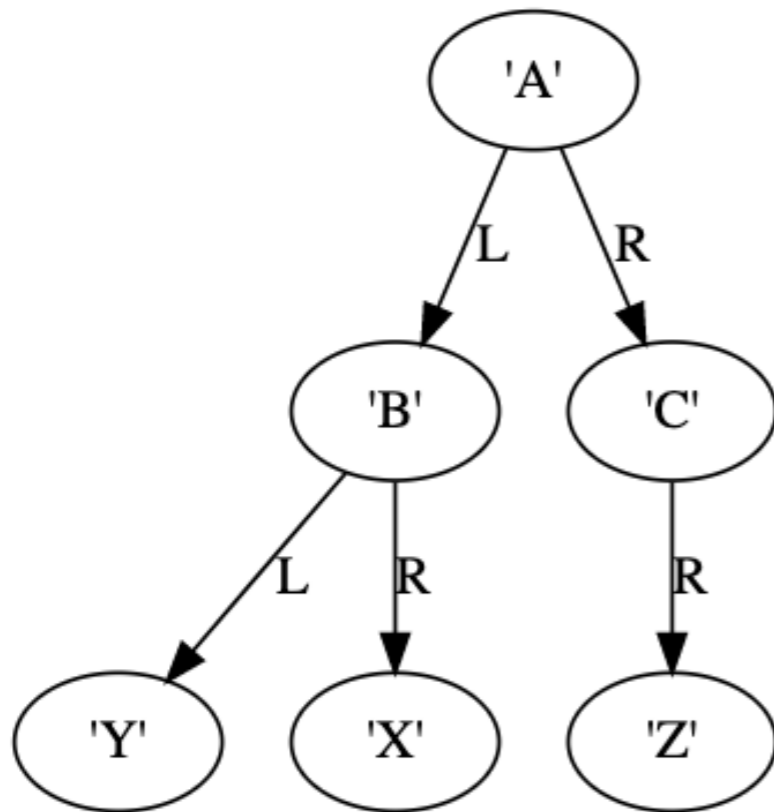# [320] Implementing Various Graph Structures

Tyler Caraza-Harter

Review

```
def contains(node, target):
    if node == None:
        return False

    if node.val == target:
        return True

    return (contains(node.left, target) or
            contains(node.right, target))
```


How many nodes will **contains(root, "Z")** check?
1. one
2. six

What will **contains(root, "C")** check first?
1. node X
2. node C

How many nodes will **contains(root, "C")** check?
1. five
2. six

Hierarchy of Graphs

Graph: nodes+edges

Directed Graph: graph with
- one-way edges

DAG: directed graph that
- does not have cycles

Tree: DAG that
- has exactly one root
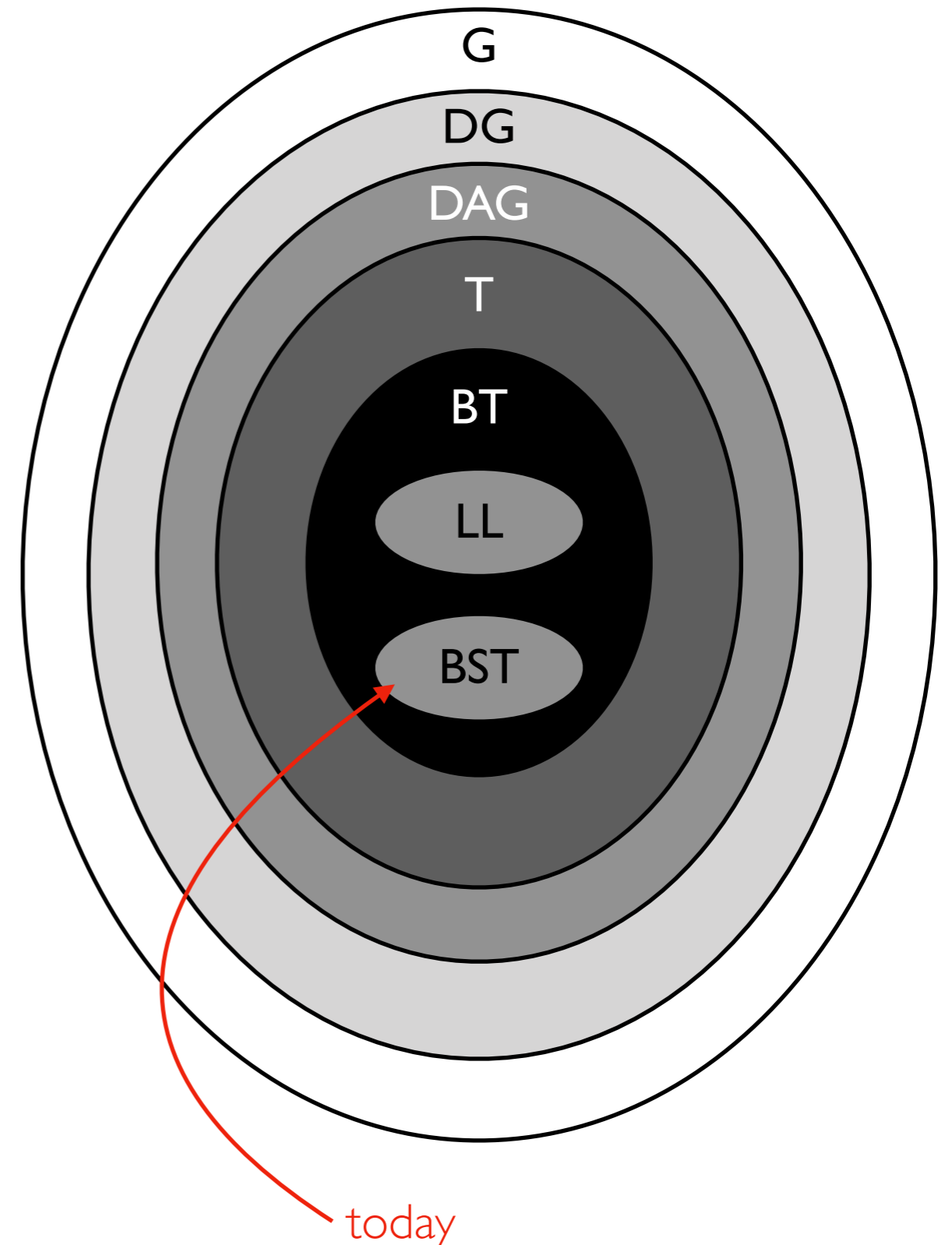- non-roots have exactly one parent

Binary Tree: tree such that
- nodes have at most 2 children

Linked List: tree such that
- nodes have at most 1 child

BST: tree such that
- vals in left subtree < parent val
- parent val < vals in right subtree

G

DG

DAG

T

BT

LL

BST

today

Hierarchy of Graphs

Graph: nodes+edges

Directed Graph: graph with
- one-way edges

D<u>A</u>G: directed graph that
- does not have cycles

Tree: DAG that
- has exactly one root
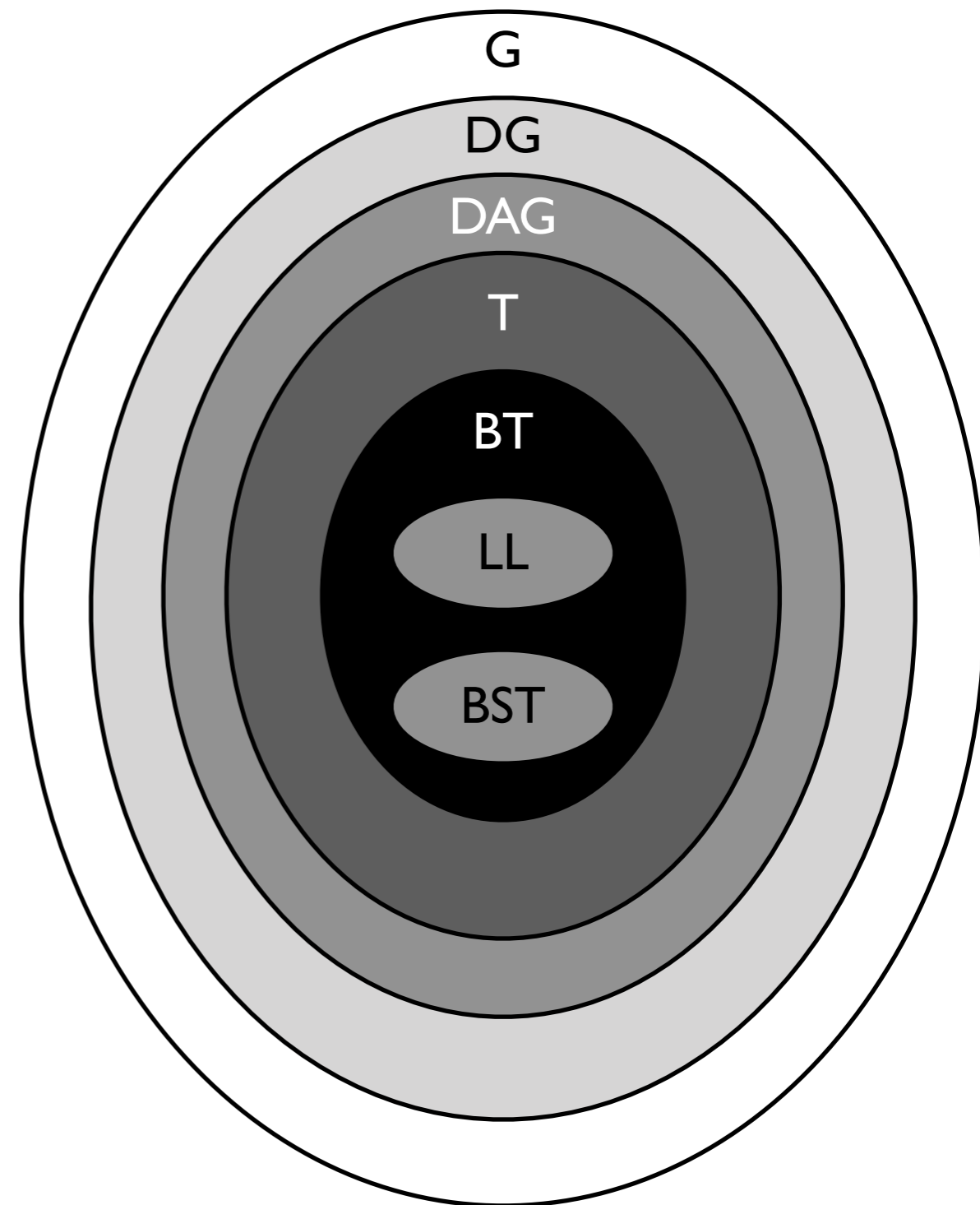- non-roots have exactly one parent

Binary Tree: tree such that
- nodes have at most 2 children

Linked List: tree such that
- nodes have at most 1 child

B<u>S</u>T: tree such that
- vals in left subtree < parent val
- parent val < vals in right subtree

all these are "weakly connected"

G
DG
DAG
T
BT
LL
BST

Weakly Connected

Graph: nodes+edges

Directed Graph: graph with
- one-way edges

DAG: directed graph that
- does not have cycles

Tree: DAG that
- has exactly one root
- non-roots have exactly one parent

Binary Tree: tree such that
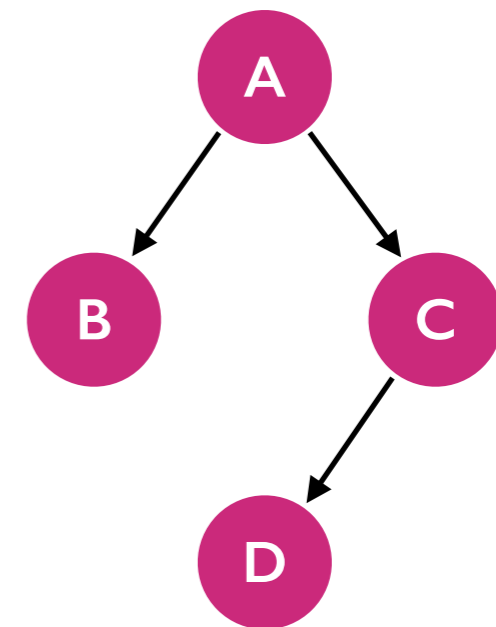- nodes have at most 2 children

Linked List: tree such that
- nodes have at most 1 child

BST: tree such that
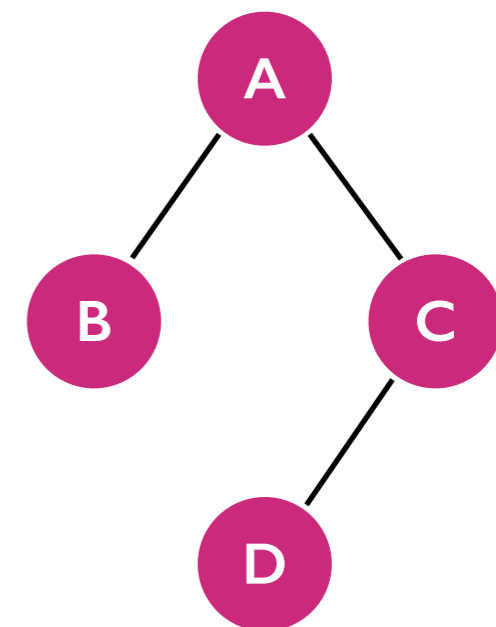- vals in left subtree < parent val
- parent val < vals in right subtree

all these are "weakly connected"

Not technically connected
because no D → A path

it is weakly connected
because there is a path
between every pair if we
ignore edge direction

Node Attributes

Graph: nodes+edges

Directed Graph: graph with
- one-way edges

D<u>A</u>G: directed graph that
- does not have cycles

Tree: DAG that
- has exactly one root
- non-roots have exactly one parent

Binary Tree: tree such that
- nodes have at most 2 children

Linked List: tree such that
- nodes have at most 1 child

B<u>S</u>T: tree such that
- vals in left subtree < parent val
- parent val < vals in right subtree

```
class Node:
    def __init__(self, val):
        self.next = None
        ...
```
A

```
class Node:
    def __init__(self, val):
        self.children = []
        ...
```
B

```
class Node:
    def __init__(self, val):
        self.left = None
        self.right = None
        ...
```
C

Implementing Graphs: Classes and Attributes

Nodes:
- usually have class for this

Edges:
- often just an attribute in a Node
- if there is edge metadata, might be a separate class just for this

Graph:
- often have a class for this to handle various cases:
  - graphs with zero nodes
  - graphs with multiple roots
  - enforce constraints (if not directed, edges come in pairs)

```python
class Graph: # undirected
    def __init__(self):
        self.nodes = {}

    def add_node(self, name, val):
        self.nodes[name] = Node(name, val)

    def add_edge(self, name1, name2):
        node1 = self.nodes[name1]
        node2 = self.nodes[name2]
        node1.children.append(node2)
        node2.children.append(node1)
```