SCHOOL OF ENGINEERING

VANDERBILT UNIVERSITY

CS 3250
Algorithms
Graphs: Depth-First Search
Lecture #7

Introduction to DFS

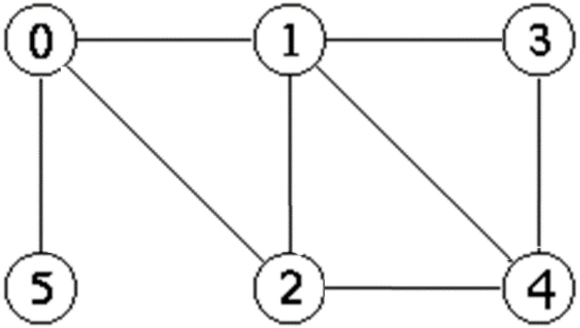
VANDERBILT UNIVERSITY

Announcements

- **HW1 Grading**. Expect a 7–10 day turnaround. Average on Asymptotic Quiz was 86.x.
- **HW2** will be released soon and due Wednesday, February 7th by 9 AM. It has two parts:
  1. **Brightspace Hashing Quiz**. Formative assessment. Graded but not timed.
  2. **Gradescope Written Questions.** Remember to keep your Gradescope answers:

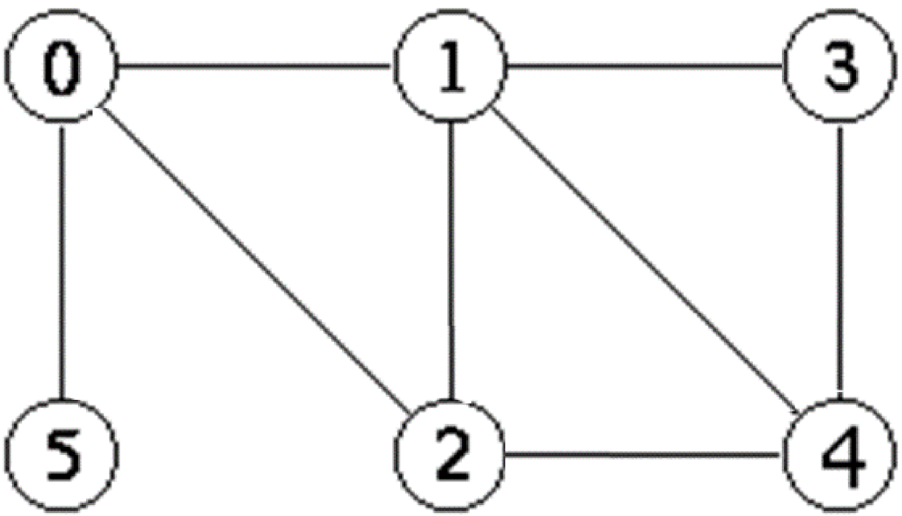
Graphs: Breadth-First Search Trees

- **Fun Fact:** In a BFS of an **undirected** graph, all edges are **tree edges** or **cross-edges**.
- **How to think about it:** Consider our graph below. In order to have a backedge from 2 to 0, it would have to be the case that 0 would have discovered 1, and 1 discovered 2, and then 2 attempts to rediscover 0. In BFS because we explore all neighbors immediately. 0 always discovers 2 via a tree edge.



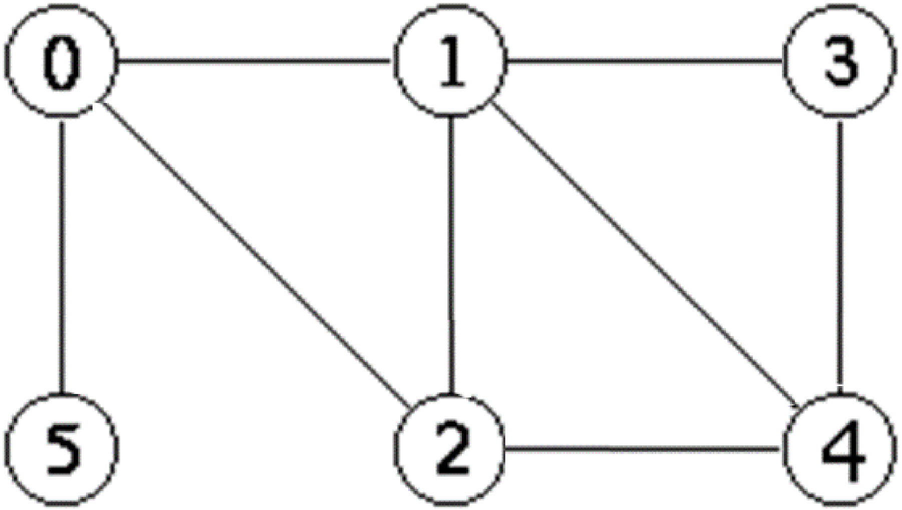
Graphs: Breadth-First Search Trees

- **Question:** If you encounter a cross edge during a BFS traversal of an undirected graph, what does that tell you?
- THINK beyond the obvious.



Graphs: Breadth-First Search Trees

- **Question:** If you encounter a cross edge during a BFS traversal of an undirected graph, what does that tell you? THINK beyond the obvious.
- **Answer:** There's a cycle in the graph!

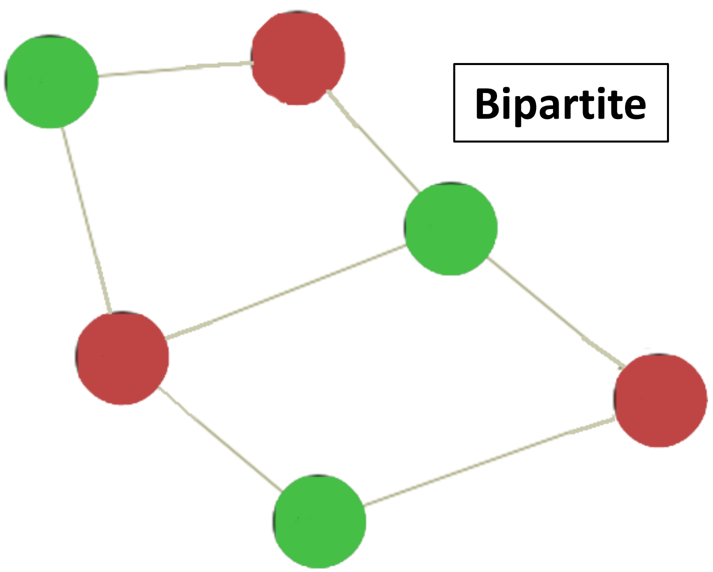


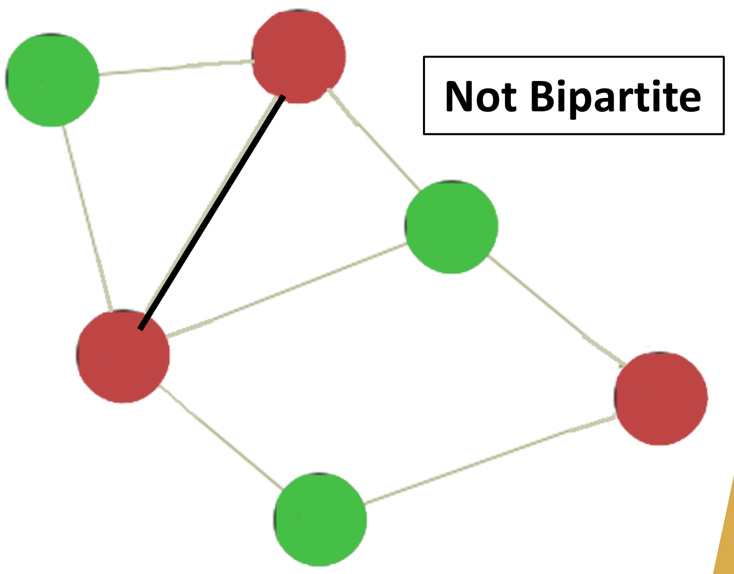
Bipartite Graphs via Breadth First Search

- Let's look at BFS in action on an application, the **two-coloring** problem.
- A bipartite graph is a graph where the vertices can be divided into two disjoint sets such that all edges connect a vertex in one set to a vertex in another set. There are no edges between vertices in the disjoint sets.

Bipartite Graphs via Breadth First Search

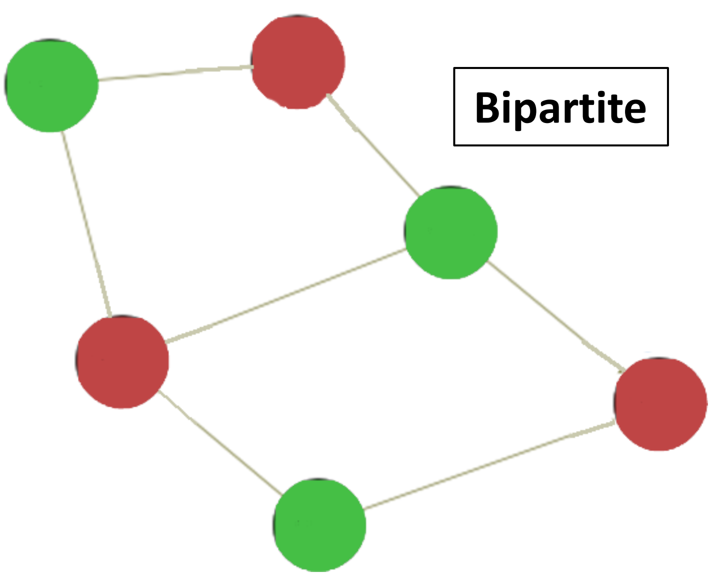
- A **bipartite** graph can be colored using only two colors such that no edge links any two vertices of the same color.
- We can use BFS to solve this problem with a simple modification.

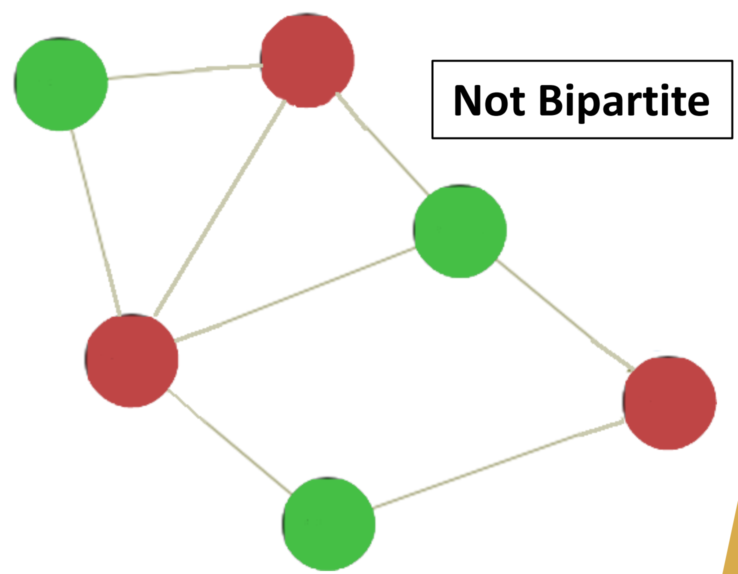




Bipartite Graphs via Breadth First Search

- **Idea:** Gradually expand the frontier of our BFS using alternate colors. While processing an edge (u, v), if v is already colored the same color as u, the graph is not bipartite.





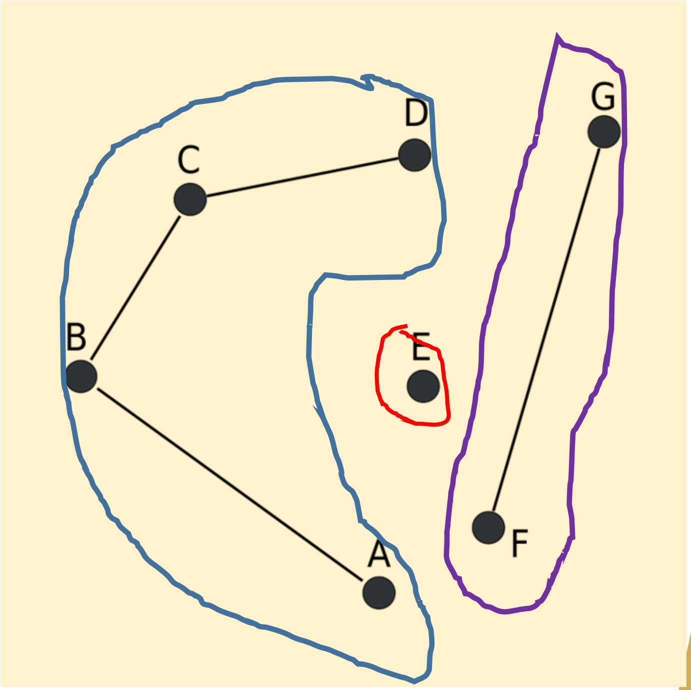
Bipartite Graphs via Breadth First Search

- We can modify BFS and add a “process edge” routine at the start of the FOR loop when we begin exploring the neighbors of the current vertex.

```
processEdge(int x, int y)
   IF (color[x] == color[y]) THEN
        bipartite = FALSE;
        PRINT Graph is not bipartite
   ELSE
       color[y] = getOppositeColor(color[x])
END METHOD
```

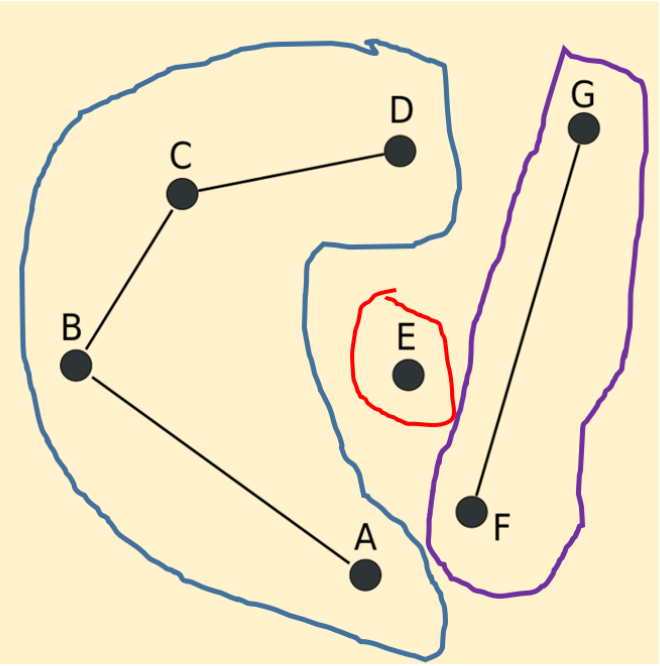
Connected Components via BFS

- A graph, G is said to be a **connected** graph if every vertex is reachable from every other vertex.
- The **connected components** of an **undirected** graph G can be defined as the equivalence classes on the relation uRv where uRv iff there exists a path from u to v.



Connected Components via BFS

- **Translation:** The connected components of a graph are the separate “pieces” that make up the graph.
- **Why do we care?** Suppose you are a network provider, and you need to run periodic checks on your network to make sure everything is still connected.



Connected Components via BFS

- **Question:** How can we use a BFS to determine the number of connected components in an **undirected** graph G.
- **Answer:** We already added a controlling FOR/WHILE loop that repeatedly calls BFS for any undiscovered vertex. We can use this to our advantage.
  - Each BFS call begins a new component.
  - All vertices discovered during each BFS call are in the same component.

Breadth-First Search: One Tool in Your Toolbox

- **Breadth-First Search** is merely one way to traverse a graph. It is not the only way.
- **Analogy:**
  - You can explore an array in different ways (i.e., walk through it forwards or backwards).
  - Sometimes the choice doesn't matter.
  - Sometimes, one approach is better or easier than another.

Graphs: Depth-First Search

- Another common graph traversal is known as **Depth-First Search (DFS).**
- If BFS and DFS are both explorers, BFS is the more **tentative** explorer.
  1. BFS peeks at all the options available nearby.
  2. BFS chooses an option, moves forward a step before retreating to look at another option.

Tentative (BFS)

Graphs: Depth-First Search

- If BFS and DFS are both explorers, DFS is the more **aggressive** explorer.
  1. DFS scans all the options available nearby and picks one.
  2. DFS then proceeds to go as far as possible on that option before retreating as little as possible to find another option (backtracking).

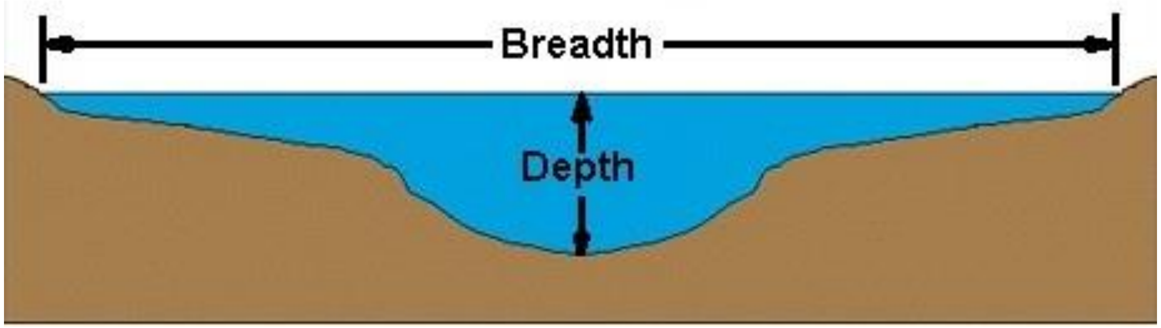
Aggressive (DFS)

Graphs: Depth-First Search

- As with BFS, no user will ever walk up to you and say, “I’ll give you a bag of cash if you can write a depth-first search for me.”
- **Depth-first search** shows up in numerous real-world applications (and job interview questions).
  - A user will describe a problem to you.
  - Your job is to recognize DFS is the right tool.

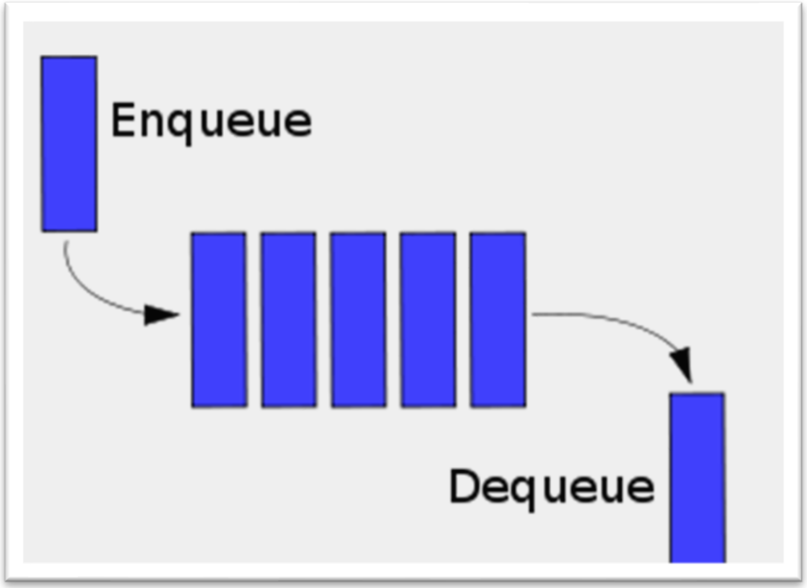
Graphs: Depth-First Search in Action

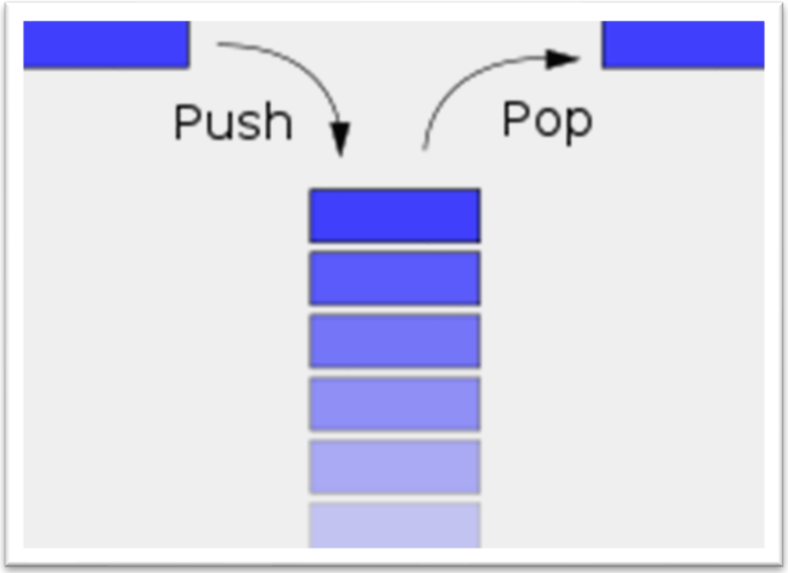
- **Depth-First Search in Action**
  - Strongly Connected Components
  - Detecting Cycles
  - Games – Sudoku/Mazes
  - Topological Sorting/Job Scheduling – List all the ways I can take the CS core requirements to complete the CS major?



Depth-First Search: How It Works

- The main difference between BFS and DFS is the choice of data structure to remember the TO-DO list.
  - In a BFS, we employ a **queue**.
  - In a DFS, we employ a **stack**.

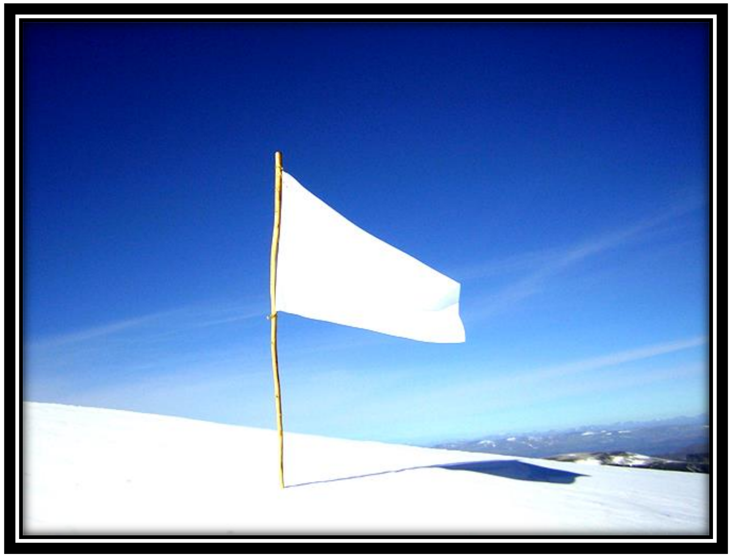




Depth-First Search: How It Works

- As with BFS, we will label each vertex in one of three states (or colors).

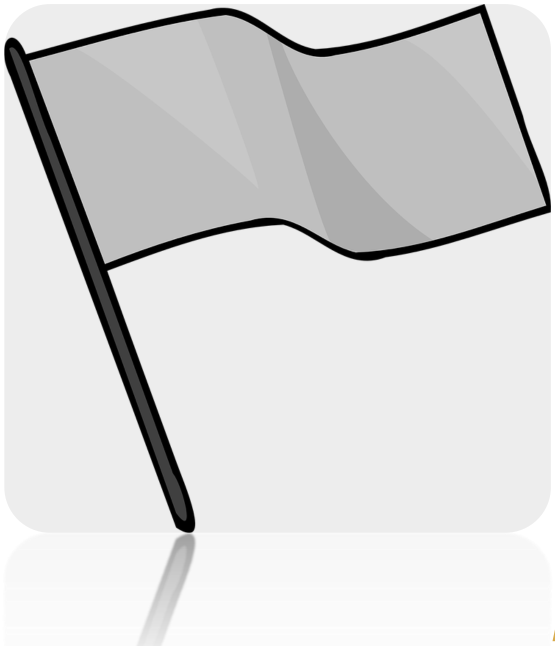
1. **Undiscovered (white)** – I haven't yet discovered this vertex. It is unchartered territory (initially all vertices are undiscovered).



Depth-First Search: How It Works

- We will label each vertex to be in one of three states (or colors).

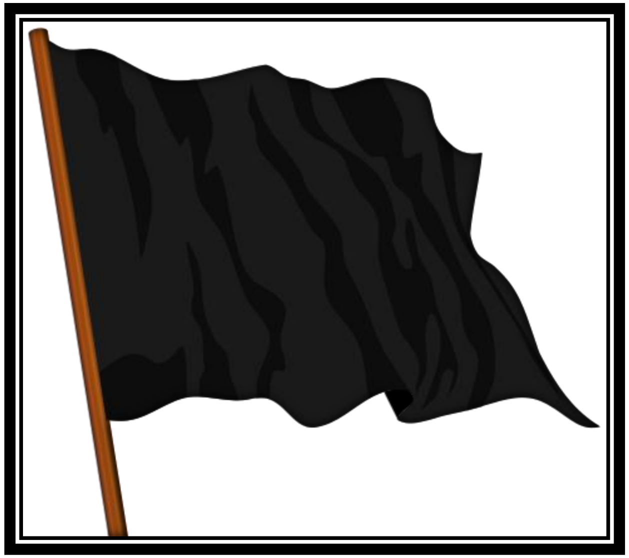
2. **Discovered but unexplored (gray)** – I have discovered this vertex and planted my gray flag here. However, I haven't thoroughly explored the area yet.



Depth-First Search: How It Works

- We will label each vertex to be in one of three states (or colors).

3. **Processed/Explored (black)** – This vertex has not only been discovered, but I have also explored every aspect of this vertex, so I have finished processing it.



Depth-First Search: Optional Tasks

- As with BFS, you may decide to perform additional work during your DFS graph traversal depending on your needs.
- Again, this is fine provided you do not sacrifice overall algorithm efficiency.

Depth-First Search: Optional Tasks

- In DFS, we often store an entry time and exit time for each vertex based on a global clock.
- This extra information will be helpful to us depending on the task at hand.

Depth-First Search: Basic Pseudocode

```
DFS(G, s) [Initially all vertices undiscovered]
  Set start vertex s to discovered
  entry_time[s] =  time
  time = time + 1
  FOR each v adjacent to s
    IF (state[v] != discovered) THEN
       parent[v] = s
       DFS(G, v)
  END FOR
  state[s] = processed
  exit_time[s] = time
  time = time + 1
END DFS
```

Depth-First Search:

- **Question:** Does the depth-first search algorithm work correctly for both an undirected graph and directed graph?
  A. Yes, definitely.
  B. Definitely not.
  C. I'm not sure.

Depth-First Search:

- **Question:** Does our depth-first search algorithm work correctly for both an undirected graph and directed graph?
  A. Yes, definitely.
  B. Definitely not.
  C. I’m not sure.

VANDERBILT UNIVERSITY

Depth-First Search:

- **Question:** Does our depth-first search algorithm always work correctly for both an undirected graph and directed graph? Why or why not?
- **Answer: No.** As with BFS, we need a wrapper loop to ensure we have visited all vertices. This is because when we check adjacent vertices, we are only looking for edges directed to other vertices. Imagine a vertex with only outgoing edges and no incoming edges.

Depth-First Search: Revised Pseudocode

DFSManager(G)

```
//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
```

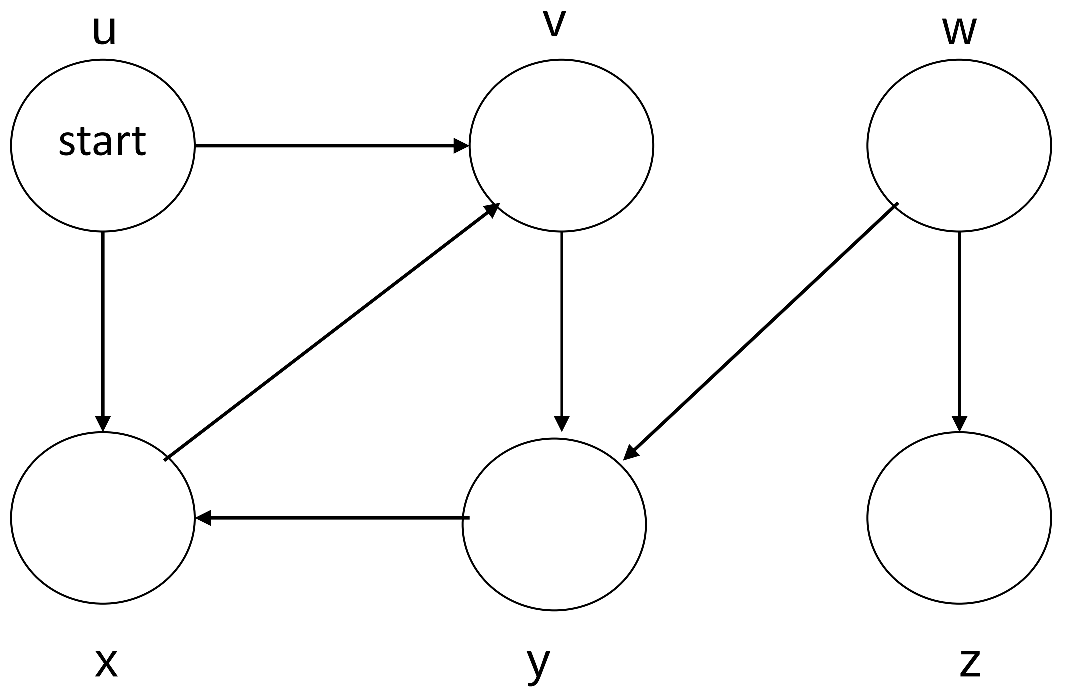
DFS(G, s) **[Initially all vertices undiscovered]**

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

VANDERBILT UNIVERSITY

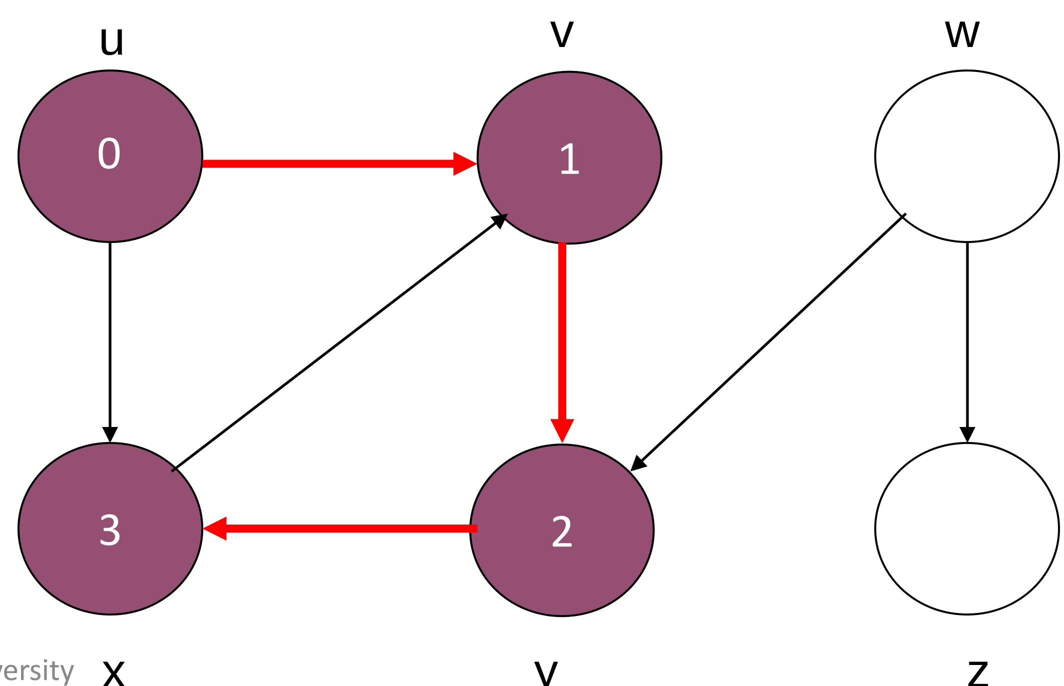
Example: Depth-First Search Walkthrough

- Let's walk through a DFS traversal on the directed graph below, indicating entry time and exit time for each vertex.
  - When given a choice, we'll choose alphabetically.
  - We'll use **purple** when vertex is first discovered.
  - We'll use **blue** when vertex is finished processing.



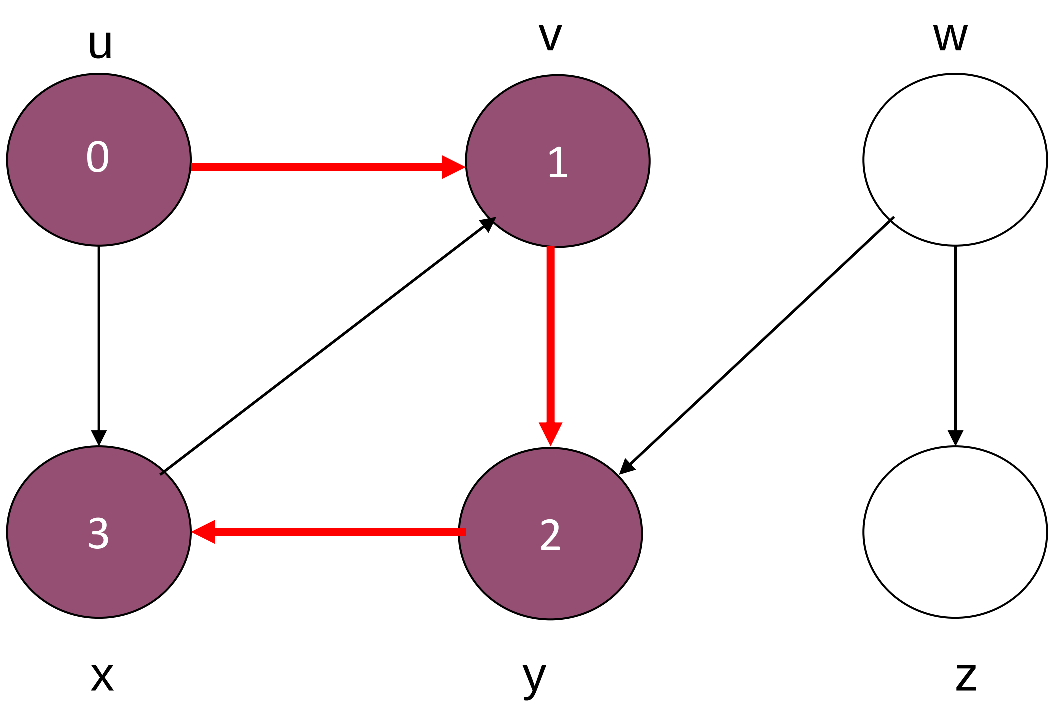
Example: Depth-First Search Walkthrough

- From vertex u at time 0, we select uv and follow it.
- v is undiscovered, so it is given a time stamp of 0+1.
- There is only edge vy to follow which leads to the undiscovered vertex y, which is time stamped 2.
- We follow edge yx to vertex x which is stamped 3.



Example: Depth-First Search Walkthrough

- **Question:** What timestamp will be given to vertex x upon exiting this DFS?



Depth-First Search:

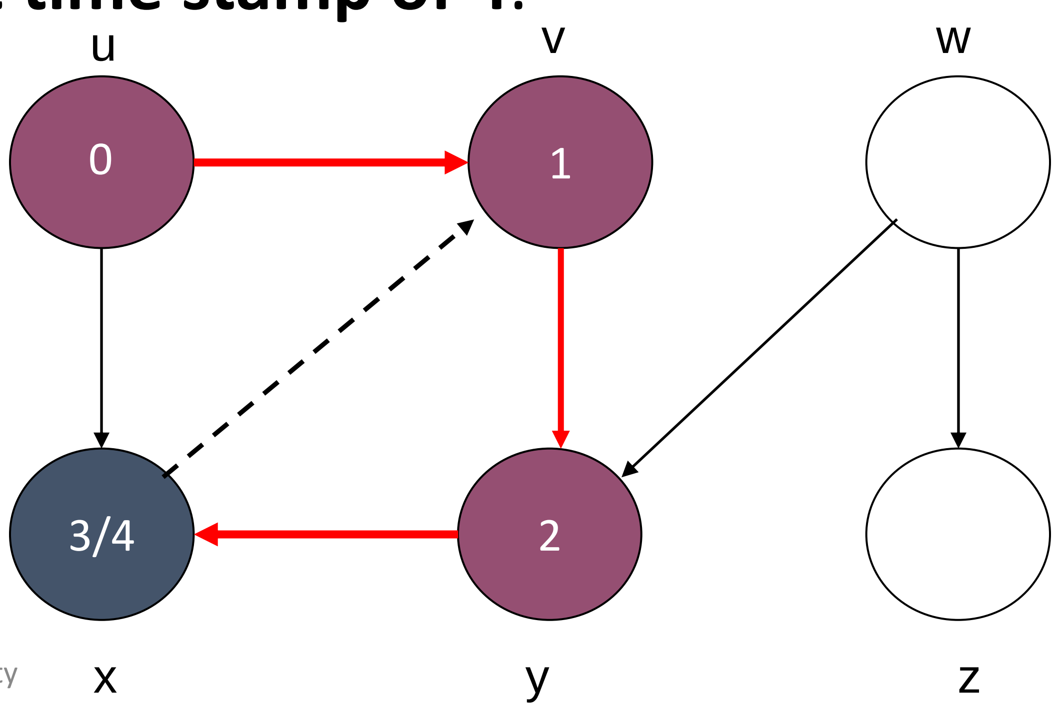
- **Question:** What timestamp will be given to vertex x upon exiting this DFS?
  A. 3
  B. 4
  C. 5
  D. None of the above

Depth-First Search:

- **Question:** What timestamp will be given to vertex x upon exiting this DFS?
  A. 3
  B. 4
  C. 5
  D. None of the above

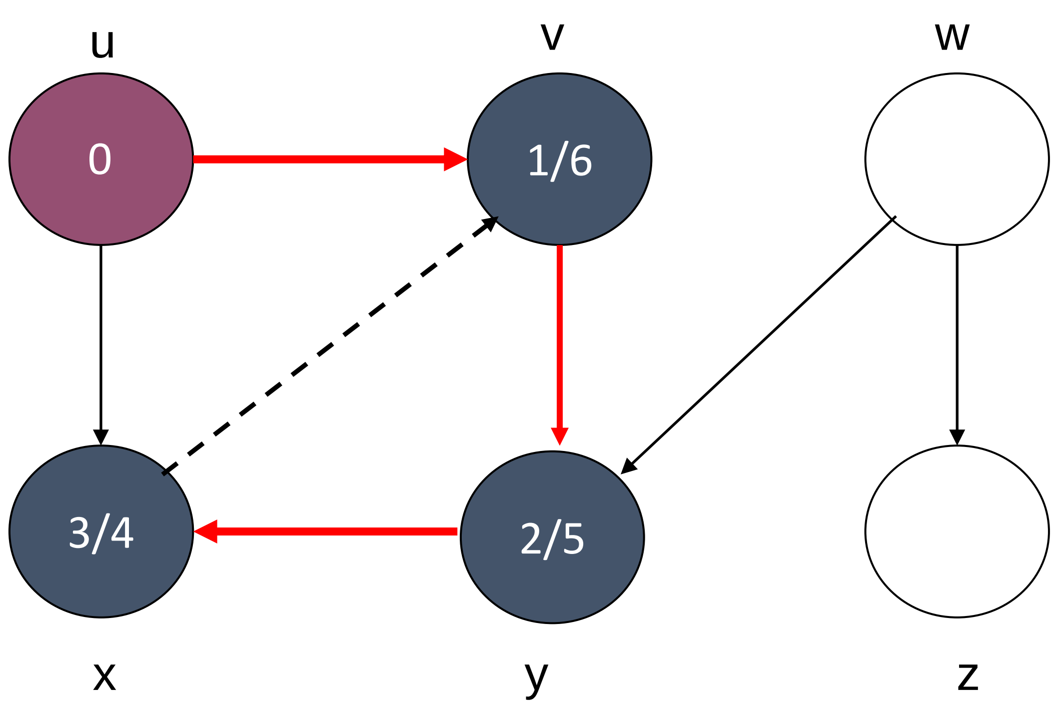
Example: Depth-First Search Walkthrough

- We follow the only edge out of vertex x, edge xv.
- This leads to already discovered vertex v, so we backtrack along xv and exclude that edge in the DFS tree of discovery.
- There are no other edges out of vertex x, so **x is finished with an exit time stamp of 4**.



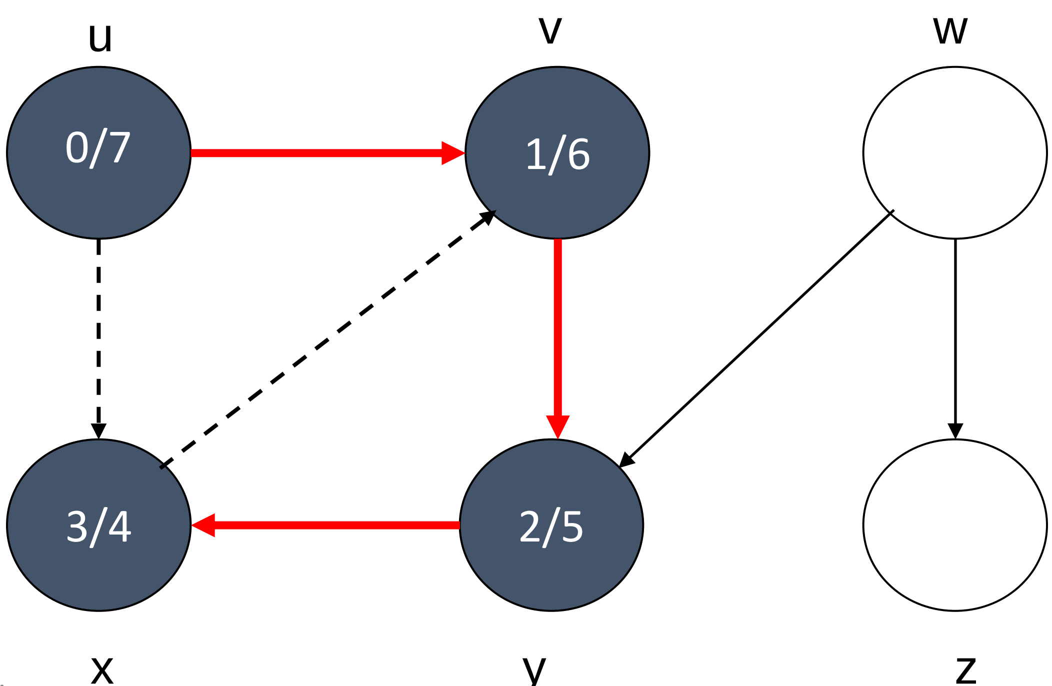
Example: Depth-First Search Walkthrough

- We backtrack to parent of vertex x, that is the vertex from which x was first discovered, which is y.
- Since there are no more edges to explore from y, it is finished given an exit time stamp of 5.
- Do the same for v which has an exit time stamp of 6.



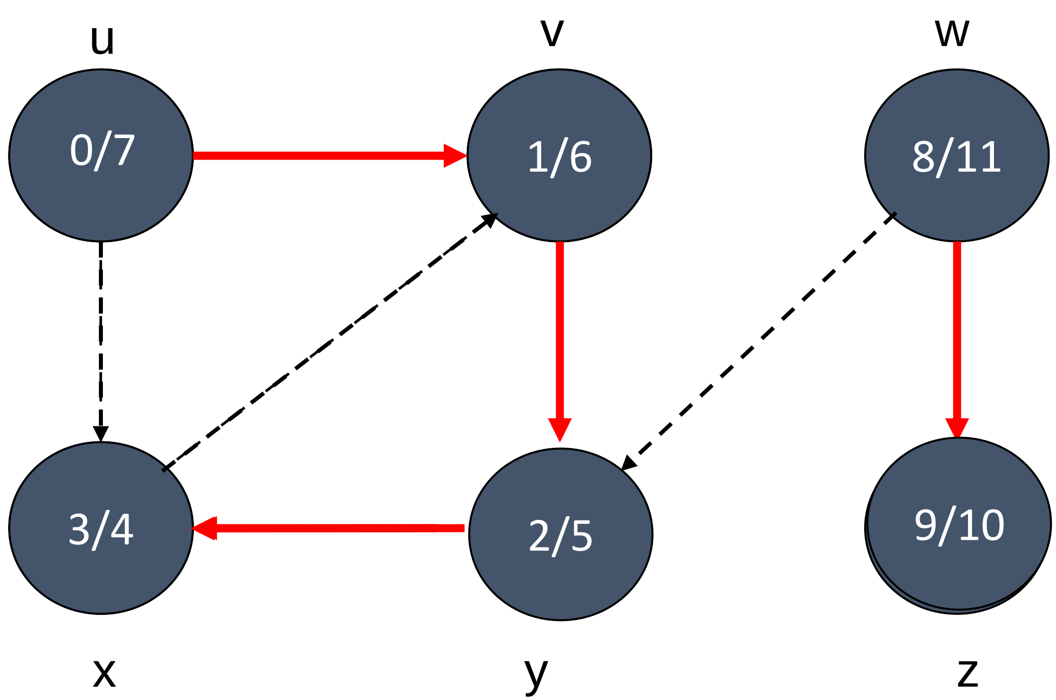
Example: Depth-First Search Walkthrough

- Back at vertex u, we now traverse the edge ux. This leads to an already discovered (and finished) vertex x so the edge is not included in the DFS tree.
- There are no more edges out of u, so it is marked as finished with an exit time stamp of 7.



Example: Depth-First Search Walkthrough

- We select another undiscovered vertex and begin the DFS again.
- When completed, we have the following results.



Depth-First Search: Analysis

DFSManager(G)

```
//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) THEN
      parent[v] = s
      DFS(G, v)
END FOR
state[s] = processed
exit_time[s] = time
time = time + 1
```

O(E)

Depth-First Search: Analysis

- As with BFS, if we are careless with our analysis, we might think it's $O(V * E)$ or being a bit more careful, $O(V^2)$.
- However, $O(V^2)$. only happens in a very dense graph where $|E|$ approaches V^2.
- It's more accurate to say that the algorithm is $\Theta(V+E)$. Many authors simply say $O(V+E)$ or $O(\max(V, E))$. In other words, same as BFS.

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 Summary

- **When to use DFS:** DFS is a useful traversal method when the solution is far from the source vertex (e.g., how can I complete the CS degree?).
- **Where DFS Shines:** Detecting cycles, topological sort, job scheduling, mazes.
- **Running Time:** O(V+E) which could approach O(V^2) in a dense graph. Same as BFS.

That’s All For Now…

- Coming to a Slideshow Near You Soon…
    1. DFS Tree of Discovery
    2. Topological Sort

That’s All For Now