1 Basic review of dfs and bfs

The following is the basic algorithm for dfs

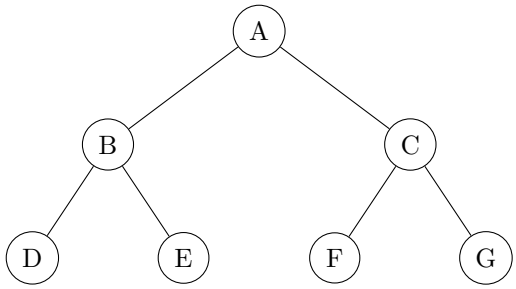
Algorithm 1 DFS-Recursive

```
1: procedure DFS(G, v, visited)
2:     mark v as visited
3:     process(v)
4:     for each neighbor u of v in G do
5:         if u is not visited then
6:             DFS(G, u, visited)
7:         end if
8:     end for
9: end procedure
```

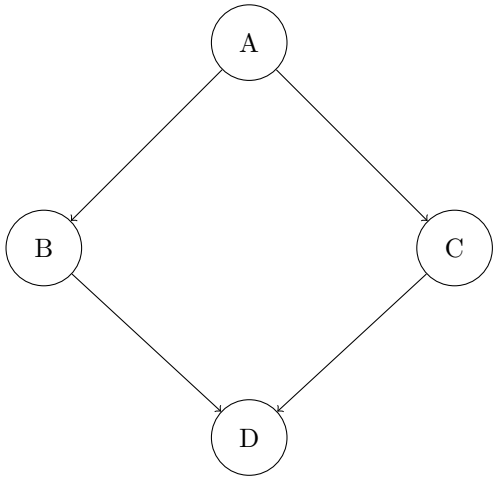
2 DAGs

Directed Acyclic graphs are basically as the name suggest uni-directional(direct) and acyclic. Example of DAGS are(edges are uni-directional from top to bottom):

1. Trees/Forests. All trees are acylic by definition. These are the most common DAGs.



2. Non-tree examples:



Essentially, DAGS can be seen as a generalization of trees; much like how squares are rectangles, rectangles are not square.

2.1 Detecting DAGS

Let us start from a simpler problem, how do you tell if a graph is a tree? That is how do you tell that example 1 is a tree algorithmically while simulatenously detecting that

2 is not a tree? We know three properties of trees must hold for all trees. They are the following:

1. The graph is connected
2. The graph is acyclic
3. The graph has n-1 edges

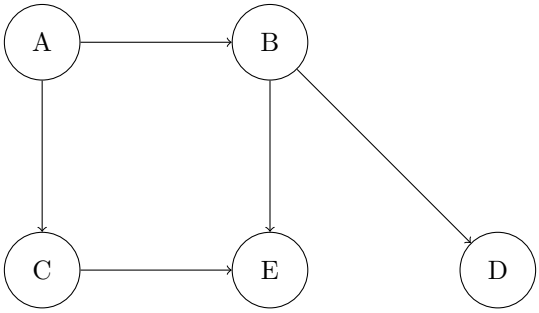
But DAGs do not satisfy the trees by the (3) condition.

2.2 DFS and DAGS

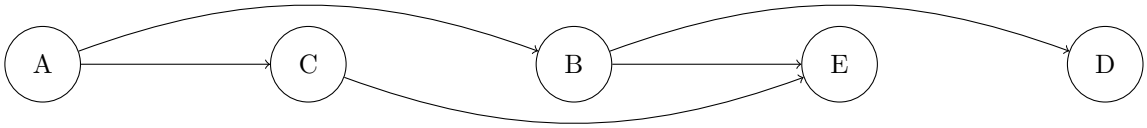
There is a clever simplification to DFS when you are given a DAG. You don't need to store a visited array if you traverse a DAG. **Why?**

2.3 Topological Sort

When you are given a DAG you are able to topologically sort the graph.



To



Note: The proper definition of a topological sort is that there exists a order

$$v_1, v_2, v_3, ..., v_n$$

of the vertices of the graph such that for all the in-flowing edges to v_b they must have come before from v_a such that $a < b$.

True or False: A graph is a DAG if and only iff it is topologically sortable.

This is an iff so it must be true both ways.

1. $\Rightarrow$ A graph is a DAG then it must be topologically sortable. This is true because we have an algorithm for it.
2. $\Leftarrow$ A topologoical sort exists on a graph then it must be a DAG. This is true too. The proof for this is like the proof for BFS being the shortest path. Say for contradiction that a topologoical sort does exist on a non-DAG graph with a cycle. The cycles is $v_a, .., v_x$. Recall the definition of topological sort. An order $v_1, v_2, ...v_n$ exists. But, if without loss of generality, v_a comes before v_x in the sort, then there is an edge going to something smaller than it. Thus it is not a topological sort.

3 Writing topological sort

A lot of ways to write topological sort.

Algorithm 2 DFS-Recursive

```
1: Create an array storing the indegree counts
2: call DFS(G,u) on all the vertices with indegree 0
3: procedure DFS(G, v)
4:     push v to the list
5:     process(v)
6:     for each neighbor u of v in G do
7:         if u is not visited then
8:             decrement the indegree array by 1 for v
9:             if the indegree is now 0, DFS(G, u)
10:        end if
11:    end for
12: end procedure
```

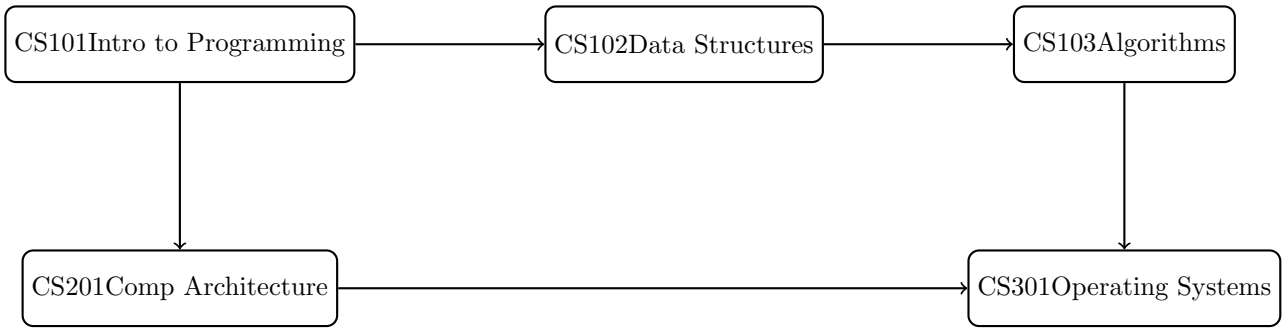
Note: How would you write topological sort by modifying BFS?

Order doesn't really matter here so there are a lot of way to write it.

4 Usages of topological sort

4.1 Scheduling algorithms

A common usage of topology sort is to schedule stuff. For example, a class prerequisite can be easily solved with topological sort.



4.2 DP usages

With DP you need to have like some induction structure which is topological. Like if you are at t_i, every $j < i$ must have been answer.

5 Problems:

1. 2020 Gold

2. 2018 GOLD