BIRZEIT UNIVERSITY

BIRZEIT UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

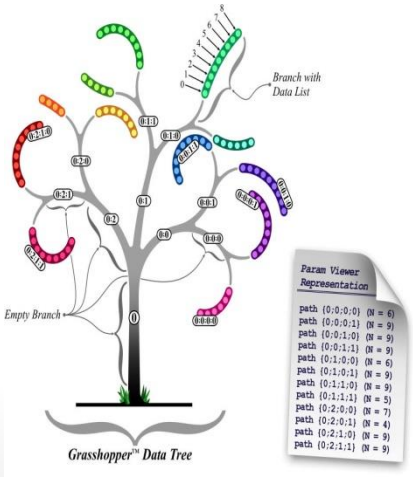
COMPUTER SCIENCE DEPARTMENT

COMP2321

Data Structures

Chapter 4 Trees

Splay Tree and B Tree

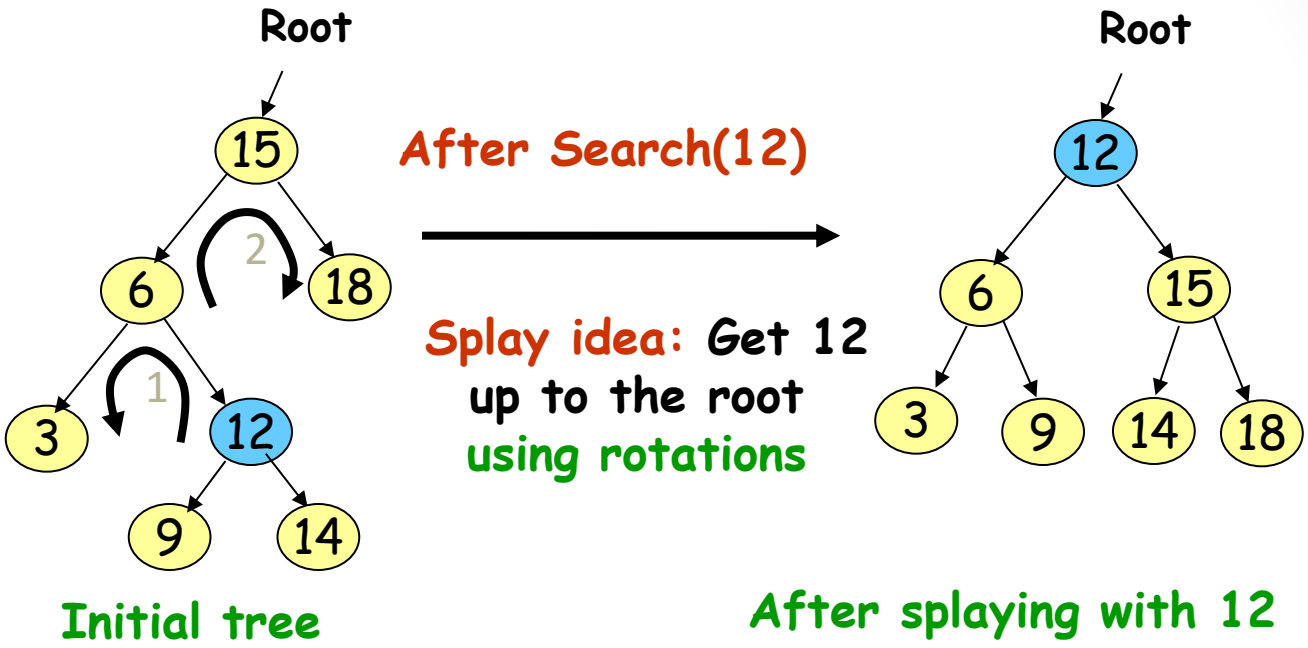


Splay Trees

- Splay Tree is binary search tree (BSTs) that:
  - Are not perfectly balanced all the time
  - It assumes that recently accessed nodes are most likely to visit them again.
  - Allow search and insertion operations to try to balance the tree so that future operations may run faster

- Based on the heuristic:
  - If X is accessed once, it is likely to be accessed again.
  - After node X is accessed, perform "splaying" operations to bring X up to the root of the tree.
  - Do this in a way that leaves the tree more or less balanced as a whole.

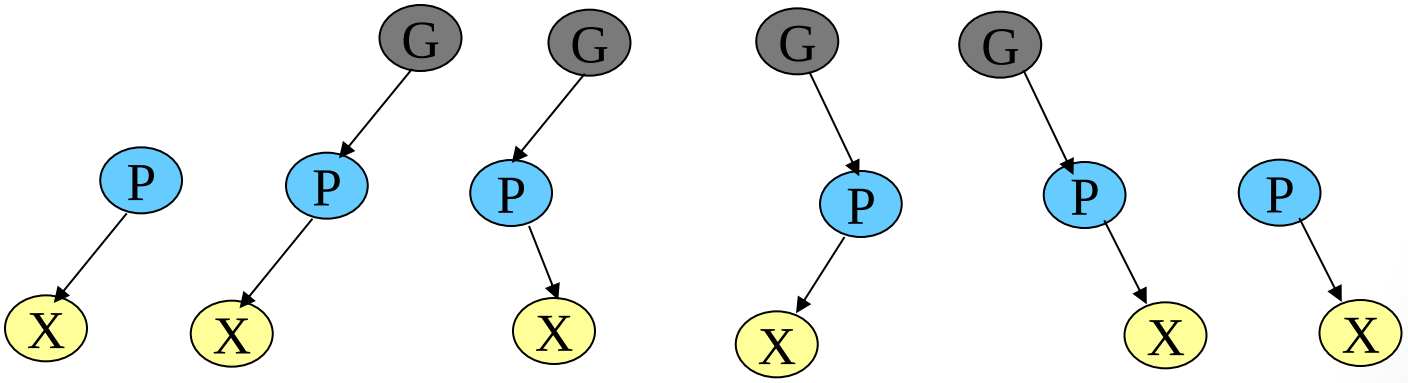
Motivating Example



- Not only splaying with 12 makes the tree **balanced**, subsequent accesses for 12 will take **O(1)** time.
- **Active (recently accessed)** nodes will move towards the root and **inactive** nodes will slowly move further from the root

Splay Tree Terminology: operations

- Let X be a **non-root** node, i.e., has at least 1 ancestor.
- Let P be its **parent** node.
- Let G be its **grandparent** node (if it exists)
- Consider a path from **G to X**:
  - Each time we go **left**, we say that we "**zig**"
  - Each time we go **right**, we say that we "**zag**"
- There are 6 possible cases:

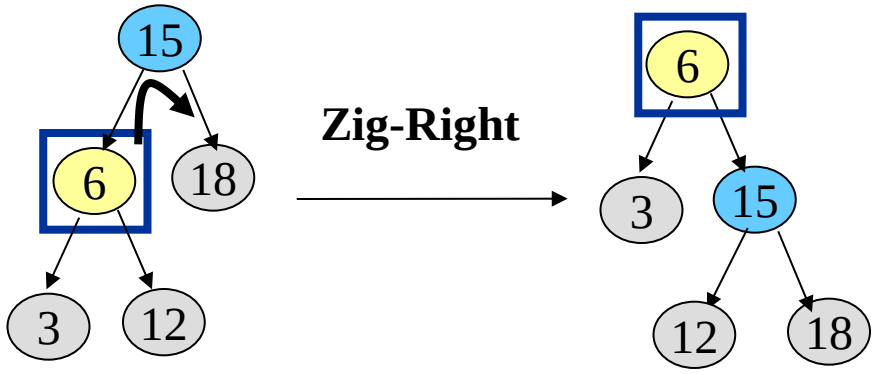


Splay Tree Operations

- When node X is accessed, apply one of six rotation operations:
  - Single Rotations (X has a P but no G)
    - zig, zag
  - Double Rotations (X has both a P and a G)
    - zig-zig, zig-zag
    - zag-zig, zag-zag

Splay Trees: Zig Operation

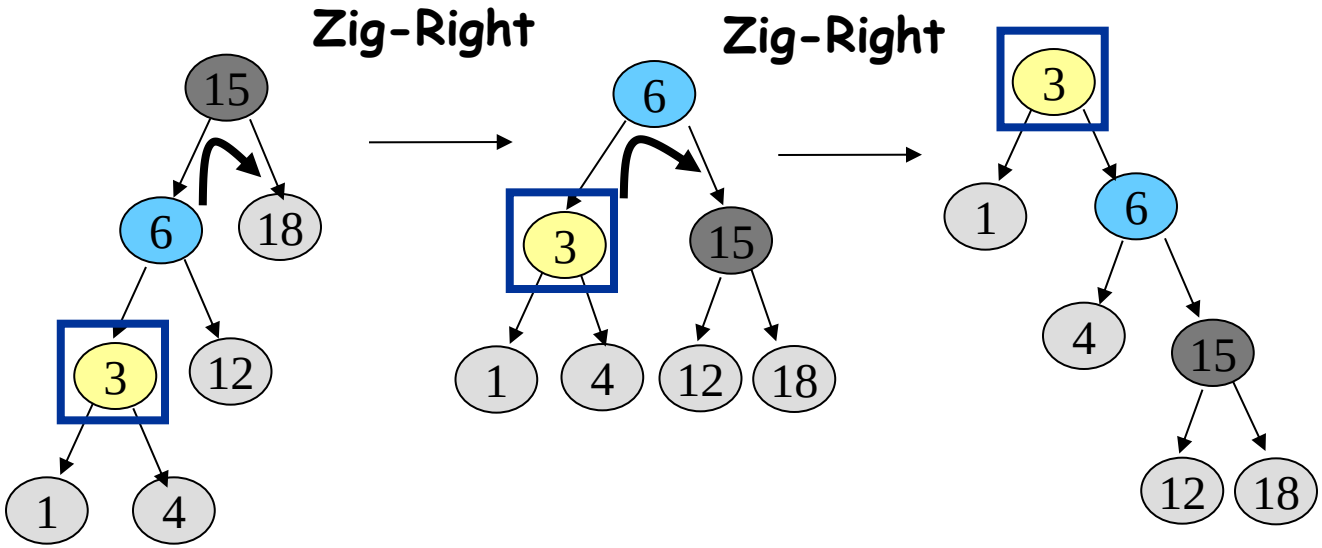
- "Zig" is just a single rotation, as in an AVL tree
- Suppose 6 was the node that was accessed (e.g. using Search)



- "Zig-Right" moves 6 to the root.
- Can access 6 faster next time: O(1)
- Notice that this is simply a right rotation in AVL tree terminology.

Splay Trees: Zig-Zig Operation

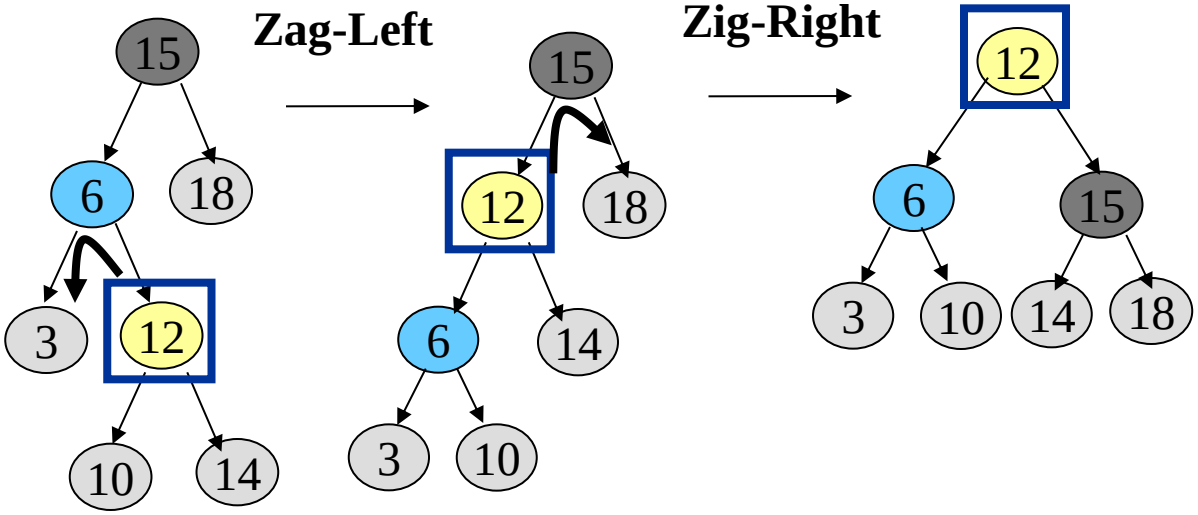
- "Zig-Zig" consists of two single rotations of the same type
- Suppose 3 was the node that was accessed (e.g., using Search)



- Due to "zig-zig" splaying, 3 has bubbled to the top!
- Note: Parent-Grandparent is rotated first.

Splay Trees: Zig-Zag Operation

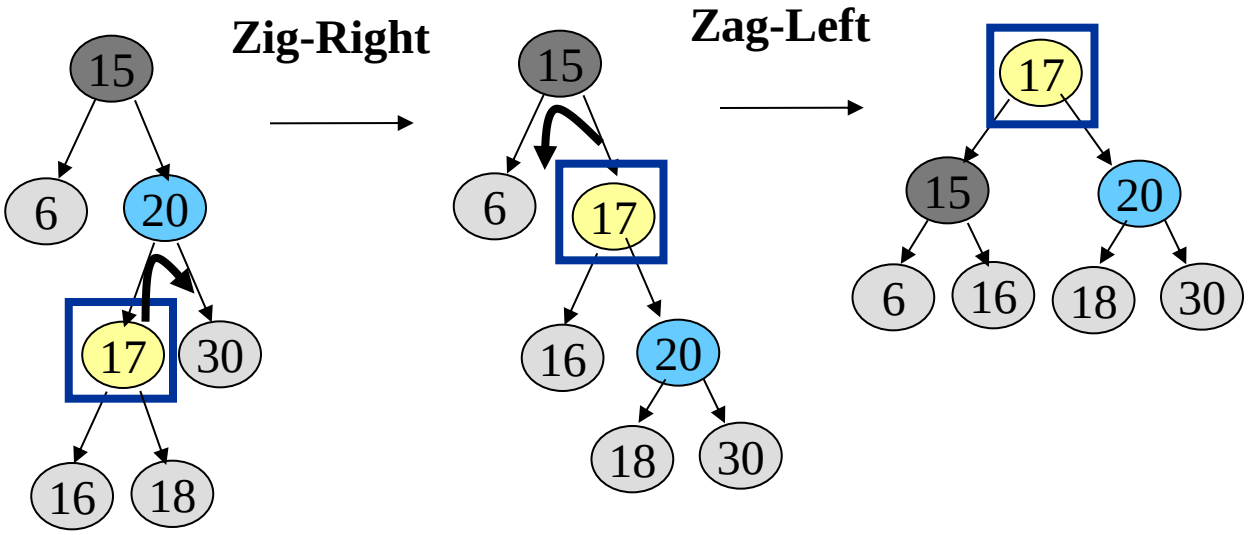
- "Zig-Zag" consists of two rotations of the opposite type
- Suppose 12 was the node that was accessed (e.g., using Search)



- Due to "zig-zag" splaying, 12 has bubbled to the top!
- Notice that this is simply an LR imbalance correction in AVL tree terminology (first a left rotation, then a right rotation)

Splay Trees: Zag-Zig Operation

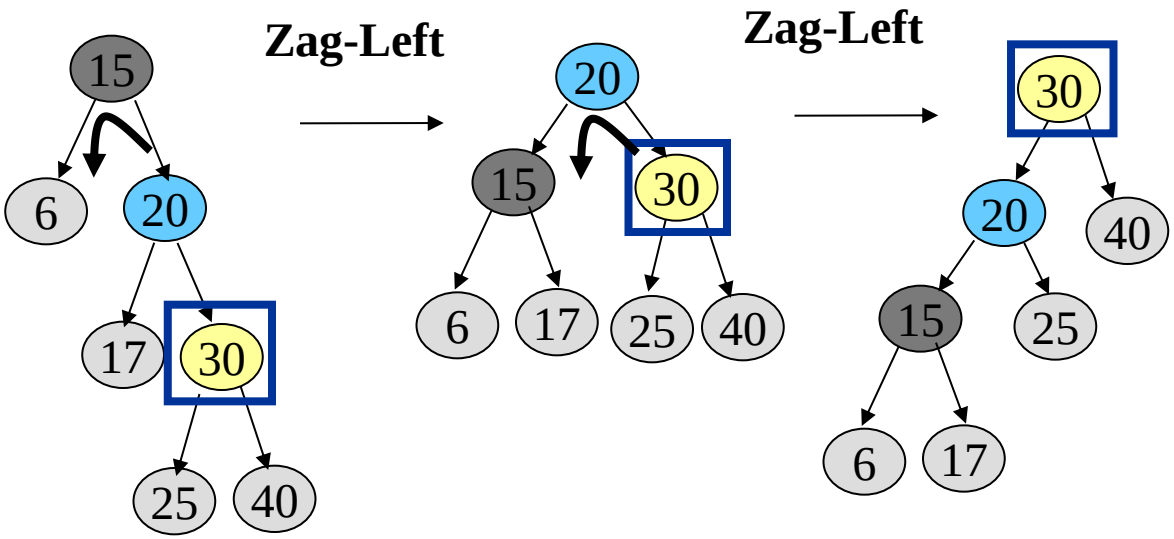
- "Zag-Zig" consists of two rotations of the opposite type
- Suppose 17 was the node that was accessed (e.g., using Search)



- Due to "zag-zig" splaying, 17 has bubbled to the top!
- Notice that this is simply an RL imbalance correction in AVL tree terminology (first a right rotation, then a left rotation)

Splay Trees: Zag-Zag Operation

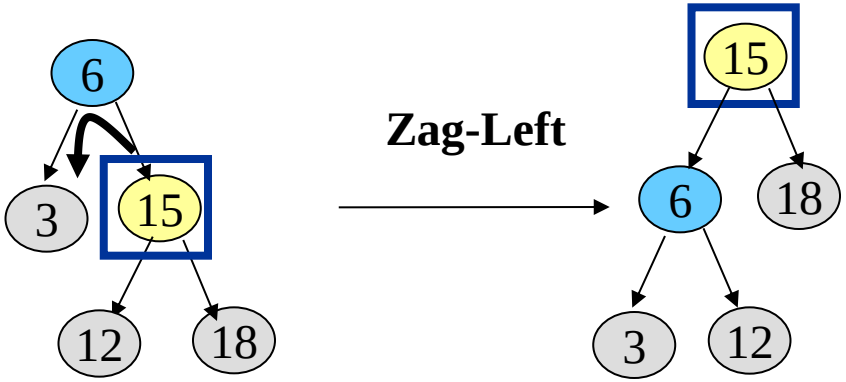
- "Zag-Zag" consists of two single rotations of the same type
- Suppose 30 was the node that was accessed (e.g., using Search)



- Due to "zag-zag" splaying, 30 has bubbled to the top!
- Note: Parent-Grandparent is rotated first.

Splay Trees: Zag Operation

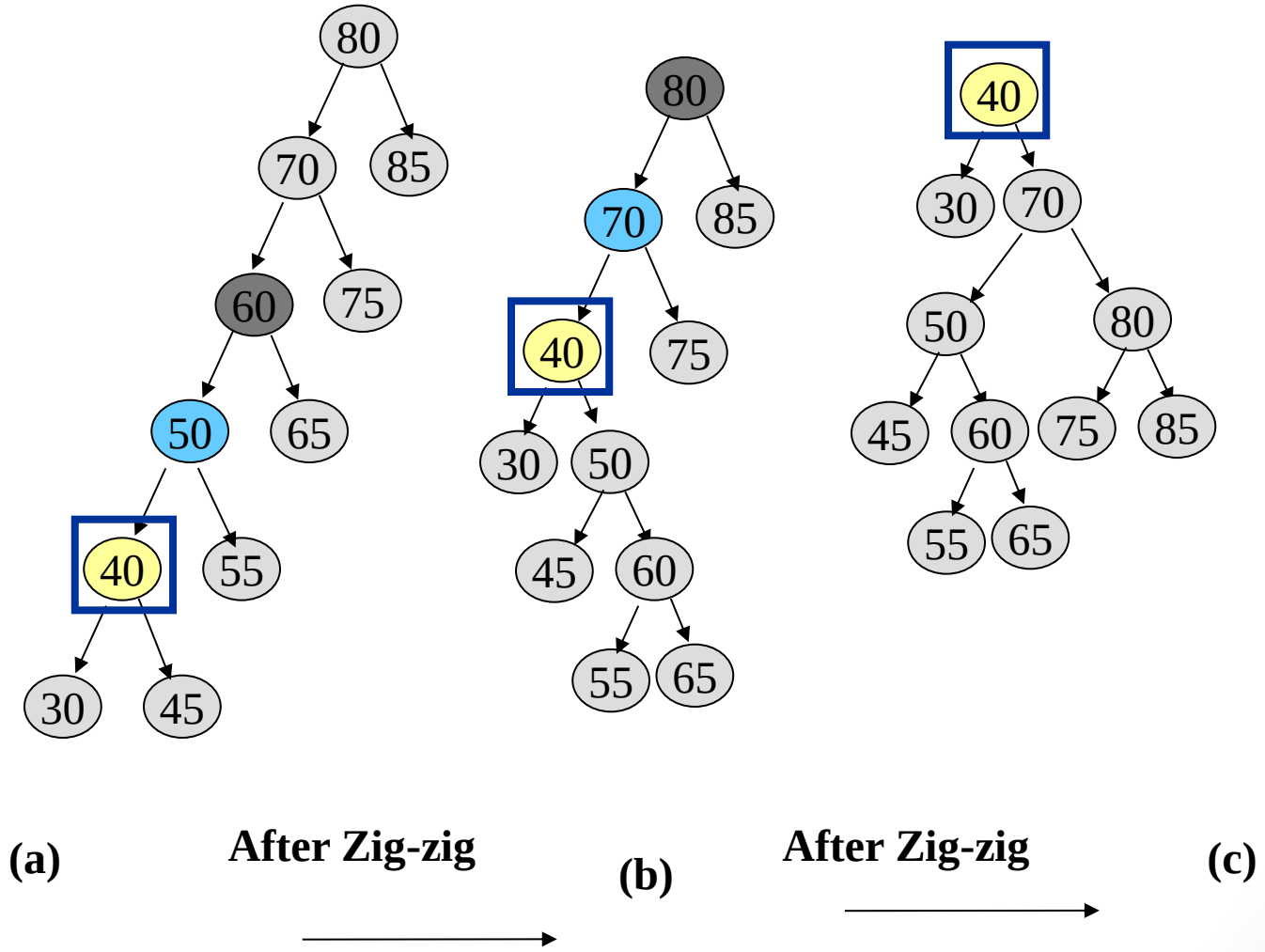
- "Zag" is just a single rotation, as in an AVL tree
- Suppose **15** was the node that was accessed (e.g., using Search)



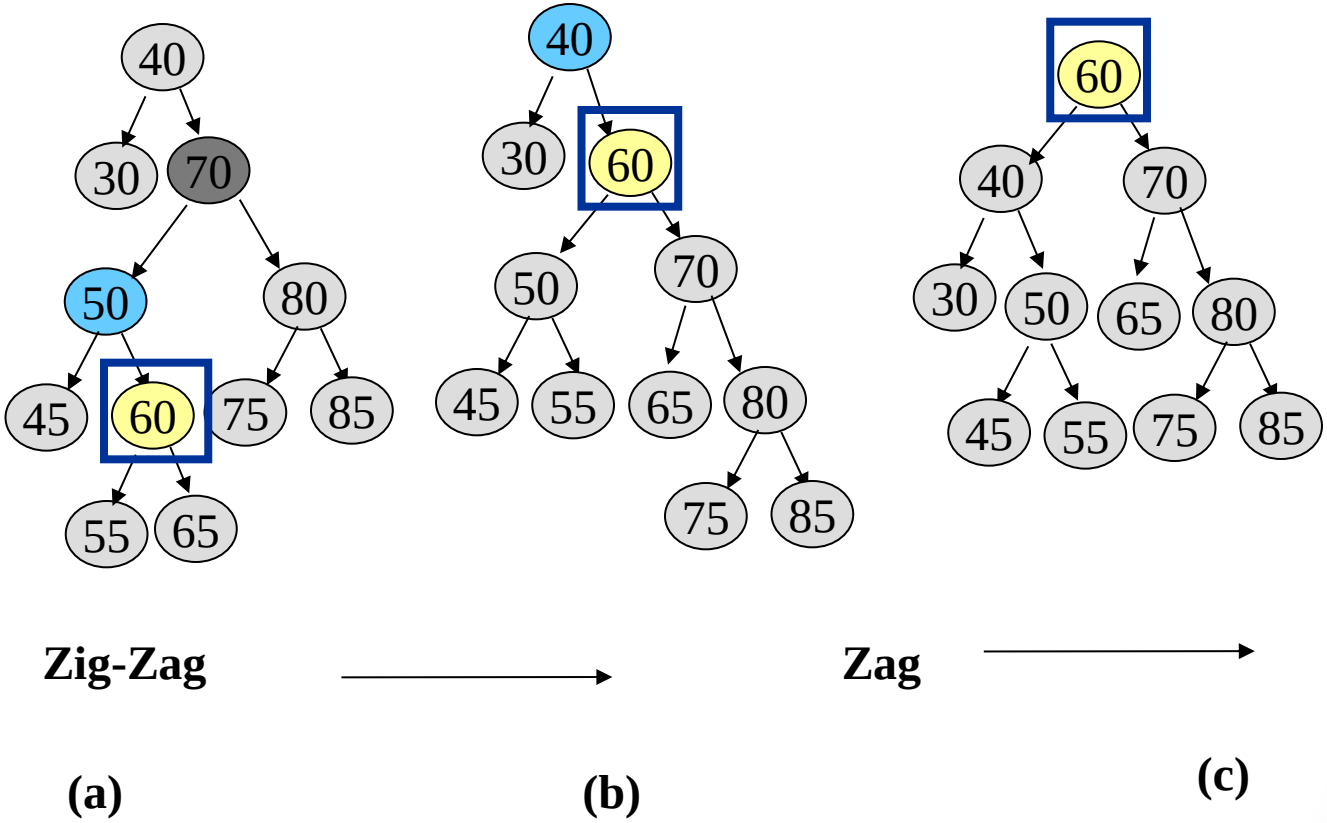
- "Zag-Left" moves 15 to the root.
- Can access 15 faster next time: O(1)
- Notice that this is simply a left rotation in AVL tree terminology

Splay Trees:

Example – 40 is accessed



Splay Trees:
Example – 60 is accessed



Splaying during other operations

BIRZEIT UNIVERSITY

- Splaying can be done not just after Search, but also after other operations such as Insert/Delete.

- **Insert X**: After **inserting X** at a leaf node (as in a regular BST), **splay X up to the root**

- **Delete X**: Do a **Search on X and get X up to the root**. Delete X at the root and move the largest item in its left sub-tree, i.e, **its predecessor, to the root using splaying.**

- **Note on Search X**: **If X was not found**, splay the leaf node that the Search **ended up with to the root**.

Any sequence of M operations on a splay tree of size N takes O(M log N) time.

Exercise: Do it by yourself

- Insert the keys 4,9,3,7,5,6 in that order into an empty splay tree.

A. Delete 9

B. Find 3

- Insert the keys 1, 2, …, 7 in that order into an empty splay tree.

What happens when you access “7”?

Hint: ensure your solution by using this website
https://www.cs.usfca.edu/~galles/visualization/SplayTree.html

B-Trees

DEF: A B-Tree of order m is an m-way tree such that

1. All leaf nodes are at the same level.
2. All non-leaf nodes (except the root) have at most m and at least m/2 children.
3. The number of keys is one less than the number of children for non-leaf nodes and at most m-1 and at least m/2 for leaf nodes.
4. The root may have as few as 2 children unless the tree is the root alone.

Example for m = 5

DEF: A B-Tree of order 5 is an 5-way tree such that

1. All leaf nodes are at the same level.
2. All non-leaf nodes (except the root) have at most 5 and at least 2 children.
3. The number of keys is one less than the number of children for non-leaf nodes and at most 4 and at least 2 for leaf nodes.
4. The root may have as few as 2 children unless the tree is the root alone.

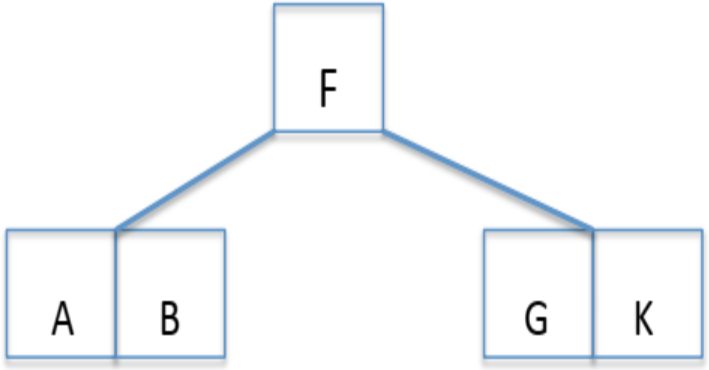
Creating a B-tree of order 5

A G F B K D H M J E S I R X C L N T U P

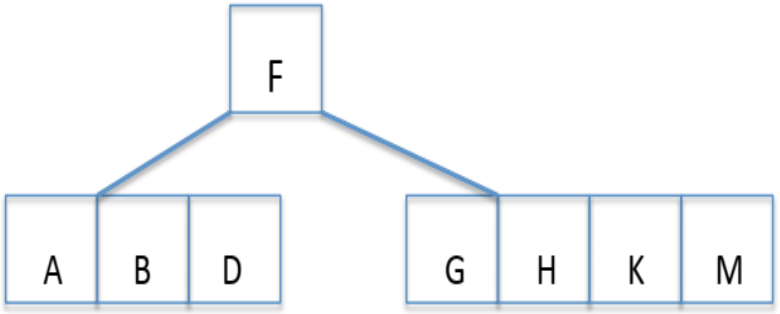
A G F B K D H M J E S I R X C L N T U P

A G F B K D H M J E S I R X C L N T U P

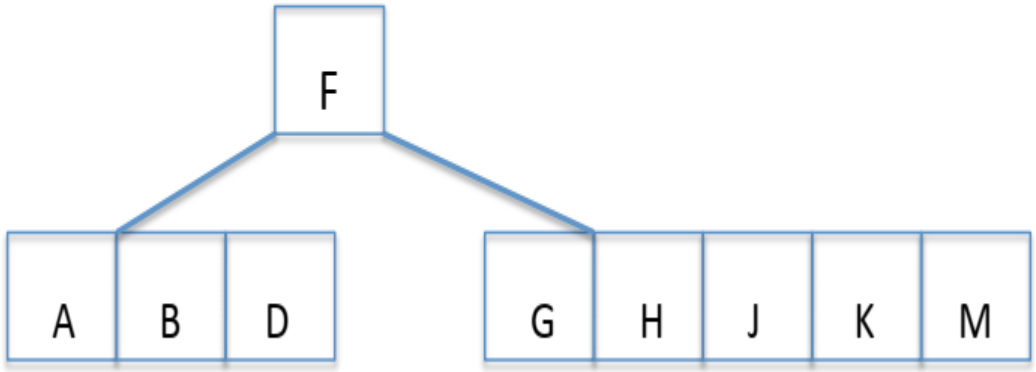
A G F B K D H M J E S I R X C L N T U P



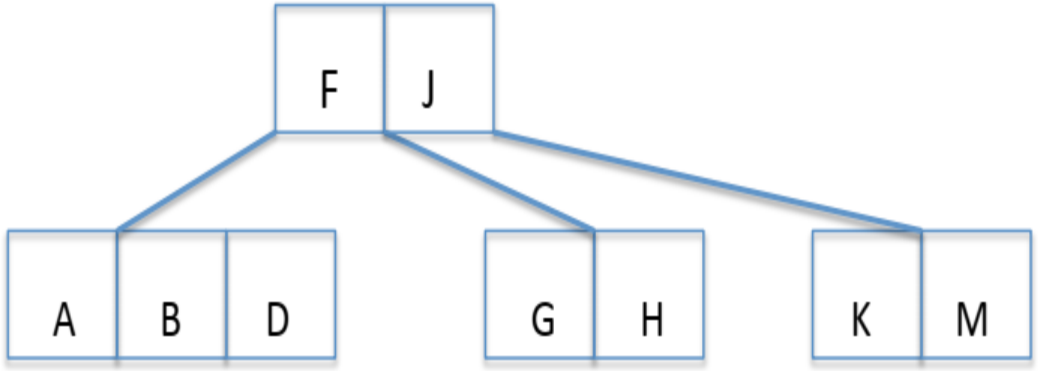
A G F B K D H M J E S I R X C L N T U P



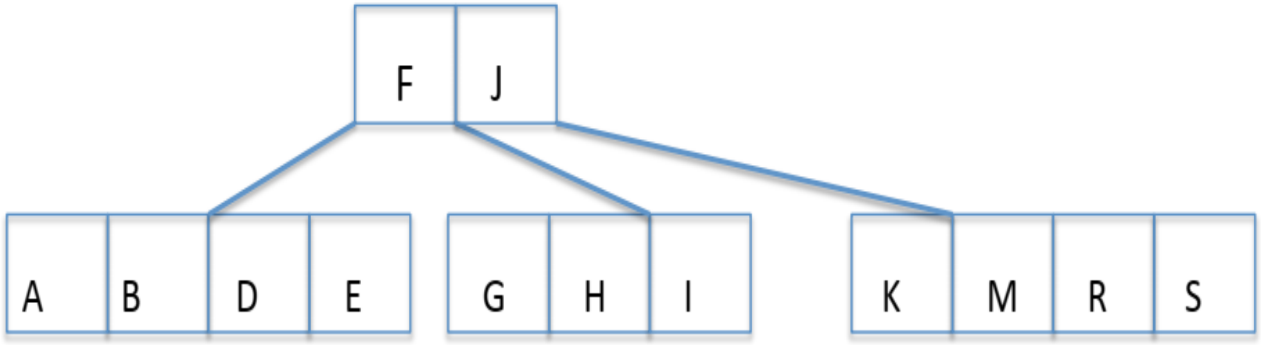
A G F B K D H M J E S I R X C L N T U P



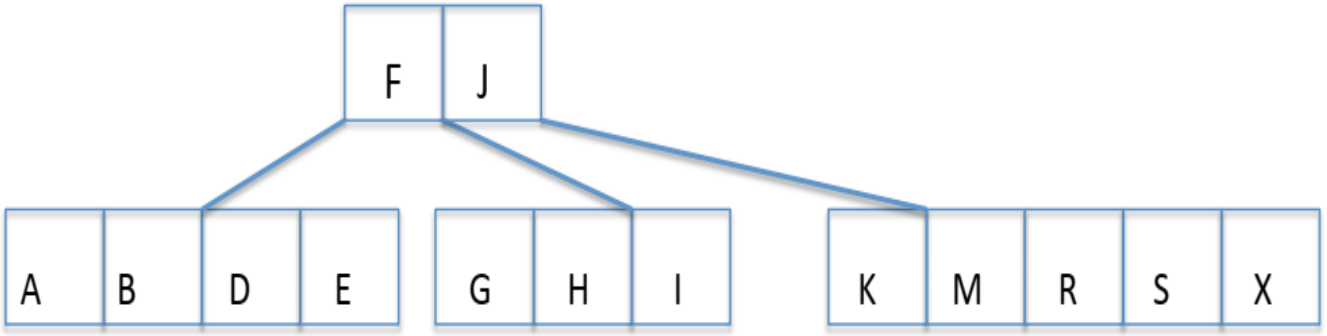
A G F B K D H M J E S I R X C L N T U P



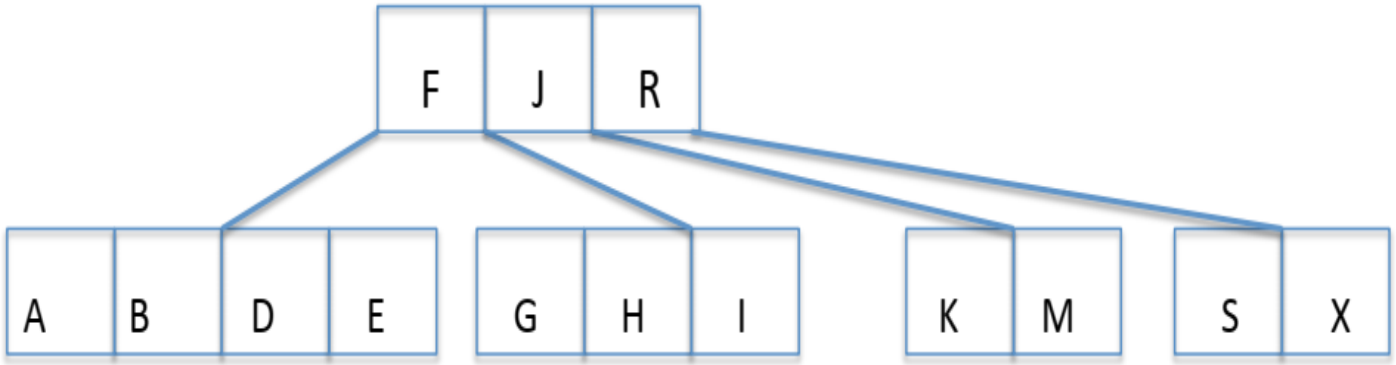
A G F B K D H M J E S I R X C L N T U P



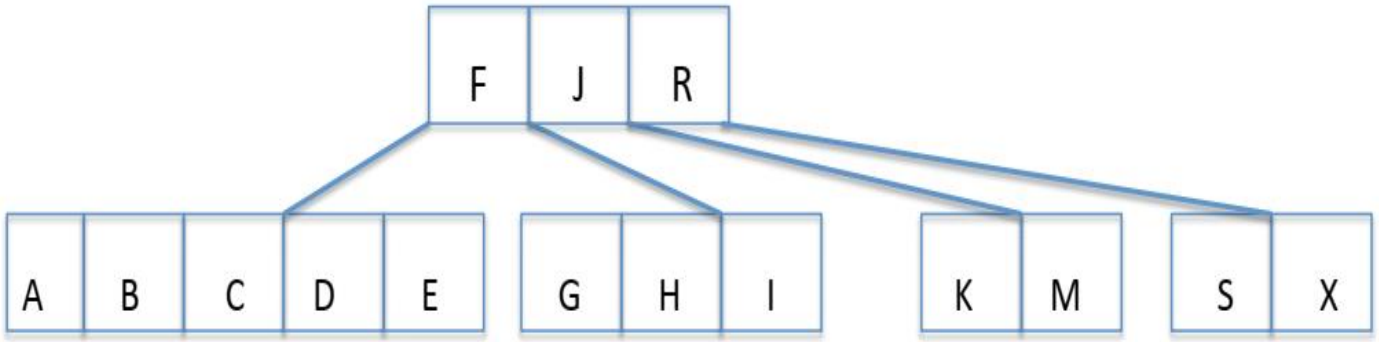
A G F B K D H M J E S I R X C L N T U P



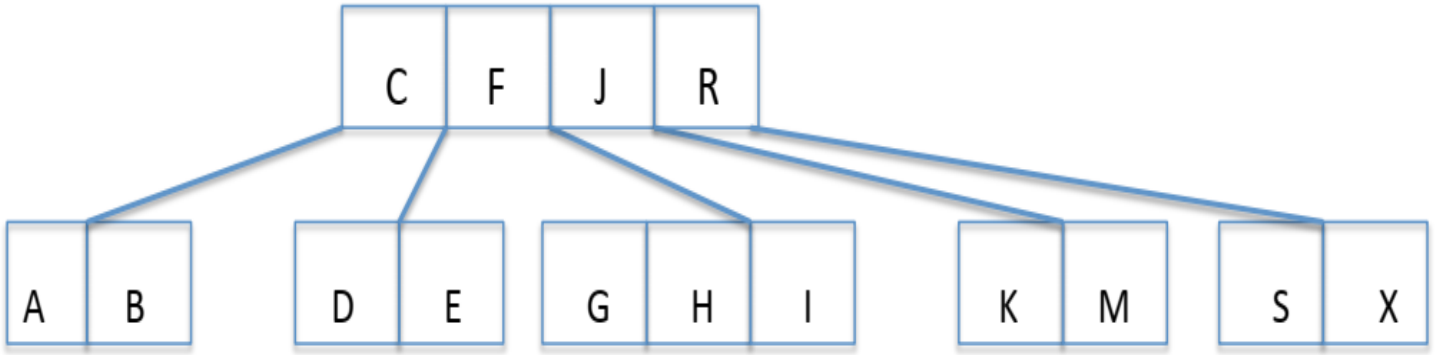
A G F B K D H M J E S I R X C L N T U P



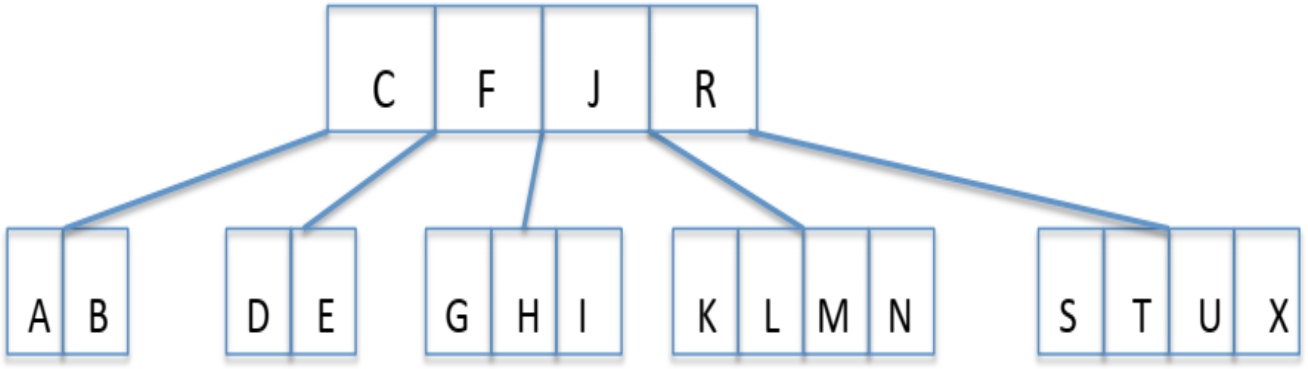
A G F B K D H M J E S I R X C L N T U P



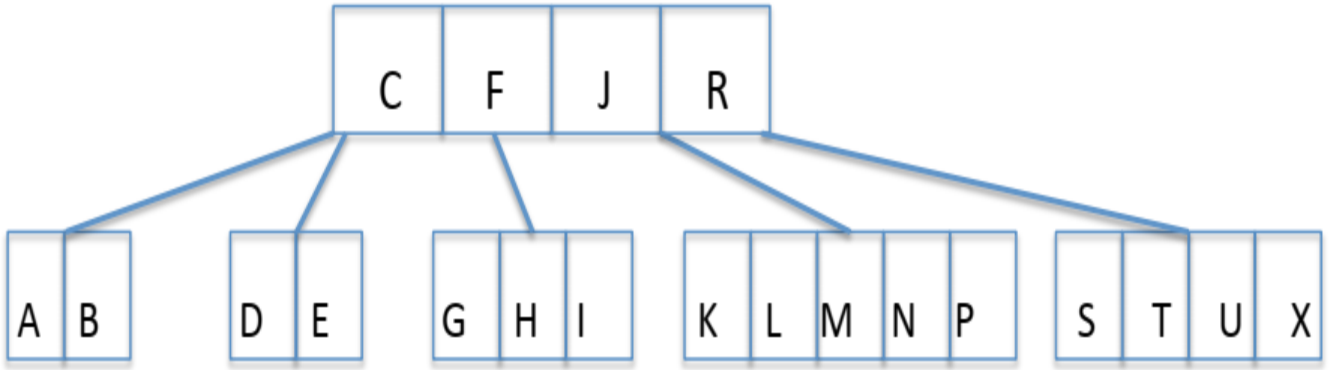
A G F B K D H M J E S I R X C L N T U P



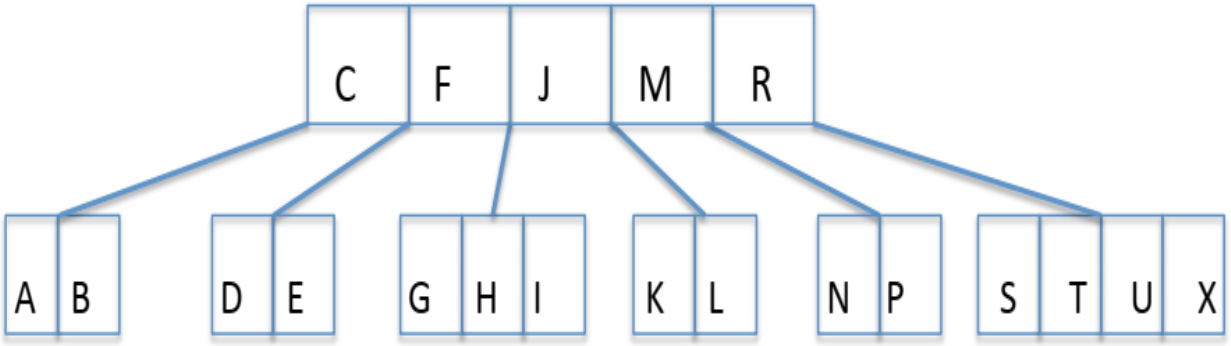
A G F B K D H M J E S I R X C L N T U P



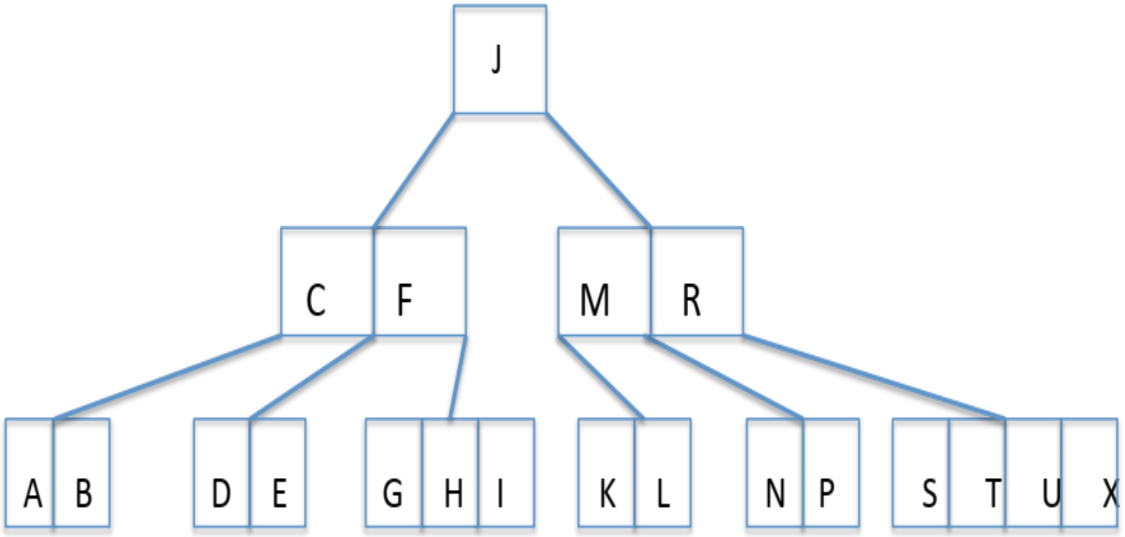
A G F B K D H M J E S I R X C L N T U P

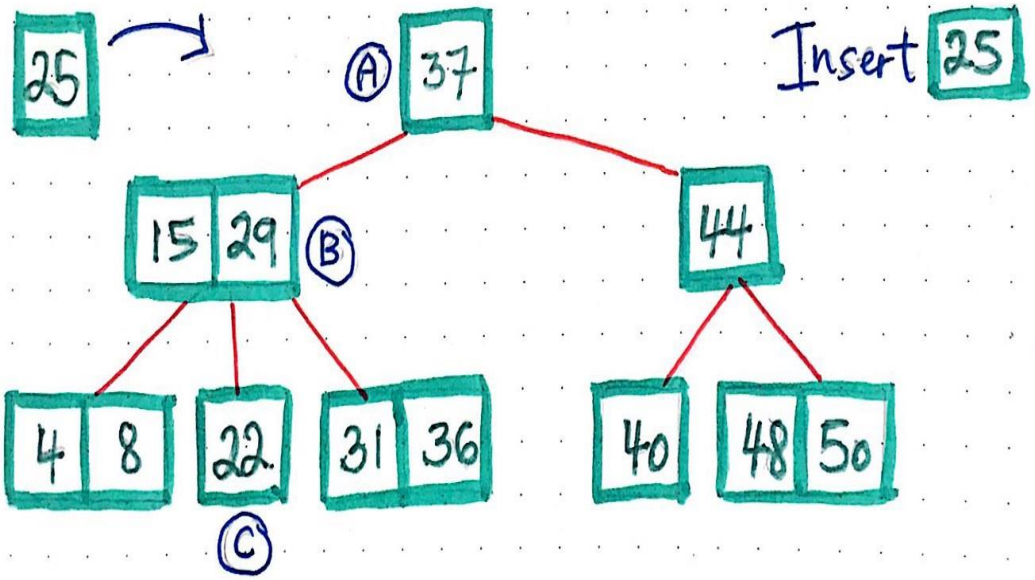


A G F B K D H M J E S I R X C L N T U P



A G F B K D H M J E S I R X C L N T U P

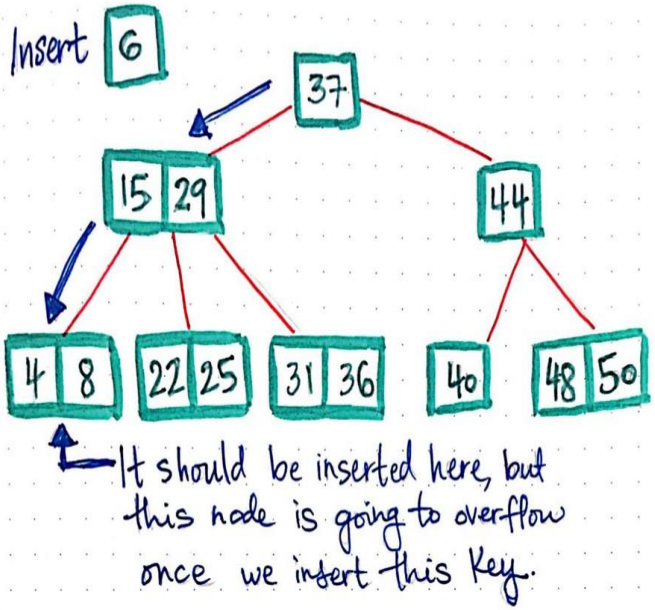




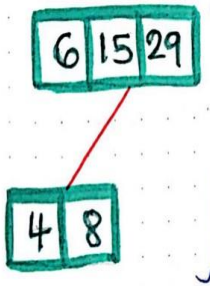
(A) Start from the root 25<37, go left.

(B) 25 is between 15 and 29, go to middle.

(C) Insert 25 into node containing Key 22.



We can **Split** this node to make room for the value that needs to be inserted.



We can take the middle element from the overflowed node & bring it up to the correct place in the parent node.

* If the parent overflows, we can split again, all the way up to the root node.

Homework :

Insert the following elements
10,20,40,50,60,70,80,30,35,5,15,60
in a B-Tree of order M=4

Solution using
https://www.cs.usfca.edu/~galles/visualization/BTree.html

Homework :

Insert the following elements 9,0,8,1,7,2,6,3,5,4, in a B-Tree of order M=3,

Solution using https://www.cs.usfca.edu/~galles/visualization/BTree.html

THANK YOU