Objectives: To understand how a graph can be represented and traversed.

To start the lab: Download and unzip the file http://www.cs.uni.edu/~fienup/cs1520s20/labs/lab11.zip

Part A: In a word-ladder puzzle (discussed in class) transforms one word into another by changing one letter at a time, e.g., transform FOOL into SAGE by FOOL \rightarrow FOIL \rightarrow FAIL \rightarrow FALL \rightarrow PALL \rightarrow PALE \rightarrow SALE \rightarrow SAGE.

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

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

b) To find the shortest transformation from FOOL to SAGE, why did we decide on using a Breadth First Search (BFS) traversal (i.e., where you find all vertices a distance 1 (directly connected) from FOOL, before finding all vertices a distance 2 from FOOL, etc) instead of a Depth-First Search (DFS) traversal?

c) Run the lab11/word_ladder_BFS.py program. Examine the “enqueue” and “dequeue” lines of output produced by the bfs(g, g.getVertex("fool")) call. Does this output match the expected “enqueues” and “dequeues” performed during a bfs of the above graph starting at “fool”?

d) The bfs algorithm sets the value of each vertex’s predecessor to point to the vertex object that enqueued it. Add code to the end of the word_ladder_BFS.py program that traverses the “linked list” of predecessor references from “sage” to “fool.” and prints the corresponding word ladder from “fool” to “sage.”

Hint: The code you need to write is similar to the __str__ code for traversing a singly-linked list:

UnorderedList Object

_data   next_data   next_data   next_data   next
__head_ 'w'     'a'     'y'     'm'     'c'

]
def __str__(self):
    resultStr = ""
    current = self._head
    while current != None:
        resultStr += " " + str(current.getData())
        current = current.getNext()
    return resultStr

Your code (see partial code at the end of the word_ladder_BFS.py program) can:
- walk currentVert down the linked list of Vertex objects using getPred instead of getNext
- append to wordLadderList the currentVert’s word gotten using getId instead of string concatenating using getData

After your while-loop executes, you can reverse the wordLadderList and print the transformation from “fool” to “sage.”

After you have answered the above questions and completed the code, raise your hand and explain your answers.

Part B: Section 7.5 uses recursion and the run-time stack to implement a DFS traversal. The DFSGraph uses a time attribute to note when a vertex if first encountered (discovery attribute) in the depth-first search and when a vertex in backtracked through (finish attribute). Consider the graph for making pancakes where vertices are steps and edges represents the partial order among the steps.

```
from graph import Graph
class DFSGraph(Graph):
    def __init__(self):
        super().__init__()
        self.time = 0
    def dfs(self):
        for aVertex in self:
            aVertex.setColor('white')
            aVertex.setPred(-1)
        for aVertex in self:
            if aVertex.getColor() == 'white':
                self.dfsvisit(aVertex)
    def dfsvisit(self,startVertex):
        startVertex.setColor('gray')
        self.time += 1
        startVertex.setDiscovery(self.time)
        for nextVertex in startVertex.getConnections():
            if nextVertex.getColor() == 'white':
                nextVertex.setPred(startVertex)
                self.dfsvisit(nextVertex)
        startVertexsetColor('black')
        self.time += 1
        startVertex.setFinish(self.time)
```

3/4 cup "milk"
1/12
1 "egg"
1 Tbl "oil"
1 cup "flour"

![Graph diagram]

- "mix" the batter
- "pour" 1/2 cup of batter
- "turn" when bubbly
- "eat"
- heat "syrup"

a) Run the lab11/make_pancake_DFS.py program. Write on the above graph the discovery and finish attributes (e.g., 1 / 12 of “milk”) assigned to each vertex by executing the dfs method..

b) A topological sort algorithm can use the dfs finish attributes 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. Outline the steps to perform a topological sort from the finish attributes.

After you have answered the above questions, raise your hand and explain your answers.

EXTRA CREDIT:
Add code to the end of the made_pancake_DFS.py program to print the topological sort for making pancakes.