

Hashing

- Dictionaries
- Chained Hashing
- Linear Probing
- Hash Functions

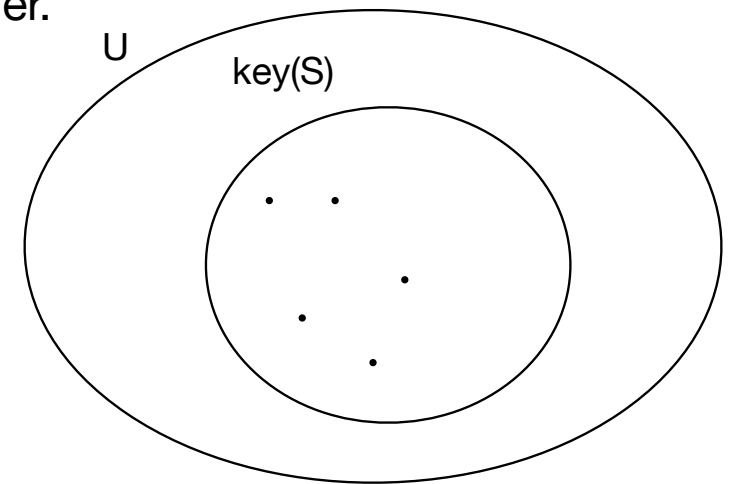
Philip Bille

Hashing

- Dictionaries
- Chained Hashing
- Linear Probing
- Hash Functions

Dictionaries

- **Dictionaries.** Maintain dynamic set $S \subseteq U$ of n integers supporting the following operations.
 - **SEARCH(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 $n \ll |U|$.
- **Satellite information.** Information associated with each integer.
- **Goal.** A compact data structure with fast operations.



Dictionaries

- Applications.
 - Basic data structures for representing a set.
 - Used in numerous algorithms and data structures.
- Which solutions do we know?

Dictionaries

Data structure	SEARCH	INSERT	DELETE	space
linked list	$O(n)$	$O(1)$	$O(1)$	$O(n)$
BBST	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$
direct addressing	$O(1)$	$O(1)$	$O(1)$	$O(U)$

- **Challenge.** Can we do significantly better?

Hashing

- Dictionaries
- Chained Hashing
- Linear Probing
- Hash Functions

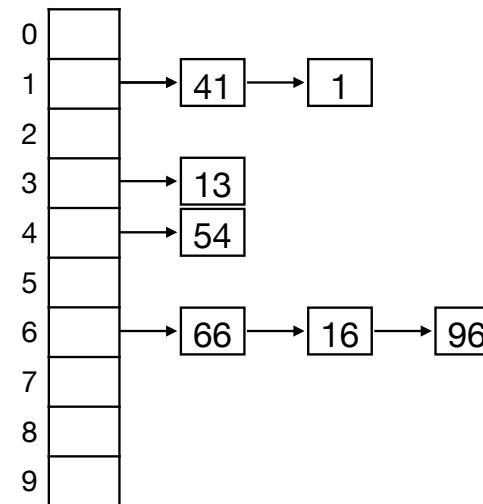
Chained Hashing

- **Idea.** Pick a crazy, chaotic, random **hash function** $h : U \rightarrow \{0, \dots, m-1\}$, where $m = \Theta(n)$. Hash function should distribute S **approximately evenly** over $\{0, \dots, m-1\}$.
- **Chained hashing.**
 - Maintain array $A[0..m-1]$ of linked lists.
 - Store element x in linked list at $A[h(x)]$.
- **Collision.**
 - x and y **collides** if $h(x) = h(y)$.

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

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

$h(x) = x \bmod 10$



Chained Hashing

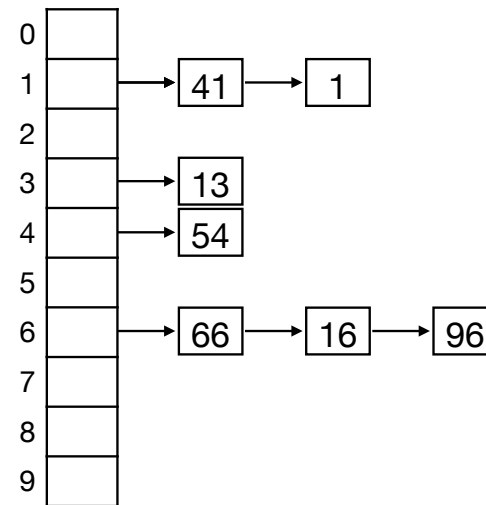
- **Operations.**

- SEARCH(x): compute $h(x)$. Scan $A[h(x)]$. Return true if x is in list and false otherwise.
- INSERT(x): compute $h(x)$. Scan $A[h(x)]$. Add x to the front of list if it is not there already.
- DELETE(x): compute $h(x)$. Scan $A[h(x)]$. If x is in list remove it. Otherwise, do nothing.

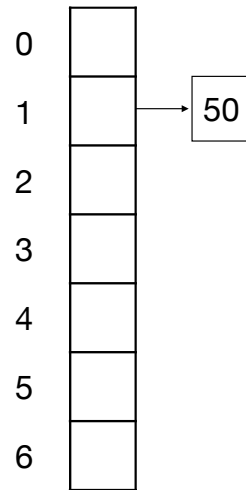
$U = \{0, \dots, 99\}$

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

$h(x) = x \bmod 10$

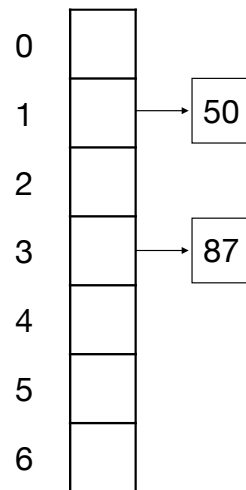


$k = 50$



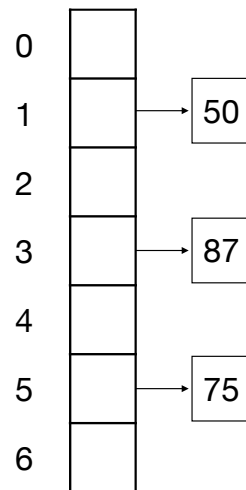
$$h(k) = k \bmod 7$$

$k = 87$



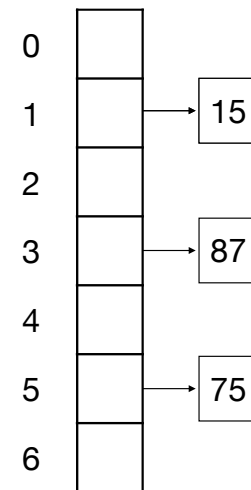
$$h(k) = k \bmod 7$$

$k = 75$



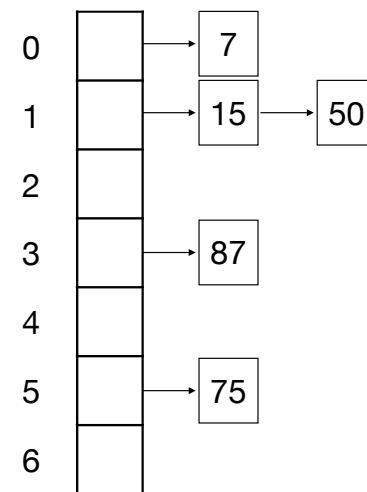
$$h(k) = k \bmod 7$$

$k = 15$



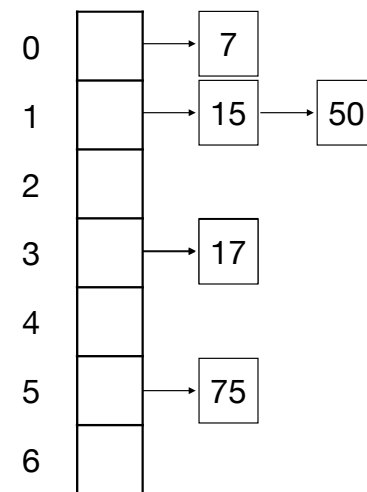
$$h(k) = k \bmod 7$$

$k = 7$



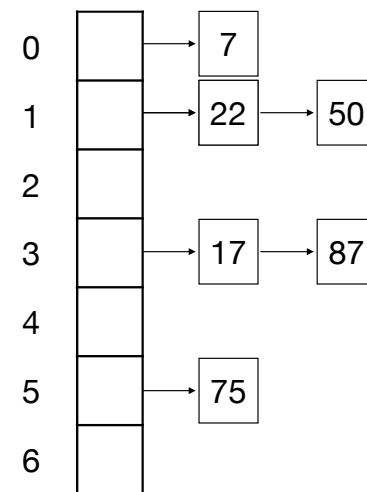
$$h(k) = k \bmod 7$$

$k = 17$



$$h(k) = k \bmod 7$$

$$k = 22$$

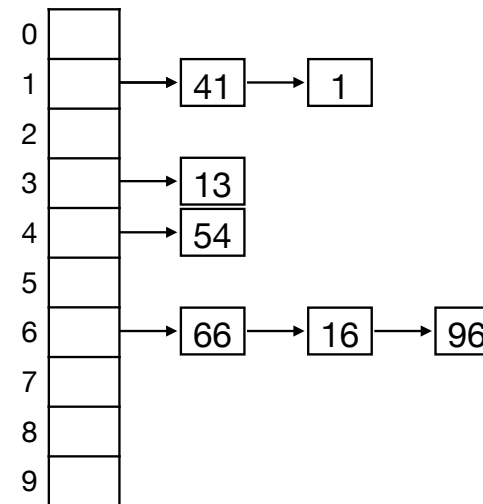


$$h(k) = k \bmod 7$$

Chained Hashing

- **Time.**
 - SEARCH, INSERT, and DELETE in $O(|A[h(x)]| + 1)$ time.
 - Length of list depends on hash function.
- **Space.**
 - $O(m + n) = O(n)$.

$U = \{0, \dots, 99\}$
 $S = \{1, 16, 41, 54, 66, 96\}$
 $h(x) = x \bmod 10$



Dictionaries

Data structure	SEARCH	INSERT	DELETE	space
linked list	$O(n)$	$O(1)$	$O(1)$	$O(n)$
BBST	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$
direct addressing	$O(1)$	$O(1)$	$O(1)$	$O(U)$
chained hashing	$O(A[h(x)] + 1)$	$O(A[h(x)] + 1)$	$O(A[h(x)] + 1)$	$O(n)$

Hashing

- Dictionaries
- Chained Hashing
- **Linear Probing**
- Hash Functions

Linear Probing

- Linear probing.
 - Maintain S in array A of size $m = \Theta(n)$.
 - Element x stored in $A[h(x)]$ or in **cluster** to the right of $A[h(x)]$.
 - Cluster = consecutive (cyclic) sequence of non-empty entries.

	41	1	11	13	54			98	
0	1	2	3	4	5	6	7	8	9

$h(k) = k \bmod 10$

- SEARCH(x): linear search for h(x) from $A[h(x)]$ in cluster.
- INSERT(x): if $A[h(x)]$ empty, put in x at $A[h(x)]$. Otherwise, put x into next empty entry to the right of $A[h(x)]$ (cyclically).
- DELETE(x): SEARCH for x and remove it. Re-insert **all** elements to the right of it in the cluster.

5 1 27 32 54 11 19

0	1	2	3	4	5	6	7	8	9	10

$$h(k) = k \bmod 11$$

Linear Probing

- Time.

- Let $C(i)$ be size of cluster at position i .
- SEARCH, INSERT, and DELETE in $O(C(h(x)) + 1)$ time.
- Size of cluster depends on hash function.
- Highly **cache-efficient**.

- Space.

- $O(m) = O(n)$.

- Variants.

- Special case of **open addressing**.
- Quadratic probing
- Double hashing.

	41	1	11	13	54			98	
0	1	2	3	4	5	6	7	8	9

$$h(k) = k \bmod 10$$

Dictionaries

Data structure	SEARCH	INSERT	DELETE	space
linked list	$O(n)$	$O(1)$	$O(1)$	$O(n)$
BBST	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$
direct addressing	$O(1)$	$O(1)$	$O(1)$	$O(U)$
chained hashing	$O(A[h(x)] + 1)$	$O(A[h(x)] + 1)$	$O(A[h(x)] + 1)$	$O(n)$
linear probing	$O(C(h(x)) + 1)$	$O(C(h(x)) + 1)$	$O(C(h(x)) + 1)$	$O(n)$

Hashing

- Dictionaries
- Chained Hashing
- Linear Probing
- Hash Functions

Hash Functions

- Hash functions.
 - $h(x) = x \bmod 11$ is not very crazy, chaotic, or random.
 - For any **deterministic** 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?
- Random input.
 - Assume input set S is random.
 - Expectation on input. Efficient on "average" input set S .
- Random hash function.
 - Choose the hash function at random.
 - Expectation on random (private) choices. **Independent** of input set S .

Simple Uniform Hashing

- Simple uniform hashing.
 - Choose h uniformly at random among all functions from U to $\{0, \dots, m-1\}$.
 - $\Rightarrow$ For every $u \in U$, $h(u)$ is chosen independently and uniformly at random from $\{0, \dots, m-1\}$.
- Lemma. For any $x, y \in U$, $x \neq y$, $\Pr[h(x) = h(y)] = 1/m$.
- Proof.
 - Let $x, y \in U$, $x \neq y$, and consider the pair $(h(x), h(y))$.
 - m^2 possible choices for pair and m of these cause a collision.
 - $\Rightarrow \Pr[h(x) = h(y)] = m/m^2 = 1/m$.

Simple Uniform Hashing

- Chained hashing with simple uniform hashing.

- What is the expected length of $A[h(x)]$?

- Let $I_y = \begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases}$

- $$E(|A[h(x)]|) = E\left(\sum_{y \in S} I_y\right) = \sum_{y \in S} E(I_y) = 1 + \sum_{y \in S \setminus \{x\}} \Pr(h(x) = h(y)) = 1 + (n-1) \cdot \frac{1}{m} = O(1)$$

- $\Rightarrow$ With simple uniform random hashing we can solve the dictionary problem in $O(n)$ space and $O(1)$ expected time per operation.

Simple Uniform Hashing

- **Uniform random hash functions.** Can we efficiently compute and store a random function?
 - We **need** $\Theta(u)$ space 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?

Simple Uniform Hashing

- Chained hashing with simple uniform hashing.

- What is the expected length of $A[h(x)]$?

- Let $I_y = \begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases}$

For any $x, y \in U, x \neq y, \Pr[h(x) = h(y)] = 1/m.$

- $E(|A[h(x)]|) = E\left(\sum_{y \in S} I_y\right) = \sum_{y \in S} E(I_y) = 1 + \sum_{y \in S \setminus \{x\}} \Pr(h(x) = h(y)) \downarrow = 1 + (n - 1) \cdot \frac{1}{m} = O(1)$

- $\Rightarrow$ With simple uniform random hashing we can solve the dictionary problem in $O(n)$ space and $O(1)$ expected time per operation.

Universal Hashing

- Universal hashing.

- Let H be a family of functions mapping a universe U to $\{0, \dots, m-1\}$.
- H is **universal** if for any $x, y \in U$, $x \neq y$, and h chosen uniformly at random from H ,
- that any $h \in H$ can be stored compactly and we can compute $h(x)$ efficiently.

$$\Pr(h(x) = h(y)) \leq \frac{1}{m}$$

- Require that any $h \in H$ can be stored compactly and we can compute $h(x)$ efficiently.
- Universal hashing + chained hashing $\Rightarrow$ dictionary in $O(n)$ space and $O(1)$ expected time operations.

Universal Hashing

- Dot product hashing.

- Assume $f |U| = m^2$ and m is prime.
- Represent $x \in U$ as pair $x = (x_1, x_2)$ where $x_1, x_2 \in \{0, \dots, m-1\}$. x is two-digit number in base m .
- Given $a = (a_1, a_2)$ define

$$h_a(x) = (a_1 \cdot x_1 + a_2 \cdot x_2) \bmod m$$

- Example.

- $m = 7, U = \{0, \dots, 49\}$
- $a = 17 = (2, 3), x = 22 = (3, 1)$
- $h_a(x) = 2 \cdot 3 + 3 \cdot 1 \bmod 7 = 9 \bmod 7 = 2$

- Family of hash functions.

- Family of hash functions: $H = \{h_a \mid a \in U\}$
- Pick random hash function $\sim$ pick random a .
- Constant time computation and constant space.
- Is H universal?

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

- **Lemma.** $H = \{h_a \mid a \in U\}$, where $h_a(x) = (a_1 \cdot x_1 + a_2 \cdot x_2) \bmod m$ is universal.

- **Proof.**

- Goal: For random $a = (a_1, a_2)$, show $\Pr(h_a(x) = h_a(y)) \leq 1/m$.

- Let $x = (x_1, x_2)$ and $y = (y_1, y_2)$, with $x \neq y$. Assume wlog that $x_2 \neq y_2$. Then,

$$h_a(x) = h_a(y) \iff a_1 x_1 + a_2 x_2 \equiv a_1 y_1 + a_2 y_2 \pmod{m} \iff a_2 \equiv \frac{a_1(y_1 - x_1)}{x_2 - y_2} \pmod{m}$$

↑
existence of inverses.

- Assume we pick a_1 randomly first $\Rightarrow$ RH is a fixed value $\neq 0$.

- Uniqueness of inverses $\Rightarrow$ exactly one value a_2 such that LH = RH.

- $\Rightarrow \Pr(h_a(x) = h_a(y)) \leq 1/m$.

Universal Hashing

- Dot product hashing for larger universes.

- What if $|U| > m^2$?
- Represent $x \in U$ as vector $x = (x_1, x_2, \dots, x_r)$ where $x_i \in \{0, \dots, m-1\}$. x is number in base m .
- For $a = (a_1, a_2, \dots, a_r)$, define

$$h_a(x = (x_1, x_2, \dots, x_r)) = a_1x_1 + a_2x_2 + \dots + a_rx_r \bmod m$$

- **Lemma.** $H = \{h_a \mid a \in U\}$ is universal.

Universal Hashing

- **Theorem.** We can solve the dictionary problem in
 - $O(n)$ space.
 - $O(1)$ expected time per operation.
- Expectation on random choice of hash function.
- Independent on input set S .

Universal Hashing

- Other universal families.

- For prime $p > 0$.

$$h_{a,b}(x) = ax + b \mod p$$

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

- Hash function from k -bit numbers to l -bit numbers.

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

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

Dictionaries

Data structure	SEARCH	INSERT	DELETE	space
linked list	$O(n)$	$O(1)$	$O(1)$	$O(n)$
BBST	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$
direct addressing	$O(1)$	$O(1)$	$O(1)$	$O(U)$
chained hashing + universal hashing	$O(1)^\dagger$	$O(1)^\dagger$	$O(1)^\dagger$	$O(n)$

$\dagger$ = **expected** time

Hashing

- Dictionaries
- Chained Hashing
- Linear Probing
- Hash Functions