princeton univ. F'08 cos 521: Advanced Algorithm Design

Lecture 2: Hashing

Lecturer: *Sanjeev Arora* Scribe:*Sushant Sachdeva*

1 Hash Function : Preliminaries

In the last lecture, we defined hash functions to construct a solution to the dictionary problem.

Definition 1 (Hash Function) *A hash function is a* "random looking" *function mapping values from a domain* $[D]$ *to its range* $[R]$

The solution to the dictionary problem using Hashing is to store the set $S \subseteq [D]$ in an array of size $\Theta(S)$, storing each element x of S in a list at the location $h(x)$. If the hash function is *good*, we can show that the expected time for `add/delete/find` operations on S is $O(1)$.

A simple scheme for constructing a family of good hash functions is as follows. Pick a prime $p \in (D, 2D]$ (such a prime is guaranteed to exist). Randomly choose $a \neq 0, b \in \mathbf{Z}_p$. The hash function is defined as follows.

$$h : x \to (ax + b \mod p) \mod R$$

Definition 2 (Pair-wise Independent family of hash functions) *A family of hash functions* $\mathcal{H}$ *is called pairwise independent if* $\forall x \neq y \in [D]$ *and* $\forall a_1, a_2 \in [R]$*, we have*

$$\Pr_{h \in \mathcal{H}}[h(x) = a_1 \wedge h(y) = a_2] = \frac{1}{R^2}$$

Note: Whenever we write $h \in \mathcal{H}$, we shall assume the uniform distribution.

We now take a look at the family of hash functions we defined above and show that they are pairwise independent (almost). We fix $x \neq y \in [D]$. We need to observe that since $\mathbf{Z}_p$ is a field, there is a one to one mapping between pairs $(a, b) \in \mathbf{Z}_p^* \times \mathbf{Z}_p$ and pairs $(c = ax + b \mod p, d = ay + b \mod p) \in \mathbf{Z}_p \times \mathbf{Z}_p$ with $c \neq d$. Thus,

$$\Pr_{h \in \mathcal{H}}[h(x) = a_1 \wedge h(y) = a_2] = \Pr_{r \neq s \in \mathbf{Z}_p^2}[r \mod R = a_1 \wedge s \mod R = a_2]$$

Since there are $p(p-1)$ such pairs of (r, s), we get

$$\frac{1}{p(p-1)}\left(\left\lfloor \frac{p}{R} \right\rfloor\right)^2 \leq \Pr_{r \neq s \in \mathbf{Z}_p^2}[r \mod R = a_1 \wedge s \mod R = a_2] \leq \frac{1}{p(p-1)}\left(\left\lfloor \frac{p}{R} \right\rfloor + 1\right)^2$$

Thus we get that the above family is a (approximately) pairwise-independent family of hash functions.

Definition 3 (Universal Family of Hash Functions) *A family of hash functions $\mathcal{H}$ is called Universal if* $\forall x \neq y \in [D]$

$$\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq \frac{1}{R}$$

The above definition can be weakened to say that a family is *c-Universal* if the upper bound on the probability is $\frac{c}{R}$.

By letting $a_1 = a_2$ in definition 2 and summing over all the possible values of a_1, it is easy to see that a Pair-wise independent family of hash functions is also universal.

As considered in the last lecture, if we have $S \subseteq D, R = 2|S|$ and we pick $h \in_R \mathcal{H}$ where $\mathcal{H}$ is a c-universal family, we can show that the expected length of the size of each list is $O(1)$.

$$\mathop{\mathbf{E}}_{h \in \mathcal{U}}[\text{length of } h(x)] = \sum_{y \in S} \Pr_h[h(y) = h(x)] \leq \frac{c|S|}{R} = O(1)$$

Since the expected time for each of the `add / find / delete` operations is the time to compute $h(x)$ plus the expected length of the list at $h(x)$, the above says that if we can compute $h(x)$ efficiently for a c-universal family, the expected time for all the operations will be $O(1)$.

1.1 What if the size of the set S is not known in advance?

We can start with a constant R, say 10. We wish to maintain $|R| \sim 2|S|$. So when S becomes larger than $|R|/2$, we can double the size of R and use a new hash function to re-hash the existing elements and also to hash the elements which are subsequently added. If we assume that the final size of R is m, the amount of extra work that is done in all the resizing and re-hashing is

$$\Theta(m + \frac{m}{2} + \frac{m}{4} + \ldots) = \Theta(m)$$

Thus, on average, it adds only a constant amount of time per element.

2 Variants and Applications of Hashing

2.1 Perfect Hashing

A hash function $h : [D] \to [R]$ is said to be perfect for S if $\forall x \neq y \in S, h(x) \neq h(y)$.

In other words, perfect hashing means that there are no collisions or that h is a one-to-one function on S.

If $\mathcal{H}$ is a universal hash family and $|S| \leq \frac{\sqrt{|R|}}{2}$, then

$$\begin{aligned}\Pr_{h\in\mathcal{H}}[h \text{ is not perfect for } S] &= \Pr_{h\in\mathcal{H}}[\exists x \neq y \in S \text{ such that } h(x) = h(y)] \\ &\leq \sum_{x\neq y\in S} \Pr_{h\in\mathcal{H}}[h(x) = h(y)] \\ &\leq \frac{|S|(|S|-1)}{2|R|} \leq \frac{1}{8}\end{aligned}$$

Thus,

$$\Pr_{h\in\mathcal{H}}[h \text{ is perfect for } S] \geq \frac{7}{8}$$

We show that the above bound is in some sense optimal up to a constant factor. We show that if we pick h randomly from the space of all the functions from $[D]$ to $[R]$, in order for the probability of h being perfect, for a randomly chosen S, to be bounded away from zero, $|S| = \Theta(\sqrt{R})$. We fix an $S \subseteq \mathcal{U}$.

$$\begin{aligned}\Pr_{h}[\forall x \neq y \in S, h(x) \neq h(y)] &= 1.\left(1-\frac{1}{R}\right).\left(1-\frac{2}{R}\right)\ldots.\left(1-\frac{|S|-1}{R}\right) \\ &\geq e^{-\frac{1}{R}}.e^{-\frac{2}{R}}\ldots.e^{-\frac{|S|-1}{R}}. \\ &\geq e^{-\frac{|S|(|S|-1)}{2R}}\end{aligned}$$

Thus in order for the above probability to be a constant, say $1/2$, we get $|S| \approx \sqrt{2R\ln 2}$.

2.2 Finger Printing

Suppose you are given 2 files, X and Y, which are stored at two different locations and you are asked to determine if the two files are non-identical. The goal is to achieve this with small amount of data communication.

A good way to solve the above problem is to compute a hash function, h on the files which maps the contents of each file, say X, to a value, say $h(X)$ which is say k bits long. We communicate the hash function of the file to the other location and compare the two hash functions and announce that the two files are different iff the two hash values are different. If we assume that the family of hash functions is universal, we have.

$$\begin{aligned}\Pr_{h\in\mathcal{H}}[\varepsilon \text{ (Error)}] &= \Pr_{h\in\mathcal{H}}[h(x) \neq h(y)|x = y] + \Pr_{h\in\mathcal{H}}[h(x) = h(y)|x \neq y] \\ &\leq 0 + \frac{1}{2^k} = \frac{1}{2^k}\end{aligned}$$

2.3 Cryptographic Hash Functions

Cryptographic hash functions are the hash functions where it is hard to find a pre-image of a hash value. Thus, a hash function h is said to be a cryptographic hash function if given

$a \in [R]$, it is intractable to compute an $x \in [D]$ such that $h(x) = a$. A stronger notion is also used commonly which says that it is intractable to find $x \neq y \in [D]$ such that $h(x) = h(y)$.

Such functions can be constructed by composing a one-way function with a hash function.

$$x \xrightarrow[\text{One way function}]{f} f(x) \xrightarrow[\text{Hash Function}]{g} g(f(x)) \text{ (Cryptographic Hash)}$$

These cryptographic hash functions have applications in protecting systems against virus attacks. The "anti-virus" maintains a hash value of the system files (or applications) and periodically checks if the files have been modified by recomputing the hash values and comparing them to the stored hash values. In order for this procedure to be secure, the virus / bug / trojan must not be able to modify the code in such a way that the hash value of the code remains the same. Thus, Cryptographic hash functions are very useful in this scenario.

2.4 k-wise Independent Family of hash functions

A family of hash functions is called a k-wise Independent family of hash functions if $\forall a_1, \ldots, a_k \in [R]$ and distinct $x_1, \ldots, x_k \in [D]$,

$$\mathop{\mathbf{Pr}}_{h \in \mathcal{H}}[h(x_1) = a_1 \wedge h(x_2) = a_2 \wedge \ldots \wedge h(x_k) = a_k] = \frac{1}{R^k}$$

This is a stronger assumption than pair-wise independence (which in turn is stronger than universal hashing) and permits us to prove better bounds on bucket sizes (see exercise problems). Hash functions from such a family provide some robustness against the situation where the events are somewhat correlated. They are also useful in distributed hashing.

2.5 Set Size Estimation[1]

The situation here is as follows. There are two parties, a prover P and a verifier V. Both the parties are concerned with a set S of elements from a universe $\mathcal{U}$ which satisfy a predicate Φ. This set is hard to compute ($\mathcal{U}$ is huge and S is relatively small) but simple to verify (i.e. to evaluate Φ). V wishes to obtain a reliable estimate of the size of S with the help of P, which has such a set of elements. All this has to be done in an efficient manner. (You can imagine the above in a situation as follows. V could be a company that wishes to estimate the number of people that own a particular brand of car and P has a list of owners and helps to convince V that the number of such people is large). For concreteness, we presume that V wishes to be reasonably sure that the set S has at least k elements and not as few as $k/2$.

First, we observe that the ratio $1/2$ is arbitrary and we can amplify the gap by working with tuples of elements rather than individual elements (in this case we will test if every element in the tuple satisfies Φ). For the purpose of the rest of the analysis, we shall assume that the gap is a factor of $1/8$.

One possible solution, as given in a paper by Goldwasser and Sipser ([1]) is that V randomly picks a hash function h which maps the elements of $\mathcal{U}$ to a domain D of size $k/2$. V them randomly selects an element $a \in D$ and asks P to show an element $x \in C$ such that

$h(x) = a$. We can show that if C has at least k elements, P will be able to produce such an element with probability at least $1/2$.

$$\mathop{\mathbf{Pr}}_{h \in \mathcal{H}, a}[\nexists x \in D, h(x) = a] \leq \frac{1}{2}$$

Moreover, if C has at most $k/8$ elements, the probability of existence of such an x is at most $1/4$.

$$\mathop{\mathbf{Pr}}_{h \in \mathcal{H}, a}[\exists x \in D, h(x) = a] \leq \frac{|S|}{k/2} \leq \frac{1}{4}$$

The gap in the probabilities can be amplified by repeating the test and using an appropriate acceptance condition.

The following is the classical view of the problem in terms of what are known as *Interactive Proofs*. The situation here is as follows. There are two parties, a computationally unbounded prover P and a probabilistic polynomial time verifier V. The two parties can exchange messages and they are looking for a set S. S,Φ and $\mathcal{U}$ are as above. The game consists of interactions between them. P thinks that she/he has a set where $|C| \geq k$ whereas V does not trust P and thinks that the set that P claims contains less than $k/2$ elements. Both of them have the power to test whether a specific element satisfies Φ. P wins the game if she/he can convince V that the set contains at least k elements and not $k/2$ as V believes. This is just a restatement of the above and can be solved in an identical manner.

References

[1] S Goldwasser and M Sipser. Private coins versus public coins in interactive proof systems. pages 59–68, 1986.