On the power of structural violations in priority queues

Jyrki Katajainen (University of Copenhagen)

Joint work with Amr Elmasry (Alexandria University) and Claus Jensen (University of Copenhagen)

These slides are available at http://www.cphstl.dk
Heaps

Examples:
- Fibonacci heaps
- Run-relaxed heaps
- Fat heaps

find-min()  
extract()  
insert()  
decrease(→6)  
delete()
Focus

- comparison complexity of heap operations
- worst-case efficiency
- constant factors

Our ultimate goal is to develop a library component that guarantees optimal complexity bounds, but unfortunately the data structures developed are not practical.
Research question

Q: If find-min, insert, extract, and decrease are required to take $O(1)$ time, can delete be realized in $O(\lg n)$ time including only $\lg n + O(1)$ element comparisons.

A: If decrease is allowed to take $O(\lg n)$ time, yes. With $O(1)$-time decrease almost, but we do not know the final answer.

$n$: # elements stored just prior to an operation
Binomial heaps

\[ n = 1010_{\text{two}} \]

\[ \min \]

\[ B_3 \]

\[ B_1 \]

- heap-ordered \( x \leq y \)
- at most \( \lceil \lg n \rceil + 1 \) binomial trees

\[ B_0 \equiv \]

\[ B_k \equiv \]

\[ B_{k-1} \]

Read [Cormen et al. 2001]
Heap-order violations

- Binomial heaps
- \( O(\lg n) \) violations

\[ \Rightarrow \]

- Run-relaxed heaps

[Driscoll et al. 1988]
Structural violations

Example:
Fibonacci heaps

[Fredman & Tarjan 1987]
Pruned heaps

$\tau$: # of trees

$\lambda$: # of phantom nodes

$\tau \leq O(\lg n)$

$\lambda \leq O(\lg n)$
## Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>find-min</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>insert</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>extract</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>decrease</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>delete</td>
<td>$3 \log_2 n + O(1)$</td>
<td>$2.73 \log_2 n + O(1)$</td>
<td>$\log_2 n + O(\log \log n)$</td>
<td>$\log_2 n + O(\sqrt{\log n})$</td>
</tr>
</tbody>
</table>

# of element comparisons
Find-min

Maintain a pointer to the current minimum
Insert

Imitate ++ for binary numbers

\[
\begin{array}{c}
1 & 1 & 1 \\
+ & & 1 \\
\hline
1 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Diagram 1} \\
\text{Diagram 2} \\
= \\
\text{Diagram 3}
\end{array}
\]
**Extract**

Imitate -- for binary numbers

\[
\begin{array}{c}
1 & 0 & 0 & 0 \\
\hline
1 & 1 & 1 & 1
\end{array}
\]

extract is our contribution for the mankind!
Problem

carries/borrows $\Rightarrow$ Use redundant zeroless number representation

$1000_{\text{two}} \times \Rightarrow 32_{\text{redundant-four}}$

[ISAAC 2006]
Decrease

- cut the subtree $T$
- put a phantom node instead
- make the element replacement
- see $T$ as a separate tree
- reduce the # of phantom nodes if necessary
Data-structural transformations

Singleton transformation I: Both $x$ and $y$ are the last children of their parents $p$ and $q$, respectively. Name the nodes such that $\text{element}[p] \not> \text{element}[q]$.

\[
\begin{align*}
&\begin{array}{c}
\text{f} \\
p
\end{array} & \begin{array}{c}
g \\
q
\end{array} & \\
\begin{array}{c}
B_k \\
x
\end{array} & \begin{array}{c}
B_k \\
y
\end{array} & \begin{array}{c}
B_k \\
q
\end{array} & \begin{array}{c}
B_{k+1}
\end{array}
\end{align*}
\]

$\Rightarrow$

$+ 4$ other transformations, see the proceedings
Delete

- cut the subtree $T$ rooted at $x$
- replace with a phantom node
- remove $x$
- extract a node $y$
- join the subtrees of $T$ and $y$
- make the new tree as a separate tree
- update the minimum pointer if necessary
- reduce the # of phantom nodes if necessary
**Analysis**

**Theorem:** A node can have at most $\lg n + O(\sqrt{\lg n})$ real children

For a proof, see the proceedings

⇒ *delete* performs at most $2\lg n + O(\sqrt{\lg n})$ element comparisons
Two-tier heaps

- Upper store
- Lower store
- Lazy deletions!
- Pointers
- Elements
Mimicking heap-order violations

heap-order violation

structural violation

phantom node

main structure

shadow structure
Main contribution

**Theorem:** Relaxed heaps (heap-order violations) and pruned heaps (structural violations) are equal in power up to $\lg n + O(\lg \lg n)$ element comparisons per delete.
Open problems

- What is the answer to our original research question, i.e. is $\lg n + O(1)$ element comparisons per `delete()` possible or not?
- Are the two types of violations in 1-1 correspondence or not?
- What is the lowest number of element comparisons performed by `delete` for heaps that are efficiently meldable?
- How to implement a worst-case efficient heap in an industry-strength program library?