http://www.cs.unh.edu/~ruml/cs758

Searching

Dictionaries

'associative array', 'map', 'look-up table', 'set'

Dictionaries

'associative array', 'map', 'look-up table', 'set'

n items, key length k

Structure	Find	Insert	Delete
List (unsorted)			
List (sorted)			
Array (unsorted)			
Array (sorted)			
Heap			
Hash table			
Binary tree (unbalanced)			
Binary tree (balanced)			

Hash Tables

applications:

1. dictionaries
2. object method tables
3. string matching
4. set operations: $\cup, \cap, -$

first methods:

1. direct-address tables: small key range. eg, bit vectors.
2. chaining: deletion?

Time Complexity

n items in m buckets

time complexity of search $=$

Time Complexity

n items in m buckets

time complexity of search = number of items per bucket

assume nice hash: $P(h(i) = x) = 1/m$

Time Complexity

n items in m buckets
time complexity of search $=$ number of items per bucket
assume nice hash: $P(h(i) = x) = 1/m$
let X_i be 1 iff $h(i) = x$, 0 otherwise

$$\begin{aligned} E[\sum_{i=1}^{n} X_i] &= \sum_{i=1}^{n} E[X_i] \\ &= \sum_{i=1}^{n} 1/m \\ &= n/m \end{aligned}$$

let $\alpha = \frac{n}{m}$ 'load factor'
expected number of items per bucket is α
expected time is $\Theta(1 + \alpha)$

More Collisions

probability that k of n elements land in same of m bins:
let $\alpha = \frac{n}{m}$ 'load factor'

$$\binom{n}{k}\left(\frac{1}{m}\right)^k\left(1-\frac{1}{m}\right)^{n-k} \approx \frac{\alpha^k}{e^\alpha k!}$$

if $n = m$, $\approx \frac{1}{ek!}$:

k	probability
0	0.37
1	0.37
2	0.18
3	0.06
4	0.015
5	0.003
> 5	0.002 total

Open Addressing

1. linear probing: $h(k,i) = (h_1(k) + i) \mod m$ for increasing i
   - the runs
2. double hashing: $h(k,i) = (h_1(k) + ih_2(k)) \mod m$ for increasing i
   - requires: $h_2 \neq 0$, $h_2(k)$ and m relatively prime
   - eg, m prime and $h_2(k) < m$
   - or, $m = 2^x$ and $h_2(k)$ odd
3. cuckoo hashing: lookups $O(1)$, insertions amortized expected $O(1)$

moral: low load factor

deletion?

Break

- asst 2
- asst 3

Hash Functions

Hash Functions

$h : key \rightarrow 0..m-1$

1. mediocre is easy, good takes effort
2. want time (at most) linear in key size
3. perfect hashing is possible (and efficient) if keys known
   - linear time to construct, linear space to store
4. minimal perfect hashing is possible!

Hash Functions

$h : key \rightarrow 0..m - 1$

1. mediocre is easy, good takes effort
2. want time (at most) linear in key size
3. perfect hashing is possible (and efficient) if keys known
   - linear time to construct, linear space to store
4. minimal perfect hashing is possible!

bad news:

- if $|keys| \geq m$, there must be collisions
- if $|keys| \geq n \cdot m$, then $\exists$ set of n that map to same bin

Hash Functions

Desiderata:

- make collisions unlikely
  - spread keys across all hashes
  - for each key, each hash equally likely
- similar keys get different hashes
  - all bits of key affect the hash
  - every bit of key affects every bit of hash
- no input always gives worst-case behavior
- fast to compute
- low memory requirement
- easy to implement

Basic Multiplicative Hashing

1. *hash* $\leftarrow 0$
2. for each *byte* of key
3. *hash* $\leftarrow (hash \times multiplier) + byte$
5. return *hash* mod m

want *multiplier* to smear bits, not shift them (to avoid interaction with table size)

multiplier $=$ 31 or 127

Tabulation Hashing

assume we have an array of 256 random integers

1. *hash* $\leftarrow 0$
2. for each *byte* of key
3. rotate the bits in *hash* by 1
4. *hash* $\leftarrow$ *hash* xor *array*[*byte*]
5. return *hash* mod m

each byte affects all bits
rotate makes order matter

universal class of hash functions : for randomly chosen keys, randomly chosen function from class has $P(\text{collision}) = 1/m$

good on average case (over inputs) $\neq$ good average case on any input

Related Data Structures

Bloom Filters

set of n items using m bits with false positives

set addition: hash element to bit index, set to true

do this with k hash functions

no removal

for false positive rate ϵ:

$m/n = -\frac{\lg \epsilon}{\ln 2} \approx -1.44 \lg \epsilon$ and

$k = -\lg \epsilon$

independent of n or m!

for 1% false positive rate:

$=$ 9.6 bits per elt

$=$ 7 hash functions

Merkle Trees

a leaf node represents a data block, contains its hash
internal nodes contain hash of hashes of children
to verify that node's data is part of n block tree, $\lg n$ hashes are required

Blockchain

block contains hash of previous block, timestamp, transaction data

linked list version of Merkle tree

EOLQs

- What's still confusing?
- What question didn't you get to ask today?
- What would you like to hear more about?

Please write down your most pressing question about algorithms and put it in the box on your way out.
Thanks!