Monday, March 11, 2019 12:45 PM

- Assignment 6 due tonight
- Quiz 7 due tonight
- Assignment 7 due Sunday ←

Today:

- Update insert() to include collision resolution via open addressing (linear probing) ↙
- Add search()
- Other open addressing approaches
- New data structure: Heap

Open Addressing via Linear Probing

insertRecord (r)

- find hash value (index)

Open addressing:

if slot is empty table([index])
↳ add record here

else

↳ iterate by incrementing the index one element at a time until empty slot is found.

(watch out for array "roll-over")

search (key)

- calculate hash value (index)
- go to given index

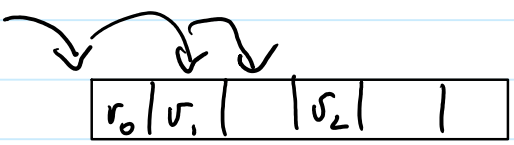
if key matches
↳ retreive record

else
↳ iterate until record w/ matching key is found
(remember circular array)

Tweaks on Open Addressing:

In prior example we used "linear probing"

↳ if slot is occupied, index++, and check.



if table[hash(key)] is occupied
↳ check hash(key) +1

↳ check hash(key) +1
+2
+3
⋮

Problem w/ linear probing: clustering

↳ elements get bunched up
and perf goes down O(1→n)

Quadratic Probing:

Instead of looking at next adjacent slot, skip over by i^2 (indecies).

if table[hash(key)] is occupied
↳ check hash(key) + 1^2

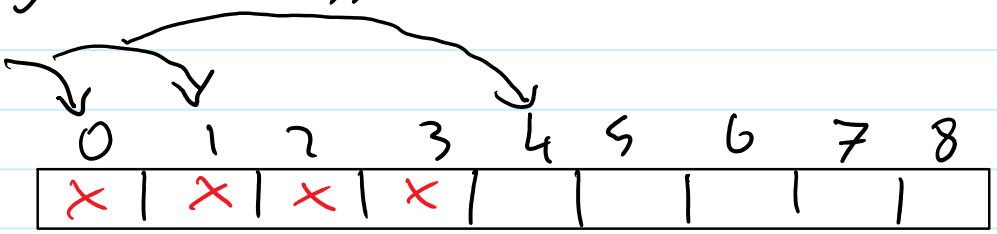
if table[hash(key) + 1^2] is full

↳ check hash(key) + 2^2

⋮

+ 3^2

eg. hash(key) = 0



Hashing Algos Summary:

A) Different hash functions can be used:

- modulo
- multiplication
- division
- djb2 - one of the best gen. purpose hash functions
  ↳ will use it in assignment

B) Collisions need to be handled somehow:

I) Open addressing

- use the array to store all records via linear or quadratic probing

- usually faster under smaller loads

Fixed size

↳ dynamic array doubling

II) Chaining

- use an array to store LL head pointers

- conflicts resolved via add LL

nodes