Data structure: Map/Dictionary

- Hashing
- Hash Functions

- data structure for map/dictionary
  - maintain a set of (key, value) pairs
  - given a key, want to find its value quickly
  - operations
    1. insert (key, value) add (key, value) pair to the dictionary
    2. find (key) return value - for this key/report key does not exist.

- goal: 1. fast insert and find (O(1) time)
  2. save space

  - for simplicity: key is a number in \(\{0, 1, 2, \ldots, m-1\}\) \(m=2^{32}\)
  - if we only want to achieve 1
    - can use a large array \(a[0 \ldots m-1]\)
      - insert: \(a[\text{key}] = \text{value}\)
      - find: \(\text{return } a[\text{key}]\)
  - if we only want to achieve 2
    - can store (key, value) pairs in a list.

- Hashing

  key: 0 \ldots 2^{32}-1

  array: \[
    \begin{array}{cccc}
      & & & \\
      & & & \\
      & & & \\
      0 & 1 & \ldots & n-1 \\
    \end{array}
  \]

  idea: map multiple keys to the same location.

  - hash function: \(\{0, \ldots, m-1\} \rightarrow \{0, \ldots, n-1\}\)
  - insert: \(a[h(\text{key})] = \text{value}\)
  - find: \(\text{return } a[h(\text{key})]\)

  \[
    \begin{array}{cccccc}
      \text{Key}: & 1 & 2 & 3 & 4 & 5 & 6 \\
      h: & X & \_ & \_ & \_ & \_ & \_ \\
    \end{array}
  \]
- Collisions: if $x \neq y$, but $h(x) = h(y)$, we say $x, y$ is a collision.

- Working with collisions

  - Instead of holding only 1 value at each location, use a linked list (array list) to store all key value pairs $(1, a)$ $(3, b)$ $(5, c)$

- Insert $(key, value)$: insert $(key, value)$ into the list at $h(key)$

- Find $(key)$: find the key in the list at $h(key)$

- Question: how to choose a hash function?

- Goal 1: minimize # of collisions.

  Observation: Cannot use a deterministic hash function for any hash function, by pigeonhole principle

  - Many values map to same location.
  - Very slow if all these key values appear.

  Solution: use a randomized hash function.

  $h_1, h_2, \ldots, h_t$ ($t$ can be very large)

  Randomly choose $i \in [1, 2, \ldots, t]$, use $h_i$ as hash function.

  Want: for any $x \neq y$, $Pr[h(x) = h(y)] = \frac{1}{n}$ (1)

  Probability for any two key values to have a collision is $\frac{1}{n}$.

  Probability is over the choice of hash function.
Lemma: if \( h(x) \) maps all key values to independent random locations then \( h \) satisfies (1).

Proof: if \( x \neq y \), \( h(x) \) random in \( 0 \ldots n-1 \)
\( h(y) \) independent, random in \( 0 \ldots n-1 \)
\[
\Pr[h(x) = h(y)] = \frac{1}{n}
\]
- Claim: expected running time of find/insert operation is \( O\left(\frac{k}{n} + 1\right) \)
where \( k \) is the number of \((\text{key}, \text{value})\) pairs in the dictionary.

Proof: For find \((\text{key})\), running time = length of the list at \( h(\text{key}) \)
Suppose the pairs in the dictionary is \((\text{key}_1, v_1) \ldots (\text{key}_k, v_k)\)
\[
\mathbb{E}[\text{length of the list}] = \sum_{i=1}^{k} \Pr[h(\text{key}) = h(\text{key}_i)]
\leq \frac{k}{n} + 1
\]

- Problem: if we select a uniformly random hash function, can not represent/store the function efficiently.

- total # of functions from \([0 \ldots m-1]\) to \([0 \ldots n-1]\)
\(n^m\), cannot sample/store such functions.

- need: function that can be represented succinctly.

- recall: modular arithmetic
  
  can do +, -, \( x \mod p \)

  \[ p = 5 \quad 3 + 4 = 2 \pmod{5} \quad 3 \times 4 = 2 \pmod{5} \]

  \[ 3-4 = 4 \pmod{5} \]
  
  if \( p \) is a prime

  for any \( x \mod p \neq 0 \), there is a unique \( y \in \{0, \ldots, p-1\} \)
  
  s.t. \( x \times y \mod p = 1 \) \( (y = x^{-1}) \)

  example: \( x = 3 \quad p = 5 \quad \rightarrow y = 2 \)

  \[ 3 \times 2 \mod 5 = 1 \]

  There is an algorithm (Extended Euclid) that can compute \( y \) given \( x, p \).
- Design a hash function: \( P \) is prime, \( n = P \)

  Key: represent the key in \( P \)-ary representation

  \[
  \text{Key} = \sum_{i=1}^{t} x_i \cdot P^{i-1} \quad x_i \in \{0, 1, \ldots, P-1\}
  \]

  hash function: randomly choose \( a_1, a_2, \ldots, a_t \in \{0, 1, \ldots, P-1\} \)

  \[
  h_{a_1, a_2, \ldots, a_t}(\text{key}) = \left( \sum_{i=1}^{t} a_i \cdot x_i \right) \mod P
  \]

  \[
  P = 11 \quad t = 2 \quad 0 \leq \text{key} < 11^2 = 121
  \]

  \[
  a_1 = 7 \quad a_2 = 4
  \]

  \[
  h_{a_1, a_2}(37)
  \]

  \[
  37 = 4 + 3 \times 11 \Rightarrow x_1 = 4 \quad x_2 = 3
  \]

  \[
  h_{a_1, a_2}(37) = (a_1 x_1 + a_2 x_2) \mod 11 = (7 \times 4 + 4 \times 3) \mod 11 = 7
  \]

  \[
  - \text{Claim: For this hash function, for any } x \neq y, \quad Pr[h(x) = h(y)] = \frac{1}{P}
  \]

  Proof: \( x \neq y \Rightarrow \) there is one location where \( x_q \neq y_q \)

  first fix \( a_1, a_2, \ldots, a_q, \ldots, a_t \)

  \[
  h(x) = \sum_{i=1}^{t} a_i \cdot x_i
  \]

  \[
  h(y) = \sum_{i=1}^{t} a_i \cdot y_i
  \]

  \[
  h(x) = h(y) \Rightarrow a_q x_q + \sum_{i \neq q} a_i \cdot x_i = a_q y_q + \sum_{i \neq q} a_i \cdot y_i \mod P
  \]

  \[
  a_q (x_q - y_q) = \sum_{i \neq q} (a_i \cdot y_i - a_i \cdot x_i) \mod P
  \]

  because \( x_q \neq y_q \)

  \[
  a_q = (x_q - y_q)^{-1} \sum_{i \neq q} (a_i \cdot y_i - a_i \cdot x_i) \mod P
  \]

  \[
  Pr[h(x) = h(y)] = Pr[a_q = b \mod P] = \frac{1}{b}
  \]