A Trees

Nariman Haidar

Fall term 2022

1 Introduction:

In this assignment we are going to write a binary tree with add and look up methods and then we are going to compare the execution time between binary search and look up. Then we are going to understand who the iterator works and we are going to implement the iterator method.

2 Method:

A binary tree

A binary tree is a tree that is recursive data structure that each data element is stored in a structure called a node. In a binary tree each node can have 2 children at most. In a binary tree store data in hierarchical form so it is a non linear data structure.

I a binary tree we started with the first node that called root and then root has two children at most one to the left and one to the right. Each node has also two children at most until the tree is ended then each of the nodes that the tree ended with has a pointer to null. In a binary tree node has data and pointer to the left and pointer to the right. If the root is a null pointer the tree is empty.

In our assignment we are going to implement the binary tree using linked list data structure. In our case the tree is sorted so all nodes with keys smaller than the root key are found in the left branch and the nodes with larger keys in the right branch.The ordering is of course recursive so if we go down the left branch we will find smaller keys to the left etc.

To implement the binary tree I started with write a class binary tree and then write an nested class Node to describe the internal structure of the tree. The tree it self only holds one property, the root of the tree. Root represents the root node of the tree and initialize it to null. In a class node is created, each node has a key and value will be set to integer and both left and right will be set to node type. Then assign key to the key and value to the value, set left and right children to null.

Then I write add method and the add method to add all nodes that we wants to add. It takes two parameter key k and value v. If the key is the same k so the value is v. if the new key is lower than the current k , we go to the left child and make a recursive add until the left side pointer to the null. If the new key is greater than the current key, we go to the right child and make a recursive add until the right side pointer to the null.

In the binary tree there is another way to search through tree, It uses the key to find the values in the tree, then the method returns the value in that particular key. To implement this method I started with method look up that takes an integer key like parameter and I set the current is root and then while the current is not null so the current key is the key and if the current key is smaller than the key the current key is going to right else it is going left. Then I set up a benchmark to compare the execution time for a growing data set between lookup algorithm compares to the binary search algorithm.

Compare execution time between Binary Search and look up:

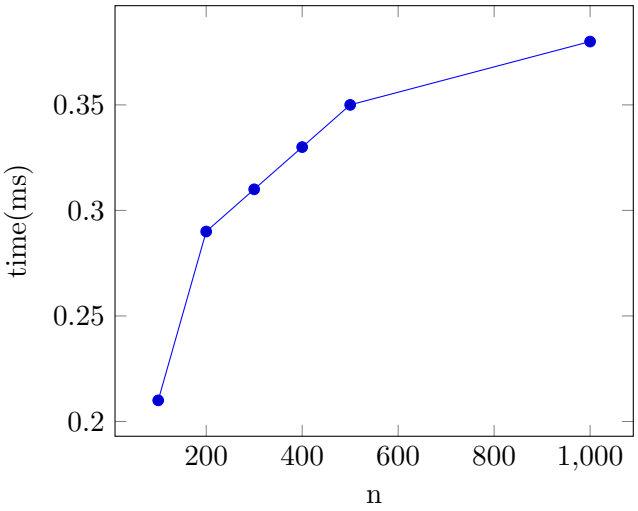
To compare the time complexity between the binary search and the look up. First to measure the time in look up method when searching for a specific key in the tree. I make some benchmark then I make random numbers as keys as well as the value i tree and then give the look up method random values as keys. We can see in this table and graph all results as it is logarithmic not as linear because of we have a balanced tree so the complexity time to the look up is O(log(n)).And In the worst case, when the tree is not balanced, which means that when we are going to search for an element in the deeper half so we need to go through n elements and then the complexity time is O(n). In the binary search the complexity time is o(nlog(n)). In the previous task we knew that when we measured the time and wrote the table and draw the graph.

table

n	**Look up**
100	0.21
200	0.29
300	0.31
400	0.33
500	0.35
1000	0.38

Table 1: This is the time out put look up method

Graph:look up



Depth first traversal:

In this part I used the assignment print out to go through all items that we have in a tree and because of our binary tree is ordered with smaller keys to the left,so the order would be to traverse the item starting with the leftmost and then work way towards the rightmost. So when we implement the print method we make use of the implicit stack of the java language. In the print method going to recursively have printed the items of the left branch we return to the node we are currently in, print the key and value and then recursively print the items of the right branch. Can we do this using an explicit stack? Yes, we can we can make a method that take a parameter st from type stack and then in the method we are going to check if the left is not null then we are going deeply and push and here it calls itself until it is null. Then we are going to check it the right is not null so it going deeply and push and here it calls itself until it is null.

An iterator:

An Iterator is an object that can be used to loop through a data structure before returning its contents.Iterator object can be created by calling iterator() method present in Collection interface. An iterator has two methods hasNext returns true if the iteration has more elements. And next returns the next element in the iteration. It throws NoSuchElementException if no more element is present. And remove removes the next element in the iteration. The iterator works first we create an Iterator object on List object. And here iterator's cursor is pointing before the first element of the List. But when we run the methods hasNext and next then the iterator's cursor

points to the first element in the list and it doing this until it pointers to the end element of the List then after reading the final element, it returns a “false” value. In this assignment the iterator will go through nodes and return the value as they persist, and we are going to use the stack that we are going traversal in order that we are going to go first to the left and then pushed them into the stack until we get the last element and then all values will be get out one after the other one until we get back to the root. And then we are going to the right side and do the same thing. To implement the iterator we pushed all values from the tree to the stack. The hasnext method calls the method isEmpty from the stack to check if the stack contains values so it going to pop them and it returns true or false. And then the next method returns all values in a specific order. If we create an iterator, retrieve a few elements and then add new elements to the tree. We just have to go to the beginning and do all the work again because when we pushed all the elements to the tree and when are they popped and then we want to add new value to this tree so it will not work and we will lose values because we have to start over.

code:In this code we can see hasNext and next methods:

```
@Override
public boolean hasNext() {
    return !stack.isEmpty();
}

@Override
public Integer next() {
    if (!hasNext()) throw new NoSuchElementException();
    Integer key = stack.pop();
    return key;
}
```