```
#!/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 dijkstra(G,src):            # G: adjacency list, src: source node
    n = len(G)                  # n: number of vertices in G
    D = [None]*n                # D[v]: best known distance from src to v
    P = [None]*n                # P[v] = x means that x is the next-hop towards u

    Q = [None]*n                # queue of discovered nodes whose distance
    Q_head = 0                  # isn't yet finalized
    Q_tail = 0

    D[src] = 0.0                # start up the algorithm from the source node
    Q[Q_tail] = src
    Q_tail = Q_tail + 1
    N = [False]*n               # N[v] == True ==> v's distance from src is finalized

    while Q_head < Q_tail:
        # find the least-cost path extension: extract node u from Q
        # such that D[u] is minimal
        for i in range(Q_head + 1, Q_tail):
            if D[Q[i]] < D[Q[Q_head]]:
                Q[i], Q[Q_head] = Q[Q_head], Q[i]
        u = Q[Q_head]
        Q_head = Q_head + 1
        N[u] = True             # node u is settled
        for v, c in G[u]:       # for every edge u --> v with cost c
            if N[v] == False:
                if D[v] == None or D[v] > D[u] + c:
                    if D[v] == None:
```

```
                    Q[Q_tail] = v
                    Q_tail = Q_tail + 1
                D[v] = D[u] + c
                P[v] = u
    return P, D                 # return "previous" and "distance" vectors

if len(sys.argv) > 1:           # read the name of the starting node
    src = Idx[sys.argv[1]]      # from the first command-line argument,
else:                           # or start from the first node
    src = 0

P,D = dijkstra(Adj, src)

for u in range(len(Adj)):
    print(Name[u], D[u], P[u])
```