CPSC 536N: Randomized Algorithms 2011-12 Term 2

Lecture 18

Prof. Nick Harvey *University of British Columbia*

In the last lecture we defined the notion of pairwise independence, and saw some ways to use this concept in reducing the randomness of a randomized algorithm. Today we will apply these ideas to designing hash functions and "perfect" hash tables.

1 Universal Hash Families

Last time we showed how to create a joint distribution on random variables $\{ Y_i : i \in U \}$ such that

- each Y_i is uniformly distributed on some finite set T, and
- the joint distribution on the Y_i's is pairwise independent.

Let S be the sample space underlying the Y_i's. Drawing a sample s from S determines values for all the Y_i's. We can think of this another way: after choosing $s \in S$, then associated with every $i \in U$ there is a value in T (namely, the value of Y_i).

Formally, for every $s \in S$ there is a function $h_s : U \to T$ given by $h_s(i) = Y_i$. (The random variable Y_i is implicitly a function of s.) This family of functions $\{ h_s : s \in S \}$ satisfies the following important property

$$\Pr_{s \in S}[\, h_s(x) = \alpha \wedge h_s(y) = \beta \,] = \frac{1}{|T|^2} \qquad \forall x \neq y \in U, \quad \forall \alpha, \beta \in T.$$

A family of functions satisfying this property is called a **2-universal hash family**. So we can restate our construction from last time as follows.

Theorem 1 *For any n, there exists a 2-universal hash family with $|S| = 2^{2n}$, $|U| = 2^n$ and $|T| = 2^n$. Sampling a function from this family requires $O(n)$ mutually independent, uniform random bits. Consequently, each hash function can be represented using $O(n)$ bits of space. Evaluating any hash function in this family at any point of U takes only a constant number of arithmetic operations involving $O(n)$-bit integers.*

One can easily modify this construction to handle any U and T with $|U| \leq 2^n$, $|T| \leq 2^n$ and $|T|$ a power of two.

2 Perfect Hashing

Let U be a "universe" of items. We will assume that our computational model has words of $\Theta(\log |U|)$ bits and that any standard arithmetic operation involving $\Theta(\log |U|)$-bit words takes $O(1)$ time.

Suppose we wish to store a given subset $N \subseteq U$ as a "dictionary", so that we can efficiently test whether any element $x \in U$ belongs to N. There are many well-known solutions to this problem. For example, a balanced binary tree allows us to store the dictionary in $O(|N|)$ words of space, while search operations take $O(\log |N|)$ time in the worst case. (Insertion and deletion of items also take $O(\log |N|)$ time.)

We will present an alternative dictionary design based on hashing. This is a "static dictionary", which does not allow insertion or deletion of items.

Theorem 2 *There is a randomized algorithm that, given a set $N \subseteq U$, stores the set N in a data structure using $O(|N|)$ words of space. There is a deterministic algorithm that, given any item $x \in U$, can search for x in that data structure in $O(1)$ time in the worst case.*

Let T be any set with $T \subseteq U$ and $|T|$ a power of two. Theorem 1 gives us a 2-universal hash family mapping U to T for which any hash function can be evaluated in $O(1)$ time.

Using this family, consider the following simple design for our dictionary. First create a table of size $|T|$ and randomly choose a function h_s from the hash family. Can we simply store each item $x \in N$ in the table in location $h_s(x)$? This would certainly be very efficient, because finding x in the table only requires evaluating $h_s(x)$, which takes only $O(1)$ time. But the trouble of course is that there might be **collisions**: there might be two items $x, y \in N$ such that $h_s(x) = h_s(y)$, meaning that x and y want to lie in the same location of the table.

Most likely you have discussed hash tables collisions in an undergraduate class on data structures. One obvious way to try to avoid collisions is to take $|T|$ to be larger; the problem is that this increases the storage requirements, and we are aiming for a data structure that uses $O(|N|)$ words. Alternatively one can allow collisions to happen and use some other technique to deal with them. For example one could use *chaining*, in which all items hashing to the same location of the table are stored in a linked list, or *linear probing*, in which some items hashing to the same location and relocated to nearby locations. Unfortunately neither of those techniques solves our problem: when the table size is $O(|N|)$ both of those techniques will typically require super-constant time to search the table in the worst case.

The solution we present today is based on a simple two-level hierarchical hashing scheme. At the top-level, a single hash function is used to assign each item $x \in N$ to a "bucket" in the top-level table. Then, for every bucket of the top-level table we create a new second-level hash table to store the items that hashed to that bucket.

Collisions. Using the pairwise independence property, we can easily determine the probability of any two items having a collision. For any unordered pair of distinct items $\{x, y\} \subset U$,

$$\Pr_{s \in S}[h_s(x) = h_s(y)] \;=\; \sum_{i \in T} \Pr_{s \in S}[\, h_s(x) = i \;\wedge\; h_s(y) = i \,] \;=\; \sum_{i \in T} \frac{1}{|T|^2} \;=\; \frac{1}{|T|}.$$

So the expected total number of collisions is

$$\mathrm{E}[\#\text{ collisions}] \;=\; \sum_{\{x,y\} \subset N,\, x \neq y} \Pr_{s \in S}[h_s(x) = h_s(y)] \;=\; \binom{|N|}{2} / |T|. \tag{1}$$

We would like to carefully choose the size of T in order to have few collisions. A natural choice is to let $|T|$ be $|N|^2$ (rounded up to a power of two), so

$$\mathrm{E}[\#\text{ collisions}] \leq 1/2 \qquad \Longrightarrow \qquad \Pr[\text{no collisions}] \geq 1/2, \tag{2}$$

by Markov's inequality. The problem is that storing this T would require $\Theta(|N|^2)$ space.

Our design. Instead, we will choose $|T|$ to be $|N|$ (rounded up to a power of two). This will typically result in some collisions, but the second-level hash tables are used to deal with those collisions. The top-level hash function is denoted $h : U \to T$, and the elements of T are called **buckets**. By (1),

$$\mathrm{E}[\#\text{ collisions}] \leq |N|/2 \qquad \text{and} \qquad \Pr[\#\text{ collisions} \leq |N|] \geq 1/2.$$

Let us condition on that event happening: the number of collisions of the top-level hash function h is at most $|N|$. For every bucket $i \in T$, let

$$N_i \;=\; \{\, x \in N \;:\; h(x) = i \,\}$$

be the set of items that are hashed to bucket i by the top-level hash function. Let $n_i = |N_i|$. For each $i \in T$ we will sample a new second-level hash function mapping $h_i : U \to T_i$ where $|T_i| = n_i^2$. With probability at least $1/2$ there will be no collisions of the hash function h_i, as shown by (2). We will repeatedly sample h_i and count its number of collisions until finding a function h_i with no collisions.

Search operation. Every item in N is stored in exactly one location in these second-level tables. So to search our data structure for an item $x \in U$ we follow a very simple procedure. First we compute $i = h(x)$, the top-level bucket containing item x. Then we compute $h_i(x)$, which gives the location of x in the second-level table T_i. This process takes $O(1)$ time.

Space requirements. The only thing left to analyze is the space requirements. The size of the top-level table T is at most $2|N|$, by construction. The total size of the second-level tables is

$$\sum_{i \in T} n_i^2 \;\le\; \sum_{i \in T} \left(2\binom{n_i}{2} + n_i \right) \;=\; |N| + 2\sum_{i \in T} \binom{n_i}{2} \;=\; |N| + 2 \cdot \#\text{ collisions} \;\le\; 3 \cdot |N|.$$

We also need to need to write down the description of the hash functions themselves. By Theorem 1 each hash function can be represented using only a constant number of words. So the total space required is $O(|N|)$ words.