Video Conference

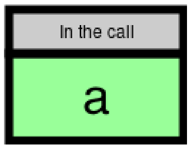
locked

by Shafaet

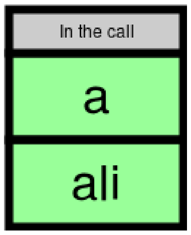
Bob is making a video conference software. Whenever a new person joins the conference, Bob displays the person's name in the interface.

However, displaying full name is tedious and takes much space. So he decided to display the shortest prefix which doesn't match with any prefix of any person who has joined earlier.

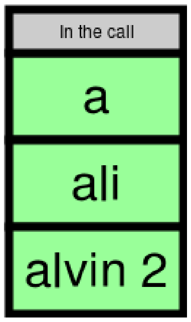
Let's suppose the first person to enter the conference is `alvin`.



Now suppose next person to join is `alice`. The shortest prefix of `alice` that doesn't match with any prefix of `alvin` is `ali`.



If the full name of a new person matches completely with the full name of any person who has joined earlier, he will display the full name and add a suffix which indicates how many times the same name has occurred in the list so far. For example, if another person name `alvin` joins, the list will look like this:



You are given the list of the persons who have joined the call in the chronological order. Your task is to figure out how the final list looks like.

Input Format

The first line contains an integer n.

The subsequent n line contains a string s_i denoting the name of the i^{th} person to join the call.

Constraints

- $1 \le n \le 10^5$
- $1 \le s_i \le 10$
- s_i will contain only lower-case english letters.

Subtask

- $1 \le n \le 1000$ for 60% of the maximum score

Output Format

Return a string array with n items, the i^{th} line should contain the prefix of name of the i^{th} person which doesn't match with any other person who has joined earlier.

Sample Input 0

```
3
alvin
alice
alvin
```

Sample Output 0

```
a
ali
alvin 2
```

Sample Input 1

```
6
mary
stacy
sam
samuel
sam
miguel
```

Sample Output 1

```
m
s
sa
samu
sam 2
mi
```

Submissions: 994
Max Score: 15
Difficulty: Easy

Rate This Challenge:
★★★★☆ Thanks!

More

Current Buffer (saved locally, editable)

Rust

```
use std::io::{self, Read};
use std::collections::HashMap;

#[derive(Clone, Copy, Debug)]
struct Trie {
    k: [usize;26],
    count: u32,
    prev: isize,
    letter: u8,
}

impl Default for Trie {
    fn default() -> Self {
        Trie { k: [ std::usize::MAX as usize; 26 ], count: 0, prev: -1, letter: 0xff }
    }
}

fn main() {
    let mut buffer = String::new();
    let stdin = io::stdin();
    let mut handle = stdin.lock();

    let mut idx_next = 1usize;
    let mut order = 1usize;
    let mut b = vec![ Trie::default(); 1_000_001 ];

    handle.read_to_string(&mut buffer).unwrap();

    let mut arr = buffer.split_whitespace().collect::<Vec<_>>();
    let s = arr.iter().skip(1).collect::<Vec<_>>();

    let mut order : Vec<(usize,u32)> = vec![];
    //---

    for i in s.iter() {

        let mut idx = 0;
        let mut added_first_new_node = false;

        for (k,j) in i.chars().enumerate() {

            let c = j as usize - 'a' as usize;

            if b[idx].k[c] != std::usize::MAX {
                //go to the next node that already exists
                idx = b[idx].k[c] as usize;
            } else {
                //add new node
                b[idx].k[c] = idx_next;
                b[idx_next as usize].prev = idx as isize; //save parent idx for backtrace later
                if !added_first_new_node {
                    order.push( ( idx_next as usize, 1 ) ); //save order of arrival
                    added_first_new_node = true;
                }
                idx = idx_next as usize;
                b[idx].letter = c as u8;
                idx_next += 1;
            }
        }
        if idx != 0 {
            b[idx].count += 1;
        }
        if !added_first_new_node {
            //means the name overlaps over a previous person and thus no new node was added
            order.push( ( idx, b[idx].count ) ); //save order of arrival and count
        }
    }

    // println!("{:?}", &b[0..idx_next as usize]);
    // println!("{:?}", order );

    //recover names from order of arrival and backtrace
    for i in order {

        let mut idx = i.0 as isize;
        let count = i.1;

        let mut v = vec![];

        while idx != -1 && idx != 0 {
            v.push( b[idx as usize].letter as u8 + 'a' as u8 );
            idx = b[idx as usize].prev;
        }
        v.reverse();

        use std::str;
        let ss = str::from_utf8(v.as_slice()).unwrap();

        if count > 1 {
            println!("{} {}", ss, count );
        } else {
            println!("{}", ss );
        }
    }
}

```

Line: 1 Col: 1

Upload Code as File

☐ Test against custom input

Run Code

Submit Code