CSE 331: Algorithms & Complexity

"BFS"

Prof. Charlie Anne Carlson (She/Her)

Lecture 11

Monday September 22nd, 2025

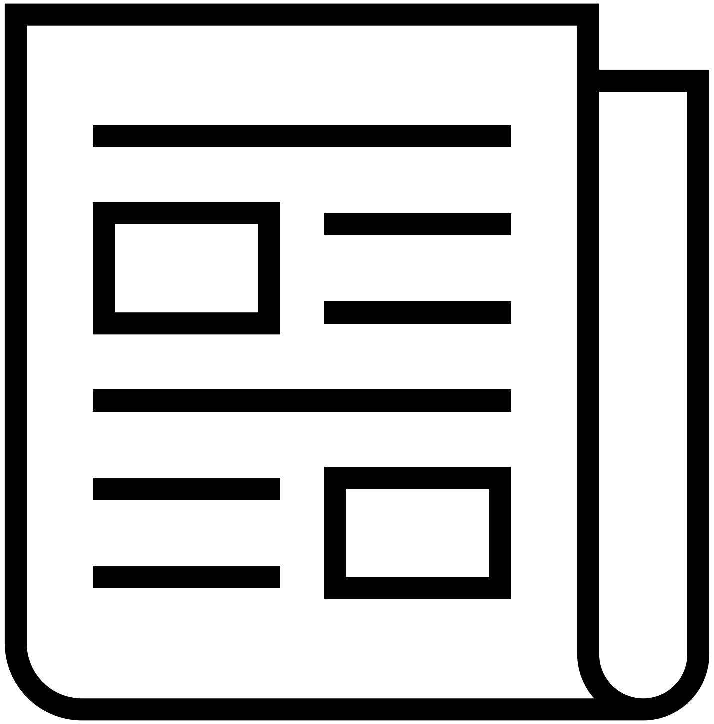
University at Buffalo

Schedule

1. Course Updates
2. Graph Connectivity
3. Graph Traversal
   1. BFS
   2. DFS
   3. WFS

Course Updates

- HW 1 Grading Out Tomorrow
- HW 2 Due Tomorrow
- HW 3 Out Tomorrow
- Group Project
  - Team Emails Soon
  - No Autolab Registration
- First Quiz NEXT Monday!



Connectivity Problem(s)

s-t Connectivity Problem:

Input: Graph $G = (V, E)$, source s, and destination t.

Output: True if there exists a path and False otherwise.

s-t Shortest Path Problem:

Input: Graph $G = (V, E)$, source s, and destination t.

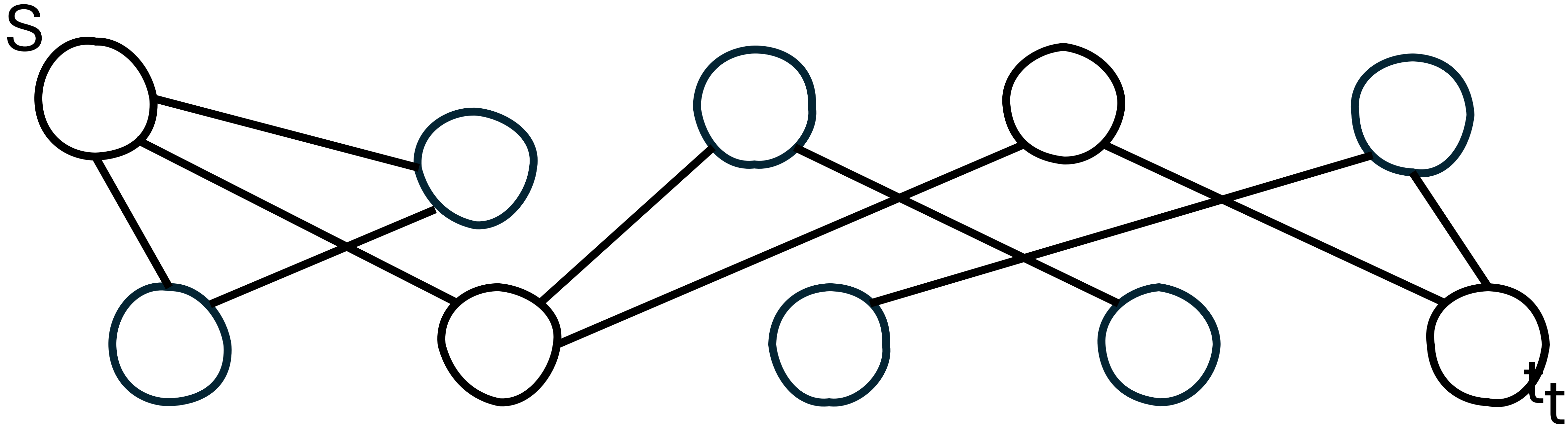
Output: The length of the shortest path from s to t (∞ if there is no path).

Another Connectivity Problem

Graph Connectivity:

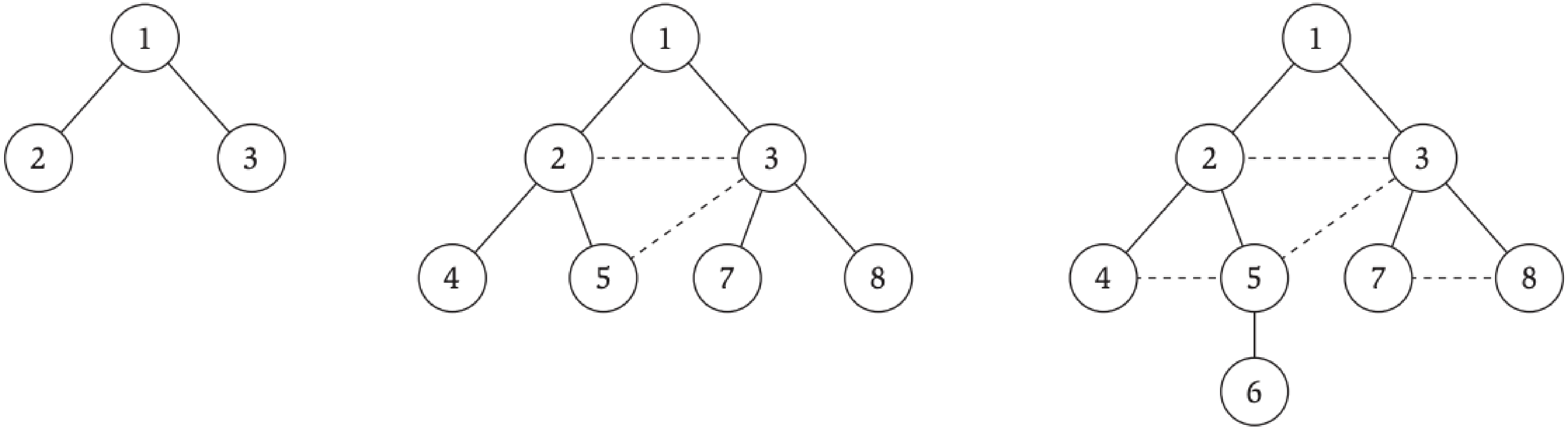
Input: $G = (V, E)$ and $s \in V$

Output: $\mathrm{T} = \{\mathrm{u} \in \mathrm{T}\text{: there exists a path from s to u in G}\}$



Breadth First Search (Layers)

- $L_0 = s$
- $L_1 =$ neighbors of L_0.
- $L_2 =$ neighbors of L_1 that are not in L_0.
- $L_\mathrm{i} =$ neighbors of $L_{\mathrm{i}-1}$ that are not in previous layer.



Breadth First Search ("Algorithm")

- Input: $G = (V, E)$ and $s \in V$
- Let $L_0 = \{s\}$
- Assume $L_0, \ldots, L_i$ have been constructed:
  - Let $L_{\mathrm{i}+1}$ be nodes do not appear in $L_0, \ldots, L_i$ and have an edge to L_i.
  - If $L_{\mathrm{i}+1}$ is empty, stop.
- Return all layers.

Breadth First Search (Properties)

- For each j ≥ 0, layer L_j produced by BFS consists of all nodes at distance j from s.
- There is a path from s to t if and only if t appears in some layer.
- For any $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$, if $u \in L_i$ and $v \in L_j$ then i and j differ by at most 1.
- You can think of the output as a tree! We call this the **BFS (discovery) Tree**.

Breadth First Search (Properties)

Claim: For any $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$, if $u \in L_i$ and $v \in L_j$ then i and j differ by at most 1.

Breadth First Search (Properties)

Claim: For any $\{u, v\} \in E$, if $u \in L_i$ and $v \in L_j$ then i and j differ by at most 1.

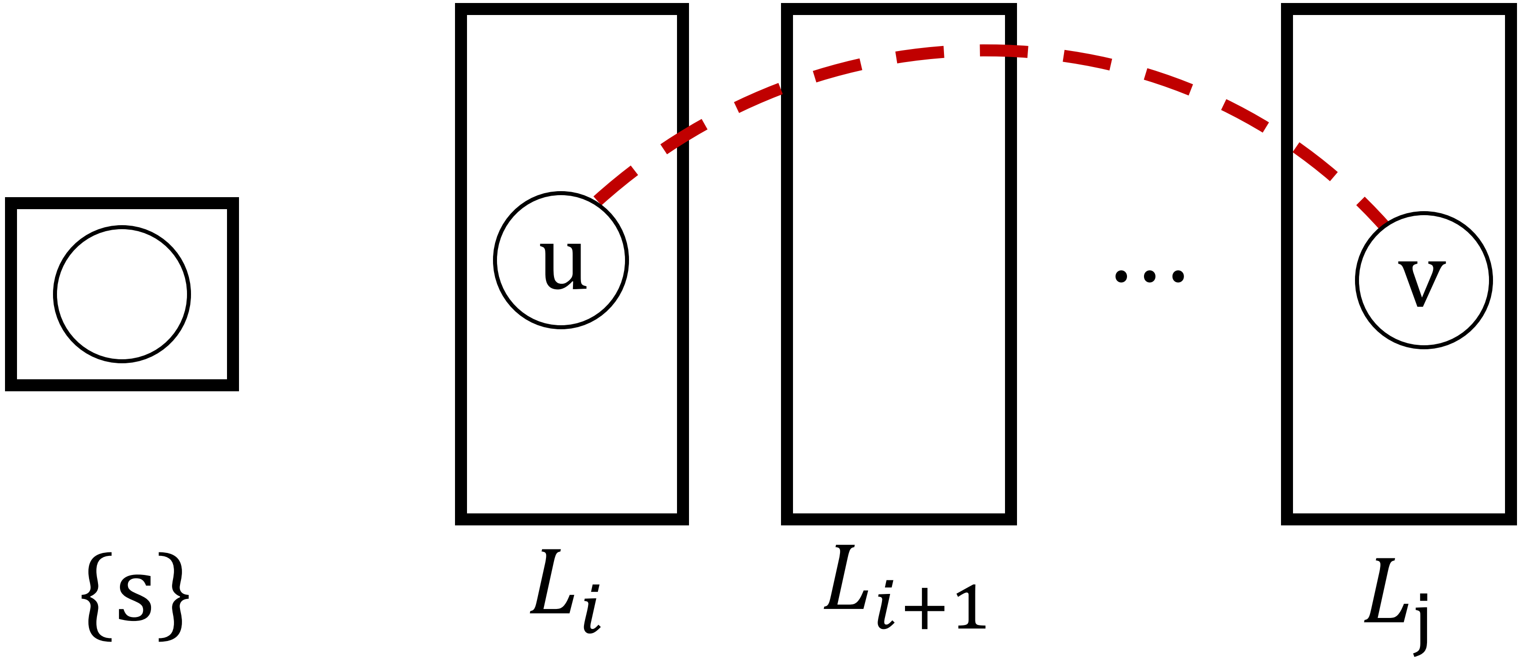
Proof Outline:

- Without loss of generality, we assume that $i \leq j$.
- Suppose for the sake of contradiction that $d(i, j) > 1$.

Breadth First Search (Properties)

Proof Outline:

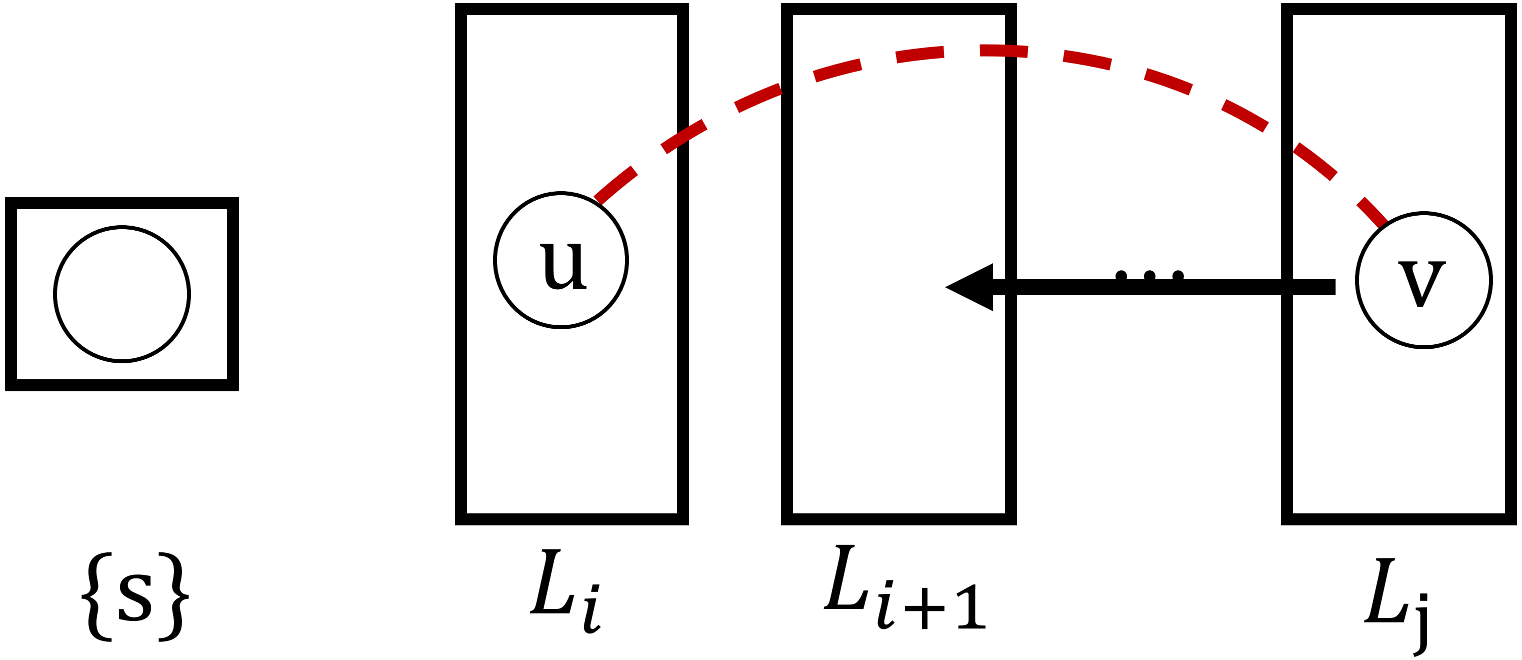
- Without loss of generality, we assume that $i < j$.
- Suppose for the sake of contradiction that $d(i, j) > 1$.



Breadth First Search (Properties)

Proof Outline:

- Without loss of generality, we assume that $i < j$.
- Suppose for the sake of contradiction that $d(i, j) > 1$.

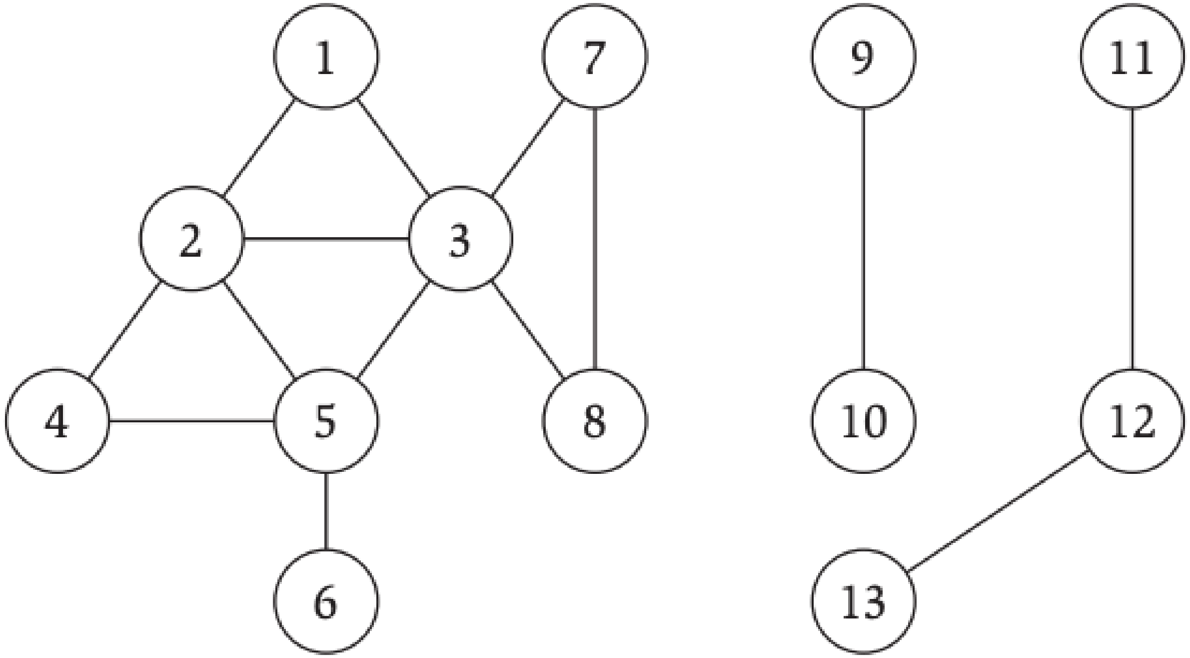


Breadth First Search (Properties)

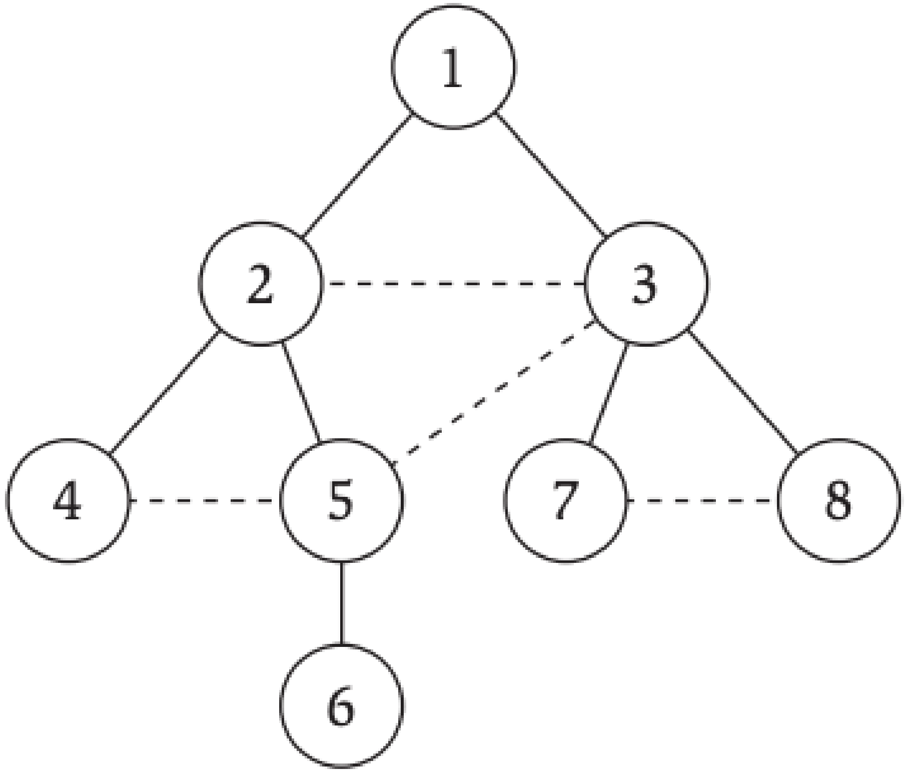
Proof:

- Without loss of generality, we assume that $\mathrm{i} < \mathrm{j}$.
- Suppose for the sake of contradiction that $\mathrm{d}(\mathrm{i}, \mathrm{j}) > 1$.
  - This implies $\mathrm{i} + 1 < \mathrm{j}$.
- Then $u \in L_i, v \notin L_0, L_1, \dots, L_i$, and $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$.
  - Hence, BFS would have put $v \in L_{i+1}$.
  - This contradicts initial assumption that $v \in L_{\mathrm{j}}$ and $\mathrm{j} > \mathrm{i} + 1$.

Breadth First Search (Tree)

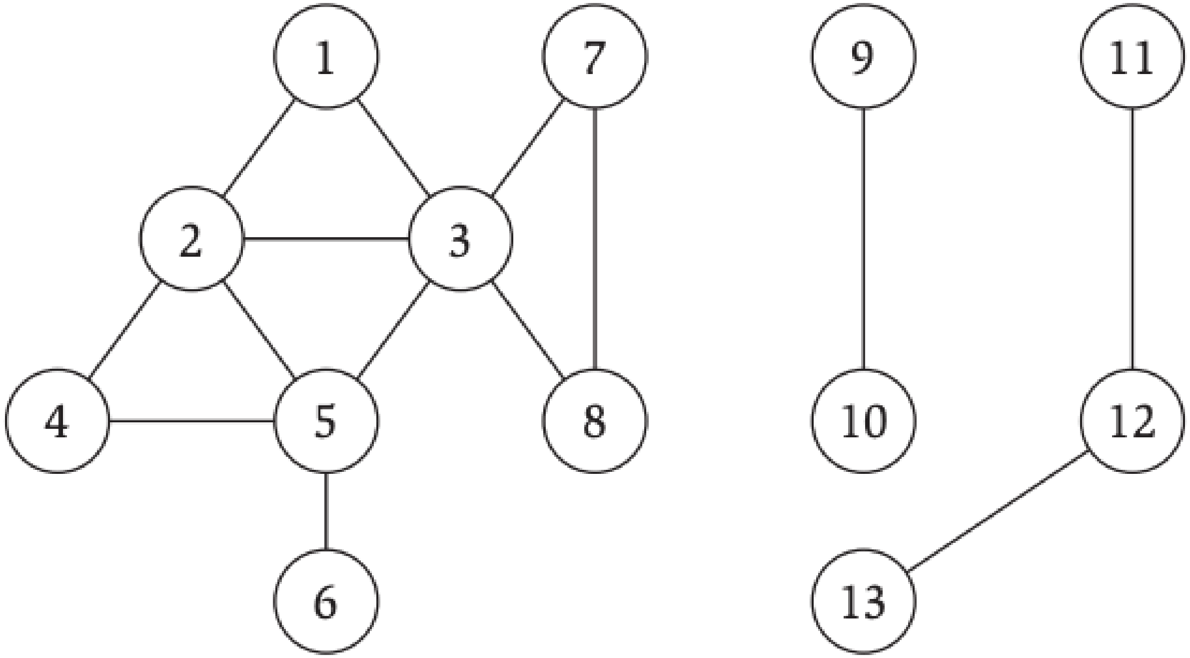


Original Graph

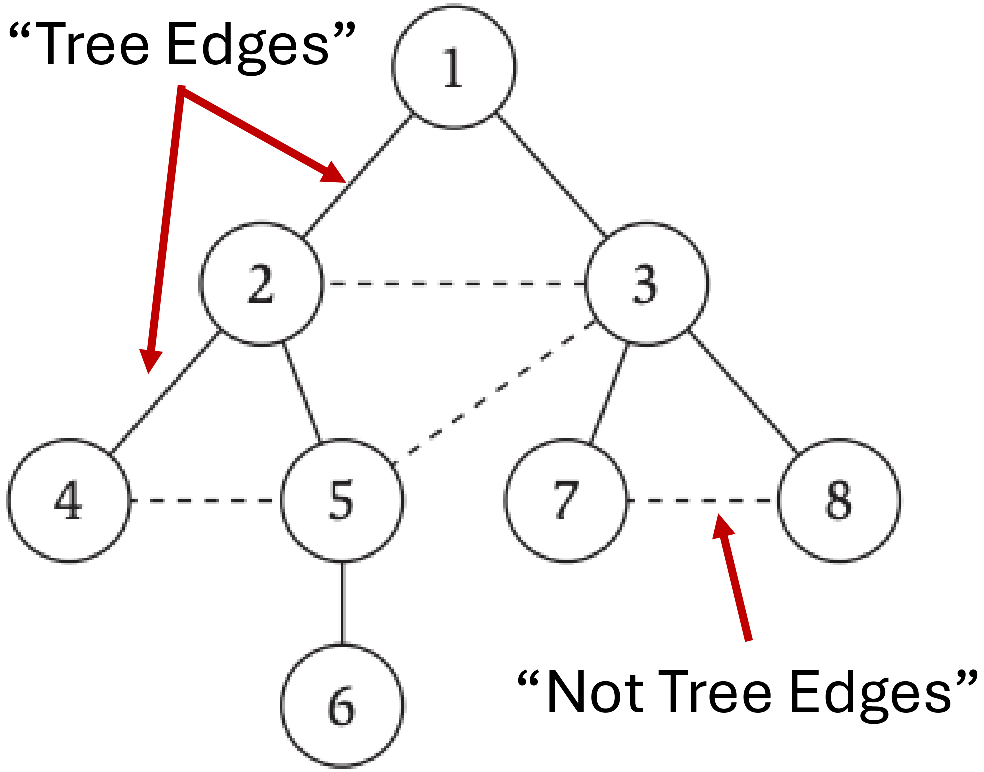


Search Tree

Breadth First Search (Tree)

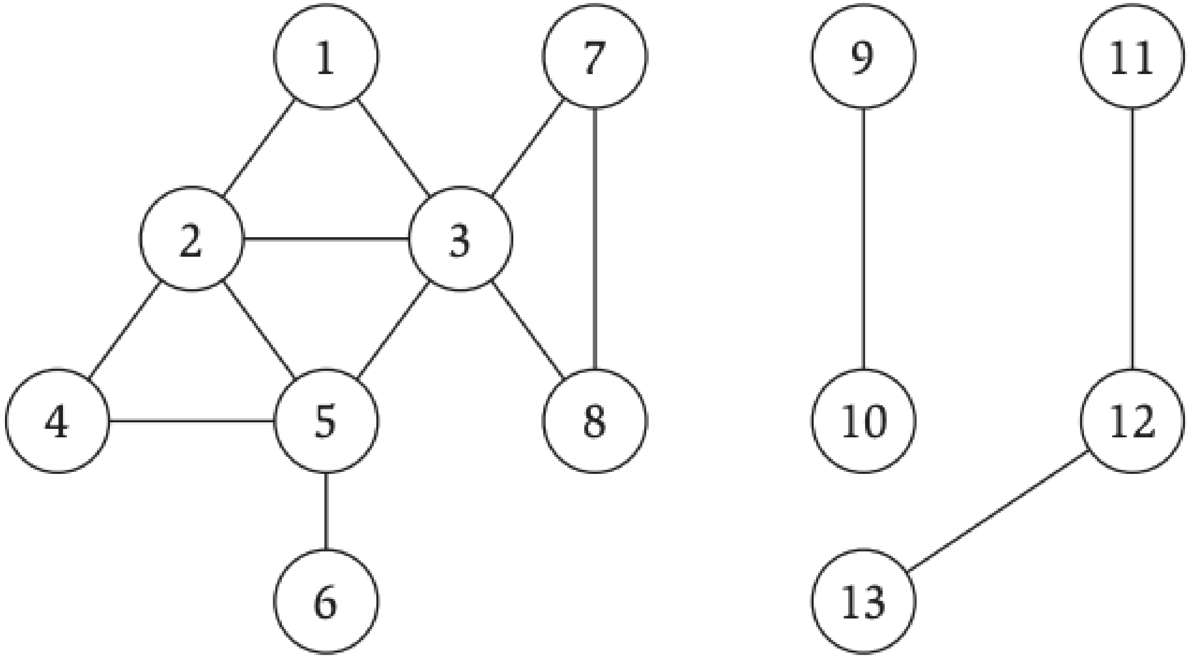


Original Graph



Search Tree

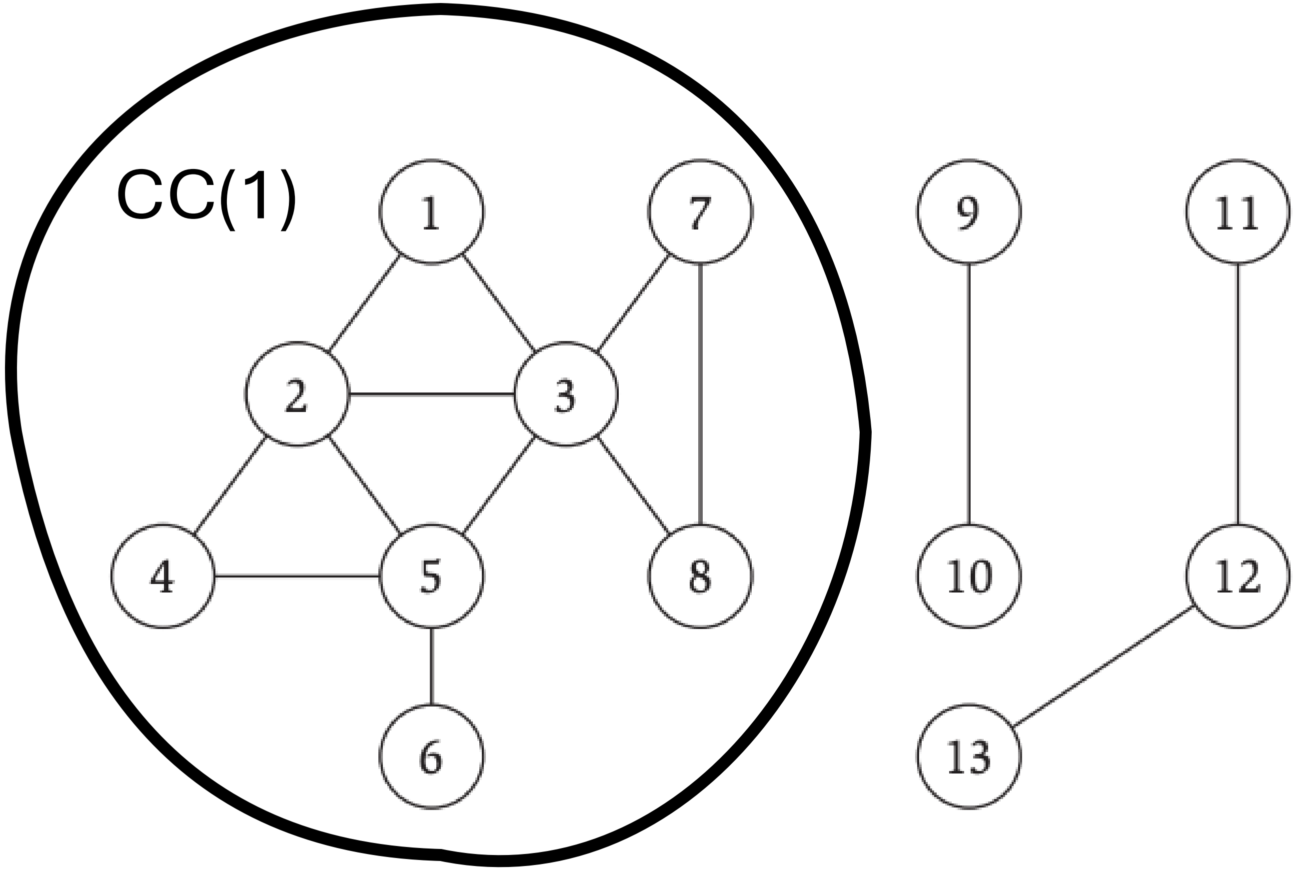
Connected Component of (s) (CC(s))



The CC(s) is the set of all vertices that are connected to s by a path.

"The set of vertices that you can reach from s using a simple path."

Connected Component of (s) (CC(s))



The CC(s) is the set of all vertices that are connected to s by a path.

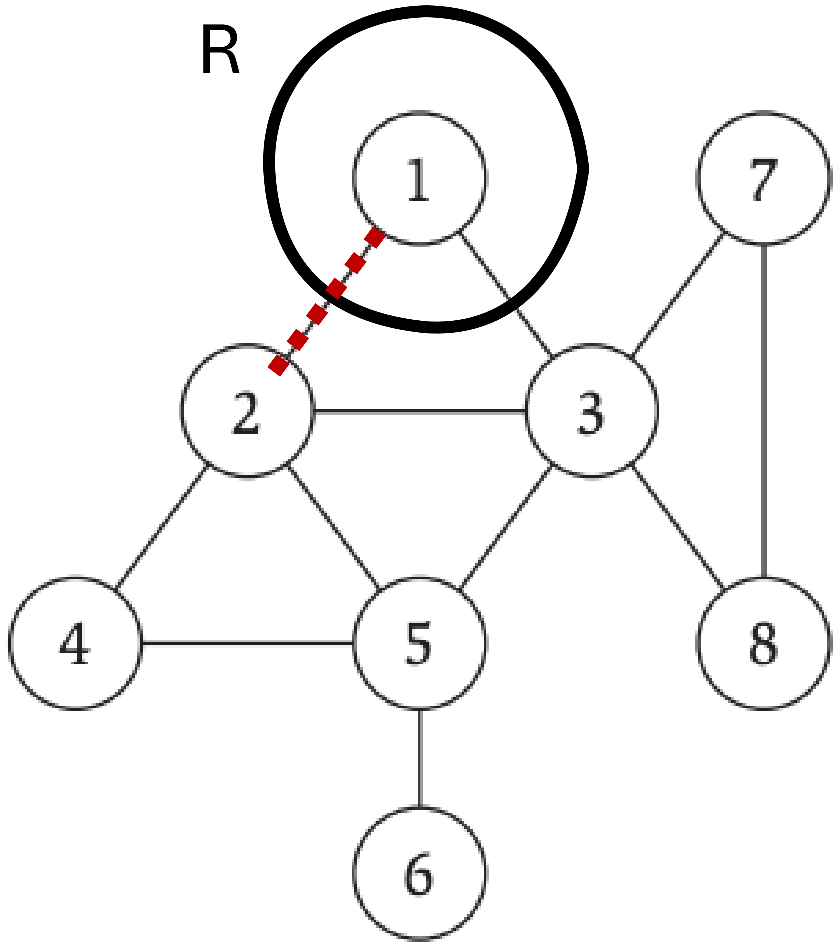
“The set of vertices that you can reach from s using a simple path.”

Explore Algorithm

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{\mathrm{s}\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
    - Add v to R
- Return R

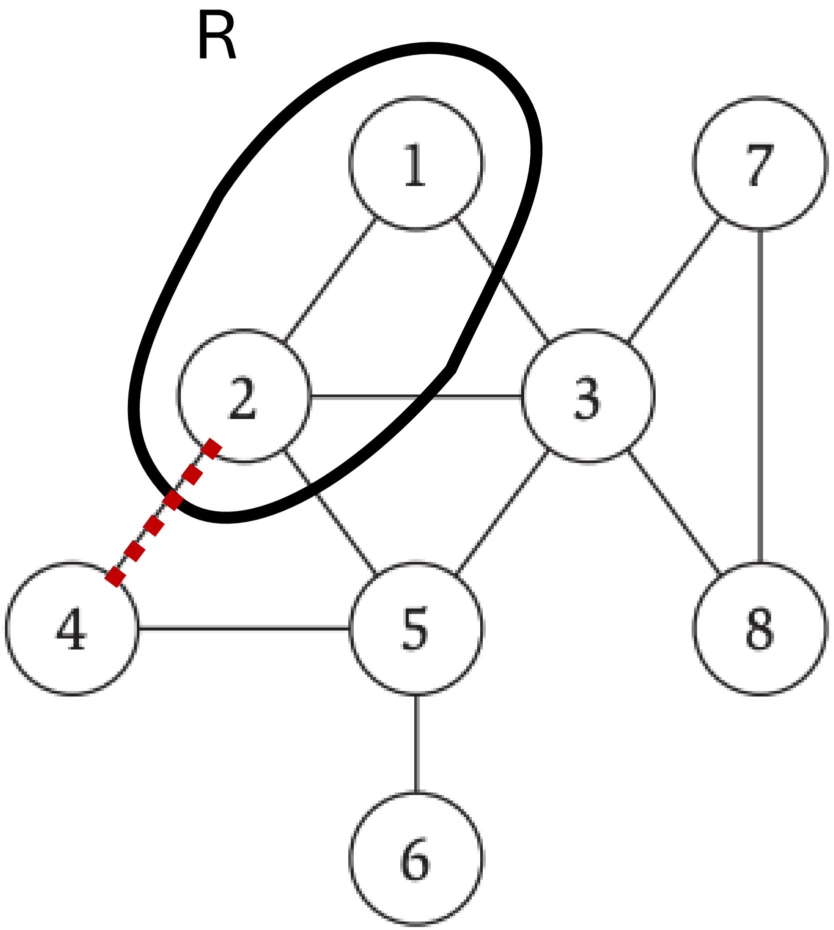
Explore Algorithm

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{\mathrm{s}\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R



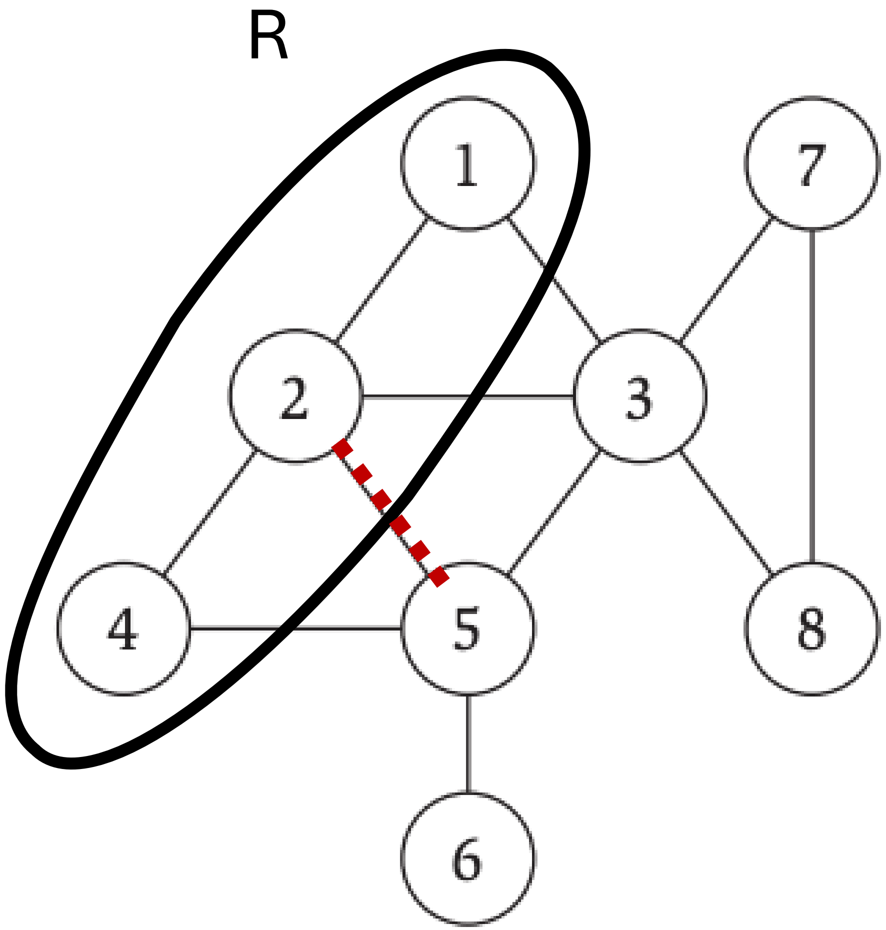
Explore Algorithm

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{\mathrm{s}\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R



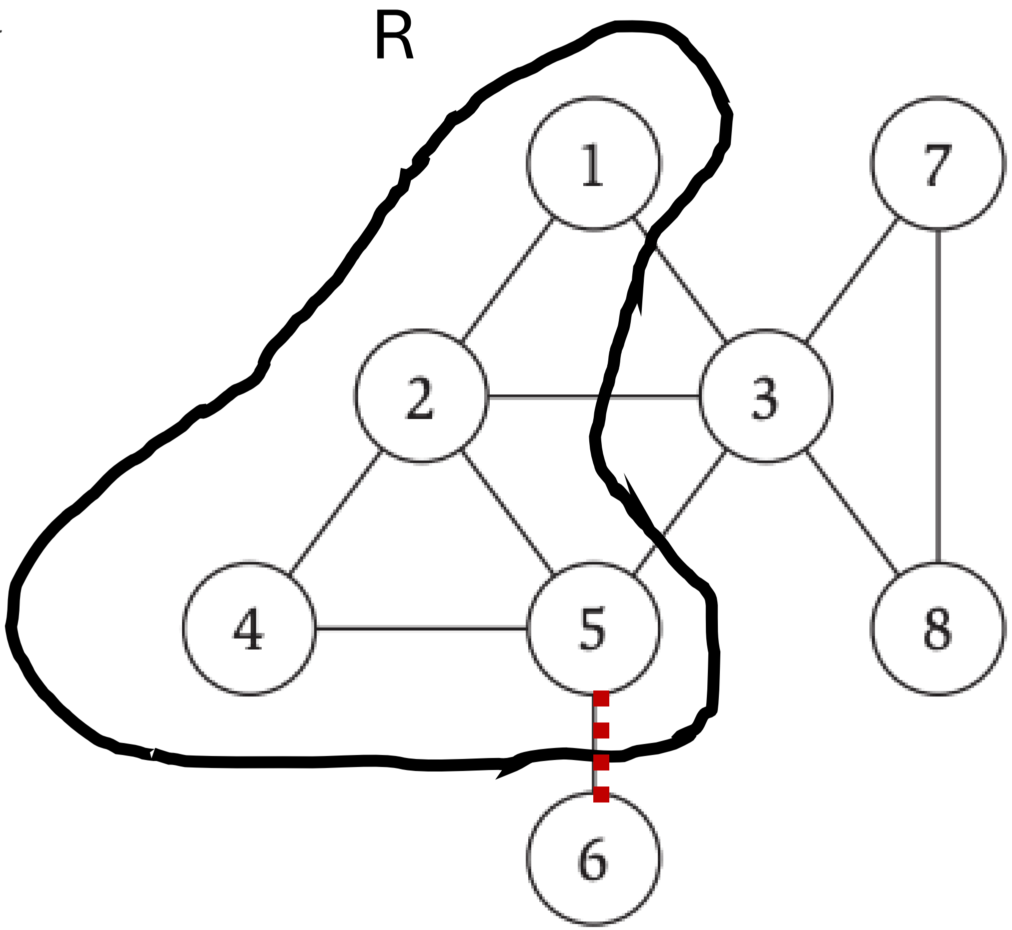
Explore Algorithm

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{\mathrm{s}\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R



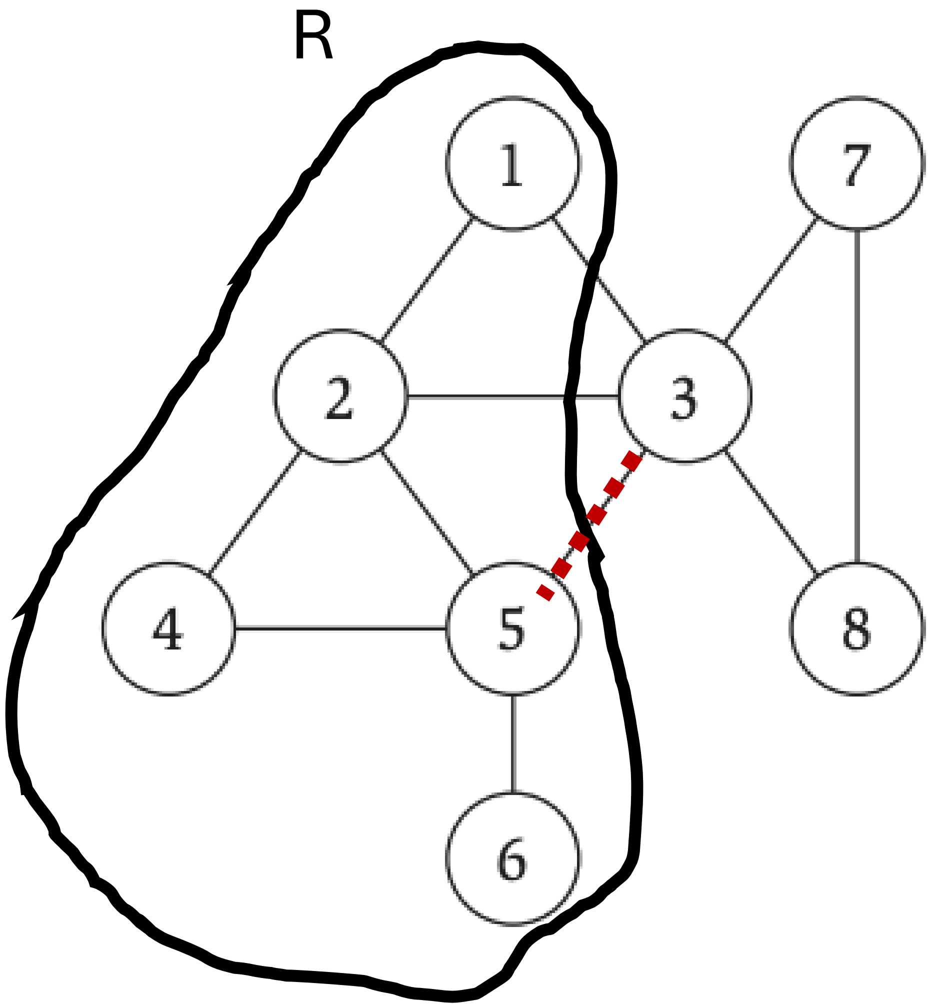
Explore Algorithm

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{\mathrm{s}\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R



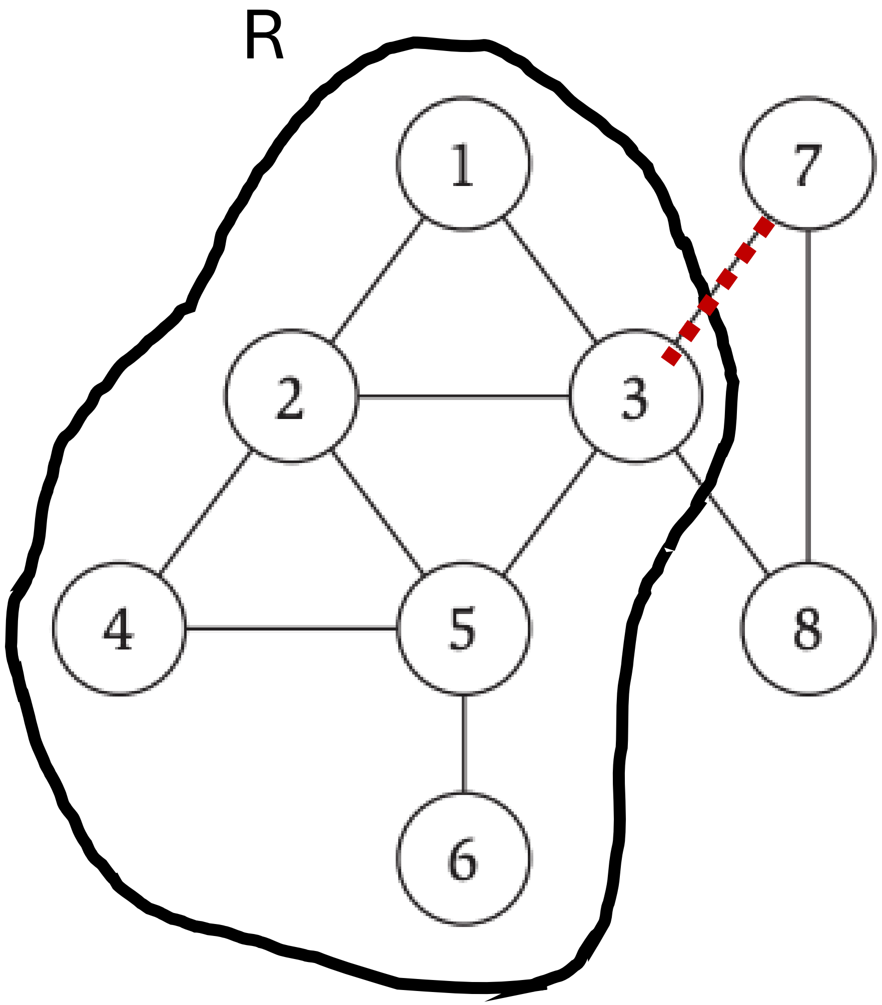
Explore Algorithm

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{\mathrm{s}\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R



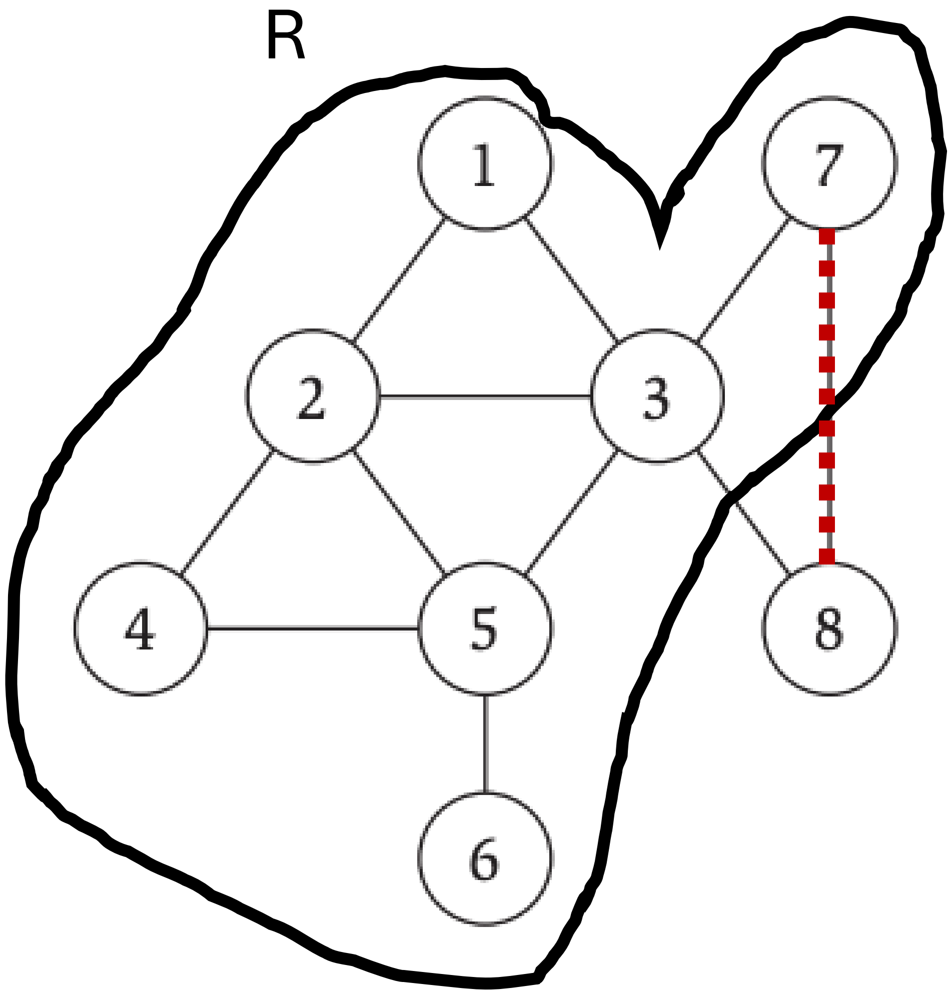
Explore Algorithm

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{\mathrm{s}\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R



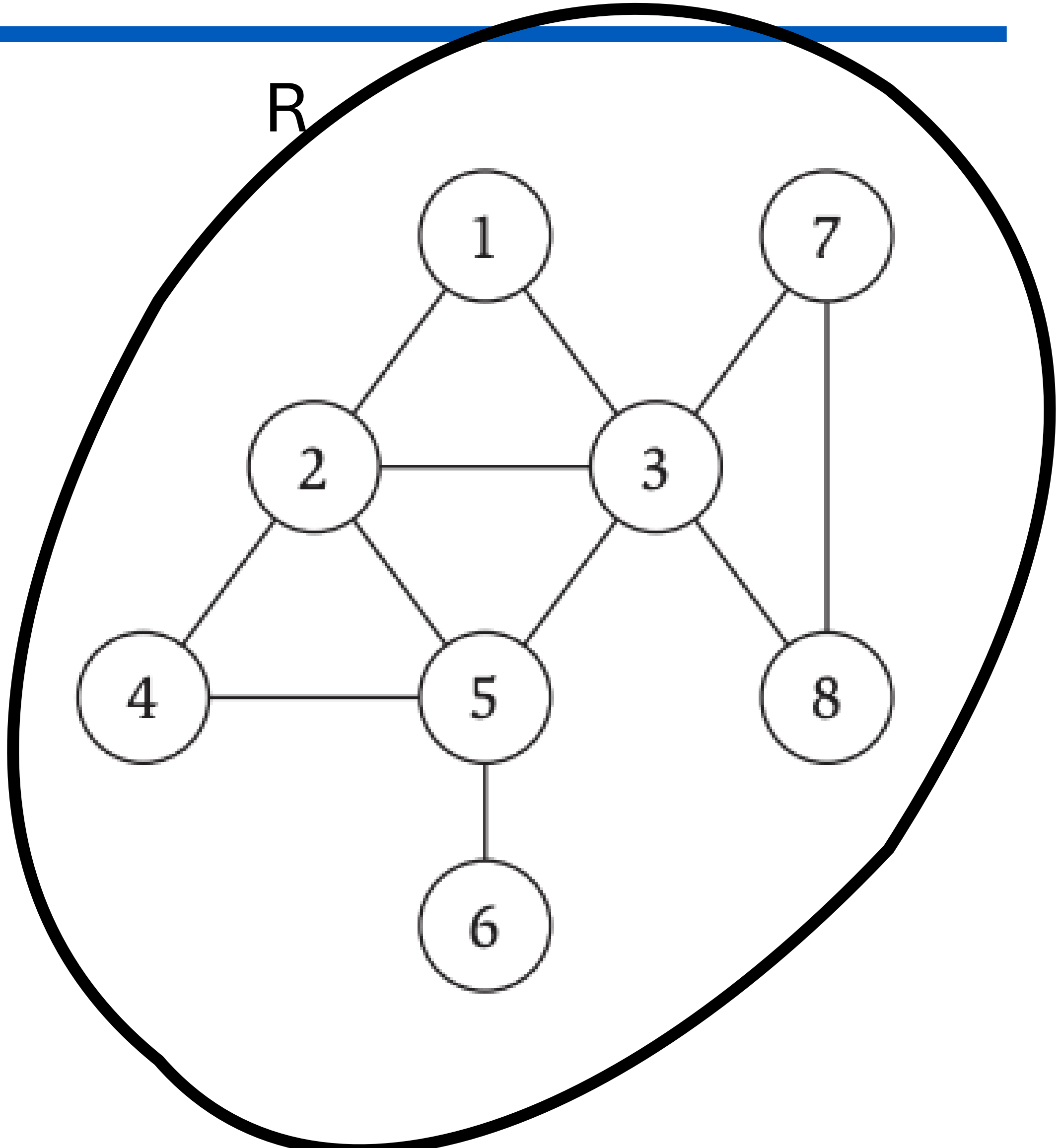
Explore Algorithm

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{\mathrm{s}\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R



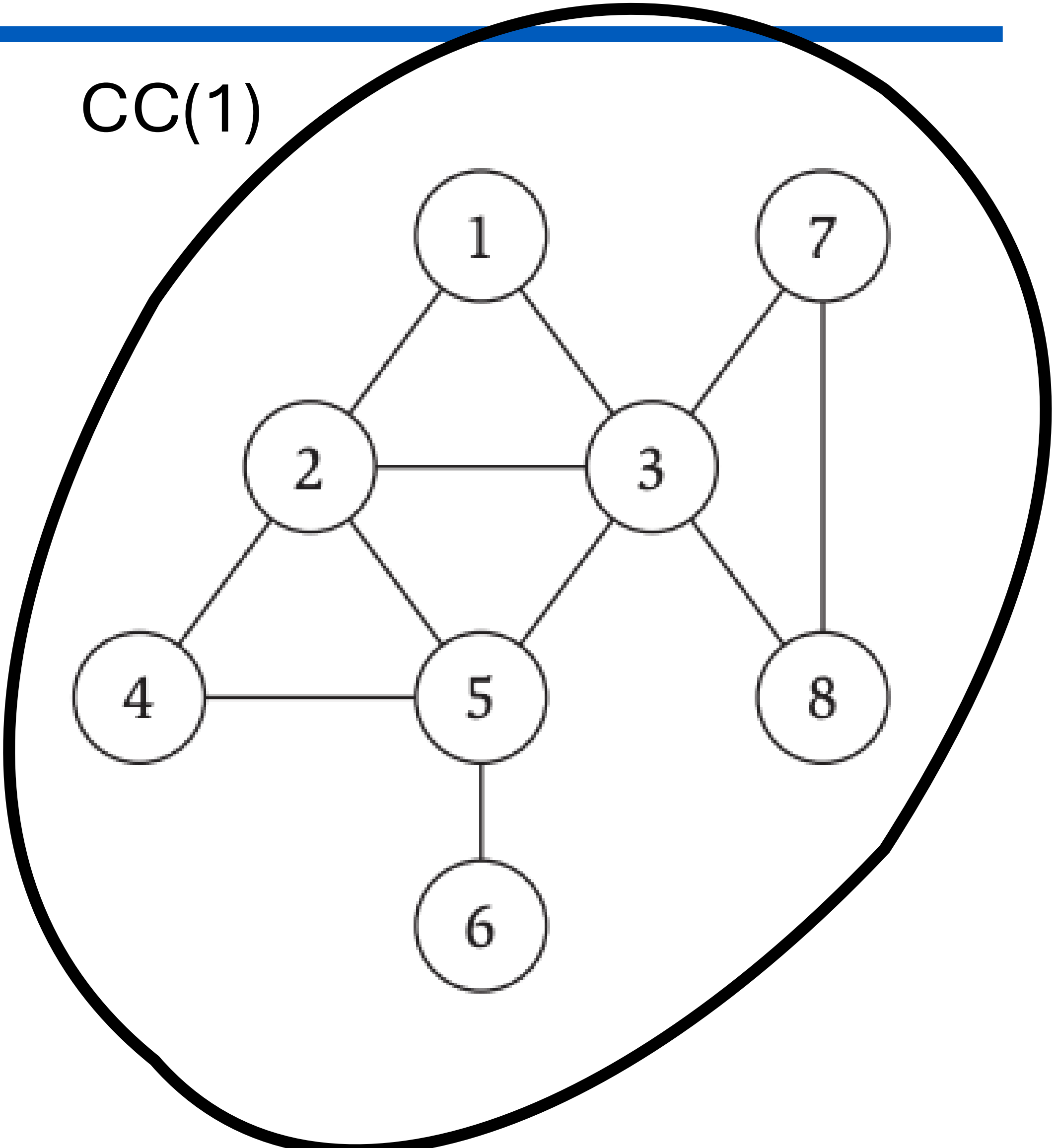
Explore Algorithm

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{\mathrm{s}\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R



Explore Algorithm

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{\mathrm{s}\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R



Q: What is the difference? (BFS vs Explore)

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{s\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R

- Input: $G = (V, E)$ and $s \in V$
- Let $L_0 = \{\mathrm{s}\}$
- Assume $L_0, \dots, L_i$ have been constructed:
  - Let $L_{\mathrm{i}+1}$ be nodes do not appear in $L_0, \dots, L_i$ and have an edge to L_i.
  - If $L_{\mathrm{i}+1}$ is empty, stop.
- Return all layers.

Q: What is the difference? (BFS vs Explore)

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{\mathrm{s}\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R

BFS is Explore but Explore isn't necessarily BFS!

- Input: $G = (V, E)$ and $s \in V$
- Let $L_0 = \{\mathrm{s}\}$
- Assume $L_0, \ldots, L_i$ have been constructed:
  - Let $L_{\mathrm{i+1}}$ be nodes do not appear in $L_0, \ldots, L_i$ and have an edge to L_i.
  - If $L_{\mathrm{i+1}}$ is empty, stop.
- Return all layers.

Q: How do we show this solves Connectivity?

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{s\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
    - Add v to R
- Return R

Q: How do we show this solves Connectivity?

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{s\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R

- Argue that R = CC(s)!
  - Q: How do we do this?

Q: How do we show this solves Connectivity?

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{s\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R

- Argue that R = CC(s)!
  - Show $\mathrm{R} \subseteq \mathrm{CC}(s)$
  - Show $\mathrm{CC}(s) \subseteq \mathrm{R}$

Breadth First Search (Properties)

Claim: $\mathrm{R} \subseteq \mathrm{CC}(s)$

Proof Idea:

- This wants us to show that everything reached by Explore is in the connected component of s.
- Let's do induction on iteration of the algorithm.
  - Do you believe the first iteration.
  - Given any iteration is true, how do you feel about the next iteration?

Q: Does this always terminate?

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{s\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
  - Add v to R
- Return R

- Well, we must be adding a vertex in each iteration and there are only so many vertices, right?

Q: Does this always terminate?

- Input: $G = (V, E)$ and $s \in V$
- Output: CC(s)
- Let $\mathrm{R} = \{s\}$
- While there exists $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$ such that $\mathrm{u} \in \mathrm{R}$ and $v \notin R$:
    - Add v to R
- Return R

Explore Proofs

Claim: $\mathrm{CC}(s) \subseteq \mathrm{R}$

Proof:

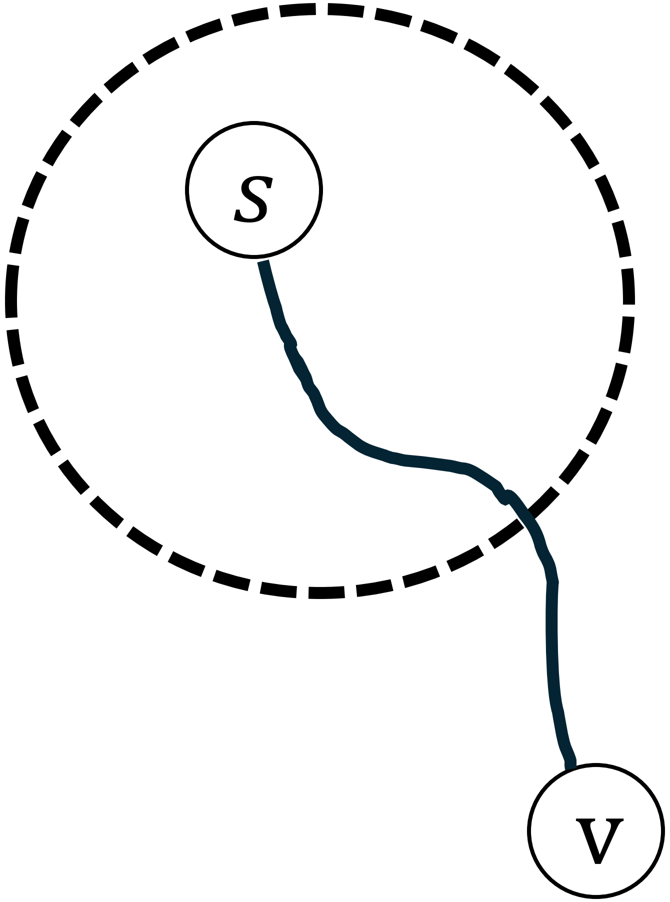
- This saying that every vertex in the connected component is added to R by Explore.
- This is saying that for every vertex v such that there is a path from s to v, v is added to R by Explore.

Explore Proofs

Claim: $\mathrm{CC}(s) \subseteq \mathrm{R}$

Proof:

- Suppose to the contrary that there exists $\mathrm{v} \in \mathrm{CC}(s)$ such that $\mathrm{v} \notin \mathrm{R}$.
  - Then there must exist a path that starts at s (inside R) and ends at v (outside R).

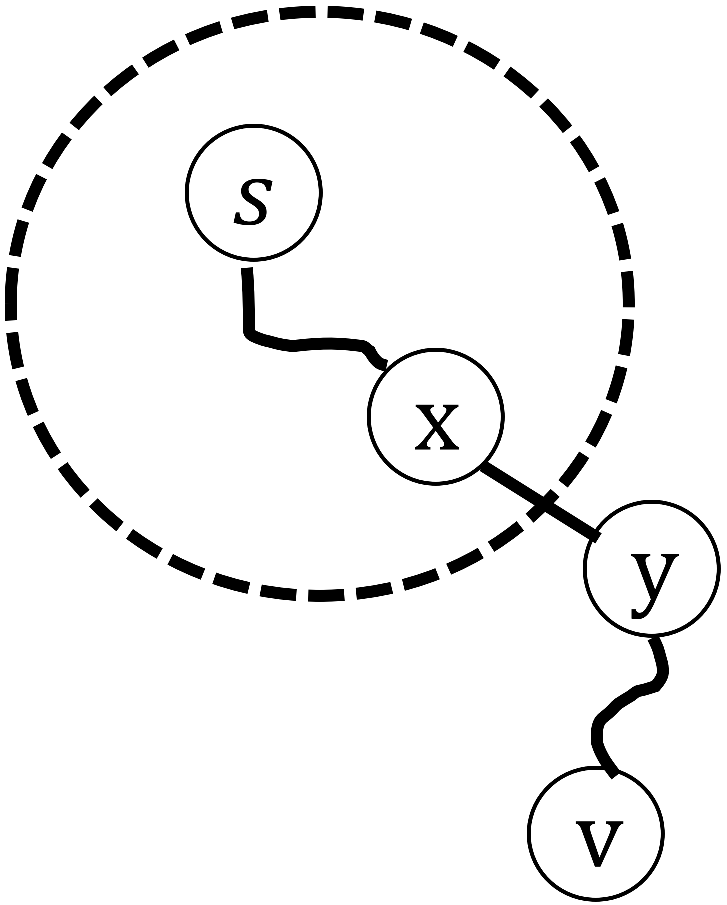


Explore Proofs

Claim: $\mathrm{CC}(s) \subseteq \mathrm{R}$

Proof:

- There then must exist $\{x, y\} \in E$ such that $x \in R$ and $y \notin R$.

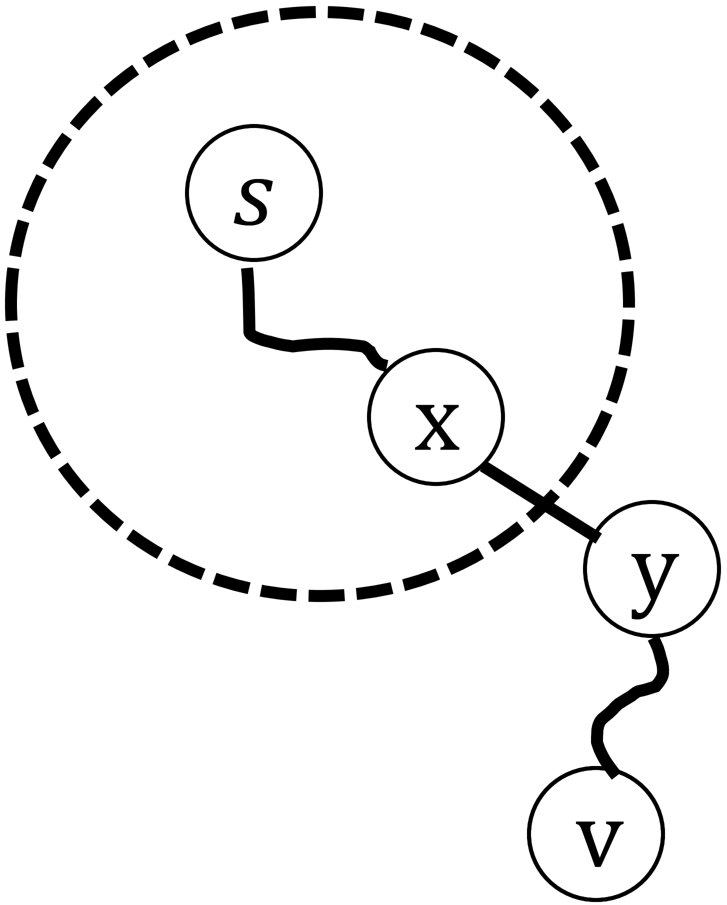


Explore Proofs

Claim: $\mathrm{CC(s)} \subseteq \mathrm{R}$

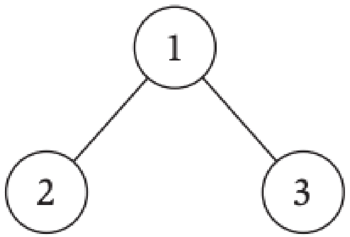
Proof:

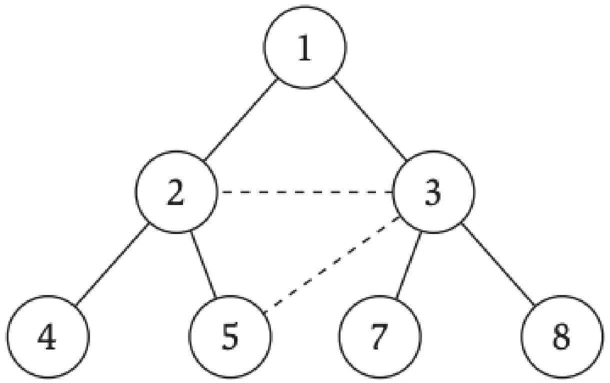
- There then must exist $\{\mathrm{x}, \mathrm{y}\} \in \mathrm{E}$ such that $\mathrm{x} \in \mathrm{R}$ and $\mathrm{y} \notin \mathrm{R}$.
- If this the case, then the algorithm wouldn’t have terminated and would have instead added y. =><=

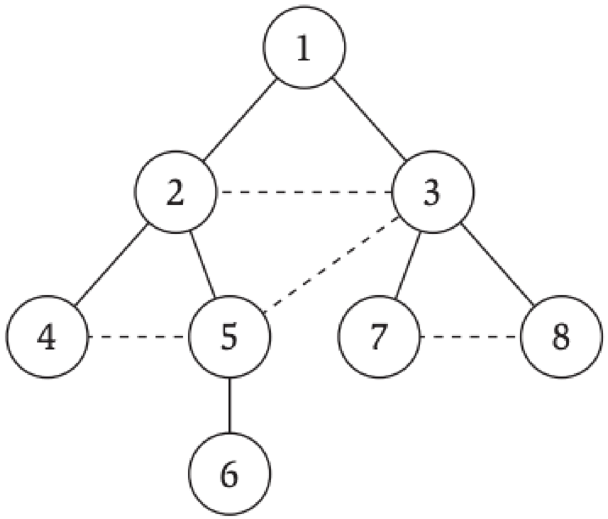


Q: How would you describe BFS?

- $L_0 = s$
- $L_1 =$ neighbors of L_0.
- $L_2 =$ neighbors of L_1 that are not in L_0.
- $L_\mathrm{i} =$ neighbors of $L_{\mathrm{i}-1}$ that are not in previous layer.







Depth First Search

- **Input**: The current vertex $\mathrm{u} \ \in V$
- **Global**: An array of exploration $A \in \{0,1\}^V$
- Mark current vertex as explored ($\mathrm{A[u]} \ = \ 1$).
- For each $\{\mathrm{u}, \mathrm{v}\} \ \in \mathrm{E}$:
  - If v is not explored ($\mathrm{A[v]} \ == \ 0$):
    - DFS(v,A)

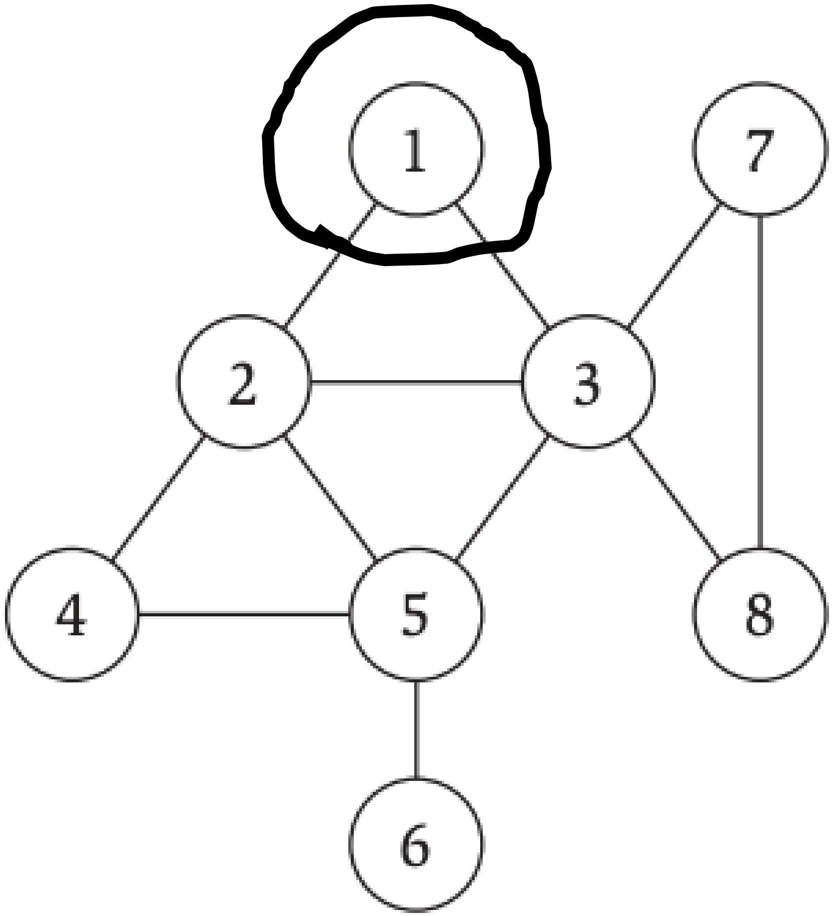
Depth First Search

- **Input**: The current vertex $\mathrm{u} \in V$
- **Global**: An array of exploration $A \in \{0,1\}^V$
- Mark current vertex as explored ($\mathrm{A[u]} = 1$).
- For each $\{\mathrm{u}, \mathrm{v}\} \in \mathrm{E}$:
  - If v is not explored ($\mathrm{A[v]} == 0$):
    - DFS(v,A)

- **Idea**: You are recursing or “drilling down”. If you get stuck, you go up a step and try the next choice.

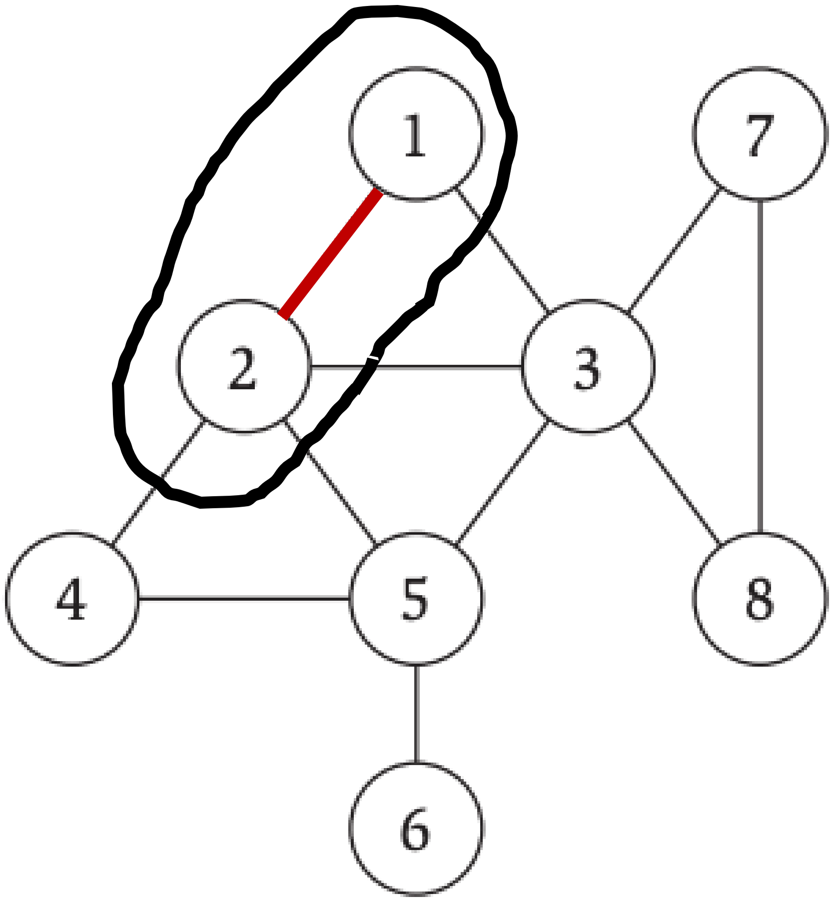
Depth First Search

A = {1}



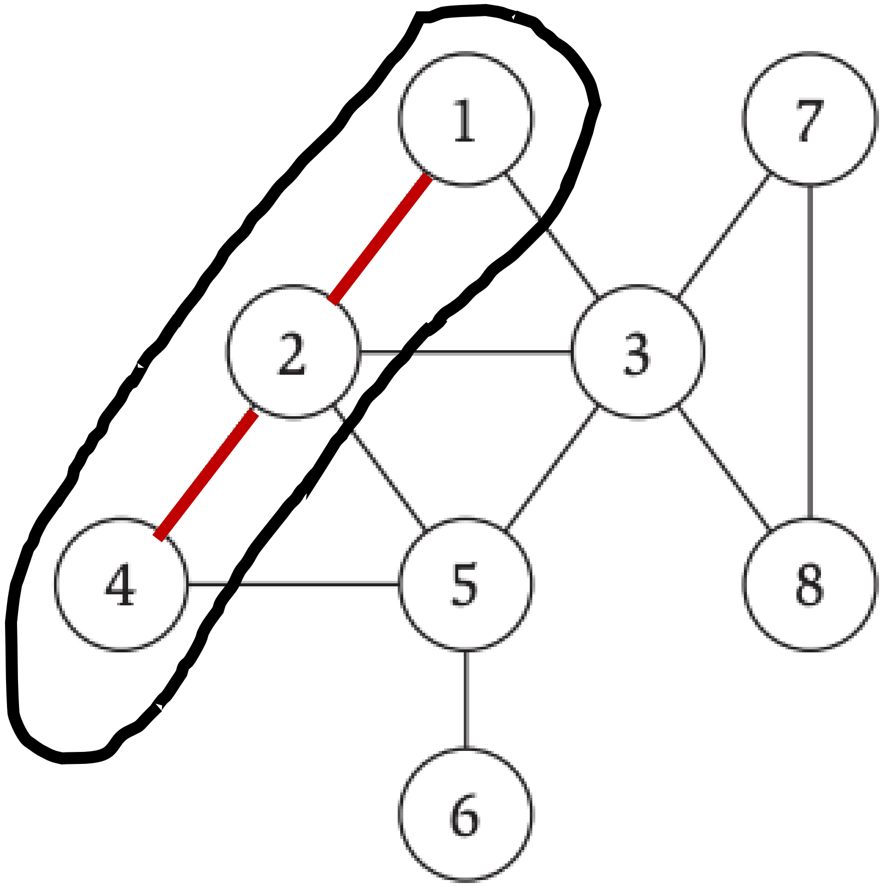
Depth First Search

A = {1,2}



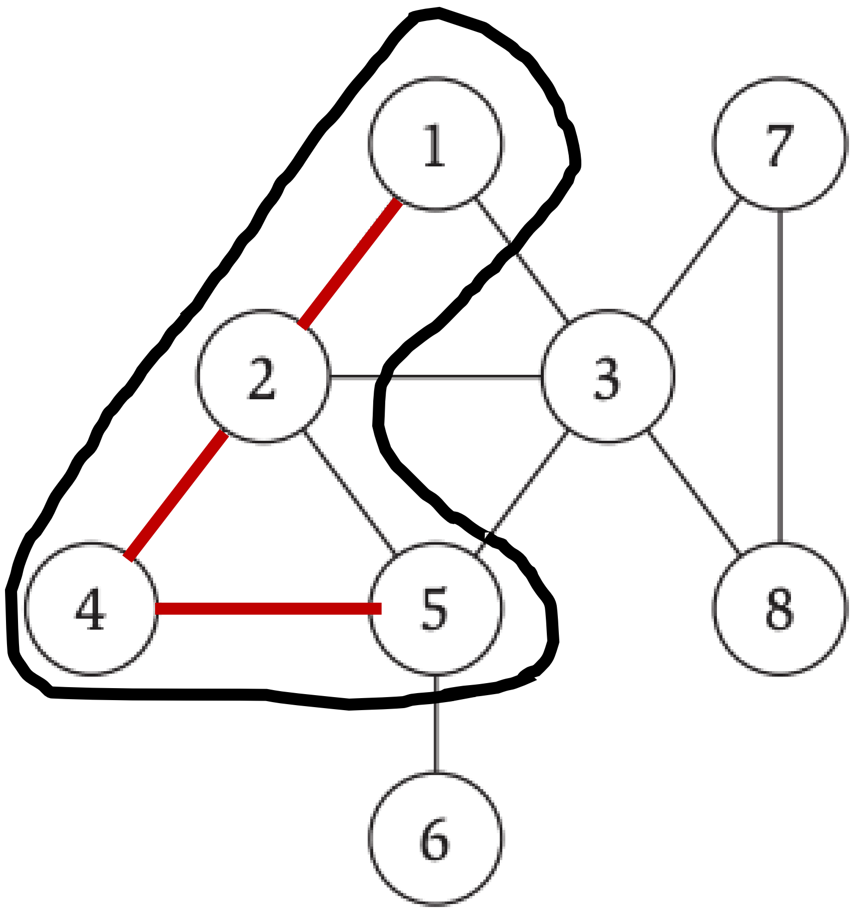
Depth First Search

A = {1,2,4}



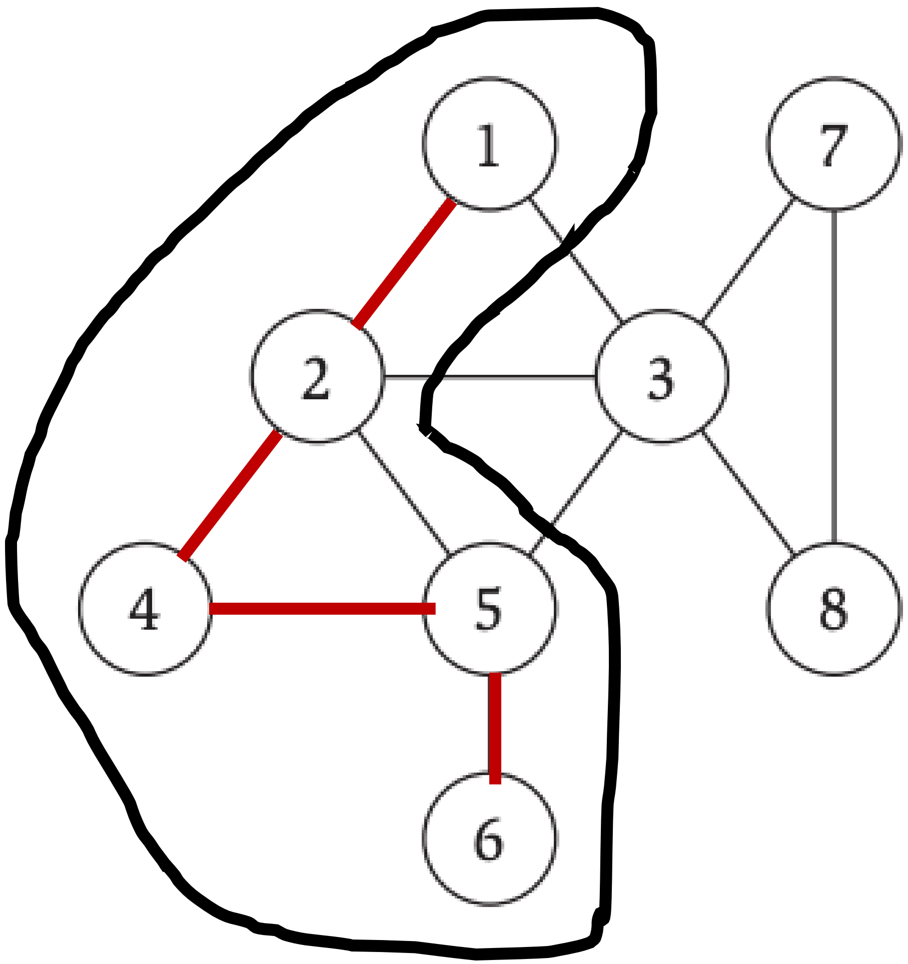
Depth First Search

A = {1,2,4,5}



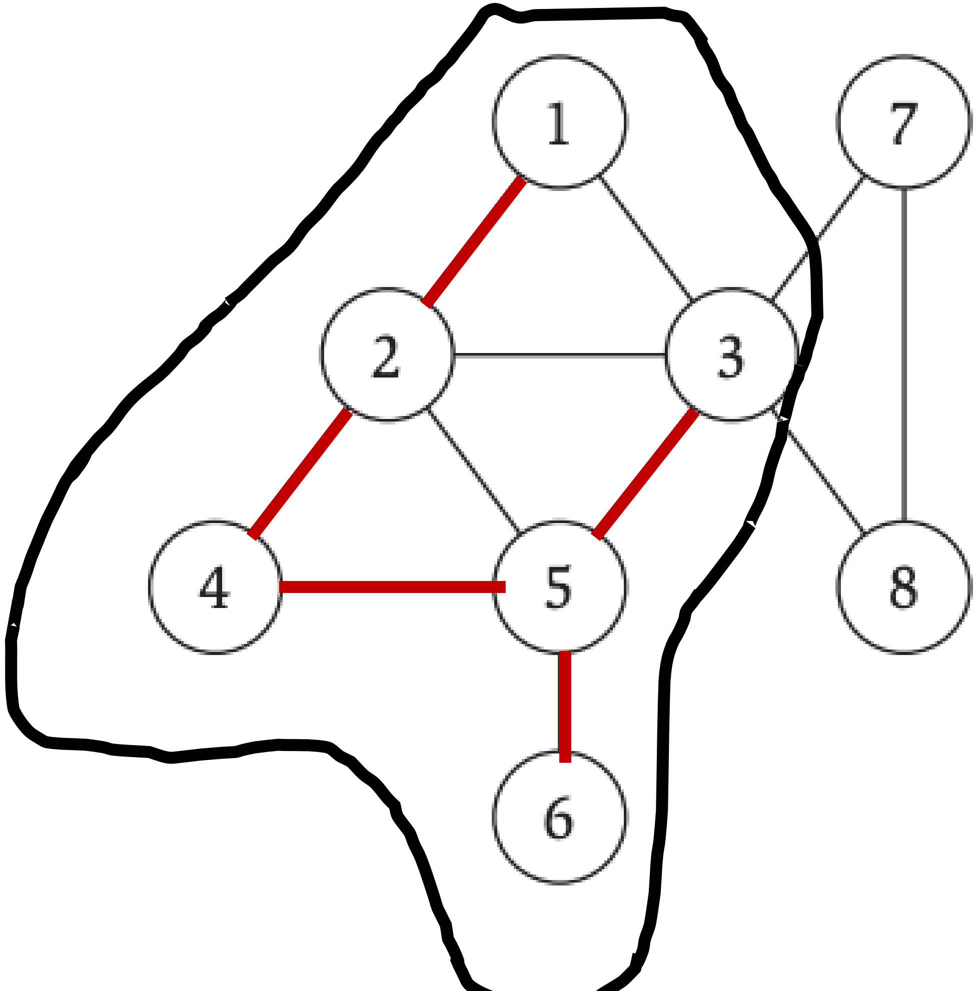
Depth First Search

A = {1,2,4,5,6}



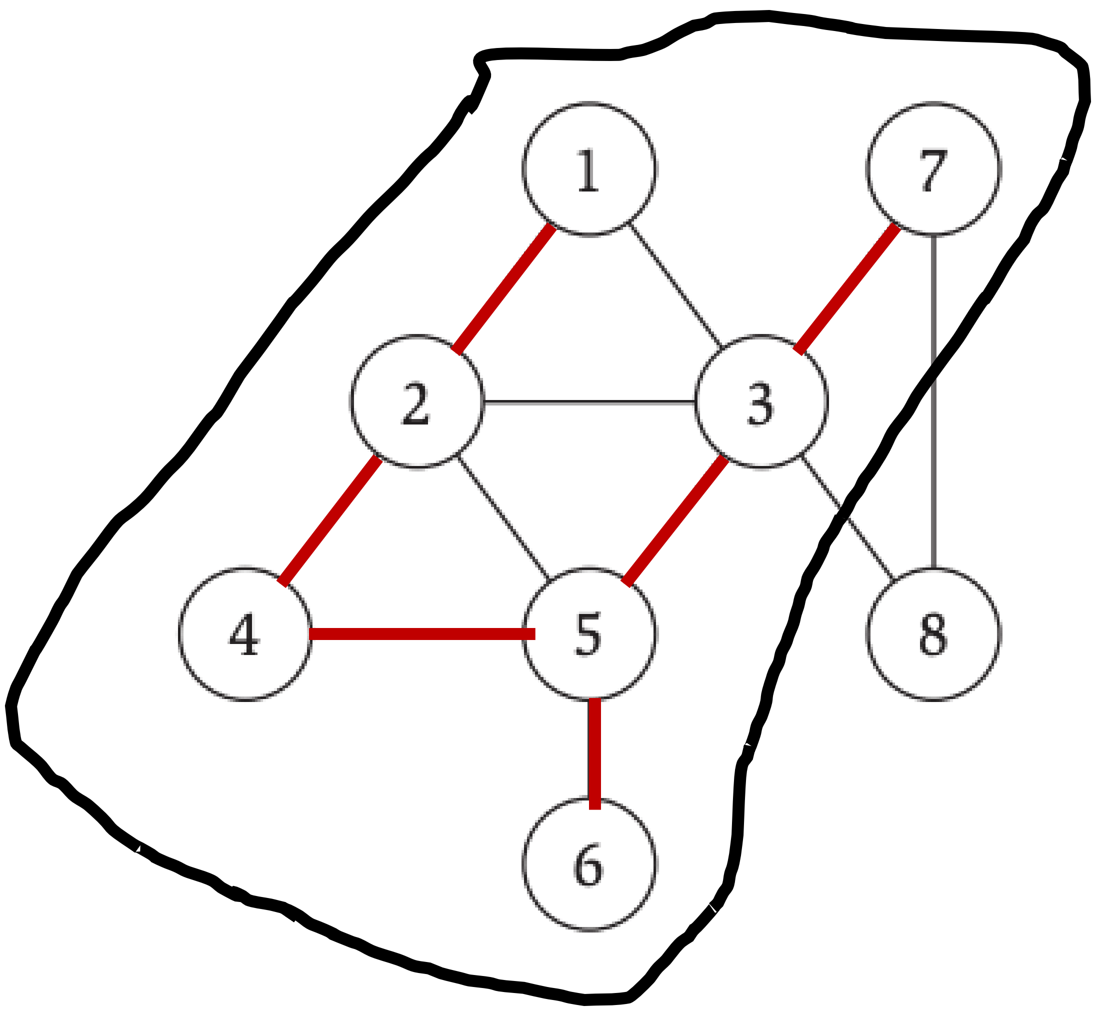
Depth First Search

A = {1,2,4,5,6,3}



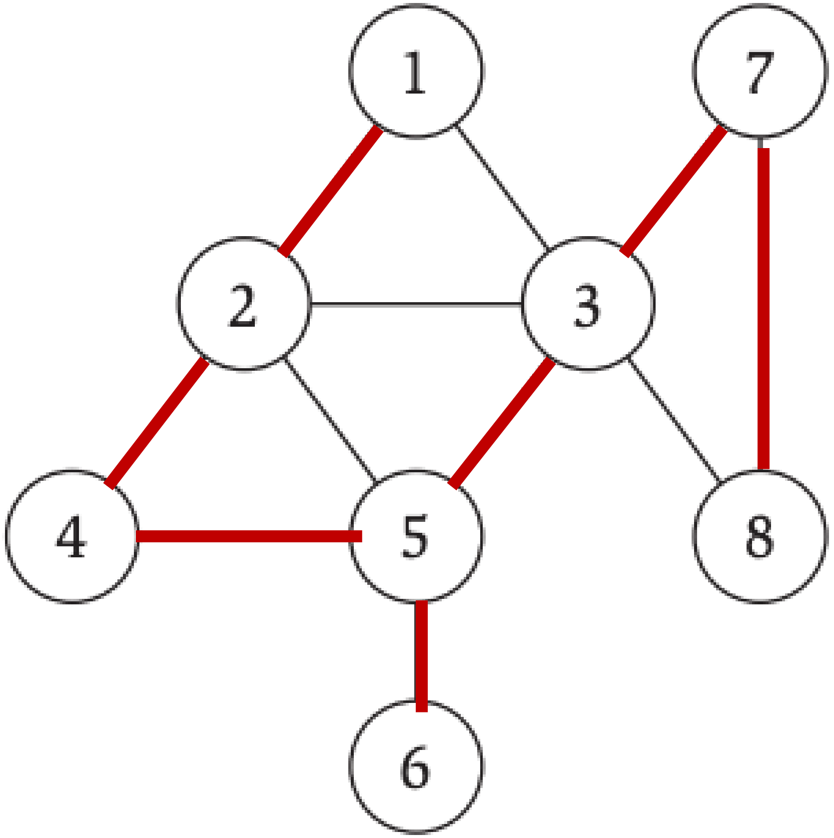
Depth First Search

A = {1,2,4,5,6,3,7}

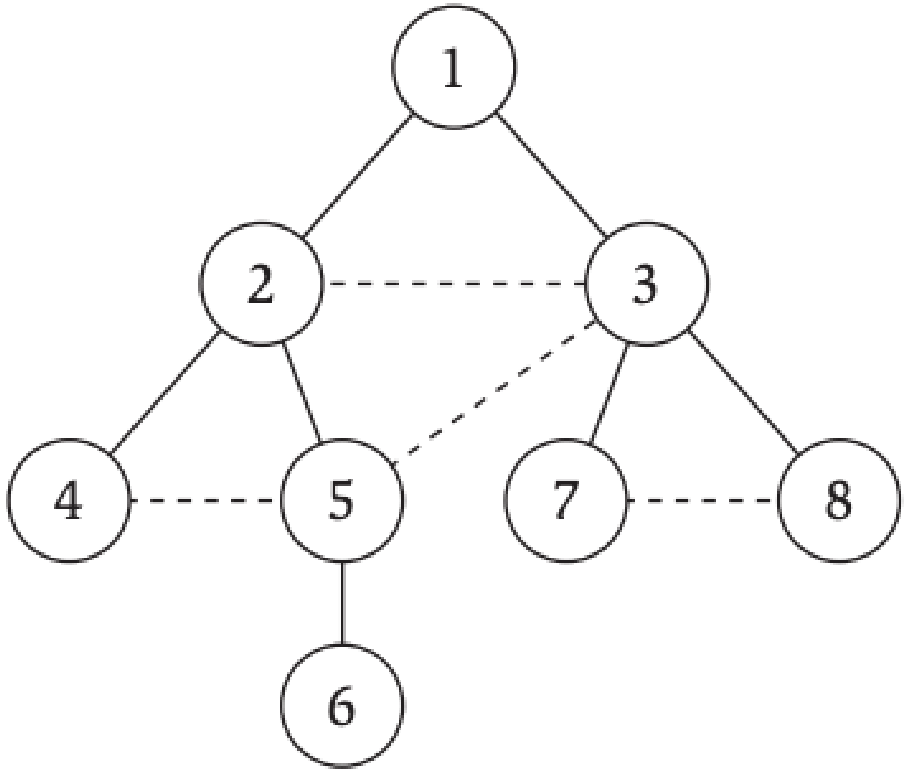


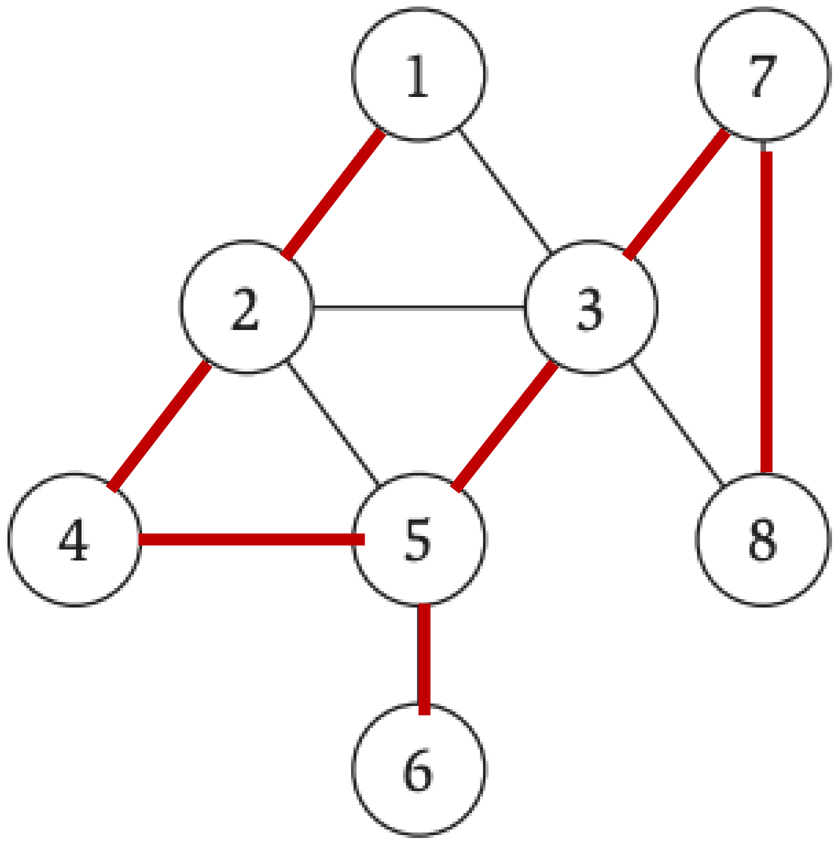
Depth First Search

A = {1,2,4,5,6,3,7,8}



DFS Trees vs BFS Trees





Q: How can you compute all the connected components of a graph?