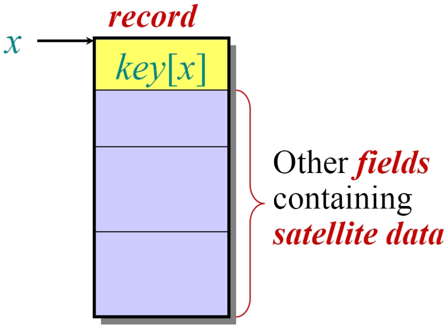
Lecture: Analysis of Algorithms (CS583 - 004)

Amarda Shehu

Spring 2019

Symbol-table Problem

- Consider the following symbol table S containing n records:



- Consider basic operations on S:
    - INSERT(S,x)
    - DELETE(S, x)
    - SEARCH(S, k)

Question: How should the data structure S be organized?

Direct-access Hash Tables

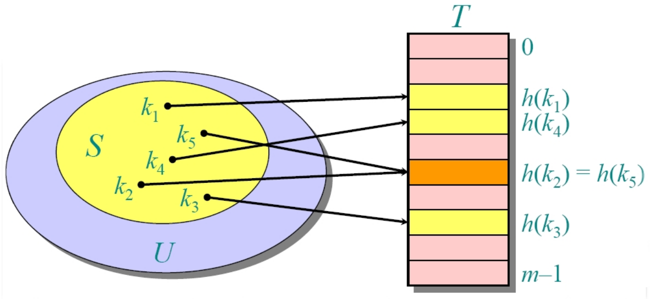
- Suppose we have a list of elements S with keys that are drawn from a set (universe) $U \subseteq \{0, 1, \ldots, m-1\}$
- Suppose that keys are distinct at this stage (no two elements have the same key)
- Set up an array $T[0, 1, \ldots, m-1]$ s.t.:

$$T[k] = \begin{cases} x & \text{if } x \in S \text{ and } \text{key}[x] = k \\ \text{NULL} & \text{otherwise} \end{cases}$$

- All the basic operations take $\theta(1)$ time
- There is a potential problem - the range of keys can be large:
  - example: 64-bit numbers (18,446,744,073,709,551,616 keys)
  - example: character strings (even larger)

Hash Functions

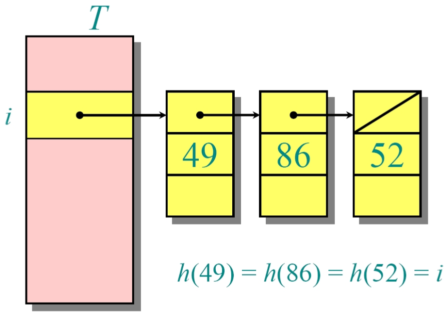
Solution: Use a *hash function* h to map the universe U of all keys into $\{0, 1, \ldots, m-1\}$:



- As each key is inserted, h maps it to a slot of T.
- When a record to be inserted maps to an already occupied slot in T, a *collision* occurs.

Resolving Collisions by Chaining

- Link records that hash to same slot into a list



- Worst-case Scenario:
  - Every key hashes to same slot
  - Access time is $\theta(n)$ if $|S| = n$

Average-case Analysis of Chaining Collisions

We make the assumption of *simple uniform hashing*:

- Each key $k \in S$ is equally likely to be hashed to any slot of table T, independent of where other keys are hashed.
- Let n be number of keys in table, and m be number of slots.
- Let load factor $\alpha = n/m$ be average number of keys per slot.

- Expected time of an *unsuccessful* search for a record with a given key is: (time to apply hash function and get slot) $+$ (time to search the "collision" list) $\in \theta(1 + \alpha)$.
- Expected search time is $\theta(1)$ if $\alpha \in \theta(1)$ (when $n \in O(m)$).
- A *successful* search has same asymptotic bound (demonstrate on board).

Choosing a Hash Function

Can we achieve simple uniform hashing?

- Assumption of simple uniform hashing is hard to guarantee.
- Common techniques tend to work well in practice as long as their deficiencies can be avoided.

Desired Property:

- A good hash function should distribute the keys uniformly into the slots of the table.
- Regularity in the key distribution should not affect this uniformity.

Division Method

- Assume all keys are integers, and define $h(k) = k \bmod m$.
- Deficiency: Do not pick an m that has a small divisor d. A preponderance of keys that are congruent mod d can adversely affect uniformity.
- Extreme deficiency: If $m = 2^r$, then the hash does not even depend on all bits of k:
  - If $k = 1011000111011010_2$ and $r = 6$, then $h(k) = 011010_2$, which is only last 6 bits of k
- Pick m to be: prime, not too close to a power of 2 or 10, and not otherwise used prominently in the computing environment.
- Annoyance: A prime table size may be inconvenient. This method is popular, although the next one is usually superior.

Multiplication Method

Assume that all keys are integers, $m = 2^r$, and our computer has w-bit words. Define:

$$h(k) = (A \cdot k \bmod 2^{\mathrm{w}}) \text{ rsh } (\mathrm{w} - \mathrm{r}),$$

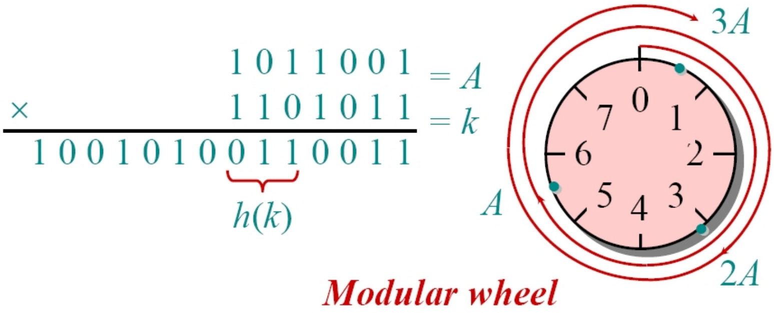
where rsh is the bitwise right-shift operator, and A is an odd integer in the range $2^{w-1} < A < 2^w$.

- Do not pick A too close to 2^{w-1} or 2^w.
- Multiplication modulo 2^w is fast compared to division.
- The rsh operator is fast.

Example of Multiplication Method

$$h(k) = (A \cdot k \bmod 2^{\mathrm{w}}) \text{ rsh } (\mathrm{w} - \mathrm{r}),$$

Suppose $m = 8 = 2^3$ and that our computer has $w = 7$-bit words:



Resolving Collisions by Open Addressing

No storage is used outside of the hash table itself.

- Insertion systematically probes table until empty slot is found.
- The hash function depends on both the key and probe number:

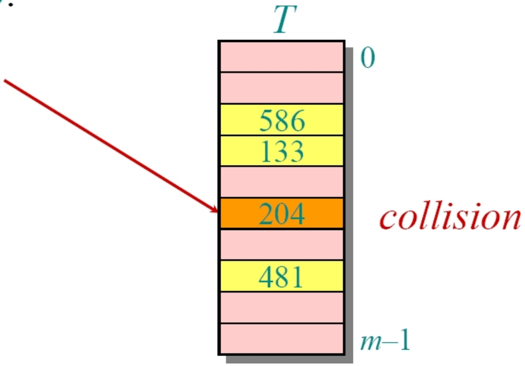
$$h : U \times \{0, 1, \ldots, m-1\} \rightarrow \{0, 1, \ldots, m-1\}$$

- The probe sequence $\langle h(k, 0), h(k, 1), \ldots, h(k, m-1) \rangle$ is a permutation of $\{0, 1, \ldots, m-1\}$.
- Potential hazards:
  - the table may fill up
  - deletion is difficult (but not impossible)

Example of Open Addressing

Insert key $k = 496$:

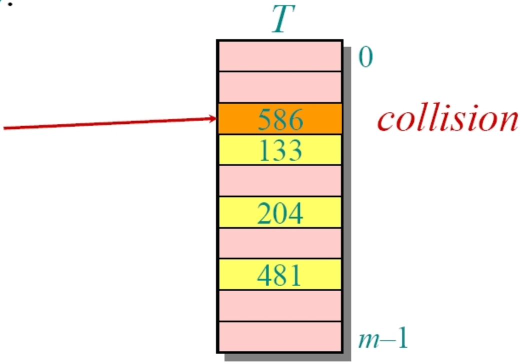
0. Probe $h(496,0)$



Example of Open Addressing

Insert key $k = 496$:

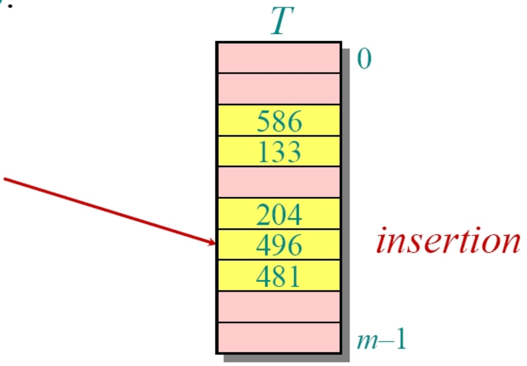
0. Probe $h(496,0)$
1. Probe $h(496,1)$



Example of Open Addressing

Insert key $k = 496$:

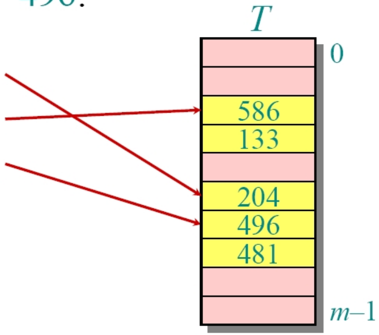
0. Probe $h(496,0)$
1. Probe $h(496,1)$
2. Probe $h(496,2)$



Example of Open Addressing

Search for key $k = 496$:

0. Probe $h(496,0)$
1. Probe $h(496,1)$
2. Probe $h(496,2)$



Search uses the same probe sequence and terminates successfully if it finds the key; unsuccessfully if it encounters an empty slot.

Probing Strategies

Linear probing: Given an ordinary hash function $h^{'}(k)$, linear probing uses the hash function $h(k, i) = (h^{'}(k) + i) \bmod m$. This method, though simple, suffers from *primary clustering*, where long runs of occupied slots build up, increasing the average search time. Moreover, the long runs of occupied slots tend to get longer.

Double Hashing: Given two ordinary hash functions $h_1(k)$ and $h_2(k)$, double hashing uses the hash function
$h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod m$.
This method generally produces excellent results, provided that $h_2(k)$ is relatively prime to m. One way is to make m power of 2 and design $h_2(k)$ to produce only odd numbers.

Analysis of Open Addressing

We make the assumption of uniform hashing: Each key is equally likely to have anyone of the $m!$ permutations as its probe sequence.

Theorem: Given an open-addressed hash table with load factor $\alpha = n/m < 1$, the expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$.

Proof of Theorem

- At least one probe is always necessary.
- With probability n/m, the first probe hits an occupied slot, and a second probe is necessary.
- With probability $(n-1)/(m-1)$, the second probe hits an occupied slot, and a third probe is necessary.
- With probability $(n-2)/(m-2)$, the third probe hits an occupied slot, and a fourth probe is necessary. (and so on)
- So, the expected number of probes is $1 + \frac{n}{m} \cdot (1 + \frac{n-1}{m-1} \cdot (1 + ...)))$.
- Observe that $\frac{n-i}{m-i} \leq \frac{n}{m}$ for $1 \leq i \leq n$.
- So, expected number of probes is bounded above by $1 + \alpha(1 + \alpha(1 + \alpha(....))) = 1 + \alpha + \alpha^2 + \ldots = \sum_{i=0}^{\infty} \alpha^i = \frac{1}{1-\alpha}$.
- A rigorous argument includes successful search in CLRS book.

Implications of this Theorem

- If α is constant, then accessing an open-addressed hash table takes constant time.
- If the table is half full, then the expected number of probes is $1/(1\text{-}0.5) = 2$.
- If the table is 90% full, then the expected number of probes is $1/(1-0.9) = 10$.

A weakness of hashing

Problem: For any hash function h, a set of keys exists that can cause the average access time of a hash table to skyrocket.

- An adversary can pick all keys from $\{k \in U : h(k) = i\}$ for some slot i.

Solution: Choose the hash function at random, independently of the keys.

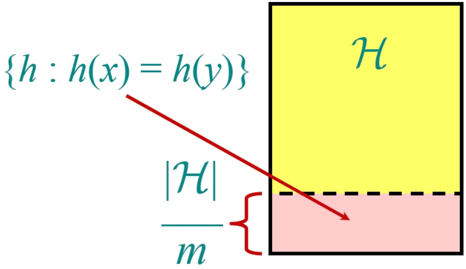
- Even if an adversary can see your code, he or she cannot find a bad set of keys, since he or she does not know exactly which hash function will be chosen.

Universal Hashing

Definition of Universality

Let U be a universe of keys, and let $\mathcal{H}$ be a finite collection of hash functions, each one mapping U to $\{0, 1, \ldots, m-1\}$.
We say that $\mathcal{H}$ is *universal* if $\forall x, y \in U$, where $x \neq y$, we have $|h \in \mathcal{H} : h(x) = h(y)| = |\mathcal{H}|/m$.

This means that the chance of a collision between x and y is $1/m$ if we choose h randomly from $\mathcal{H}$.



Why is Universality So Important?

Theorem: Let h be a hash function chosen (uniformly) at random from a universal set $\mathcal{H}$ of hash functions. Suppose h is used to hash n arbitrary keys into the m slots of a table T. Then, for a given key x, we have $E[\#\text{collisions with } x] < n/m$.

Proof: Let C_x be the random variable indicating the total number of collisions of keys in T with x, and let

$$c_{xy} = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$$

Note: $E[c_{xy}] = 1/m$ and $C_x = \sum_{y \in T - \{x\}} c_{xy}$.

Continuing Proof that Universality is Important

$E[C_x]$	$= E[\sum_{y \in T-\{x\}} c_{xy}]$	take E of both sides
	$= \sum_{y \in T-\{x\}} E[c_{xy}]$	linearity of expectation
	$= \sum_{y \in T-\{x\}} 1/m$	$E[c_{xy}] = 1/m$
	$= \frac{n-1}{m}$	simple Algebra

Constructing a Set of Universal Hash Functions

Let m be prime. Decompose k into $r+1$ digits, each one with value in the set $\{0, 1, \ldots m-1\}$. That is, let $k = \langle k_0, k_1, \ldots, k_r \rangle$, where $0 \leq k_i < m$.

Randomized Strategy: Pick $a = \langle a_0, a_1, \ldots, a_r \rangle$, where each a_i is chosen uniformly at random from $\{0, 1, \ldots, m-1\}$.

Define $h_a(k) = \sum_{i=0}^{r}(a_i k_i) \bmod m$

Hash functions defined this way are known as dot-product hash functions

How big is $\mathcal{H} = \{h_a\}$?

$|\mathcal{H}| = m^{r+1}$ - this is important to remember

Universality of Dot-product Hash Functions

Theorem: The set $\mathcal{H} = \{h_a\}$ is universal

Proof: Suppose that $x = \{x_0, x_1, \ldots, x_r\}$ and $y = \{y_0, y_1, \ldots, y_r\}$ are distinct keys. So, x and y differ in at least one digit position.

Let this digit position be 0. For how many $h_a \in \mathcal{H}$ do x and y collide?

In case of a collision, we have that $h_a(x) = h_a(y)$, which implies that $\sum_{i=0}^{r} a_i x_i = \sum_{i=0}^{r} a_i y_i \pmod{m}$

Equivalently, $\sum_{i=0}^{r} a_i(x_i - y_i) = 0 \pmod{m}$

So, $a_0(x_0 - y_0) + \sum_{i=1}^{r} a_i(x_i - y_i) = 0 \pmod{m}$, which implies that $a_0(x_0 - y_0) = -\sum_{i=1}^{r} a_i(x_i - y_i) \pmod{m}$

Continuing Proof with Help of Number Theory

Theorem: Let m be prime. For any $z \in Z_m$ s. t. $z \neq 0$, $\exists$ a unique $z^{-1} \in Z_m$ s.t. $z \cdot z^{-1} = 1 \pmod{m}$

Example: When $m = 7$, pairs of (z, z^{-1}) are $(1,1)$, $(2,4)$, $(3,5)$, $(4,2)$, $(5,3)$, $(6,6)$.

Back to our Proof: Since $a_0(x_0 - y_0) = -\sum_{i=1}^{r} a_i(x_i - y_i) \pmod{m}$ and $x_0 \neq y_0$, then $\exists$ $(x_0 - y_0)^{-1}$ and so we have:

$$a_0 = (-\sum_{i=1}^{r} a_i(x_i - y_i)) \cdot (x_0 - y_0)^{-1} \pmod{m}$$

This means that for any choices of $a_1, a_2, \ldots, a_r$, exactly one choice of a_0 causes x and y to collide

Completing the Proof

Question: How many h_a's cause x and y to collide?

Answer: There are m choices for each of $a_1, a_2, \ldots, a_r$, but once these are chosen, exactly one choice for a_0 causes x and y to collide. We showed that:

$$a_0 = (-\sum_{i=1}^{r} a_i(x_i - y_i)) \cdot (x_0 - y_0)^{-1} \pmod{m}$$

So we can conclude that the number of h_a's that cause x and y a to collide is $m^r \cdot 1 = m^r = |\mathcal{H}|/m$.

Perfect Hashing

A hash function is perfect for a dictionary S if all lookups involve $O(1)$ work. There are two methods to construct perfect hash functions for a given set S.

- Technique 1 reguires $O(n^2)$ storage, where $|S| = n$. This means we are willing to have a hash table whose size is quadratic in the size of S. Then, a perfect hash function can easily be constructed. Let $\mathcal{H}$ be universal and $m = n^2$. Then just pick a random h from $\mathcal{H}$ and try it out. The claim is there is at least a 50% chance there will be no collisions.
- Technique 2 requires $O(n)$ storage. A hash table of size n is first filled through universal hashing, which may produce some collisions. Each slot of the table is then rehashed (universal hashing is again employed), squaring the size of the slot in order to get 0 collisions. This is also called a two-level scheme.

Perfect Hashing: $O(n^2)$ Storage

Theorem: Let $\mathcal{H}$ be a class of universal hash functions for a table of size $m = n^2$. Then, if we use a random $h \in \mathcal{H}$ to hash n keys into the table, the expected number of collisions is at most $1/2$.

Proof: By the definition of universality, the probability that 2 given keys in the table collide under h is $1/m = 1/n^2$. Since there are $\binom{n}{2}$ pairs of keys that can possibly collide, the expected number of collisions is:

$$\binom{n}{2} \cdot \frac{1}{n^2} = \frac{n \cdot (n-1)}{2} \cdot \frac{1}{n^2} < \frac{1}{2}$$

Guaranteeing No Collisions in Perfect Hashing

Corollary: The probability of no collisions is at least $1/2$.

Proof: Markov's inequality says that for any nonnegative random variable X, we have $P(X \geq t) \leq E[X]/t$.

Applying this inequality with $t = 1$, we find that the probability of 1 or more collisions is at most $1/2$.

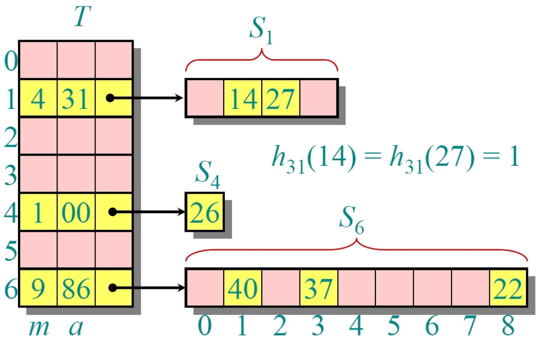
This means that just by testing random hash functions in $\mathcal{H}$, we quickly find one that works. The claim is we do not have to try more than two.

Perfect Hashing: $O(n)$ Storage

Given a set of n keys, construct a static hash table of size $m \in O(n)$ s. t. searching for a key takes $\theta(1)$ time in the worst case.

Idea: Two-level scheme with universal hashing at both levels.

No collisions at level 2!



Analysis of Storage in Perfect Hashing

For the level-1 hash table T, choose $m = n$, and let n_i be the random variable for the number of keys that hash to slot i in T.

By using n_i^2 slots for the level-2 hash table S_i, the expected total storage required for the two-level scheme is:

$$E[\sum_{i=0}^{m-1} \theta(n_i^2)] = \theta(n)$$

The analysis is identical to the analysis of the expected running time of bucket sort. It can be shown, with some mathematical manipulations, that $E[\sum_{i=0}^{m-1}(n_i^2)] < 2n$. Then, Markov's inequality gives a probability bound that $P(\sum_{i=0}^{m-1} \theta(n_i^2) > 4n) < 1/2$.
If we test a few randomly chosen hash functions from the universal family, we will quickly find one that uses a reasonable amount of storage.