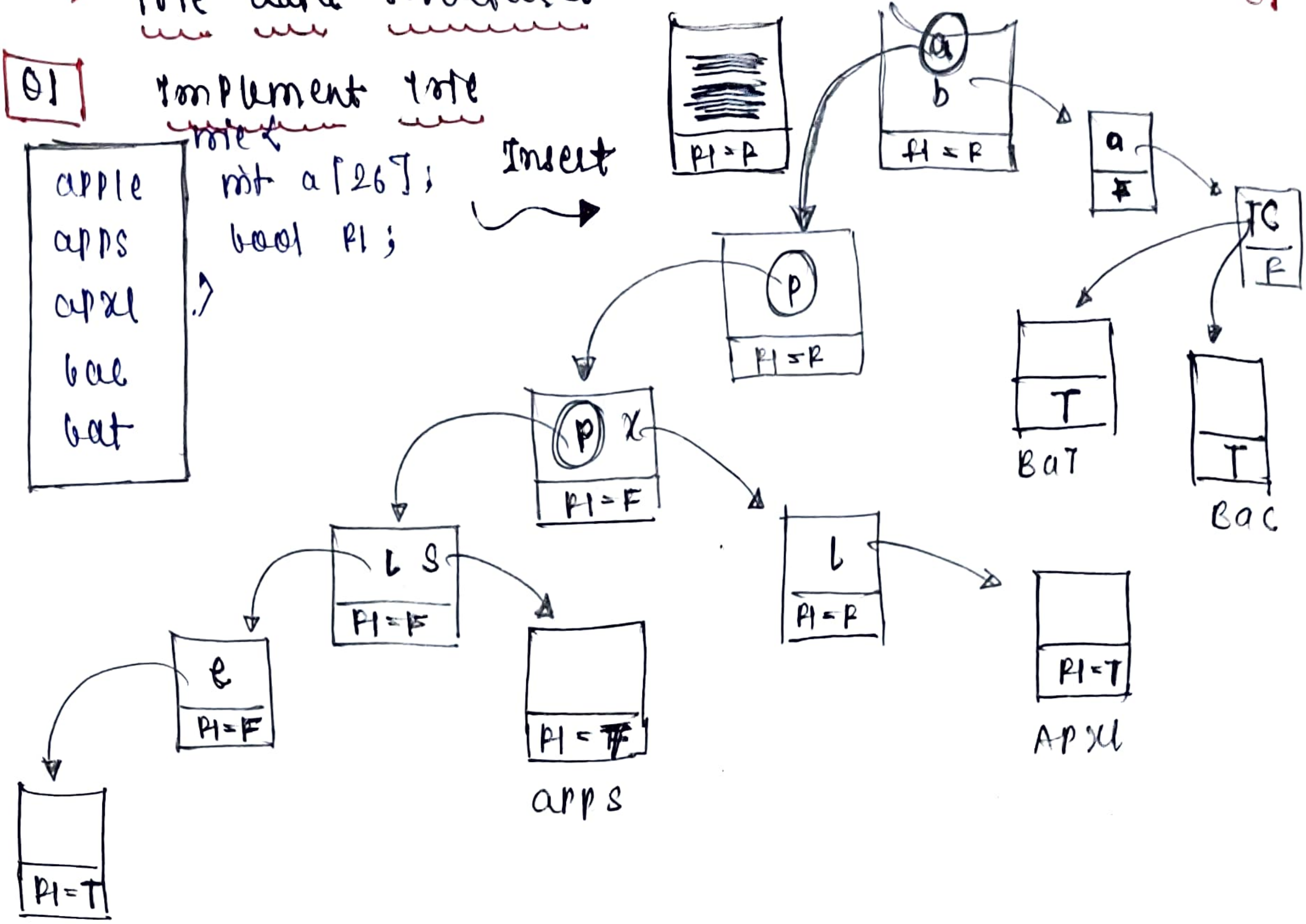
→ Trie data structures.

01 Implement trie

apple
apps
apxl
bac
bat

```
trie {
  int a[26];
  bool fl;
```

Insert →



- **Search** → go to root and move according to reference and search which word you want.
- **Starts with** → Stand at root check and reference with more if not null pos then it is true.

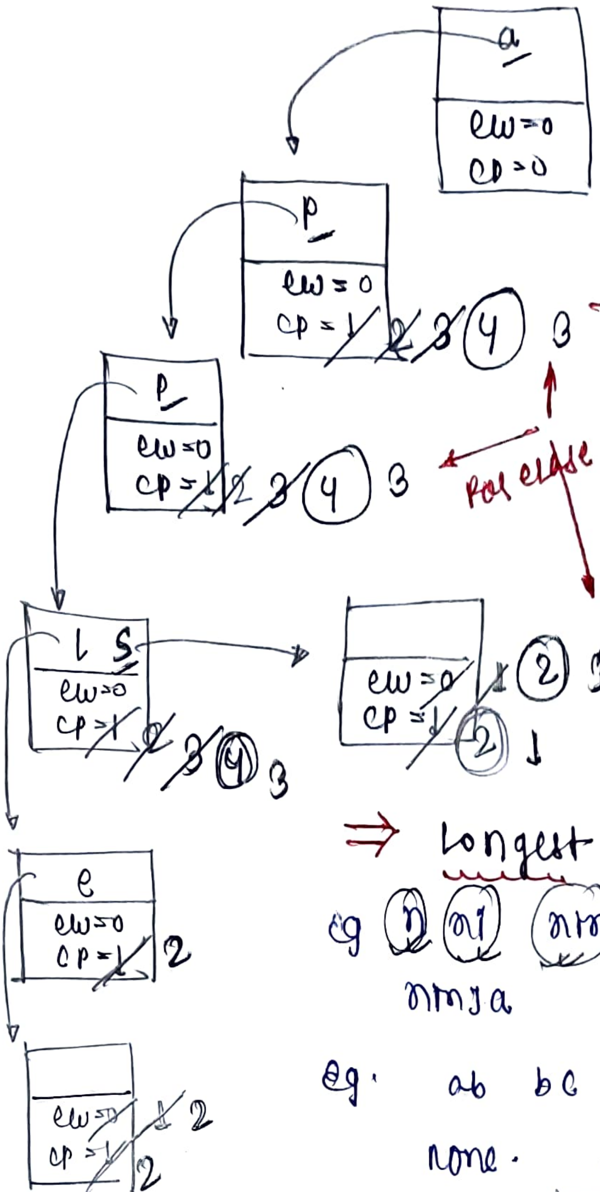
  Eg ba → yes

  Eg apd → no.

02 Implement trie II

apple
apple
apps
apps.

```
trie {
    cnt [26];
    ew = 0;
}   cp = 0;
```



→ countwordsEnd ()

- apps = 2
- abc = 0

→ Inserted

apple
apple
apps
apps

→ countwords start with ()

- ap → 4
- apple → 2

→ Erase ()

apps

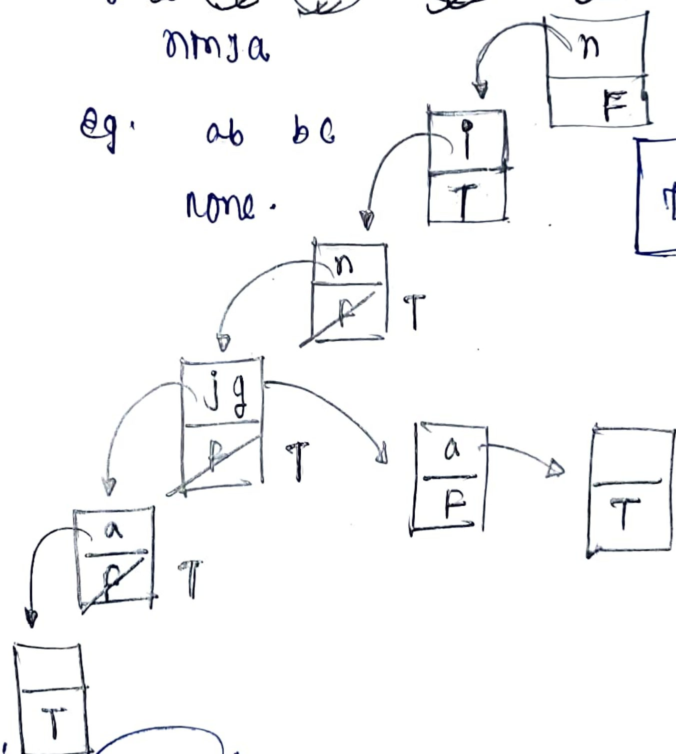
⇒ Longest word with all prefixes

eg n ni nin ninj ninja   ninga

ninja

eg. ab bc

none.



TC = O(N) x O(len)

ni
ninja
ninj
ni
nin
ninga

Trie {
arr[26];
bool flag;
}

ninja → longest string.

[04] count distinct substrings

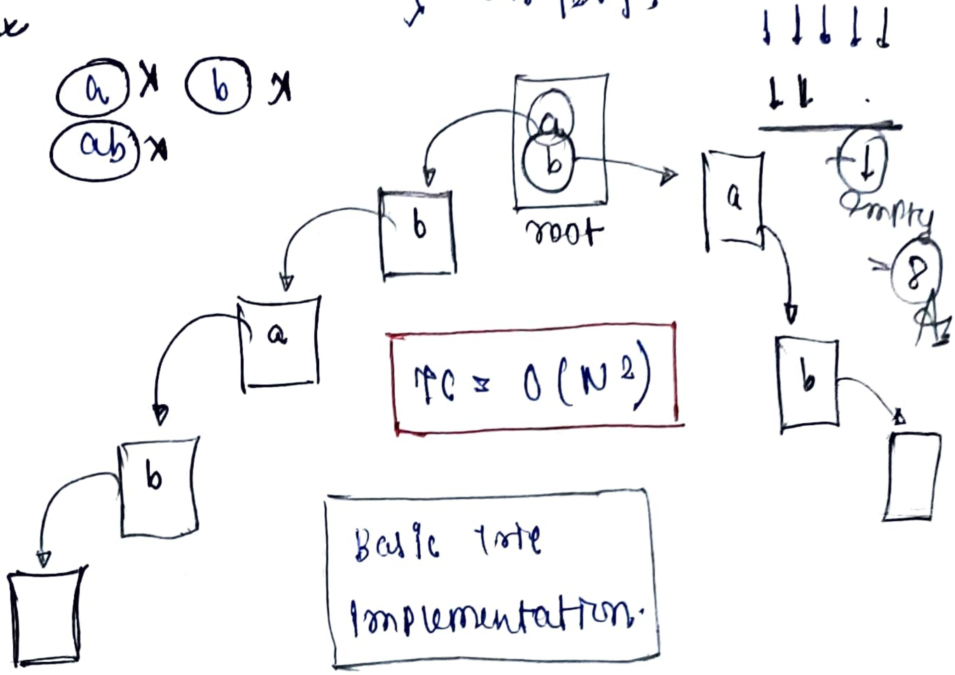
Trie {
} arr[26];

Eg. abab

- a ✓
- ab ✓
- aba
- abab
- b ✓
- ba
- bab
- a ✗
- ab ✗
- b ✗

7 + (1) = 8

empty substrings



TC = O(N2)

Basic Trie Implementation.

[05] Bit prerequisites for Tries.

XOR → different bit XOR = 1
similar bit XOR = 0.

even no of 1's → 0
odd no of 1's → 1

9 → 000.....0001001
32 bits / 64 bits

→ check if bit is set or not

9 → 1001 (3rd bit is set or not)

so, num >> (3) ;

so,
  000..01001
& 000..00001 → num >> 3

  00..00001

if ans be 1 then set
else not set.

→ How do you turn on bit.

9 → 1001 (2 1 0)

1 << 2.

now
   1001
OR 0100

   1101

1 → 10 → 100 (1 << 2)

num | (1<<2).

06 Maximum XOR of two numbers in array

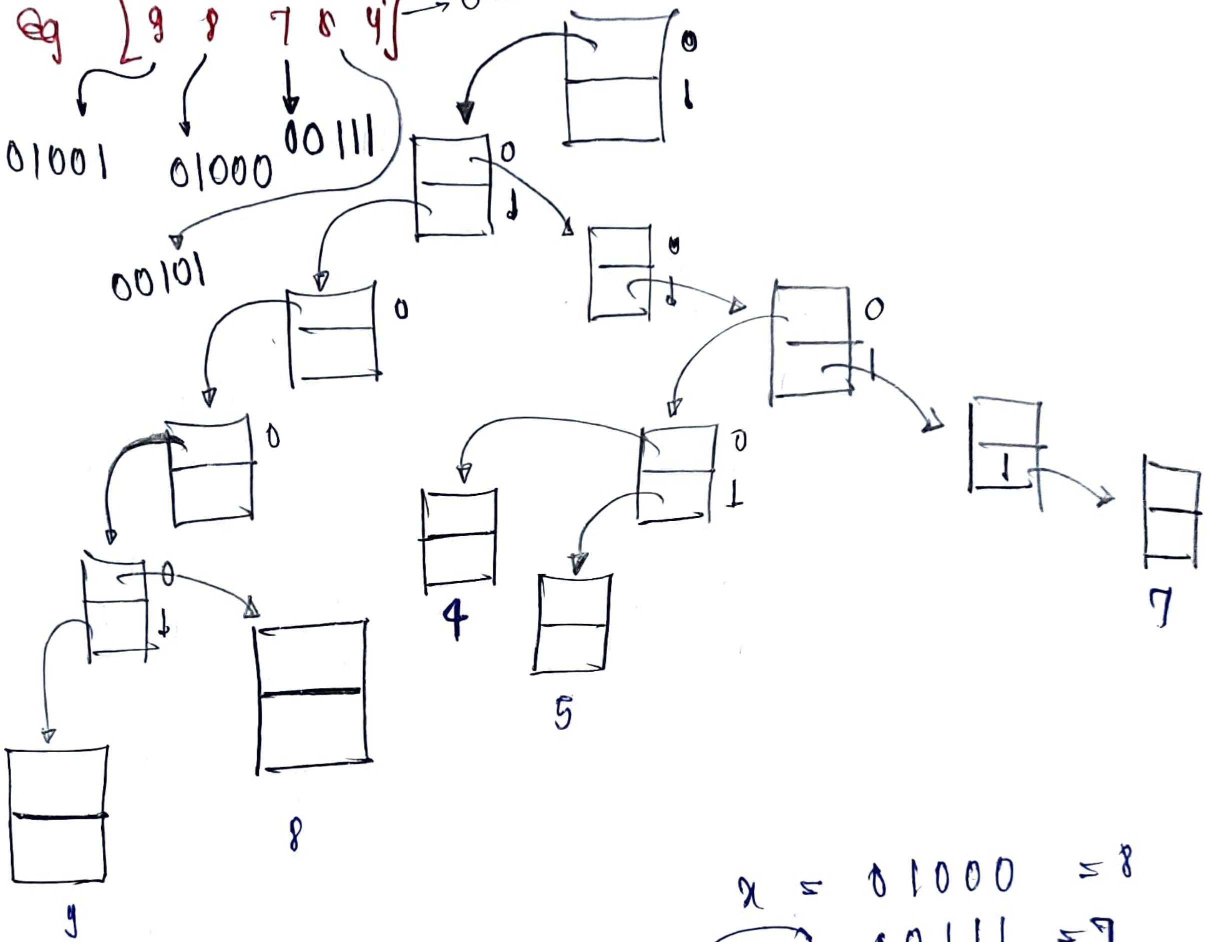
max^m XOR value of (arr[i] ^ x)

→ Insert all nos in trie. (binary bits)

→ Take x and find the max no. from array where XOR(arr[i] ^ x) ↑↑ → node { links[2] }

Now 9 → 000..001001 (32 bits) (01001) 5 bits

eg [9 8 7 5 4] → 00100



x = 8 ^ arr[i].

x = 01000 = 8

XOR 00111 = 7

01111 = 15 Ans

yse wale me max^m | change.

TC = O(N×32) + O(M×32)