Binary Search Trees
Part 02
Definition: A data structure that can be defined recursively as a collection of nodes, where each node is a data structure consisting of a value, together with a list of references (edges) to nodes, with the constraints that no reference is duplicated, and none points to the root.
• Trees Have
  – Nodes
  – Edges

• Trees CANNOT
  – Contain Self-Referencing Edges
  – Have Cycles
  – Be Disjointed
Common Terms

- **Root** – The top node in a tree.
- **Child** – A node’s reference which is at a lower level.
- **Parent** – The converse notion of child.
- **Siblings** – Nodes with the same parent.
- **Leaf** – a node with no children.
- **Degree** – number of sub trees of a node.
- **Edge** – connection between one node to another.
- **Path** – a sequence of nodes and edges connecting a node with a descendant.
- **Level** – The level of a node is defined by 1 + the number of connections between the node and the root.
- **Height of tree** –The height of a tree is the number of edges on the longest downward path between the root and a leaf.
- **Height of node** –The height of a node is the number of edges on the longest downward path between that node and a leaf.
- **Depth** –The depth of a node is the number of edges from the node to the tree’s root node.
• Tree Structure
• Node’s data must be comparable
• Node’s have at most two children
  – Left Child
  – Right Child
• Left child’s value must be LESS THAN the parent’s value
• Right child’s value must be GREATER THAN the parent’s value
• No Duplicate Values
• Assume Leaves are NULL references
- Search
  - Start from the Root
  - If it is a leaf then return false
  - If the target value matches the Node’s data then return true
  - If the target value is less than the Node’s data then recursively GO LEFT
  - If the target value is greater than the Node’s data then recursively GO RIGHT
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree
Remove
- Find the Node with the target value that is to be removed
- If that Node has no children then remove that Node’s reference from its parent
- If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
- If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree

Remove 2 Example

Target = 2
Remove

– Find the Node with the target value that is to be removed
– If that Node has no children then remove that Node’s reference from its parent
– If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
– If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree

Remove 2 Example

Target = 2
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree

Remove 2 Example
Remove

- Find the Node with the target value that is to be removed
- If that Node has no children then remove that Node’s reference from its parent
- If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
- If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree

Remove 2 Example

Target = 2
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree

Remove 2 Example

Target = 4
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree
- Remove
  - Find the Node with the target value that is to be removed
  - If that Node has no children then remove that Node’s reference from its parent
  - If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  - If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree

Remove 8 Example

Target = 8
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree

Remove 8 Example

Target = 8
• **Remove**
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree

### Remove 8 Example

#### Target = 8
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree

Remove 8 Example

Target = 8
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree

Remove 8 Example

Target = 10
• **Remove**
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree

Remove 8 Example

Target = 10
• Remove
  – Find the Node with the target value that is to be removed
  – If that Node has no children then remove that Node’s reference from its parent
  – If that Node has exactly one child (left or right), then replace that Node’s reference from its parent with reference to its child
  – If that Node has 2 children then replace its value with the SMALLEST value found in the RIGHT subtree, then remove the duplicate node from the RIGHT subtree

Remove 8 Example
• Complexity depends on the structure of the tree
• Balanced Trees
  – From the root to any leaf there are AT MOST $\log(n)$ edges
• Unbalanced Trees
  – Have at least one path from root to a leaf that is more than $\log(n)$ edges
• Complexity depends on the structure of the tree
• Balanced Trees
  – From the root to any leaf there are AT MOST $\log(n)$ edges
• Unbalanced Trees
  – Have at least one path from root to a leaf that is more than $\log(n)$ edges
• Unbalanced Tree
  – Add = O(n)
  – Search = O(n)
  – Remove = O(n)
• Balanced Tree
  – Add = $O(\lg(n))$
  – Search = $O(\lg(n))$
  – Remove = $O(\lg(n))$
Self-Balancing Trees
- Change references until the tree is balanced
- Based on criteria like Height or Node “Color”

Rotations are used to Balance the Tree
- Left Rotations
- Right Rotations

Popular Self-Balancing Trees
- AVL
- Red / Black Tree