

CMSC723: Computational Linguistics I

Assignment 1

Bonnie Dorr (professor), Nitin Madnani (co-instructor)
Hamid Shahri (TA)

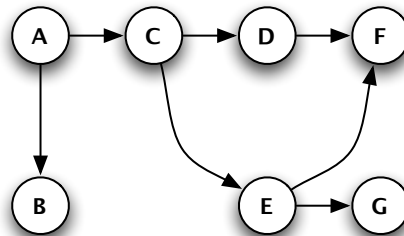
Out: **September 5, 2007**

Due: **September 19, 2007**

Note: This assignment's main purpose is to allow us to calibrate everyone's programming skills and design appropriate assignments for subsequent classes. If doing this assignment requires more than a few hours, then you might want to reconsider whether you have the programming background necessary for the class.

Introduction

A directed acyclic graph (DAG) consists of a set of nodes and a set of directed arcs (also known as directed edges) between those nodes. For example, the following is a DAG.



There are lots of ways to represent graphs. A typical choice would be to store, for each node, a list of all the nodes it's connected to via outgoing edges. For example, in Python one might use a dictionary:

```
edges = { 'a': ['c', 'b'],
          'c': ['d', 'e'],
          'd': ['f'],
          'e': ['f', 'g'] }
```

such that `edges['a']` is the list containing all nodes connected to node A.

Something you do frequently with DAGs is SEARCH: start at a node, and find another node. For example, in the above graph a search starting at D for target node F would succeed, but a search starting at D for target node B would fail. Search of this kind is the heart of a great deal of work in artificial intelligence; for example, game-playing programs (for tic-tac-toe, chess, checkers, backgammon, etc.) often view the game as a search for a winning board configuration, and parsing a sentence can be viewed as searching for a legal parse tree that fits the sentence.

There are different algorithms for searching a graph, but one of the most common is depth-first search (DFS). Algorithm 1 shows how depth-first search works on a graph.

Algorithm 1 DFS(S, T): An algorithm for depth-first search

Require: A starting node S and the target node T .

```
1: Create a list  $L$ , initially containing  $S$  as its only member
2: while  $L$  is not empty do
3:     Pop the top node  $x$  off  $L$  (think of  $L$  as a stack)
4:     if  $x$  is the target then
5:         Report success and quit — we found the target !
6:     else
7:         for each outgoing edge  $(x,y)$  of  $x$  do
8:             Put  $y$  on the front of list  $L$  (that is, push  $y$  onto stack  $L$ )
9:         end for
10:    end if
11: end while
12: Report failure —  $T$  can't be reached from  $S$  !
```

For the problems below, you are strongly encouraged to use Python. (Those who choose to use another programming language are strongly urged to convert their code to Python after the Python tutorial on September 12th.)

Problem 1

Implement the DFS algorithm. Please turn in: (a) the program listing, and (b) examples of the algorithm running on the above graph with two searches: the first search should start at **A** and look for **F** (success), and the second search should start at **C** and look for **A** (failure). Feel free to add statements to the program that print out messages as the search is in progress letting us know what's going on. We will not be grading the program itself, so informative messages are the best way to ensure we can give you partial credit.

Note: Graph information can be hard-coded in your program. However, the starting node and the edge node must be input as two command-line parameters. For example, suppose your DFS program (written in Python) is called `dfs.py`, and you want to search for target node **B** starting at node **A**. Your program must work with the following command line invocation:

```
dfs.py A B
```

Problem 2

Modify the algorithm and implementation so that cycles in the graph don't cause you to go into an infinite loop. Test this by modifying the above graph to add an edge from node **F** to node **C**, and then doing the same two searches as in Problem 1.

You can use the same program for both Problem 1 and Problem 2, e.g. distinguishing the two cases using an additional parameter on the command line. Just make sure you tell us the appropriate command lines to use.

Extra Credit

Modify the algorithm and implementation so that on success the program returns its path through the graph rather than just saying it succeeded. Demonstrate using both searches from Problem 1 and both searches from Problem 2.