B Trees & B + Trees

- A binary search tree holds only one value at the node. If there are more number of keys to be inserted into the BST, the depth of tree would be high and hence searching is time consuming.
- To overcome this problem, it is better to store more than one value in every node. This is possible using multi way search trees.
- The structure of a BST is used in an M-way search tree which has m-1 keys per node and m subtrees. In such a tree M is called the degree of the tree. In a BST, m = 2, so it has one value and 2 sub-trees. In other words, every internal node of an m-way search tree consists of pointers to M sub-trees and contains m – 1 keys, where m > 2.

Introduction

P_0	K_0	P_1	K_1	P_2	K_2	…… …	P_{n-1}	K_{m-1}	P_m

- In the structure, P_0, P_1, P_2,…, P_m are pointers to the node's sub-trees and K_0, K_1, K_2,…, K_{m-1} are the key values of the node.

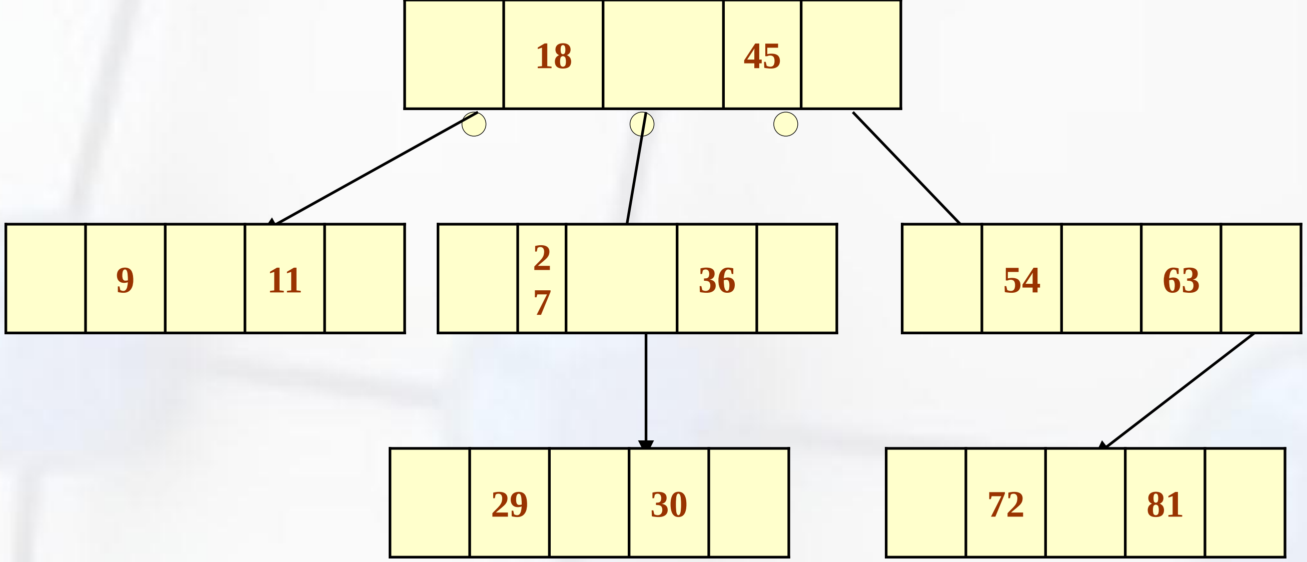
- All the key values are stored in ascending order. That is, $K_i < K_{i+1}$ for $0 \leq i \leq n-2$.

- In an M-way search tree, it is not compulsory that every node has exactly (M-1) values and have exactly M sub-trees.

- The node can have anywhere from 1 to (M-1) values, and the number of sub-trees may vary from 0 (for a leaf node) to 1 + *i*, where *i* is the number of key values in the node.

- M is thus a *fixed upper limit* that defines how much key values can be stored in the node.

M=3, Node can store 2 keys and 3 pointers



- The basic properties of an M-way search tree.
  - ➢ The key values in the sub-tree pointed by P0 are less than the key value K0. Similarly, all the key values in the sub-tree pointed by P1 are less than K1, so on and so forth. Thus, the generalized rule is that all the key values in the sub-tree pointed by Pi are less than Ki, where 0 ≤ i ≤ m–1.
  - ➢ The key values in the sub-tree pointed by P1 are greater than the key value K0. Similarly, all the key values in the sub-tree pointed by P2 are greater than K1, so on and so forth. Thus, the generalized rule is that all the key values in the sub-tree pointed by Pi are greater than Ki–1, where 1 ≤ i ≤ n.

In an M-way search tree, every sub-tree is also an M-way search tree and follows the same rules.

Searching an M-way Search Tree

- Searching an M-way search tree terminates at the leaf node. To search a value, you may need to scan more than one element in the leaf and more than one key at an interior node.
- For example, to search an M-way search tree T for key x, we start at the root node. We search the keys of the root to determine i such that $K_i \leq x \leq K_{i+1}$. If $x = K_i$, the search is complete. Else if $x \neq K_i$, then x will be in a sub-tree provided that x is in T. We then proceed to retrieve the root of the sub-tree and continue the search until we find x or determine that x is not in T.

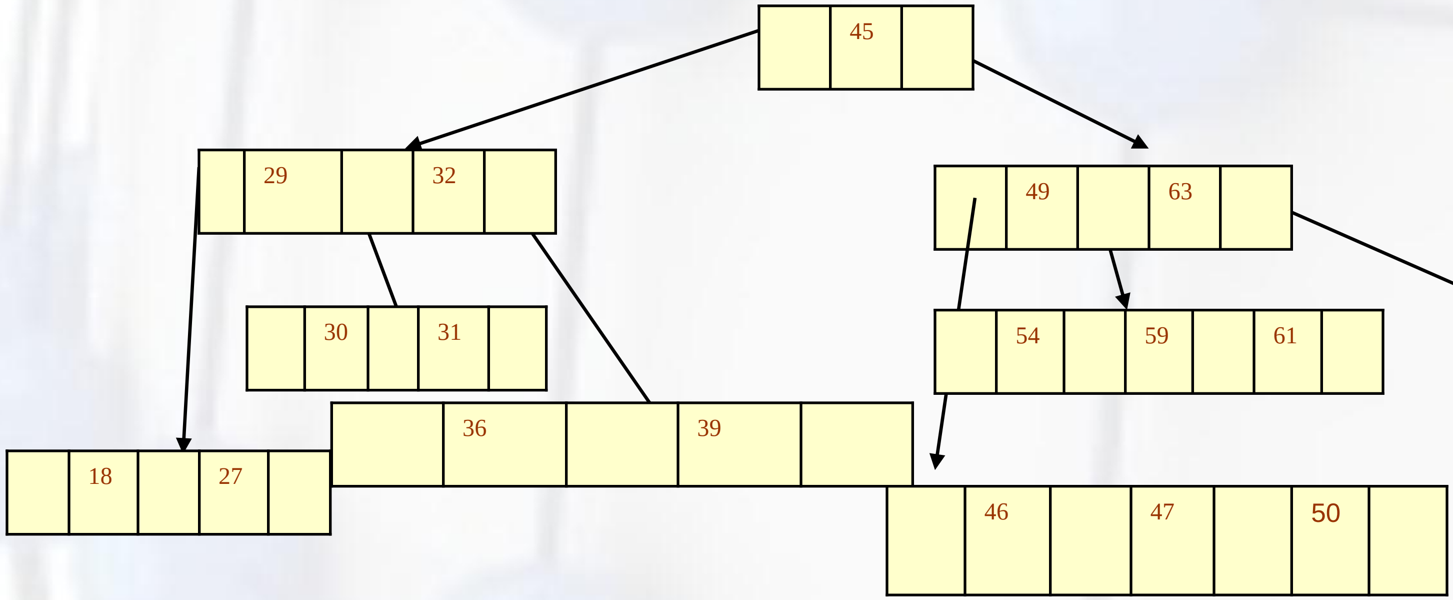
B-Trees

- A B-tree is self balancing BST and a specialized m- way tree that is widely used for disk access. It is developed by Rudolf Bayer and Ed McCreight in 1970.
- A B tree of order m can have maximum m-1 keys and m pointers to its sub-trees. A B-tree may contain a large number of key values and pointers to sub-trees. Storing a large number of keys in a single node keeps the height of the tree relatively small.
- A B-tree is designed to store sorted data and allows search, insert, and delete operations to be performed in logarithmic amortized time.
- A B-tree of order *m* (the maximum number of children that each node can have) is a tree with all the properties of an m-way search tree and in addition has the following properties:

B-Trees

- 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 have 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.
- An internal node in the B tree can have n number of children, where $0 \leq n \leq m$. It is not necessary that every node has the same number of children, but the only restriction is that the node should have at least m/2 children.

B-Trees



B-Trees

Search Operation

- Searching for an element in a B-trees is similar to that in m-way search trees.
- **Example1:** To search for 59, we begin at the root node. The root node has a value 45 which is less than 59. So, we traverse in the right sub-tree. The right sub-tree of the root node has two key values, 49 and 63. Since 49 ≤ 59 ≤ 63, we traverse the right sub-tree of 49, that is, the left sub-tree of 63. This sub-tree has three values, 54, 59, and 61. On finding the value 59, the search is successful.
- **Example 2:** To search for 9, we traverse the left sub-tree of the root node. The left sub-tree has two key values — 29 and 32. Again, we traverse the left sub-tree of 29. We find that it has two key values, 18 and 27. There is no left sub-tree of 18, hence the value 9 is not stored in the tree.

B-Trees

Insert Operation

- In a B tree all insertions are done at the leaf node level. A new value is inserted in the B tree using the algorithm given below.

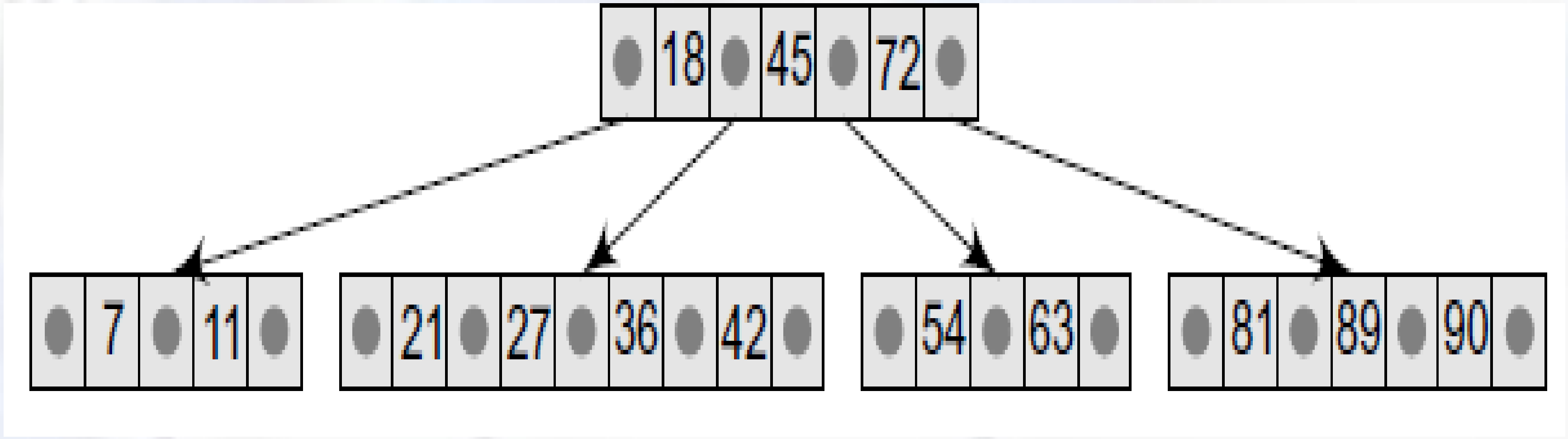
- Search the B tree to find the leaf node where the new key value should be inserted.

- If the leaf node is not full, that is it contains less than m-1 key values then insert the new element in the node, keeping the node's elements ordered.

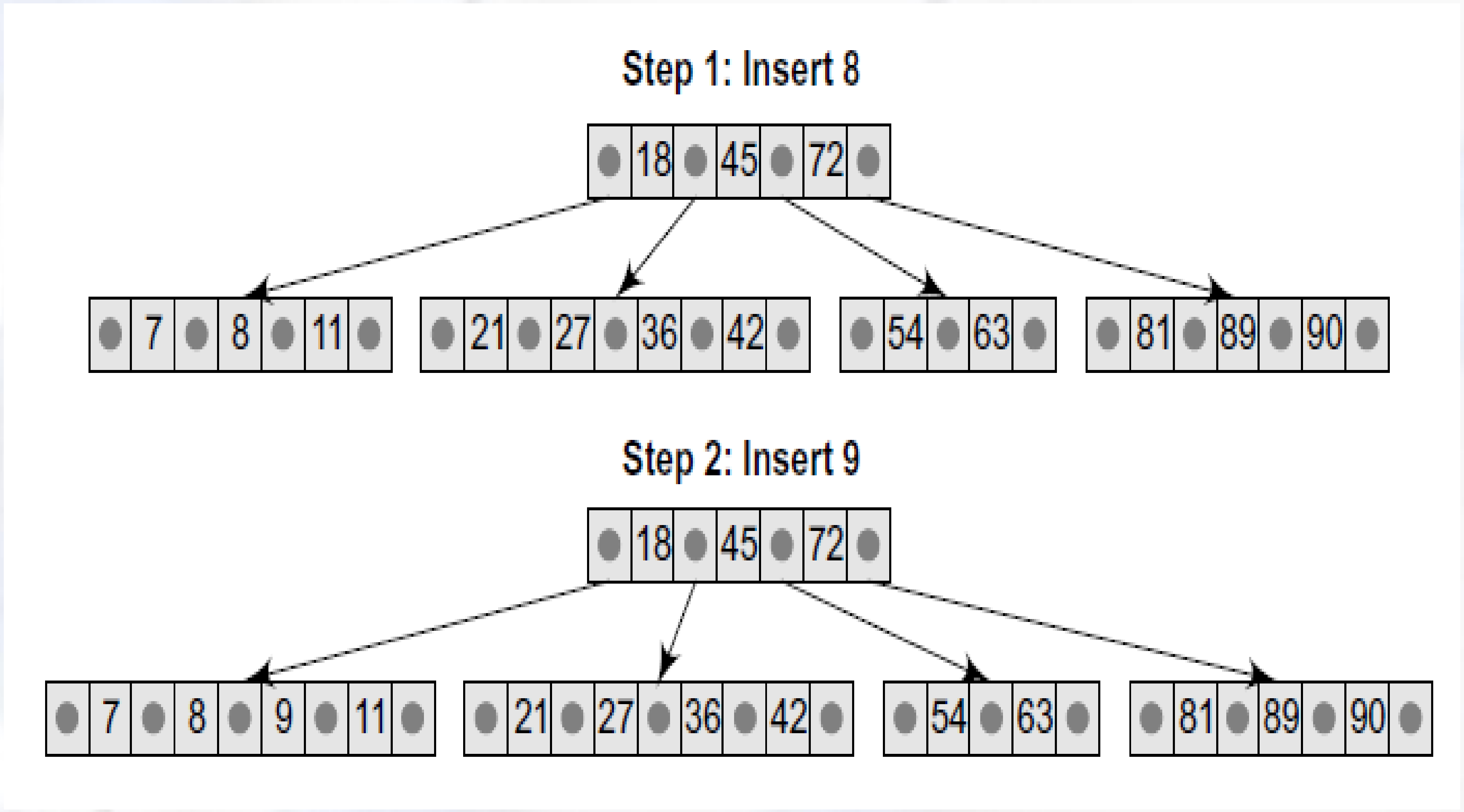
- If the leaf node is full, i.e the leaf node already contains m-1 key values then
  1. insert the new value in order into the existing set of keys
  2. split the node at its median into two nodes. note that the split nodes are half full.
  3. Push the median element up to its parent’s node. If the parent’s node is already full, then split the parent node by following the same steps.

B-Trees

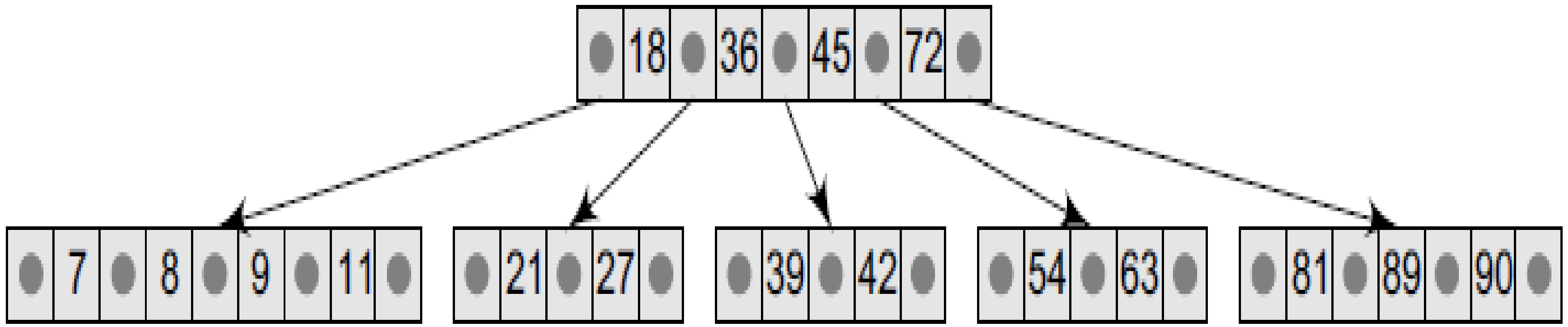
Example: Look at the B tree of order 5 given below and insert 8, 9, 39, 4 into it.



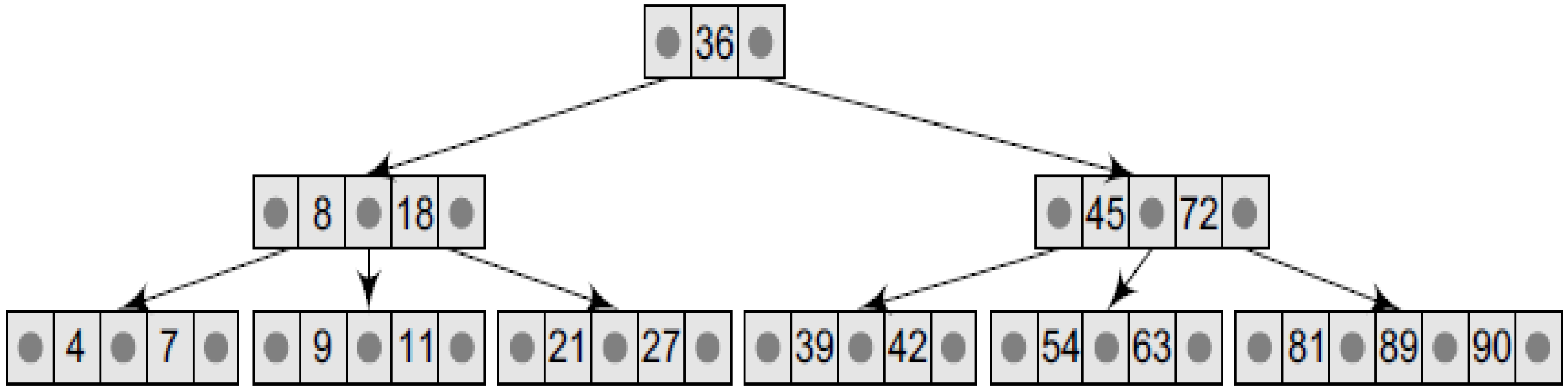
Order m = 5 indicates
Max. no. of elements in node = 4
Min. no. of elements in node = 2
(min m/2 children i.e 3 children,
ie 2 values in a node


Step 1: Insert 8
18 45 72
7 8 11
21 27 36 42
54 63
81 89 90
Step 2: Insert 9
18 45 72
7 8 9 11
21 27 36 42
54 63
81 89 90

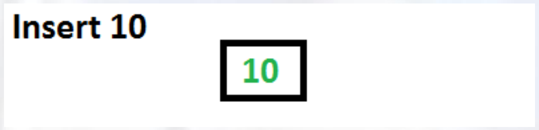
Step 3: Insert 39

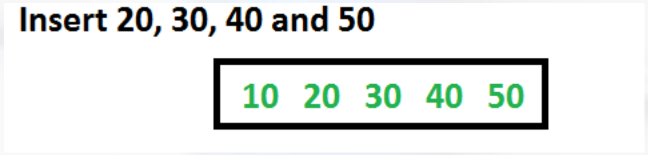


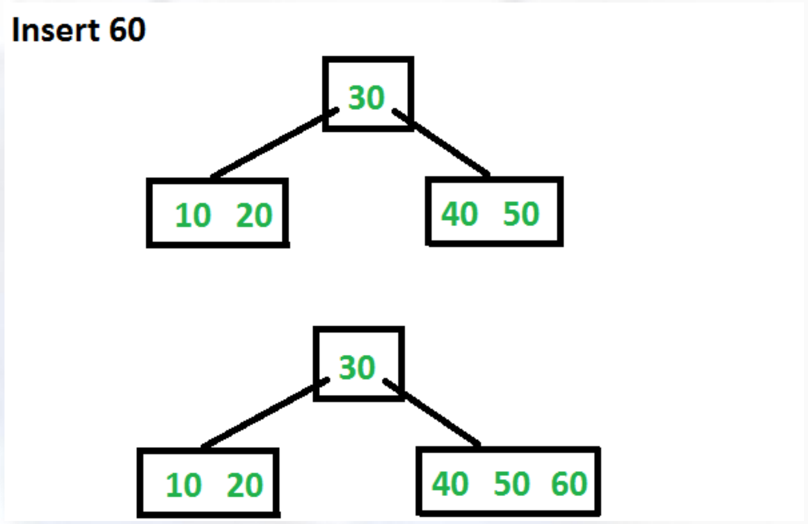
Step 4: Insert 4

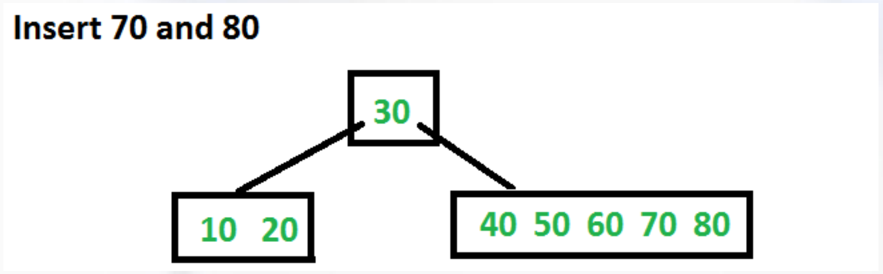


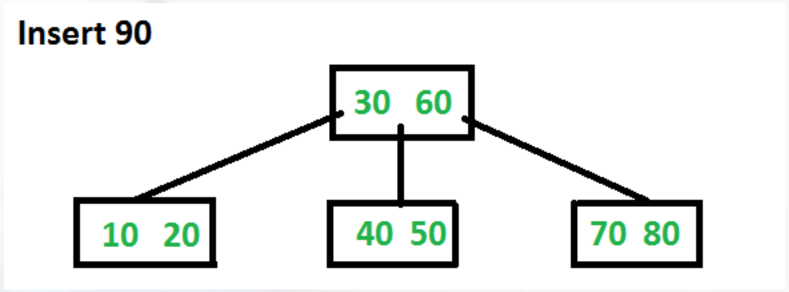
Example : Insert 10, 20, 30, 40, 50, 60, 70, 80 and 90 in an initially empty B-Tree of order 6.











Order m = 6 indicates
Max. no. of elements in node = 5
Min. no. of elements in node = 2
(min m/2 children i.e 3 children,
ie 2 values in a node

Deletion

- Like insertion, deletion is also done from the leaf nodes. There are two cases of deletion. First, a leaf node has to be deleted. Second, an internal node has to be deleted. Let us first see the steps involved in deleting a leaf node.

1. Locate the leaf node which has to be deleted
2. If the leaf node contains more than minimum number of key values (more than m/2 elements), then delete the value.
3. Else, if the leaf node does not contain even m/2 elements then, fill the node by taking an element either from the left or from the right sibling.

   (a) If the left sibling has more than the minimum number of key values, push its largest key into its parent's node and pull down the intervening element from the parent node to the leaf node where the key is deleted.

   (b) Else, if the right sibling has more than the minimum number of key values, push its smallest key into its parent node and pull down the intervening element from the parent node to the leaf node where the key is deleted.

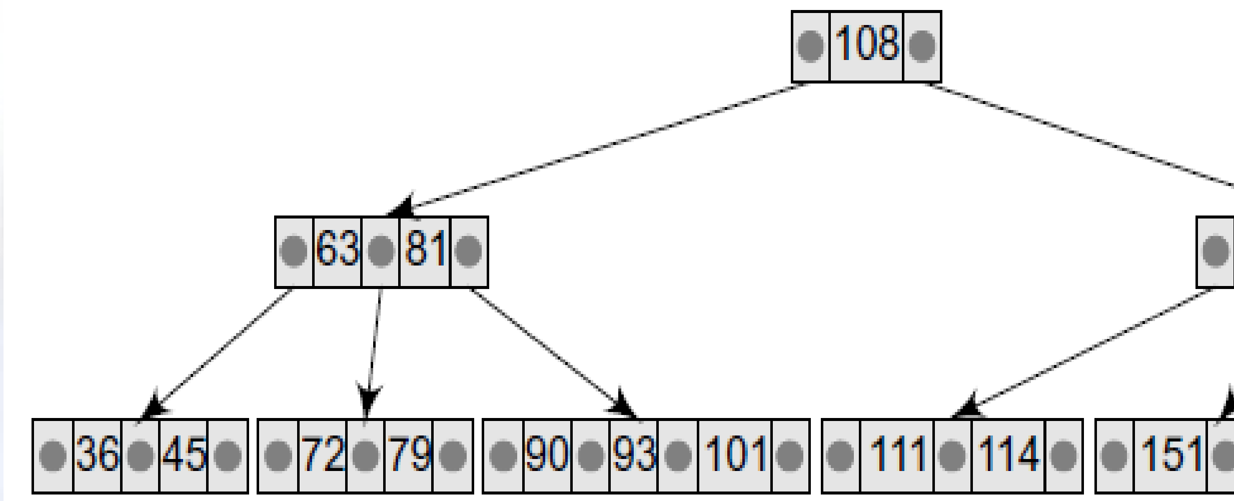
Deletion

4. Else, if both left and right siblings contain only minimum number of elements, then create a new leaf node by combining the two leaf nodes and the intervening element of the parent node (ensuring that the number of elements do not exceed the maximum number of elements a node can have, that is, m). If pulling the intervening element from the parent node leaves it with less than minimum number of keys in the node, then propagate the process upwards thereby reducing the height of the B tree.

5. To delete an internal node, promote the successor or predecessor of the key to be deleted to occupy the position of the deleted key. This predecessor or successor will always be in the leaf node. So further the processing will be done as if a value from the leaf node has been deleted.

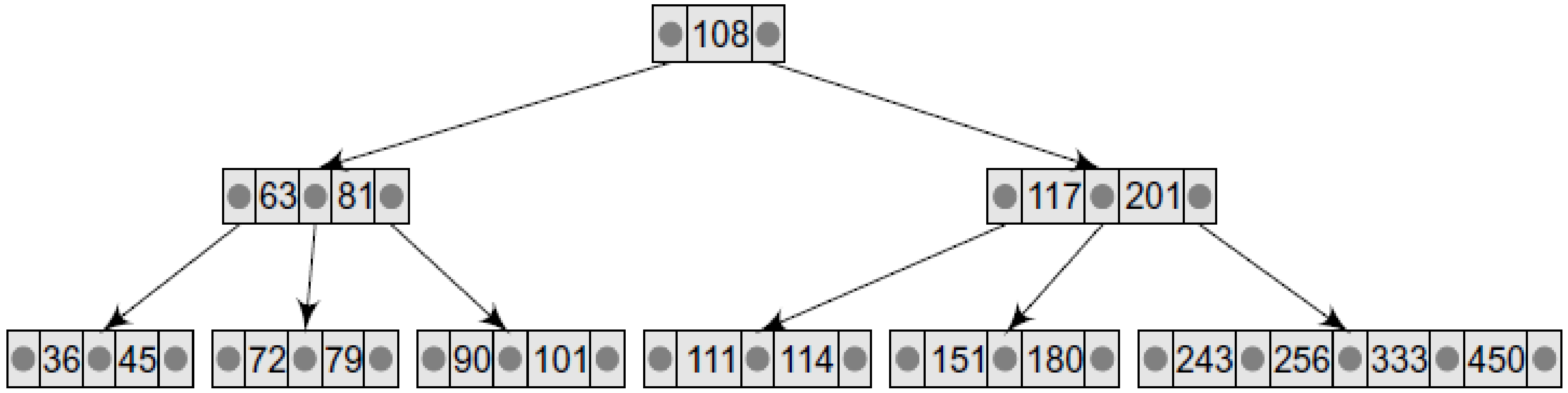
Deletion

Consider the B tree of order 5 and delete the values - 93, 201, 180, 72 from it.

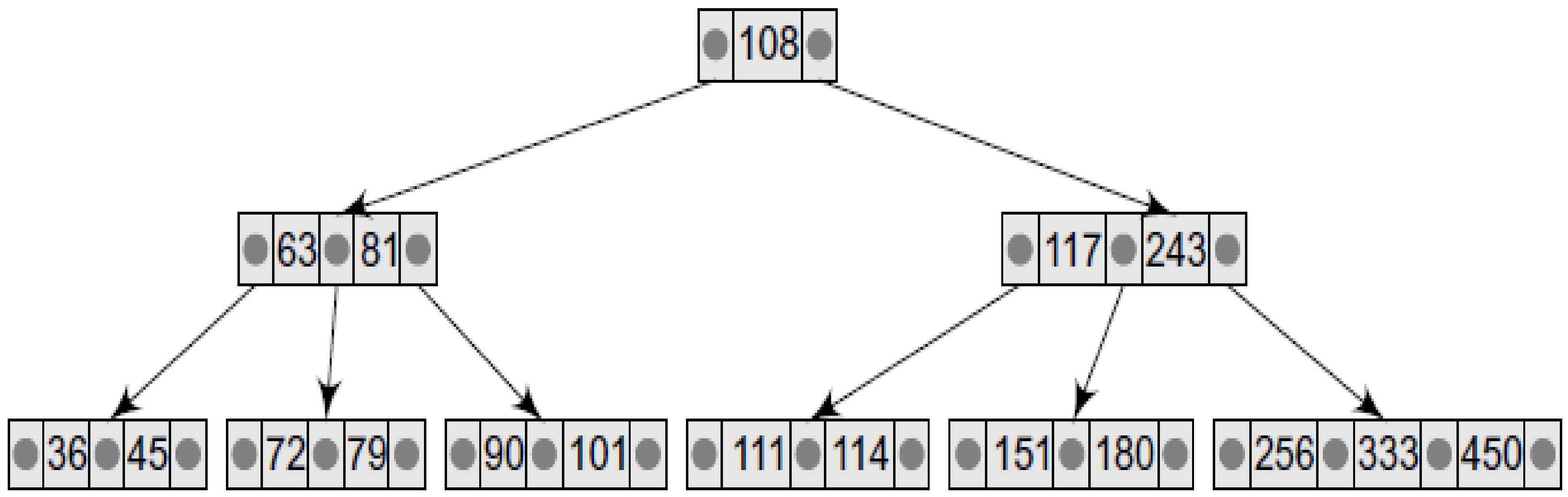


Order m = 5 indicates
Max. elements in node = 4
Min. elements in node = 2

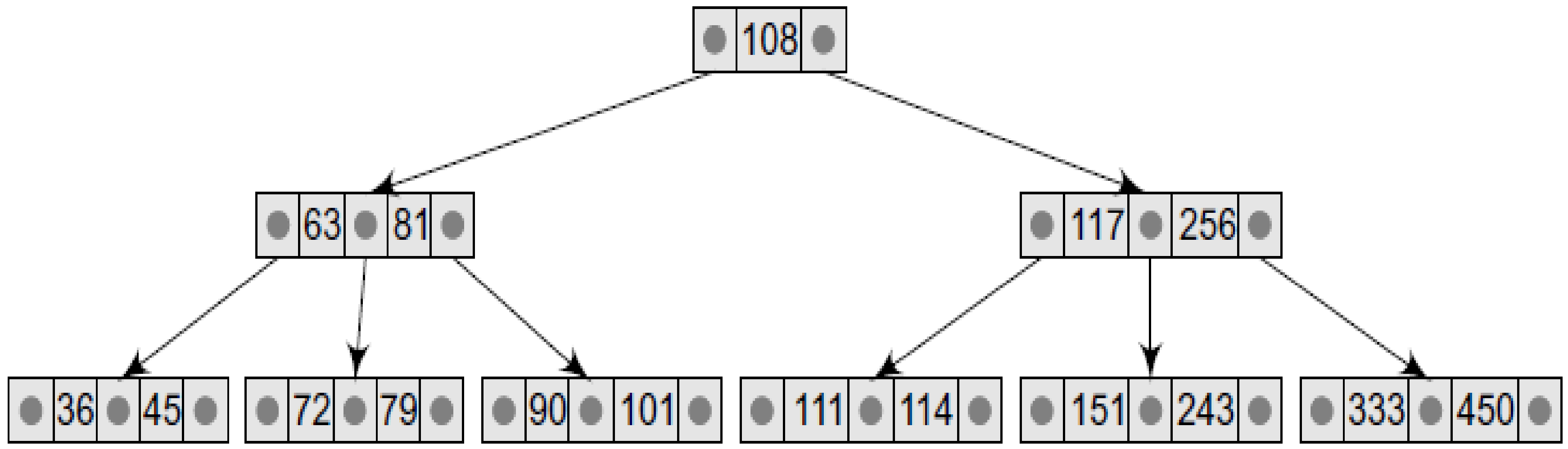
Step 1: Delete 93

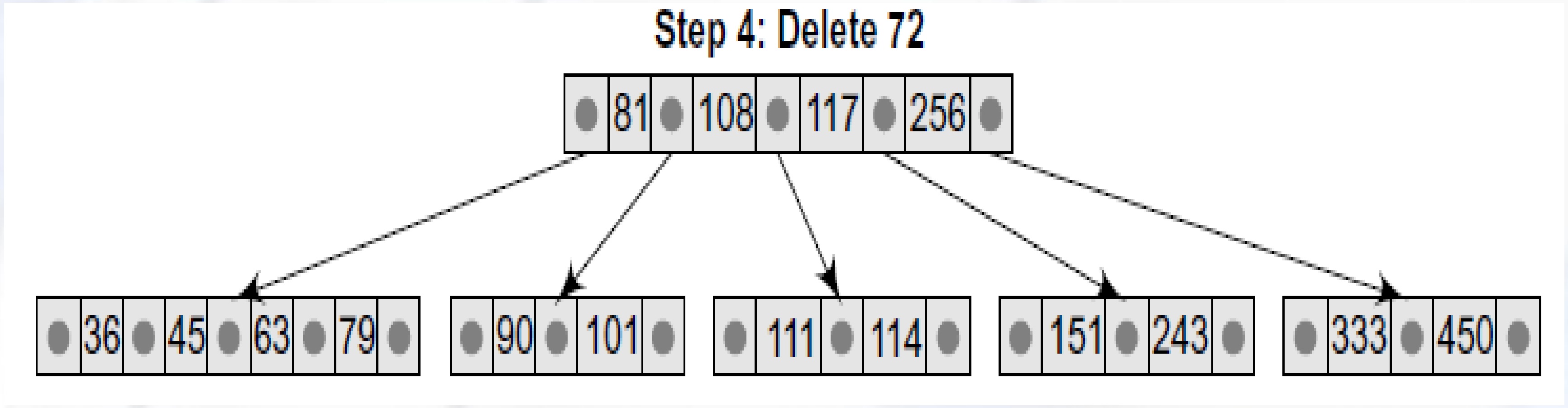


Step 2: Delete 201

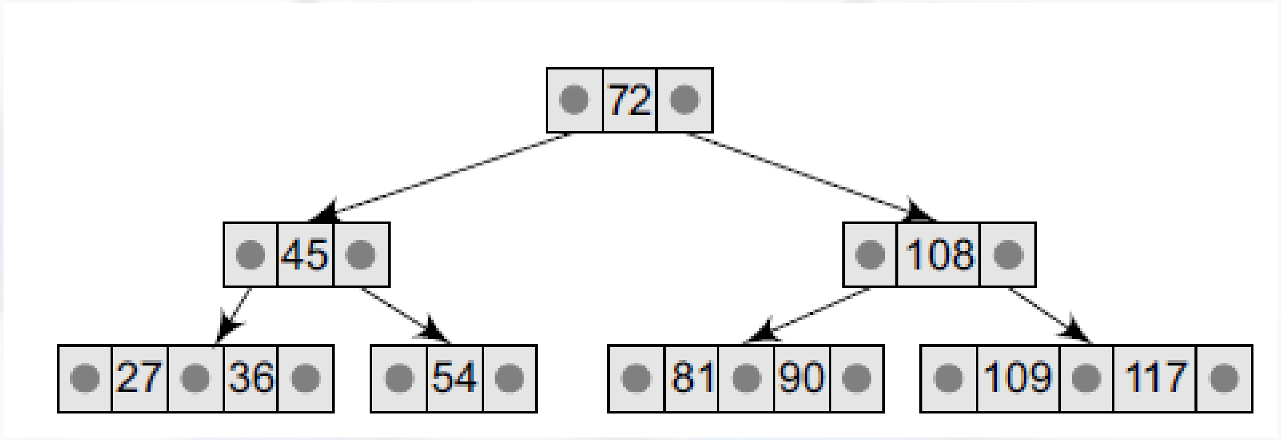


Step 3: Delete 180

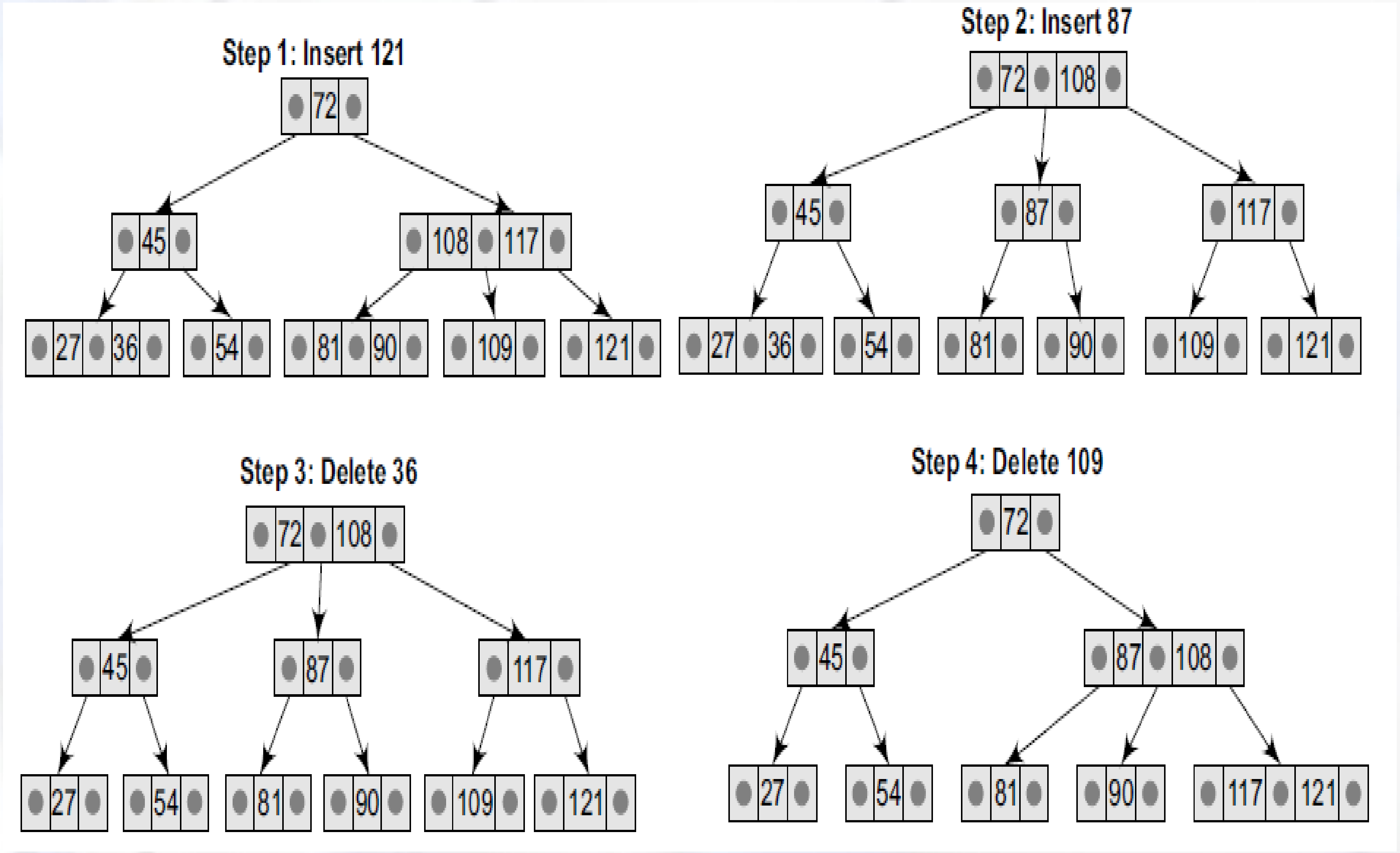




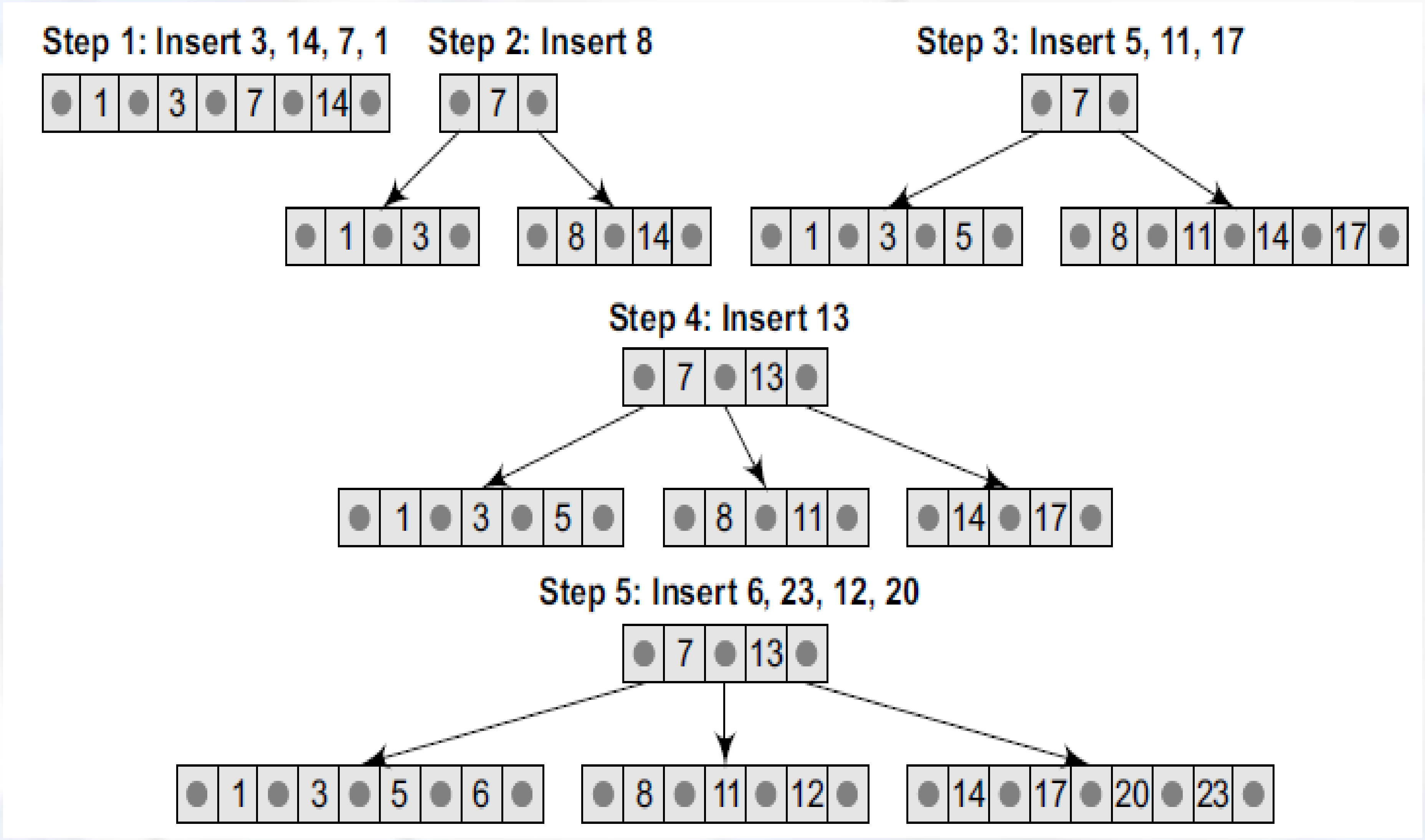
Example: Consider the B-tree of order 3 given below and perform the following operations: (a) insert 121, 87 and then (b) delete 36, 109.



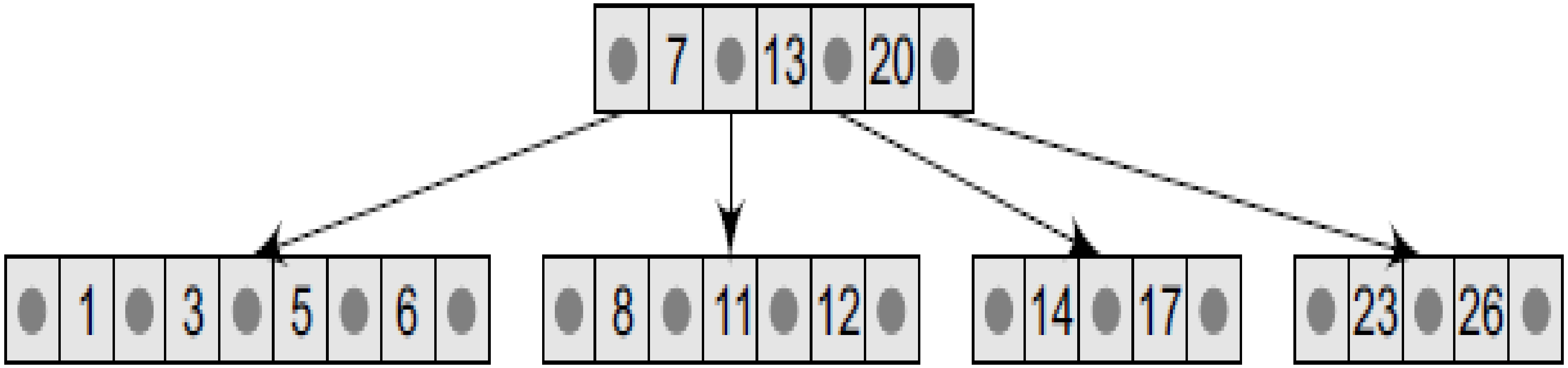
Order m = 3 indicates
Max. no. of elements in node = 2
Min. no. of elements in node = 1


Step 1: Insert 121
72
45
108 117
27 36
54
81 90
109
121
Step 2: Insert 87
72 108
45
87
117
27 36
54
81
90
109
121
Step 3: Delete 36
72 108
45
87
117
27
54
81
90
109
121
Step 4: Delete 109
72
45
87 108
27
54
81
90
117 121

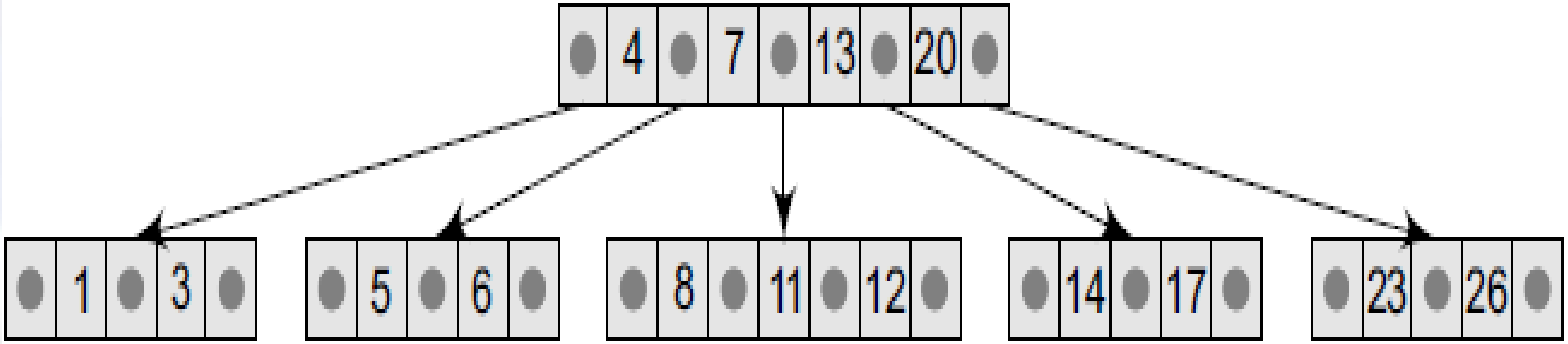
Example: Create a B-tree of order 5 by inserting the following elements: 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, and 19.



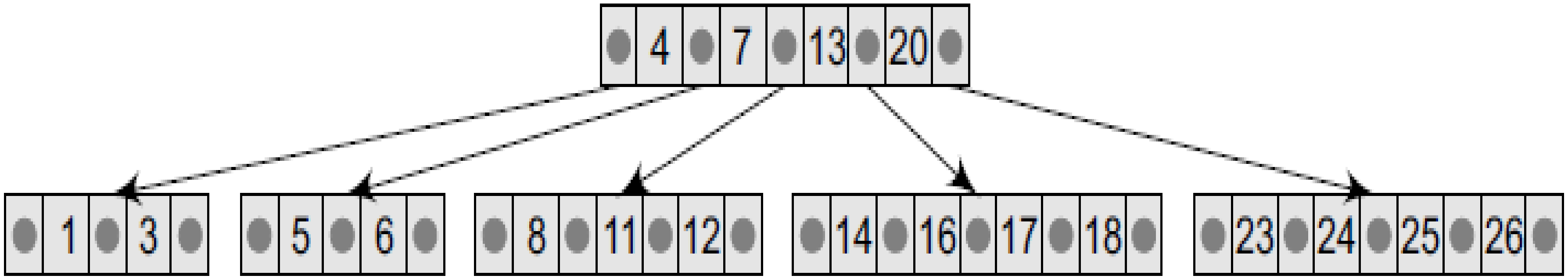
Step 6: Insert 26



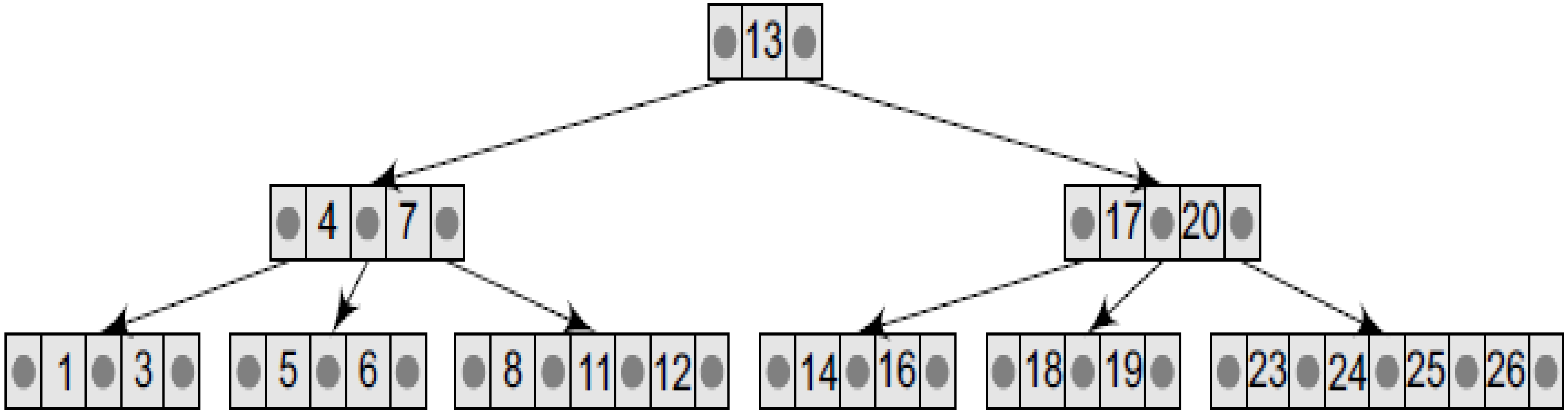
Step 7: Insert 4



Step 8: Insert 16, 18, 24, 25



Step 9: Insert 19

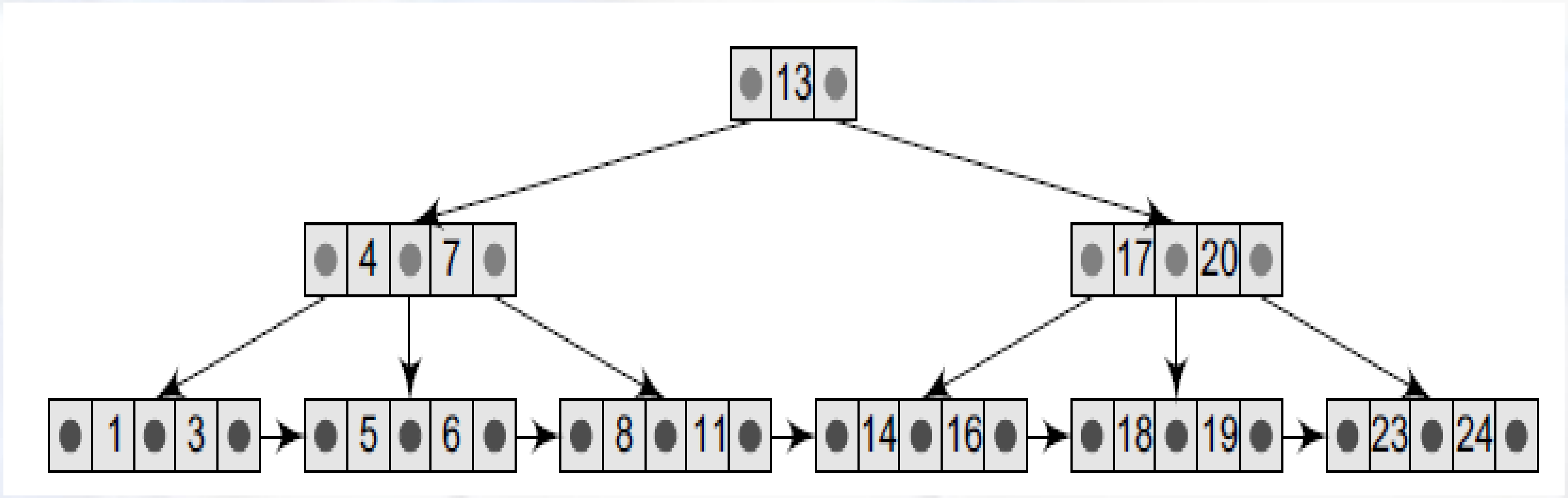


- When m=3, all internal nodes of a B- tree have a degree that is either 2 or 3 (children). For this reason a B-tree of order 3 is called a 2-3 tree.
- When m=4, all internal nodes of a B- tree have a degree that is either 2 , 3 or 4. For this reason a B-tree of order 4 is called 2-3-4 tree.
- When m=5, all internal nodes of a B- tree have a degree that is either 3 , 4 or 5. For this reason a B-tree of order 5 is called 3-4-5 tree.

B+ Trees

- 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*.
- A B tree can store both keys and records in its interior nodes, a B+ tree, in contrast, stores all records at the leaf level of the tree and only keys are stored in interior nodes.
- B+ tree stores data only in the leaf nodes. All other nodes (internal nodes) are called index nodes or i-nodes and store index values which allow us to traverse the tree from the root down to the leaf node that stores the desired data item.
- The leaf nodes of the B+ tree are often linked to one another in a linked list. This has an added advantage of making the queries simpler and more efficient.
- Typically B+ trees are used to store large amounts of data that cannot be stored in the main memory. With B+ trees, the secondary storage is used to store the leaf nodes of the tree and the internal nodes of the tree are stored in the main memory.

B+ Trees



Advantages of B+ trees:

- Because B+ trees don't have data associated with interior nodes, more keys can fit on a page of memory. Therefore, it will require fewer cache misses in order to access data that is on a leaf node.
- The leaf nodes of B+ trees are linked, so doing a full scan of all objects in a tree requires just one linear pass through all the leaf nodes. A B tree, on the other hand, would require a traversal of every level in the tree. This full-tree traversal will likely involve more cache misses than the linear traversal of B+ leaves.

Advantage of B trees:

- Because B trees contain data with each key, frequently accessed nodes can lie closer to the root, and therefore can be accessed more quickly.

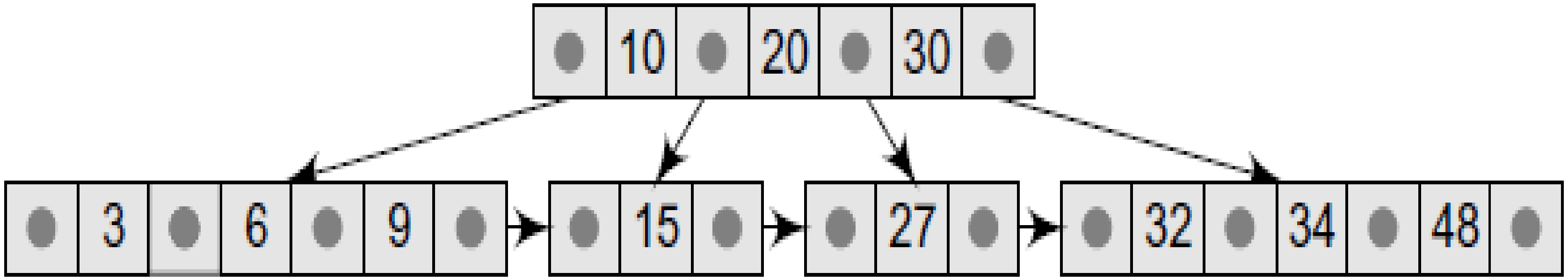
Searching a B+ Tree

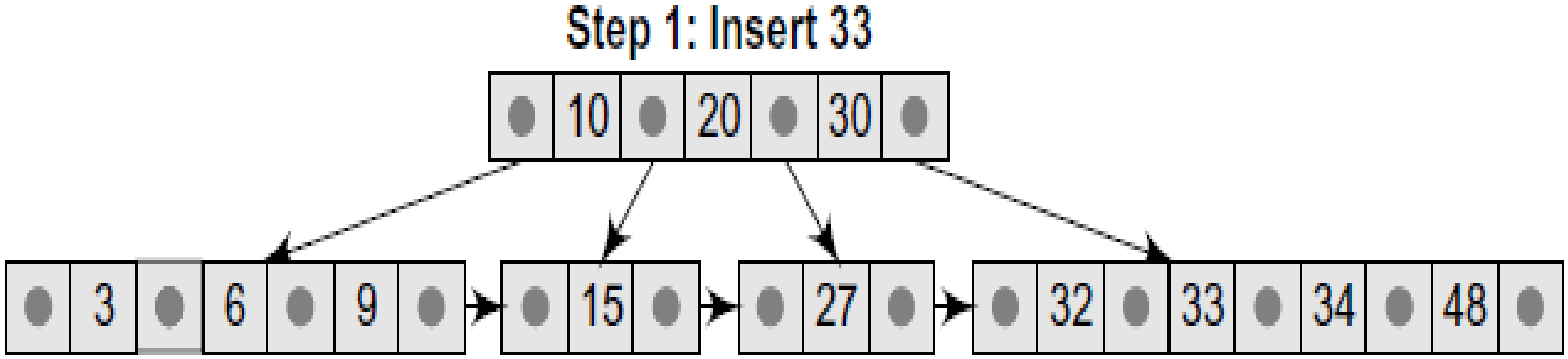
- The process of searching a B+ tree is exactly same as binary search tree or B-tree.
- Searching starts at the root node and works down to the leaf level.
- At each step, a comparison of the search value and the current key value is done.
- The flow control then moves either towards left or to the right of the tree. This process is repeated until a leaf node is reached.

Inserting a New Element

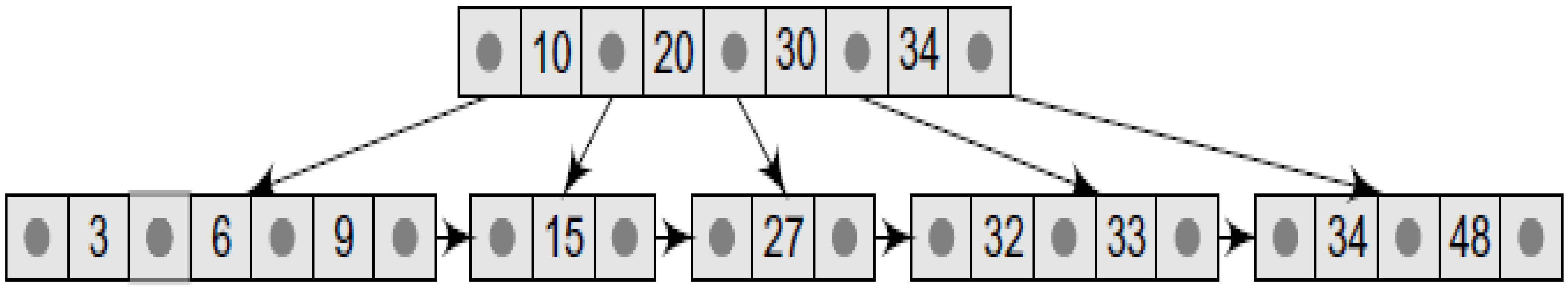
- Step 1: Insert the new node as the leaf node.
- Step 2: If the leaf node overflows, split the node and copy the middle element node(first split node max element or second split node min element) to parent index.
- Step 3: If the index node overflows, split that node and move the middle element (first split node max element or second split node min element) to parent index page.

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

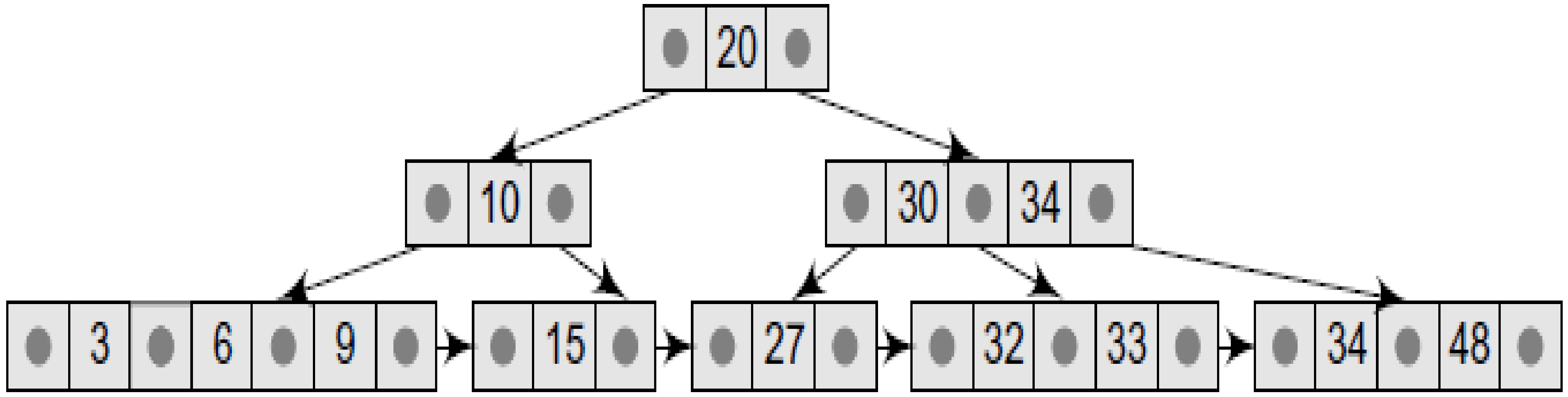




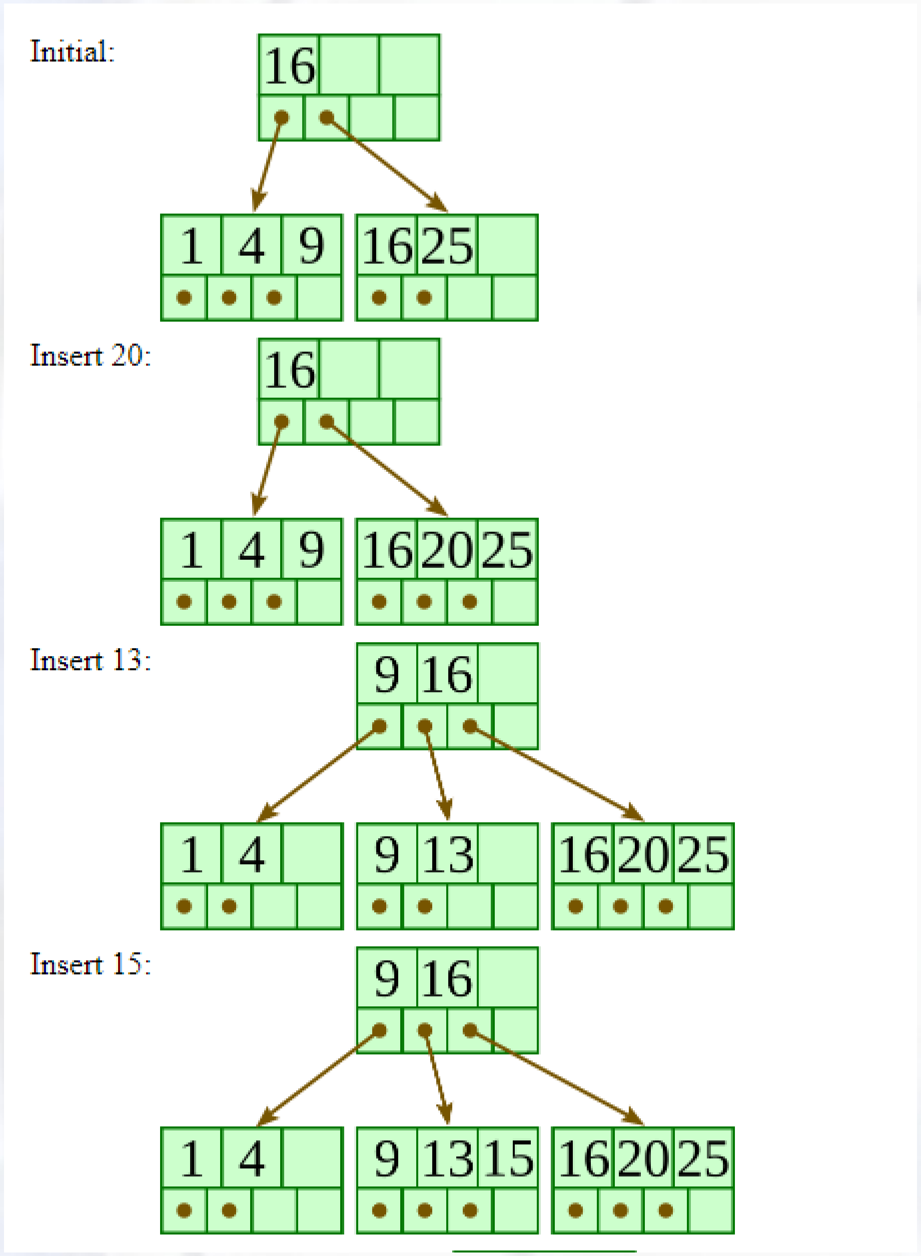
Step 2: Split the leaf node

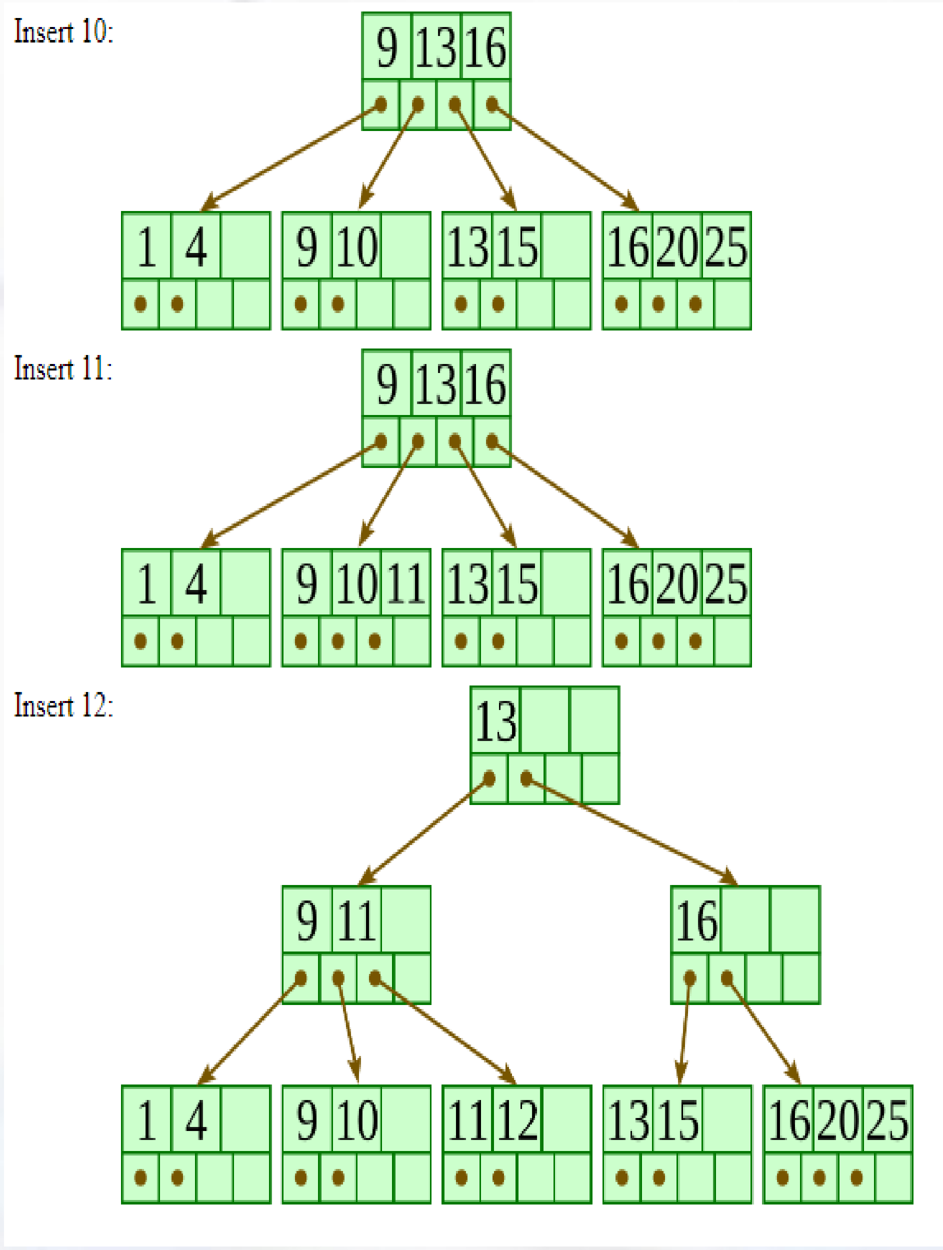


Step 3: Split the index node



Insert 20,13,15, 10, 11, 12 into a B+ tree of order 4

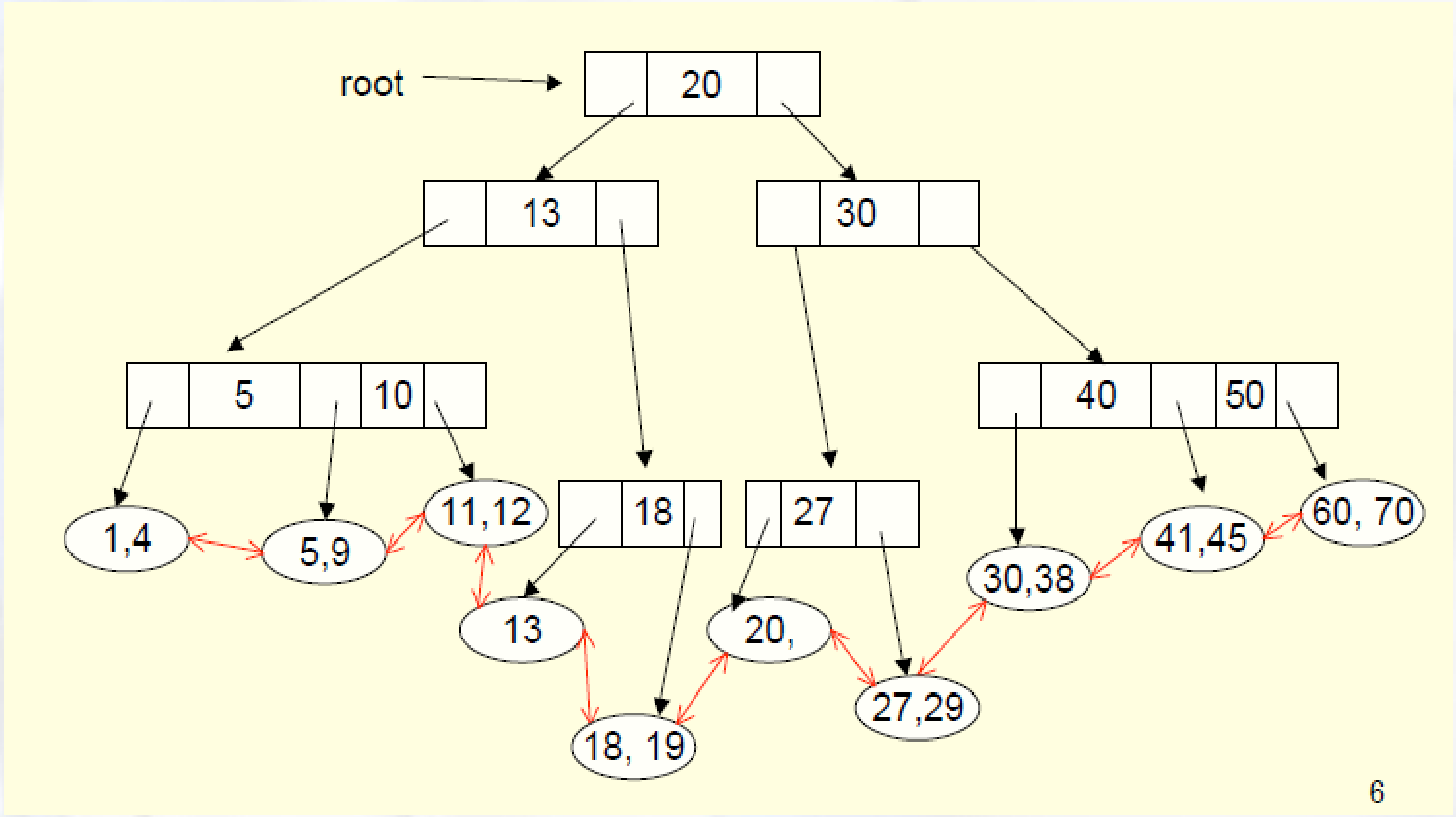




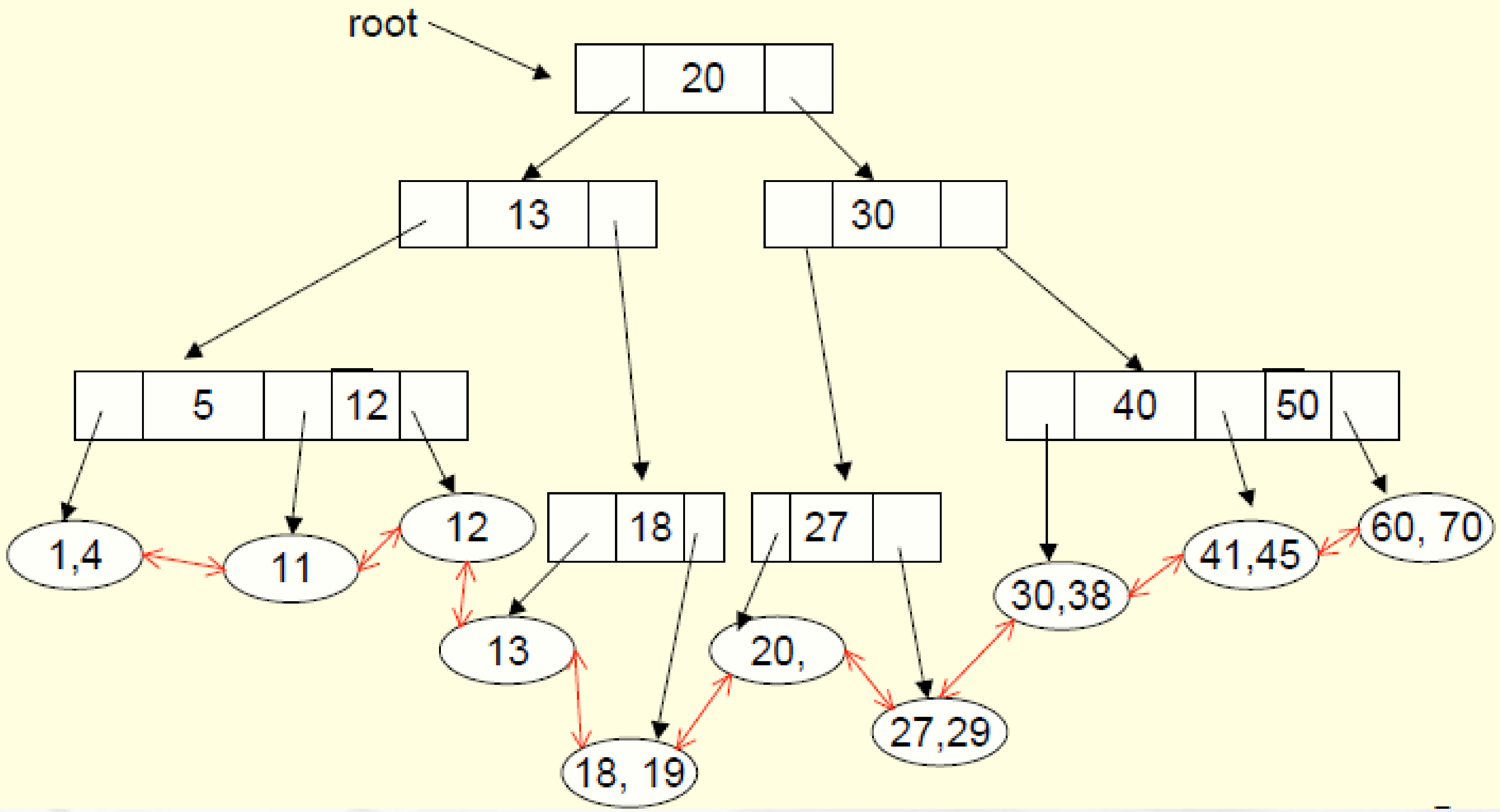
Deleting an element from a B+ Tree

- Start at root, find leaf node L where the element exists and remove it.
- If L is atleast half-full, exit
- If L underflows (has less than m/2-1 elements)
  - Try to re-distribute by borrowing from sibling (adjacent node with same parent as L). Take the max from left sibling or min value from right sibling. Update the intervening element in parent with the min element in the second node of the existing two.
  - If re-distribution fails, merge L and sibling. Delete the intervening element in parent of L. Merge could propagate to root, decreasing height.
  - If the interior(index) node underflows, merge the node with sibling and (move down the) intervening element.

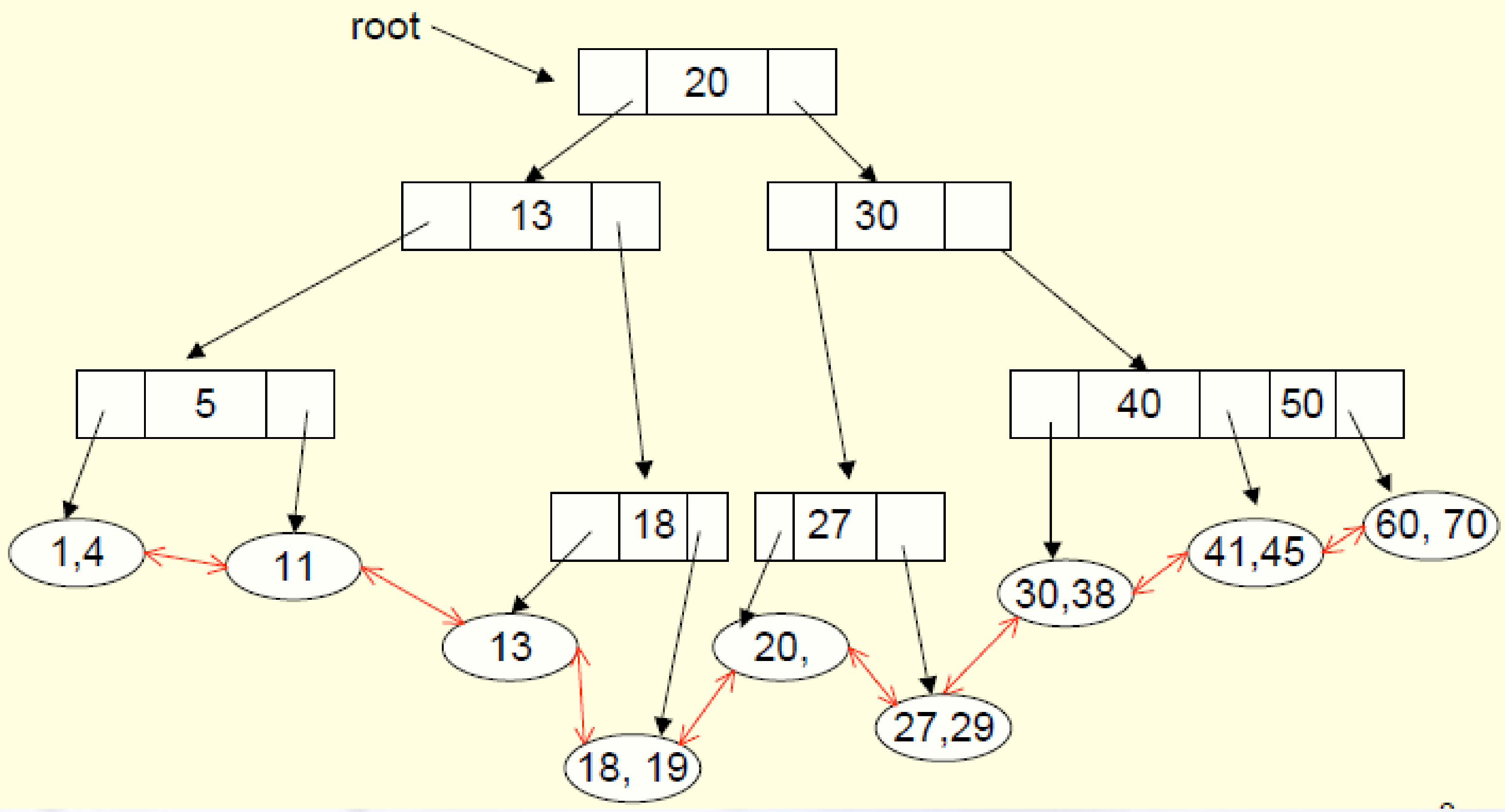
Example: Consider the B+ Tree of order 3. Delete 5, 9, 12, 4, 11



Delete 5, then 9
redistribute from right sibling



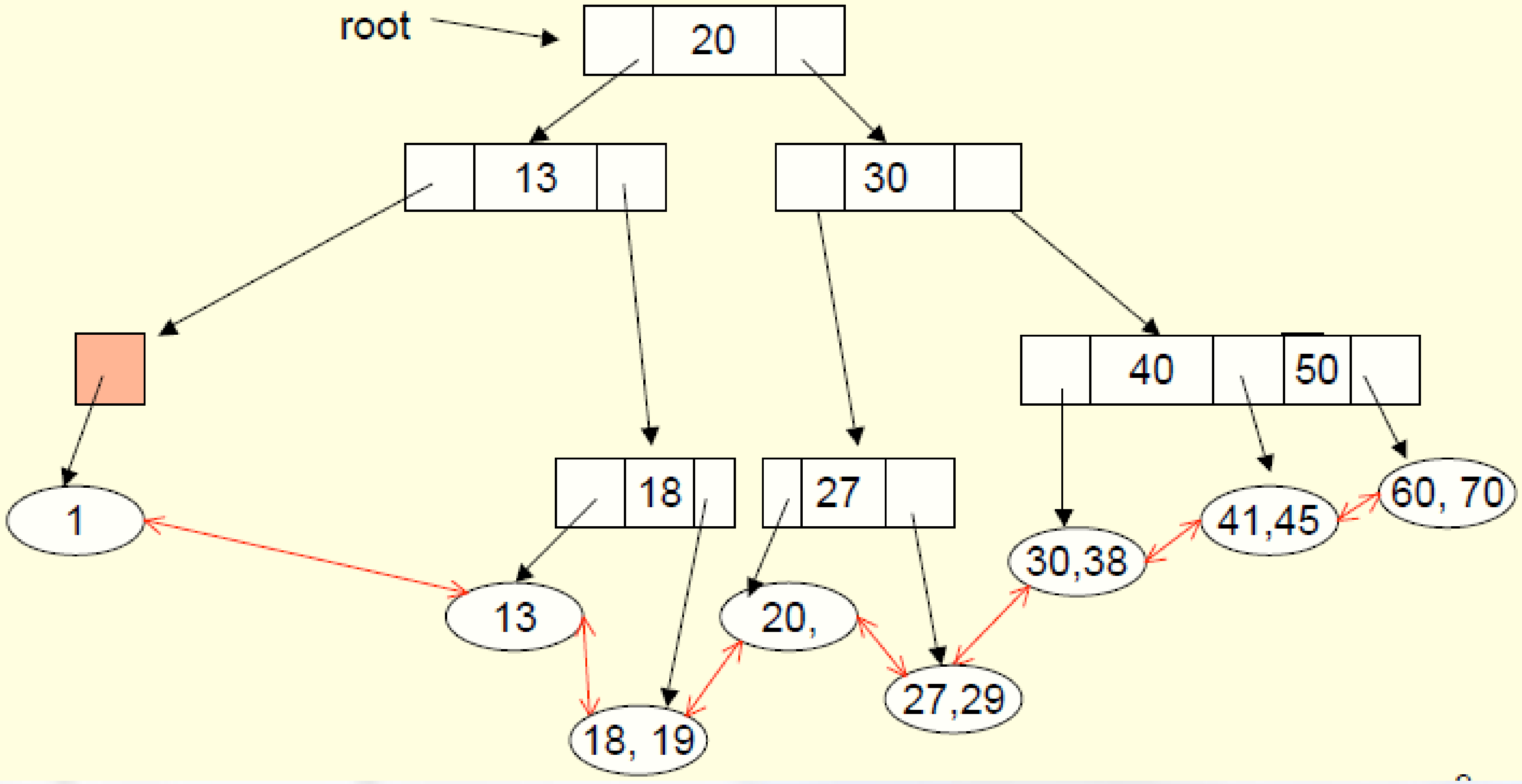
Delete 12
merge leaves, delete key from parent



Delete 4, then 11

merge leaves, delete key from parent

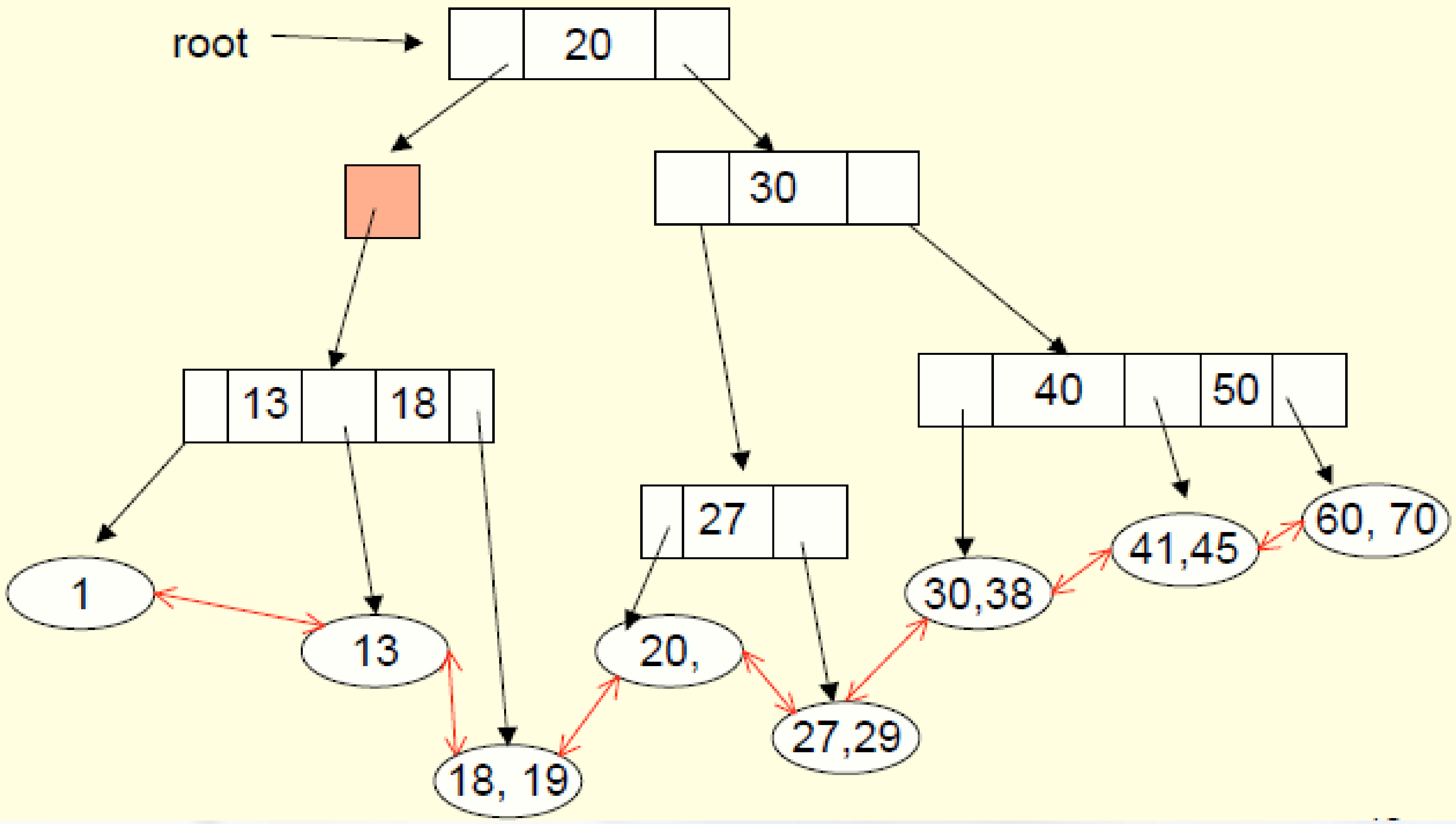
=>parent not full enough



Delete 4, then 11

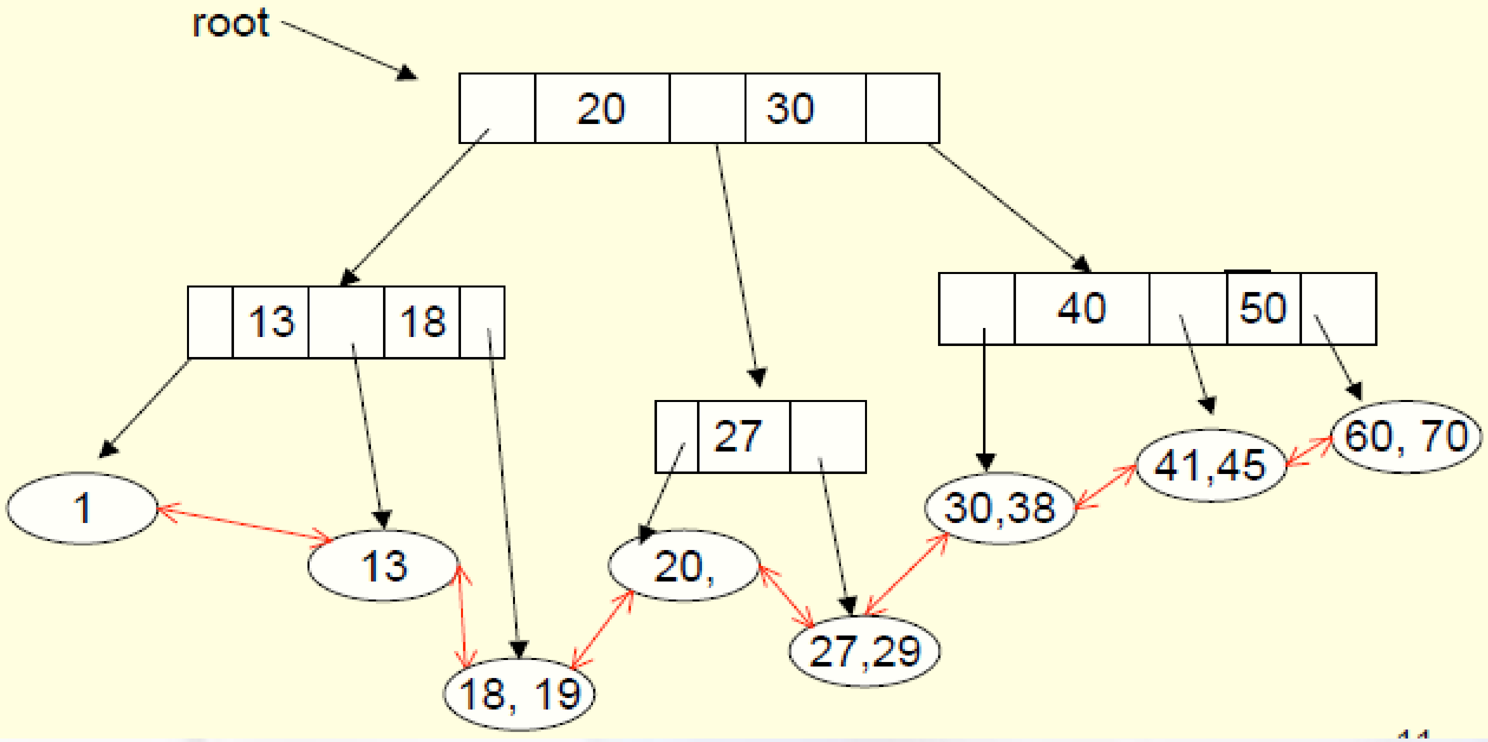
merge leaves, merge parent, bringing down key 13

=>grandparent not full enough

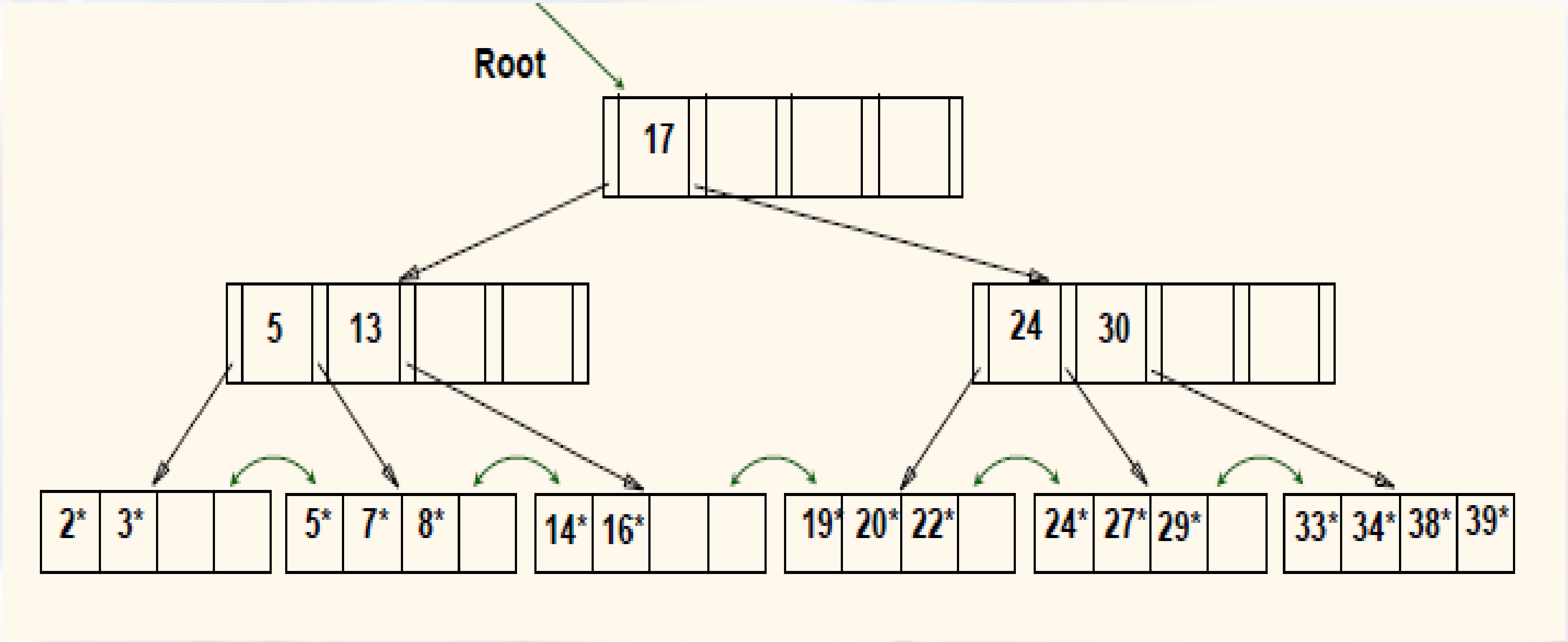


Delete 4, then 11

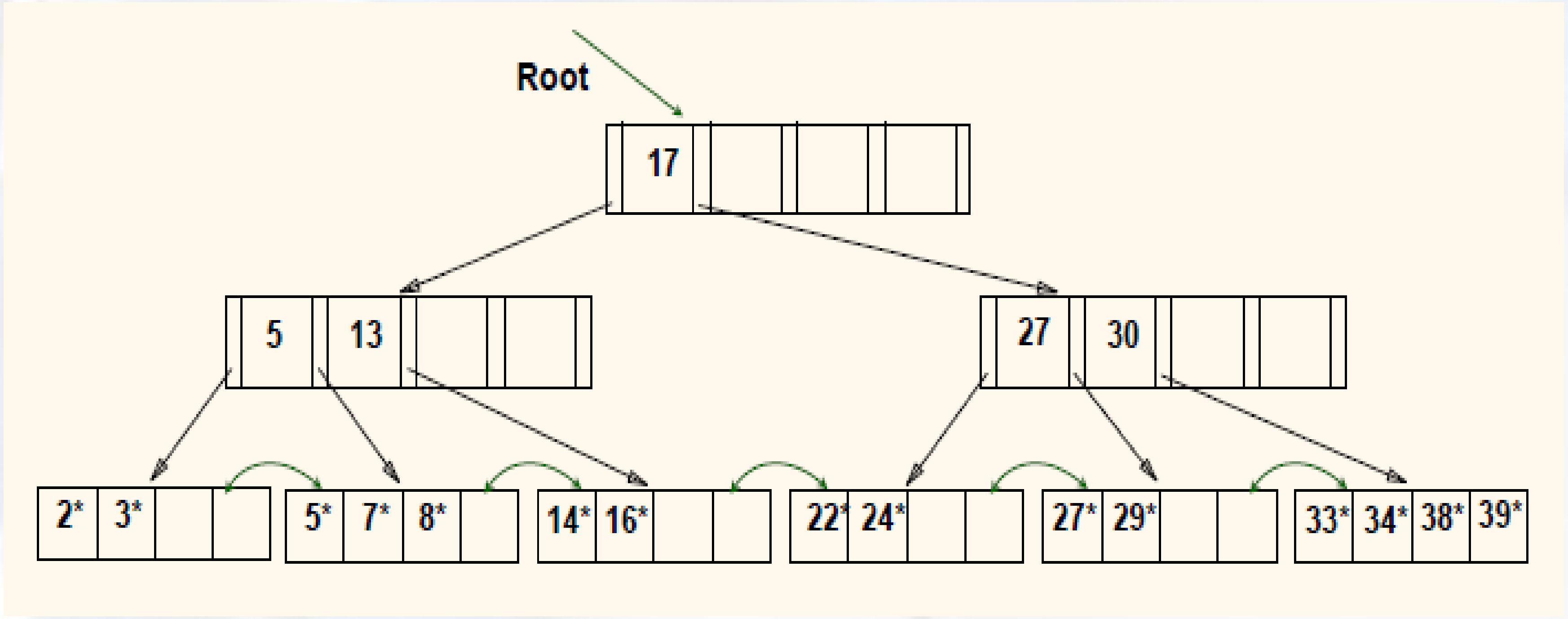
merge leaves; merge parent, bringing down key 13
merge grandparent, bring down key 20,
remove root



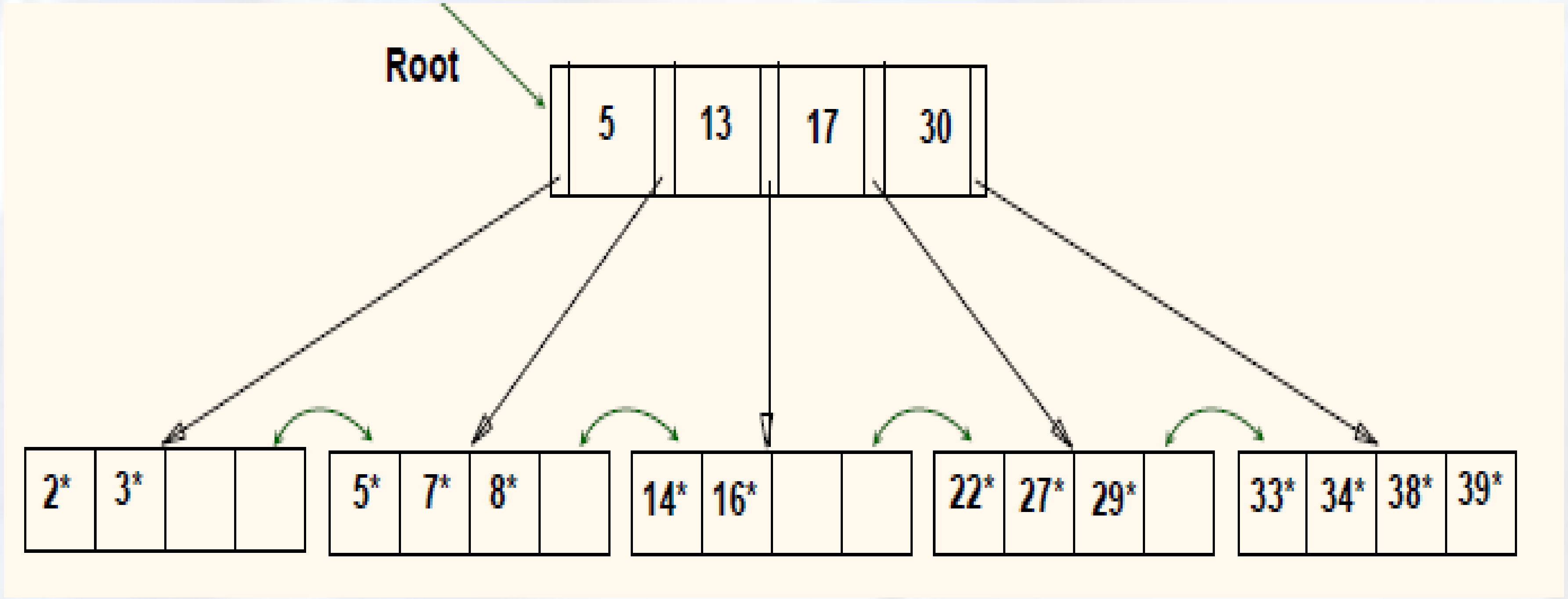
Example: Consider the B+ Tree of order 5. Delete 19, 20, 24



After deleting 19, 20



After deleting 24


Root
5
13
17
30
2*
3*
5*
7*
8*
14*
16*
22*
27*
29*
33*
34*
38*
39*