ninjassolutions.s3.amazonaws.com/0000000000000777.cpp

```
#include <iostream>
#include <vector>
#include <stack>
#include <unordered_set>
using namespace std;

void dfs(vector<int>* edges, int start, unordered_set<int> &visited, stack<int>
&finishStack) {
 visited.insert(start);
 for (int i = 0; i < edges[start].size(); i++) {
  int adjacent = edges[start][i];
  if (visited.count(adjacent) == 0) {
   dfs(edges, adjacent, visited, finishStack);
  }
 }
 finishStack.push(start);
}

void dfs2(vector<int>* edges, int start, unordered_set<int>* component,
unordered_set<int> & visited) {
 visited.insert(start);
 component->insert(start);
 for (int i = 0; i < edges[start].size(); i++) {
  int adjacent = edges[start][i];
  if (visited.count(adjacent) == 0) {
   dfs2(edges, adjacent, component, visited);
  }
 }
}

unordered_set<unordered_set<int>*>* getSCC(vector<int>* edges, vector<int>* edgesT, int
n) {
 unordered_set<int> visited;
 stack<int> finishedVertices;
 for (int i = 0; i < n; i++) {
  if (visited.count(i) == 0) {
   dfs(edges, i, visited, finishedVertices);
  }
 }
 unordered_set<unordered_set<int>*>* output = new unordered_set<unordered_set<int>*>();
 visited.clear();
 while (finishedVertices.size() != 0) {
  int element = finishedVertices.top();
  finishedVertices.pop();
  if (visited.count(element) != 0) {
   continue;
  }
  unordered_set<int>* component = new unordered_set<int>();
```

```
  dfs2(edgesT, element, component, visited);
  output->insert(component);
 }
 return output;
}

int main() {
 int n;
 cin >> n;
 vector<int>* edges = new vector<int>[n];
 vector<int>* edgesT = new vector<int>[n];
 int m;
 cin >> m;
 for (int i = 0; i < m; i++) {
  int j, k;
  cin >> j >> k;
  edges[j - 1].push_back(k - 1);
  edgesT[k - 1].push_back(j - 1);
 }
 unordered_set<unordered_set<int>*>* components = getSCC(edges, edgesT, n);
 unordered_set<unordered_set<int>*>::iterator it = components->begin();
 while (it != components->end()) {
  unordered_set<int>* component = *it;
  unordered_set<int>::iterator it2 = component->begin();
  while (it2 != component->end()) {
   cout << *it2  + 1<< " ";
   it2++;
  }
  cout << endl;
  delete component;
  it++;
 }
 delete components;
 delete [] edges;
 delete [] edgesT;
}
```