Last update: June 24, 2024 Original

Strong Orientation

A **strong orientation** of an undirected graph is an assignment of a direction to each edge that makes it a strongly connected graph. That is, after the *orientation* we should be able to visit any vertex from any vertex by following the directed edges.

Solution

Of course, this cannot be done to *every* graph. Consider a bridge in a graph. We have to assign a direction to it and by doing so we make this bridge "crossable" in only one direction. That means we can't go from one of the bridge's ends to the other, so we can't make the graph strongly connected.

Now consider a DFS through a bridgeless connected graph. Clearly, we will visit each vertex. And since there are no bridges, we can remove any DFS tree edge and still be able to go from below the edge to above the edge by using a path that contains at least one back edge. From this follows that from any vertex we can go to the root of the DFS tree. Also, from the root of the DFS tree we can visit any vertex we choose. We found a strong orientation!

In other words, to strongly orient a bridgeless connected graph, run a DFS on it and let the DFS tree edges point away from the DFS root and all other edges from the descendant to the ancestor in the DFS tree.

The result that bridgeless connected graphs are exactly the graphs that have strong orientations is called **Robbins' theorem**.

Problem extension

Let's consider the problem of finding a graph orientation so that the number of SCCs is minimal.

Of course, each graph component can be considered separately. Now, since only bridgeless graphs are strongly orientable, let's remove all bridges temporarily. We end up with some number of bridgeless components (exactly *how many components there were at the beginning* + *how many bridges there were*) and we know that we can strongly orient each of them.

We were only allowed to orient edges, not remove them, but it turns out we can orient the bridges arbitrarily. Of course, the easiest way to orient them is to run the algorithm described

above without modifications on each original connected component.

Implementation

Here, the input is n — the number of vertices, m — the number of edges, then m lines describing the edges.

The output is the minimal number of SCCs on the first line and on the second line a string of m characters, either `>` — telling us that the corresponding edge from the input is oriented from the left to the right vertex (as in the input), or `<` — the opposite.

This is a bridge search algorithm modified to also orient the edges, you can as well orient the edges as a first step and count the SCCs on the oriented graph as a second.

```
vector<vector<pair<int, int>>> adj; // adjacency list - vertex and edge pairs
vector<pair<int, int>> edges;

vector<int> tin, low;
int bridge_cnt;
string orient;
vector<bool> edge_used;
void find_bridges(int v) {
    static int time = 0;
    low[v] = tin[v] = time++;
    for (auto p : adj[v]) {
        if (edge_used[p.second]) continue;
        edge_used[p.second] = true;
        orient[p.second] = v == edges[p.second].first ? '>' : '<';
        int nv = p.first;
        if (tin[nv] == -1) { // if nv is not visited yet
            find_bridges(nv);
            low[v] = min(low[v], low[nv]);
            if (low[nv] > tin[v]) {
                // a bridge between v and nv
                bridge_cnt++;
            }
        } else {
            low[v] = min(low[v], tin[nv]);
        }
    }
}

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    adj.resize(n);
    tin.resize(n, -1);
    low.resize(n, -1);
    orient.resize(m);
    edges.resize(m);
    edge_used.resize(m);
    for (int i = 0; i < m; i++) {
        int a, b;
```

```
        scanf("%d %d", &a, &b);
        a--; b--;
        adj[a].push_back({b, i});
        adj[b].push_back({a, i});
        edges[i] = {a, b};
    }
    int comp_cnt = 0;
    for (int v = 0; v < n; v++) {
        if (tin[v] == -1) {
            comp_cnt++;
            find_bridges(v);
        }
    }
    printf("%d\n%s\n", comp_cnt + bridge_cnt, orient.c_str());
}
```

Practice Problems

- 26th Polish OI - Osiedla

Contributors:

wikku (88.78999999999999%) jakobkogler (6.03%) adamant-pwn (4.31%) Atulbedwal (0.86%)