Data Structures:
Height-Balanced Search Trees:
2-3 Tree

YoungWoon Cha
(Slide credits to Won Kim)

Spring 2022

2-3-Tree

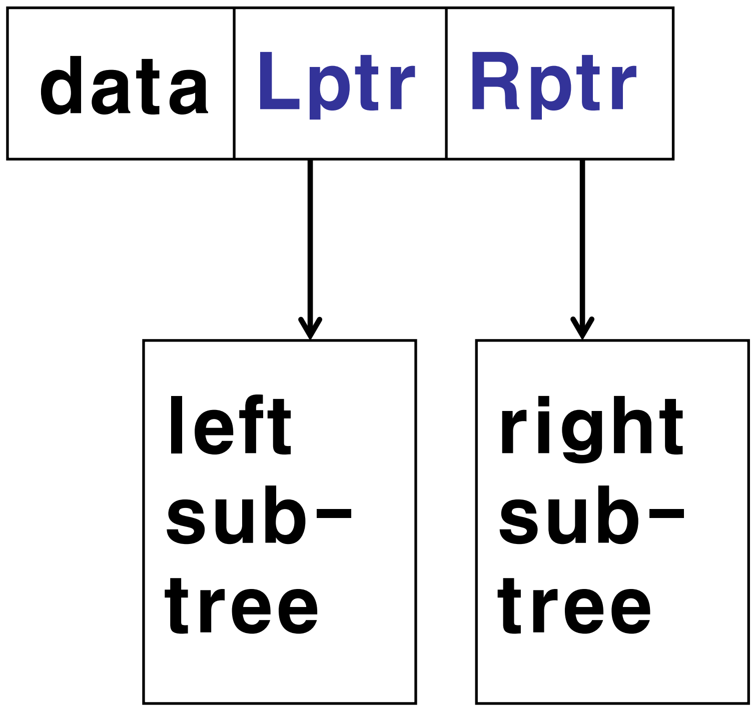
2-3 Tree

- A “Perfectly Balanced Tree”
  - All leaf nodes are on the same level
- Invented by J.E. Hopcroft in 1970.
- Not used much
- But, a special case of B Tree/B+ Tree, and base of T Tree
  - B Tree/B+ Tree is very important
  - T Tree is important

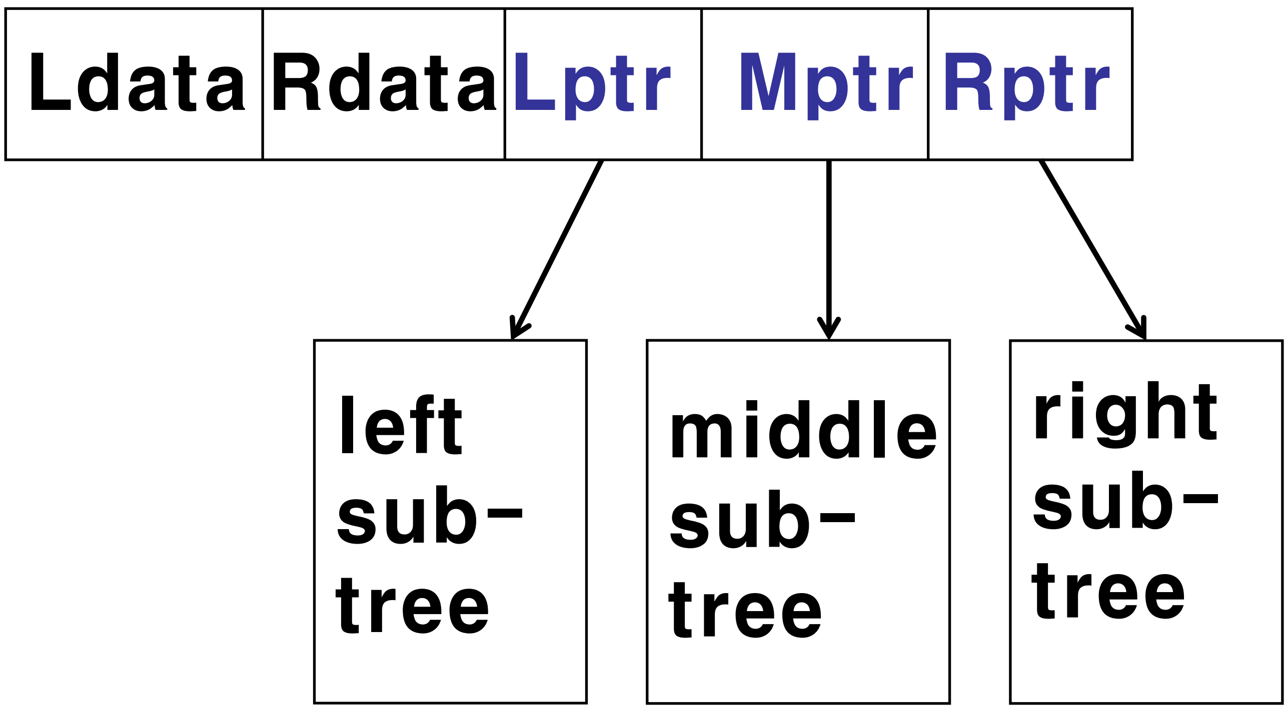
2-3 Tree

- Has Only 2-Nodes and 3-Nodes.
- smaller key to the left subtree, and larger key to the right subtree
- 2-node
  - with one key, and two child nodes (left, right)
  - root key of the left subtree < key
  - root key of the right subtree > key
- 3-node
  - with two keys (left, right), and three child nodes (left, middle, right)
  - root key of the left subtree < left key
  - root key of the middle subtree > left key AND < right key
  - root key of the right subtree > right key

2 Node (Implementation)



3 Node (Implementation)

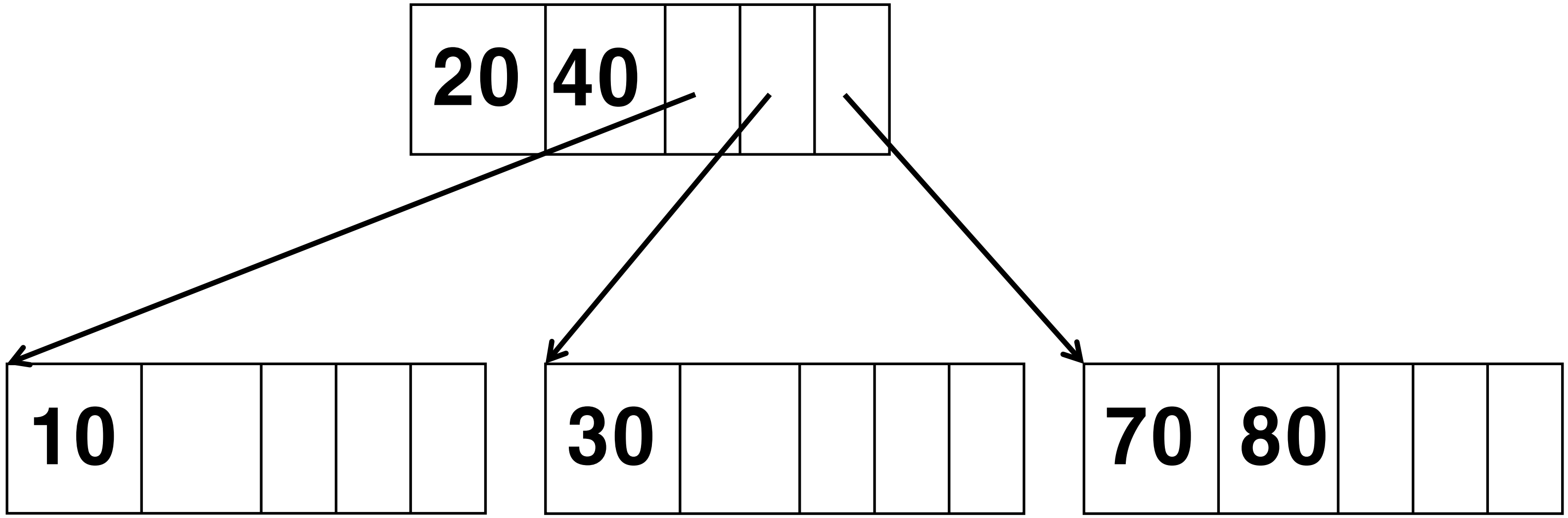


Searching a 2-3 Tree

- Search key X
- In a 2-Node
  - If X = the key of the node, search ends.
  - If X < the key of the node, search the left subtree.
  - If X > the key of the node, search the right subtree.
- In a 3-Node
  - If X = the left data or right data, search ends.
  - If X < the left data, search the left subtree.
  - If X > the left data and < the right data, search the middle subtree
  - If X > the right data, search the right subtree.
- If X is not found, search fails.

Searching a 2-3 Tree

Search for 80, 10, 25, 60



Height Balancing a 2-3 Tree

- Node Promotion and Node Demotion
  - node promotion: a 2-node becomes a 3-node
  - node demotion: a 3-node becomes a 2-node
- Data Re-Distribution
  - node split and node merge

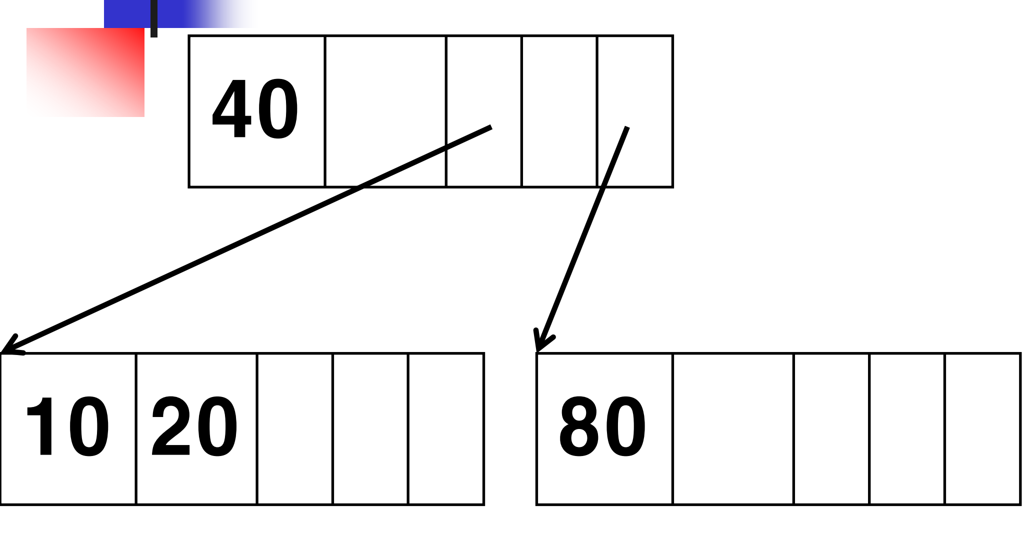
Insight on a 2-3 Tree

- A node has a minimum 1 data, and maximum 2 data.
  - maximum # of data = 2 x minimum # of data
  - overflow: 3rd data
  - underflow: 0 data
- Overflow and underflow require tree restructuring.
- Tree height increases by 1, only when all nodes are 3-nodes.

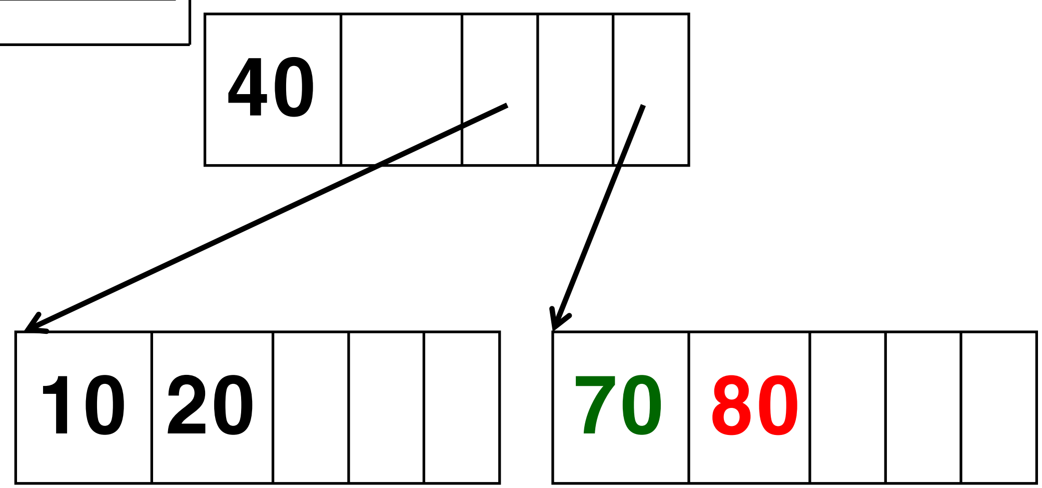
Inserting Data Into a 2-Node

- A 2-node becomes a 3-node.
- The smaller data becomes the “left” data.
- The larger data becomes the “right” data.
- Pointers (to the child nodes) in the node are adjusted.

Example



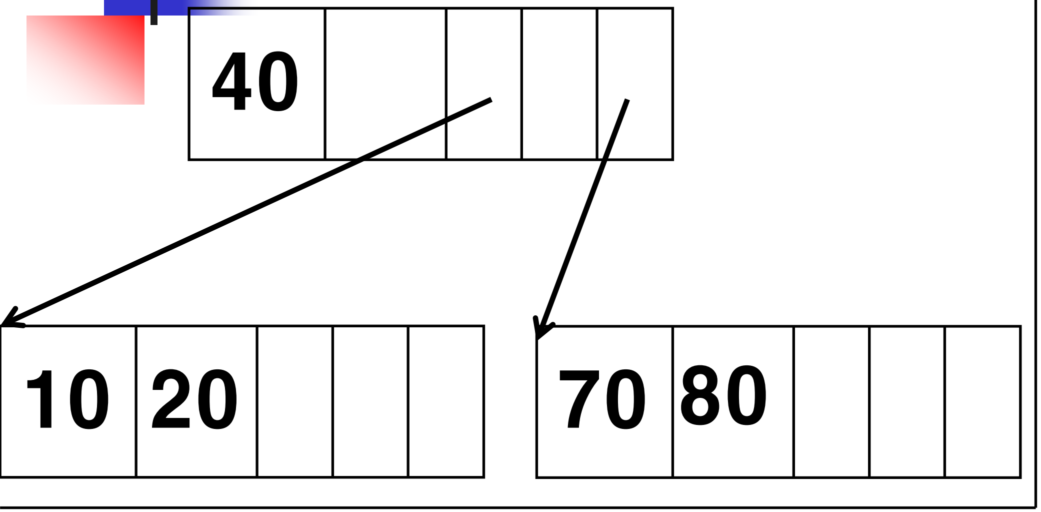
insert 70



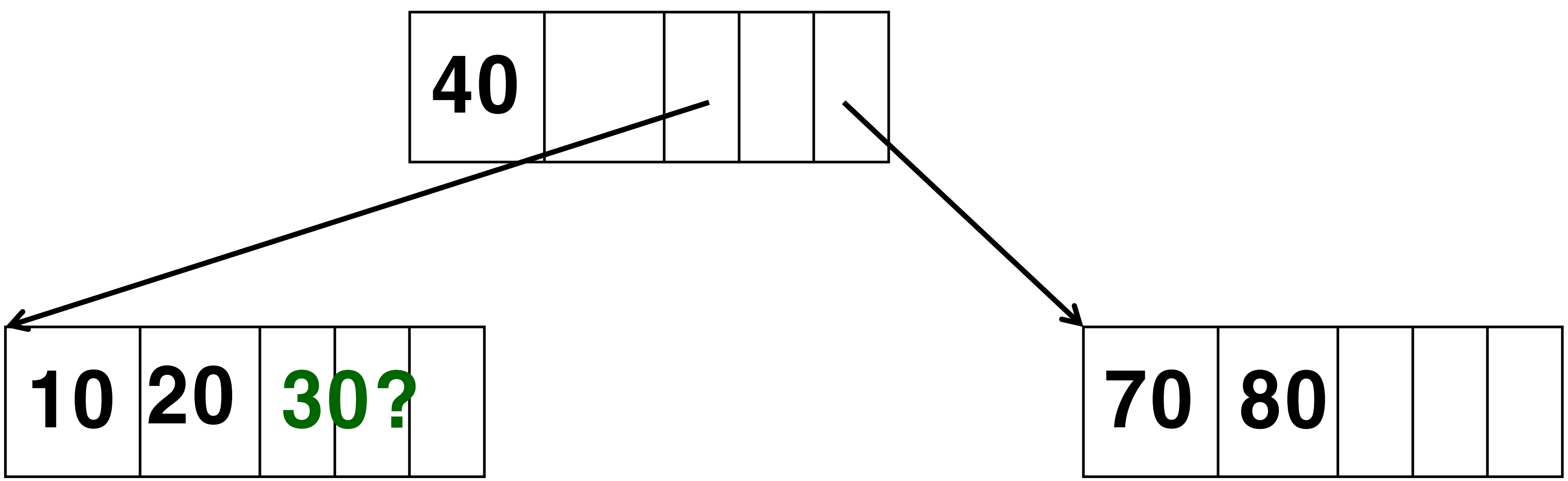
Inserting Data Into a 3-Node

- The 3-node splits into 2 separate 2-nodes (to reserve space for future inserts)
  - The “smallest” data goes to the left 2-node.
  - The “largest” data goes to the right 2-node.
  - The “middle” data goes to the parent node.
- The “middle” pointer in the parent node points to one of the two new 2-nodes.
- If the parent node is a 3-node, it is split, too, recursively.

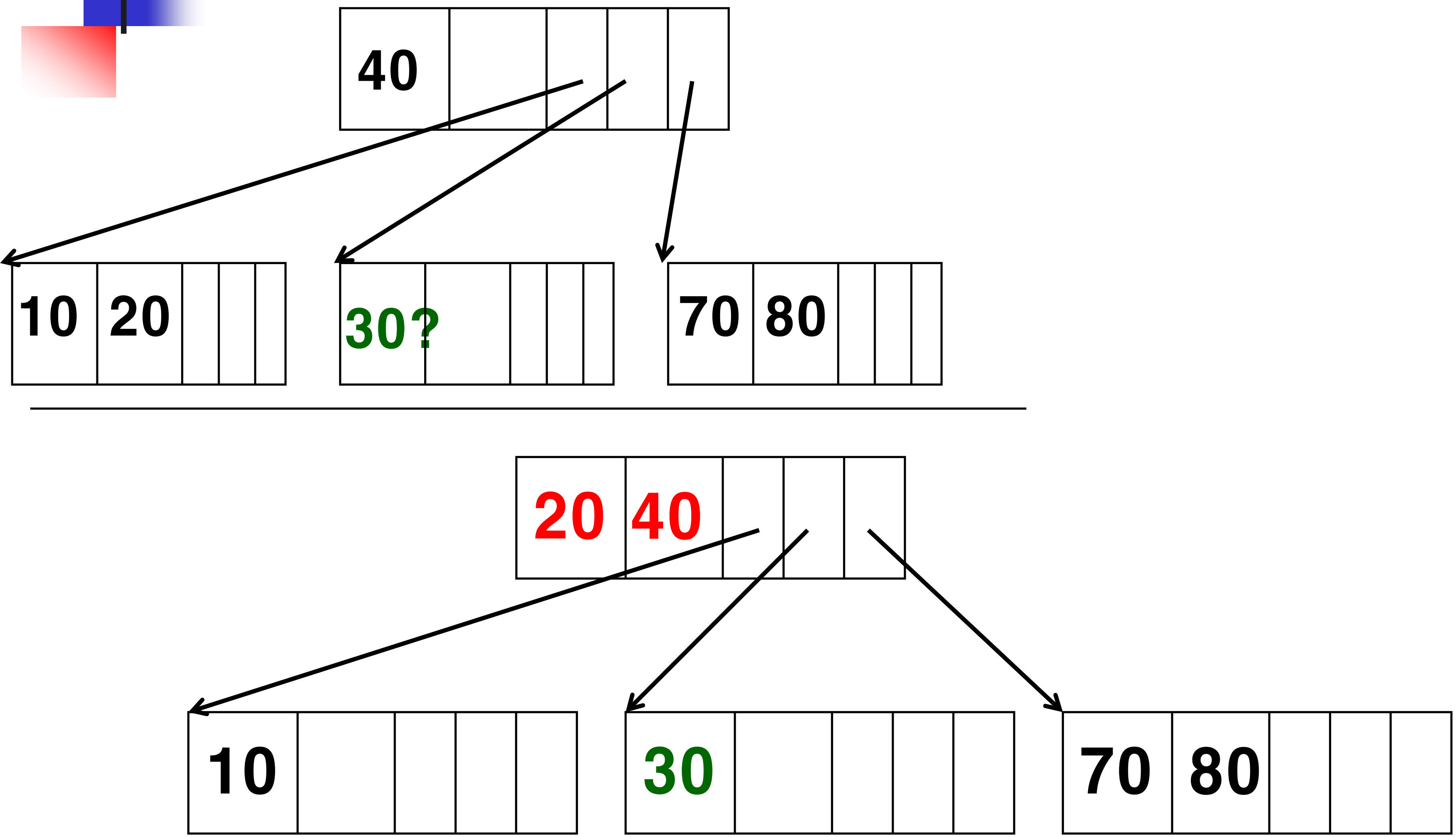
Example 1: (1/2)



insert 30

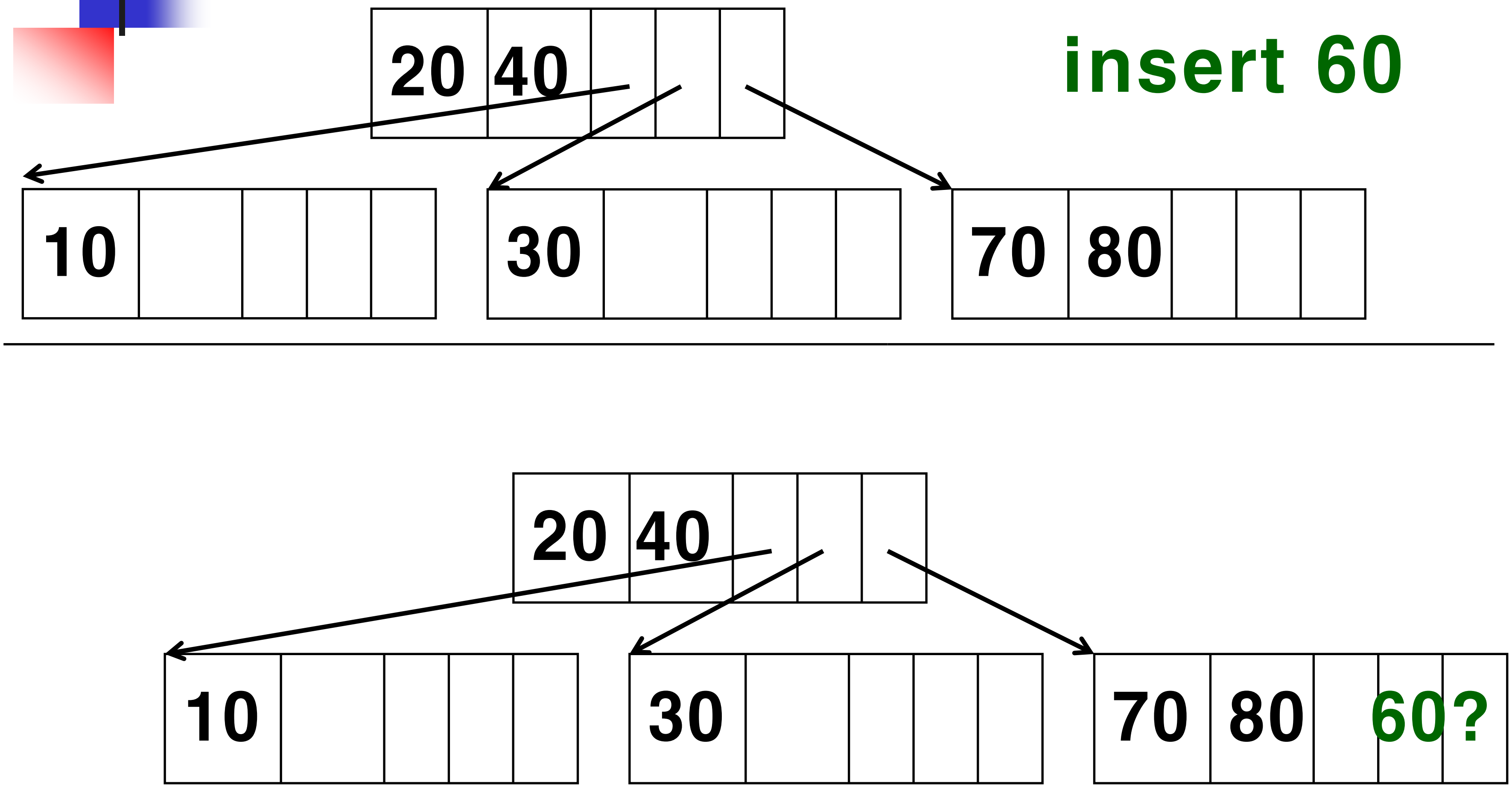


Example 1: (2/2)

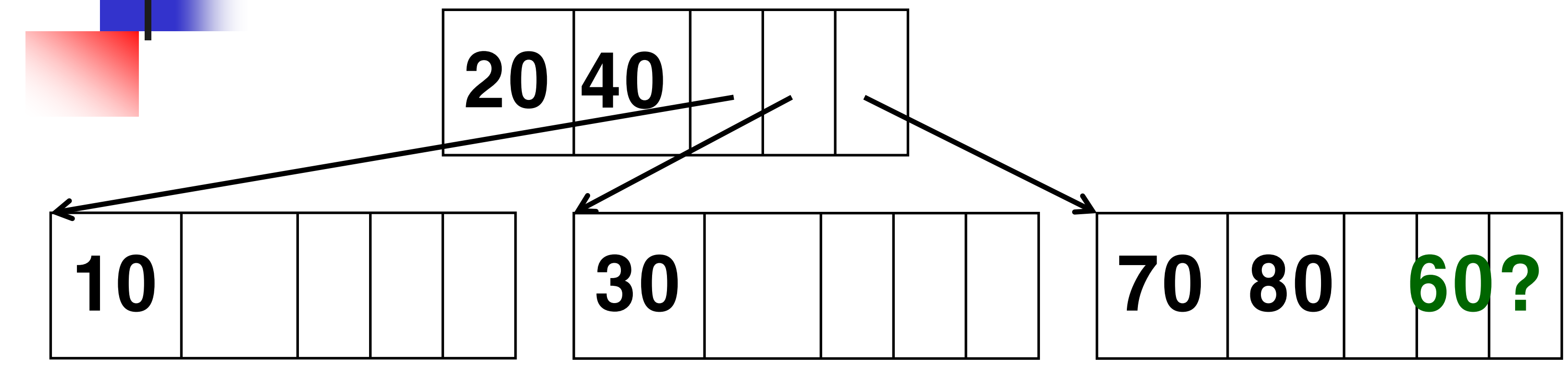


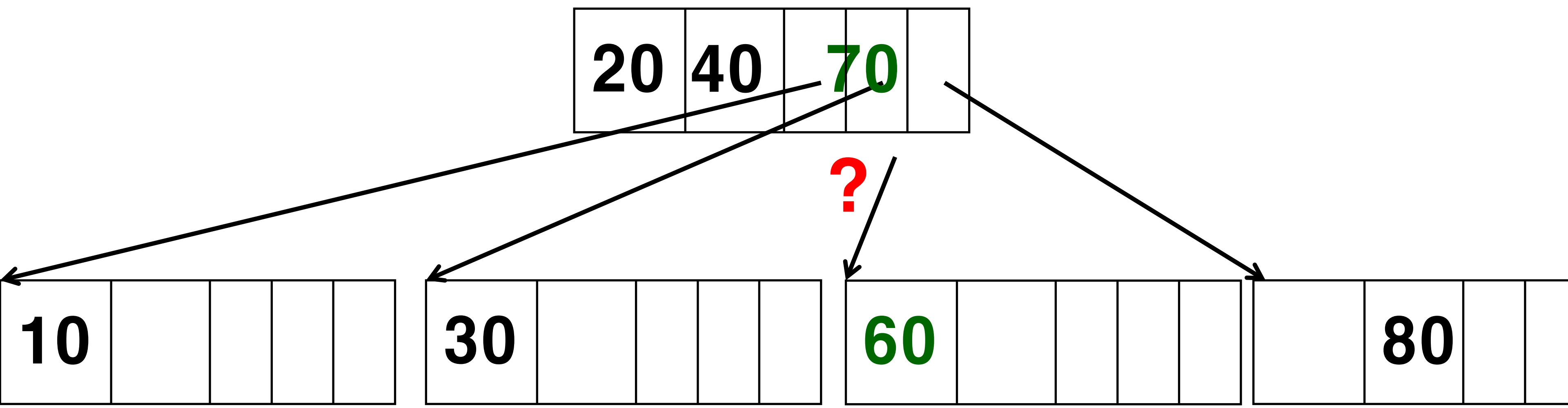
Example 2: (1/3)

insert 60

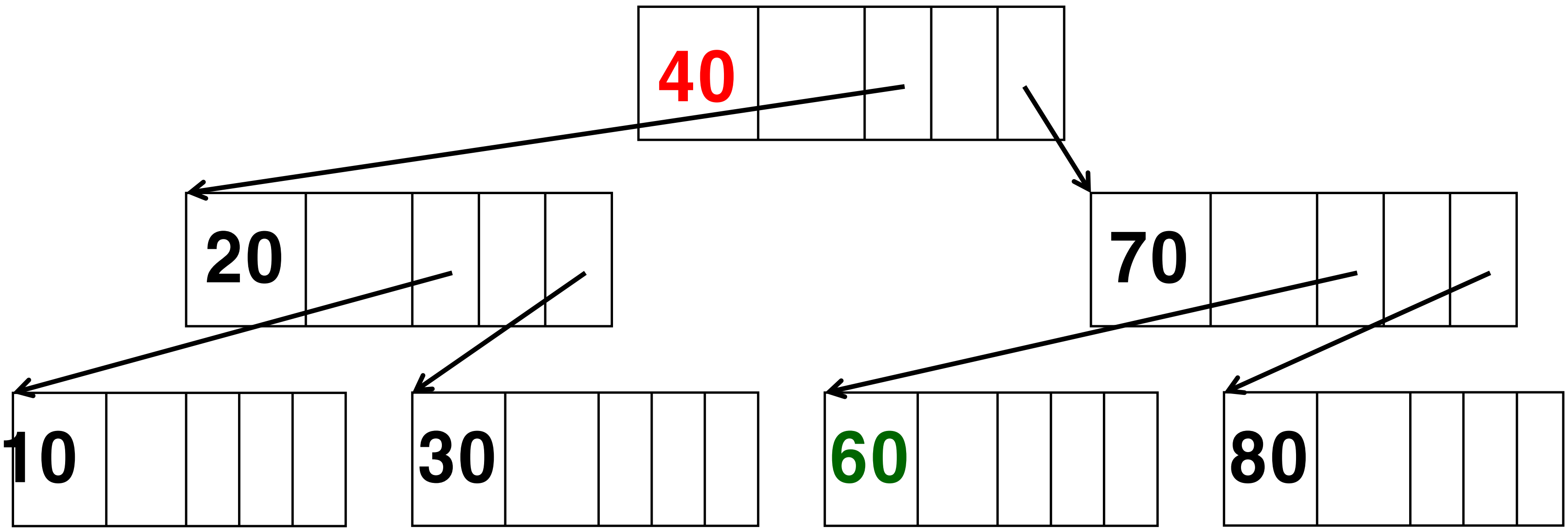


Example 2: (2/3)





Example 2: (3/3)



Example 3: (1/2)

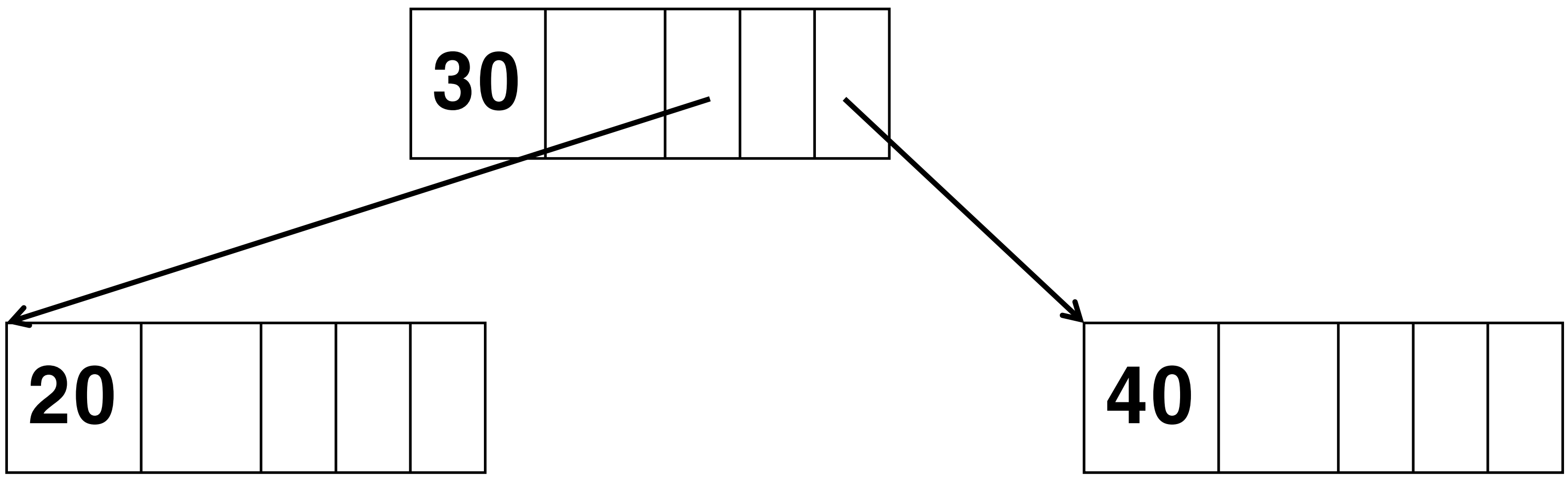
insert 30

20	40			

30?

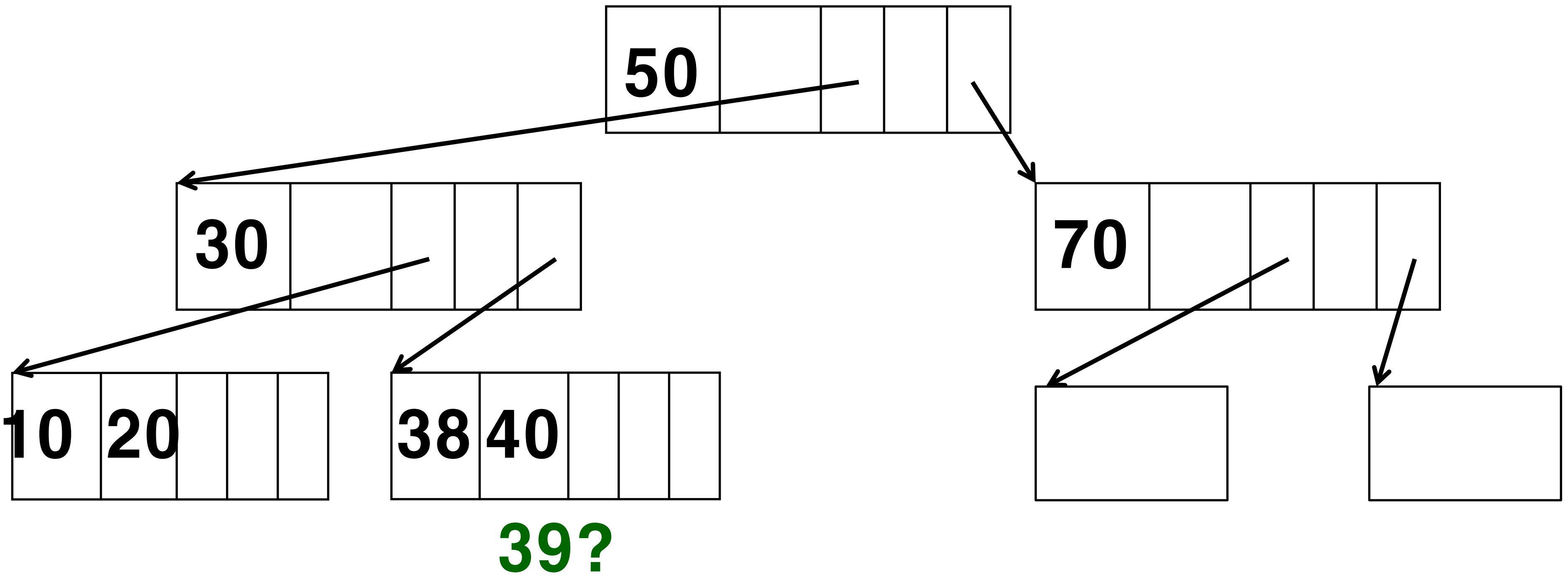
Example 3: (2/2)

insert 30



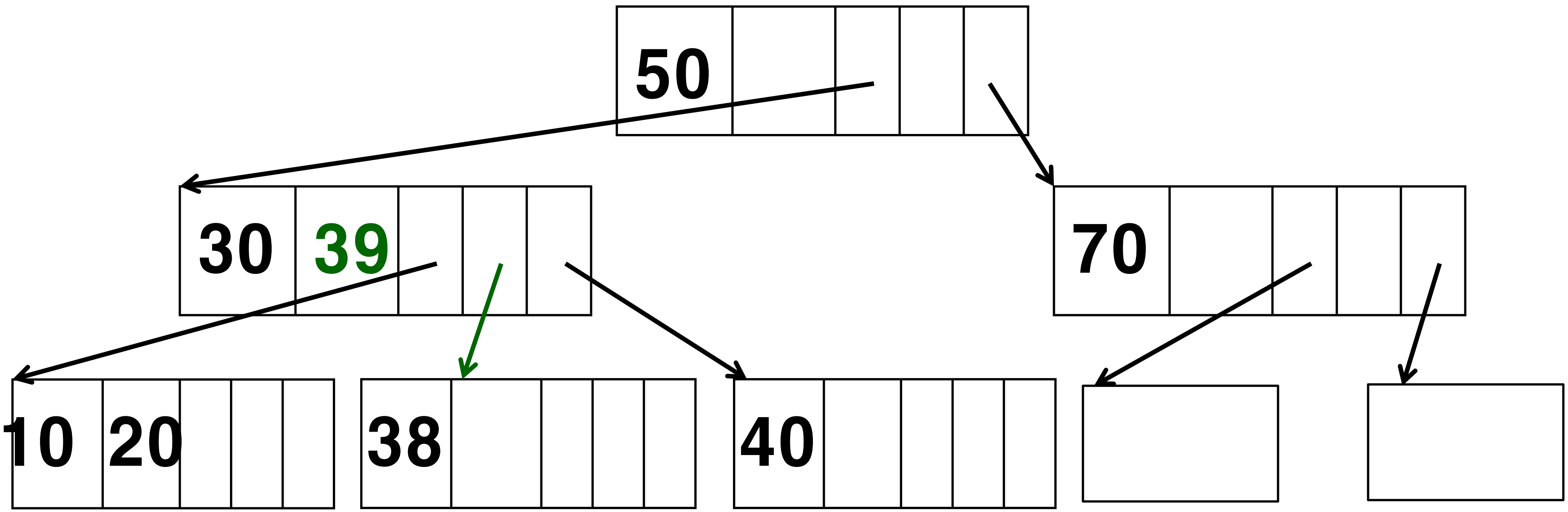
Example 4: (1/2)

insert 39



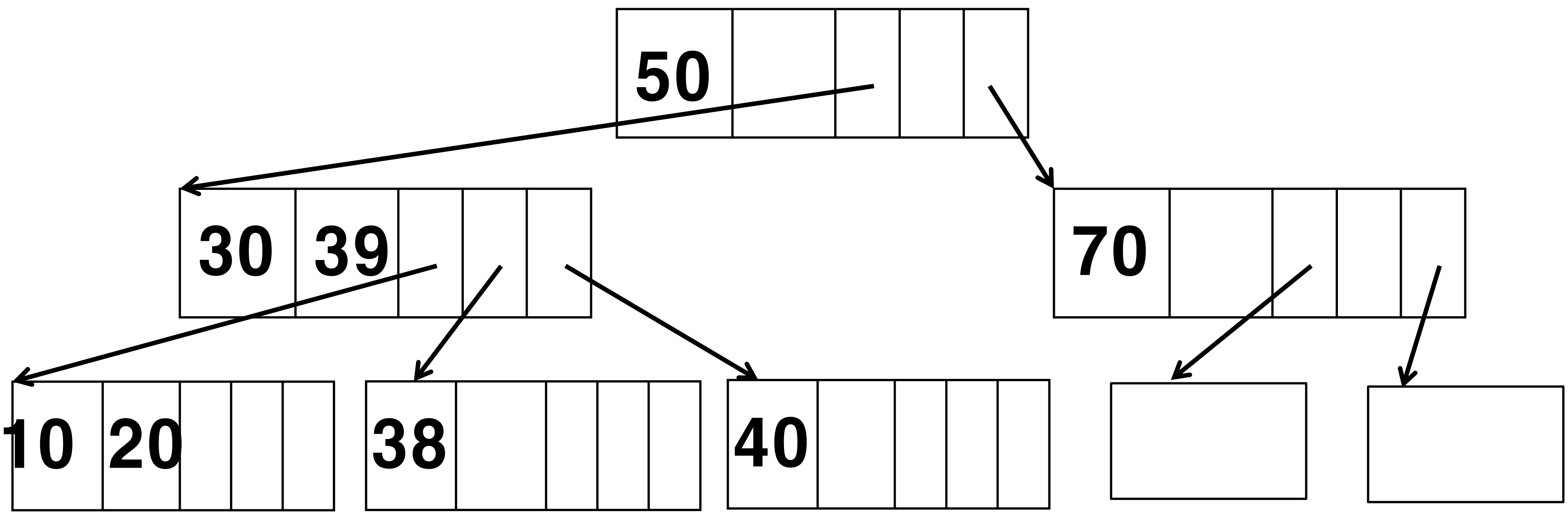
Example 4: (2/2)

insert 39



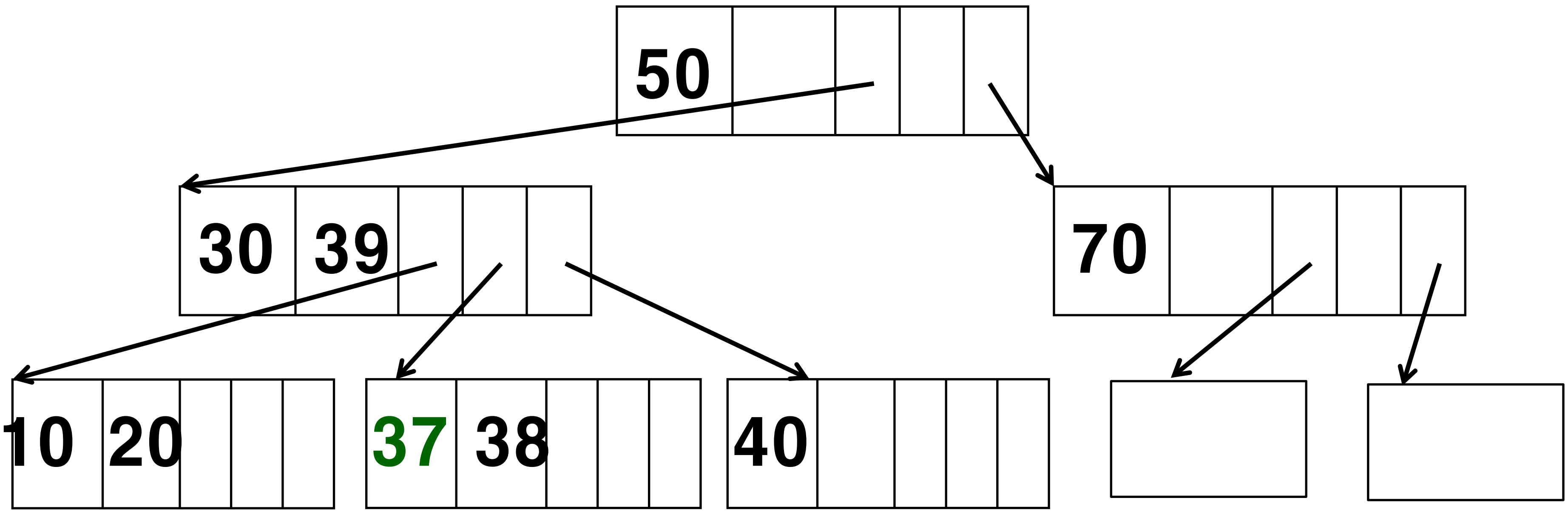
Example 5: (1/2)

insert 37



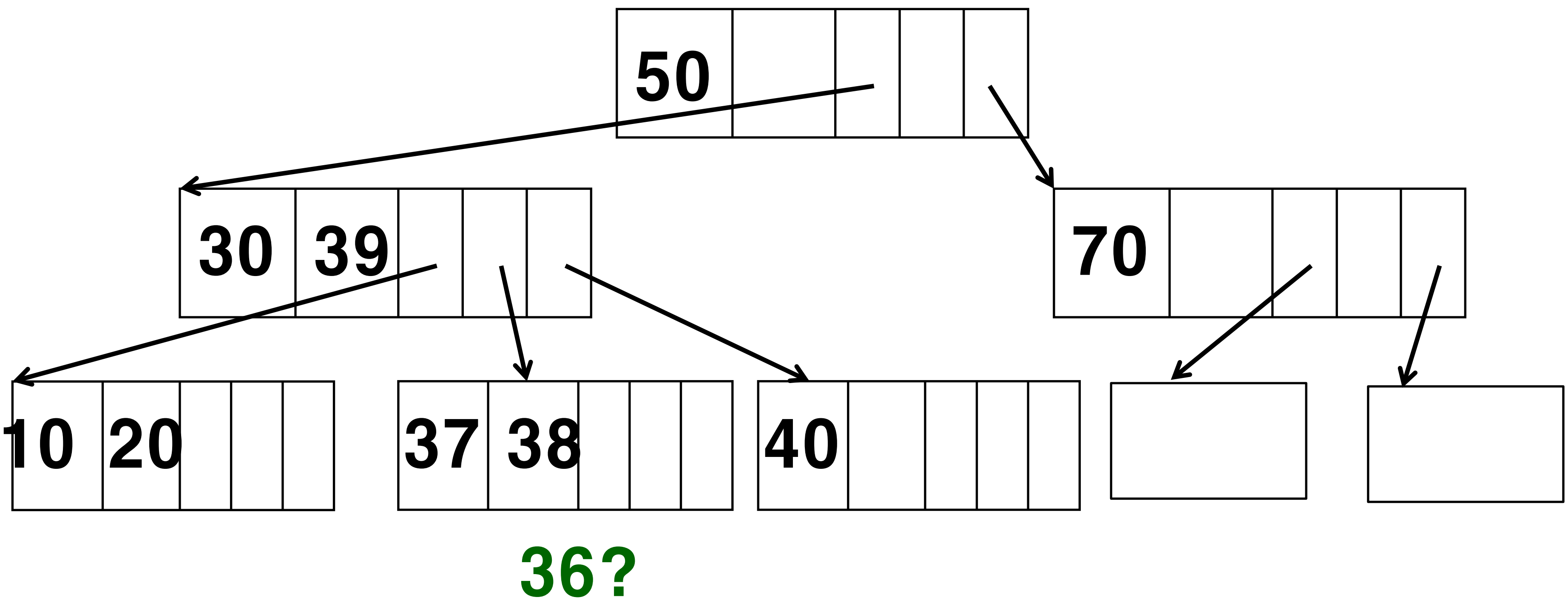
Example 5: (2/2)

insert 37



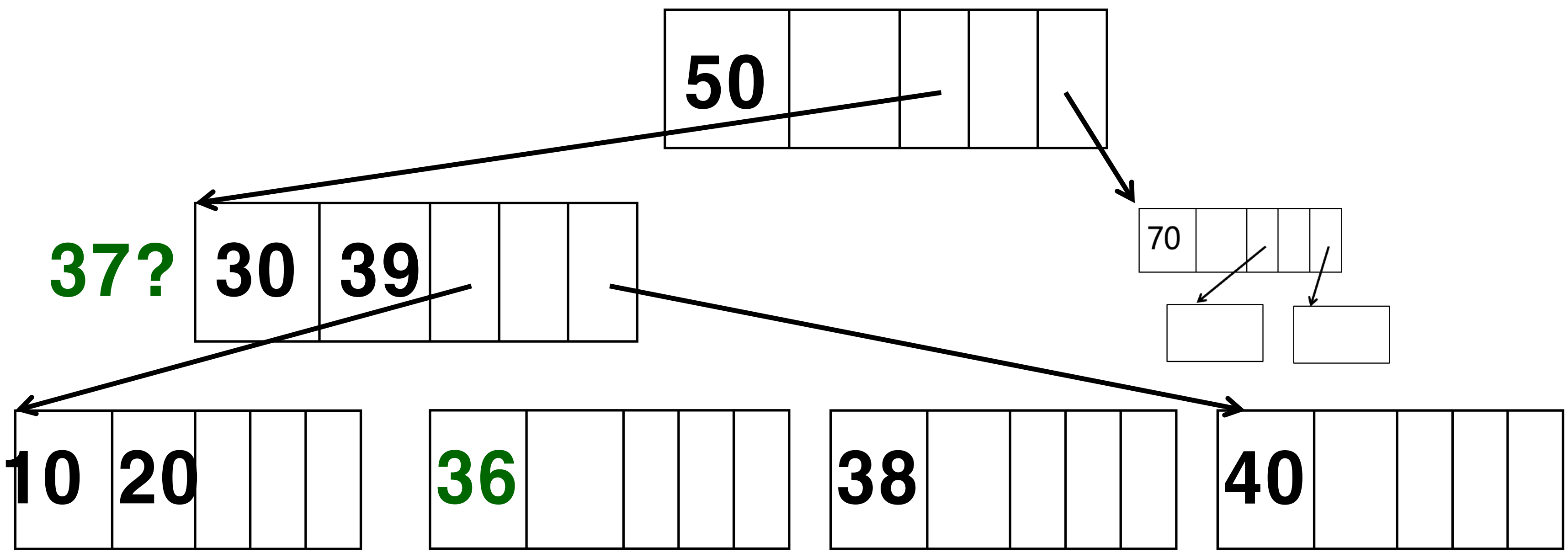
Example 6: (1/3)

insert 36



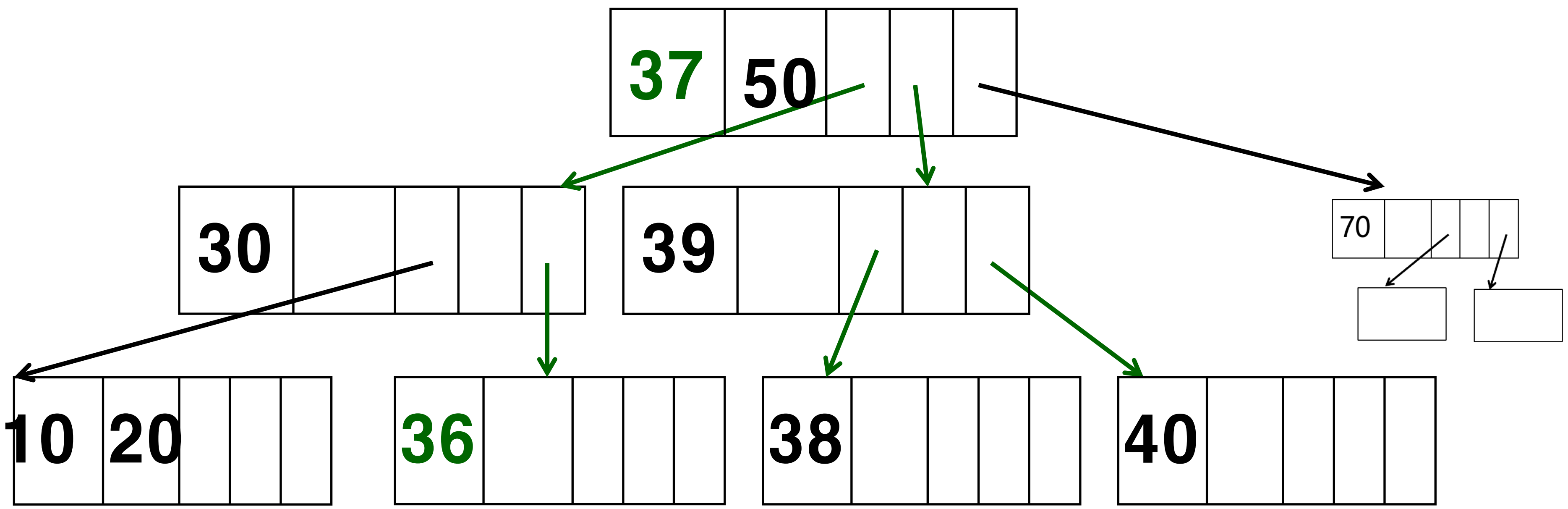
Example 6: (2/3)

insert 36



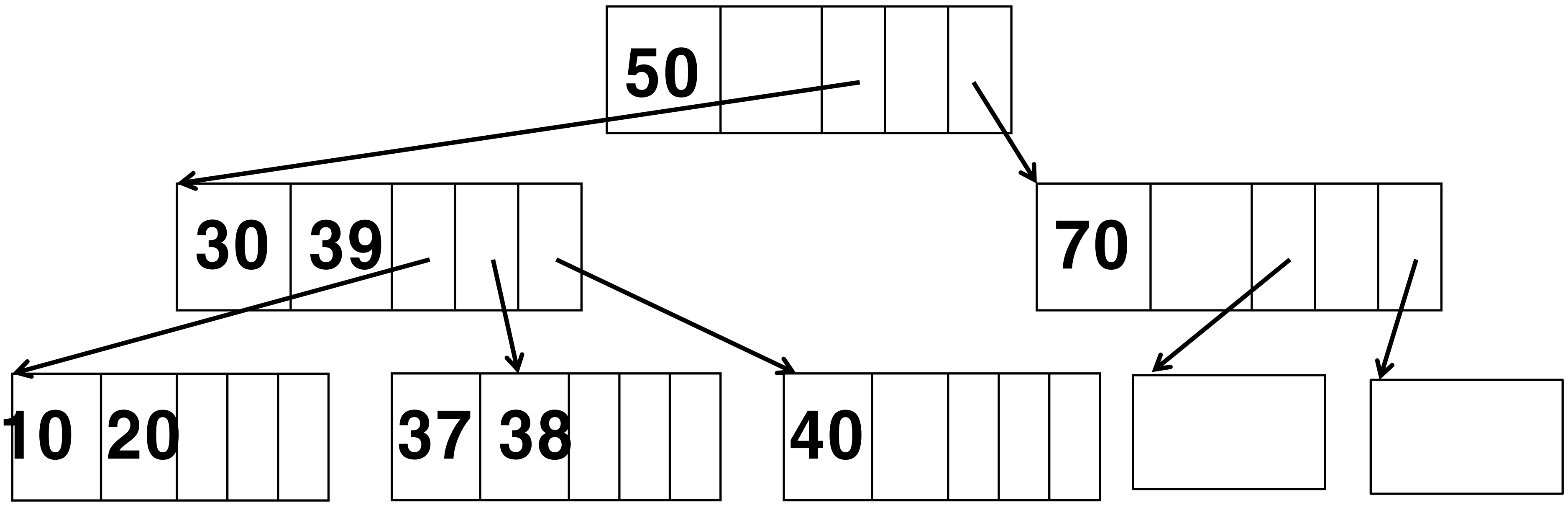
Example 6: (3/3)

insert 36



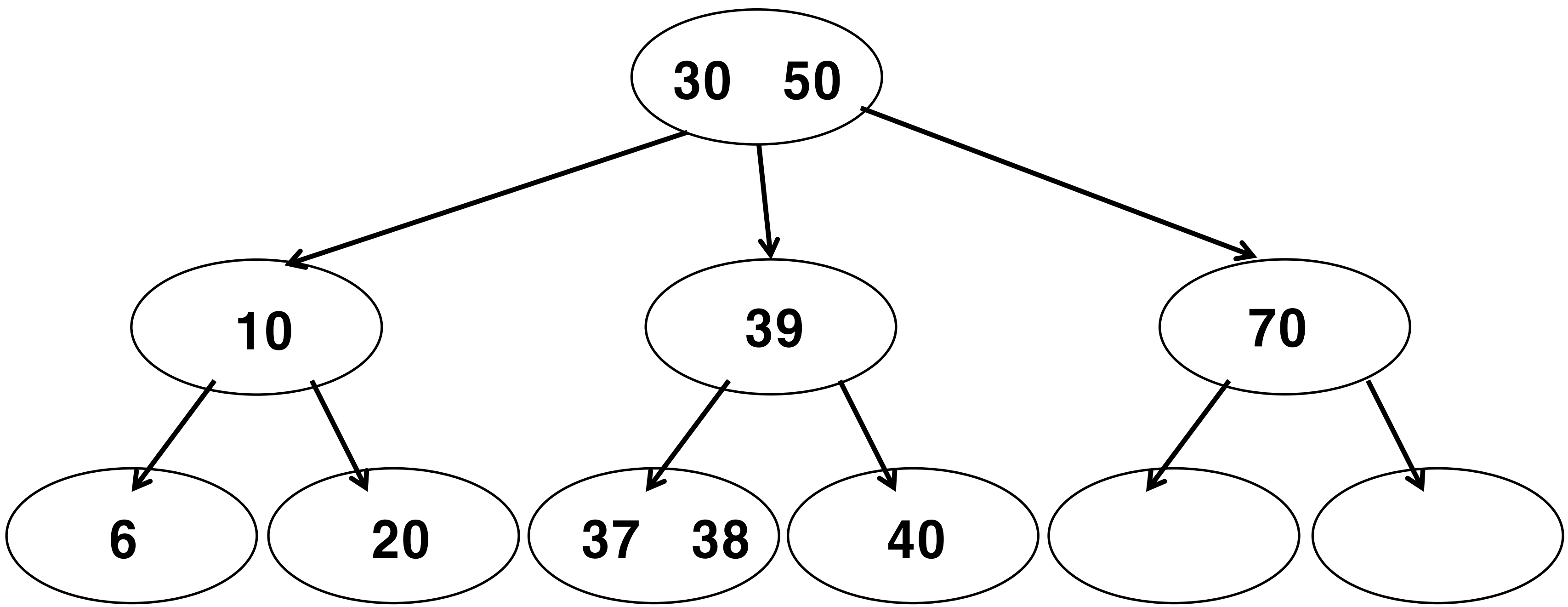
Exercise

insert 6



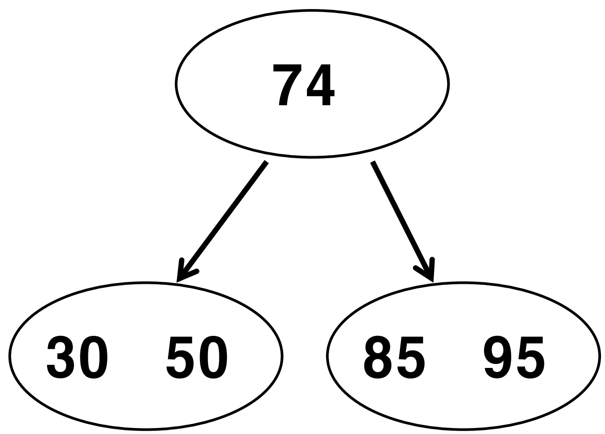
Exercise

simplified notation **insert 6**

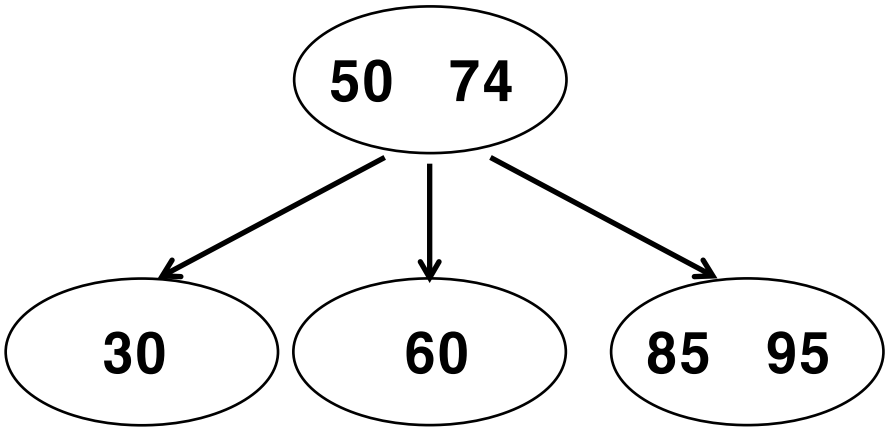


Exercise : Insert "60" Into the Following 2-3 Tree.

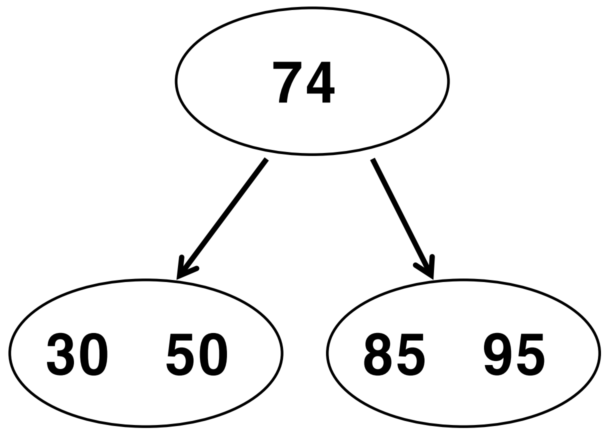
simplified notation



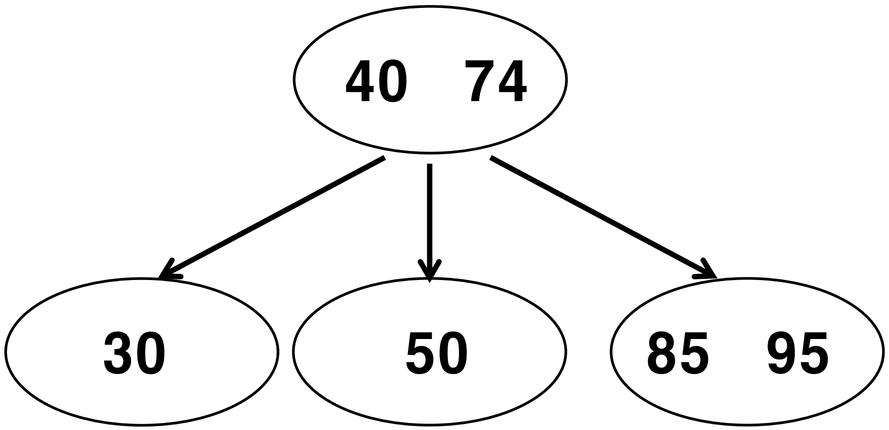
Exercise : Insert “60” Into the Following 2-3 Tree.



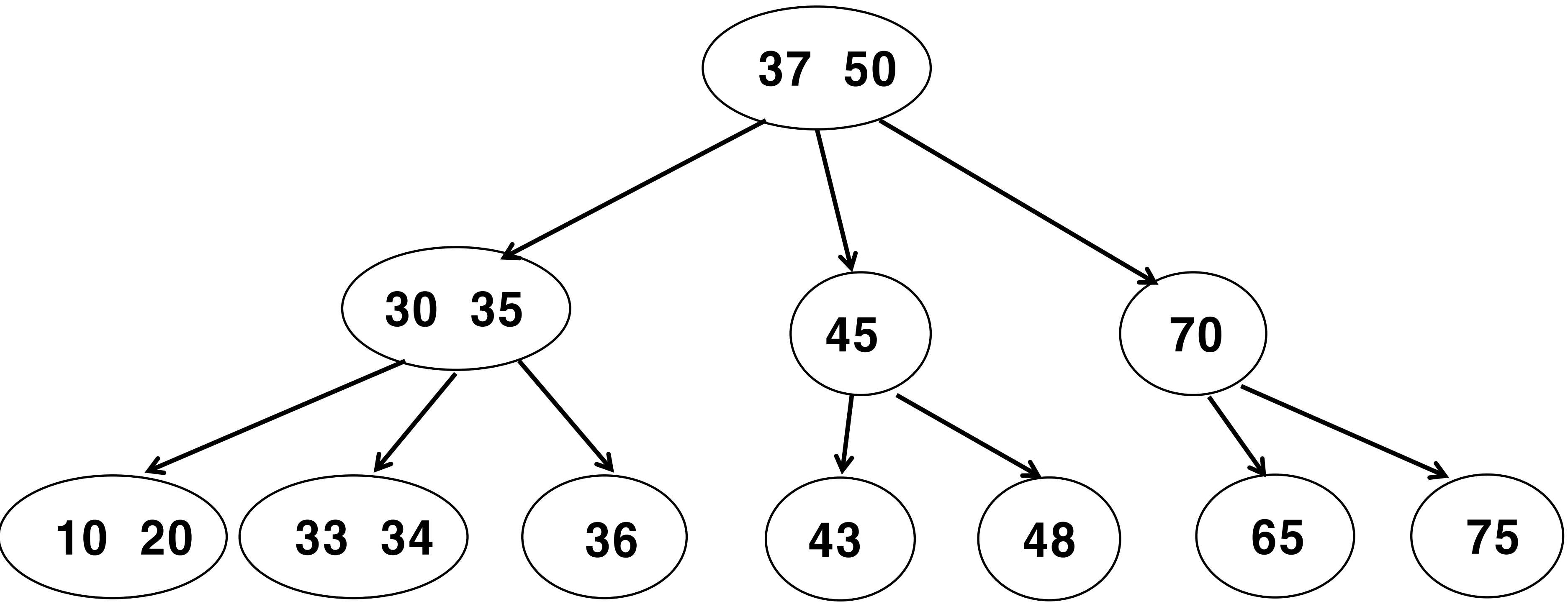
Exercise : Insert "40" Into the Following 2-3 Tree.



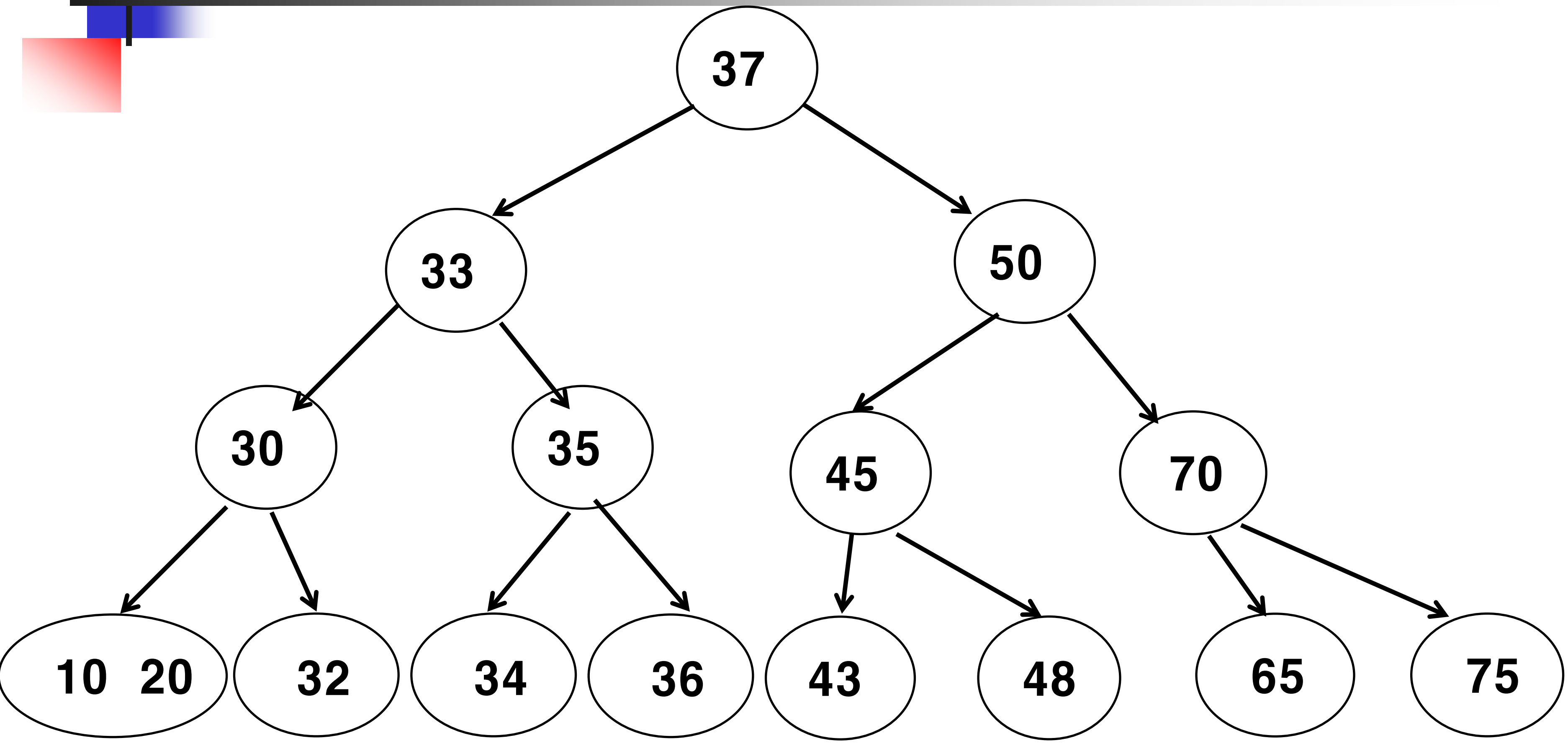
Exercise : Insert "40" Into the Following 2-3 Tree.



Exercise: Insert "32" into the Following 2-3 Tree.



Exercise: Insert “32” into the Following 2-3 Tree.



Deletion

- Node merge as the reverse of node split

Deleting Data from a 3-Node (1/2)

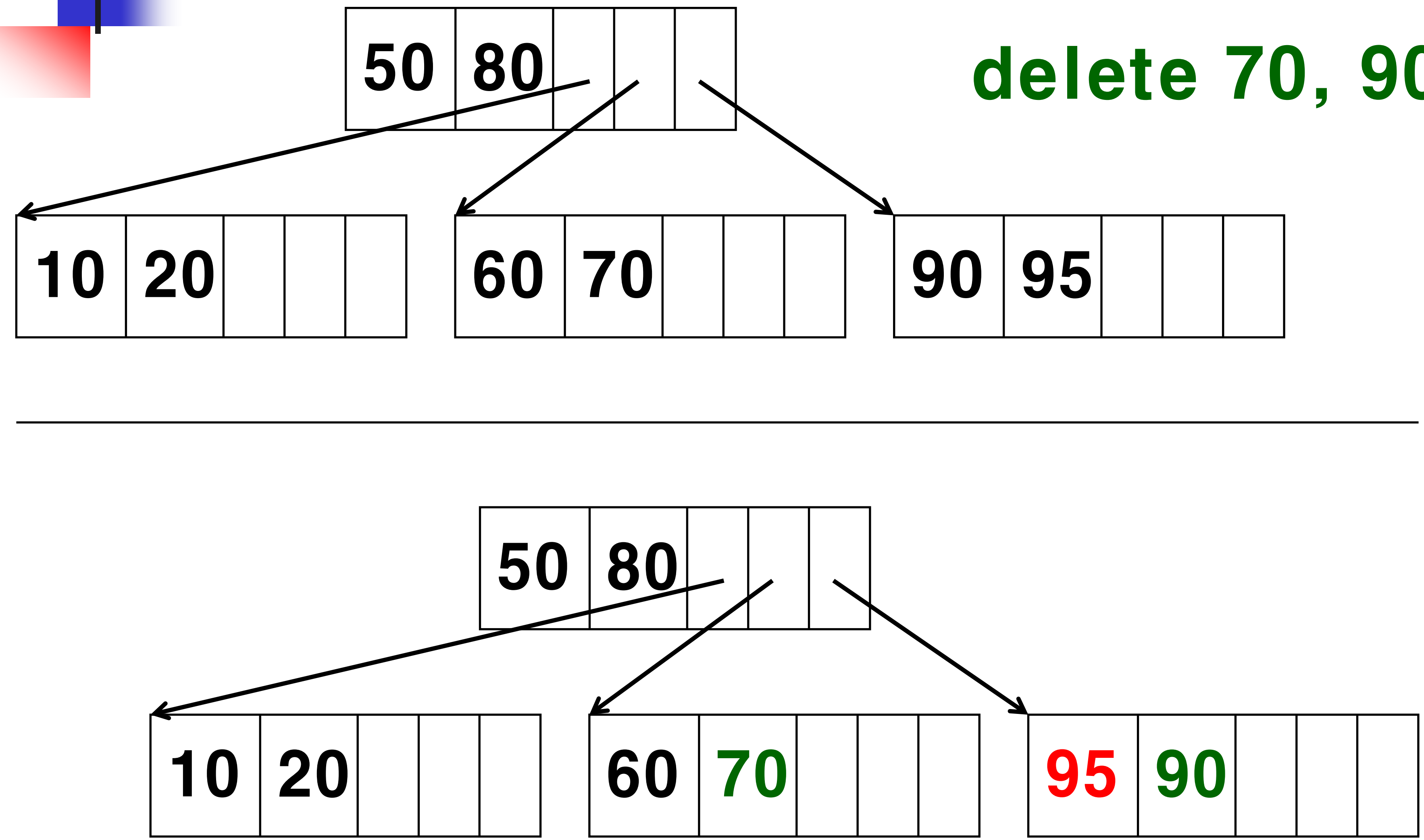
- If the 3-node is a leaf node
  - Just delete the data.
  - The node is now a 2-Node.

Deleting Data from a 3-Node (2/2)

- If the 3-node is a non-leaf node
- (with respect to the key to be deleted)
  - ** If both the left and right child nodes are 2-nodes
    - Merge the child nodes, and delete the key in the 3-node
  - *** If one of the left and right child nodes is a 3-node
    - If left data is to be deleted, swap the left data with the greatest key on the left subtree, or the smallest key on the middle subtree.
    - If right data is to be deleted, swap the right data with the greatest key on the middle subtree, or the smallest key on the right subtree.
    - Delete the data after the swap.
    - If the node underflows, solve the problem recursively.

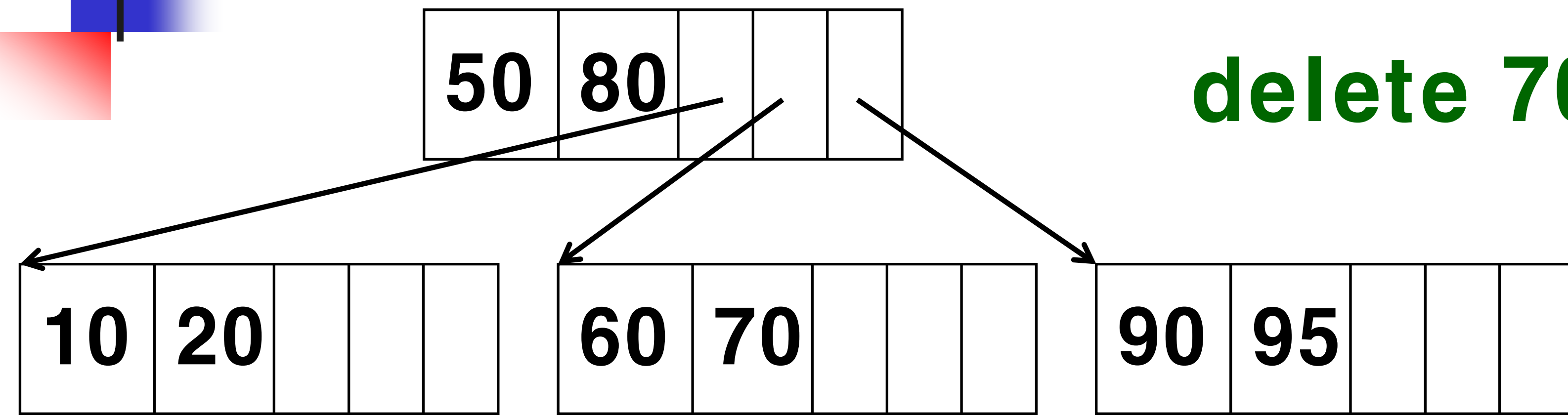
Example 1 (cf. page 37)

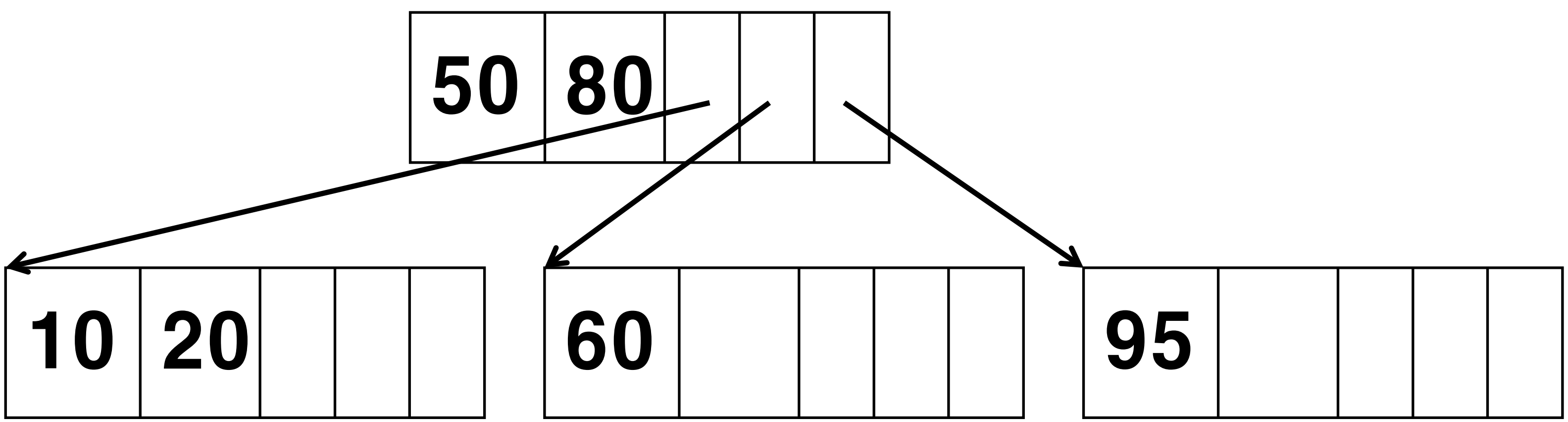
delete 70, 90



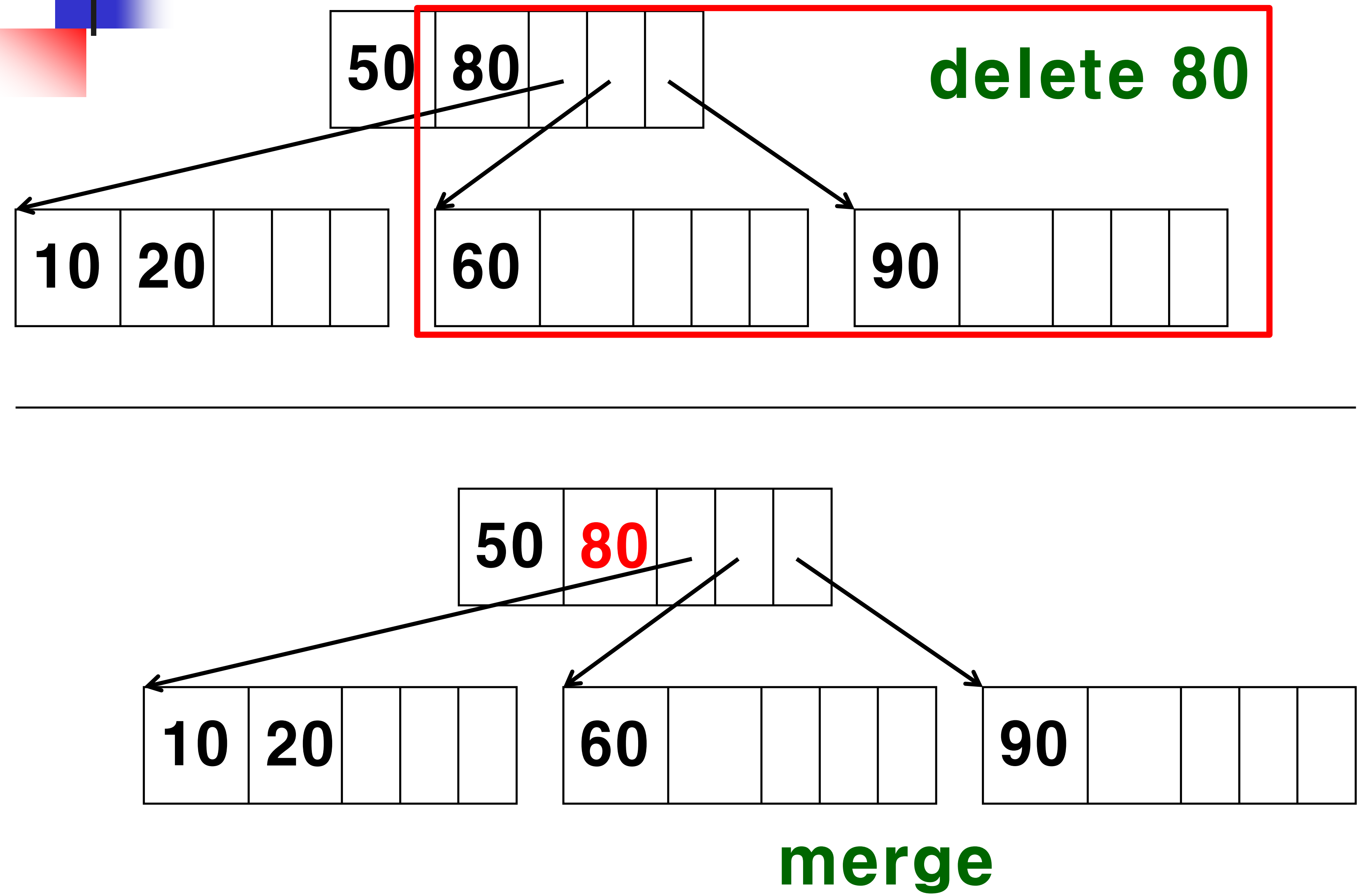
Example 1 (cont’d)

delete 70, 90

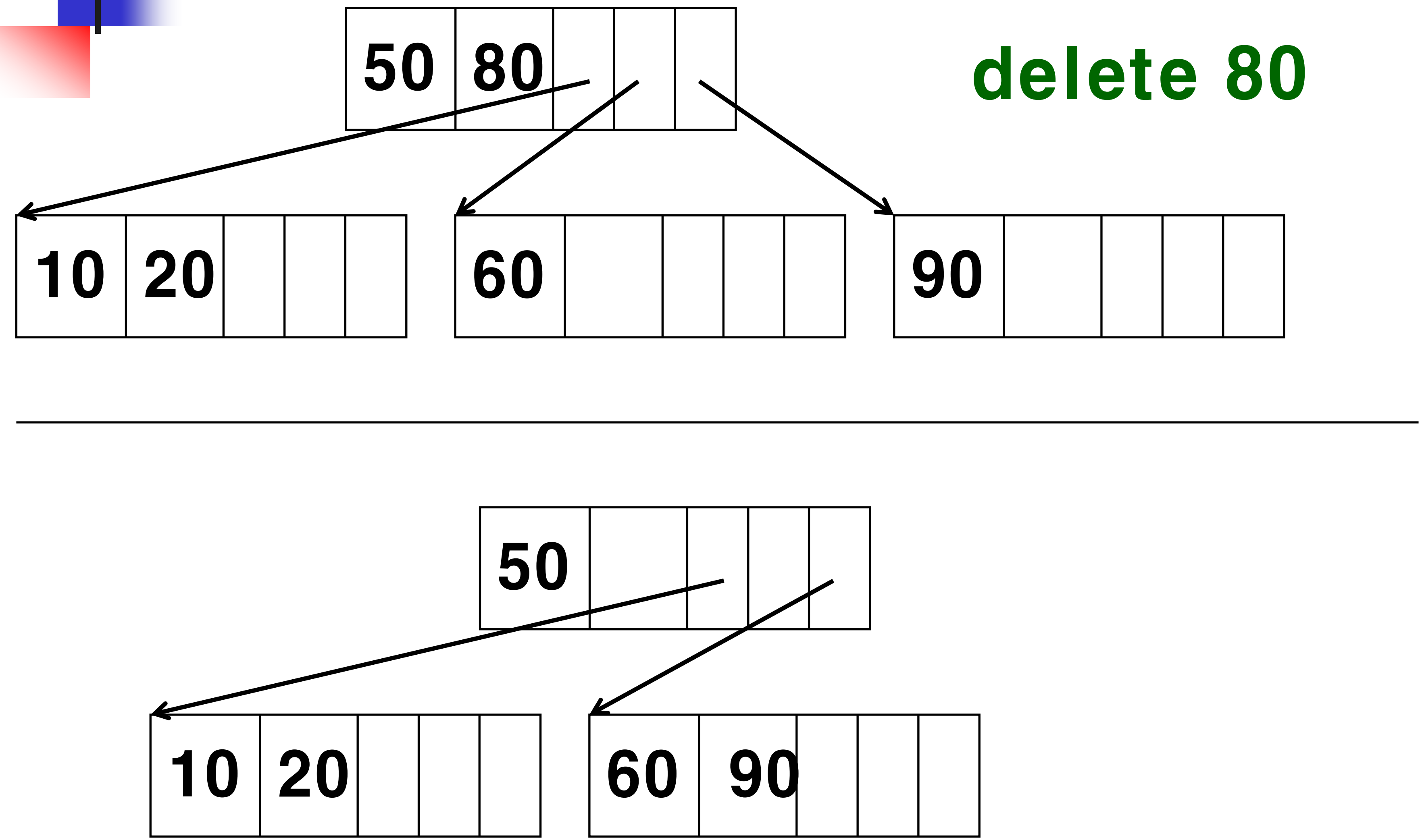




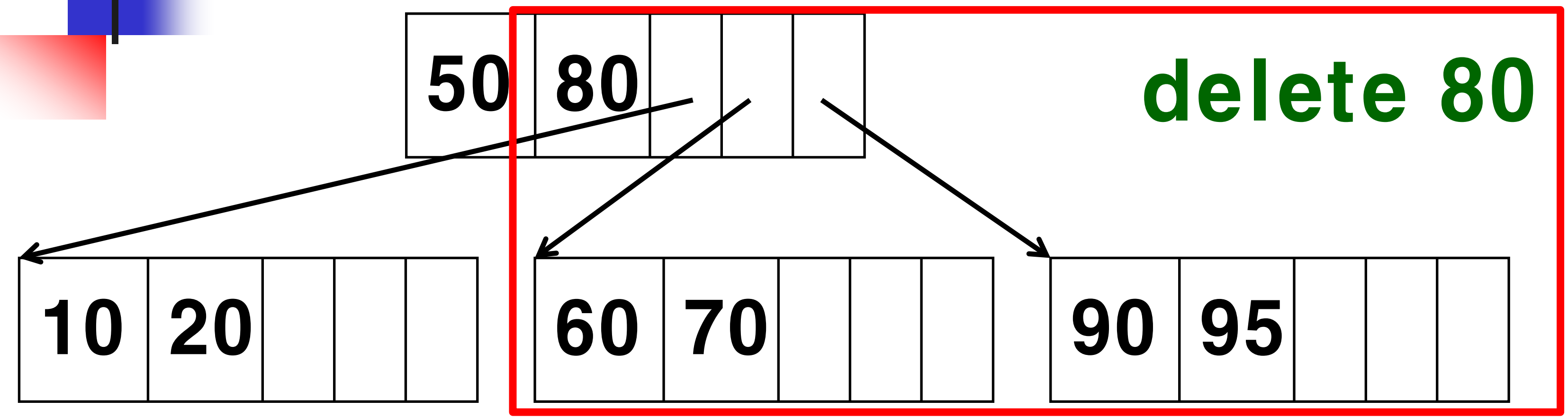
Example 2 (cf. page 38 **)

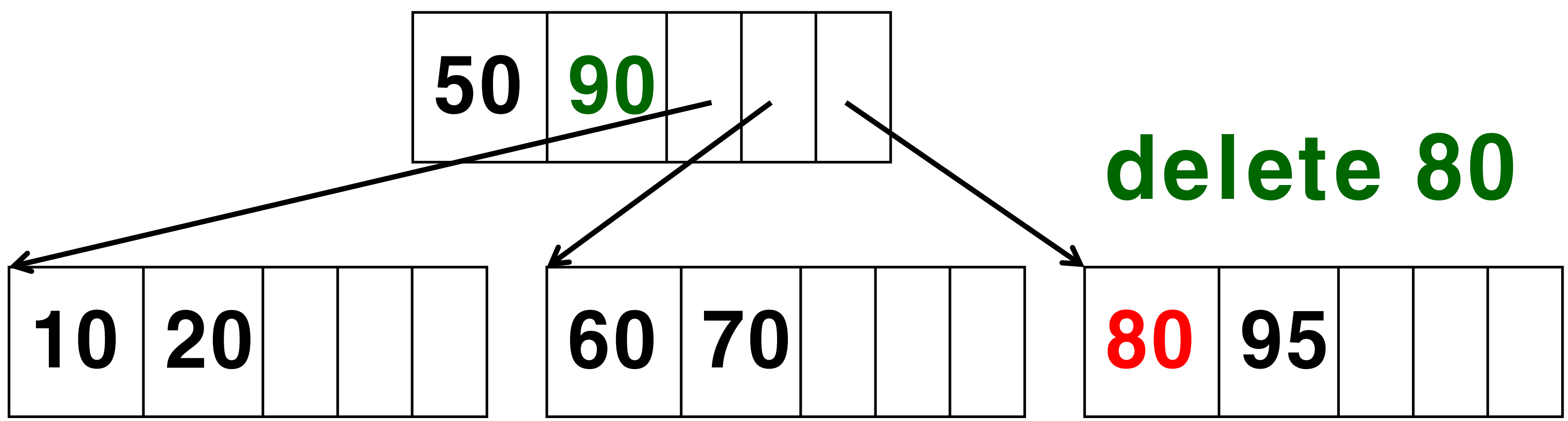


Example 2 (cont'd)



Example 3 (cf. page 37 ***)





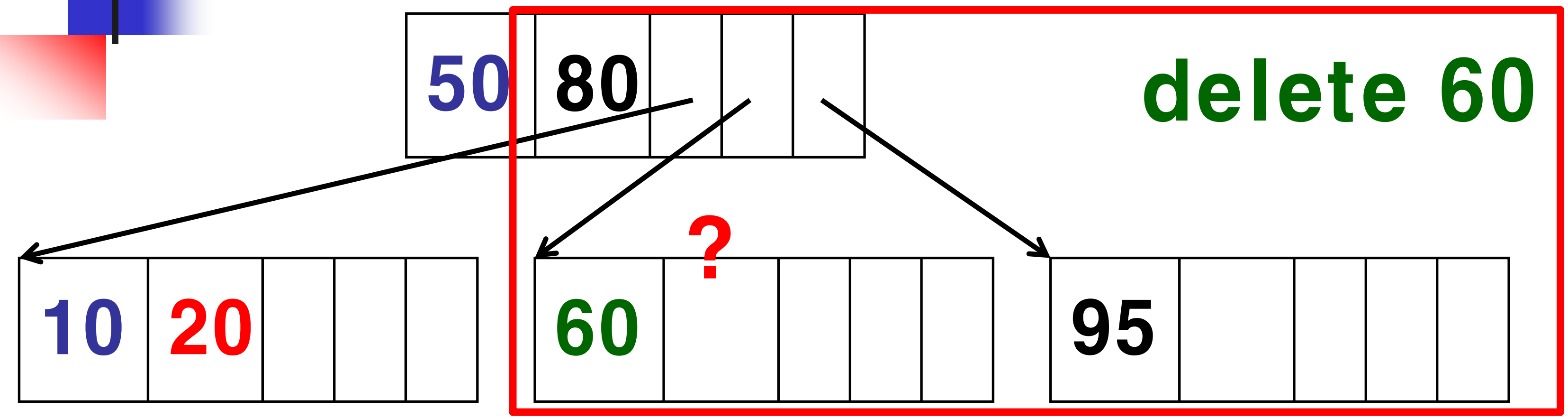
Deleting Data from a 2-Node (1/2)

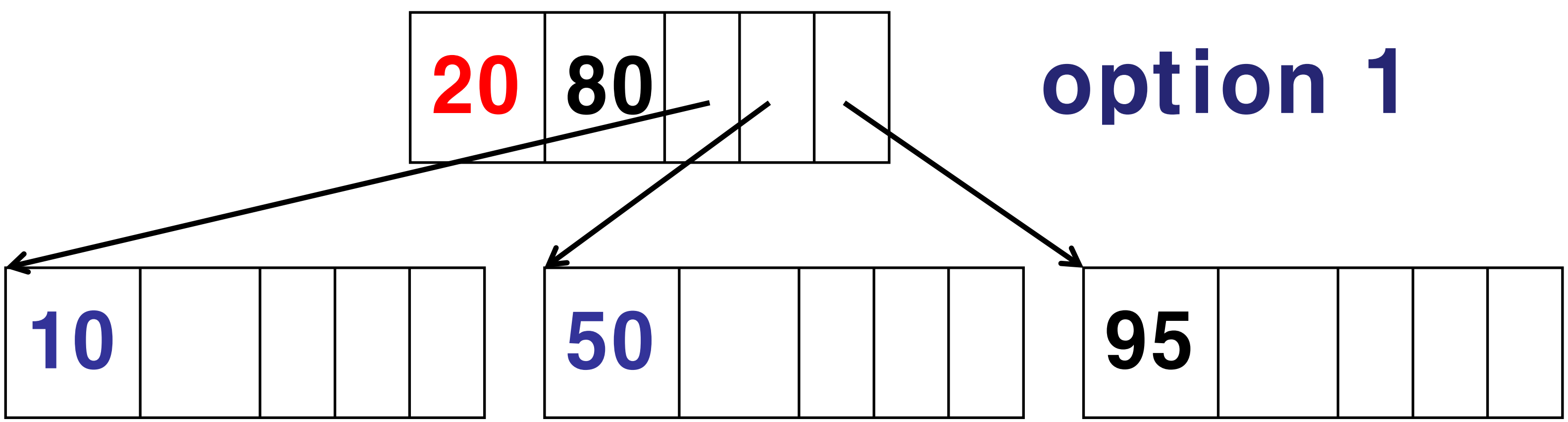
- If there is a sibling 3-node, delete the data in the 2-node (let's call it 2N), and
  - If 2N is the leftmost sibling, and
    - if the middle sibling node is a 3-Node (3N), move the smaller of the parent's data into 2N, and move the smaller of 3N's data into the parent node.
    - if the middle sibling node is a 2-Node, move the smaller of the parent's data into the middle sibling node, and delete 2N.
  - If 2N is the rightmost sibling, and
    - if the middle sibling node is a 3-Node (3N), move the larger of the parent's data into 2N, and move the larger of 3N's data into the parent node.
    - if the middle sibling node is a 2-Node, move the larger of the parent's data into the middle sibling node, and delete 2N.
  - ** If 2N is the middle sibling,
    - ** If the leftmost node is the sibling 3-Node (3N), move the smaller of the parent's data into 2N, and move the larger of 3N's data into the parent node.
    - If the rightmost node is the sibling 3-Node (3N), move the larger of the parent's data into 2N, and move the smaller of 3N's data into the parent node.
  - Adjust the pointers in the sibling node and/or the parent node.

Deleting Data from a 2-Node (2/2)

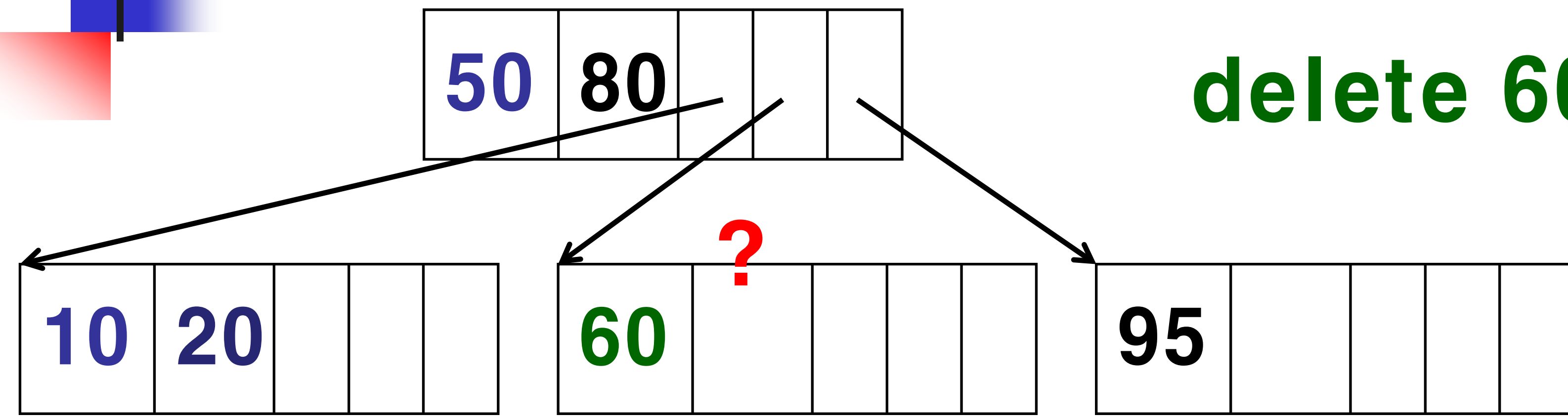
- If there is no sibling 3-node,
  - Move parent's data to the left or right sibling node of the 2-Node (2N), and delete 2N. (The parent node and the sibling node are merged.)
  - If the parent node underflows as a result, take care of the parent node deletion.
  - Adjust the pointers in the sibling node and/or the parent node.

Example 4 (cf. page 44**)

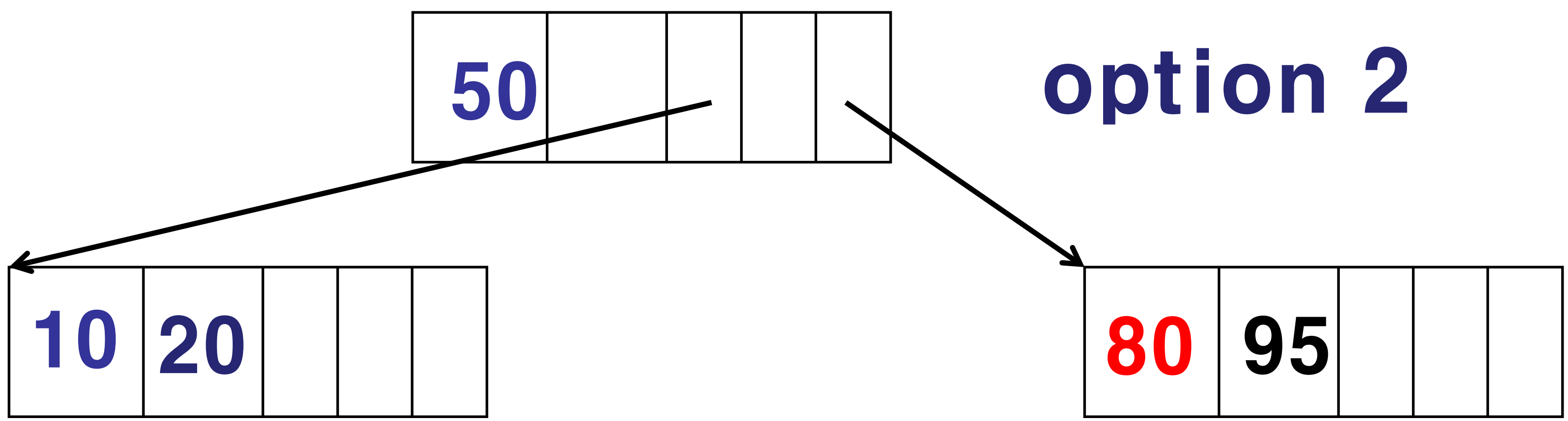




Example 4 (cont’d)

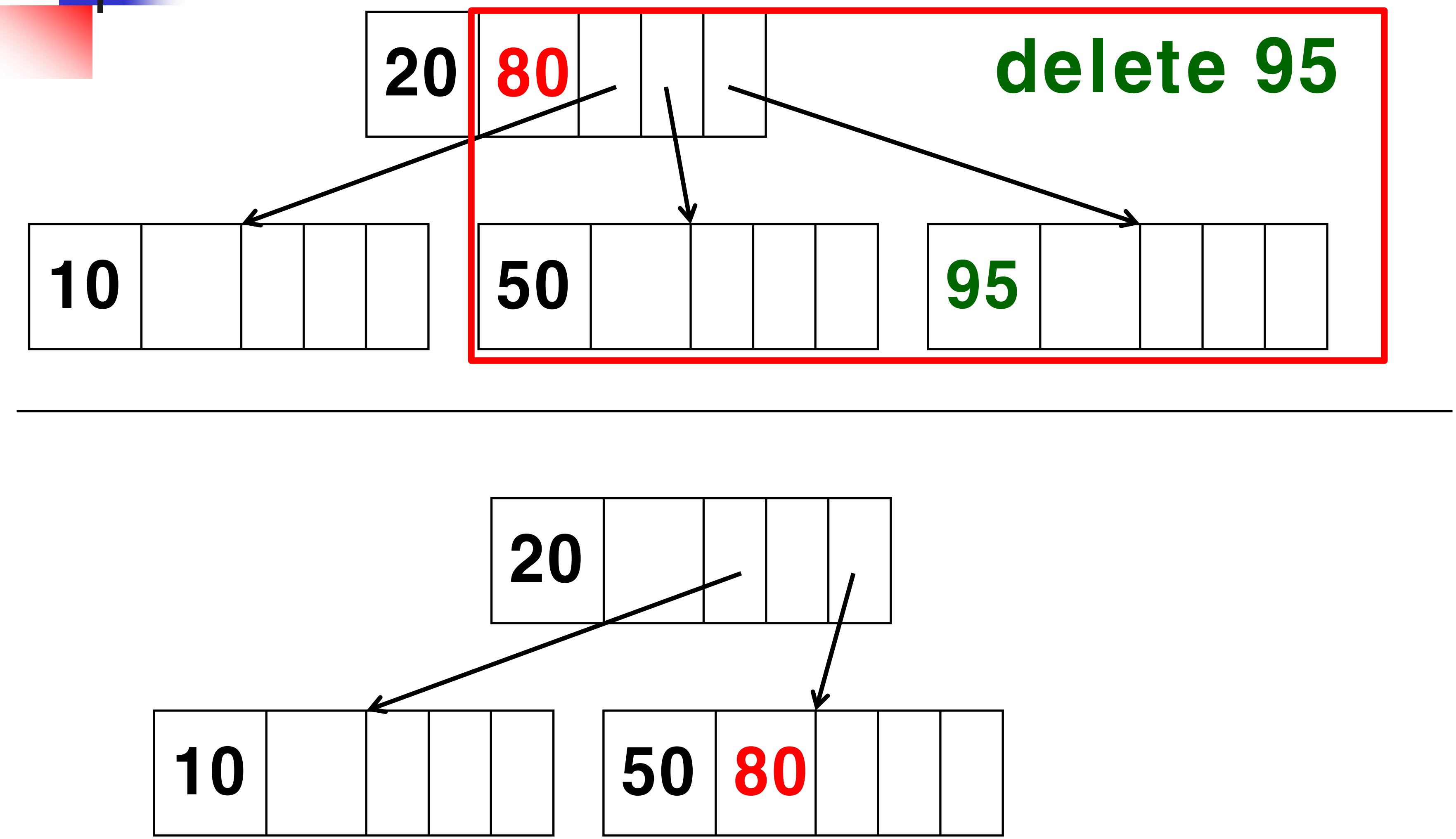


delete 60

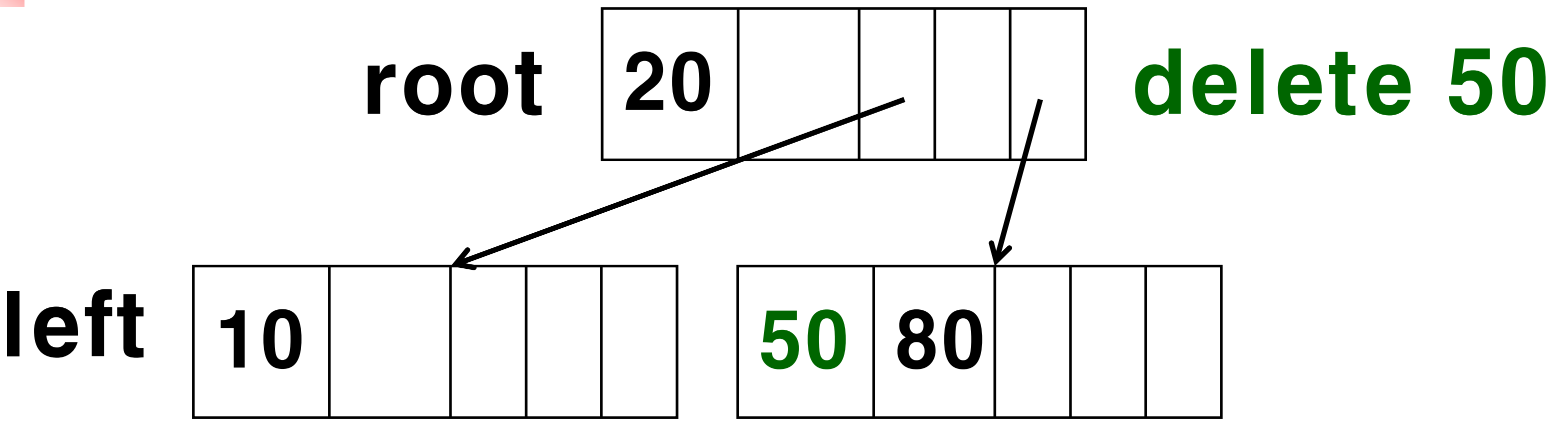


option 2

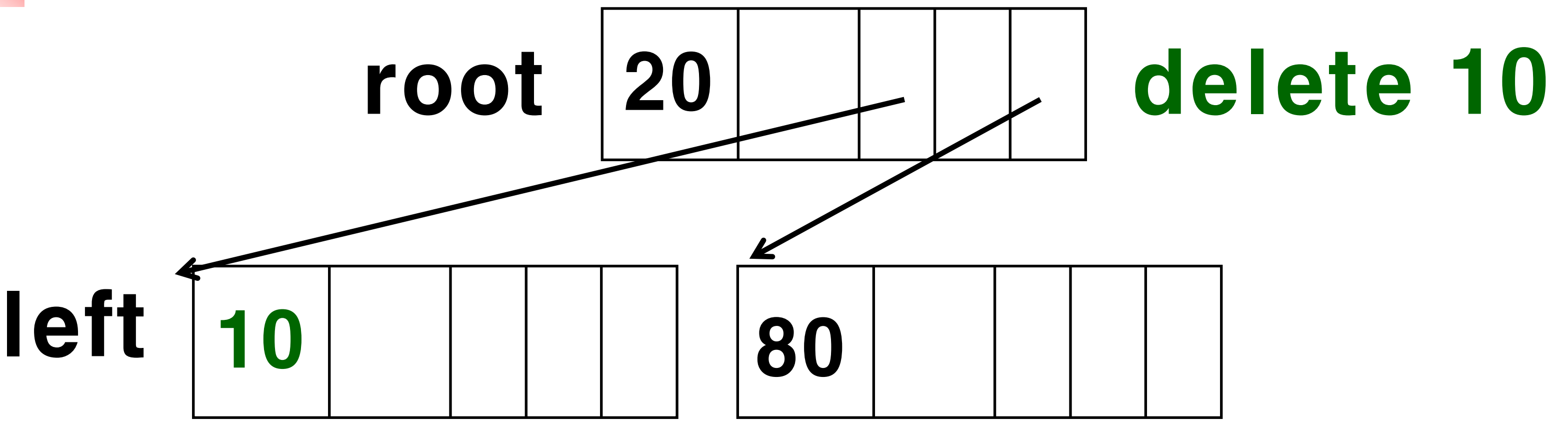
Example 5 (cf. page 45)

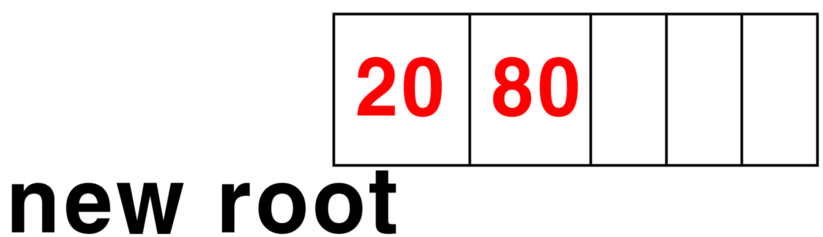


Example 6



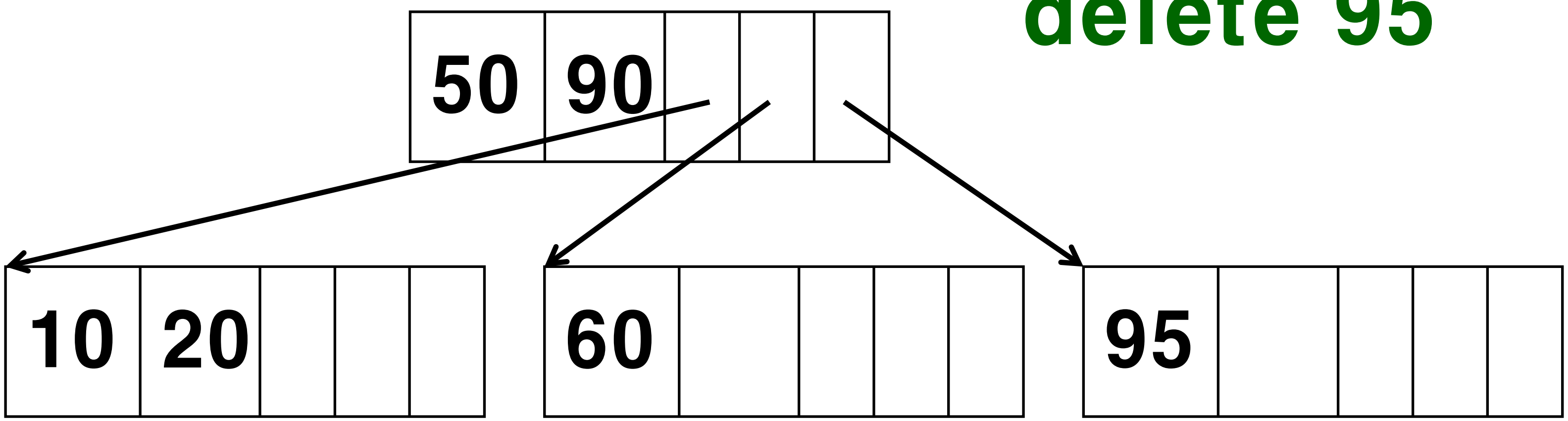
Example 7





Exercise

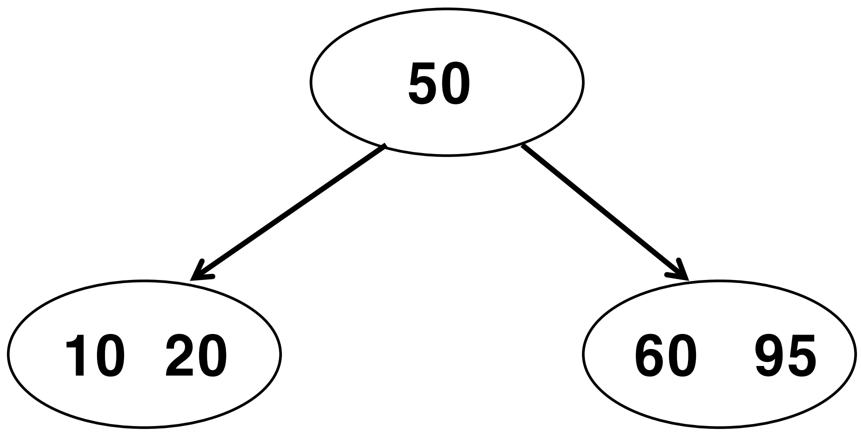
delete 90
delete 50
delete 20
delete 95



Try delete 90 first

Exercise

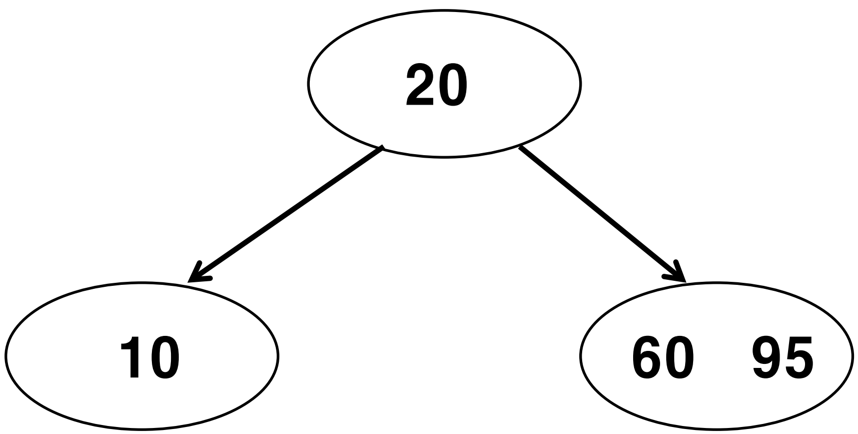
simplified notation **delete 90**



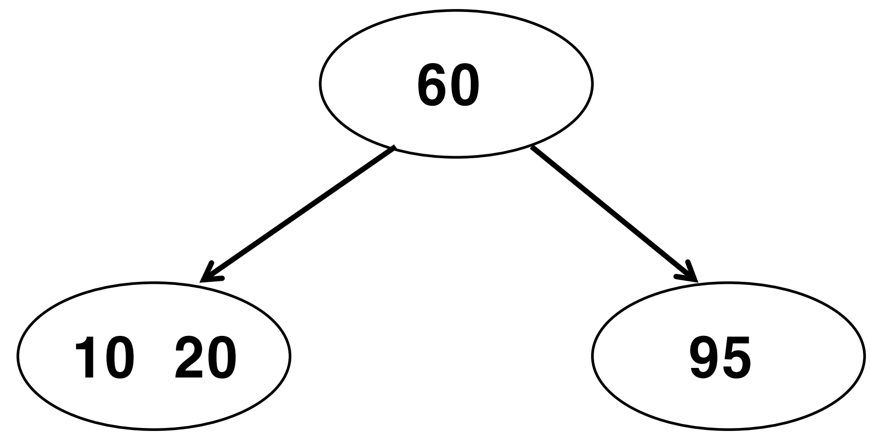
Try delete 50 next

Exercise

delete 50



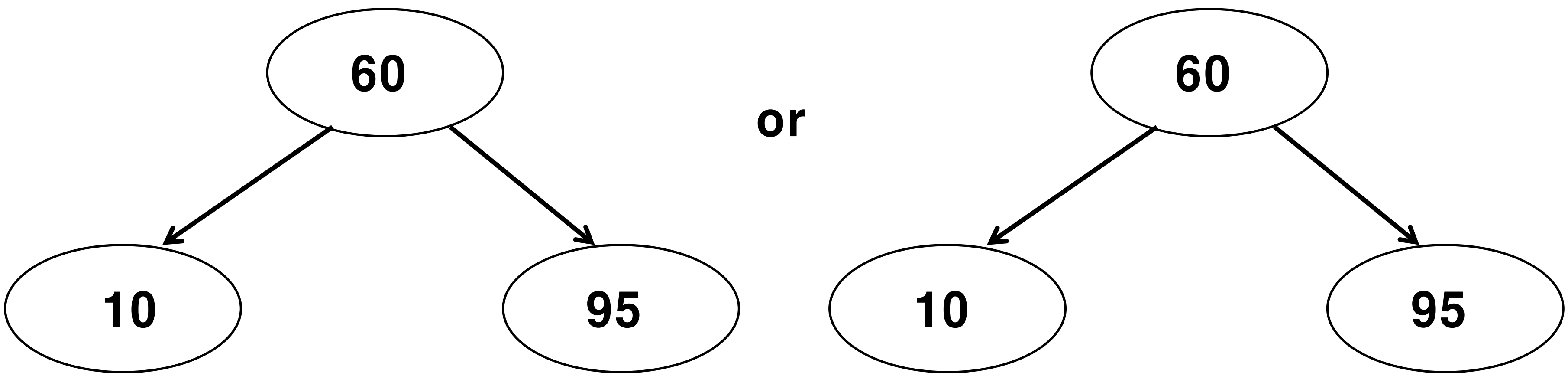
or



Try delete 20 next

Exercise

delete 20



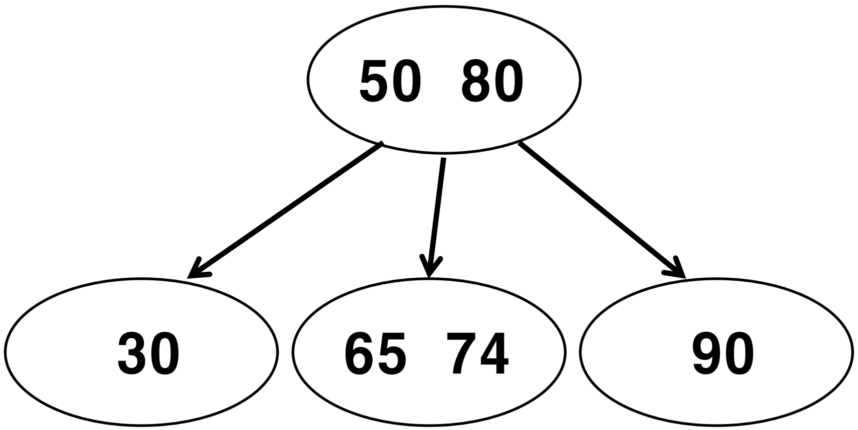
Try delete 95 next

Exercise

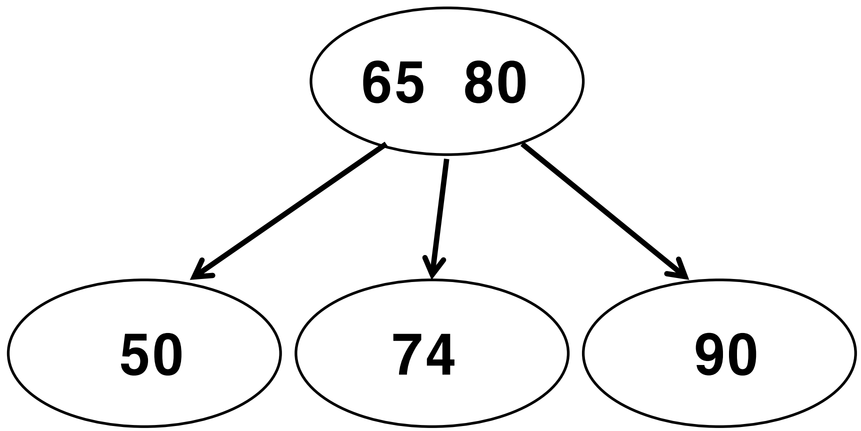
delete 95

10 60

Exercise : Delete "30" From the Following 2-3 Tree.



Exercise : Delete "30" From the Following 2-3 Tree.



Performance of a 2-3 Tree

- Average Case and Worst-Case
  - Between $O(\log_3 n)$ and $O(\log_2 n)$
  - $O(\log_2 n)$: if all nodes are 2-Nodes
  - $O(\log_3 n)$: if all nodes are 3-Nodes

End of Lecture