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

Problem | Result

Palindrome Pair

Send Feedback

Given 'n' number of words, you need to find if there exist any two words which can be joined to make a palindrome or any word itself is a palindrome.

The function should return either true or false. You don't need to print anything.

Input Format :

The first line of the test case contains an integer value denoting 'n'.

The second line of the test case will contain 'n' number of words each separated by a single space.

Output Format :

Print either true or false

Sample Input 1 :

PREVIOUS | NEXT

CUSTOM INPUT | SUBMIT SOLUTION

```
// #include "TrieNode.h"
#include <string>

/**************
 * Main function -

int main() {
    int choice;
    cin >> choice;
    Trie t;

    //cout << "asasas";
    while(choice != -1){
        string word;
        bool ans;
        switch(choice) {
            case 1 : // insert
                // getline(cin, word);
                cin >> word;
                t.insertWord(word);
                break;
            case 2 : // search
                // getline(cin, word);
                cin >> word;
                ans = t.search(word);
                if (ans) {
                    cout << "true" << endl;
                } else {
                    cout << "false" << endl;
                }
                break;
            default :
                return 0;
        }
        cin >> choice;
    }
    return 0;
}
```

CUSTOM INPUT | SUBMIT SOLUTION

```
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 :

    Trie() {
        root = new TrieNode('\0');
    }

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

CUSTOM INPUT | SUBMIT SOLUTION

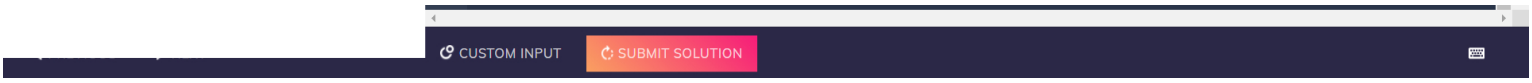
```
        if(root -> children[index] != NULL) {
            child = root -> children[index];
        }
        else {
            child = new TrieNode(word[0]);
            root -> children[index] = child;
        }

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

    // For user
    void insertWord(string word) {
        insertWord(root, word);
    }

    bool searchHelp(TrieNode *root, string word){
        if(word.size() == 0) {
            return root -> isTerminal;
        }
        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);
    }
};
```


CUSTOM INPUT
SUBMIT SOLUTION