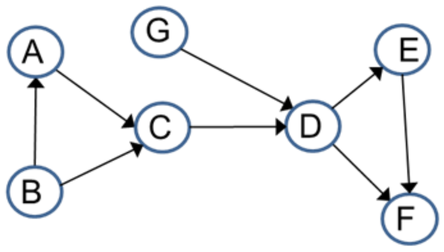
Lab 2-1 A: Graph Applications – DFS applications

Goal: Deepen understanding of the power of DFS

1. Use the Source Removal algorithm on the following graph showing the state of the calculation as each vertex is added to the linear ordering produced by the algorithm. Break any ties by using the vertex that is first in alphabetical order. Show the steps as was done in the screencast to reinforce you understanding of how the algorithm works by filling in the table with the in-degree values as the algorithm progresses.



	Initialize in-degree	Remove __B__	Remove __A__	Remove __C__	Remove __G__	Remove __D__	Remove __E__	Remove __F__
A	1	0	order					
B	0	order						
C	2	1	0	order				
D	2	2	2	1	0	order		
E	1	1	1	1	1	0	order	
F	2	2	2	2	2	1	0	order
G	0	0	0	0	order			

2. When does a directed graph have a unique topological ordering?

   There must be a directed edge between each pair of consecutive vertices. A linear path that contains all vertices must exist in the directed graph for it to have a unique topological ordering.

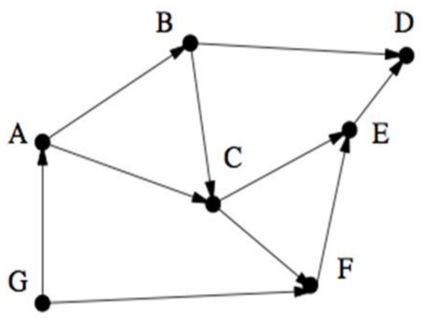
3. Show (make a convincing argument) that the condition you gave in #2 implies there is a unique topological ordering. Present your argument to your team and work with them to strengthen it.

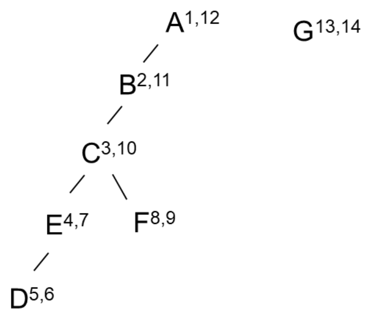
   If there is a linear path that contains all vertices, then there is a directed edge from vertex v to v + 1 for each vertex. Therefore, if I were to remove the source vertex v with in-degree of zero, vertex v + 1 now becomes the next source vertex. This ensures that there is only one source vertex at a time since v + 1 will have an in degree > 0 until v is removed. If there is only one source vertex at a time, then there is only one unique topological ordering for that directed graph.

Depth first search gives us another way of finding a topological sort of the vertices of a DAG. Consider the post numbering of the DAG below, if the DFS starts at A. Consider the graph from the screencast, and perform DFS on the graph for practice.

What do you notice about vertices A and G? What about vertex B?

They have the highest post order numbers.





This hints that perhaps higher post numbers imply that a vertex comes earlier in the any topological sort of the vertices. Recall that the pre and post numbers associated with DFS tell us something about the type of an edge determined during the depth first search. Namely:

Let (u,v) be an edge from u to v. Let u_{pre}, u_{post} and v_{pre}, v_{post} denote the pre and post numbers for vertex u and v respectively

- if the interval for v is contained entirely in the interval for u -- $\mathbf{u_{pre}, v_{pre}, v_{post}, u_{post}} \rightarrow$ the edge from u to v is either a ***tree* or a *forward*** edge, said another way v is a *descendent* of u.
- if the intervals are entirely disjoint $\mathbf{v_{pre}, v_{post}, u_{pre}, u_{post}}$, $\rightarrow$ the edge from u to v is a **cross edge**, neither is a descendent of the other.
- if the interval for u is contained entirely in the interval for v -- $\mathbf{v_{pre}, u_{pre}, u_{post}, v_{post}}$, $\rightarrow$ the edge from u to v is a ***backward*** edge, v is an *ancestor* of u.

In a DAG there are no backward edges hence the last possibility above cannot occur. Thus for an edge from u to v, $\mathbf{u_{post}}$ must be larger than $\mathbf{v_{post}}$. This means that in a DAG the vertex with the highest post number must be a source vertex!! (Stop and discuss in your group to make sure you understand why.) In addition, it means that if that vertex and its edges were removed from the DAG that the vertex with the next highest post number would now be a source. This implies the following important fact.

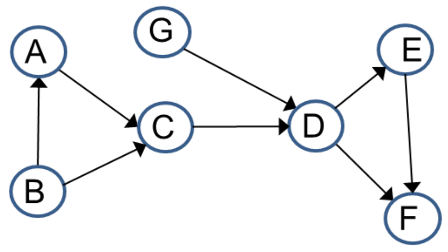
Theorem: Ordering the vertices of a DAG G in the order of the post numbers from highest to lowest results in a topological sort of the vertices !!

Thus we have the following algorithms to topologically sort the vertices of a DAG.

1. Perform DFS traversal, noting the order vertices are popped off the traversal stack
2. Reverse order (list vertices with highest post number first) solves topological sorting problem
3. Back edges encountered?→ NOT a DAG! No topological sort for the vertices of the graph exists.

4. Apply this DFS based algorithm to topologically sort the vertices of the following DAG. Show the DFS forest with the vertices labeled by their **pre and post numbers. Start using vertex A and assume that the adjacency list of each vertex is alphabetically ordered.**

On Picture below



Lab 2-1 B: Solving problems using graphs

1. **Finding shortest cycle in a undirected graph** Here is a proposed algorithm for finding the shortest cycle in an undirected graph with unit edge lengths.

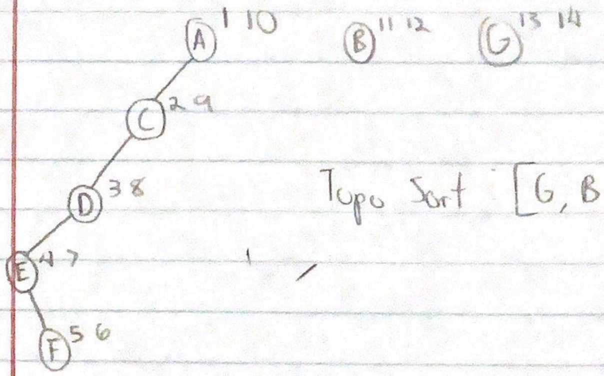
When a back edge, say (v,w) is encountered during a depth first search, it will form a cycle with the tree edges from w to v. The length of the cycle will be (level[v] – level[w] + 1, where the level of a vertex is its distance in the DFS tree from the root vertex. This suggests the following algorithm:
-- do a depth first search, keeping track of the level of each vertex
-- each time a back edge is encountered, compute the cycle length using the above formula and save it if it is smaller than the shortest one seen so far

a) Give a counterexample showing this algorithm does not work in all cases.
Hint: - Consider a graph whose DFS tree has two back edges that go from a parent-child pair of vertices to a vertex much lower in the tree that is a descendent of both vertices in the DFS tree.
On picture below.

b) Give a short 2 to 3 sentence explanation of the general reason why it will not work.
Hint: What is the underlying assumption of using the length of the shortest cycle formed during the traversal to find the shortest cycle in the original graph?

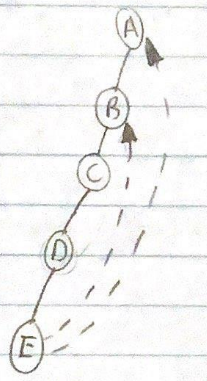
This algorithm assumes that the shortest cycle is in the DFS path. As shown in the counterexample, there is a shorter cycle that exists if you take the path using the back edges and not just the paths that the DFS creates. DFS does not give you every path, it only searches through every vertex.

4)



Topo Sort [G, B, A, C, D, E, F]

1)



By the formula, you would get shortest cycle to be [B, C, D, E, B]. But the true shortest cycle is [A, B, E, A].

2. **One-way Streets** A city has decided to make all its streets one-way. As you can imagine this has been controversial. The mayor contends that after the change it will still be possible to drive from any intersection to any other intersection in the city, but the opposition is not convinced. A computer program is needed to test whether the mayor is correct. However a vote is coming up and there is only time to run a "linear-time" algorithm.
   a) Formulate this problem as a graph problem and explain how to solve it in linear time.
   b) Suppose it turns out that the mayor's claim is false. Now she claims that if you start driving from town hall, then no matter where you get to (driving on the one way streets) there is always a way to get back to town hall legally. Formulate this as a graph problem and show how to solve it in linear time. Doing this in linear time is the hard part. After your group has thought about this for 5-10 minutes, you may want to ask for a hint.

Ignore Part C for this lab.

Lab 2-1 C: Finding Strongly Connected Components using DFS

Goal: understanding connectivity in directed graphs

TBD later

Add a brief description of how to use DFS to find the strongly connected components of a graph

5. Simple example of finding connected components

Definition: The reverse graph of a directed graph G = (V,E) is a directed graph G