# Metadata Collection

It has been well (http://www.theguardian.com/world/2013/jun/27/nsa-data-mining-authorised-obama) covered (http://www.npr.org/2014/01/10/261282601/transcript-nsa-deputy-director-john-inglis) that the NSA and other intelligence agencies collect a huge amount of communications metadata, which contains the records of who has contacted whom, but are legally constrained in how they access and use this data. One of the constraints is that they may only access the metadata of individuals who are three "hops" from an individual for whom the NSA has "reasonable articulable suspicion."

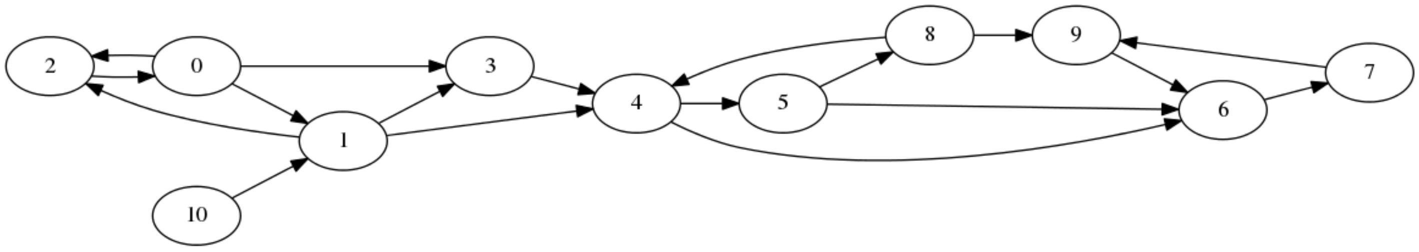
If person A is suspicious, and he or she has contacted person B, then B is one hop from A. If B then contacts C, then C is two hops from A.

The metadata, of course, creates a directed graph detailing who has contacted whom, and our graph search algorithms make a basis for a magnificent way of answering lots of interesting questions about who may and may not be investigated.

# Our Data

This file (labs/metaData.tgz) contains metadata from two different datasets, one synthetic, and one real. It can be extracted using the command `tar xzf metaData.tgz`. The real one is described here (http://snap.stanford.edu/data/email-EuAll.html). I have converted it to a .dot file for you (email.dot).

I also constructed a small synthetic example (smallExample.dot) to make debugging easier. I have visualized it for you below:


$$\begin{tikzcd}
2 \arrow[r, bend left] & 0 \arrow[l, bend left] \arrow[rr] \arrow[dr] & & 3 \arrow[r] & 4 \arrow[r] \arrow[rrrr, bend right] & 5 \arrow[ur] \arrow[rrr] & 8 \arrow[r] \arrow[ll, bend right] & 9 \arrow[r] & 6 \arrow[r] & 7 \arrow[ll] \\
 & & 1 \arrow[ull] \arrow[ur] \arrow[urr] & & & & & & & \\
 & 10 \arrow[ur] & & & & & & & &
\end{tikzcd}$$

To understand this graph, consider individuals 0 and 3. Inside the dot file, you will see a line `0 -> 3;`. This means that person 0 has contacted person 3. Person 3 is therefore one hop from person 0. Because the arrow does not go from person 3 to person 0, person 0 is NOT one hop away from person 3.

# The Assignment

**Part One**: 88 points

You'll write a file `graph.py` which stores two classes, `Graph`, and `Vertex/Node` (remember those are two words for the same thing). `Vertex` or `Node` can be set up however you like. `Graph` should have:

- a constructor that takes the filename of a .dot file as an argument to its constructor, and does something useful with it, and
- a method `hopper(self, name, hops)` for which `name` is a vertex id, and `hops` is an integer indicating the number of allowed hops. This method should return a list of the names of all vertices which are `hops` distance from `name` or closer. For example, calling `g.hopper('0',1)` on the above graph should return the list `['0','1','2','3']`, in any order. Similarly, calling `g.hopper('3',2)` should return

`['3','4','5','6']`, again in any order. Hint: this is effectively a breadth first search, with the additional constraint that you stop searching after a certain number of hops. Be sure to reread the notes on BFS if you are stuck.

Be sure to test your program on the real dataset, as well (one common mistake will create an exception if you run `g.hopper('3',2)` on the real dataset). You may find Python's implementation of a Queue, deque (https://docs.python.org/3.3/library/collections.html#deque-objects), to be useful (Python's deque is a doubly-linked list).

**Part Two**: 100 points

Add the following methods to `Graph`:

- `canSurveil(self, suspect, target, hops)`, which returns `True` if `target` is within `hops` hops from `suspect`, and False otherwise. All three inputs should be integers.
- `percentage(self, suspect, hops)`, which returns a float indicating the percentage of individuals in the data set which can be reached by taking `hops` hops from `suspect`.
- `likelyPercentage(self, hops)`, which returns a float indicating the expected percentage of individuals which can be reached in `hops` hops from a random suspect. In other words, this is the average of the above method `percentage`, given an input of all valid ids as suspect.