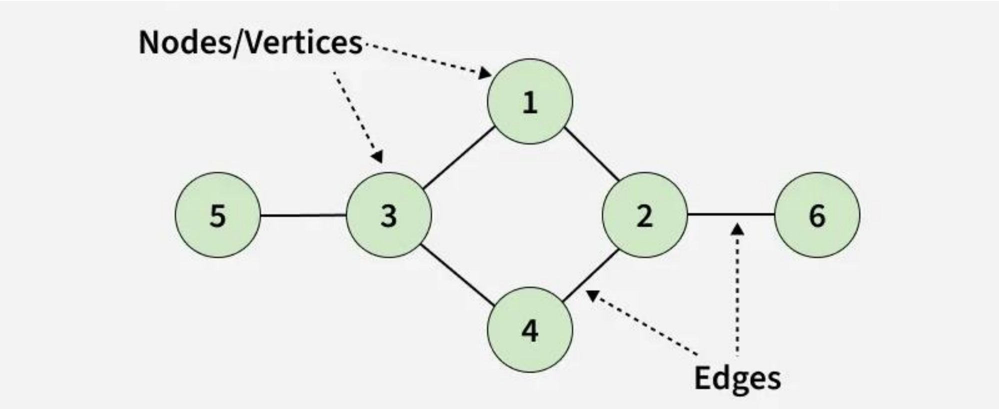
Lecture 7

Graphs II – BFS & DFS



Announcements

- Reflections on Homework 3 due Sunday night
  - **New question**: Did you use AI to assist with this assignment? If so, how?
- Group Meetings start this week
  - Self-scheduled meeting for an hour studying, working on HW, completing practice exercises
- Individual Project 1 due Friday
  - Project guide and instructions posted
  - Example posted on Ed

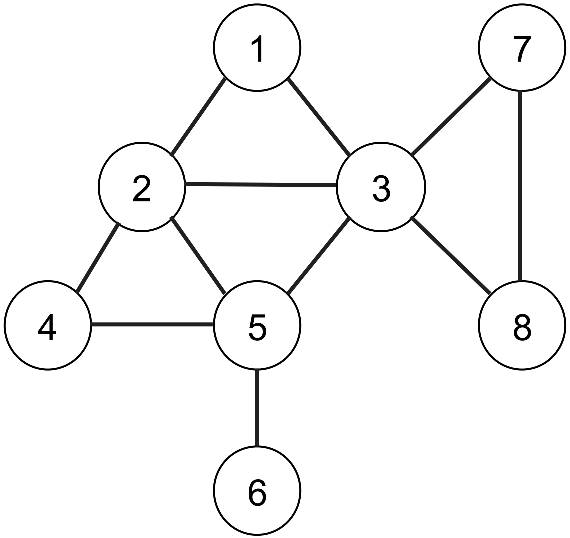
Graph Traversal

An important question about graphs:

- ❖ Can we determine if there's a path between any two vertices?

How can we solve it?

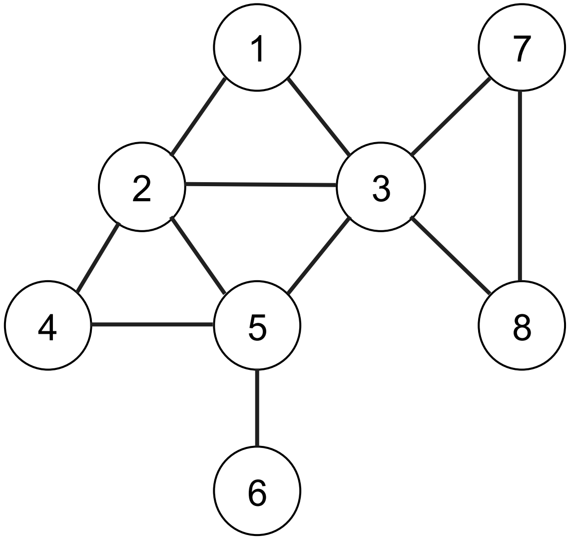
- ❖ Graph traversal
- ❖ Bread-first search (BFS): explore locally
- ❖ Depth-first search (DFS): deep dive and backtrack



Breadth-First Search

Explore outward from starting node by distance

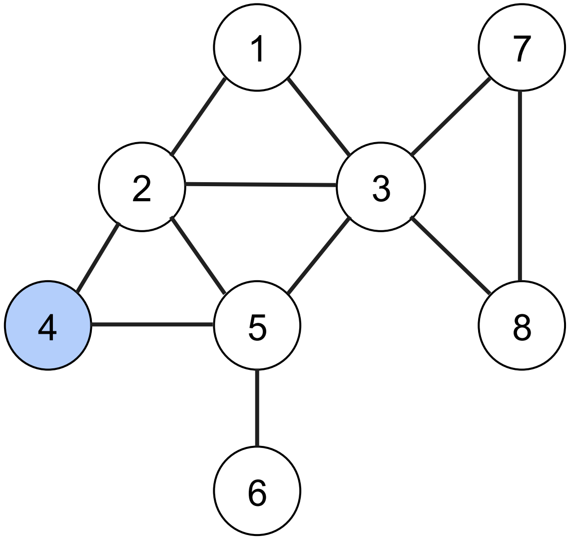
❖ "Expanding Wave"



Breadth-First Search

Explore outward from starting node by distance

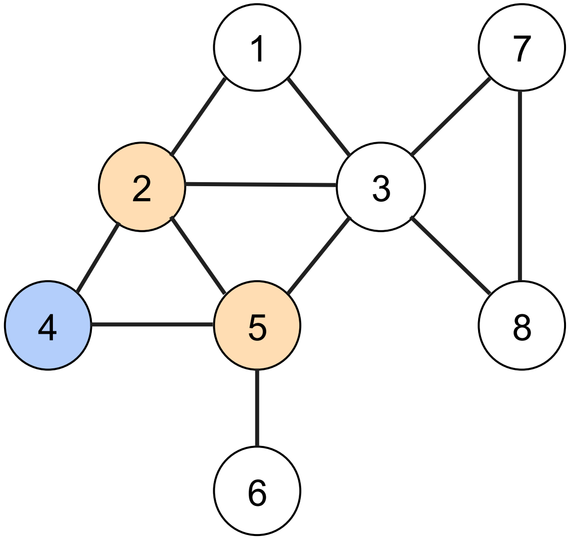
- "Expanding Wave"
- Let's start at vertex 4
    - Distance 0 from 4



Breadth-First Search

Explore outward from starting node by distance

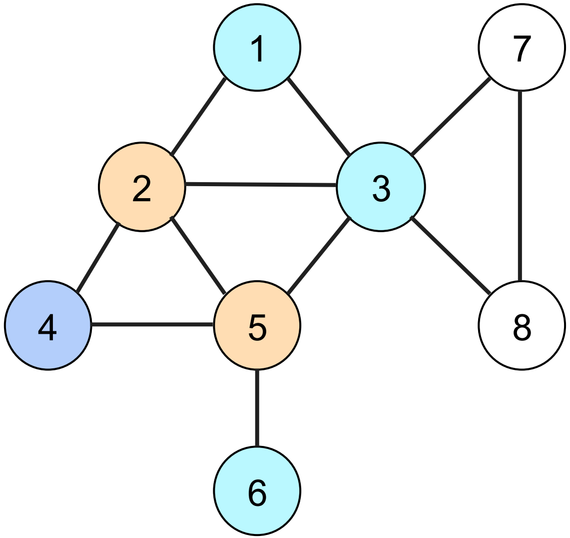
- "Expanding Wave"
- Let's start at vertex 4
  - Distance 0 from 4
  - Distance 1 from 4



Breadth-First Search

Explore outward from starting node by distance

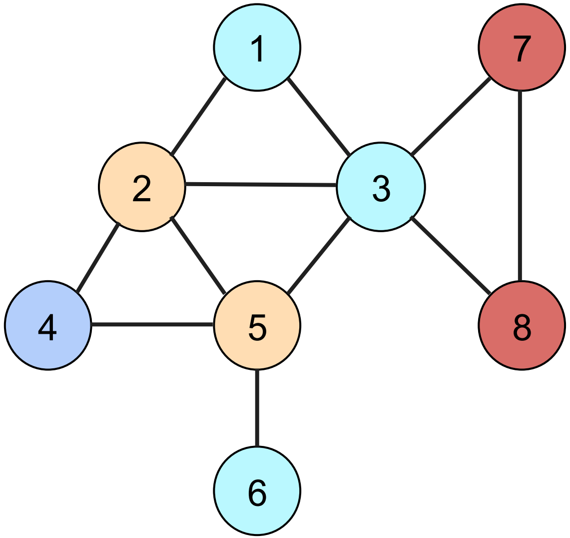
- "Expanding Wave"
- Let's start at vertex 4
  - Distance 0 from 4
  - Distance 1 from 4
  - Distance 2 from 4



Breadth-First Search

Explore outward from starting node by distance

- "Expanding Wave"
- Let's start at vertex 4
  - Distance 0 from 4
  - Distance 1 from 4
  - Distance 2 from 4
  - Distance 3 from 4

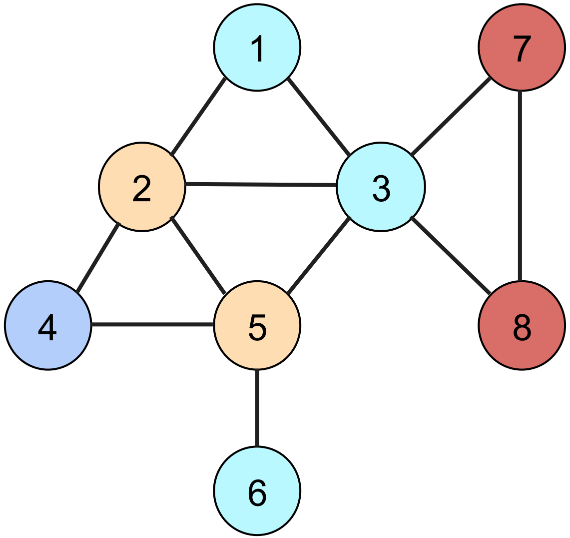


Breadth-First Search

Explore outward from starting node by distance

- "Expanding Wave"
- Let's start at vertex 4
  - Distance 0 from 4
  - Distance 1 from 4
  - Distance 2 from 4
  - Distance 3 from 4

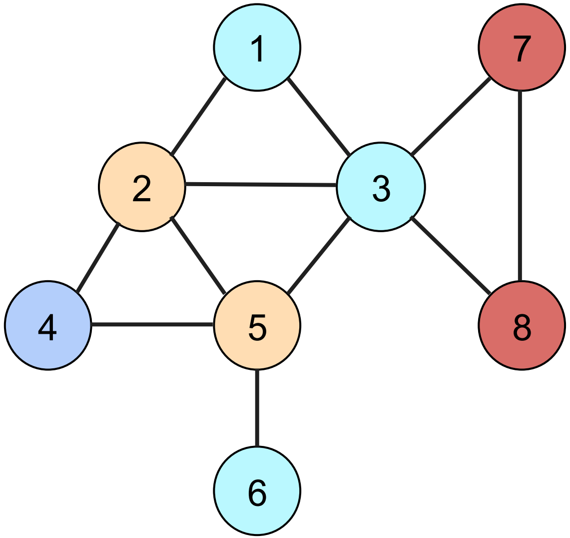
- All vertices that are reachable from 4 will be explored eventually



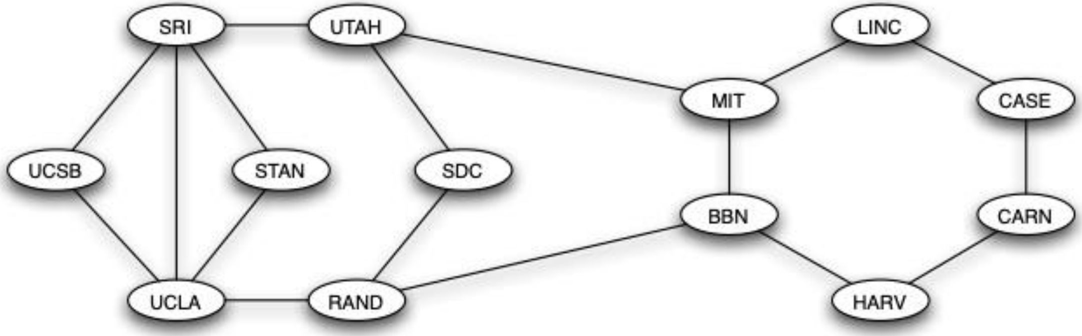
Breadth-First Search

Explore outward from starting node s by distance

- Define **layer** L_i as all vertices at distance exactly I from s
- Layers:
  - $L_0 = \{4\}$
  - $L_1 = \{2, 5\}$
  - $L_2 = \{1, 3, 6\}$
  - …
  - L_{i+1} = all vertices with an edge to a vertex in L_i that do not belong to any earlier layer
- Observation: there is a path from s to t if and only if t appears in some layer.



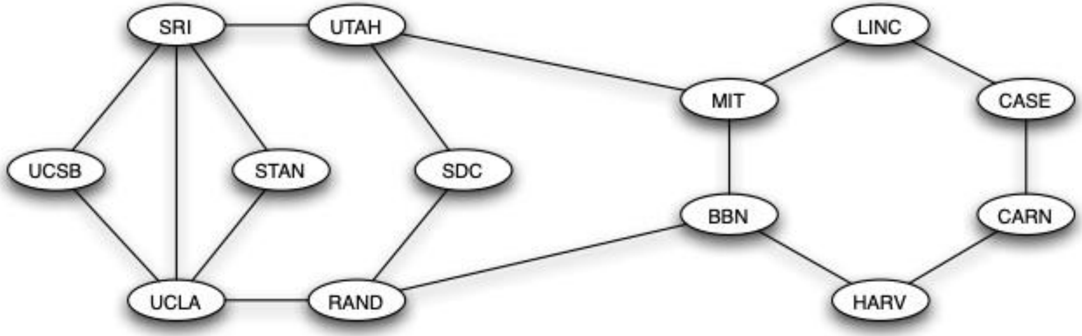
Exercise I



Q: How many vertices are in layer 2, starting a BFS from MIT?

a) 4
b) 5
c) 6
d) 42

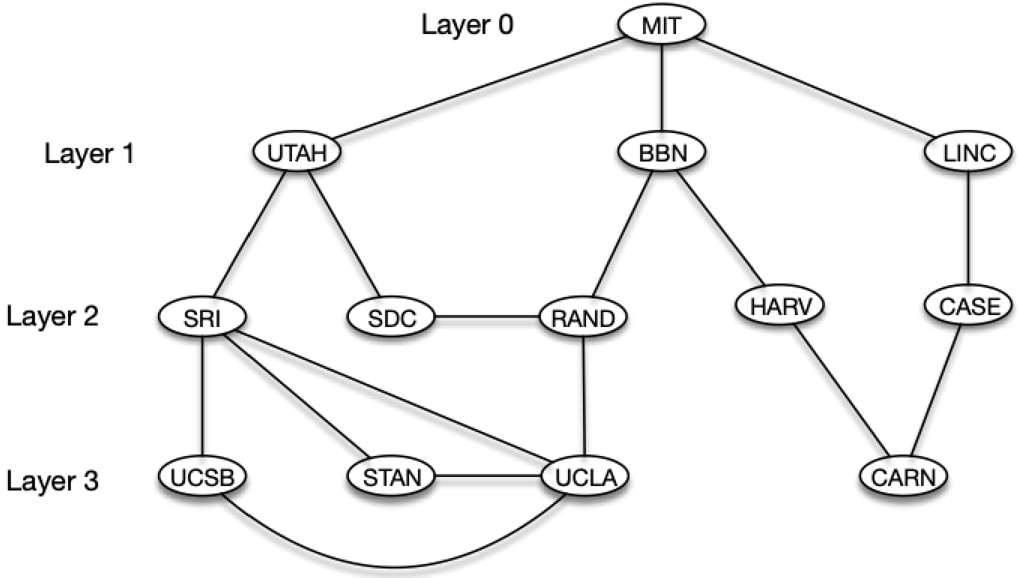
Exercise I



Q: How many vertices are in layer 2, starting a BFS from MIT?

a) 4
b) 5
c) 6
d) 42

Exercise I


Layer 0
MIT
Layer 1
UTAH
BBN
LINC
Layer 2
SRI
SDC
RAND
HARV
CASE
Layer 3
UCSB
STAN
UCLA
CARN

BFS Implementation

BFS(s):

mark s as "discovered"

$L[0] \leftarrow \{s\}; i \leftarrow 0$

while $L[i]$ is not empty **do**

$L[i+1] \leftarrow$ empty list

for all vertices v in $L[i]$ **do**

for all neighbors w of v **do**

if w is not marked "discovered" **then**

mark w as "discovered"

$L[i+1].append(w)$

$i \leftarrow i + 1$

BFS Implementation

BFS(s):

mark s as "discovered"

$L[0] \leftarrow \{s\}; i \leftarrow 0$

while $L[i]$ is not empty **do**

$L[i+1] \leftarrow$ empty list

for all vertices v in $L[i]$ **do**

for all neighbors w of v **do**

if w is not marked "discovered" **then**

mark w as "discovered"

$L[i+1].append(w)$

$i \leftarrow i + 1$

start at layer 0

iterate until we hit an empty layer

loop over all vertices in a layer and all neighbors of those vertices

if neighbor is new, add to next layer

BFS Implementation

n vertices
m edges

BFS(s):
    mark s as "discovered"
    $L[0] \leftarrow \{s\}; i \leftarrow 0$
    while $L[i]$ is not empty **do**
        $L[i+1] \leftarrow$ empty list
        for all vertices v in $L[i]$ **do**
            for all neighbors w of v **do**
                if w is not marked "discovered" **then**
                    mark w as "discovered"
                    $L[i+1].append(w)$
        $i \leftarrow i + 1$

What is the running time? Can we use the structure of the graph to obtain our bound?

BFS Implementation

n vertices
m edges

BFS(s):
    mark s as "discovered"
    $L[0] \leftarrow \{s\}; i \leftarrow 0$
    while $L[i]$ is not empty **do**
        $L[i+1] \leftarrow$ empty list
        for all vertices v in $L[i]$ **do**
            for all neighbors w of v **do**
                if w is not marked "discovered" **then**
                    mark w as "discovered"
                    $L[i+1].append(w)$
        $i \leftarrow i + 1$

constant time operations

What is the running time? Can we use the structure of the graph to obtain our bound?

BFS Implementation

n vertices
m edges

BFS(s):
    mark s as "discovered"
    $L[0] \leftarrow \{s\}; i \leftarrow 0$
    while $L[i]$ is not empty **do**
        $L[i+1] \leftarrow$ empty list ← create at most n new lists
        for all vertices v in $L[i]$ **do**
            for all neighbors w of v **do**
} looks like $n * m$ loops, but we can do better
                if w is not marked "discovered" **then**
                    mark w as "discovered"
                    $L[i+1].append(w)$
} constant time operations
        $i \leftarrow i + 1$

What is the running time? Can we use the structure of the graph to obtain our bound?

BFS Implementation

n vertices
m edges

BFS(s):

mark s as "discovered"

$L[0] \leftarrow \{s\}; i \leftarrow 0$

while $L[i]$ is not empty **do**

$L[i+1] \leftarrow$ empty list ← create at most n new lists

for all vertices v in $L[i]$ **do**

for all neighbors w of v **do**

} in total, this runs m times because there are m edges

if w is not marked "discovered" **then**

mark w as "discovered"

$L[i+1].append(w)$

} constant time operations

$i \leftarrow i + 1$

What is the running time? Can we use the structure of the graph to obtain our bound?

BFS Implementation

n vertices
m edges

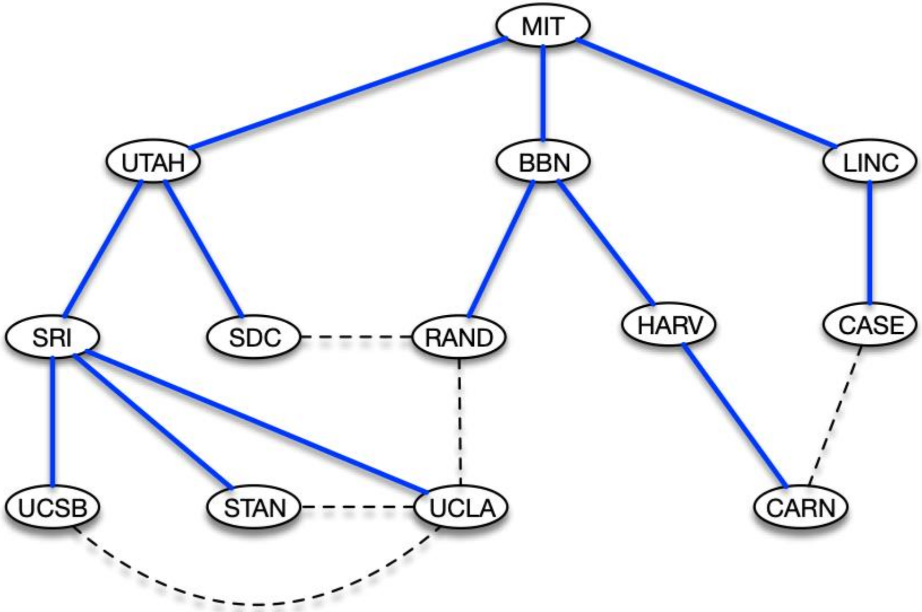
BFS(s):
    mark s as "discovered"
    $L[0] \leftarrow \{s\}; i \leftarrow 0$
    while $L[i]$ is not empty **do**
        $L[i+1] \leftarrow$ empty list ← create at most n new lists
        for all vertices v in $L[i]$ **do**
            for all neighbors w of v **do**
            } in total, this runs m times because there are m edges
                if w is not marked "discovered" **then**
                    mark w as "discovered"
                    $L[i+1].append(w)$
                } constant time operations
        $i \leftarrow i + 1$

What is the running time? $\Theta(n+m)$

BFS Tree

We can use BFS to make a tree

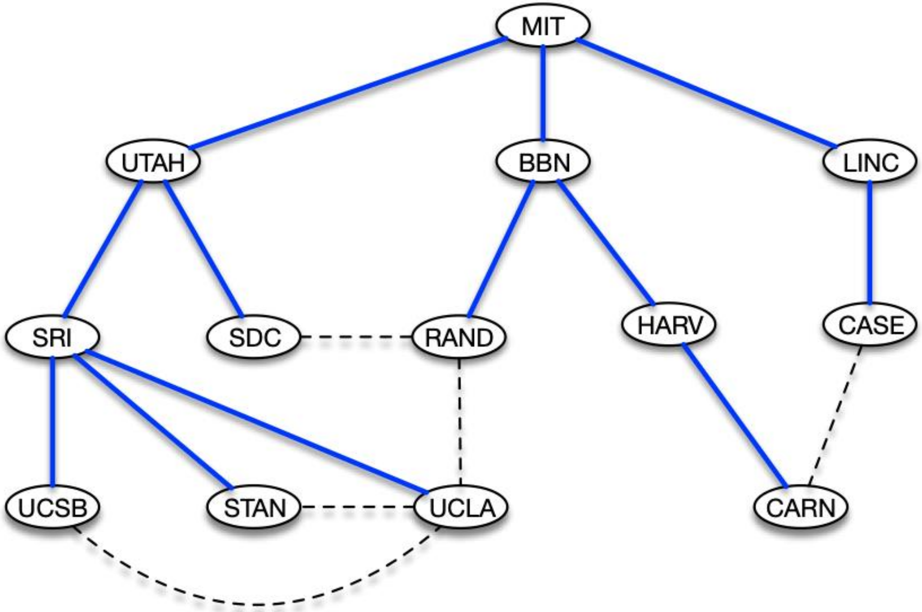
- Keep edge (v, w) if w was marked discovered as a neighbor of v
- Why does BFS make a tree?
- e.g. starting from MIT



BFS Tree

We can use BFS to make a tree

- Keep edge (v, w) if w was marked discovered as a neighbor of v
- Why does BFS make a tree?
- e.g. starting from MIT
- **Claim**: Let T be the tree discovered by BFS on graph $G = (V, E)$, and let (x, y) be any edge of G. Then the layers of x and y in T differ by at most 1.



BFS Tree

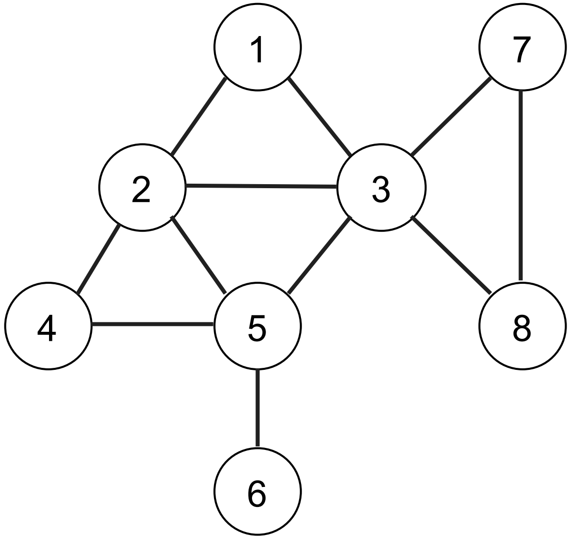
Claim: Let T be the tree discovered by BFS on graph $G = (V, E)$, and let (x, y) be any edge of G. Then the layers of x and y in T differ by at most 1.

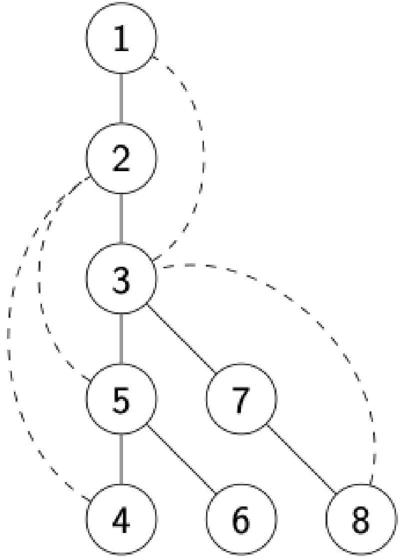
Proof:

- Let (x, y) be an edge
- Assume x is discovered first and placed in L_i
- Then $y \in L_j$ for $j \geq i$
- When neighbors of x are explored, y is either already in L_i or L_{i+1}, or is discovered and added to L_{i+1}

Depth-First Search

Keep exploring from the most recently added vertex until you reach a dead end, then backtrack





Depth-First Search

DFS(u):

  mark u as "explored"

  for all edges (u, v) **do**

    if w is not "explored" **then**

      call DFS(v) recursively

Depth-First Search

DFS(u):

mark u as "explored"

for all edges (u, v) **do**

if w is not "explored" **then**

call DFS(v) recursively

visit each vertex once

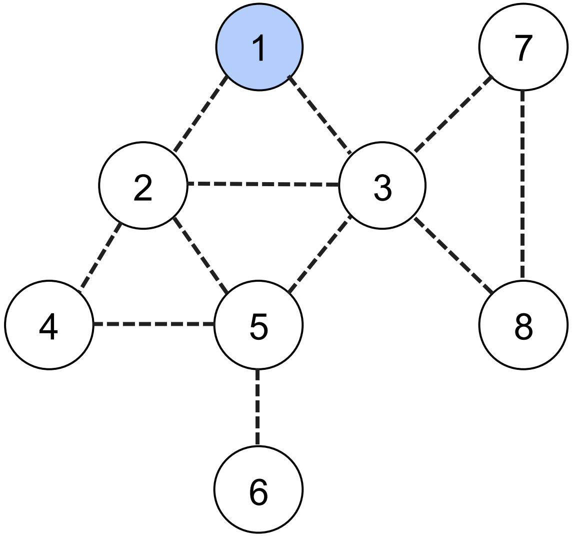
iterate over all edges

What is the running time? $\Theta(n + m)$

DFS Tree

We can use DFS to make a tree

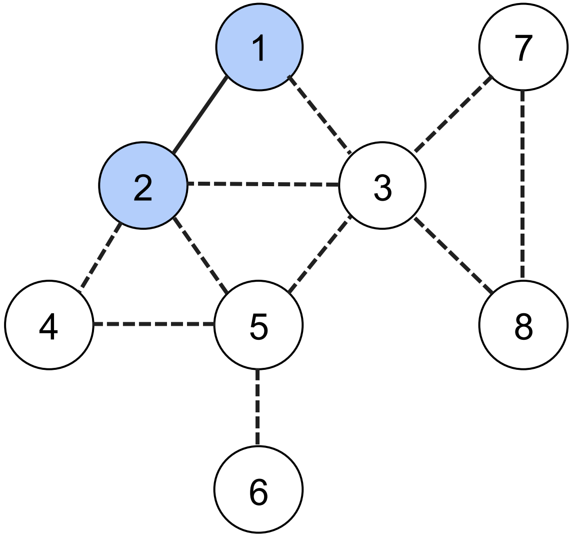
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

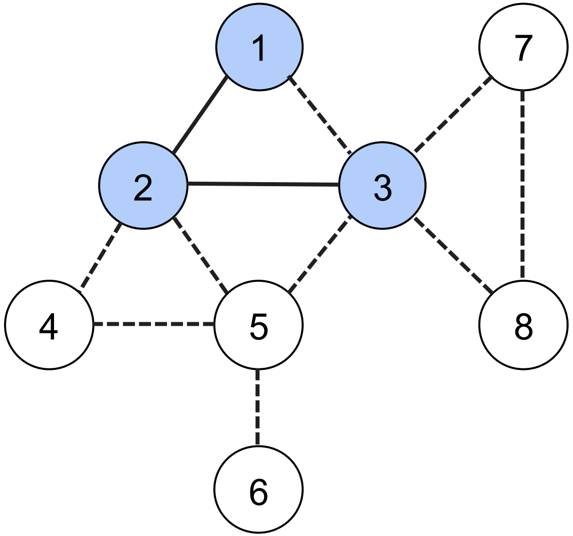
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

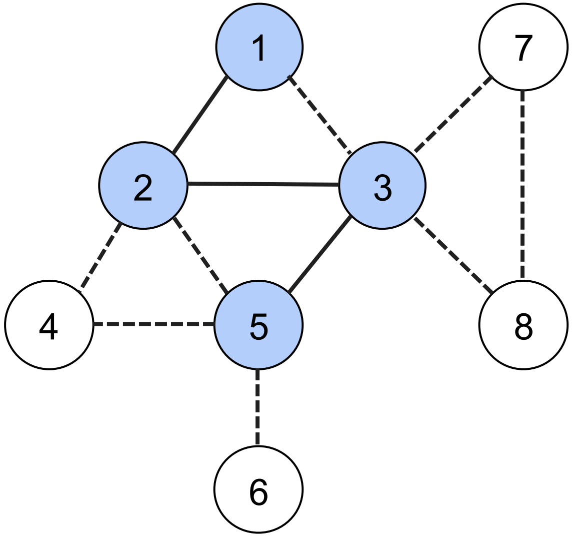
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

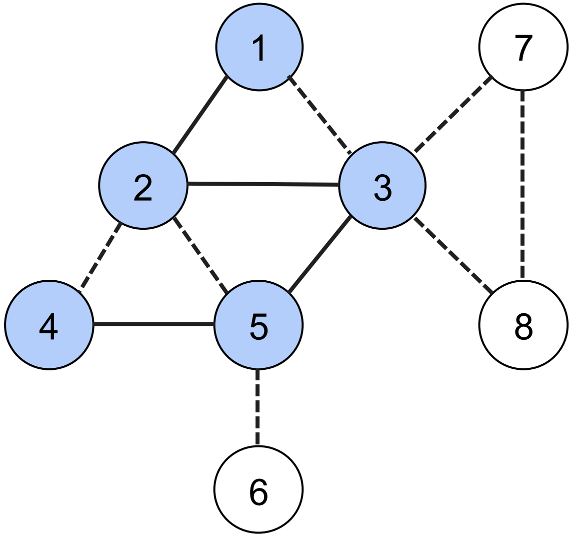
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

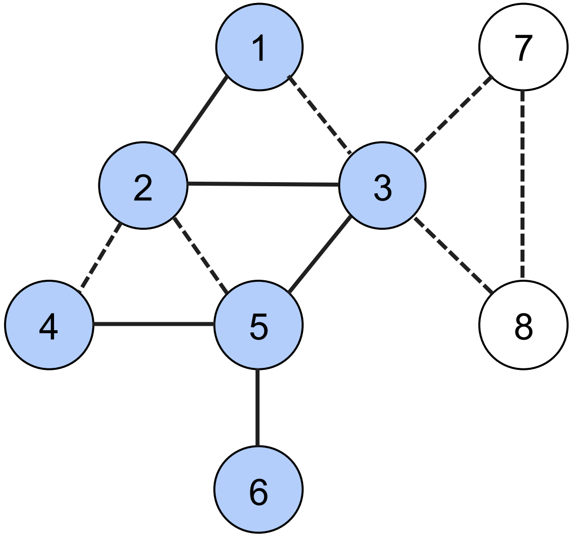
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

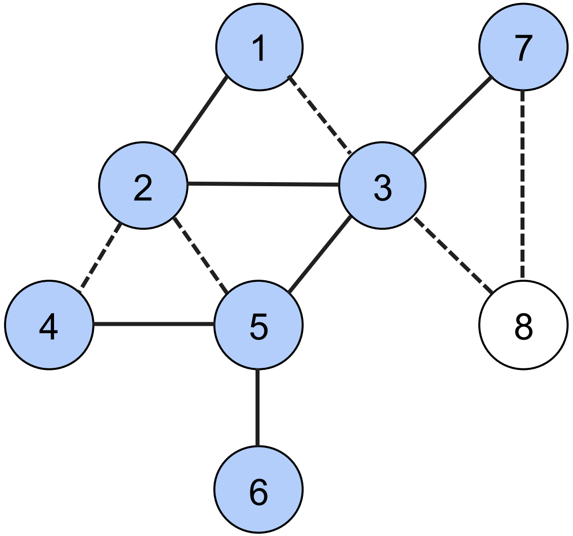
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

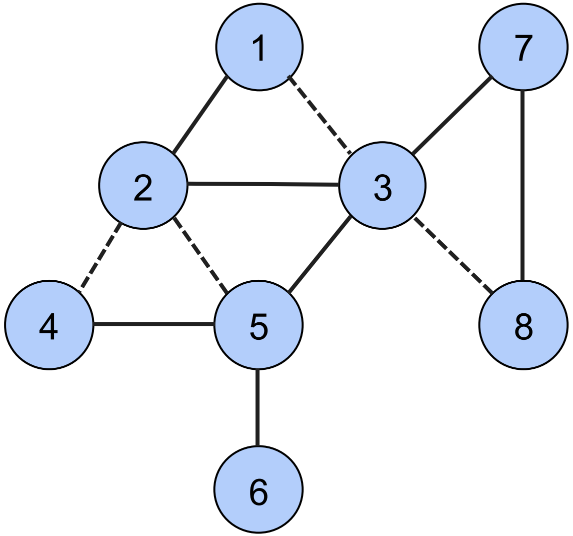
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

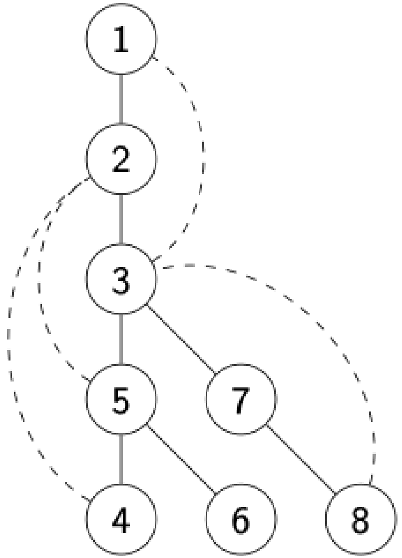
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1
- Claim: Non-tree edges lead to ancestors.



DFS Tree

Claim: Let T be the tree discovered by DFS on graph $G = (V, E)$, and let (x, y) be any edge of G that is not in T. Then one of x or y is an ancestor of the other.

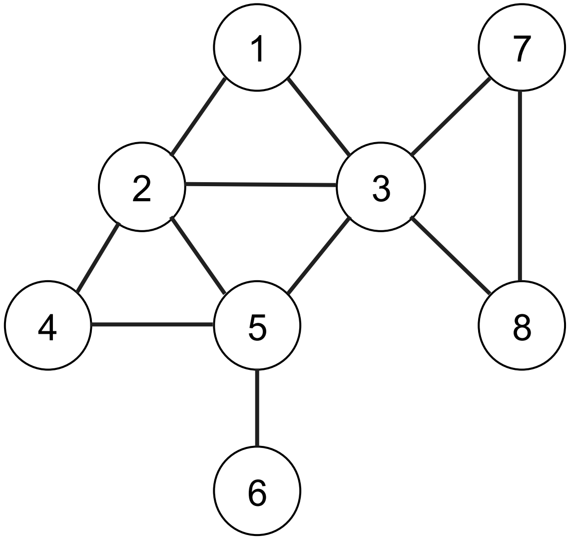
Proof:

- Let x be the first of the two vertices explored
- Is y explored at the beginning of DFS(x)? No.
- At some point during DFS(x), we examine the edge (x, y). Is y explored then? Yes, otherwise, we would put (x, y) in T
- Implies y was explored during DFS(x)
- Therefore, y is a descendant of x

Generic Traversals

Maintain a set of explored vertices and discovered vertices

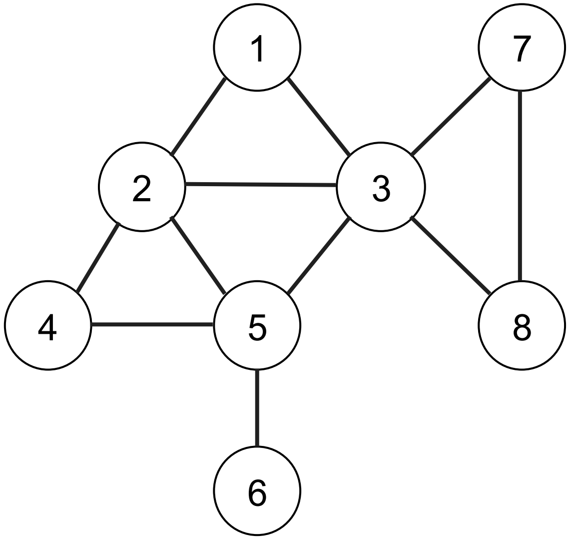
- Explored: we have seen this vertex before and explored its outgoing edges
- Discovered: the "frontier"; we have seen this vertex before, but not explored its outgoing edges
- A combination of exploring and discovering
  - See Homework 4



Generic Traversals

Maintain a set of explored vertices and discovered vertices

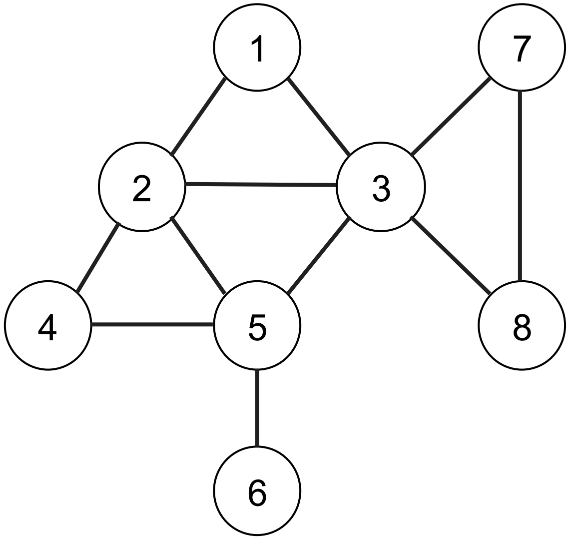
- Explored: we have seen this vertex before and explored its outgoing edges
- Discovered: the "frontier"; we have seen this vertex before, but not explored its outgoing edges
- A combination of exploring and discovering
  - See Homework 4



Exploring all Connected Components

How do you explore the entire graph even if its disconnected?

while there is an explored vertex s **do**
  Traverse(s)



Running time?

- Still $\Theta(n + m)$
- Traversal of each component takes time proportional to the number of vertices and edges in that components

Note:

- It's usually okay to assume a graph is connected. State if you are doing do and why it does not trivialize the problem.

Next Time

- ❖ Dive into BSF and DSF
- ❖ Analyze implementations using stacks and queues