1 Directed Graphs

A directed graph an be thought of as a graph of dependencies.

Definition 1.1: Topological Ordering

A **topological ordering** of a directed graph is an ordering of the vertices so that for each edge (v, w), v comes before w in the ordering.

1.1 Cycles

Definition 1.2: Cycle

A cycle in a directed graph is a sequence of vertices $v_1, v_2, \ldots, v_n$ so that there are edges:

$$(v_1, v_2), (v_2, v_3), \ldots, (v_n, v_1)$$

1.1.1 Obstacle

Proposition. *If G is a directed graph with a cycle, then G has no topological ordering.*

Proof. Suppose we have a cycle $v_1, \ldots, v_n$. Assume for the sake of contradiction taht we have an ordering. Then, we can find the earliest v_i in the ordering. But, the v_i comes before v_{i-1}, in contradiction to the order property. □

1.2 Directed Acyclic Graph (DAG)

Definition 1.3

A **directed acylic graph** (DAG) is a directed graph which contains no cycles.

The previous result said that only DAGs can be topologically ordered. However, is the reverse true? **Yes**.

1.3 Existence of Orderings

Theorem 1.1

Let G be a finite DAG. Then, G has a topological ordering.

Proof. We consider the last vertex in the ordering. This must be a sink, or a vertex with no outgoing edges. So, once we find the sink, we can put the graph at the end of the topological graph, and then order the remaining vertices. □

1.3.1 Sinks

Lemma 1.1

Every finite DAG contains at least one sink.

Proof. Start at a vertex $v = v_1$. Then, we can "follow the trail," or in other words follow the edges. Eventually, we will either find:

- Some vertices repeat (which creates a cycle).
- Gets stuck (found a sink).

So, we are done. □

1.4 Algorithm

Suppose we want to design an algorithm that, given a DAG G, computes a topological ordering on G. We can use the proof to create a naive algorithm.

```
TopologicalOrdering(G)
    If |G| = 0
        Return {}
    Let v in G
    While there is an edge (v, w)
        v = w
    Return (Ordering(G - v), v)
```

The runtime is $O(|V|^2)$. This is because we need $|V|$ time to find each sink and have $|V|$ sinks. This is suboptimal, however.

```
TopologicalOrdering(G)
    Run DFS(G) w/ Pre/Post Numbers
    Return Vertices in Reverse Postorder
```

This runs in $O(|V| + |E|)$.

1.5 Topological Sort

This is a particularly useful algorithm.

- Many graph algorithms are relatively easy to find the answer for v if you've already found the answer for everything downstream of v.
  - We can topologically sort G.
  - Then, solve for v in reverse topological order.

1.6 Connectivity in Digraphs

In undirected graphs, we had a very clean description of reachability: v was reachable from w if and only if they were in the same connected component. Well, this no longer works for digraphs.