SCHOOL OF ENGINEERING

VANDERBILT UNIVERSITY

CS 3250
Algorithms
Graphs: Breadth-First Search
Lecture 6

Tree of Discovery & Bipartite Graphs

VANDERBILT UNIVERSITY

Announcements

- **HW1 Grading**. Expect a 7–10 day turnaround.
- **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:
     - Clear
     - Correct
     - Concise

Review: Breadth-First Search

- **When to use BFS:** BFS is a useful traversal method when you know the solution isn't far from the source vertex (e.g., friend suggestions on Facebook that are one-degree away from you).
- **Where BFS Shines:** Social-Networks, Shortest Path (unweighted), Spanning Tree (undirected).

Breadth-First Search: Pseudocode

```
BFS(G, s) [Initially all vertices undiscovered]
  Set start vertex s discovered and set parent to nil,
  Add s to the TO-DO list (enqueue)
  WHILE (there are vertices on the TO-DO list)
      v = take a vertex off the To-Do list (dequeue)
      Optional: Do early vertex processing
      FOR (all of v's neighbors w)
            IF (state[w] == UNDISCOVERED/white) THEN
                  Change status of vertex w to DISCOVERED/gray)
                  Add w to the TO-DO list (enqueue)
                  Optional: Record the level where w was discovered
                  Optional: Record the parent/predecessor of w, which is v
            Move to the next neighbor w of v
      LOOP
  Change status of vertex v to PROCESSED/black
  Optional: Do late vertex processing
  LOOP
END BFS
```


VANDERBILT UNIVERSITY

Breadth-First Search: Manager Loop

BFSManager(G)

```
//housekeeping
FOR each vertex u in V(G)
  state[u] = undiscovered
  parent[u] = nil
 time = 0
END FOR
//catch all vertices
FOR each vertex u in V(G)
   IF (state[u] != discovered) THEN
        BFS(G, u)
END FOR
```

The manager loop for BFS ensures that if the graph is not connected, we still discover all vertices

Breadth-First Search: Detecting Cycles

- **Question:** Would our logic for detecting a cycle also work for determining cycles in a **directed** graph?
  A. Yes. We are experienced programmers.
  B. No. We are experienced programmers.
  C. I thought so, but now I'm not sure. I just can't figure out why.

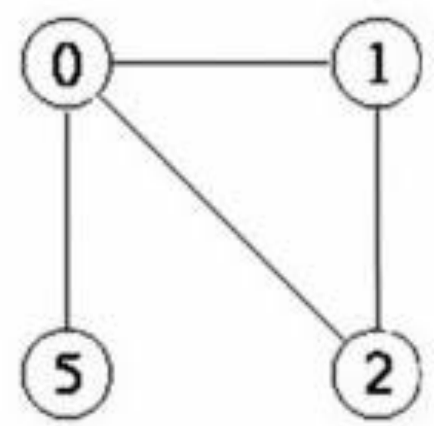
Breadth-First Search: Detecting Cycles

- **Question:** Would this logic also work for determining cycles in a **directed** graph?
- **Answer:** No, it would not work to determine cycles in a directed graph. For a path to be a cycle you must be able to get back to the first vertex. The fact that a vertex has already been discovered by a someone else, does not mean you can get from that vertex back to whoever discovered it. The edge might only be directed toward the vertex and not back.

VANDERBILT UNIVERSITY

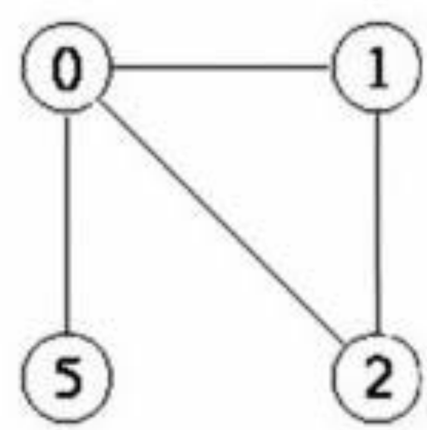
Review: Breadth-First Search

- **Question:** A BFS beginning at vertex 0 where ties are broken by the smallest numbered vertex will realize there's a cycle in the graph below when...
  A. vertex 2 tries to reach vertex 0.
  B. vertex 1 tries to reach vertex 2.
  C. vertex 1 tries to reach vertex 0.
  D. vertex 2 tries to reach vertex 1.
  E. None of the above



Review: Breadth-First Search

- **Question:** A BFS beginning at vertex 0 where ties are broken by the smallest numbered vertex will realize there's a cycle in the graph below when...
  A. vertex 2 tries to reach vertex 0.
  B. vertex 1 tries to reach vertex 2.
  C. vertex 1 tries to reach vertex 0.
  D. vertex 2 tries to reach vertex 1.
  E. None of the above



Breadth-First Search: Algorithm Analysis

- "optional" tasks have been removed since work can vary.

```
BFS(G, s) [Initially all vertices undiscovered]      ← O(V)
  Set start vertex s discovered and set parent to nil ← O(1)
  Add s to the TO-DO list (enqueue)                   ← O(1)
```

Pre-Work Housekeeping O(V)

```
  WHILE (there are vertices on the TO-DO list)        ← O(V)
     v = take a vertex off the To-Do list (dequeue)
     FOR (all of v's neighbors w)                     ← O(E)
           IF (state[w] == undiscovered) THEN
                 Change state[w] to discovered
                 Add w to the TO-DO list (enqueue)
           Move to the next neighbor w of v
     LOOP
  Change state[v] to explored/processed
  LOOP
  END BFS
```

O(V*E) ???

Breadth-First Search: Algorithm Analysis

- When doing analysis, it's important to THINK about how the algorithm works rather than just looking at the code.
- If you just look at the code, you might conclude $O(V*E)$ or V^2*E or even V^3 is a good bound. Yes, those are correct bounds, but they are sloppy.
- If we think about how the algorithm works, we can arrive at a tighter bound.

Breadth-First Search: Algorithm Analysis

- A "smarter" analysis yields the following:

```
BFS(G, s) [Initially all vertices undiscovered]      ← O(V)
  Set start vertex s discovered and set parent to nil ← O(1)
  Add s to the TO-DO list (enqueue)                   ← O(1)
```

Pre-Work Housekeeping

O(V)

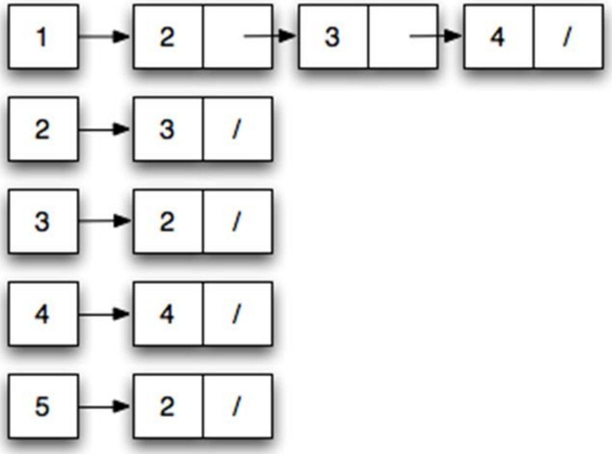
```
  WHILE (there are vertices on the TO-DO list)
     v = take a vertex off the To-Do list (dequeue)
     FOR (all of v's neighbors w) ← (*every edge looked at 1x or 2x)
           IF (state[w] == undiscovered) THEN
                 Change state[w] to discovered
                 Add w to the TO-DO list (enqueue)
           Move to the next neighbor w of v
     LOOP
  Change state[v] to explored/processed
  LOOP
END BFS
```

O(V+E)

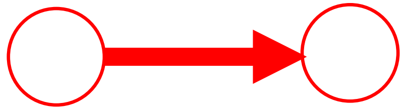
O(V + E)

Breadth-First Search: Data Structure Choice

- Let’s think visually…
- We will definitely go down the list of all vertices. No way to avoid that. That’s |V| work.
- We will definitely need to look at all the edges. No way to avoid that. That’s |E| work.
- In an undirected graph, we’ll see 2x the number of edges. That’s 2|E| work.



Breadth-First Search: Analysis Summary

- **Directed graph** 
  - Set up + Traverse each vertex/edge once
  - O(V) + E
- **Undirected graph** 
  - Set up + Traverse each edge twice
  - O(V) + 2E

Breadth-First Search: Analysis Summary

- **Overall Running Time**
  - $\Theta(V + E)$. Most authors simply write $O(V+E)$. Some say $O(\max(V, E))$.
  - Note that this could approach $O(V^2)$ if the graph is dense (has lots of edges).
  - If the graph is not dense, work is less than $O(V^2)$

VANDERBILT UNIVERSITY

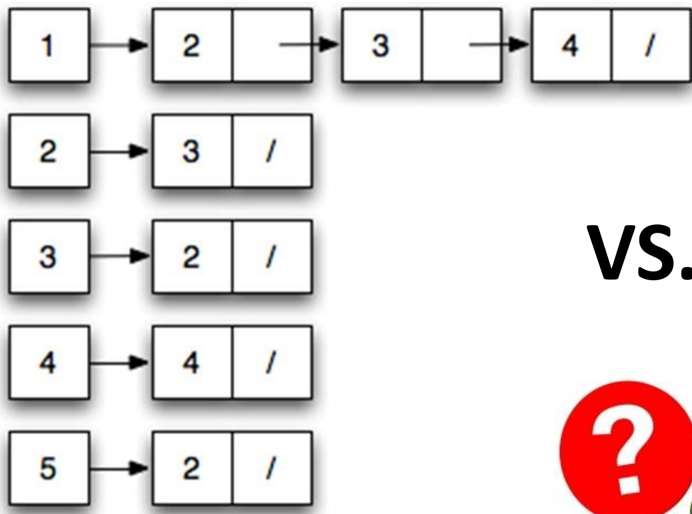
Breadth-First Search: Data Structure

- **When Choosing Your Data Structure:**

1. **Be efficient**. Always choose the most efficient data structure for the algorithm.
2. **Be obvious.** If your data structure is not obvious, you should explicitly state your data structure.
3. **Breadth-First Search.** Our BFS analysis was based on an adjacency list data structure.

Breadth-First Search: Data Structure Choice

- **Question:** What if we used an **adjacency matrix** instead of an adjacency list as our data structure? How does that effect our BFS analysis?



VS.

	1	2	3	4	5
1	0	1	1	1	0
2	0	0	1	0	0
3	0	1	0	0	0
4	0	0	0	0	0
5	0	1	0	0	0

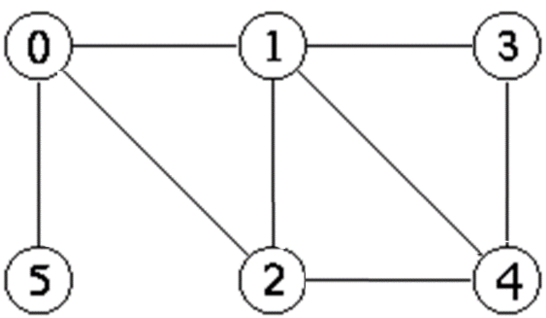
Breadth-First Search: Data Structure Choice

- **Question:** What if we used an **adjacency matrix** instead of an adjacency list as our data structure? How does that effect our BFS analysis?
- **Answer:** For each vertex visited, every other vertex needs to be checked for a "1" (in the same row). So, each vertex visited results in V checks. That means V checks for each vertex V which is O(V^2).

	1	2	3	4	5
1	0	1	1	1	0
2	0	0	1	0	0
3	0	1	0	0	0
4	0	0	0	0	0
5	0	1	0	0	0

Breadth-First Search: Tree of Discovery

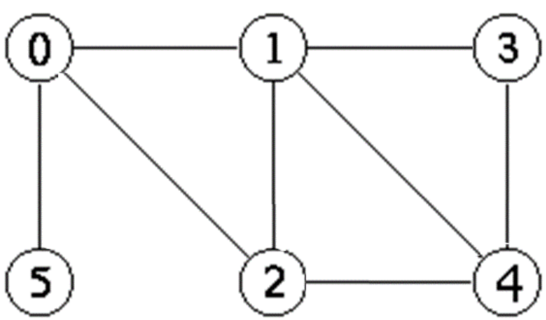
- While traversing a graph, a **spanning tree** is naturally formed known as the “**Tree of Discovery**” or **“Traversal Tree.”**
- I’ll use a black arrow to indicate who discovered a vertex first and a red dashed line from u to v to indicate u stumbled onto an already discovered, but unprocessed vertex, v (i.e., via a new edge).



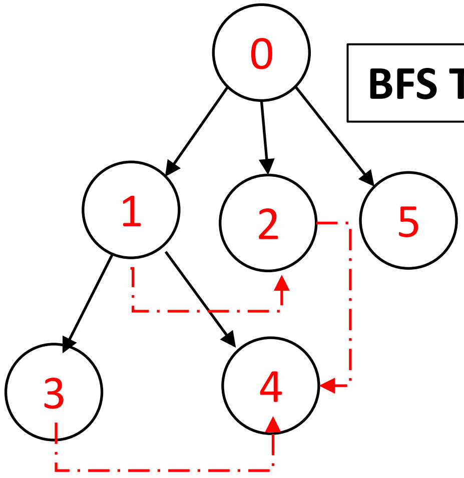
Initial Graph

Breadth-First Search: Tree of Discovery

- Here is the BFS "**Tree of Discovery**" or **"Traversal Tree"** for our undirected graph.
- A black arrow indicates who discovered a vertex first and a red dashed line from u to v indicates u stumbled onto an already discovered, but unprocessed vertex, v (i.e., via a new edge).
- Notice the Tree of Discovery uses arrows.



Initial Graph

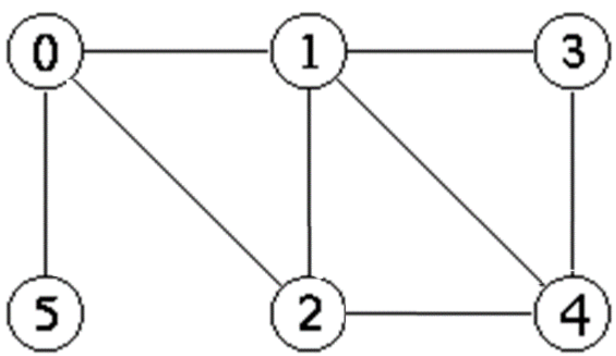


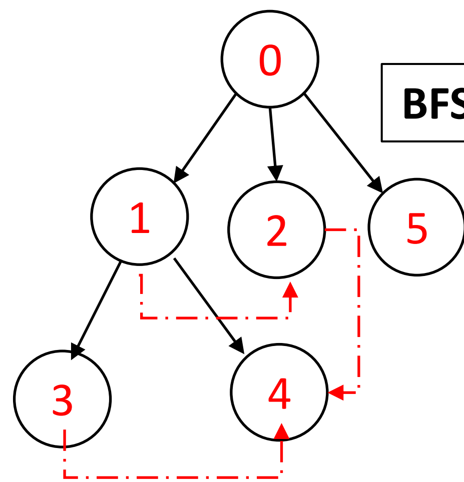
BFS Tree of Discovery

Breadth-First Search: Tree of Discovery

- **Fun Fact:** The BFS tree of discovery for an **unweighted** graph contains the shortest distance from a vertex to any other vertex from the graph.
- **Question:** How to find this shortest path?
- **Answer:** Keep track of the parent

Initial Graph

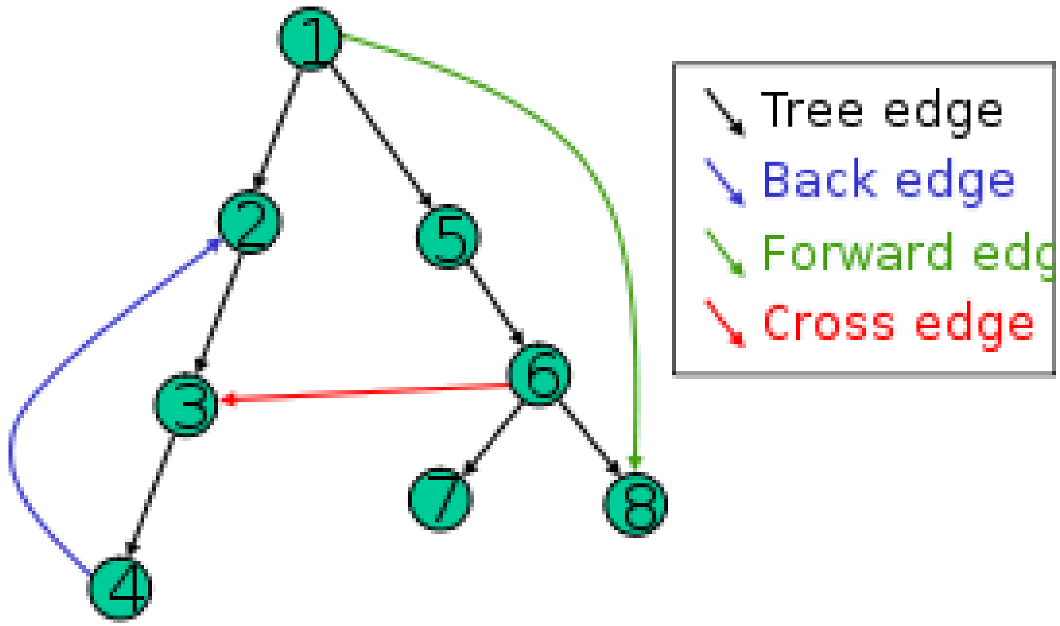




BFS Traversal Tree

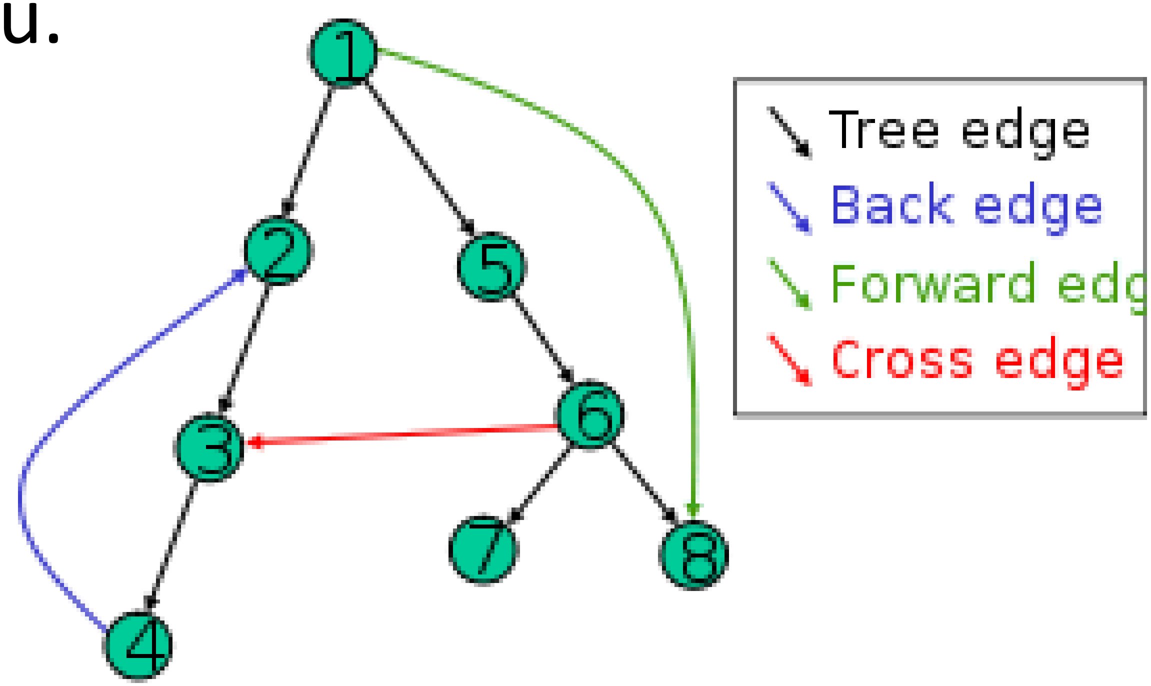
Breadth-First Search: Tree of Discovery

- The tree of discovery in a graph traversal has some useful properties. We need some additional terminology to talk about them:
  - A **tree edge** is an edge present in the tree of discovery after applying BFS to the graph.



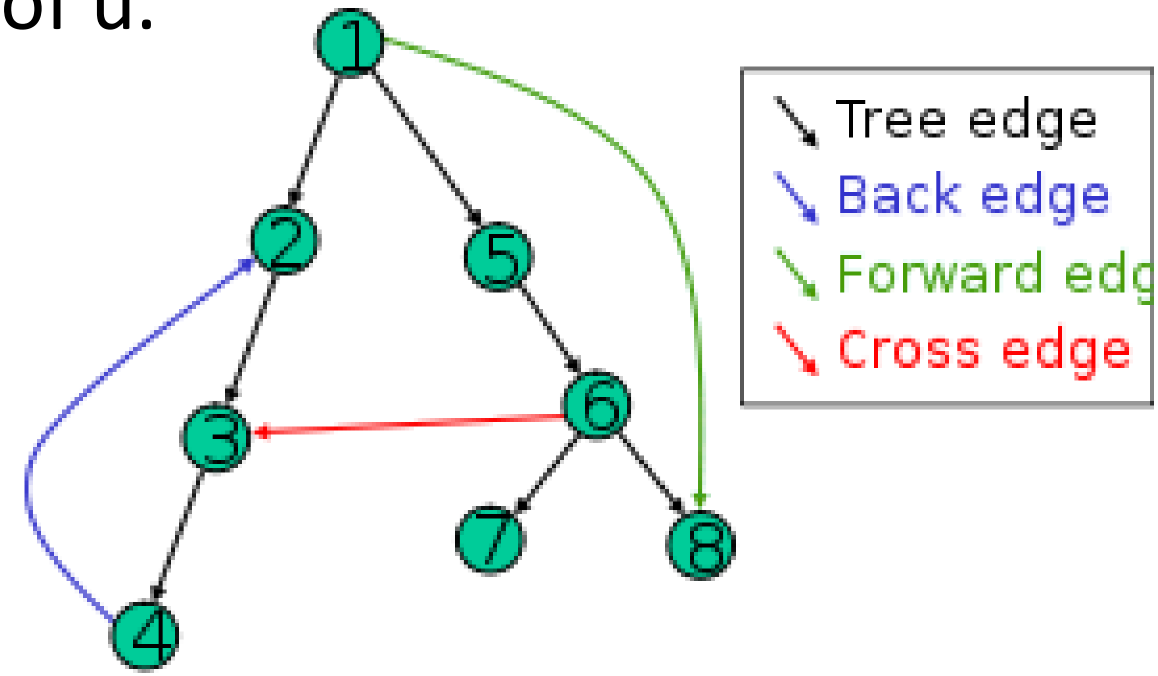
Breadth-First Search: Tree of Discovery

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



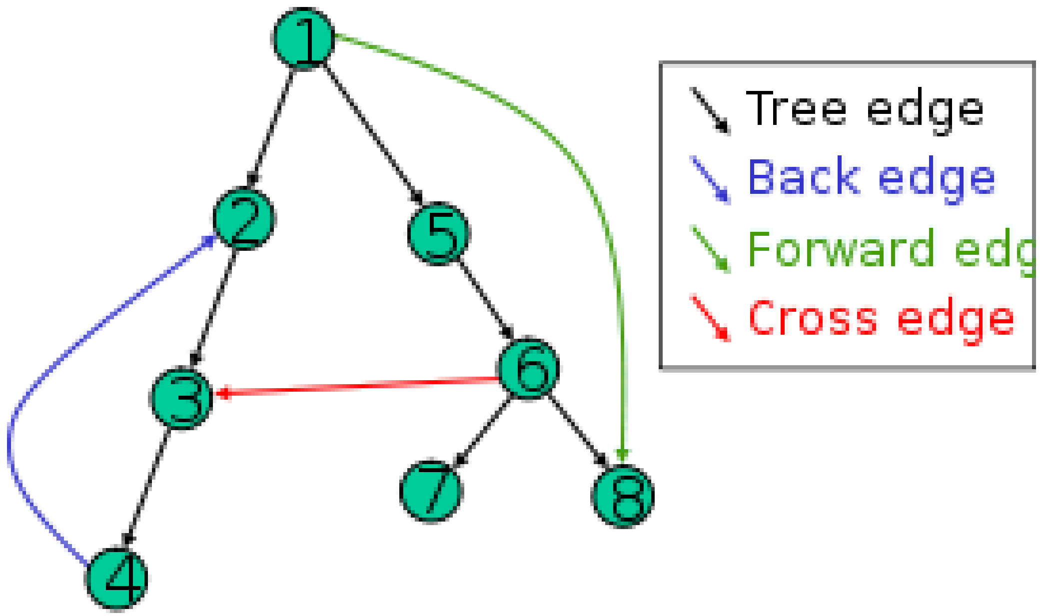
Breadth-First Search: Tree of Discovery

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



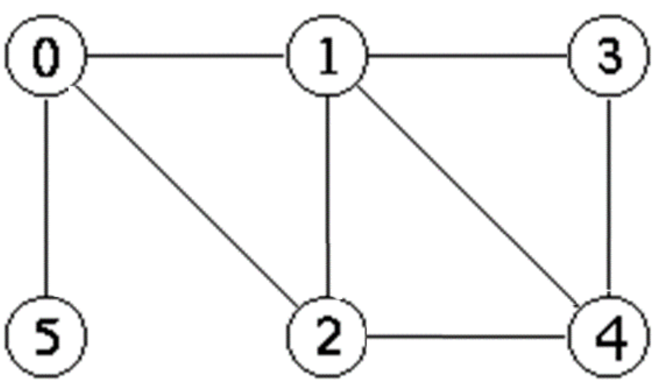
Breadth-First Search: Tree of Discovery

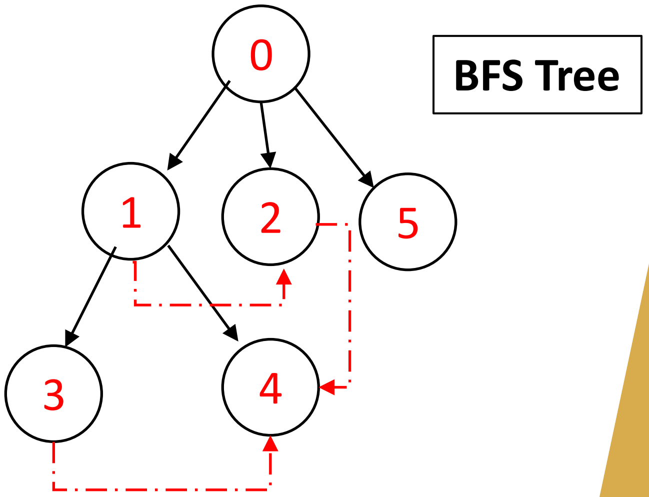
- The tree of discovery in a graph traversal has some useful properties. We need some terminology to talk about them:
  - A **cross edge** is an edge (u, v) connecting two nodes that do not have any ancestor/descendant relationship.



Graphs: Breadth-First Search Trees

- What kinds of edges exist in our BFS tree below?
  - **Tree edges** – (0, 1), (0, 2), (1, 4), etc. are tree edges formed during the natural traversal of BFS.
  - **What about the dashed lines?**
    - The edges (1, 2), (3, 4) and (2, 4) are **cross edges**





Graphs: Breadth-First Search Trees

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

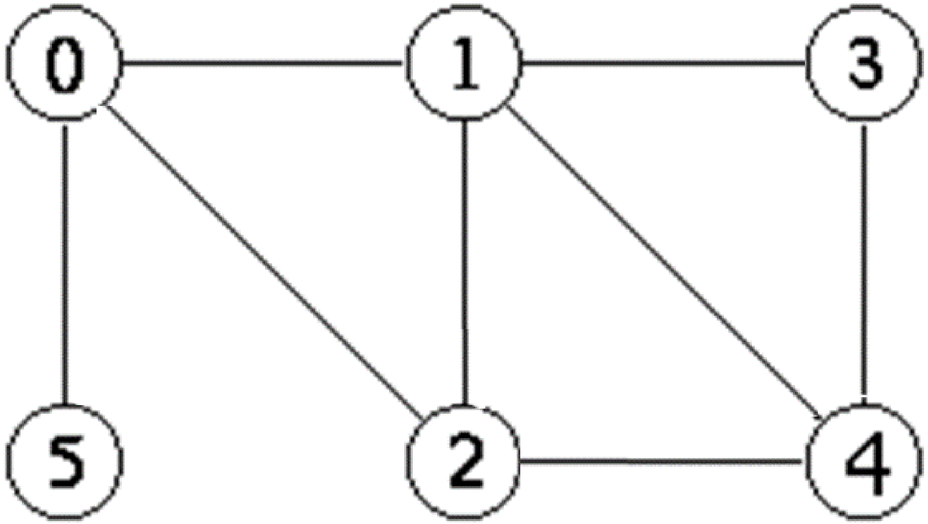
Graphs: Breadth-First Search Trees

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

- This is a subtle observation. Let’s talk about why it’s true.

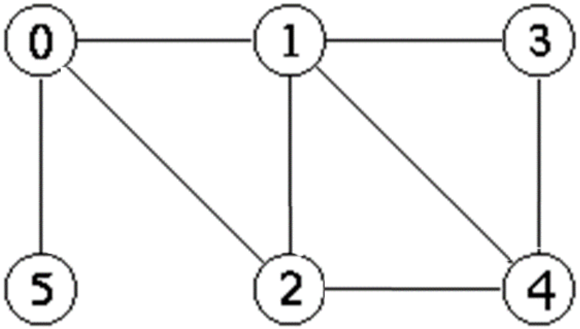
Graphs: Breadth-First Search Trees

- **Fun fact:** If you perform a BFS of our **undirected** graph G, **all edges** are either **tree edges** or **cross-edges**.
- Why aren't other kinds of edges possible?



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.



That’s All For Now…

- Coming to a Slideshow Near You Soon…
  1. Depth-First-Search
  2. Topological Sort

That’s All For Now