```
#!/usr/bin/python3

import sys

# Graph representation: adjacency list Adj[u] = [(v1,c1),(v2,c2),...]
# means edge (u,v1) with cost c1, edge (u,v2) with cost c2, etc.
Adj = []
Idx = {}                        # Idx: node name --> node index in Adj
Name = []                       # Name[v] is the name of node v

# Utility function to add a node u to the graph.  u is the node name.
def add_vertex(u):
    global V_name, Idx, Adj
    if u in Idx:
        ui = Idx[u]
    else:
        ui = len(Adj)
        Idx[u] = ui
        Adj.append([])
        Name.append(u)
    return ui

# Read an undirected graph from the standard input.  Input format: one
# edge per line.  Line format: u v c(u,v), where u and v are strings,
# and c(u,v) is a number representing the cost of edge (u,v)
#
# E.g.:
# A B 10
# B C 5.2
# ...
for l in sys.stdin:
    u, v, c = l.strip().split()
    u = add_vertex(u)
    v = add_vertex(v)
    c = float(c)
    Adj[u].append((v,c))
    Adj[v].append((u,c))


def prim(G):                    # G: adjacency list, src: source node
    n = len(G)                  # n: number of nodes in G
    T = [None]*n                # Minimal spanning tree (adjacency list), empty for now
    W = [None]*n                # W[v] cost of connecting v to the tree
    P = [None]*n                # P[v] = u means that v is connected through u

    Q = [None]*n
    Q_head = 0
    Q_tail = 0

    W[0] = 0.0                  # start up the algorithm from the first node (0)
    Q[Q_tail] = 0
    Q_tail = Q_tail + 1

    while Q_head < Q_tail:
        # find the least-cost edge, least-cost node to incorporate into the tree
        for i in range(Q_head + 1, Q_tail):
            if W[Q[i]] < W[Q[Q_head]]:
                Q[i], Q[Q_head] = Q[Q_head], Q[i]
        u = Q[Q_head]
        Q_head = Q_head + 1
        if P[u] == None:
            T[u] = []                   # connect the first node u to T
        else:
            T[u] = [(P[u],W[u])]        # connect u to T using edge (P[v],u)
            T[P[u]].append((u,W[u]))
        for v, c in G[u]:               # for every edge u --> v with cost c
            if T[v] == None:
                if W[v] == None or W[v] > c:
                    if W[v] == None:
                        Q[Q_tail] = v
                        Q_tail = Q_tail + 1
                    W[v] = c
                    P[v] = u
    return T

T = prim(Adj)
for u in range(len(Adj)):
    for v, c in T[u]:
        if u < v:
            print(Name[u],Name[v],c)
```