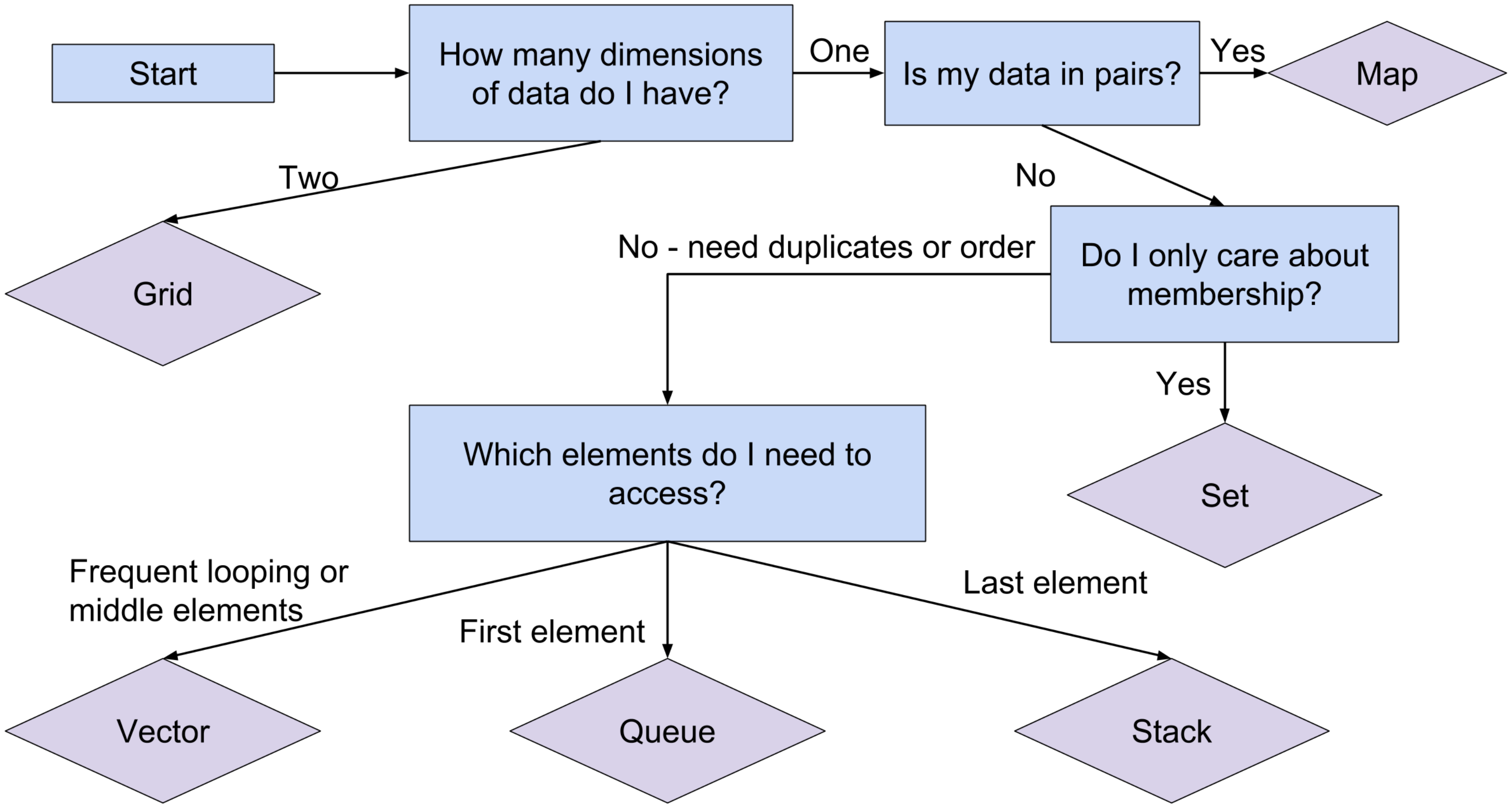
CS 106B, Lecture 23
Graphs

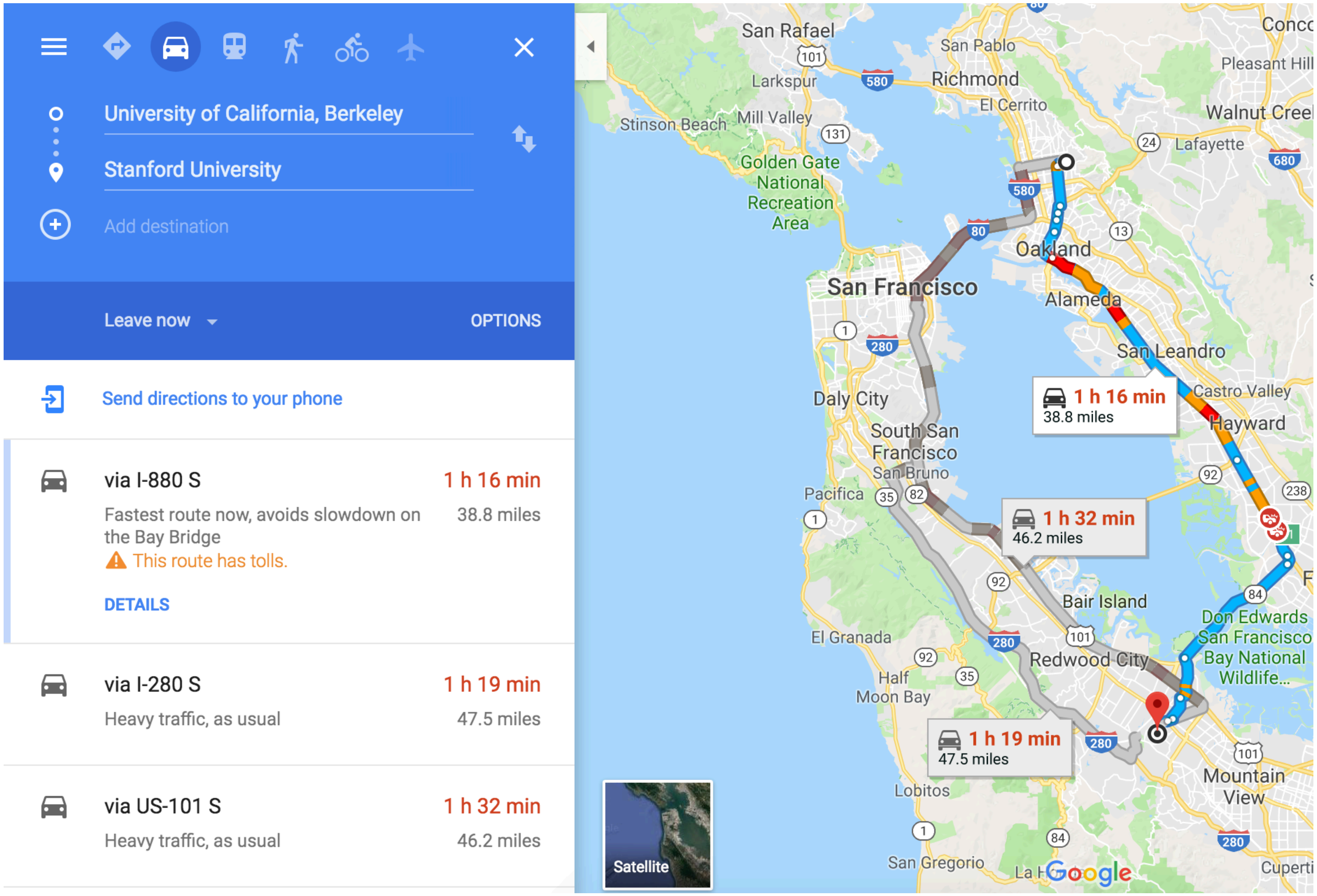
Plan for Today

- Graphs!
  - How to model problems using a graph

ADT Flowchart


Start
How many dimensions of data do I have?
One
Is my data in pairs?
Yes
Map
Two
Grid
No
Do I only care about membership?
No - need duplicates or order
Yes
Set
Which elements do I need to access?
Frequent looping or middle elements
First element
Last element
Vector
Queue
Stack

Google Maps



University of California, Berkeley

Stanford University

Add destination

Leave now

OPTIONS

Send directions to your phone

via I-880 S — 1 h 16 min

Fastest route now, avoids slowdown on the Bay Bridge — 38.8 miles

⚠ This route has tolls.

DETAILS

via I-280 S — 1 h 19 min

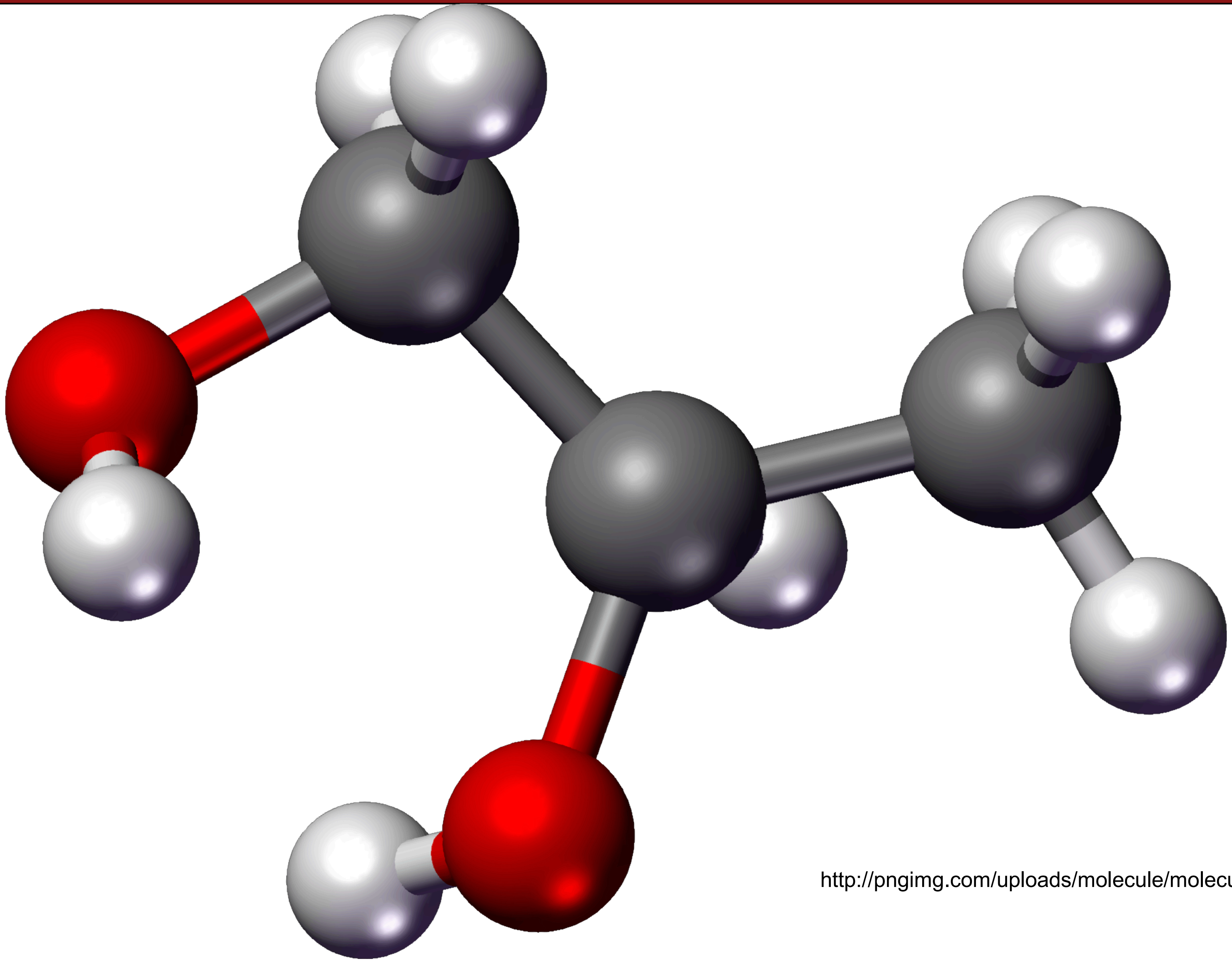
Heavy traffic, as usual — 47.5 miles

via US-101 S — 1 h 32 min

Heavy traffic, as usual — 46.2 miles

Source: https://www.google.com/maps

Molecules



http://pngimg.com/uploads/molecule/molecule_PNG50.png

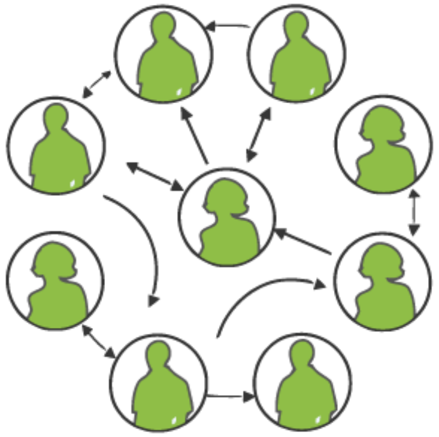
Introducing: The Graph

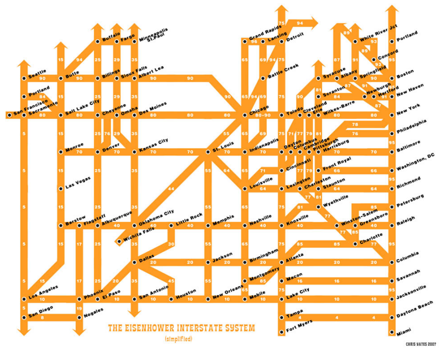
- A **graph** is a mathematical structure for representing relationships
- Consists of **nodes** (aka vertices) and **edges** (aka arcs)
  - **edges** are the relationships, **nodes** are the items

- Examples:
  - Map: cities (nodes) are connected by roads (edges)
  - Molecules: atoms (nodes) are connected by bonds (edges)

Graph examples

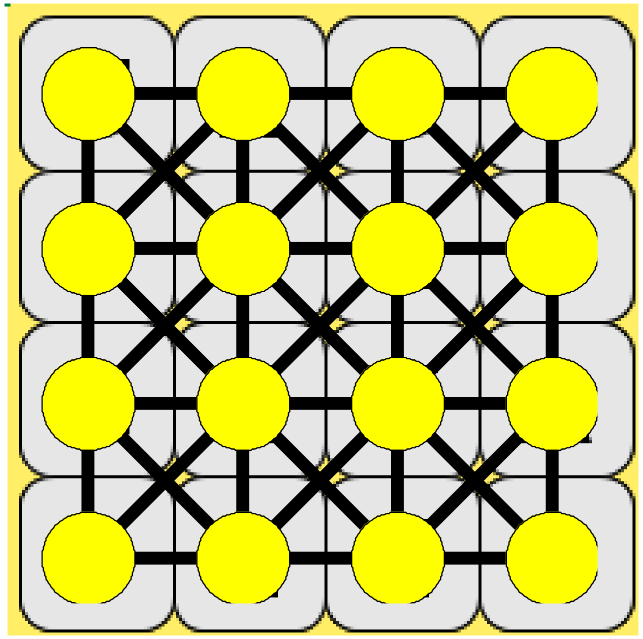
- For each, what are the nodes and what are the edges?
  - **Web pages with links**
  - **Functions in a program that call each other**
  - Airline routes
  - **Facebook friends**
  - **Course pre-requisites**
  - Family trees
  - Paths through a maze





Boggle as a graph

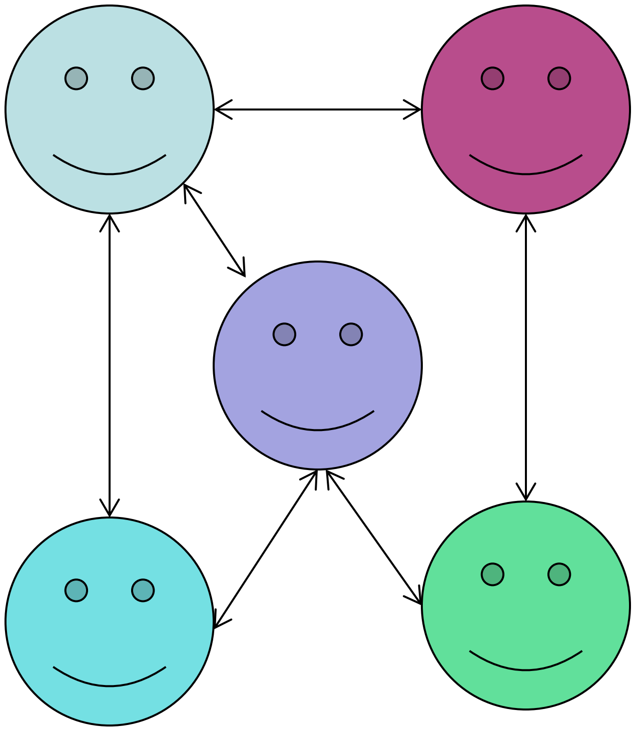
- **Q:** If a Boggle board is a graph, what is a node? What is an edge?
  - **A.** Node = letter cube, Edge = Dictionary (lexicon)
  - **B.** Node = dictionary word; Edge = letter cube
  - **C.** Node = letter; Edge = between each letter that is part of a word
  - **D.** Node = letter cube; Edge = connection to neighboring cube
  - **E.** None of the above

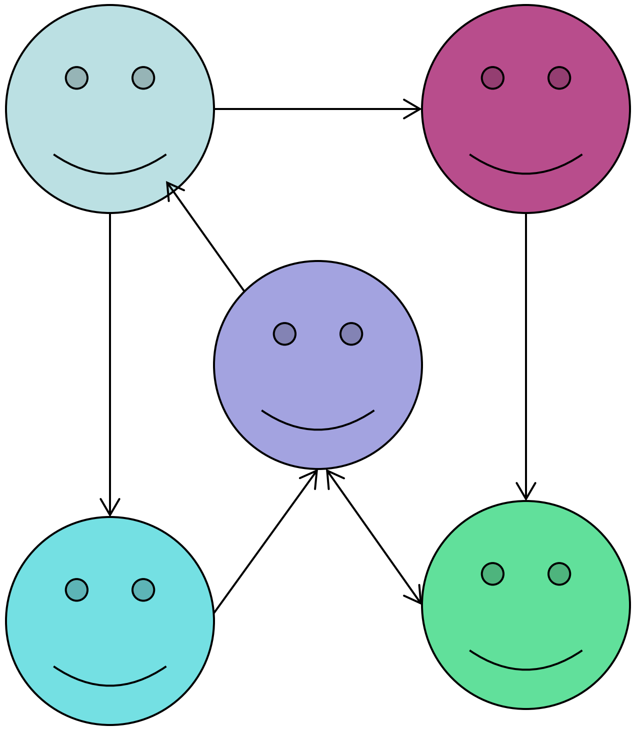


Undirected vs. Directed

- Some relationships are mutual
  - Facebook

- Some are one-way
  - Twitter/Instagram
  - Doesn't mean that all relationships are non-mutual





Representing Graphs

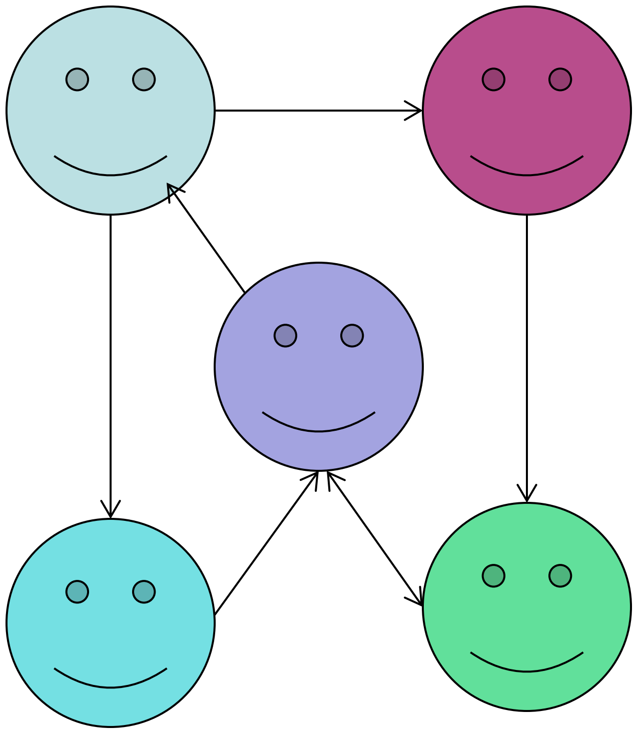
- Two main ways:
  - Have each node store the nodes it's connected to (**adjacency list**)
  - Have a list of all the edges (**edge list**)

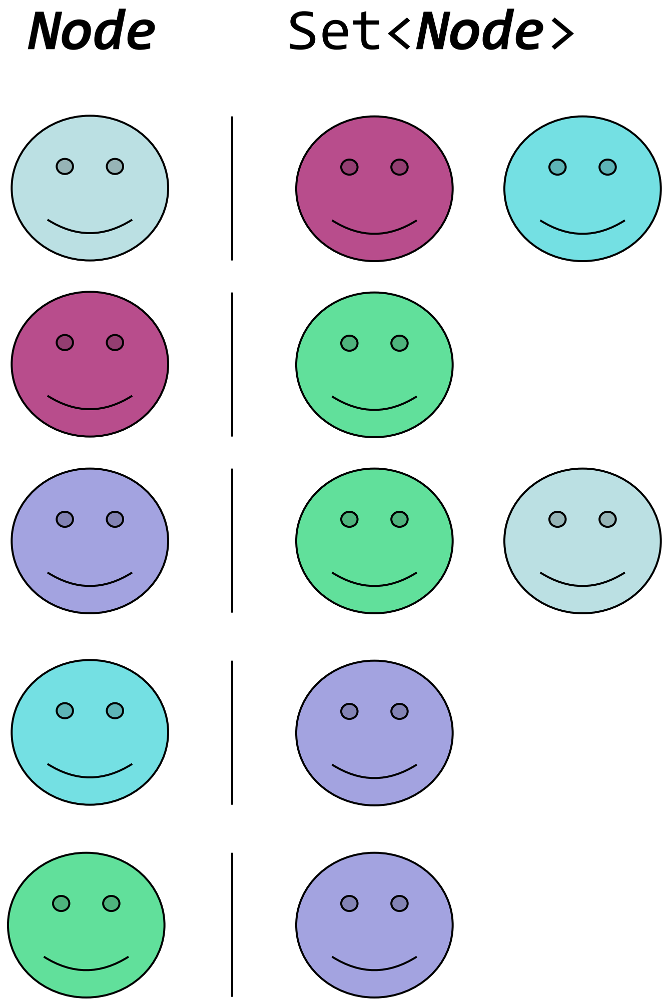
- The choice depends on the problem you're trying to solve
- You can sometimes represent graphs implicitly instead of explicitly storing the edges and nodes
  - e.g. Boggle, WordLadder
  - draw a picture to see the graph more clearly!

- Was the backtracking (wiki links) problem on the midterm a graph problem? How did we represent the graph?

Adjacency List

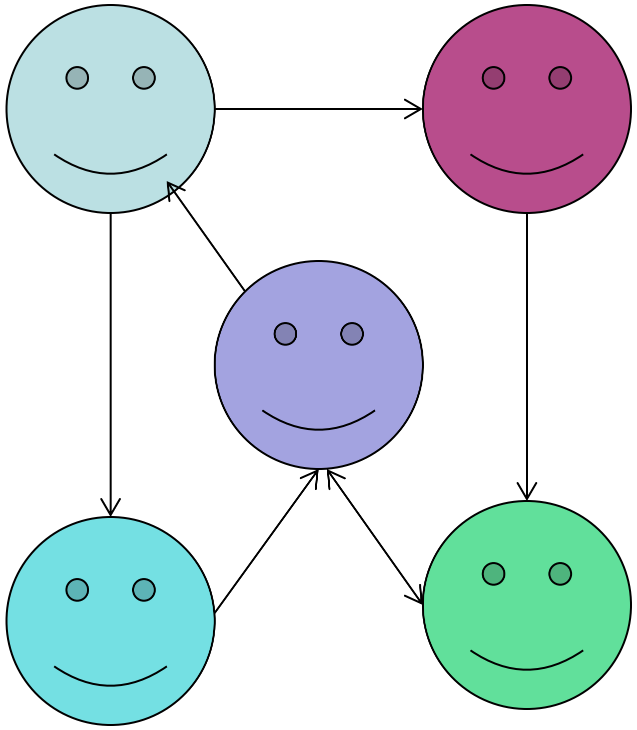
- Map<***Node***, Vector<***Node***>>
  - or `Map<`***`Node`***`, Set<`***`Node`***`>>`

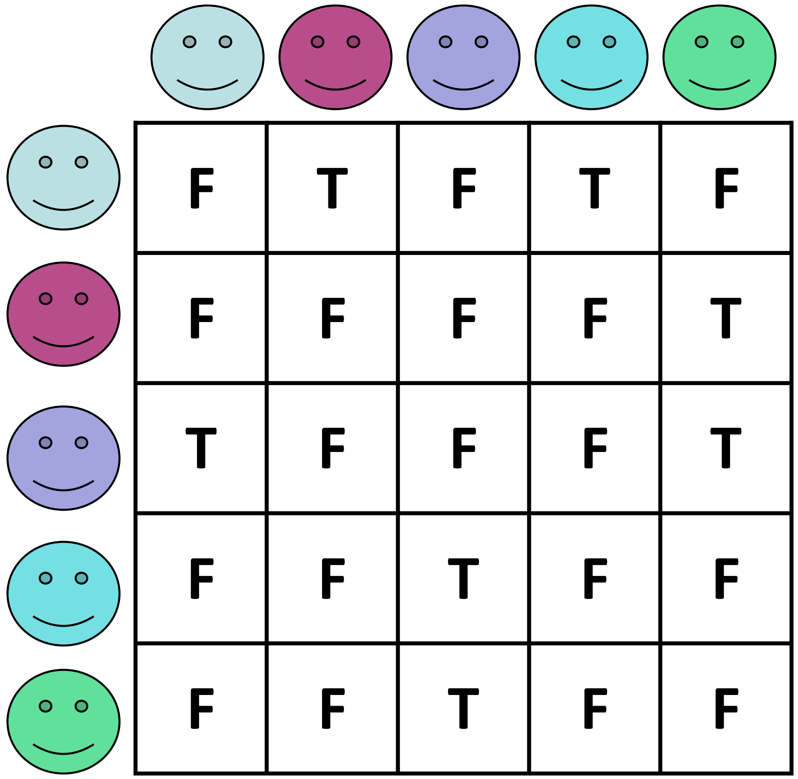




Adjacency Matrix

- Store a boolean grid, rows/columns correspond to nodes
  - Alternative to Adjacency List

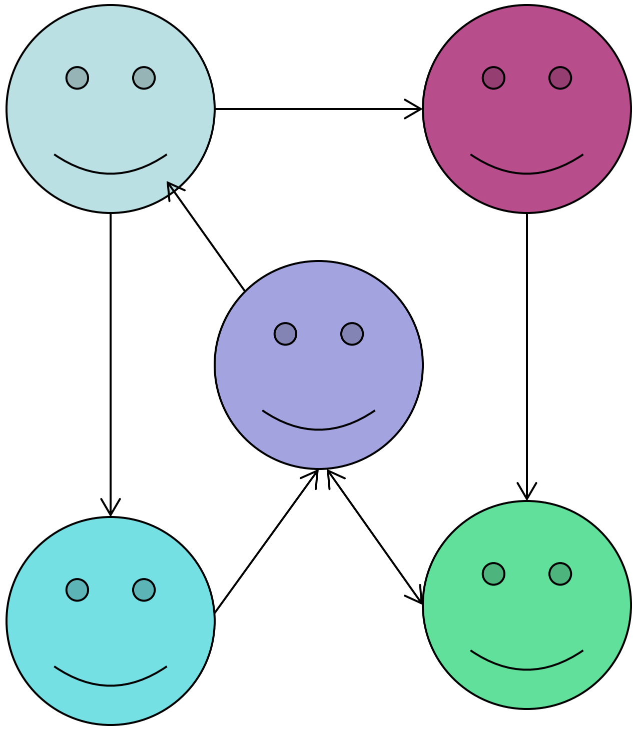




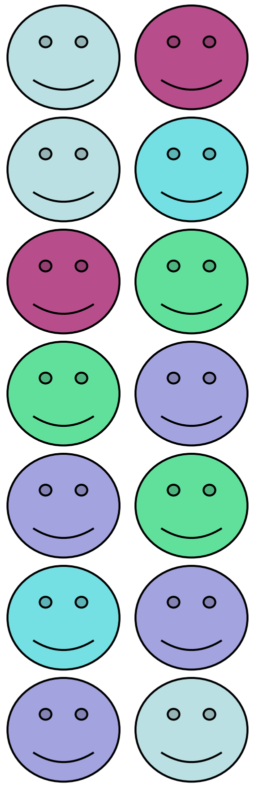
F	T	F	T	F
F	F	F	F	T
T	F	F	F	T
F	F	T	F	F
F	F	T	F	F

Edge List

- Store a `Vector<`***`Edge`***`>` (or `Set<`***`Edge`***`>`)
  - ***Edge*** struct would have the two nodes

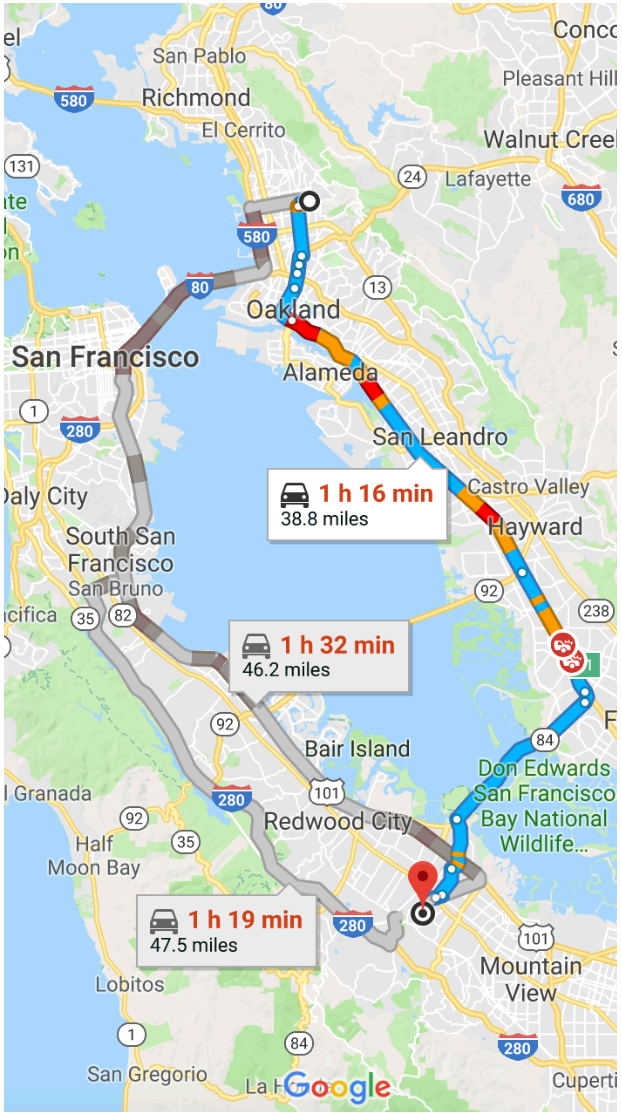


`Vector<`***`Edge`***`>`



Edge Properties

- Not all edges are created equally
  - Some have greater **weight**
- Real life examples:
  - Flight costs
  - Miles on a road
  - Time spent on a road
- Store a number with each edge corresponding to its weight

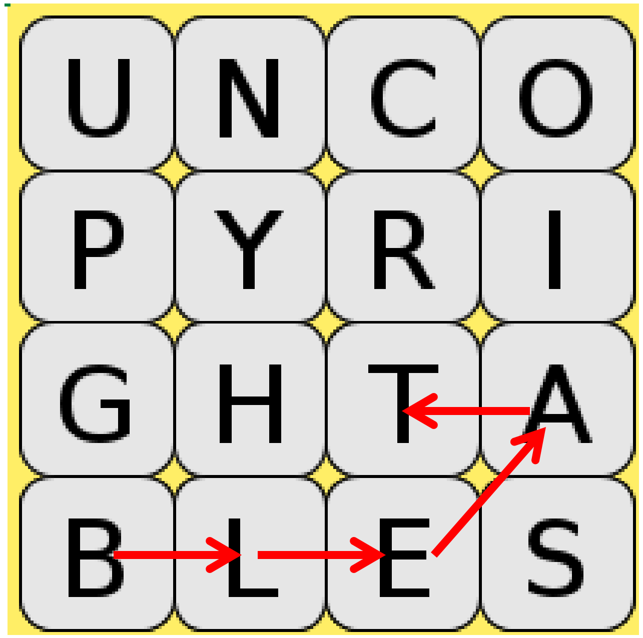


Source: https://www.google.com/maps

Paths

- I want a job at Google. Do I know anyone who works there? What about someone who knows someone?
- I want to find this word on a board made of letters "next to" each other (Boggle)
- A **path** is a sequence of nodes with edges between them connecting two nodes
  - Could store edges instead of nodes (why?)
  - You know Jane. Jane knows Sally. Sally knows knows Sergey Brin, the founder of Google, so the path is:

    You->Jane->Sally->Sergey

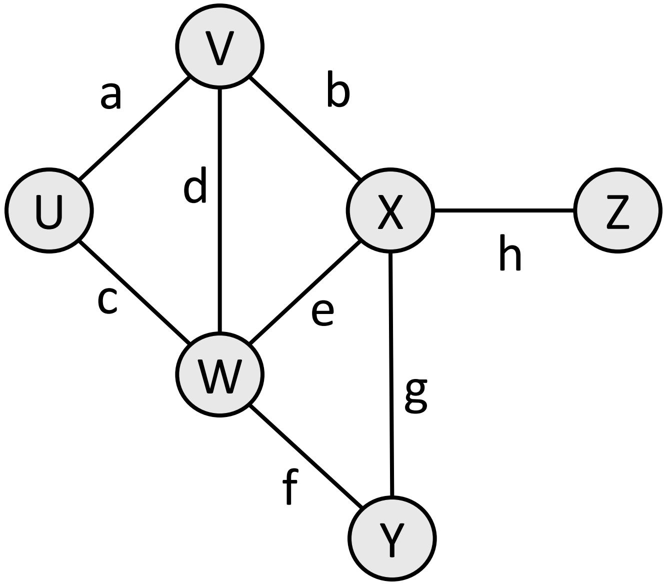


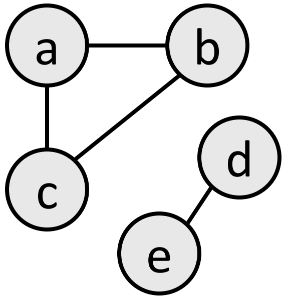
Other graph properties

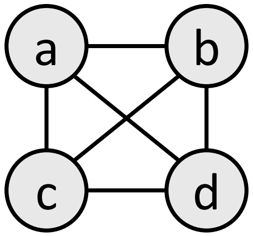
- **reachable**: Vertex *u* is *reachable* from *v* if a path exists from *u* to *v*.

- **connected**: A graph is *connected* if every vertex is reachable from every other.

- **complete**: If every vertex has a direct edge to every other.

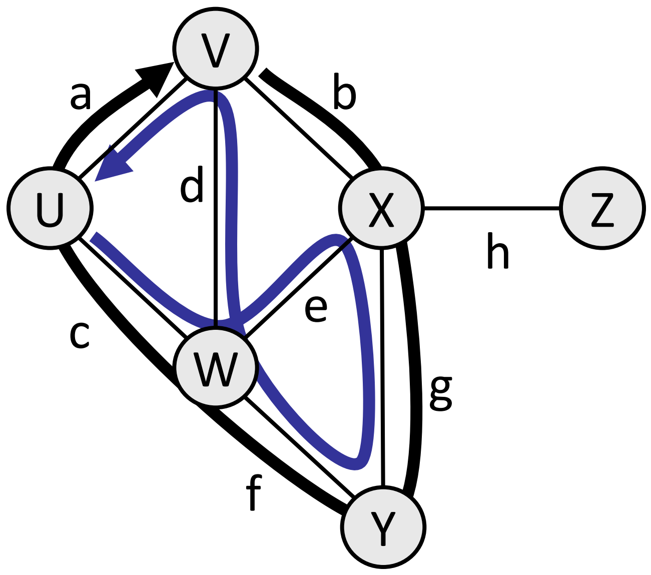






Loops and cycles

- **cycle**: A path that begins and ends at the same node.
  - example: {b, g, f, c, a} or {V, X, Y, W, U, V}.
  - example: {c, d, a} or {U, W, V, U}.
  - **acyclic graph**: One that does not contain any cycles.
- **loop**: An edge directly from a node to itself.
  - Many graphs don't allow loops.



Types of Graphs

- Boggle?
  - undirected, unweighted, cyclic, connected
- A molecule?
  - undirected, weighted, potentially cyclic, connected
- A map of flights?
  - directed, weighted, cyclic, perhaps not connected
- A tree?
  - directed, acyclic graph, not connected

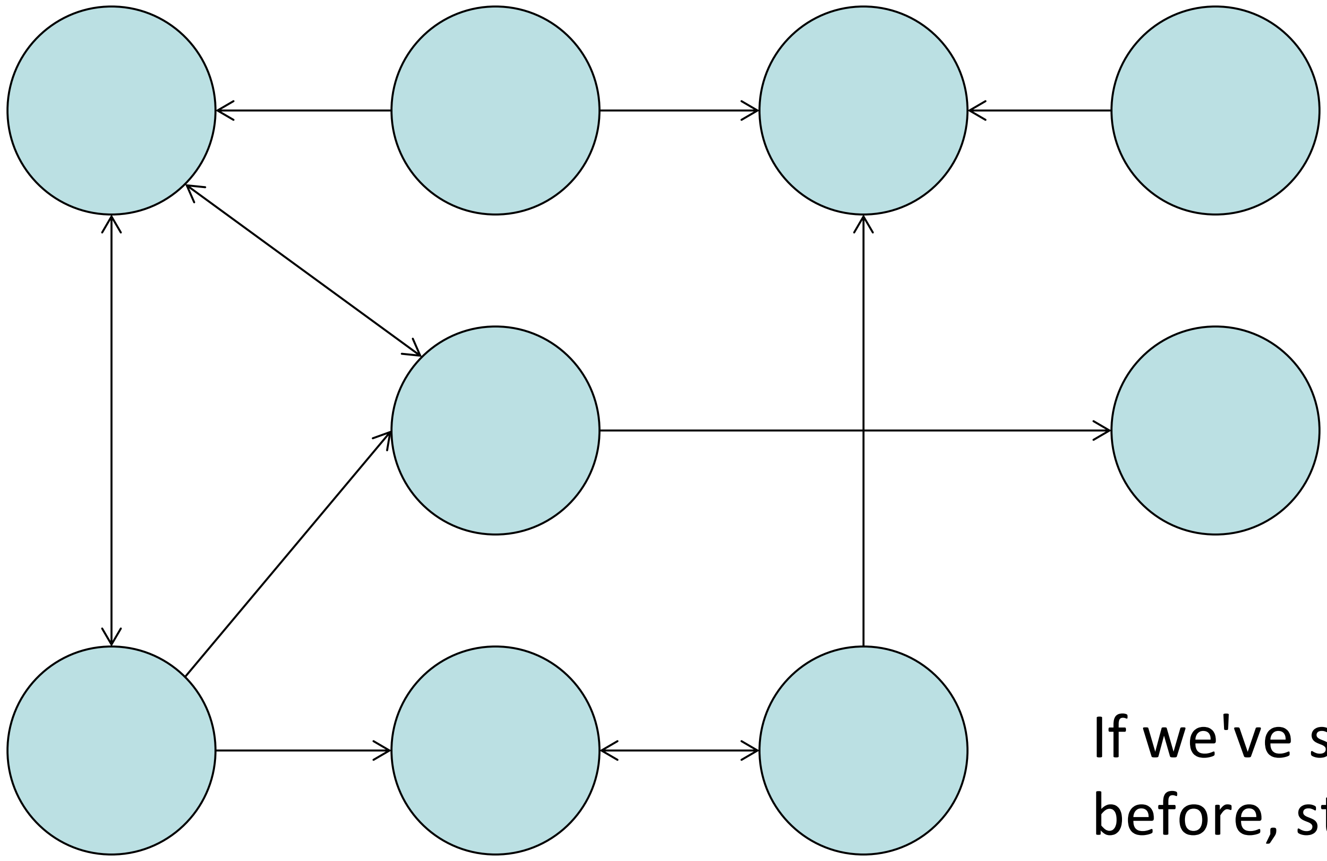
Announcements

- Assn. 6 is due Thursday

Finding Paths

- Easiest way: Depth-First Search (DFS)
  - Recursive backtracking!
- Finds a path between two nodes if it exists
  - Or can find all the nodes **reachable** from a node
    - Where can I travel to starting in San Francisco?
    - If all my friends (and their friends, and so on) share my post, how many will eventually see it?

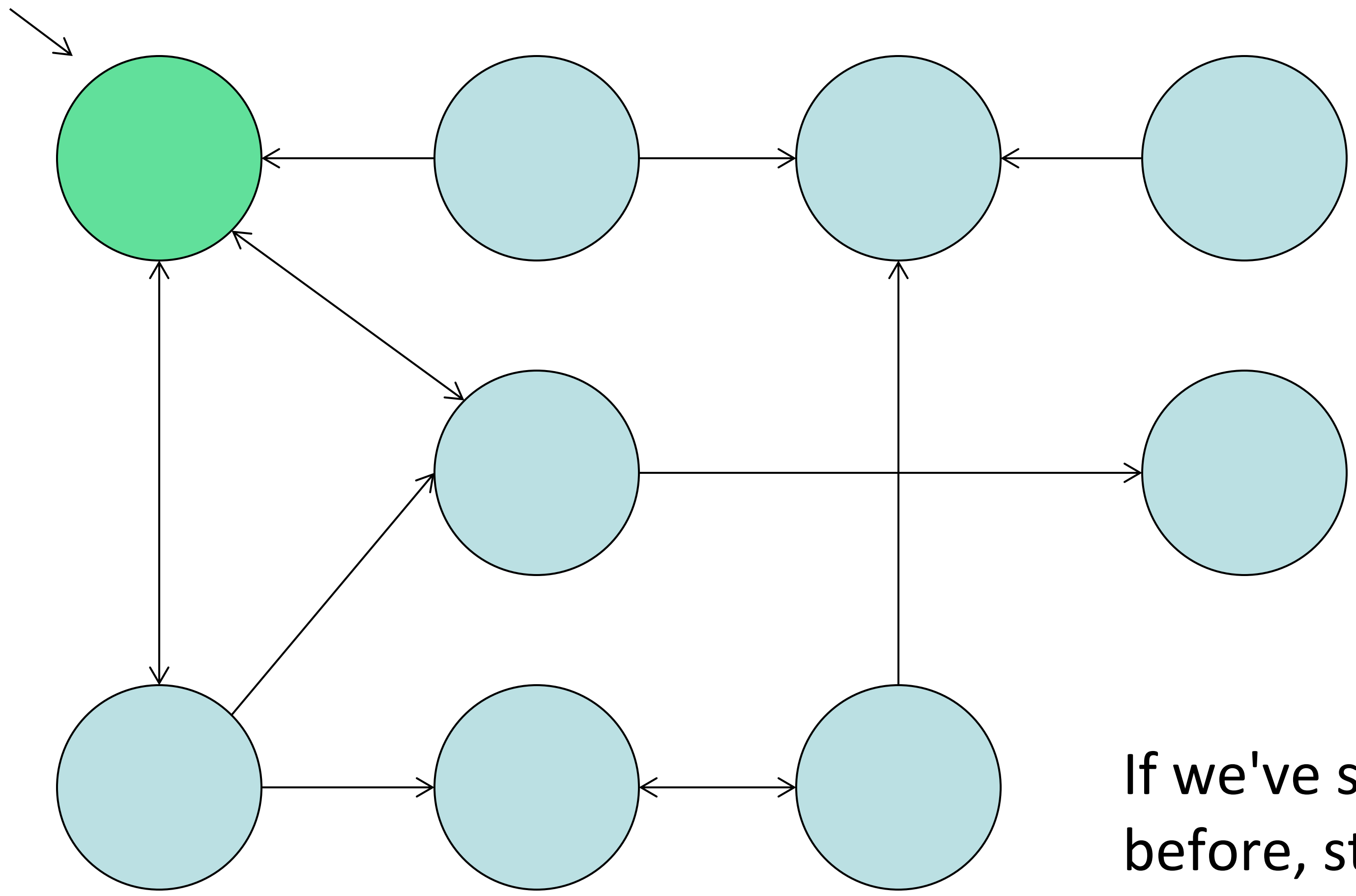
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

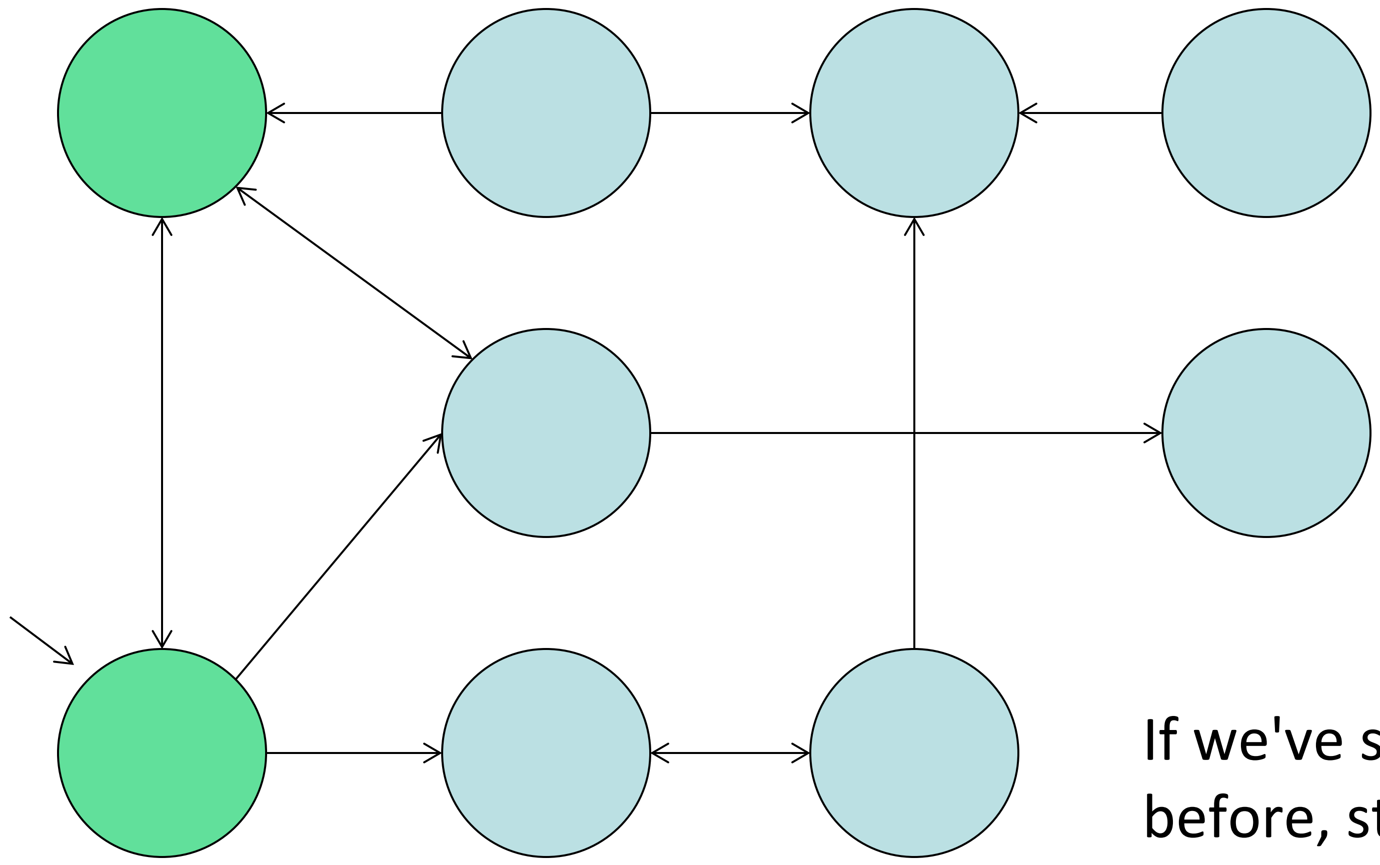
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

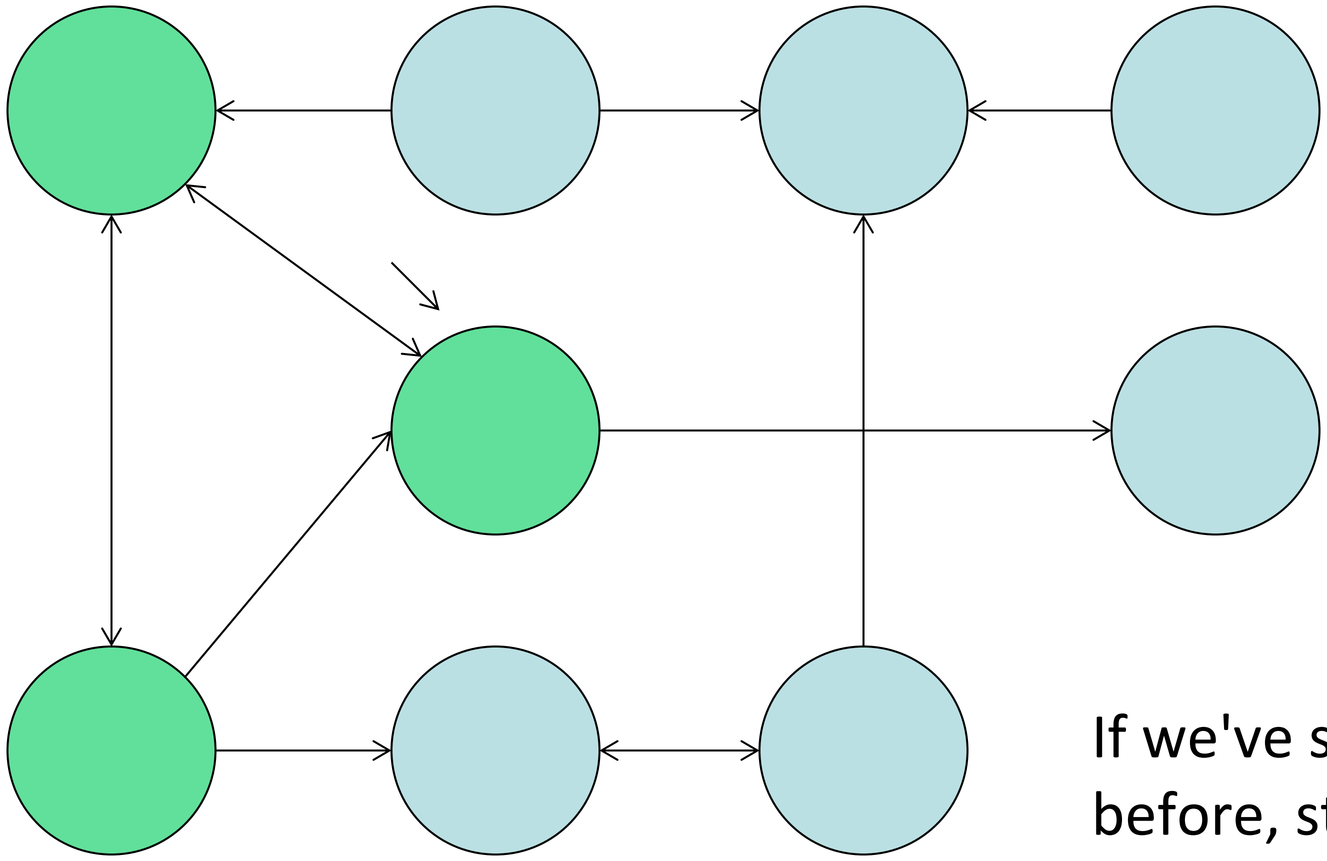
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

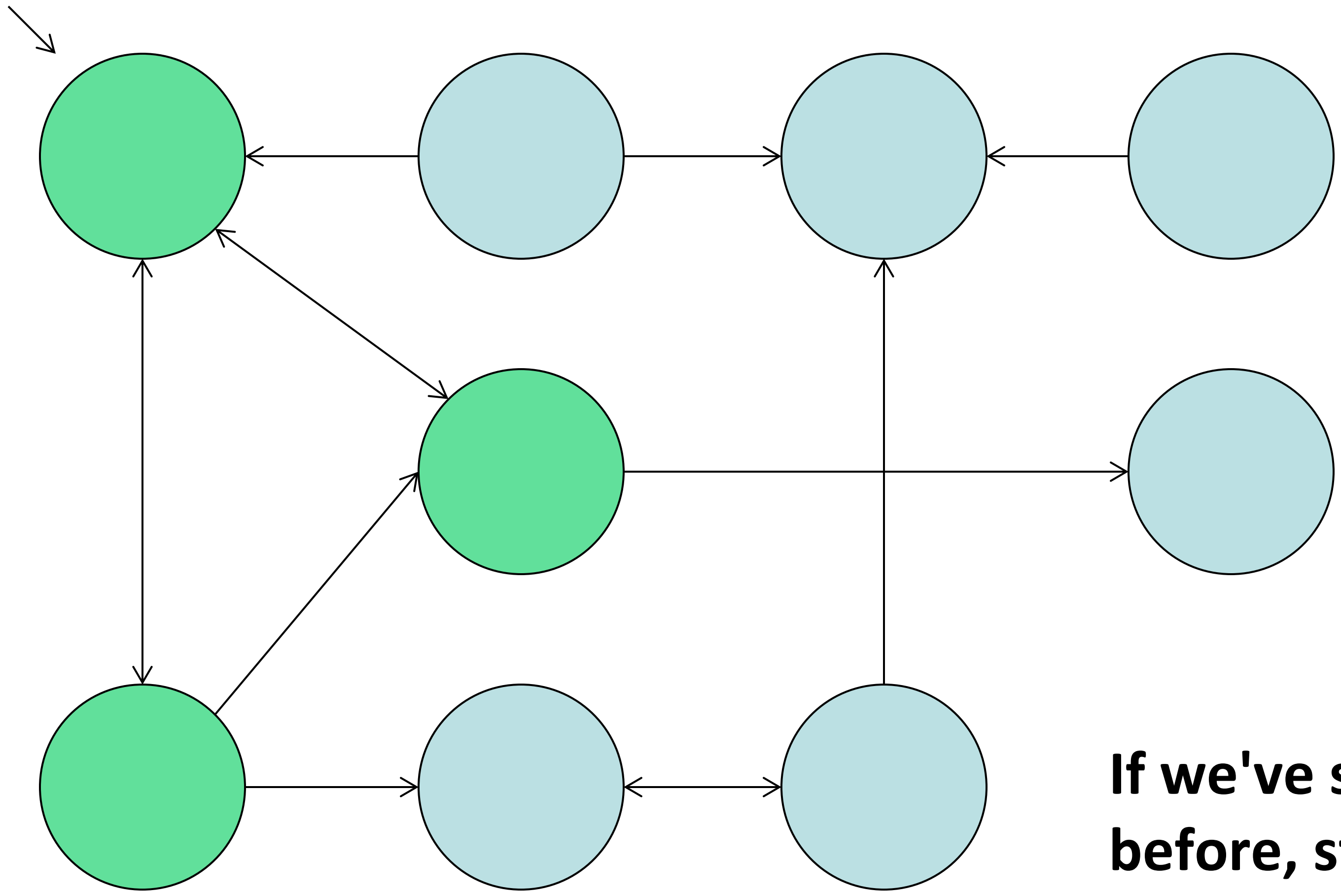
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

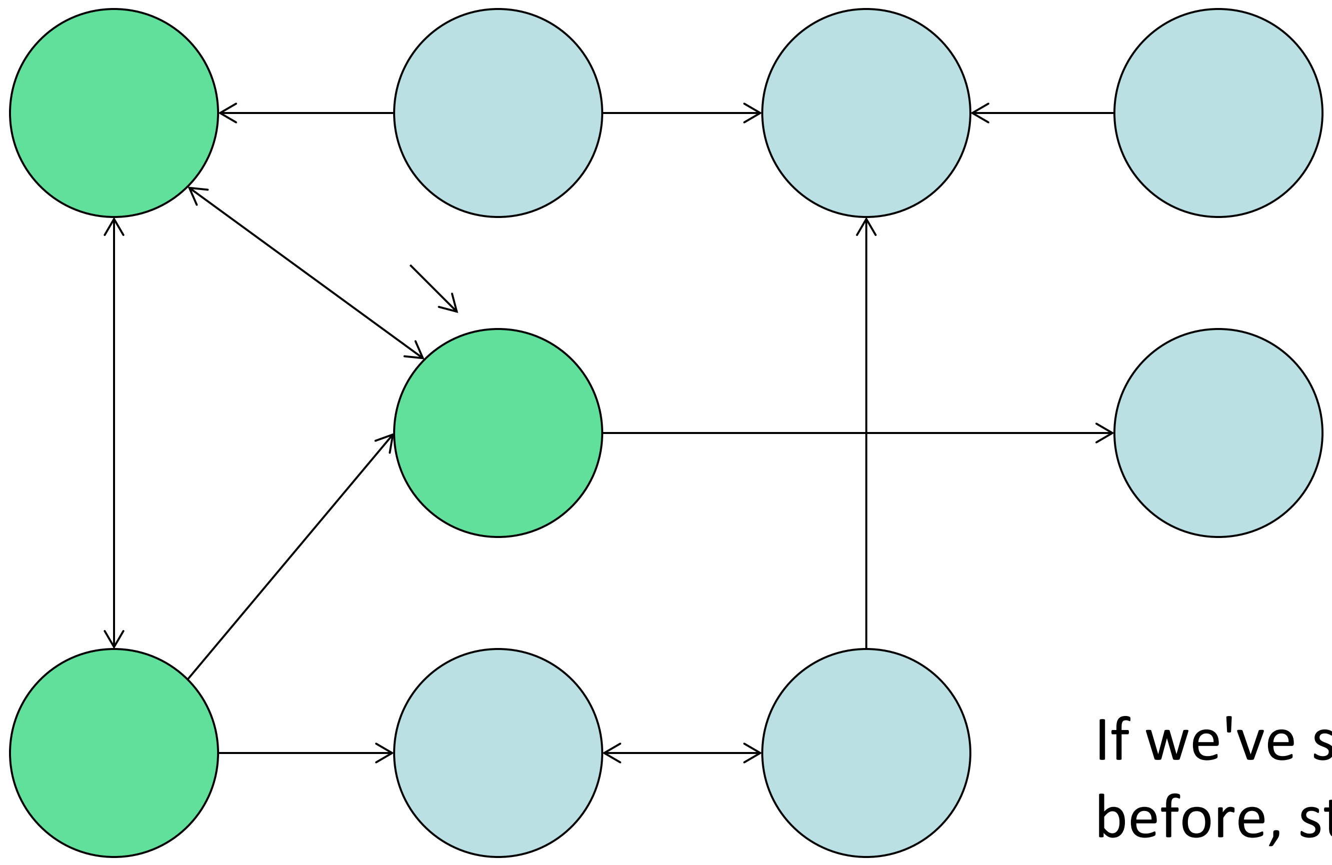
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

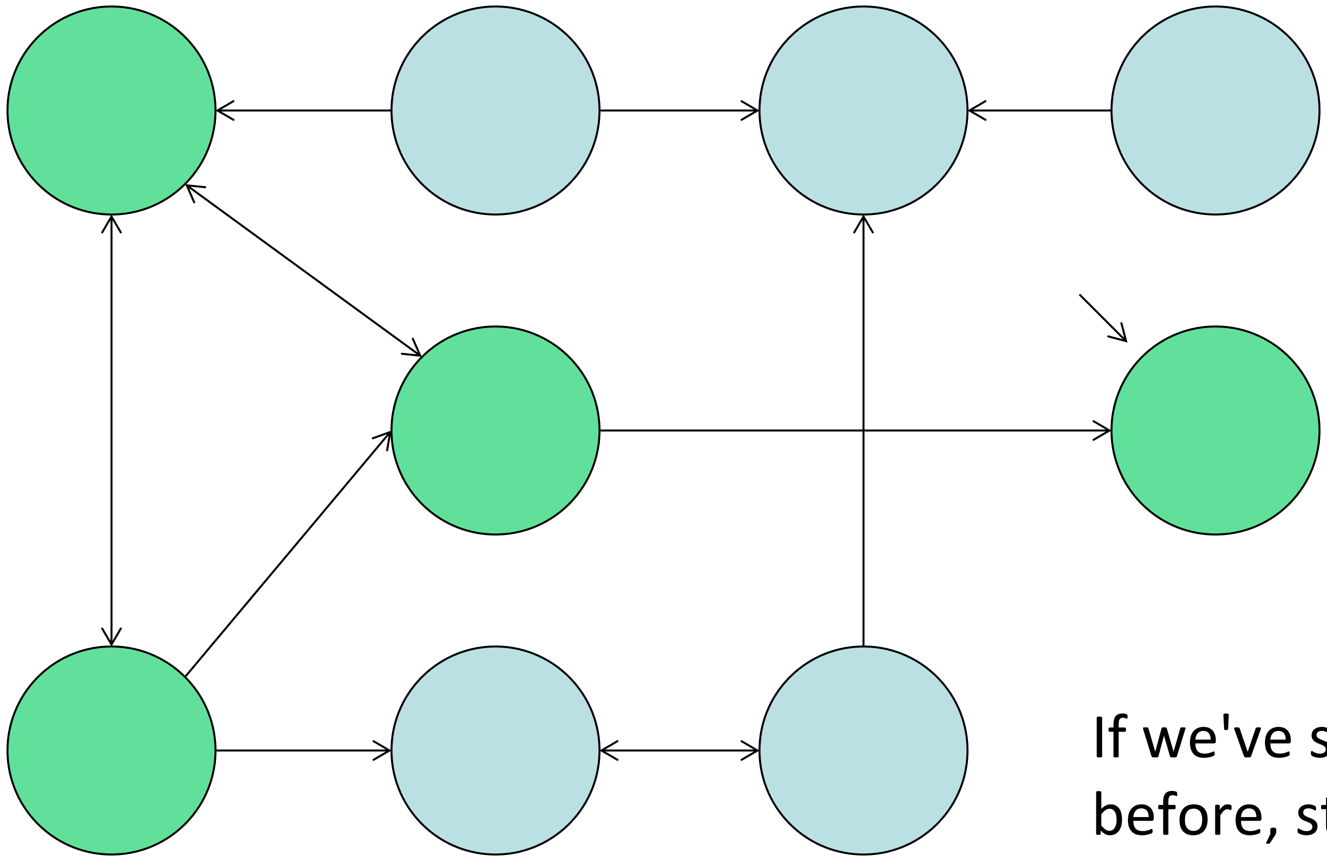
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

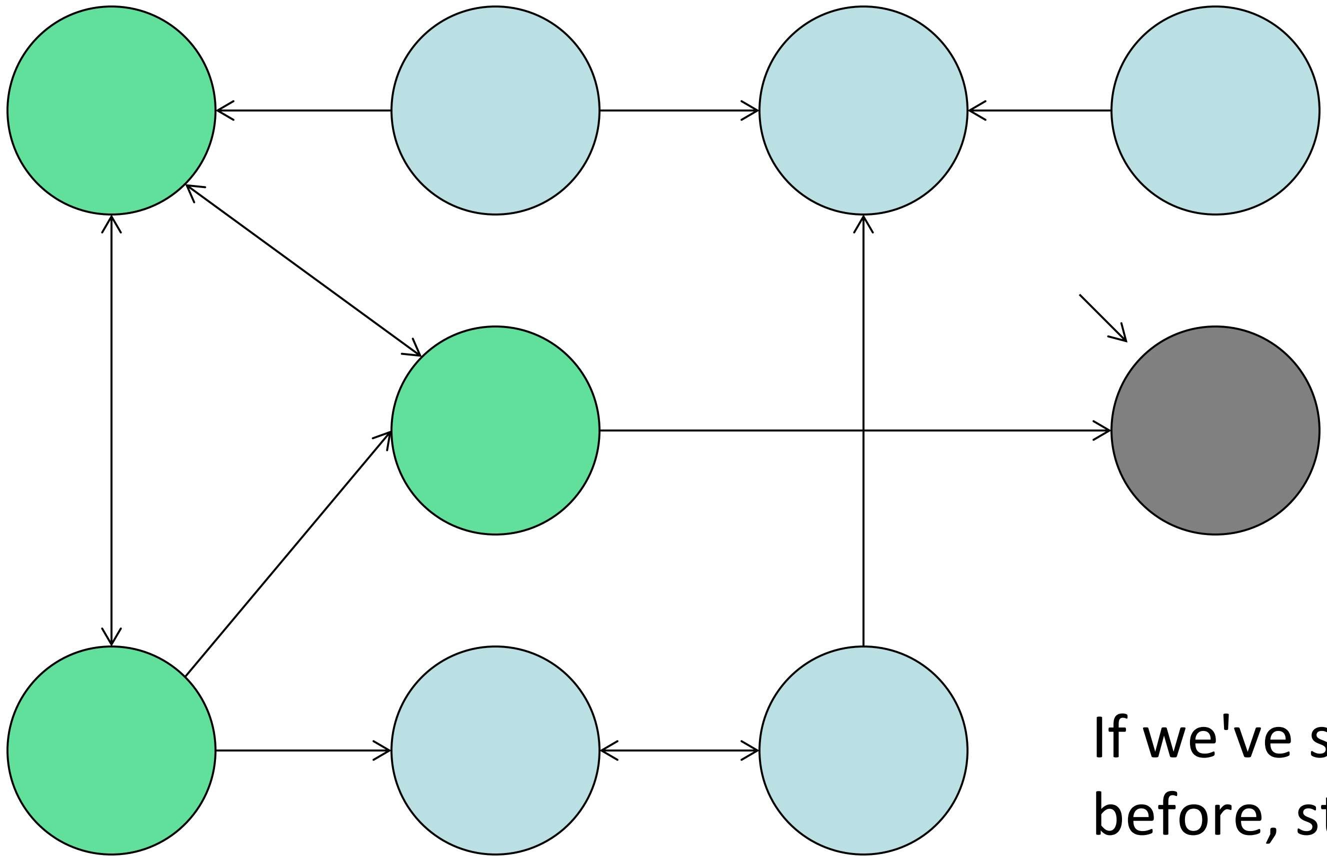
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

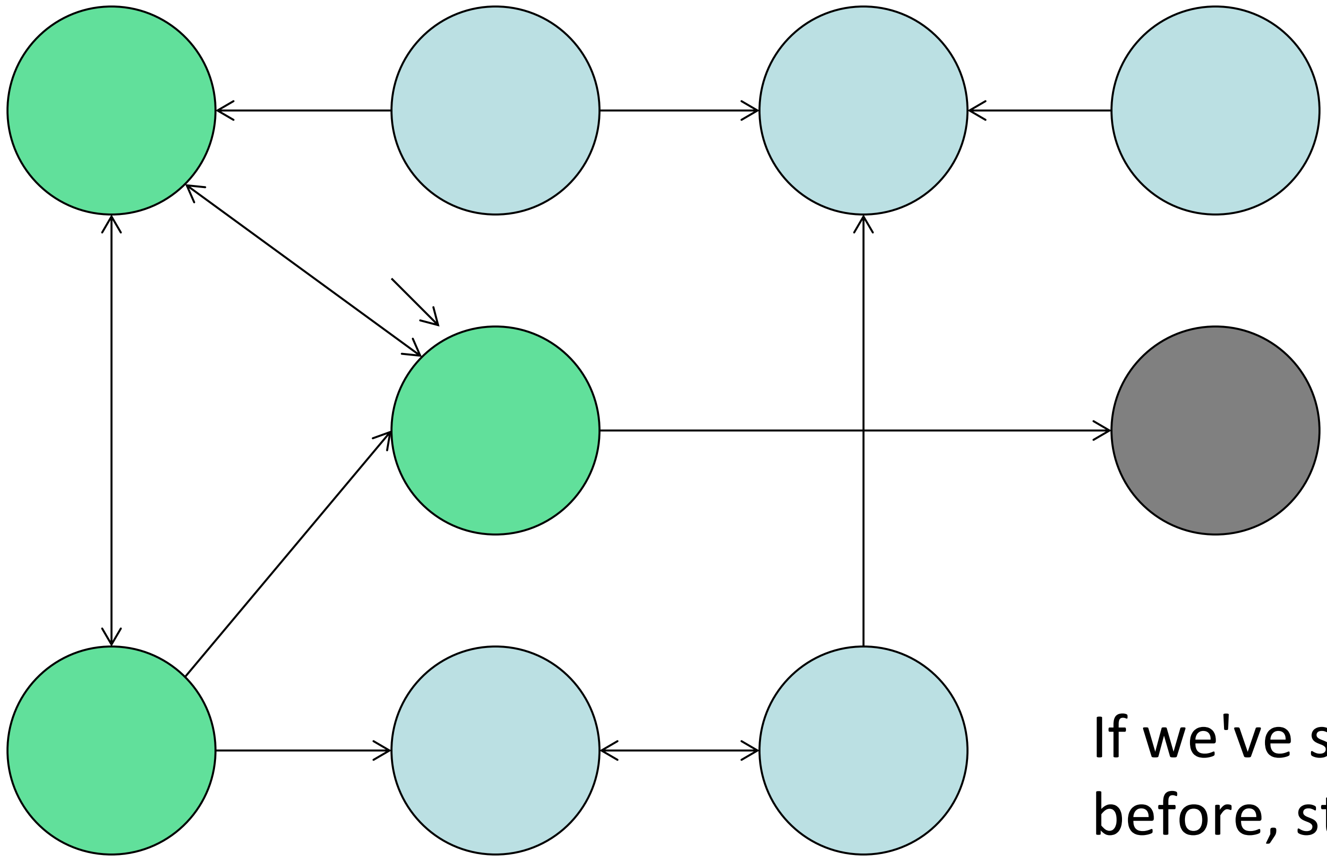
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

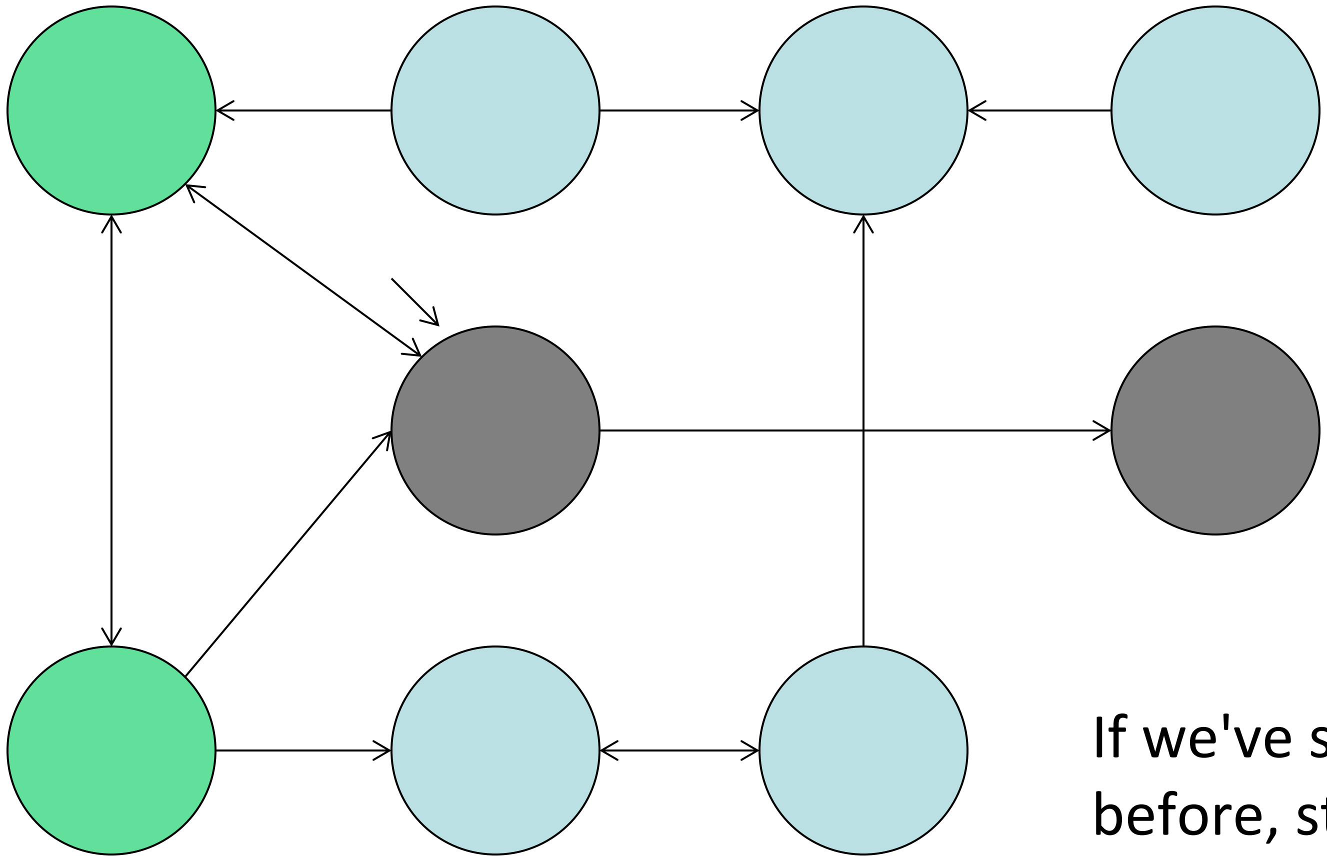
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

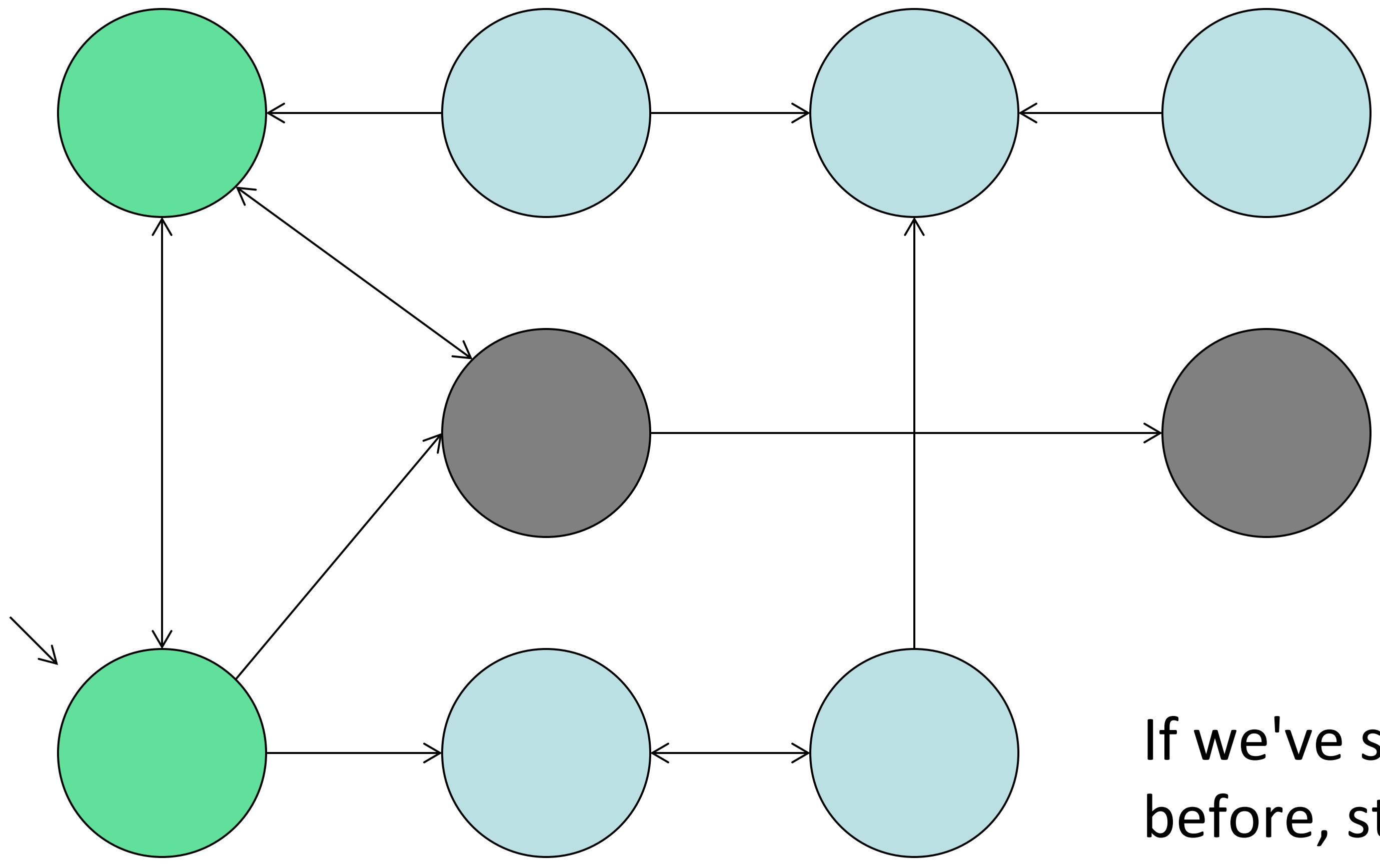
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

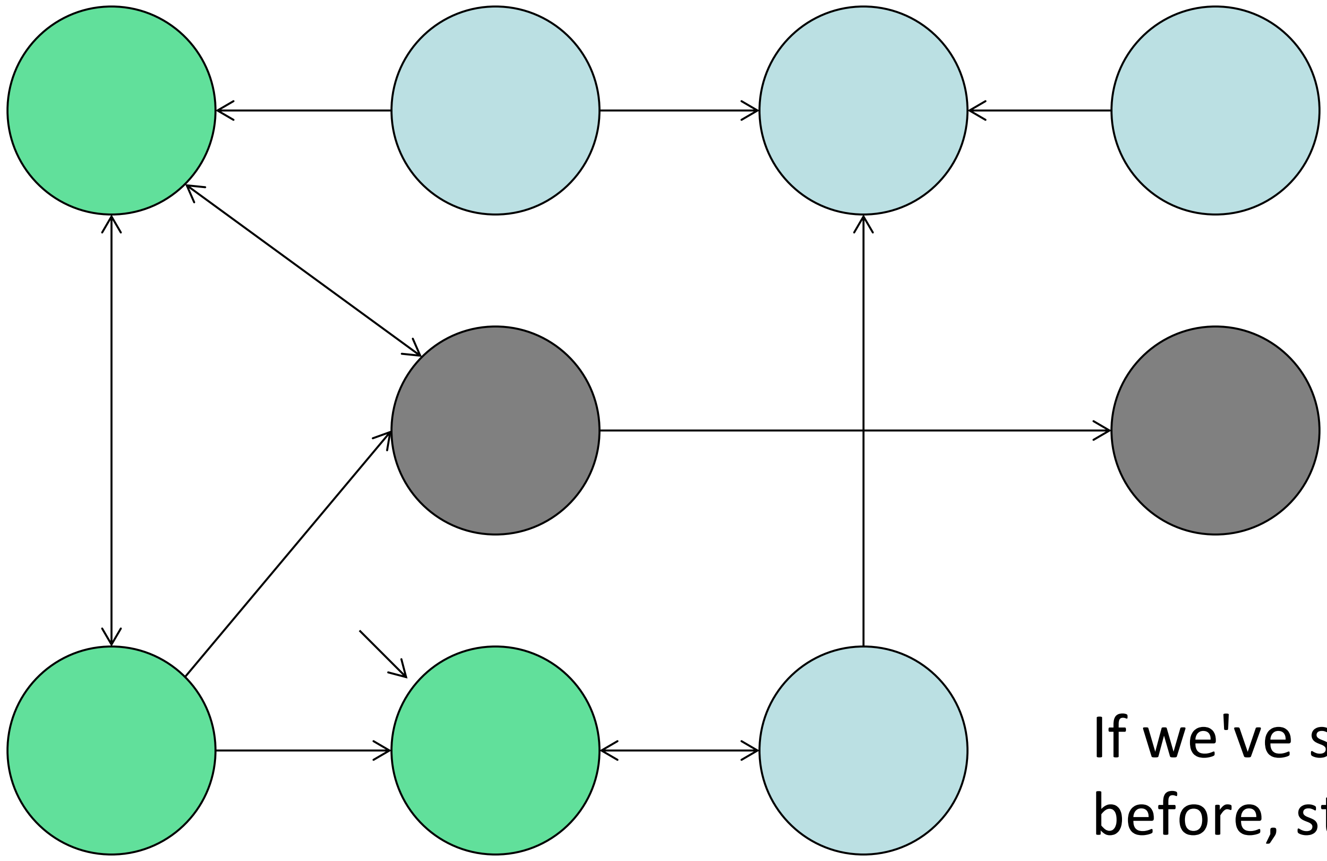
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

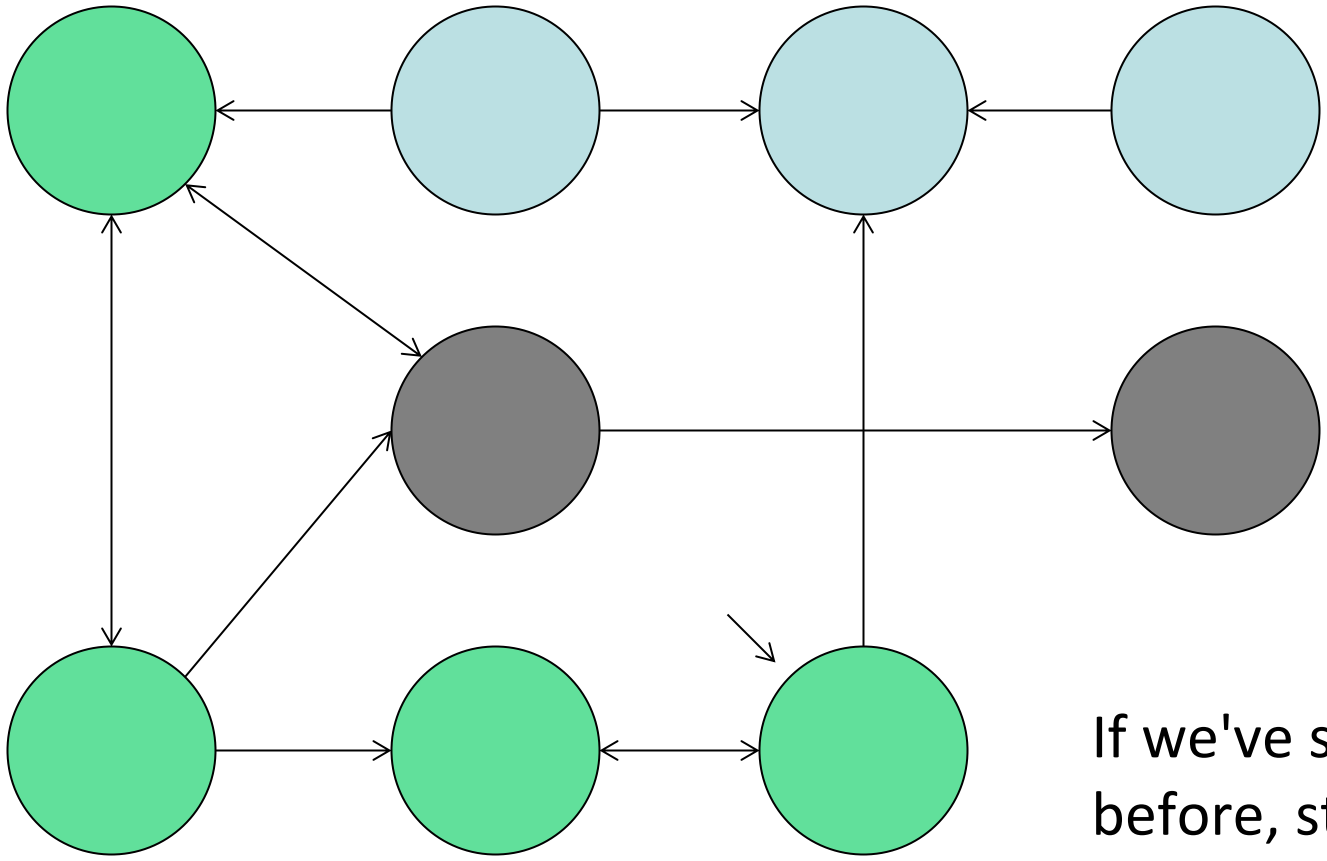
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

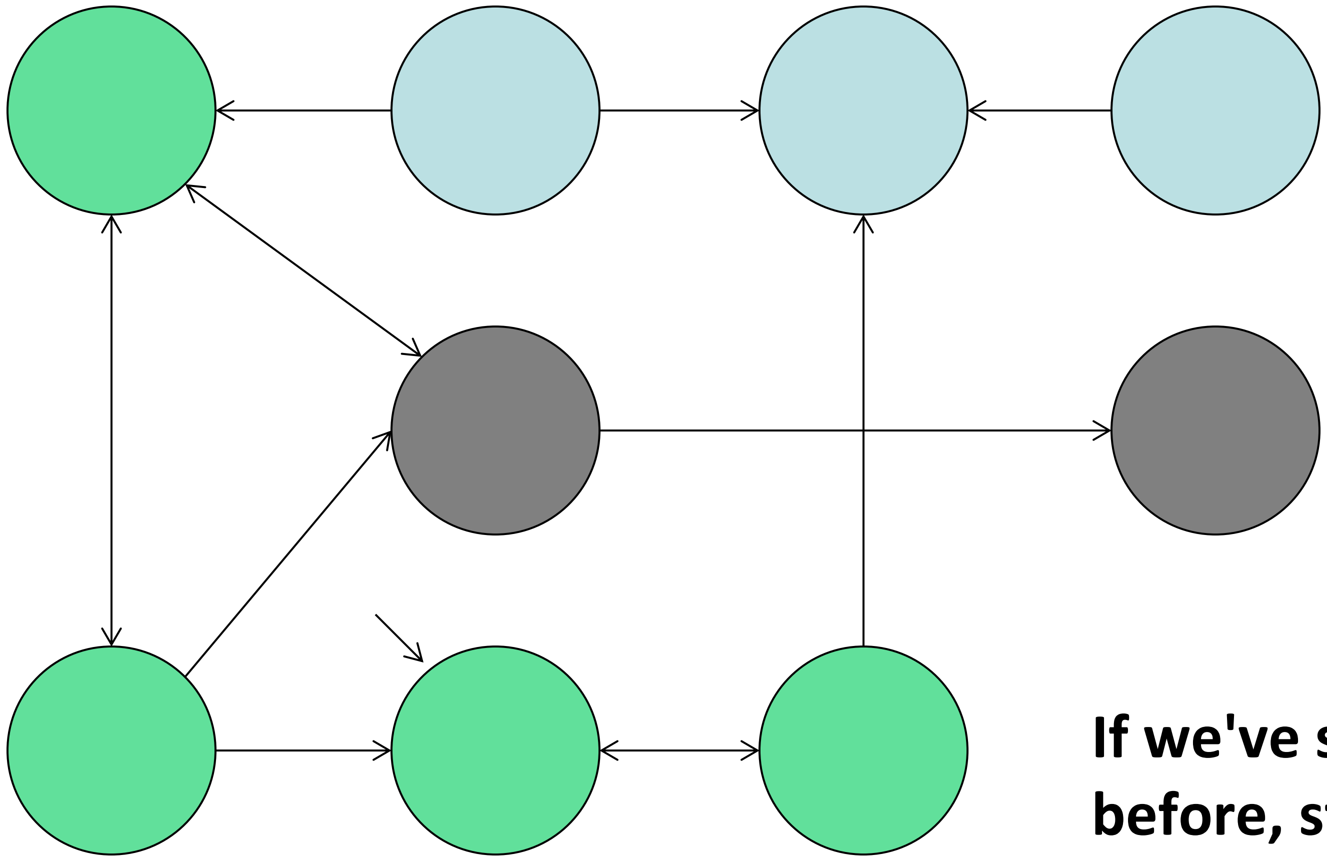
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

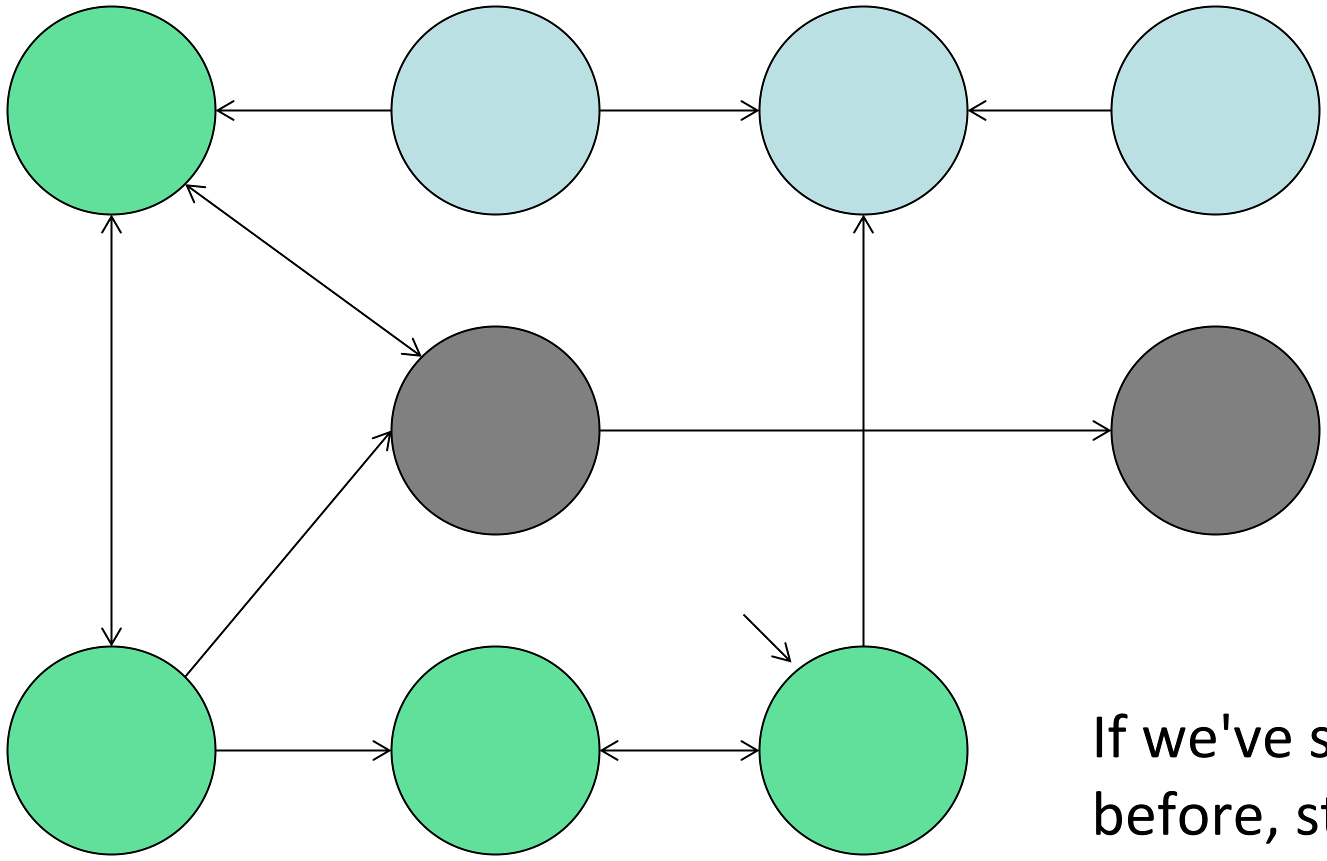
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

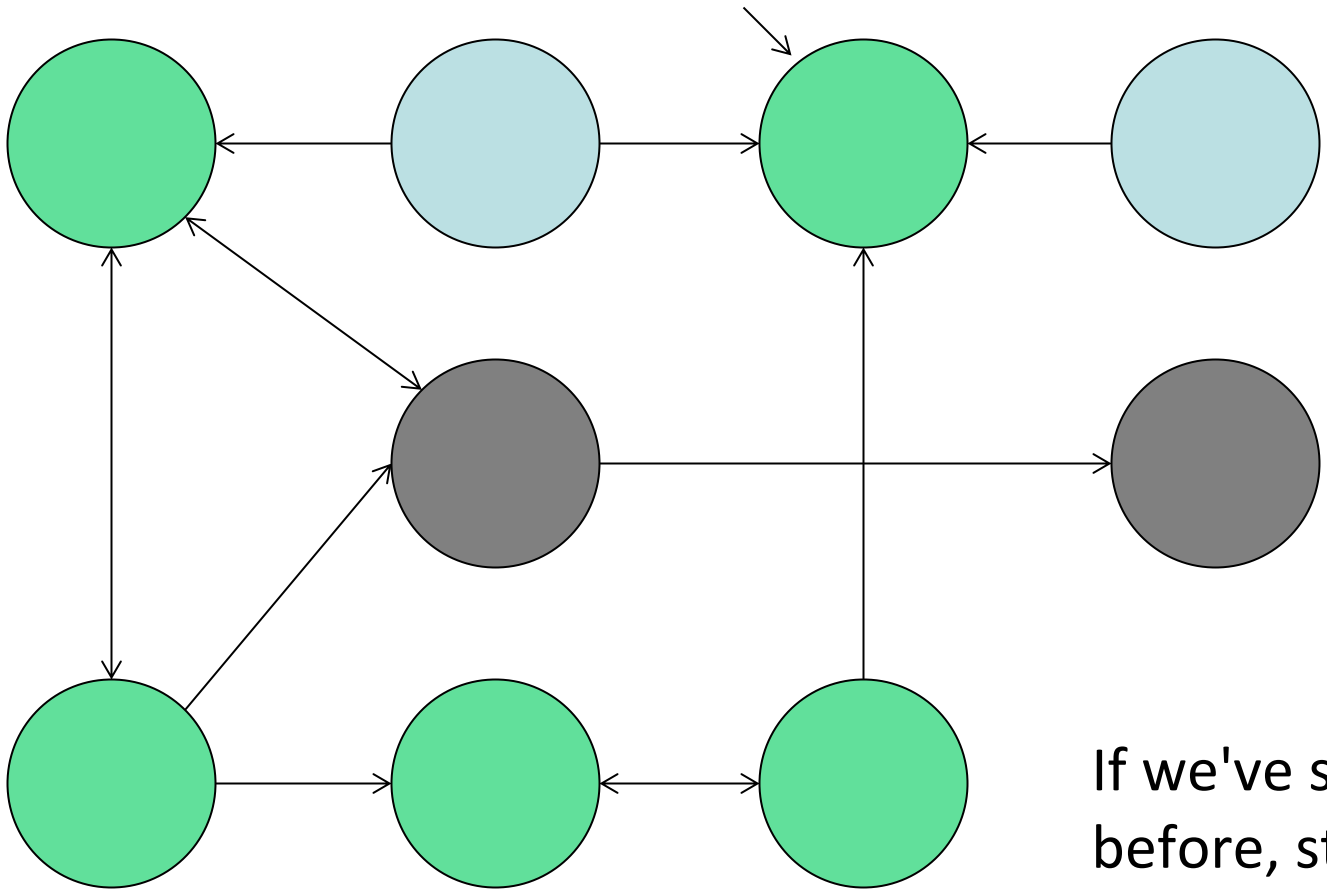
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

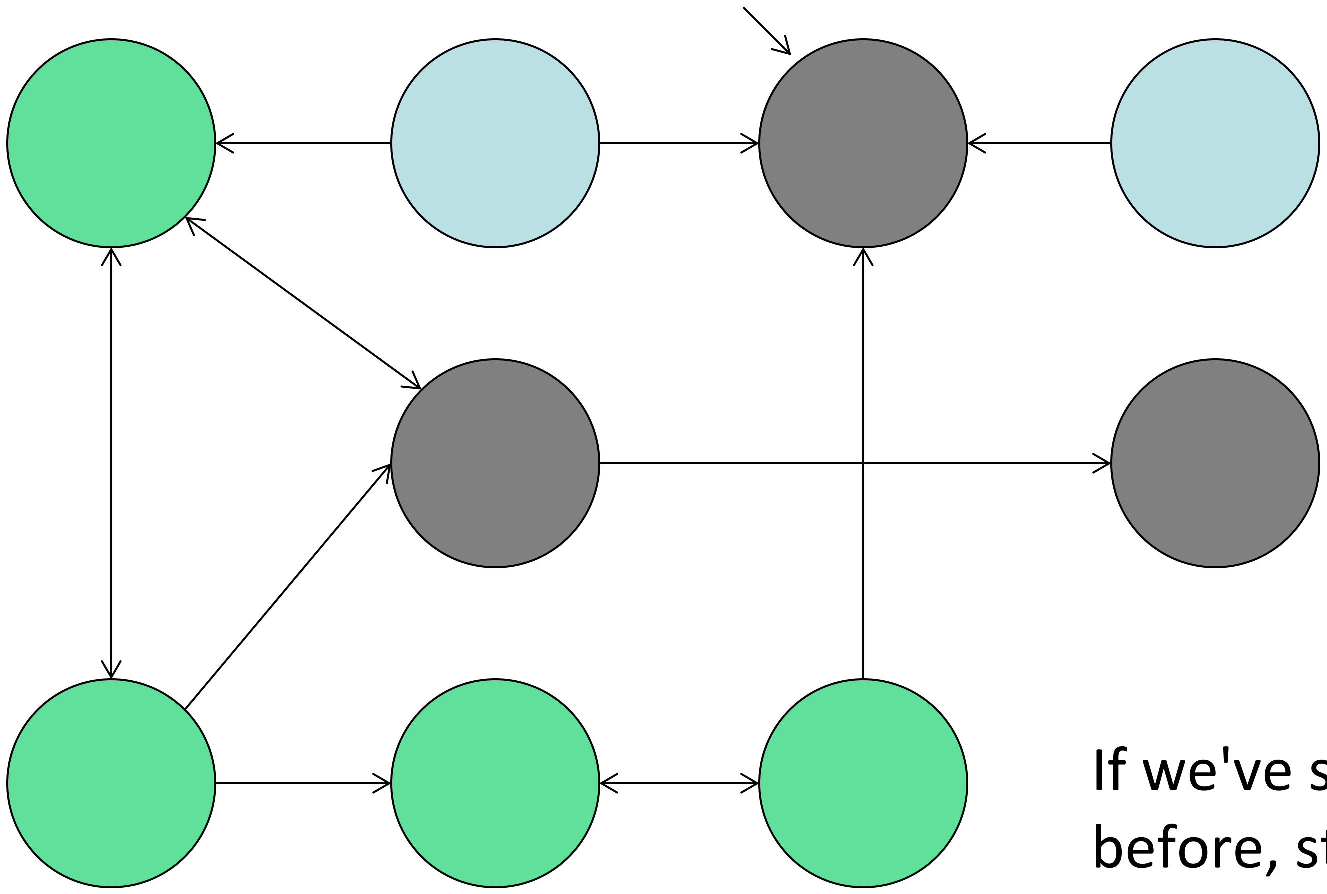
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

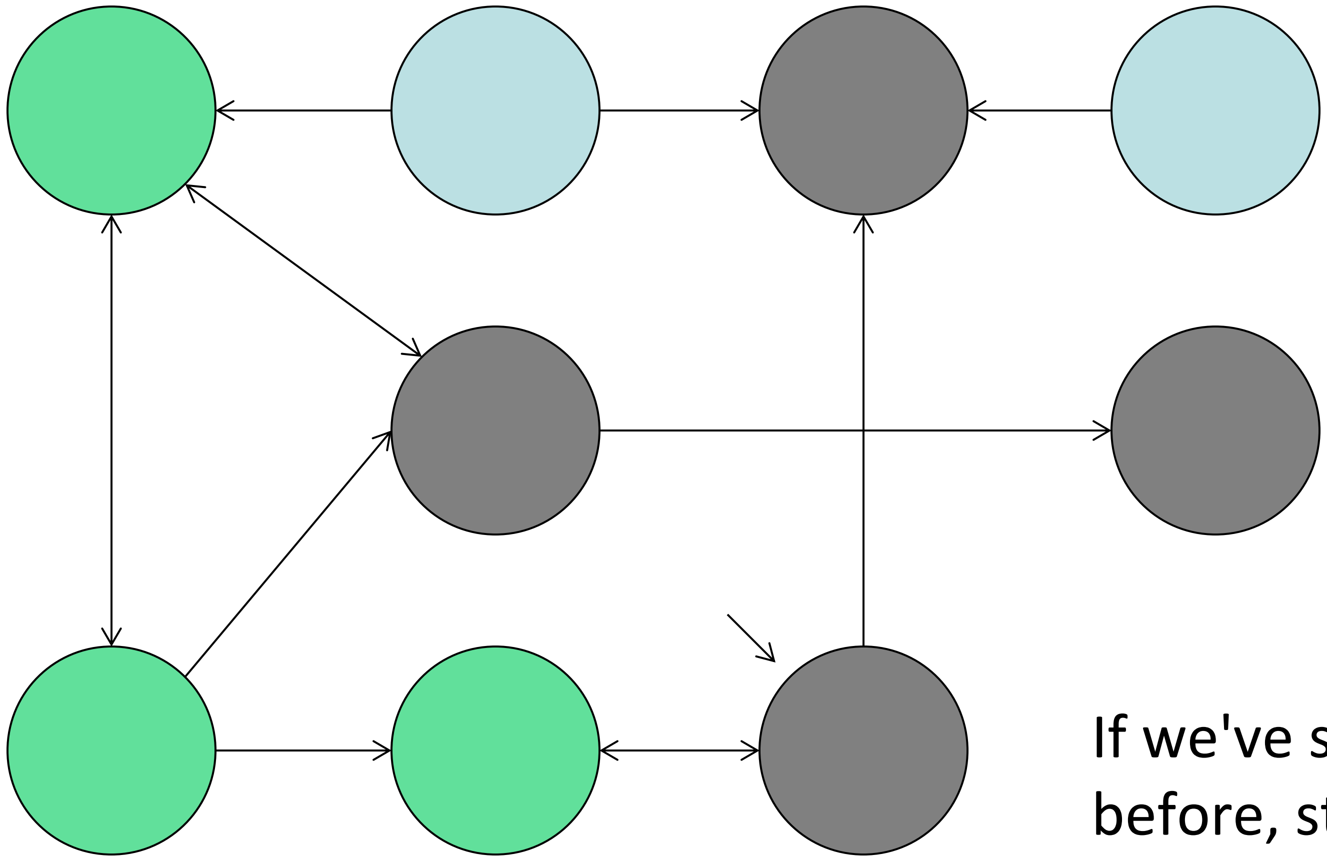
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

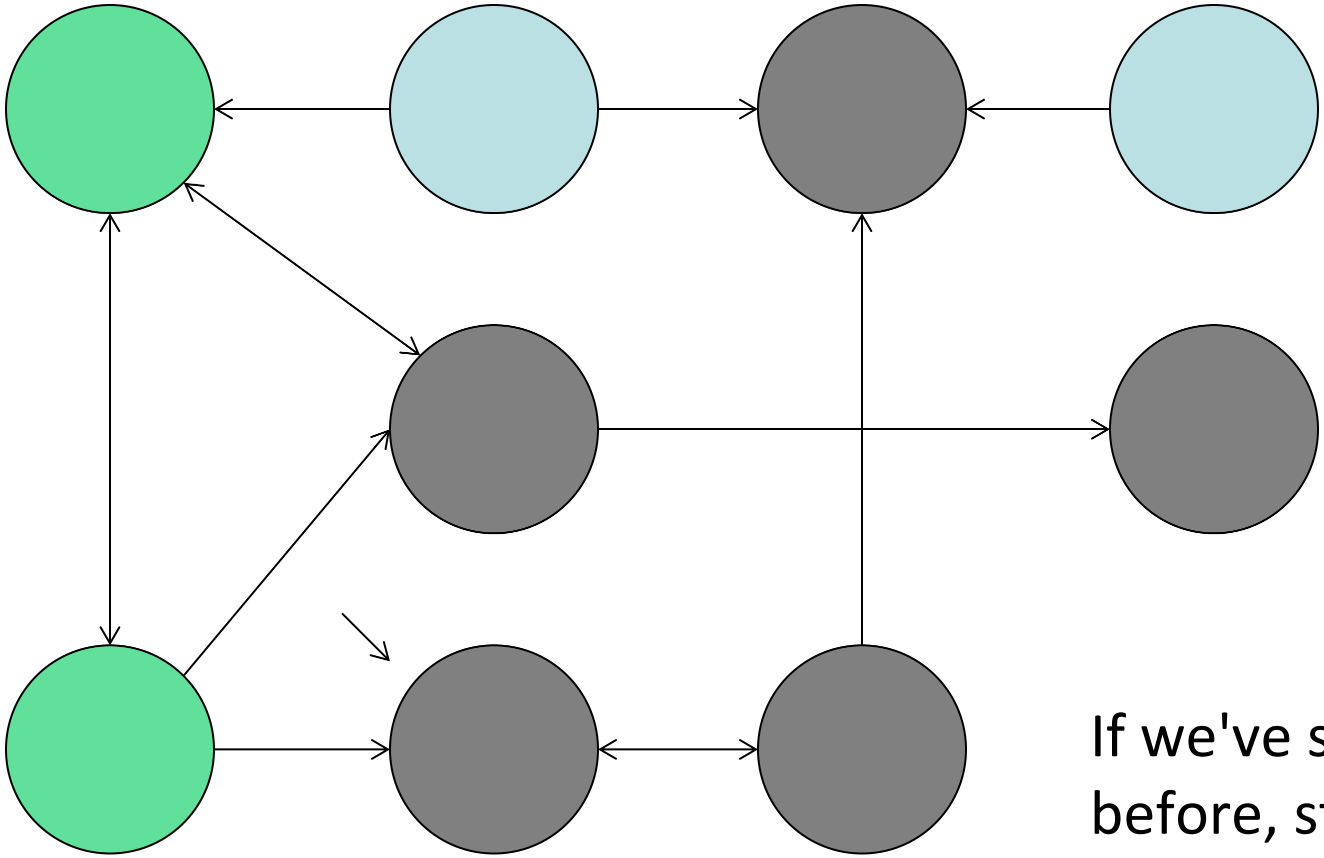
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

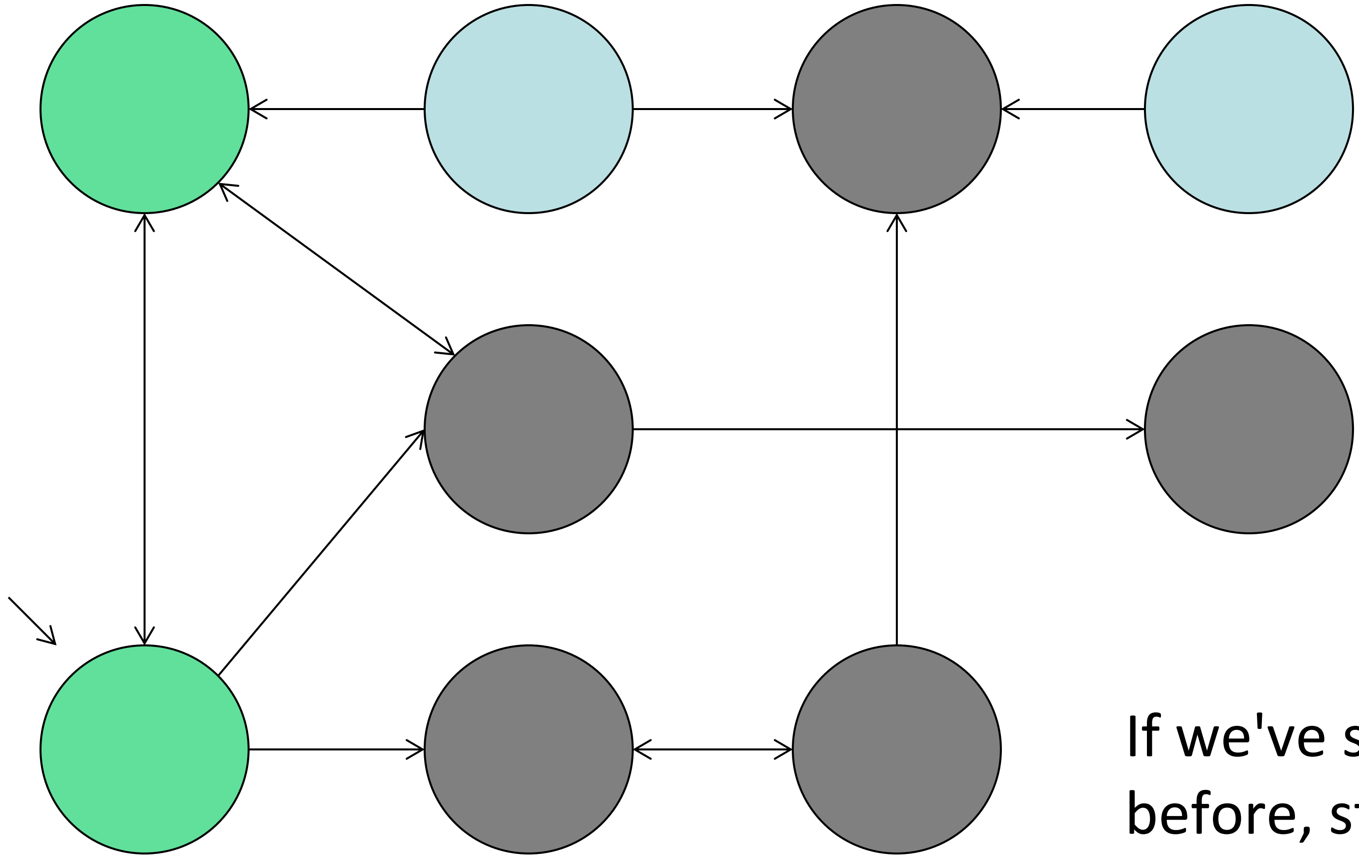
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

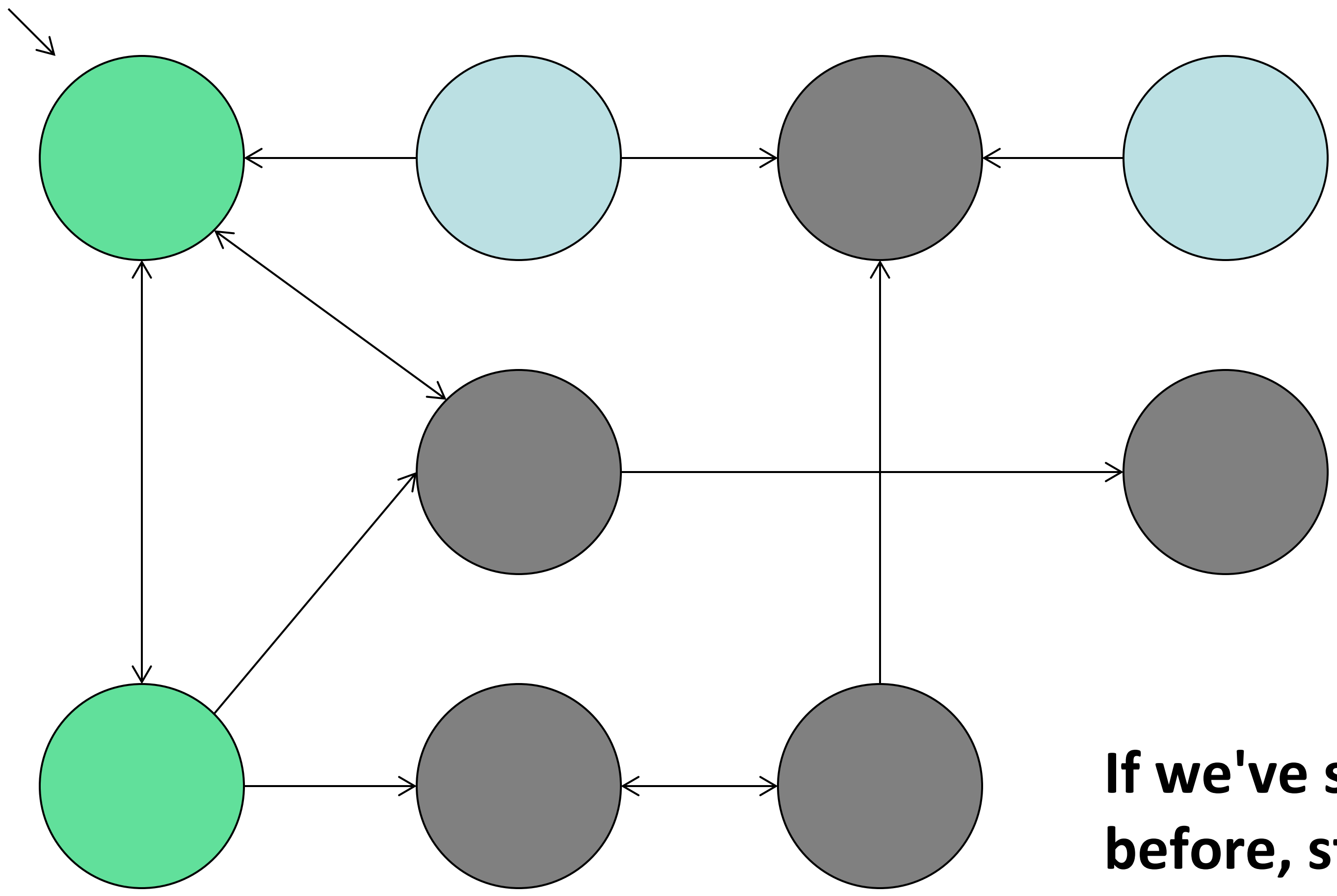
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

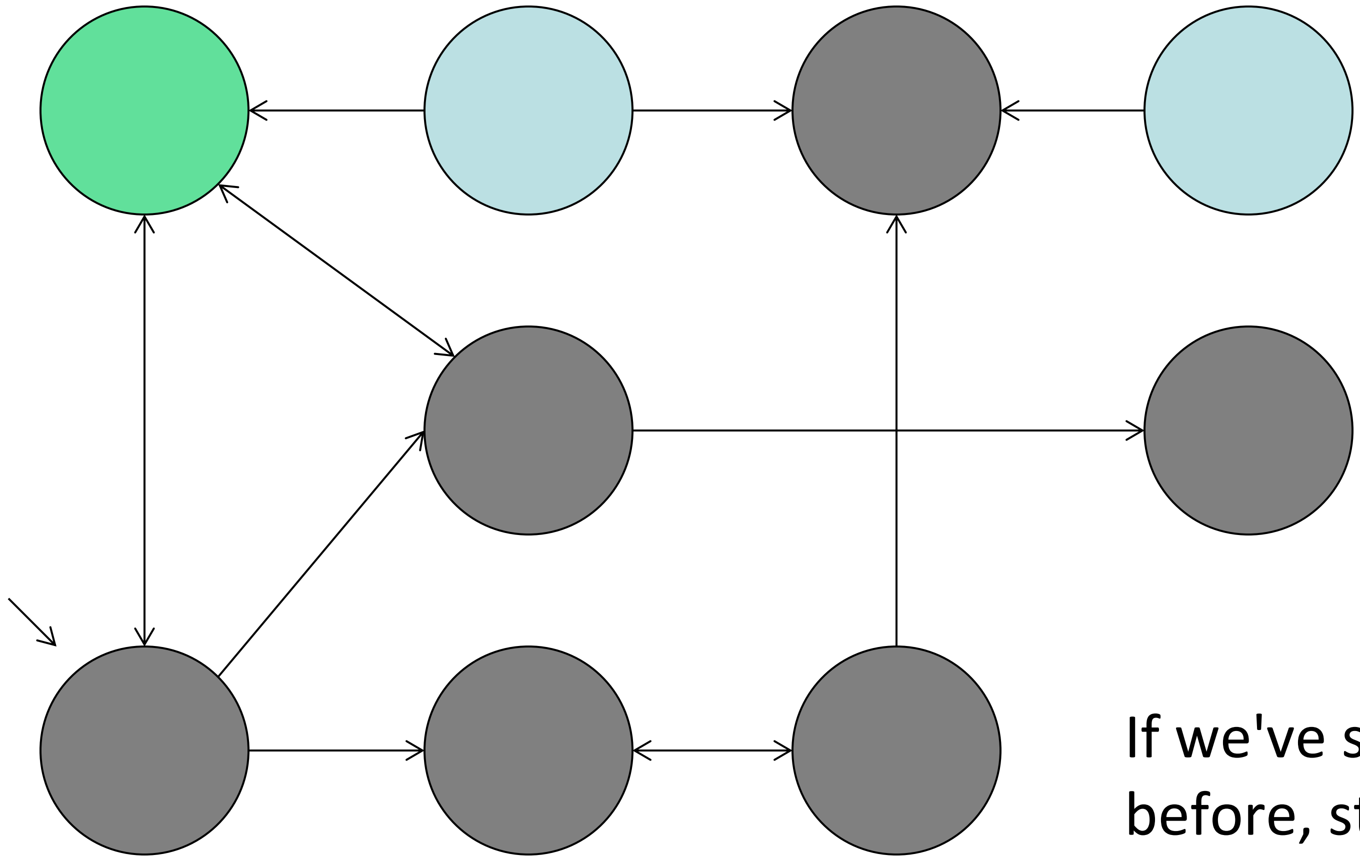
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

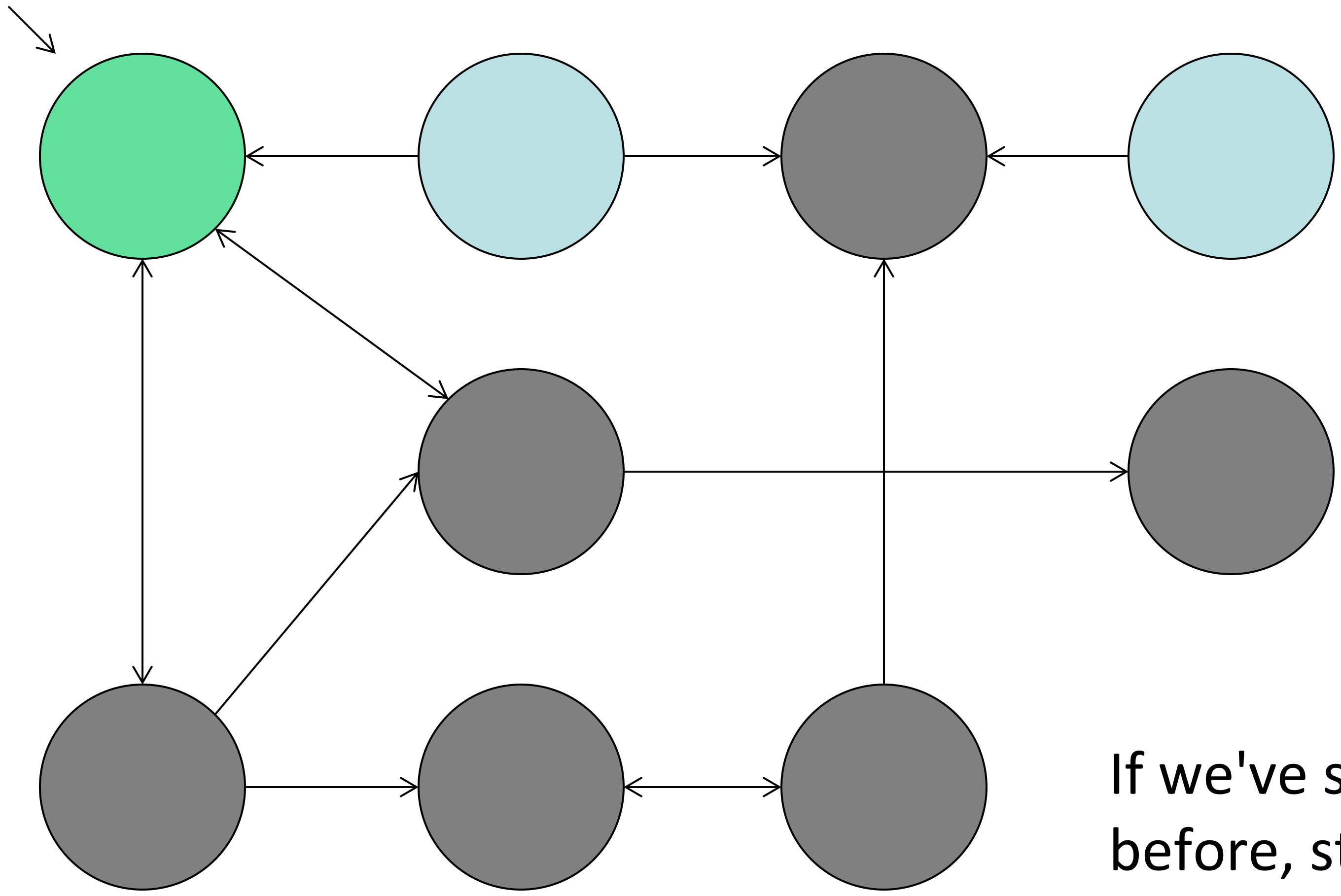
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

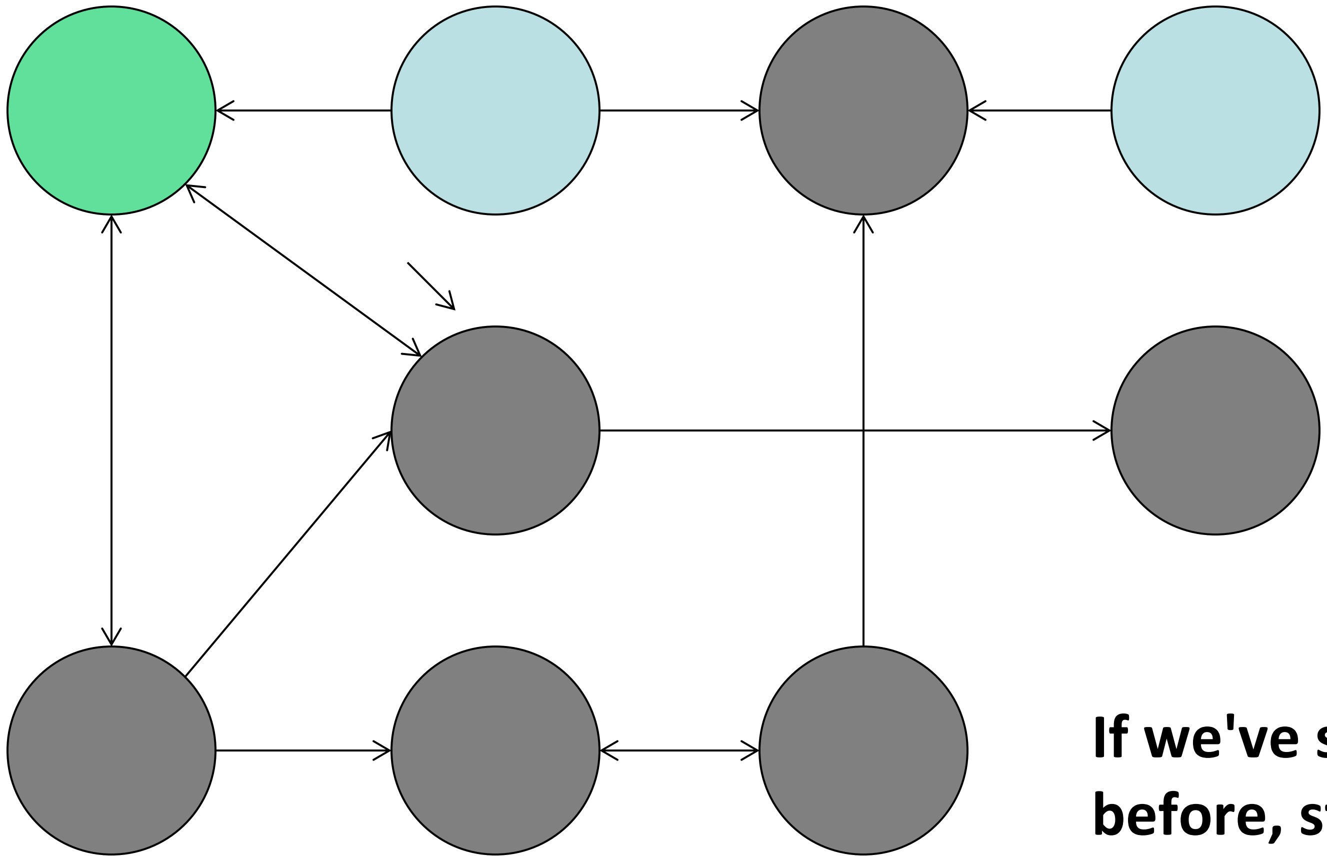
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

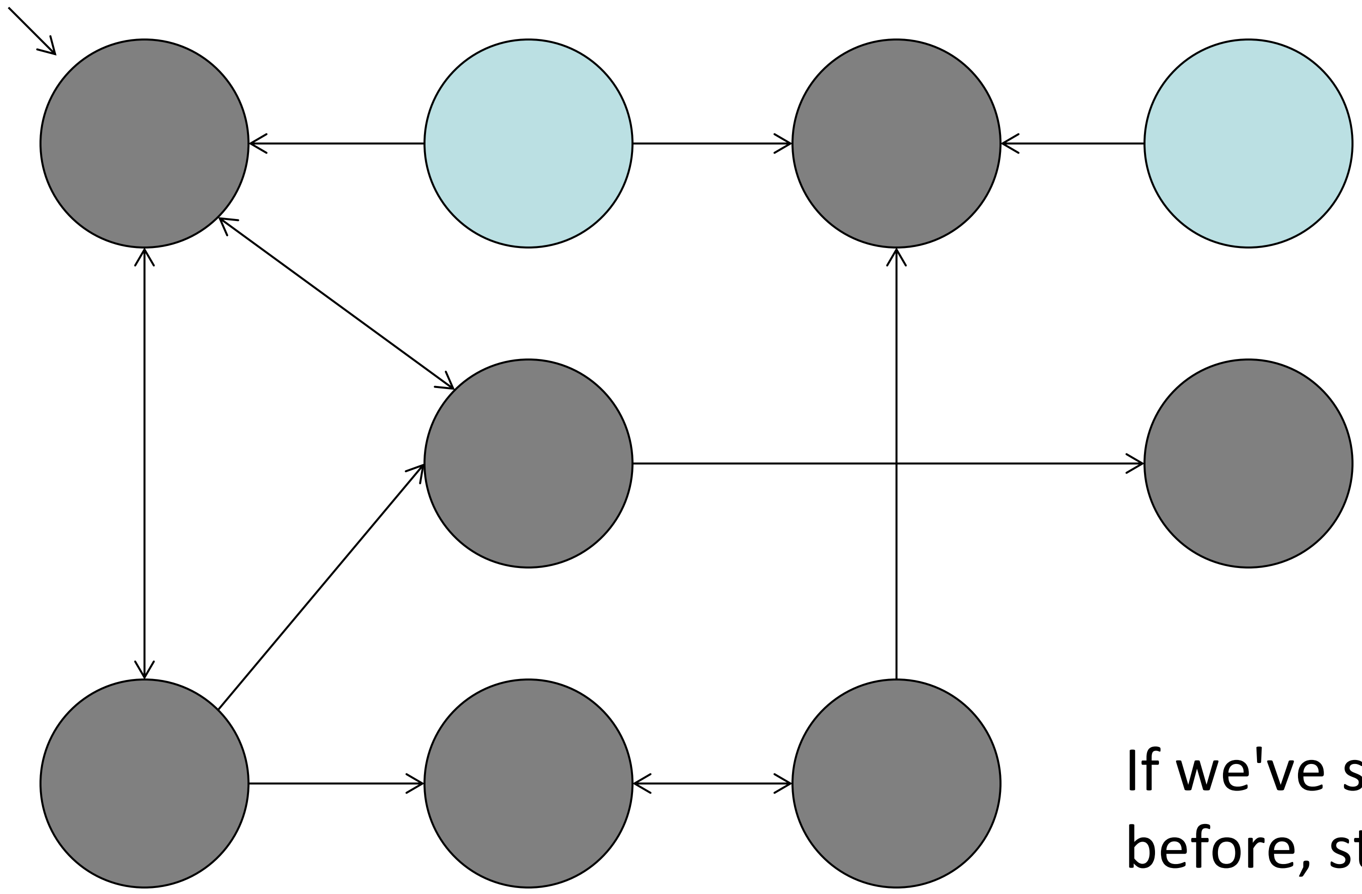
DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

DFS



If we've seen the node before, stop

Otherwise, visit all the unvisited nodes from this node

DFS Details

- In an *n*-node, *m*-edge graph, takes O(*m* + *n*) time with an adjacency list
  - Visit each edge once, visit each node at most once
- Pseudocode:

```
dfs from $v_1$:
    mark $v_1$ as seen.
    for each of $v_1$'s unvisited neighbors $n$:
      dfs($n$)
```

- How could we modify the pseudocode to look for a specific path?
  - Recursive Backtracking
  - Look at maze example from earlier in the course

Finding *Shortest* Paths

- We can find paths between two nodes, but how can we find the **shortest** path?
  - Fewest number of steps to complete a task?
  - Least amount of edits between two words?
- When have we solved this problem before?

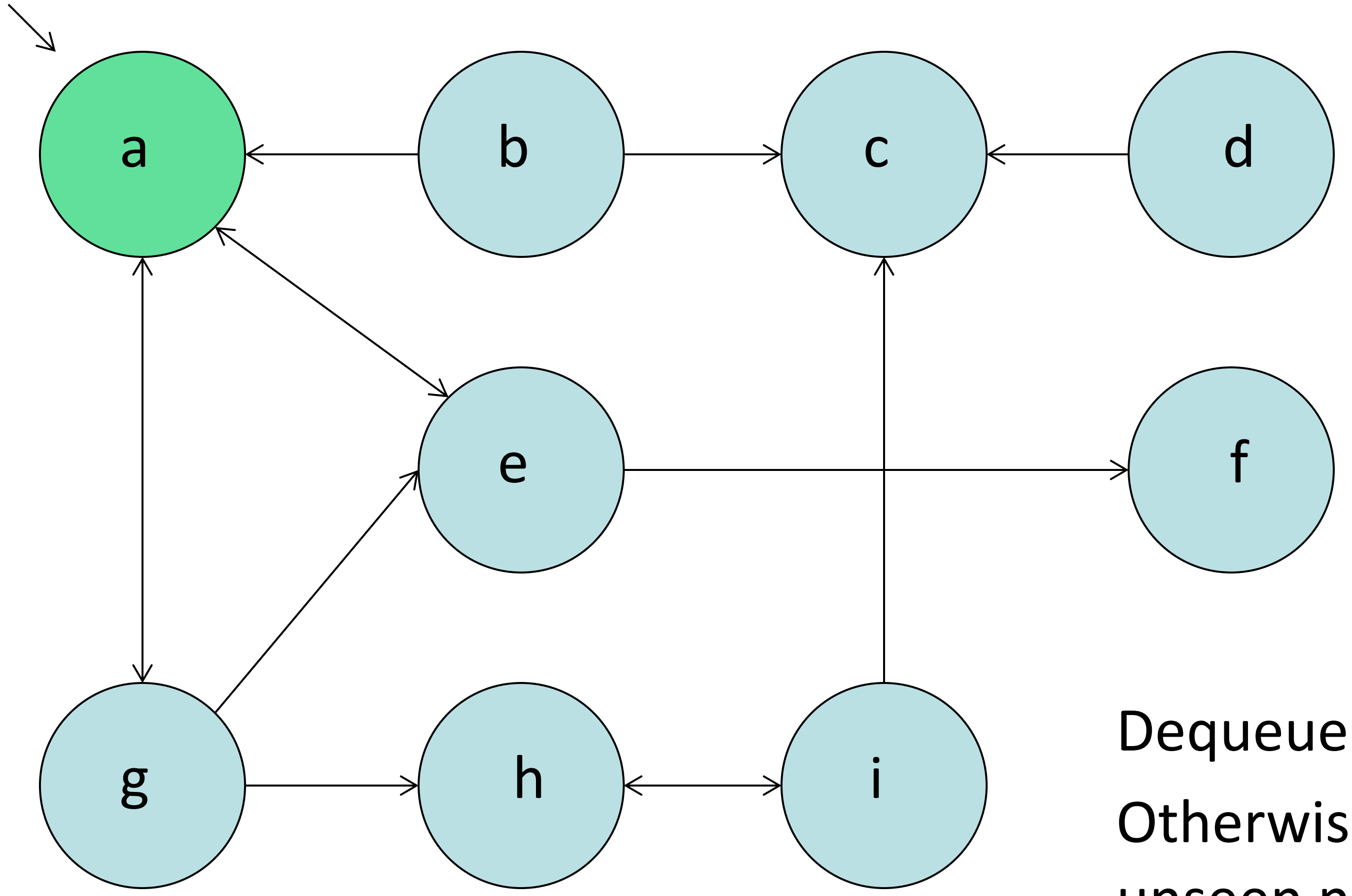
Breadth-First Search (BFS)

- Idea: processing a node involves knowing we need to visit all its neighbors (just like DFS)
- Need to keep a TODO list of nodes to process
- Which node from our TODO list should we process first if we want the shortest path?
  - The first one we saw?
  - The last one we saw?
  - A random node?

Breadth-First Search (BFS)

- Keep a Queue of nodes as our TODO list
- Idea: dequeue a node, enqueue all its neighbors
- Still will return the same nodes as reachable, just might have shorter paths

BFS

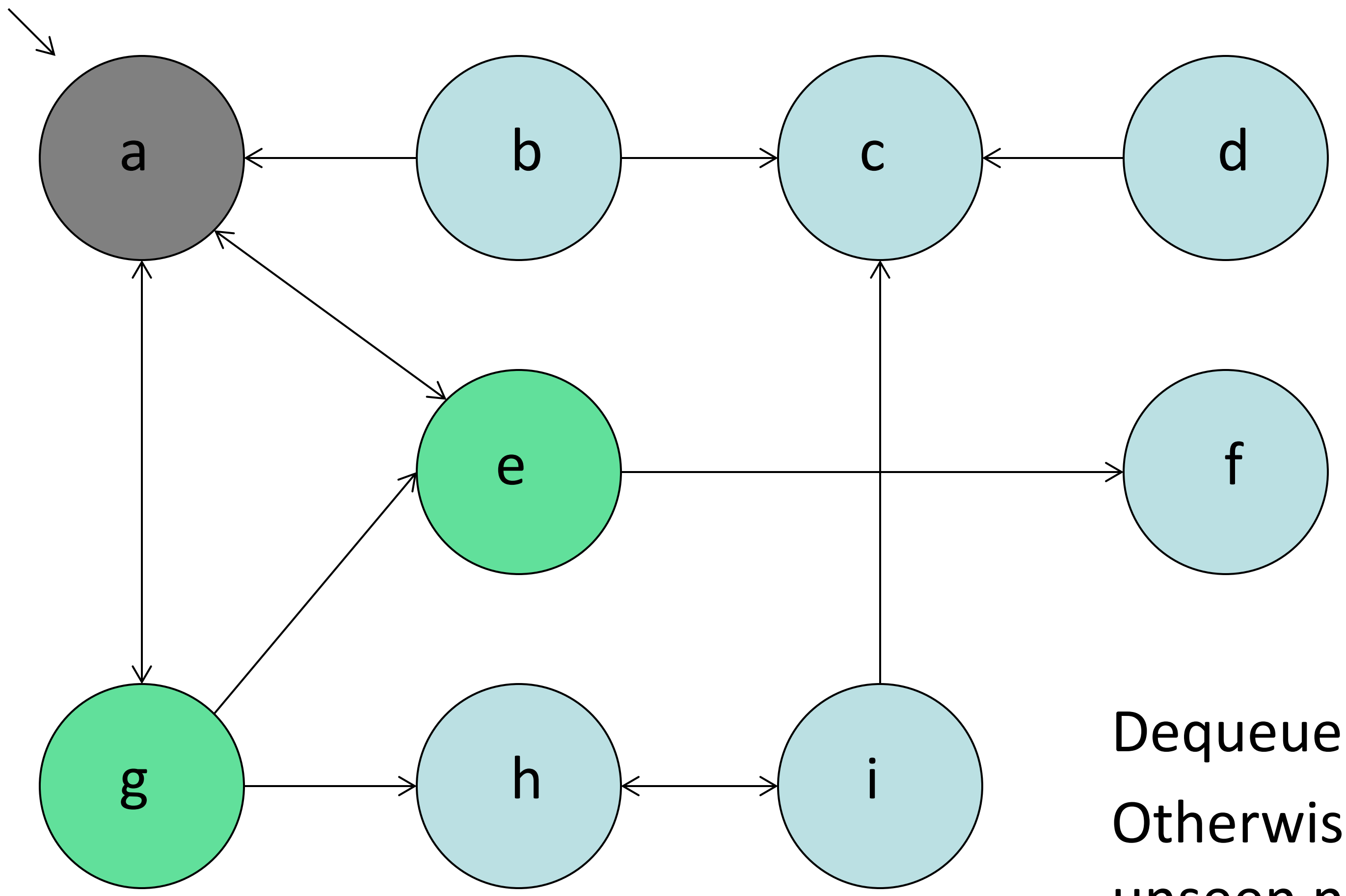


Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: a

BFS

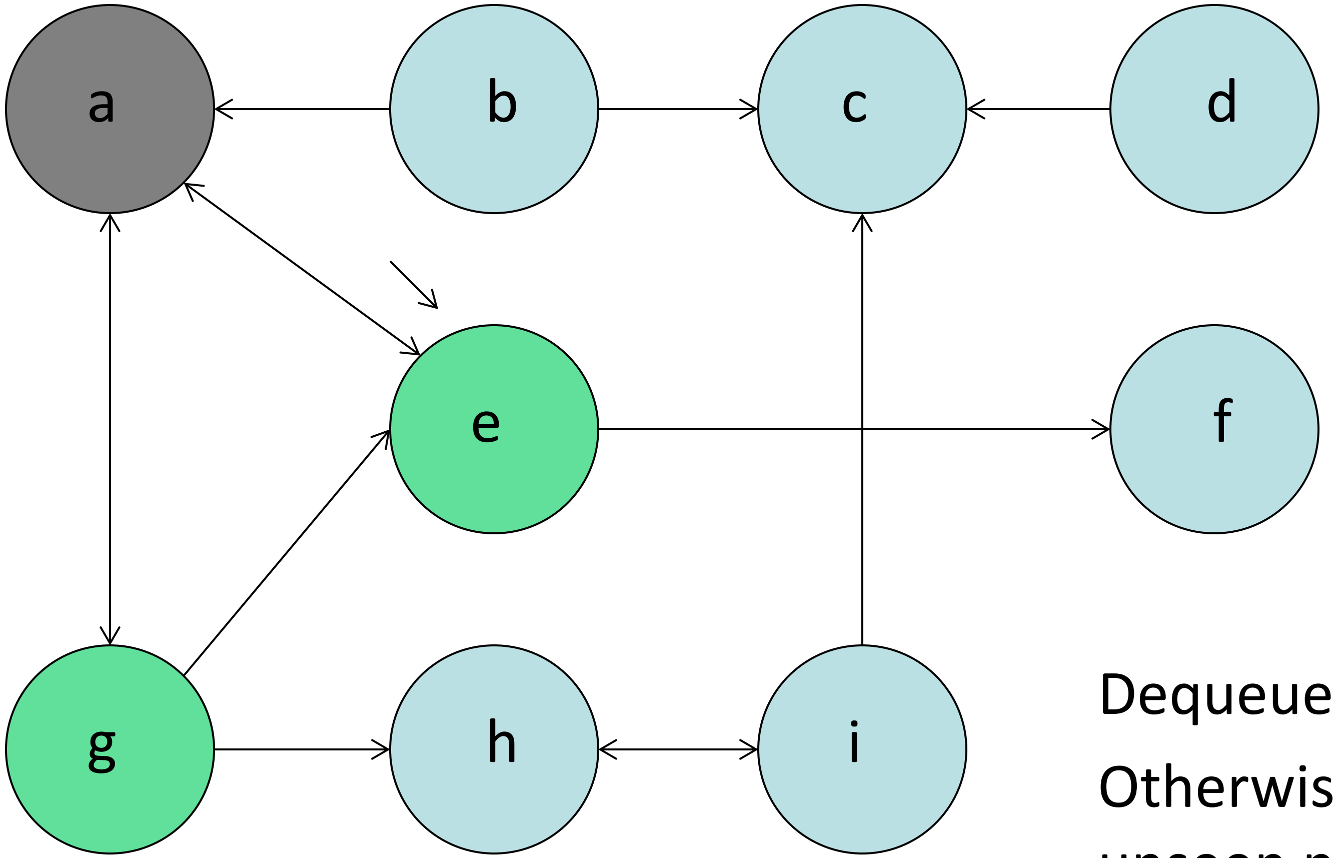


Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: e, g

BFS

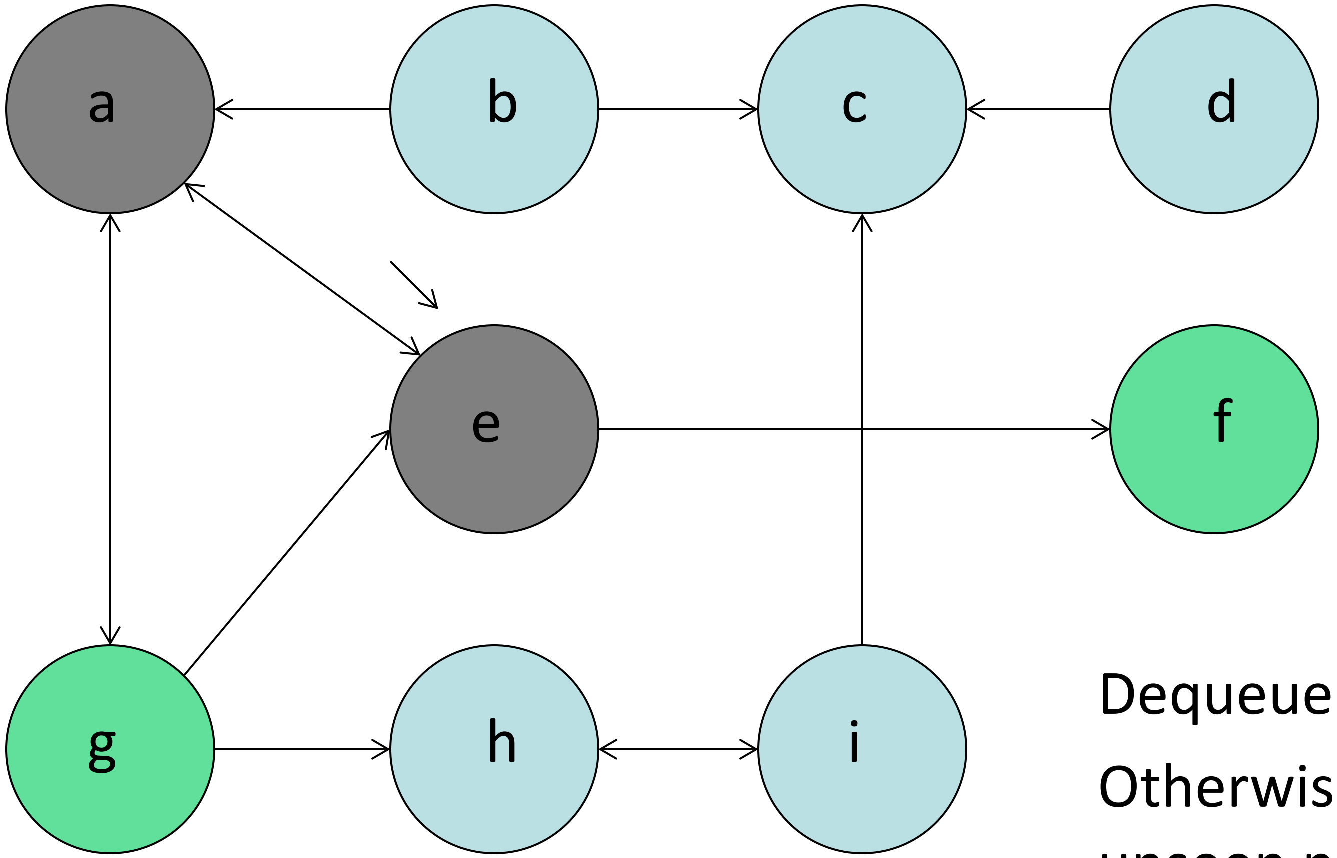


Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: e, g

BFS

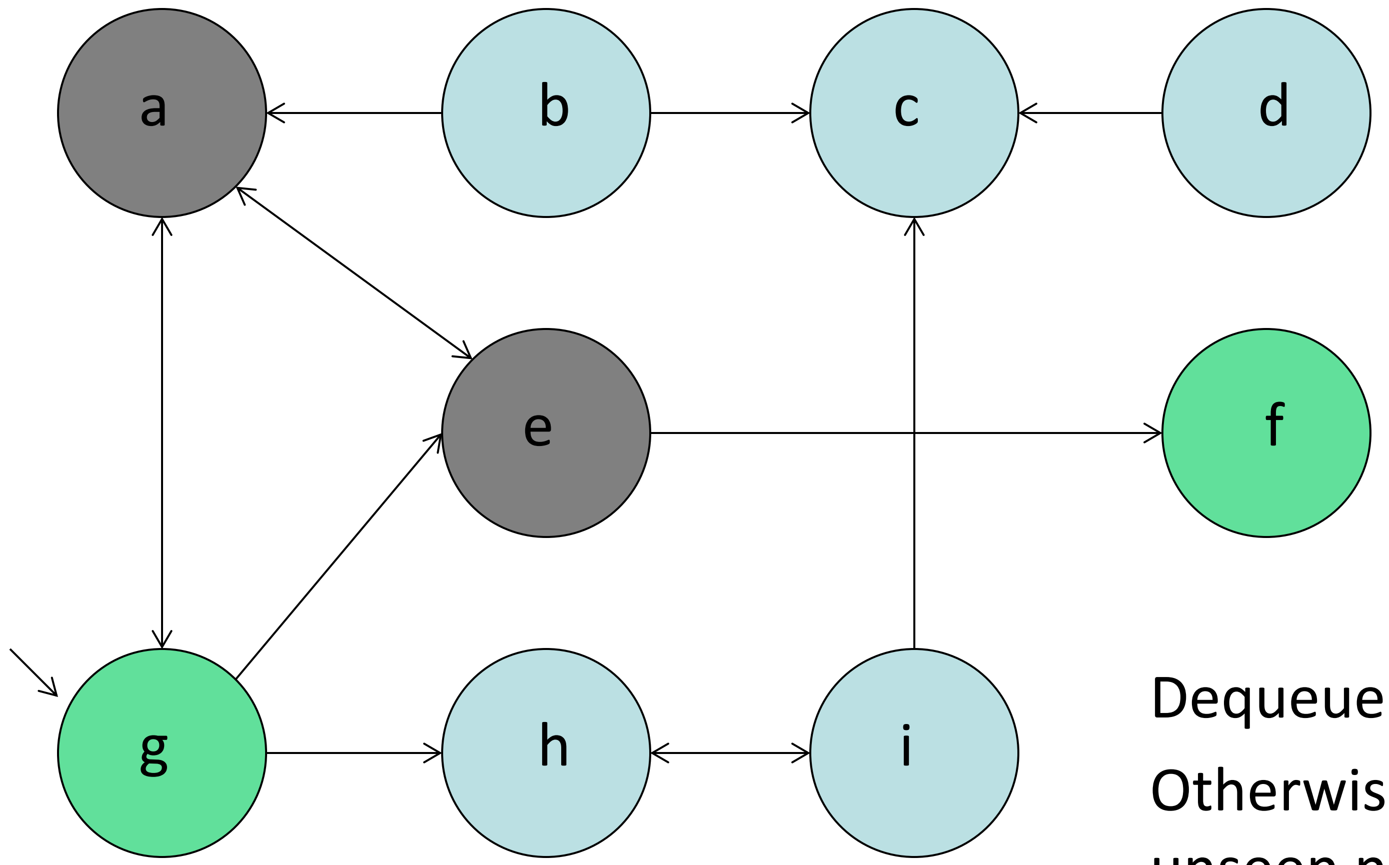


Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: g, f

BFS

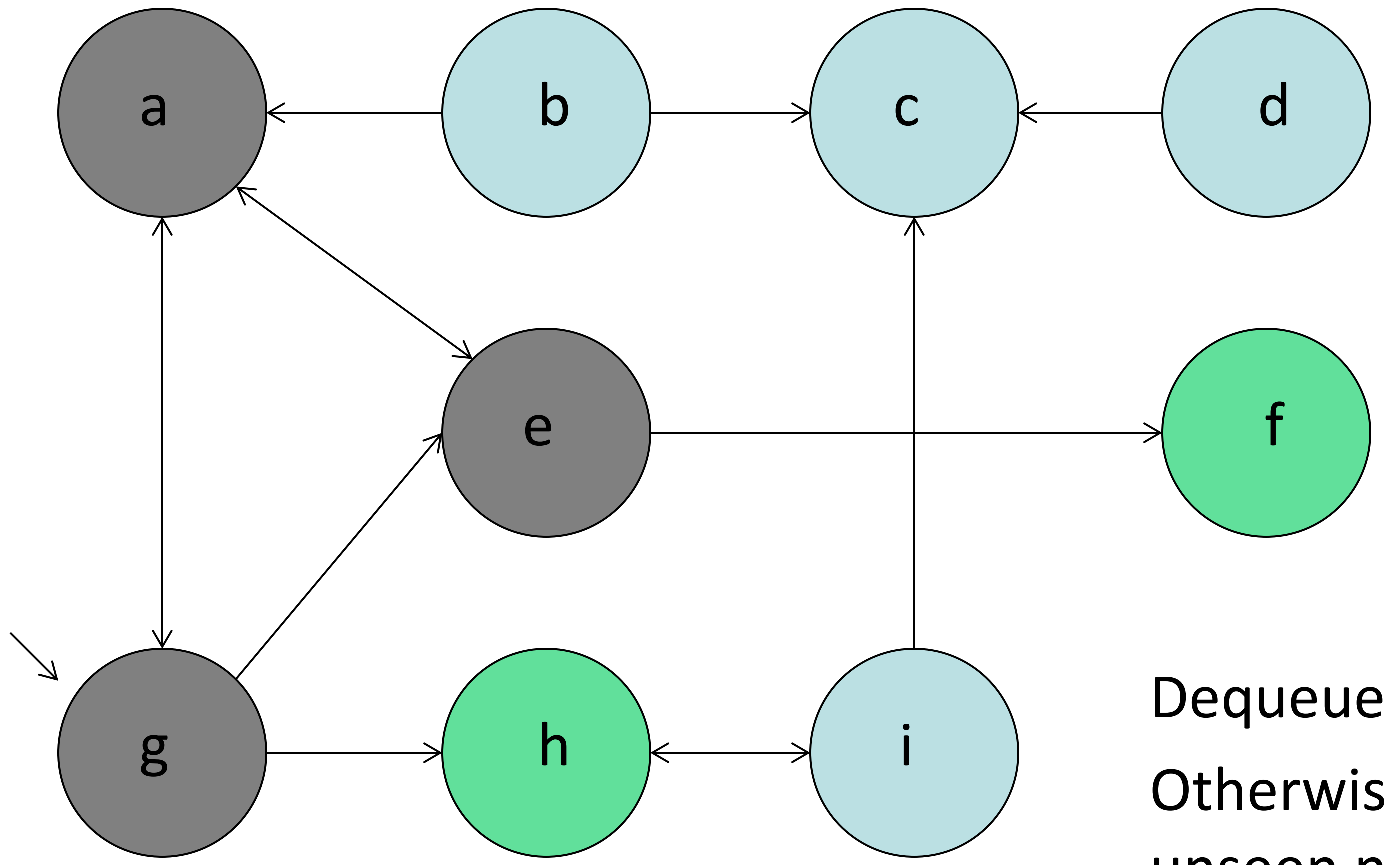


Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: g, f

BFS

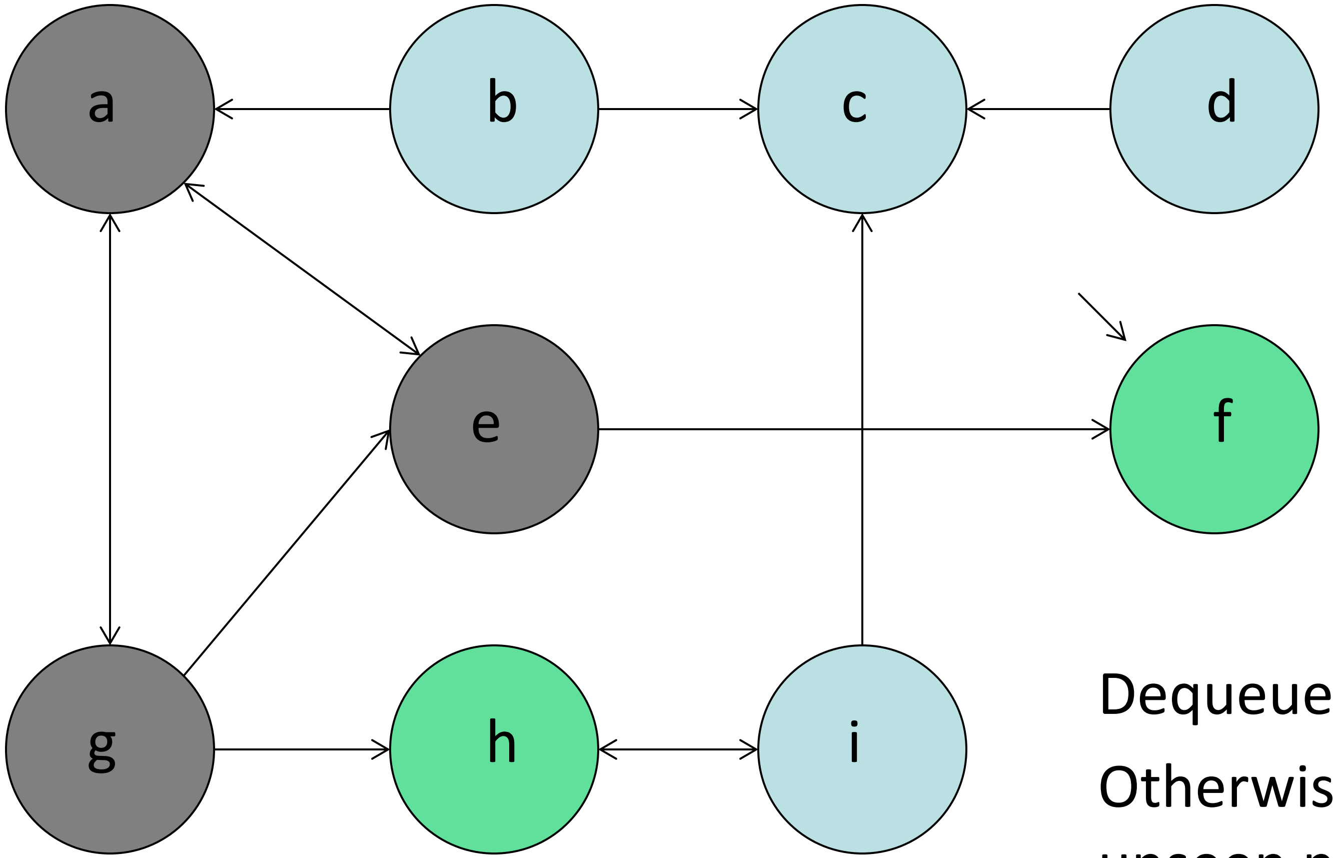


Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: f, h

BFS

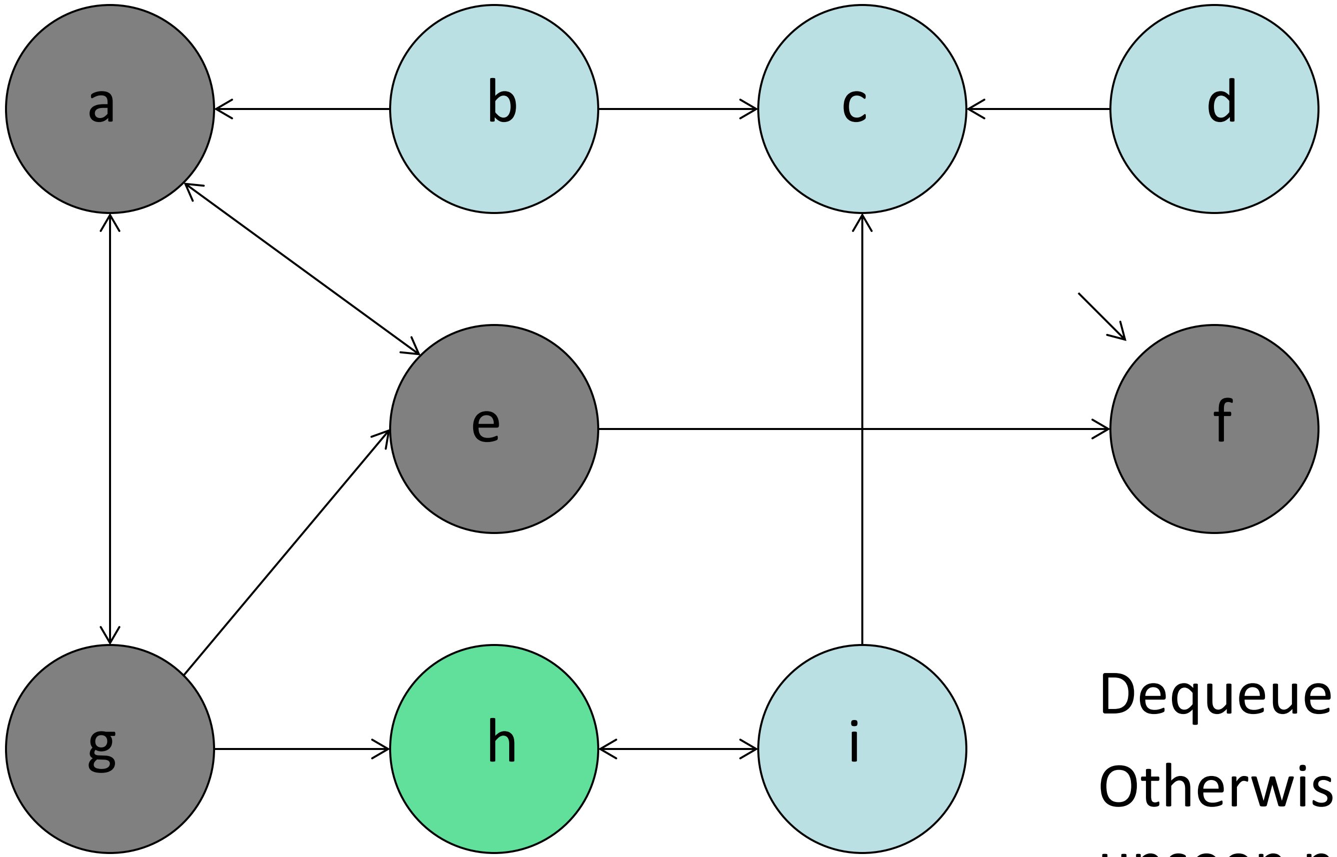


Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: f, h

BFS

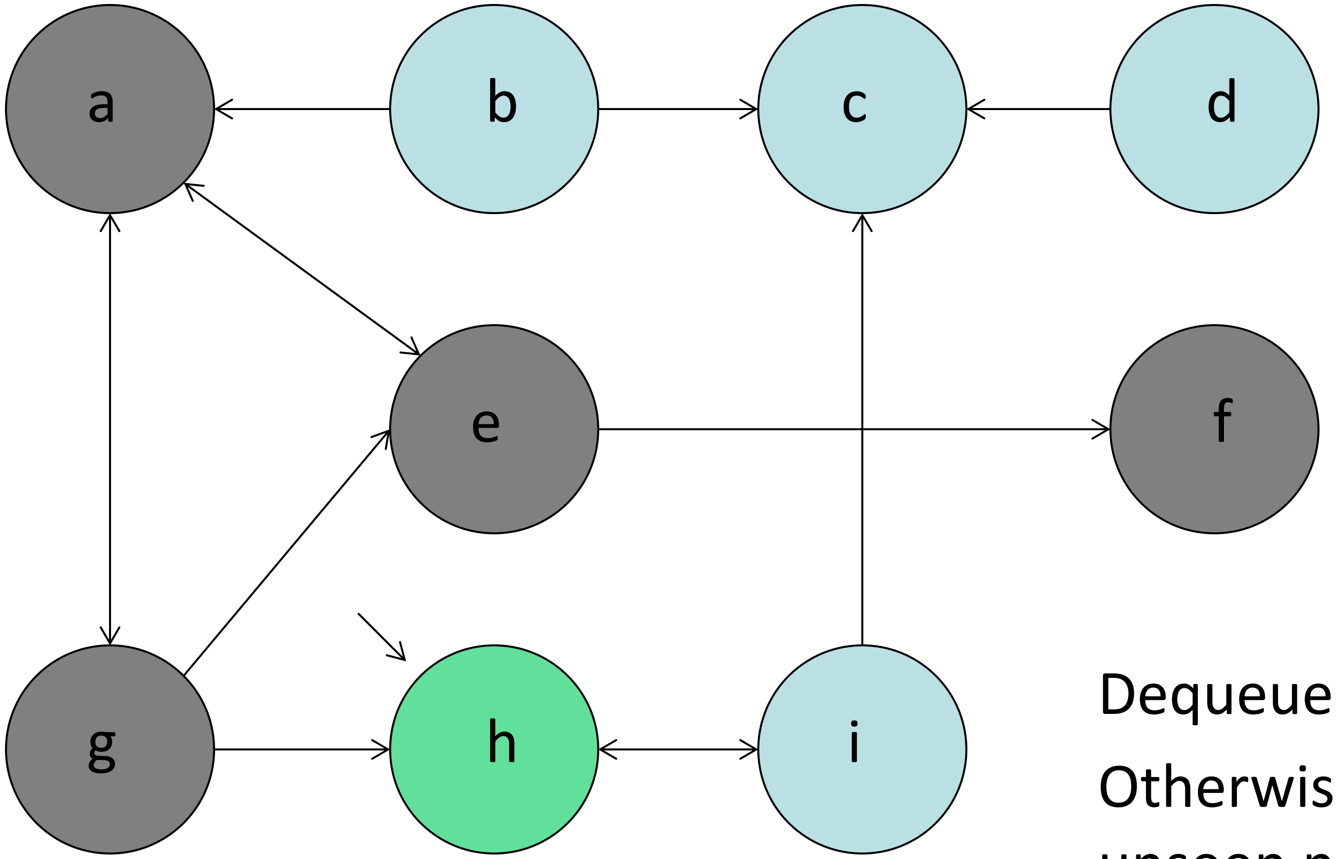


Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: h

BFS

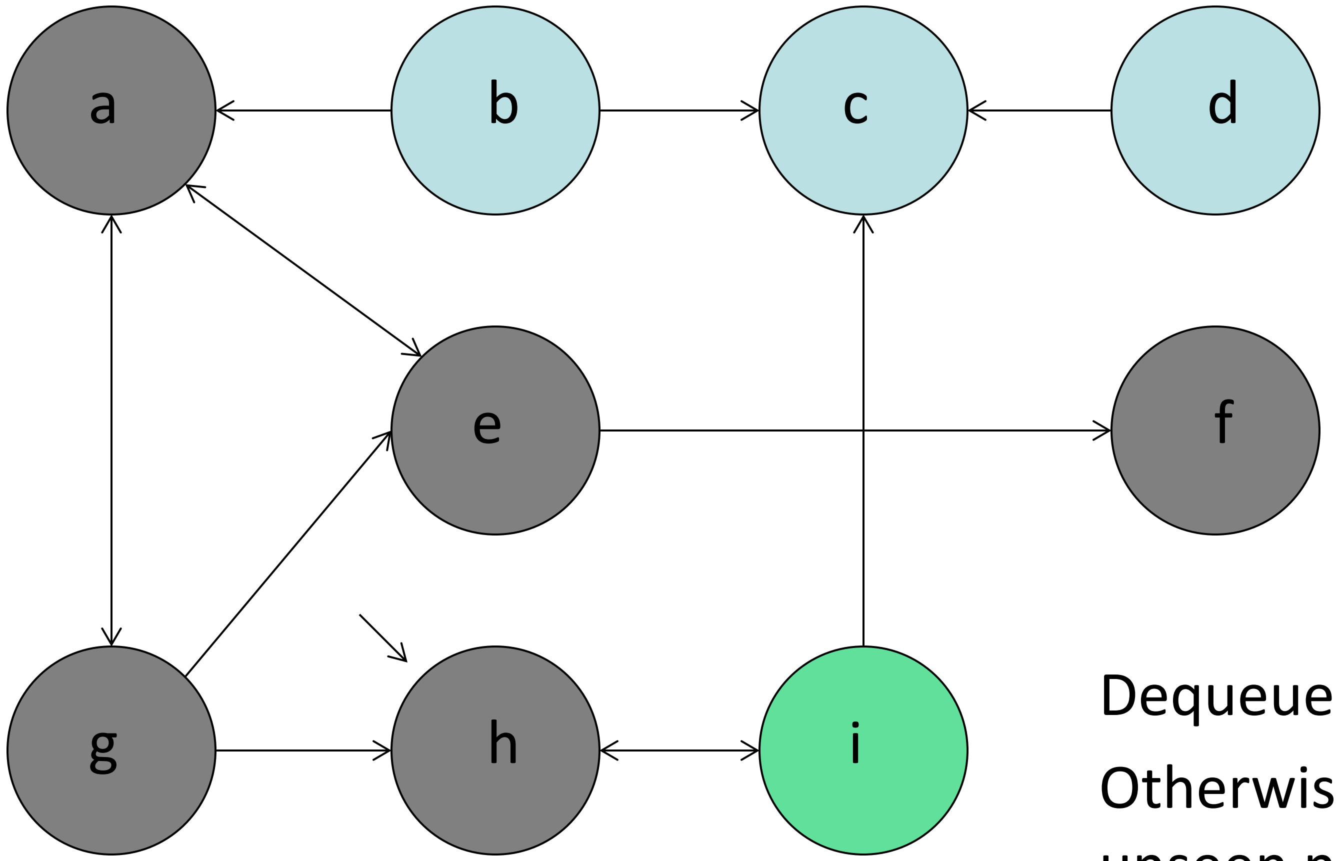


Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: h

BFS

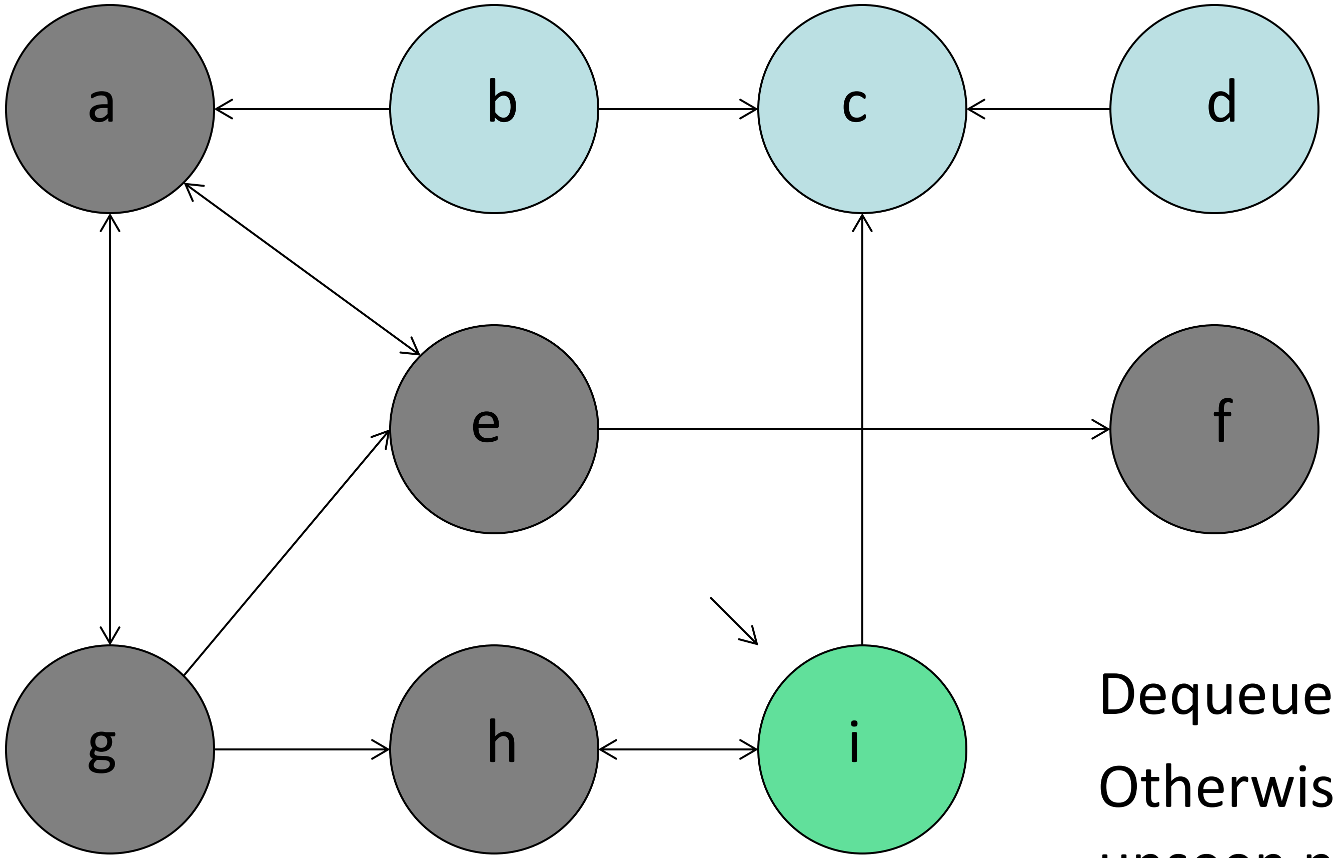


Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: i

BFS

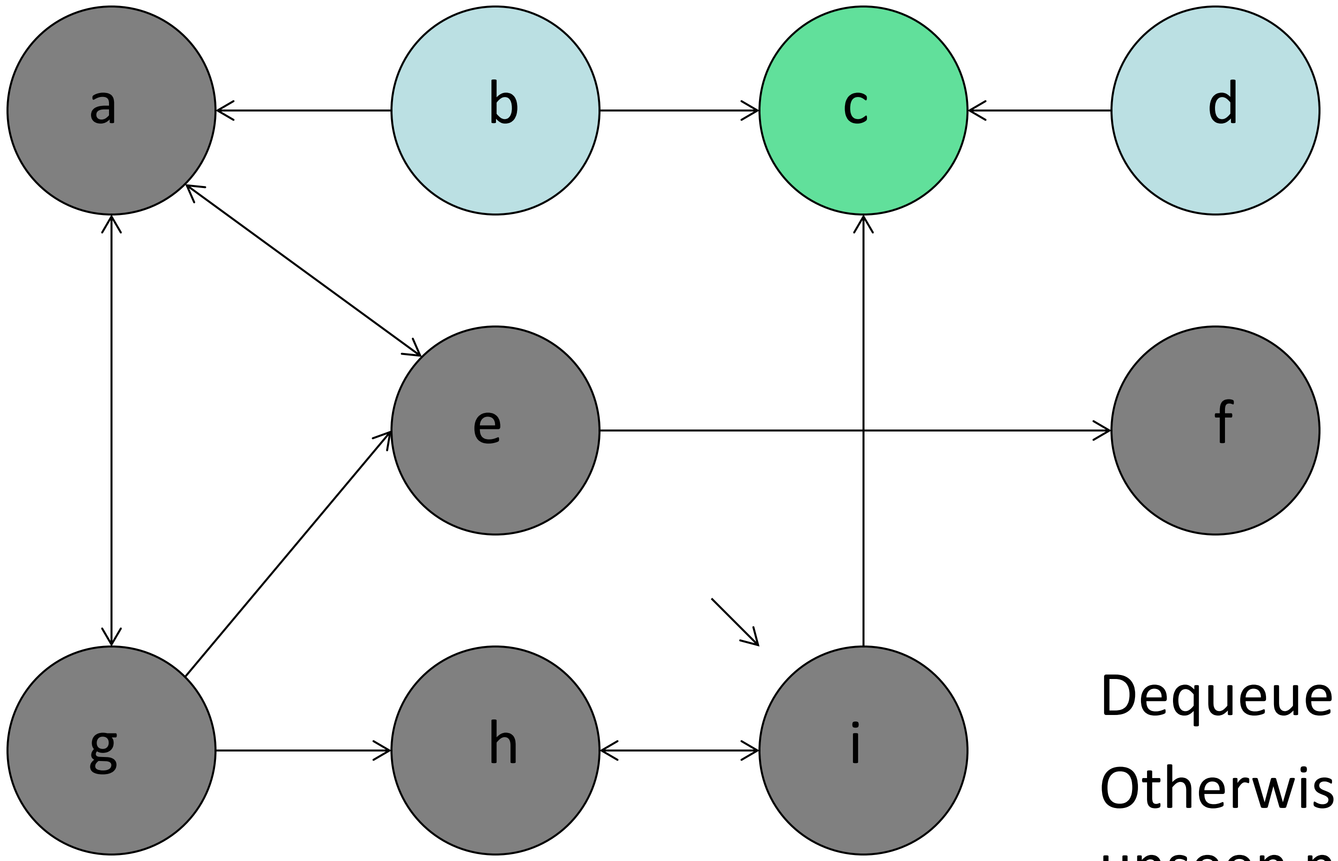


Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: i

BFS

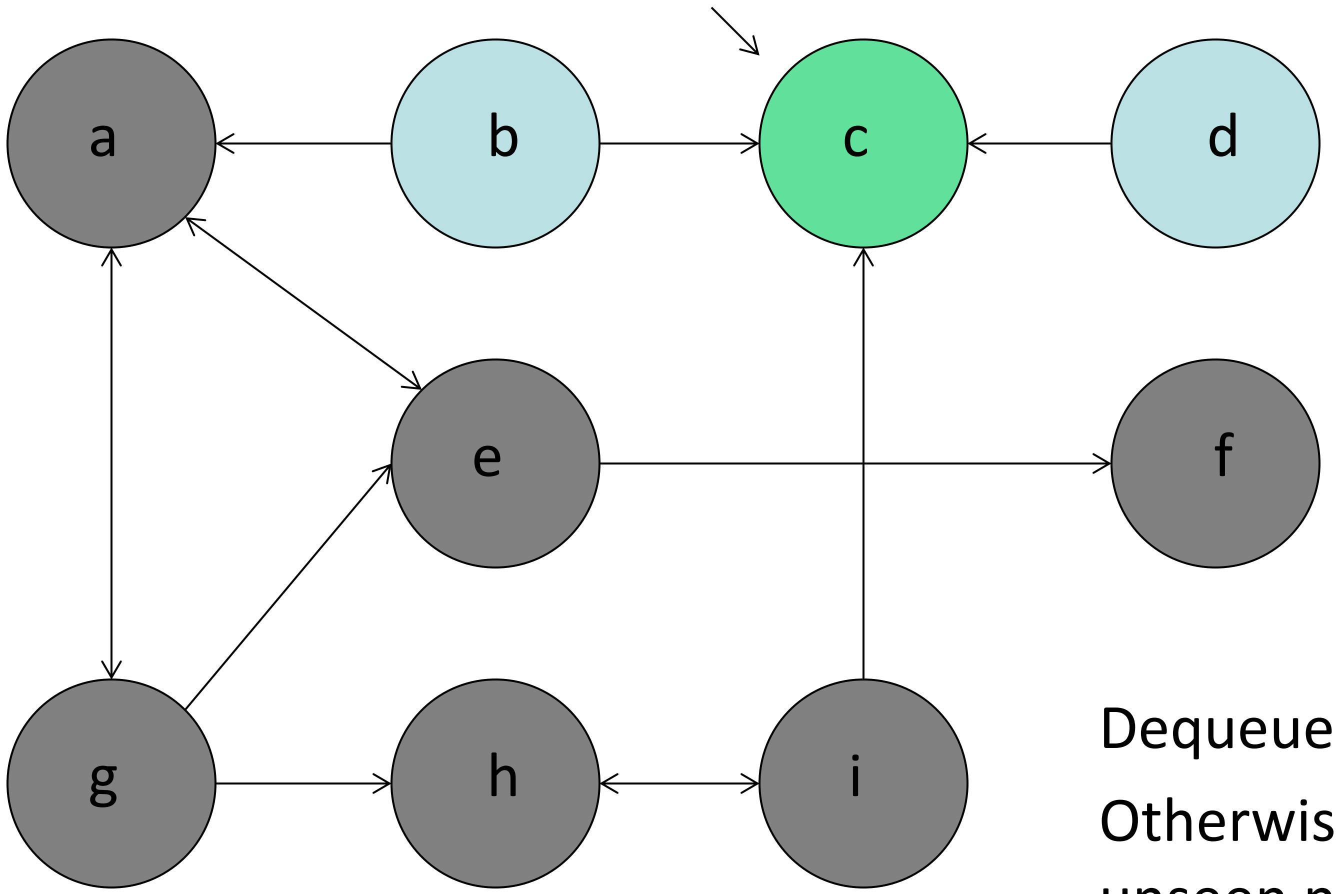


Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: c

BFS

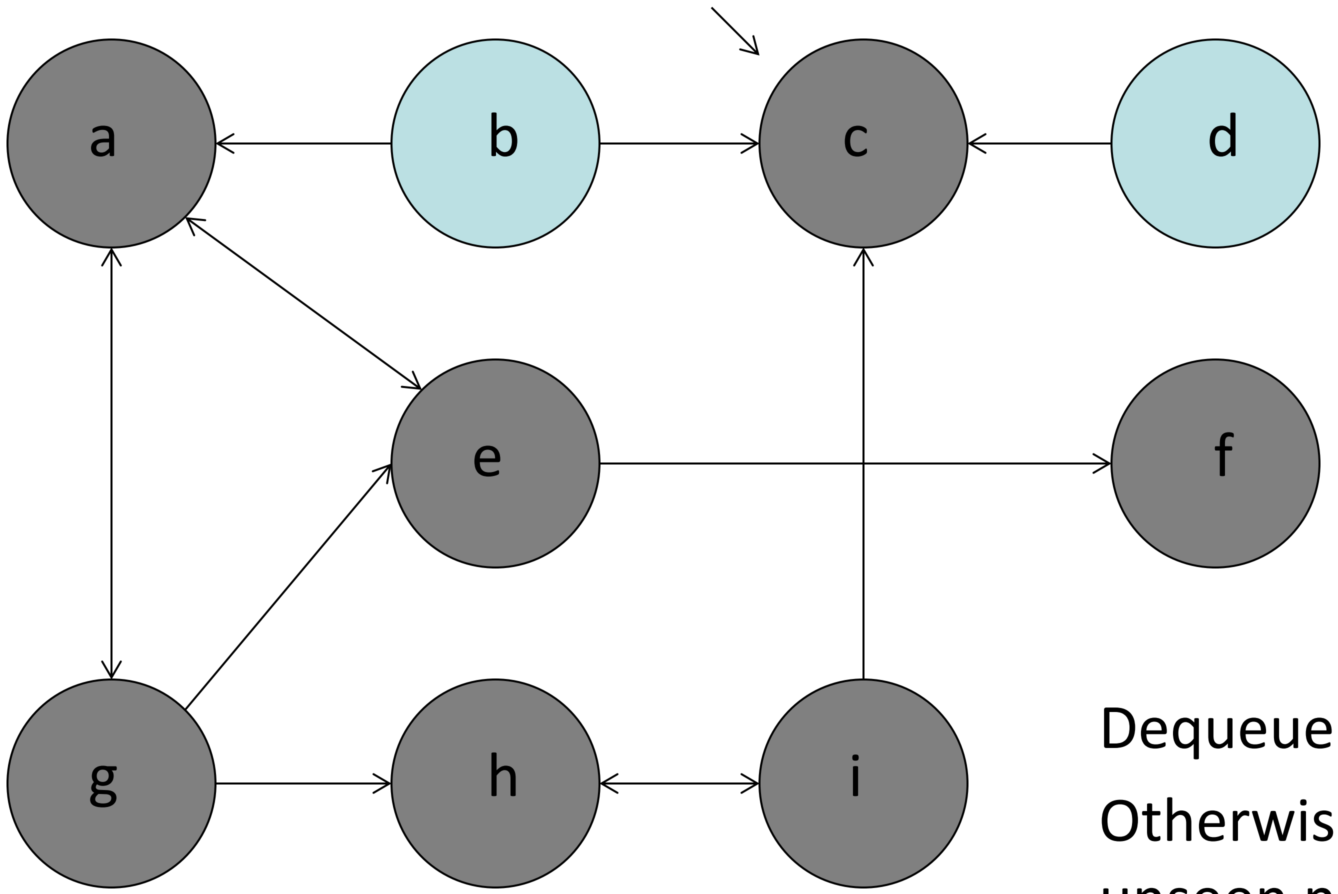


Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: c

BFS



Dequeue a node

Otherwise, add all its unseen neighbors to the queue

queue: c

BFS Details

- In an *n*-node, *m*-edge graph, takes O(*m* + *n*) time with an adjacency list
  - Visit each edge once, visit each node at most once
- Pseudocode:

```
bfs from $v_1$:
    add $v_1$ to the queue.
    while queue is not empty:
      dequeue a node $n$
      enqueue $n$'s unseen neighbors
```

- How could we modify the pseudocode to look for a specific path?