**Bridges In A Graph**

**Difficulty: MEDIUM**

**Avg. time to solve**

**25 min**

**Success Rate**

**65%**

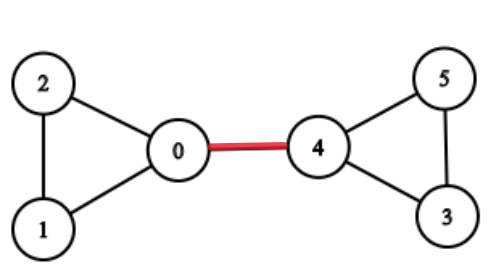
**Problem Statement**

**Suggest Edit**

**Given an undirected graph of V vertices and E edges. Your task is to find all the bridges in the given undirected graph. A bridge in any graph is defined as an edge which, when removed, makes the graph disconnected (or more precisely, increases the number of connected components in the graph).**

**For Example :**

If the given graph is :



Then the edge between 0 and 4 is the bridge because if the edge between 0 and 4 is removed, then there will be no path left to reach from 0 to 4.and makes the graph disconnected, and increases the number of connected components.

**Note :**

There are no self-loops(an edge connecting the vertex to itself) in the given graph.

There are no parallel edges i.e no two vertices are directly connected by more than 1 edge.

**Input Format :**

The first line of input contains an integer T, the number of test cases.

The first line of each test case contains two single space-separated integers V, and E.

From the second line onwards of each test case, the next 'E' lines will denote the edges of the graph where every edge is defined by two single space-separated integers 'a' and 'b', which signifies an edge between vertex 'a’ and vertex 'b'.

**Output Format :**

For each test case,

In the first line, print a single integer C denoting the count of bridges in the given graph.

From the second line onwards of each test case, Print C lines, each line contains one edge i.e two space-separated integers a and b, where C denotes the number of bridges in the given graph.

**Note :**

In the output format

Each pair of vertices of a bridge will be sorted i.e the first value should be less than or equals to the second value.

The list of bridge edges will be sorted in non-decreasing order of their first value. In case if two pairs have the same first value, the pair with a smaller second value will come first.

You do not need to print or sort anything, it has already been taken care of. Just implement the given function.

**Constraints :**

1 <= T <= 50

1 <= V <= 10 ^ 3

V-1 <= E <= 3 \* 10^3

0 <= a, b < V

Time Limit: 1 sec

**Sample Input 1 :**

2

5 4

0 1

3 1

1 2

3 4

3 3

0 1

1 2

2 0

**Sample Output 1 :**

4

0 1

1 2

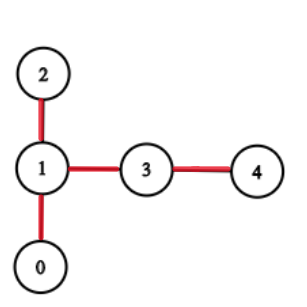
1 3

3 4

0

**Explanation For Sample Input 1 :**

For the first test case, the graph will be represented as



There are four bridges((0-1),(1-2),(1-3),(3-4)) in the above-given graph denoted by red lines.

For the second test case, there is no bridge present in the given graph.

**Sample Input 2 :**

1

6 7

1 2

1 0

0 2

0 4

5 4

5 3

3 4

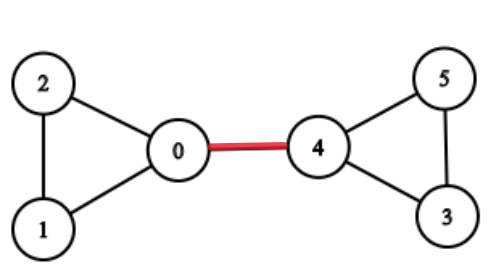
**Sample Output 2 :**

1

0 4

**Explanation For Sample Input 2 :**

For the first test case, the graph will be represented as



There is only one bridge((0-4)) in the above-given graph denoted by red lines.

#include<bits/stdc++.h>

void dfs(int i, int par, vector<vector<int>> &graph, vector<bool> &visited, vector<int> &in, vector<int> &low,

        int &timer, vector<vector<int>> &ans) {

    visited[i]=true;

    in[i]=timer++;

    low[i]=in[i];

    for (auto x : graph[i]) {

        if (!visited[x]) {

            dfs(x, i, graph, visited, in, low, timer, ans);

            if (low[x]>in[i]) ans.push\_back({i, x});

            low[i]=min(low[i], low[x]);

        }

        else {

            if (x!=par) low[i]=min(low[i], in[x]);

        }

    }

}

vector<vector<int>> findBridges(vector<vector<int>> &edges, int n, int e) {

    // Write your code here

    vector<vector<int>> graph(n);

    for (auto x : edges) {

        graph[x[0]].push\_back(x[1]);

        graph[x[1]].push\_back(x[0]);

    }

    vector<int> in(n), low(n);

    vector<vector<int>> ans;

    int timer=1;

    vector<bool> visited(n, false);

    for (int i=0; i<n; i++) {

        if (!visited[i]) dfs(i, -1, graph, visited, in, low, timer, ans);

    }

    return ans;

}