|  |  |
| --- | --- |
| **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;  } | **Input**  {  {"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"}  }  Let’s assume these are indexed from 0 to 5.  **🧩 Step 1: Mapping Emails to Accounts with Union**  We initialize a map mail → nodeIndex. As we traverse, if we see a repeated email, we perform **unionBySize** between the current index and the one in the map.   | **Index** | **Account Name** | **Emails** | **Action** | | --- | --- | --- | --- | | 0 | John | j1, j2, j3 | Add all emails to map → j1 → 0, j2 → 0, j3 → 0 | | 1 | John | j4 | j4 → 1 | | 2 | Raj | r1, r2 | r1 → 2, r2 → 2 | | 3 | John | j1 (seen), j5 | Union(3, 0) since j1 → 0 → 3 belongs to same group as 0 | | 4 | Raj | r2 (seen), r3 | Union(4, 2) since r2 → 2 → 4 belongs to same group as 2 | | 5 | Mary | m1 | m1 → 5 |   📌 After unions:   * Group 0 includes index 0 and 3 (due to shared j1) * Group 2 includes index 2 and 4 (due to shared r2)   **🔁 Step 2: Group Emails Based on Ultimate Parent (Union-Find)**  We iterate over the map and collect emails in the list mergedMail[parent].  Example:   * j1 → 0 → findUPar(0) = 0 * j5 → 3 → findUPar(3) = 0 (after union) * r3 → 4 → findUPar(4) = 2   So we get:   | **Parent Index** | **Emails** | | --- | --- | | 0 | j1, j2, j3, j5 | | 1 | j4 | | 2 | r1, r2, r3 | | 5 | m1 |   **🧱 Step 3: Construct Final Answer**  We loop over mergedMail[], and for each non-empty vector:   * Sort the emails * Use the **name from the original account at that index**  | **Group** | **Name** | **Sorted Emails** | | --- | --- | --- | | 0 | John | j1, j2, j3, j5 | | 1 | John | j4 | | 2 | Raj | r1, r2, r3 | | 5 | Mary | m1 |   **✅ Final Output**  John:j1@com j2@com j3@com j5@com  John:j4@com  Mary:m1@com  Raj:r1@com r2@com r3@com  **✅ DSU Table View (Final Parents)**  Let’s print findUPar(i) for i = 0 to 5   | **Index** | **Account Name** | **Parent (after unions)** | | --- | --- | --- | | 0 | John | 0 | | 1 | John | 1 | | 2 | Raj | 2 | | 3 | John | 0 | | 4 | Raj | 2 | | 5 | Mary | 5 | |
| **Output:-**  John:j1@com j2@com j3@com j5@com  John:j4@com  Mary:m1@com  Raj:r1@com r2@com r3@com | |