CSE 312: Foundations of Computing II Autumn 2024

Problem Set 4

Due: Wednesday, October 23, by 11:59pm

Instructions

Solutions format, collaboration policy, and late policy. See PSet 1 for further details. The same requirements and policies still apply. Also follow the typesetting instructions from the prior PSets.

Solutions submission. You must submit your solution via Gradescope. In particular:

- For the solutions to Task 1-6 submit under "PSet 4 [Written]" a *single* PDF file containing the solution to all tasks in the homework. Each numbered task should be solved on its own page (or pages). Follow the prompt on Gradescope to link tasks to your pages. Do not write your name on the individual pages – Gradescope will handle that.
- For Task 7, submit your code under "PSet 4 [Coding]" as a file called $\mathrm{bloom.py}$.

Task 1 – Fair Game [10 pts]

Consider the following game, defined by a parameter k: Roll a fair, 6-sided die three times independently. If you roll a six

- no times, then you lose 1 dollar.
- exactly once, you win 1 dollar.
- exactly twice, then you win 2 dollars.
- all three times, then you win k dollars

For what value of k is this game fair? (The game is fair if your expected payoff is 0.)

Task 2 – Raining Cats and Dogs [12 pts]

Suppose that $4n$ animals are partitioned into pairs at random, with each partition being equally likely. If the set consists of n dogs and $3n$ cats, what is the expected number of dog-cat pairs?

Task 3 – Tournament Prediction Games [14 pts]

A well-known college sports tournament starts with 64 teams and eliminates half of the teams in each round of games. There are 32 1st-round games, 16 2nd-round games, ..., and 1 6th-round game. The teams are scheduled to play each other according to a fixed full binary tree of height 6. Each team begins at one of the leaves of the tree and plays the team at the sibling node. The winner moves up to the parent node and this repeats until there is one overall champion left at the root.
You have a chance to play a game where you try to predict the winner of every one of the 63 games. You know the assignment of teams to the leaves of the tree. If you predict an i-th round game correctly, you get a payoff of 2^{i-1}. That is, for a 1st round game you get a payoff of 1, for a 2nd round game you get a payoff of 2, ..., and you get a payoff of 32 if you predict the winner of the 6th round final game.

a) Suppose that you predicted the game corresponding to each internal node of the tree by choosing a uniformly random team from the teams at the leaves in the subtree for that node – ignoring what you predicted for the nodes lower in the tree. (There are 2^i teams in the subtree if the game is in round i.) What would your total expected payoff be?

b) Suppose instead that you started at the leaves and predicted the winner of each game by flipping a separate independent fair coin for each node to select the winner of the game corresponding to that node from the two winners you predicted for the games corresponding to its two child nodes. What is your total expected payoff now?

Task 4 – Independence [18 pts]

We are given two independent random variables, M and K, both taking values in $\{0, 1, 2, 3, 4, 5\}$. Further, assume that you know that the marginal distribution of K is uniform, i.e., $\mathbb{P}(K = i) = 1/6$ for all $i \in \{0, 1, 2, \ldots, 5\}$. *You do not know the marginal distribution of* M. Finally, we define a new random variable C as

$$C = (M + K) \mod 6 .$$

a) For every $m, c \in \{0, 1, \ldots, 5\}$, what is the probability $\mathbb{P}(C = c, M = m)$ as a function of $\mathbb{P}(M = m)$?

b) For every $c \in \{0, 1, \ldots, 5\}$, what is $\mathbb{P}(C = c)$?

c) Show that M and C are (mutually) independent.

Task 5 – Some Really Odd Dice! [18 pts]

You are playing a game that uses a fair 8-sided die whose faces are numbered by the odd numbers, 1, 3, 5, ..., 15. The value of a roll is the number showing on the top of the die when it comes to rest. Give all answers as simplified fractions.

a) Let X be the value of one roll of the die. Compute $\mathbb{E}[X]$ and $\mathrm{Var}(X)$.

b) Let Y be the sum of the values of 5 independent rolls of the die. Compute $\mathbb{E}[Y]$ and $\mathrm{Var}(Y)$. Use independence, and state precisely where in your computation you are using it.

c) Let Z be the **average** of the values of n independent rolls of the die. Compute $\mathbb{E}[Z]$ and $\mathrm{Var}(Z)$. Use independence, and state precisely where in your computation you are using it.

Task 6 – Packet Failures [18 pts]

Consider three different models for sending n packets over the Internet:

1. Each packet takes a different path. Each path fails independently with probability p;
2. All packets take the exact same path which fails with probability p. Thus, either all the packets get through or none get through;
3. Half the packets take one path, and half take the other (assume n is even), and each of the two paths fails independently with probability p.

Let X_i be the number of packets lost in case i, for $i = 1, 2, 3$. Write down the probability mass function, the expectation of X_i and the variance of X_i for $i = 1, 2, 3$.

Task 7 – Bloom filters [Coding] [10 pts]

Google Chrome has a huge database of malicious URLs, but it takes a long time to do a database lookup (think of this as a typical Set). They want to have a quick check in the web browser itself, so a space-efficient data structure must be used. A **Bloom filter** is a **probabilistic data structure** which only supports the following two operations:

- add(x): Add an element x to the structure.
- contains(x): Check if an element x is in the structure. If either returns "definitely not in the set" or "could be in the set".

It does **not** support the following two operations:

- delete an element from the structure.
- return an element that is in the structure.

The idea is that we can check our Bloom filter to see if a URL is in the set. The Bloom filter is always correct in saying a URL definitely isn't in the set, but may have false positives – it may say that a URL is in the set when it isn't. Only in these rare cases does Chrome have to perform an expensive database lookup to know for sure.
Suppose that we have k **bit arrays** $t_1, \ldots, t_k$ each of length m (all entries are 0 or 1), so the total space required is only km bits or $km/8$ bytes (as a byte is 8 bits). Suppose that the universe of URL's is the set $\mathcal{U}$ (think of this as all strings with less than 100 characters), and we have k **independent and uniform** hash functions $h_1, \ldots, h_k : \mathcal{U} \to \{0, 1, \ldots, m-1\}$. That is, for an element x and hash function h_i, pretend $h_i(x)$ is a **discrete** $\text{Unif}[0, m-1]$ random variable. Suppose that we implement the add and contains function as follows:

Algorithm 1 Bloom Filter Operations

```
1: function INITIALIZE(k,m)
2:     for i = 1, ..., k: do
3:         t_i = new bit array of m 0's
4: function ADD(x)
5:     for i = 1, ..., k: do
6:         t_i[h_i(x)] = 1
7: function CONTAINS(x)
       return t_1[h_1(x)] == 1 ∧ t_2[h_2(x)] == 1 ∧ ··· ∧ t_k[h_k(x)] == 1
```

Refer to the slides from Lecture 11 and Section 9.4 of the textbook for more details on Bloom filters.
Implement the functions add and contains in the BloomFilter class of bloom.py. To solve this task, we have set up a corresponding edstem lesson here. Press the Mark button above the terminal to run the unit tests we have written for you. Passing these unit tests is not enough. We have written a number of different tests for the Gradescope autograder. Your score on Gradescope will be your actual score - you have unlimited attempts to submit.