//Articulation Point

//Complexity O(V+E)

//Tarjan, DFS

vector<int>G[101];

int dfs\_num[101], dfs\_low[101], parent[101], isAtriculationPoint[101], dfsCounter, rootChildren, dfsRoot;

void articulationPoint(int u)

{

dfs\_low[u] = dfs\_num[u] = ++dfsCounter;

for(int i = 0; i < G[u].size(); i++) {

int v = G[u][i];

if(dfs\_num[v] == 0) {

parent[v] = u;

if(u == dfsRoot) //Special case for root node

rootChildren++; //if root node has child, increment counter

articulationPoint(v);

//1 : if dfs\_num[u] == dfs\_low[v], then it is a back edge

//2 : if dfs\_num[u] < dfs\_low[v], then u is ancestor of v and there is no back edge

//so, if u is not root node, then we can chose u for Articulation Point

if(dfs\_num[u] <= dfs\_low[v] && u != dfsRoot) //Avoiding root node

isArticulationPoint[u]++;

//if there is any child node of u that is a back edge of a previous node

//then the value of dfs\_low[v] might be less than the present dfs\_low[u]

//we try to save the lowest value possible

dfs\_low[u] = min(dfs\_low[v], dfs\_low[u]);

}

//As nodes are bi-directional, avoiding direct child node

//if it is not direct child node, and visited, then there is a back edge

//so we try to decrease the value of dfs\_low[u] with the dfs\_num[v]

//the dfs\_num[v] is less than dfs\_num[u] (as it it a back edge)

else if(parent[u] != v)

dfs\_low[u] = min(dfs\_low[u], dfs\_num[v]);

}

}

int main()

{

//Actual code of Articulation Point starts here

dfsCounter = 0;

memset(dfs\_num, 0, sizeof(dfs\_num));

isArticulationPoint.reset();

for(int i = 1; i <= n; i++) {

if(dfs\_num[i] == 0) {

dfsCounter = rootChildren = 0;

dfsRoot = i;

articulationPoint(i);

isArticulationPoint[i] = (rootChildren > 1);

}

//Important

isAtriculationPoint + 1 = number of nodes that is disconnected

}

//Printing Articulation Points

/\*for(int i = 0; i < 101; i++)

if(isArticulationPoint[i])

printf("%d ", i);

printf("\n");\*/

printf("%d\n", (int)isArticulationPoint.count());

return 0;

}