Two number systems and one application

Amr Elmasry\textsuperscript{1)} and Claus Jensen\textsuperscript{2)}

Jyrki Katajainen\textsuperscript{1)}

\textsuperscript{1)} University of Copenhagen
\textsuperscript{2)} The Royal Library

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

Representation of a priority queue: An ordered collection of pointer-based perfect binary heaps, $2^{i+1} - 1$ elements for $i = 0, 1, \ldots$

Operations: find-min, insert, borrow (What is this?), delete (Is delete-min missing?), meld

Credit: [Williams 1964]
Number system

Digit set: \( d_i \in \{0, 1, \ldots \} \)

Representation of a number: \( \langle d_0, d_1, \ldots, d_{\ell-1} \rangle \) (\( d_0 \) least significant, \( d_{\ell-1} \neq 0 \))

Weight set: \( \{w_i \mid i \in \{0, 1, \ldots \}\} \)

Decimal value: \( \sum_{i=0}^{\ell-1} d_i \times w_i \)

Operations: increment, decrement, addition, cut, catenation
Some number systems

Decimal: \( d_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \); \( w_i = 10^i \) [al-Khwārizmī 825]

Unary: \( d_i \in \{1\}; w_i = 1 \)

Binary: \( d_i \in \{0, 1\}; w_i = 2^i \)

Redundant binary: \( d_i \in \{0, 1, 2\}; w_i = 2^i \)

Skew binary: \( d_i \in \{0, 1, 2\}; w_i = 2^{i+1} - 1 \)

Regular binary: \( d_i \in \{0, 1, 2\}; w_i = 2^i \); Every string of digits is of the form \( (0 \mid 1 \mid 01^*2)^* \) [Clancy & Knuth 1977]

Zeroless regular: \( d_i \in \{1, 2, 3\}; w_i = 2^i \); Every string of digits is of the form \( (1 \mid 2 \mid 12^*3)^* \) [Brodal 1995]
(1) digit $d_i$ at position $i$
(2) increment
(3) decrement
(4) addition
(5) digit transfer $2w_i + 1 = w_{i+1}$

(6) $d_i = O(1) \ \forall i$

(1) $d_i$ heaps of size $w_i$
(2) insert
(3) borrow
(4) meld
(5) siftdown

(6) $O(\lg n)$ heaps; $n$: #elements

Credit: [Vuillemin 1978]
Canonical skew

A positive integer $n$ is represented as a string $\langle d_0, d_1, \ldots, d_{\ell-1} \rangle$ of digits, least-significant digit first, such that

- $d_i \in \{0, 1, 2\} \ \forall i \in \{0, 1, \ldots, \ell - 1\}$, and $d_{\ell-1} \neq 0$,
- if $d_j = 2$, then $d_i = 0 \ \forall i \in \{0, 1, \ldots, j - 1\}$,
- $w_i = 2^{i+1} - 1 \ \forall i \in \{0, 1, \ldots, \ell - 1\}$, and
- the decimal value of $n$ is $\sum_{i=0}^{\ell-1} d_i w_i$.

**Example:** Unique representation of integer $52_{10}$

<table>
<thead>
<tr>
<th>position $i$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>digit $d_i$</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>weight $w_i$</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>15</td>
<td>31</td>
</tr>
</tbody>
</table>

Credit: [Myers 1983]
Algorithm **increment**$(\langle d_0, d_1, \ldots, d_{\ell-1} \rangle)$

1: let $d_j$ be the first non-zero digit, if any
2: if $d_j$ exists and $d_j = 2$
3: $d_j \leftarrow 0$
4: increase $d_{j+1}$ by 1 (1)
5: else
6: increase $d_0$ by 1 (2)

(1) sift down
(2) Add a new node to the collection
(3) at most 2 digit changes

Extra: Update the min pointer, if necessary

(3) $O(\log n)$ worst-case time;

$n$: #elements

Credit: [Bansal et al. 2003]
Decrement/borrow

Algorithm \textit{decrement}(\langle d_0, d_1, \ldots, d_{\ell-1} \rangle)

1: \textbf{assert} \langle d_0, d_1, \ldots, d_{\ell-1} \rangle \text{ is not empty}
2: let \(d_j\) be the first non-zero digit
3: decrease \(d_j\) by 1 \((1)\)
4: if \(j \neq 0\)
5: \(d_{j-1} \leftarrow 2 \ (2)\)

\textbf{(3)} at most 2 digit changes

\textbf{(1)} Remove the root of the smallest heap
\textbf{(2)} Add its subtrees, if any, to the collection

\textbf{(3)} \(O(1)\) worst-case time

\textbf{Problem:} \ The min pointer may be invalidated!
\textbf{Solution:} \ Swap the root with the next root, or with its own left child before the removal
Delete

(1) Borrow a node

(2) Replace with

(3) Perform siftdown or siftup for

(4) Update the min pointer, if necessary

(5) $O(\lg n)$ worst-case time; $n$: \#elements
### Summary

<table>
<thead>
<tr>
<th>Structure</th>
<th>Array-based binary heap</th>
<th>Canonical skew</th>
<th>Regular skew</th>
</tr>
</thead>
<tbody>
<tr>
<td>find-min</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>insert</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>borrow</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>delete</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>meld</td>
<td>$O(n)$</td>
<td>$O((\log n)^2)$</td>
<td>$O((\log m)^2)$</td>
</tr>
</tbody>
</table>

worst-case performance; $m \leq n$

**Open:** Is faster decrease or meld possible? (Amortized bounds matching those achievable for Fibonacci heaps can be obtained, except for decrease; this is not shown in our paper though.)
Regular skew

Cost of a digit change: $O(j)$ at position $j$

Discretization: Initially, $j$ bricks at position $j$, i.e. $b_j = j$

Digit set: $d_i \in \{0, 1, 2\}$ $\forall i$; when $b_k > 0$, $d_k$ is said to form a wall (1 or 2) of $b_k$ bricks

Incremental digit changes: Remove some bricks from some walls in addition to the normal actions; do not transfer digits across any walls

Representation of a number: Otherwise an integer is represented as in any skew system

Credit: [Carlsson et al. 1988]
Regularity conditions

**Preceding 0:** 02 or 01 or 02 ($d_0$ can be a 2)

**Absorbing 0:** 21*0 or 21*0, this 0 should not be preceding ($d_{\ell-1}$ can be a 2 or 2)

**Critical 0:** If $d_k$ forms a wall, then

(i) $\exists j < k - 1$ such that $d_j = 0$, and $b_k \leq j + 1$, or

(ii) $\exists j < k - 1$ such that $d_j = 0$, $d_j$ is absorbing, and $b_k \leq j$

**Example:** One possible representation of integer $143_{10}$

<table>
<thead>
<tr>
<th>position $i$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>critical 0</td>
<td></td>
<td></td>
<td></td>
<td>$d_1$</td>
<td>$d_2$</td>
<td></td>
</tr>
<tr>
<td>digit $d_i$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>variable $b_i$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Increment/insert

Algorithm \textit{transfer}(⟨d_0,d_1,\ldots,d_{\ell-1}⟩, j)

1: \textbf{assert} \ d_j = 2
2: \quad d_j \leftarrow 0
3: \quad \text{increase} \ d_{j+1} \text{ by } 1 \ (1)
4: \quad b_{j+1} \leftarrow j + 1 \ (2)

(1) Add a heap to the collection
(2) Initiate \textit{siftdown}

Algorithm \textit{increment}(⟨d_0,d_1,\ldots,d_{\ell-1}⟩)

1: \text{let} \ d_k \text{ be the first wall, if any}
2: \text{let} \ d_j \text{ be the first 2 for which } b_j \text{= 0, if any}
3: \textbf{if} \ d_j \text{ exists and (} d_k \text{ does not exist or } j < k \text{)} \textbf{then}
4: \quad \textit{transfer}(⟨d_0,d_1,\ldots,d_{\ell-1}⟩, j) \ (4)
5: \quad \text{reduce} \ b_{j+1} \text{ by } 1 \ (3)
6: \textbf{else}
7: \quad \text{increase} \ d_0 \text{ by } 1 \ (1)

Extra: \textbf{update} the min pointer, if necessary

(3) Advance \textit{siftdown} one level downwards (or do nothing if the heap order has already been reestablished)}
Proof of correctness

Case 1 of 12: $d_j$ is the first digit that equals 2, is not a wall, precedes any wall, $j \neq 0$, $d_{j+1} = 0$, and $d_{j+1}$ is critical. In such a case, a transfer is initiated at $d_j$. Before the operation, $d_{j-1} = 0$. After the operation, $d_j = 0$ and $d_{j+1}$ becomes a wall with $b_{j+1} = j$. At this point, $d_{j-1}$ is the critical 0 for the wall $d_{j+1}$. . . .

Case 2 of 12: $d_j$ is the first digit that equals 2, is not a wall, precedes any wall, $j \neq 0$, $d_{j+1} = 0$, and $d_{j+1}$ is not critical. . . .

Case 3 of 12: $d_j$ is the first digit that equals 2, is not a wall, precedes any wall, $j \neq 0$, and $d_{j+1} = 1$. . . .

Case 4 of 12: $d_0 = 2$, $d_1 = 0$, and $d_1$ is critical. For such a case, the created wall $d_1$ is immediately dismantled. . . .

Case 5 of 12: . . .
# Addition/meld

**Algorithm** \textit{addition}(⟨d₀, \ldots, d_{k-1}⟩, ⟨e₀, \ldots, e_{\ell-1}⟩)

1. assert \( k \leq \ell \)
2. for \( i \in \{0, 1, \ldots, k - 1\} \)
3. repeat \( d_i \) times
4. if \( b_i > 0 \)
5. reduce \( b_i \) to 0 \((1)\)
6. arbitrary-increment(⟨\( e₀, \ldots, e_{\ell-1} \⟩, i) \( (2)\)
7. return ⟨\( e₀, e₁, \ldots, e_{\ell-1} \⟩ \)

\((1)\) Finish \textit{siftdown}, if any  
\((2)\) Add a heap to the collection  
\((3)\) \( O(k) \) digit changes

\(O((\lg m)^2)\) worst-case time; 
\( m, n: \#\text{elements}, m \leq n \)
Further reading

Elmasry, Jensen, and Katajainen, Two skew number systems and one application, *Theory of Computing Systems*, (invited)