Hashing

Central concept in CS
Numerous applications:
  • Dictionary data structures, load balancing, placement, ...

Setting:
A large set of (possible) values: called universe U
Interested in only a subset of this: S
Let $|S| = N$ (typically $N \ll |U|$)

Roughly, hashing is a way to map elements of U onto smaller number of values such that with high probability there are not too many collisions among elements of S.
Hashing

Concrete running application for this module: dictionary.

Setting:
- A large universe of keys (e.g., set of all strings of certain length): denoted by $U$
- The actual dictionary $S$ (subset of $U$)

Operations:
- $\text{add}(x)$: add a key $x$
- $\text{query}(q)$: is key $q$ there?
- $\text{delete}(x)$: remove the key $x$
Hashing

“...**with high probability** there are not too many collisions among elements of S”

On what is this probability calculated over?

Two approaches:

1. Input is random
2. Input is arbitrary, but the hash function is random

Input being random is typically not valid for many applications. So we will use 2.

- We will assume a family of hash functions \( H \).
- When it is time to hash S, we choose a random function \( h \in H \)
Hashing: Desired properties

Let \([M] = \{0, 1, \ldots, M-1\}\)
We design a hash function \(h: U \rightarrow [M]\)
\((M = \text{table size})\)

What properties would we want?

1. Small probability of distinct keys colliding:
   1. If \(x \neq y \in S\), \(P[h(x) = h(y)]\) is “small”
2. Small range, i.e., small \(M\) so that the hash table is small
3. Small number of bits to store \(h\)
4. \(h\) is easy to compute
Ideal Hash Function

Perfectly random hash function:
For each $x \in S$, $h(x) = a$ uniformly random location in $[M]$

Properties:
- Low collision probability: $P[h(x) = h(y)] = 1/M$ for any $x \neq y$
- Even conditioned on hashed values for any other subset $A$ of $S$, for any element $x \in S$, $h(x)$ is still uniformly random over $[M]$

Q: Problem with this ideal approach?
1. Too large to store this hash function: $\log M$ bits needed for each element in $S$ (since it can hash to any of the $M$ locations)
2. Also unclear how to compute $h(.)$ fast other than a table lookup
Use board from here onwards
Universal Hash functions

Captures the basic desired property of non-collision of two distinct elements.
Due to Carter and Wegman (1979)

**Definition:** A family $H$ of hash functions mapping $U$ to $[M]$ is universal if for any $x \neq y \in U$,

$$P[h(x) = h(y)] \leq 1/M$$

Note: Must hold for every pair of distinct $x$ and $y \in U$. 
Universal Hash functions

A simple construction of universal hashing:

Assume $|U| = 2^u$ and $|M| = 2^m$

Let $A$ be a $m \times u$ matrix with random binary entries. For any $x \in U$, view it as a $u$-bit binary vector, and define

$$h(x) := Ax$$

where the arithmetic is modulo $2$.

Q: How many hash functions in this family?

$2^{um}$
Universal Hash functions

A simple construction of universal hashing:
Let $A$ be a $m \times u$ matrix with uniformly random binary entries.

$$h(x) := Ax$$

where the arithmetic is modulo 2.

**Theorem.** The family of hash functions defined above is universal.

**Proof.** Ideas?

$$h(x) = h(y) \iff Ax = Ay$$

for $x \neq y$

$$Ax = Ay$$

$$A(x-y) = 0$$

$$\implies A2z = 0 \text{ for } z \neq 0$$
Want to show $P(A_2 = 0) \leq \frac{1}{M}$ for any $z \neq 0$

Let $z_{i^*} \neq 0$ (i.e. since $z \neq 0$)

$A_2 = 0 \implies \sum A_j z_j = 0$

Columns of $A$

$A_{i^*} = -\sum_{j \neq i^*} A_j z_j$

Fixed vector

$M$ length random binary vector

Prob of above $= \left(\frac{1}{2}\right)^M = \frac{1}{M}$
Application: Hash table

One of the main applications of hash functions is in hash tables (for dictionary data structures)

Handling collisions:
Closed addressing
   Each location maintains some other data structure
One approach: “separate chaining”
   Each location in the table stores a linked list with all the elements mapped to that location.
Look up time = length of the linked list

To understand lookup time, we need to study the number of collisions.
Application: Hash table

Let us study the number of collisions:
Let $C_x$ be the number of other elements mapped to the value where $x$ is mapped to.
Let $L_x$ be the length of the linked list containing $x$
$\quad L_x = C_x + 1$

Q: What is $E[L_x]$?
$E[L_x] = 1 + E[C_x] = 1 + (N-1)/M$

Hence if we use $M$ (table size) $\geq N$ (size of $|S|$), $E[L_x] \leq 2$!
lookups take **constant time in expectation**.

Item deletion is also easy.
Let $C = \text{total number of collisions}$

Q: What is $E[C]$?

$\leq \binom{N}{2} 1/M$
Application: Hash table

Can we design a collision free hash table?

Suppose we choose $M \geq N^2$

Q: $P[\text{there exists a collision}] = \frac{1}{2}$

⇒ Can easily find a collision free hash table!
⇒ Constant lookup time for all elements! (worst-case guarantee)

But this is large a space requirement.
(Space measured in terms of number of keys)

Can we do better? $O(N)$? (while providing worst-case guarantee?)