Redundant connection in C++

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

using namespace std;

class UnionFind {
public:
    vector<int> parent;
    vector<int> rank;

    UnionFind(int n) {
        parent.resize(n + 1);
        rank.resize(n + 1, 1);
        for (int i = 1; i <= n; ++i) {
            parent[i] = i;
        }
    }

    int find(int x) {
        if (parent[x] != x) {
            parent[x] = find(parent[x]);  // Path compression
        }
        return parent[x];
    }

    void unionSets(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);

        if (rootX != rootY) {
            if (rank[rootX] > rank[rootY]) {
                parent[rootY] = rootX;
            } else if (rank[rootX] < rank[rootY]) {
                parent[rootX] = rootY;
            } else {
                parent[rootY] = rootX;
                rank[rootX]++;
            }
        }
    }
};

vector<int> findRedundantConnection(vector<vector<int>>& edges) {
    int n = edges.size();
    UnionFind uf(n);

    for (auto& edge : edges) {
        int u = edge[0];
        int v = edge[1];

        if (uf.find(u) == uf.find(v)) {
            return edge;  // This edge is a redundant connection
        }
        uf.unionSets(u, v);
    }
    return {};
```

You're given edges forming a graph. Initially, it's a tree (n nodes, n−1 edges), but one extra edge was added, forming a cycle.
Goal: Find the **redundant edge** forming the cycle.

📜 Input

```
edges = {
    {1, 2},
    {1, 3},
    {2, 3}
}
```

🎁 Initial Setup

- Nodes: `1, 2, 3`
- `parent[] = [0, 1, 2, 3]` (0-index unused)
- `rank[] = [0, 1, 1, 1]`

🧮 Dry Run Table (Union-Find Process)

Step	Edge	Find(u)	Find(v)	Same Root?	Action	Updated `parent[]`	Updated `rank[]`
1	1-2	1	2	✗ No	Union(1, 2)	[0, 1, 1, 3]	[0, 2, 1, 1]
2	1-3	1	3	✗ No	Union(1, 3)	[0, 1, 1, 1]	[0, 2, 1, 1]
3	2-3	1	1	✓ Yes	**! Cycle found**	—	—

✅ Output

```
2 3
```

- Edge **{2, 3}** forms the cycle.
- It is **redundant**, and hence returned.

```
}

int main() {
    // Hardcoded input
    vector<vector<int>> edges = {
        {1, 2},
        {1, 3},
        {2, 3}
    };

    vector<int> result =
findRedundantConnection(edges);
    cout << result[0] << " " << result[1] <<
endl;

    return 0;
}
```

Output:-
3