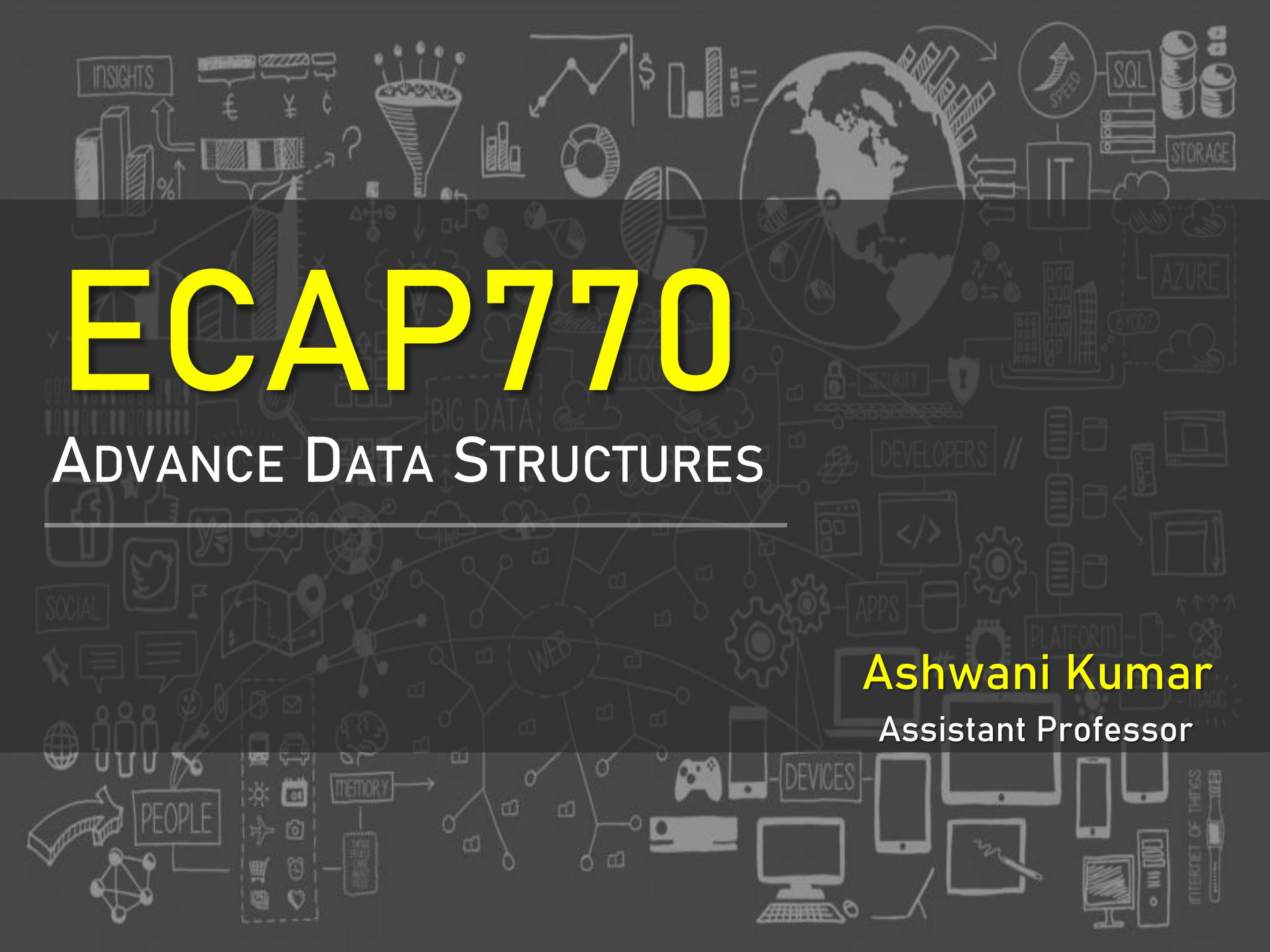


ECAP770

Advance Data Structures

Ashwani Kumar

Assistant Professor

Learning Outcomes

After this lecture, you will be able to

- Understand B tree operations
  - Search Operation
  - Insert Operation
  - Delete Operation

B Tree

- B Tree is a self-balancing data structure.
- For better memory efficiency user need to follow specific set of rules for searching, inserting, and deleting operations.
- B Tree is a m-way tree that can be used for disk access.

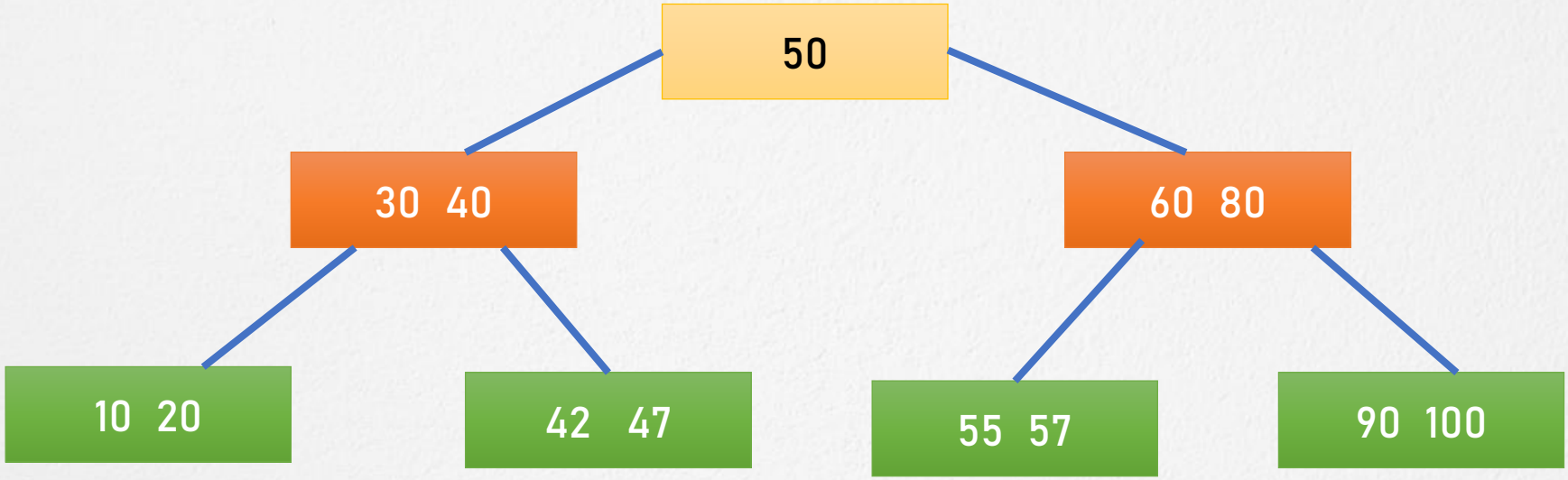
B Tree

- A B Tree of order m can have at most m-1 keys and m children.

or

- Each node can contain more than one key
- Each node can have more than two children

B tree



B Tree properties

B-Tree of Order m has the following properties:

1 - All leaf nodes must be at same level.

2 - All nodes except root must have at least [m/2]-1 keys and maximum of m-1 keys.

3 - All non leaf nodes except root (i.e. all internal nodes) must have at least m/2 children.

B Tree properties

4 - If the root node is a non leaf node, then it must have at least 2 children.

5 - A non leaf node with n-1 keys must have n number of children.

6 - All the key values in a node must be in Ascending Order.

Search Operation

- The search operation in B-Tree is similar to the search operation in Binary Search Tree.
- In B-Tree search process starts from the root node but here we make an n-way decision every time. Where 'n' is the total number of children the node has.

Search algorithm

- Let the key (the value) to be searched by “X”.
- Start searching from the root and recursively traverse down.
- If X is lesser than the root value, search left sub tree, if X is greater than the root value, search the right sub tree.
- If the node has the found X, simply return the node.
- If the X is not found in the node, traverse down to the child with a greater Key.
- If X is not found in the tree, we return NULL.

Insertion Operation

- Insertions in B-Tree performed only at the leaf node level.
- Inserting operation performed with two steps: searching the appropriate node to insert the element and splitting the node if required.

Insertion algorithm

- Check whether tree is Empty.
- If tree is Empty, then create a new node with new key value and insert it into the tree as a root node
- If tree is not Empty, then, find the appropriate leaf node at which the node can be inserted.
- If the leaf node contain less than m-1 keys then insert the element in the increasing order.

Insertion algorithm

- 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 upto its parent node.

  -If the parent node also contain m-1 number of keys, then split it too by following the same steps.

Deletion operation

- In case of deletion from B-Tree user need to follow more rule as compared to search and insertion.
- Three case for deletion from B-Tree

    If the key is in the leaf node

    If the key is in an internal node

    If the key is in a root node

Key is in the leaf node, case-1

- Target is in the leaf node, more than min keys.
- Deleting this will not violate the property of B Tree
- Target is in leaf node, it has min key nodes
- Deleting this will violate the property of B Tree

Key is in the leaf node, case-1

- Target node can borrow key from immediate left node, or immediate right node (sibling)
- The sibling will say yes if it has more than minimum number of keys
- The key will be borrowed from the parent node, the max value will be transferred to a parent, the max value of the parent node will be transferred to the target node, and remove the target value

Key is in the leaf node, case-1

- Target is in the leaf node, but no siblings have more than min number of keys Search for key
- Merge with siblings and the minimum of parent nodes
- Total keys will be now more than min
- The target key will be replaced with the minimum of a parent node

Key is in an internal node, case-2

- Either choose, in- order predecessor or in-order successor
- In case the of in-order predecessor, the maximum key from its left sub tree will be selected
- In case of in-order successor, the minimum key from its right sub tree will be selected

Key is in an internal node, case-2

- If the target key's in-order predecessor has more than the min keys, only then it can replace the target key with the max of the in-order predecessor
- If the target key's in-order predecessor does not have more than min keys, look for in-order successor's minimum key.
- If the target key's in-order predecessor and successor both have less than min keys, then merge the predecessor and successor.

Key is in a root node, Case- 3

- Replace with the maximum element of the in-order predecessor sub tree
- If, after deletion, the target has less than min keys, then the target node will borrow max value from its sibling via sibling's parent.
- The max value of the parent will be taken by a target, but with the nodes of the max value of the sibling.

That's all for now...