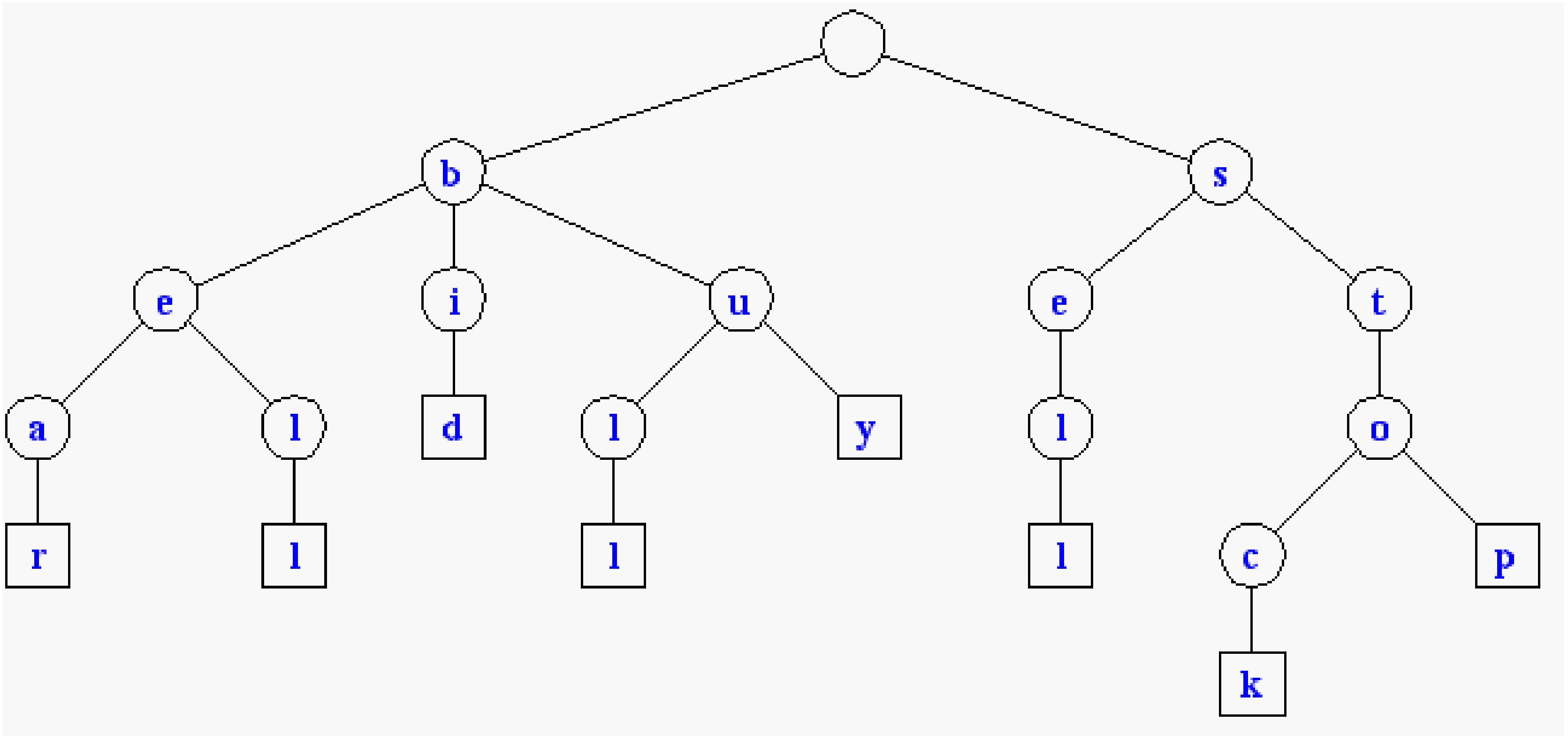
Tries

- Standard Tries
- Compressed Tries
- Suffix Tries

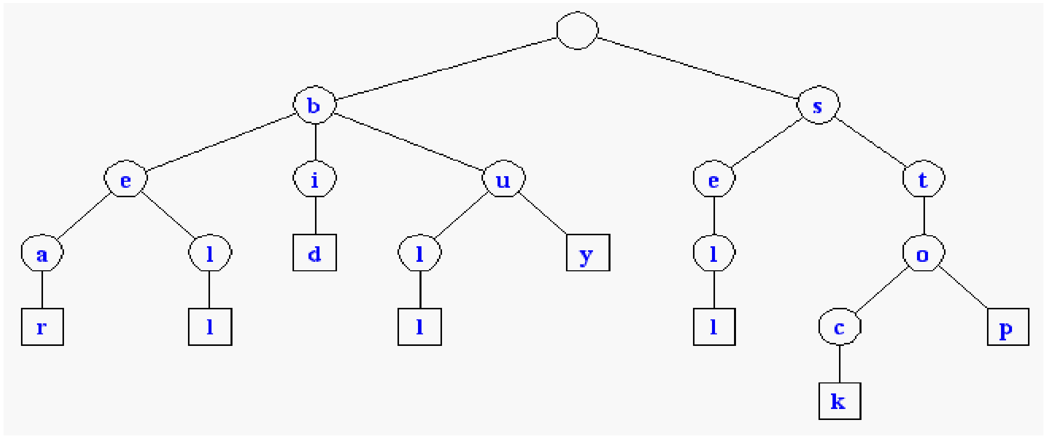


Text Processing

- We have seen that preprocessing the pattern speeds up pattern matching queries
- After preprocessing the pattern in time proportional to the pattern length, the KMP algorithm searches an arbitrary English text in time proportional to the text length
- If the text is large, immutable and searched for often (e.g., works by Shakespeare), we may want to preprocess the text instead of the pattern in order to perform pattern matching queries in time proportional to the pattern length.

Standard Tries

- The standard trie for a set of strings S is an ordered tree such that:
  - each node but the root is labeled with a character
  - the children of a node are alphabetically ordered
  - the paths from the external nodes to the root yield the strings of S
- Eg. S = { bear, bell, bid, bull, buy, sell, stock, stop }



Running time for operations

- A standard trie uses O(W) space.
- Operations (find, insert, remove) take time O(dm) each, where:
  - W = total size of the strings in S,
  - m = size of the string involved in the operation
  - d = alphabet size,

Applications of Tries

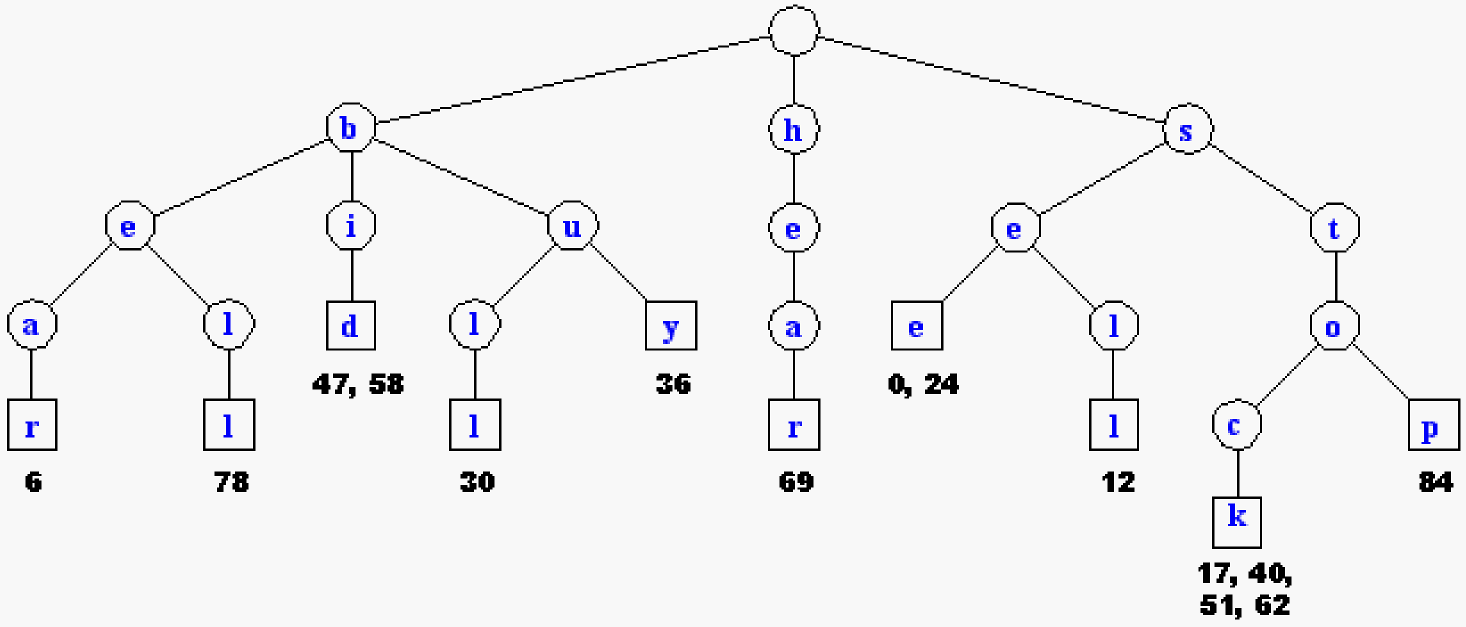
- A standard trie supports the following operations on a preprocessed text in time O(m), where m = |X|
  - word matching: find the first occurence of word X in the text
  - prefix matching: find the first occurrence of the longest prefix of word X in the text
- Each operation is performed by tracing a path in the trie starting at the root

s	e	e		a		b	e	a	r	?		s	e	l	l		s	t	o	c	k	!	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

s	e	e		a		b	u	l	l	?		b	u	y		s	t	o	c	k	!	
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46

b	i	d		s	t	o	c	k	!		b	i	d		s	t	o	c	k	!	
47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68

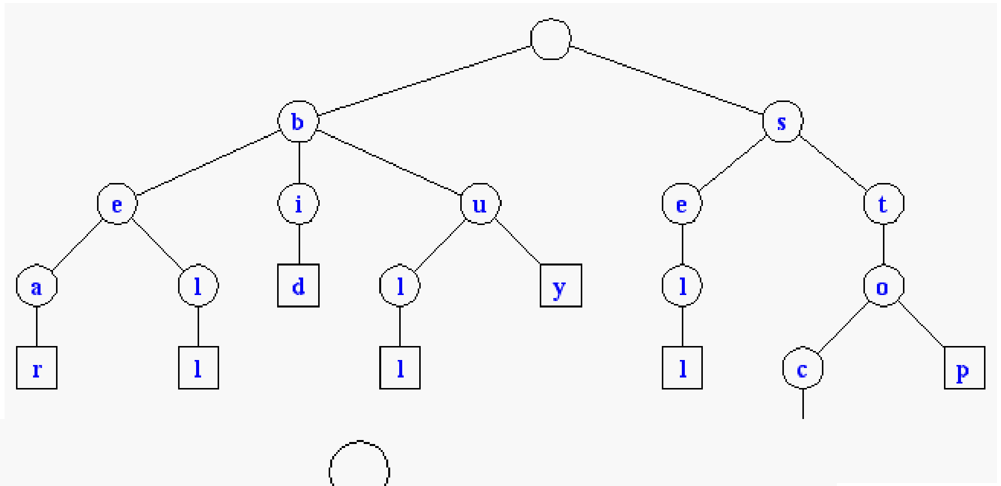
h	e	a	r		t	h	e		b	e	l	l	?		s	t	o	p	!
69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88



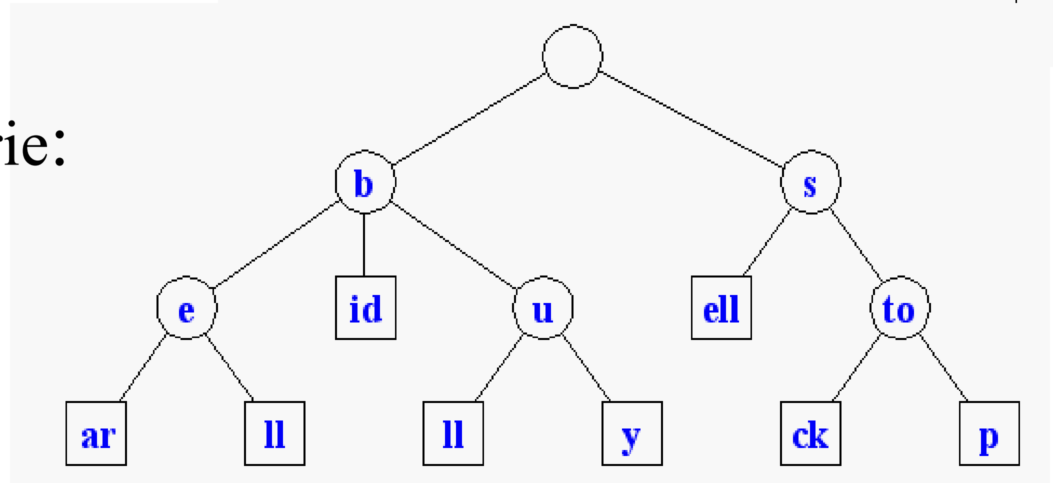
Compressed Tries

- Trie with nodes of degree at least 2
- Obtained from standard trie by compressing chains of redundant nodes.

Standard Trie:



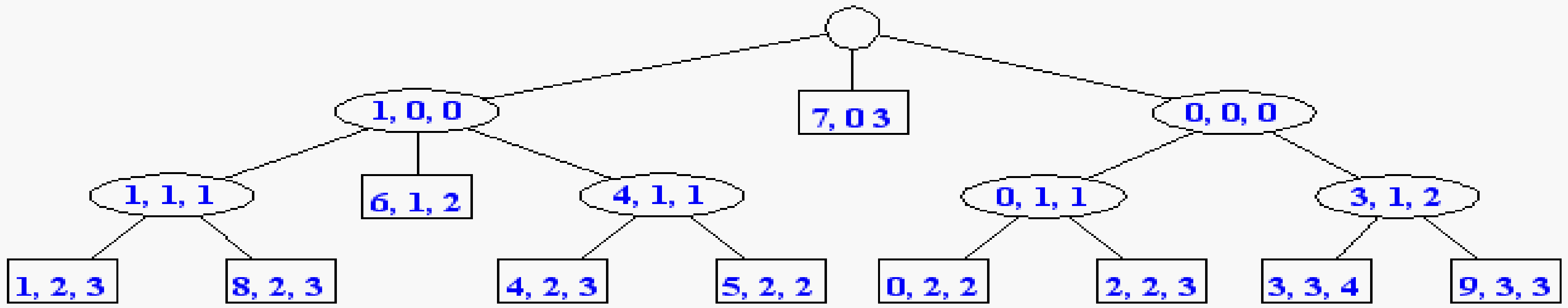
Compressed Trie:

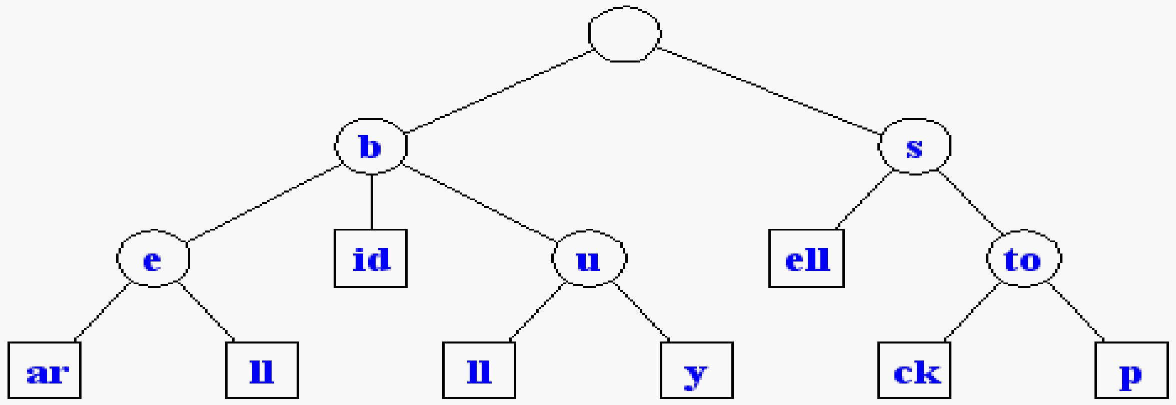


Why Compressed Tries ?

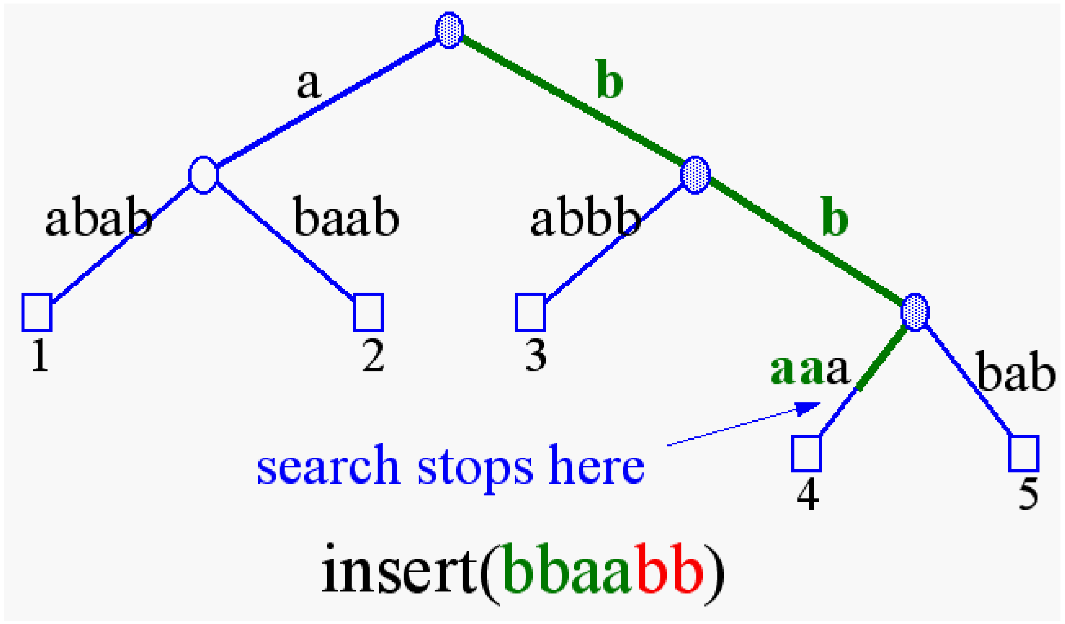
- A tree in which every node has at least 2 children has at most L-1 internal nodes, where L is the number of leaves.
- The number of nodes in a compressed trie is O(s), where s = |S|.
- The label in each node can be stored by using O(1) space index ranges at the nodes

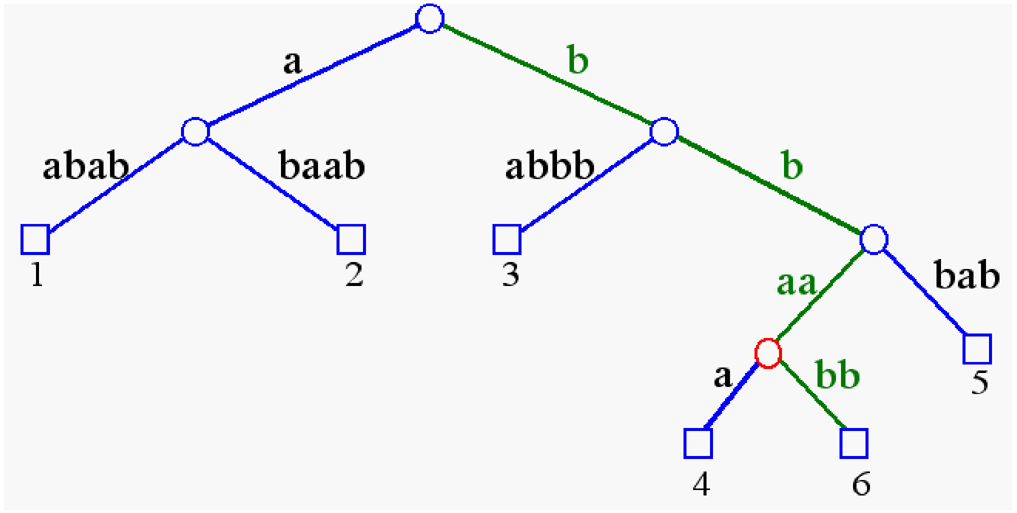
	0	1	2	3	4
S[0] =	s	e	e		
S[1] =	b	e	a	r	
S[2] =	s	e	l	l	
S[3] =	s	t	o	c	k
S[4] =	b	u	l	l	
S[5] =	b	u	y		
S[6] =	b	i	d		
S[7] =	h	e	a	r	
S[8] =	b	e	l	l	
S[9] =	s	t	o	p	





Insertion/Deletion in Compressed Tries





Tries and Web Search Engines

- The index of a search engine (collection of all searchable words) is stored into a compressed trie
- Each leaf of the trie is associated with a word and has a list of pages (URLs) containing that word, called occurrence list
- The trie is kept in internal memory
- The occurrence lists are kept in external memory and are ranked by relevance
- Boolean queries for sets of words (e.g., Java and coffee) correspond to set operations (e.g., intersection) on the occurrence lists
- Additional information retrieval techniques are used, such as
  - stopword elimination (e.g., ignore “the” “a” “is”)
  - stemming (e.g., identify “add” “adding” “added”)
  - link analysis (recognize authoritative pages)

Tries and Internet Routers

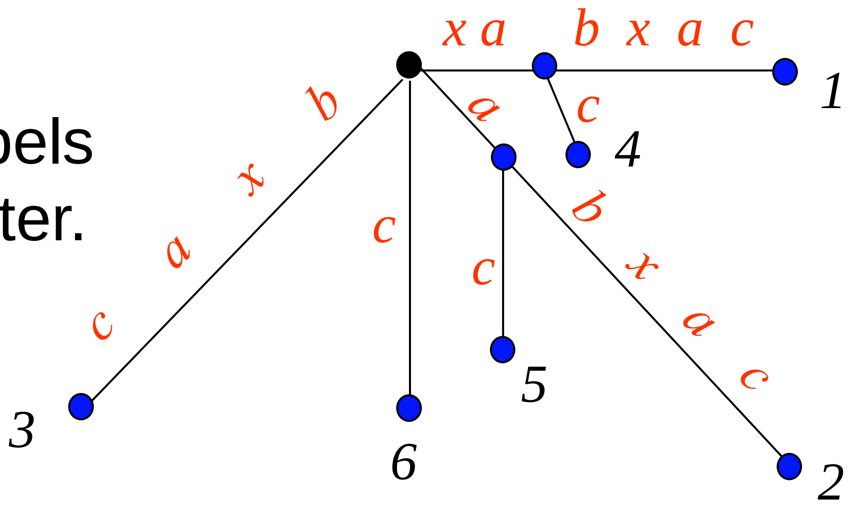
- Computers on the internet (hosts) are identified by a unique 32-bit IP (internet protocol) address, usually written in “dotted-quad-decimal” notation
- E.g., www.google.com is 64.233.189.104
- Use nslookup on Unix to find out IP addresses
- An organization uses a subset of IP addresses with the same prefix, e.g., IITD uses 10.*.*.*
- Data is sent to a host by fragmenting it into packets. Each packet carries the IP address of its destination.
- A router forwards packets to its neighbors using IP prefix matching rules.
- Routers use tries on the alphabet 0,1 to do prefix matching.

Back to Pattern Matching

- Instead of preprocessing the pattern *P*, preprocess the text *T* !
- Use a tree structure where all suffixes of the text are represented;
- Search for the pattern by looking for substrings of the text;
- You can easily test whether *P* is a substring of *T* because any substring of *T* is the prefix of some suffix of *T* .

Suffix Tree

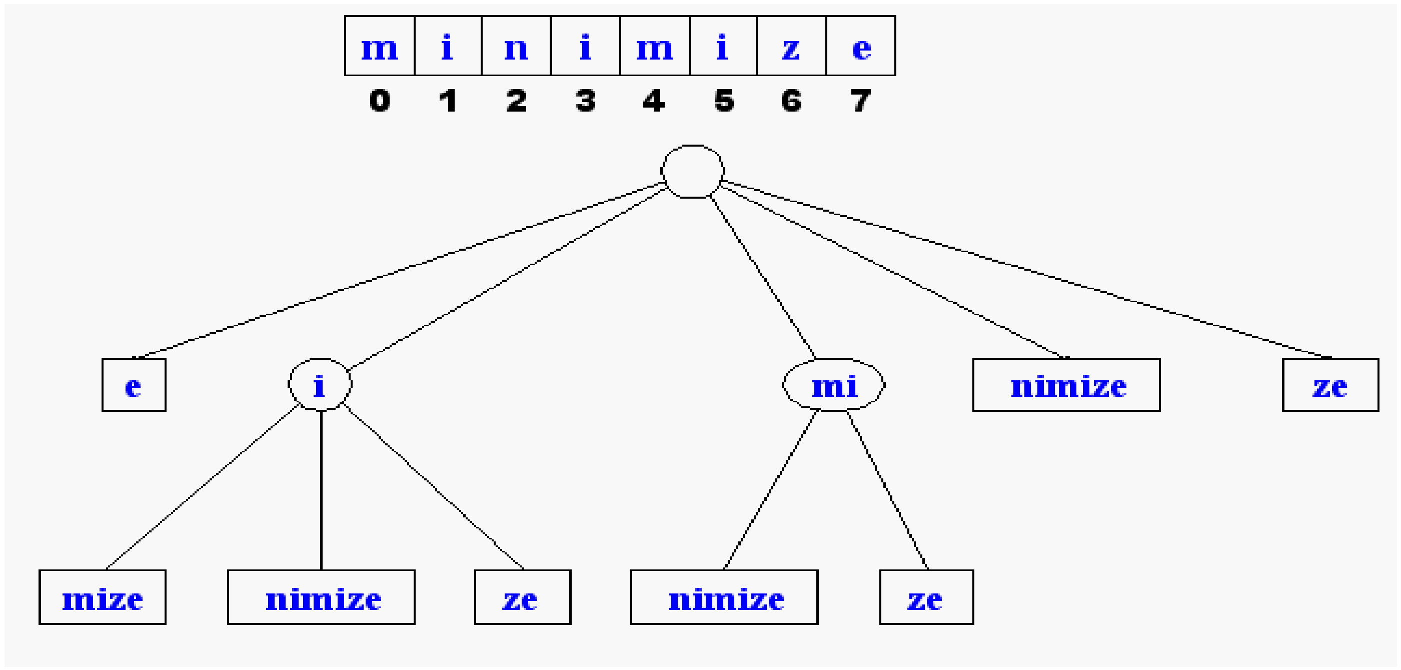
- A suffix tree *T* for string *S* is a rooted directed tree whose edges are labeled with nonempty substrings of *S*.
- Each leaf corresponds to a suffix of *S* in the sense that the concatenation of the edge-labels on the unique path from the root to the leaf spells out this suffix.
- Each internal node, other than the root, has at least two children.
- No two out-edges of a node can have edge-labels with the same first character.



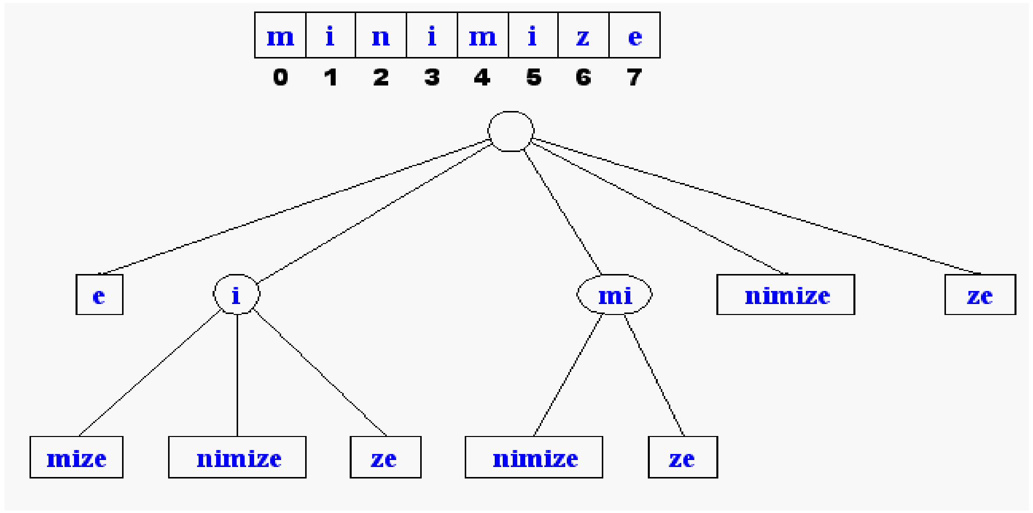
Suffix tree for string xabxac.

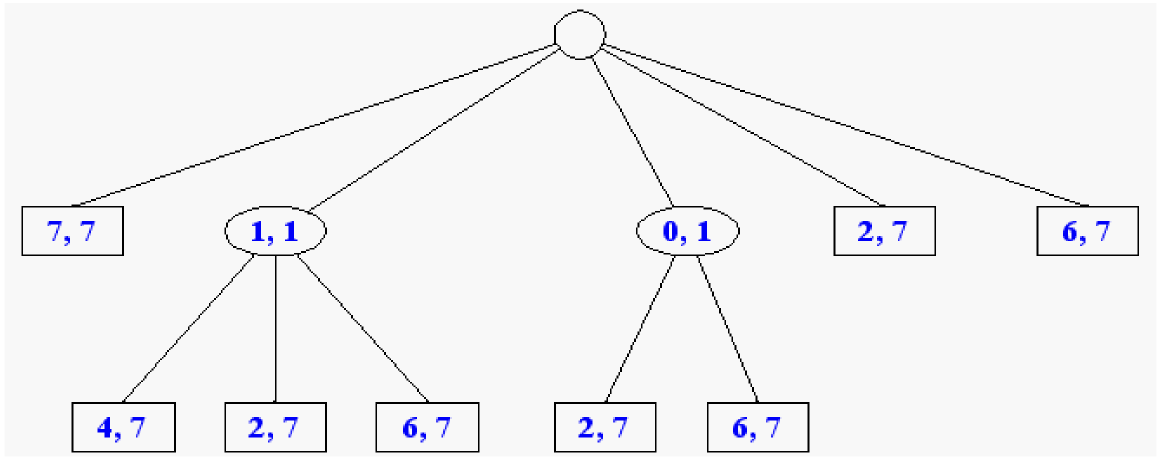
Suffix Tree (2)

- A suffix tree is essentially a compressed trie for all the suffixes of a text
- The suffix tree for a text X of size n from an alphabet of size d stores all the n suffixes of X in O(n) space.



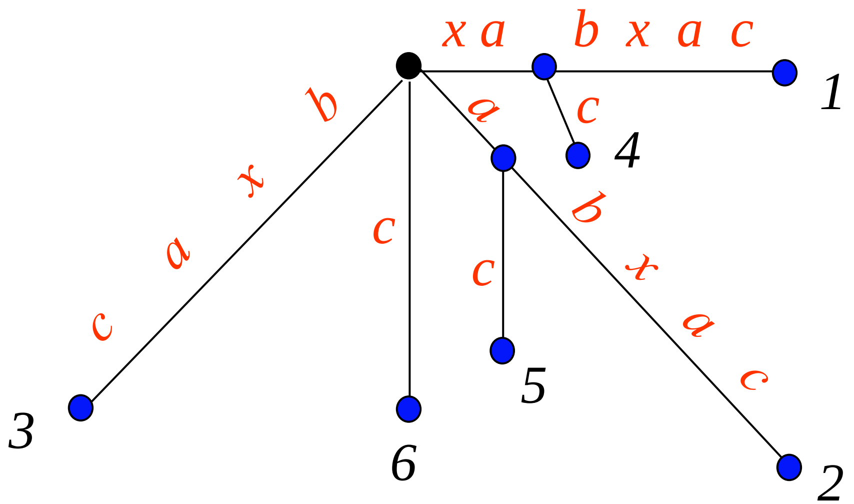
Compact Representation





Note on Suffix Tree

If two suffixes have a same prefix, then their corresponding paths are the same at their beginning, and the concatenation of the edge labels of the mutual part is the prefix.



Suffix tree for string xabxac.

For example, suffix *xabxac* and suffix *xac* have the same prefix, *xa*.

Note on Suffix Tree

- Not all strings guaranteed to have corresponding suffix trees
- For example:

  consider *xabxa:* it does not have a suffix tree, because here suffix *xa* is also a prefix of another suffix, *xabxa*.

  (The path spelling out *xa* would not end at a leaf.)
- How to fix the problem: add $ - a special “termination” character to the alphabet.

Building a Suffix Tree

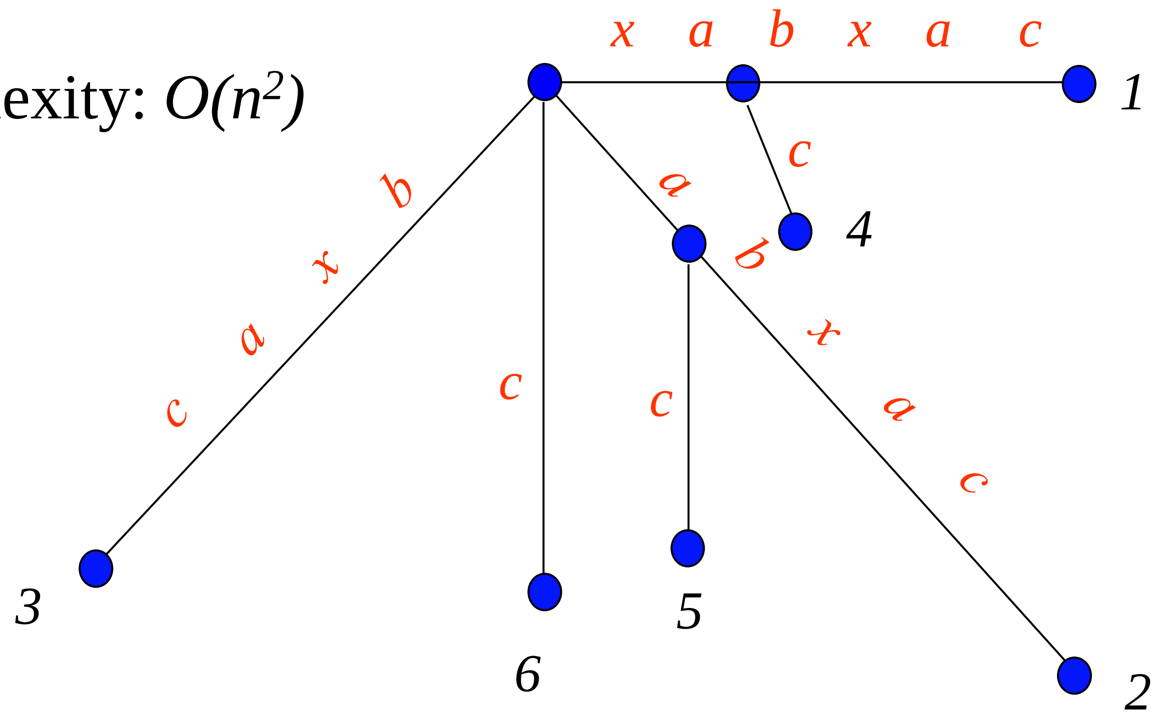
The method starts from a single edge for suffix *S[1..n]*, then it successively enters the edges for suffix *S[i..n]* into the growing tree, for *i* increasing from *2* to *n*.

The general step of the algorithm

- Enter the edge for suffix *S[i..n]* into the tree as follows.
- Starting at the root find the longest path from the root whose label matches a prefix of *S[i..n].* At some point, no further matches are possible.
    - If this point is at a node, then denote this node by *w.*
    - If it is in the middle of an edge, then insert a new node, called *w*, at this point.
- Create a new edge running from *w* to a new leaf labeled *S[i..n].*

Example

Time complexity: $O(n^2)$



Building suffix tree for string xabxac.

Suffix Trees in Pattern Matching

- Given a string ***P*** (pattern) and a longer string ***T*** (text).Our aim is to find all occurrences of pattern ***P*** in text ***T***.
- The idea of the algorithm is that every occurrence of *P* in *T* is a prefix of a suffix of *T*,
- Thus, an occurrence of *P* in *T* can be obtained as the of the labels of edges along the path beginning concatenation at the root.

Suffix Trees in Pattern Matching

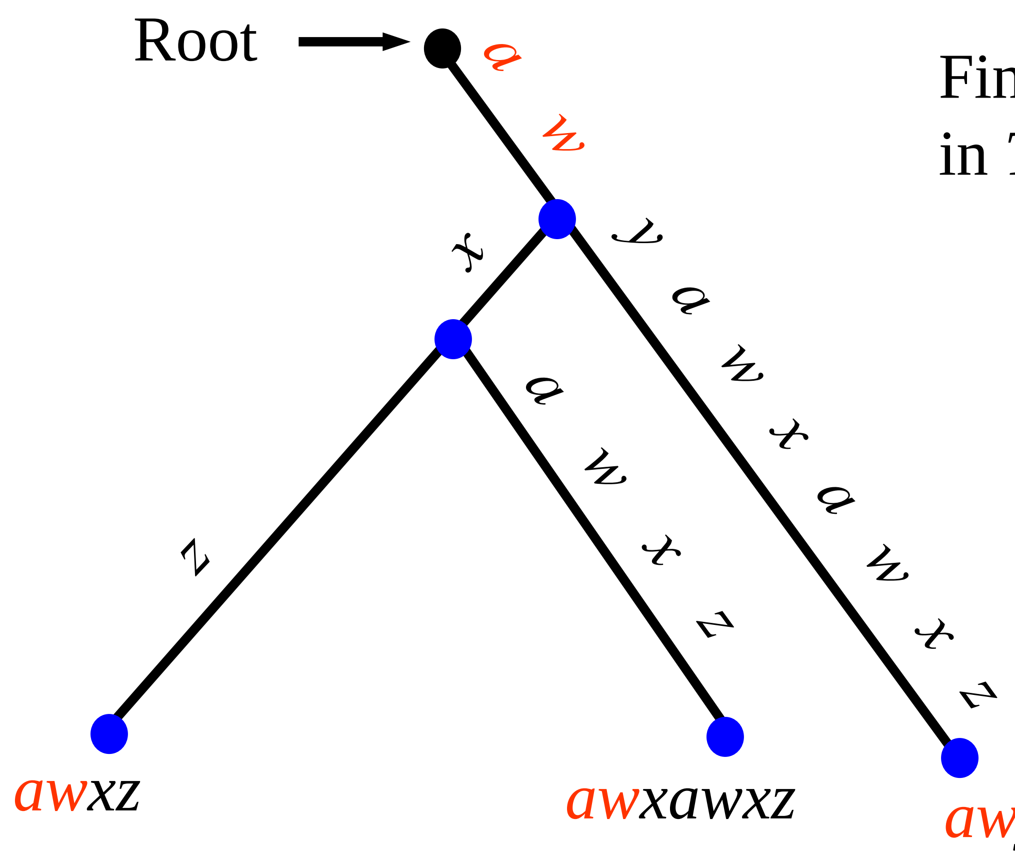
- Build a suffix tree T for text *T*.
- Match the characters of *P* along the **unique** path in T beginning at the root until
  - *P* is exhausted or
  - no more matches are possible

In case 2, *P* does not appear anywhere in *T.*

In case 1, *P* is a prefix of a suffix
obtained by extending the path until we reach a leaf.

Each extension gives a suffix a prefix of which is *P*,
thus, each extension provides an occurrence of *P* in *T*.

Example



Find all occurences of *P* = *aw* in *T* = *awyawxawxz.*

awyawxawxz

Constructing a Suffix Trees

- A suffix tree can be constructed in linear time

[Weiner73, McCreight76, Ukkonen95]

Complexity of Pattern matching

Time complexity

Preprocessing : $O(|T|)$

Searching : $O(|P| + k)$,

where k is # occurences of P in T

Space complexity

$O(|P| + |T|)$