Report for Assignment 6

Deep Kiran Shroti
Roll Number- 12EC35013

March 9, 2016

Interval Tree

Assignment Statement

- Ask user to provide l,u,n and call CREATE_INTERVAL_TREE(l,u,n).
- Ask user to provide integer i, generate i random integers inside [l,u] and insert them to the created disjoint interval list.
- Ask user to provide integer m ¡ n. Write a suitable routine which modifies the existing disjoint interval list to another disjoint interval list which should have m number of mutually exclusive and exhaustive equal sized intervals within the interval range [l,u]. This job should be entirely performed by suitable calls to MERGE.
- Print the data structure using PRETTY_PRINT after each step.

1 Implementation Details

1.1 CREATE_INTERVAL_TREE

The steps followed to create an interval tree is as follows:

- **Step: 1** Input parameters to create interval tree include (l, u, n), where, l is lower limit of interval, u is upper limit of interval, n is the number of interval.
- **Step: 2** Base case is if number of interval is zero or lower limit is more than the upper limit.
- **Step: 3** If number of interval is equal to 1, then the node is assigned the lower limit, upper limit, and lChild, rChild is assigned as NULL.
- **Step: 4** If the number of interval is not equal to 2, then lChild is NULL and rchild are allocated by recursively calling the function.

- **Step: 5** If neither of the above condition holds then the node is assigned the lower and upper value and the rest of the nodes to be created are passed on to the recursion call.
- **Step: 6** The height of the nodes are updated after returning from the recursion call.

1.2 INSERT

- **Step: 1** To insert the key value in the interval tree the bounds of the intervals are checked against the key value to be inserted.
- **Step: 2** If the key value is in the bound then the key is inserted int the list associated with the interval in a sorted fashion using *sortedInsert* function.
- **Step: 3** If the key value is not in the bound the it is passed to the left child or right child appropriately using recursion call.

1.3 MERGE

- **Step: 1** To merge a given interval tree, the given interval is again checked with the interval present in the tree. If the interval lies to the left and right then it is passed appropriately ton the left or right.
- **Step: 2** If the given interval to be merged lies any of the interval boundary of a node. Then the value of interval boundary is changed accordingly.
- **Step:3** Now, depending upon the values the interval to be merged the links for the left child or the right child are created recursively.

1.4 PRETTY_PRINT

- **Step: 1** The inorder traversal scheme is followed. First the left node is visited, then the root and the left node is visited.
- **Step: 2** While visiting the root node, the list associate with the root node is printed using *printList()* function.

2 Conclusion

In the algorithm the *MERGE()*, *GENERATE_INTERVAL_TREE()*and *INSERT()*function divides the interval tree into two trees of approximately of size n/2. So,

$$T(n) = 2T(n/2) + b$$

Therefore the complexity is O(n) for the cases functions.