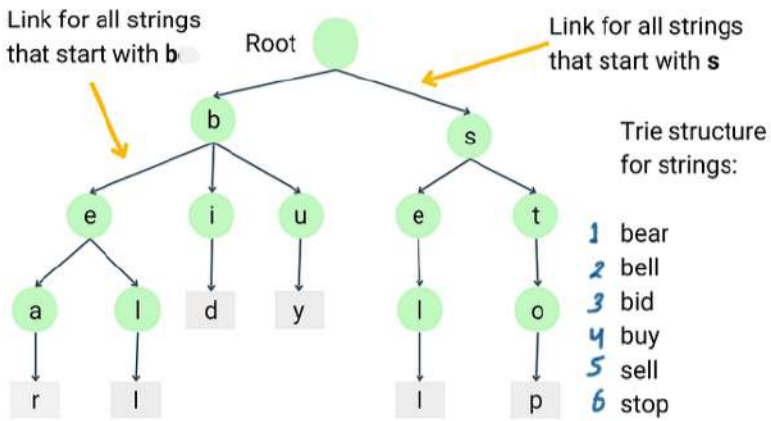
02/01/2024

HASHMAPS & TRIES
CLASS - 2

📂 1. What is a TRIE?

1. TRIE is also known as a multiway tree structure, digital tree or prefix tree.

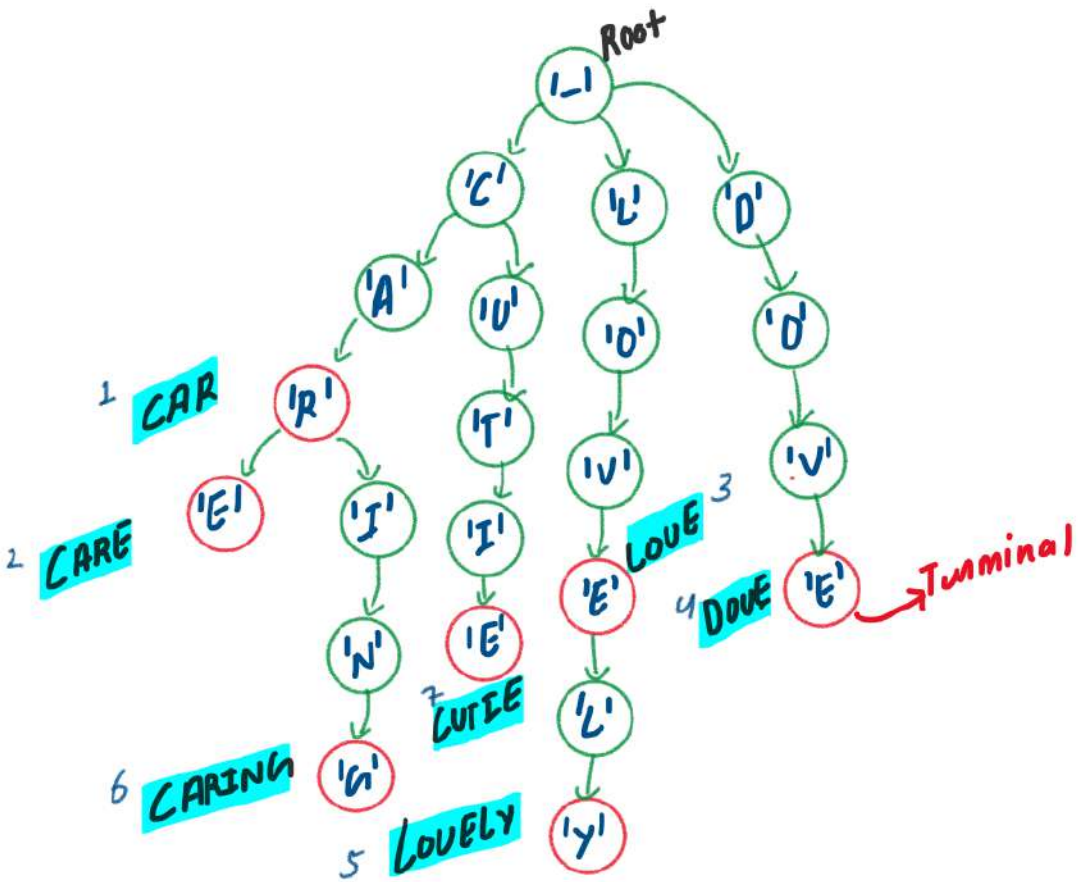
2. TRIE is a Data Structure that is used for pattern searching.



Basic Operations of TRIE:

1. Insertion
2. Searching
3. Deletion

2. How to organize the data in TRIE?



Algorithm	Input
Present ↳ TRAVERSE Absent ↳ CREATE NODE & TRAVERSE Terminal = T/F	STRINGS 1 CAR 2 CARE 3 LOVE 4 DOVE 5 LOVELY 6 CARING 7 CUTIE

3. CREATE TRIE NODE

NODE

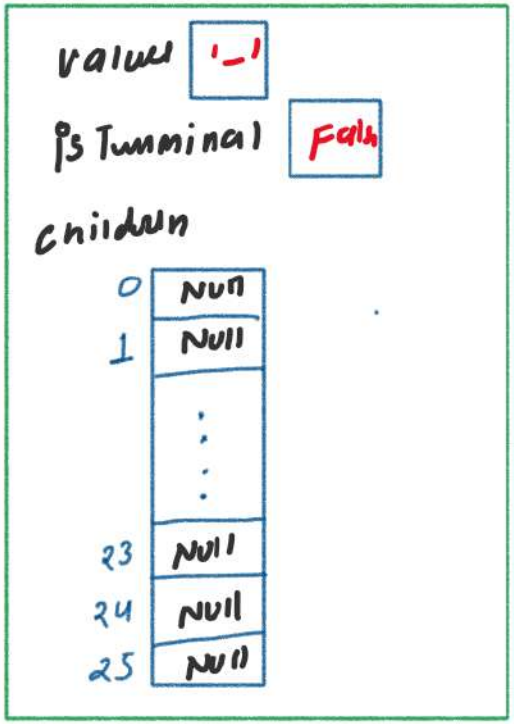
value → Character

Children → Array / map

isTerminal → T/F

```cpp
// Create Trie Node
class TrieNode
{
    public:
        char value;
        bool isTerminal;
        TrieNode* children[26];

        TrieNode(char value){
            this->value = value;
            this->isTerminal = false;
            for(int i=0; i<26; i++){
                children[i] = NULL;
            }
        }
};

int main(){
    TrieNode* root = new TrieNode('-');
    return 0;
}
```

Root node



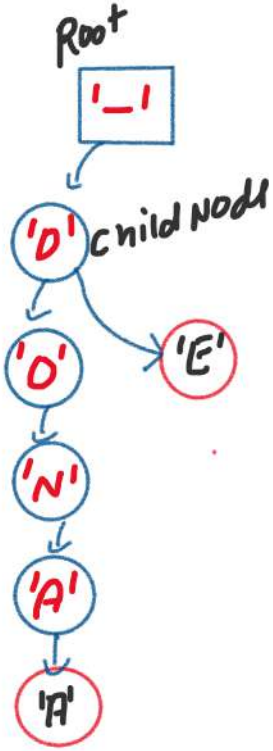
4. INSERTION METHOD OF TRIE

```cpp
// 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 -> to root ko aange traverse krdo
        childNode = root->children[index];
    }
    else{
        // Absent hai -> to ek new childNode create krke root ko aange traverse krdo
        childNode = new TrieNode(ch);
        root->children[index] = childNode;
    }

    // Baki ka recursion solve kar lega
    insertWord(childNode, word.substr(1));
}
```

DRY RUN ON WORD

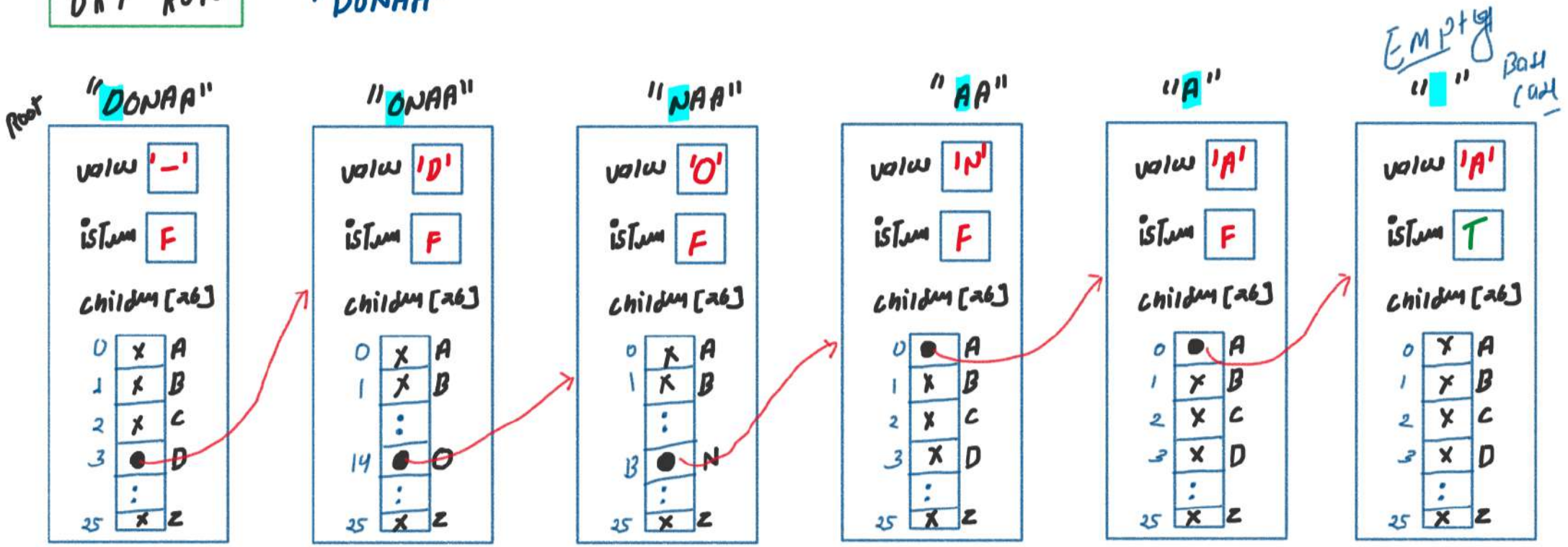
"DONAA", "DE"

T.C. ⇒ O(1)



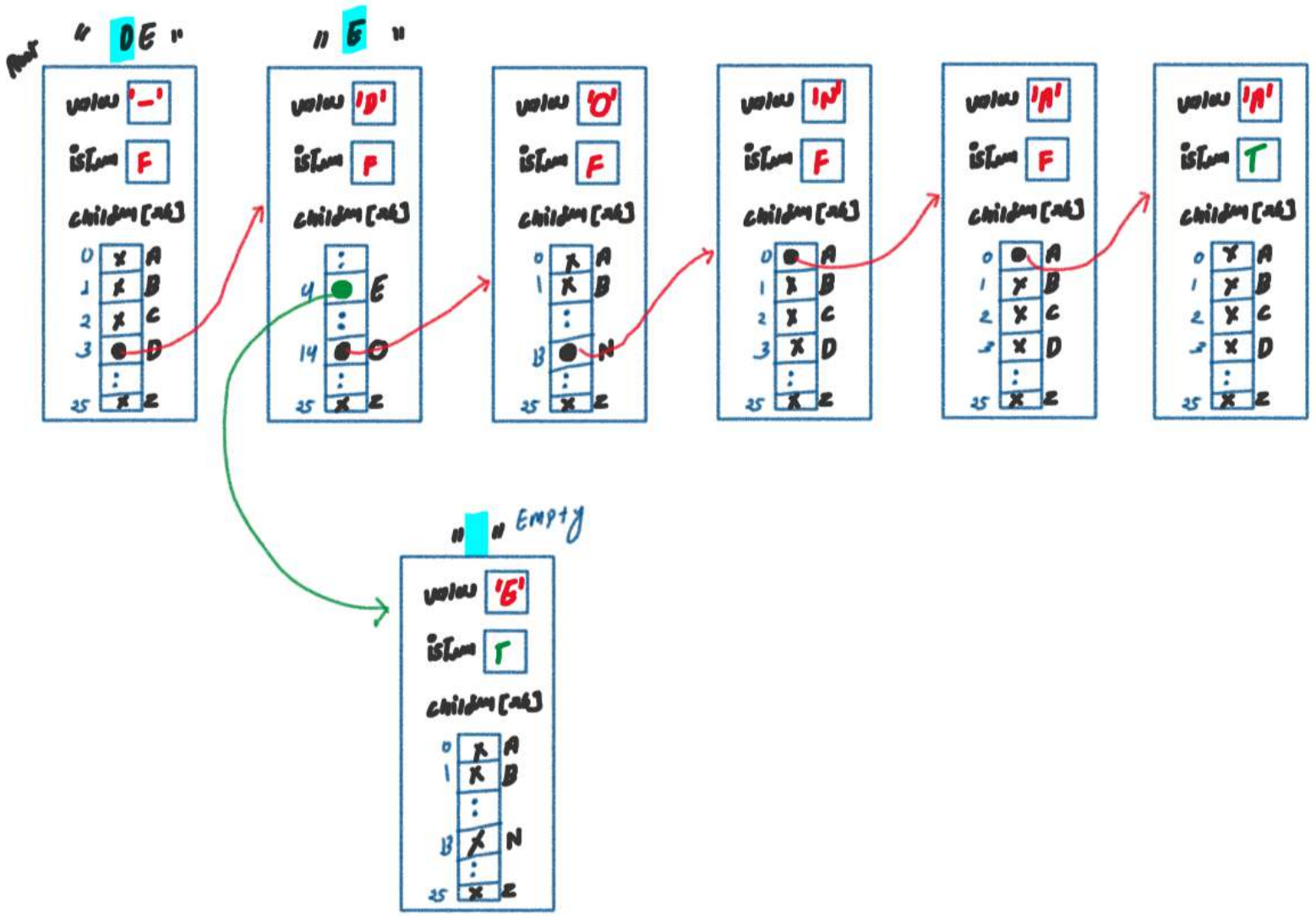
DRY RUN

"DONAA"



DRY RUN

"DE"


"DE"
"E"
root
value '-'
isEnd F
children[26]
value 'D'
isEnd F
value 'O'
isEnd F
value 'N'
isEnd F
value 'A'
isEnd F
value 'A'
isEnd T
"" Empty
value 'G'
isEnd T

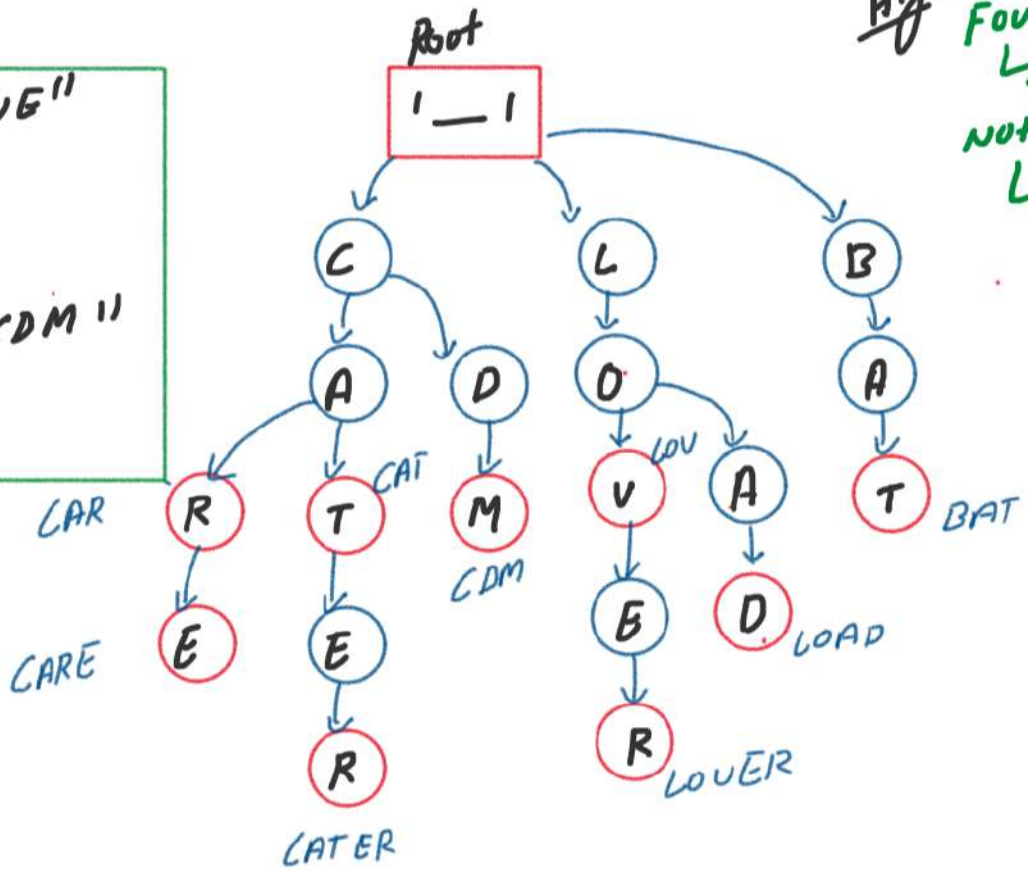
5. SEARCHING METHOD OF TRIE

Input word = "LOUG"

Output FALSE

Input word = "CDM"

Output TRUE



Algo

Found
↳ Traverse Hard

Not Found
↳ Return False

Insertion word

1 CATER
2 CARE
3 CDM
4 LOUER
5 LOAD
6 BAT
7 CAR
8 CAT
9 LOU

```cpp
// Searching of word from trie
bool searchWord(TrieNode* root, string word){
    // Base case
    if(word.length() == 0){
        return root->isTerminal;
    }

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

    if(root->children[index] != NULL){
        // Present hai -> found means traverse kro
        childNode = root->children[index];
    }
    else{
        // absent hai -> not found means word hai nhi hai
        return false;
    }

    // ab recursion solve kar lega
    bool recursionKaAns = searchWord(childNode, word.substr(1));
    return recursionKaAns;
}
```

T.C. = O(1)

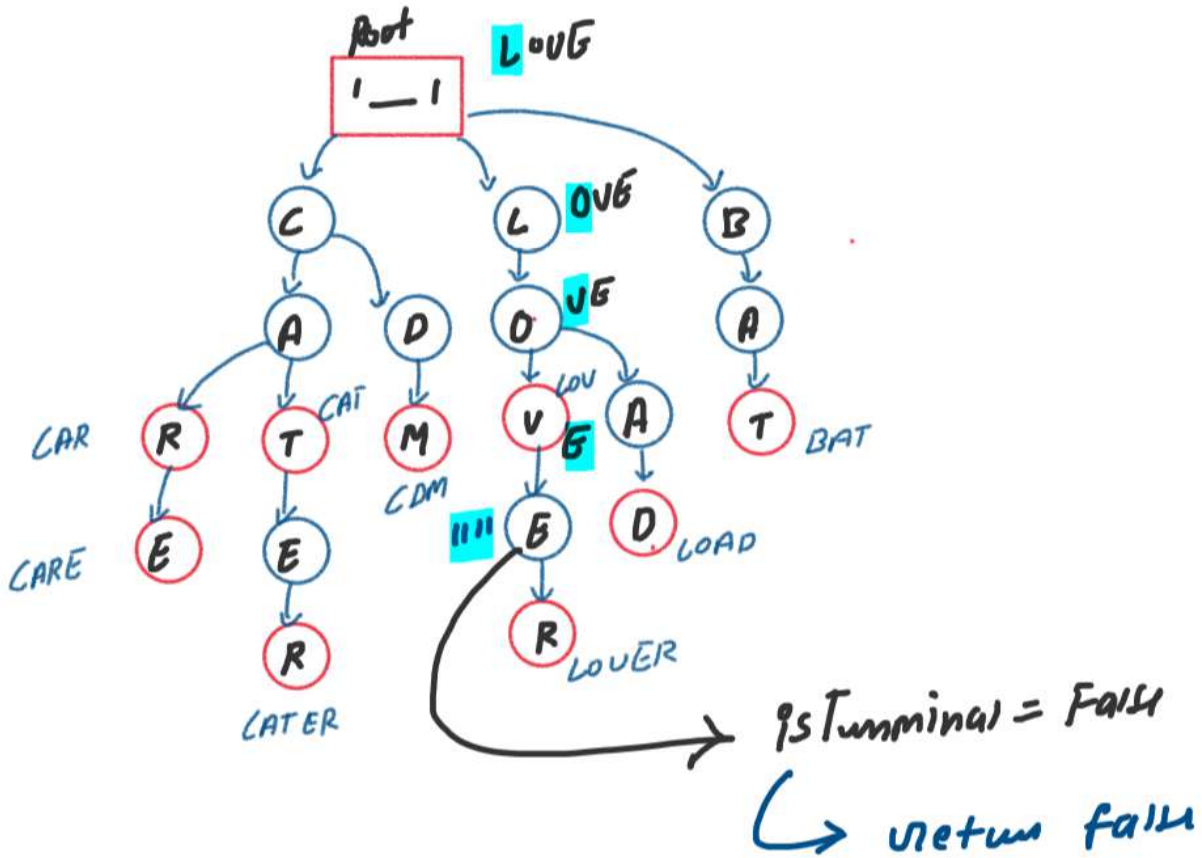
DRY RUN

Input word = "LOUG"

Output FALSE

Input word = "CDM"

Output TRUE



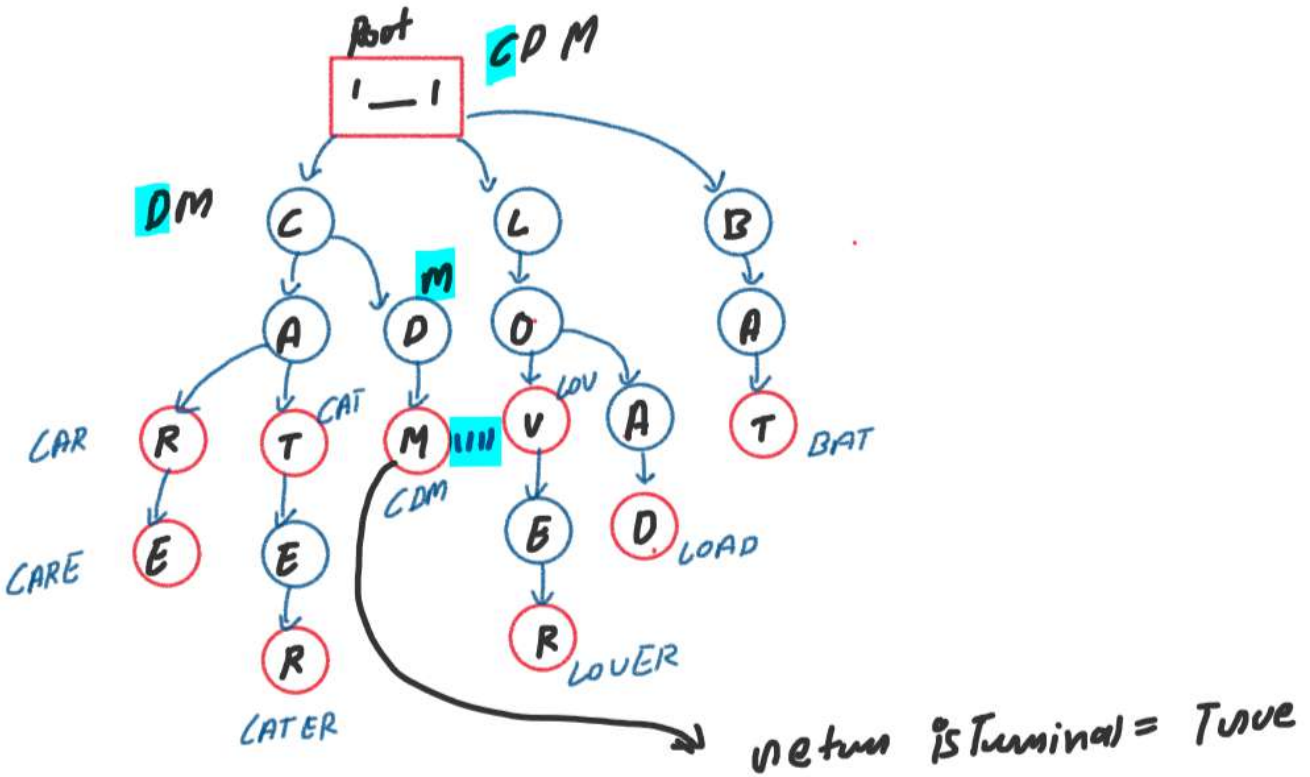
DRY RUN

Input word = "LOVE"

Output FALSE

Input word = "CDM"

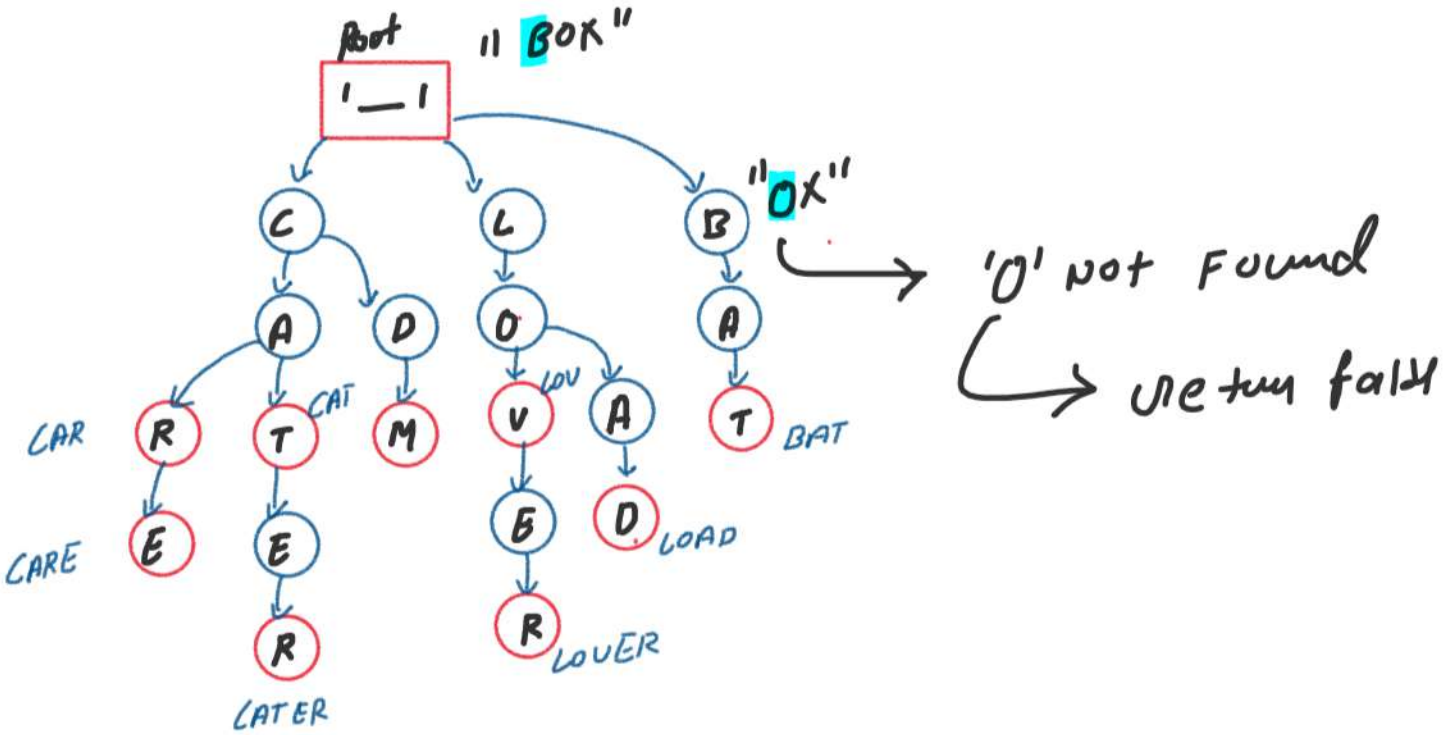
Output TRUE



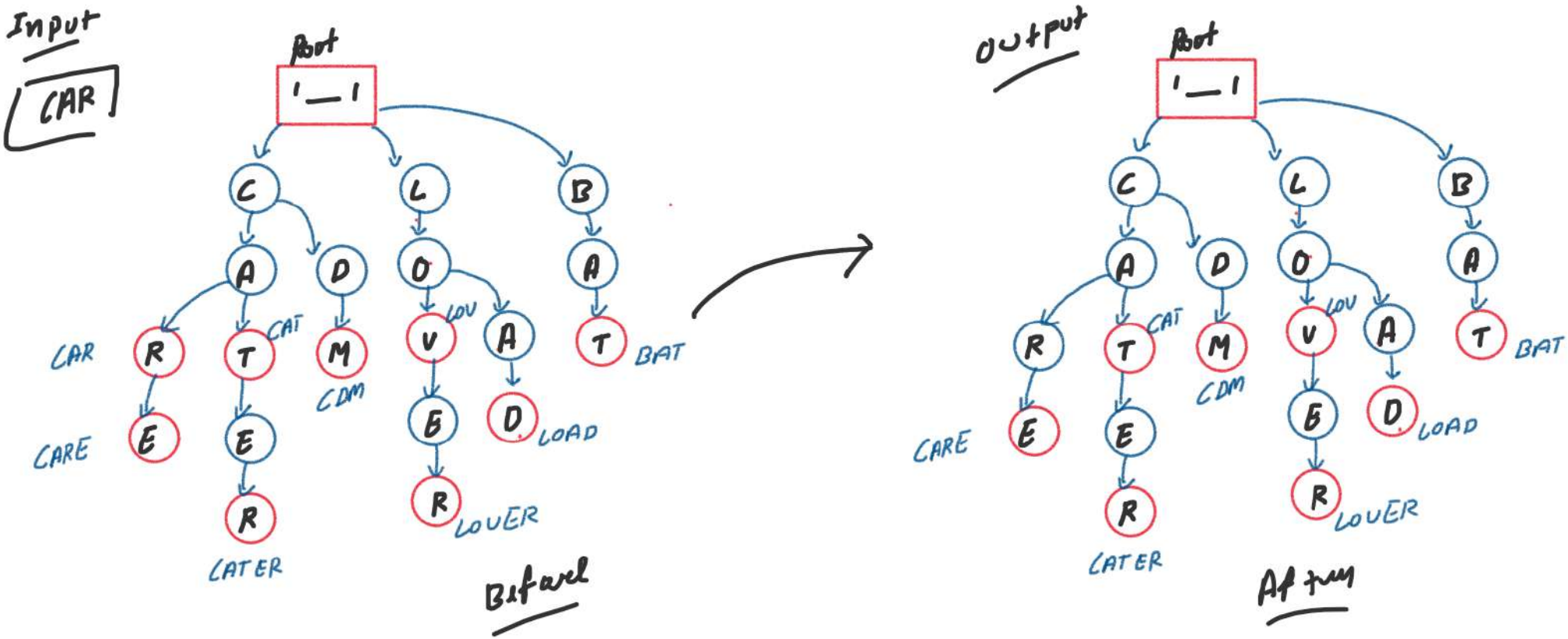
DRY RUN

Input word = "BOX"

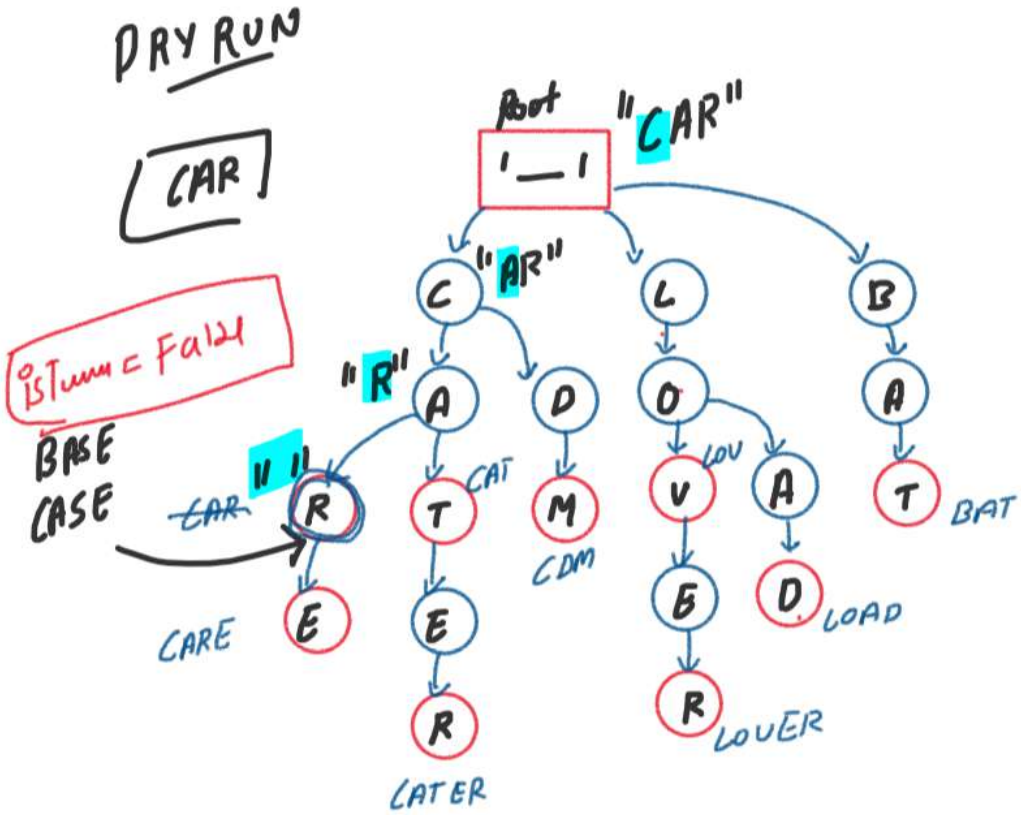
Output FALSE

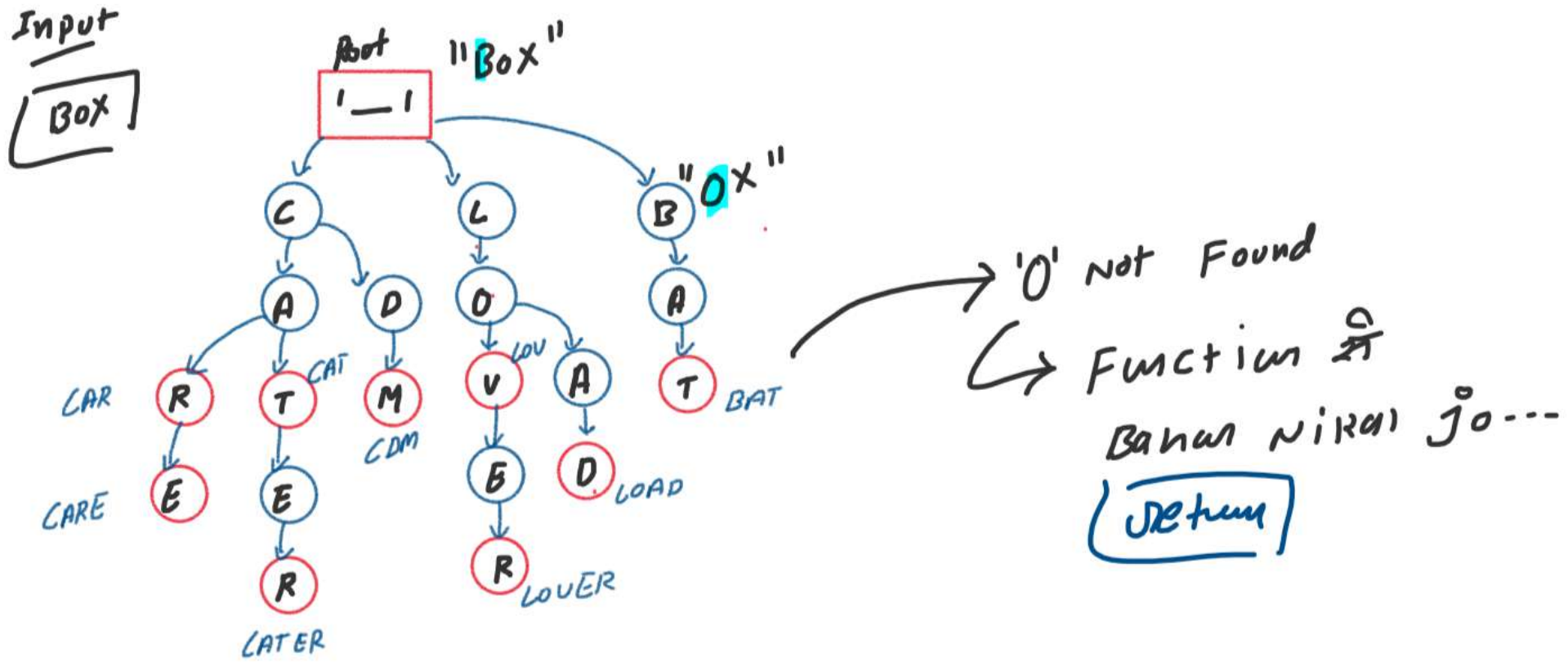


6. DELETION METHOD OF TRIE

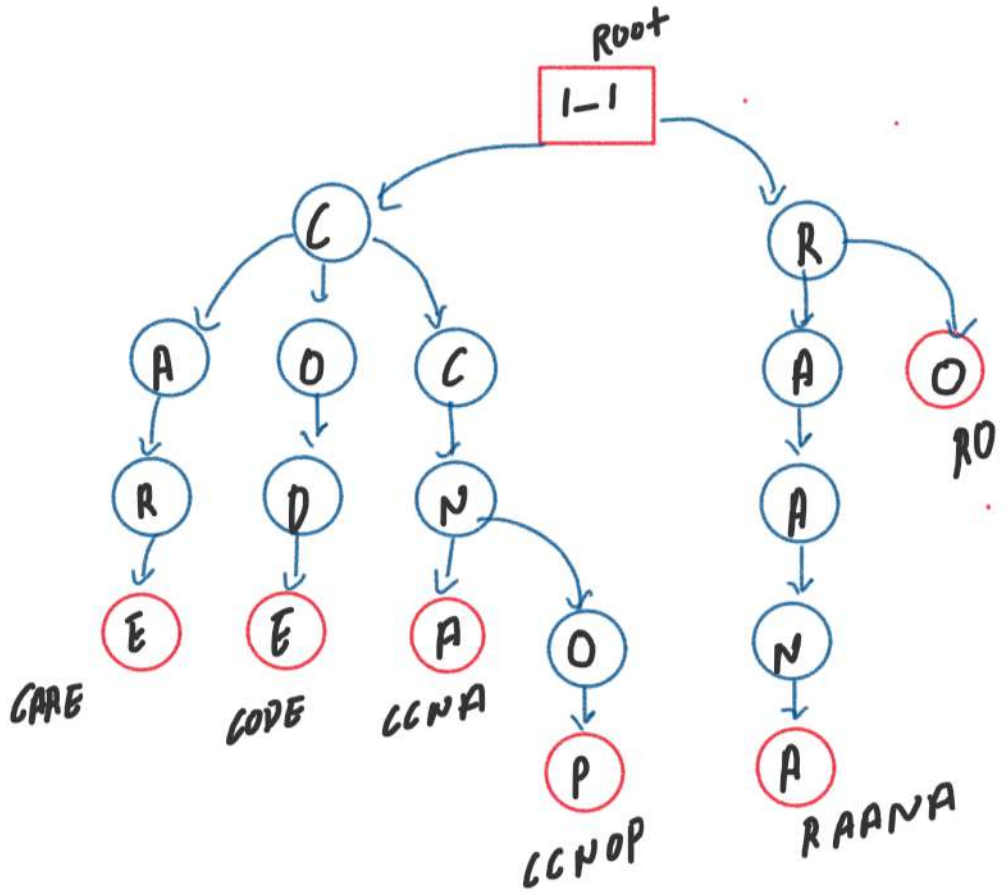


```cpp
// Deletion of word from trie
void deleteWord(TrieNode* root, string word){
    // Base case
    if(word.length() == 0){
        root->isTerminal = false;
        return;
    }

    // 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 -> function se bahar ho jaaooo
        return;
    }

    // ab recursion solve kar lega
    deleteWord(childNode, word.substr(1));
}
```




Input
BOX
Root
'—'
"BOX"
C
L
B
"OX"
A
D
O
A
CAR
R
T
CAT
M
CDM
V
LOU
A
T
BAT
CARE
E
E
B
D
LOAD
R
LOVER
CATER
'O' Not Found
Function से
Bahar Nikal जो---
Return

7. Print All Words of Given Prefix String



1 CODE
2 CARE
3 CCNA
4 CCNOP
5 RAANA
6 RO

Input prefix = "CC"

Output ["CCNA", "CCNOP"]

Input prefix = "R"

Output ["RAANA", "RO"]

Input prefix = "C"

Output ["CODE", "CARE", "CCNA", "CCNOP"]

Input prefix = "B"

Output [ ]