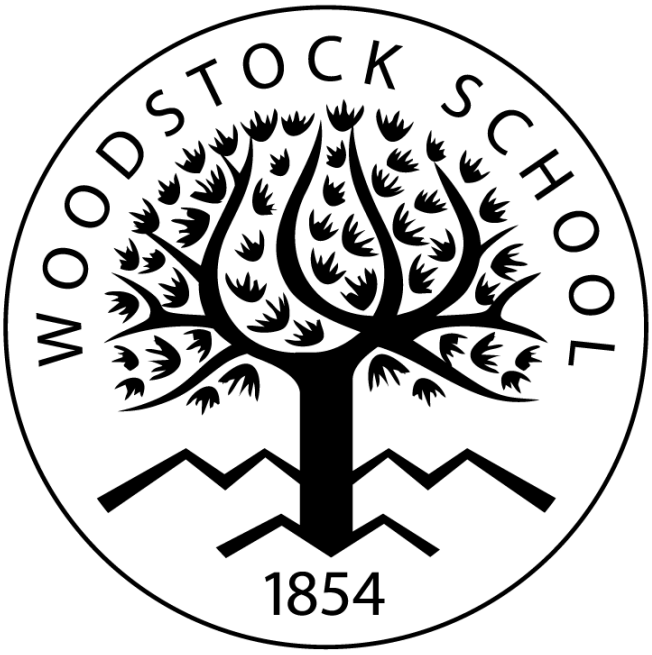
Simulation Lab

AP Computer Science A



Name: _____

Contents

Background

Computer scientists design simulations in order to test hypothesis, examine effects of different variables, and predict outcomes of different phenomena that are otherwise difficult to observe. They are employed in this capacity by a wide range of different fields and have become an integral part of many research studies.

In this lab, you will be creating, running, and examining a simulation expressed as a thought exercise for Computer Science. You will also consider how this simulation might be used to examine a variety of different applications.

Simulation: The Rumor

Imagine that Alice is throwing a party for n guests, including Bob. Bob starts a rumour about Alice by telling it to one of the other guests. A person hearing this rumour for the first time will immediately tell it to one other guest, chosen at random from all the people at the party except Alice and the person from whom they heard it. If a person (including Bob) hears the rumour for a second time, he or she will not propagate it further.

Examples

Guests: Bob, John, Judy, and Erica

When Bob arrives at the party, he tells John the rumour about Alice. John then tells Erica, who prompty tells Bob again. Since Bob already knows the rumour (having started it himself!), he stops spreading the rumour. Because the rumour stops propagating, Judy never gets to hear the juicy gossip about Alice.

Guests: Bob, Larry, Lucinda, Lucille, Lilly, Logan, Liam, and Lee

When Bob arrives at the party, he tells Lucille about Alice. Lucille tells Lilly, who then tells Lucinda. Lucinda decides to tell Lucille again, who stops propagating the rumour. Larry, Logan, Liam, and Lee have never heard the rumour!

Applications

Question #1: After Bob has told the rumour to a new guest, how many guests can that person select from to tell the rumour to? Express your answer in terms of n, the total number of guests at the party.

Question #2: What is the probability of the rumour continuing after the m^{th} person ($m \leq n$) has heard it? Express your answer in terms of n and m.

Question #3: Why is the next guest to hear the rumour chosen at random? What real-world considerations might be covered by "randomness" in this case?

Question #4: What assumptions are being made by the fact that the rumour stops propagating after a person has heard it for the second time? Why is this condition necessary for the purpose of this simulation?

Activity #1

Introduction

In this activity, you will create a Java program simulation the situation presented in the background. In particular, your simulation will calculate the number of guests who have heard the rumour about Alice before it stops propagating.

Exercises

1. Implement the method, `runSimulation()`, that will accept `n`, the number of guests (including Bob) invited to Alice's party, run the simulation once according to the details described in the background, and return the number of guests that have heard the rumour before it stops propagating.
2. Implement an overloaded `runSimulation()` method that will additionally accept `N`, the number of times to run the simulation. Your method should return an array representing the number of guests that have heard the rumour before it stops propagating after running the simulation each of `N` times.

Questions

Question #5: Briefly explain how you kept track of who in the party had already heard the rumour.

Question #6: Briefly explain how you generated a random guest to tell the rumour to while excluding who was spreading the rumour as well as the guest he/she heard the rumour from.

Activity #2

Introduction

In this activity, you will be using your simulation in order to estimate statistical and probabilistic values about the scenario presented. When run repeatedly, simulations can be used to search for average or expected values, as well as testing for the probability of rare (hard to observe) situations.

Exercises

1. Create the method, `estimateProbability()`, that will accept `n`, the number of guests (including Bob) invited to Alice's party, and run the simulation in order to return an estimation for the probability that *at least* `r` guests will hear the rumour before it stops propagating.
   Note: You will have to decide how many times to run the simulation in order to provide a good estimate of this probability.
2. Create the method, `estimateExpectedValue()`, that will accept `n`, the number of guests (including Bob) invited to Alice's party, and run the simulation in order to return the average number of people to hear the rumour.
   Note: You will have to decide how many times to run the simulation in order to provide a good estimate of this average.

Questions

Question #7: Briefly explain how you chose the number of times to run the simulation for each `estimateProbability()` and `estimateExpectedValue()`.

Question #8: How does varying the value of `n` affect the values returned by `estimateProbability()` and `estimateExpectedValue()`. Discuss trends rather than specific values.

Final Analysis

Question #9: Although this particular simulation seems contrived, it can be modified to apply to the spread of computer viruses and human disease. Do you feel that considering contrived applications of simulations contributes anything to it's application to more real-world scenarios? Explain why or why not.

Question #10: Why is it important to write simulations so that they can be run with various different input values as well as multiple times with the same input values?

Question #11: Which part of implementing `runSimulation()`, `estimateProbability()`, or `estimateExpectedValue()` did you find most challenging? How did you overcome these challenges?

Question #12: What new programming techniques or knowledge did you learn as a result of this lab?

Template Class & Test Cases

```
/**
 * Simulation Lab (Template Class and Test Cases)
 * This is the template class and test cases for the Simulation Lab;
 * Written for the Woodstock School in Mussoorie, Uttarakhand, India.
 *
 * @author Jeffrey Santos
 * @version 1.0
 */

public class Simulation {
  public static void main(String[] args) {
    //  Tests for runSimulation
    //   Outputs will vary each time your tests are run.
    System.out.println(runSimulation(10));
    System.out.println(runSimulation(50));
    System.out.println(runSimulation(100));

    //  Tests for runSimulation (overloaded):
    //   Outputs will vary each time your tests are run.
    //   Note: printArray() is implemented below.
    printArray(runSimulation(10, 5));
    printArray(runSimulation(50, 10));
    printArray(runSimulation(100, 20));

    //  Tests for estimateProbability:
    //   Outputs will vary, but should be close to indicated values.
    System.out.println(estimateProbability(10, 5));   // Output: ~0.875
    System.out.println(estimateProbability(50, 15));  // Output: ~0.220
    System.out.println(estimateProbability(100, 25)); // Output: ~0.080

    //  Tests for estimateExpectedValue:
    //   Outputs will vary, but should be close to indicated values.
    System.out.println(estimateExpectedValue(10));    // Output: ~6.0
    System.out.println(estimateExpectedValue(50));    // Output: ~11.0
    System.out.println(estimateExpectedValue(100));   // Output: ~15.0
  }

  /**
   * Runs the Alice Rumour simulation once for the indicated number of guests.
   *
   * @param n The number of guests Alice has invited (including Bob).
   *  Precondition: n > 0
   * @return  The number of guests that have heard the rumour before it stops
   *          propagating.
   */
  public static int runSimulation(int n) {
    //  To be implemented in Activity #1, Exercise 1
  }

  /**
   * Runs the Alice Rumour simulation the indicated number of times, each
   * with the indicated number of guests.
   *
   * @param n The number of guests Alice has invited (including Bob).
   *  Precondition: n > 0
   * @param N The number of times the simulation should be run.
   *  Precondition: N > 0
   * @return  An integer array in which the i-th value corresponds to the
   *          number of guests who have heard the rumour during the i-th
   *          running of the simulation.
   */
  public static int[] runSimulation(int n, int N) {
    //  To be implemented in Activity #1, Exercise 2
```

```
  }

  /**
   * Runs the Alice Rumour simulation multiple times in order to estimate the
   * probability that at least r out of n guests hear the rumour.
   *
   * @param n The number of guests Alice has invited (including Bob).
   *  Precondition: n > 0
   * @param r The target number of guests who have heard the rumour.
   *  Precondition: 0 <= r <= n
   * @return  The estimate probability that at least r out of n guests
   *          have heard the rumour.
   */
  public static double estimateProbability(int n, int r) {
    //  To be implemented in Activity #2, Exercise 1
  }

  /**
   * Runs the Alice Rumour simulation multiple times in order to estimate the
   * expected number of guests to hear the rumour before it stops propagating.
   *
   * @param n The number of guests Alice has invited (including Bob).
   *  Precondition: n > 0
   * @return  The average number of guests to have heard the rumour, calculated
   *          by running the simulation multiple times.
   */
  public static double estimateExpectedValue(int n) {
    //  To be implemented in Activity #2, Exercise 2
  }

  /**
   * A helper method to print all values in a given integer array.
   *
   * @param a The integer array to be printed.
   */
  public static void printArray(int[] a) {
    System.out.print("[" + a[0]);
    for (int i = 1; i < a.length; i++)
      System.out.print(", " + a[i]);
    System.out.println("]");
  }
}
```