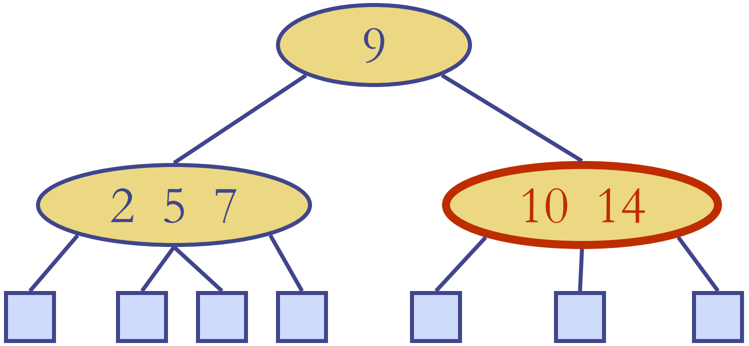
(2,4) Trees

Algorithms & Data Structures
ITCS 6114/8114

Dr. Dewan Tanvir Ahmed
Department of Computer Science
University of North Carolina at Charlotte

(2,4) Trees

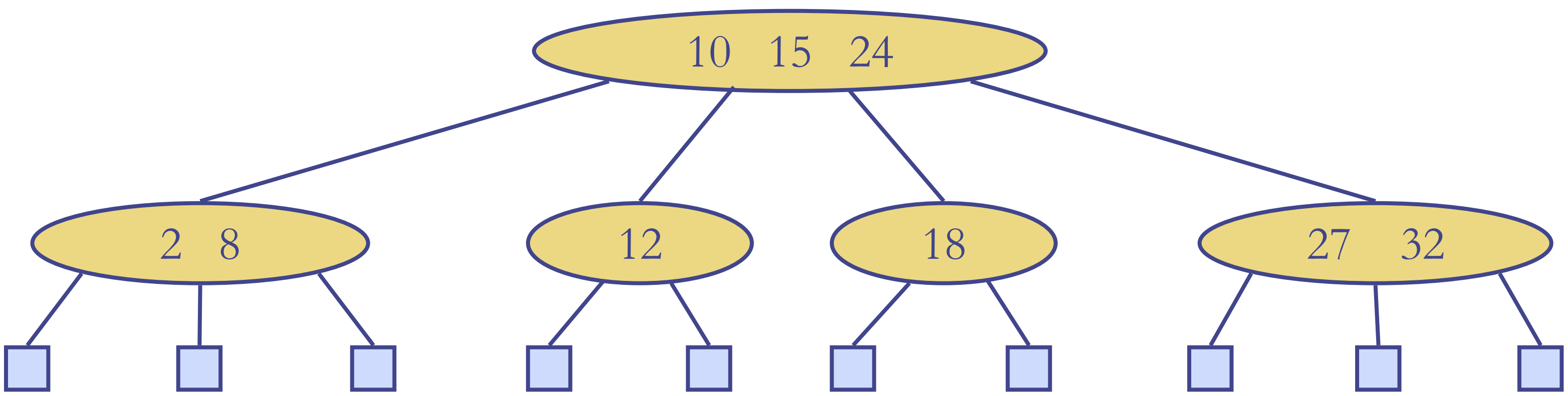


Outline and Reading

- Multi-way search tree (§ 3.3.1)
  - Definition
  - Search
- (2,4) tree (§ 3.3.2)
  - Definition
  - Search
  - Insertion
  - Deletion
- Comparison of dictionary implementations

(2,4) Tree

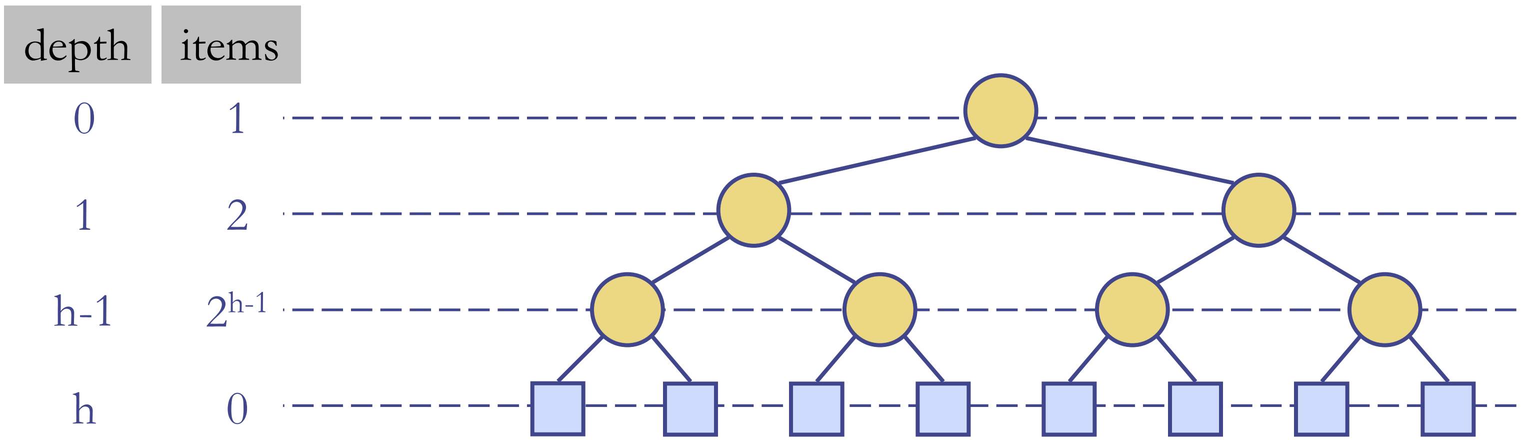
- A (2,4) tree (also called 2-4 tree or 2-3-4 tree) is a multi-way search with the following properties
  - Node-Size Property: every internal node has at most four children
  - Depth Property: all the external nodes have the same depth
- Depending on the number of children, an internal node of a (2,4) tree is called a 2-node, 3-node or 4-node



Height of a (2,4) Tree

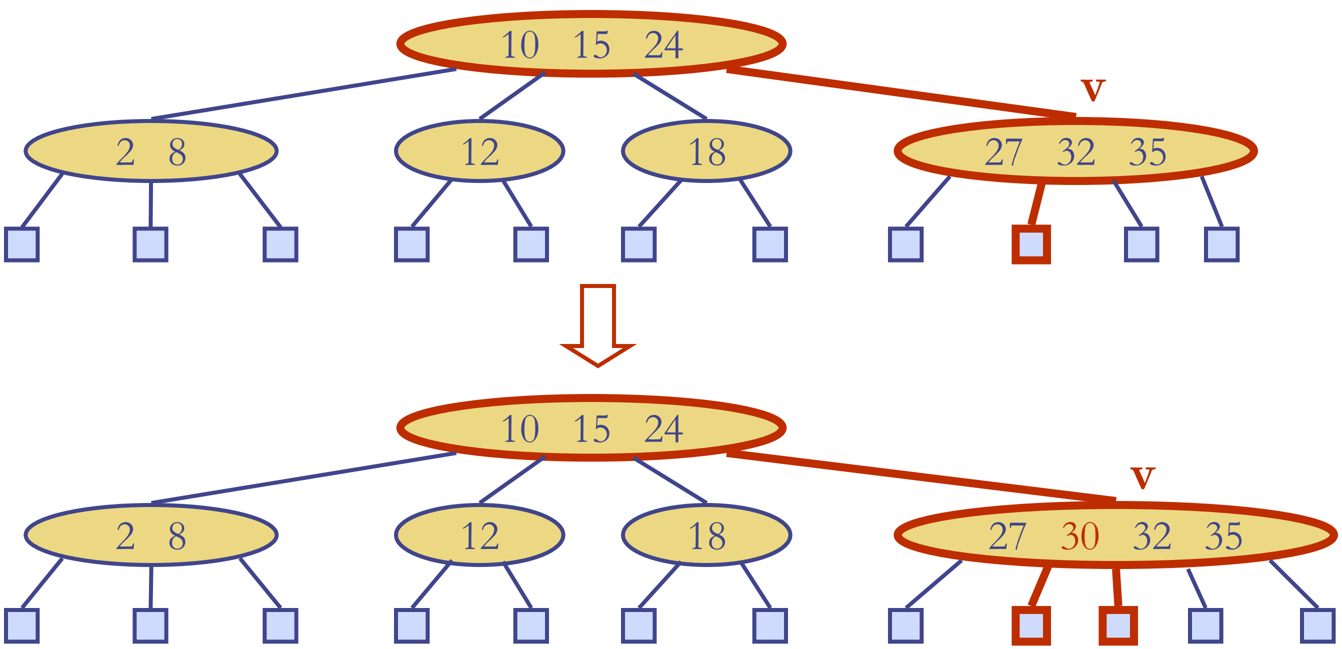
- Theorem: A (2,4) tree storing n items has height $O(\log n)$

  Proof:
  - Let h be the height of a (2,4) tree with n items
  - Since there are at least 2^i items at depth $i = 0, \dots, h-1$ and no items at depth h, we have
    $$n \geq 1 + 2 + 4 + \dots + 2^{h-1} = 2^h - 1$$
  - Thus, $h \leq \log(n+1)$
- Searching in a (2,4) tree with n items takes $O(\log n)$ time



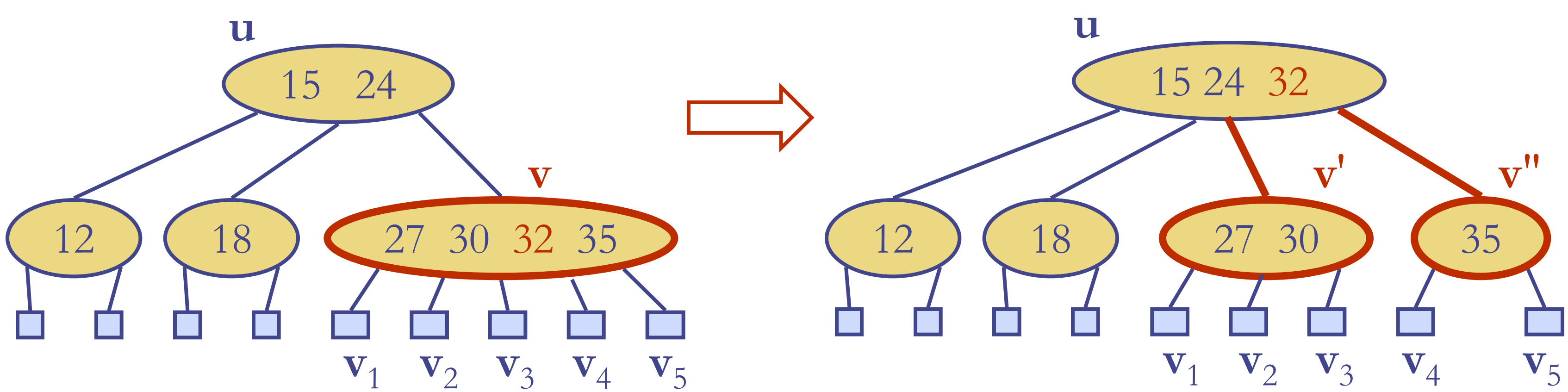
Insertion

- We insert a new item (**k**, **o**) at the parent **v** of the leaf reached by searching for **k**
  - We preserve the depth property but
  - We may cause an overflow (i.e., node **v** may become a 5-node)
- Example: inserting key 30 causes an overflow



Overflow and Split

- We handle an overflow at a 5-node **v** with a split operation:
  - let $\mathbf{v}_1 \ldots \mathbf{v}_5$ be the children of **v** and $\mathbf{k}_1 \ldots \mathbf{k}_4$ be the keys of **v**
  - node **v** is replaced nodes **v'** and **v"**
    - **v'** is a 3-node with keys $\mathbf{k}_1$ $\mathbf{k}_2$ and children $\mathbf{v}_1$ $\mathbf{v}_2$ $\mathbf{v}_3$
    - **v"** is a 2-node with key $\mathbf{k}_4$ and children $\mathbf{v}_4$ $\mathbf{v}_5$
  - key $\mathbf{k}_3$ is inserted into the parent **u** of **v** (a new root may be created)
- The overflow may propagate to the parent node **u**



Analysis of Insertion

Algorithm insertItem(k, o)

1. We search for key **k** to locate the insertion node **v**
2. We add the new item (**k**, **o**) at node **v**
3. **while overflow**(**v**)

    if isRoot(**v**)

        create a new empty root above **v**

    v ← **split**(**v**)

- Let **T** be a (2,4) tree with **n** items
  - Tree **T** has **O**(log **n**) height
  - Step 1 takes **O**(log **n**) time because we visit **O**(log **n**) nodes
  - Step 2 takes **O**(1) time
  - Step 3 takes **O**(log **n**) time because each split takes **O**(1) time and we perform **O**(log **n**) splits
- Thus, an insertion in a (2,4) tree takes **O**(log **n**) time

2-4 Tree: Insertion (A variation)

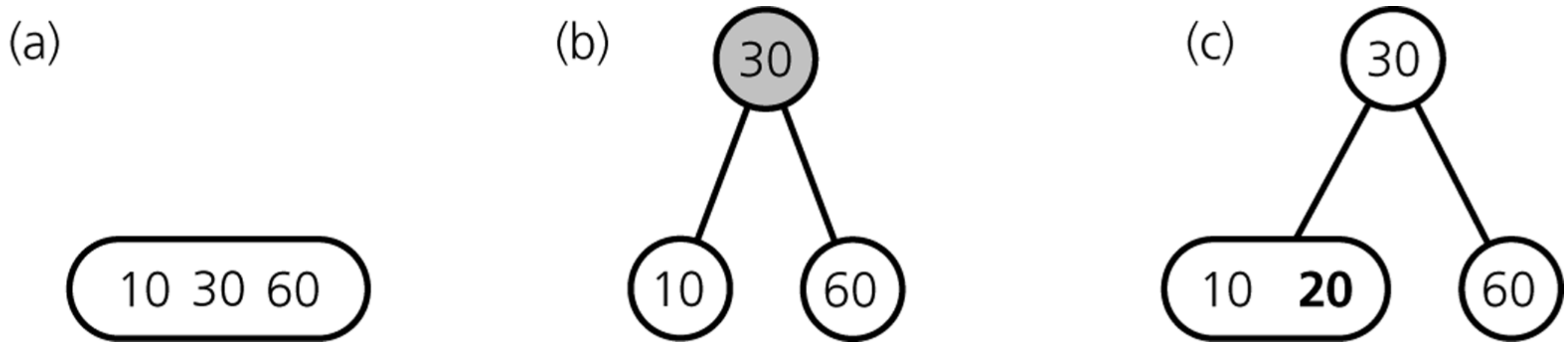
- Insertion procedure:
  - items are inserted at the leafs
  - since a 4-node cannot take another item, 4-nodes are split up during insertion process
- Strategy
  - on the way from the root down to the leaf:
    split up all 4-nodes "on the way"
  - insertion can be done in one pass

2-4 Tree: Insertion (A variation)

Insertion of 60, 30, 10, 20, 50, 40, 70, 80, 15, 90, 100

2-4 Tree: Insertion (A variation)

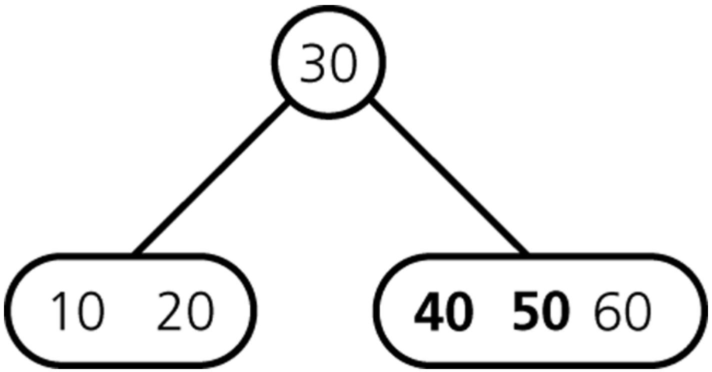
Inserting 60, 30, 10, 20 ...



Next ... 50, 40 ...

2-4 Tree: Insertion (A variation)

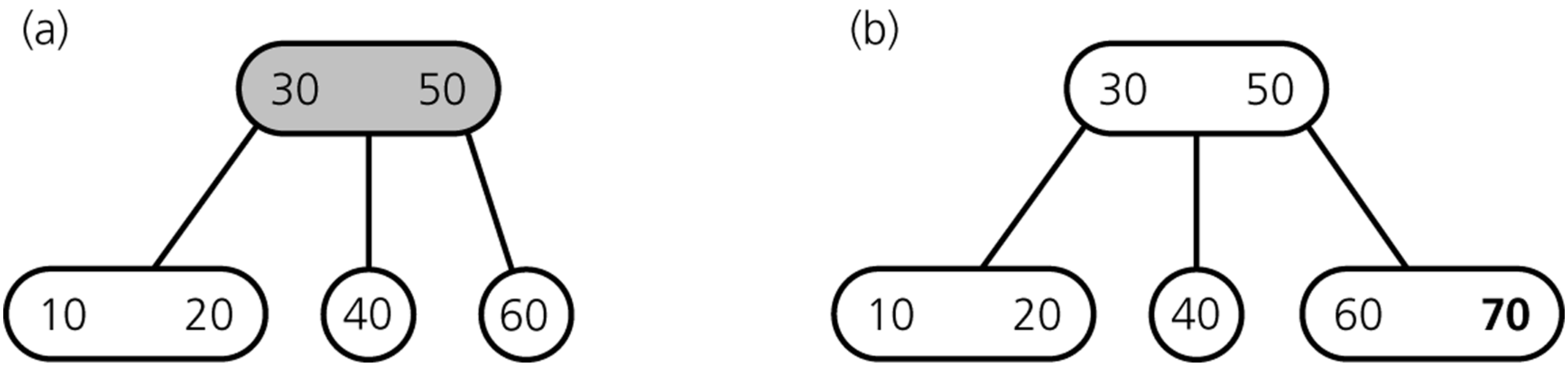
Inserting 50, 40 ...



Next ... 70, ...

2-4 Tree: Insertion (A variation)

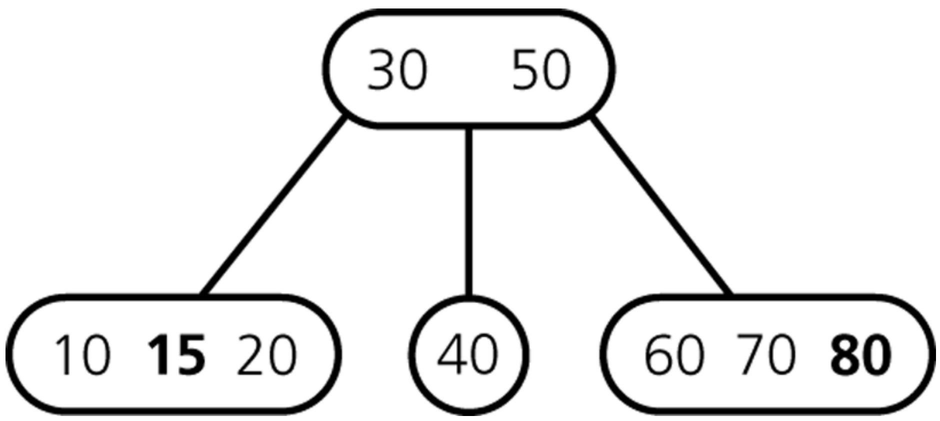
Inserting 70 ...



Next ... 80, 15 ...

2-4 Tree: Insertion (A variation)

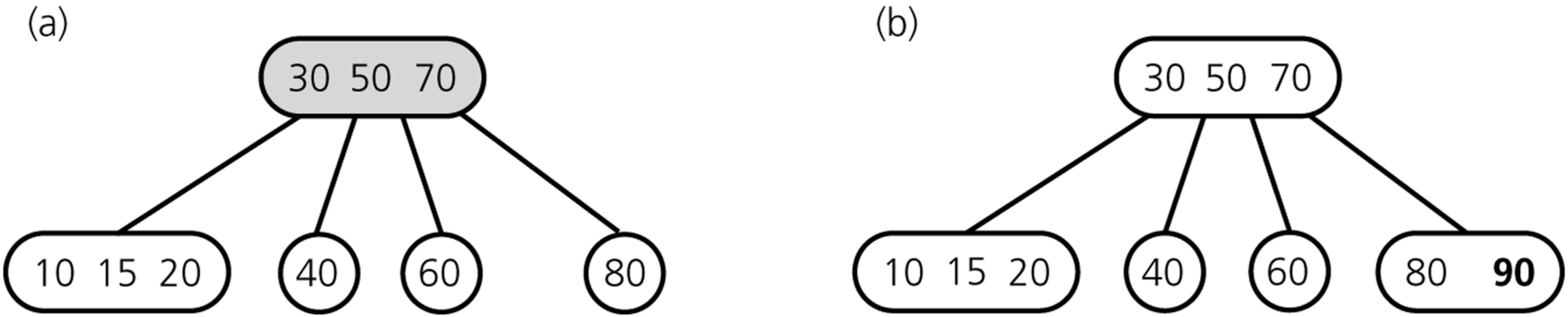
Inserting 80, 15 ...



Next: ... 90 ...

2-4 Tree: Insertion (A variation)

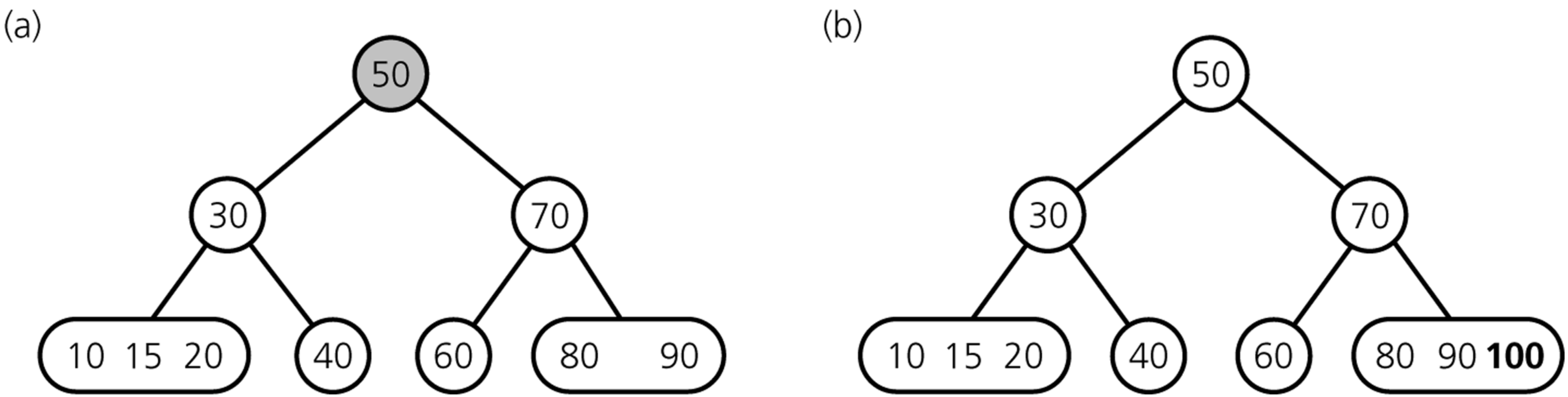
Inserting 90 ...



Next ... 100 ...

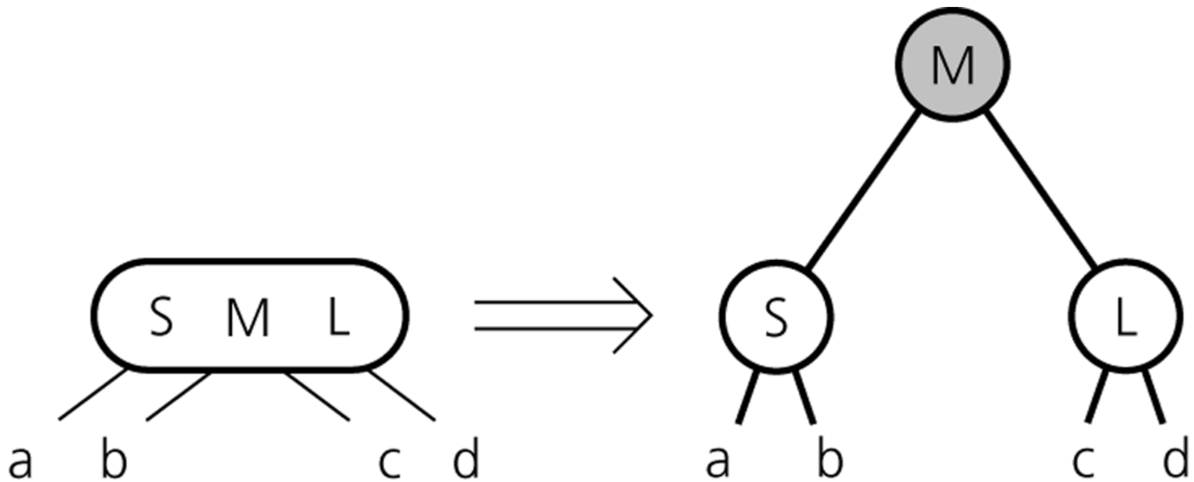
2-4 Tree: Insertion (A variation)

Inserting 100 ...



2-4 Tree: Insertion (A variation)

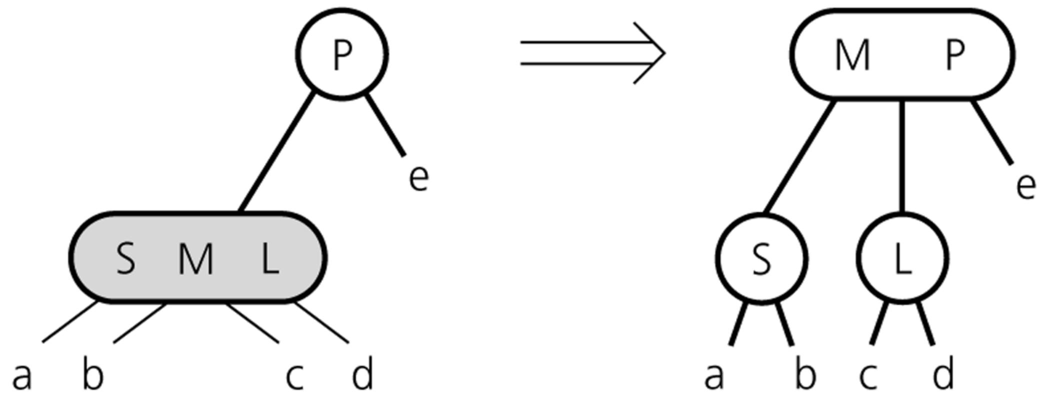
Splitting 4-nodes during Insertion



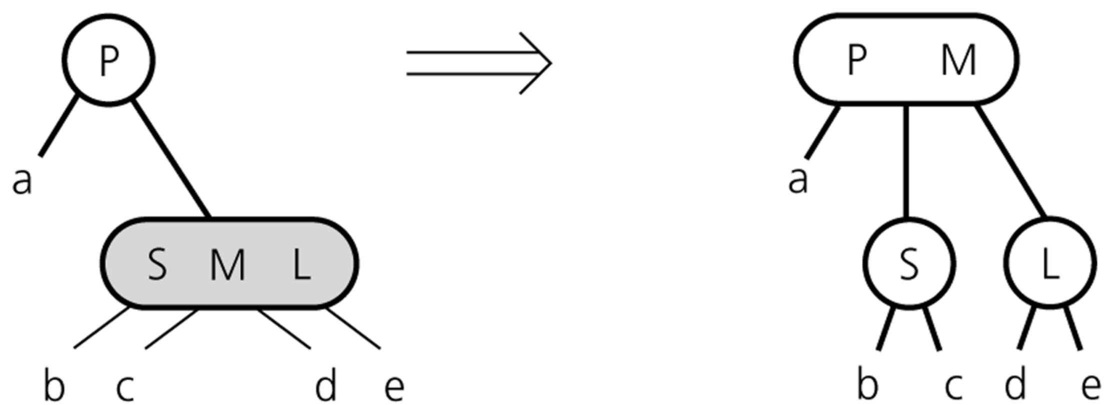
2-4 Tree: Insertion procedure

Splitting a 4-node whose parent is a 2-node during insertion

(a)

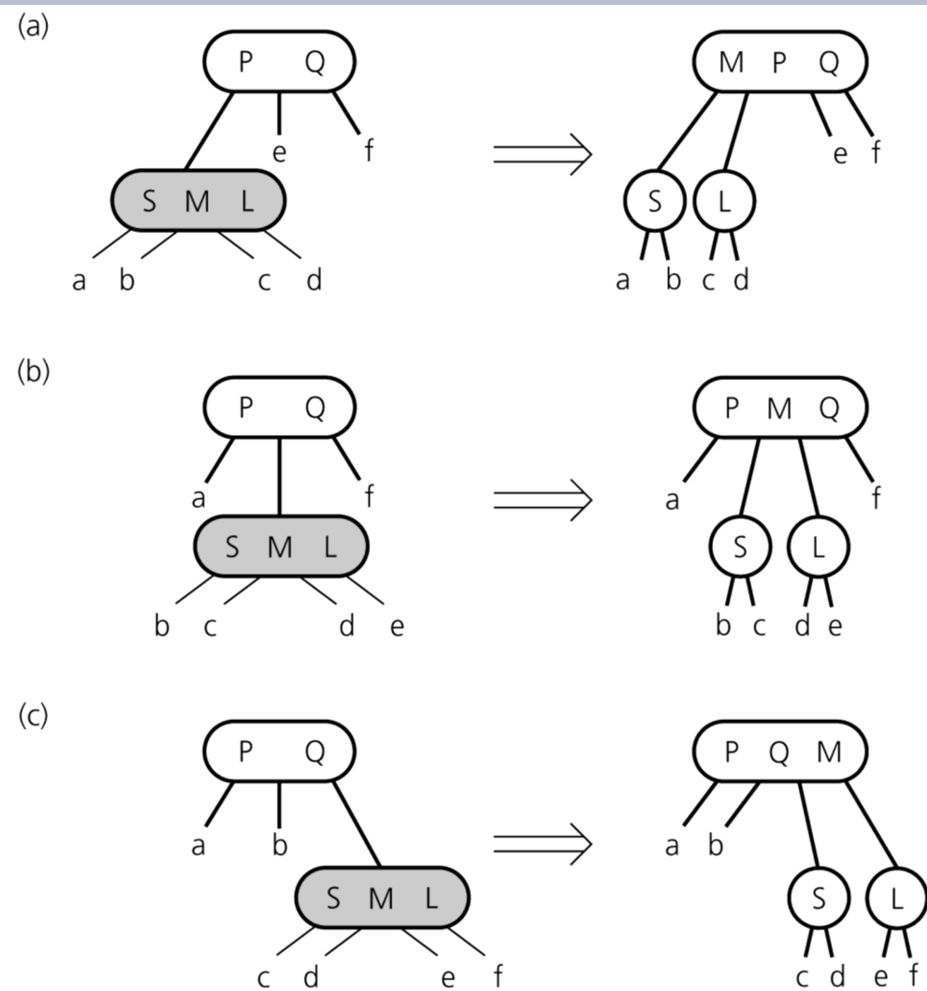


(b)



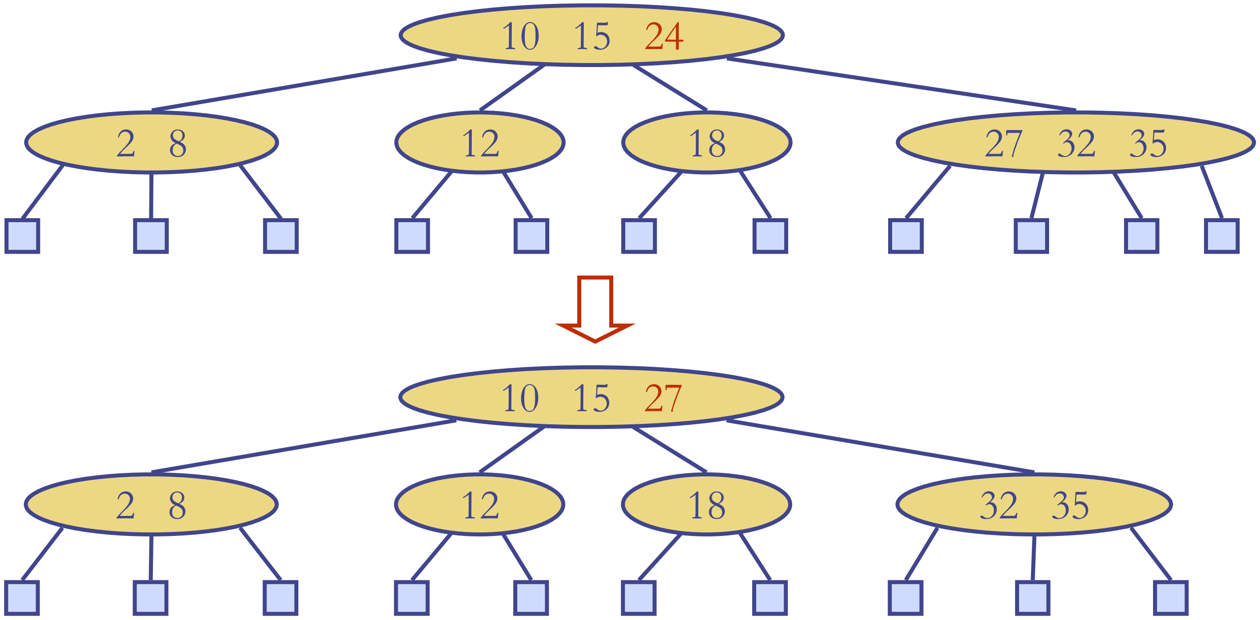
2-4 Tree: Insertion procedure

Splitting a 4-node whose parent is a 3-node during insertion



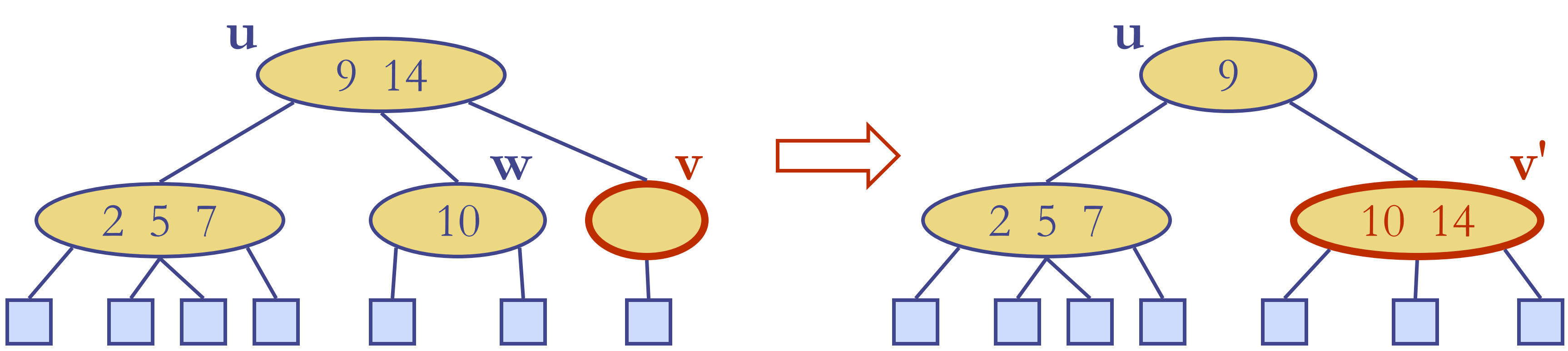
Deletion

- We reduce deletion of an item to the case where the item is at the node with leaf children
- Otherwise, we replace the item with its inorder successor (or, equivalently, with its inorder predecessor) and delete the latter item
- Example: to delete key 24, we replace it with 27 (inorder successor)



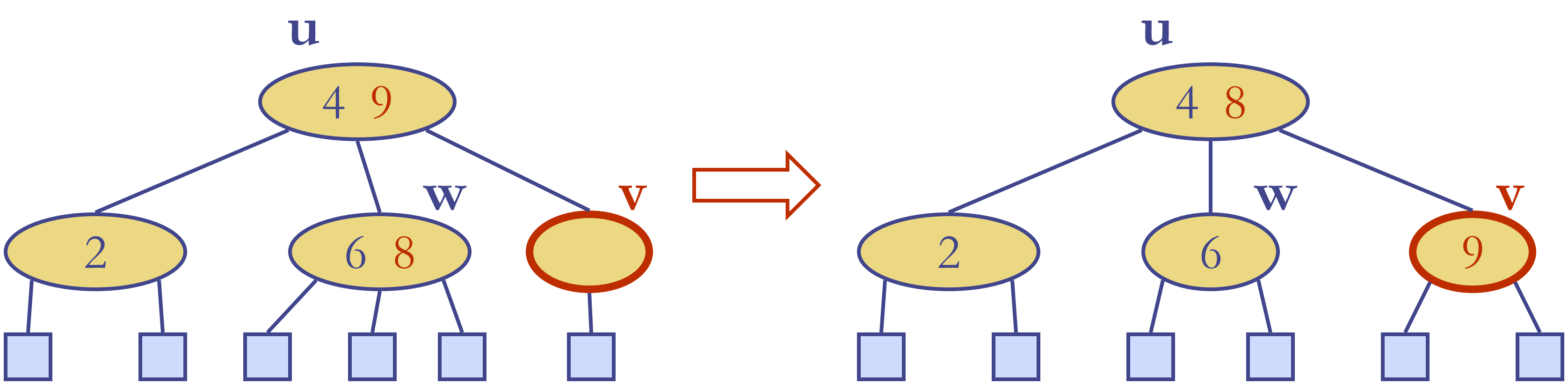
Underflow and Fusion

- Deleting an item from a node **v** may cause an underflow, where node **v** becomes a 1-node with one child and no keys
- To handle an underflow at node **v** with parent **u**, we consider two cases
- Case 1: **The adjacent siblings of v are 2-nodes**
  - Fusion operation: we merge **v** with an adjacent sibling **w** and move an item from **u** to the merged node **v'**
  - After a fusion, the underflow may propagate to the parent **u**



Underflow and Transfer

- Case 2: **an adjacent sibling w of v is a 3-node or a 4-node**
  - Transfer operation:
    1. we move a child of **w** to **v**
    2. we move an item from **u** to **v**
    3. we move an item from **w** to **u**
  - After a transfer, no underflow occurs



Analysis of Deletion

- Let **T** be a (2,4) tree with **n** items
  - Tree **T** has **O(log n)** height
- In a deletion operation
  - We visit **O**(log **n**) nodes to locate the node from which to delete the item
  - We handle an underflow with a series of **O(log n)** fusions, followed by at most one transfer
  - Each fusion and transfer takes **O(1)** time
- Thus, deleting an item from a (2,4) tree takes **O(log n)** time

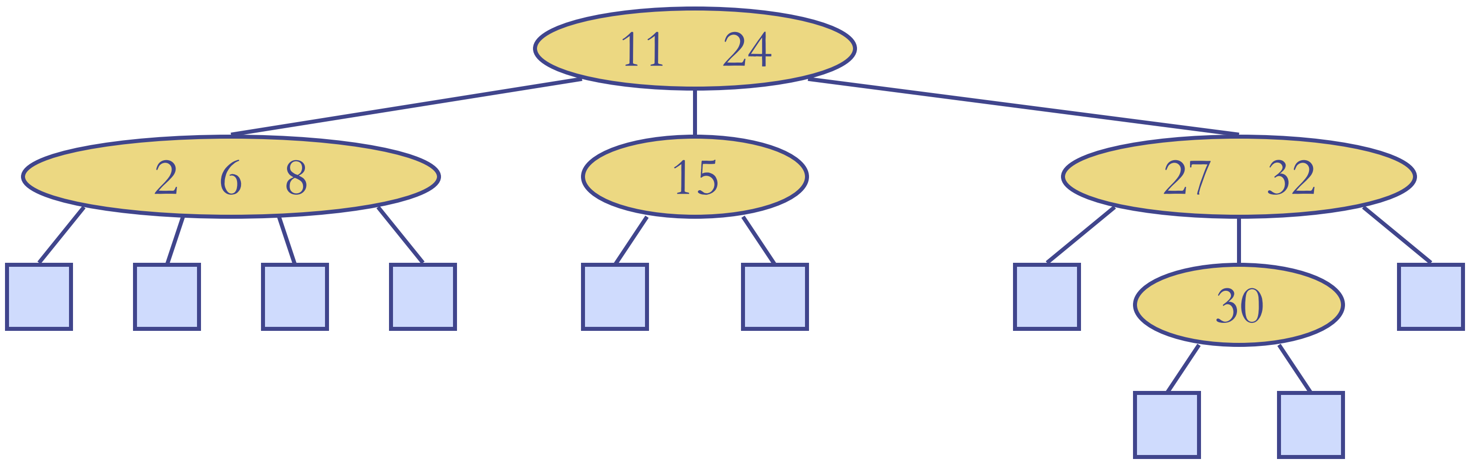
Implementing a Dictionary

- Comparison of efficient dictionary implementations

	Search	Insert	Delete	Notes
Hash Table	1 expected	1 expected	1 expected	◆ no ordered dictionary methods ◆ simple to implement
Skip List	log n high prob.	log n high prob.	log n high prob.	◆ randomized insertion ◆ simple to implement
(2,4) Tree	log n worst-case	log n worst-case	log n worst-case	◆ complex to implement

Multi-Way Search Tree

- A multi-way search tree is an ordered tree such that
  - Each internal node has at least two children and stores **d** -1 key-element items ($\mathbf{k_i}$, $\mathbf{o_i}$), where **d** is the number of children
  - For a node with children $\mathbf{v}_1\ \mathbf{v}_2 \ldots \mathbf{v_d}$ storing keys $\mathbf{k}_1\ \mathbf{k}_2 \ldots \mathbf{k_{d\text{-}1}}$
    - keys in the subtree of $\mathbf{v}_1$ are less than $\mathbf{k}_1$
    - keys in the subtree of $\mathbf{v_i}$ are between $\mathbf{k_{i\text{-}1}}$ and $\mathbf{k_i}$ ($\mathbf{i}$ = 2, …, $\mathbf{d}$ - 1)
    - keys in the subtree of $\mathbf{v_d}$ are greater than $\mathbf{k_{d\text{-}1}}$
  - The leaves store no items and serve as placeholders

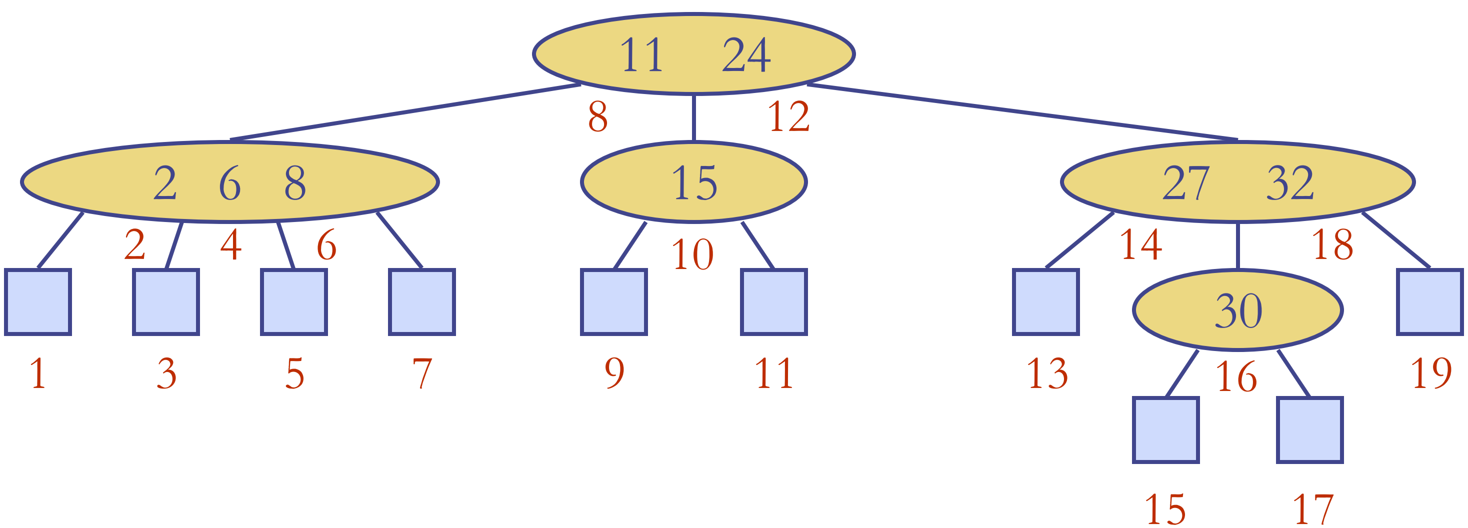


Multi-Way Searching

- Similar to search in a binary search tree
- A each internal node with children $\mathbf{v}_1\ \mathbf{v}_2 \ldots \mathbf{v}_\mathbf{d}$ and keys $\mathbf{k}_1\ \mathbf{k}_2 \ldots \mathbf{k}_{\mathbf{d}-1}$
  - $\mathbf{k} = \mathbf{k}_\mathbf{i}$ ($\mathbf{i} = 1, \ldots, \mathbf{d} - 1$): the search terminates successfully
  - $\mathbf{k} < \mathbf{k}_1$: we continue the search in child $\mathbf{v}_1$
  - $\mathbf{k}_{\mathbf{i}-1} < \mathbf{k} < \mathbf{k}_\mathbf{i}$ ($\mathbf{i} = 2, \ldots, \mathbf{d} - 1$): we continue the search in child $\mathbf{v}_\mathbf{i}$
  - $\mathbf{k} > \mathbf{k}_{\mathbf{d}-1}$: we continue the search in child $\mathbf{v}_\mathbf{d}$
- Reaching an external node terminates the search unsuccessfully

Multi-Way Inorder Traversal

- We can extend the notion of inorder traversal from binary trees to multi-way search trees
- Namely, we visit item ($\mathbf{k}_i$, $\mathbf{o}_i$) of node $\mathbf{v}$ between the recursive traversals of the subtrees of $\mathbf{v}$ rooted at children $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$
- An inorder traversal of a multi-way search tree visits the keys in increasing order



Concluding Remarks

- Advantage of 2-3 and 2-3-4 trees
  - Easy-to-maintain balance
- Allowing nodes with more than four children is counterproductive (for internal sorting)

Reference

- Algorithm Design: Foundations, Analysis, and Internet Examples. Michael T. Goodrich and Roberto Tamassia. John Wiley & Sons.
- Introduction to Algorithms. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.
- Data Abstraction and Problem Solving with Java™. Janet J. Prichard; Frank M. Carrano.

Thank you!