Tree

Fredrik Berzins

Fall 2023

Introduction

A tree structure is a node structure that's slightly more intricate version of a linked list where a node only has one pointer to the next elements. A tree structure instead has multiple pointers from each node, none of witch go back to the node before it.

Tree structures can be described further with requirements of balance or binary trees. A balanced tree has an equal amount of nodes on each branch, A binary tree will always have two pointers (can be null pointers). Binary trees are the main subject of this report.

Linked structure explanation

A linked structure also called a node structure is a structure made from pointers/references and other values(to store information).

Tree explanation

A tree is a node structure that has one known origin/root node that has some data and some references general it has at least 2 references and one data type. Compared to linked lists it has multiple pointers from each node, and the difference compared to double linked list is that it doesn't have pointers towards root it's pointers are always away from the root node.

Iterator explanation

It iterates thru the tree and pushes the nodes to a stack, by going as far left, if it's reached the end then backtrack until it can go right. when it managed to go right it tries to go as far left again and so on.

Implementation

Node implementation

It has a key,value structure for this assignment but that can be changed to a preferred data structure, it can also have more pointers for a more compact tree.

```
public class TreeNode {
    public Integer key;
    public Integer value;
    public TreeNode left, right;
    public TreeNode(Integer key, Integer value) {
        this.key = key;
        this.value = value;
        this.left = this.right = null;
    }
```

Tree implementation

It has a very simple structure and the add is equally simple as it checks if its suppose to update a value or if its suppose to be in the right or left of a current node. It has a recursive element for the add to find wear to put the value.

```
public class BinaryTree {
    TreeNode root;
    public BinaryTree() {
        root = null;
    }
    public void add(Integer key, Integer value) {
        root = add(root, key, value);
    }
    private TreeNode add(TreeNode n, Integer key, Integer value) {
        if (n == null) {
            return new TreeNode(key, value);
        } else if (n.key == key) {
            n.value = value;
        } else if (n.key > key) {
            n.left = add(n.left, key, value);
        } else if (n.key < key) {
            n.right = add(n.right, key, value);
        }
        return n;
    }
```

Iterator implementation

The iterator is implemented as a stack where all higher value towards the root are on the stack and it pushes when moving down and then pops when backtracking up and so on so the values in the stack are always in order.

```
public TreeIterator(TreeNode n) {
    s = new Stack<TreeNode>();
    while (n != null) {
        s.push(n);
        n = n.left;
    }
}
```

Benchmarks

The testing was done with length between 100 and 25600 values(doubling each time, 100,200,400...), the amount of random keys for all test was 400. Each time test was redone 1'000 times witch was doing 1000 different lookup nodes to get as accurate results as possible. This gives a min time showing avg from the group with the 1000 quickest runs.

Result

The result from the benchmark is that a tree has a time dependant on the amount of nodes in the tree(It's depth, this is a measurement of how many pointers that can be traversed at most.) The big o notation to find random keys in the tree is $log(n)$ this is since it's doing a form of binary search.

n	**Linked list**
100	11.9
200	12.9
400	14.0
800	15.2
1600	16.4
3200	17.4
6400	18.6
12800	19.6
25600	20.8

Table 1: Shows time to find a random node in the tree, in ms.

Conclusion

In conclusion a tree is a useful structure that is slightly more complicated then linked list, but it also gains some major benefits like how fast it can be searched thru and have a built in sorting as long as it don't need it to be balanced. A irritable tree also makes it be more on par with a linked list that is easily irritable.

The iterator is useful when searching thru the tree but it has it's limitation when adding nodes to the tree structure while having a partially filled stack. It wont iterate thru the new nodes with some exceptions to nodes added to a branch that not yet been pushed to the stack.