CS2100

Hashing
+ Comparisons

Recap

- HW due next Wed
- Next Week: graphs
- Review session: last day of class

- Final worksheet (not graded)
  - one question will be on final
- Exam: Wednesday, May 9 at 8am
- Today: in from noon - 2 & 3-4 pm

- HW4 is done (check b-b or git)

Hashing

Problem: Data Storage

Example

Locker #	Name
26	Dan
355	Kevin
101	Nitish
201	David
56	Erin

Goal: Given a locker #, want to be able to retrieve the name - quickly.

Let n = # of people
m = # of lockers

m is much bigger

Could store using:

1. vector/array:
   - size is: $O(m)$
   - lookup: } $O(1)$
   - insert/remove: }

2. List

   locker# " "

   Nitish → Dan → Erin → ...

   - size: $O(n)$
   - lookup: ← $O(n)$
   - insert/remove:

   $O(1)$ ↗ insert   remove ↖ $O(n)$

3. BST:

   $O(\log n)$

Other examples:

- Course # & schedule info
- URL and html page
- Flight # & arrival info
- Color and BMP
- Directors & movies

Python course

Takeaway:

- Not always clear how to get to vector indices!
- Unwilling to tolerate list penalities

$m \gg n$: Goal: $O(n)$ space
$O(1)$ lookup

Dictionary

A data structure which supports:

locker# name

- insert (key, data)
- find (key) → return data
- remove (key)

Note: An array is a kind of dictionary!

key: index or position
data: whatever is stored in it

Hashing

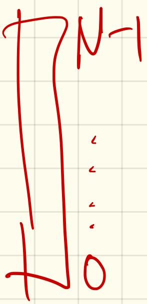
Assume $m \gg n$, so array takes too much space.

Goal: $O(n)$ space

fast lookup/insert/remove

$O(1)$

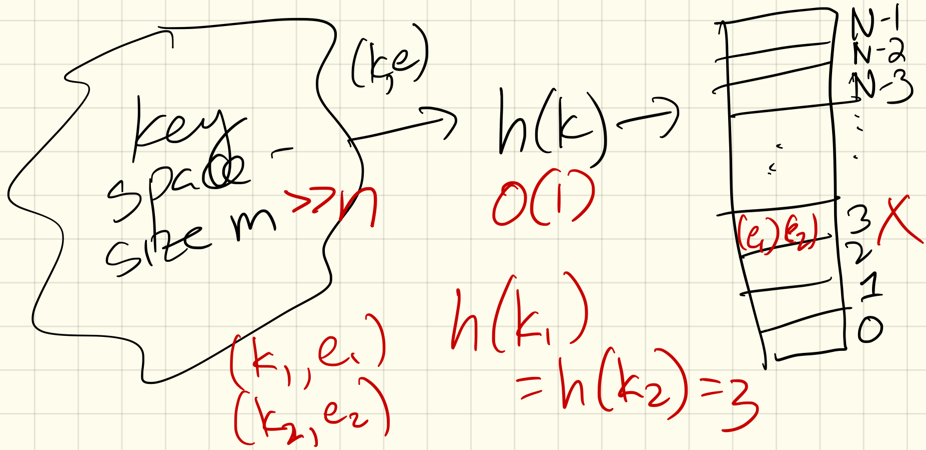
A **hash function** h maps each key to an integer in range $[0..N-1]$



Goal: N is bigger than n, but much smaller than m.

Then: Given (k,e), store it in $A[h(k)]$ (in an array).

Picture:



Good hash functions:

- are fast: $O(1)$
- avoid collisions

(& deal w/ them if they happen)

So, how to do this?

1. Make the key a #
   (binary data)
2. Compress # to $[0, \dots, N-1]$
3. Handle collisions

1 & 2: often combined, & saw some of it last week

We'll recap a bit...

First idea

For something like ASCII, can break into pieces & treat as bits:

E r i n

$$69 + 114 \otimes 105 \quad 110 = \underline{32\text{-bit } \#}$$

Then what?

Problem: this can backfire w/ words:

$$h(\text{temp01}) = h(\text{temp10}) = \text{pm0te1}$$

Want to **avoid** collisions.

So...

Polynomial Hash Codes

Split date to 32-bit pieces.

$$x = (x_0, \dots, x_{k-1})$$

Pick $a \neq 1$. ↑ 32-bit pieces (k of them)

Let $p(x) =$

$$x_0 a^{k-1} + x_1 a^{k-2} + \cdots + x_{k-2} a + x_{k-1}$$

Ex: Erin (or 69, 105, 114, 110)

and $a = 37$:

$$p(x) = 69 \cdot 37^3 + 105 \cdot 37^2 + 114 \cdot 37 + 110$$

$$p(\text{"Erin"}) \neq p(\text{"Eirn"})$$

Why?

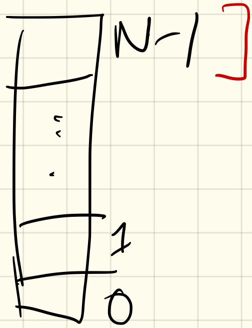
- relatively fast
- avoids collisions! (ones that result from permuting)

Next: Compress:

$h(k)$

32-bit integer

(2^{32})



Idea: Take $h(k) \bmod N$

% in C++

Recall: $3 \bmod 10 = 3$

$60 \bmod 10 = 0$

$14 \bmod 10 = 4$

(remainder)

Example: $h(k) = k \bmod \underline{11}$ ← N

A:

0	1	2	3	4	5	6	7	8	9	10
	(12, E)			(37, I)	(16, N)					(21, R)

Insert: ← key

(12, E) : $h(12) = 12 \bmod 11 = 1$

(21, R) : $h(21) = 10$

(37, I) $h(37) = 4$

(16, N) $h(16) = 5$

(26, C) $h(26) = 4$ ✗

(5, H)

Comment: Works best if ~~x~~ N is prime.

Why? Go take number theory or crypto.

Another way: M.A.D. (multiply, add & divide)

Instead of $h(k) \bmod N$,

do $h(k) = |ak+b| \bmod N$

where a & b are:

- relatively prime
- less than N

Why? - go take NT or crypto

Example: $h(k) = 3k+5 \bmod 11$

A:

0	1	2	3	4	5	6	7	8	9	10
								(12, E)		

Insert:

$h(12) =$

$(12, E) = 3 \cdot 12 + 5 \bmod 11 = 8$

$(21, R)$

$(37, I)$

$(16, N)$

$(26, C)$

$(5, H)$

(collisions may still happen)

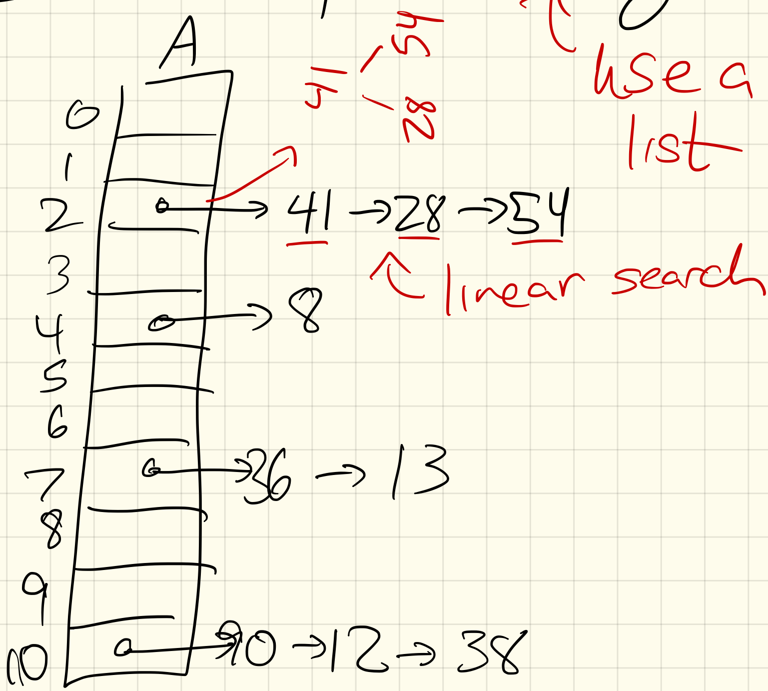
Why bother?

<u>MUCH</u> better in practice

Step 3: Handle Collisions

(Hint: What data structures can store more than 1 thing??)

Ex: Simple Chaining:

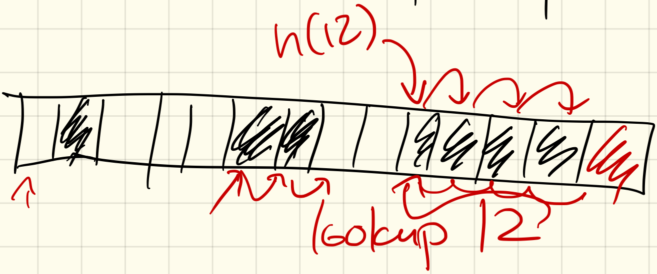


Run times: hash: $O(1)$

Collision time: $O(n)$ or $O(\log n)$ worst case

Another idea!

Linear probing: if we hash to a "full" spot, just walk down to open one.



Issues:

- remove?
  ↳ need to mark as removed, but can't actually free up the space.

Run time:

Worst: $O(n)$

average: $O(1)$ "expected"

Example: $h(k) = k \bmod 11$

A:

0	1	2	3	4	5	6	7	8	9	10
	E			I	~~N~~	C	H			R

Insert:

$(12, E)$: $12 \bmod 11 = 1$

$(21, R)$ $\quad h(21) = 10$

$(37, I)$ $\quad h(37) = 4$

$(16, N)$ $\quad h(16) = 5$

$(26, C)$ $\quad h(26) = 4$ ←

$(5, H)$

⋮

remove(N)

↳ can't!

mark it as "dirty"

Quadratic probing:

linear probing checks

$A[h(k)+j \bmod N]$ for $j = 1, 2, 3, \ldots$

Quadratic:

check $A[h(k)+j^2 \bmod N]$

where $j = 1, 2, \ldots$

Why?

- Avoids these "primary clusters"
- Still fast

Example: $h(k) = k \bmod 11$

A:

0	1	2	3	4	5	6	7	8	9	10

Insert:

(12, E)

(21, R)

(37, I)

(16, N)

(26, C)

(5, H)

Load Factors

Whatever method you use, usually starts to do badly if n gets close to N:

Want $\frac{n}{N} < .5$

Rehashing:

When more than half full, most implementations double the array size

(Still: not too bad – think vectors + our amortized analysis.)

Next time:

- Look at how all these do in practice
- Intro to graphs