Aitch	*Ex*
Are	*Eye*
Ay	*Gee*
Bee	*Jay*
Cue	*Kay*
Dee	*Oh*
Double U	*Pea*
Ee	*See*
Ef	*Tee*
El	*Vee*
Em	*Wy*
En	*Yu*
Ess	*Zee*

— Sidney Harris, "The Alphabet in Alphabetical Order"

8 Hash Tables (February 15)

8.1 Introduction

A *hash table* is a data structure for storing a set of items, so that we can quickly determine whether an item is or is not in the set. The basic idea is to pick a *hash function* h that maps every possible item x to a small integer $h(x)$. Then we store x in slot $h(x)$ in an array. The array is the hash table.

Let's be a little more specific. We want to store a set of n items. Each item is an element of some finite[1] set $\mathcal{U}$ called the *universe*; we use u to denote the size of the universe, which is just the number of items in $\mathcal{U}$. A hash table is an array $T[1\,..\,m]$, where m is another positive integer, which we call the *table size*. Typically, m is much smaller than u. A *hash function* is a function

$$h\colon \mathcal{U} \to \{0, 1, \ldots, m-1\}$$

that maps each possible item in $\mathcal{U}$ to a slot in the hash table. We say that an item x *hashes* to the slot $T[h(x)]$.

Of course, if $u = m$, then we can always just use the trivial hash function $h(x) = x$. In other words, use the item itself as the index into the table. This is called a *direct access table* (or more commonly, an *array*). In most applications, though, the universe of possible keys is orders of magnitude too large for this approach to be practical. Even when it is possible to allocate enough memory, we usually need to store only a small fraction of the universe. Rather than wasting lots of space, we should make m roughly equal to n, the number of items in the set we want to maintain.

What we'd like is for every item in our set to hash to a different position in the array. Unfortunately, unless $m = u$, this is too much to hope for, so we have to deal with *collisions*. We say that two items x and y *collide* if the have the same hash value: $h(x) = h(y)$. Since we obviously can't store two items in the same slot of an array, we need to describe some methods for *resolving* collisions. The two most common methods are called *chaining* and *open addressing*.

[1]This finiteness assumption is necessary for several of the technical details to work out, but can be ignored in practice. To hash elements from an infinite universe (for example, the positive integers), pretend that the universe is actually finite but very very large.

Then again, in *real* practice, the universe *is* actually finite but very very large. For example, on most modern computers, there are only 2^{64} integers!