Priority Queues and Heaps

Reading LDC Ch 18
Queues with Priorities

Queue:

1. cs230
2. cs231
3. cs111
4. cs110
5. cs235
6. cs115

Priority Queue:

1. cs110
2. cs111
3. cs115
4. cs230
5. cs231
6. cs235
How to implement a Priority Queue?

- **Keep them sorted!**
  (Haven’t we implemented it already?)
  - **Appropriate if**
    **the number of items is small**

- Sorted Array-based implementation

- Linked List-based implementation

- Binary search tree implementation
But then... what kind of tree is this?
Heap, minheap, maxheap

- A **minheap** is a complete binary tree in which each node’s element is less than or equal to both of its children.
- A minheap keeps the smallest element readily available.

- A **maxheap** is __________ ________________
- A maxheap keeps the _______ element readily available.

Computed Link Array Implementation!

| 110 | 115 | 111 | 307 | 230 | 349 | 112 | 342 | 331 |
**Computed Links in an Array**

- Place tree nodes in specific indices of the array
- A node’s index can be used to calculate the indices of its parent and children

D is in index location \( x = 3 \).

Find indices of D’s children:  
- \( \text{leftChild}(x) = 2x + 1 \)
- \( \text{rightChild}(x) = 2x + 2 \)

How about index of D’s parent?  
- \( \text{parent}(x) = \frac{x - 1}{2} \)
Heaps can implement PQs!

```plaintext
priority queue

110
  115
    307
  111
    230
    231

115 231 110 111 307 230
```

1 2 3 4 5 6

110 111 115 230 231 307
Maxheap, Minheap, or Neither?

110
115
111
331 230 307

zebra
tiger
moose
panda lion koala

Maxheap, Minheap, or Neither?

zebra
110
115
230
331

zebra
tiger
moose
panda lion koala

Maxheap, Minheap, or Neither?
Heaps, MaxHeaps, MinHeaps

- A minheap is a complete binary tree in which each node’s element is less than or equal to both of its children.
- A minheap keeps the smallest element readily available.

- A maxheap is ____________________________
- A maxheap keeps the ________ element readily available.

- Which Data Structure to extend to create a heap?

- Three primary operations for maxheaps:

MaxHeap extends BinaryTree

package javafoundations;

public interface MaxHeap<T extends Comparable<T>>
    extends BinaryTree<T>
{
    // Adds the specified object to the heap.
    public void add (T obj);

    // Returns a reference to the element with the highest value in
    // the heap.
    public T getMax ();

    // Removes and returns the element with the highest value in the
    // heap.
    public T removeMax ();
}
Adding an Element to a MaxHeap: **example**

- Insert new course: 349 in a maxheap!
- **Strategy**
  - Insert new item into the bottom of the tree
  - “Trickle up” (aka: swim up) the new item to appropriate spot in the tree
Removing the Max from a MaxHeap: example

- Remove old course: 331.
  - Step 1: Delete (& remember to return) the item at the root
    - Results in disjoint heaps
  - Step 2: Copy the item from the last node into the root, resulting in a “semiheap”
Heaps: heapify

- Step 3: Transform the semiheap back into a heap by “trickling down” (aka “sink down”) the smallest-of-three element
  - Performed by the recursive calls to heapify

How many times do we need to call heapify?
Can we use a Heap to sort?

- Algorithm:

- Efficiency?

- More efficient strategy
  - The second half of the array represents a bunch of (one-node) heaps
    - Use heapify() to fix the first-half nodes
(Max) Heapsort: example

Initial input array

Initial heap.  After heapify(2)  After heapify(1)  Final heap tree
Next: heapify(2)  Next: heapify(1)  Next: heapify(0)

for (int i = n/2 - 1; i >= 0; i--)  
  heapify(i);
package javafoundations;

public interface MaxHeap<T extends Comparable<T>>
    extends BinaryTree<T>
{
    // Adds the specified object to the heap.
    public void add (T obj);

    // Returns a reference to the element with the highest value in the heap.
    public T getMax ();

    // Removes and returns the element with the highest value in the heap.
    public T removeMax ();
}
Since PriorityQueue implements Queue...

```java
package javafoundations;

public interface Queue<T> {
    // Adds the specified element to the rear of the queue.
    public void enqueue (T element);

    // Removes and returns the element at the front of the queue.
    public T dequeue();

    // Returns a reference to the element at the front of the queue
    // without removing it.
    public T first();

    // Returns true if the queue contains no elements and false
    // otherwise.
    public boolean isEmpty();

    // Returns the number of elements in the queue.
    public int size();

    // Returns a string representation of the queue.
    public String toString();
}
```
A data structure that works like a queue, but instead of FIFO, always dequeues the item with the highest priority.
Uses a maxheap to store and sort items.

package javafoundations;
import javafoundations.exceptions.*;

public class PriorityQueue<T> extends Comparable<T>>
    implements Queue<T> {

    private LinkedMaxHeap<T> heap;

    // Creates a new, empty priority queue.

    public PriorityQueue()
    {
    }

    (more...)
public T first() {
    return heap.getMax();
}

public boolean isEmpty() {
    return heap.isEmpty();
}

public int size() {
    return heap.size();
}

public String toString() {
    return heap.toString();
}
Enqueues an Comparable element.

```
public void enqueue(T element) {
    heap.add(element);
}
```

Dequeues the max of all elements in the heap.

```
public T dequeue(){
    try {
        T temp = heap.removeMax();
        return temp;
    } catch(EmptyCollectionException ece) {
        System.out.println(ece);
    }
    return null;
}
```

(more...)