Homework 1:
Hashing and load balancing

Problem 1: The power of two choices [25 points]

Consider the following load-balancing setup: there are m balls that arrive one at a time, and each ball must be thrown into one of n bins $B_1, \ldots, B_n$ as it arrives. Let L denote the maximum number of balls in any bin. A good load-balancing scheme should have L small, but at the same time, should be simple to implement.

Compare the following four load balancing rules:

1. Select one of the n bins uniformly at random.
2. Select two bins uniformly at random, and place the ball into whichever bin has fewer balls so far (breaking ties arbitrarily).
3. Same as the previous strategy, but with picking 3 bins uniformly at random.
4. Select one of the n bins B_i uniformly at random, and place the ball into either B_i or $B_{(i+1) \mod n}$, whichever one has fewer balls so far (breaking ties arbitrarily).

Fun fact: strategy (2) is known as "the power of two choices," and Anna Karlin (professor at UW) was one of the people who invented it!

(a) [3 points] Write code to simulate strategies 1-4. For each strategy, there should be a function which takes as input m, n, and outputs the maximum number of balls in any bin.

(b) [5 points] Let $m = 1,000,000$, $n = 100,000$ and simulate the four strategies 100 times. For each strategy, plot the histogram of the maximum loads. Discuss the pros and cons of each strategy.

(c) [4 points] Recall the consistent hashing problem from lecture. Model consistent hashing as a balls-into-bins problem, and discuss how you may use insights from this homework problem to augment the consistent hashing scheme discussed in lecture, and what tradeoffs different strategies will give.

Some of these strategies are specific instances of a more general class of strategies. Imagine placing the n bins onto the vertices of a graph G. When a ball arrives, sample a uniformly random edge of this graph, and let B_1, B_2 be the vertices of this edge, then place the ball into whichever of B_1 and B_2 has fewer balls so far (breaking ties arbitrarily).

(d) [5 points] Which of these four strategies can be cast as an instance of this more general class? For each of these strategies, identify the associated graph.

(e) [8 points] Design a graph on $n = 100,000$ vertices for this problem, but with the constraint that every node can only have degree at most 3, so that the maximum load that you obtain is as close as possible to the load achieved by strategy (2). Simulate this strategy just as in (b), and plot the histogram of maximum loads. Explain how you chose your graph, and compare your results to the previous ones.

(g) [EC, $1 - \infty$ points] Given an arbitrary graph, characterize the maximum load as a function of m, n, and statistics of the graph. Warning: this is a famous open question!

Problem 2: Count-min sketch [30 points]

You will implement and experiment with a count-min sketch using $\ell \in \{1, \ldots, 6\}$ hash tables and $b = 256$ buckets. You will run 10 different trials of this, so that you'll get a sense for both the accuracy of the sketch, as well as how the errors are distributed.

Your sketch should be able to handle arbitrary strings (although in the homework you'll only need to deal with integers). Your code should additionally take an integer seed s as input. The hash functions $h_i : \Sigma^* \to \{0, 1, \ldots, 255\}$ for $i = \{0, \ldots, 5\}$ are computed as follows: For $x \in \Sigma^*$, define $h_i(x)$ to be the $(i+1)$-th byte of the MD5 hash of $x \circ \text{str}(s)$, where $\text{str}(\cdot)$ is the string representation of an integer and $\circ$ corresponds to concatenation of strings.

Do not implement MD5 yourself; most modern programming languages have packages to compute MD5 hashes. For example, in Python3, you can use the `hashlib` library and

```
hashlib.md5(foo.encode("utf-8")).hexdigest()
```

to compute the MD5 score of the string `foo` (returning a hexadecimal string).

(a) [4 points] Implement a data structure that implements the algorithm described above. You data structure should support the functions `inc(x, s)` and `count(x, s)`.

You will consider the following dataset, which you'll feed into your count-min sketch. Let n be a parameter we'll set later. For $i = 1, \ldots, n$, the integer i will appear in the dataset i^2 times. Then, for $i = n+1, \ldots, n^2$, the integer i will appear in the dataset once.

(b) [5 points] We say that an integer in this dataset is a *heavy hitter* if the number of times it appears in the dataset is at least 1% of the size of the total dataset. Find a general expression, as a function of n, for what elements of this dataset are heavy hitters.

For the rest of the problem, we'll set $n = 150$, and we'll use count-min sketch to find heavy hitters of this dataset with a small amount of memory, as discussed in the lecture notes. Note that if we wanted to store the entire stream in memory, we'd need to store the counts for $150^2 = 22500$ numbers. However, our count-min sketch will only need to store at most $256 \cdot 6 = 1536$ different counts!

Consider the 3 following orders the numbers can arrive:

1. Heavy first: Sort the elements in decreasing order of frequency
2. Heavy last: Sort the elements in ascending order of frequency
3. Random: Randomly permute the list

(c) [6 points] For each of the 3 data streams, feed it into your count-min sketch, and compute the following following quantities averaged over 10 trials. In trial i, you should use seed $s = i$.

- The sketch's estimate of the frequency of the number 100
- The sketch's estimate for the total number of heavy hitters in the stream

Does the order of the stream affect the estimated counts? Explain what the data says, and propose a plausible explanation.

(d) [4 points] Implement the following conservative update optimization. When you are updating the five counters, only increment the subset of counters for which the current count is smallest (if a few are tied for smallest, increment all of those).

(e) [4 points] Argue that, even with the conservative update rule, the count-min sketch never underestimates the count.

(f) [7 points] Repeat part (c) with conservative updates (including the discussion). If the outcome seems different, offer a plausible explanation.