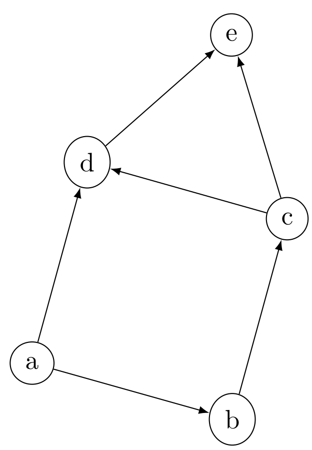
Topological Sort

1. Describe a method that checks whether or not a given permutation of a DAG's vertices is a topological ordering of that DAG.

> **Solution:** Iterate through all edges in the DAG. For each edge $v \to w$, ensure that v appears before w in the permutation. If not, the permutation is not a topological ordering of the DAG.

2. A colleague suggests an alternate method to find the topological ordering of a DAG: run BFS, and label the vertices by increasing distance to their respective source. Explain why your colleague's algorithm won't necessarily always produce a true topological ordering.

> **Solution:** BFS alone won't be able to always produce a correct topological ordering of a directed graph. Any graph where a vertex at a further away level has a connection to a vertex at a closer level to the root will cause BFS to fail in producing a correct topological order. This is because BFS will take the closer level vertex into the topological ordering before it can see that there is a connection to it from a further away level vertex. This is why DFS is absolutely necessary in any topological sort algorithm.
>
> Take, for example, the graph below. Using BFS, with a source vertex at node `a`, node `d` would come before `c`. This would not be a correct topological ordering.
>
> 
>

3. Design an algorithm to determine whether a DAG has a unique topological ordering.

   Hint: If the DAG has multiple topological orderings, then a second topological order can be obtained by swapping a pair of consecutive vertices.

> **Solution:** Run the standard topological sort algorithm to obtain a topological ordering of the graph. For each pair of vertices in the topological order, swap them. If the swap produces a valid topological ordering, the graph doesn't have a unique topological order.

Strongly Connected Components

4. Describe what the strongly connected components of a directed graph are.

Solution: Two vertices, v and w, are strongly connected in a directed graph if there is a path from v to w and a path from w to v. A strongly connected component is a set of vertices within the graph that are all strongly connected to each other.

5. A colleague doesn't believe that the the strongly connected components in G^R are the same as in G. How can you prove your colleague wrong?

Solution: Strongly connected components depend on the set of strongly connected vertices in a graph. If in the original graph, node v is connected to w but w is not connected to v, then v and w are not strongly connected. Reversing the edges that make up this path will cause w to be connected to v but v to not be connected to w. If vertices v and w were not strongly connected before the reversal, they will not be strongly connected after the reversal. Therefore, the set of vertices a vertex is strongly connected to doesn't change after edge reversal in a DAG. Therefore, the strong components of a DAG also do not change after edge reversal.

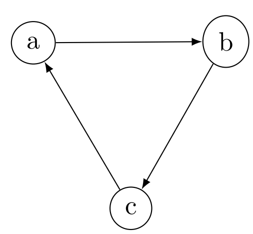
6. Why do we need Kosaraju's algorithm? Why can't we simply use DFS or BFS on a DAG to determine strongly connected componets like we do in undirected graphs?

Solution: A simple DFS or BFS on a DAG will not determine strongly connected components. For example, suppose v is connected to w but w is not connected to v. DFS passing through v would include w in the same component, even though v and w are not strongly connected.

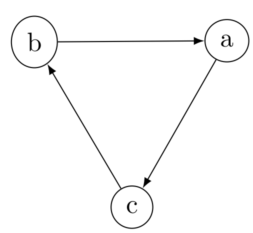
7. The reverse postorder of a graph's reverse is the same as the postorder of the graph. Is there any truth to this statement? Why or why not?

Solution: No, the reverse postorder of a graph's reverse is not the same as the postorder of the graph. In order to understand why, we must remember that we are dealing with a directed graph. After edge reversal, paths that vertex v had as an option to take are no longer an option. Therefore, the order in which vertices are visited will change.

For example, take the following graph where `dfs(a)` returns a postorder of `c`, `b`, `a`.



In contrast, the following graph is the transpose (edges reversed) of the graph above. Running `dfs(a)` would return a postorder of `b`, `c`, `a`, the reversal of which is `a`, `c`, `b`. This is not equal to the postorder of the original graph above.



8. Why is it necessary to first transpose the graph (reverse the edges) in Kosaraju's algorithm? Could the algorithm have been designed to not require the transpose?

Solution: Kosaraju's algorithm relies on a transpose of a graph for DFS to find all the vertices that are reachable from root vertex v. A DFS performed on the original graph is then used to find all vertices that can reach v. An intersection of those two vertices yields the strong components. Therefore, the algorithm would not work without the transpose. The transpose handles one of the directions of strong connection.