Problem One: Cuckoo Hashing

Here are two details about the implementation of vanilla cuckoo hashing (two hash functions, one item per slot) that might seem challenging to handle in practice:

1. We need two hash functions $h_1(x)$ and $h_2(x)$ such that $h_1(x) \neq h_2(x)$ for any key x. It seems like it would be hard to get hash functions with these properties.
2. When displacing a key x from its home, we need to move it to either position $h_1(x)$ or $h_2(x)$ depending on which of the two positions it was previously in. This seems like it requires us to compute $h_1(x)$ and $h_2(x)$ when doing the displacement, though one of those calculations isn't needed.

Turns out, there's a really nice way to address both concerns.

Let's begin by assuming that we have a table with m elements, where m is a perfect power of two. We'll assume we have access to two families of 2-independent hash functions: $\mathscr{H}_m$, which maps from the universe of keys to the set $\{0, 1, 2, \ldots, m - 1\}$, and $\mathscr{H}_{m-1}$, which maps from the universe of keys to the set $\{1, 2, 3, \ldots, m - 1\}$. We'll then sample a hash function h_1 from $\mathscr{H}_m$ and, independently, a second hash function h_Δ from $\mathscr{H}_{m-1}$. We'll then define our second hash function h_2 to be

$$h_2(x) = h_1(x) \oplus h_\Delta(x),$$

where $\oplus$ denotes the bitwise XOR operation.

i. Prove that $h_1(x) \neq h_2(x)$ for any key x.

This choice of hash function makes it easy to displace an element from its current position to the position given by its other hash. Assuming we displace key x from position i in the table, we simply move key x to position $i \oplus h_\Delta(x)$.

ii. Prove that this procedure always moves key x from $h_1(x)$ to $h_2(x)$ or vice-versa.

Now, let $\mathscr{H}_{cuckoo}$ denote the family of pairs of hash functions (h_1, h_2) produced this way. This is a family of hash functions over the set $E = \{(i, j) \mid i, j \in [m] \text{ and } i \neq j\}$.

iii. Prove that $\mathscr{H}_{cuckoo}$ is 2-independent. We're expecting a formal proof that references the definition of 2-independence.

As a note, for cuckoo hashing to work properly, a stronger degree of independence is required than what you proved here. Nonetheless, we figured it would be a good exercise to work through these details so you could appreciate the details! You often see this idea employed in practice.

Problem Two: Final Details on Count Sketches

In our analysis of count sketches from lecture, we made the following simplification when determining the variance of our estimate:

$$\mathrm{Var}\Big[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j\Big] = \sum_{j \neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) X_j]$$

In this expression, we've fixed some value for an index i, and are summing over all the other indices.

In general, the variance of a sum of random variables is not the same as the sum of their variances. That only works in the case where all those random variables are ***pairwise uncorrelated***, as you saw on Problem Set Zero.

Prove that for any indices $j \neq k$ (where $j \neq i$ and $k \neq i$) that $\boldsymbol{a}_j\, s(x_i)\, s(x_j)\, X_j$ and $\boldsymbol{a}_k\, s(x_i)\, s(x_k)\, X_k$ are pairwise uncorrelated random variables, under the assumption that both s and h are drawn uniformly and independently from separate 2-independent families of hash functions. Refer back to the slides on the count sketch for the definitions of the relevant terms here. Remember that $\boldsymbol{a}_j$ and $\boldsymbol{a}_k$ are not random variables. Two random variables X and Y are uncorrelated if $E[XY] = E[X]E[Y]$.