685.621 Algorithms for Data Science

Homework 6

Sandesh Iyer

1

Algorithm 1 Postorder Traversal

---

function POSTORDER($root$)
  if $root \neq$ null **then**
    POSTORDER($root.left$)
    POSTORDER($root.right$)
    visit $root$
  end if
end function

Time complexity

Traversal time: T(x) = Θ(x) for 'y' tree that has 'x' vertices since it visits and prints vertex once. This entails the constant costs being associated with the moving along the vertices and displaying them.

T(x) = Ω(x) - traversals visit all the vertices 'x'

T(x) = O(x) – displayed

Base case of recurrence is for x≠ NULL

T(0) = z where z is some constant greater than 0

For recurrence relation where x > 0. If we traverse on 'y' vertex 'x' with 'k' vertices in left subtree and 'n-1-k' vertices in right subtree and it takes constant time p > 0 to execute body of traversal exclusive of recursive calls.

$T(x) \leq T(k) + T(x-k-1) + p$

Inductive hypotheses: Suppose $T(m) \leq (z + p)m + z$ for all $m < x$

Base case: $T(0) = (z + p) * 0 + z = z$

Inductive proof steps

$T(x) \leq T(k) + T(x-k-1) + p$ by definition

- $((z + p) k + z) + ((z + p) (x-k-1) + z) + p$
- $((z + p)(k + x - k - 1) + z + z + p$
- $((z + p)(x - 1) + z + z + p$
- $((z + p)x + z - (z + p) + z + p$
- $((z + p)x + z$
- Thus, we can see that the algorithm runs in time Θ(n)

2

Pseudo code

```
 search(node){
    if (node == null) { #reached depth of tree ano no such key found
      return false #returns false since no structure is found
    }
    if ((node.key >= a) and (node.key <= b)) { #Checks if key in between a and b and inclusive (a<=x<=b)
      return true #yes key is present, will return true
    }
    else { #node.key is not required hence we have to move left/right
      if (b < node.key) { #b is less than current node key and we have to move left, ...
      #since in avl tree left subtree values are smaller tha current node key value
        return search(node.left) #calls left node recursively
      }
      else if (a > node.key) { #a is greater than current node key...move right
        return search(node.right) # calls right node recursively, since in avl tree right subtree values are > than current node key value
      }
    }
  }
```

Time complexity: O(log n)

We are leaving half the nodes (subtree) meaning at every level, the number of nodes needed to be checked is reduced by half. In avl tree from top to depth up-to correct key location we check (log n) node keys only. Hence time complexity is O(log n). ‘n’ is the number of keys in the avl tree.