2-3 Trees

CS223: Data Structures

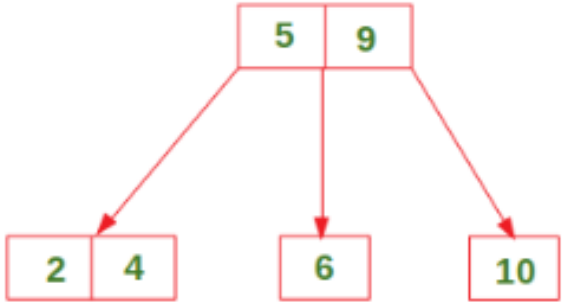
2-3 trees

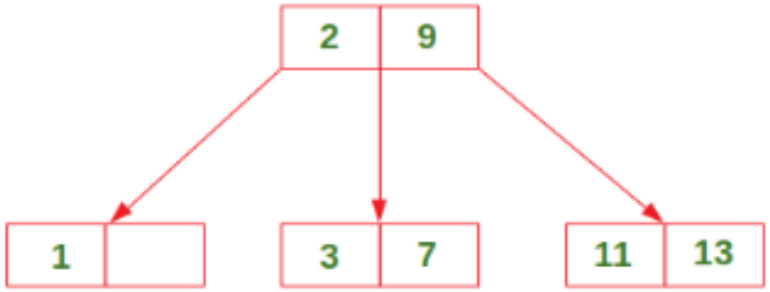
- Another balanced tree.
- 2-3 tree is not a binary tree
- In this tree, all the leaf nodes are at the same level (bottom level).
- Keys at leaf nodes are ordered left to right.

2-3 trees

- In a 2-3 tree, each interior node has either two or three children.
  - Nodes with two children are called 2-nodes. The 2-nodes have one data value and two children.
    - All keys in left subtree are smaller than key
    - All keys in right subtree are greater than key
  - Nodes with three children are called 3-nodes. The 3-nodes have two data values (k1,k2) and three children.
    - Keys in one subtree are smaller than k1
    - Keys in one subtree are greater than k2
    - Keys in one subtree are strictly between k1 and k2
- To balance the tree, use the move-up then split principle
  - Move up the middle key the split the child node

2-3 Trees





Search

Search a key K in given 2-3 tree T

Basic cases:

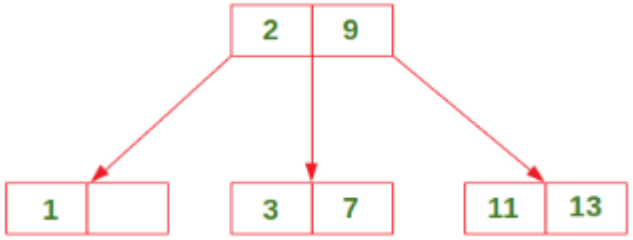
- If T is empty, return False (key cannot be found in the tree).
- If current node contains data value which is equal to K, return True.
- If we reach the leaf-node and it doesn't contain the required key value K, return False.

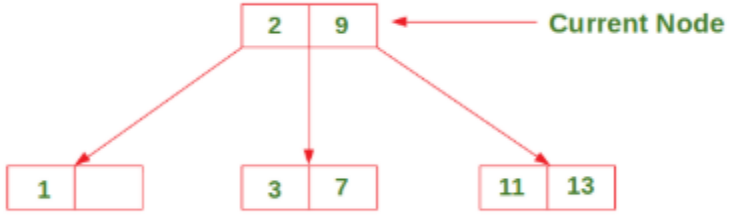
Recursive Calls:

- If **K** < currentNode.leftVal, we explore the left subtree of the current node.
- Else if currentNode.leftVal < **K** < currentNode.rightVal, we explore the middle subtree of the current node.
- Else if **K** > currentNode.rightVal, we explore the right subtree of the current node.

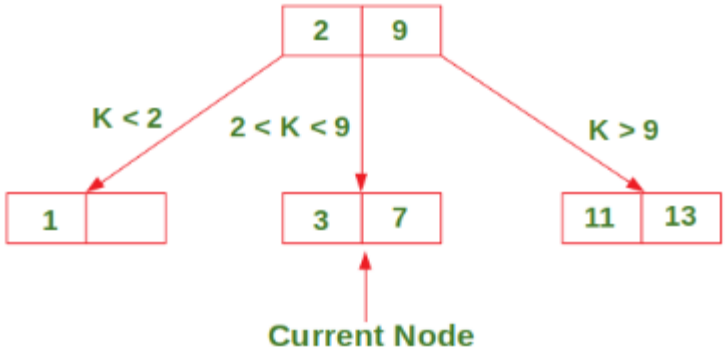
Example

Search for 5



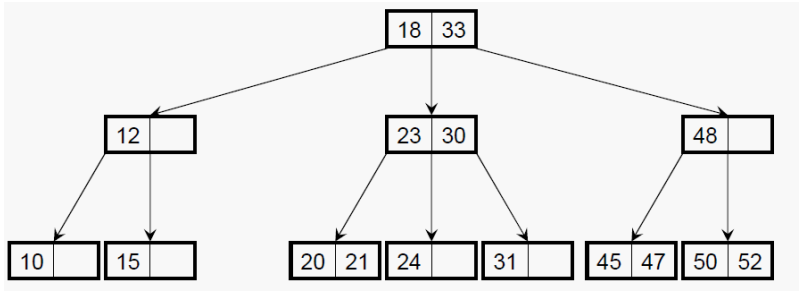


5 Not found

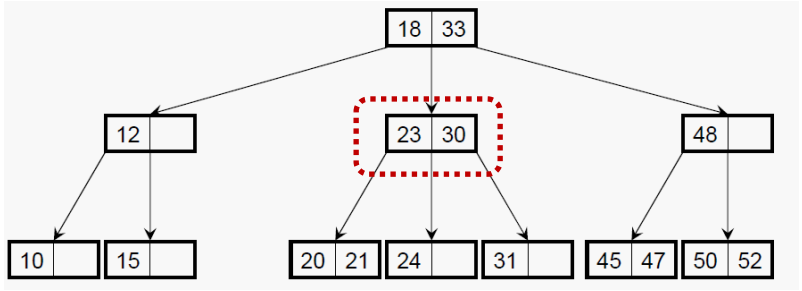


Search Example

Search for 21

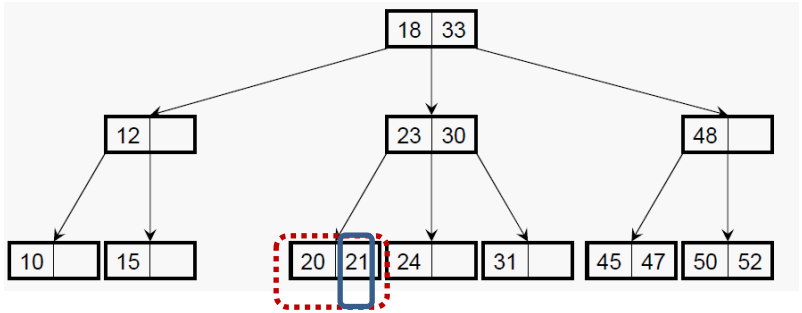


18 < 21 < 33, move to the middle child



21 < 23, move to the left child

Key 21 found



Insertion

To insert a new value in the 2-3 tree:

- An appropriate position of the value is located in one of the leaf nodes.
- If after insertion of the new value, the properties of the 2-3 tree do not get violated, then insertion is over.
- Otherwise, if any property is violated, then the violating node must be split

Insertion

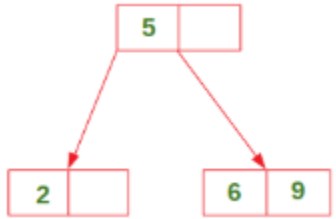
Three possible cases:

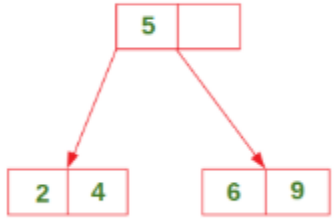
1. Insert in a node with only one data element.
2. Insert in a node with two data elements whose parent contains only one data element.
3. Insert in a node with two data elements whose parent also contains two data elements.

Insertion

1. Insert in a node with only one data element.

Insert 4

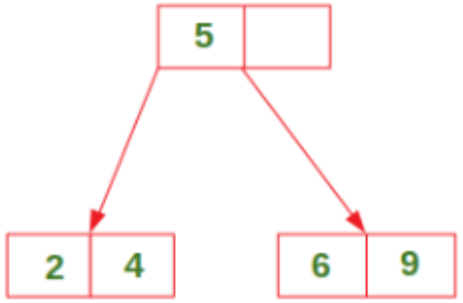




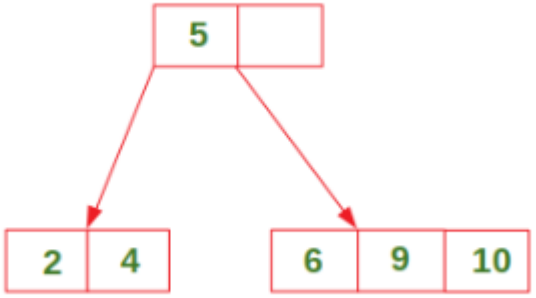
Insertion

2. Insert in a node with two data elements whose parent contains only one data element.

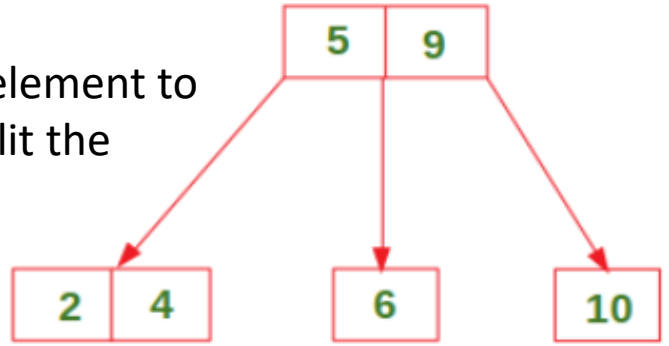
Insert 10



Temporary node with three data elements



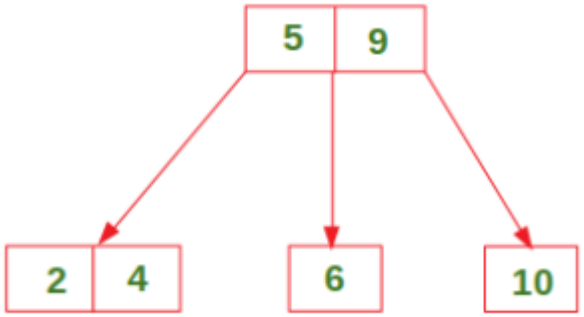
Move the middle element to the parent, and split the current node



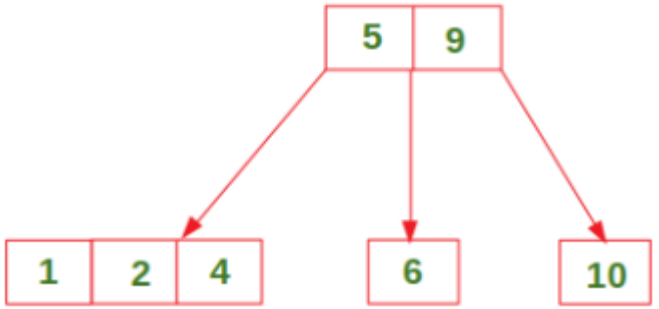
Insertion

3. Insert in a node with two data elements whose parent also contains two data elements.

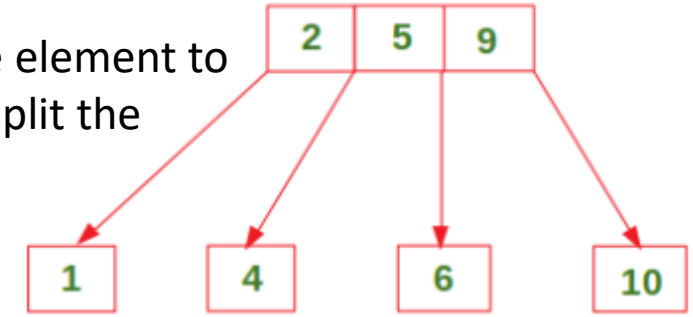
Insert 1



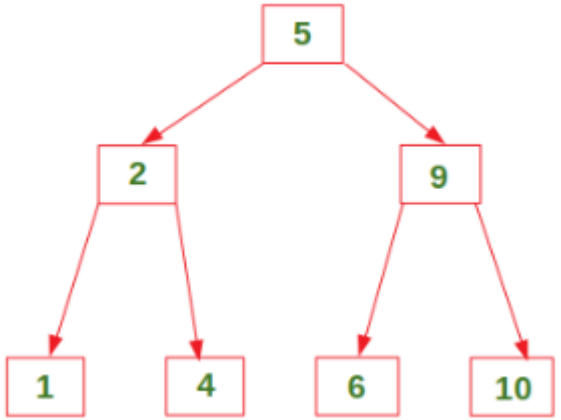
Temporary node with three data elements



Move the middle element to the parent, and split the current node

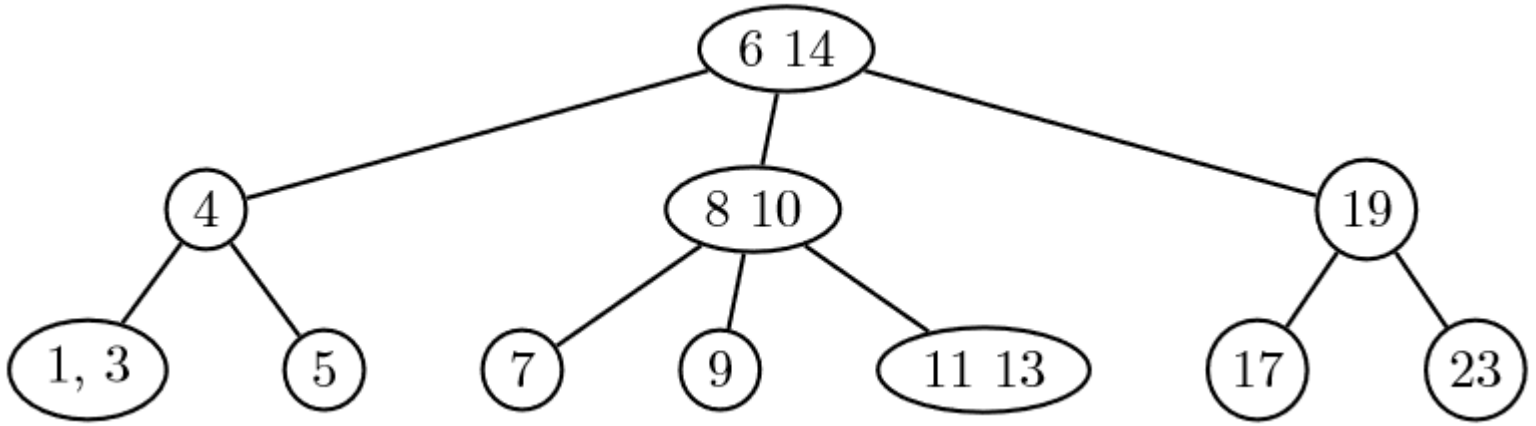


Move the middle element to the parent, and split the current node

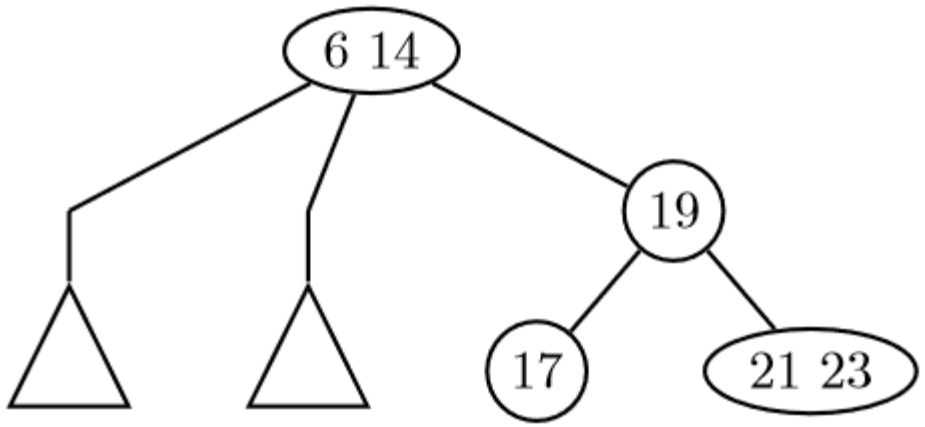


Insertion Examples

Insert 21

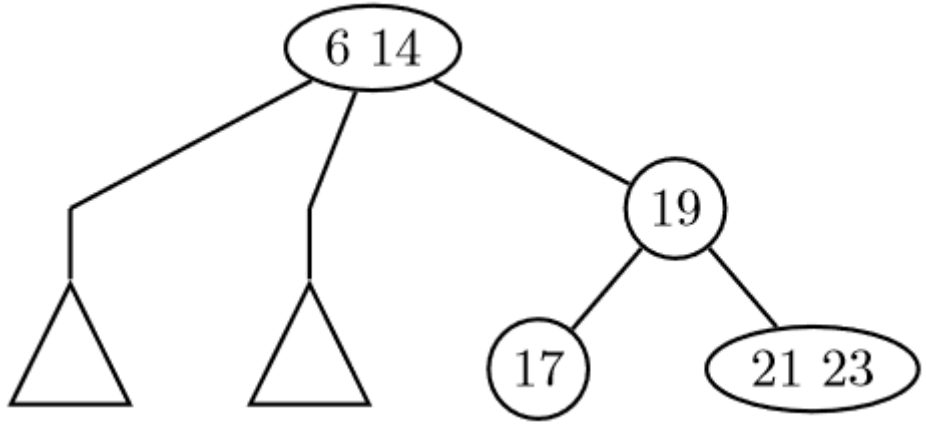


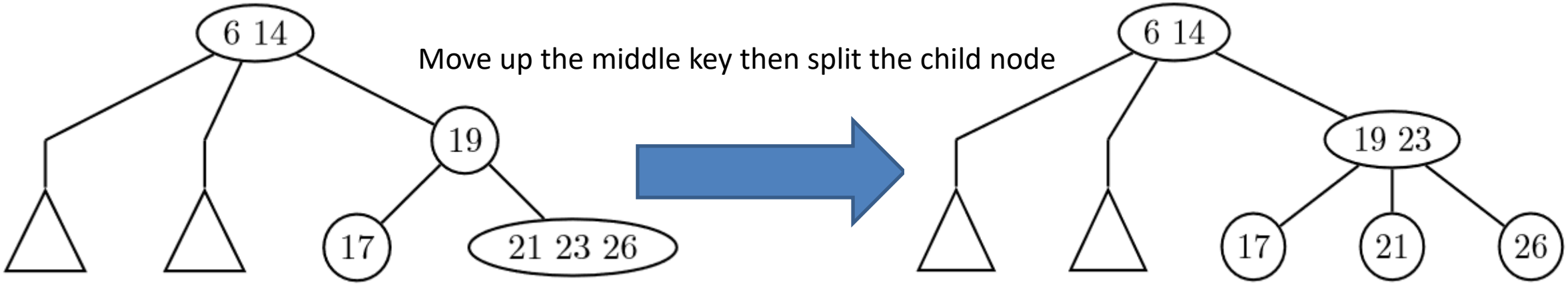
Case 1



Insertion Examples

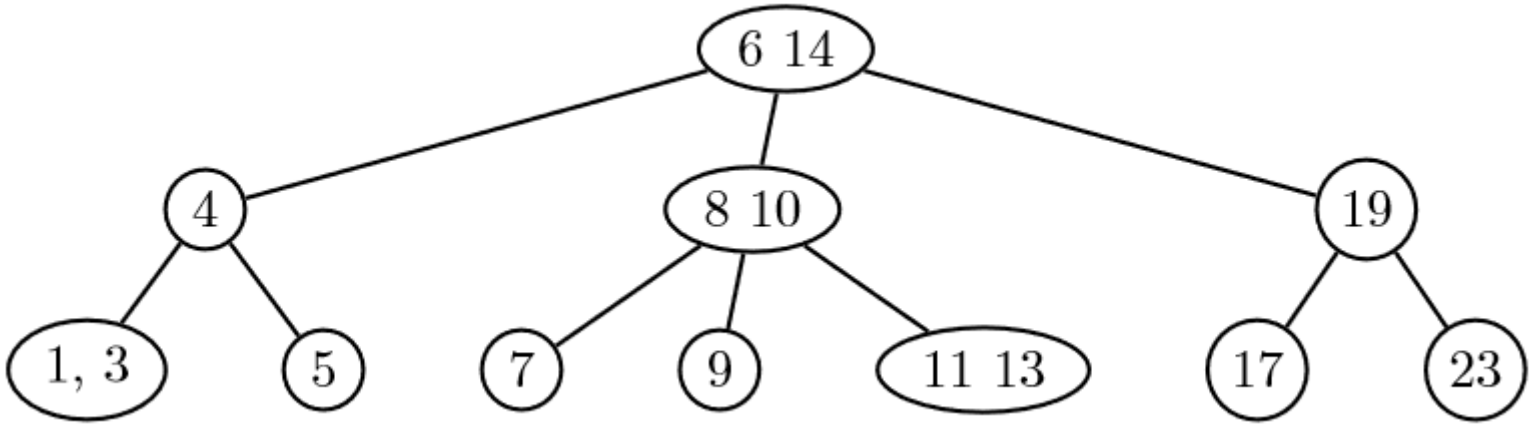
Insert 26



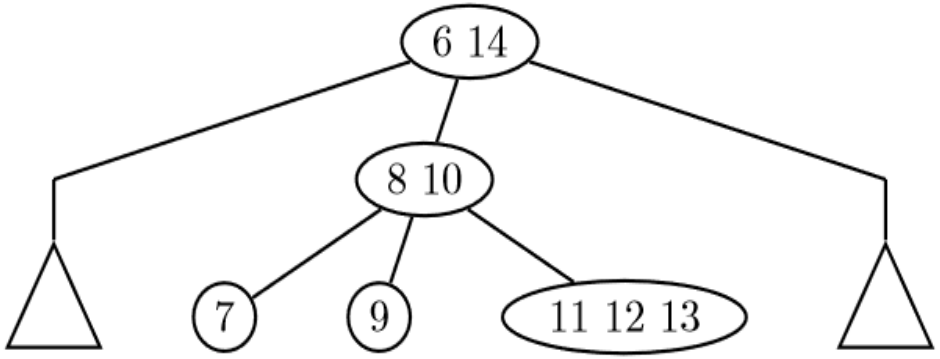


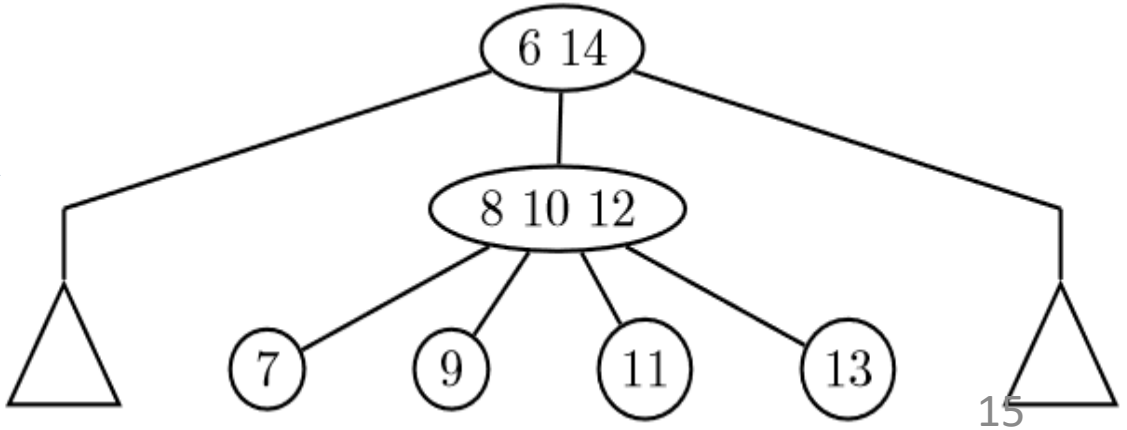
Insertion Examples

Insert 12



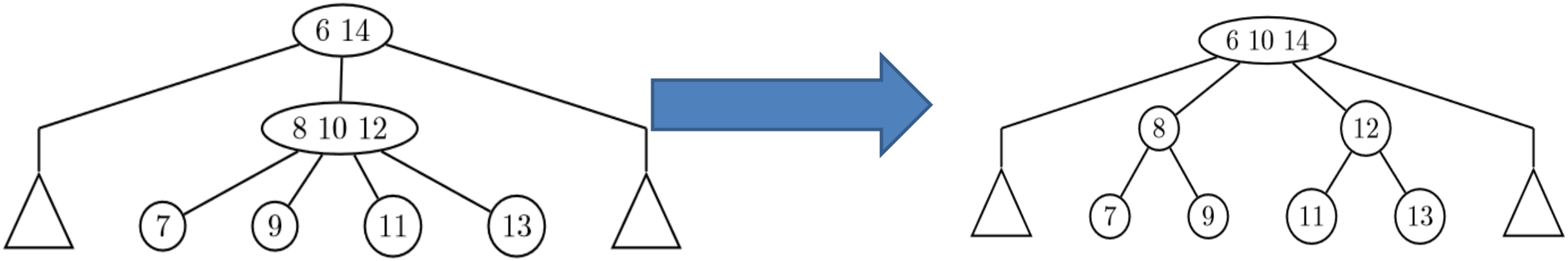
Move up the middle key then split the child node



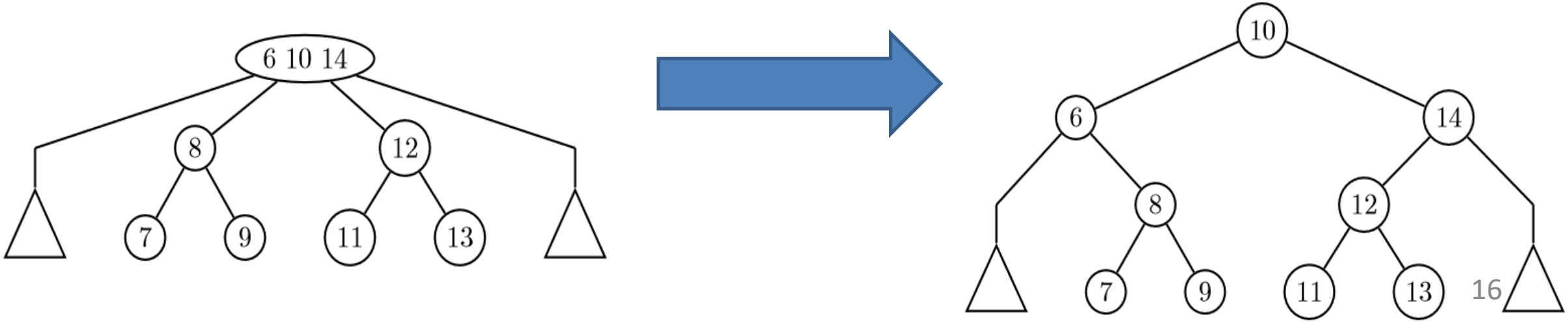


Insertion Examples

Move up the middle key then split the child node

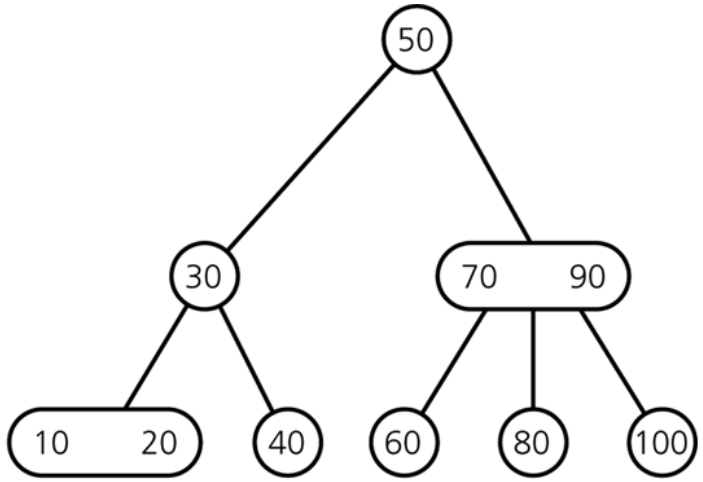


Move up the middle key then split the child node

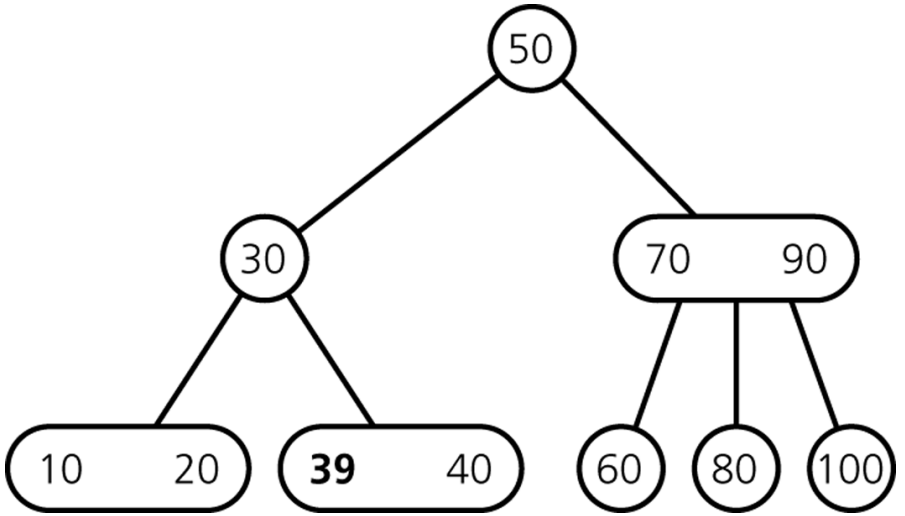


Insertion Examples

Insert 39

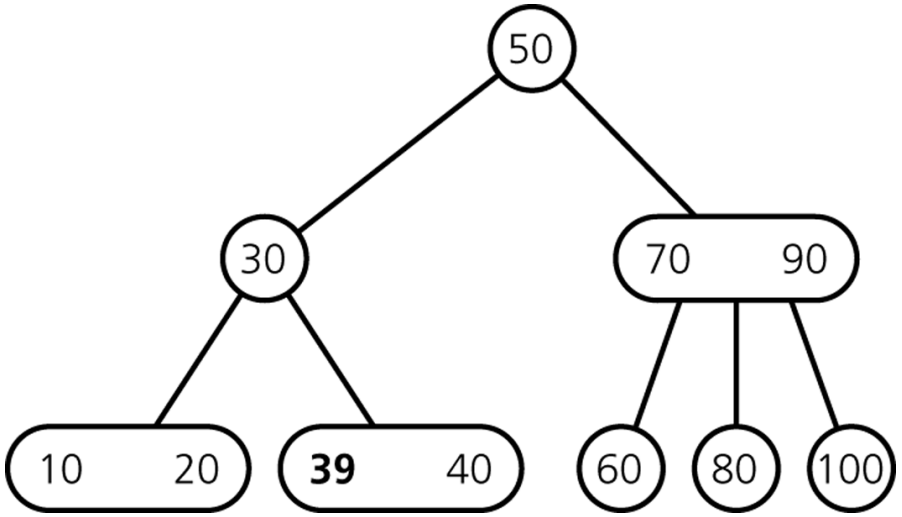


Case 1

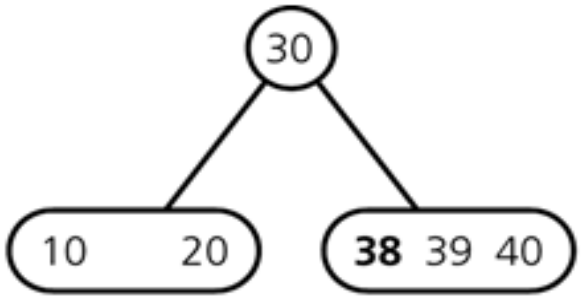


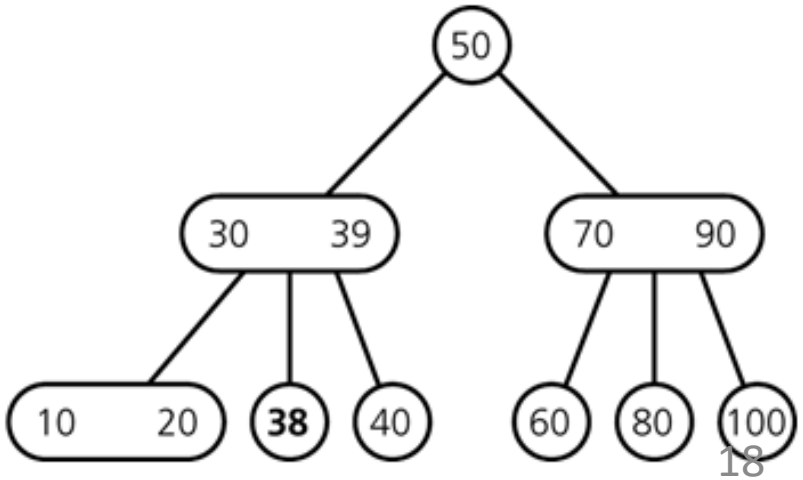
Insertion Examples

Insert 38



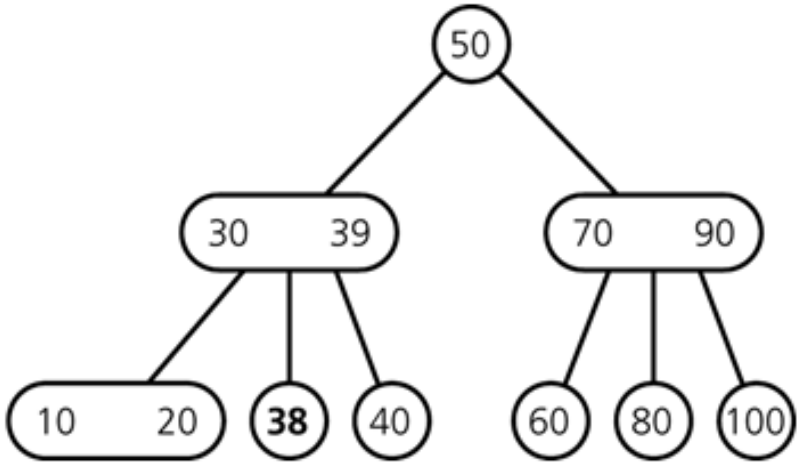
up the middle key then split the child node



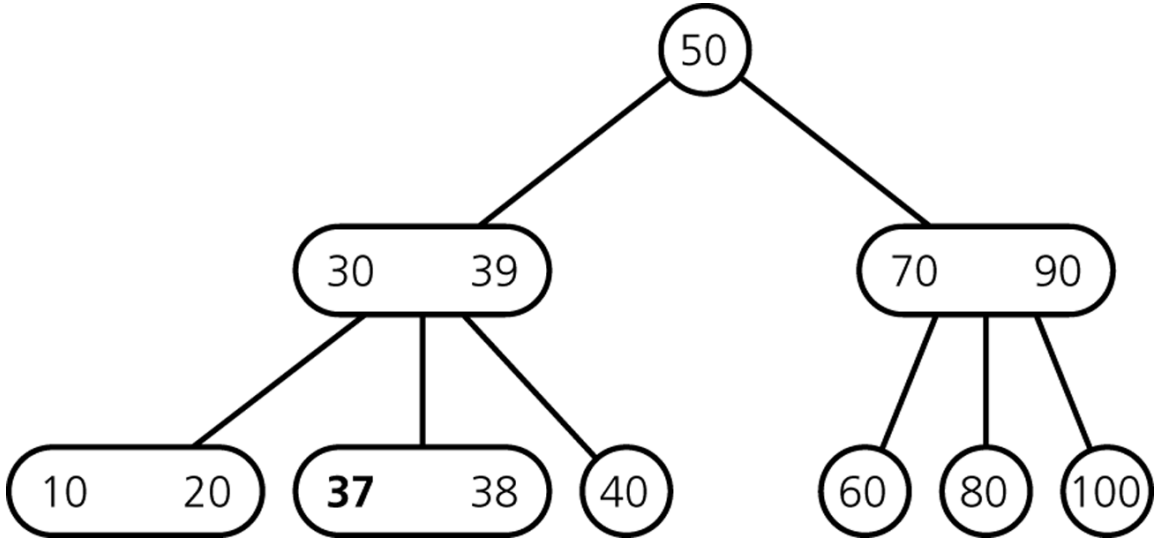


Insertion Examples

Insert 37

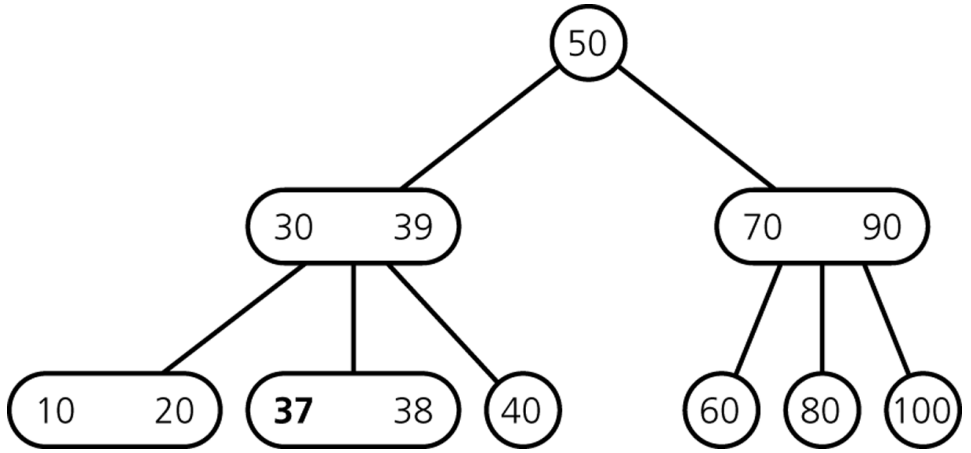


Case 1

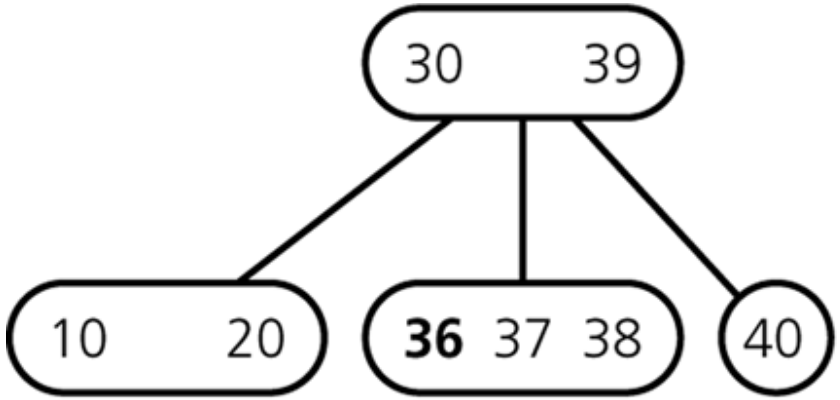


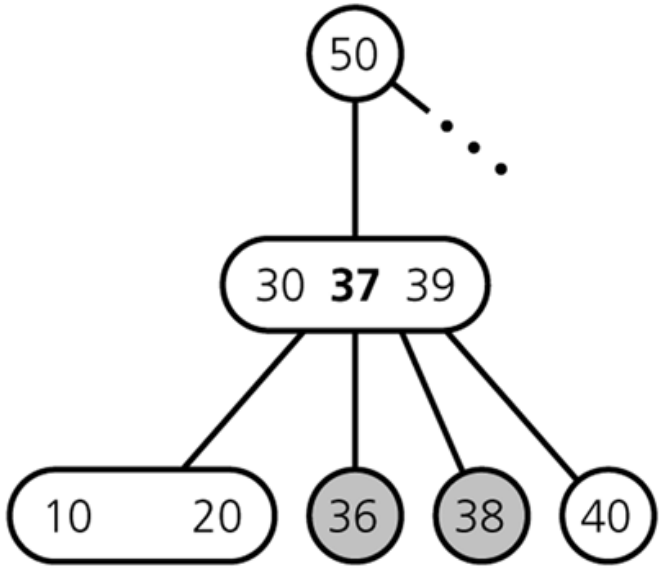
Insertion Examples

Insert 36

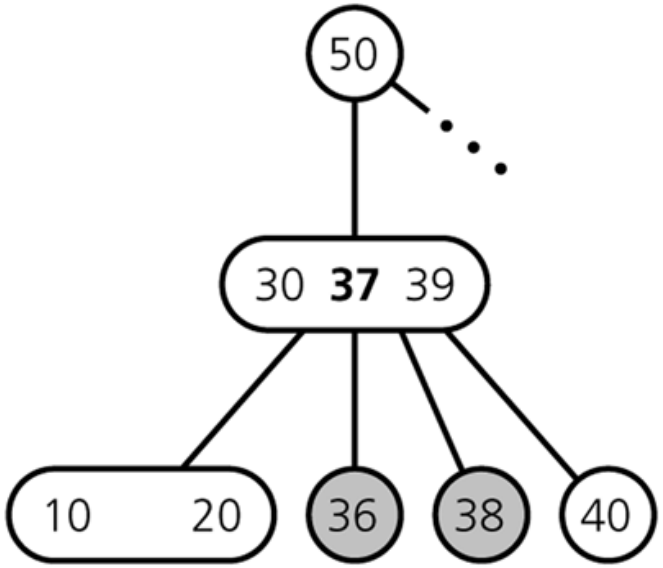


Move up the middle key then split the child node

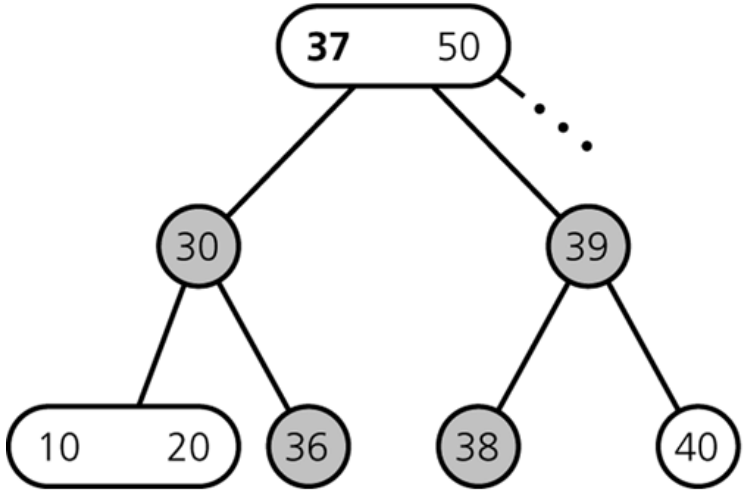




Insertion Examples

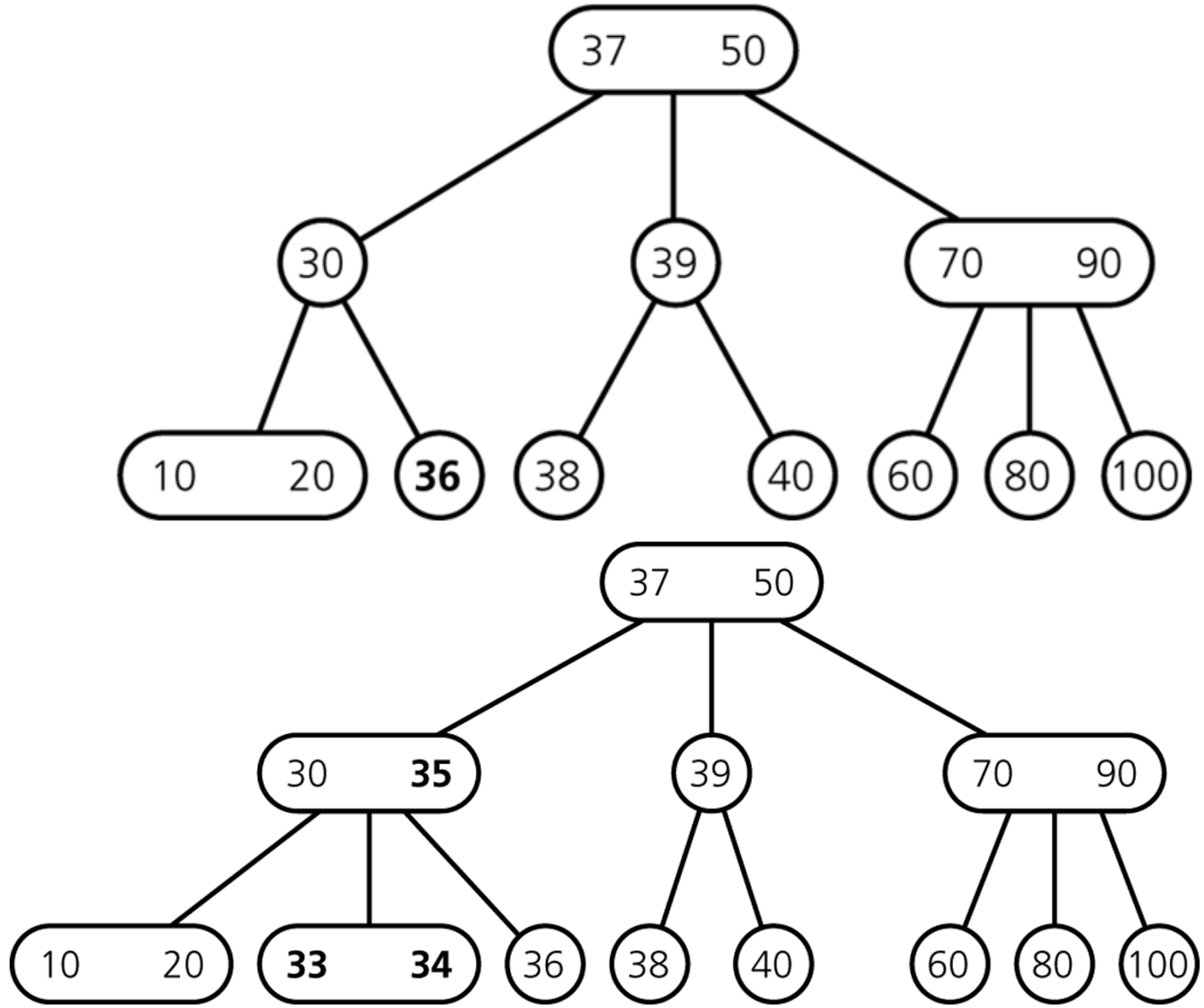


Move up the middle key then split the child node



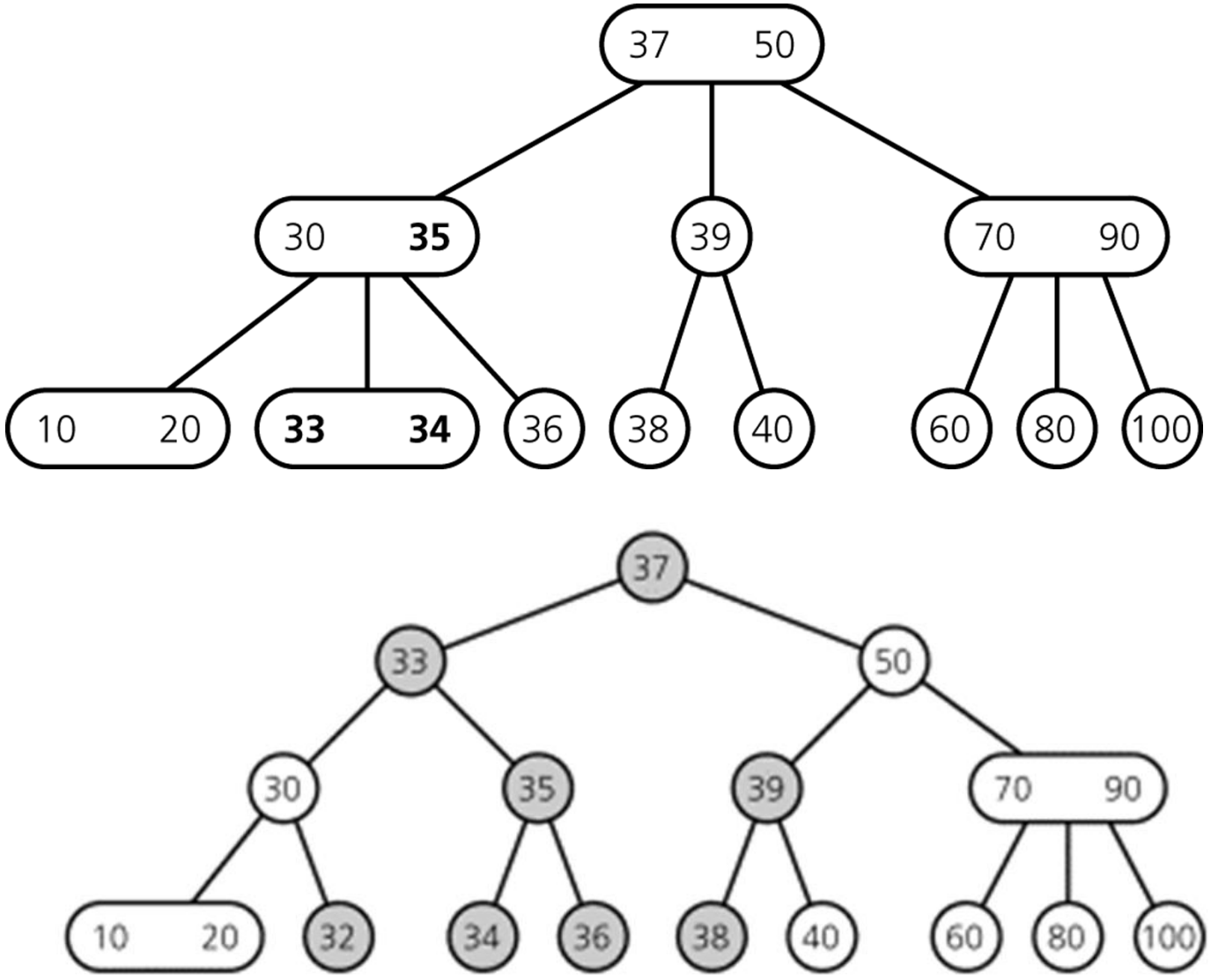
Insertion Examples

Insert 35, 34, 33



Insertion Examples

Insert 32



Deletion

To delete a value from the 2-3 tree:

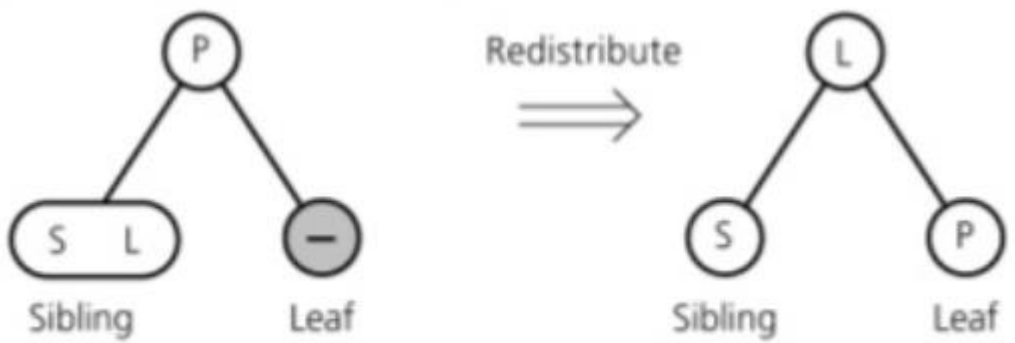
- If deleting a value from a node violates the property of a tree, that is, if a node is left with less than one data value then two nodes must be merged together to preserve the general properties of a 2-3 tree.
- In deletion, it is not necessary that the value has to be deleted from a leaf node.

Deletion

- If not in a leaf:
  - to delete a value x, it is replaced with its closest successor (which must lie in a leaf) and then delete the successor from the leaf.
- If in a leaf:
  - if the leaf is still non-empty, nothing else needs to be done
  - if the leaf is now empty, the parent has an empty child, which is not allowed, and deleting the leaf would leave the parent with too many values
    - if the left or right sibling has two values, we can borrow and redistribute values
    - if not, we merge the leaf and its sibling (if any), demoting a value from the parent

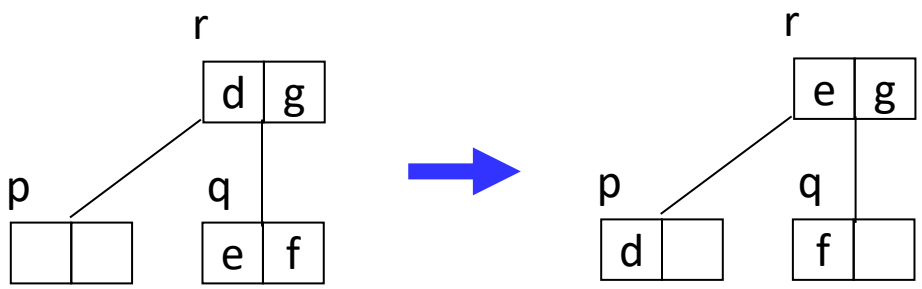
Deletion

If the left or right sibling has two values, we can borrow and redistribute values

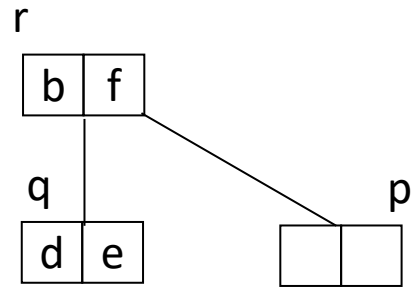


Apply process recursively

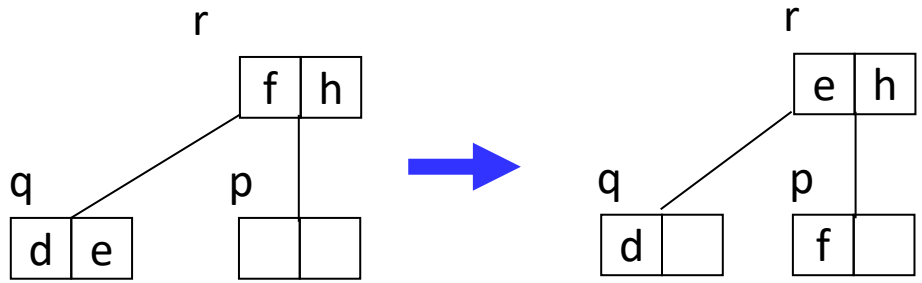
Three Cases



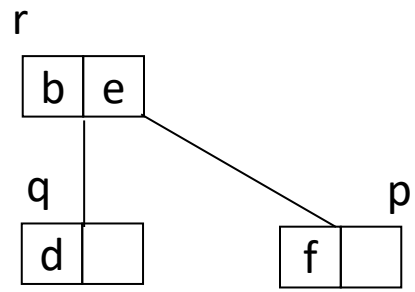
p is the left child of r



p is the right child of r

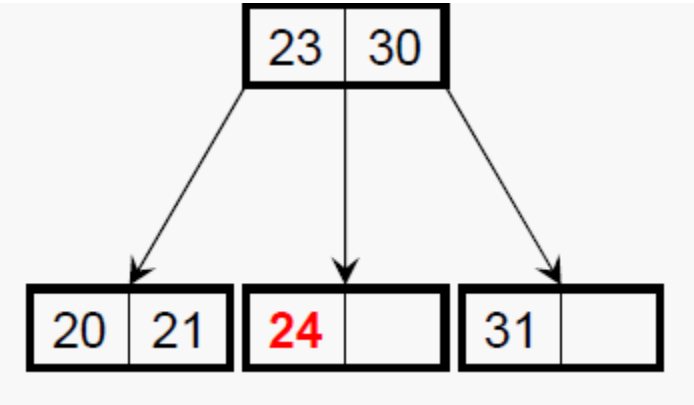


p is the middle child of r

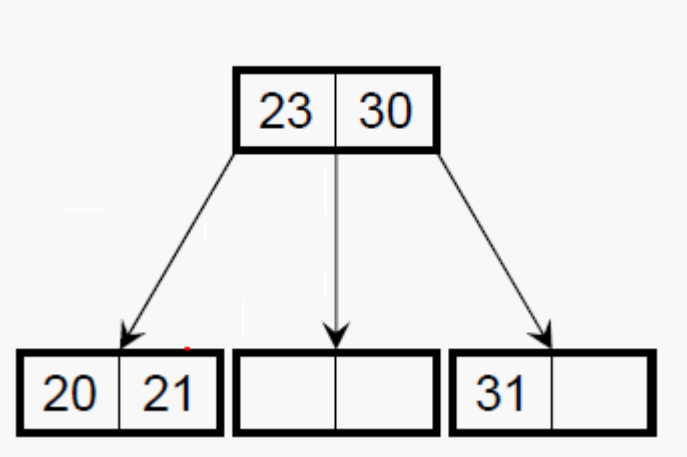


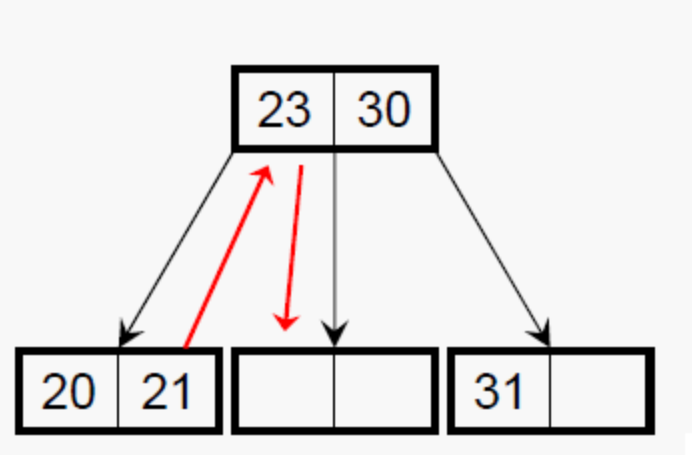
Deletion Example

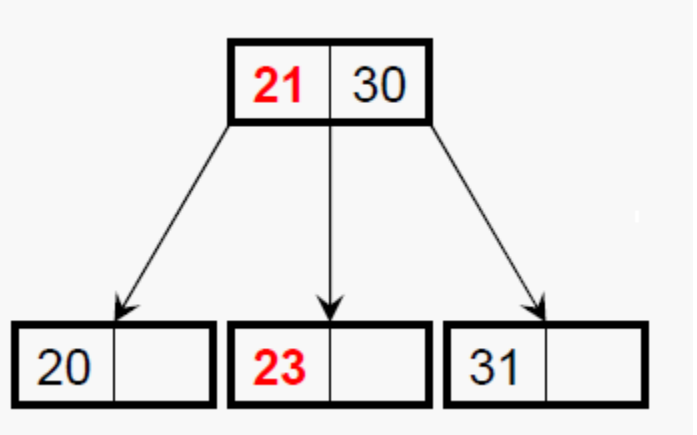
Delete 24



Demoting and borrowing

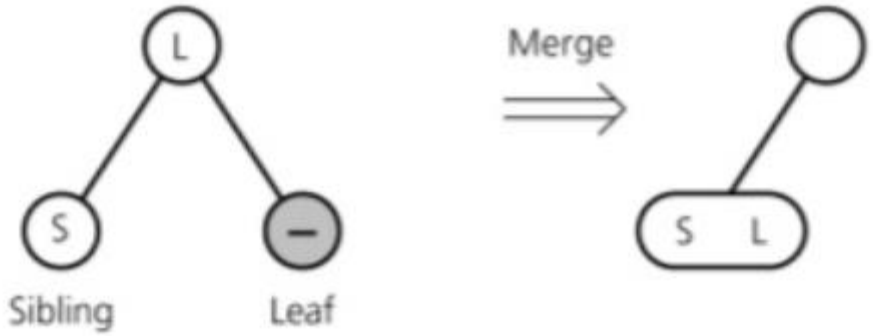






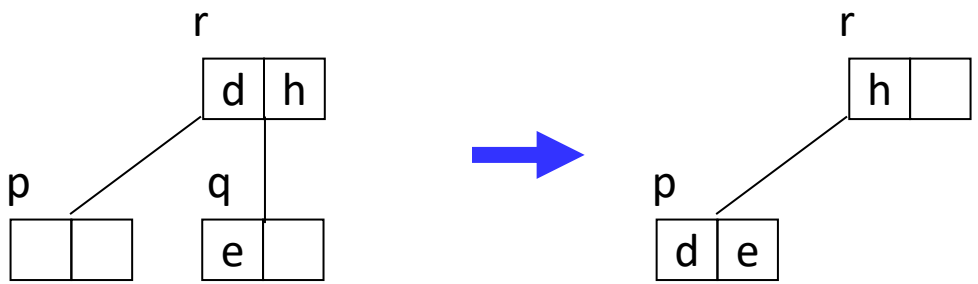
Deletion

If the left and right sibling has less than two values, we merge the leaf and its sibling (if any), demoting a value from the parent

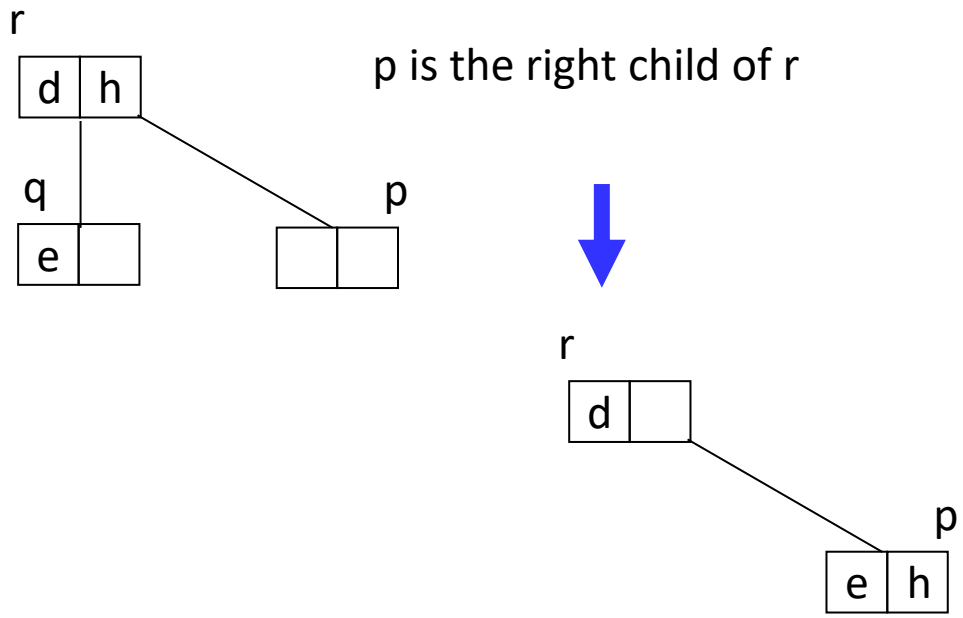


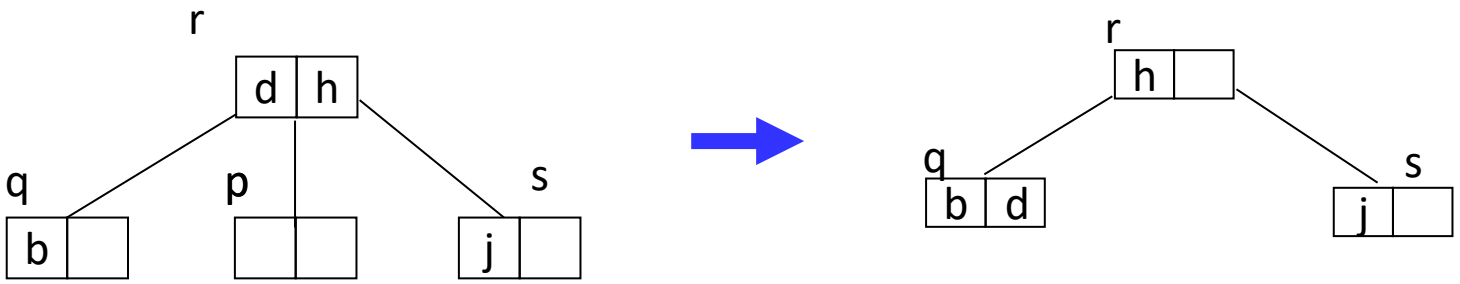
Apply process recursively

Three cases



p is the left child of r

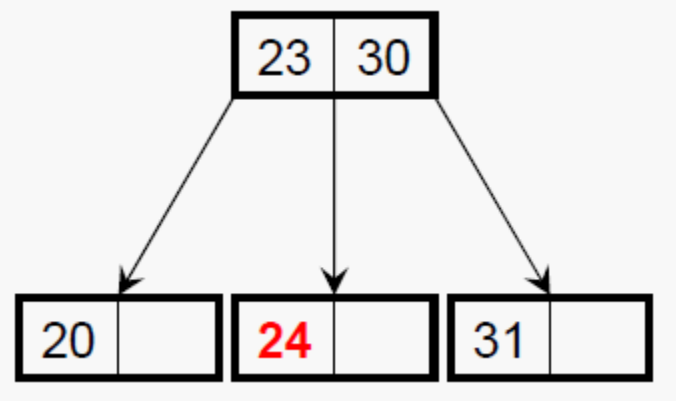


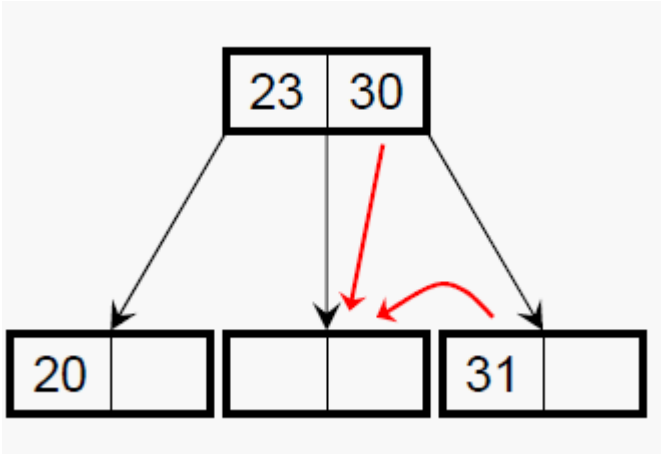


p is the middle child of r

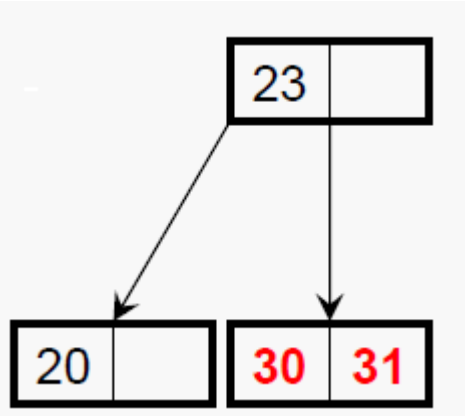
Deletion Example

Delete 24



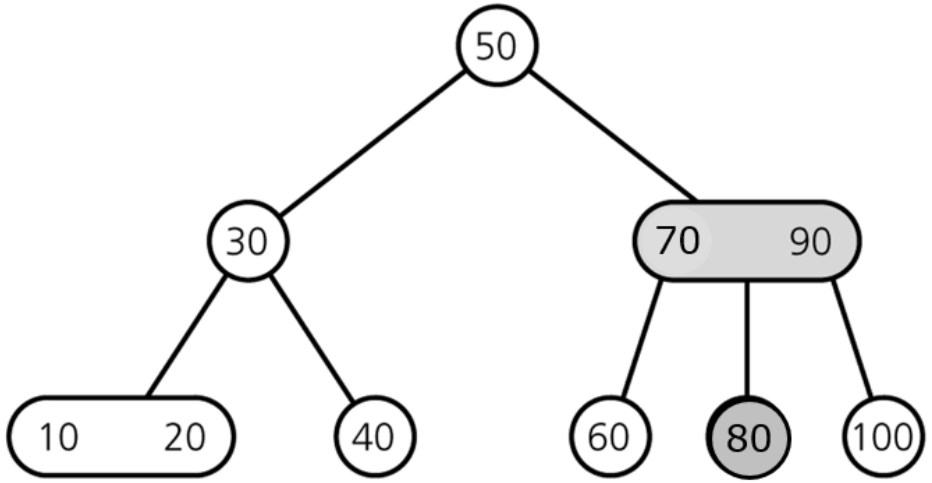


Demoting and borrowing
Delete an empty leaf

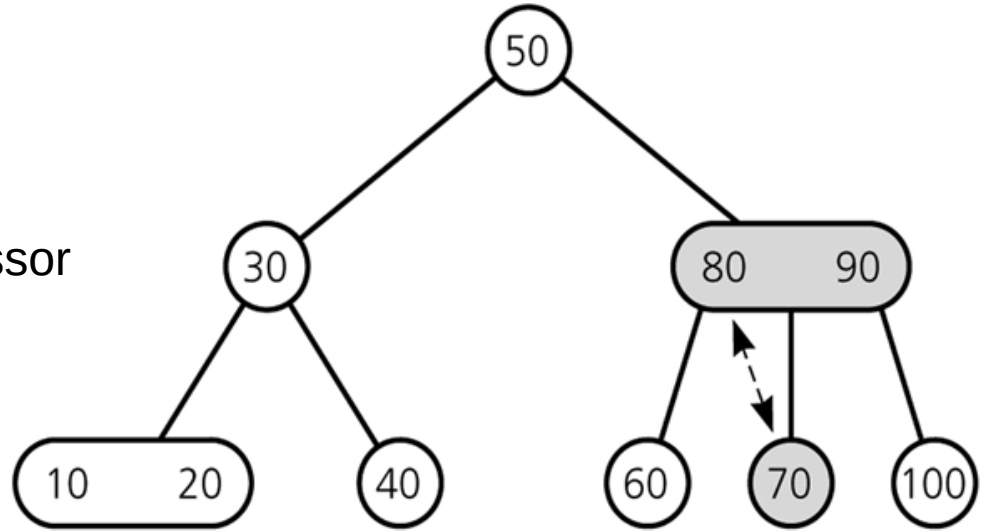


Deletion Example

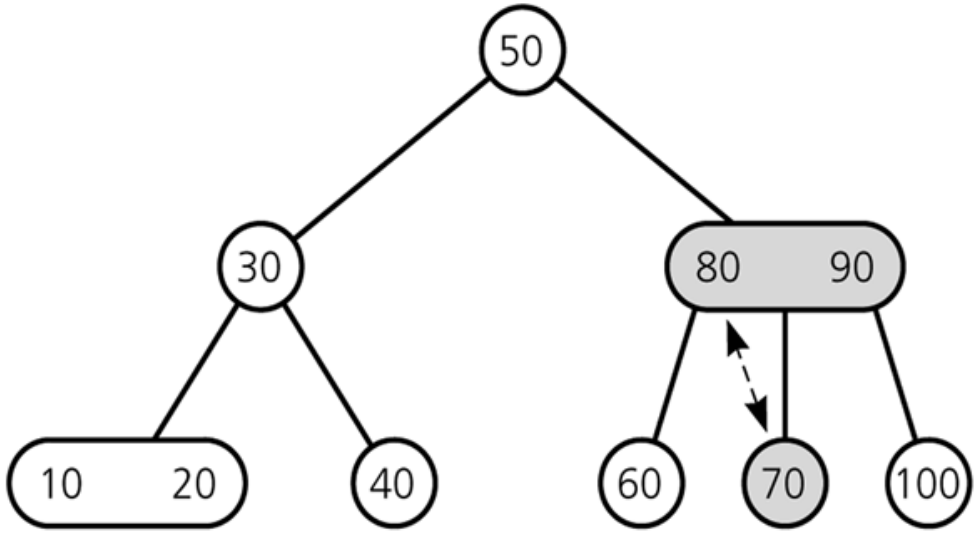
Delete 70



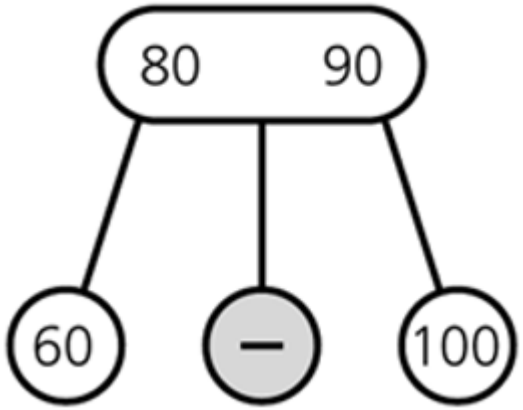
replace with its closest successor



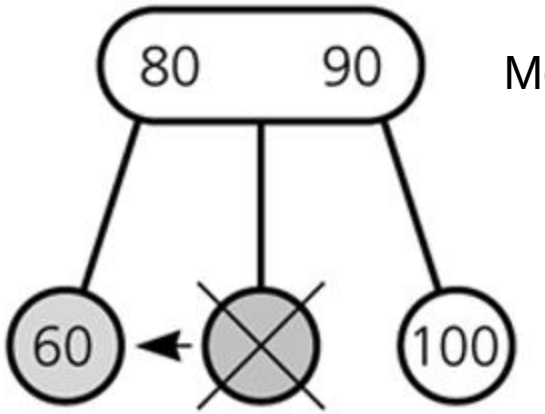
Deletion Example



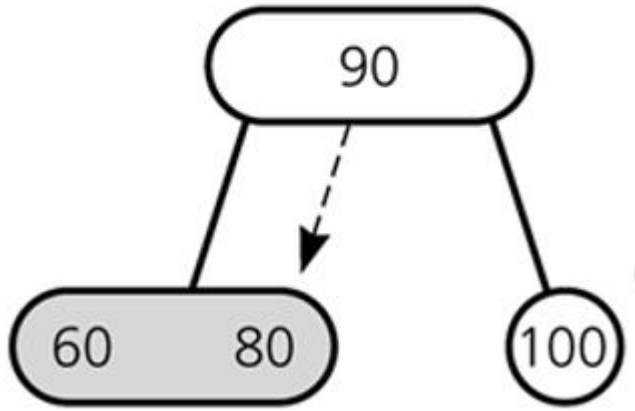
Delete value from leaf



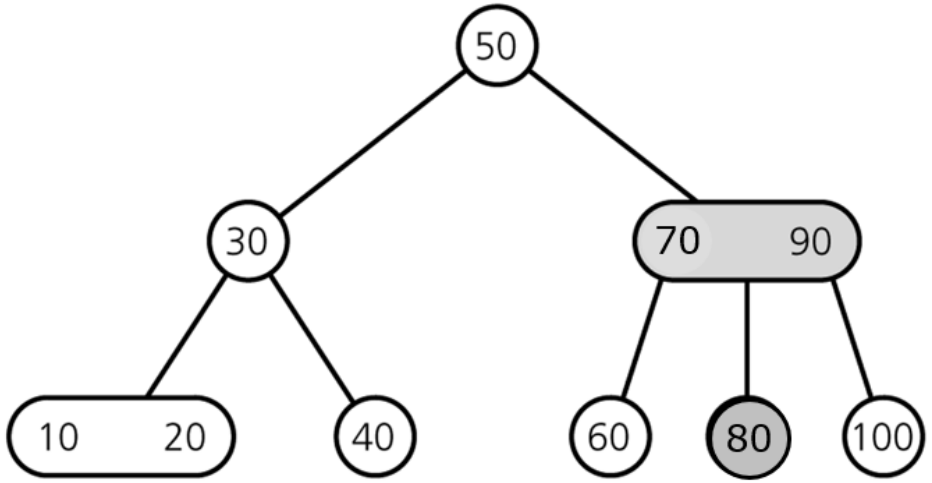
Delete empty leaf



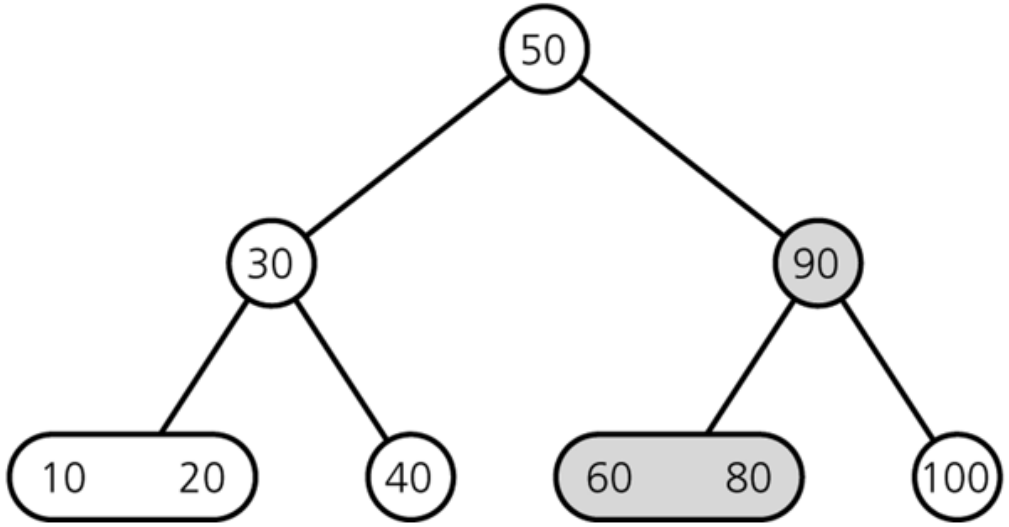
Move 80 from parent to the leaf node



Deletion Example

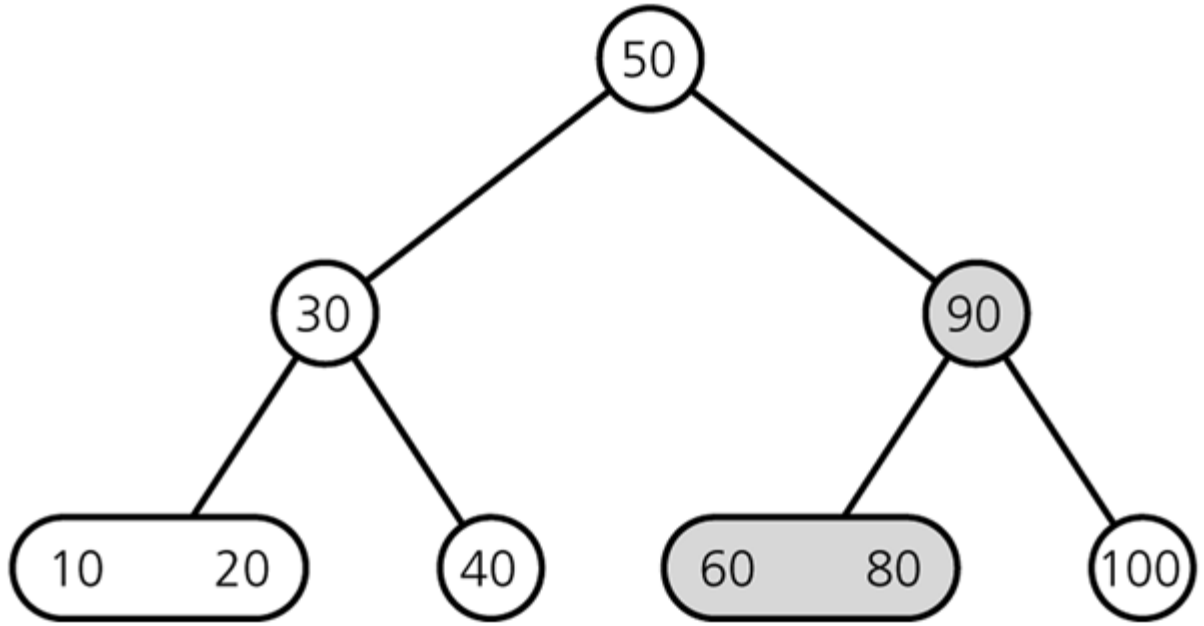


After deleting 70

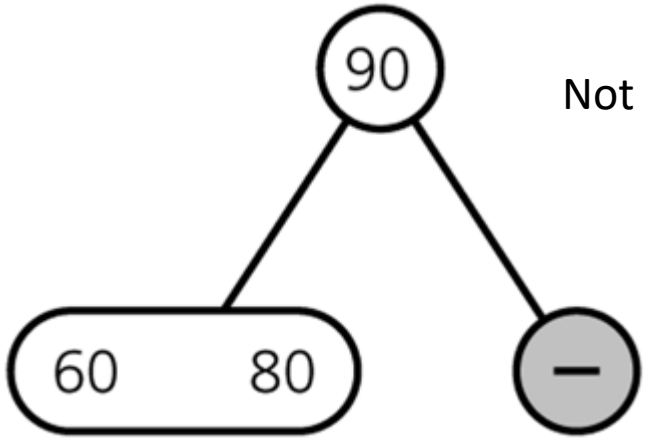


Deletion Example

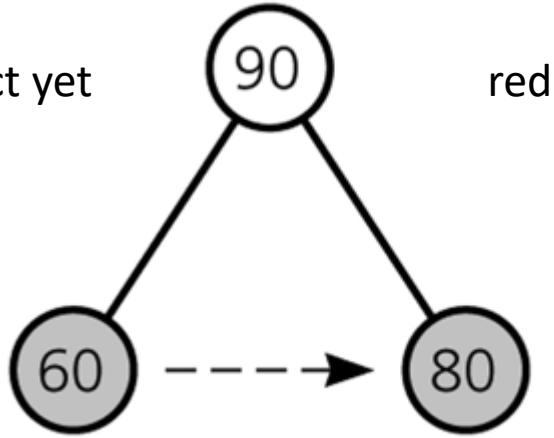
Delete 100



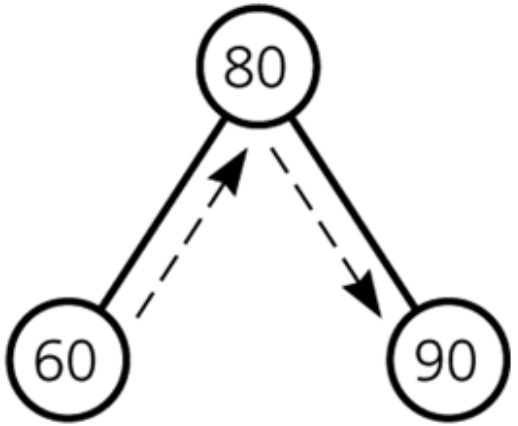
Delete from leaf



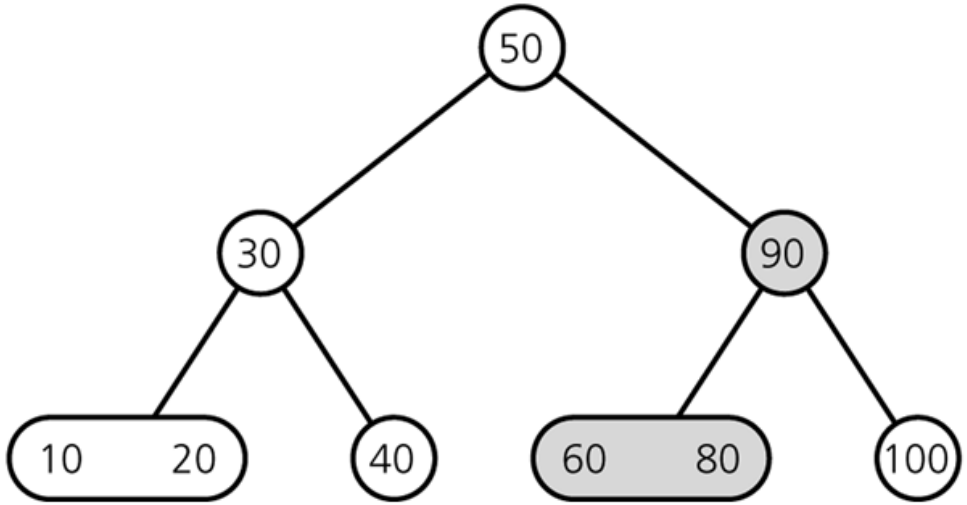
Not correct yet



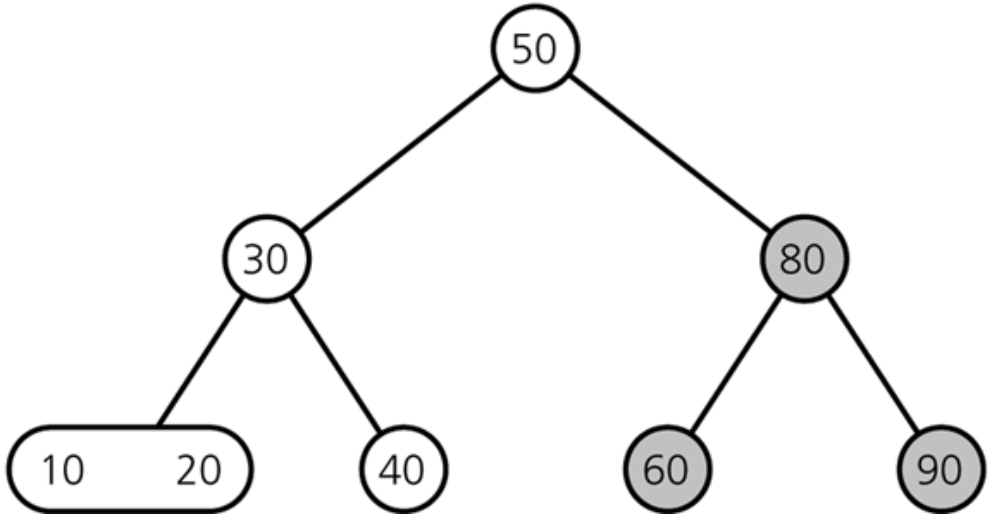
redistribute



Deletion Example

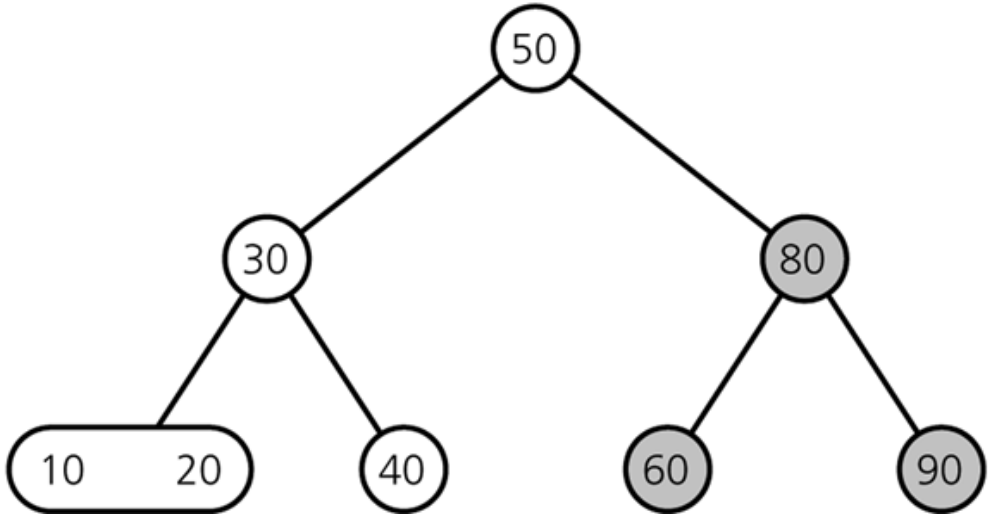


After deleting 100

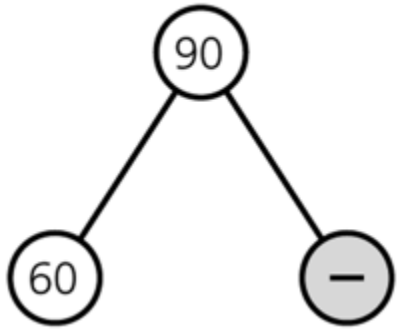


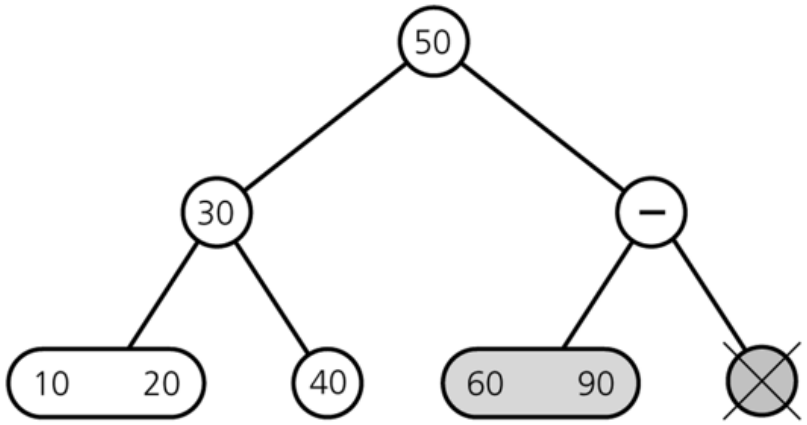
Deletion Example

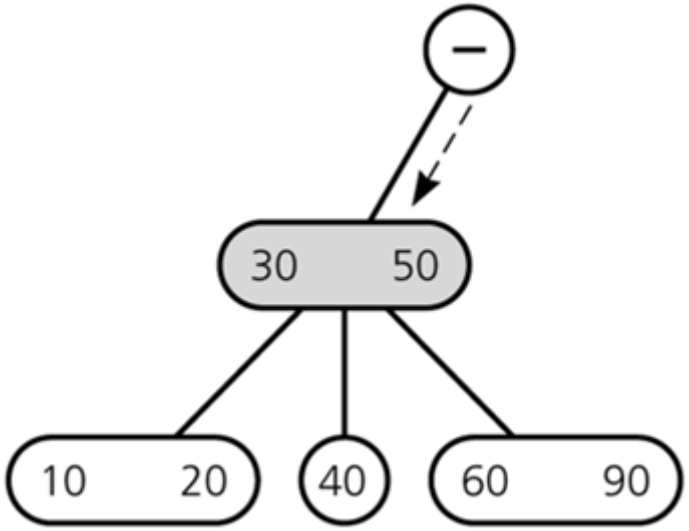
Delete 80

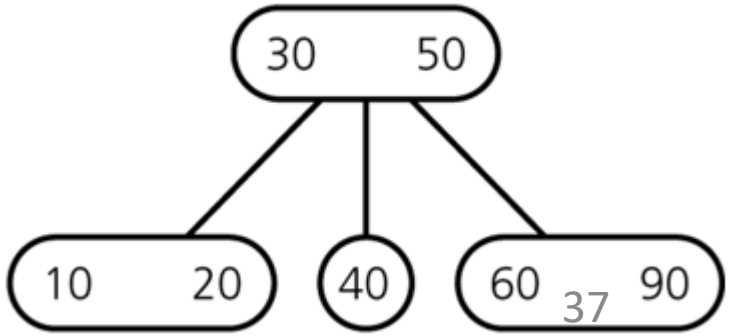


Delete from leaf





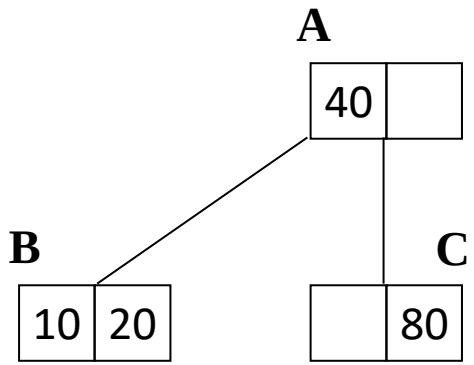


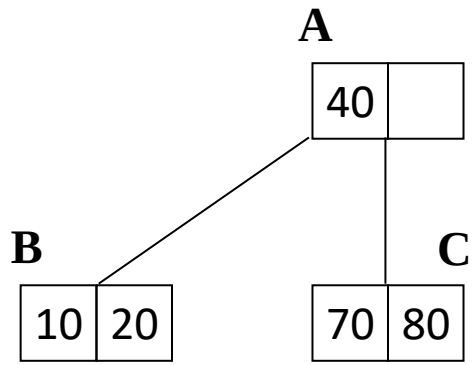


More Examples

Insertion to A 2-3 Tree Example

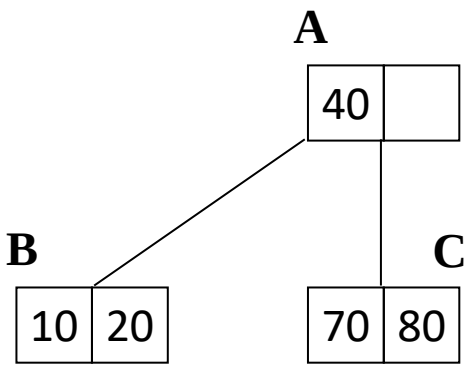
Insert 70

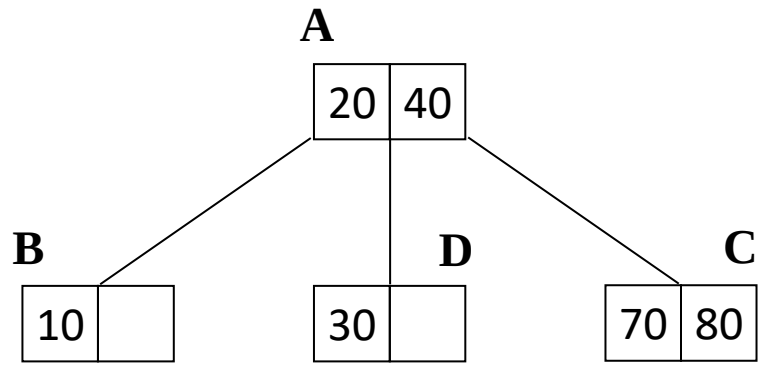




Insertion to A 2-3 Tree Example (Cont.)

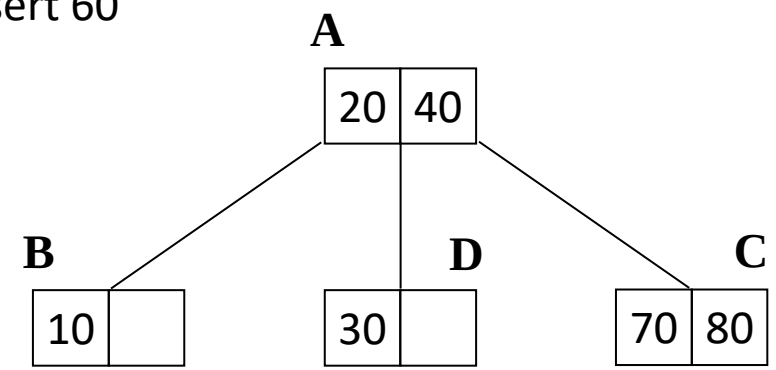
Insert 30

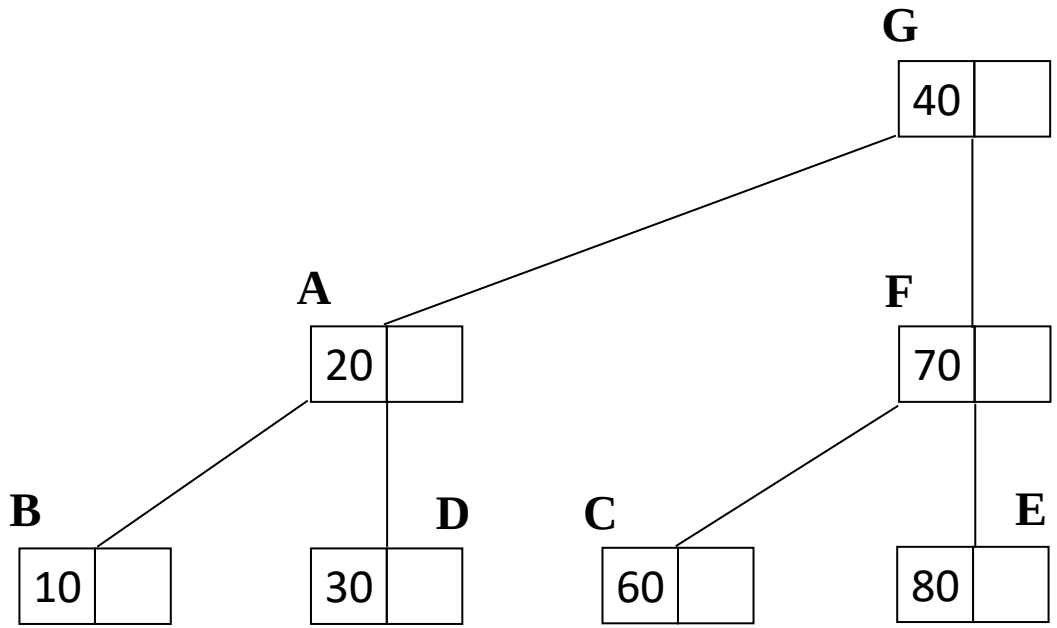




Insertion to A 2-3 Tree Example (Cont.)

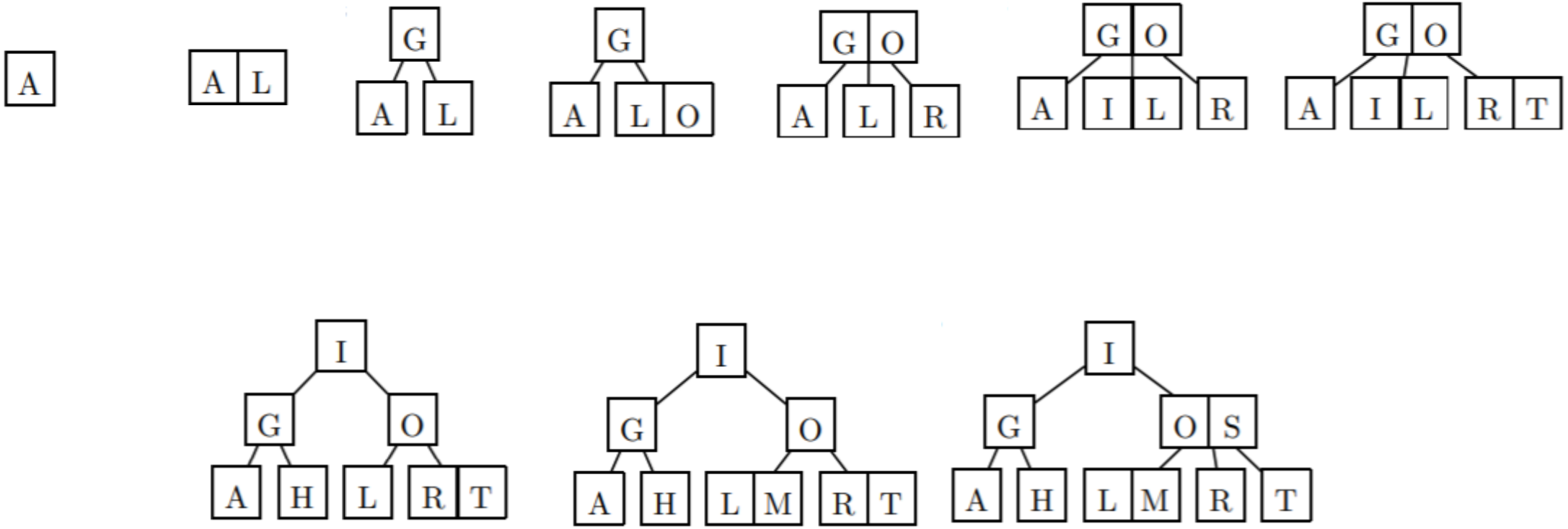
Insert 60





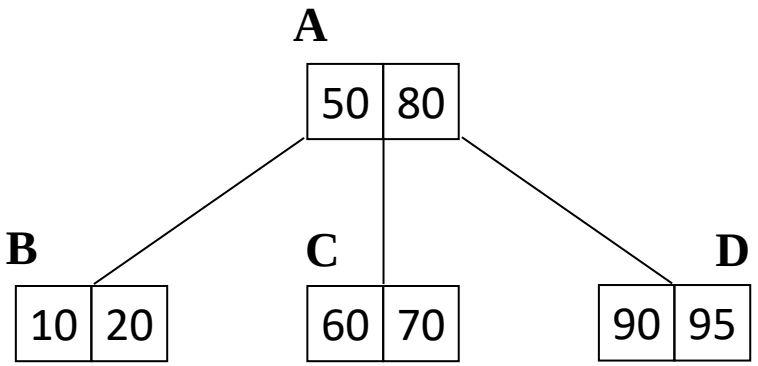
Insertion to A 2-3 Tree Example (Cont.)

Insert A L G O R I T H M S

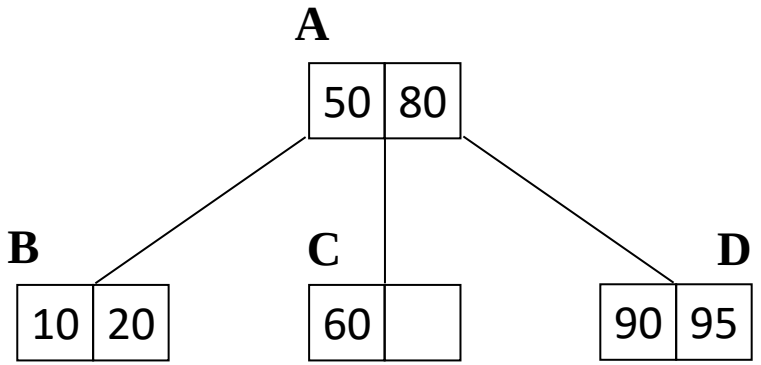


Deletion From A 2-3Tree Example

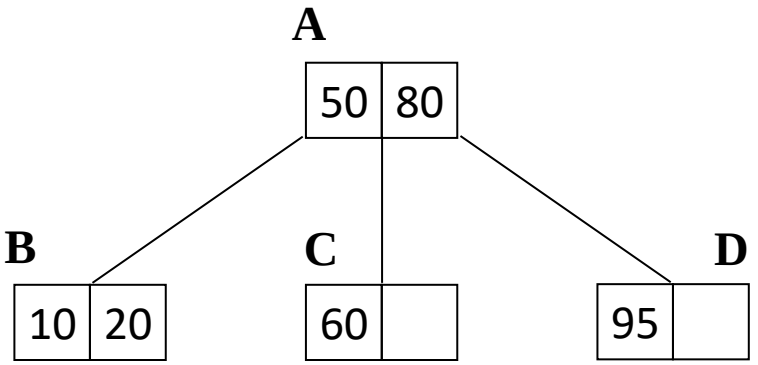
Initial 2-3 tree



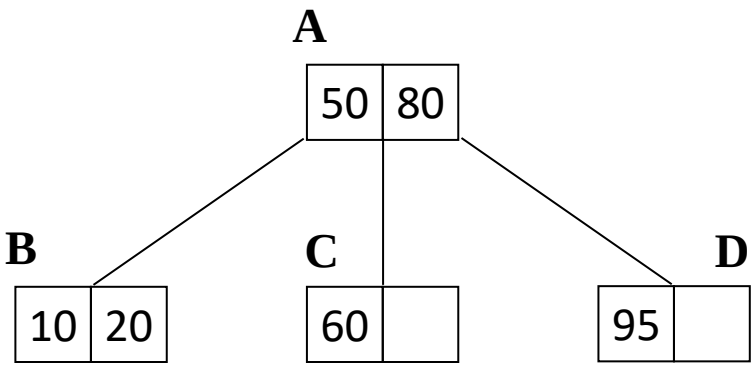
Delete 70



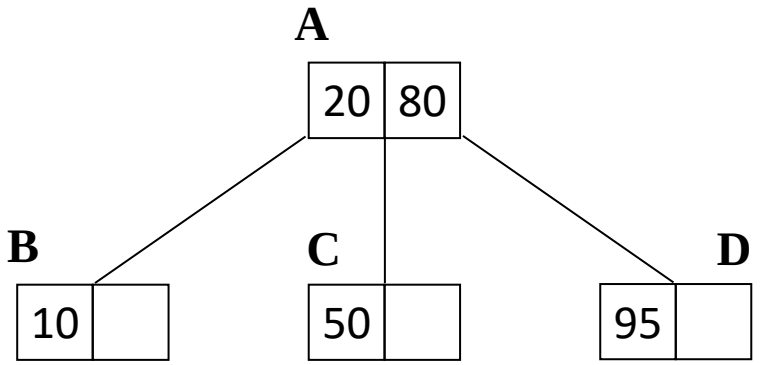
Delete 90



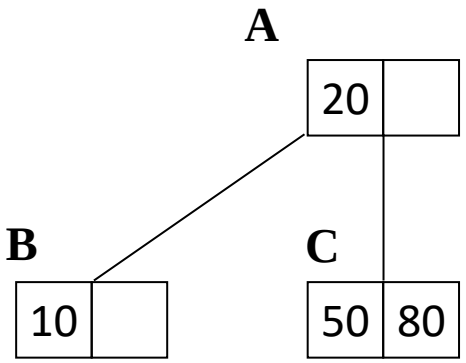
Deletion From A 2-3Tree Example (Cont.)



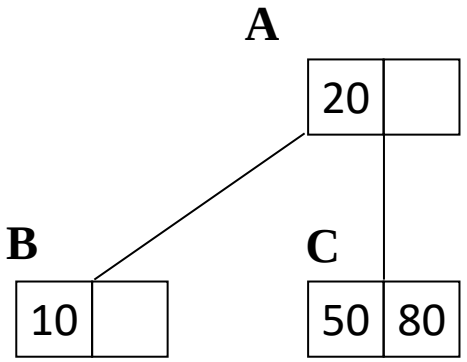
Delete 60



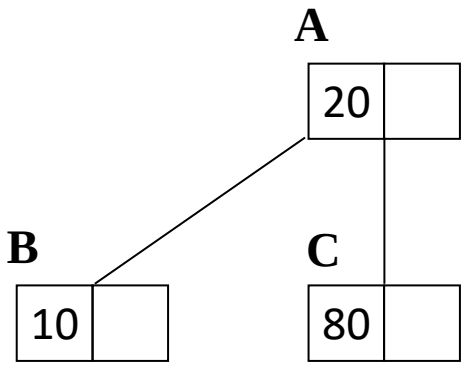
Delete 95



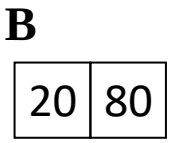
Deletion From A 2-3Tree Example (Cont.)



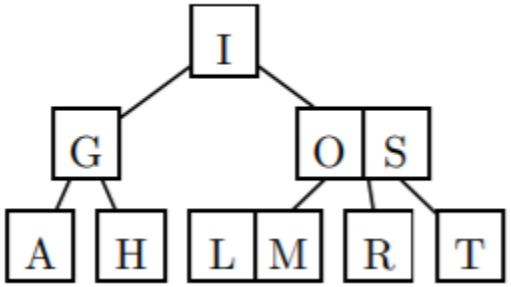
Delete 50



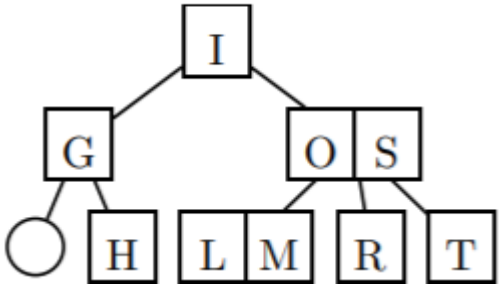
Delete 10

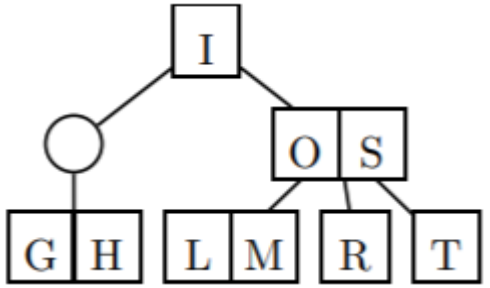


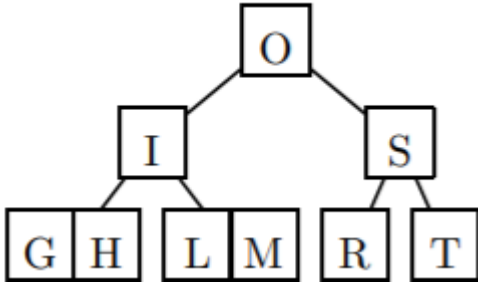
Deletion From A 2-3Tree Example (Cont.)

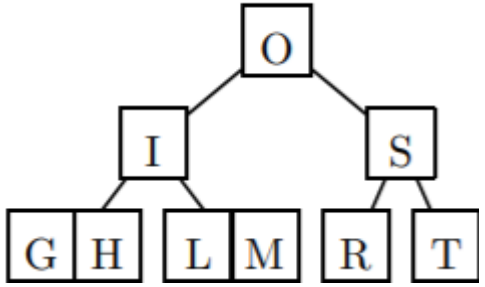


Delete A

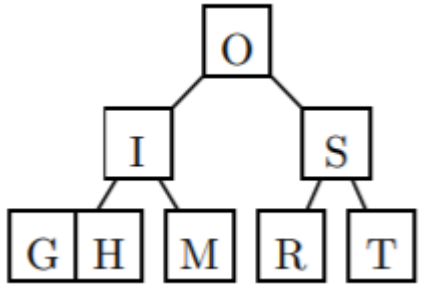




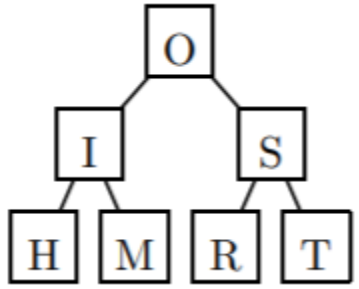




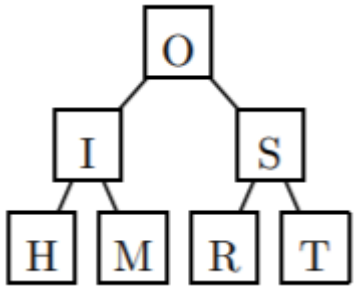
Delete L



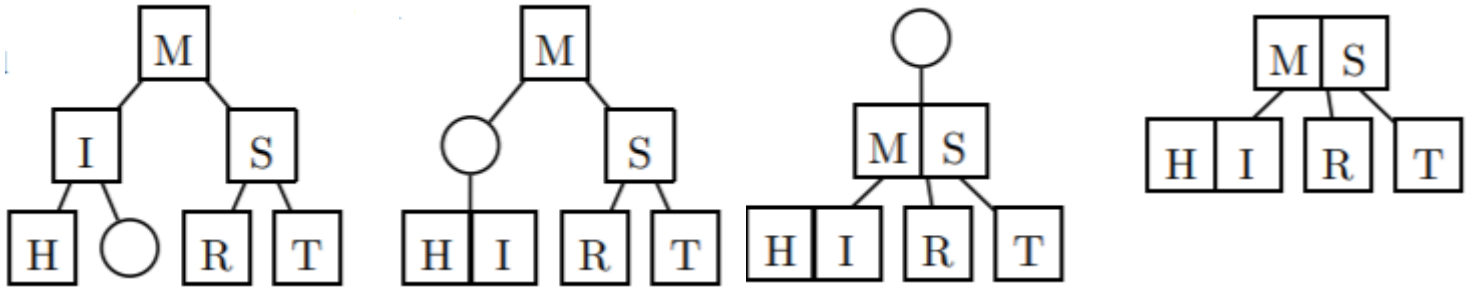
Delete G

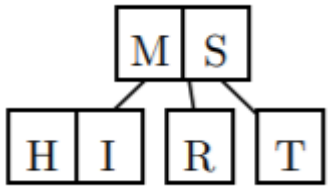


Deletion From A 2-3Tree Example (Cont.)

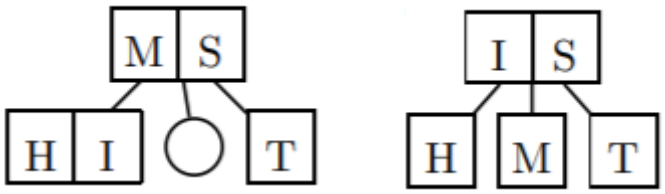


Delete O

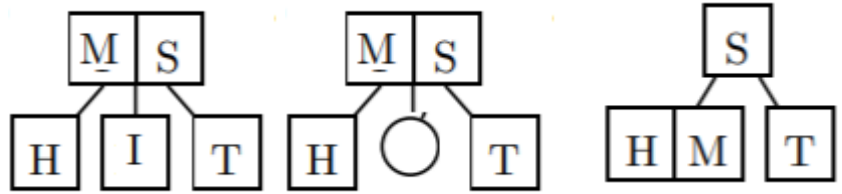


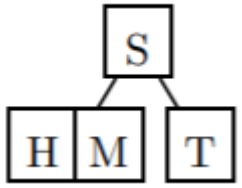


Delete R

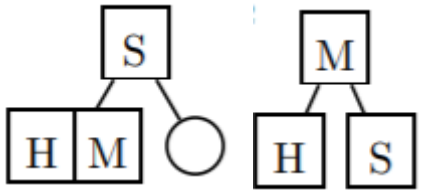


Delete I





Delete T



Delete H