# Task 3: White-box Testing: Data Flow Testing

Below are the control flow graphs and program code for the gcd and compareTo methods. Defs will appear highlighted in blue and Uses will appear highlighted in green.

GCD

public BigInteger gcd(BigInteger y)

{

int xval = ival;

int yval = y.ival;

if (words == null)

{

if (xval == 0)

return abs(y);

if (y.words == null && xval != Integer.MIN\_VALUE && yval !=

Integer.MIN\_VALUE)

{

if (xval < 0)

xval = -xval;

if (yval < 0)

yval = -yval;

return valueOf(gcd(xval, yval));

}

xval = 1;

}

if (y.words == null)

{

if (yval == 0)

return abs(this);

yval = 1;

}

int len = (xval > yval ? xval : yval) + 1;

int[] xwords = new int[len];

int[] ywords = new int[len];

getAbsolute(xwords);

y.getAbsolute(ywords);

len = MPN.gcd(xwords, ywords, len);

BigInteger result = new BigInteger(0);

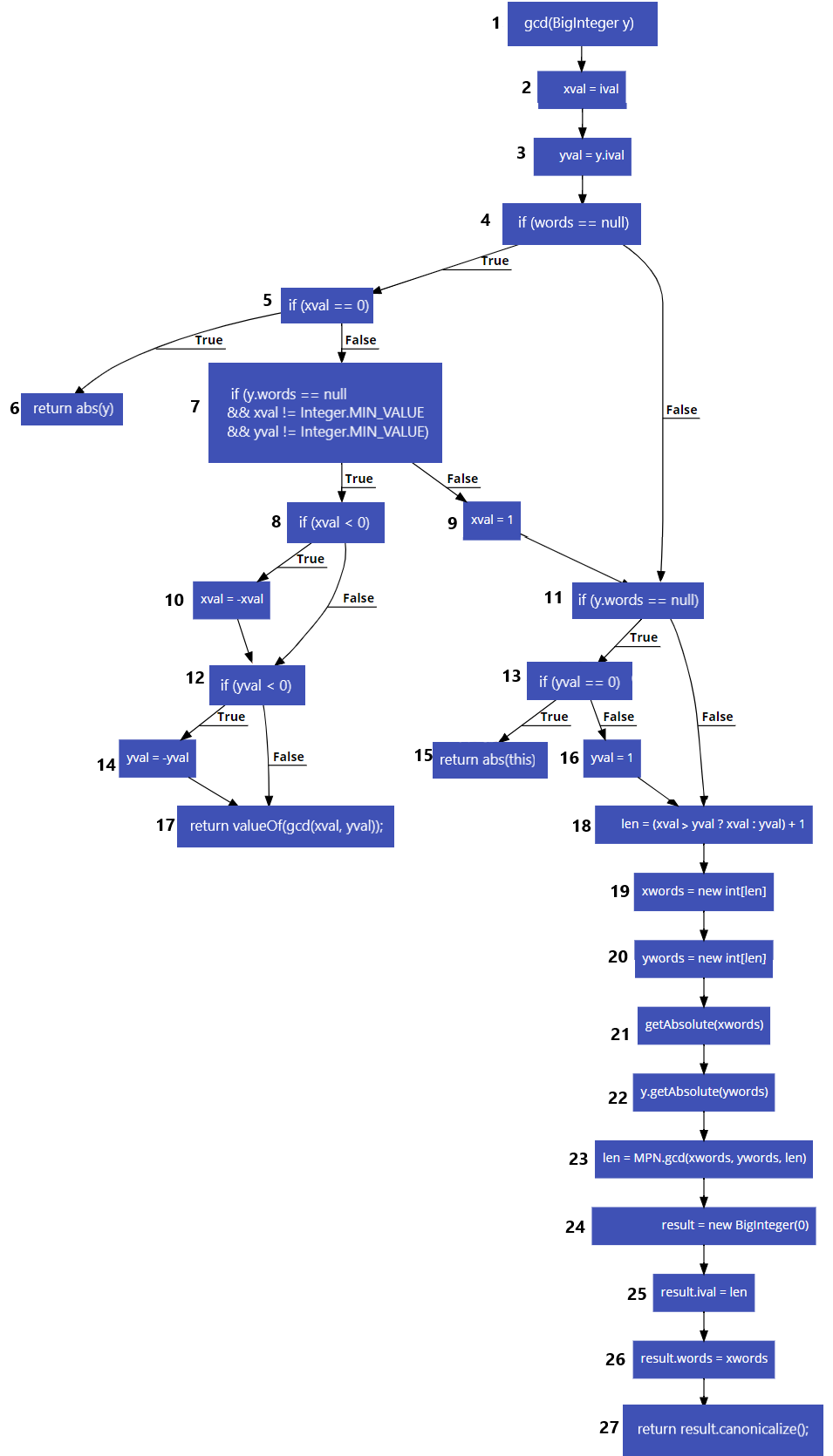
result.ival = len;

result.words = xwords;

return result.canonicalize();

}

**Variables** : y, xval, ival, yval, y.ival, Integer.MIN\_VALUE, words, y.words, len, this, xwords, ywords, result



CompareTo

private static int compareTo(BigInteger x, BigInteger y)

{

if (x.words == null && y.words == null)

return x.ival < y.ival ? -1 : x.ival > y.ival ? 1 : 0;

boolean x\_negative = x.isNegative();

boolean y\_negative = y.isNegative();

if (x\_negative != y\_negative)

return x\_negative ? -1 : 1;

int x\_len = x.words == null ? 1 : x.ival;

int y\_len = y.words == null ? 1 : y.ival;

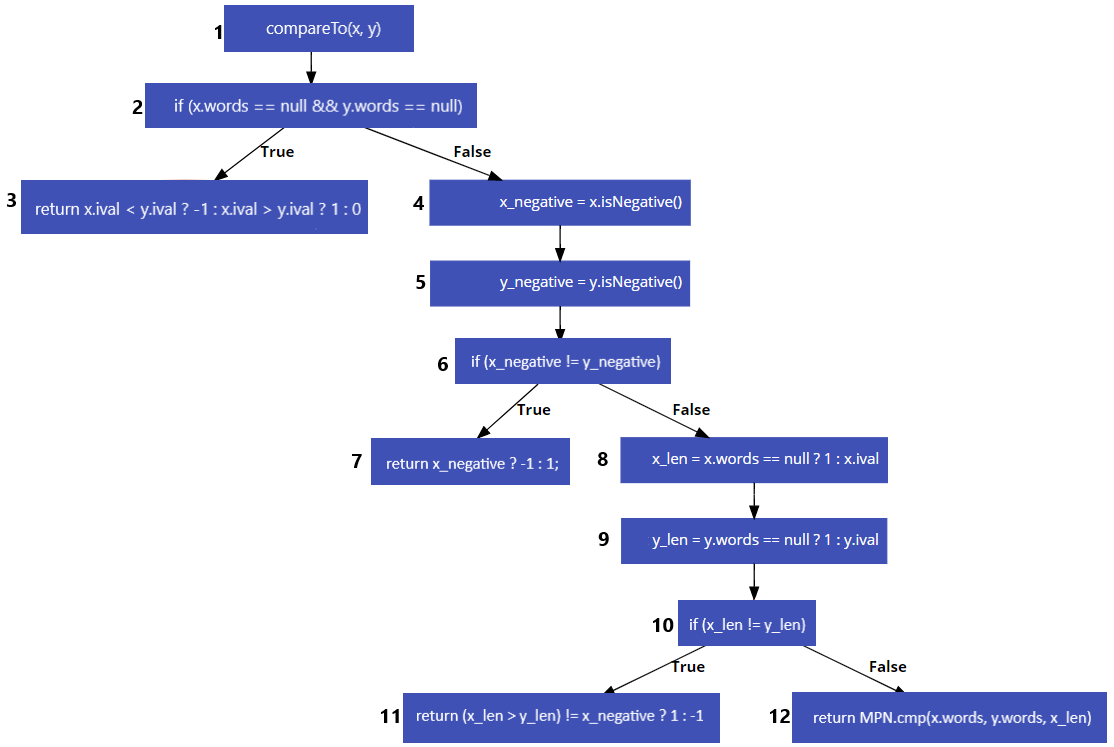
if (x\_len != y\_len)

return (x\_len > y\_len) != x\_negative ? 1 : -1;

return MPN.cmp(x.words, y.words, x\_len);

}

**Variables**: x, y, x.ival, y.ival, x.words, y.words, x\_negative, y\_negative, x\_len, y\_len



## Identify all the definition-use pairs (du-pairs)

Below are tables containing the du-pairs of each of the variables in these methods. The paths correspond to the nodes in their respective control flow graphs. The “~” symbol appears when the variable was defined outside of the method.

# GCD

|  |  |
| --- | --- |
| ival | |
| Du-pair | **Path** |
| (~, 2) | <~, 1, 2> |

|  |  |
| --- | --- |
| y | |
| Du-pair | **Path** |
| (1, 3) | <1, 2, 3> |
| (1, 6) | <1, 2, 3, 4, 5, 6> |
| (1, <7, 8>) | <1, 2, 3, 4, 5, 7, 8> |
| (1, <7, 9>) | <1, 2, 3, 4, 5, 7, 9> |
| (1, <11, 13>) | <1, 2, 3, 4, 5, 7, 9, 11, 13> |
|  | <1, 2, 3, 4, 11, 13> |
| (1, <11, 18>) | <1, 2, 3, 4, 5, 7, 9, 11, 18> |
|  | <1, 2, 3, 4, 11, 18> |
| (1, 22) | <1, 2, 3, 4, 5, 7, 9, 11, 18, 19, 20, 21, 22> |
|  | <1, 2, 3, 4, 11, 18, 19, 20, 21, 22> |

|  |  |
| --- | --- |
| y.ival | |
| Du-pair | **Path** |
| (1, 3) | <1, 2, 3> |

|  |  |
| --- | --- |
| Integer.MIN\_VALUE | |
| Du-pair | **Path** |
| (~, 7) | <~, 1, 2, 3, 4, 5, 7> |

|  |  |
| --- | --- |
| xval | |
| Du-pair | **Path** |
| (2, <5, 6>) | <2, 3, 4, 5, 6> |
| (2, <5, 7>) | <2, 3, 4, 5, 7> |
| (2, <7, 8>) | <2, 3, 4, 5, 7, 8> |
| (2, <7, 9>) | <2, 3, 4, 5, 7, 9> |
| (2, <8, 10>) | <2, 3, 4, 5, 7, 8, 10> |
| (2, <8, 12>) | <2, 3, 4, 5, 7, 8, 12> |
| (2, 10) | <2, 3, 4, 5, 7, 8, 10> |
| (2, 17) | <2, 3, 4, 5, 7, 8, 12, 14, 17> |
|  | <2, 3, 4, 5, 7, 8, 12, 17> |
| (2, 18) | <2, 3, 4, 11, 13, 16, 18> |
|  | <2, 3, 4, 11, 18> |
| (9, 18) | <9, 11, 13, 16, 18> |
|  | <9, 11, 18> |
| (10, 17) | <10, 12, 17> |
|  | <10, 12, 14, 17> |

|  |  |
| --- | --- |
| yval | |
| Du-pair | **Path** |
| (3, <7, 8>) | <3, 4, 5, 7, 8> |
| (3, <7, 8>) | <3, 4, 5, 7, 9> |
| (3, <12, 14>) | <3, 4, 5, 7, 8, 12, 14> |
|  | <3, 4, 5, 7, 8, 10, 12, 14> |
| (3, <12, 17>) | <3, 4, 5, 7, 8, 12, 17> |
|  | <3, 4, 5, 7, 8, 10, 12, 17> |
| (3, <13, 15>) | <3, 4, 5, 7, 9, 11, 13, 15> |
|  | <3, 4, 11, 13, 15> |
| (3, <13, 16>) | <3, 4, 5, 7, 9, 11, 13, 16> |
|  | <3, 4, 11, 13, 16> |
| (3, 14) | <3, 4, 5, 7, 8, 10, 12, 14> |
|  | <3, 4, 5, 7, 8, 12, 14> |
| (3, 17) | <3, 4, 5, 7, 8, 10, 12, 17> |
|  | <3, 4, 5, 7, 8, 12, 17> |
| (14, 17) | <14, 17> |
| (16, 18) | <16, 18> |

|  |  |
| --- | --- |
| words | |
| Du-pair | **Path** |
| (~, <4, 5>) | <~, 1, 2, 3, 4, 5> |
| (~, <4, 11>) | <~, 1, 2, 3, 4, 11> |

|  |  |
| --- | --- |
| this | |
| Du-pair | **Path** |
| (~, 15) | <~, 1, 2, 3, 4, 5, 7, 9, 11, 13, 15> |
|  | <~, 1, 2, 3, 4, 11, 13, 15> |

|  |  |
| --- | --- |
| xwords | |
| Du-pair | **Path** |
| (19, 21) | <19, 20, 21> |
| (19, 23) | <19, 20, 21, 22, 23> |
| (19, 26) | <19, 20, 21, 22, 23, 24, 25, 26> |

|  |  |
| --- | --- |
| ywords | |
| Du-pair | **Path** |
| (20, 22) | <20, 21, 22> |
| (20, 23) | <20, 21, 22, 23> |

|  |  |
| --- | --- |
| y.words | |
| Du-pair | **Path** |
| (1, <7, 8>) | <1, 2, 3, 4, 5, 7, 8> |
| (1, <7, 9>) | <1, 2, 3, 4, 5, 7, 9> |
| (1, <11, 13>) | <1, 2, 3, 4, 5, 7, 9, 11, 13> |
|  | <1, 2, 3, 4, 11, 13> |
| (1, <11, 18>) | <1, 2, 3, 4, 5, 7, 9, 11, 18> |
|  | <1, 2, 3, 4, 11, 18> |

|  |  |
| --- | --- |
| len | |
| Du-pair | **Path** |
| (18, 19) | <18, 19> |
| (18, 20) | <18, 19, 20> |
| (18, 23) | <18, 19, 20, 21, 22, 23> |
| (23, 25) | <23, 24, 25> |

|  |  |
| --- | --- |
| result | |
| Du-pair | **Path** |
| (24, 25) | <24, 25> |
| (24, 26) | <24, 25, 26> |
| (24, 27) | <24, 25, 26, 27> |

# compareTo

|  |  |
| --- | --- |
| x | |
| Du-pair | **Path** |
| (1, <2, 3>) | <1, 2, 3> |
| (1, <2, 4>) | <1, 2, 4> |
| (1, 3) | <1, 2, 3> |
| (1, 4) | <1, 2, 4> |
| (1, 8) | <1, 2, 4, 5, 6, 8> |
| (1, 12) | <1, 2, 4, 5, 6, 8, 9, 10, 12> |

|  |  |
| --- | --- |
| y | |
| Du-pair | **Path** |
| (1, <2, 3>) | <1, 2, 3> |
| (1, <2, 4>) | <1, 2, 4> |
| (1, 3) | <1, 2, 3> |
| (1, 5) | <1, 2, 4, 5> |
| (1, 9) | <1, 2, 4, 5, 6, 8, 9> |
| (1, 12) | <1, 2, 4, 5, 6, 8, 9, 10, 12> |

|  |  |
| --- | --- |
| x.ival | |
| Du-pair | **Path** |
| (1, 3) | <1, 2, 3> |
| (1, 8) | <1, 2, 4, 5, 6, 8> |

|  |  |
| --- | --- |
| y.ival | |
| Du-pair | **Path** |
| (1, 3) | <1, 2, 3> |
| (1, 9) | <1, 2, 4, 5, 6, 8, 9> |

|  |  |
| --- | --- |
| y.words | |
| Du-pair | **Path** |
| (1, <2, 3>) | <1, 2, 3> |
| (1, <2, 4>) | <1, 2, 4> |
| (1, 9) | <1, 2, 4, 5, 6, 8, 9> |
| (1, 12) | <1, 2, 4, 5, 6, 8, 9, 10, 12> |

|  |  |
| --- | --- |
| x.words | |
| Du-pair | **Path** |
| (1, <2, 3>) | <1, 2, 3> |
| (1, <2, 4>) | <1, 2, 4> |
| (1, 8) | <1, 2, 4, 5, 6, 8> |
| (1, 12) | <1, 2, 4, 5, 6, 8, 9, 10, 12> |

|  |  |
| --- | --- |
| x\_negative | |
| Du-pair | **Path** |
| (4, <6, 7>) | <4, 5, 6, 7> |
| (4, <6, 8>) | <4, 5, 6, 8> |
| (4, 7) | <4, 5, 6, 7> |
| (4, 11) | <4, 5, 6, 8, 9, 10, 11> |

|  |  |
| --- | --- |
| y\_negative | |
| Du-pair | **Path** |
| (5, <6, 7>) | <5, 6, 7> |
| (5, <6, 8>) | <5, 6, 8> |

|  |  |
| --- | --- |
| x\_len | |
| Du-pair | **Path** |
| (8, <10, 11>) | <8, 9, 10, 11> |
| (8, <10, 12>) | <8, 9, 10, 12> |
| (8, 11) | <8, 9, 10, 11> |
| (8, 12) | <8, 9, 10, 12> |

|  |  |
| --- | --- |
| y\_len | |
| Du-pair | **Path** |
| (9, <10, 11>) | <9, 10, 11> |
| (9, <10, 12>) | <9, 10, 12> |
| (9, 11) | <9, 10, 11> |

## Design test cases to achieve All-Defs coverage

The values along the top row represent the initial values of those variables in the test case. The variables along the left column represent the variables that have achieved full coverage after that test case.

### GCD

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Covered | this | y | xval | y.ival | yval | words | y.words | len | xwords | ywords |
| len  xwords  ywords  result | 1 | -2^31 | 1 | -2^31 | -2^31 | null | null | 2 | [,] | [,] |
| Path: <1, 2, 3, 4, 5, 7, 9, 11, 13, 16, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27> | | | | | | | | | |
| xval  yval  y | -1 | -1 | -1 | -1 | -1 | null | null | - | - | - |
| Path: <1, 2, 3, 4, 5, 7, 8, 10, 12, 14, 17> | | | | | | | | | |

### compareTo

All definitions were reached within a single test case

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Covered | x | y | x.ival | y.ival | x.words | y.words | x\_negative | y\_negative | x\_len | y\_len |
| X  Y  x.words  y.words  x.ival  y,ival  x\_negative  y\_negative  x\_len  y\_len | “1” | “2” | 1 | 2 | [1] | [2] | F | F | 1 | 2 |
| Path: <1, 2, 4, 5, 6, 8, 9, 10, 11> | | | | | | | | | |

## Design test cases to achieve All-Uses coverage

### GCD

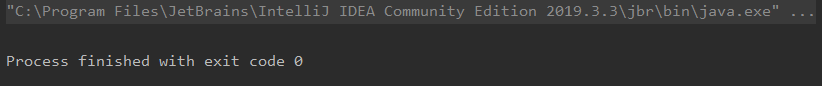
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Covered | this | y | xval | y.ival | yval | words | y.words | len | xwords | ywords |
|  | 0 | 0 | 0 | 0 | 0 | Null | null | - | - | - |
| Path: <1, 2, 3, 4, 5, 6> | | | | | | | | | |
|  | -1 | -1 | -1 | -1 | -1 | null | null | - | - | - |
| Path: <1, 2, 3, 4, 5, 7, 8, 10, 12, 14, 17> | | | | | | | | | |
|  | 1 | 1 | 1 | 1 | 1 | null | null | - | - | - |
| Path: <1, 2, 3, 4, 5, 7, 8, 12, 17> | | | | | | | | | |
| len  xwords  ywords  result  y | 1 | -2^31 | 1 | -2^31 | -2^31 | null | null | 2 | [,] | [,] |
| Path: <1, 2, 3, 4, 5, 7, 9, 11, 13, 16, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27> | | | | | | | | | |
| yval  xval | “1” | “1” | 1 | 1 | 1 | [1] | [1] | 2 | [,] | [,] |
| Path: <1, 2, 3, 4, 11, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27> | | | | | | | | | |

### compareTo

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Covered | x | y | x.ival | y.ival | x.words | y.words | x\_negative | y\_negative | x\_len | y\_len |
|  | 1 | 2 | 1 | 2 | null | null | - | - | - | - |
| Path: <1, 2, 3> | | | | | | | | | |
| y\_negative  x.ival  y.ival | “1” | “2” | 1 | 2 | [1] | [2] | F | F | 1 | 2 |
| Path: <1, 2, 4, 5, 6, 8, 9, 10, 11> | | | | | | | | | |
| x\_negative | “-1” | “2” | -1 | 2 | [-1] | [2] | T | F | - | - |
| Path: <1, 2, 4, 5, 6, 7> | | | | | | | | | |
| x  y  x.words  y.words  x\_len  y\_len | “1” | “1” | 1 | 1 | [1] | [1] | F | F | 1 | 1 |
| <1, 2, 4, 5, 6, 8, 9, 10, 12> | | | | | | | | | |

## 4) Write and execute the test cases in JUnit

package com.example.testCases;  
import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.api.Test;  
  
import java.math.BigInteger;  
  
public class TestCases {  
 public static void main(String[] args) {  
 *testBI*();  
 }  
  
 @Test  
 public static void testBI(){  
 //====BigIntegers with integer constructors are accessed through "valueOf()"  
  
 //Initialize variables  
 //\_\_All-defs coverage\_\_  
 // GCD  
 BigInteger dgcd1x = BigInteger.*valueOf*(1L);  
 BigInteger dgcd1y = BigInteger.*valueOf*(-2147483648L);  
  
 BigInteger dgcd2x = BigInteger.*valueOf*(-1L);  
 BigInteger dgcd2y = BigInteger.*valueOf*(-1L);  
  
 // compareTo  
 BigInteger dcompareTo1x = new BigInteger("1");  
 BigInteger dcompareTo1y = new BigInteger("2");  
  
  
 //\_\_All-uses coverage\_\_  
 // GCD  
 BigInteger ugcd1x = BigInteger.*valueOf*(0L);  
 BigInteger ugcd1y = BigInteger.*valueOf*(0L);  
  
 BigInteger ugcd2x = BigInteger.*valueOf*(-1L);  
 BigInteger ugcd2y = BigInteger.*valueOf*(-1L);  
  
 BigInteger ugcd3x = BigInteger.*valueOf*(1L);  
 BigInteger ugcd3y = BigInteger.*valueOf*(1L);  
  
 BigInteger ugcd4x = BigInteger.*valueOf*(1L);  
 BigInteger ugcd4y = BigInteger.*valueOf*(-2147483648L);  
  
 BigInteger ugcd5x = new BigInteger("1");  
 BigInteger ugcd5y = new BigInteger("1");  
  
 // compareTo  
 BigInteger ucompareTo1x = BigInteger.*valueOf*(1);  
 BigInteger ucompareTo1y = BigInteger.*valueOf*(2);  
  
 BigInteger ucompareTo2x = new BigInteger("1");  
 BigInteger ucompareTo2y = new BigInteger("2");  
  
 BigInteger ucompareTo3x = new BigInteger("-1");  
 BigInteger ucompareTo3y = new BigInteger("2");  
  
 BigInteger ucompareTo4x = new BigInteger("1");  
 BigInteger ucompareTo4y = new BigInteger("1");  
  
 //Run tests  
 //All-defs coverage  
 //GCD  
 *assertEquals*(BigInteger.*valueOf*(1), dgcd1x.gcd(dgcd1y));  
 *assertEquals*(BigInteger.*valueOf*(1), dgcd2x.gcd(dgcd2y));  
  
 //compareTo  
 *assertEquals*(-1, dcompareTo1x.compareTo(dcompareTo1y));  
  
 //All-uses coverage  
 //GCD  
 *assertEquals*(BigInteger.*valueOf*(0), ugcd1x.gcd(ugcd1y));  
 *assertEquals*(BigInteger.*valueOf*(1), ugcd2x.gcd(ugcd2y));  
 *assertEquals*(BigInteger.*valueOf*(1), ugcd3x.gcd(ugcd3y));  
 *assertEquals*(BigInteger.*valueOf*(1), ugcd4x.gcd(ugcd4y));  
 *assertEquals*(BigInteger.*valueOf*(1), ugcd5x.gcd(ugcd5y));  
  
 //compareTo  
 *assertEquals*(-1, ucompareTo1x.compareTo(ucompareTo1y));  
 *assertEquals*(-1, ucompareTo2x.compareTo(ucompareTo2y));  
 *assertEquals*(-1, ucompareTo3x.compareTo(ucompareTo3y));  
 *assertEquals*(0, ucompareTo4x.compareTo(ucompareTo4y));  
 }  
}



All test cases completed successfully.

# Report

### Task 3 – Tyson Grant

test environment

The BigInteger class contains many member variables that need to be tweaked in order to successfully achieve full test coverage. Beyond the various public constructors, certain private methods and variables needed to be altered in order to execute all required paths. This was achieved by using public functions, such as ‘valueOf(int)’ and ‘compareTo(BigInteger)’. ‘valueOf’ was used to create BigIntegers with null ‘words’ variables, and ‘compareTo’ which was a public member function of the BigInteger class was used to call the private ‘compareTo’ function which was required for testing.

Isolating the required variables was not as straight forward as initially thought. Some variables overlap such that some variables would be a member of another variable. For example, ‘y’ was used to depict a ‘BigInteger’ object, but it also contained ‘y.ival’ and similar sub variables. This was considered when designing test cases. Any variables that were only defined outside of the methods were not considered to have du-pairs within the scope of the methods. These variables still needed to be considered when testing for other variables.

### test objective

To achieve all-defs and all-uses coverage when executing white-box dataflow testing on the ‘gcd’ and ‘compareTo’ methods defined in the ‘BigInteger’ java class.

### test cases

The test cases were designed to follow each of the paths required to successfully achieve all-defs and all-uses coverage for the ‘gcd’ and ‘compareTo’ methods. Although some of the variables included in the test cases did not contain du-pairs, they had to be altered in order to reach all required paths for the other relevant variables. The test cases were designed to consider every du-pair with the minimum number of test cases.

### test execution results

All test cases completed with exit code zero, suggesting all test cases returned the expected results.

### test coverage

By utilising the ‘valueOf’ and ‘compareTo’ functions, it was possible to achieve all-defs and all-uses coverage.

## ReadMe

A1Task3.zip contains the intelliJ package for running the white-box dataflow test cases.