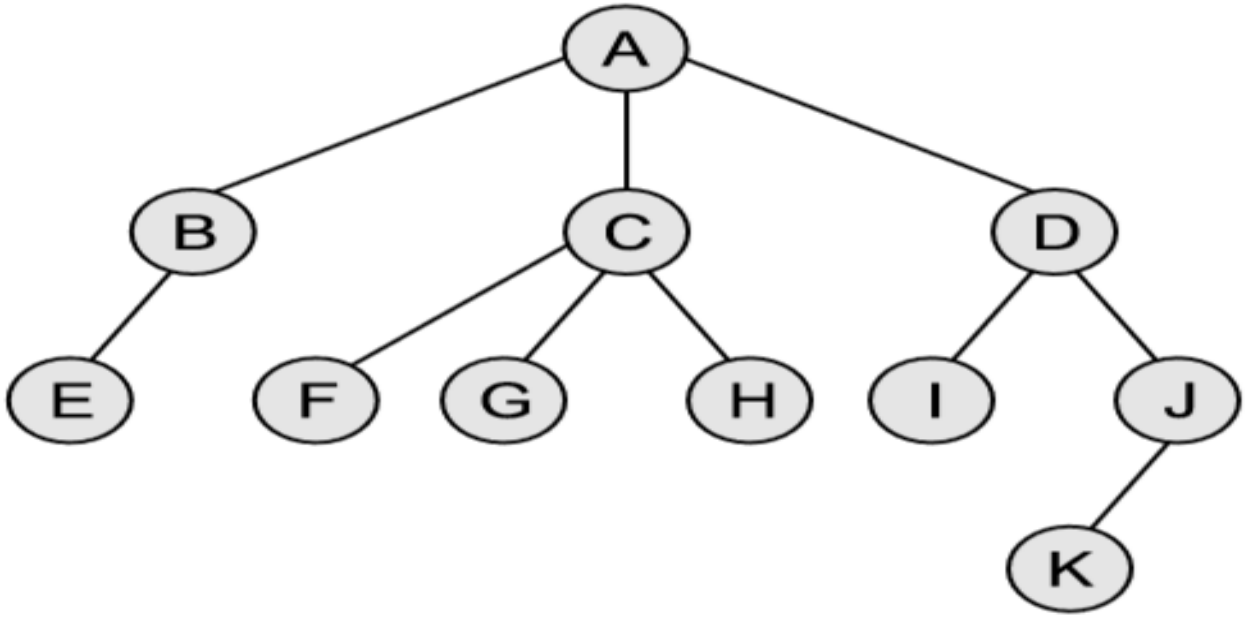
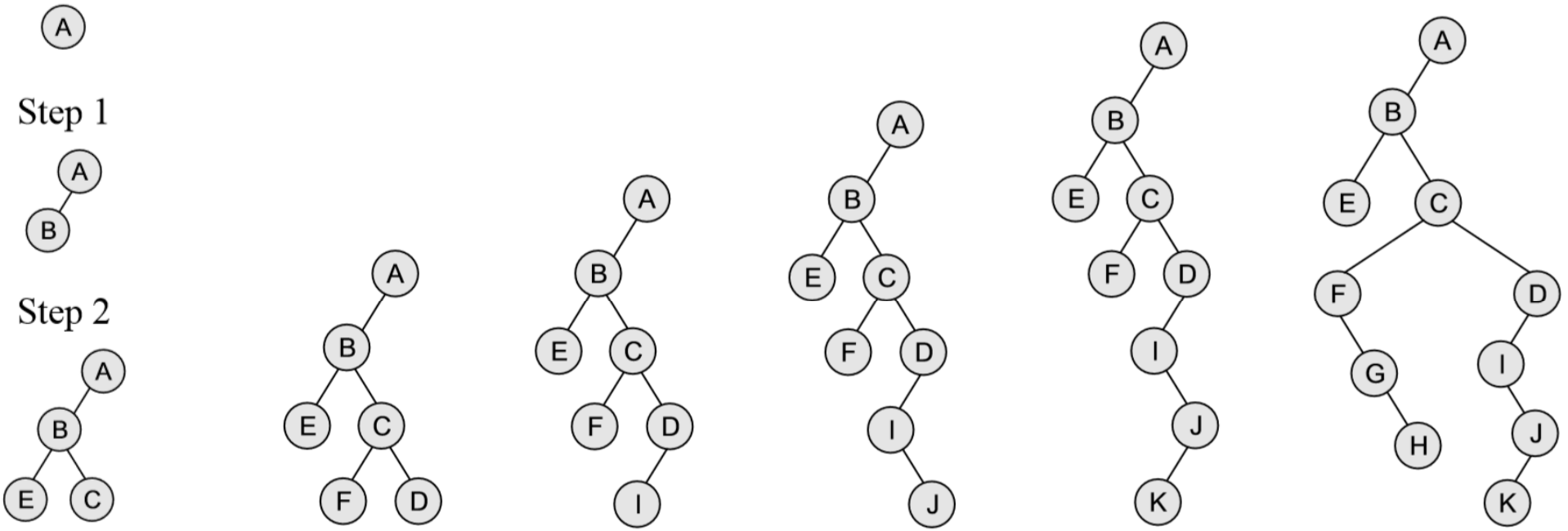
DSA UNIT – 5
TREES

DEPARTMENT OF ICT ENGINEERING
MARWADI UNIVERSITY
RAJKOT

GENERAL TREE to BINARY TREE



GENERAL TREE to BINARY TREE



Constructing a Binary Tree from Traversal Results

In-order Traversal: D B E A F C G

Pre-order Traversal: A B D E C F G

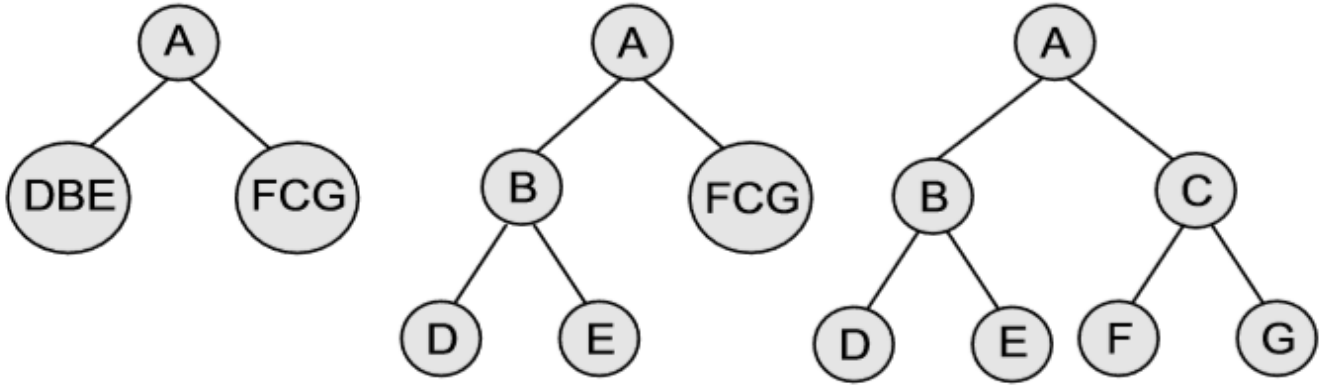


Figure 9.20

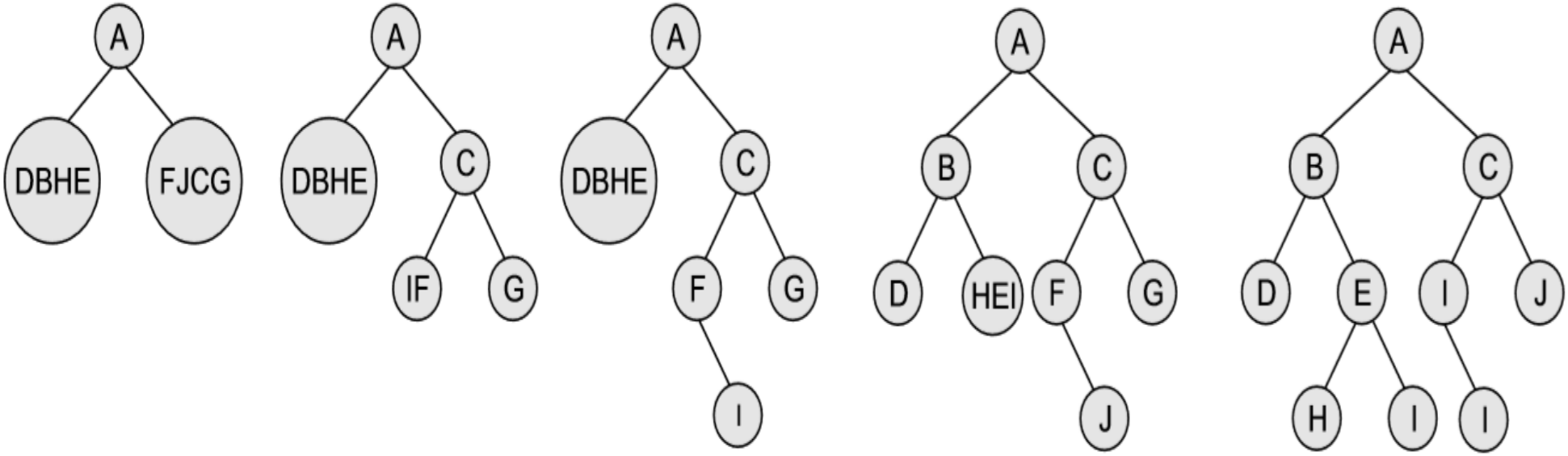
Constructing a Binary Tree from Traversal Results

In–order Traversal: D B H E I A F J C G

Post order Traversal: D H I E B J F G C A

Create Binary Tree

Constructing a Binary Tree from Traversal Results



The Huffman Algorithm

- Given n nodes and their weights, the Huffman algorithm is used to find a tree with a minimum weighted path length.
- The process essentially begins by creating a new node whose children are the two nodes with the smallest weight, such that the new node's weight is equal to the sum of the children's weight. That is, the two nodes are merged into one node. This process is repeated until the tree has only one node.
- Such a tree with only one node is known as the **Huffman tree**.

Step 1: Create a leaf node for each character. Add the character and its weight or frequency of occurrence to the priority queue.

Step 2: Repeat Steps 3 to 5 while the total number of nodes in the queue is greater than 1.

Step 3: Remove two nodes that have the lowest weight (or highest priority).

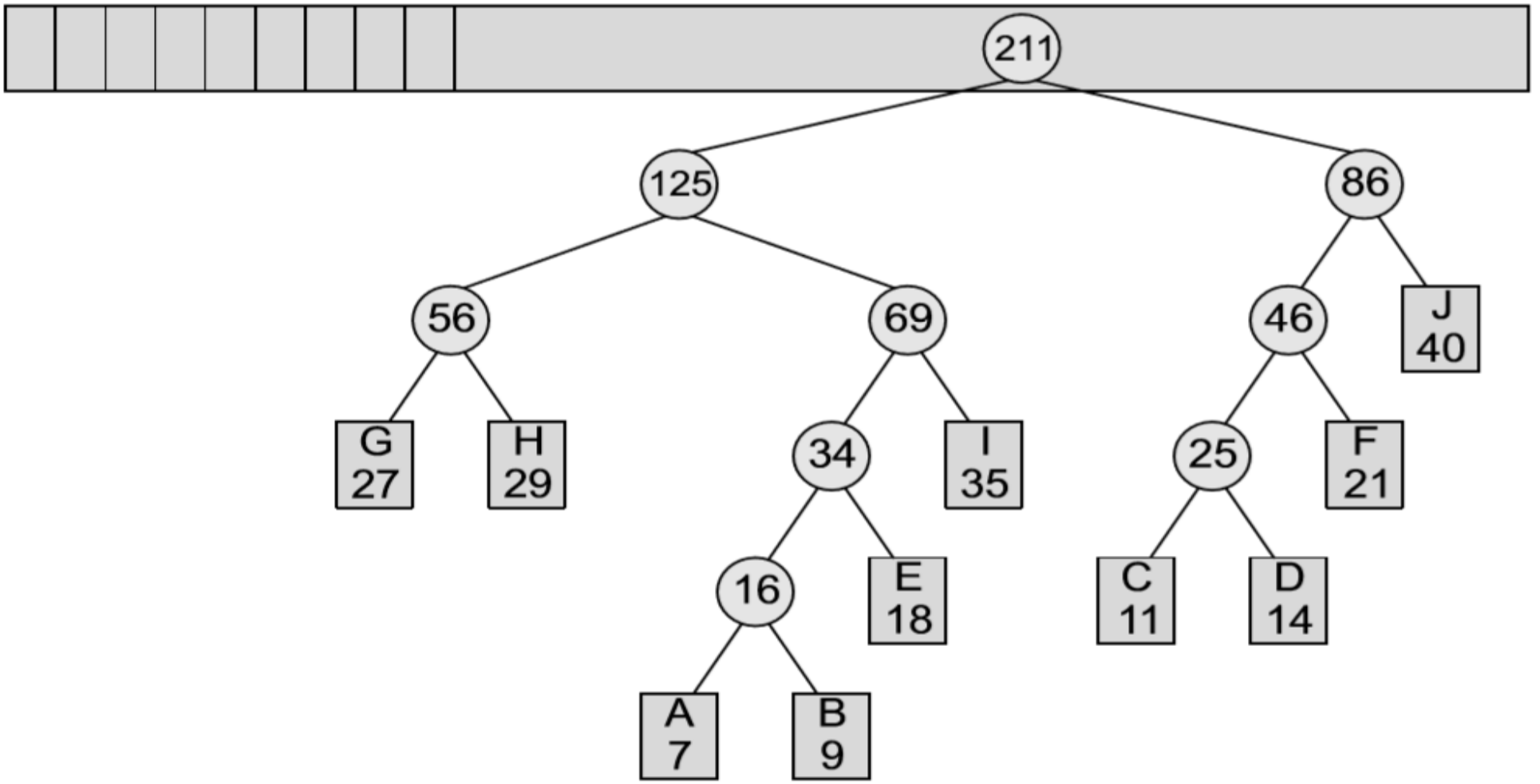
Step 4: Create a new internal node by merging these two nodes as children and with weight equal to the sum of the two nodes' weights.

Step 5: Add the newly created node to the queue.

The Huffman Algorithm

Example 9.10 Create a Huffman tree with the following nodes arranged in a priority queue.

A	B	C	D	E	F	G	H	I	J
7	9	11	14	18	21	27	29	35	40

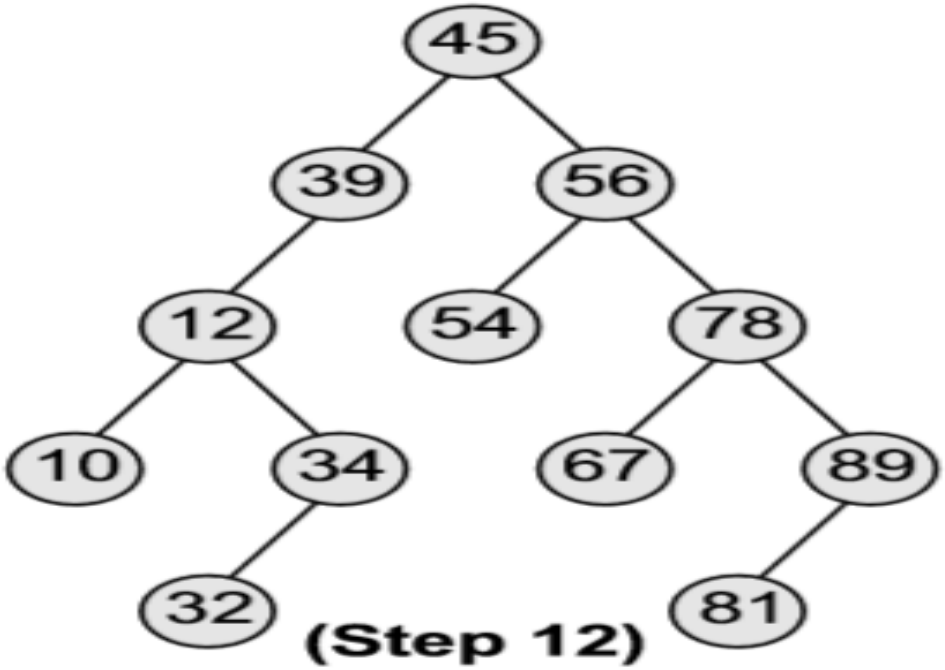

211
125
86
56
69
46
J
40
G
27
H
29
34
I
35
25
F
21
16
E
18
C
11
D
14
A
7
B
9

Binary Search Tree

Create a binary search tree using the following data elements:

45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81

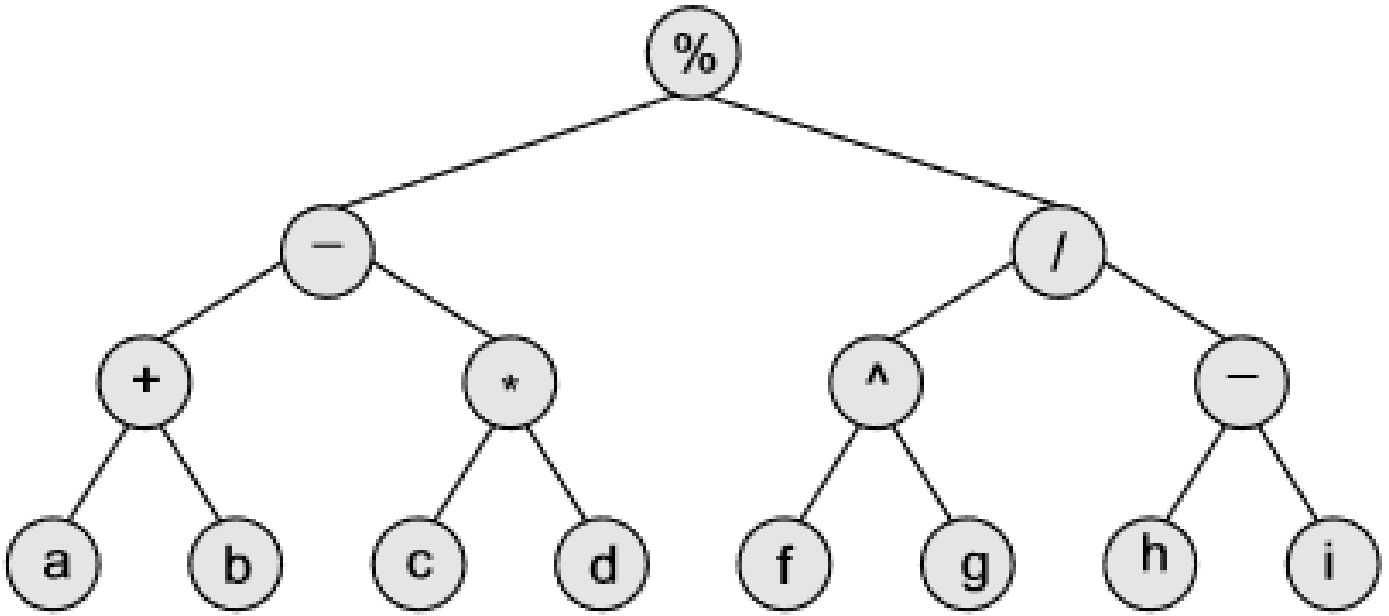
Binary Search Tree



Expression tree

Exp = ((a + b) – (c * d)) % ((e ^f) / (g – h))

Expression tree



Expression tree

Threaded Binary Tree

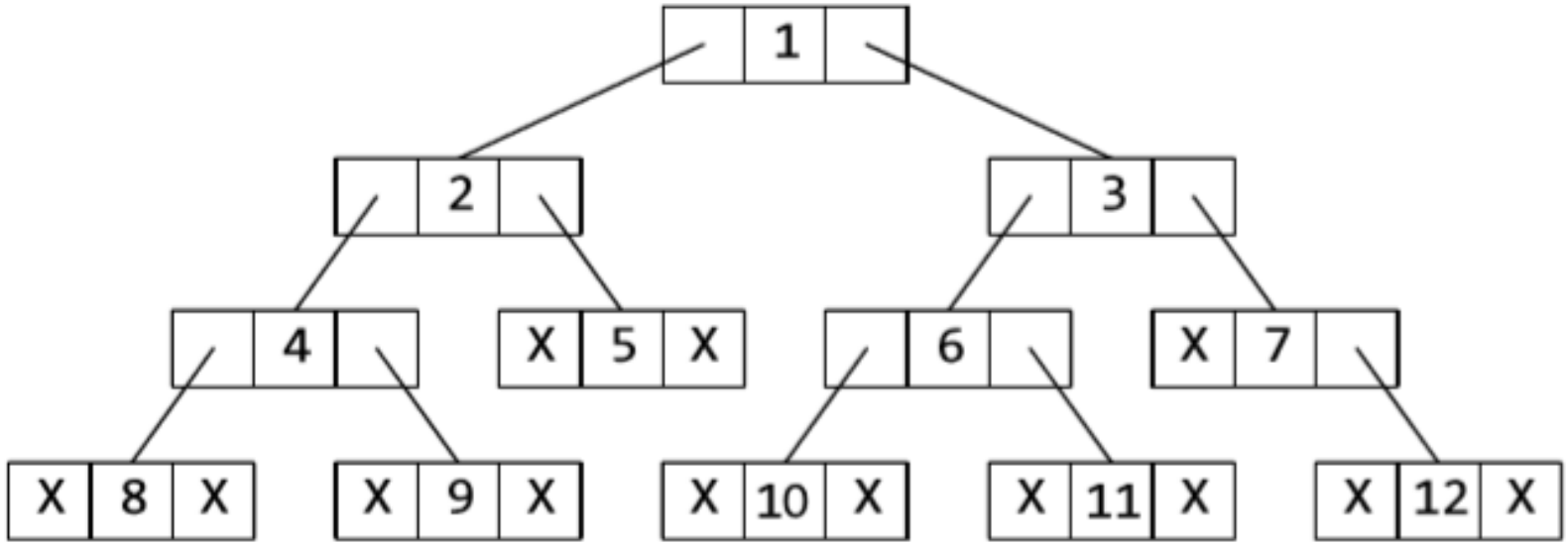
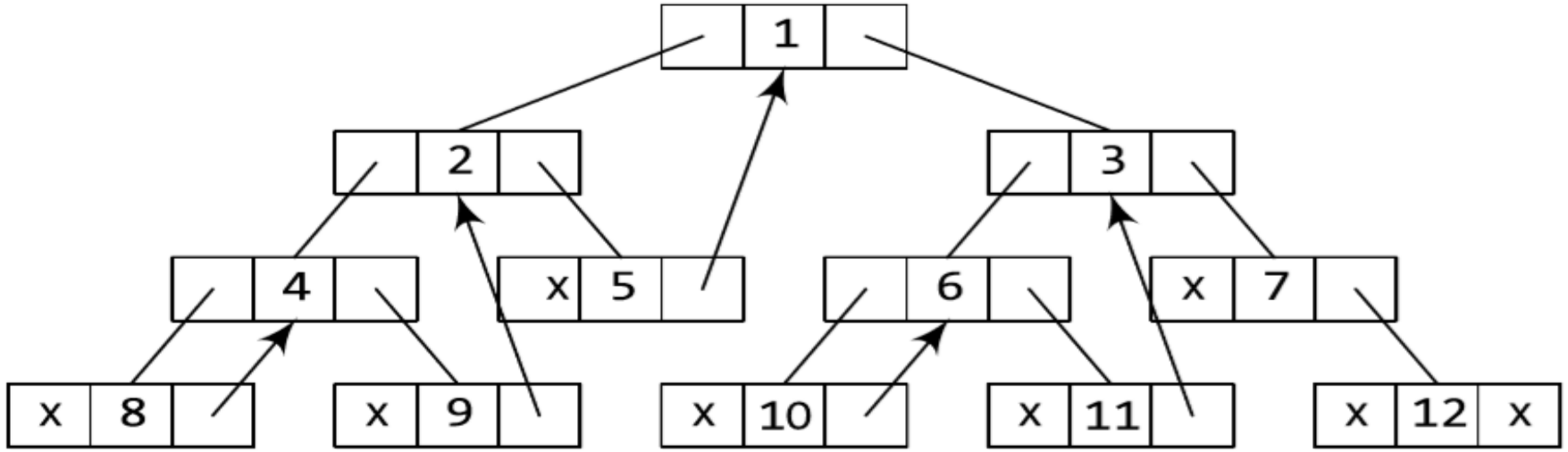


Figure 10.28 Linked representation of a binary tree

Threaded Binary Tree



These special pointers are called threads and binary trees containing threads are called threaded trees.

Threaded Binary Tree

Advantages of Threaded Binary Tree

- It enables linear traversal of elements in the tree.
- Linear traversal eliminates the use of stacks which in turn consume a lot of memory space and computer time.
- It enables to find the parent of a given element without explicit use of parent pointers.
- Since nodes contain pointers to in-order predecessor and successor, the threaded tree enables forward and backward traversal of the nodes as given by in-order fashion.

AVL TREE

- AVL tree is a self-balancing binary search tree invented by G.M. **A**delson - **V**elsky and E.M. **L**andis in 1962.
- The tree is named **AVL** in honour of its inventors.
- In an AVL tree, the heights of the two sub-trees of a node may differ by at most one.
- Due to this property, the AVL tree is also known as a **height-balanced tree**.

Multiway Search TREE

- A multiway tree is a tree that can have **more than two children**.
- A multiway tree of **order m** (or an m-way tree) is one in which a tree can have **m children**.
- Example : **AVL, B-TREE, B+ TREE**

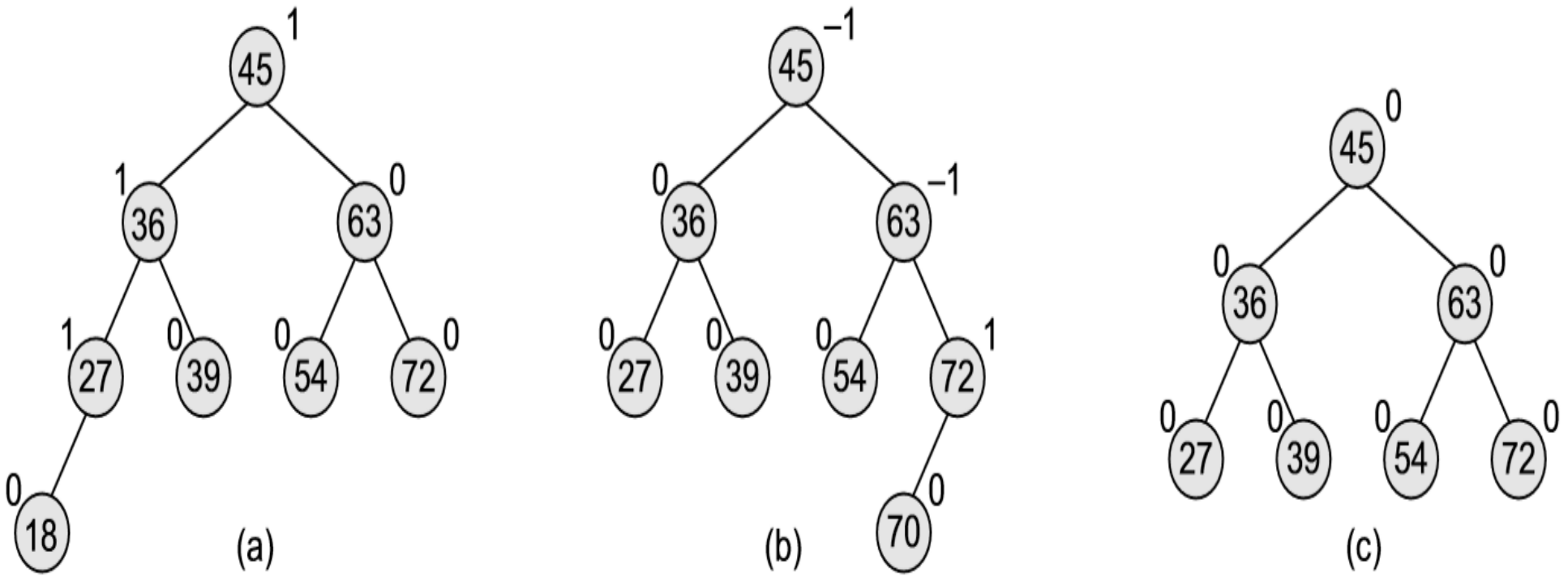


Figure 10.35 (a) Left-heavy AVL tree, (b) right-heavy tree, (c) balanced tree

AVL TREE

- Initially, the node was heavy (either left or right) and the new node has been inserted in the heavy sub-tree, thereby creating an unbalanced sub-tree. Such a node is said to be a **critical node**.

AVL TREE

The four categories of rotations are:

LL rotation : The new node is inserted in the left sub-tree of the left sub-tree of the critical node.

RR rotation : The new node is inserted in the right sub-tree of the right sub-tree of the critical node.

LR rotation : The new node is inserted in the right sub-tree of the left sub-tree of the critical node.

RL rotation : The new node is inserted in the left sub-tree of the right sub-tree of the critical node.

AVL TREE

https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

B TREE

- A B-tree of order m is a multiway search tree in which:
- Every node in the B tree has at most (maximum) m children.
- Every node in the B tree except the root node and leaf nodes has at least (minimum) m/2 children. This condition helps to keep the tree bushy so that the path from the root node to the leaf is very short, even in a tree that stores a lot of data.
- The root node has at least two children if it is not a terminal (leaf) node.
- All leaf nodes are at the same level

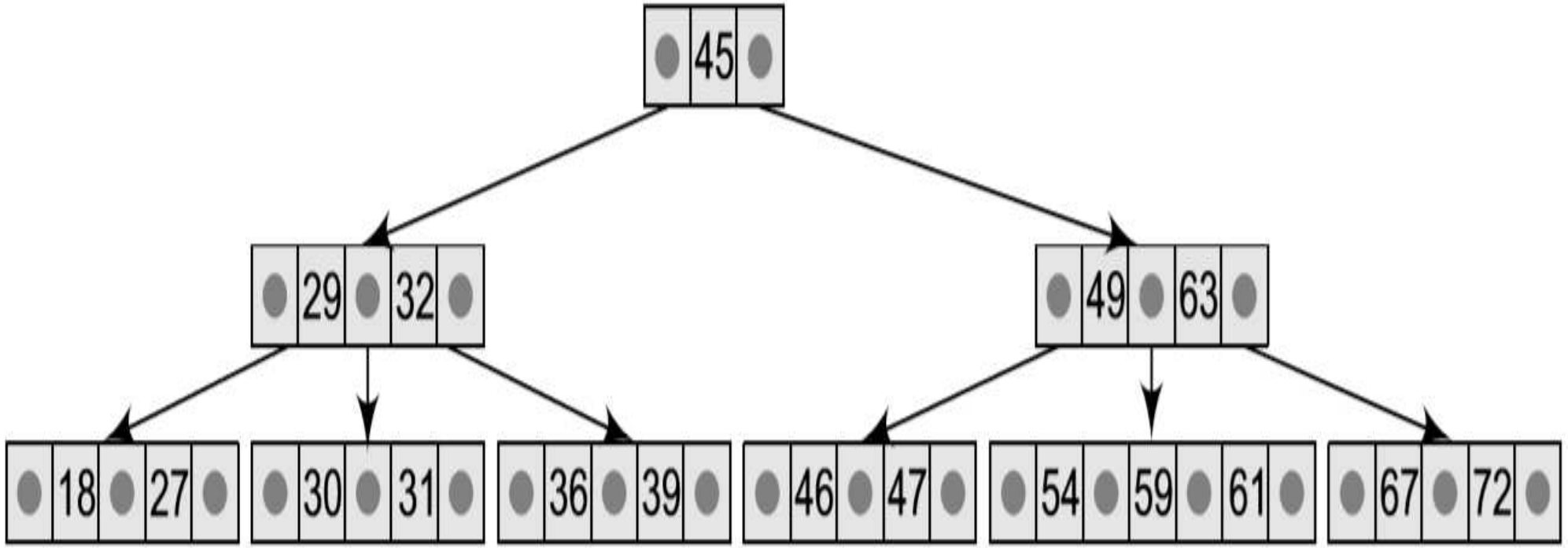
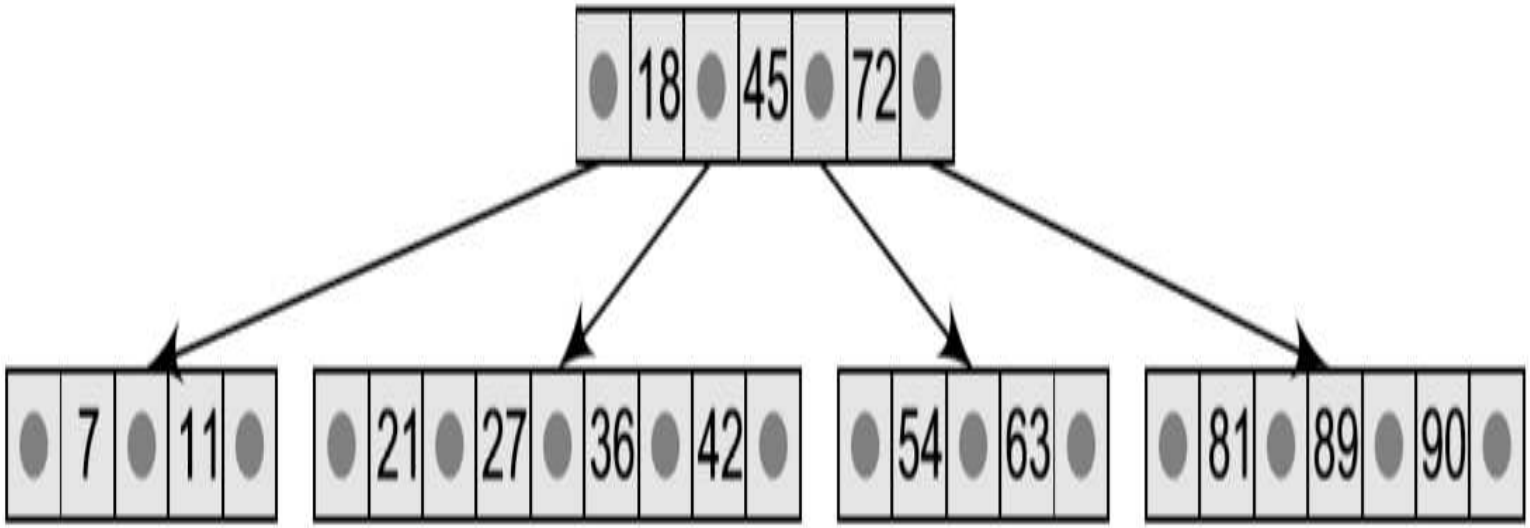
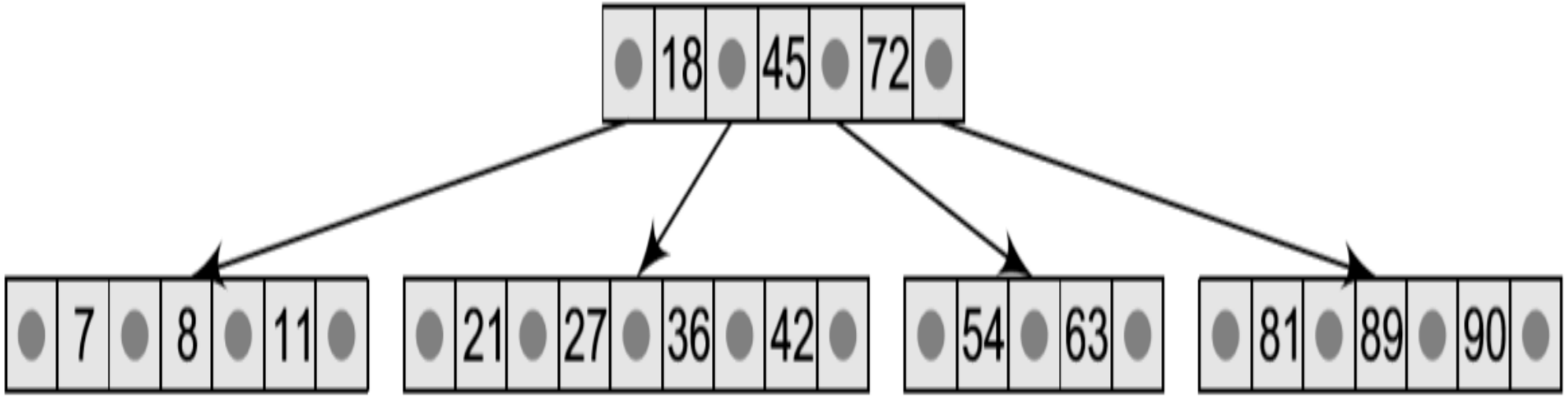


Figure 11.4 B tree of order 4

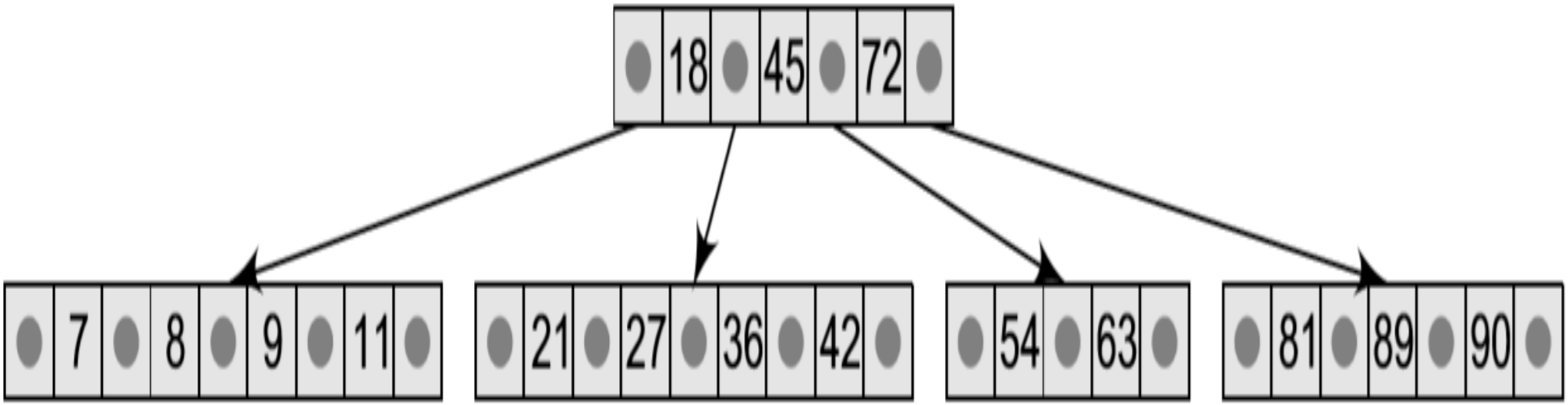
Example 11.1 Look at the B tree of order 5 given below and insert 8, 9, 39, and 4 into it.



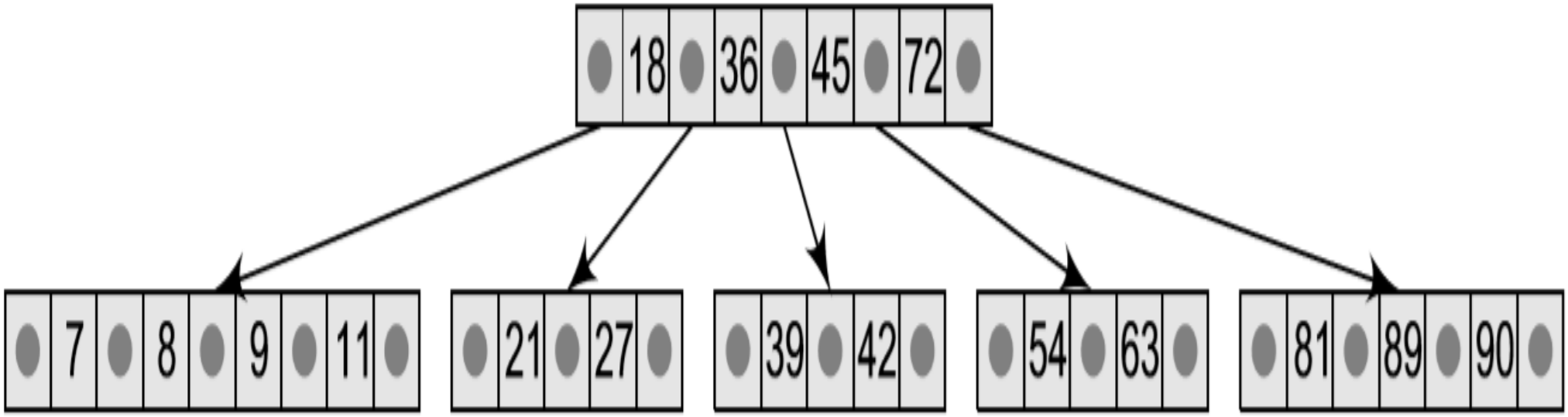
Step 1: Insert 8



Step 2: Insert 9



Step 3: Insert 39



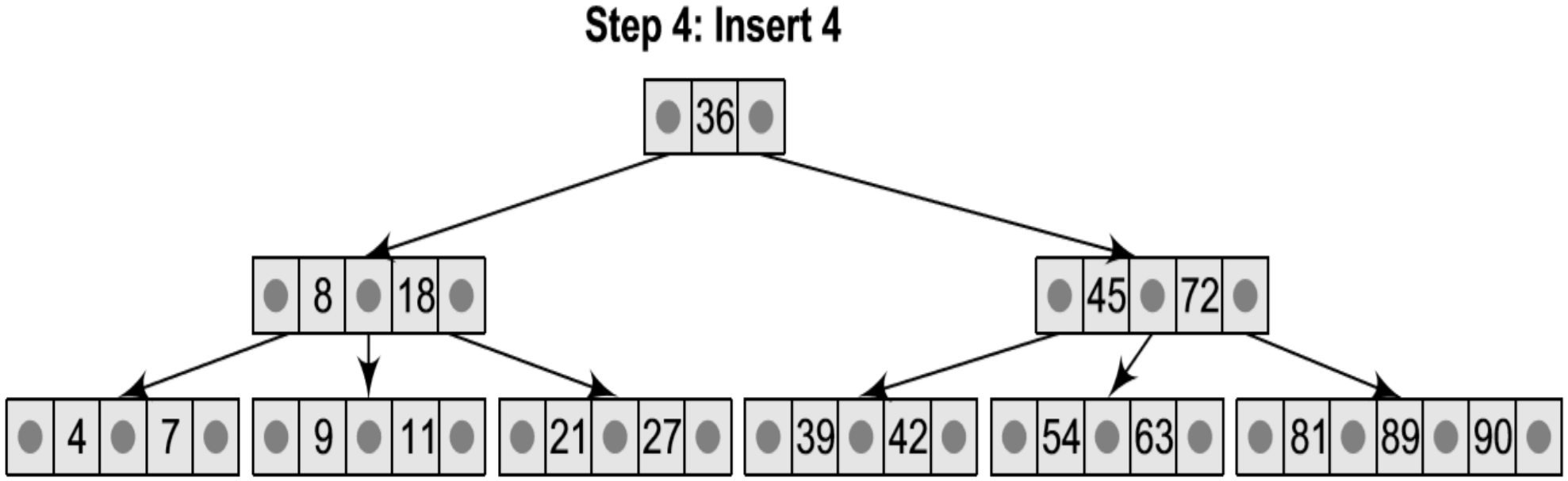


Figure 11.5(c) B tree

B TREE

Create a B tree of order 5 by inserting the following elements:

3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, and 19.

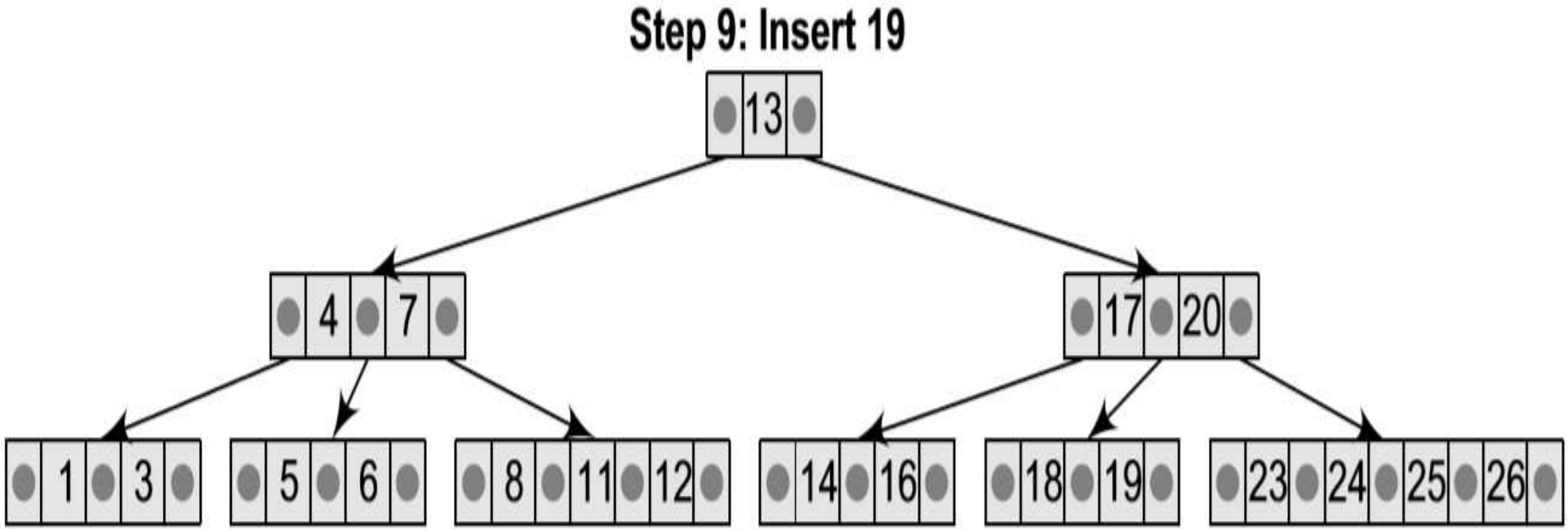


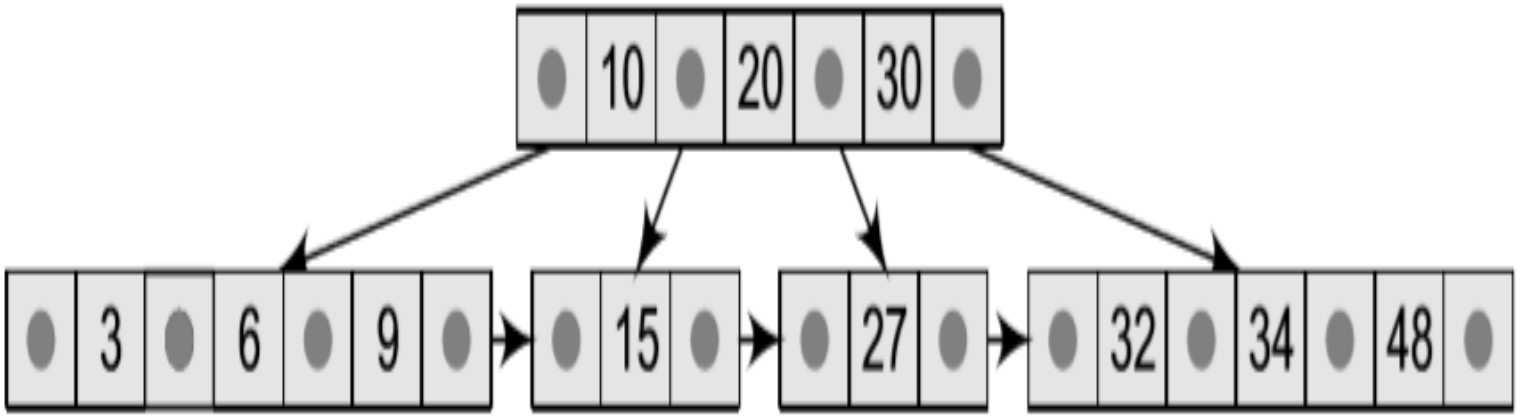
Figure 11.8 B tree

B+ TREE

- A B+ tree is a variant of a B tree which stores sorted data in a way that allows for efficient insertion, retrieval, and removal of records, each of which is identified by a key.
- While a B tree can store both keys and records in its interior nodes, a B+ tree, in contrast, stores all the records at the leaf level of the tree; only keys are stored in the interior nodes.

B+ TREE

Example 11.5 Consider the B+ tree of order 4 given and insert 33 in it.



B+ TREE

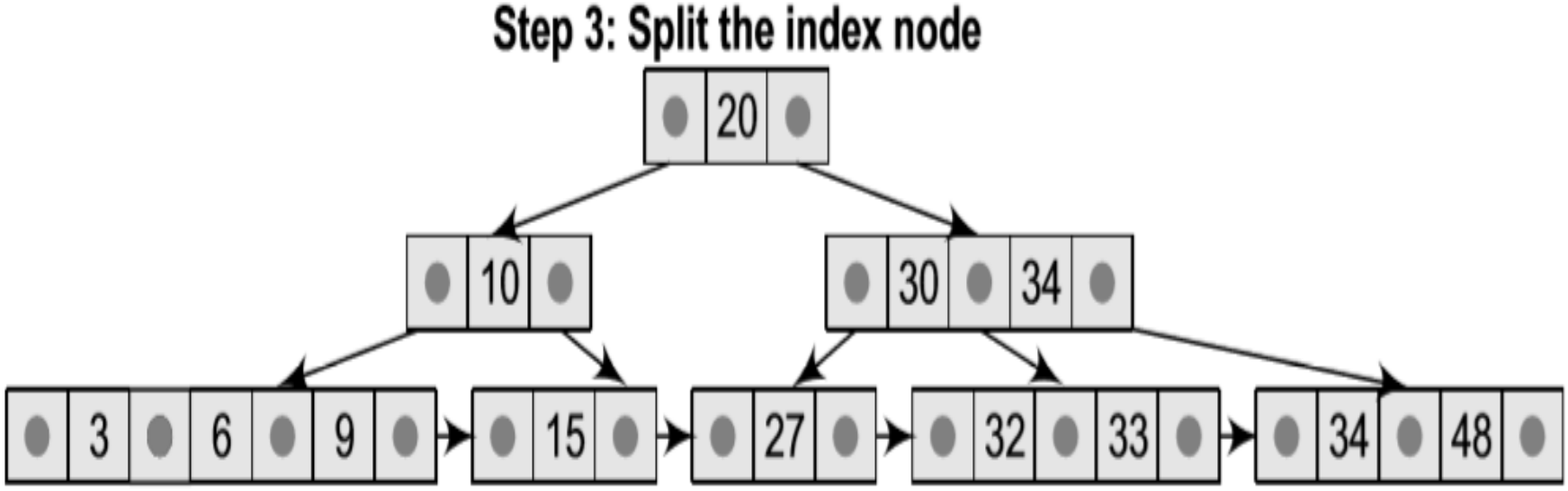
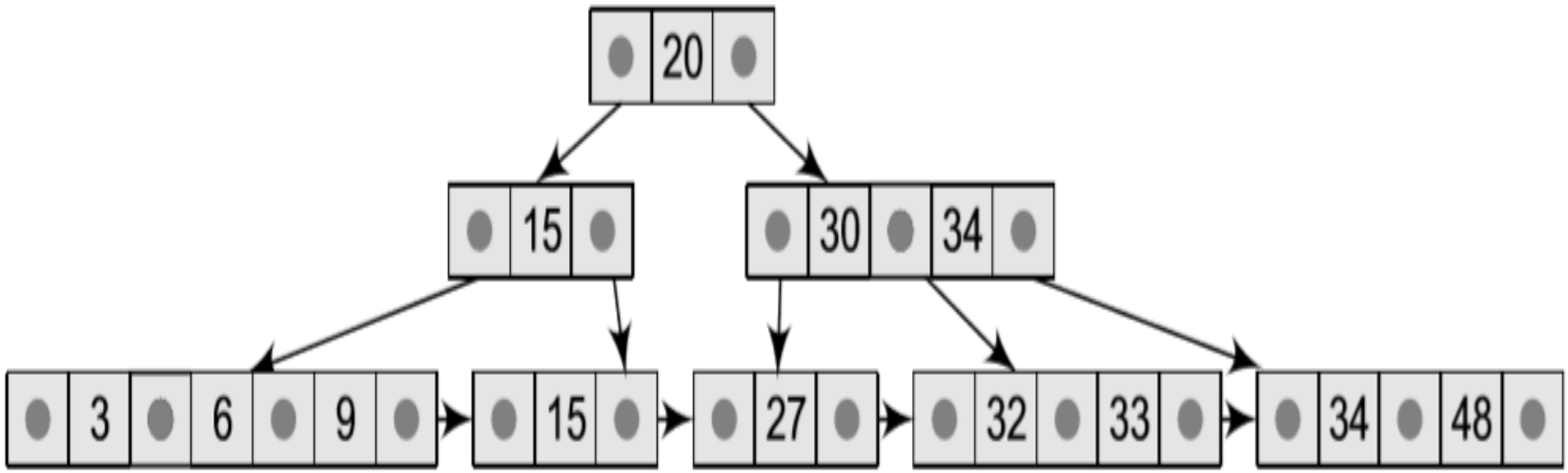


Figure 11.11 Inserting node 33 in the given B+ Tree

B+ TREE

Example 11.6 Consider the B+ tree of order 4 given below and delete node 15 from it.



B+ TREE

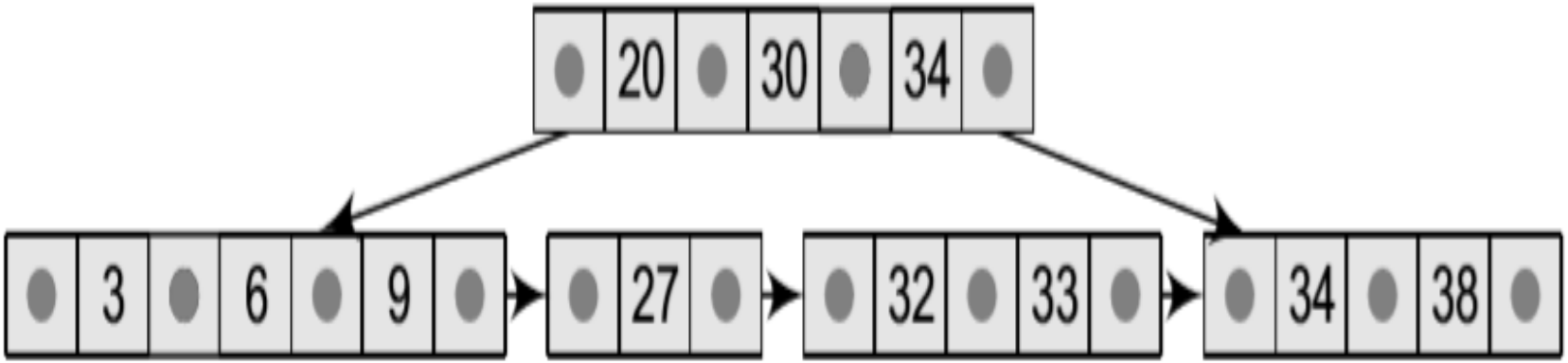


Figure 11.13 Deleting node 15 from the given B+ Tree

B+ TREE

B Tree	B+ Tree
1. Search keys are not repeated	1. Stores redundant search key
2. Data is stored in internal or leaf nodes	2. Data is stored only in leaf nodes
3. Searching takes more time as data may be found in a leaf or non-leaf node	3. Searching data is very easy as the data can be found in leaf nodes only
4. Deletion of non-leaf nodes is very complicated	4. Deletion is very simple because data will be in the leaf node
5. Leaf nodes cannot be stored using linked lists	5. Leaf node data are ordered using sequential linked lists
6. The structure and operations are complicated	6. The structure and operations are simple

Applications of TREES

- Trees are used to store simple as well as complex data. Here simple means an integer value, character value and complex data means a structure or a record.
- Trees are often used for implementing other types of data structures like hash tables, sets, and maps
- Another variation of tree, B-trees are prominently used to store tree structures on disc. They are used to index a large number of records.
- Trees are an important data structure used for compiler construction
- Trees are also used in database design.
- Trees are used in file system directories.
- Trees are also widely used for information storage and retrieval in symbol tables