ESO207 Assignment 2.1

Bhavya Garg(200270) Harshit Bansal(200428)

October 25, 2021

1 Question 1 – Merge 2-3 Trees

1.1 Pseudo Code

function findHeight(node)
    height = 0

    while node ≠ NULL **do**
        height++
        node = node.left
    end while

    return height
end function

function findMin(node)
    while node.type ≠ NULL **do**
        node = node.left
    end while
    return node.leafData
end function

function insertTreeLeft(N1, N, m)

    if $N1$.parent == NULL **then**

        if N == NULL **then**
            return $N1$
        end if

        if $N1$ == twoNode(x, A, B) **then**
            $N1$ = threeNode(m, x, N, A, B)
            N.parent = $N1$
            return $N1$
        else if $N1$ == threeNode(x, y, A, B, C) **then**
            t = x
            $N2$ = twoNode(m, N, A)
            $N1$ = twoNode(y, B, C)
            return twoNode(t, $N2$, $N1$)

```
        end if
    end if

    if N==NULL then
        return insertTreeLeft(N1.parent, NULL, NULL)
    end if

    if N1==twoNode(x, A, B) then
        N1 = threeNode(m, x, N, A, B)
        N.parent = N1
        return insertTreeLeft(N1.parent, NULL, NULL)

    else if  N1==ThreeNode(x, y, A, B, C)  then
        t = x;
        N2 = twoNode(m, N, A)
        N1 = twoNode(x, B, C)
        return insertTreeLeft(N1.parent, N2, t)
    end if
end function



function insertTreeRight(N1, N, v)
    if N1.parent == NULL then
        if N== NULL then
            return N1
        end if
        if  N1== twoNode(x,A,B then
            N1= threeNode(x, v, A, B, N)
            N.parent = N1
            return N1
        else if N1 == threeNode(x, y, A, B, C) then
            t = y
            N2 = twoNode(x, A, B)
            return twoNode(t, N1, N2)
        end if
    end if
    if N== NULL then
        return insertTreeRight(N1.parent, NULL, NULL);
    end if
    if N1== twoNode(x, A, B) then
        N1= threeNode(x, z, A, B, N)
        N.parent = N1
        return insertTreeRight(N1.parent, NULL, NULL)

    else if N1 == threeNode(x, y, A, B, C) then
        t = y;
        N2 = twoNode(z, C, N)
        N1 = twoNode(x, A, B)
        return insertTreeRight(N1.parent, N2, z)
    end if
end function
```

```
function Merge(T1, T2)
    if T1.root == NULL then
        return T2
    end if
    if T2.root == NULL then
        return T1
    end if

    h1 = findHeight(T1.root)
    h2 = findHeight(T2.root)
    min2 = findMin(T2.root)

    if h1 == h2 then
        return twoNode(min2, T1.root, T2.root)
    end if

    r1 = T1.root
    r2 = = T2.root
    if h1 > h2 then
        while h1 ≠ h2+1 do
            if r1.right then
                r1 = r1.right
            else
                r1 = r1.center
                h1−−
            end if
        end while

        return insertTreeRight(r1, r2, min2)
    end if

    if h2 > h1 then
        while h2 ≠ h1+1 do
            r2 = r2.left
            h2−−
        end while

        return insertTreeLeft(r2, r1, min2)
    end if
end function
```

1.2 Time Complexity Analysis

- Height and Minimum value of tree are calculated by traversing along the left side of tree till we reach leaf. Tree is traversed using while loop, going down the tree one step on each iteration. Each iteration executes in constant time lets say c1.
  Let height of Tree is $h(T)$.
  Then functions findMin and findHeight work execute in $c1 * h(T) + c2$
  Hence time complexity of both these functions is $\mathcal{O}(h(T))$.
- In Function Merge we find heights of both trees and minimum value of tree with larger values (T_2 here). All this can be done in $\mathcal{O}(h(T_1) + h(T_2))$ time.
  Now there are three cases - $h(T_1) = h(T_2)$, $h(T_1) > h(T_2)$ and $h(T_2) > h(T_1)$.

- Case 1 : If $h(T_1) = h(T_2)$, then we make both trees child another root node and return this node. This is a constant time operation.
- Case 2 and Case 3 : Both cases are similar in view of Time Complexity.
  Lets take $h(T_1) > h(T_2)$. We traverse down the T1 using while loop, till we reach the height $h(T_2) + 1$. This takes $\mathcal{O}(h(T_1) - h(T_2))$ time.
  Now the insertTreeLeft or insertTreeRight is called accordingly. Again time complexity of both these functions is same. Here for our case insertTreeRight is called.
  insertRightTree recursively traverses the tree T1 back to root which constant time operations on each step. On each recursive call we go up the tree one step till we reach root node. Using recursion:
  $T(h) = T(h-1) + c$
  Hence insertTreeRight works in $\mathcal{O}(h(T_1) - h(T_2))$ time.

So Merge functions works in $\mathcal{O}(h(T_1) + h(T_2)) + \mathcal{O}(|h(T_1) - h(T_2)|)$ time which is $\mathcal{O}(h(T_1) + h(T_2))$.