Introduction to Computer Programming (CS 1323)
Project 7

Instructor: Andrew H. Fagg

November 3, 2014

Introduction

In this project, we will model the transmission of the Ebola virus through a population of individuals. The progression through different disease states by an individual will be modeled essentially as a Markov Chain. There are two key differences, however: 1) we will model the number of individuals in each state, and not explicitly represent individuals, and 2) the probability of transition from a healthy state to a carrier state will be variable, depending on the number of individuals who are contagious.

Please note that this is a very abstract exercise that is intended to not only develop your programming skills, but also to show how these types of *Monte Carlo* models can be (and are being) used to understand and defend against the spread of disease. While some of the parameters that we choose for our model have some basis in reality, others are wild guesses that are "pulled out of the air" (in part, to make a point about how the model can behave). The results that you will see during the course of this project do not reflect reality.

Objectives

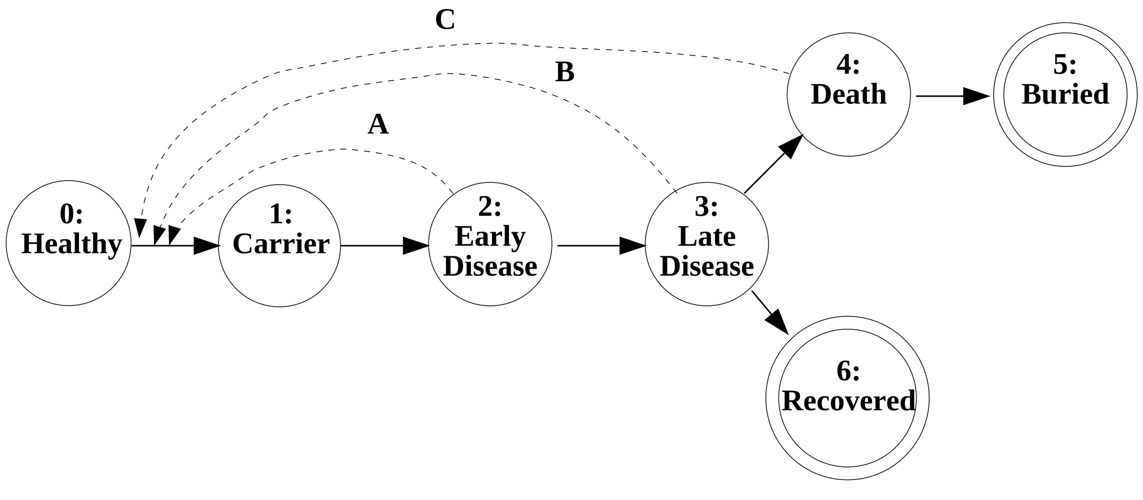
By the end of this project, you will be able to:

1. create and manipulate **primitive arrays** of ints and doubles,
2. use a random number generator to produce random samples, and
3. write a loop that iterates through a simulation until appropriate criteria are met.

Ebola Virus Transmission Model

Our model is shown below. There are a total of seven states:

- Healthy: no virus present in system
- Carrier: virus present in system, but no symptoms showing (and not contagious)
- Early Disease: symptoms have begun to show and contagious. We assume that isolation has not occurred.
- Late Disease: significant symptoms showing, contagious and isolated.
- Death: self explanatory (but still contagious)
- Buried: no longer contagious
- Recovered: immune and not contagious



Solid lines represent probabilistic transitions from one state to another. The probability of transition from one state to another is constant (and will be provided as an input to your program), except for the transition from the Healthy to the Carrier states, which depends on the number of contagious individuals in the Early Disease, Late Diseased, or Death states.

In our model, we will represent the number of individuals in each of the states using an array of ints. The initial state will be $1,000,000$ in the Healthy state and 1 in the Carrier state. There are zero individuals in each of the remaining states. The model will make a transition once per day, based on the probabilities of transition. Like the Markov Chain from project 6, we will use a random number generator to decided whether an individual makes the transition from one state to the next. The difference here is that there may be multiple individuals in the Carrier state. We will treat each one separately as a Markov Chain transition. In the detailed specification, we have included an implementation of a method **sampleN()** that is based on your **sample()** method. This method takes as input the number of individuals that are in a given state and the probability of transition for each one, and returns the number of these individuals that have actually transitioned (with the rest staying in the current state). So, the number of individuals that transition from Carrier to the Early Disease state will be:

```
    delta[T_CARRIER_EARLY_DISEASE] = sampleN(rand, prob[P_CARRIER_EARLY_DISEASE],
                                             N[S_CARRIER]);
```

where the array index constants are defined below, **delta** is an array of changes from one state to another, **prob** is an array of transition probabilities, and **N** is an array that contains the number of individuals in each state (for the current day).

For the transition from Healthy to Carrier, we will introduce the concept of a *sphere*: this is the set of individuals that have contact with a contagious individual. We will assume that the sphere is of size:

- Early Disease: 50
- Late Disease: 10
- Death: 10

and that each contagious individual has their own, independent sphere. So, for a given day, the number of individuals that transition from the Healthy to the Carrier state is computed as follows:

```
    delta[T_HEALTHY_CARRIER] =
      sampleN(rand, prob[P_HEALTHY_CARRIER_BY_EARLY_DISEASE],
                    sphere[SP_EARLY_DISEASE]*N[S_EARLY_DISEASE])
      + sampleN(rand, prob[P_HEALTHY_CARRIER_BY_LATE_DISEASE],
                    sphere[SP_LATE_DISEASE]*N[S_LATE_DISEASE])
      + sampleN(rand, prob[P_HEALTHY_CARRIER_BY_DEATH],
                    sphere[SP_DEATH]*N[S_DEATH]);

    if(delta[T_HEALTHY_CARRIER] > N[S_HEALTHY])
      delta[T_HEALTHY_CARRIER] = N[S_HEALTHY];
```

where the last two lines of code ensure that more individuals do not make the transition than there are Healthy individuals.

We must also handle the transition out of the Late Disease state in a special manner so that we do not have more individuals transitioning out of the Late Disease state than there are individuals starting in that state. For simplicity, we will model this as follows:

```
  delta[T_LATE_DISEASE_DEATH] = sampleN(rand, prob[P_LATE_DISEASE_DEATH],
                                    N[S_LATE_DISEASE]);
  delta[T_LATE_DISEASE_RECOVERED] = sampleN(rand, prob[P_LATE_DISEASE_RECOVERED],
                                    N[S_LATE_DISEASE] - delta[T_LATE_DISEASE_DEATH]);
```

User Interaction

The input to your program will be:

1. A non-negative integer that will be used as a seed for our random number generator.
2. The doubles that represent the eight transition probabilities (defined below).

The program will then respond by simulating a sequence of days. The last simulated day will be the first day with no individuals in a disease carrying state (no individuals in the Carrier, Early Disease, Late Disease or Death states).

Your program will print out a series of lines, each line representing the state for a single day (described below). This information will be reported for day 0, then every 10 days during the simulation, and finally the last day (so: day 0, 10, 20 ... last).

Limitations

Limitations to this model include:

- We make many guesses at important model parameters
- There is no notion of location of individuals
- We do not take into account policy/behavioral changes that might affect transition probabilities as the spread of the disease progresses

(but these limitations are OK for our purposes)

Project Requirements

1. Add the following to your Project7 class definition (outside of all methods):

```
    // States (indices)
    public static final int S_HEALTHY = 0;
    public static final int S_CARRIER = 1;
    public static final int S_EARLY_DISEASE = 2;
    public static final int S_LATE_DISEASE = 3;
    public static final int S_DEATH = 4;
    public static final int S_BURIED = 5;
    public static final int S_RECOVERED = 6;
    public static final int S_TOTAL = 7;

    // Transitions (indices)
    public static final int T_HEALTHY_CARRIER = 0;
    public static final int T_CARRIER_EARLY_DISEASE = 1;
    public static final int T_EARLY_DISEASE_LATE_DISEASE = 2;
    public static final int T_LATE_DISEASE_DEATH = 3;
    public static final int T_DEATH_BURIED = 4;
    public static final int T_LATE_DISEASE_RECOVERED = 5;
    public static final int T_TOTAL = 6;

    // Probabilities (indices)
    public static final int P_HEALTHY_CARRIER_BY_EARLY_DISEASE = 0;
    public static final int P_HEALTHY_CARRIER_BY_LATE_DISEASE = 1;
    public static final int P_HEALTHY_CARRIER_BY_DEATH = 2;
    public static final int P_CARRIER_EARLY_DISEASE = 3;
    public static final int P_EARLY_DISEASE_LATE_DISEASE = 4;
    public static final int P_LATE_DISEASE_DEATH = 5;
    public static final int P_DEATH_BURIED = 6;
    public static final int P_LATE_DISEASE_RECOVERED = 7;
    public static final int P_TOTAL = 8;

    // Sphere sizes (indices)
    public static final int SP_EARLY_DISEASE = 0;
    public static final int SP_LATE_DISEASE = 1;
    public static final int SP_DEATH = 2;
    public static final int SP_TOTAL = 3;
```

These constants define the indices for four different types of arrays: states, transitions, probabilities and spheres. The naming of each constant clearly defines the meaning of the index.

2. Add the following to your Project7 class definition:

```
    /**
     * Return the number of individuals out of <n> that have transitioned to
     *   a new state.  Each individual transitions with probability <p>.
     *
     * @param rand Initialized random number generator
     * @param p Probability that a single individual transitions
     * @param n Number of individuals to consider
     * @return The number of individuals that have transitioned
     */

    public static int sampleN(Random rand, double p, int n){
        int count = 0;  // Number that have transitioned
        // Loop over each individual
        for(int i = 0; i < n; ++i) {
            count += sample(rand, p);
        }
        return count;
    }
```

3. Copy your **getNonNegativeInt()**, **sample()** and **getDoubleConstrained()** methods from project 6. **DO NOT MODIFY THESE METHODS EXCEPT TO FIX ANY BUGS** (they must adhere to the original specifications).

4. Write a method with this prototype:

```
    public static int sumArray(int[] array)
```

   This method returns the sum of the ints in the input array.

5. Write a method with this prototype:

```
    public static void displayState(int day, int[] N)
```

   This method displays a single line of the state output shown in the examples. The day is first displayed, followed by the ints of the input array N. The last number is the sum of all of the integers (which should be the same for each line). All numbers in the line are separated by the tab character ('\t'), which will make the resulting table easier to read.

6. Write a method with this prototype:

```
public static int[] singleStep(int[] N, Random rand, int[] sphere,
                                   double[] prob)
```

   This method takes as input the number of individuals in each state for the current day, a random number generator, a sphere for disease carriers and the probability of transition from each state. This method creates a new array to store the number of individuals in each state for the **next day.**

7. Write a main method that prompts the user for a valid seed and each of the eight transition probabilities. This method then simulates the progression of the disease, day by day (each day-to-day transition is implemented by calling **singleStep()**. This method calls **displayState()** on the first day of the simulation, every tenth day during the simulation and the last day (if the last day happens to be divisible by ten, then it is OK for the same line to be displayed twice).

Examples

Here are some example inputs and corresponding outputs from a properly executing program. You should carefully consider the cases with which you test your program. In particular, think about the key "boundary cases" (the cases that cause your code to do one thing or another, based on very small differences in input). User inputs are shown in bold.

Example 1

```
Enter a seed: 5
Enter probability 0: .001
Enter probability 1: .01
Enter probability 2: .001
Enter probability 3: .05
Enter probability 4: .14
Enter probability 5: .07
Enter probability 6: .33
Enter probability 7: .07
 Day   Healthy   Carrier   Early   Late   Death   Buried   Recover   Check
 0:    1000000   1         0       0      0       0        0         1000001
 10:   1000000   1         0       0      0       0        0         1000001
 20:   1000000   1         0       0      0       0        0         1000001
 30:   1000000   0         0       1      0       0        0         1000001
 40:   1000000   0         0       1      0       0        0         1000001
 41:   1000000   0         0       0      0       0        1         1000001
```

(program ends)

Example 2

```
Enter a seed: **6**
Enter probability 0: **.001**
Enter probability 1: **.01**
Enter probability 2: **.001**
Enter probability 3: **.05**
Enter probability 4: **.14**
Enter probability 5: **.07**
Enter probability 6: **.33**
Enter probability 7: **.07**
 Day   Healthy  Carrier  Early  Late  Death  Buried  Recover  Check
 0:    1000000  1        0      0     0      0       0        1000001
 10:   1000000  0        1      0     0      0       0        1000001
 20:   1000000  0        0      1     0      0       0        1000001
 21:   1000000  0        0      0     0      0       1        1000001
```

(program ends)

Example 3

```
Enter a seed: **done**
Enter a seed: **-5**
Only non-negative values are allowed: **7**
Enter probability 0: **done**
Enter probability 0: **5**
Value must be between 0.0 and 1.0: **-1**
Value must be between 0.0 and 1.0: **.001**
Enter probability 1: **.01**
Enter probability 2: **.01**
Enter probability 3: **.05**
Enter probability 4: **.14**
Enter probability 5: **.07**
Enter probability 6: **.33**
Enter probability 7: **.07**
 Day    Healthy  Carrier  Early  Late  Death  Buried  Recover  Check
 0:     1000000  1        0      0     0      0       0        1000001
 10:    1000000  1        0      0     0      0       0        1000001
 20:    1000000  1        0      0     0      0       0        1000001
 :      :        :        :      :     :      :       :        :
 1850:  0        3        3      2     1      518231  481761   1000001
 1860:  0        1        3      1     0      518233  481763   1000001
 1870:  0        1        0      2     0      518235  481763   1000001
 1880:  0        0        1      1     0      518236  481763   1000001
 1890:  0        0        0      0     0      518238  481763   1000001
 1890:  0        0        0      0     0      518238  481763   1000001

    (program ends)
```

Note: program generates many more lines (only the beginning and end are shown here)

Project Documentation

Follow the same documentation procedures as we used in project 1. In particular, you must include:

- Java source file documentation (top of file)
- Method-level documentation
- Inline documentation

For full credit, you must execute Javadoc on your source file and include the results (in the *doc* directory) in your zip file.

Hints

Implement and test your program one method at a time (we call this *incremental development*). Convince yourself that each method is working properly before you move on to the next one.

In this project specification, you have been given some example code for computing the number of number of individuals that transition from one state to another (for some states). As part of your implementation, you must: 1) fill in the remaining changes and 2) use these changes to compute the new number of individuals in each state.

The last column of your output is the sum of all individuals being simulated. This number should not change from day to day. If it is, then you have something wrong in your computation.

This project specification should give you enough information to reproduce the three examples exactly. If you are unable to do so, then there is likely an error (or at least a difference in what we intended). Check your work and then consult with the instructor or the TA.

This project is more complicated than previous projects. It is imperative that you start early on this (you will find it difficult to complete in the last couple of days before the deadline).

Project Submission

This project must be submitted no later than 2:00pm on Thursday, November 20th to the project 7 D2L dropbox. The file must be called *Project7.zip* and be generated in the same way as for prior projects.

Code Review

For the office hour sessions surrounding the project deadline, we will put together a "Doodle Poll" through which you can sign up for a 5-minute code review appointment. During this review, we will execute test examples and review the code with you. If you fail to show up for your reserved time, then you will forfeit your appointment and you must attend at a later time that is not already reserved by someone else.

If either the instructor or TA are free during office hours (or another mutually-agreed upon time), then we are happy to do code reviews, as well as provide general help.

We will only perform reviews on code that has already been submitted to the dropbox on D2L. Please prepare ahead of time for your code review by performing this submission step.

Code reviews must be completed no later than Tuesday, December 2nd. If you fail to do a code review, then you will receive a score of zero for the project.

Anyone receiving a 95 or greater on Project 6 may opt to do an "offline" code review (which does not require your presence). Send the instructor email once you have submitted your project in order to schedule an offline review.

Rubric

The project will be graded out of 100 points. The distribution is as follows:

Implementation: 35 points

Program formatting: 15 points

(15) The program is properly formatted (including indentation, curly brace and semicolon locations).

(8) There is one problem with program formatting.

(0) The program is not properly formatted.

Data types and method calls: 10 points

(10) The program is using proper data types and method calls.

(5) There is one error in data type or method call selection.

(0) There are multiple errors in data type and method call selection.

Required Methods: 10 points

(10) All of the required methods are implemented correctly.

(0) The required methods are not implemented correctly.

Proper Execution: 30 points

Output: 15 points

(15) The program passes all tests.

(12) The program fails one test.

(9) The program fails two tests.

(6) The program fails three tests.

(3) The program fails four tests.

(0) The program fails five or more tests.

Execution: 15 points

(15) The program executes with no errors (thrown exceptions).

(8) The program executes, but there is one minor error.

(0) The program does not execute.

Documentation and Submission: 35 points

Project Documentation: 5 points

(5) The java file contains all of the required documentation elements at the top of the file.

(3) The java file is missing one of the required documentation elements.

(2) The java file is missing two of the required documentation elements.

(0) The java file is missing more than two of the required documentation elements.

Method-Level Documentation: 10 points

(10) Every method contains all of the required documentation elements ahead of the method prototype, and the Javadocs have been generated and included in the submission.

(7) The method documentation is missing one of the required documentation elements.

(3) The method documentation is missing two of the required documentation elements, or the Javadocs have not been submitted.

(0) The method documentation is missing more than two of the required documentation elements.

Inline: 10 points

(10) Every method contains appropriate inline documentation.

(7) There is one missing or incorrect line of inline documentation.

(3) There are two missing or incorrect lines of inline documentation.

(0) There are more than two missing or incorrect lines of inline documentation.

Submission: 10 points

(10) The correct zip file name is used.

(0) An incorrect zip file name is used.

Bonus: 6 points For each unique and meaningful error in this project description that is reported to the instructor, a student will receive an extra two points.

References

- Simulating a global Ebola outbreak:
  http://community.wolfram.com/groups/-/m/t/326240?source=frontpage