#include <bits/stdc++.h>

using namespace std;

class FindUnion {

public:

vector<int> parent, rank;

FindUnion(int n) {

parent.resize(n + 1);

rank.resize(n + 1, 0);

for (int i = 1; i <= n; ++i) {

parent[i] = i;

}

}

int find(int u) {

if (parent[u] != u) {

parent[u] = find(parent[u]);

}

return parent[u];

}

bool unite(int u, int v) {

int rootU = find(u);

int rootV = find(v);

if (rootU == rootV) return false;

if (rank[rootU] > rank[rootV]) {

parent[rootV] = rootU;

} else if (rank[rootU] < rank[rootV]) {

parent[rootU] = rootV;

} else {

parent[rootV] = rootU;

rank[rootU]++;

}

return true;

}

};

int main() {

int N, E;

cin >> N >> E;

vector<tuple<int, int, int>> edges;

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

int A, B, W;

cin >> A >> B >> W;

edges.push\_back({W, A, B});

}

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

FindUnion uf(N);

int mst\_cost = 0;

int edges\_used = 0;

int redundant\_edges = 0;

for (auto &[W, A, B] : edges) {

if (uf.unite(A, B)) {

mst\_cost += W;

edges\_used++;

} else {

redundant\_edges++;

}

}

if (edges\_used != N - 1) {

cout << "Not Possible" << endl;

} else {

cout << redundant\_edges << " " << mst\_cost << endl;

}

return 0;

}