SCHOOL OF ENGINEERING

VANDERBILT UNIVERSITY

CS 3250
Algorithms
Lecture 5

Graphs: BFS & Detecting Cycles

VANDERBILT UNIVERSITY

Announcements

- **HW1B Due** Tuesday, September 12th by 9AM.
  - **Brightspace Quiz on Asymptotic Notation** – It’s tricky by design. Be sure to think before answering. You get (2) tries at it. Your highest score is recorded.
  - **Gradescope Written HW** – A few written questions on the topic of asymptotic analysis. Submitted to Gradescope as a PDF.

Data Structures for Graphs

1. Adjacency Matrix –

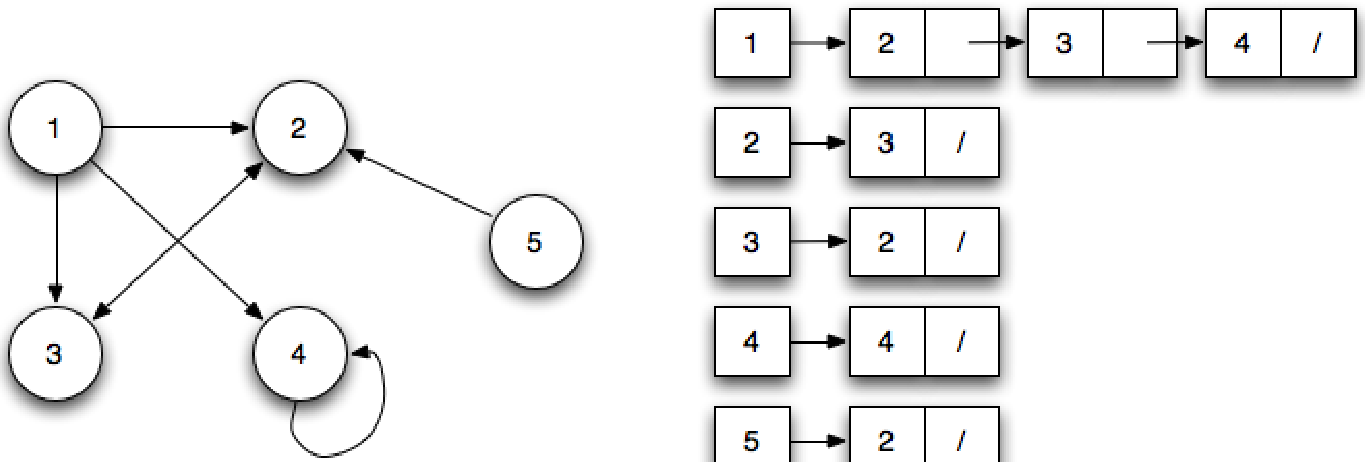
- **Space:** An array of n x n is needed $\Theta(n^2)$ regardless of a sparse or dense graph in terms of number of edges.
- **Speed:** Depends on the question being asked.
    - Is there an edge from vertex 4 to vertex 8? Fast. $\Theta(1)$
    - List all the neighbors of every vertex? Slow. $\Theta(n^2)$
    - Delete vertex 4? Slow. $O(n^2)$

	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

Data Structures for Graphs

There are two main data structures used to represent graph G = (V, E) which contains n vertices and m edges.

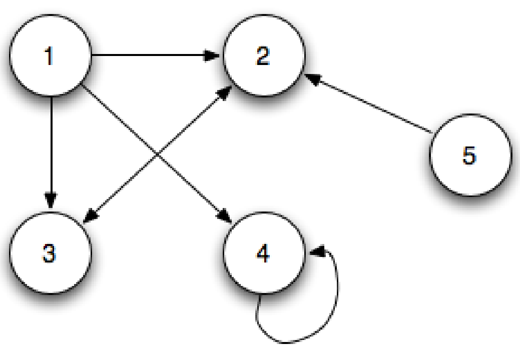
2. **Adjacency List** - An adjacency list consists of an array of |V| pointers, where the i^{th} element points to a linked list of all the edges incident to vertex i. For a **weighted graph** store the edge weight as an additional field in each edge node in the adjacency list.

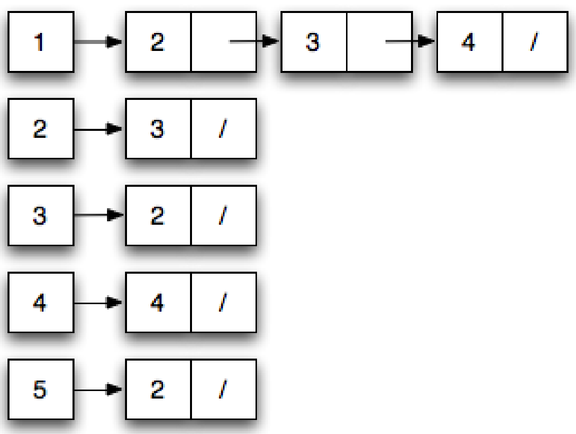


Data Structures for Graphs

2. Adjacency List –

- **Space:** $O(|V| + |E|) = O(V + E) = O(\max(V, E))$
  - Cardinality symbol usually not included.
- **Speed:** Depends on the question being asked.
  - Is there an edge from vertex 4 to vertex 8? $O(V)$ since a vertex might be adjacent to all other vertices.
  - List all the neighbors of every vertex? $\Theta(V + E)$





Adjacency Matrix vs. Adjacency List

Task	Winner
Does edge (x, y) exist?	Adjacency Matrix
What is the degree of vertex 4?	Adjacency List
Add/Delete an edge	Adjacency Matrix
Traverse the graph	Adjacency List
Better for most problems	Adjacency List

Graphs: Breadth-First Search

- Once you construct a graph, the next question becomes to do with the graph?
- Many questions are answered by exploring the graph.
- This idea of exploring a graph via its data structure is better known as a **traversal**.

Graphs: Breadth-First Search

- When traversing a graph, it's important to ensure the following:
  1. **Efficiency** – We don't waste time visiting places we've already explored.
  2. **Accuracy/Correctness** – We don't miss any places along the way (i.e., leave no stone unturned).

Graphs: Breadth-First Search

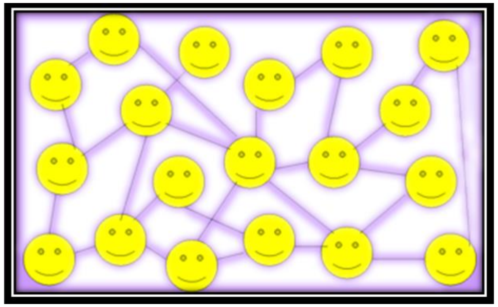
- One of the simplest graph traversals that can be used on a directed or undirected graph is known as **Breadth-First Search (BFS).** You saw it in 2201.
- **Analogy:** Imagine yourself as an explorer.
  - You **gradually** expand the frontier between discovered and undiscovered lands.
  - "lands" we discover and explore are the vertices
  - "paths" we take are the edges in the graph.

Graphs: Breadth-First Search

- Nobody (including your tech interviewer) will ever walk up to you and say, “I’ll give you a bag of cash if you can write a breadth-first search for me.”
- **Breadth-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 BFS is the right tool.

Graphs: Breadth-First Search in Action

- **Breadth-First Search in Action**
  - Shortest path in **unweighted** graph
  - Cycle detection in undirected graph
  - Connected components
  - Social Networks – Who are Kristin's friends?
  - GPS Navigation Systems – What interesting places are near my current location?
  - Search – How many sites link to this site?



Breadth-First Search: How It Works

- The key to any graph traversal is keeping track of where you are.
- **Question:** Which of the following do you think is **not** a "state" that needs to be remembered during a BFS traversal?
  - Undiscovered
  - Discovered
  - Soon to be processed
  - Processed

Breadth-First Search: How It Works

- **Question:** Which of the following do you think is **not** a "state" that needs to be remembered during a BFS traversal?
  - Undiscovered
  - Discovered
  - **Soon to be processed**
  - Processed
- We don't need to worry about soon be processed.

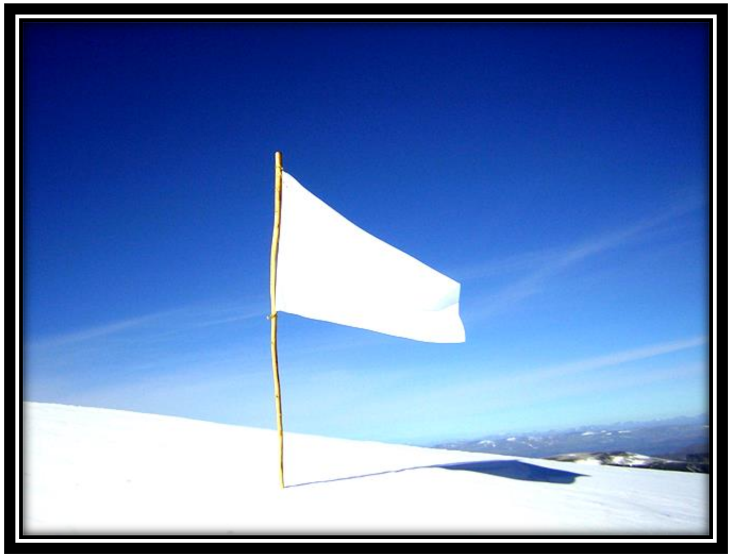
Breadth-First Search: How It Works

- The key to any graph traversal is:
  1. Marking each vertex when first **discovered**
  2. Knowing when each vertex is **finished/processed**.
- We can also leave a breadcrumb to remember our "parent" (the person who discovered us). This is optional but can provide useful information.

Breadth-First Search: How It Works

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

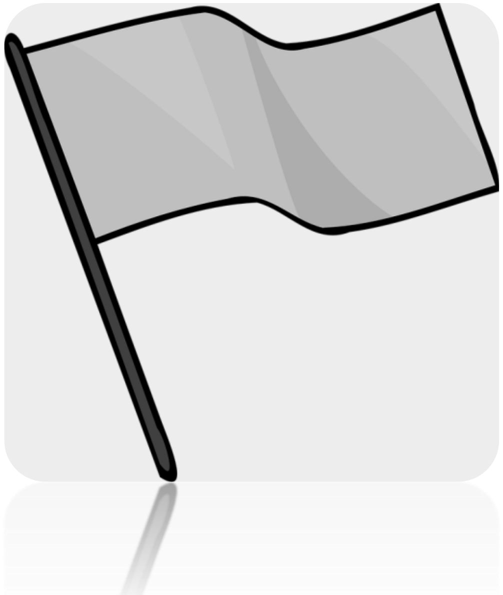
1. **Undiscovered (white)** – I haven't yet discovered this vertex. It is unchartered territory (initially all vertices are undiscovered).



Breadth-First Search: How It Works

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

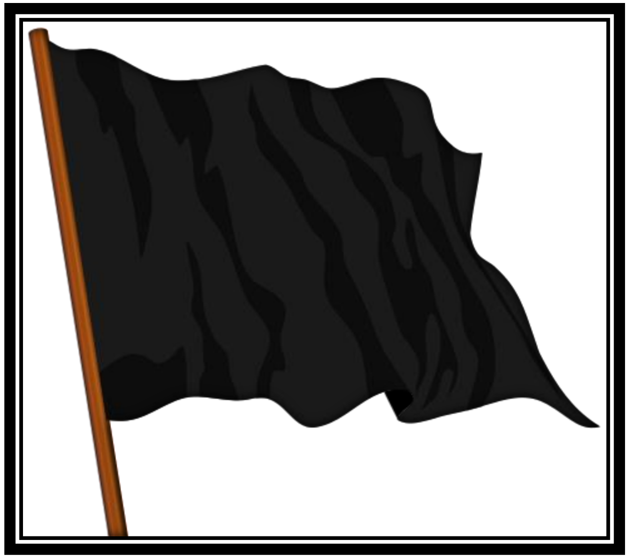
2. **Discovered but unexplored (gray)** – I have discovered this vertex and planted my gray flag here. However, I haven't thoroughly explored the area yet.



Breadth-First Search: How It Works

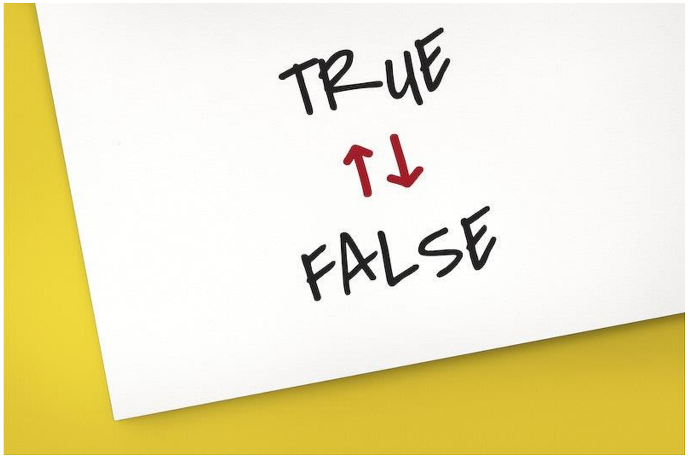
- We will label each vertex to be in one of three states (or colors).

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.



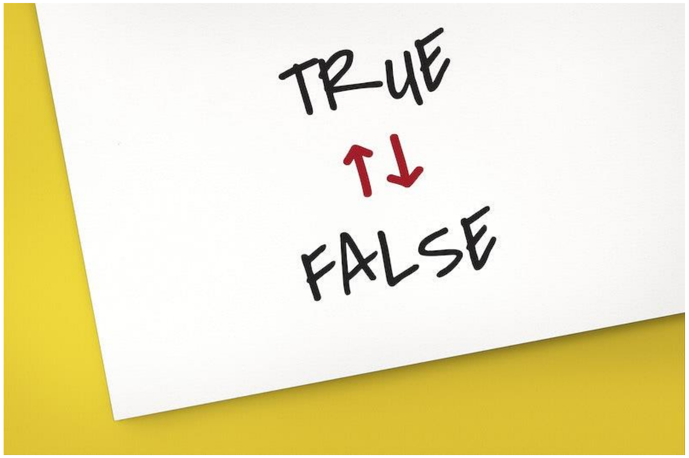
Breadth-First Search:

- **Question:** True/False. Every graph has a unique BFS traversal.



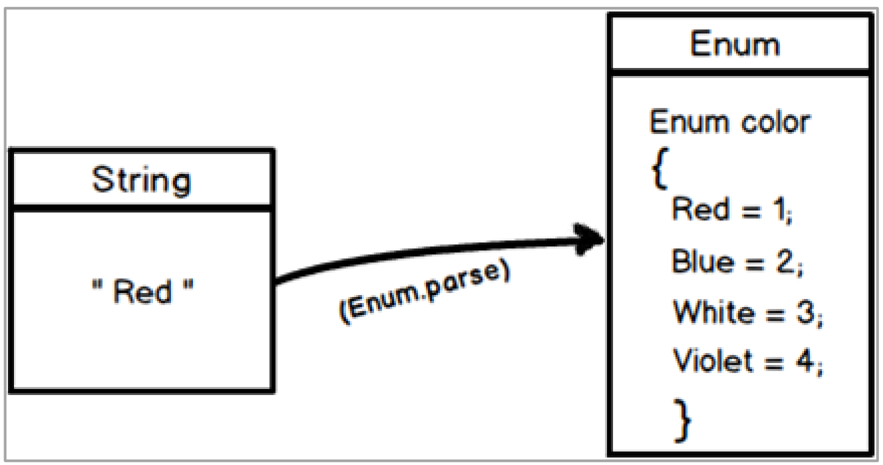
Breadth-First Search:

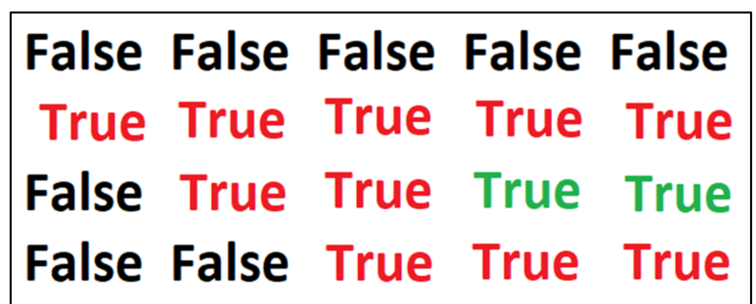
- **Question:** True/False. Every graph has a unique BFS traversal.
- **Answer:** False. It's quite possible to have multiple BFS traversals. That's why we often say on an exam or HW "when given a choice, choose the vertex that comes first alphabetically."



Breadth-First Search: Algorithm Details

- How should we implement these vertex states?
  - Some programmers use Enum types or Constants.
  - Some programmers use boolean arrays.
- The details do not matter provided they do not change the overall algorithm efficiency.





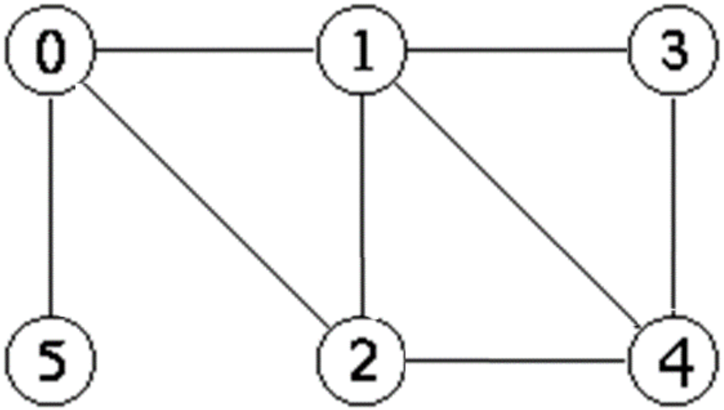
Breadth-First Search: Pseudocode

```
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
```


This Photo by Unknown Author is licensed under CC BY-NC-ND

Example: Breadth-First Search

- **Example:** Perform a BFS on the undirected graph below starting at vertex 0. When given a choice between two vertices, choose the one that comes first numerically.



TO DO LIST

STATE

PROCESSED:

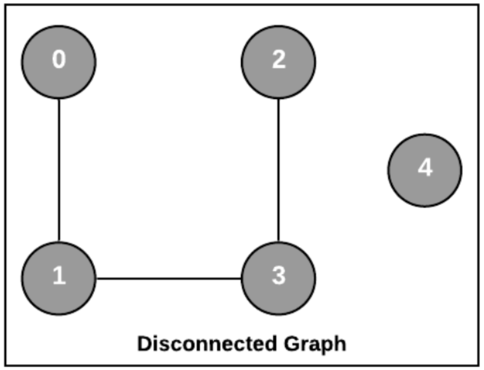
 This Photo courtesy of Zybooks

Breadth-First Search:

- **Question:** Does our breadth-first search algorithm work correctly for any undirected graph G?
  A. Yes of course. We’re experienced programmers.
  B. Definitely not. We’re experienced programmers.
  C. I thought it did, but now I'm not sure.

Breadth-First Search:

- **Answer: Definitely not**. Our current BFS algorithm does not work correctly for all types of undirected graphs. Consider the disconnected graph G shown below and a BFS traversal that starts at vertex 0. Our current algorithm would never find vertex 4.
- We need to add a controlling loop to ensure we hit all vertices in the graph.

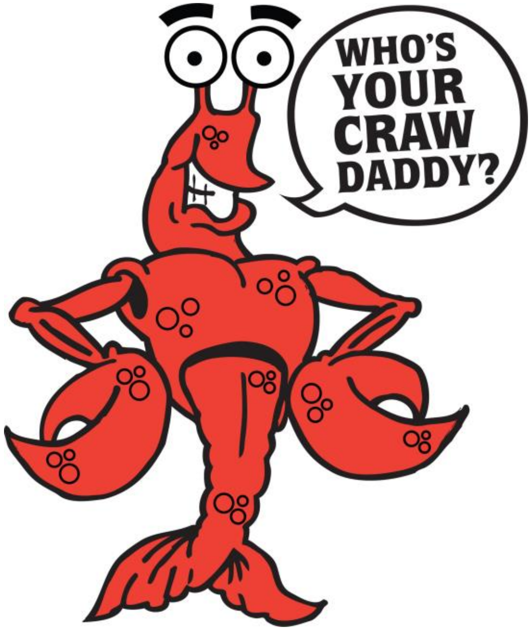


Breadth-First Search: Optional Tasks

- You may decide to perform additional work during your graph traversal depending on your needs.
- This is fine provided you do not sacrifice overall algorithm efficiency.
- Here are some tasks commonly performed with BFS.

Breadth-First Search: Optional Tasks

- **Task: Who's Your Daddy?**
  - **What:** Who discovered us (i.e., our "parent")?
  - **When:** Helpful if you need to report the shortest path from one vertex to another.



Breadth-First Search: Optional Tasks

- **Task: Level Up!**
  - **What:** Remember every vertex's distance from the starting vertex (e.g., "six degrees of separation").
  - **When:** Helpful if you need to determine the shortest path from one vertex to another.

Breadth-First Search: Optional Details

Placeholders for tasks in our BFS algorithm:

1. **processVertexEarly(v)** – Optional method you can include if you need to do pre-work before exploring a vertex.
2. **processVertexLate(v)** – Optional method you can include if you need to do post-work after exploring a vertex.
3. **processEdge (v, w)** – Optional method you can include if you need to do any work while processing an edge.

Thinking About Breadth-First Search

Now, let's THINK about the algorithm as an algorist might.

- **Question:** Given an undirected graph, G, how many times can we discover a particular vertex?

Thinking About Breadth-First Search

Let’s THINK about the algorithm as an algorist might.

- **Question:** Given an undirected graph, G, how many times can we discover a particular vertex?
- **Answer: Once.** I can only discover it once at which point, I plant my flag in it. If I (or anyone else) sees that land again, they will see my flag and know it’s already been discovered.

Thinking About Breadth-First Search

Questions: Now, THINK about the algorithm as an algorist might. Given an undirected graph, G:

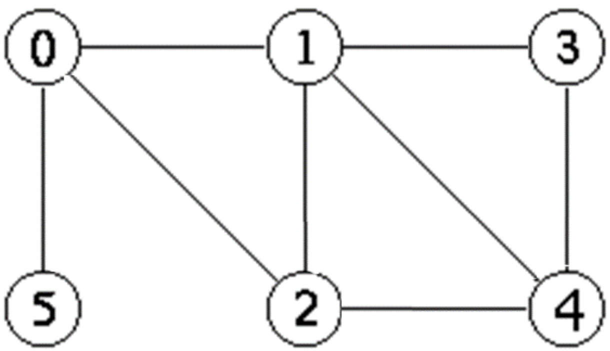
1. ~~How many times can we discover a vertex?~~
2. Could a neighbor of the active vertex already be…
   a. Discovered but not processed (gray)?
   b. Discovered and processed (black)?
3. What color are the vertices on the TO-DO queue?
4. Can there be more than one copy of a vertex on the queue?
5. How can you tell if there's a cycle in an undirected graph?

Thinking About Breadth-First Search

Questions: Now, THINK about the algorithm as an algorist might. Given an undirected graph, G:

2. **Could a neighbor of the active vertex be…**
    a. Discovered but not processed (gray)?

Answer: Yes. In the graph below, consider vertex 1 which we discover while actively working on vertex 0, but will not be processed until later in the BFS.

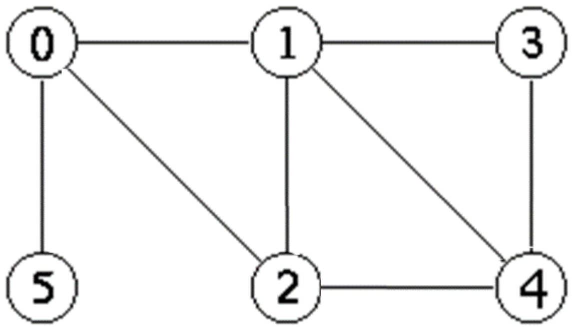


Thinking About Breadth-First Search

Questions: Now, THINK about the algorithm as an algorist might. Given an undirected graph, G:

2. Could a neighbor of the active vertex be…
   b. Discovered and processed (black)?

Answer: Yes, of course. At some point during the BFS, the vertex 0 will be flagged as both discovered and then processed.

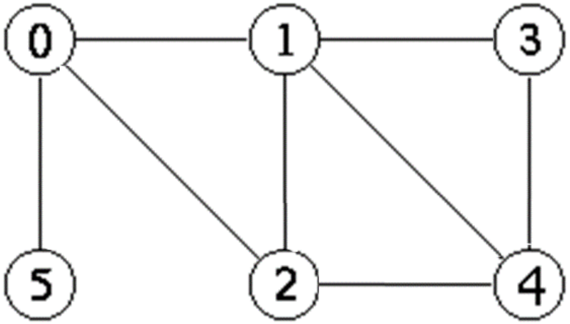


Thinking About Breadth-First Search

Questions: Now, THINK about the algorithm as an algorist might. Given an undirected graph, G:

3. What color are the vertices on the TO-DO queue?

Answer: They are gray because they are discovered but not processed.

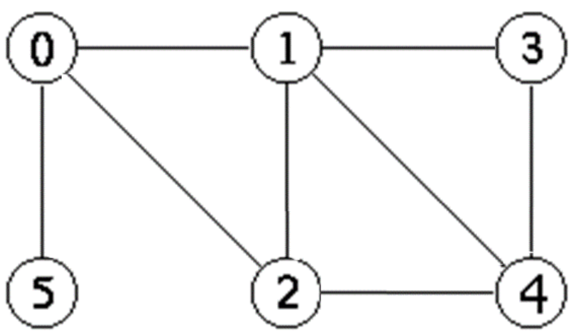


Thinking About Breadth-First Search

Questions: Now, THINK about the algorithm as an algorist might. Given an undirected graph, G:

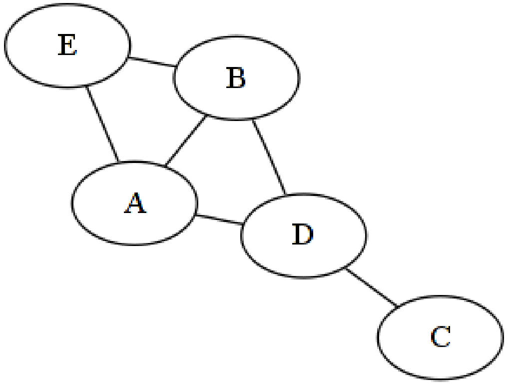
4. Can there be more than one copy of a vertex on the queue?

Answer: No. Once a vertex is placed on the queue, it is marked gray/discovered. As we've previously noted, a vertex can only be discovered once.



Breadth-First Search: Detecting Cycles

- **Question:** 5. How can we detect a cycle in an **undirected** graph?
- **Answer:** If we encounter a vertex during traversal that is in the discovered state, but "we" didn't discover it, we've found a cycle (i.e., got to same vertex via a different edge).



Breadth-First Search: Detecting Cycles

- Let’s expand on this idea and modify our BFS to detect cycles in a **connected**, **undirected** graph.
- Hmm…what do we need to do and where?
  - If we are looking at an edge that has not been processed, we should ask whether this edge leads to a vertex that has already been discovered by someone else.
  - If that vertex has already been discovered by someone else, then there must be two ways to get to it.
  - That means we just found a cycle.

Breadth-First Search: Detecting Cycles

```
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)
    FOR (all of v's neighbors w)
          IF (state[w] != PROCESSED) THEN
                processEdge(v, w)
          IF (state[w] == UNDISCOVERED) THEN
                Change status of vertex w to DISCOVERED)
                Add w to the TO-DO list (enqueue)
                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
 LOOP
END BFS
```

Breadth-First Search: Detecting Cycles

- Now for the important piece…

```
processEdge(x, y)
   IF ((state[y] == DISCOVERED) THEN
        PRINT "Cycle found:” + x + "," + y)
END Method
```

- **Question:** Would our current code also work for determining cycles in a **directed** graph?

Breadth-First Search: Detecting Cycles

- **Question:** Would our current code 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

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 Summary

- **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:** Shortest Path (unweighted), Spanning Tree (undirected), Social-Networks.

That’s All For Now…

- Coming to a Slideshow Near You Soon…
  1. BFS Tree of Discovery
  2. Analysis of the BFS algorithm
  3. Applications of BFS algorithm

That’s All For Now