TECHNO INDIA UNIVERSITY
WESTBENGAL

Data Structures & Algorithms
(PCC-CS 301)

Dr. Debashis Das
Associate Professor
Department of CSE
Techno India University, Kolkata

Topics Covered

1. m-way search tree
2. Balanced m-way search tree
   2.1. B tree
   2.2. B+ tree

m-way search Tree

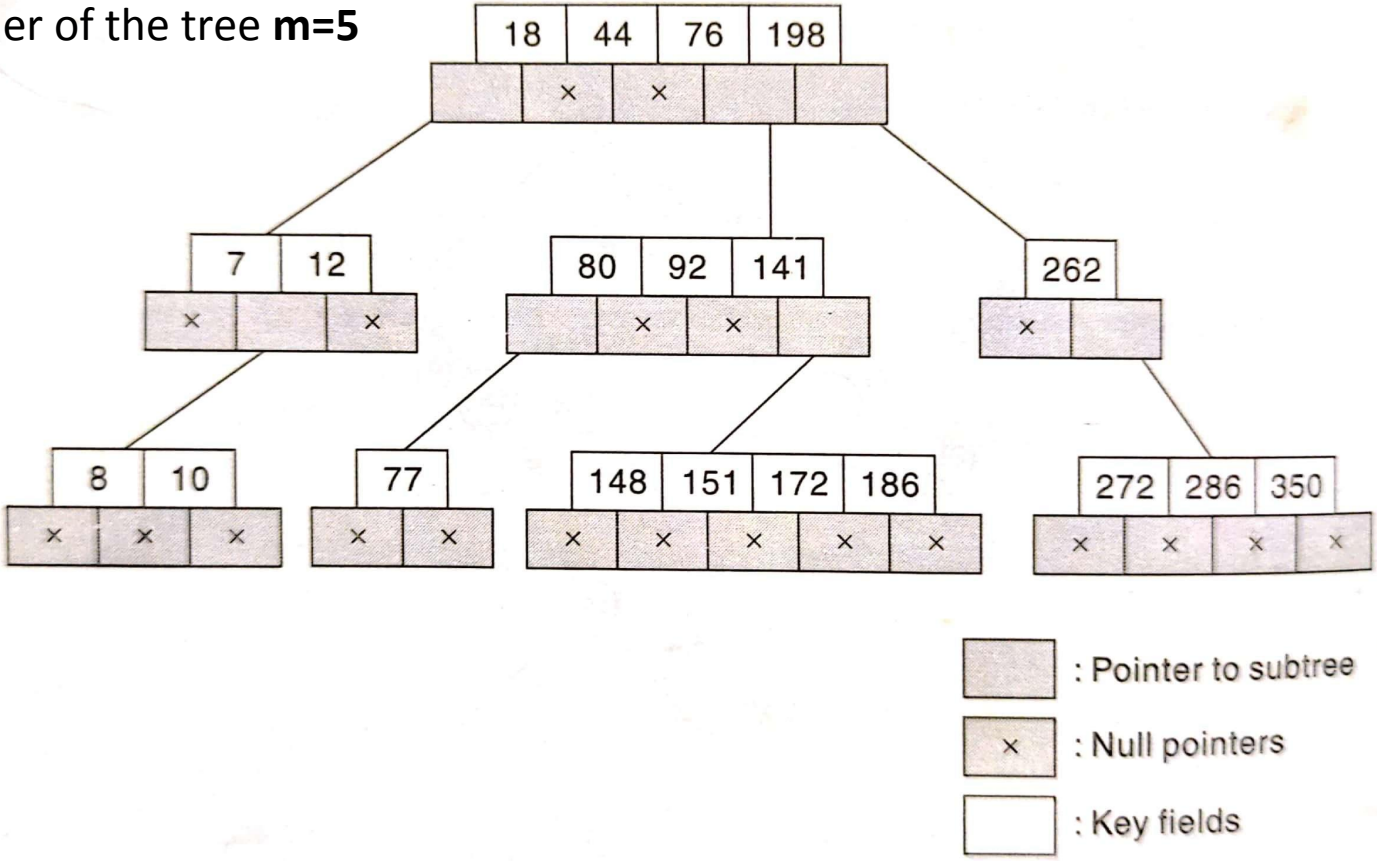
- Introduction
  - It is the generalized version of Binary Search Tree
  - For some integer ***m***, which is order of the tree, each node may have maximum ***m*** child nodes
  - A node may be represented as $<A_0,(K_1,A_1),\ldots(K_{m-1},A_{m-1})>$ where
    - K_i, $1 \le i \le m-1$ are keys (values)
    - A_i, $0 \le i \le m-1$ are the pointers to sub-trees

  - For any node the key values $k_i \le k_{i+1}$
    - The sub-tree nodes hold by A_i are lesser than k_i
    - The sub-tree nodes hold by A_{i+1} are greater than k_i
  - Height of m-way search tree is $\log_m(n)$ if total node is n

m-way search Tree

- Example

Order of the tree **m=5**



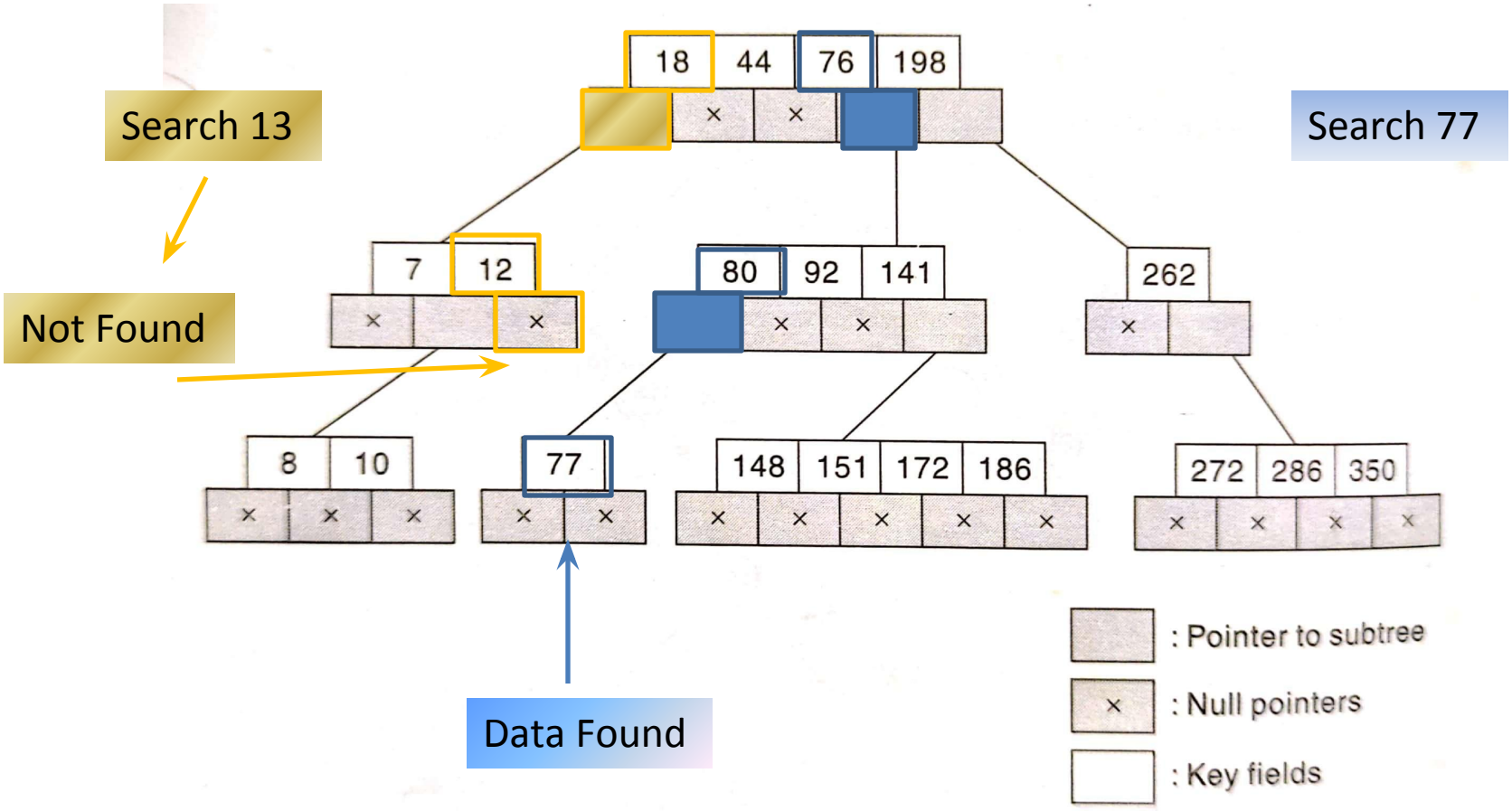
m-way Search Tree

- Data Searching

  - Starts searching from the root node
  - In each node, searches from the first key if data found return
  - If searching data is lesser than any key, go to the left pointer of that key
  - If searching data is larger than the key, go to the right sub-tree of that key
  - Searching time is $O(\log_m n)$, m is the order of tree and n is the total number of nodes

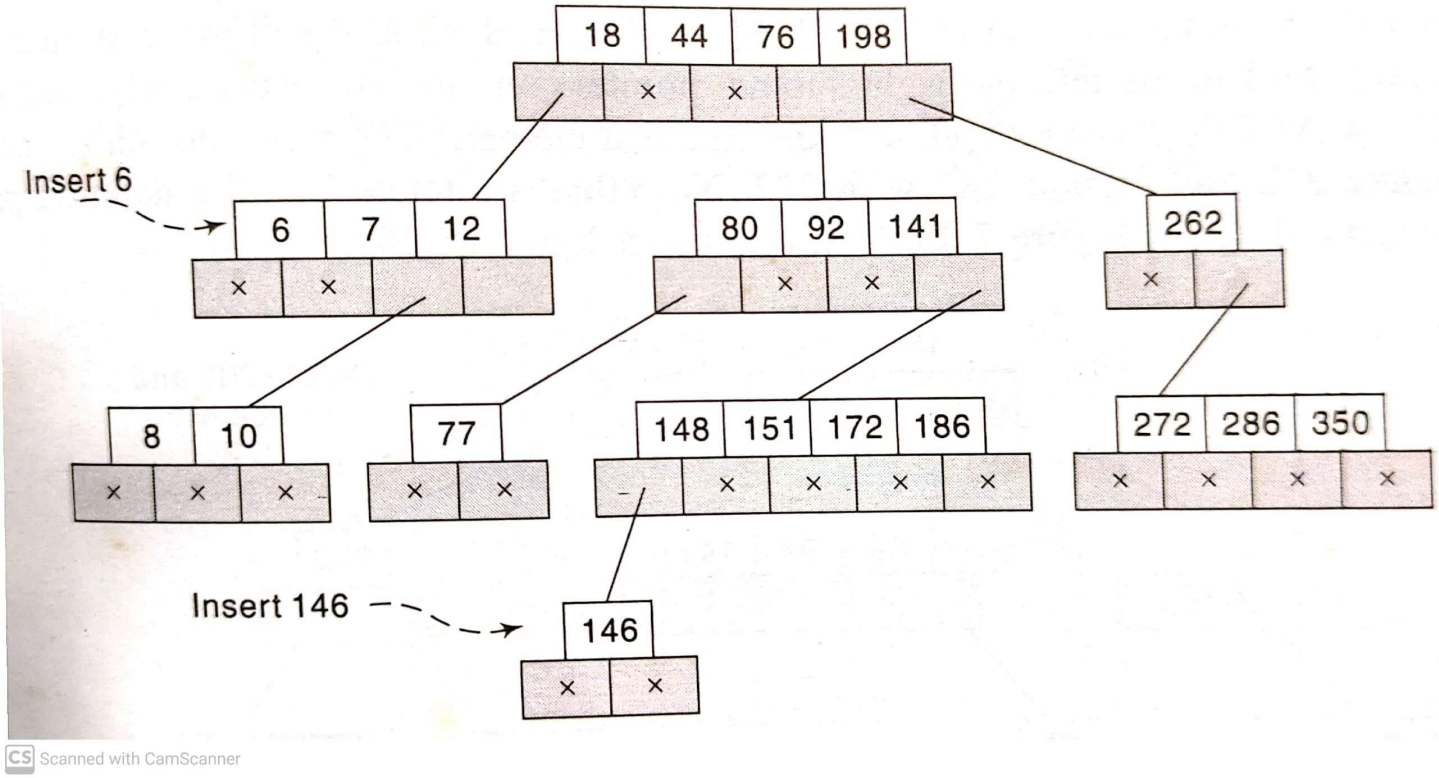
m-way Search Tree

- Data Searching (example)



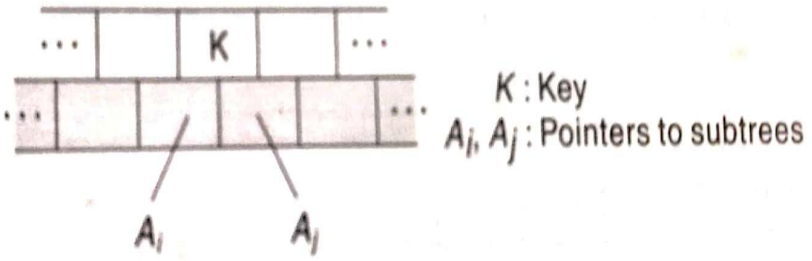
m-way Search Tree

- Data Insertion
  - Apply searching technique to find the place of insertion



m-way Search Tree

- Data deletion



- Let key **K** to be deleted
- May arise 4 conditions
  - If $A_i = A_j = NULL$ then delete **K**
  - If $A_i \neq NULL$ but $A_j = NULL$

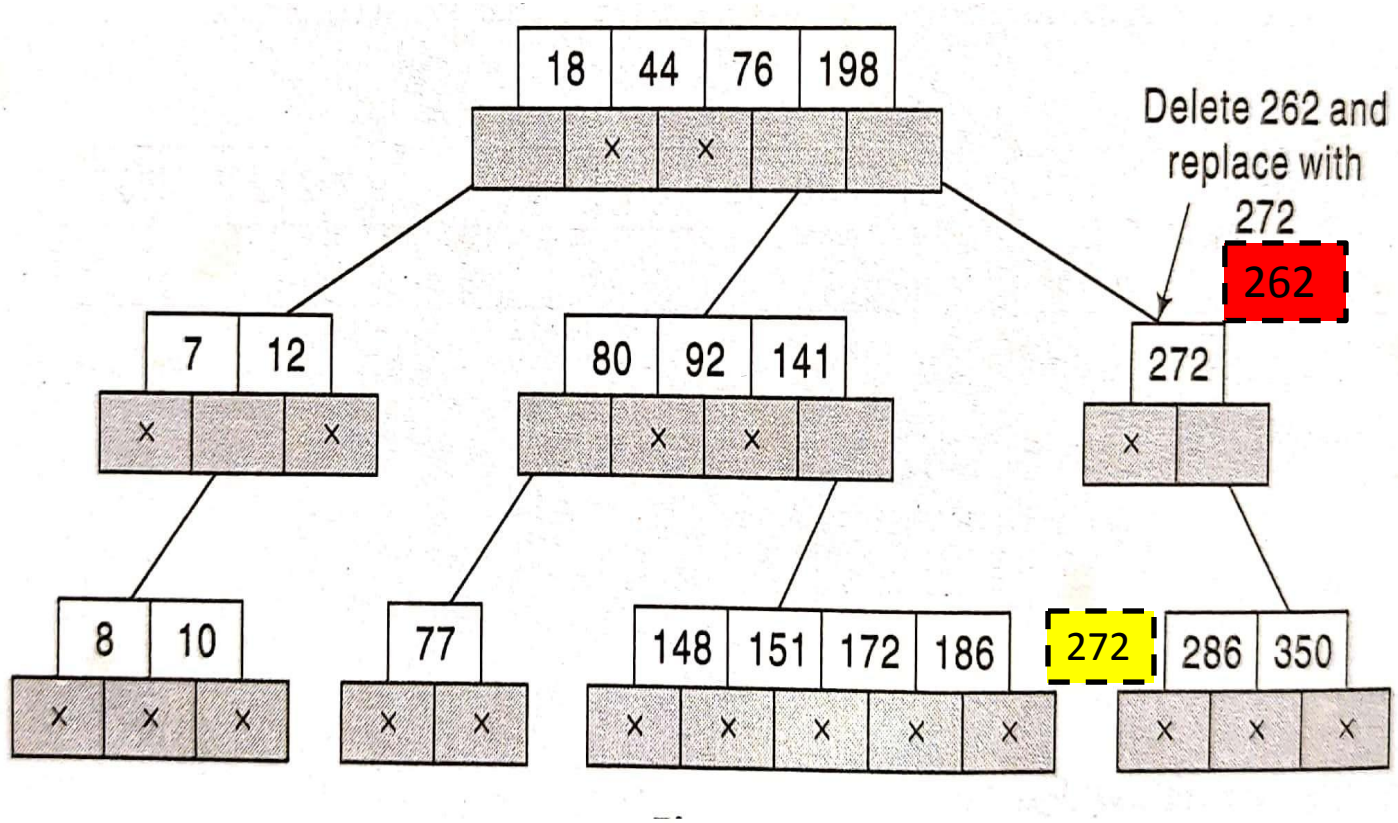
    *choose the **largest** key **K'** from the sub-tree pointed by A_i and replace with **K***
  - If $A_i = NULL$ but $A_j \neq NULL$

    *choose the **smallest** key **K'** from the sub-tree pointed by A_j and replace with **K***
  - If $A_i \neq NULL$, $A_j \neq NULL$

    *choose either the **smallest** key from the sub-tree pointed by A_j or the **largest** key from the sub-tree pointed by A_i and replace with **K***

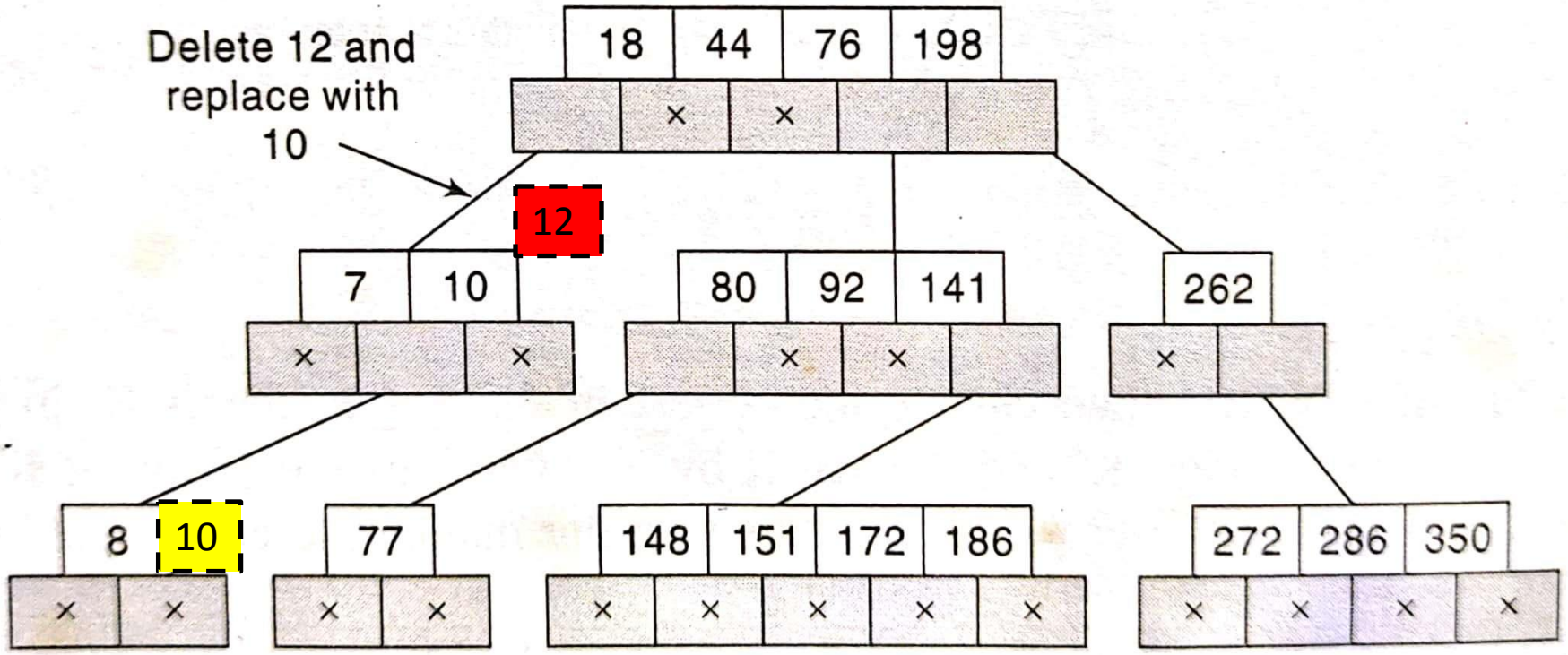
m-way Search Tree

- Data deletion (example-1)



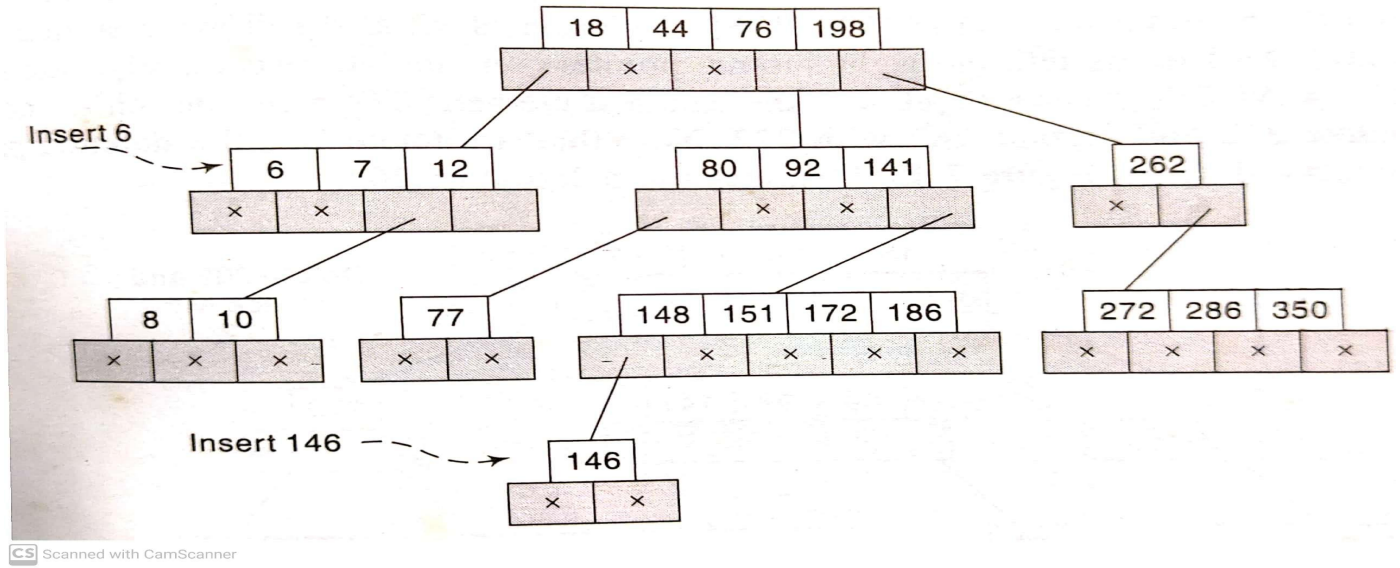
m-way Search Tree

- Data deletion (example-2)



Balanced m-way Search Tree

- Motivation



Problem

Due to the insertion of 146, tree height increased by 1 although some internal node are not full

solution

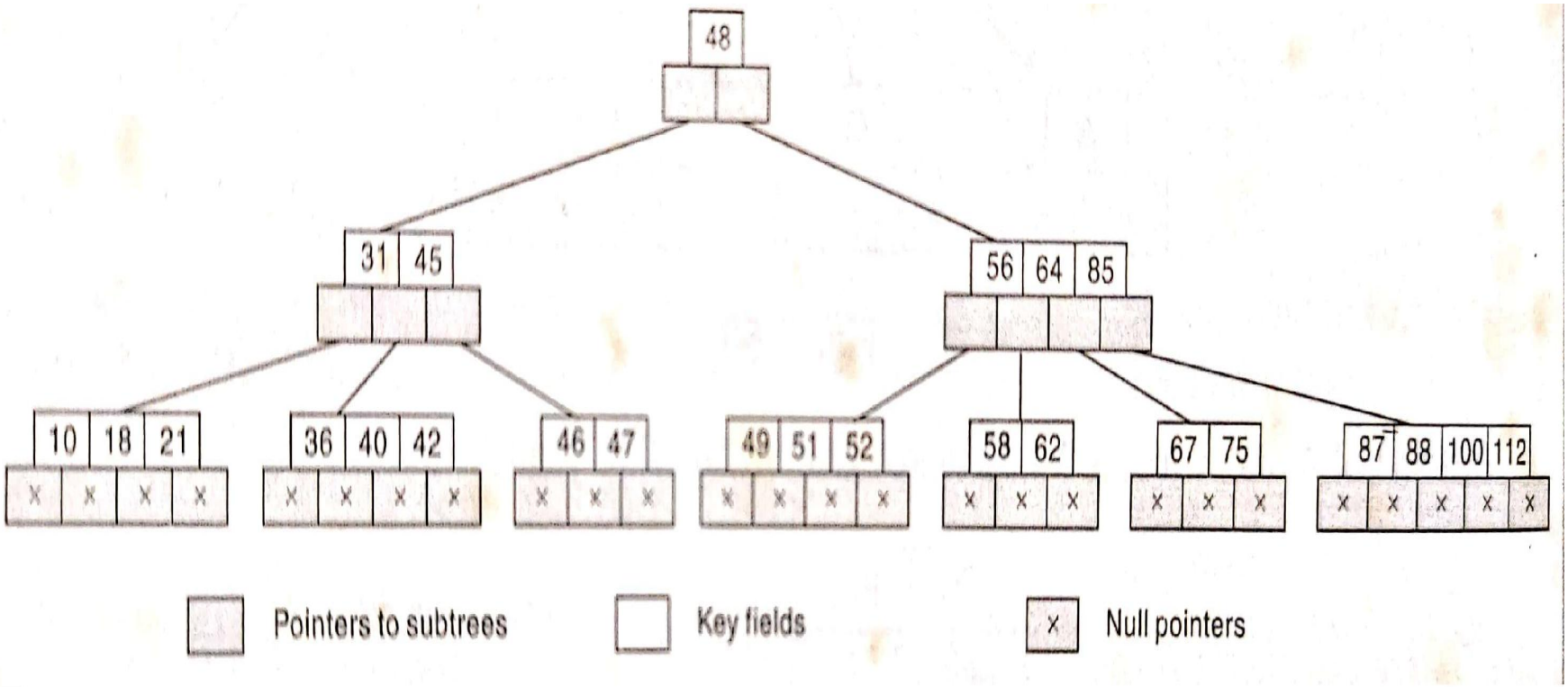
- B tree
- B+ tree

B Tree

- Introduction
  - B-tree of order **m** is an m-way search tree in which
    - The root has at least **2** child nodes and at most **m** child nodes
    - The internal nodes except the root have at least $\left\lceil \frac{m}{2} \right\rceil$ child nodes and at most **m** child nodes
    - The number of keys in each internal node is **one less** than the number of child nodes
    - All leaf nodes are on the same level
  - B-tree of order 3 is referred to as 2-3 tree since the internal nodes are of degree 2 or 3 only

B Tree

- Example



B Tree

- Data Searching

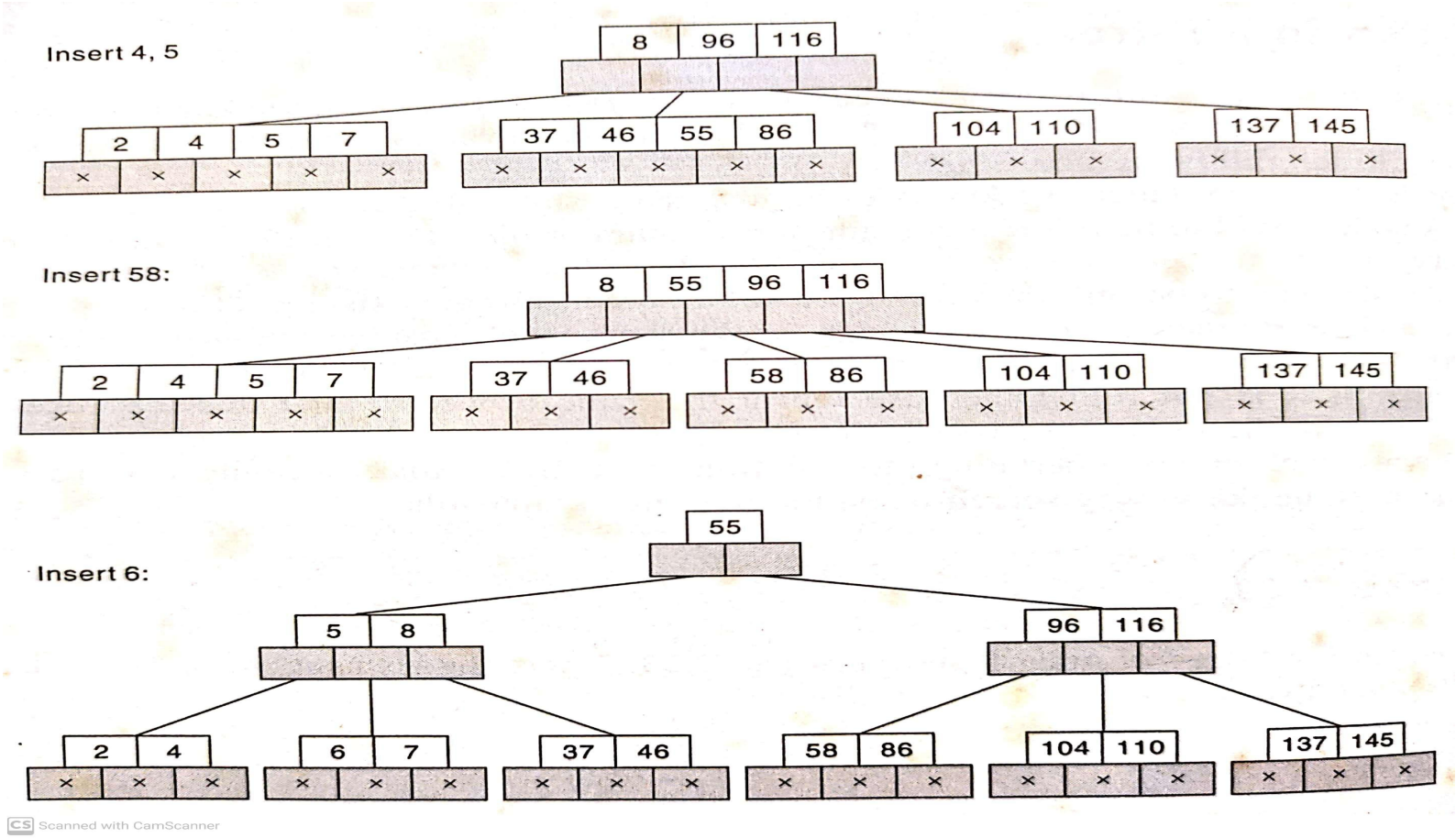
  - Searching of a key in a B-tree is similar to the m-way search tree
  - The number of access depends on the height of the B-tree

B Tree

- Data Insertion
  - Search the proper place for the key to be inserted (in the leaf)
  - If the leaf node, where the key to be inserted, is not full then insert the data in that node
  - If the node will be full after inserting the new key, insert the key and split the node at it's median into 2 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. This may rearrange the keys in the root node or the formation of a new root itself
  - This is a bottom-up approach

B Tree

- Data Insertion (example)



B Tree

- Data Deletion
  - While deleting a key, it is desirable that the keys in the leaf node are removed
  - If an internal node to be deleted, promote a successor or predecessor of the key to be deleted to occupy the position of the actual deleted key
  - While removing a key from a leaf node
    - If the node contains more than the minimum elements, key is deleted
    - If the leaf node contains just the minimum number of elements, scout for an element from either left or right sibling
    - If the left sibling has more than minimum elements, pull the largest key up to the parent and move down the intervening element to the leaf node

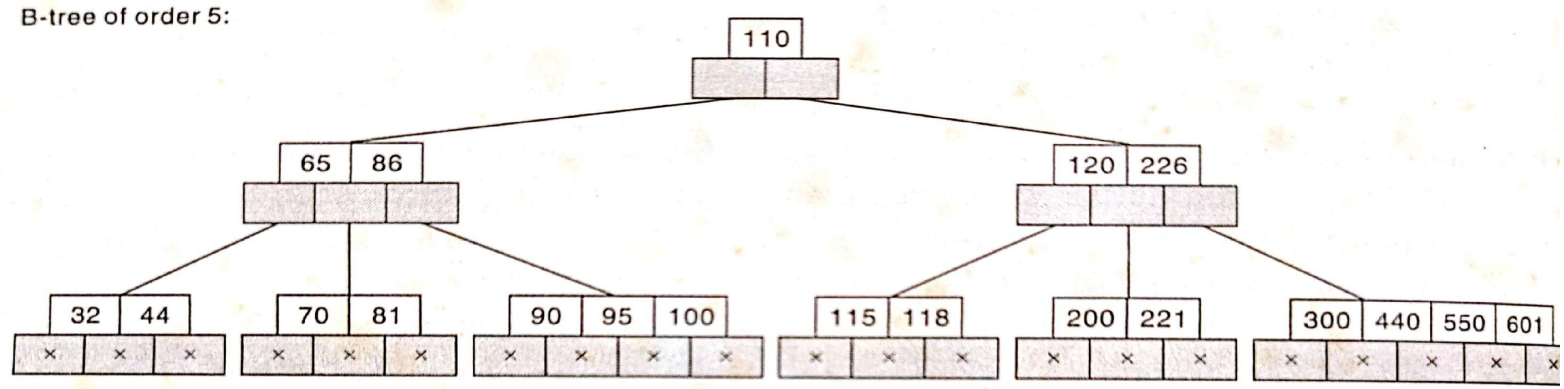
B Tree

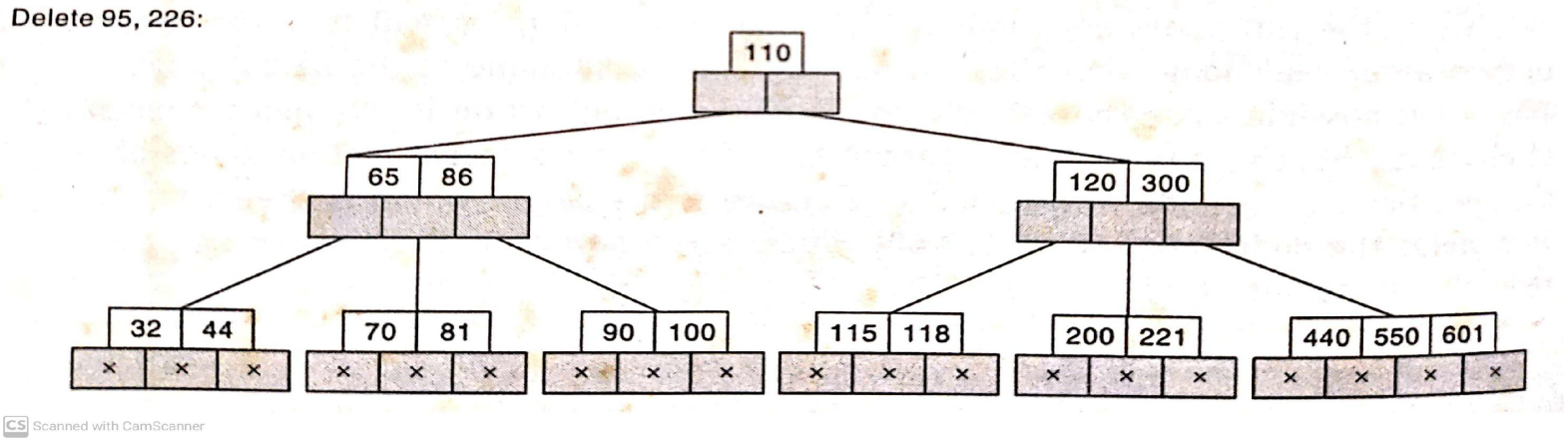
- Data Deletion (continued)

  - While removing a key from a leaf node (continuing…)
    - Otherwise, pull the smallest key of the right sibling to the parent and move down the intervening parent element to the leaf
    - If both the sibling nodes have only minimum number of entries, create a new leaf node out of the two leaf nodes and the intervening element of the parent node
    - If while borrowing the intervening element from the parent node, it leaves the number of keys in parent node to be below the minimum number, propagate the process upwards ultimately resulting in a reduction of height of the B-tree

B Tree

- Data deletion (example-1)

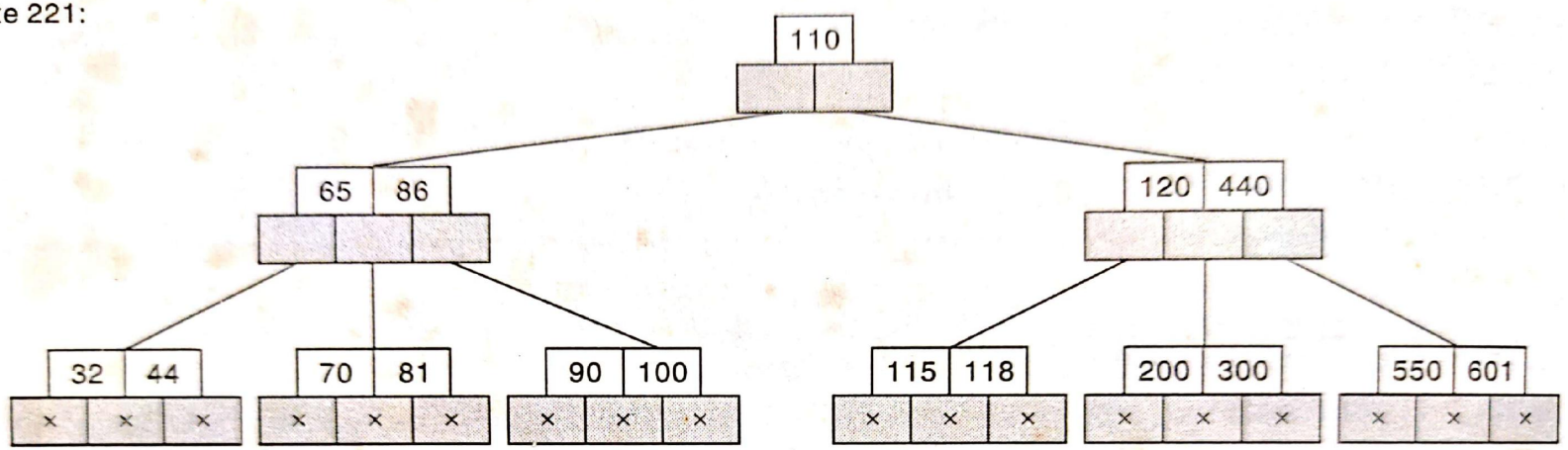




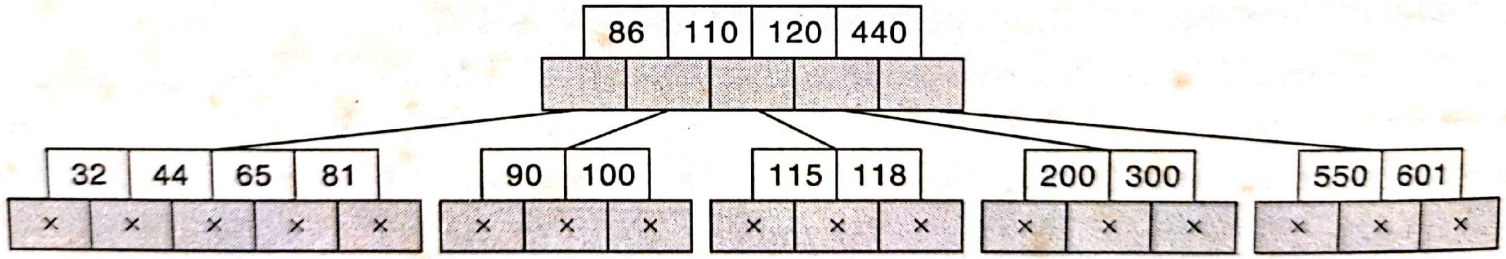
B Tree

- Data deletion (example-2)

Delete 221:



Delete 70:



Queries?

Practice Problem

1. Consider the following data that need to be inserted in a B tree having order 5

   { 25, 40, 15, 10, 6, 28, 32, 35, 50, 5, 12, 8, 10, 38, 3, 45, 23, 42, 1, 50, 22, 17 }

2. Delete the following set of data from the previously maintained B tree data structure

   { 25, 6, 50, 5, 3, 45, 17 }