**C. Condensed Component Graph**

For a given directed graph, create the condensed component graph where each SCC is represented as a single node, and edges exist between components if there is at least one edge between any two nodes of different components in the original graph.

**Input Format:**

First line: N (number of vertices) and M (number of edges)

Next M lines: u v (representing a directed edge from u to v)

**Output Format:**

First line: K (number of components)

Next lines: edges in the condensed graph in format u v (using component numbers)

Test Case:

8 10

0 1

1 2

2 0

2 3

3 4

3 7

4 5

5 6

6 4

7 3

Output:

3

0 1

1 2

**This problem is the basic concept of creating a component graph from SCC**

import java.io.\*;

import java.util.\*;

public class Main {

static int N, M;

static List<Integer>[] graph, revGraph;

static boolean[] visited;

static List<Integer> order;

static int[] comp;

public static void main(String[] args) throws IOException {

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

StringTokenizer st = new StringTokenizer(in.readLine());

N = Integer.parseInt(st.nextToken());

M = Integer.parseInt(st.nextToken());

graph = new ArrayList[N];

revGraph = new ArrayList[N];

for (int i = 0; i < N; i++) {

graph[i] = new ArrayList<>();

revGraph[i] = new ArrayList<>();

}

int u, v;

for (int i = 0; i < M; i++) {

st = new StringTokenizer(in.readLine());

u = Integer.parseInt(st.nextToken());

v = Integer.parseInt(st.nextToken());

graph[u].add(v);

revGraph[v].add(u);

}

in.close();

visited = new boolean[N];

order = new ArrayList<>(N);

for (int i = 0; i < N; i++) {

if (!visited[i]) dfs1(i);

}

comp = new int[N];

Arrays.fill(comp, -1);

int compCnt = 0;

for (int i = N - 1; i >= 0; i--) {

int node = order.get(i);

if (comp[node] == -1) {

dfs2(node, compCnt++);

}

}

List<Set<Integer>> condensed = new ArrayList<>(compCnt);

for (int i = 0; i < compCnt; i++) {

condensed.add(new HashSet<>());

}

for (int x = 0; x < N; x++) {

for (int y : graph[x]) {

int cx = comp[x], cy = comp[y];

if (cx != cy) {

condensed.get(cx).add(cy);

}

}

}

StringBuilder sb = new StringBuilder();

sb.append(compCnt).append('\n');

for (int i = 0; i < compCnt; i++) {

List<Integer> targets = new ArrayList<>(condensed.get(i));

Collections.sort(targets);

for (int t : targets) {

sb.append(i).append(' ').append(t).append('\n');

}

}

System.out.print(sb);

}

static void dfs1(int u) {

visited[u] = true;

for (int v : graph[u]) {

if (!visited[v]) dfs1(v);

}

order.add(u);

}

static void dfs2(int u, int cid) {

comp[u] = cid;

for (int v : revGraph[u]) {

if (comp[v] == -1) dfs2(v, cid);

}

}

}