Find Bidirectional Edges

Problem

Submissions

Discussion Coming Soon

You are given a graph with N nodes and M **directed**(one-way) edges.

Your goal is to find bidirectional edges. If a pair of nodes have two-way connection than that means there is a bidirectional edge between them.

Such as; if there are 2 nodes in a graph, and a directed edge from $node_1$ towards $node_2$, and another directed edge from $node_2$ towards $node_1$. It means there is a bidirectional edge between these two nodes.

Print the bidirectional edges in the lexicographical order.

Input Format

First line N, and M - the number of nodes in the graph and the number of edges respectively

M lines each containing u_i and v_i, meaning there is an directed edge connecting node u_i to v_i

Output Format

Print number of bidirectional edges in the first line. Next lines should contain node numbers of edges in lexicographical order.

Constraints

$1 \leq N \leq 10^5$

$1 \leq M \leq 10^5$

$1 \leq u, v \leq N$

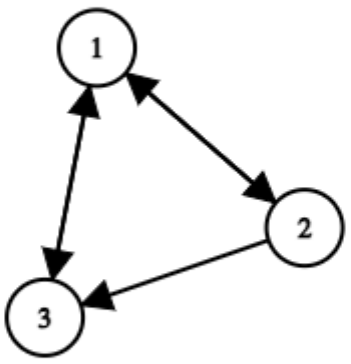
Sample Input 1

```
4 5
1 3
1 2
3 1
2 1
2 3
```

Sample Output 1

```
2
1 2
1 3
```

Explanation 1



C++ (GCC 9.2.0) Bright **Memory Limit (kB) :** 256000 **Time Limit (s) :** 1

```cpp
#include <iostream>
#include <map>
#include <set>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    int N,M;
    cin >> N >> M;

    map<int, set<int>> edges;

    for(int i = 0; i < M; i++){
        int u, v;
        cin >> u >> v;
        edges[u].insert(v);
    }

    vector<pair<int,int>> bidirect;

    for(auto edge : edges){
        for(auto dest : edge.second){
            if(edges[dest].count(edge.first))
                bidirect.push_back(make_pair(min(edge.first, dest), max(edge.first, dest)
        }
    }

    sort(bidirect.begin(), bidirect.end());
    bidirect.erase(unique(bidirect.begin(), bidirect.end()), bidirect.end());
```

Upload File Test against custom test case Run Code Submit

Sample Test Case 0

Accepted

Input(stdin)

```
4 5
1 3
1 2
3 1
2 1
2 3
```

Output(stdin)

```
2
```

```
1 2
1 3
```

Expected Output

```
2
1 2
1 3
```