DESIGN AND ANALYSIS OF ALGORITHMS

Surabhi Narayan

Department of Computer Science & Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

B Trees

Surabhi Narayan
Department of Computer Science & Engineering

B Tree

- Self-balancing search tree
- Each node can contain more than one key
- Can have more than two children.
- All data records (or record keys) are stored at the leaves, in increasing order of the keys each parental node contains $n - 1$ ordered keys.

It is also known as a height-balanced m-way tree.

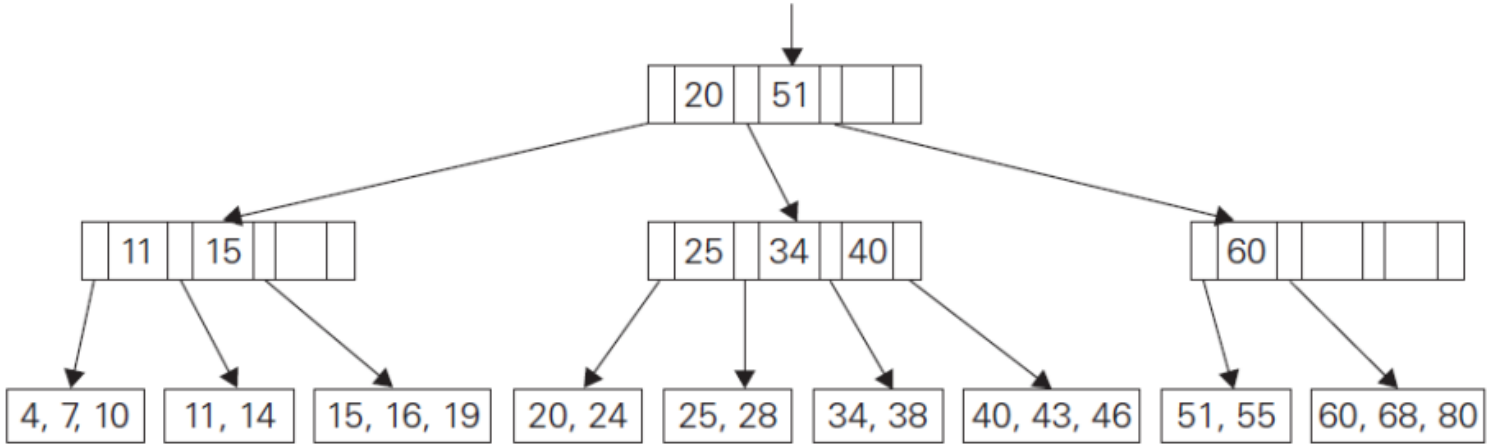


FIGURE 7.8 Example of a B-tree of order 4.

B Tree

Structural properties:

- All leaves are at the same level.
- For each node x, the keys are stored in increasing order.
- If n is the order of the tree, each internal node can contain at most n - 1 keys along with a pointer to each child.
- Each node except root can have at most n children and at least n/2 children.
- All leaves have the same depth (i.e. height-h of the tree).
- The root has at least 2 children and contains a minimum of 1 key.
- If $n \geq 1$, then for any n-key B-tree of height h and minimum degree $t \geq 2$, $h \geq \log_t (n+1)/2$
- B-Tree grows and shrinks from the root which is unlike Binary Search Tree. Binary Search Trees grow downward and also shrink from downward.
- Like other balanced Binary Search Trees, time complexity to search, insert and delete is O(log n).

B Tree

Operations on a B Tree:

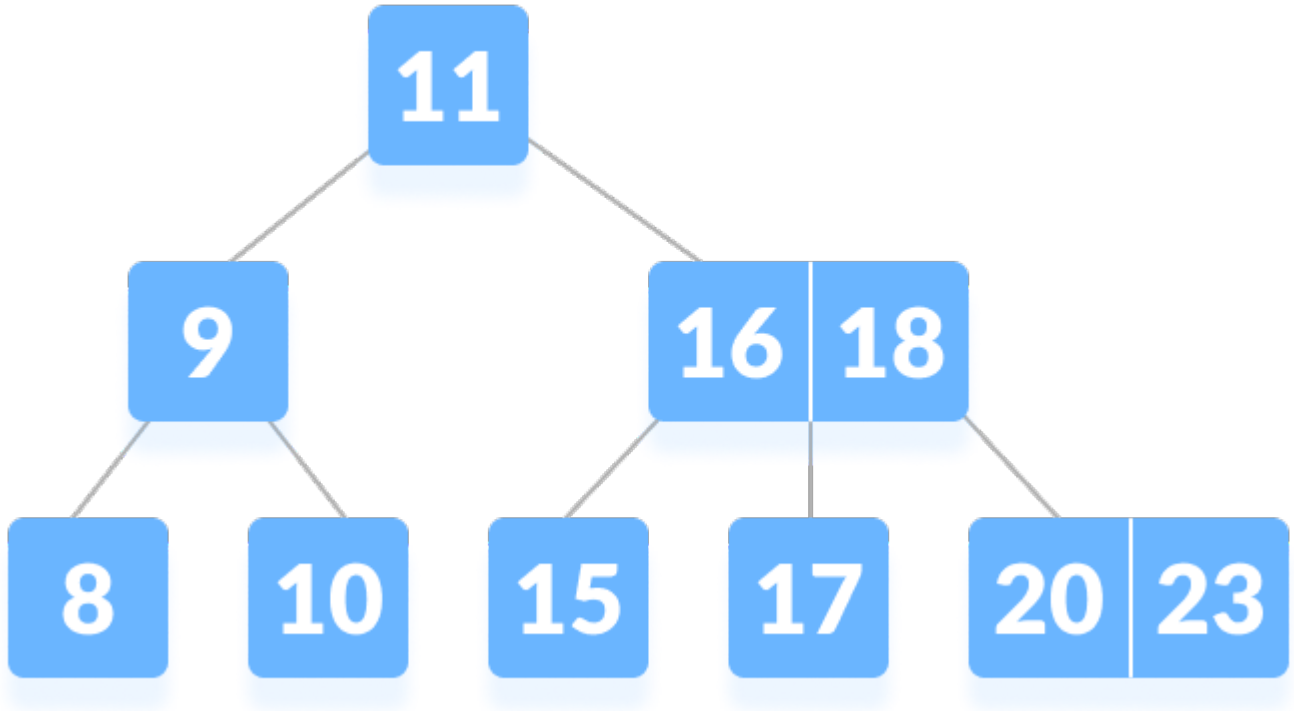
- Searching
- Insertion
- Deletion

Search

1. Starting from the root node, compare k with the first key of the node.
2. If k = the first key of the node, return the node and the index
3. If k.leaf = true, return NULL, i.e. not found
4. If k < the first key of the root node, search the left child of this key recursively.
5. If there is more than one key in the current node and k > the first key compare k with the next key in the node.
6. If k < next key search the left child of this key (ie. k lies in between the first and the second keys). Else, search the right child of the key
7. Repeat steps 1 to 4 until the leaf is reached.

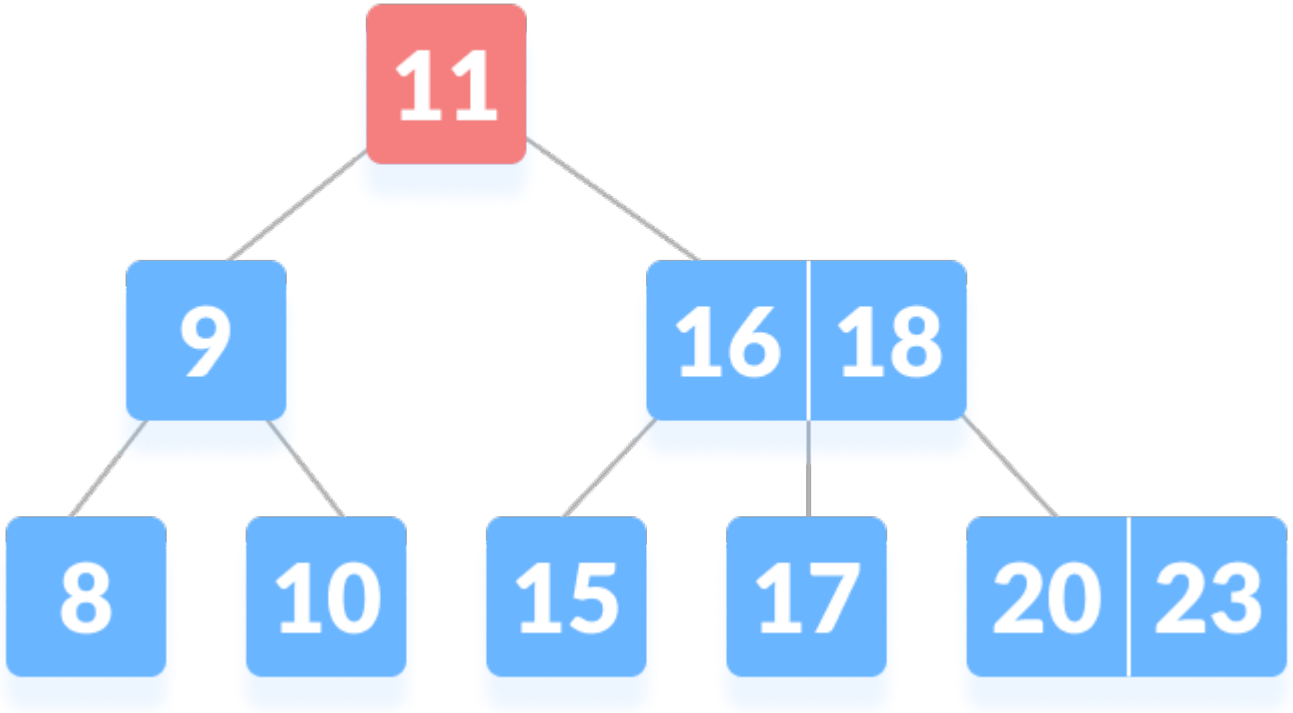
B Tree

Let us search key , k = 17



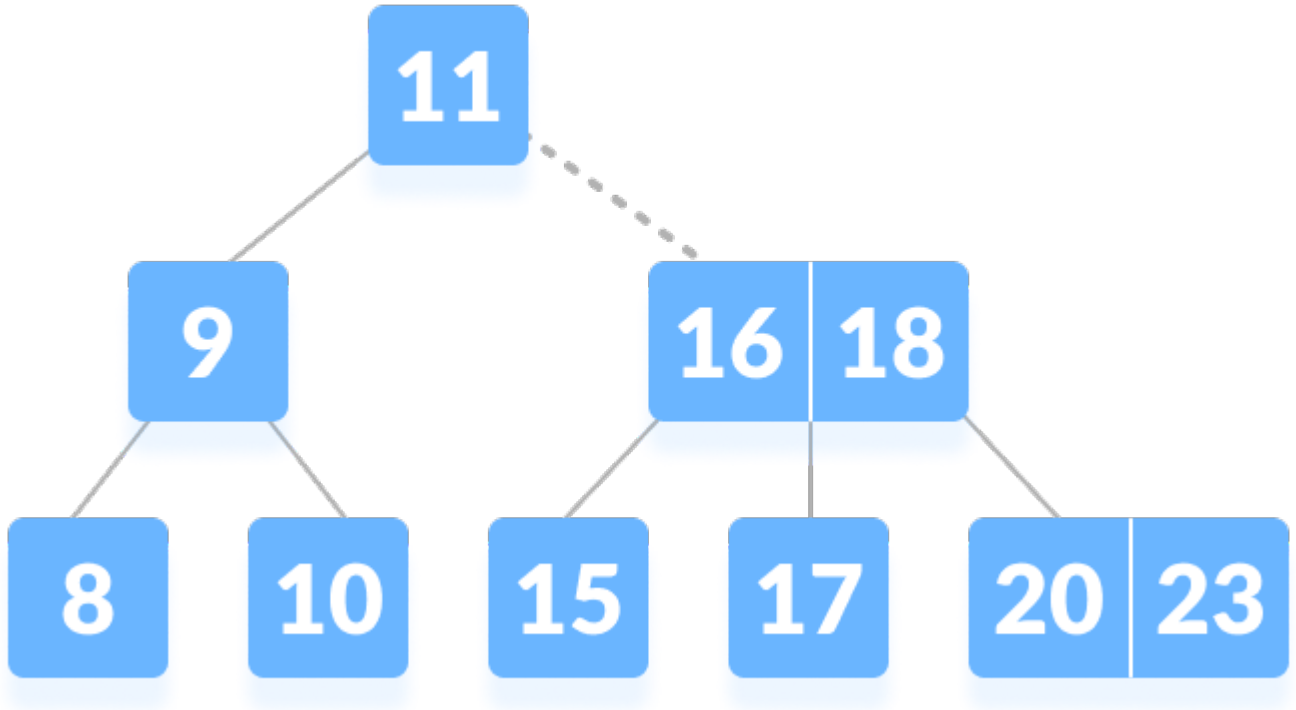
B Tree

K is not found in the root so, compare it with the root key



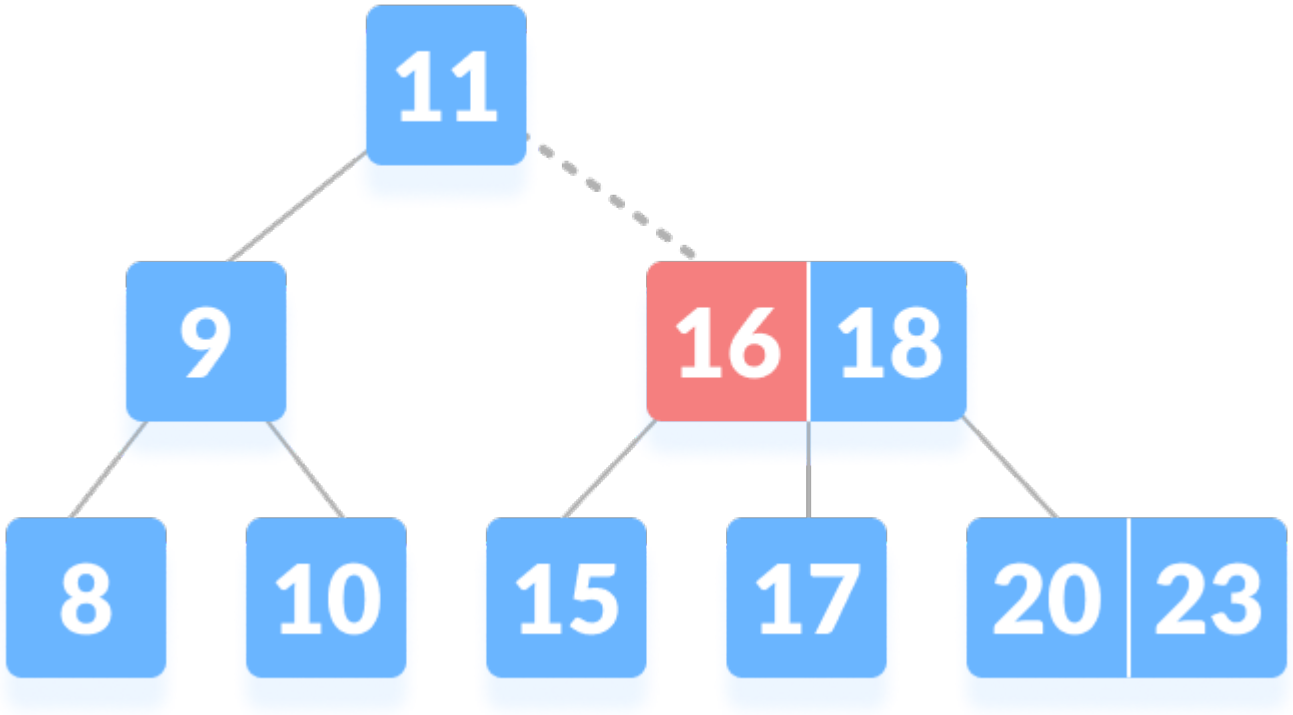
B Tree

Since k > 11 go to the right child of the root node.



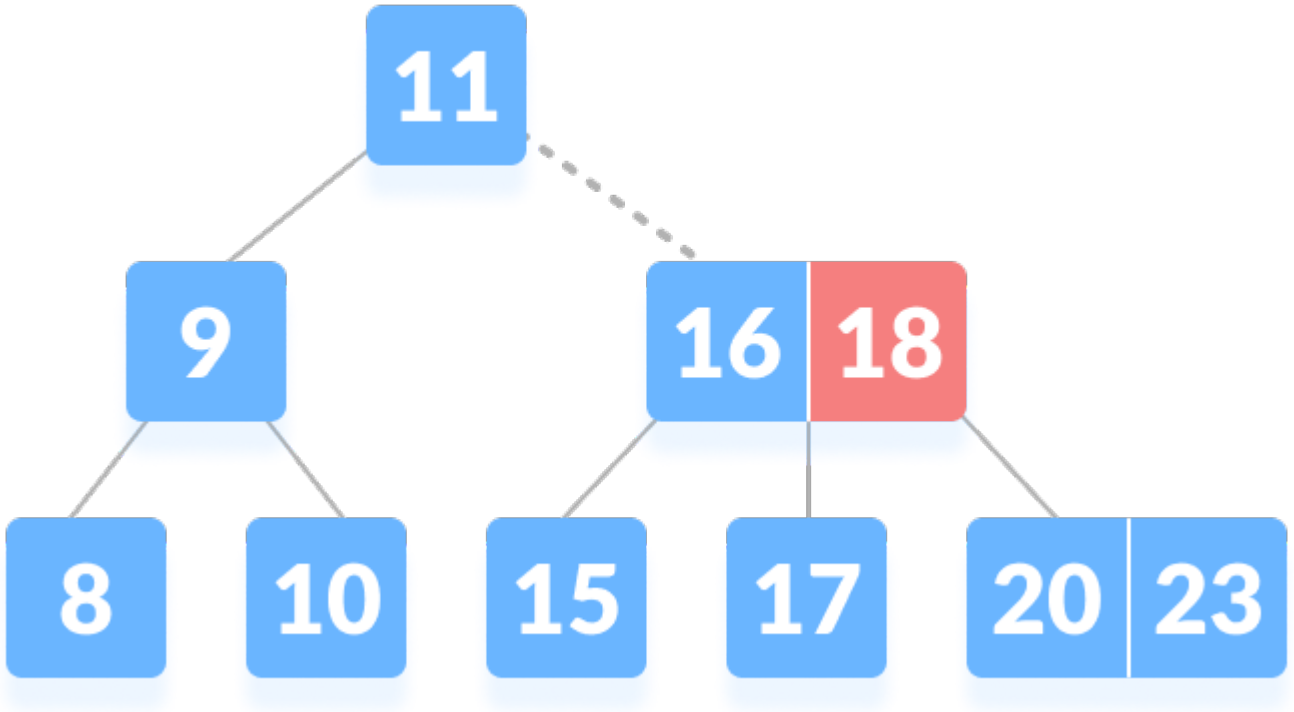
B Tree

Compare k with 16. Since k > 16, compare k with the next key 18



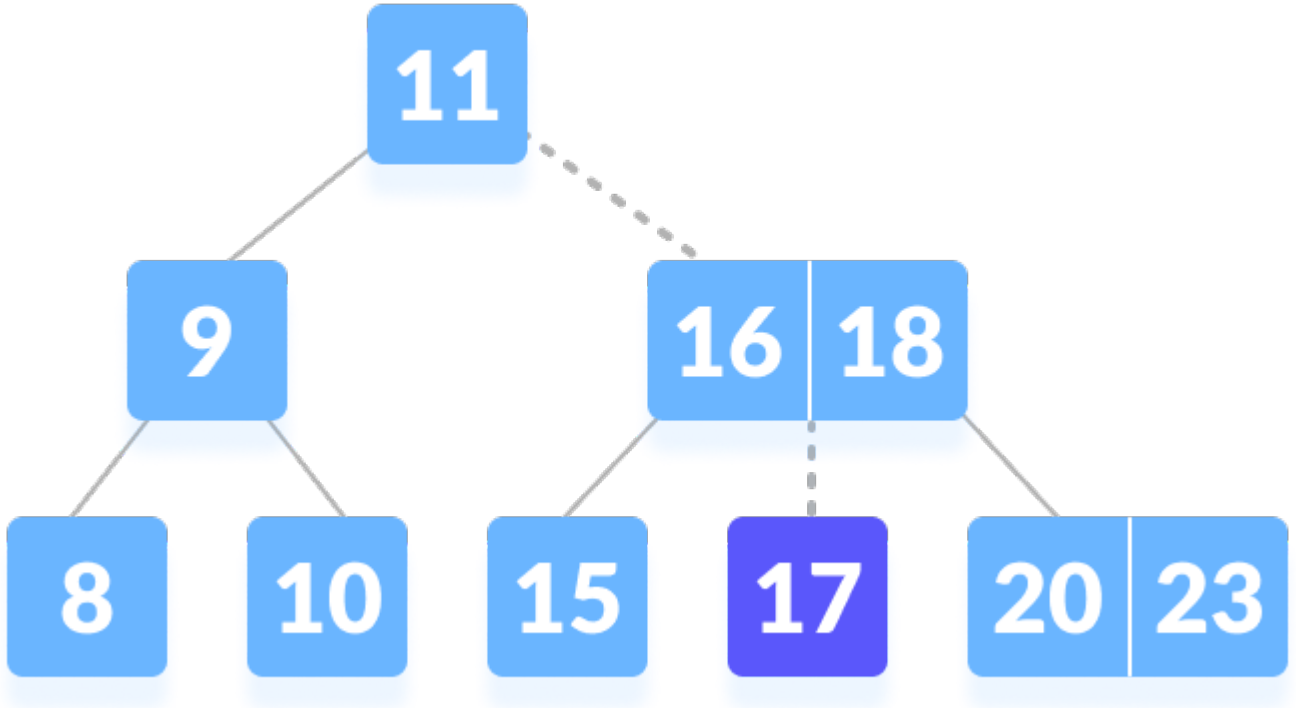
B Tree

Since k < 18, k lies between 16 and 18. Search in the right child of 16 or the left child of 18.



B Tree

k is found



Insertion into a B-tree

Inserting an element on a B-tree consists of two events: **searching the appropriate node** to insert the element and **splitting the node** if required.Insertion operation always takes place in the bottom-up approach.

Insertion Operation

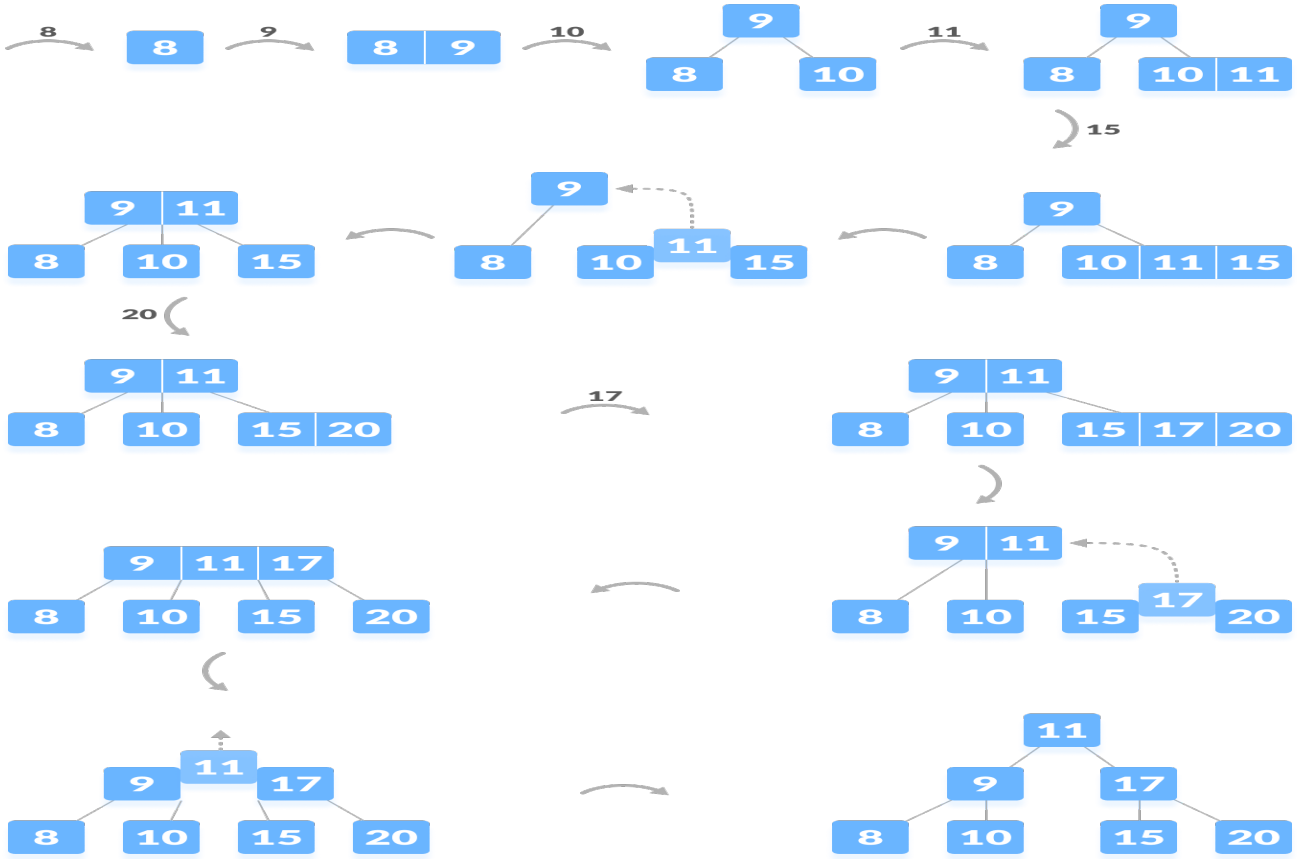
1.Traverse the B Tree in order to find the appropriate leaf node at which the node can be inserted.
2.If the leaf node contain less than m-1 keys then insert the element in the increasing order.

B Tree

3. Else, if the leaf node contains m-1 keys, then follow the following steps.
   - Insert the new element in the increasing order of elements.
   - Split the node into the two nodes at the median.
   - Push the median element upto its parent node.
   - If the parent node also contain m-1 number of keys, then split it too by following the same steps.

DESIGN AND ANALYSIS OF ALGORITHMS

B Tree



Insertion Example

Let us understand the insertion operation with the illustrations below.
The elements to be inserted are 8, 9, 10, 11, 15, 16, 17, 18, 20, 23.

B Tree

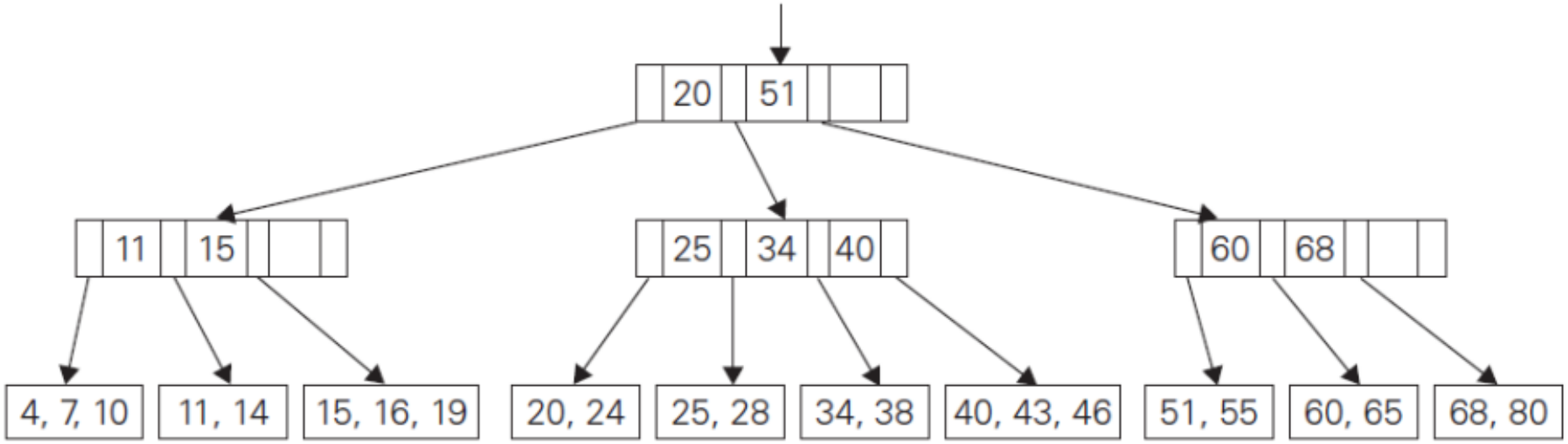


FIGURE 7.9 B-tree obtained after inserting 65 into the B-tree in Figure 7.8.

THANK YOU

Surabhi Narayan
Department of Computer Science &Engineering

surabhinarayan@pes.edu