Data Structure

Lab Session #5:
Binary Trees

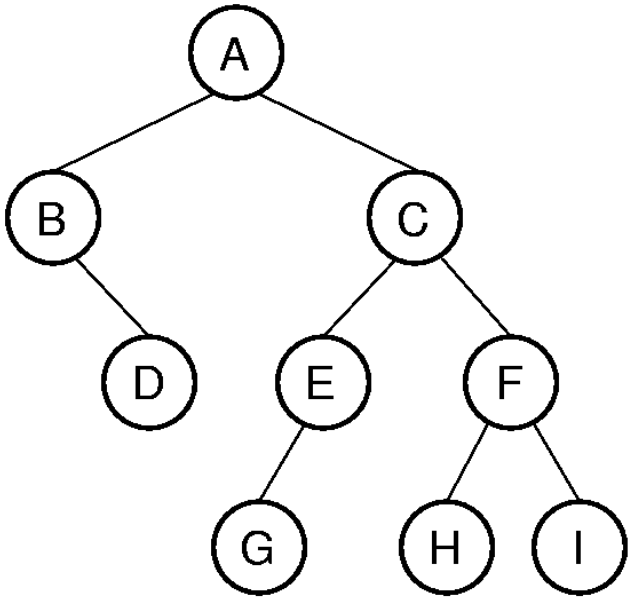
U Kang
Seoul National University

Goals

- Implement **Binary Search Tree, Traverses**
  - Fill your code in the methods in BinaryTree.Node class.
- Print the sample output corresponding to the sample input
  - Please carefully observe the I/O specification.

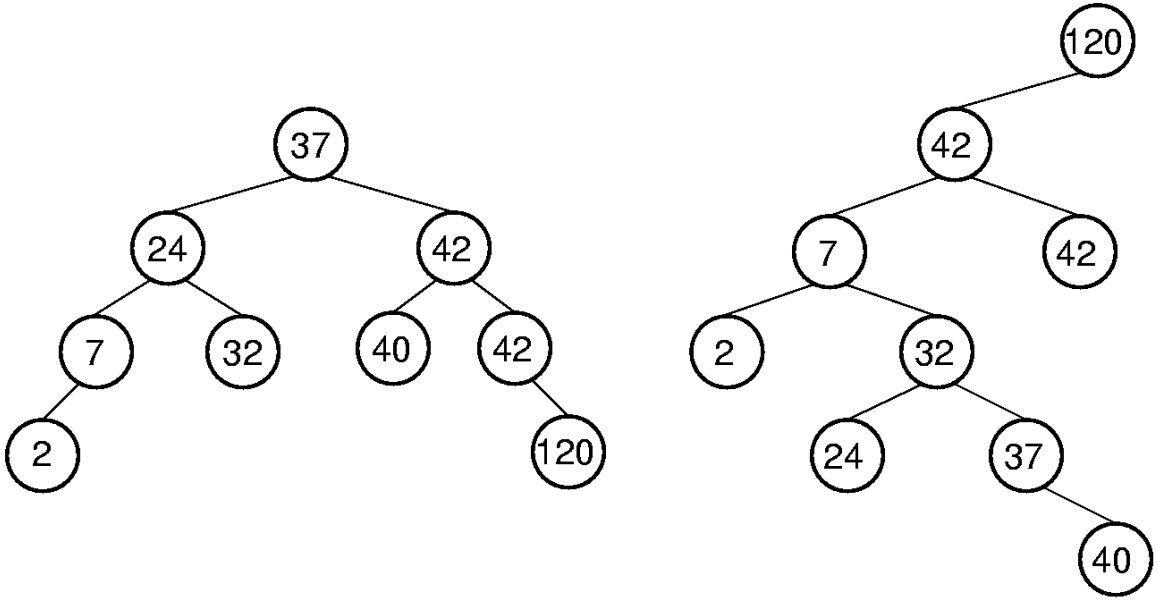
Binary Trees

- A binary tree is made up of a finite set of nodes that is either empty or consists of a node called the root together with two binary trees, called the left and right subtrees, which are disjoint from each other and from the root.



Binary Search Trees

- BST Property: All elements stored in the left subtree of a node with value K have values $< K$. All elements stored in the right subtree of a node with value K have values $\geq K$.



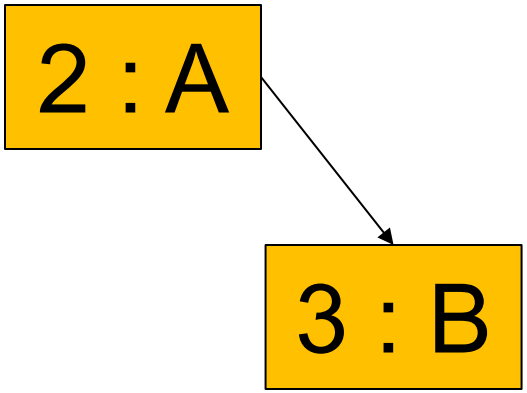
BST insert

- insert 2 A

2 : A

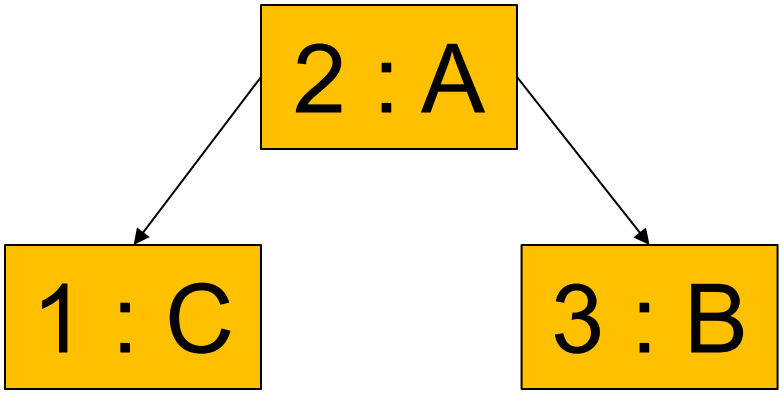
BST insert

- insert 3 B



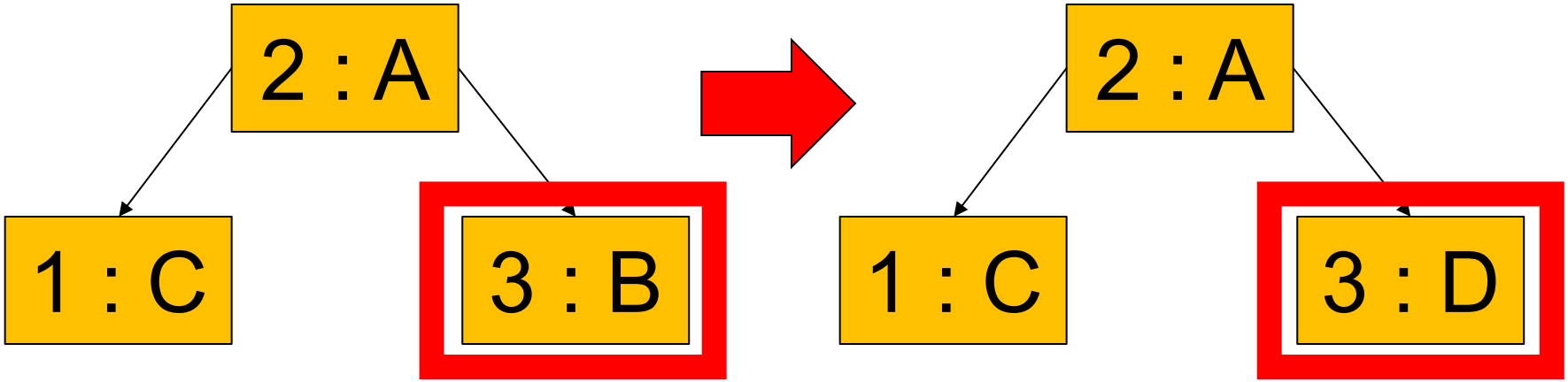
BST insert

- insert 1 C



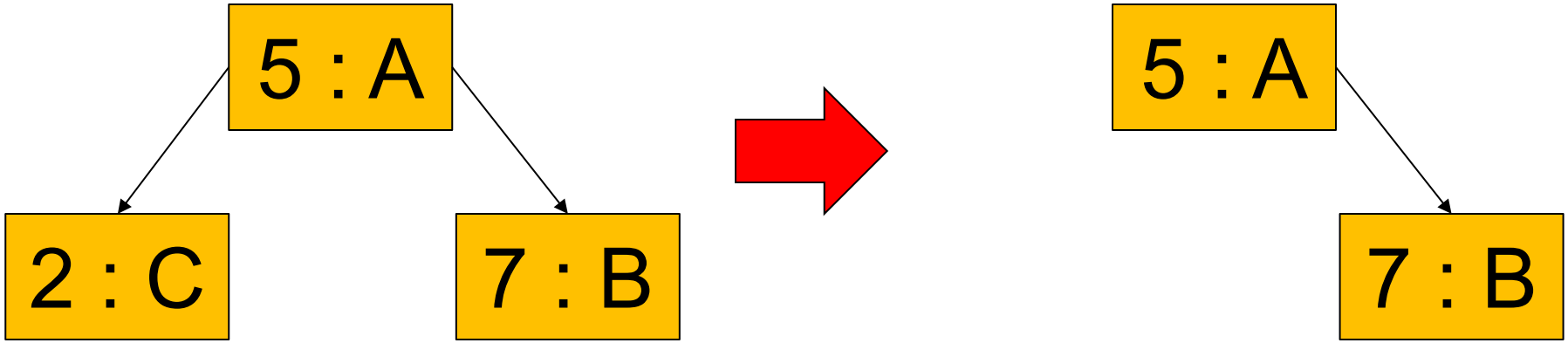
BST insert

- insert 3 D
  - If BST already contains given key, replace the value.



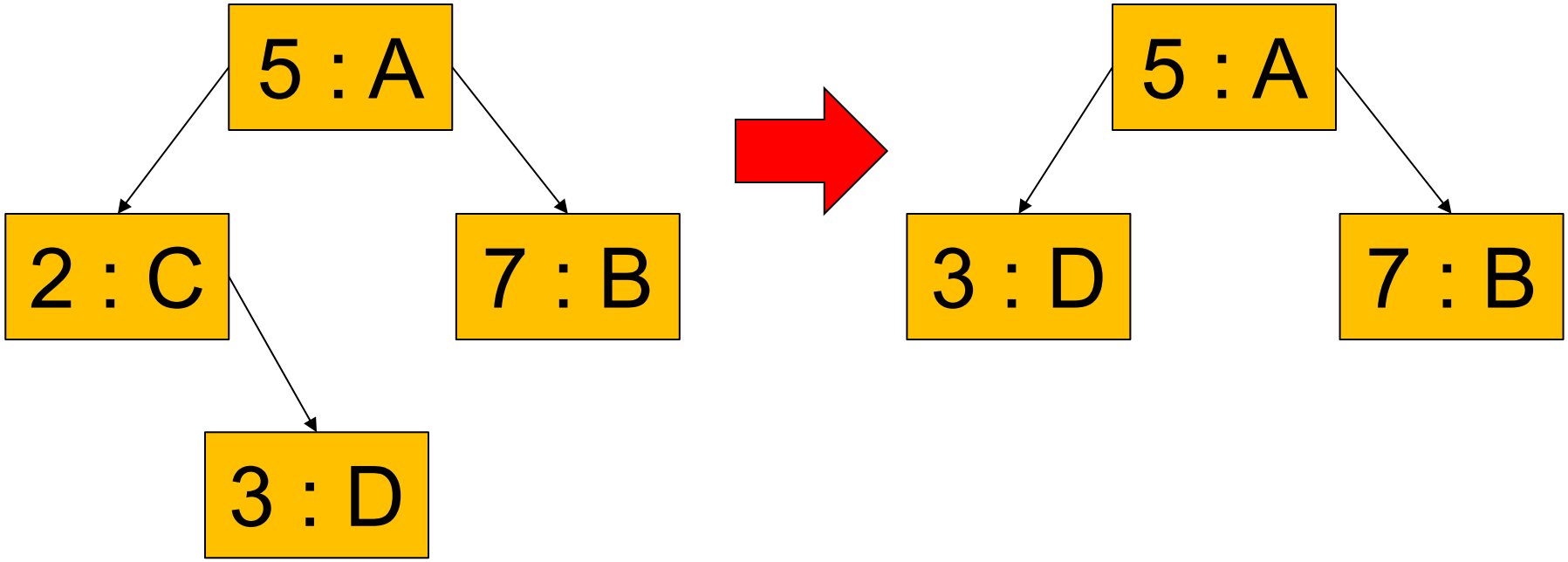
BST delete

- delete 2
  - If there are no children in the node to be deleted



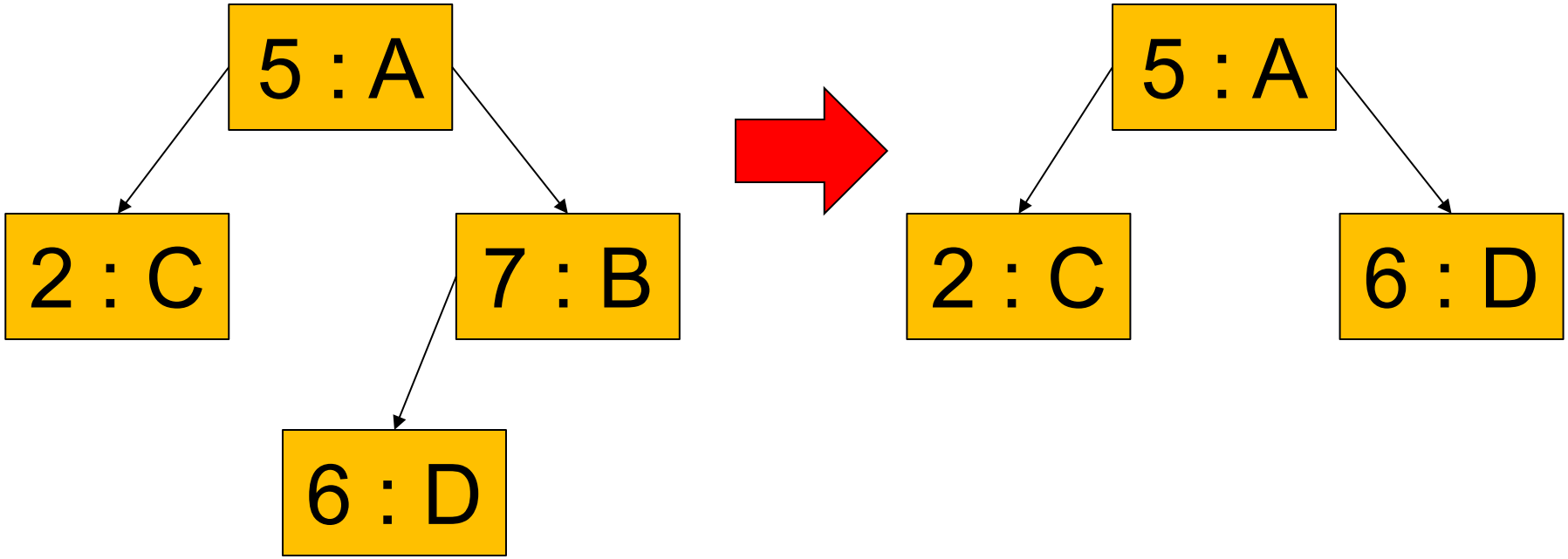
BST delete

- delete 2
  - If there is a child in the node to be deleted



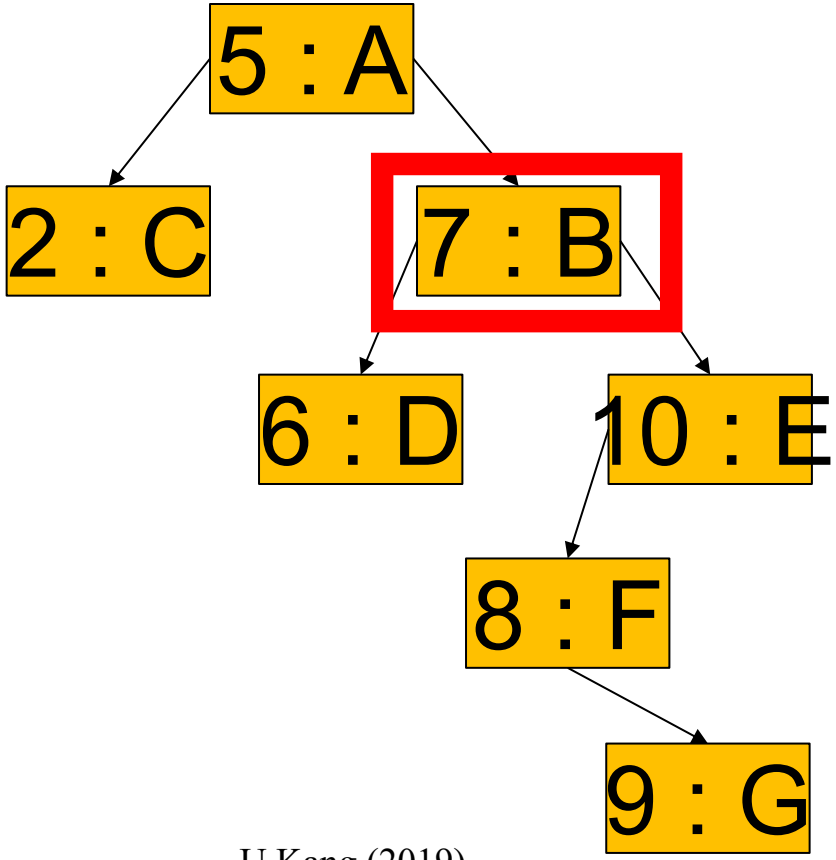
BST delete

- delete 7
  - If there is a child in the node to be deleted



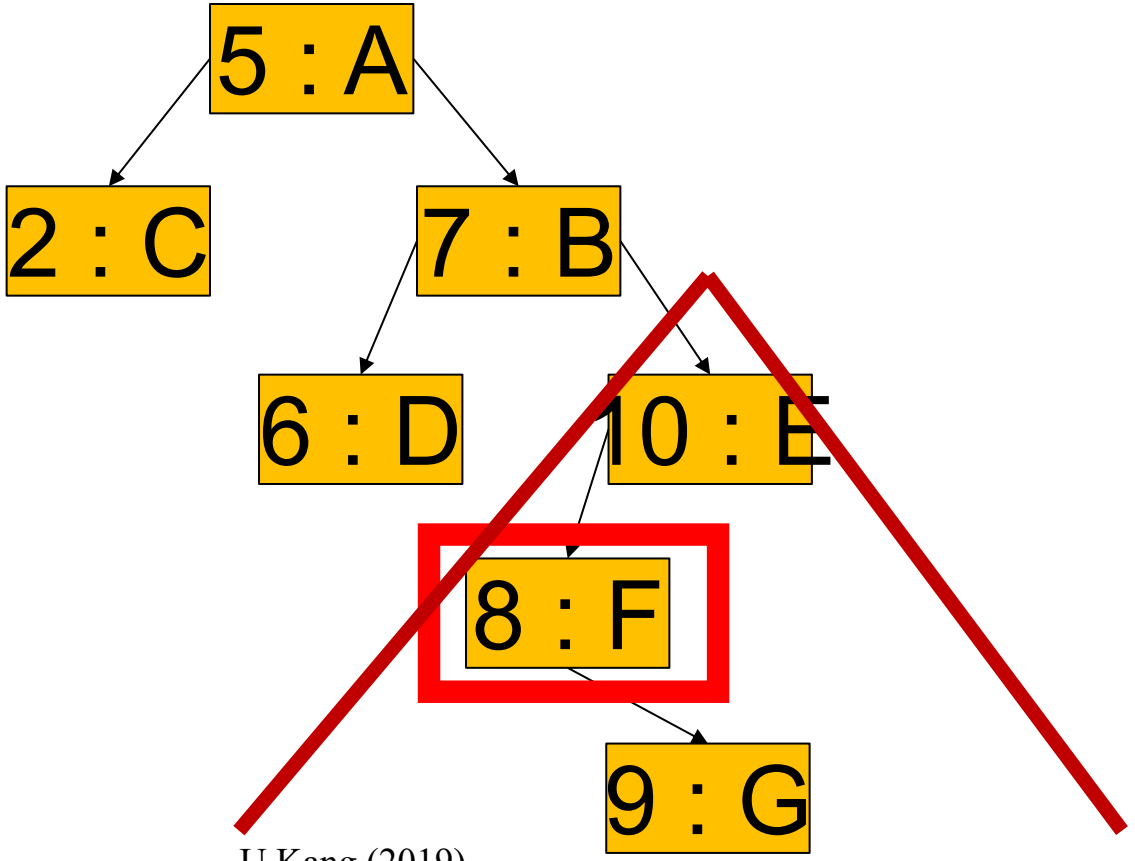
BST delete

- delete 7
  - If there are two children in the node to be deleted



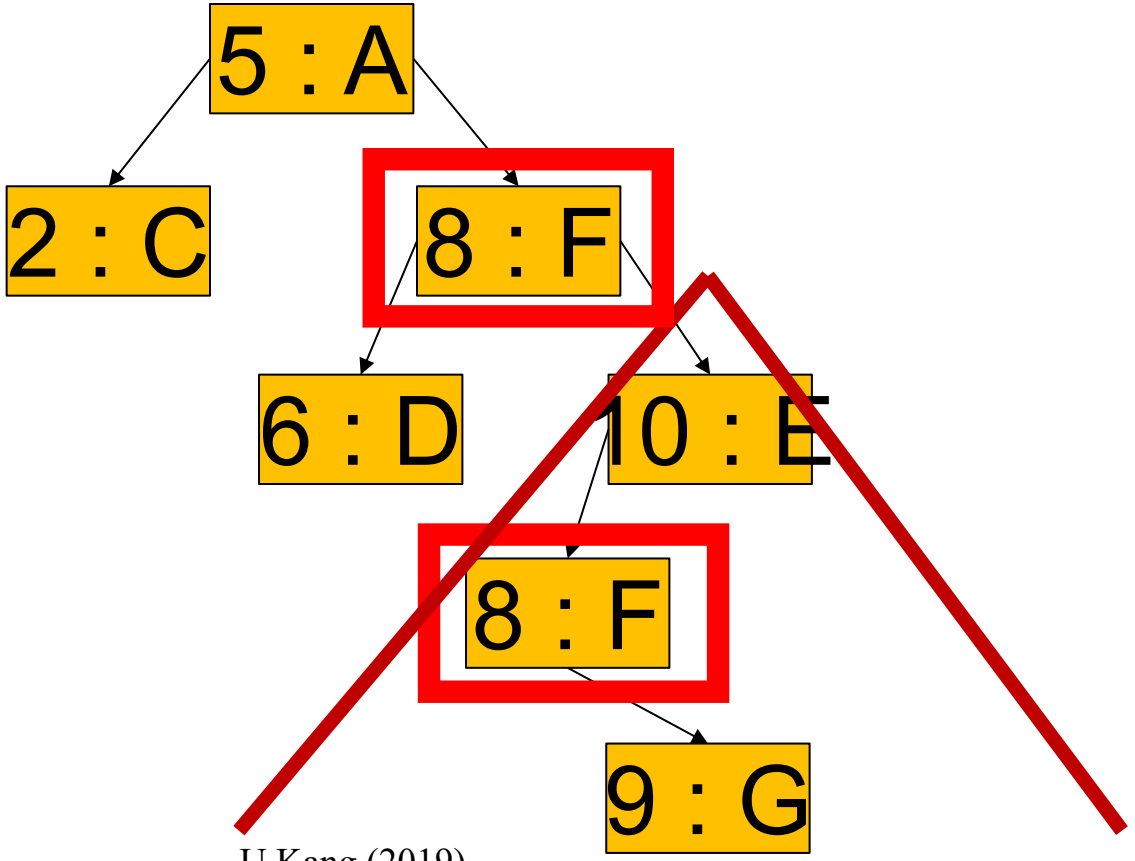
BST delete

- delete 7 (continued)
  - (step1) Find the minimum-key node in the right subtree



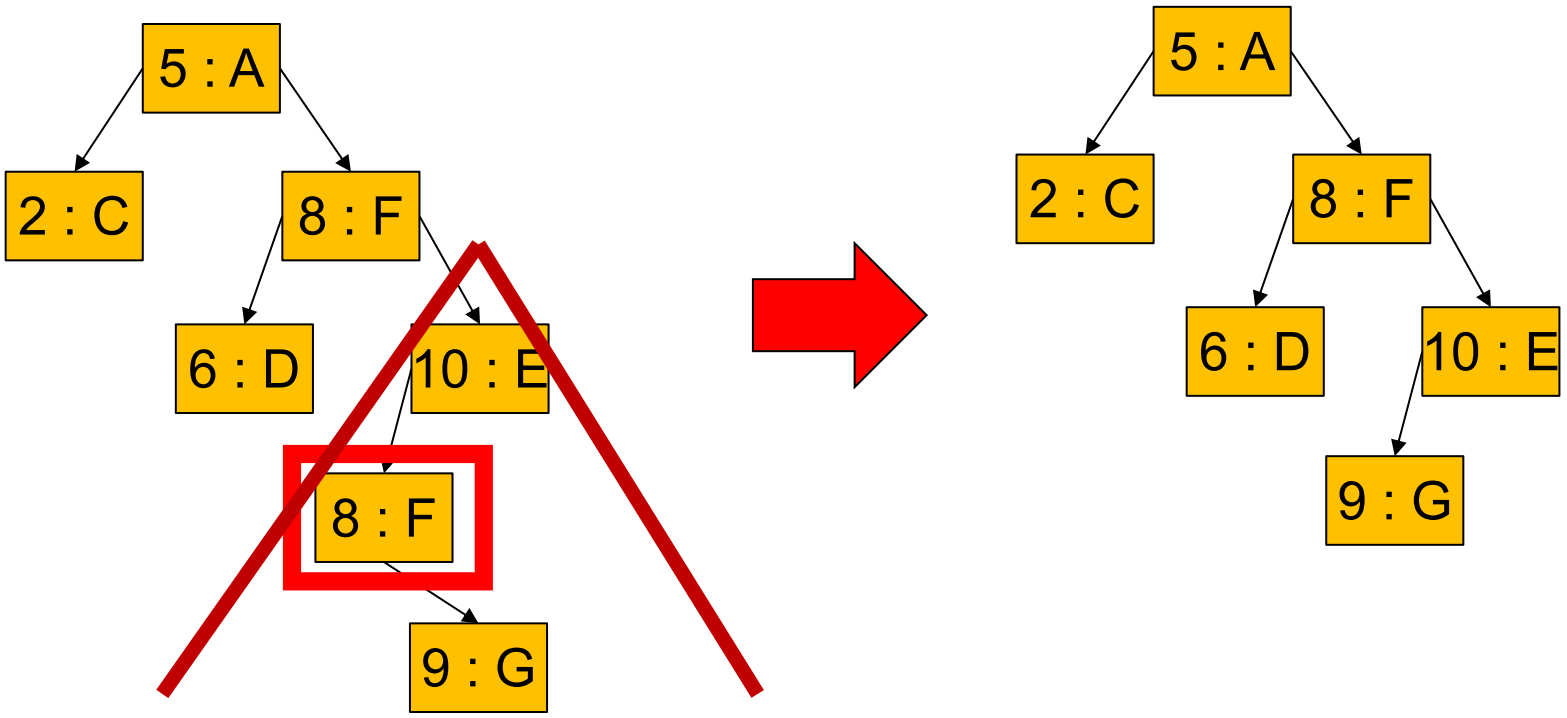
BST delete

- delete 7 (continued)
  - (step2) exchange the (key, value) of node to be deleted



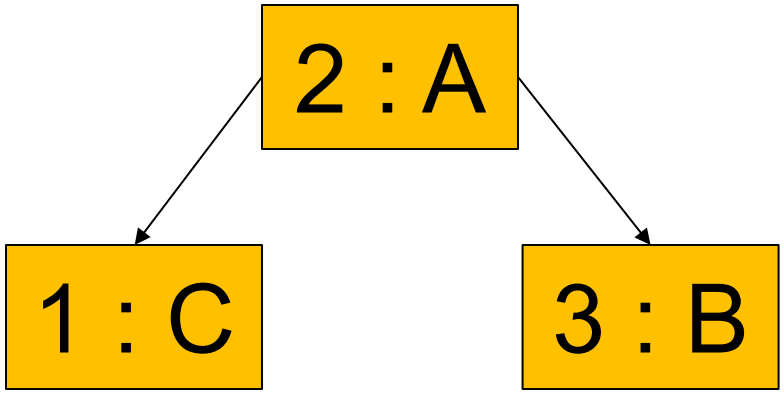
BST delete

- delete 7 (continued)
  - (step3) delete the minimum node in the right subtree



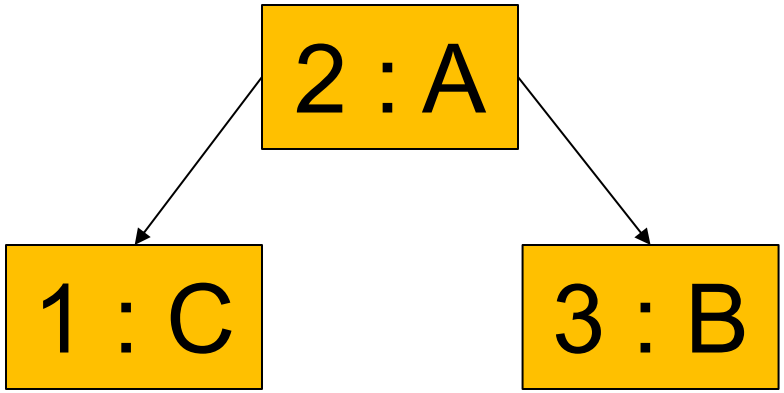
BST traverse

- Preorder
  - Visit the node before its children.
  - Let be the left first.
  - [2:A][1:C][3:B]



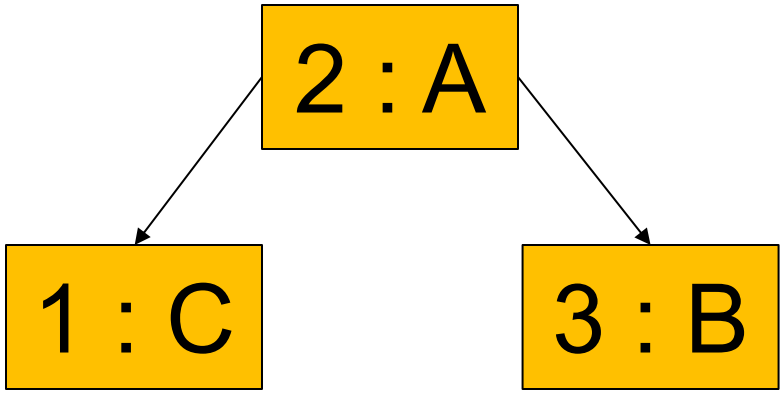
BST traverse

- Inorder
  - Visit the node after the left child.
  - Then, visit right child.
  - [1:C][2:A][3:B]



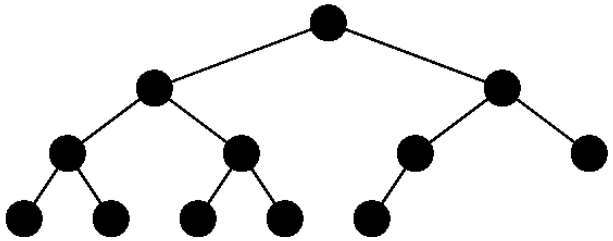
BST traverse

- Postorder
  - Visit the node after its children.
  - Let be the left first.
  - [1:C][3:B][2:A]



Complete Binary Trees

- Complete binary tree:
  - 1) all levels except possibly the last level are completely full
  - 2) the bottom level has all nodes to the left side.



Complete binary tree

I/O Specification

- insert

Input form	Output form
`insert (key) (value)`	
Description	
- **Insert a node of (key, value) into the binary search tree.** - **The type of (key) is integer and (value) is string.** - **Each (key) is unique in the binary search tree.**	
Example Input	**Example Output**
`insert 1 BTS`	

I/O Specification

- find

Input form	Output form
`find (key)`	
Description	
- **Find a node in the binary search tree by key and return the value.** - **Return null if there is no matching node.**	

Example Input	Example Output
`find 1`	

I/O Specification

- preorder / inorder / postorder

Input form	Output form
`preorder`	`(a series of key,value sets)`
Description	
- **Do a preorder traversal for the binary search tree, and print out key-value sets in order of the traversal.** - **(a series of key-value sets) is a sequence of the key-value sets.**	
Example Input	**Example Output**
`preorder`	`[1:BTS][2:IOI]`

I/O Specification

- iscomplete

Input form	Output form
`iscomplete`	

Description

- **Check if all levels except the last level are completely full**
- **Check if the bottom level has all nodes to the left side**
- **Return "True" if it's complete binary tree, otherwise return "False"**

Example Input	Example Output
`iscomplete`	`True`

I/O Specification

- height

Input form	Output form
`height`	
Description	
- **Return the height of the subtree, the root of which is this node.**	

Example Input	Example Output
`height`	

I/O Specification

- delete

Input form	Output form
`delete (key)`	
Description	
- **Delete a node by the key.** - **If there is no matching node in the binary search tree, do nothing.** - **Return the node parent should point.**	
Example Input	**Example Output**
`delete 1`	

Sample Input

```
insert 1 BTS
iscomplete
find 1
insert 2 Blackpink
insert 3 Redvelvet
insert 4 TWICE
insert 5 IZONE
insert 6 EXO
insert 7 GFriend
preorder
height
delete 4
preorder
height
delete 1
delete 2
delete 3
```

```
delete 5
delete 6
delete 7
preorder
height
insert 4 TWICE
insert 2 Blackpink
insert 3 Redvelvet
insert 1 BTS
insert 6 EXO
insert 5 IZONE
insert 7 Gfriend
delete 4
preorder
inorder
postorder
height
find 5
```

Sample Output

The tree is complete binary tree.
Value for 1 is BTS
preorder : [1:BTS][2:Blackpink][3:Redvelvet][4:TWICE][5:IZONE][6:EXO][7:GFriend]
Height of this tree is 7
preorder : [1:BTS][2:Blackpink][3:Redvelvet][5:IZONE][6:EXO][7:GFriend]
Height of this tree is 6
preorder : None
Height of this tree is 0
preorder : [5:IZONE][2:Blackpink][1:BTS][3:Redvelvet][6:EXO][7:GFriend]
inorder : [1:BTS][2:Blackpink][3:Redvelvet][5:IZONE][6:EXO][7:GFriend]
postorder : [1:BTS][3:Redvelvet][2:Blackpink][7:GFriend][6:EXO][5:IZONE]
Height of this tree is 3
Value for 5 is IZONE

Questions?