Even Tree

by HackerRank

Problem Submissions Leaderboard Discussions Editorial

You are given a tree (a simple connected graph with no cycles). The tree has N nodes numbered from 1 to N and is rooted at node 1.

Find the maximum number of edges you can remove from the tree to get a forest such that each connected component of the forest contains an even number of vertices.

Input Format

The first line of input contains two integers N and M. N is the number of vertices, and M is the number of edges.
The next M lines contain two integers u_i and v_i which specifies an edge of the tree.

Constraints

- $2 \leq N \leq 100$

Note: The tree in the input will be such that it can always be decomposed into components containing an even number of nodes.

Output Format

Print the number of removed edges.

Sample Input

```
10 9
2 1
3 1
4 3
5 2
6 1
7 2
8 6
9 8
10 8
```

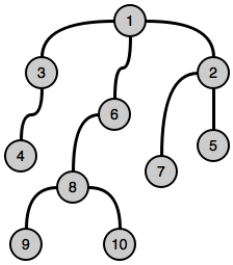
Sample Output

```
2
```

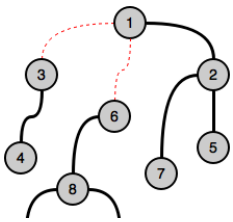
Explanation

On removing edges $(1, 3)$ and $(1, 6)$, we can get the desired result.

Original tree:



Decomposed tree:



Submissions: 20141
Max Score: 50
Difficulty: Medium

Rate This Challenge:

More

Current Buffer (saved locally, editable)

C++14

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

//#define DEBUG

int edge_count(vector<int> *g, int edge) {
    if (g[edge].size() == 0) {
        return 0;
    } else {
        int sum = 0;

        for (int i = 0; i < g[edge].size(); i++) {
            sum += 1 + edge_count(g, g[edge][i]);
        }

        return sum;
    }
}

void print_graph(vector<int> *g, int m) {
    for (int i = 1; i <= m; i++) {
        if (g[i].size() == 0) {
            continue;
        }

        cout << i << ": ";

        for (int j = 0; j < g[i].size(); j++) {
            cout << g[i][j] << " ";
        }

        cout << "(count: " << 1 + edge_count(g, i) << ")\n";
    }
}

int max_removals(vector<int> *g, int edge) {
    if (g[edge].size() == 0) {
        return 0;
    } else {
        int removals = 0;

        for (int i = 0; i < g[edge].size(); i++) {
            int count = 1 + edge_count(g, g[edge][i]);
                removals += (count % 2 == 0 ? 1 : 0) + max_removals(g, g[edge][i]);
#ifdef DEBUG
            if (count % 2 == 0) {
                cout << "remove edge " << g[edge][i] << "," << edge << "\n";
            }
#endif
        }

        return removals;
    }
}

void read_graph(vector <int> *g, int m) {
    for (int i = 0; i < m; i++) {
        int edgeA, edgeB;
        cin >> edgeA >> edgeB;
        if (g[edgeA].size() > 0) g[edgeA].push_back(edgeB);
        else g[edgeB].push_back(edgeA);
    }

}

int main() {
    int n, m, removals = 0;

    cin >> n >> m;
    vector<int> graph[n + 1];

    read_graph(graph, m);

    cout << max_removals(graph, 1) << "\n";
#ifdef DEBUG
    print_graph(graph, m);
#endif
    return 0;
}

```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code