Day 9

Done	☑
Topic	Shortest Unique Prefix Problem(Hard) Start With Problem(Easy)
Languages	Java
Difficulty	⭐⭐⭐
Date Started	@March 9, 2023 11:57 PM
Date Completed	@March 10, 2023 2:33 AM
Related to Progress (Days)	Your Progress

What I Learned Today

To find shortest unique prefix for every word in given list.

To check whether given prefix string is in the list of words.

Key Concepts

Start With Problem

Check if the word inserted previously has the prefix “prefix” or not.

It is similar to a search operation. Start from the root of the trie and traverse through the “prefix”.If the reference trie for a character is not present return false else transverse to the reference trie.Once the “prefix’ is traversed completely character by character **return true**.

Shortest Unique Prefix Problem

Find shortest unique prefix for every word in a given list | Set 1 (Using Trie)

Given an array of words, find all shortest unique prefixes to represent each word in the given array. Assume that no word is prefix of another.

Examples:

```
Input: arr[] = {"zebra", "dog", "duck", "dove"}
Output: dog, dov, du, z
Explanation: dog => dog
             dove => dov
             duck => du
             zebra => z

Input: arr[] =  {"geeksgeeks", "geeksquiz", "geeksforgeeks"};
Output: geeksf, geeksg, geeksq}
```

A **Simple Solution** is to consider every prefix of every word (starting from the shortest to largest), and if a prefix is not prefix of any other string, then print it.

An **Efficient Solution** is to use **Trie**.

The idea is to maintain a count in every node. Below are steps.

1) Construct a **Trie** of all words. Also maintain frequency of every node (Here frequency is number of times node is visited during insertion). Time complexity of this step is O(N) where N is total number of characters in all words.

Quic

Link

Starts

Tutor

Docu

Uniqu

Prefix

Tutor

Docu

2) Now, for every word, we find the character nearest to the root with frequency as 1. The prefix of the word is path from root to this character. To do this, we can traverse Trie starting from root. For every node being traversed, we check its frequency. If frequency is one, we print all characters from root to this node and don’t traverse down this node.

Time complexity if this step also is O(N) where N is total number of characters in all words.

Code Snippets

```
public class StartsWith {
    static class Node {
        Node[] children = new Node[26];
        boolean eow = false;

        Node() {
            for (int i = 0; i < 26; i++) {
                children[i] = null;
            }
        }
    }

    public static void insert(String word) { //O(L) where L is length of largest word
        Node curr = root;
        for (int level = 0; level < word.length(); level++) {
            int idx = word.charAt(level) - 'a';
            if (curr.children[idx] == null) {
                curr.children[idx] = new Node();
            }
            curr = curr.children[idx];
        }
        curr.eow = true;
    }
    public static boolean startsWith(String prefix){ //O(L) where l is the length of prefix
        Node curr = root;
        for (int level = 0; level < prefix.length(); level++) {
            int idx = prefix.charAt(level) - 'a';
            if (curr.children[idx] == null) {
                return false;
            }
            curr = curr.children[idx];
        }
        return true;
    }

    public static Node root = new Node();

    public static void main(String[] args) {
        String[] word = {"the", "a", "there", "their", "any", "thee"};
        for (int i = 0; i < word.length; i++) {
            insert(word[i]);
        }
        System.out.println(startsWith("tho"));
    }
}
```

```
public class ShortestPrefix {
    static class Node {
        Node[] children = new Node[26];
        boolean eow = false;
        int freq;

        public Node() {
            for (int i = 0; i < children.length; i++) {
                children[i] = null;
            }
            freq = 1;
        }
    }
    public static void insert(String word){
        Node curr=root;
        for (int level=0; level<word.length();level++){
            int idx = word.charAt(level)-'a';
            if (curr.children[idx]==null){
                curr.children[idx]= new Node();
            }
            else {
                curr.children[idx].freq++;
            }
            curr=curr.children[idx];
```

```
        }
        curr.eow=true;
    }
    public static void shortestPrefix(Node root,String ans){ //O(L) where l is level in my trie or longest word
        if (root==null){
            return;
        }
        if (root.freq==1){
            System.out.println(ans);
            return;
        }
        for (int i=0;i<root.children.length;i++){
            if (root.children[i]!=null){
                shortestPrefix(root.children[i],ans+(char)(i+'a'));
            }
        }
    }

    public static Node root = new Node();
    public static void main(String[] args) {
        String[] arr = {"zebra","dog","duck","dove"};
        for (int i=0;i<arr.length;i++){
            insert(arr[i]);
        }
        root.freq=-1;
        shortestPrefix(root,""); //answer will be in alphatical order because by default trie me string alphabatically store hote hai
    }
}
```

Challenges Experienced

In Shortest Unique Path at

```
if (root.children[i]!=null){
                shortestPrefix(root.children[i],ans+(char)(i+'a'));
            }
```

Resources Used

Alpha, GeeksForGeeks