LEC 17

CSE 123

Hashing

Questions during Class?
Raise hand or send here

sli.do #cse123

BEFORE WE START

Talk to your neighbors:

Favorite breakfast cereal?

Respond on sli.do!

Instructor: Miya Natsuhara

TAs:

Arohan	Shiven	Yuntong	Anya
Sreshta	Vrinda	Amiya	Anisha
Rushil	Gavin	Sahana	Trien
Sean B	Shreya	Anirudh	Neal
Chloe	Jonah	Rohan	Evan
Jenny	Renee	Crystal	Rena
Nate	Chris	Eeshani	
Saachi	Ishita	Prakshi	
Hawa	Kuhu	Aidan	
Maggie	Kavya	Cora	
Sean E	Misha	Nhan	

Music: CSE 123 26sp Lecture Tunes

Announcements

- Programming Project 3 due today, Friday 5/29 at 11:59pm
- Creative Project 3 out, due Friday, 6/5 at 11:59pm
- Resubmission Cycle 6 due **tonight at 11:59pm**
  - ***P1***, C2, P2 eligible
- R7 / R-Gumball will open on Monday
- Final Exam: Thursday, June 11 12:30pm – 2:20pm
  - Information has been posted on the course website
  - Most of next week's class sessions are dedicated to final exam review!
  - TA-led review session on Tuesday, June 9 4:30pm – 7:00pm in JHN 102
- Gumball and friends visit on Monday, June 8 1:00pm – 2:30pm around Drumheller Fountain 🐾

Data structures so far

- **Lists**
  - Maintain an ordered sequence of elements
  - Provides `get()`, `add()`, `remove()`, …
  - Studied two implementations: `ArrayIntList` and `LinkedIntList`

- **Sets**
  - Maintain a collection of elements
  - Provides `contains()`, `add()`, `remove()`, …
  - Implementations?

Set implementations: ArraySet, LinkedSet

	ArraySet(?)	LinkedSet(?)	SortedArraySet	TreeSet	HashSet
contains()	O(n)	O(n)			
add()	O(n)	O(n)			
remove()	O(n)	O(n)			

Set implementations: SortedArraySet

	ArraySet(?)	LinkedSet(?)	SortedArraySet	TreeSet	HashSet
contains()	O(n)	O(n)	O(log(n))		
add()	O(n)	O(n)	O(n)		
remove()	O(n)	O(n)	O(n)		

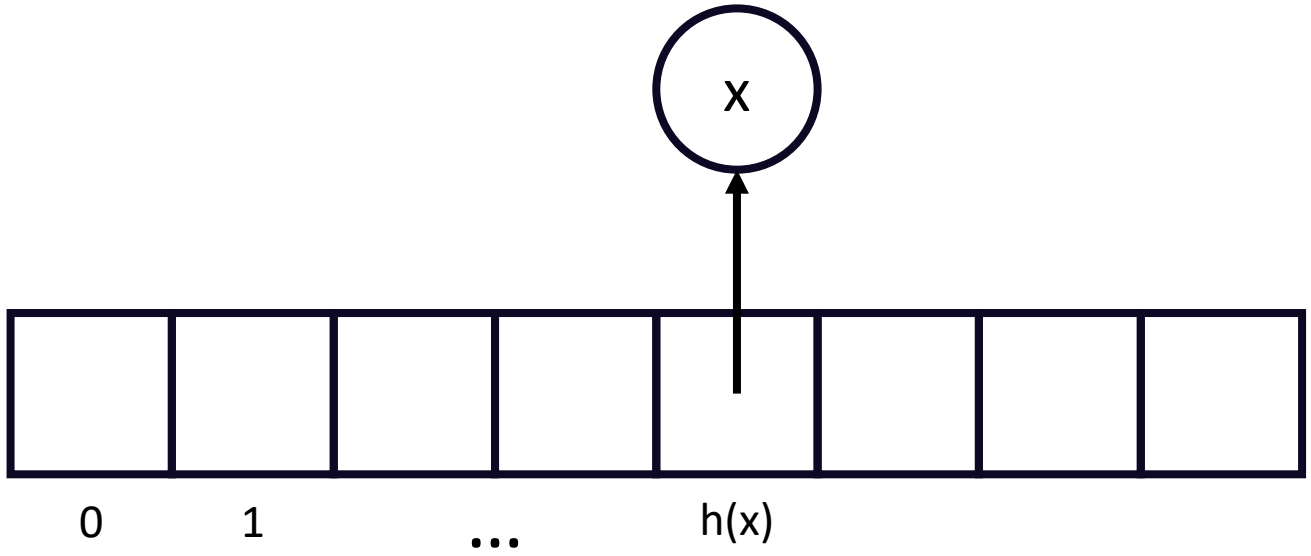
Set implementations: TreeSet

	ArraySet(?)	LinkedSet(?)	SortedArraySet	TreeSet	HashSet
contains()	O(n)	O(n)	O(log(n))	O(log(n))*	
add()	O(n)	O(n)	O(n)	O(log(n))*	
remove()	O(n)	O(n)	O(n)	O(log(n))*	

* assuming tree is balanced

Hash Table

- Define a **hash function** h(x) that turns any value into an integer
  - Call this the values **hash code**
- Create a big array
- Store each value in the array at index h(x)



Set implementations: HashSet

	ArraySet(?)	LinkedSet(?)	SortedArraySet	TreeSet	HashSet
contains()	O(n)	O(n)	O(log(n))	O(log(n))*	O(1)**
add()	O(n)	O(n)	O(n)	O(log(n))*	O(1)**
remove()	O(n)	O(n)	O(n)	O(log(n))*	O(1)**

* assuming tree is balanced

** with some assumptions

What if two items hash to the same slot?

- Nothing stops the hash function from returning the same number for two different items!
- Our hash table needs some way to handle this...
- Any ideas?
- Common solutions:
  - Probing: look at some other related slots
  - Chaining: make each slot a list(!)

What is Linear Probing?

A way to resolve collisions by adding the element in the next available spot

Regular Hash Function

h(x) = x % size

Hash Function (if collision)

h'(x) = [h(x) + f(i)] % size

f(i) = i

What is Quadratic Probing?

A way to resolve collisions by adding the element in the next available spot (quadratically)

Regular Hash Function

h(x) = x % size

Hash Function (if collision)

h'(x) = [h(x) + f(i)] % size

f(i) = i^2

What is Chaining?

A way to resolve collisions by creating a LinkedList at that Index (also called a “bucket”)

- Combines both features of ArrayList Indexing and the ease of adding values using LinkedLists

Recap (Comparison)

Linear Probing	Quadratic Probing	Chaining
A way to resolve collisions by adding the element in the next available spot	A way to resolve collisions by adding the element in the next available spot (quadratically)	A way to resolve collisions by creating a LinkedList at that Index (also called a “bucket”)

Why Chaining? Clustering

Clustering - A tendency for data to clump together when using solutions to Collisions like Linear and Quadratic probing

- Linear and Quadratic Probing often result in “Clustering”
- Inefficient use of space in the table
- This means the Runtimes will also be slower

1	2	3	4	5	6				7							8	9		

Why Chaining? Efficiency

- Hashing can reduce it down to O(1)
- "Load Factor" - lambda ($\boldsymbol{\lambda}$)
  - the number of values in each LinkedList
- Finding the index in the Table is O(1)
- Finding value in LinkedList is O($\boldsymbol{\lambda}$) or essentially O(1)

Standard Java hashCode

```
hash = 0
for (each field) {
    hash = 31 * hash + hash(field)
}
```