Chap.2

B-Tree and B+ Tree

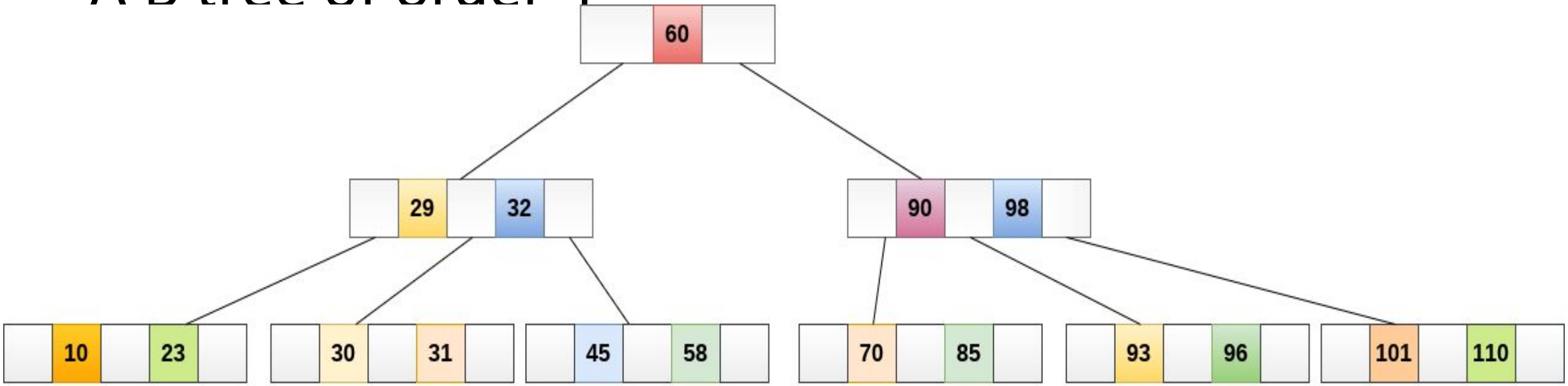
Introduction

- B Tree is a specialized m-way tree that can be widely used for disk access.
- A B-Tree of order m can have at most m-1 keys and m children.
- One of the main reason of using B tree is its capability to store large number of keys in a single node and large key values by keeping the height of the tree relatively small.

B Tree

- A B tree of order m contains all the properties of an M way tree. In addition, it contains the following properties.
  - Every node in a B-Tree contains at most m children.
  - Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.
  - The root nodes must have at least 2 nodes.
  - All leaf nodes must be at the same level.
- It is not necessary that, all the nodes contain the same number of children but, each node must have m/2 number of nodes.

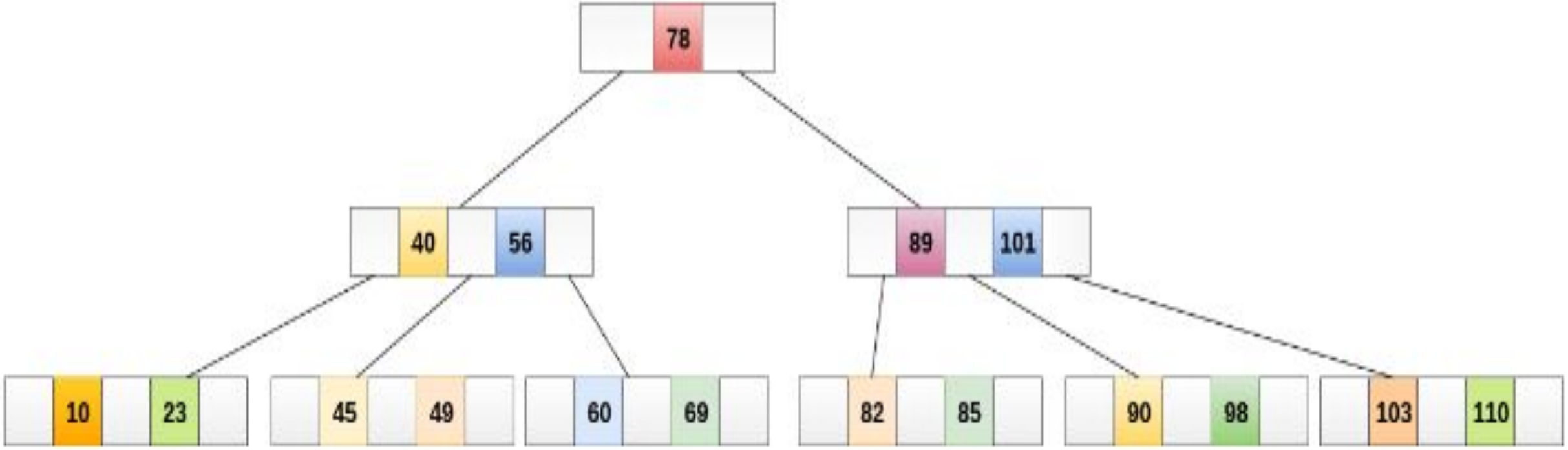
A B tree of order 4



- While performing some operations on B Tree, any property of B Tree may violate such as number of minimum children a node can have. To maintain the properties of B Tree, the tree may split or join.

Operations : Searching :

- Searching in B Trees is similar to that in Binary search tree. For example, if we search for an item 49 in the following B Tree. The process will something like following :
  1. Compare item 49 with root node 78. since 49 < 78 hence, move to its left sub-tree.
  2. Since, 40<49<56, traverse right sub-tree of 40.
  3. 49>45, move to right. Compare 49.
  4. match found, return.
- Searching in a B tree depends upon the height of the tree. The search algorithm takes O(log n) time to search any element in a B tree.

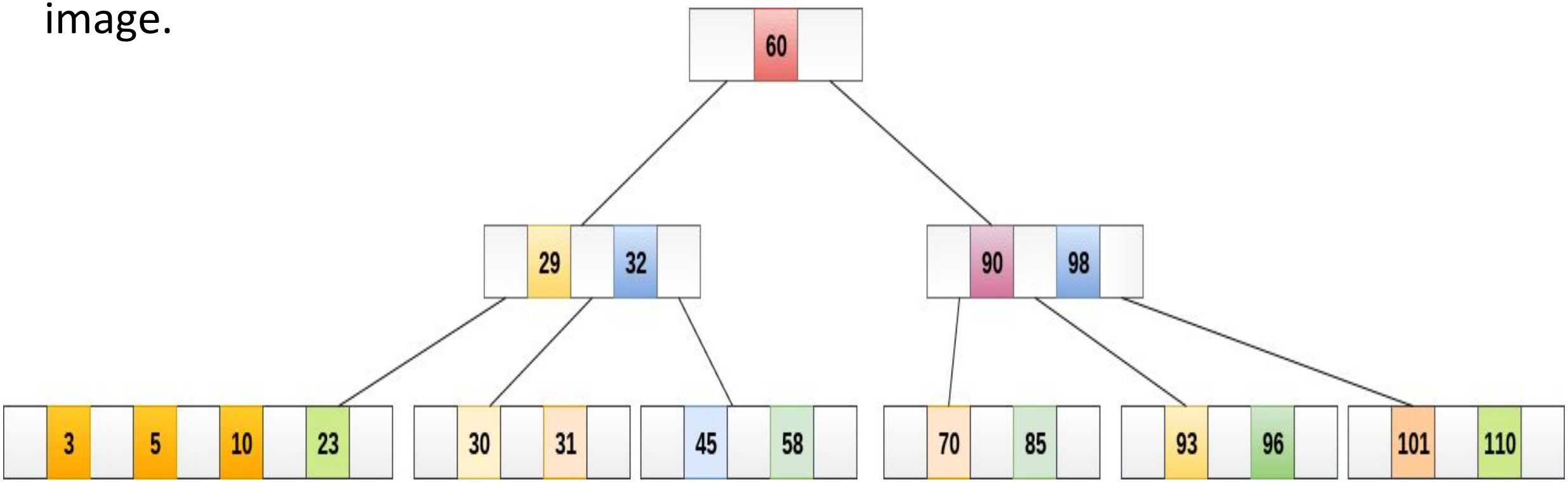


Inserting

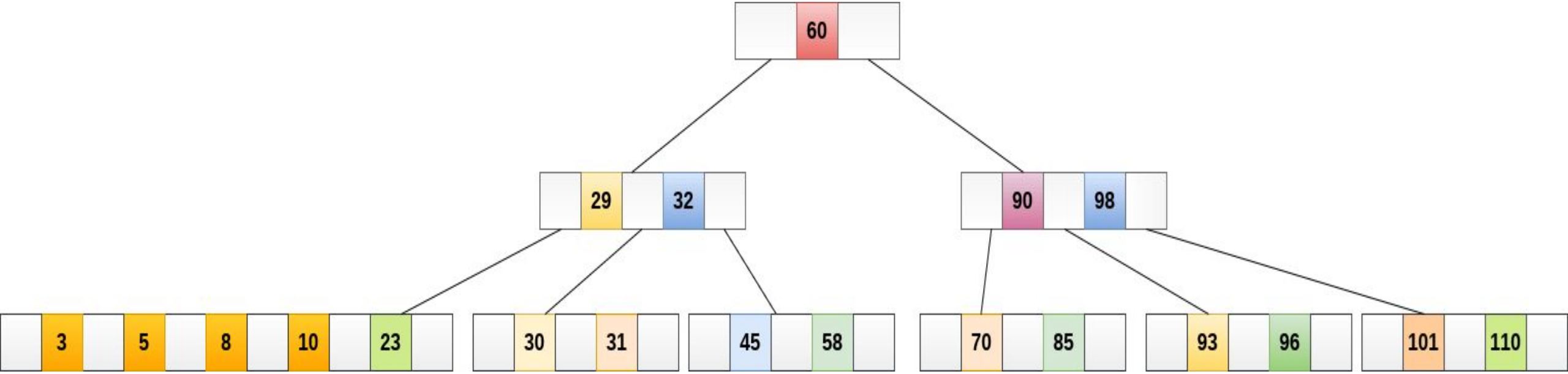
- Insertions are done at the leaf node level. The following algorithm needs to be followed in order to insert an item into B Tree.
  1. Traverse the B Tree in order to find the appropriate leaf node at which the node can be inserted.
  2. If the leaf node contain less than m-1 keys then insert the element in the increasing order.
  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 up to its parent node.
     - If the parent node also contain m-1 number of keys, then split it too by following the same steps.

Example

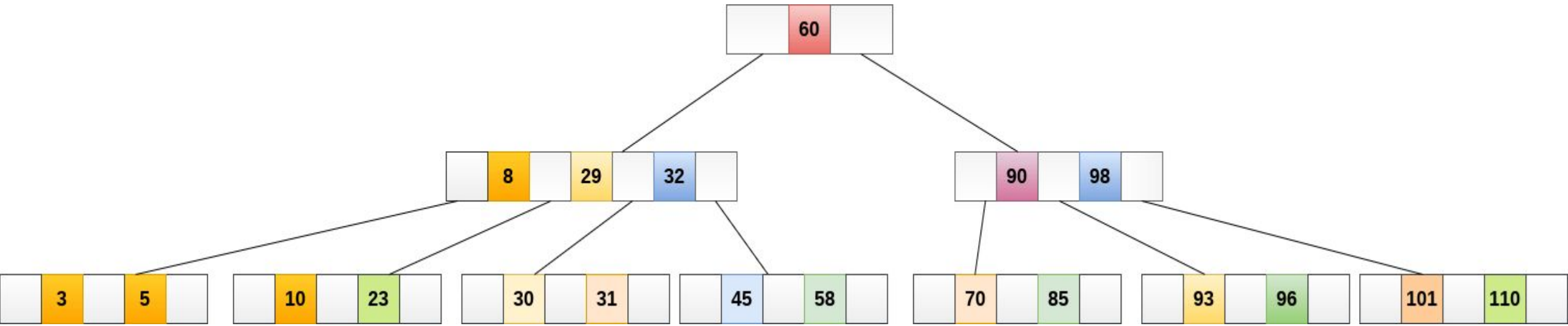
- Insert the node 8 into the B Tree of order 5 shown in the following image.



- 8 will be inserted to the right of 5, therefore insert 8.



- The node, now contain 5 keys which is greater than (5 -1 = 4) keys. Therefore split the node from the median i.e. 8 and push it up to its parent node shown as follows.


60
8
29
32
90
98
3
5
10
23
30
31
45
58
70
85
93
96
101
110

Deletion

- Deletion is also performed at the leaf nodes.
- The node which is to be deleted can either be a leaf node or an internal node.

Cont..

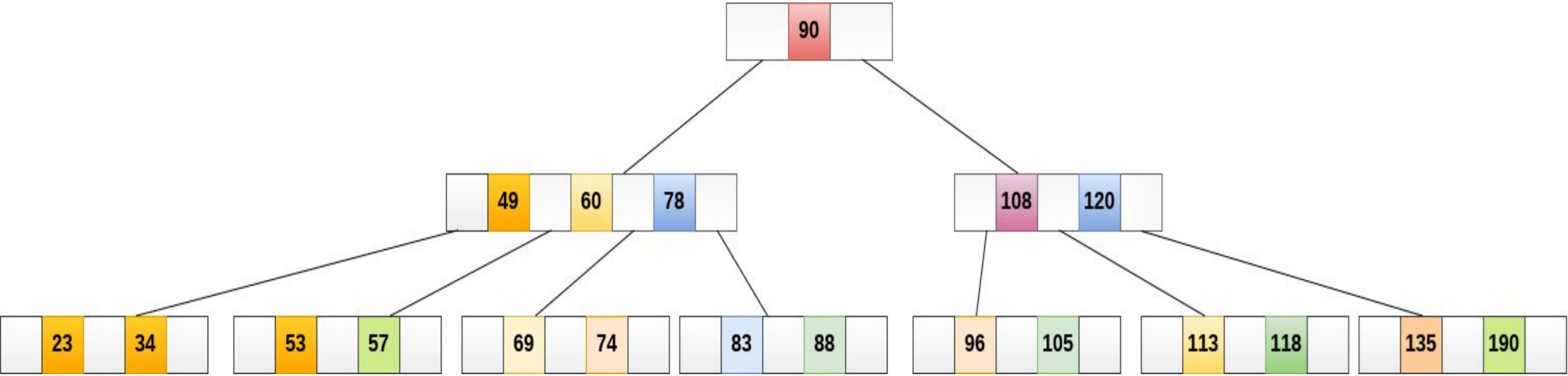
Following algorithm needs to be followed in order to delete a node from a B tree.

- Locate the leaf node.
- If there are more than m/2 keys in the leaf node then delete the desired key from the node.
- If the leaf node doesn't contain m/2 keys then complete the keys by taking the element from right or left sibling.
  - If the left sibling contains more than m/2 elements then push its largest element up to its parent and move the intervening element down to the node where the key is deleted.
  - If the right sibling contains more than m/2 elements then push its smallest element up to the parent and move intervening element down to the node where the key is deleted.
- If neither of the sibling contain more than m/2 elements then create a new leaf node by joining two leaf nodes and the intervening element of the parent node.
- If parent is left with less than m/2 nodes then, apply the above process on the parent too.

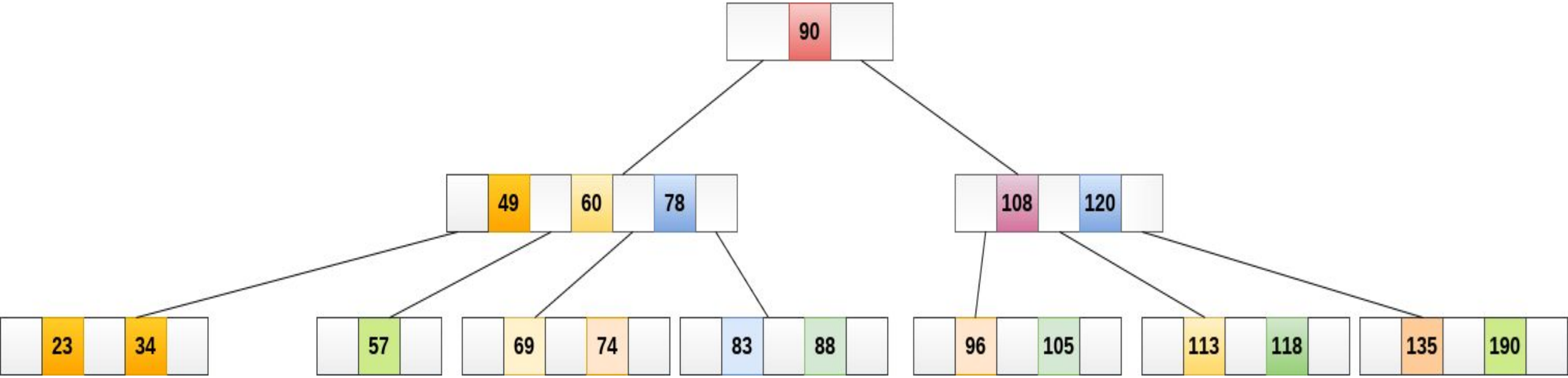
Cont..

- If the the node which is to be deleted is an internal node, then replace the node with its in-order successor or predecessor. Since, successor or predecessor will always be on the leaf node hence, the process will be similar as the node is being deleted from the leaf node.

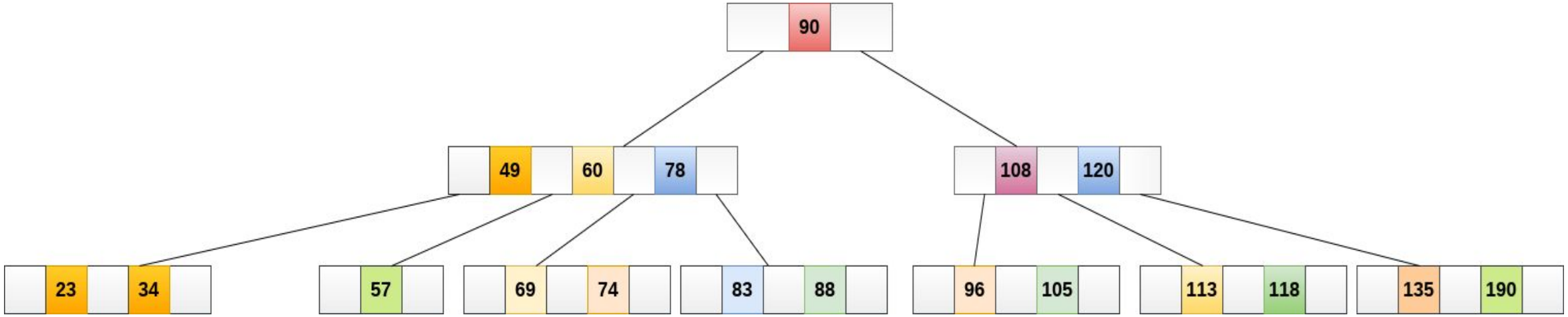
- Delete the node 53 from the B Tree of order 5 shown in the



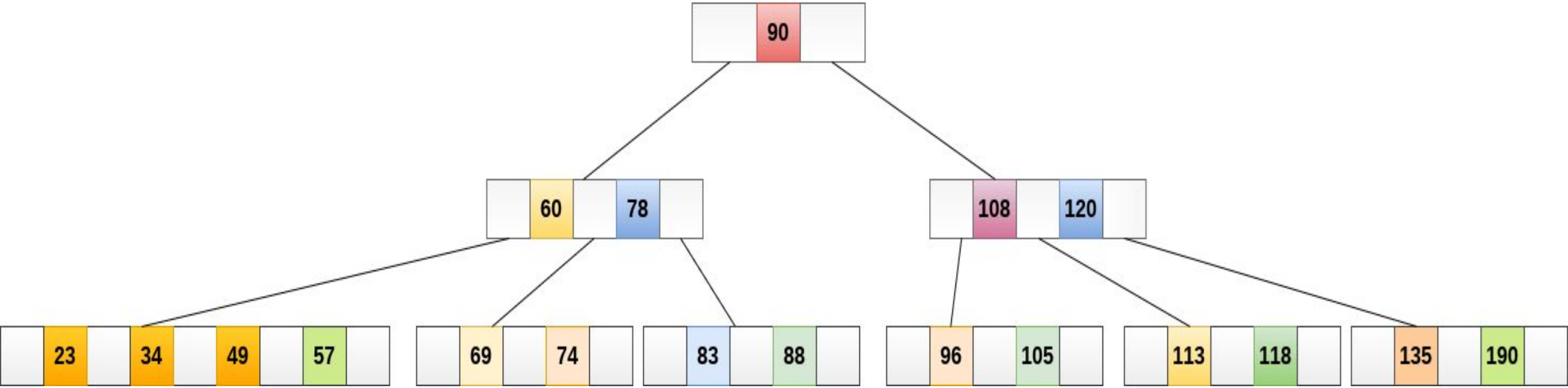
- 53 is present in the right child of element 49. Delete it.



- Now, 57 is the only element which is left in the node, the minimum number of elements that must be present in a B tree of order 5, is 2. it is less than that, the elements in its left and right sub-tree are also not sufficient therefore, merge it with the left sibling and intervening element of parent i.e. 49.



- The final B tree is shown as follows.



Application of B tree

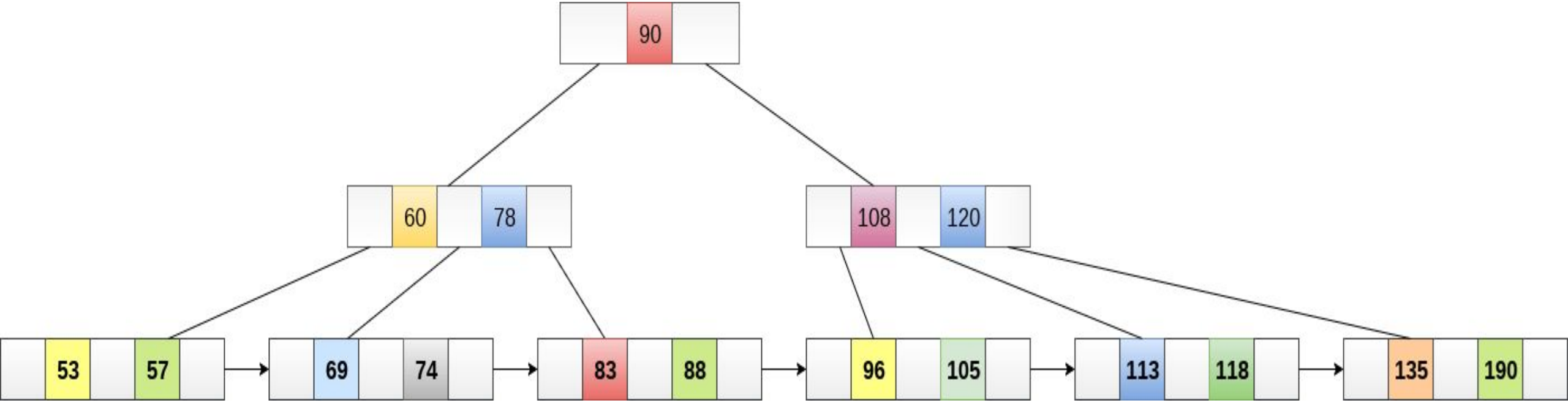
- B tree is used to index the data and provides fast access to the actual data stored on the disks since, the access to value stored in a large database that is stored on a disk is a very time consuming process.
- Searching an un-indexed and unsorted database containing n key values needs O(n) running time in worst case. However, if we use B Tree to index this database, it will be searched in O(log n) time in worst case.

B+ Tree

- B+ Tree is an extension of B Tree which allows efficient insertion, deletion and search operations.
- In B Tree, Keys and records both can be stored in the internal as well as leaf nodes. Whereas, in B+ tree, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values.
- The leaf nodes of a B+ tree are linked together in the form of a singly linked lists to make the search queries more efficient.
- B+ Tree are used to store the large amount of data which can not be stored in the main memory.
- Due to the fact that, size of main memory is always limited, the internal nodes (keys to access records) of the B+ tree are stored in the main memory whereas, leaf nodes are stored in the secondary memory.

B+ tree

- The internal nodes of B+ tree are often called index nodes. A B+ tree of order 3 is shown in the following figure.



Advantages of B+ Tree

- Records can be fetched in equal number of disk accesses.
- Height of the tree remains balanced and less as compare to B tree.
- We can access the data stored in a B+ tree sequentially as well as directly.
- Keys are used for indexing.
- Faster search queries as the data is stored only on the leaf nodes.

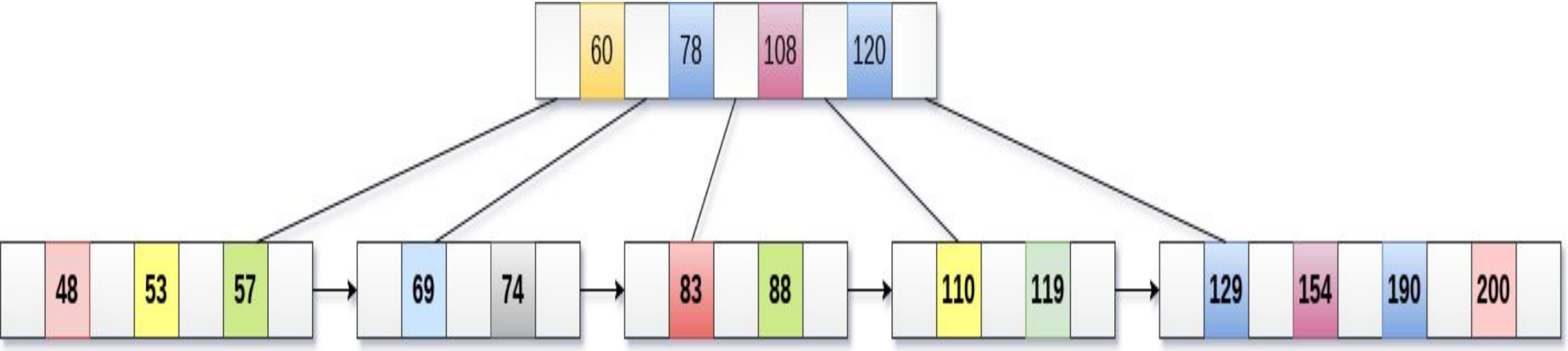
B Tree VS B+ Tree

SN	B Tree	B+ Tree
1	Search keys can not be repeatedly stored.	Redundant search keys can be present.
2	Data can be stored in leaf nodes as well as internal nodes	Data can only be stored on the leaf nodes.
3	Searching for some data is a slower process since data can be found on internal nodes as well as on the leaf nodes.	Searching is comparatively faster as data can only be found on the leaf nodes.
4	Deletion of internal nodes are so complicated and time consuming.	Deletion will never be a complexed process since element will always be deleted from the leaf nodes.
5	Leaf nodes can not be linked together.	Leaf nodes are linked together to make the search operations more efficient.

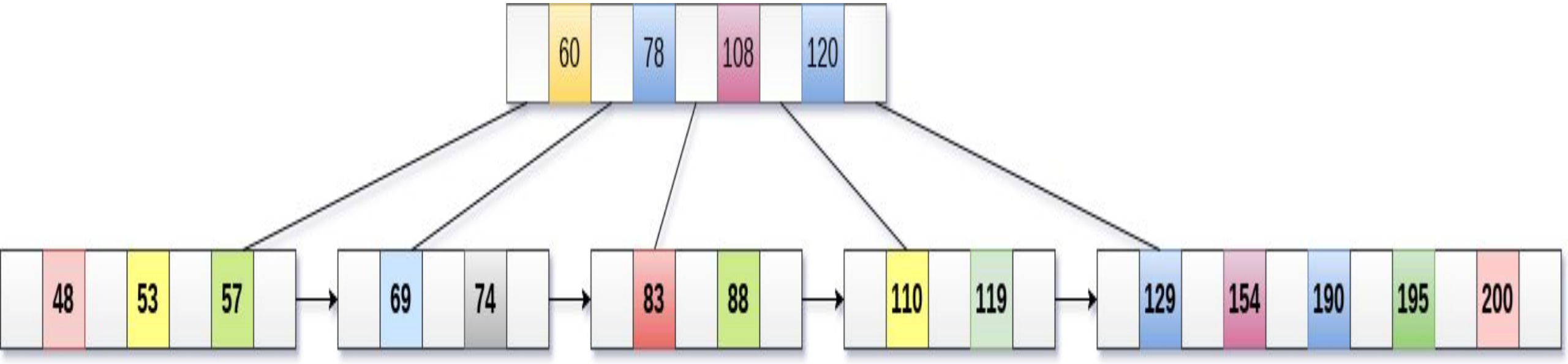
Insertion in B+ Tree

- **Step 1:** Insert the new node as a leaf node
- **Step 2:** If the leaf doesn't have required space, split the node and copy the middle node to the next index node.
- **Step 3:** If the index node doesn't have required space, split the node and copy the middle element to the next index page.

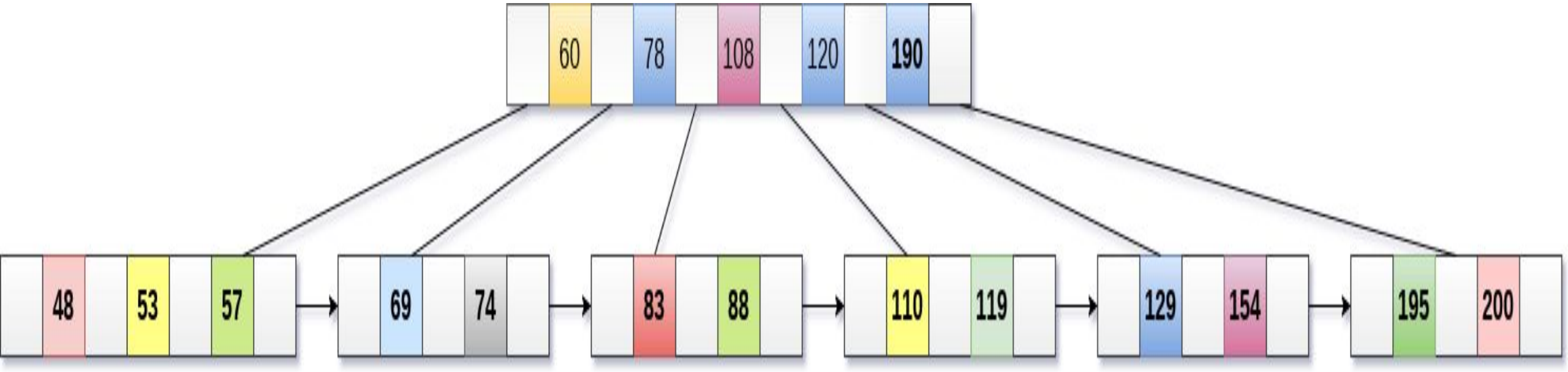
- Insert the value 195 into the B+ tree of order 5 shown in the following figure



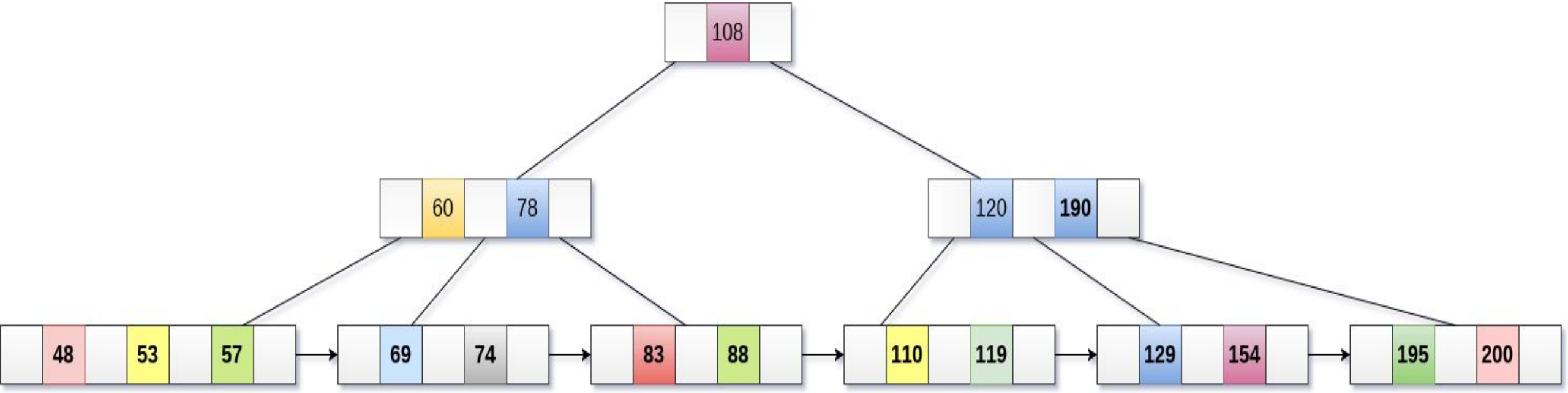
- 195 will be inserted in the right sub-tree of 120 after 190. Insert it at the desired position.



- The node contains greater than the maximum number of elements i.e. 4, therefore split it and place the median node up to the parent.



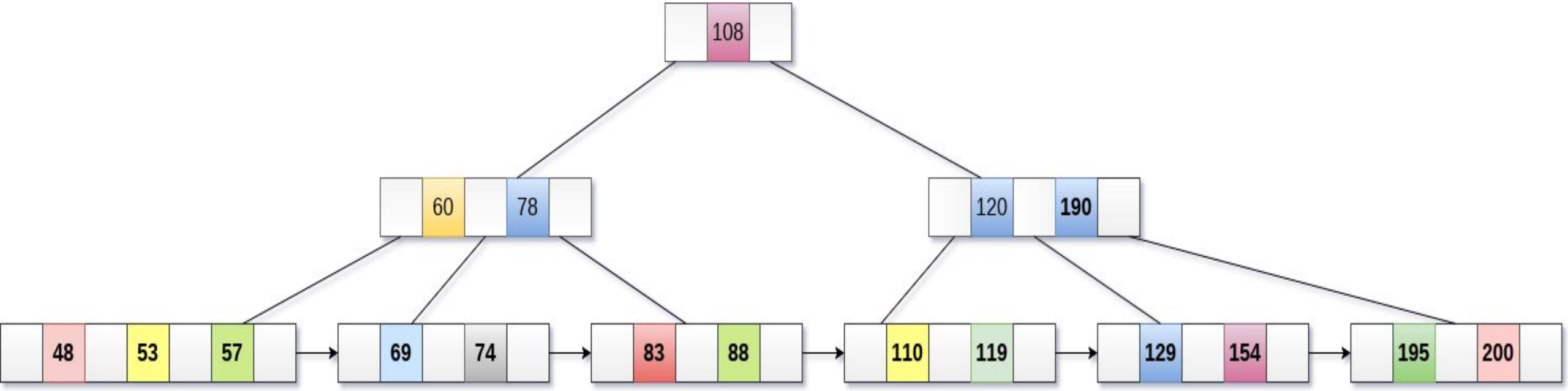
- Now, the index node contains 6 children and 5 keys which violates the B+ tree properties, therefore we need to split it, shown as follows.



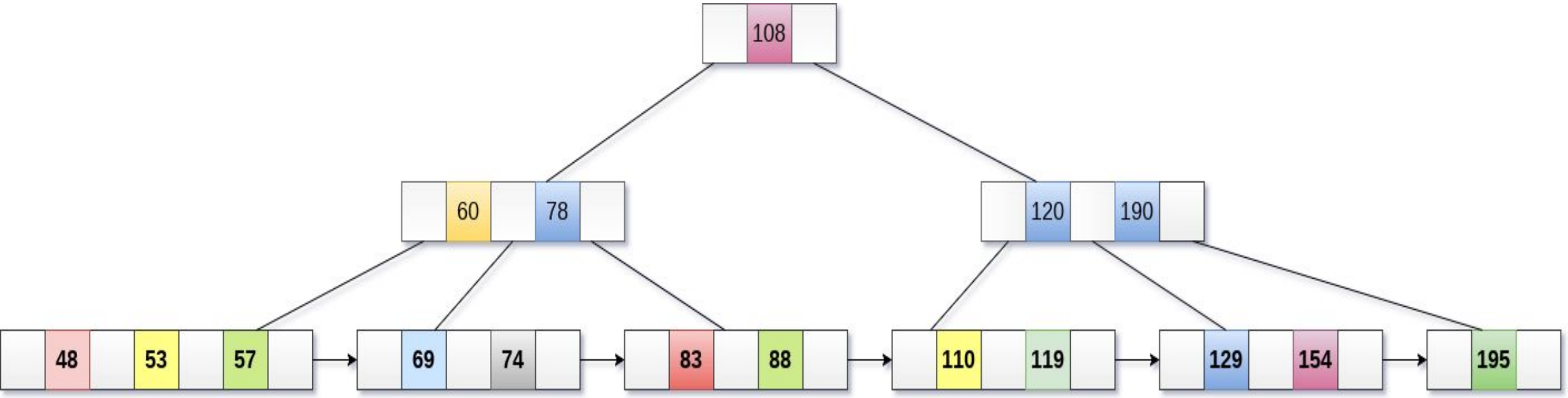
Deletion in B+ Tree

- **Step 1:** Delete the key and data from the leaves.
- **Step 2:** if the leaf node contains less than minimum number of elements, merge down the node with its sibling and delete the key in between them.
- **Step 3:** if the index node contains less than minimum number of elements, merge the node with the sibling and move down the key in between them.
-

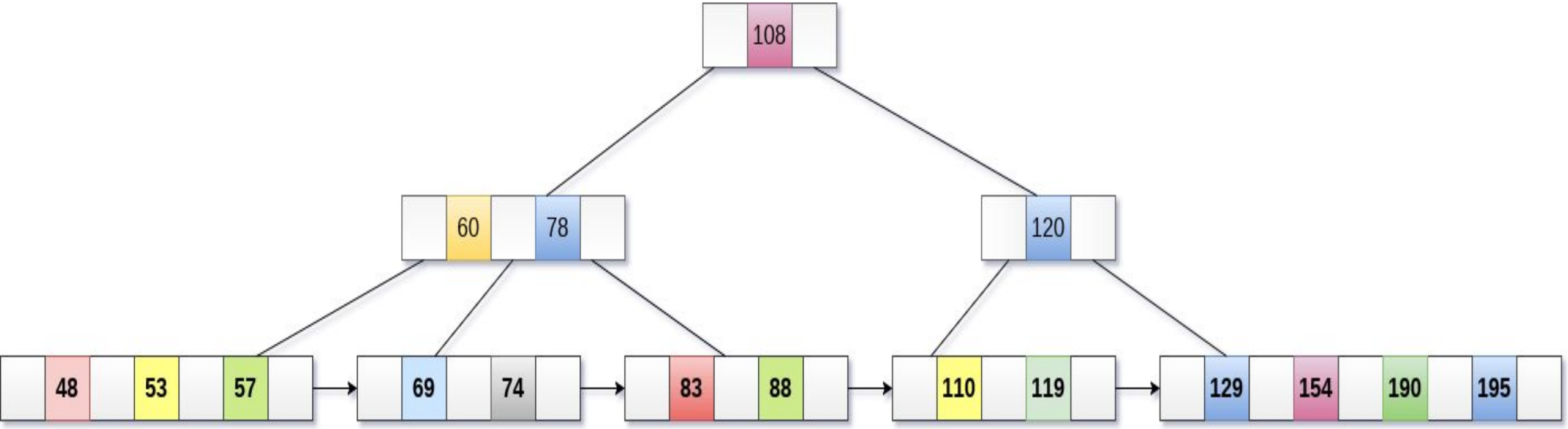
- Delete the key 200 from the B+ Tree shown in the following figure.



200 is present in the right sub-tree of 190, after 195. delete it.

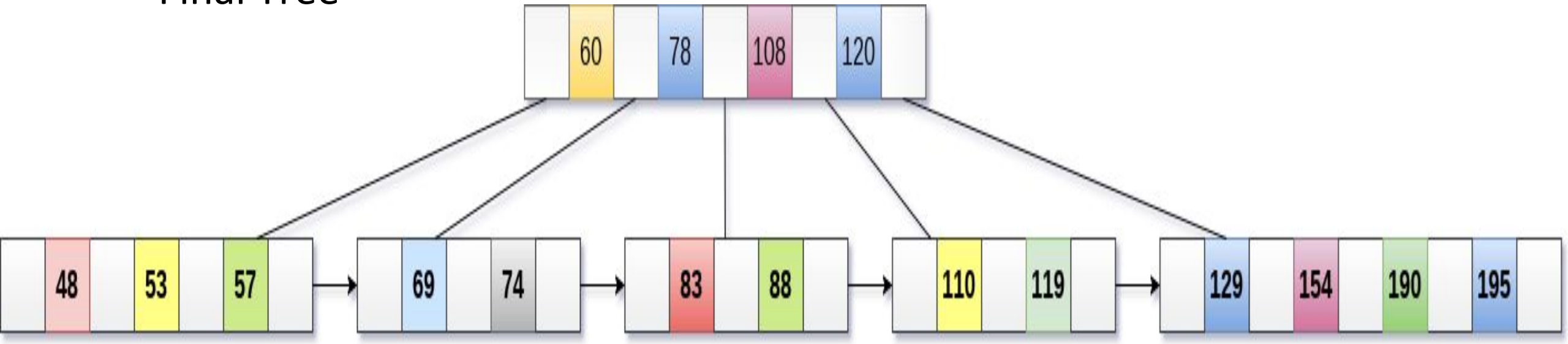


- Merge the two nodes by using 195, 190, 154 and 129.



- Now, element 120 is the single element present in the node which is violating the B+ Tree properties. Therefore, we need to merge it by using 60, 78, 108 and 120.
- Now, the height of B+ tree will be decreased by 1.

- Final Tree



Red-Black Tree

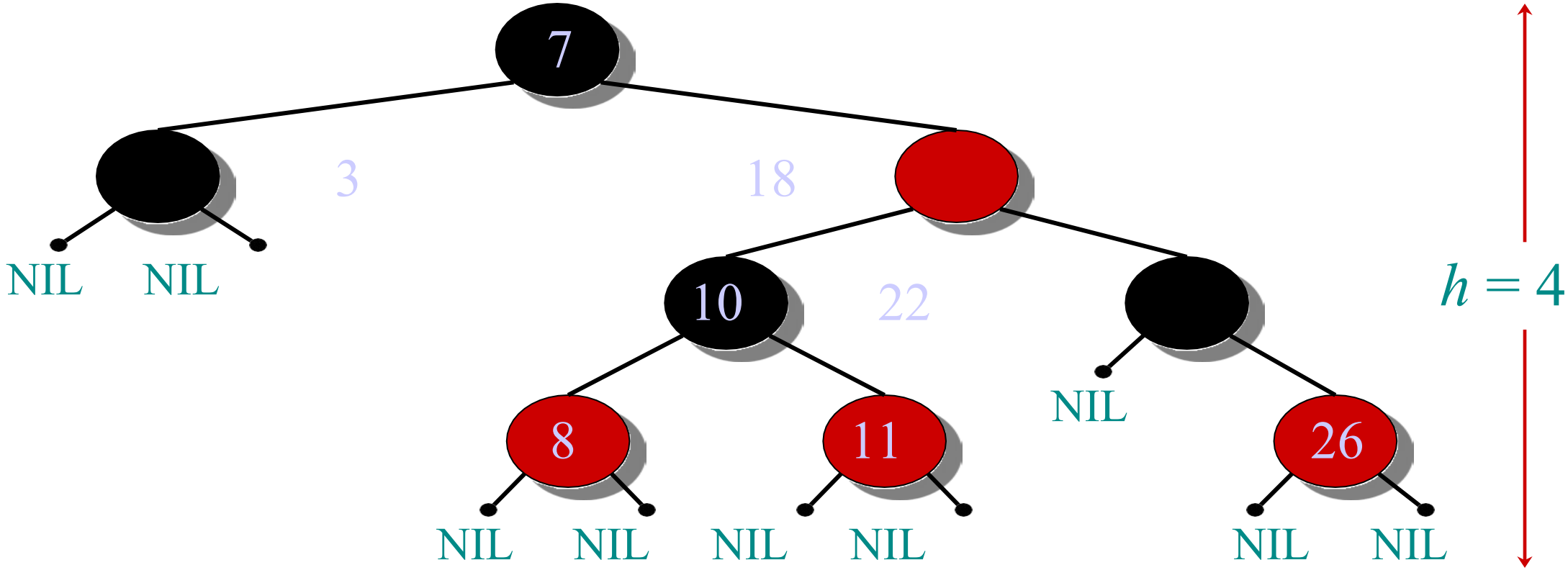
Red-black trees

- Red-Black tree is a self-balancing binary search tree in which each node contains an extra bit for denoting the color of the node, either red or black.
- This data structure requires an extra one- bit color field in each node.

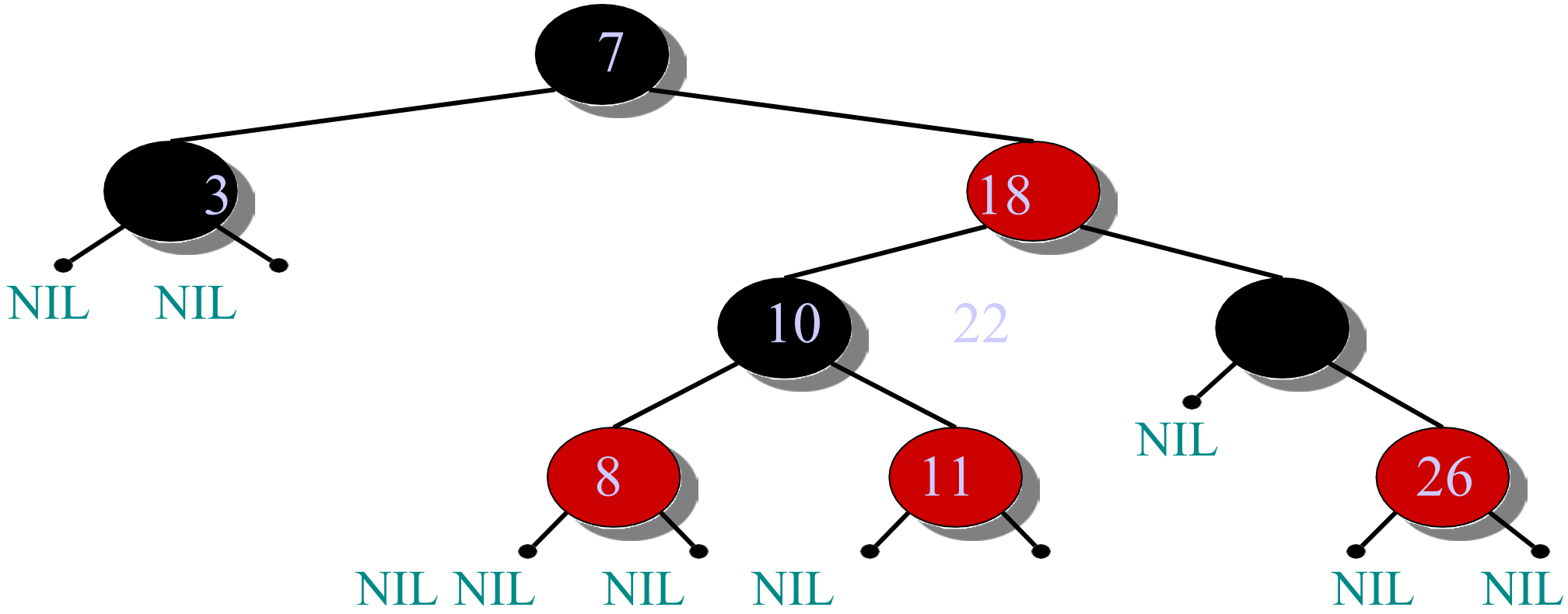
Red-black properties:

1. Every node is either red or black.
2. The root and leaves (NIL's) are black.
3. If a node is red, then its parent is black.
4. All simple paths from any node x to a descendant leaf have the same number of black nodes = black-height(x).

Example of a red-black tree

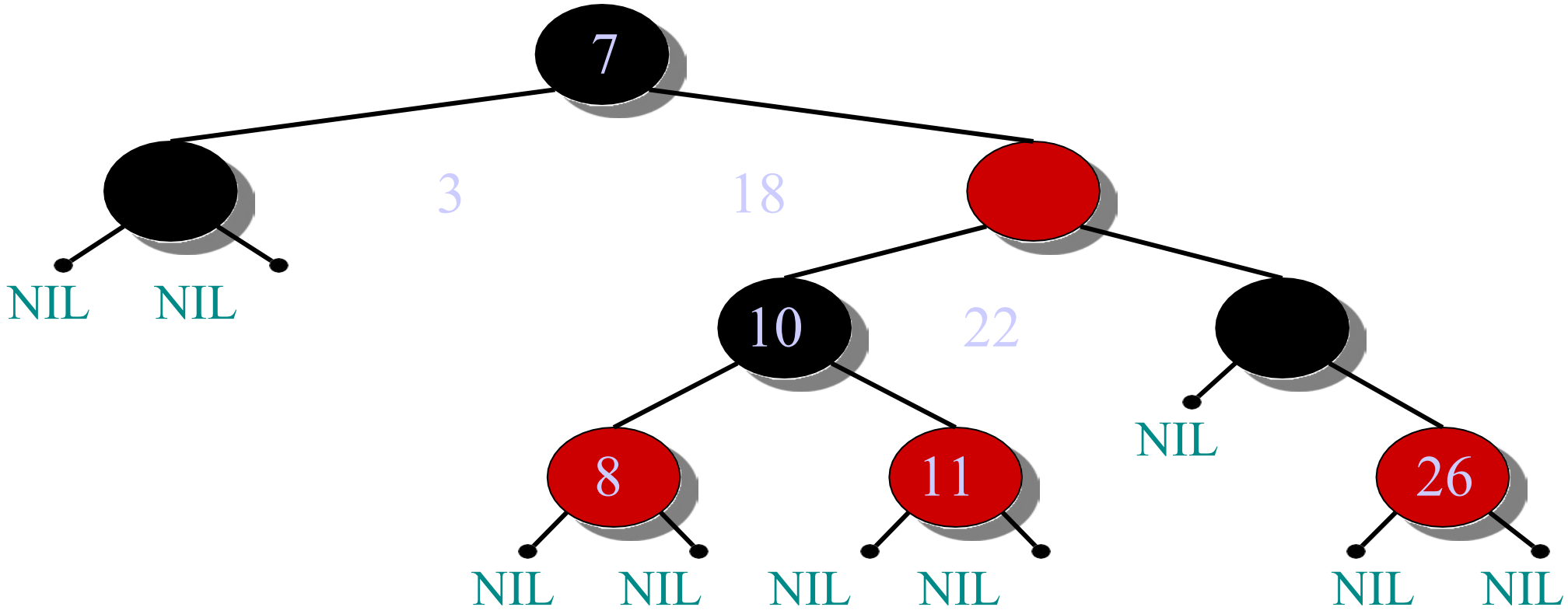


Example of a red-black tree



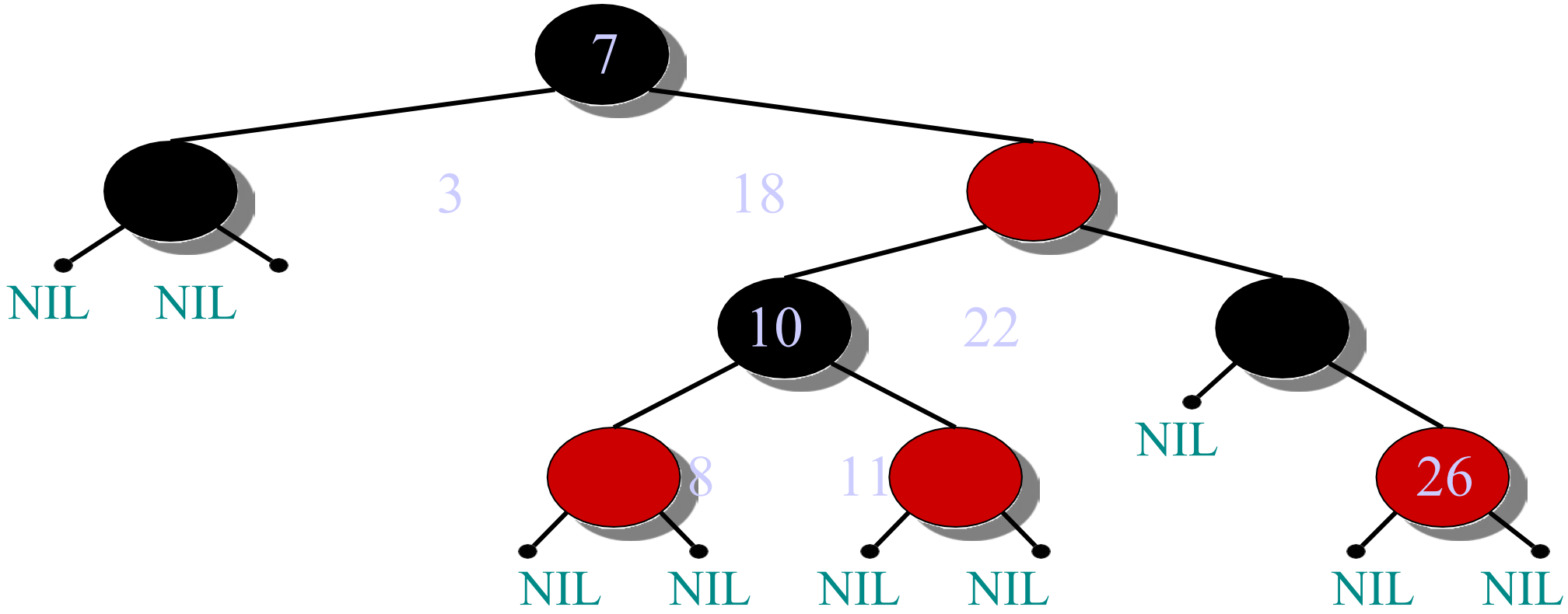
1. Every node is either red or black.

Example of a red-black tree



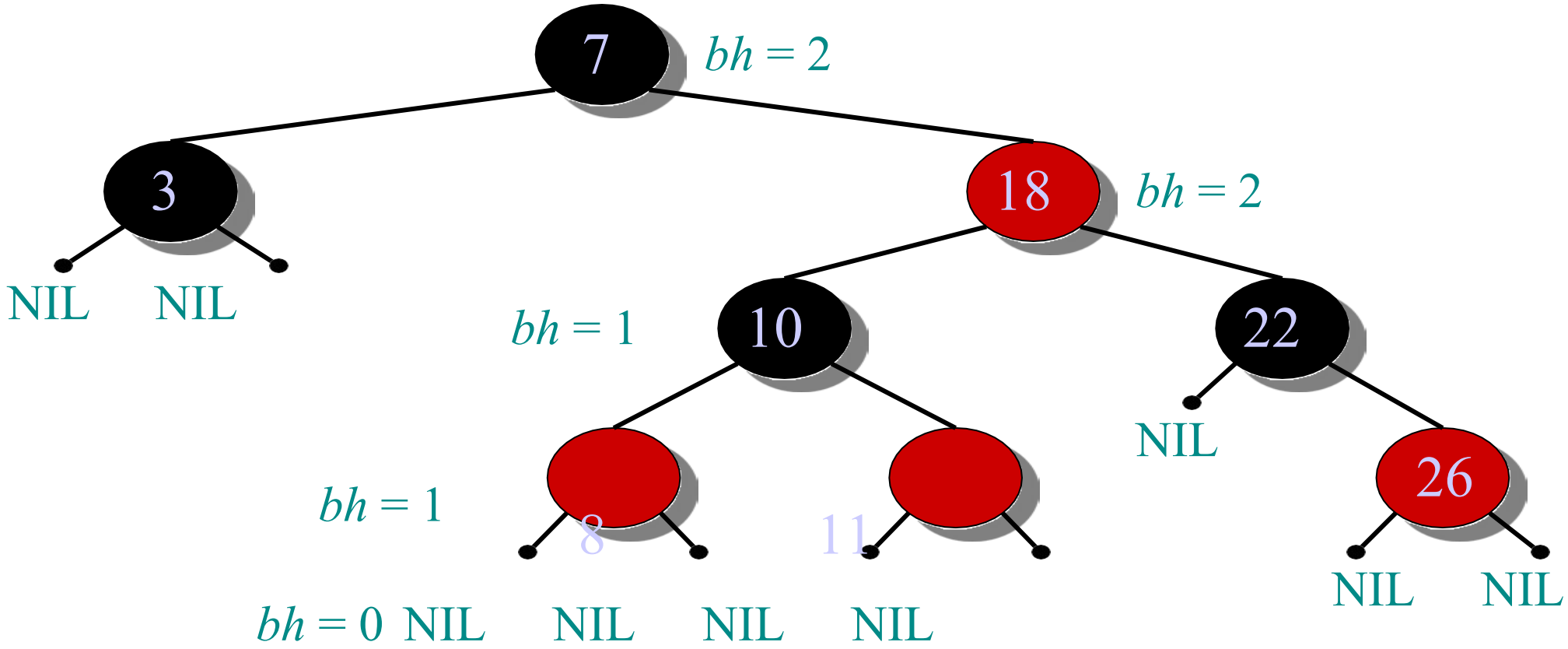
2. The root and leaves (NIL's) are black.

Example of a red-black tree



3. If a node is red, then its parent is black.

Example of a red-black tree



4. All simple paths from any node x to a descendant leaf have the same number of black nodes = *black-height*(x).

Insertion in Red-Black Tree

- While inserting a new node, the new node is always inserted as a RED node. After insertion of a new node, if the tree is violating the properties of the red-black tree then, we do the following operations.
- Recolor
- Rotation

Insertion in Red-Black Tree

1. If tree is empty, create new node as root node with color Black
2. If tree is not empty, create new node as leaf node with color Red
3. If parent of new node is black then exit
4. If parent of new node is Red, then check the color of Parent's sibiling of network.
   a) If color is black or null then do suitable rotation and recolor
   b) If color is Red then recolor and also check if parent 's parent of new node is not root node then recolor it and recheck

- 10,18,7,15,16,30,25,40,60,2,1,70