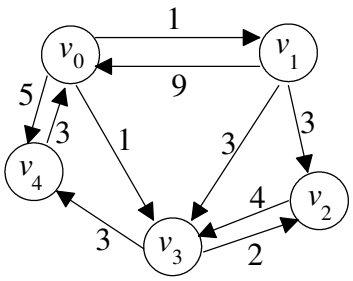
1. Consider the following directed graph (diagraph) $G = (V, E)$:



a) What is the set of vertices? $V =$

b) An edge can be represented by a tuple (from vertex, to vertex [, weight]). What is the set of edges?

$E =$

c) A path is a sequence of vertices that are connected by edges. In the graph G above, list two different pathes from v_0 to v_3.

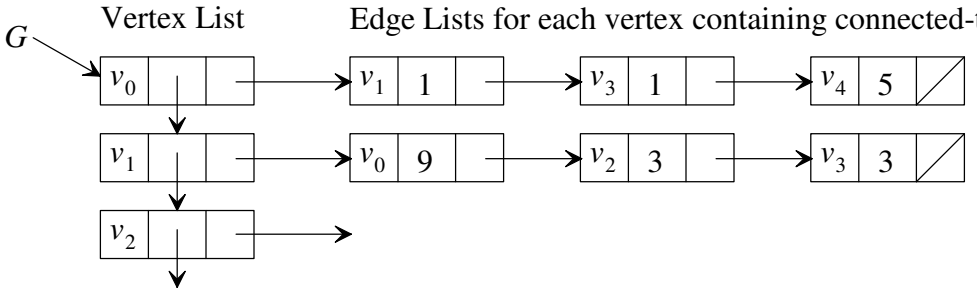
d) A cycle in a directed graph is a path that starts and ends at the same vertex. Find a cycle in the above graph.

2. Like most data structures, a graph can be represented using an array, or as a linked list of nodes.
The array representation is a two-dimensional array (called an *adjacency matrix*) whose elements contain information about the edges and the vertices corresponding to the indices. (Python could use a list-of-lists)

a) Complete the following adjacency matrix for the above graph. (Here a missing edge is represented by ∞.)

(from vertex) \ (to vertex)	v_0	v_1	v_2	v_3	v_4
v_0	0	1	∞	1	5
v_1	9	0	3	3	∞
v_2					
v_3					
v_4					

b) The linked representation maintains a linked-list (or Python dictionary) of vertices with each vertex maintaining a linked list of other vertices that it connects to. Complete the adjacency list representation below:



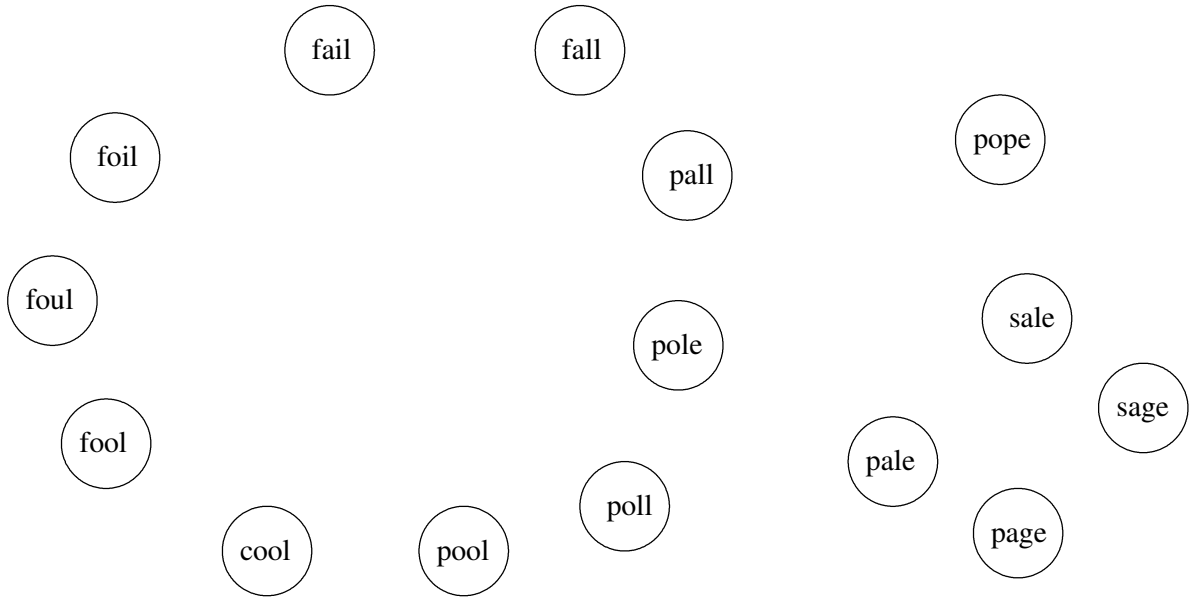
3. Graphs can be used to solve many problems by modeling the problem as a graph and using "known" graph algorithm(s). For example, consider the *word-ladder puzzle* where you tranform one word into another by changing one letter at a time, e.g., transform FOOL into SAGE by FOOL → FOIL → FAIL → FALL → PALL → PALE → SALE → SAGE.

We can use a graph algorithm to solve this problem by constructing a graph such that

- a word represents a vertex
- an edge represents?

- a word ladder transformation from one word to another represents?

4. For the words listed below, draw the graph of question 3



a) List a different transformation from FOOL to SAGE

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

c) There are two general approaches for traversing a graph from some starting vertex *s*:

- Breadth First Search (BFS) where you find all vertices a distance 1 (directly connected) from s, before finding all vertices a distance 2 from s, etc.
- Depth First Search (DFS) where you explore as deeply into the graph as possible. If you reach a "dead end," we backtrack to the deepest vertex that allows us to try a different path.

Which of these traversals would be helpful for finding the **shortest** solution to the word-ladder puzzle?