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 {};
}

int main() {
```

Dry Run

Input:

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

Step-by-Step Execution:

- **Initialization**:
  - UnionFind:
    - parent = [1, 2, 3] (each node is its own parent initially)
    - rank = [1, 1, 1] (all ranks start as 1)

Processing Edges:

1. **Edge (1, 2)**:
   - find(1) = 1, find(2) = 2 → Different components.
   - Union the components:
     - Set parent[2] = 1.
     - Update rank[1] = 2.
   - Updated state:
     - parent = [1, 1, 3]
     - rank = [1, 2, 1].
2. **Edge (1, 3)**:
   - find(1) = 1, find(3) = 3 → Different components.
   - Union the components:
     - Set parent[3] = 1.
   - Updated state:
     - parent = [1, 1, 1]
     - rank = [1, 2, 1].
3. **Edge (2, 3)**:
   - find(2) = 1, find(3) = 1 → Same component (cycle detected).
   - Return the edge (2, 3) as the redundant connection.

```
    // 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:-
2 3