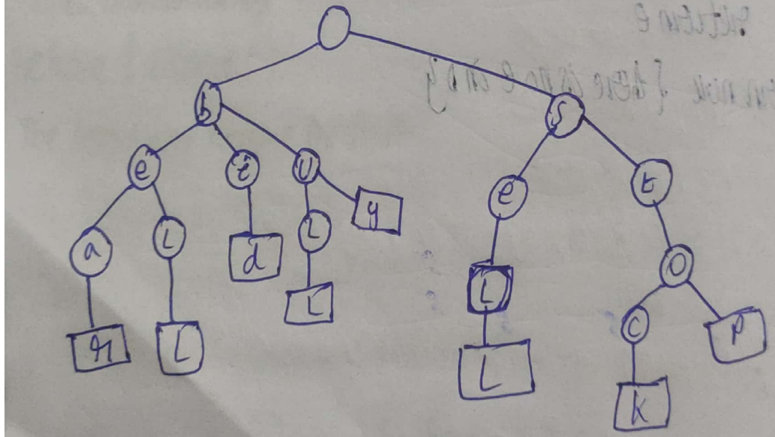
UNIT-4

Text processing : String operations , Brute-force pattern matching. The boyer moore Algorithm , The knuth-morris pratt Algorithm Standard tries , compressed tries , suffix tries , Huffman coding algorithm , the longest subsequence problem (LCS), Applying dynamic programming to the LCS problem.

TRIE:

A data structure for storing a set of strings (name from the word "retrieval"):



PROPERTIES OF TRIE:

(1) multiway tree

(2) Each node has from 1 to d children.

(3) each edge of the tree is labeled with a Character.

(4) Each leaf node corresponds to the stored string, which is a concatenation of characters on a path from the root to this node.

TYPES OF TRIE:

(1) standard trie

(2) compressed trie

(3) suffix trie

STANDARD TRIE:

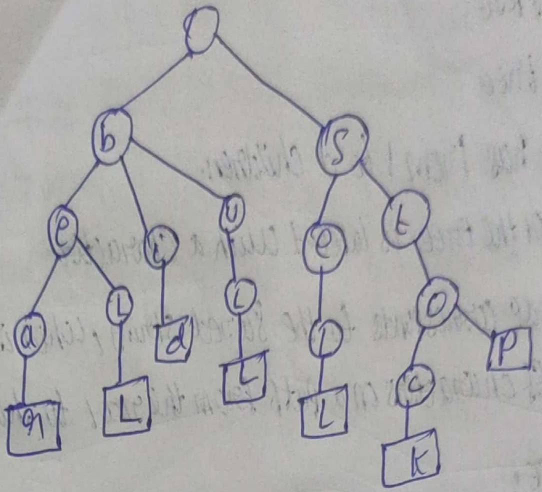
1. Each node but the root is labelled with a character

2. The children of a node are alphabetically ordered.

3. The paths from the external nodes to the root yield to the strings of S.

4- Example:- standard trie for the set of strings S

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



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

n ~ total size of strings chs

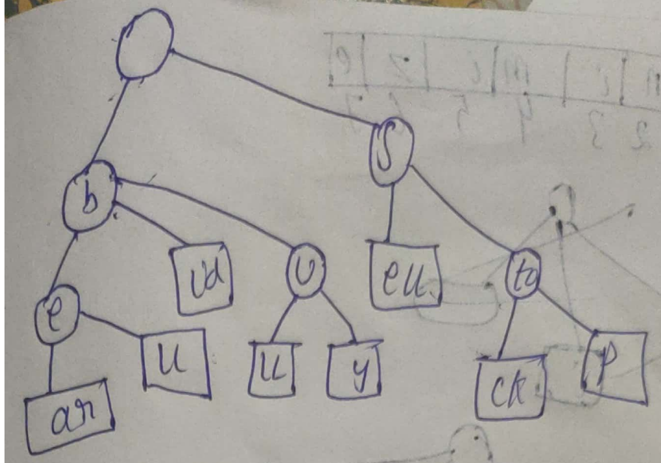
d - alphabet size

m - size of the string parameter of the operation

Compressed tries:

(1) Tree with nodes of degree atleast 2

(2) Obtained from standard trie by compressing the chains of redundant blocks



A compressed trie can be stored in space O(s) wheres= |s| by using O(1)

SUFFIX TRIE:

1) A suffix trie is a compressed trie for all the suffixes of a text.

(2) The suffix trie for a text x of size n from an alphabet of size d.

(3) suffix tries stores all the n(n-1)/2 suffixes of x in O(n) space.

(4) suffix tries supports arbitary pattern matching and prefixes matching queries in O(dm) time, where m is the length of the pattern.

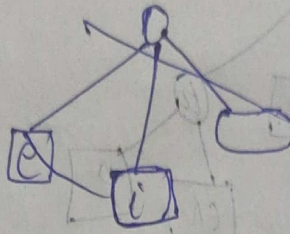
(5) suffix tries can be constructed in O(dn) time.

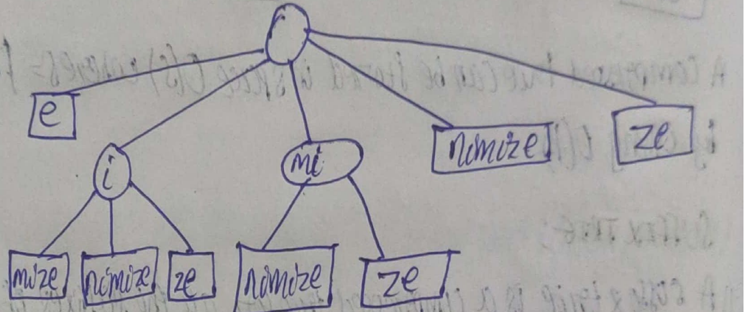
Applications:

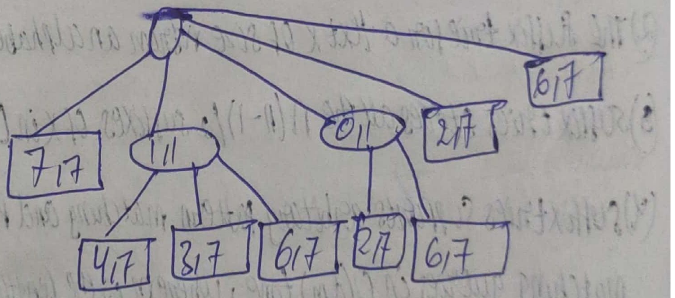
(1) word matching

(2) prefix matching

m	i	n	i	m	i	z	e
0	1	2	3	4	5	6	7



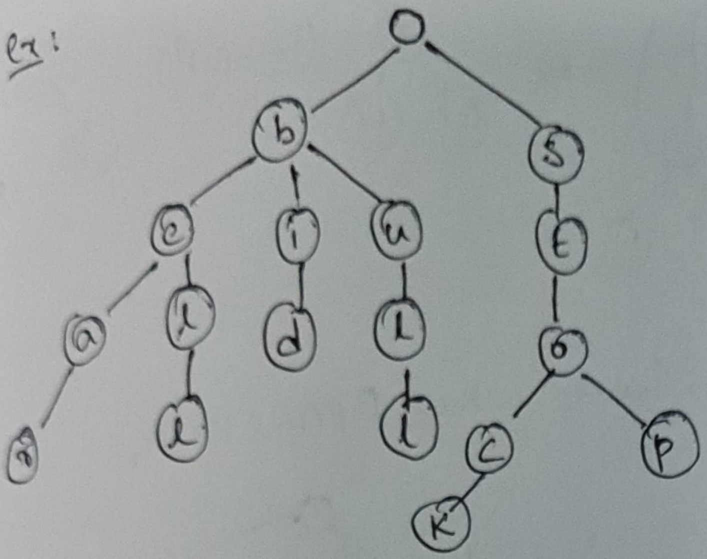




Standard Trie:

In standard trie no word in s should be the prefix of other.

ex:



→
```
class Node {
        Node[] children=new node[26];
        // to check for end of st ring
        boolean iswordEnd;
    }
```

→ It is an ordered tree like data structure

→ Each node (except the root node) is a standard trie is labeled with a character.

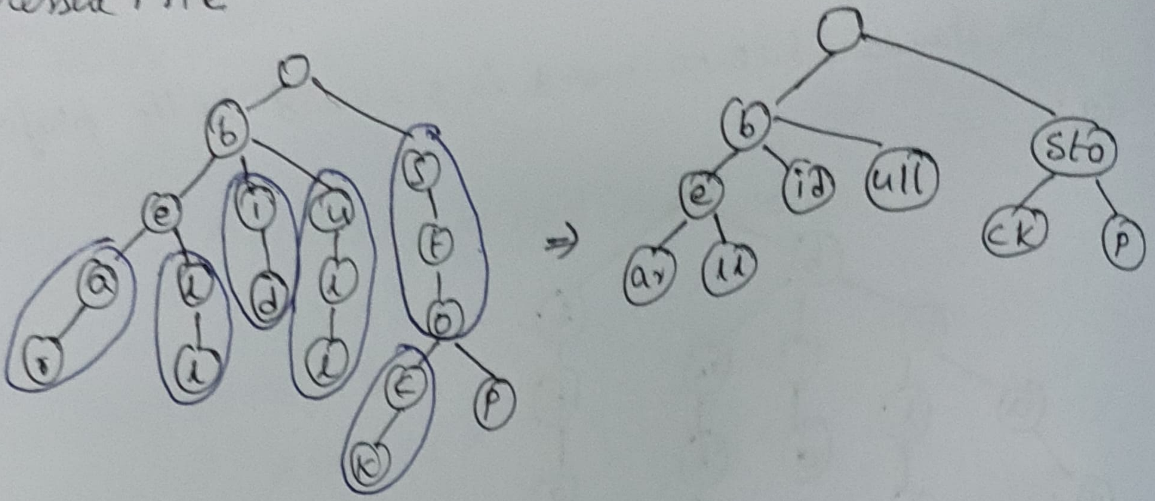
→ The children of a node are in alphabetical order.

→ Each node or branch represents a possible character of keys or words.

→ Each node or branch may have multiple branches.

→ The last node of every key or word is used to mark the end of word or node.

Compressed Trie



NOTE: only add a new node when BRANCHING

i   0 1 2 3 4
s[0] = bear

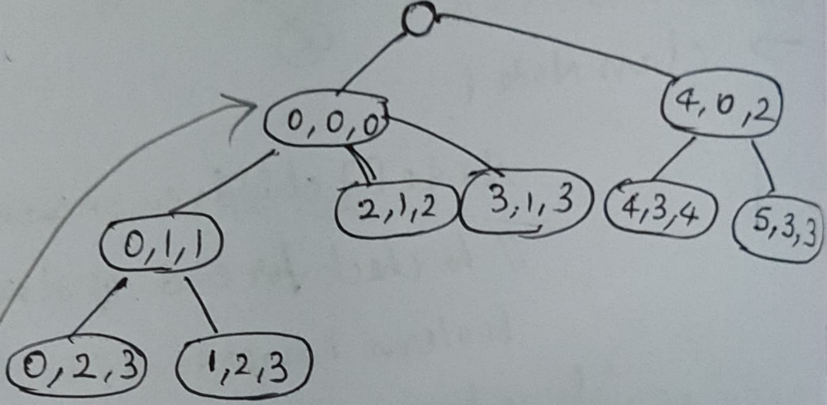
s[1] = bell

s[2] = bid

s[3] = bull

s [4) stock

s[5] = stop



Node (i, j, k)

i - index of S

j - start

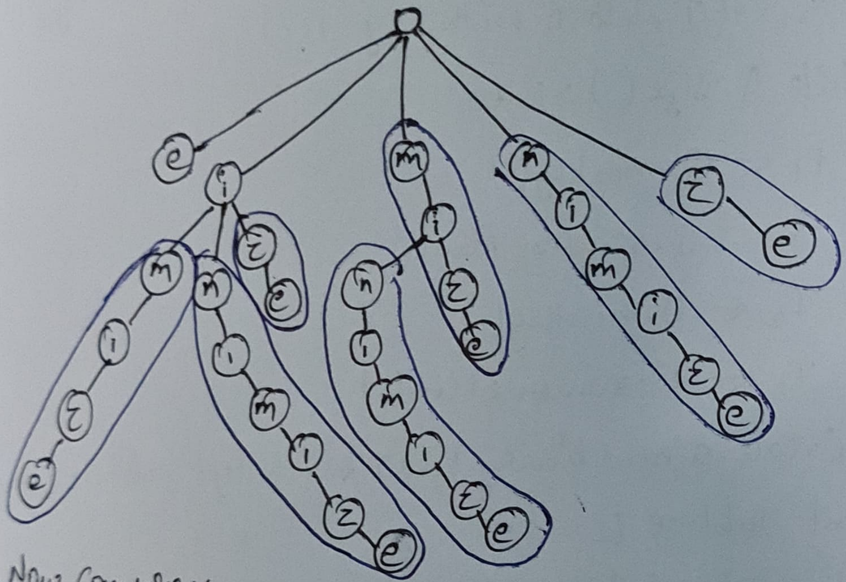
k - end

Suffix ~~Node~~ Trie:

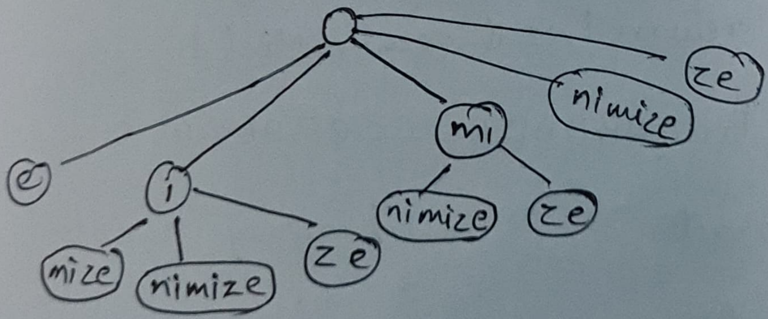
Ex: Minimize

Suffixes:

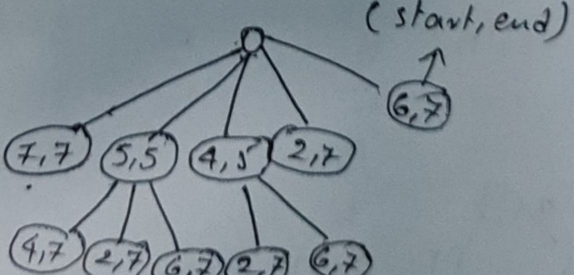
e
ze
ize
mize
imize — S
nimize
inimize
Minimize



→ Now Compress



0 1 2 3 4 5 6 7
minimize



(start, end)
7,7
5,5
4,5
2,7
6,7
4,7
2,7
6,7
2,7
6,7