SCHOOL OF ENGINEERING

VANDERBILT UNIVERSITY

CS 3250
Algorithms
Graphs: Depth-First Search
Lecture #8

DFS Tree of Discovery
Articulation Points

VANDERBILT
UNIVERSITY

Announcements

- **HW2** is due Wednesday, February 7th by 9 AM.
- There are two parts:
  1. **Brightspace Hashing Quiz**. Formative assessment. Graded but not timed.
  2. **Gradescope Questions.**
     - For the first Gradescope question, you will need to reference the graph generator quiz on Brightspace to generate your random graph for Exercise #1.
     - You should be able to do Exercise #1 after Wednesday's lecture.

Review: Graph Applications

- Remember the goal is not to reinvent the wheel.
- If there’s an excellent well-known proven algorithm for a problem, use it.
- When possible, you should try to maintain the runtime of BFS and DFS without creating a new bottleneck.
- Up to this point, we have covered two graph algorithms – BFS and DFS.

Depth-First Search: Analysis

```
//housekeeping
FOR each vertex u in V(G)
  state[u] = undiscovered
  parent[u] = nil
  set all entry/exit times to -1
END FOR
//catch all vertices
FOR each vertex u in V(G)
  IF (state[u] != discovered) THEN
       DFS(G, u)
END FOR
```

O(V)

O(V)

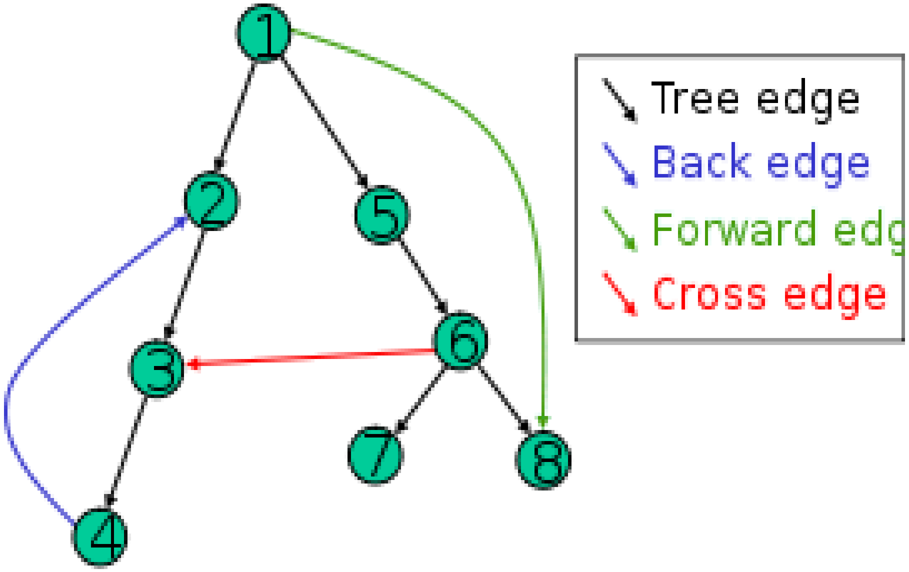
```
DFS(G, s)
  state[s] = discovered
  entry_time[s] =  time
  time = time + 1
  FOR each v adjacent to s
    IF (state[v] != discovered)
      parent[v] = s
      DFS(G, v)
END FOR
  state[s] = processed
  exit_time[s] = time
  time = time + 1
END DFS
```

O(E)

Total = O(V+E) or O(max(V, E))

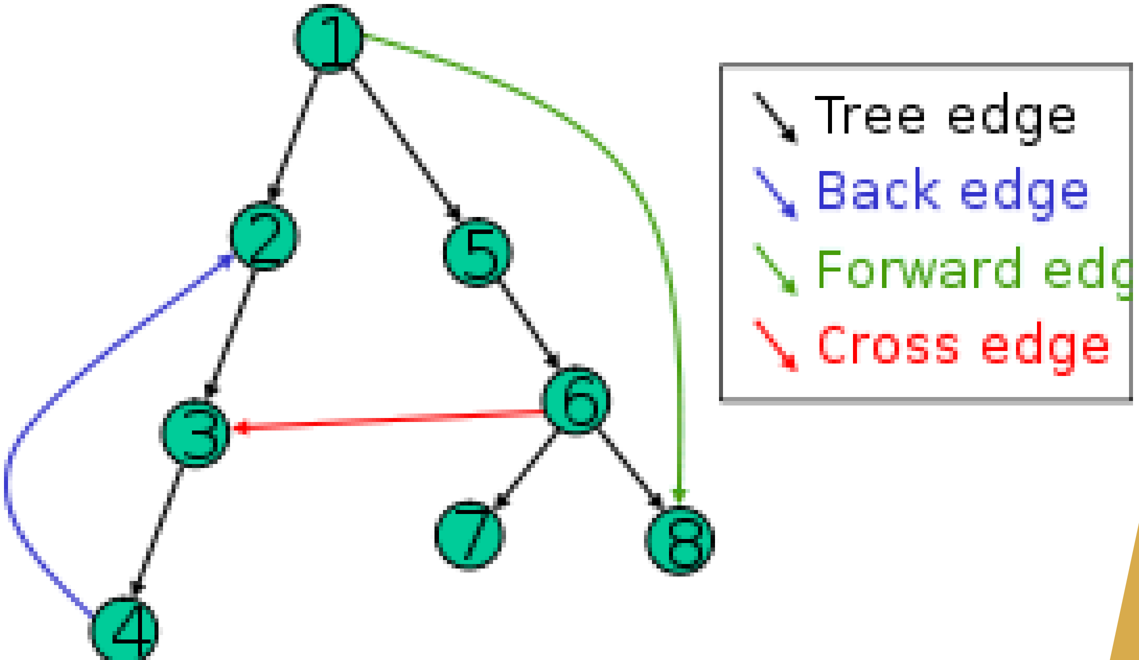
Depth-First Search: Tree of Discovery

- The tree of discovery in a DFS graph traversal has some useful properties.
- As a reminder…
  - A **tree edge** is an edge present in the tree of discovery after applying DFS to the graph.



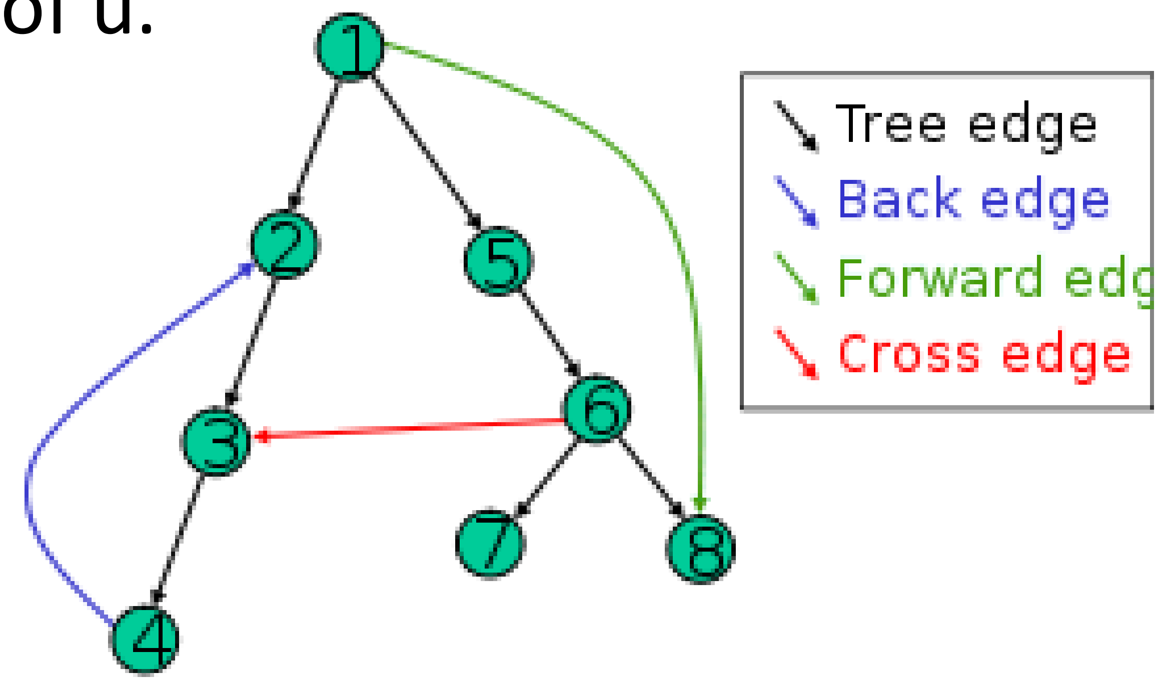
Depth-First Search: Tree of Discovery

- The tree of discovery in a graph traversal has some useful properties, but we need some terminology:
  - A **back edge** is an edge (u, v) such that v is an ancestor of u.



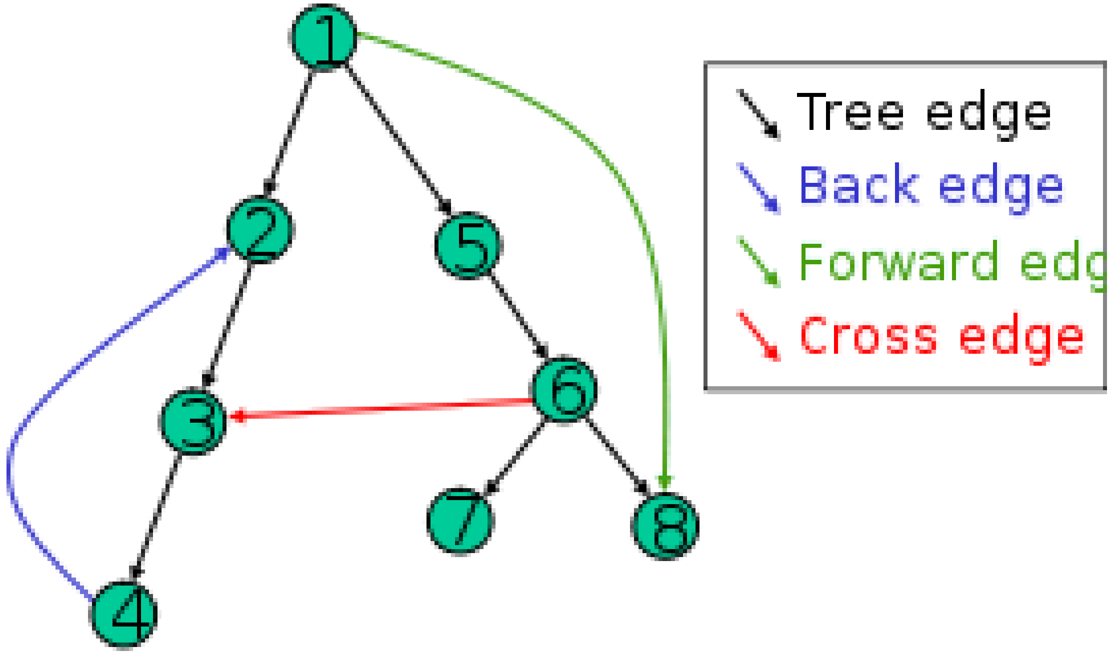
Depth-First Search: Tree of Discovery

- The tree of discovery in a graph traversal has some useful properties, but we need some terminology:
  - A **forward edge** is an edge (u, v) such that v is a descendant of u.



Depth-First Search: Tree of Discovery

- The tree of discovery in a graph traversal has some useful properties, but we need some terminology:
  - A **cross edge** is an edge (v, u) connecting two nodes that do not have any ancestor/descendant relationship.



Thinking About Depth-First Search

- **Question:**
  - Using DFS on an **undirected** graph, what kinds of edges will you find in the tree of Discovery?
  - If a tree of discovery cannot have a particular type of edge, think about why it can't.

Graphs: Depth-First Search Trees

- Aside from tree edges, what other kind of edges are possible in any **undirected** DFS tree of discovery?
  - ❑ Cross Edges only
  - ❑ Back Edges only
  - ❑ Forward Edges only
  - ❑ A combination of all the above

Graphs: Depth-First Search Trees

- Aside from tree edges, what other kind of edges are possible in any **undirected** DFS tree of discovery?
  - ❑ Cross Edges only
  - ❑ **Back Edges only**
  - ❑ Forward Edges only
  - ❑ A combination of the above

Thinking About Depth-First Search

- **Question:**
  - Using DFS on an **undirected** graph, what kinds of edges will you find in the DFS tree of Discovery?
  - If a tree of discovery cannot have a particular type of edge, explain why.

- **Answer:**
  - We can only have tree or back edges in the DFS tree of discovery for an undirected graph.
  - Why can't we have cross or forward edges?

VANDERBILT UNIVERSITY

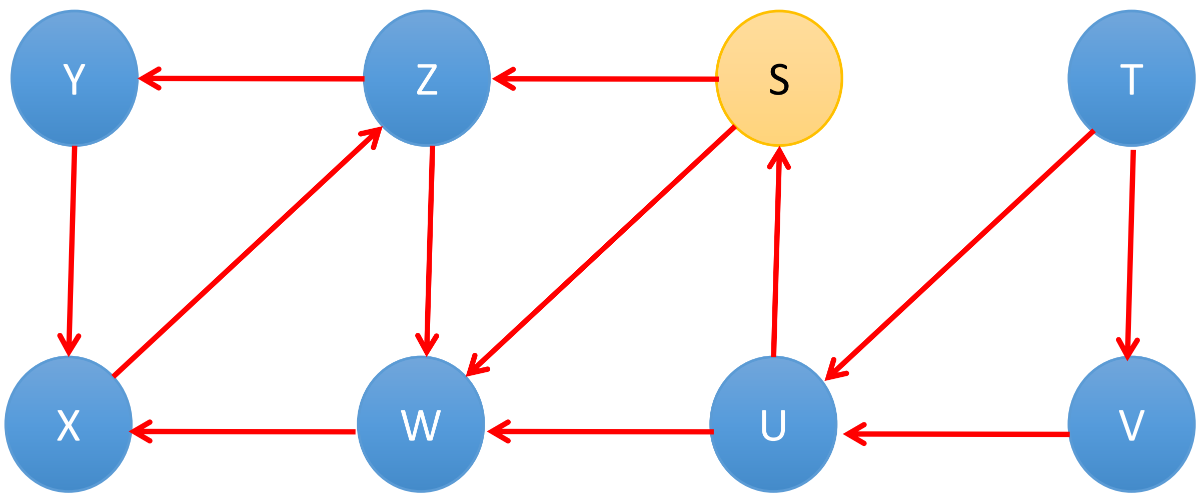
Depth-First Search: Tree of Discovery

- Some people find it helpful to use a combination of start times and vertex states to distinguish between types of edges formed in the DFS tree of discovery.
- The types of edges discovered in a **directed** graph:

Start/Entry Times	Vertex States	Edge Type v to u
start[u] > start[v]	v discovered; u not yet discovered	Tree edge
start[u] < start[v]	u, v discovered; u not yet finished	Back edge
start[v] < start[u]	u, v discovered; u finished	Forward edge
start[u] < start[v]	u, v discovered; u finished	Cross edge

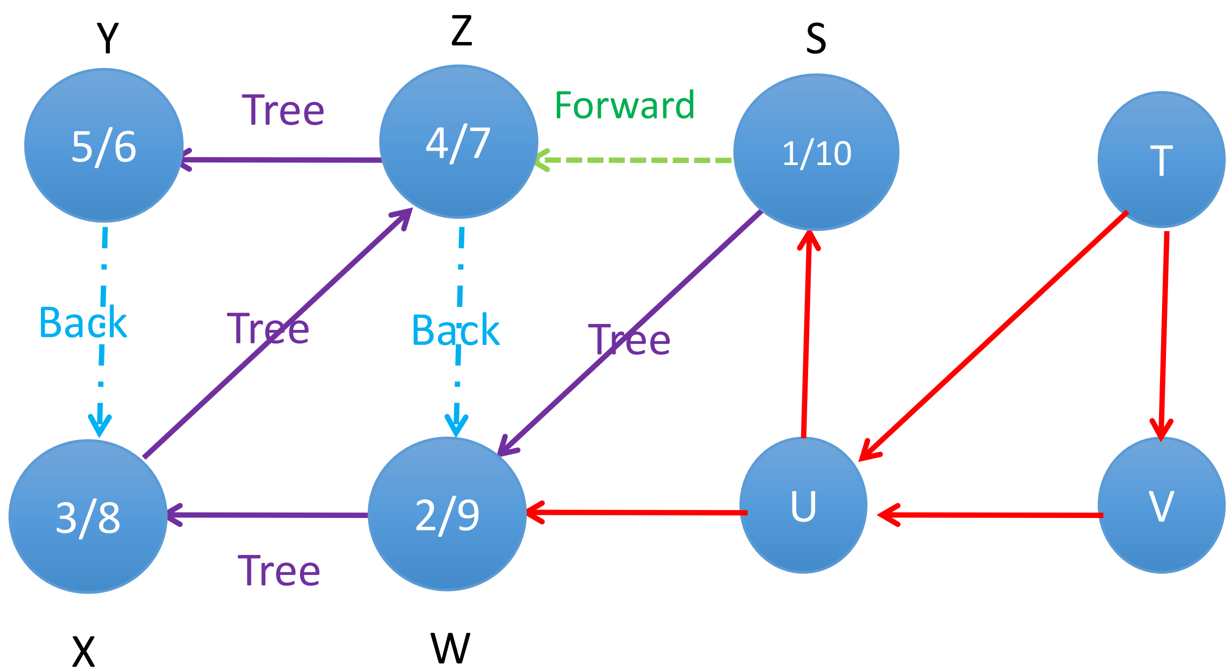
Depth-First Search: Tree of Discovery

- Let’s walk through a DFS on a **directed** graph starting at vertex **S** noting start/exit time for each vertex. When given a choice, we’ll choose the vertex that comes first lexicographically.
- Also draw the tree of discovery from the DFS process.
- This time, let’s start our DFS numbering with 1.



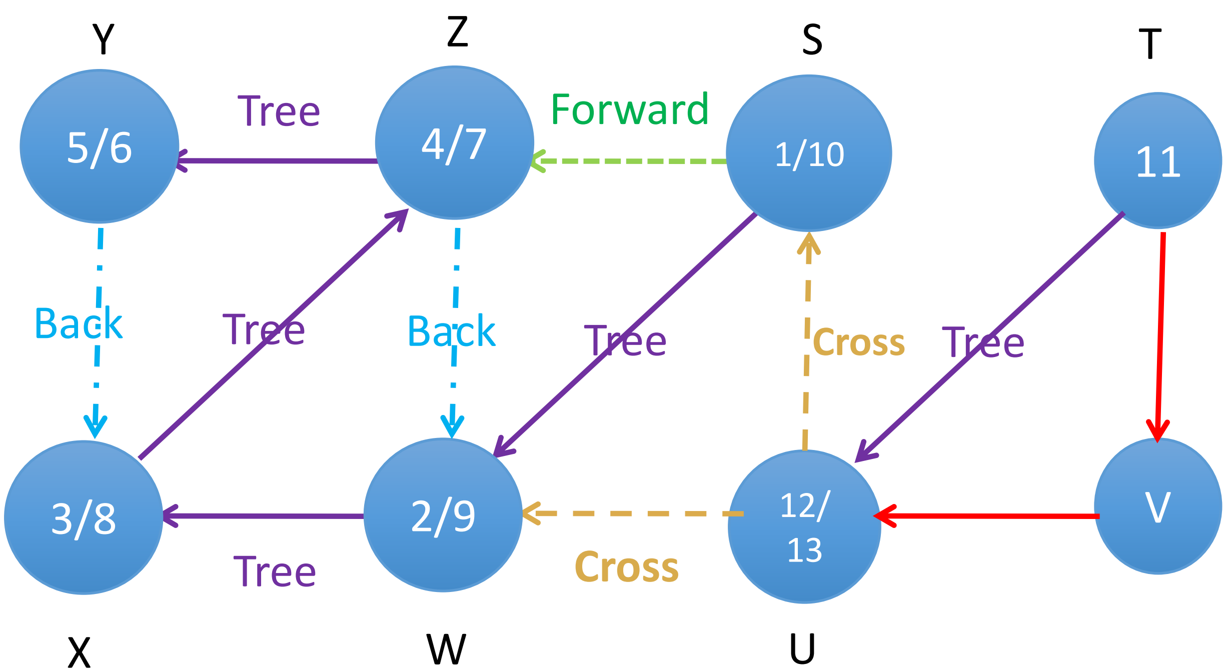
Depth-First Search: Tree of Discovery

- Solid purple = Tree edges
- Dashed blue = Back edges
- Dashed green = Forward edges
- Dashed gold = Cross edges



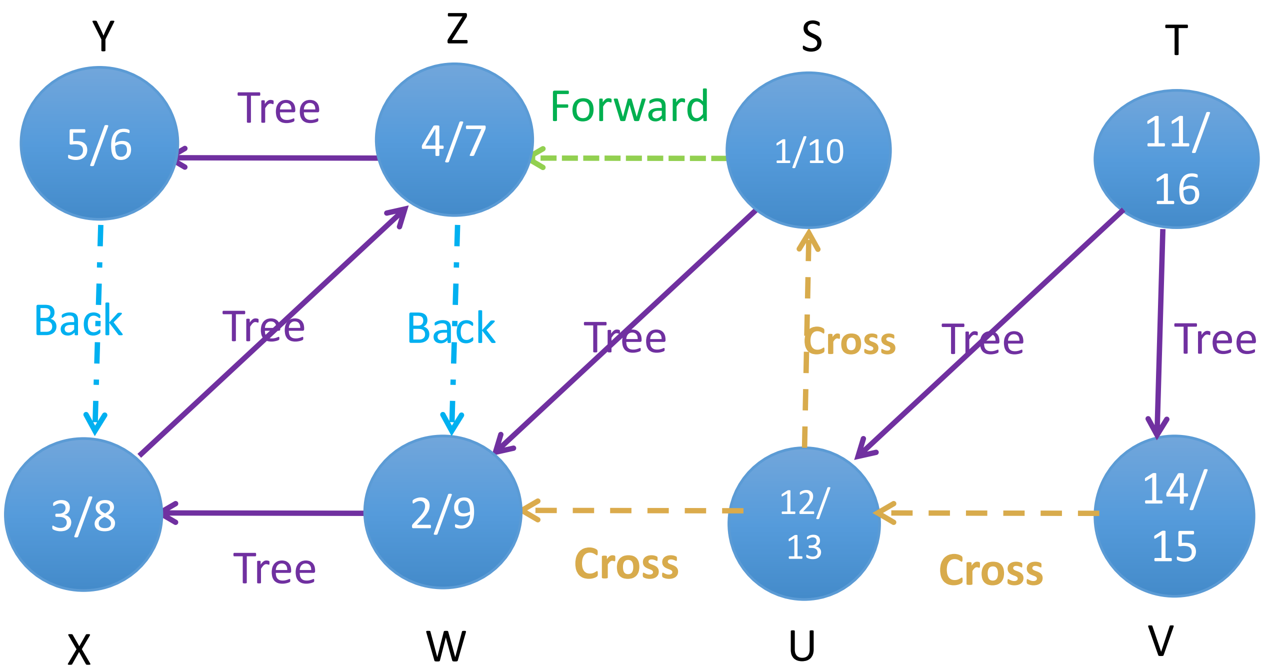
Depth-First Search: Tree of Discovery

- Solid purple = Tree edges
- Dashed blue = Back edges
- Dashed green = Forward edges
- Dashed gold = Cross edges

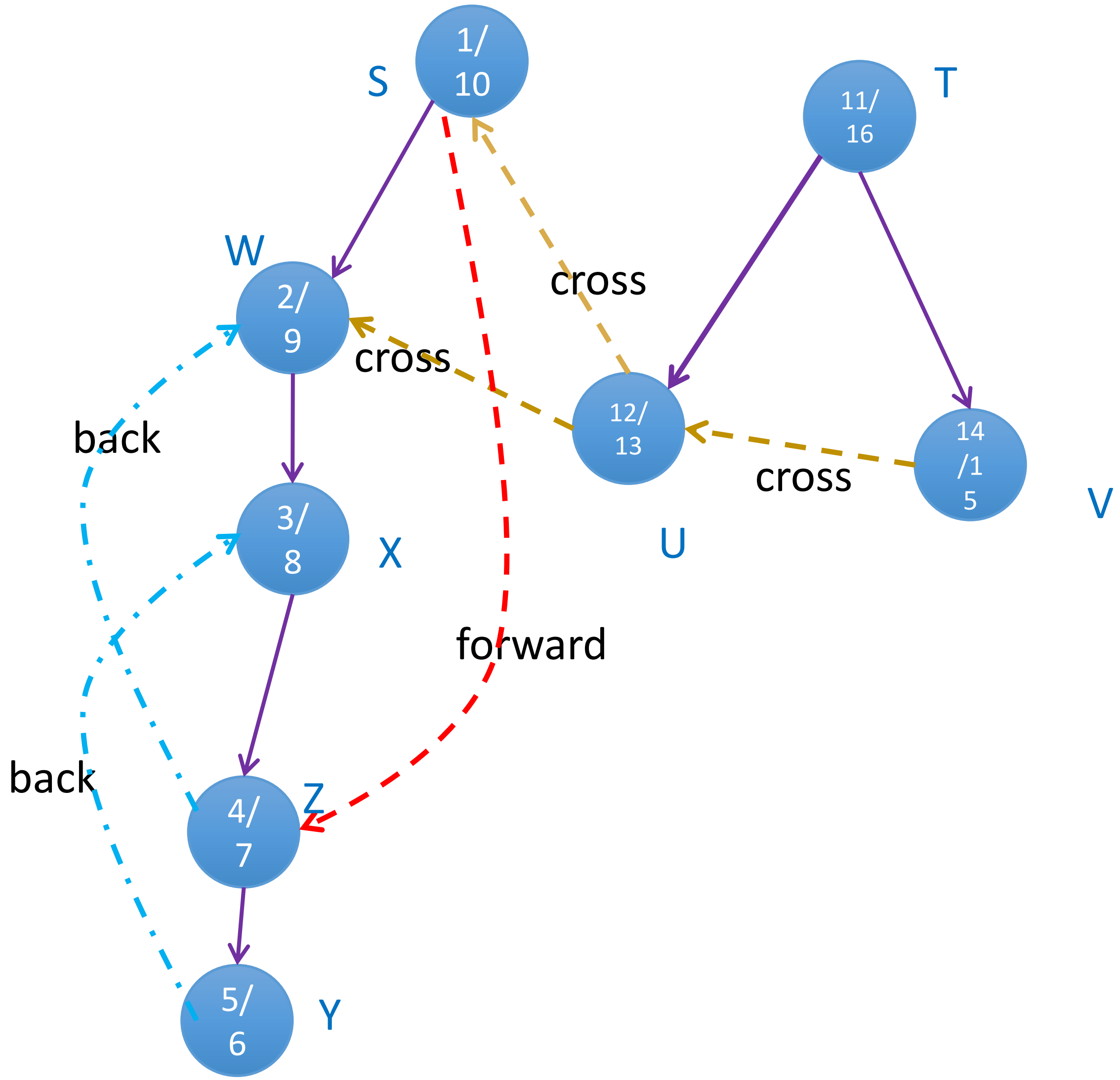


Depth-First Search: Tree of Discovery

- Solid purple = Tree edges
- Dashed blue = Back edges
- Dashed green = Forward edges
- Dashed gold = Cross edges

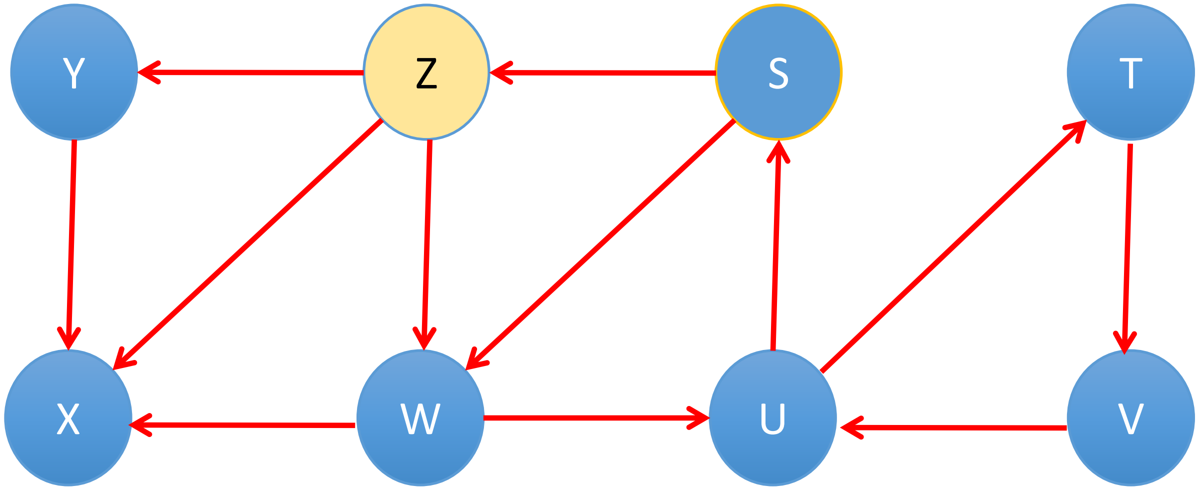


Depth-First Search: Tree of Discovery



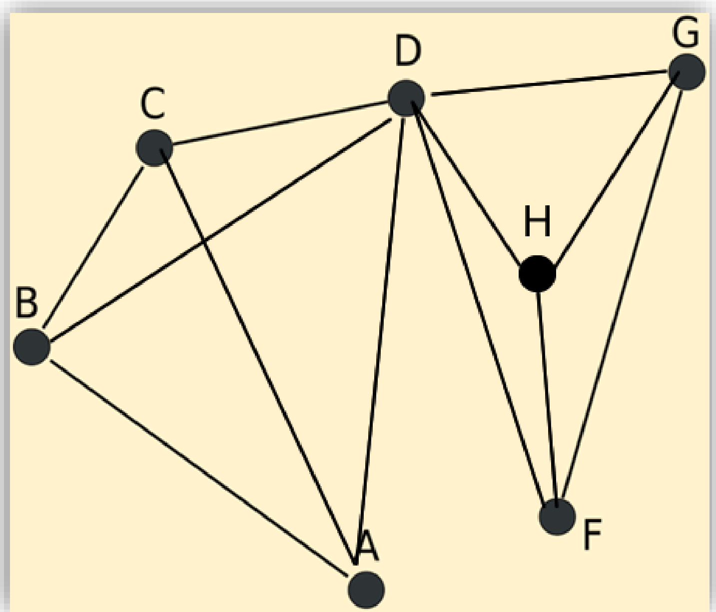
Depth-First Search Exercise

- **For Additional Practice:** Run DFS beginning at vertex z and draw the resulting tree of discovery. Start the entry time at 0. When given a choice, select neighbors in lexicographic order.



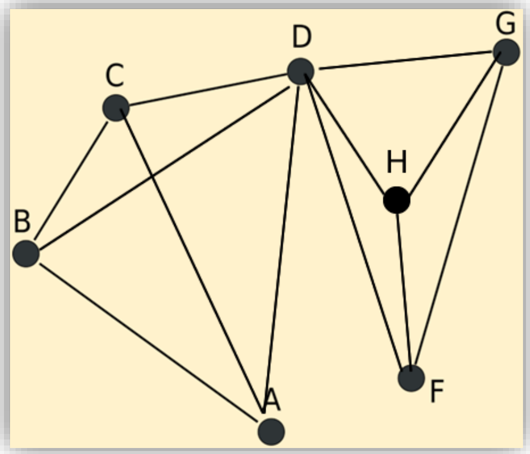
Graphs: Articulation Points

- **Question:** Suppose you are a hacker seeking to disrupt network traffic? Which server below do you target?
- **Answer:** The server at node D
- **Why?** It breaks the network into two pieces.



Graphs: Articulation Points

- An **articulation point** in an **undirected** graph is a vertex whose deletion breaks the graph into separate pieces or components.
- A graph is said to be **biconnected** if it has no articulation points.
- **Connectivity** is critical to the design of any network -- road networks, social networks, computer networks, etc.

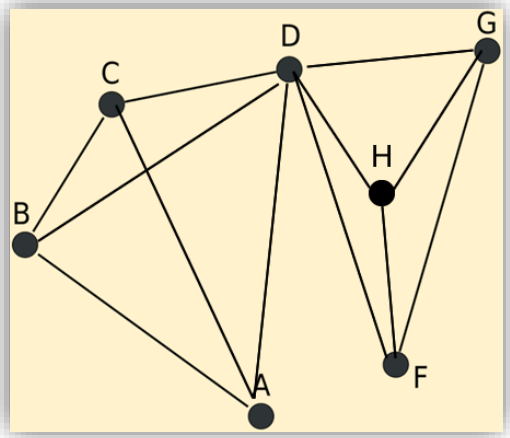


Graphs: Articulation Points

- **Connectivity in real life.**
- **Example:** There's a small village that has only a single road entering or leaving the village. That road leads to a bridge over a body of water that connects to a big city where all supplies originate.
- If the bridge is destroyed, there is no way to get the supplies from the city to the village by road.

Graphs: Articulation Points

- Many real-world problems rely on algorithms to locate these “weak links” in a connected graph.
- Locating articulation points is an **undirected** graph problem (a directed graph talks about strong articulation points and strong bridges).
- We can utilize DFS to help us locate the articulation points in an **undirected** graph.



Graphs: Articulation Points

- **First Attempt:** Given a connected, undirected graph G:

  For every vertex x in graph G

  - Remove x from the graph
  - Run DFS and see if we can still get to all other vertices. If not, x is an articulation point.
  - Add x back into the graph and repeat.

  Loop
- Will the above work? Yes!
- What's the running time?

Graphs: Articulation Points

- What's the run time of our articulation points algorithm?

  For every vertex x in graph G

    Remove x from the graph. Run DFS.

    See if we can still get to all other vertices.

    If not, x is an articulation point.

    Add x back into the graph and repeat.

  Loop

  A. O(V)

  B. O(V+E)

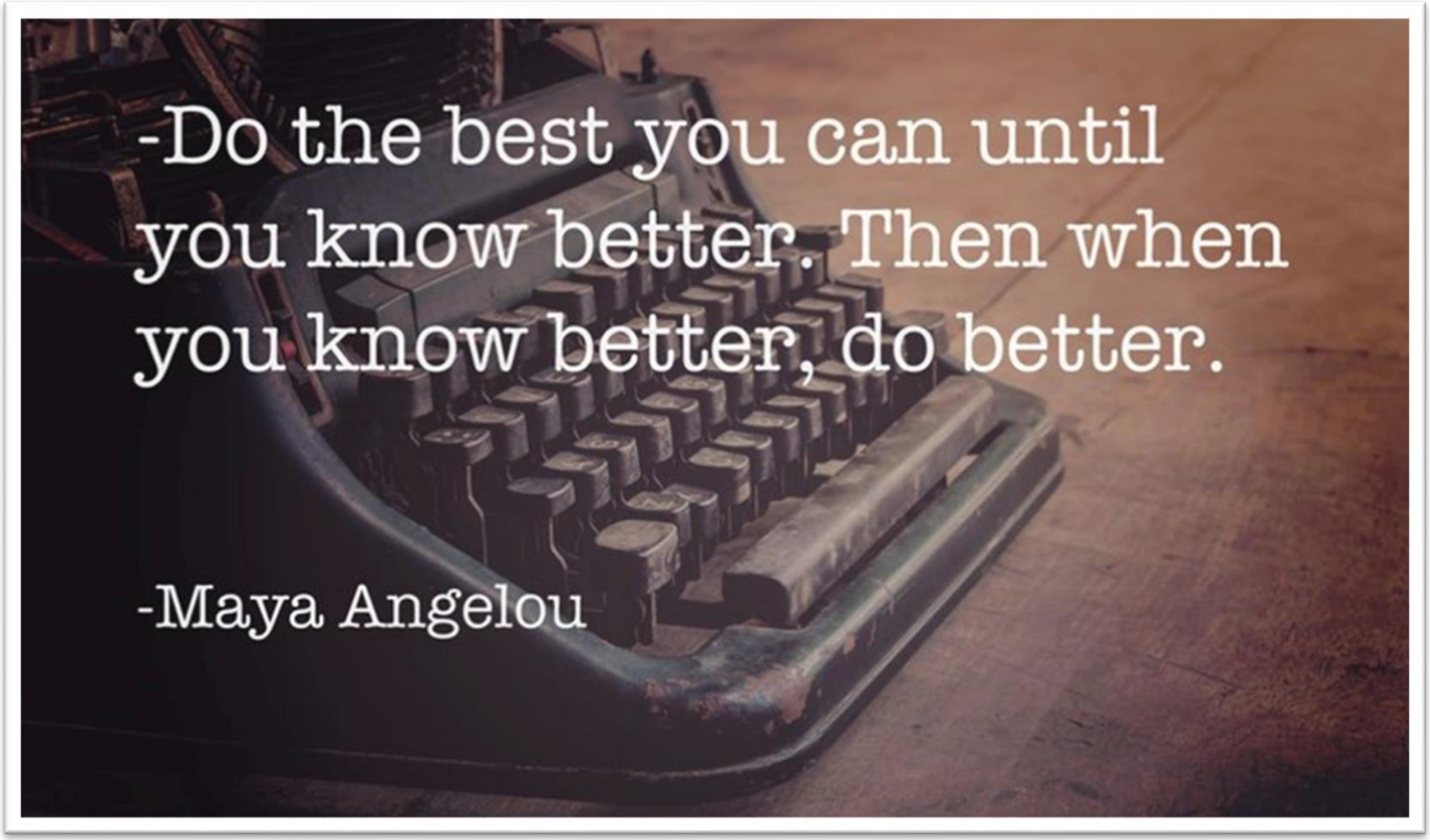
  C. O(V(V+E))

  D. O(V^2+E)

Graphs: Articulation Points

- **Answer:** Total work is as follows. We run DFS multiple times…once for each vertex. That is **O(V * (V + E)).**
- Wow, that might be V^3 for a dense graph.
- Hmmm…I wonder if we can somehow do this more efficiently? If not, we'll go with this approach.

Graphs: Articulation Points

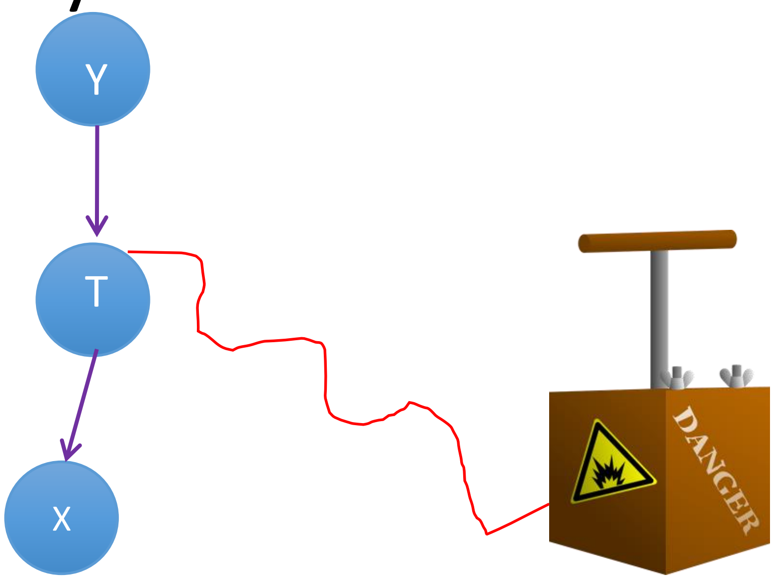


A Better Articulation Point Algorithm

- It turns out we can do better by making use of the information provided by DFS in the tree of discovery.
- In the tree of discovery for a graph G, all edges are shown as directed even when the graph is undirected.
- Think of the **back edges** in a tree as lifelines, or safety cables that link some vertex x safely back to one of its ancestors y.

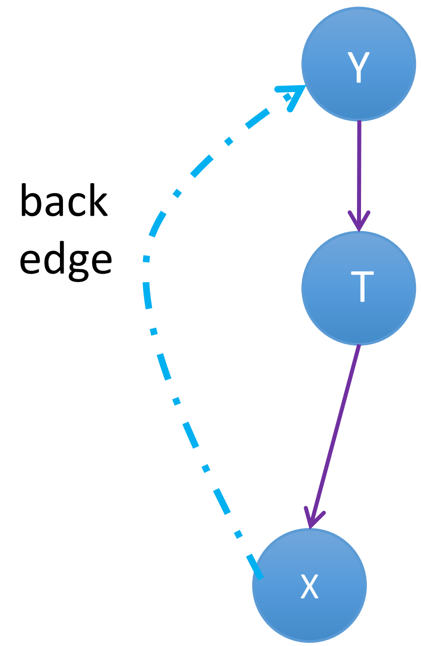
A Better Articulation Point Algorithm

- Given the DFS tree of discovery below, if we blow up (i.e., delete) vertex T, vertex x will have no way to get back to vertex Y.
- This means T is an **articulation point** whose deletion breaks the graph into two or more pieces, separating x from y.



A Better Articulation Point Algorithm

- Suppose the tree of discovery contains a back edge from x to y.
- If T is destroyed, x still has a way to safely reach vertex y.
- In this case T would **not** be an articulation point.



Graphs: Articulation Points

- **Question:** The root of the DFS tree of discovery is an articulation point when it has two or more children.
  A. Always
  B. Sometimes
  C. Never

Graphs: Articulation Points

- **Question:** The root of the DFS tree of discovery is an articulation point when it has two or more children.

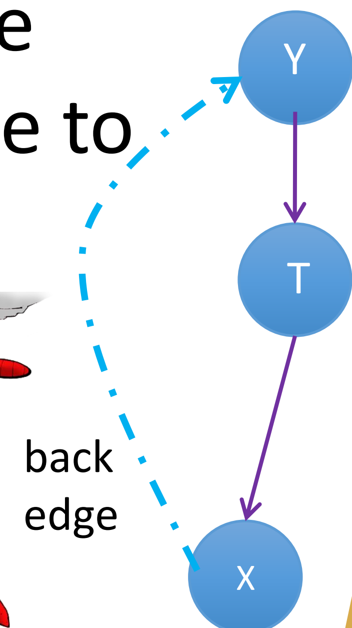
  A. Always

  B. Sometimes

  C. Never

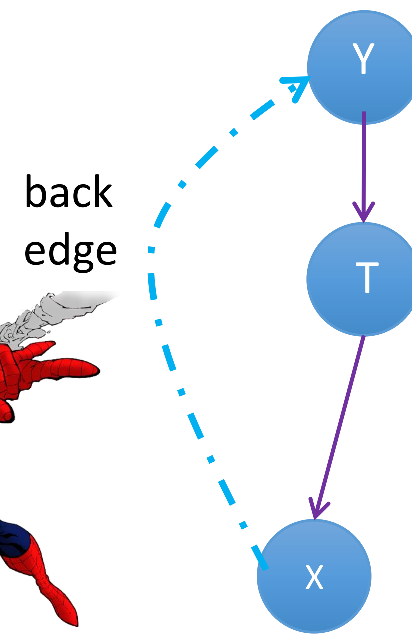
A Better Articulation Point Algorithm

- **Observations:** A vertex x in a DFS tree of discovery is an articulation point iff…
  1. x is the root of the DFS tree of Discovery and x has at least two children.
  2. x is not root of DFS tree of Discovery and has a child v where no vertex in the subtree rooted at v has a back edge to one of the ancestors of x.



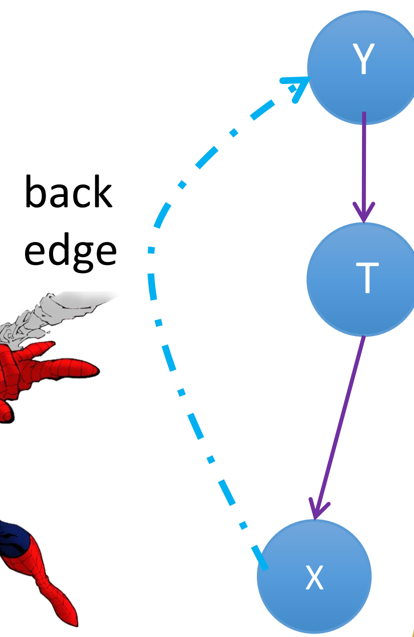
Question: Articulation Points

- **Question:** Why do we need to specify that the second observation does not apply to the root x of the DFS tree of Discovery?
- **Answer:** It is not possible in a subtree of x for one of the nodes to have a backedge to one of x's ancestors since x is the root.



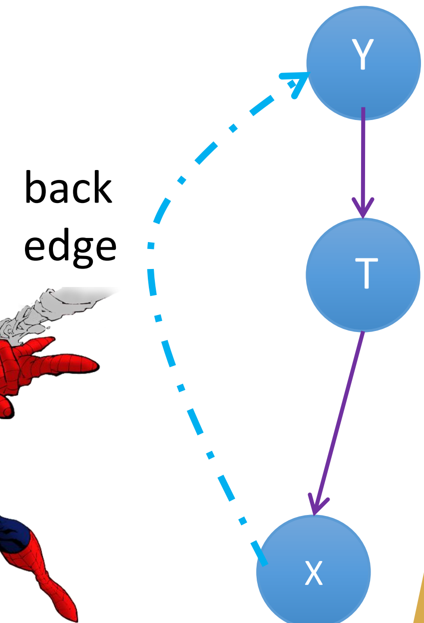
Question: Articulation Points

- **Question:** True/False. It’s not possible for a leaf in the tree of discovery to be an articulation point?
- True
- False



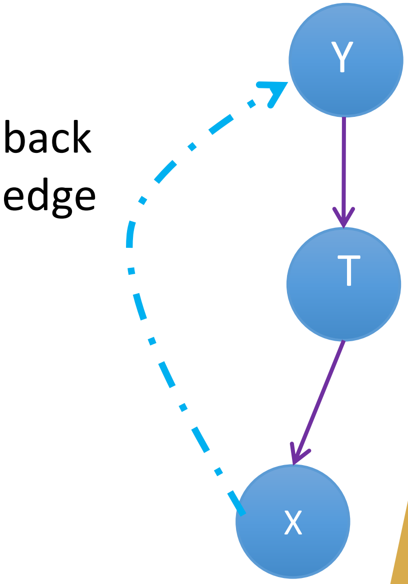
Question: Articulation Points

- **Question:** True/False. It's not possible for a leaf in the tree of discovery to be an articulation point?
- **Answer: True.** By definition, a leaf has no children so eliminating it cannot break the graph into pieces that separate the descendants of the leaf from the ancestors of the leaf.



A Better Articulation Point Algorithm

- The key to writing a smart algorithm to locate articulation points is understanding how reachability affects whether any vertex x is an articulation point.
- Provided we maintain the parent of each discovered node and the entry time from DFS, we have enough information to determine “reachability.”
- Enter **Robert** “Spiderman” **Tarjan**.



That’s All For Now…

- Coming to a Slideshow Near You Soon…
  1. DFS and Articulation Points
  2. Topological Sort
  3. DFS and Strongly Connected Components

That’s All For Now

VANDERBILT UNIVERSITY