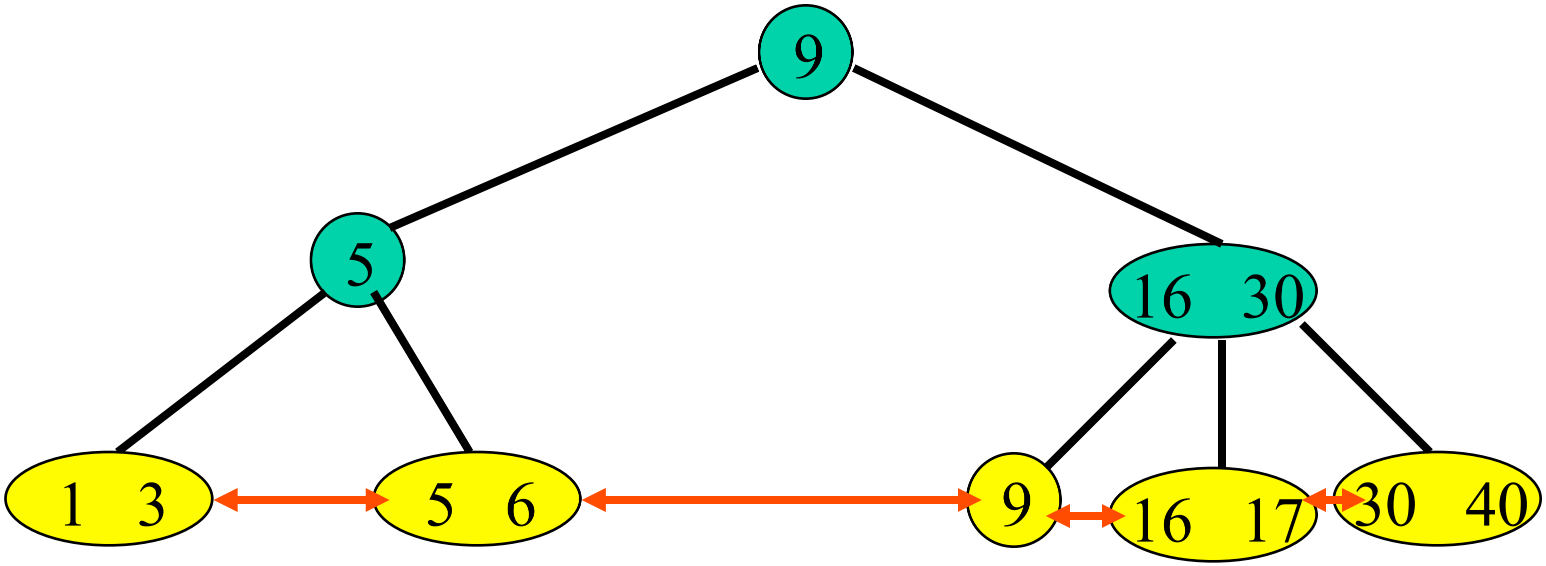
B$^+$-Trees

- 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}.

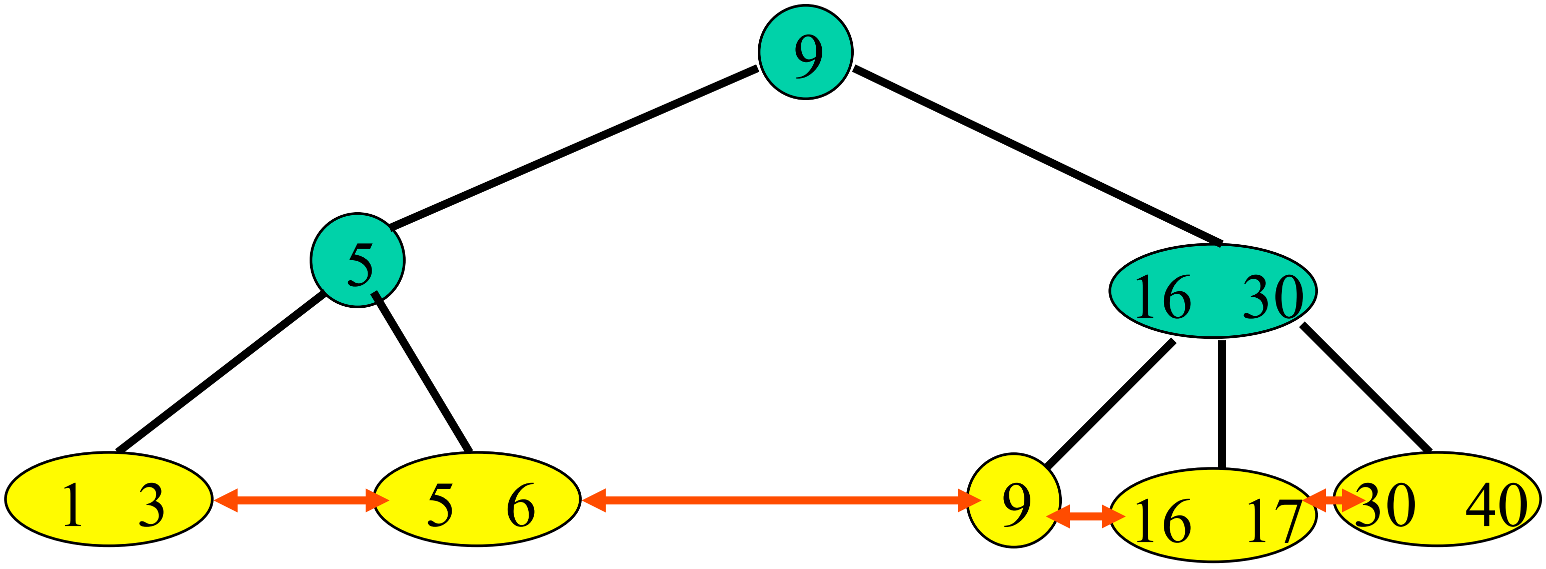
Example B+-tree



🟩 → index node

🟨 → leaf/data node

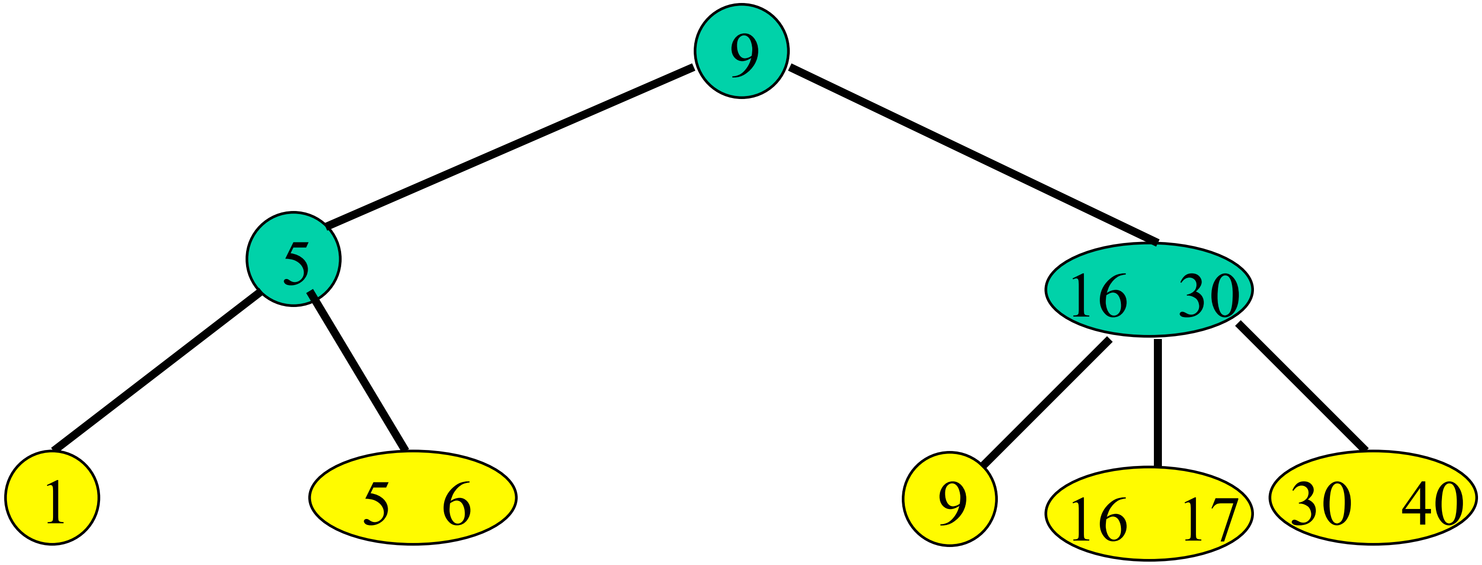
B+-tree—Search



key = 5

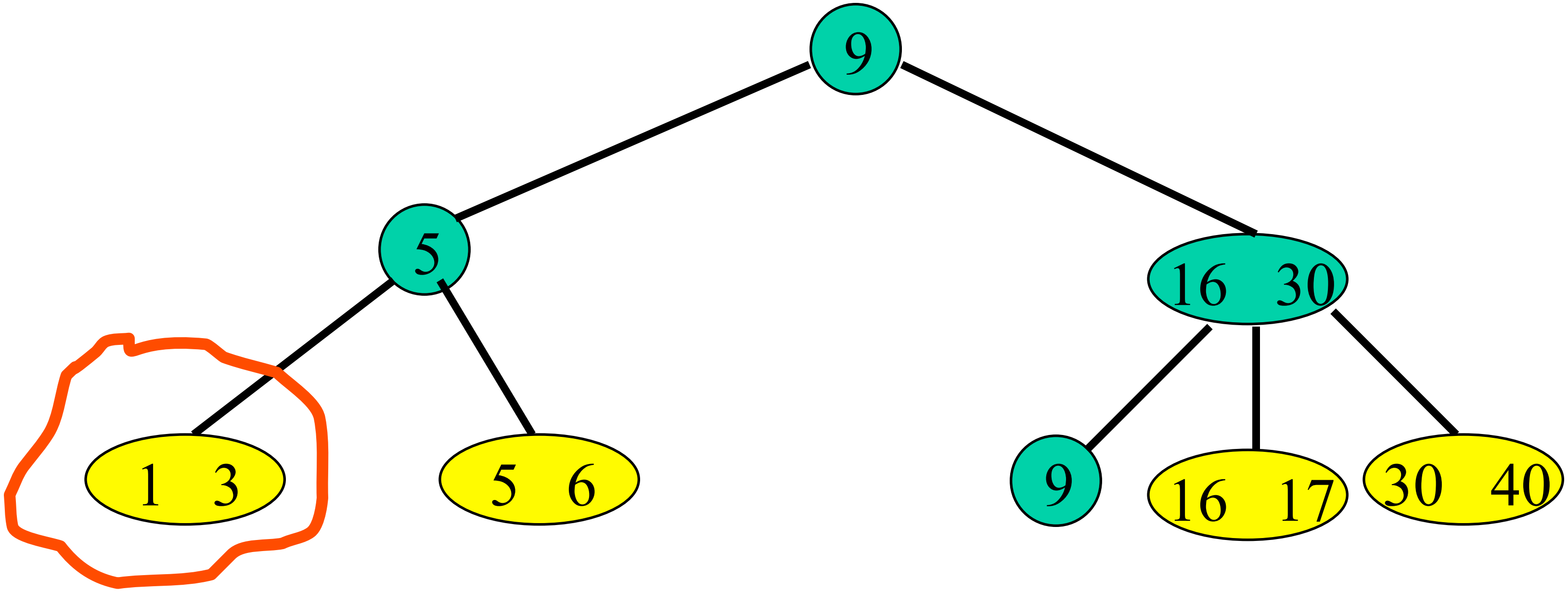
6 <= key <= 20

B+-tree—Insert



Insert 10

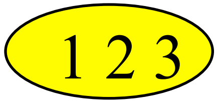
Insert



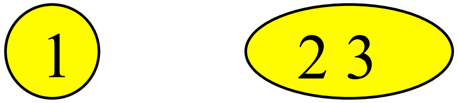
- Insert a pair with key = 2.
- New pair goes into a 3-node.

Insert Into A 3-node

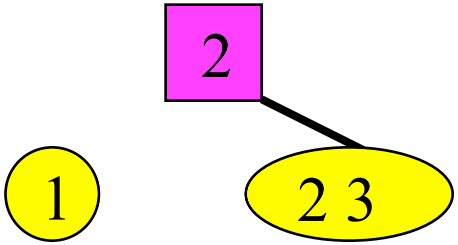
- Insert new pair so that the keys are in ascending order.



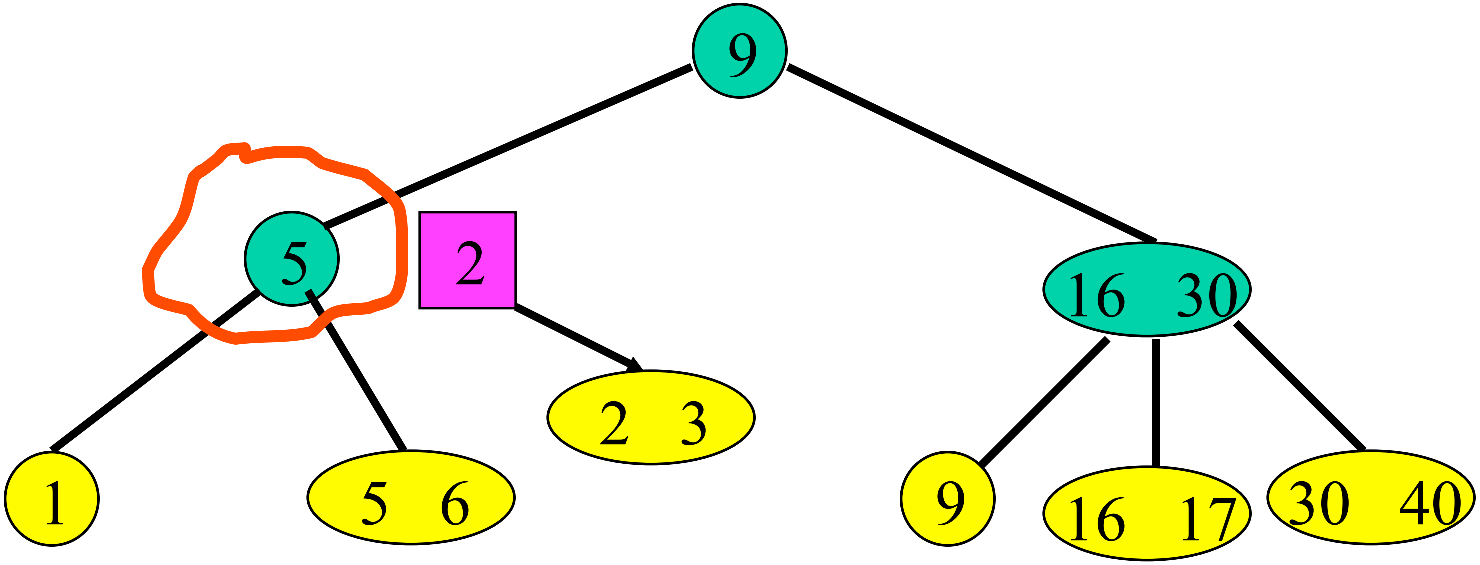
- Split into two nodes.



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

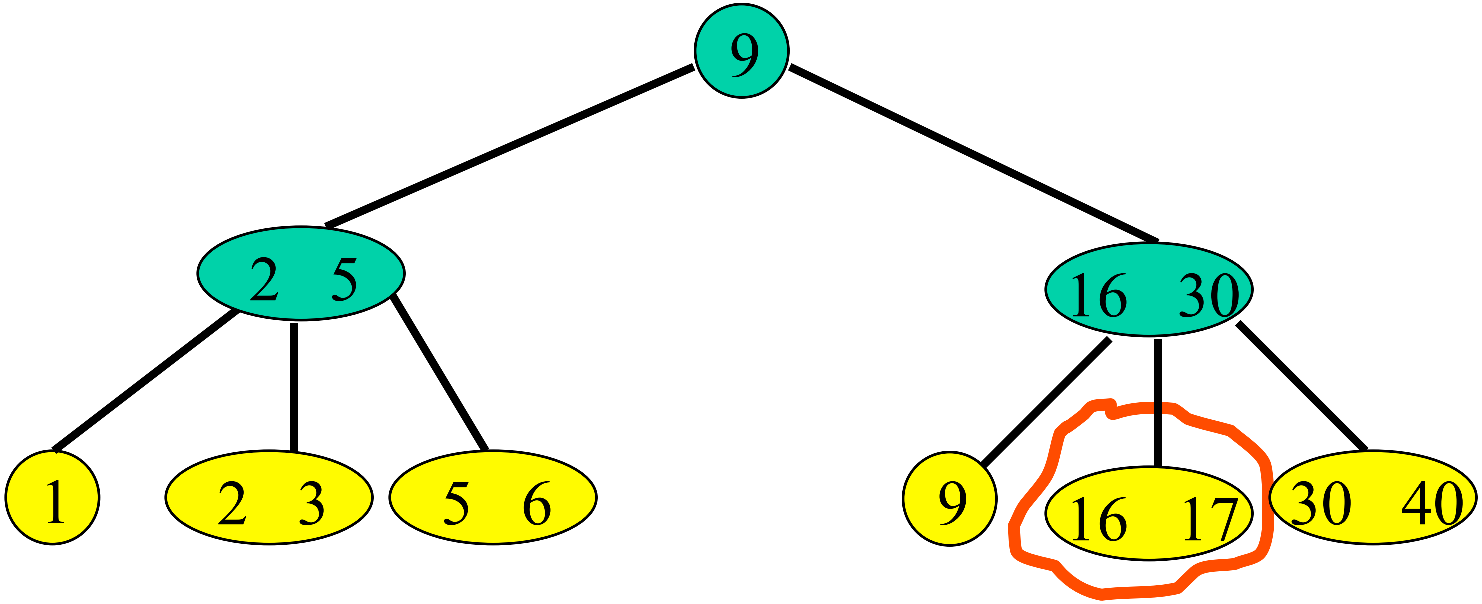


Insert



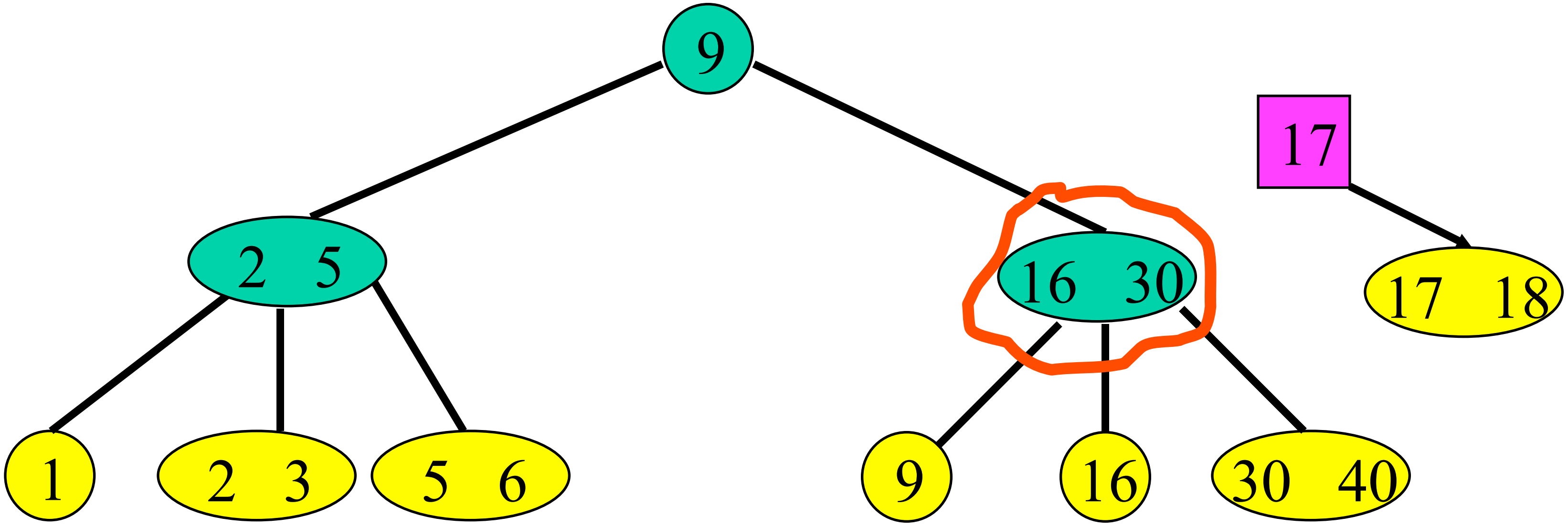
- Insert an index entry 2 plus a pointer into parent.

Insert



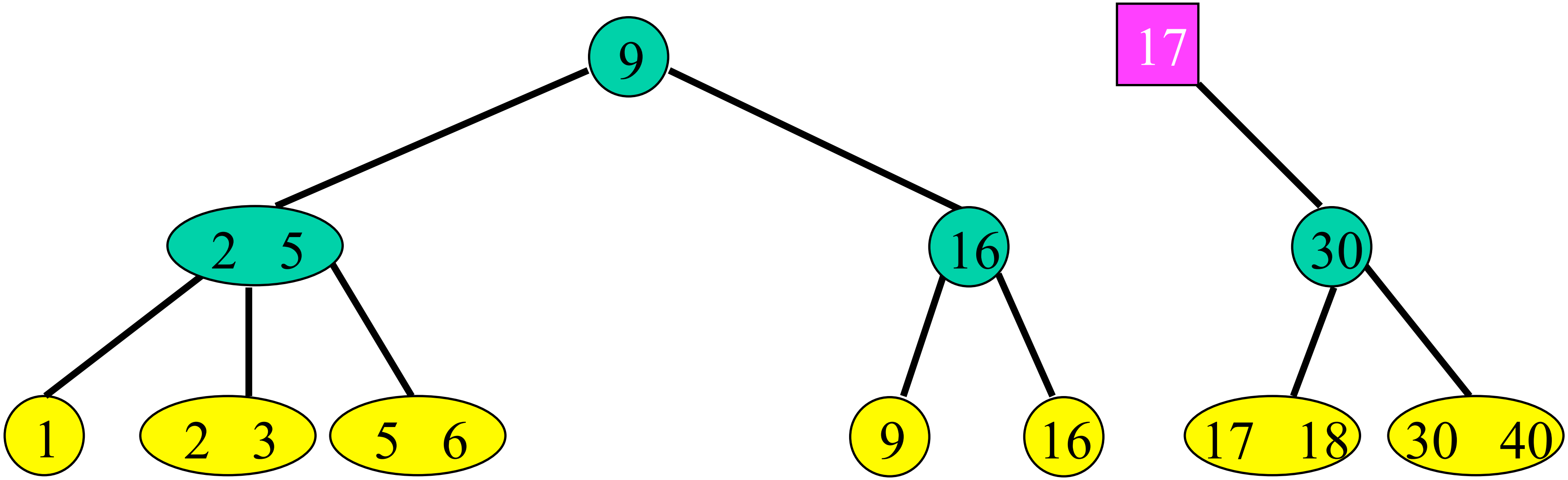
- Now, insert a pair with key = 18.

Insert



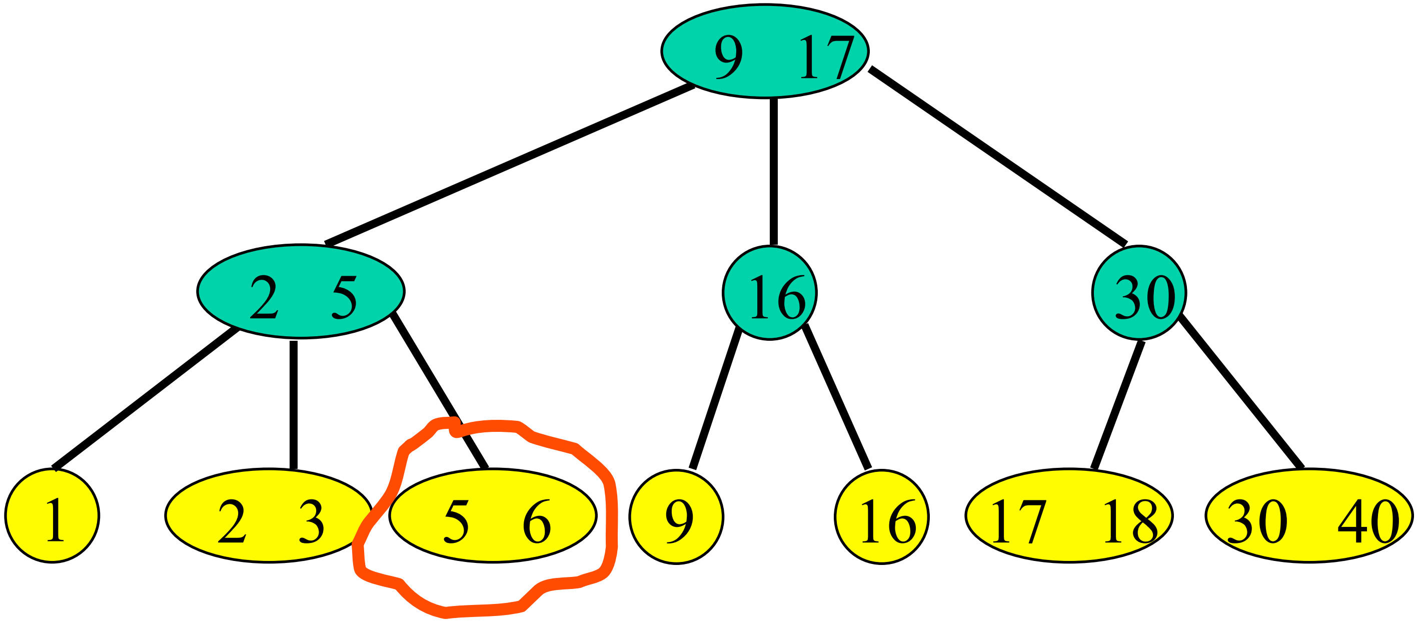
- Now, insert a pair with key = 18.
- Insert an index entry 17 plus a pointer into parent.

Insert



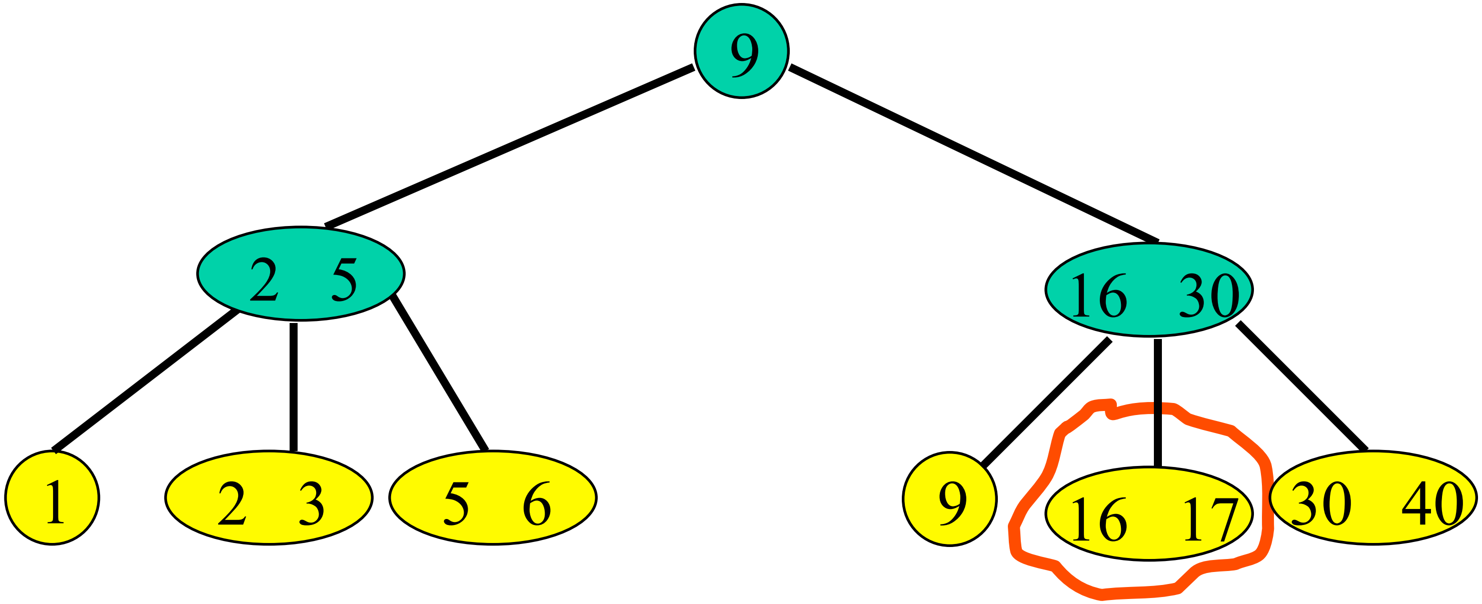
- Now, insert a pair with key = 18.
- Insert an index entry17 plus a pointer into parent.

Insert



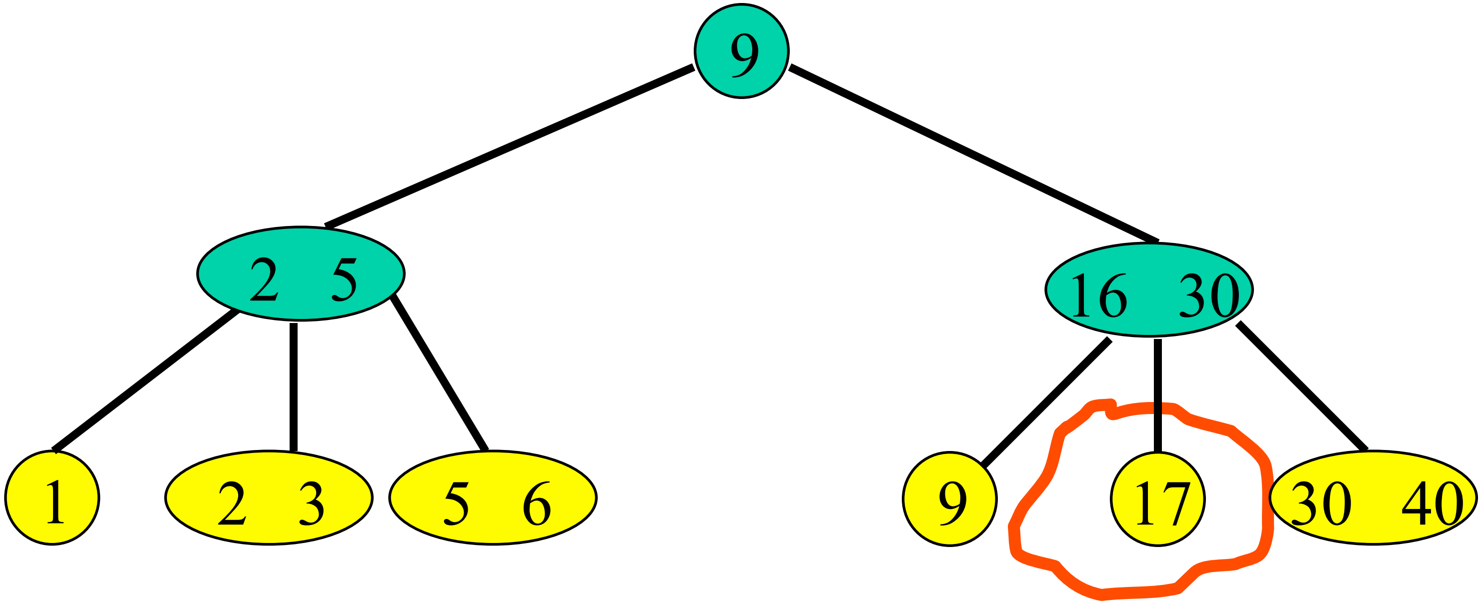
- Now, insert a pair with key = 7.

Delete



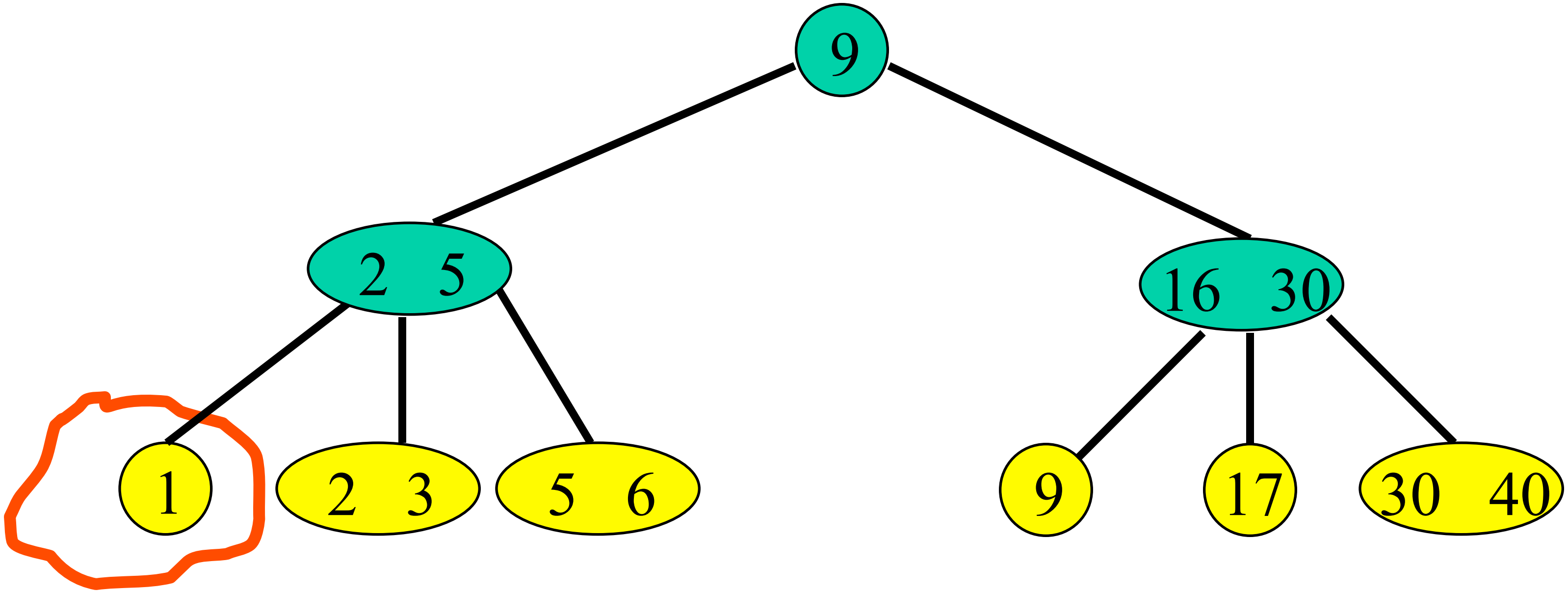
- Delete pair with key = 16.
- Note: delete pair is always in a leaf.

Delete



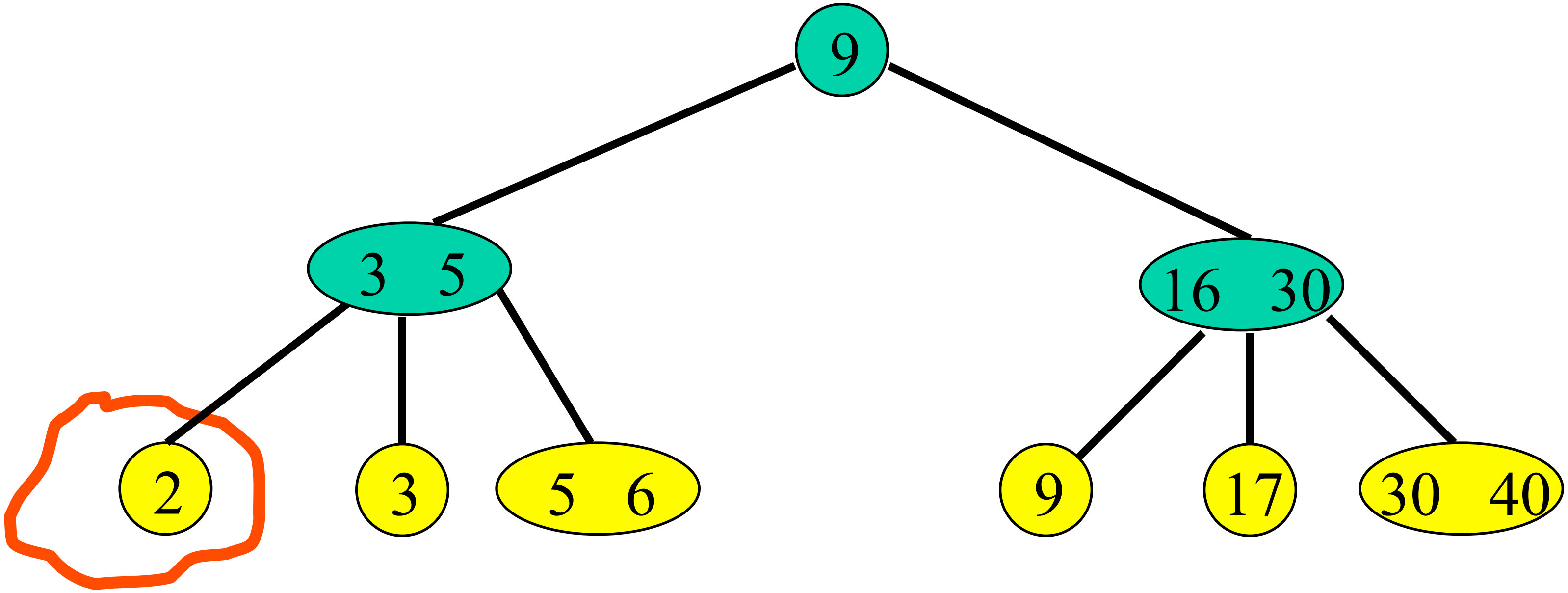
- Delete pair with key = 16.
- Note: delete pair is always in a leaf.

Delete



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

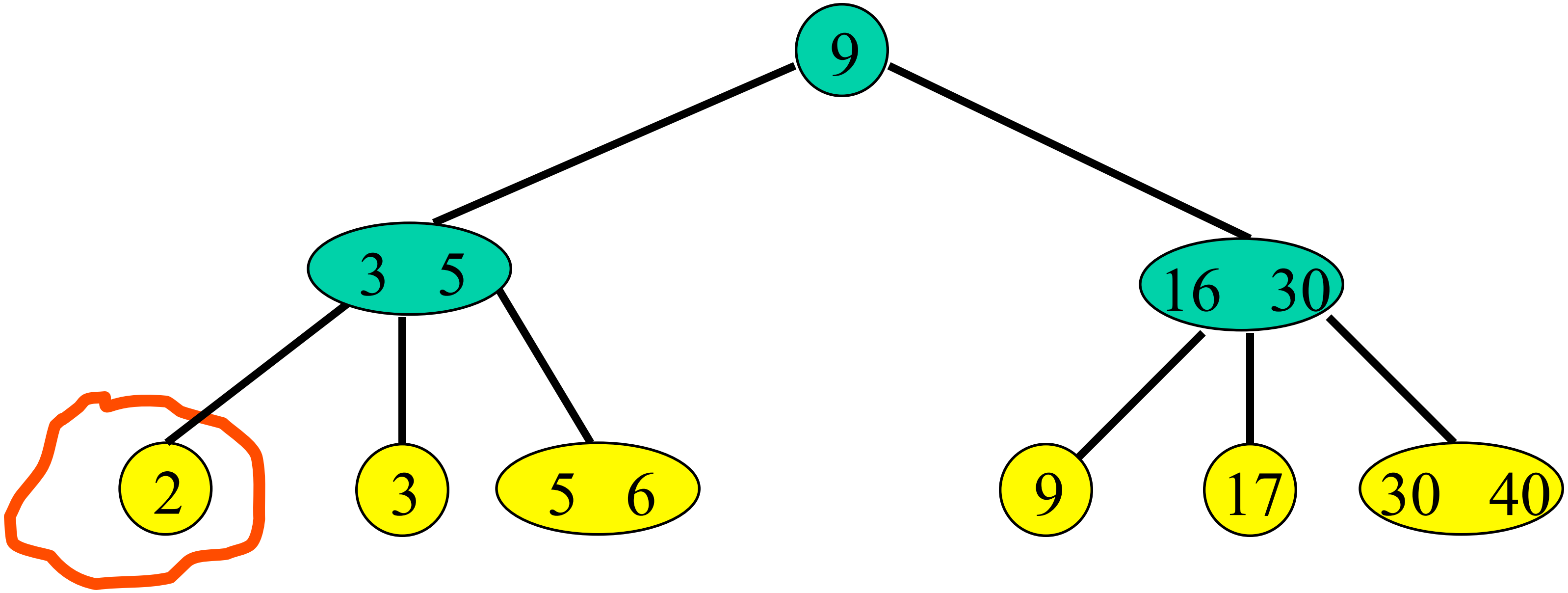
Delete



- Delete pair with key = 1.

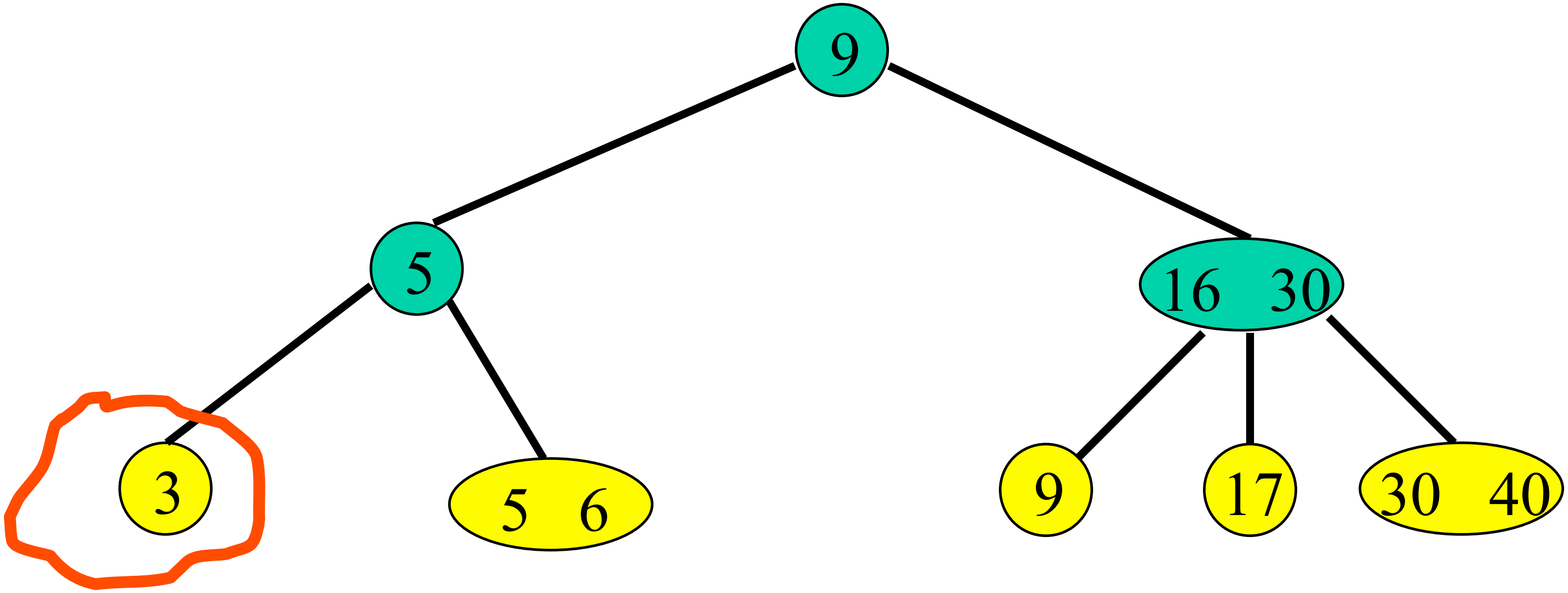
- Get >= 1 from sibling and update parent key.

Delete



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

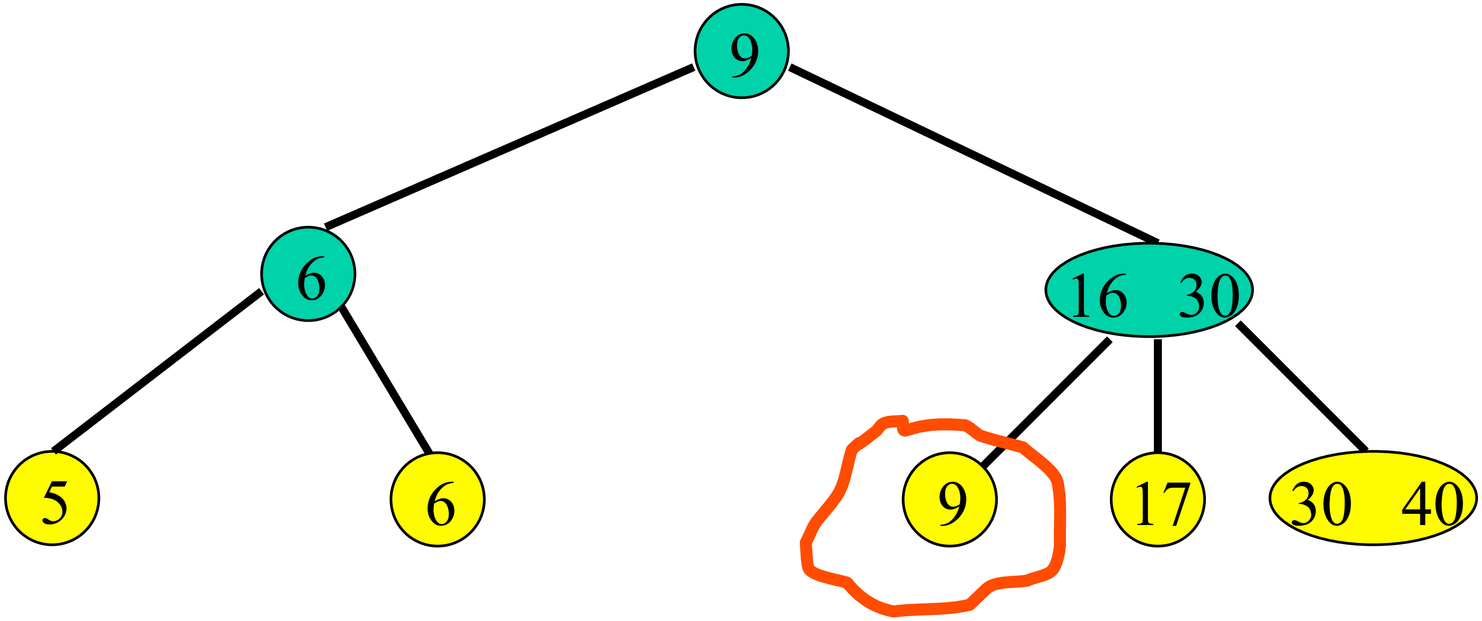
Delete



- Delete pair with key = 3.

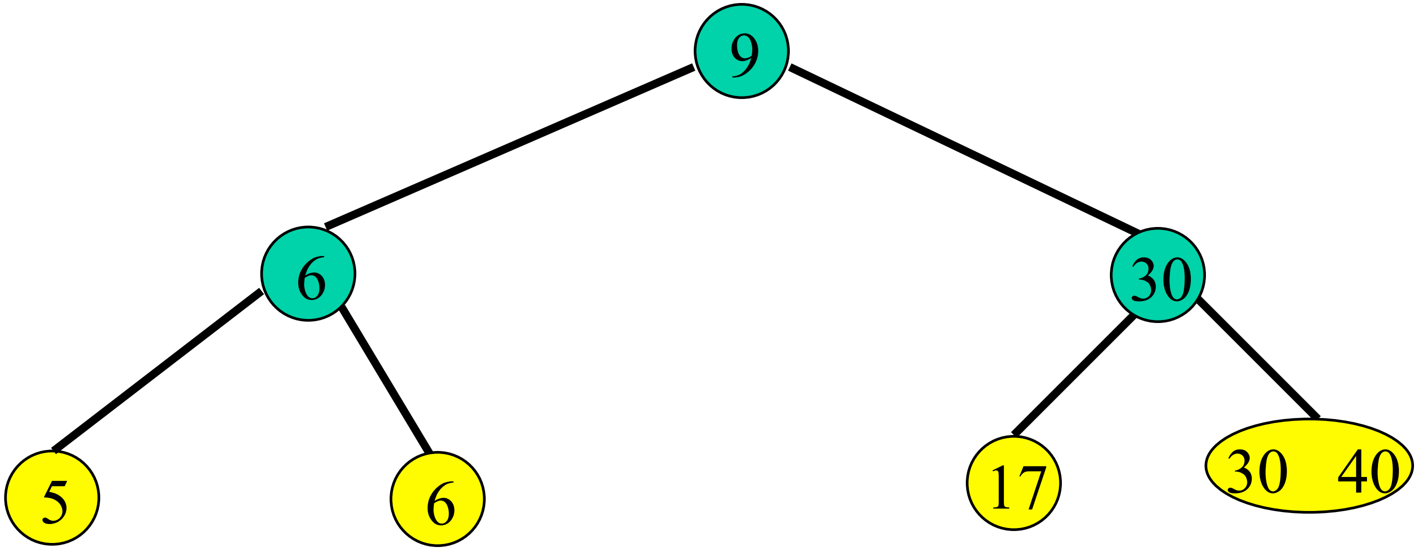
•Get >= 1 from sibling and update parent key.

Delete

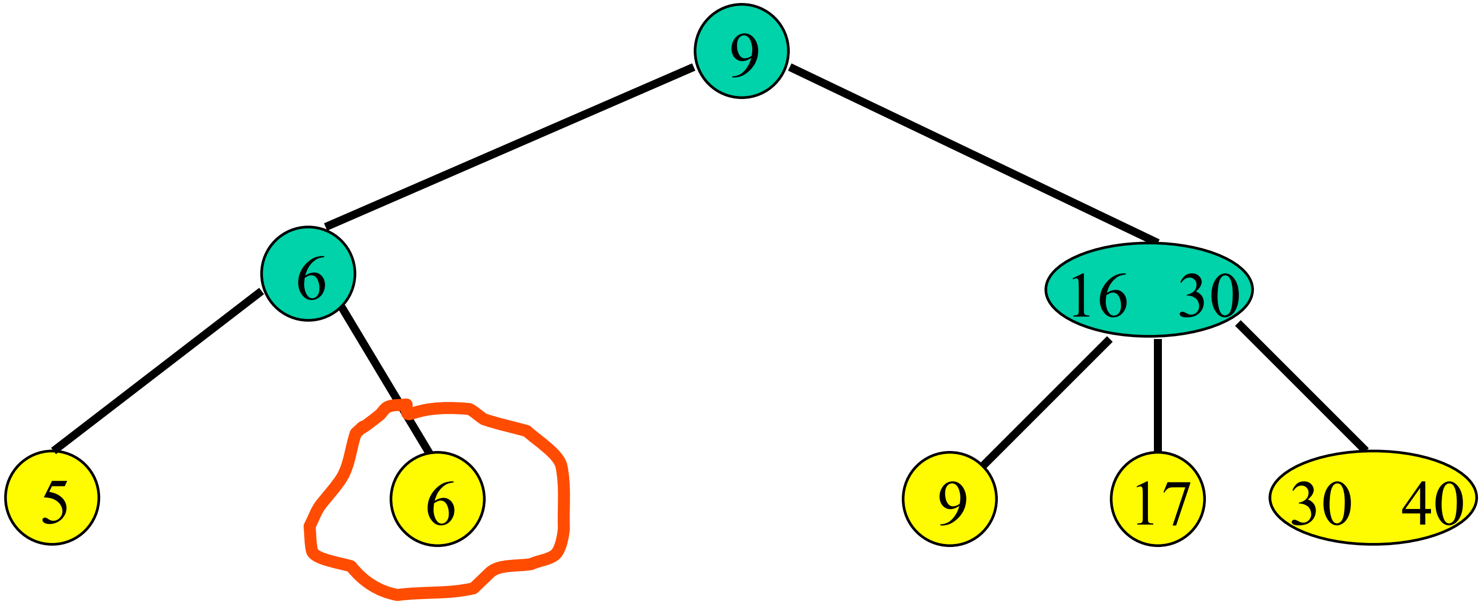


- Delete pair with key = 9.
- Merge with sibling, delete in-between key in parent.

Delete



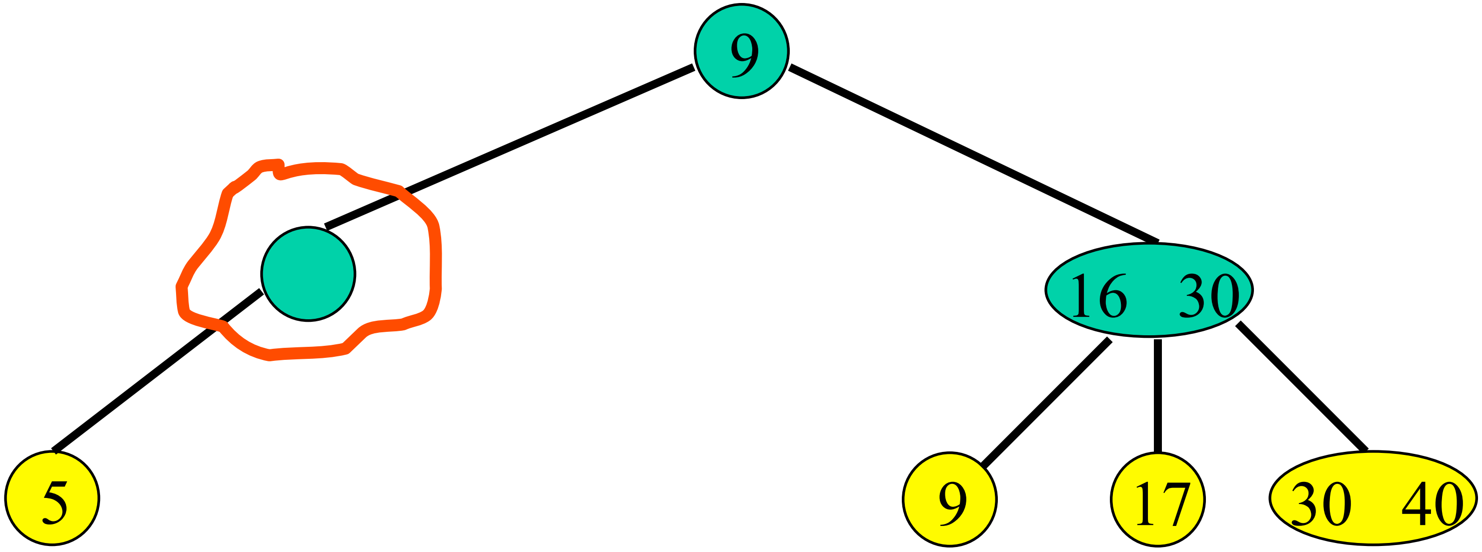
Delete



- Delete pair with key = 6.

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

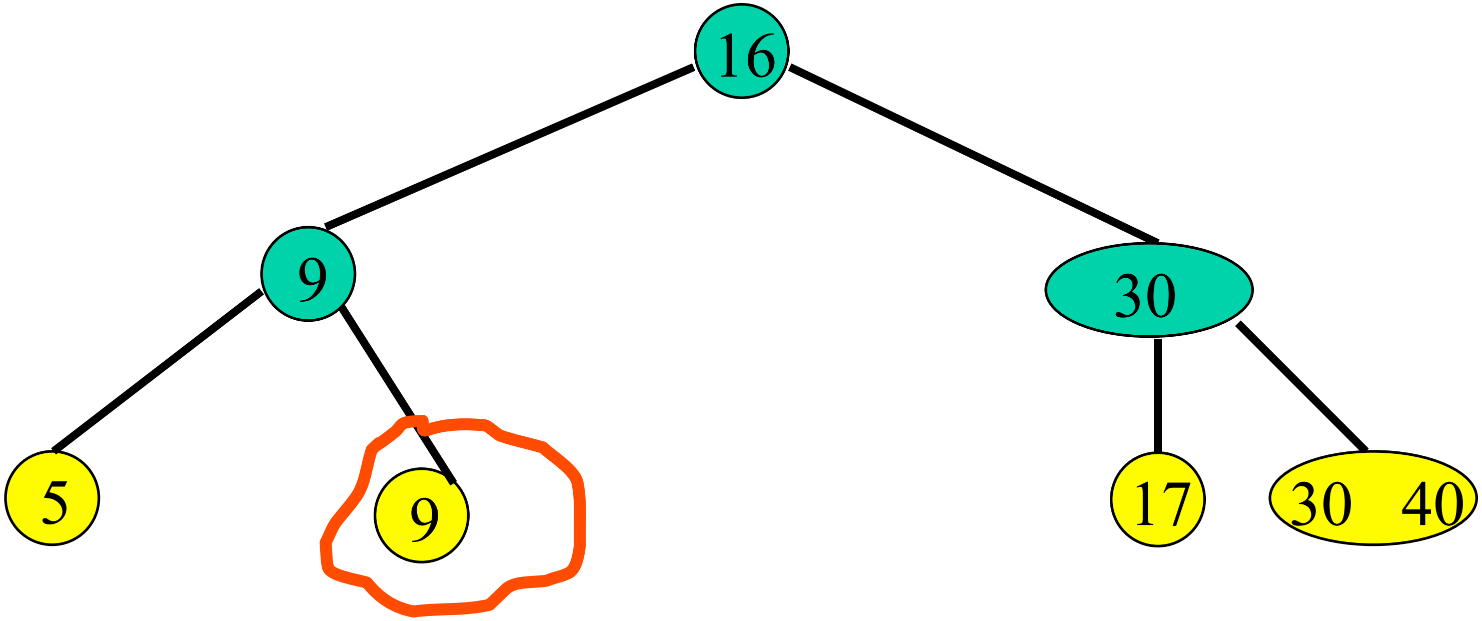
Delete



- Index node becomes deficient.

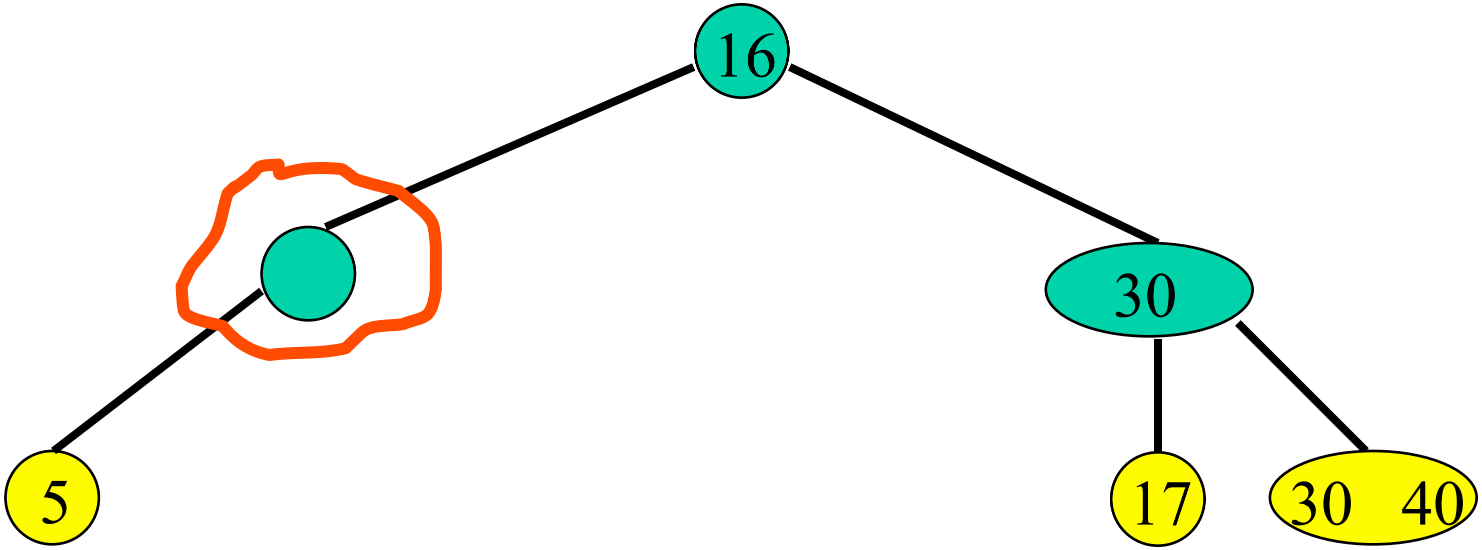
•Get >= 1 from sibling, move last one to parent, get parent key.

Delete



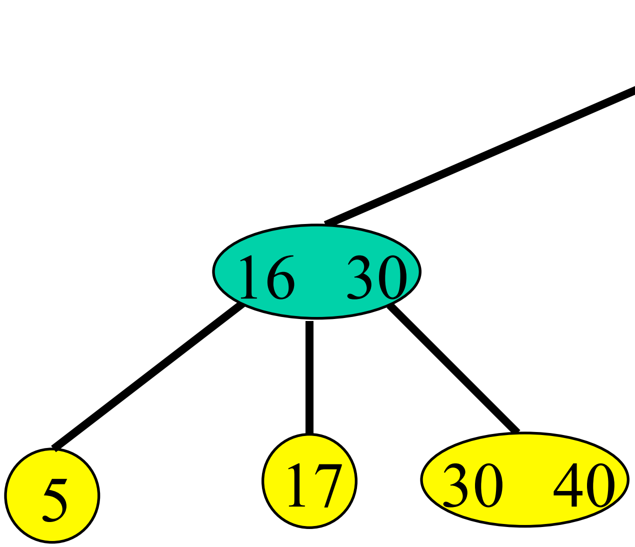
- Delete 9.
- Merge with sibling, delete in-between key in parent.

Delete



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

Delete



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

Key Sections

- 1.1 Overview: System Life Cycle
- 1.3 Data Abstraction and Data encapsulation
- 1.5 Algorithm Specification
- 1.7 Performance Analysis and Measurement
- 2.2 Array as an Abstract Data Type
- 2.3 The Polynomial Abstract Data Type
- 4.2 Representation Chains in C++

Key Sections

- 5.2 Binary Trees
- 5.3.1-6 Binary Tree Traversal
- 5.6 Heaps
- 5.7.1-4 Binary Search Trees
- 5.9 Transforming a Forest into a Binary Tree
- 5.10 Representation of Disjoint Sets

Key Sections

- 6.1.2 Definitions(Graph)
- 6.1.3 Graph Representations
- 6.2.1-2 Elementary Graph Operations
- 6.5 Activity Networks
- 7.2 Insertion Sort
- 7.3 Quick Sort
- 7.5.1-2 Iterative Merge Sort

Key Sections

- 7.6 Heap Sort
- 7.10.2 k-way Merging
- 8.2.1,2,4 Static Hashing
- 10.2 AVL Trees
- 11.1 m-way Search Trees
- 11.2.1 Definition and Properties(B-Trees)