Algorithms and Complexity

Project 2

Hrachya Harutyunyan

1. Description of the Task

In our project we should compare two type of trees BST(Binary trees) and Avl trees.We will see their behavior during insertion and search in the three datasets 1) random 2) nearly ordered and 3)ordered. In the next paragraph we will talk about the trees.

* 1. Binary Search Tree

Binary tree is data structure which consist from Node. Each Node can have maximum two child right and left and for each node we have a rule that left one should be less than his parent node and the right one should be bigger or equal than parent node.

In the Binary search tree insertion and search is O(h) where h is height of the tree. By saying height of tree we mean longest path from root node to leaf node.

The height of Binary search tree in the best case is O(logn) where n is the size of the tree. Worst case for the height is O(n) this can happen when Binary tree is ordered .

* 1. Avl Trees

Avl tree is self balancing binary tree which always has O(logn) for height hence it is eliminating O(n) worst case. So as we considered for the binary tree worst case can happen when the data is ordered .

So the insertion in the Avl will take place in the following steps

1. Insert the node like in BST
2. After addition we should check balance factor for each of the nodes. By the balance factor we mean height of left subtree – height of right subtree.
3. In this step we should find nodes where balance factor is not okey and we need to perform required rotation(bellow every case will be mentioned). If there is no violation of balance so tree should stay in the same structure as before.

Cases for rotation

1. If balancefactor(node)>1 and balancefactor(node->left)=1; we should perform left rotation

2)if balancefactor(node)>1 and balancefactor(node->left) = -1; we should perform left right rotation

3)if balancefactor(node)<-1 and balancefactor(node->right) = 1; we should perform right rotation

4)if balancefactor(node)>1 and balancefactor(node->right) = -1; we should perform right left rotation

2)Methodology

I used C++ language to implement my data structures. Each of data structures in the separate header files with their implementations. Each tree is tested on different sizes with three datasets randomly, ordered and nearly ordered.

To generate ordered tree I am adding elements from 0 to size of the tree. For random tree I have random number generator which is adding random number from 0 to size of the tree in each iteration. For nearly ordered tree I defined function where function first takes ordered tree or list for bst and second argument is number of swaps to be done to get nearly ordered tree. Number of swaps is equal to 20% of size of the tree. And After function returns nearly ordered array which can be used for benchmarking.

So after having all data ready we can start benchmarking for insertion to measure efficiently insertion in ordered Bst. We should find biggest element in the bst tree to do that we should always go right after having this we can do our insertion to the tree with the element biggest element in tree +1 . Same element is also inserted to the other datasets to see their behavior.

Same methodology is being used to measure search(if the element is included) . To have worst case for ordered binary search tree we should find biggest element in the tree. After founding the element we are giving for the search for all datasets to see their behavior.

For the excluded element search I used INT\_MAX because it is guaranteed that I will not have that value in My tree.

3 Results

After running algorithms we are getting this results on the graphs.

Bellow is the graph where we compared Binary search tree insertion in three different datasets.

Chart, line chart

Description automatically generated

Bellow is the graph where we compared Avl tree insertion in three different datasets.

Chart, line chart

Description automatically generated

Bellow is the graph where we have Avl and Binart search tree together to see there difference

Chart, line chart

Description automatically generated

Bellow is the graph of binary search when element is included in Avl and binary search tree

Chart, line chart

Description automatically generated

And in the end we have graph for search when element is not in the list

Chart, line chart

Description automatically generated

In the next chapter of the conclusion we will go through for each of the graphs.

4 Conclusion

In the first graph we can see when we have ordered Bst the insertion takes long time because it should go until the last element and in the end the algorithm should make insertion.

Also we can see that for some values nearly ordered Bst is slower than randomly data. Therefore for the first graph we can conclude that insertion for random< insertion for nearly ordered < insertion for ordered.

In the second graph we can see that in Avl tree ordered is again slower but nearly ordered and random are almost the same.

In the third graph we are investigating the case for both binary and avl . We can see from our graph that insertion for avl is slower for all compared to binary . This has his own reasons because avl has a lot of memory manipulation swaps when it is balancing the nodes.

In the fourth graph we will investigate case for the binary search in both of trees and when element is included in the trees. From the graph we can see that searching in Avl tree is faster this is coming from balancing . For the binary tree we don’t have it so that’s why it is slower than in Avl case .

In the fifth graph we will investigate when element is exlcluded during binary search for both of trees. This one also has same behavior as the last graph and the reason is height of avl . It is making faster than binary search tree case.