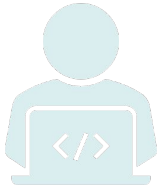
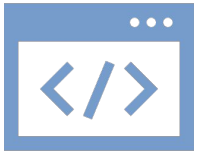
Tries

Prepared by:
Dr Annushree Bablani

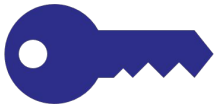
Introduction



Numbers as key values: are data items of constant size, and can be compared in constant time.



In real applications, text processing is more important than the processing of numbers



We need different structures for strings than for numeric keys.

Motivating Example

- Example: 112 < 467 , Numerical comparison in O(1).
- Compare Strings lexicographically does not reflect the similarity of strings.
  - Western > Eastern , Strings comparison in O(min(|s1|,|s2|)). where |s| denotes the length of the string s
- Text fragments have a length; they are not elementary objects that the computer can process in a single step.
  - Pneumonoultramicroscopicsilicovolcanoconiosis !!!

Applications

Bioinformatics

Search Engines.

Spell checker.

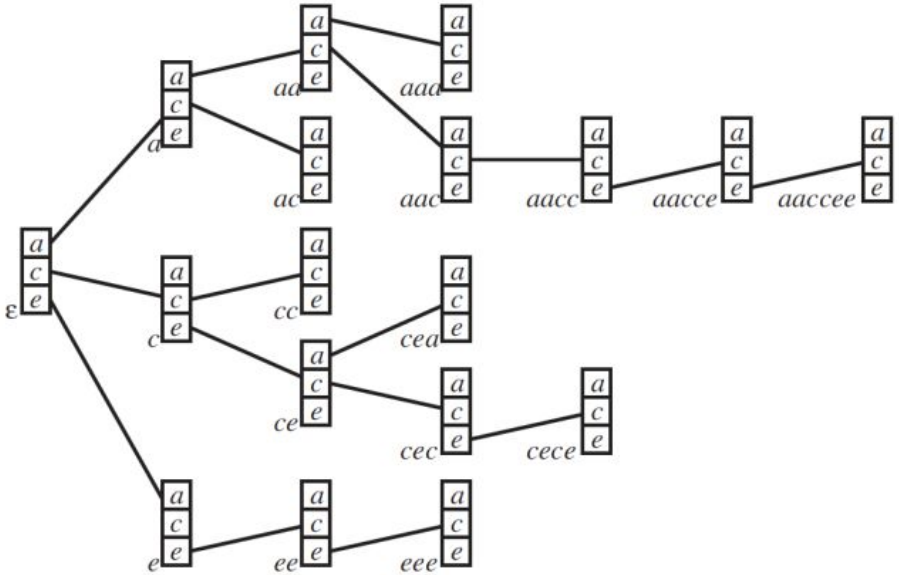
Tries

- The basic tool for string data structures, similar in role to the balanced binary search tree, is called “trie”
- Derive from “retrieval.” (Pronounced either try or tree)
- In this tree, the nodes are not binary. They contain potentially one outgoing edge for each possible character, so the degree is at most the alphabet size $|A|$.

Tries cont.

- Prefix Vs. Suffix.
- Ex. “computer”.
    - Prefix:(c, co, com).
    - Suffix: (r, er, ter)
- Each node in this tree structure corresponds to a prefix of some strings of the set.
- If the same prefix occurs several times, there is only one node to represent it.
- The root of the tree structure is the node corresponding to the empty prefix.

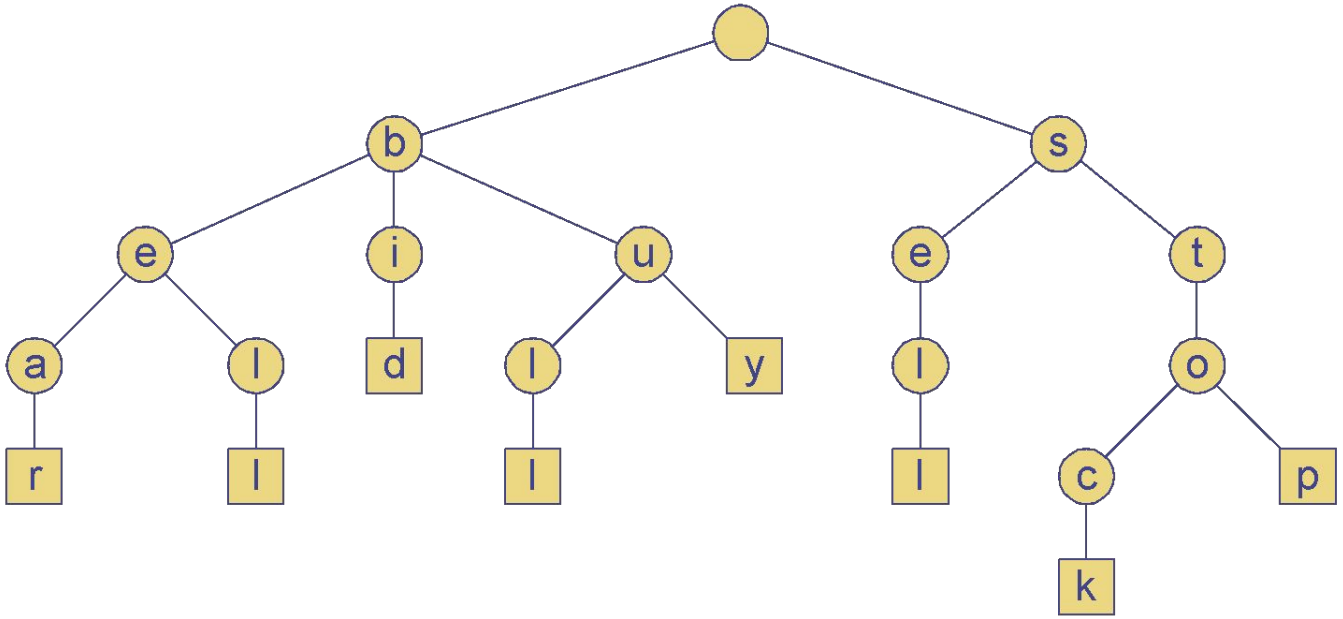
Tries Example



TRIE OVER ALPHABET $\{a, v, e\}$ WITH NODES FOR THE WORDS *aaa*, *aaccee*, *ac*, *cc*, *cea*, *cece*, *eee*, AND THEIR PREFIXES

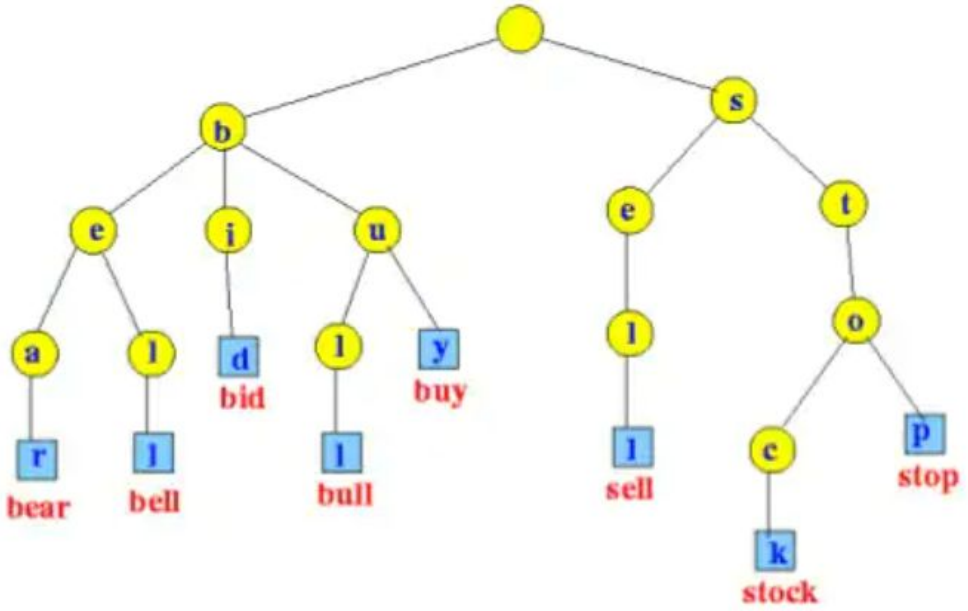
Tries Example

- Example: standard trie for the set of strings
  - S = { bear, bell, bid, bull, buy, sell, stock, stop }



Tries

- Standard Tries
- Compressed Tries
- Suffix Tries



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
- Example: standard trie for

  the set of strings

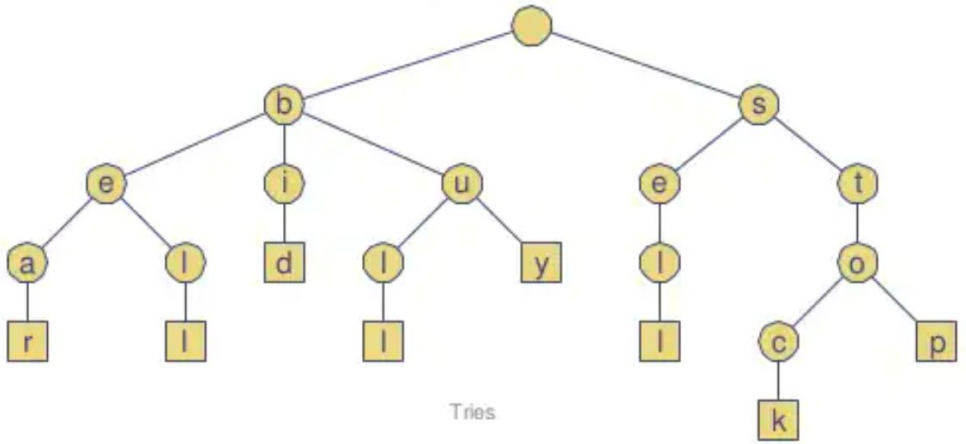
  S = { bear, bell, bid, bull,
  buy, sell, stock, stop }

- A standard trie uses O(n) space. Operations (find, insert, remove) take time O(dm) each, where:

-n = total size of the strings in S,

-m =size of the string parameter of the operation

-d =alphabet size,



TRIE Representation

```
struct Trie
{
        struct Trie* S[26];   //a-z  or A-Z
        bool isEndOfWord;
};
```

Applications of Tries

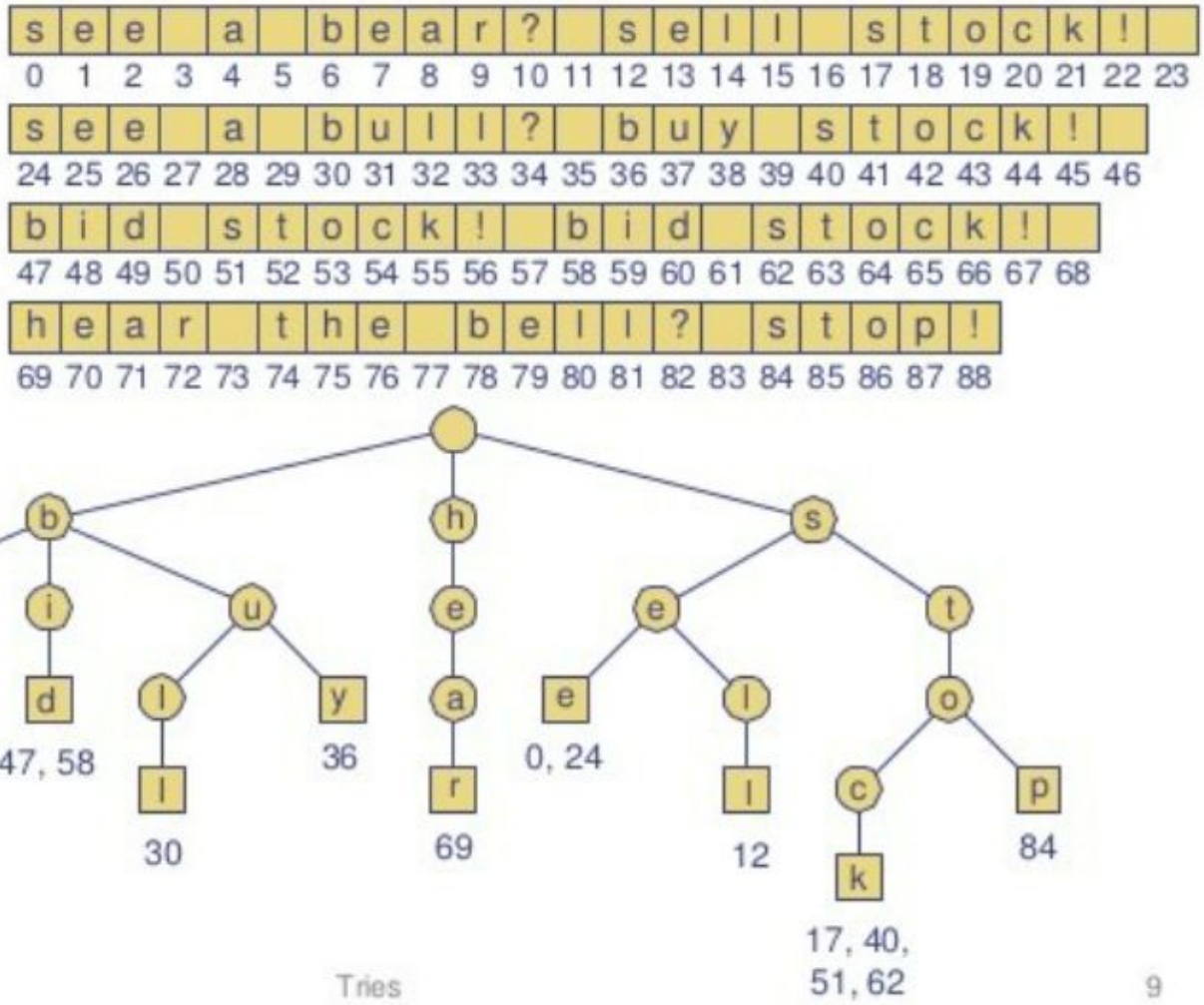
- A standard trie supports the following operations on a preprocessed text in time O(m), where m = |X|

  -***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

Word Matching with a Trie

- We insert the words of the text into a trie
- Each leaf stores the occurrences of the associated word in the text



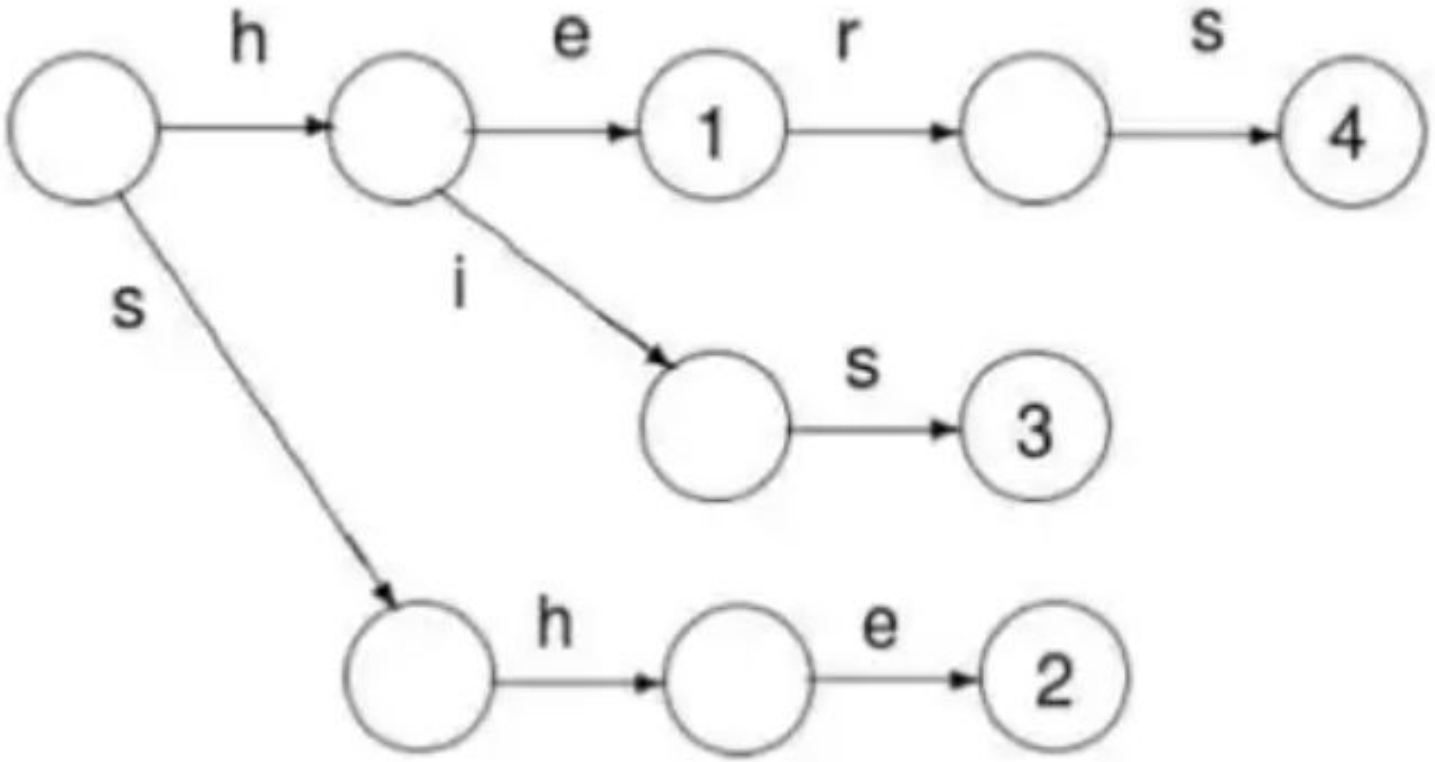
Prefix : What is prefix:

- The prefix of a string is nothing but any n letters $n \leq |S|$ that can be considered beginning strictly from the starting of a string.
- For example , the word "abacaba" has the following prefixes:

a
ab
aba
abac
abaca
abacab

Common Prefix

Trie for arr[] = {he, she, his, hers}



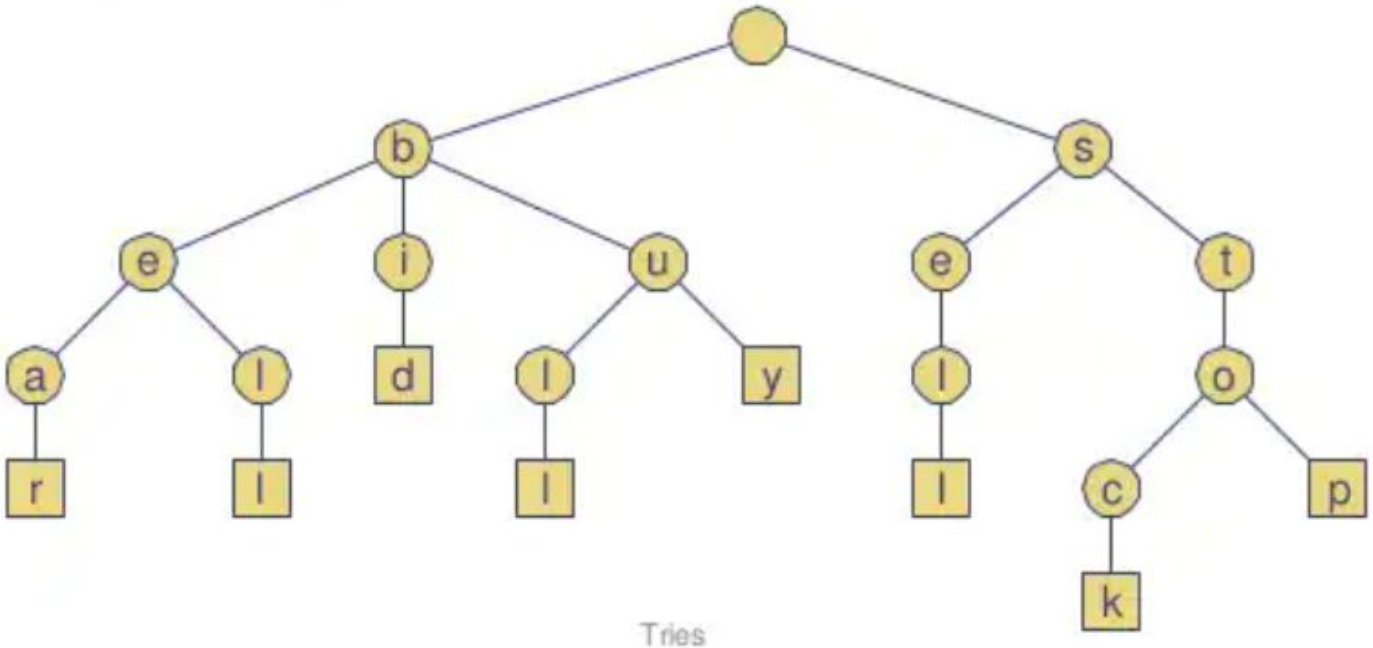
Suffix

- The suffix of a string is nothing but any n letters $n \leq |S|$ that can be considered ending strictly at the end of a string.
- For example , the word "abacaba" has the following prefixes:

a
ba
aba
caba
acaba
bacaba

Analysis of Standard Tries

- A standard trie uses $O(n)$ space and supports searches, insertions and deletions in time $O(dm)$, where:
  - n total size of the strings in S
  - m size of the string parameter of the operation
  - d size of the alphabet



Find, Insert and Delete

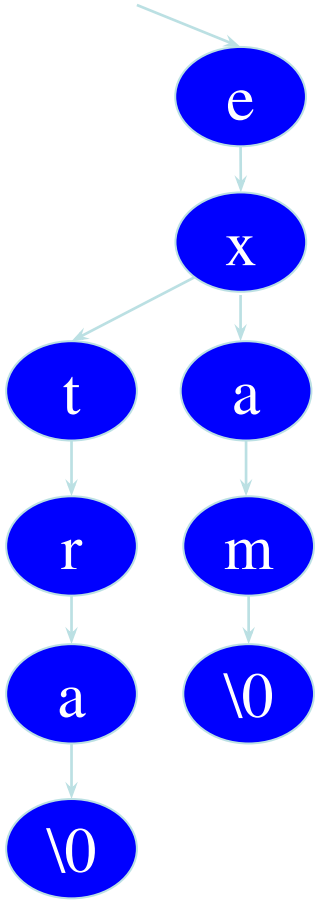
- To perform a *find* operation in this structure:

1. Start in the node corresponding to the empty prefix.
2. Read the query string, following for each read character the outgoing pointer corresponding to that character to the next node.
3. After we read the query string, we arrived at a node corresponding to that string as prefix.
4. If the query string is contained in the set of strings stored in the trie, and that set is prefix-free, then this node belongs to that unique string.

Find, Insert and Delete

- To perform an *insert* operation in this structure:
  1. Perform *find*
  2. Any time we encounter a nil pointer we create a new node.

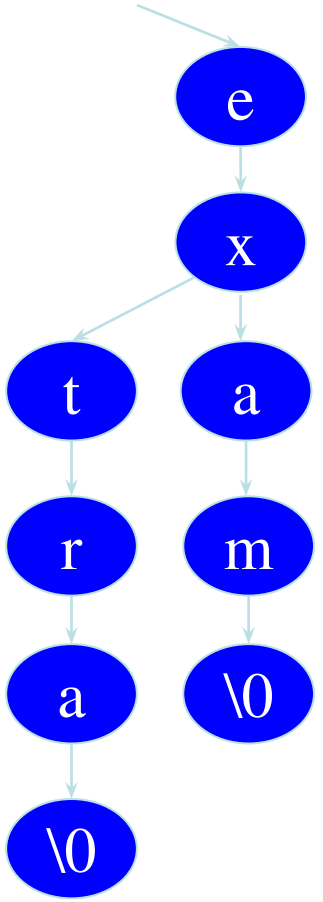
- Example:
  Insert “extra”



Find, Insert and Delete

- To perform a *delete* operation in this structure:
  1. Perform *find*
  2. Delete all nodes on the path from '\0' to the root of the tree unless we reach a node with more than 1 child

- Example:
  - Delete "extra"



Performance

Theorem. The basic trie structure stores a set of words over an alphabet A. It supports a `find` operation on a query string q in time $O(\text{length}(q))$ and `insert` and `delete` operations in time $O(|A|\,\text{length}(q))$. The space requirement to store n strings $w_1, \ldots, w_n$ is $O(|A| \sum_i \text{length}(w_i))$.

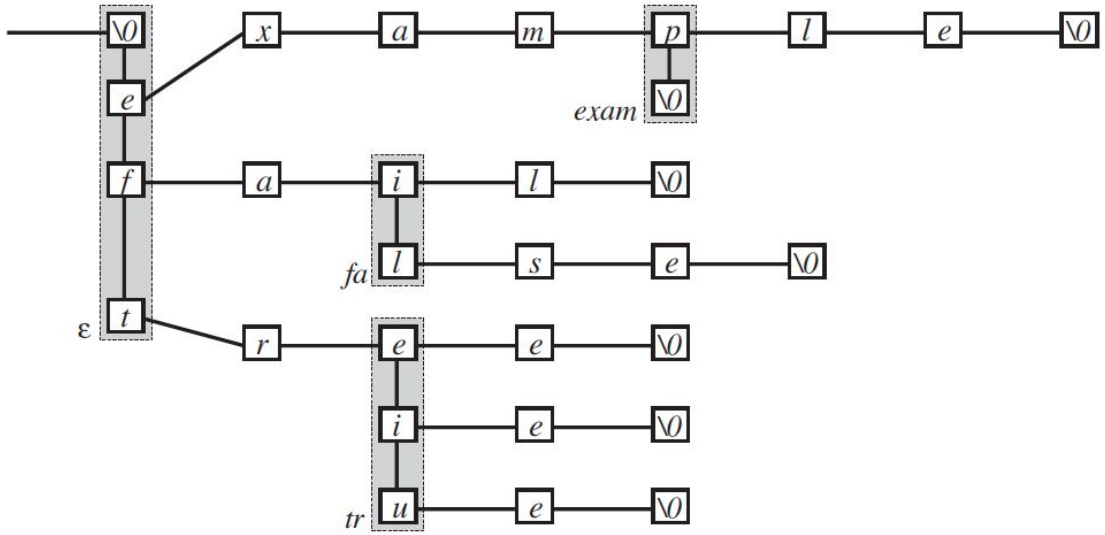
- *Find*: All the characters in the word = O(|q|) where q*: query string*
- *Insert*: first *find* then *insert* an array of length |A| as a node = O(|q|.|A|)
- *Delete*: first *find* then *delete* an array of length |A| as a node = O(|q|.|A|)

- We can get rid of the |A| dependence in the *delete* operation by using
- reference counts.
- All new nodes must be initialized with NULL pointers, so the |A| dependence in the insert operation does not disappear.

Alphabet Size

- The problem here is the dependence on the size of the alphabet which determines the size of the nodes.
- There are several ways to reduce or avoid the problem of the alphabet size.
- A simple method, is to replace the big nodes by linked lists of all the entries that are really used.

List Example



TRIE FOR THE STRINGS *exam, example, fail, false, tree, trie, true* IMPLEMENTED WITH LIST NODES: ALL POINTERS GO RIGHT OR DOWN

Lists Performance

Theorem. The trie structure with nodes realized as lists stores a set of words over an alphabet A. It supports a `find` operation on a query string q in time $O(|A|\,\text{length}(q))$ and `insert` and `delete` operations in time $O(|A|\,\text{length}(q))$. The space requirement to store n strings $w_1, \ldots, w_n$ is $O\left(\sum_i \text{length}(w_i)\right)$.

Find, Insert and *delete:* O(|q|.|A|)

Alphabet Size

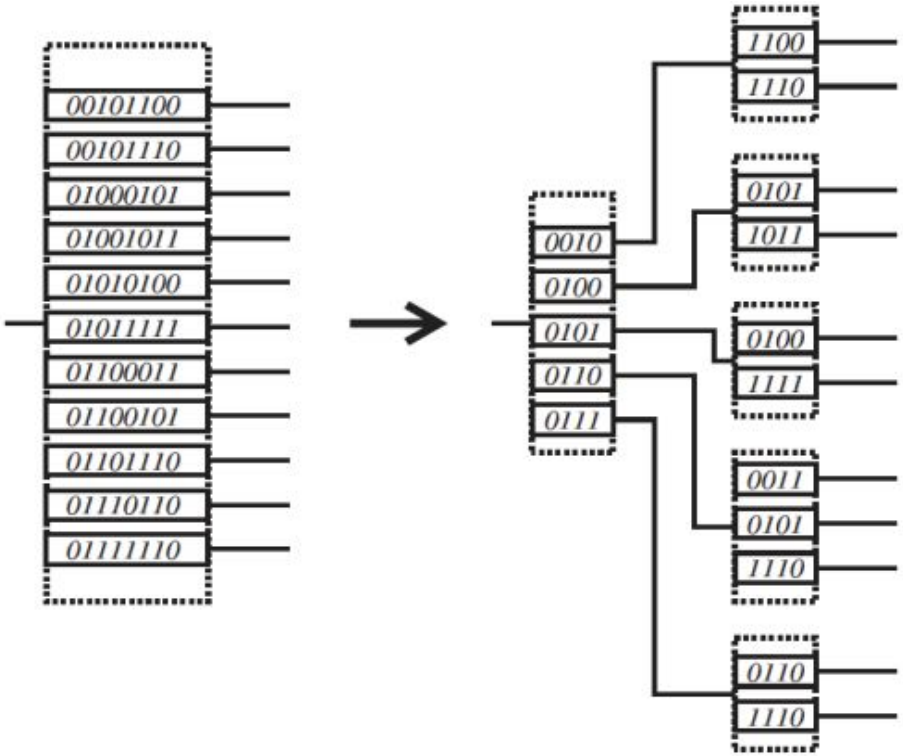
- Another way to avoid the problem with the alphabet size |A| is alphabet reduction.
- We can represent the alphabet A as set of k -tuples from some direct product

$$A_1 \times \cdots \times A_k$$

- By this each string gets longer by a factor of k , but the alphabet size can be reduced to $\lceil |A|^{\frac{1}{k}} \rceil$

Alphabet Reduction Example

- For the standard ASCII codes, we can break each 8-bit character by two 4-bit characters, which reduces the node size from 256 pointers to 16 pointers



ALPHABET REDUCTION: INSTEAD OF ONE NODE WITH 256 ENTRIES, OF WHICH ONLY 11 ARE USED, WE HAVE FIVE NODES WITH 16 ENTRIES EACH

Reduction Performance

Theorem. The trie structure with k-fold alphabet reduction stores a set of words over an alphabet A. It supports `find` and `delete` operations on a query string q in time $O(k \text{ length}(q))$ and `insert` operations in time $O(k|A|^{\frac{1}{k}} \text{ length}(q))$. The space requirement to store n strings $w_1, \ldots, w_n$ is $O(k|A|^{\frac{1}{k}} \sum_i \text{length}(w_i))$.

Other Reduction Techniques

- The trie structure with balanced search trees as nodes:
  - *Find, insert* and *delete* Time: $O(\log |A| \text{length}(q))$
  - Space: $O(\sum_i \text{length}(w_i))$

- The ternary trie structure: nodes are arranged in a manner similar to a binary search tree, but with up to three children. each node contains one character as key and one pointer each for query characters that are smaller, larger, or equal
  - *Find* time: $O(\log n + \text{length}(q))$
  - Space: $O\left(\sum_i \text{length}(w_i)\right)$

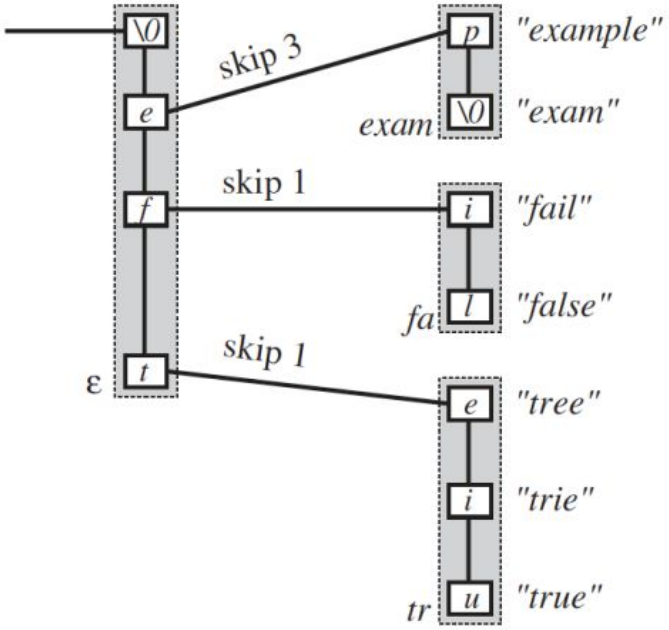
Patricia Tree

- "Practical Algorithm To Retrieve information Coded in Alphanumeric."
- A path compression trie.
- Instead of explicitly storing nodes with just one outgoing edge, we skip these nodes and keep track of the number of skipped characters.
- The path compressed trie contains only nodes with at least two outgoing edges.

Patricia Tree

- It contains a number, which is the number of characters that should be skipped before the next relevant character is looked at.
- This reduces the required number of nodes from the total length of all strings to the number of words in our structure.
- We need in each access a second pass over the string to check all those skipped characters of the found string against the query string.
- this technique to reduce the number of nodes is justified only if the alphabet is large.

Patricia Tree: Example



PATRICIA TREE FOR THE STRINGS *exam, example, fail, false, tree, trie, true:* NODES IMPLEMENTED AS LISTS; EACH LEAF CONTAINS ENTIRE STRING

Patricia Tree: *Insert & Delete*

- The insertion and deletion operations create significant difficulties.
- We need to find where to insert a new branching node, but this requires that we know the skipped characters.

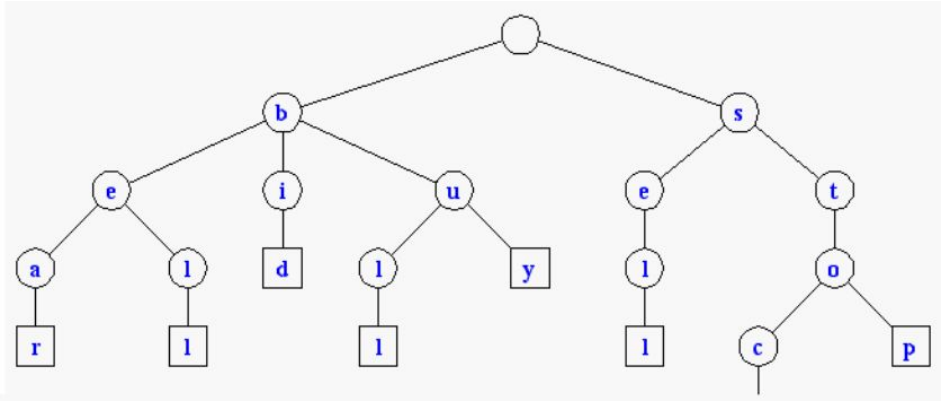
- One (clumsy) solution would be a pointer to one of the strings in the subtrie reached through that node, for there we have that skipped substring already available.

Theorem. The Patricia tree structure stores a set of words over an alphabet A. It supports `find` operations on a query string q in time $O(\text{length}(q))$ and `insert` and `delete` operations in time $O(|A|\,\text{length}(q))$. The space requirement to store n strings $w_1, \ldots, w_n$ is $O(n|A| + \sum_i \text{length}(w_i))$.

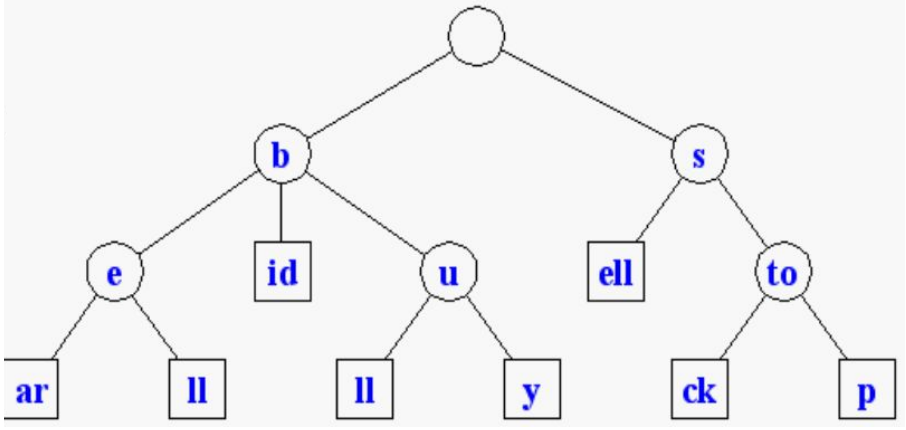
Compressed Tries

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

Standard Trie:

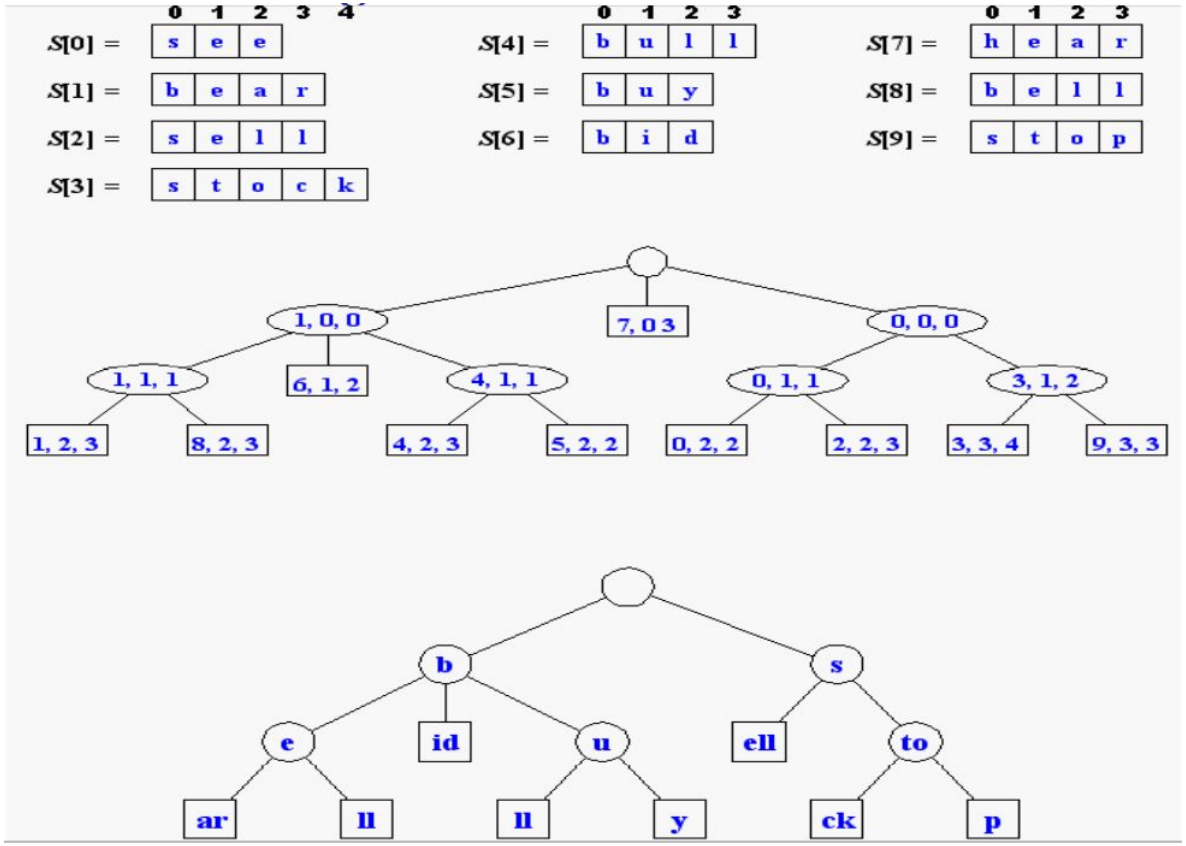


Compressed Trie:

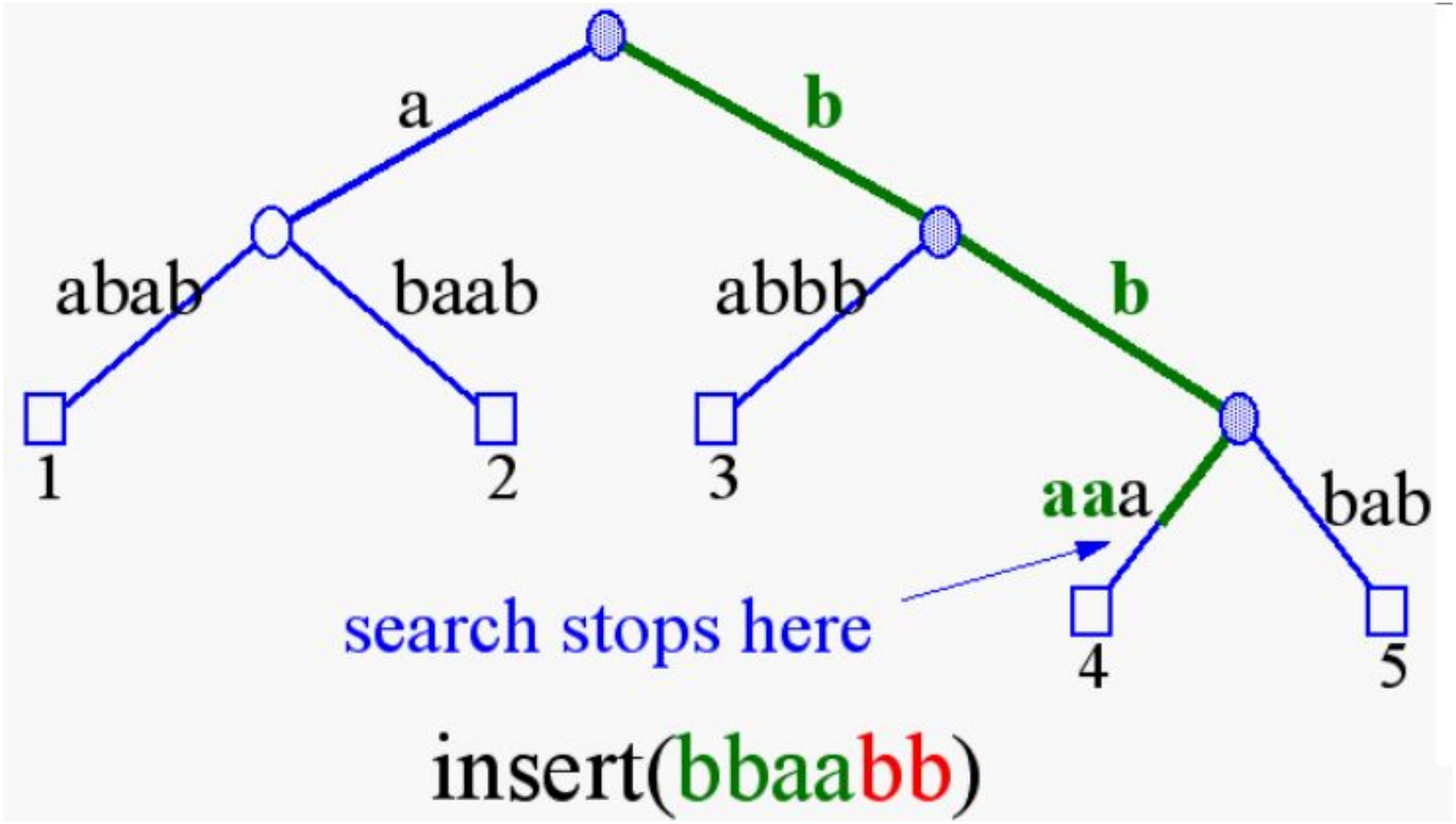


Compact Storage of Compressed Tries

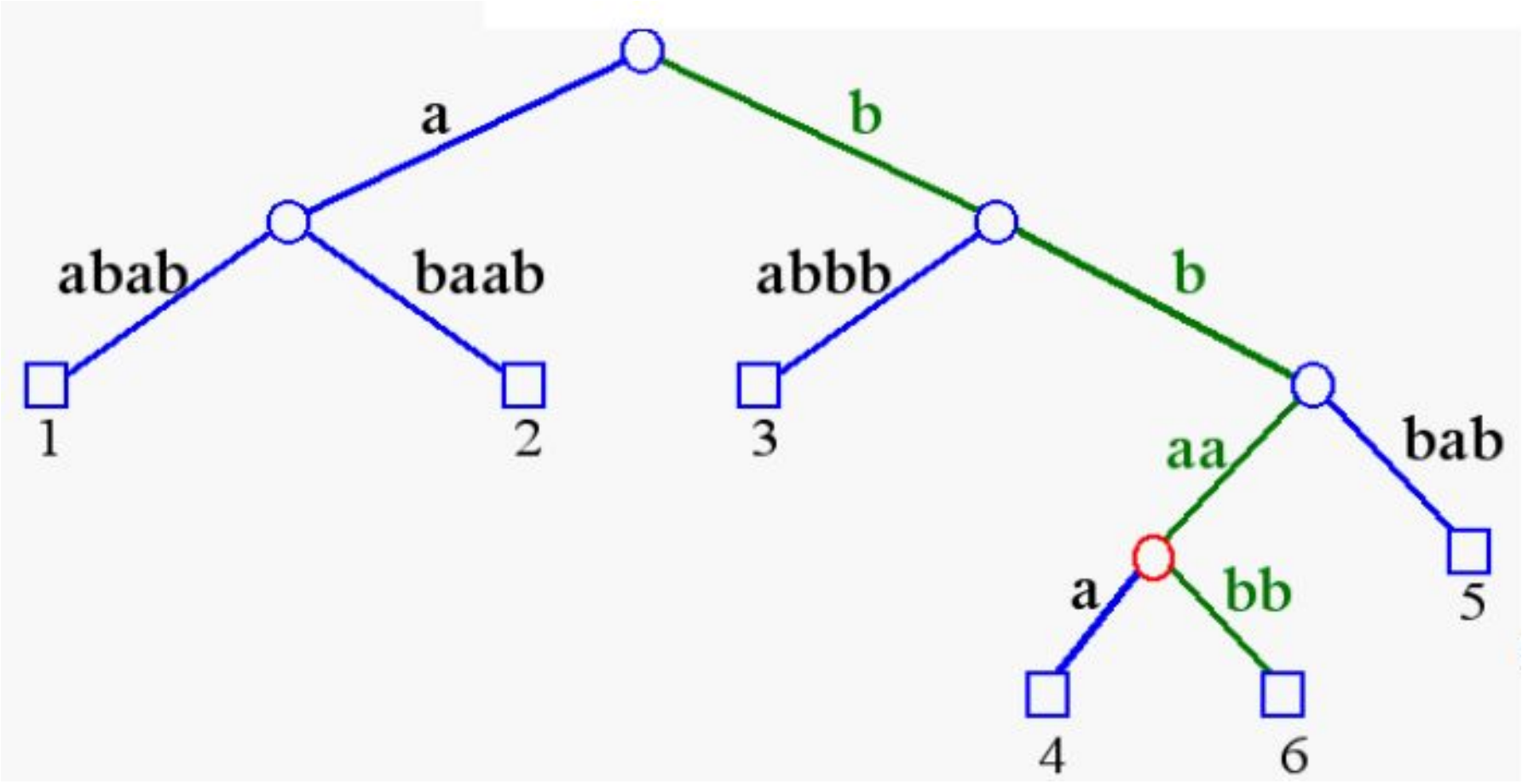
- A compressed trie can be stored in space O(s), where s = |S|, by using O(1) space ***index ranges*** at the nodes



Insertion and Deletion into/from a Compressed Trie



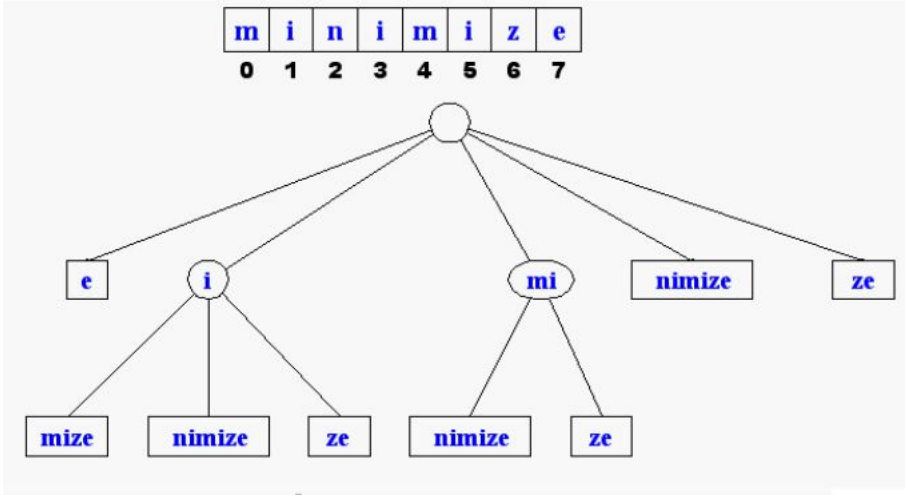
Insertion and Deletion into/from a Compressed Trie



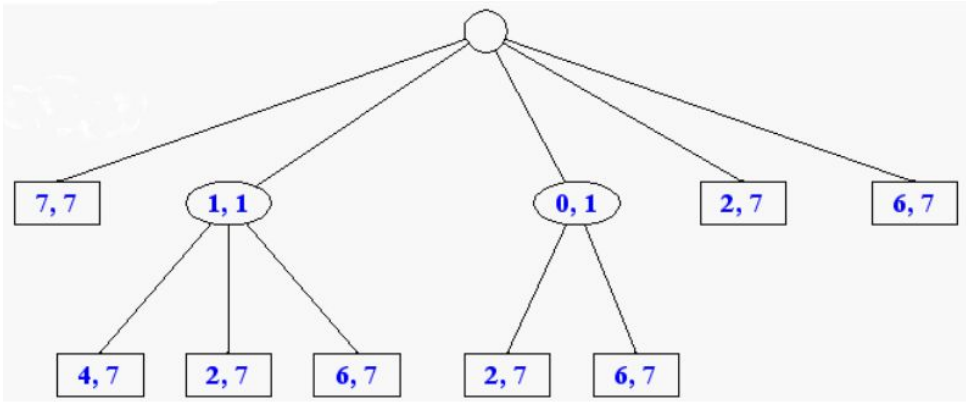
Suffix Tries

- A ***suffix trie*** is a compressed trie for all the suffixes of a text

Example:

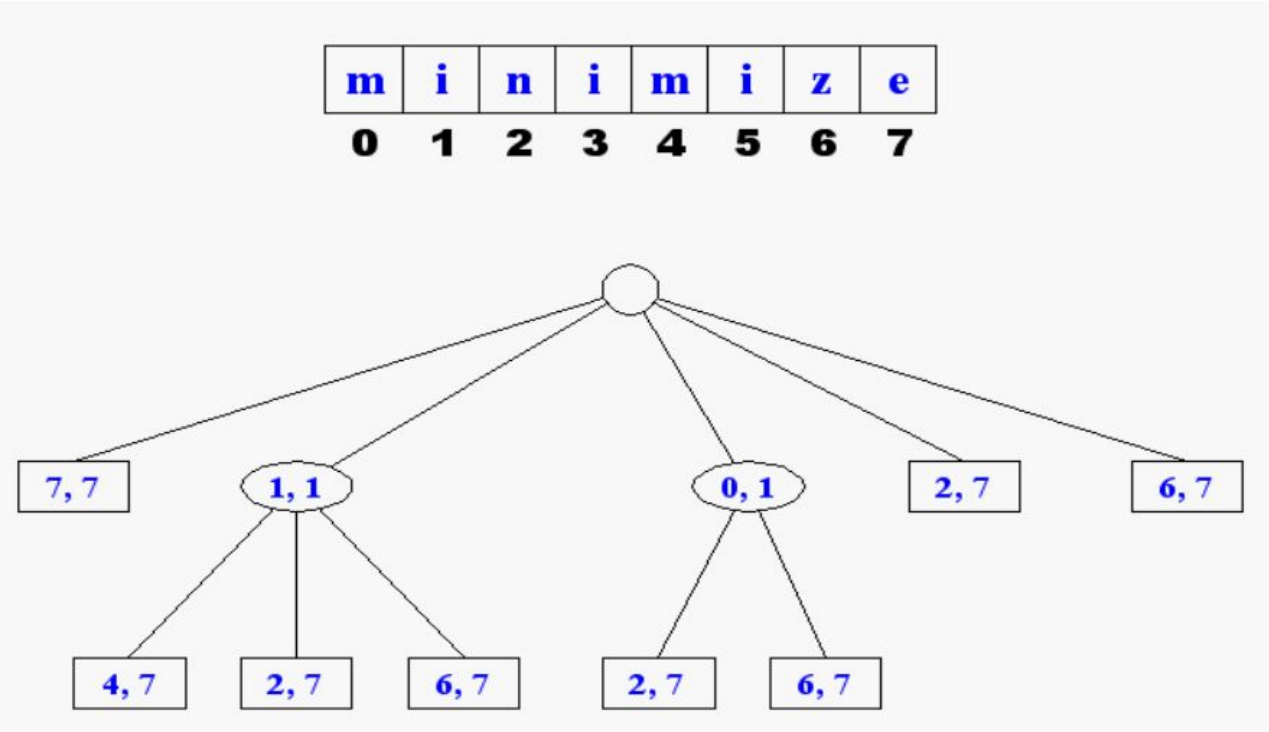


Compact representation:



Properties of Suffix Tries

- The ***suffix trie*** for a text X of size ***n*** from an alphabet of size ***d***
- -stores all the ***n(n-1)/2 suffixes*** of X in O(n) space
- -supports arbitrary ***pattern matching*** and prefix matching queries in ***O(dm) time***, where m is the length of the pattern
- -can be constructed in ***O(dn) time***



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***) addres, usually written in "dotted-quad-decimal" notation
- E.g., www.cs.brown.edu is 128.148.32.110
- Use nslookup on Unix to find out IP addresses
- An organization uses a subset of IP addresses with the same prefix, e.g., Brown uses 128.148.*.*, Yale uses 130.132.*.*
- Data is sent to a host by fragmenting it into packets. Each packet carries the IP address of its destination.
- The internet whose nodes are ***routers***, and whose edges are communication links.
- A router forwards packets to its neighbors using IP ***prefix matching*** rules. E.g., a packet with IP prefix 128.148. should be forwarded to the Brown gateway router.
- Routers use tries on the alphabet 0,1 to do prefix matching.