|  |  |
| --- | --- |
| **AccountMerge in C++** | |
| #include <bits/stdc++.h>  using namespace std;  //User function Template for C++  class DisjointSet {  vector<int> rank, parent, size;  public:  DisjointSet(int n) {  rank.resize(n + 1, 0);  parent.resize(n + 1);  size.resize(n + 1);  for (int i = 0; i <= n; i++) {  parent[i] = i;  size[i] = 1;  }  }  int findUPar(int node) {  if (node == parent[node])  return node;  return parent[node] = findUPar(parent[node]);  }  void unionByRank(int u, int v) {  int ulp\_u = findUPar(u);  int ulp\_v = findUPar(v);  if (ulp\_u == ulp\_v) return;  if (rank[ulp\_u] < rank[ulp\_v]) {  parent[ulp\_u] = ulp\_v;  }  else if (rank[ulp\_v] < rank[ulp\_u]) {  parent[ulp\_v] = ulp\_u;  }  else {  parent[ulp\_v] = ulp\_u;  rank[ulp\_u]++;  }  }  void unionBySize(int u, int v) {  int ulp\_u = findUPar(u);  int ulp\_v = findUPar(v);  if (ulp\_u == ulp\_v) return;  if (size[ulp\_u] < size[ulp\_v]) {  parent[ulp\_u] = ulp\_v;  size[ulp\_v] += size[ulp\_u];  }  else {  parent[ulp\_v] = ulp\_u;  size[ulp\_u] += size[ulp\_v];  }  }  };  class Solution {  public:  vector<vector<string>> accountsMerge(vector<vector<string>> &details) {  int n = details.size();  DisjointSet ds(n);  sort(details.begin(), details.end());  unordered\_map<string, int> mapMailNode;  for (int i = 0; i < n; i++) {  for (int j = 1; j < details[i].size(); j++) {  string mail = details[i][j];  if (mapMailNode.find(mail) == mapMailNode.end()) {  mapMailNode[mail] = i;  }  else {  ds.unionBySize(i, mapMailNode[mail]);  }  }  }  vector<string> mergedMail[n];  for (auto it : mapMailNode) {  string mail = it.first;  int node = ds.findUPar(it.second);  mergedMail[node].push\_back(mail);  }  vector<vector<string>> ans;  for (int i = 0; i < n; i++) {  if (mergedMail[i].size() == 0) continue;  sort(mergedMail[i].begin(), mergedMail[i].end());  vector<string> temp;  temp.push\_back(details[i][0]);  for (auto it : mergedMail[i]) {  temp.push\_back(it);  }  ans.push\_back(temp);  }  sort(ans.begin(), ans.end());  return ans;  }  };  int main() {  vector<vector<string>> accounts = {{"John", "j1@com", "j2@com", "j3@com"},  {"John", "j4@com"},  {"Raj", "r1@com", "r2@com"},  {"John", "j1@com", "j5@com"},  {"Raj", "r2@com", "r3@com"},  {"Mary", "m1@com"}  };  Solution obj;  vector<vector<string>> ans = obj.accountsMerge(accounts);  for (auto acc : ans) {  cout << acc[0] << ":";  int size = acc.size();  for (int i = 1; i < size; i++) {  cout << acc[i] << " ";  }  cout << endl;  }  return 0;  } | Dry Run: Let's dry run the algorithm with the input:  vector<vector<string>> accounts = {  {"John", "j1@com", "j2@com", "j3@com"},  {"John", "j4@com"},  {"Raj", "r1@com", "r2@com"},  {"John", "j1@com", "j5@com"},  {"Raj", "r2@com", "r3@com"},  {"Mary", "m1@com"}  }; Step 1: Initialize Disjoint Set:  * Rank array: [0, 0, 0, 0, 0, 0, 0] * Parent array: [0, 1, 2, 3, 4, 5, 6] * Size array: [1, 1, 1, 1, 1, 1, 1]  Step 2: Loop through the accounts:Account 1: {"John", "j1@com", "j2@com", "j3@com"}  * For j1@com, map it to account 0. * For j2@com, map it to account 0. * For j3@com, map it to account 0.  Account 2: {"John", "j4@com"}  * For j4@com, map it to account 1.  Account 3: {"Raj", "r1@com", "r2@com"}  * For r1@com, map it to account 2. * For r2@com, map it to account 2.  Account 4: {"John", "j1@com", "j5@com"}  * For j1@com, it already maps to account 0. **Union** account 3 and 0. * For j5@com, map it to account 3.  Account 5: {"Raj", "r2@com", "r3@com"}  * For r2@com, it already maps to account 2. **Union** account 4 and 2. * For r3@com, map it to account 4.  Account 6: {"Mary", "m1@com"}  * For m1@com, map it to account 5.  Step 3: Union-find operations:  * Union operations are performed for common emails. For example:   + j1@com in Account 1 and Account 4, so **union** Account 0 and Account 3.   + r2@com in Account 3 and Account 4, so **union** Account 2 and Account 4.   After performing all unions, the parent array is updated as follows:   * Parent array: [0, 1, 2, 0, 2, 5] * Rank array: [1, 0, 1, 0, 0, 0] * Size array: [4, 1, 3, 1, 2, 1]  Step 4: Group emails by the root parent:  * For each email, find the root parent and group them.   + Group 0: {"j1@com", "j2@com", "j3@com", "j5@com"}   + Group 2: {"r1@com", "r2@com", "r3@com"}   + Group 5: {"m1@com"}   + Group 1: {"j4@com"}  Step 5: Sort and return:  * Sort each group of emails. * Sort the result by the names (account names). |
| **Output:-**  John:j1@com j2@com j3@com j5@com  John:j4@com  Mary:m1@com  Raj:r1@com r2@com r3@com | |