CIA DLAB2

**The Problem: Network Bridges

Problem Statement:

You are given a network represented by a list of undirected edges. The network has `n` nodes (labeled from 1 to `n`) and `m` edges. Your task is to determine if the network is connected. If it's connected, find any spanning tree of the network. If it is not connected print `Disconnected`. If connected, output `Connected` followed by the edges of one spanning tree, each on a new line.

Input:

- The first line contains two integers, `n` (1 <= n <= 10^5) and `m` (0 <= m <= 2 * 10^5), representing the number of nodes and the number of edges respectively.
- The next `m` lines each contain two integers `a` and `b` (1 <= a, b <= n, a != b), representing an undirected edge between nodes `a` and `b`. There may be multiple edges between the same nodes.

Output:

- If the network is disconnected, print "Disconnected".
- If the network is connected, print "Connected" on the first line. Then, print `n-1` lines, each containing two integers representing the edges of one possible spanning tree. The order of the edges does not matter.

Input:

```
5 6
1 2
1 3
2 3
2 4
3 5
4 5
```

Output:

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

Solution:

```cpp
#include <iostream>
#include <vector>

using namespace std;

vector<int> adj[100001]; // Adjacency list to represent the graph
bool visited[100001];    // Keep track of visited nodes
vector<pair<int, int>> spanning_tree; // Vector to store edges of the spanning tree

void dfs(int u, int parent = -1) {
    visited[u] = true;

    for (int v : adj[u]) {
        if (!visited[v]) {
            spanning_tree.push_back({u, v});
            dfs(v, u);
        } // else if v != parent, then it is not used in the spanning tree needed
    }
}

int main() {
    int n, m;
    cin >> n >> m;

    for (int i = 0; i < m; ++i) {
        int a, b;
        cin >> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    // Check connectivity.
    int components = 0;
    for (int i = 1; i <= n; ++i) {
        if (!visited[i]) {
            dfs(i);  // Start DFS from an unvisited node
            components++;  // Increment component count
        }
    }

    if (components > 1) {
        cout << "Disconnected" << endl;
    } else {
        cout << "Connected" << endl;
        for (const auto& edge : spanning_tree) {
            cout << edge.first << " " << edge.second << endl;
        }
    }

    return 0;
}
```

PROBLEM 2:

The Problem: Product Subarray (CSES Level: Introductory/Intermediate)

Problem Statement:

You're given an array of `n` integers. Your task is to find the maximum product of any contiguous subarray within the array. The array can contain positive, negative, and zero elements.

Input:

- The first line contains an integer `n` (1 <= n <= 2 * 10^5), representing the number of elements in the array.
- The second line contains `n` integers `a[1], a[2], ..., a[n]` (-10 <= a[i] <= 10) representing the elements of the array.

Output:

- Print a single integer: the maximum product of any contiguous subarray within the input array. The absolute value of max product is known to be at most `10^18`. Therefore, use `long long` instead of `int`.

Example Input:

```
5
```

```
2 3 -2 4 -1
```

Example Output:

```
48
```

Solution:

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    int n;
    cin >> n;

    long long arr[n];
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }

    long long max_product = arr[0];  // Initialize with the first element
    long long max_so_far = arr[0];
    long long min_so_far = arr[0];

    for (int i = 1; i < n; ++i) {
        //  Positive current element:  extend both max and min streak.
        //  Negative current element: flip max and min streak, extending them.
        long long current = arr[i];
        long long temp_max = max({current, max_so_far * current, min_so_far * current});
        min_so_far = min({current, max_so_far * current, min_so_far * current});

        max_so_far = temp_max; // Assign max_so_far to temp_max

        max_product = max(max_product, max_so_far);
    }

    cout << max_product << endl;

    return 0;
}
```