CSE 250 Recitation

April 29-30: HashTable Review, ADT Matrix of Fun!

Blooket Review

https://dashboard.blooket.com/set/6627c6c3a688259d39444174

CSE 250 Matrix of ~~Doom~~ Fun

Roll a d6 twice:
https://g.co/kgs/1WcoMvv

The first roll tells you your Data Structure (row)

The second roll tells you your ADT (column)
(if you roll a 6 you get to choose the ADT)

Come up with an implementation for ADT using the Data Structure. Determine the runtime of each key method.

	List	**Stack**	**Queue**	**Priority Queue**	**Set/Bag/ Map**
`ArraryList`	Lecture	Lecture	Lecture		
`LinkedList`	Lecture	Lecture	Lecture	Lecture	
`Heap`				Lecture	
`General BST`					Lecture
`Balanced BST`				Midterm2	Lecture
`Hash Table`					Lecture

Sequence and List Methods

The List ADT extends the Sequence ADT; the following methods are part of both Sequences and Lists

`T get(int idx)`
- Get the element at the specified index

`T set(int idx, T value)`
- Change the element at the specified index to a new value

`int length()`
- Return the length of the Sequence/List

`Iterator<T> iterator()`
- Creates an iterator for the Sequence/List that allows you to access all the elements in order

List Methods

You are unable to change the length of a Sequence. The List ADT extends Sequences by allowing you to add and remove elements

`void add(T value)`
- Add a new value to the end of the list

`void add(int idx, T value)`
- Add a new value at the specified index

`T remove(int idx)`
- Remove the element at the specified index

Stack Methods

Stacks and Queues are extensions upon Lists that enforce a specific order on insertion and removal. **The key feature of Stacks is that they are LIFO.**

`void push(T value)`

- Add an element to the top of the stack

`T pop()`

- Remove and return the element on top of the stack (the most recently added element)

`T peek()`

- Return the top element without removing it

Queue Methods

Stacks and Queues are extensions upon Lists that enforce a specific order on insertion and removal. **The key feature of Queues is that they are FIFO.**

`void enqueue(T value)`

- Add an element to the end of the queue

`T dequeue()`

- Remove and return the element at the front of the queue (the first element that was added)

`T peek()`

- Return the front element without removing it

Priority Queue Methods

Priority Queues provide another way of ordering removals from a collection based on the value rather than when the value was inserted

`void add(T value)`

- Insert the given value into the priority queue

`T poll()`

- Remove and return the value with the highest priority

`T peek()`

- Return the value with the highest priority without removing it

Sets, Bags, and Maps

Sets, Bags and Maps have very subtle differences between them, but these differences give each one unique use cases

- Sets are an unordered collection of **unique** elements
- Bags are an unordered collection of elements **that aren't necessarily unique**
- Maps are an unordered collection of key-value pairs, where all keys are **unique** (AKA it's a set of key-value pairs)

Set Methods

`void add(T element)`

- Store a copy of the given element if a copy doesn't already exist in the set

`boolean contains(T element)`

- Return true if the element exists in the set or false otherwise

`boolean remove(T element)`

- Remove the element if present or return false if the element doesn't exist

Bag Methods

`void add(T element)`

- Store another copy of element in the bag

`int contains(T element)`

- Return the number of copies of element in the bag

`boolean remove(T element)`

- Remove one copy of the element if present or return false if the element doesn't exist

Map Methods

`Optional<V> put(K key, V value)`

- Add the key-value pair to the map, or update it if it already exists, returning the old value

`Optional<V> get(K key)`

- Return the value associated with the key if the key exists or Optional.empty() otherwise

`Optional<V> remove(K key)`

- Remove the key-value pair and return the value, or Optional.empty() if it doesn't exist

ArrayList as List ADT

`T get(int idx)`

- Find index using *base address + (idx * sizeof(T))*. Return value you find there.
- Runtime: O(1)

`T set(int idx, T value)`

- Find index using *base address + (idx * sizeof(T))*. Change value you find there to the new value.
- Runtime: O(1)

`int length()`

- Maintain an integer that represents the length of the Sequence/List. Return the integer.
- Runtime: O(1)

ArrayList as List ADT

`void add(T value)`

- If there is space in the arraylist simply set the value in the last index to the passed in value
- If there is no space, double the length of the array and copy contents over with the new value
- Runtime: Amortized O(1)

`void add(int idx, T value)`

- Double the size of the arraylist if the arraylist is full
- Set the value at the specified index, sliding all subsequent values one position to the right
- Runtime: O(n)

`T remove(int idx)`

- Move all elements to the to the right of idx to the left one spot
- Runtime: O(n)

LinkedList as List ADT

`T get(int idx)`
- Traverse the linkedlist until you reach the specified index. Return the value you find there
- Runtime: O(n)

`T set(int idx, T value)`
- Traverse the linkedlist until you reach the specified index. Change value you find there
- Runtime: O(n)

`int length()`
- Maintain an integer that represents the length of the Sequence/List. Return the integer.
- Runtime: O(1)

LinkedList as List ADT

`void add(T value)`

- Travel to the end of the linkedlist, either by traversal or using a tail reference
- Change the pointers of the old tail to add a new element to the end of the linkedlist
- Runtime: O(n), O(1) by reference

`void add(int idx, T value)`

- Traverse the linkedlist until you reach the specified index
- Change the pointers of the adjacent elements to insert the new element
- Runtime: O(n), O(1) if you already have a reference to the node at idx

`T remove(int idx)`

- Traverse the linkedlist until you reach the specified index
- Change the pointers of the adjacent elements to remove the old element
- Runtime: O(n), O(1) if you already have a reference to the node at idx