Code: Search word in Trie

Send Feedback

Implement the function SearchWord for the Trie class.

For a Trie, write the function for searching a word. Return true if the word is found successfully, otherwise return false.

Note : main function is given for your reference which we are using internally to test the code.

```
class TrieNode{


	char data;

	boolean isTerminating;

	TrieNode children[];

	int childCount;


	public TrieNode(char data) {

		this.data = data;

		isTerminating = false;

		children = new TrieNode[26];

		childCount = 0;

	}

}



public class Trie {


	private TrieNode root;

	public int count;

	public Trie() {

		root = new TrieNode('\0');

	}


	public boolean search(String word){
```

```
                // Write your code here
    return search(root,word);
        }
private boolean search(TrieNode root,String word){
    if(word.length()==0){
        if(root.isTerminating){
            return true;
        }
        return false;
    }
    int childIndex = word.charAt(0)-'a';
    TrieNode child = root.children[childIndex];
    if(child==null){
        return false;
    }
    return search(child,word.substring(1));

}



        private void add(TrieNode root, String word){
                if(word.length() == 0){
                        root.isTerminating = true;
                        return;
                }
                int childIndex = word.charAt(0) - 'a';
                TrieNode child = root.children[childIndex];
                if(child == null){
                        child = new TrieNode(word.charAt(0));
```

```
                root.children[childIndex] = child;
                root.childCount++;
            }
            add(child, word.substring(1));

        }

        public void add(String word){
            add(root, word);
        }
}
```

Count Words in Trie

Send Feedback

You are given the Trie class with following functions -

1. insertWord
2. removeWord

Now, you need to create one more function (named "countWords") which returns the number of words currently present in Trie in O(1) time complexity.

Note : You can change the already given functions in the Trie class, if required.

```
class TrieNode{
    char data;
    boolean isTerminating;
    TrieNode children[];
    int childCount;

    public TrieNode(char data) {
        this.data = data;
        isTerminating = false;
        children = new TrieNode[26];
        childCount = 0;
    }
}
```

```
public class Trie {

	private TrieNode root;
	private int numWords;


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


	public void remove(String word){
		if(remove(root, word)) {
			numWords--;
		}
	}


	private boolean remove(TrieNode root, String word) {
		if(word.length() == 0){
			if(root.isTerminating) {
				root.isTerminating = false;
				return true;
			}
			else {
				return false;
			}
```

```
        }
        int childIndex = word.charAt(0) - 'a';
        TrieNode child = root.children[childIndex];
        if(child == null){
            return false;
        }
        boolean ans = remove(child, word.substring(1));
        // We can remove child node only if it is non terminating and its number of children are
0

        if(!child.isTerminating && child.childCount == 0){
            root.children[childIndex] = null;
            child = null;
            root.childCount--;
        }
        return ans;
    }

    private boolean add(TrieNode root, String word){
        if(word.length() == 0){
            if(root.isTerminating) {
                return false;
            }
            else {
                root.isTerminating = true;
                return true;
            }
        }
        int childIndex = word.charAt(0) - 'a';
        TrieNode child = root.children[childIndex];
```

```
            if(child == null){
                child = new TrieNode(word.charAt(0));
                root.children[childIndex] = child;
                root.childCount++;
            }
            return add(child, word.substring(1));
        }

        public void add(String word){
            if(add(root, word)) {
                numWords++;
            }
        }

        public int countWords() {
            // Write your code here
    return numWords;
        }

}
```

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 if the pattern is present and false otherwise.

Input Format :

The first line of input contains an integer, that denotes the value of n.
The following line contains n space separated words.
The following line contains a string, that denotes the value of the pattern p.

Output Format :

The first and only line of output contains true if the pattern is present and false otherwise.

Constraints:

Time Limit: 1 sec

Sample Input 1 :

```
4
abc def ghi cba
de
```

Sample Output 2 :

```
true
```

Sample Input 2 :

```
4
abc def ghi hg
hi
```

Sample Output 2 :

```
true
```

Sample Input 3 :

```
4
abc def ghi hg
hif
```

Sample Output 3 :

```
false
```

```
import java.util.ArrayList;


class TrieNode{

	char data;
	boolean isTerminating;
	TrieNode children[];
	int childCount;

	public TrieNode(char data) {
		this.data = data;
		isTerminating = false;
		children = new TrieNode[26];
		childCount = 0;
	}
}


public class Trie {

	private TrieNode root;
	public int count;
	public Trie() {
```

```
                root = new TrieNode('\0');
        }
private void add(TrieNode root, String word){
   if(word.length() == 0){
      root.isTerminating = true;
      return;
   }
   int childIndex = word.charAt(0) - 'a';
   TrieNode child = root.children[childIndex];
   if(child == null){
      child = new TrieNode(word.charAt(0));
      root.children[childIndex] = child;
      root.childCount++;
   }
   add(child, word.substring(1));
}

public void add(String word){
   add(root, word);
}



        public boolean search(String word){
                return search(root, word);
        }

        private boolean search(TrieNode root, String word) {
                if(word.length() == 0){
                        return true;
```

```
                }
                int childIndex = word.charAt(0) - 'a';
                TrieNode child = root.children[childIndex];
                if(child == null){
                        return false;
                }
                return search(child, word.substring(1));
        }


        public boolean patternMatching(ArrayList<String> vect, String pattern) {
    // Write your code here
    for (int i = 0; i < vect.size(); i++) {
    String word = vect.get(i);
    for (int j = 0; j < word.length(); j++) {
       add(word.substring(j)); }}
    // for(int i=0;i<input.size();i++)
    // {
    //  int j=1;
    //  while(j<input.get(i).length()){
    //     add(pattern.substring(j));
    //     j++;
    //  }
    // }
    return searchHelper(root,pattern);
}
private boolean searchHelper(TrieNode root,String pattern){
    if(pattern.length()==0)
       return true;
```

```
        int childIndex=pattern.charAt(0)-'a';
        TrieNode child=root.children[childIndex];
        if(child==null)
            return false;
        return searchHelper(child,pattern.substring(1));
    }
}
```

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, which itself is a palindrome.
The function should return either true or false. You don't have to print anything.

Input Format :

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

The following contains 'n' number of words each separated by a single space.

Output Format :

The first and only line of output contains true if the conditions described in the task are met and false otherwise.

Constraints:

0 <= n <= 10^5
Time Limit: 1 sec

Sample Input 1 :

```
4
abc def ghi cba
```

Sample Output 1 :

```
true
```

Explanation of Sample Input 1:

"abc" and "cba" forms a palindrome

Sample Input 2 :

```
2
abc def
```

Sample Output 2 :

```
false
```

```
import java.util.ArrayList;


class TrieNode {
        char data;
        boolean isTerminating;
        TrieNode children[];
        int childCount;
```

```
        public TrieNode(char data) {
                this.data = data;
                isTerminating = false;
                children = new TrieNode[26];
                childCount = 0;
        }
}

public class Trie {

        private TrieNode root;
        public int count;

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

        private void add(TrieNode root, String word){
                if(word.length() == 0){
                        root.isTerminating = true;
                        return;
                }

                int childIndex = word.charAt(0) - 'a';
                TrieNode child = root.children[childIndex];

                if(child == null) {
                        child = new TrieNode(word.charAt(0));
                        root.children[childIndex] = child;
```

```
                root.childCount++;
        }

        add(child, word.substring(1));
}

public void add(String word){
        add(root, word);
}

private boolean search(TrieNode root, String word) {
        if(word.length() == 0) {
                return true;
        }

        int childIndex=word.charAt(0) - 'a';
        TrieNode child=root.children[childIndex];

        if(child == null) {
                return false;
        }

        return search(child,word.substring(1));

}

public boolean search(String word) {
        return search(root,word);
}
```

```
	private void print(TrieNode root, String word) {
		if (root == null) {
			return;
		}

		if (root.isTerminating) {
			System.out.println(word);
		}

		for (TrieNode child : root.children) {
			if (child == null) {
				continue;
			}
			String fwd = word + child.data;
			print(child, fwd);
		}
	}

	public void print() {
		print(this.root, "");
	}

	/*..... Palindrome Pair..... */
public String reverse(String word) {
```

```
        String xString="";
        for(int i=word.length()-1;i>=0;i--) {
                xString+=word.charAt(i);
        }
        return xString;

}
public boolean isPalindromePair(ArrayList<String> words) {

        for(int i=0;i<words.size();i++) {

                String string = reverse(words.get(i));

                Trie suffixTrie = new Trie();
                for(int j=0;j<string.length();j++) {
                        suffixTrie.add(string.substring(j));
                }
                for(String word : words) {
                        if(suffixTrie.search(word)) {
                                return true;
                        }
                }

        }
        return false;

}
```

}

Auto complete

Send Feedback

Given n number of words and an incomplete word w. You need to auto-complete that word w.

That means, find and print all the possible words which can be formed using the incomplete word w.

Note : Order of words does not matter.

Input Format :

The first line of input contains an integer, that denotes the value of n.
The following line contains n space separated words.
The following line contains the word w, that has to be auto-completed.

Output Format :

Print all possible words in separate lines.

Constraints:

Time Limit: 1 sec

Sample Input 1 :

```
7
do dont no not note notes den
no
```

Sample Output 2 :

```
no
not
note
notes
```

Sample Input 2 :

```
7
do dont no not note notes den
de
```

Sample Output 2 :

```
den
```

Sample Input 3 :

```
7
do dont no not note notes den
nom
```

```
import java.util.ArrayList;



class TrieNode{

	char data;

	boolean isTerminating;

	TrieNode children[];

	int childCount;



	public TrieNode(char data) {

		this.data = data;
```

```
                isTerminating = false;
                children = new TrieNode[26];
                childCount = 0;
        }
}

public class Trie {
        private TrieNode root;
        public int count;
        public Trie() {
                root = new TrieNode('\0');
        }

        private void add(TrieNode root, String word){
                if(word.length() == 0){
                        root.isTerminating = true;
                        return;
                }
                int childIndex = word.charAt(0) - 'a';
                TrieNode child = root.children[childIndex];
                if(child == null){
                        child = new TrieNode(word.charAt(0));
                        root.children[childIndex] = child;
                        root.childCount++;
                }
                add(child, word.substring(1));
        }

        public void add(String word){
```

```
                add(root, word);
        }
public static void print(TrieNode root,String output)
        {
                if(root == null)
                        return;
                output+=root.data;
                if(root.isTerminating)
                        System.out.println(output);
                for(int i=0;i<26;i++)
                {
                        print(root.children[i],output);
                }
        }


        public void autoComplete(ArrayList<String> input, String word) {

   // Write your code here
   for(int i=0;i<input.size();i++)
   {
      add(input.get(i));
   }

   int flag = 0;
   TrieNode temp = root;
   int index = 0;
   TrieNode outer = root;
   for(int i=0;i<word.length();i++)
```

```
        {
            index = word.charAt(i) - 'a';
            if(temp.children[index]!=null)
            {
                    outer = temp;
                    temp = temp.children[index];
            }
            else
            {
                    flag = 1;
                    break;
            }
        }
        if(flag == 1)
            return;
        index = word.charAt(0)-'a';
        print(temp,word.substring(0,word.length()-1));

    }

}
```