CS 170 DIS 12

Released on 2018-04-17

1 2-Universal Hashing

Let $\mathcal{H}$ be a class of hash functions in which each $h \in \mathcal{H}$ maps the universe $\mathcal{U}$ of keys to $\{0, 1, \ldots, m-1\}$. We say that $\mathcal{H}$ is 2-universal if, for every fixed pair, $\langle x, y \rangle$ of keys where $x \neq y$, and for any h chosen uniformly at random from $\mathcal{H}$, the fair $\langle h(x), h(y) \rangle$ is equally likely to be any of the m^2 pairs of elements from $\{0, 1, \ldots, m-1\}$. (The probability it taken only over the random choise of the hash function.)

(a) Show that, if $\mathcal{H}$ is 2-universal, then it is universal.

Solution: If $\mathcal{H}$ is 2-universal, then for every pair of distinct keys x and y, and for every $i \in \{0, 1, \ldots, m-1\}$,

$$\Pr_{h \in \mathcal{H}}[\langle h(x), h(y) \rangle = \langle i, i \rangle] = \frac{1}{m^2}$$

There are exactly m possible ways for us to have x and y collide, i.e., $h(x) = h(y) = i$ for $i \in \{0, 1, \ldots, m-1\}$. Thus,

$$\Pr_{h \in \mathcal{H}}[h(x) = h(y)] = \sum_{i=0}^{m-1} \left(\Pr_{h \in \mathcal{H}}[\langle h(x), h(y) \rangle = \langle i, i \rangle] \right) = \frac{m}{m^2} = \frac{1}{m}$$

Therefore, by definition, $\mathcal{H}$ is universal.

(a) Suppose that an adversary knows the hash family $\mathcal{H}$ and controls the keys we hash, and the adversary wants to force a collision. In this problem part, suppose that $\mathcal{H}$ is universal. The following scenario takes place: we choose a hash function h randomly from $\mathcal{H}$, keeping it secret from the adversary, and then the adversary chooses a key x and learns the value $h(x)$. Can the adversary now force a collision? In other words, can it find a $y \neq x$ such that $h(x) = h(y)$ with probability greater than $1/m$?

Solution: We can construct a scenario where the adversary can force a collision. On a universe $\mathcal{U} = \{x, y, z\}$, consider the following family $\mathcal{H}$:

	x	y	z
h_1	0	0	1
h_2	1	0	1

$\mathcal{H}$ is a universal hash family: x and y collide with probability $1/2$, x and z collide with probability $1/2$, and y and z collide with probability $0 < 1/2$.

The adversary can determine whether we have selected h_1 or h_2 by giving us x to hash. If $h(x) = 0$, then we have chosen h_1, and the adversary then gives us y. Otherwise, if $h(x) = 1$, we have chosen h_2 and the adversary gives us z.

2 Markov Bound Review

Recall Markov's Inequality from CS 70. That is, for any non-negative random variable X, $Pr(X \geq a) \leq \frac{E[X]}{a}$. Provide a simple proof for Markov's bound.

Solution: $E[X] = \sum_x xPr[X = x] \geq \sum_{x \geq a} xPr[X = x] \geq \sum_{x \geq a} aPr[X = x] = a\sum_{x \geq a} Pr[X = x] = aPr[X \geq a]$.

3 Streaming for Voting

Consider the following scenario. Votes are being cast for a major election, but due to a lack of resources, only one computer is available to count the votes. Furthermore, this computer only has enough space to store one vote at a time, plus a single extra integer. Each vote is a single integer 0 or 1, denoting a vote for Candidate A and Candidate B respectively.

(a) Come up with an algorithm to determine whether candidate A or B won, or if there was a tie.

(b) Consider now an election with 3 candidates. Say there is a winner only if a candidate recieves more than 50 percent of the vote, otherwise there is no winner. If we're given another integer's worth of storage, come up with an algorithm to determine the winner if there is one. For simplicity, your algorithm can output any of the candidates in the case that there is no winner (not necessarily the one with the most votes). Votes are now numbered 0, 1, 2.

Solution:

(a) Initialize one of the integers i to 0. Use the other to store the incoming votes. For every vote for Candidate A, decrement i by one. For every vote to candidate B, incremenent i by 1.

If at the end i is negative, Candidate A won. If at the end i is positive, Candidate B won. If i is 0, there was a tie.

(b) Let m be a variable which will hold either 0, 1, or 2. Let i be a counter similar to in the previous part. For each element in the stream, do the following. If i is 0, set m equal to the value of the current vote, and set i to 1. Else if $i > 0$, and m is equal to the current vote in the stream, increment i. Else decrement i.

At the end, if there is a majority vote, m will contain it. However, if there is no majority vote, m will still contain some element of the stream.

We can see that this is the case with the following short inductive proof:

Base case: Our algorithm will definitely detect the majority element of a 1 element stream.

Inductive Hypothesis: Assume our algorithm works for a stream of n or fewer elements.

We have two cases. Either the first candidate's count drops to zero at some point during the streaming, or it doesn't.If the first candidate's count never drops to zero, then the first candidate must be the majority candidate.

If the first candidate's count does drop to zero (let's say after the xth vote), then we can say the following. Suppose we have not yet encountered the true majority element. Then it must be the majority of the rest of the stream. By the inductive hypothesis we will find it by the end of the algorithm. If we have encountered the true majority element already, we can say the following. This element could have appeared only up to $\frac{x}{2}$ times so far. Of the remaining $n - x$ elements, at least $(\frac{n}{2} + 1 - \frac{x}{2}) = \frac{n-x}{2} + 1$ must be the majority element. Thus the true majority element will also be the majority of the rest of the stream. We can then apply the inductive hypothesis again, completing the proof.