Tree

Instructors:
Md Nazrul Islam Mondal &
Rizoan Toufiq
Department of Computer Science & Engineering
Rajshahi University of Engineering &
Technology Rajshahi-6204

Outline

- Binary Tree
- Representing Binary Trees in Memory
- Traversing Binary Trees
- Traversal Algorithm using Stacks
- Header Nodes: Threads
- Binary Search Trees
- Searching and Inserting in Binary Search Trees
- Deleting in Binary Search Tree
- AVL Search Trees
- Insertion in an AVL Search Tree
- Deletion in an AVL Search Tree
- **m-way Search Trees**
- **Searching, Insertion and Deletion in an m-way Search Tree**
- **B Trees**
- **Searching, Insertion and Deletion in a B-tree**

m-way Search Trees

m-way Search Trees

- **Application**: File Indexing
- **Goal**: to minimize the accesses while retrieving a key from a file
- Height of n elements:
  - Minimum Height: logm(n+1)
  - Maximum Height: n
- B trees are balanced m-way search tree.

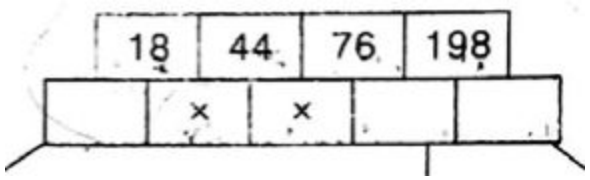
m-way Search Trees (Definition)

- An m-way search tree T may be an empty tree.
- If T is non empty, it satisfies the following properties:
  - Each node has at most m (order of the tree) child nodes, a node may be represented as A_0, (K_1, A_1), (K_2, A_2) … (K_{m-1},A_{m-1}) where, K_i, $1 \le i \le m-1$, are keys and A_i, $0 \le i \le m-1$ are the pointers to subtrees of T

	K_1	K_2	K_3	K_4
A_0	A_1	A_2	A_3	A_4

m-way Search Trees (Definition)

- **If T is non empty, it satisfies the following properties:**
  - If a node has k child where k≤m, then the node have only (k-1) keys, $K_1, K_2, \ldots, K_{k-1}$ contained nodes such that $K_i < K_{i+1}$ and each of the keys partitions all the keys in the subtree into k subsets.

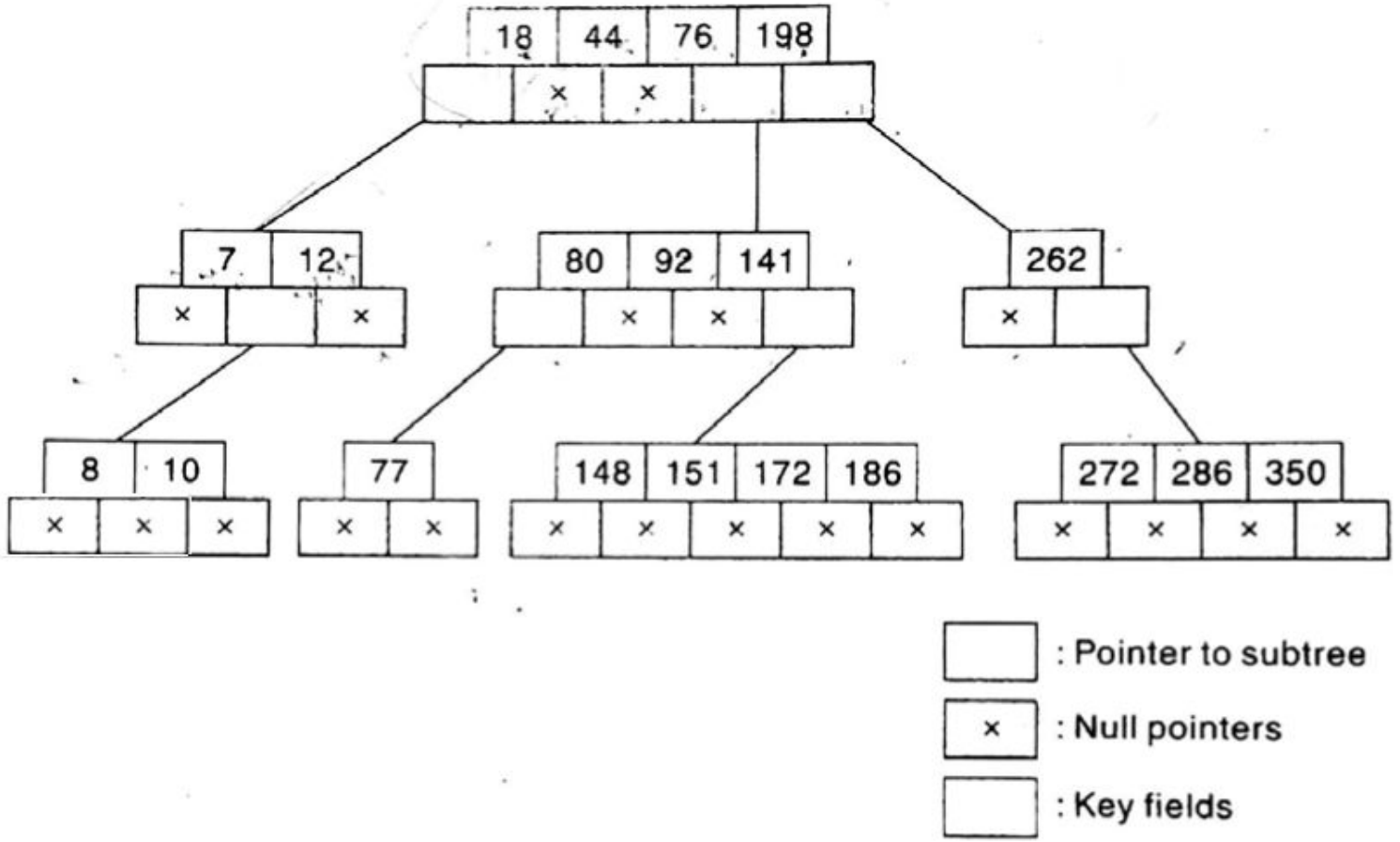


  - 18<44<76<198
  - A_0 point subtree which contain the keys less than 18
  - A_1 points subtree which contain the keys less then 44
  - So on
  - Keys partitions all the keys in the subtree into k subsets

m-way Search Trees (Definition)

- **If T is non empty, it satisfies the following properties:**
  - For a node A_0, (K_1,A_1), (K_2, A_2)… (K_{m-1}, A_{m-1}), all key values in the subtree pointed to by A_i are less than the key K_{i+1} , $0 \leq i \leq m-2$, and all key values in the subtree pointed to by A_{m-1} are greater than K_{m-1}.
  - Each of the subtrees A_i , $0 \leq i \leq m-1$, are also m-way search tree

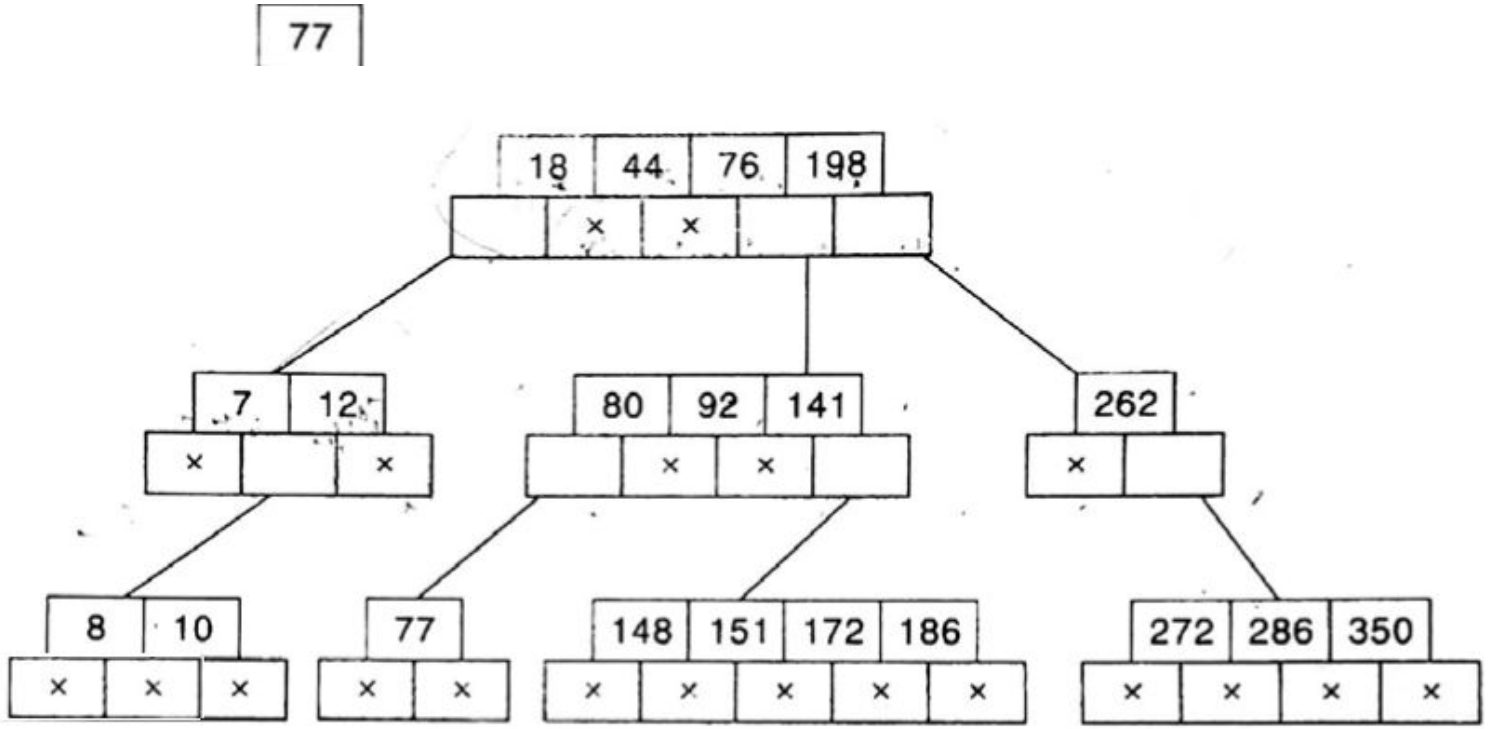
m-way Search Trees (Definition)



Searching, Insertion and Deletion in an m-way Search Tree

Searching

- Search 77.

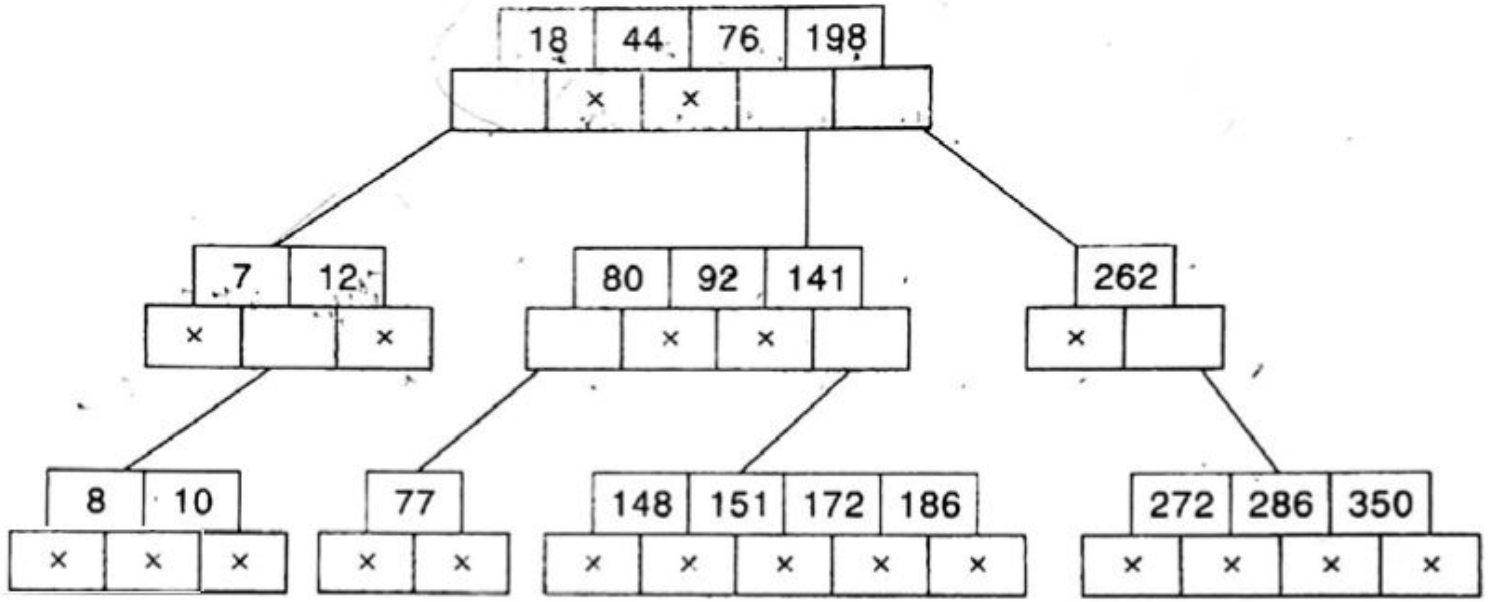


Searching

- Search 13.

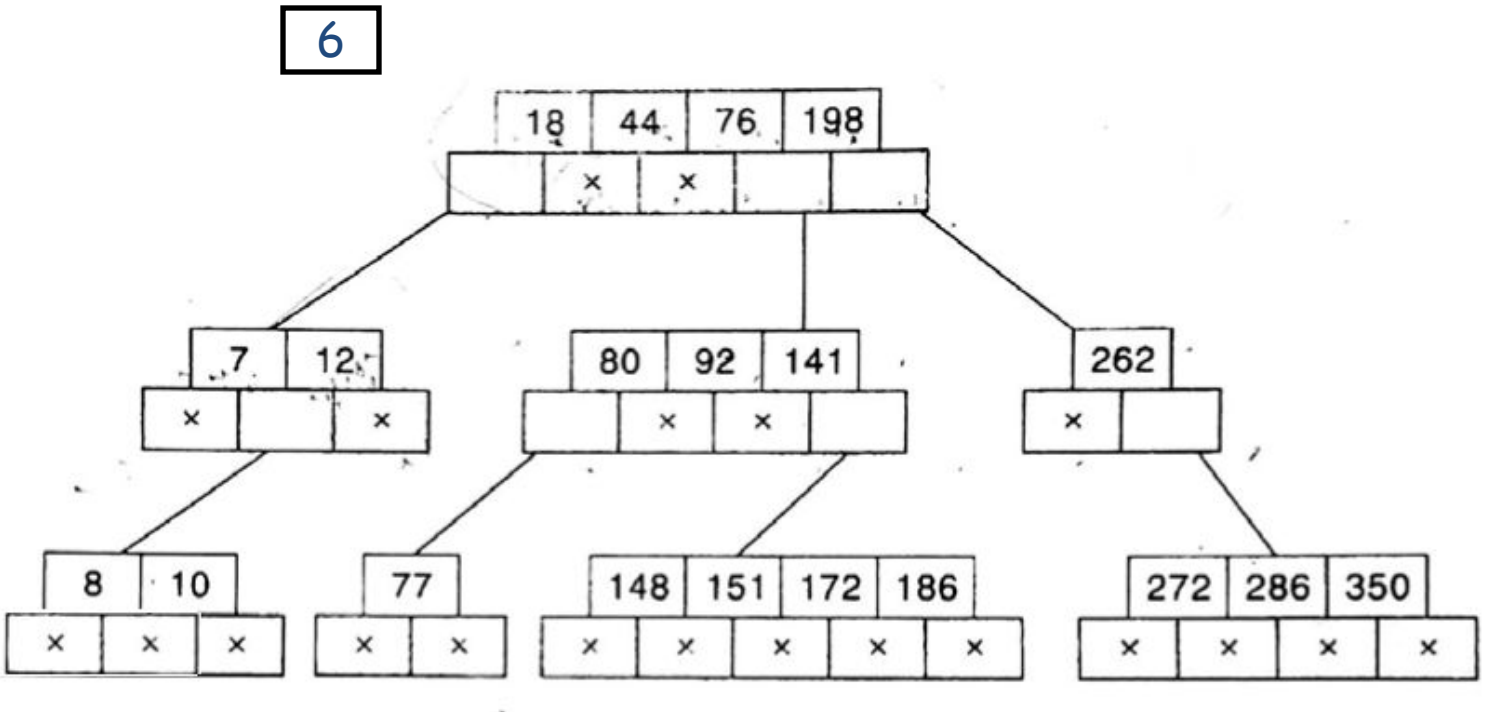
13

Unsuccessful



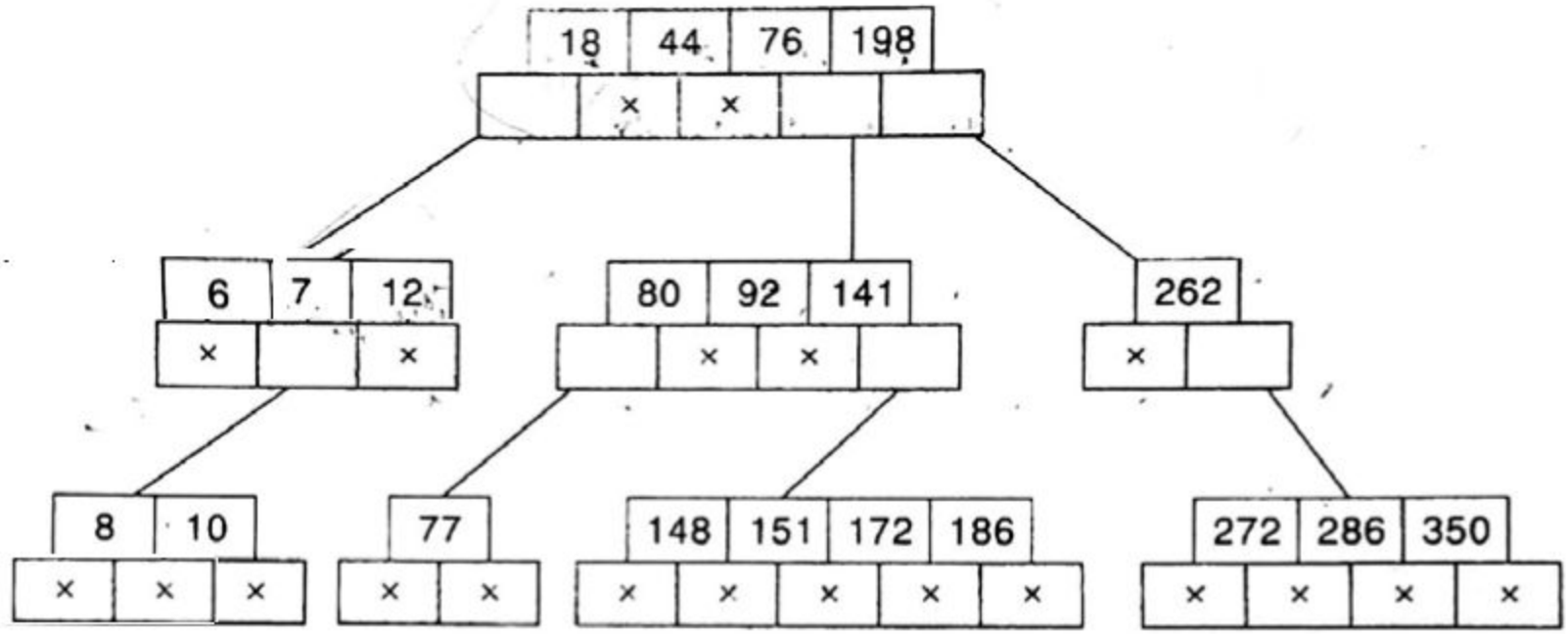
Insertion

- Insert 6



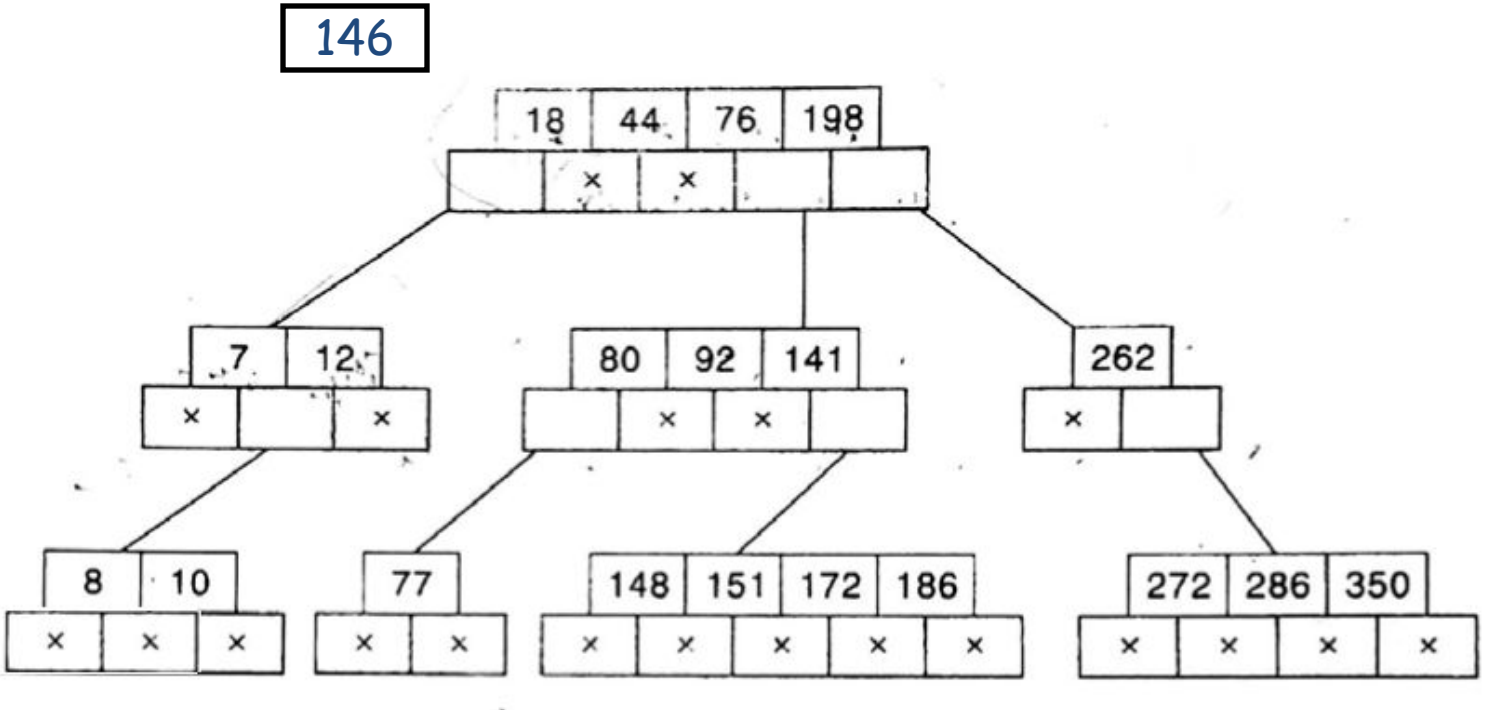
Insertion

- Insert 6



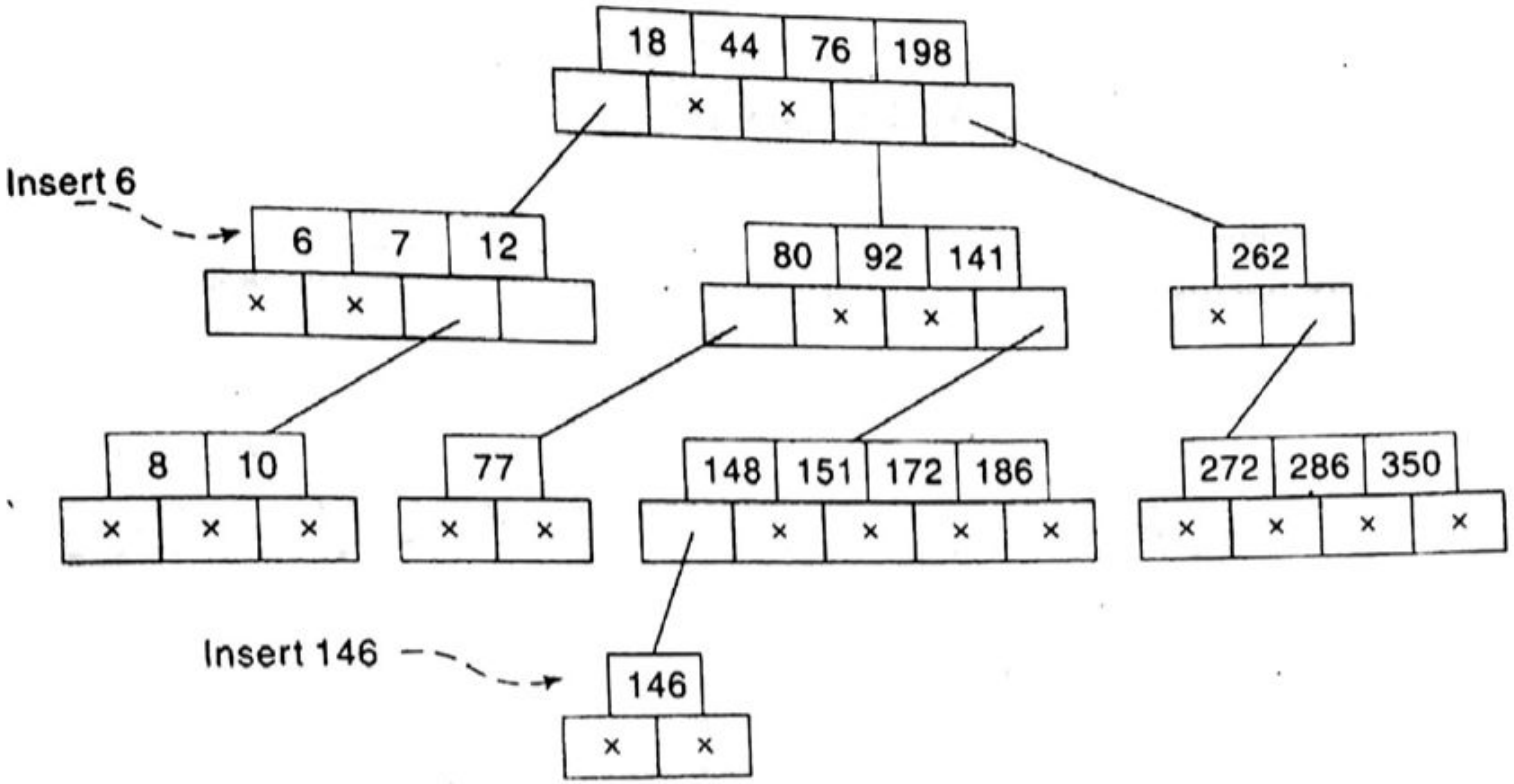
Insertion

- Insert 146



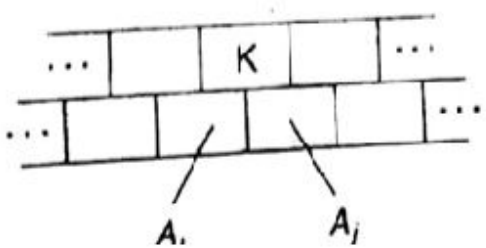
Insertion

- Insert 146



Deletion

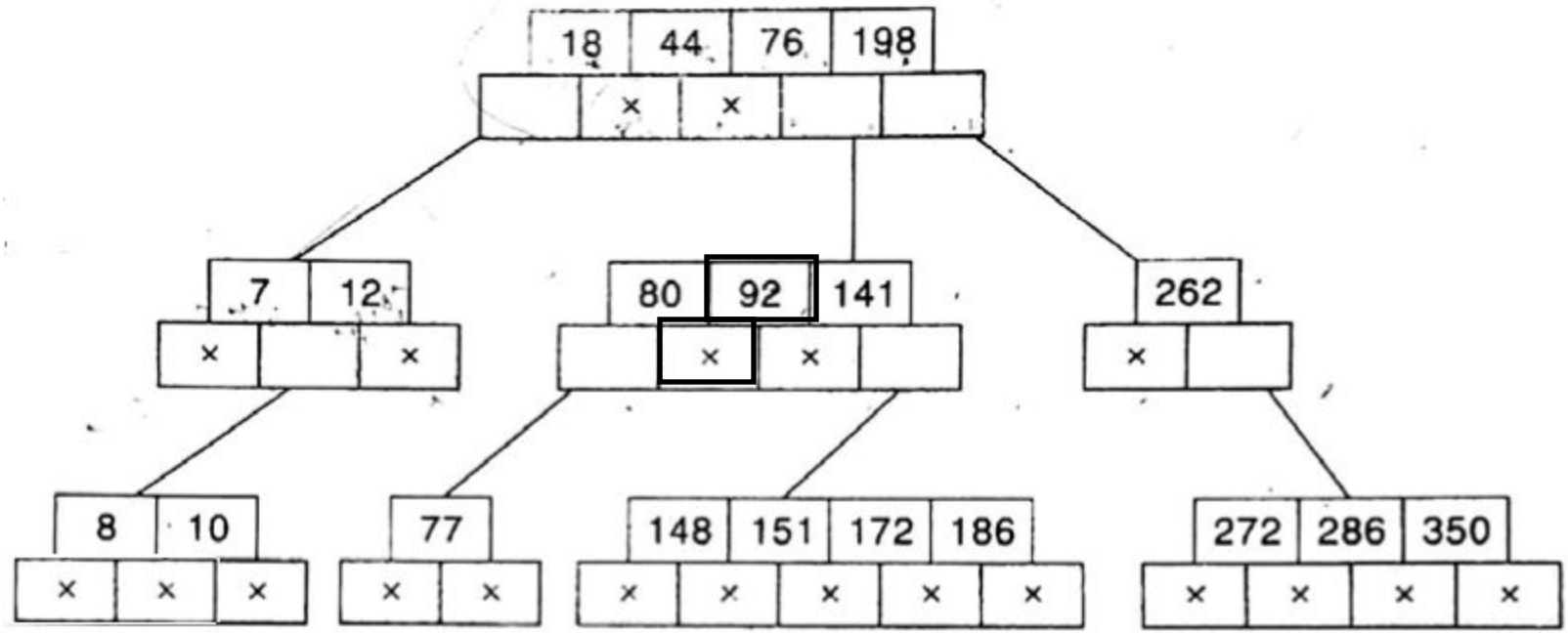
1) If $A_i = A_j = NULL$, then delete K
2) If $A_i \neq NULL$ and $A_j = NULL$, then
   - Chose the largest of the key element K' in the child node pointed to by A_i
   - Delete K'
   - Replace K by K'
3) If $A_i = NULL$ and $A_j \neq NULL$, then
   - Chose the smallest of the key element K" in the child node pointed to by A_j
   - Delete the key K"
   - Replace K by K"
4) If $A_i \neq NULL$ and $A_j \neq NULL$, then
   - Follow the rule 2 or 3



K : Key
A_i, A_j : Pointers to subtrees

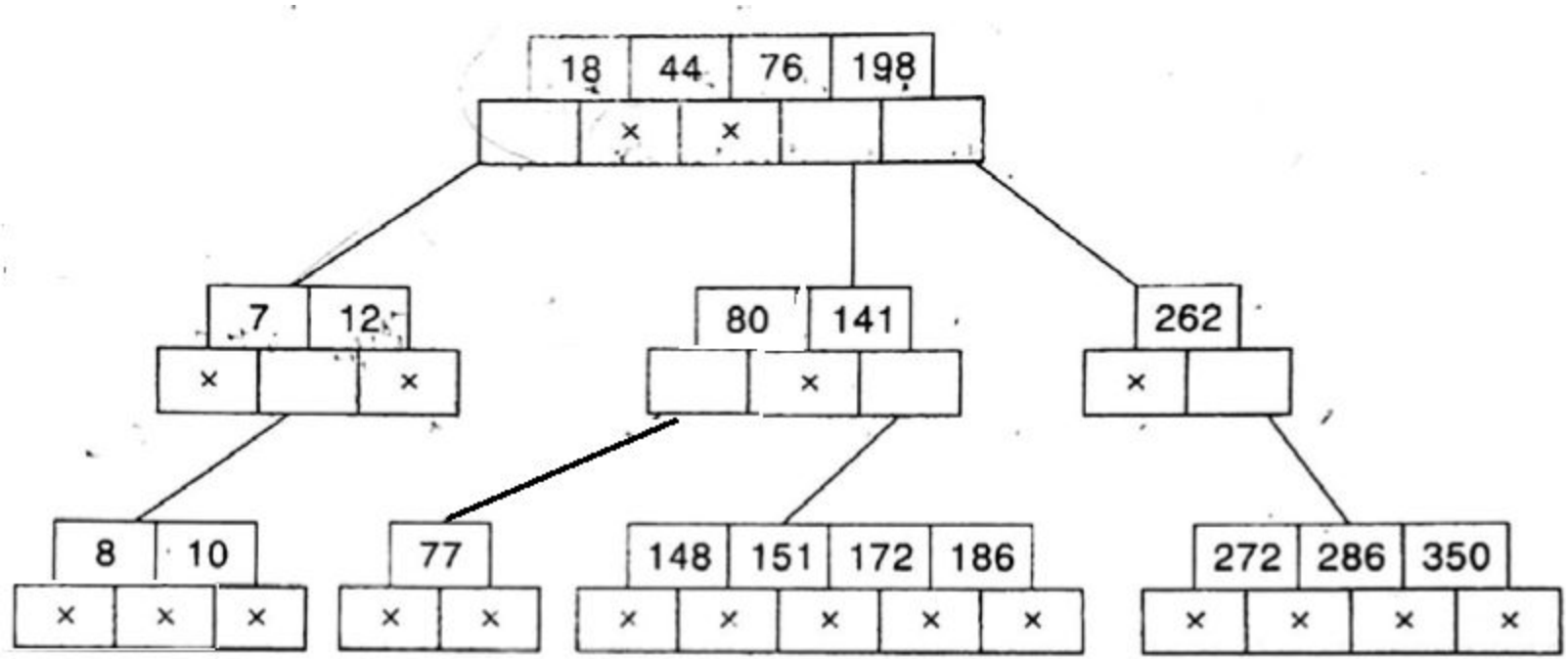
Deletion

- Delete 92



Deletion

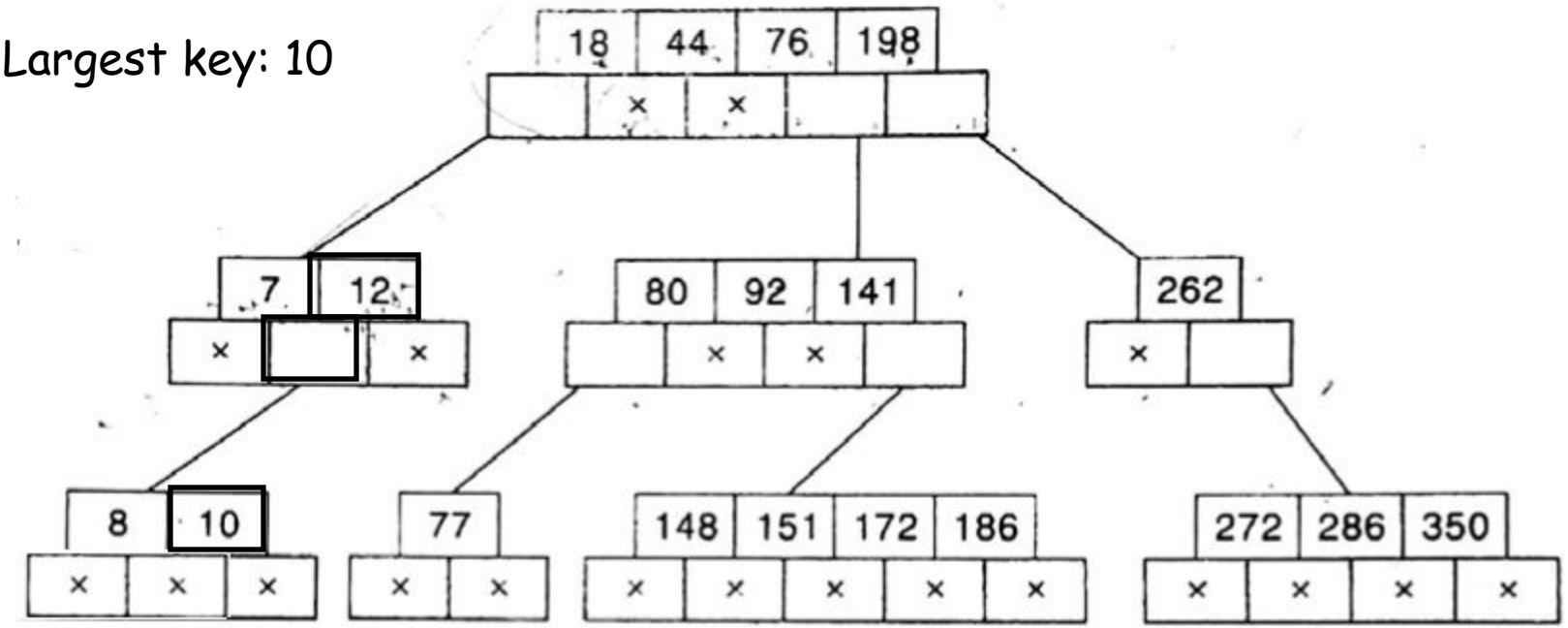
- Delete 92



Deletion

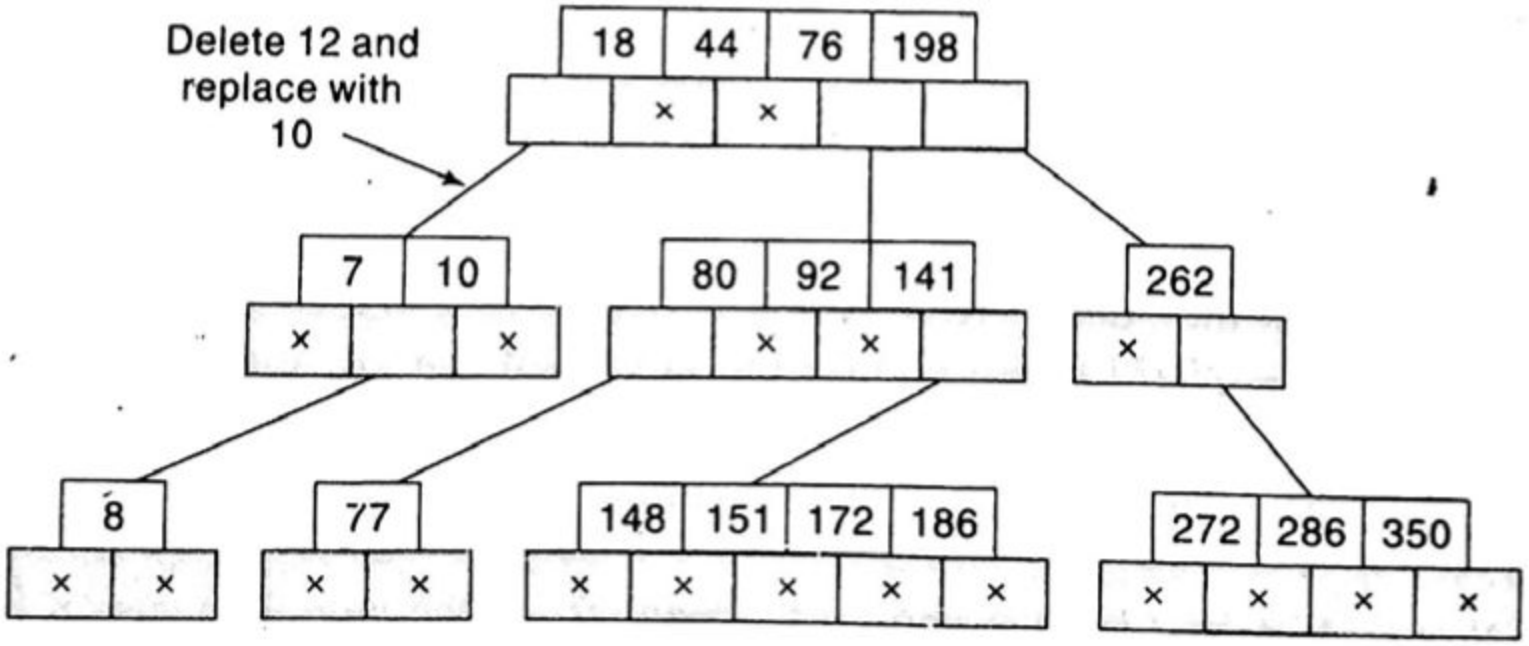
- Delete 12

- Largest key: 10



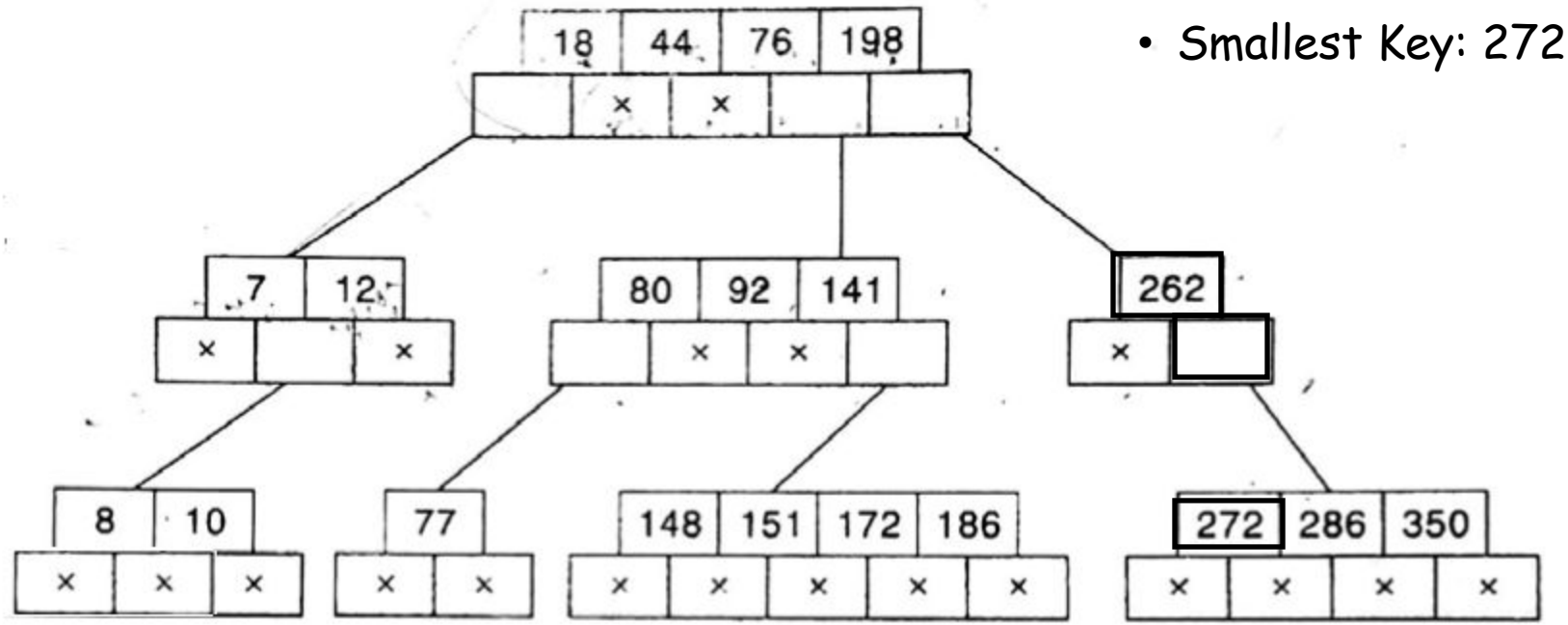
Deletion

- Delete 12



Deletion

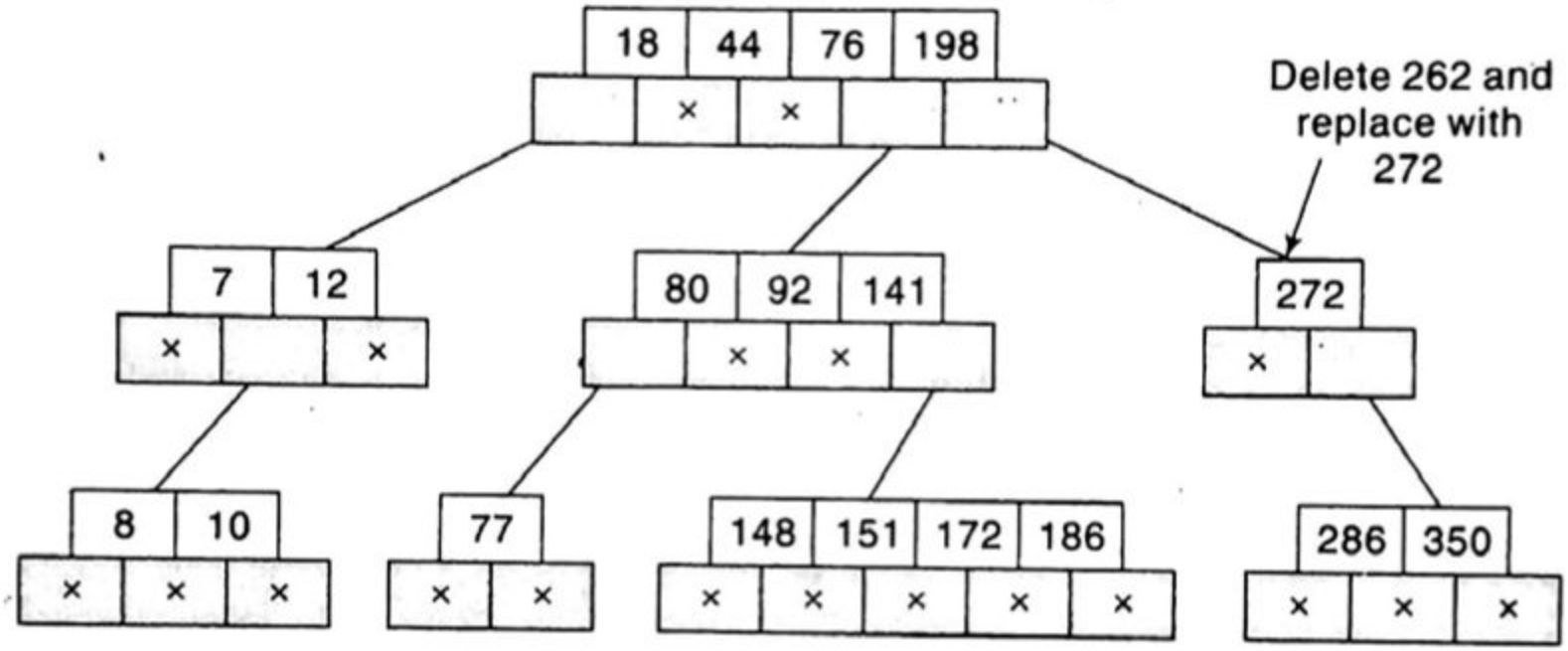
- Delete 162



- Smallest Key: 272

Deletion

- Delete 162

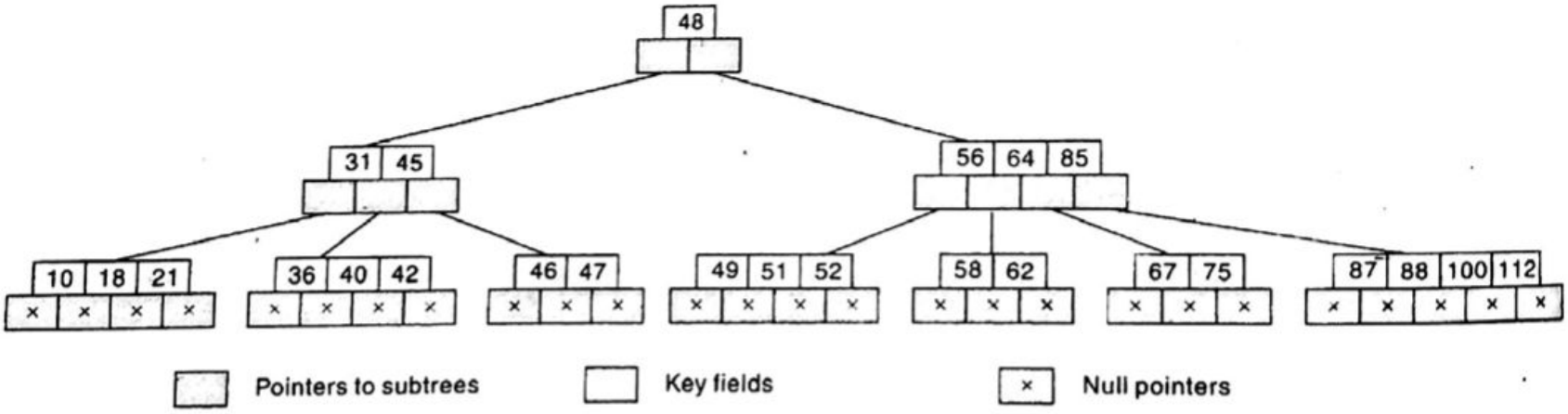


B Trees

B Trees (Definition)

- A **B-tree** of order **m**, if non empty, is an m-way search tree in which:
  i. The root has at least two child nodes and at most m child nodes
  ii. The internal nodes except root have at least $\lceil m/2 \rceil$ child nodes and at most m child nodes
  iii. The number of keys in each internal node is one less than the number of child nodes and these keys partition the keys in the subtrees of the node in a manner similar to that of m-way search trees.
  iv. All leaf nodes are on the same level.

B Trees (Definition)



- B tree with order 5

B Trees (Search)

- Same as m-way search tree.

B Trees (Insertion)

- If the leaf node in which the key is to be inserted is not full, then insertion is done in the node.
- If the node were to be full, then
  - insert the key in order into the existing set of keys in the node,
  - split the node at its median into two nodes at the same level,
  - pushing the median element up by one level.
  - Accommodate the median element in the parent node if it is not full.
  - Otherwise repeat the same procedure

B Trees (Insertion)

- Insert 4,5,58 and 6

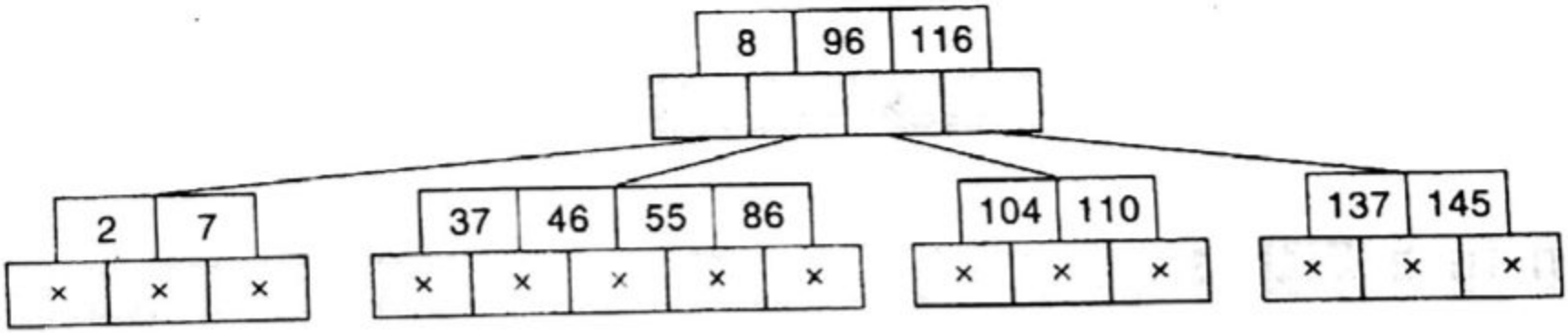
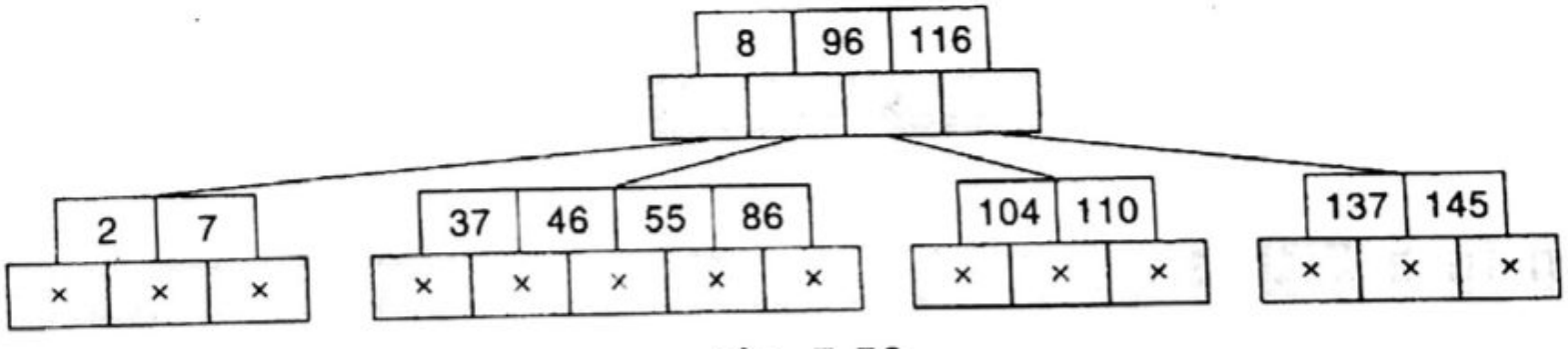


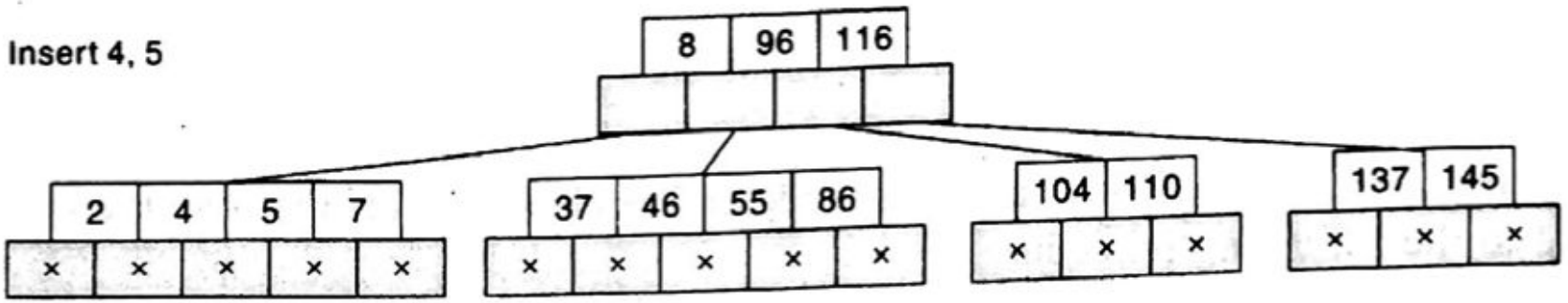
Fig. 7.53

B-tree of order 5

B Trees (Insertion)

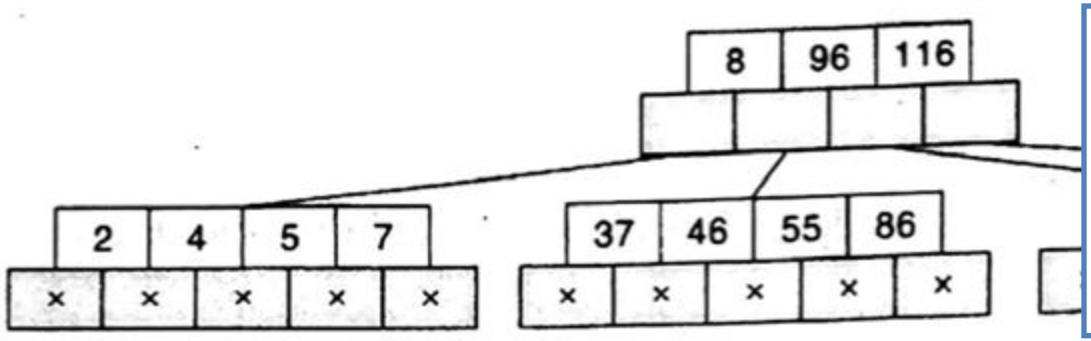


Insert 4, 5



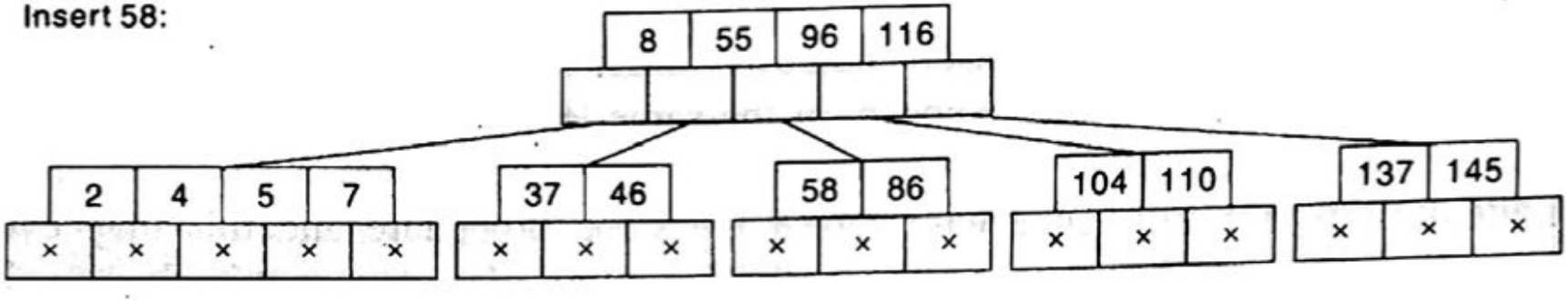
- Leaf node is not full

B Trees (Insertion)

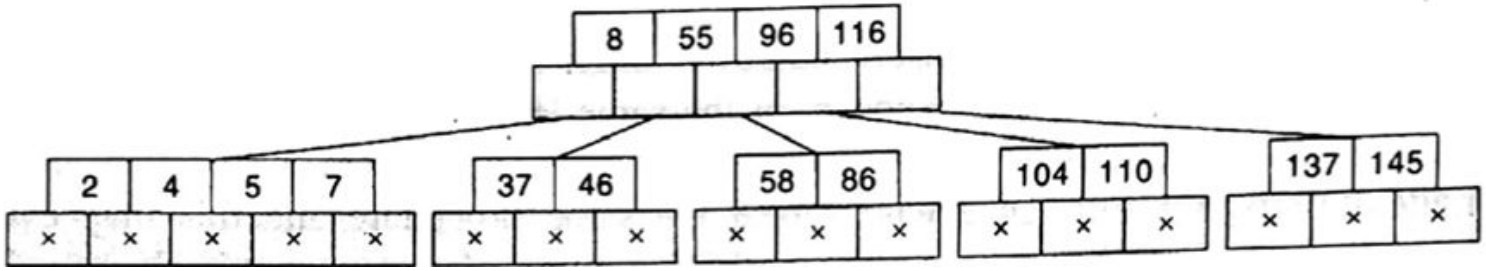


- Leaf node is full,
  - List: 37 46 55 58 86,
- Median: 55,
- 55 add with parent node since parent node is not full

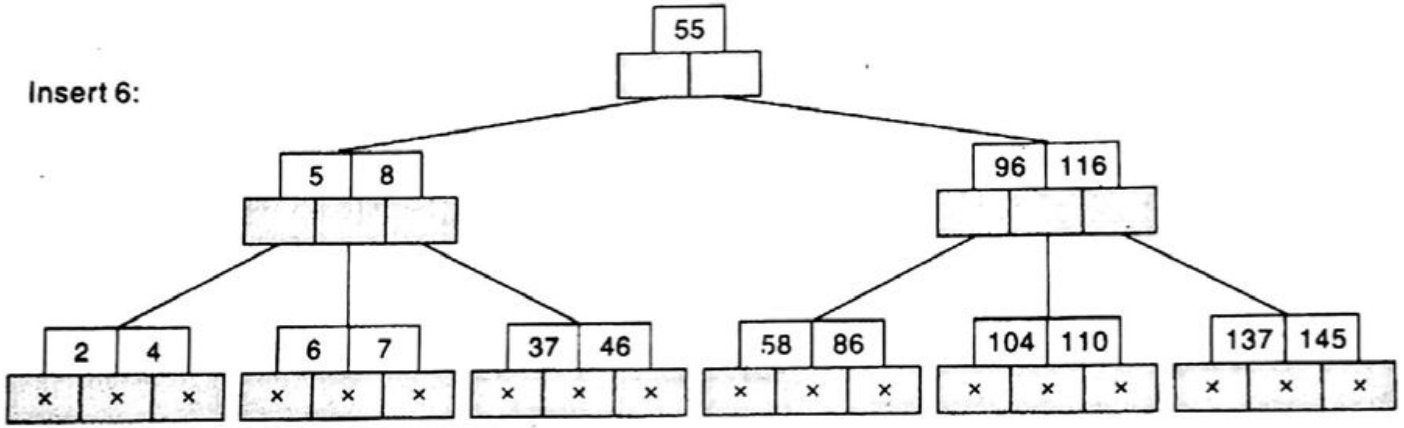
Insert 58:



B Trees (Insertion)

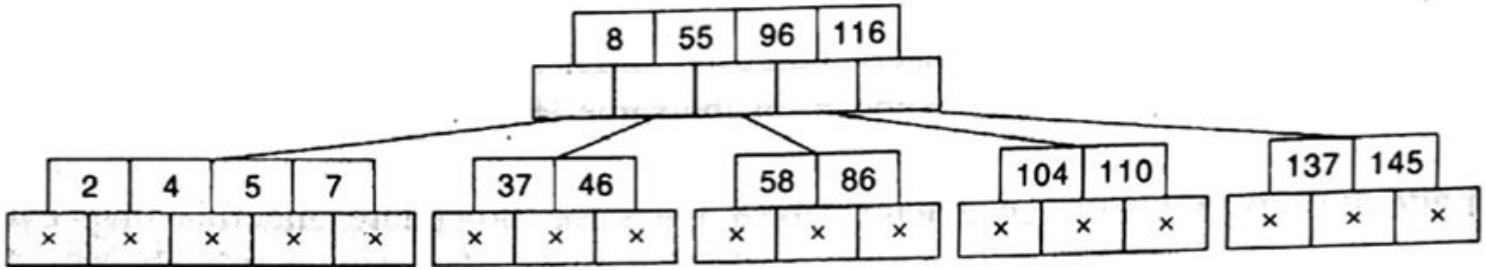


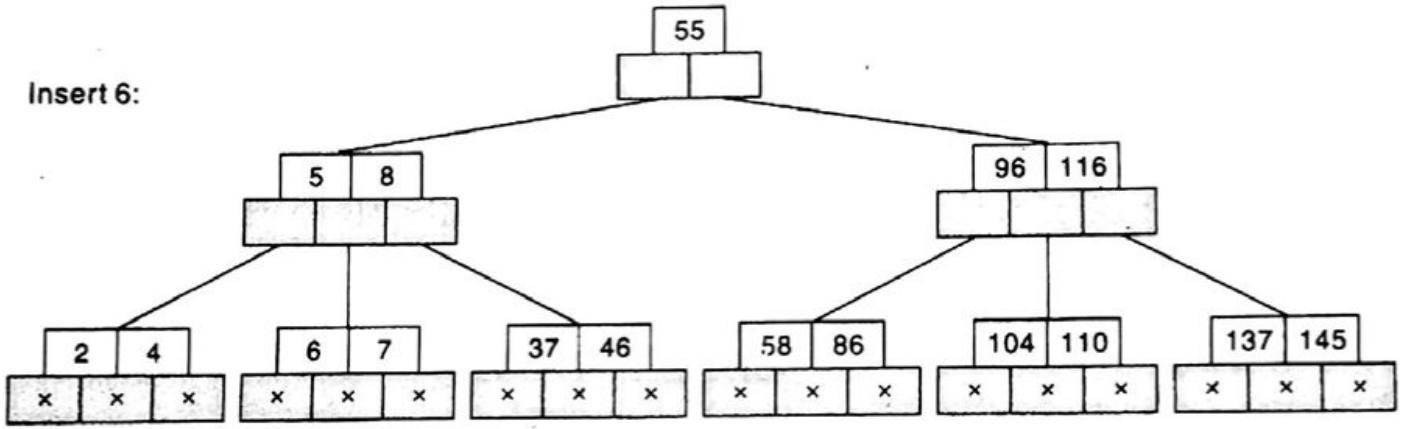
Insert 6:



- Leaf node is full, List: {2 4} 5 {6 7},
- Median: 5,
- 5 add with parent node since parent node is also full

B Trees (Insertion)





- Leaf node is full, List: {5 8} 55 {96 116},
- Median: 55,
- 5 add with parent node since parent node is not full (root)

B Trees (Deletion)

- **Objective**: the keys in the leaf node are removed
- **Case**: A key that is in an internal node to be deleted
- **Solution**:
  - we promote **a successor or a predecessor** of the key to be deleted, to occupy the position of the deleted key and such key is bound to occur in a leaf node.

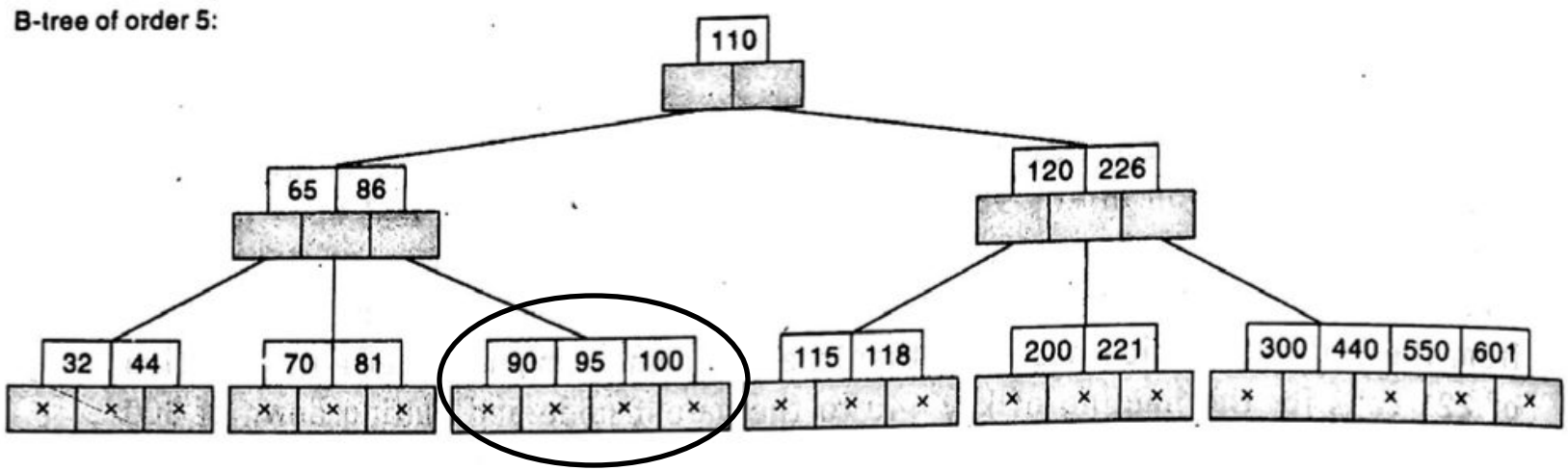
B Trees (Deletion)

- While removing a key from a leaf node, if the node contains **more than the minimum number of elements**, then the key can be **easily removed**.

- Example (Next Slide)

B Trees (Deletion)

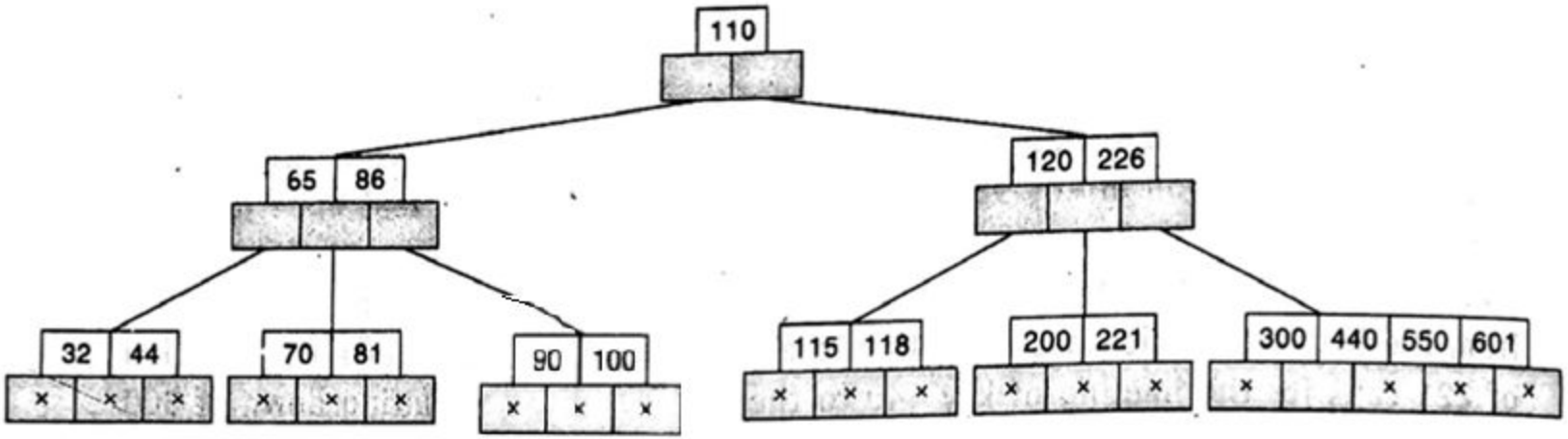
- **Delete 95**
  - Minimum number of element for B-tree of order 5 : $\lceil m/2 \rceil = 3$



- 4 elements are here

B Trees (Deletion)

- **Delete 95**
  - Minimum number of element for B-tree of order 5 : $\lceil m/2 \rceil = 3$

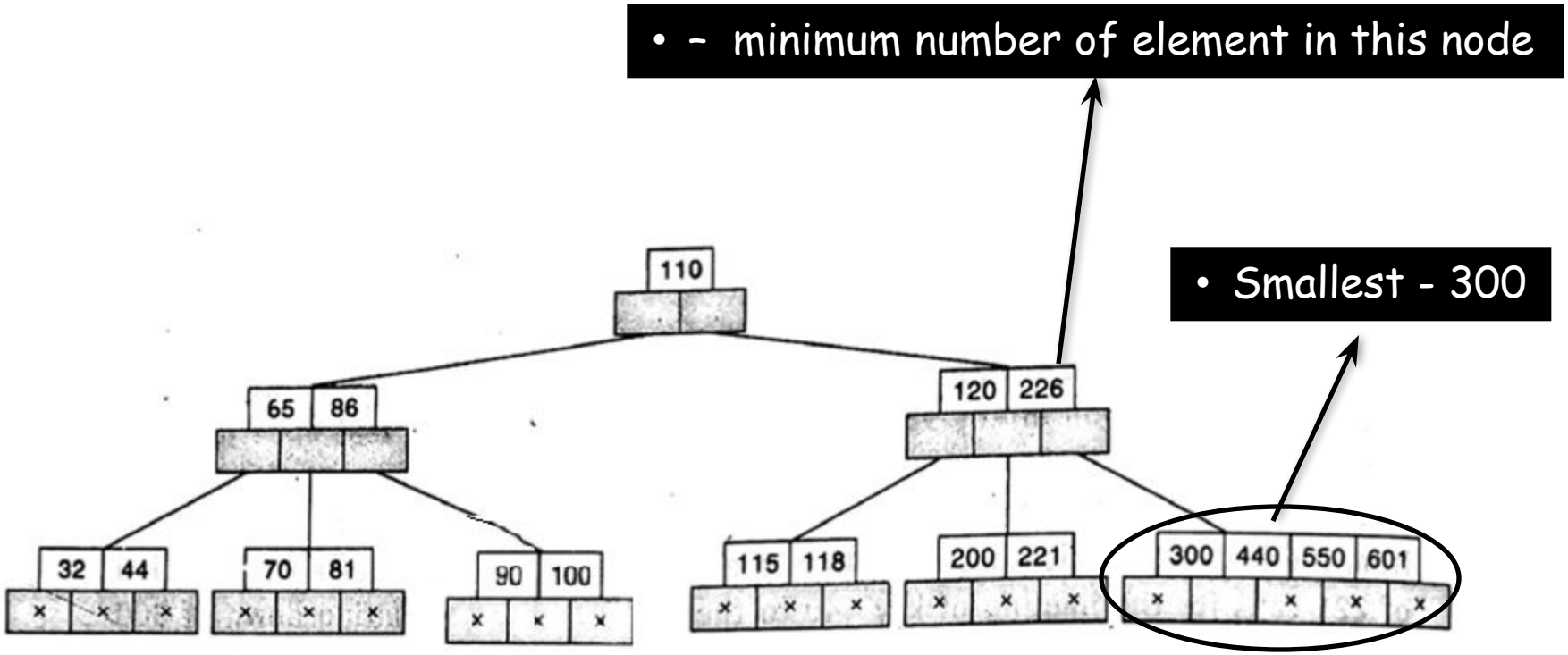


B Trees (Deletion)

- If the leaf node contains just the minimum number of elements, then **search for an element** from either the left sibling node or right sibling node to fill the vacancy.
  - If the left sibling node has more than the minimum number of keys, pull the largest key up into the parent node and move down the **intervening entry** from the parent node to the leaf node where the key is to be deleted.
  - Otherwise, pull the smallest key of the right sibling node to the parent node and move down the intervening parent element to the leaf node.

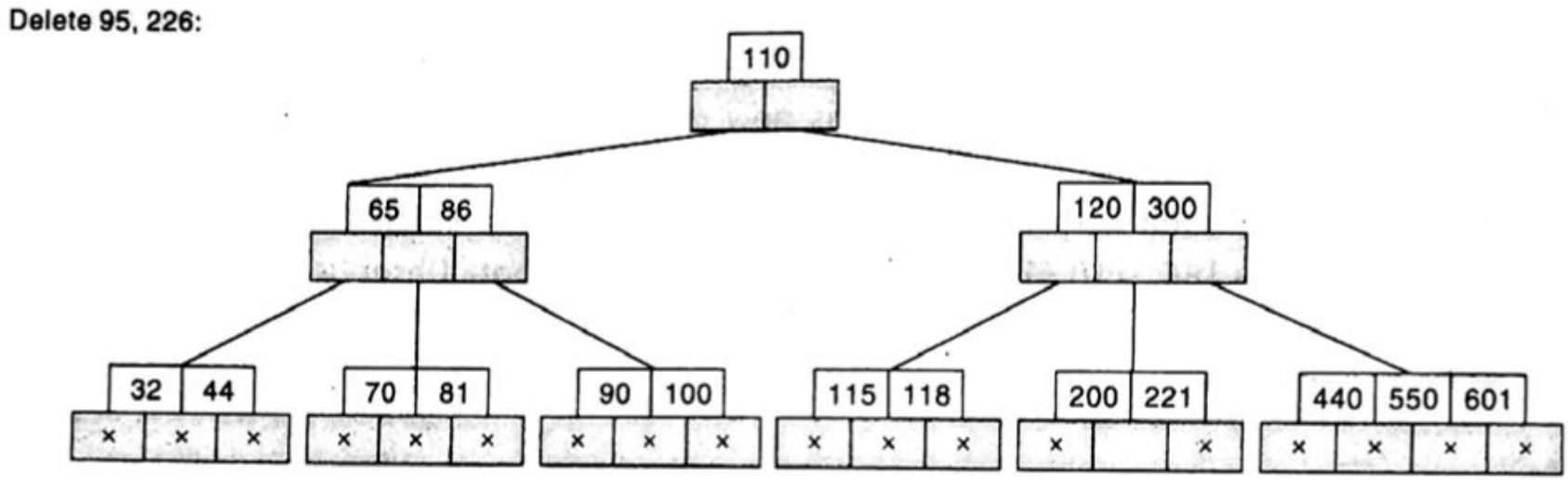
B Trees (Deletion)

- Delete 226



B Trees (Deletion)

- Delete 226



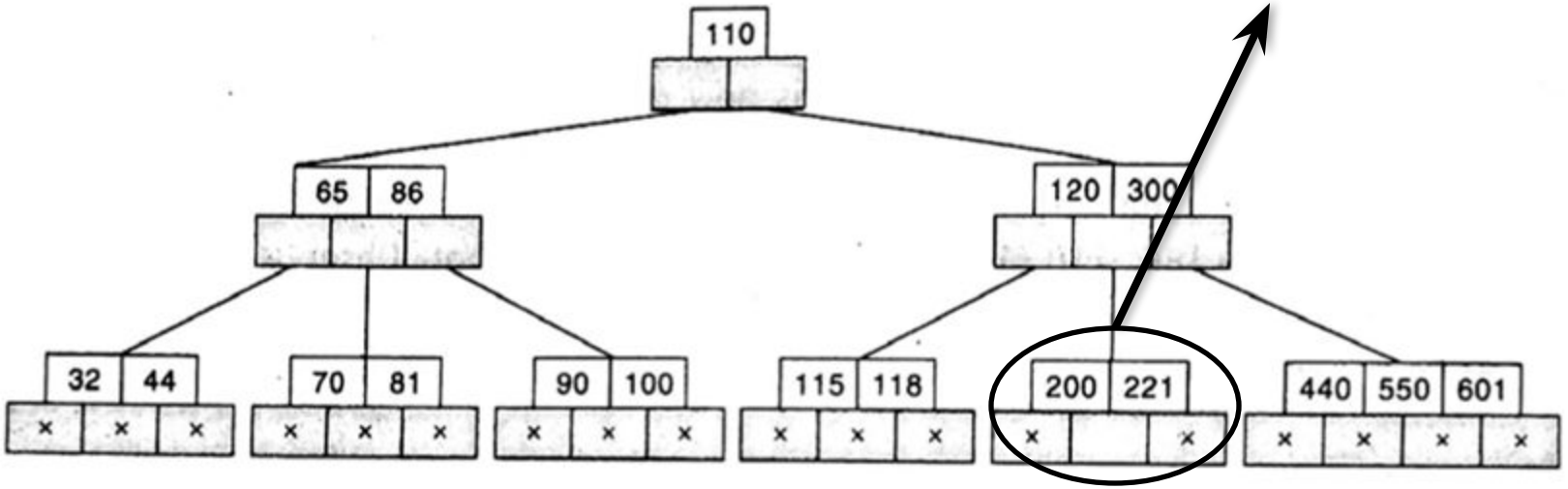
B Trees (Deletion)

- If both the sibling nodes have only minimum number of entries, then create a new leaf node out of the two leaf node and **intervening element of the parent node**, ensuring that the total number does not exceed the maximum limit for a node.
  - Remember: If while borrowing the intervening element from the parent node, it leaves the number of keys in the parent node to be below the minimum number, then we propagate the process upwards ultimately resulting in a reduction of height of the B-tree.

B Trees (Deletion)

- Delete 221

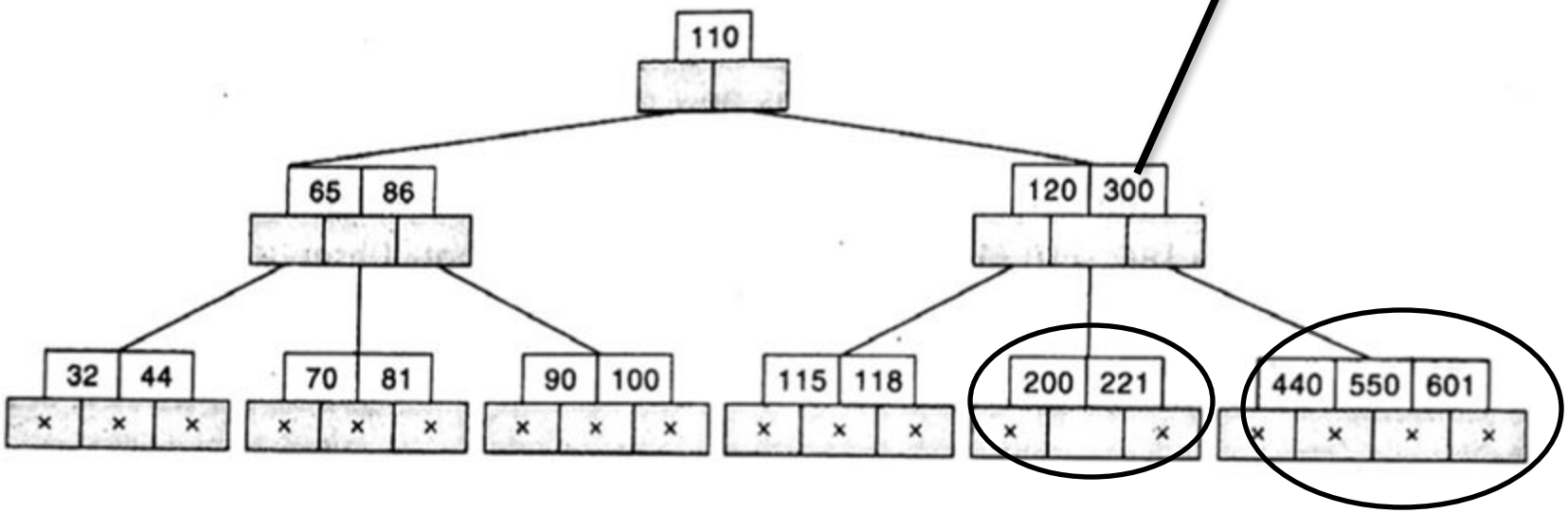
- both the sibling nodes have only minimum number of entries



B Trees (Deletion)

- Delete 221

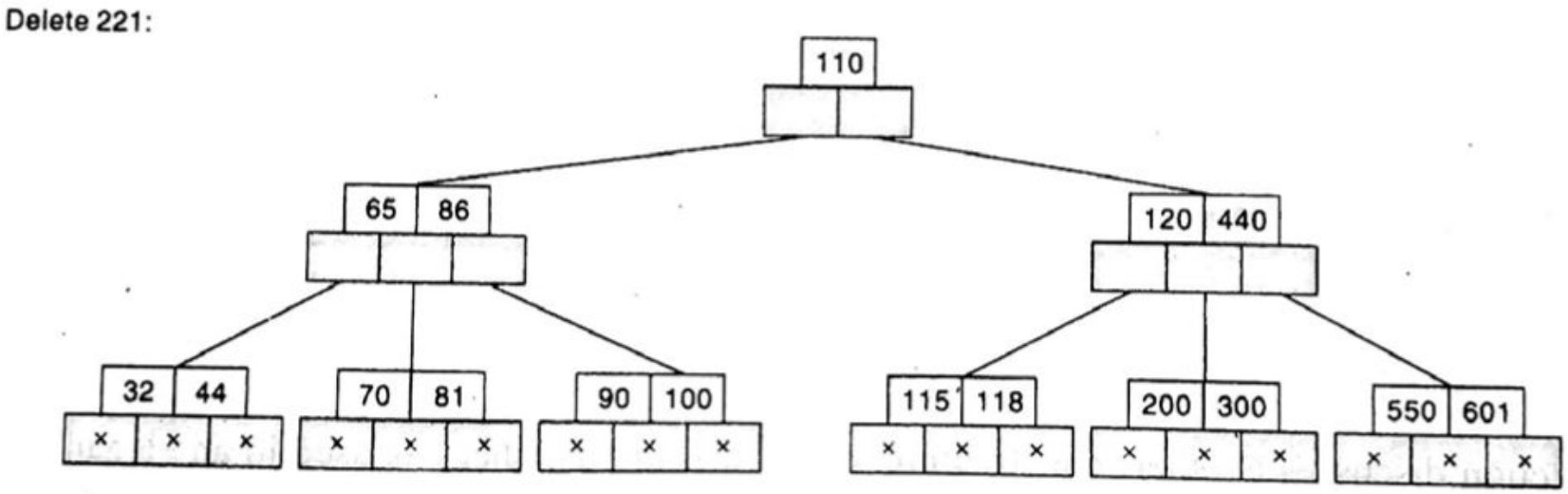
- Intervening element of parent



B Trees (Deletion)

- Delete 221

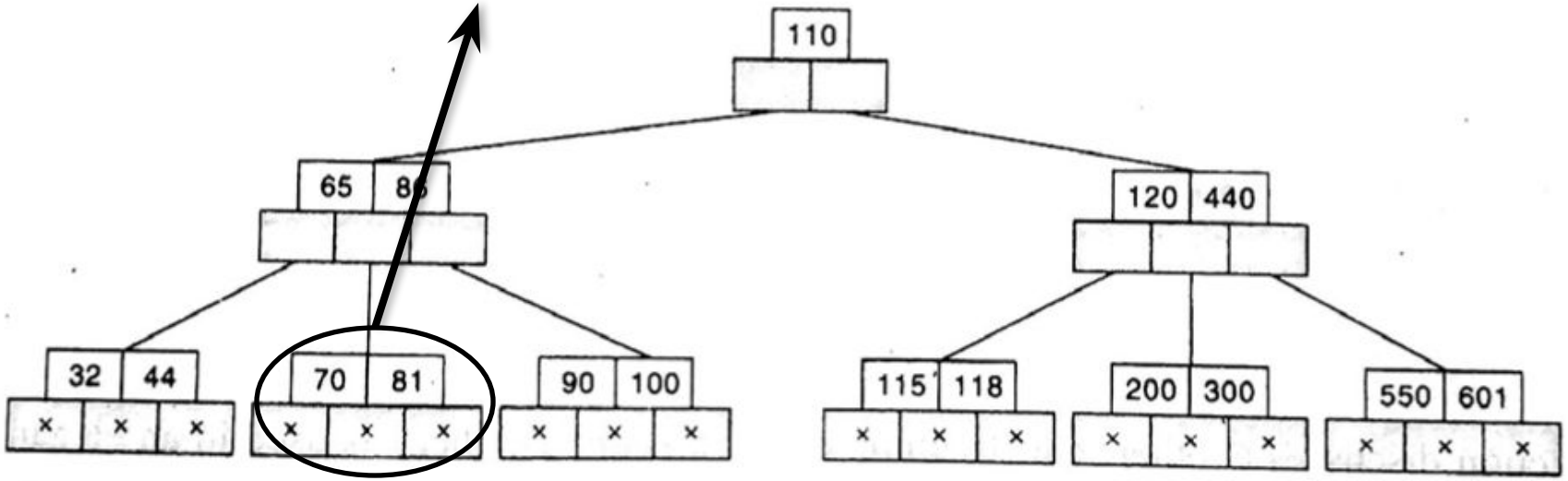
- Merge: (200 300) **440** (550 601)



B Trees (Deletion)

- Delete 70

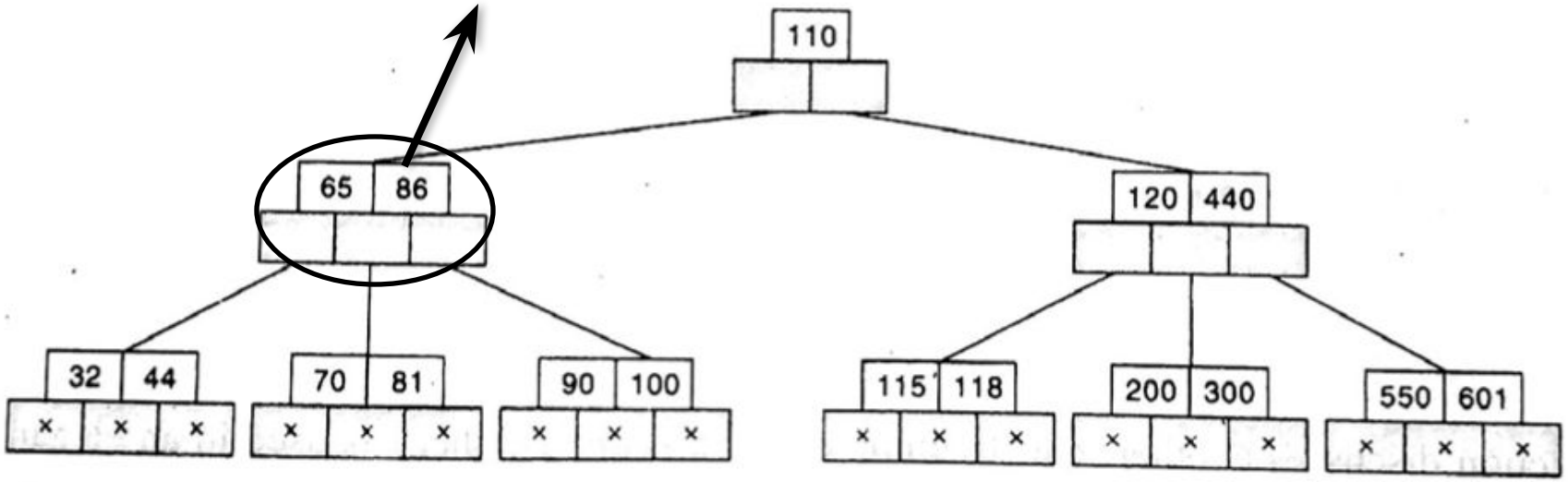
- both the sibling nodes have only minimum number of entries



B Trees (Deletion)

- Delete 70

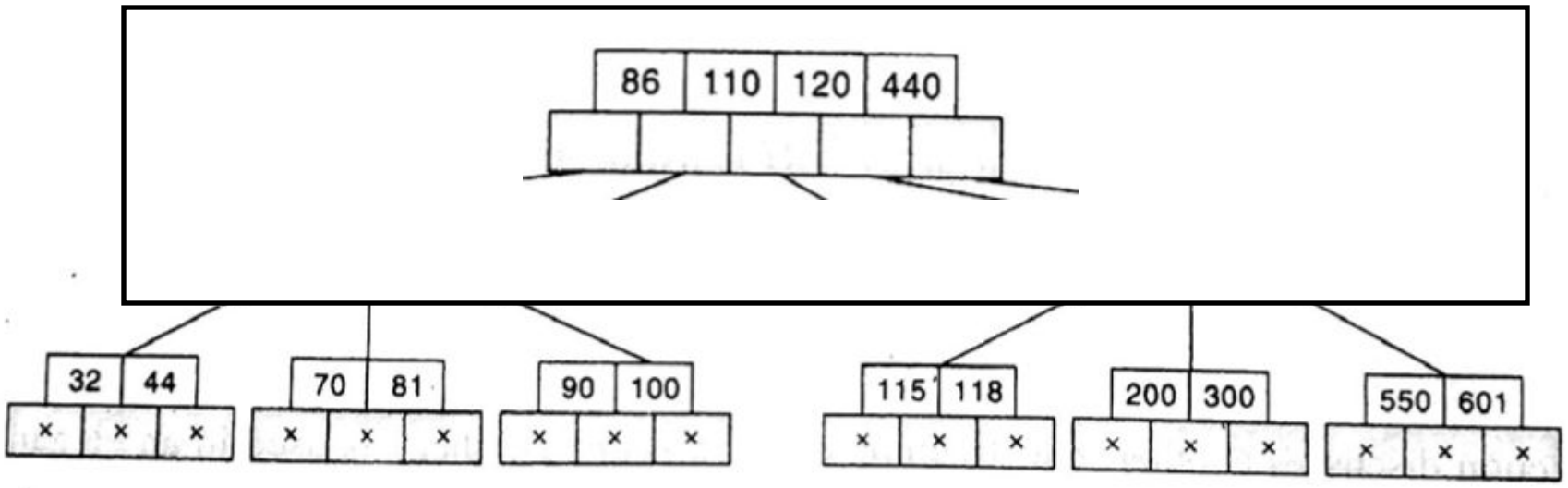
- Parent nodes have only minimum number of entries



B Trees (Deletion)

- Delete 70

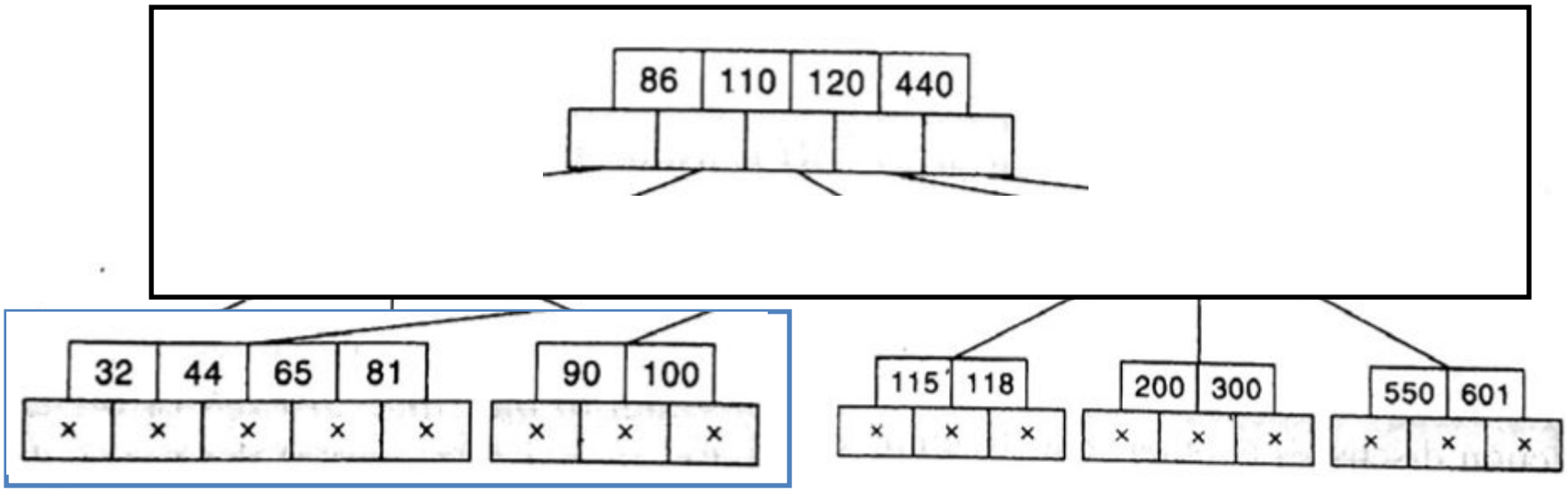
  - So consider root nodes and two sibling with respect ot root
  - Merge: 65 86 110 120 440
  - Borrow 65
  - See the below figure



B Trees (Deletion)

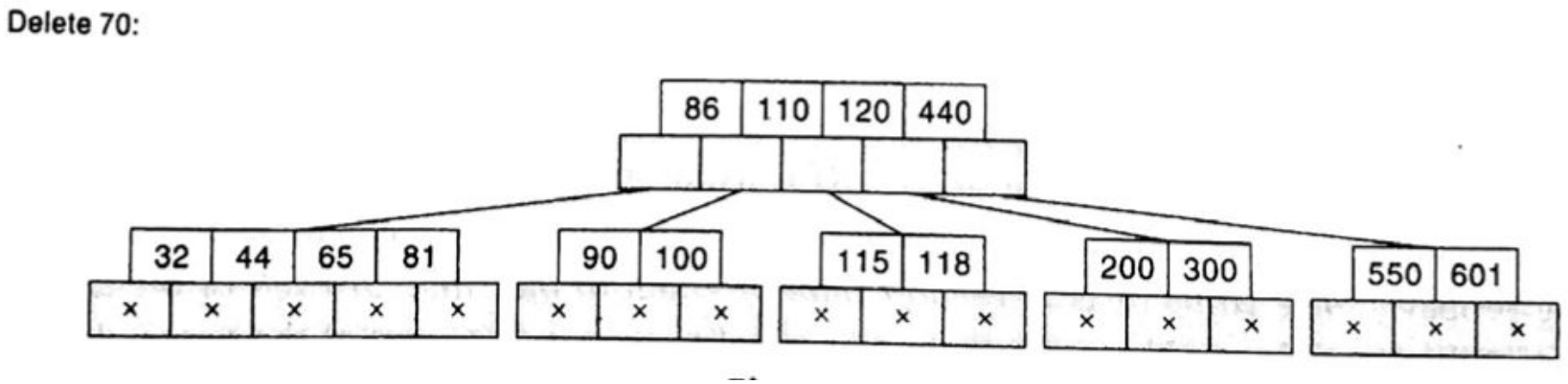
- Delete 70

  - Merge Leaf nodes: (32 44 65 (borrowing) 81) (90 100)



B Trees (Deletion)

- Delete 70



Any Query?