Part-1: Construct a B-tree of order 3:

- Inserting the following values in the given order.
- 50, 20, 75, 10, 60, 40, 90, 30, 80, 100, 15, 70, 85, 25, 65, 35, 95, and 5

B-tree of order-3

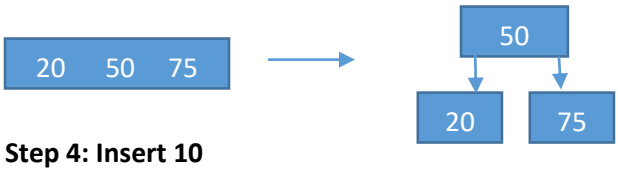
- Each node can contain at most 2 keys (order - 1).
- Each node can have at most 3 children.

Step by step solution of part-1.

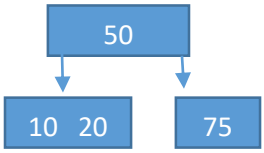
Step 1: Insert 50

Step 2: Insert 20

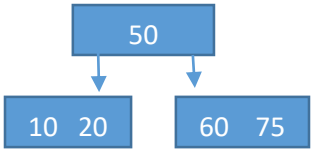
Step 3: Insert 75.
Split nodes if they exceed the maximum allowed keys (2 in this case)



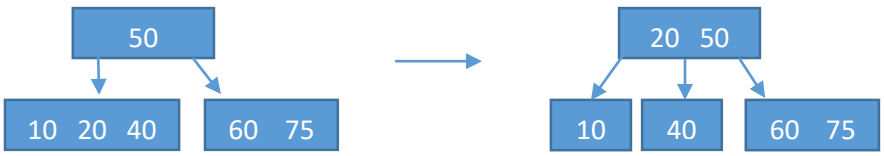
Step 4: Insert 10



Step 5: Insert 60



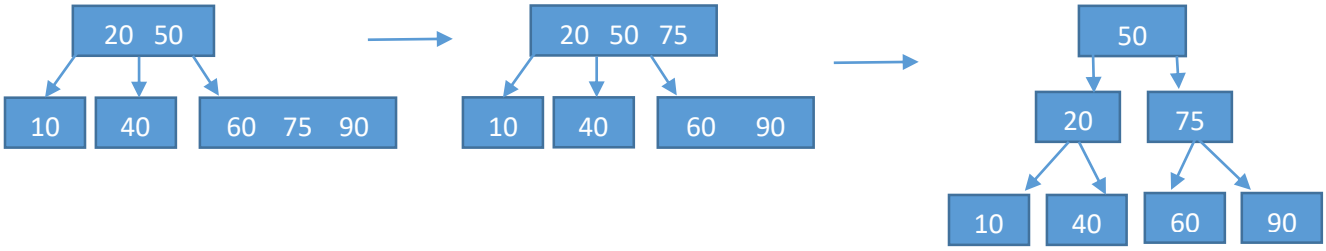
Step 6: Insert 40
After inserting 40 the node will exceed the maximum number of keys.
The key 20 will be promoted



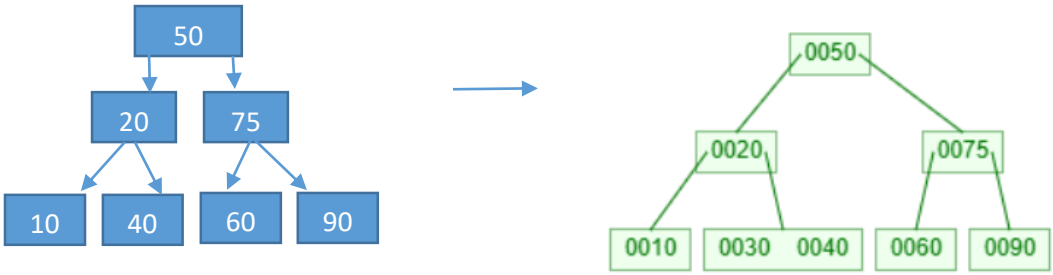
Step 7: Insert 90

Split the node and promote 75 because inserting 90 will result in exceeding the maximum number of keys in this node

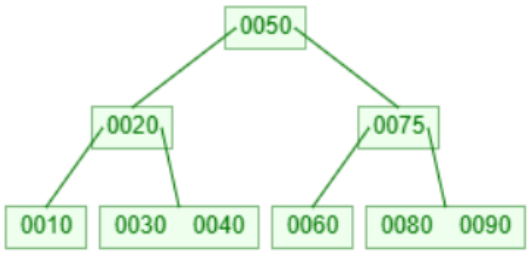
After promoting 75, you will find that the root node is also exceeding the maximum number of keys so split the root node and promote the middle value which is fifty.



Step 8: Insert 30.

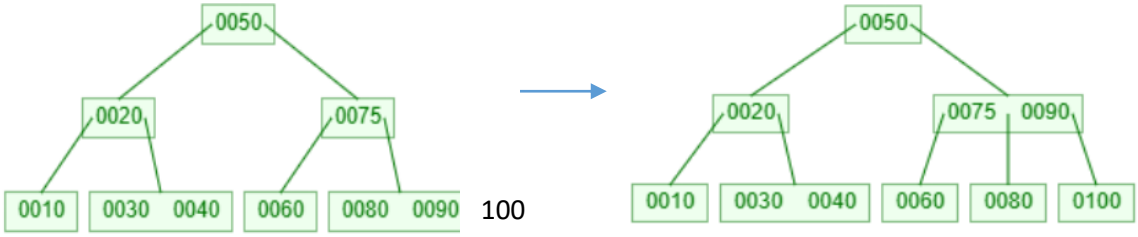


Step 9: Insert 80.

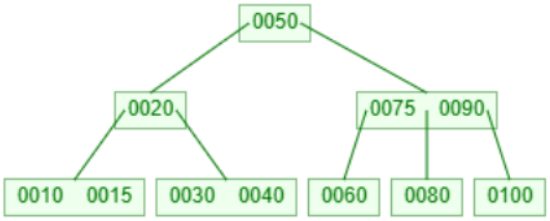


Step 10: Insert 100.

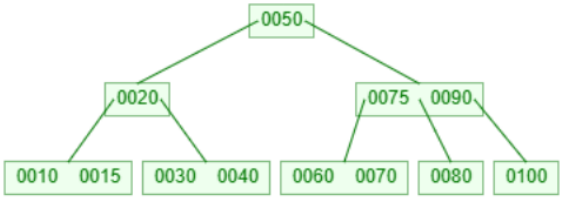
After inserting 100, the node will exceed the maximum number of keys.
The key 90 will be promoted.



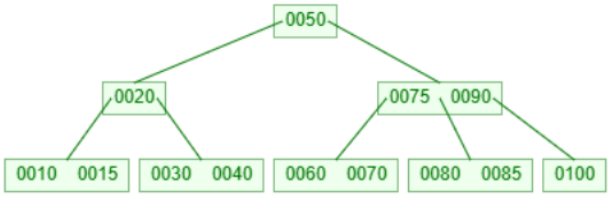
Step 11: Insert 15.



Step 12: Insert 70.



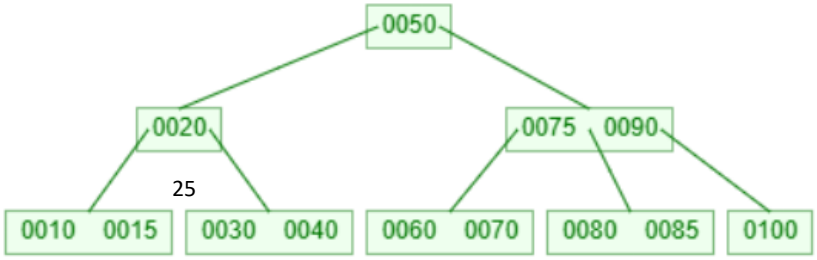
Step 13: Insert 85.



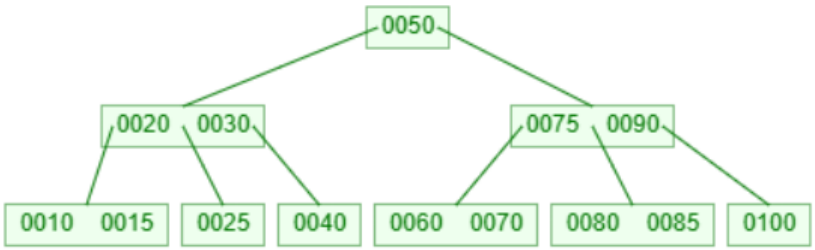
Step 14: Insert 25.

After inserting 25, the node will exceed the maximum number of keys.

The key 30 will be promoted



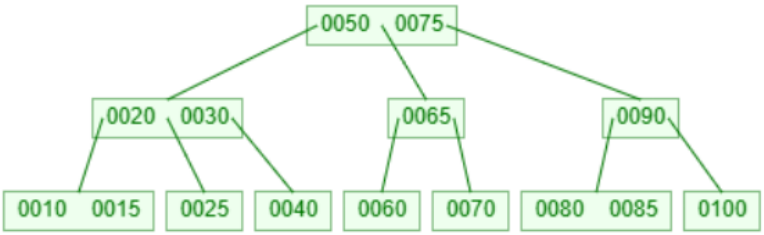
Output of step 14.



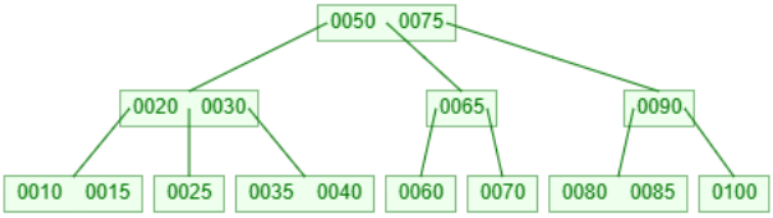
Step 15: Insert 65

After inserting 65, the node will exceed the maximum number of keys.

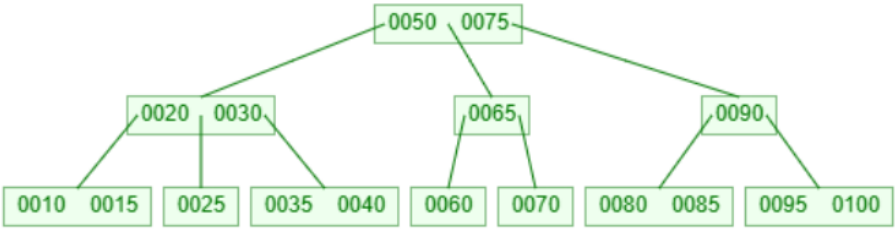
The key 30 will be promoted.



Step 16: Inserting 35.

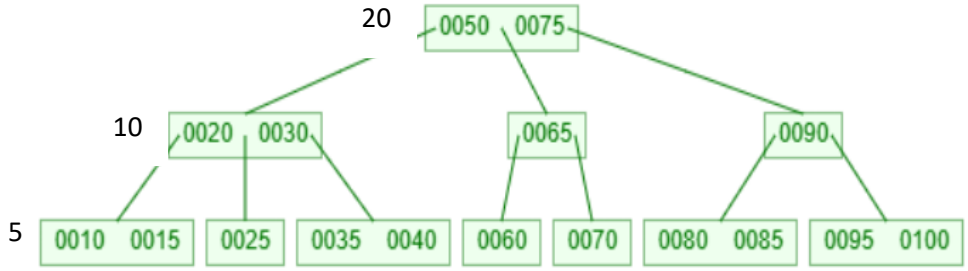


Step 17: Inserting 95.

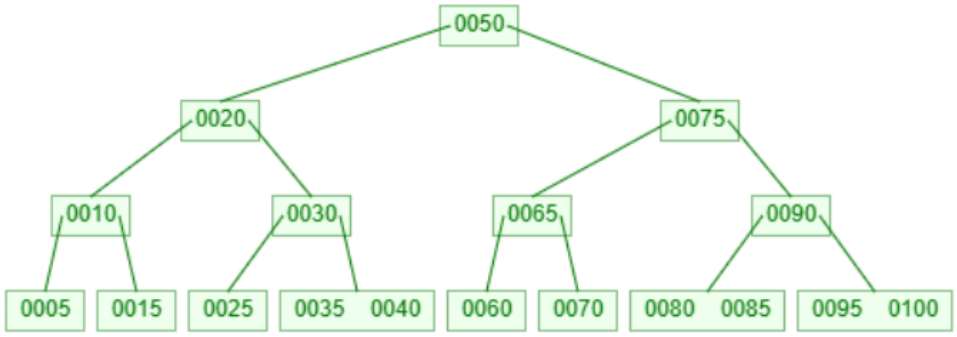


Step 18: Insert 5.

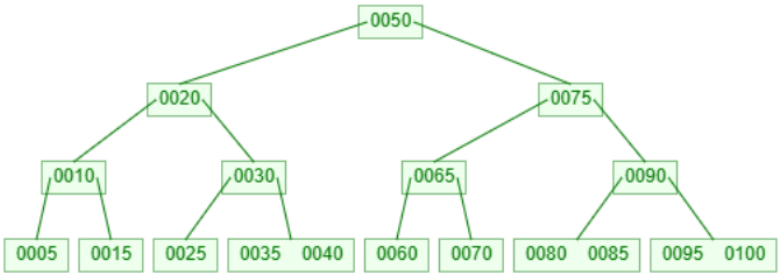
- After inserting 5, the node will exceed the maximum number of keys, requiring a split.
- The key 10 will be promoted.
- The parent node will also exceed the maximum limit, causing 20 to be promoted.
- The parent node of 20 will again exceed the limit, resulting in 50 being promoted as the root node.



Final B-tree after Inserting 5.



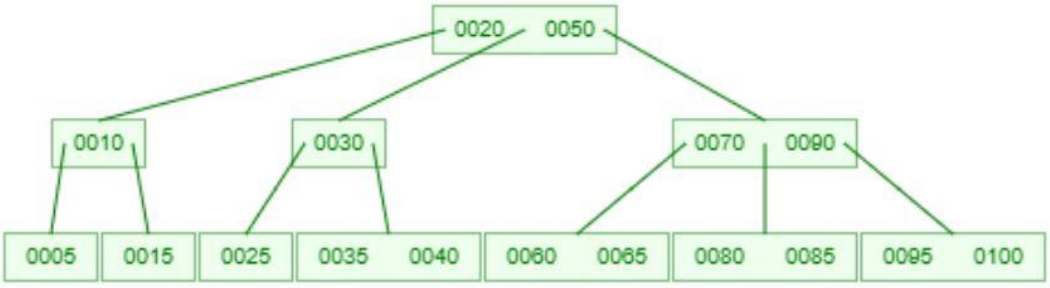
Part 2 Deletion



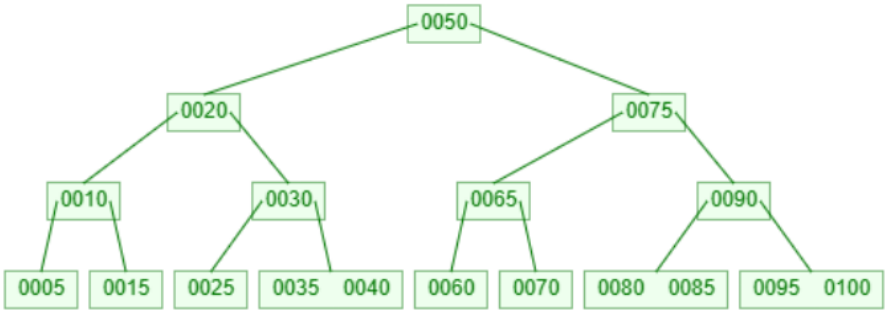
Delete 75

- 75 has two children 65 and 90, so it cannot be deleted directly. Instead, we replace it with either its **in-order predecessor** or **in-order successor**.
- The in-order predecessor of 75 is the largest key in its left subtree, which is 70. So replace 75 with 70.
- After replacing 75 with its in-order predecessor, 70 needs to be removed from its original position so the right child of 65 becomes empty due to deletion of 70. This causes an underflow in the right child of 65.
- To fix this borrow a key from the parent so 65 will come down and merged with 60 and leave an empty space behind.
- In similar way 75 will come down to fill the space and merged with 90.
- Lastly 50 will come down and merged with 20.

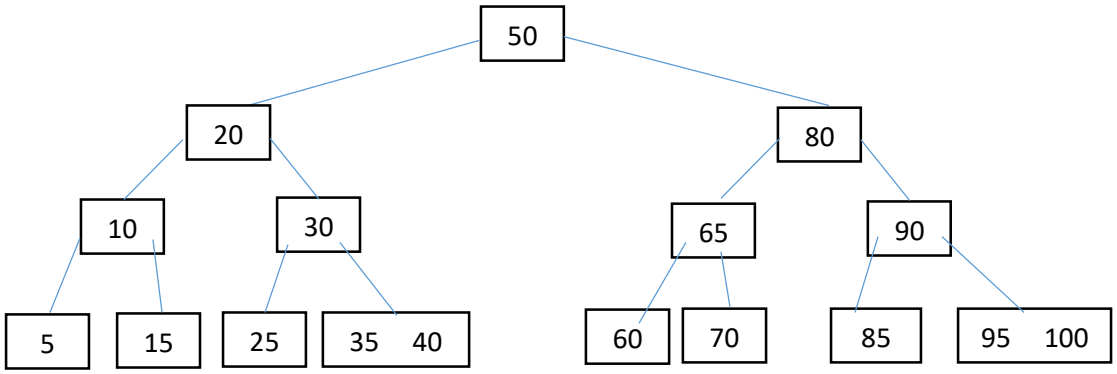
Here is the output B-tree.



Second Possible Solution with in order successor



- The in-order successor of 75 is the smallest key in its right subtree, which is 80.
- Replace 75 with 80.
- 80 is in the left child of 90, which contains the keys [80, 85].
- After removing 80, the node [80, 85] becomes [85]. Since it still has at least one key, no rebalancing is needed here.



Delete 40

Deletion of 40 is simple just locate the value and delete it.

Right node 30 contain two values 35 and 40 so if you will delete 40 then 35 will be remaining.

B-tree of order 3 can have minimum one value so after deleting 40 you don’t need to rebalance the tree.