University of West Florida
Department of Computer Science
Data Structures and Algorithms II
Probabilistic Algorithms Project

Project Description

The goals of this project are to:

- Part 1: implement a Monte Carlo simulation
- Part 2: implement a simple Monte Carlo algorithm
- observe how expected results compare to observed results

Part 1: A Monte Carlo Simulation

Develop capability to carry out a simple Monte Carlo simulation and compute the value of the analytical model. Input will be:

- Number of simulations
- Foreach simulation (all `ints`):
  - Categories in the simulation (eg: 6)
  - Associated frequencies for each category
  - Number of events to simulate

Your program should read the above input from a file named "`SimParameters.dat`" and set about computing the probability distribution and cumulative probability distribution, establishing the random number intervals and carrying out each simulation. It will also report the expected value for each simulation from the analytical model. You will create "`SimParameters.dat`" as a binary file.

Output

This program should report results in a table similar to this:

```
Simulation 1

     N: 500
     Simulated result: 3.63
     Expected value: 3.33
     Error percent: 0.0901

Simulation 2

     N: 300
     Simulated result: 62.63
     Expected value: 54.33
     Error percent: 0.153
...
Simulation n
     N: 3000
     Simulated result: 62.63
     Expected value: 61.33
     Error percent: 0.021
```

Compute error percent as: `abs(sim-expected)/expected`

Part 2: A Monte Carlo algorithm implementation
Develop and test a program for the bad microchip problem. The idea is that some batches of chips might not be tested. The goal, of course, is to detect bad batches without testing all the chips in the batch. We will be simulating the process of sampling chips from a collection of batches of chips.

Step 1: Generate data sets.
Automate creation of a user-specified number of datasets with a user-specified number of batches, batch size, percentage of the datasets containing bad chips, and percentage of bad chips in a dataset.
When the program runs, it will read four (4) configuration files titled `c1.txt, c2.txt, c3.txt,` and `c4.txt,` containing specs for each run. These configuration files should have the following values written as integers, one per row:

Specification	c1.txt	c2.txt	c3.txt	c4.txt
`Number of batches of items`	100	100	500	500
`Number of items in each batch`	2000	2000	1000	1000
`Percentage of batches containing bad items`	24	10	10	1
`Percentage of items that are bad in a bad set`	7	10	10	1
`Items sampled from each set`	30	50	50	50

Generate a dataset from the input specification. The dataset will contain an individual file for each batch of items. Save each file in the dataset as `ds1.txt,  ds2.txt, ... , dsn.txt`. To create an individual file, decide if it has bad items or not. Run a loop for the number of items in the batch. If it is a good batch, just write 'g' to the file (one per line) for the total number of items in the batch. If it is a bad batch, use a random number generator for the input-specified percentage of bad chips: Example - assume the spec is that 10% of chips are bad. Generate datasets by generating random numbers on [0..99] if 0 .. 9 comes up, add a bad chip (write the char 'b' to the file), otherwise, add a good chip to the data set (write a 'g' to the file).

Step 2: Create the Monte Carlo process to determine which of the chip batches are bad. It should know how many data sets there are, read them one at a time, sample the appropriate number of items, and report good batch or bad batch. Your output should look like the output below. Create a summary report detailing the summary data shown at the beginning and ending of the following for each of the four simulations.

Example output:

```
Running:
   Number of batches of items:                   100
   Number of items in each batch                2000
   Percentage of batches containing bad items     24%
   Percentage of items that are bad in a bad set   7%
   Items sampled from each set                    30

Generating data sets:
  Create bad set batch #  4, totBad =  133 total =  2000 badpct =  7
  Create bad set batch #  8, totBad =  145 total =  2000 badpct =  7
  Create bad set batch # 12, totBad =  122 total =  2000 badpct =  7
  Create bad set batch # 16, totBad =  142 total =  2000 badpct =  7
  Create bad set batch # 20, totBad =  160 total =  2000 badpct =  7
  Create bad set batch # 24, totBad =  148 total =  2000 badpct =  7
  Create bad set batch # 28, totBad =  166 total =  2000 badpct =  7
  Create bad set batch # 32, totBad =  137 total =  2000 badpct =  7
  Create bad set batch # 36, totBad =  145 total =  2000 badpct =  7
  Create bad set batch # 40, totBad =  123 total =  2000 badpct =  7
  Create bad set batch # 44, totBad =  123 total =  2000 badpct =  7
  Create bad set batch # 48, totBad =  165 total =  2000 badpct =  7
  Create bad set batch # 52, totBad =  142 total =  2000 badpct =  7
  Create bad set batch # 56, totBad =  117 total =  2000 badpct =  7
  Create bad set batch # 60, totBad =  144 total =  2000 badpct =  7
  Create bad set batch # 64, totBad =  124 total =  2000 badpct =  7
  Create bad set batch # 68, totBad =  152 total =  2000 badpct =  7
  Create bad set batch # 72, totBad =  120 total =  2000 badpct =  7
  Create bad set batch # 76, totBad =  141 total =  2000 badpct =  7
  Create bad set batch # 80, totBad =  158 total =  2000 badpct =  7
  Create bad set batch # 84, totBad =  119 total =  2000 badpct =  7
  Create bad set batch # 88, totBad =  144 total =  2000 badpct =  7
  Create bad set batch # 92, totBad =  145 total =  2000 badpct =  7
  Create bad set batch # 96, totBad =  158 total =  2000 badpct =  7
  Total bad sets = 24

Analyzing Data Sets:
  batch #0 is bad
  batch #4 is bad
  batch #12 is bad
  batch #16 is bad
  batch #20 is bad
  batch #24 is bad
  batch #28 is bad
  batch #32 is bad
  batch #36 is bad
  batch #40 is bad
  batch #44 is bad
  batch #48 is bad
  batch #52 is bad
  batch #60 is bad
  batch #64 is bad
  batch #68 is bad
  batch #72 is bad
  batch #76 is bad
  batch #80 is bad
  batch #84 is bad
  batch #88 is bad
  batch #92 is bad

Base = 0.930000 exponent = 30
P(failure to detect bad item) = 0.113367
P(batch is good) = 0.886633
Percentage of bad batches detected =  88%
```

So, the output at the end (after all the raw data has been displayed) for one run should be:

```
Summary:

Run 1:
Number of batches of items:                      100
Number of items in each batch                   2000
Percentage of batches containing bad items        24%
Percentage of items that are bad in a bad set   7%
Items sampled from each set                    3
Base = 0.930000 exponent = 30
P(failure to detect bad item) = 0.113367
P(batch is good) = 0.886633
Percentage of bad batches detected =  88%
```

Program behavior

When your program runs, it should present the user with a two-choice menu to run Part 1 or Part 2. After executing either part 1 or part 2, the program should display the manu again. After the user makes a choice, the program should automate everything else for that part of the run.

Deliverables

You will submit the following for this project:
1. The functional decomposition for your program.
2. A User's manual that describes how to set up and run your program.
3. Your source code in C.
4. Your files: `SimParameters.dat, c1.txt, c2.txt, c3.txt,` and `c4.txt.`
5. A makefile for the project

Submission Requirements:

Note: you will lose 10% if you do not follow these instructions *exactly*. The reason for this policy is that eLearning puts zip files inside zip files and it is a time-consuming, tedious, and error-prone policy to unzip them all one-by-one. I have a utility I wrote that nicely unzips all these into a usable folder structure. If you do not follow these instructions, your files will get all mashed up together with others, often with name collisions and it is an unusable mess.

1. Compile and run your program one last time before submitting it. Your program must run with gcc in the Linux lab.
2. Place every file in your submission in a SINGLE DIRECTORY named <last name>-<first initial>-p<number>. For instance, I would create directory:
   coffey-j-p4 for project 4.
3. zip that FOLDER into a .zip file with the SAME NAME. This means that inside your zip file, you will have exactly one folder (from the example: coffey-j-p3) showing at the top level. Inside that folder will be ALL the files in your project.
4. *DO NOT* make separate folders for documentation and source files (or anything else) inside the main folder. Having such a setup simply necessitates more navigation time to get where we need to go to grade.
5. Any needed input files should be in the top-level folder along with the source code.
6. MacOS users - remove the _MacOS_ utility folder before you zip up the file. If you cannot, delete it from the archive once the archive is created. It just takes up space and is not needed for anything we do with your submissions.
7. Login to UWF's eLearning system at http://elearning.uwf.edu/. Select our course.
9. Select the Dropbox option. Then select the appropriate project folder.
10. Upload your zip file to that folder. Check the dropbox to insure that the file was uploaded. ALWAYS give yourself enough time. If you are trying to submit at 11:57pm on your machine, the clock might be off and the dropbox might already be closed.

Please review the policy on ACADEMIC MISCONDUCT. This is an *individual assignment.*