List vs Tree.

Sam Khosravi

Spring Term 2022

Introduction

In this report I will compare the run times of a tree and a list, and also explain how these are implemented.

Implementation of List

To begin with, we need to be able to instansiate a new list. This is done by calling on the first function. The second function is if our list we have is empty, and we will just put in the element as it is the only one on the list. Therefore we do not need to think about the order. The last insert will take in the the element (e) we want to add, and then we take in the list in the form of head—tail. We then have an if clause, where if our element we want to insert is smaller than the current head we will add it to the start of the list. Else we will save the head and then call the list function again, but with the tail as the list parameter.

```
def list_new() do [] end

def list_insert(e, []) do
   [e]
end

def list_insert(e, [h|t]) do
   if e < h do [e|[h|t]]
   else [h|list_insert(e, t)]
end
end
```

Implementation of Tree

To start with we will create a empty tree, represented as :nil. The second function is if we want to insert an element to a empty tree, and then we will return a tuple containing a node, element and then the left and right branches, in this case represented as empty and therefore nil. The last insert function is where we will insert nodes if we already have a tree structure. Here if the element we want to insert, e, is smaller than the element already in the tree, ae, we will traverse down the left branch, else we will do the same but down the right one. This is done by in each of these if and else clauses we return a tuple, but the right/left parameter will be a recursive call on the tree with our element.

```
def tree_new() do :nil end

def tree_insert(e, :nil) do
  {:node, e, :nil, :nil}
end

def tree_insert(e, {:node, ae, left, right} ) do
   if e < ae do {:node, ae, tree_insert(e,left), right}
   else {:node, ae, left, tree_insert(e, right)}
end
end
```

Table and comparison

elements	list	tree	ratio
16	1 ms	0.3 ms	3.3
32	0.5 ms	0.2 ms	2.5
64	1.5 ms	1.3 ms	1.2
128	4.9 ms	1.4 ms	3.5
256	16 ms	3.3 ms	4.8
512	30 ms	8 ms	3.8
1024	0.2 s	0.02 s	10
2*1024	1.1 s	0.09 s	12
4*1024	4.3 s	0.2 s	22
8*1024	20 s	0.3 s	53

Table 1: List vs Tree, different tries of different number elements and their runtimes

In this table we see that in all cases we traverse a tree faster than a list. This difference however starts making sense when applied to larger numbers of elements. We see that the larger the input of elements, the more of a difference we will see. This is because a binary tree has the big O complexity of log(n), while the list is traveresed with the complexity of N.

Conclusion

The implementation of the insert in a list and tree were pretty straight forward and easy. I followed the structure for trees that were given during the lecture. The difference between list and tree was also expected, as I know from before that a binary search tree has a good time complexity compared to one that is linear, which in our case is our list.