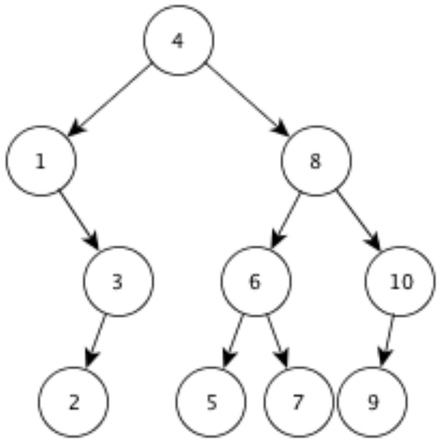
CECS 277 – Lecture 10 – Binary Search Trees

Binary Search Trees (BST) are great for quickly (O(log n)) adding, removing, and searching for data. However, they are only useful as long as position does not matter, and the insertion of nodes is not going to be ordered. Binary Search Trees are binary because each node in the tree can only have up to two children.

Nodes – A binary search tree is made up of nodes. The first node in the tree is called the root. Each node is made up of data, its two children, and usually its parent.

Inserting Nodes – When you are inserting the first node into the tree, it is automatically placed as the root. Each subsequent node is compared to the nodes already in the tree: If the value of the node being inserted is less than or equal to the value of the node that is already in the tree, then it is moved to left of that node, otherwise the right. This continues until an empty spot in the tree is located, which is where it is finally placed. Avoid adding nodes in order, this leads to having an unbalanced tree that eliminates the benefits of using a BST.

Example: Given the following values: 4, 8, 6, 1, 3, 10, 7, 2, 5, 9.



Searching the Tree – Since the tree is stored in an ordered fashion, it makes searching for a particular value, quick and easy. If the value that you are searching for is less than the node you're at, then move left, otherwise move right. If the value does not exist, it will find a null node at the location it expects it to be at, and you can return a null value, or state that the value is not in the tree.

Removing a Node – There are three possible cases for removing a node:

1. Removing a node with no children – remove the node
2. Removing a node with one child – remove the node and replace it with its child.
3. Removing a node with two children – replace the value with either the left-most child of its right subtree, or the right-most child of its left subtree. Then remove that node using either case 1 or 2. Avoid always using the left-most or right-most child, this can cause the tree to become unbalanced.

Traversing a Binary Search Tree – Visiting the nodes in order works as follows:

1. Move down the left of the tree as far as possible.
2. Access the node.
3. If there is a right node, move to the right. If not, go up a level, access the node, and then move to the right.
4. Repeat from step 1.

Example:

```
public class Node {
	private int data;
	private Node left;
	private Node right;
	private Node parent;
	public Node(int d) {
		data = d;
		left = null;
		right = null;
		parent = null;
	}
	public void setData(int d) {
		data = d;
	}
	public int getData() {
		return data;
	}
	public void setLeft(Node l) {
		left = l;
	}
	public Node getLeft() {
		return left;
	}
	public void setRight(Node r) {
		right = r;
	}
	public Node getRight() {
		return right;
	}
	public void setParent(Node p) {
		parent = p;
	}
	public Node getParent() {
		return parent;
	}
}
```

```
//Binary Search Tree Class
//Add, Remove, Search, and Print methods included
public class BST {
     private Node root;
     public BST() {
          root = null;
     }

     //Add a new node
     public void add(int d) {
          if (root == null) {
               root = new Node(d);
          } else {
               add(d, root);
          }
     }
     private void add(int d, Node n) {
          if (d <= n.getData()) {
               if (n.getLeft() == null) {
                    n.setLeft(new Node(d));
                    n.getLeft().setParent(n);
                    return;
               }
               add(d, n.getLeft());
          } else {
               if (n.getRight() == null) {
                    n.setRight(new Node(d));
                    n.getRight().setParent(n);
                    return;
               }
               add(d, n.getRight());
          }

     }


     //Remove a node
     public void remove(int d) {
          if (root == null) {
               System.out.println("No items to remove");
          } else {
               remove(d, root);
          }
     }
```

```java
private void remove(int d, Node n) {
	if (n == null) {
		System.out.println("Item Not Found");
	} else if (d < n.getData()) {
		remove(d, n.getLeft());
	} else if (d > n.getData()) {
		remove(d, n.getRight());
	} else {// found location
		if(n.getLeft()!= null && n.getRight() != null){
			int rand = (int) (Math.random() * 2);
			if (rand == 0) {// right-most child of left
				Node c = n.getLeft();
				while (c.getRight() != null) {
					c = c.getRight();
				}
				n.setData(c.getData());
				remove(c.getData(), c);
			} else {// left-most child of right tree
				Node c = n.getRight();
				while (c.getLeft() != null) {
					c = c.getLeft();
				}
				n.setData(c.getData());
				remove(c.getData(), c);
			}
		} else if (n.getLeft() != null) {// has LChild
			Node p = n.getParent();
			if(p == null){
				root = n.getLeft();
				n.getLeft().setParent(null);
			}else{
				n.getLeft().setParent(n.getParent());
				if (p.getLeft() == null) {
					p.setRight(n.getLeft());
				} else {
					if (p.getLeft() == n) {
						p.setLeft(n.getLeft());
					} else {
						p.setRight(n.getLeft());
					}
				}
			}
		} else if (n.getRight() != null) {// has RChild
			Node p = n.getParent();
```

```java
                if(p == null){
                    root = n.getRight();
                    n.getRight().setParent(null);
                }else{
                    n.getRight().setParent(n.getParent());
                    if (p.getLeft() == null) {
                        p.setRight(n.getRight());
                    } else {
                        if (p.getLeft() == n) {
                            p.setLeft(n.getRight());
                        } else {
                            p.setRight(n.getRight());
                        }
                    }
                }
            }else if( n.getLeft() == null &&
                        n.getRight() == null ) {
                Node p = n.getParent();
                if(p == null){
                    root = null;
                }else{
                    if (p.getLeft() == null) {
                        p.setRight(null);
                    } else {
                        if (p.getLeft() == n) {
                            p.setLeft(null);
                        } else {
                            p.setRight(null);
                        }
                    }
                }
            }
        }
}

//Search the tree for a particular value
public Node search(int d) {
    if (root == null) {
        System.out.println("No items to search");
        return null;
    } else {
        return search(d, root);
    }
}
```

```
    private Node search(int d, Node n) {
        if (n.getData() == d) {
            return n;
        }
        if (d <= n.getData()) {
            if (n.getLeft() == null) {
                System.out.println("Item Not Found");
                return null;
            } else {
                return search(d, n.getLeft());
            }
        } else {
            if (n.getRight() == null) {
                System.out.println("Item Not Found");
                return null;
            } else {
                return search(d, n.getRight());
            }
        }
    }

    //Print the tree in order
    public void printBST() {
        if (root == null) {
            System.out.println("No items to print");
        } else {
            printBST(root);
        }
    }
    private void printBST(Node n) {
        if (n.getLeft() != null) {
            printBST(n.getLeft());
        }
        System.out.println(n.getData());
        if (n.getRight() != null) {
            printBST(n.getRight());
        }
    }
}
```

```java
//Main
import java.util.Scanner;
public class TestBST {
     public static void main(String[] args) {
          Scanner in = new Scanner(System.in);
          BST bTree = new BST();
          System.out.print("Added: ");
          for (int i = 0; i < 10; i++) {
               int rand = (int) (Math.random() * 100) + 1;
               bTree.add(rand);
               System.out.print(rand+" ");
          }
          System.out.println();
          bTree.printBST();
          System.out.print("Enter Value to Find: ");
          int val = in.nextInt();
          if (bTree.search(val) == null) {
               System.out.println("Value not found");
          } else {
               System.out.println("Value was found");
          }
          System.out.print("Enter Value to Remove: ");
          int rem = in.nextInt();
          bTree.remove(rem);
          bTree.printBST();
     }
}
```