CSE 312: Foundations of Computing II — Winter 2021

Problem Set 7 (due Wednesday, March 3, 11:59pm)

Directions:

***Answers**: For each problem, remember you must briefly explain/justify how you obtained your answer, as correct answers without an explanation will receive **no credit**. Moreover, in the event of an incorrect answer, we can still try to give you partial credit based on the explanation you provide. It is fine for your answers to include summations, products, factorials, exponentials, or combinations; you dont need to calculate those all out to get a single numeric answer, for instance 26^7 or $26!/7!$ or $26 \cdot \binom{26}{7}$.*

Your solutions need to be concise and clear. We will take off points for lack of clarity or for excess verbosity. Please see section worksheet solutions (posted on the course website) to gauge the level of detail we are expecting.

Please clearly indicate your final answer, in such a way as to distinguish it from the rest of your explanation.

***Groups:** This homework is to be completed in groups of 1 or 2. Specific guidelines about collaboration are available on the syllabus, but every group will be submitting their own submission. Please cite any collaboration at the top of your submission. Instructions are available here as to how to add groupmates to your submission.*

Before you start your homework, write down the list of people you collaborated with. Remember, you can only collaborate outside your group by discussing the problems at a high level. Provide names and email addresses.

***Submission**: You must upload a **pdf** of your written solutions to Gradescope under "PSet 7 [Written]". Problem 5 is an EXTRA CREDIT coding problem, so under "Pset 7 [Extra Coding]" you may optional upload a .py file called cse312_pset7_heavy_hitters.py. Problem 6 is also an extra credit problem, so if you answer it upload your solution to "Pset 7 [Extra]". (Instructions as to how to upload your solutions to Gradescope are on the course web page.) The use of latex is highly recommended.*

Note that if you want to hand-write your solutions, you'll need to scan them. We will take off points for hand-written solutions that are difficult to read due to poor handwriting and neatness.)

Please cite any collaboration at the top of your submission (beyond your group members, who should already be listed).

1. Statistics Books (24 points)

Alice is going shopping for statistics books for H hours, where H is a random variable, equally likely to be 1, 2 or 3. The number of books B she buys is random and depends on how long she is in the store for. We are told that

$$\Pr(B = b \mid H = h) = \frac{c}{h}, \qquad \text{for } b = 1, \ldots, h,$$

for some constant c.

(a) [3 Points] Compute c.

(b) [3 Points] Find the joint distribution of B and H.

(c) [5 Points] Find the marginal distribution of B.

(d) [3 Points] Find the conditional distribution of H given that $B = 1$ (i.e., $Pr(H = h|B = 1)$ for each possible h in 1,2,3).

(e) [5 Points] Suppose that we are told that Alice bought either 1 or 2 books. Find the expected number of hours she shopped conditioned on this event.

(f) [5 Points] The cost of each book is a random variable with mean 3, and is independent of the number of books Alice buys. What is the expected amount of money Alice spends?

2. Coal Mining (10 points)

A miner is trapped in a mine containing 4 doors, and each door is equally likely to be chosen. The first door leads to a tunnel that will take him to safety after a number of hours which is Poisson with parameter 2. The second door leads to a tunnel that will take him to safety after a number of hours which is Geometric with parameter $\frac{1}{5}$. The third door leads to a tunnel that will take him to safety after a number of hours which is binomial with parameters $n = 100$ and $p = 1/20$. The fourth door leads to a tunnel which brings him back to where he started after 2 hours. Use the law of total expectation to compute the expected number of hours until the miner reaches safety.

3. Markov's Inequality (8 points)

The expected grade for a student in CSE 312 is 3.4 (out of 4.0). Use Markov's inequality to upper bound the probability that a student receives less than or equal to 2.8.

4. Chernoff Bound (20 points)

A certain city is experiencing a terrible fire. The city decides that it needs to put its firefighters out into the streets all across the city to ensure that the fire can be put out. The city is conveniently arranged into a 100×100 grid of streets. Each street intersection can be identified by two integers (a, b) where $1 \leq a \leq 100$ and $1 \leq b \leq 100$. The city only has 1000 firefighters, so it decides to send each police officer to a uniformly random grid location, independent of each other (i.e., multiple firefighters can end up at the same intersection). The city wants to make sure that every 20×20 subgrid (corresponding to grid points (a, b) with $A \leq a \leq A+19$ and $B \leq b \leq B + 19$ for valid A, B) gets more than 10 firefighters (subgrids can overlap).

(a) [12 Points] Use the Chernoff bound to compute the probability that a single subgrid gets at most 10 firefighters.

(b) [8 Points] Use the union bound together with the result from above to calculate an upper bound on the probability that the city fails to meet its goal.

5. Extra Credit: Count-Min Sketch Coding (15 points)

[Assignment courtesy of Greg Valiant, CS 168, Stanford University]

Count-Min Sketch, as an algorithm, was not discussed in class this quarter. Thus, this coding assignment and the subsequent analysis is extra credit, and entirely optional.
The goal of this part is to understand Count-Min Sketch via an implementation, and to explore the benefits of a conservative update optimization. Count-Min Sketch is a probabilistic data structure that serves as a frequency table of events in a data stream. It uses hash functions to map events to frequencies, but uses much less space than a hash table, at the expense of over-counting. Upon completion of the assignment, you will be turning in your cse312_pset7_heavy_hitters.py in "Pset 7 [Extra Coding]" on Gradescope.

In addition to the following notes, you may also find the Stanford Notes on Heavy Hitters helpful.

The Count-Min Sketch consists of $l = 4$ independent hash tables t_i $(1 \leq i \leq l)$. Each hash table is an integer array of size $b = 256$, and correspond to a certain hash function h_i.

Hash function: For this assignment, you will write your own hash function. We first fix a prime number $\mathbf{p}$ that is greater than the largest element in the data stream. In this assignment, we set $\mathbf{p = 10007}$. The hash function for table i (for $i = 1, 2, 3, 4$) is chosen as follows:

- Choose an integer e_i such that $1 \leq e_i \leq \mathbf{p} - 1$ uniformly at random. Then choose another integer g_i such that $0 \leq g_i \leq \mathbf{p} - 1$ uniformly at random. The integers e_i and g_i are chosen independently and randomly for each i.
- The hash function for table j is then defined as $h_i(x) = (e_i x + g_i) \mod \mathbf{p} \mod b$.

The implementation of Count-Min Sketch is as follows:

Algorithm 1 Count-Min Sketch operations

```
function REGULAR_UPDATE(x)
    for i = 1, ..., l: do
        bucket ← h_i(x)
        t_i[bucket] ← t_i[bucket] + 1
function CONSERVATIVE_UPDATE(x)
    min ← count(x)
    for i = 1, ..., l: do
        bucket ← h_i(x)
        if t_i[bucket] == min then
            t_i[bucket] ← t_i[bucket] + 1
function COUNT(x)
    return min{t_1[h_1(x)], t_2[h_2(x)], ..., t_l[h_l(x)]}
```

Implementation Requirements:

(a) Implement the hash function as described above.

(b) Implement count(x), which returns the approximated frequency of an element x in the data stream.

(c) Implement regular update, where the counters in the hash tables are incremented by 1.

(d) Implement conservative update optimization. When updating the counters during an insert, instead of incrementing all the counters, only increment the subset of these that have the lowest current count (if two or more of them are tied for the minimum current count, then we still increment each of these).

Data: Inside the data folder, there are 3 data stream files: **forward_stream.txt**, **backward_stream.txt**, **random_stream.txt**. Each line in a data stream file is an element in the stream. Each of the 3 data stream files has 87,925 elements (integers, not necessarily distinct).

- Forward: the elements appear in non-decreasing order.
- Reverse: the elements appear in non-increasing order.
- Random: the elements appear in a random order.

In the main function, we feed each of the three data streams into the Count-Min Sketch (i.e., successively insert its elements) and compute the average estimate of the frequency of element $\mathbf{9050}$ after seeing all the elements

in the stream in $\mathbf{10}$ trials.

Note: We call an integer in the stream a **heavy hitter** if its frequency is at least $\mathbf{1}\%$ of the total number of stream elements. Each data stream has $\mathbf{87,925}$ elements. 9050 is a heavy hitter in the data stream.

6. Extra Credit: Count-Min Sketch Analysis (15 points)

(a) [4 Points] Does the order of the stream affect the estimated count of 9050 and the estimated count of heavy hitters in your implementation? In either case, explain why. Your answers should address both standard updates and conservative updates:

  (a) Without conservative updates, did order affect the estimated count of heavy hitters? Did order affect the estimated count of 9050?

  (b) With conservative updates, did order affect the estimated count of heavy hitters? Did order affect the estimated count of 9050?

(b) [4 Points] Explain why, even with conservative updates, the count-min sketch never underestimates the count of a value.

(c) [5 Points]

- What does Markov's Inequality guarantee about the probability that after seeing t elements $Count(x) \geq f_x^t + 4n/b$? [Recall the notation from class: f_x^t is the number of times element x has occurred among the first t elements of the stream. n is the number of stream elements, b is the size of each hash table and ℓ is the number of hash tables used.]
- In the coding section, we had $\ell = 4$ hash tables and the size of each table was $b = 256$. Suppose that you implemented it so that any element that has frequency more than $0.01n$ is output. Suppose also that y is an element with frequency $0.001n$ (i.e., not a heavy hitter). Use Markov's inequality to bound the probability that your implementation outputs element y, assuming that the hash functions are completely uniformly random and independent? In other words, give an upper bound on the probability that at the end
$$Count(y) \geq 0.01n?$$

(d) [2 Points] Briefly describe 2 real world applications of the Count-Min Sketch.