Day 8

Done	☑
Topic	Tries : Implementation | Searching Word Break Problem(Hard)
Languages	Java
Difficulty	⭐⭐⭐
Date Started	@March 8, 2023 11:40 PM
Date Completed	@March 9, 2023 1:46 AM
Related to Progress (Days)	Your Progress

What I Learned Today

Tries, Its Implementation, Searching in Tries

Word Break Problem(Hard)

Key Concepts

Trie

Trie is a type of k-ary search tree used for storing and searching a specific key from a set. Using Trie, search complexities can be brought to optimal limit (key length).

The tries are used in spell checking programs.

- Preprocessing pattern improves the performance of pattern matching algorithm. But if a text is very large then it is better to preprocess text instead of pattern for efficient search.
- A trie is a data structure that supports pattern matching queries in time proportional to the pattern size.

If we store keys in a **binary search tree**, a well balanced BST will need time proportional to **M * log N** , where **M** is the maximum string length and **N** is the number of keys in the tree. Using Trie, the key can be searched in **O(M)** time. However, the penalty is on Trie storage requirements (Please refer to **Applications of Trie** for more details).

Quic

Link

Trie Tutor

Docu

Word Break

Tutor

Docu

Insert Operation in Trie:

Inserting a key into Trie is a simple approach.

- Every character of the input key is inserted as an individual Trie node. Note that the **children** is an array of pointers (or references) to next-level trie nodes.
- The key character acts as an index to the array **children**.
- If the input key is new or an extension of the existing key, construct non-existing nodes of the key, and mark the end of the word for the last node.
- If the input key is a prefix of the existing key in Trie, Simply mark the last node of the key as the end of a word.

Search Operation in Trie:

Searching for a key is similar to the insert operation. However, It only **compares the characters and moves down**. The search can terminate due to the end of a string or lack of key in the trie.

- In the former case, if the **isEndofWord** field of the last node is true, then the key exists in the trie.

- In the second case, the search terminates without examining all the characters of the key, since the key is not present in the trie.

Word Break Problem (Hard)

Given an input string and a dictionary of words, find out if the input string can be segmented into a space-separated sequence of dictionary words. See following examples for more details.

This is a famous Google interview question, also being asked by many other companies now a days.

```
Consider the following dictionary
{ i, like, sam, sung, samsung, mobile, ice, cream, icecream, man, go, mango}

Input:  ilike
Output: Yes
The string can be segmented as "i like".

Input:  ilikesamsung
Output: Yes
The string can be segmented as "i like samsung"
or "i like sam sung".
```

Recursive implementation:

The idea is simple, we consider each prefix and search for it in dictionary. If the prefix is present in dictionary, we recur for rest of the string (or suffix).

If the recursive call for suffix returns true, we return true, otherwise we try next prefix. If we have tried all prefixes and none of them resulted in a solution, we return false.

Code Snippets

```
public class TriesBasics {
    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 search(String word) { //O(L)
        Node curr = root;
        for (int level = 0; level < word.length(); level++) {
            int idx = word.charAt(level) - 'a';
            if (curr.children[idx] == null) {
                return false;
            }
            curr = curr.children[idx];
        }
        return curr.eow;
    }

    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(search("thee"));
    }
}
```

```
public class WordBreak {
    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 search(String word) { //O(L)
        Node curr = root;
        for (int level = 0; level < word.length(); level++) {
            int idx = word.charAt(level) - 'a';
            if (curr.children[idx] == null) {
                return false;
            }
            curr = curr.children[idx];
        }
        return curr.eow;
    }

    public static boolean wordBreak(String key) {
        if (key.length() == 0) {
            return true;
        }
        for (int i = 1; i <= key.length(); i++) {
            //substring(beg idx, last idx) -> ye last index jo hai exclusive hota hai ye aayega hi nahi
            // suppose i=0 rakha to to substring(0,i) invalid ho jayega
            if (search(key.substring(0, i)) && wordBreak(key.substring(i))) { // wordBreak ki substring me i ko as a beggining ki tara
                return true;
            }
        }
        return false;
    }

    public static Node root = new Node();

    public static void main(String[] args) {
        String[] arr = {"i", "like", "sam", "samsung", "mobile", "ice"};
        for (int i = 0; i < arr.length; i++) {
            insert(arr[i]);
        }

        String key = "ilikesamsung";
        System.out.println(wordBreak(key));

    }
}
```

Challenges Experienced

Nothing that much

Resources Used

Alpha, GeeksForGeeks