19 Mar 2019

Fun with Hashing

1. Dictionaries
2. Fingerprinting
3. Bloom filters
4. Count-min sketch

Topic: data structures for answering membership, approximate membership, counting queries in a set or multiset.

Standing assumptions:

Elements of the set/multiset are drawn from a universe of N potential elements. $N >>>> 1$.

The set/multiset has at most m elements in total.

$N >> m >> 1$.

($m \approx$ amount of space one might potentially allocate for a data structure.)

($N \approx$ exponentially greater than that.)

E.g. password checking $N \approx 2^{256}$, $m \approx 2^{19}$.

Elements may be inserted or queried, (usually) never deleted.

0. Trivial solution. Bit vector of length N.
   Too much space.

1. Deterministic solution. Balanced binary search tree (e.g. red/black) storing members of the set.
   Supports $O(\log m)$ insertion, deletion, lookup
   Space $O(m \log N)$.

2. Hash table.

chose based on m.

Array of size n.
Hash function $h: [N] \to [n]$.
Chosen randomly / pseudorandomly from a hash family $\mathcal{H}$, a set of functions $[N] \to [n]$.

A good hash function:

1 "looks random": for any $i \in [N]$ $h(i)$ unif distrib over $[n]$,
For any $i \neq j$, the pair $(h(i), h(j))$ unif distributed over $[n] \times [n]$. "pairwise independent"
For any $i_1, i_2, \ldots, i_k$ all distinct, $(h(i_1), \ldots, h(i_k))$ unif distrib over $[n]^k$ "k-wise indep"

2 Can be stored in small space and evaluated quickly.

Example of a pairwise independent h.

$h(x) := ax + b \pmod{n}$

$\left.\begin{array}{ll} a \in (\mathbb{Z}/(n)) & \text{random} \\ b \in (\mathbb{Z}/(n)) & \text{random} \end{array}\right\}$ uniformly

Interesting question: Fastest algo for generating a random number co-prime to n, given n in binary?

Read "Generating Random Factored Integers, Easily" by Adam Kalai.

Analysis. $h(x)$ unif distributed in $\mathbb{Z}/(n)$
because even holding a fixed,
so $a \cdot x$ is fixed, b is still random.
So $ax+b$ is unif distrib.

If n is prime, then for $x \neq y$,
$h(x)$ unif distrib.
$h(x) - h(y) = ax + \cancel{b} - ay \cancel{b} = a \cdot (x-y)$ unif distrib.

Hash function → dictionary.

Data structure is array of size n. Elements called "buckets."

1. Chain hashing: each bucket i stores a linked list of all $x \in S$ such that $h(x) = i$.

2. Linear probing: each bucket i stores at most one element of S.

insert(x): compute $h(x)$ and find first unoccupied bucket starting from $h(x)$. Insert x in first empty bucket.

query(x): start at $h(x)$ and search forward until:
- find x, answer "present"
- find empty bucket, answer "absent".

Both methods support $O_c(1)$ expected time per insert/query provided $n > c \cdot m$ for some $c > 1$.

Space requirement $O(m \cdot \log N)$ bits.

Advantage: $O(1)$ insert/query rather than $O(\log m)$

Disadvantage: randomized, running time guarantee only in expectation.

Trie: a data structure that matches these bounds asymptotically, deterministically.

Approximate membership using Bloom filters

Array $A[i]$ $(0 \le i < n)$. Array values are bits.

$$n = \left\lceil \left(\frac{k}{\ln 2}\right) \cdot m \right\rceil$$

k = parameter related to failure probability. E.g. $k=6$ or 7 usually a good choice.

∴ about 8-10 bits per element, in practice.

Hash functions $h_1, \ldots, h_k : [N] \to [n]$.

(Same k as before, $k=6$ or 7 typically)

$h_1, \ldots, h_k$ indep rand samples from $\mathcal{H}$.

insert (x): set $A[h_1(x)] = A[h_2(x)] = \cdots = A[h_k(x)] = 1$.

query (x): check if $A[h_1(x)], \ldots, A[h_k(x)]$ all equal 1.

Both operations take $O(k)$.

No false negatives.
Some false positives. $\Pr(\text{false positive}) = 2^{-k}$ if the array has half 1's, half 0's.

Sketchy analysis that shows why $A[\cdot]$ is half 1's, half 0's.

After inserting m elements.

$$\mathbb{E}[\#\text{ of occupied hash buckets}] = \sum_i \Pr[A[i] = 1]$$

$$= \sum_i \Pr(\exists x \in S \ \exists j \in [k] \ \ h_j(x) = i)$$

$$= \sum_i 1 - \Pr(\forall x \in S \ \forall j \in [k] \ \boxed{h_j(x) \neq i})$$

Assume these events all indep as j, x vary.

$$\approx \sum_i \left[1 - \prod_{x \in S} \prod_{j=1}^{k} \left(1 - \frac{1}{n}\right)\right]$$

$$= n \cdot \left[1 - \left(1 - \frac{1}{n}\right)^{km}\right]$$

$n = \frac{k}{\ln 2} m$, $km = n \cdot \ln(2)$.

$$= n\left[1 - \left\{\left(1 - \frac{1}{n}\right)^n\right\}^{\ln(2)}\right]$$

$$\approx \frac{n}{2}.$$