

CSE 4304-Data Structures Lab. Winter 2022

Date: November 1, 2022.

Target Group: 1A

Topic: Trie

Instructions:

- Task naming format: fullID_L05_T01_1A.c/CPP
- **Solutions with less efficient approaches will be considered for partial marks.**

A [trie](#) (pronounced as "try") or prefix tree is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. This data structure has various applications, such as autocomplete and spellchecker. Implement the Trie class:

- **Trie()**: Initializes the trie object.
- **void insert(String word)**: Inserts the string **word** into the trie.
- **boolean search(String word)**: Returns true if the string **word** is in the trie (i.e., was inserted before), and false otherwise.
- **boolean startsWith(String pre)**: Returns true if there is a previously inserted string **word** that has the prefix **pre**, and false otherwise.

```
class Trie {
public:
    int trie[300005][50];
    int id = 1;
    int endmark[300005];
    int pre[300005];
    Trie() {
        memset(trie, 0, sizeof(trie));
        memset(endmark, 0, sizeof(endmark));
        memset(pre, 0, sizeof(pre));
    }

    void insert(string word) {
        int row = 1;
        for(int i = 0; i < word.size(); i++)
        {
            int ch = word[i] - 'a';
            if(trie[row][ch] == 0)
            {
                trie[row][ch] = ++id;
            }
            row = trie[row][ch];
            pre[row]++;
        }
        endmark[row] = 1;
    }
}
```

```

bool search(string word) {
    int row = 1;
    for(int i = 0; i < word.size();i++)
    {
        int ch = word[i] - 'a';
        if(trie[row][ch] == 0) return false;
        row = trie[row][ch];
    }
    return (endmark[row] == 1);
}

bool startsWith(string prefix) {
    int row = 1;
    for(int i = 0; i < prefix.size();i++)
    {
        int ch = prefix[i] - 'a';
        if(trie[row][ch] == 0) return false;
        row = trie[row][ch];
    }
    return (pre[row] >= 1);
}
};

/**
 * Your Trie object will be instantiated and called as such:
 * Trie* obj = new Trie();
 * obj->insert(word);
 * bool param_2 = obj->search(word);
 * bool param_3 = obj->startsWith(prefix);
 */

```

- Leetcode Problem Link: [Implement Trie \(Prefix Tree\)](#)

Task 01:

Design a data structure that supports adding new words and finding if a string matches any previously added string.

Implement the WordDictionary class:

- **WordDictionary()**: Initializes the object.
- **void addWord(word)**: Adds **word** to the data structure, it can be matched later.
- **bool search(word)**: Returns true if there is any string in the data structure that matches **word** or false otherwise. word may contain dots '.' where dots can be matched with any letter.

| Sample Input | Sample Output |
|---|---|
| <ul style="list-style-type: none">• WordDictionary()• addWord('bad')• addWord('dad')• addWord('mad')• search('pad')• search('bad')• search('.ad')• search('b..') | <ul style="list-style-type: none">• Object Created• Word Added• Word Added• Word Added• Word Not Found• Word Found• Word Found• Word Found |

Constraints:

- All words will contain only English Lower Case Letters
- Word Lengths will be less than 26

Task 02:

You are given an array of strings called ***products*** and a string **searchWord**.

Design a system that suggests at most three product names from ***products*** array after each character of **searchWord** is typed. Suggested products should have a common prefix with **searchWord**. If more than three products have a common prefix, output the three lexicographically minimum products.

| | |
|---------------|--|
| Sample Input | products = ["mobile","mouse","moneypot","monitor","mousepad"] searchWord = "mouse" |
| Sample Output | [[["mobile","moneypot","monitor"],["mobile","moneypot","monitor"],["mouse","mousepad"],["mouse","mousepad"],["mouse","mousepad"]]] |