Hashing

- Dictionaries
- Chained Hashing
- Linear Probing
- Hash Functions

Philip Bille

Hashing

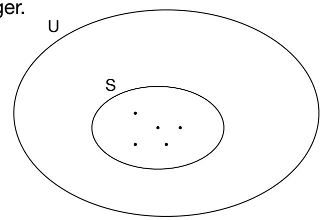
- Dictionaries
- Chained Hashing
- Linear Probing
- Hash Functions

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(n)	O(n)	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

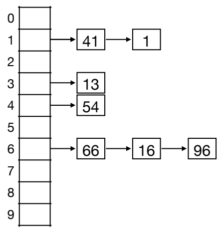
Chained Hashing

- Idea. Pick a crazy, chaotic, random hash function h : U → {0, ... , m-1}, where m = Θ(n). Hash function should distribute S approximately evenly over {0, ..., 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).

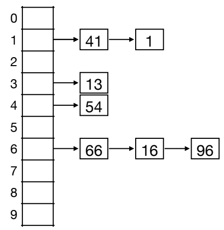
U = {0, ..., 99}
S = {1, 16, 41, 54, 66, 96}
h(x) = x mod 10



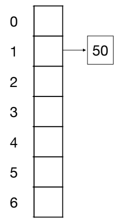
Chained Hashing

- 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, ..., 99}
S = {1, 16, 41, 54, 66, 96}
h(x) = x mod 10

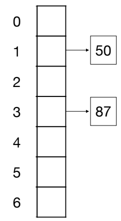


k = 50



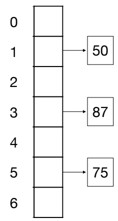
h(k) = k mod 7

k = 87



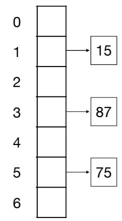
h(k) = k mod 7

k = 75



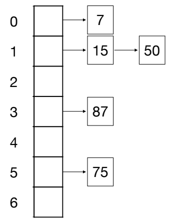
h(k) = k mod 7

k = 15



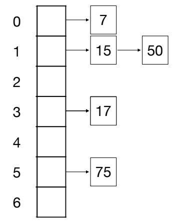
h(k) = k mod 7

k = 7



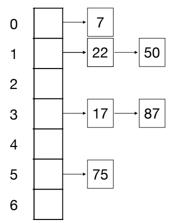
h(k) = k mod 7

k = 17



h(k) = k mod 7

k = 22



h(k) = k mod 7

Chained Hashing

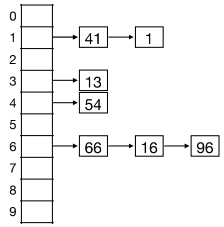
- SEARCH(x): compute h(x). Scan A[h(x)]. Return true if x is in list and false otherwise.
- INSERT(x): compute h(x). Scan A[h(x)]. Add x to the front of list if it is not there already.
- DELETE(x): compute h(x). Scan A[h(x)]. If x is in list remove it. Otherwise, do nothing.

- Exercise. Insert sequence of keys K = 5, 28, 19, 15, 20, 33, 12, 17, 10 in an initially empty hash table of size 9 using chained hashing with hash function h(k) = k mod 9.

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, ..., 99}
S = {1, 16, 41, 54, 66, 96}
h(x) = x mod 10



Dictionaries

Data structure	Search	Insert	Delete	space
linked list	O(n)	O(n)	O(n)	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 = Θ(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 mod 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 mod 11

Linear Probing

- Time.
  - Let C(i) be size of cluster at position i.
  - SEARCH and INSERT in O(C(h(x)) + 1) time.
  - DELETE in O(n^2) 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 mod 10

Dictionaries

Data structure	SEARCH	INSERT	DELETE	space
linked list	O(n)	O(n)	O(n)	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(n^2)	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.
  - ⇒ Chained hashing becomes a single linked list.
  - How can we overcome this?
- Random input.
  - Assume input set S is random.
  - Expectation on input. Good for random input set S, very bad for worst-case 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,..,m-1}.
  - ⇒ For every $u \in U$, $h(u)$ is chosen independently and uniformly at random from {0, …, m-1}.
- Lemma. Let h be a simple uniform hash function. 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.
  - ⇒ $\Pr[h(x) = h(y)] = m/m^2 = 1/m$.

Simple Uniform Hashing

- Chained hashing with simple uniform hashing.
  - What is the expected length of $A[h(x)]$?
- Let $I_y = \begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases}$
- $E\left(|A[h(x)]|\right) = E\left(\sum_{y \in S} I_y\right) = \sum_{y \in S} E\left(I_y\right) = 1 + \sum_{y \in S \setminus \{x\}} \Pr\left(h(x) = h(y)\right) = 1 + (n-1) \cdot \frac{1}{m} = O(1)$
- ⇒ 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 ∈ U, x ≠ y, Pr[h(x) = h(x)] = 1/m.

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

- ⇒ 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, ..., m-1}.
  - H is universal if for any x,y ∈ U, x ≠ y, and h chosen uniformly at random from H,

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

  - Require that any h ∈ H can be stored compactly and we can compute h(x) efficiently.

- Chained hashing with universal hash function.
  - Chained hashing with a universal hash function
  - ⇒ we can solve the dictionary problem in O(n) space and O(1) expected time operations.

Universal Hashing

- Positional number systems. For integers x and m, the base-m representation of x is x written in base m.

- Example.
  - $(10)_{10} = (1010)_2 \ (1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)$
  - $(107)_{10} = (212)_7 \ (2 \cdot 7^2 + 1 \cdot 7^1 + 2 \cdot 7^0)$

Universal Hashing

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

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

- Example.
  - m = 7, U = {0,...,49}
  - $a = (17)_{10} = (23)_7$, $x = (22)_{10} = (31)_7$
  - $h_a(x) = 2 \cdot 3 + 3 \cdot 1 \bmod 7 = 9 \bmod 7 = 2$

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

Universal Hashing

- Lemma. Let m be a prime and let $z \neq 0 \mod m$. Then $\alpha \cdot z = \beta \mod m$ has exactly one solution for $\alpha \in \{0,\ldots, m-1\}$.
- Lemma. $H = \{h_a \mid a \in U\}$, where $h_a(x) = (a_1 \cdot x_1 + a_2 \cdot x_2) \mod m$ is universal.
- Proof.
  - Goal: For random $a = (a_1a_2)_m$, show that $\Pr(h_a(x) = h_a(y)) \leq 1/m$.
  - Let $x = (x_1x_2)_m$ and $y = (y_1y_2)_m$, with $x \neq y$. Assume wlog that $x_2 \neq y_2$. Then,

$$\begin{aligned} h_a(x) = h_a(y) &\iff a_1x_1 + a_2x_2 = a_1y_1 + a_2y_2 \mod m \\ &\iff a_2\underbrace{(x_2 - y_2)}_{z} = \underbrace{a_1(y_1 - x_1)}_{\beta} \mod m \end{aligned}$$

  - Assume we pick a_1 randomly first ⇒ RH is a fixed value.
  - m is prime and $x_2 \neq y_2$ ⇒ $a_2 \cdot z = \beta \mod m$ has exactly one solution among m possible.
  - ⇒ $\Pr(h_a(x) = h_a(y)) = 1/m$.

Universal Hashing

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

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

- 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 $p > 0$.

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

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

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

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

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

Dictionaries

Data structure	SEARCH	INSERT	DELETE	space
linked list	O(n)	O(n)	O(n)	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)†	O(1)†	O(1)†	O(n)

† = expected time

Hashing

- Dictionaries
- Chained Hashing
- Linear Probing
- Hash Functions