CSE 241 Class 7

Jeremy Buhler

September 21, 2015

Today: analysis of hashing

1 Constraints of Double Hashing

How does using OA w/double hashing constrain our hash function design?

- Need to avoid bad behavior of slot sequences. For example, suppose $m = 6$, but $h_2(k) = 3$? We only ever touch two slots of table!
- Recall $s_i = (h_1(k) + ih_2(k)) \bmod m$.
- For double hashing, want slot sequence to be as long as table size m
- To ensure non-repetition for $i < m$, suffices to require that

$$s_i \neq s_0, 1 \leq i < m$$

- By definition of our slot sequence, this means

$$ih_2(k) \not\equiv 0 \pmod{m}, 1 \leq i < m$$

- true iff $\gcd(h_2(k), m) = 1$.
- *One possibility*: make m a prime number – every smaller step size is OK.
- Requires finding suitable primes for a range of possible table sizes, and computing indices modulo these primes (could be expensive!)
- *Alternative*: make m a power of 2, and ensure that $h_2(k)$ is always an odd number!
- Avoids issues with general primes, but reduces the space of step values by half – possibly more collisions.

2 Hashing Performance Model

Worst-case performance of hashing is a dismal $\Theta(n)$. How can we do a more useful performance analysis?

- study **average case** behavior

- first, assume we have a "good" hash function
- assume **simple uniform hashing**:

  1. Suppose hash function $h(k)$ maps keys to a range $0 \ldots m-1$.
  2. Each key is equally likely to map to each slot in the table, independent of all others.
  3. That is, for each key k and slot s,

$$\Pr[k \text{ maps to slot } s] = \frac{1}{m}$$

What is a sensible measure of performance for hashing?

- **find is the important operation**; in general, searching the table is what we care about
- time spent searching is proportional to **number of collisions**
- **for chaining**: collisions with key k determined by length of chain in slot $h(k)$
- **for open addressing**: collisions with key k determined by length of slot sequence for k until first empty slot found.

What is a sensible average case?

- table holds n keys
- keys in table were chosen at random from keyspace, so their distribution over slots is as predicted by SUH.
- we search for an arbitrary key (in table or not)

What are limitations of this model? (1) imperfect hash functions are not really uniform; (2) table contents may not be "random." Can try to improve (1), but nothing to be done about (2) if *adversary* gets to pick keys to insert, then picks search keys maliciously to maximize running time.

3 Chaining in Particular

- an unsuccessful search always traverses its entire chain
- for a successful search, the record is equally likely to be anywhere in its chain (since chain contents were chosen in a random order)
- *conclude*: average collisions for searches in a chain is $\Theta(\text{chain length})$, so average time to search is $\Theta(1 + \text{chain length})$.
- Because insertion process chooses keys randomly, and the hash function distributes them uniformly, every chain must have same *average* length (symmetry!).
- *conclude*: for an arbitrary search key, average search time is proportional to **average chain length** in table!

How can we compute average chain length?

4 Probability Background

There are a couple of ways to compute the average chain length in a hash table. I'm going to show you one that uses an important basic analysis trick: *linearity of expectation.*

(Review of probability: CLR Appendix C)

- **Reminder 1**: marginal probabilities
- Let x, y be random variables over sets A, B (need not be independent)
- sample simultaneously from A, B
- Can write **joint probability** $\Pr(x = a \wedge y = b)$ for any $a \in A$, $b \in B$.
- What is **marginal probability** $\Pr(x = a)$ by itself?

$$\Pr(x = a) = \sum_{b \in B} \Pr(x = a \wedge y = b)$$

- Easy to see with diagram:

- **Reminder 2**: definition of expectation
- Let x be a numerically-valued random variable over set A
- the *expected value of* x, denoted $E[x]$, is given by

$$E[x] = \sum_{a \in A} a \Pr(x = a)$$

- If every value of x is equiprobable (i.e. prob is $\frac{1}{|A|}$), expectation is just the usual notion of average

These two reminders are sufficient to prove *linearity of expectation*, a very powerful idea.

Theorem: for any two random variables x and y,

$$E[x + y] = E[x] + E[y].$$

(Note that the variables need not be independent!)

Proof: assume x and y are r.v.'s over sets A, B.

$$\begin{aligned}
E[x+y] &= \sum_{a\in A}\sum_{b\in B}(a+b)\Pr(x=a\wedge y=b) \\
&= \sum_{a\in A}\sum_{b\in B}a\Pr(x=a\wedge y=b)+\sum_{a\in A}\sum_{b\in B}b\Pr(x=a\wedge y=b) \\
&= \sum_{a\in A}a\sum_{b\in B}\Pr(x=a\wedge y=b)+\sum_{b\in B}b\sum_{a\in A}\Pr(x=a\wedge y=b) \\
&= \sum_{a\in A}a\Pr(x=a)+\sum_{b\in B}b\Pr(y=b) \\
&= E[x]+E[y] \qquad \text{QED.}
\end{aligned}$$

5 Average Chain Length

- Let L_s be the length of the chain in slot s of the table
- We want to compute average chain length $E[L_s]$ after adding n randomly chosen keys to table.

We will use the idea of *indicator random variables.* Define

$$x_{is}=\begin{cases}1 & \text{if key } i \text{ hashes to slot } s\\ 0 & \text{otherwise.}\end{cases}$$

Notice that

$$L_s=\sum_{i=1}^{n}x_{is}.$$

[stop and explain]
Observe that

$$\begin{aligned}
E[x_{is}] &= \Pr(\text{key } i \text{ hashes to slot } s) \\
&= \frac{1}{m}
\end{aligned}$$

by simple uniform hashing assumption.

By linearity of expectation, we have

$$\begin{aligned}
E[L_s] &= E\left[\sum_{i=1}^{n}x_{is}\right] \\
&= \sum_{i=1}^{n}E[x_{is}] \\
&= \sum_{i=1}^{n}\frac{1}{m} \\
&= \frac{n}{m}.
\end{aligned}$$

That last expression looks familiar! Remember load factor α for a hash table? We have shown that **under simple uniform hashing model,**

$$\text{average chain length} = \alpha = \frac{n}{m}.$$

Conclude that **if $\alpha = O(1)$ (i.e. table size is multiple of input size n), we do only $\Theta(1)$ work on average per search**! Hence, we normally set α to some small constant, e.g. $\frac{1}{3}$.

6 What About Open Addressing?

- Don't have time to do full analysis (CLR Sec 11.4), but will state result
- Assume simple uniform double hashing – slot sequence for a given key is a *random permutation* of $0 \ldots m-1$
- Can show that average length of slot sequence for *failed* search is at most

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

  (compare to α records checked on average failure with chaining)
- Can show that average length of slot sequence for *successful* search is at most

  $$\frac{1}{\alpha} \ln\left(\frac{1}{1-\alpha}\right)$$

- Even though slot sequences can cross, average number of checks is only a bit worse than with chaining for small α.

7 Choosing Good Hash Functions

What are criteria for good hash functions?

- approximate uniform distribution on $0 \ldots m-1$ in average case
- common key sequences should not cause worst-case behavior. E.g., if keys $1, 2, 3 \ldots$ might be inserted in the table, they shouldn't all hash to same slot.
- hash value should depend on *entire key*
- (*aside*: can we ever guard against malicious key sequences?)

Two basic kinds of hash function: **division** and **multiplication**

- **division**: for table of size m,

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

- what happens if m is power of 2?

- slot number ignores high-order bits of key (picture)

- similarly, if m is power of 10, slot number ignores high-order digits.
- much safer to use a modulus that is not close to a common counting base, e.g. a prime number p that is not close to a power of 2, 5, or 10.

Division method isn't particularly good because arbitrary integer division and modulus are *expensive operations* on modern computers.

- **multiplication**: let A be a constant, $0 < A < 1$.

$$h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$$

  (in other words, floor of m times the *fractional* part of kA).

- low-order bits of truncated multiply are pretty well scrambled
- A should probably not have a lot of repeating structure (e.g. 0.5 is bad, 1/3 is bad)
- *Good choice*: irrational such as $A = \frac{\sqrt{5}-1}{2}$ [Knuth]
- Choice of A should not be too small – otherwise, all smaller values of k will map to slot 0. (Suggest $A > 0.5$.)
- This method does not (by itself) constrain value of m
- If m is power of 2, can use shift and mask operations instead of multiply and floor to derive $h(k)$.