

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 or false

Sample Input 1 :

4
abc def ghi cba
de

```
1 // #include "TrieNode.h"
2 #include <string>
3 #include <vector>
4 class TrieNode {
5 public :
6     char data;
7     TrieNode **children;
8     bool isTerminal;
9
10     TrieNode(char data) {
11         this->data = data;
12         children = new TrieNode*[26];
13         for(int i = 0; i < 26; i++) {
14             children[i] = NULL;
15         }
16         isTerminal = false;
17     }
18 };
19
20 class Trie {
21     TrieNode *root;
22
23 public :
24     int count;
25
26     Trie() {
27         this->count = 0;
28         root = new TrieNode('\0');
29     }
30
31     bool insertWord(TrieNode *root, string word) {
32         // Base case
33         if(word.size() == 0) {
34             if (!root->isTerminal) {
35                 root->isTerminal = true;
36                 return true;
37             } else {
38                 return false;
39             }
40         }
41
42         int index = word[0] - 'a';
43         TrieNode *child;
44         if(root->children[index] != NULL) {
45             child = root->children[index];
46         } else {
47             child = new TrieNode(word[0]);
48             root->children[index] = child;
49         }
50
51         // Recursive call
52         return insertWord(child, word.substr(1));
53     }
54
55     // For user
56     void insertWord(string word) {
57         for(int i=0;i<word.size();i++){
58             insertWord(root, word.substr(i));
59         }
60         // if (insertWord(root, word)) {
61         //     this->count++;
62         // }
63     }
64
65     bool searchHelp(TrieNode *root, string word){
66         if(word.size() == 0) {
67             // return root->isTerminal;
68             return true;
69         }
70         int index = word[0] - 'a';
71         if(root->children[index] != NULL) {
72             return searchHelp(root->children[index],word.substr(1));
73         } else {
74             return false;
75         }
76     }
77
78     bool search(string word) {
79         // Write your code here
80         return searchHelp(root,word);
81     }
82
83     bool patternMatching(vector<string> vect, string pattern) {
84         // Complete this function
85         // Return true or false
86         for(int i=0;i<vect.size();i++){
87             insertWord(vect[i]);
88         }
89         return search(pattern);
90     }
91 };
92
93
94
95
96
97
98
99
100
```

```
42 // Small Calculation
43 int index = word[0] - 'a';
44 TrieNode *child;
45 if(root->children[index] != NULL) {
46     child = root->children[index];
47 } else {
48     child = new TrieNode(word[0]);
49     root->children[index] = child;
50 }
51
52 // Recursive call
53 return insertWord(child, word.substr(1));
54 }
55
56 // For user
57 void insertWord(string word) {
58     for(int i=0;i<word.size();i++){
59         insertWord(root, word.substr(i));
60     }
61     // if (insertWord(root, word)) {
62     //     this->count++;
63     // }
64 }
65
66 bool searchHelp(TrieNode *root, string word){
67     if(word.size() == 0) {
68         // return root->isTerminal;
69         return true;
70     }
71     int index = word[0] - 'a';
72     if(root->children[index] != NULL) {
73         return searchHelp(root->children[index],word.substr(1));
74     } else {
75         return false;
76     }
77 }
78
79 bool search(string word) {
80     // Write your code here
81     return searchHelp(root,word);
82 }
83
84 bool patternMatching(vector<string> vect, string pattern) {
85     // Complete this function
86     // Return true or false
87     for(int i=0;i<vect.size();i++){
88         insertWord(vect[i]);
89     }
90     return search(pattern);
91 }
92
93
94
95
96
97
98
99
100
```