Hashing

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

Philip Bille

Hashing

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

Dictionaries

- Dictionary problem. Maintain a set $S \subseteq U = \{0, ..., u-1\}$ supporting
  - lookup(x): return true if $x \in S$ and false otherwise.
  - insert(x): set $S = S \cup \{x\}$
  - delete(x): set $S = S - \{x\}$

- Think universe size $u = 2^{64}$ or 2^{32} and $|S| \ll u$.

- Satellite information. We may also have associated satellite information for each key.

- Goal. A compact data structure (linear space) with fast operations (constant time).

Dictionaries

- Applications.
  - Maintain a dictionary (!)
  - Key component in many data structures and algorithms. (Examples in exercises and later lectures).

Dictionaries

- Which solutions do we know?
  - Direct addressing (bitvector)
  - Linked lists.
  - Binary search trees (balanced)
  - Chained hashing

Hashing

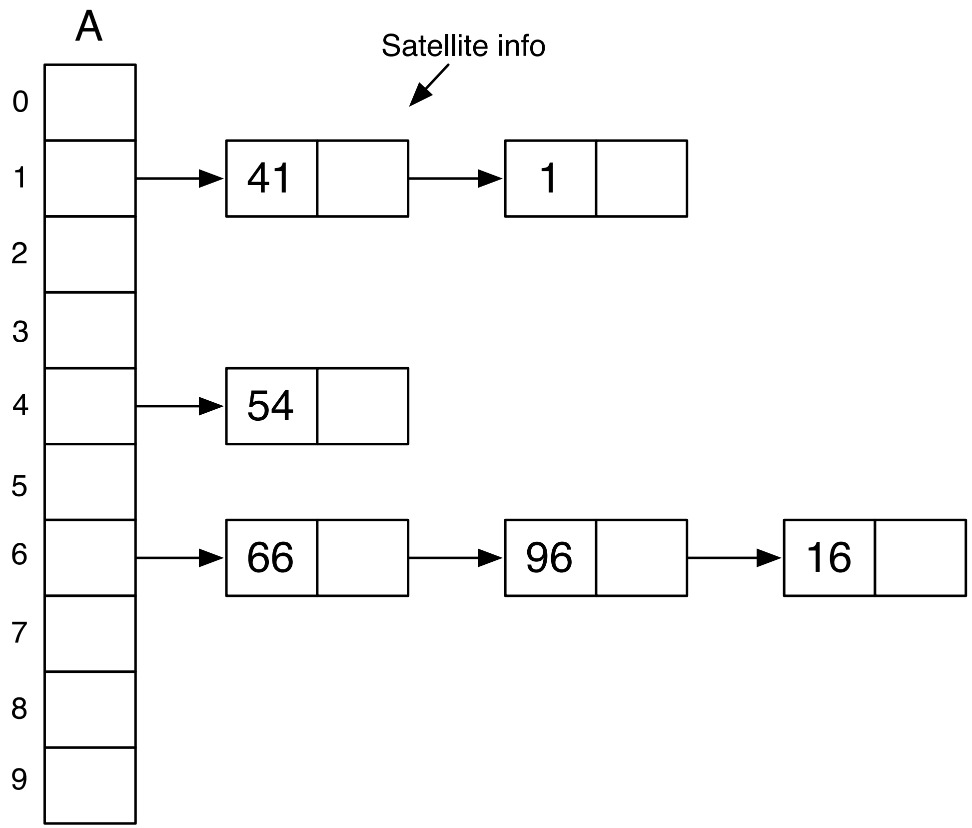
- Dictionaries
- Chained Hashing
- Universal Hashing
- Static Dictionaries and Perfect Hashing

Chained Hashing

- Simplifying assumption. $|S| \leq N$ at all times and we can use space $O(N)$.
- Chained hashing [Dumey 1956].
  - Pick some crazy, chaotic, random function h (the hash function) mapping U to {0, ..., N-1}.
  - Initialize an array A[0, ..., N-1].
  - A[i] stores a linked list containing the keys in S whose hash value is i.

Chained Hashing

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



Chained Hashing

- Operations. How can we support lookup, insert, and delete?
  - 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 + length of linked list for h(x))

Chained Hashing

- Hash functions. A crazy, chaotic hash function (like $h(x) = x \bmod 10$) sounds good, but there is a big problem.
  - For any fixed choice of h, we can find 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

- Chained hashing for random hash functions.
  - Assumption 1. h: U → {0, ..., N-1} is chosen uniformly at random from the set of all functions from U to {0, ..., N-1}.
  - Assumption 2. h can be evaluated in constant time.
- What is the expected time for an operation OP(x), where OP = {lookup, insert, delete}?

Chained Hashing

$$\begin{aligned}
\text{Time for OP}(x) &= O\left(1 + E\left[\text{length of linked list for } h(x)\right]\right) \\
&= O\left(1 + E\left[\left|\{y \in S \mid h(y) = h(x)\}\right|\right]\right) \\
&= O\left(1 + E\left[\sum_{y \in S} \begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases}\right]\right) \\
&= O\left(1 + \sum_{y \in S} E\left[\begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases}\right]\right) \\
&= O(1 + \sum_{y \in S} \Pr[h(x) = h(y)]) \\
&= O(1 + 1 + \sum_{y \in S \setminus \{x\}} \Pr[h(x) = h(y)]) \\
&= O(1 + 1 + \sum_{y \in S \setminus \{x\}} 1/N) \\
&= O(1 + 1 + N(1/N)) = O(1)
\end{aligned}$$

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

Chained Hashing

- Theorem. With a random hash function (under assumptions 1 + 2) we can solve the dictionary problem in
  - O(N) space.
  - O(1) expected time per operation (lookup, insert, delete).

- Expectation is over the choice of hash function.
- Independent of the input set.

Random Hash Functions

- Random hash functions. Can we efficiently compute and store a random function?
  - We need u log N bits to store an arbitrary function from {0,..., u-1} to {0,..., N-1} (specify for each element x in U the value h(x)).
  - We need a lot of random bits to generate the function.
  - We need a lot of time to generate the function.

Random Hash Functions

- Do we need a truly random hash function?
- When did we use the fact that h was random in our analysis?

$$
\begin{aligned}
\text{Time for } \mathrm{OP}(x) &= O\left(1 + E\left[\text{length of linked list for } h(x)\right]\right) \\
&= O\left(1 + E\left[\left|\{y \in S \mid h(y) = h(x)\}\right|\right]\right) \\
&= O\left(1 + E\left[\sum_{y \in S} \begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases}\right]\right) \\
&= O\left(1 + \sum_{y \in S} E\left[\begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases}\right]\right) \\
&= O(1 + \sum_{y \in S} \Pr[h(x) = h(y)]) \\
&= O(1 + 1 + \sum_{y \in S \setminus \{x\}} \Pr[h(x) = h(y)]) \\
&= O(1 + 1 + \sum_{y \in S \setminus \{x\}} 1/N) \\
&= O(1 + 1 + N(1/N)) = O(1)
\end{aligned}
$$

For all $x \neq y, \ \Pr[h(x) = h(y)] \leq 1/N$

Random Hash Functions

- We do not need a truly random hash function!
- We only need: For all $x \neq y$, $\Pr[h(x) = h(y)] \leq 1/N$
- Captured in definition of universal hashing.

Hashing

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

Universal Hashing

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

$$\Pr[h(x) = h(y)] \leq 1/N$$

- Universal hashing and chaining.
  - If we can find family of universal hash functions such that
    - we can store it in small space
    - we can evaluate it in constant time
  - ⇒ efficient chained hashing without special assumptions.

Universal Hashing

- Positional number systems. For integers x and p, the base-p representation of x is x written in base p.
- 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 $N < p < 2N$ and $a = (a_1a_2\ldots a_r)_p$, define

$$h_a(x = (x_1x_2\ldots x_r)_p) = a_1x_1 + a_2x_2 + \ldots + a_rx_r \bmod p$$

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

- Universal family.
  - $H = \{h_a \mid a = (a_1a_2\ldots a_r)_p \in \{0, \ldots, p\text{-}1\}^r\}$
  - Choose random hash function from H ~ choose random a.
  - H is universal (next slides).
  - O(1) time evaluation.
  - O(1) space.
  - Fast construction (find suitable prime).

Universal Hashing

- Lemma. Let p be a prime. For any $a \in \{1, ..., p-1\}$ there exists a unique inverse a^{-1} such that $a^{-1} \cdot a \equiv 1 \mod p$. ($Z_p$ is a field)
- Example. $p = 7$

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 $a = (a_1a_2\ldots a_r)_p$, show that if $x = (x_1x_2\ldots x_r)_p \neq y = (y_1y_2\ldots y_r)_p$ then $\Pr[h_a(x) = h_a(y)] \leq 1/N$
- $(x_1x_2\ldots x_r)_p \neq y = (y_1y_2\ldots y_r)_p \Longrightarrow x_i \neq y_i$ for some i. Assume wlog. that $x_r \neq y_r$.

$$
\begin{aligned}
&\Pr[h_a((x_1\ldots x_r)_p) = h_a((y_1\ldots,y_r)_p)]\\
&= \Pr\left[a_1x_1 + \cdots + a_rx_r \equiv a_1y_1 + \cdots + a_ry_r \bmod p\right]\\
&= \Pr\left[a_rx_r - a_ry_r \equiv a_1y_1 - a_1x_1 + \cdots + a_{r-1}y_{r-1} - a_{r-1}x_{r-1} \bmod p\right]\\
&= \Pr\left[a_r(x_r - y_r) \equiv a_1(y_1 - x_1) + \cdots + a_{r-1}(y_{r-1} - x_{r-1}) \bmod p\right]\\
&= \Pr\left[a_r(x_r - y_r)(x_r - y_r)^{-1} \equiv (a_1(y_1 - x_1) + \cdots + a_{r-1}(y_{r-1} - x_{r-1}))(x_r - y_r)^{-1} \bmod p\right]\\
&= \Pr\left[a_r \equiv (a_1(y_1 - x_1) + \cdots + a_{r-1}(y_{r-1} - x_{r-1}))(x_r - y_r)^{-1} \bmod p\right] = \frac{1}{p} \leq \frac{1}{N}
\end{aligned}
$$

existence of inverses

p choices for a_r, exactly one causes a collision by uniqueness of inverses.

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).

Other Universal Families

- For prime p > 0, a ∈ {1, .., p-1}, b ∈ {0, ..., p-1}

$$h_{a,b}(x) = (ax + b \bmod p) \bmod N$$
$$H = \{h_{a,b} \mid a \in \{1, \ldots, p-1\}, b \in \{0, \ldots, p-1\}\}$$

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

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

$$h_a(x) = (ax \bmod 2^k) \gg (k - l)$$
$$H = \{h_a \mid a \text{ is an odd integer in } \{1, \ldots, 2^k - 1\}\}$$

Hashing

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

Static Dictionaries and Perfect Hashing

- Static dictionary problem. Given a set $S \subseteq U = \{0,..,u-1\}$ 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 $\Longrightarrow$ 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 $\Longrightarrow$ solution with O(N) space and O(1) worst-case lookup time. (Why?)
  - 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?

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

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

$$= \sum_{x,y \in S, x \neq y} \Pr[h(x) = h(y)] = \binom{N}{2} \frac{1}{N^2} \leq \frac{N^2}{2} \cdot \frac{1}{N^2} = 1/2$$

#distinct pairs

Universal hashing into N^2 range

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

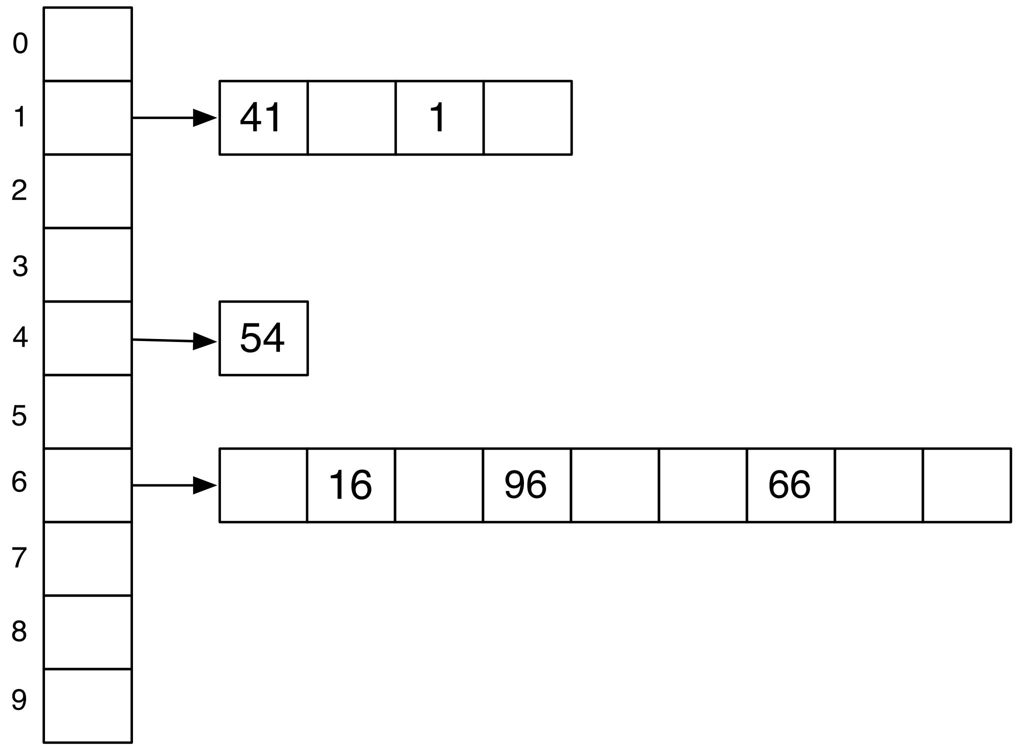
Static Dictionaries and Perfect Hashing

- 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?

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

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?



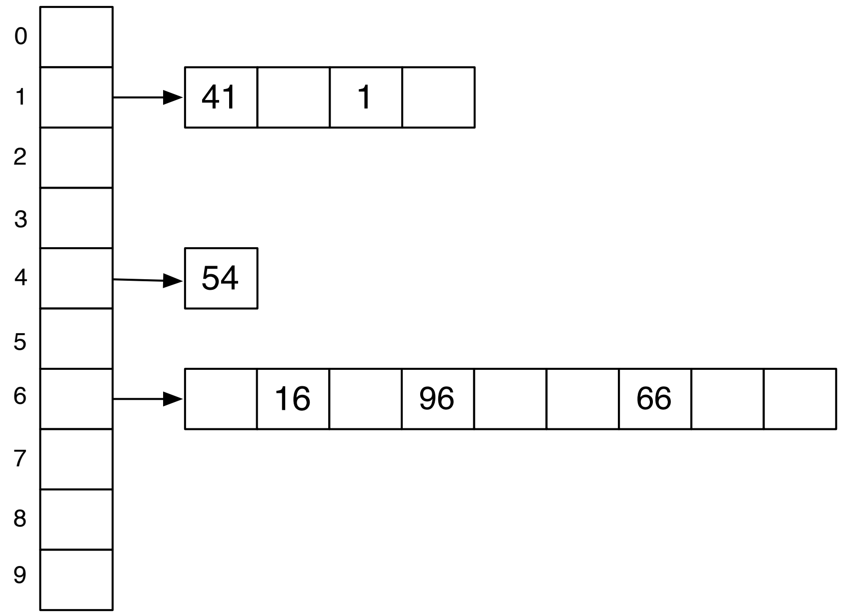
Static Dictionaries and Perfect Hashing

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

$$\text{space} = O\left(N + \sum_{i \in \{0,\ldots,N-1\}} |S_i|^2\right)$$

$$\#\text{collisions} = O(N)$$

$$\#\text{collisions} = \sum_{i \in \{0,\ldots,N-1\}} \binom{|S_i|}{2}$$



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

$$\text{space} = O\left(N + \sum_i |S_i|^2\right) = O\left(N + \sum_i \left(|S_i| + 2\binom{|S_i|}{2}\right)\right)$$

$$= O\left(N + \sum_i |S_i| + 2\sum_i \binom{|S_i|}{2}\right) = O(N + N + 2N) = O(N)$$

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