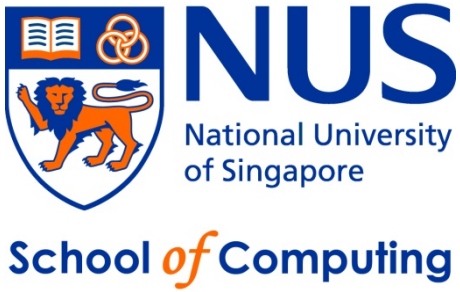
CS2040S – Data Structures and Algorithms

Lecture 13 – Splay Tree

chongket@comp.nus.edu.sg



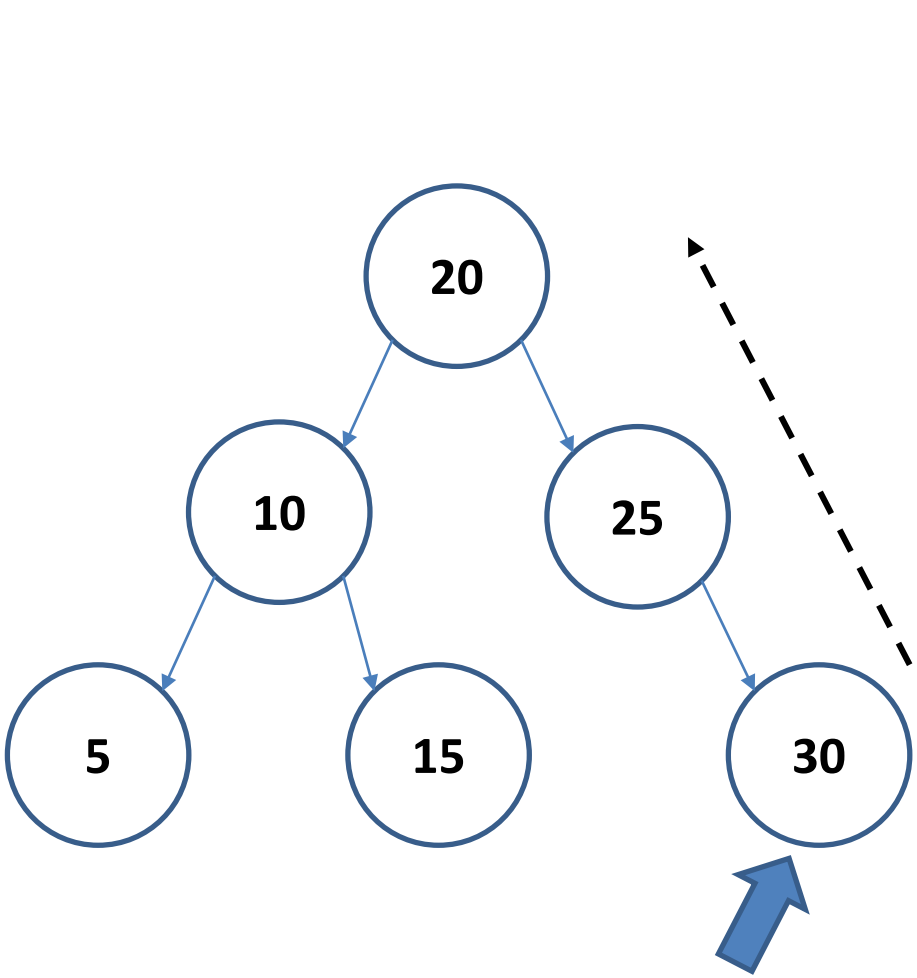
Splay Tree – Another self-balancing BST

- Balancing is based on the following heuristic:
  - The most frequently accessed key will most likely be accessed again and so should be placed at the top of the tree, making future accesses O(1) time
  - No need to keep height information

- Search is modified so that whenever a search key X is found, the node containing X is shifted to the root using a series of rotation operations (Splay Steps) ← Splaying

- Insertion/deletion is their standard BST counterpart with additional Splaying after the insertion/deletion

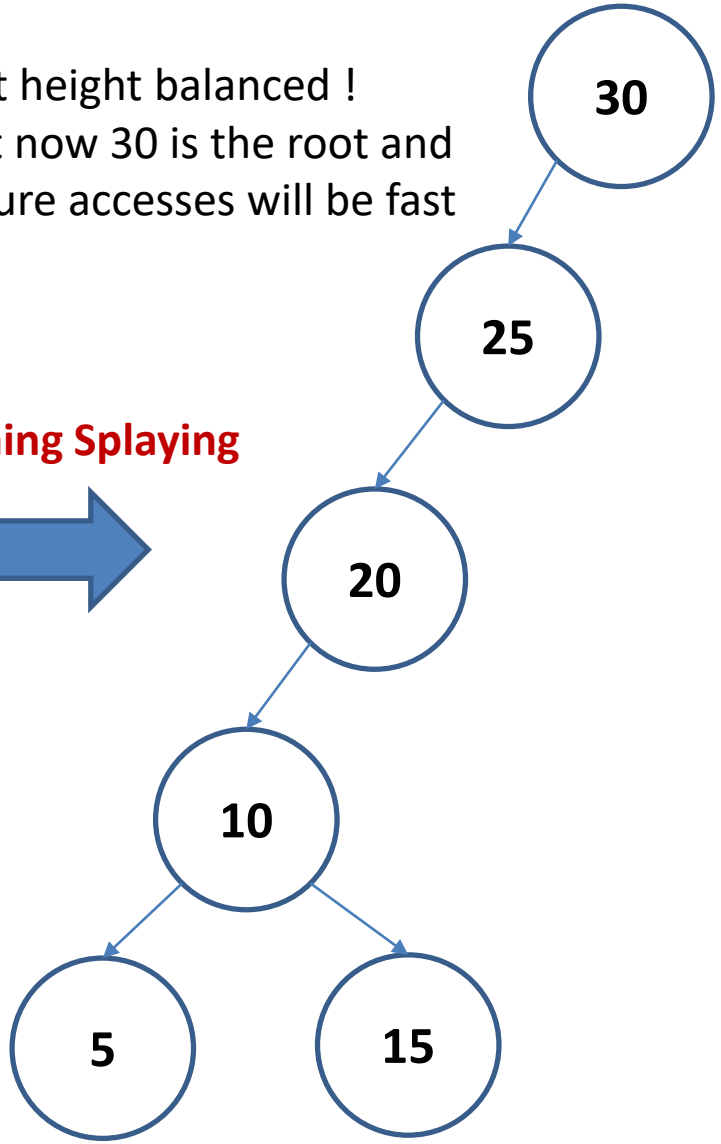
Example of balancing in Splay Tree



If 30 was searched, thus increasing frequency of its access, 30 will be shifted towards the root of the tree

Not height balanced !
But now 30 is the root and future accesses will be fast

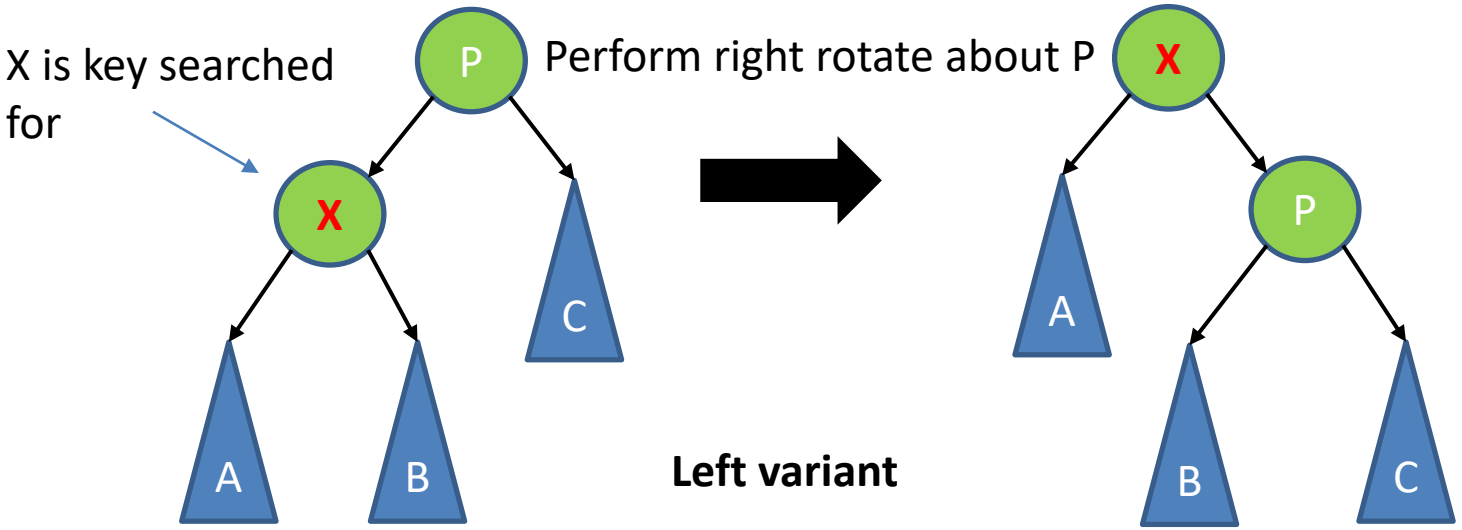
Performing Splaying



Splay Step – 6 cases to consider (1)

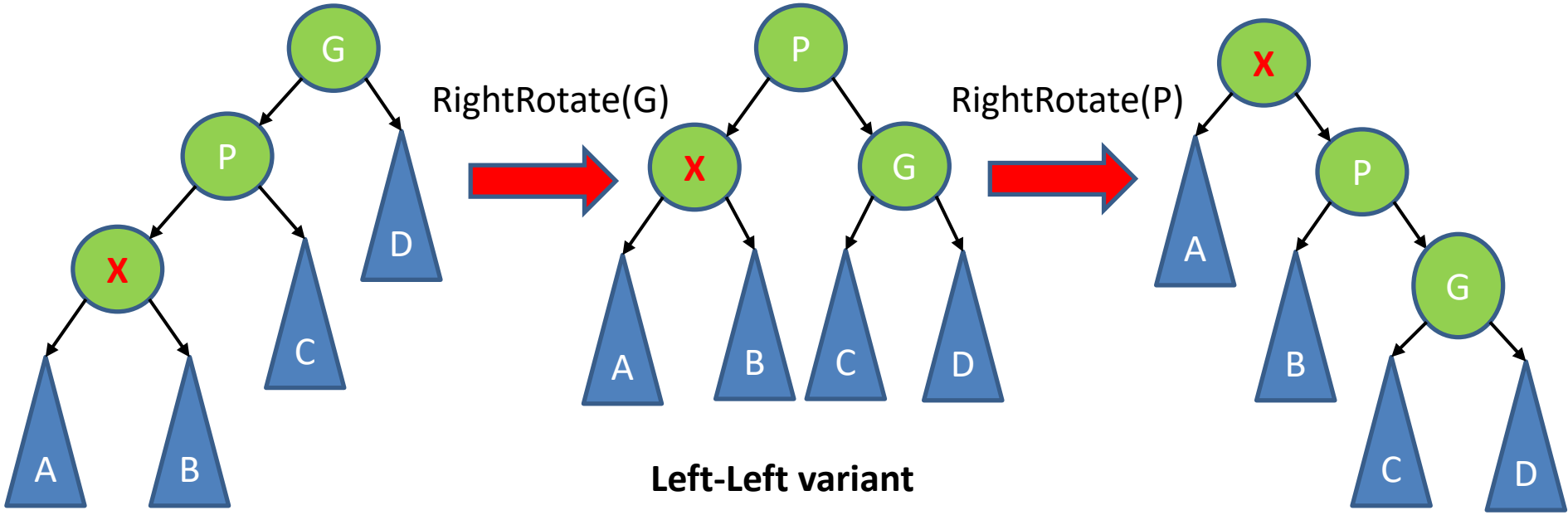
- 3 cases (each case has two variant so 6 cases in all)

- Case 1: Zig Step (last step in a series of splay steps)
  - If X = P.left and P = root (left variant) → RightRotate(P)
  - If X = P.right and P = root (right variant) → LeftRotate(P)



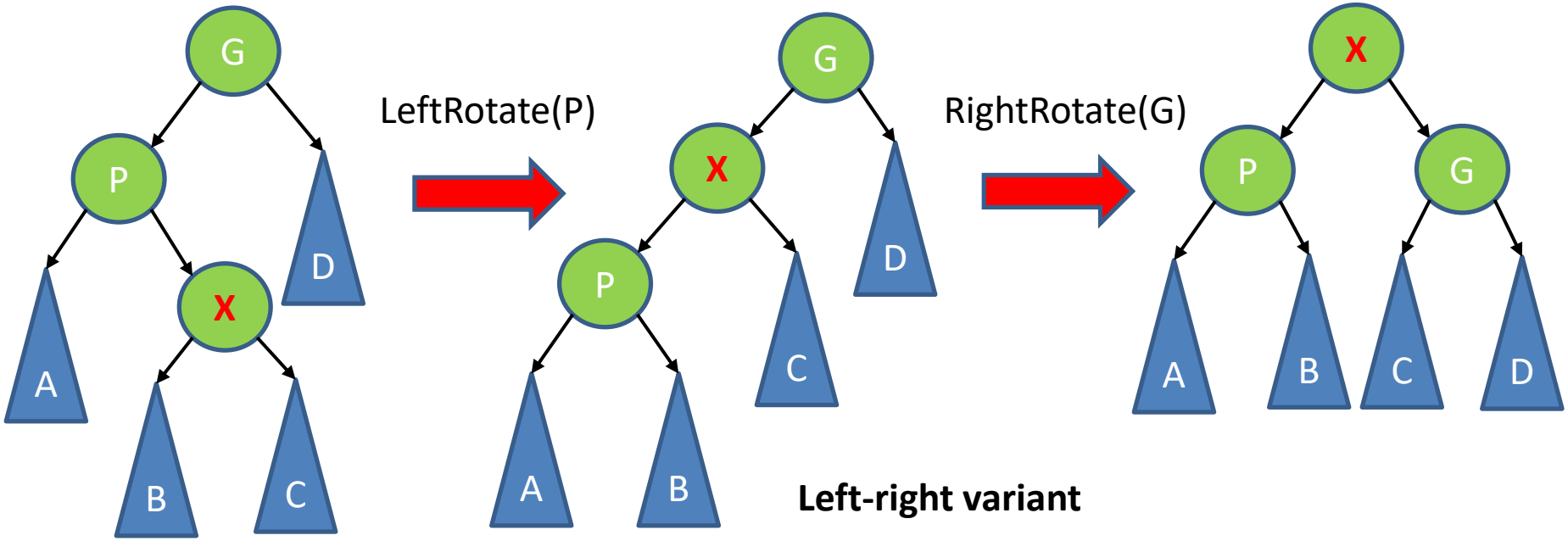
Splay Step – 6 cases to consider (2)

- Case 2: Zig-Zig Step
  - If X = P.left and P = G.left (left-left variant)
    → RightRotate(G) then RightRotate(P)
  - If X = P.right and P = G.right (right-right variant)
    → LeftRotate(G) then LeftRotate(P)



Splay Step – 6 cases to consider (3)

- Case 3: Zig-Zag Step
  - If P = G.left and X = P.right (left-right variant)
    → LeftRotate(P) then RightRotate(G)
  - If P = G.right and X = P.left (right-left variant)
    → RightRotate(P) then LeftRotate(G)



Splay Tree Operations

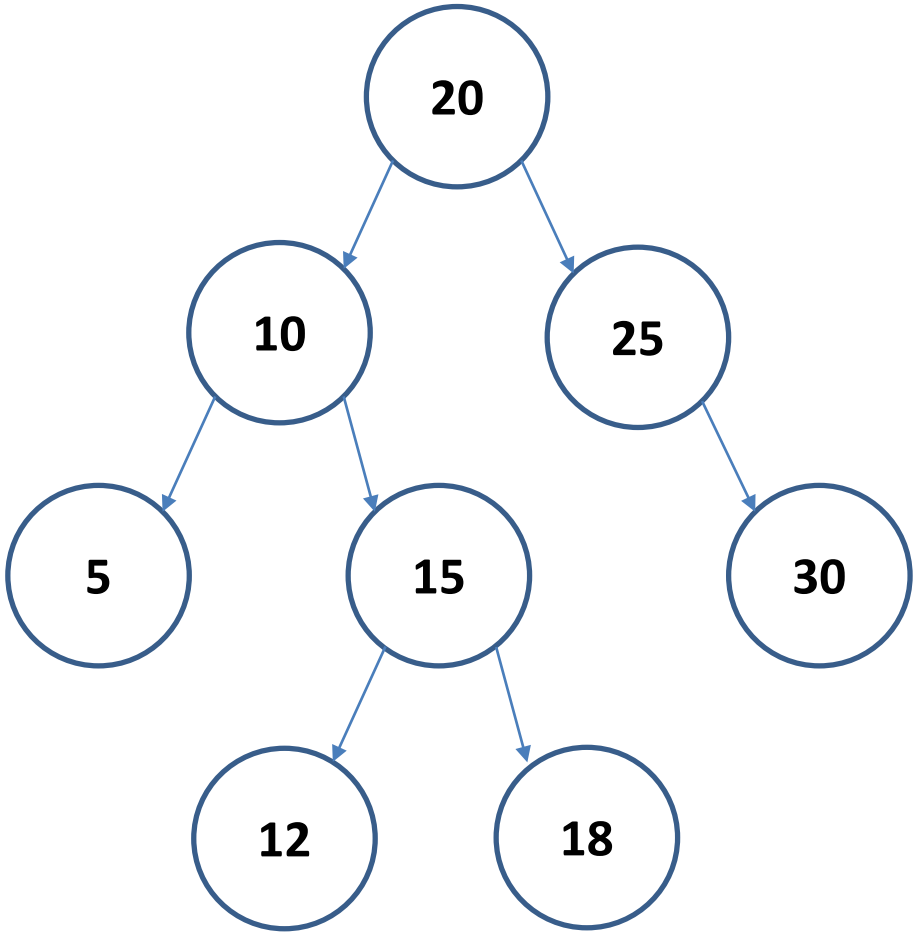
- Search
  - If successful, for the searched node x, repeated perform splay steps on x until it is the root
  - If unsuccessful, repeated splay the last node before null was reached
- Insert
  - After the new node x is inserted, repeated perform splay steps on x until it is the root

Splay Tree Operations

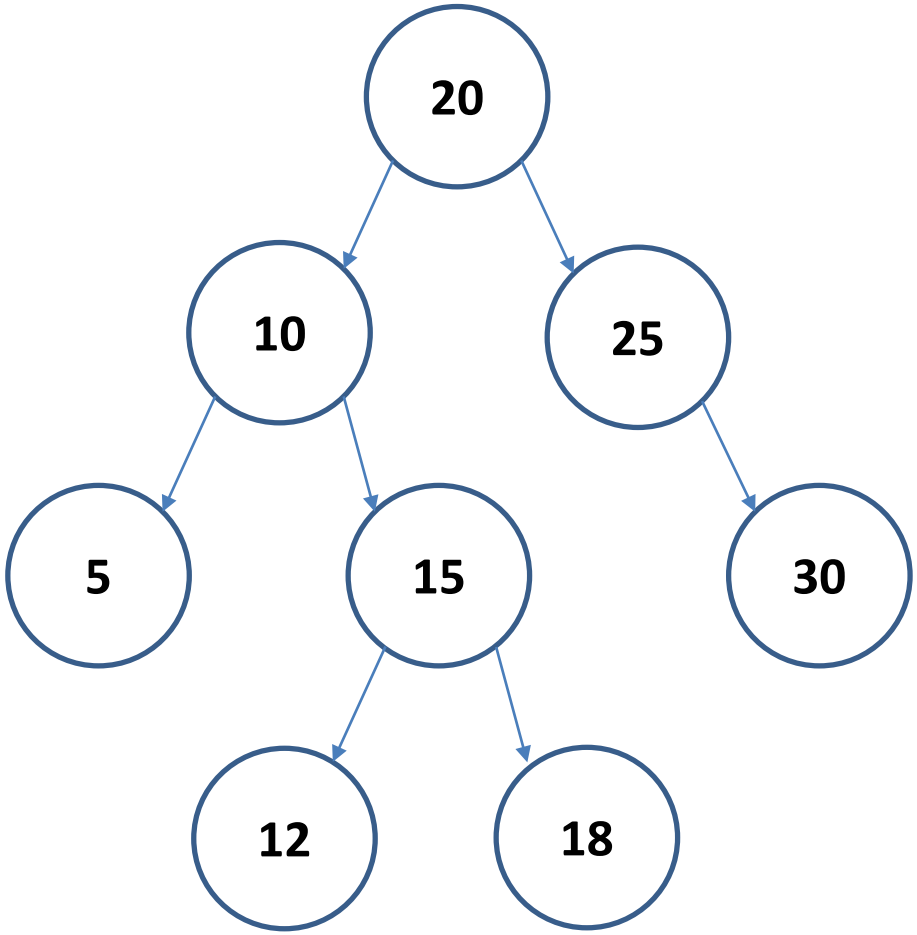
- Delete
  - If node x to be deleted is the only node in the tree, do nothing after x is deleted
  - If node x to be deleted has 0 or 1 child, after x is deleted (standard BST deletion), splay x's parent to the root
  - If node x to be deleted has 2 children, after x's successor is deleted (again standard BST deletion), splay x's successor's parent to the root

Exercises

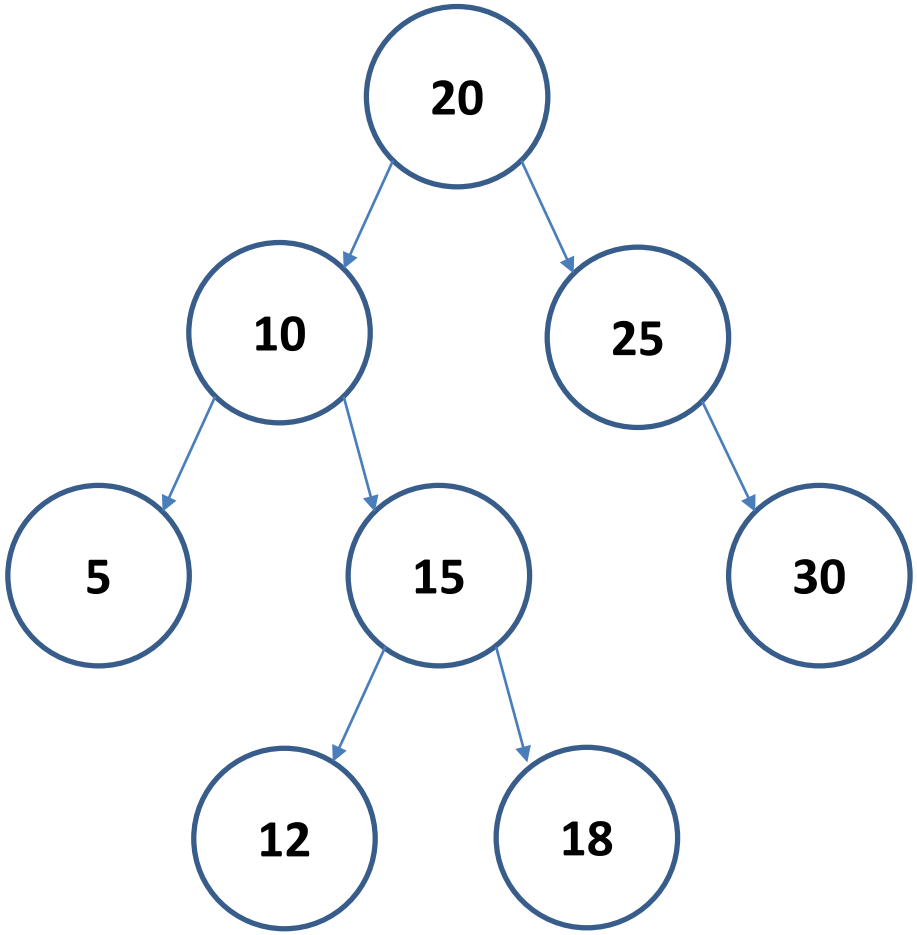
- Search for 12



- Insert for 35



- Remove 15



Summary Splay Tree

- Heuristic approach to balancing a BST which can achieve good results with real life applications
- No need to height balance the tree !
- On average search/insert/delete will still run in O(lgN) time although in the worst case they can run in O(N) time