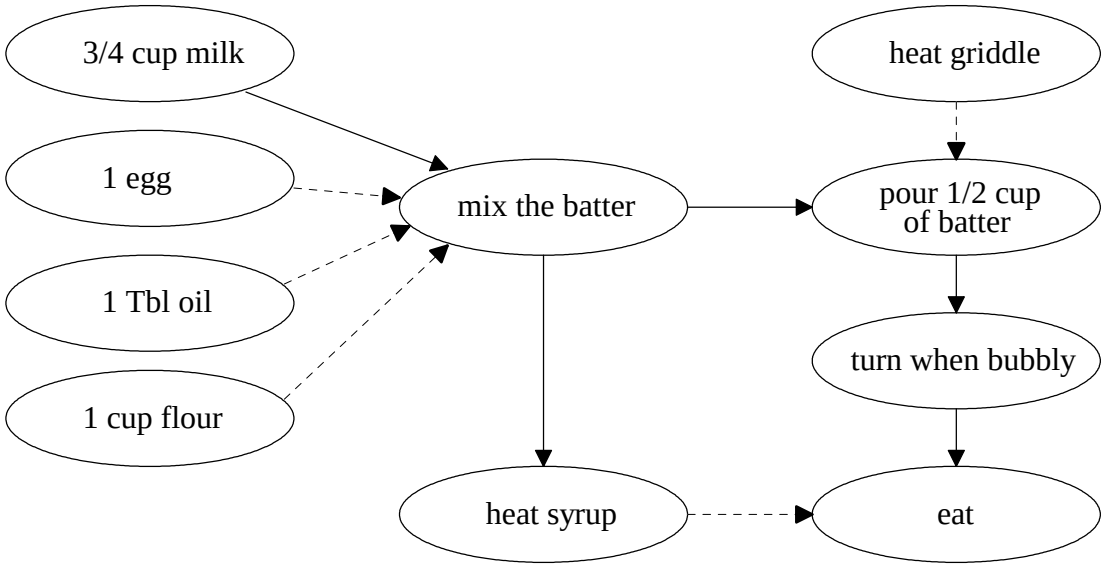
Data Structures Lab 12 Graphs Name:_____

Objectives: To understand how a graph can be represented and traversed. How we can use and implement graph algorithms for computing a topological sorting and strongly connected components.

To start the lab: Download and unzip the file at: www.cs.uni.edu/~fienup/cs052sum09/labs/lab12.zip

Lab A: The case study in section 20.9, discusses a menu-driven graph-testing program which uses the LinkedDirectedGraph class. The graph is input by specifying the directed edges as strings of the form: "p>q:0", where p and q are vertice labels with a directed edge from p to q with a weight of 0. Vertices are inferred from the edges, except for disconnected vertices that are strings of just vertex labels. Consider the graph for making pancakes. Vertices are ingredients or steps, and edges represents the partial order among the steps.



The file pancakes.txt has two lines, where the first describes the edges and the second containing a start vertex (needed for some algorithms).

```
milk>mix:0 egg>mix:0 oil>mix:0 flour>mix:0 mix>syrup:0 mix>pour:0 heat>pour:0 syrup>eat:0 pour>turn:0 turn>eat:0
milk
```

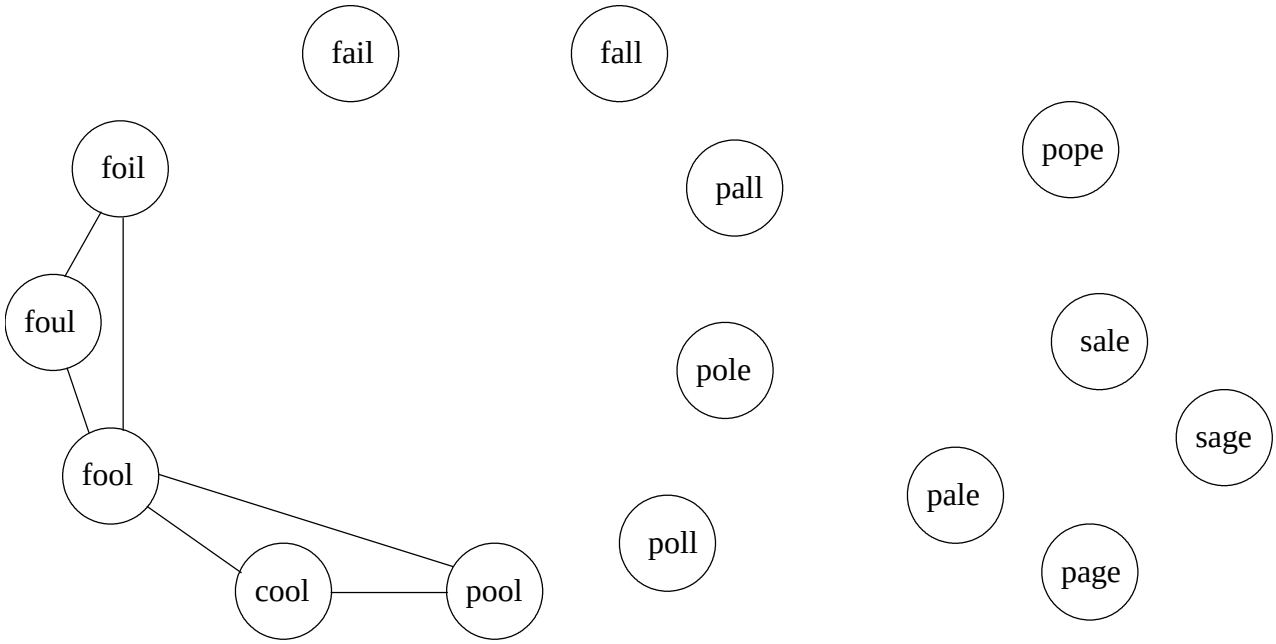
A topological sort algorithm (in the `algorithms.py` module) uses a recursive dfs to determine a proper order to avoid putting the "cart before the horse." For example, we don't want to "pour ½ cup of batter" before we "mix the batter", and we don't want to "mix the batter" until all the ingredients have been added. I've modified the `algorithms.py` module to include some additional print statements in the `topoSort` and `dfs` functions. Run the `view.py` program, load the graph defined in the `pancakes.txt` file (option 2), view the graph (option 3), and then perform a topological sort (option 6). Study the output and the code to answer the following questions:
a) Draw the recursion tree(s) for all calls to the `dfs` function performed during the `topoSort` function.

b) Define a function `bfs`, which performs a breadth-first search traversal of a graph, given a starting vertex. This function should return a list of vertex labels in the order that they are visited. Test the function thoroughly with the case study program.

Part B: In a word ladder puzzle you tranform one word into another by changing one letter at a time, e.g., transform FOOL into SAGE by FOOL → FOIL → FAIL → FALL → PALL → PALE → SALE → SAGE.

We can use a graph algorithm to solve this problem by constructing a graph such that
- words are represented by the vertices, and
- the edges connect words that differ by only one letter

a) For the words listed below, complete the graph by adding edges as defined above.



b) List a **different** transformation from FOOL to SAGE then the one given above

c) If we wanted to find the shortest transformation from FOOL to SAGE, what does that represent in the graph?

d) Which graph traversal algorithm (breadth-first search or depth-first search) would be helpful for finding the shortest solution of a word ladder puzzle?

e) Implement a word ladder puzzle solver. To aid input of the graph, the file `word_ladder.py` contains a function buildGraph that returns a list of edges (as tuples) for the words in the above example. The words.dat file contains the list of words for this example. (Untested) Hints:

- an undirected graph can be modeled by a directed graph by having an undirected edge being replaced by two directed edges between the same two vertices going each direction.
- a "distance" attribute associated with each vertex object might be useful to contain the shortest distance from the start node (which can be filled in by the graph traversal). A new vertex class that inherits the LinkedVertex can be created to include the distance attribute and associated assessor and mutator methods.