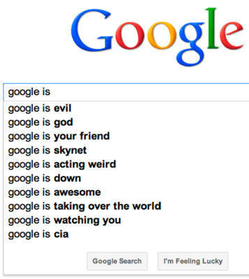
Autocomplete Suggestions

locked

Problem | Submissions | Leaderboard | Discussions



Submissions: 115
Max Score: 100
Difficulty: Medium

Rate This Challenge:

More

Трябва да имплементирате една от най-важните функции на търсачките - **Autocomplete**. Например когато пишете думи в Google, техният Autocomplete ви предлага някакви възможности, базирани на написаното до момента.

За целта ви е даден речник с **N** на брой популярни думи - знаете, че тези думи са често търсени от потребителите и съответно трябва да ги предлагате, когато е възможно. Напишете програма, която да може бързо да намира по дадено начало на дума колко от думите от речника я autocomplete-ват (т.е на колко думи от речника е префикс).

Ще трябва да отговаряте на **Q** на брой заявки. Всяка заявка се състои от един низ - начало на дума. Програмата ви трябва да намери колко от думите в речника започват с дадения префикс.

Input Format

Първия ред на входа съдържа числата **N** и **Q** - съответно броят на думите в речника и броят на заявките.

На следващия ред има **N** низа - всички думи от речника.

Следват **Q** реда с по един низ - префиксът за който се търсят броят думи, които го autocomplete-ват.

Constraints

1 ≤ N ≤ 200 000

1 ≤ Q ≤ 200 000

1 ≤ дължина на дума от речника ≤ 20

1 ≤ дължина на префикс ≤ 20

Думите и префиксите се състоят само от малки латински букви.

Output Format

Изведете **Q** реда с по едно число за всеки префикс - съответно броят думи от речника, които го autocomplete-ват.

Sample Input 0

```
4 5
baba banica boza ba
b
ba
ban
bi
a
```

Sample Output 0

```
4
3
1
0
0
```

Current Buffer (saved locally, editable) C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include<string>
using namespace std;
const int ALPHABET_SIZE=26;
struct TrieNode{
  TrieNode *children[ALPHABET_SIZE];
  bool isWordEnd;
  int prefixes;
  TrieNode(){
    isWordEnd=false;
    prefixes=0;
    for(int i=0;i<ALPHABET_SIZE;i++){
      children[i]=nullptr;
    }

  }

  void insert(TrieNode *root,string key)
{
  TrieNode *current=root;
  for(int i=0;i<key.size();i++)
  {
    TrieNode *node=current->children[key[i]-'a'];
    if(!node)
    {
      node=new TrieNode();
      current->children[key[i]-'a']=node;
    }
    current=node;
    current->prefixes++;
  }
  current->isWordEnd=true;
}
  int autocompleteOptions(TrieNode *root,string key)
{
  TrieNode *current=root;
  for(int i=0;i<key.size();i++)
  {

    TrieNode *node=current->children[key[i]-'a'];
    if(!node)
    {
      return 0;
    }
    current=node;
  }
  return current->prefixes;
}
};

int main() {
  int wordCount;
  int queriesCount;
  cin>>wordCount;
  cin>>queriesCount;
  string word;
  TrieNode* root=new TrieNode();
  for(int i=0;i<wordCount;i++){
    cin>>word;
    root->insert(root,word);
  }
  string query;
   for(int i=0;i<queriesCount;i++){
     cin>>query;
     cout<<root->autocompleteOptions(root,query)<<endl;
   }
    return 0;
}

```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code