```
#!/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 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 a directed graph from the standard input.  Input format: one
# arc 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))

def bellman_ford(G,src,dst):    # G: adjacency list, src: source, dst: destination
    #
    # return the minimal total cost of any path from src to dst
    #
    if src == dst:
        return 0
    d_min = None                # min total cost of src -...-> dst
    for x,w in G[src]:
        d_x = bellman_ford(G, x, dst) + w
        if d_x == None:
            continue
        if d_min == None or d_min > d_x:
            d_min = d_x
    return d_min

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

if len(sys.argv) > 2:           # read the name of the destination node
    dst = Idx[sys.argv[2]]      # from the second command-line argument,
else:                           # or use the last node
    dst = len(Adj) - 1

print ('The distance between', Name[src], 'and', Name[dst], 'is', bellman_ford(Adj, src,
dst))
```