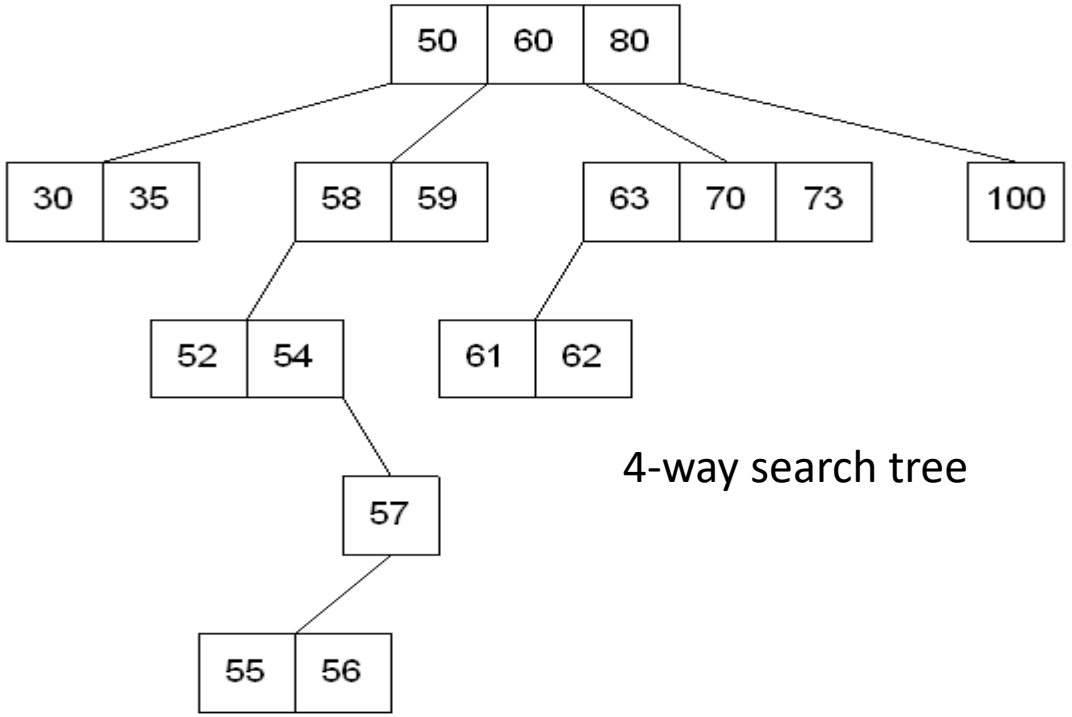
Multiway search trees

Multiway Trees

- A **multiway tree** is a tree that can have more than two children. A **multiway tree of order m** (or an **m-way tree**) is one in which a tree can have m children.
- As with the other trees that have been studied, the nodes in an **m**-way tree will be made up of key fields, in this case **m-1** key fields, and pointers to children.
- To make the processing of m-way trees easier some type of order will be imposed on the keys within each node, resulting in a **multiway search tree of order m** (or an **m-way search tree**).
- By definition an m-way search tree is a m-way tree in which:
  - Each node has m children and m-1 key fields
  - The keys in each node are in ascending order.
  - The keys in the first i children are smaller than the ith key
  - The keys in the last m-i children are larger than the ith key



4-way search tree

- M-way search trees give the same advantages to m-way trees that binary search trees gave to binary trees - they provide fast information retrieval and update.
- However, they also have the same problems that binary search trees had - they can become unbalanced, which means that the construction of the tree becomes of vital importance.

m-way Search Trees

Definition: An *m*-way search tree, either is empty or satisfies the following properties:

(1) The root has at most *m* subtrees and has the following structures:

$$n, A_0, (K_1, A_1), (K_2, A_2), \ldots, (K_n, A_n)$$

where each A_i, $0 \le i \le n \le m$, is a pointer to a subtree, and each K_i, $1 \le i \le n \le m$, is a key value.

(2) $K_i < K_{i+1}$, $1 \le i \le n\text{-}1$

(3) Let $K_0 = -\infty$ and $K_{n+1} = \infty$. All key values in the subtree A_i are less than K_{i+1} and greater than K_i, $0 \le i \le n$

(4) The subtrees A_i, $0 \le i \le n$, are also *m*-way search trees.

B-tree

- **It can have multiple keys for a single node.** A single node can have more than two children.
- **All the leaf nodes must be at the same level.** Leaf node means a node that doesn't have any child nodes.
- Suppose the order of a B-tree is N. It means every node can have a maximum of N children. Therefore, every node can have **maximum (N-1) keys and minimum {(N/2)-1} keys** except the root node.
  - Let's take a B-tree of order N = 6. According to the properties of the B-tree, any node can have a maximum of (N-1) keys. Hence 5 keys in this case. Furthermore, any node can have minimum {(N/2)-1} keys. Hence, {(6/2) -1} = 2 keys.

B-tree Operations

- Three primary operations can be performed on a B-tree:
  - searching
  - insertion
  - deletion.

While inserting any element, we must remember that we can only insert values in ascending order.
Moreover, we always insert elements at the leaf node so that the B-tree always grows in the upward direction.

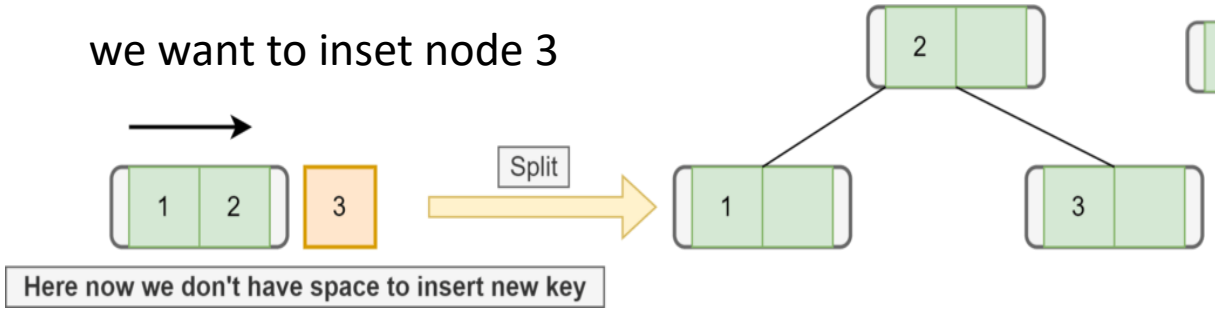
Let's take the example of inserting some nodes in an empty B-tree. Here, we want to insert 10 nodes in an empty B-tree: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10).

Let's assume the order of the B-tree is 3.

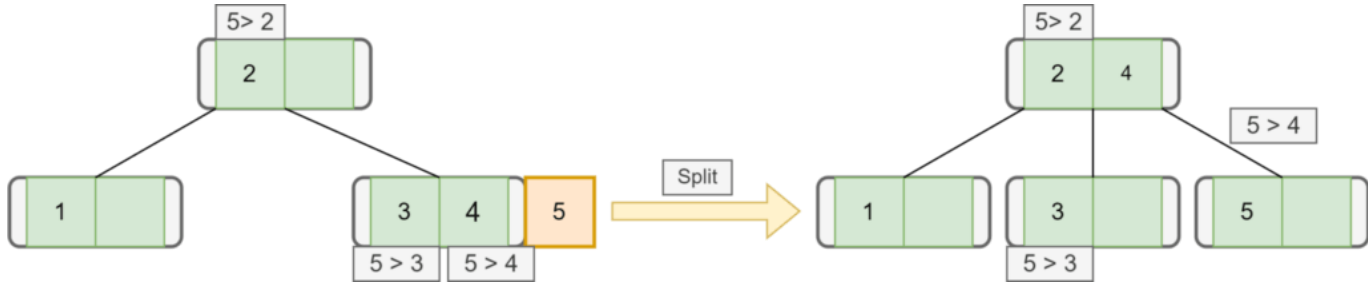
Hence, the maximum number of children a node can contain is 3. Additionally, the maximum and the minimum number of keys for a node, in this case, would be 2 and 1 (rounded off).

Let's start the insertion process with nodes 1 and 2:

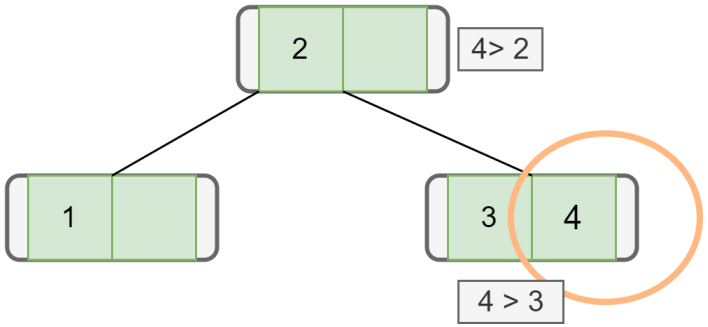
we want to inset node 3



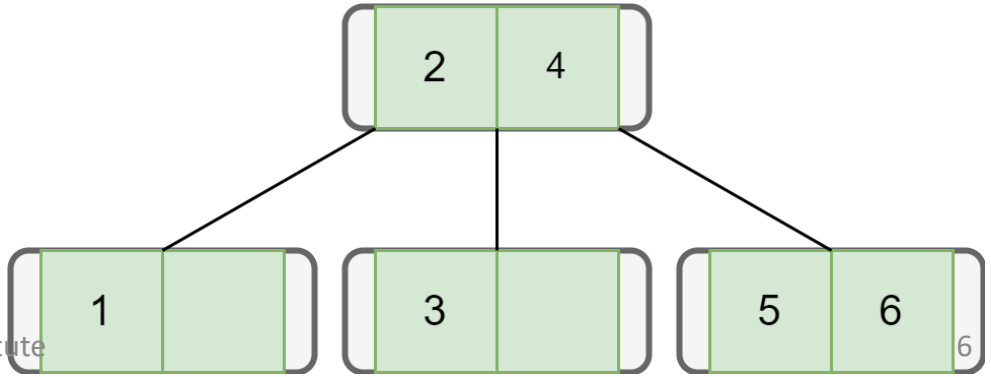
Let's insert the next node with key 5.



Now we want to insert the next element, which is 4.



Similarly, we insert our next node with key 6:

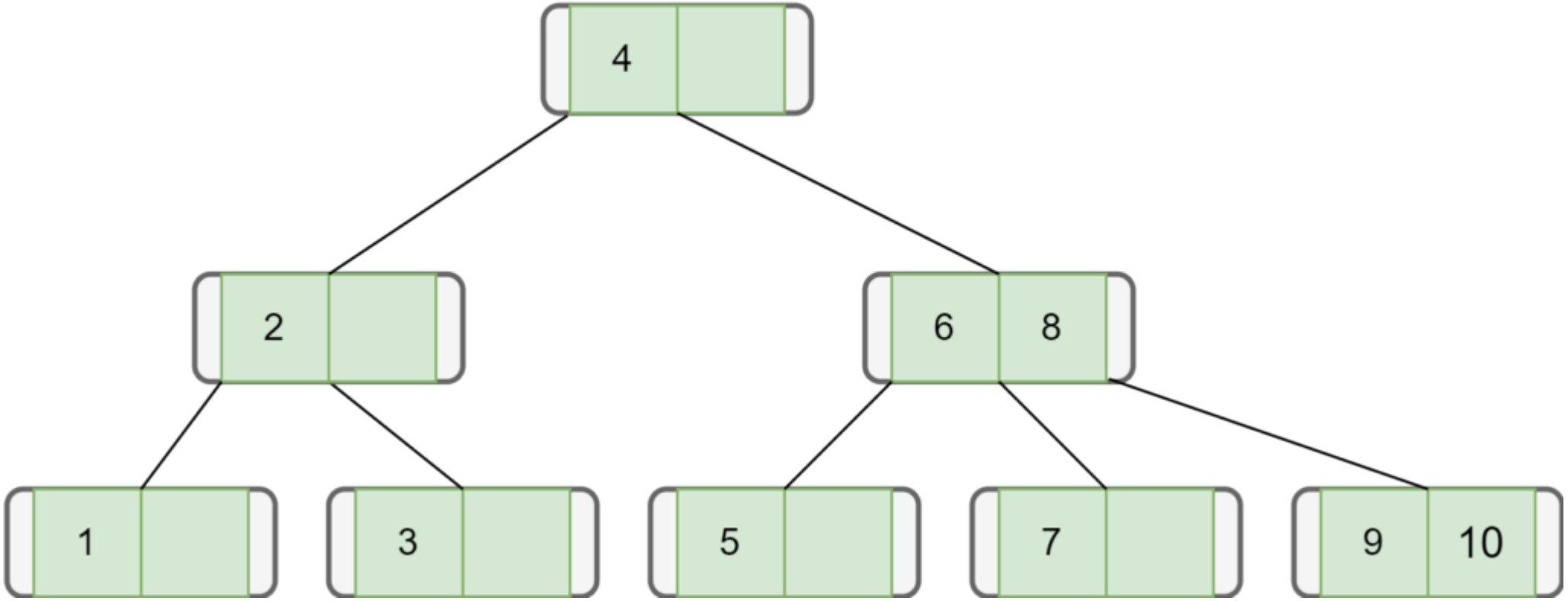


Let's take the example of inserting some nodes in an empty B-tree. Here, we want to insert 10 nodes in an empty B-tree: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10).

Let's assume the order of the B-tree is 3.

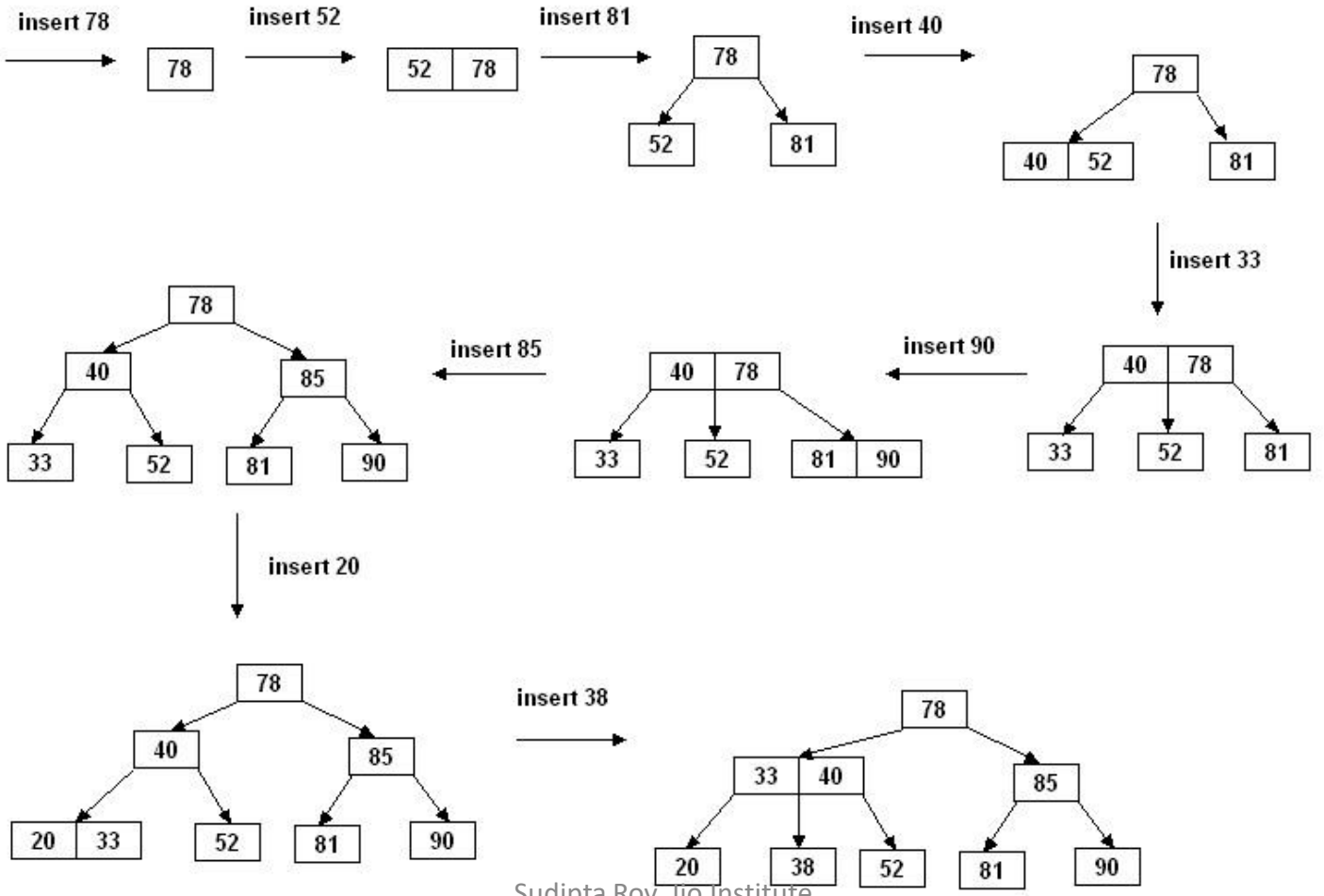
Hence, the maximum number of children a node can contain is 3. Additionally, the maximum and the minimum number of keys for a node, in this case, would be 2 and 1 (rounded off).

Finally, let's add all the remaining nodes one by one following the properties of the B-tree:



Example Insertion in B-Trees

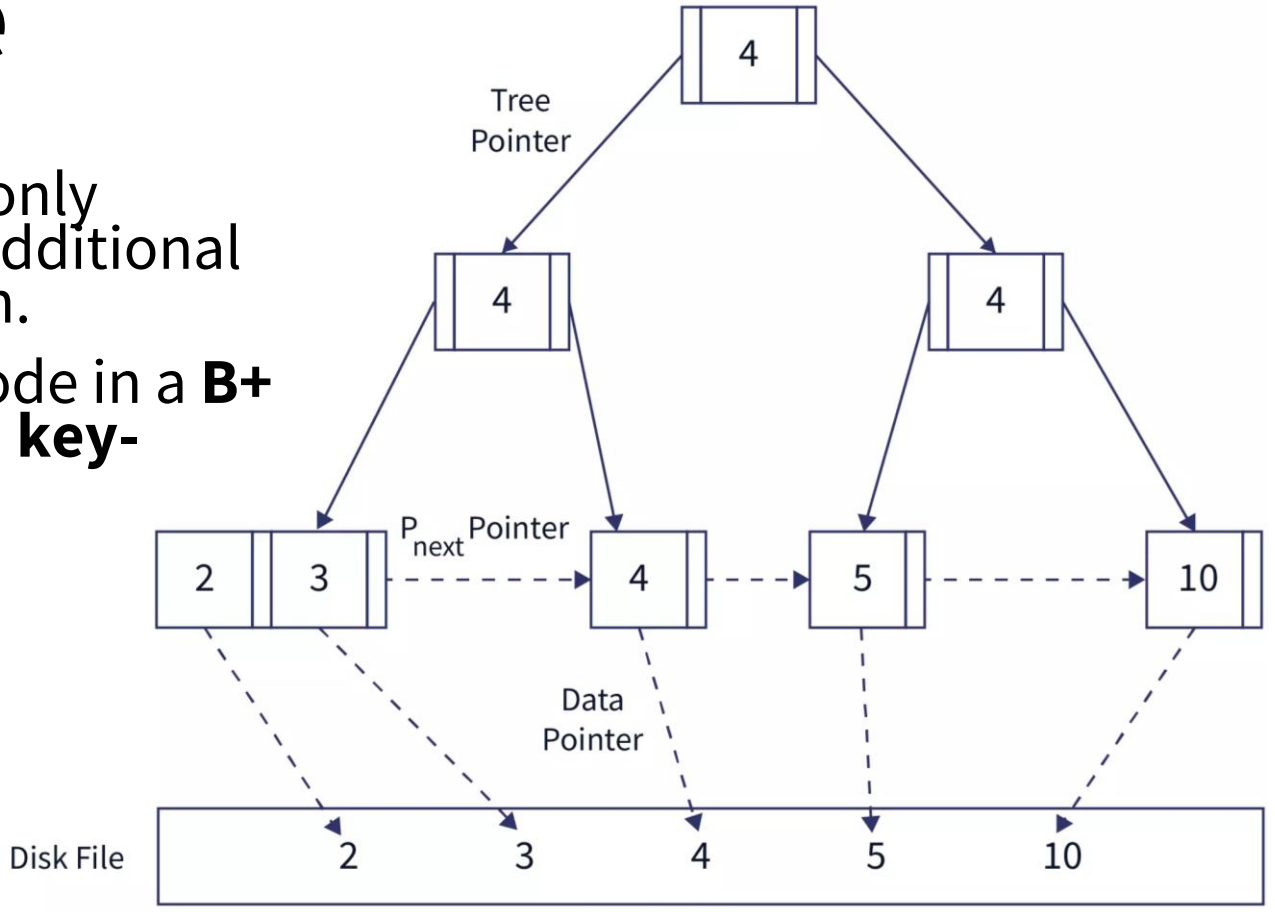
- **Insertion in a B-tree of odd order**
- Example: Insert the keys 78, 52, 81, 40, 33, 90, 85, 20, and 38 in this order in an initially empty B-tree of order 3



The time complexity of the search process of a B-tree is O(logn).

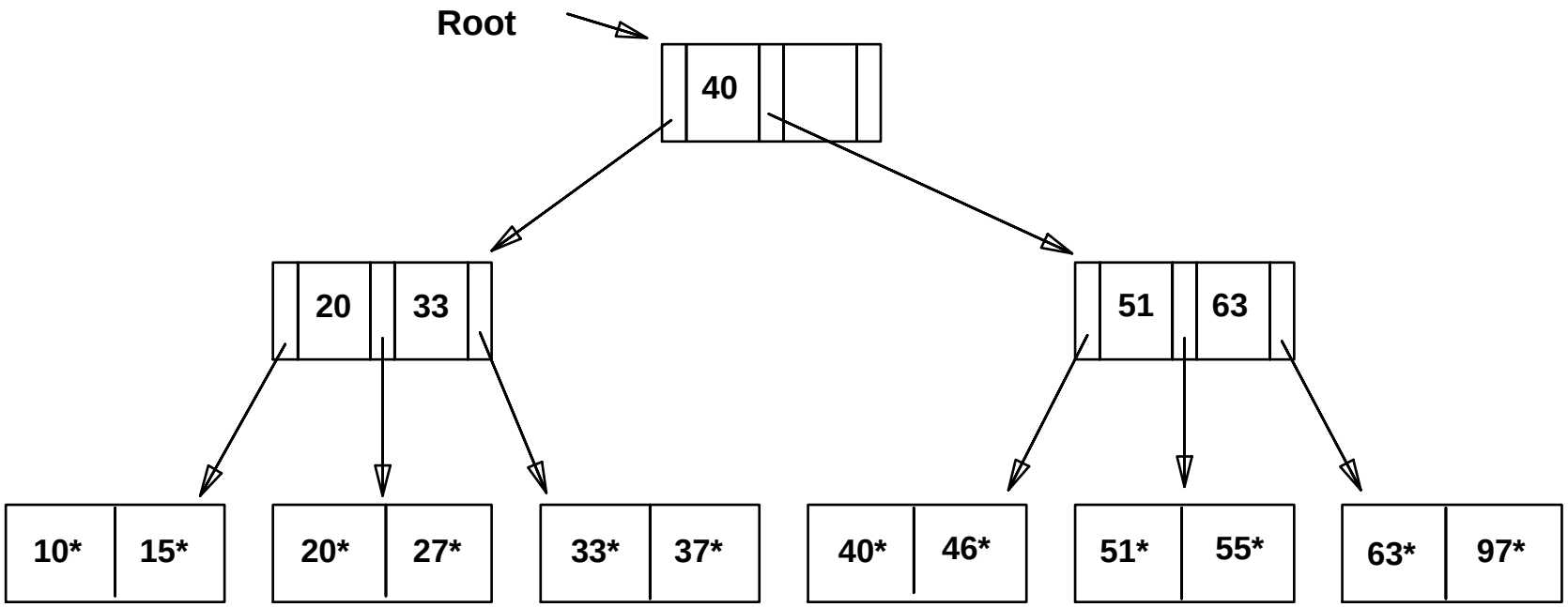
Structure of B+ Tree

- A B+ tree is the same as a B tree; the only difference is that the B+ tree has an additional level with linked leaves at the bottom.
- In addition, unlike the B tree, each node in a **B+ tree contains only keys rather than key-value pairs**.

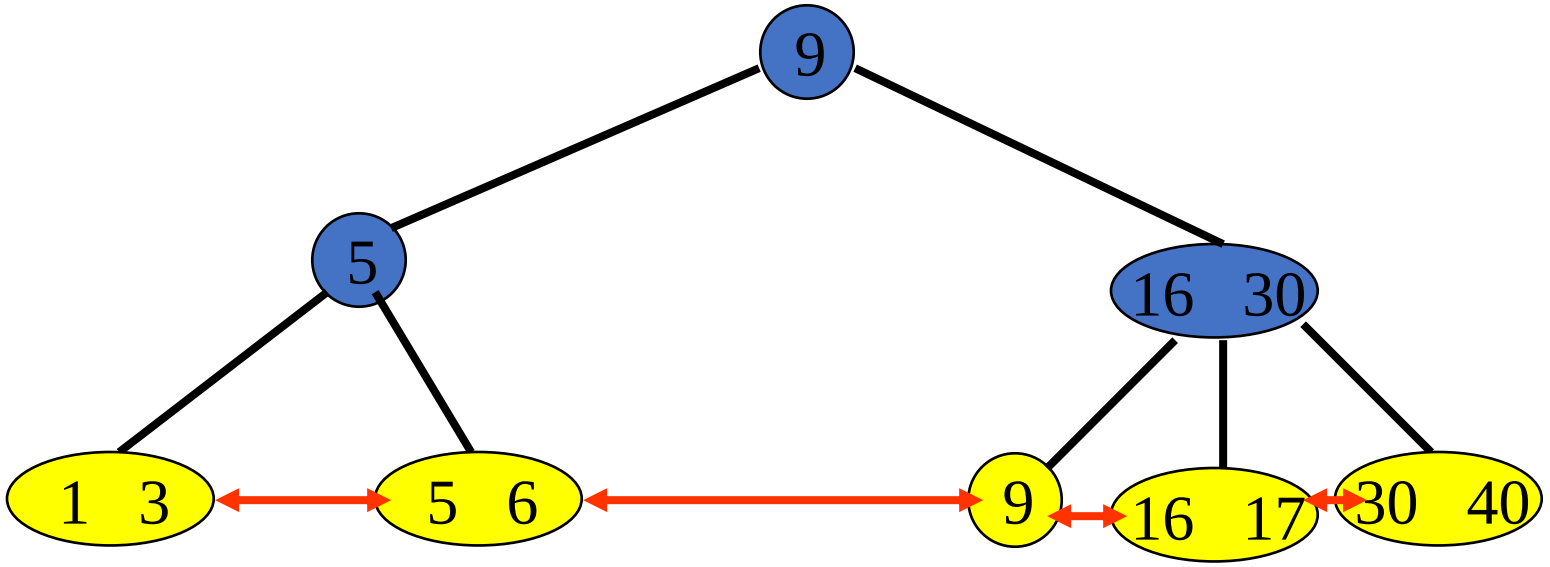


Example B+ Tree

Order=3



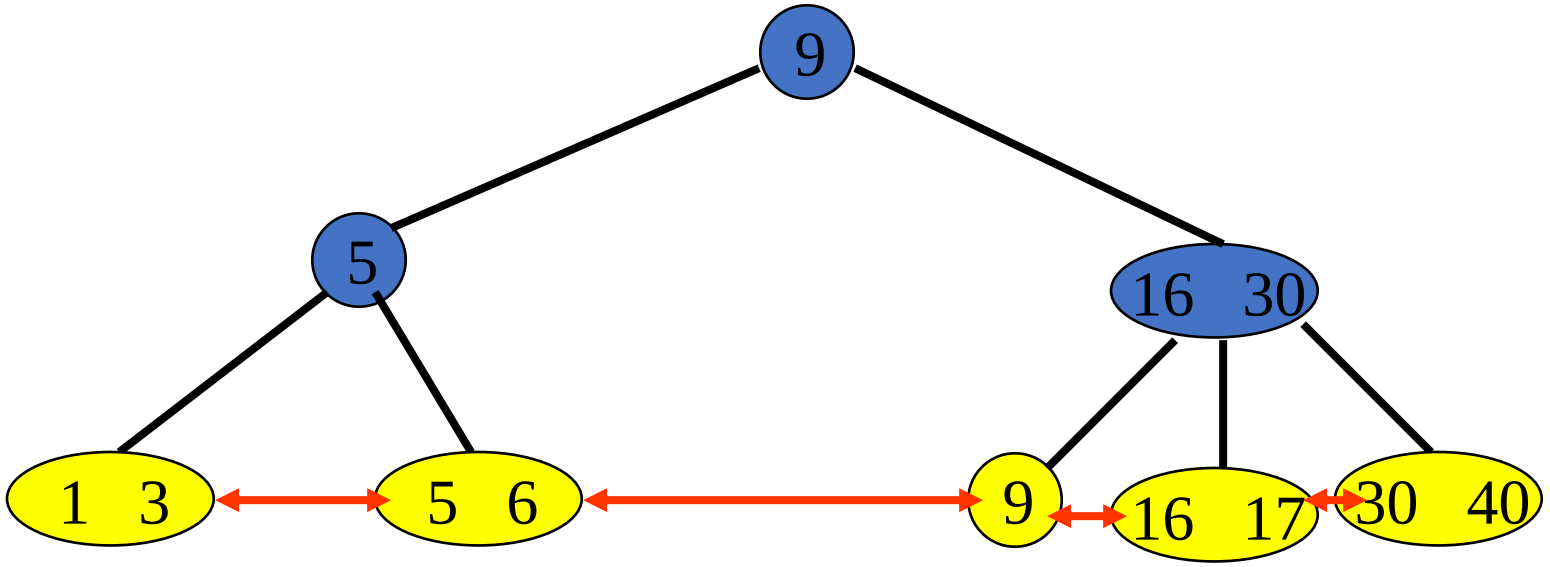
Example B+-tree



→ index node

→ leaf/data node

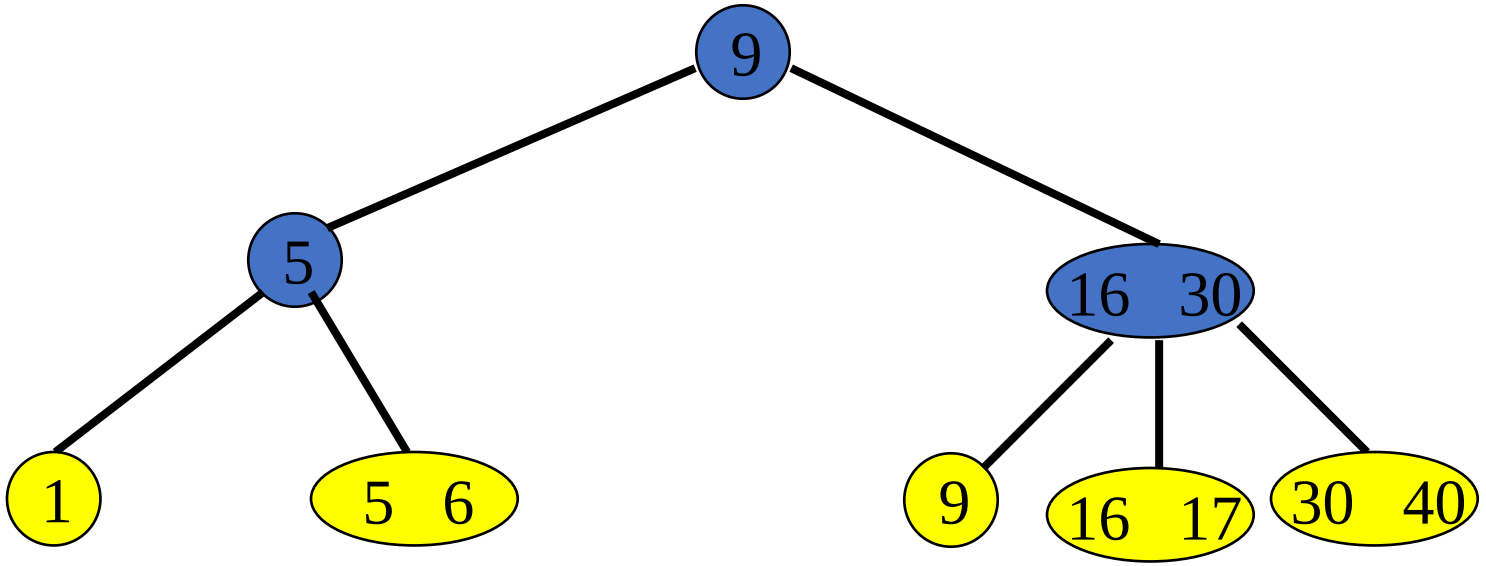
B+-tree—Search



key = 5

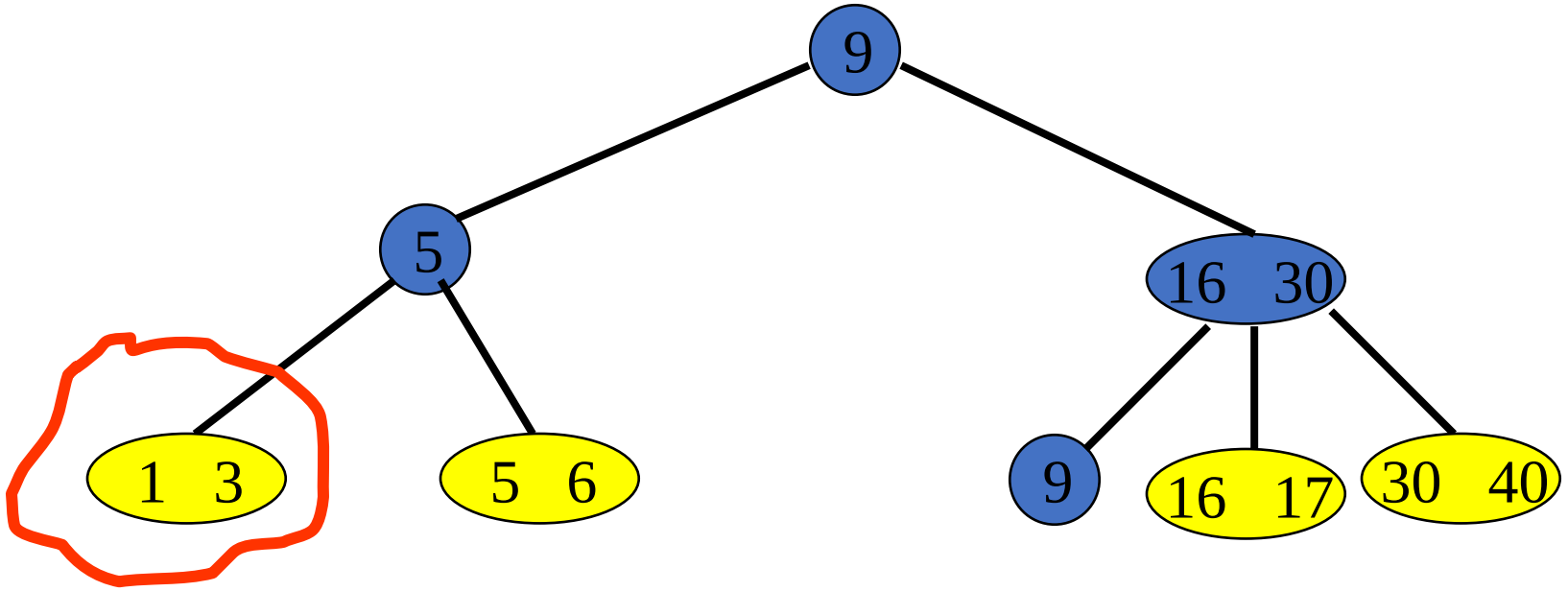
3 <= key <= 6

B+-tree—Insert



Insert 10

Insert

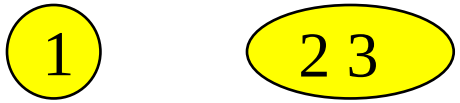


- Insert a pair with key = 2.
- New pair goes into a 3-node.

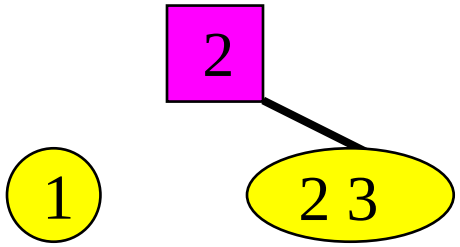
Insert Into A 3-node

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

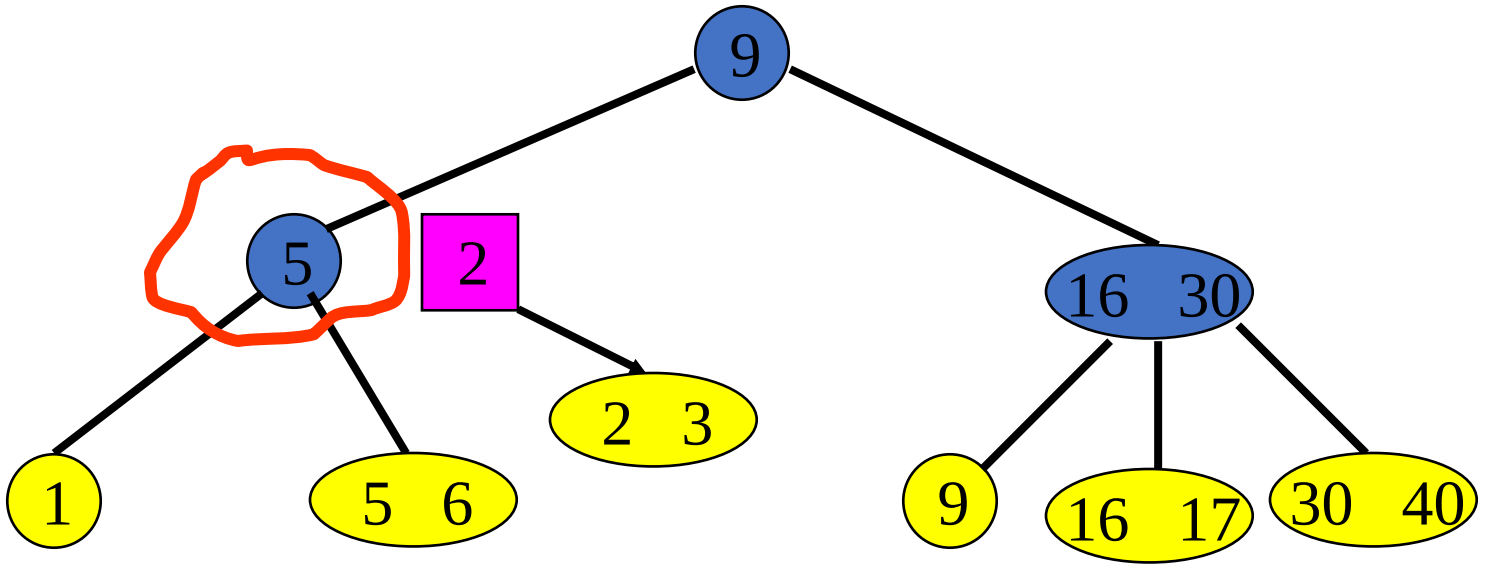
- Split into two nodes.



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

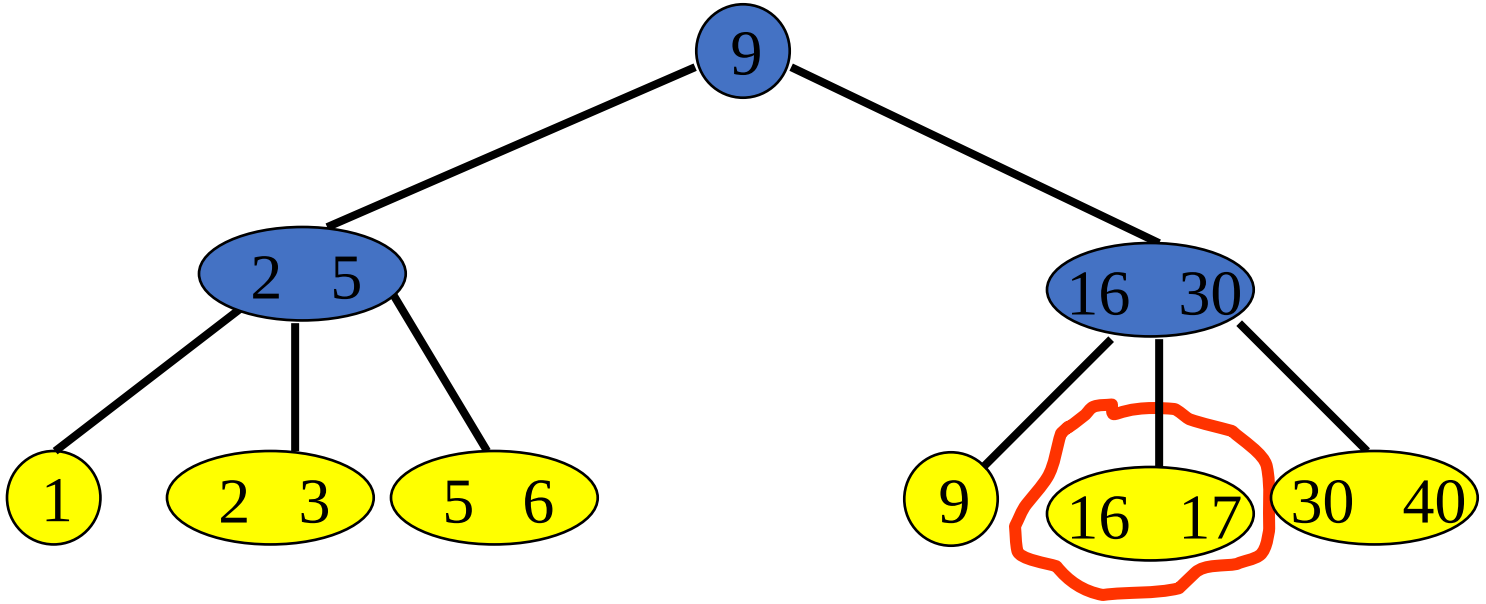


Insert



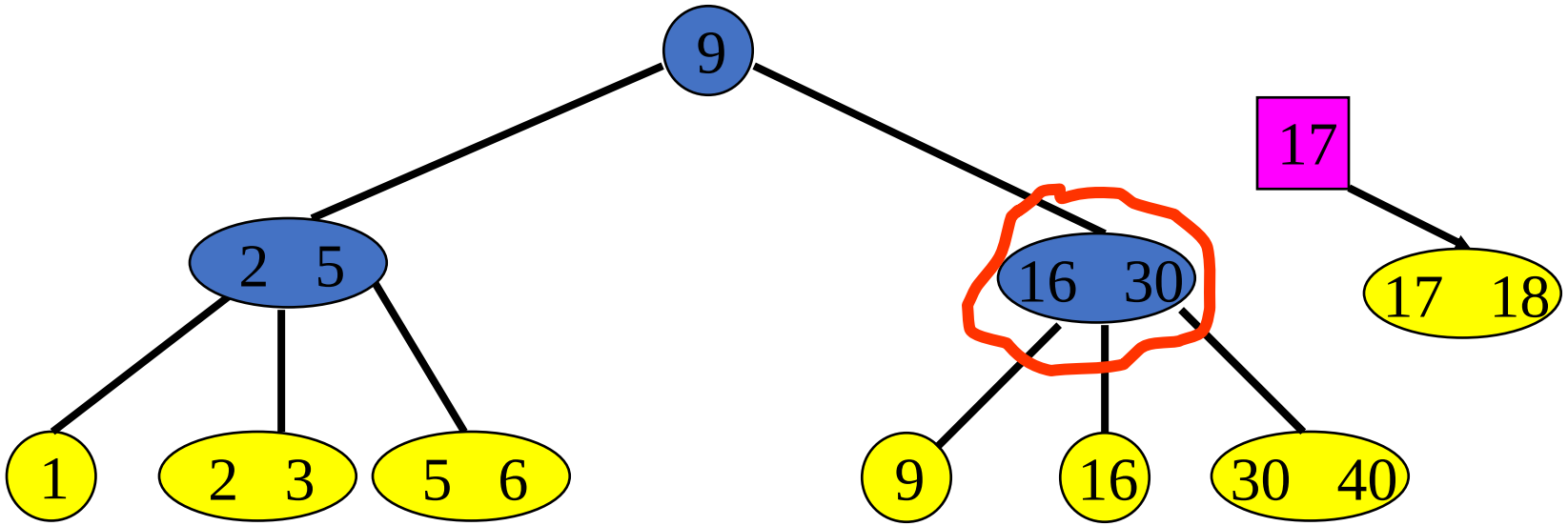
- Insert an index entry 2 plus a pointer into parent.

Insert



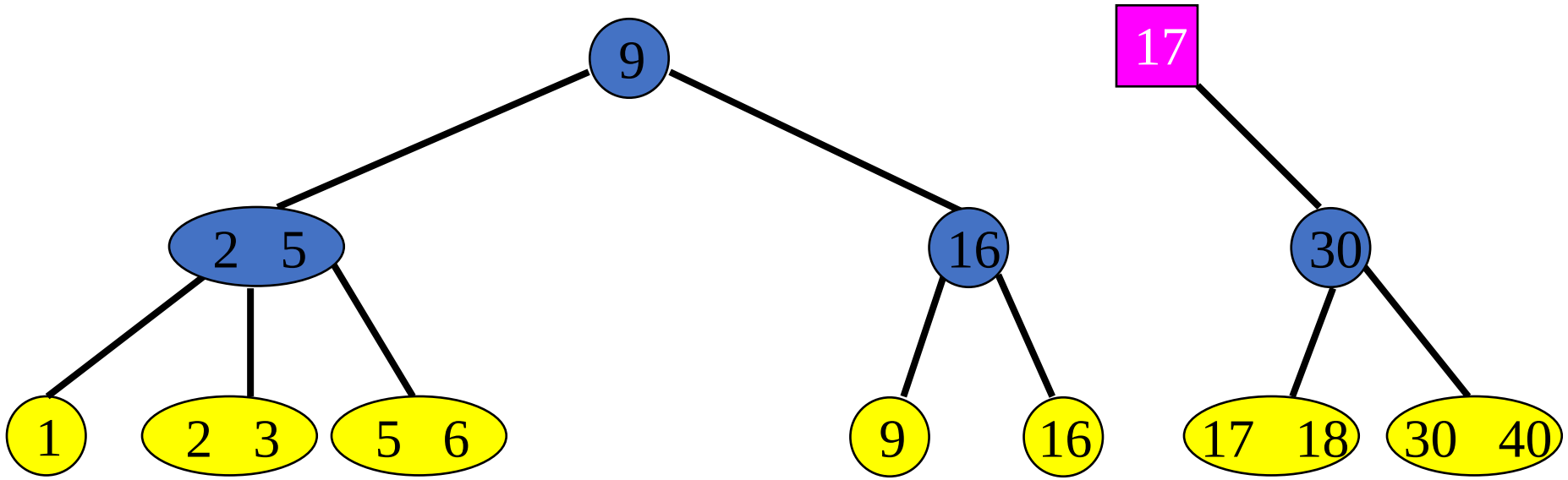
- Now, insert a pair with key = 18.

Insert



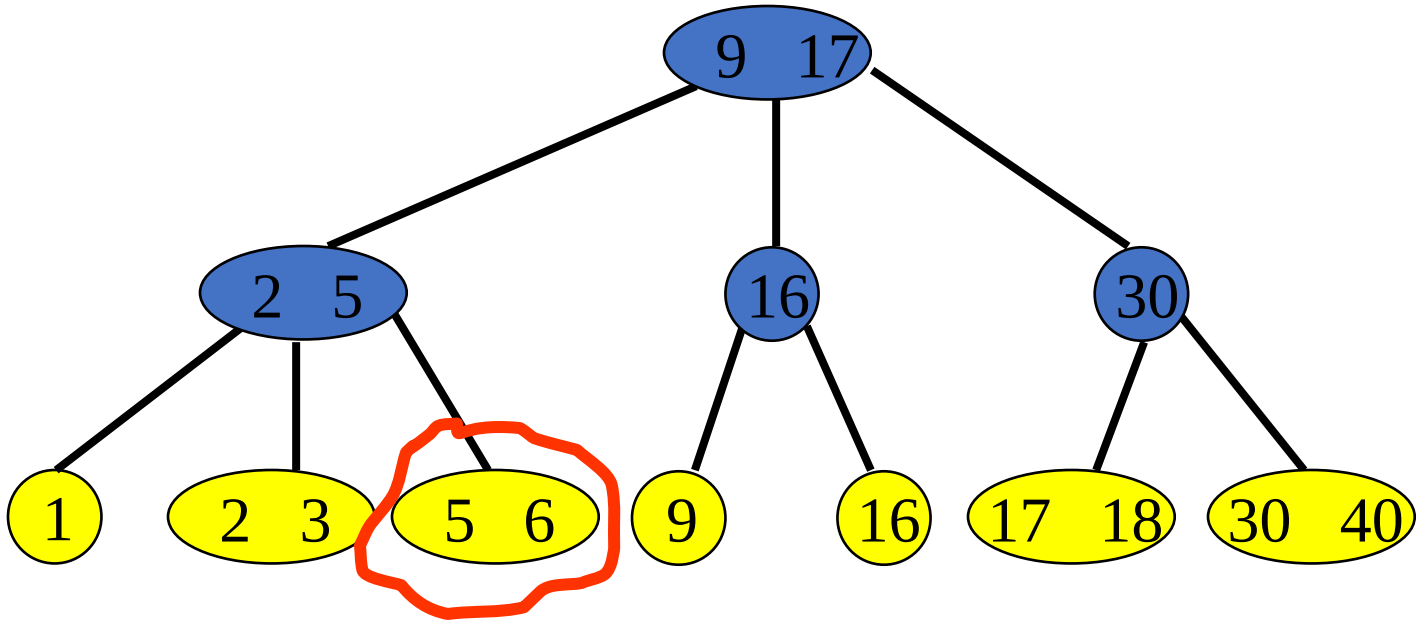
- Now, insert a pair with key = 18.
- Insert an index entry17 plus a pointer into parent.

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.

S No.	B Tree	B+ Tree
1	All keys and records in the B tree are stored in both internal and leaf nodes.	The keys in the B+ tree are the indexes stored in the internal nodes, while the all records are stored in the leaf nodes.
2	Keys in the B tree cannot be stored repeatedly, so there is no key or record duplication.	The occurrence of the keys in the B+ tree may be redundant. In this case, the records are stored in the leaf nodes, while the keys are stored in the internal nodes, which means that redundant keys may **exist in the internal nodes**.
3	Leaf nodes in the B tree are not connected to one another.	The leaf nodes in a B+ tree are linked to each other to provide sequential access.
4	Searching in a B tree is inefficient because records are stored in either leaf or internal nodes.	Because all records are stored in the leaf nodes of the B+ tree, searching is very **efficient or faster**.
5	Insertion takes longer and can be **unpredictable** at times.	Insertion is simpler, and the results are consistent.
7	Sequential access is not possible in the B tree.	All of the leaf nodes in the B+ tree are linked to each other via a pointer, allowing for sequential access.
8	The greater the number of splitting operations performed in a B tree, the greater the **height relative to width**.	The width of a B+ tree is greater than its height.
9	Because each node in the B tree has at least two branches and each node contains some records, we don't need to traverse to the leaf nodes to get the data.	Internal nodes in a B+ tree contain **only pointers**, whereas leaf nodes contain records. Because all of the leaf nodes are at the same level, we must traverse until we reach the leaf nodes to obtain the data.

K Dimensional tree

K Dimensional tree

- A **K-D Tree**(also called as **K-Dimensional Tree**) is a binary search tree where data in each node is a K-Dimensional point in space.
- In short, it is a space partitioning(details below) data structure for organizing points in a K-Dimensional space.
- **A non-leaf node in K-D tree divides the space into two parts, called as half-spaces**.
- **Points to the left of this space are represented by the left subtree of that node and points to the right of the space are represented by the right subtree**.
- One of the primary advantages of K-D trees is that **they allow for efficient k-nearest neighbor (KNN) queries**, which are useful in applications such as image recognition and recommendation systems.
- They can also be used for range queries, which allow for fast searches of points within a given radius or bounding box.
- We will soon be explaining the concept on how the space is divided and tree is formed.

Creation of a 2-D Tree: Consider following points in a 2-D plane: (3, 6), (17, 15), (13, 15), (6, 12), (9, 1), (2, 7), (10, 19)

Point (10, 19) will divide the space to the right of line X = 3 and above line Y = 15 into two parts. Draw line Y = 19 to the right of line X = 3 and above line Y = 15.