APA 254
Data Structures

Lecture 10.1
(Binary Search Trees)

Dept. of Information Systems
Hanyang University

Search Trees

- Search trees are ideal for implementing dictionaries
  - Similar or better performance than skip lists and hash tables
  - Particularly ideal for accessing data sequentially or by rank
- In this chapter, we will learn
  - **Binary search trees**
  - **Indexed binary search trees**

Binary Search Tree

Definition

- A binary tree that may be empty. A nonempty binary search tree satisfies the following properties:

1. Each node has a key (or value), and no two nodes have the same key (i.e., all keys are distinct).
2. For every node x, all keys in the left subtree of x are smaller than that in x.
3. For every node x, all keys in the right subtree of x are larger than that in x.
4. The left and right subtrees of the root are also binary search trees

Examples of Binary Trees

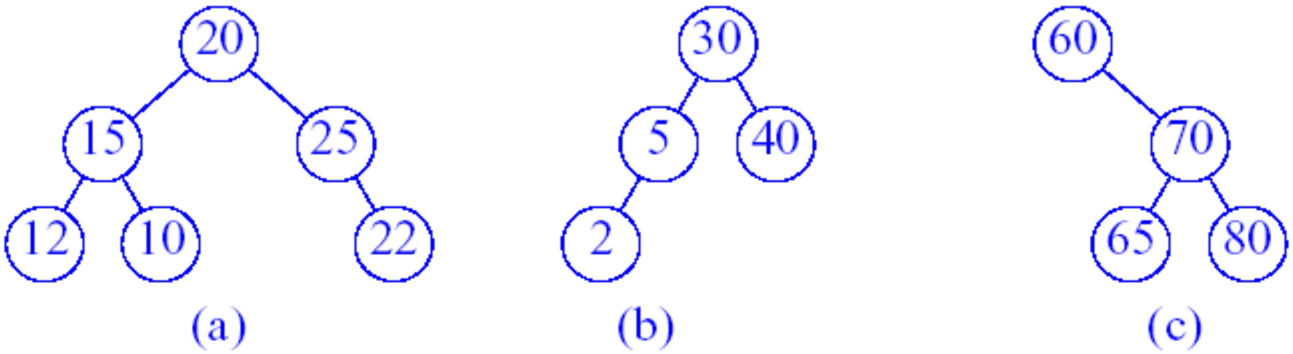


Figure 14.1 Binary Trees

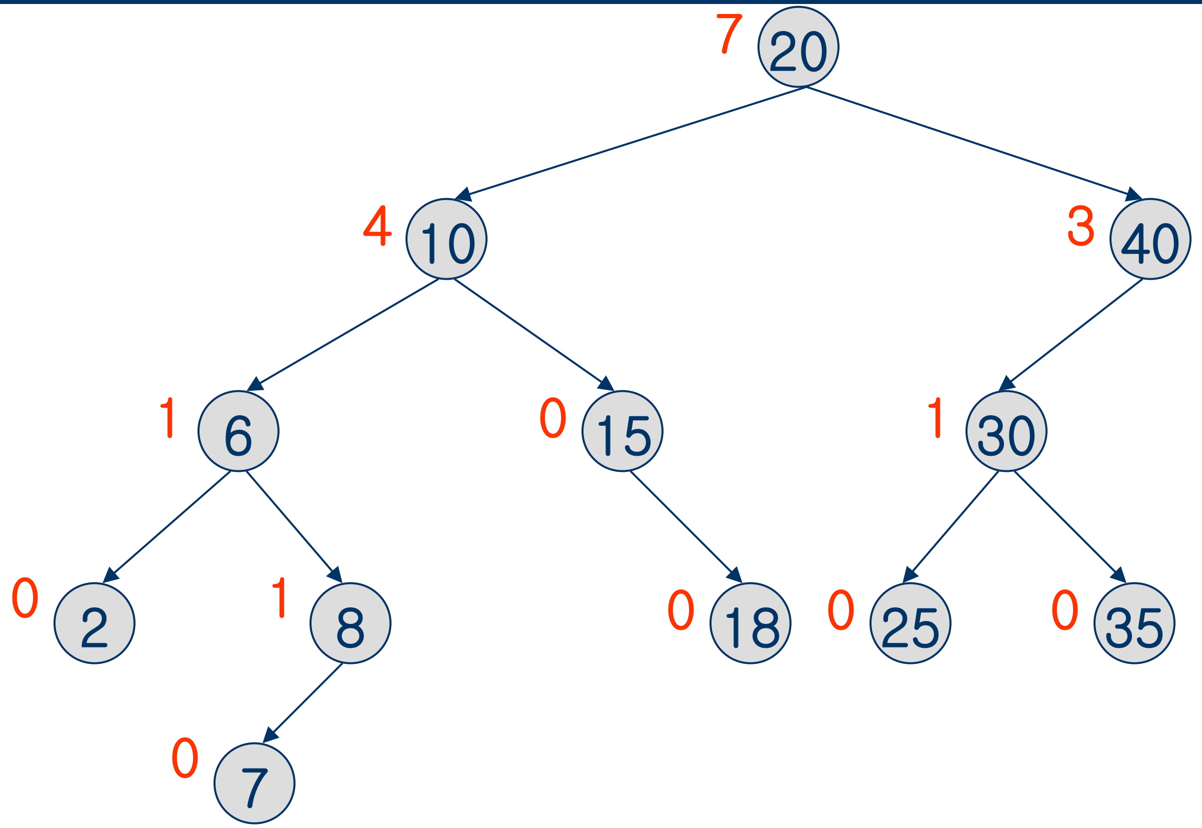
- Which of the above trees are binary search trees?
  → (b) and (c)
- Why isn't (a) a binary search tree?
  → It violates the property #3

Indexed Binary Search Trees

Definition

- Binary search tree.
- Each node has an additional field 'LeftSize'.
- LeftSize
  - the number of elements in its left subtree
  - the rank of an element with respect to the elements in its subtree (e.g., the fourth element in sorted order)

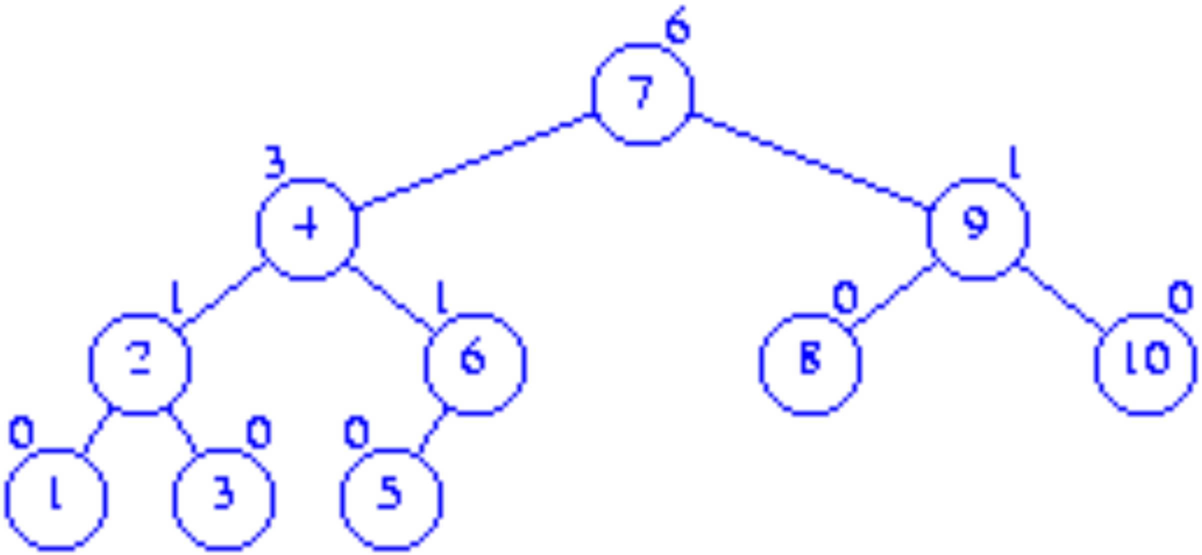
Indexed Binary Search Tree Example



- What is the Leftsize for each node?
- LeftSize values are in red.

Exercise

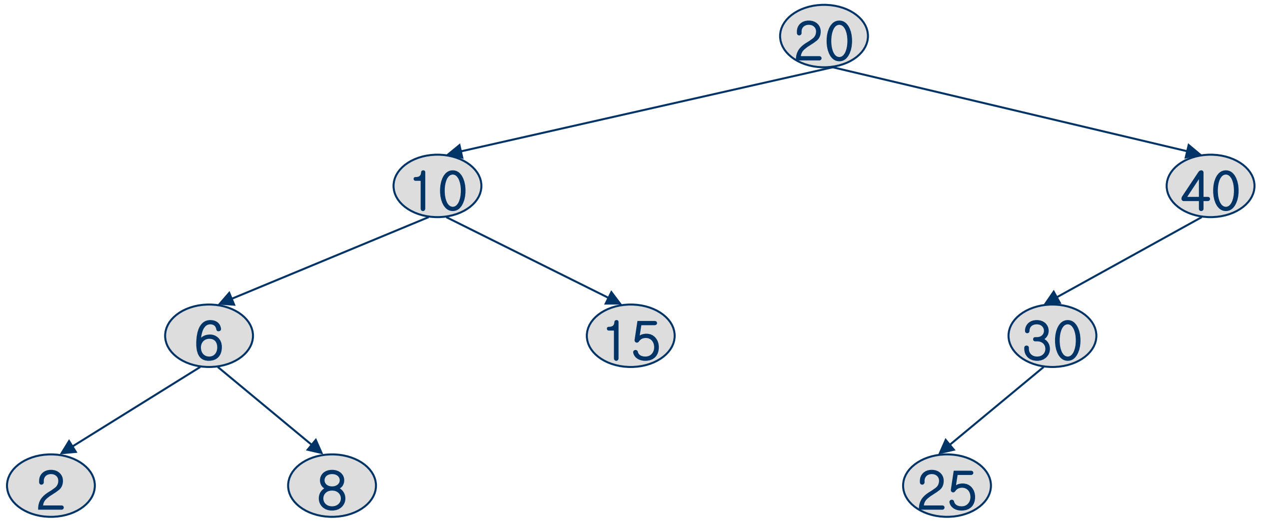
- Do Exercise 14.1



The Class binarySearchTree

- Since the number of elements in a binary search tree as well as its shape changes as operations are performed, a binary search tree is usually represented using the linked representation of Section 11.4.2
- We can define binarySearchTree as a derived class of linkedBinaryTree (Section 11.8)

The Operation Ascend()



- How can we output all elements in ascending order of keys?
  - Do an inorder traversal (left, root, right).
- What would be the output?
  - 2, 6, 8, 10, 15, 20, 25, 30, 40

The Operation Search(key, e)

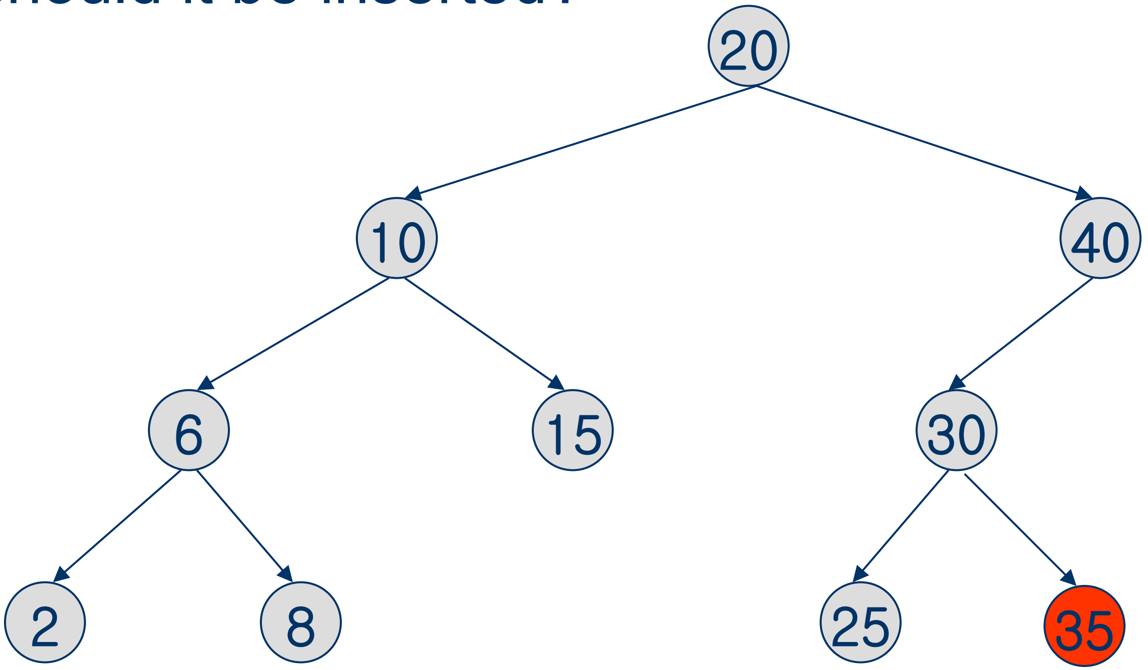
- Search begins at the root
- If the root is NULL, the search tree is empty and the search fails.
- If key is less than the root, then left subtree is searched
- If key is greater than the root, then right subtree is searched
- If key equals the root, then the search terminates successfully
- The time complexity for search is O(height)
- See Program 14.4 for the search operation code

The Operation Insert(key, e)

- To insert a new element e into a binary search tree, we must first verify that its key does not already exist by performing a search in the tree
- If the search is successful, we do not insert
- If the search is unsuccessful, then the element is inserted at the point the search terminated
  - Why insert it at that point?
- The time complexity for insert is O(height)
- See Figure 14.3 for examples
- See Program 14.5 for the insert operation code

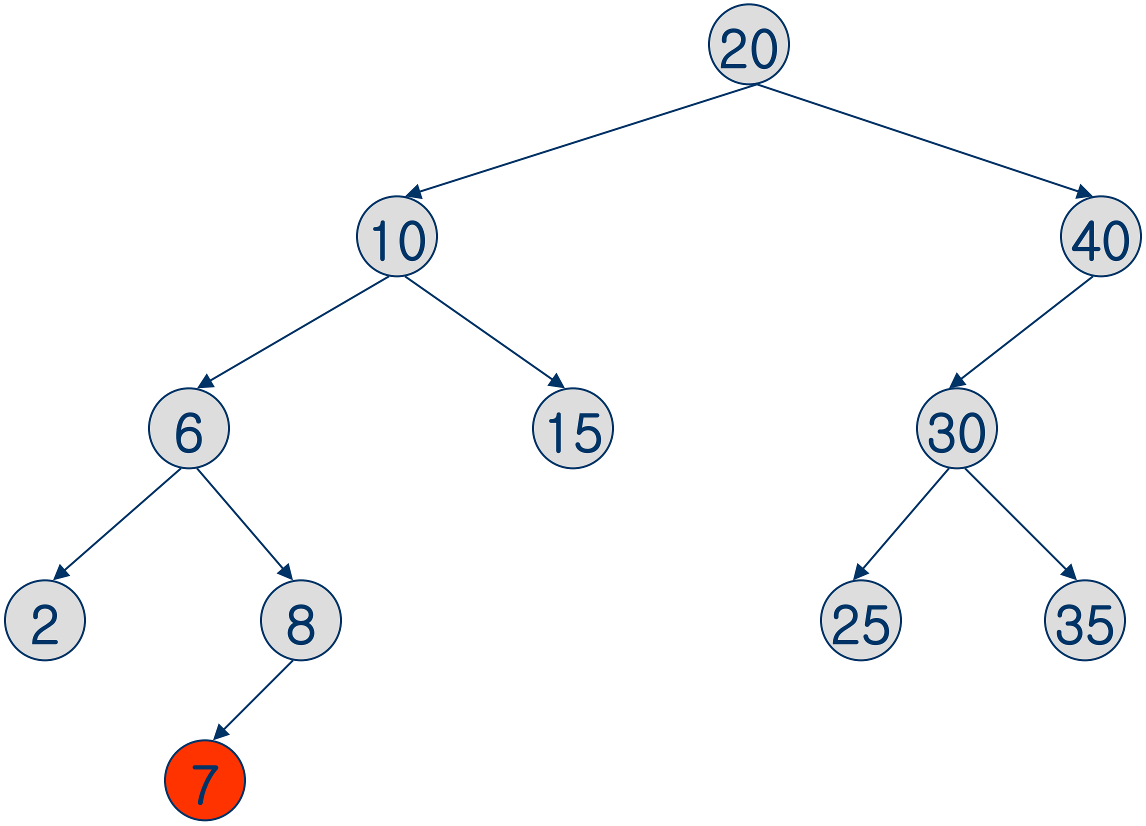
Insert Example

We wish to insert an element with the key 35.
Where should it be inserted?



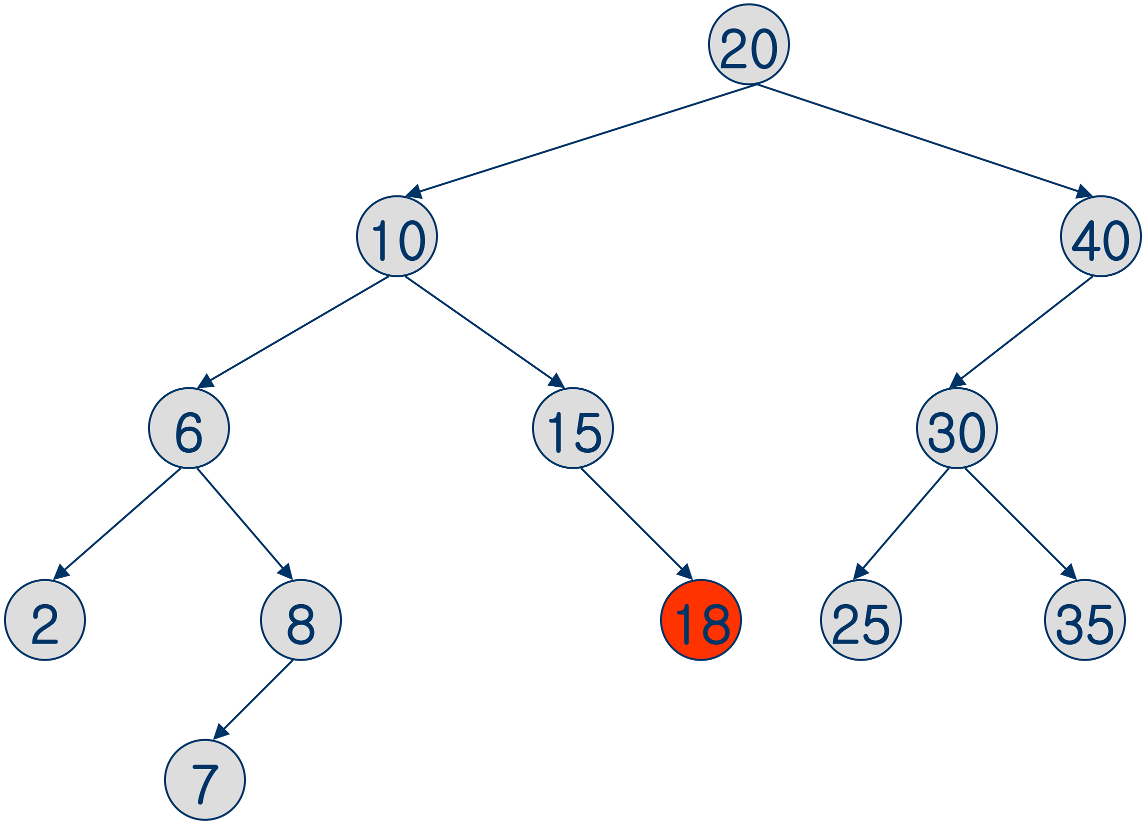
Insert Example

Insert an element with the key 7.



Insert Example

Insert an element with the key 18.

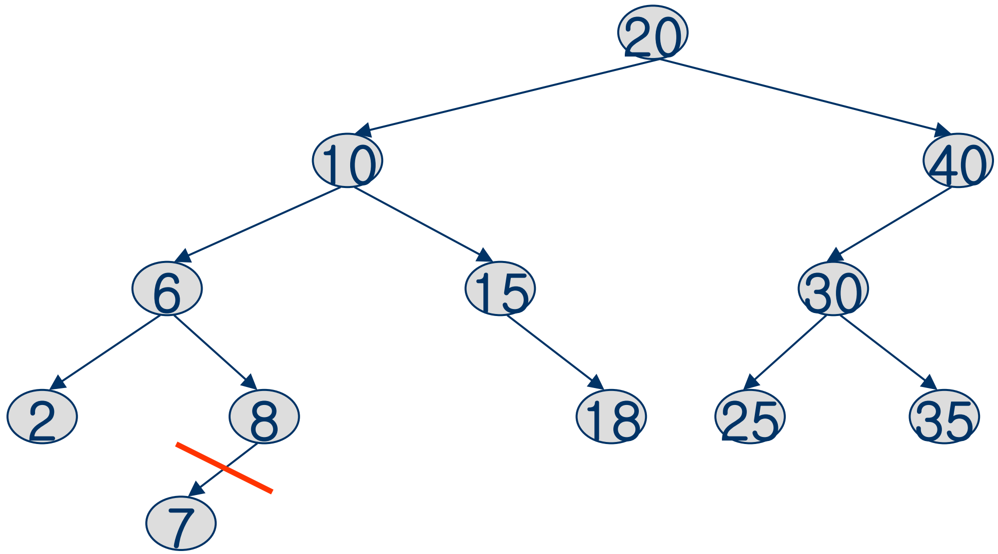


The Operation Delete(key, e)

- For deletion, there are three cases for the element to be deleted:
  1. Element is in a leaf.
  2. Element is in a degree 1 node (i.e., has exactly one nonempty subtree).
  3. Element is in a degree 2 node (i.e., has exactly two nonempty subtrees).

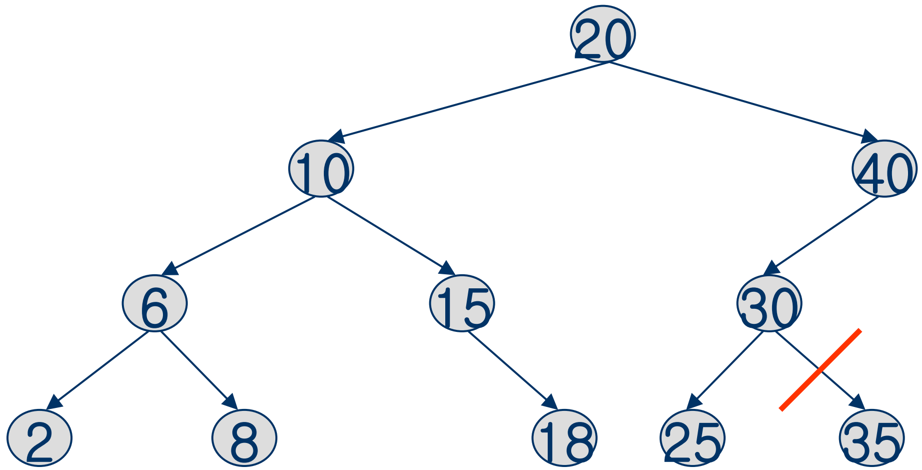
Case 1: Delete from a Leaf

- For case 1, we can simply discard the leaf node.
- Example, delete a leaf element. key=7

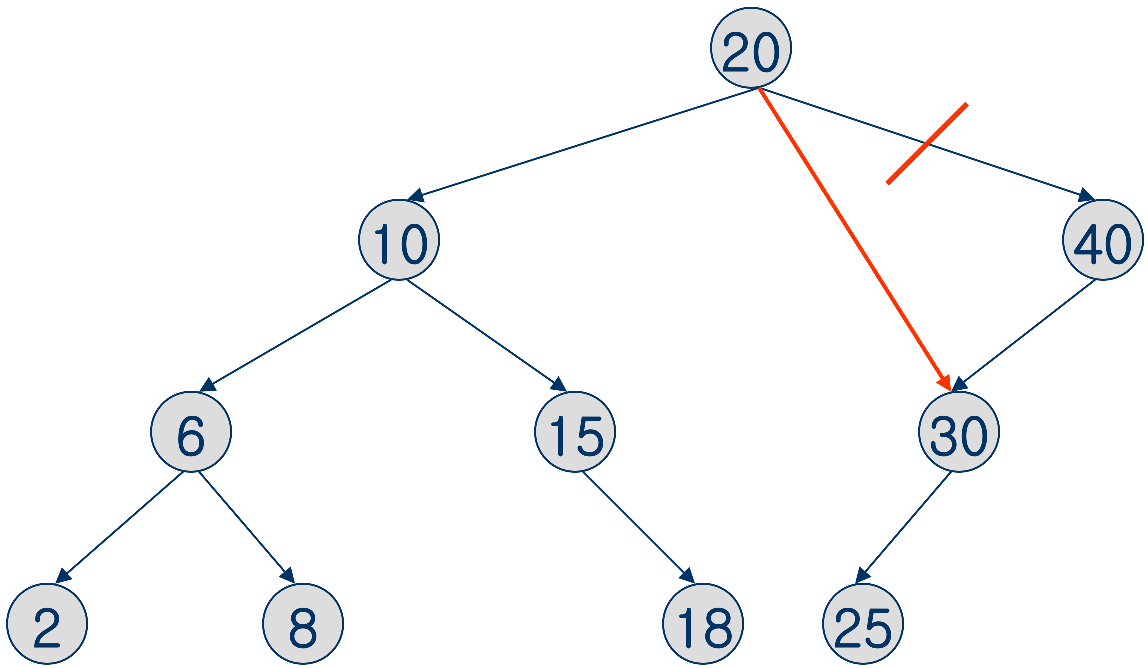


Case 1: Delete from a Leaf

Delete a leaf element. key=35



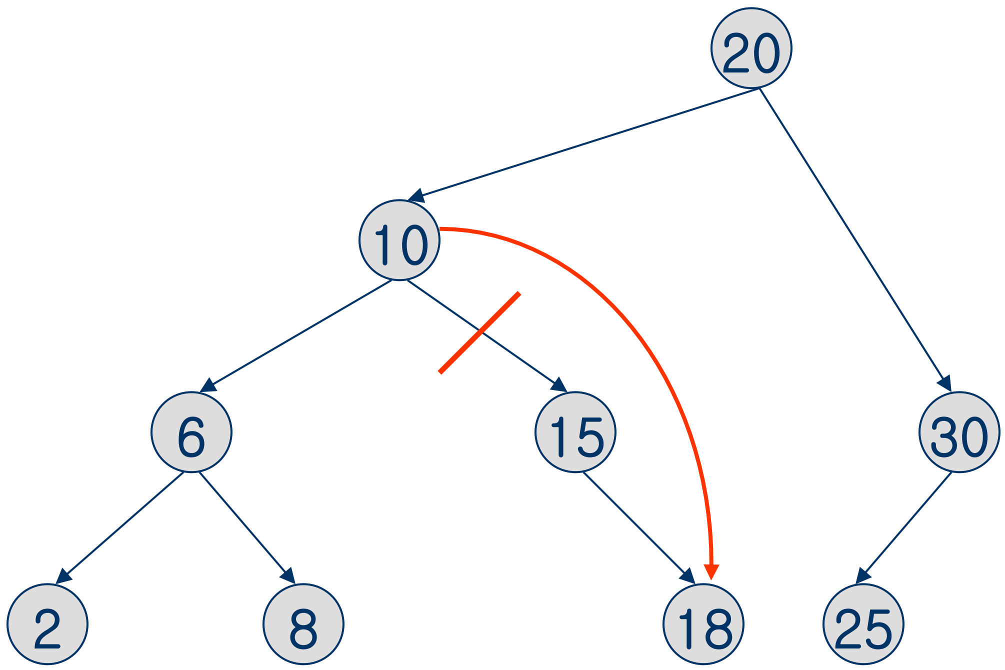
Case 2: Delete from a Degree 1 Node



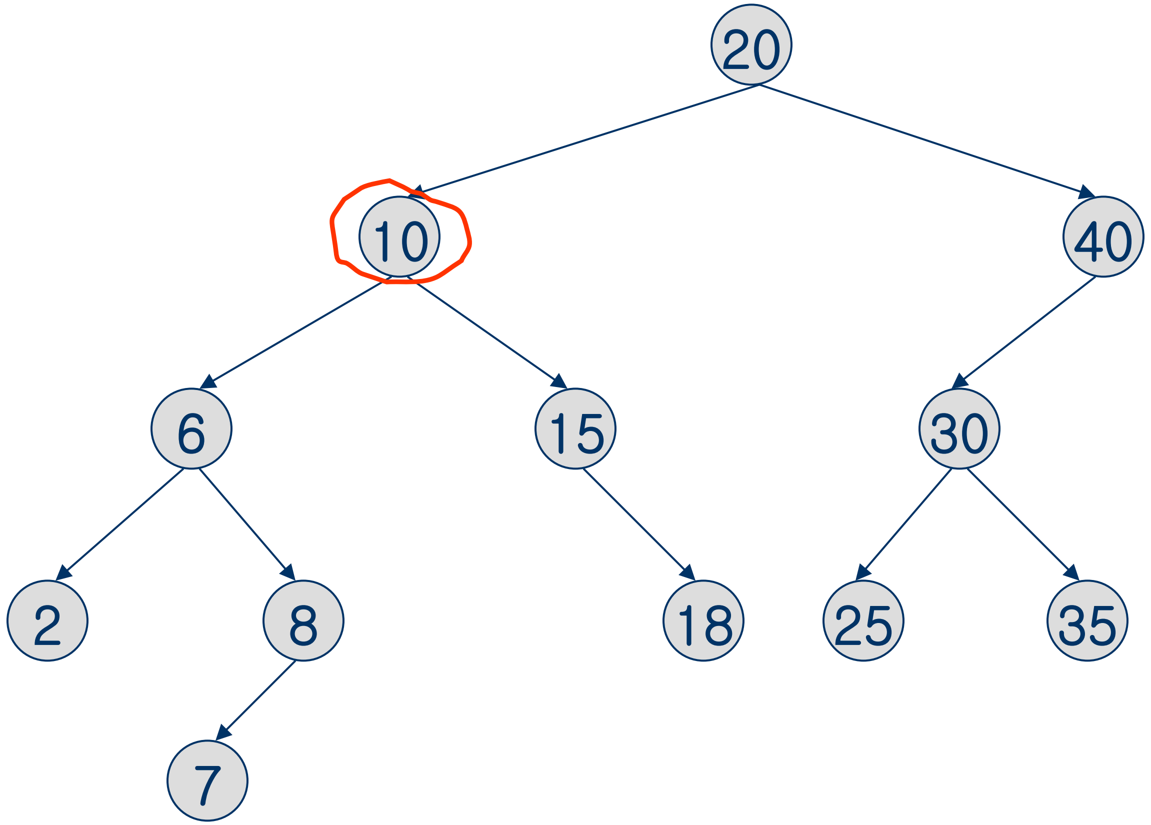
- Which nodes have a degree 1?
- Example: Delete key=40

Case 2: Delete from a Degree 1 Node

Delete from a degree 1 node. key=15

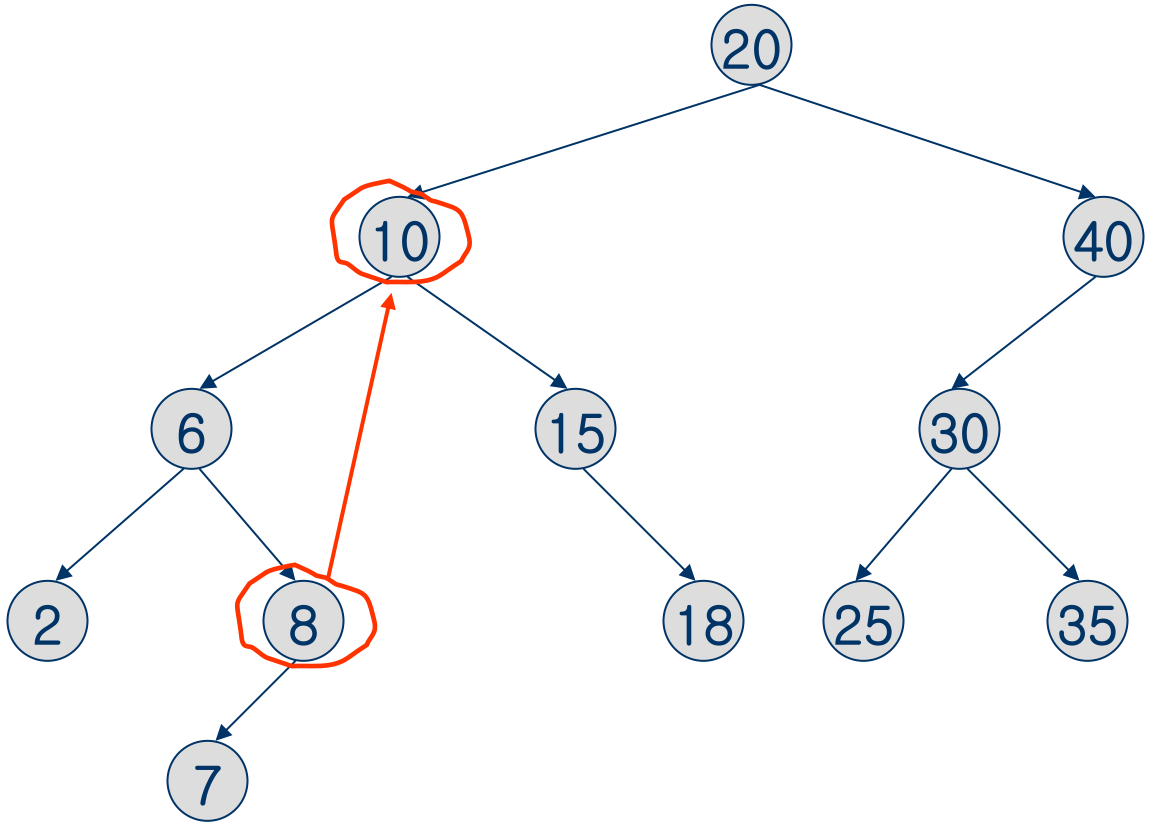


Case 3: Delete from a Degree 2 Node



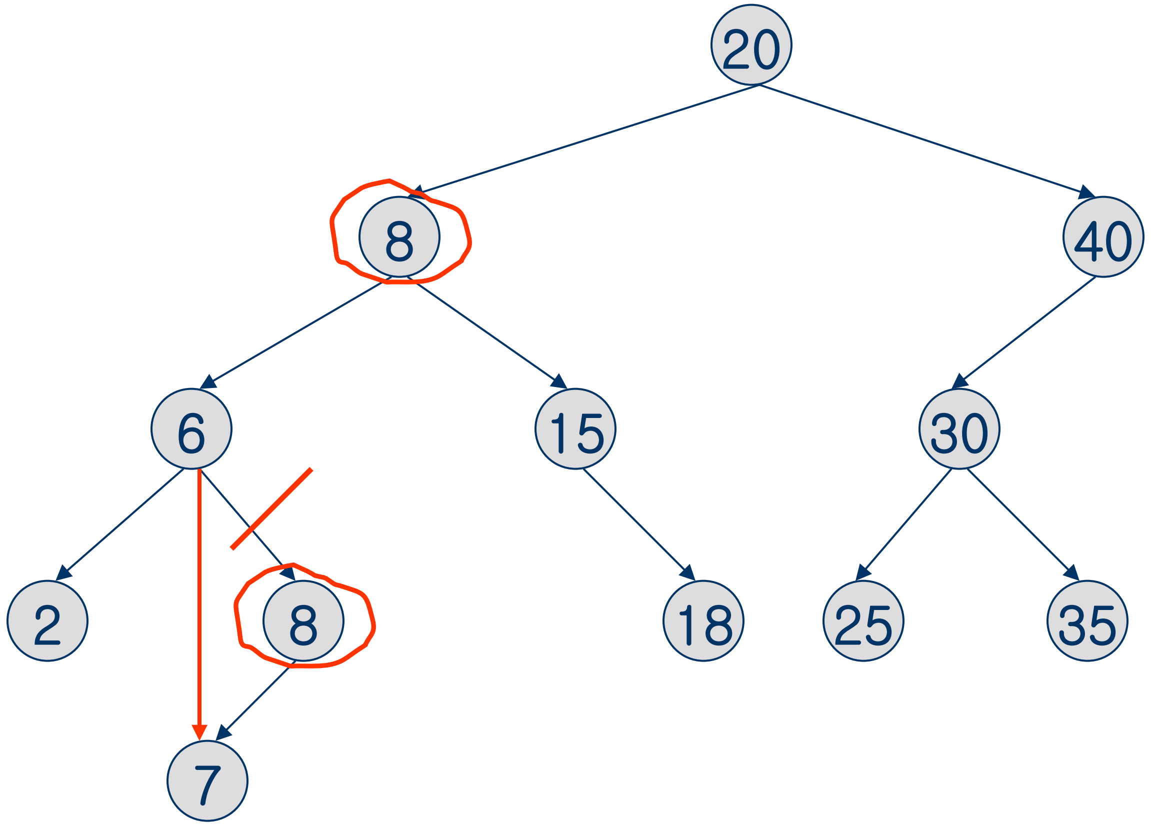
- Which nodes have a degree 2?
- Example: Delete key=10

Case 3: Delete from a Degree 2 Node



- Replace with the largest key in the left subtree (or the smallest in the right subtree)
- Which node is the largest key in the left subtree?

Case 3: Delete from a Degree 2 Node



The largest key must be in a leaf or degree 1 node.

Case 3: Delete from a Degree 2 Node

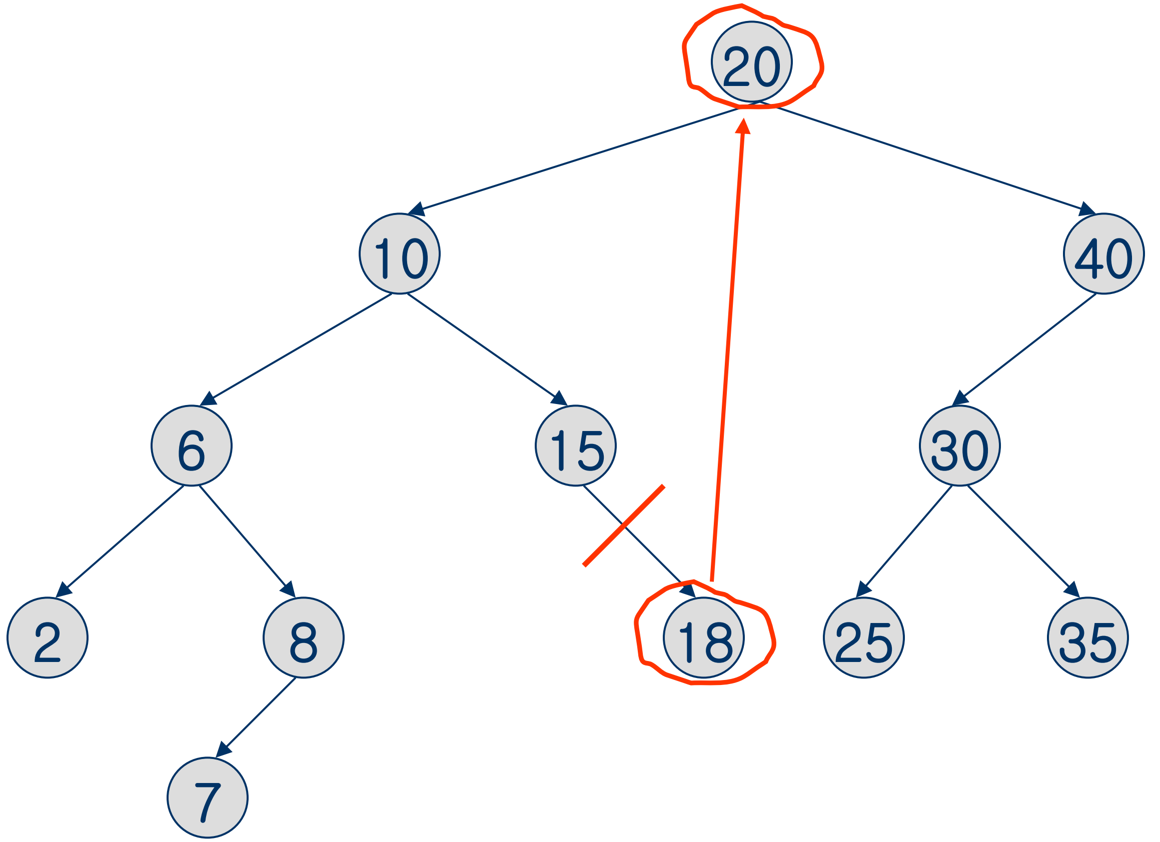
- Note that the node with largest key in the left subtree (as well as that with smallest in the right subtree) is guaranteed to be in a node with either zero or one nonempty subtree
- How can we find the node with **largest key** in the left subtree of a node?

  ➔ by moving to the root of that subtree and then following a sequence of right-child pointers until we reach a node whose right-child pointer is NULL
- How can we find the node with **smallest key** in the right subtree of a node?

  ➔ by moving to the root of that subtree and then following a sequence of left-child pointers until we reach a node whose left-child pointer is NULL

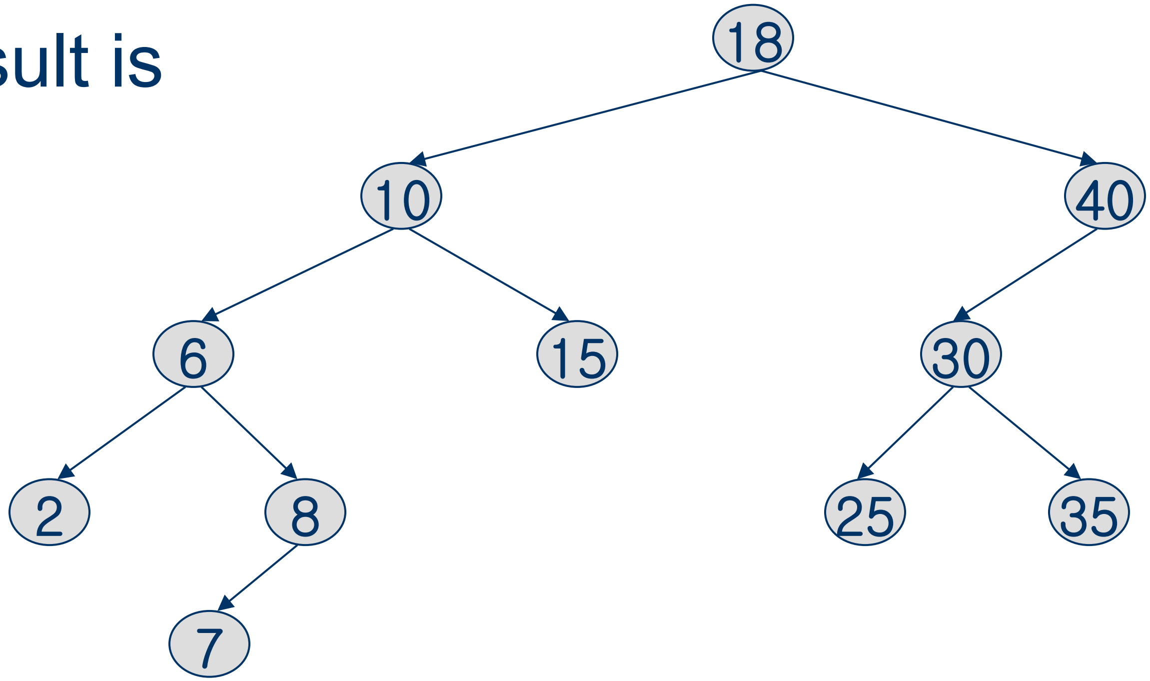
Another Delete from a Degree 2 Node

- Delete from a degree 2 node. key=20
- Replace with the largest in the left subtree.



Another Delete from a Degree 2 Node

The result is



- The time complexity of delete is O(*height*).
- See more delete examples in Figure 14.4
- See Program 14.6 for the delete operation code

Binary Search Trees with Duplicates

- We can remove the requirement that all elements in a binary search tree need distinct keys
- How?
  - Replace “smaller” in property 2 by “smaller or equal to”
  - Replace “larger” in property 3 by “larger or equal to”
- Then binary search trees can have duplicate keys

Things to Remind

- Questions like
  - A lot of short answers
  - Fill in the black, like coding questions
  - Time complexity for sure
  - You must know how to read ADT defintions.
    - Let’s discuss more in class

- Offline final exam
  - Dec. 14
  - I will make easy.