W4231: Analysis of Algorithms

9/30/1999 (Revised 10/4)

- More probability
- Randomized hash functions

Conditional Probability

The probability that an event A happens conditioned upon an event B happening is defined as

$$\mathbf{Pr}[A|B] = \frac{\mathbf{Pr}[A\&B]}{\mathbf{Pr}[B]}$$

Note that this is undefined when $\mathbf{Pr}[B] = 0$.

This formalizes the intuition that knowing a certain information (that event B happened) can change our understanding of the chances of something else happening.

Examples

If I toss two dice, what is the probability that their sum is 8?

If I toss two dice, what is the probability that their sum is 8 conditioned upon the first die showing a 5?

Observation

Suppose you have two random variables X and Y, and you know the probabilities of all the events of the form $(X = x \& Y = y)$. How do you compute the probability of an event of the form $(X = x)$?

$$\mathbf{Pr}[X = x] = \sum_y \mathbf{Pr}[X = x \& Y = y]$$

Linearity of Expectation

If X and Y are random variables, then

$$\mathbf{E}[X+Y] = \mathbf{E}[X] + \mathbf{E}[Y]$$

Proof:

$$\begin{aligned}
\mathbf{E}[X+Y] &= \sum_{x,y} (x+y)\,\mathbf{Pr}[X = x \& Y = y] \\
&= \sum_x x \sum_y \mathbf{Pr}[X = x \& Y = y] \\
&\quad + \sum_y y \sum_x \mathbf{Pr}[X = x \& Y = y] \\
&= \sum_x x\,\mathbf{Pr}[X = x] + \sum_y y\,\mathbf{Pr}[Y = y] \\
&= \mathbf{E}[X] + \mathbf{E}[Y]
\end{aligned}$$

Use of Linearity of Expectation

If we toss 6 coins, what is the expected number of heads that we find?

Of course it's 3, but the computation is not so easy.

Define a random variable X being equal to the number of heads. Then $\mathbf{Pr}[X = k] = \binom{6}{k}2^{-6}$ for $k = 0, \ldots, 6$ and the average is

$$\begin{aligned}\mathbf{E}[X] &= 0 \cdot 1 \cdot 2^{-6} + 1 \cdot 6 \cdot 2^{-6} + 2 \cdot 15 \cdot 2^{-6} \\ &\quad +3 \cdot 20 \cdot 2^{-6} + 4 \cdot 15 \cdot 2^{-6} + 5 \cdot 6 \cdot 2^{-6} + 6 \cdot 1 \cdot 2^{-6} \\ &= 192/64 = 3\end{aligned}$$

Complicated!

Using Linearity of Expectation

Define $X_1, \ldots, X_6$ to be the number of heads at the first, second,. . . , sixth toss.

Then for every i we have $\mathbf{E}[X_i] = 0 \cdot 1/2 + 1 \cdot 1/2 = 1/2$.

Then, since $X = X_1 + \ldots + X_6$, we have

$$\mathbf{E}[X] = \mathbf{E}[X_1 + \ldots + X_6] = \mathbf{E}[X_1] + \ldots + \mathbf{E}[X_6] = 6 \cdot 1/2 = 3$$

Be careful with the other operations

It is not true in general that if X and Y are random variables then

$$\mathbf{E}[X \cdot Y] = \mathbf{E}[X] \cdot \mathbf{E}[Y]$$

or then

$$\mathbf{E}[X/Y] = \mathbf{E}[X]/\mathbf{E}[Y]$$

Independence

Two random variables X and Y are independent if for every x and y

$$\mathbf{Pr}[X = x | Y = y] = \mathbf{Pr}[X = x]$$

so if we know the value of Y this does not change our understanding of the possible values of X (because X is not dependent on Y).

If X and Y are independent, then $\mathbf{E}[X \cdot Y] = \mathbf{E}[X] \cdot \mathbf{E}[Y]$.

Randomized Hash

We define h using randomness.

The randomness is used at the very beginning. Then h is completely deterministic.

We would like to show that for a good random choice of h, each `insert` and `delete` takes average constant time.

The average is over the initial choice of h, and it is constant for every possible set of data we want to store.

Full Randomization

Suppose h is a uniformly chosen random function from $\{1, \ldots, M\}$ to $\{1, \ldots, m\}$.

Good news: a fixed set of inputs, hashed with a fully random h, gives a random set of outputs.

Bad news: with the time and space needed to generate/store h, we might as well use a vector of size M and have worst-case $O(1)$ operations.

Universal Hashing

A random hash function is universal if for every two different inputs x and y, $\mathbf{Pr}[h(x) = h(y)] \leq 1/m$.

[Note: in CLR definition $\mathbf{Pr}[h(x) = h(y)] = 1/m$]

A fully random function is an example of universal hash function.

There are other examples that are quicker to generate and store.

A Simple Universal Hash Function

We want to describe a family of functions $h : \{1, \ldots, M\} \to \{1, \ldots, m\}$. Fix a prime $p > M$.

We have a function in the family for each $a \in \{1, \ldots, p-1\}$ and every $b \in \{0, \ldots, p-1\}$. The definition is

$$h_{a,b}(v) = (av + b \pmod p) \pmod m$$

Analysis

Let x, y be different. Let us count the number of different a, b such that $h_{a,b}(x) = h_{a,b}(y)$.

Essentially, we have to consider all possible $s, t \in \{0, \ldots, p-1\}$ such that $s = t \pmod m$, and then for each of them count the number of possible a, b such that $h_{a,b}(x) = s$ and $h_{a,b}(x) = t$.

First Step

For fixed $x \neq y$ and for each fixed s and t, it is true that there is only one possible choice of $a \neq 0, b$ such that

$$ax + b = s \pmod p$$

and

$$ay + b = t \pmod p$$

if $s \neq t$, and no choice otherwise.

Second Step

There at most $p(\lceil p/m \rceil - 1) < p(p-1)/m$ values $s \neq t$ such that $s = t \pmod m$.

Then there at most $p(p-1)/m$ values $a \neq 0, b$ such that $h_{a,b}(x) = h_{a,b}(y)$, and the probability that one of them is selected is at most $1/m$.

Use of Universal Hash Function

For every possible key x. For every possible n other key $y_1, \ldots, y_n$.

If we pick $h : \{1, \ldots, M\} \to \{1, \ldots, m\}$ from a universal family of hash functions, then the expected number of collisions between $h(x)$ and $h(y_1), \ldots, h(y_n)$ is n/m.

Analysis of Use of Universal Hashing

Let $y_1, \ldots, y_n$ be an arbitrary set of keys that have been inserted in the table. Let x be an arbitrary key.

Call C the random variable (depending on the choice of h) that counts the number of collisions between $h(x)$ and $h(y_1), \ldots, h(y_n)$. I.e. $C = |\{j : h(y_j) = h(x)\}|$.

Call C_y the random variable (depending on the choice of h) that is 1 if $h(x) = h(y)$, and 0 otherwise.

Then $C = \sum_{i=1}^n C_{y_i}$ and $\mathbf{E}[C_{y_i}] \leq 1/m$.

So $\mathbf{E}[C] = \mathbf{E}[\sum_{i=1}^n C_{y_i}] = \sum_{i=1}^n \mathbf{E}[C_{y_i}] \leq n/m$.

Interpretation

If we build the table with a random universal hash function, no matter which sequence of keys we have to insert, and which key we are considering at a given moment, the average (over h) of the time needed to do a find or a delete on x is $O(n/m)$, the *load factor*, the same as having random inputs.