Notes for Lecture 9

1 Hashing

We assume that all the basics about hash tables have been covered in 61B.

We will make the simplifying assumption that the keys that we want to hash have been encoded as integers, and that such integers are in the range $1, \ldots, M$. We also assume that collisions are handled using linked lists.

Suppose that we are using a table of size m, that we have selected a hash function $h : \{1, \ldots, M\} \rightarrow \{0, \ldots, m-1\}$ and that, at some point, the keys $y_1, \ldots, y_n$ have been inserted in the data structure, and that we want to find, or insert, or delete, the key x. The running time of such operation will be a big-Oh of the number of elements y_i such that $h(y_i) = h(x)$.

No matter what h does, if $M > m(n+1)$, there will always be a worst-case situation where $y_1, \ldots, y_n, x$ are all mapped by h into the same bucket, and then the running time of find, insert and delete will be $\Theta(n)$. However, in practice, hash tables work well. In order to explain the real behavior of hash tables we need to go beyond a worst-case analysis, and do a probabilistic analysis.

A simple analysis can be done assuming that the keys to be inserted in the table come from the uniform distribution over $\{1, \ldots, M\}$. It is not hard to come up with a function $h : \{1, \ldots, M\} \rightarrow \{0, \ldots, m-1\}$ with the property that if $y_1, \ldots, y_n, x$ are uniformly and independently distributed over $\{1, \ldots, M\}$, then $h(y_1), \ldots, h(y_n), h(x)$ are uniformly (or almost uniformly) distributed over $\{0, \ldots, m-1\}$, and then argue that, on average, the number of y_i such that $h(x) = h(y_i)$ is about n/m, and so an operation on x can be performed in $O(1)$ time assuming that, say, $m = 2n$.

In practice, however, inputs do not come from a uniform distribution, and choices of h that make the above proof work may or may not work well if the inputs come from different distributions.

A much more sound approach is to consider the behavior of the data structure on arbitrary (worst-case) inputs, but to let randomness come in the definition of the hash function h. Then we will look at the average running time of find, insert and delete operations, but this time the average will be over the randomness used by the algorithm, and there will be no assumption on the way the input is distributed. This kind of analysis is the object of today's lecture.

2 Universal Hashing

We want to consider hash functions whose definition involves random choices. Equivalently, we consider *families* of functions, and consider the randomized process of selecting at random a function from the family.

A collection H of hash functions $h : \{1, \ldots, M\} \to \{0, \ldots, m-1\}$ is said to be *2-universal* if for every two different $x, y \in \{1, \ldots, M\}$ we have

$$\mathbf{Pr}_{h \in H}[h(x) = h(y)] \leq \frac{1}{m}$$

(Note that the CLR/CLRS definition has '=' instead of '$\leq$')

Consider the following construction. Fix a prime $p > M$. It is known that there is at least one (in fact, there are several) prime between M and $2M$. A procedure for finding such a p could be to generate at random a number between M and $2M$ and then test for primality. (There is an efficient randomized algorithms for testing primality, but we will probably not see it in this course.)

Then define, for every $a \in \{1, \ldots, p-1\}$ and every $b \in \{0, \ldots, p-1\}$ the function

$$g_{a,b}(x) = ax + b \pmod{p}$$

For each such g we define a hash function

$$h_{a,b}(x) = g_{a,b}(x) \pmod{m} ,$$

we will prove that this construction is 2-universal

Lemma 1

For every fixed two distinct values $x, y \in \{0, \ldots, m-1\}$, the number of pairs a, b such that $h_{a,b}(x) = h_{a,b}(y)$ is at most $p(p-1)/m$.

Proof: We will consider all possible $s, t \in \{0, \ldots, p-1\}$ such that $s = t \pmod{m}$, and then for each such pair (s, t), we will count how many pairs (a, b) are there such that $h_{a,b}(x) = s$ and $h_{a,b}(y) = t$.

The number of choices for the pair (s, t) is $p(\lceil p/m \rceil - 1) < p(p-1)/m$.

For each pair s, t we are now asking how many values $a, b \in \{0, \ldots, p-1\}$ are there that satisfy the following system

$$\begin{cases} ax + b = s \pmod{p} \\ ay + b = t \pmod{p} \end{cases}$$

Algebra tells us that, if p is prime, the solution is unique. So the number of pairs (a, b) for which $h_{a,b}(x) = h_{a,b}(y)$ is at most $p(p-1)/m$. $\Box$

Since there are $p(p-1)$ functions in our family, the probability that $h_{a,b}(x) = h_{a,b}(y)$ is at most $1/m$, and so our family is indeed 2-universal.

3 Hashing with 2-Universal Families

Suppose now that we pick at random h from a family of 2-universal hash functions, and we build a hash table by inserting elements $y_1, \ldots, y_n$. Then we are given a key x that we want to find, insert or delete from the table. What will the expected number of collisions between x and $y_1, \ldots, y_n$ be?

Lemma 2

Let H be a 2-universal family of hash functions mapping $\{1,\dots,M\}$ into $\{0,\dots,m-1\}$, and let $x, y_1, \dots, y_n$ be elements of $\{1,\dots,M\}$.

If we pick h at random from H, then the average number of elements y_i such that $h(x) = h(y_i)$ is at most n/m; in symbols

$$\mathbf{E}[|\{i : h(x) = h(y_i)\}|] \leq n/m$$

Proof: Call C the random variable (depending on the choice of h) that counts the number of collisions between $h(x)$ and $h(y_1), \dots, h(y_n)$. 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 for every y

$$\begin{aligned}\mathbf{E}[C_y] &= 0 \cdot \mathbf{Pr}[h(x) \neq h(y)] + 1 \cdot \mathbf{Pr}[h(x) = h(y)] \\ &= \mathbf{Pr}[h(x) = h(y)] \leq \frac{1}{m}\end{aligned}$$

Since $C = \sum_{i=1}^n C_{y_i}$, we have $\mathbf{E}[C] = \mathbf{E}[\sum_{i=1}^n C_{y_i}] = \sum_{i=1}^n \mathbf{E}[C_{y_i}] \leq n/m$ □

So, if we choose $m = \Theta(n)$, each operation has $O(1)$ average running time.