**Threaded Binary Search Tree**

Author: Akhila Preethi Senapathi

Roll No: 214101050

Mtech, CSE, IIT Guwahati

**Threaded Binary Search Tree:**

For fully threaded binary tree, each node has five fields. Three fields like normal binary tree node, another two fields to store boolean value to denote whether link of that side is actual link or thread.

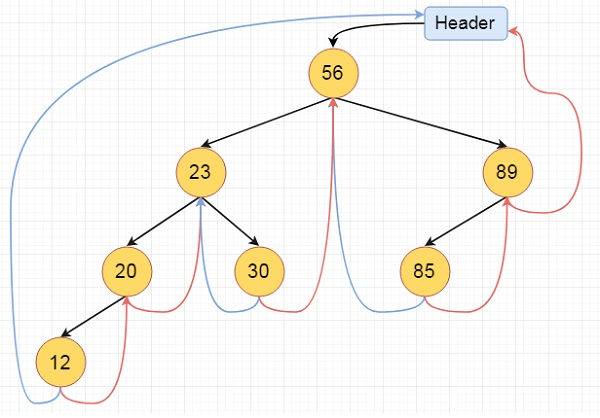
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| lthread | left | data | right | rthread |

* Data – stores the integer node value
* Left and right pointers – to store the next pointers
* lthread and rthread – Boolean value to represent whether left and right pointers point to a thread or a child.
* If lthread is true => left points a child

else if lthread is false => left points to a thread or predecessor

* If rthread is true => right points to a child

else if rthread is false => right points to a thread or successor



|  |  |
| --- | --- |
| **Index** | **Details** |
| **Module** | Insert |
| **Overview** | To add a new element of integer value into the tree as a node. |
| **Business case description** | * **Case-1:** check if the tree is empty * Create a new temporary node and insert into the tree to the left of the dummy node(which acts as a root of the tree). * Reroute all pointers of root to this temporary node. * **Case-2:** check if the duplicate of node is already present in the tree * if duplicate is present, then do not add the new node and remove it. * **Case-3:** check if the if value is larger than the current node data then * insert if current node points to successor * or move to the right if rthread is true(child node) * **Case-4:** check if the if value is smaller than the current node data then * insert if current node points to predecessor * or move to the left if lthread is true(child node) |
| **Output:** | o1.JPG |

|  |  |
| --- | --- |
| **Index** | **Details** |
| **Module** | Search |
| **Overview** | Finds a target node in a tree. |
| **Business case description** | * Uses the helper method contains( ) which acts as an interface method to get the target value. * Once the target value is retrieved, the value is searched in the binary tree from root. * Base condition:   If target data is found, return the pointer object.   * If value is smaller than the current node data then search in the left subtree. * If value is larger than the current node data then search in the right subtree. * Exit Condition: * If both the pointers are pointing to threads, return the null pointer. * when target data is not found, return the null pointer. |
| **Output** | output3.JPGoutput4.JPG |

|  |  |
| --- | --- |
| **Index** | **Details** |
| **Module** | Delete |
| **Overview** | To remove node with provided data value. |
| **Business case description** | * Set parent to dummy node and our pointer to the real root. * Traverse tree until the target node is found and remember parent. * If not found then end method here and value is not found in the tree. * Otherwise, we now have the target node and parent node as well, remove and link the new nodes depending on their children. * **Case-1:** If pointer has two children, then both threads will be true and we would need to reroute the pointers for both children * find the successor nodes for pointer and the parent * Traverse to the leftmost of the pointers successor node. * Now assign pointers data to the successors data * If successor is a leaf, then repeat case-2 * Otherwise, repeat case-3 * **Case-2:** If pointer has only one child * Check whether pointer has right or left child. * Assign parents left or right pointer accordingly. * find successor and predecessor of the pointer node * Assign predecessor right to successor if pointer is not left threaded. * Assign successor left to predecessor if pointer is not right threaded. * **Case-3:** If pointer has no children( leaf node) * Check whether the target is the left or right child of the parent. * If its left child, then make left thread of parent as a thread and assign pointer left to the parent left. * If its right child, then make right thread of parent as a thread and assign pointer right to the parent right. |
| **Output:** | output1.JPG  o3.JPG |

|  |  |
| --- | --- |
| **Index** | **Details** |
| **Module** | Inorder Successor |
| **Overview** | Looks for the next successor of the target node in the inorder. |
| **Business case description** | * **Case-1:** If there is a thread to the right, then return the next threaded node * **Case-2:** If there is no thread then inorder successor is the leftmost node of the node to the right. |

|  |  |
| --- | --- |
| **Index** | **Details** |
| **Module** | Inorder Predecessor |
| **Overview** | Looks for the predecessor to the target node in the inorder. |
| **Business case description** | * **Case-1:** If there is a thread to the left, then return the next threaded node. * **Case-2:** If there is no thread then inorder predecessor is the rightmost node of the node to left. |

|  |  |
| --- | --- |
| **Index** | **Details** |
| **Module** | Inorder |
| **Overview** | Inorder traversal of the tree. |
| **Business case description** | * Check if the tree is empty. * Reach the leftmost node. * Retrieve all the successors from the leftmost node till rightmost node is reached. |
| **Output** | o8.JPG |

|  |  |
| --- | --- |
| **Index** | **Details** |
| **Module** | Reverse Inorder |
| **Overview** | Reverse inorder traversal of the tree. |
| **Business case description** | * Check if the tree is empty. * Reach the rightmost node. * Retrieve all the predecessors from the rightmost node till leftmost node is reached. |
| **Output** | o2.JPG |

|  |  |
| --- | --- |
| **Index** | **Details** |
| **Module** | Range Traversal |
| **Overview** | Retrieve all the elements in the specified range |
| **Business case description** | * Check if the tree is empty. * Traverse the tree for lower bounded value in the given range. * Now as we got the starting point of range, continue to look at the data values and add inorder successors until range exceeds. |
| **Output** | o4.JPG |

|  |  |
| --- | --- |
| **Index** | **Details** |
| **Module** | Kth largest |
| **Overview** | Retrieve the Kth largest element in the tree |
| **Business case description** | * Precondition: Maintain the count of nodes in the right subtree for every node. * Check if the tree is empty. * Start with the root node and if it is not a leaf node then * Case-1: If the count of right subtree incremented by 1 is same as k then return the same as the kth largest element. * Case-2:If the count of the right subtree is less than k then we move to it's left.   -Decrease the k value by the number of elements in the right subtree i.e count + 1 for that node   * Case-3:Otherwise, move to the right of current pointer. * If we reach leaf node then check if the k value is reached. * Otherwise k value is said to be invalid since all the nodes are traversed. |
| **Output** | o5.JPG  o7.JPG |

|  |  |
| --- | --- |
| **Index** | **Details** |
| **Module** | Split |
| **Overview** | To split the BST into 2 BSTs T1 and T2 where keys(T1)<=k and keys(T2)>k |
| **Business case description** | * Consider 2 BSTs with roots initialized. * Start from the root node of BST * Case-1:If the root node value is less than k then , this root node and left subtree of the node will be included in the first tree after splitting as all values in the left subtree will be less than k if root value is less than k. * Case-2: If the root node value is greater than k then , this root node and right subtree of the node will be included in the second tree after splitting as all values in the right subtree will be greater than k if root is greater than k. |

|  |  |
| --- | --- |
| **Index** | **Details** |
| **Module** | Print Tree |
| **Overview** | Print the threaded binary tree using the tool Graphviz. |
| **Business case description** | * Create a file graph.gv which is the input to the Graphviz. * Example dot file(**graph.gv** here)  |  | | --- | | digraph {  5 -> 3;  5 -> 7;  3 -> 1;  3 -> 5[color=red];  nullleft1[shape=point];  1 -> nullleft1;  1 -> 3[color=red];  7 -> 6;  7 -> 9;  6 -> 5[color=blue];  6 -> 7[color=red];  nullright9[shape=point];  9 -> nullright9;   1. -> 7[color=blue];   } |  * Graph can be then generated using the graphviz by the giving the following command in the command prompt:   **Format:** dot.exe –Tsvg <inputfile> -o <outputfile>  Or  dot.exe –Tpng <inputfile> -o <outputfile>   * **Example:** D:\Applications\GraphViz\bin\dot.exe -Tpng E:\graph.gv -o E:\graph.png * Output Grapg(graph.png):   graph.png  Here red pointers used for right threads  Blue pointers used for left threads |