18 Probability in Hashing

A popular method for storing a collection of items to support fast look-up is hashing them into a table. Trouble starts when we attempt to store more than one item in the same slot. The efficiency of all hashing algorithms depends on how often this happens.

Birthday paradox. We begin with an instructive question about birthdays. Consider a group of n people. Each person claims one particular day of the year as her birthday. For simplicity, we assume that nobody claims February 29 and we talk about years consisting of $k = 365$ days only. Assume also that each day is equally likely for each person. In other words,

$$P(\text{person } i \text{ is born on day } j) \quad = \quad \frac{1}{k},$$

for all i and all j. Collecting the birthdays of the n people, we get a multiset of n days during the year. We are interested in the event, A, that at least two people have the same birthday. Its probability is one minus the probability that the n birthdays are distinct, that is,

$$\begin{aligned} P(A) &= 1 - P(\bar{A}) \\ &= 1 - \frac{k}{k} \cdot \frac{k-1}{k} \cdot \ldots \cdot \frac{k-n+1}{k} \\ &= 1 - \frac{k!}{(k-n)!k^n}. \end{aligned}$$

The probability of A surpasses one half when n exceeds 21, which is perhaps surprisingly early. See Figure 22 for a display how the probability grows with increasing n.

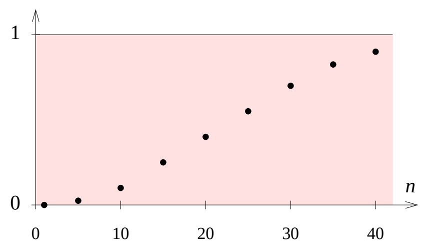

Figure 22: The probability that at least two people in a group of n share the same birthday.

Hashing. The basic mechanism in hashing is the same as in the assignment of birthdays. We have n items and map each to one of k slots. We assume the n choices of slots are independent. A *collision* is the event that an item is mapped to a slot that already stores an item. A possible resolution of a collision adds the item at the end of a linked list that belongs to the slot, but there are others. We are interested in the following quantities:

1. the expected number of items mapping to same slot;
2. the expected number of empty slots;
3. the expected number of collisions;
4. the expected number of items needed to fill all k slots.

Different hashing algorithms use different mechanisms for resolving collisions. The above quantities have a lot to say about the relative merits of these algorithms.

Items per slot. Since all slots are the same and none is more preferred than any other, we might as well determine the expected number of items that are mapped to slot 1. Consider the corresponding indicator random variable,

$$X_i \quad = \quad \begin{cases} 1 & \text{if item } i \text{ is mapped to slot } 1; \\ 0 & \text{otherwise.} \end{cases}$$

The number of items mapped to slot 1 is therefore $X = X_1 + X_2 + \ldots + X_n$. The expected value of X_i is $\frac{1}{k}$, for each i. Hence, the expected number of items mapped to slot 1 is

$$E(X) \quad = \quad \sum_{i=1}^{n} E(X_i) \quad = \quad \frac{n}{k}.$$

But this is obvious in any case. As mentioned earlier, the expected number of items is the same for every slot. Writing Y_j for the number of items mapped to slot j, we have $Y = \sum_{j=1}^{k} Y_j = n$. Similarly,

$$E(Y) \quad = \quad \sum_{j=1}^{k} E(Y_j) \quad = \quad n.$$

Since the expectations are the same for all slots, we therefore have $E(Y_j) = \frac{n}{k}$, for each j.

Empty slots. The probability that slot j remains empty after mapping all n items is $(1 - \frac{1}{k})^n$. Defining

$$X_j \quad = \quad \begin{cases} 1 & \text{if slot } j \text{ remains empty;} \\ 0 & \text{otherwise,} \end{cases}$$

we thus get $E(X_j) = (1 - \frac{1}{k})^n$. The number of empty slots is $X = X_1 + X_2 + \ldots + X_k$. Hence, the expected number of empty slots is

$$E(X) \quad = \quad \sum_{j=1}^{k} E(X_j) \quad = \quad k\left(1 - \frac{1}{k}\right)^n .$$

For $k = n$, we have $\lim_{n\to\infty}(1 - \frac{1}{n})^n = e^{-1} = 0.367\ldots$ In this case, we can expect about a third of the slots to remain empty.

Collisions. The number of collisions can be determined from the number of empty slots. Writing X for the number of empty slots, as before, we have $k - X$ items hashed without collision and therefore a total of $n - k + X$ collisions. Writing Z for the number of collisions, we thus get

$$\begin{aligned} E(Z) &= n - k + E(X) \\ &= n - k + k\left(1 - \frac{1}{k}\right)^n . \end{aligned}$$

For $k = n$, we get $\lim_{n\to\infty} n(1 - \frac{1}{n})^n = \frac{n}{e}$. In words, about a third of the items cause a collision.

Filling all slots. How many items do we need to map to the k slots until they store at least one item each? For obvious reasons, this question is sometimes referred to as the coupons collector problem. The crucial idea here is to define X_j equal to the number of items it takes to go from $j - 1$ to j filled slots. Filling the j-th slot is an infinite Bernoulli process with success probability equal to $p = \frac{k-j+1}{k}$. Last lecture, we learned that the expected number of trials until the first success is $\frac{1}{p}$. Hence, $E(X_j) = \frac{k}{k-j+1}$. The number of items needed to fill all slots is $X = X_1 + X_2 + \ldots + X_k$. The expected number is therefore

$$\begin{aligned} E(X) &= \sum_{j=1}^{k} E(X_j) \\ &= \sum_{j=1}^{k} \frac{k}{k-j+1} \\ &= k\sum_{j=1}^{k} \frac{1}{j} \\ &= kH_k . \end{aligned}$$

As mentioned during last lecture, this is approximately k times the natural logarithm of k. More precisely, we have $k\ln(k+1) \le kH_k \le k(1 + \ln k)$.

Summary. Today, we applied basic probabilistic concepts to get insight into the performance of hashing algorithms. Most importantly, we computed expectations by decomposing random variables into indicator random variables, which we added using the Linearity of Expectation.