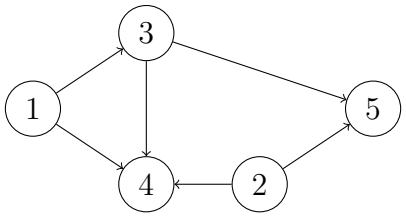
Homework 7: Graph Traversal and Components

Elias Gabriel

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).



Solution: As a pre-processing step, we can determine if finding a topological ordering is possible by using a depth-first-search. If at any point in the DFS we reach a node that we've already seen, then we know that a cycle exists and that no topological ordering is possible. No ordering is possible since, by definition, any given node would be its own child, and thus no ordering of the nodes would result in a non-cyclic hierarchy.

If we know that there are no cycles, we can find a valid topological sort by focusing on the edges. Starting with a set of nodes, we can loop through the edges one at a time following these steps:

(a) Given edge `(i, j)`, check if `i` comes before `j` in our set of nodes

(b) If `i` does come before `j`, we can continue. Otherwise, move `i` to the position directly before `j` in the set, preserving the order of the rest of the set.

By going through each edge and, critically, preserving the order of the work we've already done in steps prior, we can be sure that by the end of the iteration our new set has been ordered topologically. Replicating that algorithm in pseudocode:

```
let S = the list of nodes in G
for (i,j) ∈ arcs of G:
    if index of i > index of j:
        move i before j such that (index of j)-1 == index of i
```

If we wanted to avoid the overhead of pre-processing, we can simply run through the algorithm twice. The first time, we assume we are sorting it correctly. If we then check each edge ordering again and find a failure, we know that a cycle exists and that a topologically sorted graph is not possible.

Practice

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.

Solution: As the number of nodes in the graph increases so does the probability required to make the graph connected decreases. This makes sense, as even at low per-edge probabilities, there are more opportunities for a path to form between two nodes as the number of possible edges out of a node increases. Interestingly, this relationship doesn't seem to follow a linear scale. Rather, the long tail and constant log-log slope seem to suggest that it follows some power law. While not 100% certain, I've extended the x range and plotted it on a log-log scale for clarity.