CS 170 DIS 12

Released on 2018-11-26

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 $\{0, 1, \ldots, m-1\}$. Recall that $\mathcal{H}$ is *universal* if for any $x \neq y \in \mathcal{U}$, $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq 1/m$.

We say that $\mathcal{H}$ is 2-universal if, for every fixed pair (x, y) of keys where $x \neq y$, and for any h chosen uniformly at random from $\mathcal{H}$, the pair $(h(x), h(y))$ is equally likely to be any of the m^2 pairs of elements from $\{0, 1, \ldots, m-1\}$. (The probability is taken only over the random choice of the hash function.)

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

(b) Suppose that you choose a hash function $h \in \mathcal{H}$ uniformly at random. Your friend, who does not know which hash function you picked, tells you a key x, and you tell her $h(x)$. Can your friend tell you $y \neq x$ such that $h(x) = h(y)$ with probability greater than $1/m$ (over your choice of h) if:

(i) $\mathcal{H}$ is universal?

(ii) $\mathcal{H}$ is 2-universal?

In each case, either give a choice of $\mathcal{H}$ which allows your friend to find a collision, or prove that they cannot for any choice of $\mathcal{H}$.

Solution:

(a) 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.

(b) (i) 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.

(ii) Suppose that your friend uses the function $f : \mathcal{U} \times \{0, \dots m-1\} \to \mathcal{U}$ to find a collision. We can assume that $f(x, i) \neq x$ for all x, i. The probability that your friend wins is then

$$\Pr_{h \in \mathcal{H}}[h(x) = h(f(x, h(x)))] = \sum_{i=0}^{m-1} \Pr_{h \in \mathcal{H}}[(h(x), h(f(x, i)) = (i, i)] = \frac{1}{m} \ .$$

2 Markov Bound Review

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

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 Document Comparison with Streams

You are given a document A and then a document B, both as streams of words. Find a streaming algorithm that returns the degree of similarity between the words in the documents, given by $\frac{|I|}{|U|}$, where I is the set of words that occur in both A and B, and U is the set of words that occur in at least one of A and B.

Clearly explain your algorithm and briefly justify its correctness and memory usage (at most $\log(|A| + |B|)$. Can we achieve accuracy to an arbitrary degree of precision? That is, given any $\epsilon > 0$ can we guarantee that the solution will always be within a factor of $1 \pm \epsilon$ with high probability?

Solution: Simply use the number of distinct elements streaming algorithm we saw in class, on the streams of A, B, and $C := A \cup B$ (for this third stream, process words in A and words in B). Let $|A|, |B|$, and $|C|$ be the output of the algorithm on the corresponding set, our estimate for the number of distinct words in it. Then, $|U| = |C|$, and $|I| = |A| + |B| - |U|$. Thus our estimate for $|I|/|U| = (|A| + |B| - |C|)/|C|$.

Memory usage is logarithmic in the length of the documents, as we're using a constant number (three) copies of the streaming algorithm shown in class, which used logarithmic memory. Correctness flows from the correctness of the underlying streaming algorithm for distinct elements, combined with the elementary axiom in set theory that $|A \cap B| = |A| + |B| - |A \cup B|$.

We can achieve accuracy within an arbitrary factor, because the accuracy of the similarity estimate depends on the accuracy of the underlying number of distinct elements algorithm, which we saw in class gives a result which is with high probability a fraction $(1 \pm \epsilon)$ of the true value.

The full mathematical details are beyond our scope, but we can do a rough analysis to gauge our algorithm's accuracy: the numerator is the sum of three outputs of the streaming algorithm, so is with high probability $1 \pm 3\epsilon$, and the denominator is within $1 \pm \epsilon$, of their true values, so the overall fraction is with high probability within $(1 \pm 3\epsilon)/(1 \mp 1\epsilon)$. The worst case in this range would be $(1 + 3\epsilon)/(1 - \epsilon)$, (or swap the + and - signs), which simplifies to $\frac{(1+3\epsilon)(1+\epsilon)}{(1-\epsilon)(1+\epsilon)} = \frac{1+4\epsilon+3\epsilon^2}{1-\epsilon^2} \approx 1 + 4\epsilon$, because ϵ^2 is negligible. In the other case, we obtain $1 - 4\epsilon$, so we conclude the overall accuracy is likely to be within $1 \pm 4\epsilon$. This calculation is not exact, but suffices to argue that the accuracy of the overall algorithm is a function of the accuracy of the underlying number of distinct elements algorithm it used, which can be set arbitrarily low.

4 Lower Bounds for Streaming

(a) Consider the following simple 'sketching' problem. Preprocess a sequence of bits $b_1, \ldots, b_n$ so that, given an integer i, we can return b_i. How many bits of memory are required to solve this problem exactly?

(b) Given a stream of integers $x_1, x_2, \ldots$, the *majority element* problem is to output the integer which appears most frequently of all of the integers seen so far. Prove that any algorithm which solves the majority element problem exactly must use $\Omega(n)$ bits of memory, where n is the number of elements seen so far.

Solution:

(a) n bits. Intuitively, this is because at the end of the preprocessing, there are 2^n different 'states' the algorithm has to be in, one for each bitstring. The number of states of a machine with ℓ bits of memory is 2^ℓ, so $\ell \geq n$. A more detailed argument follows.

The preprocessing algorithm is a function $f\colon \{0,1\}^n \to \{0,1\}^\ell$, where ℓ is the number of bits of memory needed to answer queries. The query algorithm is a function $q\colon [n] \times \{0,1\}^\ell \to \{0,1\}$. Observe that we can use the query algorithm to invert f on its image: if $g(y) = (q(1,y), q(2,y), \ldots, q(n,y))$, then $g(f(x)) = x$ for all $x \in \{0,1\}^n$. Hence f is injective, which means that $\ell \geq n$.

(b) We can prove this by reduction from the previous problem. For any string of bits $b_1, \ldots, b_\ell$, we define a stream of integers $0, 0, (i,i)_{b_i=1}$. Now we can query b_i by adding i to the stream and checking if it is the majority element. The length of the sequence is $n \leq 2\ell + 1$, so the memory usage is at least $\frac{n-1}{2}$ bits.

An alternative approach is for the stream to be $((-1)^{b_i} \cdot i)_{i \in [n]}$. Then we can query b_i by adding $(i, -i)$ to the stream. If $-i$ is the majority element, then $b_i = 1$, otherwise $b_i = 0$.