```
/*
    Time complexity: O(V + E)
    Space complexity: O(V^2)

    where V is the number of vertices in the input graph and
    E is the number of edges in the input graph
*/

#include <algorithm>
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

vector<int> getOneComponent(bool** graph, int startingVertex, int v, bool* visited) {
    vector<int> component;
    queue<int> pendingNodes;
    pendingNodes.push(startingVertex);
    visited[startingVertex] = true;

    while (!pendingNodes.empty()) {
        int current = pendingNodes.front();
        pendingNodes.pop();
        component.push_back(current);
        for (int i = 0; i < v; ++i) {
            if (graph[current][i] && !visited[i]) {
                pendingNodes.push(i);
                visited[i] = true;
            }
        }
    }

    return component;
}

vector<vector<int>> getAllComponents(bool** graph, int v) {
    vector<vector<int>> result;
    bool* visited = new bool[v]();

    for (int i = 0; i < v; ++i) {
        if (!visited[i]) {
            vector<int> component = getOneComponent(graph, i, v, visited);
            result.push_back(component);
        }
    }

    delete[] visited;
    return result;
```

```
}

int main() {
    int v, e;
    cin >> v >> e;

    bool** graph = new bool*[v];

    for (int i = 0; i < v; i++) {
        graph[i] = new bool[v]();
    }

    for (int i = 0, a, b; i < e; ++i) {
        cin >> a >> b;
        graph[a][b] = true;
        graph[b][a] = true;
    }

    vector<vector<int>> connectedComponenents = getAllComponents(graph, v);

    for (int i = 0; i < connectedComponenents.size(); i++) {
        sort(connectedComponenents[i].begin(), connectedComponenents[i].end());
        for (int j = 0; j < connectedComponenents[i].size(); j++) {
            cout << connectedComponenents[i][j] << " ";
        }

        cout << "\n";
    }
}
```