  $L_2 : LCA(22, 27)$,
  L_3 : none,
  so $U(20) = 3$.

Longest k-Common Substring (Linear Time) ▸ Example (cont.)

Strategy

- Number the leaves as they are encountered in a depth-first traversal of tree.
- For each string identifier i,
  extract the ordered list L_i of leaves with identifier i,
  compute the LCA of each consecutive pair of leaves in L_i,
  and increment $h(w)$ by one each time that w is the computed LCA.
- With a bottom-up traversal of the tree,
  compute $S(v)$ and $U(v) += \sum_{i:n_i(v)>0}(n_i(v)-1) = \sum h(w)$
  where w is a child node of v.
- Set $C(v) = S(v) - U(v)$ for each node v.

Maximal Repeats ▸ Example

Definition

A maximal repeat α is a substring in S that is the substring defined by a maximal pair of S.

Example

S = `xabcyiiizabcqabcyrxar`

`abc` and `abcy` are maximal repeats of S.

`abc` is represented only once.

Maximal Repeats ▸ Example (cont.)

Lemma

If α is a maximal repeat in S, then α is the path-label of an internal node v in suffix tree. That is, a maximal repeat does not end in the middle of an edge and next character after α is distinct.

Theorem

There are at most m maximal repeats in any string of length m since there are m leaves and internal nodes except the root have at least two children.

Maximal Repeats ▸ Example (cont.)

Definition

Character $S[i-1]$ is the left character of i. The left character of a leaf of a suffix tree is the left character of the suffix position represented by that leaf. A node v is called *left diverse* if at least 2 leaves in the subtree of v have different left characters.

Theorem

String α labeling the path to an internal node v is a maximal repeat if and only if v is left diverse.

Input

- A single string S of lenth m

Output

- The set of maximal repeats of S

Maximal Repeats ▸ Example (cont.)

Solution

- Construct suffix tree for S.
- During bottom up traversal of tree, compute left diverse of v or a left character label of a node v.
  - Leaf node: label leaves with their left character.
  - Internal node:
    - If any child is left diverse, so is v.
    - If two children have different left character labels, v is left diverse.
    - Otherwise, take on left character value of children.
    - $O(|v|)$ time where $|v|$ is the number of children.

  If v is left diverse, then the path label to v is a maximal repeat.
- Totally $O(m)$ time, $O(m)$ space.

Supermaximal Repeats ▸ Example

Definition

A supermaximal repeat α is a maximal repeat of S that never occurs as a substring of another maximal repeat of S

Example

S = `xabcyiiizabcqabcyrxar`

`abcy` is a supermaximal repeat of S.

`abc` is NOT a supermaximal repeat of S.

Theorem

A left diverse node v represents a supermaximal repeat if and only if all of its children are leaves, and each has a distinct left character.

Supermaximal Repeats ▸ Example (cont.)

Input

- A single string S of lenth m

Output

- The set of supermaximal repeats of S

Solution

- Same as maximal repeat problem.
- Look for all nodes whose children are leaves and each has a distinct left character.

Maximal Pair ▸ Example

Definition

A maximal pair in S, denoted by (p_1, p_2, ℓ) is a pair of identical substrings α and β in S such that the character to the immediate left (right) of α is different than the character to the immediate left (right) of β.

Example

S = xabcyiiizabcqabcyrxar
12345678901234567890

(2, 10, 3) and (10, 14, 3) are a maximal pairs

(2, 14, 3) is NOT a maximal pair, but (2, 14, 4) is a maximal pair

Note positions 2 and 14 are the start positions of two distinct maximal pairs

Maximal Pair ▸ Example (cont.)

Input

- A single string S of lenth m

Output

- The set of maximal pairs of S

Solution

- Build a suffix tree for S
- $L_x(v)$: the list of leaf numbers in the subtree of node v with a left character x.
  e.g., $L_a(10) = \{17, 18\}, L_t(11) = \{12\}$.
- Do bottom up traversal of tree.
  For a node v, create a linked lists $L(v)$ indexed by all left character.
  - Leaf node: create $L_x(v) = \{i\}$ where x is a left character and i is a suffix number for v.

Maximal Pair ▸ Example (cont.)

- Internal node:
  - $C(v)$: a set of children of v.
  - $D(v)$: a set of left characters of leaves in v's subtree.
  - For each $x \in D(v)$ and each $w \in C(v)$, do cartesian product of $L_x(w)$ with $L_{x'}(w')$ for $x' \neq x$ and $w' \neq w$.
  - Any pair in the list gives the starting positions of a maximal pair.
  - Finally, for each x, compute $L_x(v)$ by union of $L_x(w)$ for every w.
- Let k' be the number of maximal pairs from cartesian product, then cartesian product takes $O(k')$ time.
- union can be done in $O(|\Sigma|)$ time using linked list.

- Totally $O(m + k)$ time and $O(m)$ space where k is the number of maximal pairs.

Maximal Exact Matches ▸ Example

Definition

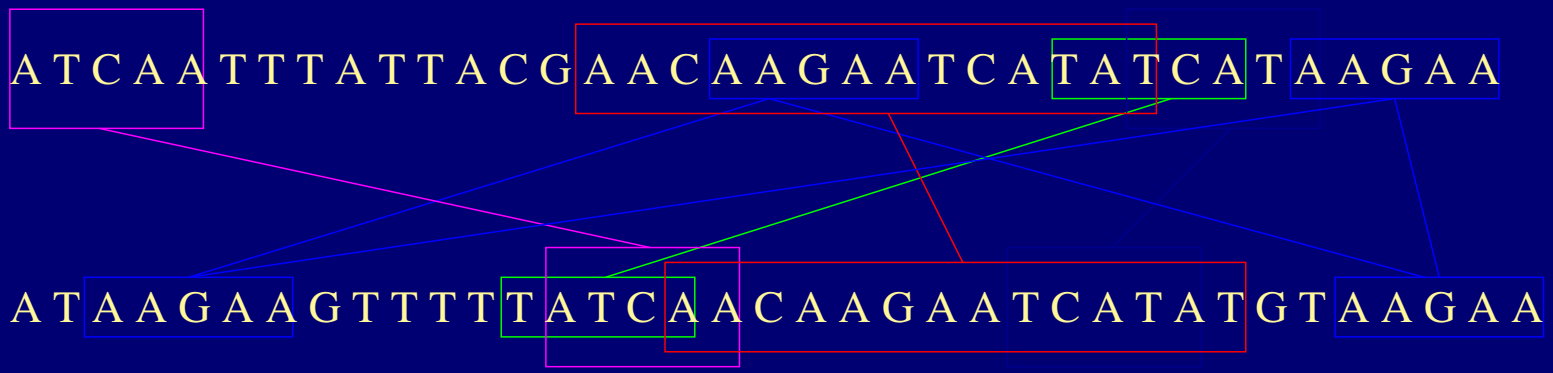
Maximal Exact Macth (MEM) is a substring and exact match in two strings, S_1 and S_2, and cannot simultaneously be extended to the left and right in corresponding strings.
Maximal Unique Match (MUM) is unique MEM in two strings.

Let a MEM denoted by (p_1, p_2, ℓ)
where p_1 (p_2) is the position in S_1 (S_2) and ℓ is the length of the MEM.
So $S_1[p_1 + i] = S_2[p_2 + i]$ for $i \in [0, \ell]$, $S_1[p_1 - 1] \neq S_2[p_2 - 1]$, and $S_1[p_1 + \ell] \neq S_2[p_2 + \ell]$.

Maximal Exact Matches ▸ Example (cont.)

Example

For the following sequences, MEMs of length 5 or more



`ATCAA`: (1, 14, 5) `AACAAGAATCAT`: (15, 19, 14)
`AAGAA`: (18, 3, 5), (18, 35, 5), (32, 3, 5), (32, 35, 5)
`TATCA`: (26, 13, 5) `TCATA`: (28, 12, 5)

Maximal Exact Matches ▸ Example (cont.)

Input

- Two strings S and T of lenth m

Output

- The set of maximal exact matches of S

Solution

- Same as maximal pairs problem.
- leaf node: make a linked list for each pair of genome and left character
- branch node: Cartesian product and union children's linked lists if the length of its path label is above a threshold T_m

Maximal Exact Matches ▸ Example (cont.)

Node 1:

Seq.	Left	Pos list
1	T	2

Node 2:

Seq.	Left	Pos list
2	C	2

Node 3:

Seq.	Left	Pos list
1	T	2
2	C	2

`ACT`: (2, 2, 3)

Node 3:

Seq.	Left	Pos list
1	T	2
2	C	2

Node 6:

Seq.	Left	Pos list
1	T	5
2	C	6

Node 7:

Seq.	Left	Pos list
1	T	2→5
2	C	2→6

`A`: (5, 2, 1), (2, 6, 1)

Summary of Suffix Tree

Suffix tree

Suffix tree has all substrings which are path labels starting from root node.

Internal node

Internal node of suffix tree has at least two leaves, so its path label is common substring of them and occurs the number of leaves.

All internal nodes

ST: k-repeated substring where k is the number of leaves below its subtree.

GST: k-common substring where k is the number of distinct leaf string identifiers below its subtree

Summary of Suffix Tree (cont.)

Internal nodes whose the length is at least k

GST: common substring of length at least k.

Internal nodes which is left diverse

GS: maximal repeat

GST: maximal exact match

Internal nodes which is left diverse and parent of each leaf node

GS: supermaximal repeat

GST: maximal unique match

References

- http://www-igm.univ-mlv.fr/~lecroq/string/.
- http://www.mat.unb.br/~ayala/TCgroup/PatternMatching/kmpbm.html.
- http://www.i.kyushu-u.ac.jp/~takeda/PM_DEMO/e.html.
- P. Weiner.
  Linear pattern matching algorithm.
  In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11, Washington, DC, 1973.
- E. M. McCreight.
  A space-economical suffix tree construction algorithm.
  J. Algorithms, 23(2):262–272, 1976.

References (cont.)

- E. Ukkonen.
  On-line construction of suffix trees.
  Algorithmica, 14(3):249–260, 1995.
- http://homepage.usask.ca/~ctl271/857/suffix_tree.shtml.
- http://pauillac.inria.fr/~quercia/documents-info/Luminy-98/albert/JAVA+html/SuffixTreeGrow.html.
- http://sequence.rutgers.edu/st/.
- http://wwwmayr.in.tum.de/konferenzen/Ferienakademie99/maass/applet.html.

References (cont.)

D. Gusfield.

Algorithms on strings, trees and sequences: Computer science and computational biology.

Cambridge University Press, Cambridge, 1997.