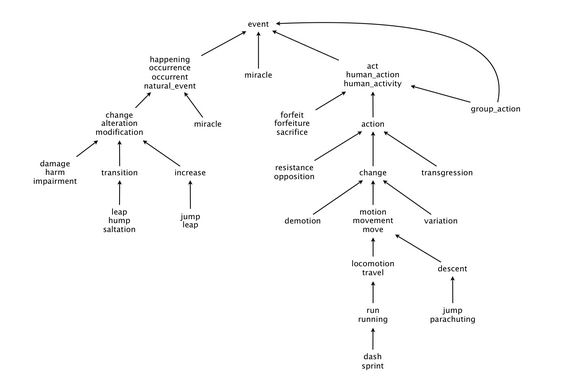
Program 7: CS 2420

Spring 2019 Graphs - Wordnet

# History: WordNet

Suppose you had a graph between sets of nouns in which an arc A -> B indicates that the set of nouns A is a more specific case of the set of nouns B. Such a graph would allow us to reason about similarities and differences. The graph below is an example of what this might look like.



**Measuring the semantic relatedness of two nouns**. Semantic relatedness refers to the degree to which two concepts are related. Measuring semantic relatedness is a challenging problem. For example, you consider Ronald Reagan and John F. Kennedy (two U.S. presidents) to be more closely related than Ronald Reagan and chimpanzee (two primates). It might not be clear whether Ronald Reagan and Eric Arthur Blair are more related than two arbitrary people. However, both Ronald Reagan and Eric Arthur Blair (aka George Orwell) are famous communicators and, therefore, closely related.

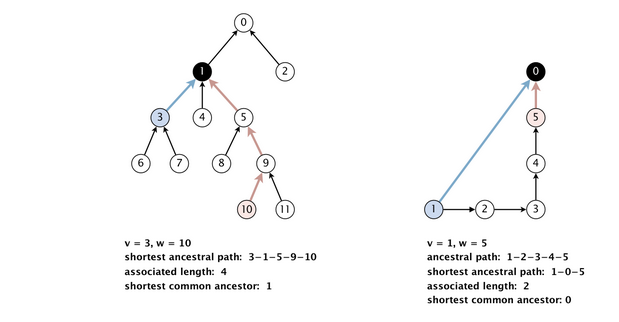
[WordNet](http://wordnet.princeton.edu/) is a semantic lexicon for the English language that computational linguists and cognitive scientists use extensively. For example, WordNet was a key component in IBM's Jeopardy-playing [Watson](http://en.wikipedia.org/wiki/Watson_(computer)) computer system.

# Your Assignment:

For simplicity, we’ll just use nodes that have a numeric name (rather than a set of strings), but the principle is the same. Given a directed acyclic graph, find the shortest path between any two nodes, the length of the path, and the shortest common ancestor.

You have been given starter code which has the test cases provided. As long as you keep the same test cases, you can change anything you want about the code.

**Shortest common ancestor.** An ancestral path between two vertices (v and w) in a rooted DAG (directed acyclic graph) is a directed path from v to a common ancestor x, together with a directed path from w to the same ancestor x. A shortest ancestral path is an ancestral path of minimum total length. We refer to the common ancestor in a shortest ancestral path as a shortest common ancestor. Note that a shortest common ancestor always exists because the root is an ancestor of every vertex. Note also that an ancestral path is a path, but not a directed path.



Hint (you do NOT need to do it this way):

Every ancestor of a node v (in the above example) needs to know

(1) the length of the shortest path to it from v

(2) the predecessor responsible for that shortest path. Storing the predecessor helps you in recreating the best path.

In my code, I called that information (known by each node) PathInfo.

Since we need to find the shortest path to a node from two different nodes, the code has two PathInfo variables. p1 stores the best path from one node and p2 stores the best path from another node.

So for example, for digraph1, lca(5,6) may store the following information:

digraph1.txt Best lca 5 6 Distance: 3 Ancestor 1 Path:5 1 3 6

The Graph digraph1.txt

0[2 Pred:1] [99 Pred:-1]

1[1 Pred:5] [2 Pred:3] 1->0

2[99 Pred:-1] [99 Pred:-1] 2->0

3[99 Pred:-1] [1 Pred:6] 3->1

4[99 Pred:-1] [99 Pred:-1] 4->1

5[0 Pred:5] [99 Pred:-1] 5->1

6[99 Pred:-1] [0 Pred:6] 6->3

7[99 Pred:-1] [99 Pred:-1] 7->3

8[99 Pred:-1] [99 Pred:-1] 8->5

9[99 Pred:-1] [99 Pred:-1] 9->5

10[99 Pred:-1] [99 Pred:-1] 10->9

11[99 Pred:-1] [99 Pred:-1] 11->9

**Outcast detection.** Given a list of nodes *x*1, *x*2, ..., *xn*, which node is the least related to the others? To identify *an outcast*, compute the sum of the path lengths between each node and every other one:

*di*   =   pathL*ength*(*xi*, *x*1)   +   pathL*ength* (*xi*, *x*2)   +   ...   +   pathL*ength( xi*, *xn*)

and return the node *xt* for which *dt* is maximum.

**Input:**

The input will be the number of nodes followed by the edges. Each edge A-> B is represented by A B.

|  |  |
| --- | --- |
| There are the two provided input files:  digraph1.txt  12 //Number of nodes  6 3 // an edge from 6 to 3  7 3  3 1  4 1  5 1  8 5  9 5 10 9 11 9  1 0  2 0 |  |

|  |  |
| --- | --- |
| digraph2.txt  25 1 0 2 0 3 1  4 1 5 2 6 2 7 3 8 3 9 3  10 1 10 5 11 5  12 5 12 6 13 7 14 7 15 9 16 9 17 10 18 10 19 12 20 12 21 16 22 16 23 20 24 20 13 3 9 4 23 4 |  |

**Testing:**

Run the tests shown in Graph.java.

The lca computation does the following. Given two nodes in the graph, find the shortest common ancestor, shortest ancestral path, and associated length.

The outcast computation prints the outcast (and, as debug, shows all the lengths that were computed in order to compute an outcast)

Sample Output:

(I printed the graph name so I didn’t get confused)

digraph1.txt Best lca 3 7 Distance: 1 Ancestor 3 Path:3 7

digraph1.txt Best lca 5 6 Distance: 3 Ancestor 1 Path:5 1 3 6

digraph1.txt Best lca 9 1 Distance: 2 Ancestor 1 Path:9 5 1

(I showed the computations involved in computing the Outcast)

digraph1.txt Best lca 7 10 Distance: 5 Ancestor 1 Path:7 3 1 5 9 10

digraph1.txt Best lca 7 2 Distance: 4 Ancestor 0 Path:7 3 1 0 2

digraph1.txt Best lca 10 2 Distance: 5 Ancestor 0 Path:10 9 5 1 0 2

digraph1.txt For [7, 10, 2] Outcast is 10 with distance sum of 10