15 → 25

written ↓

CS473 Algorithms: Lecture 14 (2022-03-08)

logistics: - pset 5 due F17
- exam 1 grading by tomorrow evening

last lecture: - randomized algs
- geometric rand var [how long to find rare item]
- randomized selection [random split will shrink problem by 1/4] [wait until good split → geometric rv] [bracket]
- randomized quicksort [divide and conquer] [divide by median] ← randomized selection

today: randomized algs

Q: how to store covid vaccine records, [vaccine mandate] [UIUC building access]

[key] [value] [insert]
(miforbes, vaccine.jpg) → database

miforbes [lookup] →

vaccine.jpg ←

def. A dictionary over $U = \{0, \dots, N-1\}$ is a data structure for storing a set $S \subseteq U$ of [integer] keys x, along w/ associated [integer] values y.

It supports: insert(x,y): add key x to S, w/ value y [edge case: reinserting x w/ new value]

lookup(x): decide if $x \in S$, & return value y

The complexity is measured in terms of $n = |S|$ [at end] formal def.

The time complexity is ...

The space complexity is the number of integers stored by the dictionary [$\geq |S|$]

rmk. per course convention, all integers are $O(\lg n)$ bits $\Rightarrow N \leq poly(n)$
[allows us to reasonably assume $O(1)$ arithmetic ops]
[ideas extend to larger N w/ adjustment of time bounds]

eg: array 1 ... x ... N — not infinite

insert(x,y) — x, y

lookup(z) — z ?

[one slot per universe element]

parameters: space: $N = |U|$ [bad, large]

insert: $O(1)$ [good]

lookup: $O(1)$ [good]

rmk. often $|S| \ll |U|$

actual netids — all possible netids [not practical]

eg: linked list [start] → □ → □ → □ → □ → ... → □

insert(x,y) — (x,y)

[edge case] insert(x,y') — (x,y')

lookup(z)

parameters: space: $O(|S|) = O(n)$ [only store actual elements] [good] [optimal, $\geq n$ storage]

insert: $O(1)$ [insert at front] [good]

lookup: $O(|S|)$ [walk through entire list] [bad]

Q: can we do better? [best of both worlds]

eg: space: $O(n)$

insert: $O(1)$

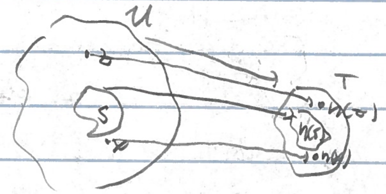
lookup: $O(1)$

A: yes [today], via $\boxed{\text{randomization}}$ [array, linked list deterministic]

↳ $\boxed{\text{but}}$ we accept $\boxed{\text{expected}}$ runtime bounds [so we lose something]

idea: hashing [reduce universe size via $\boxed{\text{hash}}$ function]

$h: U \to T$



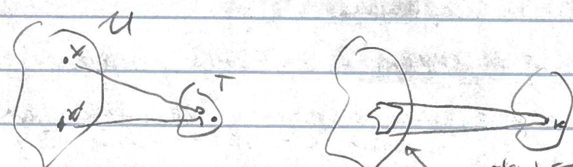
$|T| \approx |S|$ so we can afford an array of size $|T|$

insert(x,y)

lookup(z)

Q: what is the problem?

A: collisions



$h^{-1}(k)$ ← can be $\boxed{\text{large}}$

Q: how to handle collisions?

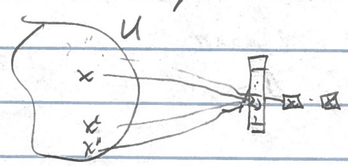
def: A hash table of chaining has — hash function $h: U \to T$, $|T| = m$ [same]

insert(x,y) = insert (x,y) into linked list $L[h(x)]$ — array L of size m [same], of linked lists

lookup(z) = lookup z in linked list $L[h(z)]$

· x vs z ← key

eg:



· correctness

· space of keys

prop: insert(x,y) takes $O(1)$ time, plus 1 evaluation of h [good, [if] h efficient] [insert to linked list] [and h]

[lookup?]

def: the load of a hash function on a set S at key k is $|L[k]|$ [eg]

prop: lookup(x) takes $O(|L[h(x)]|)$ time, plus 1 evaluation of h [need to search entire linked list]

Q: choose h so loads are small?

A: no $\boxed{\text{single}}$ h can work for $\boxed{\text{all}}$ S.

prop: $h: U \to T$ exists $S \subseteq U$ w/ $h(S) = \{k\}$ [all to 1 key] $|S| \geq |U|/|T|$ [large]

pf: $|U| = \sum_{k \in T} \underbrace{|\{x : h(x) = k\}|}_{h^{-1}(k)} \underset{\text{averaging}}{\Rightarrow} \exists k \in T$ w/ $\underbrace{|h^{-1}(k)|}_{=:S} \geq |U|/|T|$

idea: choose h [randomly]

prop: $S \subseteq \mathcal{U}$. $h: \mathcal{U} \to T$ [random] function. Any $z \in \mathcal{U}$

$$\mathbb{E}\left[\,|L[h(z)]|\,\right] \le 1 + \frac{|S|}{|T|} \quad [\text{indicator}], \quad = \Theta(1) \text{ if } |T| = \Theta(S).$$

pf: $$= \sum_{x \in S} \mathbb{1}[h(x) = h(z)] = \mathbb{1}[z \in S] + \sum_{\substack{x \in S \\ x \ne z}} \mathbb{1}[h(x) = h(z)]$$

$$\mathbb{E}[\,—\,] = \underbrace{\mathbb{1}[z \in S]}_{\le 1} \ [\text{linearity}]\ [\text{not random}] + \underbrace{\sum}_{\le |S|} \underbrace{\Pr_h[h(x) = h(z)]}_{= 1/T} \quad [\text{indicator}]$$

$$\le 1 + \frac{|S|}{|T|} \qquad \blacksquare$$

Q: does this work?

A. no: storing $h: \mathcal{U} \to T$ takes $|\mathcal{U}|$ space $\Rightarrow$ back to an array!

idea: choose h [pseudo]randomly - "enough randomness" so

- "not too random" to avoid

def. A universal hash family is a collection of hash functions

$$\mathcal{H} = \{h: \mathcal{U} \to T\} \text{ s.t. } \forall x \ne y \in \mathcal{U},\ \Pr_{h \in \mathcal{H}}[h(x) = h(y)] = 1/T$$

prop: $\mathcal{H}$ universal. Any $z \in \mathcal{U}$ $$\mathbb{E}_{h \in \mathcal{H}}\left[|L[h(z)]|\right] \le 1 + \frac{|S|}{|T|} \quad [\text{same}]$$

[construct?]

prop: p prime

$H: \mathbb{Z}_p^k \times \mathbb{Z}_p^k \to \mathbb{Z}_p$ [integers mod p] given by $H(x, b) = \sum x_i b_i$ [mod p]

$\mathcal{H} = \{h: \mathbb{Z}_p^k \to \mathbb{Z}_p,\ h(x) = H(x, b),\ b \in \mathbb{Z}_p^k\}$ is a universal hash family

each $h \in \mathcal{H}$ - can be stored in $O(k)$ space

~~[typical, then interpret]~~ - can be evaluated in $O(k)$ time

pf - space: h given by k integers [unit cost arithmetic convention]

evaluation: $h(x) = \sum_{i=1}^{k} x_i b_i \Leftarrow O(k)$ ops [$\nearrow$]

universality: [needs more work]

lem: define $\mu_x: \mathbb{Z}_p \to \mathbb{Z}_p$ multiplication map

$y \mapsto xy$

then μ_x is injective [iff] $x \ne 0$

pf: $y, z \in \mathbb{Z}_p$ $\mu_x(y) = \mu_x(z) \equiv xy = xz \ (p)$

$x(y - z) \equiv 0 \ (p)$

$p \mid x(y - z)$ but $p \nmid x$

$\overset{p \text{ prime}}{\Rightarrow} p \mid y - z$

$y \equiv z \ (p) \Rightarrow y = z \quad \blacksquare$

• discuss order of quantifiers
• choose hash function $\boxed{\text{once}}$

lem: μ_x is bijective ($x \neq 0$)

pf: $\mathbb{Z}_p \xrightarrow{\mu_x} \mathbb{Z}_p$ injective $\Rightarrow$ μ_x bijective
counting
[no collision, $\mathbb{Z}_p$] [p, vs p]

lem: $x \neq 0$, Y uniform over $\mathbb{Z}_p$ $\Rightarrow$ $x \cdot Y$ uniform over $\mathbb{Z}_p$

pf: $Pr[\underbrace{x \cdot Y}_{\mu_x(Y)} = z] = Pr[Y = \mu_x^{-1}(z)] = 1/p$
[bijection] [Y uniform]

lem: X random over $\mathbb{Z}_p$, Y uniform over $\mathbb{Z}_p$ $\Rightarrow$ $X+Y$ uniform over $\mathbb{Z}_p$

pf:

$$Pr[X+Y=z] = \sum_x \underbrace{Pr[x+Y=z \mid X=x]}_{} \cdot Pr[X=x] \quad \text{[conditioning]} \; X,Y \text{ indep}$$

$$= \underbrace{Pr[Y=z-x \mid X=x]}_{} \quad \text{[rewrite]}$$

$$= Pr[Y=z-x] \quad \text{[indep]}$$

$$= 1/p \quad \text{[uniform]}$$

$$= 1/p \underbrace{\sum_x Pr[X=x]}_{=1} = 1/p \quad \text{[prob]} \quad \square$$

lem: $\underbrace{x \neq y}_{} \in \mathbb{Z}_p^k$ $\quad \Pr_b[H(x,b) = H(y,b)] = 1/p$ [universality]

pf: $\Rightarrow \exists i_0 \quad x_{i_0} \neq y_{i_0}$

$$\Pr_b[H(x,b) = H(y,b)] = Pr[\sum_{i=1}^{k} x_i b_i = \sum y_i b_i]$$

$$= Pr[\sum b_i(x_i - y_i) = 0]$$

$$= Pr[b_{i_0}\underbrace{(x_{i_0} - y_{i_0})}_{\neq 0} + \underbrace{\sum_{i \neq i_0} b_i(x_i - y_i)}_{} = 0]$$

b_{i_0} [b_i] $i \neq i_0$ uniform indep; uniform, indep

$$= Pr[Z = 0] = 1/p \quad \square$$

uniform $\mathbb{Z}_p$

fact: for any n, can in $O(n)$ ~~deterministic~~ expected time construct a prime p w/ $n \leq p \leq 2n$
[non trivial to prove prime $\boxed{\text{exists}}$]
[— find —]

• use weak GRH

thm: $S \subseteq U$, $|S| = n$, $|U| = N \leq poly(n)$. One can in $O(n)$ deterministic time construct a hash function family $\mathcal{H}: U \to T$ where

- $|T| \in O(n)$
- choosing $h \in \mathcal{H}$ takes $O(1)$ space to store
- $h \in \mathcal{H}$ takes $O(1)$ time to eval
- $\forall x \in S$, $\mathbb{E}_{h \in \mathcal{H}}[\text{time of insert}(x,y)] \in O(1)$
- $\forall z \in U$, $\mathbb{E}_{h \in \mathcal{H}}[\text{time of lookup}(z)] \in O(1)$

pf: choose p w/ $n \leq p \leq 2n$ in $O(n)$ time
take k st $p^k \geq |U|$ $\Rightarrow$ $k \in O(1)$ suffices
$|U| \leq poly(n)$

today: randomized algo
- dictionary problem [insert, delete, lookup?]
- hashing [no $\boxed{\text{simple}}$ / many, linked lists] [randomization] hash function [good expected load] [costly to store] [$\mathbb{Z}$]
- universal hashing [pseudorandom $\Rightarrow$ good expected load, cheap to store] [construction]

$H(x,b) = \sum_{i=1}^{k} x_i b_i$ ($O(k) = O(1)$ ops)
universality $\Rightarrow$ expected $O(1)$ load $\Rightarrow$ $O(1)$ insert / lookup