//Finding Bridges (Graph)

//Complexity : O(V+E)

#define MAX 150

vector<int> G[MAX];

vector<pair<int, int> >ans;

int dfs\_num[MAX], dfs\_low[MAX], parent[MAX], dfsCounter;

void bridge(int u) {

//dfs\_num[u] is the dfs counter of u node

//dfs\_low[u] is the minimum dfs counter of u node (it is minimum if a backedge exists)

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

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

int v = G[u][i];

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

parent[v] = u;

bridge(v);

//if dfs\_num[u] is lower than dfs\_low[v], then there is no back edge on u node

//so u - v can be a bridge

if(dfs\_num[u] < dfs\_low[v])

ans.push\_back(make\_pair(min(u, v), max(u, v)));

//obtainig lower dfs counter (if found) from child nodes

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

}

//if v is not parent of u then it is a back edge

//also dfs\_num[v] must be less than dfs\_low[u]

//so we update it

else if(parent[u] != v)

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

}

}

int main() {

//Bridge finding code

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

dfsCounter = 0;

for(int i = 0; i < n; i++)

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

bridge(i);

//Output

sort(ans.begin(), ans.end());

for(int i = 0; i < ans.size(); i++)

printf("%d - %d\n", ans[i].first, ans[i].second);

printf("\n");

return 0;

}