SE 2C03 Assignment 3

Shazil Arif, 400201970, arifs2

Feb 24th, 2020

1 Answers

3.2.15

- floor("Q") is Q itself. First Node E is examined. Q is greater than E. Then the right sub tree of Node E is examined. So Node Q is then examined. Here the keys are equal so we stop. The order is:
  E $->$ Q $->$ return
- select(5) E $->$ Q $->$ return
- ceiling("Q") E $->$ Q $->$ return
- rank("J") E $->$ Q $->$ J $->$ return
- size("D","T") E $->$ D $->$ Q $->$ J $->$ M $->$ T $->$ S $->$ return
- keys("D", "T") E $->$ D $->$ Q $->$ J $->$ M $->$ T $->$ S $->$ return

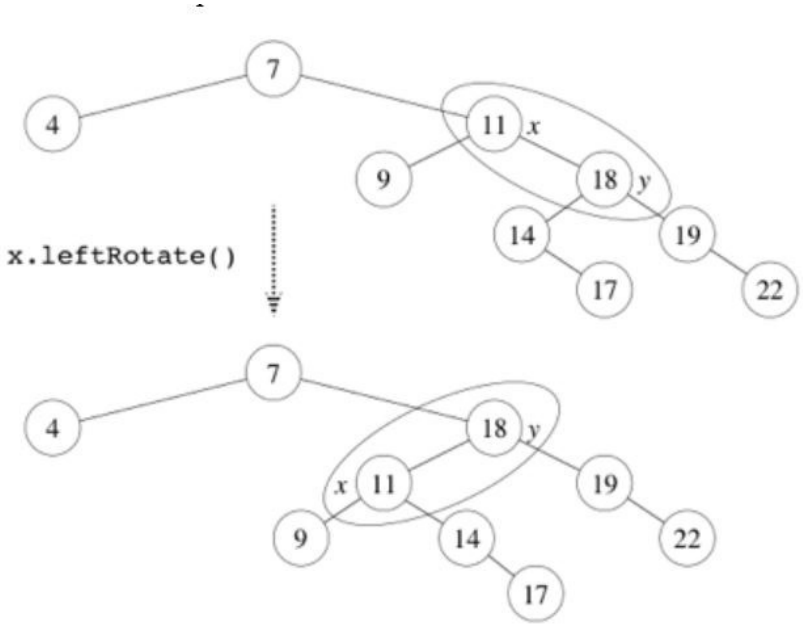
3.2.24 When building a BST, each insertion takes $\sim lgN$ compares. If we insert N keys and each takes $\sim lgN$ compares then we need at least $\sim NlgN$ comparison.

3.2.26 Yes it can be done using a stack. We do the following:

1. Add root of tree and its left children onto a stack
2. while the stack is not empty, pop the top element and then set a current element variable to the right child of the popped element
3. Add right child and its left children onto stack

4. Repeat above two steps until stack is empty

3.3.38 Any BST can be transformed into another BST by a sequence of left and right rotations because the rotation operation preserves the key property of a BST that every left child is less than its parent and every right child is greater than its parent. Consider this example which was borrowed from:
(https://www.cs.dartmouth.edu/ thc/cs10/lectures/0519/0519.html):



Doing a left rotation on node x makes it a left child of its right child (node y). Since node x right child (node y) is greater than it all of node y's children are greater than node x. So when we do the rotation the BST property is still satisfied

3.4.15 In the worst case with linear probing, all keys hash to the same index. Then the number of compares will be 1 if their is only key. 2 if there are 2 keys, 3, 4 and so on until N/2 when the table is re sized to a size of $2N$. There are already N/2 keys which need to be re hashed and re inserted which will again take 1 compares, 2 compares, 3, 4 .... N/2. Now when new keys are inserted, they hash to the same value so there are already N/2 keys with the same hash value so it takes N/2 + 1 compares then N/2 + 2, N/2 + 3 .... until N when the table would be re sized again. Doing an analysis of the comparison

The followings calculations use the formula: $\frac{n}{2} * (\frac{n}{2} + 1)$

Initial insertion: $1 + 2 + 3 .... + \frac{N}{2} = \frac{N}{\frac{2}{2}} * (\frac{N}{2} + 1) = \frac{N^2}{8} + \frac{N}{4}$.

Re sizing: $1 + 2 + 3 .... + \frac{N}{2} = \frac{N^2}{8} + \frac{N}{4}$. (Same as above)

Inserting new keys after re sizing: $(\frac{N}{2} + 1) + (\frac{N}{2} + 2) .... (\frac{N}{2} + \frac{N}{2})$
$= \frac{N^2}{4} + \frac{N^2}{8} + \frac{N}{4}$
Totaling up all of these: $2 * (\frac{N^2}{8} + \frac{N}{4}) + \frac{N^2}{4} + \frac{N^2}{8} + \frac{N}{4}$
$(\frac{N^2}{4} + \frac{N}{2}) + \frac{N^2}{4} + \frac{N^2}{8} + \frac{N}{4}$
$\frac{N^2}{2} + \frac{N^2}{8} + \frac{2*N}{4} + \frac{N}{4}$
$\frac{5*N^2}{8} + \frac{3*N}{4}$

And we can see that the order of growth is O(N^2)

3.5.25 The school should use an array/list for each instructor. This list/array would behave like a set and allow no duplicates. To add a new class to teach for an instructor the class time can be added to their list, if an existing time is attempted to be added, it will throw an error due to a conflict. Some pseudo code would look like

```
add_class(time, instructor_list){

    valid_times = [9,10,11,1,2,3]
    if instructor_list.contains(time) and valid_times.contains(time){
            throw Exception
    }
    else{
        instructor_list.add(time)
    }

}
main(){
    instructor1 = new Set()
    instructor2 = new Set()
    add_class(9,instructor1)
    add_class(10,instructor2)
    .
    .
```

.

}