03/01/2024

HASHMAPS & TRIES
CLASS - 3

📂 1. Print All Words of Given Prefix String - I

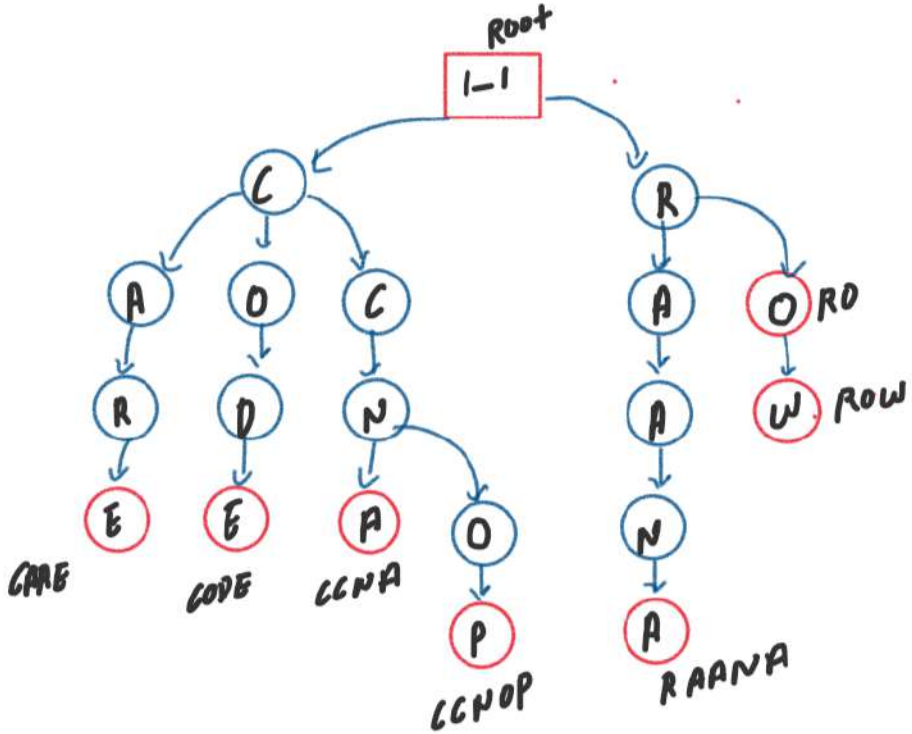
TRIE WORDS: "CCNA", "CCNOP", "CODE", "CARE", "RAANA", "RO", "ROW"

Example 1:
Input: inputStr = "CC"
Output: ["CCNA", "CCNOP"]

Example 2:
Input: inputStr = "R"
Output: ["RAANA", "RO", "ROW"]

Example 3:
Input: inputStr = "C"
Output: ["CCNA", "CCNOP", "CODE", "CARE"]

Example 4:
Input: inputStr = "B"
Output: []

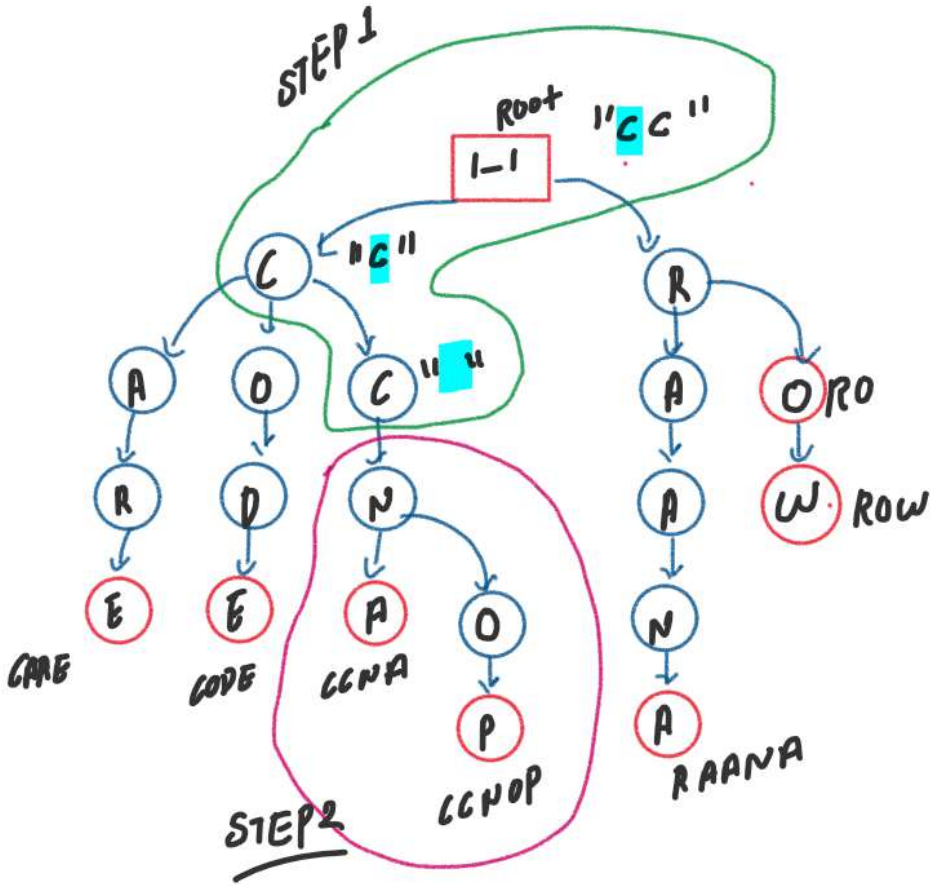


Algorithm

TRIE WORDS: "CCNA", "CCNOP", "CODE", "CARE", "RAANA", "RO", "ROW"

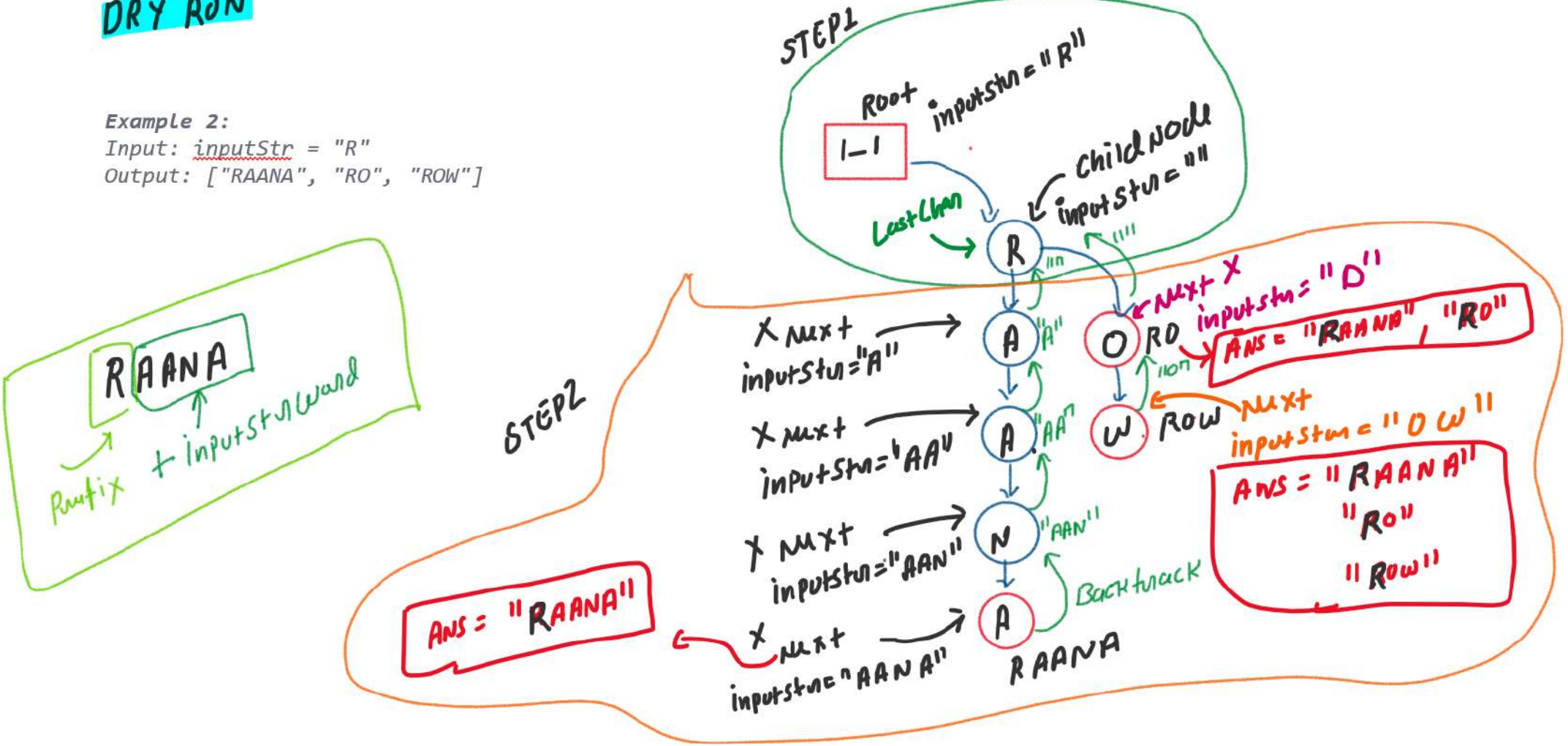
Example 1:
Input: inputStr = "CC"
Output: ["CCNA", "CCNOP"]

Algorithm:
Step 1: *traverse to the last character of input string*
Step 2: *find all solution below the last character of input string*



DRY RUN

Example 2:
Input: inputStr = "R"
Output: ["RAANA", "RO", "ROW"]



```cpp
// Print All Words of Given Prefix String - I

// Step 1: traverse to the last character of input string
void findPrefix(TrieNode* root, vector<string> &ans, string inputStr, string prefixStr){
    // Base case
    if(inputStr.length() == 0){
        TrieNode* lastChar = root;

        // Step 2: find all solution below the last character of input string
        storeWord(lastChar, ans, inputStr, prefixStr);

        return;
    }

    // 1 case hum solve kar lenge
    char ch = inputStr[0];
    int index = ch - 'a';
    TrieNode* childNode;

    if(root->children[index] != NULL){
        // Present hai-> Travese the root copy childNode
        childNode = root->children[index];
    }
    else{
        // Absent hai-> Yanhi se bapas chle jaoo
        return;
    }

    // Ab recursion solve kar lega
    findPrefix(childNode, ans, inputStr.substr(1), prefixStr);
}
```

```cpp
// Step 2: find all solution below the last character of input string
void storeWord(TrieNode* root, vector<string> &ans, string &inputStr, string &prefixStr){
    // Base case
    if(root->isTerminal == true){
        // store ans
        ans.push_back(prefixStr + inputStr);

        // return nhi karna hai kyunki ho skta hai ki
        // isTerminal ke bad koi or bhi character ho
    }

    for(char ch = 'a'; ch < 'z'; ch++){
        int index = ch - 'a';
        TrieNode* next = root->children[index];

        if(next != NULL){
            // Child exist karta hai
            inputStr.push_back(ch);
            // ab recursion solve kar lega
            storeWord(next, ans, inputStr, prefixStr);
            // Backtrack
            inputStr.pop_back();
        }
    }
}
```

2. Print All Words of Given Prefix String - II

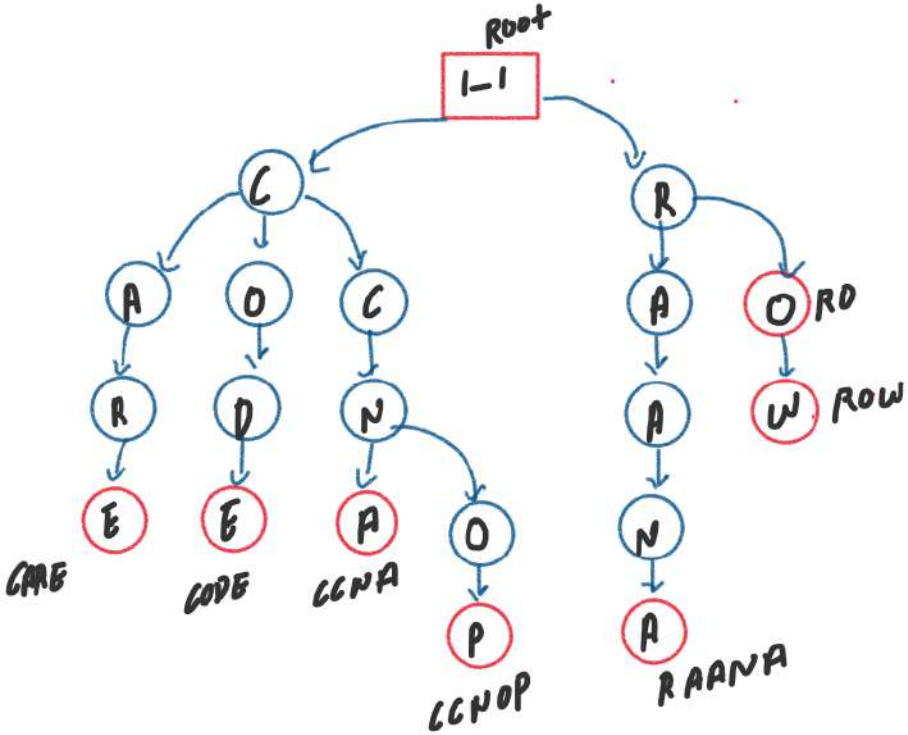
TRIE WORDS: "CCNA", "CCNOP", "CODE", "CARE", "RAANA", "RO", "ROW"

Example 1:
Input: inputStr = "CC"
Output: [["CCNA", "CCNOP", "CODE", "CARE"],["CCNA", "CCNOP"]]

Example 2:
Input: inputStr = "RO"
Output: [["RAANA", "RO", "ROW"],["RO", "ROW"]]

Example 3:
Input: inputStr = "C"
Output: [["CCNA", "CCNOP", "CODE", "CARE"]]

Example 4:
Input: inputStr = "B"
Output: []



Algo

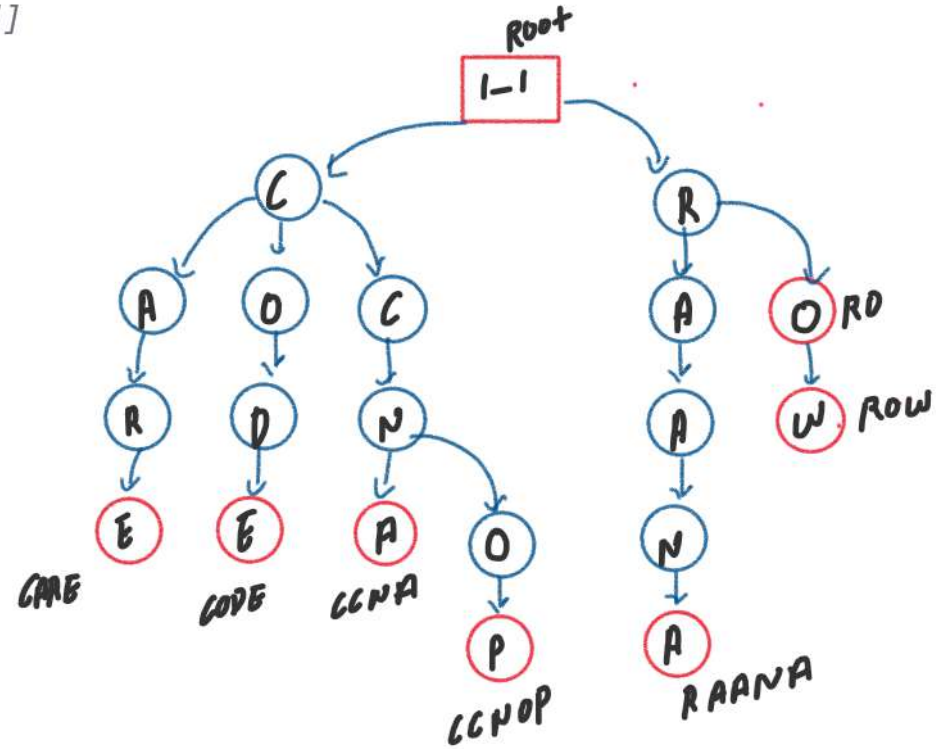
Example 1:
Input: inputStr = "CC"
Output: [["CCNA", "CCNOP", "CODE", "CARE"],["CCNA", "CCNOP"]]

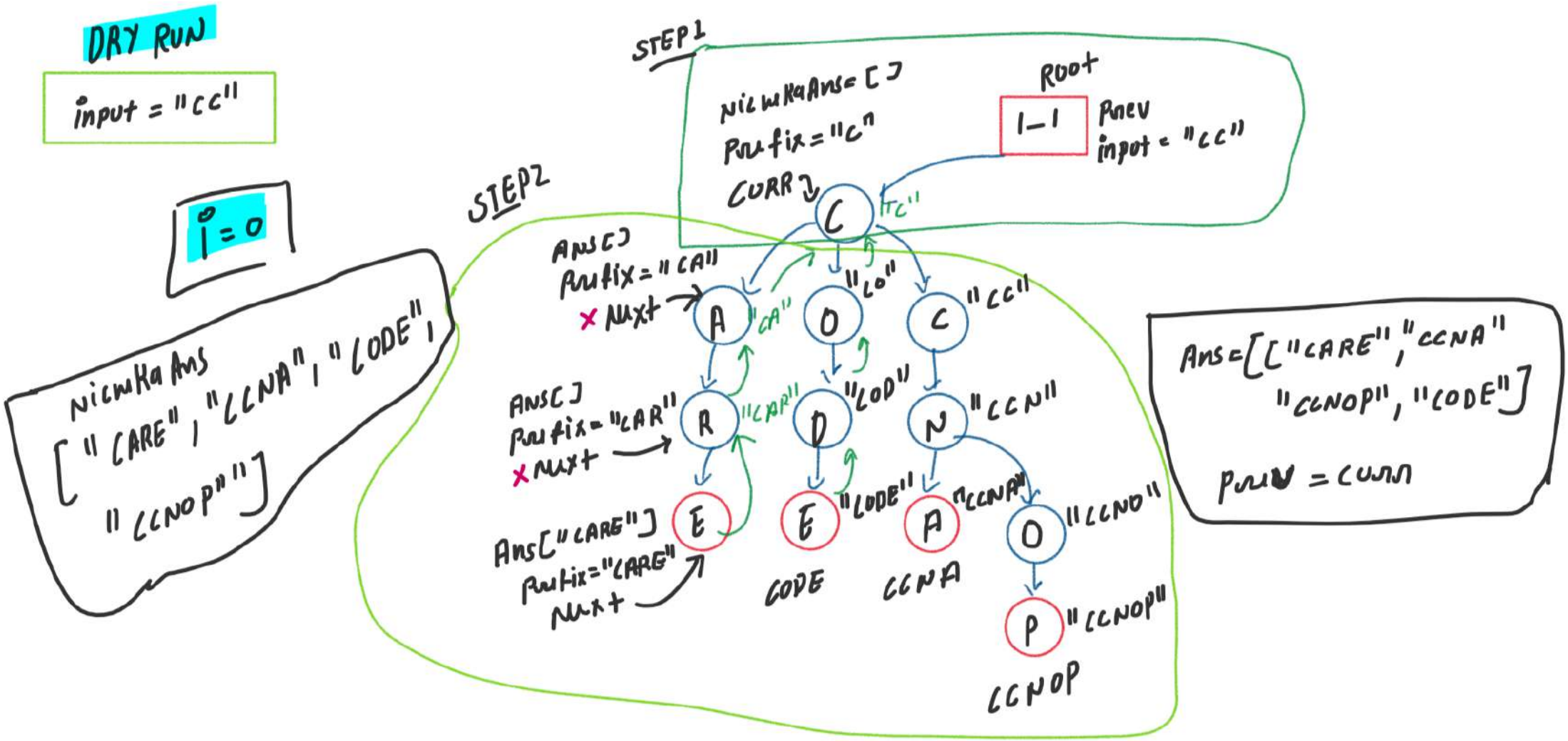
ALGORITHM:
Step 1: *input string ke har ek character ko as a last character assume karlo*

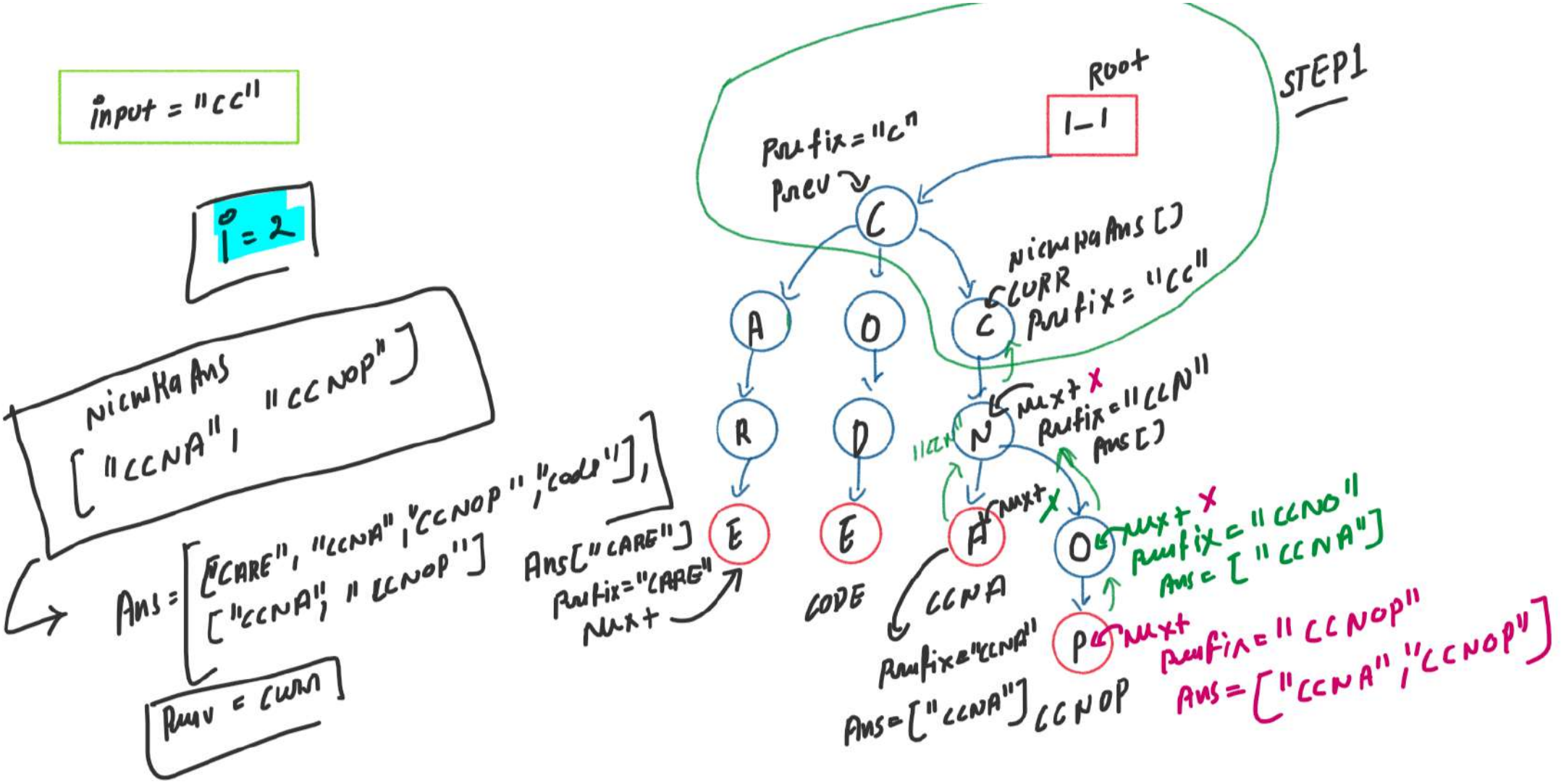
Step 2: *find all solution below the last character of input string*

C → CARE, CODE, CCNA, CCNOP

CC → CCNA, CCNOP



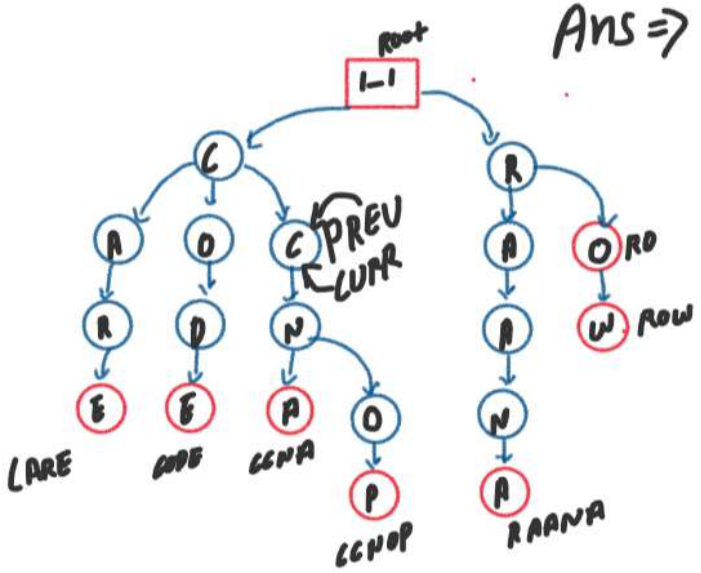

DRY RUN
input = "CC"
i = 0
NicwKaAns
["CARE", "CCNA", "CODE", "CCNOP"]
STEP1
NicwKaAns = []
Prefix = "C"
CURR
Root
1-1
Prev
input = "CC"
STEP2
Ans[]
Prefix = "CA"
Next
Ans[]
Prefix = "CAR"
Next
Ans["CARE"]
Prefix = "CARE"
Next
CODE
CCNA
CCNOP
Ans = [["CARE", "CCNA"
"CCNOP", "CODE"]
Prev = Curr


input = "CC"
i = 2
NicwHqAns
["CCNA", "CCNOP"]
Ans = [["CARE", "CCNA", "CCNOP", "code"], ["CCNA", "CCNOP"]]
Prev = Curr
Root
1-1
STEP1
Prefix = "C"
Prev
C
A
O
C
NicwHqAns []
CURR
Prefix = "CC"
R
D
N
Nxt X
Prefix = "CCN"
Ans []
E
E
A
O
P
Nxt X
Prefix = "CCNO"
Ans = ["CCNA"]
Nxt
Prefix = "CCNOP"
Ans = ["CCNA", "CCNOP"]
Ans["CARE"]
Prefix = "CARE"
Nxt
CODE
CCNA
Prefix = "CCNA"
Ans = ["CCNA"]
CCNOP

i = 3

→ STOP

Final Ans



Ans => [["CARE","CCNA","CCNOP","CODE"],["CCNA","CCNOP"]]

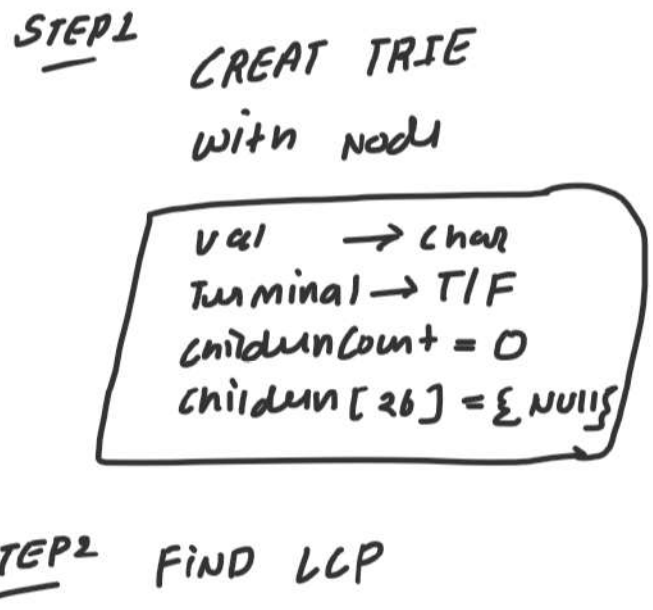
```cpp
// Print All Words of Given Prefix String - II

// Step 1: input string ke har ek character ko as a last character assume karlo
void findPrefix(TrieNode* root, vector<vector<string>> &ans, string inputStr){
    TrieNode* prev = root;
    string prefixStr = "";

    for (int i = 0; i < inputStr.length(); i++)
    {
        char lastChar = inputStr[0];
        int index = lastChar - 'a';
        TrieNode* curr = prev->children[index];

        if(curr == NULL){
            // Absent hai-> Loop se bahar ho jaao
            break;
        }
        else{
            // Present hai->
            // Step 2: find all solution below the last character of input string
            vector<string> nicheKaAns;
            prefixStr.push_back(lastChar);

            storeWord(curr, nicheKaAns, prefixStr);

            // Store nicheKaAns
            ans.push_back(nicheKaAns);
            // Prev ko update krna hamesha bhool jata hu (IMPORTANT)
            prev = curr;
        }
    }
}
```

```cpp
// Step 2: find all solution below the last character of input string
void storeWord(TrieNode* root, vector<string> &ans, string &prefixStr){
    // Base case
    if(root->isTerminal == true){
        // store ans
        ans.push_back(prefixStr);

        // return nhi karna hai kyunki ho skta hai ki
        // isTerminal ke bad koi or bhi character ho
    }

    for(char ch = 'a'; ch < 'z'; ch++){
        int index = ch - 'a';
        TrieNode* next = root->children[index];

        if(next != NULL){
            // Child exist karta hai
            prefixStr.push_back(ch);
            // ab recursion solve kar lega
            storeWord(next, ans, prefixStr);
            // Backtrack
            prefixStr.pop_back();
        }
    }
}
```

📂 3. Longest Common Prefix (Leectcode-14)

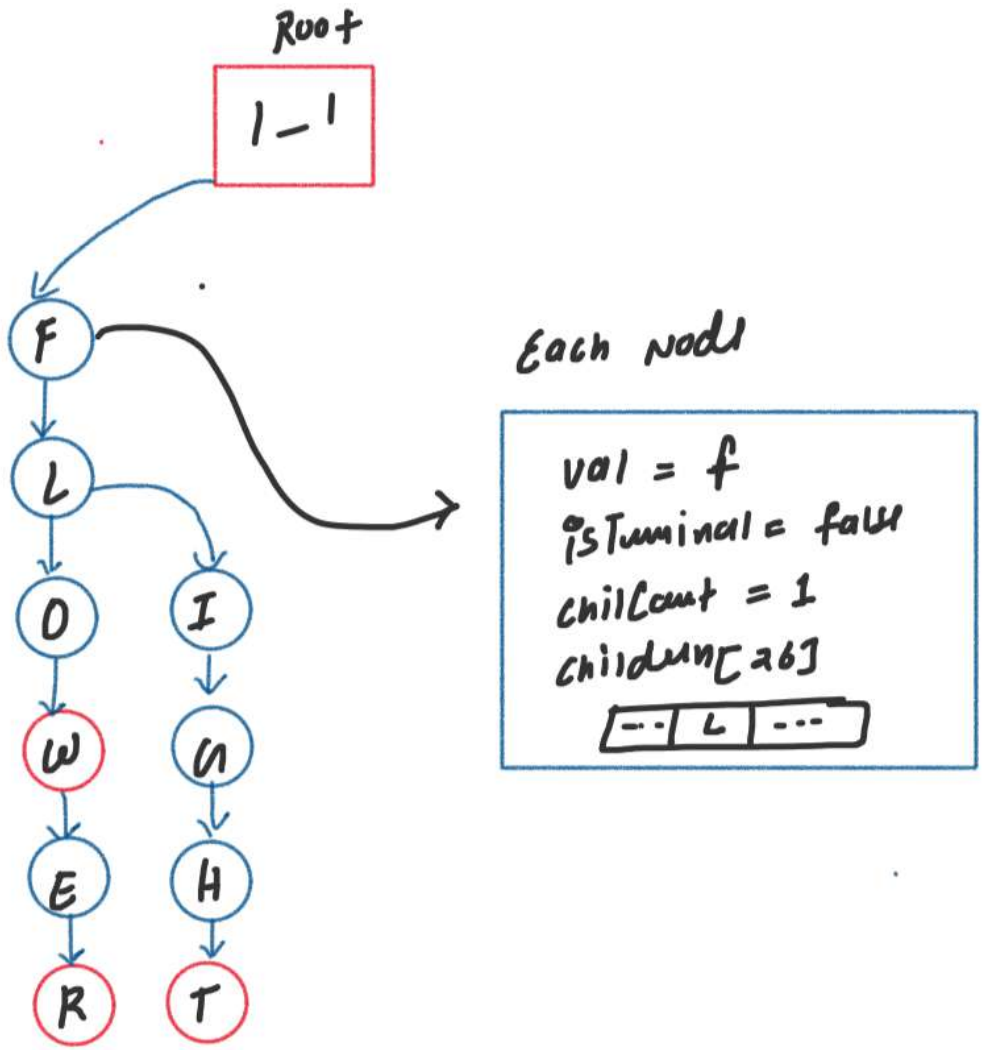
Example 1:
Input: strs = ["flower","flow","flight"]
Output: "fl"

Example 2:
Input: strs = ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings.



Example 1:
Input: strs = ["flower","flow","flight"]
Output: "fl"

STEP 1

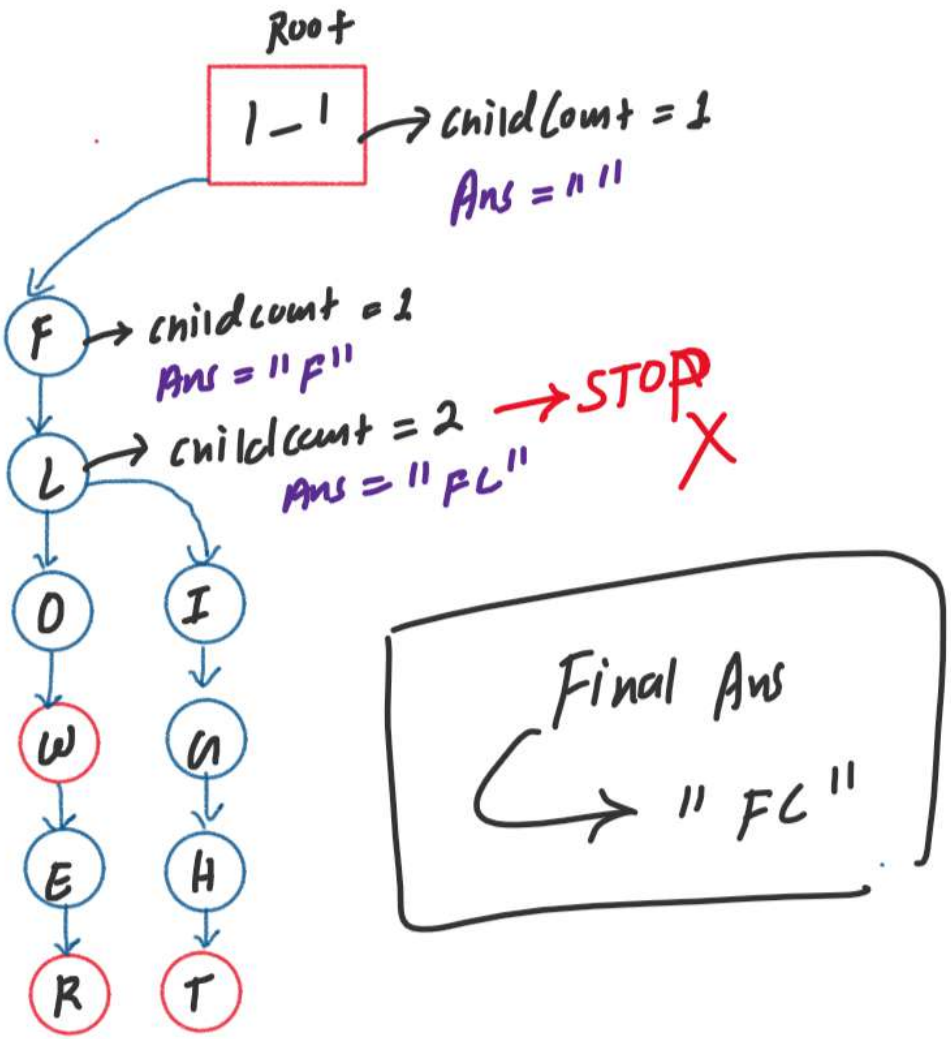


Example 1:
Input: strs = ["flower","flow","flight"]
Output: "fl"

STEP 2

if (child_count == 1)
↳ Traverse
↳ store Ans

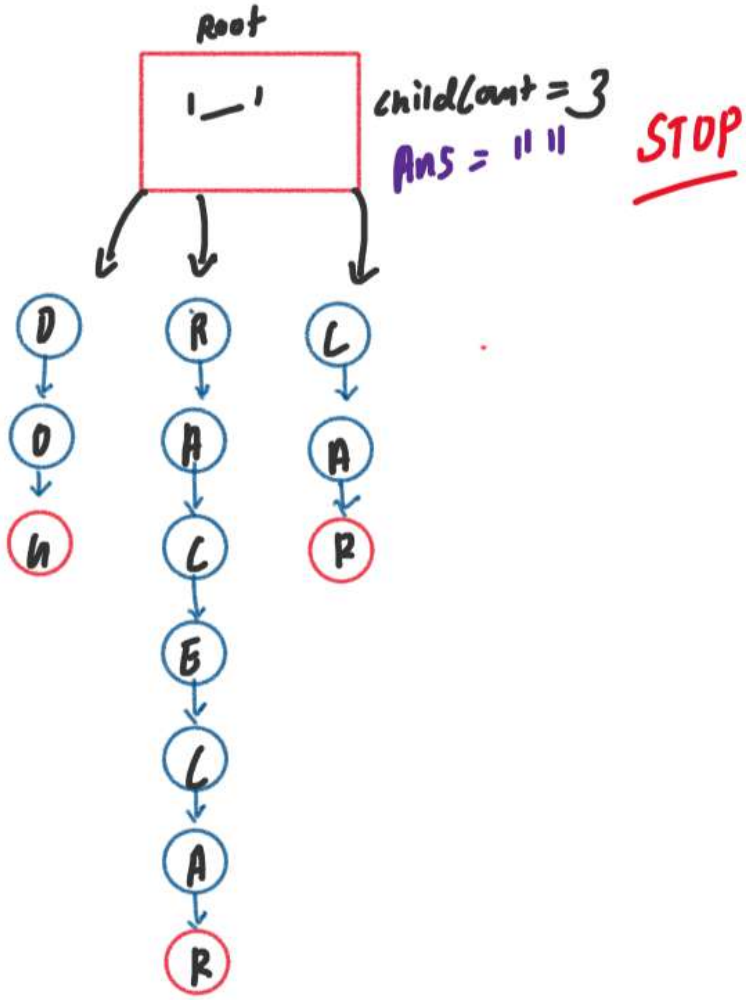
Else
↳ return



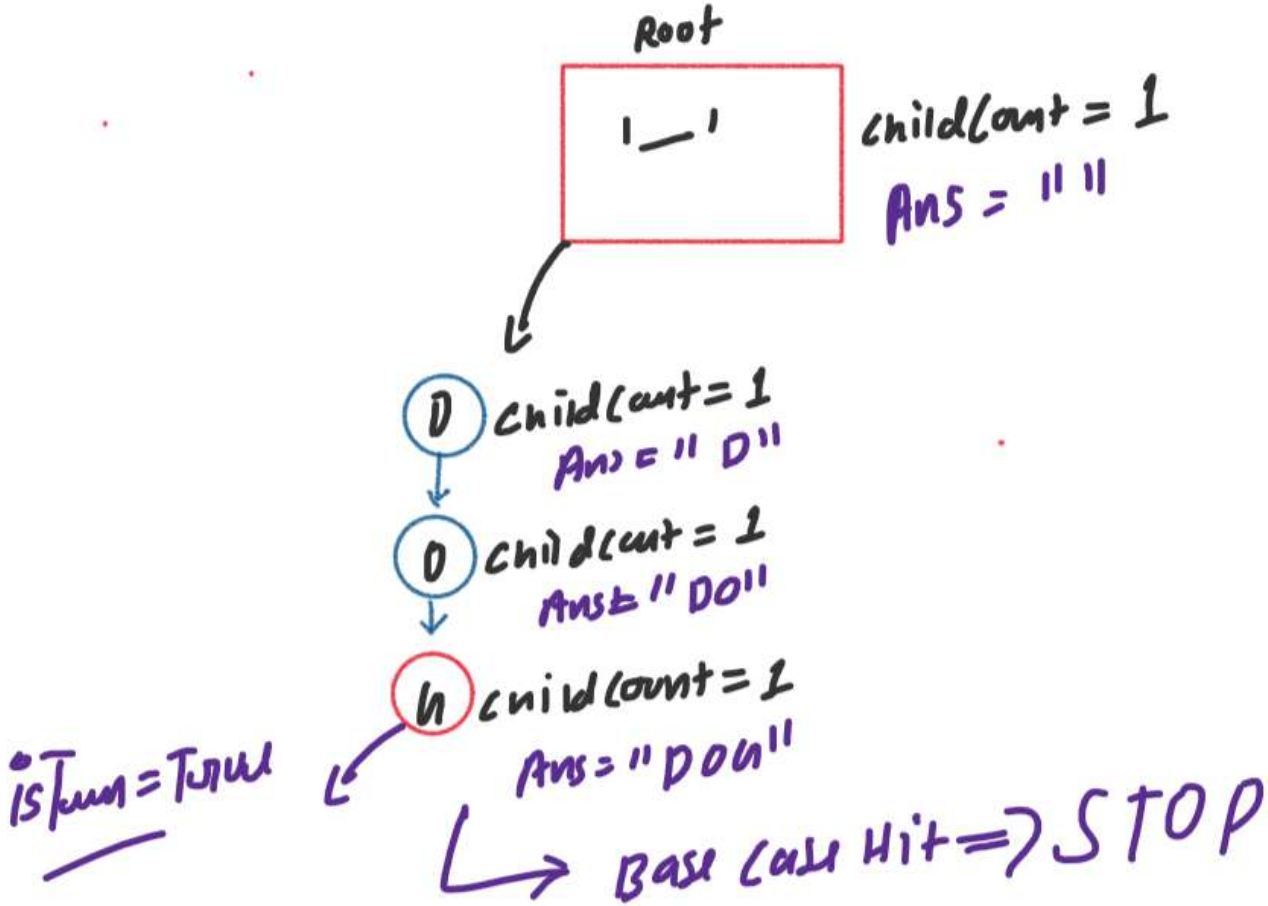
Example 2:

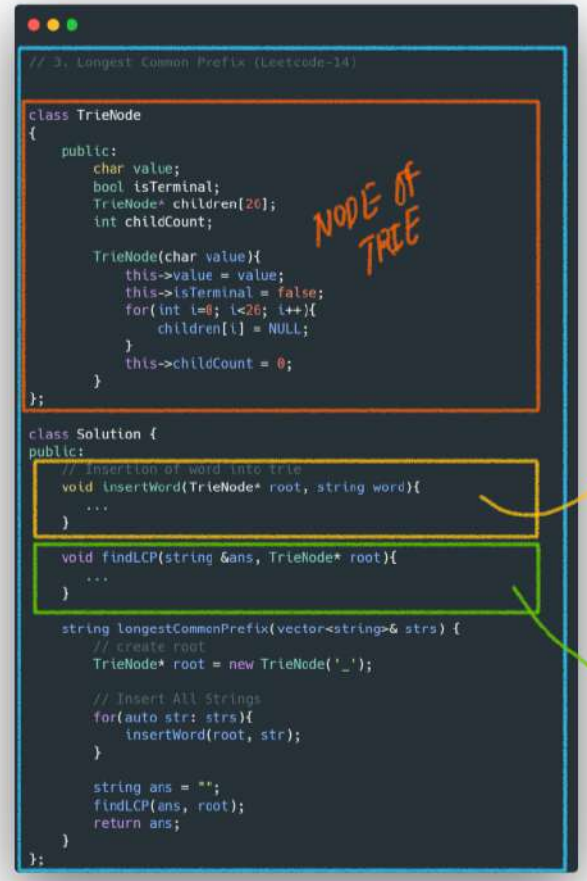
Input: strs = ["dog","racecar","car"]

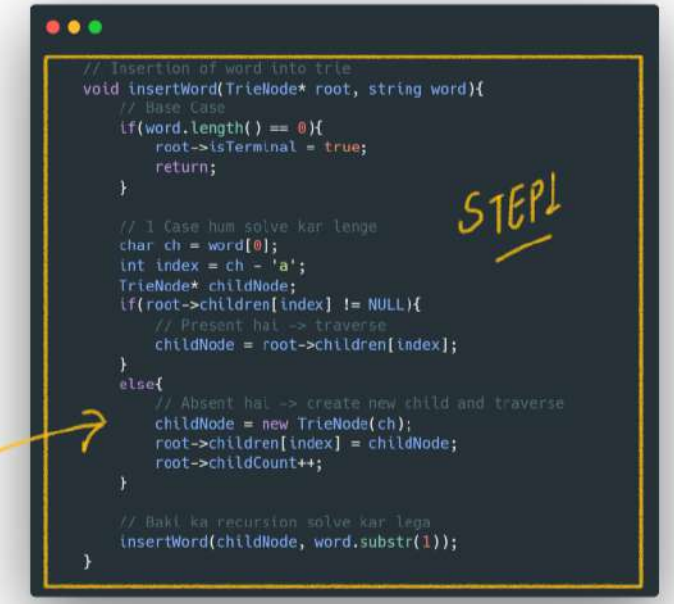
Output: ""



Example 3:
Input: strs = ["dog"]
Output: "dog"




// 3. Longest Common Prefix (Leetcode-14)
class TrieNode
{
public:
char value;
bool isTerminal;
TrieNode* children[26];
int childCount;
TrieNode(char value){
this->value = value;
this->isTerminal = false;
for(int i=0; i<26; i++){
children[i] = NULL;
}
this->childCount = 0;
}
};
NODE OF TRIE
class Solution {
public:
// Insertion of word into trie
void insertWord(TrieNode* root, string word){
...
}
void findLCP(string &ans, TrieNode* root){
...
}
string longestCommonPrefix(vector<string>& strs) {
// create root
TrieNode* root = new TrieNode('_');
// Insert All Strings
for(auto str: strs){
insertWord(root, str);
}
string ans = "";
findLCP(ans, root);
return ans;
}
};


// Insertion of word into trie
void insertWord(TrieNode* root, string word){
// Base Case
if(word.length() == 0){
root->isTerminal = true;
return;
}
// 1 Case hum solve kar lenge
char ch = word[0];
int index = ch - 'a';
TrieNode* childNode;
if(root->children[index] != NULL){
// Present hai -> traverse
childNode = root->children[index];
}
else{
// Absent hai -> create new child and traverse
childNode = new TrieNode(ch);
root->children[index] = childNode;
root->childCount++;
}
// Baki ka recursion solve kar lega
insertWord(childNode, word.substr(1));
}
STEP1


void findLCP(string &ans, TrieNode* root){
// Base case
if(root->isTerminal == true){
return;
}
// Abhi me root par hi khada hu
TrieNode* childNode;
if(root->childCount == 1){
// Child Tak Jaaooo
for(int i=0; i<26; i++){
if(root->children[i] != NULL){
// Mujhe child Node Mil Gya
childNode = root->children[i];
}
}
// store child node value into ans
ans.push_back(childNode->value);
}
else{
return;
}
// Ab me child Node par khda hu
// ab recursion solve kar lega
findLCP(ans, childNode);
}
STEP2