Full-text indexes

Indexing sequence data

- Large sequence data should be stored into data structures (*indexes*) that support efficient and fast query and retrieval
- *Example 1*: to look up for *specific* patterns, we can use an *inverted index*
- *Example 2*: to look up for patterns (strings) of fixed length *k* (*k-mers*) we can use a *hash table*
- What if we need to search for patterns of *arbitrary length*?
- *full-text indexes* allow a search for patterns of *any length* occurring at *any position* of the text

What is a full-text index?

- Text index = a data structure built from a given text (string, sequence) that supports certain type of (typically pattern look-up) queries

- genomic sequences don't have word structure (cf *inverted indexes*) and query size is arbitrary (cf *hash tables*)
- other examples:
  - Chinese, Korean languages
  - agglutinative/inflectional languages (Finnish, German, …): looking for particles
  - MIDI files, audio signals, program code, numerical sequences, …

- **full-text indexes** allow a search for patterns of *any length* occurring at *any position* of the text

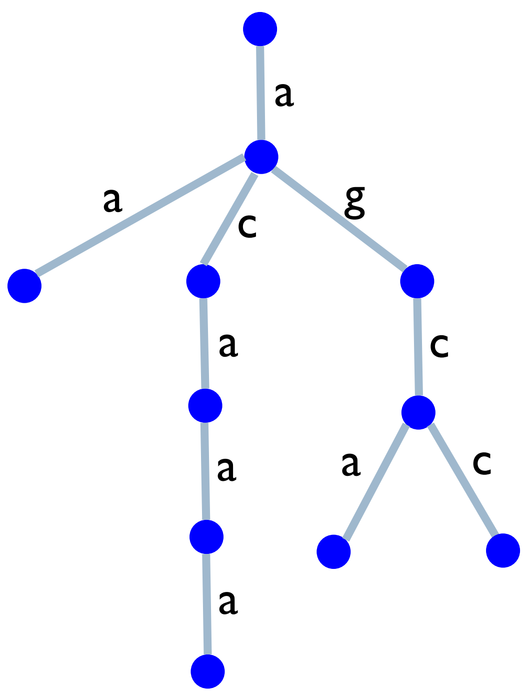
“Ideal” index structure for $T[1..n]$

- Takes space $O(n)$
- Can be constructed in time $O(n)$
- All occurrences of a query pattern P can be reported in time $O(|P| + occ)$

Suffix tree

Trie (aka digital tree)

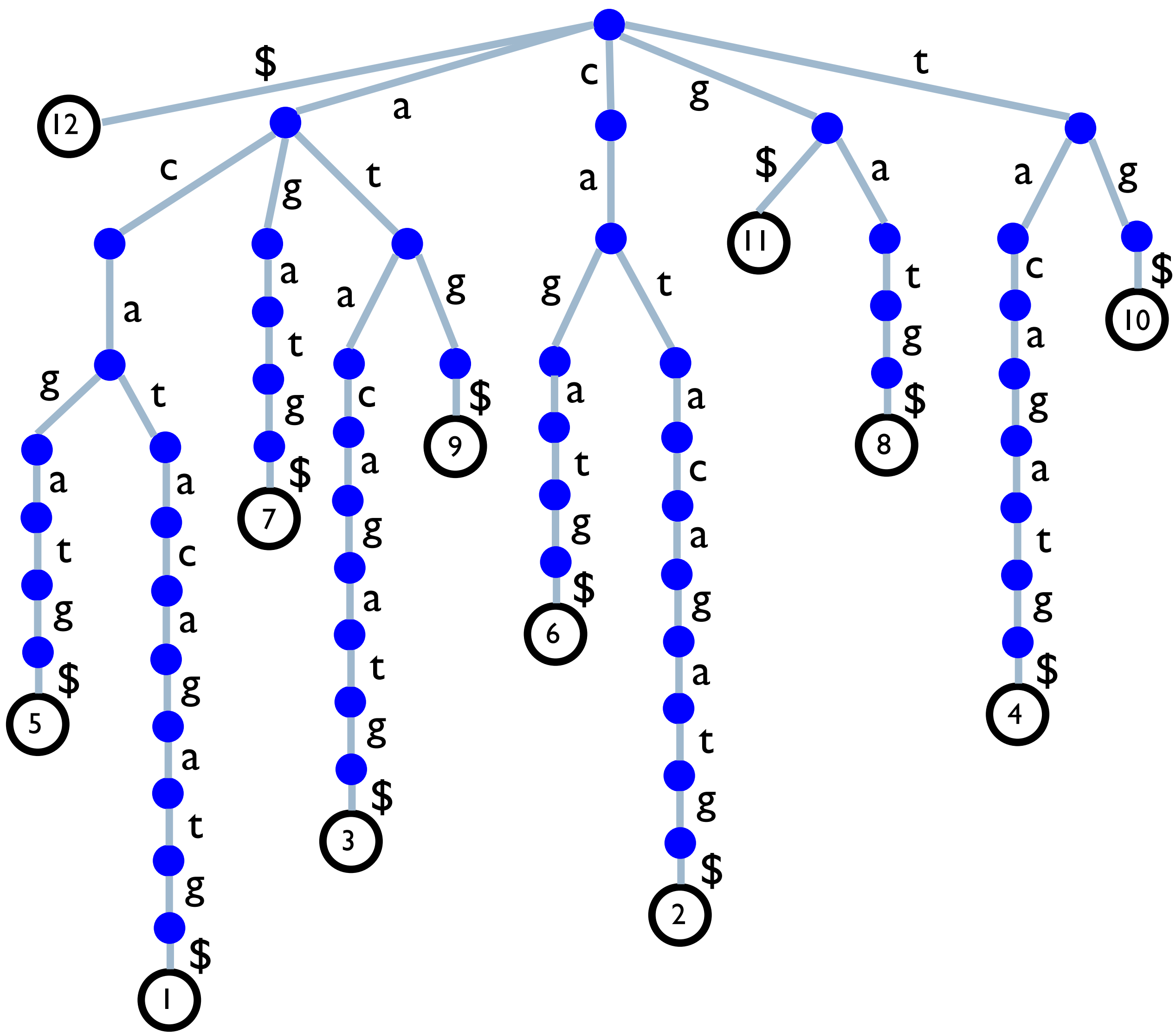
trie for {agca,acaaa,agcc,aa}



- every string of the set is "spelled" starting from root
- edges outgoing from a node are labeled by different characters
- trie can be viewed as an automaton recognizing the given set of strings (or all their prefixes)

Suffix trie

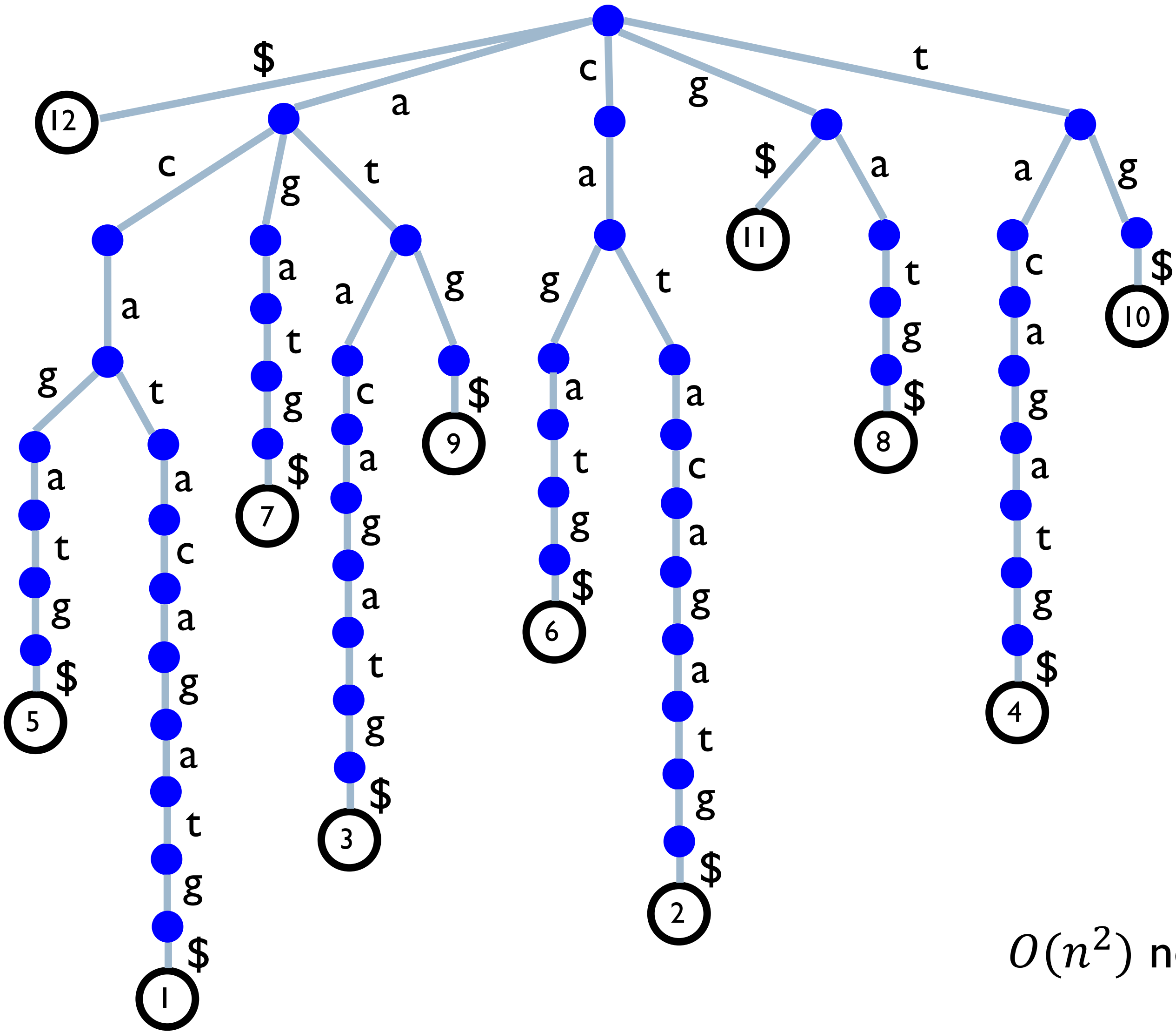
T=acatacagatg$



acatacagatg$	1
catacagatg$	2
atacagatg$	3
tacagatg$	4
acagatg$	5
cagatg$	6
agatg$	7
gatg$	8
atg$	9
tg$	10
g$	11
$	12

Suffix trie

T=acatacagatg$

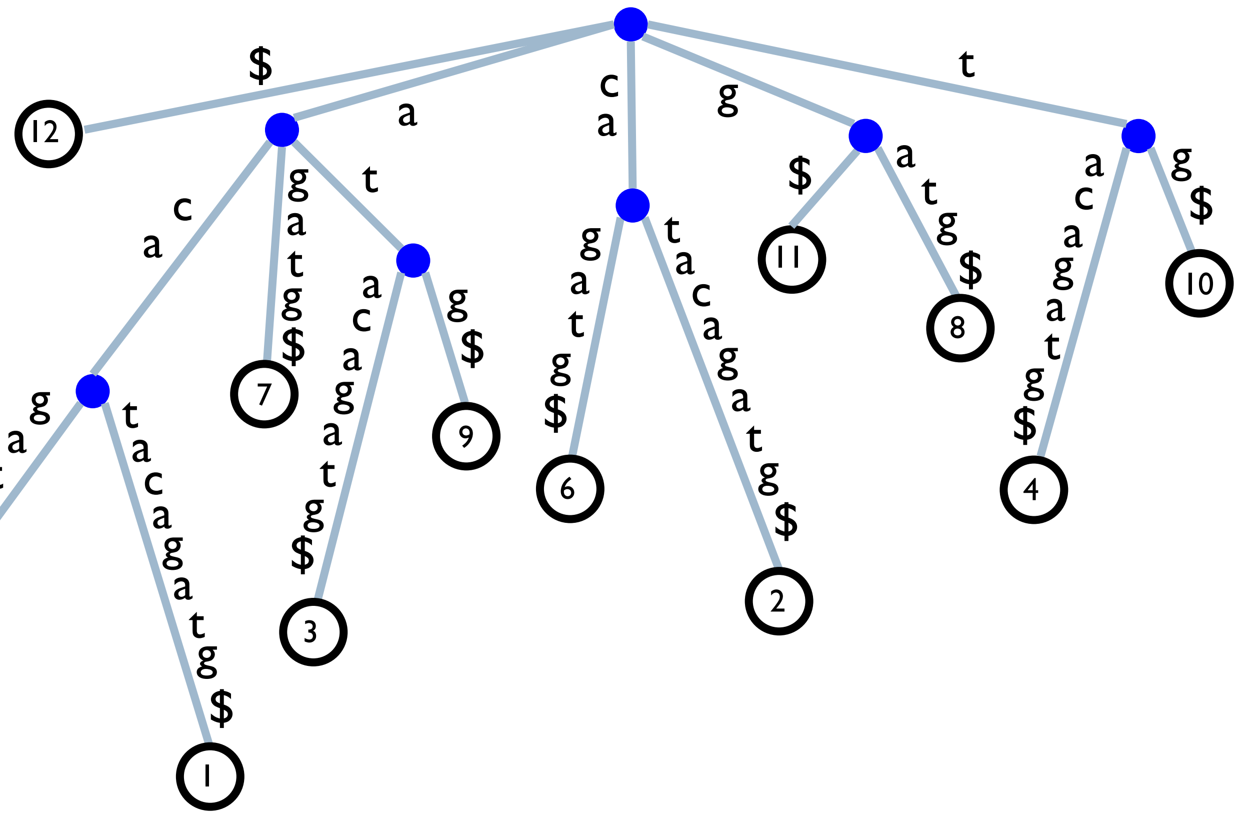


acatacagatg$	1
catacagatg$	2
atacagatg$	3
tacagatg$	4
acagatg$	5
cagatg$	6
agatg$	7
gatg$	8
atg$	9
tg$	10
g$	11
$	12

$O(n^2)$ nodes $a^n b^n$

Suffix tree

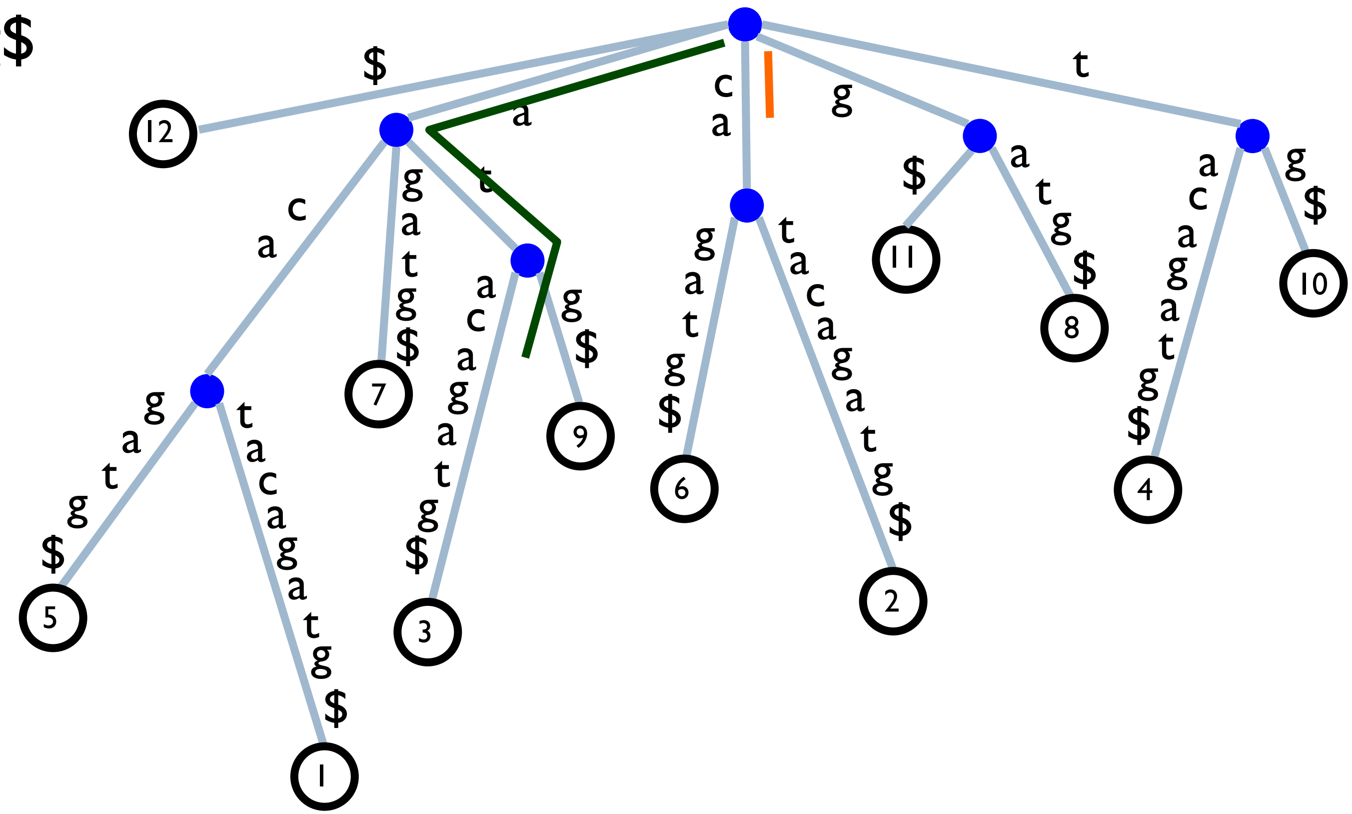
T=acatacagatg$



- explicit vs implicit nodes
- an edge label is start and end positions of corresponding substring (rather than substring itself)
- takes space $O(n)$

Suffix tree: applications

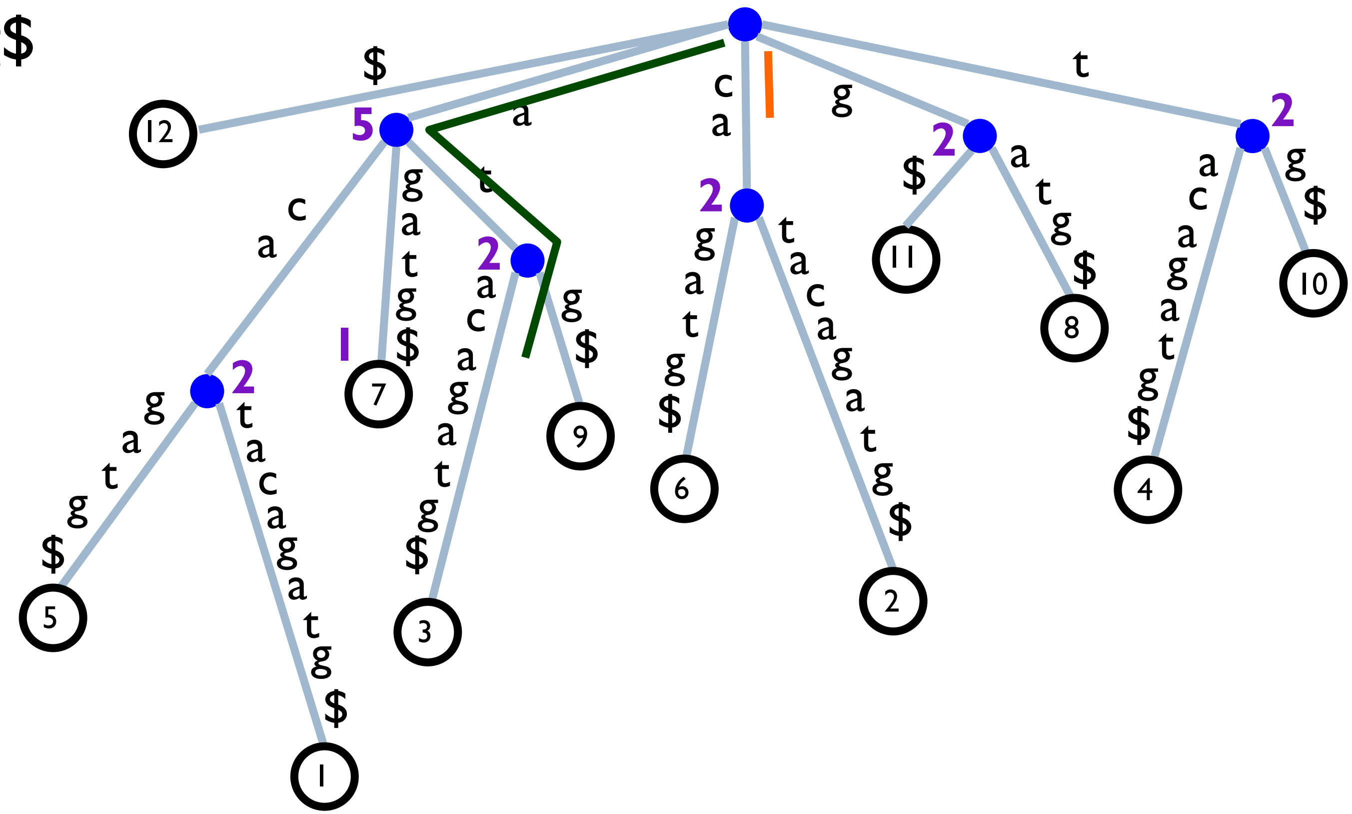
T=acatacagatg$



- check if a pattern P occurs in T in time $O(|P|)$. *Ex*: P_1=atac P_2=c

Suffix tree: applications

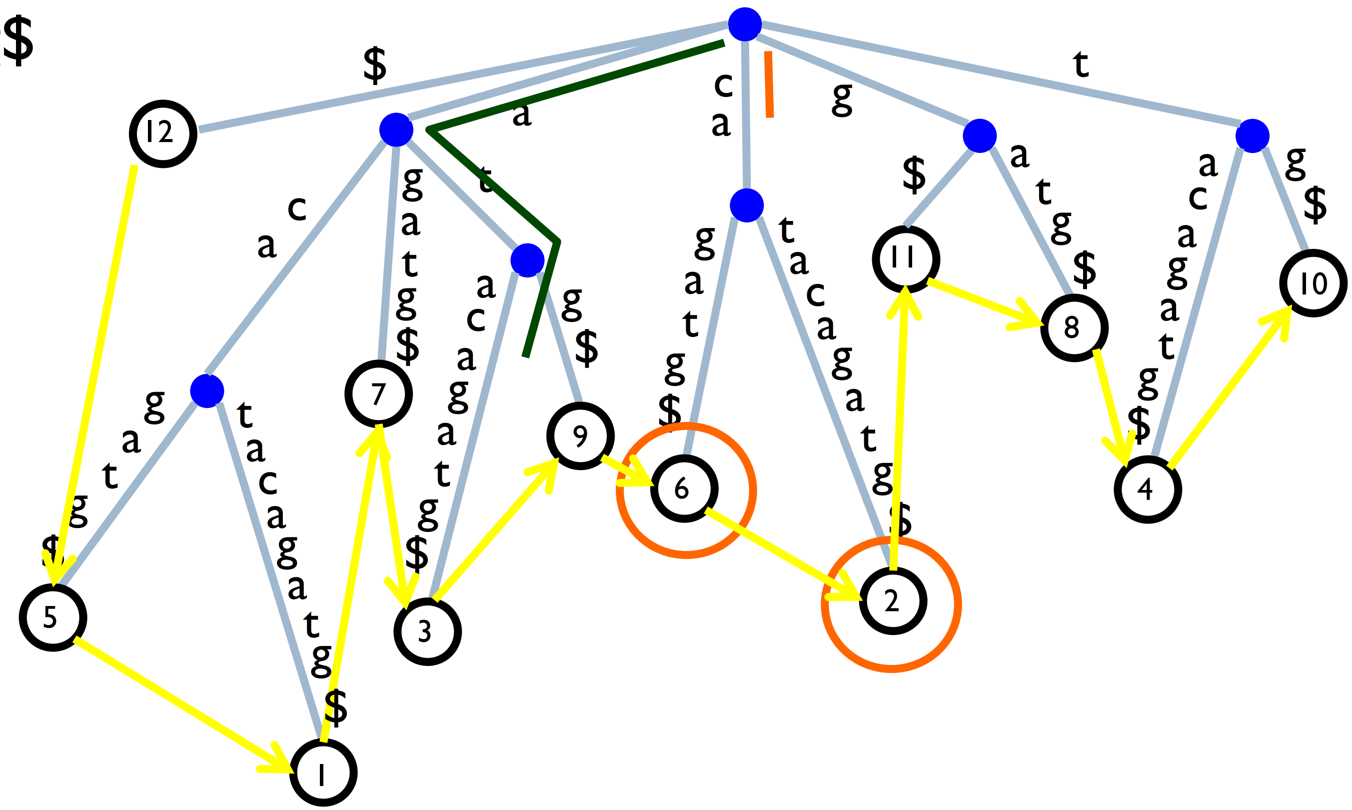
T=acatacagatg$



- check if a pattern P occurs in T in time $O(|P|)$. *Ex*: P_1=atac P_2=c
- report the number of occurrences in $O(|P|) \Rightarrow$ preprocess number of leaves

Suffix tree: applications

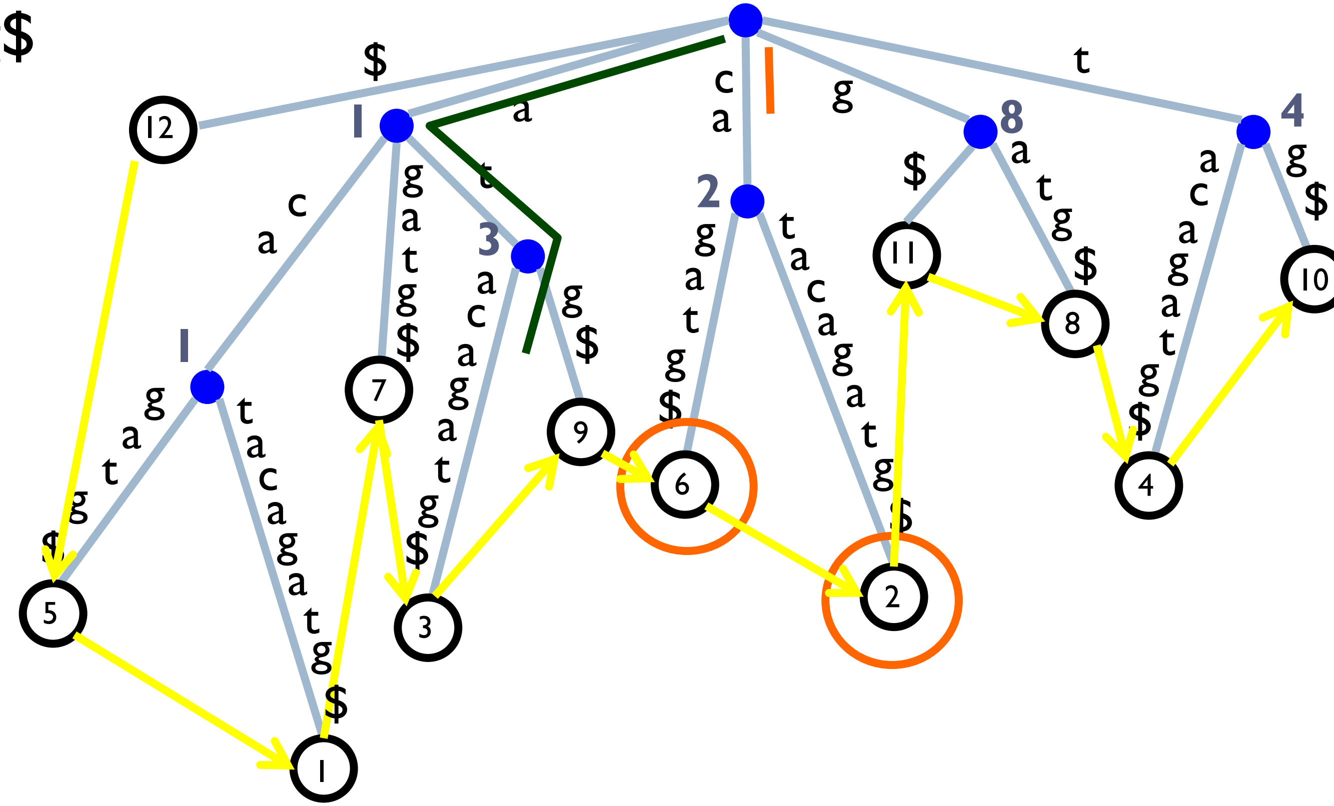
T=acatacagatg$



- check if a pattern P occurs in T in time $O(|P|)$. *Ex*: P_1=atac P_2=c
- report the number of occurrences in $O(|P|) \Rightarrow$ preprocess number of leaves
- report all occurrences in $O(|P| + occ) \Rightarrow$ chain leaves and preprocess leftmost and rightmost leaves

Suffix tree: applications

T=acatacagatg$



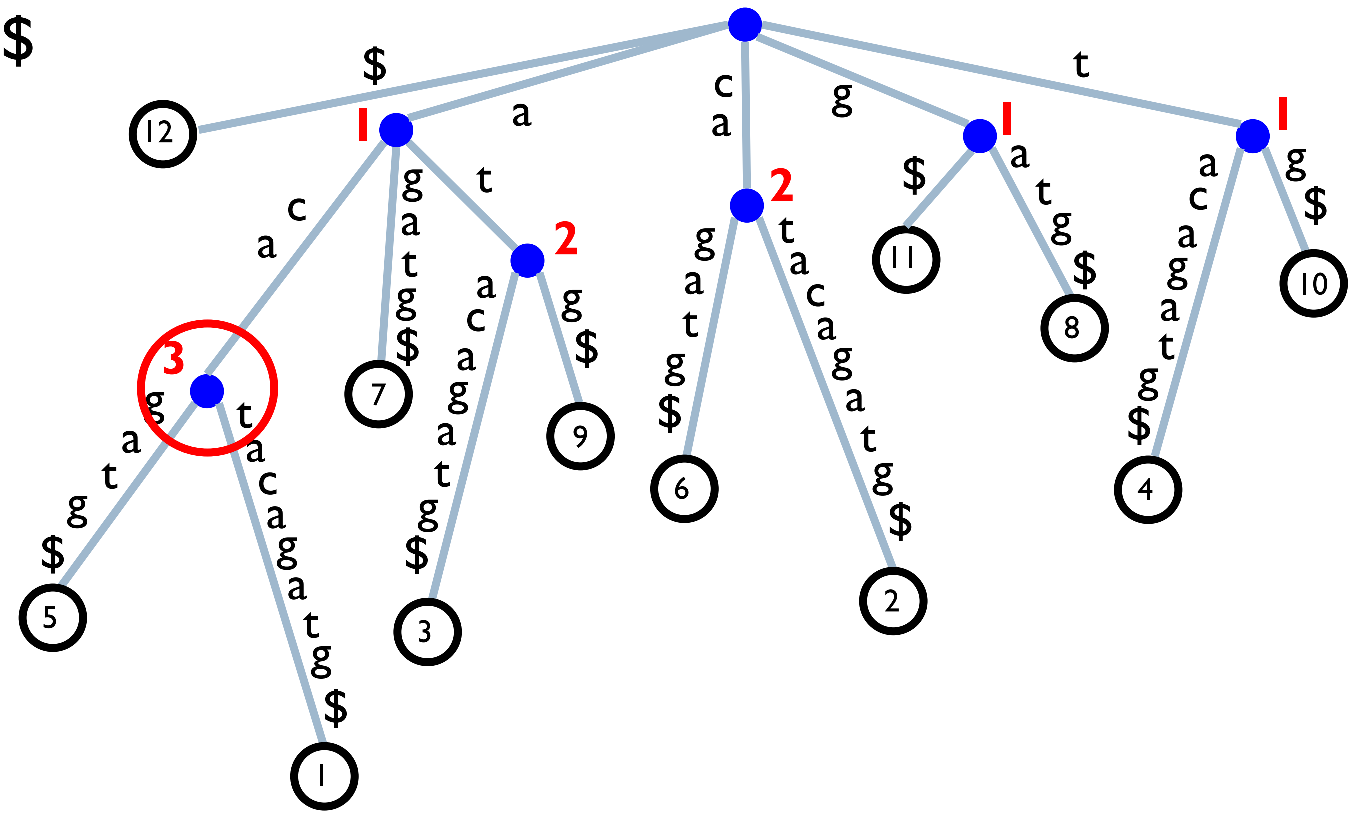
- check if a pattern P occurs in T in time $O(|P|)$. *Ex*: P_1=atac P_2=c
- report the number of occurrences in $O(|P|)\Rightarrow$ preprocess number of leaves
- report all occurrences in $O(|P| + occ)\Rightarrow$ chain leaves and preprocess leftmost and rightmost leaves
- report the first (leftmost) occurrence in $O(|P|)\Rightarrow$ preprocess minimal leaf label

Quiz 9.2

- Construct the suffix tree for string MISSISSIPPI$
- How many internal nodes does it have?

Suffix tree: applications

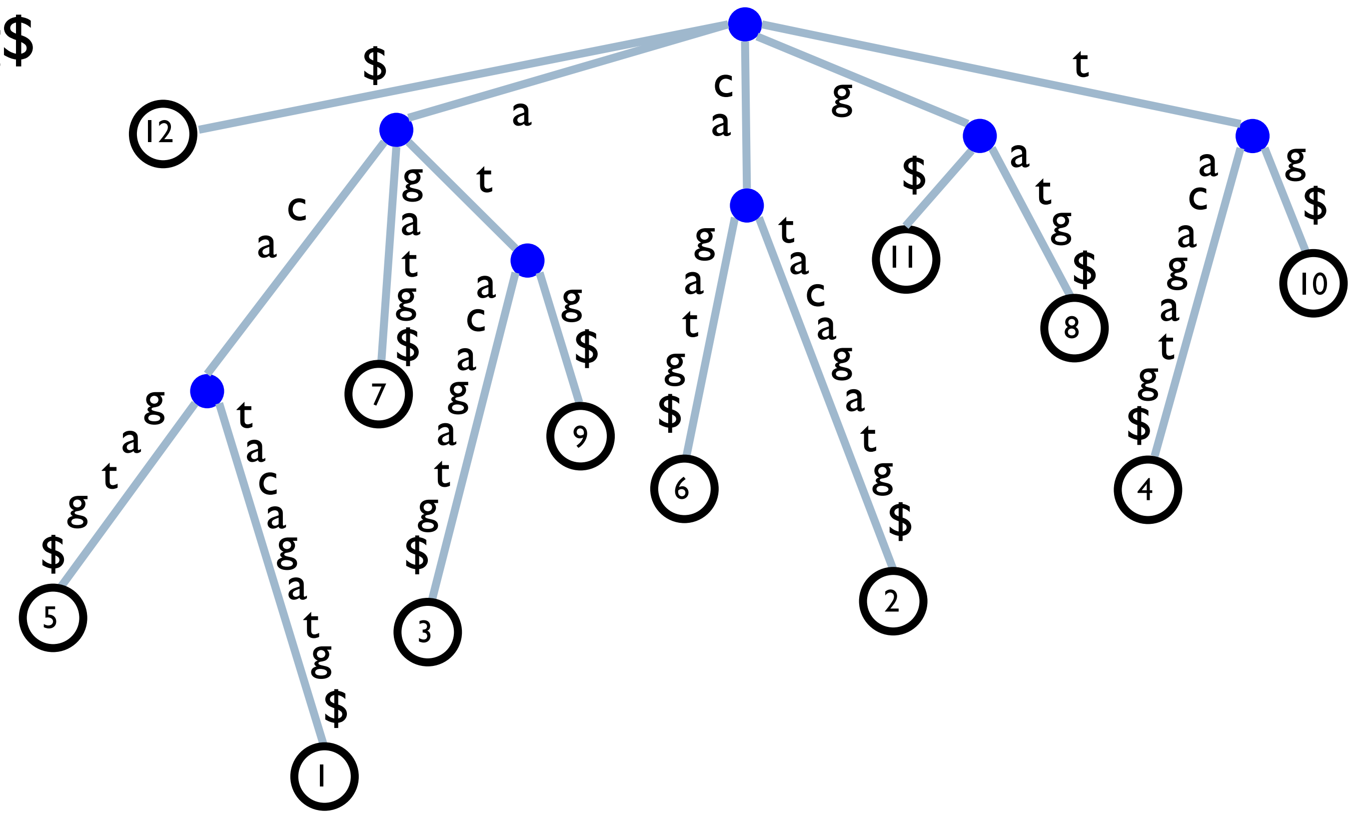
T=acatacagatg$



- longest repeated substring ⇒ deepest (**w.r.t. string depth**) internal node in the suffix tree (aca)

Suffix tree: applications

T=acatacagatg$



- longest extension queries: given *two positions i,j, output the length of the longest common substring starting at i,j*
- reduces to *lowest common ancestor (lca)* queries
- *lca* queries can be answered in $O(1)$ time after linear-time preprocessing of the tree [Harel, Tarjan 84], [Bender, Farach-Colton 00]

Suffix tree: some history

- Suffix tree can be constructed in time O(n)
  - Weiner 1973: right-to-left construction

LINEAR PATTERN MATCHING ALGORITHMS

Peter Weiner

The Rand Corporation, Santa Monica, California*

Abstract

In 1970, Knuth, Pratt, and Morris [1] showed how to do basic pattern matching in linear time. Related problems, such as those discussed in [4], have previously been solved by efficient but sub-optimal algorithms. In this paper, we introduce an interesting data structure called a bi-tree. A linear time algorithm for obtaining a compacted version of a bi-tree associated with a given string is presented. With this construction as the basic tool, we indicate how to solve several pattern matching problems, including some from [4], in linear time.

"Algorithm of the year 1973" - Donald Knuth

I. Introduction

In 1970, Knuth, Morris, and Pratt [1-2] showed how to match a given pattern into another given string in time proportional to the sum of the lengths of the pattern and string. Their algorithm was derived from a result of Cook [3] that the 2-way deterministic pushdown languages are recognizable on a random access machine in time O(n). Since 1970, attention has been given to several related problems in pattern matching [4-6], but the algorithms developed in these investigations usually run in time which is slightly worse than linear, for example O(n log n). It is of considerable interest to either establish that there exists a non-linear lower bound on the run time of all algorithms which solve a given pattern matching problem, or to exhibit an algorithm whose run time is of O(n).

In the following sections, we introduce an interesting data structure, called a bi-tree, and show how an efficient calculation of a bi-tree can be applied to

... giving a formal definition of a bi-tree, we re... definitions and terminology concerning t-ary trees ... Knuth [7] for further details.)

A ... T over $\Sigma = \{\sigma_1, \ldots, \sigma_t\}$ is a set of nodes N which is ... empty or consists of a *root*, $n_0 \in N$, and t ordered, ... joint t-ary trees.

Clearly, every node ... $\in N$ is the root of some t-ary tree T^i which itself consists of n_i and t ordered, disjoint t-ary trees ... $T^i_1, \ldots, T^i_t$. We call the tree T^i_j a *sub-tree* of ... all sub-trees of T^i_j are considered to be sub-trees of ... It is natural to associate with a tree T a *successor function*

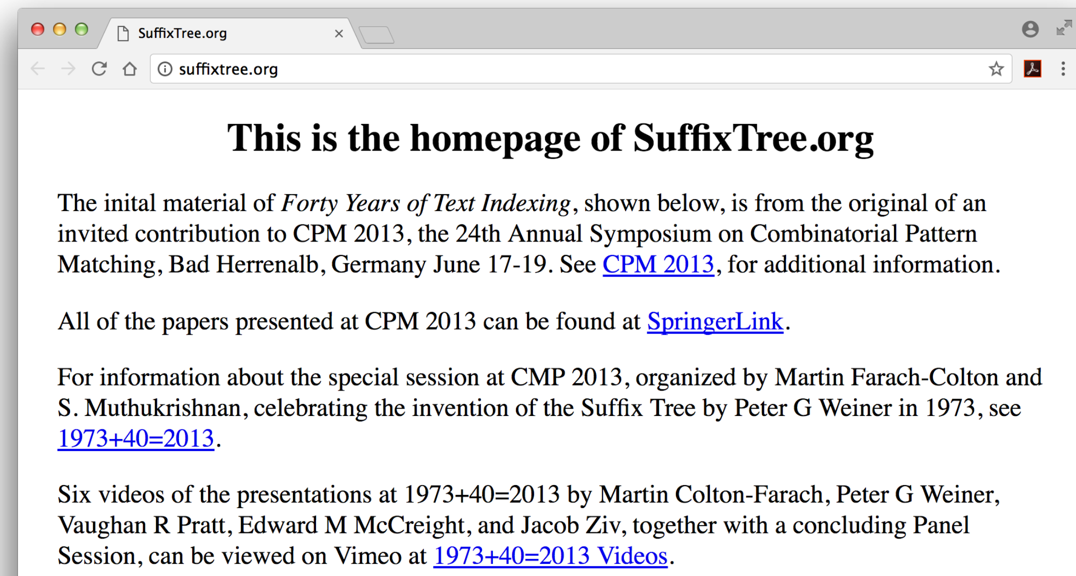
$$S: N \times \Sigma \to (N - \{n_0\}) \cup \{NIL\}$$

defined for all $n_i \in N$ and $\sigma_j \in \Sigma$ by

n^i_j the root of T^i_j if T^i_j is non-empty

Suffix tree: some history

- Suffix tree can be constructed in time O(n)
  - Weiner 1973: right-to-left construction
  - McCreight 1976: left-to-right
  - Ukkonen 1995: left-to-right and *online*
  - Farach 1997: for integer alphabets



This is the homepage of SuffixTree.org

The inital material of *Forty Years of Text Indexing*, shown below, is from the original of an invited contribution to CPM 2013, the 24th Annual Symposium on Combinatorial Pattern Matching, Bad Herrenalb, Germany June 17-19. See CPM 2013, for additional information.

All of the papers presented at CPM 2013 can be found at SpringerLink.

For information about the special session at CMP 2013, organized by Martin Farach-Colton and S. Muthukrishnan, celebrating the invention of the Suffix Tree by Peter G Weiner in 1973, see 1973+40=2013.

Six videos of the presentations at 1973+40=2013 by Martin Colton-Farach, Peter G Weiner, Vaughan R Pratt, Edward M McCreight, and Jacob Ziv, together with a concluding Panel Session, can be viewed on Vimeo at 1973+40=2013 Videos.

DOI:10.1145/2810036

Tracing the first four decades in the life of suffix trees, their many incarnations, and their applications.

BY ALBERTO APOSTOLICO, MAXIME CROCHEMORE, MARTIN FARACH-COLTON, ZVI GALIL, AND S. MUTHUKRISHNAN

40 Years of Suffix Trees

WHEN WILLIAM LEGRAND finally decrypted the string, it did not seem to make much more sense than it did before.

53‡‡‡305))6*,48264‡.)4z);806",48†8P60))85;1‡
(;:‡*8†83(88)5*†,46(;88*96*?;8)*‡(;485);5*†2:*‡
(;4956*2(5*Ñ4)8P8*;4069285);)6‡8)4‡‡;1(‡9;48081;8:
8‡1;4885;4)485†528806*81(ddag9;48;(88;4(‡?34;
48)4‡;161;:188;‡?;

The decoded message read: "A good glass in the bishop's hostel in the devil's seat forty-one degrees and thirteen minutes northeast and by north main branch seventh limb east side shoot from the left eye of the death's-head a bee line from the tree through the shot fifty feet out." But at least it did sound more like natural language, and eventually guided the main character of Edgar Allan Poe's "The Gold-Bug"[36] to discover the treasure he had been after. Legrand solved a substitution cipher using symbol frequencies. He first looked for the most frequent symbol and changed it into the most frequent letter of English, then similarly inferred the most frequent word, then punctuation marks, and so on.

Both before and after 1843, the natural impulse when faced with some mysterious message has been to count frequencies of individual tokens or subassemblies in search of a clue. Perhaps one of the most intense and fascinating subjects for this kind of scrutiny have been biosequences. As soon as some such sequences became available, statistical analysts tried to link characters or blocks of characters to relevant biological functions. With the early examples of whole genomes emerging in the mid-1990s, it seemed natural to count the occurrences of all blocks of size 1, 2, and so on, up to any desired length, looking for statistical characterizations of coding regions, promoter regions, among others.

This article is not about cryptography. It is about a data structure and its variants, and the many surprising and useful features it carries. Among these is the fact that, to set up a statistical table of occurrences for all substrings (also called *factors*), of any length, of a text string of n characters, it only takes time and space linear in the length of the text string. While nobody would be so foolish as to solve the problem by first generating all exponentially many possible strings and then counting their occurrences one by one, a text string may still contain $\Theta(n^2)$ distinct substrings, so that tabulating all of them in linear space, never mind linear time, already seems puzzling.

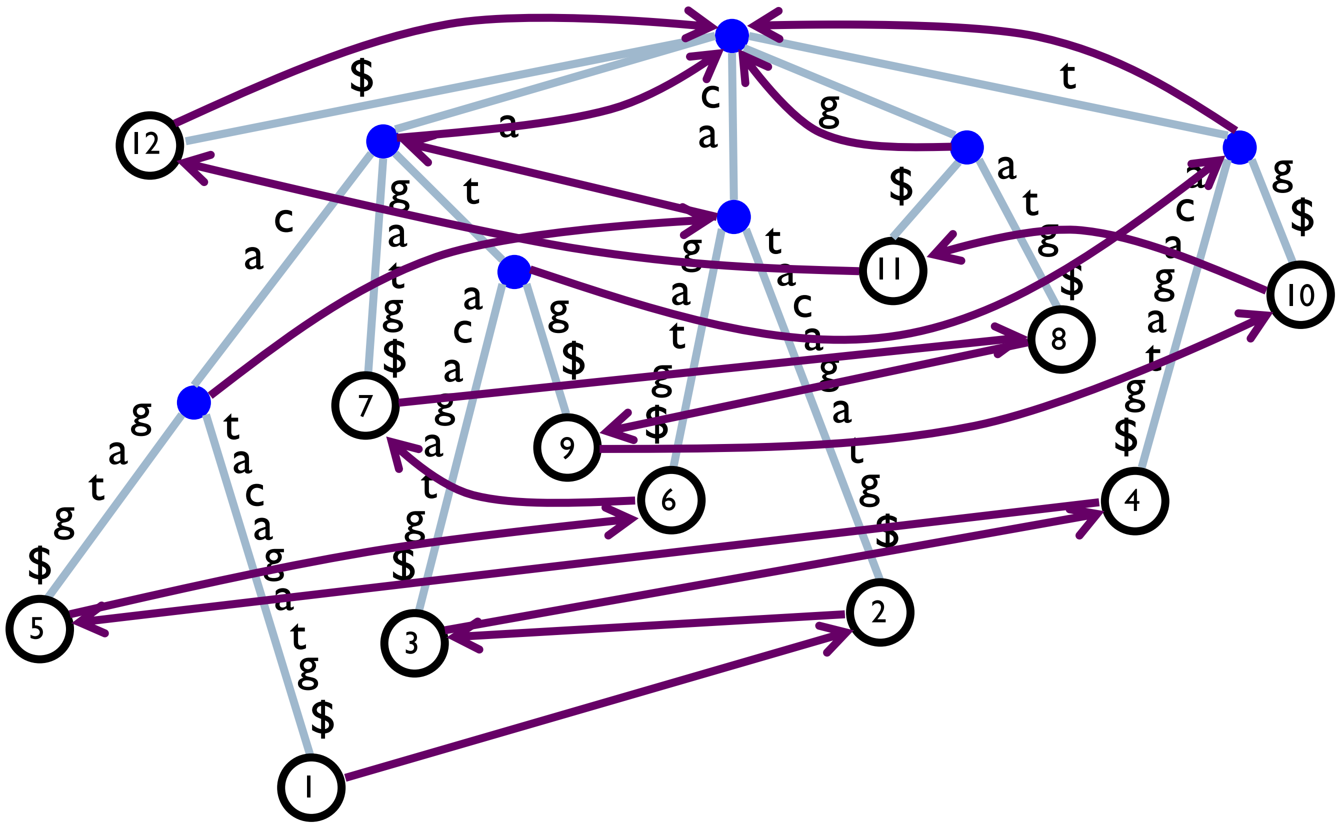
We dedicate this article to our friend and colleague, Alberto Apostolico (1948–2015), who passed away on July 20. He was a major figure in the development of algorithms on strings.

IMAGE BY PAPUCHALKA

Suffix tree augmented with suffix links

McCreight and Ukkonen use *suffix links*

suf-link($\overline{au}$)=$\overline{u}$ for explicit nodes $\overline{au}$



Suffix tree augmented with suffix links

McCreight and Ukkonen use *suffix links*

suf-link($\overline{au}$)=$\overline{u}$ for explicit nodes $\overline{au}$