CS 3250: Software Testing (Summer 2026)

Activity: Graph for source code (Structural) – LCM

(no submission)

Purpose: Create graph representation for source code; apply structural graph coverage to source code; get ready for the learning portfolio.

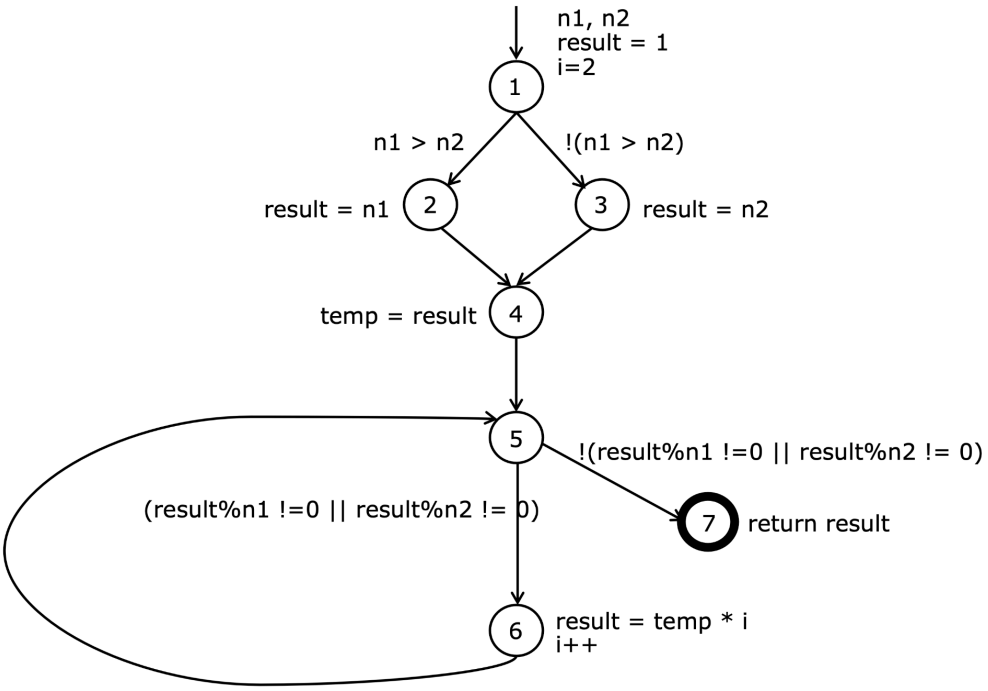
You may make a copy of a worksheet and complete this activity, or type your answers in any text editor. You may work alone or with at most two other students in this course.

Consider the following Java method

```
// Calculate_LCM takes two numbers and return LCM of the two numbers
// Examples:  LCM of 12 and 15 = 2 * 2 * 3 * 5 = 60
//            LCM of 12 and 18 = 2 * 2 * 3 * 3 = 36
//            LCM of 15 and 18 = 2 * 3 * 3 * 5 = 90
public static int Calculate_LCM(int n1, int n2) {
    int result = 1;
    int i = 2;
    if (n1 > n2)
        result = n1;
    else
        result = n2;

    int temp = result;
    while (result % n1 != 0 || result % n2 != 0) {
        result = temp * i;
        i++;
    }
    return result;
}
```

1. Draw a **Control Flow Graph** for the `Calculate_LCM` method (You may draw the graph by hand, take a picture of your graph, and embed it in your write-up)



1. Apply **Node Coverage** (NC) to design tests

Test requirements	Test paths	Test cases (input values and expected output)
{1, 2, 3, 4, 5, 6, 7}	{ [1,2,4,5,7], [1,3,4,5,6,5,7] }	For test path [1,2,4,5,7] Input: n1=4, n2=2 Expect: result=4 For test path [1,3,4,5,6,5,7] Input: n1=10, n2=15 Expect: result=30

2. Apply **Edge Coverage** (EC) to design tests

Test requirements	Test paths	Test cases (input values and expected output)
{ (1,2), (1,3), (2,4), (3,4), (4,5), (5,6), (5,7), (6,5) }	{ [1,2,4,5,7], [1,3,4,5,6,5,7] }	For test path [1,2,4,5,7] Input: n1=4, n2=2 Expect: result=4 For test path [1,3,4,5,6,5,7] Input: n1=10, n2=15 Expect: result=30

3. Apply **Edge-Pair Coverage** (EPC) to design tests

Test requirements	Test paths	Test cases (input values and expected output)
{ (1,2,4), (1,3,4), (2,4,5), (3,4,5), (4,5,6), (4,5,7), (5,6,5), (6,5,6), (6,5,7) }	{ [1,2,4,5,7], [1,3,4,5,7], [1,3,4,5,6,5,6,5,7] } [1,2,4,5,7] directly tours (1,2,4), (2,4,5), (4,5,7) [1,3,4,5,7] directly tours (1,3,4), (3,4,5), (4,5,7) [1,3,4,5,6,5,6,5,7] directly tour (1,3,4), (3,4,5), (4,5,6), (5,6,5), (6,5,6), (6,5,7) -- also tours (3,4,5), (4,5,6), (5,6,5), 6,5,7) with sidetrip	For test path [1,2,4,5,7] Input: n1=4, n2=2 Expect: result=4 For test path [1,3,4,5,7] Input: n1=2, n2=4 Expect: result=4 For test path [1,3,4,5,6,5,6,5,7] Input: n1=3, n2=14 Expect: result=42

4. Derive prime paths

[1] [2] [3] [4] [5] [6] ~~[7]~~!	[1,2] [1,3] [2,4] [3,4] [4,5] [5,6] ~~[5,7]~~! [6,5]	[1,2,4] [1,3,4] [2,4,5] [3,4,5] ~~[4,5,6]~~! **[5,6,5]*** **[6,5,6]*** **[6,5,7]!**	[1,2,4,5] [1,3,4,5] ~~[2,4,5,6]~~! ~~[2,4,5,7]~~! ~~[3,4,5,6]~~! ~~[3,4,5,7]~~!	**[1,2,4,5,6]!** **[1,2,4,5,7]!** **[1,3,4,5,6]!** **[1,3,4,5,7]!**

Expanding [4,5,6], [2,4,5,6], [3,4,5,6], [1,2,4,5,6] will result in internal loop (i.e., no longer simple paths)
Reminder: prime path is a simple path that is not a subpath of any other paths.

5. Apply **Prime Path Coverage** (PPC) to design tests

Test requirements	Test paths	Test cases (input values and expected output)
{ (1,2,4,5,7), (1,2,4,5,6), (1,3,4,5,6), (1,3,4,5,7), (5,6,5), (6,5,6), (6,5,7) }	{ [1,2,4,5,7], [1,2,4,5,6,5,7] [1,3,4,5,7], [1,3,4,5,6,5,6,5,7] } [1,2,4,5,7] tours (1,2,4,5,7) directly [1,2,4,5,6,5,7] tours (1,2,4,5,6), (5,6,5), (6,5,7) directly -- also tours (1,2,4,5,7) with sidetrip [1,3,4,5,7] tours (1,3,4,5,7) directly [1,3,4,5,6,5,6,5,7] tours (1,3,4,5,6), (5,6,5), (6,5,6), (6,5,7) directly -- also tours (1,3,4,5,6), (5,6,5), (6,5,6) with sidetrip	For test path [1,2,4,5,7] Input: n1=4, n2=2 Expect: result=4 For test path [1,2,4,5,6,5,7] Input: n1=15, n2=10 Expect: result=30 For test path [1,3,4,5,7] Input: n1=2, n2=4 Expect: result=4 For test path [1,3,4,5,6,5,6,5,7] Input: n1=3, n2=14 Expect: result=42

Last updated 2026-05-30 23:42