CSE 422 Wi26 sample solutions

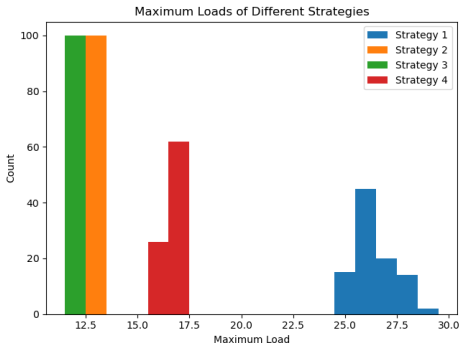
January 2026

1 Homework 1

Problem 1

Part b

- **Strategy 1:** The advantage of this strategy is that it is the fastest and easiest to implement, requiring a generation of one random number and accessing only one bin. However, the downside is that it has the highest maximum load across all the strategies.
- **Strategy 2:** The advantage of this strategy it is still relatively fast, and has the second-lowest maximum load with little variance between each trial. The disadvantage is that it is slower than strategy 1, and has a higher maximum load than strategy 3.
- **Strategy 3:** The advantage of this strategy is that has the lowest maximum load across all strategies with little variance between each trial. However, it is the slowest and hardest to implement across all three strategies, requiring accessing three bins for each insertion.
- **Strategy 4:** The advantage of this strategy is that it is faster than all strategies except strategy 1 (generating only 1 random number per insertion), and has better maximum load than strategy 1. The disadvantage



is that it has higher maximum load than strategies 2 and 3, and is slower than strategy 1.

Part c

We can model consistent hashing as a balls-in-bins problem by representing the balls as the URLs (or entries), and the bins as the cache servers. In the simplest implementation, we would pick bins in a manner similar to strategy 1 except the probability of picking a bin depends on the size of the interval between the hash of the current server and the next server.

However, this homework problem has shown that instead of considering only one server, we could use multiple hashes and pick 2 or more servers, placing the entry in the server that is less loaded. For lookup, the client would query both servers for the entry by using the two hashes, as the entry if it exists would be stored on one of the servers. Here are the tradeoffs of using the following strategies:

- **Strategy 1:** This is the same as the original consistent hashing implementation, where lookup for the server is fast, but servers can be unequally loaded.
- **Strategy 2:** This has much better load balancing than the original implementation, but requires two hash functions, and clients would need to query 2 servers each time.
- **Strategy 3:** This has the best load balancing between the servers, but requires three hash functions per entry, and querying would require checking three servers.
- **Strategy 4:** This has better load balancing than the original implementation, and still only uses one hash function. However, it has worse maximum load compared to strategies 2 and 3, and clients would still need to query two servers (the server with interval matching the hash, and the next server).

Part d

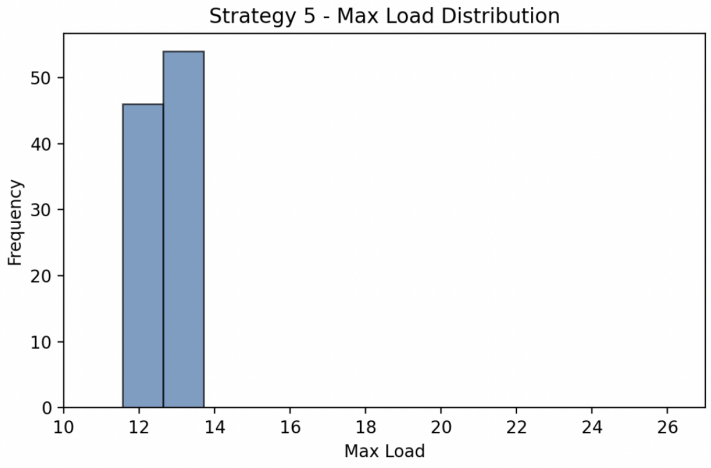
Strategies 1, 2, and 4 can be modeled as a specific instance of this more general problem.

- **Strategy 1:** Each node is its own disjoint set (not connected to any other node), and there is a self-edge connecting every node to itself. Therefore, B_1 and B_2 would always be the same bin.
- **Strategy 2:** A complete graph where each node is connected to every other node by an edge. Therefore, it is uniformly probable to choose any combination of two bins.

- **Strategy 4:** A cycle graph where each node is connected to both the previous node, and the next node. Therefore, it is uniformly possible to pick nodes i and $i+1 \pmod{n}$ for all i.

Part e

For my approach, I used all three of my degrees to build as dense a graph as possible while following the constraint. With the first two degrees I connected each index to the bin following it and the bin before it, which is similar to strategy 4. However strategy 4 suffered by collecting load locally, so I used my last degree for each vertex to connect to the vertex at the opposite end of the circle, or index (i + (n//2)) mod n, which could spread out the localization to other bins. It looks like this strategy succeeded in doing only slightly worse than the fully random strategy 2, which I'm happy with.



It's also clear that this strategy beats strategy 4, which makes sense as in strategy 4 vertices only have a degree of 1. This also beats strategy 1, which was worse than 4, and loses to strategy 3, which was better than 2.

Problem 2

Part b (by Aidan Yu)

The total size of the dataset is given by:

$$N = \sum_{i=1}^{n} i^2 + \sum_{i=n+1}^{n^2} 1 = \frac{n(n+1)(2n+1)}{6} + n(n-1) = \frac{2n^3 + 9n^2 - 5n}{6}$$

A heavy hitter appears at least 1% of the size of the data set, thus $\geq \frac{N}{100}$ times. For integers $i = \{1, \ldots, n\}$, i is a heavy hitter if:

$$i^2 \geq \frac{2n^3 + 9n^2 - 5n}{600} \rightarrow i \geq \sqrt{\frac{2n^3 + 9n^2 - 5n}{600}}$$

For integers $i = \{n+1, \ldots, n^2\}$, i is a heavy hitter if:

$$1 \geq \frac{2n^3 + 9n^2 - 5n}{600} \rightarrow 600 \geq 2n^3 + 9n^2 - 5n$$

which is only true for $n \leq 5$ assuming $n \in \mathbb{Z}$.
So overall, the set of all heavy hitter is:

$$\left\{ i \in \mathbb{Z}, 1 \leq i \leq n | i \geq \sqrt{\frac{2n^3 + 9n^2 - 5n}{600}} \right\}$$

if $n > 5$ and

$$\{1, \ldots, n^2\}$$

if $n \leq 5$.

Part c (by Vincent Chau)

Estimate of the Frequency of 100

Strategy	Frequency
Strategy 1	10080.6
Strategy 2	10080.6
Strategy 3	10080.6

Estimate of the Total Number of Heavy Hitters in the Stream

Strategy	Number
Strategy 1	43.3
Strategy 2	43.0
Strategy 3	43.2

Note that the 10 trials were done with seeds from $\{1, \ldots, 10\}$.

The order of the stream **does not affect the frequency of the number 100** (queried from the CMS table after processing the stream). It **does affect the total number of heavy hitters in the stream** (which were determined

while processing the stream). This is consistent with the data - the frequency of the number 100 remained the same across strategies for the same trial (same seed), but was different for the total number of heavy hitters in the stream.

For the estimate of the frequency of 100, since we query after processing the stream, it does not matter what order the stream is processed. Everything that hashes to the same cells of the CMS table that "100" hashes to will have incremented those cell counts (and, all cells that correspond with a data point are incremented regardless). Since addition is commutative, the final counts of the entire table are the same after processing the stream, regardless of the order it was processed.

However, for determining the total number of heavy hitters, since we determine what is a heavy hitter while we are processing the stream, the order matters.

Consider the extreme case when $\ell = 1$ and $b = 1$, and our dataset consists of 2 unique elements - a, b. There is exactly one instance of a, but some large number instances of b. Both will hash to the same cell (since the table is 1×1). If a is processed at the beginning of the stream, clearly it is not considered a heavy hitter. However, if a is processed as the last element of the stream, it will be a heavy hitter, since all of the instances of b have incremented the cell count that a hashes to.

In other words, it could be that, for an element of the dataset, in one order of the strean it was processed before other elements incremented the cell count it hashes to (so, not a heavy hitter), and in another order of the stream it was processed after other elements had incremented the cell count it hashes to (so, a heavy hitter).

Part e (By Eric Ye)

Proof. We can show this via induction.

Inductive hypothesis: $\min_{i \in \{1,\ldots,l\}} \text{CMS}[i, h_i(x)] \geq c(x)$ holds for the first d elements of the data.

Base Case: We have $d = 0$ and $c(x) = 0$. Since the buckets are initialized to 0, we have $\min_{i=\{1,\ldots,\ell\}} \text{CMS}[i, h_i(x)] = 0 \geq c(x)$, and the inductive hypothesis holds.

Inductive step: Consider the first $d + 1$ elements of the data. We have two cases:

- Case 1: the $d+1^{\text{th}}$ element is not x. This means that $c(x)$ is unchanged. If there is a hash collision, the bucket would get incremented, and it will stay the same otherwise. Therefore, $\min_{i=1,\ldots,\ell} \text{CMS}[i, h_i(x)]$ for the first $d+1$ elements is $\geq \min_{i=\{1,\ldots\ell\}} \text{CMS}[i, h_i(x)]$ for the first d elements. Since $\min_{\{1,\ldots,\ell\}} \text{CMS}[i, h_i(x)]$ holds for the first d elements by the inductive hypothesis, $\min_{i=\{1,\ldots,\ell\}} \text{CMS}[i, h_i(x)]$ will hold for the first $d+1$ elements.

- Case 2: the $d+1^{\text{th}}$ element is x. This means that $c(x)$ is incremented by 1. For all hashed buckets, we have two cases:
  - If it is the minimum, it is incremented by 1.
  - If it is not the minimum, it is already $\geq \min_{\{i=1,\ldots,\ell\}}(\text{CMS}[i, h_i(x)])+1$, and the value stays the same.

  Therefore, $\min_{\{i=1,\ldots,\ell\}} \text{CMS}[i, h_i(x)]$ for the first $d+1$ elements is $\geq \min_{\{i=1,\ldots,\ell\}}(\text{CMS}[i, h_i(x)]) + 1$ for the first d elements. By the inductive hypothesis, $\min_{\{i=1,\ldots,\ell\}} \text{CMS}[i, h_i(x)] \geq c(x)$ for the first d elements, and since both sides were incremented by 1 for the first $d+1$ elements, $\min_{\{i=1,\ldots,\ell\}} \text{CMS}[i, h_i(x)] \geq c(x)$ holds for the first $d+1$ elements.

Therefore, even with the conservative update rule, we never overestimate the count. □

Part f (by Vincent Chau)

Estimate of the Frequency of 100

Strategy	Frequency
Strategy 1	10000.0
Strategy 2	10033.8
Strategy 3	10000.0

Estimate of the Total Number of Heavy Hitters in the Stream

Strategy	Number
Strategy 1	43.2
Strategy 2	43.0
Strategy 3	43.0

Note that the 10 trials were done with seeds from $\{1, \ldots, 10\}$.

The order of the stream **does affect the frequency of the number 100** (queried from the CMS table after processing the stream), and **does affect the total number of heavy hitters in the stream** (which were determined while processing the stream) using the conservative update optimization. This is consistent with the data - the frequency of the number 100 was different across strategies for the same trial (same seed), and was different for the total number of heavy hitters in the stream.

For the estimate of the frequency of 100, the final state of the CMS table depends on the order of the stream. This is because the subset of cells that an element hashes to that we consider are the minimum valued ones. If we reorder the stream, a different ordering of cell subsets would be considered first (so the cells that are minimal at each update are different), so we can end up incrementing a different subset of cells.

As a minimal example, consider when $\ell = 2$ and $b = 2$, and our dataset consists of 3 unique elements a, b, c that all occur once. a, b each hash to different columns of the (2×2 CMS table), while c hashes along the diagonal. If the

ordering has c last, the processing of a, b will both update 2 cells each. If c is first, a, b will both update 1 cell each.

The final state of the CMS table differs because conservative update increments only the current minimum cells, and changing the stream order changes which counters are minimal at each consideration.

For determining the total number of heavy hitters, since we determine what is a heavy hitter while we are processing the stream, the order matters.

The reasoning for this is the same as part (c).