Trie

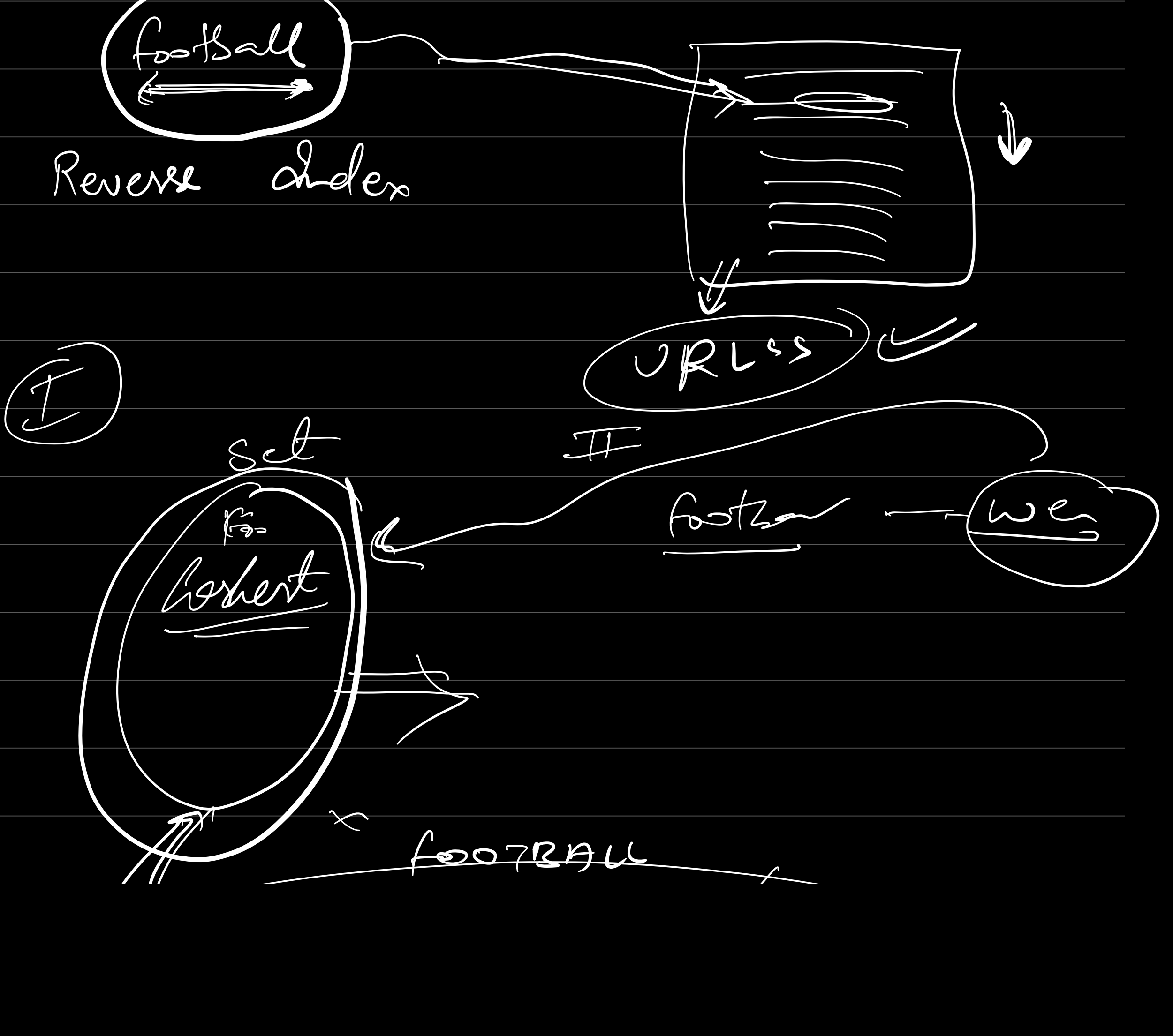
⇒ Tree like structure

re TRIE val

↳ helps to optimize retrieval.

Q. Design a web crawler

Google ⇒ search engine

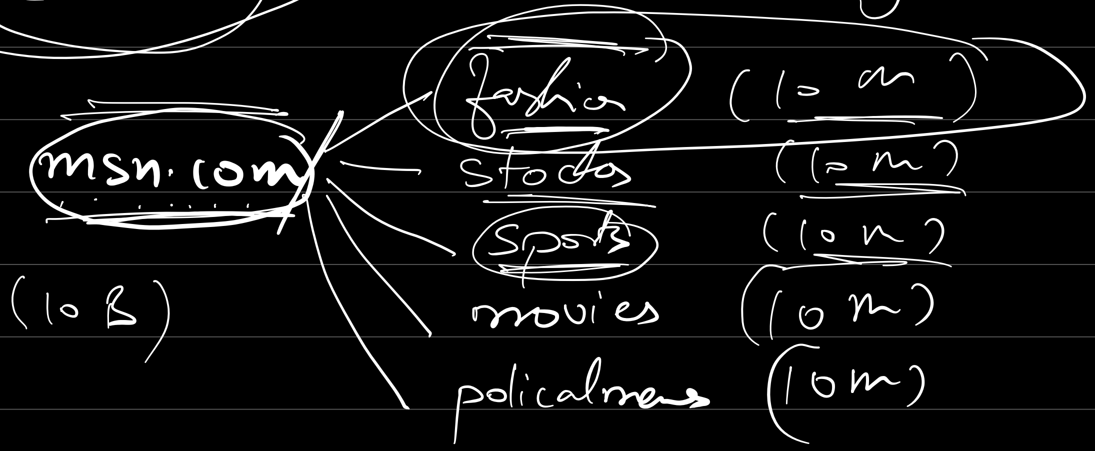


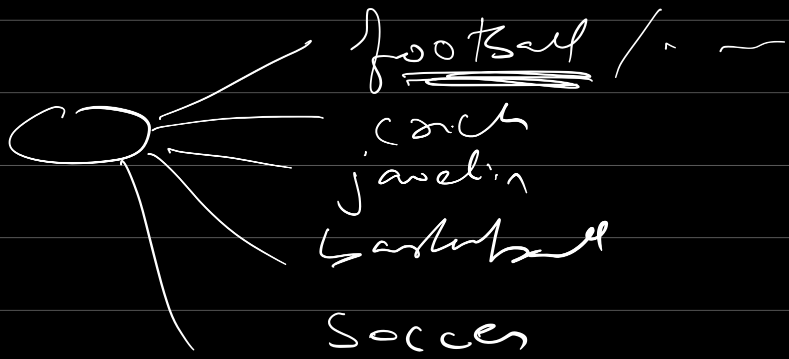
msn.com/sports/football/manu vs barca

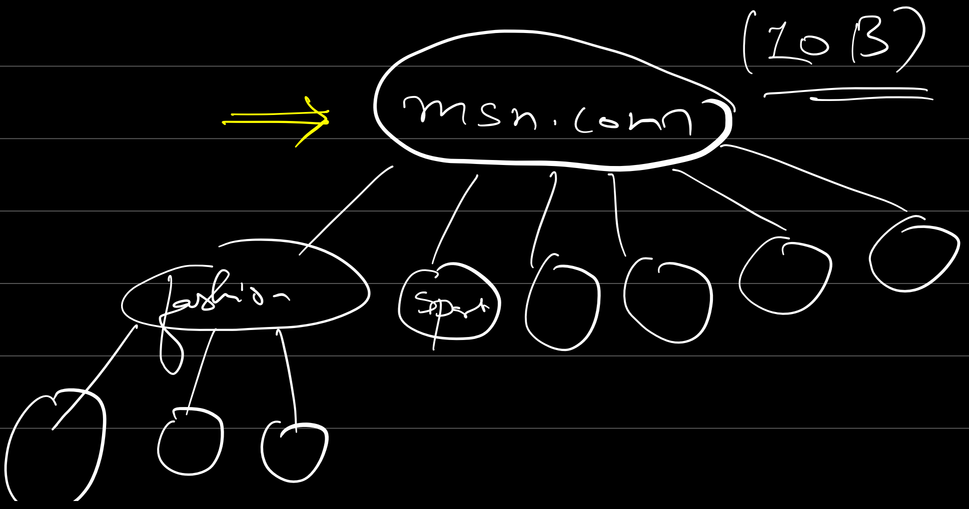
9 Dec 2021.

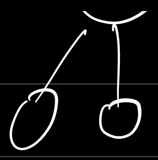
msn.com

msn.com/fash/—





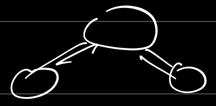




“have shared spaces for shared prefix”

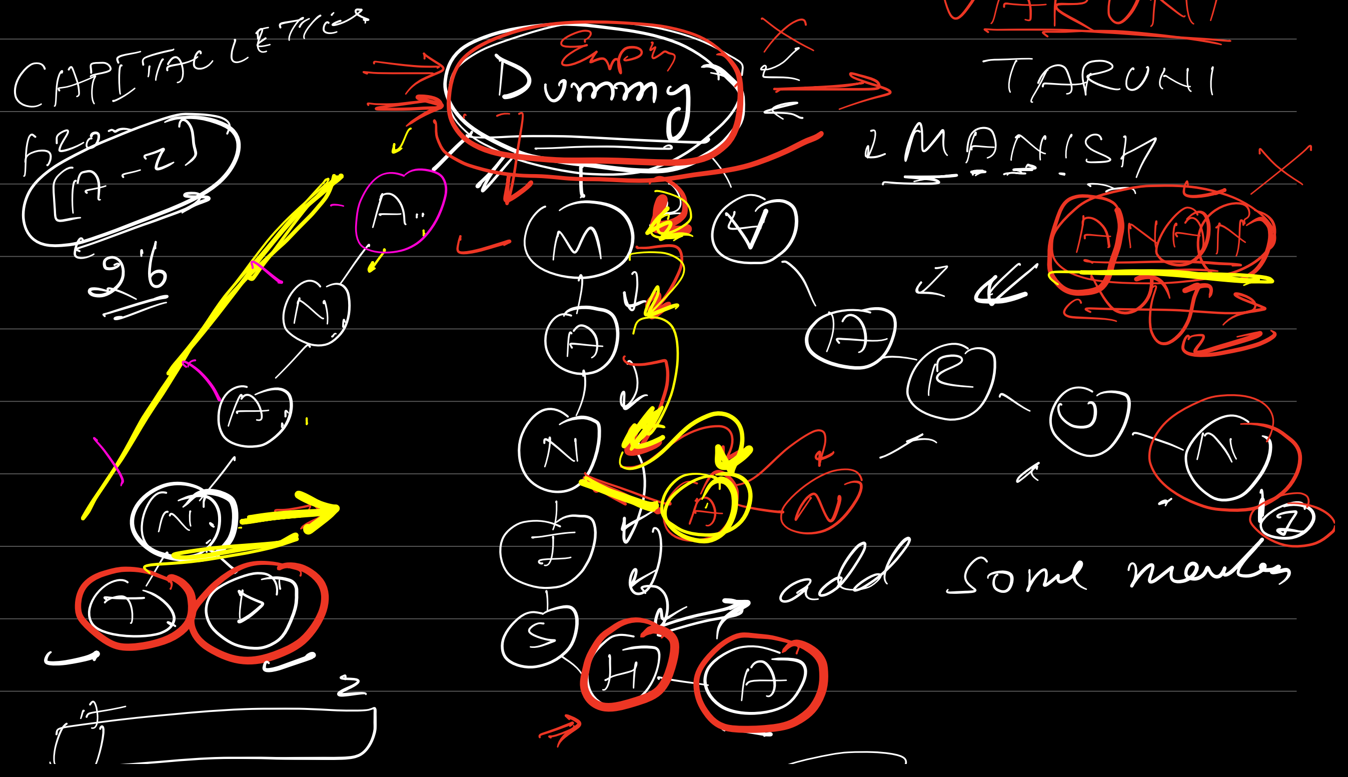
Anant
Prateek
Shruti
Manisha
Manish
Sam
Sampath

~~Tarun~~
Varun
Anshul
Anand
MANAN
M



Every node stores a character.

VARUNI
TARUNI
MANISH

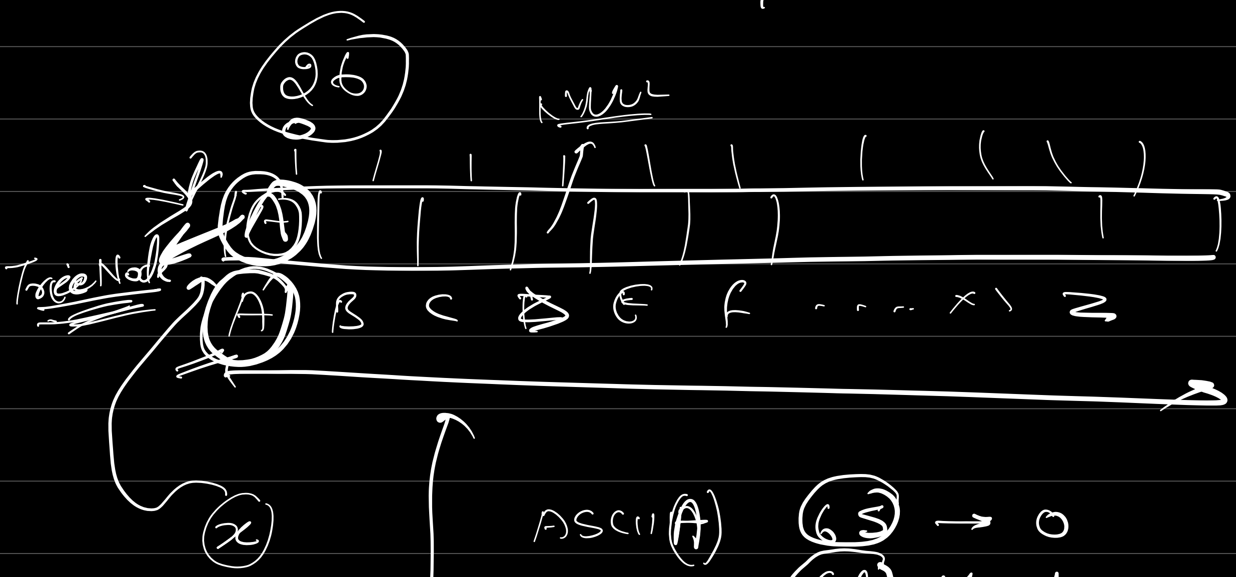


add some marker

(S→A), S→B ... S→Z

Structure of a Trie Node

```
TrieNode {
    char c
    bool isEnd;
    hash array of size 26
```



ASCII(A)	65 →	0
B	66 -65	1
C	67-65 →	2
D	68-65 →	3
E	69-65 →	4

ASCII char − ASCII of 'A'

O(26) for every node

(2)

```
Map < Key       Value >
       ⇓          ⇓
      char      TrieNode
```

Structure

```
TrieNode {
    char c;
    bool isEnd;
    Map<char, TrieNode> children;
}
```

Search

```
                                 ≠ φ
search (root, str) {
  if (root == null) { return false;

  for (c : str) {
      if(!root.children.contains(c)){
          return false;
      }
      root = root.children[c];
  }
  if (root.isEnd == True) {
```

```
            return true;
        }
        return false;
    }
```

L = Length of word to find.

T.C. = $O(L)$

Q⇒ Insert ??

```
Insert ( root , str) {

    for ( c : str) {
        if (root.children.contains(c)){
            root = root.children[c]
            continue;
        }
        Trienode temp = new TrieNode()
        temp.char = c;
        root.children.add(c, temp);
        root = temp;
    }
    root.isEnd = true;
}
```

$\underline{O(N)}$

T.C. = O(N)

1) Node ⇒ c —

2) Node ⇒ bit —

Q Given a set of words find how many words match a given prefix

ayan, ayush, ayushi, ayushman

tarun, shounak, aryan

Pref: ayush ⇒ 3

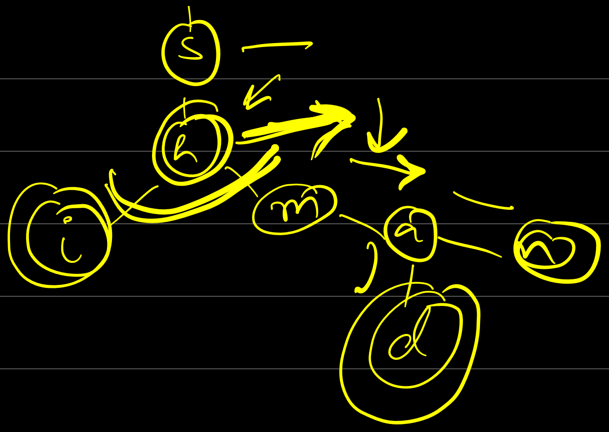
L × |N|

a ⇒ 5

Solⁿ (1) Brute force

ayush

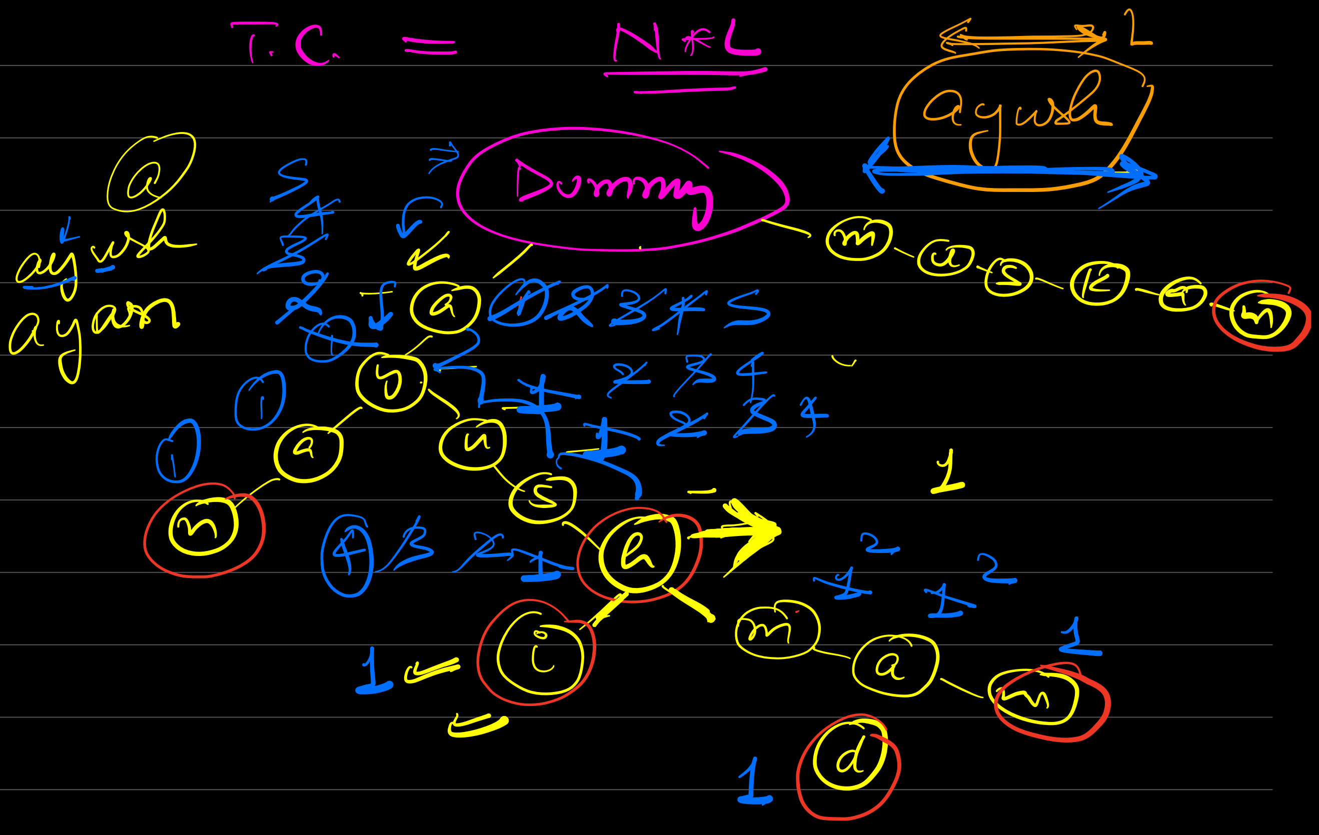
ayushmad



# words = N

size of prefix = L

T.C. = $\underline{N * L}$



$O\left(\sum \text{length of all words}\right)$

```
TrieNode {
    char c
    boolean isEnd
    Map <          >
    int counter = 1 ;
}
```

T.C. = (Create Trie + Search for prefix)

Create Trie ⇓ Σ length of all words

Search for prefix ⇓ O(L)

Q queries

where $Q >>>> (\Sigma$ length of all words$)$

$L \times Q$

Q Given a set of words.

How many words contains a given suffix ??

Q queries

Tarun

run

Varun

Arun

Tarun

Soln Store words in a reverse order then apply the concept of prev problem

prefix = reverse of suffix

Q Given an array of words.

A = ["dog", "zebra", "duck", "dawn"]

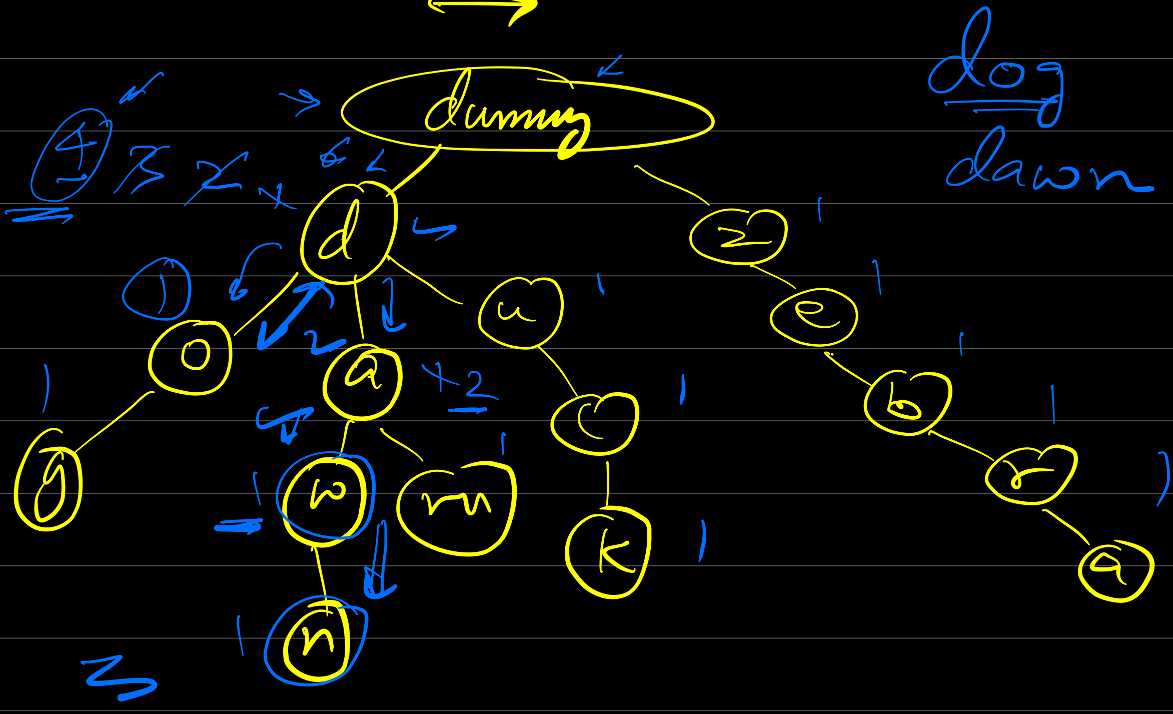
"dam"

Replace every word with the smallest unique prefix.

dog → do

duck → du

dawn → ~~da~~ daw



Q