Lecture 5: Hashing

Elaine Shi

(slides due to David Woodruff)

虎年
2022
大吉
福
福
财

Homework 2 is out!

Hashing

- Universal hashing
- Perfect hashing

Maintaining a Dictionary

- Let U be a universe of “keys”
  - all strings of ASCII characters of length at most 80
  - set of all URLs

128^{80}

- Let S be a subset of U, which is a small “dictionary”
  - all English words
  - set of all URLs you visited this month

$|S| << |U|$

- Support operations to maintain the dictionary
  - Insert(x): add the key x to S
  - Query(x): is the key x in S?
  - Delete(x): remove the key x from S

Dictionary Models

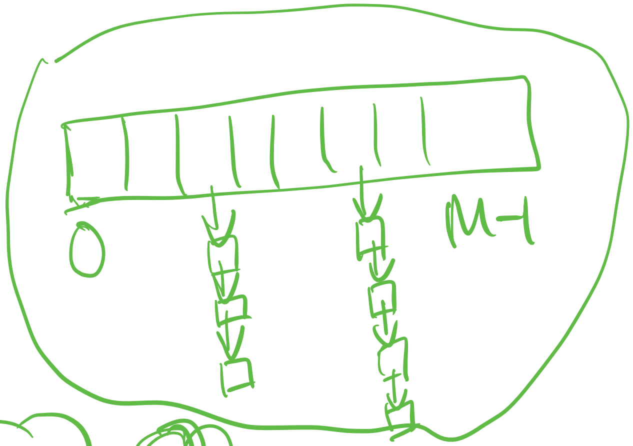
- **Static:** don't support insert and delete operations, just optimize for fast query operations
  - e.g., the English dictionary does not change much
  - Could use a sorted array with binary search

- **Insertion-only:** just support insert and query operations

- **Dynamic:** support insert, delete, and query operations
  - Could use a balanced search tree (AVL trees, splay trees) to get $O(\log |S|)$ time per operation

- Hashing is an alternative approach, often the fastest and most convenient

Formal Hashing Setup



- Universe U is very large
  - E.g., set of ASCII strings of length 80 is 128^{80}
- Care about a small subset $S \subset U$. Let N = |S|.
  - S could be the names of all students in this class

$M \approx |S|$

- Our data structure is an array A of size M and a 'hash function' $h: U \rightarrow \{0, 1, \ldots, M\text{-}1\}$.
  - Typically $M \ll U$, so can't just store each key x in A[x]
  - Insert(x) will try to place key x in A[h(x)]
- But what if h(x) = h(y) for $x \neq y$? We let each entry of A be a linked list.
  - To insert an element x into A[h(x)], insert it at the top of the list
  - Hope linked lists are small

hash: chop and mix

hacher (Fr.): chop

"chop" the input domain into many sub-domains that get "mixed" into the output range to improve the uniformity of the key distribution.

How to Choose the Hash Function h?

$N = |S|$

- Desired properties:
  - unlikely that h(x) = h(y) for different keys x and y
  - array size M to be O(N), where N is number of keys
  - quickly compute h(x) given x --- treat as O(1) time
- How long do Query(x) and Delete(x) take?
  - O(length of list A[h(x)]) time
- How long does Insert(x) take?
  - O(1) time no matter what
  - may first want to check for a duplicate though – that is O(length of list A[h(x)]) time
- **How long can the lists A[h(x)] be?**

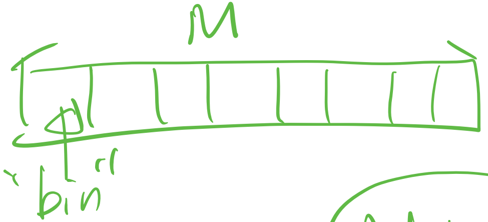
Bad Sets Exist for any Hash Function

- **Claim:** For any hash function h: U -> {0, 1, 2, ..., M-1}, if $|U| \geq (N-1)M + 1$, there is a set S of N elements of U that all hash to the same location
- **Proof:** If every location had at most N-1 elements of U hashing to it, we would have $|U| \leq (N-1)M$
- There's no good hash function h that works for every S. Thoughts?

Randomly choose h!

  - Show for *any* sequence of insert, query, and delete operations, the expected number of operations, over a random h, is small

Idea 1: random hashing



N balls

Suppose h(x) is a **random oracle**

- gives a random answer for any fresh input x
- gives the same answer for a previously queried input

$$E[\text{load in bin } 1] = \frac{N}{M}$$

$$I_i = \begin{cases} 0 & \text{ball } i \notin \text{bin } 1 \\ 1 & \text{ball } i \in \text{bin } 1 \end{cases}$$

$$E\left[\sum_{i=1}^{N} I_i\right] = \sum_{i=1}^{N} E[I_i] = N \cdot \frac{1}{M} = \frac{N}{M}$$

$$M = N$$

$0 \ldots M-1$ $\log_2 M$ bits

Idea 1: random hashing

Space: $U \cdot \log_2 M$

Size of hash family: $2^{U \cdot \log_2 M}$

$\log M$

1 $h(1)$

2 $h(2)$

Suppose h(x) is a **random oracle**

- gives a random answer for any fresh input x
- gives the same answer for a previously queried input

In other words, we choose a random hash function from the family of all functions that map U to {0, … M-1}

$U \cdot \log M$

Problem: does not have a succinct description

Idea 2: Universal Hashing

- Definition: A set H of hash functions h, where each h in H maps U -> {0, 1, 2, ..., M-1} is universal if for all $x \neq y$,

$$\Pr_{h \leftarrow H}[h(x) = h(y)] \leq \frac{1}{M}$$

- The condition holds for every $x \neq y$, and the randomness is only over the choice of h from H

- Equivalently, for every $x \neq y$, we have: $\frac{|h \in H \mid h(x)=h(y)|}{|H|} \leq \frac{1}{M}$

Universal Hashing Examples

Example 1: The following three hash families with hash functions mapping the set $\{a, b\}$ to $\{0, 1\}$ are universal, because at most $1/M$ of the hash functions in them cause a and b to collide, were $M = |\{0, 1\}|$.

	a	b
h_1	0	0
h_2	0	1

	a	b
h_1	0	1
h_2	1	0

	a	b
h_1	0	0
h_2	1	0
h_3	0	1

Examples that are Not Universal

	a	b
h_1	0	0
h_2	1	1

	a	b	c
h_1	0	0	1
h_2	1	1	0
h_3	1	0	1

- Note that a and b collide with probability more than 1/M = 1/2

Universal Hashing Example

- The following hash function is universal with M = |{0,1,2}|

	a	b	c	
h_0	0	0	0	$\leftarrow$ Note!
h_1	0	1	2	
h_2	1	2	0	
h_3	2	0	1	

Using Universal Hashing

- Theorem: If H is universal, then for any set $S \subseteq U$ with $|S| = N$, for any $x \in S$, if we choose h at random from H, the **expected** number of collisions between x and other elements in S is less than N/M.

Using Universal Hashing

- Theorem: If H is universal, then for any set $S \subseteq U$ with |S| = N, for any $x \in S$, if we choose h at random from H, the **expected** number of collisions between x and other elements in S is less than N/M.

- Proof: For $y \in S$ with $y \neq x$, let $\boxed{C_{xy} = 1}$ if h(x) = h(y), otherwise $C_{xy} = 0$

  Let $C_x = \sum_{y \neq x} C_{xy}$ be the total number of collisions with x

  $E[C_{xy}] = \Pr[h(x) = h(y)] \leq \frac{1}{M}$

  By linearity of expectation, $E[C_x] = \sum_{y \neq x} E[C_{xy}] \leq \frac{N-1}{M}$

Using Universal Hashing

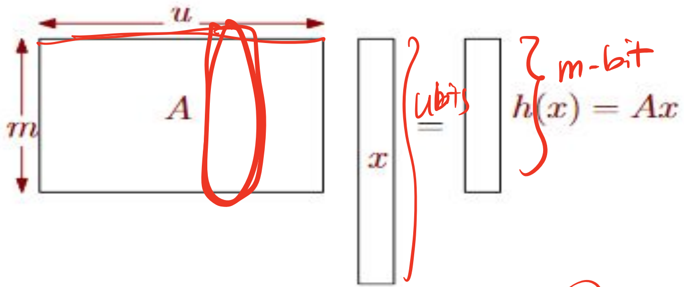
- Corollary: If H is universal, for any **sequence** of L insert, query, and delete operations in which there are at most M keys in the data structure at any time, the expected cost of the L operations for a random $h \in H$ is O(L)
  - Assumes the time to compute h is O(1)

- Proof: For any operation in the sequence, its expected cost is O(1) by the last theorem, so the expected total cost is O(L) by linearity of expectation

But how to Construct a Universal Hash Family?

$x \in U$

- Suppose $|U| = 2^u$ and $M = 2^m$
- Let A be a random m x u binary matrix, and h(x) = Ax mod 2



- Claim: for $x \neq y$, $\Pr_h[h(x) = h(y)] = \frac{1}{M} = \frac{1}{2^m}$

But how to Construct a Universal Hash Family?

$$\Pr[A(x-y)=0] = \sum_{A_{i^*}} \Pr[A_{i^*} = b(A_{-i^*}) \mid A_{i^*}] \cdot \Pr[A_{i^*}]$$

$$= \sum_{A_{i^*}} \frac{1}{M} \cdot \Pr[A_{-i^*}] = \frac{1}{M} \cdot \sum_{A_{-i^*}} \Pr[A_{-i^*}]$$

- **Claim:** For $x \neq y$, $\Pr_h[h(x) = h(y)] = \frac{1}{M} = \frac{1}{2^m}$

i-th column of A

- **Proof:** $A \cdot x \bmod 2 = \sum_i A_i x_i \bmod 2$, where A_i is the i-th column of A

  If h(x) = h(y), then Ax=Ay mod 2, so A(x-y) = 0 mod 2

  If $x \neq y$, there exists an i^* for which $x_{i^*} \neq y_{i^*}$

  Fix A_j for all $j \neq i^*$, which fixes $b = \sum_{j \neq i^*} A_j(x_j - y_j) \bmod 2$

  A(x-y) = 0 mod 2 if and only if $A_{i^*} = b$

  $\Pr_{A_{i^*}}[A_{i^*} = b] = \frac{1}{2^m} = \frac{1}{M}$

A_{-i^*}

$A(x-y) = 0$

b

$= \sum_{j \neq i^*} A_j(x_j - y_j) + A_{i^*}$

all columns that are not A_{i^*}

$$\Pr[A_{i^*} = b(A_{-i^*}) \mid A_{i^*}] = \frac{1}{M}$$

So h(x) = Ax mod 2 is universal

k-wise Independent Families

k-wise independent

- **Definition:** A hash function family H is k-universal if for every set of k distinct keys $x_1, \ldots, x_k$ and every set of k values $v_1, \ldots, v_k \in \{0, 1, \ldots, M-1\}$,

$$\Pr[h(x_1) = v_1 \text{ AND } h(x_2) = v_2 \text{ AND } \ldots \text{ AND } h(x_k) = v_k] = \frac{1}{M^k}$$

$\forall x \neq y$

$\Pr[h(x) = h(y)] \leq \frac{1}{M}$

- If H is 2-universal, then it is universal. Why?

$= \sum_{v \in \{0, \ldots, M-1\}} \Pr[h(x) = v, h(y) = v]$

- h(x) = Ax mod 2 for a random binary A is not 2-universal. Why?

$x = 0$

$= \sum_{v \in \{0, \ldots, M-1\}} \frac{1}{M^2} = \frac{1}{M}$

$k=2$ "pair wise independent"

k-wise Independent Families

- Definition: A hash function family H is k-universal if for every set of k distinct keys $x_1, \dots, x_k$ and every set of k values $v_1, \dots, v_k \in \{0, 1, \dots, M-1\}$,

$$\Pr[h(x_1) = v_1 \text{ AND } h(x_2) = v_2 \text{ AND } \dots \text{ AND } h(x_k) = v_k] = \frac{1}{M^k}$$

- If H is 2-universal, then it is universal. Why?

- h(x) = Ax mod 2 for a random binary A is not 2-universal. Why?

- Exercise: Show Ax + b mod 2 is 2-universal, where A in $\{0,1\}^{m \times u}$ and $b \in \{0,1\}^m$ are chosen independently and uniformly at random


MY DESK ISN'T MESSY
I'M JUST USING
AN ADVANCED HASH ALGORITHM
TO STORE AND RETRIEVE STUFF

More Universal Hashing

- Given a key x, suppose x = $[x_1, \ldots, x_k]$ where each $x_i \in \{0, 1, \ldots, M-1\}$

- Suppose M is prime

- Choose random $r_1, .., r_k \in \{0, 1, \ldots, M-1\}$ and define
$$h(x) = r_1 x_1 + r_2 x_2 + \ \ldots + r_k x_k \bmod M$$

- Claim: the family of such hash functions is universal, in fact, $\Pr_h[h(x) = h(y)] = \frac{1}{M}$ for all distinct x and y

More Efficient Universal Hashing

- Claim: the family of such hash functions is universal, that is, $\Pr_h[h(x) = h(y)] = \frac{1}{M}$ for all $x \neq y$

- Proof: Since $x \neq y$, there is an i^* for which $x_{i^*} \neq y_{i^*}$

  Let $h'(x) = \sum_{j \neq i^*} r_j x_j$, and $h(x) = h'(x) + r_{i^*} x_{i^*} \bmod M$

  If $h(x) = h(y)$, then $h'(x) + r_{i^*} x_{i^*} = h'(y) + r_{i^*} y_{i^*} \bmod M$

  So $r_{i^*}(x_{i^*} - y_{i^*}) = h'(y) - h'(x) \bmod M$, or $r_{i^*} = \frac{h'(y) - h'(x)}{x_{i^*} - y_{i^*}} \bmod M$

  This happens with probability exactly 1/M

Perfect Hashing

static dictionary

- If we fix the dictionary S of size N, can we find a hash function h so that all query(x) operations take constant time?

universal hash

- Claim: If H is universal and $M = N^2$, then $\Pr_{h \leftarrow H}[\text{no collisions in S}] \geq \frac{1}{2}$

"Perfect Hashing"

Static data structure

$Pr[E_1 \text{ or } E_2] \le Pr[E_1] + Pr[E_2]$

- If we fix the dictionary S of size N, can we find a hash function h so that all query(x) operations take constant time?

- Claim: If H is universal and $M = N^2$, then $\Pr_{h \leftarrow H}[\text{no collisions in S}] \ge \frac{1}{2}$

- Proof: How many pairs {x,y} of distinct x,y in S are there?

  Answer: N(N-1)/2

  For each pair, the probability of a collision is at most 1/M

  Pr[exists a collision] ≤ (N(N-1)/2)/M $\le \frac{1}{2}$

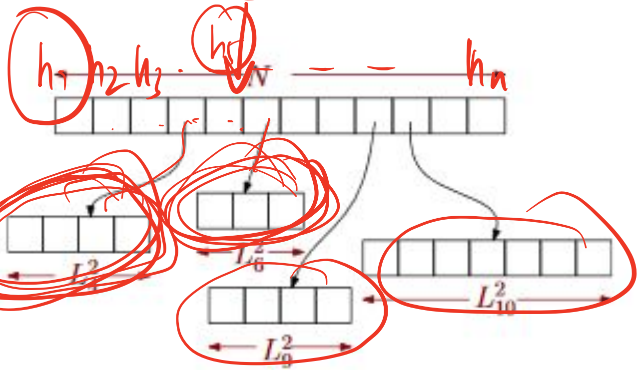
$E[\#\text{tries until I find a good } h] \le 2$

Just try a random h and check if there are any collisions

Problem: our hash table has $M = N^2$ space! How can we get O(N) space?

Perfect Hashing in O(N) Space: 2 Level Scheme

- Choose a hash function $h: U \to \{1, 2, \ldots, N\}$ from a universal family
- Let L_i be the number of items x in S for which h(x) = i
- Choose N "second-level" hash functions $h_1, h_2, \ldots, h_N$, where $h_i: U \to \{1, \ldots, L_i^2\}$



By previous analysis, can choose hash functions $h_1, h_2, \ldots, h_N$ so that there are no collisions, so O(1) time

Hash table size is $\sum_{i=1,\ldots,n} L_i^2$

How big is that??

Perfect Hashing in O(N) Space: 2 Level Scheme

- Theorem: If we pick h from a universal family H, then

$$\Pr_{h \leftarrow H}\left[\sum_{i=1,\dots,N} L_i^2 > 4N\right] \leq \frac{1}{2}$$

$$\sum_{i=1 \dots N} L_i^2 \leq 4N$$

Perfect Hashing in O(N) Space: 2 Level Scheme

- **Theorem:** If we pick h from a universal family H, then

$$\Pr_{h \leftarrow H}\left[\sum_{i=1,\ldots,N} L_i^2 > 4N\right] \leq \frac{1}{2}$$

$$\sum_{i=1 \cdots N} L_i^2$$

- **Proof:** It suffices to show $E[\sum_i L_i^2] < 2N$ and apply Markov's inequality

  Let $C_{x,y} = 1$ if h(x) = h(y). By counting collisions on both sides, $\sum_i L_i^2 = \sum_{x,y} C_{x,y}$

  If x = y, then $C_{x,y} = 1$. If x ≠ y, then $E[C_{x,y}] = \Pr[C_{x,y} = 1] \leq \frac{1}{N}$

  $$E[\sum_i L_i^2] = \sum_{x,y} E[C_{x,y}] = \underbrace{N}_{\sum_x C_{xx}} + \sum_{x \neq y} E[C_{x,y}] \leq N + \underbrace{N(N-1)/N}_{2 \cdot \binom{N}{2} \cdot \frac{1}{N}} < 2N$$

So choose a random h in H, check if $\sum_{i=1,\ldots,n} L_i^2 \leq 4N$, and if so, then choose $h_1, \ldots, h_N$

FKS hashing

ARTICLE

Storing a Sparse Table with O(1) Worst Case Access Time

Authors: Michael L. Fredman, János Komlós, Endre Szemerédi Authors Info & Claims

Journal of the ACM, Volume 31, Issue 3 • July 1984 • pp 538–544 • https://doi.org/10.1145/828.1884