# Is It A Red-Black Tree

## Project1(normal)

朱理真

**Teacher: Feng Yan**

**Date: 2020-10-25**

# Chapter 1: Introduction

The red-black tree is a balanced search tree with 5 properties:

- a) Every node is either red or black.
- b) The root is black.
- c) Every leaf (NULL) is black.
- d) If a node is red, then both its children are black.
- e) For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

Our algorithm is used to decide whether a tree is a red-black tree with the preorder traversal sequence of the tree as input. Since it is not a tree-like structure, extra operations may used to find the left and right children of the root of each subtree recursively.

# Chapter 2: Algorithm Specification

For convenience, we denote **the number of black nodes in the path from node N to one of its leaf** as **“Black Height”**

## Data structure:

We only need an array to store the indices of the nodes sorted by pre-order:

```
    int nodes[MAXNODES];
```

## Procedure:

### The main structure:

```
main(){
    Read how many test cases are there,
    and store them in variable “batches”.

    nodes = A new array with size MAXNODES(=30);
    for each batch,
        TestCase(node);
}
```

### In Each Test Case:

```
TestCase(nodes[]){
    Call Input() to get the information of size and nodes.

    if the root is red (property 1)
        flag = BADVALUE;
    else
       Call Judge to decide whether the tree meets another 4 properties
       of RBT.
       If it is not satisfied, flag = BADVALUE.

    if flag!=BADVALUE
        Print("Yes");
    else
        Print("No");
}
```

## To read Input:

```
InPut(nodes[]){
    read size.

    i=0;
    while(i<size){
        read nodes[i]
        i++;
    }
    return size;
}
```

## Judge whether a tree is a subtree of RBT:

```
Return Black Height.
Judge(nodes[],RootIndex, size, UpBound){
    if nodes[RootIndex] is the last one in the nodes{
        Tell its parent it has no right bound.

        if nodes[RootIndex] is black
            Tell its parent its black height is 2
        else
            Tell its parent its black height is 1
       return
    }
    if the node behind the node[RootIndex] in preorder sequence is 
larger than it{
        let the black height of left child = 1 (it is a leaf(nil))

        let the index of left child = NULLPOS

        let the index of right child = RootIndex + 1
    }/*Look if the left child is next to the root or be NULL(leaf).*/
    else{
        let the index of left child = RootIndex + 1

        if node[RootIndex] is red and  left child is red
            return and tell parent the tree is not a Red Black Tree.

        Call the function Judge() on its left child, and get the black 
        height of left child, and let the index of right child be the 
        right bound of left child.

        But if left child tells the tree is not a Red Black Tree,
        return and tell the parent of current node that the tree is not 
        a Red Black Tree.
    }

    if the index of right child is NULLPOS or right child is larger 
than UpBound){

        let the black height of left child = 1 (it is a leaf(nil))

        if the index of right child is not NULLPOS
            Tell its parent its index of right bound is RootIndex+1

    }//right child is found a leaf(null)
    else{


        if node[RootIndex] is red and  right child is red
            return and tell parent it is not a Red Black Tree.

        Call the function Judge() on its right child, and get the black 
        height of right child, and tell the parent its right bound is 
        the right bound of right child.
        But if right child tells the tree is not a Red Black Tree,
        return and tell the parent of current node that the tree is not
        a Red Black Tree
    }

    if the black height of left child != right height of right child
        tell the parent the tree is not a Red Black Tree

    if(node[RootIndex] = black)
        return the black height of left child+1 as its black height;
    else
        return  the black height of left child as its black height;
}
```

# Chapter 3: Testing Results

All inputs are also stored in txt file in the folder "document". You should copy the data there to test the program. Inputs here just for read.

| | |
|---|---|
| TestCase 1 | case purpose |
| | sample |
| | expected result |
| | Yes, No, No(Normally return) |
| | actual behavior |
| | Yes,No,No(Normally return) |
| | possible cause |
| | |
| | current status |
| | pass |
| | Input |
| | 3 |
| | 9 |
| | 7 -2 1 5 -4 -11 8 14 -15 |
| | 9 |
| | 11 -2 1 -7 5 -4 8 14 -15 |
| | 8 |

| | |
|---|---|
| | 10 -7 5 -6 8 15 -11 17 |
| TestCase 2 | case purpose |
| | contrast: red root and black root |
| | expected result |
| | No,Yes(Normally return) |
| | actual behavior |
| | No,Yes(Normally return) |
| | possible cause |
| | |
| | current status |
| | pass |
| | Input |
| | 2<br>9<br>-7 2 1 5 -4 11 8 14 -15<br>9<br>7 2 1 5 -4 11 8 14 -15 |
| TestCase 3 | case purpose |
| | a right child be the INT_MAX(black) |
| | expected result |
| | Yes(Normally return) |
| | actual behavior |
| | No(Normally return) |
| | possible cause |
| | using '>=' instead '>' to compare with upper bound |
| | current status |
| | corrected |
| | input |
| | 1<br>3<br>7 1 2147483647 |

| | |
|---|---|
| TestCase 4 | case purpose |
| | Property e) of red-black tree violated |
| | expected result |
| | Yes,No(Normally return) |
| | actual behavior |
| | Yes,No(Normally return) |
| | possible cause |
| | |
| | current status |
| | pass |
| | input |
| | 2<br>5<br>10 1 -12 11 13<br>4<br>10 -12 11 13 |
| TestCase 5 | case purpose |
| | Property d) of red-black tree violated |
| | expected result |
| | Yes,No(Normally return) |
| | actual behavior |
| | Yes,No(Normally return) |
| | possible cause |
| | |
| | current status |
| | pass |
| | input |
| | 2<br>3<br>3 -1 -4<br>4 |

| | |
|---|---|
| | 3 -1 -2 -4 |
| TestCase 6 | case purpose |
| | the size reaches the MAXNODES |
| | expected result |
| | Yes(Normally return) |
| | actual behavior |
| | Yes(Normally return) |
| | possible cause |
| | |
| | current status |
| | pass |
| | Input |
| | 1<br>32<br>100 -54 22 10 5 20 -21 35 27 50 77 61 50 70 83 80 90 176 -132 116 108 124 153 140 160 -290 220 200 278 1000 500 2000 |
| TestCase 7 | case purpose |
| | Show after inserting a node(written by myself),RBT properties hold |
| | expected result |
| | Yes*10(Normally return) |
| | actual behavior |
| | Yes*10(Normally return) |
| | possible cause |
| | |
| | current status |
| | pass |
| | input |
| | 10<br>1<br>1<br>2 |

|  |  |
|---|---|
|  | 1 -10<br><br>3<br><br>8 -1 -10<br><br>4<br><br>8 1 -4 10<br><br>5<br><br>8 4 -1 -7 10<br><br>6<br><br>8 -4 1 7 -6 10<br><br>7<br><br>8 -4 1 6 -5 -7 10<br><br>8<br><br>8 -4 1 6 -5 -7 10 -9<br><br>9<br><br>8 -4 1 -2 6 -5 -7 10 -9<br><br>10<br><br>8 -4 2 -1 -3 6 -5 -7 10 -9 |
| TestCase 8 | case purpose |
|  | Random Insert Of MAXNODES |
|  | expected result |
|  | Yes(Normally return) |
|  | actual behavior |
|  | Yes(Normally return) |
|  | possible cause |
|  |  |
|  | current status |
|  | pass |
|  | input |
|  | 1<br><br>30 |

| | |
|---|---|
| | 17631 -4266 3237 1205 -1173 3550 11904 -10439 6810 -5004 -6863 11245 -15194 12723 -13123 17133 -25746 24006 18682 -18395 -19693 24092 27220 27161 -28484 27879 -27794 -27913 30295 -32131 |
| TestCase 9 | case purpose |
| | MAXNODES=90 |
| | expected result |
| | Yes(Normally return) |
| | actual behavior |
| | Yes(Normally return) |
| | possible cause |
| | |
| | current status |
| | pass |
| | Input<br>(you should change the MAXNODES of the source code to test this input) |
| | 1<br>90<br>22313 -13018 5393 -3918 2460 -1635 932 -904 1825 -1666 -2370 3192 -2821 -3564 4580 4079 -4539 -4955 4628 4972 -5352 7921 -7379 7357 -7000 7779 -7524 -11088 10501 -9754 -11038 11948 -11139 -12304 19977 -16045 13438 13131 -14889 14111 -14355 15430 -16032 18991 -17140 16918 -16353 -17115 17937 19746 -19456 -19806 21527 -21195 20679 -20359 21212 21875 -21680 -22129 27120 -24525 23579 22502 24161 -24057 -24246 26303 -25811 24908 -25709 26067 -26685 26398 26802 -27048 -29325 27149 27133 -27134 27411 -28891 30504 -29862 29726 -29668 30131 30960 -30884 -31796 |
| TestCase 10 | case purpose |
| | batches = 13,MAXNODES=100 |
| | expected result |
| | Yes*13(Normally return) |
| | actual behavior |
| | Yes*13(Normally return) |

| | |
|---|---|
| | possible cause |
| | |
| | current status |
| | pass |
| | Input<br>(you should change the MAXNODES of the source code to test this input) |
| | 13<br>10<br>11039 -5799 1167 10072 -8454 -18716 11621 29873 -29317<br>-31970<br>96<br>15289 -10485 4506 -1987 1604 -799 74 1515 -1425 1859 -<br>1794 -1904 3315 2171 -2039 -2450 -3497 3340 -3358 4051<br>-4274 6801 -6102 5546 -4559 6505 -6689 -8964 7607 -6999<br>-7698 9600 -9567 -9726 14651 11202 10495 -10989 -11849<br>11225 12702 -12097 -14096 14710 14661 -14675 15097 -<br>15102 -22896 17773 16424 16232 -16524 16502 16534 -<br>17436 -20015 19769 18621 -17895 -19094 19839 -19927<br>22386 -21873 20347 -21740 22100 22825 -22826 28110<br>25746 -25054 24656 -24043 -24687 25384 27161 -26331 -<br>27552 -29962 28555 28248 -28334 28975 -28586 -29524<br>31976 -30179 29990 30757 -31688 -32322 32220 32739 -<br>32657<br>44<br>14094 6564 3451 2828 -1157 -3408 4810 -4758 8297 -6998<br>6834 8061 -8030 -11273 11210 -9885 -11260 12677 19999<br>16086 -14780 14190 -14213 15440 -14874 -15598 -17484<br>17142 -16980 18386 -30045 25469 20956 -20943 -23969 -<br>26885 25625 -25474 -25669 28971 31855 30169 31974 -<br>32393<br>71<br>12167 3535 1224 -733 523 -677 841 -2490 1389 3302 -<br>2763 -3367 -5843 3614 3589 -4814 4364 -4756 5558 -5013<br>9482 -7570 6311 7976 -9239 -11041 10918 -10364 11668 -<br>11243 -18958 16250 14770 -12730 12673 13246 -13162 -<br>13472 15479 -15959 16740 16485 -18355 16983 -17608<br>18409 -18656 26232 24012 -21602 19660 23194 -23067 - |

|  |  |
|---|---|
|  | 24896 24170 -24863 25138 -25006 -25293 -31271 28633 -27093 26666 27578 -28091 29667 -29549 32416 31934 -31303 32579<br><br>96<br><br>13956 2910 823 559 497 -214 811 -813 2381 -1469 1304 2268 -1995 2870 -2668 -2897 7914 6503 5045 -3976 -6214 6669 -6581 -6767 -10609 9199 8240 10504 11275 10758 -11045 -12836 11292 13634 -12900 25127 -20343 16561 14836 -14547 14314 -14391 14589 -15684 15516 -15286 15850 -19437 17212 16748 17610 -17419 -19083 19634 19589 -19955 19680 -19704 20203 22098 21120 20628 21799 -21727 23008 22592 -22232 -22724 -23752 23226 23818 -24251 28368 27830 -25823 25602 -25137 26625 -27741 -28174 28042 28291 -28260 31984 -29633 29362 -28754 31142 -29765 -32165 32067 -32030 -32120 32508 -32355 -32721<br><br>65<br><br>14411 10708 -5844 2244 -150 107 -104 1780 -4383 2869 -2632 -3601 5174 -5081 -5829 8472 6896 -5881 -9650 9257 -8870 10107 -10147 12264 10833 13708 -13390 -14342 22999 -19207 16981 -15050 14585 -14991 16487 -15752 -18339 18203 18767 -19187 20159 19942 -19803 20364 -22982 -28659 24806 23503 -23288 -25728 25024 -25688 26464 -26223 -26686 29805 -29488 28865 29624 -29660 -31772 31395 -31392 32079 -32424<br><br>8<br><br>20625 -2903 2855 8815 -5644 31460 -29067 -31511<br><br>28<br><br>23568 -14138 5196 77 -3210 -12290 8211 -6875 13055 -12868 17162 14287 -14268 -16377 19210 -21292 -25758 24799 24259 25361 -25608 30512 -29032 27534 -26874 29166 32128 -30593<br><br>22<br><br>18763 7359 -3829 34 -622 7270 -4553 -13649 8337 -7442 14444 -16093 27486 -22598 19177 -19793 23517 -23864 -29804 29346 32127 -32629<br><br>77 |

| | |
|---|---|
| | 17681 -7821 4900 -3765 2439 939 -1989 3618 4583 -4156 4021 -3937 4289 4732 6202 5297 -5194 -5740 -6998 6967 7789 -7168 13919 11908 -9687 8283 10667 -9955 -11177 -12682 12124 -12458 13609 -13162 15815 -14843 14479 -14459 15657 16180 -15861 -16454 26688 -20545 18899 -18250 17954 -17741 18744 -18863 -19608 19154 19622 -20113 23790 -22100 20992 -20757 -21196 22664 -22877 25332 -24895 -25936 27876 -27465 26962 -27271 27757 -27665 -27831 -29297 28765 -27958 30551 -30476 -32430 |
| | 11 |
| | 4967 3314 1612 -3215 3462 20303 -14778 7891 16458 -17266 29488 |
| | 52 |
| | 16119 11246 -2626 1969 1569 -1810 2590 4379 2705 -2828 -5783 5400 7465 -10390 13339 -11962 11891 12960 -12866 -14077 14011 14250 -14228 -14365 24170 20643 -17953 17286 -17230 -17761 19961 -18604 -22730 21606 -21232 23631 -23861 -26726 25663 25010 -25445 26266 -26716 29901 -28243 28175 -27923 29165 -29175 30362 -30011 -31345 |
| | 70 |
| | 14268 9760 -4321 2012 -403 1 579 -551 -1364 -3325 3251 -2554 4078 -3946 6172 5261 -8981 6332 9679 -9279 -12654 11211 10276 -10435 -11597 11491 12215 -11839 -12442 13582 -13097 12726 -12717 -12939 13159 -13716 13648 14087 -13851 22562 17674 -16159 14991 -14428 -15168 16957 -16369 -17059 -21080 18469 -18325 -19008 21507 -21315 -22108 -28507 27062 -25392 25042 25580 -25756 27763 -27752 -28437 29936 29250 -28991 -29463 32210 -31089 |

# Chapter 4: Analysis and Comments

**Using the detailed pseudo code to analyze.**

## Time Complexity:

## (one test case)

**Analyze:**

**In function Input:**

Declaration and assignment of the formal parameter cost $\Theta(1)$ time;

read operation cost $\Theta(1)$ time;

The loop is executed (size) times, costing $\Theta(\text{size})$ time;

**Therefore T(Input)= $\Theta$(size);**

**In function Judge:**

Declaration and assignment of the formal parameter cost $\Theta(1)$ time;

The statements except two callings of function **Judge** itself cost $\Theta(1)$ time;

The **Judge** called on left child costs T(Judge, left)+ $\Theta(1)$

The **Judge** called on right child costs T(Judge, right)+ $\Theta(1)$

For Convenience, we set T(Judge, leaf)= 0;

Hence, for any internal node,

$$\mathrm{T(Judge, node)} = \mathrm{T(Judge, right)} + \mathrm{T(Judge, left)} + \Theta(1);$$

Let N(root) be the number of descendant internal nodes of root including itself.

We will show that T(Judge, root)= $\Theta(\mathrm{N(root)})$ by strong induction.

Let P(n) be the proposition that for all $k \in \mathbf{N}$ and k<=n, if N(root)=k, then T(Judge, root)= $\Theta(\mathrm{N(root)})$.To show P(n) is true for $n \in \mathbf{N}$,

BASIC STEP:

if n = 0, then the root with N(root)=0 is a leaf itself.

Thus,

$$\mathrm{T(Judge, root)} = 0 = \Theta(0);$$

INDUCTIVE STEP:

Inductive hypothesis : P(k) holds, that is, for all $t \in \mathbf{N}^+$ and t<=k, if N(root)=t, then T(Judge, root)= $\Theta(\mathrm{N(root)})$.

Using inductive hypothesis, we will show that P(k+1) holds.

When  N(root)=k+1, $\mathrm{T(Judge, root)} = \mathrm{T(Judge, right)} + \mathrm{T(Judge, left)} + \Theta(1) = \Theta\big(\mathrm{N(left)}\big) + \Theta\big(\mathrm{N(right)}\big) + \Theta(1) = \Theta(\mathrm{N(left)} + \mathrm{N(right)} + 1) = \Theta(\mathrm{N(root)})$.

Hence, we have shown that P(n) holds.

**Therefore, T(Judge, root)=** $\boldsymbol{\Theta(\mathrm{N(root)})}$**,N(root) is the number of internal nodes in the tree rooted by root.**

**In function TestCase:**

statements except function callings cost $\Theta(1)$ time

the calling of function **InPut** cost $\Theta(\mathrm{size})$ time

the calling of function **Judge** cost $\Theta(\mathrm{size})$ time (may not be excuted)

**Therefore, T(TestCase)=O(size) (some statements may not be executed).**

**In a word, for each test case, the time complexity is O(n), n is the number of internal nodes given.**

**In function main:**

**There are test cases of number (batches). Assume the** $\mathbf{i}^{th}$ **test case has** $\mathbf{N}_i$ **nodes (i= 1,2,3, … , batches), the total time complexity is**

$$\boldsymbol{T} = \sum_{i=1}^{batches} \boldsymbol{O}(\mathbf{N}_i) = \boldsymbol{O}(\boldsymbol{N})$$

## Space Complexity:

## (one test case)

**Analyze:**

**In function Input:**

***Space Used:***

**dynamic(not counted):**

nodes[]: O(size)

**local:**

node: O(1)

i :O(1)

size :O(1)

**Therefore, S(Input)=O(size)**

**In function Judge:**

***Space Used:***

**dynamic(not counted):**

nodes[]=O(size)

**local:**

nodes :O(1)

RootIndex: O(1)

size: O(1)

UpBound: O(1)

ReturnRightBoundIndex: O(1)

LeftBlackNodes: O(1)

LeftChildIndex: O(1)

RightChildIndex: O(1)

LeftBlackNodes: O(1)

max{S(Judge, left),S(Judge, right) (This two statements take turns to use memory.)

Therefore, S(Judge, right) = max{S(Judge, left),S(Judge, right)}+O(1).

Assume the depth of the tree rooted by root is p, then S(Judge, root)= O(p) (You can verify it using mathematical induction).

**Since $\log_2 N(root) < p < N(root)$ , S(Judge, root)=O(N(root)) and S(Judge, root)=Ω(N(root)). (N(root) is the number of descendant internal nodes of root).**

**In function TestCase:**

**dynamic(not counted):**

node[]: O(size)

**local:**

size: O(1)

NoUse[]: O(1)

flag: O(1)

S(Judge, nodes[0]): O(size) and Ω(log(size))

**Therefore, S(TestCase)=O(size) and Ω(log(size))**

**If we add the space of dynamic memory, that is O(size), the space complexity of one test case is S=O(size).**

**In function main:**

***Space Used:***

**dynamic:**

nodes[]: O(max size)

**local:**

batches: O(1)

**Therefore, the total space complexity of the program is**

$$\mathbf{S = O(\max size) = \boldsymbol{O(max\{N_i\}), i = 1, 2, 3, \dots, batches}}$$

## Comments:

Possible Improvement of Algorithm:

1)

Using Online Algorithm, that is, processing the input one by one, instead begin processing the data after the input is all read and stored.

By Online Algorithm, the space complexity in best case can be Θ(log(N)),namely S=Ω(log(N)).(If the tree to be test is a RBT, S=Θ(log(N))

2)

We can rewrite the function Judge in an iterative way, that is, using loops and stack to replace the recursive statements.

Though the time and space complexity will not be changed by this way, the space of some arguments of function Judge like nodes, size that are always the same can be reduced.

## Appendix:    Source Code (in C)

```c
//Project1 Normal

#include <assert.h>
#include <stdlib.h>
#include <stdio.h>

#define MAXNODES 30
#define BADVALUE -
1 //If a function returns BADVALUE, it tells the program that the test
is false and to terminate immediately
#define NULLPOS 1 //If the index of a node is assigned as NULLPOS, it m
eans this node is a leaf(null)
#define INTMAX 2147483647

/* The function TestCase is to
 * receive input and judge whether
 * it is a Red-Black Tree. the
 * parameter nodes are used to
 * store the data. It prints
 * "Yes" or "No" to show the result.*/
void TestCase(int nodes[]);

/* The function Input is to
 * receive input, store them
 * in the array-type variable
 * "nodes", and return the
 * size of the input*/
int InPut(int nodes[]);

/* This is the core function
 * of this program to judge if
 * the search tree put in is a subtree of
 * Red-Black Tree. Actually, The
 * tree is stored in an array
 * rather than a linked structure,
 * thus, extra operations are
 * needed to find the left and
 * right children.
 *
 * The function will be recursively
 * called by itself with argument
 * RootIndex being the index of the root of a
 * subtree.
 *
 * Argument size tells the function
 * the size of nodes;
 *
 * Argument UpBound assign the
 * least up bound of current
 * subtree;
 *
 * Argument ReturnRightBoundIndex is a pointer
 * given by the calling function,
 * and is expected to return the
 * right bound of the tree in the
 * called function.
 */
int Judge(int nodes[],int RootIndex,int size,int UpBound,int * ReturnRi
ghtBoundIndex);

int main(){
    int nodes[MAXNODES];
    int k;
    scanf("%d",&k);
    while (k--){
        TestCase(nodes);
        if(k) putchar('\n');
    }
    return 0;
}

void TestCase(int nodes[]){
    int size=InPut(nodes);
    int buf;//It is only used to fullfil the fourth parameter(ReturnRig
htBoundIndex) of function Judge.
    int flag= (nodes[0]<0)? BADVALUE: Judge(nodes,0,size,INTMAX,&buf);
    if(flag!=BADVALUE) printf("Yes");
    else printf("No");
}

int InPut(int nodes[]){
    int size , i;
    scanf("%d",& size);
    for(i=0;i< size ;i++) scanf("%d", nodes +i);

    return size;
```

```
}

int Judge(int nodes[],int RootIndex,int size,int UpBound,int * ReturnRi
ghtBoundIndex){
    int LeftBlackNodes,RightBlackNodes,LeftChildIndex,RightChildIndex;
    /* LeftBlackNodes: the number of black nodes in the left subtree
     * RightBlackNodes: the number of black nodes in the right subtree
     * LeftChildIndex: the index of left child
     * RightChildIndex: the index of right child*/

    /*The root's children are both leaves(null)*/
    if(RootIndex>=size-1){
        *ReturnRightBoundIndex=NULLPOS;
        return 1+(nodes[RootIndex]>0);
    }

    /* Judge whether root’s left child is a leaf(null) */
    if(abs(nodes[RootIndex+1])>abs(nodes[RootIndex])) LeftBlackNodes=1,
LeftChildIndex=NULLPOS,RightChildIndex=RootIndex+1;


    else
    {
        LeftChildIndex=RootIndex+1;
        if(nodes[RootIndex]<0 && nodes[LeftChildIndex]<0) return BADVAL
UE;//Red Node's left child is not black
        LeftBlackNodes=Judge(nodes,LeftChildIndex,size,abs(nodes[RootIn
dex]),&RightChildIndex);
        if(LeftBlackNodes==BADVALUE) return BADVALUE;//subtree violates
 the rule
    }

    if(RightChildIndex==NULLPOS || abs(nodes[RightChildIndex])>=UpBound
){//right child is found a leaf(null)
        RightBlackNodes=1;
        if(RightChildIndex!=NULLPOS) *ReturnRightBoundIndex=RightChildI
ndex;
    }
    else{
        if(nodes[RootIndex]<0 && nodes[RightChildIndex]<0) return BADVA
LUE;//Red Node's left child is not black
        RightBlackNodes=Judge(nodes,RightChildIndex,size,UpBound,Return
RightBoundIndex);
        if(RightBlackNodes==BADVALUE) return BADVALUE;//subtree violate
s the rule
```

```
    }

    /*The property cannot be met that the numbers of black nodes
     in all simple path from the i_th node to an arbitrary decendant
     leaf are not equal.
    */
    if(LeftBlackNodes!=RightBlackNodes) return BADVALUE;

    return LeftBlackNodes+(nodes[RootIndex]>0);// return the number of
black nodes in the simple path from the i_th node to one of its leaf.
}
```

## Declaration

***I hereby declare that all the work done in this project titled " Is It A Red-Black Tree" is of my independent effort.***