Examination and Report on Binary Search Trees (BST)

Oberon Ilano

Assignment Description

Investigate how the height of a Binary Search Trees (BST) and the number of nodes in the tree are related. Show supporting evidence of the procedure and result findings of the experiment hypothesis.

Performed Date: December 1, 2018

Due Date: December 4, 2018

CS 255 – Computer Science II

Fall 2018

Dr. Janet Jenkins

*1. Introduction*

A binary search tree (BST) is a data structure that stores numbers of elements. BST is a rooted tree that can be used for sorting and searching data from a static data structure. In the first element, a root node is created, and every parent has at most two children. The following elements are designated either to the left child or the right child and cannot be both at the same time. There is only, at the most one left child or one right child. The left and right subtrees are also binary search trees themselves. The following elements are also being placed according to their value if they are greater or less than the element in the root node. The value in the right child of a node if it exists, is greater than the value in its parent node. The value in the left child of a node if exists is less than the value in its parent node. If any specified parent in a binary tree, and if the parent has a right child then the right subtree of the parent is the binary tree whose root is the right child of the parent, whose vertices be made up of the right child of the parent and all its children, and whose edges of binary tree that link the vertices of the right subtree. And with what has happened to the right is the same that will happen to the left. A full binary search tree is when each parent has exactly two children. The height of the binary tree is the longest or the deepest path from the root node to the leaf. The height is equivalent to the log(n), where n is the number of leaf nodes and the log is the log to the base K, where K is the maximum number of the children allowed in every node. The maximum comparison to search from BST is equivalent to its height.

Given the hypothesis that some numbers being inserted from smallest to the biggest value of the node such as -1, 0, 1, 4, and 10. Inserting biggest to the smallest value of the nodes such as 143, 91, 87, 31, and 30. Lastly, inserting a different sequence with random value of nodes and defining if it would be either balance or unbalance. What would be the relationships of the height of a BST to the number of nodes?

*2. Experimental Methodology*

Depending on the sequence, the node value, and how many are being inserted will determine the outcome of the shape of the tree and the height of a BST. Suggesting an algorithm that balances a BST, becomes left-skewed or right-skewed. If the insertion is random then there are four possible outcomes that the tree shape can be. The illustrations below represented the different possible shapes of a tree.

Figure 1 Figure 2

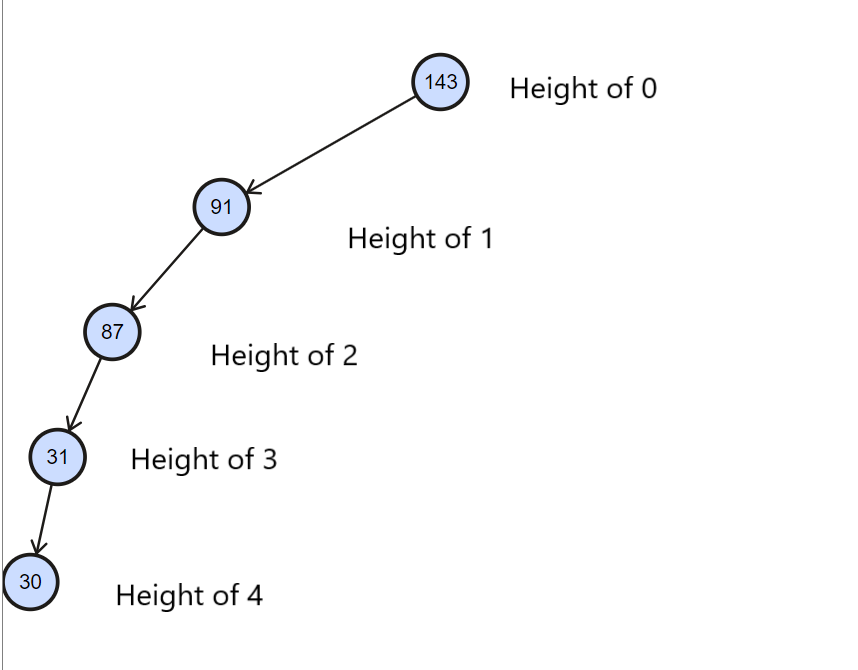
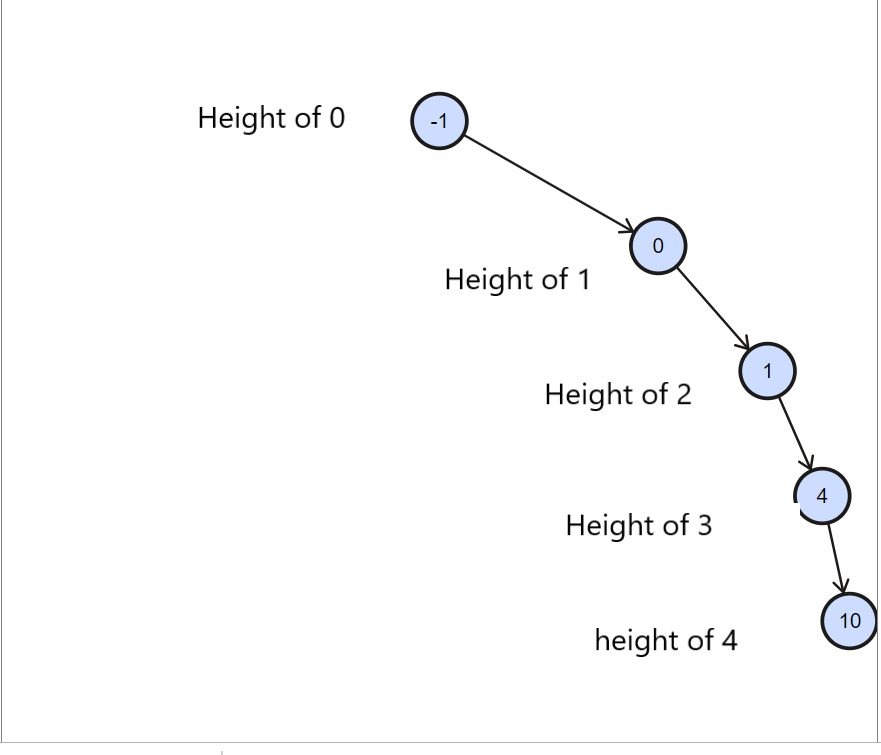
 

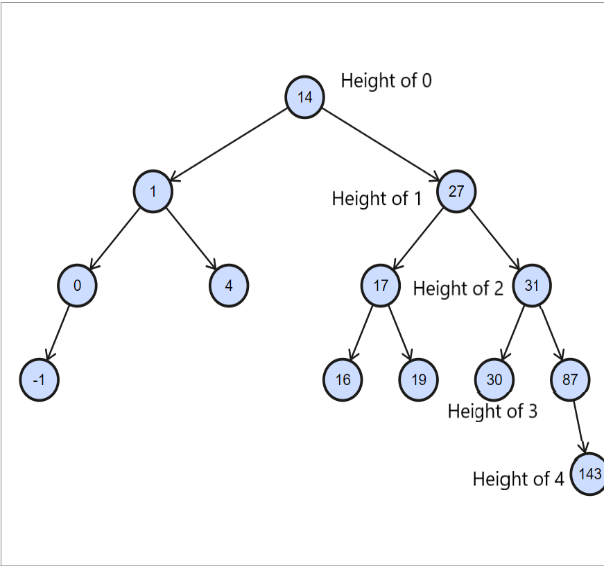
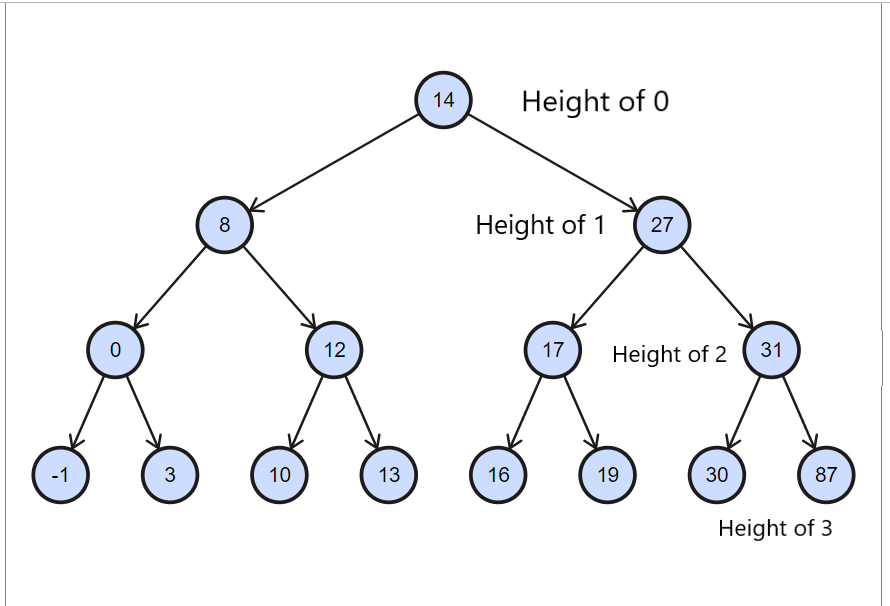
Figure 3 Figure 4 

Figure 1

The root has a value of 143, if 91 was less than the root (143) then 91 was assigned to the left child. If 87 was less than the root (143) and the left sub root (91) then 87 was assigned to the left child of the left sub root 91. If 31 was less than the root (143), left sub root (91), and the left sub root (87) then 31 was assigned to the left child of the left sub root 87. Lastly, if 30 was less than the root (143), left sub root (91), left sub root (87), and the left sub root (31) then 30 was assigned to the left child of left sub root 31. Figure 1 has the maximum height of , when the number of nodes was

The code down below was applied to find the height of Figure 1.

First the root, BNode\* left, and BNode\* right is all equal to null.

*BNode::BNode(): left(NULL), right(NULL){}*

*BST::BST(): root(NULL){}*

Then inserting all the node values in a sequence into a linked list in the main driver.

*int main(){*

*int height, depth;*

*BST b1; b1.Insert(143); b1.Insert(91); b1.Insert(87); b1.Insert(31); b1.Insert(30);*

I displayed the count of the nodes in BST.

*cout << "Count of Nodes" << endl; cout << count << endl;*

To find the height, I set up a height variable to the height function then displayed the height.

*height = b1.BSTMaxHeight();*

*cout << "BST Height" << endl; cout << height << endl;*

*cout << "BST Height path" << endl;*

I printed the nodes that corresponds to the path of the longest or deepest leaf.

*b1.PrintMaxHeight(); cout << endl;*

*return 0;*

*}*

Figure 2

The root has a value of -1, if 0 was greater than the root (-1) then 0 was assigned to right child. If 1 was greater than the root (-1), and the right sub root (0) then 1 was assigned to the right child of the right sub root 0. If 4 was greater than the root (-1), right sub root (0) , and the right sub root (1) then 4 was assigned to the right child of the right sub root 1. Lastly, if 10 was greater than the root (-1), the right sub root (0), right sub root (1), and the right sub root (4) then 10 was assigned to the right child of the right sub root 4. Figure 2 has the maximum height of , when the number of nodes was .

The code down below was applied to find the height of Figure 2.

First the root, BNode\* left, and BNode\* right is all equal to null.

*BNode::BNode(): left(NULL), right(NULL){}*

*BST::BST(): root(NULL){}*

Then inserting all the node values in a sequence into a linked list in the main driver.

*int main(){*

*int height, depth;*

*BST b1; b1.Insert(-1); b1.Insert(0); b1.Insert(1); b1.Insert(4); b1.Insert(10);*

I displayed the count of the nodes in BST.

*cout << "Count of Nodes" << endl; cout << count << endl;*

To find the height, I set up a height variable to the height function then displayed the height.

*height = b1.BSTMaxHeight();*

*cout << "BST Height" << endl; cout << height << endl;*

*cout << "BST Height path" << endl;*

I printed the nodes that corresponds to the path of the longest or deepest leaf.

*b1.PrintMaxHeight(); cout << endl;*

*return 0;*

*}*

Figure 3

The root has a value of 14, if 27 is greater than the root (14) then 27 was assigned to right child. If 17 was greater than the root (14) but less than the right child (27) then 17 was assigned to the left child of the right sub root (27). If 19 is greater than the root (14), less than right sub root (27), but greater than sub root (17) then 19 was assigned to the right child of sub root (17). If 31 is greater than the root (14), greater than right sub root (27) then 31 was assigned to the right child of the right sub root (27). If 1 is less than the root (14) then 1 was assigned to left child. If 4 is less than the root (14) but greater than left child (1) then 4 was assigned to the right child of left sub root (1). If 30 is greater than root (14), greater than right sub root (27), but less than the right sub root (31) then 30 was assigned to the left child of 31. If 16 is greater than the root (14), less than the right sub root (27) and less than sub root (17) then 16 was assigned to the left child of sub root (17). If 87 is greater than root (14), right sub root (27), right sub root (31) then 87 was assigned to the right child of right sub root (31). If 0 is less than the root (14), left sub root (1) then 0 was assigned to the left child of left sub root (1). If 143 is greater than root (14), right sub root (27), right sub root (31), and right sub root (87) then 143 was assigned to the right child of right sub root (87). Lastly, if -1 is less than the root (14), left sub root (1), left sub root (0) then -1 was assigned to the left child of left sub root (0). Figure 3 has the height of 4.

The code down below was applied to find the height of Figure 3.

First the root, BNode\* left, and BNode\* right is all equal to null.

*BNode::BNode(): left(NULL), right(NULL){}*

*BST::BST(): root(NULL){}*

Then inserting all the node values in a sequence into a linked list in the main driver.

*int main(){*

*int height, depth;*

*BST b1; b1.Insert(14); b1.Insert(27); b1.Insert(17); b1.Insert(19); b1.Insert(31);*

*b1.Insert(1); b1.Insert(4); b1.Insert(30); b1.Insert(16); b1.Insert(87); b1.Insert(0); b1.Insert(143); b1.Insert(-1);*

I displayed the count of the nodes in BST.

*cout << "Count of Nodes" << endl; cout << count << endl;*

To find the height, I set height variable to the height function then displayed the height.

*height = b1.BSTMaxHeight();*

*cout << "BST Height" << endl; cout << height << endl;*

*cout << "BST Height path" << endl;*

I printed the nodes that corresponds to the path of the longest or deepest leaf.

*b1.PrintMaxHeight(); cout << endl;*

*return 0;*

*}*

Figure 4

The root has a value of 14, if 8 is less than the root (14) then 8 was assigned to the left child. If 12 is less than the root (14) but greater than the left child (8) then 12 was assigned to the right child of left sub root (8). If 0 is less than the root (14), left sub root (8) then 0 was assigned to the left child of left sub root (8). If 27 is greater than root (14) then 27 was assigned to the right child. If 31 is greater than the root (14) and the right sub root (27) then 31 was assigned to the right child of right sub root (27). If 10 is less than the root (14), greater than left sub root (8), but less than the sub root (12) then 10 is assigned to the left child of sub root (12). If -1 is less than If the root (14), left sub root (8), and left sub root (0) then -1 was assigned to the left child of left sub root (0). If 17 is greater than the root (14), but less than the right sub root (27) then 17 was assigned to the left child of right sub root (27). If 16 is greater the root (14), less than the right sub root (27), and less than the sub root (17) then 16 was assigned to the left child of sub root (17). If 19 is greater than the root (14), less than the right sub root (27), but greater than the sub root (17) then 19 was assigned to the right child of sub root (17). If 3 I less than the root (14), left sub root (8) but greater than left sub root (0) then 3 was assigned to the right child of left sub root (0). If 87 is greater than the root (14), right sub root (27), right sub root (31) then 87 was assigned to the right child of right sub root (31). If 30 is greater than the root (14), right sub root (27) but less than right sub root (31) then 30 was assigned to the left child of the right sub root (31). If 13 is less than the root (14), greater than left sub root (8), sub root (12) then 13 was assigned to the right child of sub root (12). Figure 4 has a height of 3.

The code down below was applied to find the height of Figure 4.

First the root, BNode\* left, and BNode\* right is all equal to null.

*BNode::BNode(): left(NULL), right(NULL){}*

*BST::BST(): root(NULL){}*

Then inserting all the node values in a sequence into a linked list in the main driver.

*int main(){*

*int height, depth;*

*BST b1; b1.Insert(14); b1.Insert(8); b1.Insert(12); b1.Insert(0); b1.Insert(27); b1.Insert(31); b1.Insert(10); b1.Insert(-1); b1.Insert(17); b1.Insert(16); b1.Insert(19); b1.Insert(3); b1.Insert(87); b1.Insert(30); b1.Insert(13);*

Displayed the count of the nodes in BST.

*cout << "Count of Nodes" << endl; cout << count << endl;*

To find the height, I set up a height variable to the height function then displayed the height.

*height = b1.BSTMaxHeight();*

*cout << "BST Height" << endl; cout << height << endl;*

*cout << "BST Height path" << endl;*

I printed the nodes that corresponds to the path of the longest or deepest leaf.

*b1.PrintMaxHeight(); cout << endl;*

*return 0;*

*}*

I set up the following function code below that allowed me to determine the height of a BST

*int BST::BSTMaxHeight(){*

*return MaxHeightHelper(root);*

*}*

I set up a function helper that returned and helped determined if either left or right has the longest leaf.

*int BST::MaxHeightHelper(BNode\* subroot){*

*if(subroot){*

*return 1 + max(MaxHeightHelper(subroot->left), MaxHeightHelper(subroot->right));*

*}*

*else return 0;*

*}*

This function displayed the nodes that are related to the path of the height.

*void BST::PrintMaxHeight(){*

*PrintMaxHeightHelper(root);*

*}*

*void BST::PrintMaxHeightHelper(BNode\* subroot){*

*if(subroot){*

*if(MaxHeightHelper(subroot -> left) > MaxHeightHelper(subroot -> right))*

*PrintMaxHeightHelper(subroot -> left);*

*else*

*PrintMaxHeightHelper(subroot -> right);*

*}*

*else*

*return; cout << subroot -> data << "\t" << ' ';*

*}*

*3. Result and Justification*

*How height is related to the number of nodes?*

In figure 1, the number of nodes was 5 and was inserted from smallest to biggest value, therefore the maximum height was 4, where the root’s height is 0. Defined as , where .

The minimum height of figure 1 if inserted randomly was 2 and could be defined as , where

Figure 2 has the same situation with figure 1 except the nodes was inserted from biggest to smallest value therefore, the possible minimum height and possible maximum height was also the same as figure 2.

In figure 3, the number of nodes was 12 and was inserted in a random sequence, therefore the height was 4 as shown above. The possible maximum height of figure 3 if it was inserted from smallest to biggest value or vice versa can be defined as , where

therefore, the possible maximum .

The possible minimum height could be defined as H=-1, where .

therefore, possible minimum

In figure 4, the number of nodes was 15 and was inserted in a random sequence, therefore the height was 3 as shown above. The possible maximum height of figure 4 if it was inserted from smallest to biggest value or vice versa can be defined as , where .

therefore, the possible maximum

The possible minimum height could be defined as where

therefore, the possible minimum

The height of a BST can determine the possible minimum or the possible maximum number of nodes.

If the height of a BST equals 4 then the possible maximum number of nodes when full binary tree was present defined as , where

The possible minimum number of nodes defined as

as shown in figure 1 and 2.

If the height of a BST equals 3 then the possible maximum number of nodes when full binary tree was presents defined as ,

where

as shown in figure 4.

The possible minimum number of nodes defined as

In conclusion of this experiment. I saw multiple possibilities of relationships between the height and the number of nodes. The number of nodes and the sequence it was inserted defined the possible minimum or the possible maximum of the height of the BST. Given that the insertion of the value of the nodes is from biggest to smallest, smallest to biggest, and is random that is either even (full binary tree) or uneven. The height of a BST can determine the possible minimum and possible maximum number of the nodes.