#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Edge {

int u, v, weight;

bool operator<(const Edge& other) const {

return weight < other.weight;

}

};

struct DSU {

vector<int> parent, rank;

DSU(int n) : parent(n), rank(n, 0) {

for (int i = 0; i < n; i++) parent[i] = i;

}

int findSet(int u) {

if (u != parent[u]) parent[u] = findSet(parent[u]);

return parent[u];

}

void unionSets(int a, int b) {

a = findSet(a);

b = findSet(b);

if (a != b) {

if (rank[a] < rank[b]) swap(a, b);

parent[b] = a;

if (rank[a] == rank[b]) rank[a]++;

}

}

};

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

DSU dsu(V);

vector<Edge> result;

int numComponents = V;

while (numComponents > 1) {

vector<Edge> minEdge(V, {-1, -1, INT\_MAX});

for (const Edge& e : edges) {

int setU = dsu.findSet(e.u), setV = dsu.findSet(e.v);

if (setU != setV) {

if (e.weight < minEdge[setU].weight) minEdge[setU] = e;

if (e.weight < minEdge[setV].weight) minEdge[setV] = e;

}

}

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

if (minEdge[i].u != -1) {

int setU = dsu.findSet(minEdge[i].u), setV = dsu.findSet(minEdge[i].v);

if (setU != setV) {

result.push\_back(minEdge[i]);

dsu.unionSets(setU, setV);

numComponents--;

}

}

}

}

return result;

}

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 = boruvkaMST(V, edges);

for (const Edge& e : mst) {

cout << e.u << " - " << e.v << " : " << e.weight << endl;

}

return 0;

}