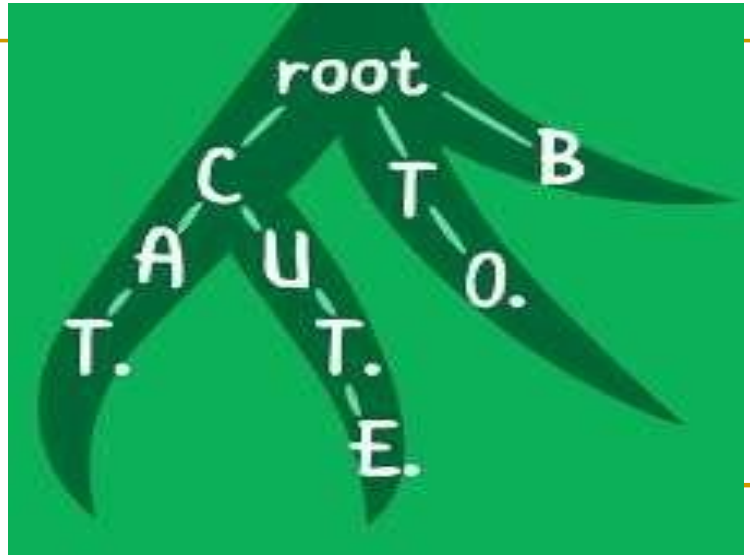


Tries

What is a TRIE?

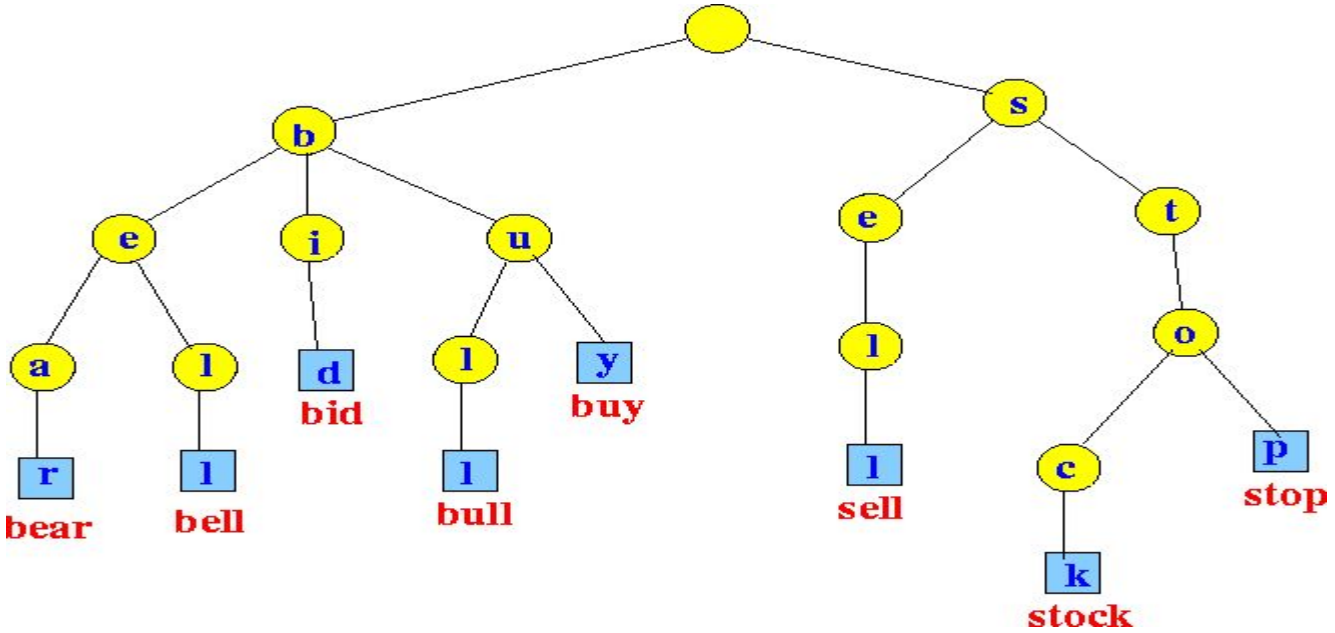
- Tree based data structure used for Information Re**TRIE**val task
- Also called as Digital Tree, Prefix tree, Radix Tree
- Trie is used mostly for storing strings in a *compact* way. E.g. words in dictionary
- A tries supports pattern matching queries in time proportional to the pattern size

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 }

Trie – An ordered Tree

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



TRIE Representation

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

Tries for number

- Name | Social Security Number (SS#)

Jack | 951-94-1654

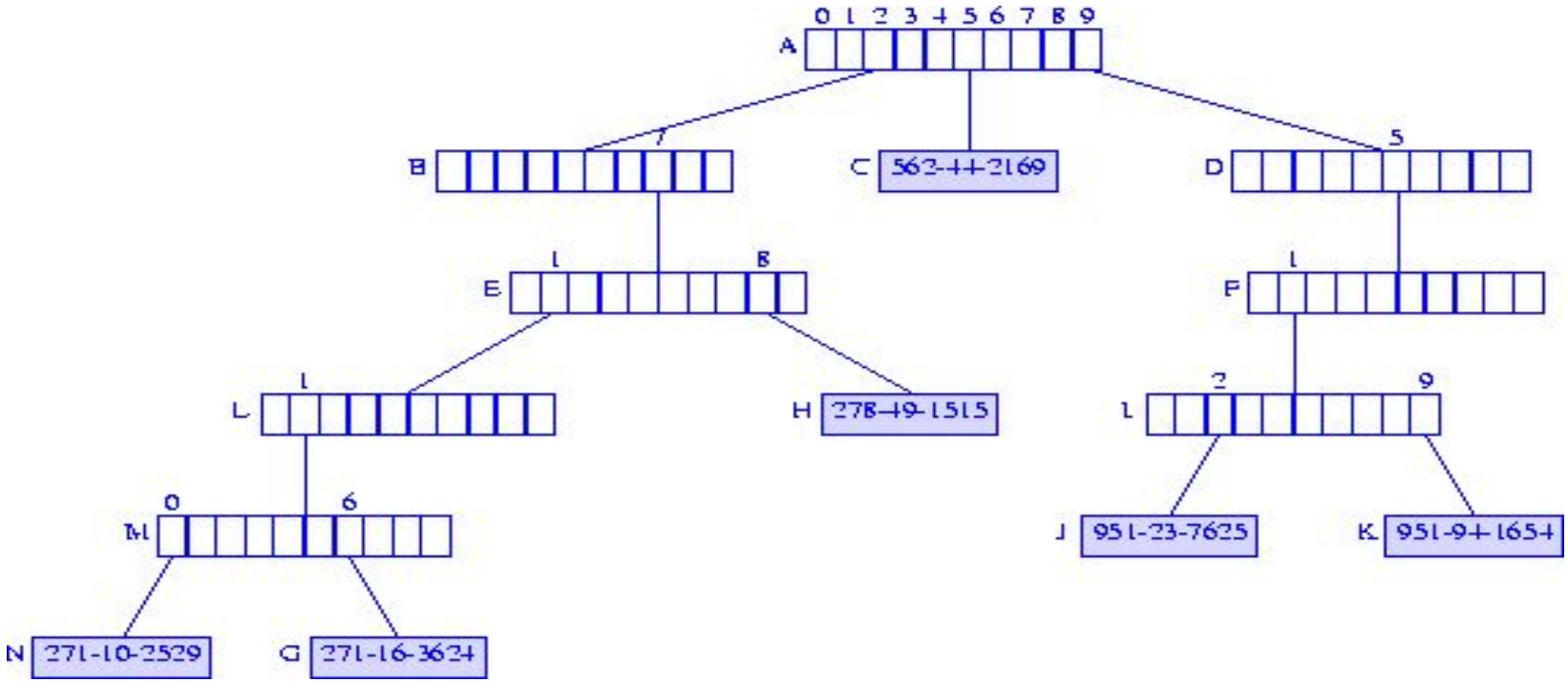
Jill | 562-44-2169

Bill | 271-16-3624

Kathy | 278-49-1515

April | 951-23-7625

Bill | 271-16-3624
Kathy | 278-49-1515



Operations

- Insert – top-down traversal
- Delete – bottom-up
- Search – top-down

Inserting into a Trie

- Proceed before as if doing an normal lookup, adding in new nodes as needed.
- Set the “is word” bit in the final node visited this way.

Insert Word into TRIE

Insert “their”

Code for Insertion of a string into a TRIE

```
void insert(struct TrieNode *root, string key)
{
    struct TrieNode *pCrawl = root;

    for (int i = 0; i < key.length(); i++)
    {
        int index = key[i] - 'a';
        if (!pCrawl->children[index])
            pCrawl->children[index] = getNode();

        pCrawl = pCrawl->children[index];
    }

    // mark last node as leaf
    pCrawl->isEndOfWord = true;
}
```

```
struct TrieNode *getNode(void)
{
    struct TrieNode *pNode =  new TrieNode;

    pNode->isEndOfWord = false;

    for (int i = 0; i < ALPHABET_SIZE; i++)
        pNode->children[i] = NULL;

    return pNode;
}
```

Search

- To search a trie for an element with a given key,
  - we start at the root and follow a path down the trie until we either fall off the trie (i.e., we follow a null pointer in a branch node)

    or
  - we reach an element node; The path we follow is determined by the alphabets/digits of the search key.

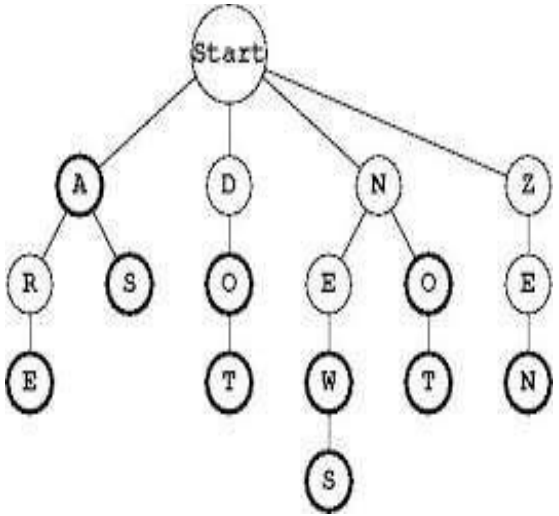
Code to check whether a single word exists in a TRIE

```
bool search(struct TrieNode *root, string key)
{
    struct TrieNode *pCrawl = root;

    for (int i = 0; i < key.length(); i++)
    {
        int index = key[i] - 'a';
        if (!pCrawl->children[index])
            return false;

        pCrawl = pCrawl->children[index];
    }

    return (pCrawl != NULL && pCrawl->isEndOfWord);
}
```



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

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 occurrence of word X in the text

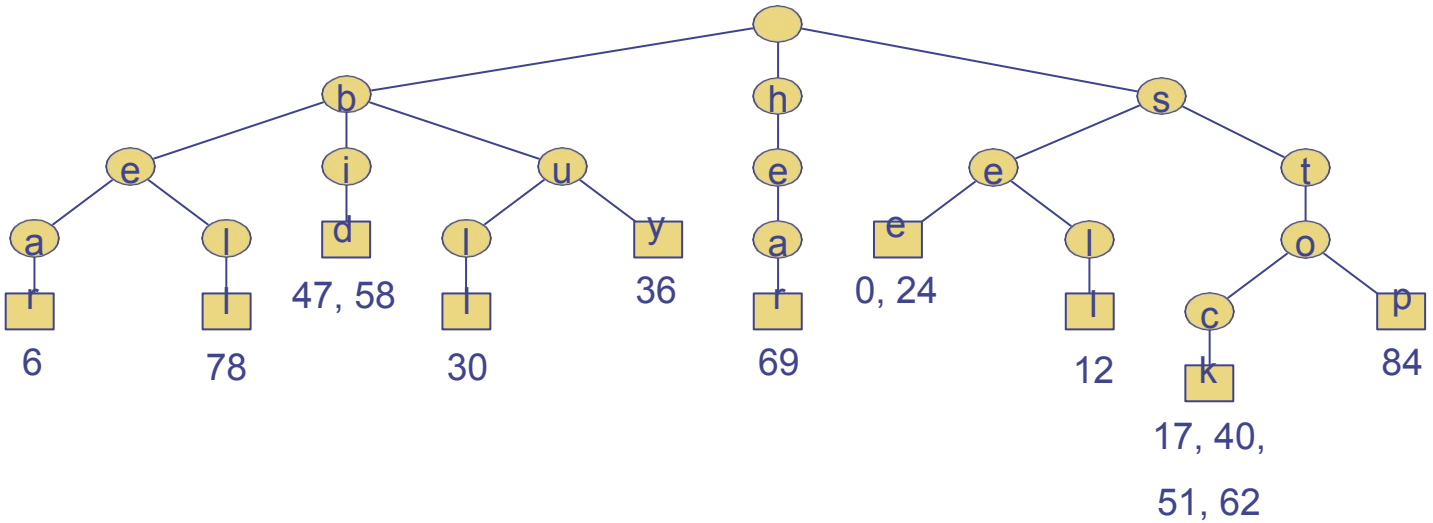
  -***prefix matching***:

  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:

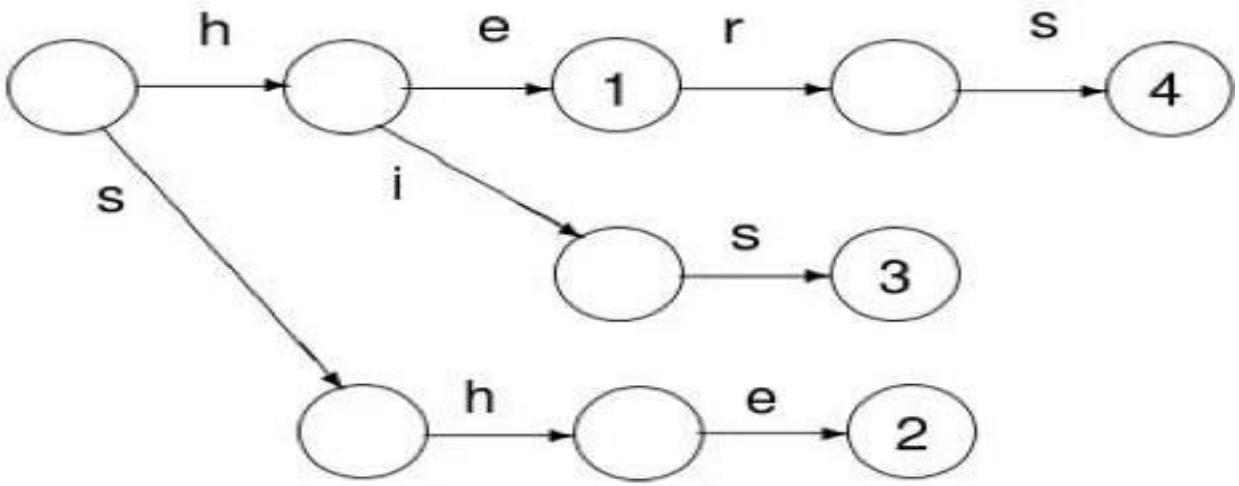
- The prefix of a string is nothing but

  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, abacaba

Common Prefix

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



Longest Common Prefix

Let $S=S[1],\ldots,S[m]$ and $T=T[1],\ldots,T[n]$ be two strings over alphabet Σ.

The Longest Common Prefix (LCP) of S and T is the string $a[1],\ldots,a[k]$ such that $a[i]=S[i]=T[i]$, $i=1,\ldots,k$ and such that $S[k+1]\neq T[k+1]$.

Example: The LCP of

ABCAABCDABCCC and

ABCAABCDACACC is: ABCAABCDA

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, abacaba

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