ASK/VIEW DOUBT | SOLUTION | HINT | Your Code | Main Code | Code Pair | C++ (g++ 5)

Problem | Result

Pattern Matching

Send Feedback

Given a list of n words and a pattern p that we want to search. Check if the pattern p is present the given words or not.

Return true or false.

Input Format :

Line 1 : Integer n
Line 2 : n words (separated by space)
Line 3 : Pattern p (a string)

Output Format :

true ot false

Sample Input 1 :

4
abc def ghi cba
de

PREVIOUS | NEXT | CUSTOM INPUT | SUBMIT SOLUTION

```cpp
// #include "TrieNode.h"
#include <string>
#include <vector>
class TrieNode {
    public :
    char data;
    TrieNode **children;
    bool isTerminal;

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

class Trie {
    TrieNode *root;

    public :
    int count;

    Trie() {
        this->count = 0;
        root = new TrieNode('\0');
    }

    bool insertWord(TrieNode *root, string word) {
        // Base case
        if(word.size() == 0) {
            if (!root->isTerminal) {
                root -> isTerminal = true;
                return true;
            } else {
                return false;
            }
        }

        // Small Calculation
        int index = word[0] - 'a';
        TrieNode *child;
        if(root -> children[index] != NULL) {
            child = root -> children[index];
        }
        else {
            child = new TrieNode(word[0]);
            root -> children[index] = child;
        }

        // Recursive call
        return insertWord(child, word.substr(1));
    }

    // For user
    void insertWord(string word) {

        for(int i=0;i<word.size();i++){
            insertWord(root, word.substr(i));
        }
        // if (insertWord(root, word)) {
        //  this->count++;
        // }
    }

    bool searchHelp(TrieNode *root, string word){
        if(word.size() == 0) {
            // return root -> isTerminal;
            return true;
        }
        int index = word[0] - 'a';
        if(root -> children[index] != NULL) {
            return searchHelp(root->children[index],word.substr(1));
        }
        else {
            return false;
        }
    }


    bool search(string word) {
        // Write your code here
        return searchHelp(root,word);
    }

    bool patternMatching(vector<string> vect, string pattern) {
        // Complete this function
        // Return true or false
        for(int i=0;i<vect.size();i++){
            insertWord(vect[i]);
        }
        return search(pattern);
    }
};
```

CUSTOM INPUT | SUBMIT SOLUTION