

02/06/2023

Q1 Longest common prefix

i/p \rightarrow [flower, flow, flight]O/p \rightarrow fl

Optimal solution does not come from tries. Optimal solution can come from brute force.

flower

 \rightarrow Prefix (Reading from left to right)

Approach-1

Let's assume we have 4 strings code, coder, coding, codehelp. Pick one string.

	c	o	d	e
code				
coder	✓	✓	✓	✓
coding	✓	✓	✓	x
codehelp	✓	✓	✓	✓

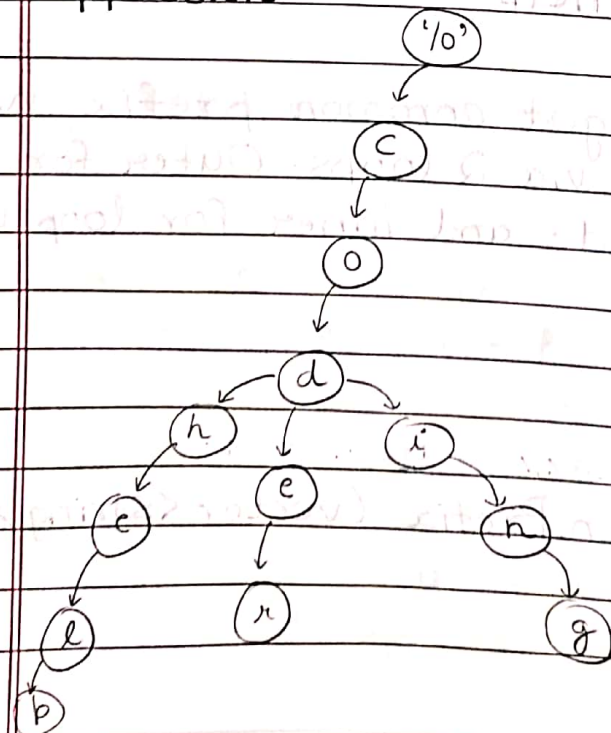
Hence cod is the longest common prefix. We can do this approach via 2 loops. Outer for loop will run on code and inner for loop will run for $n-1$ strings.

Code

```
String longestCommonPrefix (vector<String> &strs)
{
    String ans = " ";
```

```
// loop for 1st string
for (int i = 0 ; i < strs[0].length() ; i++) {
    char ch = strs[0][i];
    bool match = true;
    // compare ch with all string's jth index
    for (int j = 1 ; j < strs.size() ; j++) {
        // invalid                // ↑ not equal
        if (strs[j].size() < i || ch != strs[j][i]) {
            match = false;
            // Hence exit as prefix not common
            break;
        }
        // Simply break as prefix match not true
        if (match == false)
            break;
        else // Insert character in ans
            ans.push_back(ch);
    }
    return ans; // Longest common prefix
}
```

Approach-2



code
 coder
 coding
 codehelp

TimeComplexity : $O(n*m)$
 SpaceComplexity : $O(n*m)$

- 1) Insert all characters of strings in trie.
- 2) Whenever we find that there are multiple child, then stop but if we find that a node is terminal, then stop.

Code

```
void findLCP (String first, String &ans,
TrieNode * root){
```

```
    // Here i can do mistake (handle empty string)
```

```
    if (root → isTerminal)
```

```
        return ;
```

```
    for (int i=0 ; i<first.length(); i++) {
```

```
        char ch = first [i];
```

```
        if (root → childCount == 1) {
```

```
            ans.push-back (ch);
```

```
            // Move to next child
```

```
            int index = ch - 'a';
```

```
            root = root → children [index];
```

```
        }
```

```
    } else { // More than one children
```

```
        break ;
```

```
    } // Terminal found before multiple child
```

```
    if (root → isTerminal)
```

```
        break ;
```

```
    }
```

```
}
```

```
String longestCommonPrefix (vector <string> &strs){
```

```
    TrieNode * root = new TrieNode ('-');
```

```
    // Insert characters in trie
```

```
    for (int i=0 ; i<strs.size(); i++) {
```

```
        insertWord (root, strs[i]);
```

```
    }
```

```

String ans = " ";
// Move forward by using characters of 1st
String first = strs[0];
findLCP (first, ans, root);
return ans;
}

```

Q2 Suggestions when string is written in search bar.

i/p → codef

O/p →

c ⇒ 7 strings

co ⇒ 7 strings

cod ⇒ 7 strings

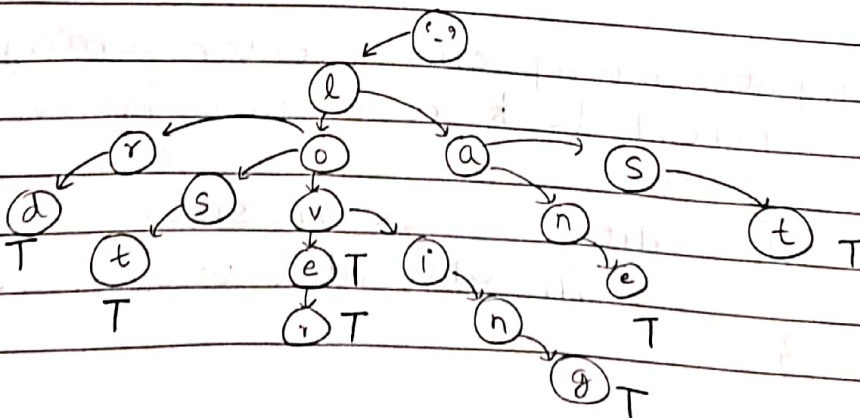
code ⇒ 5 strings

codef ⇒ 1 string

Here we are storing the vector in which some strings have been mentioned

Dry run

vector → [love, loving, lover, lane, last, lost, lord]



i/p \Rightarrow lovi

last character

l \rightarrow 7 strings

l

lo \rightarrow 5 strings

o

lov \rightarrow 3 strings

v

lovi \rightarrow 1 string

i

On the basis of last character, recursively print all the possible strings

Code (Refer to video once)

```

void storeSuggestion (TrieNode * curr,
vector <string> & temp, string & prefix) {
    // Store answer when we arrive on terminal
    if (curr -> isTerminal) {
        temp.push_back (prefix);
    }

    for (char ch = 'a' ; ch <= 'z' ; ch++) {
        int index = ch - 'a';
        TrieNode * next = curr -> children[index];
        if (next != NULL) {
            // children exist
            prefix.push_back (ch);
            storeSuggestion (next, temp, prefix);
            // Backtracking
            prefix.pop_back ();
        }
    }
}

vector <vector <string>> getSuggestions (TrieNode
* root, string input) {

```

```

TrieNode* prev = root;
vector<vector<string>> output;
string prefix = "";
for (int i=0; i<input.length(); i++){
    char lastCh = input[i];
    int index = lastCh - 'a';
    TrieNode* curr = prev->children[
        index];

    if (curr == NULL)
        break;
    else {
        vector<string> temp;
        prefix.push-back(lastCh);
        storeSuggestion(curr, temp, prefix);
        output.push-back(temp);
        prev = curr;
    }
}
return output;
}

```

Time complexity = $O(nm^2)$
 $n \rightarrow$ no. of strings
 $m \rightarrow$ average length of string.