Assignment : A 3.2 COL106.

ABHINAV SINGHAL 2019CS50768

Before coming to Allocate, Free and defragment, we need to see the time complexity of Find, insert, and delete operations of my BS Tree.

→ Insertion in BSTree : Assuming in the worst case, insertion can be called on any Node, hence we need to get to the root of the tree first. This will take O(h) time, where h is the height of the tree.
Then, we will have to go down the tree and find the correct Node which will be the parent of the inserted node, which also takes O(h) time.

Finally, we change a few pointers to insert the node, which takes constant time.

Hence, Overall worst case time complexity of insertion is O(h). (h = n in worst case, where n is the total number of nodes in the BSTree)

→ Deletion in BS Tree : Again, first we need to get to the root to start searching for the Node to be deleted. It takes O(h) time.

In the worst case, the node found has both left and right child not null, and we need to swap with the successor, and delete the successor.

Finding the node takes O(h) time, and then find its successor also takes O(h) time. Swapping Data with successor node and deleting the successor node (which can have at max. only right child)

ABHINAV SINGHAL 2019CS50768.

takes O(1) i.e, constant time. Hence, worst case time complexity for deletion in BSTree is O(h). (h = n, in worst case).

→ Find in BSTree: For find also, we first get to the root (O(h)) and by simple (key, add) comparisons move down the tree and search according to exact = true or false. Hence, Find in worst case takes O(h) time, approximately (If we found the key, minimizing address happens in approx constant time).

I) Allocate: After n operations, we have O(n) nodes in FreeBlk and AllocBlk.
In a worst case, we might have to find a free Block of size greater than or equal to required size, which takes approximately O(h) time. (where h is O(n) in worst case)

Then, once found, we will split the free Blk and perform insert and delete in the 2 BSTrees, free Blk and Alloc Blk, which takes O(h) time.

Hence, After considering all steps, Allocate takes O(h) worst case (h = O(n) if there is a straight chain of nodes in the BSTree).

II) Free: First we need to find the Node to be deleted in AllocBlk → O(h).

Then, we delete from alloc Blk and insert in FreeBlk, → both take O(h) time.

Hence, Free method takes $O(h)$ time in worst case ($h = O(n)$) if there is a straight chain.

III) → DEFRAGMENT : Here, we create ~~as~~ 2 trees, one as AddressasKey(Tree), and the other as Size as Key (Tree).,

First, we do inorder traversal of freeBlk and add all the nodes in the Address as Key Tree, which takes $O(n*h)$ time because, we are inserting n ~~[illegible]~~ nodes and for each node, we perform getnext and insert, which take $O(h)$ time.

Then, we do inorder traversal of this Address Tree, and see what nodes we can merge. This Step also takes $O(n*h)$ time because of Similar reasoning. After merging, we insert in the Size as Key tree.
Finally we assign FreeBlk to this new tree - Size as Key Tree.

Hence, defragment operation takes $O(nh)$ time in worst case. ($h = O(n)$ in worst case).

Assignment : A3.3 COL106.

ABHINAV SINGHAL 2019CS50768.

For AVL Tree also, I follow the same procedure - First, see time complexity of insert, delete and find. Then, we see Time complexity of Allocate, free and Defragment:

→ Insertion in AVL Tree : Following a similar procedure as in BSTree, Once, we get the parent of the node to be inserted, we insert the node (O(1) : just change a few pointers).

Then, we go up till the root of the tree, and update height simultaenously. If we find an improper node whose children are not balanced, we have to perform rotations (which take O(1)). This process also takes O(h).

In totallity, AVL Tree takes O(h) time. Since for AVL Trees, h = logn , Insertion is O(logn).

→ Deletion in AVL Tree : After performing deletion like in BSTree, we need to go all the way upto the root and correct all the nodes at which there is height imbalance. This process takes O(h) time, because corrections, i.e, rotations take constant time.

Hence, Deletion in AVL takes O(h), i.e O(log n) in worst case.

Assignment : A 3.3 COL106.

ABHINAV SINGHAL 2019CS50768.

→ Find in AVL Tree : Exactly same as BS Tree.
But now we know $h = \log n$.

Hence, Time complexity is $O(\log n)$.

—×—.

I) Allocate : Reasoning is very Similar to Allocate for BSTree. All steps are same. But now $h = \log n$, Hence, Time complexity is $O(\log n)$.

II) Free : Again, we do the same thing as BS Tree.
Time complexity : $O(\log n)$.

III) Defragment : The code doesn't differentiate much between AVL and BS Tree.

It was $O(nh)$ for BS Tree. So, for avl trees it will be $O(n \log n)$ in worst case.