Author

Kang Shentu

001569432

Conclusion

> **N is the number of steps and D is the man from the lamp post.**

Delta means the difference between the expected value and the actual value. The more tests, the more likely Delta will be to zero

$$\sqrt{n} = d \pm \Delta$$

Prove

According to the given topic, we can only get the expected value of the distance, that is, to find the following expected value

$$E_n(X^2+Y^2) = \sum (x^2+y^2)P(X=x, Y=y)$$

According to the same possibility of the four directions, it can be concluded that

$$P(X=x+1, Y=y) = P(X=x+1, Y=y|X=x, Y=y)P(X=x, Y=y) = \frac{1}{4}P(X=x, Y=y)$$

Therefore

$$E_{n+1}(X^2+Y^2) = \frac{1}{4}\sum[(x+1)^2+y^2]+[x^2+(y+1)^2]+[(x-1)^2+y^2]+[x^2+(y-1)^2]P(X=x, Y=y)$$

It can be obtained by simplification

$$E_{n+1}(X^2+Y^2) = \sum(x^2+y^2+1)P(X=x, Y=y) = E_n(X^2+Y^2)+\sum P(X=x, Y=y)$$

Absolutely

$$\sum P(X=x, Y=y) = 1$$

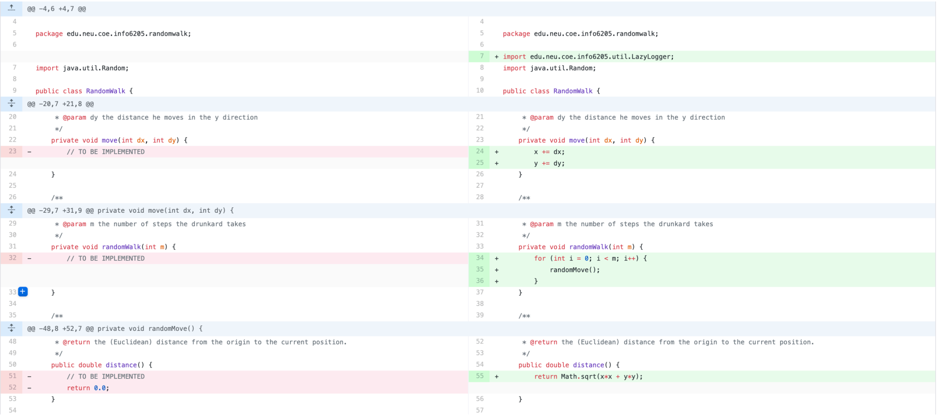
So, we can get

$$E_n(X^2+Y^2) = n$$

That is to say, the number of steps is the square of the expected Euclidean distance

Code Change

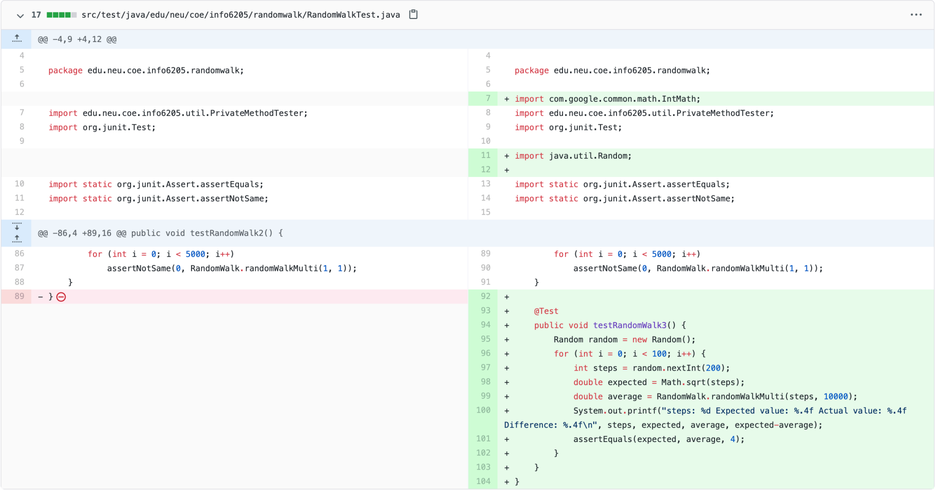
1. RandomWork.java



```
@@ -4,6 +4,7 @@

  package edu.neu.coe.info6205.randomwalk;


  import java.util.Random;

  public class RandomWalk {
@@ -20,7 +21,8 @@
       * @param dy the distance he moves in the y direction
       */
      private void move(int dx, int dy) {
-         // TO BE IMPLEMENTED

      }

      /**
@@ -29,7 +31,9 @@ private void move(int dx, int dy) {
       * @param m the number of steps the drunkard takes
       */
      private void randomWalk(int m) {
-         // TO BE IMPLEMENTED



      }

      /**
@@ -48,8 +52,7 @@ private void randomMove() {
       * @return the (Euclidean) distance from the origin to the current position.
       */
      public double distance() {
-         // TO BE IMPLEMENTED
-         return 0.0;
      }

```

```

  package edu.neu.coe.info6205.randomwalk;

+ import edu.neu.coe.info6205.util.LazyLogger;
  import java.util.Random;

  public class RandomWalk {

       * @param dy the distance he moves in the y direction
       */
      private void move(int dx, int dy) {
+         x += dx;
+         y += dy;
      }

      /**

       * @param m the number of steps the drunkard takes
       */
      private void randomWalk(int m) {
+         for (int i = 0; i < m; i++) {
+             randomMove();
+         }
      }

      /**

       * @return the (Euclidean) distance from the origin to the current position.
       */
      public double distance() {
+         return Math.sqrt(x*x + y*y);

      }

```

2. RandomWorkTest.java



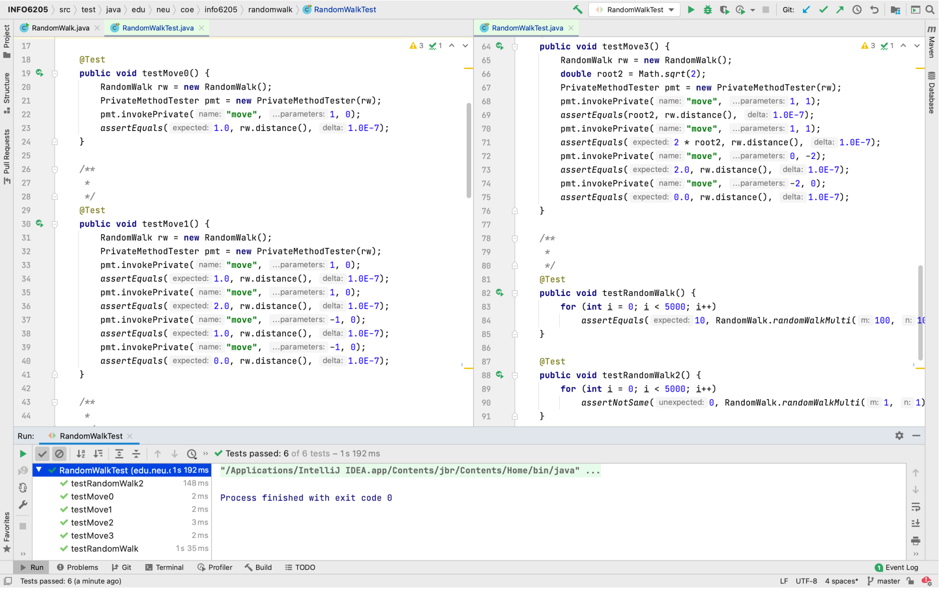
17 src/test/java/edu/neu/coe/info6205/randomwalk/RandomWalkTest.java

```
@@ -4,9 +4,12 @@

  package edu.neu.coe.info6205.randomwalk;


  import edu.neu.coe.info6205.util.PrivateMethodTester;
  import org.junit.Test;



  import static org.junit.Assert.assertEquals;
  import static org.junit.Assert.assertNotSame;

@@ -86,4 +89,16 @@ public void testRandomWalk2() {
          for (int i = 0; i < 5000; i++)
              assertNotSame(0, RandomWalk.randomWalkMulti(1, 1));
      }
- }
```

```

  package edu.neu.coe.info6205.randomwalk;

+ import com.google.common.math.IntMath;
  import edu.neu.coe.info6205.util.PrivateMethodTester;
  import org.junit.Test;

+ import java.util.Random;
+
  import static org.junit.Assert.assertEquals;
  import static org.junit.Assert.assertNotSame;


          for (int i = 0; i < 5000; i++)
              assertNotSame(0, RandomWalk.randomWalkMulti(1, 1));
      }
+
+     @Test
+     public void testRandomWalk3() {
+         Random random = new Random();
+         for (int i = 0; i < 100; i++) {
+             int steps = random.nextInt(200);
+             double expected = Math.sqrt(steps);
+             double average = RandomWalk.randomWalkMulti(steps, 10000);
+             System.out.printf("steps: %d Expected value: %.4f Actual value: %.4f Difference: %.4f\n", steps, expected, average, expected-average);
+             assertEquals(expected, average, 4);
+         }
+     }
+ }
```

Screen Shot

Given Test



```
    @Test
    public void testMove0() {
        RandomWalk rw = new RandomWalk();
        PrivateMethodTester pmt = new PrivateMethodTester(rw);
        pmt.invokePrivate( name: "move",  ...parameters: 1, 0);
        assertEquals( expected: 1.0, rw.distance(),  delta: 1.0E-7);
    }

    /**
     *
     */
    @Test
    public void testMove1() {
        RandomWalk rw = new RandomWalk();
        PrivateMethodTester pmt = new PrivateMethodTester(rw);
        pmt.invokePrivate( name: "move",  ...parameters: 1, 0);
        assertEquals( expected: 1.0, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: 1, 0);
        assertEquals( expected: 2.0, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: -1, 0);
        assertEquals( expected: 1.0, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: -1, 0);
        assertEquals( expected: 0.0, rw.distance(),  delta: 1.0E-7);
    }

    /**
     *
```

```
    public void testMove3() {
        RandomWalk rw = new RandomWalk();
        double root2 = Math.sqrt(2);
        PrivateMethodTester pmt = new PrivateMethodTester(rw);
        pmt.invokePrivate( name: "move",  ...parameters: 1, 1);
        assertEquals(root2, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: 1, 1);
        assertEquals( expected: 2 * root2, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: 0, -2);
        assertEquals( expected: 2.0, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: -2, 0);
        assertEquals( expected: 0.0, rw.distance(),  delta: 1.0E-7);
    }

    /**
     *
     */
    @Test
    public void testRandomWalk() {
        for (int i = 0; i < 5000; i++)
            assertEquals( expected: 10, RandomWalk.randomWalkMulti( m: 100,  n: 1
    }

    @Test
    public void testRandomWalk2() {
        for (int i = 0; i < 5000; i++)
            assertNotSame( unexpected: 0, RandomWalk.randomWalkMulti( m: 1,  n: 1)
    }
```

Run: RandomWalkTest

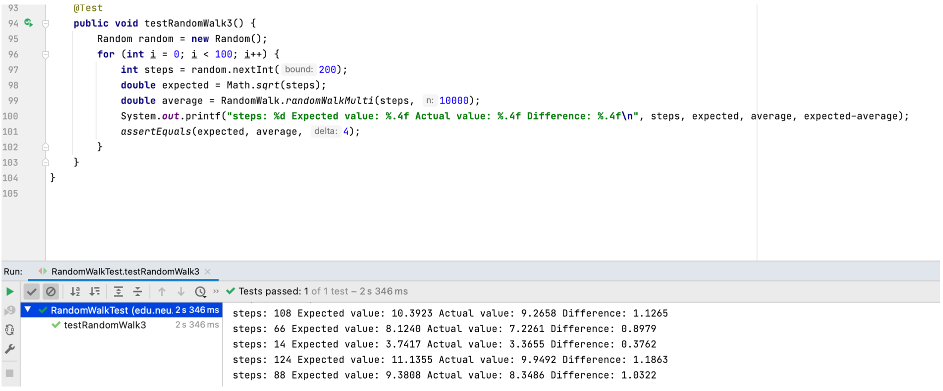
Tests passed: 6 of 6 tests – 1s 192 ms

```
RandomWalkTest (edu.neu.( 1s 192 ms
    testRandomWalk2     148 ms
    testMove0             2 ms
    testMove1             2 ms
    testMove2             3 ms
    testMove3             2 ms
    testRandomWalk    1s 35 ms
```

```
"/Applications/IntelliJ IDEA.app/Contents/jbr/Contents/Home/bin/java" ...

Process finished with exit code 0
```

Tests passed: 6 (a minute ago)

My Test



```
    @Test
    public void testRandomWalk3() {
        Random random = new Random();
        for (int i = 0; i < 100; i++) {
            int steps = random.nextInt( bound: 200);
            double expected = Math.sqrt(steps);
            double average = RandomWalk.randomWalkMulti(steps,  n: 10000);
            System.out.printf("steps: %d Expected value: %.4f Actual value: %.4f Difference: %.4f\n", steps, expected, average, expected-average);
            assertEquals(expected, average,  delta: 4);
        }
    }
}
```

Run: RandomWalkTest.testRandomWalk3

Tests passed: 1 of 1 test – 2 s 346 ms

```
RandomWalkTest (edu.neu. 2 s 346 ms
    testRandomWalk3      2 s 346 ms
```

```
steps: 108 Expected value: 10.3923 Actual value: 9.2658 Difference: 1.1265
steps: 66 Expected value: 8.1240 Actual value: 7.2261 Difference: 0.8979
steps: 14 Expected value: 3.7417 Actual value: 3.3655 Difference: 0.3762
steps: 124 Expected value: 11.1355 Actual value: 9.9492 Difference: 1.1863
steps: 88 Expected value: 9.3808 Actual value: 8.3486 Difference: 1.0322
```