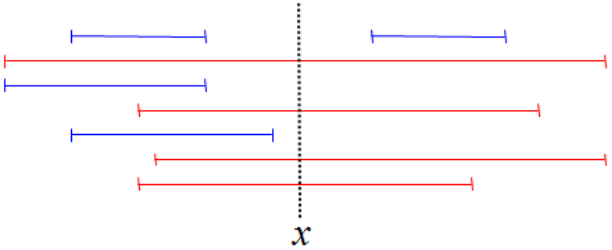
Massive Data Algorithmics

Lecture 6: Interval Trees

Interval Management

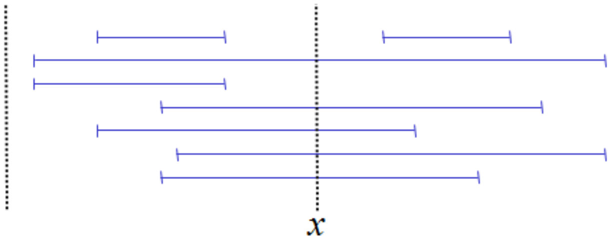
- Problem:
  - Maintain N intervals with unique endpoints dynamically such that stabbing query with point x can be answered efficiently



- As in (one-dimensional) B-tree case we are interested in
  - $O(N/B)$ space
  - $O(\log_B N)$ update
  - $O(\log_B N + T/B)$

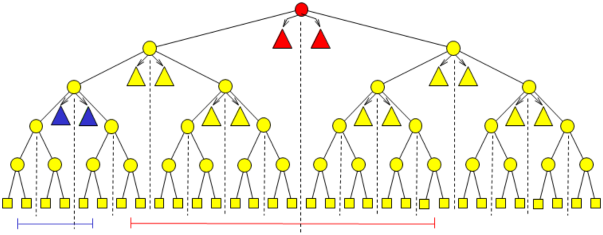
Interval Management: Static Solution

- Sweep from left to right maintaining persistent B-tree
  - Insert interval when left endpoint is reached
  - Delete interval when right endpoint is reached



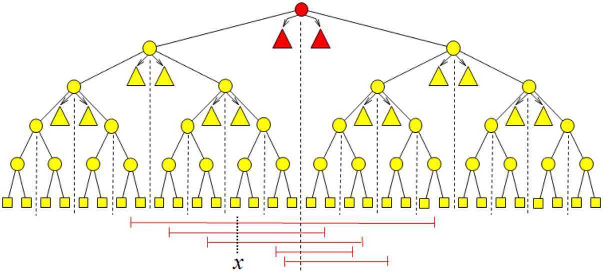
- Query x answered by reporting all intervals in B-tree at time x
  - $O(N/B)$ space
  - $O(\log_B N)$ update
  - $O(\log_B N + T/B)$

Internal Interval Trees



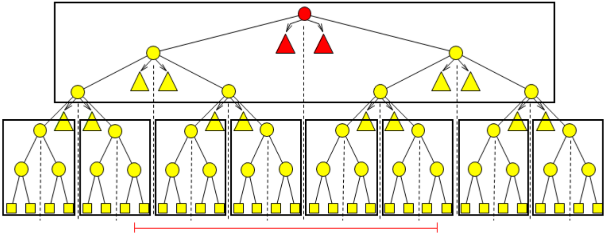
- Base tree on endpoints slab X_v associated with each node v
- Interval stored in highest node v where it contains midpoint of X_v
- Intervals I_v associated with v stored in
  - Left slab list sorted by left endpoint (search tree)
  - Right slab list sorted by right endpoint (search tree)
- Linear space and $O\log n)$ update

Internal Interval Trees



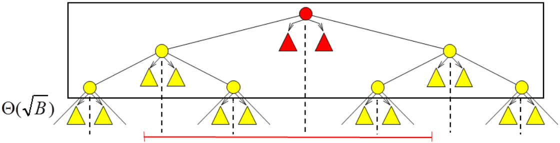
- Query with x on left side of midpoint of X_{root}
  - Search left slab list left-right until finding non-stabbed interval
  - Recurse in left child
- $\Rightarrow O(\log N + T)$ query bound

Externalizing Interval Tree



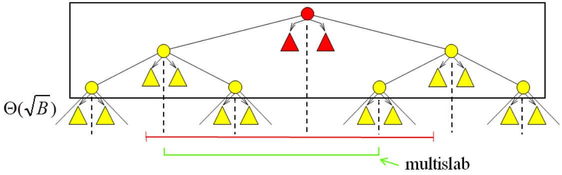
- Natural idea:
  - Block tree
  - Use B-tree for slab lists
- Number of stabbed intervals in large slab list may be small (or zero)
  - We can be forced to do I/O in each of $O(\log N)$ nodes

Externalizing Interval Tree



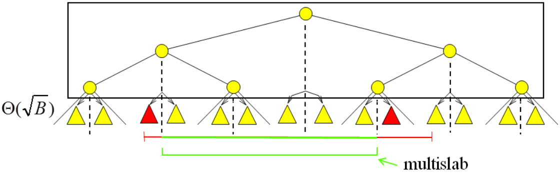
- Idea:
  - Decrease fan-out to $\Theta(\sqrt{B}) \Rightarrow$ height remains $O(\log_B N)$
  - $\Theta(\sqrt{B})$ slabs define $\Theta(B)$ multislabs
  - Interval stored in two slab lists (as before) and one multislab list
  - Intervals in small multislab lists collected in underflow structure
  - Query answered in v by looking at 2 slab lists and not $O(\log B)$

Externalizing Interval Tree



- Idea:
  - Decrease fan-out to $\Theta(\sqrt{B}) \Rightarrow$ height remains $O(\log_B N)$
  - $\Theta(\sqrt{B})$ slabs define $\Theta(B)$ multislabs
  - Interval stored in two slab lists (as before) and one multislab list
  - Intervals in small multislab lists collected in underflow structure
  - Query answered in v by looking at 2 slab lists and not $O(\log B)$

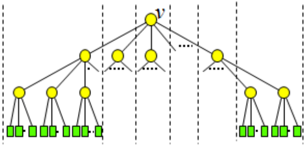
Externalizing Interval Tree



- Idea:
  - Decrease fan-out to $\Theta(\sqrt{B}) \Rightarrow$ height remains $O(\log_B N)$
  - $\Theta(\sqrt{B})$ slabs define $\Theta(B)$ multislabs
  - Interval stored in two slab lists (as before) and one multislab list
  - Intervals in small multislab lists collected in underflow structure
  - Query answered in v by looking at 2 slab lists and not $O(\log B)$

Externalizing Interval Tree

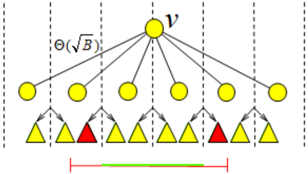
- Base tree: Weight-balanced B-tree with branching parameter $1/4\sqrt{B}$ and leaf parameter B on endpoints
  - Interval stored in highest node v where it contains slab boundary
- Each internal node v contains:
  - Left slab list for each of $\Theta(\sqrt{B})$ slabs
  - Right slab list for each of $\Theta(\sqrt{B})$ slabs
  - $\Theta(B)$ multislab lists



- Interval in set I_v of intervals associated with v stored in
  - Left slab list of slab containing left endpoint
  - Right slab list of slab containing right endpoint
  - Widest multislab list it spans
- If $< B$ intervals in multislab list they are instead stored in underflow structure ($\Rightarrow$ contains $= B^2$ intervals)

Externalizing Interval Tree

- Base tree: Weight-balanced B-tree with branching parameter $1/4\sqrt{B}$ and leaf parameter B on endpoints
  - Interval stored in highest node v where it contains slab boundary
- Each internal node v contains:
  - Left slab list for each of $\Theta(\sqrt{B})$ slabs
  - Right slab list for each of $\Theta(\sqrt{B})$ slabs
  - $\Theta(B)$ multislab lists



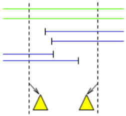
- Interval in set I_v of intervals associated with v stored in
  - Left slab list of slab containing left endpoint
  - Right slab list of slab containing right endpoint
  - Widest multislab list it spans
- If $< B$ intervals in multislab list they are instead stored in underflow structure ($\Rightarrow$ contains $= B^2$ intervals)

Externalizing Interval Tree

- Each leaf contains $< B/2$ intervals (unique endpoint assumption)
  - Stored in one block
- Slab lists implemented using B-trees
  - $O(1 + T_v/B)$ query
  - Linear space
    * We may wasted a block for each of the $\Theta(\sqrt{B})$ lists in node
    * But only $\Theta(\frac{N}{B\sqrt{B}})$ internal nodes
- Underflow structure implemented using static structure
  - $O(\log_B B^2 + T_v/B) = O(1 + T_v/B)$ query
  - Linear space
- Linear space

Externalizing Interval Tree

- Query with x
  - Search down tree for x while in node v reporting all intervals in I_v stabbed by x
- In node v

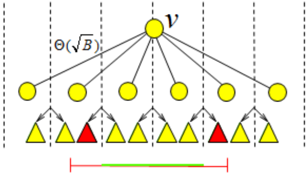


  - Query two slab lists
  - Report all intervals in relevant multislab lists
  - Query underflow structure
- Analysis:
  - Visit $O(\log_B N)$ nodes
  - Query slab lists $O(1+T_v/B)$
  - Query multislab lists $O(1+T_v/B)$
  - Query underflow structure $O(1+T_v/B)$

$\Rightarrow O(\log_B N + T_v/B)$

Externalizing Interval Tree

- Update ignoring base tree update/rebalancing:
  - Search for relevant node: $O(\log_B N)$
  - Update two slab lists: $O(\log_B N)$
  - Update multislab list or underflow structure



- Update of underflow structure in $O(1)$ I/Os amortized:
  - Maintain update block with $\leq B$ updates
  - Check of update block adds O(1) I/Os to query bound
  - Rebuild structure when B updates have been collected using $O(B^2/B\log_B B^2) = O(B)$ I/Os (Global Rebuilding)
- $\Rightarrow$ Update in $O(\log_B N)$ I/Os amortized

Externalizing Interval Tree

- Note:
  - Insert may increase number of intervals in underflow structure for some multislab to B
  - Delete may decrease number of intervals in multislab to B

$\Rightarrow$ Need to move B intervals to/from multislab/underflow structure

- We only move
  - Intervals from multislab list when decreasing to size $B/2$
  - Intervals to multislab list when increasing to size B

$\Rightarrow$ O(1) I/Os amortized used to move intervals

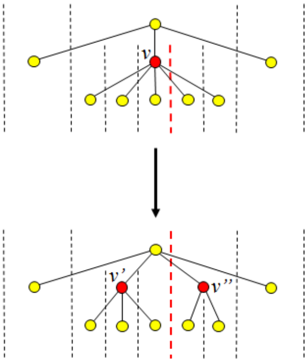
Base Tree Update

- Before inserting new interval we insert new endpoints in base tree using $O(\log_B N/B)$ I/Os

  - Leads to rebalancing using splits

    $\Rightarrow$ Boundary in v becomes boundary in parent(v)

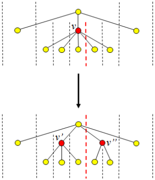
    $\Rightarrow$ Intervals need to be moved



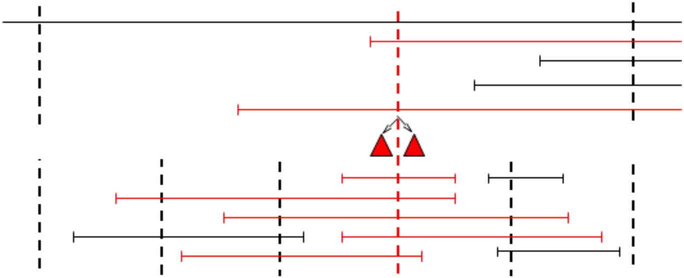
- Move intervals (update secondary structures) in $O(w(v))$ I/Os
  $\Rightarrow$ $O(1)$ amortized split bound (weight balanced B-tree)
  $\Rightarrow$ $O(\log_B N/B)$ amortized insert bound

Splitting Interval Tree Node

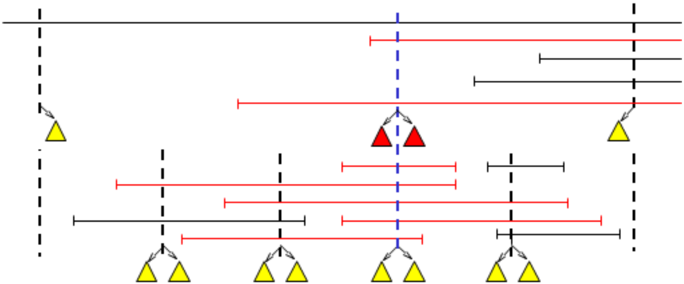
- When v splits we may need to move $O(w(v))$ intervals
  - Intervals in v containing boundary
  - Intervals in parent(v) with endpoints in X_v containing boundary



- Intervals move to two new slab and multislab lists in parent(v)

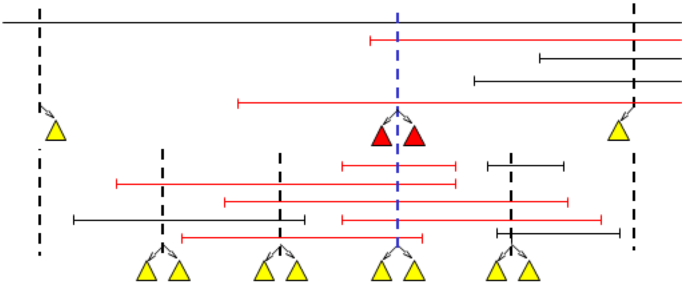


Splitting Interval Tree Node



- Moving intervals in v in $O(w(v))$ I/Os
  - Collected in left order (and remove) by scanning left slab lists
  - Collected in right order (and remove) by scanning right slab lists
  - Removed multislab lists containing boundary
  - Remove from underflow structure by rebuilding it
  - Construct lists and underflow structure for v and v similarly

Splitting Interval Tree Node



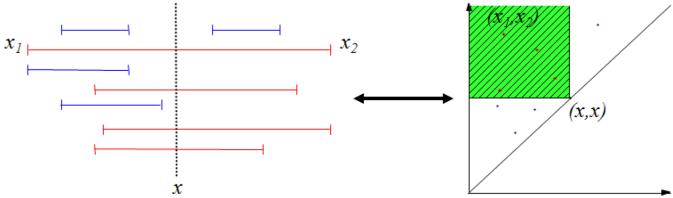
- Moving intervals in parent(v) in $O(w(v))$ I/Os
  - Collect in left order by scanning left slab list
  - Collect in right order by scanning right slab list
  - Merge with intervals collected in $v \Rightarrow$ two new slab lists
  - Construct new multislab lists by splitting relevant multislab list
  - Insert intervals in small multislab lists in underflow structure

Splitting Interval Tree Node

- Split in $O(1)$ I/Os amortized
  - Space: $O(N/B)$
  - Query: $O(\log_B N/B + T/B)$
  - Insert: $O(\log_B N/B)$ I/Os amortized
- Deletes in $O(\log_B N/B)$ I/Os amortized using global rebuilding:
  - Delete interval as previously using $O(\log_B N/B)$ I/Os
  - Mark relevant endpoint as deleted
  - Rebuild structure in $O(\log_B N/B)$ after $N/2$ deletes
- Note: Deletes can also be handled using fuse operations

Summary/Conclusion: Interval Management

- Interval management corresponds to simple form of $2d$ range search
  - Diagonal corner queries
- We obtained the same bounds as for the $1d$ case
  - Space: $O(N/B)$
  - Query: $O(\log_B N/B + T/B)$
  - Update: $O(\log_B N/B)$



Summary/Conclusion: Interval Management

- Main problem in designing structure:
  - Binary $\rightarrow$ large fan-out
- Large fan-out resulted in the need for
  - Multislabs and multislab lists
  - Underflow structure to avoid O(B)-cost in each node
- General solution techniques:
  - Filtering: Charge part of query cost to output
  - Bootstrapping:
    * Use $O(B^2)$ size structure in each internal node
    * Constructed using persistence
    * Dynamic using global rebuilding
  - Weight-balanced B-tree: Split/fuse in amortized $O(1)$