CS32 Week 6: Trees & BSTs

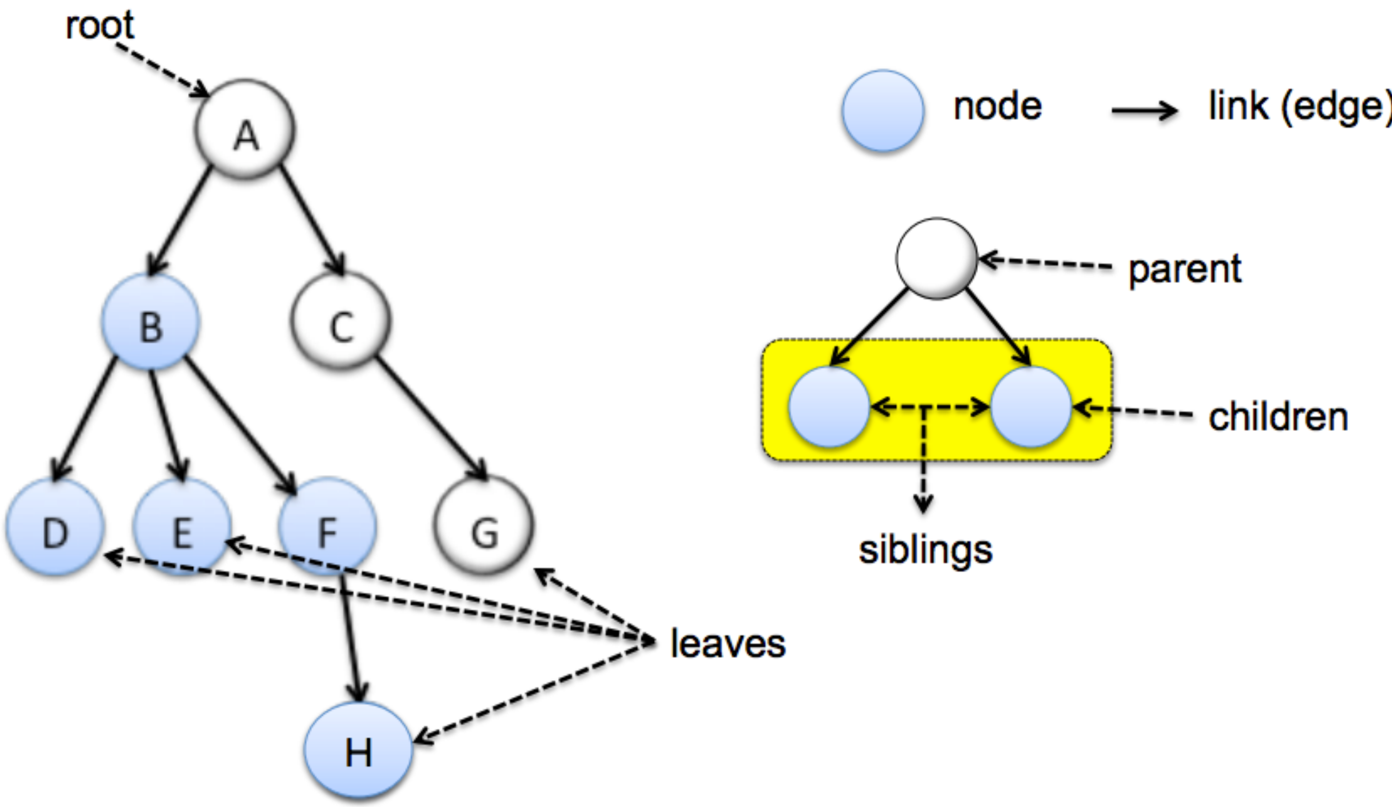
Doga Kisa

Tree

- Common data structure in computer science
  - Organizing data hierarchy
  - Make decisions – decision tree
  - Fast retrieval – binary search tree

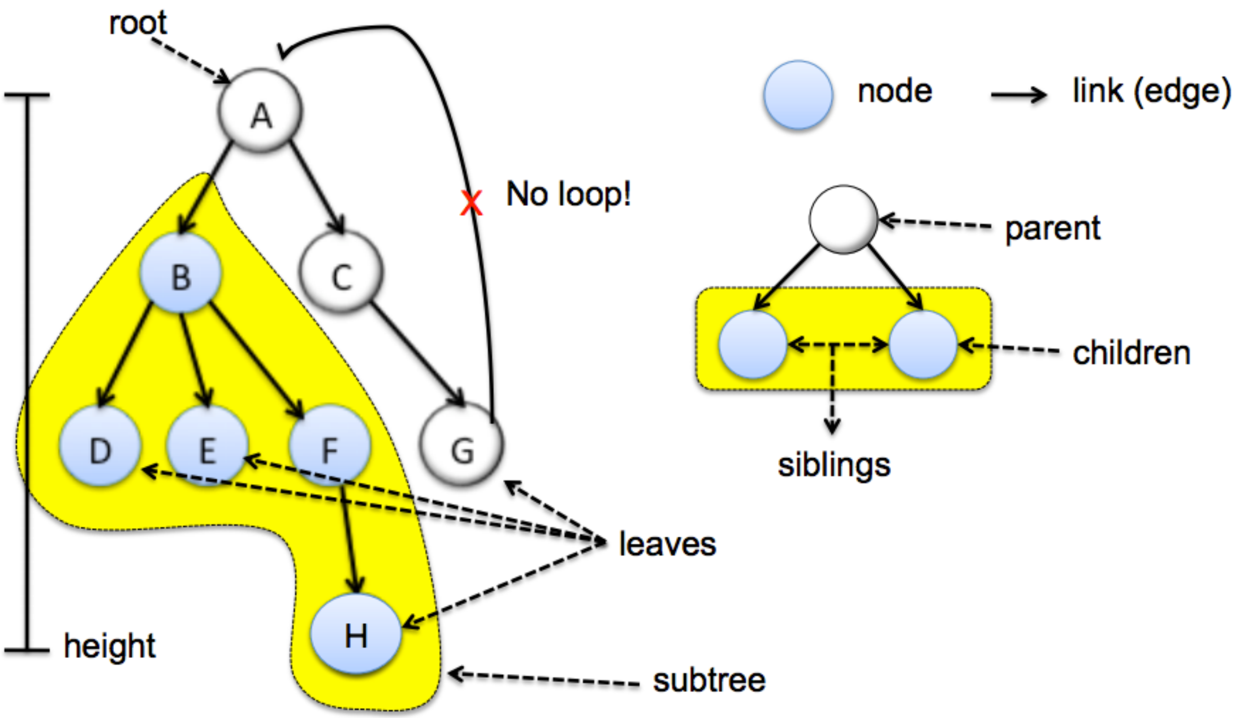
Tree

- Concepts
  - root
  - leaf
  - parent
  - children
  - sibling
  - ancestor
  - descendant
- Directed link
  - Only link to children, no link to parent
  - Only one parent for each node (except for root)



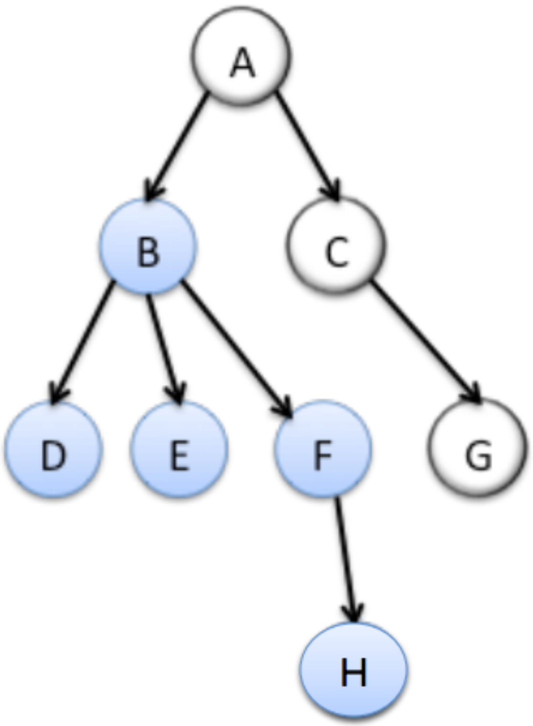
Tree

- Concepts
  - All nodes are connected.
  - No loops.
  - height/depth
  - subtree



Tree

- How many edges should there be in a tree of n nodes?
  - n-1
  - Each node has a link pointing to it
    - Except root node.
  - Proof of at least n-1 edges:
    - All nodes are connected.
  - Proof of at most n-1 edges:
    - A graph with n nodes and n edges
    - have to have a loop.



Tree

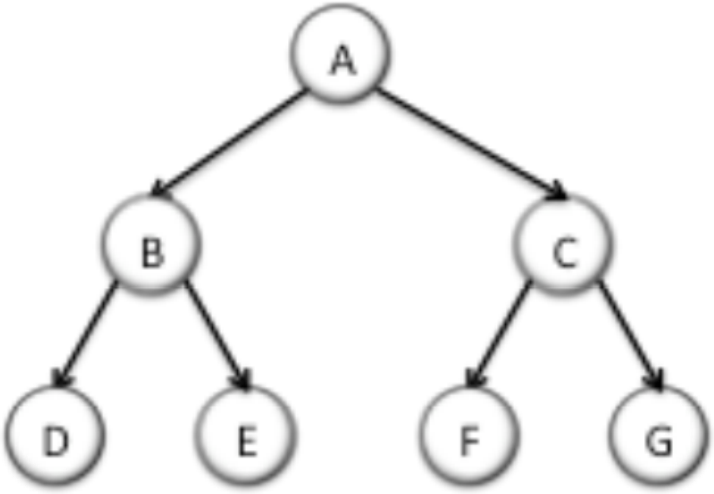
- Binary Tree

  Each internal (non-leaf) node has at most 2 children
- Full Binary Tree

  Each internal node has exactly 2 children
- Perfect Binary Tree

  Full binary tree, in which

  all leaves are at the same depth
  - For h: height, what is n?
    - 2^(h+1) - 1



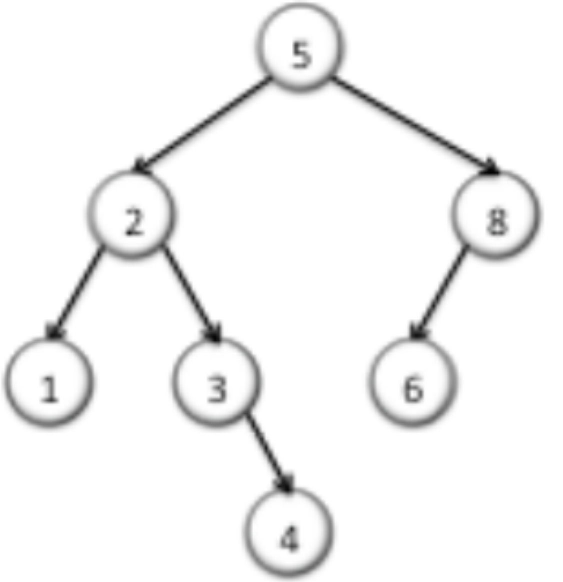
Tree

- Tree traversal
  - Pre-order
    - current, left, right
  - In-order
    - left, current, right
  - Post-order
    - left, right, current

Tree

- Tree traversal
  - What's the output for
    - Pre-order traversal
    - In-order traversal
    - Post-order traversal

    - pre: 5213486
    - in: 1234568
    - post: 1432685

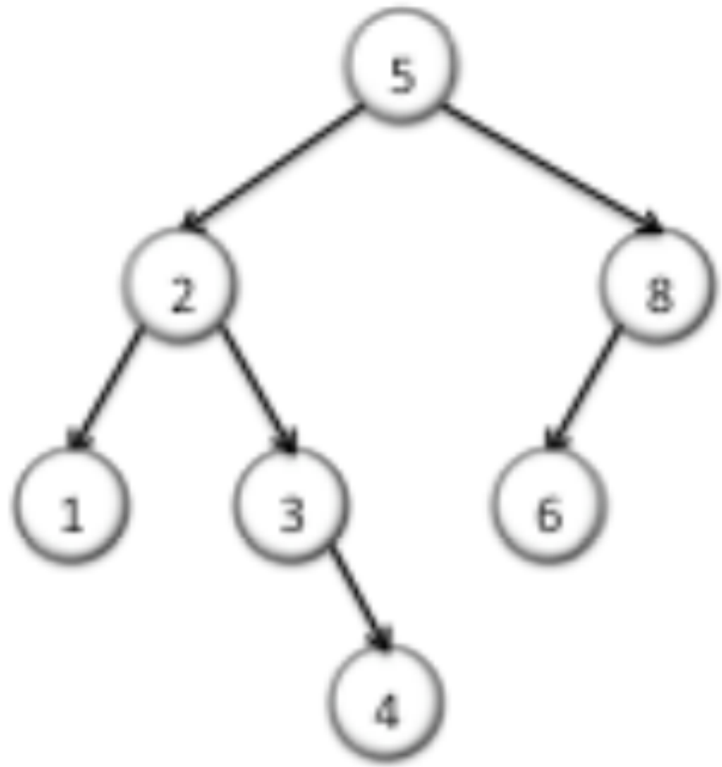


Binary Search Tree

- Definition

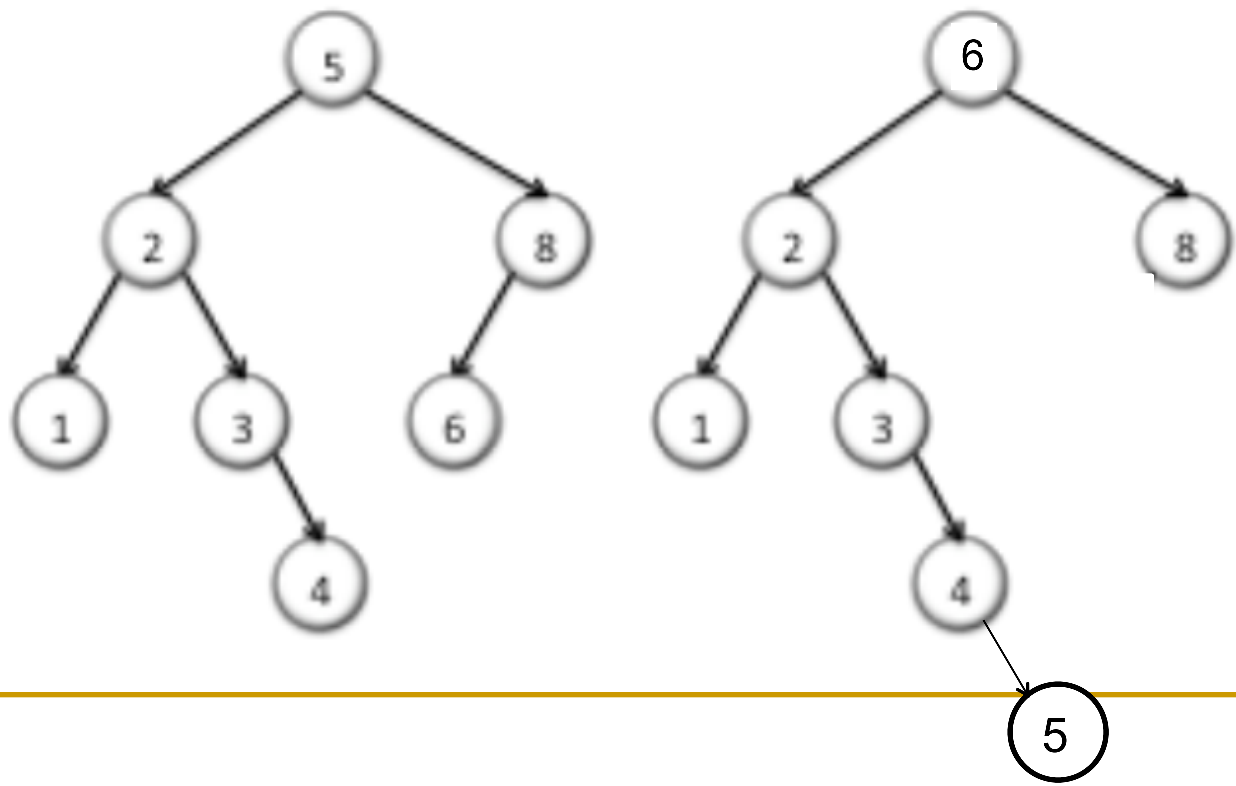
  For any node with value *v*,

  - all nodes on the left subtree have smaller values than *v*,
  - all nodes on the right subtree have larger values than *v*.



Binary Search Tree

- For a same set of data, there exists multiple binary search trees.



Binary Search Tree

- Structure for storing data
  - Pros and cons

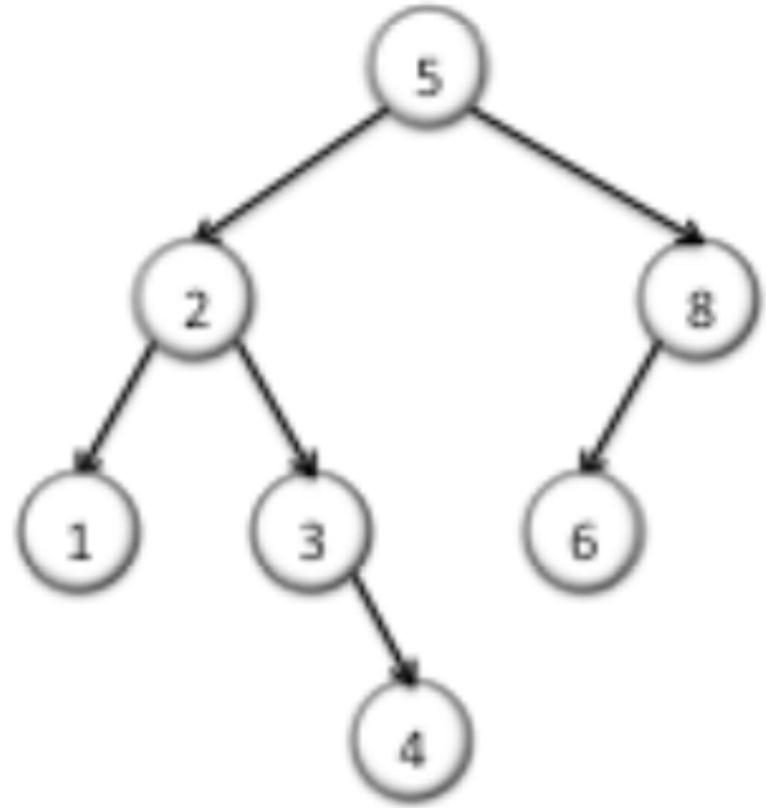
  - Operations:
    - Search
    - Insert
    - Delete
    - Traversal

Binary Search Tree

- Search for a value
  - At the root, compare the value *v* of current node with desired value *x*
    - If $x = v$, found it!
    - If $x < v$
      - Search for x in the **left** subtree
    - If $x > v$
      - Search for x in the **right** subtree

Binary Search Tree

- Search
  - Search for 6
  - Search for 3
  - Search for 9



  - Complexity
    - For balanced binary search
      - O(log n)
    - What affects the performance of search?
      - Tree height!

Binary Search Tree

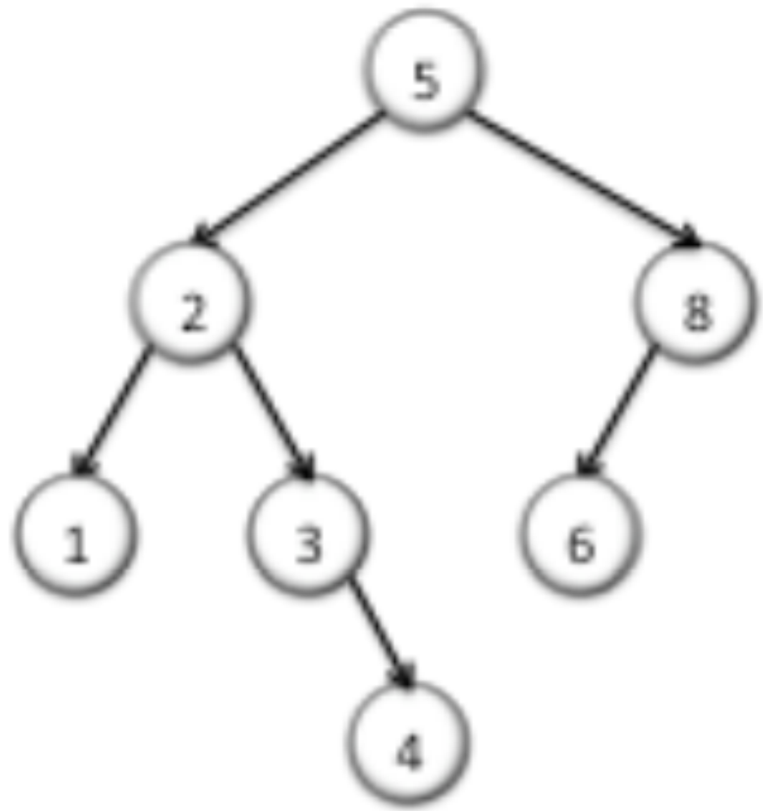
- Search
  - Using binary search on a sorted array
    - Complexity?
      - O(log n)
    - Why use a binary search tree?
      - More flexibility for modification – insertion/ deletion

Binary Search Tree

- Insertion
  - Put the new value in the correct position
  - First do a search
    - Until we hit the NULL pointer
  - Create a new Node
    - Put it at the position of the NULL pointer
    - Fix the link

Binary Search Tree

- Insertion
  - Insert 7
  - Insert 0
- Complexity?
  - Search + insert
  - O(log n) + O(1)
  - = O(log n)



Binary Search Tree

- Insertion
  - Given a sorted array of integers
  - How to construct a BST?
    - Insertion by linear order?
    - Insertion by random order?
    - Pick the middle one as the root?

Binary Search Tree

- Challenge:
  - Given a sorted array, build the best BST.
  - Node* buildBST(int array[], int size);
  - Hint: use recursion

```
struct Node {
    int m_value;
    Node* m_left;
    Node* m_right;
    Node(int val){
        m_value = val;
        m_left = NULL;
        m_right = NULL;
    }
};
```

Binary Search Tree

- Challenge:
  - Given a sorted array, build the best BST.
  - Node* buildBST(int array[], int size);
  - {
  -     return buildBST(array, 0, size-1);
  - }

  - `Node* buildBST(int array[], int start, int end)`

Binary Search Tree

- Challenge:
  - Given a sorted array, build the best BST.
  - `Node* buildBST(int array[], int start, int end)`
  - Recursion
    - How to break down the problem?
      - Use first half to construct left subtree
      - Use second half to construct right subtree
    - How to merge results?
      - Use middle one to create a node, link left and right subtree
    - Base case?
      - When start > end.

Binary Search Tree

- Challenge:
  - Given a sorted array, build the best BST.

```
Node* buildBST(int array[], int start, int end)
{
    if (start > end) return NULL;

    int mid = (start + end) / 2;
    Node* root = new Node(array[mid]);
    Node* left = buildBST(array, start, mid - 1);
    Node* right = buildBST(array, mid + 1, end);
    root->m_left = left;
    root->m_right = right;
    return root;
}
```

Binary Search Tree

- Traversal
  - Basic tree traversal techniques
    - Pre-order
    - Post-order
    - In-order
  - Which one is special for BST?
    - In-order traversal produce the values in order.

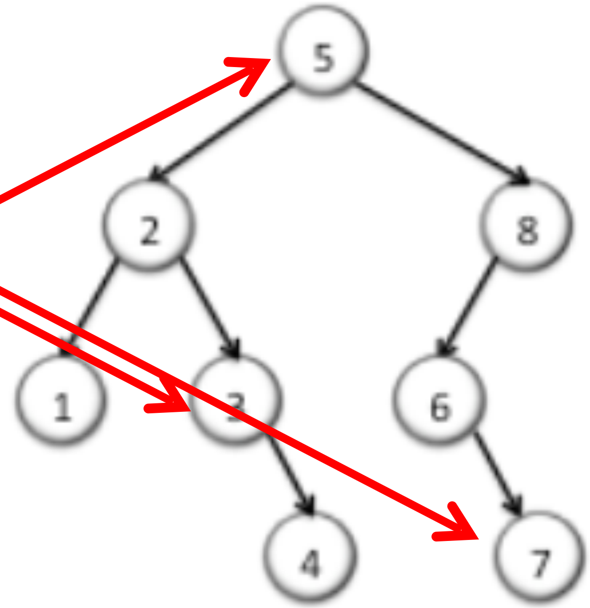
Binary Search Tree

- Deletion
  - Remove one node from the tree
    - Fix links, move node if necessary
  - Tree still remain a valid BST

  - Run a search first and find the node to delete
  - Delete the node
    - 3 cases

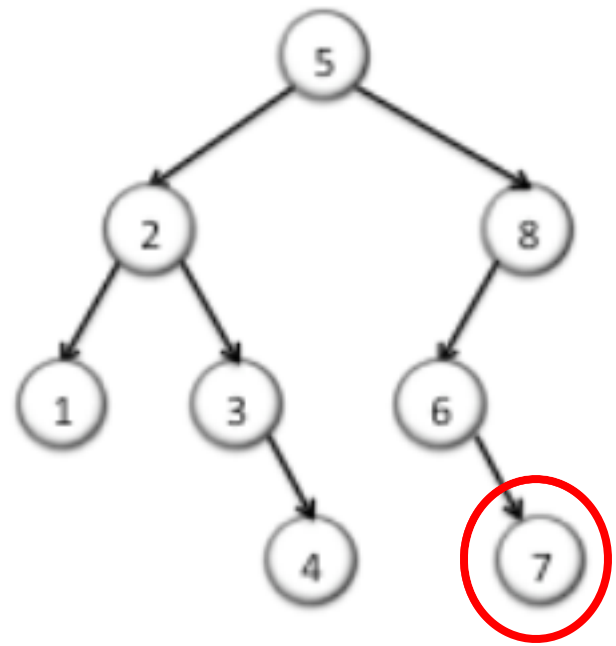
Binary Search Tree

- Deletion
  - 3 cases
    - The node is a leaf
    - The node has one child
    - The node has two children



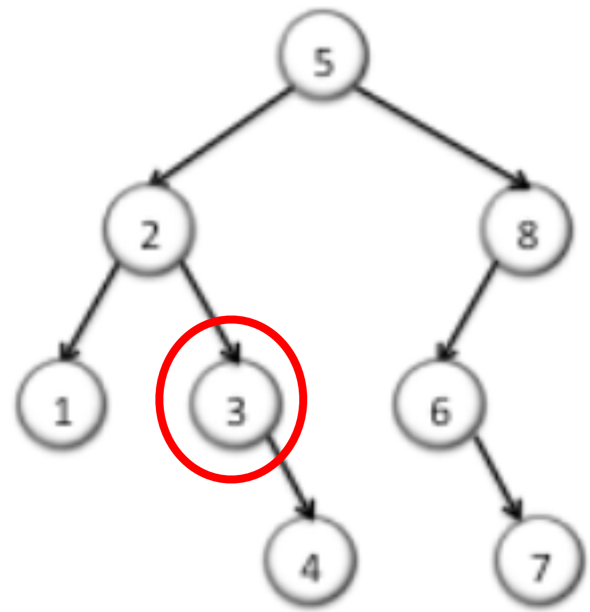
Binary Search Tree

- Deletion
  - The node is a leaf
    - Delete the node
    - Set the parent's child pointer to NULL



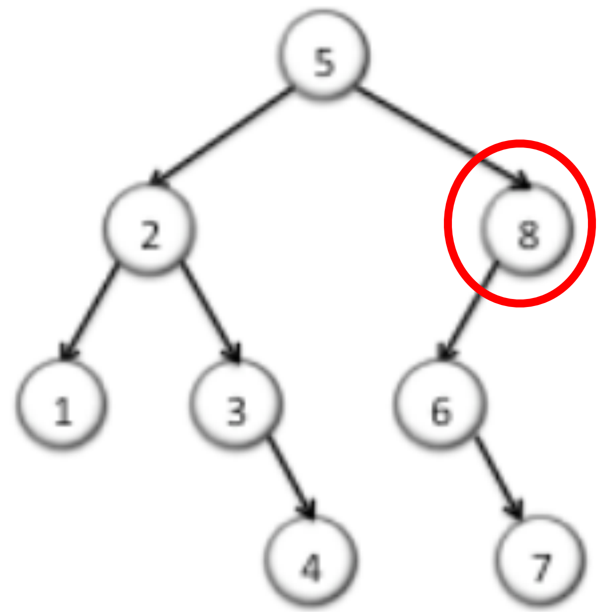
Binary Search Tree

- Deletion
  - The node has one child
    - Link the *parent* to *child*
      - If node is a left child, then
        - *parent->left = child;*
      - If node is a right child, then
        - *parent->right = child;*
    - Order remains the same.



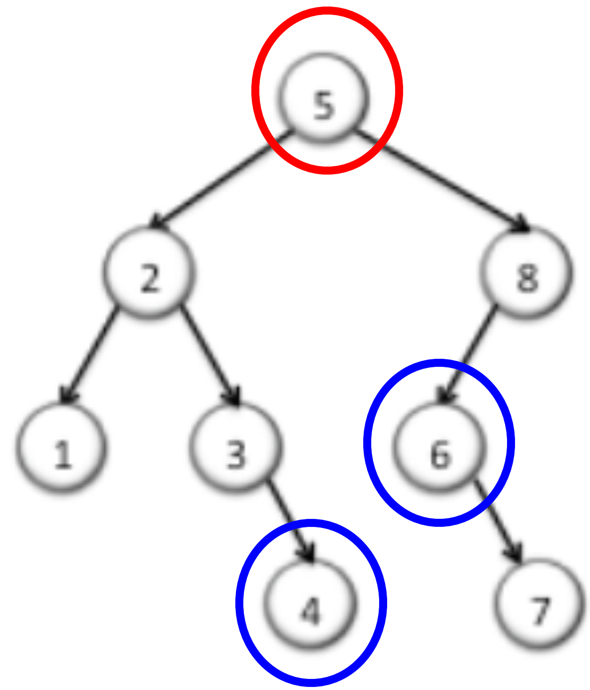
Binary Search Tree

- Deletion
  - The node has one child
    - Link the *parent* to *child*
      - If node is a left child, then
        - *parent->left = child;*
      - If node is a right child, then
        - *parent->right = child;*
    - Order remains the same.



Binary Search Tree

- Deletion
  - The node has two children
    - Find a replacement
      - The one next to current node's value from left or right
      - Largest one in the left subtree
      - Smallest one in the right subtree
      - This node has no children or only one child.
    - Update the value of the current node
    - Delete the replacement node (use case 1 or 2)



Binary Search Tree

- Height of a BST?
  - int getHeight(Node* p){ }

  - **height** of a tree: the length of the longest path from the root to a leaf.
  - Hint: use recursion

Binary Search Tree

- Height of a BST?

```
int getHeight(Node* p)
{
    if (p == NULL)
        return -1;

    int left_height = getHeight(p->m_left);
    int right_height = getHeight(p->m_right);
    if (left_height > right_height)
        return left_height + 1;
    else
        return right_height + 1;
}
```

Binary Search Tree

- Find the max in BST
  - int GetMax(Node* p){ }

```
int GetMax(node *p)
{
  if (p == NULL)          // empty
    return(INT_MIN);
  while (p->right != NULL)
     p = p->right;

  return(p->value);
}
```

Binary Search Tree

- Find the max in a binary tree
  - int GetMax(Node* p){ }

Binary Search Tree

```
int GetMax(Node* p)
{
    if (p == NULL) return INT_MIN; // empty

    int max = p->m_value;

    int left_max = GetMax(p->m_left);
    if (left_max > max)
        max = left_max;
    int right_max = GetMax(p->m_right);
    if (right_max > max)
        max = right_max;

    return max;
}
```

Balanced Binary Search Tree

- Balanced
  - for any node, the difference between height for left subtree and right subtree is at most 1.
- Height affects the performance of searching for BST.
  - a balanced tree will have the smallest height among the BSTs.
  - Rotation techniques.
  - AVL tree, etc.