CSCI 2244 – Homework 1

Out: Friday, August 30, 2019
Due: Friday, September 6, 2019, 11:59pm

This homework consists of written exercises and coding problems. You *must* type your solutions. See the "Assignments" section in the syllabus for advice about doing this. You should submit your homework via Canvas. In particular, you should upload a zip file called:

`FirstName_LastName_Homework1.zip`

Please use your full first name and last name, as they appear in official university records. The reason for doing so is that the TAs and I must match up these names with the entries in the gradebook.

This zip file should contain 2 files:

- `written.pdf` – containing your answers to all the tasks in section 1, and the results of task 2.3.
- `coin.py` – containing the code you wrote for section 2

1 Written Exercises

Let (S, P) be a discrete probability space. Recall from class that events are subsets of S. If A and B are events, then $A \cup B$ is the event representing A *or* B happening, $A \cap B$ is A *and* B, and $S \setminus A$ represents *not* A. It is sometimes convenient to introduce the notation $\overline{A}$ to represent $S \setminus A$. (Other notations you will see are A' or A^c. I prefer $\overline{A}$ because $'$ is used for too many other things like derivatives, and A^c looks like A to the power of c.)

Task 1.1 (4 pts). Let $A, B \subseteq S$ be events. Show that if A and B are independent, then $\overline{A}$ and B are independent.

Task 1.2 (4 pts). Let $A, B \subseteq S$ be events. Show that if $P(A) = 1$, then A and B are independent.

Task 1.3 (2 pts). Suppose we toss a fair coin 3 times. Let E be the event that represents getting exactly two tails. Write down the elements of the set E. What is the probability of E?

Task 1.4 (2 pts). Let A, B, C be events. Use set operations to express the following events:

- Exactly one of A, B, or C occurs.
- Both A and B occur, but not C.

Task 1.5 (5 pts). In American football, games begin with a coin toss to determine which team will receive the first kickoff. A player from the visiting team calls heads or tails, and then a referee flips a coin. If the coin lands on the side the player called, the visiting team is said to "win" the coin toss.

Imagine that the BC football team is visiting a rival team, and they are concerned that the coin the referee will use might be biased, so that heads occurs with probability p. However, they do not know the value of p. Having taken CSCI-2244, some of the players propose the following strategy: before the game, they will secretly flip their own fair coin, and then in the official coin toss, they will call whatever the outcome of their private flip was.

Do the following:

- Write down the sample space S representing the two coin tosses.
- Write the probability for each outcome in S, recalling that the the probability of the official coin turning up heads is p.
- Show that the probability that BC wins the coin toss is $1/2$, no matter what the value of p is.

2 Coding

In class, we discussed von Neumann's algorithm for generating fair coin flips from a biased coin. Recall that the idea is to flip the coin twice. If we get HT then we call that a "heads", and if we get TH then we call that a "tails". Otherwise, if we get HH or TT then we haven't been able to generate a fair coin flip, so we repeat the procedure.

If the coin we use returns heads with probability p, then HT and TH both occur with probability $p \cdot (1-p)$, so the resulting coin flips we generate are fair. However, we may have to flip the coin many times to generate one fair coin toss. Let's call this the "Level 0" version of von Neumann's algorithm. A summary of the algorithm is shown in Figure 1.

Coin Flips	Return Value
HT	H
TH	T
HH	None
TT	None

Figure 1: Level 0 Algorithm. When None is returned, we would have to repeat the process again from the beginning until we finally generate a H or T.

We saw that there is a more advanced version of the algorithm that does not completely start over from scratch if we get HH or TT. The "Level 1" version is shown in Figure 2. It starts the same by flipping the coin twice and returning H if we get HT and T if we get TH. However, if we get HH or TT it flips the coin two more times. If these two coin flips are HT or TH it again returns H or T respectively. If not, then there are four remaining possibilities for the total of four coins we have flipped so far. Since HHTT is just as likely as TTHH, we call the former "heads" and the latter "tails". If we got HHHH or TTTT then we have to try again. The idea is that we need fewer

Coin Flips	Return Value
HT	H
TH	T
HH HT	H
HH TH	T
TT HT	H
TT TH	T
HH TT	H
TT HH	T
HH HH	None
TT TT	None

Figure 2: Level 1 Algorithm

total tosses to generate our fair coin flip, on average. As with the level 0 version, the algorithm fails to generate a flip whenever all of our tosses were heads or all were tails.

We can recursively generalize this process. Conceptually, in order to make it recursive, it is helpful to have the algorithm return not just H, T, or None. Instead, in the None case it should return information on whether the sequences of biased coin flips was all heads or all tails. Then the "Level $n+1$" version of the algorithm is:

- Run the level n algorithm. If this yielded a fair coin flip, stop and return it.
- If not, run level n again. If this second run yielded a fair coin flip, return it.
- If not, then:
    1. If the first call to level n gave a sequence of all heads, and the second call was all tails, return heads.
    2. If the first call to level n was all tails, and the second call was all heads, return tails.
    3. Otherwise, return None and whether the sequence was all heads or all tails.

In this section of the homework, you will implement the above algorithm in Python 3 and then investigate some of its properties. On the course website, there is starter code which shows how to define a "random coin" class, which lets you create an object that generates biased coin flips. Rather than returning "heads" or "tails", it returns True or False. The coin object also keeps track of how many times it has been flipped.

Task 2.1 (15 pts). Write a function called `multilevel(c, n)` which takes as arguments a coin object `c` and a number `n` of levels and runs the level `n` version of von Neumann's algorithm above. More precisely, it should return a pair of values where the first component represents whether a fair coin flip has been generated, with value True, False, or None as appropriate. The second component should record whether the sequence of flips generated by `c` in this call was all True or all False. The exact way you track that is an implementation detail up to you, but you will use it as part of the recursive steps.

Task 2.2 (3 pts). Write a function `unbias(c, n)` which loops, repeatedly calling the level `n` algorithm with coin object `c` until it generates a fair coin flip. It should then return a pair, where the first component is the coin flip outcome, and the second component is the total number of times `c` was flipped while generating the fair coin value.

Task 2.3 (3 pts). Run `unbias` using different level numbers and coin biases. Specifically, you should try $p = 0.25, 0.5$ and $n = 0, 1, 2, 3$. For each combination of p and n value, you should run `unbias` 1,000,000 times, and sum the total number flips needed for that many trials. Report these values as part of written solution. Comment on how going from level 0 to level 1 compares to going from level 2 to level 3?