

Hashing

- Dictionaries
- Chained Hashing
- Universal Hashing
- Static Dictionaries and Perfect Hashing

Hashing

- Dictionaries
- Chained Hashing
- Universal Hashing
- Static Dictionaries and Perfect Hashing

Dictionaries

- **Dictionary problem.** Maintain a dynamic set of integers $S \subseteq U$ subject to following operations
 - LOOKUP(x): return true if $x \in S$ and false otherwise.
 - INSERT(x): set $S = S \cup \{x\}$
 - DELETE(x): set $S = S \setminus \{x\}$
- **Universe size.** Typically $|U| = 2^{64}$ or $|U| = 2^{32}$ and $|S| \ll |U|$.
- **Satellite information.** Information associated with each integer.
- **Goal.** A compact data structure with fast operations.

Dictionaries

- Applications.
 - Many!
 - Key component in other data structures and algorithms.

Dictionaries

- Which solutions do we know?

Hashing

- Dictionaries
- **Chained Hashing**
- Universal Hashing
- Static Dictionaries and Perfect Hashing

Chained Hashing

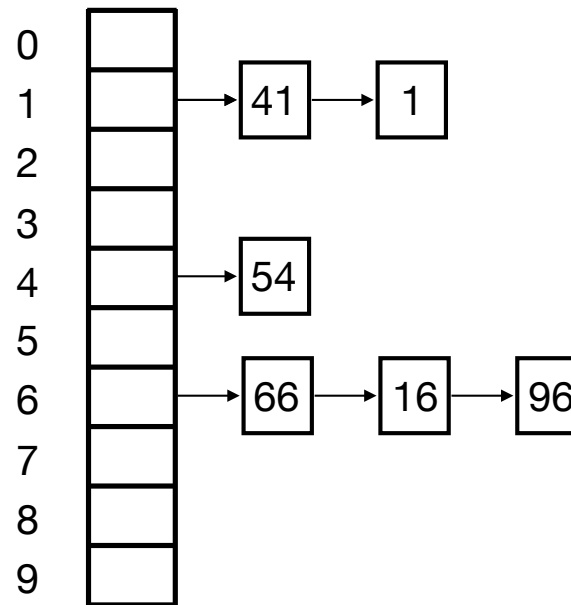
- [Chained hashing \[Dumey 1956\]](#).
 - [Hash function](#). Pick some crazy, chaotic, random function h that maps U to $\{0, \dots, m-1\}$, where $m = \Theta(n)$.
 - Initialize an array $A[0, \dots, m-1]$.
 - $A[i]$ stores a linked list containing the keys in S whose hash value is i .

Chained Hashing

$U = \{0, \dots, 99\}$

$S = \{1, 16, 41, 54, 66, 96\}$

$h(x) = x \bmod 10$



- **Operations.**

- **LOOKUP(x):** Compute $h(x)$. Scan through list for $h(x)$. Return true if x is in list and false otherwise.
- **INSERT(x):** Compute $h(x)$. Scan through list for $h(x)$. If x is in list do nothing. Otherwise, add x to the front of list.
- **DELETE(x):** Compute $h(x)$. Scan through list for $h(x)$. If x is in list remove it. Otherwise, do nothing.

- **Time.** $O(1 + \text{length of linked list for } h(x))$

Chained Hashing

- Hash functions.
 - $h(x) = x \bmod 10$ is not very crazy, chaotic, or random.
 - For any **fixed** choice of h , there is a set whose elements all map to the same slot.
 - \Rightarrow We end up with a single linked list.
 - How can we overcome this?
- Use randomness.
 - Assume the input set is random.
 - Choose the hash function at random.

Chained Hashing

- **Random hash functions.** Assume that:
 1. h is chosen uniformly at random among all functions from U to $\{0, \dots, m-1\}$
 2. We can store h in $O(n)$ space.
 3. We can evaluate h in $O(1)$ time
- What is the expected length of the linked lists?

Chained Hashing

$$E(\text{length of linked list for } h(x)) = E(|\{y \in S \mid h(y) = h(x)\}|)$$

$$= E\left(\sum_{y \in S} \begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases}\right)$$

$$= \sum_{y \in S} E\left(\begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases}\right)$$

$$= \sum_{y \in S} \Pr(h(x) = h(y))$$

$$= 1 + \sum_{y \in S \setminus \{x\}} \Pr(h(x) = h(y))$$

$$= 1 + \sum_{y \in S \setminus \{x\}} \frac{1}{m}$$

m^2 choices for pair $(h(x), h(y))$,
 m of which cause collision

$$= 1 + (n - 1) \cdot \frac{1}{m} = O(1)$$

Chained Hashing

- **Theorem.** We can solve the dictionary problem (under assumptions 1+2+3) in
 - $O(n)$ space.
 - $O(1)$ expected time per operation.
- Expectation is over the choice of hash function.
- Independent of the input set.

Chained Hashing

- **Random hash functions assumptions.**
 1. h is chosen uniformly at random among all functions from U to $\{0, \dots, m-1\}$
 2. We can store h in $O(n)$ space.
 3. We can evaluate h in $O(1)$ time
- **Random hash functions.** Can we efficiently compute and store a random function?
 - We **need** $\Theta(u \log m)$ bits to store an arbitrary function $h: \{0, \dots, u-1\} \rightarrow \{0, \dots, m-1\}$
 - We **need** a lot of random bits to generate the function.
 - We **need** a lot of time to generate the function.
- Do we **need** a truly random hash function?
- When did we use the fact that h was random in our analysis?

Chained Hashing

$$\begin{aligned} E(\text{length of linked list for } h(x)) &= E(|\{y \in S \mid h(y) = h(x)\}|) \\ &= E\left(\sum_{y \in S} \begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases}\right) \\ &= \sum_{y \in S} E\left(\begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases}\right) \\ &= \sum_{y \in S} \Pr(h(x) = h(y)) \\ &= 1 + \sum_{y \in S \setminus \{x\}} \Pr(h(x) = h(y)) \\ &= 1 + \sum_{y \in S \setminus \{x\}} \frac{1}{m} \quad \text{For all } x \neq y, \Pr(h(x) = h(y)) \leq 1/m \\ &= 1 + (n - 1) \cdot \frac{1}{m} = O(1) \end{aligned}$$

Hashing

- Dictionaries
- Chained Hashing
- **Universal Hashing**
- Static Dictionaries and Perfect Hashing

Universal Hashing

- Universal hashing [Carter and Wegman 1979].
 - Let H be a family of functions mapping U to $\{0, \dots, m-1\}$.
 - H is **universal** if for any $x \neq y$ in U and h chosen uniformly at random in H ,

$$\Pr(h(x) = h(y)) \leq 1/m$$

Universal Hashing

- **Positional number systems.** For integers x and m , the **base- m representation** of x is x written in base m .
- **Example.**
 - $(10)_{10} = (1010)_2 \quad (1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)$
 - $(107)_{10} = (212)_7 \quad (2 \cdot 7^2 + 1 \cdot 7^1 + 2 \cdot 7^0)$

Universal Hashing

- **Hash function.** Given a prime m and $a = (a_1 a_2 \dots a_r)_m$, define

$$h_a((x_1 x_2 \dots x_r)_m) = a_1 x_1 + a_2 x_2 + \dots + a_r x_r \pmod{m}$$

- **Example.**

- $m = 7$
- $a = (107)_{10} = (212)_7$
- $x = (214)_{10} = (424)_7$
- $h_a(x) = 2 \cdot 4 + 1 \cdot 2 + 2 \cdot 4 \pmod{7} = 18 \pmod{7} = 4$

- **Universal family.**

- $H = \{h_a \mid (a_1 a_2 \dots a_r)_m \in \{0, \dots, m-1\}^r\}$
- Choose random hash function from $H \sim$ choose random a .
- H is universal (analysis next).
- $O(1)$ time evaluation.
- $O(1)$ space.
- Fast construction.

Universal Hashing

- **Lemma.** Let m be a prime. For any $a \in \{1, \dots, m-1\}$ there exists a unique inverse a^{-1} such that $a^{-1} \cdot a \equiv 1 \pmod{m}$. (\mathbb{Z}_m is a field)
- **Example.** $m = 7$

a	1	2	3	4	5	6
a^{-1}						

a	1	2	3	4	5	6
a^{-1}	1	4	5	2	3	6

Universal Hashing

- **Goal.** For random $a = (a_1 a_2 \dots a_r)_m$, show that if $x = (x_1 x_2 \dots x_r)_m \neq y = (y_1 y_2 \dots y_r)_m$ then $\Pr[h_a(x) = h_a(y)] \leq 1/m$
- $(x_1 x_2 \dots x_r)_m \neq (y_1 y_2 \dots y_r)_m \implies x_i \neq y_i$ for some i . Assume wlog. that $x_r \neq y_r$.

$$\begin{aligned} & \Pr(h_a((x_1 \dots x_r)_m) = h_a((y_1 \dots y_r)_m)) \\ &= \Pr(a_1 x_1 + \dots + a_r x_r \equiv a_1 y_1 + \dots + a_r y_r \pmod{m}) \\ &= \Pr(a_r x_r - a_r y_r \equiv a_1 y_1 - a_1 x_1 + \dots + a_{r-1} y_{r-1} - a_{r-1} x_{r-1} \pmod{m}) \quad \text{existence of inverses} \\ &= \Pr(a_r(x_r - y_r) \equiv a_1(y_1 - x_1) + \dots + a_{r-1}(y_{r-1} - x_{r-1}) \pmod{m}) \\ &= \Pr(a_r(x_r - y_r)(x_r - y_r)^{-1} \equiv (a_1(y_1 - x_1) + \dots + a_{r-1}(y_{r-1} - x_{r-1}))(x_r - y_r)^{-1} \pmod{m}) \\ &= \Pr(a_r \equiv (a_1(y_1 - x_1) + \dots + a_{r-1}(y_{r-1} - x_{r-1}))(x_r - y_r)^{-1} \pmod{m}) = \frac{1}{m} \end{aligned}$$

for any choice of a_1, a_2, \dots, a_{r-1} , the RH defines a unique a_r that matches (uniqueness of inverses).
Of the m^r choices for a_1, a_2, \dots, a_r exactly m^{r-1} cause a collision \implies probability is $m^{r-1}/m^r = 1/m$

Universal Hashing

- **Lemma.** H is universal with $O(1)$ time evaluation and $O(1)$ space.
- **Theorem.** We can solve the dictionary problem (without special assumptions) in:
 - $O(n)$ space.
 - $O(1)$ expected time per operation (lookup, insert, delete).

Universal Hashing

- Other universal families.

- For prime $p > 0$, $a \in \{1, \dots, p-1\}$, $b \in \{0, \dots, p-1\}$

$$h_{a,b}(x) = ax + b \bmod m$$

$$H = \{h_{a,b} \mid a \in \{1, \dots, m-1\}, b \in \{0, \dots, m-1\}\}$$

- Hash function from k -bit numbers to l -bit numbers. a is an odd k -bit integer.

l most significant bits of the k least significant bits of ax

$$h_a(x) = (ax \bmod 2^k) \gg (k - l)$$


$$H = \{h_a \mid a \text{ is an odd integer in } \{1, \dots, 2^k - 1\}\}$$

Hashing

- Dictionaries
- Chained Hashing
- Universal Hashing
- **Static Dictionaries and Perfect Hashing**

Static Dictionaries and Perfect Hashing

- **Static dictionary problem.** Given a set $S \subseteq U = \{0, \dots, u-1\}$ of size n for preprocessing support the following operation
 - `lookup(x)`: return true if $x \in S$ and false otherwise.
- As the dictionary problem with no updates (insert and deletes).
- Set given in advance.

Static Dictionaries and Perfect Hashing

- **Dynamic solution.** Use chained hashing with a universal hash function as before \implies solution with $O(n)$ space and $O(1)$ expected time per lookup.
- Can we do better?
- **Perfect Hashing.** A **perfect hash function** for S is a **collision-free** hash function on S .
 - Perfect hash function in $O(n)$ space and $O(1)$ evaluation time \implies solution with $O(n)$ space and $O(1)$ **worst-case lookup time**.
 - Do perfect hash functions with $O(n)$ space and $O(1)$ evaluation time exist for any set S ?

Static Dictionaries and Perfect Hashing

- **Goal.** Perfect hashing in linear space and constant worst-case time.
- **Solution in 3 steps.**
 - **Solution 1.** Collision-free but with too much space.
 - **Solution 2.** Many collisions but linear space.
 - **Solution 3: FKS scheme [Fredman, Komlós, Szemerédi 1984].** Two-level solution. Combines solution 1 and 2.
 - At level 1 use solution with lots of collisions and linear space.
 - Resolve collisions at level 1 with collision-free solution at level 2.
 - lookup(x): look-up in level 1 to find the correct level 2 dictionary. Lookup in level 2 dictionary.

Static Dictionaries and Perfect Hashing

- **Solution 1.** Collision-free but with too much space.
- Use a universal hash function to map into an array of size n^2 . What is the expected total number of collisions in the array?

$$\begin{aligned} E(\#collisions) &= E \left(\sum_{x,y \in S, x \neq y} \begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases} \right) \\ &= \sum_{x,y \in S, x \neq y} E \left(\begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases} \right) \\ &= \sum_{x,y \in S, x \neq y} \Pr(h(x) = h(y)) = \binom{n}{2} \frac{1}{n^2} \leq \frac{n^2}{2} \cdot \frac{1}{n^2} = 1/2 \end{aligned}$$

#distinct pairs universal hashing into n^2 range

- With probability $1/2$ we get perfect hashing function. If not perfect try again.
- \implies Expected number of trials before we get a perfect hash function is $O(1)$.
- \implies For a static set S we can support lookups in $O(1)$ worst-case time using $O(n^2)$ space.

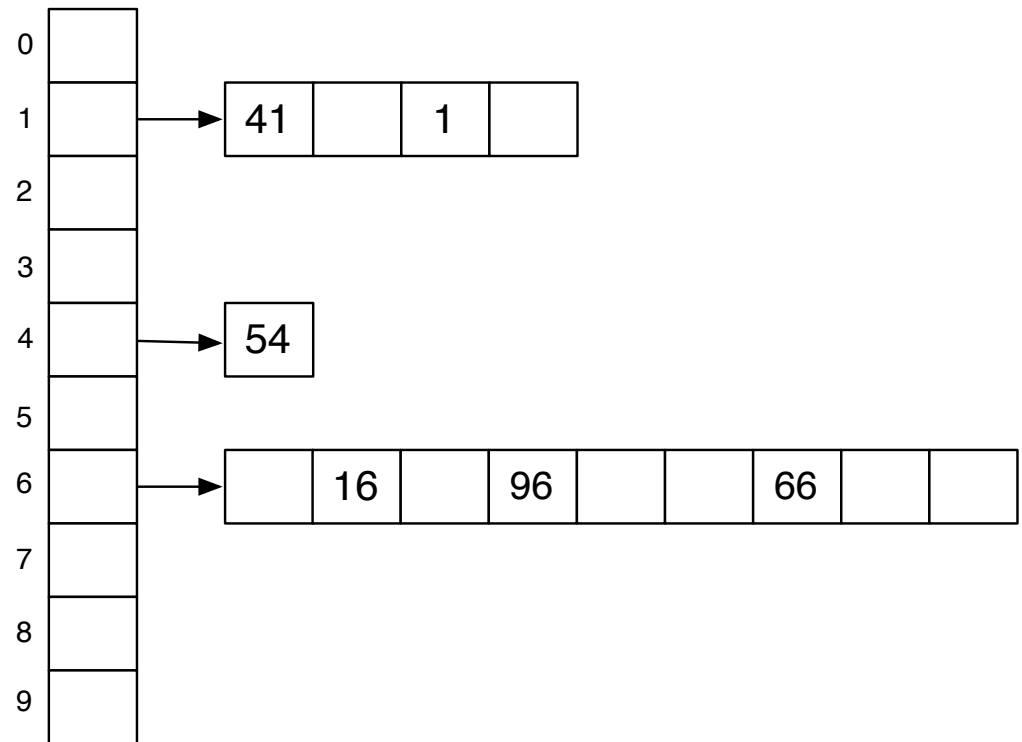
Static Dictionaries and Perfect Hashing

- **Solution 2.** Many collisions but linear space.
- As solution 1 but with array of size n . What is the expected total number of collisions in the array?

$$\begin{aligned} E(\#\text{collisions}) &= E \left(\sum_{x,y \in S, x \neq y} \begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases} \right) \\ &= \sum_{x,y \in S, x \neq y} E \left(\begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases} \right) \\ &= \sum_{x,y \in S, x \neq y} \Pr(h(x) = h(y)) = \binom{n}{2} \frac{1}{n} \leq \frac{n^2}{2} \cdot \frac{1}{n} = \frac{1}{2}n \end{aligned}$$

Static Dictionaries and Perfect Hashing

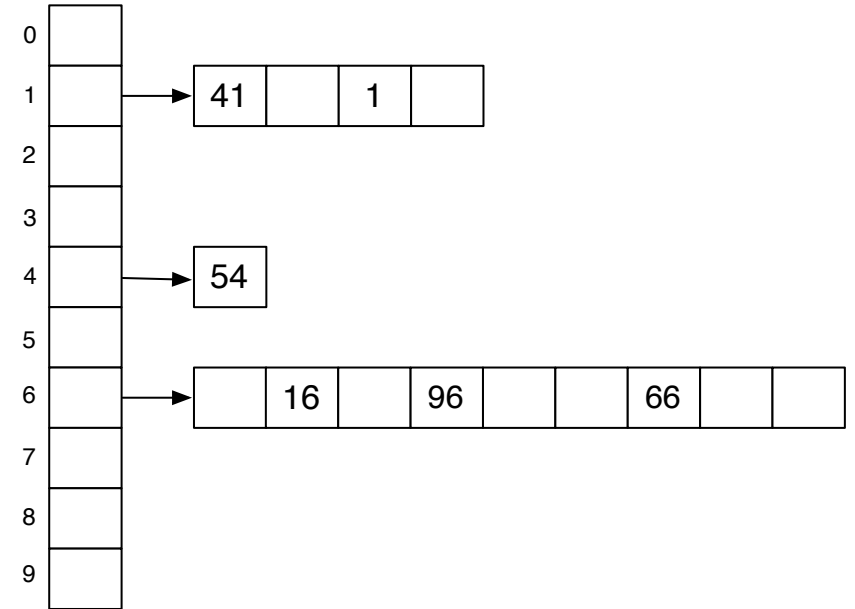
- **Solution 3.** Two-level solution.
 - At level 1 use solution with lots of collisions and linear space.
 - Resolve each collisions at level 1 with collision-free solution at level 2.
 - lookup(x): look-up in level 1 to find the correct level 2 dictionary. Lookup in level 2 dictionary.
- **Example.**
 - $S = \{1, 16, 41, 54, 66, 96\}$
 - Level 1 collision sets:
 - $S_1 = \{1, 41\}$,
 - $S_4 = \{54\}$,
 - $S_6 = \{16, 66, 96\}$
 - Level 2 hash info stored with subtable.
 - (size of table, multiplier a, prime p)
- **Time.** $O(1)$
- **Space?**



Static Dictionaries and Perfect Hashing

- **Space.** What is the the total size of level 1 and level 2 hash tables?

$$\#collisions = \sum \binom{|S_i|}{2} = O(n)$$



$a^2 = a + 2\binom{a}{2}$, for any integer a

$$\begin{aligned} \text{space} &= O\left(n + \sum_i |S_i|^2\right) = O\left(n + \sum_i \left(|S_i| + 2\binom{|S_i|}{2}\right)\right) \\ &= O\left(n + \sum_i |S_i| + 2\sum_i \binom{|S_i|}{2}\right) = O(n + n + 2n) = O(n) \end{aligned}$$

Static Dictionaries and Perfect Hashing

- **FKS scheme.**
 - $O(n)$ space and $O(n)$ expected preprocessing time.
 - Lookups with two evaluations of a universal hash function.
- **Theorem.** We can solve the static dictionary problem for a set S of size n in:
 - $O(n)$ space and $O(n)$ expected preprocessing time.
 - $O(1)$ worst-case time per lookup.
- **Multilevel data structures.**
 - FKS is example of **multilevel** data structure technique. Combine different solutions for same problem to get an improved solution.

Hashing

- Dictionaries
- Chained Hashing
- Universal Hashing
- Static Dictionaries and Perfect Hashing