# CSC 172– Data Structures and Algorithms

Lecture #20

Spring 2018

Please put away all electronic devices

Fundamental data structure for

- Storing (key, value) pairs
- Allowing for efficient insertion, deletion, and search for values given keys

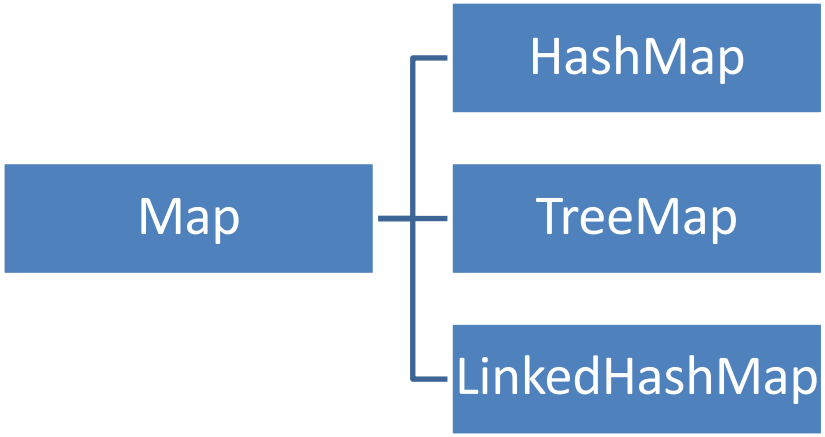
# BINARY SEARCH TREES

# Managing (Key, Value) Pairs

- (username, password)
- MapReduce framework
- Domain Name System
- Dictionary lookup
- Map (string →int)

- Binary Search Tree is a good data structure for maintaining (key, value) pairs
    - TreeMap in Java

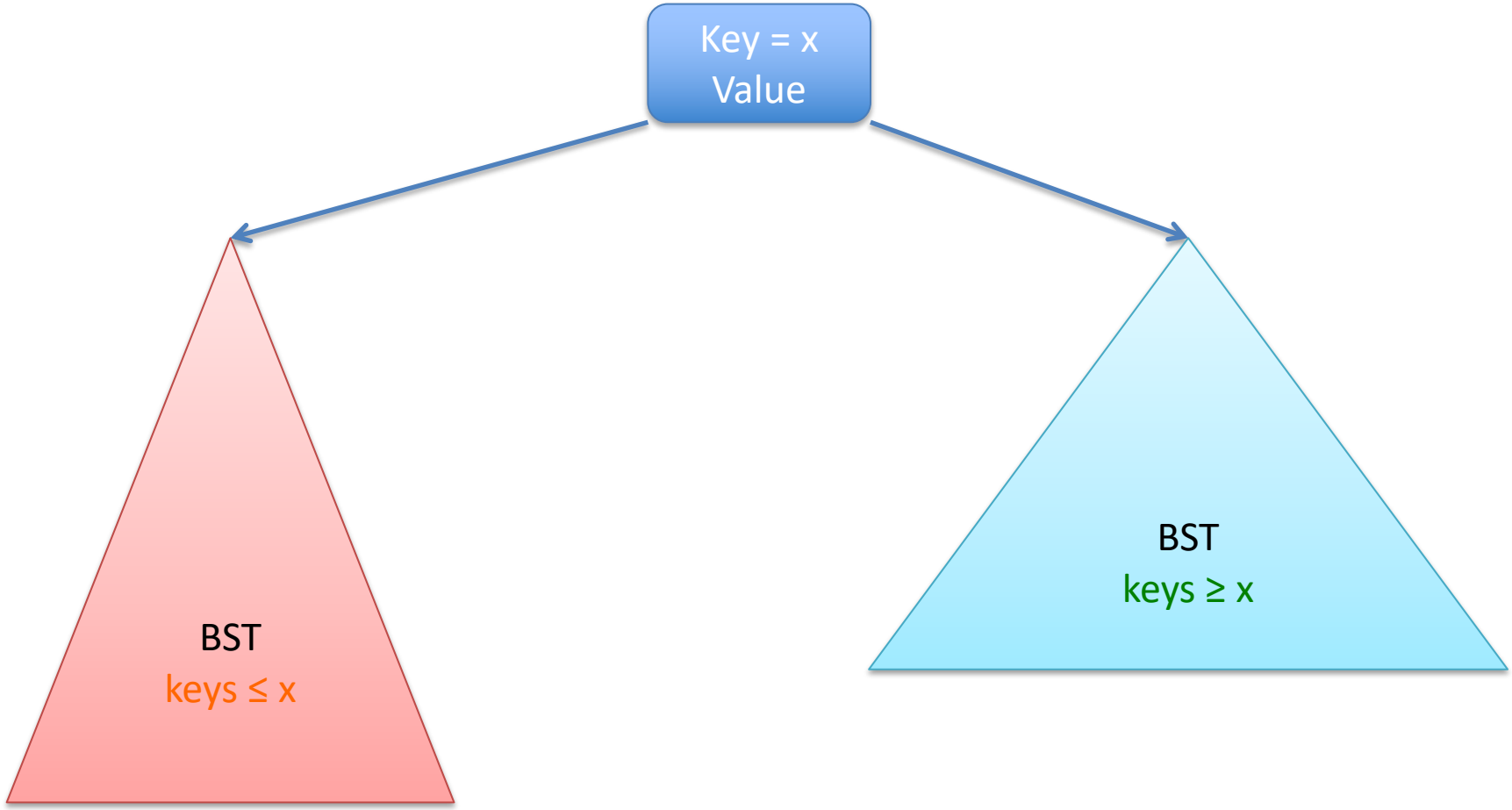
# A Little Digression



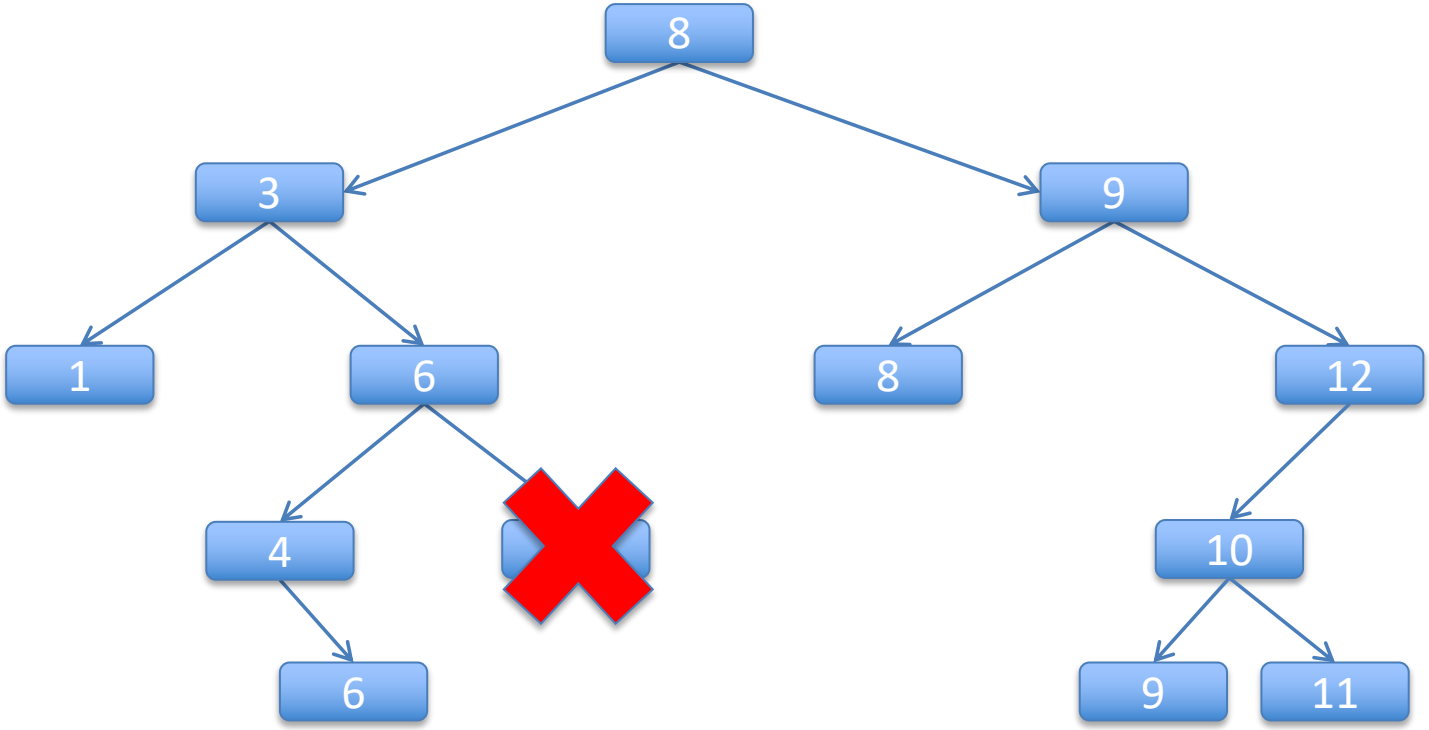

- A Map is an object that maps keys to values.
- A map cannot contain duplicate keys
- Each key can map to at most one value.

- HashMap uses Hashing (we will learn later)
- TreeMap uses balanced Binary Search Tree

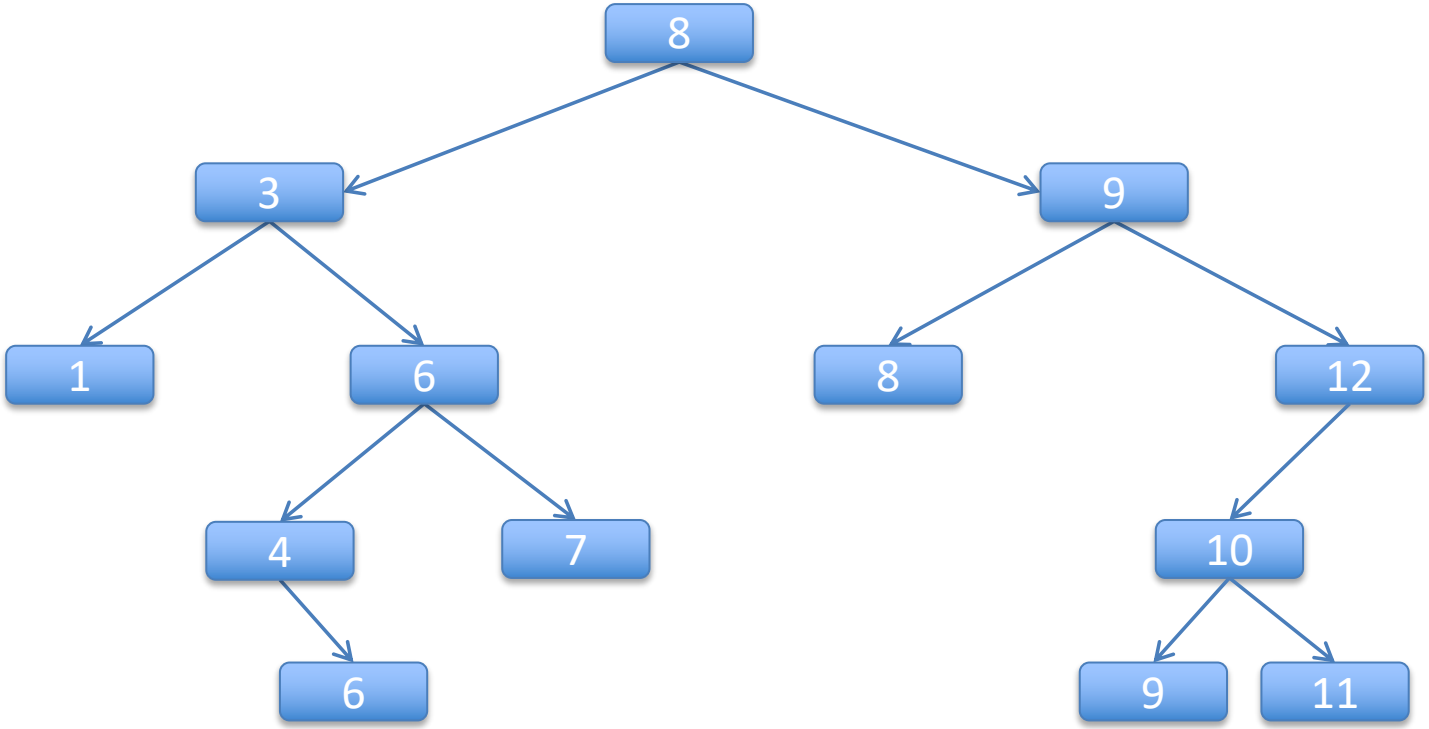
# Binary Search Tree & Its Main Property



# Example of BST



# Example of BST




`Inorder traversal` lists all keys in non-decreasing order!

# Basic Operations

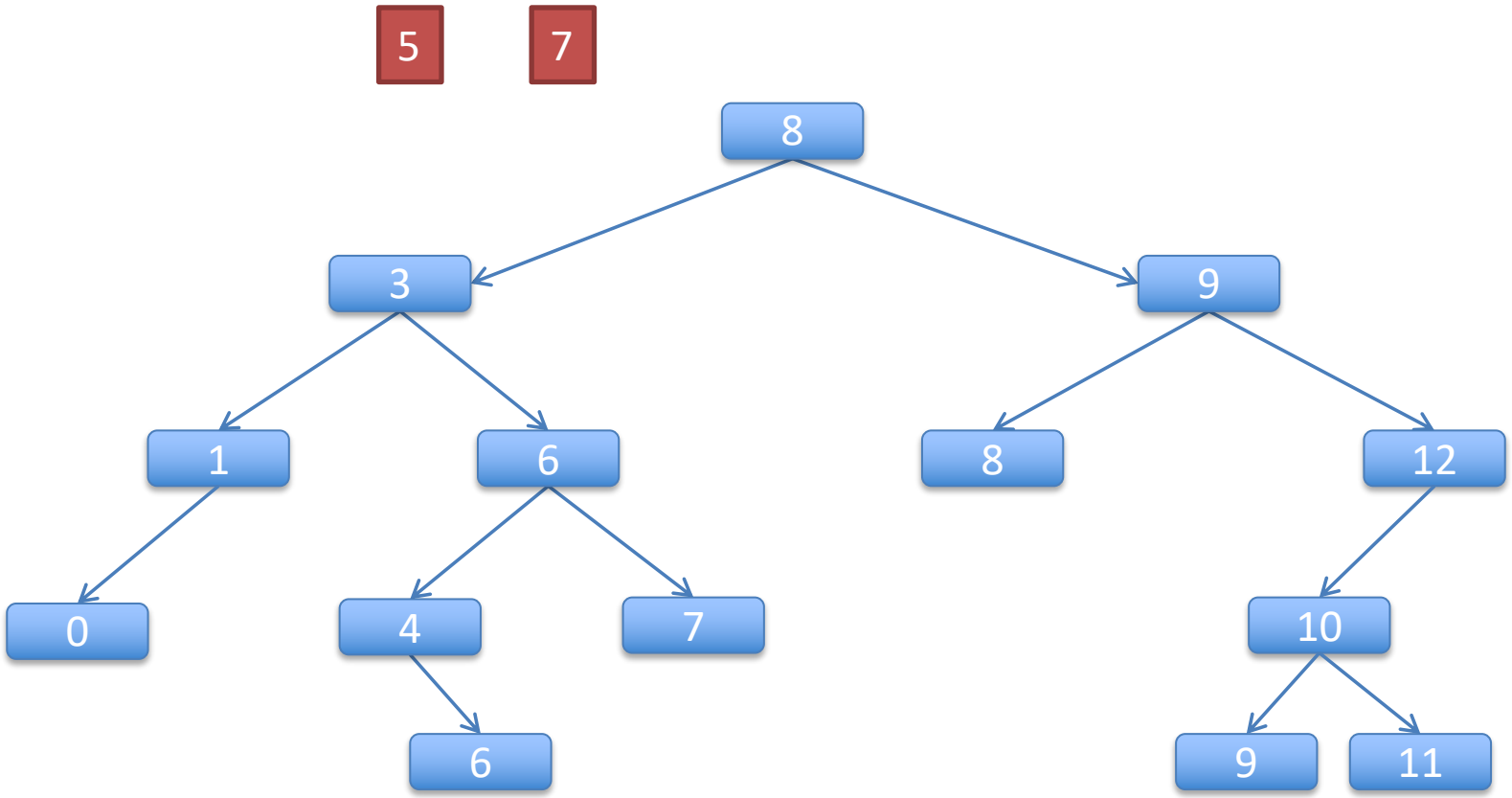
- `search`(tree, key)
- `minimum`(tree), `maximum`(tree)
- `successor`(tree, node)
  `predecessor`(tree, node)
- `insert`(tree, node) – node has (key, value)
  `delete`(tree, node) – node has (key, value)

# BSTNode in Java

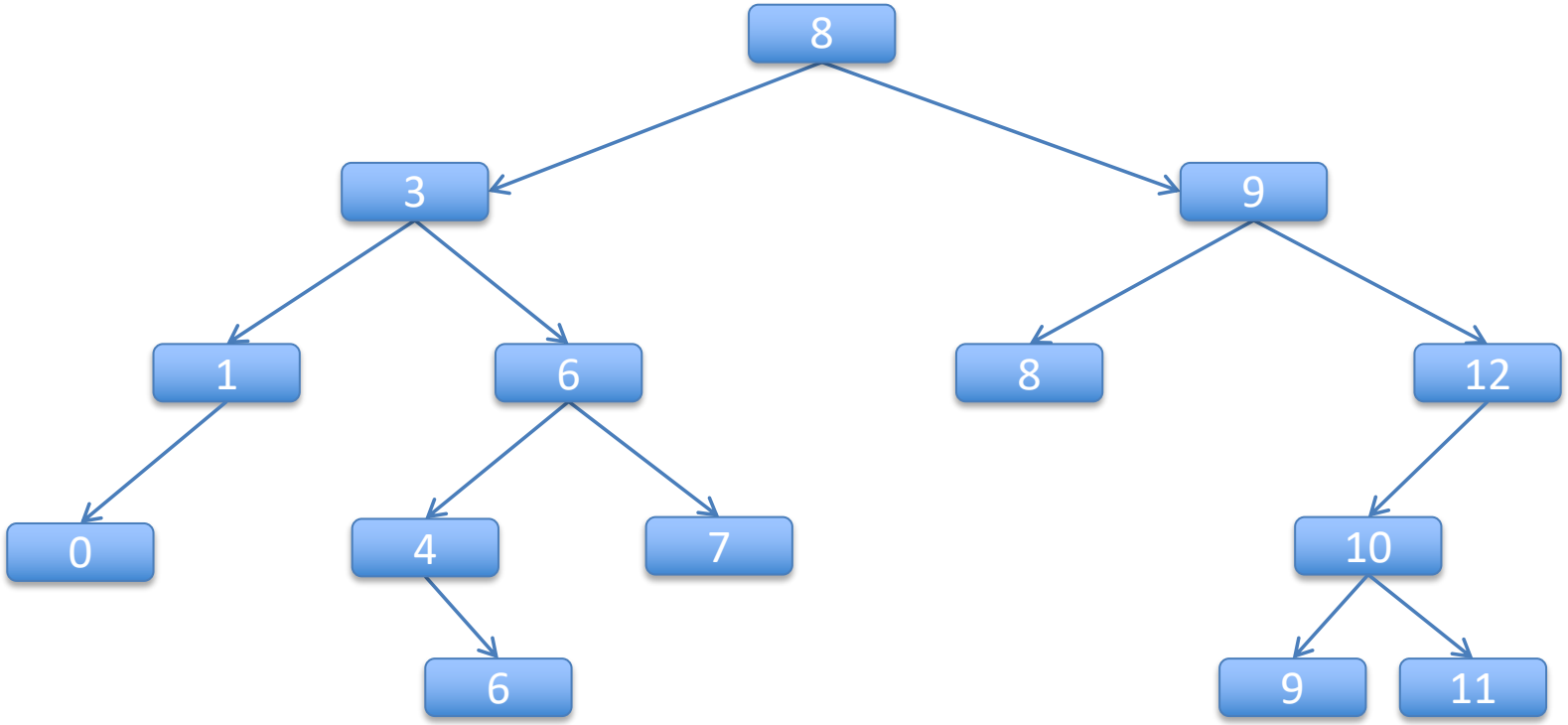
```
class BSTNode <K, V> {
    K       key;
    V     value;
    BSTNode<K,V> left;
    BSTNode<K,V> right;
    BSTNode<K,V> parent; //Note: we can include parent too!

};
```

# Search in a BST



# Minimum and Maximum



# Successor and Predecessor

- If we perform inorder traversal of a binary tree,
  - the neighbors of a given node are called
    - **Predecessor** (the node lies behind of given node)
    - **Successor** (the node lies ahead of given node).

- Assume: In-order traversal of BST Produces:
  - 1, 3, 4, 6, 7, 8, 9, 12
  - Then:
    - Predecessor of 8 is 7
    - Successor of 8 is 9

# Successor and Predecessor (cont.)

- Assume: In-order traversal of BST Produces:
  - 1, 3, 4, 6, 7, 8, 9, 12
  - Then:
    - Predecessor of 8 is 7
    - Successor of 7 is 8

# How to find Successor of a Node v

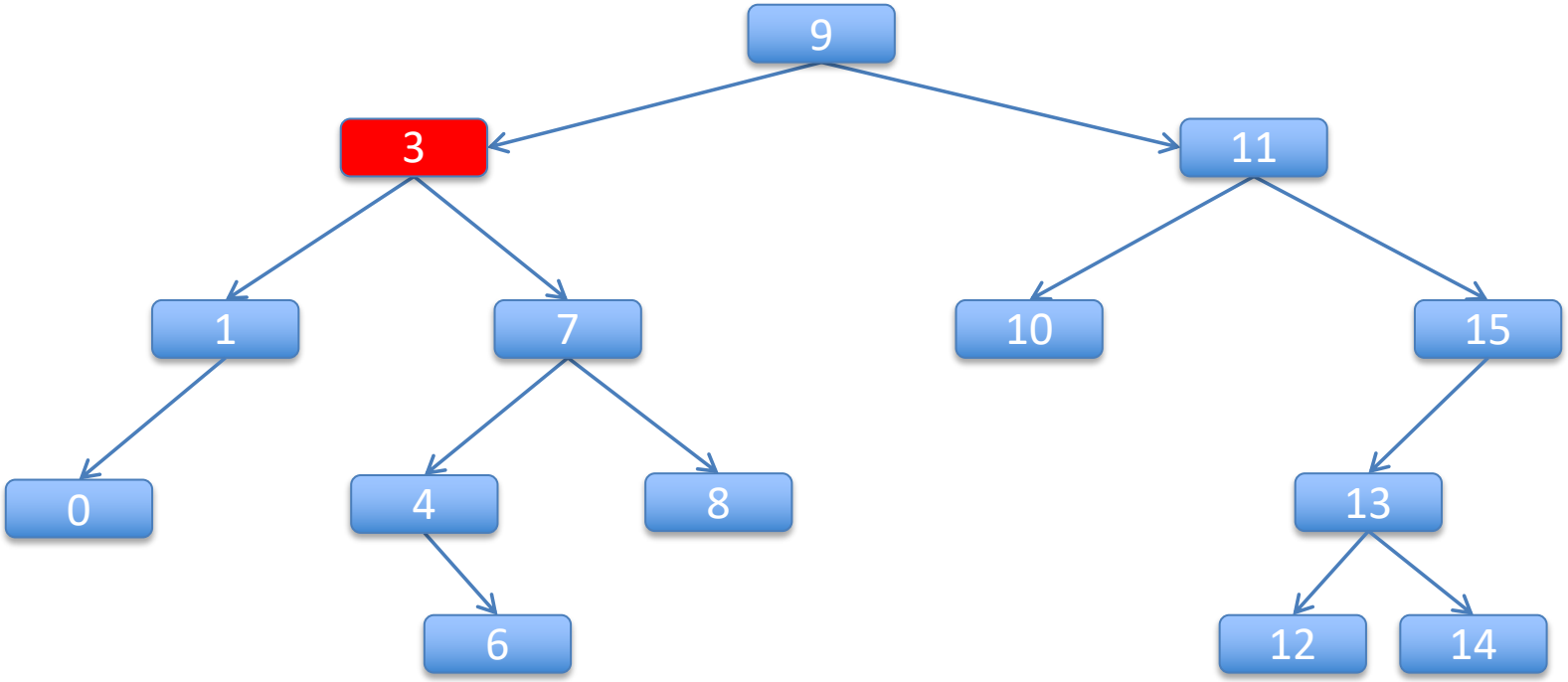
- If v has a right sub-tree, we will find the successor there
  - Min of Right Subtree

- If v does not have a right sub-tree:
  - We need to find the node x whose predecessor is v

How to find predecessor of node x?:
Maximum node of x's left sub-tree
Go left, and then go all the way
down towards right.

Do just the reverse:
Go up (y → z) … until
y is the left child of z.

# Successor



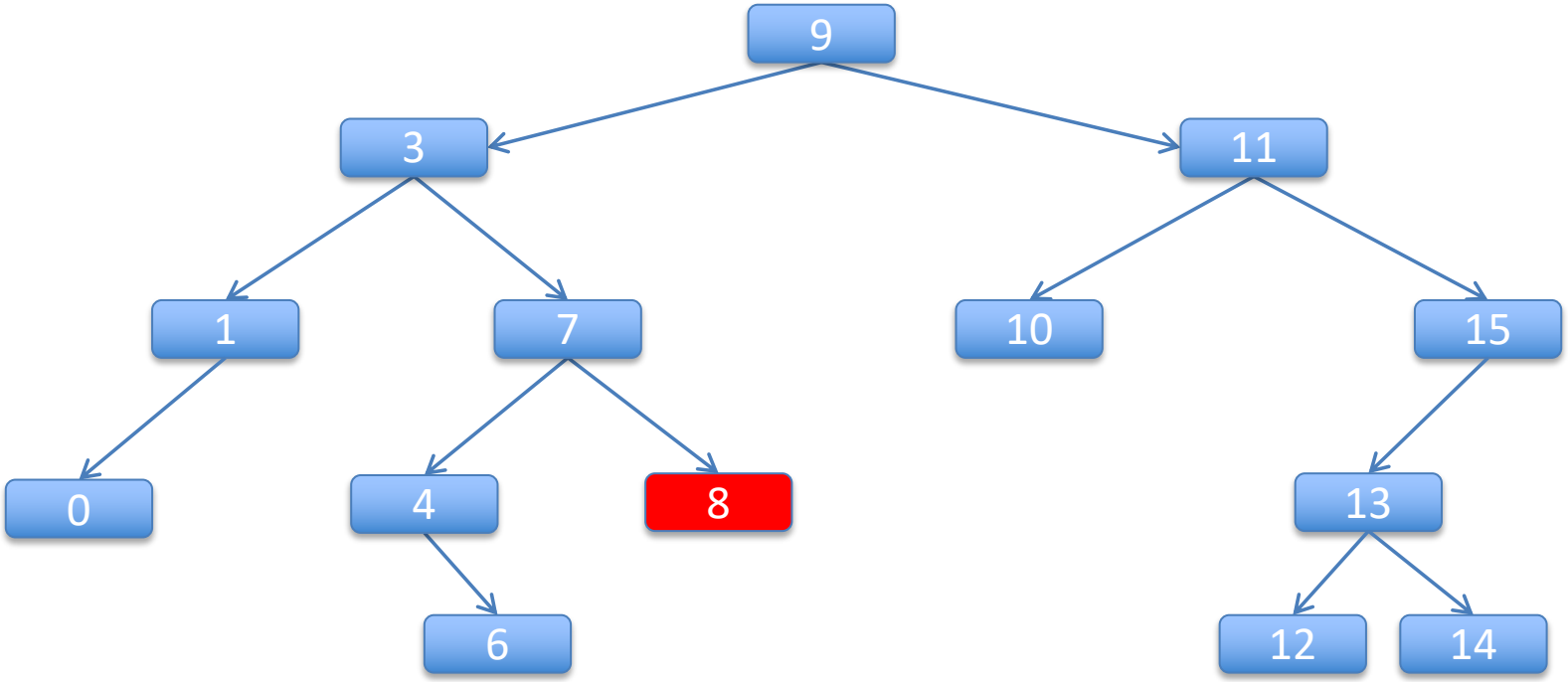

If v has a right branch:

`successor`(v) = `minimum`(right-branch)

Else,

`successor`(v) = the first ancestor u with another ancestor as a left child

# Successor



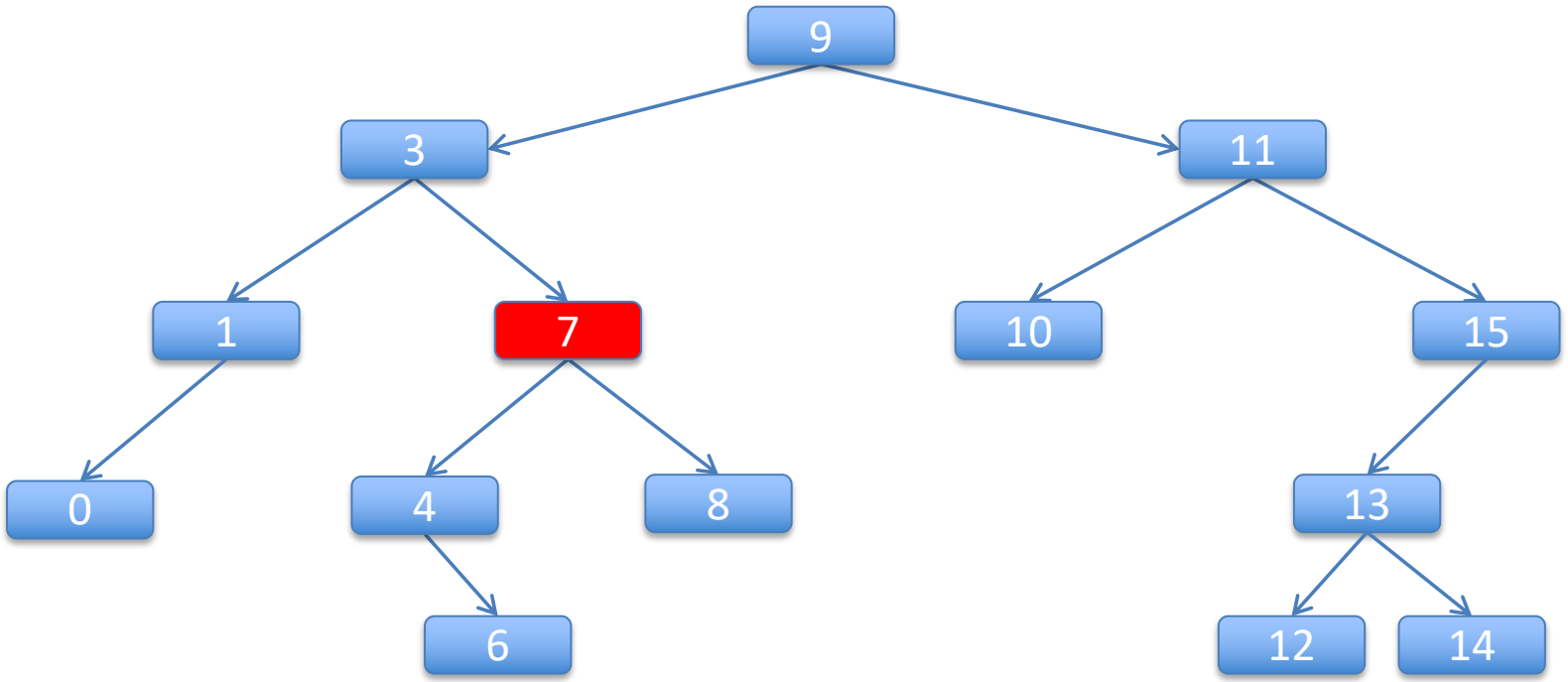

If v has a right branch:

`successor`(v) = `minimum`(right-branch)

Else,

`successor`(v) = the first ancestor u with another ancestor as a left child

# Predecessor



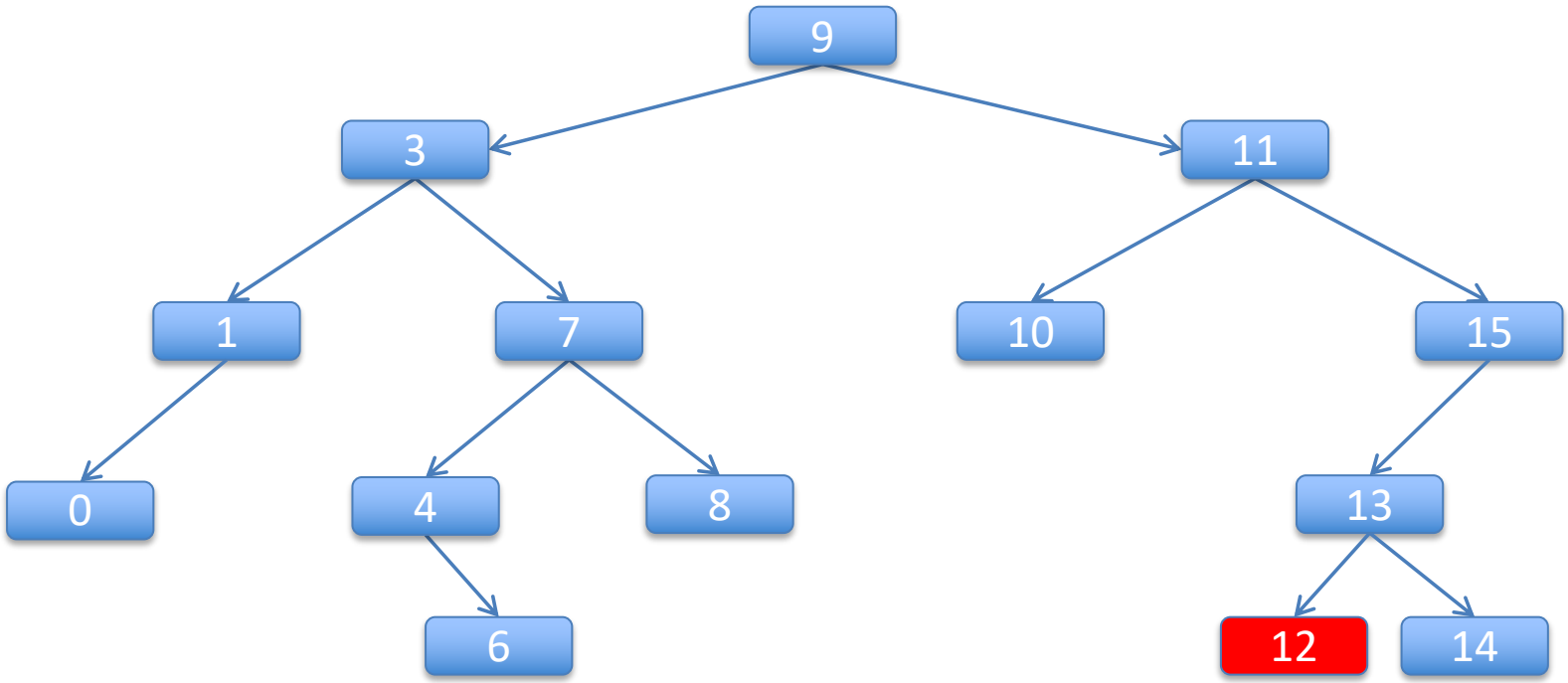

If v has a left branch:

`predecessor`(v) = `maximum`(left-branch)

Else,

`predecessor`(v) = the first ancestor u with another ancestor as a right child

# Predecessor



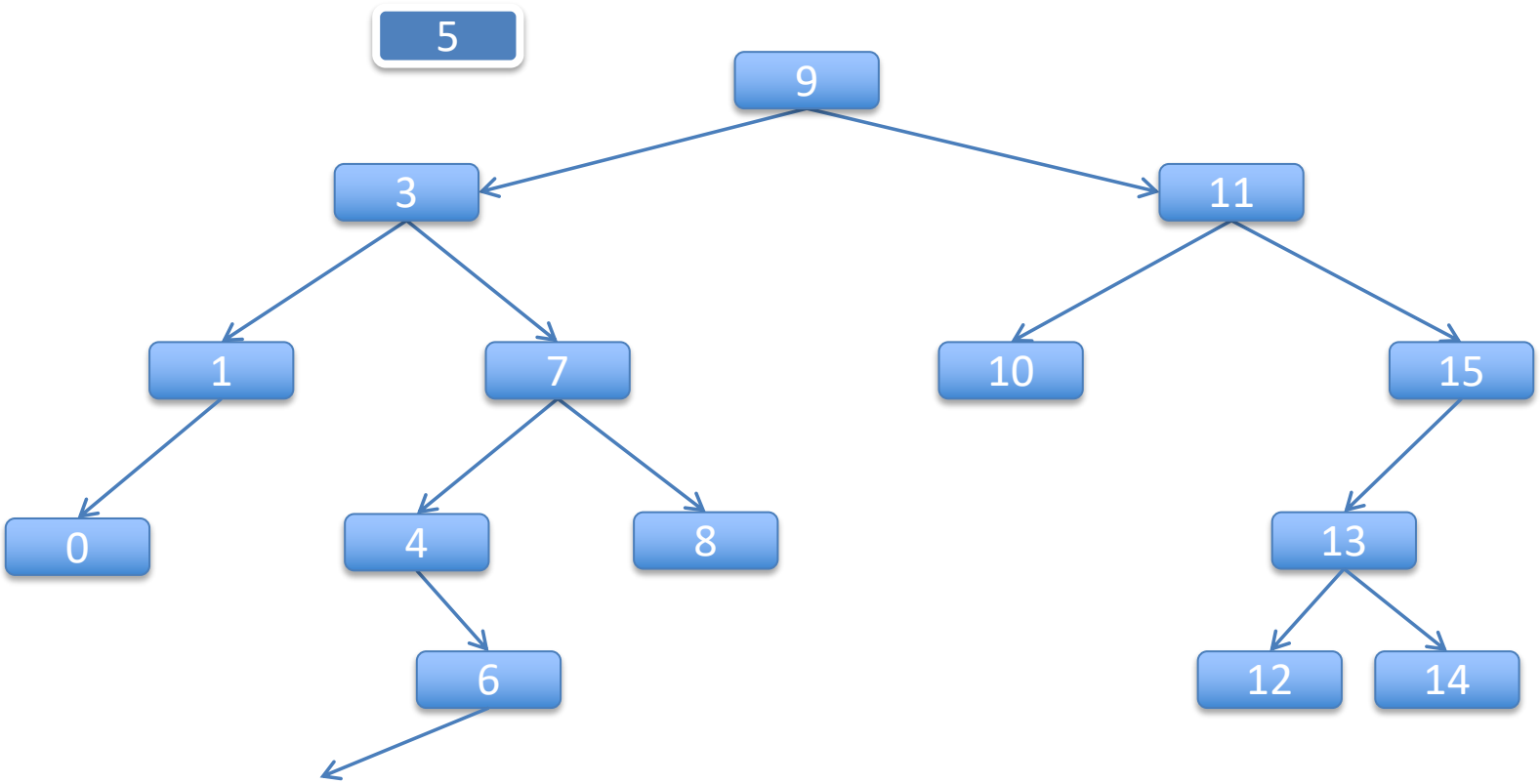

If v has a left branch:

`predecessor`(v) = `maximum`(left-branch)

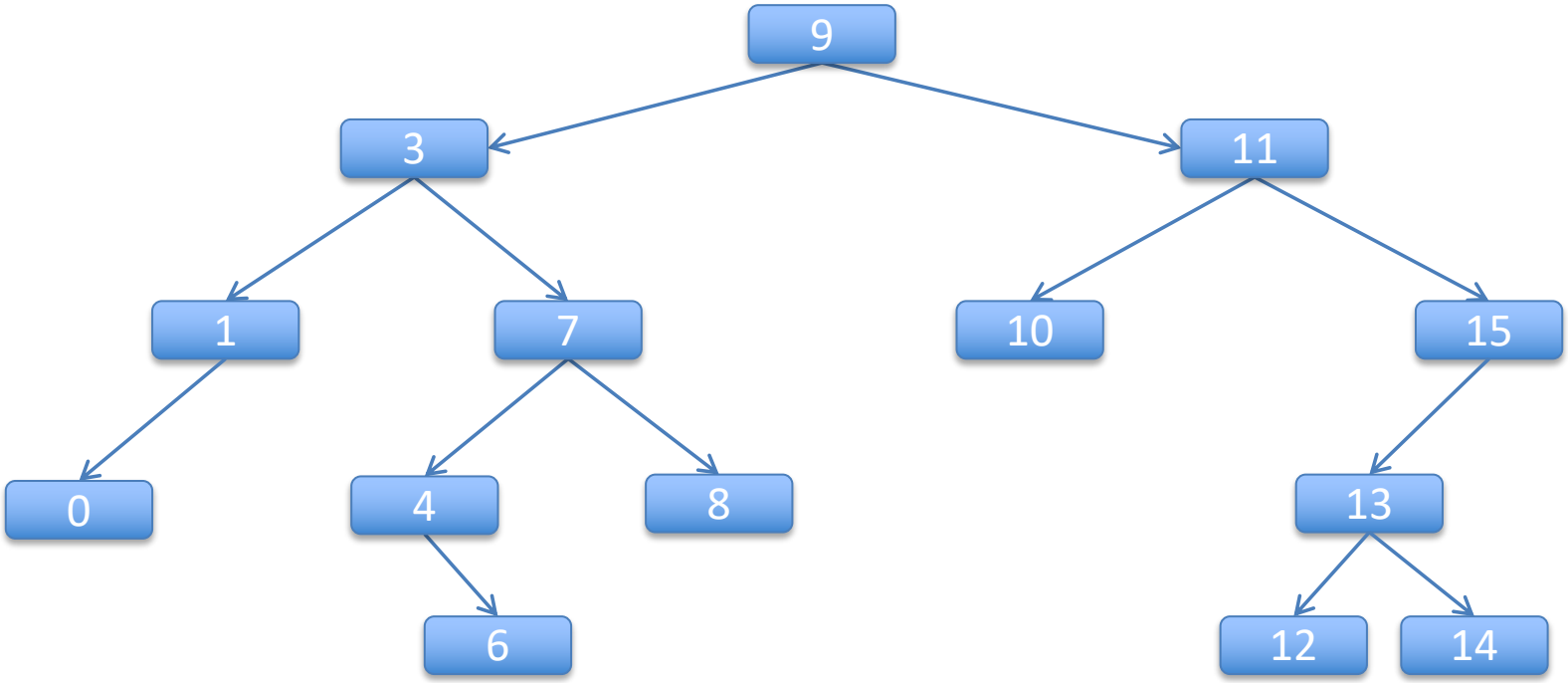
Else,

`predecessor`(v) = the first ancestor u with another ancestor as a right child

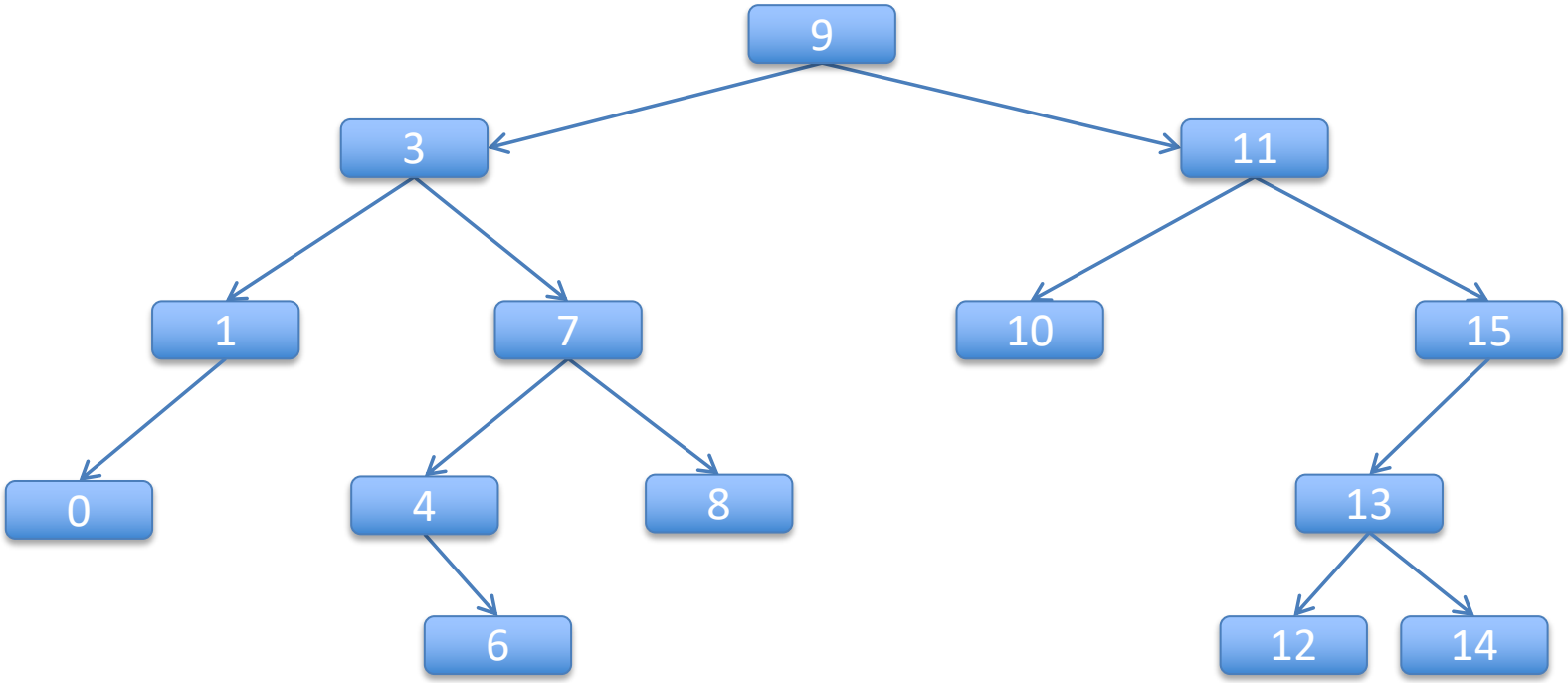
# Insert



# Delete – Node has ≤ 1 Child



# Delete – Node Has 2 Children



# Run Times of Basic Operations

- `search`(tree, key)
- `minimum`(tree)
  `maximum`(tree)
- `successor`(tree, node)
  `predecessor`(tree, node)
- `insert`(tree, node) – node has (key, value)
  `delete`(tree, node) – node has (key, value)

- All run in time `O(h)`
  - `h` is the height of the tree

# Random BST

- Consider storing a dictionary using a BST
- Randomize the word order
- Insert (word, meaning) pairs into the BST
- It can be shown that the expected height of a random BST is $O(\log n)$

# Summary

- A good data structure for storing (key, value ) pair
- If the tree is well-balanced, we can perform all the major operations in O(log *n*) time
- Our next topic would be: How to create a balanced binary search tree?

# Acknowledgement


- Douglas Wilhelm Harder.
  - Thanks for making an excellent set of slides for ECE 250 *Algorithms and Data Structures* course

- Prof. Hung Q. Ngo:
  - Thanks for those beautiful slides created for CSC 250 (Data Structures) course at UB.

- Many of these slides are taken from these two sources.