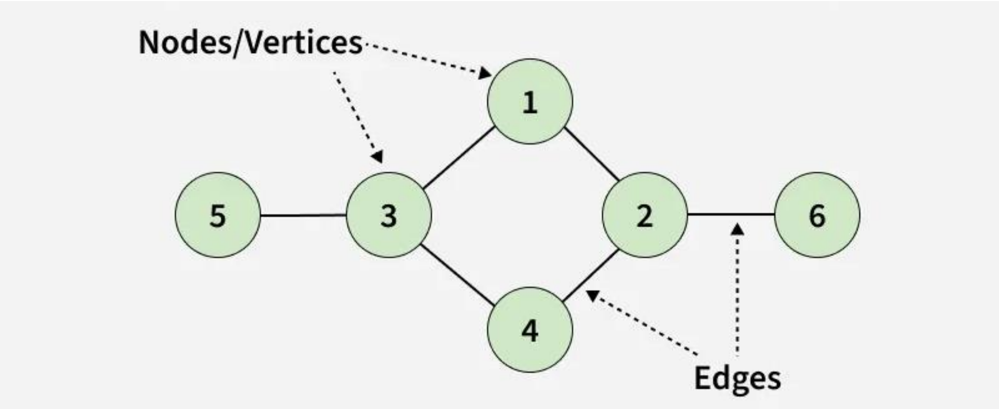
Lecture 8

Graphs III –DFS & Bipartite Testing



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

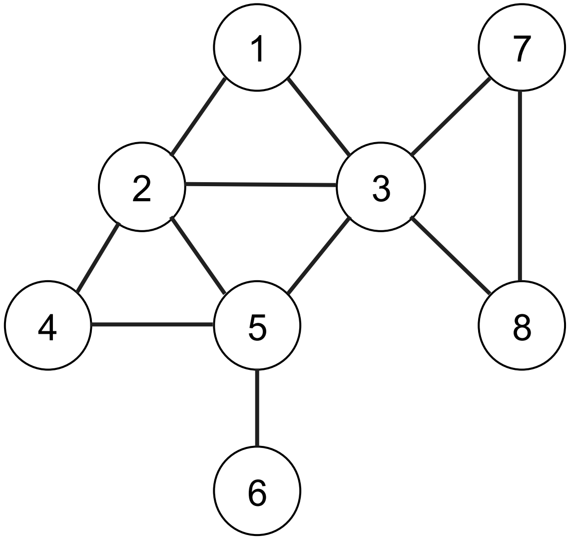
Graph Traversal

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

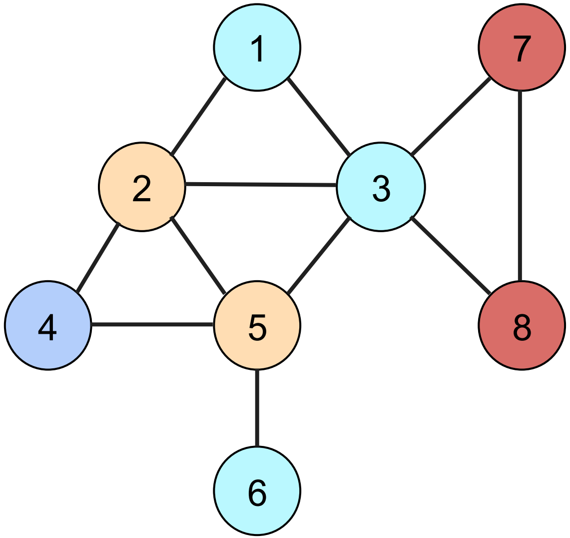


Breadth-First Search

Explore outward from starting node by distance

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

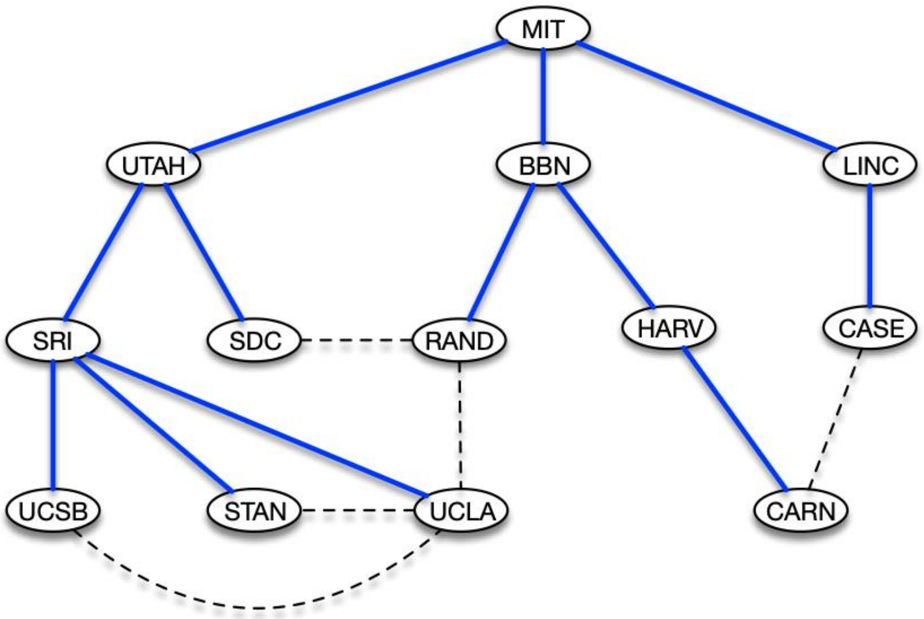
- All vertices that are reachable from 4 will be explored eventually



BFS Tree

We can use BFS to make a tree

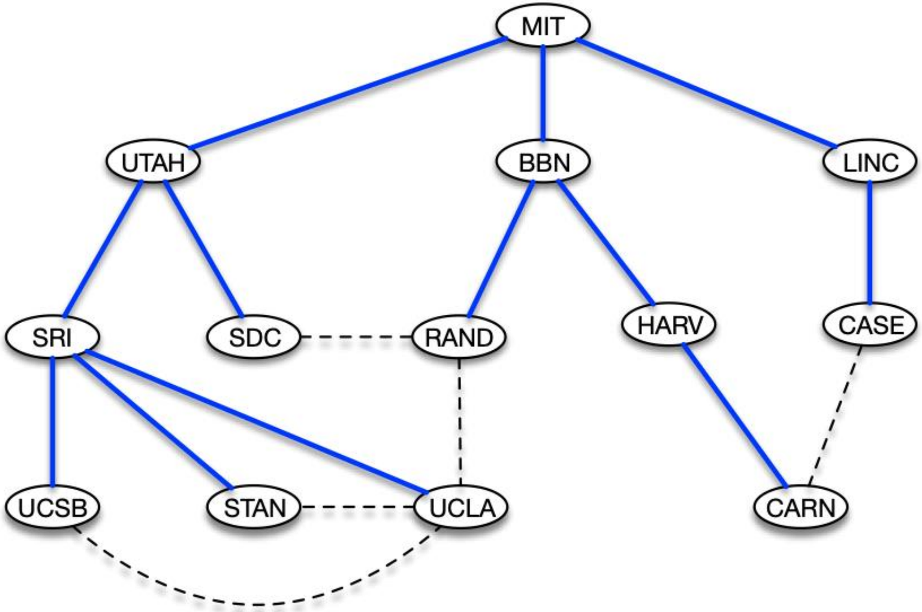
- Keep edge (v, w) if w was marked discovered as a neighbor of v
- Why does BFS make a tree?
- e.g. starting from MIT



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

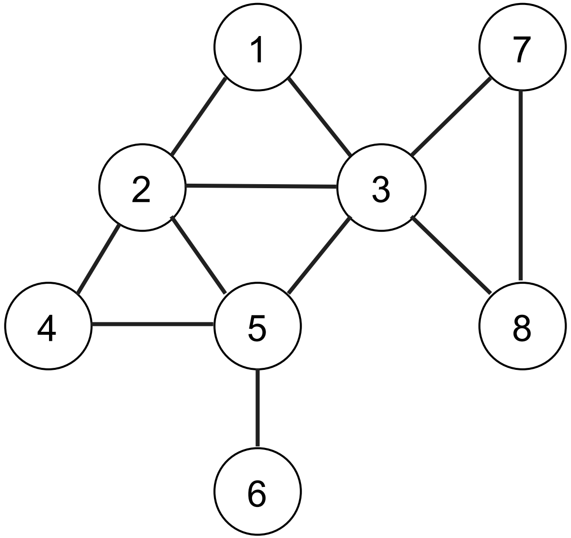
Claim: Let T be the tree discovered by BFS on graph $G = (V, E)$, and let (x, y) be any edge of G. Then the layers of x and y in T differ by at most 1.

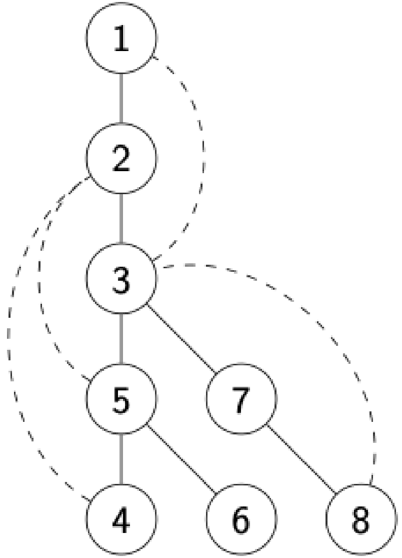
Proof:

- 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

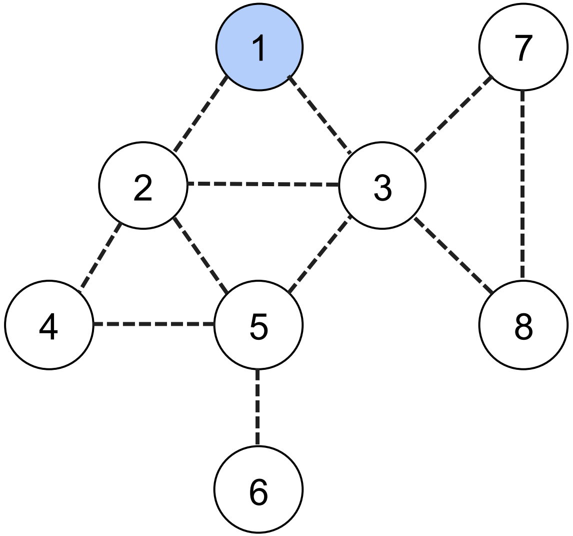
iterate over all edges

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

DFS Tree

We can use DFS to make a tree

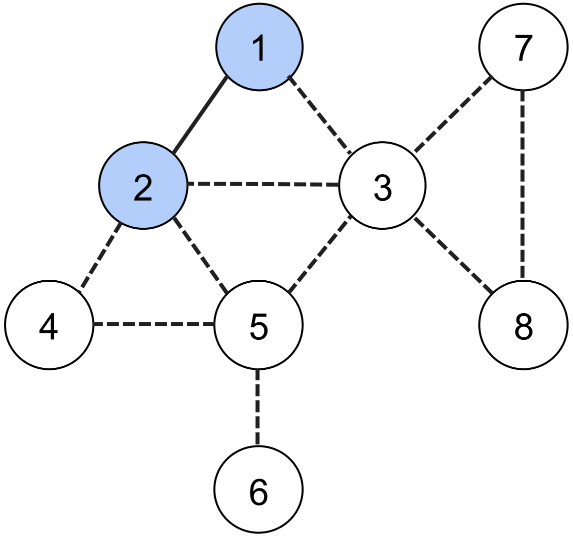
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

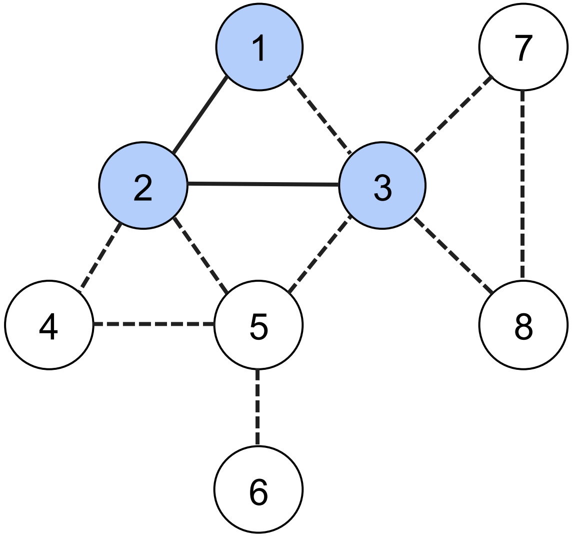
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

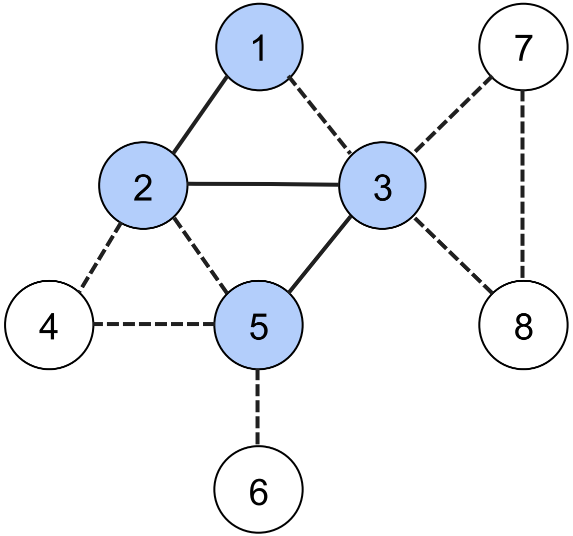
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

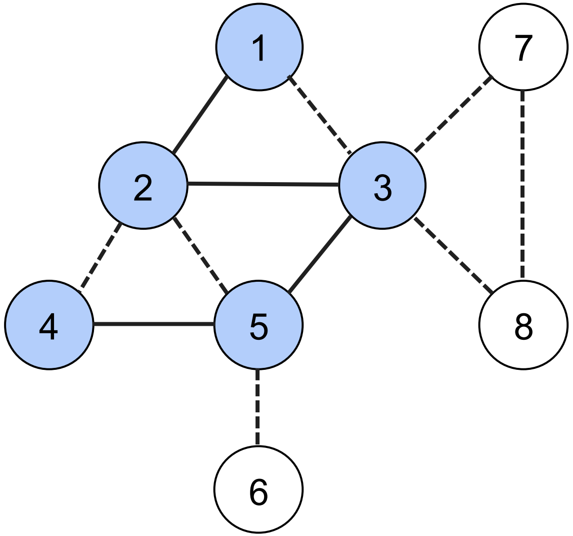
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

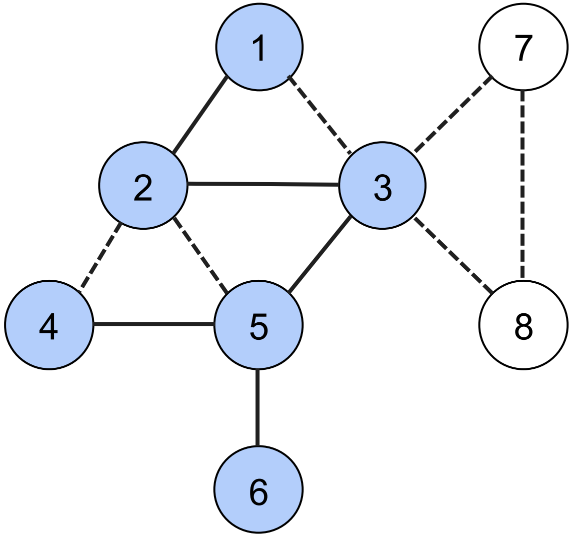
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

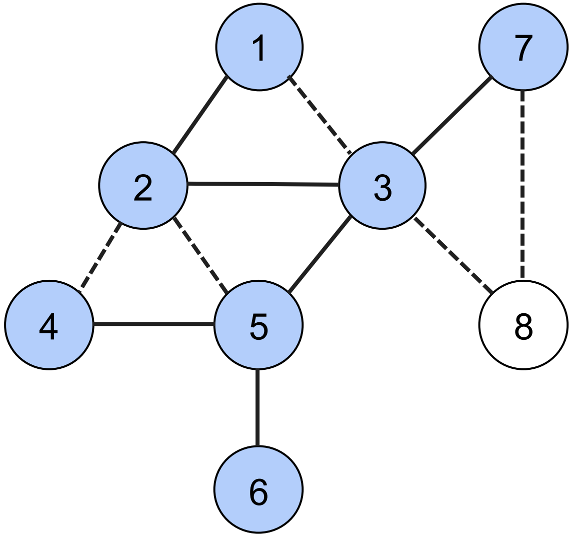
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

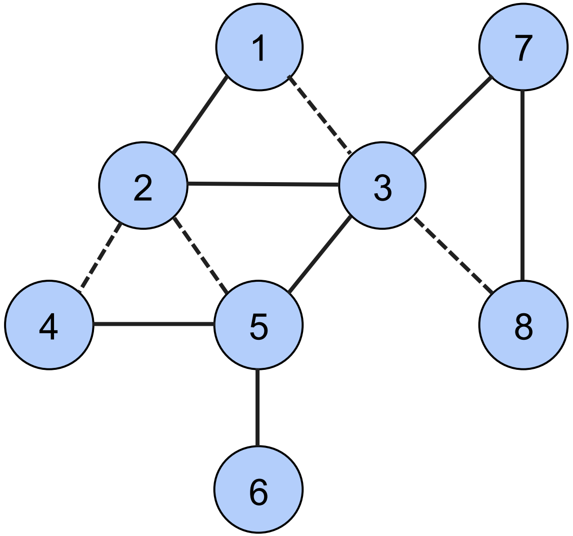
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



DFS Tree

We can use DFS to make a tree

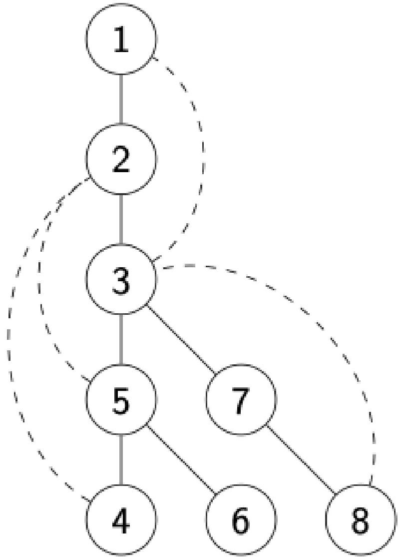
- Keep edge (v, w) if w was explored as a neighbor of v
- Why does DFS make a tree?
- e.g. starting from 1



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.

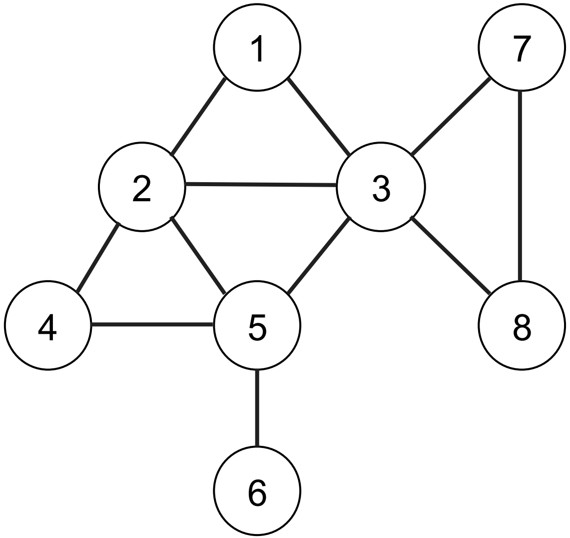
Proof:

- 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

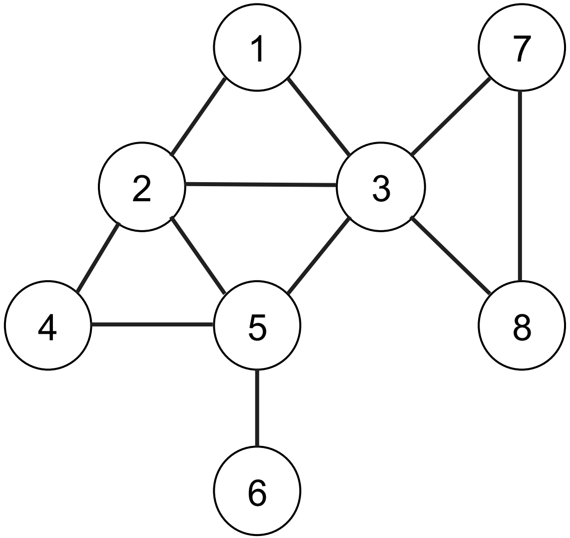
- Explored: we have seen this vertex before and explored its outgoing edges
- Discovered: the "frontier"; we have seen this vertex before, but not explored its outgoing edges
- A combination of exploring and discovering
  - See Homework 4



Generic Traversals

Maintain a set of explored vertices and discovered vertices

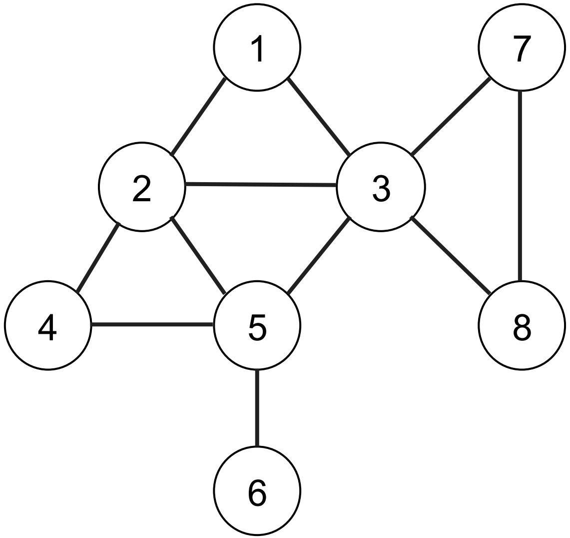
- Explored: we have seen this vertex before and explored its outgoing edges
- Discovered: the "frontier"; we have seen this vertex before, but not explored its outgoing edges
- A combination of exploring and discovering
  - See Homework 4



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.

Bipartite Graphs

A graph $G = (V, E)$ is **bipartite** if V can be partitioned into sets X, Y such that every edge has one end in X and one in Y

- Means you can color vertices red/blue s.t. no edges between vertices have the same color

Examples

Bipartite Graphs

A graph $G = (V, E)$ is **bipartite** if V can be partitioned into sets X, Y such that every edge has one end in X and one in Y

- Means you can color vertices red/blue s.t. no edges between vertices have the same color

Examples

- Resident-Hospital graph in a stable matching is bipartite

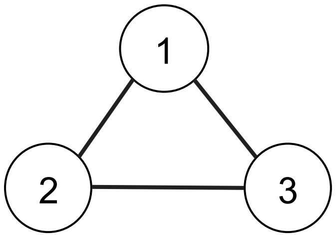
Bipartite Graphs

A graph $G = (V, E)$ is **bipartite** if V can be partitioned into sets X, Y such that every edge has one end in X and one in Y

- Means you can color vertices red/blue s.t. no edges between vertices have the same color

Examples

- Resident-Hospital graph in a stable matching is bipartite
- An odd cycle (a cycle with an odd # of vertices) is not bipartite



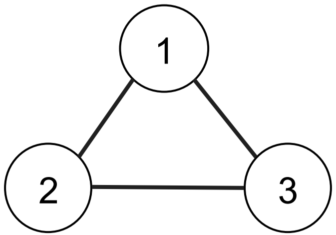
Bipartite Graphs

A graph $G = (V, E)$ is **bipartite** if V can be partitioned into sets X, Y such that every edge has one end in X and one in Y

- Means you can color vertices red/blue s.t. no edges between vertices have the same color

Examples

- Resident-Hospital graph in a stable matching is bipartite
- An odd cycle (a cycle with an odd # of vertices) is not bipartite
  - Suppose it is. Color 1 red, 2 blue, 3 red, then 1 blue… oops!



Bipartite Testing

Question: Given a graph G, is G bipartite?

Idea: run BFS from any vertex s

- $L_0 =$ red
- $L_1 =$ blue
- $L_2 =$ red
- …
- Even layers red; odd layers blue

Bipartite Testing

Question: Given a graph G, is G bipartite?

Idea: run BFS from any vertex s

- $L_O =$ red
- $L_1 =$ blue
- $L_2 =$ red
- …
- Even layers red; odd layers blue

What could go wrong?

Bipartite Testing

Question: Given a graph G, is G bipartite?

Idea: run BFS from any vertex s

- L_0 = red
- L_1 = blue
- L_2 = red
- …
- Even layers red; odd layers blue

What could go wrong? What about an edge between two vertices in the same layer?

Bipartite Testing

Run BFS from any vertex s

if there is an edge between any two vertices in the same layer **then**

    Output "not bipartite"

else

    Output "bipartite" with X being the even layers and Y being the odd layers

Correctness

Remember the fact about BFS: every edge connects vertices in the same layer or adjacent layers

Proof sketch:

- If the algorithm outputs "bipartite", then all edges connect notes in an even layer (X) and an odd layer (Y), so G is bipartite
- If the algorithm outputs "not bipartite", then there is an edge between two vertices in the same layer. We will show this implies that G has an odd cycle, so G is not bipartite.

Proof

Claim: if there is an edge between two vertices in the same layer, then G has an odd cycle.

- Let T be a BFS tree of G and suppose (x, y) is an edge between two nodes in the layer j
- Let $z \in L_i$ be the least common ancestor of x and y (useful technique in proofs)
  - Let P_{zz} be the path from z to x in T
  - Let P_{yz} be the path from z to y in T
  - The path that follows P_{zx} then edge (x, y) then P_{yz} is a cycle of length $2(j-1)+1$, which is odd
- The claim is proved, which completes the proof of the algorithm

Exercise

Q: Which of the following is true?

a) If G is bipartite, then G does note have an odd cycle

b) If G does not have an odd cycle, then G is bipartite

c) Both

d) Neither

Exercise

Q: Which of the following is true?

a) If G is bipartite, then G does note have an odd cycle

b) If G does not have an odd cycle, then G is bipartite

c) Both

d) Neither

Being bipartite is equivalent to not having an odd cycle in an undirected graph

Next Time

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