Priority Queues (Heaps)

Chapter 6 in Weiss

**Reading:** pp.211-222, pp. 222-225 examples.

---

Priority Queue ADT

- Checkout line at the supermarket
- Printer queues
- operations: insert, deleteMin

---

Implementations of Priority Queue ADT

<table>
<thead>
<tr>
<th></th>
<th>insert</th>
<th>deleteMin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted list (Array)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unsorted list (Linked-List)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted list (Array)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted list (Linked-List)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVL tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hash Table</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Binary Heap Properties

1. Structure Property
2. Ordering Property

---

Brief interlude: Some Definitions:

**A Perfect** binary tree – A binary tree with all leaf nodes at the same depth. All internal nodes have 2 children.

- height $h$
- $2^{h+1} - 1$ nodes
- $2^h - 1$ non-leaves
- $2^h$ leaves

**Full** Binary Tree

- A binary tree in which each node has exactly zero or two children.
- (also known as a proper binary tree)
- (we will use this later for Huffman trees)
Heap Structure Property

- A binary heap is a **complete** binary tree.

**Complete binary tree** – binary tree that is completely filled, with the possible exception of the bottom level, which is filled left to right.

Examples:

```
    5
   / \
  3   4
 /   /\  /
2   1  3 6
```

Complete binary tree of height h

- For $h = 0$, just a single node.
- For $h = 1$, left child or two children.
- For $h \geq 2$, either
  - the left subtree of the root is full with height $h-1$ and the right is complete with height $h-1$, **OR**
  - the left is complete with height $h-1$ and the right is full with height $h-2$.

Representing Complete Binary Trees in an Array

From node $i$:
- left child: $2i + 1$
- right child: $2i + 2$
- parent: $(i - 1) / 2$

Implicit (array) implementation:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Why better than pointers?

Heap Order Property

**Heap order property**: For every non-root node $X$, the key in the parent of $X$ is less than (or equal to) the key in $X$.

```
    10
   /\   /
  20 80
 /\  /\  \
65 85 80 99
```

not a heap

Heap Operations

- findMin:
- insert(val): percolate up.
- deleteMin: percolate down.
Heap – Insert(val)

Basic Idea:
1. Put val at “next” leaf position
2. Repeatedly exchange node with its parent if needed

Heap – Deletemin

Basic Idea:
1. Remove root (that is always the min!)
2. Put “last” leaf node at root
3. Find smallest child
4. Swap node with smallest child if needed.
5. Repeat steps 3 & 4 until no swaps needed.

Other Heap Operations

decreaseKey(process, amount): raise the priority of a process, percolate up
increaseKey(processID, amount): lower the priority of a process, percolate down
remove(processID): remove a process, move to top, then delete.
   1) decreaseKey(processID, ∞)
   2) deleteMin()

Worst case Running time for all of these: O(log N)
FindMax?
ExpandHeap – when heap fills, copy into new space.
Heaps (summary)

- insert: percolate up. \( O(\log N) \) time.
- deleteMin: percolate down. \( O(\log N) \) time.

- Heapsort?