Heaps and Priority Queue

Outline

- Priority Queue
- Binary Heaps
- Implementation and demo
- HeapSort
Example 1: Scheduling

- **EDF (Earliest Deadline First) Scheduling**
  - Tasks wait in the queue
  - A task with a shorter deadline has a higher priority
  - Executes a job with the earliest deadline

Q \[ T_1 \quad T_3 \quad T_2 \quad \ldots \quad T_n \quad \ldots \]
Example 1: Cont.

- Task T1 is dispatched and removed from the Task waiting queue.

- Before T1 is completed, Task Tn+1 arrives. It has the earliest deadline. Tn+1 will be dispatched next.
Priority Queue

• EDF scheduler processes Tasks in order. But not necessarily in full sorted order and not necessarily all at once.

• An appropriate data type for Task Waiting Queue supports two operations: remove the maximum priority task and insert new tasks. Such a data type is called a priority queue.

• Priority queues are characterized by the remove the maximum and insert operations.
public interface PriorityQueue <T extends Comparable<T> >
{
    void insert(T t);
    void remove() throws EmptyQueueException;
    T top() throws EmptyQueueException;
    boolean empty();
}
Example 2: Statistics

- Find the largest $M$ items in a stream of $N$ items ($N$ huge, $M$ large)
  - $N$ is huge, cannot sort in memory
  - $M$ is large, insert, remove must be fast.

Order of growth of finding the largest $M$ in a stream of $N$ items

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort</td>
<td>$N \log N$</td>
<td>$M$</td>
</tr>
<tr>
<td>Array</td>
<td>$N \cdot M$</td>
<td>$M$</td>
</tr>
</tbody>
</table>
Elementary Implementations

- Unordered Array:
  - Order of growth of running time for priority queue with N items

- Ordered Array:

- Linked List:

- Binary Tree

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Insert</th>
<th>Remove Max</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unordered Array</td>
<td>1</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Ordered Array</td>
<td>N</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Linked List (unsorted)</td>
<td>1</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td>Log N</td>
<td>Log N</td>
<td>1</td>
</tr>
</tbody>
</table>
Binary Heap

- Complete Binary Tree
- Each node is larger than (or equal to) its two children (if any).
Complete Binary Tree in Nature
Binary Heap Properties

- The largest is found at the root.
- Height of complete tree with N nodes is $\lceil \log_2 N \rceil$
- Height only increases when N is a power of 2
Binary Heap Representations

- Array representation of a complete binary tree
  - Take nodes in level order
  - No explicit links needed
Binary Heap Representations

- Largest key is \( a[1] \), which is root of binary tree.
- Can use array indices to move through tree.
- Parent of node at \( k \) is at \( k/2 \).
- Two children of the node at \( k \) are in positions \( 2k \) and \( 2k + 1 \).
**Promotion**: Child's key becomes larger key than its parent's key.

To eliminate the violation:
- Exchange key in child with key in parent.
- Repeat until heap order restored.

```java
private void swim(int k) {
    while (k > 1 && less(k/2, k)) {
        swap(k, k/2);
        k = k/2;
    }
}
```
Insertion in a heap:
- Insert. Add node at end, then swim it up.
- Cost. At most $\lg N$ compares.

```java
public void insert(T t){
    pqArray.add(t);
    Size++;
    swim(Size);
}
```
Demotion: Parent's key becomes smaller than one (or both) of its children's keys.

To eliminate the violation:
- Exchange key in parent with key in larger child.
- Repeat until heap order restored.

```java
private void sink(int k) {
    while (2 * k <= Size) {
        int j = 2*k;
        if (j < Size && less(j, j+1)) j++;
        if (!less(k, j)) break;
        swap(k, j);
        k = j;
    }
}
```
Remove the maximum in a heap:
- Delete max: Replace root with node at end, then sink it down.
- Cost: At most $2 \lg N$ compares.

```java
public void remove(){
    if(Size == 0){
        throw new EmptyQueueException("Queue is empty.");
    }
    pqArray.set(1,pqArray.get(Size));
pqArray.remove(Size);
    Size--;
sink(1);
}
```
Binary Heap Demo

Insertion

Insert 34

Violation. swim
Binary Heap Demo

Insertion

Violation.

swim
Binary Heap Demo

Insertion

Violation.

swim
Binary Heap Demo

Insertion

---

Done!

Violation.

swim
Binary Heap Demo

Remove max:

Delete the last leaf
Move the last leaf to root
Binary Heap Demo

Remove

Violation. sink
Violation. sink

7/14/18  Prioriry Queue
Binary Heap Demo

Remove

Violation. sink
Binary Heap Demo

Remove

Done!
<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PriorityQueue.java</td>
<td>Interface</td>
</tr>
<tr>
<td>MaxPQ.java</td>
<td>PQ implementation</td>
</tr>
<tr>
<td>GraphVizWrite.java</td>
<td>Visualize the heap</td>
</tr>
<tr>
<td>EmptyQueueException.java</td>
<td>Exception</td>
</tr>
<tr>
<td>MaxPQTest.java</td>
<td>main method</td>
</tr>
<tr>
<td>InputHelper.java</td>
<td>input utility</td>
</tr>
</tbody>
</table>
## Cost summary

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Insert</th>
<th>Remove Max</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unordered Array</td>
<td>1</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Ordered Array</td>
<td>N</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Linked List (unsorted)</td>
<td>1</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td><strong>Binary Heap</strong></td>
<td><strong>Log N</strong></td>
<td><strong>Log N</strong></td>
<td><strong>1</strong></td>
</tr>
</tbody>
</table>
Immutability of keys

- Assumption: client does not change keys while they're on the PQ.
- Best practice: use immutable keys.

Immutability: implementing in Java
- Immutable data type. Can't change the data type value once created.
- Immutable. **String, Integer, Double, Color, Vector, Transaction, Point2D.**
- Mutable. **StringBuilder, Stack, Counter, Java array.**