2-3 Search Tree

2-3 Search Trees

- Each node can contain 1 or 2 keys.

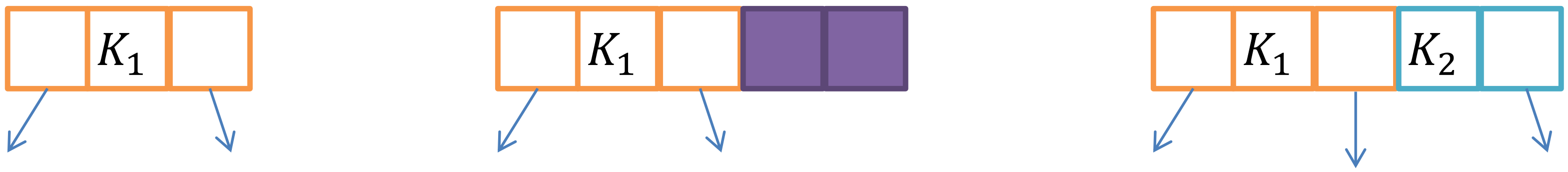
- Each node has 2 or 3 children, hence 2-3 trees.

- The keys in the nodes are ordered from **small to large**.

- All leaves are at the same (bottom most) level, meaning **<u>we always add at the bottom</u>**.

2-3 Search Tree Node

```
struct Node23{
  Node23 *left, *middle, *right;
  Key key1, key2;
};
```



2-node (not showing empty)

2-node (showing empty)

3-node

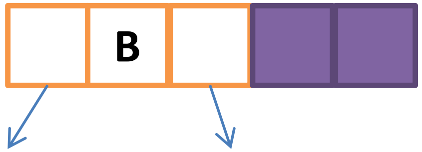
$$K_1 < K_2$$

Properties of 2-3 Search Trees

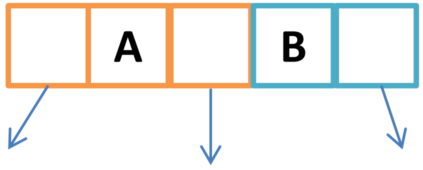
- 2-3 search trees guarantee to be balanced at all times.
- Searches are O(log N) in worst case.
- Balance is maintained during insertion
  - Splitting nodes, worst case O(log N), average case O(1)

Insertion

- If the node you insert is a 2-node, simply grow the node to a 3-node

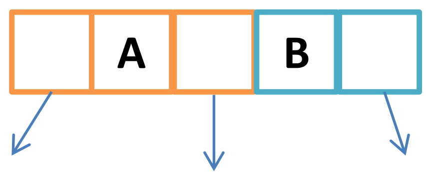


Inserting **'A'**

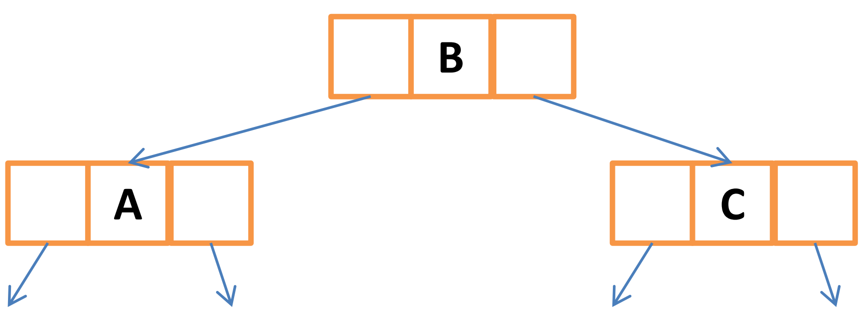


Insertion

- If the node you insert is a 3-node, we cannot grow the node more
  - We split it!



Inserting **'C'**

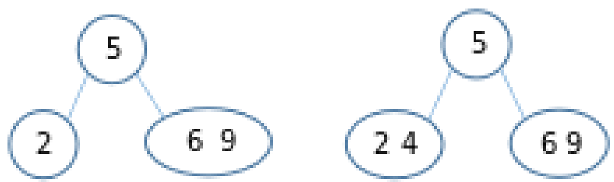


Insertion

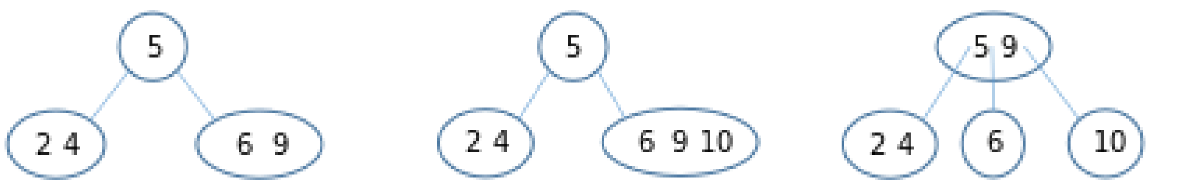
- Splitting this way is called bottom-up balancing
  - Insert the node at the bottom-most level at correct location.
  - If the node is a 3-node, split it and pass the middle key to the parent.
    - If the parent is also a 3-node, split the parent and pass the middle key up
      - Etc…
  - Eventually, the root will also be a 3-node and splitting it will grow the tree one level.

Insertion - Cases

Insert in a 2-node :



Insert in a 3-node (2 node parent) :

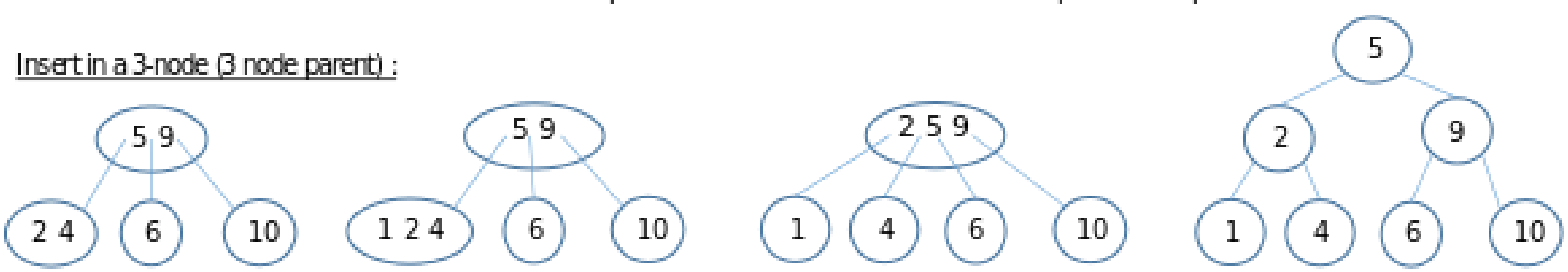


initial

Temp4node

Move middle to parent and split

Insert in a 3-node (3 node parent) :



initial

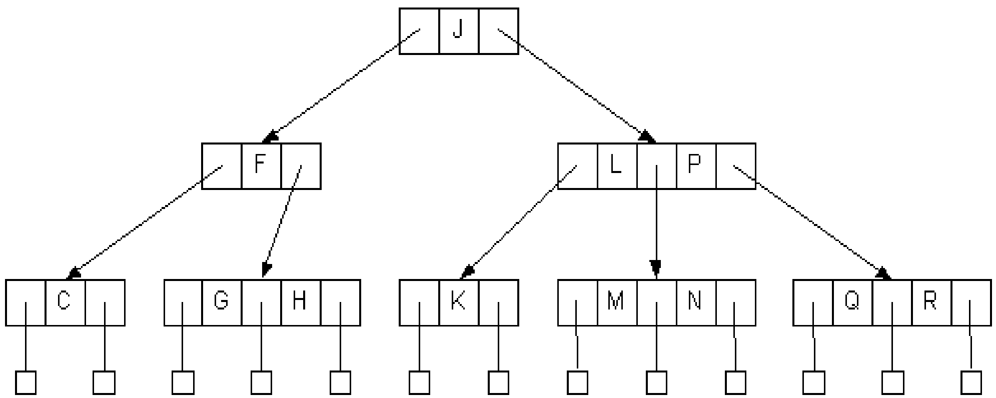
Temp4node

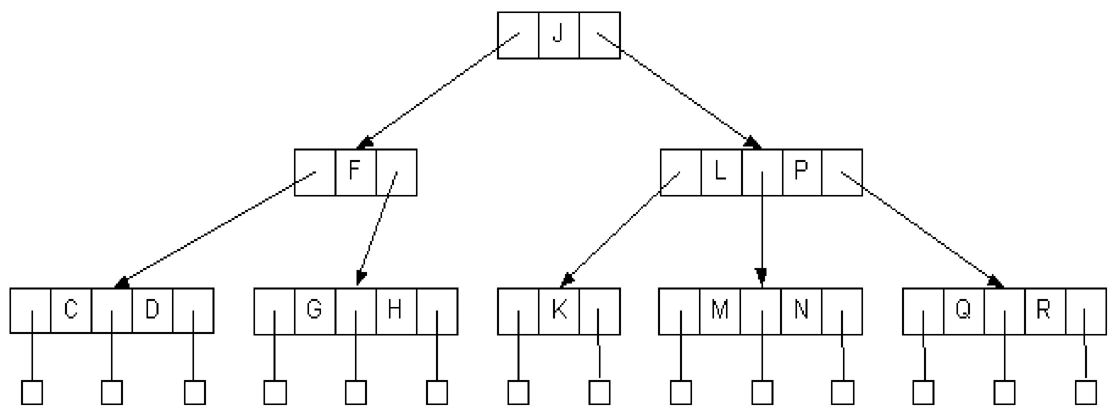
Move middle to parent and split

Move middle to parent and split

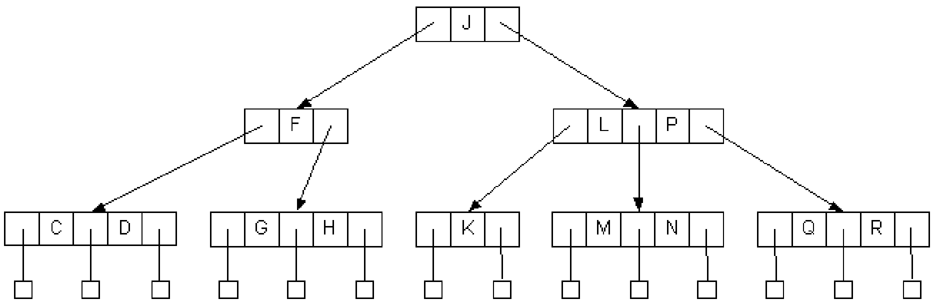
Example

Insert D

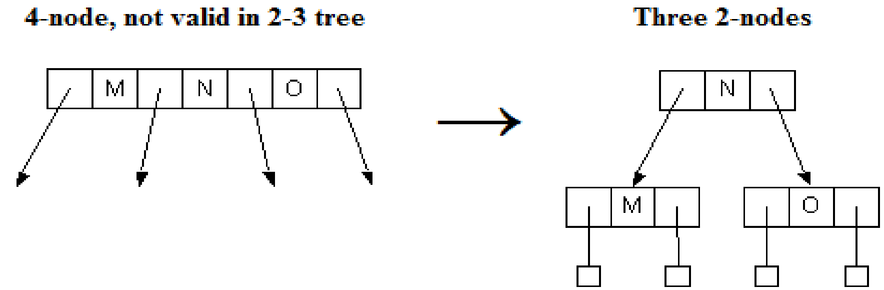




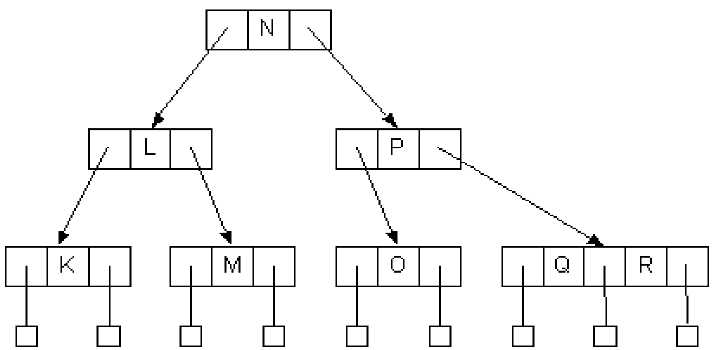
Insert O



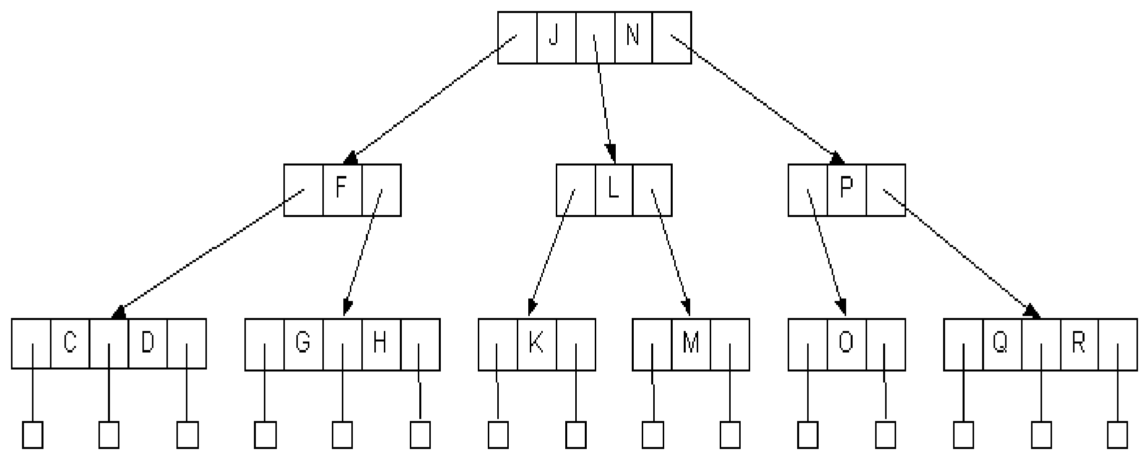
J
F
L
P
C
D
G
H
K
M
N
Q
R



4-node, not valid in 2-3 tree
M
N
O
Three 2-nodes
N
M
O



N
L
P
K
M
O
Q
R



J
N
F
L
P
C
D
G
H
K
M
O
Q
R

Practice

- Build a BST with 50, 70, 20, 60, 40, 10
- Build a 2-3 tree with 50, 70, 20, 60, 40, 10

- What is the height of the tree in both the trees?

- Height of the tree is proportional to the access time of a nodes

Summary

- 2-3 Trees are always balanced.
- Nodes are ALWAYS inserted at the bottom-most level.
- Balance is maintained by splitting full nodes and passing up the middle node.
- This makes the tree’s height increase by one, only when the root is split.

2-3-4 Search Tree

Basic Properties

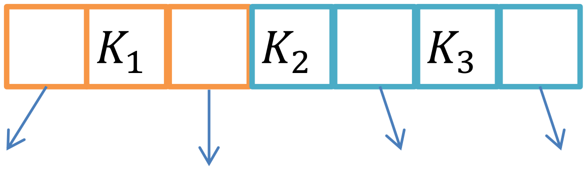
- Similar to 2-3 trees
- Nodes can contain 1, 2, or 3 keys.
- Nodes can has 2, 3, 4 children, hence 2-3-4 tree.
  - Each can have at most 4 children.
- Similarly to 2-3 trees, 2-3-4 trees are guaranteed to be always balanced.
- Balancing algorithm also relies on Splitting nodes
- Number of splits in the worst-case is O(log N)
  - When is the worst-case?
- Average number of splits is very few.

Balancing Algorithm

- Balancing also occurs on insertion.
- Modifying the algorithms for balancing can produce **better efficiency.**
- Previously, with 2-3 Trees, we have seen bottom-up balancing.
- We will see top-down balancing
  - As you go down the tree to insert a node, <u>split any full node</u>.
  - **A full node is a 4-node**.

2-3-4 Node

```
struct Node234
{
  Node23 *left, *midleft, *midright, *right;
  Key key1, key2, key3;
};
```



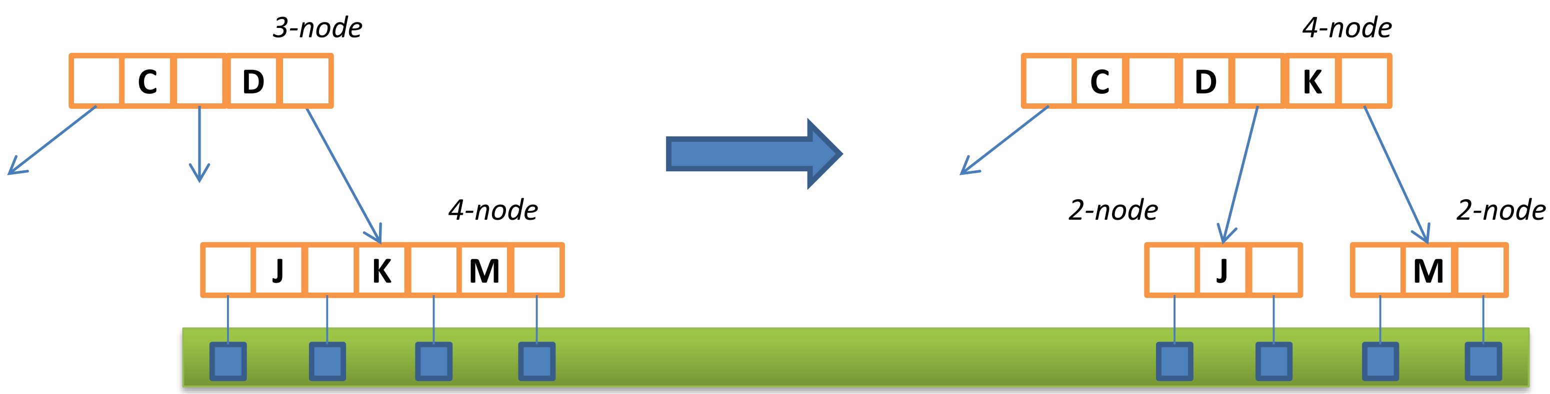
3-node

$$K_1 < K_2 < K_3$$

How to split a 2-3-4 node?

Advantage of Splitting 2-3-4 Trees

- Splitting a node is cleaner.
- Splitting a 4-node into two 2-nodes preserves the number of child links.
- Changes do not have to be propagated. Change remains local to split.



Top-Down Balancing

- Split nodes on the way down.
  - Guarantees that each node we pass through is not a 4-node.
  - When we reach the bottom, we will not be on a 4-node (think about it)
- This way, we only traverse the tree once, when inserting/balancing.
- After each insertion, check if the root is a 4-node
  - If it is, split it directly. This will avoid to do it at next insertion.
  - Splitting the root is the only way to grow the tree.

Example

- Build a 2-3-4 tree with the sequence: 3, 1, 5, 19, 15, 20, 13, 10, 4, 17, 18

Practice

- Build a 2-3-4 tree with the sequence: 6, 3, 9, 4, 5, 8, 11, 2, 1, 7, 10, 12, 14.

Next Lecture

- Red-Black Tree