Syed Hussain

Project\_1

Empl# 23466869

Goal: My goal is to create a self-avoiding random walk. A self-avoiding random walk is when a random walk passes through a point, it cannot pass through the same point again. In my project, I am using pseudorandom numbers to generate a sample set of self-avoiding random walks for two, three and four dimensions. From those walks, I am ignoring all the cases where the walks are not self-avoiding or reaching a point from where they cannot move any further, resulting in a dead end. Also, I am using multi-threading to perform calculations in parallel. Then, I am taking all the completed steps value and calculating R squared to understand the length of the random walk. Then I will divide R squared by the number of total self-avoiding walks to get the mean of R squared to find out the average distance. Finally, I will calculate F\_SAW by dividing the number of total self-avoiding walks with total number of threads used and total number of walks.

Software design:

In Main Class I have main method that drives the program and Print method that prints the output. From Main I am passing two values to dimensions class. One is the number of steps and another is the number of dimensions.

In Dimensions class I Dimensions constructor, run method, walk2D method, walk3D method, walk4D method. From Dimensions Class I am calling path Class and passing default coordinates to addToPath method which is in Path Class.

In Path Class I have Path constructor, addToPath method, addNext2D method, addNext3D method, addNext4D method, isDead2D method, isDead3D method, isDead4D method, getRSquare2D, getRSquared3D, getRSquared4D.

I also have a Coordinate Class and a Globals Class. In Coordinate Class, I have all my getters and coordinate defined and in Globals class I have some global variables defined.

I have 5 java files. (Coordinates, Dimensions, Globals, Main, Path)

Main file runs the program.

Optimizations:

Implementation of efficient algorithms and data structures are key to write a fast and good program. I believe my program have a good algorithm that is making my program fast. One big mistake in my program is of similar codes repeating. It is bad programming and my next step would have been removing the duplicate code and reusing the code. I could have used array for my coordinate which would have further reduced the line of repeated codes. Removing duplicate codes will reduce bugs in the program and will give better results.

For algorithms, this primarily consists of ensuring that algorithms are constant O(1), logarithmic O(log n), linear O(n), or in some cases log-linear O(n log n) in the input (both in space and time). Algorithms with quadratic complexity O(n2) fail to scale, and even linear algorithms cause problems if repeatedly called, and are typically replaced with constant or logarithmic if possible.

Two Dimensional

|  |  |  |
| --- | --- | --- |
| n | R^2 | F\_SAW |
| 10 | 26.20691 | 0.042085 |
| 11 | 30.02388 | 0.028659 |
| 12 | 34.21355 | 0.019393 |
| 13 | 38.29825 | 0.013161 |
| 14 | 42.68716 | 0.008861 |
| 15 | 47.18456 | 0.005949 |
| 16 | 52.03342 | 0.004028 |
| 17 | 57.10005 | 0.002703 |
| 18 | 61.87298 | 0.001819 |
| 19 | 67.13193 | 0.001219 |
| 20 | 72.14191 | 8.19E-04 |
| 21 | 76.88435 | 5.47E-04 |
| 22 | 83.77957 | 3.85E-04 |
| 23 | 89.54726 | 2.41E-04 |
| 24 | 92.14773 | 1.61E-04 |
| 25 | 100.4706 | 1.09E-04 |
| 26 | 106.7231 | 6.97E-05 |
| 27 | 109.5517 | 4.93E-05 |
| 28 | 119.7888 | 3.03E-05 |
| 29 | 126.4561 | 2.28E-05 |
| 30 | 130.4429 | 1.40E-05 |
| 31 | 149.7312 | 9.30E-06 |
| 32 | 112.2857 | 5.60E-06 |
| 33 | 163.902 | 5.10E-06 |
| 34 | 165.1333 | 3.00E-06 |
| 35 | 168.7895 | 1.90E-06 |
| 36 | 185.3333 | 1.20E-06 |
| 37 | 185.4444 | 9.00E-07 |
| 38 | 170 | 2.00E-07 |
| 39 | 189 | 2.00E-07 |
| 40 | 104 | 4.00E-07 |

α = 0.0603

β = 0.408

Three Dimensional

|  |  |  |
| --- | --- | --- |
| n | R^2 | F\_SAW |
| 10 | 16.8276 | 0.145808 |
| 11 | 18.85205 | 0.115611 |
| 12 | 20.95406 | 0.09143 |
| 13 | 23.07452 | 0.072276 |
| 14 | 25.25219 | 0.056918 |
| 15 | 27.38282 | 0.045073 |
| 16 | 29.62531 | 0.035486 |
| 17 | 31.82059 | 0.027938 |
| 18 | 34.06729 | 0.022035 |
| 19 | 36.44539 | 0.017345 |
| 20 | 38.69006 | 0.013687 |
| 21 | 41.03648 | 0.010756 |
| 22 | 43.51249 | 0.008462 |
| 23 | 45.87739 | 0.006625 |
| 24 | 48.2799 | 0.005221 |
| 25 | 50.71784 | 0.004057 |
| 26 | 52.77243 | 0.003243 |
| 27 | 55.44801 | 0.00253 |
| 28 | 58.359 | 0.001981 |
| 29 | 60.31135 | 0.001547 |
| 30 | 63.54414 | 0.00124 |
| 31 | 65.66605 | 9.67E-04 |
| 32 | 69.02984 | 7.61E-04 |
| 33 | 71.66952 | 5.85E-04 |
| 34 | 72.53782 | 4.64E-04 |
| 35 | 74.77035 | 3.65E-04 |
| 36 | 80.01596 | 2.88E-04 |
| 37 | 80.78914 | 2.21E-04 |
| 38 | 85.43083 | 1.77E-04 |
| 39 | 87.37427 | 1.37E-04 |
| 40 | 90.40778 | 1.08E-04 |

α = 0.0529

β = 0.241

Four Dimensional

|  |  |  |
| --- | --- | --- |
| n | R^2 | F\_SAW |
| 10 | 14.01829 | 0.257477 |
| 11 | 15.57291 | 0.219709 |
| 12 | 17.15298 | 0.187045 |
| 13 | 18.73624 | 0.158987 |
| 14 | 20.32899 | 0.135322 |
| 15 | 21.91085 | 0.114945 |
| 16 | 23.54017 | 0.097721 |
| 17 | 25.14724 | 0.083091 |
| 18 | 26.74946 | 0.070457 |
| 19 | 28.38921 | 0.059829 |
| 20 | 30.05698 | 0.050802 |
| 21 | 31.69001 | 0.043154 |
| 22 | 33.35801 | 0.036762 |
| 23 | 34.99038 | 0.031199 |
| 24 | 36.7197 | 0.026421 |
| 25 | 38.38728 | 0.022452 |
| 26 | 40.01031 | 0.019179 |
| 27 | 41.68468 | 0.016157 |
| 28 | 43.37245 | 0.013674 |
| 29 | 45.18504 | 0.011568 |
| 30 | 46.69797 | 0.009865 |
| 31 | 48.25878 | 0.008327 |
| 32 | 49.96779 | 0.007097 |
| 33 | 51.90442 | 0.006001 |
| 34 | 53.51257 | 0.005092 |
| 35 | 55.16112 | 0.004302 |
| 36 | 56.94363 | 0.003656 |
| 37 | 58.5418 | 0.003106 |
| 38 | 60.26736 | 0.002637 |
| 39 | 62.04889 | 0.002244 |
| 40 | 63.91716 | 0.001869 |

α = 0.0477

β = 0.164