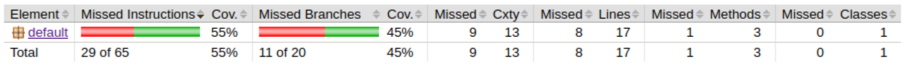
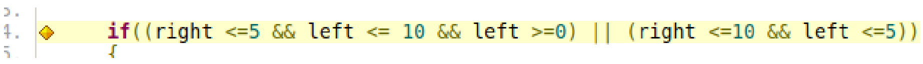
COMS 417 Assignment 1

List the Test cases you have added and for each state:

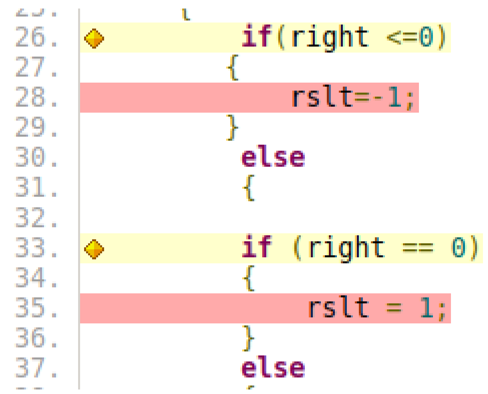


Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
default	🟥🟩	55%	🟥🟩	45%	9	13	8	17	1	3	0	1
Total	29 of 65	55%	11 of 20	45%	9	13	8	17	1	3	0	1

Coverage Before:

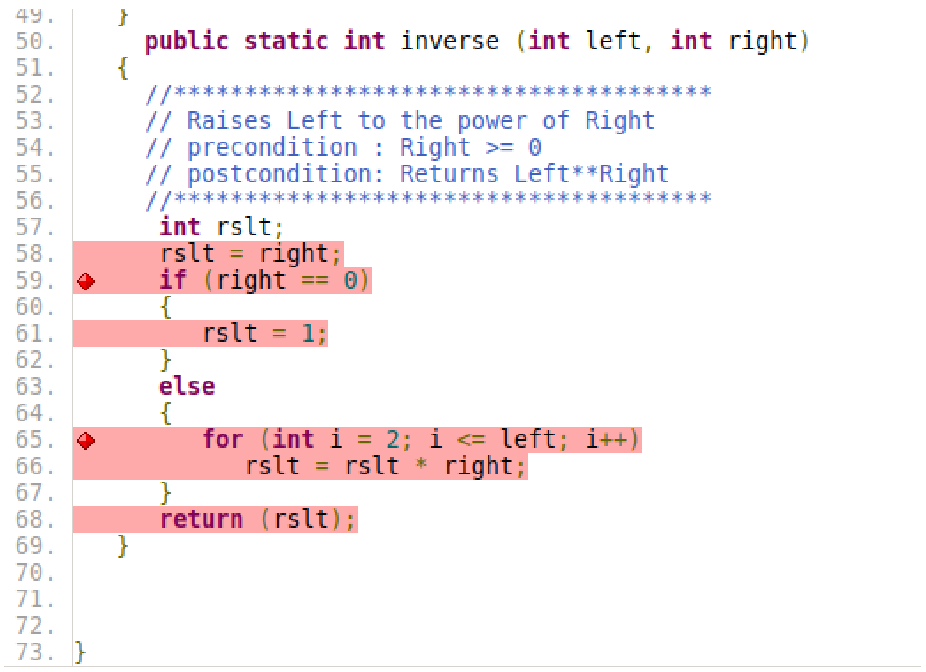


```
    if((right <=5 && left <= 10 && left >=0) || (right <=10 && left <=5))
```



```
           if(right <=0)
         {
              rslt=-1;
         }
           else
           {

           if (right == 0)
           {
              rslt = 1;
           }
           else
```



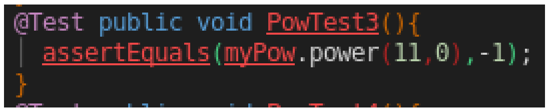
```
    }
    public static int inverse (int left, int right)
    {
    //****************************************
    // Raises Left to the power of Right
    // precondition : Right >= 0
    // postcondition: Returns Left**Right
    //****************************************
     int rslt;
     rslt = right;
     if (right == 0)
     {
        rslt = 1;
     }
     else
     {
        for (int i = 2; i <= left; i++)
           rslt = rslt * right;
     }
     return (rslt);
    }




}
```

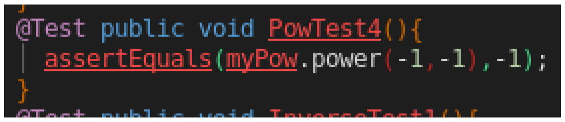
These reports tell me that there is not enough testing to go through all of the branches, and test all of the code The inverse method is not tested at all, and there is not enough testing on the power method either

---



```
@Test public void PowTest3(){
    assertEquals(myPow.power(11,0),-1);
}
```

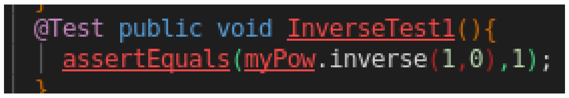
- This test is testing lines 24 and 26 in Power.java, going through the case that `right` is <= 0, and that `left` is > 10



```
@Test public void PowTest4(){
    assertEquals(myPow.power(-1,-1),-1);
}
```

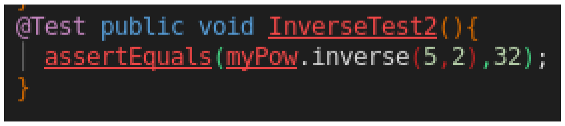
- This test is testing the big `if` statement on line 24 where both the `left` and `right` are negative

---



```
@Test public void InverseTest1(){
    assertEquals(myPow.inverse(1,0),1);
```

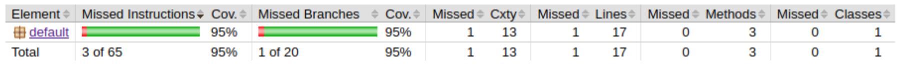
- This test is testing the inverse function, going through the branch where `right == 0`



```
@Test public void InverseTest2(){
    assertEquals(myPow.inverse(5,2),32);
}
```

- This test is making sure that the `else` statements works on line 63, that the branch returns $right^{left}$

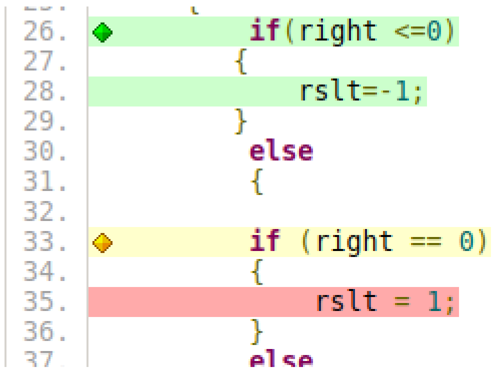
Coverage After:



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
default	🟥🟩	95%	🟥🟩	95%	1	13	1	17	0	3	0	1
Total	3 of 65	95%	1 of 20	95%	1	13	1	17	0	3	0	1

Maximum Coverage

- The maximum coverage is 95%, because there is a line of unreachable code, which is because the `if` statement on line 26 will always accept when `right <= 0`, so line 33 will always be covered above on line 26

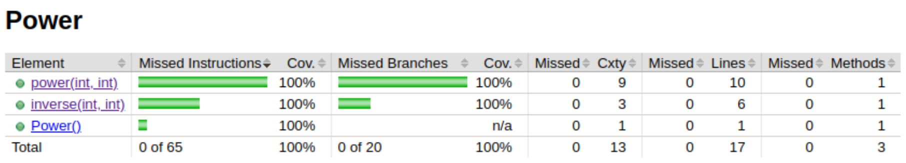


```
26.             if(right <=0)
27.            {
28.                 rslt=-1;
29.            }
30.             else
31.             {
32.
33.             if (right == 0)
34.             {
35.                 rslt = 1;
36.             }
37.             else
```

Two Faults not due to numeric overflow

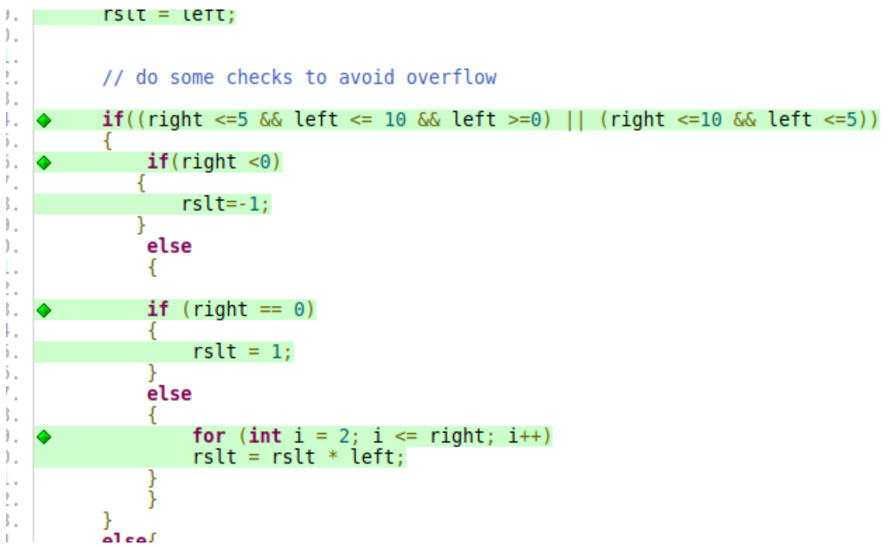
1. the case x^0 will return `-1` because line 26 will return -1 if `right` is zero
2. the inverse function can actually overflow when using very large numbers

Fixing the Faults The fault in this program can be fixed by changing the `right <= 0` to `right < 0`



Power

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
power(int, int)	🟩	100%	🟩	100%	0	9	0	10	0	1
inverse(int, int)	🟩	100%	🟩	100%	0	3	0	6	0	1
Power()	🟩	100%		n/a	0	1	0	1	0	1
Total	0 of 65	100%	0 of 20	100%	0	13	0	17	0	3



```
            rslt = left;


        // do some checks to avoid overflow

        if((right <=5 && left <= 10 && left >=0) || (right <=10 && left <=5))
        {
            if(right <0)
           {
                rslt=-1;
           }
            else
            {

            if (right == 0)
            {
                rslt = 1;
            }
            else
            {
                for (int i = 2; i <= right; i++)
                rslt = rslt * left;
            }
            }
        }
        else{
```

Finding the overflow The inverse function does not have any overflow protection, whereas the power function does. This means that the inverse function can fail when very large values are input. Adding this does not increase the test coverage, since it was already at 100%