Computer Science 2530
April 17, 2020

Happy Friday, April 17.

We are very close to the end. Today, we will look briefly at hash functions.

Tables

Page **43A** discusses *tables*. Think of a table as a telephone book or a dictionary. You look up one piece of information, such as a name, and you get some associated information, such as a telephone number.

With minor modifications, height-balanced binary search trees can be turned into an implementation of tables where lookup, insertion and removal for a table of size n takes $\Theta(\log_2(n))$ time even in the worst case. That is very fast. Since $\log_2(1,000,000,000)$ is about 30, the cost is small even if you have a billion things in your table.

(Recall that $\Theta(f(n))$ means approximately proportional to $f(n)$. Remember that it takes $\Theta(\log_2(n))$ time per operation for a height-balanced binary search tree. You will see that on a test.)

But can we do better still? It turns out we can, if instead of worst case we are willing to settle for average case.

Hash functions

Page **43B** describes the idea of a *hash function*, and gives an example of one. Hash functions are not pretty, and you do not need to understand the example. The idea is that a hash function takes a value (typically a string) and gives a large integer.

A hash function $h(x)$ is a function, and if you compute h("kangaroo") twice, you will get exactly the same result both times. But a hash function should *appear* to choose its answers at random, if you don't look too carefully.

Hash tables

A hash table is an array. Suppose the array has size N. The idea is to store string s (and any information associated with s) at index

$h(s)$ mod n in the array. If you want to look up s, you recompute $h(s)$ and look at index $h(s)$ mod n, and there is what you are looking for!

Of course, there is a catch. It can be the case that $h(s_1)$ mod N and $h(s_2)$ mod N are the same value. What then? That is called a *collision*.

That is not really such a big problem. We store a linked list at each index in the array. If five different strings are stored at index 1000, then the linked list at index 1000 will have 5 values in it. We can search the list.

A key feature of hash tables is that we try to keep the array size N *very roughly* the same as the number n of things in the table. Then, *on average*, there is only one thing in each linked list. (It is good enough for N and n to be within a factor of 10 of one another.) As more things are inserted, we need to make the array larger.

Since the hash function appears to be random, it spreads values roughly uniformly among the linked lists.

Assuming all goes well, the average time to do a lookup, insertion or removal in a hash table is $\Theta(1)$. That is, the average is a constant, *independent of the number of things in the table*. If you get nothing else out of this, remember that.

Suppose that you are Google and you have an unbelievably large amount of data scraped off the internet. You can see the appeal of hash tables. It does not matter how much information you are remembering. The time needed to look something up is always the same. That goes a long way toward explaining how Google can respond so quickly to a query.