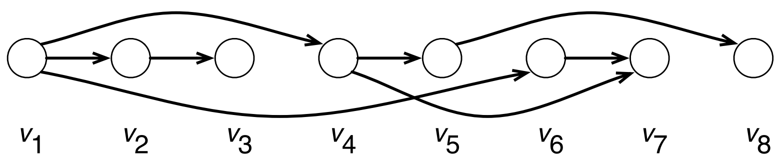
Topological Ordering via Depth-First Search

Let G be a directed graph. A topological ordering of G is an ordering $v_1, \ldots, v_n$ of its vertices such that for every edge (v_i, v_j) we must have $i < j$. Example:



1. If G contains a cycle, then it does not have a topological ordering. Why?

2. For any two vertices u, v, if u can reach v by some path then every topological ordering of G must have u appearing before v. Why?

3. Create an ordered list of vertices using a depth-first search where a vertex v is added to the list the moment after all neighbours of v are recursively explored. In Python:

```
def do_dfs(curr, prev):
    if curr in reached:
        return
    reached[curr] = prev
    for succ in g.neighbours(curr):
        do_dfs(succ, curr)
    order.append(curr) # new part for this worksheet
```

   Here, `order` is in the same scope as `reached` and is initially `[]`.

   Consider the contents of `order` after one depth-first search. Show that if G does not have a cycle, then there is no directed edge (u, v) such that u appears before v in `order`. Thus, reversing `order` would produce a topological ordering of all vertices that were reached in the search.

   Hint: If such an edge (u, v) did exist, argue that v is already visited but is still on the recursion/call stack when u is placed in `order`. Conclude there must be a path from v to u in G.

4. Use these ideas to design an algorithm that will find a topological ordering for any directed graph that does not contain a cycle, even if no single vertex can reach all others.

   $O(|V| + |E|)$ running time is possible. **Hint**: try the above depth-first search modification on an example where the start vertex cannot reach all other vertices. What can you do once this search is done to order the remaining vertices?