**Solution Notes (Nathan Pinsker):** The problem of finding the length of a minimal spanning tree is fairly well-known; two simplest algorithms for finding a minimum spanning tree are Prim's algorithm and Kruskal's algorithm. Of these two, Kruskal's algorithm processes edges in increasing order of their weights. There is an important key point of Kruskal's algorithm to consider, though: when considering a list of edges sorted by weight, edges can be greedily added into the spanning tree (as long as they do not connect two vertices that are already connected in some way).

Now consider a partially-formed spanning tree using Kruskal's algorithm. We have inserted some number of edges with lengths less than N, and now have to choose several edges of length N. The algorithm states that we must insert these edges, if possible, before any edges with length greater than N. However, we can insert these edges in any order that we want. Also note that, no matter which edges we insert, it does not change the connectivity of the graph at all. (Let us consider two possible graphs, one with an edge from vertex A to vertex B and one without. The second graph must have A and B as part of the same connected component; otherwise the edge from A to B would have been inserted at one point.)

These two facts together imply that our answer will be the product of the number of ways, using Kruskal's algorithm, to insert the edges of length K (for each possible value of K). Since there are at most three edges of any length, the different cases can be brute-forced, and the connected components can be determined after each step as they would be normally.

```
#include <iostream>
#include <algorithm>
#include <cstdlib>
#include <cstdio>
#include <set>
#include <cassert>

using namespace std;

#define MOD 1000000007
#define MAXN 40010
#define MAXE 100010

int P[MAXN];
pair<int, pair<int, int> > E[MAXE];

int find(int x) {
  return x == P[x] ? x : P[x] = find(P[x]);
}

bool merge(int x, int y) {
  int X = find(x);
  int Y = find(y);
  if(X == Y) return false;
  P[X] = P[Y] = P[x] = P[y] = rand() & 1 ? X : Y;
  return true;
}

int main() {
  freopen("simplify.in", "r", stdin);
  freopen("simplify.out", "w", stdout);

  int N, M; cin >> N >> M;
  assert(1 <= N && N <= MAXN && 1 <= M && M <= MAXE);
  for(int i = 0; i < M; i++) {
    int u, v, c; cin >> u >> v >> c;
    assert(!(u < 1 || v < 1 || u > N || v > N));
    assert(1 <= c && c <= 1000000);
    E[i] = make_pair(c, make_pair(u - 1, v - 1));
  }
  sort(E, E + M);

  for(int i = 0; i < N; i++) P[i] = i;

  long long cst = 0;
  unsigned cnt = 1;
  int mergs = 0;
  for(int i = 0; i < M; ) {
    int j;
    int num = 0;
    int tot = 0;

    set<pair<int, int> > st;
    for(j = i; j < M && E[j].first == E[i].first; j++) {
      int A = find(E[j].second.first);
      int B = find(E[j].second.second);
      if(B < A) swap(A, B);
      if(A != B) {
        st.insert(make_pair(A, B));
        tot++;
      }
    }
    assert(j - i <= 3);
    for(; i < j; i++) {
      num += merge(E[i].second.first, E[i].second.second);
    }

    mergs += num;
    cst += num * E[i - 1].first;
    if(tot == 3) {
      if(num == 1 || num == 2 && st.size() == 3) cnt = (cnt * 3) % MOD;
      if(num == 2 && st.size() == 2) cnt = (cnt * 2) % MOD;
    }
    if(tot == 2 && num == 1) cnt = (cnt * 2) % MOD;
  }
  assert(mergs == N - 1);

  cout << cst << ' ' << cnt << endl;
}
```