الدكتور
اثير العاني

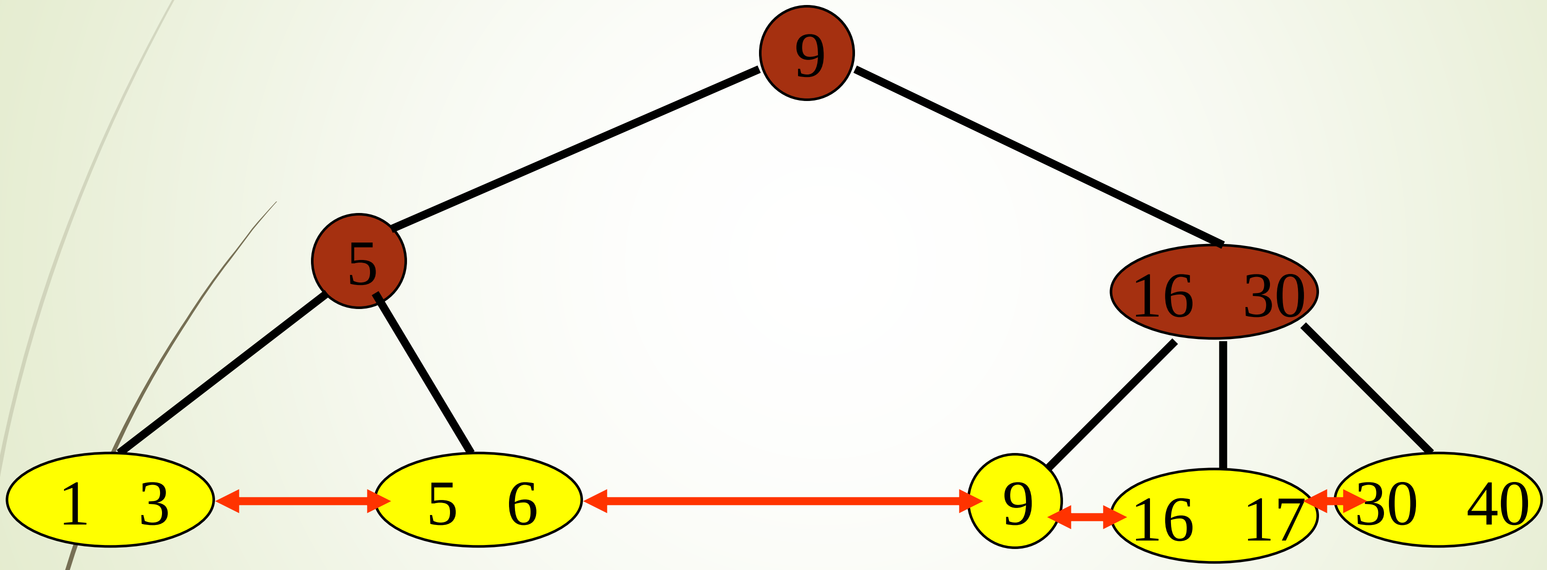
B^+-Trees

- Same structure as B-trees.
- Dictionary pairs are in leaves only. Leaves form a doubly-linked list.
- Remaining nodes have following structure:

$$j\ a_0\ k_1\ a_1\ k_2\ a_2\ \ldots\ k_j\ a_j$$

- j = number of keys in node.
- a_i is a pointer to a subtree.
- $k_i <=$ smallest key in subtree a_i and $>$ largest in a_{i-1}.

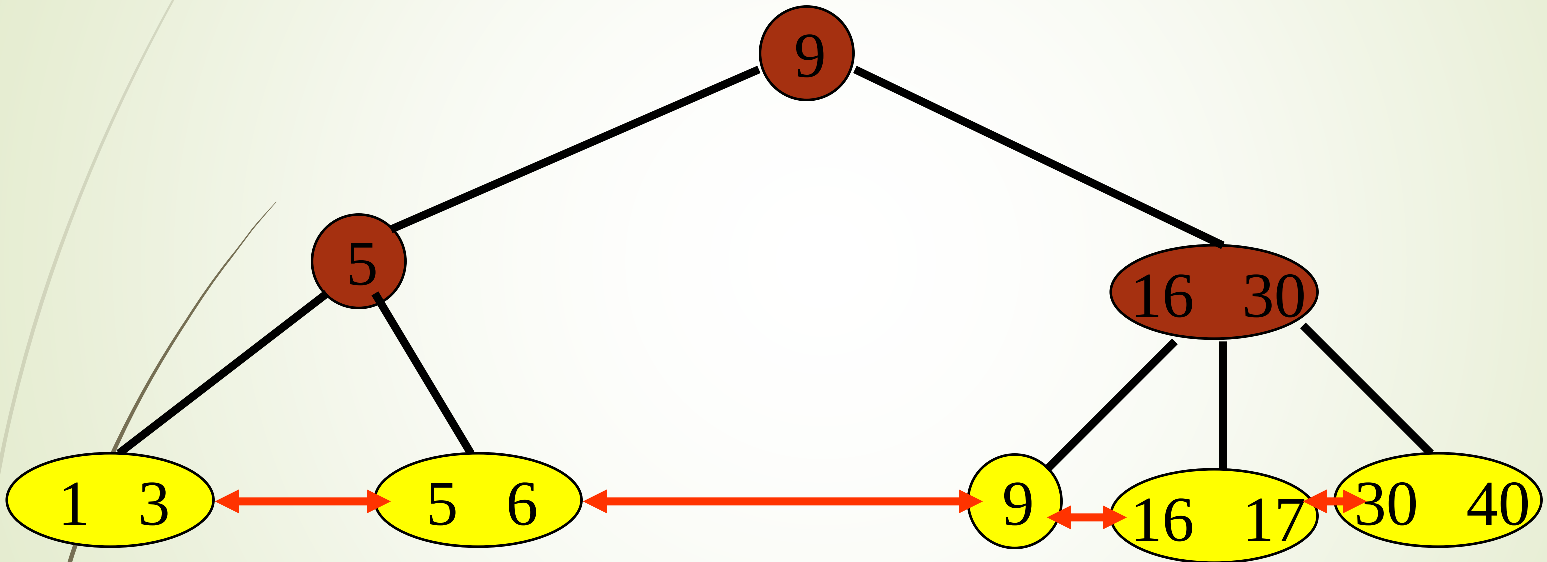
Example B+-tree



index node

leaf/data node

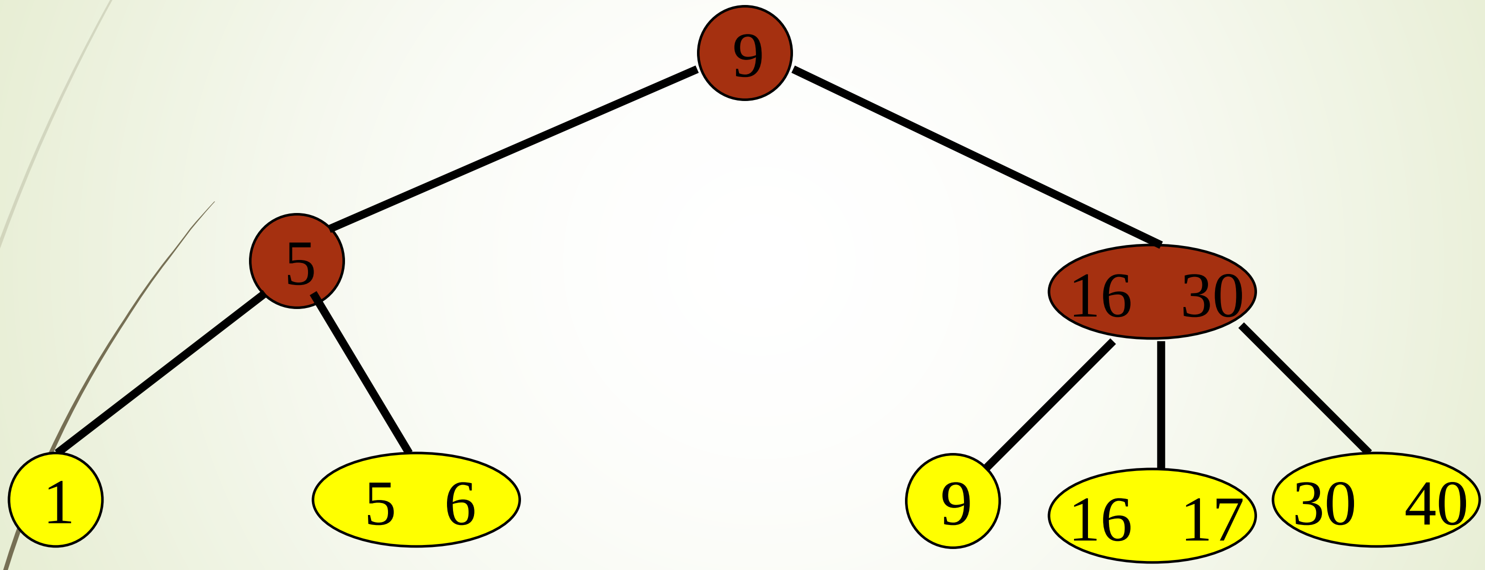
B+-tree—Search



key = 5

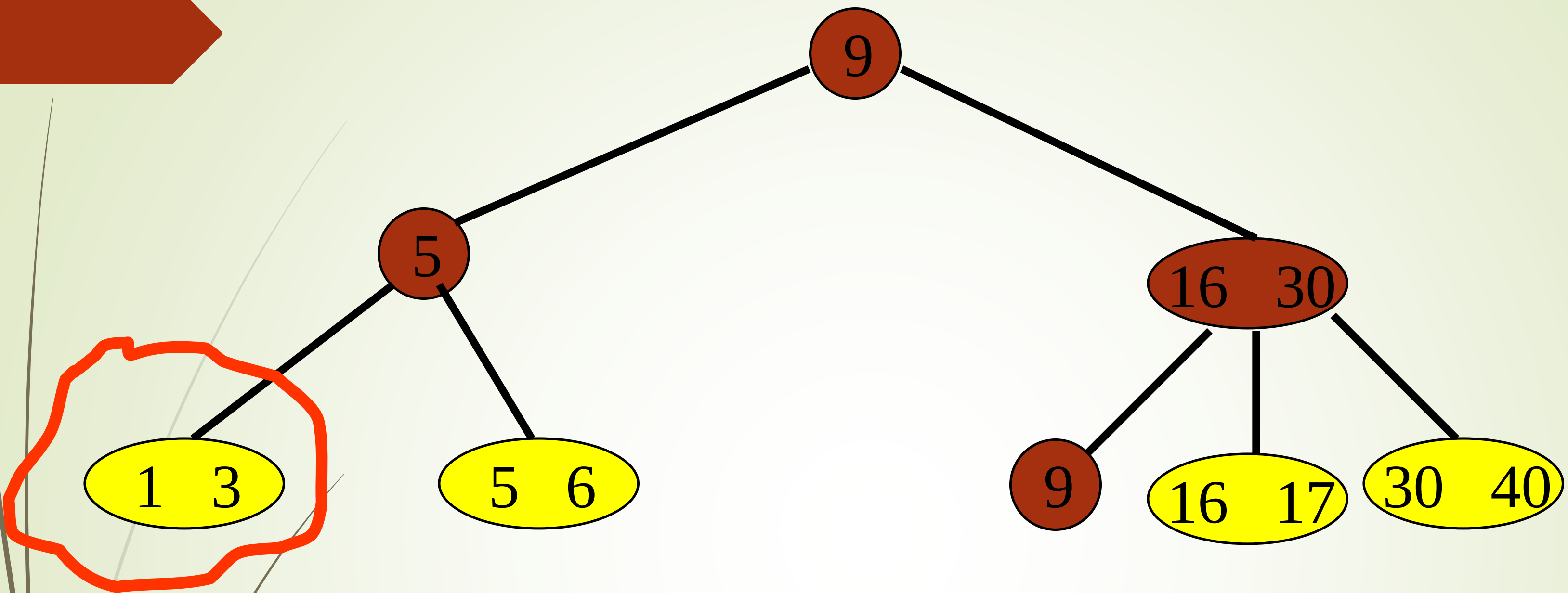
6 <= key <= 20

B+-tree—Insert



Insert 10

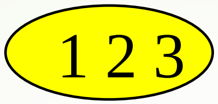
Insert



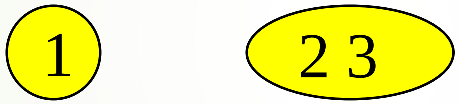
- Insert a pair with key = 2.
- New pair goes into a 3-node.

Insert Into A 3-node

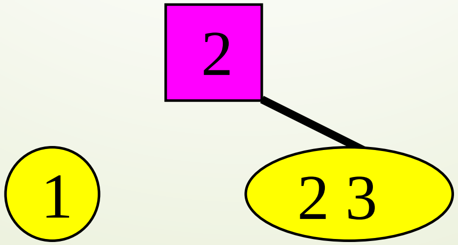
Insert new pair so that the keys are in ascending order.



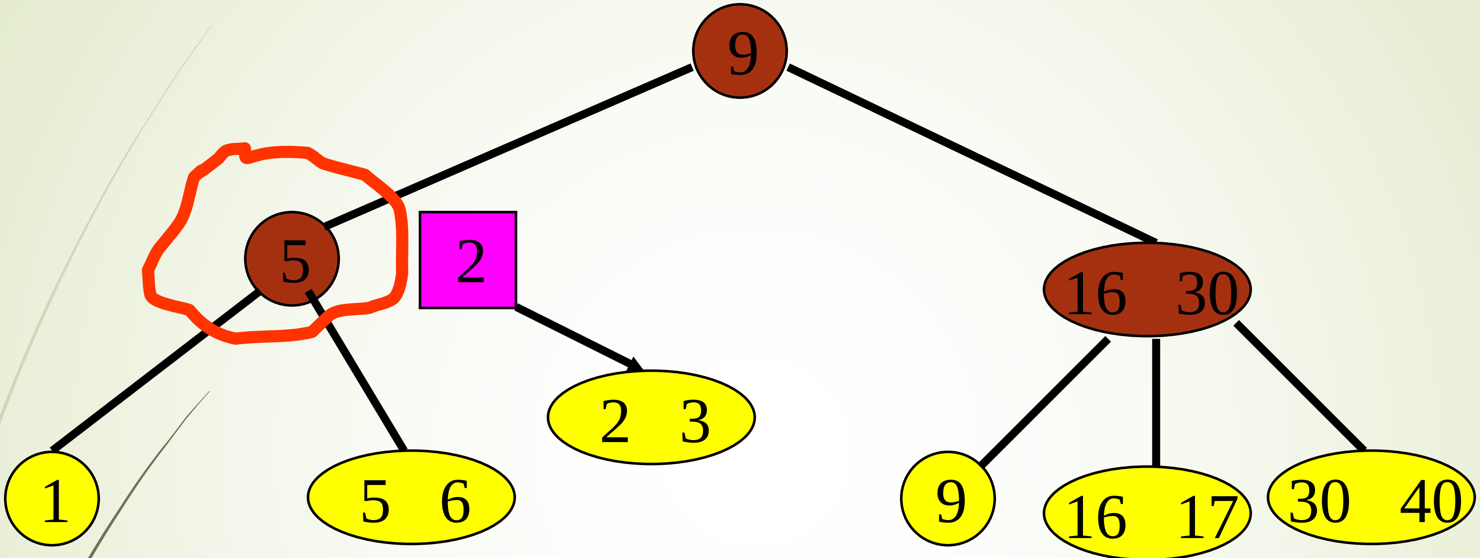
- Split into two nodes.



- Insert smallest key in new node and pointer to this new node into parent.

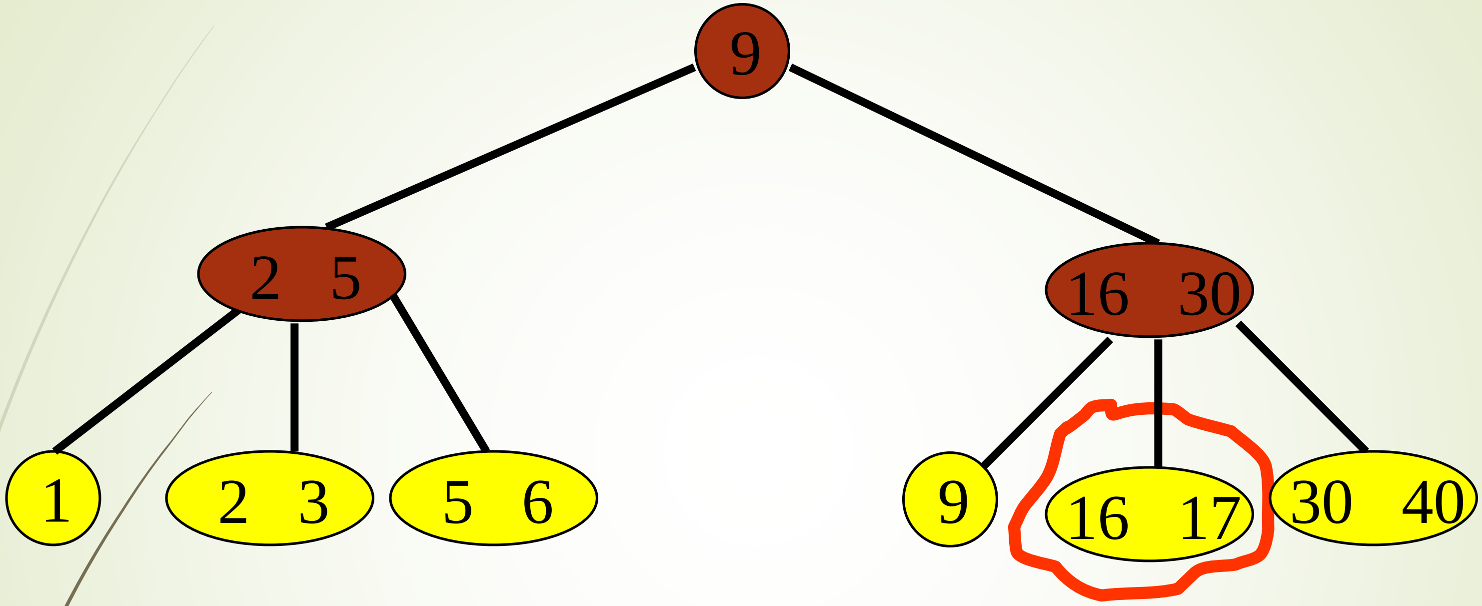


Insert



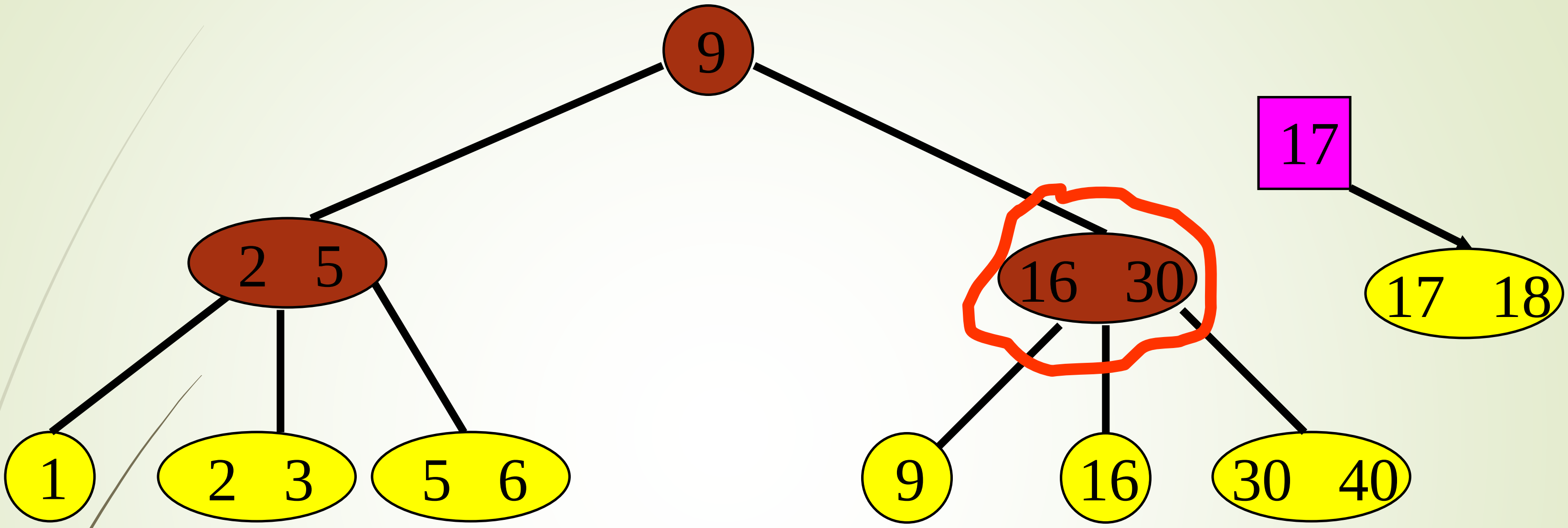
- Insert an index entry 2 plus a pointer into parent.

Insert



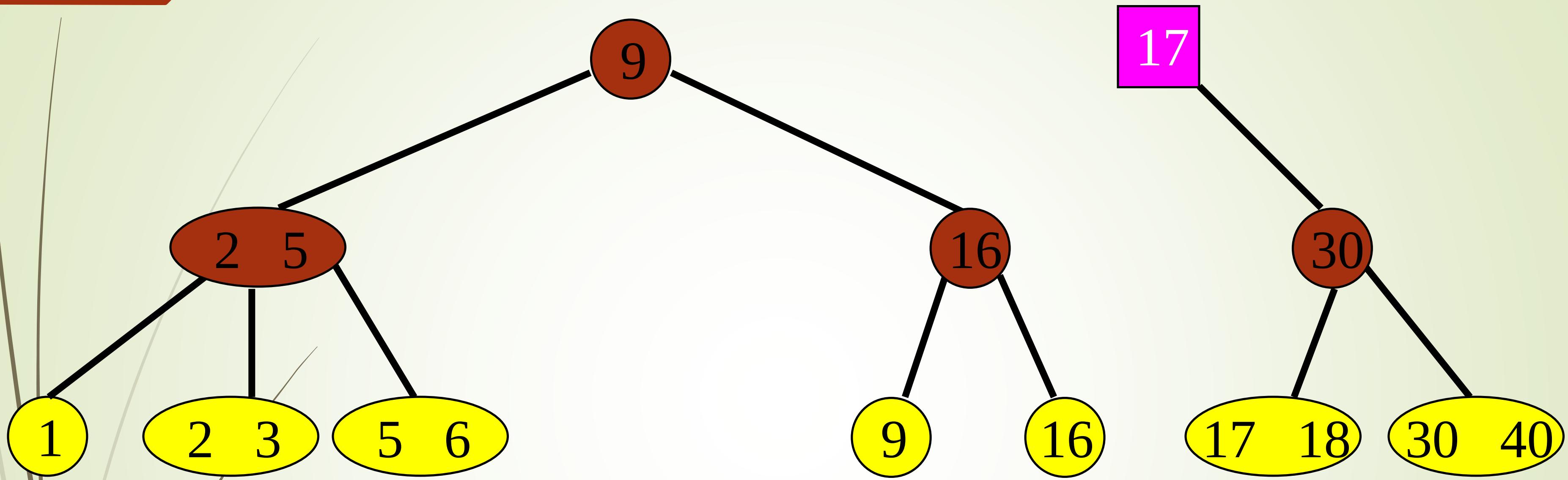
- Now, insert a pair with key = 18.

Insert



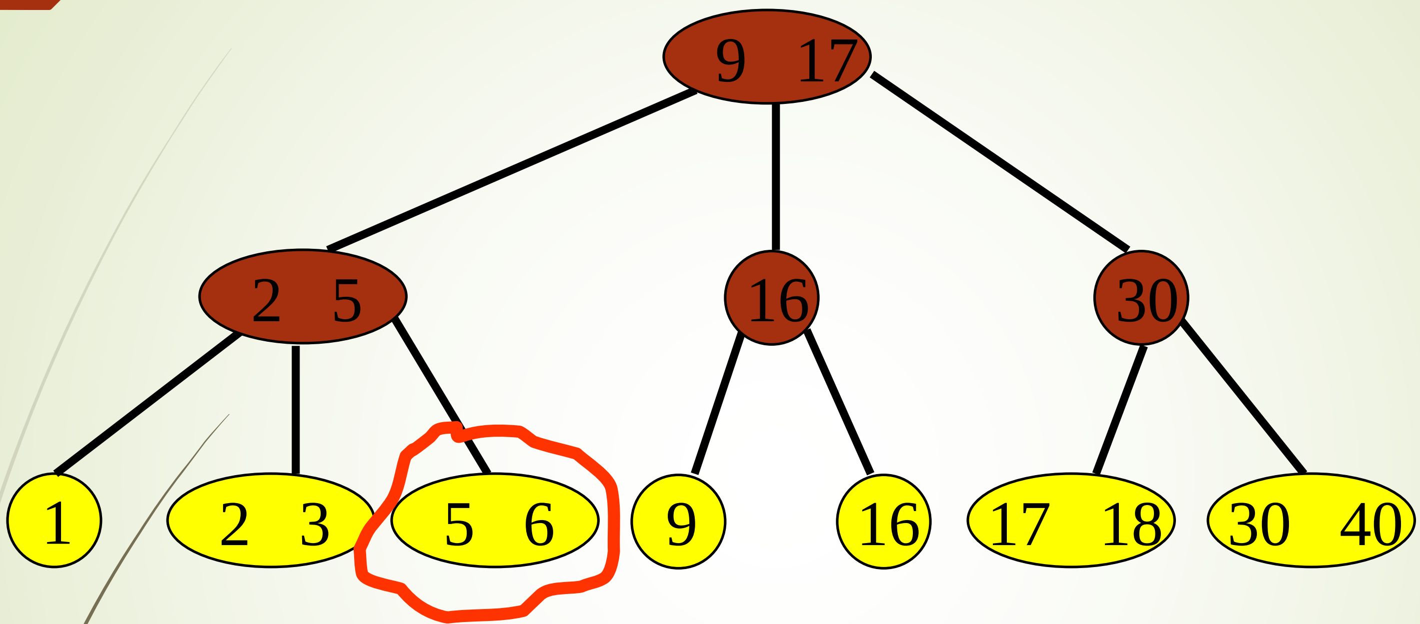
- Now, insert a pair with key = 18.
- Insert an index entry17 plus a pointer into parent.

Insert



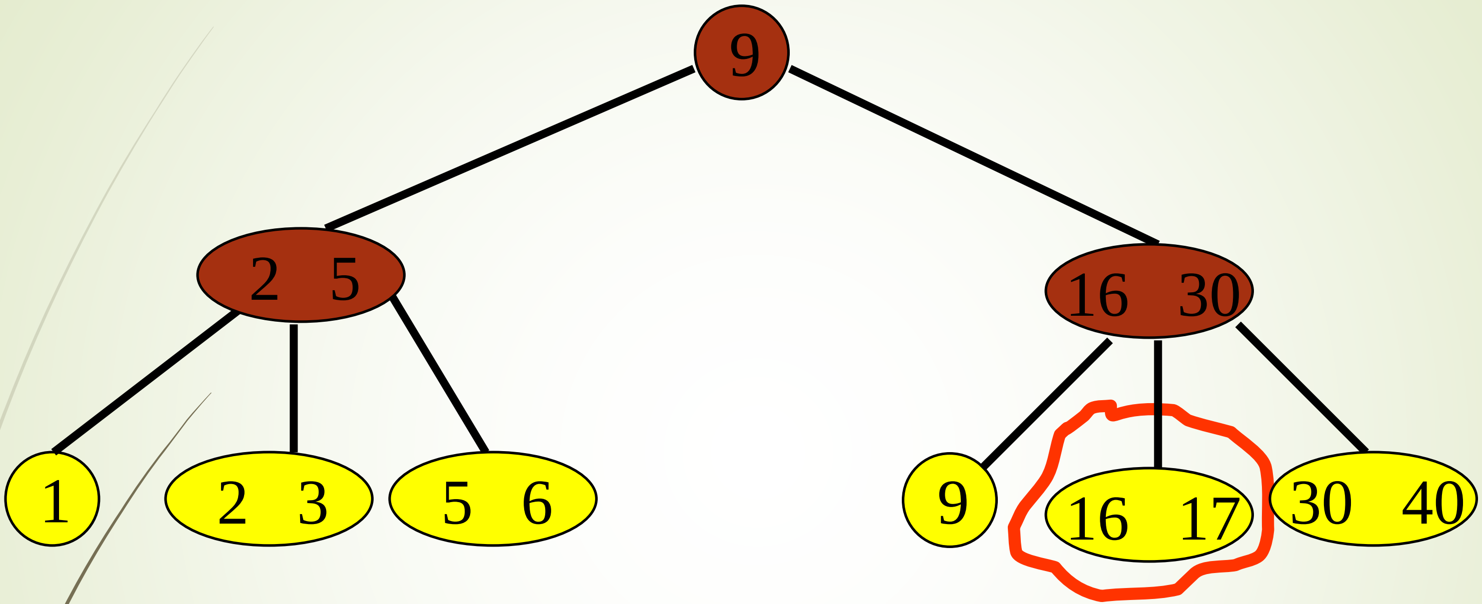
- Now, insert a pair with key = 18.
- Insert an index entry17 plus a pointer into parent.

Insert



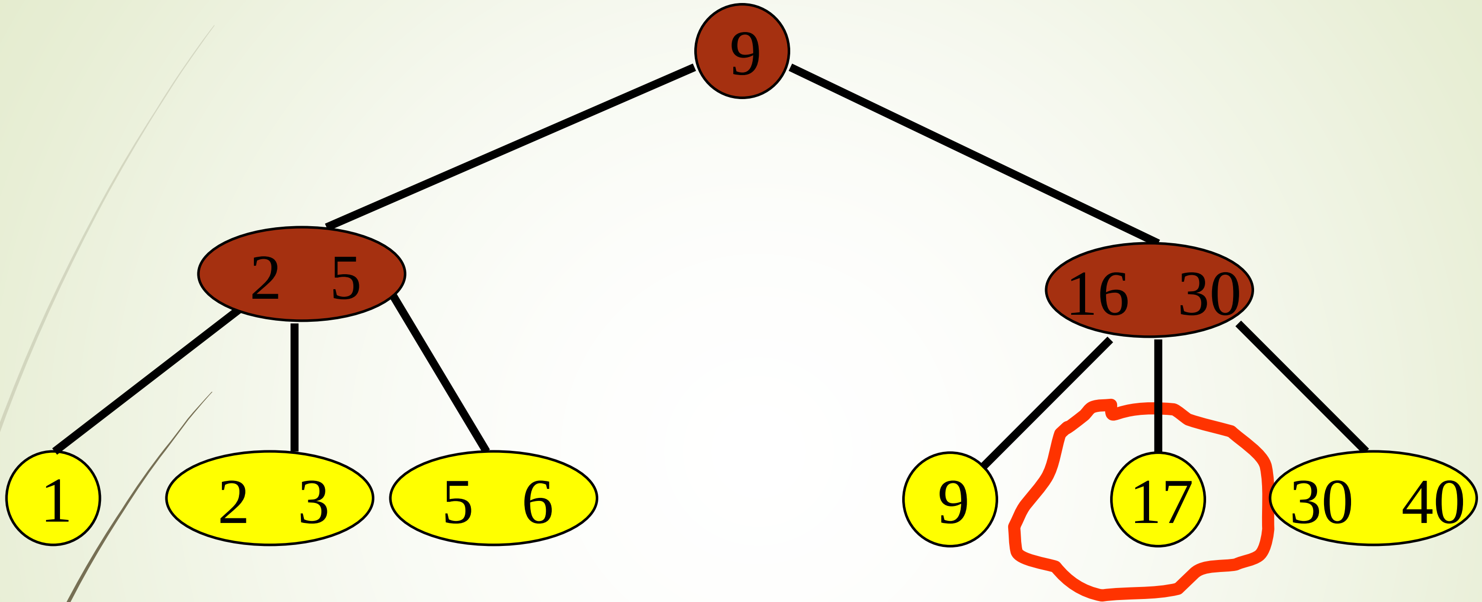
- Now, insert a pair with key = 7.

Delete



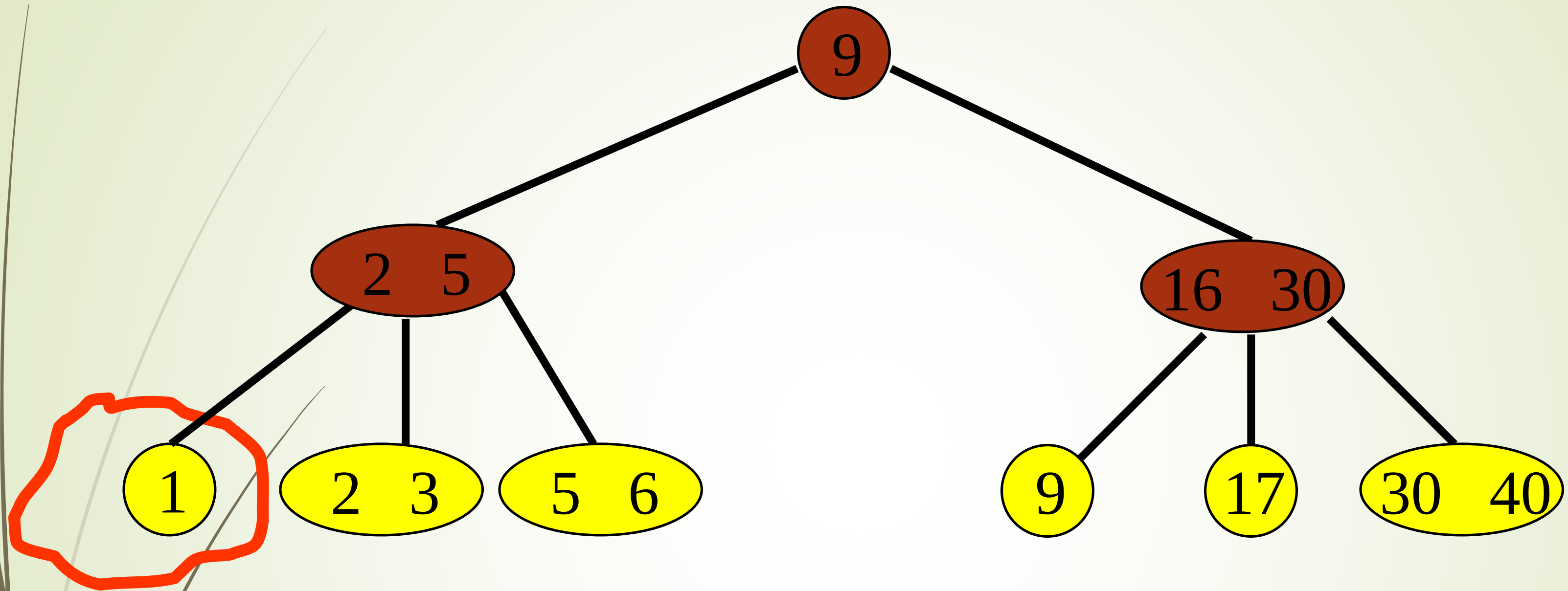
- Delete pair with key = 16.
- Note: delete pair is always in a leaf.

Delete



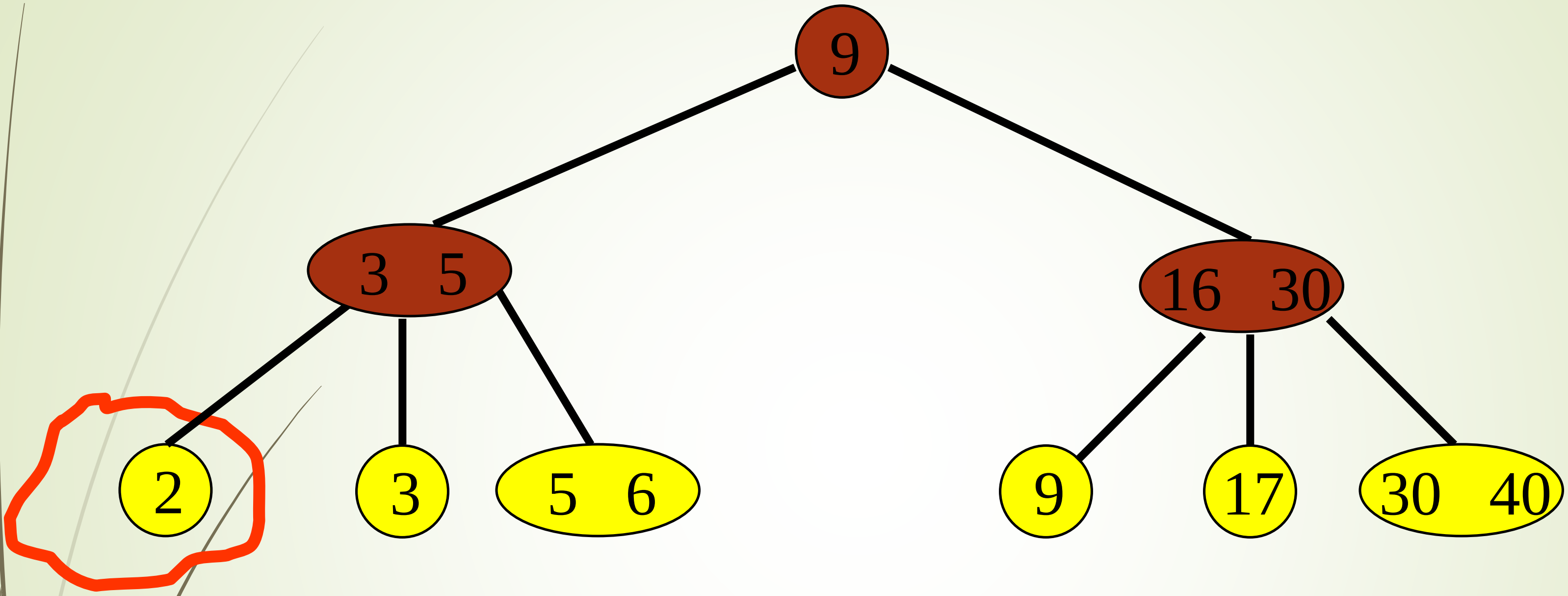
- Delete pair with key = 16.
- Note: delete pair is always in a leaf.

Delete



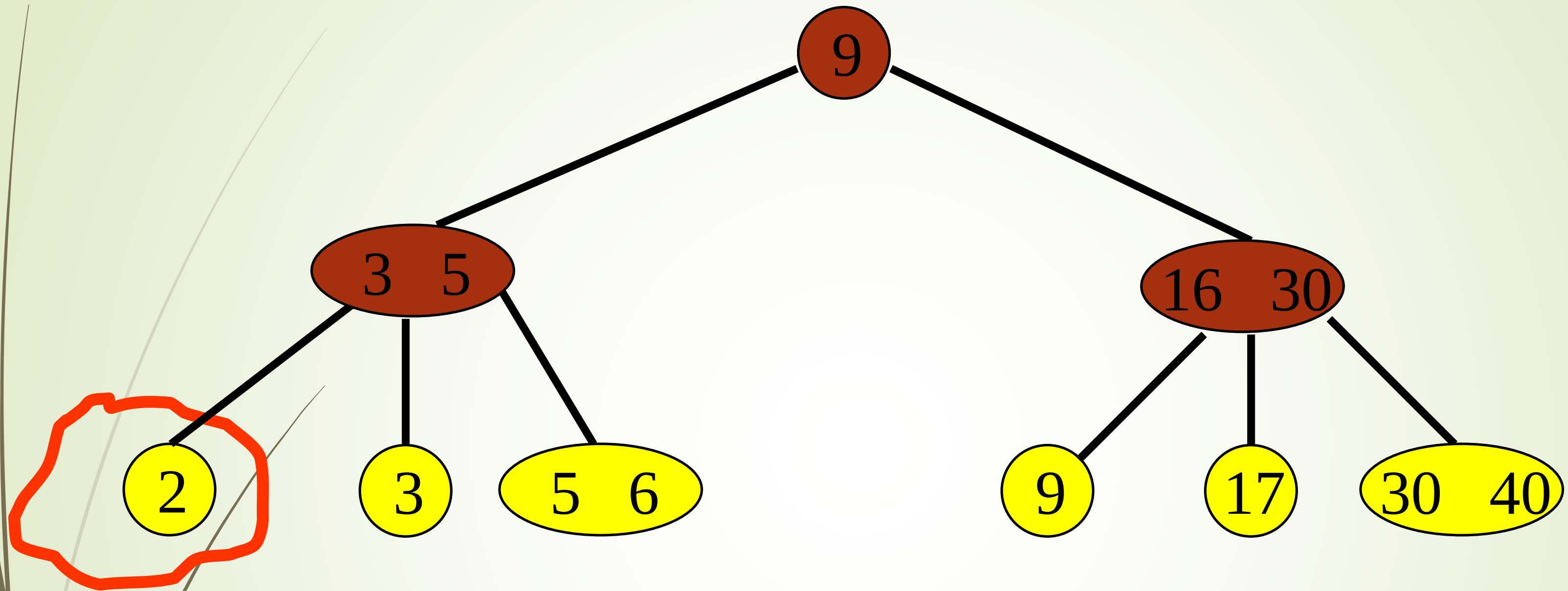
- Delete pair with key = 1.
- Get >= 1 from sibling and update parent key.

Delete



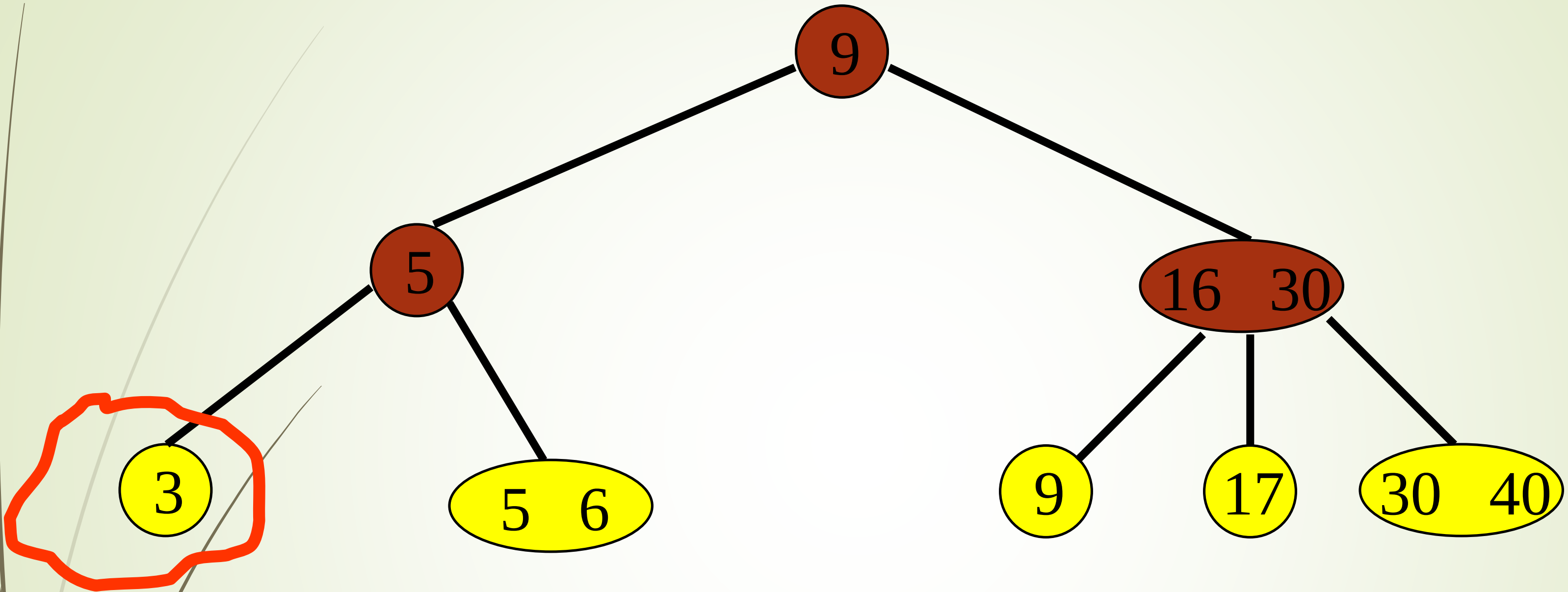
- Delete pair with key = 1.
- Get >= 1 from sibling and update parent key.

Delete



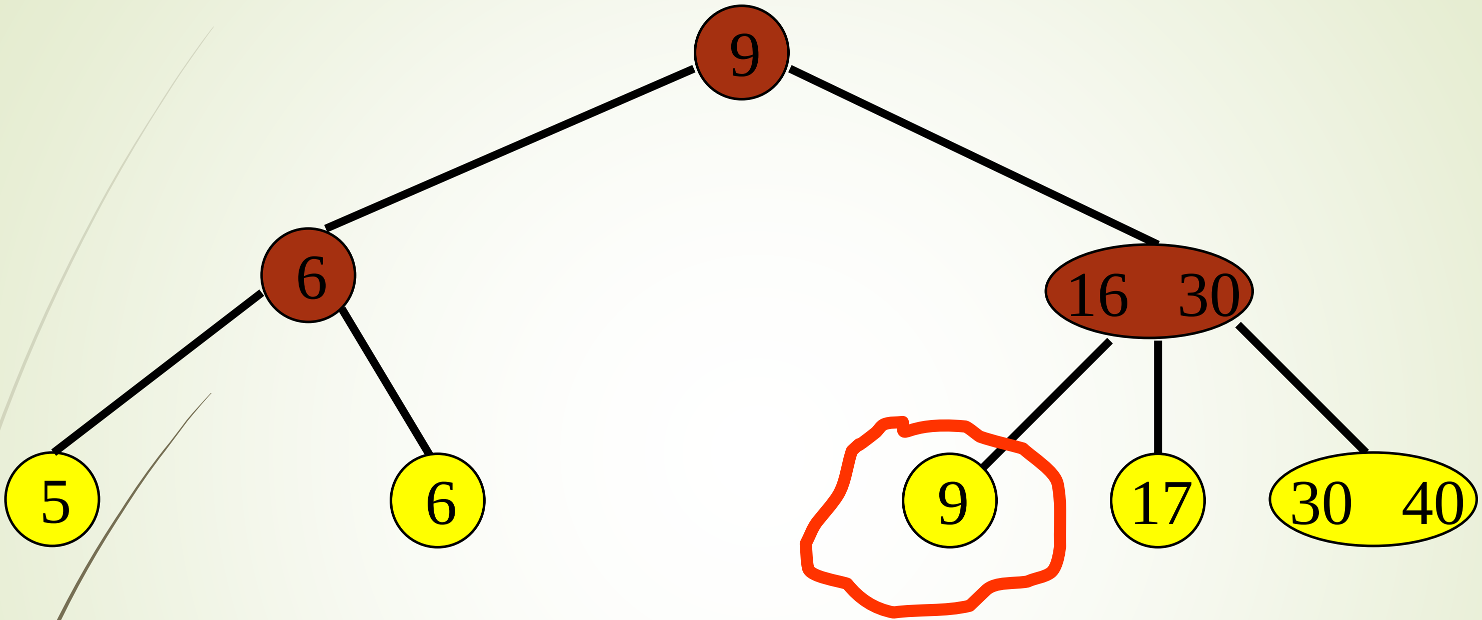
- Delete pair with key = 2.
- Merge with sibling, delete in-between key in parent.

Delete



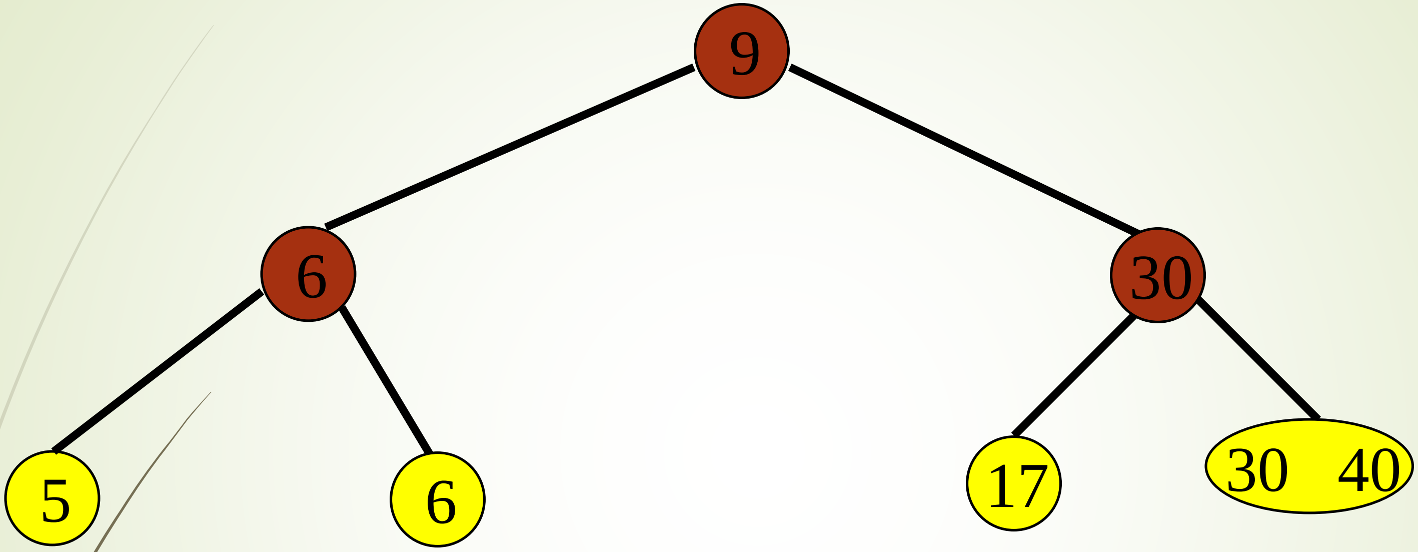
- Delete pair with key = 3.
- Get >= 1 from sibling and update parent key.

Delete

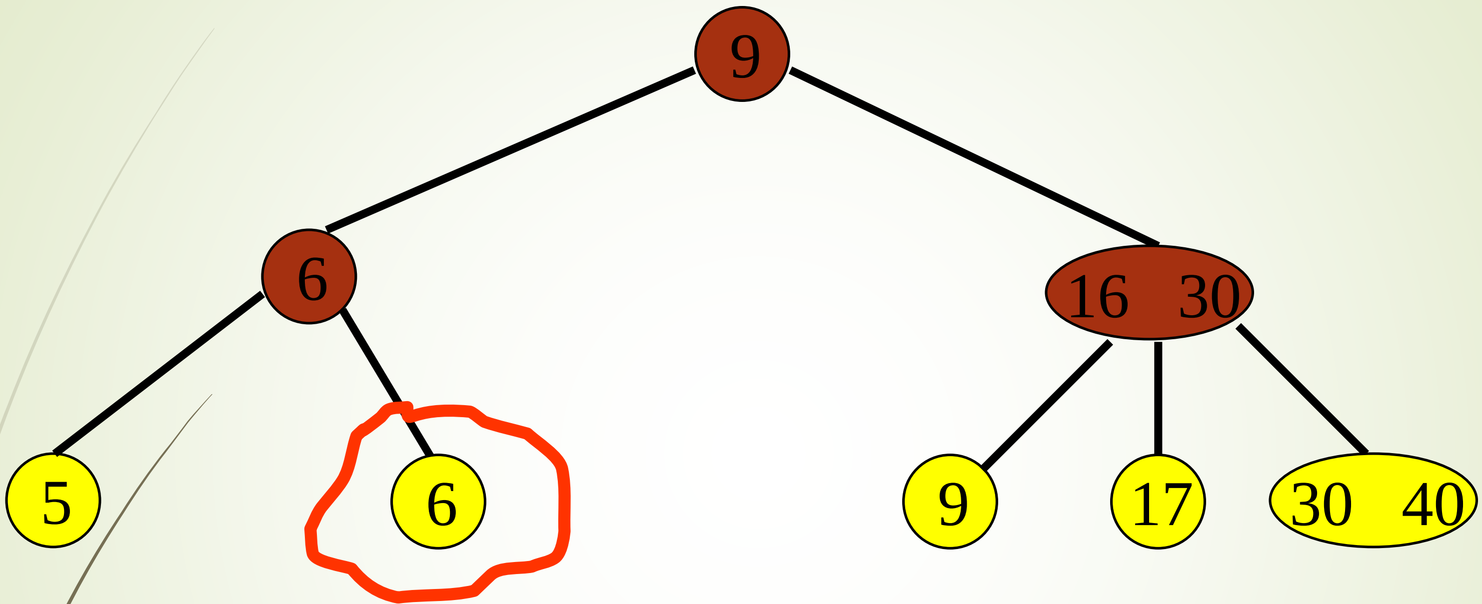


- Delete pair with key = 9.
- Merge with sibling, delete in-between key in parent.

Delete

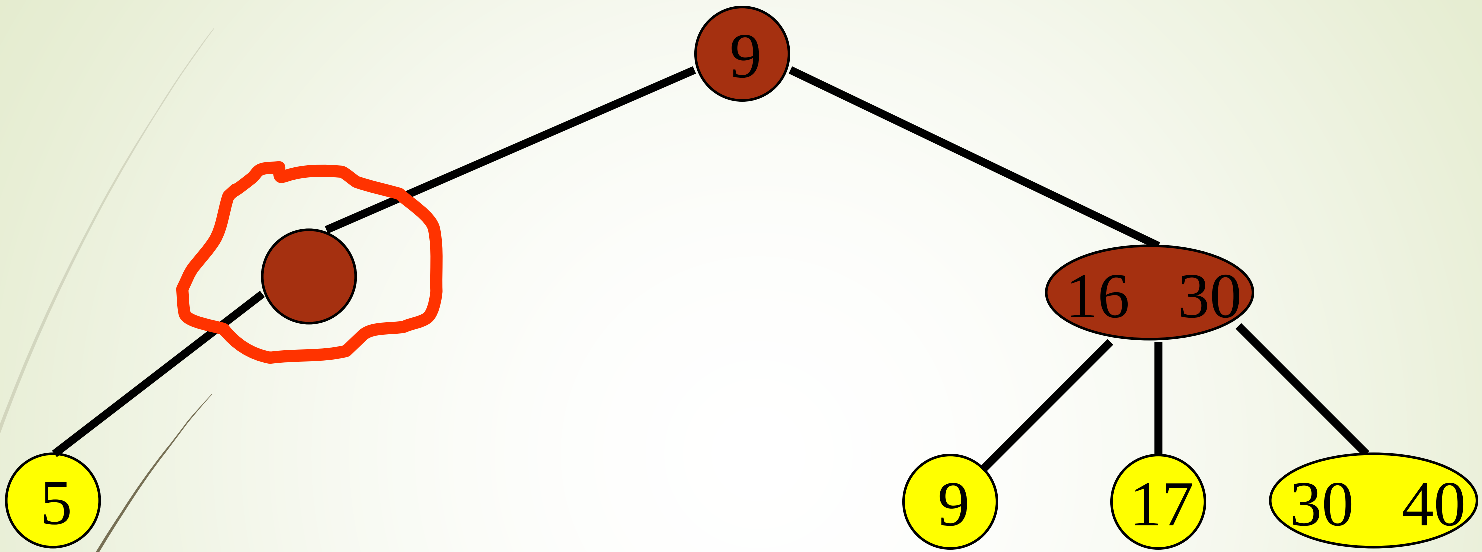


Delete



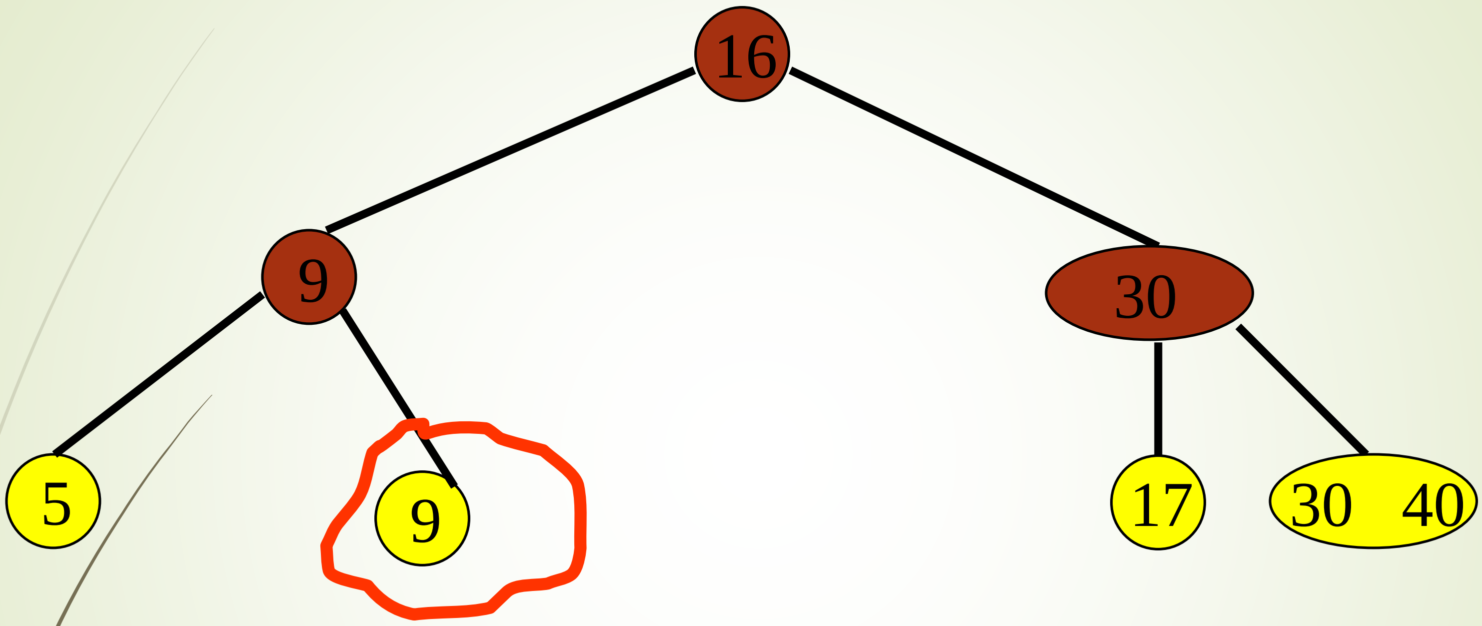
- Delete pair with key = 6.
- Merge with sibling, delete in-between key in parent.

Delete



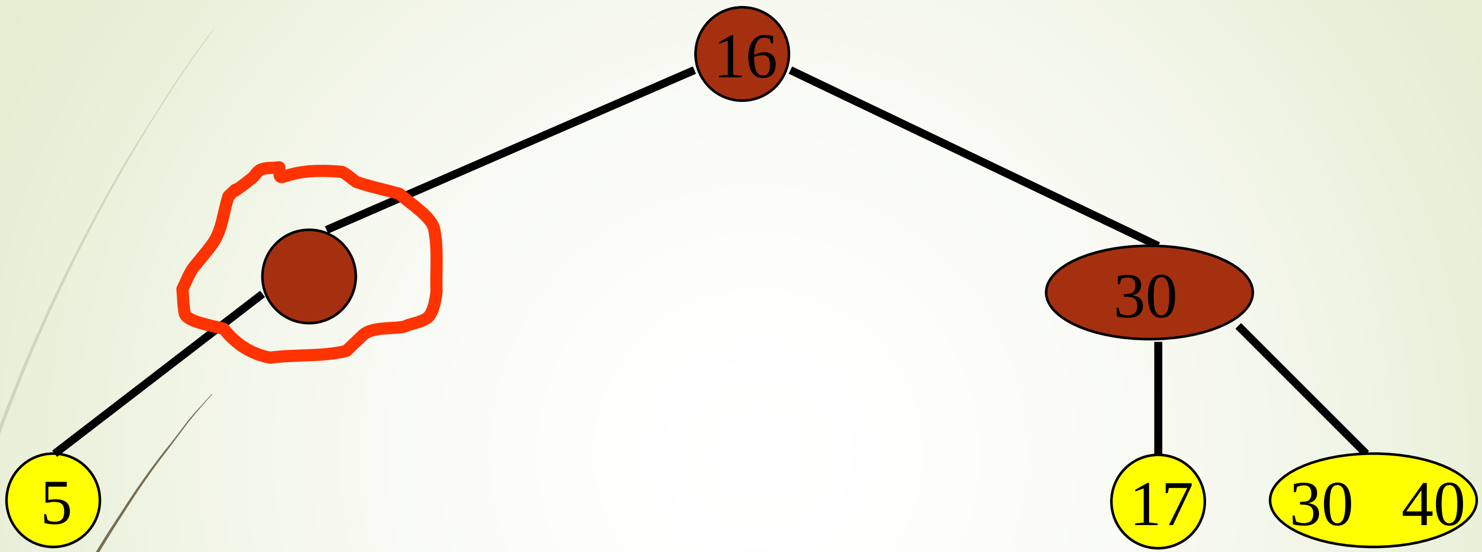
- Index node becomes deficient.
- Get >= 1 from sibling, move last one to parent, get parent key.

Delete



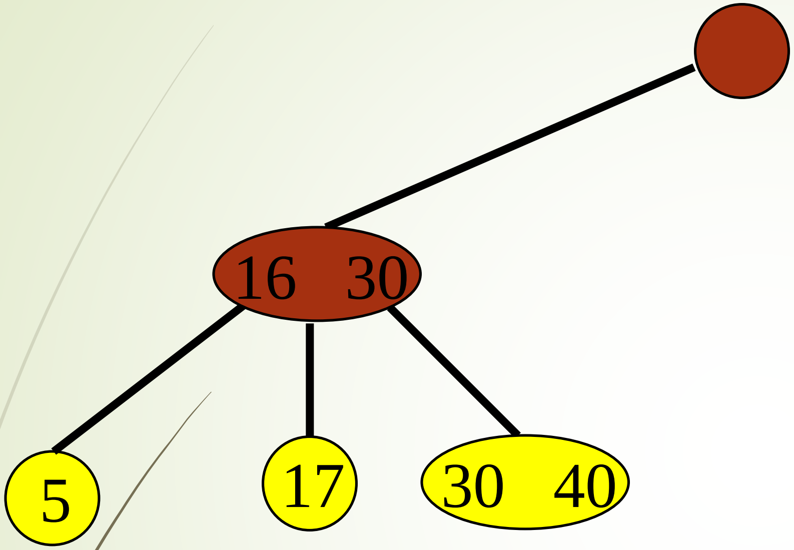
- Delete 9.
- Merge with sibling, delete in-between key in parent.

Delete



- Index node becomes deficient.
- Merge with sibling and in-between key in parent.

Delete



- Index node becomes deficient.
- It’s the root; discard.

B*-Trees

- Root has between 2 and 2 * floor((2m – 2)/3) + 1 children.
- Remaining nodes have between ceil((2m – 1)/3) and m children.
- All external/failure nodes are on the same level.

Insert

- When insert node is overfull, check adjacent sibling.
- If adjacent sibling is not full, move a dictionary pair from overfull node, via parent, to nonfull adjacent sibling.
- If adjacent sibling is full, split overfull node, adjacent full node, and in-between pair from parent to get three nodes with floor((2m – 2)/3), floor((2m – 1)/3), floor(2m/3) pairs plus two additional pairs for insertion into parent.

Delete

- When combining, must combine 3 adjacent nodes and 2 in-between pairs from parent.
  - Total # pairs involved = 2 * floor((2m-2)/3) + [floor((2m-2)/3) – 1] + 2
  - Equals 3 * floor((2m-2)/3) + 1.
- Combining yields 2 nodes and a pair that is to be inserted into the parent.
  - m mod 3 = 0 => nodes have m – 1 pairs each.
  - m mod 3 = 1 => one node has m – 1 pairs and the other has m – 2
  - m mod 3 = 2 => nodes have m – 2 pairs each.