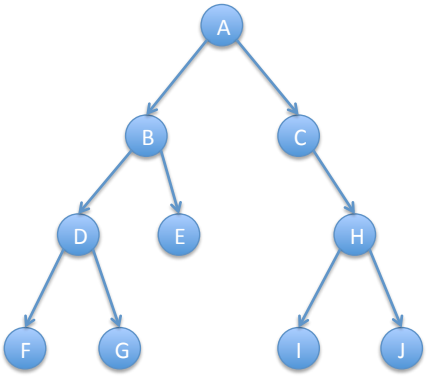
Methods in Computational Linguistics 2
Spring 2015

Homework #3 - Data Structures and Word Net
Due at 11:59pm on April 6

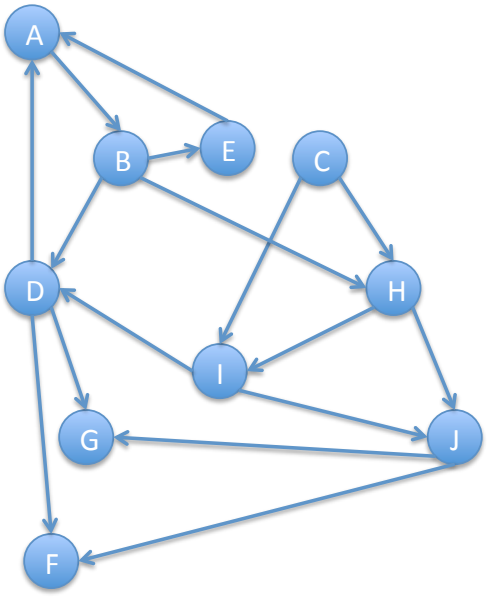
1. [5] Trees.

A. Write the order each node in the Tree is visited in a Depth First Search.
B. Write the order each node in the Tree is visited in a Breadth First Search.



2. [10] Graphs.

A. Write the order each node in the Graph is visited in a Depth First Search starting at node A.
B. Write the order each node in the Graph is visited in a Depth First Search starting at node B.
C. Write the order each node in the Graph is visited in a Breadth First Search starting at node A.
D. Write the order each node in the Graph is visited in a Breadth First Search starting at node B.



3. [30] Trees.

Write a Tree Class. It should include (minimally) members label, left, and right.

A. (15) Write a function that converts your Tree **to** a string of the following format. Each node is represented by 3 elements, recursively. It's label, a representation of its left child and a representation of its right child. The representation of an empty Tree (i.e. if the left or right child doesn't exist) should be 'None'

For example, the following tree would be represented by the associated string:

[1, [2, [4, None, None], [5, None, None]], [3, None, [6, None, None]]]

B. (15) Write a function that constructs a Tree **from** a string. This should be an exact inverse of the above function. Here the input should be the string, formatted as above, and the output should be a new Tree containing the corresponding elements.

4. [55] Open ended question: Wordnet and Word Similarity.

Wordnet includes the following functions to find the similarity between word senses:

```
wordnet.path_similarity
wordnet.lch_similarity
wordnet.wup_similarity
wordnet.res_similarity
wordnet.jcn_similarity
wordnet.lin_similarity
```

Consider the problem of extending this functionality to calculating the similarity between two sentences as a measure of coherence.

Write a program wordnet_sim.py that is called as follows.

```
wordnet_sim.py "This is the first sentence." "This is sentence two."
```

where the sentences can be replaced by any other sentence. The output of this function should be a single number representing the similarity between the two sentences.

A. (10) Write a function to compare all pairwise similarities between words from sentence 1 and sentence 2. That is, for each word in sentence 1, calculate the similarity to each word in sentence 2. This should give you n*m similarity measurements, where n is the number of words in sentence 1 and m is the number of words[1] in sentence 2.

---

[1] Specifically "words that have entries in wordnet"

B. (5) Define the similarity between the two sentences as the average pairwise similarity across all comparisons. (i.e. sum all of the n*m similarity measure and divide by n*m)
C. (10) Redefine the similarity between the two sentences as the average *best* similarity per word in sentence 1. To do this, for each word[2] in sentence 1, find the word in sentence 2 that it is *most* similar to. Define this as the best similarity score for that word. Take the average of these "best similarity scores". There should be n of them, one for each word in sentence 1.
D. (10) Try out each of these definitions on a variety (say at least 10 pairs) of sentences that you would consider to be similar and dissimilar. For example, sentences that follow each other in a news article, poem or narrative would be expected to be "similar", while words from different genres, times, or topics would be expected to be different. Report the similarity measures of these.
E. (10) Experiment with different similarity measures. (Hint: A list of functions would allow you to quickly and easily evaluate each of them.)
F. (5) Discuss your results. Is one definition of similarity better than another? Is either of them effective? Is one of the wordnet similarity functions better than the other?
G. (5) How else would you define similarity between sentences? Could this approach be extended? What aspects are missing here? What could be done to extend this approach?

[2] as before "word that has an entry in "wordnet"