

CSCI 256 Lecture 6:

Depth-First Search

Shikha Singh

Announcements

- No exercise to hand in, Pick up **Exercise #5**
- No late exercises: they are scanned in right after lecture
 - If you didn't pick up an exercise, can print it from lecture page
 - Will drop four lowest at the end
- **Assignment 1 due tonight at 10 pm**
 - Office hours 1.30-3 pm in CS common room/TCL 304
 - TA hours: 7-9.30 pm in TCL 206
- Graded feedback returned on Assignment 0, Exercise 1+2
- Assignment 2 partners are now available on GLOW

Last Time and Today

- Discussed graph data structures: adjacency list and matrix
- Discussed BFS
 - BFS computes shortest path from root s to all nodes
 - (Leftover) Implemented using queues in $O(n + m)$ time
- Today: Depth first search

[Leftovers] BFS Implementation

BFS (G, s):

Store G using an adjacency list

for all $v \neq s$: set $L[v] = \infty$

Set $L[s] = 0$

Put s in the **queue** Q

While Q is not empty

 Extract v from Q

 For each edge (v, w) :

 if $L[w] = \infty$ then // w is unmarked

 Put w into the **queue** Q

$L[w] = L[v] + 1$

Why does queue ensure we visit nodes level-by-level?

[Leftovers] BFS Using a Queue

- **Claim.** When BFS is about to ***begin exploring*** a vertex at level L_i , then all vertices in the queue must have level i
- Can prove this by induction

[Leftovers] BFS Using a Queue

- **Claim.** When BFS is about to begin **exploring** a vertex at level L_i , then all vertices in the queue must have level i
- **Proof.** Base case. $i = 0$: queue only contains s
- Suppose this holds until level i : as BFS explores these vertices all their neighbors (with level $i + 1$ are added to the back of the queue)
- Since all nodes with level i are before $i + 1$, they all have be explored before BFS begins exploring any node at level $i + 1$ (FIFO property)

[Leftovers] BFS Analysis

- Inserting and extracting an edge from a queue
 - $O(1)$ time
- For each marked node v , we run the for loop for its edges: $O(n)$
- Overall running time?
 - Easy to prove $O(n^2)$ time
- Can improve the analysis to $O(n + m)$
 - Node u has $\text{degree}(u)$ incident edges (u, v)
 - Total time processing edges: $\sum_{u \in V} \text{degree}(u) = 2m$

each edge (u, v) is counted exactly twice
in sum: once in $\text{degree}(u)$ and once in $\text{degree}(v)$

Depth-First Search

Depth-first Search

- Search "as deep as you can go" starting from the source s
- Visits all nodes reachable from the root s
- Natural recursive interpretation

DFS(u) :

Set status of u to marked # discovered u

for each edge (u, v) :

if v 's status is unmarked:

DFS(v)

done exploring neighbors of u

DFS Tree

- Consists of edges between a node v and the node u that discovered it (referred to as its parent)

Recursive-DFS(u):

Set status of u to marked # discovered u

for each edge (u, v) :

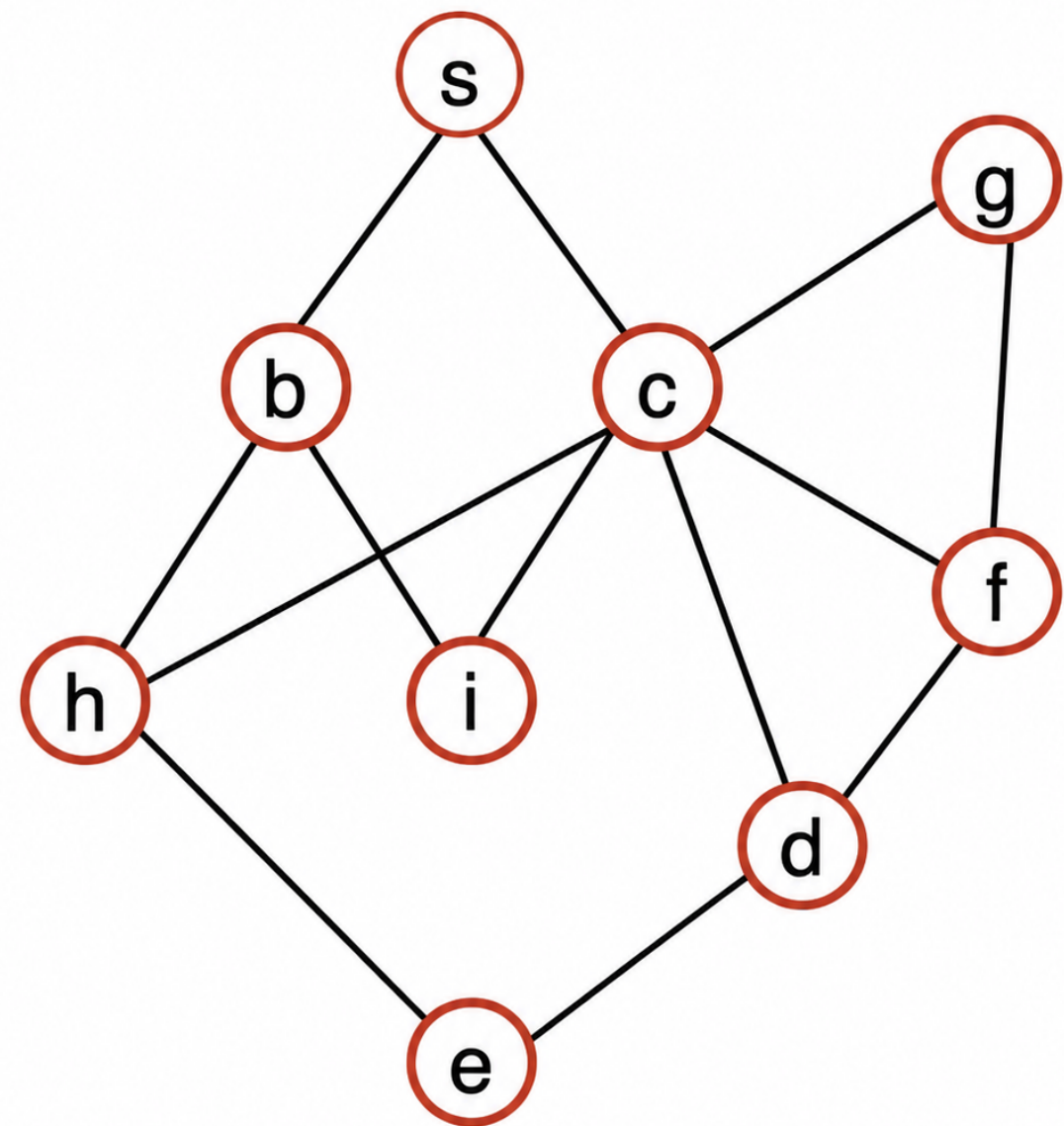
if v 's status is unmarked:

Add (u, v) to T

DFS(v)

done exploring neighbors of u

DFS Example



DFS(u):

Set status of u to marked # discovered u

for each edge (u, v):

if v's status is unmarked:

DFS(v)

done exploring neighbors of u

Assume adjacency list is in alphabetical order

DFS Correctness

- DFS finds precisely the set of **nodes reachable from start node s**
- That is, $\text{DFS}(s)$ marks node x iff node x is reachable from s
- **Proof.** ($\Rightarrow$)
- Since x is marked, $(x, \text{parent}(x))$ is an edge in the graph
- There is a path $x \rightarrow \text{parent}(x) \rightarrow \text{parent}(\text{parent}(x)) \rightarrow \dots$ that leads to s using tree edges
 - Induction on the order in which vertices are marked
 - Suppose claim holds for all vertices before some vertex u
 - $\text{Parent}(u)$ must be discovered before u , and thus the claim holds for it, since $(u, \text{parent}(u))$ is an edge, we have a path from u to s

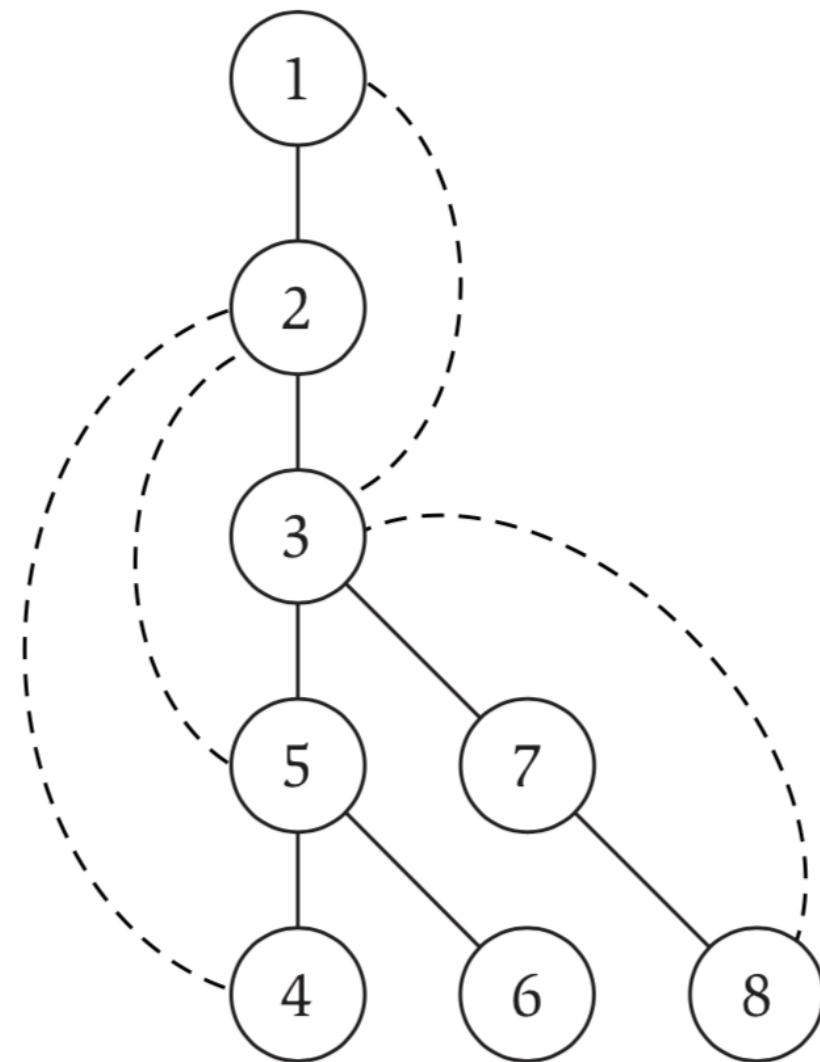
DFS Correctness

- DFS finds precisely the set of **nodes reachable from start node**
- That is, $\text{DFS}(s)$ marks node x iff node x is reachable from s
- **Proof.** ($\Leftarrow$)
- Suppose node x is reachable from s via path P
- Since s is marked by DFS and x is not, there must be a first node v on P that is not marked by DFS
- Thus, there is an edge $(u, v) \in P$ s.t. u is marked and v is not marked
- But this cannot happen, since when u is marked, all its neighbors are also marked

DFS Tree Property

Property. For a given recursive call $\text{DFS}(u)$, all nodes that are marked between the invocation and end of this recursive call are descendants of u in T

- A node y is a descendant of x and x is an ancestor of y if you can reach x from y by following the edges
 $y \rightarrow \text{parent}(y) \rightarrow \text{parent}(\text{parent}(y)) \rightarrow \dots$
- Equivalently if x is on the path from the root to y

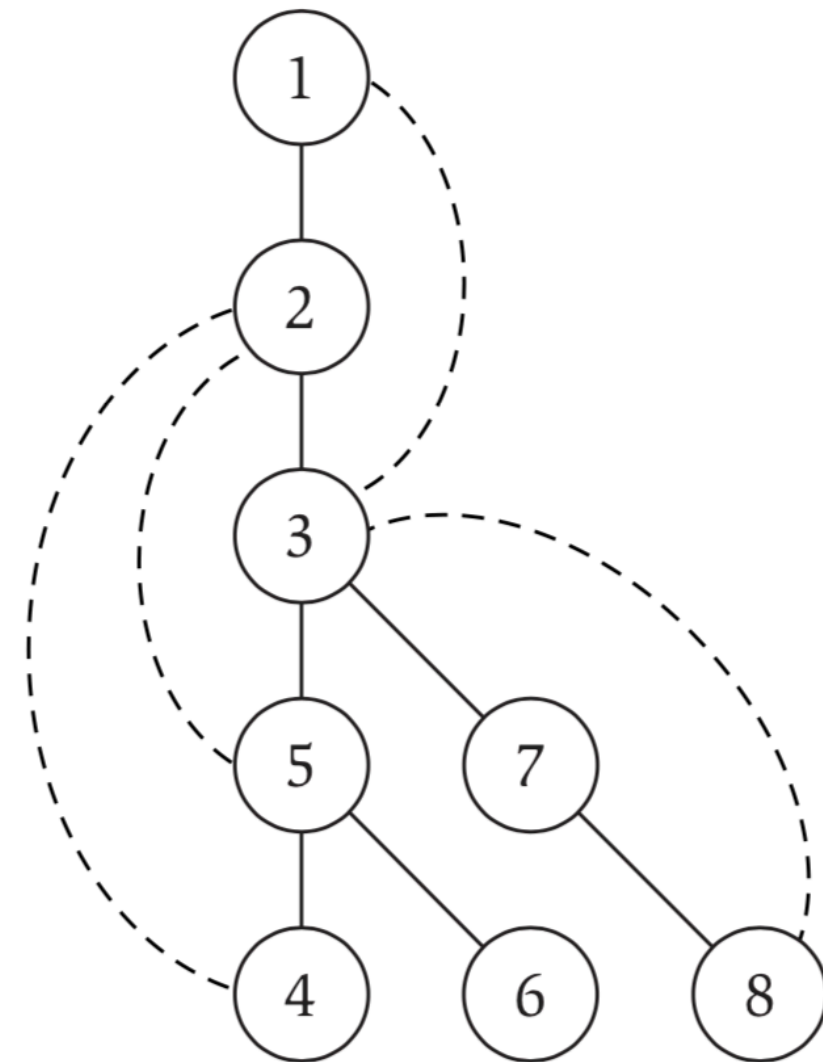


DFS Tree Property

Lemma. For every edge $e = (u, v)$ in G , one of u or v is an ancestor of the other in T .

Proof. (Case I) Suppose e is in T , then why is this true

- u is parent(v) or vice versa

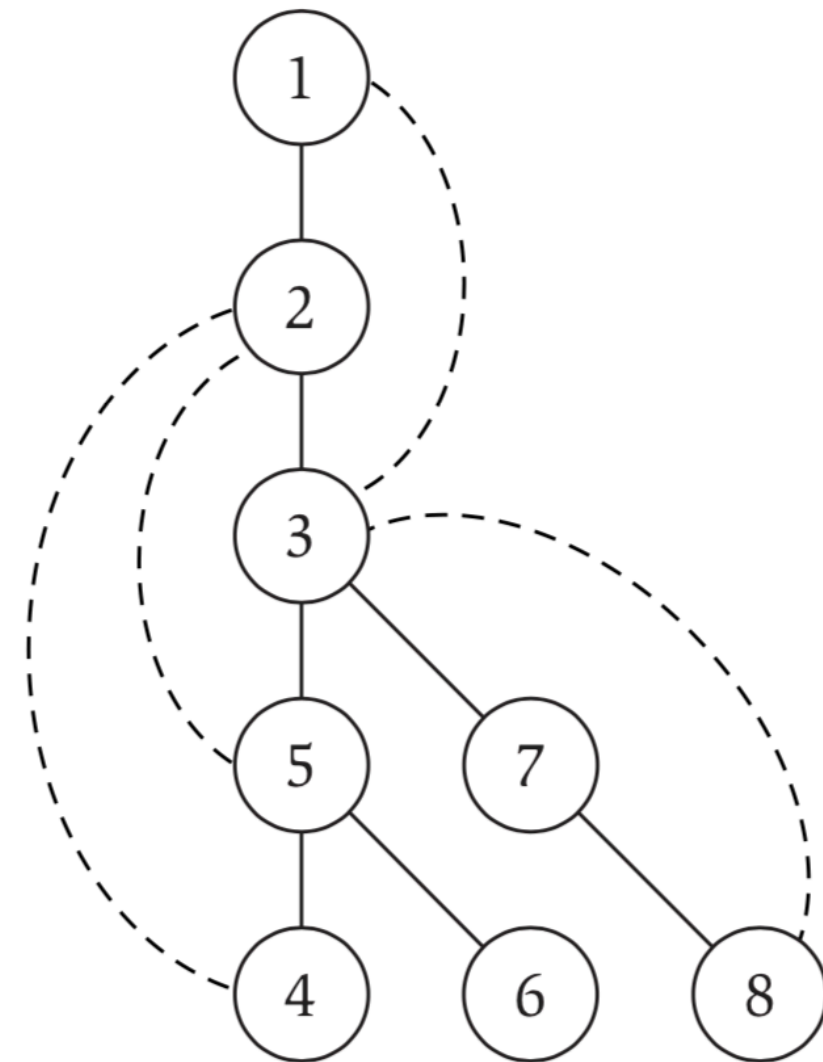


DFS Tree Property

Lemma. For every edge $e = (u, v)$ in G , one of u or v is an ancestor of the other in T .

Proof. (Case 2) Suppose e is not in T . **Without loss of generality**, suppose DFS is called on u before v .

- When the edge (u, v) is inspected **v must have been already marked visited** (why?)
 - Or else $(u, v) \in T$
- Since $(u, v) \notin T$, v is not marked visited during the DFS call on u
- Must have been marked during a recursive call within $\text{DFS}(u)$: thus v is a descendant of u ■



Iterative Implementation: Stack

- We can implement DFS iteratively using a stack

DFS (G, s):

Initialize **stack** S with node s

While S is not empty

 Extract u from S

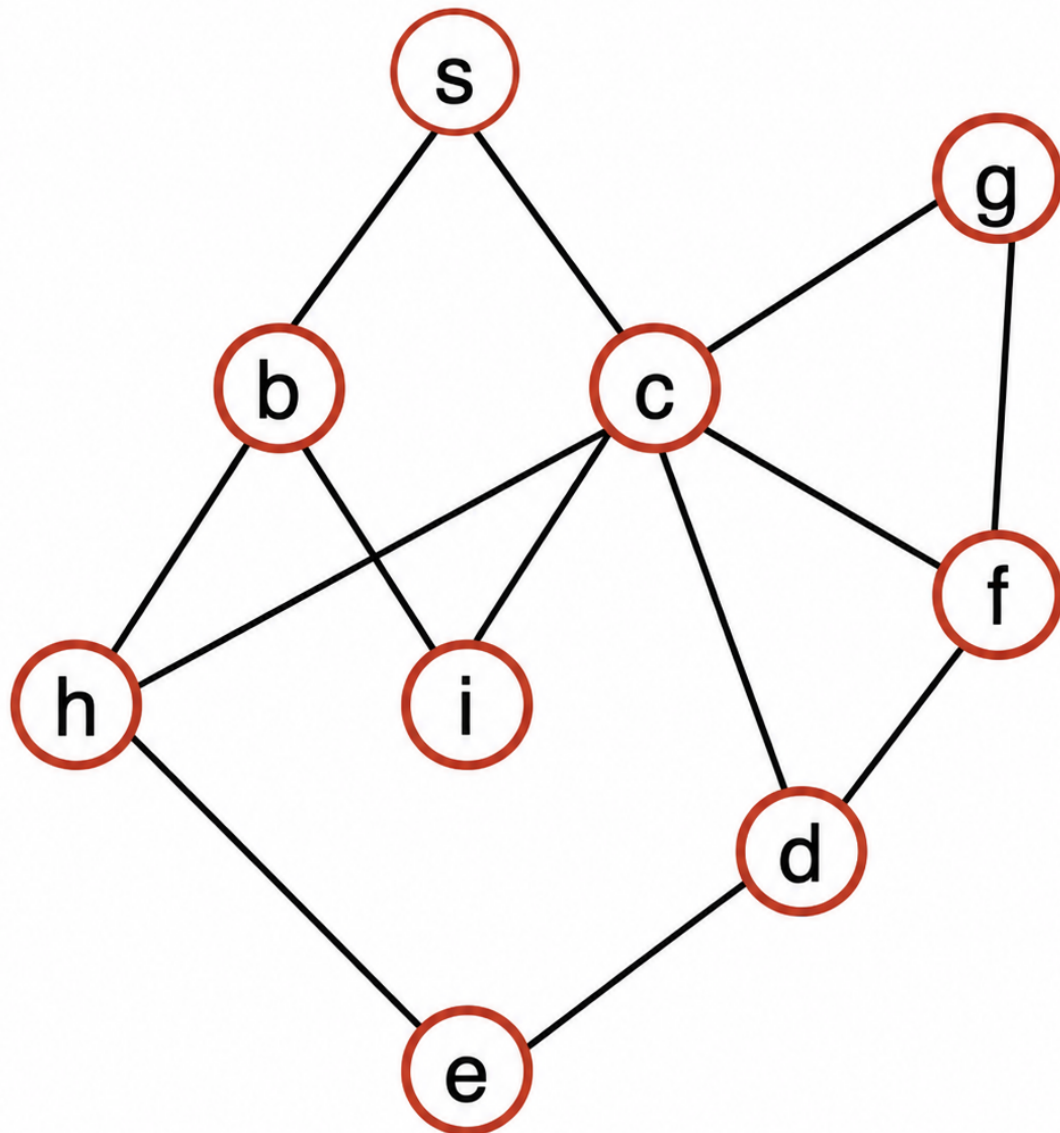
 If u is unmarked: # check marked status on pop

 Mark u

 For each edge (u, v):

 Add v to the **stack** S

DFS Example



DFS (G, s):

Initialize **stack** S with node s

While S is not empty

 Extract u from S

 If u is unmarked: # check marked status

 Mark u

 For each edge (u, v):

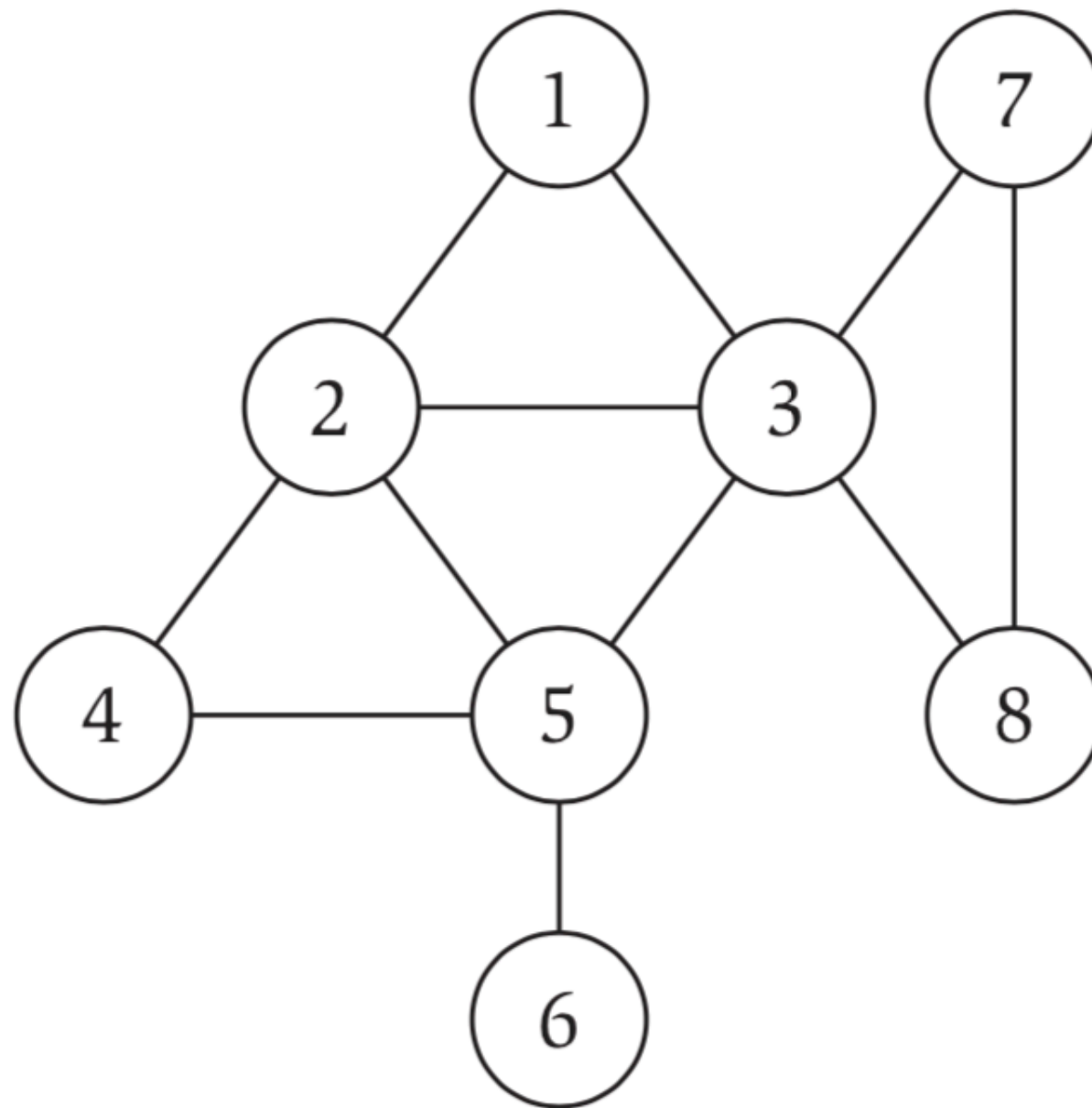
 Add v to the **stack** S

To get same tree as recursive DFS: explore adjacency list is **reverse order**

DFS Running Time

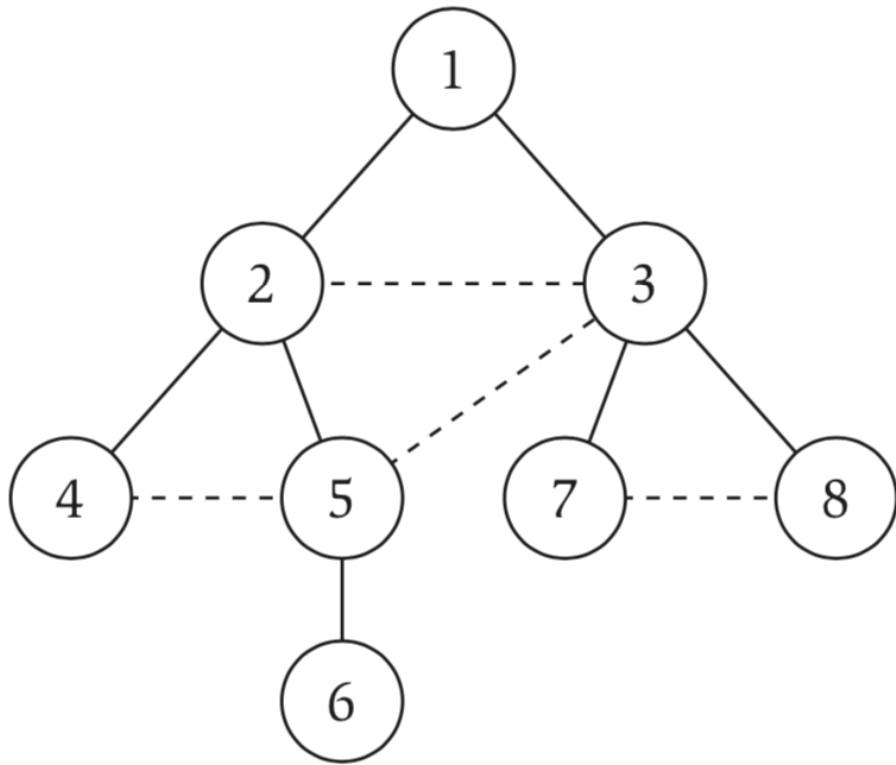
- Same analysis as BFS
- Inserts and extracts to a stack: $O(1)$ time
- Setting status of each node to unmarked: $O(n)$
- Each node is set marked at most once; equivalently DFS(u) is called at most once for each node
- For every node v , explore degree(v) edges
 - $\sum_v \text{degree}(v) = 2m$
- Overall, running time $O(n + m)$

Example Graph



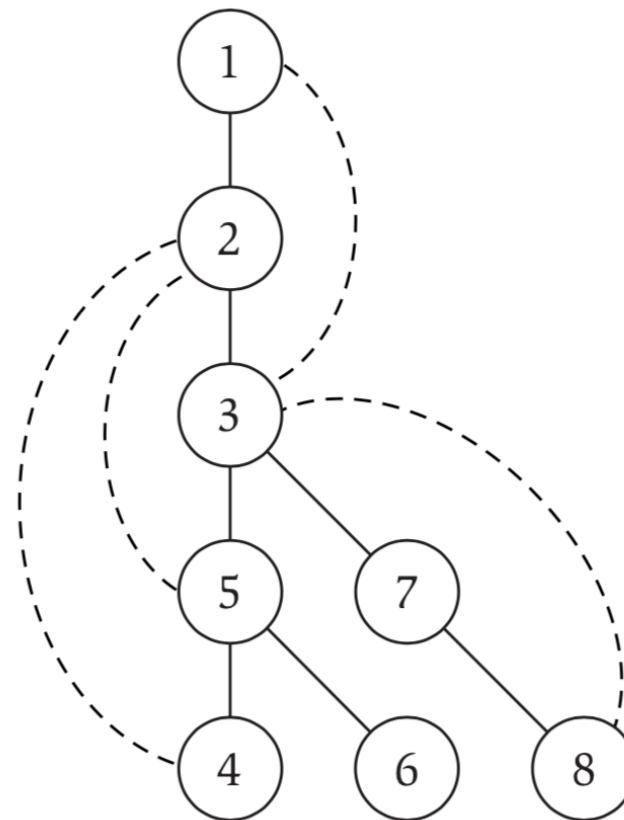
BFS vs DFS Trees

Property. All edges are between nodes at same or adjacent level



BFS tree

Property. All edges are between an ancestor and descendant



DFS tree