Strongly Connected Component

Given a directed graph G=(V,E) where V={1,. . ., N} is the number of nodes and the set E has M arcs. Compute number of strongly connected components of G

Input

- Line 1: two positive integers N and M (1 <= N <= 10^5, 1 <= M <= 10^6)
- Line i+1 (i=1,. . ., M): contains two positive integers u and v which are endpoints of i^{th} arc

Output

Write the number of strongly connected components of G

Strongly Connected Component

Input

```
8 13
1 2
1 8
2 3
2 6
3 6
4 3
4 6
5 4
6 5
7 1
7 2
7 6
8 7
```

Output

```
3
```

Strongly Connected Component

Algorithm

- Run DFS on G → compute the finishing time f(v) of each node v of G
- Build residual graph G^T of G
- Run DFS on G^T: the nodes are considered in a decreasing order of f
    - Each run DFS(u) will visit all nodes of the strongly connected component containing u

Implementation – Strongly Connected Component

```
#include <stdio.h>
#include <bits/stdc++.h>
#include <vector>
#include <iostream>
using namespace std;
#define MAX_N 100001

int n;
vector<int> A[MAX_N];
vector<int> A1[MAX_N];// residual graph

// data structure for DFS
int f[MAX_N];// finishing time
char color[MAX_N];
int t;
int icc[MAX_N];// icc[v] index of the strongly connected component containing v
int ncc;// number of connected components in the second DFS
int x[MAX_N];// sorted-list (decreasing of finishing time) of nodes visited by DFS
int idx;
```

Implementation – Strongly Connected Component

```
void buildResidualGraph(){// xay dung do thi bu
    for(int u = 1; u <= n; u++){
        for(int j = 0; j < A[u].size(); j++){
            int v = A[u][j];
            A1[v].push_back(u);
        }
    }
}
void init(){
    for(int v = 1; v <= n; v++){
        color[v] = 'W';
    }
    t = 0;
}
```

Implementation – Strongly Connected Component

```
// DFS on the original graph
void dfsA(int s){
    t++;    color[s] = 'G';
    for(int j = 0; j < A[s].size(); j++){
        int v = A[s][j];
        if(color[v] == 'W'){   dfsA(v);    }
    }
    t++;
    f[s] = t;
    color[s] = 'B';
    idx++;
    x[idx] = s;
}
void dfsA(){
    init();
    idx = 0;
    for(int v = 1; v <= n; v++){
        if(color[v] == 'W'){
            dfsA(v);
        }
    }
}
```

Implementation – Strongly Connected Component

```
// DFS on the residual graph
void dfsA1(int s){
    t++;    color[s] = 'G';    icc[s] = ncc;
    //for(set<int>::iterator it = A1[s].begin(); it != A1[s].end(); it++){
    for(int j = 0; j < A1[s].size(); j++){
        int v= A1[s][j];
        if(color[v] == 'W'){    dfsA1(v);   }
    }
    color[s] = 'B';
}

void dfsA1(){
    init();
    ncc = 0;
    for(int i = n; i >= 1; i--){
        int v = x[i];
        if(color[v] == 'W'){
            ncc++;
            dfsA1(v);
        }
    }
}
```

Implementation – Strongly Connected Component

```
void solve(){
    dfsA();
    buildResidualGraph();
    dfsA1();
    cout << ncc;
}
void input(){
    int m;
    cin >> n >> m;
    for(int k = 1; k <= m; k++){
        int u,v;
        cin >> u >> v;
        A[u].push_back(v);
    }
}

int main(){
    input();
    solve();
}
```