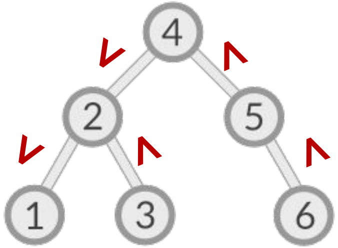
Fu-Yin Cherng

National Taiwan University

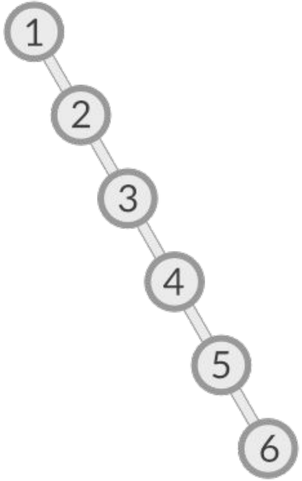
Balanced Search Tree

Review of Balanced Search Tree

- Week 10 - tree
- Binary Tree
  - height, level
- Balanced Binary Tree
  - complete binary tree
- Binary Search Tree
  - The more balanced tree, more efficient



balanced binary search tree
height: 3

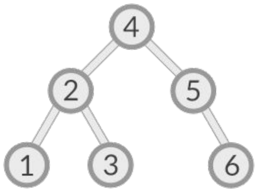


imbalanced binary search tree
height: 6

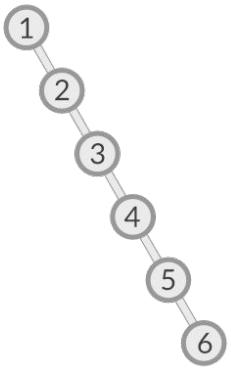
Efficiency of binary tree

- related to the tree’s height
  - maximum number of comparisons is equal to tree’ height
- tree’s height is sensitive to order of insertion and removal

insert *1,2,3,4,5,6* into binary tree



insert in the order:
4,2,5,1,3,6
(minimum height)



insert in the order:
1,2,3,4,5,6
(maximum height)

Efficiency of binary tree

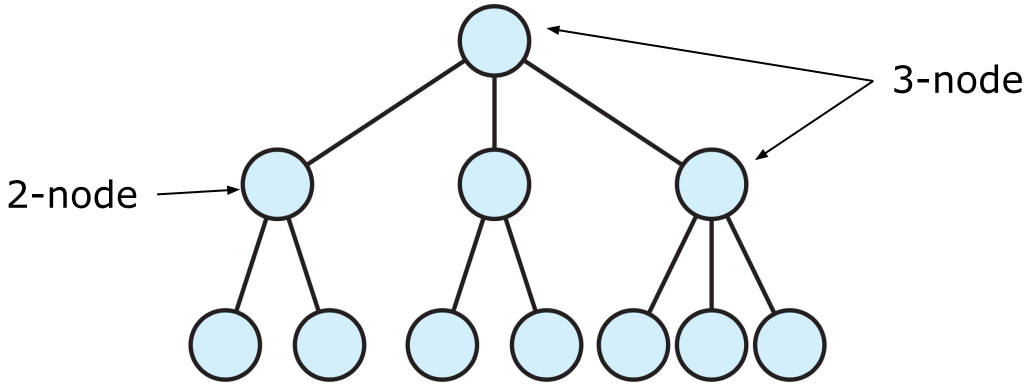
- For binary search tree, insertions and removals can cause the tree to lose its balance and approach a linear shape
  - loose the advantage of binary tree
  - no better than a linear chain of linked nodes
- Use variation of basic binary search tree
  - retain their balance despite insertions and removals
  - easier to maintain than a minimum-height binary search tree
- Introduce some populer variations
  - assume that there are no duplicates in tree

Outline

- 2-3 Trees
- 2-3-4 Trees
- Red-Black Trees
- AVL Trees

2-3 Trees

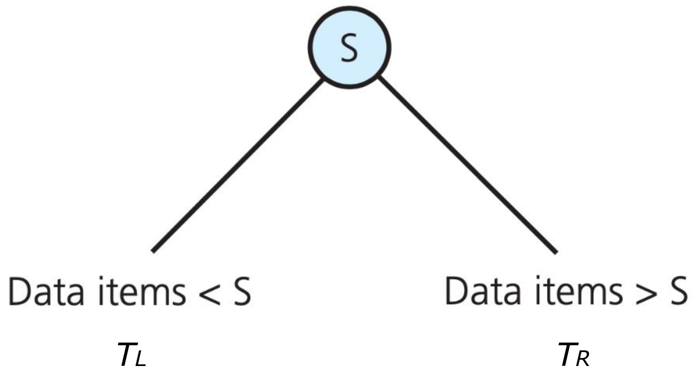
- each internal node (nonleaf) has either two or three children
- all leaves are at the same level



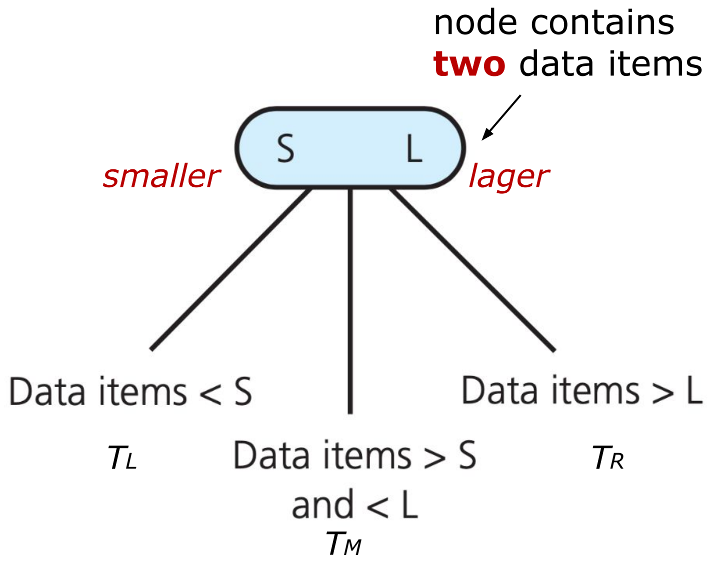
2-3 Trees

- not a binary tree, but
  - if a 2-3 tree only contains 2-nodes -> a full binary tree
- a 2-3 tree of height *h* always has at least as many nodes as a full binary tree of height *h*
- A 2-3 tree is never taller than a minimum-height binary tree with same number of nodes
  - minimun height: $\lceil \log_2(n + 1) \rceil$ (n: node numbers)

2-3 (Search) Trees



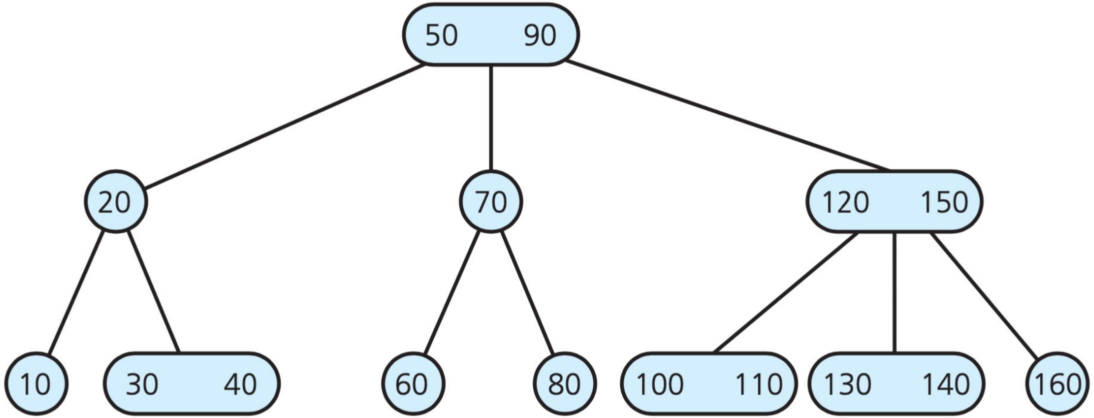
a 2-node



a 3-node

2-3 (Search) Trees

- A leaf may contain either 1 or 2 data items
- Items in a 2-3 tree are ordered

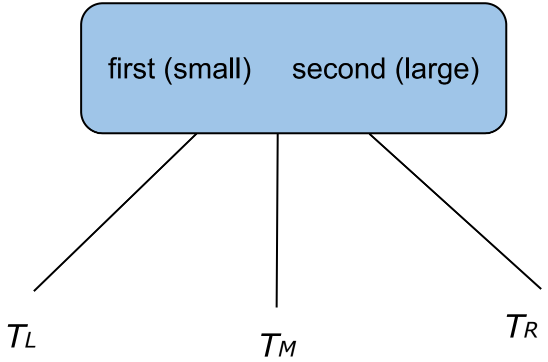


Traversing a 2-3 tree

- inorder traversal (assume no empty tree)

```
inorder(23Tree: TwoThreeTree): void

    if (23Tree's root node r is a leaf )
        Visit the data item(s)
    else if (r has two data items) { //3-nodes
        inorder(left subtree of 23Tree's root)
        Visit the first data item
        inorder(middle subtree of 23Tree's root)
        Visit the second data item
        inorder(right subtree of 23Tree's root) }
    else //r has one data item, 2-nodes
    {
        inorder(left subtree of 23Tree's root)
        Visit the data item
        inorder(right subtree of 23Tree's root)
}
```



Searching a 2-3 Tree

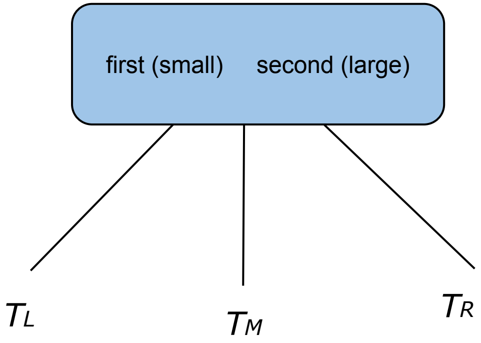
```
findItem(23Tree: TwoThreeTree, target: ItemType): ItemType

     if (target is in 23Tree’s root node r) {
          // The item has been found
          treeItem = the data portion of r
          return treeItem // Success
     }
     else if (r is a leaf)//r is not rarget
          throw NotFoundException // Failure
```

Searching a 2-3 Tree

```
findItem(23Tree: TwoThreeTree, target: ItemType): ItemType

     …
     // Else search the appropriate subtree
     else if (r has two data items) //3-node
     {
          if (target < smaller item in r)
               return findItem(r's left subtree, target)
          else if (target < larger item in r)
               return findItem(r's middle subtree, target)
          else
               return findItem(r's right subtree, target)
     }
     else{ // r has one data item, 2-node
          if (target < r's data item)
               return findItem(r's left subtree, target)
          else
               return findItem(r's right subtree, target)
     }
```



Searching a 2-3 Tree

- efficiency of searching a 2-3 tree ~= searching the shortest binary search tree
  - A binary search tree with n nodes cannot be shorter than $\lceil \log 2(n + 1) \rceil$ (minimum height/length)
  - A 2-3 tree with n nodes cannot be taller than $\lceil \log_2(n + 1) \rceil$
- So the height of a 2-3 tree is shorter than the shortest possible binary search tree, given the same number of node

Searching a 2-3 Tree

- But, you need to compare two values instead of one in a 2-3 tree
- So, the comparison number of searching 2-3 tree ~= comparison number of searching a balanced binary tree.
- Then, why use a 2-3 tree?

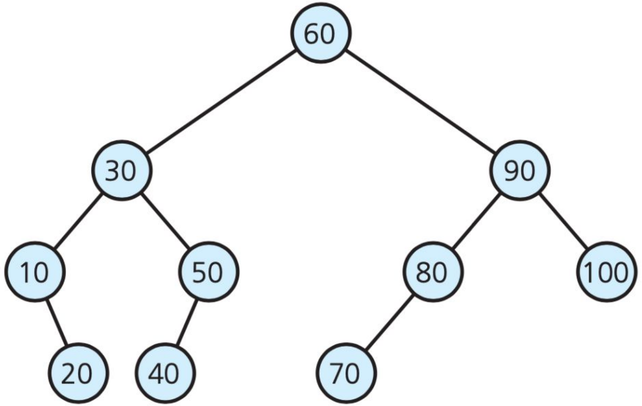
Advantage of using 2-3 tree compared to binary tree

- maintaining balance of a binary search tree is difficult
  - in the face of insertion and removal operations
- maintaining balance of a 2-3 tree is relatively simple

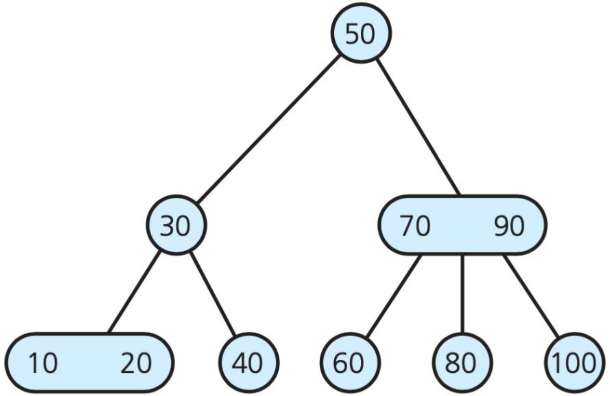
Advantage of using 2-3 tree compared to binary tree: example

binary search tree and 2-3 tree with the same data item

balanced binary tree, so with same efficiency with 2-3 tree when searching



Binary search tree

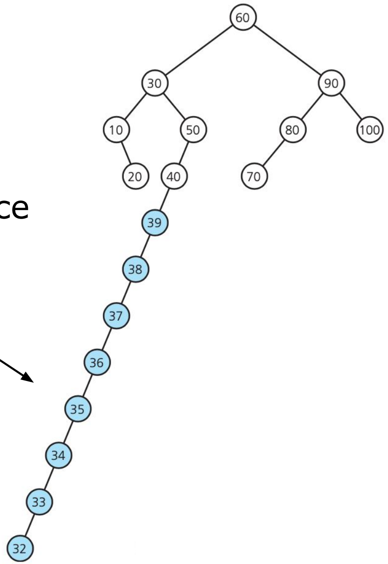


2-3 tree

Advantage of using 2-3 tree compared to binary tree: example

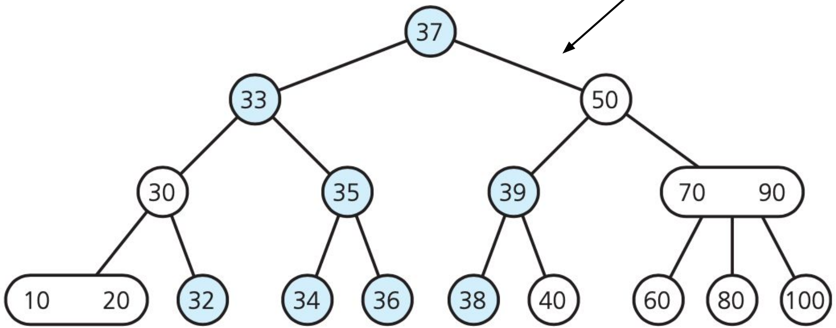
Perform insertion of 39-32 in order

break the balance structure



Binary search tree

retain the structure



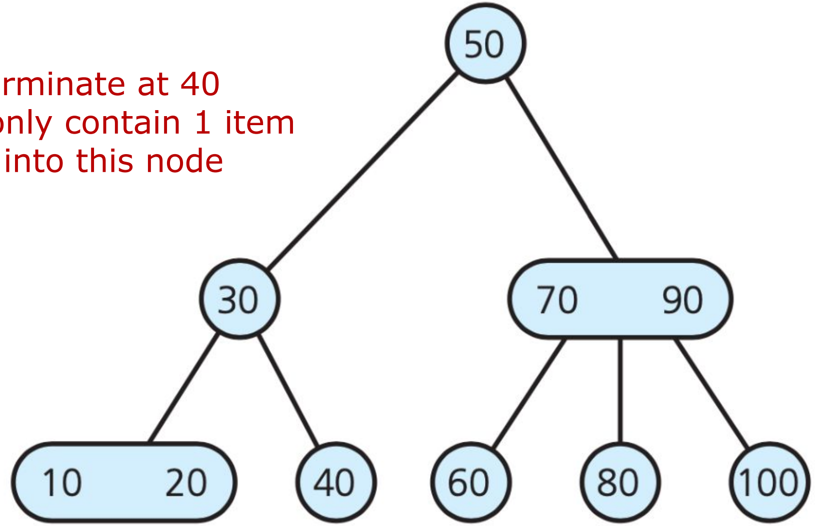
2-3 tree

Insertion to a 2-3 tree: example

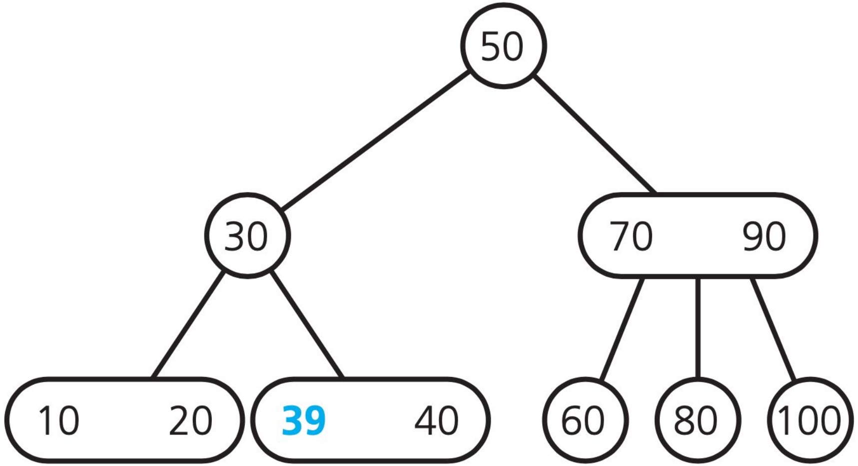
Show the sequence of insertion 39-32 to 2-3 tree

Inserting 39

- search for 39, terminate at 40
- since this node only contain 1 item
- simply insert 39 into this node



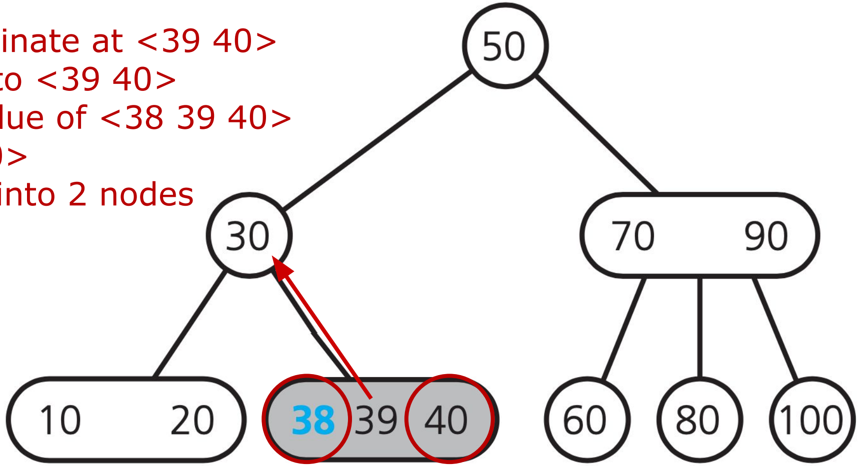
Insertion to a 2-3 tree: example



Insertion to a 2-3 tree: example

Inserting 38

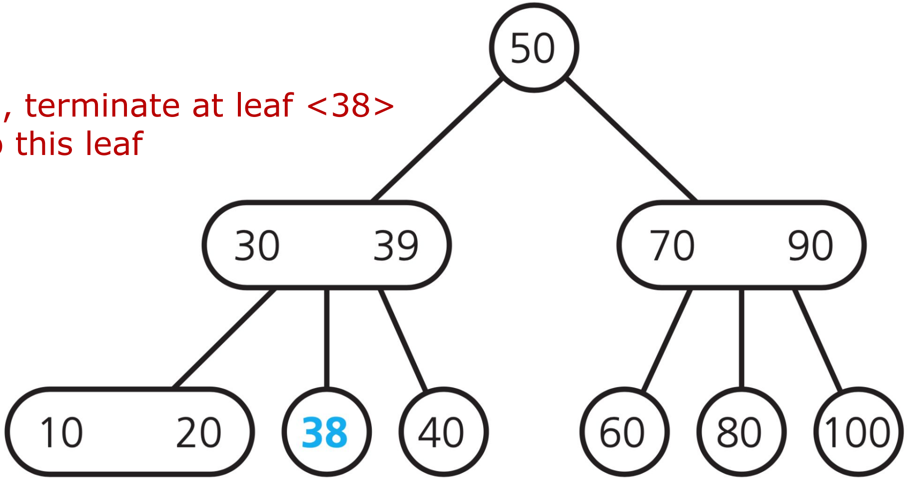
- search for 38, terminate at <39 40>
- cannot insert 38 into <39 40>
- move up middle value of <38 39 40> to parent node <30>
- separate <38 40> into 2 nodes



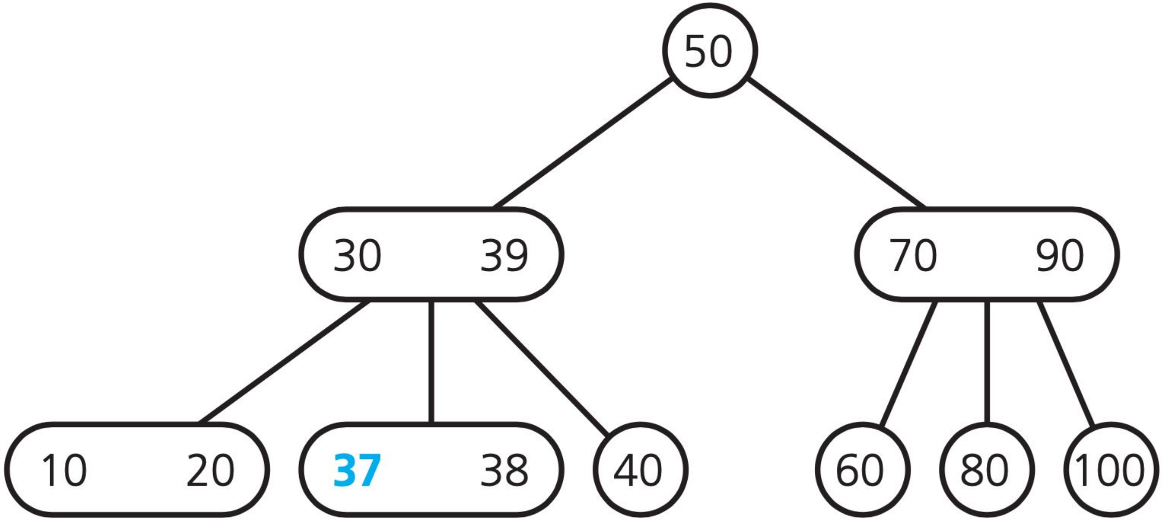
Insertion to a 2-3 tree: example

Inserting 37

- search for 37, terminate at leaf <38>
- insert 37 into this leaf



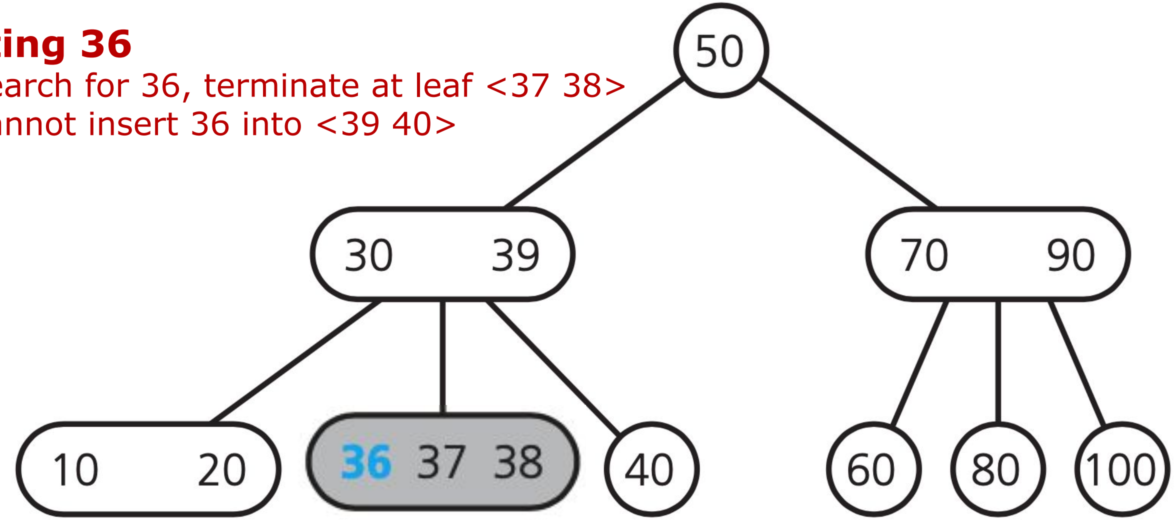
Insertion to a 2-3 tree: example



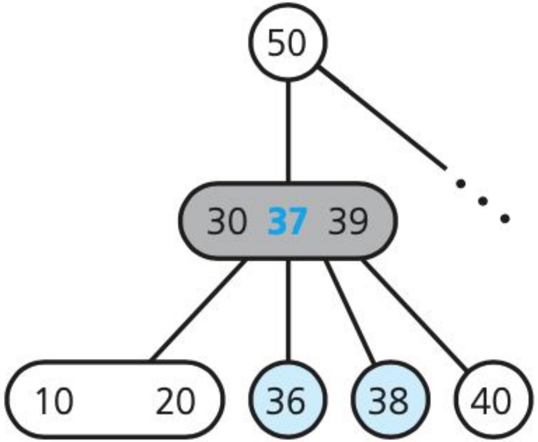
Insertion to a 2-3 tree: example

Inserting 36

- search for 36, terminate at leaf <37 38>
- cannot insert 36 into <39 40>

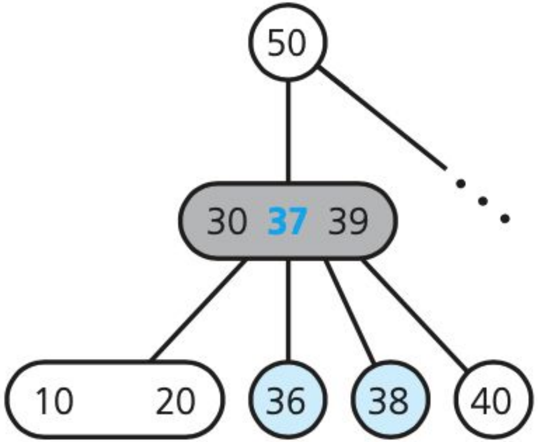


Insertion to a 2-3 tree: example

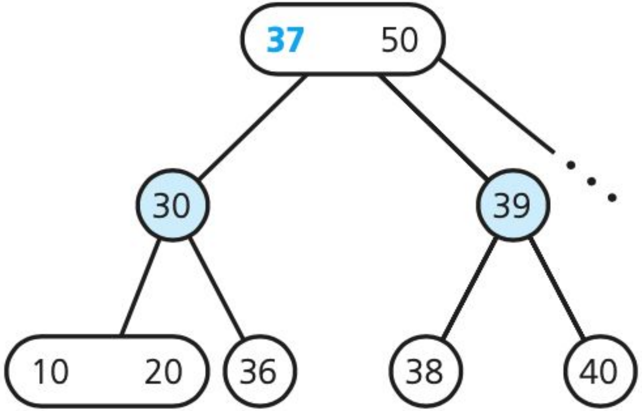


- move up middle value of <36 37 28> to parent node <30 39> & separate <36 38> into 2 nodes
- <30 39> cannot has 3 value & have 4 children!!

Insertion to a 2-3 tree: example



- move up middle value of <36 37 28> to parent node <30 39> & separate <36 38> into 2 nodes
- <30 39> cannot has 3 value & have 4 children!!

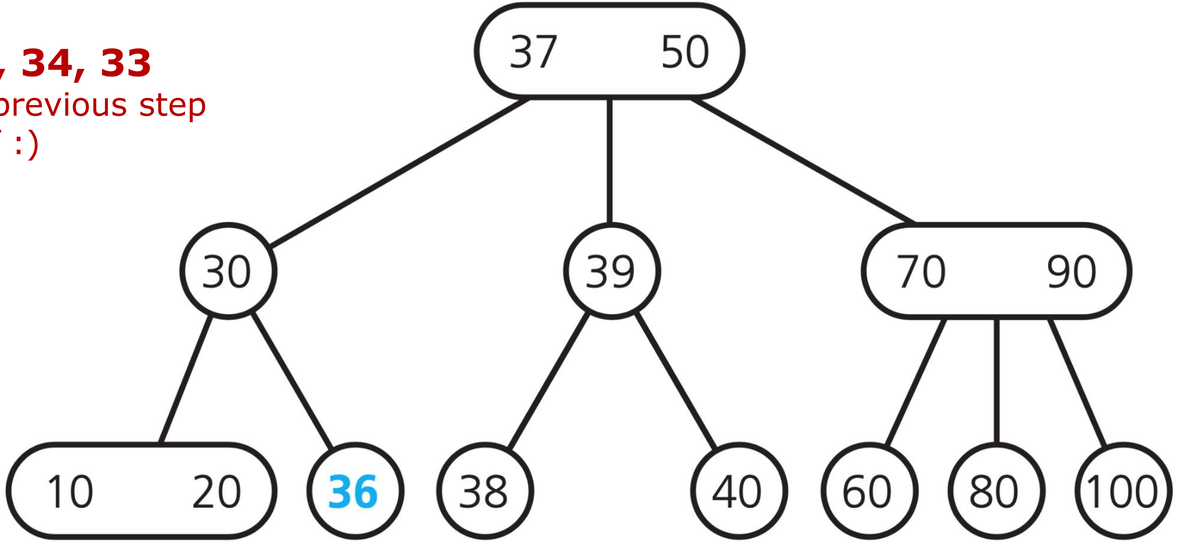


- similar, move the middle of <30 37 39> up to <50>
- separate <30 39>, what happen to their children nodes?
- attach left pair to smllest value <30>
- attach right pair to largest value <39>

Insertion to a 2-3 tree: example

Inserting 35, 34, 33

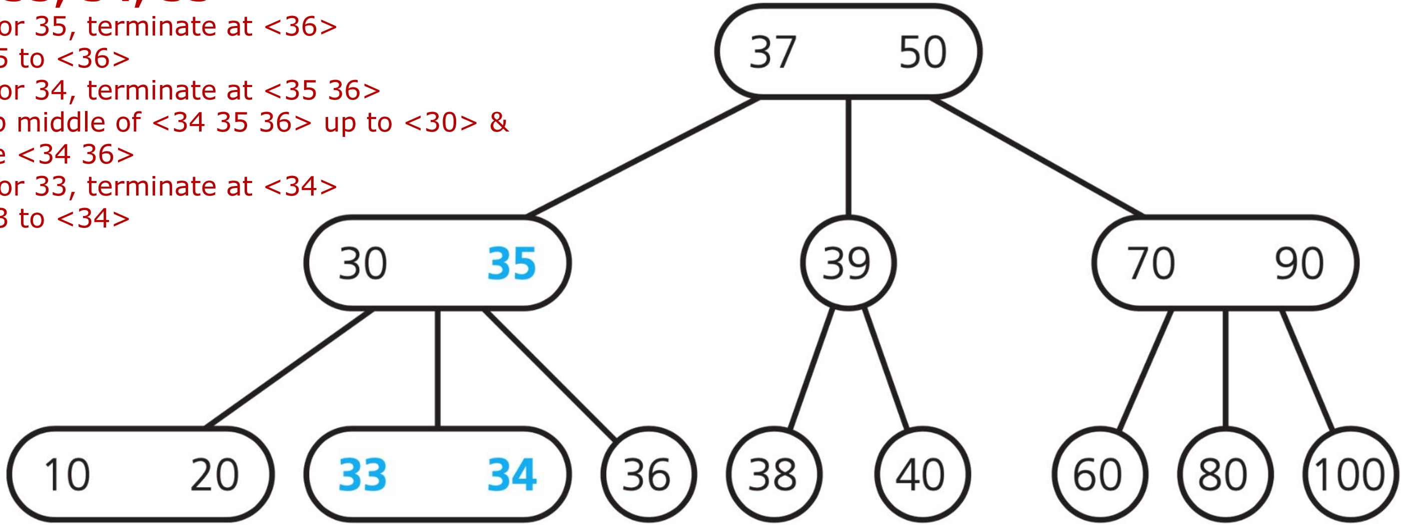
- similar to previous step
- try youself :)



Insertion to a 2-3 tree: example

Inserting 35, 34, 33

- search for 35, terminate at <36>
- insert 35 to <36>
- search for 34, terminate at <35 36>
- move up middle of <34 35 36> up to <30> & separate <34 36>
- search for 33, terminate at <34>
- insert 33 to <34>



Insertion to a 2-3 tree: general stratey

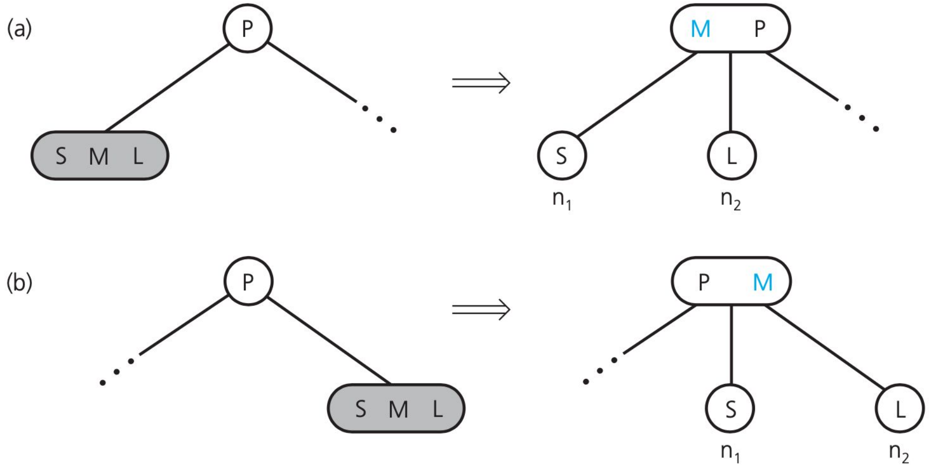
- 1. locate the leaf where the search for new item would terminate
- 2. if the leaf doesn’t have 2 items, insert the new item into the leaf *(inserting 39)*

Insertion to a 2-3 tree: general stratey

- 3. if the leaf already have 2 items, arrange the new item and the 2 items in order (<S M L>)
- 4. move up the Middle to parent node, split Smallest and Largest into 2 nodes

the leaf is left child

the leaf is right child



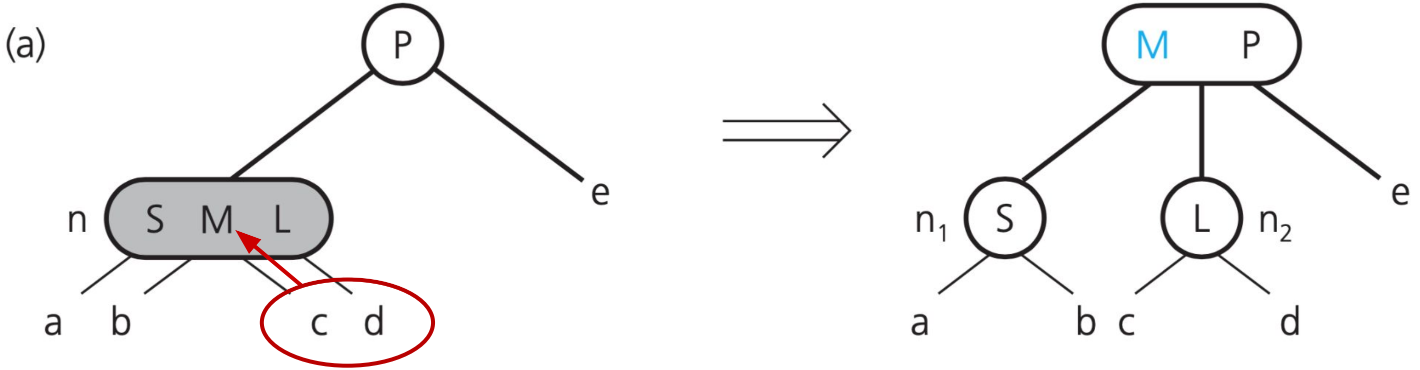
S to n1 (left node
L to n2 (right node

Insertion to a 2-3 tree: general stratey

- 5. if parent node already have two items, cannot accept the item moving up (Middle)
- 6. move up the Middle to the parent node of the parent (<P>)
- 7. split parent node (<S L>) into 2 nodes

Insertion to a 2-3 tree: general stratey

- 8. take care the 4 children node
  - create 1 more child node when move up M from the leaf
- 9. two leftmost children nodes to n_1 (<S>), 2 rightmost to n_2 (<L>)

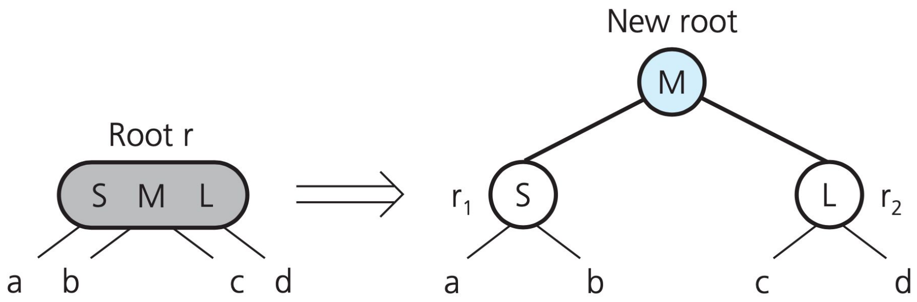


Insertion to a 2-3 tree: general stratey

- recursively execute the process of splitting a node and moving an item up (Middle) to the parent
- base case (termination):
  - only 1 item in a node before the insertion
  - only 2 items in a node after taking on the new item
- 2-3 tree can effectively postponed the growth of the tree’s height

Insertion to a 2-3 tree: general stratey

- When will the height of a 2-3 tree grow?
  - every node on the path to root have 2 items
  - recursively move up the new item to the root
  - create a new root for M to move up
  - increase height from the top



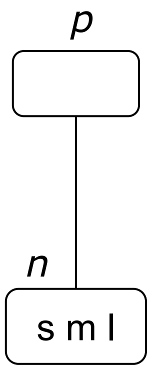
Insertion to a 2-3 tree: pseudocode

```
insertItem(23Tree: TwoThreeTree, newItem: ItemType)
    Locate the leaf, leafNode, in which newItem belongs
    Add newItem to leafNode

    if (leafNode has three items)
        split(leafNode)
```

Insertion to a 2-3 tree: pseudocode

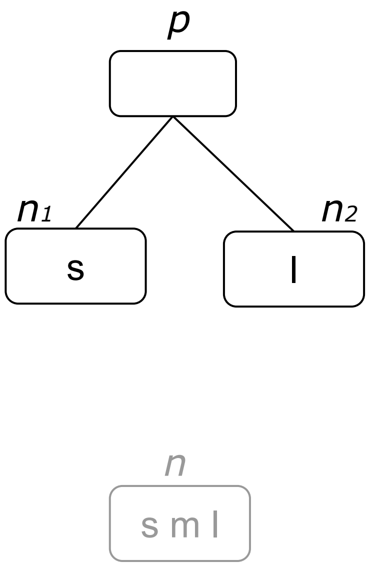
```
split(n: TwoThreeNode)
     if (n is the root)
          Create a new node p //new root
     else
          Let p be the parent of n
```



Insertion to a 2-3 tree: pseudocode

```
split(n: TwoThreeNode)
     if (n is the root)
          Create a new node p
     else
          Let p be the parent of n

     Replace node n with 2 nodes, n1 and n2, so that p is their
parent
          Give n1 the item in n with the smallest value
          Give n2 the item in n with the largest value
```

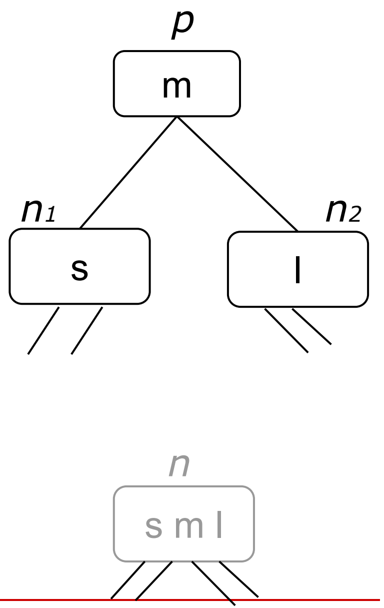


Insertion to a 2-3 tree: pseudocode

```
split(n: TwoThreeNode)
     if (n is the root)
          Create a new node p
     else
          Let p be the parent of n

     Replace node n with 2 nodes, n1 and n2, so that p is their
parent
          Give n1 the item in n with the smallest value
          Give n2 the item in n with the largest value

     if (n is not a leaf){//deal with 4 children nodes
          n1 becomes the parent of n's two leftmost children
          n2 becomes the parent of n's two rightmost children
     }
     Move the item in n that has the middle value up to p
     if (p now has three items)
          split(p)
```



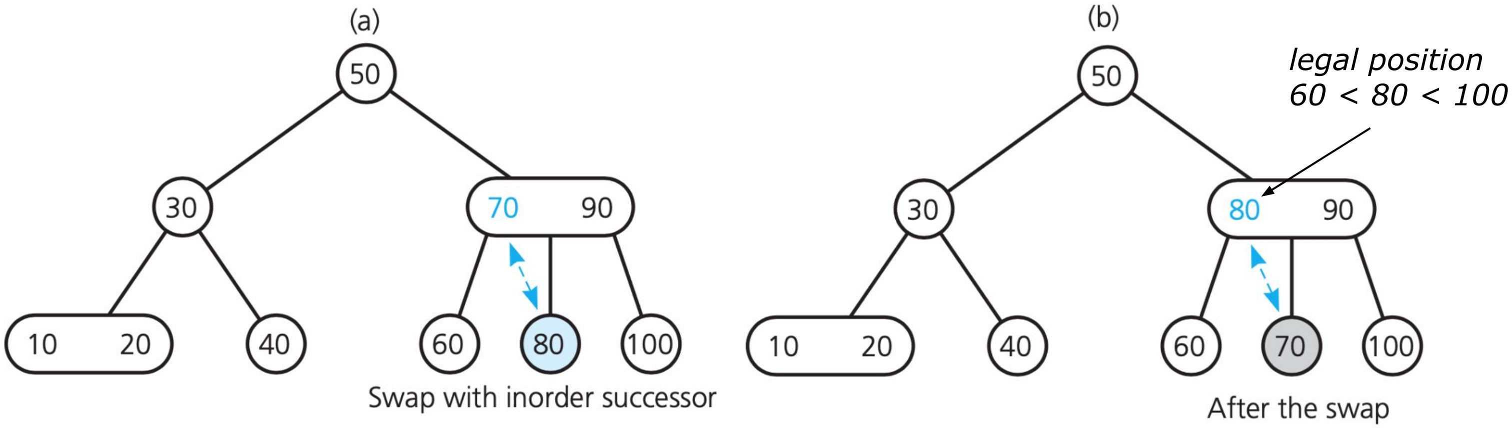
Removing data from a 2-3 tree

- inverse insertion strategy
  - spreads insertions throughout the tree by splitting nodes when they would become too full
  - spreads removals throughout the tree by merging nodes when they become empty

Removing data from a 2-3 tree: example

Removing 70 (1/2)

- search for 70, find it at <70 90>
- want to begin removal <u>at a leaf</u>
- swap 70 with inorder successor 80



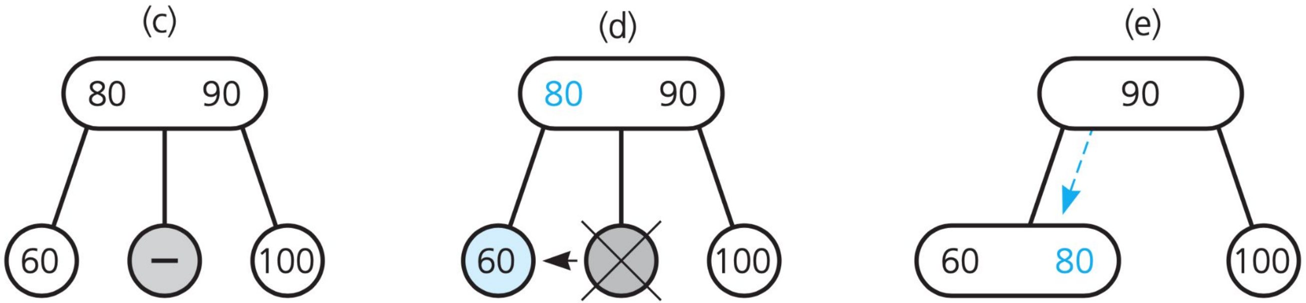
Removing data from a 2-3 tree: example

Removing 70 (2/2)

- delet value 70 from a leaf
- if the leaf contain an other value, then removal is done
- but in this case, the leaf remain empty after removing 70
- then, delete the empty node, 2-value parent node with 2 children node (illegal)
- <u>move</u> smaller value (80) <u>down</u> from parent to child node (**merging**)

illegal in a 2-3 tree

merging: deleting the leaf node and moving a value down to a sibling of the leaf



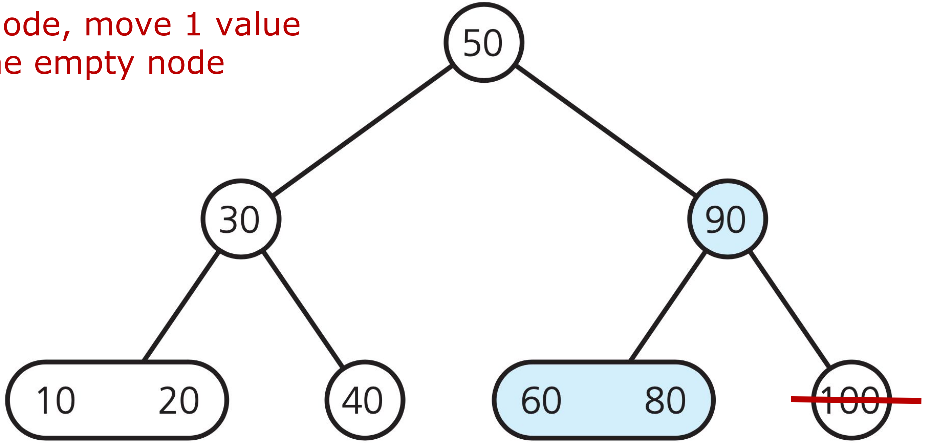
Delete value from leaf

Merge nodes by deleting empty leaf and moving 80 down

Removing data from a 2-3 tree: example

Removing 100 (1/2)

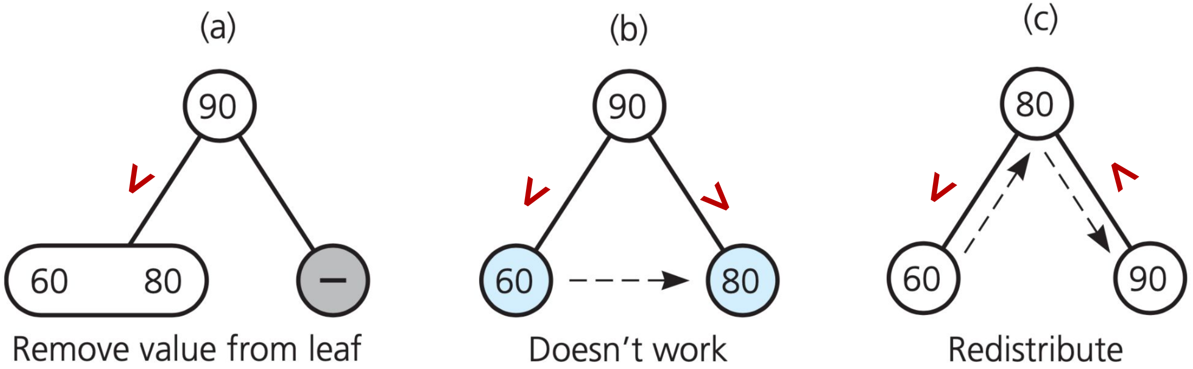
- search for 100, find it at leaf <100>
- remove value 100 from leaf
- no need to merge node, move 1 value from <60 80> to the empty node



Removing data from a 2-3 tree: example

Removing 100 (2/2)

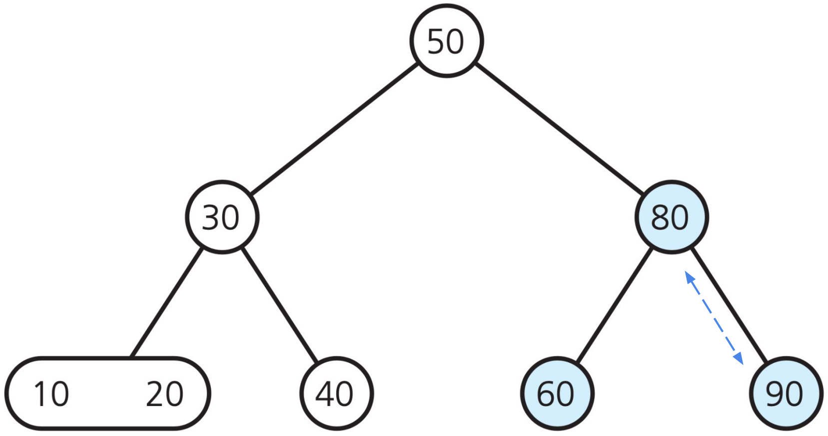
- if move 80 to the empty leaf, but the order of 2-3 search tree is incorrect (90 > 80)
- instead, move 80 to parent and move 90 down into the empty node
- this distribution preserve search-tree order (60 < 80 < 90)



Removing data from a 2-3 tree: example

Removing 80 (1/4)

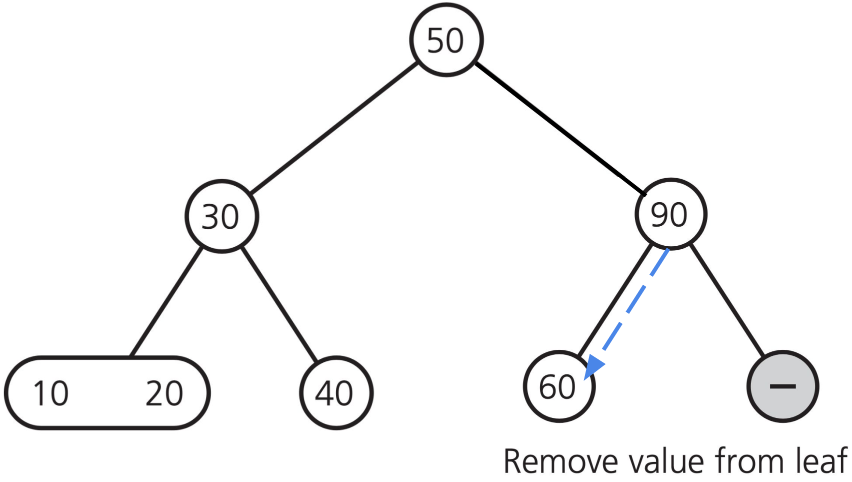
- search for 80, find it’s in an internal node
- swap 80 with its’ inorder successor (90), and remove 80 from the leaf



Removing data from a 2-3 tree: example

Removing 80 (2/4)

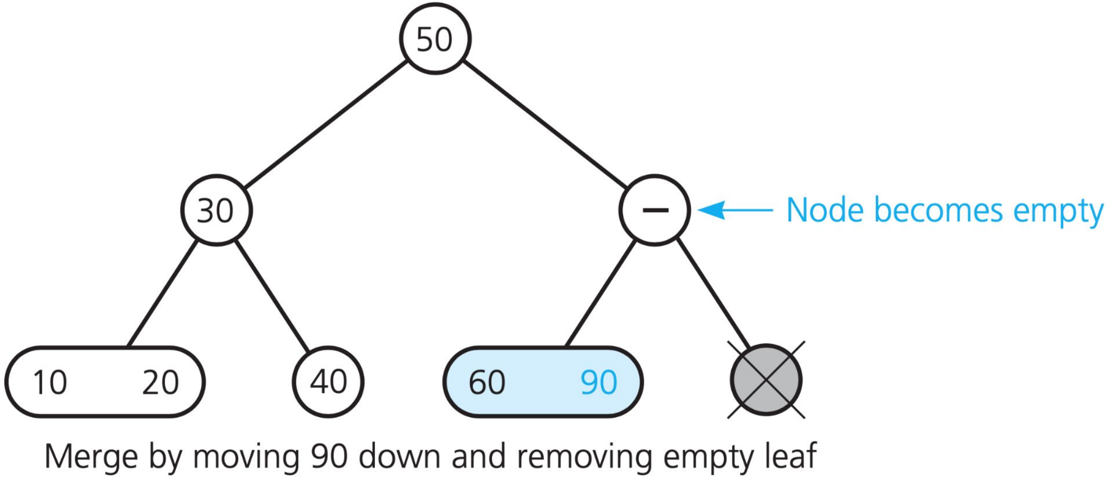
- sibling (<60>) only has 1 value, cannot redistribute as the previous removal of 100
- must merge! move 90 down to <60> and delete the empty leaf



Removing data from a 2-3 tree: example

Removing 80 (3/4)

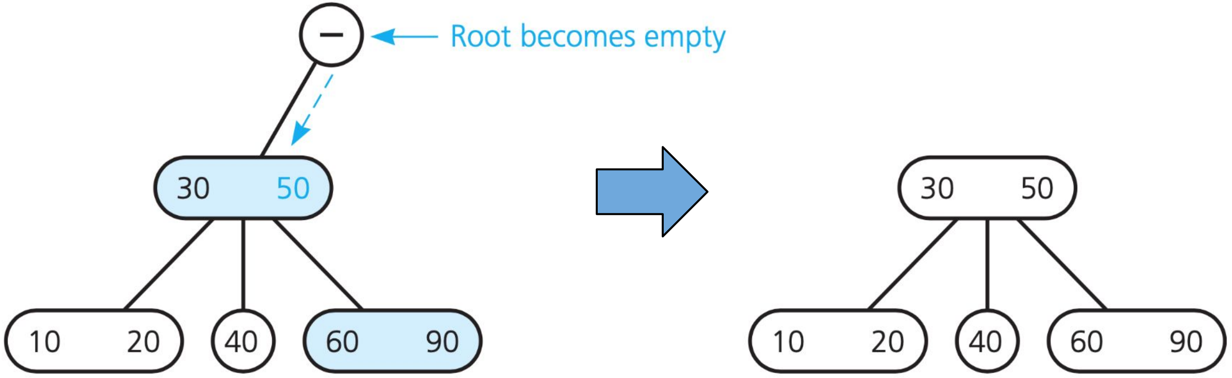
- parent has no data but has 1 child
- <u>recursively</u> apply removal strategy: check if the node’s sibling (<30>) can spare a value



Removing data from a 2-3 tree: example

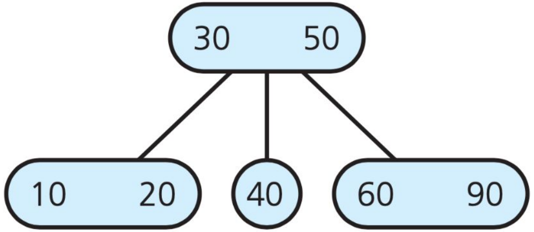
Removing 80 (4/4)

- if no, do merging
  - move down 50 to <30>
  - adpot empty leaf’s child <60 90>
  - delet empty node
- delete empty root (if the node is not root, apply removal strategy *recursively*

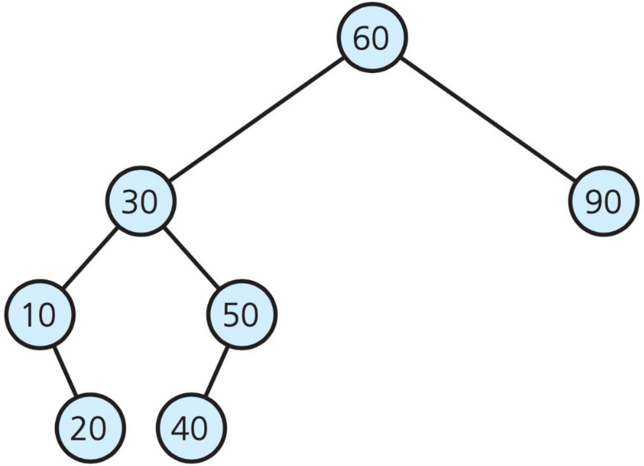


Comparing 2-3 tree with binary search tree

After removing 70, 100, 80



2-3 tree
reduce height by 1



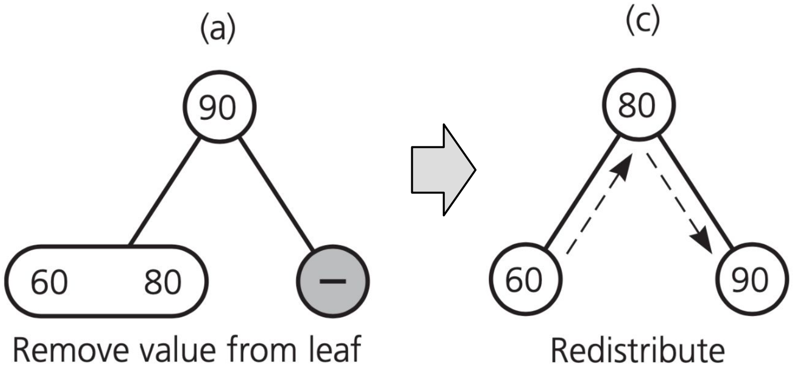
binary search tree
removal only affect 1 part of tree
lose balance

Removal of a 2-3 tree: general stratey

- 1. remove *I* from a 2-3 tree
- 2. locate node *n* that contains *I*
- 3. if *n* is not a leaf
  - 3.1 find *I*'s inorder sucessor
  - 3.2 swap it with *I*
  - 3.3 delete the value *I* from the leaf
  - 3.4 if the leaf contains item in addition to *I* (has other item and I), simply remove *I*
  - 3.5 if the leaf has only *I*, removing *I* will leave a empty leaf, need to do some additional work

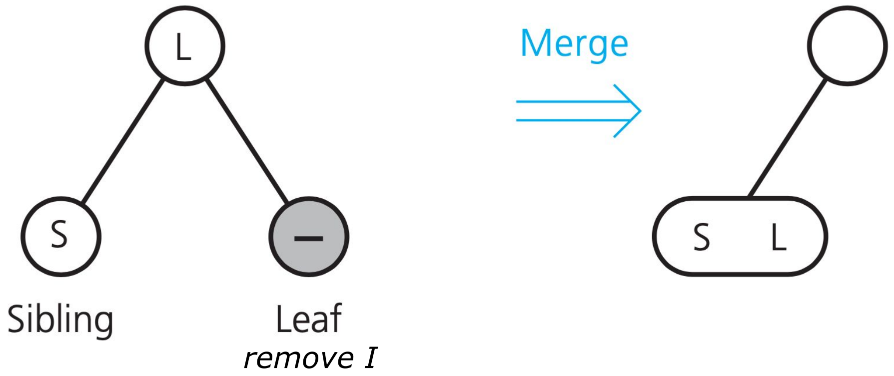
Removal of a 2-3 tree: general stratey

- 3.5 if the leaf has only *I*, removing *I* will leave a empty leaf, need to do some additional work
  - check the siblings of the now-empty leaf, if a sibling has 2 items, redistribute



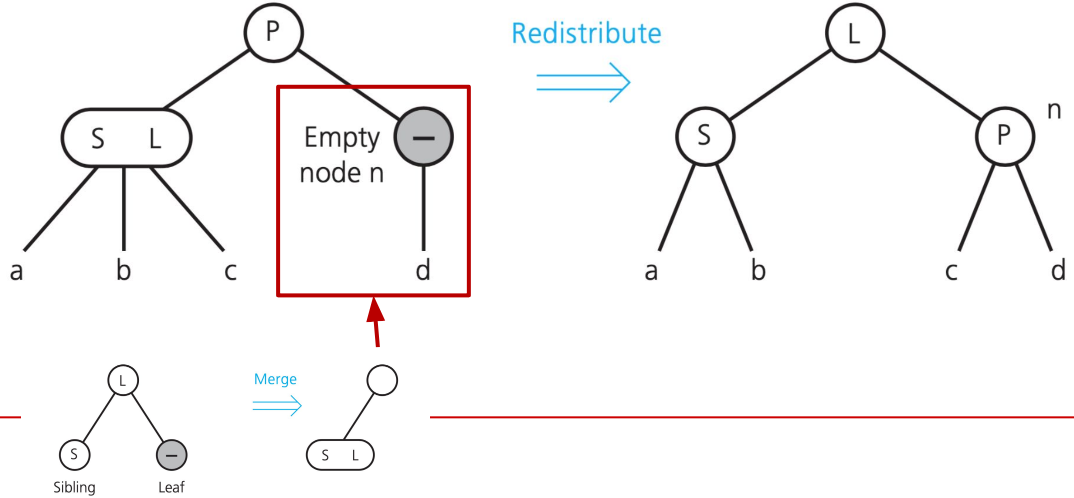
Removal of a 2-3 tree: general stratey

- 3.5 if the leaf has only *I*, removing *I* will leave a empty leaf, need to do some additional work
  - if no sibling has 2 items, do merging: moving an item down from the leaf's parent and delete the empty leaf



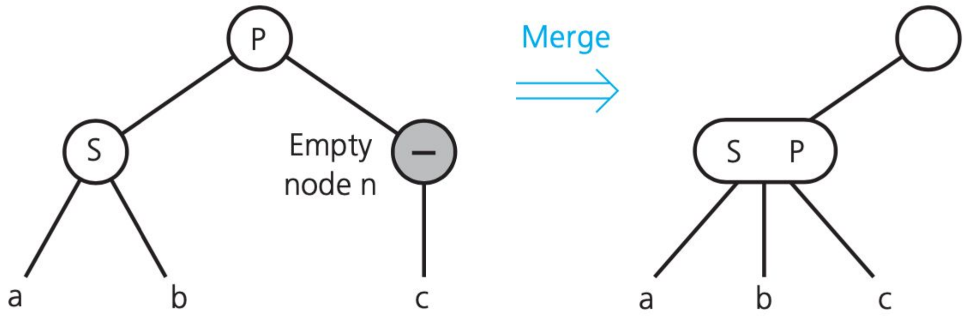
Removal of a 2-3 tree: general stratey

- if no sibiling has 2 items, do merging
  - may cause *n* left without data item and with 1 child
  - recursively apply removal algorithm to *n*
    - check sibling with 2 items, redistribute, adopt children



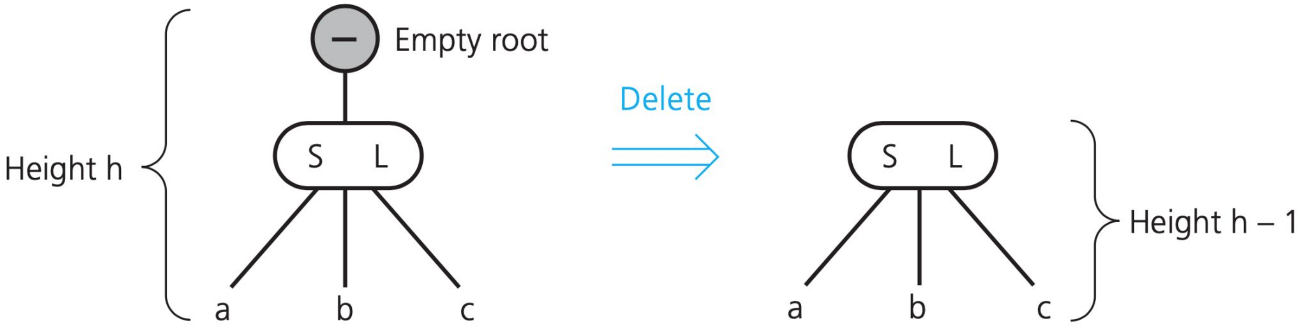
Removal of a 2-3 tree: general stratey

- if no sibiling has 2 items, do merging
  - recursively apply removal algorithm to *n*
    - if no sibling with 2 items, do merging (recursion)
    - if merging make parent be without an item, recursively do another round of removal process



Removal of a 2-3 tree: general stratey

- if no sibiling has 2 items, do merging
  - recursively apply removal algorithm to *n*
    - if merging continue to root and root is without item, delete the root (height reduced by 1



Removal of a 2-3 tree: pseudocode

```
removeItem(23Tree: TwoThreeTree, dataItem: ItemType): boolean
     Attempt to locate dataItem
     if (dataItem is found){
          if (dataItem is not in a leaf)
               Swap dataItem with its inorder successor, which will be in leaf leafNode

          // The removal always begins at a leaf
          Remove dataItem from leaf leafNode
          if (leafNode now has no items) //empty leafNode
               fixTree(leafNode)
               return true

     {
     else
          return false //didn’t locate dataItem in the given tree
```

Removal of a 2-3 tree: pseudocode

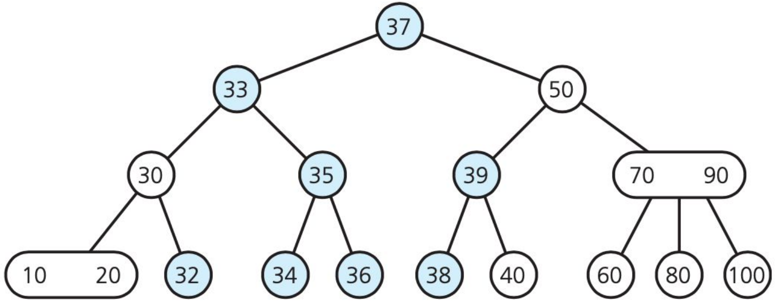
```
fixTree(n: TwoThreeNode)
      if (n is the root)
            Delete the root
      else{
            Let p be the parent of n
            if (some sibling of n has  two items){
                  Distribute items appropriately among n, the sibling, and p
                  if (n is internal) //not a leaf
                        Move the appropriate  child from sibling to n  //adopting child node
            }
            else{ //no sibling has 2 items, do merging
                  Choose an adjacent  sibling s of n
                  Bring appropriate item  down from p into s  //move item down from parent
                  if (n is internal)
                        Move n's child to s
                  Remove node n  //arrange child so can delete empty node n
                  if (p is now empty)
                        fixTree(p) //recursion
            }
      }
```

Summary of 2-3 tree

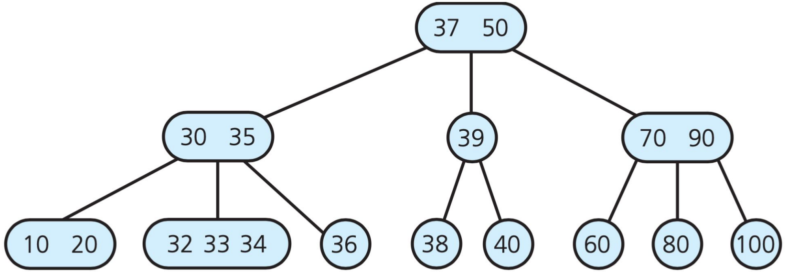
- 2-3 tree is always balanced
- can search a 2-3 tree in all situation with the efficiency of a balanced binary search tree
  - extra work required to maintain the structure of 2-3 tree (merging, splitting) is not signigicant
- A 2-3 tree implementation of a dictionary is O(log n) for all of its operations

2-3-4 Trees

- similar to 2-3 tree, but allows 4-nodes
- can perform insertions and removals on a 2-3-4 tree with fewer steps than a 2-3 tree requires.
- but 2-3-4 tree need greater storage requirements



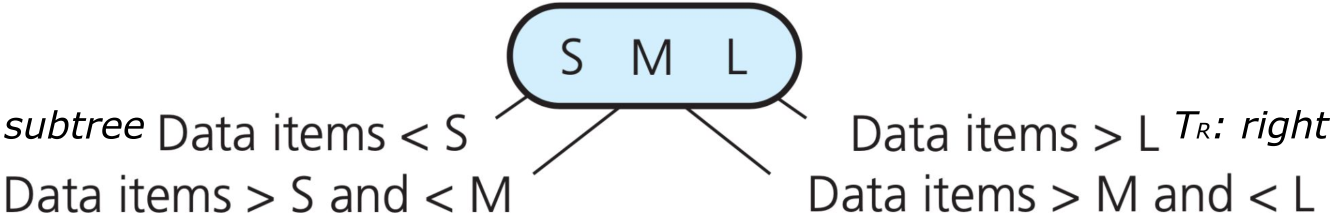
2-3 tree
height: 4



2-3-4 tree
height: 3

2-3-4 Trees: 4-nodes

- rules of 2-nodes and 3-nodes are the same with 2-3 tree



T_L: left subtree

T_{ML}: middle-left subtree

T_{MR}: middle-right subtree

T_R: right subtree

2-3-4 Trees

- 2-3-4 tree need greater storage requirements than 2-3 tree

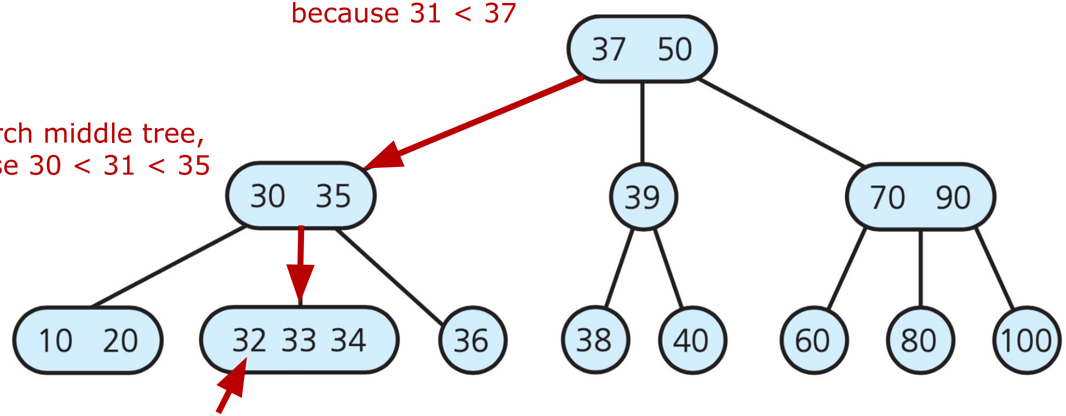
```
template<class ItemType>
class QuadNode{
      ItemType smallItem, middleItem, largeItem;
      QuadNode<ItemType>* leftChildPtr;
      QuadNode<ItemType>* leftMidChildPtr;
      QuadNode<ItemType>* rightMidChildPtr;
      QuadNode<ItemType>* rightChildPtr;
      ...
}
```

Searching and Traversing in a 2-3-4 tree

- simple extenions of search for 2-3 tree
- for example, search for 31

1.search left tree of the root,
because 31 < 37

2. search middle tree,
because 30 < 31 < 35



3. terminate at left pointer becuase 31 < 32, 31 is not in the tree

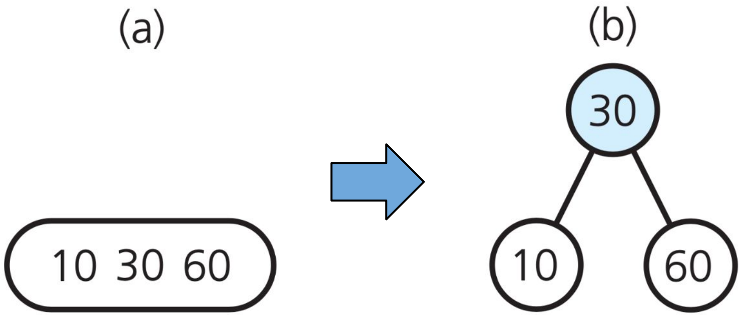
Inserting Data into a 2-3-4 Tree

- inserting 60, 30, and 10 into a empty 2-3-4 tree
  - generate a 4-nodes

(a)

Inserting Data into a 2-3-4 Tree

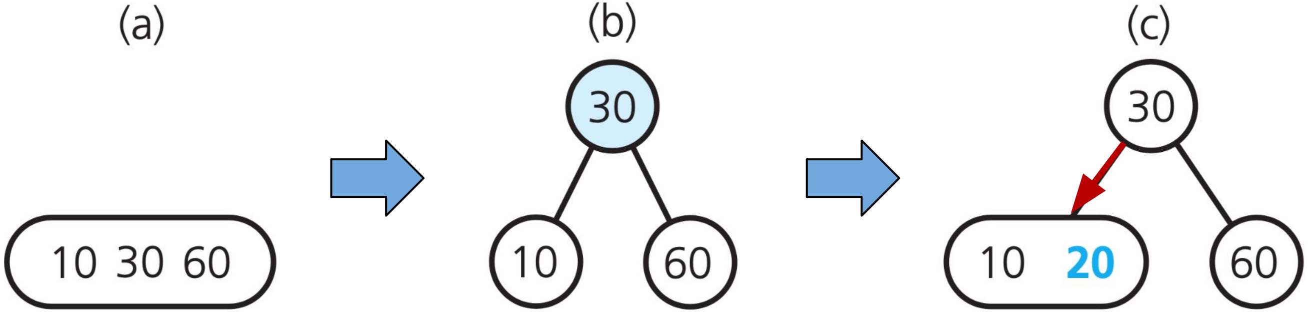
- inserting 60, 30, and 10 into a empty 2-3-4 tree
- inserting 20
  - For insertion of 2-3-4 tree, split 4-node as they are encountered



split by moving middle
value 30 up to new root

Inserting Data into a 2-3-4 Tree

- inserting 60, 30, and 10 into a empty 2-3-4 tree
- inserting 20
  - For insertion of 2-3-4 tree, split 4-node as they are encountered

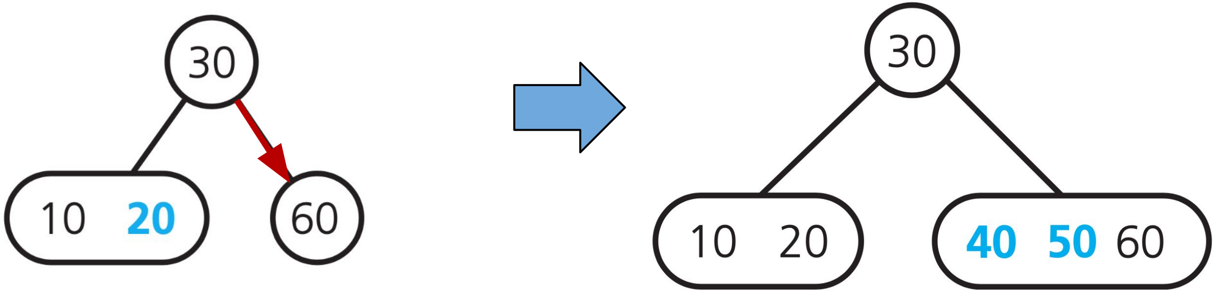


split by moving middle value 30 up to new root

search for 20, 20 < 30 so locate the left node <10>, then insert 20

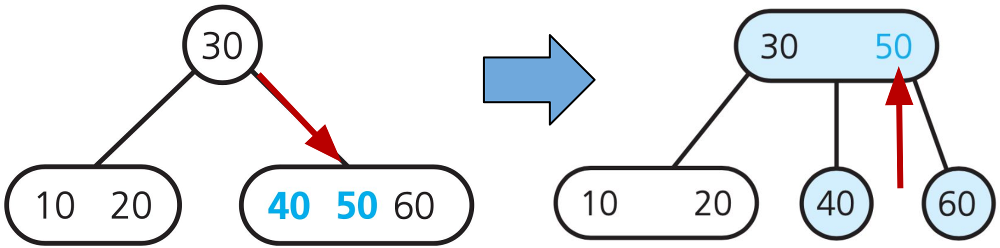
Inserting Data into a 2-3-4 Tree

- inserting 50 and 40: don't require splitting
  - search for 50, 50 > 30, locate right node <60>, insert 50
  - search for 40, 40 > 30, locate right node <50 60>, insert 40



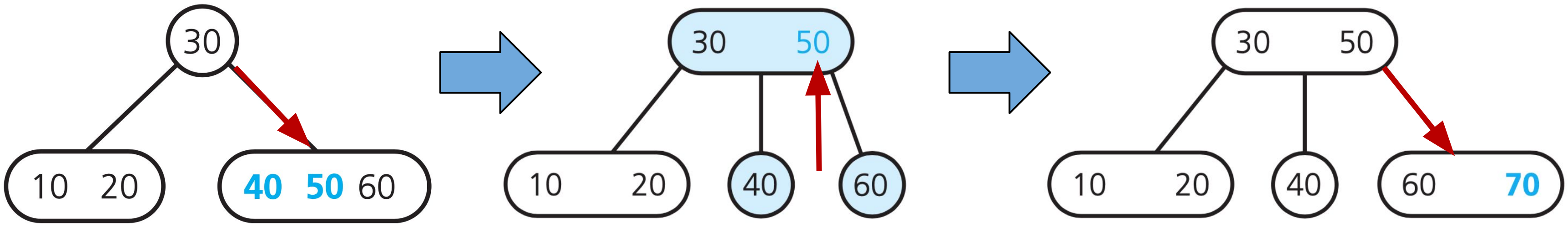
Inserting Data into a 2-3-4 Tree

- inserting 70
  - search for 70, 70 > 30, locate left node <40 50 60>
  - encounter 4-nodes, do splitting
    - move up middel 50 to parent node <30>



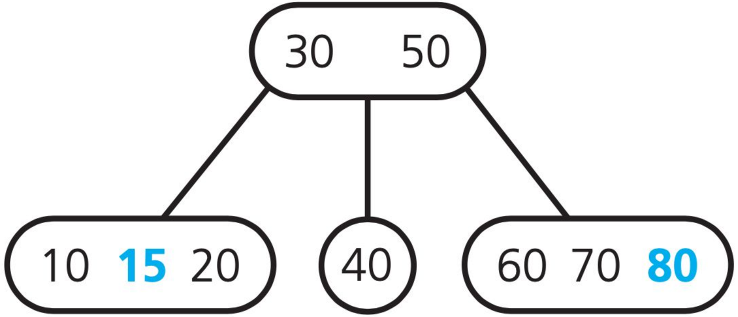
Inserting Data into a 2-3-4 Tree

- inserting 70
  - search for 70, 70 > 30, locate left node <40 50 60>
  - encounter 4-nodes, do splitting
    - move up middel 50 to parent node <30>
    - insert 70 to right node <60>



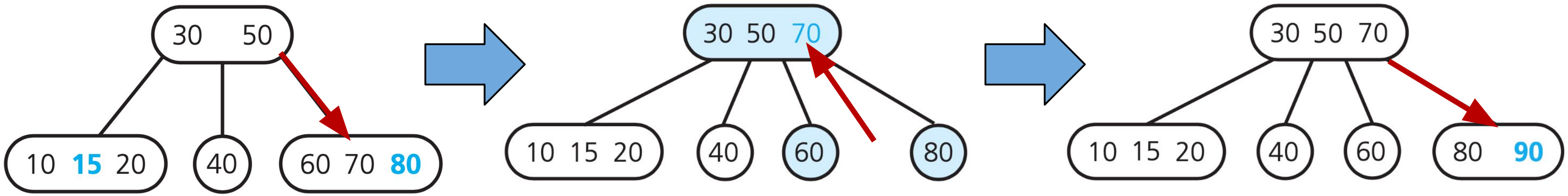
Inserting Data into a 2-3-4 Tree

- inserting 80 and 15
  - do not require split nodes
  - can see that more capacity in each node can prevent changing tree's structure



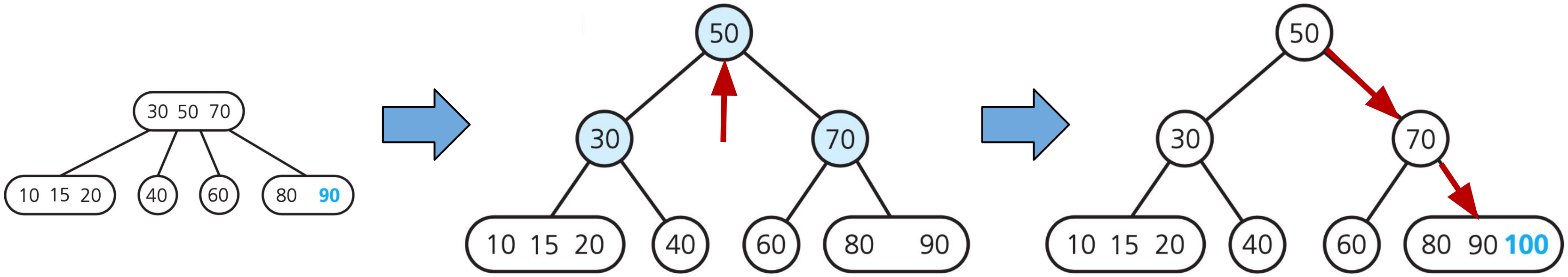
Inserting Data into a 2-3-4 Tree

- inserting 90
  - search for 90, 90 > 50, encounter right node <60 70 80>, do splitting
  - 90 > 70, insert 90 to right node <80>



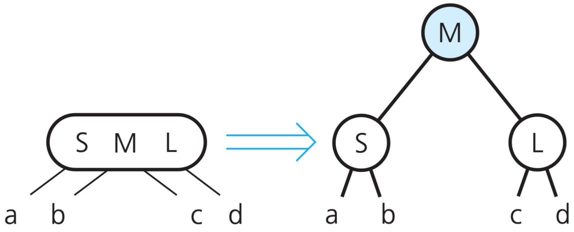
Inserting Data into a 2-3-4 Tree

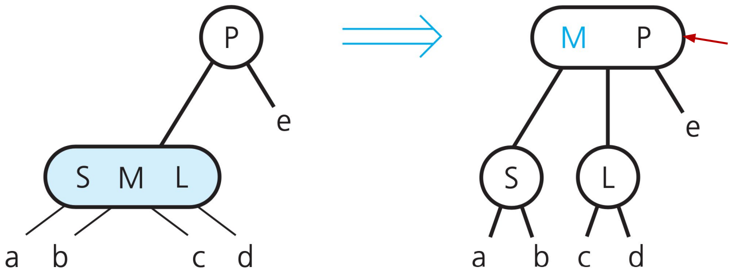
- inserting 100
  - search for 100, encounter 4-node at root, split
  - move up 50 to new root, split <30 70>
  - 100 > 50 & 100 > 70, locate left node <80 90>, insert 100



Inserting Data into a 2-3-4 Tree

- splitting 4-node during insertion
  - splitted new nodes can accommodate new item to be inserted
  - each 4-node will become:
    - be the root
    - have 2-node or 3-node parent
  - adopting children nodes by dividing left and right pairs

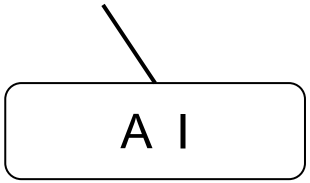




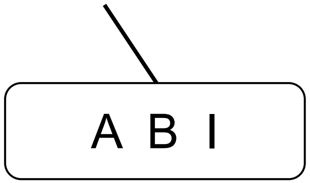
place moved-up value
in order
<small middle large>

Removing Data from a 2-3-4 tree

- similar with removal algorithm for a 2-3 tree
- locate/search for node that has the item *I* (that you want to remove
- removal will always be at a leaf
  - if *I* is not in a leaf, find *I*'s inorder successor and swap it with *I*
  - simply remove *I* from the leaf if it's a 3- or 4-node



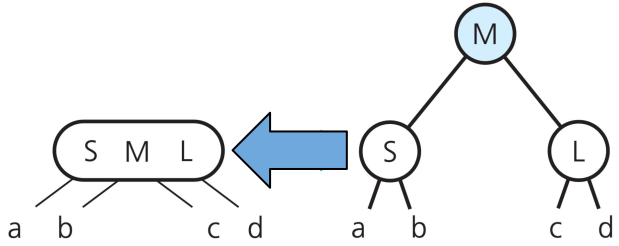
3-node leaf



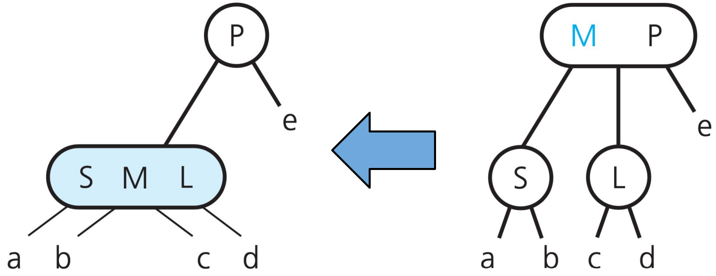
4-node leaf

Removing Data from a 2-3-4 tree

- if *I* is not in a 2-node, removal is simple, to ensure *I* doesn't occur in a 2-node
- transform each encountered 2-node into 3-node or 4-node



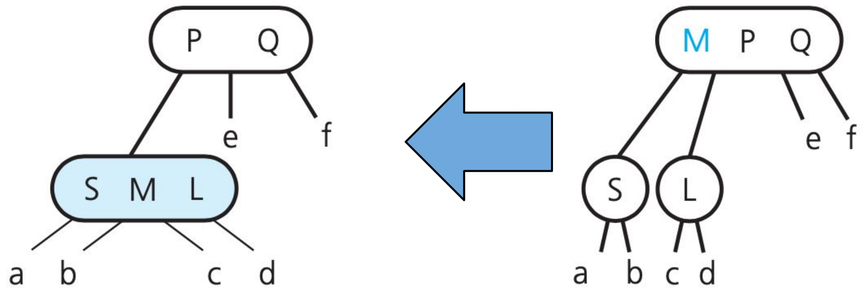
parent & nearest sibling is 2-node



parent is 3-node,
nearest sibling is 2-node

Removing Data from a 2-3-4 tree

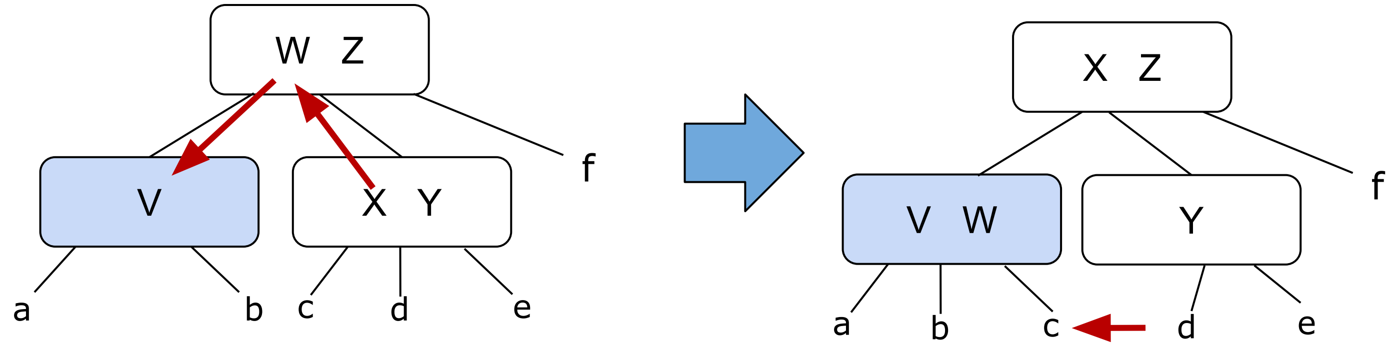
- if I is not in a 2-node, removal is simple, to ensure I doesn't occur in a 2-node
- transform each encountered 2-node into 3-node or 4-node



parent is 4-node,
nearest sibling is 2-node

Removing Data from a 2-3-4 tree

- what if the nearest sibling is not a 2-node?
- when nearest sibling is 3-node
  - redistribut between parent and sibling
  - adopting child node
- apply same stratey for 4-node nearest sibling



reference: http://smile.ee.ncku.edu.tw/old/Links/MTable/Course/DataStructure/2-3,2-3-4&red-blackTree_952.pdf

2-3 tree versus 2-3-4 tree

- Advantage of both 2-3 and 2-3-4 trees is their easy-to-maintain balance
- 2-3-4 tree's insertion and removal algorithm require fewer steps
- so 2-3-4 tree is more efficient than those for 2-3 tree

Nodes with more than 4 children?

- although node with more children can reduce the height of a tree
- require more comparisons when searching at each node
- allowing nodes with more than 4 children is counterproductive

Summary

- 2-3 Trees
- 2-3-4 Trees
- insertion and removal
- Advantages

next week

- Red-Black Trees
- AVL Trees