SPLAY TREES

Jason Aguirre, Afolabi Abayomi, Gavin Doherty

BACKGROUND INFORMATION

- Splay tree invented by Daniel Sleator and Robert Tarjan in 1985.
- A splay tree is a modified version of binary search tree.
- Main difference between other binary search trees: recently accessed node is brought to the root of the tree.

RECALLING BINARY SEARCH TREES

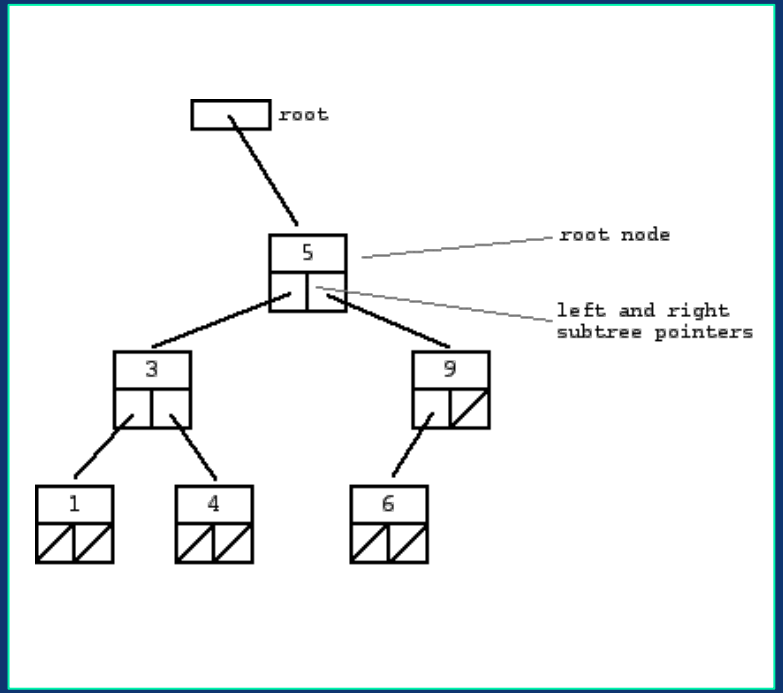
A binary tree:

- Should hold elements in nodes.
- A root node pointer that always points to the initial value.

Each node has a left pointer and a right pointer:

- The left pointer points to a node whose key is less than the parent node's key.
- The right pointer points to a node whose key is greater than the parent node's key.

This pattern continues throughout the tree.



ADVANTAGES OVER OTHER BINARY SEARCH TREES

- Self-Balancing:

  Small memory footprint as there is no need to store balancing information.

- Efficient in situations where some data is accessed more often than the rest.

  (eg. network routers)
  - Utilizes spatial locality in order to reach *O(log n)* performance
  - They have low memory overhead like in Scapegoat trees, which makes them attractive for memory-sensitive programs.

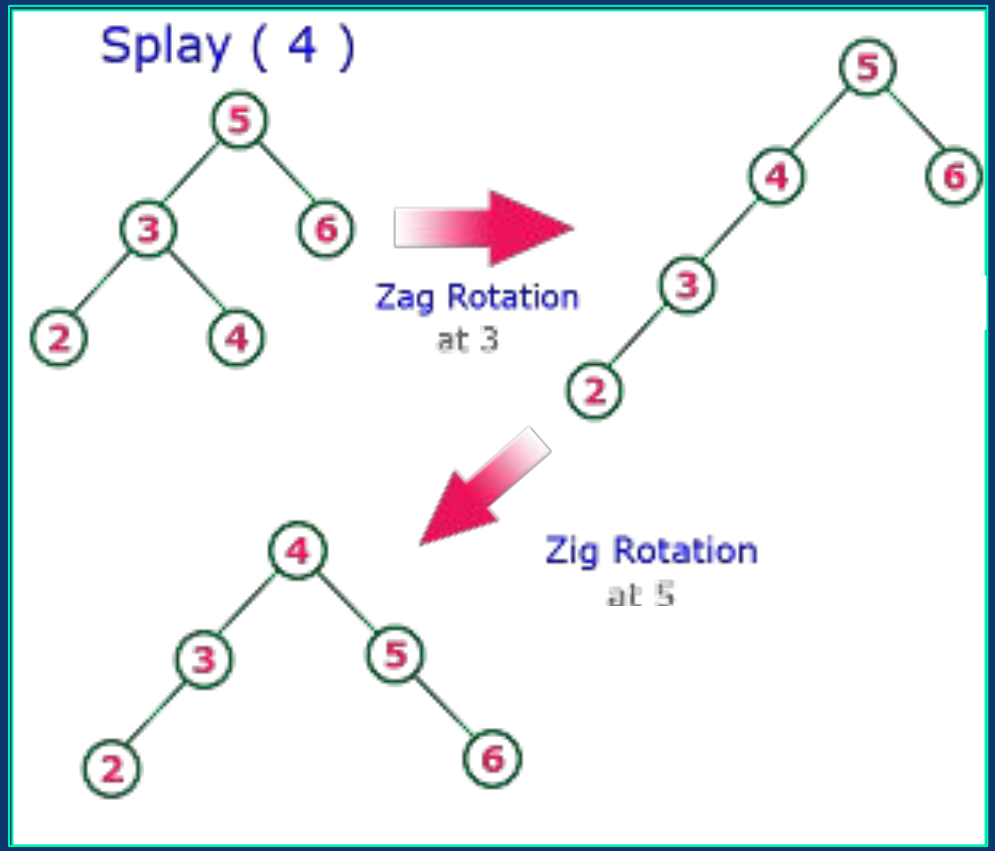
SPLAYING

Splay trees utilize a method called splaying, which occurs after any element is accessed.

Three rotation patterns/ six possible outcomes or a rotation:

- Zig/zag
- Zig-zig/zag-zag
- Zig-zag/zag-zig

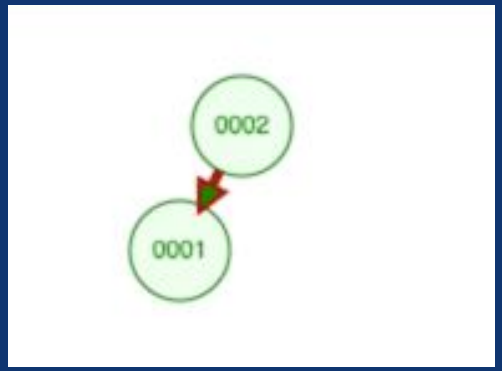
A left rotation is a zag and a right rotation is a zig.



ZIG

Occurs when:

- No grandparent node exists
- Current node is a left child

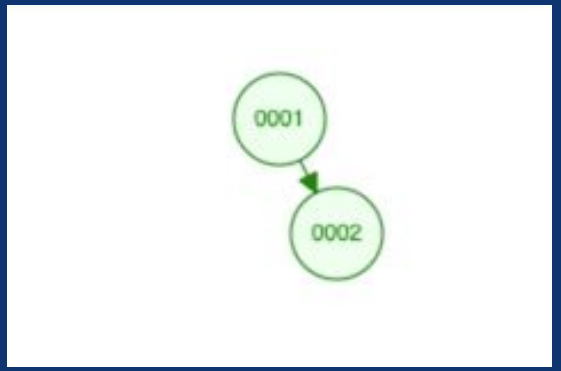


Parent node becomes right child of current node.

ZAG

Occurs when:

- No grandparent node exists
- Current node is a right child

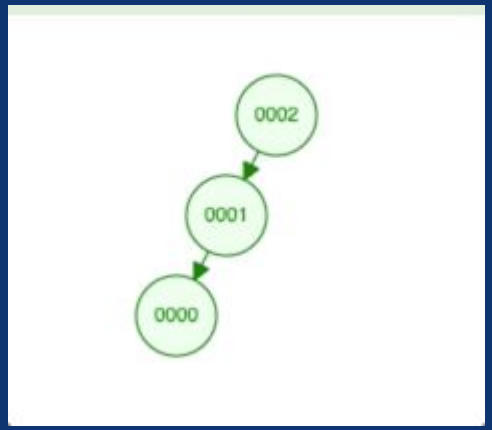


Parent node becomes left child of current node.

ZIG-ZIG

Occurs when:

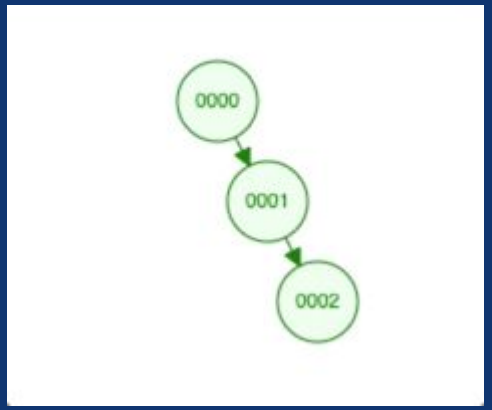
- Current node is a left child
- Current node's parent is also a left child



ZAG-ZAG

Occurs when:

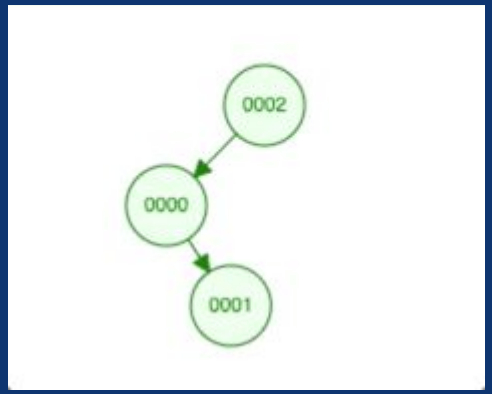
- Current node is a right child
- Current node's parent is also a right child



ZIG-ZAG

Occurs when:

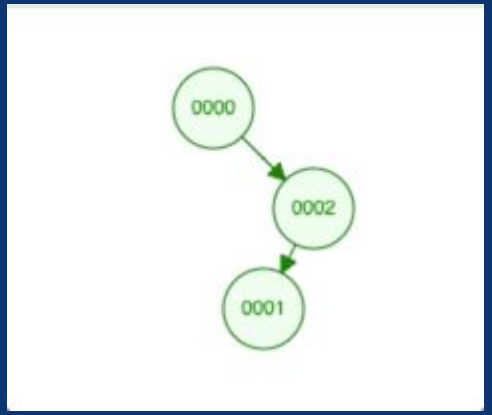
- Current node is a right child
- Current node's parent is a left child



ZAG-ZIG

Occurs when:

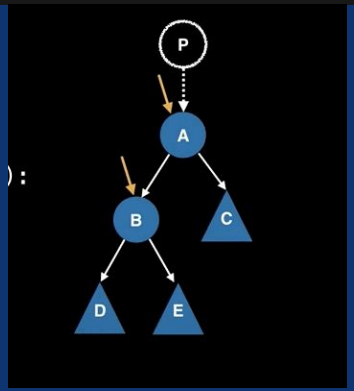
- Current node is a left child
- Current node's parent is a right child



ZIG AND ZAG IMPLEMENTATION

```
//Zig rotation of tree to take the left child node and making it the parent node
    node* rightRotate(node* curr) { //takes in the memory address of the node we give it
        node* leftChild = curr->left; //creates a node called leftChild giving it the memory address of "curr" left child
        curr->left = leftChild->right; //now taking the curr left and setting it to leftChilds right child
        leftChild->right = curr; //leftChilds right child is now the current node at the top
        return leftChild;
    }
```

```
//Zag rotation of tree to take the left child node and making it the parent node
    node* leftRotate(node* curr) { //takes in the memory address of the node we give it
        node* rightChild = curr->right;//creates a node called rightChild giving it the memory address of "curr" right child
        curr->right = rightChild->left;//now taking the curr right and setting it to RightChilds left child
        rightChild->left = curr;//leftChilds right child is now the current node at the top
        return rightChild;
    }
```



OPERATIONS

Worst case: n = # of elements
If nodes are accessed in ascending order, the complexity of splay tree operations would be O(n)

Otherwise; splay, search, insert, and delete boast an amortized time complexity of O(log n)

Search():

- Apply basic binary search
- If key is found, splay key to root
- Else, splay last accessed leaf node to root

OPERATIONS (CONT.)

Insert():

- If the root is NULL, create a new node and return it as root.
- Bring the closest leaf node up to the root
- If new root’s key is same as k, don’t do anything as k is already present.
- Else, create a new node
- If root's key is greater, make root the right child of new node and copy the left child of root to newnode
- If root's key is smaller, make root the left child of newnode and copy the right child of root to newnode
- Return new node, new node becomes root

OPERATIONS (CNT.)

deleteNode():

- If the current node is NULL then return the current node
- Enter comparisons of key value to child nodes of the current node
  - If the key is greater than, call deleteNode again with current nodes right child
  - If the key is less than, call deleteNode again with current nodes left child
- Repeat this cycle until either:
  - Both left and right children of parent are NULL (return NULL)
  - Left Child is NULL (return right child)
  - Right Child is NULL (return left child)
- At which point we can return and delete our current node