Randomized Algorithms II

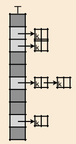
Inge Li Gørtz

Thank you to Kevin Wayne and Philip Bille for inspiration to slides

Randomized algorithms

- Last weeks
  - Contention resolution
  - Global minimum cut
  - Expectation of random variables
    - Guessing cards
  - Quicksort
  - Selection

- Today
  - Hash functions and hash tables



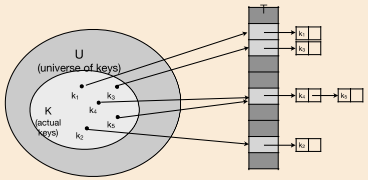
Hashing

Dictionaries

- Dictionary problem. Maintain a dynamic set of $S \subseteq U$ subject to the 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 $|U| = 2^{64}$ and $|S| << |U|$.
- Satellite information. Information associated with each element.
- Goal. A compact data structure with fast operations.
- Applications. Many! A key component in other data structures and algorithms.

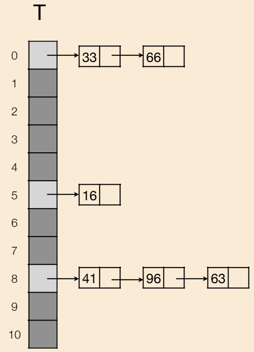
Chained Hashing

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



Chained Hashing

- S = {16, 33, 41, 63, 66, 96}
- U = {0,…,99}
- h(x) = x mod 11.



Uniform random hash functions

- *E.g. h(x) = x mod 11*. Not crazy, chaotic, random.
- Suppose |U| ≥ n²: For any hash function h there will be a set S of n elements that all map to the same position!
  => we end up with a single linked list.
- Solution: randomization.
  - For every element u ∈ U: select h(u) uniformly at random in {0, …, m-1} independently from all other choices.

- Claim. *The probability that h(u) = h(v) for two elements u ≠ v is 1/m.*
- Proof.
  - m² possible choices for the pair of values (h(u),h(v)). All equally likely.
  - Exactly m of these gives a collision.

Chained Hashing with Random Hash Function

- Expected length of the linked list for h(x)?
- Random variable L_x = length of linked list for x. $L_x = |\{y \in S \mid h(y) = h(x)\}|$
- Indikator random variable:

$$I_y = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{otherwise} \end{cases} \qquad L_x = \sum_{y \in S} I_y \qquad E[I_y] = \Pr[h(y) = h(x)] = \frac{1}{m} \text{ for } x \neq y.$$

- The expected length of the linked list for x:

$$E[L_x] = E\left[\sum_{y \in S} I_y\right] = \sum_{y \in S} E[I_y] = 1 + \sum_{y \in S \setminus \{x\}} \frac{1}{m} = 1 + (n-1) \cdot \frac{1}{m} = \Theta(1).$$

Chained Hashing with Random Hash Function

- Constant time and O(n) space for the hash table.
- But:
  - Need O(|U|) space for the hash function.
  - Need a lot of random bits to generate the hash function.
  - Need a lot of time to generate the hash function.
- Do we need a truly random hash function?
- When did we use the fact that h was random in our analysis?

Chained Hashing with Random Hash Function

- Expected length of the linked list for h(x)?
- Random variable L_x = length of linked list for x. $L_x = |\{y \in S \mid h(y) = h(x)\}|$
- Indikator random variable:

$$I_y = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{otherwise} \end{cases} \qquad L_x = \sum_{y \in S} I_y \qquad E[I_y] = \boxed{\Pr[h(y) = h(x)] = \frac{1}{m} \text{ for } x \neq y.}$$

- The expected length of the linked list for x:

$$E[L_x] = E\left[\sum_{y \in S} I_y\right] = \sum_{y \in S} E[I_y] = 1 + \sum_{y \in S \setminus \{x\}} \frac{1}{m} = 1 + (n-1) \cdot \frac{1}{m} = \Theta(1).$$

Universal hash functions

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

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

- Require that any $h \in H$ can be represented compactly and that we can compute the value $h(u)$ efficiently for any $u \in U$.

Universal Hashing

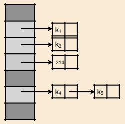
- Positional number systems. For integers x and b, the base-b representation of x is x written in base b.
- Example.
  - $(10)_{10} = (1010)_2$ $\quad (1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)$
  - $(107)_{10} = (212)_7$ $\quad (2 \cdot 7^2 + 1 \cdot 7^1 + 2 \cdot 7^0)$

Universal Hashing

- Hash function. Given a prime p and a = $(a_1a_2\ldots a_r)_p$, define

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

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

Uniform Hashing

- Lemma 1. For any prime p, any integer $z \neq 0 \mod p$, and any two integers α, β:

$$\alpha z = \beta z \mod p \quad \Rightarrow \quad \alpha = \beta \mod p.$$

- Proof.
  - Show $(\alpha - \beta)$ is divisible by p:
    - $\alpha z = \beta z \mod p \quad \Rightarrow \quad (\alpha - \beta)z = 0 \mod p.$
    - By assumption z not divisible by p.
    - Since p is prime $\alpha - \beta$ must be divisible by p.
  - Thus $\alpha = \beta \mod p$ as claimed.

Universal Hashing

- Goal. For random $a = (a_1a_2\ldots a_r)_p$, show that if $x \neq y$ then $\Pr[h_a(x) = h_a(y)] \leq 1/p$.
  - Recall: $x = (x_1x_2\ldots x_r)_p$ and $y = (y_1y_2\ldots y_r)_p$:

$$x \neq y \Leftrightarrow (x_1x_2\ldots x_r)_p \neq (y_1y_2\ldots y_r)_p \Rightarrow x_j \neq y_j \text{ for some } j.$$

- Lemma 2. Let j be such that $x_j \neq y_j$. Assume the coordinates a_i have been chosen for all $i \neq j$. The probability of choosing a_j such that $h_a(x) = h_a(y)$ is $1/p$.
  - $h_a(x) = h_a(y) \quad \Leftrightarrow \quad \sum_{i=1}^{r} a_ix_i \mod p = \sum_{i=1}^{r} a_iy_i \mod p \quad \Leftrightarrow \quad a_j\boxed{(x_j - y_j)} = \boxed{\sum_{i\neq j} a_i(x_i - y_i)} \mod p$

fixed value z ≠ 0

fixed value since all a_i fixed for i≠j. = c

  - There is exactly one value $0 \leq a_j < p$ that satisfies $a_j z = c \bmod p$.
    - Assume there was two such values a_j and a_j'.
      - Then $a_j z = a_j' z \mod p$.
      - Lemma 1 $\Rightarrow a_j = a_j' \mod p$. Since $a_j < p$ and $a_j' < p$ we have $a_j = a_j'$.
  - Probability of choosing a_j such that $h_a(x) = h_a(y)$ is $1/p$.

Universal Hashing

- Lemma 2. Let j be such that $x_j \neq y_j$. Assume the coordinates a_i have been chosen for all $i \neq j$. The probability of choosing a_j such that $h_a(x) = h_a(y)$ is $1/p$.

- Theorem. For random $a = (a_1a_2\ldots a_r)_p$, if $x \neq y$ then

$$\Pr[h_a(x) = h_a(y)] = 1/p\,.$$

- Proof.
  - E : the event that $h_a(x) = h_a(y)$.
  - F_b : the event that the values a_i for $i \neq j$ gets the sequence of values b.
  - Lemma 2 shows that $\Pr[E \mid F_b] = 1/p$ for all b.
  - Thus

$$\Pr[E] = \sum_b \Pr[E \mid F_b] \cdot \Pr[F_b] = \sum_b \frac{1}{p} \cdot \Pr[F_b] = \; = \frac{1}{p} \sum_b \cdot \Pr[F_b] = \frac{1}{p}$$

Chained Hashing with Random Hash Function

- Expected length of the linked list for h(x)?
- Random variable L_x = length of linked list for x. $L_x = |\{y \in S \mid h(y) = h(x)\}|$
- Indikator random variable:

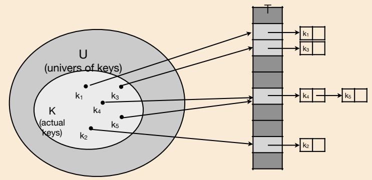
$$I_y = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{otherwise} \end{cases} \qquad L_x = \sum_{y \in S} I_y \qquad E[I_y] = \boxed{\Pr[h(y) = h(x)] = \frac{1}{m} \text{ for } x \neq y.}$$

- The expected length of the linked list for x:

$$E[L_x] = E\left[\sum_{y \in S} I_y\right] = \sum_{y \in S} E[I_y] = 1 + \sum_{y \in S \setminus \{x\}} \frac{1}{m} = 1 + (n-1) \cdot \frac{1}{m} = \Theta(1).$$

Dictionaries

- 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.

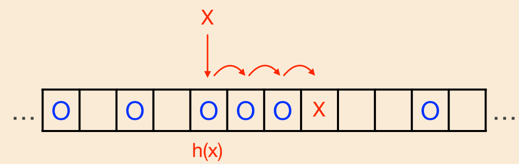
$$h_{a,b}(x) = ax + b \mod p$$
$$H = \{h_{a,b} \mid a \in \{1,\ldots,p-1\}, b \in \{0,\ldots,p-1\}\}.$$

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

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

Open Addressing

- Use a single array for data structure
- linear probing:
  - Insert(x): if h(x) not empty insert at next free slot.
  - Search(x): start from h(x). Search for x until you find it or you find a free slot.



Open Addressing

- Use a single array for data structure
- linear probing:
  - Insert(x): if h(x) not empty insert at next free slot.
  - Search(x): start from h(x). Search for x until you find it or you find a free slot.
  - Delete(x): Find x and mark it deleted.

- Insertions treat tombstones as free. Queries do not.
- Rebuild occasionally (approximately every n operations).
- Keep elements sorted by hash => faster queries.