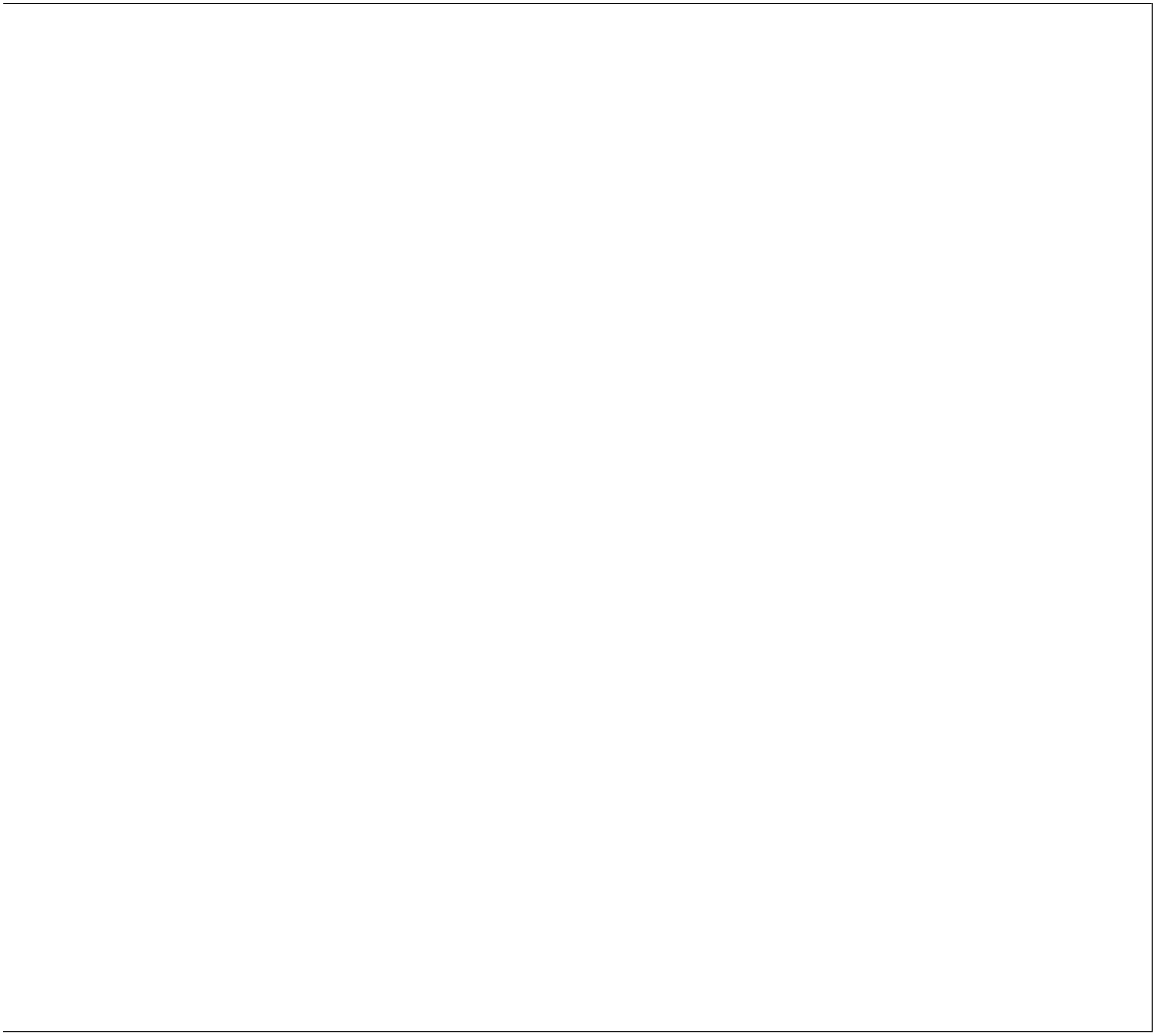
Precept Outline
- Non-Recursive DFS
- Tarjan's SCC Algorithm
- Algorithm/Data Structure Design

A. Strongly Connected Components and Tarjan's Algorithm

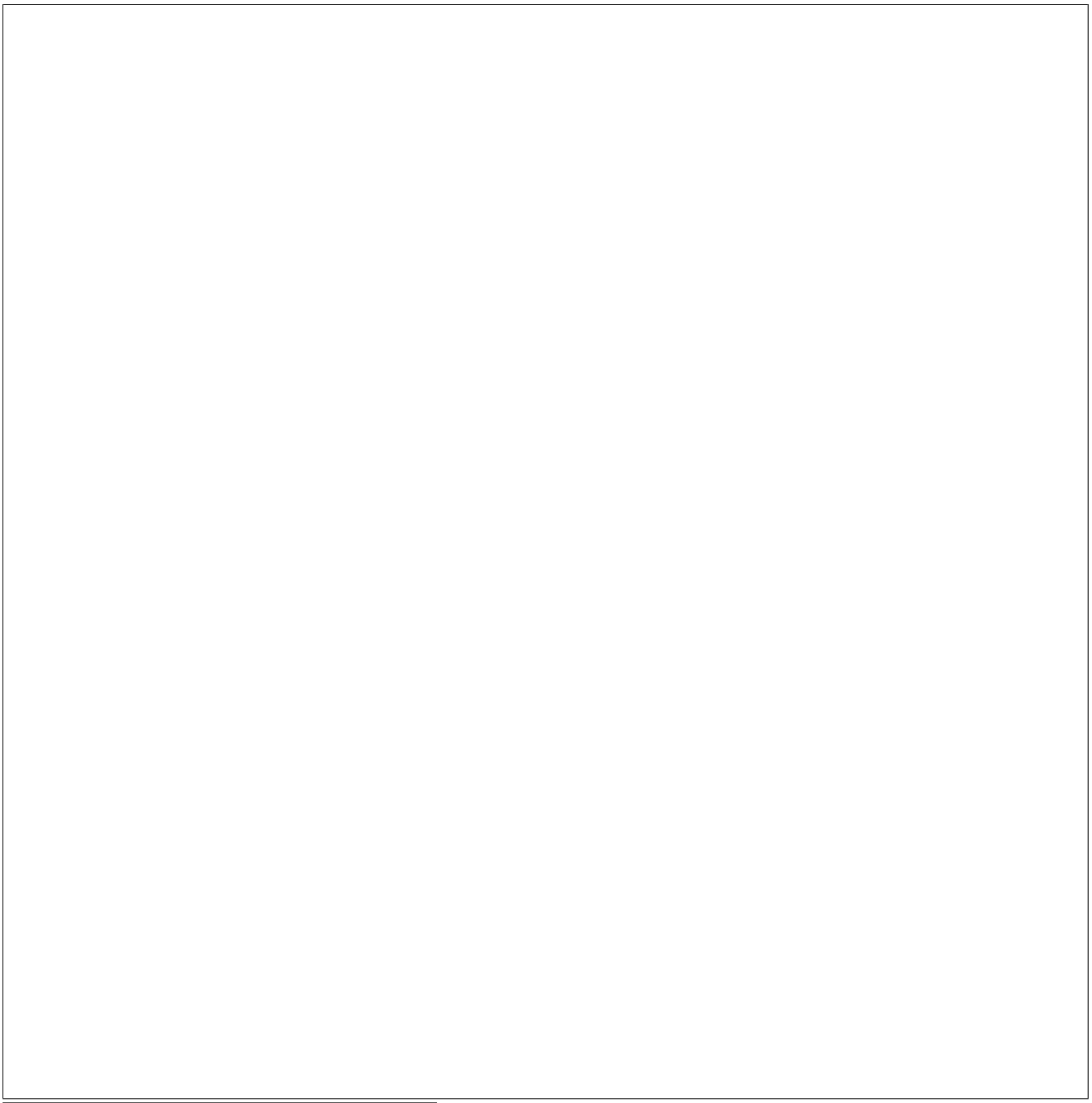
DFS is a surprisingly versatile algorithm which, with small modifications, can solve problems far beyond exploring a graph. We've already seen one such application in lecture (finding a topological order), and will learn another here: finding the *strongly connected components* of a directed graph.

Let's start with a simpler problem (which will also be useful in the WordNet assignment, and is a building block towards an SCC algorithm): find a variant of DFS that detects if a graph contains a directed cycle. *Hint: consider the vertices in the function call stack.*



A strongly connected component of a graph is a set of vertices with paths between any pair in the set; equivalently, two vertices are in the same SCC if and only if there is a directed cycle (possibly with repeated nodes) containing both. Every graph can be partitioned into SCCs.[1]

Design a variant of DFS that finds all connected components of a graph; more precisely, populate an `id[]` array so that `id[v] == id[w]` if and only if `v` and `w` are in the same SCC. (This is known as *Tarjan's algorithm*.) *Hint: as a first step, consider how to augment the cycle detector to a cycle finder.*



[1] For those interested in why and the math behind it: the condition "contained in the same directed cycle" is an equivalence relation on vertices, so SCCs are equivalence classes.

B. Non-Recursive DFS

We've seen a recursive implementation of DFS and a non-recursive one of BFS in this course. A natural question to ask is whether we can make DFS non-recursive (both as an intellectual curiosity, and to make it more efficient).

Let's start with a warm-up: show that replacing `Queue<Integer>` with `Stack<Integer>` in `BreadthFirstDirectedPaths` (as well as enqueues/dequeues by pushes/pops) yields a graph search algorithm that is not DFS. In particular, find a graph such that neither the sequence or pushed nor the sequence of popped vertices under this algorithm matches the DFS preorder.

Design a non-recursive linear-time DFS using a `Stack<Integer>` auxiliary object. Your stack may store $O(E)$ space.

Design a non-recursive DFS linear-time using one or more auxiliary stacks and arrays with $O(V)$ worst-case space requirements. *Hint: mimic the recursive DFS implementation.*

C. Algorithm Design

This problem was adapted from the Fall'24 Final Exam.

Design an algorithm that, given a graph G, a start vertex s, and a target vertex t, determines whether there exists a directed path of even length from s to t.

The algorithm should run in $O(E + V)$ time in the worst case, where V is the number of vertices and E is the number of edges in G.