Check whether a graph is bipartite

Table of Contents

A bipartite graph is a graph whose vertices can be divided into two disjoint sets so that every edge connects two vertices from different sets (i.e. there are no edges which connect vertices from the same set). These sets are usually called sides.

You are given an undirected graph. Check whether it is bipartite, and if it is, output its sides.

Algorithm

There exists a theorem which claims that a graph is bipartite if and only if all its cycles have even length. However, in practice it's more convenient to use a different formulation of the definition: a graph is bipartite if and only if it is two-colorable.

Let's use a series of breadth-first searches, starting from each vertex which hasn't been visited yet. In each search, assign the vertex from which we start to side 1. Each time we visit a yet unvisited neighbor of a vertex assigned to one side, we assign it to the other side. When we try to go to a neighbor of a vertex assigned to one side which has already been visited, we check that is has been assigned to the other side; if it has been assigned to the same side, we conclude that the graph is not bipartite. Once we've visited all vertices and successfully assigned them to sides, we know that the graph is bipartite and we have constructed its partitioning.

Implementation

```
int n;
vector<vector<int>> adj;

vector<int> side(n, -1);
bool is_bipartite = true;
queue<int> q;
for (int st = 0; st < n; ++st) {
    if (side[st] == -1) {
        q.push(st);
        side[st] = 0;
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int u : adj[v]) {
                if (side[u] == -1) {
```

```
                    side[u] = side[v] ^ 1
                    q.push(u);
                } else {
                    is_bipartite &= side[u] !=
                }
            }
        }
    }
}

cout << (is_bipartite ? "YES" : "NO") << endl;
```

Practice problems:

- SPOJ - BUGLIFE