# Test Data

### Path Constraints

getRoot():

Use on RBTreeWithFaults: RBTreeWithFaults

RBTreeWithFaults ():

None

RBTreeWithFaults (RBNode root):

None: root = new RBNode(params)

createInfinityNode ():

None:

createInfinityNode (RBNode root):

None root = new RBNode(params)

createNullNode(RBNode parent):

None parent = this.Root

empty()

this.Root.Left!=null

search(**int** k)

DEPENDENT ON SEARCHNODE

SearchNode(int k, RBNode node)

k!=null, node!=null node.Key == Integer.MAX\_VALUE

node.Key == Integer.MIN\_VALUE,

node.Parent.Key == Integer.MAX\_VALUE

node.Key == Integer.MIN\_VALUE,

node.Parent.Key != Integer.MAX\_VALUE

node.Key != (Integer.MAX\_VALUE || Integer.MIN\_VALUE),

node.Key > k

node.Key != (Integer.MAX\_VALUE || Integer.MIN\_VALUE),

node.Key < k

node.Key != (Integer.MAX\_VALUE || Integer.MIN\_VALUE),

node.Key == k

insert(int k, string v)

Dependent on search, new RB node, fix up tree

fixUpTree(RBNode z)

z!=null z.Parent.Black == false

z.Parent == z.Parent.Parent.Left

y.Black == true

this.Root.Left.Black ==false

z.Parent.Black == false

z.Parent == z.Parent.Parent.Left

y.Black == false

z == z.Parent.Right

this.Root.Left.Black ==false

z.Parent.Black == false

z.Parent != z.Parent.Parent.Left

y.Black == false

this.Root.Left.Black ==false

z.Parent.Black == false

z.Parent != z.Parent.Parent.Left

y.Black == true

z == z.Parent.Left

this.Root.Left.Black == false

leftChild(RBNode x,RBNode y)

x!=null, y!=null x= new RBNode(params) y = new RBNode(params)

rightChild(RBNode z, RBNode y)

x!=null, y!=null x = new RBNode(params) y = new RBNode(params)

transplate(RBNode x, RBNode y)

x!=null, y!=null, x==x.Parent.Left x= new RBNode(params).parent.left, y = new RBNode(params)

x!=null, y!=null, x!=x.Parent.Left x= new RBNode(params).parent.right, y = new RBNode(params)

leftRotate(RBNode y)

y!=null, y.Right!=null

rightRotate(RBNode y)

y!=null, y.Left!=null

delete(int k)

k!=null z.Key != k

z.Key == k

z.Left.Key == Integer.MIN\_VALUE

isBlackOriginalY == true

z.Key == k

z.Right.Key == Integer.MIN\_VALUE

isBlackOriginalY == true

z.Key == k

z.Left.Key != Integer.MIN\_VALUE

z.Right.Key != Integer.MIN\_VALUE

y.Parent == z

isBlackOriginalY == true

z.Key == k

z.Left.Key != Integer.MIN\_VALUE

z.Right.Key != Integer.MIN\_VALUE

y.Parent != z

IsBlackOriginalY == true

deleteFixup(RBNode x,RBNode y)

dependent on many things

min()

this.Root.Left!=null

minimumNode(RBNode node)

node!=null, node.Key==integer.MAX\_VALUE

node!=null, node.Key==integer.MAX\_VALUE, , node.Left.Key==integer.MAX\_VALUE

node!=null, node.Key!=integer.MAX\_VALUE, node!=null, node.Left.Key!=integer.MAX\_VALUE

isInfinityNode(RBNode node)

node!=null

isNullNode(RBNode node)

node!=null

max()

this.Root.Left!=null this.empty() == true

this.empty() != true

maxValue(RBNode node)

node!=null, node.Right.Key==integer.MAX\_VALUE

node!=null, node.Right.Key!=integer.MAX\_VALUE

keysToArray()

keysString!=null keysString.equals(“”) == true

keysString.equals(“”) == false

ArrayOfStringsToArrayOfInts(String[] strArr)

strArr={“1”}

ElementsToString(RBNode node, boolean key)

node!=null, key!=null node == Integer.MIN\_VALUE

node == Integer.MAX\_VALUE

node != Integer.MIN\_VALUE

node != Integer.MAX\_VALUE

key == true

node.Left != Integer.MIN\_VALUE

node.Right != Integer.MIN\_VALUE

node != Integer.MIN\_VALUE

node != Integer.MAX\_VALUE

key == true

node.Right != Integer.MIN\_VALUE

node.Left == Integer.MIN\_VALUE

node != Integer.MIN\_VALUE

node != Intger.MAX\_VALUE

key == ture

node.Right == Integer.MIN\_VALUE

node.Left != Integer.MIN\_VALUE

node != Integer.MIN\_VALUE

node != Integer.MAX\_VALUE

key == true

node.Right == Integer.MIN\_VALUE

node.Left == Integer.MIN\_VALUE

node != Integer.MIN\_VALUE

node != Integer.MAX\_VALUE

key != true

node.Left != Integer.MIN\_VALUE

node.Right != Integer.MIN\_VALUE

node != Integer.MIN\_VALUE

node != Integer.MAX\_VALUE

key != true

node.Right != Integer.MIN\_VALUE

node.Left == Integer.MIN\_VALUE

node != Integer.MIN\_VALUE

node != Intger.MAX\_VALUE

key != ture

node.Right == Integer.MIN\_VALUE

node.Left != Integer.MIN\_VALUE

node != Integer.MIN\_VALUE

node != Integer.MAX\_VALUE

key != true

node.Right == Integer.MIN\_VALUE

node.Left == Integer.MIN\_VALUE

valuesToArray()

**this**.Root.Left!= null

print()

queue!=null queue.isEmpty == false

size > 0

curNode.Left != null

curNode.Right != null

Size()

**this**.Root.Left!=null

sizeCalc(RBNode node)

node.Left!=null, node.Right!=null node.Left != Integer.MIN\_VALUE

node.Right != Integer.MIN\_VALUE

node.Left == Integer.MIN\_VALUE

node.Right != Integer.MIN\_VALUE

node.Left != Integer.MIN\_VALUE

node.Right == Integer.MIN\_VALUE

node.Left == Integer.MIN\_VALUE

node.Right == Integer.MIN\_VALUE

RBNode(String value,**int** key, RBNode left, RBNode right,RBNode parent)

None

RBNode(String value, int key, RBNode parent)

value!=null, key!=null

## Data Definitions:

A,B,C,D,E= RBNode(“String”, {1-5}, given)

V,W,X,Y,Z= RBNode(“String”, {6-10}, given)

Tree= RBTreeWithFaults(given)