Notes, 8(b)

ECE 606

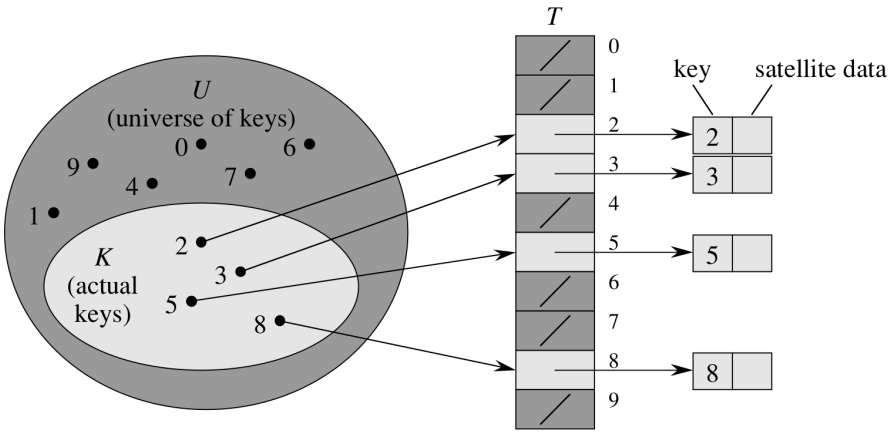
Hash Tables

A good example of the use of randomization is the Hash Table data structure.

Intent with a hash table: storage of a set keys + satellite data, and then, an efficient algorithm to check whether a given key k is in the set. Depending on application, may need to insert and/or delete as well.

Assumption: the keys we store are drawn from a universe, U. E.g., all strings in the alphabet $\{0, \ldots, 9\}$ of length $= 8$.

If $|U|$ is small, simply use a "direct address table," with a sentinel NIL.



DIRECT-ADDRESS-SEARCH(T, k)
  return $T[k]$

DIRECT-ADDRESS-INSERT(T, x)
  $T[key[x]] \leftarrow x$

DIRECT-ADDRESS-DELETE(T, x)
  $T[key[x]] \leftarrow$ NIL

All operations are $\Theta(1)$-time.

Think of yourself as the designer of the data structure. You are told:

- Universe U from which keys are drawn.
- Possible that $|U| >> |K|$.

Hash table: a table $T[0, 1, \ldots, m-1]$ of m slots, and a function $h\colon U \to \{0, \ldots, m-1\}$.

- You, as designer, get to choose m, h.
- Ideally, space $= O(|K|)$. So, ideally, $m = O(|K|)$.
- h should be computable efficiently.
- Search-time should be $\Theta(1)$, i.e., independent of $|K|$.

A solution: similar to direct address table: to insert x, $T[h(key[x])] \leftarrow x$.

A potential issue: *collisions*. A collision is two keys that hash to the same slot.

- Sometimes, prudent choice of h can mitigate collisions. What is such a prudent choice?
- Notwithstanding, for $|U| >> |K|$, collisions are guaranteed immaterial of h.
  - E.g., if $|U| > m|K|$, then immaterial of our choice of h, there exists $K \subseteq U$ such that all keys in K hash to the same slot.
- So question: how do we *resolve* collisions?

Two standard techniques that have been investigated:

- Chaining: each slot in T is a linked list of keys from K that hash to that slot.
- Open addressing: find another slot in T to store a colliding entry.

CHAINED-HASH-SEARCH(T, k)
search for item with key k in the list $T[h(k)]$

CHAINED-HASH-INSERT(T, x)
insert x at the head of the list $T[h(key[x])]$

CHAINED-HASH-DELETE(T, x)
delete x from the list $T[h(key[x])]$

Worst-case time-efficiency:

- Insert: $\Theta(1)$
- Search, Delete: $\Theta(n)$, where $n = |K|$.

But in expectation? Depends on $h\colon U \to \{0, 1, \ldots, m-1\}$.

- *Simple uniform hashing*:
  for all $k \in U, i \in \{0, \ldots, m-1\}$, $\Pr\{h(k) = i\} = 1/m$

Let $\alpha = n/m$, *load factor*.

Claim 1. *Both a successful and an unsuccessful search, under simple uniform hashing, have expected time-efficiency of* $\Theta(1+\alpha)$.

Let n_j be length of linked list at $T[j]$. Then, $E[n_j] = n/m$. So, expected time-efficiency of unsuccessful search $= \Theta(1+\alpha)$.

For successful search, depends on which item we are looking for. Assume: every key in T is equally likely to be searched for. We compute the expectation of the average over all the keys in T.

Assume the n keys were inserted in the order: $k_1, k_2, \ldots, k_n$. Let $X_{i,j} = I\{h(k_i) = h(k_j)\}$. Then, expected # items compared in a successful search:

$$\begin{aligned} E\left[\frac{1}{n}\sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n} X_{i,j}\right)\right] &= \frac{1}{n}\sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n} E[X_{i,j}]\right) \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n} \frac{1}{m}\right) \\ &= 1 + \frac{\alpha}{2} - \frac{\alpha}{2n} = \Theta(1+\alpha) \end{aligned}$$

Issue: how to ensure good h that performs like in simple uniform hashing?

Turns out, there is more practical approach: *universal hashing*. We can achieve this provided we can generate sufficiently large prime numbers.

Given U, m, let $\mathscr{H}$ be a set of (hash) functions each with domain U and codomain $\{0, 1, \ldots, m-1\}$. We call such a set $\mathscr{H}$ *universal* if: for any two distinct $k, l \in U$, the number of functions $h \in \mathscr{H}$ such that $h(k) = h(l)$ is $\leq |\mathscr{H}|/m$.

Claim 2. *If we pick $h \in \mathscr{H}$ uniformly at random, where $\mathscr{H}$ is universal, then for any two distinct keys $k, l \in U$, $Pr\{h(k) = h(l)\} \leq 1/m$.*

So here is our process to designing our hash table. We are told U and perhaps $|K|$. Based on that, we pick an m and generate a universal set $\mathscr{H}$ for that U and m. Then, we pick h uniformly from $\mathscr{H}$. We store our chosen h along with T.

Claim 3. *For h chosen as above, let $n_{h(k)}$ be the length of the chain that corresponds to key $k \in U$. Then, if k is not in the table, $E\left[n_{h(k)}\right] \leq \alpha$. If k is in the table, then $E\left[n_{h(k)}\right] \leq 1+\alpha$.*

Claim 4. *Under universal hashing with collisions resolved by chaining, it takes expected time $\Theta(n)$ to handle any sequence of n* Insert, Search *and* Delete *operations that contain $O(m)$* Insert *operations.*

Textbook includes a way to generate a universal class of hash functions given a large enough prime number. We need some basic number theory to prove that it is indeed universal, which is not part of the course.

It turns out that if we know K before we design our hash table, and it is static, i.e., no insertions nor deletions, then using an approach called *perfect hashing*, we can carry out SEARCH in worst-case $O(1)$-time with the space we need, in expectation, $O(n)$ only.

We first make the following observation about a regular hash table, under universal hashing.

Claim 5. *If we store $|K| = n$ keys in a hash table of size $m = n^2$ using an h chosen uniformly from a universal set, then ~~$Pr\{h(k) = h(l)\}$ for any two $k, l \in K$~~ the probability that the number of collisions is at least 1 is $< 1/2$.*

Proof. Let X be the random variable that is the number of collisions. Then,

$$\begin{aligned} E[X] &= \binom{n}{2} \cdot \frac{1}{m} \\ &= \frac{n^2 - n}{2} \cdot \frac{1}{n^2} = \frac{1}{2} - \frac{1}{2n} < 1/2 \end{aligned}$$

$$\begin{aligned} \text{But, } E[X] &= 1 \cdot \Pr\{X = 1\} + 2 \cdot \Pr\{X = 2\} + \ldots + \binom{n}{2} \cdot \Pr\left\{X = \binom{n}{2}\right\} \\ &\geq \Pr\{X = 1\} + \Pr\{X = 2\} + \ldots + \Pr\left\{X = \binom{n}{2}\right\} = \Pr\{X \geq 1\} \end{aligned}$$

Therefore, $\Pr\{X \geq 1\} < 1/2$

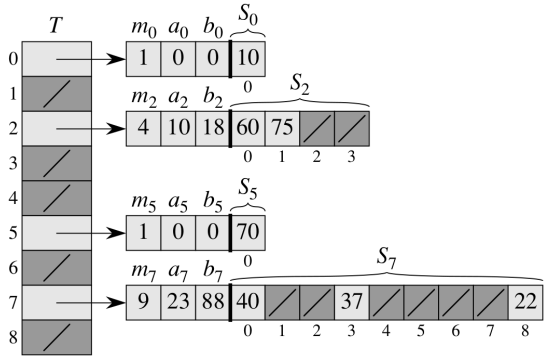
□

So: if we are given enough space, specifically, $m \geq n^2$, we can ensure that we have no collisions the following way. Uniformly picking h from the universal set and check for collisions. If there is a collision, repeat, i.e., pick h uniformly again. Within a few trials, we will find a hash function which yields no collisions for our K. E.g., $\Pr\{\cdot\}$ that after 10 trials we still have not found one is $< 2^{-10} < 10^{-3}$.

Following is how perfect hashing work. We allocate $m = n$ for the "outer" level hash table. For each slot $i \in \{0, \ldots, m-1\}$, we identify the number of items, n_i, that hash to it, under universal hashing. Then, for each $n_i > 0$, we: (i) have another hash table with $m_i = n_i^2$ slots and pick a hash function from a corresponding universal set such that we have no collisions in each "inner" hash table. Of course, we need to store each hash function with its corresponding table.

Thus, to SEARCH for k, we first compute $i \leftarrow h(k)$. Then, we compute $h_i(k)$ and check whether it is NIL, where h_i is the hash function for the inner hash table that corresponds to slot i of the outset table.

Following the example picture from your textbook, in which each inner hash function is represented as a triple $\langle m_i, a_i, b_i \rangle$.



Claim 6. *Under perfect hashing,* $E\left[\sum_{j=0}^{m-1} n_j^2\right] < 2n$.

Claim 7. *Under perfect hashing,* $Pr\left\{\sum_{j=0}^{m-1} n_j^2 > 4n\right\} < 1/2$.

The second claim above suggests a similar strategy as we discuss in the previous page: test a few randomly chosen hash functions from each corresponding universal family to ensure that the total space requirement is not large.