Priority Queues

- Priority Queues
- Trees and Heaps
- Representations of Heaps
- Algorithms on Heaps
- Building a Heap
- Heapsort

Philip Bille

Priority Queues

- Priority Queues
- Trees and Heaps
- Representations of Heaps
- Algorithms on Heaps
- Building a Heap
- Heapsort

Applications.

- Scheduling
- Shortest paths in graphs (Dijkstra's algorithm)
- Minimum spanning trees in graphs (Prim's algorithm)
- Compression (Huffman's algorithm)
- ...

Challenge. How can we solve problem with current techniques?
Priority Queues

• Solution 1: Linked list. Maintain S in a doubly-linked list.

  \[ 7 \rightarrow 42 \rightarrow 18 \rightarrow 23 \rightarrow 5 \rightarrow 10 \rightarrow 56 \rightarrow 2 \]

  • `MAX()`: linear search for largest key.
  • `EXTRACTMAX()`: linear search for largest key. Remove and return element.
  • `INCREASEKEY(x, k)`: set x.key = k.
  • `INSERT(x)`: add element to front of list (assume element does not exist in S beforehand).

• Time.
  • `MAX` and `EXTRACTMAX` in O(n) time (n = |S|).
  • `INCREASEKEY` and `INSERT` in O(1) time.

• Space.
  • O(n).

Priority Queues

• Solution 2: Sorted linked list. Maintain S in a sorted doubly-linked list.

  \[ 56 \rightarrow 42 \rightarrow 23 \rightarrow 18 \rightarrow 10 \rightarrow 7 \rightarrow 5 \rightarrow 2 \]

  • `MAX()`: return first element.
  • `EXTRACTMAX()`: return and remove first element.
  • `INCREASEKEY(x, k)`: set x.key = k. Linear search to move x to correct position.
  • `INSERT(x)`: linear search to insert x at correct position.

• Time.
  • `MAX` and `EXTRACTMAX` in O(1) time.
  • `INCREASEKEY` and `INSERT` in O(n) time.

• Space.
  • O(n).

Priority Queues

- Challenge. Can we do significantly better?

<table>
<thead>
<tr>
<th>Data structure</th>
<th>Max</th>
<th>EXTRACTMAX</th>
<th>INCREASEKEY</th>
<th>INSERT</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>linked list</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>sorted linked list</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

Priority Queues

• Trees and Heaps
  - Representations of Heaps
  - Algorithms on Heaps
  - Building a Heap
  - Heapsort
Trees

- Rooted trees.
  - Nodes (or vertices) connected with edges.
  - Connected and acyclic.
  - Designated root node.
  - Special type of graph.

- Terminology.
  - Children, parent, descendant, ancestor, leaves, internal nodes, path,..

- Depth and height.
  - Depth of \( v \) = length of path from \( v \) to root.
  - Height of \( v \) = length of path from \( v \) to descendant leaf.
  - Depth of \( T \) = height of \( T \) = length of longest path from root to a leaf.

Trees

- Binary tree.
  - Rooted tree.
  - Each node has at most two children called the left child and right child.

- Complete binary tree. Binary tree where all levels of tree are full.
- Almost complete binary tree. Complete binary tree with 0 or more rightmost leaves deleted.

- Lemma. Height of an (almost) complete binary tree with \( n \) nodes is \( \Theta(\log n) \).
  - Pf. See exercises.

Heaps

- Heaps. Almost complete binary tree that satisfies heap-order.

- Heap-order.
  - All nodes store one element.
  - For all nodes \( v \).
    - all keys in left subtree and right subtree are \( \leq v.key \).

- Max-heap vs min-heap.

Priority Queues

- Priority Queues
- Trees and Heaps
- Representations of Heaps
  - Algorithms on Heaps
  - Building a Heap
  - Heapsort
Heap

- Data structure. We need the following navigation operations on a heap.
  - PARENT(x): return parent of x.
  - LEFT(x): return left child of x.
  - RIGHT(x): return right child of x.

- Challenge. How can we represent a heap compactly to support fast navigation?

Heap

- Linked representation. Each node stores
  - v.key
  - v.parent
  - v.left
  - v.right
  - PARENT, LEFT, RIGHT by following pointer.

- Time. O(1)
- Space. O(n)

Heap

- Array representation.
  - Array H[0..n]
  - H[0] unused
  - H[1..n] stores nodes in level order.

  - PARENT(x): return ⌈x/2⌉
  - LEFT(x): return 2x.
  - RIGHT(x): return 2x + 1

- Time. O(1)
- Space. O(n)

Priority Queues

- Priority Queues
- Trees and Heaps
- Representations of Heaps
- Algorithms on Heaps
- Building a Heap
- Heapsort
**Algorithms on Heaps**

- **BubbleUp(x):**
  - If heap order is violated at node x because key is larger than key at parent:
  - Swap x and parent
  - Repeat with parent until heap order is satisfied.

- **BubbleDown(x):**
  - If heap order is violated at node x because key is smaller than key at left or right child:
  - Swap x and child c with largest key.
  - Repeat with child until heap order is satisfied.

**Priority Queues**

- **Max()**
  - return H[1]

- **ExtractMax()**
  - r = H[1]
  - H[1] = H[n]
  - n = n - 1
  - BubbleDown(1)
  - return r

- **Insert(x)**
  - n = n + 1
  - H[n] = x
  - BubbleUp(n)

- **IncreaseKey(x, k)**
  - H[x] = k
  - BubbleUp(x)

- **Ex.** Trace execution of following sequence in initially empty heap: 2, 5, 7, 6, 4, E, E
- Numbers mean Insert og E is ExtractMax. Draw heap after each operation.

**Time.**
- **BubbleUp** and **BubbleDown** in $\Theta(\log n)$ time.
- How can we use them to implement a priority queue?
Heaps with array data structure is an example of an implicit data structure.

### Priority Queues

<table>
<thead>
<tr>
<th>Data structure</th>
<th>MAX</th>
<th>EXTRACTMAX</th>
<th>INCREASE KEY</th>
<th>INSERT</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>linked list</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>sorted linked list</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>heap</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

- Heaps with array data structure is an example of an implicit data structure.

### Priority Queues

- Priority Queues
- Trees and Heaps
- Representations of Heaps
- Algorithms on Heaps
- Building a Heap
- Heapsort

### Prioritetskøer

- Prioritetskøer
- Træer og hobe
- Repræsentation af hobe
- Algoritmer på hobe
- Hobkonstruktion
- Hobsortering

### Building a Heap

- **Building a heap.** Given $n$ integers in an array $H[0..n]$, convert array to a heap.
Building a Heap

- **Solution 1: top-down construction**
  - For all nodes in increasing level order apply BUBBLEUP.

  ![Diagram of a binary tree with arrows indicating BUBBLEUP process]

- **Time.**
  - For each node of depth d, we use $O(d)$ time.
  - $1$ node of depth $0$, $2$ nodes of depth $1$, $4$ nodes of depth $2$, ..., $\sim n/2$ nodes of depth $\log n$.
  - $\Rightarrow$ total time is $\Theta(n \log n)$
- **Challenge.** Can we do better?

Building a Heap

- **Solution 2: bottom-up construction**
  - For all nodes in decreasing level order apply BUBBLEDOWN.

  ![Diagram of a binary tree with arrows indicating BUBBLEDOWN process]

- **Time.**
  - For each node of height $h$ we use $O(h)$ time.
  - $1$ node of height $\log n$, $2$ nodes of height $\log n - 1$, ..., $n/4$ nodes of height $1$, $n/2$ nodes of height $0$.
  - $\Rightarrow$ total time is $\Theta(n)$ (see exercise)

Priority Queues

- Priority Queues
- Trees and Heaps
- Representations of Heaps
- Algorithms on Heaps
- Building a Heap
- Heapsort

Heapsort

- **Sorting.** How can we sort an array $H[1..n]$ using a heap?
- **Solution.**
  - Build a heap for $H$.
  - Apply $n$ EXTRACTMAX.
  - Insert results in the end of array.
  - Return $H$.

  ![Diagram of a binary tree with arrows indicating EXTRACTMAX process]

- **Time.**
  - Heap construction in $\Theta(n)$ time.
  - $n$ EXTRACTMAX in $\Theta(n \log n)$ time.
  - $\Rightarrow$ total time is $\Theta(n \log n)$. 
**Heapsort**

- **Theorem.** We can sort an array in $\Theta(n \log n)$ time.
- Uses only $O(1)$ extra space.
- **In-place** sorting algorithm.
- **Equivalence** of sorting and priority queues.

**Priority Queues**

- Priority Queues
- Trees and Heaps
- Representations of Heaps
- Algorithms on Heaps
- Building a Heap
- Heapsort