zachary_hightower_433_bonus_cointoss_report

Provide detailed documentation of your algorithm and program. Explain how the game logic is implemented, how probabilities are calculated, and any optimizations or approximations used for large values of n.

Intro

Our Java program tosser simulates the coin toss game between Alice and Bob. We go through and calculate theoretical and experimental probabilities according to the game logic that we've gone over in class and through the project .pdf

Structure

The program is a single class that contains all the necessary algorithms and methods necessary for accomplishing what we want. It is constructed this way so that it can be compiled into a single class file without any trouble.

Main

- Checks number of arguments and provides the formula should the arguments be incorrect
- We take in the args and turn them into `ints` so that we can use them in the program's logic
- We go through our three main methods
    - `randomGame` We go through a single random game and show it in the print. This helps us to confirm that things are working correctly, and allows us to see the logic in action
    - `findTheoreticalProbabilities` We go through and calculate the theoretical probabilities based on the game logic and the growth of the two winning conditions among all the different possible iterations of games.
    - `findExperimentalProbabilities` We go through and find the experimental probabilities of all the games we've simulated and print them in an easily digestible format.

`randomGame`

- Simulates one random game between our two players
- Scores set to 0 to prevent any issues with pulling variables used elsewhere
- Use our random function to assign coin tosses

- Print scores and who won, or a tie if nobody did
  Time Complexity: O(coins)
  The time complexity here is defined by the number of coins we need to iterate over.
  Space Complexity: O(1)
  The space complexity is constant, we're only using a few integer variables for the storage of our scores

`findTheoreticalProbabilities`

- Calculates the theoretical probabilities of our game, according to the ways in which we expect win conditions to become more prevalent in the full set of game conditions. Who wins more often as we consider going up from H, T -> HH, TT, HT, TH -> etc.
- We use two arrays to keep track of possible winning scores, scoreH, and scoreT
- We iterate over all the possible combos of coins and update the score arrays based on the logic
- We calculate the probabilities using the arrays we've built up and print
  Time Complexity: $O(n^2)$
  We have nested loops here that compute our score arrays, resulting in an exponential time complexity to the power of two.
  Space Complexity: O(n)
  Our score arrays are of a linear size, so we can log a linear space complexity/load here.

`findExperimentalProbabilities`

- Perform multiple trials of the game to find what the experimental probabilities are based on the results of the simulated games
- We start up our counters for A wins, B wins, and tie conditions
- We simulate games for the number of trials and log their results
- Then we calculate the experimental probabilities based on the outcomes of our various trials and print
  Time Complexity: O(trials * coins)
  The time complexity is based on our trials and coins. We're doing what is essentially trials by coins matrices.
  Space Complexity: O(1)
  We're only using integer variables to store the scores, so this can be recorded and linear space complexity.

Probability Calculation

- We use BigDecimal to maintain precision throughout

- We using the rounding mode that give us probabilities rounded to 5 decimal places with HALF_UP mode

Optimizations

- There are no particular optimizations to note

Approximations

- We do some small approximation with rounding, but that doesn't really count towards approximating in a huge way throughout the program for gains in efficiency

Overall Complexity

The program is efficient for relatively low values of trials and coins, but will become slow for large values due to the nested loops approach used in theoretical probability calculation.

Calculations

n =2

Theoretical Probabilities

P(A),Chance that Alice wins = 0.25000
P(B),Chance that Bob wins = 0.25000
P(tie),Chance for a tie = 0.50000

Experiment Probabilities

P(A),Chance that Alice wins = 0.00000
P(B),Chance that Bob wins = 1.00000
P(tie),Chance for a tie = 0.00000

n=3

Theoretical Probabilities

P(A),Chance that Alice wins = 0.25000
P(B),Chance that Bob wins = 0.37500
P(tie),Chance for a tie = 0.37500

Experiment Probabilities

P(A),Chance that Alice wins = 0.33333
P(B),Chance that Bob wins = 0.33333
P(tie),Chance for a tie = 0.33333

n=4

Theoretical Probabilities

P(A),Chance that Alice wins = 0.25000
P(B),Chance that Bob wins = 0.37500
P(tie),Chance for a tie = 0.37500

Experiment Probabilities

P(A),Chance that Alice wins = 0.00000
P(B),Chance that Bob wins = 0.25000
P(tie),Chance for a tie = 0.75000

n=5

Theoretical Probabilities

P(A),Chance that Alice wins = 0.31250
P(B),Chance that Bob wins = 0.40625
P(tie),Chance for a tie = 0.28125

Experiment Probabilities

P(A),Chance that Alice wins = 0.40000
P(B),Chance that Bob wins = 0.00000
P(tie),Chance for a tie = 0.60000