Worst Case Efficient Data Structures

Gerth Stølting Brodal

BRICS
Department of Computer Science
University of Aarhus
and
Max-Planck-Institut für Informatik
Saarbrücken
Overview

- Priority queues
  - Comparison based data structures
  - Lower bounds for comparison based data structures
  - RAM data structures
  - Parallel data structures
- Partial persistent data structures
Priority Queues

Maintain a set of $n$ elements from a totally ordered universe (say integers) under the operations:

- **FindMin**($Q$)
- **Insert**($Q$, $e$)
- **DeleteMin**($Q$)
- **Delete**($Q$, $e$)*
- **Meld**($Q_1$, $Q_2$)
- **DecreaseKey**($Q$, $e$, $e'$)*

*Assume the location of $e$ is known.
Priority Queues

Many priority queues are based on heap ordered trees, i.e., each node stores an element and the element stored at a node is $\geq$ the element stored at the node’s parent.

The priority queues (heaps) of Williams are based on one heap ordered binary tree.  
⇒ INSERT and DELETEMIN can be performed in $O(\log n)$ time.  

Williams 1964
Linking Heap Ordered Trees

More recent data structures are based on linking heap ordered trees of the same size.

Ex.: Binomial Queues are based on $O(\log n)$ heap ordered trees.

**Theorem** Binomial queues support **INSERT** and **MELD** in amortized constant time and **DELETEMIN** in amortized $O(\log n)$ time.

Vuillemin 1978
**Theorem** INSERT and MELD can be supported in worst case constant time and DELETEMIN in worst case $O(\log n)$ time.

$\leftarrow$ One heap ordered tree.
$\leftarrow$ One linking per INSERT and MELD.
$\leftarrow$ 1, 2 or 3 sons of each rank less than the parent’s rank, + one arbitrary ranked son.
$\leftarrow$ The root has rank 0.

Brodal 1995
**Constant Time DecreaseKey**

**Theorem** Fibonacci heaps support DecreaseKey in amortized constant time and DeleteMin in amortized $O(\log n)$ time.  
Fredman, Tarjan 1984

**Theorem** Relaxed heaps support DecreaseKey in worst case constant time and DeleteMin in worst case $O(\log n)$ time. MELD requires $\Theta(\log n)$ time.  
Driscoll, Gabow, Shrairman, Tarjan 1988

**Theorem** DecreaseKey and MELD can be supported in worst case constant time and DeleteMin in worst case $O(\log n)$ time.  
Brodal 1996

$\Leftarrow O(1)$ heap ordered trees.

$\Leftarrow$ Relaxed heaps.

$\Leftarrow$ A number of invariants to solve the technical dependencies!
## Comparison Based Priority Queues

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Heaps</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binomial Queues</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FindMin</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Meld</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Delete(Min)</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>DecreaseKey</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

*Amortized bounds
Comparison Based Priority Queues

Lower Bounds

**Theorem**  Comparison based sorting requires $\Omega(n \log n)$ comparisons.

$\Rightarrow$ **Insert** or **DELETEMIN** require $\Omega(\log n)$ comparisons.

**Theorem**  If **INSERT** and **DELETE** make $O(t)$ comparisons, then

**FINDMIN** requires $\frac{n}{2^{O(t)}}$ comparisons.

$\Rightarrow$ a doubly linked list is an optimal priority queue implementation!

**Theorem**  If **MELD** makes $o(n)$ comparisons, and **FINDMIN** $O(n^\epsilon)$ comparisons, $\epsilon < 1$, then **DELETE** and **DELETEMIN** require $\Omega(\log n)$ comparisons.

Brodal, Chaudhuri, Radhakrishnan 1996
RAM Priority Queues

**Model:** A unit cost Random Access Machine with word size $w$.

**Word operations:** $+$, shifting, bit-wise boolean operations.

**Elements:** Integers in the range $0..2^w - 1$.

**Operations:**

- $\text{FindMin}(Q)$
- $\text{Insert}(Q, e)$
- $\text{Delete}(Q, e)$
- $\text{Pred}(Q, e)$, $\text{Pred}(\{13, 15, 24, 36, 45, 67\} , 31) = 24$.

**Theorem** The above operations can be performed in $O(\log w)$ time.

van Emde Boas 1977

$\Rightarrow$ an $O(\log \log n)$ priority queue for $w = \log^{O(1)} n$. 

G. S. Brodal: Worst case efficient data structures
## RAM Priority Queues

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FindMin</strong></td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Insert</strong></td>
<td>$O(\log w)$</td>
<td>$O(\log \log n)$</td>
<td>$O(\sqrt{\log n})$</td>
<td>$O(f(n))$</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>$O(\log w)$</td>
<td>$O(\log \log n)$</td>
<td>$O(\sqrt{\log n})$</td>
<td>$O(f(n))$</td>
</tr>
<tr>
<td><strong>Pred</strong></td>
<td>$O(\log w)$</td>
<td>$O(\sqrt{\log n})$</td>
<td>$O\left(\frac{\log n}{f(n)}\right)$</td>
<td>$O\left(\frac{\log n}{\log \log n}\right)$</td>
</tr>
</tbody>
</table>

\[ \log \log n \leq f(n) \leq \sqrt{\log n} \]

*Amortized bounds*
Outline of RAM Priority Queue

Packed search trees of degree $2^{\Theta(f(n))}$ with buffers of delayed INSERT and DELETE operations, supporting INSERT and DELETE in worst case $O(f(n))$ time.

⇦ Two level data structure (van Emde Boas and packed).
⇨ Buffer trees for external memory, Arge 1995.
⇨ List merging in $O(1)$ words, Albers, Hagerup 1992.
⇨ Standard deamortization techniques.

Brodal 1997
Adopting Parallelism to Priority Queues

Question: Is it possible to obtain comparison based priority queues supporting operations in $o(\log n)$ time by using a non-constant number of processors?

Answer: Yes, $O(n)$ processors can support INSERT and DELETEMIN in constant time.

![Priority Queue Diagram]

G. S. Brodal: Worst case efficient data structures
Processor $P_i$ maintains 1, 2 or 3 trees of size $2^i$.

Parallel linking and unlinking of trees.

**Theorem** $O(\log n)$ processors can support INSERT, MELD and DELETEMIN in constant time. An extension of the data structure supports DELETE and DECREASEKEY in constant time too.

Brodal 1996
## Parallel Priority Queues

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FindMin</strong></td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(\log \log n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Insert</strong></td>
<td>$O(1)$</td>
<td>$O(\log \log n)$</td>
<td>$O(\log \log n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>DeleteMin</strong></td>
<td>$O(1)$</td>
<td>$O(\log \log n)$</td>
<td>$O(\log \log n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Meld</strong></td>
<td>$O(1)$</td>
<td>$O(\log \log n)$</td>
<td>$O(\log \log n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>$O(1)$</td>
<td>$O(\log \log n)$</td>
<td>$O(\log \log n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>DecreaseKey</strong></td>
<td>$O(1)$</td>
<td>$O(\log \log n)$</td>
<td>$O(\log \log n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#processors</th>
<th>$n$</th>
<th>$\frac{\log n}{\log \log n}$</th>
<th>$\frac{\log n}{\log \log n}$</th>
<th>$\log n$</th>
<th>$\log n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Array</td>
<td>EREW PRAM</td>
<td>CREW PRAM</td>
<td>Array</td>
<td>CREW PRAM or Array</td>
</tr>
</tbody>
</table>
Parallel $\text{DECREASEKEY}$ operations

Essential to algorithms like Dijkstra’s algorithm for the single-source shortest path problem are the operations

- $\text{DELETEMIN}(Q)$
- $\text{DECREASEKEY}(Q, (e_1, e'_1), \ldots, (e_k, e'_k))$

**Theorem** There exists an EREW PRAM data structure supporting the above operations in constant time, provided $e'_1 \leq e'_2 \leq \cdots \leq e'_k$.

Brodal, Träff, Zaroliagis 1997

Previous parallel data structures only supported parallel $\text{INSERT}$ and $\text{DELETEMIN}$ operations.
The Single-Source Shortest Path Problem

**Theorem** The single-source shortest path problem can be solved by Dijkstra’s algorithm in sequential $O(n \log n + m)$ time by using Fibonacci heaps.  

Fredman, Tarjan 1987

<table>
<thead>
<tr>
<th></th>
<th>Han</th>
<th>Driscoll</th>
<th>Paige</th>
<th>Brodal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pan</td>
<td>Gabow</td>
<td>Shrairman</td>
<td>Tarjan</td>
</tr>
<tr>
<td></td>
<td>Reif</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Time      | $O(n \log^2 n)$ | $O(n \log n)$ | $O(n \log \log n)$ | $O(n)$ | $O(n)$ |
| Work      | $O(n^3 \log n)$ | $O(n \log n + m)$ | $O(n^2)$ | $O(n^{2+\epsilon})$ | $O(m \log n)$ |
| Model     | EREW          | EREW          | CRCW    | CRCW    | CREW   |
Partial Persistence

A data structure is said to be partial persistent if

- old versions are remembered and can be accessed,
- only the latest version can be modified.

Some naive approaches:

- Store a copy of each version of the data structure
  \( \Rightarrow \) space and time overhead per update operation is \( O(n) \).
- Only store the sequence of changes done to the data structure
  \( \Rightarrow \) update steps in constant time and space, but accesses require
  \( O(n) \) time.
## Partial Persistence Techniques

<table>
<thead>
<tr>
<th></th>
<th>Driscoll</th>
<th>Sarnak</th>
<th>Sleator</th>
<th>Tarjan</th>
<th>Brodal</th>
<th>Dietz</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Data structures</th>
<th>pointer based</th>
<th>pointer based</th>
<th>pointer based</th>
<th>arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indegree</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$\log^{O(1)} n$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Access steps</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(\log \log n)$</td>
</tr>
<tr>
<td>Update steps</td>
<td>$O(1)^*$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(\log \log n)$</td>
</tr>
</tbody>
</table>

*amortized, °technique requires the RAM, △expected amortized.
Summary

Comparison based priority queues [WADS’95, SODA’96, NJC’96]
- MELD can be supported in worst case constant time.
- MELD and DECREASEKEY can be supported in worst case constant time.
- A lower bound tradeoff between updating a priority queue and the query time.

RAM priority queues [STACS’97]
- DELETE can be supported in worst case $O(\log \log n)$ time.
-Pred can be supported in $o(\log n)$ time while having $O(\log \log n)$ update time.

Parallel priority queues [SWAT’96, IPPS’97]
- $O(\log n)$ processors can support all operations in constant time.
- Parallel DECREASEKEY in constant time.

Partial persistent data structures [NJC’96]
- Bounded indegree data structures can be made partially persistent in worst case constant time.
**Approximate Dictionary Queries**

\( \text{Dist}_H(u, v) \) is the Hamming distance between two binary strings \( u \) and \( v \) of equal length.

\[
\begin{array}{c}
\text{u : 1 0 0 1 1 0 0 0 1 1 1} \\
v : 1 0 0 0 1 0 1 1 1 1 0 \\
\end{array}
\]

\( \uparrow \uparrow \uparrow \uparrow \uparrow \)

\( \text{Dist}_H(u, v) = 4 \)

- **Dictionary** \( W = \{w_1, w_2, \ldots, w_n\} \), \( w_i \in \{0, 1\}^m \) for \( i = 1, \ldots, n \).
- **Query string** \( \alpha \in \{0, 1\}^m \).
- **Question** Is there any \( w_i \in W \) such that \( \text{Dist}_H(\alpha, w_i) \leq d \)?

<table>
<thead>
<tr>
<th>( d = 1 )</th>
<th>Yao, Yao 1995</th>
<th>Brodal, Gąsieniec 1996</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query time</td>
<td>( O(m \log \log n) )</td>
<td>( O(m) )</td>
</tr>
<tr>
<td>Space</td>
<td>( O(nm \log m) )</td>
<td>( O(nm) )</td>
</tr>
<tr>
<td>Preprocessing time</td>
<td>( O(nm \log m) )</td>
<td>( O(nm) )</td>
</tr>
</tbody>
</table>