מסמך תיעוד תרגיל מעשי 1 - מבנת

עץ אדום שחור

Classes:

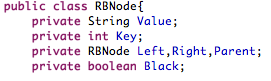
An implementation of a Red Black Tree with non-negative, distinct integer keys and values.

Root is of type RBNode and is the infinity node(sentinel - will be explained later) of the given RBTree.

Constructors:

Construct a new empty RBTree (Only infinity node with a left child (null node) and no right child).

Construct a new RBTree with a root. (the root will be the left child of the infinity node).

The RBTree described is constructed of RBNode objects.

Value - the value that the node carries

Key - the key of the node. Only node in the tree need to have a different key and our tree supports only non negative keys values.

Left- the left child node. Can be RBNode with value or a NULL node (will be explained later)

right - the right child node. Can be RBNode with value or a NULL node.

Parent - the parent of this node. has to be a non NULL node with a value or the sentinel root (the infinity node)

Black - if true the node is black O.W the node is red.

Constructors:

Construct a new RBNode with a value , key and a parent. (the children will be 2 null nodes)

Construct a new RBNode with value, key, parent and 2 children. can set the children to be null nodes.

Note: All the constructors are of complexity of O(1) (const number of operations)

Methods:

RBTree:

Return the Root of the tree. Our tree has a sentinel “root” we call it the “infinity node” when asking for the root this method return the first node which is not the infinity node. (Tree.Root.Left)

O(1)

create infinity node with Null children and return it.

O(1)

create infinity node with a left child (infinity node cant have a right child) and return it.

O(1)

create a null node and connect it to a parent. return this node.

O(1)

Return true if the tree is empty and false O.W.

O(1)

Returns the value of an item with key k if it exists in the tree otherwise, returns null.

O(log(n)) - this method perform const number of operation + calls the method “SearchNode” once.

this method “cost” O(log(n)) time in the worst case (will be explained next)

Look for a node with the key “k” if found in the tree return in and if not returns the node which would have been its parent if it was exist in the tree.

“node” is the root of the sub tree where this method is going to search.

this is a recursive method which moving “down” in the tree height in every iteration. Because we use a Red Black Tree. we know that the maximum depth of the tree is Log(n) and this is why in the worst case this method will perform O(1) operations Log(n) times.

O(log(n))

Insert a new node to the tree.

this method will work in 3 steps:

1. Look for the right location to insert the node and insert a “red” node with the value and the key in this location. (if a node with the same value is already exist return “-1”)
2. Call another method called “fixUpTree”. this method will check if the red rule or the black rule was compromised and fix it. “fixUpTree” runs in O(log(n)) and return the number of colour changes performed during the fix.
3. return the amount of colour changes.

O(log(n))

This method is called after inserting a new node to the tree. This method will check if the tree is valid (following the RB rules) and if not fix them.

This method will make Log(n) operation in the worst case.

We discussed in class about the way this method works. when giving a “z” node this method will fix the local problem (with a const number of operations) and move the problem up to the next upper level in the tree. If this level is not valid it will fix it and keep on going.

The maximum number of iterations will be log(n).

This method will also count the amount of colour changes that been made to the nodes of this tree and return it. We suppurate this method to 4 cases and each case changes the number of colours a known number of times. We count this amount during the iterations and finally return it.

This 3 cases were also discussed in class.

this method calls 2 other methods calls (leftRotate and rightRotate). this methods run in a const time and that is why we care only about the number of iterations when it comes to calculate the running complexity of this method

O(log(n))



Put y as a left child of x.

O(1)



Put y as a right child of x.

O(1)



put y instead of x. Uses rightChild and LeftChild (both run on O(1))

O(1)



Left Rotate around x as discussed and described in class.

The rotation includes only const number of operations.

O(1)

Right Rotate around x as discussed and described in class.

The rotation includes only const number of operations.

O(1)



Find and delete the node with the key k in the tree.

If it doesn't exist return -1

Return the number of colour switches that performed in order to keep the tree as a valid

red black tree.

This method uses a constant number of operations to delete the node and reassemble the tree and than call “deleteFixup” to fix the tree and keep it valid.

deleteFixup as will be described next run in a O(log(n)) time.

O(log(n))



X is the location where we have a problem (probably with the black rule). In order to fix the problem we will have to deal with 1 of 4 delete cases discussed in class and in the course book.

Each case will take a constant time and will move the problem up.

Only case 2 can move the problem down but this can happen only once and this is why we can assume in general that the problem always goes up and this is why the maximum amount of fixes we will have to perform is log(n) (if the problem remains until we reach the root).

This is why this method cost O(log(n)) time.

(This method iterates with a while loop until x is the root or x is red. Every iteration x changes)



If the tree is empty returns null, else return the value of the node with the minimum key in the tree.

This method first check if the tree is empty and this cost O(1) operations. After that it calls another method named: minimumNode which runs in a O(log(n)) time (will be explained next).

O(log(n))



This method run recursively on the height of the sub tree with the root node. Every call “node” is in a lower level of the tree. because we want to find the minimum node, when using a balanced tree this method will perform O(log(n)) operations after calling itself log(n) times.



Return true if the given node is the tree InfinityNode

O(1)



Return true if the given node is the tree null Node

O(1)



If the tree is empty returns null, else return the value of the node with the maximum key in the tree.

This method first check if the tree is empty and this cost O(1) operations. After that it calls another method named: maxValue which runs in a O(log(n)) time (will be explained next).

O(log(n))



This method run recursively on the height of the sub tree with the root node. Every call “node” is in a lower level of the tree. because we want to find the maximum node, when using a balanced tree this method will perform O(log(n)) operations after calling itself log(n) times.



This method returns an array of the keys in the tree.

This method calls anther method named ElementsToString, this method returns a string with

the keys separated with “,”.

After getting the string result the method will replace the string with an array of all the keys and return it.



ElementsToString run recursively and in O(n) run complexity.

O(n)

Runs recursively and go over all the elements in the tree.

Returns a string of the “elements” separated with “,”. If key is true the string contains the keys of the elements and if key is false the return string will contains the values of the elements.

this method go over all of the elements in the tree and cost O(n) operations

This method gets an array of strings that can be parsed into int.

Returns an array of ints with the parsed strings from the given array.

this method runs in a linear order over the given array O(strArr length)

in our case this method will get an array with all of the tree elements O(n)

This method returns an array of the values in the tree.

This method calls anther method named ElementsToString, this method returns a string with

the values separated with “,”.

After getting the string result the method will replace the string with an array of all the keys and return it.

ElementsToString run recursively and in O(n) run complexity.

O(n)

This method return the number of nodes in the tree.

This method return 0 if the tree is empty and otherwise calls sizeCalc with the tree root and return the size value.

sizeCalc run in a O(n) time.

O(n)

This method runs recursively on the tree depth and go over all of the nodes.

It sum the number of the nodes and return it.

O(n)