School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

COMS4032A
Applications of Algorithms
Assignment 3

Moses Katlego Modupi (1614669)

October 8, 2020

1 Part A

1.1 Test Input generation

The graph for building a tree on n elements was obtained by running inserting randomly shuffled of numbers in the range $[0, n-1]$. We ran 20 reruns for each n and we increased the size of n by 20000 starting from 20000 till we reached 1 million. The graphs for the insert and delete where obtained by inserting/deleting a node from trees that were built in the tree building simulation.

1.2 Graphs

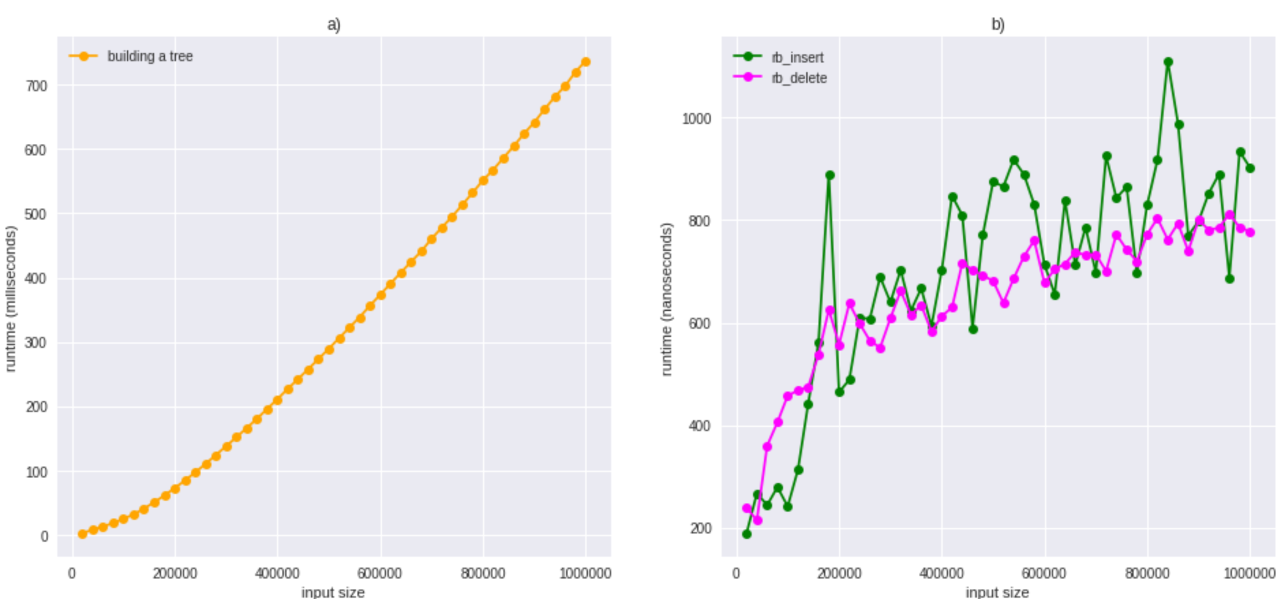

Figure 1

2 Part B

To solve the given problem, we add the attributes minimum and maximum to the tree nodes. The minimum attribute keeps track of the maximum node in the sub tree rooted at node x. The same applies for the maximum. We update the insert function by checking if the min(max), for each node we traverse, is larger(smaller) than the node to be inserted. For the delete, we update the min(max) for nodes up the tree whenever we remove a node that has a left(right) nil node. Querying the minimum of a sub tree rooted at node x just requires a reference to x.minimum. Querying the successor of sub tree rooted at node x is given by x.maximum.parent. Algorithms 1 and 2 represent the revised insert and delete operations.

Algorithm 1: rb_insert_aug(T, z)

```
z.minimum = z
z.maximum = z
y = T.nil
x = T.root
while x ≠ T.nil do
    y = x
    if z.key < x.minimum then
        x.minimum = z
    end if
    if z.key > x.maximum then
        x.maximum = z
    end if
    if z.key < x.key then
        x = x.left
    else
        x = x.right
    end if
end while
z.p = y
if y==T.nil then
    T.nil = z
else if z.key<y.key then
    y.left = z
else
    y.right = z
end if
z.left = T.nil
z.right = T.nil
z.color = RED
rb_insert_fixup(T, z)
```

Algorithm 2: rb_delete_aug(T, z)

```
if z.parent != T.nil and z.minimum == z then
    rb_fix_minimum(T , z->parent, z->parent, z)
end if
if z.parent != T.nil and z.maximum == z then
    rb_fix_maximum(T , z->parent, z->parent, z)
end if
y = z
y-original-color = y.color
if z.left == T.nil then
    x = z.right
    rb_transplant(T, z, z->right)
else if z.right == T.nil then
    x = z.left
    rb_transplant(T, z, z->left)
else
    y = tree_minimum(z.right)
    y-original-color = y.color
    x = y.right
    if y.p == z then
        x.p = y
    else
        rb_transplant(T, y, y.right)
        y.right = z.right
        y.right.p = y
    end if
    y.left = z.left
    y.left.p = y
    y.color = z.color
end if
if y-original-color == BLACK then
    rb_delete_fixup(T, x)
end if
```

Algorithm 3: rb_fix_minimum(T, x, new_min, old_min)

```
while x != T.nil and x.minimum == old_min do
    x.minimum = new_min;
    x = x.parent;
end while
```

Algorithm 4: rb_fix_maximum(T, x, new_max, old_max)

```
while x != T.nil and x.maximum == old_max do
    x.maximum = new_max;
    x = x.parent;
end while
```

Algorithm 5: rb_get_minimum(T, z)

```
return z.minimum;
```

Algorithm 6: rb_get_successor(T, z)

```
if z.key == z.maximum.key then
    return T.nil //cant have a successor
end if
return z.maximum.parent;
```