CS302 - Data Structures *using C++*

Topic: Balanced Search Trees - Introduction

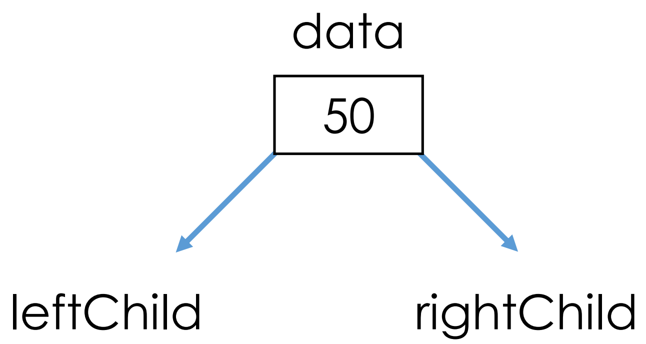
Kostas Alexis

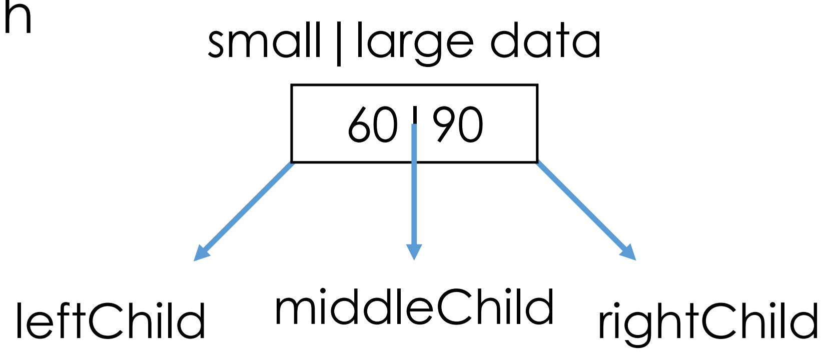
2-3 Trees

- General Search Tree
- Interior nodes are either 2-nodes or 3-nodes
  - 2-node has one data item and two children
  - 3-node has two data items and three children
    - Simple implementations may use 3-nodes for both

2-3 Trees

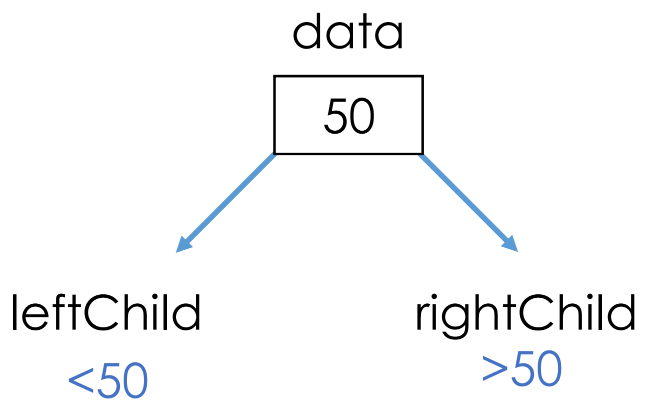
- General Search Tree
- Interior nodes are either 2-nodes or 3-nodes
  - 2-node has one data item and two children
  - 3-node has two data items and three children
    - Simple implementations may use 3-nodes for both

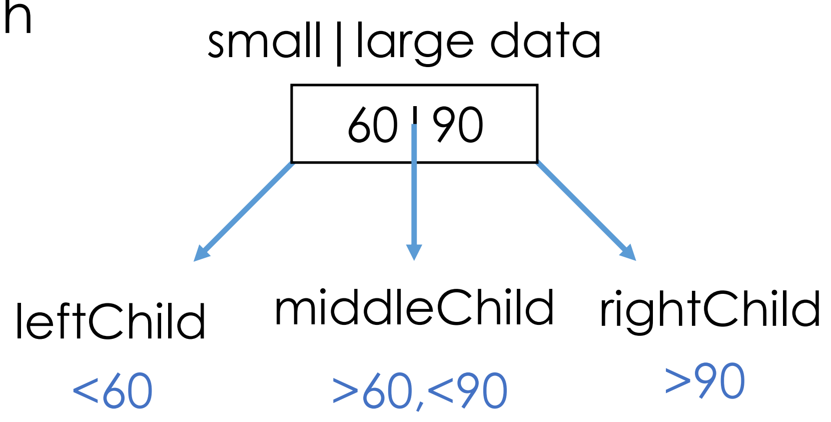




2-3 Trees

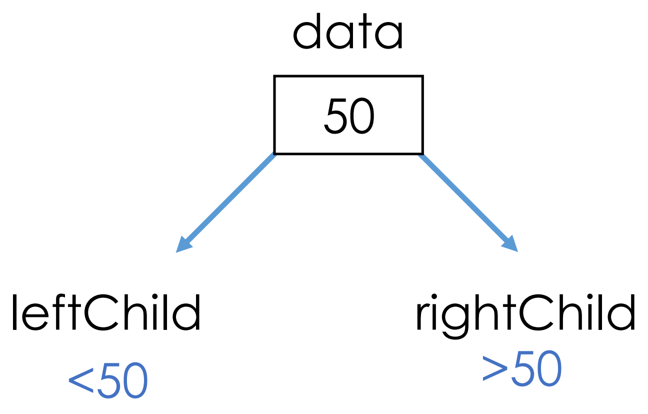
- General Search Tree
- Interior nodes are either 2-nodes or 3-nodes
    - 2-node has one data item and two children
    - 3-node has two data items and three children
        - Simple implementations may use 3-nodes for both

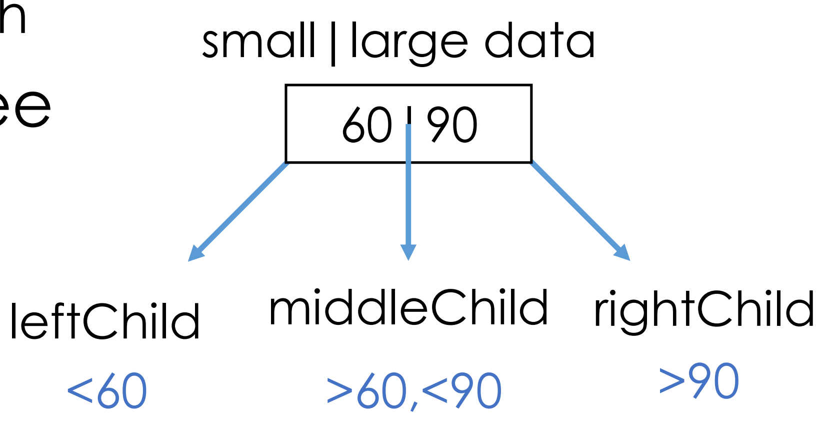




2-3 Trees

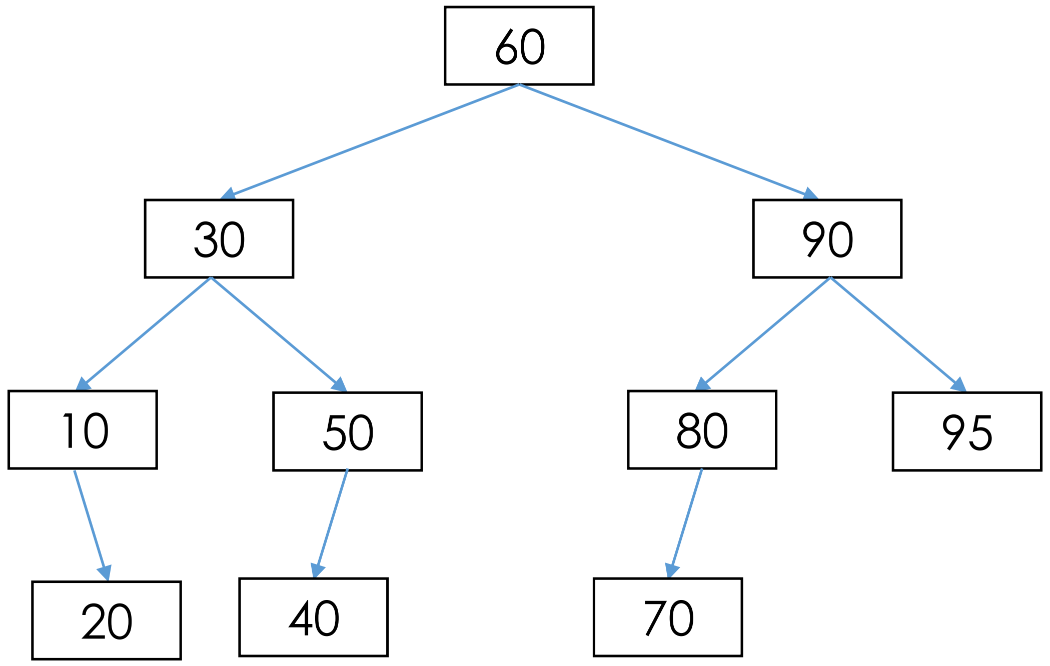
- General Search Tree
- Interior nodes are either 2-nodes or 3-nodes
  - 2-node has one data item and two children
  - 3-node has two data items and three children
    - Simple implementations may use 3-nodes for both
- Are never taller than minimum-height binary tree
  - A 2-3 tree with n nodes never has height greater than $\log_2(n+1)$



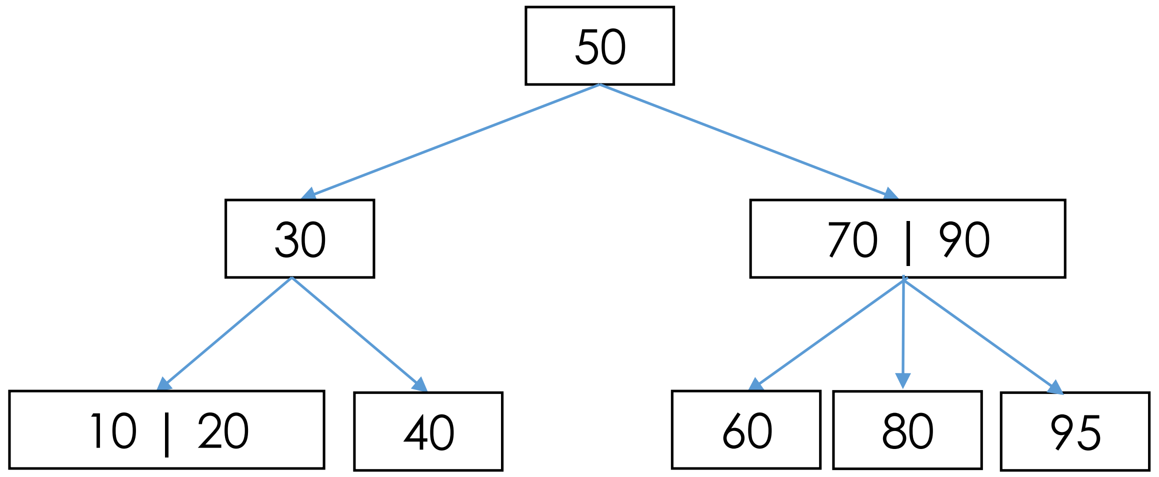


2-3 Trees

A Balanced Binary Search Tree

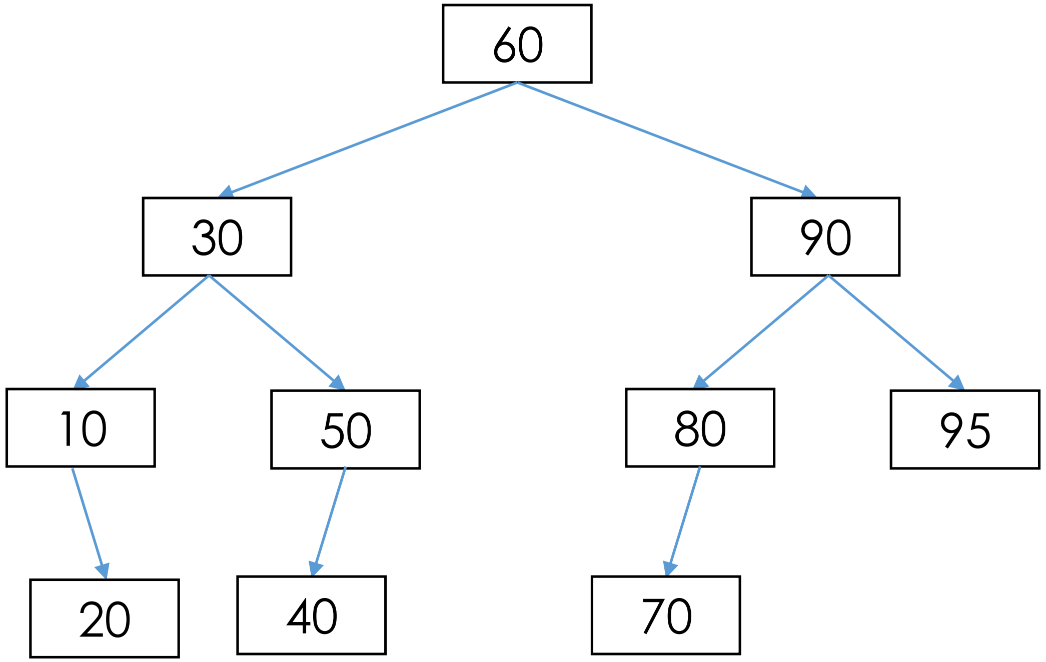


2-3 Tree with the same elements

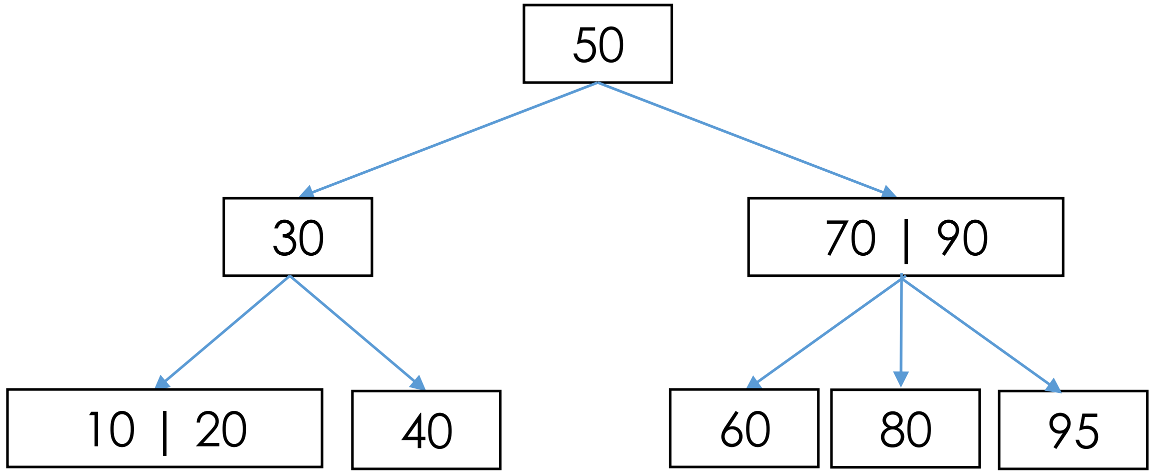


2-3 Trees

A Balanced Binary Search Tree – Height = 4

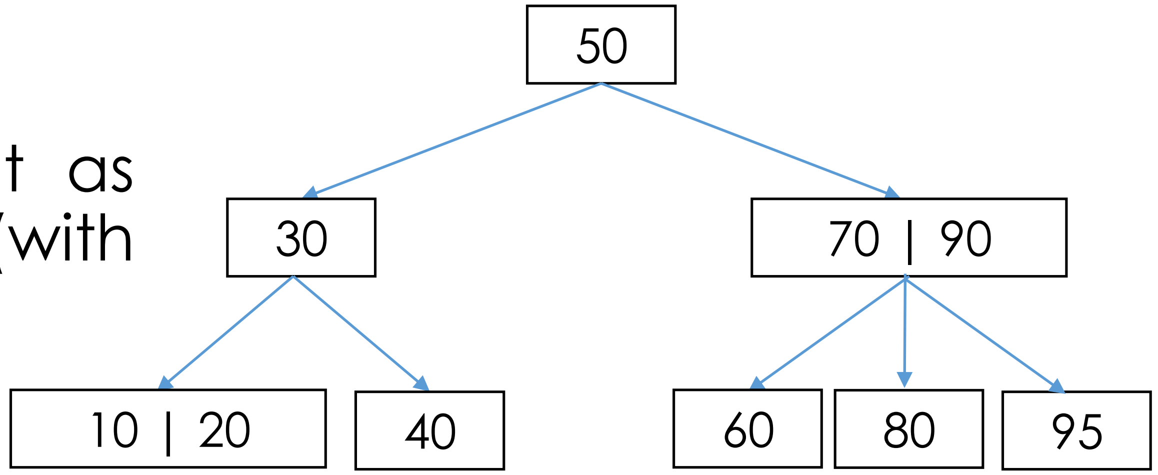


2-3 Tree with the same elements – Height = 3



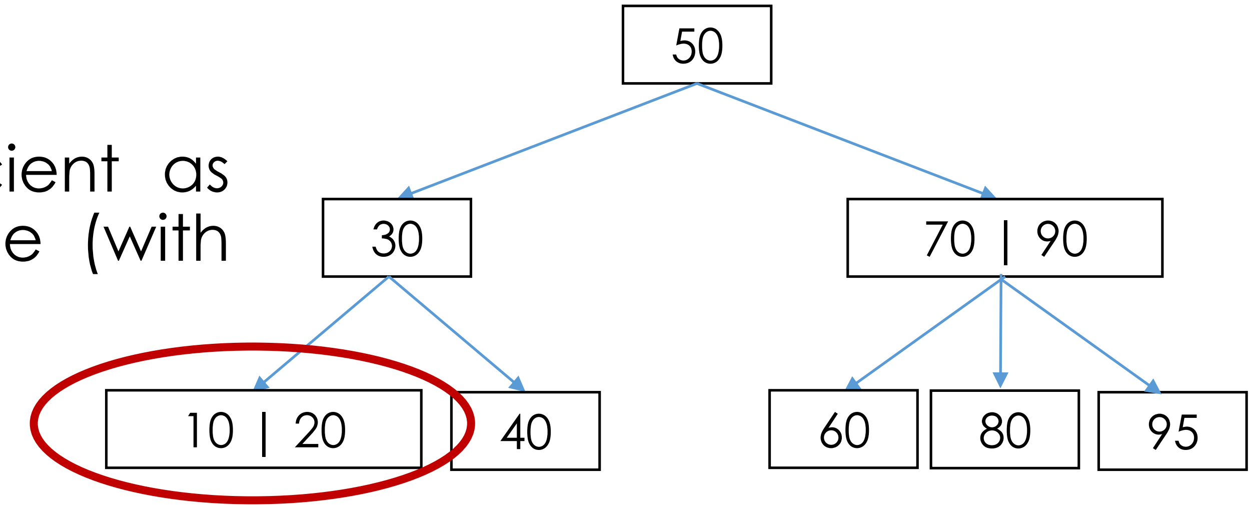
Traversing 2-3 Trees

- To traverse a 2-3 Tree
  - Perform the analogue of an in-order traversal
    - Leftmost subtree,
    - Left value,
    - Center subtree,
    - Right value,
    - Rightmost subtree
- Searching a 2-3 tree is as efficient as searching the shorted binary tree (with the same values)
  - $O(\log_2 n)$



Traversing 2-3 Trees

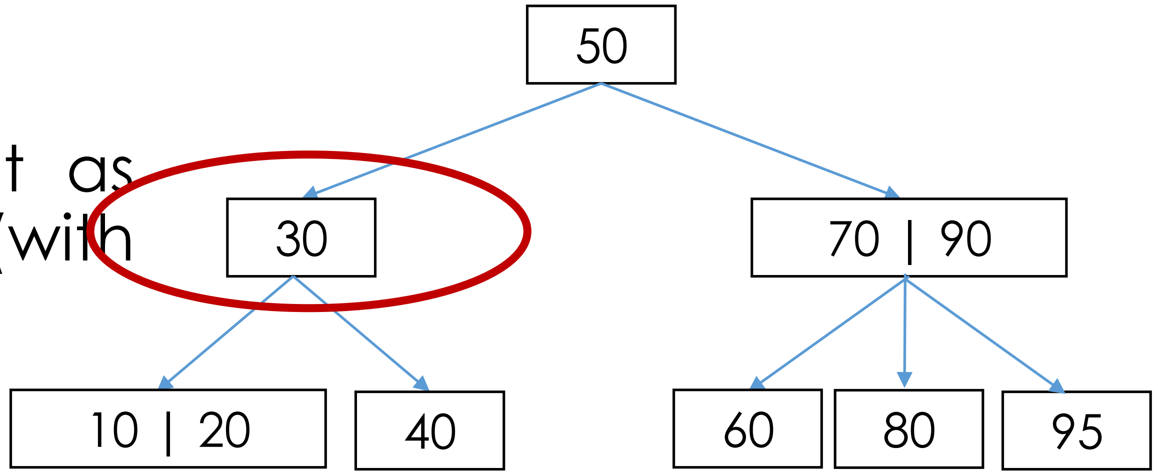
- To traverse a 2-3 Tree
  - Perform the analogue of an in-order traversal
    - Leftmost subtree,
    - Left value,
    - Center subtree,
    - Right value,
    - Rightmost subtree
- Searching a 2-3 tree is as efficient as searching the shorted binary tree (with the same values)
  - $O(\log_2 n)$



```
10 20
```

Traversing 2-3 Trees

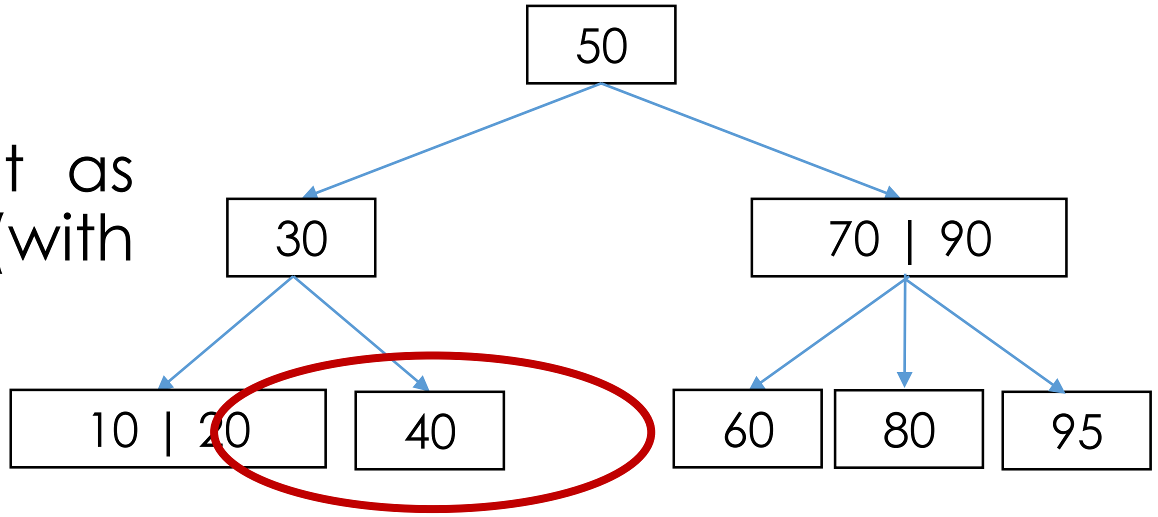
- To traverse a 2-3 Tree
  - Perform the analogue of an in-order traversal
    - Leftmost subtree,
    - Left value,
    - Center subtree,
    - Right value,
    - Rightmost subtree
- Searching a 2-3 tree is as efficient as searching the shorted binary tree (with the same values)
  - $O(\log_2 n)$



10 20 30

Traversing 2-3 Trees

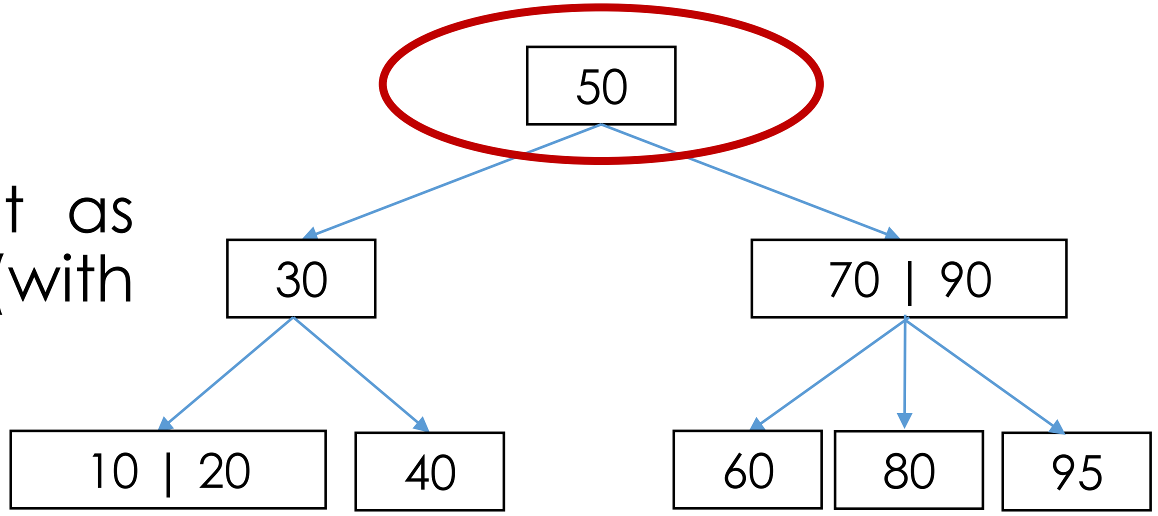
- To traverse a 2-3 Tree
  - Perform the analogue of an in-order traversal
    - Leftmost subtree,
    - Left value,
    - Center subtree,
    - Right value,
    - Rightmost subtree
- Searching a 2-3 tree is as efficient as searching the shorted binary tree (with the same values)
  - $O(\log_2 n)$



10 20 30 40

Traversing 2-3 Trees

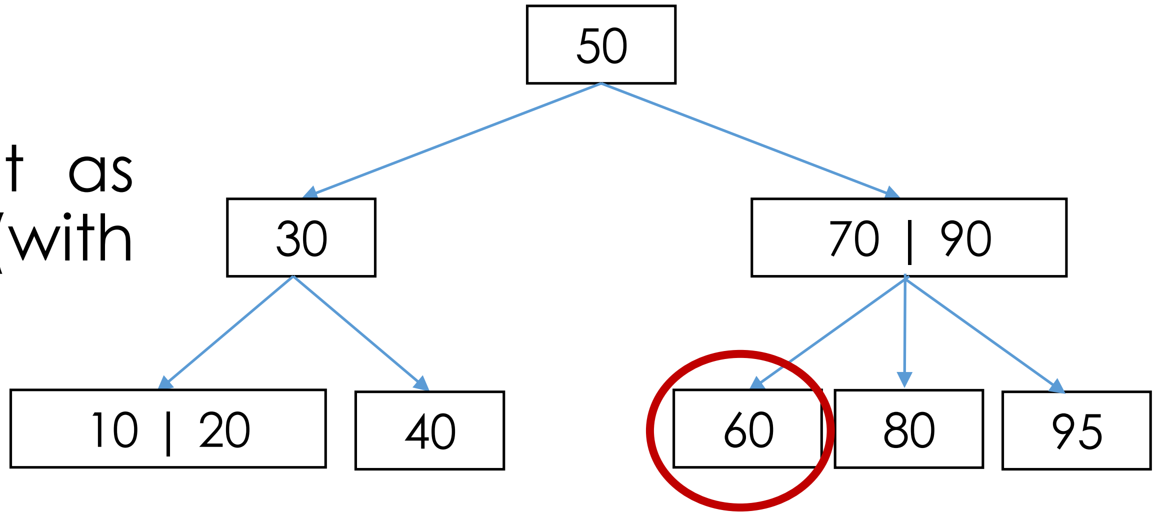
- To traverse a 2-3 Tree
  - Perform the analogue of an in-order traversal
    - Leftmost subtree,
    - Left value,
    - Center subtree,
    - Right value,
    - Rightmost subtree
- Searching a 2-3 tree is as efficient as searching the shorted binary tree (with the same values)
  - $O(\log_2 n)$



```
10 20 30 40 50
```

Traversing 2-3 Trees

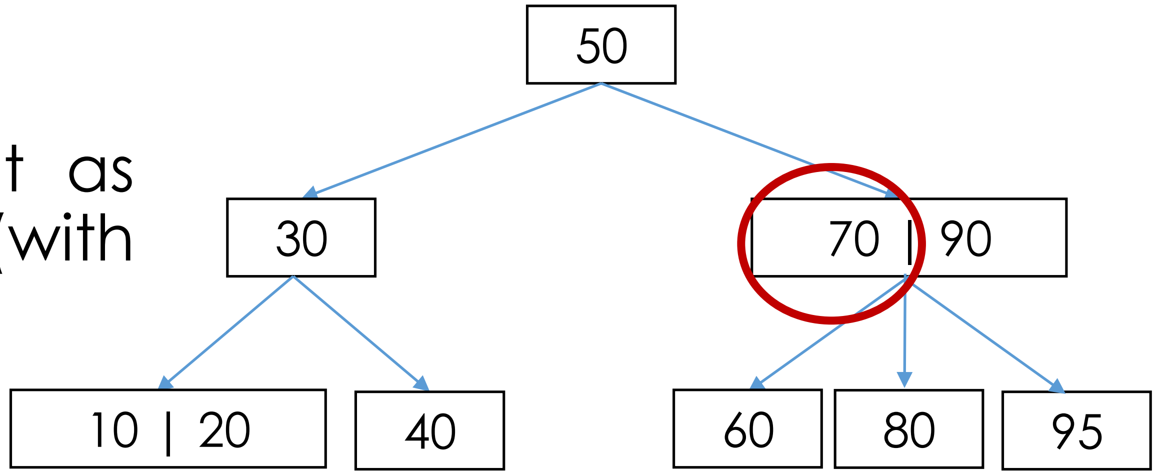
- To traverse a 2-3 Tree
  - Perform the analogue of an in-order traversal
    - Leftmost subtree,
    - Left value,
    - Center subtree,
    - Right value,
    - Rightmost subtree
- Searching a 2-3 tree is as efficient as searching the shorted binary tree (with the same values)
  - $O(\log_2 n)$



10 20 30 40 50 60

Traversing 2-3 Trees

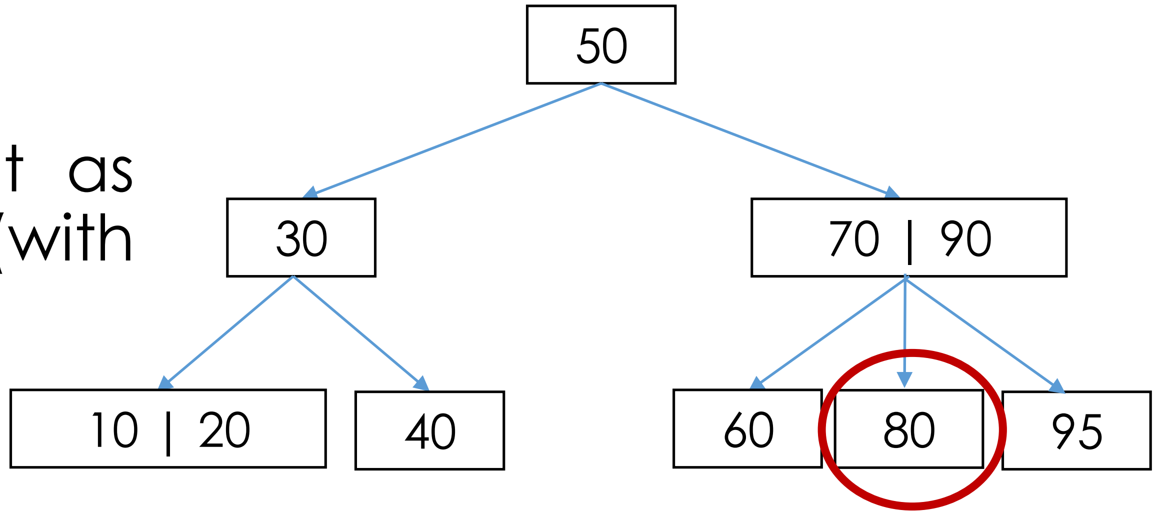
- To traverse a 2-3 Tree
  - Perform the analogue of an in-order traversal
    - Leftmost subtree,
    - Left value,
    - Center subtree,
    - Right value,
    - Rightmost subtree
- Searching a 2-3 tree is as efficient as searching the shorted binary tree (with the same values)
  - $O(\log_2 n)$



10 20 30 40 50 60 70

Traversing 2-3 Trees

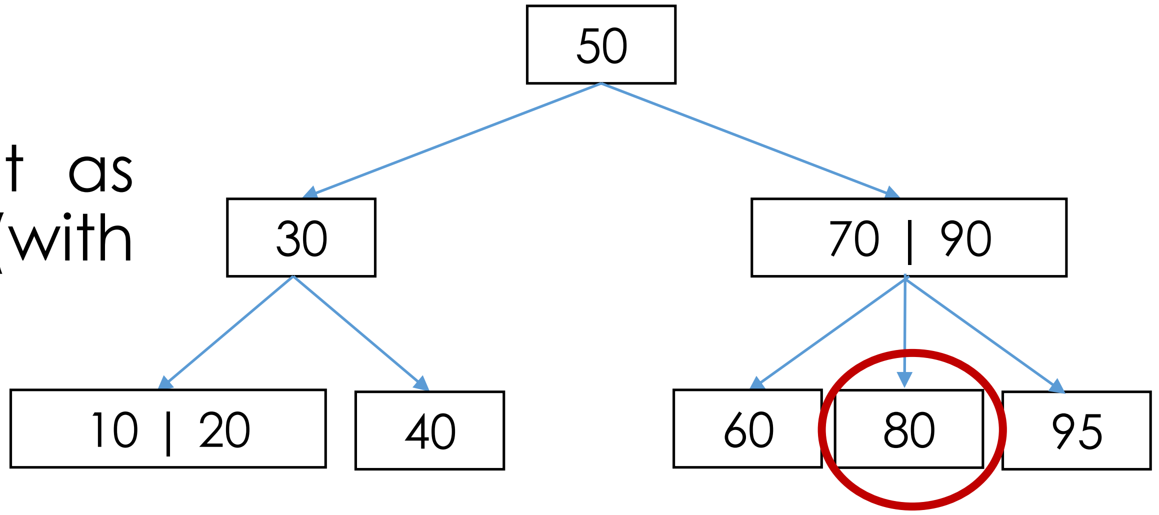
- To traverse a 2-3 Tree
  - Perform the analogue of an in-order traversal
    - Leftmost subtree,
    - Left value,
    - Center subtree,
    - Right value,
    - Rightmost subtree
- Searching a 2-3 tree is as efficient as searching the shorted binary tree (with the same values)
  - $O(\log_2 n)$



10 20 30 40 50 60 70

Traversing 2-3 Trees

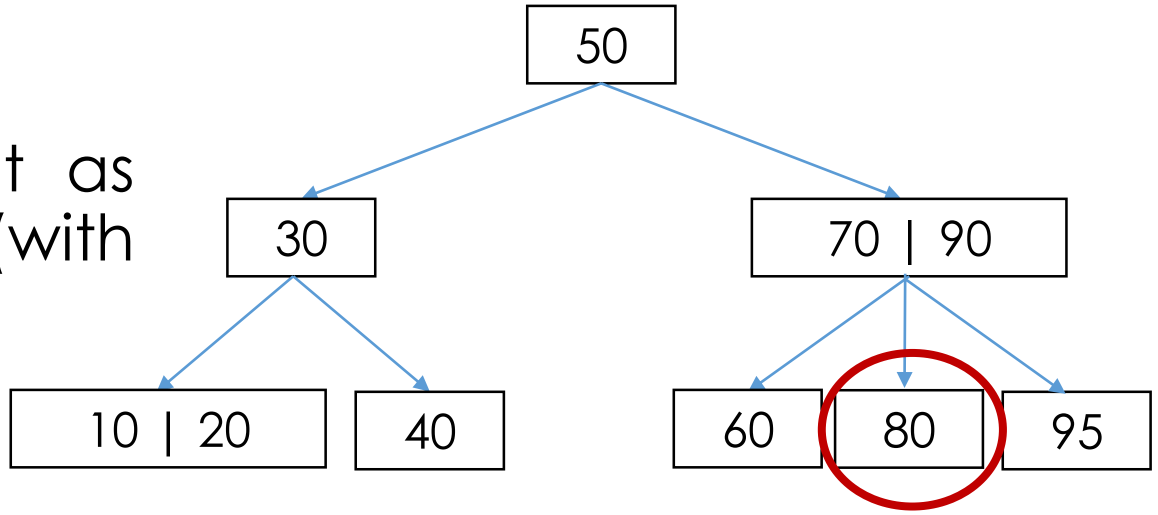
- To traverse a 2-3 Tree
  - Perform the analogue of an in-order traversal
    - Leftmost subtree,
    - Left value,
    - Center subtree,
    - Right value,
    - Rightmost subtree
- Searching a 2-3 tree is as efficient as searching the shorted binary tree (with the same values)
  - $O(\log_2 n)$



```
10 20 30 40 50 60 70 80
```

Traversing 2-3 Trees

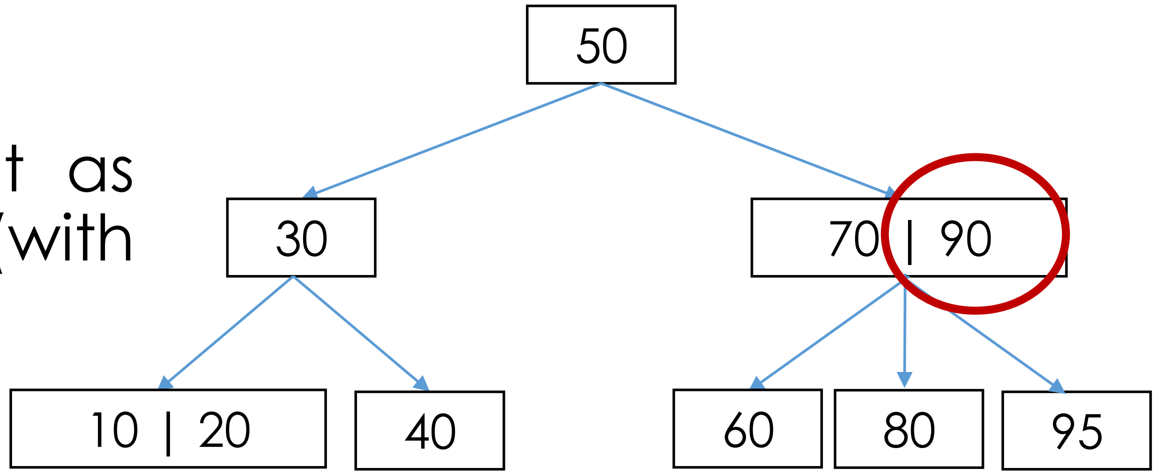
- To traverse a 2-3 Tree
  - Perform the analogue of an in-order traversal
    - Leftmost subtree,
    - Left value,
    - Center subtree,
    - Right value,
    - Rightmost subtree
- Searching a 2-3 tree is as efficient as searching the shorted binary tree (with the same values)
  - $O(\log_2 n)$



10 20 30 40 50 60 70 80

Traversing 2-3 Trees

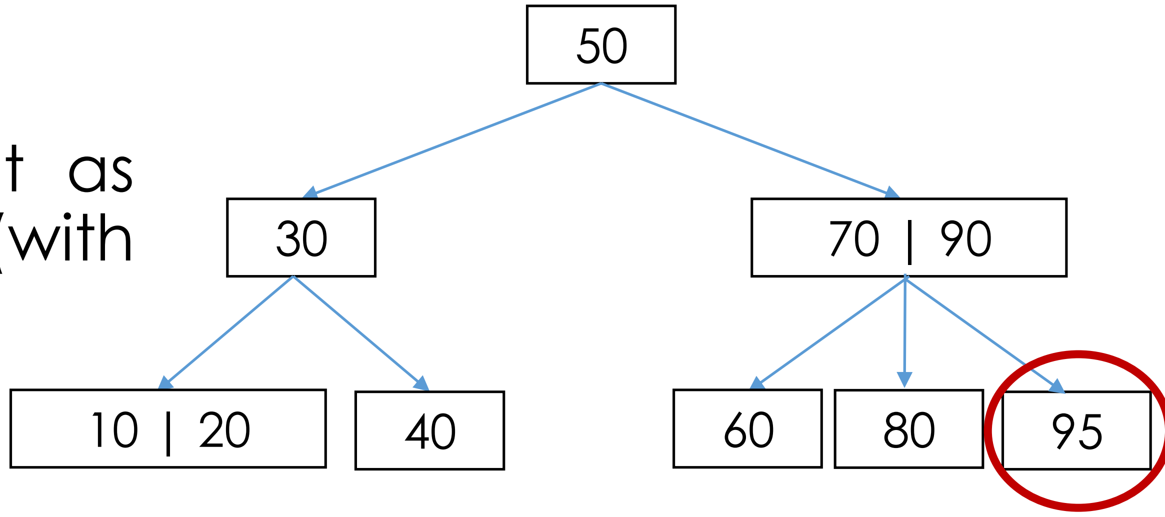
- To traverse a 2-3 Tree
  - Perform the analogue of an in-order traversal
    - Leftmost subtree,
    - Left value,
    - Center subtree,
    - Right value,
    - Rightmost subtree
- Searching a 2-3 tree is as efficient as searching the shorted binary tree (with the same values)
  - $O(\log_2 n)$



10 20 30 40 50 60 70 80 90

Traversing 2-3 Trees

- To traverse a 2-3 Tree
  - Perform the analogue of an in-order traversal
    - Leftmost subtree,
    - Left value,
    - Center subtree,
    - Right value,
    - Rightmost subtree
- Searching a 2-3 tree is as efficient as searching the shorted binary tree (with the same values)
  - $O(\log_2 n)$



10 20 30 40 50 60 70 80 90 95

Adding to 2-3 Trees

- Inserting values into a 2-3 Tree
  - Always insert values into an EXISTING leaf

Adding to 2-3 Trees

- Inserting values into a 2-3 Tree
  - Always insert values into an EXISTING leaf
  - Only exception – root (first value)
    - Create a 2-node and insert value

Adding to 2-3 Trees

- Inserting values into a 2-3 Tree
  - Always insert values into an EXISTING leaf
  - Only exception – root (first value)
    - Create a 2-node and insert value
  - Inserting a value into a 2-node turns it into a 3-node

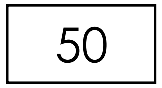
Adding to 2-3 Trees

- Inserting values into a 2-3 Tree
  - Always insert values into an EXISTING leaf
  - Only exception – root (first value)
    - Create a 2-node and insert value
  - Inserting a value into a 2-node turns it into a 3-node
  - Inserting a value into a 3-node causes it to divide
    - Result is a subtree of 2-nodes

Adding to 2-3 Trees

50 70 90

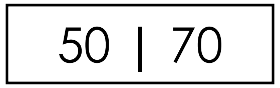
- Inserting values into a 2-3 Tree
  - Always insert values into an EXISTING leaf
  - Only exception – root (first value)
    - Create a 2-node and insert value
  - Inserting a value into a 2-node turns it into a 3-node
  - Inserting a value into a 3-node causes it to divide
    - Result is a subtree of 2-nodes



Adding to 2-3 Trees

- Inserting values into a 2-3 Tree
  - Always insert values into an EXISTING leaf
  - Only exception – root (first value)
    - Create a 2-node and insert value
  - Inserting a value into a 2-node turns it into a 3-node
  - Inserting a value into a 3-node causes it to divide
    - Result is a subtree of 2-nodes

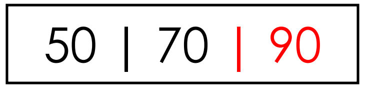
50 70 90



Adding to 2-3 Trees

50 70 90

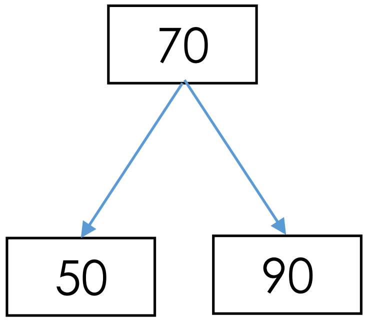
- Inserting values into a 2-3 Tree
  - Always insert values into an EXISTING leaf
  - Only exception – root (first value)
    - Create a 2-node and insert value
  - Inserting a value into a 2-node turns it into a 3-node
  - Inserting a value into a 3-node causes it to divide
    - Result is a subtree of 2-nodes



Adding to 2-3 Trees

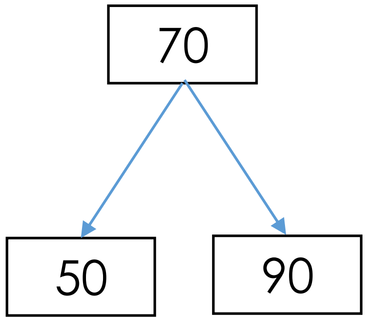
- Inserting values into a 2-3 Tree
  - Always insert values into an EXISTING leaf
  - Only exception – root (first value)
    - Create a 2-node and insert value
  - Inserting a value into a 2-node turns it into a 3-node
  - Inserting a value into a 3-node causes it to divide
    - Result is a subtree of 2-nodes

50 70 90



Adding to 2-3 Trees

50 70 90



- Inserting values into a 2-3 Tree
  - Always insert values into an EXISTING leaf
  - Only exception – root (first value)
    - Create a 2-node and insert value
  - Inserting a value into a 2-node turns it into a 3-node
  - Inserting a value into a 3-node causes it to divide
    - Result is a subtree of 2-nodes
- To insert an item with key k into a 2-3 tree
  - Locate the leaf at which the search for k would terminate
  - Insert the new item k into the leaf
  - If the leaf now contains only two items – done
  - If the leaf now contains three items, split the leaf into two nodes and move the middle value into parent

Adding to 2-3 Trees

- When an internal node contains three items
  - Split the node into two nodes
  - Accommodate the node’s children
- When the root contains three items
  - Split the root into two nodes
  - Create a new root node
- Tree grows in height

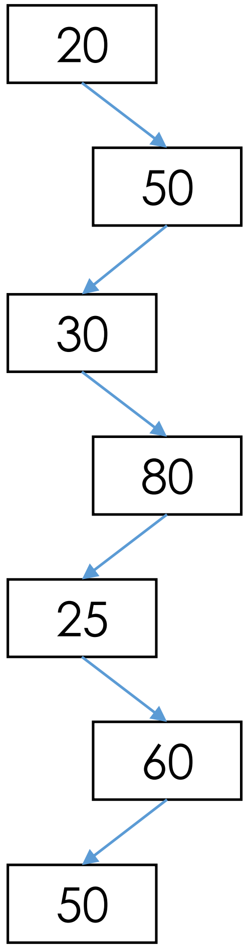
Adding to 2-3 Trees

- When an internal node contains three items
  - Split the node into two nodes
  - Accommodate the node’s children
- When the root contains three items
  - Split the root into two nodes
  - Create a new root node
- Tree grows in height

Adding to 2-3 Trees

```
Order of Values Given
20 90 30 80 25 60 50
```

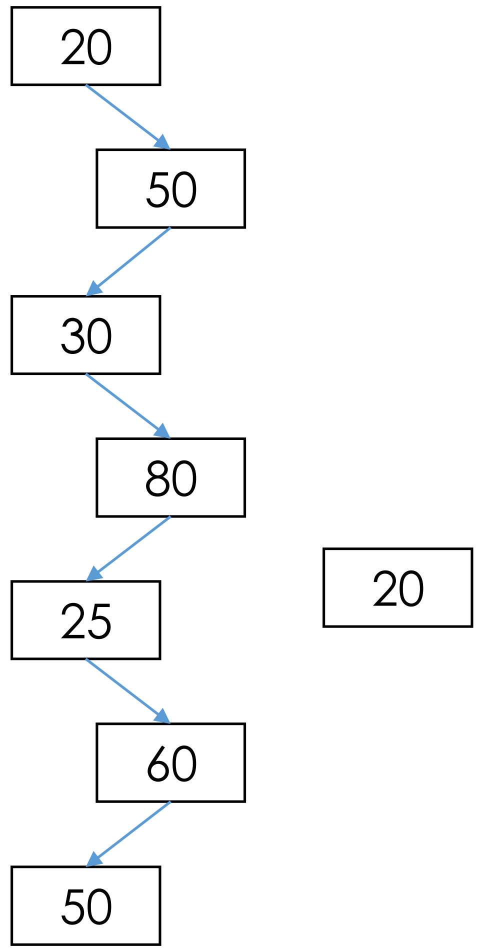
- When an internal node contains three items
  - Split the node into two nodes
  - Accommodate the node’s children
- When the root contains three items
  - Split the root into two nodes
  - Create a new root node
- Tree grows in height



Adding to 2-3 Trees

Order of Values Given
20 90 30 80 25 60 50

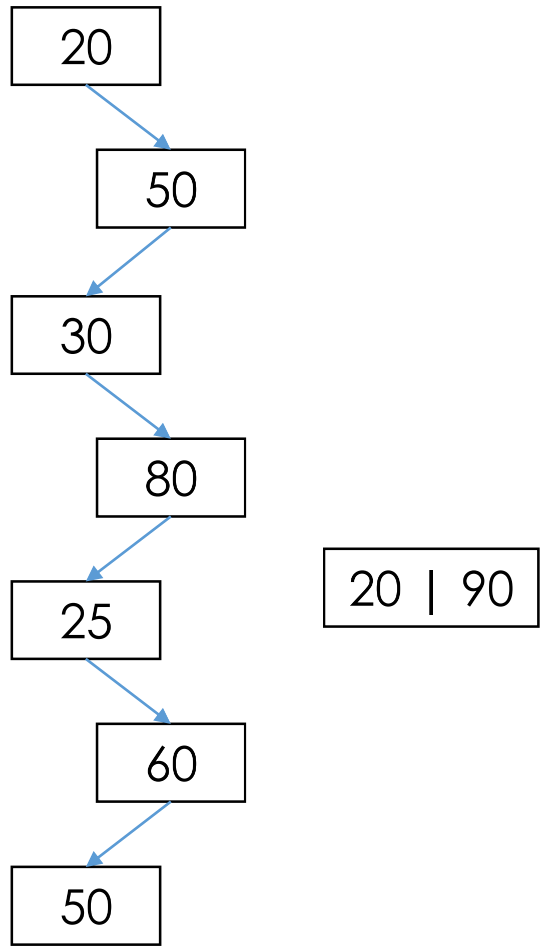
- When an internal node contains three items
  - Split the node into two nodes
  - Accommodate the node’s children
- When the root contains three items
  - Split the root into two nodes
  - Create a new root node
- Tree grows in height



Adding to 2-3 Trees

Order of Values Given
20 90 30 80 25 60 50

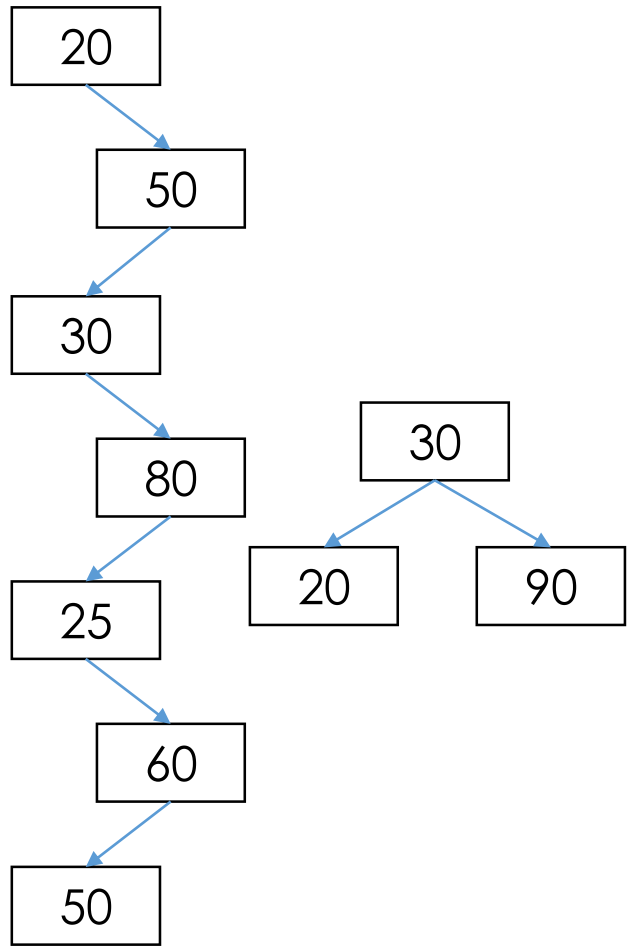
- When an internal node contains three items
  - Split the node into two nodes
  - Accommodate the node's children
- When the root contains three items
  - Split the root into two nodes
  - Create a new root node
- Tree grows in height



Adding to 2-3 Trees

Order of Values Given
20 90 30 80 25 60 50

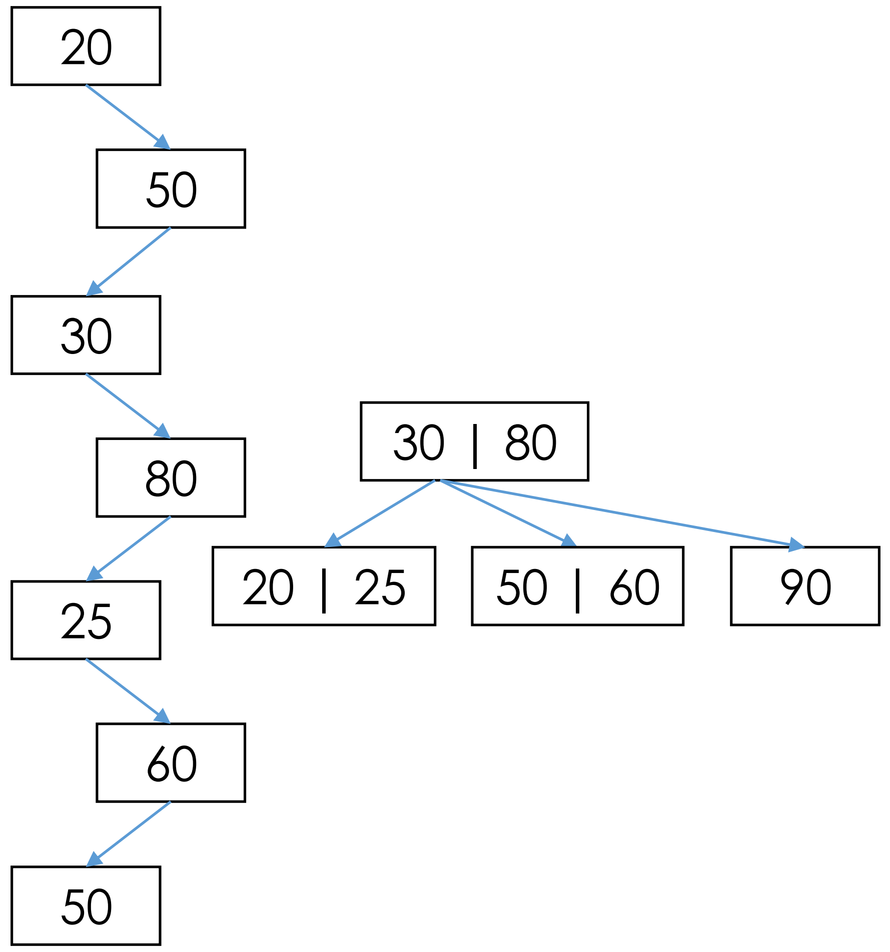
- When an internal node contains three items
  - Split the node into two nodes
  - Accommodate the node's children
- When the root contains three items
  - Split the root into two nodes
  - Create a new root node
- Tree grows in height



Adding to 2-3 Trees

Order of Values Given
20 90 30 80 25 60 50

- When an internal node contains three items
  - Split the node into two nodes
  - Accommodate the node's children
- When the root contains three items
  - Split the root into two nodes
  - Create a new root node
- Tree grows in height

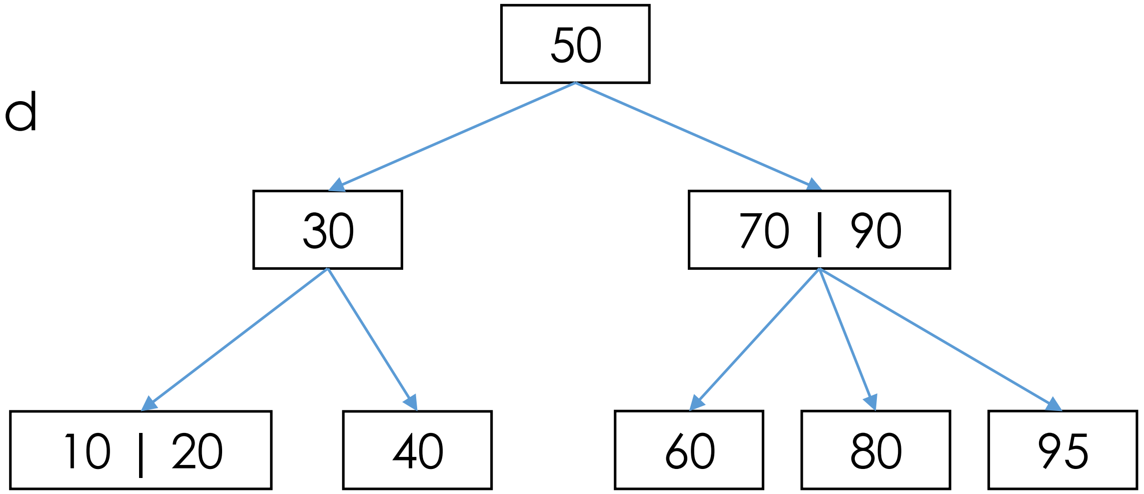


Removing From a 2-3 Tree

- Removing values from 2-3 trees
  - Always remove from a leaf
  - Values (and children) will be redistributed
  - Nodes can be merged
  - Only root node is deleted
    - And only if it is empty (contains no values)

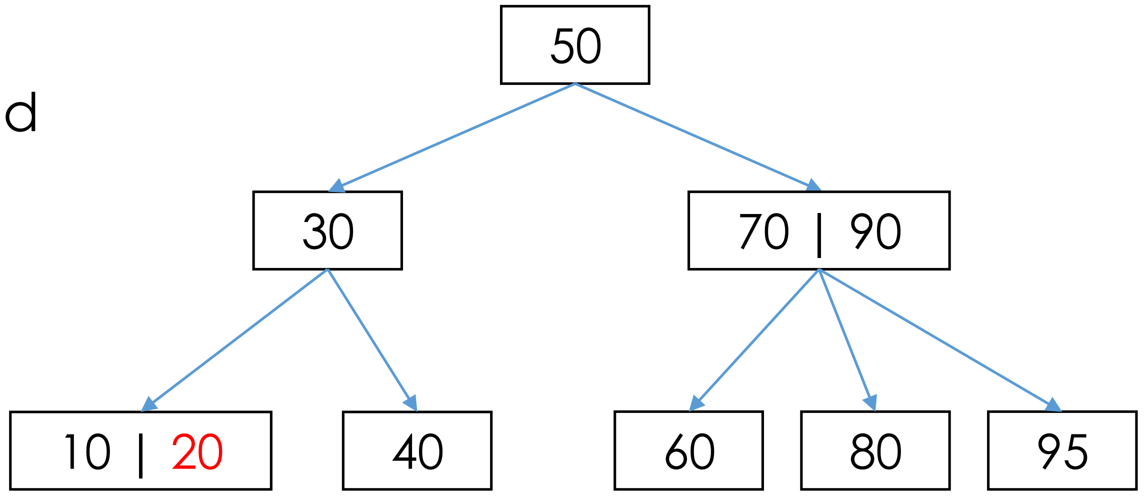
Removing From a 2-3 Tree

- Removing values from 2-3 trees
  - Always remove from a leaf
  - Values (and children) will be redistributed
  - Nodes can be merged
  - Only root node is deleted
    - And only if it is empty (contains no values)



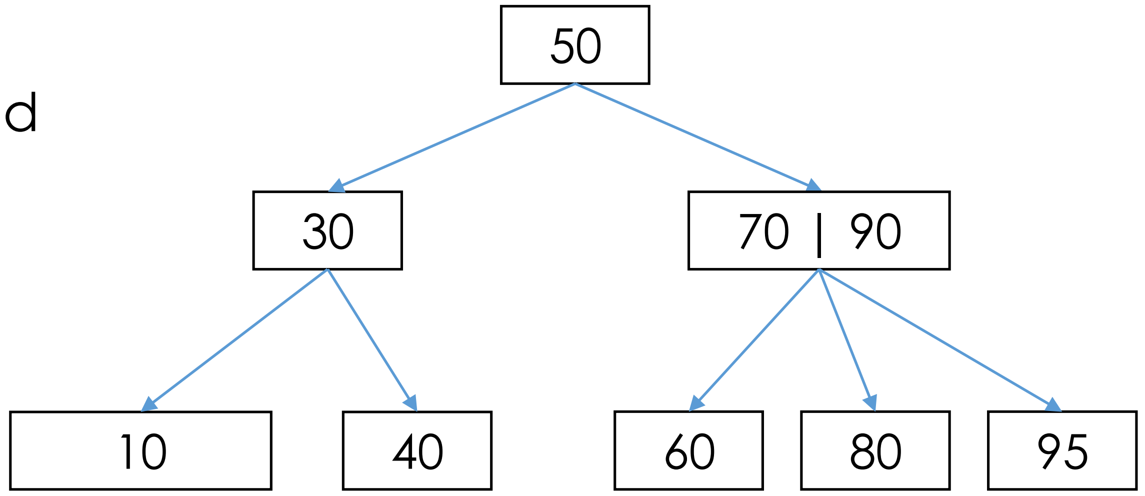
Removing From a 2-3 Tree

- Removing values from 2-3 trees
  - Always remove from a leaf
  - Values (and children) will be redistributed
  - Nodes can be merged
  - Only root node is deleted
    - And only if it is empty (contains no values)



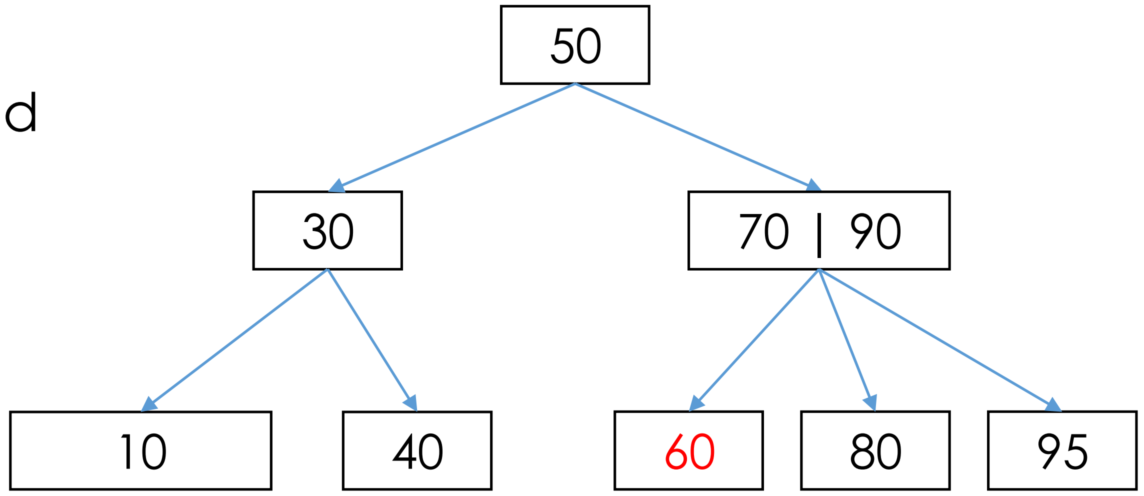
Removing From a 2-3 Tree

- Removing values from 2-3 trees
  - Always remove from a leaf
  - Values (and children) will be redistributed
  - Nodes can be merged
  - Only root node is deleted
    - And only if it is empty (contains no values)



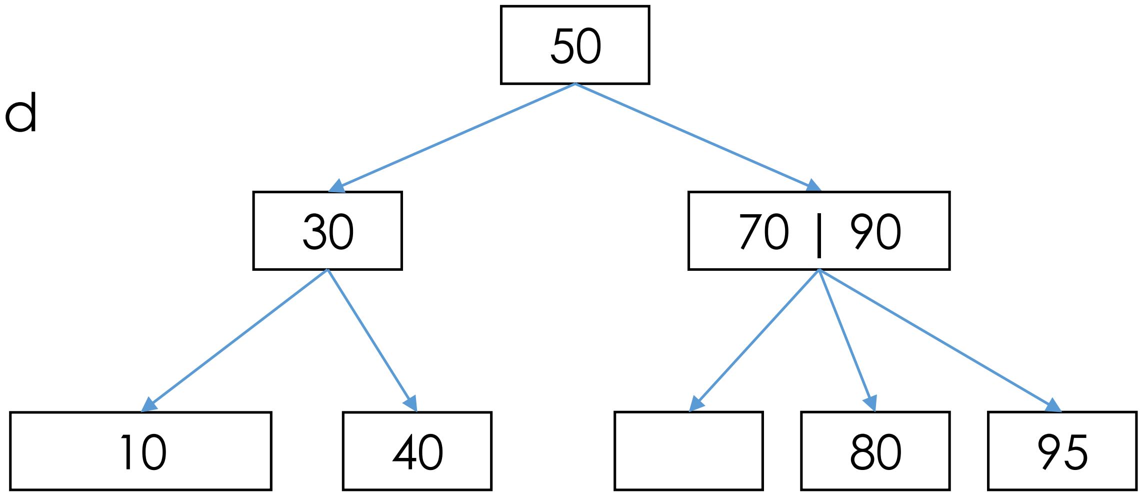
Removing From a 2-3 Tree

- Removing values from 2-3 trees
  - Always remove from a leaf
  - Values (and children) will be redistributed
  - Nodes can be merged
  - Only root node is deleted
    - And only if it is empty (contains no values)



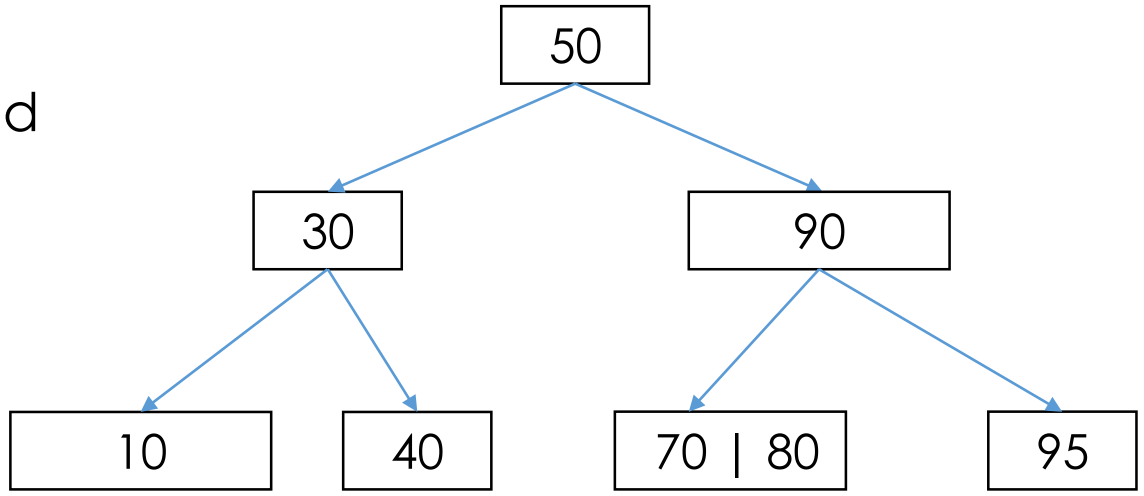
Removing From a 2-3 Tree

- Removing values from 2-3 trees
  - Always remove from a leaf
  - Values (and children) will be redistributed
  - Nodes can be merged
  - Only root node is deleted
    - And only if it is empty (contains no values)



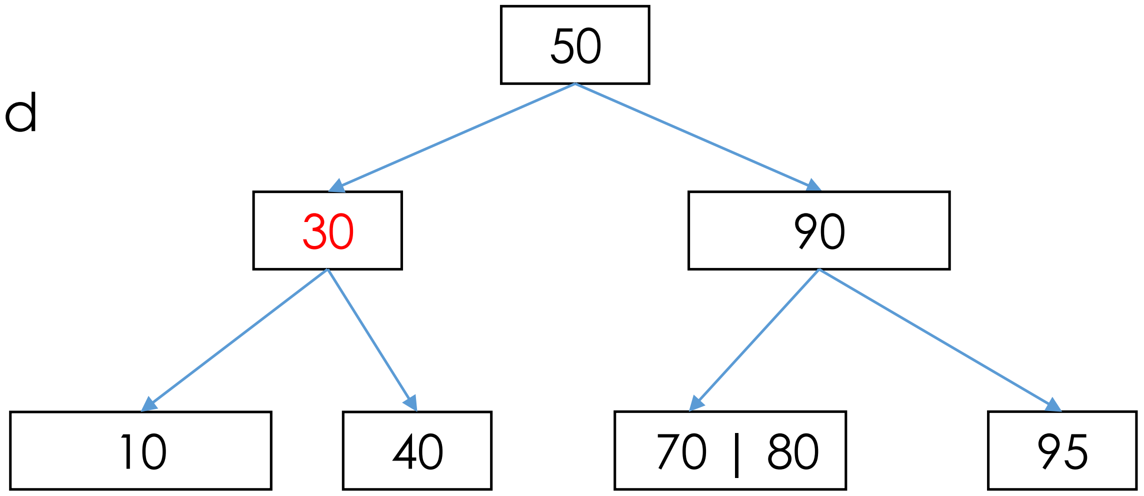
Removing From a 2-3 Tree

- Removing values from 2-3 trees
  - Always remove from a leaf
  - Values (and children) will be redistributed
  - Nodes can be merged
  - Only root node is deleted
    - And only if it is empty (contains no values)



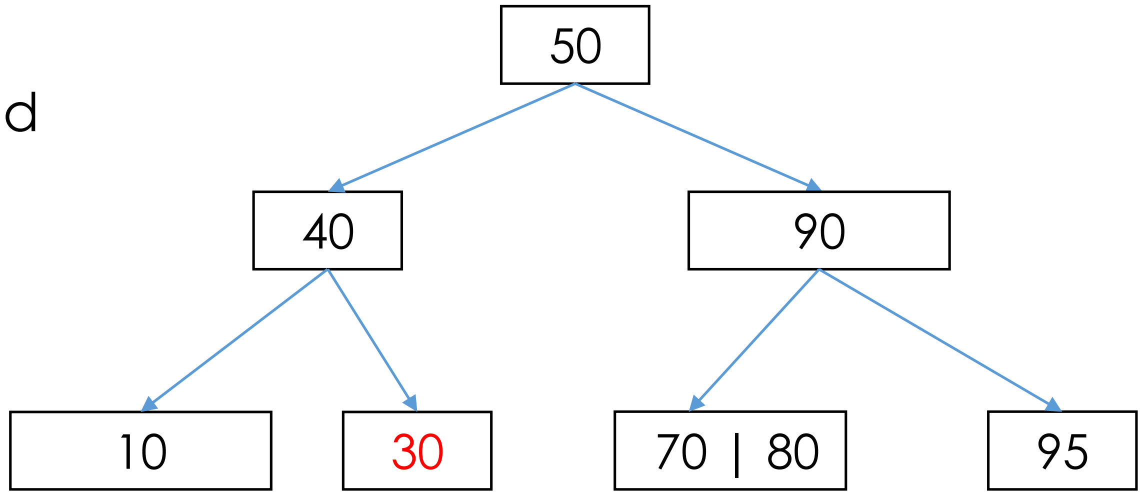
Removing From a 2-3 Tree

- Removing values from 2-3 trees
  - Always remove from a leaf
  - Values (and children) will be redistributed
  - Nodes can be merged
  - Only root node is deleted
    - And only if it is empty (contains no values)



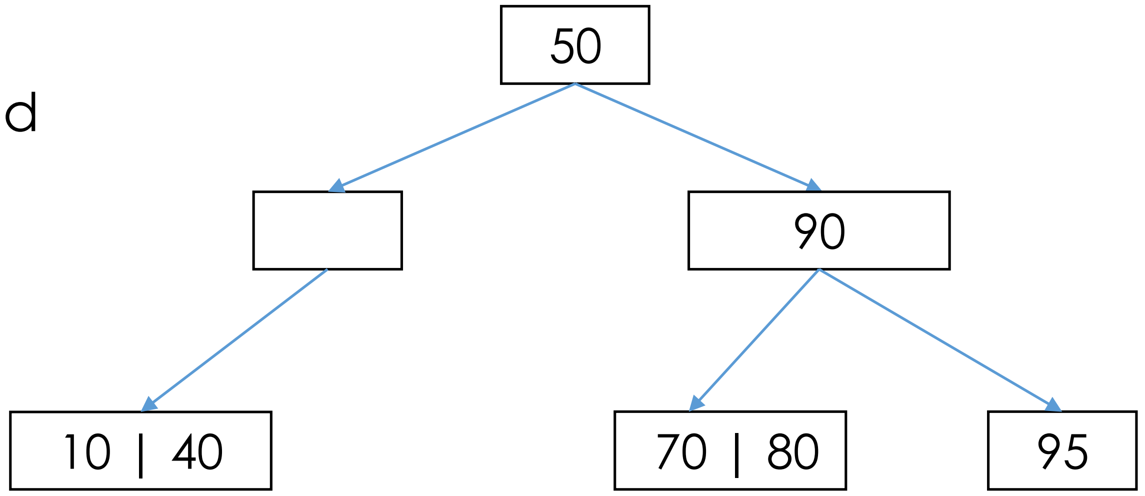
Removing From a 2-3 Tree

- Removing values from 2-3 trees
  - Always remove from a leaf
  - Values (and children) will be redistributed
  - Nodes can be merged
  - Only root node is deleted
    - And only if it is empty (contains no values)



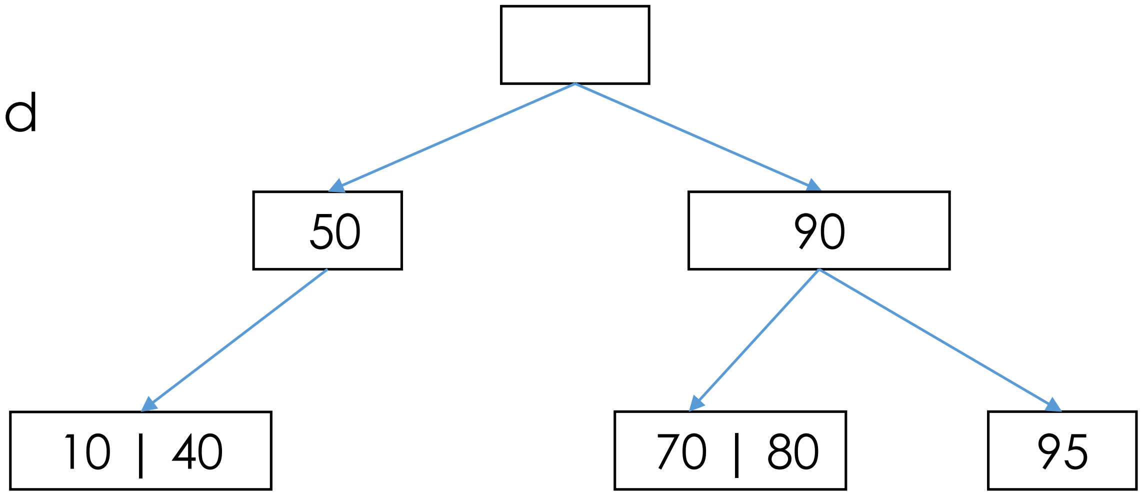
Removing From a 2-3 Tree

- Removing values from 2-3 trees
  - Always remove from a leaf
  - Values (and children) will be redistributed
  - Nodes can be merged
  - Only root node is deleted
    - And only if it is empty (contains no values)



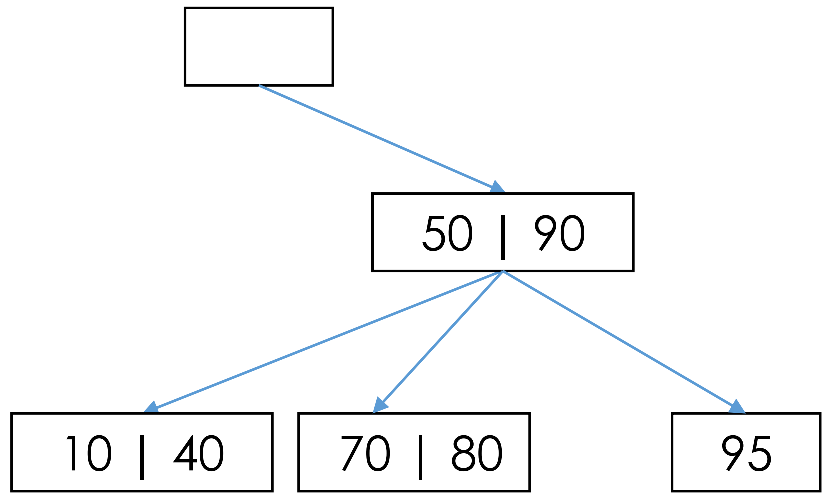
Removing From a 2-3 Tree

- Removing values from 2-3 trees
  - Always remove from a leaf
  - Values (and children) will be redistributed
  - Nodes can be merged
  - Only root node is deleted
    - And only if it is empty (contains no values)



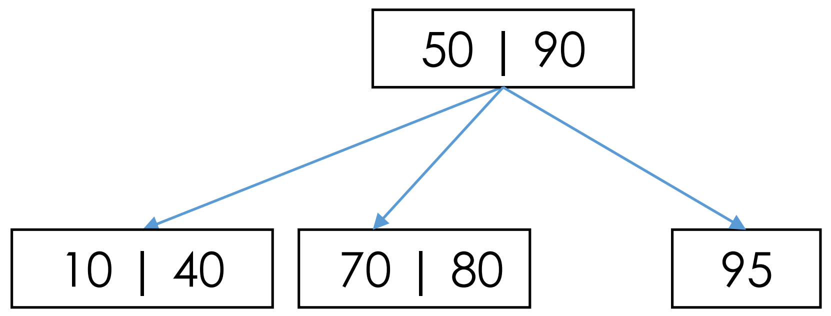
Removing From a 2-3 Tree

- Removing values from 2-3 trees
  - Always remove from a leaf
  - Values (and children) will be redistributed
  - Nodes can be merged
  - Only root node is deleted
    - And only if it is empty (contains no values)



Removing From a 2-3 Tree

- Removing values from 2-3 trees
  - Always remove from a leaf
  - Values (and children) will be redistributed
  - Nodes can be merged
  - Only root node is deleted
    - And only if it is empty (contains no values)



Thank you