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. $N >> m >> 1$.

($m$ = amount of space we might potentially allocate for a data structure.)

($N$ = exponentially greater than that.)

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

(usually)

Elements may be inserted or queried, never deleted.

0. Trivial solution. Bit vector of length $N$.

Too much space.

1. Deterministic solution. Balanced binary search tree (e.g. red/black)

Supports $O(\log m)$ insertion, deletion, lookup.

Space $O(m \log N)$. 


2 Hash table. choose 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 \( i \in [N] \), \( h(i) \) unit distrib over \([n]\).
   For any \( i \neq j \), the pair \( (h(i), h(j)) \) unit distrib over \([n] \times [n] \). "pairwise independent"
   For any \( i, i_2, \ldots, i_k \) all distinct, \((h(i), \ldots, h(i_k))\) unit distrib over \([n]^k \). "k-wise indep"

2. Can be stored in small space and evaluated quickly.

Example of a pairwise independent \( h \).
\[
h(x) := ax + b \quad (\text{mod } n) \\
\begin{align*}
a & \in (\mathbb{Z}/n) \quad \text{random} \\
b & \in (\mathbb{Z}/n) \quad \text{random}
\end{align*}
\]

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. \( h(x) \) unit distrib in \( \mathbb{Z}/n \)

Because even holding a fixed, 
so \( ax \) is fixed, \( b \) is still random.
So \( ax + b \) is unit distrib.

If \( n \) is prime, then for \( x \neq y \),
\[
h(x) \quad \text{unit distrib} \\
h(x) - h(y) = ax - ay = a \cdot (x-y) \quad \text{unit distrib}
\]
Hash function \( \rightarrow \) 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 one element of \( S \).

   - \( \text{insert}(x) \): compute \( h(x) \) and find first unoccupied bucket starting from \( h(x) \). Insert \( x \) in first empty bucket.
   - \( \text{query}(x) \): start at \( h(x) \) and search forward until:
     - Find \( x \), answer "present"
     - Find empty bucket, answer "absent".

Both methods support \( O(1) \) 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 \( O(\log m) \) in expectation.

Disadvantage: randomized, running time guarantee only in expectation.

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

Approximate membership using Bloom filters

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

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

- \( k \) = parameter related to failure probability
- E.g. \( k = 6 \) or \( 7 \) usually a good choice.

\( \therefore \) about 8-10 bits per element, in practice.
Hash functions \( h_1, \ldots, h_k : [N] \rightarrow [h] \).

(Same \( k \) as before; \( k=6 \) or 7 typically)

\( h_1,\ldots, h_k \) indep and samples from \( \mathcal{H} \).

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

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,

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

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

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

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

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

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

\[ \approx \frac{n}{2} . \]

Assume these events all indep as \( k, n \) vary.

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