CS5101: Design and Analysis of Algorithms

Hash tables



Arijit Mondal

Dept of CSE

arijit@iitp.ac.in

https://www.iitp.ac.in/~arijit/

Hash tables

- Many applications require a dynamic set that supports only the dictionary operations insert, search, and delete
  - eg. Symbol table in compiler
- Hash table facilitates super-fast searches
- Conceptually, hash table can be thought as an array

Direct-address tables

- Suppose that an application needs a dynamic set in which each element has a distinct key drawn from the universe $U = \{1, \ldots, m-1\}$, where m is not too large
- To represent the dynamic set, we can use array (direct address table) $T[0 : m-1]$
- Slot-k points to an element with key k. If no elements with key k, then $T[k] = \mathit{null}$

Hash tables-1

Hash tables-2

- Direct address table is not viable when $|U|$ is very large
- With hash function h the key k goes into the slot $h(k)$
- So, $h : U \rightarrow \{0, \ldots, m-1\}$ where $m \ll |U|$
- An ideal hashing function h would have, for each possible input k in the domain U, an output $h(k)$ that is an element randomly and independently chosen uniformly from the range $\{0, \ldots, m-1\}$
- Two keys can hash to the same slot – **collision**
- Collisions are handled by chaining
- Given a hash table T with m slots that stores n elements, we deûne the load factor α for T as n/m, that is, the average number of elements stored in a chain
- Worst-case behavior of hashing with chaining is terrible

Search cost

- The expected time for an unsuccessful search for a record with a given key is $\Theta(1+\alpha)$
- Expected search time is $\Theta(1)$ if $\alpha = O(1)$ or equivalently if $n = O(m)$
- A successful search has same asymptotic bound $\Theta(1+\alpha)$

Choosing hash function

- The assumption of simple uniform hashing is hard to guarantee, but several common techniques tend to work well in practice as long as their deficiencies can be avoided
- 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 $h(k) = k \mod m$
- Don't pick an m that has a small divisor d
- If $m = 2^r$, then the hash doesn't even depend on all the bits of k
- Pick m to be a prime not too close to a power of 2 or 10 and not otherwise used prominently in the computing environment
- Annoyance: Sometimes, making the table size a prime is inconvenient

Multiplication method

- Define

$$h(k) = \lfloor m(kA \mod 1) \rfloor$$

  where A is in the range $0 < A < 1$, $kA \mod 1 = kA - \lfloor kA \rfloor$
- Assume that all keys are integers, $m = 2^r$, and our computer has w-bit words
- Multiplication-shift method, $a = A2^w$
- Hash function

$$h_a(k) = (ka \mod 2^w) \gg (w - r)$$

Thank you!