# Homework 7: Graph Traversal and Components

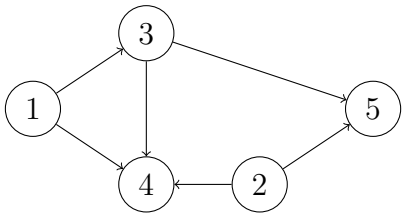
Due: Friday, March 27 at 11:59 pm on Canvas

**Concepts:** graphs, graph traversal, random graphs

## Theory

1. (6 points) A topological ordering (also called topological sort) of a directed graph $G = (V, A)$ is an ordering of the nodes in $G$ such that for all directed arcs $(u, v)$, $u$ comes before $v$ in the ordering. An example is given below where the node label is the ordering. Explain how you could either find a valid topological ordering or determine that no such ordering exists. Include pseudocode for your algorithm and be sure to explain the logic behind your ordering.

   **Hint:** It may help to know that a topological ordering exists if and only if there is not a cycle in $G$ (think about why this is true).




## Practice

2. (5 points) Write a function that finds the number of components in an undirected graph $G = (V, E)$ and write a corresponding test function. **Hint:** Think about using depth-first or breadth-first search to find a single component.

   For your graph representation, I recommend that you use the library `networkx`, but you should not rely on any `networkx` algorithms for this implementation. A snippet of code using this library is shown below.

```
import networkx as nx

G = nx.Graph()
G.add_node(1) # adds node with label 1
G.add_node(2) # adds node with label 2
G.add_edge(1,2) # adds an undirected edge between nodes 1 and 2

G.edges() # returns an iterator over all edges
G.nodes() # returns an iterator over all nodes
G.neighbors(1) # returns the neighbors of node 1
```

3. (5 points) A *random binomial graph* $G(n, p)$ is a graph on $n$ nodes such that for every pair of nodes $(i, j)$ there exists an edge between $i$ and $j$ with probability $p$. For $n = 5, 6, \ldots, 50$, find the smallest probability $p$ (that is a multiple of 0.01) such that ten generated binomial random graphs $G(n, p)$ each consists of a single component. Plot this probability as a function of $n$ and comment on the results.

   To create the random graphs, it may be useful to use `np.random.uniform`.