Balanced Search Tree

Balanced + ***Binary Search Tree***

Outline

- 2-3 tree
- 2-3-4 tree
- AVL tree
  - [Adelson-Velskii & Landis, 1962]
- Red-black tree
  - [Rudolf Bayer, 1972]… B-tree

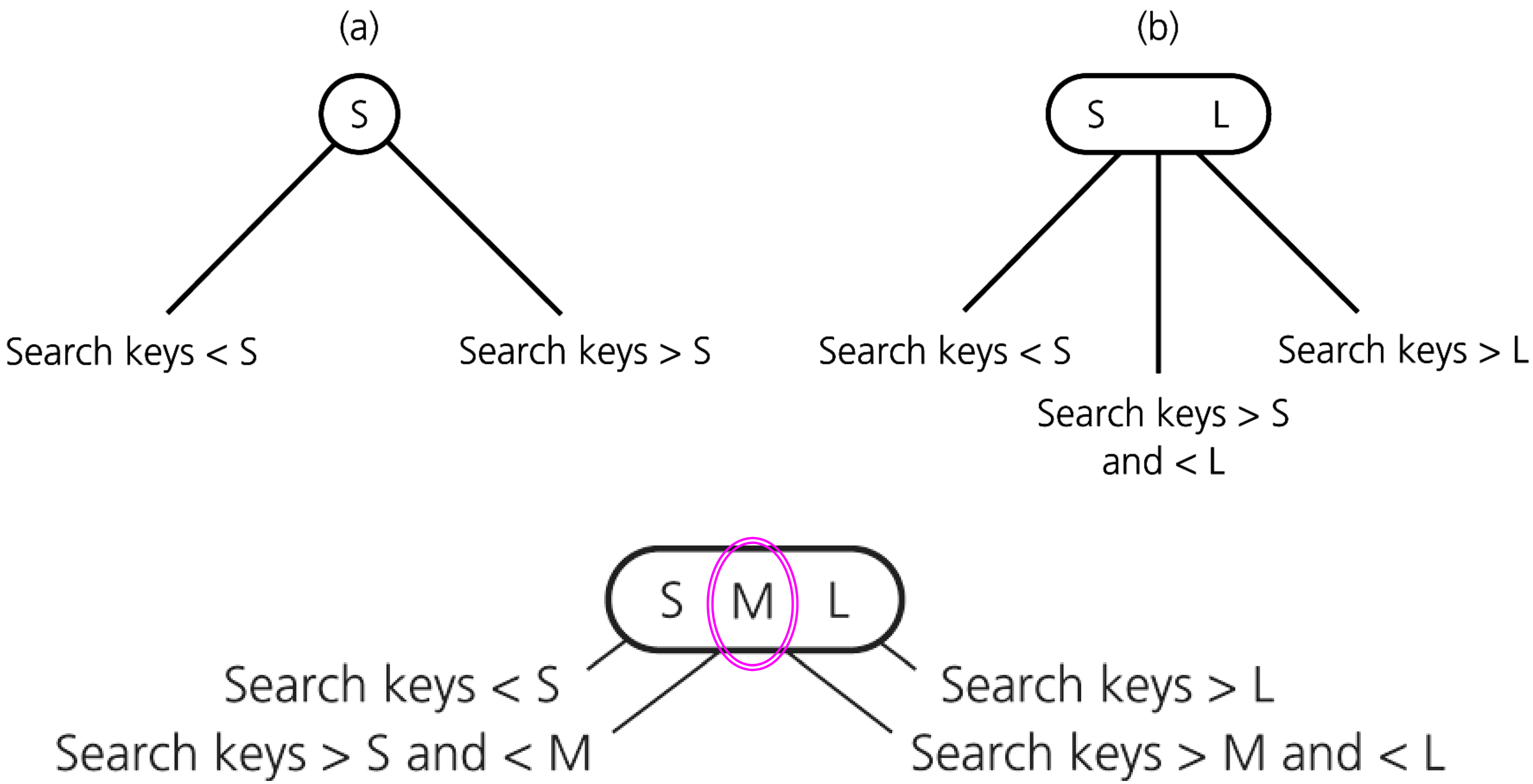
2-3-4 Tree

Data Structures

- **2-3-4 trees have 2-nodes, 3-nodes, and 4-nodes**
  - A 2-node has *one* data item and *two* children
  - A 3-node has *two* data items and *three* children
  - A 4-node has *three* data items and *four* children
- **Are general trees, not binary trees**
- **Are never taller than a 2-3 tree**
- **Search and traversal algorithms for a 2-3-4 tree are simple *extensions* of the corresponding algorithms for a 2-3 tree**

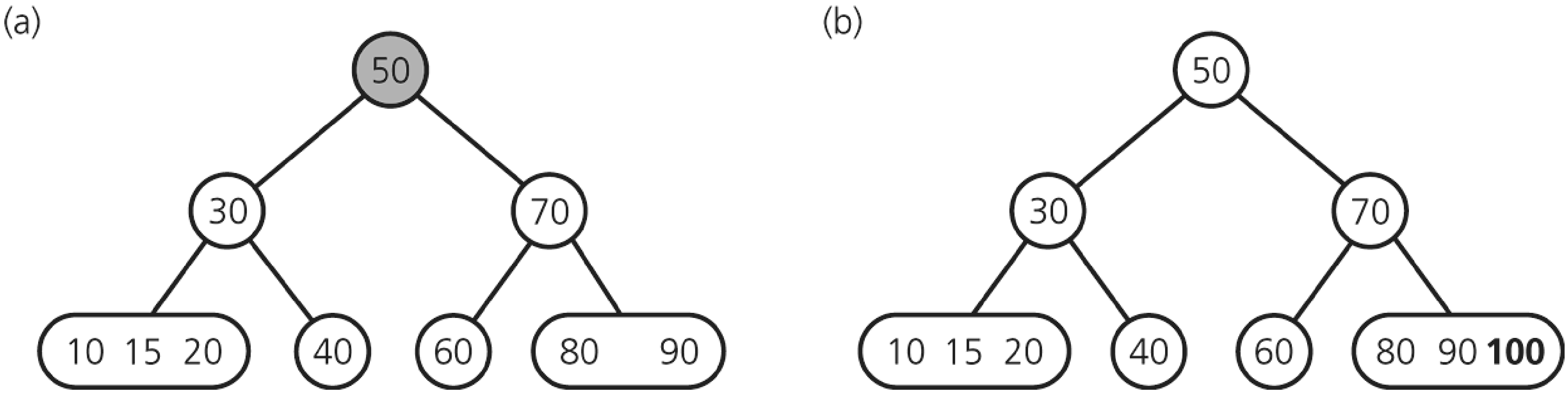
Placing Data Items in a 2-3-4 Tree

Data Structures



Placing Data Items in a 2-3-4 Tree

- A leaf can contain either one, two, or three data items



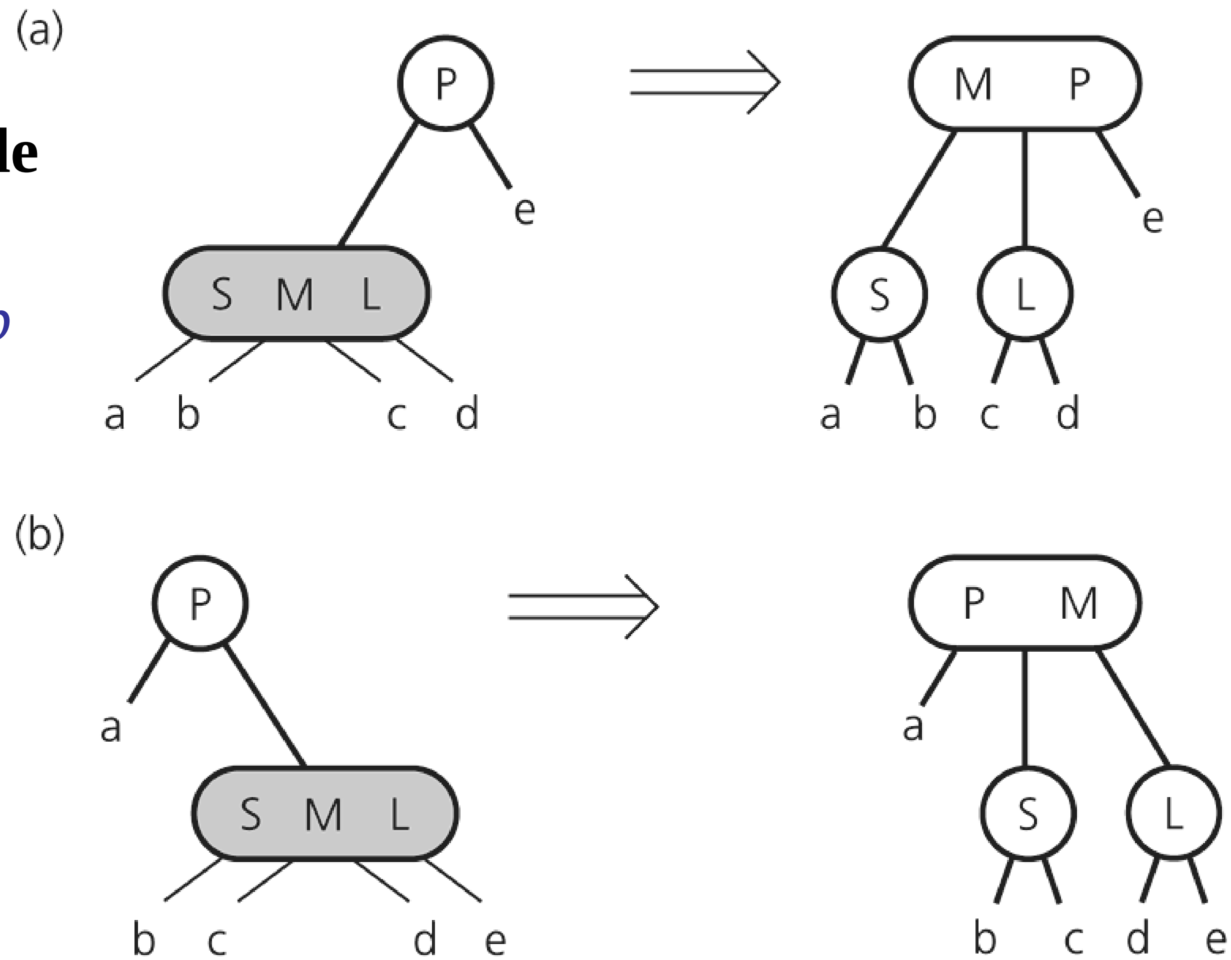
2-3-4 Tree: Split 4-nodes during Insertion

Data Structures

- **The insertion algorithm for a 2-3-4 tree**
  – Split a 4-node by moving the middle item up to its parent node
  – Split 4-nodes as soon as its encounters them during a search from the root to a leaf (*downward*)
- **A 4-node that is split will eventually become**
  – the root, or the parent of a 2-node or 3-node
- **Impact**
  – When a 4-node is split, its parent cannot possibly be a 4-node (*recursion*), so it can accommodate the item moved up from the 4-node.

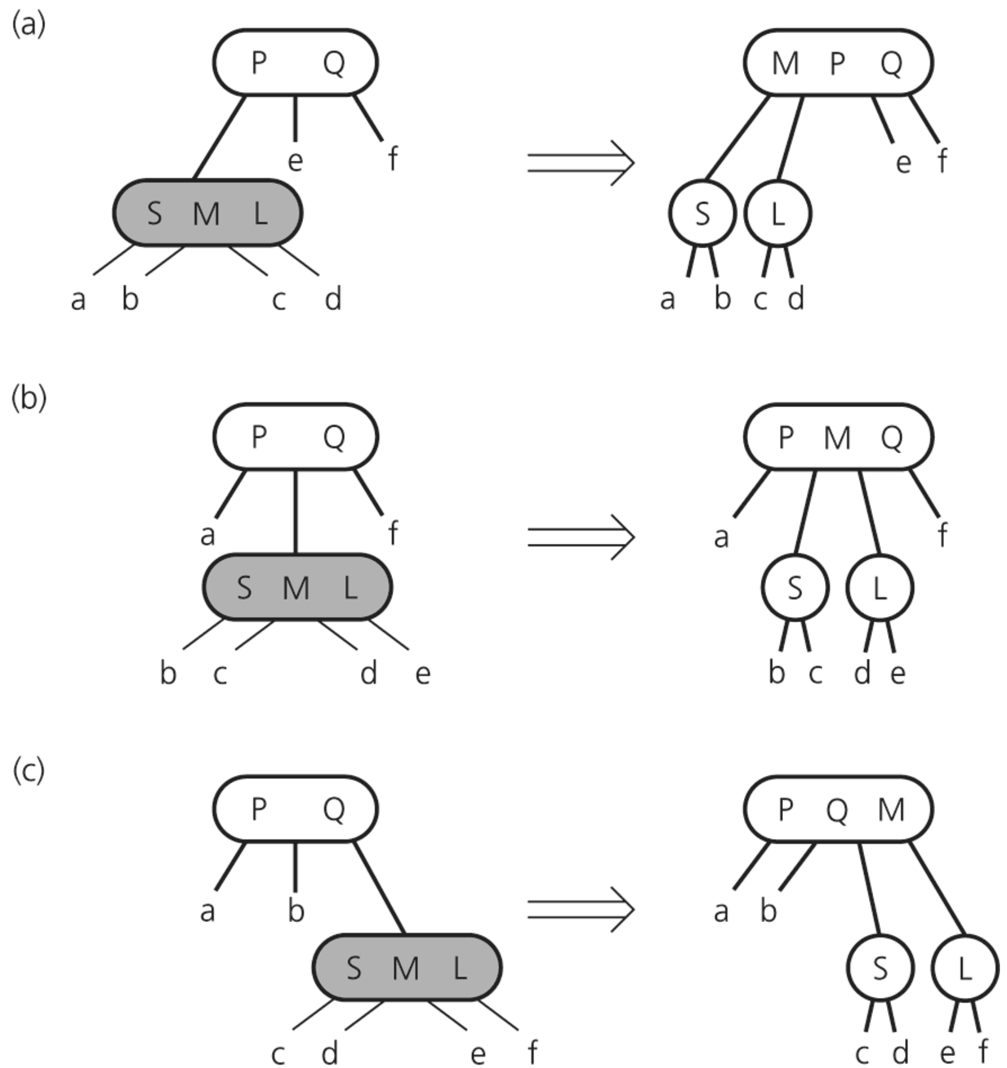
2-3-4 Tree: Split 4-nodes during Insertion

- **If parent is 2-node**
  (a), (b) *move the middle item up*



2-3-4 Tree: Split 4-nodes during Insertion

- **If parent is 2-node**
  - (a), (b) *move the middle item up*
- **If parent is 3-node**
  - (c), (d), (e) *move the middle item up*



2-3-4 Tree: Split 4-nodes during Insertion

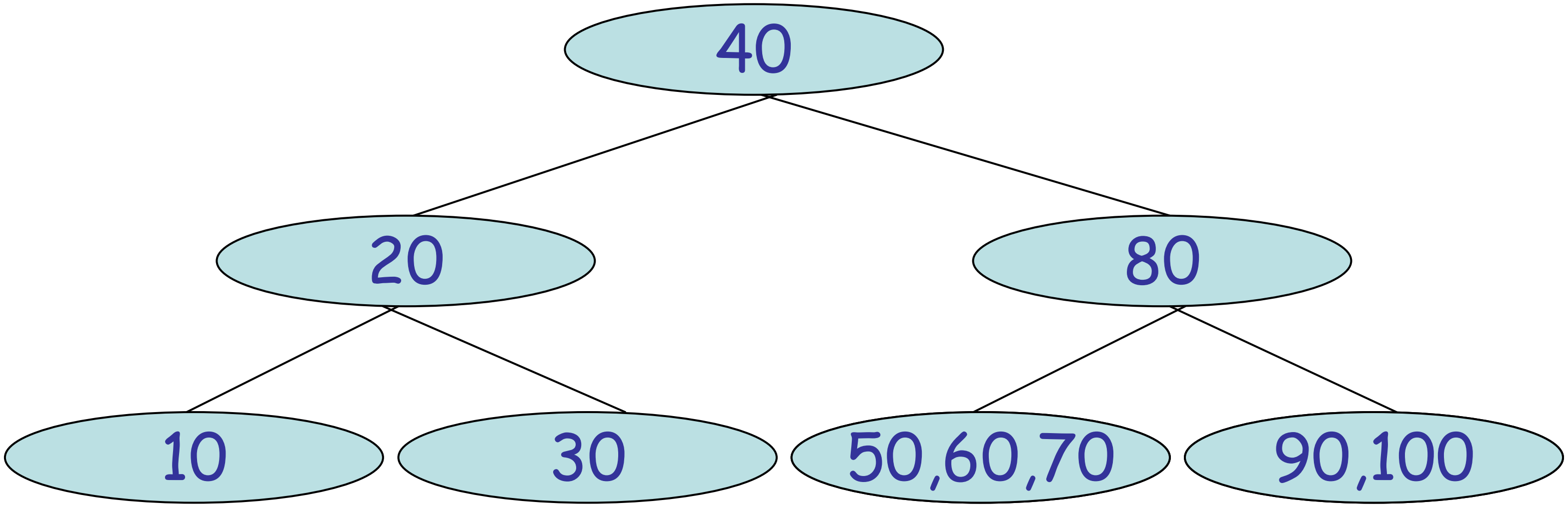
- **Splits occur only at the path from the root to a leaf (*downward*)!**

- **No upward recursion is needed!**

2-3-4 Tree: *Insertion*

Build *2-3-4 tree* by insertion:

10, 20, 30, 40, 50, 90, 80, 70, 60, 100



Insert into a leaf:

- Split (4-nodes on the path)

Practice 3: *Inserting* into a 2-3-4 Tree

- **Input order: 10 12 30 8 60 40 70**

Deleting from a 2-3-4 Tree

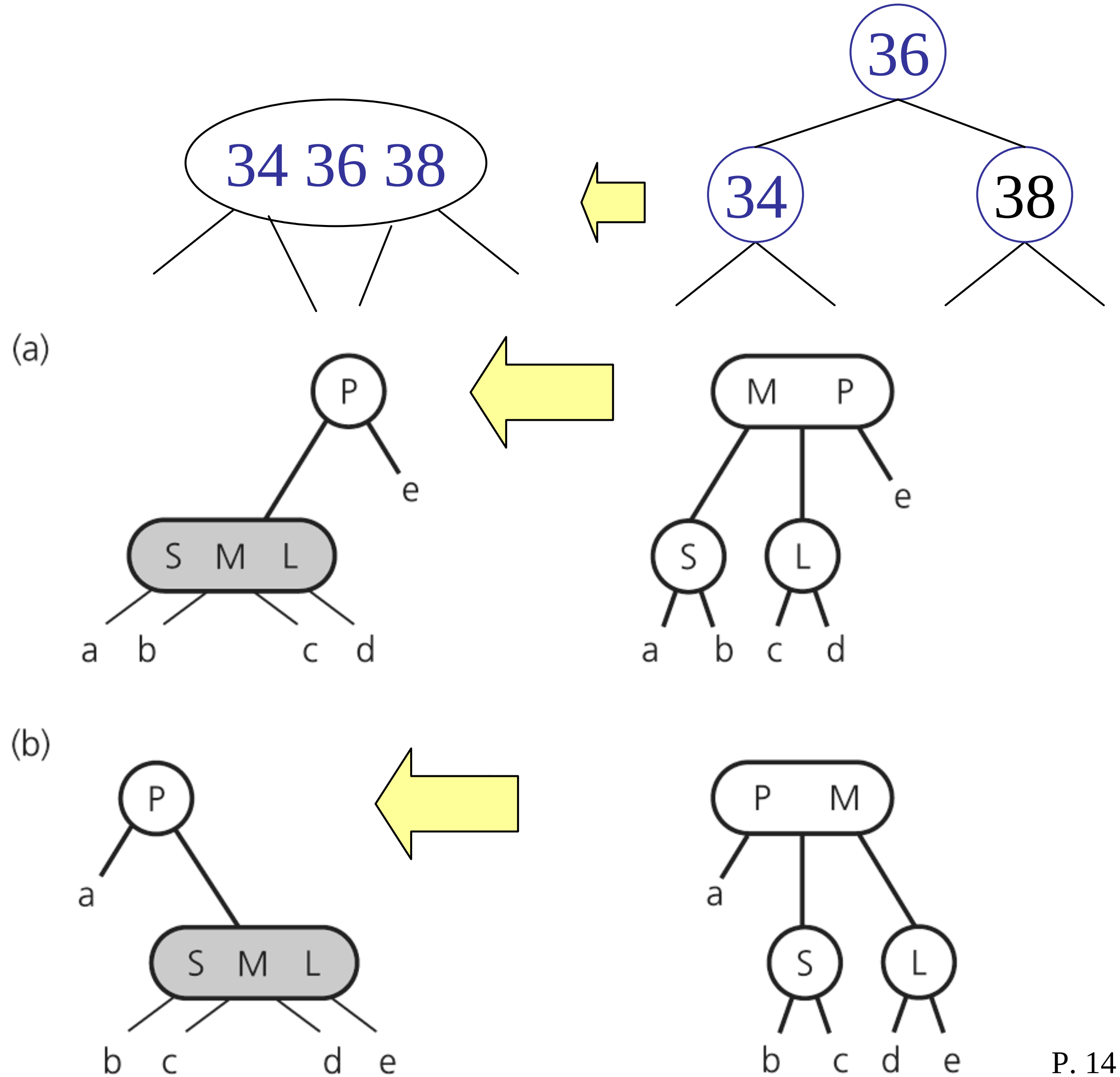
1. Locate the node *n* that contains the item *theItem*
2. Find *theItem*'s inorder successor and swap it with *theItem* (deletion will always occur at a leaf)
3. If that leaf is a 3-node or a 4-node, remove *theItem*

Deleting from a 2-3-4 Tree

- **To ensure that *theItem* does not occur in a 2-node**
  - *Transform* each 2-node encountered into a 3-node or a 4-node
  - Apply the appropriate transformation for splitting nodes during insertions, but in reverse

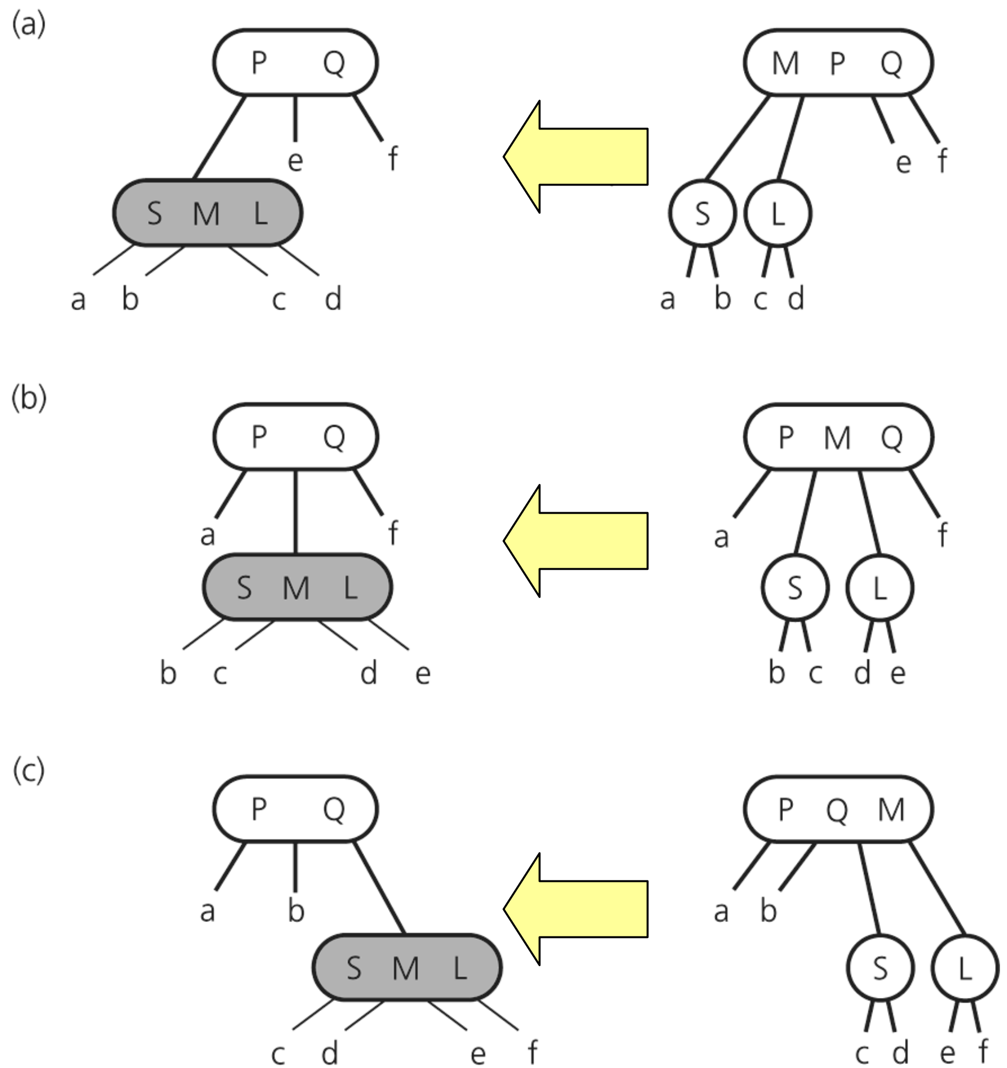
Transform 2-nodes during Deletion

- If parent and sibling are 2-nodes
- If the parent is a 3-node



Transform 2-nodes during Deletion

- If parent and sibling are 2-nodes
- If the parent is a 3-node
- If the parent is a 4-node



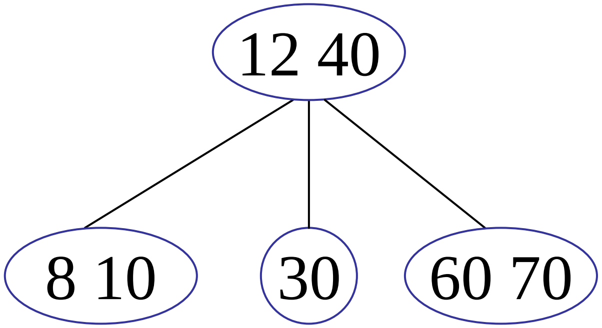
Transform 2-nodes during Deletion

- **Transformations occur only at the path from the root to a leaf (*downward*)!**

- **No upward recursion is needed!**

Practice 4: *Deleting* from a 2-3-4 Tree

- After the deletions of 30, 10, 60



2-3-4 Tree: *Summary*

Data Structures

- **Insertion/deletion algorithms for a 2-3-4 tree require fewer steps than those for a 2-3 tree**
  - Only one pass from root to a leaf
- **A 2-3-4 tree is always balanced**
- **A 2-3-4 tree requires more storage than a binary search tree**
- **Allowing nodes with more data and children is counterproductive, unless the tree is in *external storage***

Summary

Data Structures

- **A 2-3 tree and a 2-3-4 tree are variants of a *binary search tree* in which nodes can contain more than one data item and have more than two children**
- **The *balance* of a 2-3 tree or a 2-3-4 tree is easily maintained**
- **The insertion and deletion algorithms for a 2-3-4 tree are more efficient than the corresponding algorithms for a 2-3 tree**