Buffer Trees

Computational Geometry
**Pairwise Rectangle Intersection**

**Input**  
$N$ rectangles

**Output**  
all $R$ pairwise intersections

**Example**  
$(A, B) (B, C) (B, F) (D, E) (D, F)$

**Intersection Types**

<table>
<thead>
<tr>
<th>Intersection</th>
<th>Identified by...</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Orthogonal Line Segment Intersection" /></td>
<td>Orthogonal Line Segment Intersection on $4N$ rectangle sides</td>
</tr>
<tr>
<td><img src="image" alt="Batched Range Searching" /></td>
<td>Batched Range Searching on $N$ rectangles and $N$ upper-left corners</td>
</tr>
</tbody>
</table>

**Algorithm**  
Orthogonal Line Segment Intersection + Batched Range Searching + Duplicate removal
Orthogonal Line Segment Intersection

**Input**  \( N \) segments, vertical and horizontal

**Output** all \( R \) intersections

**Sweepline Algorithm**

- Sort all endpoints w.r.t. \( x \)-coordinate
- Sweep left-to-right with a range tree \( T \) storing the \( y \)-coordinates of horizontal segments intersecting the sweepline
- Left endpoint \( \Rightarrow \) insertion into \( T \)
- Right endpoint \( \Rightarrow \) deletion from \( T \)
- Vertical segment \([y_1, y_2]\) \( \Rightarrow \) report \( T \cap [y_1, y_2] \)

Total (internal) time \( O(N \cdot \log_2 N + R) \)
Range Trees

<table>
<thead>
<tr>
<th>Create</th>
<th>Create empty structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert($x$)</td>
<td>Insert element $x$</td>
</tr>
<tr>
<td>Delete($x$)</td>
<td>Delete the inserted element $x$</td>
</tr>
<tr>
<td>Report($x_1, x_2$)</td>
<td>Report all $x \in [x_1, x_2]$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary search trees (internal)</th>
<th>B-trees (# I/Os)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updates</td>
<td>$O(\log_2 N)$</td>
</tr>
<tr>
<td>Report</td>
<td>$O(\log_2 N + R)$</td>
</tr>
</tbody>
</table>

Orthogonal Line Segment Intersection using B-trees

\[ O(\text{Sort}(N) + N \cdot \log_B N + \frac{R}{B}) \] I/Os ...
Batched Range Searching

**Input**  $N$ rectangles and points

**Output**  all $R(r, p)$ where point $p$ is within rectangle $r$

**Sweepline Algorithm**

- Sort all points and left/right rectangle sides w.r.t. $x$-coordinate
- Sweep left-to-right while storing the $y$-intervals of rectangles intersecting the sweepline in a segment tree $T$
- Left side $\Rightarrow$ insert interval into $T$
- Right side $\Rightarrow$ delete interval from $T$
- Point $(x, y) \Rightarrow$ stabbing query : report all $[y_1, y_2]$ where $y \in [y_1, y_2]$

Total (internal) time $O(N \cdot \log_2 N + R)$
Segment Trees

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Create empty structure</td>
</tr>
<tr>
<td>$\text{Insert}(x_1, x_2)$</td>
<td>Insert segment $[x_1, x_2]$</td>
</tr>
<tr>
<td>$\text{Delete}(x_1, x_2)$</td>
<td>Delete the inserted segment $[x_1, x_2]$</td>
</tr>
<tr>
<td>$\text{Report}(x)$</td>
<td>Report the segments $[x_1, x_2]$ where $x \in [x_1, x_2]$</td>
</tr>
</tbody>
</table>

**Assumption** The endpoints come from a fixed set $S$ of size $N + 1$

- Construct a balanced binary tree on the $N$ intervals defined by $S$
- Each node spans an interval and stores a linked list of intervals
- An interval $I$ is stored at the $O(\log N)$ nodes where the node
  intervals $\subseteq I$ but the intervals of the parents are not

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>$O(N \log_2 N)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$O(\log_2 N)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$O(\log_2 N)$</td>
</tr>
<tr>
<td>Report</td>
<td>$O(\log_2 N + R)$</td>
</tr>
</tbody>
</table>
Computational Geometry – Summary

Pairwise Rectangle Intersection
Orthogonal Line Segment Intersection
Batched Range Searching

\[ O(N \cdot \log_2 N + R) \]

Range Trees
Segment Trees

Updates \( O(\log_2 N) \)
Queries \( O(\log_2 N + R) \)
Observations on Range and Segment Trees

- Only inserted elements are deleted, i.e. Delete does not have to check if the elements are present in the structure
- Applications are off-line, i.e. amortized performance is sufficient
- Queries to the range trees and segment trees can be answered lazily, i.e. postpone processing queries until there are sufficient many queries to be handled simultaneously
- Output can be generated in arbitrary order, i.e. batched queries
- The deletion time of a segment in a segment tree is known when the segment is inserted, i.e. no explicit delete operation required
Buffer Trees

On-line Internal $\rightarrow$ “General transformation” $\rightarrow$ Batched External
Buffer Trees

- \((a, b)\)-tree, \(a = m/4\) and \(b = m\)
- Buffer at internal nodes \(m\) blocks
- Buffers contain delayed operations, e.g. \(\text{Insert}(x)\) and \(\text{Delete}(x)\)
- Internal memory buffer containing \(\leq B\) last operations
  Moved to root buffer when full
**Buffer Emptying: Insertions Only**

**Emptying internal node buffers**
- Distribute elements to children
- For each child with more than \( m \) blocks of elements recursively empty buffer

**Emptying leaf buffers**
- Sort buffer
- Merge buffer with leaf blocks
- Rebalance by splitting nodes bottom-up (buffers are now empty)

**Corollary** Optimal sorting by top-down emptying all buffers

\[ O\left(\frac{n}{m}\right) \text{ buffer empty operations per internal level, each of } O(m) \text{ I/Os} \Rightarrow \text{in total } O(\text{Sort}(N)) \text{ I/Os} \]
Priority Queues

- Operations: \texttt{Insert}(x) and \texttt{DeleteMin}
- Internal memory \texttt{min-buffer} containing the $\frac{1}{4}mB$ smallest elements
- Allow nodes on leftmost path to have degree between 1 and $m$
  $\implies$ rebalancing only requires node splittings
- Buffer emptying on leftmost path
  $\implies$ two leftmost leaves contain $\geq mB/4$ elements
- \texttt{Insert} and \texttt{DeleteMin} amortized $O\left(\frac{1}{B} \log_{M/B} \frac{N}{B}\right)$ I/Os

\[
O\left(\log_{m} n\right)
\]
Batched Range Trees

Delayed operations in buffers: $\text{Insert}(x)$, $\text{Delete}(x)$, $\text{Report}(x_1, x_2)$

Assumption: Only inserted elements are deleted
**Definition** A buffer is in *time order representation (TOR)* if

1. Report queries are older than Insert operations and younger than Delete operations
2. Insertions and deletions are in sorted order
3. Report queries are sorted w.r.t. $x_1$

\[
\begin{align*}
\text{Delete: } & \quad x_1, x_2, \ldots \\
\text{Report: } & \quad [x_{11}, x_{12}], [x_{21}, x_{22}], \ldots \\
\text{Insert: } & \quad y_1, y_2, \ldots
\end{align*}
\]
Constructing Time Order Representations

**Lemma**  A buffer of $O(M)$ elements can be made into TOR using $O\left(\frac{M+R}{B}\right)$ I/Os where $R$ is the number of matches reported

**Proof**

- Load buffer into memory
- First *Inserts are shifted up thru time*
  - If $\text{Insert}(x)$ passes $\text{Report}(x_1, x_2)$ and $x \in [x_1, x_2]$ then a match is reported
  - If $\text{Insert}(x)$ meets $\text{Delete}(x)$, then both operations are removed
- *Deletes are shifted down thru time*
  - If $\text{Delete}(x)$ passes $\text{Report}(x_1, x_2)$ and $x \in [x_1, x_2]$ then a match is reported
- Sort Deletions, Reports and Insertion internally
- Output to buffer
Merging Time Order Representations

Lemma  Two list $S_1$ and $S_2$ in TOR where the elements in $S_2$ are older than the elements in $S_1$ can be merged into one time ordered list in $O\left(\frac{|S_1|+|S_2|+R}{B}\right)$ I/Os

Proof

1. Swap $i_2$ and $d_1$ and remove canceling operations
2. Swap $d_1$ and $s_2$ and report matches
3. Swap $i_2$ and $s_1$ and report matches
4. Merge lists

\[\text{Step 1} \quad \text{Step 2} \quad \text{Step 3} \quad \text{Step 4}\]
Emptying All Buffers

**Lemma** Emptying all buffers in a tree takes $O\left(\frac{N+R}{B}\right)$ I/Os

**Proof**

- Make all buffers into time order representation, $O\left(\frac{N+R}{B}\right)$ I/Os
- Merge buffers top-down for complete layers $\Rightarrow$ since layer sizes increase geometrically, #I/Os dominated by size of lowest level, i.e $O\left(\frac{N+R}{B}\right)$ I/Os

Note The tree should be rebalanced afterwards
Emptying Buffer on Overflow

**Invariant**  Emptying a buffer distributes information to children in TOR

1. Load first $m$ blocks in and make TOR and report matches
2. Merge with result from parent in TOR that caused overflow
3. Identify which subtrees are spanned completely by a Report($x_1, x_2$)
4. Empty subtrees identified in 3.
   - Merge with Delete operations
   - Generate output for the range queries spanning the subtrees
   - Merge Insert operations
5. Distribute remaining information to trees not found in 3.
**Batched Range Trees - The Result**

**Rebalancing**  As in \((a, b)\)-trees, except that buffers must be empty. For Fusion and Sharing a forced buffer emptying on the sibling is required, causing \(O(m)\) additional I/Os. Since at most \(O(n/m)\) rebalancing steps done \(\Rightarrow O(n)\) additional I/Os.

**Total #I/Os**  Bounded by generated output \(O(R/B)\), and \(O(1/B)\) I/O for each level an operation is moved down.

**Theorem**  Batched range trees support

<table>
<thead>
<tr>
<th></th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updates</td>
<td>(O\left(\frac{1}{N}\text{Sort}(N)\right)) amortized I/Os</td>
</tr>
<tr>
<td>Queries</td>
<td>(O\left(\frac{1}{N}\text{Sort}(N) + \frac{R}{B}\right)) amortized I/Os</td>
</tr>
</tbody>
</table>
Batched Segment Trees

- Internal node:
  - Partition \( x \)-interval in \( \sqrt{m} \) slabs/Intervals
  - \( O(m) \) multi-slabs defined by continuous ranges of slabs
  - Segments spanning at least one slab (long segment) stored in list associated with largest multi-slab it spans
  - Short segments, as well as ends of long segments, are stored further down the tree
Batched Segment Trees

- Buffer-emptying process in $O(m + \frac{R}{B})$ I/Os:
  - Load buffer — $O(m)$
  - Store long segments from buffer in multi-slab lists — $O(m)$
  - Report “intersections” between queries from buffer and segments in relevant multi-slab lists — $O(\frac{R}{B})$
  - “Push” elements one level down — $O(m)$
Batched Segment Trees

**Theorem** Batched segment trees support

<table>
<thead>
<tr>
<th>Updates</th>
<th>$O\left(\frac{1}{N} \text{Sort}(N)\right)$ amortized I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries</td>
<td>$O\left(\frac{1}{N} \text{Sort}(N) + \frac{R}{B}\right)$ amortized I/Os</td>
</tr>
</tbody>
</table>
Orthogonal Line Segment Intersection

- Sort all endpoints w.r.t. $x$-coordinate
- Sweep left-to-right with a batched range tree $T$
- Left endpoint $\Rightarrow$ insertion into $T$
- Right endpoint $\Rightarrow$ deletion from $T$
- Vertical segment $\Rightarrow$ batched report

\[
\begin{align*}
\text{Sort}(N) & \quad O\left(\frac{N}{B}\right) \\
\text{left endpoint} & \quad O\left(\frac{1}{B} \log \frac{M}{B} \frac{N}{B}\right) \\
\text{right endpoint} & \quad O\left(\frac{1}{B} \log \frac{M}{B} \frac{N}{B} + \frac{R}{B}\right) \\
\text{Vertical segment} & \quad O\left(\text{Sort}(N) + \frac{R}{B}\right) \text{ I/Os}
\end{align*}
\]
Batched Range Searching

- Sort w.r.t. $x$-coordinate
- Sweep left-to-right with a batched segment tree $T$
- Left side $\Rightarrow$ insert interval into $T$
- Right side $\Rightarrow$ delete interval from $T$
- Point $\Rightarrow$ batched stabbing query

\[
\text{Sort}(N) \quad O\left(\frac{N}{B}\right)
\]

\[
\begin{align*}
\text{\{ } & O\left(\frac{1}{B} \log_{M/B} \frac{N}{B}\right) \\
\text{\} } & \quad O\left(\frac{1}{B} \log_{M/B} \frac{N}{B} + \frac{R}{B}\right) \\
\end{align*}
\]

\[
O(\text{Sort}(N) + \frac{R}{B}) \text{ I/Os}
\]
Pairwise Rectangle Intersection

Orthogonal line segment intersection  Batched range searching  Duplicate removal

4N rectangle sides  N rectangles and N upper-left corners

Trick  Only generate one intersection between two rectangles

\[ O(\text{Sort}(N) + \frac{R}{B}) \text{ I/Os} \]
Buffer Tree Applications – Summary

Pairwise Rectangle Intersection
Orthogonal Line Segment Intersection
Batched Range Searching

Batched Range Trees
Batched Segment Trees
Priority Queues

\[
\begin{align*}
\text{Updates} & \quad O\left(\frac{1}{N}\text{Sort}(N)\right) \\
\text{Queries} & \quad O\left(\frac{1}{N}\text{Sort}(N) + \frac{R}{B}\right)
\end{align*}
\]