B-Trees

Lesson 6.4

Learning Objectives

LO 6.4.1 **Insert** correctly an element into a B-Tree

LO 6.4.2 **Delete** an element successfully in an existing B-Tree

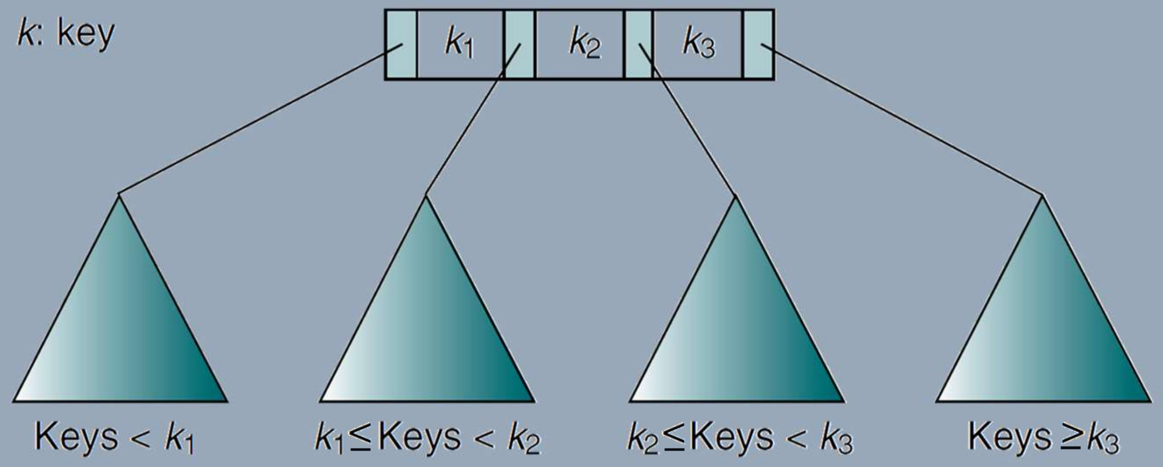
LO 6.4.3 **Compare** the difference of the operations between binary search trees and B-trees in terms of its operations

M-way Search Trees

- An **m-way tree** is a search tree in which each node can have from 0 to m subtrees, where m is defined as the **B-tree order**. Given a nonempty multiway tree, we can identify the following properties:

    1. Each node has 0 to m subtrees.

    2. A node with $k < m$ subtrees contains k subtrees and $k - 1$ data entries.



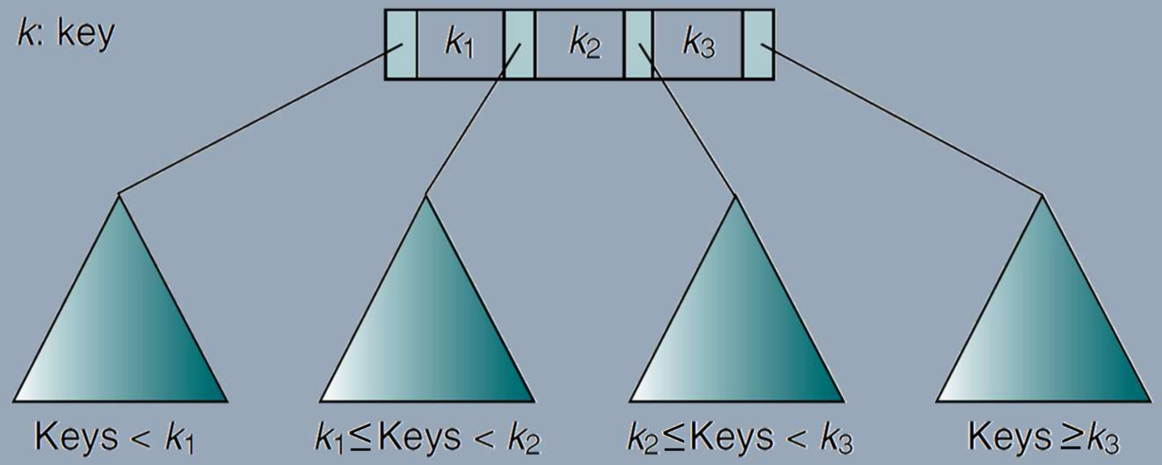
An m-way tree of order 4

M-way Search Trees

3. The key values in the first subtree are all less than the key value in the first entry; the key values in the other subtrees are all greater than or equal to the key value in their parent entry.

4. The keys of the data entries are ordered $key_1 \leq key_2 \leq \ldots \leq key_k$.

5. All subtrees are themselves multiway trees



An m-way tree of order 4

B-Trees

- An **b-tree** is an m-way search tree with the following additional properties:
  1. The root is either a leaf or it has 2…m subtrees.
  2. All internal nodes have at least $\lceil m / 2 \rceil$ non-null subtrees and at most m non-null subtrees.
  3. All leaf nodes are at the same level; that is, the tree is perfectly balanced.
  4. A leaf node has at least $\lceil m / 2 \rceil - 1$ and at most $m - 1$ entries

B-Tree Operations

- *Inserting an element into the B-Tree*
    1. Check whether tree is empty.
    2. If the tree is empty, then create a new node with new key value and insert it into the tree as a root node.
    3. If the tree is not empty, then find the suitable leaf node to which the new key value is added using BST logic.
    4. If that leaf node has empty position, add the new key value to that leaf node in ascending order of key value within the node.

B-Tree Operations

- *Inserting an element into the B-Tree*

5. If that leaf node is already full, split that leaf node by sending middle value to its parent node. Repeat the same until the sending value is fixed into a node.

6. If the splitting is performed at root node then the middle value becomes new root node for the tree and the height of the tree is increased by one.

LO 6.4.1 Insert correctly an element into a B-Tree

Insert the following respectively into an initially empty B-Tree in different orders: (a) order-3, (b) order-4, and (c) order-5.

26, 8, 13, 5, 10, 25, 2, 4, 9, 16, 18, 19, 24, 15, 17, 21, 12, 14, 1, 11

B-Tree Operations

- *Deleting an element from the B-Tree*

  Let k be the key to be deleted, x the node containing the key then, there are 3 possible case for deletion in B-Tree:

  1. Case 1. If the key, k, is already in a leaf node, and removing it doesn't cause that leaf node to have too few keys (*the node will not be empty*), then simply remove the key to be deleted.

B-Tree Operations

- *Deleting an element from the B-Tree*

  Let k be the key to be deleted, x the node containing the key then, there are 3 possible case for deletion in B-Tree:

2. <u>Case 2</u>. If k is in node x and x is an internal node, there are three cases to consider:

   a. If the child y that precedes k in node x has at least t keys (*more than the minimum*), then find the predecessor key k' in the subtree rooted at y. Recursively delete k' and replace k with k' in x

   b. Symmetrically, if the child z that follows k in node x has at least t keys, find the successor k' and delete and replace as before. Note that finding k' and deleting it can be performed in a single downward pass.

   c. Otherwise, if both y and z have only $t - 1$ (minimum number) keys, merge k and all of z into y, so that both k and the pointer to z are removed from x. y now contains $2t - 1$ keys, and subsequently k is deleted.

B-Tree Operations

- *Deleting an element from the B-Tree*

  Let k be the key to be deleted, x the node containing the key then, there are 3 possible case for deletion in B-Tree:

3. Case 3. If key k is not present in an internal node x, determine the root of the appropriate subtree that must contain k. If the root has only t – 1 keys, execute either of the following two cases to ensure that we descend to a node containing at least t keys. Finally, recurse to the appropriate child of x.

   a. If the root has only t −1 keys but has a sibling with t keys, give the root an extra key by moving a key from x to the root, moving a key from the roots immediate left or right sibling up into x, and moving the appropriate child from the sibling to x.

   b. If the root has only t−1 keys but has a sibling with t keys, give the root an extra key by moving a key from x to the root, moving a key from the roots immediate left or right sibling up into x, and moving the appropriate child from the sibling to x.

LO 6.4.2 Delete an element successfully in an existing B-Tree

Delete the following respectively into the AVL BST you have just previously finished inserting:

12, 18, 13, 25

LO 6.4.3 Compare the difference of the operations between binary search trees and B-trees in terms of its operations

How does the insertion, searching, and deletion functions of BST differ from those of B-Trees?