Software Testing Project Report

Maheedhar (IMT2019074)
D V N P S M Manohar (IMT2019025)

What is CFG?

A control flow graph (CFG) is the graphical representation of control flow or computation during the execution of programs or applications. Control flow graphs are mostly used in static analysis as well as compiler applications, as they can accurately represent the flow inside of a program unit.

Project Information

The Source code for the project is a terminal based menu interface to execute different binary tree algorithms. The terminal menu interface looks as follows -

```
********************************************************************************
Binary Tree operations Menu

1.) PreOrder Traversal of Binary tree

2.) InOrder Traversal of Binary tree

3.) PostOrder Traversal of Binary tree

4.) Level Order Traversal of Binary tree

5.) Height of Binary tree

6.) Check if the Binary Tree is Symmetric

7.) Check if two Binary Trees are Mirrors of each other

8.) Bottom View of a Binary Tree

9.) Check the Children Sum Property in a Binary Tree

10.) Number of Nodes of a Binary Tree

0.) Exit

********************************************************************************
```

We have implemented 10 binary tree algorithms.The user would be asked which algorithm they want and the user should provide the number corresponding to that algorithm.

Tools Used

- Java
- JUnit
- Code2flow

GitHub Repository Link

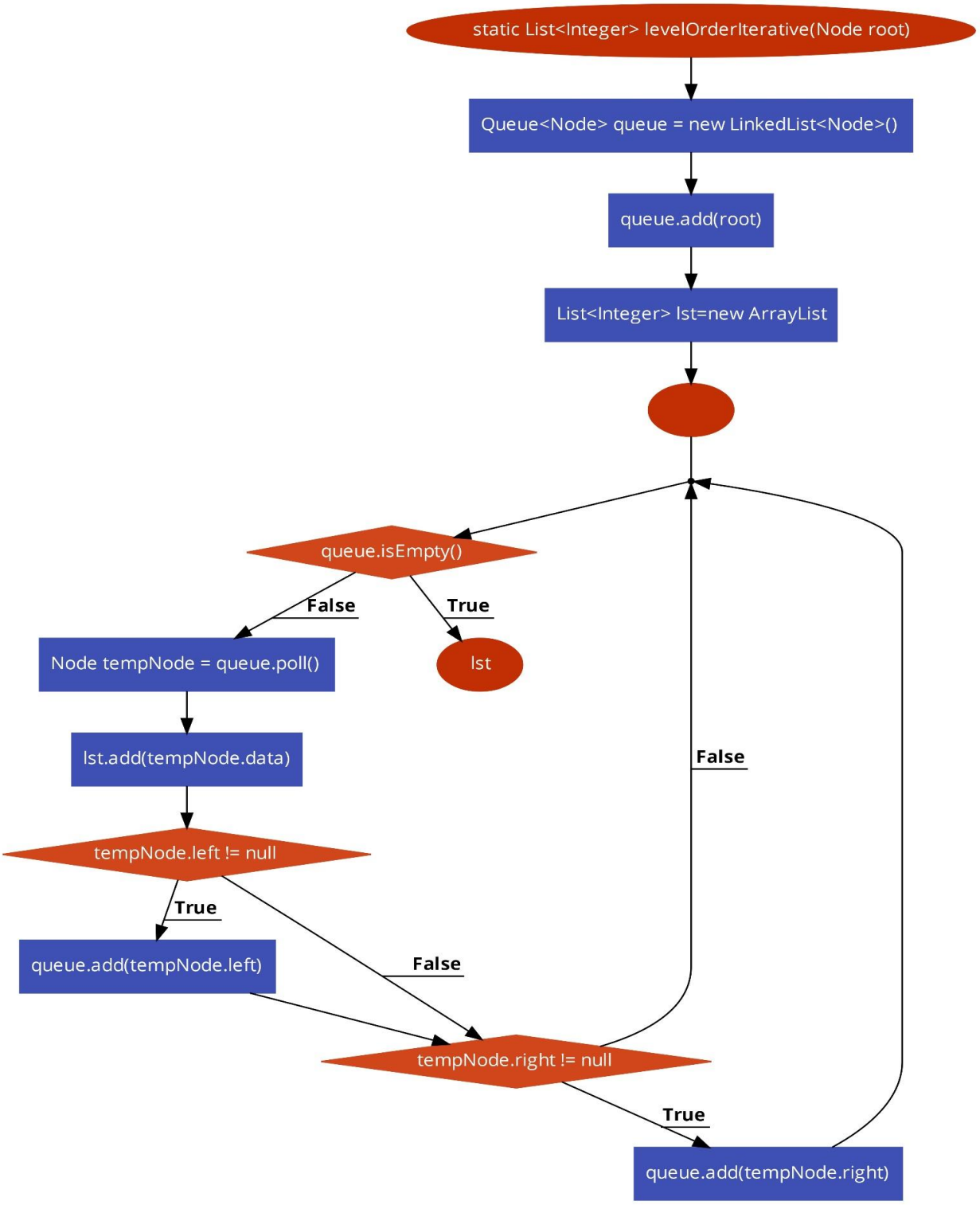
https://github.com/manoharsuggula/SoftwareTesting-Project-Report

Functions used in the source code

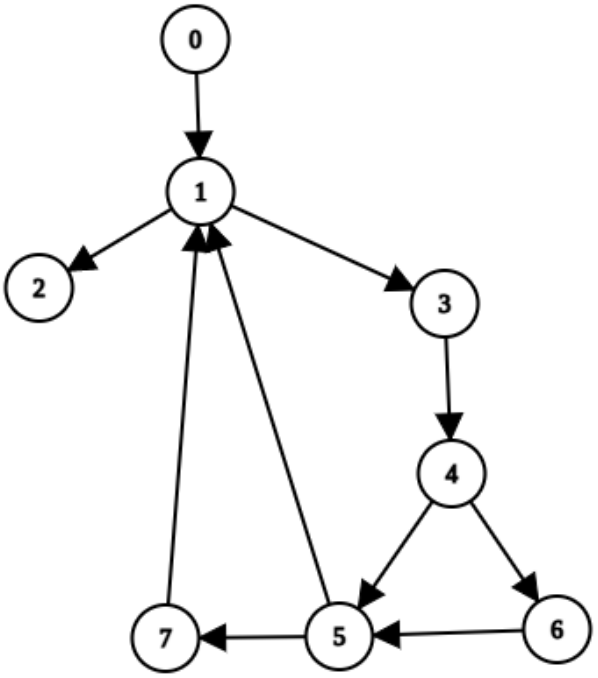
Function	Description
PreOrderIterative	Prints the preOrder traversal of a Binary Tree
PostOrderIterative	Prints the postOrder traversal of a Binary Tree
InOrderIterative	Prints the InOrder traversal of a Binary Tree
levelOrderIterative	Prints the levelOrder traversal of a Binary Tree
bottomView	Prints the bottom View of a Binary Tree
areMirrors	Checks whether the 2 trees are mirrors to each other
CheckChildrenSum	For every node of the tree, it checks whether the sum of values of children is equal to the value of the node.
isSymmetric	Checks whether the Tree is symmetric to itself or not
numNodes	Returns the number of nodes of a Binary Tree
Height	Returns the Height of Tree

Level order

The control flow graph of Level Order Traversal is



This is simplified into a simple graph as follows



We performed the Edge coverage and Prime coverage on this simplified graph.

Edge Coverage:

The Test requirements for Edge Coverage are
{ [0,1], [1,2], [1,3], [3,4], [4,5], [4,6], [6,5], [5,1], [7,1], [5,7] **}**

The Test Paths are
[0,1,3,4,6,5,7,1,2]
[0,1,3,4,5,1,3,4,5,1,2]
[0,1,2]

Prime Path Coverage:

The Test requirements for Prime path Coverage are
{ [3,4,6,5,7,1,2], [3,4,6,5,7,1,3], [1,3,4,6,5,7,1], [0,1,3,4,6,5,7], [4,6,5,7,1,3,4], [5,7,1,3,4,6,5], [7,1,3,4,6,5,7], [6,5,7,1,3,4,6], [3,4,5,7,1,3], [3,4,5,7,1,2], [3,4,6,5,1,2], [3,4,6,5,1,3], [0,1,3,4,5,7], [1,3,4,5,7,1], [1,3,4,6,5,1], [4,5,7,1,3,4], [7,1,3,4,5,7], [5,1,3,4,6,5], [5,7,1,3,4,5], [4,6,5,1,3,4], [6,5,1,3,4,6], [3,4,5,1,3], [4,5,1,3,4], [3,4,5,1,2], [1,3,4,5,1], [5,1,3,4,5], [0,1,2] **}**

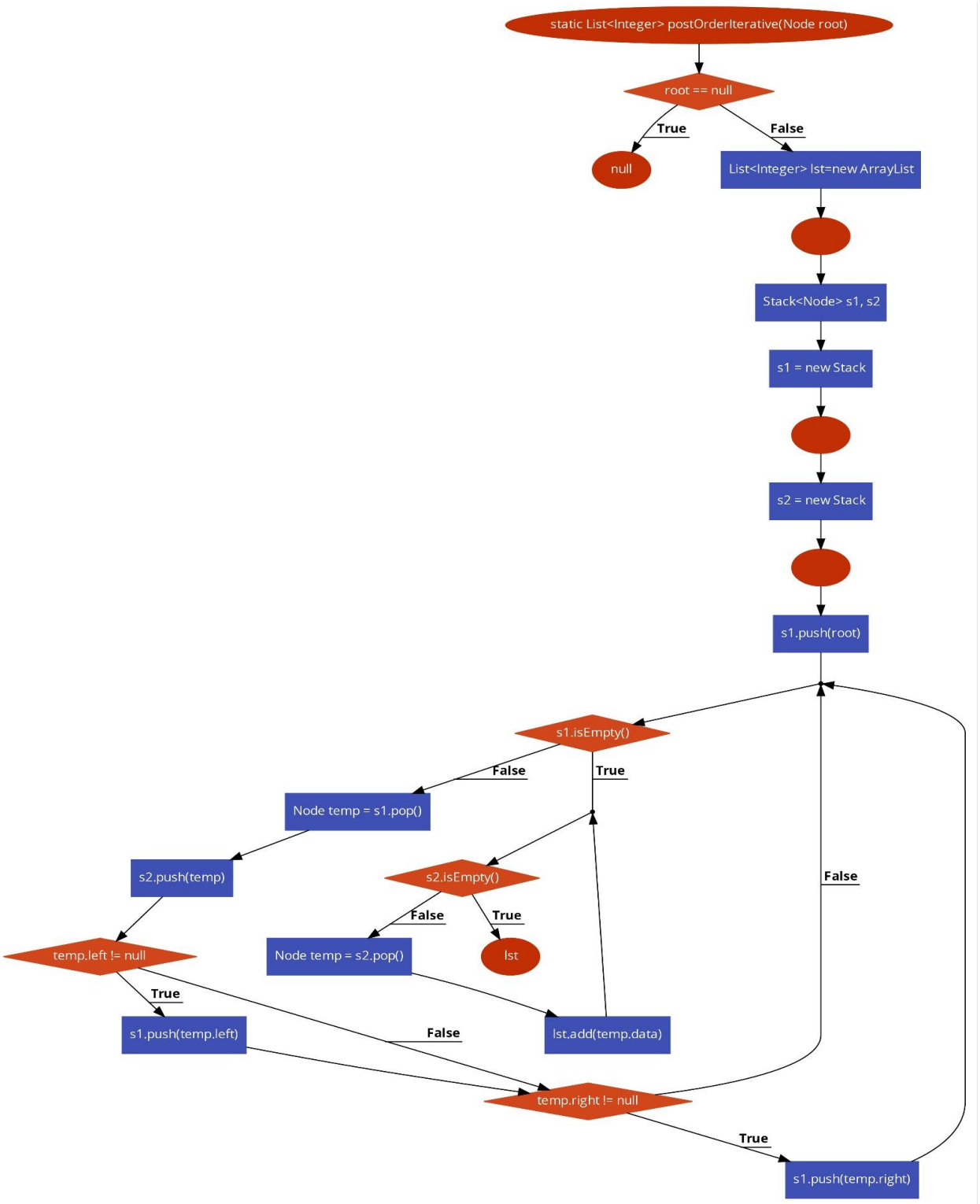
The Test Paths for prime path coverage are

7 test paths are needed for Prime Path Coverage using the prefix graph algorithm

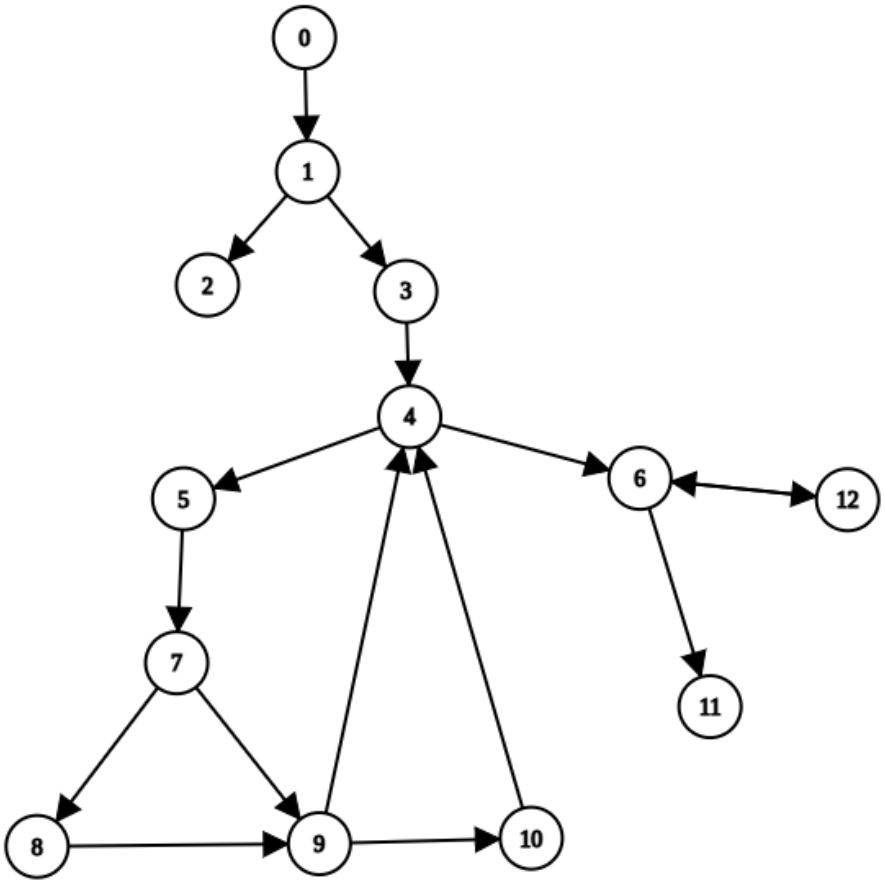
Test Paths	**Test Requirements that are toured by test paths directly / side trips**	**Test Case Number**
[0,1,2]	[0,1,2]	1
[0,1,3,4,5,1,2]	[3,4,5,1,2], [1,3,4,5,1]	2
[0,1,3,4,5,7,1,3,4,6,5,1,3,4,5,1,2]	[5,7,1,3,4,6,5], [3,4,5,7,1,3], [3,4,6,5,1,2], [0,1,3,4,5,7], [1,3,4,5,7,1], [1,3,4,6,5,1], [4,5,7,1,3,4], [3,4,5,1,2], [3,4,5,7,1,3], [5,7,1,3,4,6,5],[0,1,3,4,5,7], [1,3,4,5,7,1], [4,5,7,1,3,4],	3
[0,1,3,4,6,5,7,1,3,4,5,1,3,4,5,1,2]	[3,4,5,1,3], [4,5,1,3,4], [1,3,4,5,1], [5,1,3,4,5],[3,4,6,5,7,1,3], [1,3,4,6,5,7,1], [4,6,5,7,1,3,4],[5,7,1,3,4,5]	4
[0,1,3,4,6,5,7,1,3,4,6,5,7,1,3,4,5,1,3,4,5,1,3,4,5,1,2]	[7,1,3,4,6,5,7] ,[3,4,6,5,7,1,3], [1,3,4,6,5,7,1] [0,1,3,4,6,5,7],[4,6,5,7,1,3,4], [5,7,1,3,4,6,5], [6,5,7,1,3,4,6],[5,1,3,4,5]	5
[0,1,3,4,6,5,1,3,4,5,7,1,3,4,5,1,2]	[3,4,6,5,1,3],[1,3,4,5,7,1],[1,3,4,6,5,1], [4,5,7,1,3,4], [5,7,1,3,4,5], [4,6,5,1,3,4],	6
[0,1,3,4,6,5,1,3,4,6,5,1,3,4,6,5,7,1,3,4,5,7,1,3,4,6,5,1,3,4,5,7,1,3,4,5,7,1,2]	[7,1,3,4,5,7], [5,1,3,4,6,5], [6,5,1,3,4,6],	Not Feasible

The prime path coverage subsumes the Edge Coverage.

Post order


static List<Integer> postOrderIterative(Node root)
root == null
True
False
null
List<Integer> lst=new ArrayList
Stack<Node> s1, s2
s1 = new Stack
s2 = new Stack
s1.push(root)
s1.isEmpty()
False
True
Node temp = s1.pop()
s2.push(temp)
s2.isEmpty()
False
True
temp.left != null
Node temp = s2.pop()
lst
True
s1.push(temp.left)
False
lst.add(temp.data)
False
temp.right != null
True
s1.push(temp.right)

This is simplified into a simple graph as follows



We performed the Edge coverage and Prime coverage on this simplified graph.

Edge Coverage:

The Test requirements for Edge Coverage are
{ [0,1], [1,2], [1,3], [3,4], [4,5], [4,6], [5,7], [7,8], [7,9], [8,9], [9,4], [10,4], [9,10], [6,11], [6,12], [12,6] **}**

The Test Paths are

[0,1,3,4,6,12,6,12,6,11]
[0,1,3,4,5,7,9,4,6,11]
[0,1,3,4,5,7,8,9,10,4,6,11]
[0,1,2]

Prime Path Coverage:

The Test requirements for Prime path Coverage are
{ [0,1,3,4,5,7,8,9,10], [0,1,3,4,5,7,9,10], [5,7,8,9,10,4,6,12], [5,7,8,9,10,4,6,11], [5,7,8,9,4,6,11], [4,5,7,8,9,10,4], [5,7,8,9,4,6,12], [5,7,8,9,10,4,5],

[8,9,10,4,5,7,8], [10,4,5,7,8,9,10], [9,10,4,5,7,8,9], [5,7,9,10,4,6,11], [5,7,9,10,4,6,12], [7,8,9,10,4,5,7], [5,7,8,9,4,5], [4,5,7,9,10,4], [5,7,9,4,6,11], [5,7,9,10,4,5], [5,7,9,4,6,12], [0,1,3,4,6,12], [0,1,3,4,6,11], [4,5,7,8,9,4], [7,8,9,4,5,7], [9,4,5,7,8,9], [9,10,4,5,7,9], [10,4,5,7,9,10], [8,9,4,5,7,8], [7,9,10,4,5,7], [5,7,9,4,5], [4,5,7,9,4], [9,4,5,7,9], [7,9,4,5,7], [0,1,2], [6,12,6], [12,6,12], [12,6,11] **}**

The Test Paths for prime path coverage are

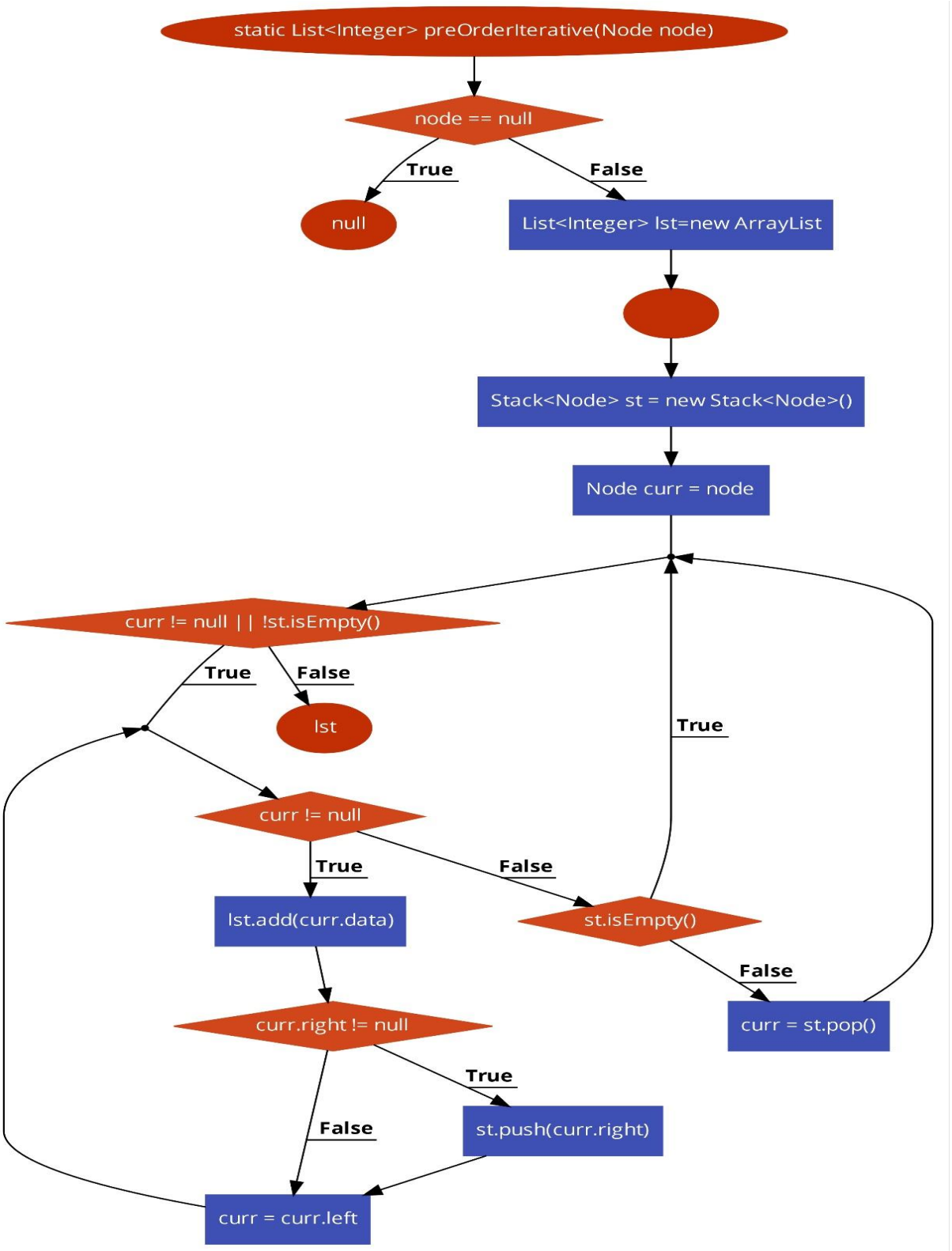
Primepath coverage

Test Paths	Test Requirements that are toured by test paths directly	Test case number
[0,1,2]	[0,1,2]	1
[0,1,3,4,5,7,9,4,6,12,6,11]	[5,7,9,4,6,12], [4,5,7,9,4]	2
[0,1,3,4,5,7,8,9,4,5,7,9,4,6,12,6,12,6,11]	[5,7,8,9,4,5],[4,5,7,9,10,4],[4,5,7,8,9,4], [7,8,9,4,5,7], [9,4,5,7,9],[6,12,6], [12,6,12], [12,6,11]	3
[0,1,3,4,5,7,9,10,4,5,7,9,4,6,12,6,12,6,11]	[0,1,3,4,5,7,9,10], [5,7,9,10,4,5], [9,10,4,5,7,9], [7,9,10,4,5,7]	4
[0,1,3,4,5,7,8,9,10,4,5,7,8,9,10,4,5,7,9,4,5,7,9,4,5,7,9,4,5,7,9,4,6,12,6,12,6,12,6,12,6,12,6,12,6,12,6,11	[9,4,5,7,8,9], [10,4,5,7,9,10], [8,9,4,5,7,8], [4,5,7,8,9,10,4], [5,7,8,9,10,4,5], [8,9,10,4,5,7,8], [10,4,5,7,8,9,10], [9,10,4,5,7,8,9], [0,1,3,4,5,7,8,9,10], [7,8,9,10,4,5,7]	5
[0,1,3,4,5,7,9,4,5,7,9,10,4,5,7,9,4,6,11]	[4,5,7,9,10,4], [5,7,9,4,6,11], [5,7,9,10,4,5], [9,10,4,5,7,9], [7,9,10,4,5,7], [5,7,9,4,5], [4,5,7,9,4], [9,4,5,7,9], [7,9,4,5,7]	Not feasible
[0,1,3,4,5,7,8,9,4,5,7,8,9,4,5,7,8,9,10,4,5,7,8,9,10,4,6,11]	[5,7,8,9,10,4,6,11], [4,5,7,8,9,10,4], [5,7,8,9,10,4,5], [8,9,10,4,5,7,8], [10,4,5,7,8,9,10], [9,10,4,5,7,8,9], [7,8,9,10,4,5,7], [5,7,8,9,4,5], [4,5,7,8,9,4], [7,8,9,4,5,7], [9,4,5,7,8,9], [8,9,4,5,7,8]	Not feasible

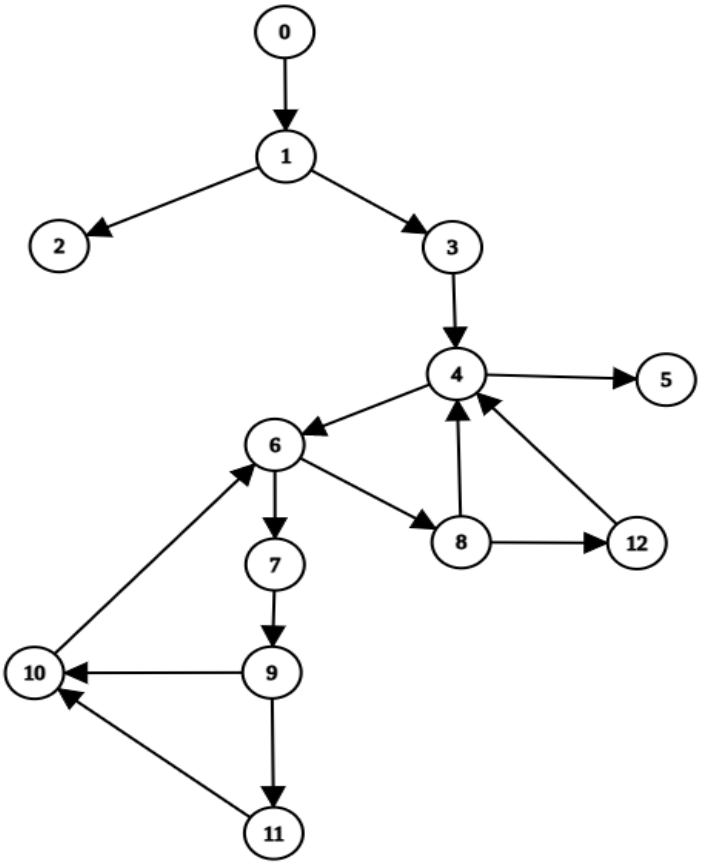
[0,1,3,4,6,12,6,12,6,12,6,11]	[0,1,3,4,6,12], [6,12,6], [12,6,12], [12,6,11]	Not feasible
[0,1,3,4,6,12,6,11]	[0,1,3,4,6,12], [6,12,6], [12,6,11]	Not feasible
[0,1,3,4,5,7,9,4,5,7,9,4,6,12,6,11]	[5,7,9,4,6,12], [5,7,9,4,5], [4,5,7,9,4], [9,4,5,7,9], [7,9,4,5,7], [6,12,6], [12,6,11]	Not feasible
[0,1,3,4,6,11]	[0,1,3,4,6,11]	Not feasible
[0,1,3,4,5,7,9,10,4,5,7,9,10,4,6,11]	[0,1,3,4,5,7,9,10], [5,7,9,10,4,6,11], [4,5,7,9,10,4], [5,7,9,10,4,5], [9,10,4,5,7,9], [10,4,5,7,9,10], [7,9,10,4,5,7]	Not feasible
[0,1,3,4,5,7,8,9,10,4,6,12,6,11]	[0,1,3,4,5,7,8,9,10], [5,7,8,9,10,4,6,12], [4,5,7,8,9,10,4], [6,12,6], [12,6,11]	Not feasible
[0,1,3,4,5,7,8,9,4,6,11]	[5,7,8,9,4,6,11], [4,5,7,8,9,4]	Not feasible
[0,1,3,4,5,7,8,9,4,6,12,6,11]	[5,7,8,9,4,6,12], [4,5,7,8,9,4], [6,12,6], [12,6,11]	Not feasible
[0,1,3,4,5,7,9,10,4,6,11]	[0,1,3,4,5,7,9,10], [5,7,9,10,4,6,11], [4,5,7,9,10,4]	Not feasible
[0,1,3,4,5,7,9,10,4,6,12,6,11]	[4,5,7,9,10,4], [6,12,6], [12,6,11]	Not feasible

The prime path coverage subsumes the Edge Coverage.

PreOrder


static List<Integer> preOrderIterative(Node node)
node == null
True
False
null
List<Integer> lst=new ArrayList
Stack<Node> st = new Stack<Node>()
Node curr = node
curr != null || !st.isEmpty()
True
False
lst
True
curr != null
True
False
lst.add(curr.data)
st.isEmpty()
False
curr = st.pop()
curr.right != null
True
False
st.push(curr.right)
curr = curr.left

This is simplified into a simple graph as follows



We performed the Edge coverage and Prime coverage on this simplified graph.

Edge Coverage:

The Test requirements for Edge Coverage are
{ [0,1], [1,2], [3,4], [1,3], [4,5], [4,6], [6,7], [6,8], [7,9], [9,10], [9,11], [11,10], [10,6], [8,12], [8,4], [12,4] **}**

The prime path coverage subsumes the Edge Coverage.

The Test Paths are

[0,1,3,4,6,7,9,11,10,6,8,4,5]
[0,1,3,4,6,8,4,6,7,9,10,6,8,12,4,5]
[0,1,3,4,5]
[0,1,2]

Prime Path Coverage:

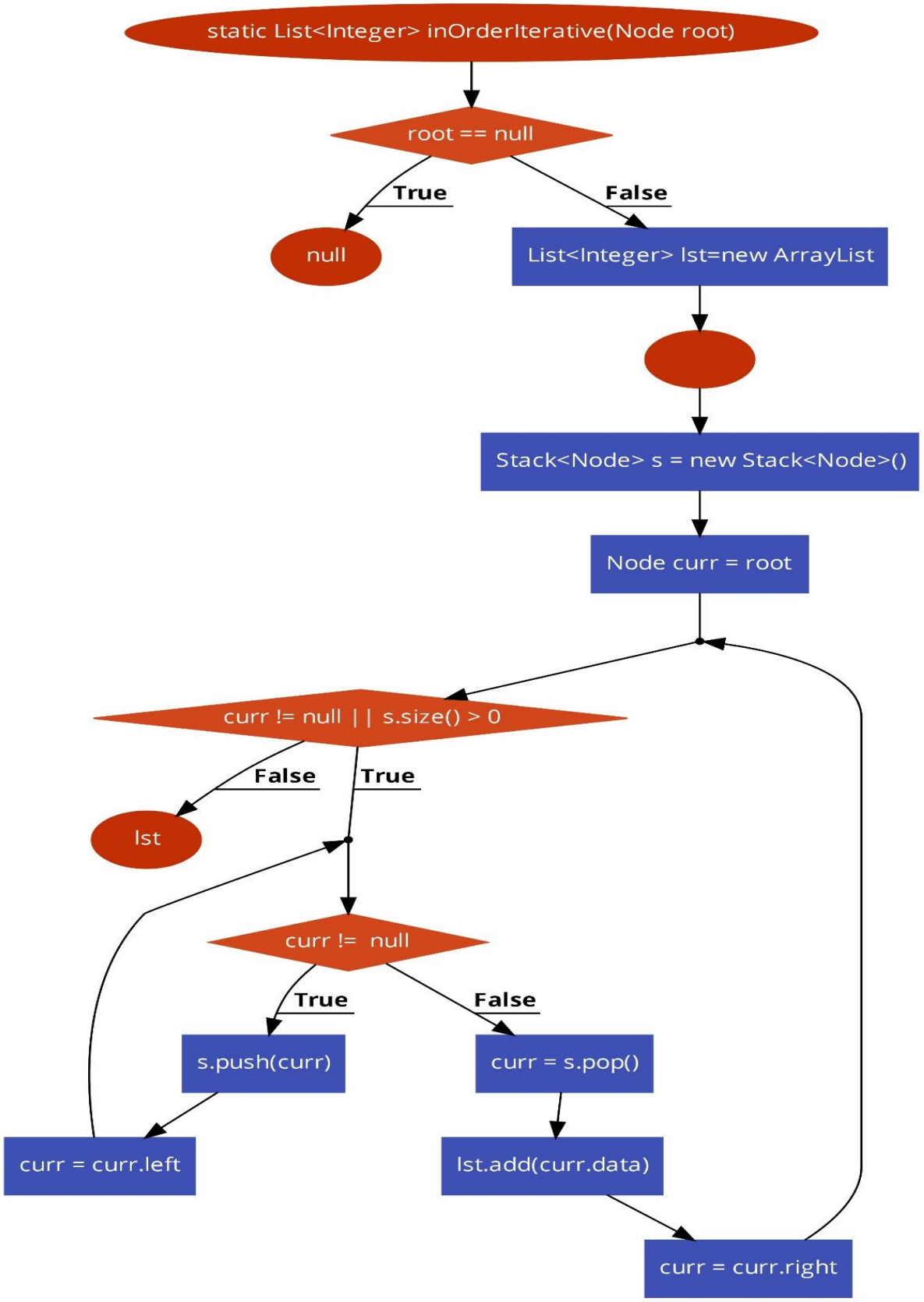
The Test requirements for Prime path Coverage are

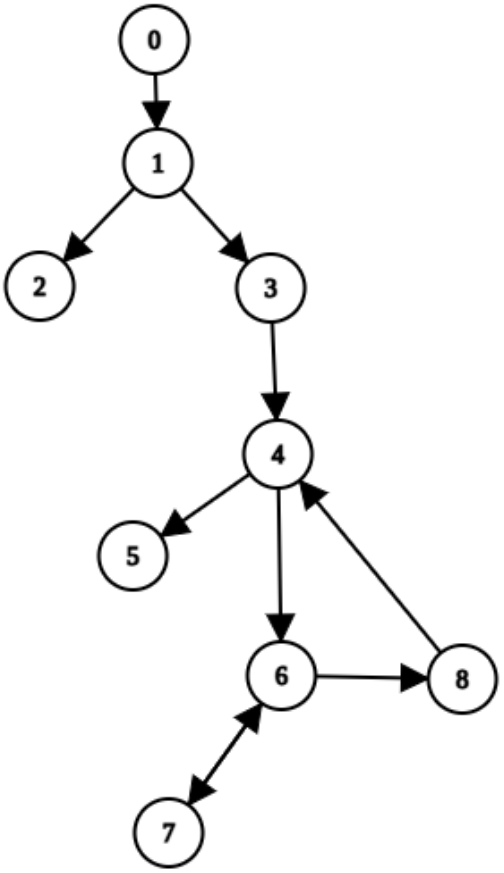
{ [0,1,3,4,6,7,9,11,10], [7,9,11,10,6,8,12,4,5], [7,9,10,6,8,12,4,5], [0,1,3,4,6,7,9,10], [8,12,4,6,7,9,11,10], [7,9,11,10,6,8,4,5], [7,9,10,6,8,4,5], [0,1,3,4,6,8,12], [8,12,4,6,7,9,10], [8,4,6,7,9,11,10], [6,7,9,11,10,6], [7,9,11,10,6,7], [8,4,6,7,9,10], [10,6,7,9,11,10], [9,11,10,6,7,9], [11,10,6,7,9,11], [6,7,9,10,6], [4,6,8,12,4], [6,8,12,4,6], [7,9,10,6,7], [0,1,3,4,5], [9,10,6,7,9], [8,12,4,6,8], [12,4,6,8,12], [10,6,7,9,10], [4,6,8,4], [6,8,4,6], [8,4,6,8], [0,1,2] }

The Test Paths for prime path coverage are

Test Paths	Test Requirements that are toured by test paths directly	Test case number
[0,1,2]	[0,1,2]	1
[0,1,3,4,6,7,9,10,6,8,4,5]	[0,1,3,4,6,7,9,10], [7,9,10,6,8,4,5], [6,7,9,10,6]	2
[0,1,3,4,6,7,9,11,10,6,8,12,4,6,7,9,10,6,8,4,5]	[0,1,3,4,6,7,9,11,10], [6,7,9,11,10,6], [7,9,10,6,8,4,5], [8,12,4,6,7,9,10], [6,7,9,10,6], [6,8,12,4,6]	3
[0,1,3,4,6,7,9,10,6,7,9,10,6,8,4,5]	[0,1,3,4,6,7,9,10], [7,9,10,6,8,4,5], [6,7,9,10,6], [7,9,10,6,7], [9,10,6,7,9], [10,6,7,9,10]	4
[0,1,3,4,6,7,9,11,10,6,7,9,11,10,6,7,9,10,6,8,12,4,6,7,9,10,6,8,12,4,6,7,9,10,6,8,4,5]	[7,9,11,10,6,7], [0,1,3,4,6,7,9,11,10], [6,7,9,10,6], [0,1,3,4,6,7,9,11,10]	5
[0,1,3,4,6,7,9,10,6,8,12,4,5]	[7,9,10,6,8,12,4,5], [0,1,3,4,6,7,9,10], [6,7,9,10,6]	Not feasible
-	[0,1,3,4,5], [0,1,3,4,6,8,12], [6,8,4,6], [4,6,8,4], [8,4,6,8], [8,4,6,7,9,10], [7,9,11,10,6,8,12,4,5], [8,4,6,7,9,11,10], [8,12,4,6,8], [12,4,6,8,12],[4,6,8,12,4],	Not feasible
[0,1,3,4,6,8,12,4,6,7,9,11,10,6,8,4,5]	[8,12,4,6,7,9,11,10], [7,9,11,10,6,8,4,5], [0,1,3,4,6,8,12], [6,7,9,11,10,6], [4,6,8,12,4], [6,8,12,4,6]	Not feasible

InOrder


static List<Integer> inOrderIterative(Node root)
root == null
True
False
null
List<Integer> lst=new ArrayList
Stack<Node> s = new Stack<Node>()
Node curr = root
curr != null || s.size() > 0
False
True
lst
curr != null
True
False
s.push(curr)
curr = s.pop()
curr = curr.left
lst.add(curr.data)
curr = curr.right



We performed the Edge coverage and Prime coverage on this simplified graph.

Edge Coverage:

The Test requirements for Edge Coverage are
{ [0,1], [1,2], [1,3], [3,4], [4,5], [4,6], [6,7], [6,8], [7,6], [8,4] **}**

The Test Paths are
[0,1,3,4,6,8,4,6,7,6,8,4,5]
[0,1,3,4,5]
[0,1,2]

Prime Path Coverage:

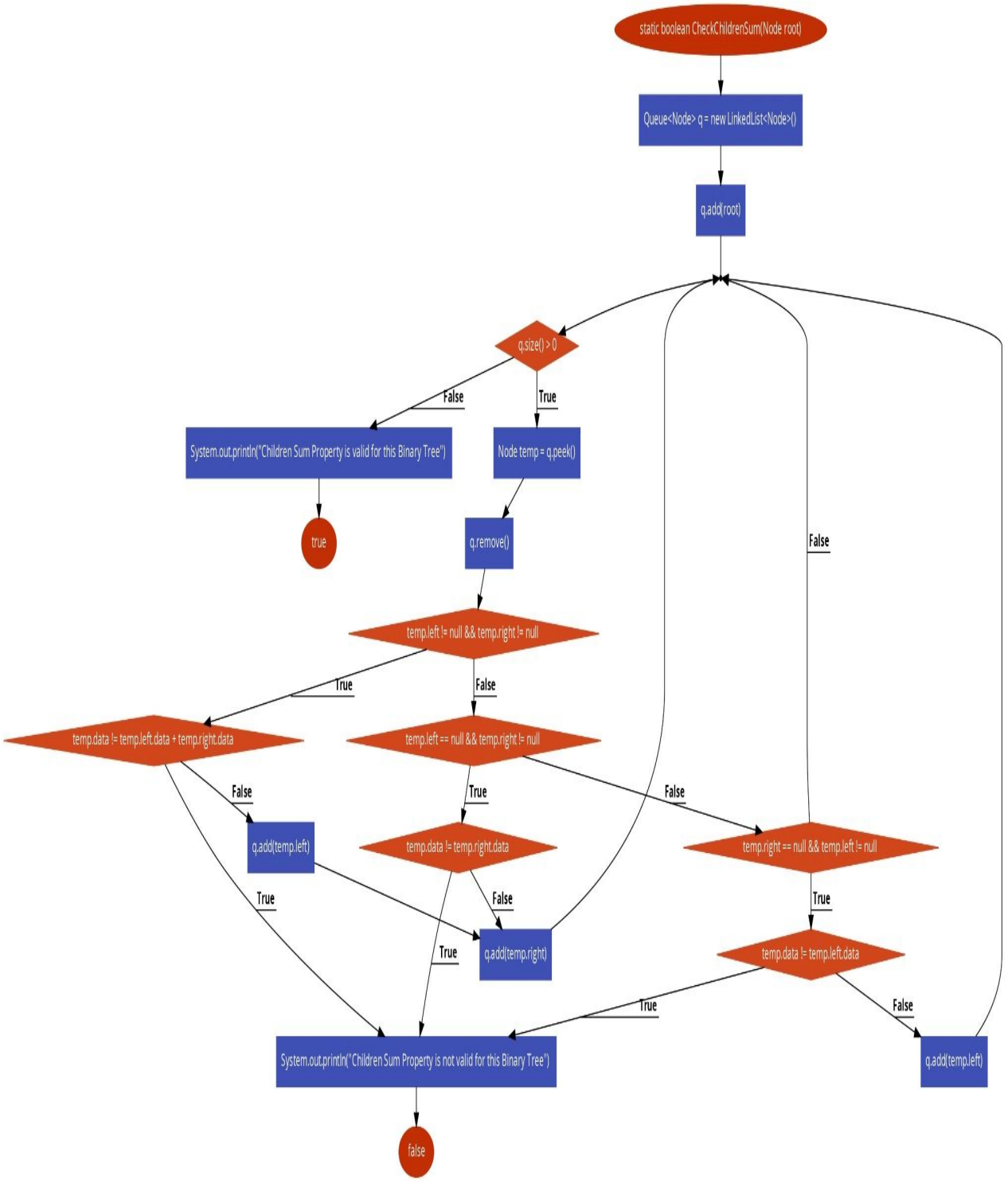
The Test requirements for Prime path Coverage are
{ [0,1,3,4,6,7], [0,1,3,4,6,8], [0,1,3,4,5], [7,6,8,4,5], [4,6,8,4], [8,4,6,7], [8,4,6,8], [6,8,4,6], [0,1,2], [7,6,7], [6,7,6] **}**

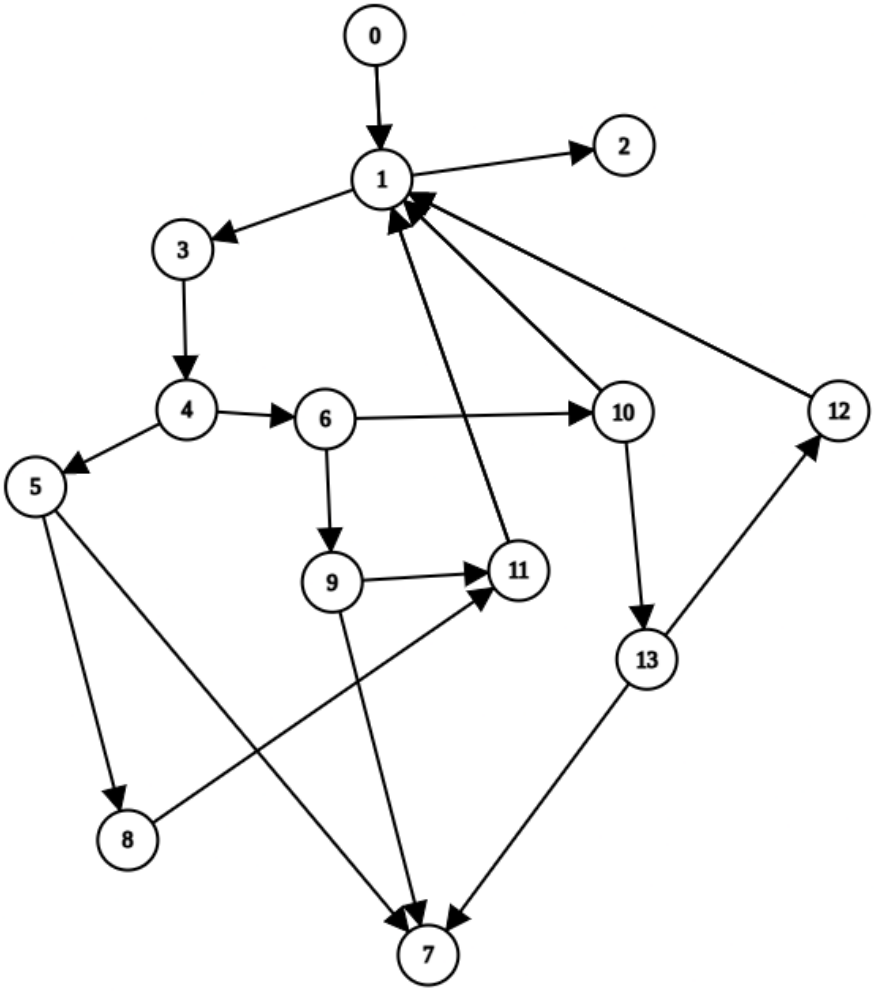
The Test Paths for prime path coverage are

Test Paths	Test Requirements that are toured by test paths directly	Test case number
[0,1,2]	[0,1,2]	1
[0,1,3,4,6,7,6,8,4,5]	[0,1,3,4,6,7], [7,6,8,4,5], [6,7,6]	2
[0,1,3,4,6,7,6,7,6,8,4,6,8,4,5]	[8,4,6,8], [7,6,7], [6,8,4,6], [4,6,8,4]	3
[0,1,3,4,6,7,6,8,4,6,7,6,8,4,5]	[8,4,6,7], [7,6,8,4,5], [6,7,6]	4
-	[0,1,3,4,5], [0,1,3,4,6,8]	Not feasible

The prime path coverage subsumes the Edge Coverage.

CheckChildrenSum





We performed the Edge coverage and Prime coverage on this simplified graph.

Edge Coverage:

The Test requirements for Edge Coverage are

{ [3,4], [4,5], [4,6], [5,7], [5,8], [6,9], [6,10], [9,11], [9,7], [8,11], [10,13], [13,12], [13,7], [0,1], [1,3], [1,2], [11,1], [10,1], [12,1] **}**

The Test Paths are

[0,1,3,4,6,10,13,12,1,2]
[0,1,3,4,5,8,11,1,2]
[0,1,3,4,6,9,11,1,2]
[0,1,3,4,5,7]
[0,1,3,4,6,9,7]
[0,1,3,4,6,10,13,7]
[0,1,3,4,6,10,1,2]

Prime Path Coverage:

The Test requirements for Prime path Coverage are

{ [5,8,11,1,3,4,6,10,13,12], [5,8,11,1,3,4,6,10,13,7], [6,10,13,12,1,3,4,5,8,11], [9,11,1,3,4,6,10,13,12], [6,10,13,12,1,3,4,5,7], [5,8,11,1,3,4,6,9,7], [9,11,1,3,4,6,10,13,7], [10,13,12,1,3,4,6,9,11], [10,13,12,1,3,4,6,9,7], [6,10,1,3,4,5,8,11], [6,10,13,12,1,3,4,6], [6,9,11,1,3,4,5,8], [4,6,10,13,12,1,3,4], [3,4,6,10,13,12,1,2], [3,4,6,10,13,12,1,3], [6,9,11,1,3,4,5,7], [1,3,4,6,10,13,12,1], [0,1,3,4,6,10,13,7], [12,1,3,4,6,10,13,7], [12,1,3,4,6,10,13,12], [0,1,3,4,6,10,13,12], [10,13,12,1,3,4,6,10], [13,12,1,3,4,6,10,13], [6,10,1,3,4,5,7], [6,9,11,1,3,4,6], [9,11,1,3,4,6,9], [8,11,1,3,4,5,8], [8,11,1,3,4,5,7], [3,4,6,9,11,1,2], [3,4,6,9,11,1,3], [3,4,5,8,11,1,3], [3,4,5,8,11,1,2], [4,5,8,11,1,3,4], [5,8,11,1,3,4,5], [4,6,9,11,1,3,4], [11,1,3,4,6,9,11], [11,1,3,4,5,8,11], [10,1,3,4,6,9,11], [10,1,3,4,6,9,7], [0,1,3,4,5,8,11], [1,3,4,5,8,11,1], [1,3,4,6,9,11,1], [0,1,3,4,6,9,11], [0,1,3,4,6,9,7], [6,10,1,3,4,6], [4,6,10,1,3,4], [3,4,6,10,1,3], [3,4,6,10,1,2], [10,1,3,4,6,10], [0,1,3,4,5,7], [1,3,4,6,10,1], [0,1,2] **}**

The Test Paths for prime path coverage are

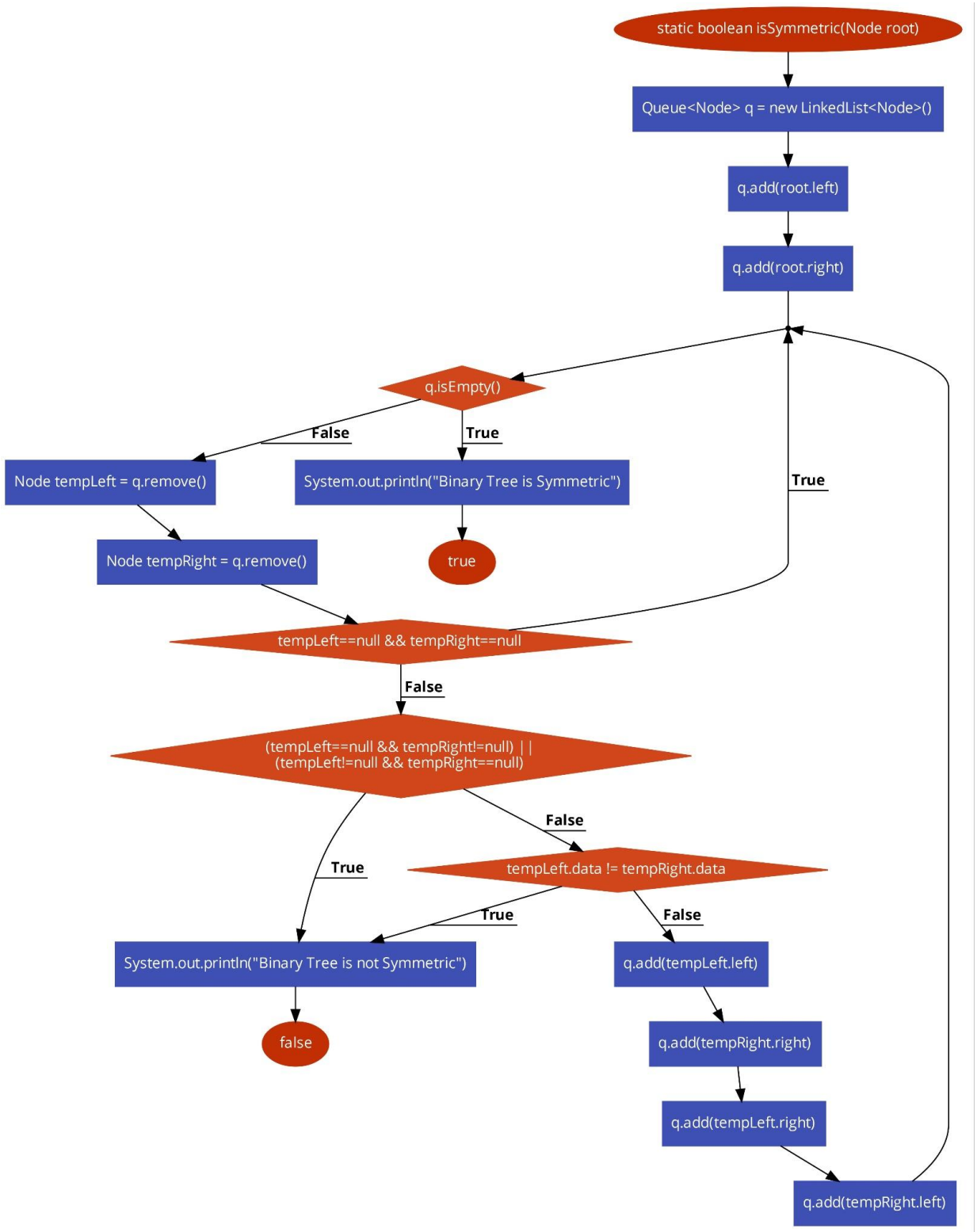
Test Paths	Test Requirements that are toured by test paths directly	Test case number
[0,1,2]	[0,1,2]	1
[0,1,3,4,5,7]	[0,1,3,4,5,7]	2
[0,1,3,4,6,10,1,2]	[3,4,6,10,1,2], [1,3,4,6,10,1]	3
[0,1,3,4,6,9,11,1,3,4,6,9,7]	[6,9,11,1,3,4,6], [9,11,1,3,4,6,9], [3,4,6,9,11,1,3], [4,6,9,11,1,3,4], [1,3,4,6,9,11,1], [0,1,3,4,6,9,11]	4
[0,1,3,4,5,8,11,1,3,4,5,7]	[8,11,1,3,4,5,7], [3,4,5,8,11,1,3], [4,5,8,11,1,3,4], [5,8,11,1,3,4,5], [0,1,3,4,5,8,11], [1,3,4,5,8,11,1]	5
[0,1,3,4,6,9,7]	[0,1,3,4,6,9,7]	6
[0,1,3,4,6,10,13,7]	[0,1,3,4,6,10,13,7]	7

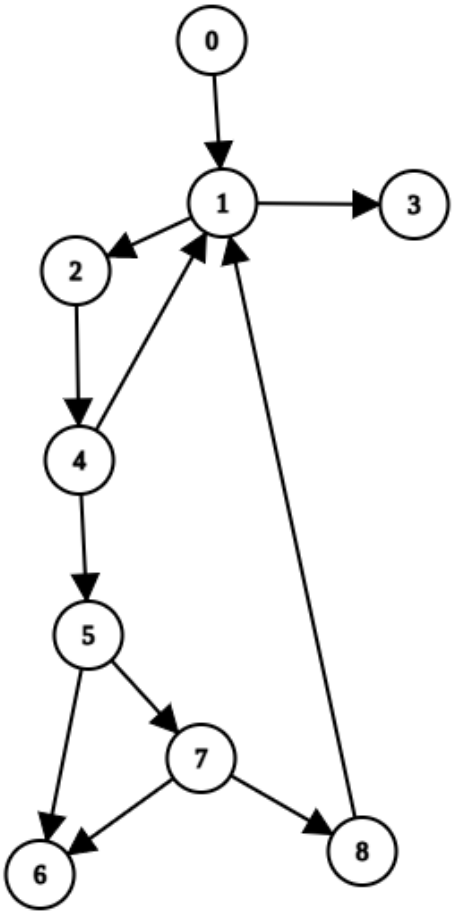
[0,1,3,4,6,10,13,12,1,3,4,6,10,13,7]	[6,10,13,12,1,3,4,6], [4,6,10,13,12,1,3,4], [3,4,6,10,13,12,1,3], [1,3,4,6,10,13,12,1], [12,1,3,4,6,10,13,7], [0,1,3,4,6,10,13,12], [10,13,12,1,3,4,6,10], [13,12,1,3,4,6,10,13]	8
[0,1,3,4,5,8,11,1,3,4,6,9,7]	[5,8,11,1,3,4,6,9,7], [3,4,5,8,11,1,3], [4,5,8,11,1,3,4], [0,1,3,4,5,8,11], [1,3,4,5,8,11,1]	9
[0,1,3,4,6,9,11,1,3,4,6,10,13,7]	[9,11,1,3,4,6,10,13,7], [6,9,11,1,3,4,6], [3,4,6,9,11,1,3], [4,6,9,11,1,3,4], [1,3,4,6,9,11,1], [0,1,3,4,6,9,11]	10
[0,1,3,4,6,10,1,3,4,6,10,13,12,1,3,4,6,10,1,3,4,5,8,11,1,3,4,6,10,13,7]	[5,8,11,1,3,4,6,10,13,7], [6,10,1,3,4,5,8,11], [6,10,13,12,1,3,4,6], [4,6,10,13,12,1,3,4], [3,4,6,10,13,12,1,3], [1,3,4,6,10,13,12,1], [10,13,12,1,3,4,6,10], [3,4,5,8,11,1,3], [4,5,8,11,1,3,4], [1,3,4,5,8,11,1], [6,10,1,3,4,6], [4,6,10,1,3,4], [3,4,6,10,1,3], [10,1,3,4,6,10], [1,3,4,6,10,1]	Not feasible
[0,1,3,4,6,10,1,3,4,6,9,11,1,3,4,5,7]	[6,9,11,1,3,4,5,7], [3,4,6,9,11,1,3], [4,6,9,11,1,3,4], [10,1,3,4,6,9,11], [1,3,4,6,9,11,1], [6,10,1,3,4,6], [4,6,10,1,3,4], [3,4,6,10,1,3], [1,3,4,6,10,1]	Not feasible
[0,1,3,4,6,9,11,1,3,4,5,8,11,1,2]	[6,9,11,1,3,4,5,8], [3,4,6,9,11,1,3], [3,4,5,8,11,1,2], [4,6,9,11,1,3,4], [11,1,3,4,5,8,11], [1,3,4,5,8,11,1], [1,3,4,6,9,11,1], [0,1,3,4,6,9,11]	Not feasible
[0,1,3,4,6,10,13,12,1,3,4,6,10,13,12,1,3,4,6,9,7]	[10,13,12,1,3,4,6,9,7], [6,10,13,12,1,3,4,6], [4,6,10,13,12,1,3,4], [3,4,6,10,13,12,1,3], [1,3,4,6,10,13,12,1], [12,1,3,4,6,10,13,12], [0,1,3,4,6,10,13,12], [10,13,12,1,3,4,6,10], [13,12,1,3,4,6,10,13]	Not feasible
[0,1,3,4,5,8,11,1,3,4,6,9,11,1,2]	[3,4,6,9,11,1,2], [3,4,5,8,11,1,3], [4,5,8,11,1,3,4], [11,1,3,4,6,9,11], [0,1,3,4,5,8,11], [1,3,4,5,8,11,1], [1,3,4,6,9,11,1]	Not feasible
[0,1,3,4,6,10,13,12,1,3,4,6,9,11,1,3,4,6,10,13,12,1,3,4,5,7]	[9,11,1,3,4,6,10,13,12], [6,10,13,12,1,3,4,5,7], [10,13,12,1,3,4,6,9,11], [6,10,13,12,1,3,4,6], [4,6,10,13,12,1,3,4], [3,4,6,10,13,12,1,3], [1,3,4,6,10,13,12,1], [0,1,3,4,6,10,13,12], [6,9,11,1,3,4,6], [3,4,6,9,11,1,3],	Not feasible

	[4,6,9,11,1,3,4], [1,3,4,6,9,11,1]	
[0,1,3,4,6,10,1,3,4,5,7]	[6,10,1,3,4,5,7], [4,6,10,1,3,4], [3,4,6,10,1,3], [1,3,4,6,10,1]	Not feasible
[0,1,3,4,5,8,11,1,3,4,5,8,11,1,2]	[8,11,1,3,4,5,8], [3,4,5,8,11,1,3], [3,4,5,8,11,1,2], [4,5,8,11,1,3,4], [5,8,11,1,3,4,5], [11,1,3,4,5,8,11], [0,1,3,4,5,8,11], [1,3,4,5,8,11,1]	Not feasible
[0,1,3,4,6,10,1,3,4,6,9,7]	[10,1,3,4,6,9,7], [6,10,1,3,4,6], [4,6,10,1,3,4], [3,4,6,10,1,3], [1,3,4,6,10,1]	Not feasible
[0,1,3,4,6,9,11,1,2]	[3,4,6,9,11,1,2], [1,3,4,6,9,11,1], [0,1,3,4,6,9,11]	Not feasible
[0,1,3,4,6,10,13,12,1,2]	[3,4,6,10,13,12,1,2], [1,3,4,6,10,13,12,1], [0,1,3,4,6,10,13,12]	Not feasible
[0,1,3,4,6,10,13,12,1,3,4,5,8,11,1,3,4,6,10,13,12,1,2]	[5,8,11,1,3,4,6,10,13,12], [6,10,13,12,1,3,4,5,8,11], [4,6,10,13,12,1,3,4], [3,4,6,10,13,12,1,2], [3,4,6,10,13,12,1,3], [1,3,4,6,10,13,12,1], [0,1,3,4,6,10,13,12], [3,4,5,8,11,1,3], [4,5,8,11,1,3,4], [1,3,4,5,8,11,1]	Not feasible

The prime path coverage subsumes the Edge Coverage.

isSymmetric





We performed the Edge coverage and Prime coverage on this simplified graph.

Edge Coverage:

The Test requirements for Edge Coverage are
{ [0,1], [1,2], [1,3], [2,4], [4,1], [4,5], [5,6], [5,7], [7,6], [7,8], [8,1] **}**

The Test Paths are

[0,1,2,4,5,7,8,1,3]
[0,1,2,4,5,7,6]
[0,1,2,4,5,6]
[0,1,2,4,1,3]

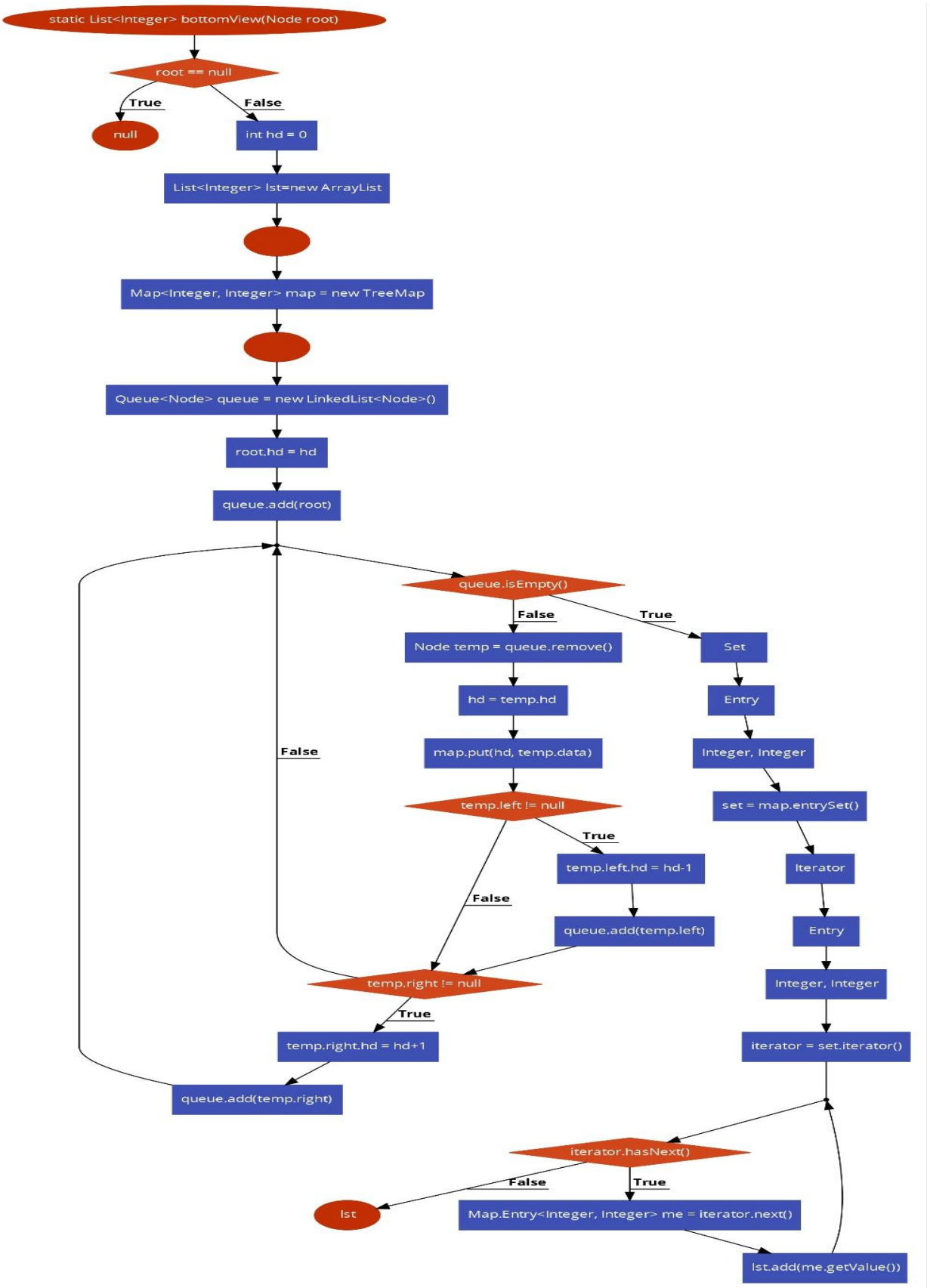
Prime Path Coverage:

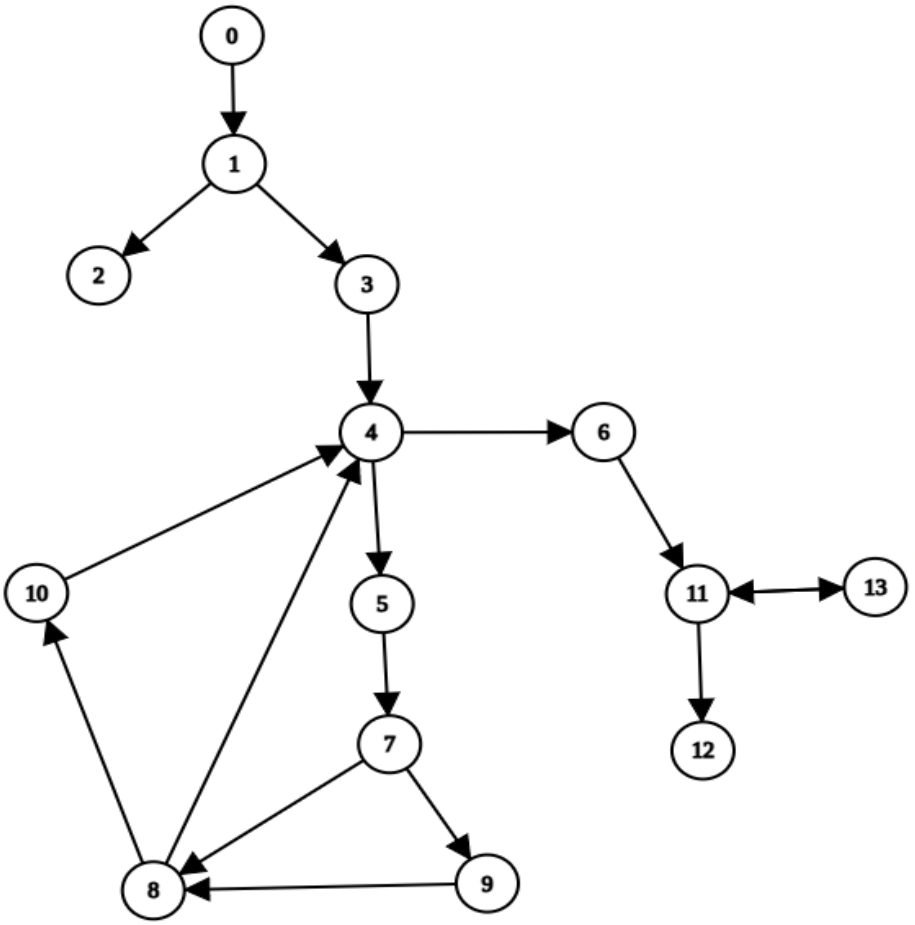
The Test requirements for Prime path Coverage are
{ [2,4,5,7,8,1,2], [2,4,5,7,8,1,3], [1,2,4,5,7,8,1], [0,1,2,4,5,7,6], [0,1,2,4,5,7,8], [4,5,7,8,1,2,4], [8,1,2,4,5,7,6], [8,1,2,4,5,7,8], [7,8,1,2,4,5,7], [5,7,8,1,2,4,5], [7,8,1,2,4,5,6], [0,1,2,4,5,6], [2,4,1,3], [2,4,1,2], [1,2,4,1], [4,1,2,4], [0,1,3] **}**

The Test Paths for prime path coverage are

Test Paths	Test Requirements that are toured by test paths directly	Test case number
[0,1,3]	[0,1,3]	2
[0,1,2,4,5,6]	[0,1,2,4,5,6]	3
[0,1,2,4,5,7,6]	[0,1,2,4,5,7,6]	4
[0,1,2,4,5,7,8,1,2,4,5,6]	[2,4,5,7,8,1,2], [1,2,4,5,7,8,1], [0,1,2,4,5,7,8], [4,5,7,8,1,2,4], [5,7,8,1,2,4,5], [7,8,1,2,4,5,6]	5
[0,1,2,4,5,7,8,1,2,4,5,7,6]	[2,4,5,7,8,1,2], [1,2,4,5,7,8,1], [0,1,2,4,5,7,8], [4,5,7,8,1,2,4], [8,1,2,4,5,7,6], [7,8,1,2,4,5,7], [5,7,8,1,2,4,5]	6
[0,1,2,4,5,7,8,1,2,4,1,2,4,5,7,6]	[1,2,4,5,7,8,1], [0,1,2,4,5,7,8],	7
[0,1,2,4,5,7,8,1,2,4,1,2,4,5,7,8,1,3]	[2,4,5,7,8,1,3],	8
[0,1,2,4,5,7,8,1,2,4,1,2,4,5,6]	[2,4,1,2],[1,2,4,1], [4,1,2,4],[4,5,7,8,1,2,4],	9
[0,1,2,4,1,3]	[2,4,1,3],[8,1,2,4,5,7,8],	Not feasible

Bottom View


static List<Integer> bottomView(Node root)
root == null
True
False
null
int hd = 0
List<Integer> lst=new ArrayList
Map<Integer, Integer> map = new TreeMap
Queue<Node> queue = new LinkedList<Node>()
root.hd = hd
queue.add(root)
queue.isEmpty()
False
True
Node temp = queue.remove()
hd = temp.hd
map.put(hd, temp.data)
temp.left != null
True
temp.left.hd = hd-1
queue.add(temp.left)
False
temp.right != null
True
temp.right.hd = hd+1
queue.add(temp.right)
False
Set
Entry
Integer, Integer
set = map.entrySet()
Iterator
Entry
Integer, Integer
iterator = set.iterator()
iterator.hasNext()
False
True
lst
Map.Entry<Integer, Integer> me = iterator.next()
lst.add(me.getValue())



Edge Coverage:

The Test requirements for Edge Coverage are

{ [0,1], [1,2], [1,3], [3,4], [4,5], [4,6], [5,7], [7,8], [7,9], [9,8], [8,10], [10,4], [6,11], [11,12], [11,13], [13,11], [8,4] **}**

The Test Paths are

[0,1,3,4,6,11,13,11,13,11,12]
[0,1,3,4,5,7,9,8,4,6,11,12]
[0,1,3,4,5,7,8,10,4,6,11,12]
[0,1,2]

The prime path coverage subsumes the Edge Coverage.

Prime Path Coverage:

The Test requirements for Prime path Coverage are

{ [0,1,3,4,5,7,9,8,10], [5,7,9,8,10,4,6,11,12], [5,7,9,8,10,4,6,11,13], [5,7,8,10,4,6,11,12], [5,7,8,10,4,6,11,13], [0,1,3,4,5,7,8,10], [5,7,9,8,4,6,11,13], [5,7,9,8,4,6,11,12], [4,5,7,9,8,10,4], [5,7,8,4,6,11,12], [5,7,9,8,10,4,5], [5,7,8,4,6,11,13], [0,1,3,4,6,11,12], [0,1,3,4,6,11,13], [9,8,10,4,5,7,9], [10,4,5,7,9,8,10], [8,10,4,5,7,9,8], [7,9,8,10,4,5,7], [5,7,8,10,4,5], [4,5,7,9,8,4], [7,8,10,4,5,7], [5,7,9,8,4,5], [4,5,7,8,10,4], [8,10,4,5,7,8], [8,4,5,7,9,8], [10,4,5,7,8,10], [9,8,4,5,7,9], [7,9,8,4,5,7], [5,7,8,4,5], [4,5,7,8,4], [7,8,4,5,7], [8,4,5,7,8], [0,1,2], [13,11,12], [11,13,11], [13,11,13]

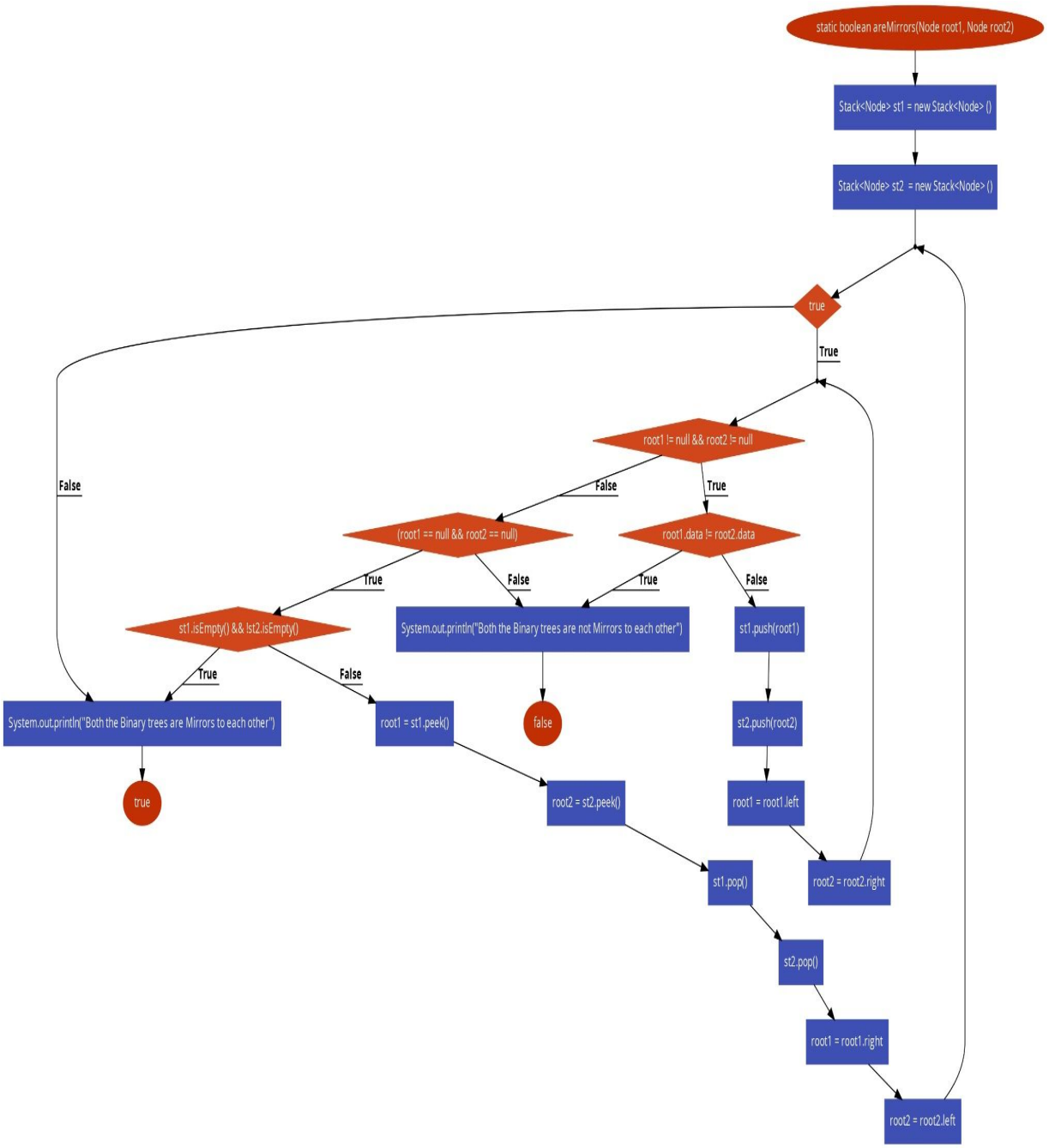
}

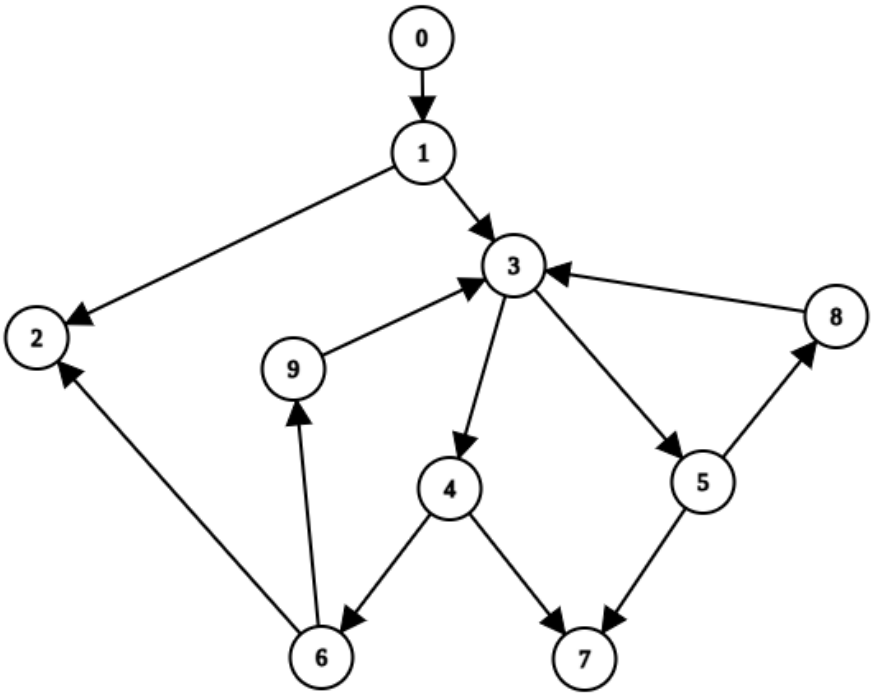
The Test Paths for prime path coverage are

Test Paths	Test Requirements that are toured by test paths directly	Test case number
[0,1,2]	[0,1,2]	1
[0,1,3,4,5,7,8,4,6,11,13,11,12]	[5,7,8,4,6,11,12], [4,5,7,8,4], [13,11,12], [11,13,11]	2
[0,1,3,4,5,7,9,8,10,4,5,7,8,4,5,7,8,4,6,11,13,11,13,11,12]	[0,1,3,4,5,7,9,8,10], [5,7,9,8,10,4,6,11,12], [4,5,7,9,8,10,4]	3
[0,1,3,4,5,7,9,8,4,5,7,8,4,6,11,13,11,13,11,12]	[5,7,8,4,6,11,12], [4,5,7,9,8,4], [5,7,9,8,4,5], [7,9,8,4,5,7], [4,5,7,8,4], [8,4,5,7,8]	4
[0,1,3,4,5,7,8,10,4,5,7,8,10,4,5,7,8,4,6,11,13,11,13,11,13,11,12]	[0,1,3,4,5,7,8,10], [5,7,8,4,6,11,13], [5,7,8,10,4,5], [7,8,10,4,5,7], [4,5,7,8,10,4], [8,10,4,5,7,8], [10,4,5,7,8,10], [4,5,7,8,4], [13,11,12], [11,13,11]	5
[0,1,3,4,5,7,9,8,4,5,7,9,8,4,6,11,13,11,13,11,13,11,12]	[5,7,9,8,4,6,11,12], [4,5,7,9,8,4], [5,7,9,8,4,5], [8,4,5,7,9,8], [9,8,4,5,7,9], [7,9,8,4,5,7]	6
[0,1,3,4,6,11 ,13,11,12]	[0,1,3,4,6,11,13],	Not feasible
[0,1,3,4,5,7,8,4,5,7,8,4,6,11,12]	[5,7,8,4,6,11,12], [5,7,8,4,5], [4,5,7,8,4], [7,8,4,5,7], [8,4,5,7,8]	Not feasible

[0,1,3,4,5,7,8,10,4,6,11,12]	[5,7,8,10,4,6,11,12], [0,1,3,4,5,7,8,10], [4,5,7,8,10,4]	Not feasible
[0,1,3,4,5,7,8,4,5,7,9,8,10,4,5,7,9,8,4,6,11,13,11,13,11,12]	[5,7,9,8,4,6,11,13], [4,5,7,9,8,10,4], [5,7,9,8,10,4,5], [9,8,10,4,5,7,9], [8,10,4,5,7,9,8], [7,9,8,10,4,5,7], [4,5,7,9,8,4], [8,4,5,7,9,8], [5,7,8,4,5], [4,5,7,8,4], [7,8,4,5,7], [13,11,12], [11,13,11], [13,11,13]	Not feasible
[0,1,3,4,6,11,12]	[0,1,3,4,6,11,12]	Not feasible
[0,1,3,4,5,7,8,10,4,5,7,9,8,10,4,6,11,12]	[5,7,9,8,10,4,6,11,12], [0,1,3,4,5,7,8,10], [4,5,7,9,8,10,4], [10,4,5,7,9,8,10], [8,10,4,5,7,9,8], [5,7,8,10,4,5], [7,8,10,4,5,7], [4,5,7,8,10,4]	Not feasible
[0,1,3,4,5,7,8,10,4,6,11,13,11,12]	[5,7,8,10,4,6,11,13], [0,1,3,4,5,7,8,10], [4,5,7,8,10,4], [13,11,12], [11,13,11]	Not feasible
[0,1,3,4,5,7,9,8,4,6,11,12]	[5,7,9,8,4,6,11,12], [4,5,7,9,8,4]	Not feasible
[0,1,3,4,5,7,9,8,10,4,6,11,13,11,12]	[0,1,3,4,5,7,9,8,10], [5,7,9,8,10,4,6,11,13], [4,5,7,9,8,10,4], [13,11,12], [11,13,11]	Not feasible

areMirrors


static boolean areMirrors(Node root1, Node root2)
Stack<Node> st1 = new Stack<Node> ()
Stack<Node> st2 = new Stack<Node> ()
true
True
root1 != null && root2 != null
False
True
(root1 == null && root2 == null)
root1.data != root2.data
True
False
True
False
st1.isEmpty() && !st2.isEmpty()
System.out.println("Both the Binary trees are not Mirrors to each other")
st1.push(root1)
True
False
System.out.println("Both the Binary trees are Mirrors to each other")
root1 = st1.peek()
false
st2.push(root2)
true
root2 = st2.peek()
root1 = root1.left
st1.pop()
root2 = root2.right
st2.pop()
root1 = root1.right
root2 = root2.left



Edge Coverage:

The Test requirements for Edge Coverage are

{ [0,1], [1,2], [1,3], [3,4], [3,5], [4,6], [4,7], [5,7], [5,8], [8,3], [6,2], [6,9], [9,3] **}**

The Test Paths are

[0,1,3,4,6,9,3,5,8,3,5,7]
[0,1,3,4,6,2]
[0,1,2]
[0,1,3,4,7]

The prime path coverage subsumes the Edge Coverage.

Prime Path Coverage:

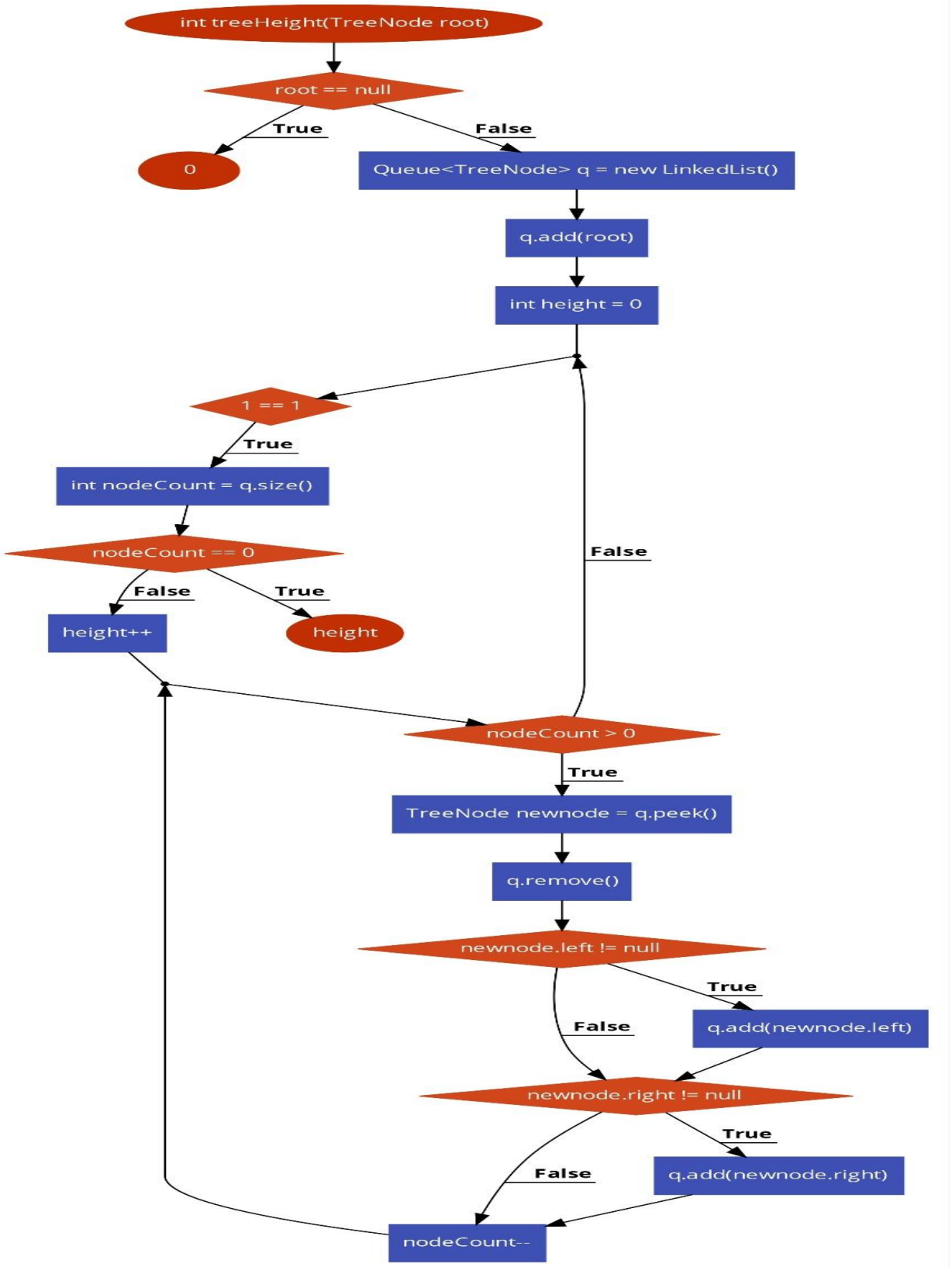
The Test requirements for Prime path Coverage are

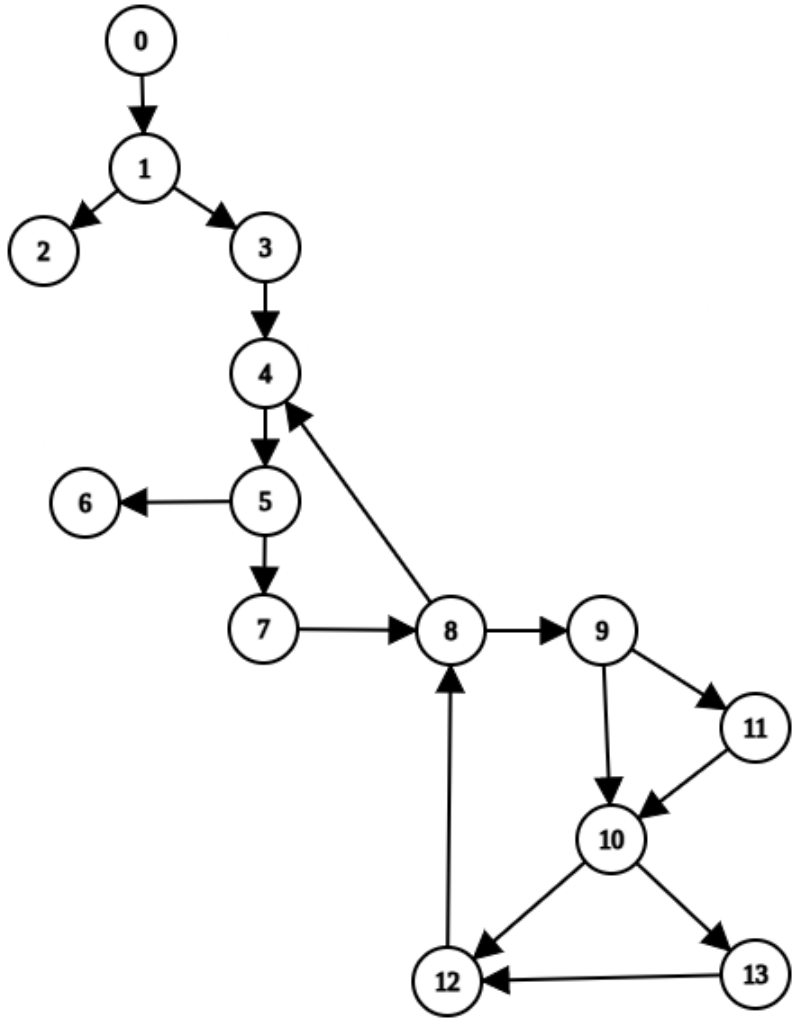
{ [4,6,9,3,5,7], [0,1,3,4,6,9], [0,1,3,4,6,2], [4,6,9,3,5,8], [5,8,3,4,6,9], [5,8,3,4,6,2], [3,4,6,9,3], [4,6,9,3,4], [0,1,3,4,7], [0,1,3,5,7], [0,1,3,5,8], [9,3,4,6,2], [9,3,4,6,9], [6,9,3,4,7], [5,8,3,4,7], [6,9,3,4,6], [3,5,8,3], [5,8,3,5], [8,3,5,7], [8,3,5,8], [0,1,2] **}**

The Test Paths for prime path coverage are

Test Paths	Test Requirements that are toured by test paths directly	Test case number
[0,1,3,4,6,2]	[0,1,3,4,6,2]	1
[0,1,3,4,7]	[0,1,3,4,7]	2
[0,1,3,5,7]	[0,1,3,5,7]	3
[0,1,3,5,8,3,4,6,9,3,4,7]	[4,6,9,3,5,8], [5,8,3,4,6,9], [3,4,6,9,3], [0,1,3,5,8], [5,8,3,4,7], [3,5,8,3]	4
[0,1,3,5,8,3,4,6,2]	[0,1,3,5,8], [3,5,8,3]	5
[0,1,3,4,6,9,3,4,6,9,3,4,7]	[0,1,3,4,6,9], [3,4,6,9,3], [4,6,9,3,4], [9,3,4,6,9], [6,9,3,4,7], [6,9,3,4,6]	Not feasible
[0,1,3,5,8,3,5,8,3,5,8,3,4,7]	[0,1,3,5,8], [5,8,3,4,7], [3,5,8,3], [5,8,3,5], [8,3,5,8]	Not feasible
[0,1,2]	[0,1,2]	Not feasible
[0,1,3,5,8,3,5,7]	[0,1,3,5,8], [3,5,8,3], [5,8,3,5], [8,3,5,7]	Not feasible
[0,1,3,5,8,3,4,6,2]	[5,8,3,4,6,2], [0,1,3,5,8], [3,5,8,3]	Not feasible
[0,1,3,4,6,9,3,5,7]	[4,6,9,3,5,7], [0,1,3,4,6,9], [3,4,6,9,3]	Not feasible
[0,1,3,4,6,9,3,4,6,2]	[0,1,3,4,6,9], [3,4,6,9,3], [4,6,9,3,4], [9,3,4,6,2], [6,9,3,4,6]	Not feasible

Height


int treeHeight(TreeNode root)
root == null
True
False
0
Queue<TreeNode> q = new LinkedList()
q.add(root)
int height = 0
1 == 1
True
int nodeCount = q.size()
nodeCount == 0
False
True
height++
height
False
nodeCount > 0
True
TreeNode newnode = q.peek()
q.remove()
newnode.left != null
True
False
q.add(newnode.left)
newnode.right != null
True
False
q.add(newnode.right)
nodeCount--



Edge Coverage:

The Test requirements for Edge Coverage are

{ [0,1], [1,2], [1,3], [3,4], [4,5], [5,6], [5,7], [7,8], [8,4], [8,9], [9,11], [9,10], [11,10], [10,13], [13,12], [10,12], [12,8] **}**

The Test Paths are

[0,1,3,4,5,7,8,9,10,13,12,8,9,11,10,12,8,4,5,6]
[0,1,3,4,5,7,8,4,5,6]
[0,1,3,4,5,6]
[0,1,2]

The prime path coverage subsumes the Edge Coverage.

Prime Path Coverage:

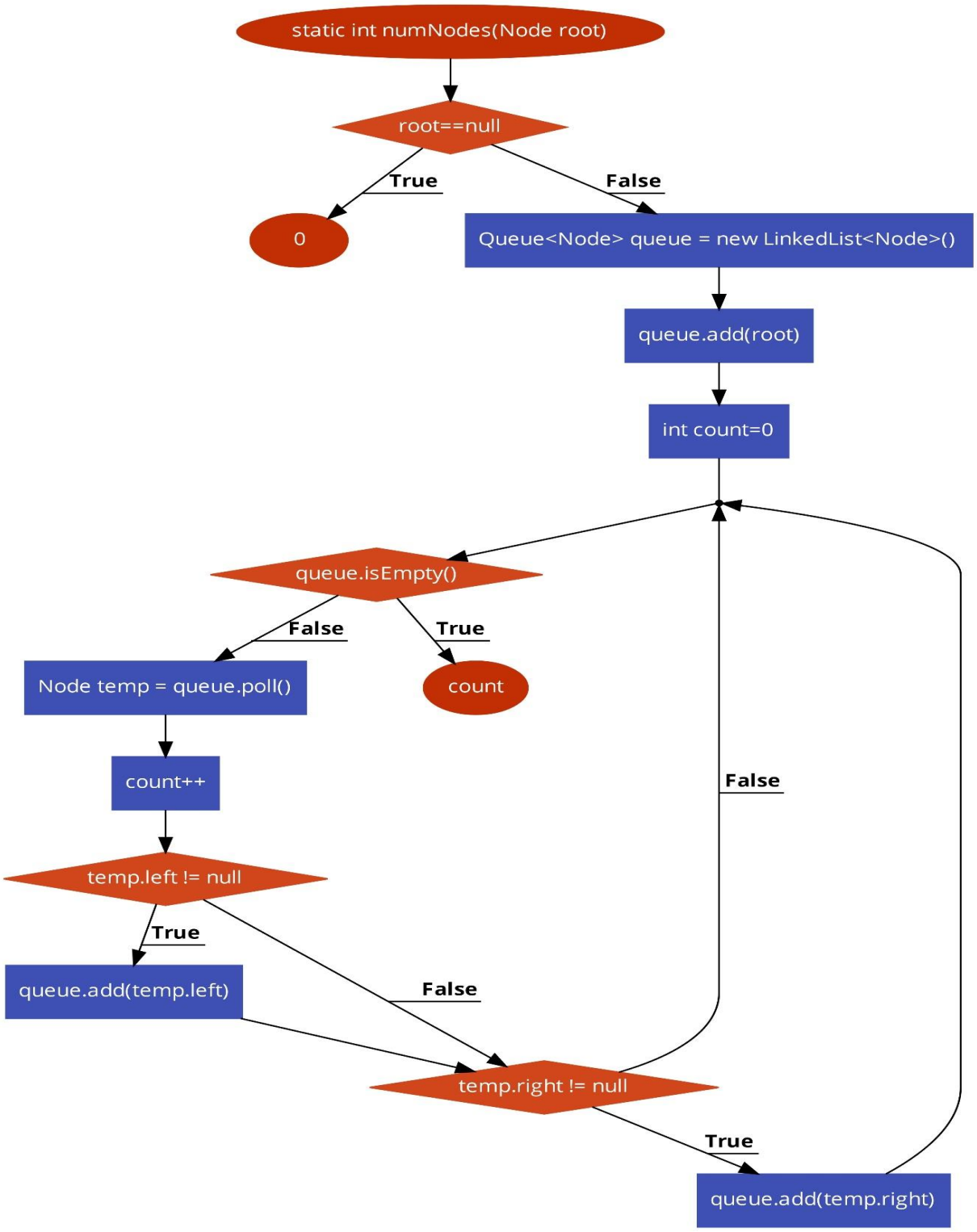
The Test requirements for Prime path Coverage are
{ [0,1,3,4,5,7,8,9,11,10,13,12], [0,1,3,4,5,7,8,9,11,10,12], [0,1,3,4,5,7,8,9,10,13,12], [0,1,3,4,5,7,8,9,10,12], [9,11,10,13,12,8,4,5,6], [9,11,10,13,12,8,4,5,7], [9,10,13,12,8,4,5,7], [9,10,13,12,8,4,5,6], [9,11,10,12,8,4,5,7], [9,11,10,12,8,4,5,6], [8,9,11,10,13,12,8], [9,11,10,13,12,8,9], [10,13,12,8,9,11,10], [12,8,9,11,10,13,12], [13,12,8,9,11,10,13], [9,10,12,8,4,5,6], [9,10,12,8,4,5,7], [11,10,13,12,8,9,11], [8,9,11,10,12,8], [8,9,10,13,12,8], [9,11,10,12,8,9], [0,1,3,4,5,6], [9,10,13,12,8,9], [13,12,8,9,10,13], [12,8,9,11,10,12], [12,8,9,10,13,12], [10,12,8,9,11,10], [11,10,12,8,9,11], [10,13,12,8,9,10], [8,4,5,7,8], [7,8,4,5,7], [8,9,10,12,8], [7,8,4,5,6], [4,5,7,8,4], [5,7,8,4,5], [10,12,8,9,10], [12,8,9,10,12], [9,10,12,8,9], [0,1,2] **}**

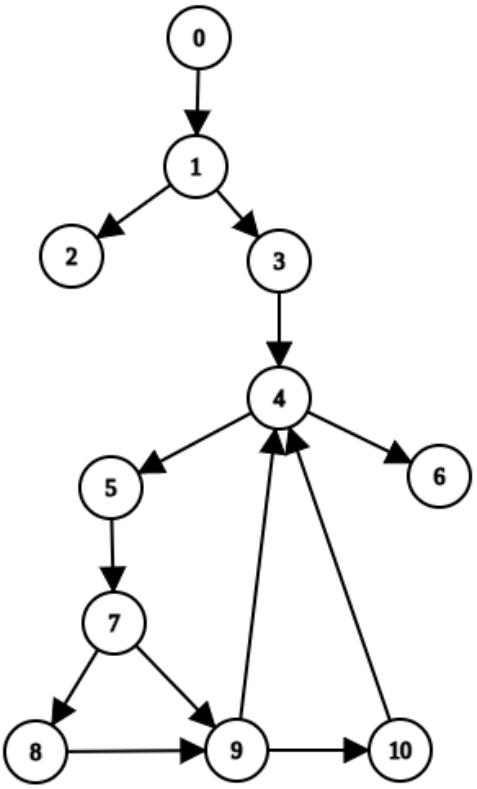
The Test Paths for prime path coverage are

Test Paths	Test Requirements that are toured by test paths directly	Test case number
[0,1,2]	[0,1,2]	1
[0,1,3,4,5,7,8,9,10,12,8,4,5,6]	[8,4,5,7,8], [7,8,4,5,7], [7,8,4,5,6], [4,5,7,8,4], [5,7,8,4,5]	2
[0,1,3,4,5,7,8,9,11,10,12,8,4,5,7,8,9,10,12,8,4,5,6]	[0,1,3,4,5,7,8,9,11,10,12], [9,11,10,12,8,4,5,7], [8,9,11,10,12,8], [8,4,5,7,8], [7,8,4,5,6], [4,5,7,8,4], [5,7,8,4,5]	3
[0,1,3,4,5,7,8,9,10,13,12,8,45,7,8,9,10,12,8,4,5,6]	[0,1,3,4,5,7,8,9,10,13,12], [9,11,10,13,12,8,4,5,6], [8,9,11,10,13,12,8], [10,13,12,8,9,11,10], [12,8,9,11,10,13,12], [13,12,8,9,11,10,13], [8,9,10,13,12,8], [9,10,13,12,8,9]	4
[0,1,3,4,5,7,8,9,11,10,13,12,8,4,5,7,8,9,10,12,8,9,10,12,8,4,5,6]	[0,1,3,4,5,7,8,9,11,10,13,12], [9,11,10,13,12,8,4,5,7], [8,9,11,10,13,12,8], [8,4,5,7,8], [7,8,4,5,6], [4,5,7,8,4], [5,7,8,4,5]	5
[0,1,3,4,5,6]	[0,1,3,4,5,6]	Not feasible

[0,1,3,4,5,7,8,9,10,13,12,8,9,10,13,12,8,9,10,13,12,8,9,10,12,8,4,5,7,8,4,5,6]	[0,1,3,4,5,7,8,9,10,13,12], [9,10,12,8,4,5,7], [8,9,10,13,12,8], [9,10,13,12,8,9], [13,12,8,9,10,13], [12,8,9,10,13,12], [10,13,12,8,9,10], [8,4,5,7,8], [8,9,10,12,8], [7,8,4,5,6], [4,5,7,8,4], [5,7,8,4,5], [12,8,9,10,12]	Not feasible
[0,1,3,4,5,7,8,9,11,10,13,12,8,4,5,6]	[0,1,3,4,5,7,8,9,11,10,13,12], [9,11,10,13,12,8,4,5,6], [8,9,11,10,13,12,8]	Not feasible
[0,1,3,4,5,7,8,9,10,13,12,8,4,5,7,8,4,5,6]	[0,1,3,4,5,7,8,9,10,13,12], [9,10,13,12,8,4,5,7], [8,9,10,13,12,8], [8,4,5,7,8], [7,8,4,5,6], [4,5,7,8,4], [5,7,8,4,5]	Not feasible
[0,1,3,4,5,7,8,9,10,13,12,8,4,5,6]	[0,1,3,4,5,7,8,9,10,13,12], [9,10,13,12,8,4,5,6], [8,9,10,13,12,8]	Not feasible
[0,1,3,4,5,7,8,9,10,12,8,9,10,12,8,9,10,12,8,9,11,10,13,12,8,9,11,10,12,8,4,5,6]	[0,1,3,4,5,7,8,9,10,12], [9,11,10,12,8,4,5,6], [8,9,11,10,13,12,8], [9,11,10,13,12,8,9], [10,13,12,8,9,11,10], [12,8,9,11,10,13,12], [11,10,13,12,8,9,11], [8,9,11,10,12,8], [12,8,9,11,10,12], [10,12,8,9,11,10], [8,9,10,12,8], [10,12,8,9,10], [12,8,9,10,12], [9,10,12,8,9]	Not feasible
[0,1,3,4,5,7,8,9,10,12,8,4,5,6]	[0,1,3,4,5,7,8,9,10,12], [9,10,12,8,4,5,6], [8,9,10,12,8]	Not feasible
[0,1,3,4,5,7,8,9,11,10,12,8,9,11,10,12,8,9,10,12,8,4,5,6]	[0,1,3,4,5,7,8,9,11,10,12], [9,10,12,8,4,5,6], [8,9,11,10,12,8], [9,11,10,12,8,9], [12,8,9,11,10,12], [10,12,8,9,11,10], [11,10,12,8,9,11], [8,9,10,12,8], [10,12,8,9,10], [12,8,9,10,12]	Not feasible

NumNodes


static int numNodes(Node root)
root==null
True
False
0
Queue<Node> queue = new LinkedList<Node>()
queue.add(root)
int count=0
queue.isEmpty()
False
True
Node temp = queue.poll()
count
count++
False
temp.left != null
True
queue.add(temp.left)
False
temp.right != null
True
queue.add(temp.right)



Edge Coverage:

The Test requirements for Edge Coverage are

{ [0,1], [1,2], [1,3], [3,4], [4,5], [4,6], [5,7], [7,8], [7,9], [8,9], [9,4], [9,10], [10,4] **}**

The Test Paths are

[0,1,3,4,5,7,9,4,6]
[0,1,3,4,5,7,8,9,10,4,6]
[0,1,2]

The prime path coverage subsumes the Edge Coverage.

Prime Path Coverage:

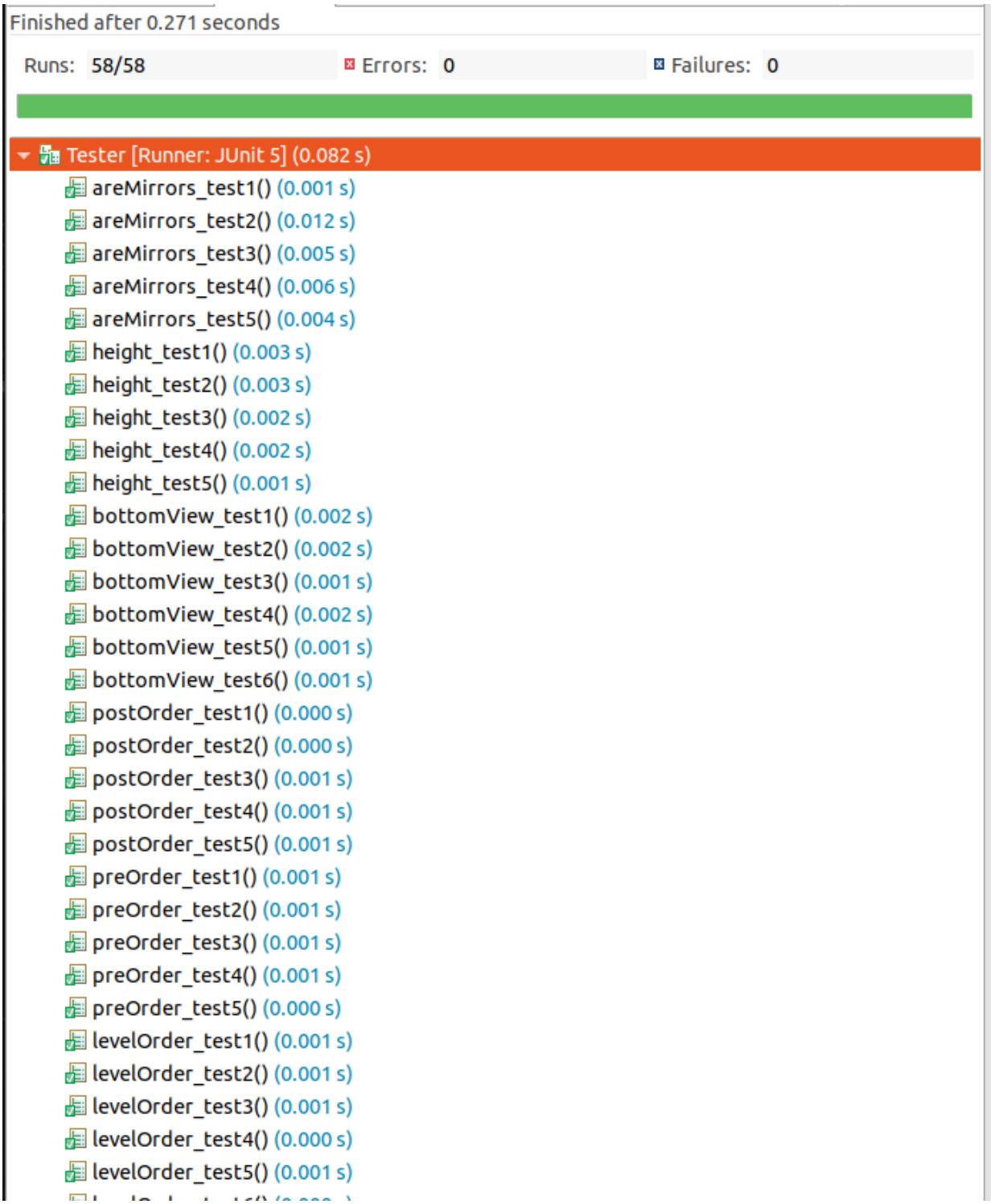
The Test requirements for Prime path Coverage are

{ [0,1,3,4,5,7,8,9,10], [0,1,3,4,5,7,9,10], [4,5,7,8,9,10,4], [5,7,8,9,10,4,5], [9,10,4,5,7,8,9], [10,4,5,7,8,9,10], [8,9,10,4,5,7,8], [5,7,8,9,10,4,6], [7,8,9,10,4,5,7], [5,7,8,9,4,5], [4,5,7,9,10,4], [5,7,8,9,4,6], [5,7,9,10,4,5], [4,5,7,8,9,4], [5,7,9,10,4,6], [9,4,5,7,8,9], [9,10,4,5,7,9], [10,4,5,7,9,10], [7,8,9,4,5,7], [8,9,4,5,7,8], [7,9,10,4,5,7], [5,7,9,4,5], [5,7,9,4,6], [4,5,7,9,4], [0,1,3,4,6], [9,4,5,7,9], [7,9,4,5,7], [0,1,2] **}**

The Test Paths for prime path coverage are

Test Paths	Test Requirements that are toured by test paths directly	Test case number
[0,1,2]	[0,1,2]	1
[0,1,3,4,5,7,8,9,4,5,7,9,4,6]	[5,7,8,9,4,6], [4,5,7,8,9,4]	2
[0,1,3,4,5,7,9,10,4,5,7,9,4,6]	[0,1,3,4,5,7,9,10], [4,5,7,9,10,4], [5,7,9,10,4,6]	3
[0,1,3,4,5,7,8,9,4,5,7,8,9,4,5,7,9,4,6]	[5,7,8,9,4,5], [4,5,7,8,9,4], [9,4,5,7,8,9], [7,8,9,4,5,7], [8,9,4,5,7,8], [5,7,9,4,6], [4,5,7,9,4], [9,4,5,7,9]	Not feasible
[0,1,3,4,5,7,9,4,5,7,9,4,5,7,9,4,6]	[5,7,9,4,5], [5,7,9,4,6], [4,5,7,9,4], [9,4,5,7,9], [7,9,4,5,7]	Not feasible
[0,1,3,4,5,7,8,9,10,4,5,7,8,9,10,4,6]	[0,1,3,4,5,7,8,9,10], [4,5,7,8,9,10,4], [5,7,8,9,10,4,5], [9,10,4,5,7,8,9], [10,4,5,7,8,9,10], [8,9,10,4,5,7,8], [5,7,8,9,10,4,6], [7,8,9,10,4,5,7]	Not feasible
[0,1,3,4,5,7,9,4,5,7,9,10,4,5,7,9,10,4,5,7,9,10,4,6]	[4,5,7,9,10,4], [5,7,9,10,4,5], [5,7,9,10,4,6], [9,10,4,5,7,9], [10,4,5,7,9,10], [7,9,10,4,5,7], [5,7,9,4,5], [4,5,7,9,4], [9,4,5,7,9], [7,9,4,5,7]	Not feasible
[0,1,3,4,6]	[0,1,3,4,6]	Not feasible

The Output of the Junit after the tests are run. All 58/58 test cases have run successfully.



levelOrder_test5() (0.001 s)
levelOrder_test6() (0.000 s)
numNodes_test1() (0.001 s)
numNodes_test2() (0.001 s)
numNodes_test3() (0.001 s)
inOrder_test1() (0.000 s)
inOrder_test2() (0.001 s)
inOrder_test3() (0.001 s)
inOrder_test4() (0.000 s)
CheckChildrenSum_test10() (0.001 s)
isSymmetric_test1() (0.001 s)
isSymmetric_test2() (0.001 s)
isSymmetric_test3() (0.001 s)
isSymmetric_test4() (0.000 s)
isSymmetric_test5() (0.000 s)
isSymmetric_test6() (0.001 s)
isSymmetric_test7() (0.001 s)
isSymmetric_test8() (0.000 s)
isSymmetric_test9() (0.001 s)
CheckChildrenSum_test1() (0.000 s)
CheckChildrenSum_test2() (0.001 s)
CheckChildrenSum_test3() (0.000 s)
CheckChildrenSum_test4() (0.000 s)
CheckChildrenSum_test5() (0.000 s)
CheckChildrenSum_test6() (0.000 s)
CheckChildrenSum_test7() (0.001 s)
CheckChildrenSum_test8() (0.000 s)
CheckChildrenSum_test9() (0.006 s)

Contributions

We have a total of 10 functions to test. So, we have split them among ourselves as follows

Maheedhar

1.PreOrderIterative
2.LevelOrderIterative
3.NumNodes
4.CheckChildrenSum
5.areMirrors

Manohar

1.PostOrderIterative
2.InOrderIterative
3.Height
4.isSymmetric
5.BottomView

THANK YOU