721. Accounts Merge

Given a list `accounts`, each element `accounts[i]` is a list of strings, where the first element `accounts[i][0]` is a *name*, and the rest of the elements are *emails* representing emails of the account.

Now, we would like to merge these accounts. Two accounts definitely belong to the same person if there is some email that is common to both accounts. Note that even if two accounts have the same name, they may belong to different people as people could have the same name. A person can have any number of accounts initially, but all of their accounts definitely have the same name.

After merging the accounts, return the accounts in the following format: the first element of each account is the name, and the rest of the elements are emails **in sorted order**. The accounts themselves can be returned in any order.

Example 1:

```
Input:
accounts = [["John", "johnsmith@mail.com", "john00@mail.com"], ["John",
"johnnybravo@mail.com"], ["John", "johnsmith@mail.com",
"john_newyork@mail.com"], ["Mary", "mary@mail.com"]]
Output: [["John", 'john00@mail.com', 'john_newyork@mail.com',
'johnsmith@mail.com'],  ["John", "johnnybravo@mail.com"], ["Mary",
"mary@mail.com"]]
Explanation:
The first and third John's are the same person as they have the common email
"johnsmith@mail.com".
The second John and Mary are different people as none of their email addresses
are used by other accounts.
We could return these lists in any order, for example the answer [['Mary',
'mary@mail.com'], ['John', 'johnnybravo@mail.com'],
['John', 'john00@mail.com', 'john_newyork@mail.com', 'johnsmith@mail.com']]
would still be accepted.
```

Note:

- The length of `accounts` will be in the range `[1, 1000]`.
- The length of `accounts[i]` will be in the range `[1, 10]`.
- The length of `accounts[i][j]` will be in the range `[1, 30]`.

Seen this question in a real interview before?

- Difficulty:Medium
- Total Accepted:1.7K
- Total Submissions:7.3K
- Contributor: awice
-

- Subscribe to see which companies asked this question.

Related Topics : Depth_Frist search, Union Find

We give each account an ID, based on the index of it within the list of accounts.

```
[
["John", "johnsmith@mail.com", "john00@mail.com"], # Account 0
["John", "johnnybravo@mail.com"], # Account 1
["John", "johnsmith@mail.com", "john_newyork@mail.com"],  # Account 2
["Mary", "mary@mail.com"] # Account 3
]
```

Next, build an `emails_accounts_map` that maps an email to a list of accounts, which can be used to track which email is linked to which account. This is essentially our graph.

```
# emails_accounts_map of email to account ID
{
  "johnsmith@mail.com": [0, 2],
  "john00@mail.com": [0],
  "johnnybravo@mail.com": [1],
  "john_newyork@mail.com": [2],
  "mary@mail.com": [3]
}
```

python 2.7

```
class Solution(object):
    def accountsMerge(self, accounts):
        from collections import defaultdict
        visited_accounts = [False] * len(accounts)
        emails_accounts_map = defaultdict(list)
        res = []
        # Build up the graph.
        for i, account in enumerate(accounts):
            for j in range(1, len(account)):
                email = account[j]
                emails_accounts_map[email].append(i)
        # DFS code for traversing accounts.
        def dfs(i, emails):
            if visited_accounts[i]:
                return
            visited_accounts[i] = True
            for j in range(1, len(accounts[i])):
                email = accounts[i][j]
                emails.add(email)
                for neighbor in emails_accounts_map[email]:
                    dfs(neighbor, emails)
        # Perform DFS for accounts and add to results.
        for i, account in enumerate(accounts):
            if visited_accounts[i]:
                continue
            name, emails = account[0], set()
            dfs(i, emails)
            res.append([name] + sorted(emails))
```

```
        return res
```

C++

```
// author: MR.Black
#include<stdio.h>
#include<iostream>
#include<vector>
#include<algorithm>
#include<unordered_map>
#include<unordered_set>
using namespace std;
// compile usage g++ -Wall -o "accountsMerge" "accountsMerge.cpp" -std=c++11
void dfs(int index,unordered_set<string> &emails,vector<bool> &visited,
           unordered_map<string,vector<int>> &account_map,vector<vector<string>>& accounts)
{
     if (visited[index])
          return;
     visited[index]=true;
     for(int i=1;i<(int)accounts[index].size();i++)
     {
          string emailstr = accounts[index][i];
          emails.insert(emailstr);
          vector<int> neighbors = account_map[emailstr];
          for(auto nei:neighbors)
          {
               dfs(nei,emails,visited,account_map,accounts);
          }
     }
}

vector<string> sortEmailSet(unordered_set<string> &set)
{
     vector<string> res;
     for(auto it:set)
     {
```

```
            res.push_back(it);
        }
        sort(res.begin(),res.end());
        return res;
}

vector<vector<string>> accountsMerge(vector<vector<string>>& accounts){
        int n = accounts.size();vector<vector<string>> res;
        vector<bool>visited(n,false);
        unordered_map<string,vector<int>> account_map;
        for(int i=0;i<n;i++)
        {
            for(int j=1;j<(int)accounts[i].size();j++)
            {
                string email = accounts[i][j];
                account_map[email].push_back(i);
            }
        }
        /*
        for(auto elem:account_map)
        {
            cout<<elem.first<<"   ";
            auto tmp = elem.second;
            for(int i:tmp) cout << i << " ";
            printf("\n");
        }
        */
        for(int i=0;i<(int)accounts.size();++i)
        {
            if(visited[i]) continue;
            unordered_set<string> emails;
            dfs(i,emails,visited,account_map,accounts);
            //name = account[i][0]
            vector<string> restmp;
            restmp.push_back(accounts[i][0]);
            vector<string> sortedEmail = sortEmailSet(emails);
```

```
            for(int k=0;k<(int)sortedEmail.size();++k)
restmp.push_back(sortedEmail[k]);
            res.push_back(restmp);
      }
      return res;
}
int main(int argc, char *argv[])
{
      vector<vector<string>> accounts = {{"John", "johnsmith@mail.com",
"john00@mail.com"}
            , {"John", "johnnybravo@mail.com"}, {"John",
"johnsmith@mail.com", "john_newyork@mail.com"},
            {"Mary", "mary@mail.com"}};
      vector<vector<string>> ans = accountsMerge(accounts);
      for(int i=0;i<(int)ans.size();++i)
      {
            for(auto elem:ans[i]) printf("%s ",elem.c_str());
            printf("\n");
      }
      return 0;
}
```