CS151 Intro to Data Structures

Priority Queues & Maps

Announcements

HW6 released, due Friday (11/14)

Lab8 due Friday (11/14)

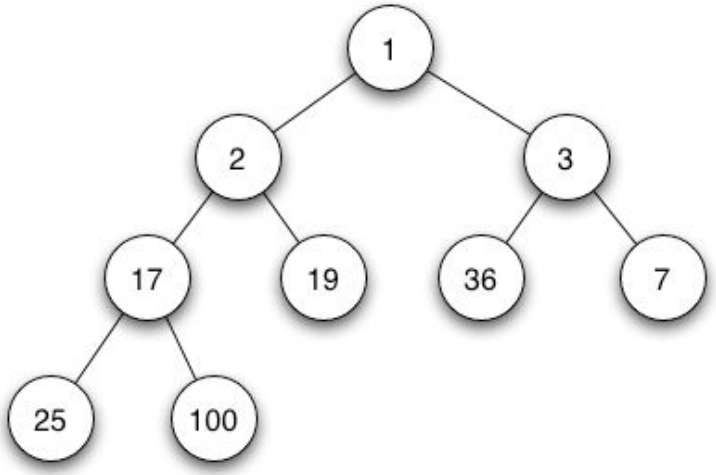
Heap Review

Binary Heap Properties

1. Each node has **at most 2 children**

1. For every node n (except for the root): **n.key >= parent(n).key**

1. **Complete:** all levels of the tree, except possibly the last one are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right

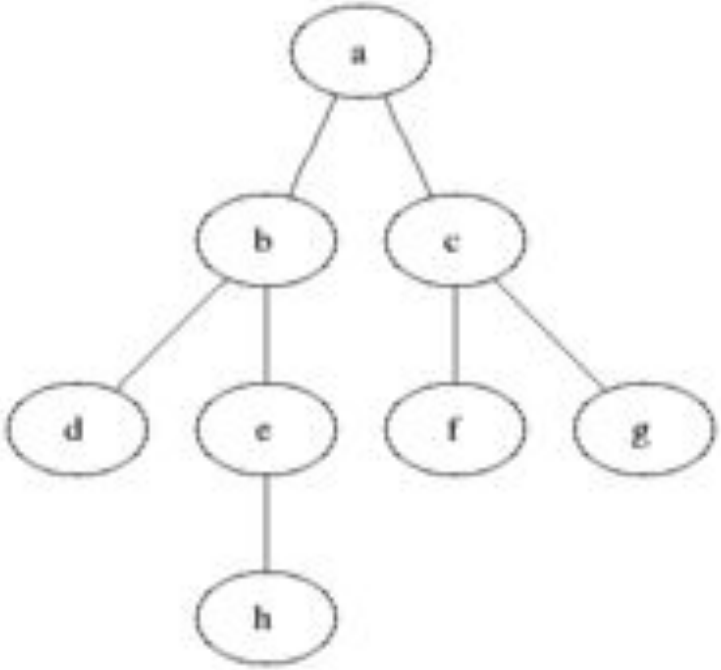


Heap Operations:

1. Insert
   a. Upheap
2. Poll
   a. Downheap
3. Search
4. Remove

Breadth First Search (BFS)

- Breadth First Search (BFS)
  - Starts at the root and explores all nodes at the present “depth” before moving to nodes on the next level
  - Extra memory is usually required to keep track of the nodes that have not yet been explored



Priority Queue

Priority Queues

- What is a queue?

- What if we want to create a graduation queue of students who pick up their diplomas in alphabetical order?
  - Queues and Stacks removal is based on when it was added to the data structure
  - Instead we want removal order to be based on the student's name

Priority Queue

A queue that maintains removal order of the elements according to some priority

- generally not related to insertion time (although time of insertion COULD be one criteria)
- each element has an associated priority with it which indicates when it should be removed
- Usually removed based on min priority
- **What data structure can we use to implement a priority queue?**

Priority Queue ADT

insert(k, v): Creates an entry with key k and value v in the priority queue.

min( ): Returns (but does not remove) a priority queue entry (k,v) having minimal key; returns null if the priority queue is empty.

removeMin( ): Removes and returns an entry (k,v) having minimal key from the priority queue; returns null if the priority queue is empty.

size( ): Returns the number of entries in the priority queue.

isEmpty( ): Returns a boolean indicating whether the priority queue is empty.

Priority Queue

Entries (elements) are Key/Value pairs

The key represents the priority

Key type must be comparable

Entry Interface

```
public interface Entry<K extends Comparable<K>, V> {

  K getKey();
  V getValue();

}
```

Example - minPQ

Method	Return Value	Priority Queue Contents
insert(5,A)		{ (5,A) }
insert(9,C)		{ (5,A), (9,C) }
insert(3,B)		{ (3,B), (5,A), (9,C) }
min( )	(3,B)	{ (3,B), (5,A), (9,C) }
removeMin( )	(3,B)	{ (5,A), (9,C) }
insert(7,D)		{ (5,A), (7,D), (9,C) }
removeMin( )	(5,A)	{ (7,D), (9,C) }
removeMin( )	(7,D)	{ (9,C) }
removeMin( )	(9,C)	{ }
removeMin( )	null	{ }
isEmpty( )	true	{ }

Example - minPQ

Method	Return Value	Priority Queue Contents
insert(5,A)		

Example - minPQ

Method	Return Value	Priority Queue Contents
insert(5,A)		{ (5,A) }

Example - minPQ

Method	Return Value	Priority Queue Contents
insert(5,A)		
insert(9,C)		
insert(3,B)		
min( )		
removeMin( )		
insert(7,D)		
removeMin( )		
removeMin( )		
removeMin( )		
removeMin( )		
isEmpty( )		

Example - minPQ

Method	Return Value	Priority Queue Contents
insert(5,A)		{ (5,A) }
insert(9,C)		{ (5,A), (9,C) }
insert(3,B)		{ (3,B), (5,A), (9,C) }
min( )	(3,B)	{ (3,B), (5,A), (9,C) }
removeMin( )	(3,B)	{ (5,A), (9,C) }
insert(7,D)		{ (5,A), (7,D), (9,C) }
removeMin( )	(5,A)	{ (7,D), (9,C) }
removeMin( )	(7,D)	{ (9,C) }
removeMin( )	(9,C)	{ }
removeMin( )	null	{ }
isEmpty( )	true	{ }

Priority Queues can be Min or Max

Minimum Priority Queue vs Maximum Priority Queue

- Ascending vs Descending Order

min():	Returns (but does not remove) a priority queue entry (k,v) having minimal key; returns null if the priority queue is empty.
removeMin():	Removes and returns an entry (k,v) having minimal key from the priority queue; returns null if the priority queue is empty.

`poll`: removeMin() or removeMax()

`peek`: min() or max()

Updating Key (Priority of an element)

What should happen when you change the key of an existing element in a heap?

What are the cases?

- increaseKey
- decreaseKey

Priority Queue Implementations

Ways to implement a priority queue

1. Heap
2. List
3. Sorted List

Implementing a Priority Queue – Binary Heap

Method	Running Time
size, isEmpty	
min	
insert	
removeMin	

*amortized, if using dynamic array

Implementing a Priority Queue – Binary Heap

Method	Running Time
size, isEmpty	$O(1)$
min	$O(1)$
insert	$O(\log n)^*$
removeMin	$O(\log n)^*$

*amortized, if using dynamic array

Implementing a Priority Queue – **List**

insert(k, v)

- Add the new item to the end of the list

min():

- Search through all the elements and find the element with the smallest key

Implementing a Priority Queue - List

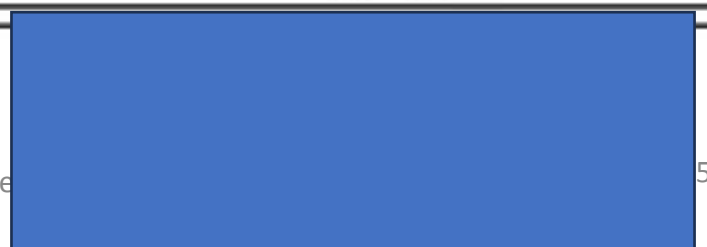
insert(k, v)

- Add the new item to the end of the list

min():

- Search through all the elements and find the element with the smallest key

Method	Running Time
size	$O(1)$
isEmpty	$O(1)$
insert	$O(1)$
min	$O(n)$
removeMin	$O(n)$



Implementing a Priority Queue - List

insert(k, v)

- Add the new item to the end of the list

min():

- Search through all the elements and find the element with the smallest key

Method	Running Time
size	$O(1)$
isEmpty	$O(1)$
insert	$O(1)$
min	$O(n)$
removeMin	$O(n)$

Implementing a Priority Queue - **SortedList**

insert(k, v)

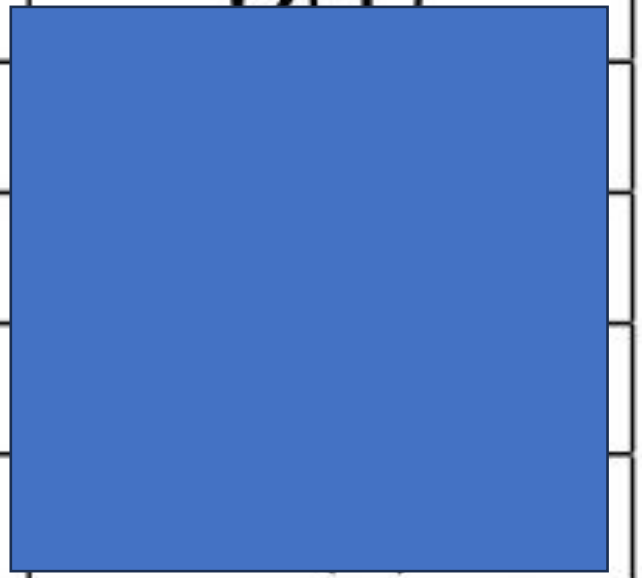
- Find where to put the item based on k, then move other items over

min():

- Find the first element in the list

Implementing a Priority Queue - SortedList

Method	Unsorted List	Sorted List
size	$O(1)$	$O(1)$
isEmpty	$O(1)$	
insert	$O(1)$	
min	$O(n)$	
removeMin	$O(n)$	



Implementing a Priority Queue - SortedList

Method	Unsorted List	Sorted List
size	$O(1)$	$O(1)$
isEmpty	$O(1)$	$O(1)$
insert	$O(1)$	$O(n)$
min	$O(n)$	$O(1)$
removeMin	$O(n)$	$O(1)$

Implementing a Priority Queue

Method	Unsorted List	Sorted List	Binary Heap
size	$O(1)$	$O(1)$	$O(1)$
isEmpty	$O(1)$	$O(1)$	$O(1)$
insert	$O(1)$	$O(n)$	$O(\log n)^*$
min	$O(n)$	$O(1)$	$O(1)$
removeMin	$O(n)$	$O(1)$	$O(\log n)^*$

Maps

Map Motivation

Suppose I have an array of Students. What is the runtime complexity of the following operations?

Update the midterm grade of the second student
O(1)

Update the midterm grade for Liam
O(n) I need to search!

Student:

String name;
double[] hwGrades;
double[] labGrades;
double midtermGrade;
...

Student name = Emily	**Student** name = Aiden	**Student** name = Sophia	**Student** name = Liam	**Student** name = Isabella	**Student** name = Noah	**Student** name = Ava ....

Maps

- Also called “dictionaries” or “associative arrays”

- Similar syntax to an array:
  - `m[key]` retrieves a value
  - `m[key] = value` assigns a value
  - keys need not be ints

- data structure that stores a collection of key-value pairs

Key-Value Pairs

- Each element in a map consists of (K, V)

- The key is used to identify the value
  - In an array, this would be the index

- **Examples**: what are the keys and values here?
  - Dictionary
  - Phone Book
  - Student grades

Map

- A indexable collection of key-value pairs
- Multiple entries with the same key are not allowed

Map ADT

- `get(k)`: if the map `M` has an entry with key `k`, return its associated value; else, return null
- `put(k, v)`: insert entry `(k, v)` into the map `M`; if key `k` is not already in M, then return null; else, replace old value with `v` and return old value associated with `k`
- `remove(k)`: if the map `M` has an entry with key `k`, remove it from `M` and return its associated value; else, return null
- `size()`, `isEmpty()`
- `keySet()`: return an iterable collection of the keys in `M`
- `values()`: return an iterator of the values in `M`
- `entrySet()`: return an iterable collection of the entries in `M`

Example

Method	Return Value	Map
isEmpty()		

Example

Method	Return Value	Map
isEmpty()	true	

Example

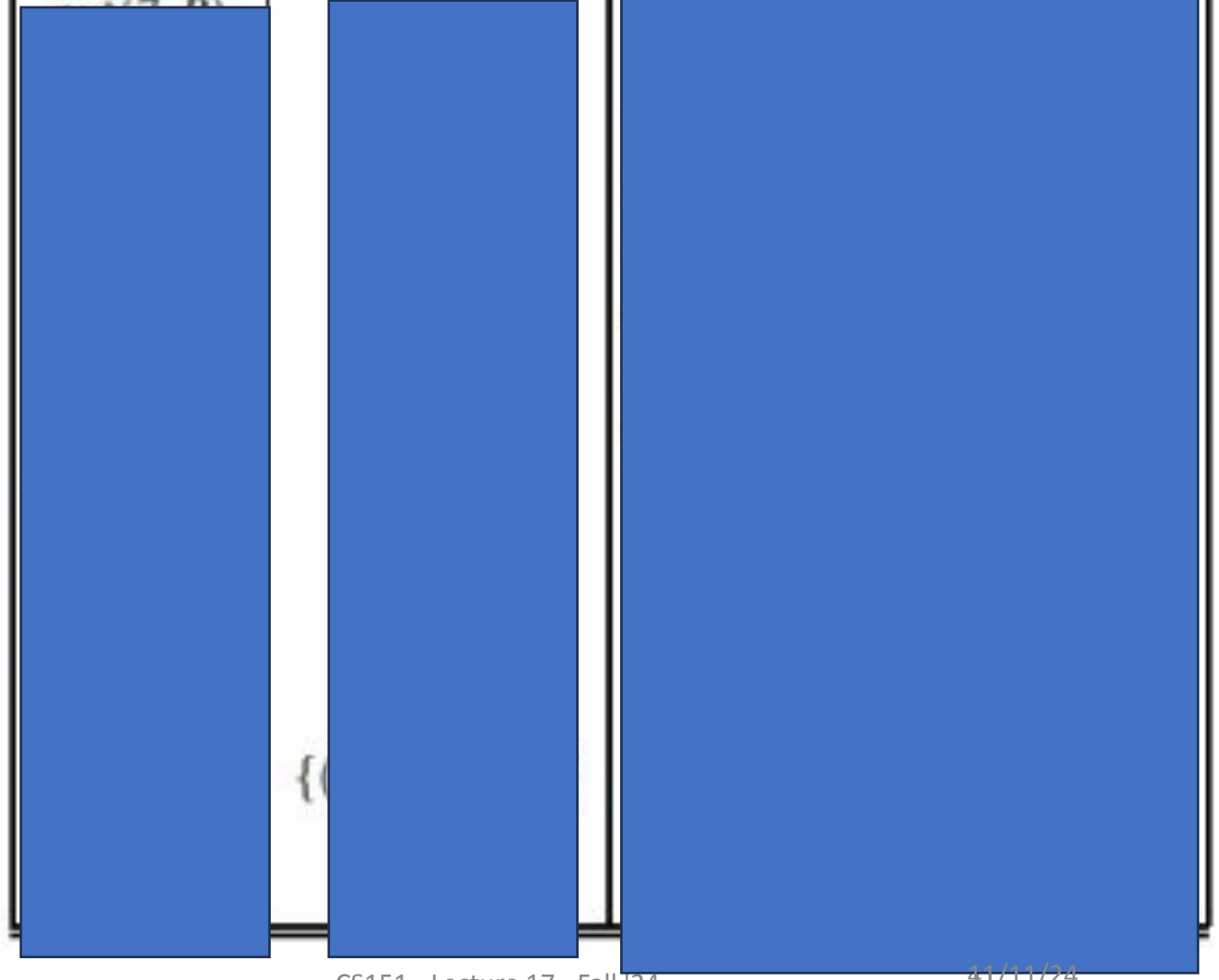
Method	*Return Value*	*Map*
isEmpty()	true	{}

Example

Method	*Return Value*	*Map*
isEmpty()	true	{}
put(5,*A*)		

Example

Method	*Return Value*	*Map*
isEmpty()	true	{}
put(5,*A*)	null	



Example

Method	*Return Value*	*Map*
isEmpty()	true	{}
put(5,*A*)	null	{(5,*A*)}

Example

Method	Return Value	Map
isEmpty()	true	{}
put(5,A)	null	{(5,A)}
put(7,B)		
put(2,C)		
put(8,D)		
put(2,E)		
get(7)		
get(4)		
get(2)		
size()		
remove(5)		
remove(2)		
get(2)		
remove(2)		
isEmpty()		
entrySet()		
keySet()		
values()		

Example

Method	Return Value	Map
isEmpty()	true	{}
put(5,A)	null	{(5,A)}
put(7,B)	null	{(5,A),(7,B)}
put(2,C)	null	{(5,A),(7,B),(2,C)}
put(8,D)	null	{(5,A),(7,B),(2,C),(8,D)}
put(2,E)	C	{(5,A),(7,B),(2,E),(8,D)}
get(7)	B	{(5,A),(7,B),(2,E),(8,D)}
get(4)	null	{(5,A),(7,B),(2,E),(8,D)}
get(2)	E	{(5,A),(7,B),(2,E),(8,D)}
size()	4	{(5,A),(7,B),(2,E),(8,D)}
remove(5)	A	{(7,B),(2,E),(8,D)}
remove(2)	E	{(7,B),(8,D)}
get(2)	null	{(7,B),(8,D)}
remove(2)	null	{(7,B),(8,D)}
isEmpty()	false	{(7,B),(8,D)}
entrySet()	{(7,B),(8,D)}	{(7,B),(8,D)}
keySet()	{7,8}	{(7,B),(8,D)}
values()	{B,D}	{(7,B),(8,D)}

Map ADT

- Map class is abstract
- Concrete Implementations of Map:
  - `UnsortedTableMap`
  - `HashMap`

Map

- How can we implement a map?
  - Array !

Map.Entry Interface

- A (Key, Value) pair
- Keys and Values can be any reference type
- Methods:
  - `getKey()`
  - `getValue()`
  - `setValue(V val)`
- Implementation: `SimpleEntry`

ArrayMap

Let’s implement a `Map` as an array of `SimpleEntry`s

Performance Analysis

	Map implemented with underlying Array	LinkedList
get		
put		
remove		

LinkedList Map

- `get(K key)`
- `put(K key, V value)`
  - If k is not in the map add it. If it is in the map, replace with the new value.
- `remove(K key)`

Performance Analysis

	Array	LinkedList
get	O(n)	
put	O(n)	
remove	O(n)	

Summary

1. Priority Queue
   a. removal order based on a “priority”
   b. efficiently implemented with a heap
2. Maps
   a. stores key-value pairs