

# LECTURE 20

## BINARY SEARCH TREES

MCS 275 Spring 2021

Emily Dumas

# LECTURE 20: BINARY SEARCH TREES

## Course bulletins:

- Project 2 due 6pm CST today.
- Starting on worksheet 8, problem 1 will have special instructions.
  - Tue discussion: Problem 1 will be presented as an example.
  - Thu discussion: Please do problem 1 ahead of time.

# SAMPLE CODE

I've created a new directory `trees` in the course sample code repository.

Live coding examples from the next couple of lectures will be added there.

# GOAL

Learn about **search** and **insert** operations on binary search trees.

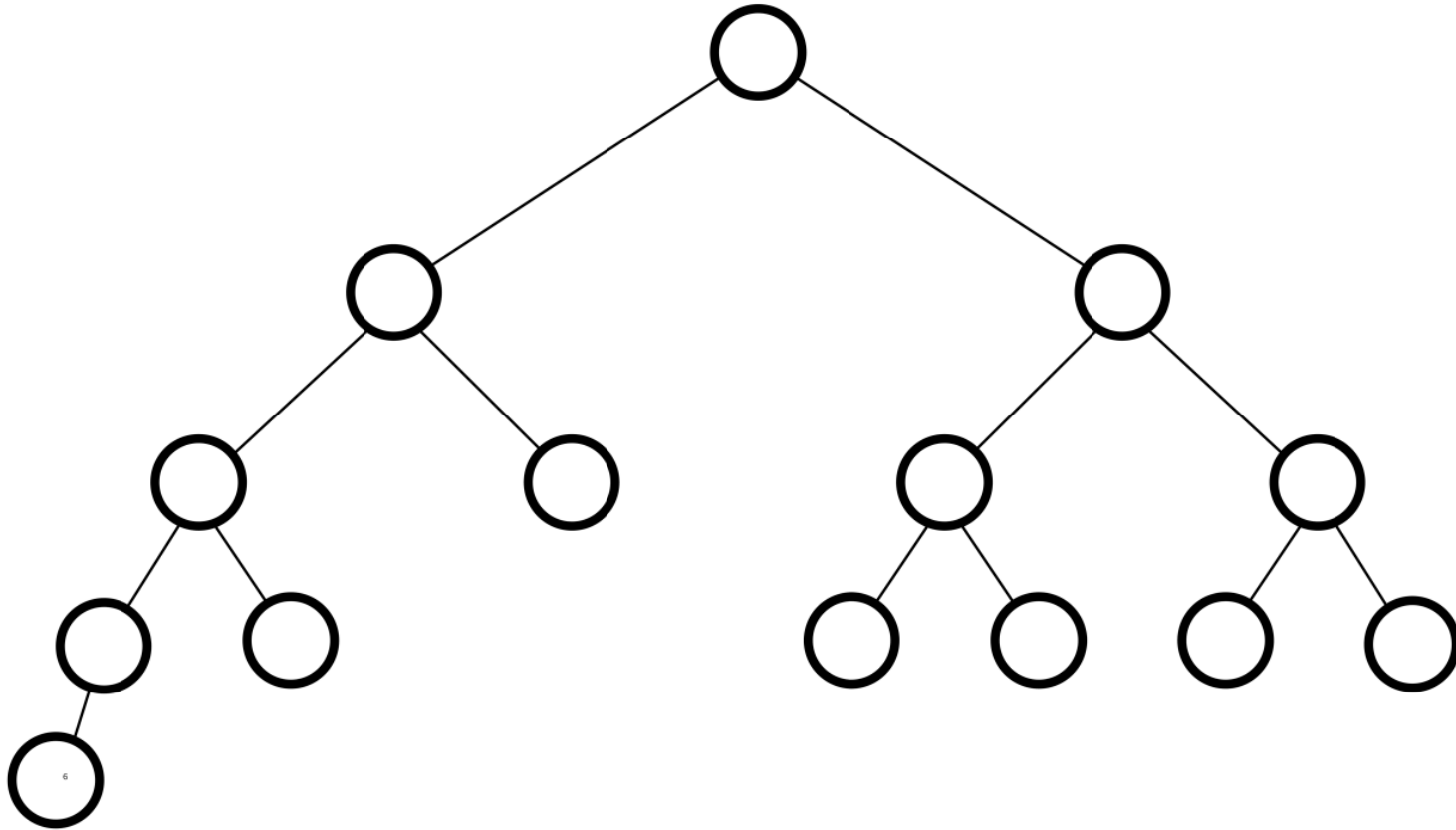
Explore an application to a fast data structure for storing a set of integers.

# BINARY SEARCH TREE (BST)

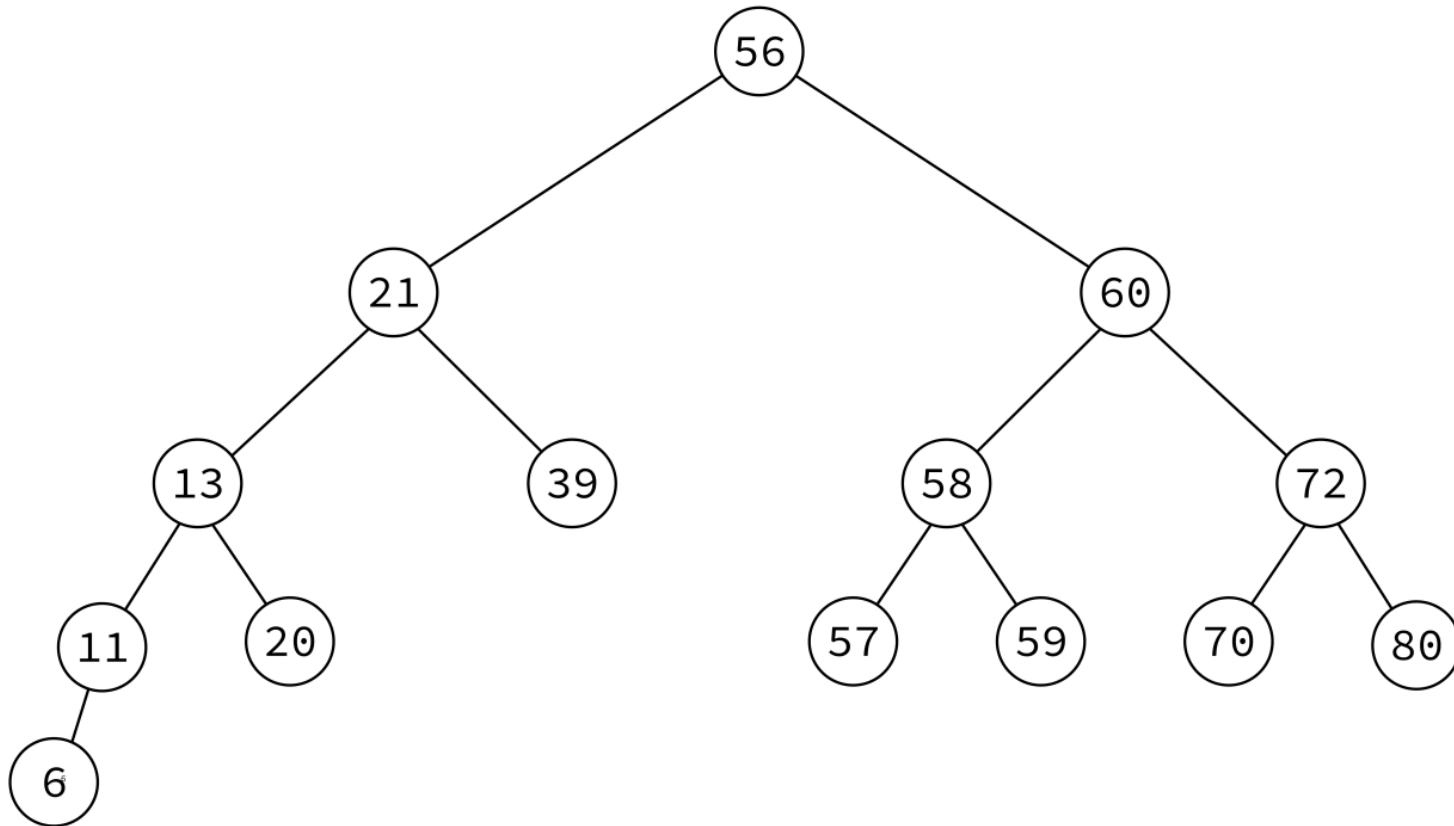
A binary tree in which:

- Nodes have **keys** that can be compared
- The key of a node is greater than or equal to any key in its left subtree.
- The key of a node is less than or equal to any key in its right subtree.

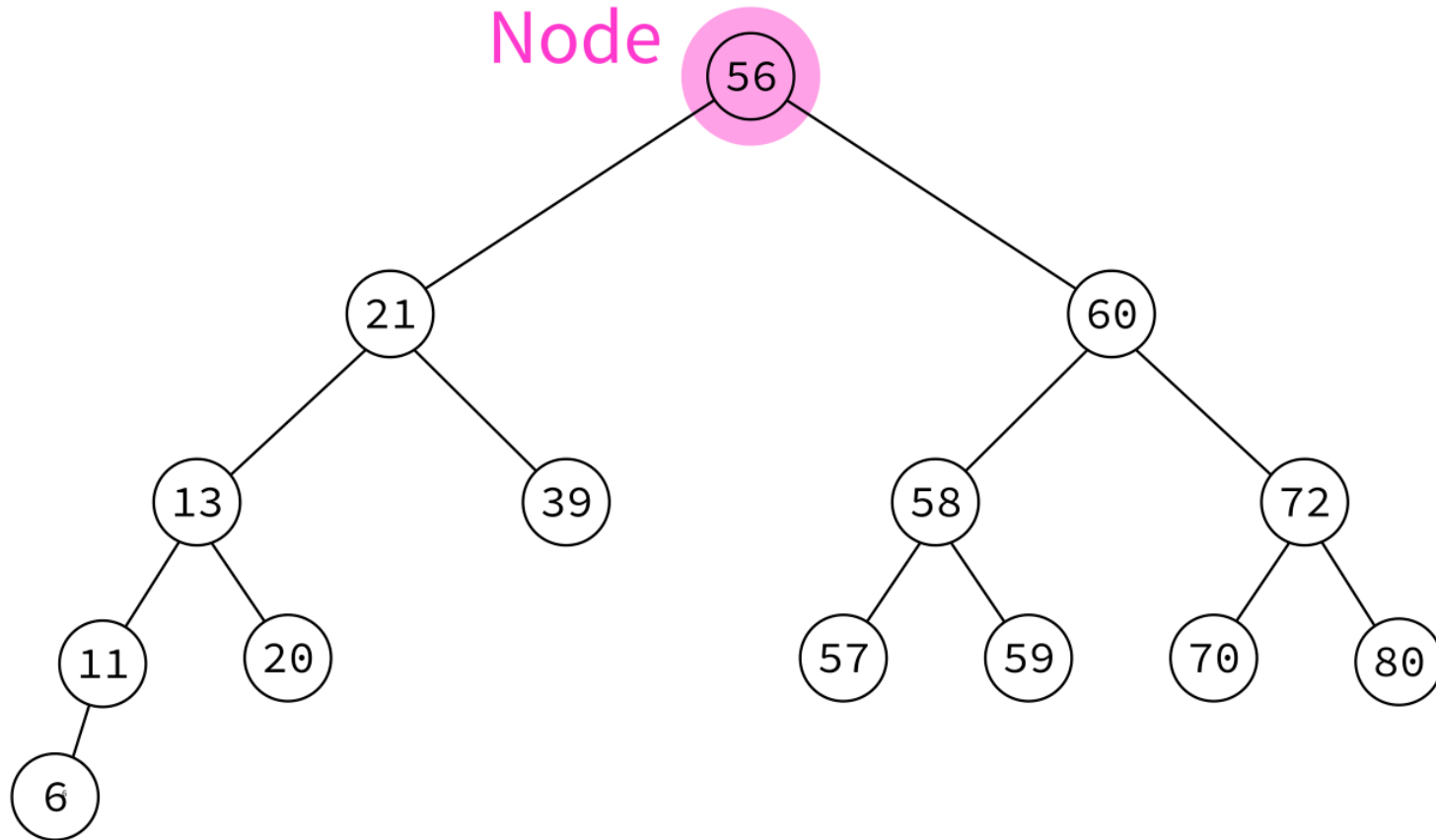
# BINARY TREE



# BST

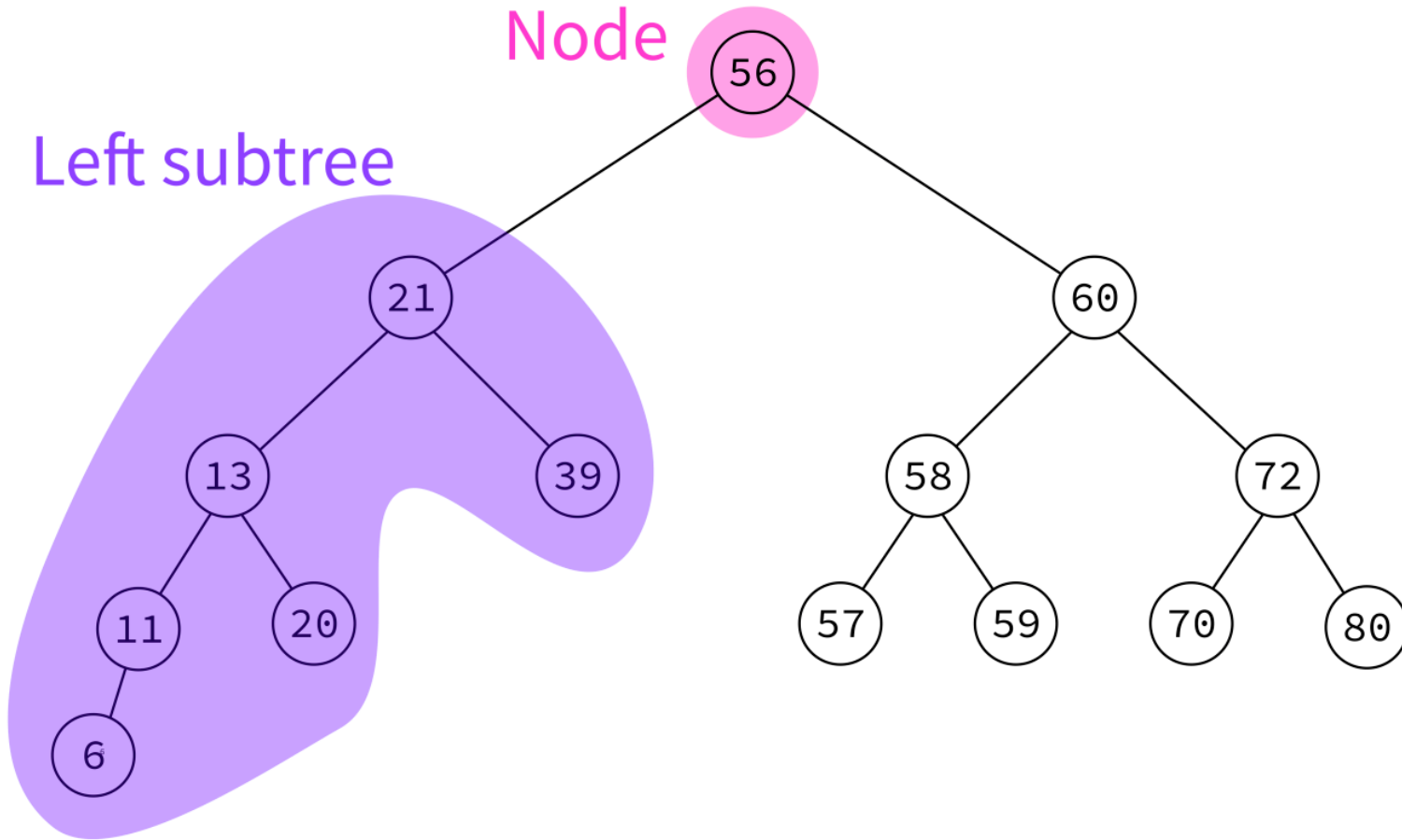


# BST

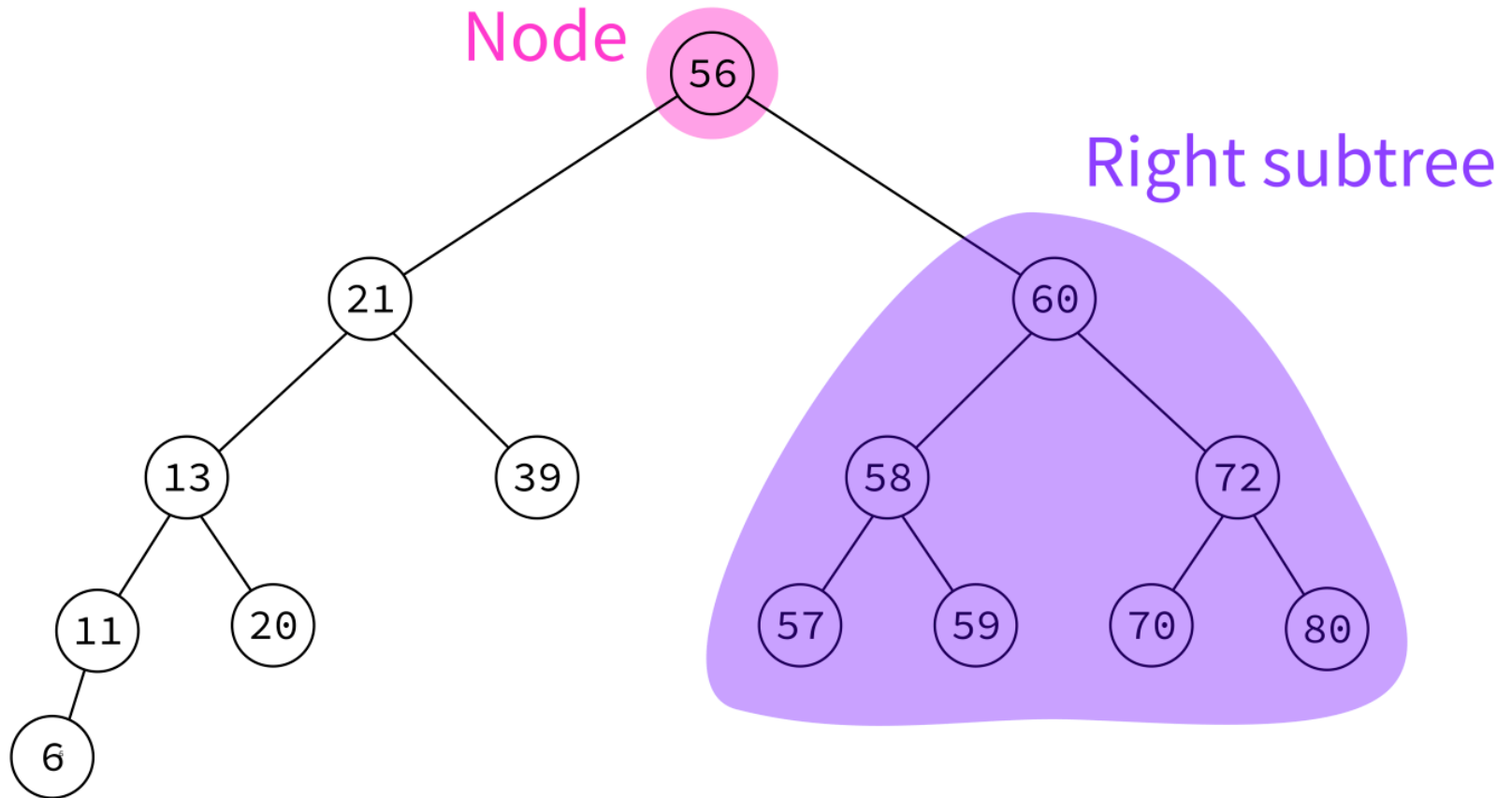




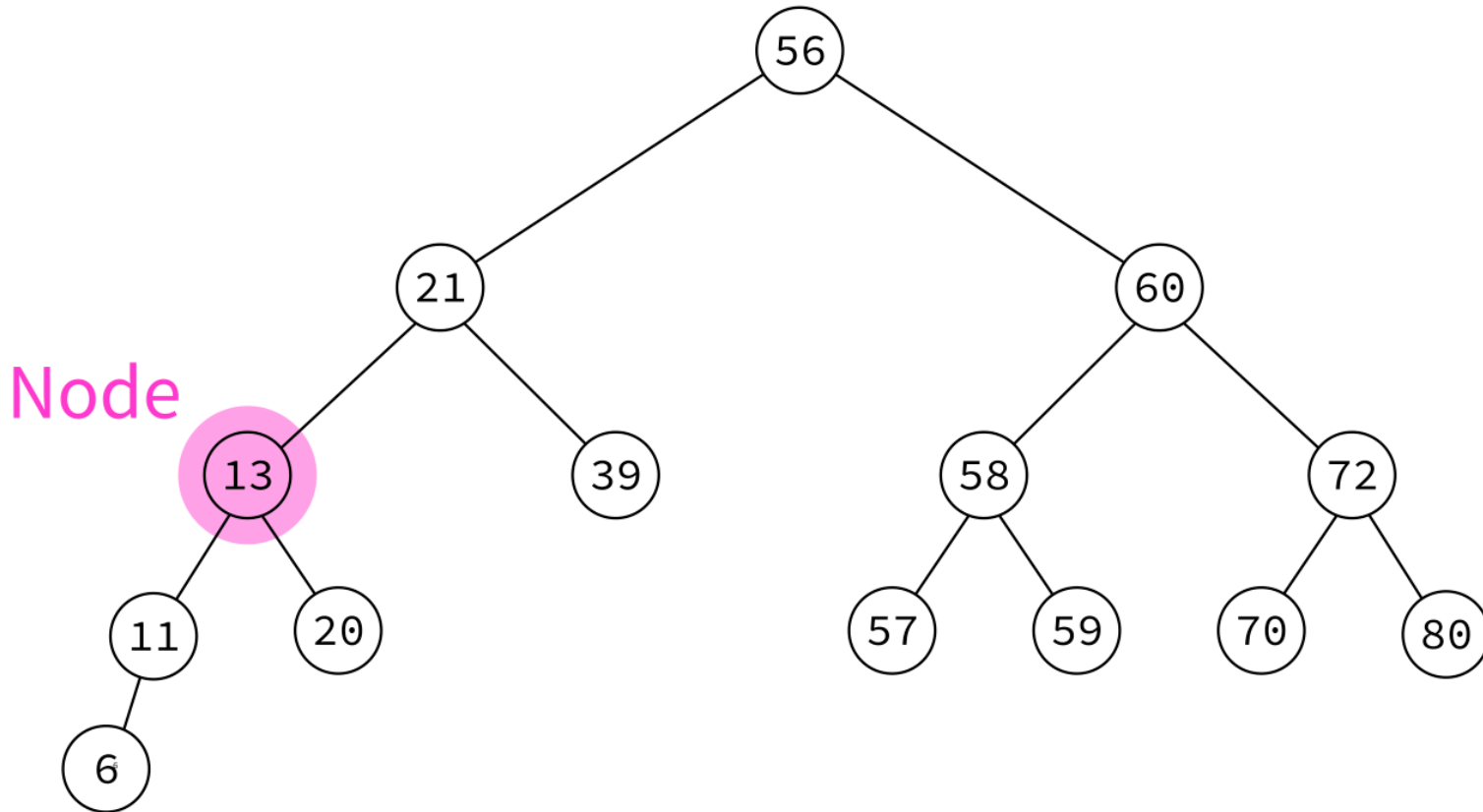
# BST



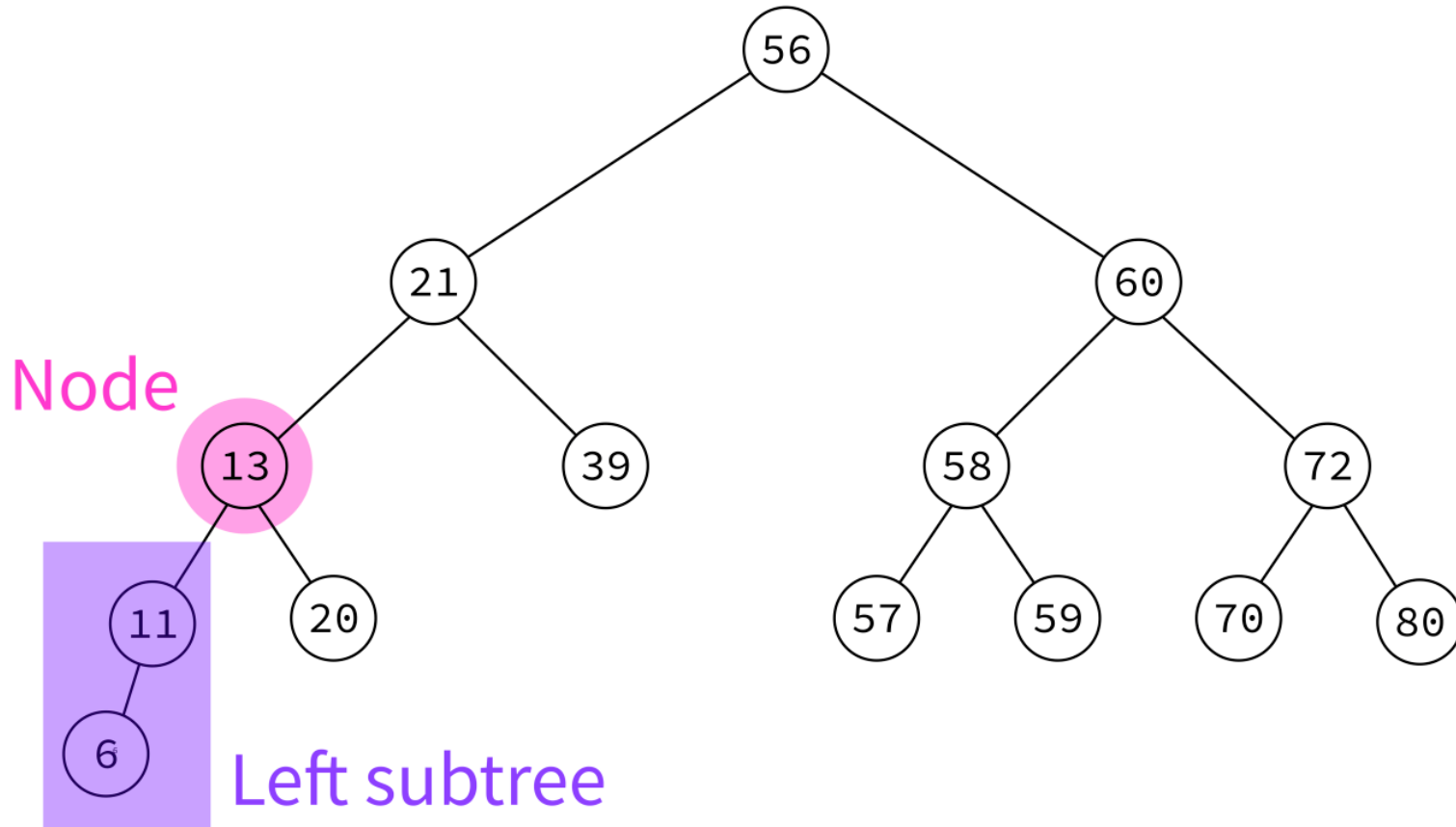
# BST



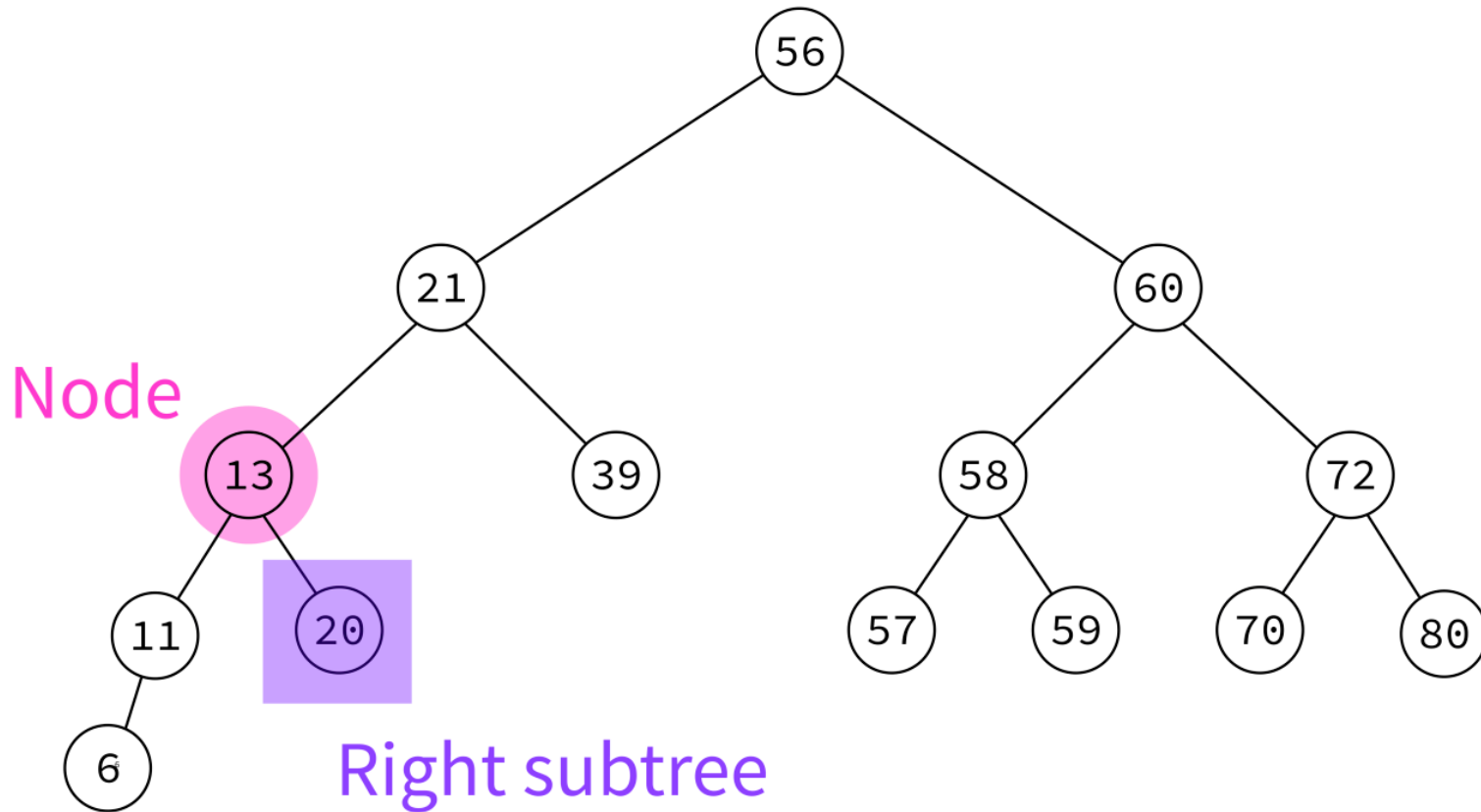
# BST



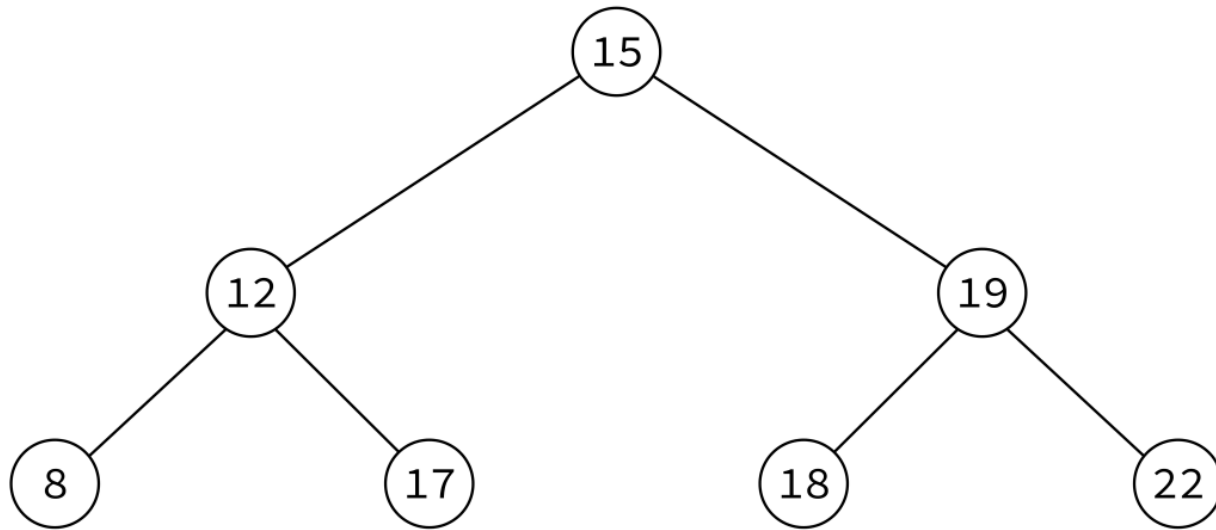
# BST



# BST

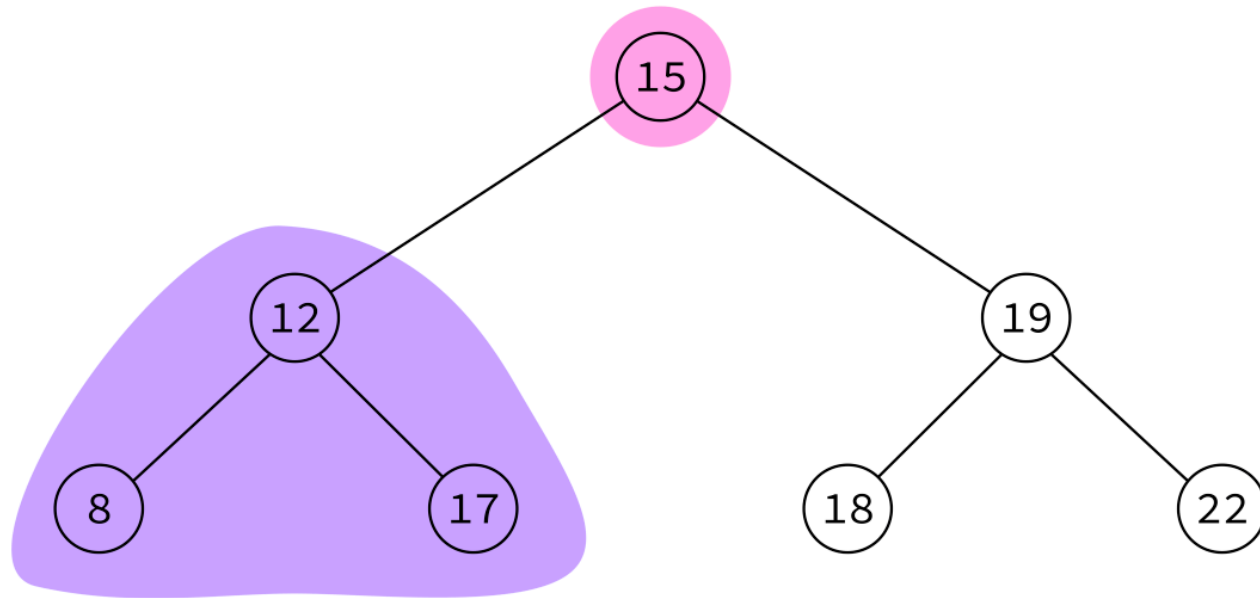


# NOT A BST



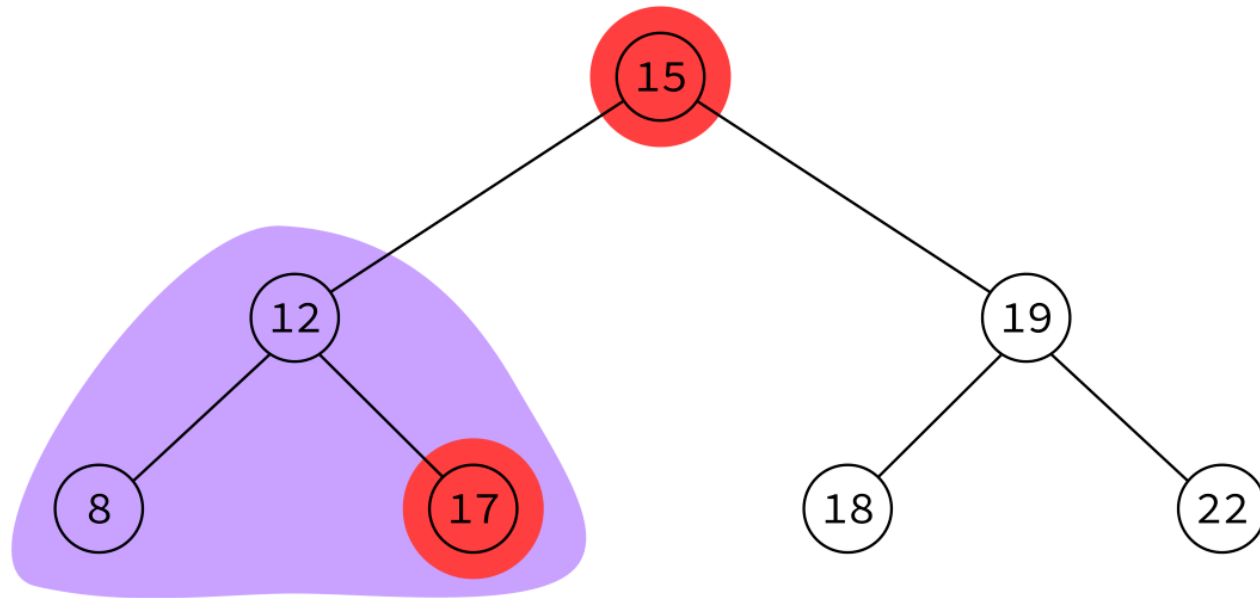
This "just" is a binary tree with keys.

# NOT A BST



This "just" is a binary tree with keys.

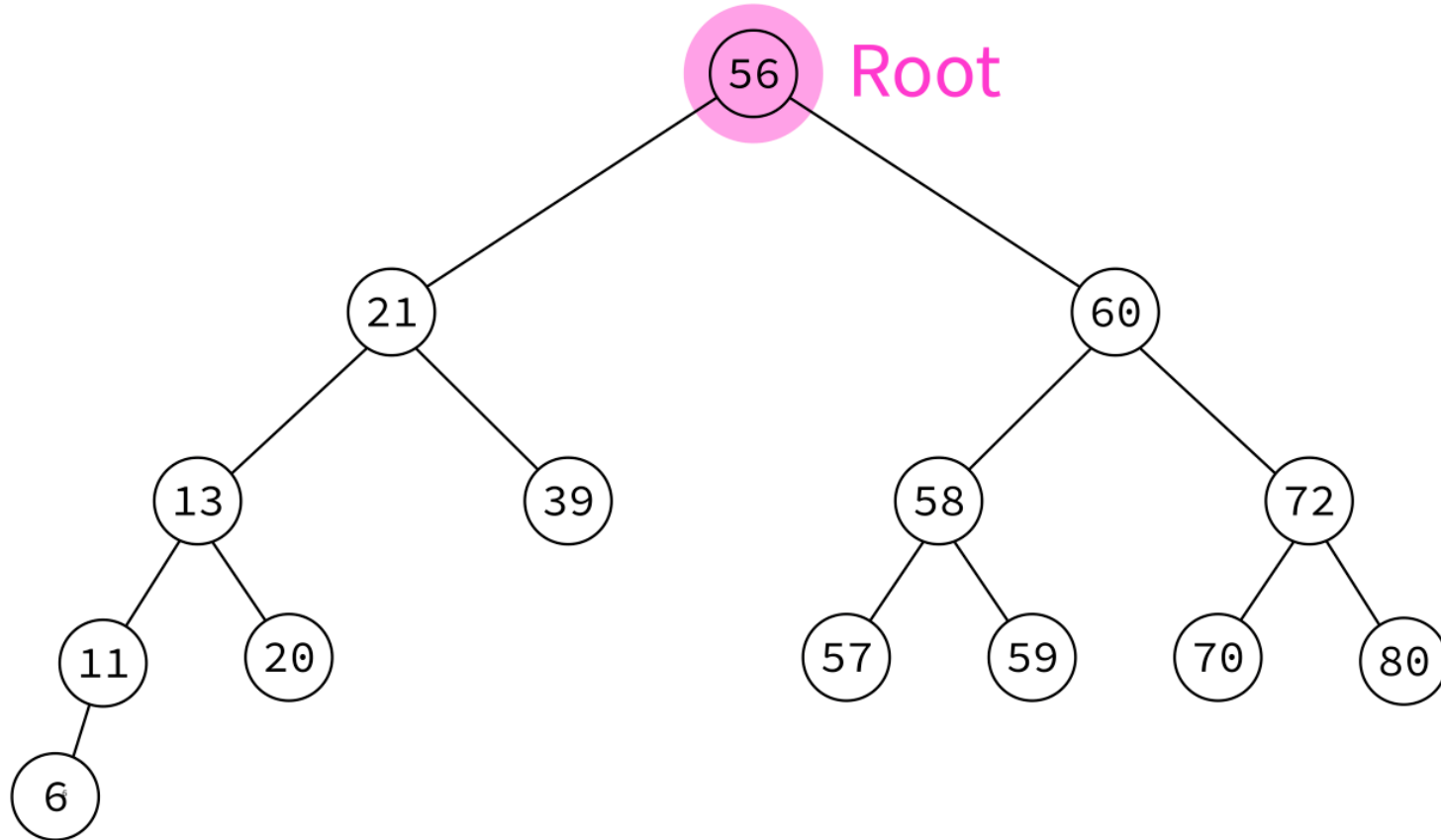
# NOT A BST



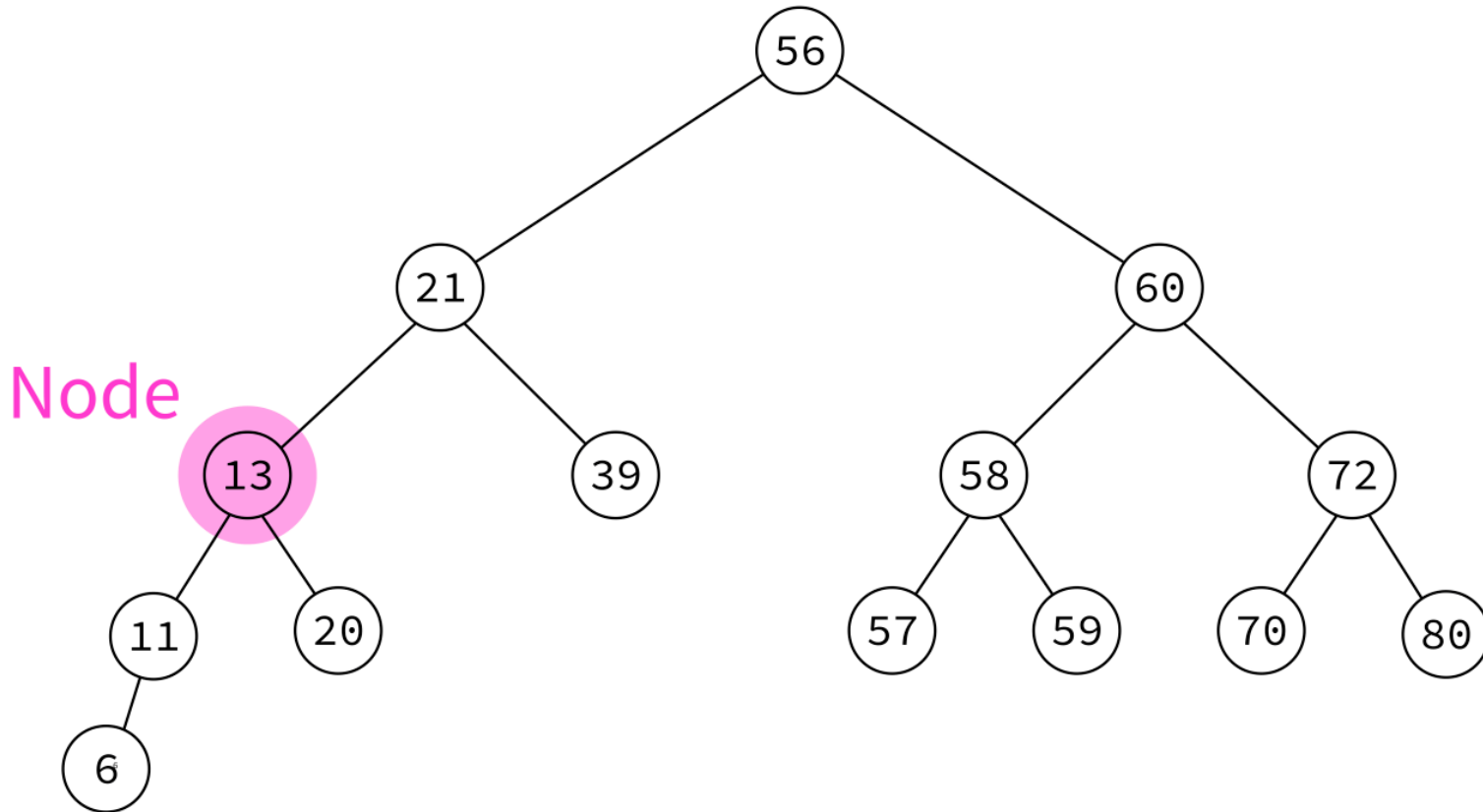
This "just" is a binary tree with keys.



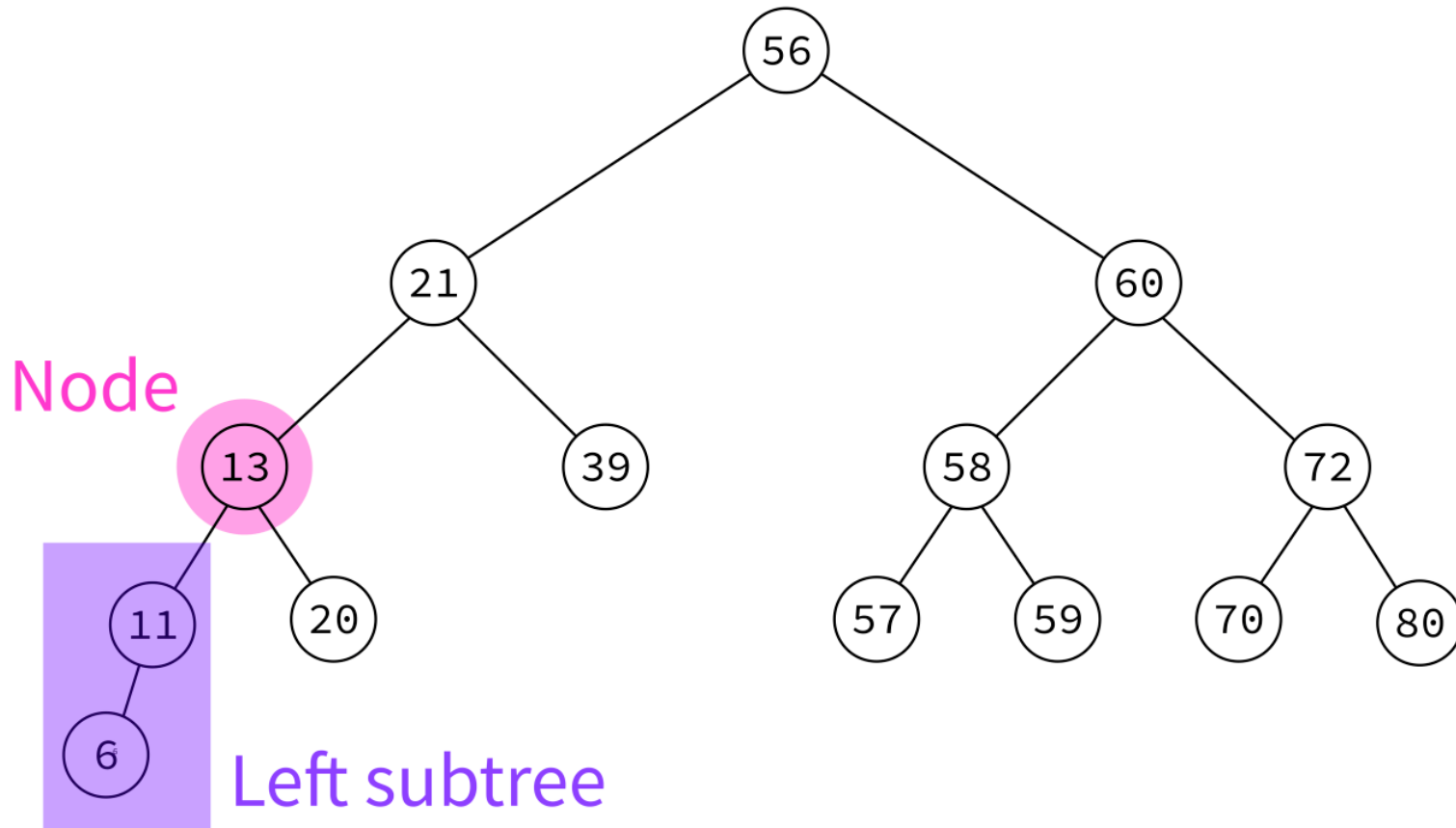
# TREE TERMS



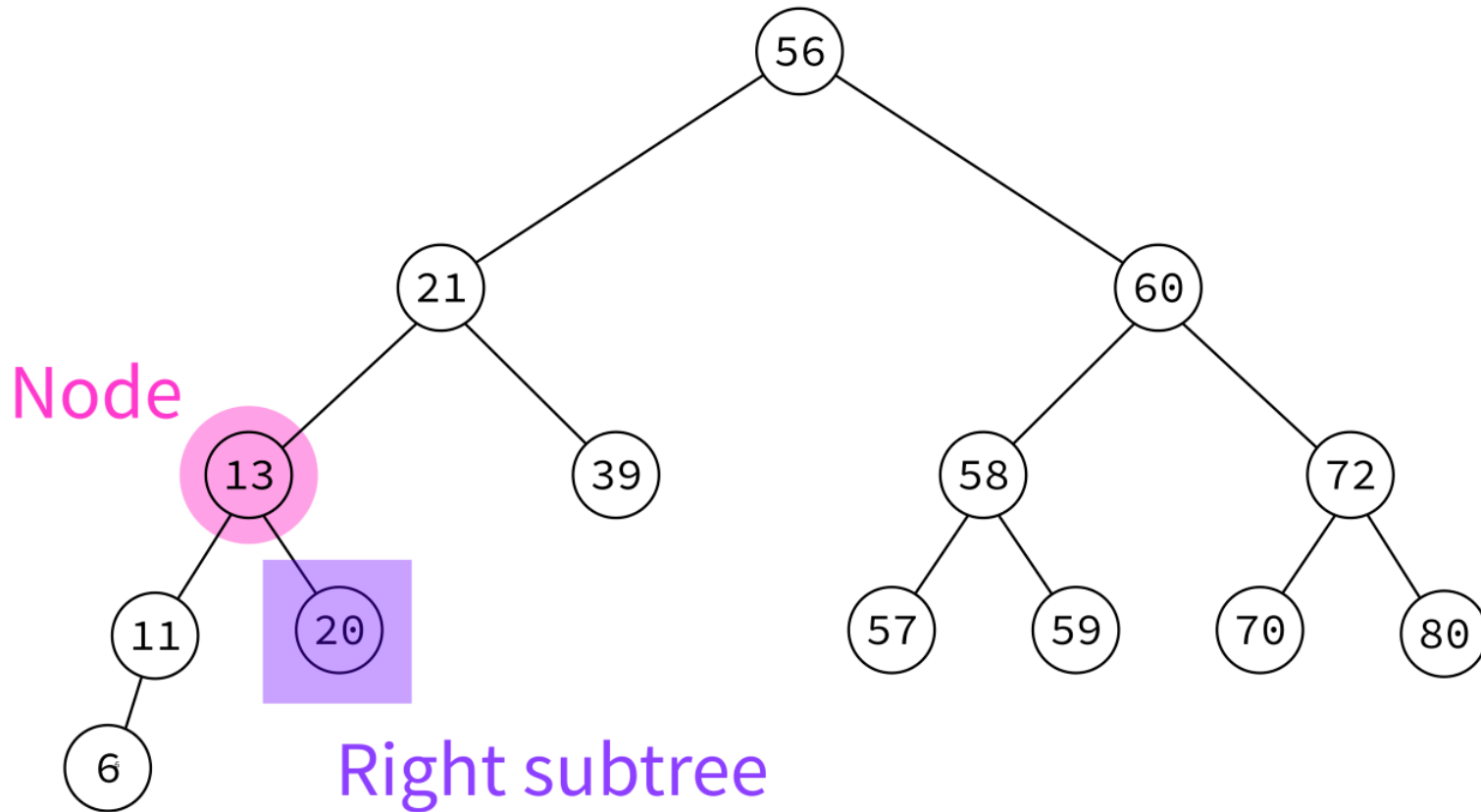
# TREE TERMS



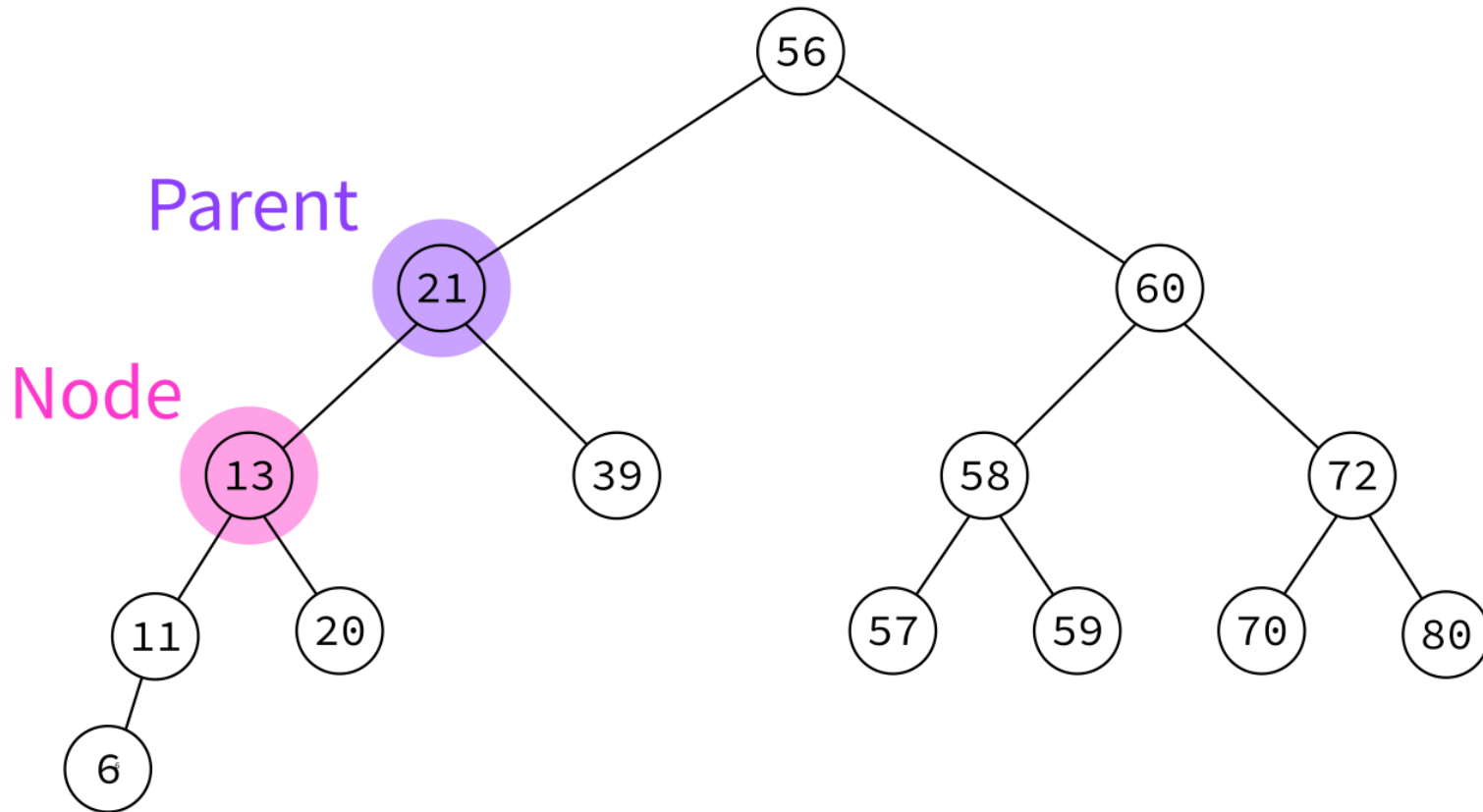
# TREE TERMS



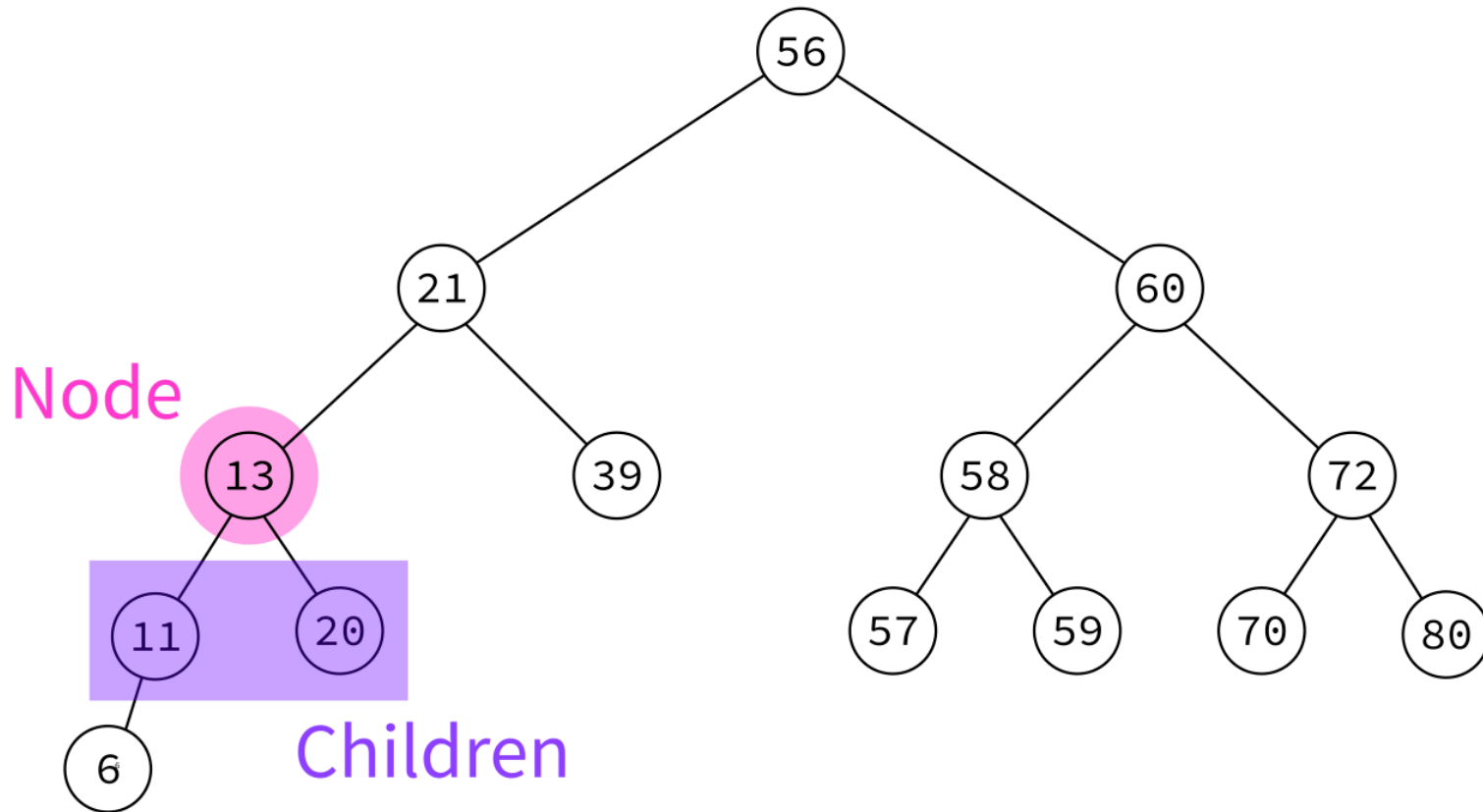
# TREE TERMS



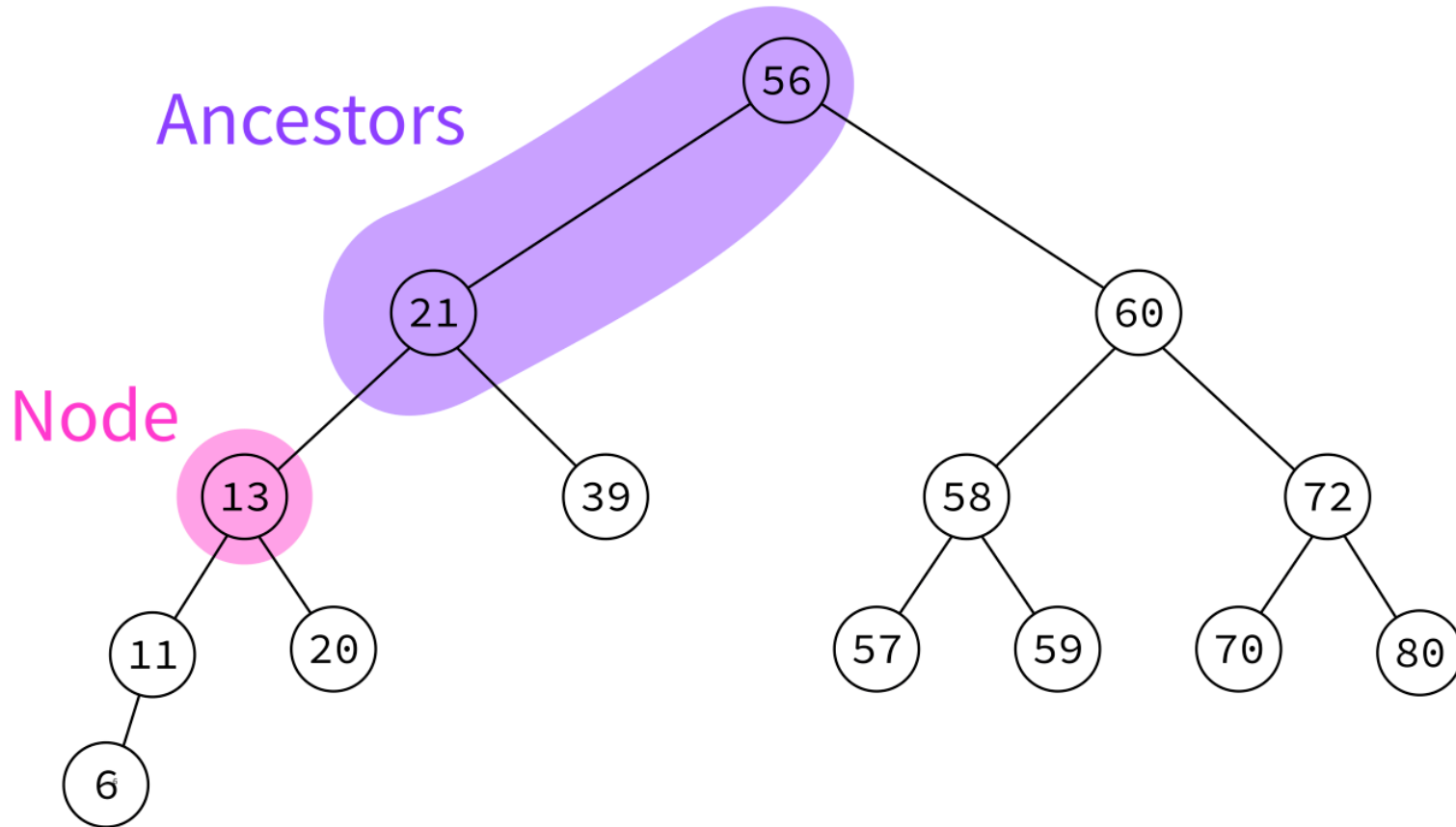
# TREE TERMS



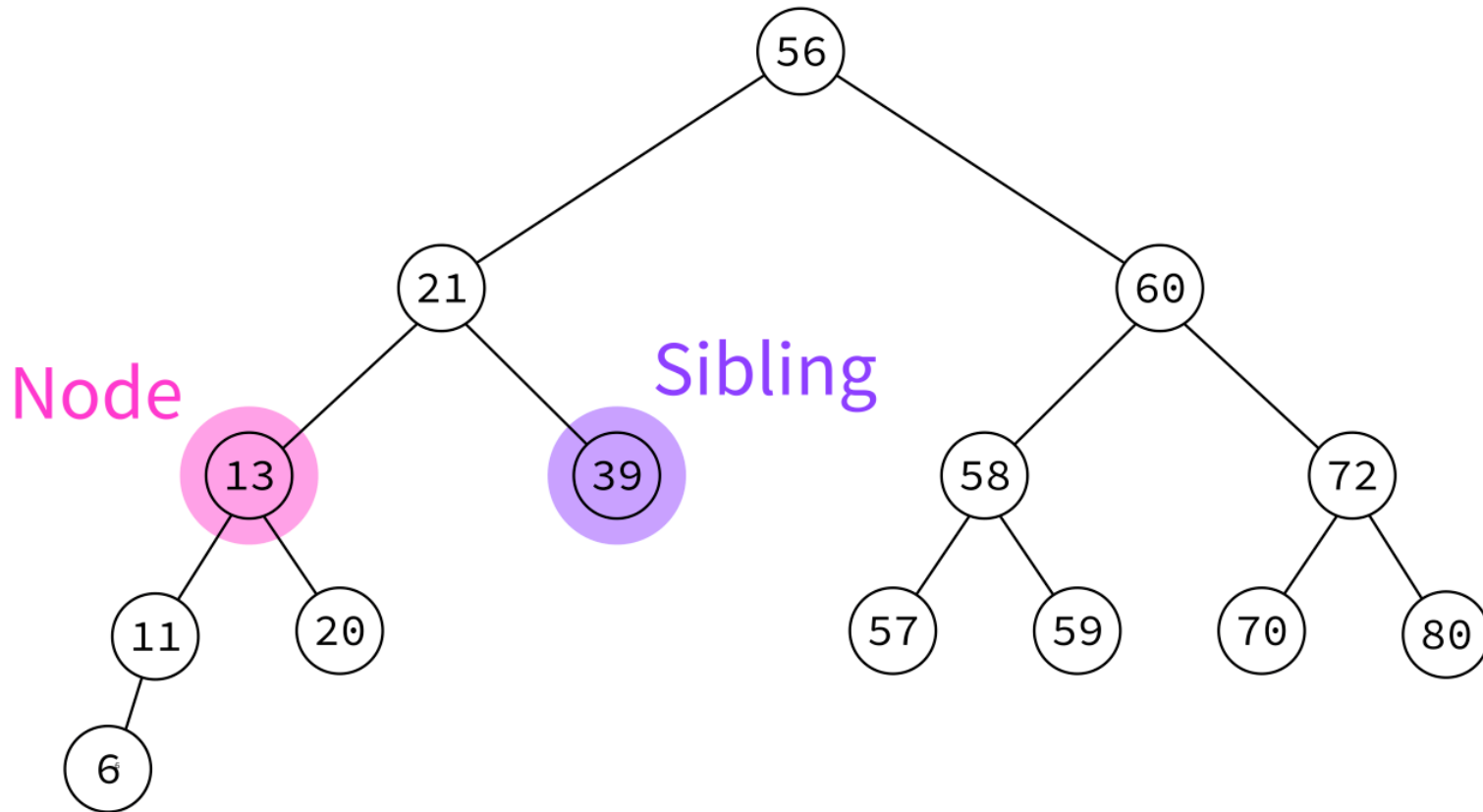
# TREE TERMS



# TREE TERMS

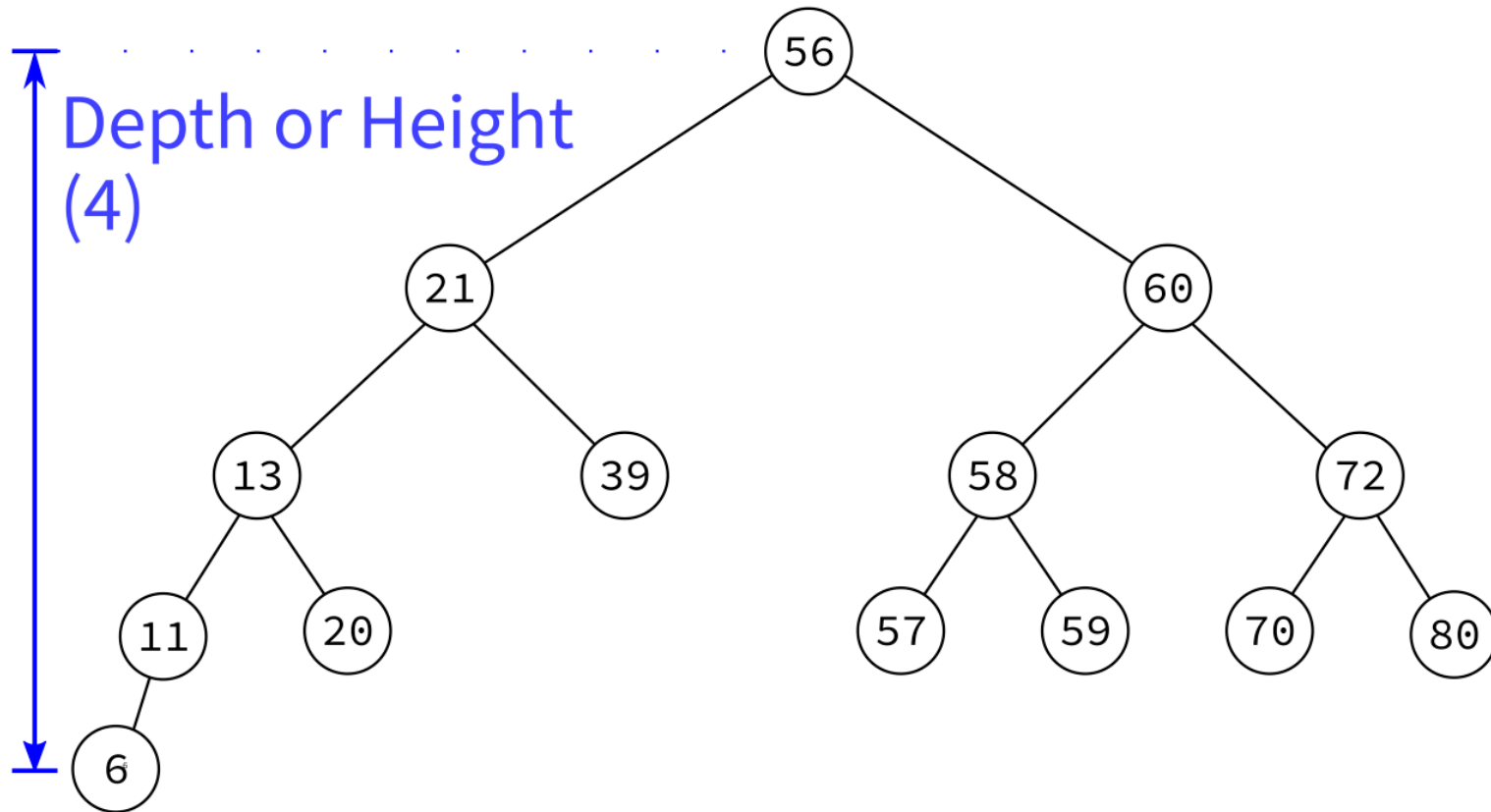


# TREE TERMS





# TREE TERMS



# CODING

Let's build a class to represent nodes of a binary tree that also store keys.

# TREEVIS

I provide a module `treevis` in the sample code repository that can "pretty print" a tree with the function `treeprint(root)`.

Challenge: Read the source of `treevis` and figure out how it works!

# FROM TREE TO BST

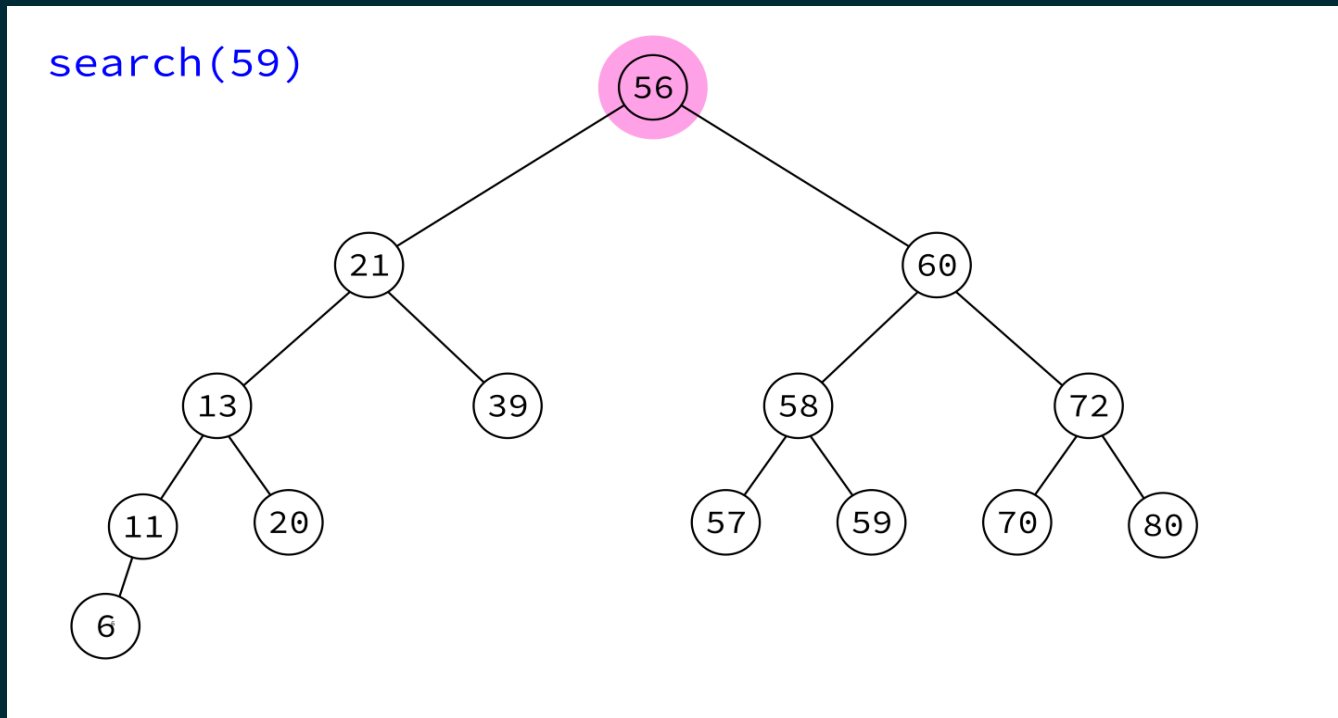
Now let's build a class that builds and manages a BST made of `Node` objects.

Desired features:

- Insert nodes (maintaining BST property)
- Search for nodes by key

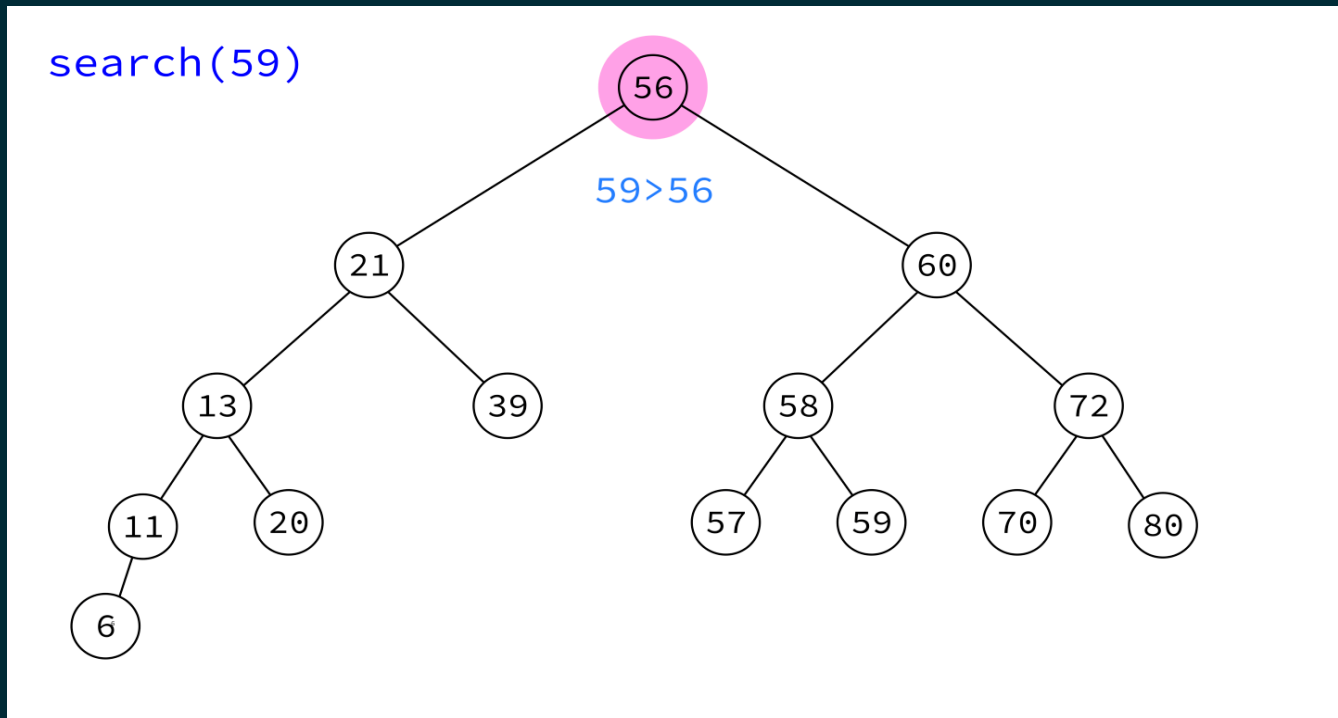
# SEARCH

Given  $x$ , find and return a node with key  $x$ . Return `None` if no such node exists.



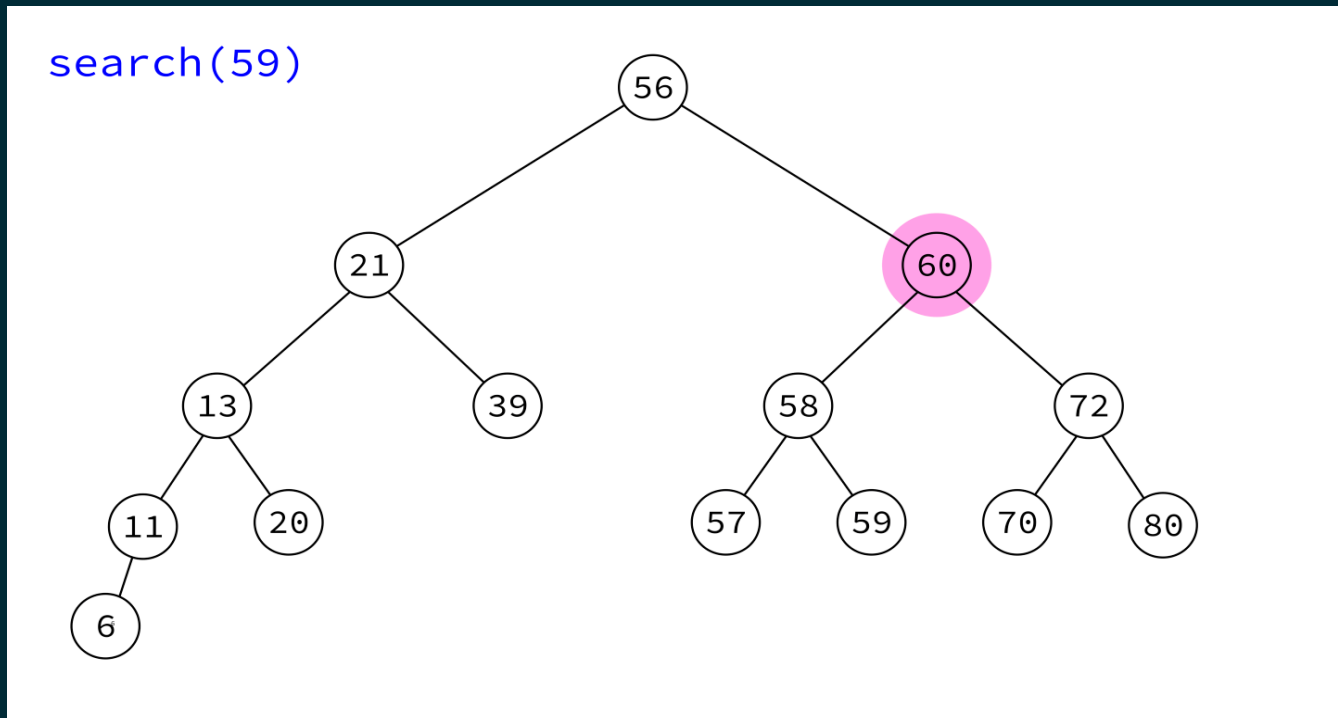
# SEARCH

Given  $x$ , find and return a node with key  $x$ . Return `None` if no such node exists.



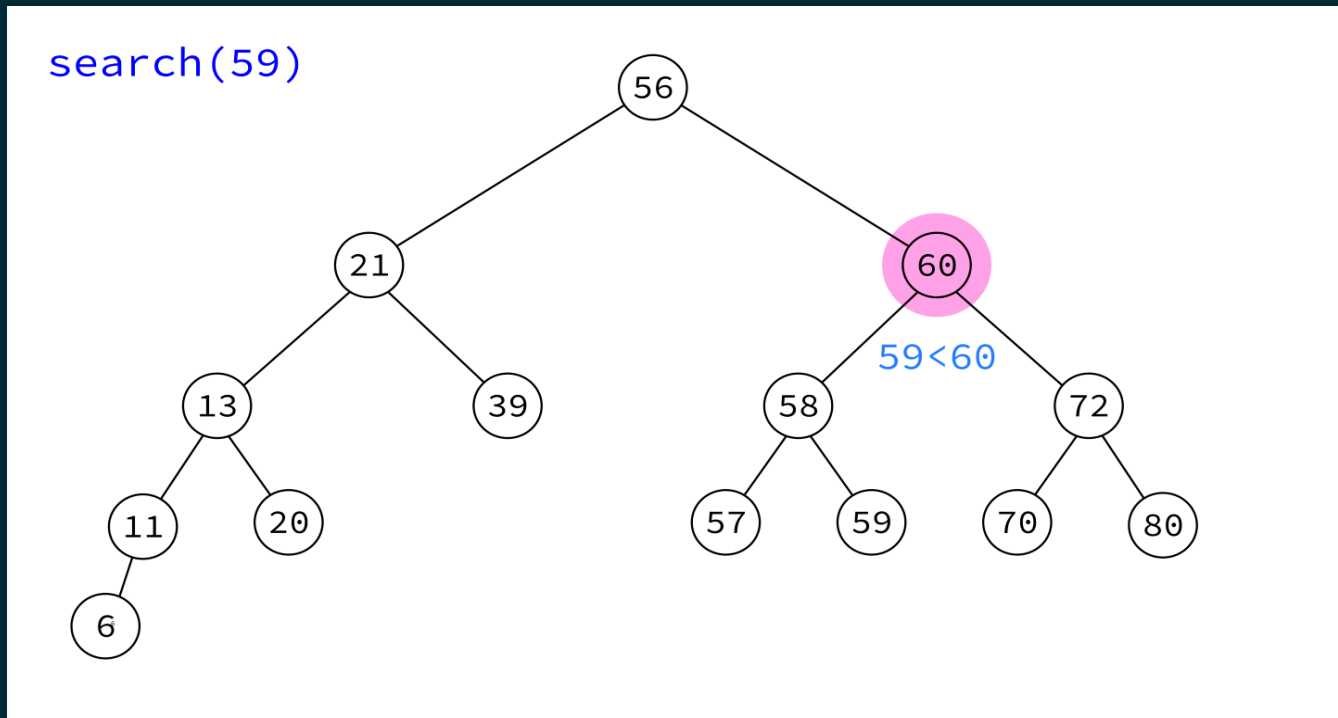
# SEARCH

Given  $x$ , find and return a node with key  $x$ . Return `None` if no such node exists.



# SEARCH

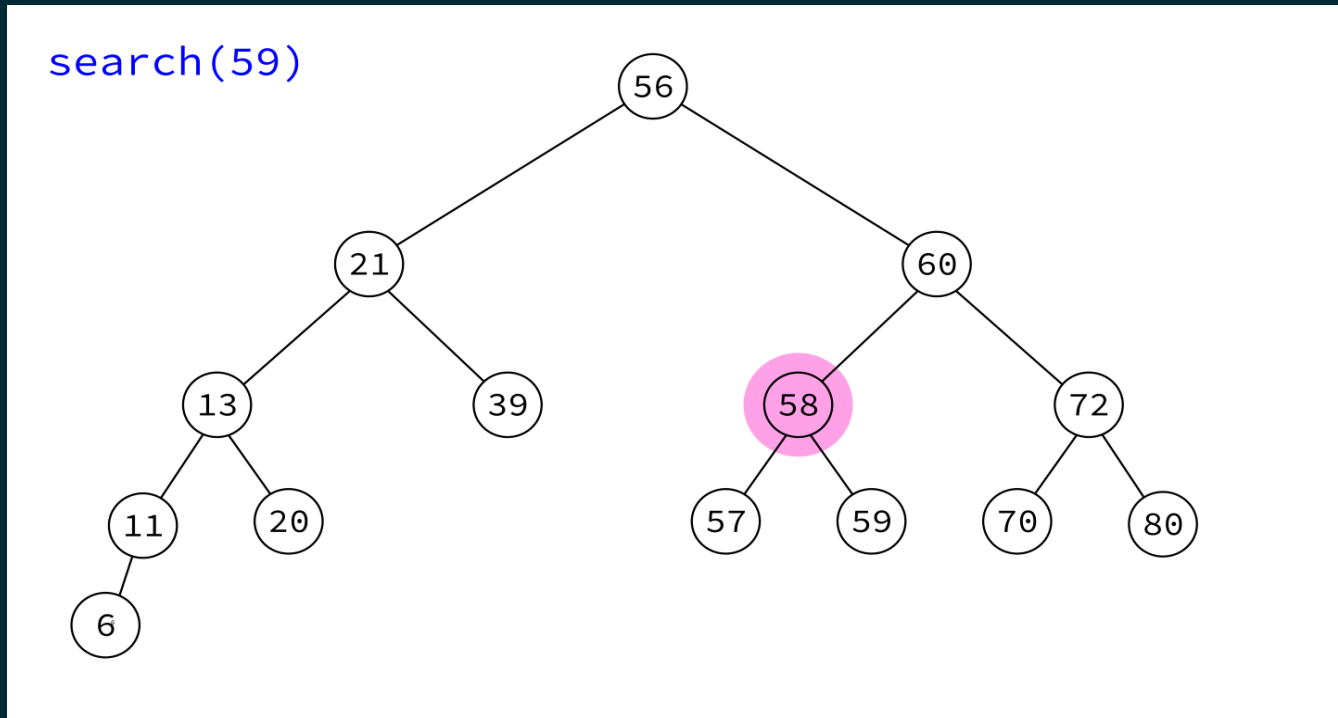
Given  $x$ , find and return a node with key  $x$ . Return `None` if no such node exists.





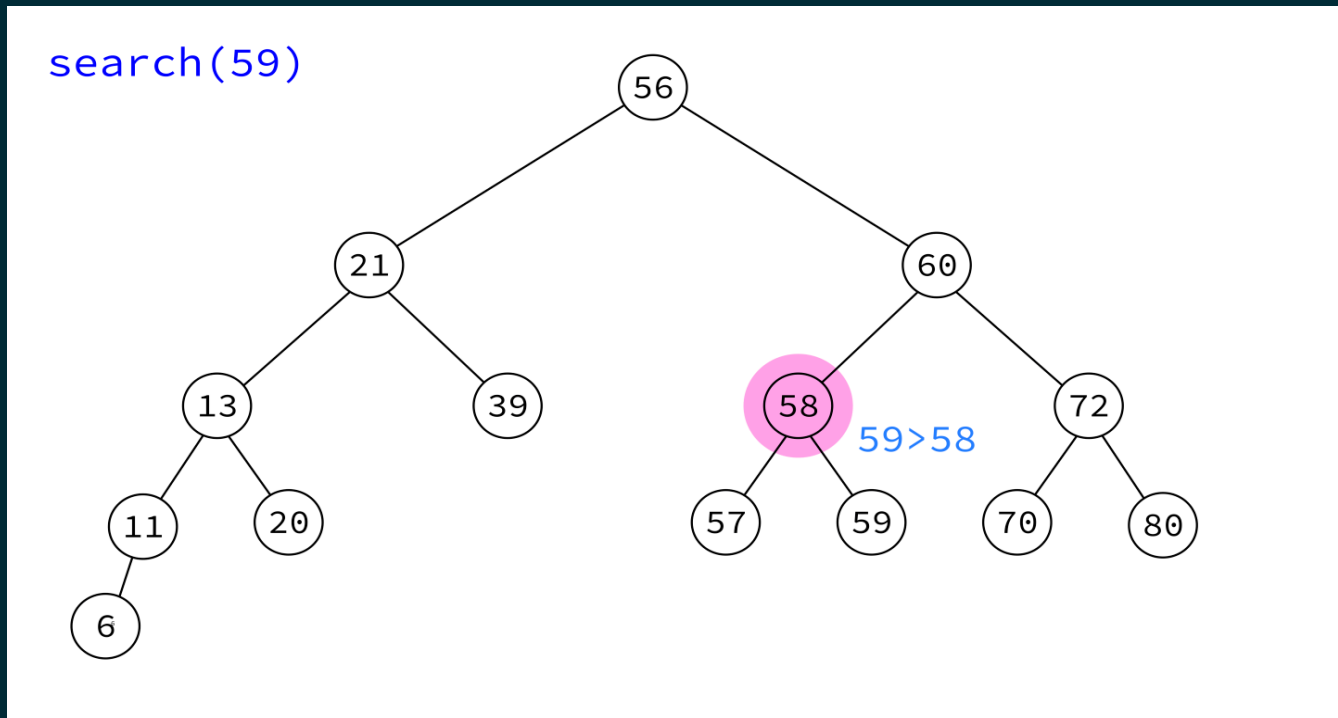
# SEARCH

Given  $x$ , find and return a node with key  $x$ . Return `None` if no such node exists.



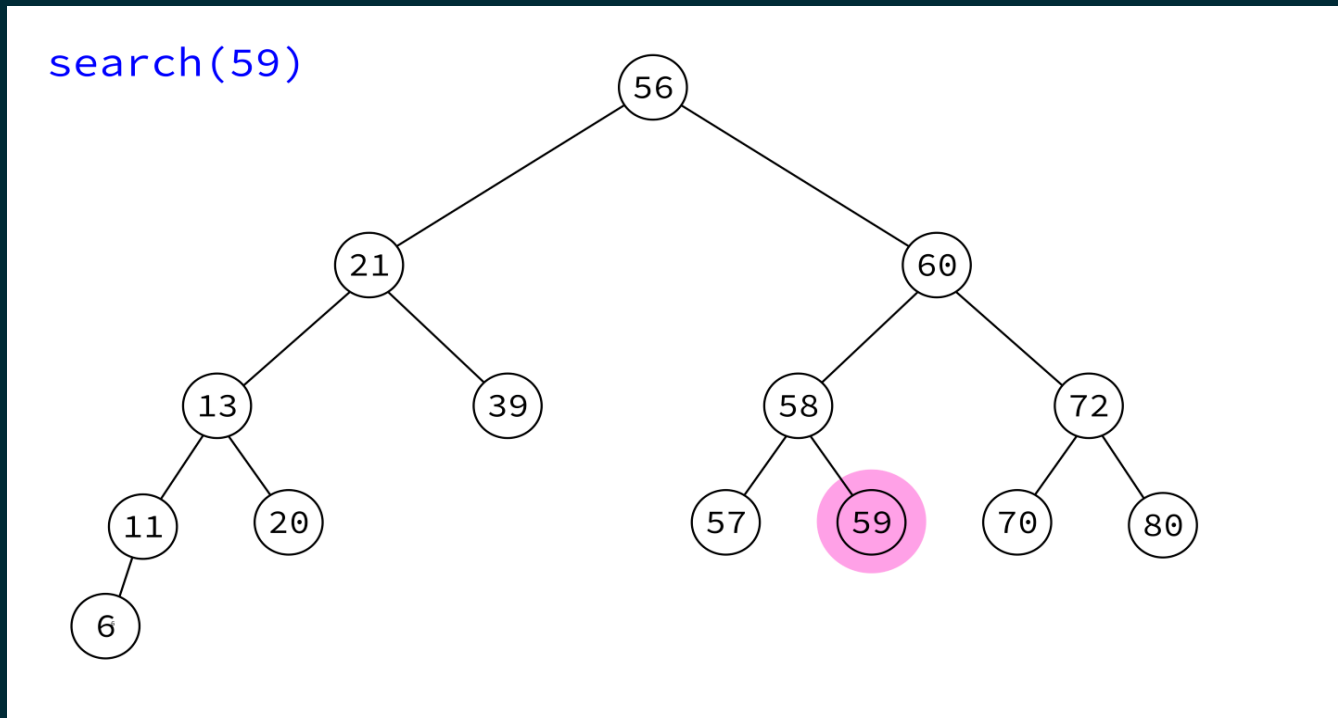
# SEARCH

Given  $x$ , find and return a node with key  $x$ . Return `None` if no such node exists.



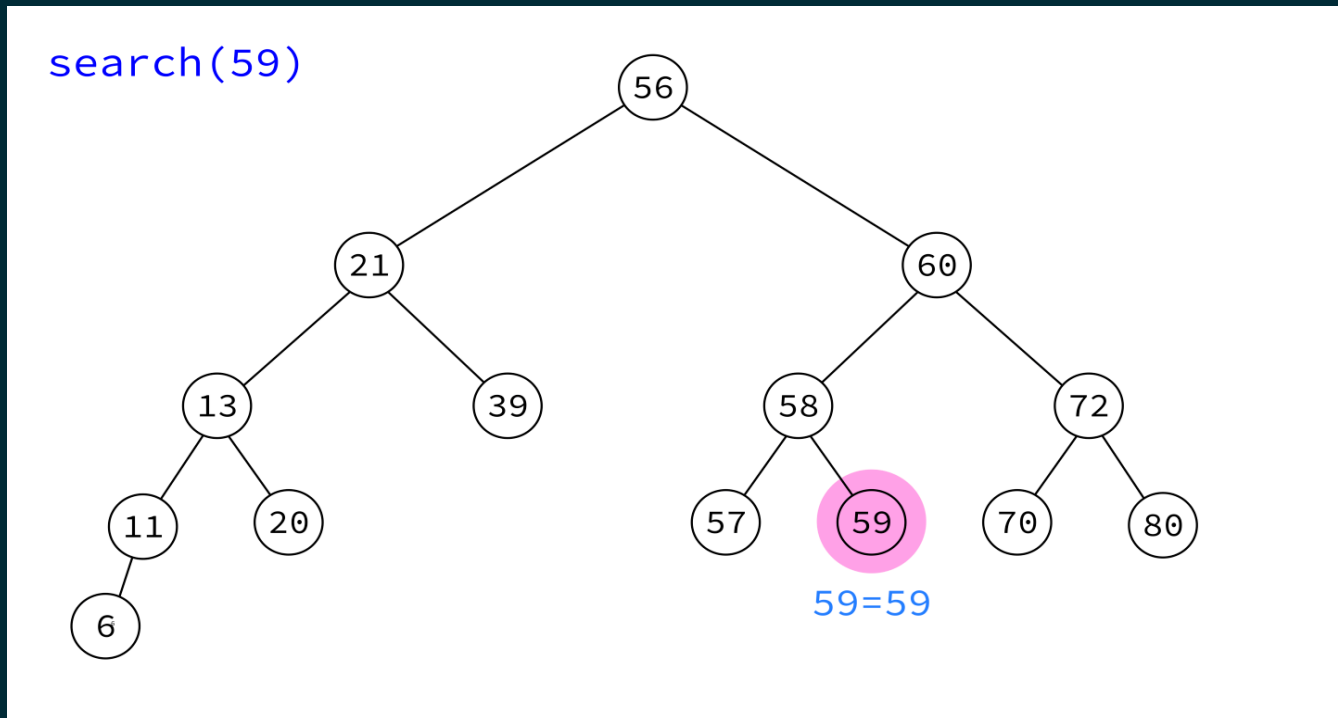
# SEARCH

Given  $x$ , find and return a node with key  $x$ . Return `None` if no such node exists.



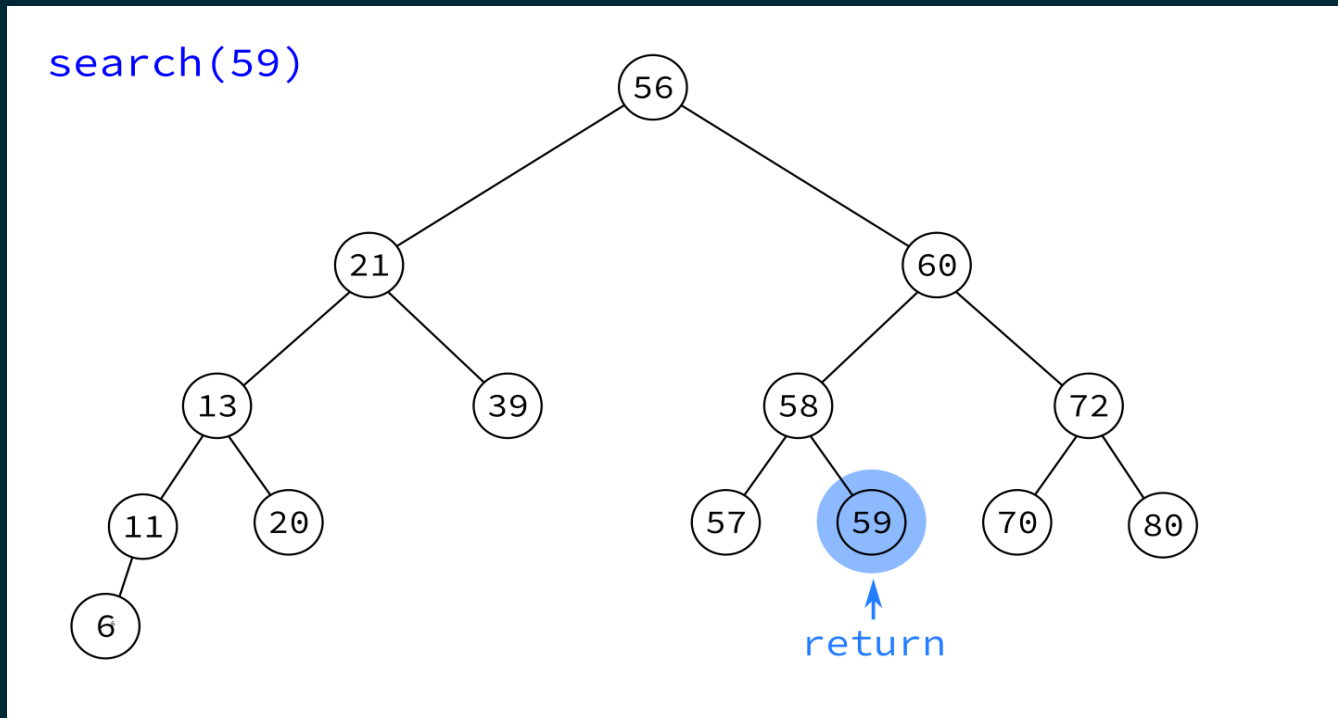
# SEARCH

Given  $x$ , find and return a node with key  $x$ . Return `None` if no such node exists.



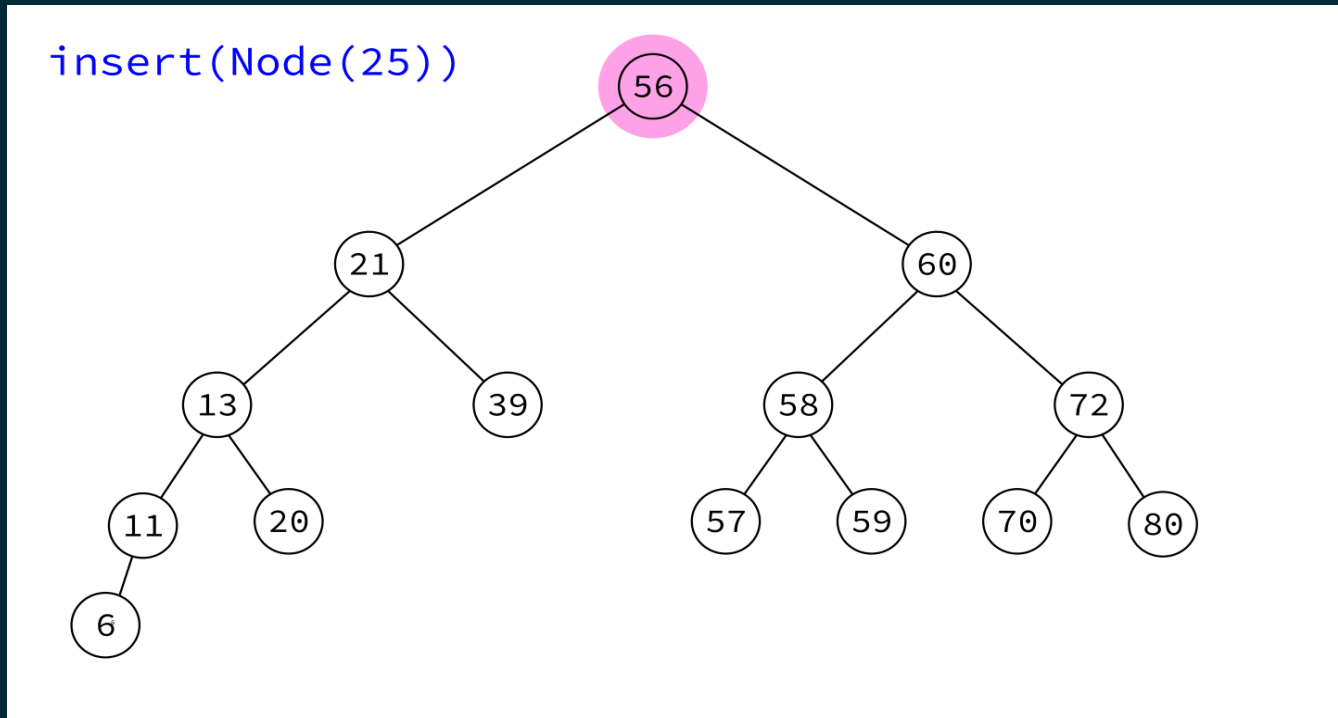
# SEARCH

Given  $x$ , find and return a node with key  $x$ . Return `None` if no such node exists.



# INSERT

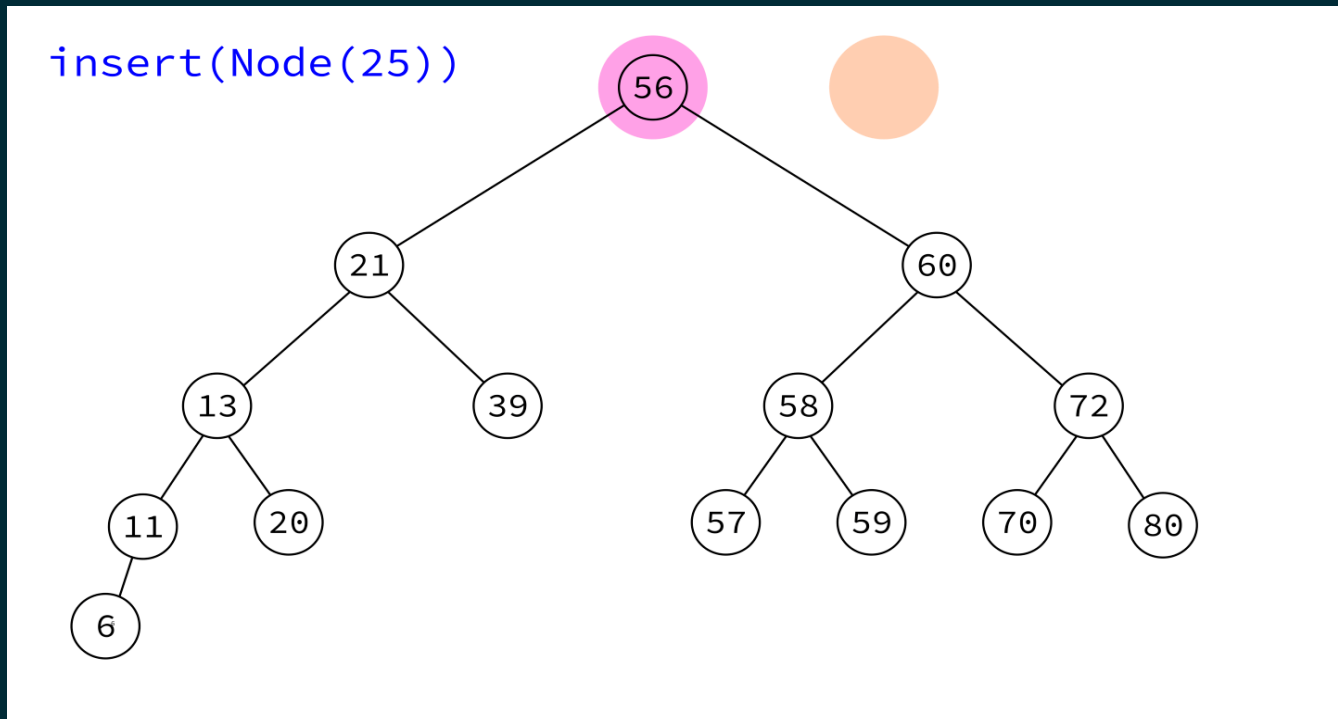
Take a `Node` object and add it to the tree in a place that maintains the BST property.



The node must have its `key` set, but not its parent and children. Insert will handle those.

# INSERT

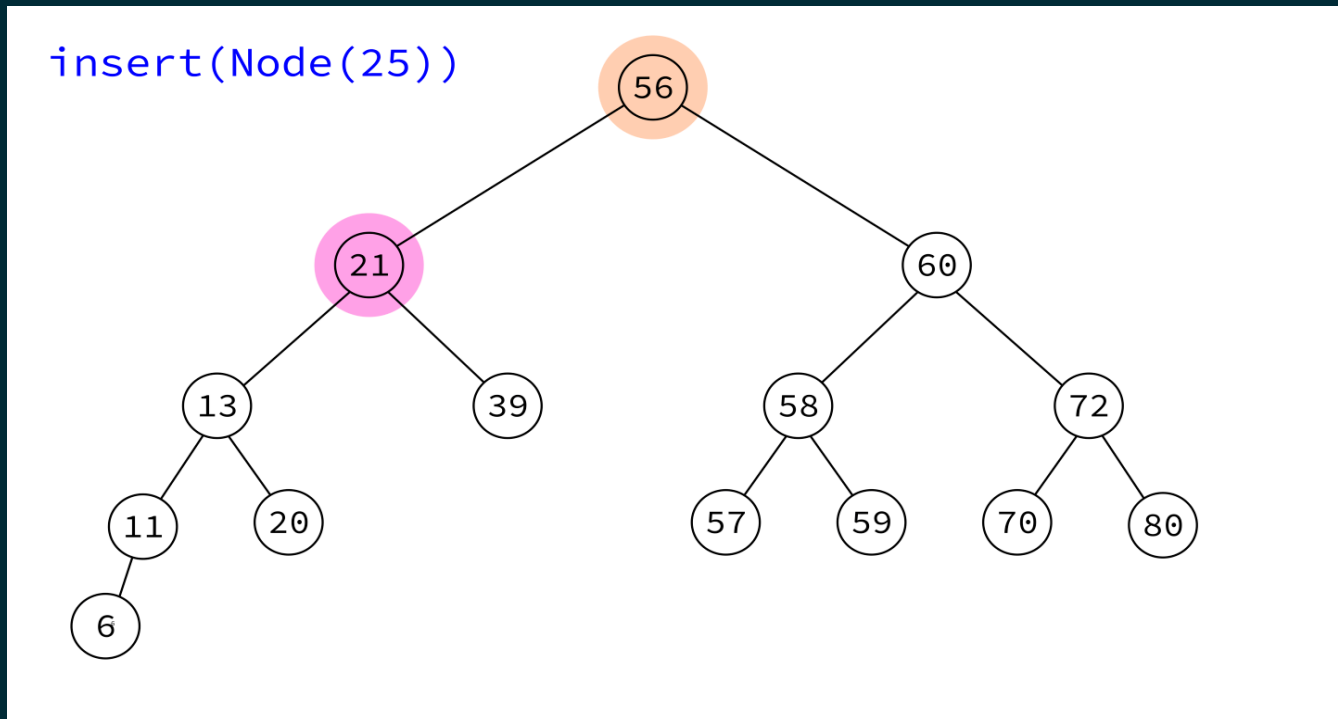
Take a `Node` object and add it to the tree in a place that maintains the BST property.



The node must have its `key` set, but not its parent and children. Insert will handle those.

# INSERT

Take a `Node` object and add it to the tree in a place that maintains the BST property.

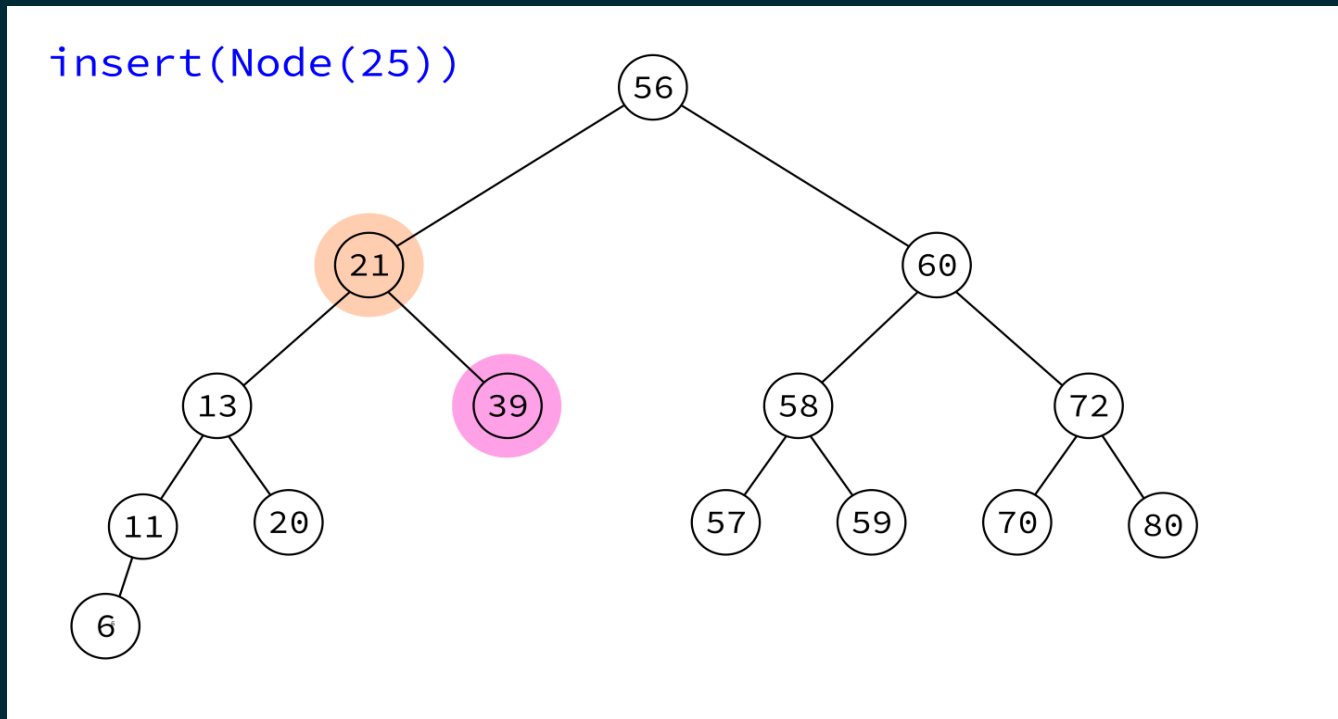


The node must have its `key` set, but not its parent and children. Insert will handle those.



# INSERT

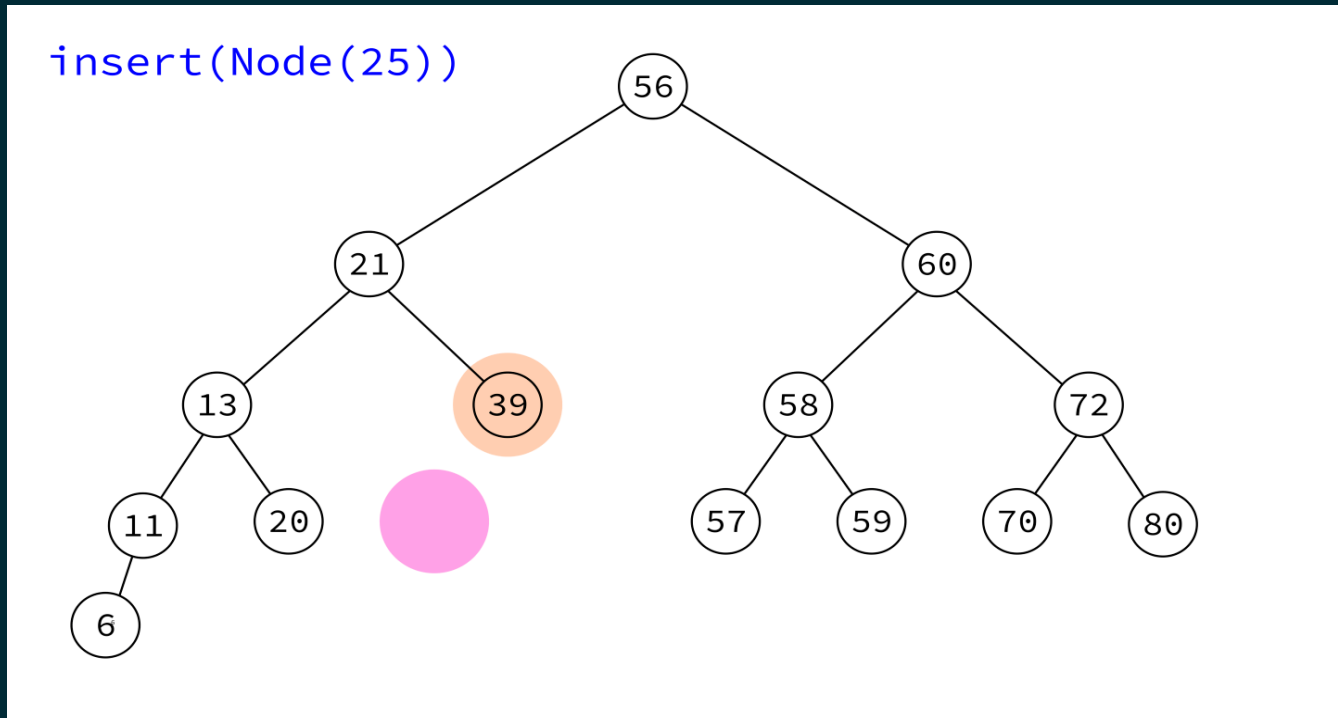
Take a `Node` object and add it to the tree in a place that maintains the BST property.



The node must have its `key` set, but not its parent and children. Insert will handle those.

# INSERT

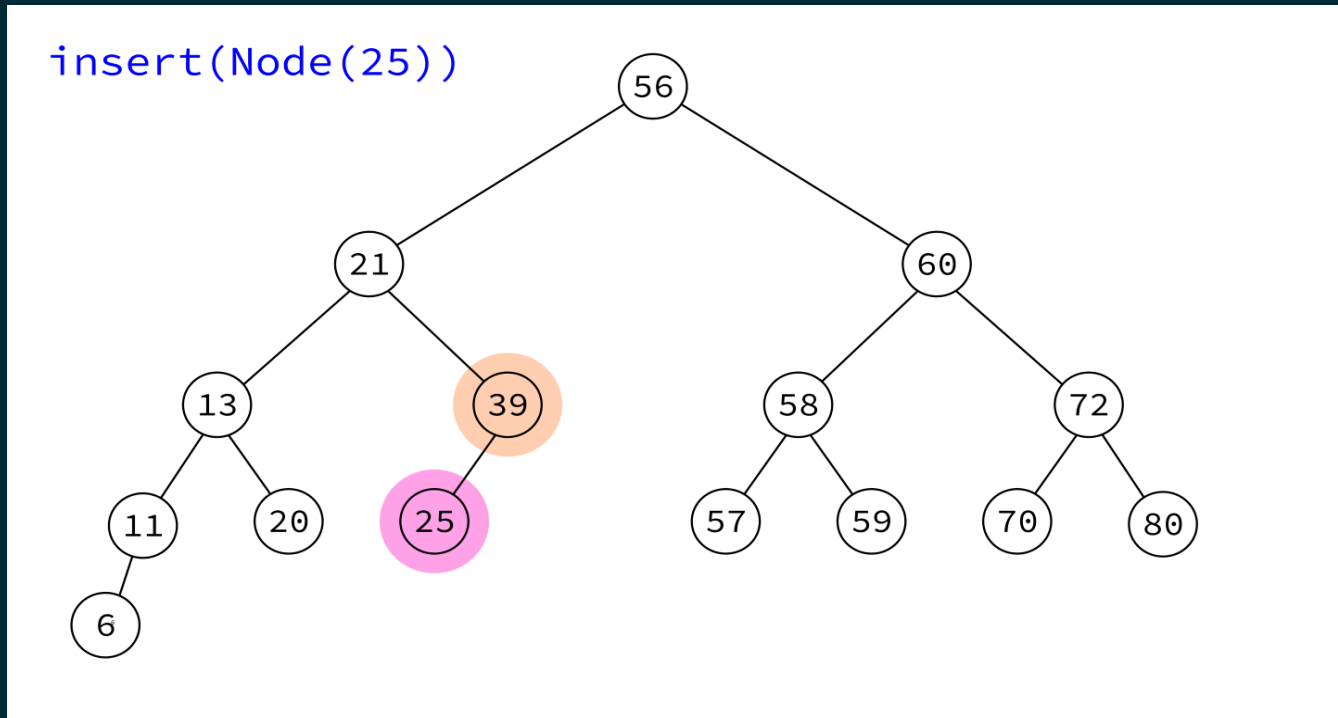
Take a `Node` object and add it to the tree in a place that maintains the BST property.



The node must have its `key` set, but not its parent and children. Insert will handle those.

# INSERT

Take a `Node` object and add it to the tree in a place that maintains the BST property.



The node must have its `key` set, but not its parent and children. Insert will handle those.

# INTEGERSET

Let's use this to build a class to store a collection of integers that supports fast insertion and membership testing.

# IMPLEMENTATION HIDING

To use `BST`, you need to know about and work with `Node` objects.

In contrast, `IntegerSet` has an interface based directly on the values to be stored. It hides the fact that its implementation uses a `BST`.

# REFERENCES

- In optional course texts:
  - *Problem Solving with Algorithms and Data Structures using Python* by Miller and Ranum, discusses binary trees in Chapter 7.
- Elsewhere:
  - Cormen, Leiserson, Rivest, and Stein discusses graph theory and trees in Appendices B.4 and B.5, and binary search trees in Chapter 12.

# REVISION HISTORY

- 2021-02-26 Fix incorrect integer comparison in one figure; add link to sample code
- 2021-02-25 Initial publication

