Cache-Oblivious Dynamic Dictionaries with Update/Query Tradeoff

Gerth Stølting Brodal
Erik D. Demaine
Jeremy T. Fineman
John Iacono
Stefan Langerman
J. Ian Munro

Result presented at SODA 2010
Dynamic Dictionary

Search(k)

Insert(e)

Delete(k)
I/O Model

[Aggarwal, Vitter 88]

Cost: the number of block transfers (I/Os)
Cache-Oblivious Algorithms
[Frigo, Leiserson, Prokop, Ramachandran 99]

- Algorithms not parameterized by $M$ or $B$
- Analyze in ideal-cache model — I/O model, except optimal replacement policy is assumed
## Cache-Oblivious Dynamic Dictionaries

<table>
<thead>
<tr>
<th>Cache-Aware</th>
<th>Search</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree [BM72]</td>
<td>$\mathcal{O}(\log_B N)$</td>
<td>$\mathcal{O}(\log_B N)$</td>
</tr>
<tr>
<td>Buffered B-tree [BF03]</td>
<td>$\mathcal{O}((1/\epsilon)\log_B N)$</td>
<td>$\mathcal{O}((1/\epsilon B^{1-\epsilon})\log_B N)^*$</td>
</tr>
</tbody>
</table>

### Cache-Oblivious

<table>
<thead>
<tr>
<th>Cache-Oblivious</th>
<th>Search</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO B-tree [BDF-00, BDIW04, BFJ02]</td>
<td>$\mathcal{O}(\log_B N)$</td>
<td>$\mathcal{O}(\log_B N + \ldots)$</td>
</tr>
<tr>
<td>COLA [BFF-CFKN07]</td>
<td>$\mathcal{O}(\log_2 N)$</td>
<td>$\mathcal{O}((1/B)\log_2 N)^*$</td>
</tr>
<tr>
<td>Shuttle Tree [BFF-CFKN07]</td>
<td>$\mathcal{O}(\log_B N)$</td>
<td>$\mathcal{O}(1/B^{\Omega(1/(\log \log B^2))}) \log_B N + \ldots)^*$</td>
</tr>
<tr>
<td>xDict [this paper]</td>
<td>$\mathcal{O}((1/\epsilon)\log_B N)$</td>
<td>$\mathcal{O}((1/\epsilon B^{1-\epsilon})\log_B N)^*$†</td>
</tr>
</tbody>
</table>

* amortized

† assumes $M = \Omega(B^2)$
Building an xDict ($\varepsilon = 1/2$)

$lglgN$ x-boxes of squaring capacities

**Insert**: insert into smallest box
- When a box reaches capacity, **Flush** it and **Batch-Insert** into the next box
- $O((1/\sqrt{B}) \log_B N)$ cost is dominated by largest box
  $\Rightarrow O((1/\sqrt{B}) \log_B N)$

**Search**: search in each x-box
- $O(\log_B N)$ cost is dominated by largest box $O(\log_B N)$
\(x\text{-Box} = \text{dictionary with capacity } x^2\)

\textbf{Batch-Insert}(D,A): insert \(\Theta(x)\) presorted objects

\begin{itemize}
  \item cost \(\mathcal{O}\left(\frac{1}{\sqrt{B}}\log_B x\right)\) per element
\end{itemize}

\textbf{Search}(D,K):

\begin{itemize}
  \item cost is \(\mathcal{O}(\log_B x)\)
\end{itemize}

\textbf{Flush}(D): produce a size-\(x^2\) sorted array \(A\) containing all the elements in the \(x\)-box \(D\)

\begin{itemize}
  \item cost is \(\mathcal{O}(1/B)\) per element
\end{itemize}
Recursive $x$-Box

- **Upper level:** at most $x^{1/2}/4$ subboxes
- **Lower level:** at most $x/4$ subboxes

Subboxes stored contiguously in arbitrary order

Unused (currently empty) subboxes are preallocated
Theorem:  An $x$-Box uses at most $cx^2$ space

(within constant factor of capacity/output buffer)
Fractional Cascading within $x$-Box

Propagate samples upwards + Lookahead pointers
Searching in an x-Box

Describe searches by the recurrence

\[ S(x) = 2S(\sqrt{x}) + O(1) \]

with base case \( S(<\sqrt{B}) = 0 \)

Solves to \( O(\log_B N) \)
Flush

- Moves all real elements to the output buffer in sorted order.
- Lookahead pointers are rebuilt to facilitate searches. Most subboxes remain empty.
Batch-Insert

1. Merge sorted input into input buffer.
1. Merge sorted input into input buffer.
2. If input buffer is “full enough,” Batch-Insert into upper-level subboxes (in chunks of $\Theta(\sqrt{x})$)
1. Merge sorted input into input buffer.
2. If input buffer is “full enough,” Batch-Insert into upper-level subboxes (in chunks of $\Theta(\sqrt{x})$)
3. Whenever a subbox is near capacity, Flush it, then split it into two subboxes
1. **Merge** sorted input into input buffer.
2. If input buffer is “full enough,” **Batch-Insert** into upper-level subboxes (in chunks of \( \Theta(\sqrt{x}) \))
3. Whenever a subbox is near capacity, **Flush** it, then split it into two subboxes
4. If no empty subboxes remain, **Flush** all of them and merge output buffers into middle buffer.
Generalizing to $O((1/\varepsilon B^{1-\varepsilon}) \log_B N)$

Parameterize by $0 < \alpha \leq 1$, where $\alpha = \varepsilon/(1-\varepsilon)$

$1/\varepsilon$ overhead comes from geometric sum in xDict
# Results Summary

<table>
<thead>
<tr>
<th>Cache-Aware</th>
<th>Search</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree [BM72]</td>
<td>$\mathcal{O}(\log_B N)$</td>
<td>$\mathcal{O}(\log_B N)$</td>
</tr>
<tr>
<td>Buffered B-tree [BF03]</td>
<td>$\mathcal{O}\left(\frac{1}{\varepsilon}\log_B N\right)$</td>
<td>$\mathcal{O}\left(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B N\right)^*$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache-Oblivous</th>
<th>Search</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO B-tree [BDF-00, BDIW04, BFJ02]</td>
<td>$\mathcal{O}(\log_B N)$</td>
<td>$\mathcal{O}(\log_B N + \ldots)$</td>
</tr>
<tr>
<td>COLA [BFF-CFKN07]</td>
<td>$\mathcal{O}(\log_2 N)$</td>
<td>$\mathcal{O}\left(\frac{1}{B}\log_2 N\right)^*$</td>
</tr>
<tr>
<td>Shuttle Tree [BFF-CFKN07]</td>
<td>$\mathcal{O}(\log_B N)$</td>
<td>$\mathcal{O}\left(\frac{1}{B^{\Omega\left(\frac{1}{\log\log B^2}\right)}}\log_B N + \ldots\right)^*$</td>
</tr>
<tr>
<td>xDict [this paper]</td>
<td>$\mathcal{O}\left(\frac{1}{\varepsilon}\log_B N\right)$</td>
<td>$\mathcal{O}\left(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B N\right)^*$†</td>
</tr>
</tbody>
</table>

* amortized

† assumes $M = \Omega(B^2)$