Piacente Cristian 866020

Homework H7 – MCDC

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

The following program fragment adds an object to the right branch of a tree structure.

```
if((o != null) && !tree.contains(o) && (tree.isBalanced() || (tree.right().size() < tree.left().size()))) {
    tree.addRight(o);
}
```

Derive a set of test cases that satisfy the MC/DC adequacy criterion. The solution should indicate the elementary conditions, specify the combinations of truth values that satisfy the MC/DC criterion, and provide a set of corresponding concrete test cases.

To derive a test suite that satisfies the MC/DC adequacy criterion, we can follow the following steps:

- Start from the truth table of the compound condition
- If short-circuit evaluated, delete all the basic conditions which don't get evaluated
- For each basic condition c choose a row r1 and a row r2 such that:
    - r1 and r2 have different values for c
    - r1 and r2 have different outcomes
    - Each other basic condition has the same evaluation in r1 and r2 (or is not evaluated)
- Add r1 and r2 to the test suite.

Let's define

A: o != null
B: !tree.contains(o)
C: tree.isBalanced()
D: tree.right().size() < tree.left().size()

The compound condition can be written as **A && B && (C || D)**.

We have the following truth table, before applying short-circuit evaluation:

	A	B	C	D	Outcome
1	False	False	False	False	False
2	False	False	False	True	False
3	False	False	True	False	False
4	False	False	True	True	False
5	False	True	False	False	False
6	False	True	False	True	False
7	False	True	True	False	False
8	False	True	True	True	False
9	True	False	False	False	False
10	True	False	False	True	False
11	True	False	True	False	False
12	True	False	True	True	False
13	True	True	False	False	False
14	True	True	False	True	True
15	True	True	True	False	True
16	True	True	True	True	True

Piacente Cristian 866020

Homework H7 – MCDC

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

Let's apply short-circuit evaluation to get the table below, where – means the basic condition doesn't get evaluated and the redundant rows have been deleted:

	A	B	C	D	Outcome
1	False	–	–	–	False
9	True	False	–	–	False
13	True	True	False	False	False
14	True	True	False	True	True
15	True	True	True	–	True

Now, let's choose for each basic condition two rows that satisfy MC/DC criterion.

Basic condition A:

	A	B	C	D	Outcome
1	**False**	–	–	–	False
9	True	False	–	–	False
13	True	True	False	False	False
14	**True**	True	False	True	True
15	True	True	True	–	True

Basic condition B:

	A	**B**	C	D	Outcome
1	False	–	–	–	False
9	True	**False**	–	–	False
13	True	True	False	False	False
14	True	**True**	False	True	True
15	True	True	True	–	True

Basic condition C:

	A	B	**C**	D	Outcome
1	False	–	–	–	False
9	True	False	–	–	False
13	True	True	**False**	False	False
14	True	True	False	True	True
15	True	True	**True**	–	True

Basic condition D:

	A	B	C	**D**	Outcome
1	False	–	–	–	False
9	True	False	–	–	False
13	True	True	False	**False**	False
14	True	True	False	**True**	True
15	True	True	True	–	True

Every row has been used for satisfying MC/DC criterion, so we can't exclude any combination of truth values.

We can now provide a concrete test case for each row.

W.r.t the definition of the basic conditions A, B, C, D given before (also shown below)

A: o != null
B: !tree.contains(o)
C: tree.isBalanced()
D: tree.right().size() < tree.left().size()

we have the following **concrete set of test cases**, where each – has been replaced with a concrete value:

Piacente Cristian 866020

Homework H7 – MCDC

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

1. o == null
   ➔ o is a null reference (A = False), the tree does NOT contain null (B = True), the tree is NOT balanced (C = False), the right size is less than the left size (D = True).
   We have **a null reference**, the other conditions don't really matter.

9. o != null && tree.contains(o)
   ➔ o is NOT a null reference (A = True), the tree contains o (B = False), the tree is NOT balanced (C = False), the right size is less than the left size (D = True).
   We have **an object which is already in the tree**, the other conditions don't really matter.

13. o != null && !tree.contains(o) && !tree.isBalanced() && tree.right().size() >= tree.left().size()
   ➔ o is NOT a null reference (A = True), the tree does NOT contain o (B = True), the tree is NOT balanced (C = False), the right size is greater than the left size (D = False; logically it could also be equal, but it wouldn't make sense to have a not balanced tree with the right size that equals to the left size).
   We have an **object which is NOT already in the tree**, and the tree has the **right size greater than the left size**, so it's **NOT balanced**.

14. o != null && !tree.contains(o) && !tree.isBalanced() && tree.right().size() < tree.left().size()
   ➔ o is NOT a null reference (A = True), the tree does NOT contain o (B = True), the tree is NOT balanced (C = False), the right size is less than the left size (D = True).
   We have an **object which is NOT already in the tree**, and the tree has the **right size less than the left size**, so it's **NOT balanced**.

15. o != null && !tree.contains(o) && tree.isBalanced() && tree.right().size() >= tree.left().size()
   ➔ o is NOT a null reference (A = True), the tree does NOT contain o (B = True), the tree is balanced (C = True), the right size equals to the left size (D = False; logically it could also be greater, but it wouldn't make sense to have a balanced tree with the right size that is greater than the left size).
   We have an **object which is NOT already in the tree**, and the tree is **balanced**.

In conclusion, this is the **concrete test suite** that satisfies MC/DC criterion:

Test case number	o != null	!tree.contains(o)	tree.isBalanced()	tree.right().size() < tree.left().size()	Outcome
1	False	True	False	True	False
9	True	False	False	True	False
13	True	True	False	False	False
14	True	True	False	True	True
15	True	True	True	False	True