#include <iostream>

#include <vector>

#include <algorithm>

#include <set>

using namespace std;

struct Edge {

int u, v, weight;

};

bool comp(const Edge& a, const Edge& b) {

return a.weight > b.weight;

}

void DFS(int v, vector<bool>& visited, const vector<vector<int>>& adj) {

visited[v] = true;

for (int u : adj[v]) {

if (!visited[u]) {

DFS(u, visited, adj);

}

}

}

bool isConnected(int V, const vector<vector<int>>& adj) {

vector<bool> visited(V, false);

DFS(0, visited, adj);

for (bool v : visited) {

if (!v) return false;

}

return true;

}

vector<Edge> reverseDeleteMST(int V, vector<Edge>& edges) {

sort(edges.begin(), edges.end(), comp);

vector<vector<int>> adj(V);

for (const Edge& e : edges) {

adj[e.u].push\_back(e.v);

adj[e.v].push\_back(e.u);

}

for (int i = 0; i < edges.size(); ) {

int u = edges[i].u, v = edges[i].v;

adj[u].erase(remove(adj[u].begin(), adj[u].end(), v), adj[u].end());

adj[v].erase(remove(adj[v].begin(), adj[v].end(), u), adj[v].end());

if (isConnected(V, adj)) {

edges.erase(edges.begin() + i);

} else {

adj[u].push\_back(v);

adj[v].push\_back(u);

i++;

}

}

return edges;

}

int main() {

int V, E;

cin >> V >> E;

vector<Edge> edges(E);

for (int i = 0; i < E; i++) {

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

}

vector<Edge> mst = reverseDeleteMST(V, edges);

for (const Edge& e : mst) {

cout << e.u << " - " << e.v << " : " << e.weight << "\n";

}

return 0;

}