

Lecture XI

HASHING

Hashing is a practical technique to implement a dictionary. Its space usage can be made optimal, $\mathcal{O}(n)$. Under some simplifying probabilistic assumptions, its search times can also be shown to be optimal, $\Theta(1)$. We look at some extensions of the basic hashing framework: including universal hash sets and extendible hashing. Another important special topic is the problem of optimal static hashing.

Hash is one of the oldest and most widely used data structures in computer science. The first paper on hashing was by Arnold I Dumeyin in **Computers and Automation**, 5(12), Dec 1956. Peterson [9] is another early major paper. The survey of Robert Morris (1968) mark the first time that the term “hashing” appeared in publication (although the concept of hashing was well-known by then); this paper also introduce methods beyond linear probing. See Knuth [7] for a survey of the early history and the basic techniques in hashing.

Q: *What is the most important data structure technique that arises in your research in Yahoo?*

A: *Hashing, hashing, hashing*

— Udi Manber, Chief Scientist at Yahoo.com
(responding to a question in SODA Conference 2001)

§1. Elements of Hashing

Recall that a dictionary (§III.2) is an abstract data type that stores a set of items under three basic operations: `lookUp`, `insert` and `delete`. Each item is a pair $(Key, Data)$. For simplicity, assume that distinct items have identical keys.

- `insert(Item)`: returns a node which points to where item is stored.
- `lookUp(Key)`: returns a node which points to item with this key. If no such item exists, return the nil node.
- `delete(Node)`: removes item stored at this node. This *Node* may be obtained from a prior `lookUp`.

There are two important special cases: if a dictionary supports insertions and lookups, but not deletions, we call it a **semi-dynamic dictionary**. If it supports only lookups, but not insertions or deletions, it is called a **static dictionary**. For instance, conventional books (such as lexicons, encyclopaedias, phone directories) are static dictionaries for ordinary users. We might view a personal address book as a semi-dynamic dictionary.

The following notations will be used in this lecture. Let U be the **universe of keys** (sometimes called **key space**). At any moment $t \geq 0$, the dictionary will only contain some proper subset $K_t \subseteq U$ of keys. Let $n_t = |K_t|$. Also, let $u = |U|$. We usually omit the subscript t and simply write K and n .

(H1) *The first premise of hashing is*

$$n = |K| \ll |U| = u. \tag{1}$$

For example, let $U = [0..9]^9$ represent the set of all possible social security numbers in the USA. If a personnel database uses social security numbers as keys then the number n of actual keys is much less than $u = 10^9$.

There are three simple ways to implement a dictionary: (a) as a linked list, (b) as an array, and (c) as a binary search tree. Using linked lists, the space used is $\Theta(n)$ to store the keys in K , but the time to lookup a key may take $\Theta(n)$ time. The space is optimal but the time is considered too slow. Using an array, the space is $\Theta(u)$ and the time is $\mathcal{O}(1)$; this time is optimal but the space usage is probably suboptimal if not infeasible. If we use binary search trees, we can achieve $\mathcal{O}(n)$ space with $\mathcal{O}(\log n)$ search times.

Hashing provides another way to implement dictionaries in which space and time are both optimal, although the time is optimal only in an expected sense. It is not hard to implement and is efficient when correctly implemented. Every practitioner should have some hashing knowledge under his or her belt.

The basic hashing scheme is as follows: it uses an array called the **hash table** $T[0..m-1]$. We note that there is no obvious relationship between m and n : their relationship depends on the particular hashing technique to be used. E.g., it is not always true that $m \geq n$ although in most situations, $m = \Theta(n)$. Each entry in this table is called a **slot**. The key of an item is used to compute an index into the hash table. So another key element of hashing is the use of a **hash function**

$$h : U \rightarrow \mathbb{Z}_m$$

from U to array indices. Recall that $\mathbb{Z}_m = \{0, 1, \dots, m-1\}$. We say a key k is **hashed** to the **hash value** $h(k)$.

(H2) *The second premise of hashing is that the hash value of any key can be evaluated quickly. In complexity analysis, we usually assume this takes $\mathcal{O}(1)$ time.*

Under the assumption (1), the map h will be many-one in a very “strong” sense. Two keys $x, y \in U$ are in **collision** if $x \neq y$ but $h(x) = h(y)$. If no pairs in K are in collision, we could of course simply store $k \in K$ in slot $T[h(k)]$. But in general, some **collision resolution scheme** must be deployed. Different collision resolution schemes give rise to different flavors of hashing.

Since we do not want to make any assumptions about the set K to be stored, the best we can hope for is that h distributes the set K evenly among the slots. That is, for all $i, j \in \mathbb{Z}_m$, we want the size of the sets $h^{-1}(i) \cap K$ and $h^{-1}(j) \cap K$ to be approximately equal. We say h is **K -perfect** if

$$| |h^{-1}(i) \cap K| - |h^{-1}(j) \cap K| | \leq 1. \quad (2)$$

In case $K = U$, we simply¹ say h is **perfect**. Note that it is easy to have a perfect hash function: just hash each $x \in U$ to a unique slot of the hash table. This gives $\mathcal{O}(1)$ worst case time for all the dictionary operations! The catch is that the space usage is u , and by assumption (H1), this is not feasible.

Let

$$[U \rightarrow \mathbb{Z}_m]$$

denote the set of all functions from U to \mathbb{Z}_m . We say a set $H \subseteq [U \rightarrow \mathbb{Z}_m]$ of functions is **n -perfect** if for every $K \subseteq U$ of size n , there is an $h \in H$ that is K -perfect.

Example: An everyday illustration of hashing is your (non-electronic) personal address book. Each item is a pair of the form $(name, address \& data)$. The hash function applied to a *name* returns the first letter in *name*. E.g., if *name* = “yap” then $h(\text{“yap”}) = \text{“y”}$. The collision resolution technique amounts to a linear search of the page allocated to that letter. Deletion is done by marking an item as deleted. If a page allocated to a letter is filled up, additional entries may be placed in an overflow area.

Example: A standard application of hashing in Computer Science is for looking up compiler symbol table. Here, for each symbol in a program, the compiler needs to lookup the associated procedure. These

¹This generalizes the definition sometimes used in the literature, where h is said to be “ K -perfect” if $|h^{-1}(j) \cap K| \leq 1$, i.e., there are no collisions for keys in K .

symbols are either keywords in the programming language or user defined names of variables, procedures, etc.

To summarize: in hashing, the fundamental decision of the algorithm designer is to choose a hash function $h : U \rightarrow \mathbb{Z}_m$. Here, U is normally given in advance but m is a design decision that is based on other parameters such as n , the maximum number of items that will be in the dictionary at any given moment. The second major decision is the choice of a collision resolution strategy.

Practical Construction of Hash Functions. A common response to the construction of hash functions is to “do something really complicated and mysterious”. Unfortunately, such schemes inevitably fail to perform as well as two simple and effective methods. Following Knuth [7], we call these the division and multiplication methods, respectively.

A) Division method: The simplest is to treat a key k as a non-negative integer, and to define

$$h(k) = k \bmod m.$$

So choosing a hash function amounts to selecting m . Usually, choosing m to be a prime number is a good idea. If we have some target value for m (say, $m \sim 2^{16} = 65536$), then we usually choose m to be a prime close to this target (in this case, $m = 65521$ or $m = 65537$). There is an obvious pitfall to avoid when choosing m : assuming k a d -ary integer, then it is a bad idea for m to be a power of k . This is because if $m = k^\ell$ then $h(k)$ is simply the low order ℓ digits of k . We usually like h to depend on all the digits of k . For example, if k is a sequence of ASCII characters then k can be viewed as a d -ary integer where $d = 128$. Since d is a power of 2 here, it is also a bad idea for m to be a power of 2.

B) Multiplication method. Let $0 < \alpha$ be an irrational number. Then define

$$h(k) = \lfloor m((k \cdot \alpha) \bmod 1) \rfloor. \quad (3)$$

Note that in this formula, substituting $\alpha - k$ for α (for any integer k) does not affect the hash function. An empirically a good choice is $\alpha = \phi$, the golden ratio. Numerically,

$$\phi = (1 + \sqrt{5})/2 = 1.61803\ 39887\ 49894\ 84820\ \dots$$

Remark that since $\phi > 1$, we might as well use $\alpha = \phi - 1 = 0.61803\dots$ in our calculations. With this choice, and for $m = 41$, we have

$$h(1) = \lfloor 25.339 \rfloor = 25, \quad h(2) = \lfloor 9.678 \rfloor = 9, \quad h(3) = \lfloor 35.018 \rfloor = 35.$$

The choice $\alpha = \phi$ has an interesting theoretical basis as well, related to a remarkable theorem of Vera Turán Sós [7, p. 511] which we quote:

THEOREM 1 (THREE DISTANCE THEOREM) *Let α be an irrational number. Consider the $n+1$ subsegments formed by placing the n numbers*

$$\{\alpha\}, \{2\alpha\}, \dots, \{n\alpha\} \quad (4)$$

in the unit interval $[0, 1]$. Here, $\{x\} = x - \lfloor x \rfloor$ denotes the fractional part of a real number x . Then there are at most three different lengths among these $n+1$ subsegments. Furthermore, the next point $\{(n+1)\alpha\}$ lies in one of the largest subsegment.

It is evident that if $\{\alpha\}$ is very close to 0 or 1, then the ratio of the lengths of the largest to the smallest subsegments will be large. Hence it is a good idea to choose α so that $\{\alpha\}$ is closer to $1/2$ than to 0 or 1. It

turns out that the choice $\alpha = \phi = 1.61803\dots$ leads to the most evenly distributed subsegment lengths. The proof of this theorem uses continued fraction.

Let us discuss how to implement (3) in practice. Suppose we are using machine arithmetic of a computer. Most modern machines uses computer words with w bits where $w = 32, 64, 128$, etc. If we are designing the hash function, we have freedom to choose m , the size of the hash table. To exploit machine arithmetic, let us choose m so that $m = 2^k$ for some $1 < m \leq w$. We may freely choose α which we may assume satisfies $0 < \alpha < 1$. This determines an integer $0 < A < 2^w$ such $A/2^w$ is the binary fraction that is the closest w -bit approximation to α . CLAIM: $h(k)$ is equal to $2^{k-w}Ak$. We leave the proof for an exercise.

Here are some values of A when ϕ is the Golden Ratio: when $w = 32$, $A = 2, 654, 435, 769$.

Other methods. A very common hashing situation is where U is a variable length string (we do not like to place any *a priori* bound on the length of the string. Assuming each character is byte-size, we may take $U = \mathbb{Z}_{256}^*$ (an infinite key space). The exercises give a practical way to generate hash keys for this situation. In general, we can view each character as the coefficients of a polynomial $P(X)$ and we can evaluate this polynomial at some $X = a$ to give a hash code.

EXERCISES

Exercise 1.1:

- (a) Compute the sequence (4) (with $n = 15$) for the choice $\alpha = \phi$. You may compute to just 3 decimal positions.
 (b) Let

$$\ell_0 > \ell_1 > \ell_2 > \dots$$

be the new lengths of subsegments, in order of their appearance as we insert the points $\{n\phi\}$ (for $n = 0, 1, 2, \dots$) into the unit interval. For instance, $\ell_0 = 1, \ell_1 = 0.61803, \ell_2 = 0.38197$. Compute ℓ_i for $i = 0, \dots, 15$. \diamond

Exercise 1.2: Let K be the following set of 40 keys

A, ABOUT, AN, AND, ARE, AS, AT, BE, BOY, BUT,
 BY, FOR, FROM, HAD, HAVE, HE, HER, HIS, HIM, I,
 IN, IS, IT, NOT, OF, ON, OR, SHE, THAT, THE,
 THEY, THIS, TO, WAS, WHAT, WHERE, WHICH, WHY, WITH, YOU

Experimentally find some simple hash functions to hash K into $T[0..m-1]$, where m is chosen between 50 and 60. Your goal is to minimize the maximum size of a bucket (a bucket is the set of keys that are hashed into one slot). (You need not be exhaustive – but report on what you tried before picking your best choice.)

- (a) Use a division method.
 (b) Use the multiplication method with $\alpha = \phi$.
 (c) Invent some other hashing rule not covered by the multiplication or division methods. \diamond

Exercise 1.3: (Pearson [8]) A common hashing situation is the following: given a fixed alphabet $V = \mathbb{Z}_2^n$, we want to hash from $U = V^*$ to V . In practice, we may regard $U = \cup_{i=0}^s V^i$ for some large value of s . Typically, $n = 8$ (so V is a byte-size alphabet). Let $T : V \rightarrow V$ be stored as an array. Then we have a hash function h_T computed by the following:

```

HASH( $w$ ):
  Input:  $w = w_1w_2 \cdots w_n \in \Sigma^*$ .
  Output: hash value in  $h(w) \in \Sigma$ .
  1.  $v \leftarrow 0$ .
  2. for  $i \leftarrow 1$  to  $n$  do
  3.    $v \leftarrow T[v \oplus w_i]$ .
  4. return( $v$ ).

```

In line 3, $v \oplus w_i$ is viewed as a bitwise exclusive-or operation.

(a) Show that if $d(w, w') = 1$ then $h(w) \neq h(w')$. Here, $d(w, w')$ is the Hamming distance (the number of symbols in w, w' that differ).

(b) Use fact (a) to give a probe sequence $h(w, i)$ (where $i = 1, 2, \dots, N$) such that $(h(w, 1), h(w, 2), \dots, h(w, N))$ will cycle through all values of Σ .

(c) Suppose $T[i] = i$ for all i . What does this hash function compute?

(d) Suppose T is a random permutation of V . Show that h_T is not universal. HINT: consider the case $n = 1$ and $s = 3$. There are two choices for T . Find $x \neq y$ such that $\Pr\{h_T(x) = h_T(y)\} > 1/2$.

◇

Exercise 1.4: Here is an alternative and common solution in the hash function for the previous question.

```

HASH( $w$ ):
  Input:  $w = w_1w_2 \cdots w_n \in \Sigma^*$ .
  Output: hash value in  $h(w) \in \Sigma$ .
   $v \leftarrow 0$ .
  for  $i \leftarrow 1$  to  $n$  do
     $v \leftarrow (v + w_i) \bmod N$ .
  return( $v$ ).

```

Discuss the relative merits of the two methods (such as the efficiency of evaluating the hash function).

◇

 END EXERCISES

§2. Collision Resolution

The two basic methods of resolving collisions are called **chaining** and **open addressing**.²

Chaining Schemes. In chaining, the hash table $T[0..m-1]$ is only the initial entry into auxiliary structures which are usually linked lists (“chains”). There are two variants of chaining. The simpler variant is called **separate chaining**. Here each table slot $T[i]$ is used as the header of a linked list of items. The linked list is called a chain or bucket. An inserted item with key k will be put in the chain of $T[h(k)]$. Another variant is called **coalesced chaining**. Here each slot $T[i]$ is potentially the node of some chain, and all nodes are allocated from the hash table T . In this way, we avoid the use of dynamic memory management

²The terms **closed hashing** and **open hashing** are sometimes used instead of “open addressing” and “chaining”. We avoid this terminology here, as the juxtaposition of “open” and “close” for the same concept is obviously confusing.

which is required by separate chaining. To lookup a key k , we look for the head of the chain corresponding to the hash value $h(k)$. We first look at $T[h(k)]$, but this slot may be a node for some chain with different hash value. In that case, we search sequentially for the head of the chain for $h(k)$: that is, we search in slots $T[h(k) + 1]$, $T[h(k) + 2]$, etc, until we find the head of the chain for $h(k)$, or we find an “empty slot” (see explanation next). In the former case, we now follow the links for this chain looking for k . In the latter case, we conclude that k is not in the dictionary.

Here each slot in T is initially **empty**. When an empty slot is first used, it becomes **occupied**. It remains **occupied** until **deleted**. Deleted slots can become **occupied** again, deleted again, alternately. However, such slots never become empty again. This is important to ensure that we will not erroneously declare a search unsuccessful.

Open Addressing Schemes. Such schemes store all keys in the table T itself. Here, for key k , we need to generate an infinite sequence of hash table addresses:

$$h(k, 0), h(k, 1), h(k, 2), \dots \quad (5)$$

This is called the **probe sequence** for k , and it specifies that after the i th unsuccessful probe, we next search in slot $h(k, i)$. In practice, the sequence (5) is cyclic: for some $1 \leq m' \leq m$, $h(k, i) = h(k, i + m')$ for all i . Ideally, we want $m' = m$ and the sequence $(h(k, 0), h(k, 1), \dots, h(k, m - 1))$ to be a permutation of \mathbb{Z}_m . This ensures that we will find an empty slot if any exists. In open addressing, as in coalesced chaining, we need to classify slots as empty, occupied or deleted.

There are three basic methods for producing a probe sequence:

Linear Probing This is the simplest:

$$h(k, i) = h_1(k) + i \pmod{m}$$

where h_1 is the usual hash function. One advantage (besides simplicity) is that this probe sequence will surely find an empty slot if there is one. The problem with this method is **primary clustering**, not unlike the phenomenon (see Exercise) of a cluster of empty buses, arriving in succession. That is, a maximally contiguous sequence of occupied slots is called a **cluster**. A long cluster will be bad for insertions since it means we may have to traverse its length before we can insert a new key. Indeed, assuming a uniform probability of hashing to any slot, the probability of hitting a particular cluster is proportional to its length. Worse, insertion grows the length of a cluster – it grows by at least one but may grow by more when two adjacent clusters are joined. Thus, larger clusters have a higher probability of growing. Similarly, a maximal sequence of deleted and occupied slots forms a cluster for lookups.

Quadratic Probing Here, the i th probe involves the slot

$$h(k, i) = h_1(k) + a \cdot i + b \cdot i^2 \pmod{m}$$

for some integer constants $a, b \geq 0$. For instance, $a = 0, b = 1$. We avoid primary clustering but there is a possibility of missing available slots in our probe sequence unless we take special care in our design of the probe sequence.

Double Hashing Here, we use another auxiliary (ordinary) hash function $h_2(k)$.

$$h(k, i) = h_1(k) + i \cdot h_2(k) \pmod{m}.$$

To ensure that the probe sequence will visit every slot, it is sufficient to ensure that $h_2(k)$ is relatively prime to m . For example, this is true if m is prime and $h_2(k)$ is never a multiple of m . Other variants of double hashing can be imagined.

Note that both quadratic and double hashing are generalizations of linear probing.

EXERCISES

Exercise 2.1: In the separate chaining method, we have a choice about how to view the slot $T[i]$. Assume that each node in the chain has the form $(item, next)$ where $next$ is a pointer to the next node.

(i) The slot $T[i]$ can simply be the first node in the chain (and hence stores an item).

(ii) An alternative is for $T[i]$ to only store a pointer to the first node in the chain. Discuss the pros and cons of the two choices. Assume that an item requires k words of storage and a pointer requires ℓ word of storage. Your discussion may make use of the parameters k, ℓ and the load factor α . \diamond

Exercise 2.2: In quadratic hashing, we can avoid multiplications when computing successive addresses in the probe sequence. Show how to do this, *i.e.*, from $h(k, i)$, show how to derive $h(k, i + 1)$ by additions alone. \diamond

Exercise 2.3: Show that in double hashing, if $h_2(k)$ is relative prime to m , then all slots will eventually be probed. \diamond

Exercise 2.4: Buses start out at the beginning of a day by being evenly spaced out, say distance L apart. Let us assume that the bus route is a loop and the distance between bus i and bus $i + 1$ is $g_i \geq 0$ (the i th gap). So initially $g_i = L$. Each time a bus picks up passengers, it is more likely that the immediately following bus will have fewer or no passengers to pick up. The bus behind will therefore close up upon the first bus, forming a cluster. Moreover, the larger a cluster, the more likely the cluster will grow. In this way, the bus clustering phenomenon has similarities to the primary clustering phenomenon of hashing.

(i) Do a simulation or analytical study of the evolution of the gaps g_i over time, assuming that the probability of passengers joining bus i is proportional to g_i , and this contributes proportionally to the slow down of bus i (so that g_{i-1} will decrease and g_{i+1} will increase). [You need not handle the case of the g_i 's going negative.]

(ii) Let us say that two consecutive buses belong to the same cluster if their distance is $< L/2$. The size of a cluster is the distance between the leading bus and the last bus in its cluster, and the intercluster gap is defined as before. Unlike part (i), we need not worry about a bus overtaking another bus since they belong to the same cluster. So we may interpret g_i as the i th gap, but as the gap in front of the i th bus. \diamond

END EXERCISES

§3. Simplified Analysis of Hashing

Let us analyze the complexity of hashing operations. Notice that `delete` is $\Theta(1)$ in these methods and so the interest is in `lookUp` and `insert`. However, it is easy to see that an `insert` is preceded by a `lookUp`, and only if this `lookUp` is unsuccessful can we then insert the new item. The actual insertion takes $\Theta(1)$ time. Hence it suffices to analyze `lookUps`. In our analysis, the **load factor** defined as

$$\alpha := n/m$$

will be critical. Note that α will be ≤ 1 for open addressing and coalesced chaining but it is unrestricted for separate chaining.

We make several simplifying assumptions:

- **Random Key Assumption (RKA)**: it is assumed that every key in U is equally likely to be used in a lookup or an insertion. We assume that for deletion, every key in the current dictionary is equally likely to be deleted.
- **Perfect Hashing Assumption (PHA)**: This says our hash function is perfect as defined in equation (2). Combined with (RKA), it means each lookup key k is equally likely to hash to any of the m slots. Intuitively, this is the best possible behavior we can expect from our hash function and so it is important to understand what we can expect under this condition.
- **Uniform Hashing Assumption (UHA)**: this is assumption about the probe sequence (5) in open addressing. We assume that the probe sequence (5) is cyclic and generates a permutation of \mathbb{Z}_m . Moreover, a random key k in U is equally likely to generate any of the $m!$ permutations of \mathbb{Z}_m .

THEOREM 2 (RKA+PHA) *Using separate chaining for collision resolution, the average time for a `lookup` is $\mathcal{O}(1 + \alpha)$.*

Proof. In the worst case, a `lookup` of a key k needs to traverse the entire length $L(k)$ of its chain. By (RKA), the expected cost is $\mathcal{O}(1 + \bar{L})$ where \bar{L} is the average of $L(k)$ over all $k \in U$. The assumption (PHA) implies that \bar{L} is at most $n/m = \alpha$. To see this:

$$\begin{aligned}
 \bar{L} &= \frac{1}{u} \sum_{k=1}^u L(k) \\
 &= \frac{1}{u} \sum_{j=1}^m \left(\sum_{k \in U: h(k)=j} L(k) \right) \\
 &\leq \frac{1}{u} \sum_{j=1}^m \left(1 + \frac{u}{m} \right) L_j \quad (\text{by (PHA) and rewriting } L(k) \text{ as } L_{h(k)}) \\
 &= \left(\frac{1}{u} + \frac{1}{m} \right) \sum_{j=1}^m L_j \\
 &= \left(\frac{n}{u} + \frac{n}{m} \right) \\
 &< 2\alpha.
 \end{aligned}$$

Q.E.D.

In order to ensure that this average time is $\mathcal{O}(1)$, we try to keep the load factor bounded in an application.

Let us analyze the average number of probes in a `lookup` under open hashing. Recall that in this setting, when we lookup a key k , we compute a sequence of probes into $h(k, 1), h(k, 2), \dots$ until we find the key we are looking for, or we find a slot that is unoccupied. These two cases corresponds to a **successful** and an **unsuccessful lookup**, respectively. The average time for a lookup is just the number of probes made before we determine either success or otherwise. It is also easy to see that the average number of probes in an unsuccessful lookup will serve as an upper bound for the average number probes in a successful lookup.

THEOREM 3 (UHA) *Using open addressing to resolve collisions, the average number of probes for an unsuccessful lookUp is less than*

$$\frac{1}{1 - \alpha}.$$

Proof. Clearly the expected number of probes is

$$\bar{T} = 1 + \sum_{i=1}^{\infty} ip_i$$

where p_i is the probability of making exact i probes into occupied slots. (The term “1+” in this expression accounts for the final probe into an unoccupied slot, at which point the lookUp procedure terminates.) But if q_i is the probability of making at least i probes into occupied slots, then we see that

$$\bar{T} = 1 + \sum_{i=1}^{\infty} i(q_i - q_{i+1}) = 1 + \sum_{i=1}^{\infty} q_i.$$

Note that $q_1 = n/m = \alpha < 1$. The assumption (UHA) implies that $q_2 = \frac{n(n-1)}{m(m-1)} < \alpha^2$. In general,

$$q_i = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdots \frac{n-i+1}{m-i+1} < \alpha^i.$$

Hence $\bar{T} < 1 + \sum_{i=1}^{\infty} \alpha^i = 1/(1 - \alpha)$.

Q.E.D.

Note that $\bar{T} \rightarrow \infty$ as $\alpha \rightarrow 1$. In order that $\bar{T} = \mathcal{O}(1)$, we need to ensure that α is bounded away from 1, say $\alpha < 1 - \varepsilon$ for some constant $\varepsilon > 0$. For instance $\varepsilon = 1/2$ ensures $\bar{T} < 2$. Since all keys are stored in the table T , we often say that open addressing schemes uses no auxilliary storage (in contrast to separate chaining). Nevertheless, if α is bounded away from 1, some of the slots in T are really auxilliary storage.

EXERCISES

Exercise 3.1: Show that the average time to perform a successful lookup under the chaining scheme is $\Theta(1 + \alpha)$. ◇

END EXERCISES

§4. Universal Hash Sets

The above analysis depends on the random key assumption (RKA). To get around this, a fundamentally new hashing idea was proposed by Carter and Wegman [1] in 1977. Let H be a set of (hash) functions from some U to \mathbb{Z}_m . We call H a **universal hash set** if for all $x, y \in U$, $x \neq y$,

$$|\{h \in H : h(x) = h(y)\}| \leq \frac{|H|}{m}. \tag{6}$$

We intend to use H by “randomly” picking an element from H and using it as our hashing function in our usual sense. Of course, we still need to use some collision resolution method such as chaining or open addressing methods.

We will employ the useful “ δ -notation” from [1]. For $h \in [U \rightarrow \mathbb{Z}_m]$ and $x, y \in U$, define

$$\delta_h(x, y) := \begin{cases} 1 & \text{if } x \neq y, h(x) = h(y) \\ 0 & \text{else.} \end{cases}$$

Thus $\delta_h(x, y)$ indicates the presence of a conflict. We can replace any of h, x, y in this notation by sets: if $H \subseteq [U \rightarrow \mathbb{Z}_m]$ and $X, Y \subseteq U$ then

$$\delta_H(X, Y) = \sum_{h \in H} \sum_{x \in X} \sum_{y \in Y} \delta_h(x, y).$$

Variations such as $\delta_H(x, Y)$ or $\delta_h(X, Y)$ have the obvious meaning. So H is universal means $\delta_H(x, y) \leq |H|/m$ for all $x, y \in U$.

Motivation. In the following we will let \mathbf{h} denote a random function in H . This means that for all $h \in H$, $\Pr\{\mathbf{h} = h\} = 1/|H|$. Let us first see why universality is a natural definition. It is easy to see that

$$\Pr\{\mathbf{h}(x) = \mathbf{h}(y)\} = \frac{|\{h \in H : h(x) = h(y)\}|}{|H|}.$$

This makes no assumptions about H . But if $x \neq y$ then H is universal if and only if the last expression is $\leq 1/m$. This shows:

LEMMA 4 *H being universal is equivalent to*

$$\Pr\{\mathbf{h}(x) = \mathbf{h}(y)\} \leq \frac{1}{m} \tag{7}$$

whenever $x \neq y$.

Below we will show that this definition is essentially optimal. Let us now contrast universality to our assumptions in the simplified analysis of hashing (§3). The random key assumption (RKA) says that we are interested in analyzing \mathbf{k} , a random key in U , *i.e.*, $\Pr\{\mathbf{k} = k\} = 1/u$ for any $k \in U$. Combined with the perfect hashing assumption (PHA),

$$\Pr\{h(\mathbf{k}) = i\} = 1/m \tag{8}$$

for any $i = 0, \dots, m-1$. So we have replaced the randomness assumption about keys in equation (8) by a randomness about hashing functions in equation (7). The latter assumption is better because in hashing applications, the algorithm designer is supposed to choose the hash function, and preferably, imposes no condition on the set of keys to be inserted or searched. This is what universal hashing achieves.

The following theorem shows that universal hash sets gives us the “expected” behavior:

THEOREM 5 *Let $H \subseteq [U \rightarrow \mathbb{Z}_m]$ be a universal hash set and \mathbf{h} be a random function in H . For any subset $K \subseteq U$ of n keys, and for any $x \in K$, the expected number of collisions of \mathbf{h} involving x is $< n/m = \alpha$.*

Proof. Recall $\delta_{\mathbf{h}}(x, y)$ is the 0/1 function that is 1 iff $\mathbf{h}(x) = \mathbf{h}(y)$. Since \mathbf{h} is a random function, $\delta_{\mathbf{h}}(x, y)$ is a random variable. We have $\mathbb{E}[\delta_{\mathbf{h}}(x, y)] = \Pr\{\delta_{\mathbf{h}}(x, y) = 1\} \leq 1/m$. The expected number of collisions involving $x \in K$ is given by

$$\begin{aligned} \mathbb{E}[\delta_{\mathbf{h}}(x, K)] &= \mathbb{E}\left[\sum_{y \in K, y \neq x} \delta_{\mathbf{h}}(x, y)\right] \\ &= \sum_{y \in K, y \neq x} \mathbb{E}[\delta_{\mathbf{h}}(x, y)] \\ &= \frac{n-1}{m} < \alpha. \end{aligned}$$

Q.E.D.

Generalization of Universality. If $h : U \rightarrow V$ and $x_1, \dots, x_t \in U$ then we write

$$h(x_1, \dots, x_t) = (y_1, \dots, y_t)$$

to mean $h(x_i) = y_i$ for all $i = 1, \dots, t$. We say the set $H \subseteq [U \rightarrow V]$ is **t -universal** ($t \in \mathbb{N}$) if for all $\{x_1, \dots, x_t\} \in \binom{U}{t}$, and all $y_1, \dots, y_t \in V$,

$$|\{h \in H : h(x_1, \dots, x_t) = (y_1, \dots, y_t)\}| \leq \frac{|H|}{m^t}.$$

For instance, H is 2-universal means for all $x, x' \in U$ where $x \neq x'$, and all $y, y' \in V$,

$$|\{h \in H : h(x, x') = (y, y')\}| \leq \frac{|H|}{m^2}.$$

Note that we allow $y = y'$ in this definition. Alternatively, if \mathbf{h} is a random function in H , then

$$\Pr\{\mathbf{h}(x, x') = (y, y')\} \leq \frac{1}{m^2}.$$

THEOREM 6 *If $H \subseteq [U \rightarrow \mathbb{Z}_m]$ is 2-universal, then it is universal.*

Proof. Let $x \neq y \in U$ and \mathbf{h} be a random function of H .

$$\begin{aligned} \Pr\{\mathbf{h}(x) = \mathbf{h}(y)\} &= \sum_{i=0}^{m-1} \Pr\{\mathbf{h}(x) = \mathbf{h}(y) = i\} \\ &\leq \sum_{i=0}^{m-1} \frac{1}{m^2}, \quad (\text{by 2-universality}) \\ &= 1/m. \end{aligned}$$

By lemma 4, this is equivalent to the universality of H .

Q.E.D.

The converse is not true: consider the set

$$S_U \subseteq [U \rightarrow U]$$

of permutations of U . Thus $|S_U| = u!$ and for all $x \neq x'$,

$$|\{h \in S_U : h(x) = h(x')\}| = 0.$$

Thus S_U is universal. But for all $y, y' \in U$,

$$|\{h \in S_U : h(x, x') = (y, y')\}| = \begin{cases} 0 & \text{if } y \neq y', \\ (u-2)! & \text{else.} \end{cases}$$

So S_U is not 2-universal, since $(u-2)! > |S_U|/u^2$. But S_U is rather close to being 2-universal, and it might be advantageous to modify the definition of t -universality so that S_U is considered 2-universal (Exercise).

On the Definition of Universality. Carter and Wegman show that their definition of universal hash sets is essentially the best possible.

LEMMA 7 For all H , there exists $x, y \in U$ such that

$$\delta_H(x, y) > |H| \left(\frac{1}{m} - \frac{1}{u} \right).$$

Proof. First, fix $f \in H$ and let $U = \uplus_{i=0}^{m-1} U_i$ where $U_i = f^{-1}(i)$ ($i \in \mathbb{Z}_m$). Let $u_i = |U_i|$. Then

$$\delta_f(U_i, U_j) = \begin{cases} u_i(u_i - 1) & \text{if } i = j \\ 0 & \text{else.} \end{cases}$$

Hence

$$\delta_f(U, U) = \sum_i \sum_j \delta_f(U_i, U_j) = \sum_i \delta_f(U_i, U_i) = \sum_{i=0}^{m-1} u_i(u_i - 1).$$

It is easily seen that the expression $E(u_0, \dots, u_{m-1}) = \sum_{i=0}^{m-1} u_i(u_i - 1)$ is minimized when $u_i = u/m$ for all i (Exercise). Hence

$$\delta_f(U, U) \geq \sum_{i=0}^{m-1} \frac{u}{m} \left(\frac{u}{m} - 1 \right) = u^2 \left(\frac{1}{m} - \frac{1}{u} \right).$$

Hence

$$\delta_H(U, U) \geq |H| u^2 \left(\frac{1}{m} - \frac{1}{u} \right). \quad (9)$$

But

$$\delta_H(U, U) = \sum_{x \in U} \sum_{y \in U} \delta_H(x, y). \quad (10)$$

There are u^2 choices of x, y in (10). From (9), it follows that at least one of these choices will satisfy the lemma. **Q.E.D.**

This shows that, in general, the right hand side of (7) cannot be replaced by $\frac{1}{m} - \varepsilon$, for any constant $\varepsilon > 0$. On the other hand, it might be advantageous to replace (7) by $\frac{1}{m} + \varepsilon$ ($\varepsilon = \Theta(1/m^2)$, see Exercise).

EXERCISES

Exercise 4.1: Student Quick claims out the universal hash set approach still does not overcome the problem of bad behavior for specialized sets $K \in U$. That is, for any $h \in H$, we can still find a K that causes h to behave badly. Do you agree? \diamond

Exercise 4.2: Quick Search Company has implemented a dictionary data structure using universal hashing. You are a hacker who wants to make the boss of Quick Search Company look bad, by making its dictionary operations slow. You can read all files (data, source code, etc) of the company, but you may not modify any file directly. However, you are a legitimate user who is allow enter new items into the dictionary. The dictionary is designed for 10,000 records (and will not accept more). It is currently half full. Discuss how you can accomplish your evil goals. Also what can the Quick Search Company do to avoid such kind of attacks? \diamond

Exercise 4.3: In the practical usage of a universal hash set H , suppose that after the choice of an $h_1 \in H$, the system administrator may find that the current set K of keys is causing suboptimal performance. The idea is that he should now discard h_1 and pick randomly another $h_2 \in H$ and re-insert all the keys in K . Give some guidelines about how to do this. E.g., how and when do you decide that K is causing suboptimal performance? \diamond

Exercise 4.4: Suppose we modify the definition of “ t -universality” of H to mean that for all $\{x_1, \dots, x_t\} \in \binom{U}{t}$, and all $y_1, \dots, y_t \in V$,

$$|\{h \in H : h(x_1, \dots, x_t) = (y_1, \dots, y_t)\}| \leq \frac{|H|}{m(m-1) \cdots (m-t+1)}.$$

- (a) What are the advantages of this definition?
 (b) Suppose we also modify the definition of universality of H to mean

$$|\{h \in H : h(x) = (y)\}| \leq \frac{|H|}{m-1}.$$

Show that 2-universality (in this modified sense) implies modified universality. Are there any some disadvantage in this definition? \diamond

END EXERCISES

§5. Construction of Universal Hash Sets

It is actually trivial to show the existence of universal hash sets: we can just choose H to be the set $[U \rightarrow \mathbb{Z}_m]$. This H is universal (Exercise). It is unfortunately not very useful: to use H , we intend to pick a random function h from H and use it as our hashing function. First of all, to represent an arbitrary element of $[U \rightarrow \mathbb{Z}_m]$ would require $\lg |H| = u \lg m$ bits. Since $u = |U|$ is huge by assumption (H1), this is infeasible. It would also defeat an original motivation to use hashing in order to avoid $\Omega(u)$ space. Second, to use $h \in H$ as a hash function, each h must be easy to compute by assumption (H2). But not all functions in $[U \rightarrow \mathbb{Z}_m]$ have this property. Let us summarize our requirements on H :

- $|H|$ be moderate in size (typically $u^{\mathcal{O}(1)}$).
- There is a simple method to name each member of H , and to randomly pick members of H .
- Each $h \in H$ must be easy to compute.

The latter two properties are usually coupled together as follows: the set $H = \{h_i : i \in I\}$ is indexed by a finite set I , and there is a fixed universal program $M_{(\cdot)}$ such that, given an index $i \in I$, $M_{(i)}$ computes h_i . In that case, $\log |I|$ can essentially be regarded as the **program size** of H .

We now construct some universal hash sets that satisfy these requirements.

A Class of Universal Hash Sets. Fix a finite field F with q elements. Typically, $F = \mathbb{Z}_q$ where q is prime. We are interested in hash functions in $[U \rightarrow F]$ where

$$U = F^r$$

for any fixed $r \geq 1$. If $k = \langle k_1, \dots, k_r \rangle \in U$ and $a = \langle a_0, a_1, \dots, a_r \rangle \in F^{r+1}$, we define the hash function

$$\begin{aligned} h_a &: U \rightarrow F \\ h_a(x) &= a_0 + \sum_{i=1}^r a_i x_i \end{aligned}$$

where $x = \langle x_1, \dots, x_r \rangle \in U$. Set

$$H_q^r := \{h_a : a \in F^{r+1}\} \quad (11)$$

so that $|H| = q^{r+1}$.

THEOREM 8 *The set H_q^r is 2-universal. More precisely, if \mathbf{h} is a random function in H_q^r then*

$$\Pr\{\mathbf{h}(x) = i, \mathbf{h}(y) = j\} = \frac{1}{q^2}$$

for all $x, y \in K$, $x \neq y$, and $i, j \in F$.

Proof. First write x and y as $x = \langle x_1, \dots, x_r \rangle$ and $y = \langle y_1, \dots, y_r \rangle$. Since $x \neq y$, we may, without loss of generality, assume $x_1 \neq y_1$. CLAIM: for any choice of a_2, \dots, a_r and $0 \leq i, j < m$, there exists unique a_0, a_1 such that if $a = \langle a_0, a_1, \dots, a_r \rangle$ then

$$h_a(x) = i, \quad h_a(y) = j. \quad (12)$$

To see this, note that (12) can be rewritten as

$$\begin{bmatrix} x_1 & 1 \\ y_1 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} i - \sum_{\ell=2}^r a_\ell x_\ell \\ j - \sum_{\ell=2}^r a_\ell y_\ell \end{bmatrix}.$$

The right-hand side is a constant since we have fixed i, j and a_2, \dots, a_r . The matrix M on the right-hand side is non-singular because $x_1 \neq y_1$. Hence we may multiply both sides by M^{-1} , giving a unique solution for a_0, a_1 . This proves our CLAIM. There are q^{r-1} choices for a_2, \dots, a_r . It follows that there are exactly q^{r-1} functions in H such that (12) is true. Therefore,

$$\Pr\{\mathbf{h}(x) = i, \mathbf{h}(y) = j\} = \frac{q^{r-1}}{|H|} = \frac{1}{q^2}.$$

Q.E.D.

Thus H_q^r in (11) is universal.

Example: Consider a typical application. Again, let $U = [0..9]^9$ be the set of social security numbers. We wish to construct a dictionary (=database) in which $n = 50,000$ (e.g., n is an upper bound for the number enrolled students at Universal University). Our problem is to choose an m such that $\alpha = n/m$ is some small constant, say

$$1 < \alpha < 10. \quad (13)$$

The motivation for $\alpha < 10$ is to bound the expected size of a chain which, according to theorem 5, is bounded by α . The motivation of $\alpha > 1$ is to limit the pre-allocated amount of storage (which is the table $T[0..m-1]$) to less than n . Note that U and n are given *a priori*.

Solution: We reduce this problem to the construction of a universal hash set of the form (11). Let us assume q is a prime. First of all, note that q should be somewhere between 5,000 and 50,000. We also need to choose r so that each $k \in U$ is viewed as an r -tuple $\langle k_1, \dots, k_r \rangle$. For this purpose, we divide the 9 digits in k into $r = 3$ blocks of 4, 4, 1 digits (respectively). E.g., $k = 123456789$ is viewed as the triple $\langle 1234, 5678, 9 \rangle$. Let q be the smallest prime larger than 10^4 , i.e., $q = 10007$. Hence $\alpha = 50000/10007 \approx 5$. Note that even though k_3 in any key $\langle k_1, k_2, k_3 \rangle \in U$ is never more than 9, it did not affect our application of theorem 5: the result does not depend on the choice of K ! This method can be generalized (Exercise)

Weighted Universal Hash Sets. Consider the following situation. Let U, V, W be three finite sets. Suppose

$$H \subseteq [U \rightarrow V]$$

is a universal hash set, and

$$g : V \rightarrow W$$

of a perfect hash function. This means

$$|\{x \in F : g(x) = i\}| \leq \lceil q/m \rceil.$$

For instance, g may be the modulo m function, $g(x) = x \bmod m$. Let

$$H_g := \{g \circ h : h \in H\}$$

where $(g \circ h)(x) = g(h(x))$ denotes function composition. Under what condition is H_g universal?

Before proceeding, we need a clarification: notice that it may happen that $h \neq h'$ but $g \circ h = g \circ h'$. This means $|H_g| < |H|$ when this happens. In the following, we shall assume

$$|H_g| = |H|.$$

One way to allow this to hold without restriction is to interpret H_g as a multiset. Formally, a **multiset** is a pair (S, μ) where $\mu : S \rightarrow \mathbb{N}$ assigns a **multiplicity** $\mu(x)$ to each $x \in S$. We usually simply refer to S as the “multiset” with μ implicit. We shall generalize this further and allow $\mu(x)$ to be any non-negative real number. In this case, we call S a **weighted set**. For any set $X \subseteq S$, write $\mu(X)$ for $\sum_{x \in X} \mu(x)$. It is obvious that our concept of universality extends naturally to weighted set of functions: a weighted set $H \subseteq [U \rightarrow V]$ is **universal** if for all $x, y \in U$, $x \neq y$,

$$\mu(\{h \in H : h(x) = h(y)\}) \leq \frac{\mu(H)}{m}.$$

We use a weighted universal set H by picking a “random” function \mathbf{h} in H : this means for any $h \in H$, $\Pr\{\mathbf{h} = h\} = \mu(h)/\mu(H)$.

Another Construction Scheme. Rather than proving the most abstract result possible, we begin with a concrete example. Suppose $U = V$ is a finite field F where $|F| = q$, and $W = \mathbb{Z}_m$ for $m > 1$. For any $a, b \in F$, define the hash function

$$h_{a,b}(x) = ax + b. \quad (14)$$

Let $H = \{h_{a,b} : a, b \in F, a \neq 0\}$. Now consider the multiset

$$H_g = \{g_{a,b} : a, b \in F, a \neq 0\} \quad (15)$$

where $g_{a,b} = g \circ h_{a,b}$. We do not consider the case $a = 0$ in this definition since $h_{0,b}$ is a constant function. Thus

$$|H| = |H_g| = (q-1)q$$

(as weighted sets). Indeed, notice that $g_{a,b} = g_{c,d}$ iff $a \equiv c \pmod{m}$ and $b \equiv d \pmod{m}$. Consider the simultaneous equation in the unknowns $a, b \in F$:

$$ax + b = i, \quad ay + b = j \quad (16)$$

for $x, y \in F$ and $i, j \in F$. This can be written

$$\begin{bmatrix} x & 1 \\ y & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} i \\ j \end{bmatrix}. \quad (17)$$

This has a non-trivial solution iff $x \neq y$. The solution (a, b) is unique. However, if $i = j$, this solution has $a = 0$.

LEMMA 9 *The multiset H_g is universal.*

Proof. Fix $x, y \in F$, $x \neq y$. Our lemma is to show

$$\delta_{H_g}(x, y) \leq \frac{|H_g|}{m} = \frac{(q-1)q}{m}.$$

Call $g^{-1}(i)$ the i th bin and let $b_i = |g^{-1}(i)|$ be its size. Now $b_i \leq \lceil q/m \rceil$ so that

$$b_i - 1 \leq \frac{q+m-1}{m} - 1 = \frac{q-1}{m}.$$

For each $h \in H_g$, we charge h to the i th bin if $h(x) = h(y) = i$. According to above remarks on the simultaneous equation (16), the number of charges to the i th bin is exactly $b_i(b_i - 1)$. Since the bins are disjoint, the total number of charges is $\sum_{i=0}^{m-1} b_i(b_i - 1)$. It is easy to see that $\delta_h(x, y) = 1$ iff h charges some bin. Hence

$$\begin{aligned} \delta_{H_g}(x, y) &= \sum_{i=0}^{m-1} b_i(b_i - 1) \\ &\leq \frac{q-1}{m} \sum_{i=0}^{m-1} b_i \\ &= \frac{(q-1)q}{m}. \end{aligned}$$

Q.E.D.

We generalize this result as follows:

THEOREM 10 *Let $H \subseteq [U \rightarrow V]$ be universal, and $g : V \rightarrow W$ be a perfect hash function. Define the multiset*

$$H_g := \{g \circ h : h \in H\}.$$

Let $|H| = h, |U| = u, |V| = v, |W| = w$. Then H_g is universal under either one of the following conditions:

(i) H is 2-universal and v divides w .

(ii) $v > w$ and $h \geq \frac{v^2(v-1)}{v-w}$. (For instance, if $v > w$ and $h \geq v^3$.)

Proof. (i) We have

$$\begin{aligned} |\{h \in H : g(h(x)) = g(h(y))\}| &= \sum_{i \in W} |\{h \in H : g(h(x)) = g(h(y)) = i\}| \\ &= \sum_{i \in W} \sum_{x', y' \in g^{-1}(i)} |\{h \in H : h(x, y) = (x', y')\}| \\ &\leq \sum_{i \in W} \sum_{x', y' \in g^{-1}(i)} \frac{|H|}{v^2} \\ &\leq \frac{|H|}{v^2} \sum_{i \in W} b_i^2 \\ &\leq \frac{|H|}{v^2} \sum_{i \in W} \left(\frac{v}{w}\right)^2 \quad (\text{since } \lceil v/w \rceil = v/w) \\ &= \frac{|H|}{w}. \end{aligned}$$

(ii) We have

$$\begin{aligned}
 |\{h \in H : g(h(x)) = g(h(y))\}| &= |\{h \in H : h(x) = h(y)\}| + \sum_{i \in W} |\{h \in H : h(x) \neq h(y), g(h(x)) = g(h(y)) = i\}| \\
 &\leq \frac{h}{v} + \sum_{i \in W} b_i(b_i - 1), \quad \text{where } b_i := |g^{-1}(i)| \leq \lceil v/w \rceil \\
 &\leq \frac{h}{v} + \left(\lceil \frac{v}{w} \rceil - 1\right) \sum_{i \in W} b_i \\
 &\leq \frac{h}{v} + \left(\frac{v-1}{w}\right)v
 \end{aligned}$$

Hence, universality of H follows if $w < v$ and

$$\begin{aligned}
 \frac{h}{w} &\geq \frac{h}{v} + \left(\frac{v-1}{w}\right)v, \\
 h\left(\frac{1}{w} - \frac{1}{v}\right) &\geq \frac{v(v-1)}{w}, \\
 h &\geq \frac{v(v-1)}{w} \bigg/ \left(\frac{1}{w} - \frac{1}{v}\right) \\
 &= \frac{v^2(v-1)}{v-w}.
 \end{aligned}$$

Q.E.D.

EXERCISES

Exercise 5.1: (a) Is the set $H_0 = [U \rightarrow \mathbb{Z}_m]$ universal? 2-universal? Useful as a universal hash set?

(b) Is the set $H_U \subseteq [U \rightarrow U]$ of permutations on U universal? 2-universal? Useful as a universal hash set? \diamond

Exercise 5.2: Consider the universal hash set H_g above. Suppose $|F| = q$ and $m_1 = (q \bmod m)$. Give an exact expression for the cardinality of $\delta_H(x, y)$ for $x, y \in F$ in terms of m, q, m_1 . HINT: let $r = \lfloor q/m \rfloor$. Then there are m_1 bins of g of size $r + 1$, and $m - m_1$ bins of size r . Determine the contribution of each bin to $\delta_H(x, y)$. \diamond

Exercise 5.3: (Carter-Wegman) Suppose we modify the multiset H_g by omitting those functions in $h_{a,b} \in H_g$ where $b \neq 0$. Let \widehat{H}_g be this new class. In other words, \widehat{H}_g has all functions of the form $h_a(x) = g(ax)$. Show that $\delta_{\widehat{H}_g}(x, y) \leq 2|\widehat{H}_g|/m$. That is, the class is “universal within a constant factor of 2”. \diamond

Exercise 5.4: Suppose we define \widehat{H}_q^r similarly to H_q^r , except that we fix $a_0 = 0$. Hence $|\widehat{H}_q^r| = q^r$.

(a) Show that theorem 8 fails for \widehat{H}_q^r .

(b) Show that \widehat{H}_q^r is still universal. \diamond

Exercise 5.5: Consider the example above in which we choose to interpret a social security number as a triple $\langle k_1, k_2, k_3 \rangle$ where the 9 digits are distributed among k_1, k_2, k_3 in the proportions 4 : 4 : 1. Can I choose the proportion 3 : 3 : 3? What are the new freedoms I get with this choice? HINT: what other m 's are now available to me? How close can α get to 10? \diamond

Exercise 5.6: Generalize the above methods for construct t -universal hash sets for any $t \in \mathbb{N}$. ◇

Exercise 5.7: Let $U = [1..t]^s$ for integers $t, s \geq 2$ and let n be given. What is a good way to construct a universal hash set H of functions from U to \mathbb{Z}_m , where m is chosen to satisfy $0.5 < \alpha = n/m < t$. NOTE: t is typically small, e.g., $t = 10, 26, 128, 256$. You may use the fact (Bertrand's postulate) that for any $n \geq 1$, there is a prime number p satisfying $n < p \leq 2n$. ◇

END EXERCISES

§6. Optimal Static Hashing

Recall (§1) that a static dictionary is one that supports lookups, but no insertion or deletions. The question arises: for any set $K \subseteq U$, can we find hashing scheme that has worst-case $\mathcal{O}(1)$ access time and $\mathcal{O}(|K|)$ space? An elegant affirmative answer is provided by Fredman, Komlós and Szemerédi [5].

For brevity, we call this the “optimal hashing problem”, since the space $\mathcal{O}(|K|)$ is optimal and the worst-case $\mathcal{O}(1)$ time is also optimal. The consideration of worst-case time stands in contrast to the usual average time bounds in hashing analysis. Also, the combination of small space with $\mathcal{O}(1)$ worst case time is necessary since we can otherwise obtain $\mathcal{O}(1)$ worst case time trivially, by using space $\mathcal{O}(|U|)$ and hashing each k into its own slot.

The following basic setup will be used in our analysis: assume $U = \mathbb{Z}_p$ for some prime p , and let $K \subseteq U$, $|K| = n$ be given. We want to define a hash function $h : U \rightarrow \mathbb{Z}_m$ with certain properties that are favorable to K .

Our hash functions is a special case of (15): for any $k \in \mathbb{Z}_p$ and $x \in U$,

$$h_{k,m}(x) = ((kx \bmod p) \bmod m).$$

We write $h_k(x)$ instead of $h_{k,m}(x)$ when m is understood. We avoid $k = 0$ in the following, since $h_0(x) = 0$ for all x . For any $k \in \mathbb{Z}_p$ and $i \in \mathbb{Z}_m$, define the i th bin to be $\{x \in K : h_k(x) = i\}$, and let its size be

$$b_k(i) := |\{x \in K : h_k(x) = i\}|.$$

Note that the number of pairs $\{x, y\}$ that collide in the i th bin is $\binom{b_k(i)}{2}$. We have the following bound:

LEMMA 11

$$\sum_{k=1}^{p-1} \sum_{i=0}^{m-1} \binom{b_k(i)}{2} < \frac{pn^2}{2m}.$$

Proof. The left-hand side counts the number of pairs

$$(k, \{x, y\}) \in \mathbb{Z}_p^+ \times \binom{K}{2}$$

such that $h_k(x) = h_k(y)$. Let us count this in another way: we say that $k \in \mathbb{Z}_p$ “charges” the pair $\{x, y\} \in \binom{K}{2}$ if $h_k(x) = h_k(y)$. The k 's that charge $\{x, y\}$ satisfies

$$(xk \bmod p) - (yk \bmod p) \equiv 0 \pmod{m},$$

$$(x - y)k \bmod p \equiv 0 \pmod{m},$$

$$(x - y)k \bmod p \in S := \{m, 2m, \dots, \lfloor \frac{p-1}{m} \rfloor m\}.$$

But for each element jm in the set S above, there is a unique k such that $(x - y)k \bmod p = jm$. Hence the number of k 's that charge $\{x, y\}$ is

$$|S| = \lfloor \frac{p-1}{m} \rfloor.$$

Thus the total number of charges, summed over all $\{x, y\} \in \binom{K}{2}$ is

$$\binom{n}{2} \lfloor \frac{p-1}{m} \rfloor < \frac{n(n-1)(p-1)}{2m}$$

and the lemma follows. **Q.E.D.**

COROLLARY 12 (i) *There exists a $k \in \mathbb{Z}_p^+$ such that*

$$\sum_{i=0}^{m-1} \binom{b_k(i)}{2} < \frac{n^2}{2m}.$$

(ii) *There are at least $p/2$ choices of $k \in \mathbb{Z}_p^+$ such that*

$$\sum_{i=0}^{m-1} \binom{b_k(i)}{2} < \frac{n^2}{m}.$$

We have an immediate application. Choosing $m = n^2$, corollary 12(i) says that there is a k such that

$$\sum_{i=0}^{m-1} \binom{b_k(i)}{2} < 1. \tag{18}$$

This means for each $i \in \mathbb{Z}_m$, $\binom{b_k(i)}{2} = 0$ and hence $b_k(i) = 0$ or 1. This means h_k is a perfect hash function for K .

The FKS Scheme. We now describe the FKS scheme [5] to solve the optimal hashing problem. This scheme is illustrated in figure 1.

There are two global variables k, n and these are used to define the primary hash function,

$$\tilde{h}(x) = ((xk \bmod p) \bmod n). \tag{19}$$

There is a main hash table $T[0..n-1]$. The i th entry $T[i]$ points to a secondary hash table that has two parameters k_i, b_i and these define the secondary hash functions

$$h_{(i)}(x) = ((xk_i \bmod p) \bmod b_i^2). \tag{20}$$

We shall choose b_i to be the size of the i th bin,

$$b_i = |\{x \in K : \tilde{h}(x) = i\}|.$$

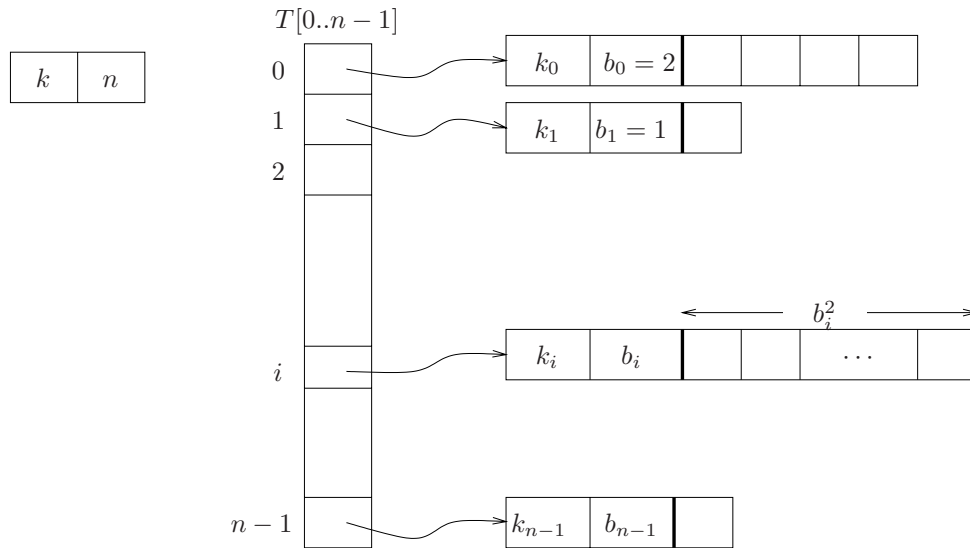


Figure 1: FKS Scheme

Hence, according to the remark above, we could choose k_i in (20) so that (18) holds, and so $h_{(i)}$ is a perfect hash function.

How much space does the FKS scheme take? The primary table takes $n + 2$ cells (the “+2” is for storing the values n and k). The secondary tables use space

$$\sum_{i=0}^{n-1} (2 + b_i^2) = 2n + \sum_{i=0}^{n-1} b_i^2. \tag{21}$$

According to corollary 12(i), we can choose the key k in the primary hash function (19) such that

$$\sum_{i=0}^{n-1} \binom{b_i}{2} < \frac{n^2}{m} = n \tag{22}$$

($m = n$). Thus (22) implies $\sum_{i=0}^{n-1} b_i(b_i - 1) < 2n$ and hence

$$\sum_{i=0}^{n-1} b_i^2 < 2n + \sum_{i=0}^{n-1} b_i = 3n.$$

This, combined with (21), implies the secondary tables use space $5n$. The overall space usage is therefore less than

$$n + 2 + 5n = 6n + 2.$$

Constructing a FKS Solution. Given p and K , how do we find the keys k and k_0, \dots, k_{n-1} specified by the FKS scheme? For simplicity, let us first assume that all arithmetic operations (including taking modulus) is constant time.

A straightforward way is to search through \mathbb{Z}_p to find a primary hash key k . Checking each k to see if corollary 12(i) is fulfilled takes $\mathcal{O}(n)$ time. Since there are p keys, this is $\mathcal{O}(pn)$ time. To find a suitable secondary k_i for each i takes another $\mathcal{O}(pb_i)$ time; summing over all i , this is $\mathcal{O}(pn)$ time. So the overall time is $\mathcal{O}(pn)$.

Since p can be very large relative to n , this solution is sometimes infeasible. If we use a bit more space (but still linear), we can use corollary 12(ii) to give a randomized method of construction (Exercise). We next present a deterministic time solution.

The solution uses a simple trick to reduce the size of the universe. For this, we use a useful fact from number theory. If $\pi(m)$ is the number of primes less than or equal to m , then

$$\pi(m) = C_m \frac{m}{\ln m}, \quad \left(\frac{1}{8} < C_m < 12\right) \tag{23}$$

(see [6, p.79]).

LEMMA 13 *Let $|K| \geq 16$. There exists a prime $q \leq n^2 \lg p \lg \lg p$ that for all $x, y \in K$,*

$$x \neq y \Rightarrow (x \bmod q) \neq (y \bmod q). \tag{24}$$

Proof. Let

$$N = \prod_{x,y} |x - y|$$

where $\{x, y\}$ range over $\binom{K}{2}$. There are at most $\lg N < \binom{n}{2} \lg p$ primes that divide N . The result follows if

$$\pi(n^2 \lg p \lg \lg p) > \binom{n}{2} \lg p.$$

From (23),

$$\pi(n^2 \lg p \lg \lg p) > \frac{n^2 \lg p \lg \lg p}{8(2 \ln n + \ln \lg p + \ln \lg \lg p)}.$$

Hence it is sufficient to show

$$\begin{aligned} n^2 \lg p \lg \lg p &> 8(2 \ln n + \ln \lg p + \ln \lg \lg p) \binom{n}{2} \lg p \\ &= 8n(n-1) \ln n \lg p + 4 \ln \lg p + \ln \lg \lg p \binom{n}{2} \lg p + 32 \lg n \lg \lg p + 16 \lg p \lg \lg p. \end{aligned}$$

But it is easily checked that ... INCOMPLETE

Q.E.D.

An asymptotically stronger result than the preceding lemma can be obtained, albeit with somewhat less accessible constant: let $\theta(x) = \prod_{q \leq x} q$, where the q 's range over primes not exceeding x . Then

$$Ax \leq \ln \theta(x) \leq Bx$$

for some $B > A > 0$. Thus for some $C > 0$, $\ln \theta(Cn^2 \lg p) > \lg N$. So there is a prime $q \leq Cn^2 \lg p$ such that q does not divide N .

THEOREM 14 *For any subset $K \subseteq \mathbb{Z}_p$, $n = |K|$, there is a hashing scheme to store K in $\mathcal{O}(n)$ space and with $\mathcal{O}(1)$ worst case lookup time. This scheme can be constructed deterministically in time*

$$\mathcal{O}(n^3 \lg p \lg \lg p).$$

Proof. If $p < n^2 \lg p \lg \lg p$, then we can use the FKS scheme for this problem. The straightforward method to construct the FKS scheme takes $\mathcal{O}(pn)$ time, which achieves our stated bound.

So assume $p \geq n^2 \lg p \lg \lg p$. We can find a prime q that satisfies the preceding lemma in time $\mathcal{O}(n^3 \lg p \lg \lg p)$. We now construct a FKS scheme for the set of keys

$$K' = \{k \bmod q : k \in K\}$$

viewed as a subset of the universe \mathbb{Z}_q . The only difference is that, in the secondary tables, in the slot for key $k' \in K'$, we store the original value $k \in K$ corresponding to k' .

The straightforward method of constructing this scheme is $\mathcal{O}(qn)$ which is within our stated bound. To lookup a key k^* , we first compute $k' = k^* \bmod q$, and then use the FKS scheme to lookup the key k' . Searching for k' will return the key $k \in K$ such that $k \bmod q = k'$. Then k^* is in K iff if $k^* = k$. **Q.E.D.**

Bit Complexity Model. We can convert the above results into the bit complexity model. First, we have assumed $\mathcal{O}(1)$ space for storing each number in $U = \mathbb{Z}_p$. In the big complexity model, we just need to multiply each space bound by $\lg p$. As for time, each arithmetic operation that we have assumed is constant time really involves $\lg p$ bit numbers, and each uses

$$\mathcal{O}(\lg p \lg \lg p \lg \lg \lg p)$$

bit operations. Again, multiplying all our time bounds by this quantity will do the trick.

EXERCISES

Exercise 6.1: Construct a FKS scheme for the following input: $p = 31$, $K = \{2, 4, 5, 15, 18, 30\}$. ◇

Exercise 6.2: Construct a FKS scheme for the 40 common English words in §1 (Exercise 1.2). ◇

Exercise 6.3: In many applications, the key space U comes with some specific structure. Suppose $U = \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times \cdots \times \mathbb{Z}_{n_r}$ where n_1, \dots, n_r are pre-specified. In a certain transaction processing application, we have $(n_1, \dots, n_r) = (2, 9, 4, 9, 5)$. Construct a FKS scheme for this application. ◇

Exercise 6.4: Show that the expected time to construct the above hashing scheme for any given K is $\mathcal{O}(n^2)$. That is, find the values $k, k_0, \dots, k_{n-1}, b_0, \dots, b_{n-1}$ in expected $\mathcal{O}(n)$ time. ◇

Exercise 6.5: The above $\mathcal{O}(pn)$ deterministic time algorithm for constructing the FKS scheme was only sketched. Please fill this in the details. Program this in a programming language of your choice. ◇

END EXERCISES

§7. Perfect Hashing

Let $h : U \rightarrow V$ and $K \subseteq U$. We said (§1) h is perfect for K if for all $i, j \in V$, we have $|b_i - b_j| \leq 1$ where $b_i = |h^{-1}(i) \cap K|$. In the literature, this definition is further restricted to the case $|K| \leq |V|$. In this case,

we have $b_i = 0$ or $b_i = 1$. In this section, we assume this restriction. If h is perfect for K and $|K| = |V|$, then we say h is **minimal perfect**. A comprehensive survey of perfect hashing may be found in [2].

Following Mehlhorn, we say a set $H \subseteq [U \rightarrow V]$ is (u, v, n) -**perfect** if $|U| = u, |V| = v$ and for all $K \in \binom{U}{n}$, there is a $h \in H$ that is perfect for K . Extending this notation slightly, we say H is $(u, v, n; k)$ -**perfect** if, in addition, $|H| = k$. Such a set H can be represented as $k \times u$ matrix M whose entries are elements of V . Each row of M represents a function in H . Moreover, if M' is the restriction of M to any n columns, there is a row of M' whose entries are all distinct.

Let us give a construction for such a matrix based on the theory of finite combinatorial planes. Let F_q be any finite field on q elements. Let M be a $(q+1) \times q^2$ matrix with entries in F_q . The rows of M are indexed by elements of $F \cup \{\infty\}$ and the columns of M are indexed by elements of F^2 . Let $r \in F \cup \{\infty\}$ and $(x, y) \in F^2$. The $(r, (x, y))$ -th entry is given by

$$M(r, (x, y)) = \begin{cases} xr + y & \text{if } r \neq \infty \\ x & \text{else.} \end{cases}$$

It is easy to see that for any two columns of M , there is exactly one row at which these two columns have identical entries. It easily follows:

THEOREM 15 *If $q+1 > \binom{n}{2}$ then M represents a $(q^2, q, n; q+1)$ -perfect set of hash function.*

We consider lower bounds on $|H|$ for perfect families.

THEOREM 16 (MEHLHORN) *If H is (u, v, n) -perfect then*

$$(a) |H| \geq \binom{u}{n} \left(\frac{u}{v}\right)^2 \binom{v}{n}.$$

$$(b) |H| \geq \frac{\log u}{\log v}.$$

EXERCISES

Exercise 7.1: Let $m \geq n \geq 1$. What is the probability that a random function in $[\mathbb{Z}_n \rightarrow \mathbb{Z}_m]$ is perfect? Compute this probability if $m = 13, n = 10$. Or if $m = n = 10$? \diamond

Exercise 7.2: Compare the relative merits of the FKS scheme and the scheme in theorem ?? for constructing perfect hash functions. What are the respective program sizes in these two schemes? \diamond

Exercise 7.3: Let $x = (x_1, \dots, x_n)$ be a vector of real numbers. Let $f(x) = \prod_{i=1}^n x_i$ and $g(x) = \sum_{i=1}^n x_i$. We want to maximize $f(x)$ subject to $g(x) = c$ (for some constant $c > 0$) and also $x_i \geq 0$ for all i . HINT: a necessary condition according to the theory of Lagrange multipliers is that $\nabla f = \lambda \nabla g$ for some real number λ . Why is this also sufficient? \diamond

END EXERCISES

§8. Extendible Hashing

The previous section restricts dictionaries to be static. We now investigate a generalization. Extendible hashing [3] is a technique to overcome the most obvious handicap of hashing – the size of the hash table is fixed in advance. It can also be an alternative to B -trees, which are extensively used in database management. Extendible hashing has a two-level structure comprising a **directory** and a variable set of **pages**. The directory is usually small enough to be in main memory while the pages store items and are kept in secondary memory. See figure 2 for an illustration.

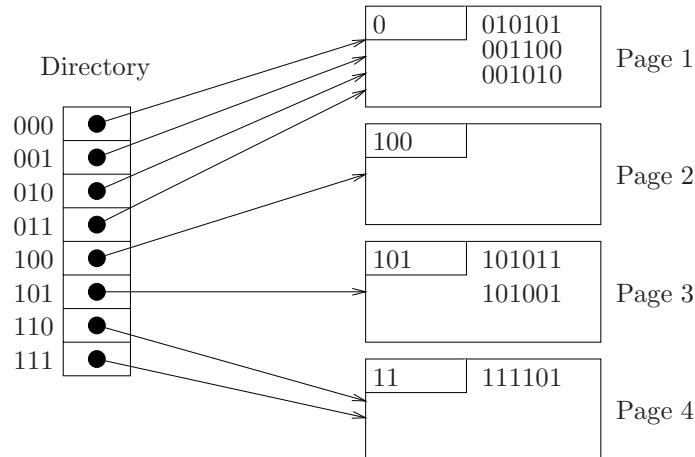


Figure 2: Extendible Hashing data structure: some hash values in the pages represents items stored under that hash value

We postulate a hash function of the form

$$h : U \rightarrow \{0, 1\}^L$$

for some $L > 1$. All pages have the same size, say, accommodating B items. Each page has its own **prefix** which is a binary string of length at most L . An item with key k will be stored in the page whose prefix p is a prefix of $h(k)$. For instance, in page 1 of figure 2, we store three items (as represented by the hash values of their keys: 010101, 001100 and 001010). The **depth** of the page is the length of its prefix. The **depth of the directory**, denoted by d , is the maximum depth of the pages. We require that the collection of page prefixes forms a prefix-free code. Recall (§IV.1, Huffman code) that a set of strings is a prefix-free code if no string in the set is a prefix of another. For instance, in figure 2, the prefix of each page is shown in the top left corner of the page; these prefixes form the prefix-free code

$$0, 100, 101, 11.$$

A directory of depth d is an array of size 2^d , where the entry $T[i]$ is a pointer to the page whose prefix is a prefix of the binary representation of i . So if a page has prefix of depth $d' \leq d$ then there will be $2^{d-d'}$ pointers pointing to it.

The actual storage method within a page is somewhat independent of extendible hashing method. For instance, any hashing scheme that uses a fixed size table but no extra storage will do (e.g., coalesced chaining or open addressing schemes). Search times for extendible hashing thus depends on the chosen method for organizing pages. It can be shown that the expected number of pages to store n items is about $n(B \ln 2)^{-1}$. This means that the expected load factor is $\ln 2 \approx 0.693$.

Knuth [7] is the basic reference on the classical topics in hashing. The article [4] considers minimal perfect hash functions for large databases.

References

- [1] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *J. of Computer and System Sciences*, 18:143–154, 1979.
- [2] Z. J. Czech, G. Havas, and B. S. Majewski. Perfect hashing. *Theor. Computer Science*, 182:1–143, 1997.
- [3] R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong. Extendible hashing – a fast access method for dynamic files. *ACM Trans. on Database Systems*, 4:315–344, 1979.
- [4] E. A. Fox, L. S. Heath, Q. F. Chen, and A. M. Daoud. Practical minimal perfect hash functions for large databases. *J. of the ACM*, 35(1):105–121, 1992.
- [5] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. of the ACM*, 31:538–544, 1984.
- [6] L. K. Hua. *Introduction to Number Theory*. Springer-Verlag, Berlin, 1982.
- [7] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, Boston, 1972.
- [8] P. K. Pearson. Fast hashing of variable-length text strings. *Comm. of the ACM*, 33(6):677–680, 1990.
- [9] W. W. Peterson. Addressing for random access storage. *IBM Journal of Research and Development*, 1(2):130–146, 1957. Early major paper on hashing –perhaps the second paper on hashing? See Knuth v.3.