

Program Structures and Algorithms

Spring 2024

Name : Shaofan Wei

NUID: 002815198

GITHUB LINK: <https://github.com/Arthurccone123/info-6205/tree/main/RandomWalk>

Task:

This task will implement and analyze random walk experiments, in which a drunk moves a specific number of steps in a two-dimensional space in a randomly selected direction, and finally calculates its Euclidean distance from the starting point. The relationship between the number of steps m and the distance d is derived by running experiments with different steps.

Relationship Conclusion:

In the random walk experiment, the average Euclidean distance d between the drunk and the starting point after moving m steps randomly on a two-dimensional plane increases with the increase of the number of steps m . But this increase is not linear, but increases with the square root of the number of steps.

Evidence to support that conclusion:

Here is the code:

```
* Copyright (c) 2017. Phasmid Software
*/

package edu.neu.coe.info6205.randomwalk;

import java.util.Random;

public class Randomwalk {

    private int x = 0;
    private int y = 0;

    private final Random random = new Random();

    /**
     * Private method to move the current position, that's to say the drunkard
     moves
     *
     * @param dx the distance he moves in the x direction
     * @param dy the distance he moves in the y direction
     */
    private void move(int dx, int dy) {
```

```

        x += dx;
        y += dy;
    }

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

    /**
     * Private method to generate a random move according to the rules of the
    situation.
     * That's to say, moves can be (+-1, 0) or (0, +-1).
     */
    private void randomMove() {
        boolean ns = random.nextBoolean();
        int step = random.nextBoolean() ? 1 : -1;
        move(ns ? step : 0, ns ? 0 : step);
    }

    /**
     * Method to compute the distance from the origin (the lamp-post where the
    drunkard starts) to his current position.
     *
     * @return the (Euclidean) distance from the origin to the current position.
     */
    public double distance() {
        return Math.sqrt(x * x + y * y);
    }

    /**
     * Perform multiple random walk experiments, returning the mean distance.
     *
     * @param m the number of steps for each experiment
     * @param n the number of experiments to run
     * @return the mean distance
     */
    public static double randomWalkMulti(int m, int n) {
        double totalDistance = 0;
        for (int i = 0; i < n; i++) {
            RandomWalk walk = new RandomWalk();
            walk.randomWalk(m);
            totalDistance = totalDistance + walk.distance();
        }
        return totalDistance / n;
    }

    // all unit test

```

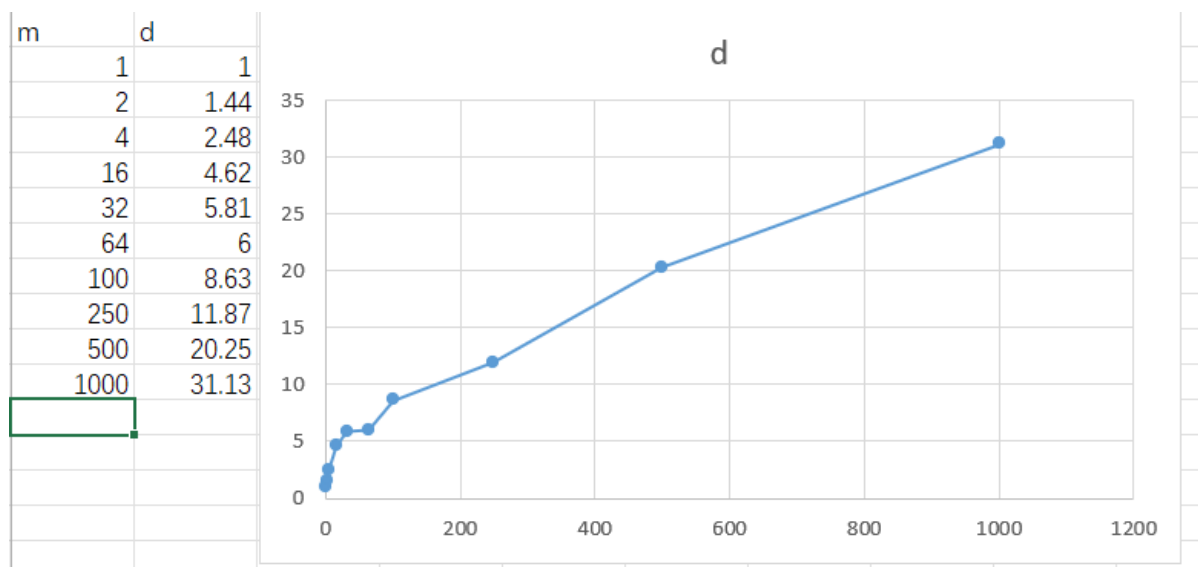
```

/* public static void main(String[] args) {
    if (args.length == 0)
        throw new RuntimeException("Syntax: RandomWalk steps [experiments]");
    int m = Integer.parseInt(args[0]);
    int n = 30;
    if (args.length > 1) n = Integer.parseInt(args[1]);
    double meanDistance = randomWalkMulti(m, n);
    System.out.println(m + " steps: " + meanDistance + " over " + n + "
experiments");
}*/

// For particular assignment
public static void main(String[] args) {
    int[] mValues = {1, 2, 4, 16, 32, 64, 100, 250, 500, 1000};
    int n = 10;
    for (int m : mValues) {
        double totalDistance = 0;
        for (int i = 0; i < n; i++) {
            double distance = randomWalkMulti(m, 1);
            totalDistance += distance;
        }
        double meanDistance = totalDistance / n;
        System.out.println(m + " steps: " + meanDistance + " over " + n + "
experiments");
    }
}
}

```

Below is a scatterplot of the results of 10 runs:



```

87 // For particular assignment
88 public static void main(String[] args) {
89     int[] mValues = {1, 2, 4, 16, 32, 64, 100, 250, 500, 1000};
90     int n = 10;
91     for (int m : mValues) {
92         double totalDistance = 0;
93         for (int i = 0; i < n; i++) {
94             double distance = randomWalkMulti(m, 1);

```

Console ×

<terminated> RandomWalk [Java Application] C:\Users\Arthur\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64.jre\bin\java.exe

```

1 steps: 1.0 over 10 experiments
2 steps: 1.4485281374238572 over 10 experiments
4 steps: 2.488245611270737 over 10 experiments
16 steps: 4.625454192648752 over 10 experiments
32 steps: 5.8169475976099205 over 10 experiments
64 steps: 6.001063767754602 over 10 experiments
100 steps: 8.63084971284923 over 10 experiments
250 steps: 11.87838109899681 over 10 experiments
500 steps: 20.255398550270577 over 10 experiments
1000 steps: 31.135377234642554 over 10 experiments

```

Unit Test Screenshots:

Package Explorer ×

inited after 0.171 seconds

Runs: 6/6 Errors: 0 Failures: 0

edu.neu.coe.info6205.randomwalk.RandomWalkTest (Runn...

- testRandomWalk2 (0.000 s)
- testMove0 (0.001 s)
- testMove1 (0.001 s)
- testMove2 (0.001 s)
- testMove3 (0.001 s)
- testRandomWalk (0.153 s)

Failure Trace

RandomWalk.java

```

48 assertEquals(1.0, rw.distance(), 1.0E-7);
49 pmt.invokePrivate("move", 0, 1);
50 assertEquals(2.0, rw.distance(), 1.0E-7);
51 pmt.invokePrivate("move", 0, -1);
52 assertEquals(1.0, rw.distance(), 1.0E-7);
53 pmt.invokePrivate("move", 0, -1);
54 assertEquals(0.0, rw.distance(), 1.0E-7);
55 }
56
57 /**
58 *
59 */
60 @Test
61 public void testMove3() {
62     RandomWalk rw = new RandomWalk();
63     double root2 = Math.sqrt(2);
64     PrivateMethodTester pmt = new PrivateMethodTester(rw);
65     pmt.invokePrivate("move", 1, 1);
66     assertEquals(root2, rw.distance(), 1.0E-7);
67     pmt.invokePrivate("move", 1, 1);
68     assertEquals(2 * root2, rw.distance(), 1.0E-7);
69     pmt.invokePrivate("move", 0, -2);
70     assertEquals(2.0, rw.distance(), 1.0E-7);
71     pmt.invokePrivate("move", -2, 0);
72     assertEquals(0.0, rw.distance(), 1.0E-7);
73 }
74
75 /**
76 *
77 */
78 @Test // slow
79 public void testRandomWalk() {
80     for (int i = 0; i < 1000; i++)

```

Console ×