Daniel C. Landon Jr.

Program # 2

Markov Chain

February 28, 2020

***Abstract***

Given a single transaction matrix the program will take this matrix and compute the values of a Markov chain using two separate algorithms. From what I can gather the purpose is to demonstrate that the second algorithm with the Fibonacci sequence is faster and a better option to use.

The algorithms are basically the same as listed below. The difference between the two will be indicated when that step is reached.

* Initialize both transaction matrix
* Initialize results matrix and supporting variables for time tracking etc.
* Iterate and process the matrix based on iteration counter
  + Multiply the first transaction matrix by the second to get current SX
  + ALGORITHMS DIFFERENCE
    - For the second algorithm we are, from what I can understand using a Fibonacci sequence to increment the state so 1,1,2,3,5,8,13 etc. I feel that I do not understand what was really required here. But this is my best guess.
  + Copy results of above math to second transaction matrix so SX + 1 can be calculated
* Print the results matrix
* Calculate run time and display

The code for this program is rough and “nasty”. The Fibonacci sequence portion of algorithm 2 is sloppy and weak in my opinion and not even deserving of a junior level programmer. This part of the code bothers me to no end and requires that I come back to it once I have time to resolve the way it is. It can be done better.

I have ran this multiple times and for the most part the second algorithm is faster. While both will complete in under 150ms there are instances where the second algorithm completes in less than 10 for the same number of iterations ran. This is not always the case and I am uncertain as to why this is.

***App.java***

package app;

import java.text.DecimalFormat;

import java.time.Duration;

import java.time.Instant;

import java.util.Arrays;

/\*\*

 \* There is unfortunately a lot of duplicated code in this program. I wanted to work out how to use generics for the matrix math. Had trouble resolving the problem that java has no idea what generics are at compile time so you cannot use math against them without some funky moves...ill figure this one out.

 \*/

public class App {

    /\*\*

     \*

     \* @param args command line not used

     \* @throws Exception errors

     \*/

    public static void main(String[] args) throws Exception {

        // final int A1\_ITERATION = 50;

        // variables

        double[][] \_mOne = {

            {0.90, 0.05, 0.05},

            {0.05, 0.90, 0.05},

            {0.05, 0.05, 0.90} };

        double[][] \_mTwo = {

            {0.90, 0.05, 0.05},

            {0.05, 0.90, 0.05},

            {0.05, 0.05, 0.90} };

        // time trap

        Instant \_startTime = null;

        Instant \_endTime = null;

        Duration \_timeElapsed = null;

        Algorithim\_One(\_mOne, \_mTwo, \_startTime, \_endTime, \_timeElapsed);

        Algorithim\_Two(\_mOne, \_mTwo, \_startTime, \_endTime, \_timeElapsed);

    } // end main

    /\*\*

     \*

     \* @param \_a1 first array to process

     \* @param \_a2 second array to process

     \* @param \_sTime start time, just passing so I do not have to declare more than once

     \* @param \_eTime end time, just passing so I do not have to declare more than once

     \* @param \_tElapsed time elapsed, just passing so I do not have to declare more than once

     \*/

    public static void Algorithim\_Two (double[][] \_a1, double[][] \_a2,

        Instant \_sTime, Instant \_eTime, Duration \_tElapsed) {

        final int A2\_ITERATION = 50;

        double[][] \_mResults = new double[\_a1.length][\_a2.length];

        System.out.println("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Algorithm 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

        // start time

        \_sTime = Instant.now();

        System.out.println("\nSTART TIME: " + \_sTime + "\n");

        System.out.println("Iterations: " + A2\_ITERATION + "\n");

        // variables to do slopy math for fibo sequence

        int \_mCount1 = 1;

        int \_mCount2 = 1;

        int \_mTotal = 0;

        // loop the math

        while (\_mTotal <= A2\_ITERATION){

            if(\_mCount1 + \_mCount2 > A2\_ITERATION) {

                // loop the matrix

                for (int \_loopOutter = 0; \_loopOutter < \_a1.length; \_loopOutter++){

                    for (int \_loopInner = 0; \_loopInner < \_a2.length; \_loopInner ++){

                        // clear the matrix index

                        \_mResults[\_loopOutter][\_loopInner] = 0;

                        for(int \_k = 0; \_k < \_a2.length; \_k++){

                            // funcky math

                            \_mResults[\_loopOutter][\_loopInner]

                                += \_a1[\_loopOutter][\_k]

                                \* \_a2[\_k][\_loopInner];

                        } // \_k

                    } // end \_loopInner

                } // end \_loopOutter

                // loop and change the first matrix

                for (int \_loopOutter = 0; \_loopOutter< \_a1.length; \_loopOutter++){

                    for(int \_k = 0; \_k < \_a1.length; \_k++){

                        \_a1[\_loopOutter][\_k] =\_mResults[\_loopOutter][\_k];

                    } // \_k

                } // end \_loopOutter

                \_mTotal++;

            }

            else{

                System.out.println(\_mCount1 + " " + \_mCount2 + " " + \_mTotal);

                // loop the array again for the fibo sequence stuff

                for (int \_loopOutter = 0;\_loopOutter < \_a1.length;\_loopOutter++){

                    for (int \_loopInner = 0; \_loopInner< \_a2.length; \_loopInner++){

                        \_mResults[\_loopOutter][\_loopInner] = 0;

                        for(int \_k = 0; \_k < \_a2.length; \_k++){

                            // more funky matrix math

                            \_mResults[\_loopOutter][\_loopInner]

                                += \_a1[\_loopOutter][\_k]

                                \* \_a1[\_k][\_loopInner];

                        } // \_k

                    } // end \_loopInner

                } // end \_loopOutter

                // copy \_a2 into \_a1

                \_a1 = Arrays.stream(\_a2).map(\_tempItem ->  Arrays.copyOf(\_tempItem, \_tempItem.length)).toArray(double[][]::new);

                // copy \_mResults into \_a2

                \_a2 = Arrays.stream(\_mResults).map(\_tempItem ->  Arrays.copyOf(\_tempItem, \_tempItem.length)).toArray(double[][]::new);

                // math clean-up

                \_mTotal = \_mCount1 + \_mCount2;

                \_mCount1 = \_mCount2;

                \_mCount2 = \_mTotal;

            }

        } // end while

        System.out.println(PrintMatrix(\_mResults));

        // end time

        \_eTime = Instant.now();

        // differrence in start to end time

        \_tElapsed = Duration.between(\_sTime, \_eTime);

        System.out.println("\nEND TIME: " + \_eTime);

        System.out.println("\nTime for completion (milliseconds): " + \_tElapsed.toMillis() + "\n");

        System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Algorithm 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

    } // end Algorithim\_Two

    /\*\*

     \*

     \* @param \_a1 first array to process

     \* @param \_a2 second array to process

     \* @param \_sTime start time, just passing so I do not have to declare more than once

     \* @param \_eTime end time, just passing so I do not have to declare more than once

     \* @param \_tElapsed time elapsed, just passing so I do not have to declare more than once

     \*/

    public static void Algorithim\_One(double[][] \_a1, double[][] \_a2,

        Instant \_sTime, Instant \_eTime, Duration \_tElapsed) {

        final int A1\_ITERATION = 50;

        double[][] \_mResults = new double[\_a1.length][\_a2.length];

        System.out.println("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Algorithm 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

        // start time

        \_sTime = Instant.now();

        System.out.println("\nSTART TIME: " + \_sTime + "\n");

        System.out.println("Iterations: " + A1\_ITERATION + "\n");

        // loop iteration

        for (int \_iteration = 1; \_iteration <= A1\_ITERATION; \_iteration++){

            // loop outter index

            for (int \_loopOutter = 0; \_loopOutter < \_a1.length; \_loopOutter++){

                // loop inner index

                for (int \_loopInner = 0; \_loopInner< \_a2.length; \_loopInner++){

                    // make sure the array index is empty

                    \_mResults[\_loopOutter][\_loopInner] = 0;

                    // loop the math

                    for(int \_k = 0; \_k < \_a2.length; \_k++){

                        // set array value

                        \_mResults[\_loopOutter][\_loopInner]

                            += \_a1[\_loopOutter][\_k]

                            \* \_a2[\_k][\_loopInner];

                    } // \_k

                } // end \_loopInner

            } // end \_loopOutter

            // copy array

            \_a2 = Arrays.stream(\_mResults).map(\_tempItem ->  Arrays.copyOf(\_tempItem, \_tempItem.length)).toArray(double[][]::new);

        } // end \_iteration

        System.out.println(PrintMatrix(\_mResults));

        // end time

        \_eTime = Instant.now();

        // differrence in start to end time

        \_tElapsed = Duration.between(\_sTime, \_eTime);

        System.out.println("\nEND TIME: " + \_eTime);

        System.out.println("\nTime for completion (milliseconds): " + \_tElapsed.toMillis() + "\n");

        System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Algorithm 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

    } // end Algorithim\_One

    /\*\*

     \*

     \* @param \_aMtraix matrix to print

     \* @return formatted matrix ready to print

     \*/

    public static String PrintMatrix(double[][] \_aMtraix){

        // varaibles

        String \_ans = "";

        DecimalFormat \_decimalFormat = new DecimalFormat("#.###");

        // loop row

        for(int \_loopRow = 0; \_loopRow < \_aMtraix.length; \_loopRow++){

            // loop col

            for(int \_loopCol = 0; \_loopCol < \_aMtraix.length; \_loopCol++){

                // create return for matrix

                \_ans+= \_decimalFormat.format(\_aMtraix[\_loopRow][\_loopCol]) + "\t";

            } // end for \_loopCol

            // create final output

            \_ans+= "\n";

        } // end for \_loopRow

        return \_ans;

    } // end toString

} // end App

***Console Output***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Algorithm 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

START TIME: 2020-02-28T23:03:01.161Z

Iterations: 50

0.334 0.333 0.333

0.333 0.334 0.333

0.333 0.333 0.334

END TIME: 2020-02-28T23:03:01.257Z

Time for completion (milliseconds): 96

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Algorithm 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Algorithm 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

START TIME: 2020-02-28T23:03:01.260Z

Iterations: 50

1 1 0

1 2 2

2 3 3

3 5 5

5 8 8

8 13 13

13 21 21

0.333 0.333 0.333

0.333 0.333 0.333

0.333 0.333 0.333

END TIME: 2020-02-28T23:03:01.264Z

Time for completion (milliseconds): 4

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Algorithm 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*