**Implement Phone directory**

**Hard**Accuracy: 38.32% Submissions: 5492 Points: 8

Given a list of contacts **contact[ ]** of length **n** where each contact is a string which exist in a phone directory and a query string **s**. The task is to implement a search query for the phone directory. Run a search query for each prefix **p** of the query string **s**(*i.e.* from  index 1 to |s|) that prints all the distinct contacts which have the same prefix as p in **lexicographical increasing order**. Please refer the explanation part for better understanding.  
**Note:**If there is no match between query and contacts, print "0".

**Example 1:**

**Input:**

n = 3

contact[] = {"geeikistest", "geeksforgeeks",

"geeksfortest"}

s = "geeips"

**Output:**

geeikistest geeksforgeeks geeksfortest

geeikistest geeksforgeeks geeksfortest

geeikistest geeksforgeeks geeksfortest

geeikistest

0

0

**Explaination:** By running the search query on

contact list for "g" we get: "geeikistest",

"geeksforgeeks" and "geeksfortest".

By running the search query on contact list

for "ge" we get: "geeikistest" "geeksforgeeks"

and "geeksfortest".

By running the search query on contact list

for "gee" we get: "geeikistest" "geeksforgeeks"

and "geeksfortest".

By running the search query on contact list

for "geei" we get: "geeikistest".

No results found for "geeip", so print "0".

No results found for "geeips", so print "0".

**Your Task:**  
Youd do not need to read input or print anything. Your task is to complete the function **displayContacts()** which takes **n, contact[ ]**and**s** as input parameters and returns a list of list of strings for required prefixes. If some prefix has no matching contact return "0" on that list.

**Expected Time Complexity:** O(|s| \* n \* max|contact[i]|)  
**Expected Auxiliary Space:** O(n \* max|contact[i]|)

**Constraints:**  
1 ≤ n ≤ 50  
1 ≤ |contact[ i ]| ≤ 50  
1 ≤ |s| ≤ 6

class TrieNode {

    public:

    TrieNode\* children[26];

    bool isLeaf;

    TrieNode() {

        this->isLeaf=false;

        for (int i=0; i<26; i++) {

            this->children[i]=NULL;

        }

    }

};

void insert(TrieNode\* root, string s) {

    TrieNode\* p=root;

    for (int i=0; i<s.length(); i++) {

        int index=s[i]-'a';

        if (p->children[index]) {

            p=p->children[index];

        }

        else {

            TrieNode\* newNode=new TrieNode();

            p->children[index]=newNode;

            p=newNode;

        }

    }

    p->isLeaf=true;

}

void printList(int x, vector<string> &aux, TrieNode\* n, string s) {

    if (n->isLeaf) {

        aux.push\_back(s);

    }

    for (int i=0; i<26; i++) {

        if (n->children[i]) {

            char c=i+'a';

            printList(i, aux, n->children[i], s+c);

        }

    }

}

class Solution{

public:

    vector<vector<string>> displayContacts(int len, string contact[], string s) {

        // code here

        TrieNode\* root=new TrieNode();

        vector<vector<string>> ans;

        for (int i=0; i<len; i++) {

            insert(root, contact[i]);

        }

        string prefix="";

        TrieNode\* n=root;

        bool flag=true;

        for (int i=0; i<s.length(); i++) {

            prefix+=s[i];

            int index=s[i]-'a';

            vector<string> aux;

            if (flag and n->children[index]) {

                n=n->children[index];

                printList(index, aux, n, prefix);

                sort(aux.begin(), aux.end());

                ans.push\_back(aux);

            }

            else {

                flag=false;

                ans.push\_back({"0"});

            }

        }

        return ans;

    }

};