Job Interview in the AI-Era: Coding, Systems, Agents

Wei Chen

05/20/2026

Lecture 2

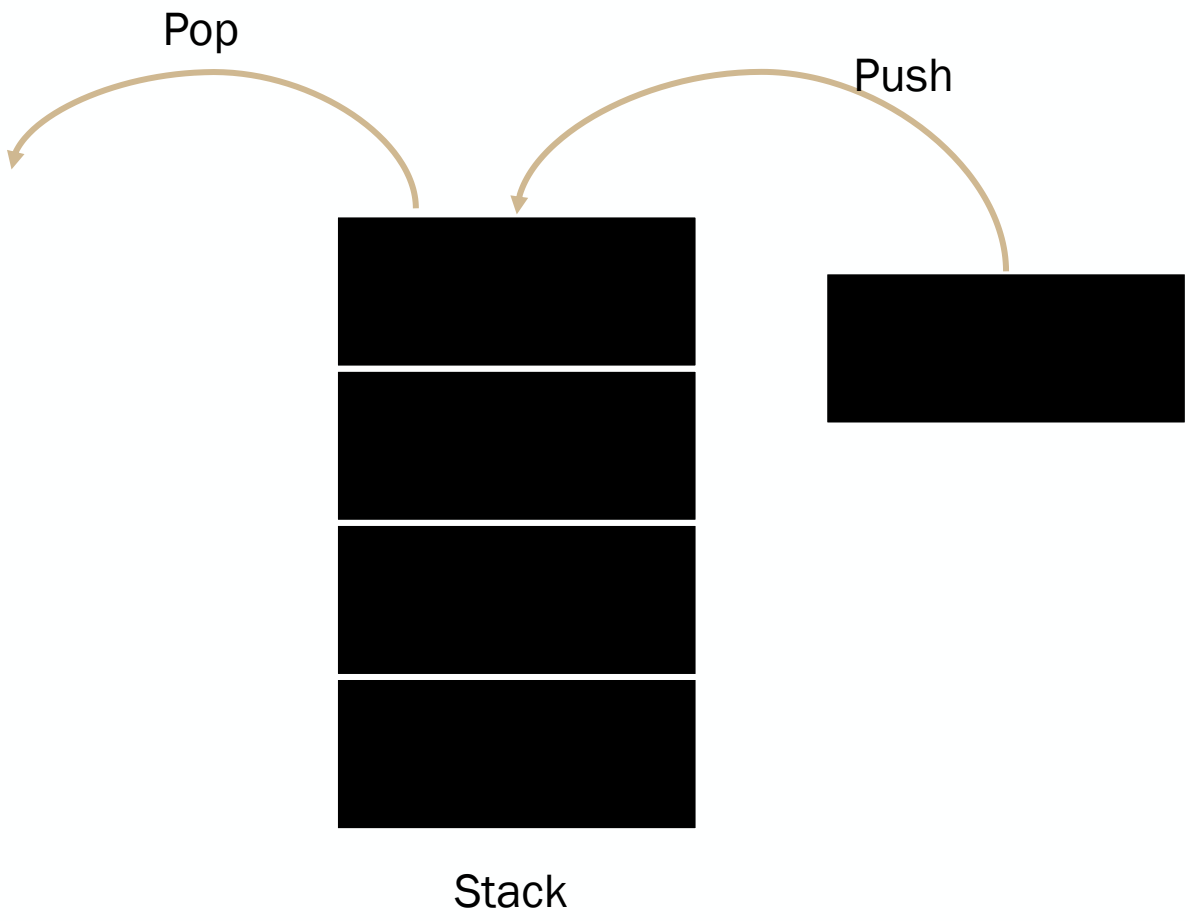
Foundation: Data & Scalability

Outline

- Stacks and queues
- Trees and graphs
- Problems: HashMap, stacks, queues
- Problems: binary trees, binary search trees
- Problems: heaps, graph, combined pattern

Stack & Queue

Stack



Examples: undo operation

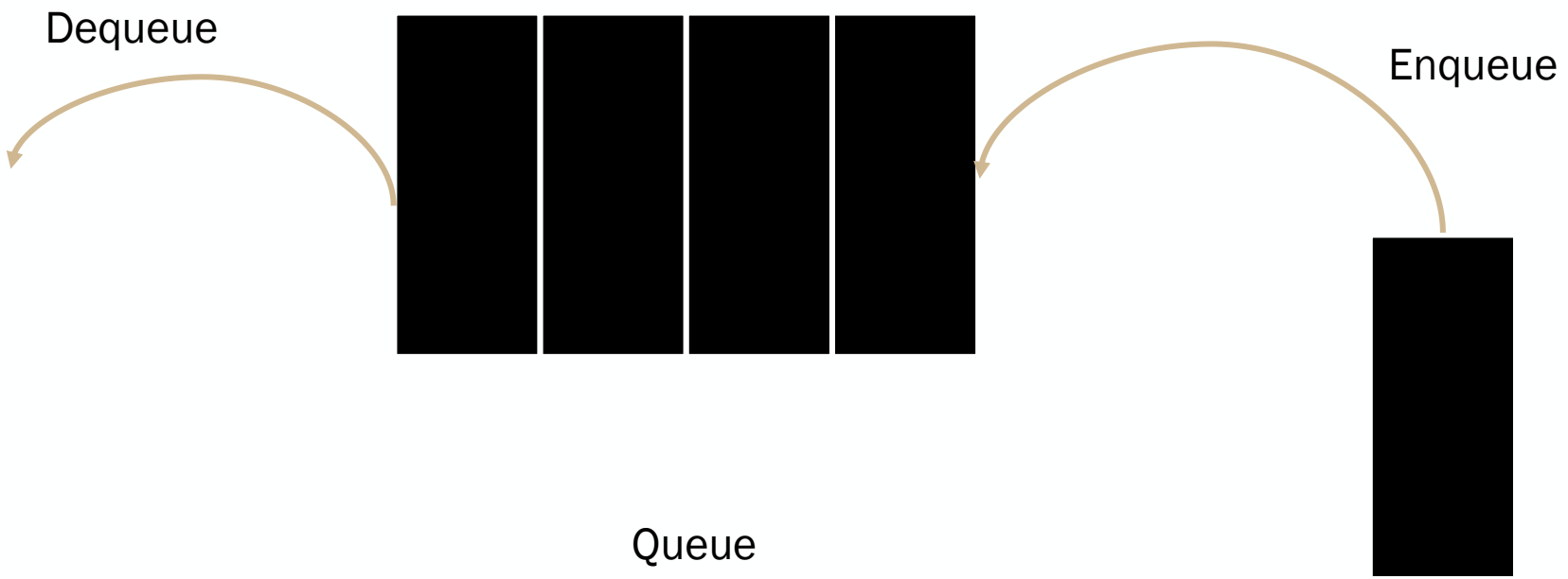
```
stack = []

# push
stack.append("A")
stack.append("B")
stack.append("C")

print(stack)
# ['A', 'B', 'C']

# peek / top
print(stack[-1])
# C

# pop
item = stack.pop()
print(item)
# C

print(stack)
# ['A', 'B']
```

PURDUE UNIVERSITY

Stack & Queue

Queue

Examples: waiting line




```python
from collections import deque

queue = deque()

# enqueue
queue.append("A")
queue.append("B")
queue.append("C")

print(queue)
# deque(['A', 'B', 'C'])

# front / peek
print(queue[0])
# A

# dequeue
item = queue.popleft()
print(item)
# A

print(queue)
# deque(['B', 'C'])
```

PURDUE UNIVERSITY

Problems: Stacks

Valid Parentheses

20. Valid Parentheses

Easy | Topics | Companies | Hint

Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 2:

> **Input:** `s = "()[]{}"`
>
> **Output:** `true`

Example 3:

> **Input:** `s = "(]"`
>
> **Output:** `false`

()[]{}	True
(]	False
(])[	False

PURDUE UNIVERSITY

Problems: Stacks

Valid Parentheses

( ) [ ] { } True

(

( ) ~~( )~~

[

[ ] ~~[ ]~~

{

{ } ~~{ }~~

( ]) [False

(

( ]

( ])

( ]) [

Problems: Queues

Time Needed to Buy Tickets

2073. Time Needed to Buy Tickets

Easy | Topics | Companies | Hint

There are `n` people in a line queuing to buy tickets, where the `0th` person is at the **front** of the line and the `(n - 1)th` person is at the **back** of the line.

You are given a **0-indexed** integer array `tickets` of length `n` where the number of tickets that the `ith` person would like to buy is `tickets[i]`.

Each person takes **exactly 1 second** to buy a ticket. A person can only buy **1 ticket at a time** and has to go back to **the end** of the line (which happens **instantaneously**) in order to buy more tickets. If a person does not have any tickets left to buy, the person will **leave** the line.

Return the **time taken** for the person **initially** at position **k** (0-indexed) to finish buying tickets.

Problems: Queues

Time Needed to Buy Tickets

Example 1:

> **Input:** `tickets = [2,3,2], k = 2`
>
> **Output:** 6
>
> **Explanation:**
>
> - The queue starts as [2,3,2], where the kth person is underlined.
> - After the person at the front has bought a ticket, the queue becomes [3,2,1] at 1 second.
> - Continuing this process, the queue becomes [2,1,2] at 2 seconds.
> - Continuing this process, the queue becomes [1,2,1] at 3 seconds.
> - Continuing this process, the queue becomes [2,1] at 4 seconds. Note: the person at the front left the queue.
> - Continuing this process, the queue becomes [1,1] at 5 seconds.
> - Continuing this process, the queue becomes [1] at 6 seconds. The kth person has bought all their tickets, so return 6.

Problems: Queues

Time Needed to Buy Tickets

[2, 3, **2**]

[3, **2**] Dequeue: 2, operation: -1, obtain: 1

[3, **2,** 1] Enqueue: 1

[**2,** 1] Dequeue: 3, operation: -1, obtain: 2

[**2,** 1, 2] Enqueue: 2

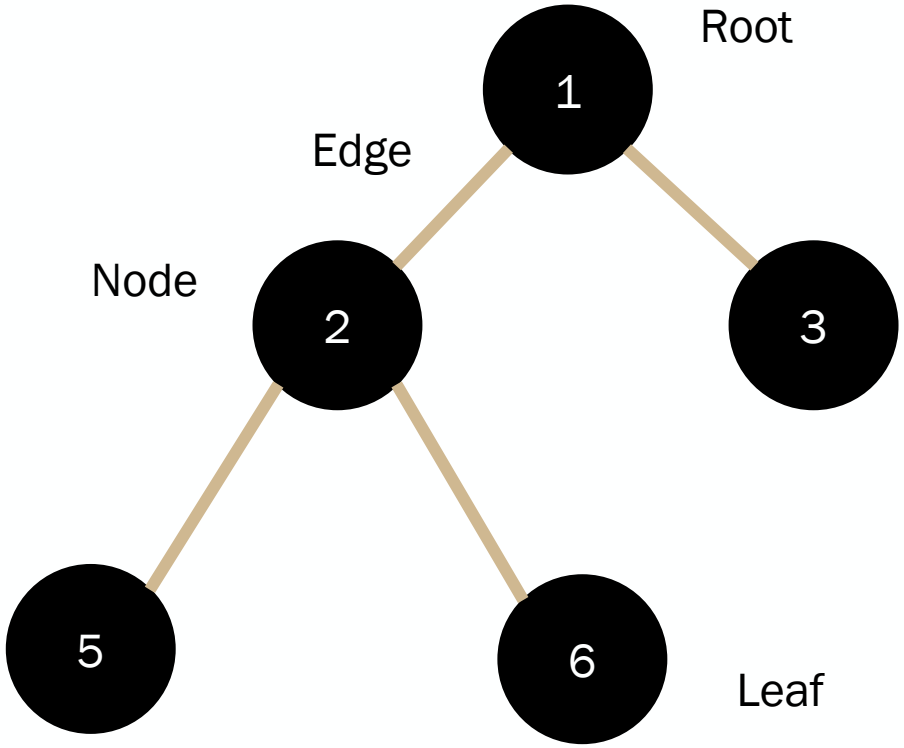
Trees and Graphs

- Key Concepts
  - Hierarchical structure: parent → children
  - Special case of graph (no cycles)
  - Common types:
    - Binary Tree
    - Binary Search Tree (BST)
- Traversals
  - DFS:
    - Preorder
    - Inorder (sorted output in BST)
    - Postorder
  - BFS (level-order)

PURDUE UNIVERSITY

Trees and Graphs

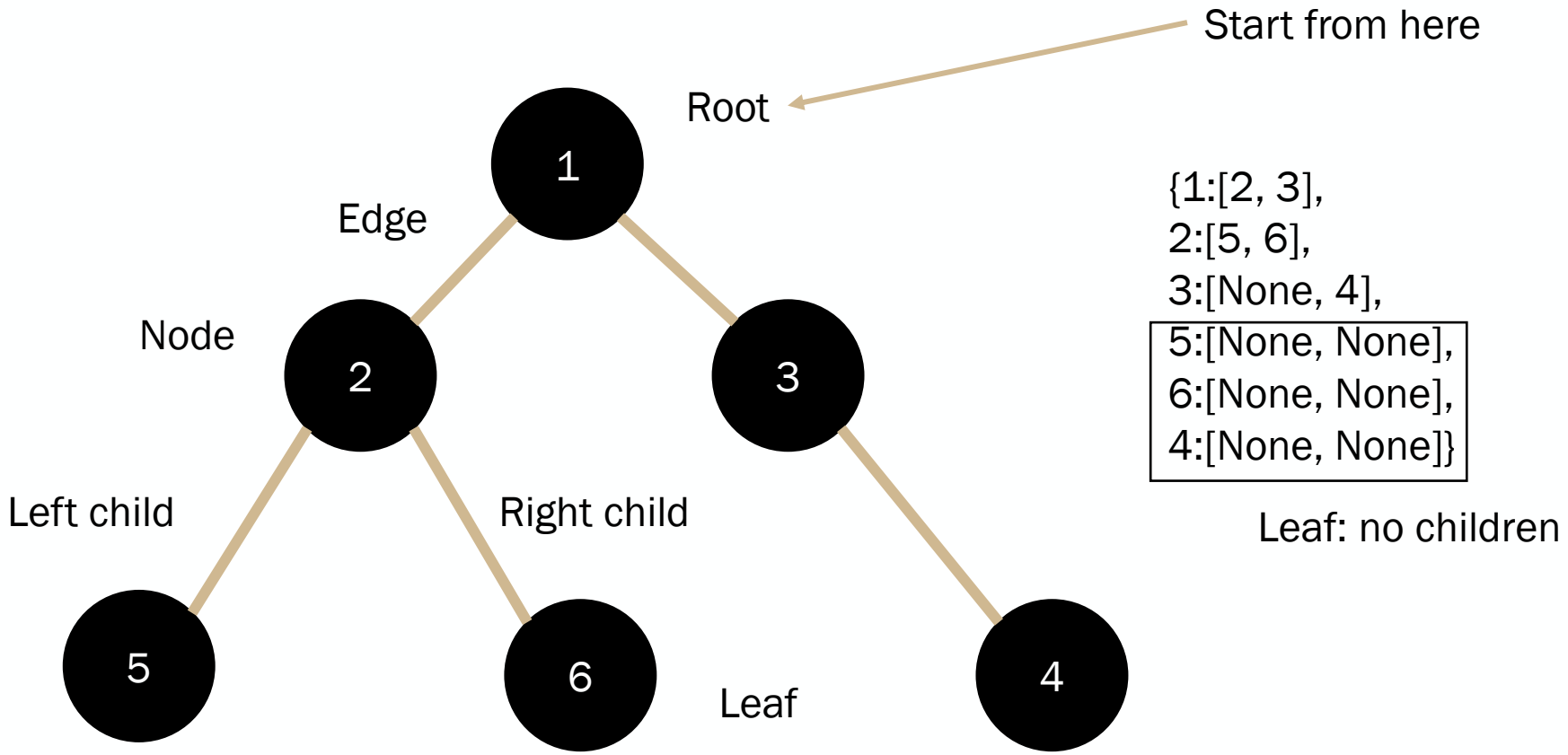
Binary Tree



PURDUE UNIVERSITY

Trees and Graphs

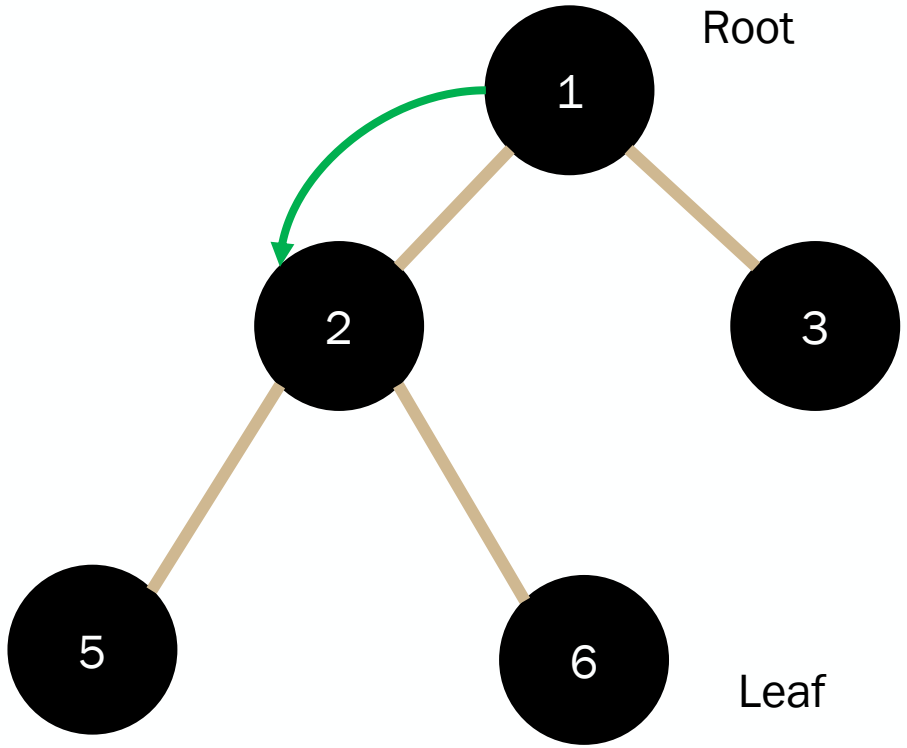
Binary Tree

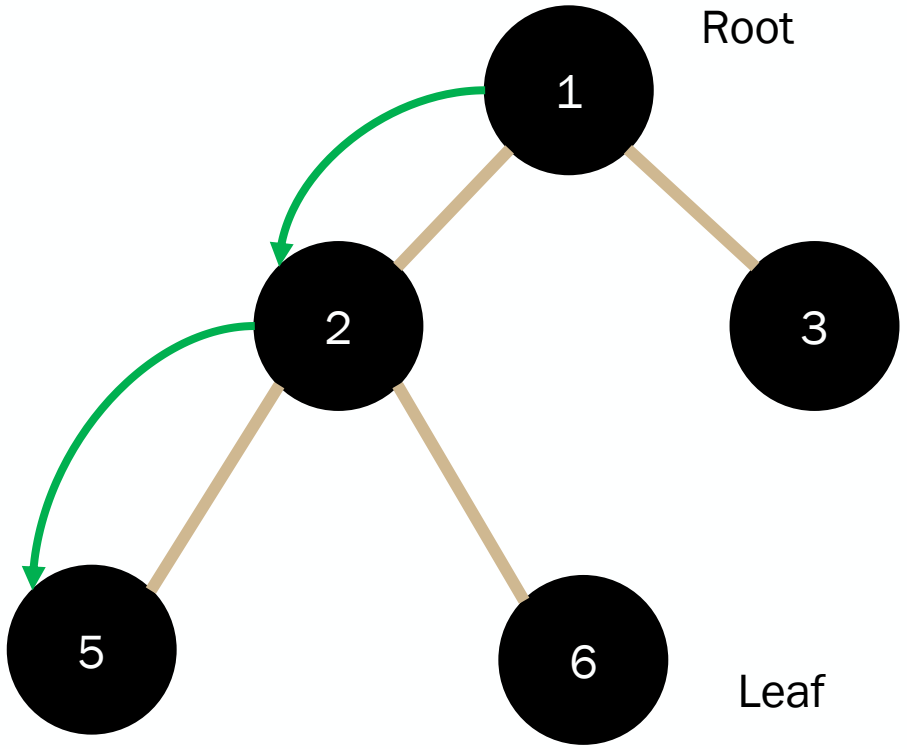


PURDUE UNIVERSITY

Trees and Graphs

DFS

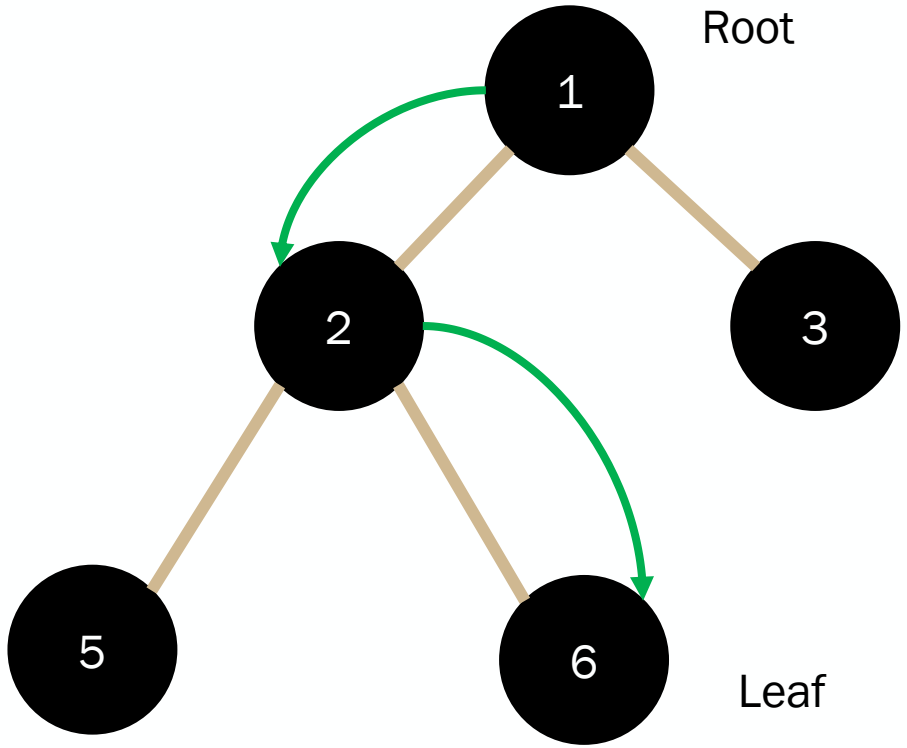


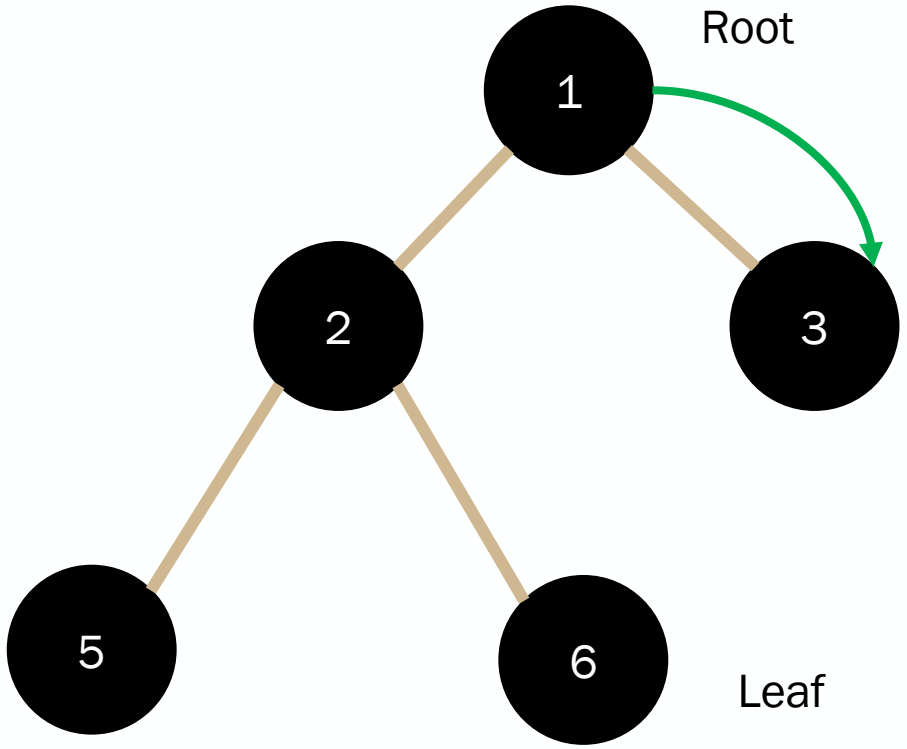


PURDUE UNIVERSITY

Trees and Graphs

DFS



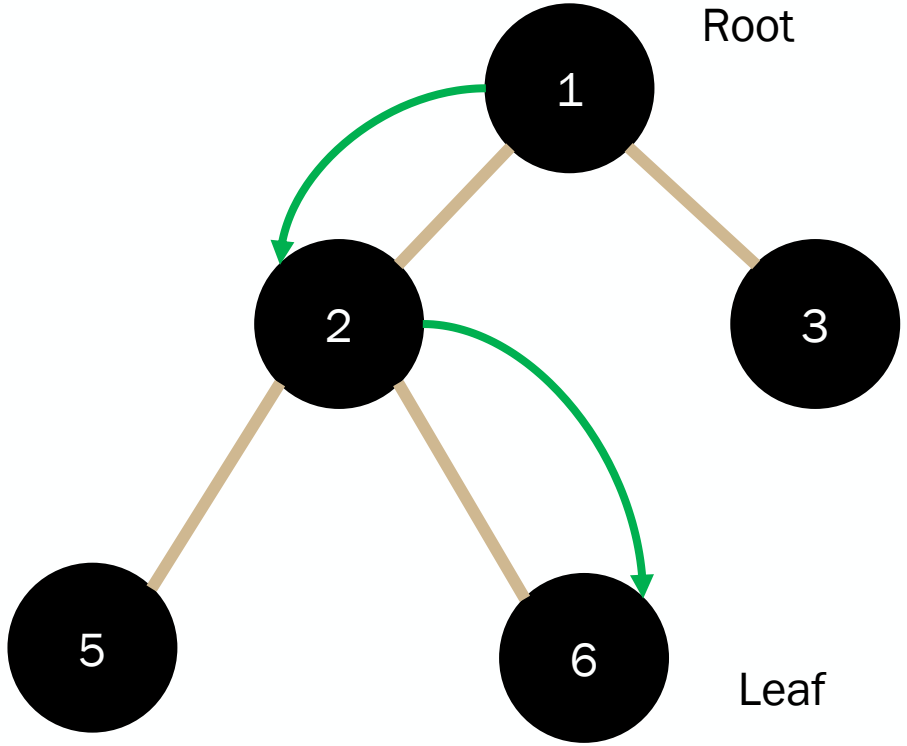


PURDUE UNIVERSITY

DFS

Algorithm

{1:[2, 3], 2:[5, 6], 3:[], 5:[], 6:[]}



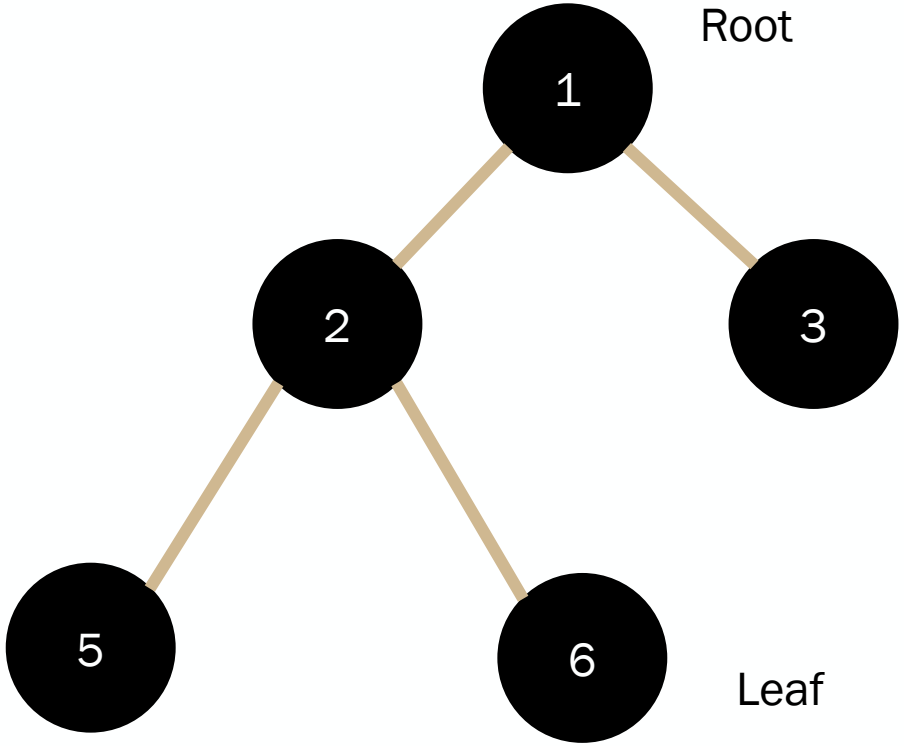

```python
def dfs(graph, start):
    visited = set()
    stack = [start]

    while stack:
        node = stack.pop()

        if node not in visited:
            print(node)
            visited.add(node)

            # Add neighbors to stack
            for neighbor in graph[node]:
                if neighbor not in visited:
                    stack.append(neighbor)
```

PURDUE UNIVERSITY

DFS

Algorithm

{1:[2, 3], 2:[5, 6], 3:[], 5:[], 6:[]}




```
def dfs(graph, start):
    visited = set()
    stack = [start]

    while stack:
        node = stack.pop()

        if node not in visited:
            print(node)
            visited.add(node)

            # Add neighbors to stack
            for neighbor in graph[node]:
                if neighbor not in visited:
                    stack.append(neighbor)
```

Stack=[1], visited={}

Stack=[], visited={}, node=1

Stack=[], visited={1}, node=1

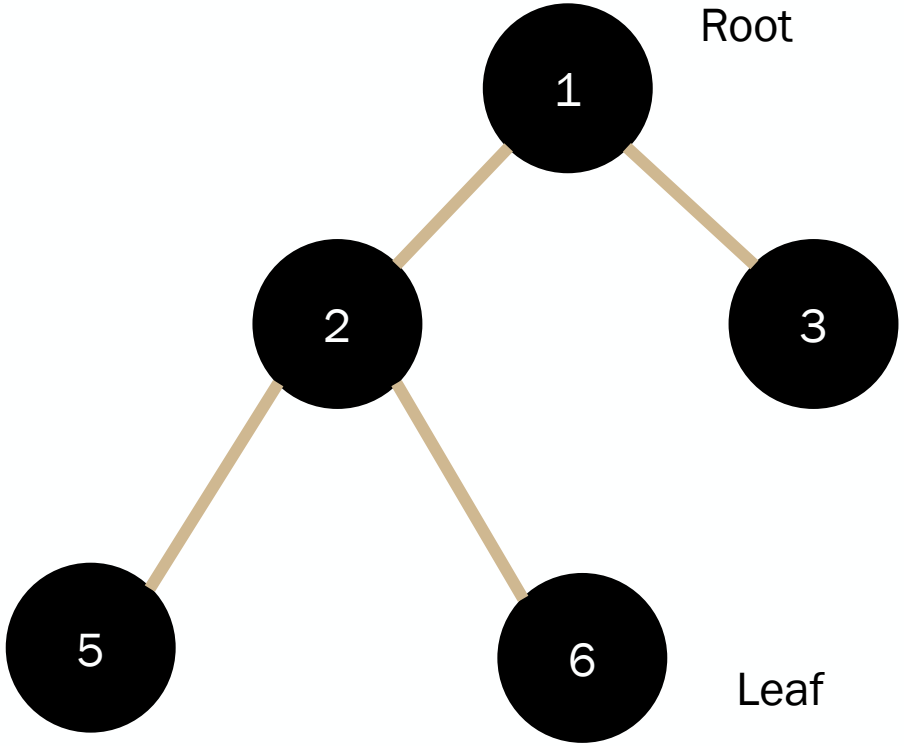
Stack=[2, 3], visited={1}, node=1

PURDUE UNIVERSITY

DFS

Algorithm

{1:[2, 3], 2:[5, 6], 3:[], 5:[], 6:[]}




```python
def dfs(graph, start):
    visited = set()
    stack = [start]

    while stack:
        node = stack.pop()

        if node not in visited:
            print(node)
            visited.add(node)

            # Add neighbors to stack
            for neighbor in graph[node]:
                if neighbor not in visited:
                    stack.append(neighbor)
```

Stack=[2], visited={1}, node=3

Stack=[2], visited={1, 3}, node=3

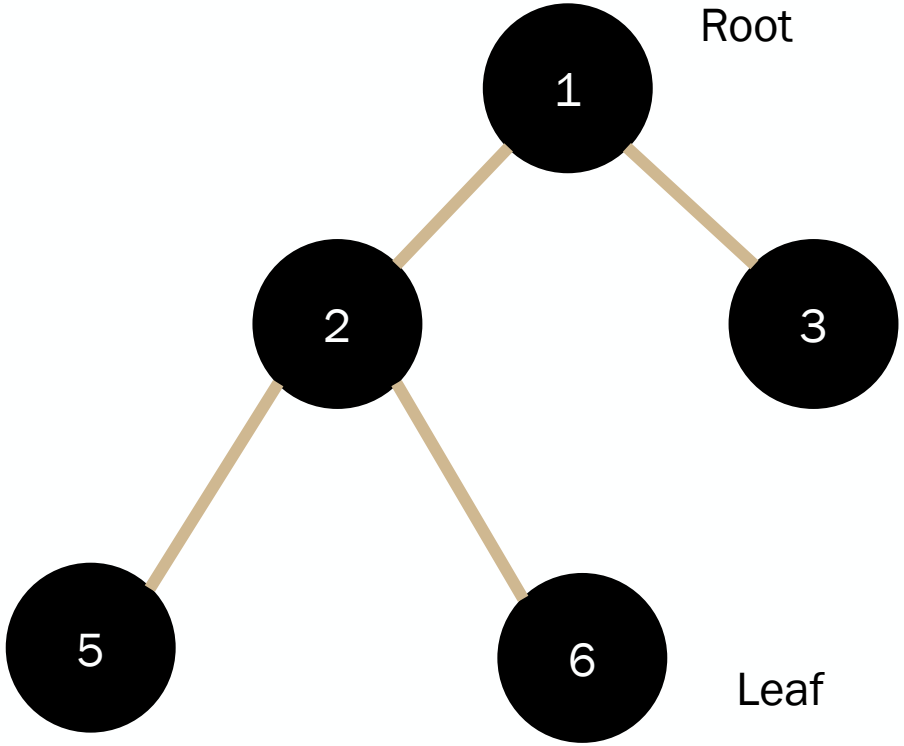
Stack=[2], visited={1, 3}, node=3

PURDUE UNIVERSITY

DFS

Algorithm

{1:[2, 3], 2:[5, 6], 3:[], 5:[], 6:[]}




```python
def dfs(graph, start):
    visited = set()
    stack = [start]

    while stack:
        node = stack.pop()

        if node not in visited:
            print(node)
            visited.add(node)

            # Add neighbors to stack
            for neighbor in graph[node]:
                if neighbor not in visited:
                    stack.append(neighbor)
```

Stack=[], visited={1, 3}, node=2

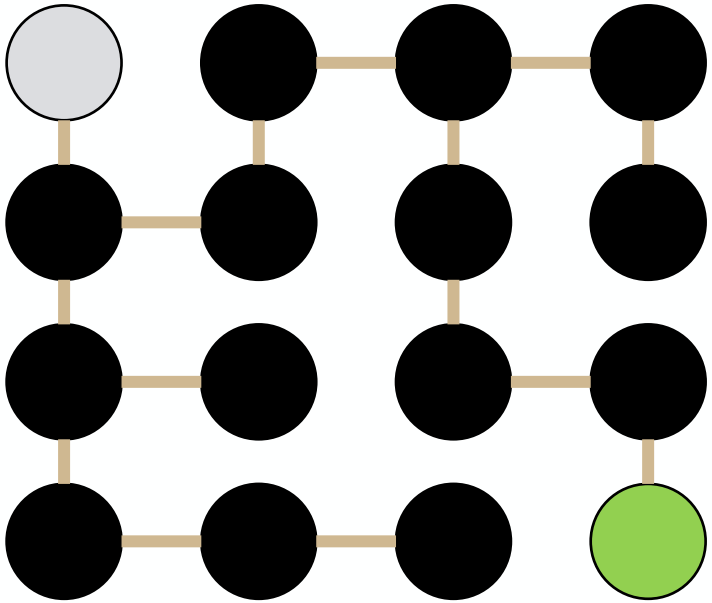
Stack=[], visited={1, 3, 2}, node=2

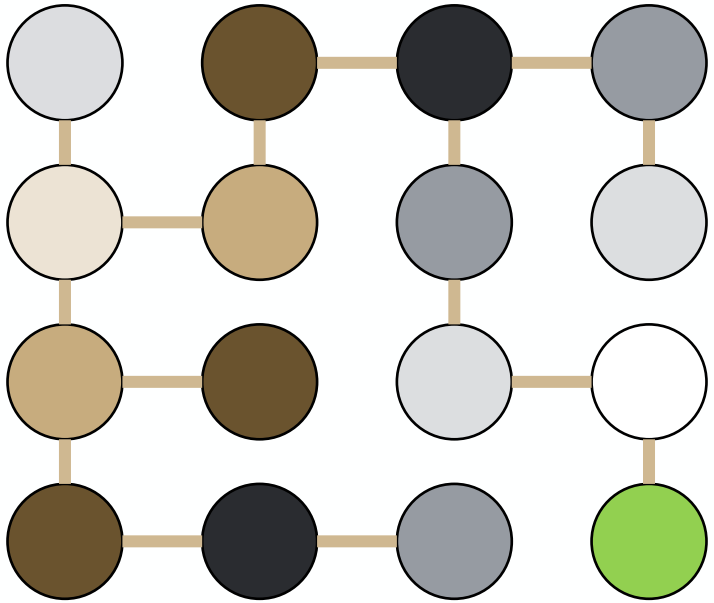
Stack=[5, 6], visited={1, 3, 2}, node=2

PURDUE UNIVERSITY

Trees and Graphs

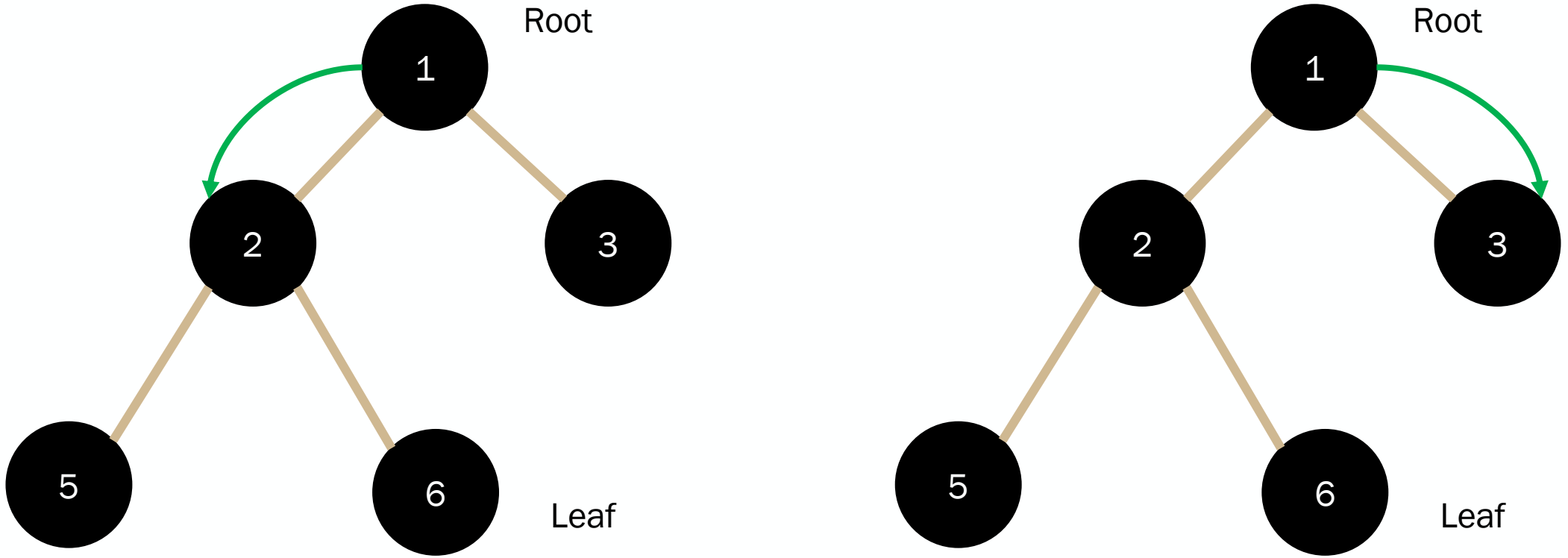
DFS





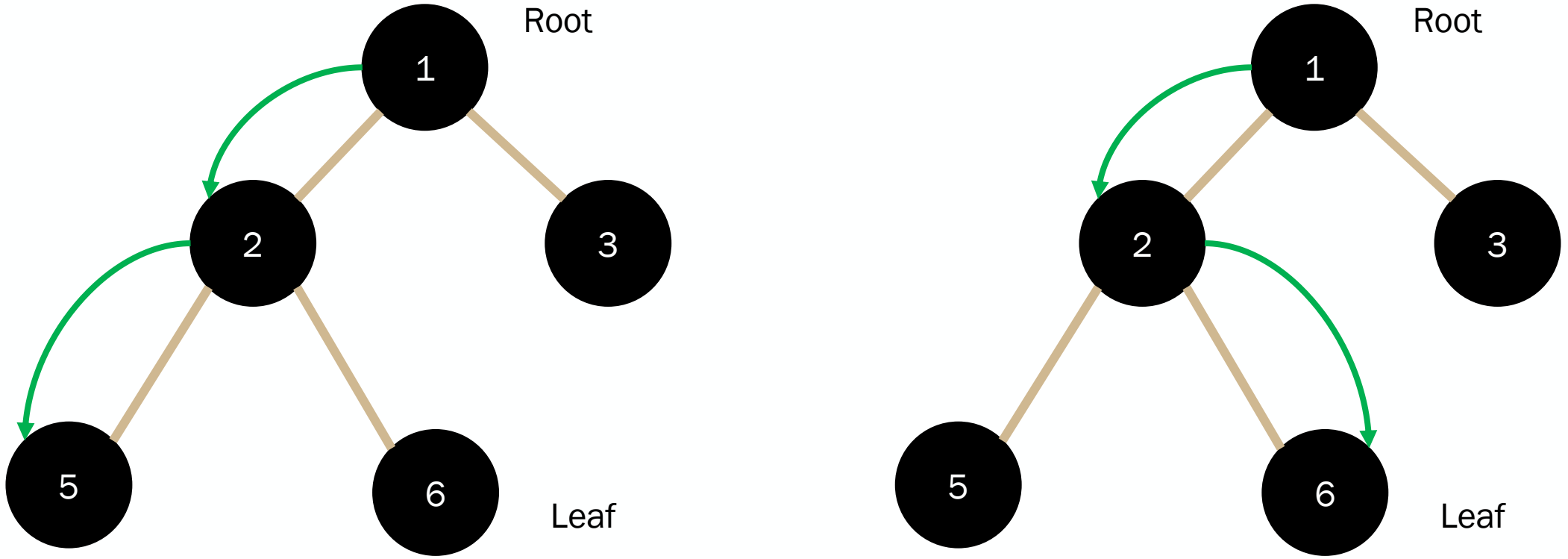
Trees and Graphs

BFS



Trees and Graphs

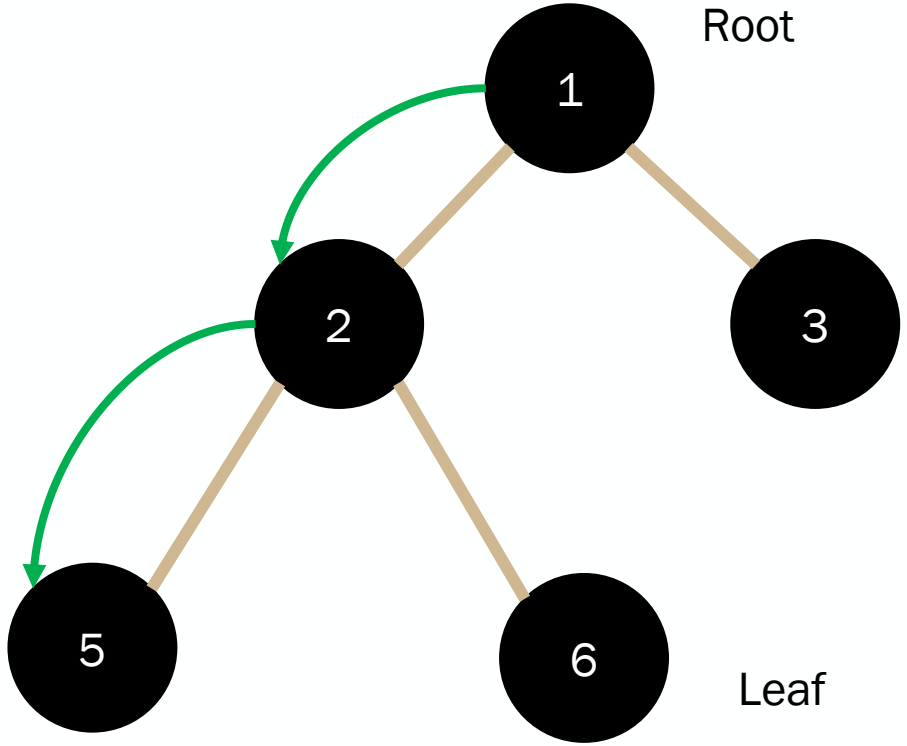
BFS



BFS

Algorithm

{1:[2, 3], 2:[5, 6], 3:[], 5:[], 6:[]}



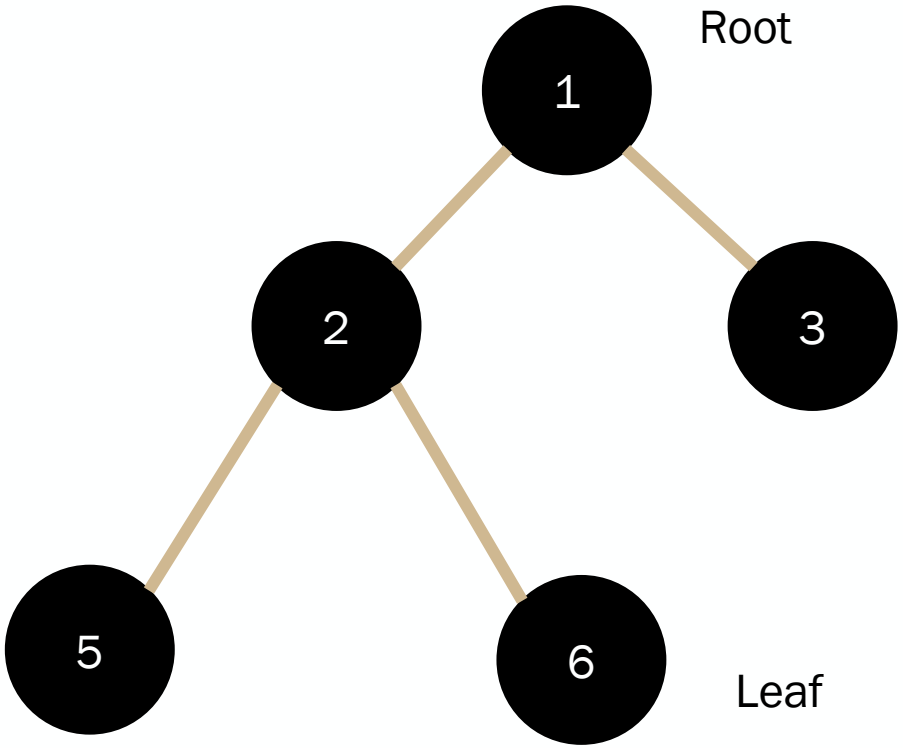

```python
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])

    while queue:
        node = queue.popleft()

        if node not in visited:
            print(node)
            visited.add(node)

            # Add neighbors to queue
            for neighbor in graph[node]:
                if neighbor not in visited:
                    queue.append(neighbor)
```

PURDUE UNIVERSITY

BFS

Algorithm

{1:[2, 3], 2:[5, 6], 3:[], 5:[], 6:[]}




```python
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])

    while queue:
        node = queue.popleft()

        if node not in visited:
            print(node)
            visited.add(node)

            # Add neighbors to queue
            for neighbor in graph[node]:
                if neighbor not in visited:
                    queue.append(neighbor)
```

Queue=[1], visited={}

Queue=[], visited={}, node=1

Queue=[], visited={1}, node=1

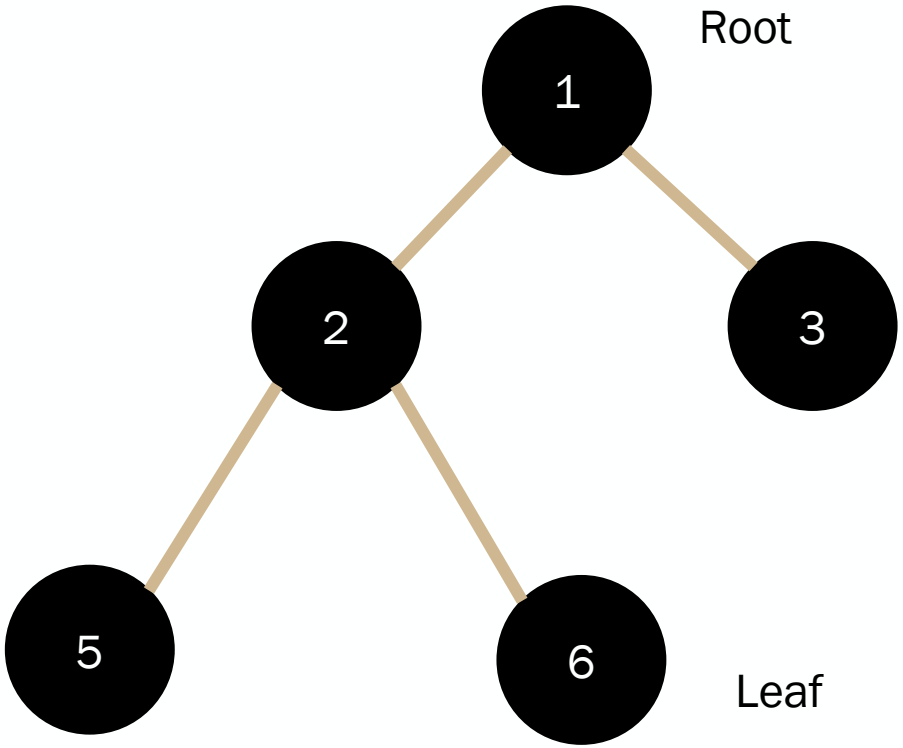
Queue=[2, 3], visited={1}, node=1

PURDUE UNIVERSITY

BFS

Algorithm

{1:[2, 3], 2:[5, 6], 3:[], 5:[], 6:[]}




```python
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])

    while queue:
        node = queue.popleft()

        if node not in visited:
            print(node)
            visited.add(node)

            # Add neighbors to queue
            for neighbor in graph[node]:
                if neighbor not in visited:
                    queue.append(neighbor)
```

Queue=[3], visited={1}, node=2

Queue=[3], visited={1, 2}, node=2

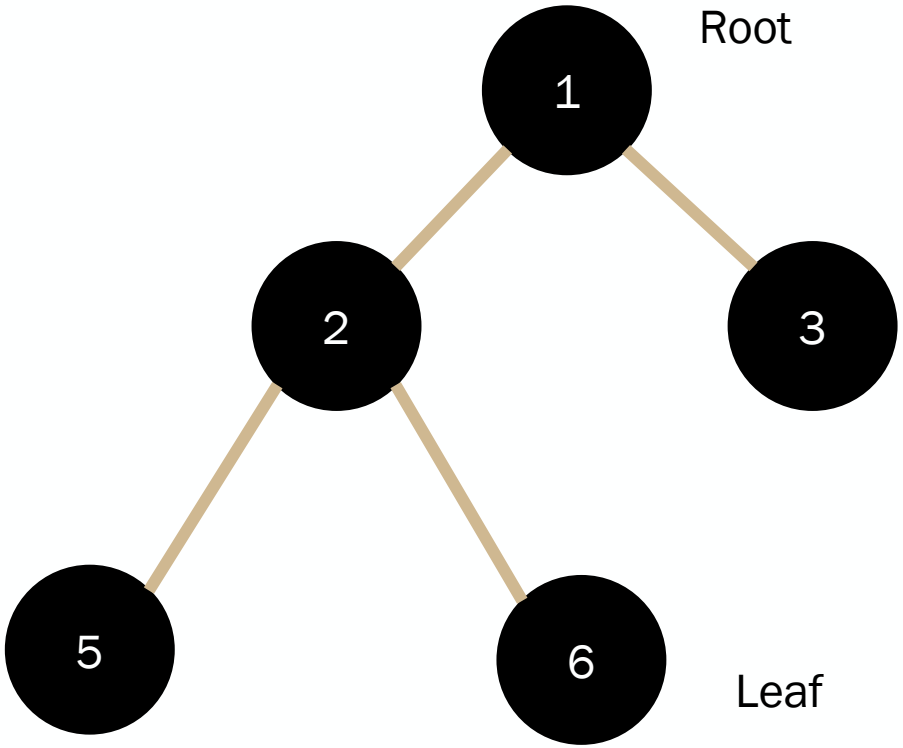
Queue=[3, 5, 6], visited={1, 2}, node=2

PURDUE UNIVERSITY

BFS

Algorithm

{1:[2, 3], 2:[5, 6], 3:[], 5:[], 6:[]}




```python
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])

    while queue:
        node = queue.popleft()

        if node not in visited:
            print(node)
            visited.add(node)

            # Add neighbors to queue
            for neighbor in graph[node]:
                if neighbor not in visited:
                    queue.append(neighbor)
```

Queue=[5, 6], visited={1, 2}, node=3

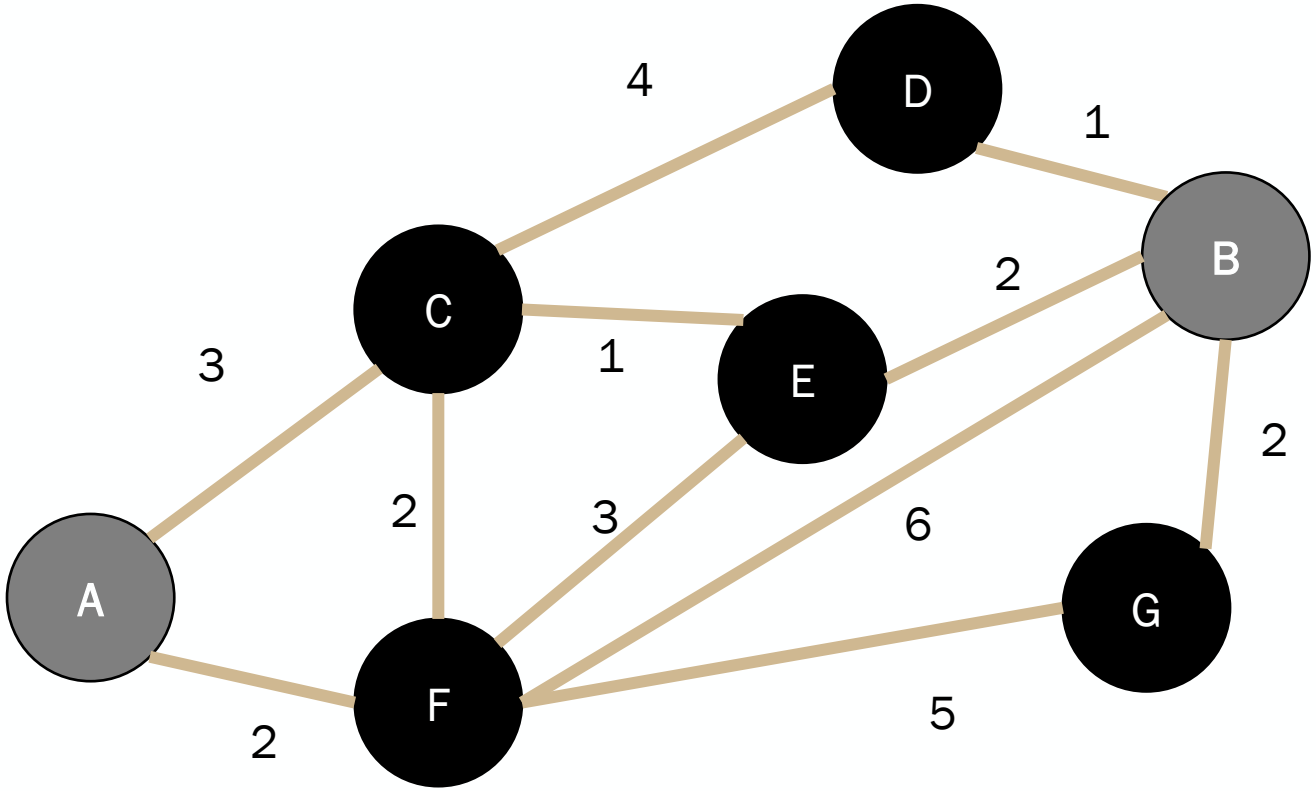
Queue=[5, 6], visited={1, 2, 3}, node=3

Queue=[5, 6], visited={1, 2, 3}, node=3

PURDUE UNIVERSITY

Trees and Graphs

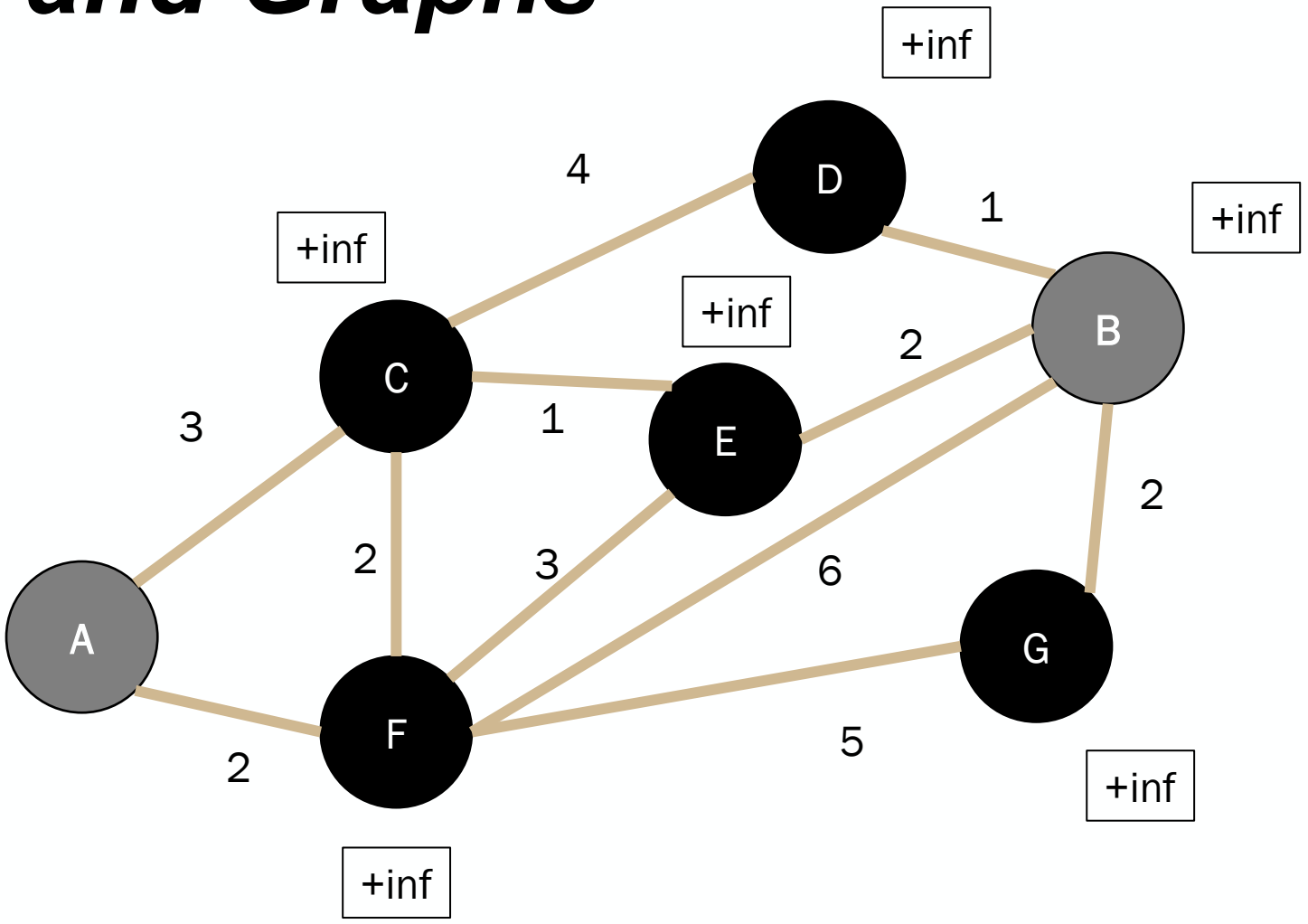
BFS



PURDUE UNIVERSITY

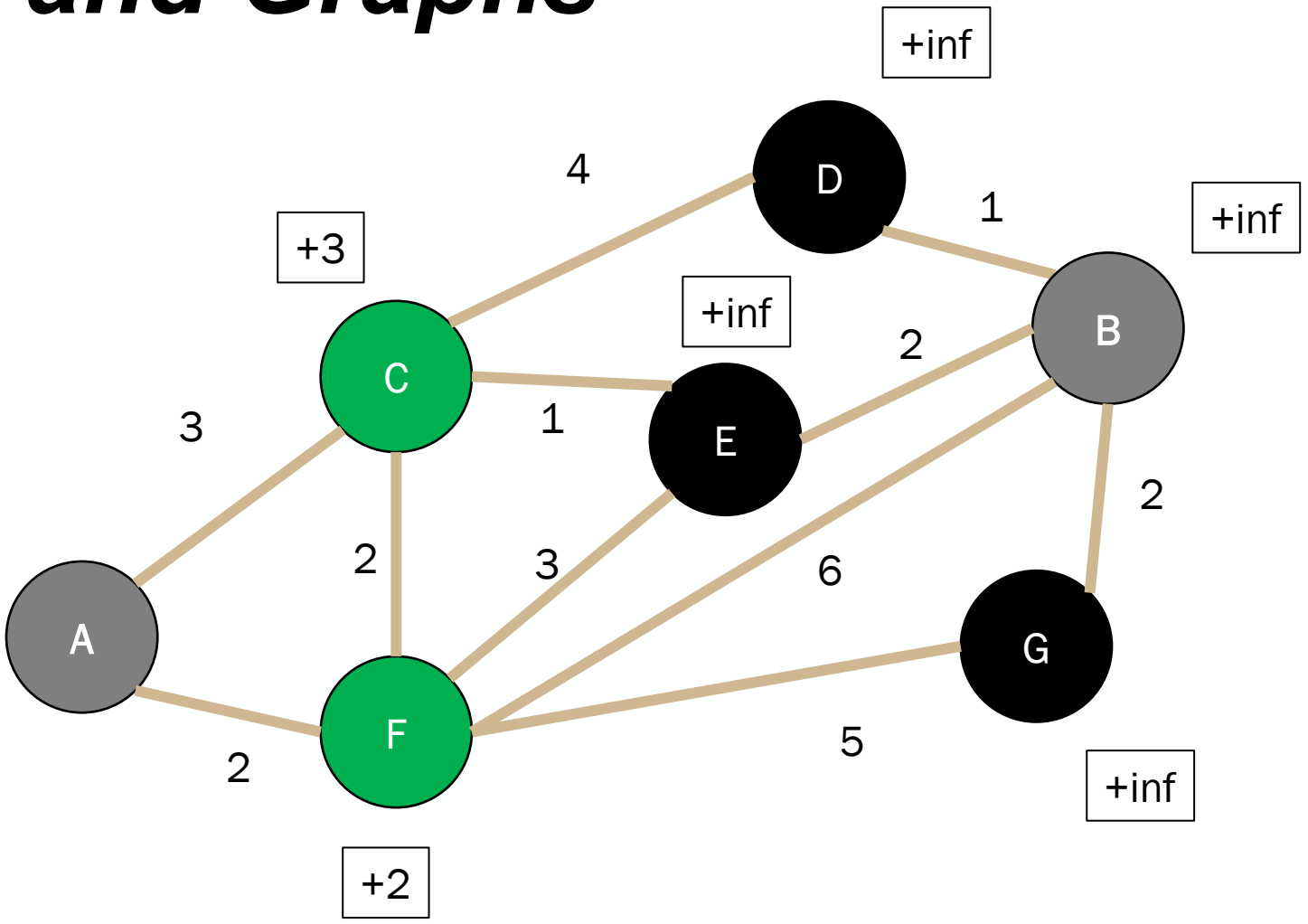
Trees and Graphs

Dijkstra



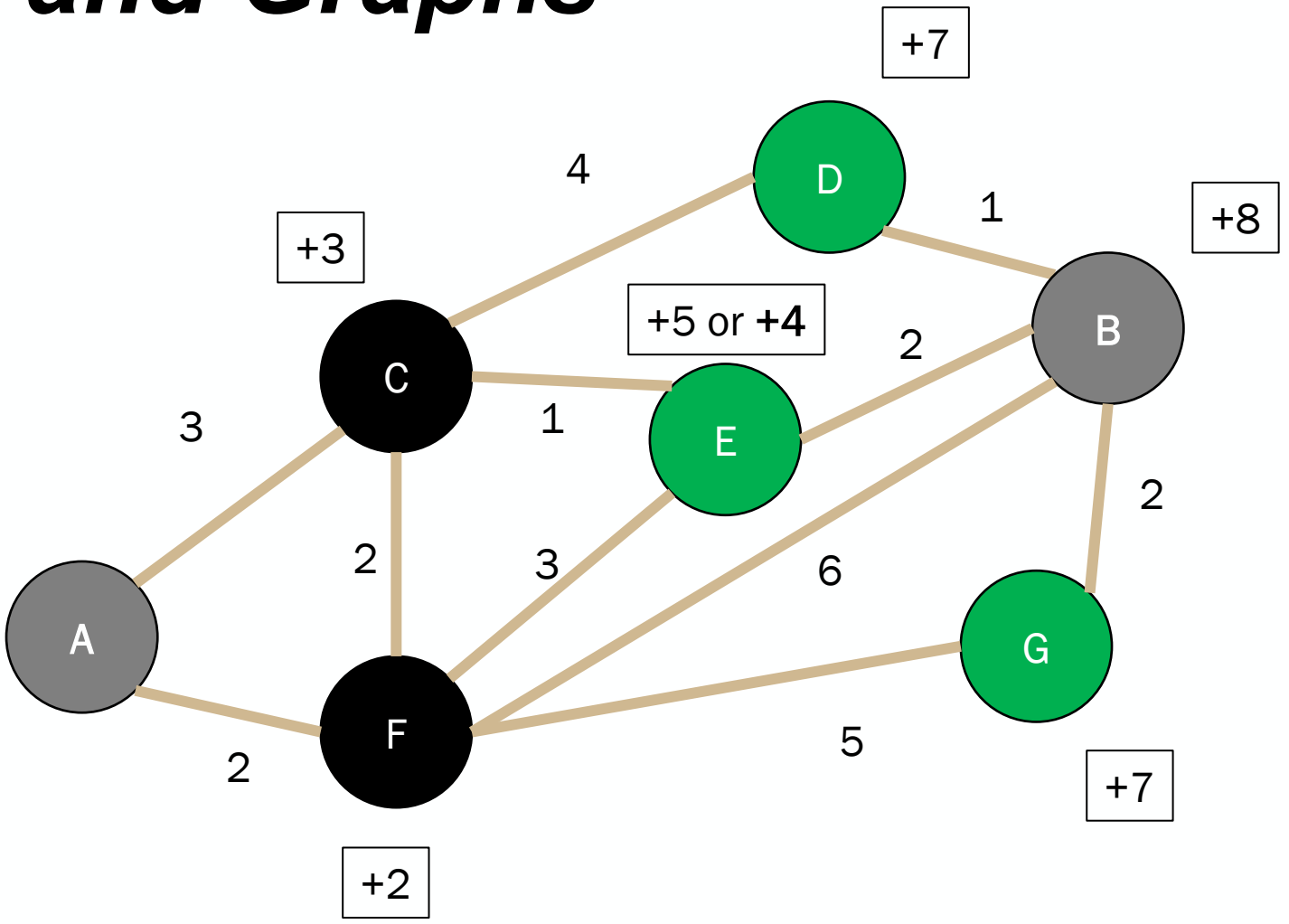
Trees and Graphs

Dijkstra



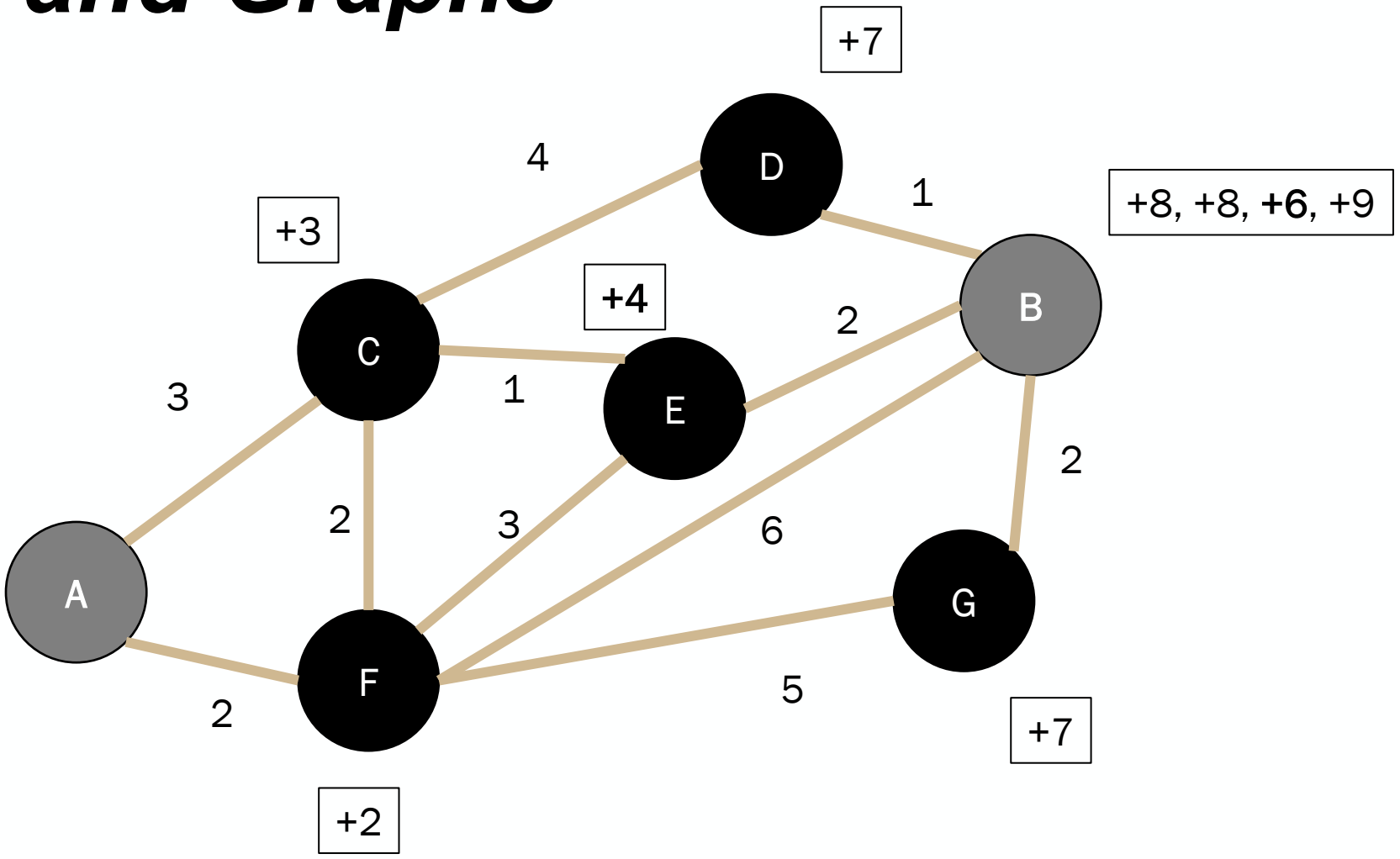
Trees and Graphs

Dijkstra



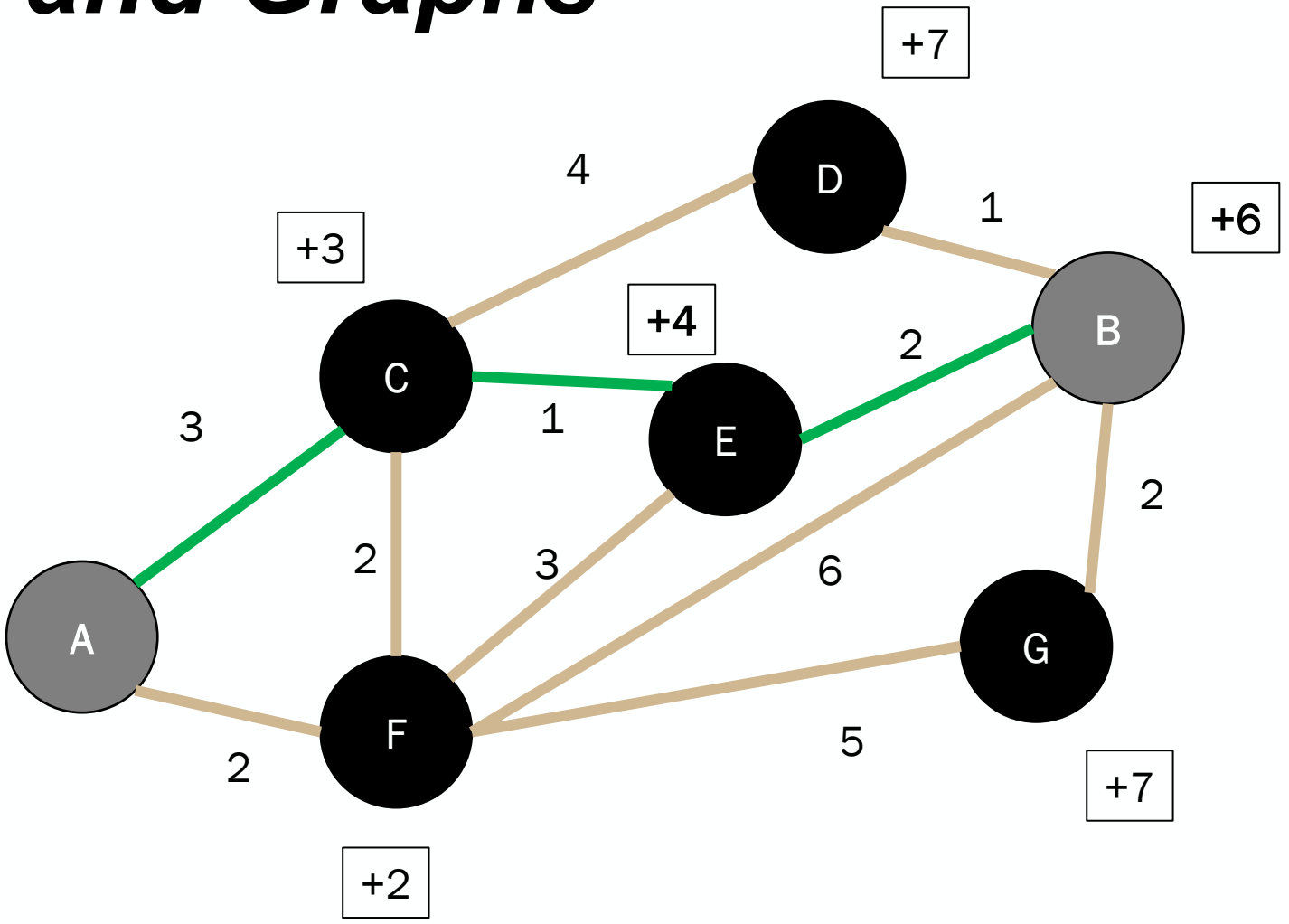
Trees and Graphs

Dijkstra



Trees and Graphs

Dijkstra



Problems: Binary trees

Path Sum

112. Path Sum

Solved

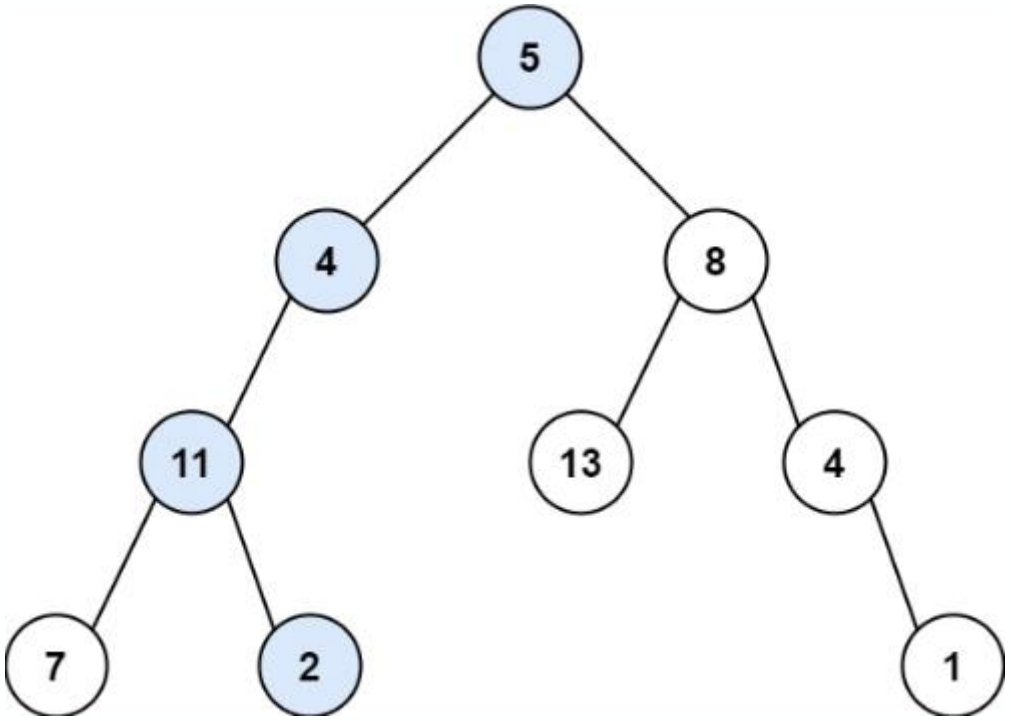
Easy | Topics | Companies

Given the `root` of a binary tree and an integer `targetSum`, return `true` if the tree has a **root-to-leaf** path such that adding up all the values along the path equals `targetSum`.

A **leaf** is a node with no children.

Problems: Binary trees

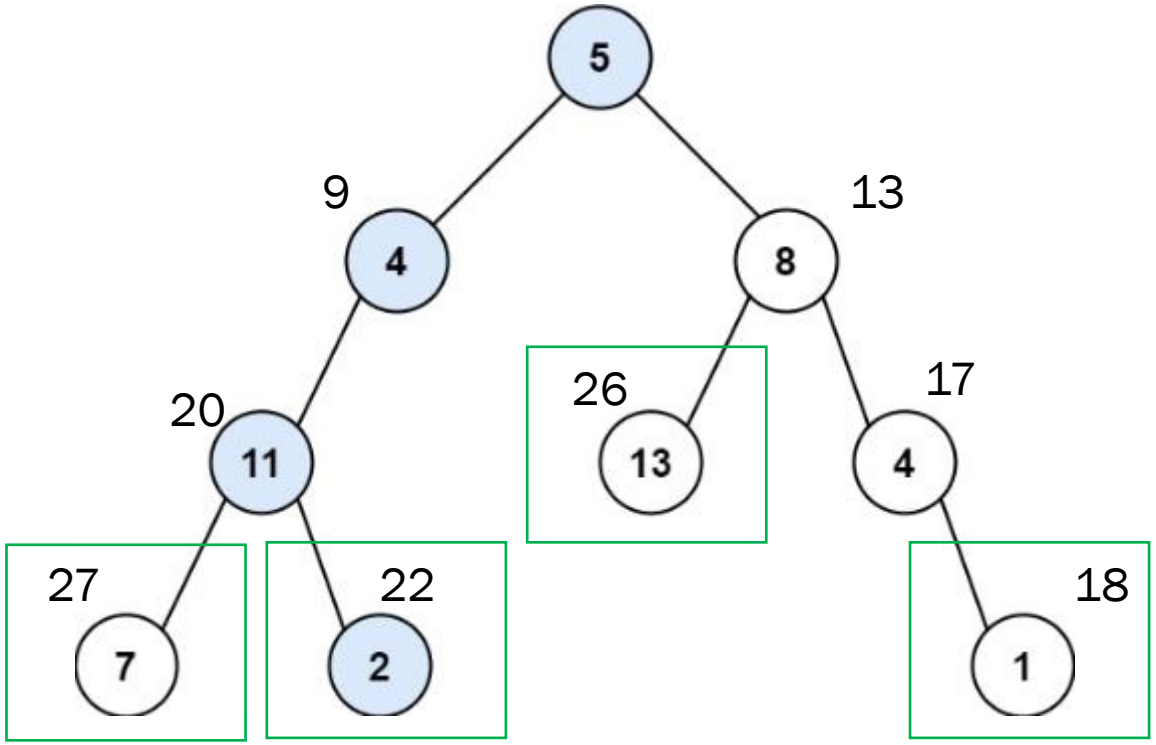
Path Sum




```
Input: root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22
Output: true
Explanation: The root-to-leaf path with the target sum is shown.
```

Problems: Binary trees

Path Sum

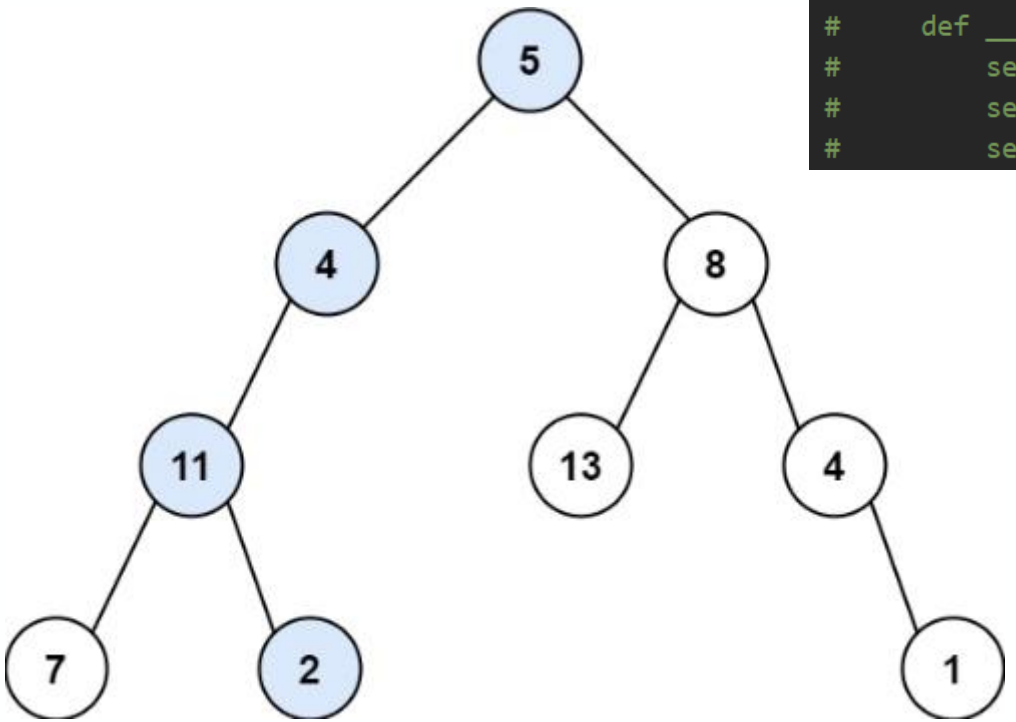



```
Input: root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22
Output: true
Explanation: The root-to-leaf path with the target sum is shown.
```

PURDUE UNIVERSITY

Problems: Binary trees

Path Sum



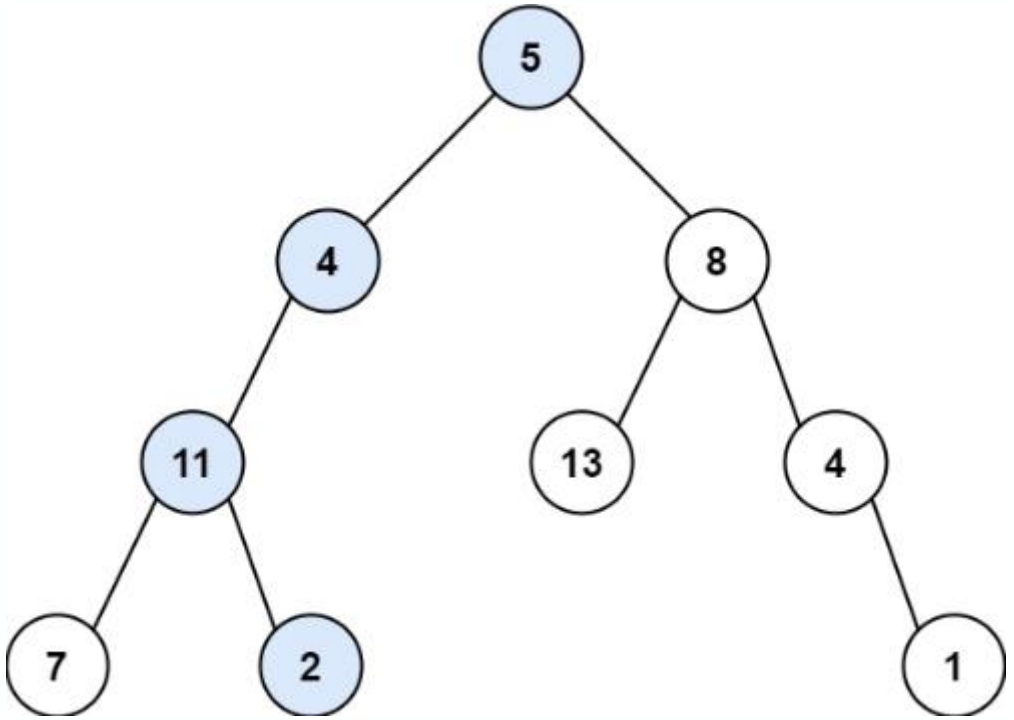

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
```

Python Class

Input: root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22
Output: true
Explanation: The root-to-leaf path with the target sum is shown.

Problems: Binary trees

Path Sum




```python
class Solution:
    def hasPathSum(self, root: Optional[TreeNode], targetSum: int) -> bool:
        # check edge case
        if root is None:
            return False

        from collections import deque

        # BFS
        queue = deque([root])
        while queue:
            node = queue.popleft()
            num = node.val

            node_l = node.left
            node_r = node.right
            if node_l is not None:
                node_l.val += num
                queue.append(node_l)
            if node_r is not None:
                node_r.val += num
                queue.append(node_r)
            # if this node is leaf
            if node_l is None and node_r is None:
                if num == targetSum:
                    return True

        return False
```

PURDUE UNIVERSITY

Problems: Binary trees

Diameter of Binary Tree

543. Diameter of Binary Tree

Solved

Easy | Topics | Companies

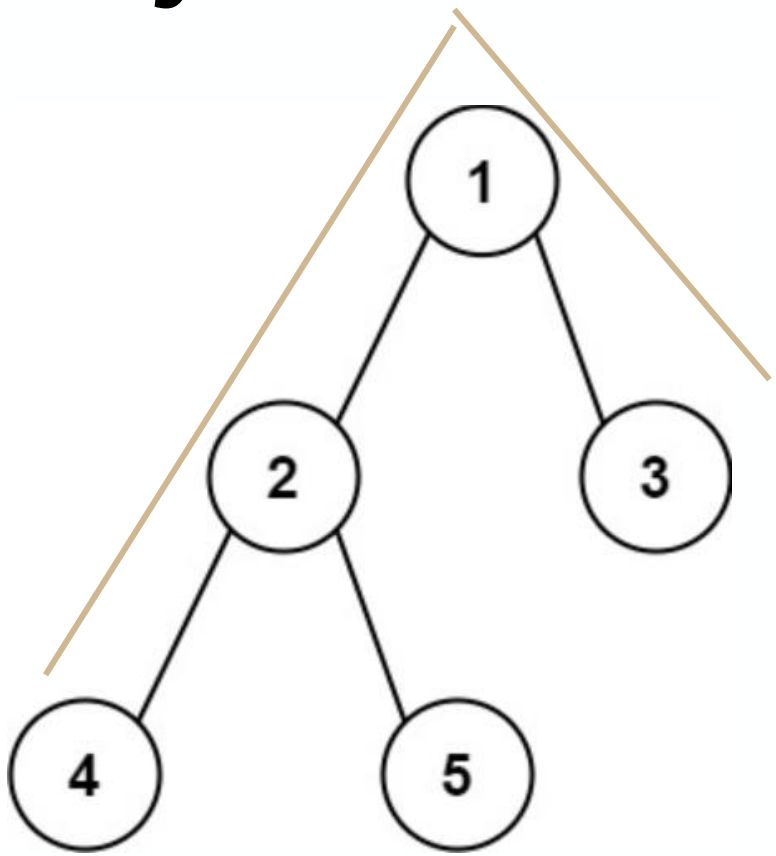
Given the `root` of a binary tree, return *the length of the **diameter** of the tree*.

The **diameter** of a binary tree is the **length** of the longest path between any two nodes in a tree. This path may or may not pass through the `root`.

The **length** of a path between two nodes is represented by the number of edges between them.

Problems: Binary trees

Diameter of Binary Tree

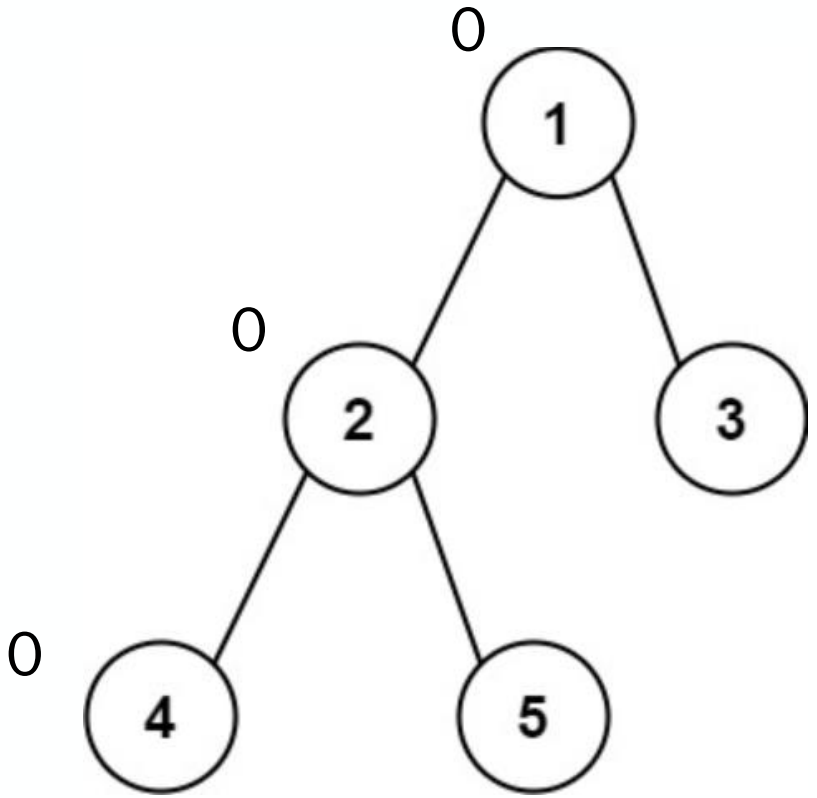


Input: root = [1,2,3,4,5]
Output: 3
Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3].

PURDUE UNIVERSITY

Problems: Binary trees

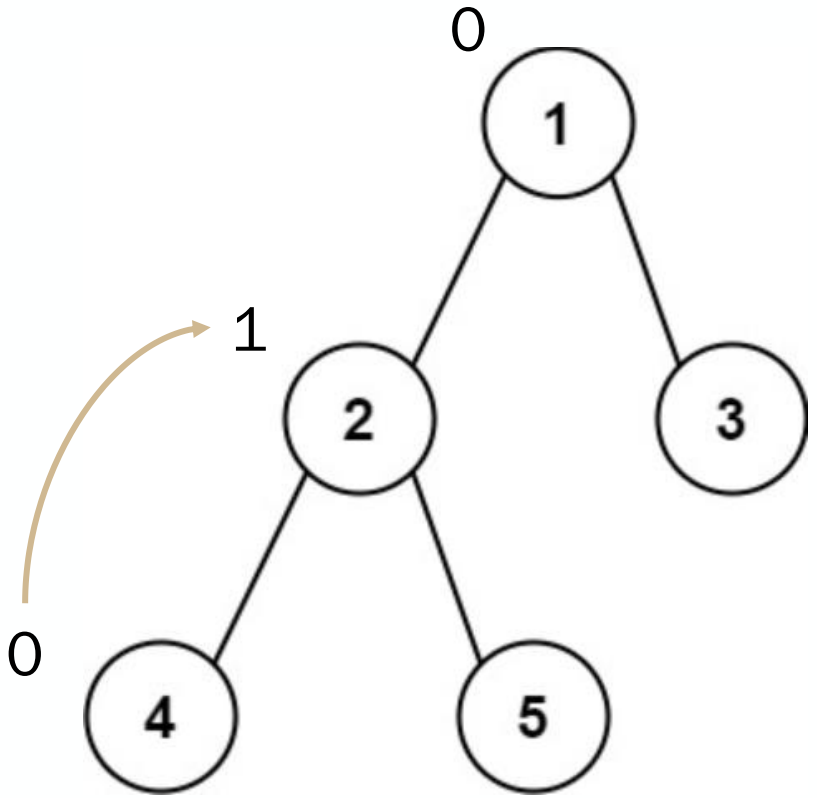
Diameter of Binary Tree



Input: root = [1,2,3,4,5]
Output: 3
Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3].

Problems: Binary trees

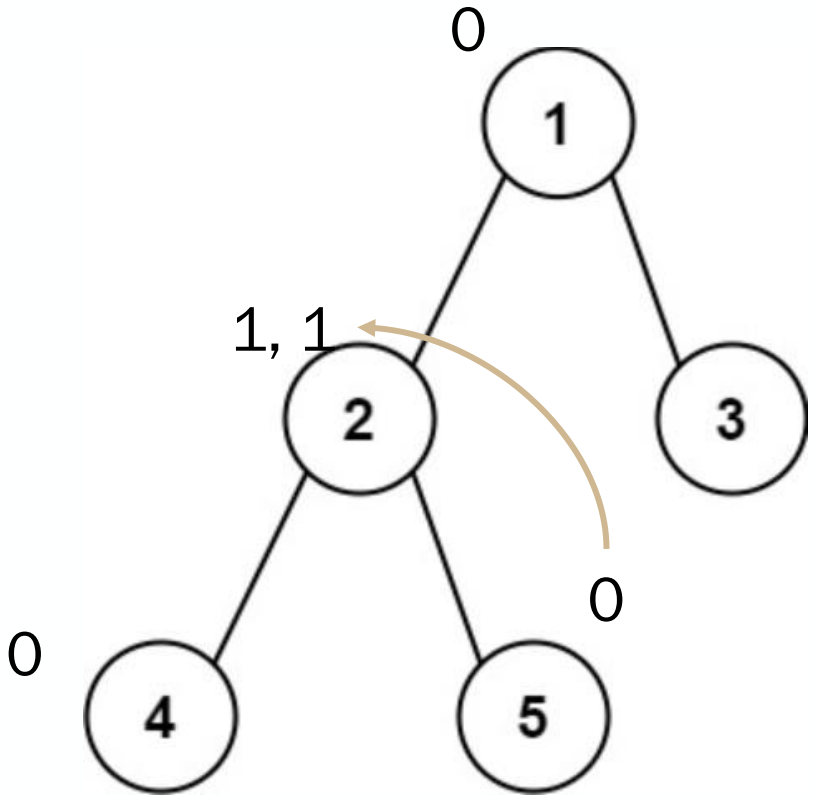
Diameter of Binary Tree



Input: root = [1,2,3,4,5]
Output: 3
Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3].

Problems: Binary trees

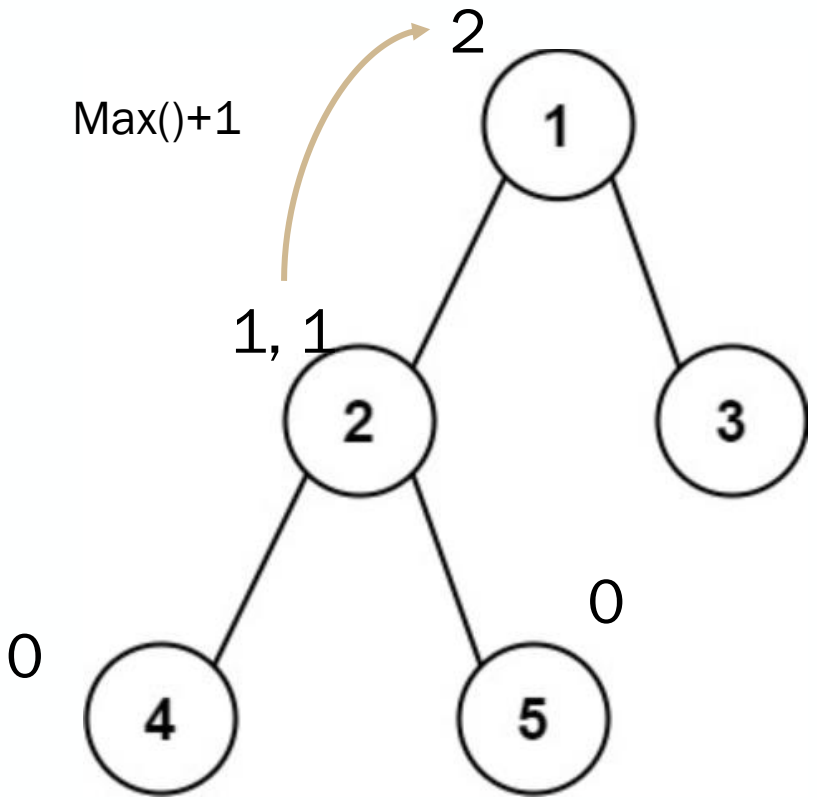
Diameter of Binary Tree



Input: root = [1,2,3,4,5]
Output: 3
Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3].

Problems: Binary trees

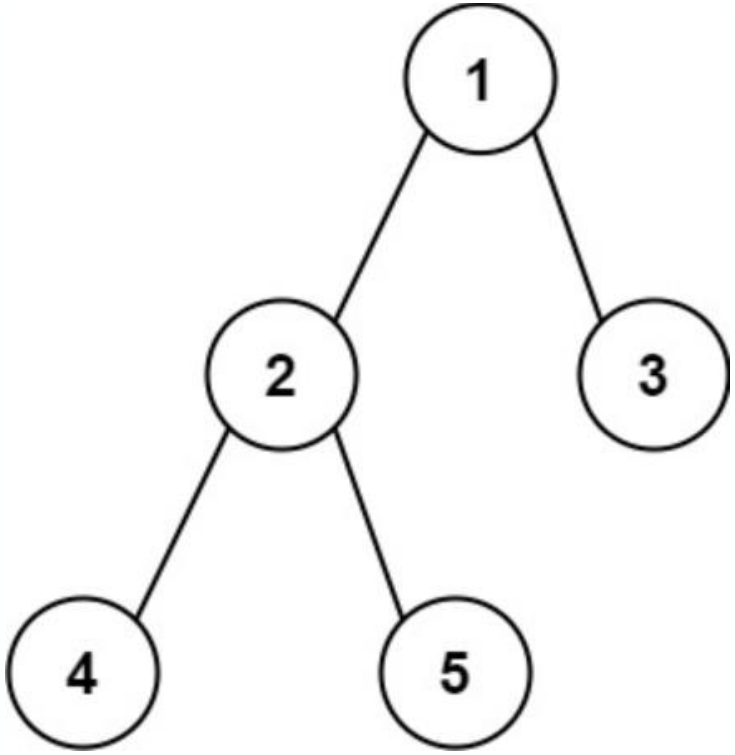
Diameter of Binary Tree



Input: root = [1,2,3,4,5]
Output: 3
Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3].

Problems: Binary trees

Diameter of Binary Tree



Input: root = [1,2,3,4,5]
Output: 3
Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3].

```python
class Solution:
    def diameterOfBinaryTree(self, root: Optional[TreeNode]) -> int:
        if root is None:
            return 0

        global diameter
        diameter = 0

        def dfs(node):
            node_l = node.left
            node_r = node.right

            # is leaf, distance to parent is 1
            if node_l is None and node_r is None:
                node.val = 1
                return 1

            # recursively check left / right node
            d_l, d_r = 0, 0
            if node_l is not None:
                d_l = dfs(node_l)
            if node_r is not None:
                d_r = dfs(node_r)

            d = d_l + d_r
            global diameter
            diameter = max(d, diameter)
            return max(d_l, d_r) + 1

        dfs(root)
        return diameter
```

Python basics

Class

ClassName

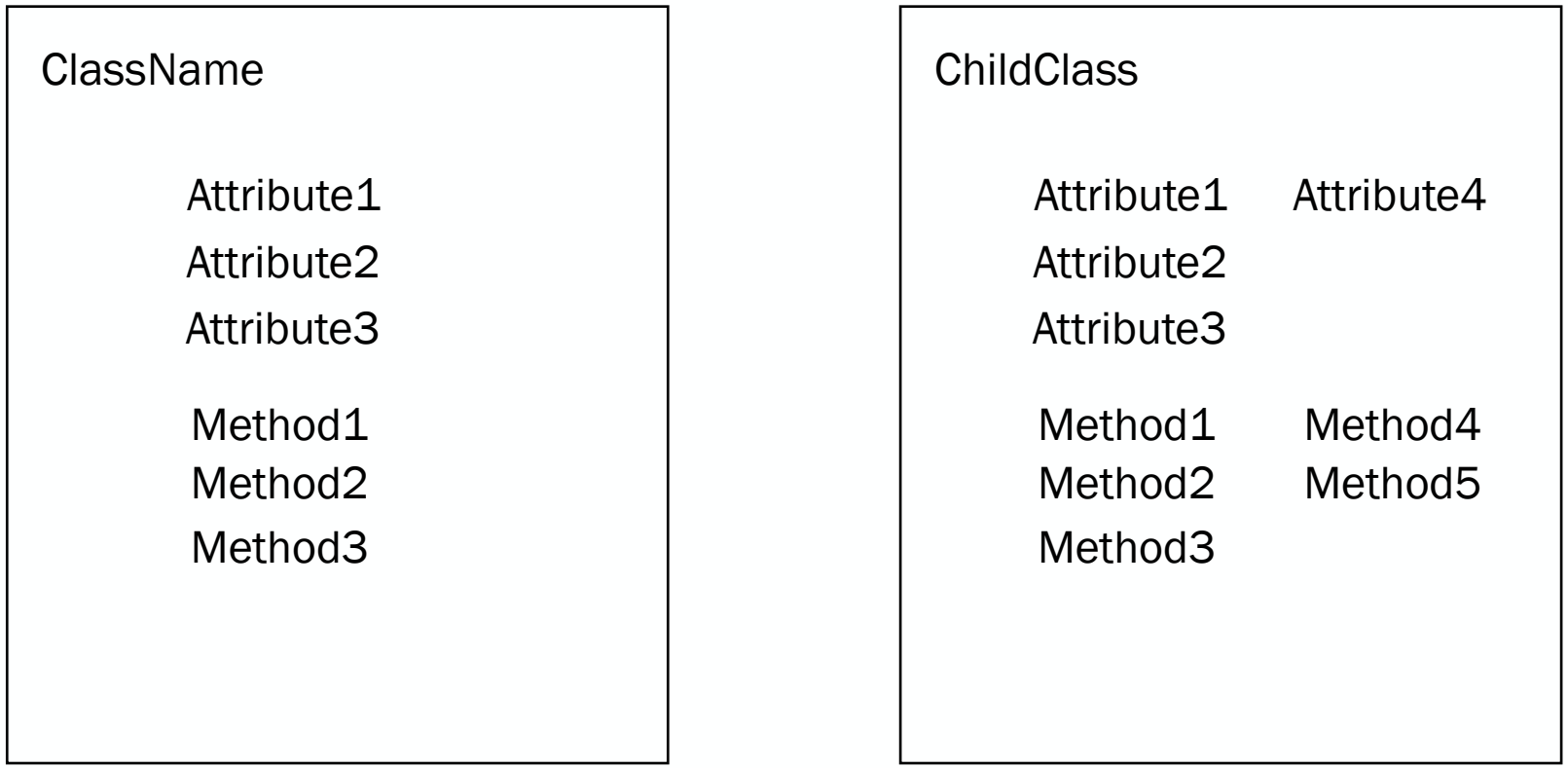
Attributes

Methods

```python
class Human:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
        print(f"My name is {self.name}. I am {self.age} years old.")

    def eat(self):
        print(f"{self.name} is eating.")
```

Python basics

Class, inherit



Python basics

Class, inherit

```python
class Human:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
        print(f"My name is {self.name}. I am {self.age} years old.")

    def eat(self):
        print(f"{self.name} is eating.")
```

```python
class Student(Human):
    def __init__(self, name, age, school, major):
        super().__init__(name, age)
        self.school = school
        self.major = major

    def study(self):
        print(f"{self.name} is studying {self.major} at {self.school}.")
```

```python
class Worker(Human):
    def __init__(self, name, age, company, job_title):
        super().__init__(name, age)
        self.company = company
        self.job_title = job_title

    def work(self):
        print(f"{self.name} works as a {self.job_title} at {self.company}.")
```

```python
student1 = Student("Alice", 20, "Purdue University", "Computer Science")
worker1 = Worker("Bob", 35, "Google", "Software Engineer")

student1.introduce()
student1.eat()
student1.study()

worker1.introduce()
worker1.eat()
worker1.work()
```

PURDUE UNIVERSITY

Job Interview in the AI-Era: Coding, Systems, Agents

Additional Materials

Wei Chen

05/20/2026

PURDUE UNIVERSITY®

Python basics

- Tuples: immutable lists
  - Can be used as dictionary keys
- Copy vs. reference
  - Deep copy: Nested structures -> need deep copy
- Functions
  - Lambda Functions
- Strings
- Memory & performance awareness

Problems: Arrays two pointers

- Best time to buy and sell stock
- Squares of a Sorted Array
- 3 sum
- Container with most water
- Longest Mountain in Array

Problems: Arrays sliding window

- Contains Duplicate II
- Minimum Absolute Difference
- Minimum Size Subarray Sum

Problems: Strings

- Valid palindrome
- Longest substring without repeating characters

Problems: Matrix & memory

- Set matrix zeroes

Problems: Backtracking

- Letter Case Permutation
- Subsets
- Combinations
- Permutations

Problems: Linked lists

- Middle of Linked List
- Linked List Cycle
- Reverse Linked List
- Remove Linked List Elements
- Reverse Linked List II
- Palindrome Linked List
- Merge Two Sorted Lists

PURDUE UNIVERSITY

Problems: HashMap

- Valid anagram
- Group anagrams
- Topk frequent elements

Problems: Stacks

- Min Stack
- Valid Parentheses
- Evaluate Reverse Polish Notation
- Stack Sorting

Problems: Queues

- Implement Stack using Queues
- Time Needed to Buy Tickets
- Reverse the First K Elements of a Queue

Problems: Binary trees

- Average of Levels in Binary Tree
- Minimum Depth of Binary Tree
- Maximum Depth of Binary Tree
- Min/Max Value Binary Tree
- Binary Tree Level Order Traversal
- Same Tree
- Path Sum
- Diameter of a Binary Tree
- Invert Binary Tree
- Lowest Common Ancestor of a Binary Tree

Problems: Binary search trees

- Search in a Binary Search Tree
- Insert into a Binary Search Tree
- Convert Sorted Array to Binary Search Tree
- Two Sum IV - Input is a BST
- Lowest Common Ancestor of a Binary Search Tree
- Minimum Absolute Difference in BST
- Balance a Binary Search Tree
- Delete Node in a BST
- Kth Smallest Element in a BST

Problems: Heaps

- Kth Largest Element in an Array
- K Closest Points to Origin
- Top K Frequent Elements
- Task Scheduler

Problems: Graphs

- Breadth and Depth First Traversal
- Clone Graph
- Core Graph Operations
- Cheapest Flights Within K Stops
- Course Schedule

Problems: Combined patterns

- Subarray sum equals K
- Product of array except self
- Daily temperatures
- LRU Cache