B!

#include <iostream>

#include <vector>

#include <queue>

#include <fstream>

#include <unordered\_map>

using namespace std;

void jobSort(int *n*, const vector<pair<int, int>>& *dependencies*, vector<int>& *ordered\_jobs*) {

    vector<vector<int>> adjacency\_list(*n* + 1);

    vector<int> priority(*n* + 1, 0);

    for (const auto& dep : *dependencies*) {

        int u = dep.first;

        int v = dep.second;

        adjacency\_list[u].push\_back(v);

        priority[v]++;

    }

    queue<int> q;

    for (int i = 1; i <= *n*; i++) {

        if (priority[i] == 0) {

            q.push(i);

        }

    }

    while (!q.empty()) {

        int u = q.front();

        q.pop();

*ordered\_jobs*.push\_back(u);

        for (int v : adjacency\_list[u]) {

            priority[v]--;

            if (priority[v] == 0) {

                q.push(v);

            }

        }

    }

}

int main() {

    int n, m;

*ifstream* input\_file("jobs.txt");

    input\_file >>n >> m;

    vector<pair<int, int>> dependencies;

    for (int i = 0; i < m; i++) {

        int u, v;

        input\_file >> u >> v;

        dependencies.push\_back({u, v});

    }

    vector<int> ordered\_jobs;

    jobSort(n, dependencies, ordered\_jobs);

*ofstream* output\_file("jobs.out");

    for (int i = 0; i < n; i++) {

        output\_file << ordered\_jobs[i] << " ";

    }

    return 0;

}

B2

#include <iostream>

#include <vector>

#include <fstream>

#include <algorithm>

using namespace std;

struct *Edge* {

    int u, v, cost;

    bool operator<(const *Edge*& *other*) const {

        return cost < *other*.cost;

    }

};

class *UnionFind* {

private:

    vector<int> parent;

    vector<int> rank;

public:

    UnionFind(int *n*) {

        parent.resize(*n*);

        rank.assign(*n*, 0);

        for (int i = 0; i < *n*; i++) {

            parent[i] = i;

        }

    }

    int find(int *x*) {

        if (*x* != parent[*x*]) {

            parent[*x*] = find(parent[*x*]);

        }

        return parent[*x*];

    }

    bool unite(int *x*, int *y*) {

*x* = find(*x*);

*y* = find(*y*);

        if (*x* == *y*) {

            return false;

        }

        if (rank[*x*] < rank[*y*]) {

            parent[*x*] = *y*;

        } else if (rank[*x*] > rank[*y*]) {

            parent[*y*] = *x*;

        } else {

            parent[*y*] = *x*;

            rank[*x*]++;

        }

        return true;

    }

};

int main() {

    int n, m;

    cin >> n >> m;

    vector<*Edge*> edges(m);

    for (int i = 0; i < m; i++) {

         cin >> edges[i].u >> edges[i].v >> edges[i].cost;

    }

    sort(edges.begin(), edges.end());

*UnionFind* uf(n + 1);

    vector<*Edge*> min\_spanning\_tree;

    int total\_cost = 0;

    for (const *Edge*& edge : edges) {

        if (uf.unite(edge.u, edge.v)) {

            min\_spanning\_tree.push\_back(edge);

            total\_cost += edge.cost;

        }

    }

    cout << total\_cost << endl;

    for (const *Edge*& edge : min\_spanning\_tree) {

        cout << edge.u << " " << edge.v << " " << edge.cost << endl;

    }

    return 0;

}