['wrt',] t→f.
'wrf',] w→e
'er', r→t
'ett',
'rftt', e→f.
]

abc
abcd ⇒ ✓

abcd
abc ⇒ ✗

abc
ad

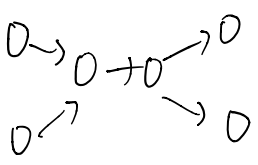
```
const fn = (words) => {
    const graph = {};
    for(let i=0; i<words.length; i++){
        for(const c of words[i]){
            if(!(c in graph)){
                graph[c] = [];
            }
        }
        if(i>=words.length-1){
            break;
        }
    } ← end here for the last word.
    let j = 0;
    while(j<words[i].length &&
          j<words[i+1].length &&
          words[i][j] === words[i+1][j]){
        j+=1;
    }
```

```
        if( j < words[i].length &&
            j >= words[i+1].length ) {
            return '';
        } else if( j < words[i].length &&
                   j < words[i+1].length ) {
            graph[words[i][j]].push(words[i+1][j]);
        }
    }
    const output = new Set();
    const visited = {};
    for( const v in graph) {
        if( hasCycle(graph, v, visited, output)) {
            return '';
        }
    }
    return output;
}
```

output

dfs approach is the reversed version of solution

⇒ [...output].reverse().join('');



```
function hasCycle(graph, v, visited, output) {
    if (v in visited) {
        if (!output.has(v)) {
            return true;
        }
        return false;
    }
    visited[v] = true;
    for (const u of graph[v]) {
        if (hasCycle(graph, u, visited, output)) {
            return true;
        }
    }
    output.push(v);
    return false;
}
```

add => we are using 'Set'