TRIES

Motivation:

T.C to compare 2 strings?

l_1 l_2 are the lengths: $O(\min(l_1, l_2))$

d f i d l_1
a b c d
a b c d e l_2
↑ ↑ ↑

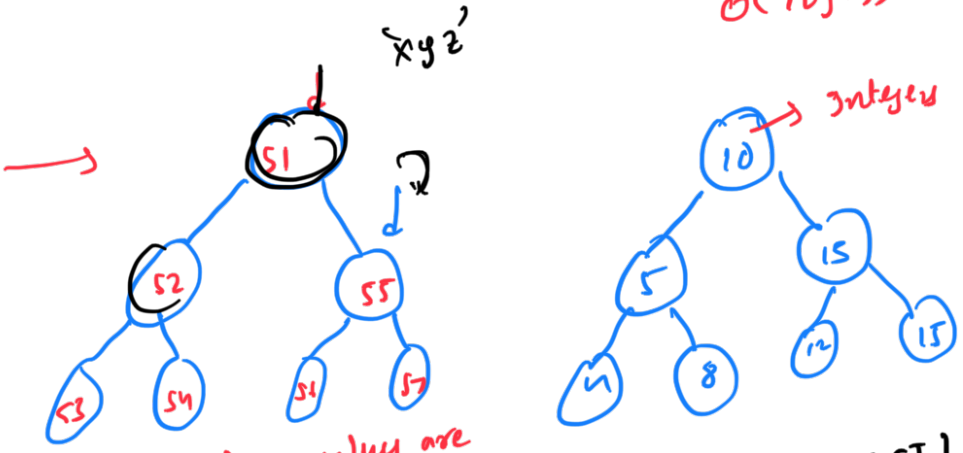
Both the strings have same length: $O(l)$

T.C for insertion, deletion & searching in B-BST?

$O(\log n)$

$H = \log n$

Height $\log H \Rightarrow H$



BBST where values are strings

ab
xyz

O(1) ⇓ O(1)

T.C: $O(l \times \log n)$ [BBST]
$O(l \times h)$ [BST]

Assume 'l' is the length of the longest string

T.C of Insertion, Search, Deletion when keys are strings is $O(l \times \log n)$

Tries does all these operations in? : $O(l)$

→ 'l': Maximum length of any string in our dictionary

space

→ Compromise a little on

Hash table (unordered Map) ⇒ O(1)

T.C for insertion, Deletion & search: O(1) Amortized

Key: strings

unordered_map <string, int>

x % 10 = 3 ⇒ O(1)

string ⇒ a b c d → O(1)

T.C: O(l)

T.C for hash table of strings: O(l) Amortized

1) Print all the words in Alphabetical :
2) Prefix matching
3) Auto complete.

Tries, O(l) is the upperbound ⇒ Tries

O(l) is amortized ⇒ Hashmaps.

Applications

1) Auto Complete (Google)
2) Prefix Matching
3) Contacts

Codeforces string matching Algos

TRIE :

Helps in Optimal Retrieval and storing

Lower-case vector< >

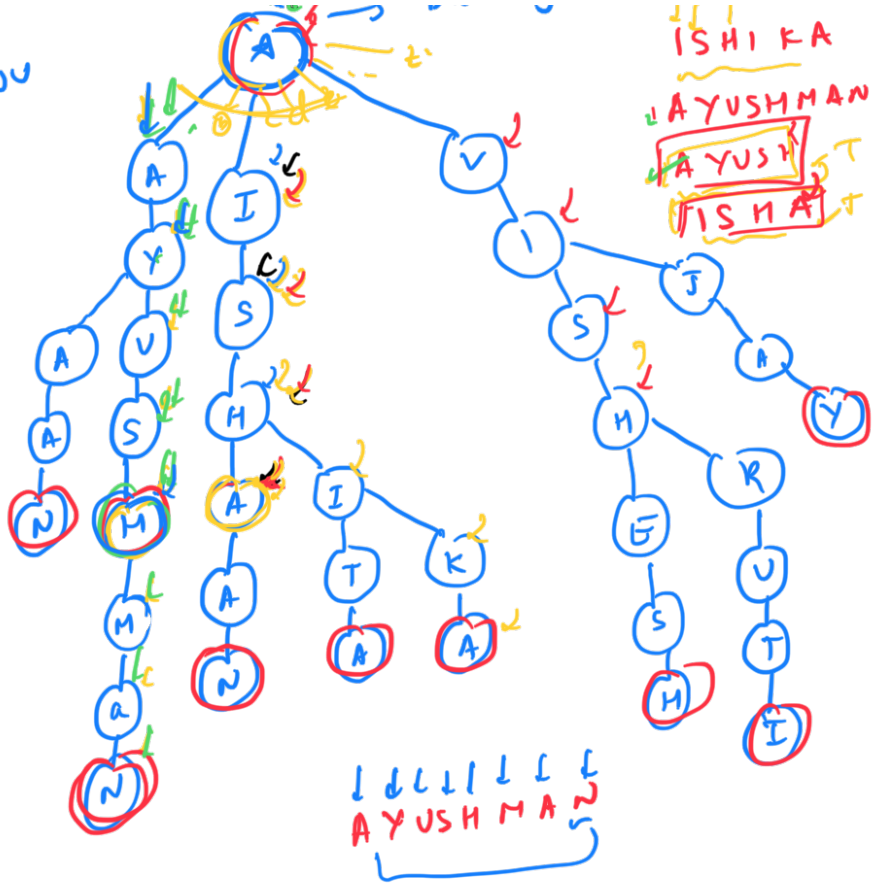
RETRIEVAL

AYUSH

Dummy Node.

Ishaan
Ishita
Ayush
Ayaan
Ayushman
Ishika
Vishesh
Vishruti
Vijay

VISHNU



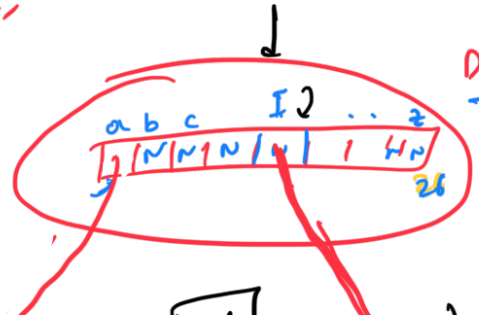
AYUSHMAN

Search
1) Ishika
2) Ayushman, Ayush
3) Isha

Trie:
1) Dummy Node.
2) p1, p2, p3 . . . . . p26
Array of Size 26
Node

class TrieNode {
char data;
TrieNode children[26];
bool endOfWord;
}

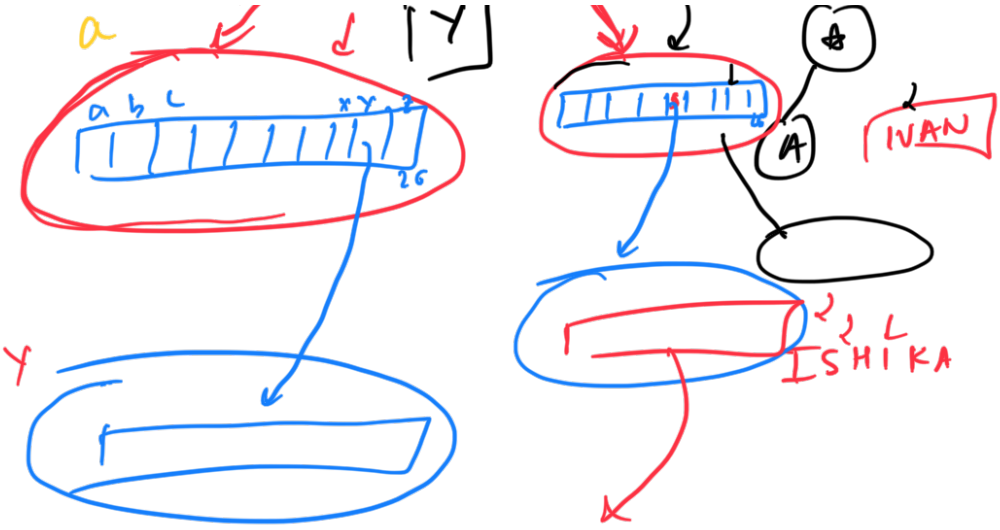
[Not Required]

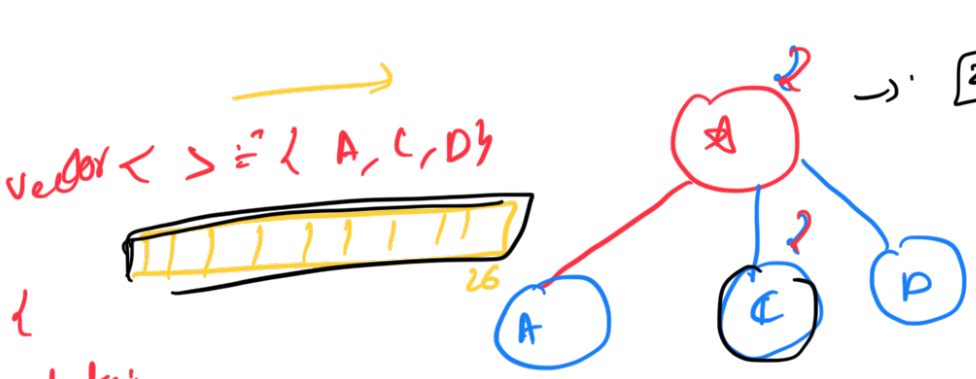


b → 1

c → 2

z → 25





X

'a' = 97 — 97

'b' : 98 — 97

a — 0

b — 1

z — 25

ch - 'a'

D => 100

100 - 'a'

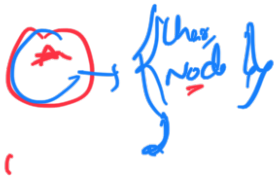
100 - 97

= 3

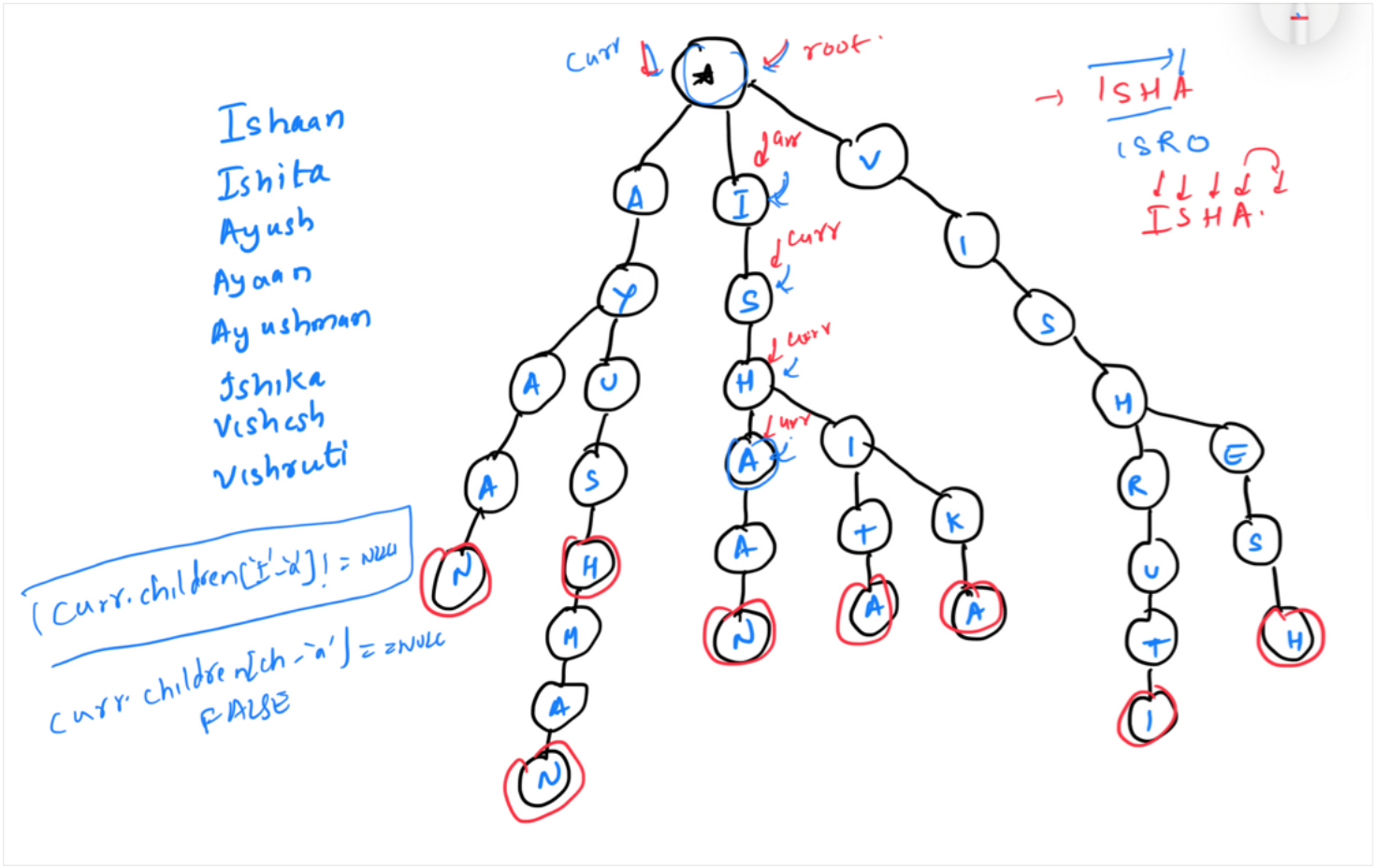
```
Node {
  char data;
  vector < Node >
}
```

```
TrieNode setNode {
    node = new TrieNode();
    for(i=0; i<26; i++) {
        node.children[i] = NULL;
    }
    node.isEnd = false;
}
```

HashMap < char, Node >



Searching:

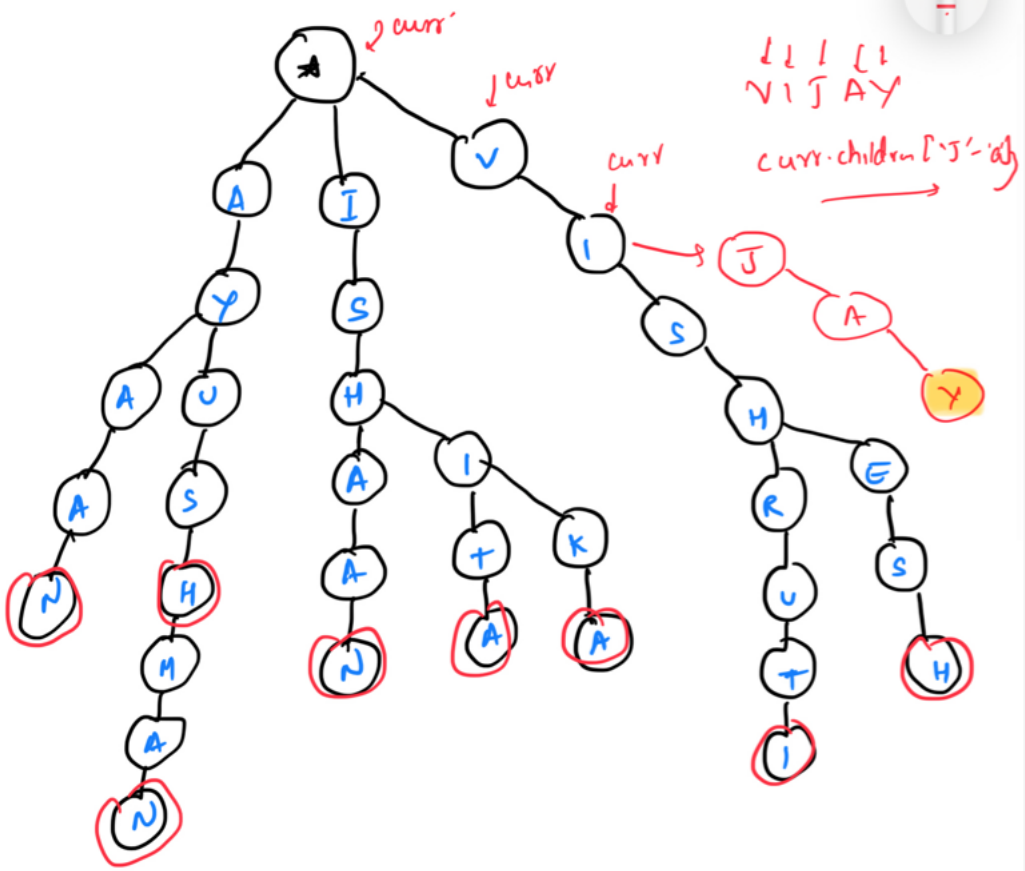

Ishaan
Ishita
Ayush
Ayaan
Ayushman
Ishika
Vishesh
Vishruti
curr
root
curr
curr
curr
curr
ISHA
ISRO
ISHA.
(curr.children['i'-'a'] != NULL
curr.children['h'-'a'] = = NULL
FALSE

```
bool search(TrieNode root, string s){
    curr = root;
    for(i=0; i<s.length; i++){
        if(curr.children[s[i]-'a'] == NULL)
            return false;
        curr = curr.children[s[i]-'a'];
    }
    if(curr.isEnd == true) return true;
    return false;
}
```

T.C: O(L)

Insertion:

Ishaan
Ishita
Ayush
Ayaan
Ayushman
Ishika
Vishesh
Vishruti

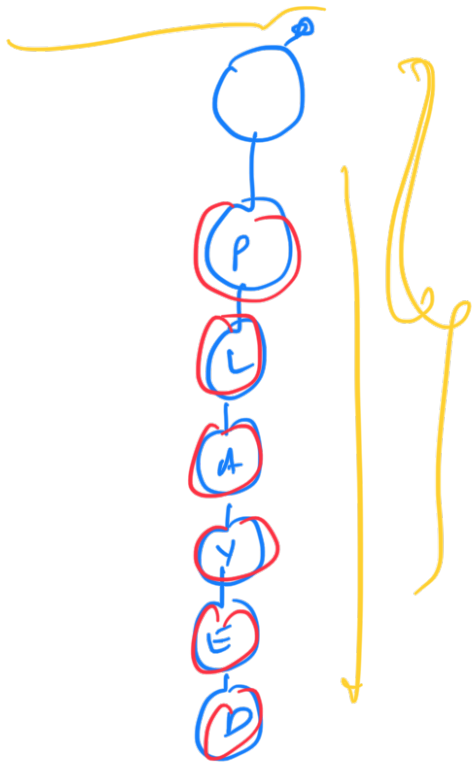


T.C: O(L)

Space Complexity

[P, Pl, Pla, Play, Playe, Played] => N words

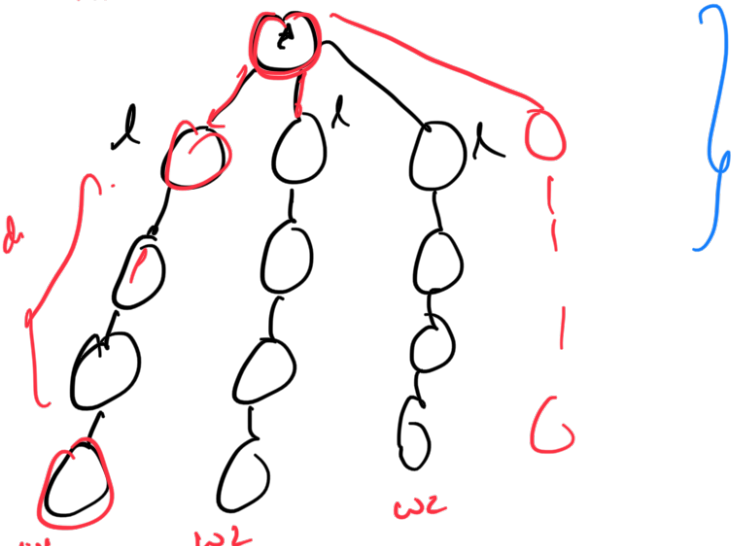
=> Every word => length N



6 Node

[ab, bc, cd, de. . . .]

-) No node in the Trie has a overlap



(N*L) words

T.C: O(N.L.26)

O(N·

N words

B.B.ST: O(n×l)

Question: No. of words with a given prefix

k k

words = [play, played, playing, args, arrange, arrow]

k

Len(pla) = k

prefix = "pla" => 3

"arg" => 2

"ar" => 3

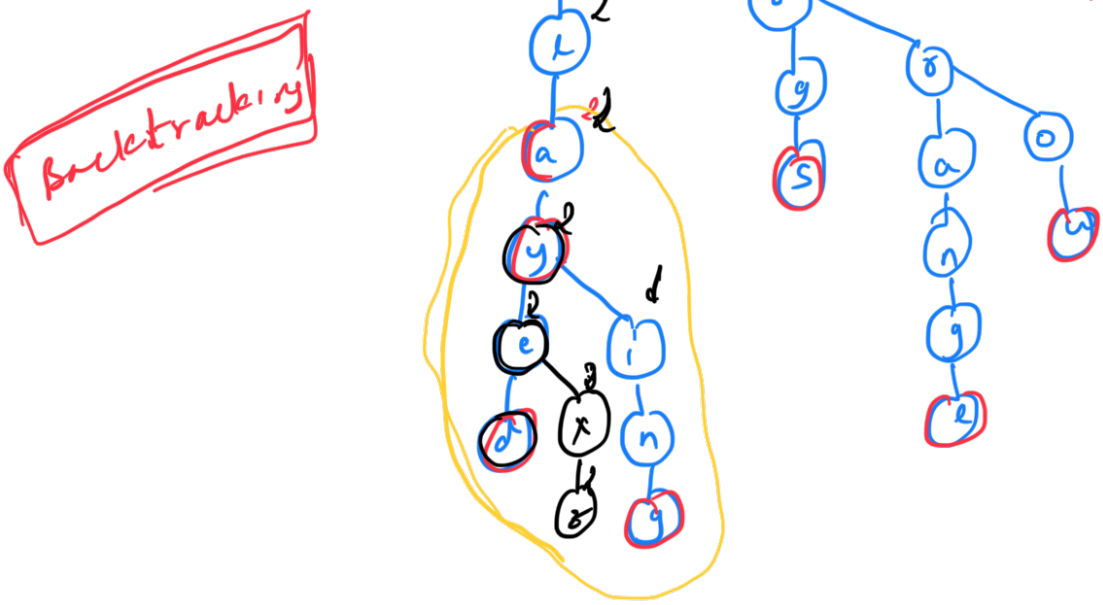
prefix

Trie

play

pla = 3

ans = 1+1



Brute force:

=> N words

=> l : length of biggest

=> K: length of prefix

T-C. $O(n \cdot k)$ ⇒ Worst Case: $O(n^2)$

Approach:

Approach 2: $O(n \cdot l) + O(n \cdot l)$ (query: $O(n \cdot l)$)

↑ query: $O(k)$: $O(1)$ (query: $O(q \cdot n \cdot l)$)

$O(q \cdot l)$

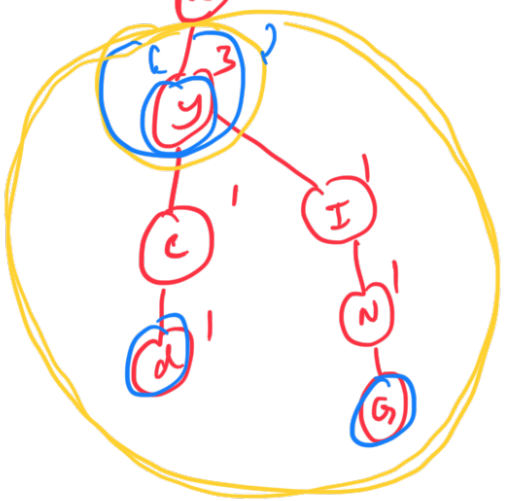
play, played, playing args, arrange, arrow

prefix = a

T-C: $O(n \cdot l) + k$

```
for(i=0; i<26; i++){
  if(child[i] != null){
  }
}
```

play



prefix = "pl" → Play $O(n \cdot l)$

arr

$O(1)$?

Step1: Construct Trie: $O(n \times l)$

Step2: Iterate the given prefix: k

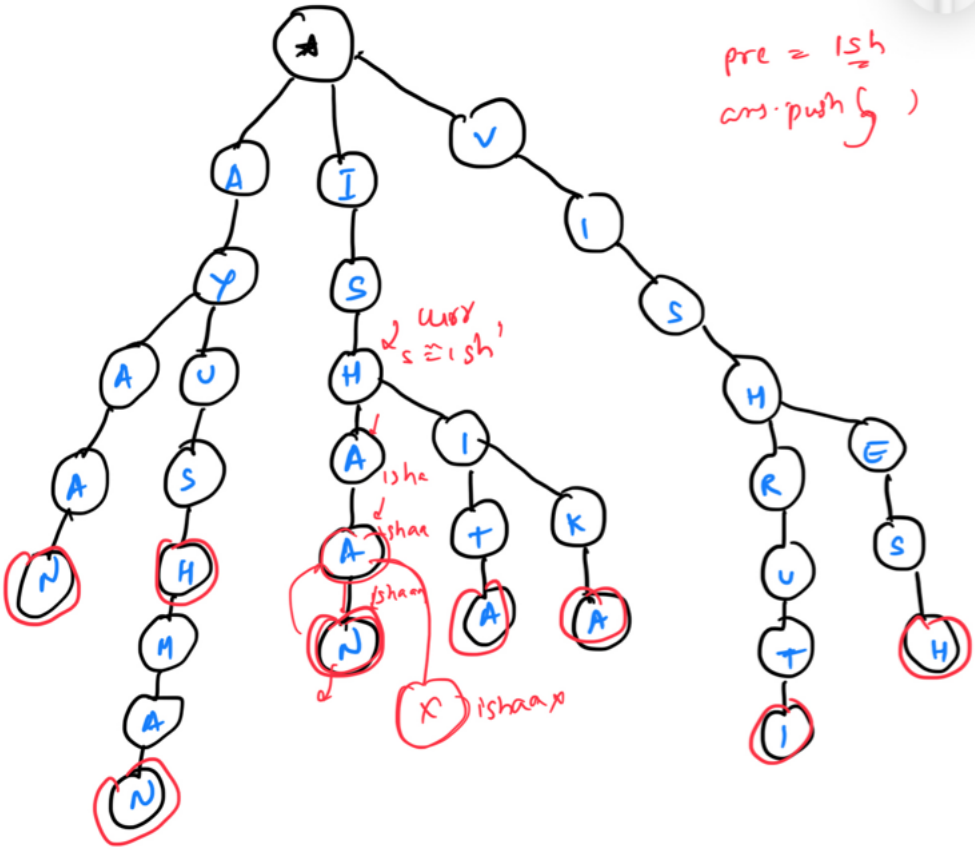
$O(n \cdot l + k)$

all the words with that prefix

Question: Print

Back Tracking

Ishaan
Ishita
Ayush
Ayaan
Ayushman
Ishika
Vishesh
Vishruti



String.prototype

Question: Given an array of words with suffixes. Tell no of words with ith suffix.

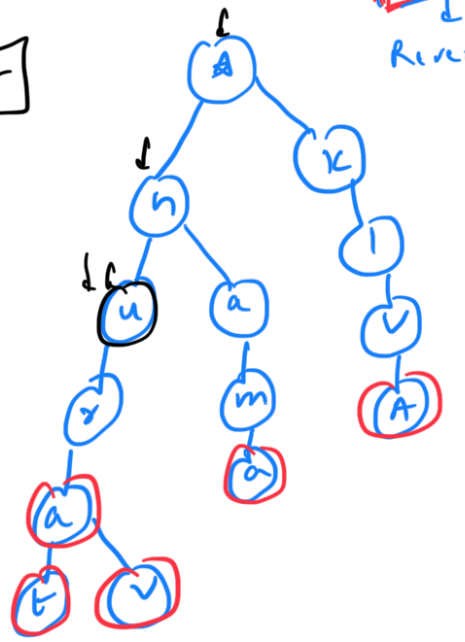
[tarun, varun, arun, avik, aman] for(i = s.length-)

tarun =
Prefix: t, ta, tar, taru, tarun
Suffix: n, un, run, arun, tarun

N(un) = 3
N(n) = 4
N(run) = 3

Count

un
Reverse: nu



1) Construct a Trie by reversing the words
2) Reverse the search suffix and

Question: Smallest unique prefix for every word

words = [dog, zebra, duck, pan, paddy]
d o
z
d u
pan
pad

prefix(dog) = d, do, dog

Ex dog doggy

Not given these types of Input

dog

Dog

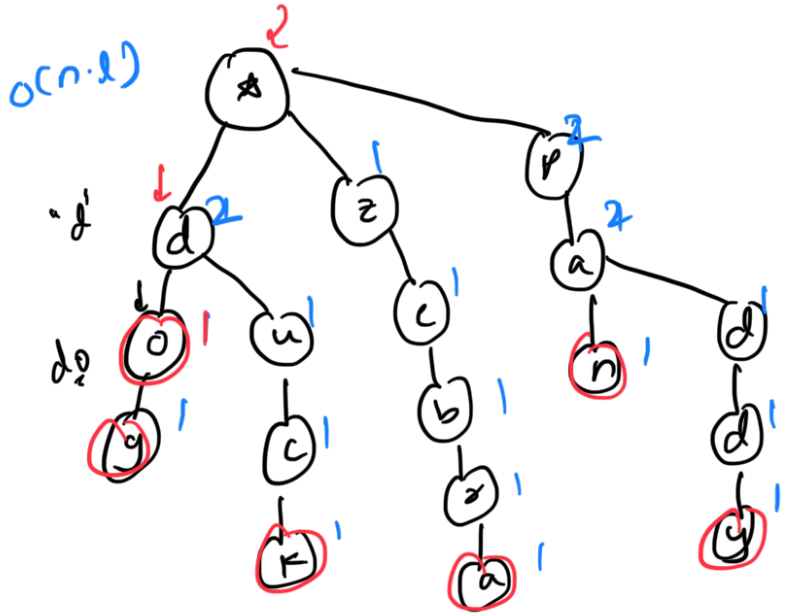
Step1: Insert in Trie ⇒ O(n·l)

Step2: For every word, iterate until you get a count

O(n × l)

T.C: O(n·l)

S.C: O(n·l)



S.C

Question: Pair with maximum XOR

A: [1, 5, 4, 3, 2]

0 1 2 3 4

001 101 100 011 010

$1 \wedge 1 = 0$

$0 \wedge 0 = 0$

$1 \wedge 0 = 1$

$0 \wedge 1 = 1$

(5, 2)

101
010

111

⇒ 7

(4, 3)

100
011

111

⇒ 7

100000

01111

14, 15, 16, 17, 18

Brute Force:

Consider all pairs

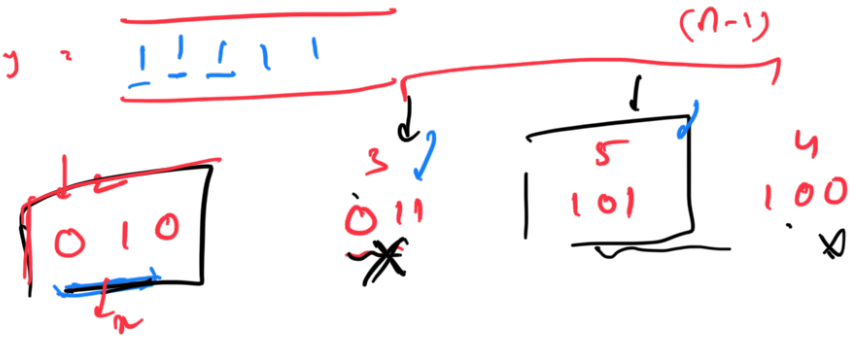
T.C: $O(n^2)$

Approach 2:

x = 101101

y = 010010

x^y = 111111



x = 010

y = 101

x^y = 111

T.C for 1 iteration.

= 10

32 bits

for (i = 32; i ≥ 1; i--) {

// flip all words

T.C: O(n^2 x log(max(A[i])))

T.C: O(n * 32)

= [O(n x log(max(A[i])))]

010, 011, 101, 100

n x

101 | 32 bits

x =

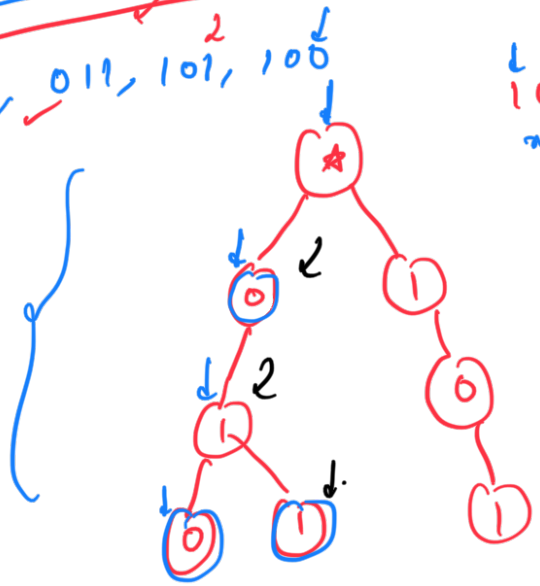
x = 111,

x = 11001111

00111000

111

100

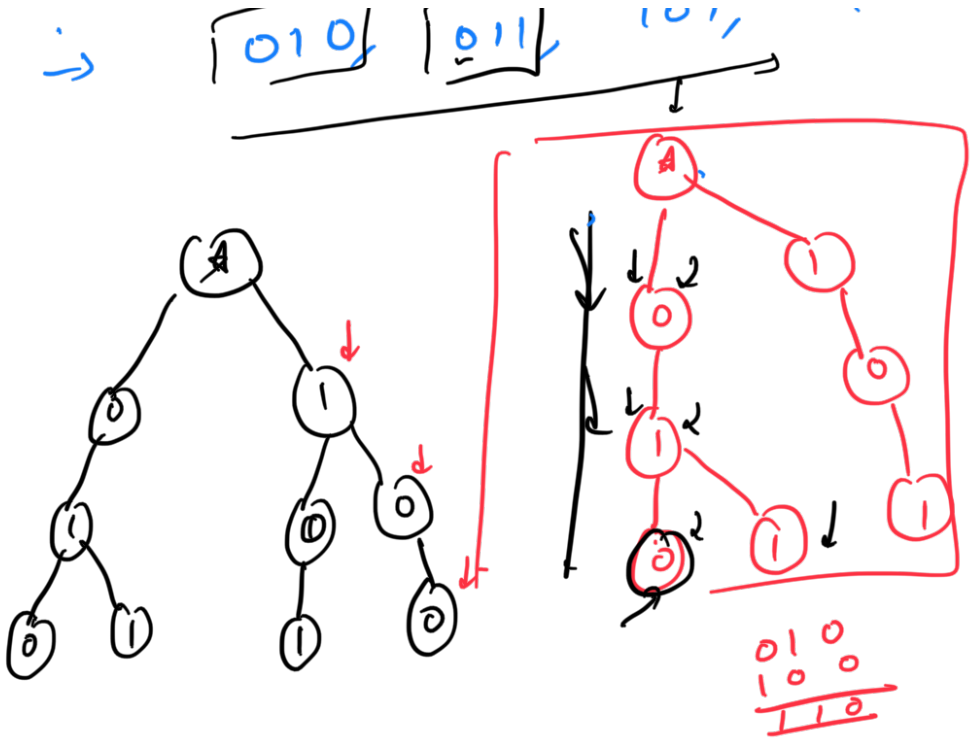


101 ^ 010

= 111 = T

ans = max(ans, res)

100 ⇒ 011 = S



ans = 7

$0\,1\,1$
$0\,1\,0$
$0\,0\,1$

$101 \wedge 010 = 7$

$100 \wedge 011 = 7$

$0\,1\,0$
$1\,0\,0$
$1\,1\,0$

T.C : $O(n \times 32) + O(n \times 32) \Rightarrow O(n)$

S.C : $O(n \times 1) = O(n)$

Question : Subarray with maximum XOR.

A = 1 4 3

[1] [4] [3] [1,4] [4,3] [1,4,3]

→ Subarray [l, r] ⇒ Prefix XOR $a \wedge b \wedge c$

$(0, a \wedge b \wedge c)$

$x \wedge y = a \wedge b \wedge c$

Pref = 0 a $a \wedge b$ $a \wedge b \wedge c$ $a \wedge b \wedge c \wedge d$ $a \wedge b \wedge c \wedge d \wedge e$

Prefix 0 a

abcde

(l, r) ⇒ O(1) ⇒

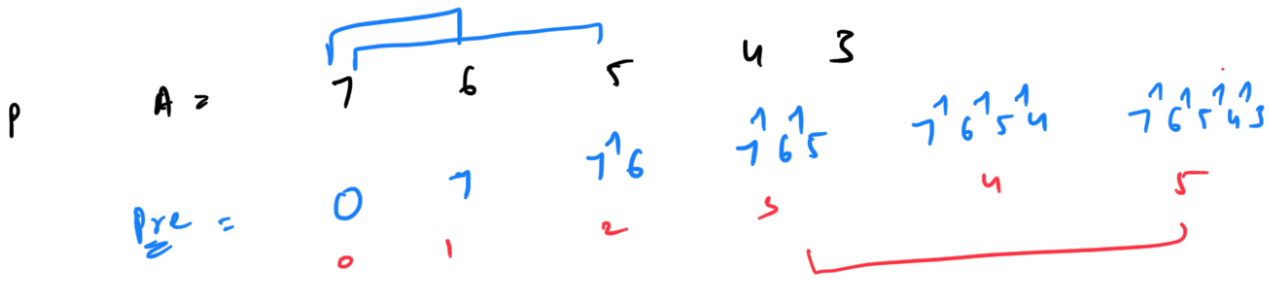
$(a \wedge b \wedge c \wedge d) \wedge a = b \wedge c \wedge d$

Find number in prefix xor array with

→ Find ... maximum XOR [7]

Edge

a = 111



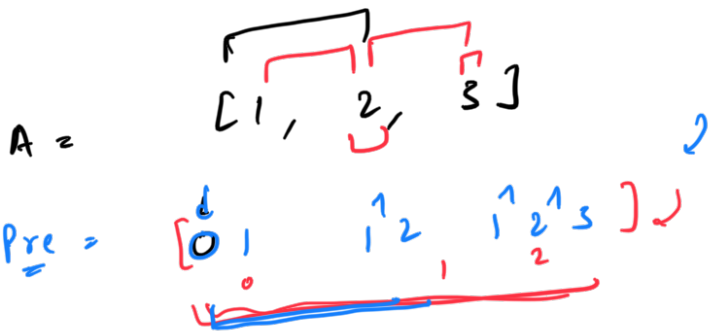
(0, 1) ⇒ [l+1, r)

l r
(2, 5)

7^6 ↓ 7^6^5^4^3

= 5^4^3

[7]



[1,2] as 1^2

(0)^(1^2) = [1^2]

x^0 = x

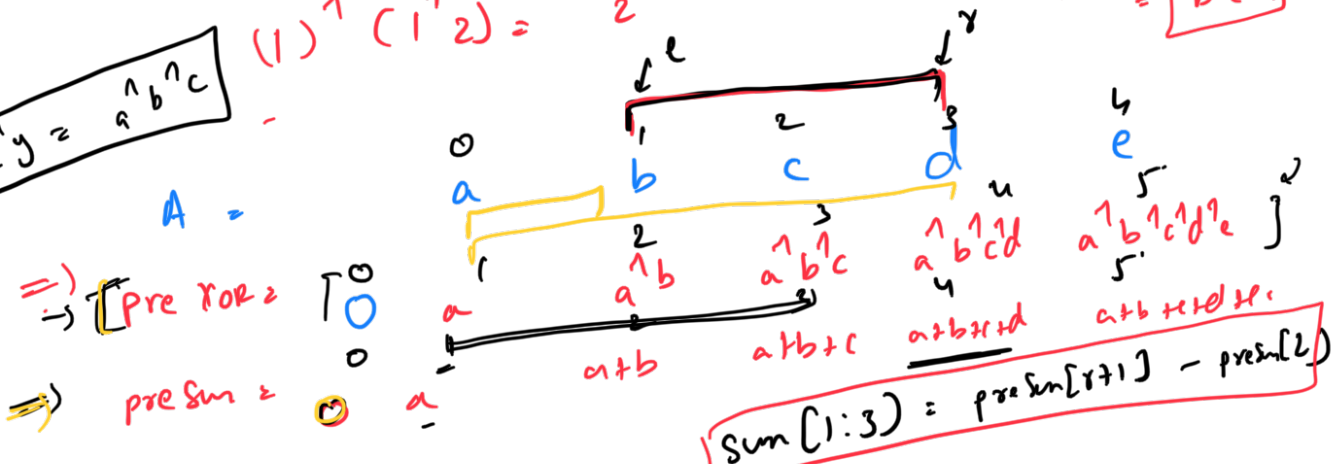
(1)^(1^2^3) = 2^3

(1^2)^(1^2^3) = 3

(1)^(1^2) = 2

x^y = a^b^c

A^b^c^d^d = b^c^d



⇒ [pre XOR] [0 a ...]

⇒ preSum = [0 a ...]

sum[1:3] = preSum[r+1] − preSum[l]

XOR[l..r] = [a+b+c+d − a]

preSum[r] − preSum[l−1]

preXOR[r] − preXOR[l−1]

for(i=1; i<=r; i++)
ans = ans ^ a[i]

sum(l .... R) =

sum(0 .... r) = pre[r] - pre[l-1]

pre[-1] = 0

if(l == 0)
return presum[r]

A = a b c d e

	0	1	2	3	4
	a	b	c	d	e
PreXOR =	0	a	a^b	a^b^c	a^b^c^d	a^b^c^d^e
	0	1	2	3	4	5

L=0 R=0 2

→ Find Subarray with XOR

→

[0 ... 5]

[l, r] => pre[r] - pre[l-1]

[1, 2]
[l, R-1]

(a) (a^b)
(a^b^c)^a = b^c

[l, r] = pre[r] - pre[l-1]

If [L, R] are indexes of pref XOR => [L, R-1]

(L ≤ R)

[0, N-1]

[0 ≤ L ≤ N-1]

1 ≤ R ≤ N

1-1 ≤ R-1 ≤ N-1

0 ≤ R-1 ≤ N-1

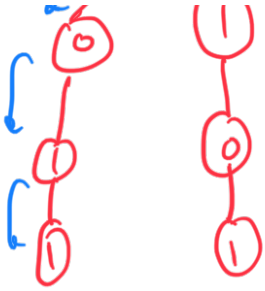
[0...N-1]

P1 P2

[0...N-1]

[0...N-1].

011 101 100



Create a trie node

```
trieNode *getNode(){
      trieNode* newNode = new trieNode;
      for(int i = 0 ; i < 26; i ++){
            newNode->children[i] = NULL;
      }
      newNode->isEnd = false;
      newNode->count = 0;
      return newNode;
}
```

Searching in a trie

```
bool search(trieNode* root, int str){
      trieNode* temp = root;
      for(int i = 0 ; i < str.length(); i++){
            if(temp->children[str[i] - 'a' ] == NULL){
                  return false;
            }
            temp = temp->children[str[i] - 'a'];
      }
      if(temp->isEnd) return true;
      return false;
}
```

Inserting in a Trie

```
void insert(trieNode* root, int str){
      trieNode* temp = root;
      for(int i = 0 ; i < str.length(); i++){
            if(temp->children[str[i] - 'a' ] == NULL){
                  trieNode* newNode = getNode();
                  temp->children[str[i] - 'a'] = newNode;
            }
            temp = temp->children[str[i] - 'a'];
            temp->count++;
      }
      temp->isEnd = true;
}
```

Pair with Maximum XOR

```
bool ithBit(int x, int i){
      if(x && (1 << i)) return 1;
      return false;
}
void insert(trieNode* root, int x){
      temp = root;
      for(int i = 31; i>=0; i--){
            bit = ithBit(x, i);
            if(temp->children[bit] == NULL)
                  temp->children[bit] = new trieNode();

            temp = temp->children[bit];
      }
      temp->val = x;
}
```

```
void findXOR(trieNode* root, int x){
      temp = root;
      for(int i = 31; i>=0; i--){
            bit = isIthBitSet(x, i);
            if(temp->children[!bit] != NULL)
                  temp = temp->children[!bit];
            else
                  temp = temp->children[bit];
      }
      ans = x^ temp->val;
      return ans;
}

int main(int arr[], int n){
      root = new Node();
      insert(root, arr[0]);
      maxXor = 0;
      for(int i = 1; i<n;i++){
            a = findXor(root, A[i]);
            maxXor = max(maxXor, a);
            insert(root, A[i]);
      }
      return ans;
}
```