```python
#!/usr/bin/python3

# Implementation of the Breadth-First Search algorithm (BFS) using an
# adjacency-list representation of a graph.

# Graph representation: V(G) = {0,1,2,...,n-1}, The graph is
# represented by an array of arrays G such that G[u]=[v1,v2,...]
# means that the graph contains edges u-->v1, u-->v2, etc.

def bfs(G,src):                 # G: adjacency list, src: source node
    n = len(G)                  # n: number of nodes in G
    D = [None]*n                # D[v]: hop-distance from src to v
    P = [None]*n                # P[v]: previous node: src --> ... u --> v

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

    Q[Q_tail] = src             # We start the exploration from src
    Q_tail = Q_tail + 1         #
    D[src] = 0                  # src is at distance 0 from itself
    P[src] = src                # by convention, P[src] is src itself

    while Q_tail > Q_head:
        u = Q[Q_head]           # extract u = head of Q
        Q_head = Q_head + 1
        for v in G[u]:          # for every adjacent node v
            if D[v] == None:
                D[v] = D[u] + 1
                P[v] = u
                Q[Q_tail] = v
                Q_tail = Q_tail + 1

    return P, D                 # return previous and distance vectors


# We read an undirected graph from the standard input. The input
# contains a sequence of undirected edges.  For example:
#
# TI VS
# UR TI
# GR TI
# ...

Adj = []                        # adjacency list
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_name):
    global Name, Idx, Adj
    if u_name in Idx:
        u = Idx[u_name]
    else:
        u = len(Adj)
        Idx[u_name] = u
        Adj.append([])
        Name.append(u_name)
    return u

import sys

for line in sys.stdin:
    uname, vname = line.strip().split()
    u = add_vertex(uname)
    v = add_vertex(vname)
    Adj[u].append(v)
    Adj[v].append(u)

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 = bfs(Adj, src)

print('BFS starting from', Name[src])
c = 0
for u in range(len(Adj)):
    if D != None:
        print(Name[u], D[u], Name[P[u]])
        c = c + 1
print('reached',c,'out of',len(Adj),'nodes')
```