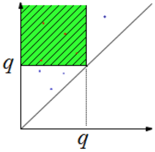
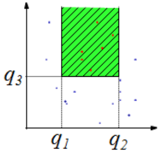
Massive Data Algorithmics

Lecture 8: Range Searching

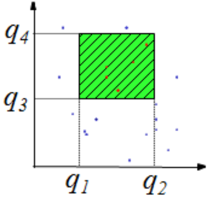
Range Searching

- We have now discussed structures for special cases of two-dimensional range searching
  - Space: $O(N/B)$
  - Query: $O(\log_B N + T/B)$
  - Updates: $O(\log_B N)$





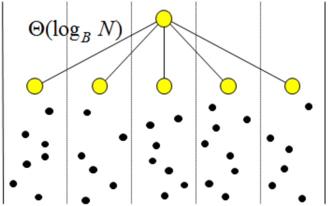
- Cannot be obtained for general (4-sided) $2d$ range searching:



  - $O(\log_B^c N)$ query requires $\Omega(\frac{N}{B}\frac{\log_B N}{\log_B \log_B N})$ space
  - $O(N/B)$ space requires $\Omega(\sqrt{N/B})$ query

External Range Tree

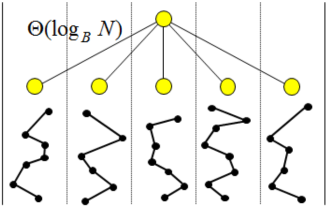
- Base tree: Weight balanced tree with branching parameter $\Theta(\log_B N)$ and leaf parameter B on x-coordinates
  $\Rightarrow O(\log_{log_B N} N) = O(\frac{\log_B N}{\log_B \log_B N})$ height
- Points below each node stored in 4 linear space secondary structures:
  - Right priority search tree
  - Left priority search tree
  - B-tree on y-coordinates
  - Interval (priority search) tree



$\Downarrow$
$O(\frac{N}{B} \frac{\log_B N}{\log_B \log_B N})$

External Range Tree

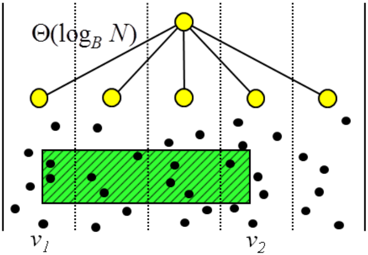
- Secondary interval tree:
- Points below each node stored in 4 linear space secondary structures:
  - Connect points in each slab in y-order
  - Project obtained segments in y-axis



  - Intervals stored in interval tree
    * Interval augmented with pointer to corresponding points in y-coordinate B-tree in corresponding child node

External Range Tree

- Query with (q_1, q_2, q_3, q_4) answered in top node with q_1 and q_2 in different slabs v_1 and v_2



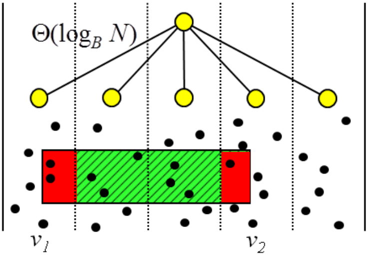
- Found with 3-sided query in v_1 using right priority search tree
- Found with 3-sided query in v_2 using left priority search tree

- Points in slabs between v_1 and v_2
  - Answer stabbing query with q_3 using interval tree $\Rightarrow$ first point above q_3 in each of the $O(\log_B N)$ slabs
  - Find points using y-coordinate B-tree in $O(\log_B N)$ slabs

External Range Tree

- Query with (q_1, q_2, q_3, q_4) answered in top node with q_1 and q_2 in different slabs v_1 and v_2

- Found with 3-sided query in v_1 using right priority search tree
- Found with 3-sided query in v_2 using left priority search tree

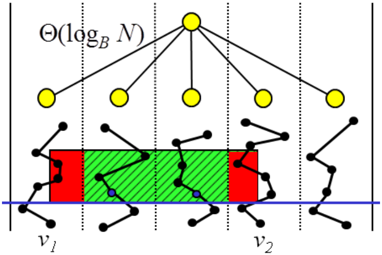


- Points in slabs between v_1 and v_2
  - Answer stabbing query with q_3 using interval tree $\Rightarrow$ first point above q_3 in each of the $O(\log_B N)$ slabs
  - Find points using y-coordinate B-tree in $O(\log_B N)$ slabs

External Range Tree

- Query with (q_1, q_2, q_3, q_4) answered in top node with q_1 and q_2 in different slabs v_1 and v_2

- Found with 3-sided query in v_1 using right priority search tree
- Found with 3-sided query in v_2 using left priority search tree

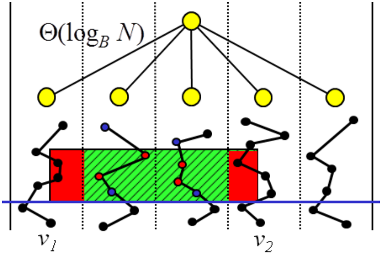


- Points in slabs between v_1 and v_2
  - Answer stabbing query with q_3 using interval tree $\Rightarrow$ first point above q_3 in each of the $O(\log_B N)$ slabs
  - Find points using y-coordinate B-tree in $O(\log_B N)$ slabs

External Range Tree

- Query with (q_1, q_2, q_3, q_4) answered in top node with q_1 and q_2 in different slabs v_1 and v_2

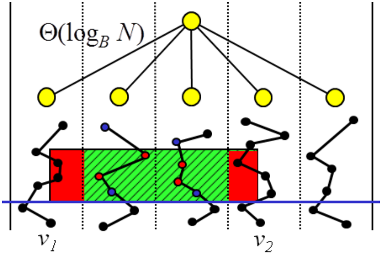
- Found with 3-sided query in v_1 using right priority search tree
- Found with 3-sided query in v_2 using left priority search tree



- Points in slabs between v_1 and v_2
  - Answer stabbing query with q_3 using interval tree $\Rightarrow$ first point above q_3 in each of the $O(\log_B N)$ slabs
  - Find points using y-coordinate B-tree in $O(\log_B N)$ slabs

- Query analysis:
  - $O(\log_B N)$ I/Os to find relevant node
  - $O(\log_B N + T/B)$ I/Os to answer two 3-sided queries
  - $O(\log_B N + \log_B N/B) = O(\log_B N)$ I/Os to query interval tree
  - $O(\log_B N + T/B)$ I/Os to traverse $O(\log_B N)$ B-trees

$\Rightarrow O(\log_B N + T/B)$ I/Os



External Range Tree

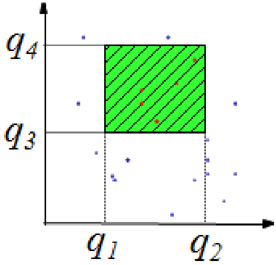
- Insert:
  - Insert x-coordinate in weight-balanced B-tree
    - Split of v can be performed in $O(w(v) \log_B w(v))$ I/Os

  $\Rightarrow O(\frac{\log_B^2 N}{\log_B \log_B N})$ I/Os
  - Update secondary structures in all $O(\frac{\log_B N}{\log_B \log_B N})$ nodes on one root-leaf path
    - Update priority search trees
    - Update interval tree
    - Update B-tree

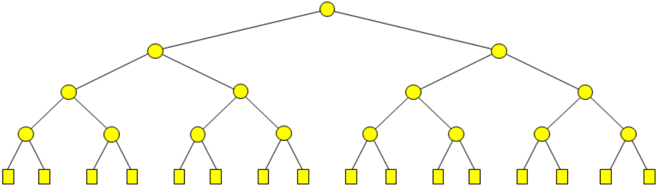
  $\Rightarrow O(\frac{\log_B^2 N}{\log_B \log_B N})$ I/Os
- Delete:
  - Similar and using global rebuilding

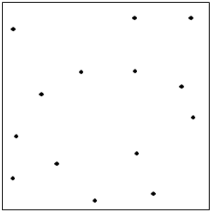
Summary: External Range Tree

- 2d range searching in $O(\frac{N}{B}\frac{\log_B N}{\log_B \log_B N})$ space
  - $O(\log_B N + T/B)$ I/O query
  - $O(\frac{\log_B^2 N}{\log_B \log_B N})$ update
- Optimal among $O(\log_B N + T/B)$ query structures



kdB-tree

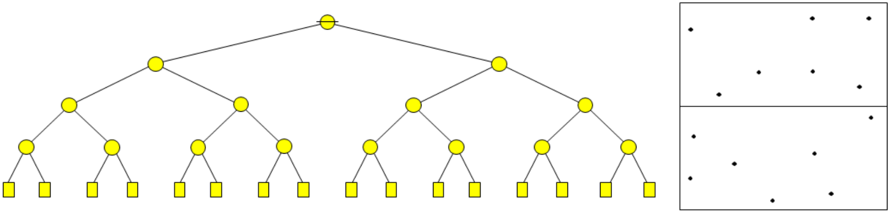




- kd-tree
  - Recursive subdivision of point-set into two half using vertical/horizontal line
  - Horizontal line on even levels, vertical on uneven levels
  - One point in each leaf

$\Rightarrow$ Linear space

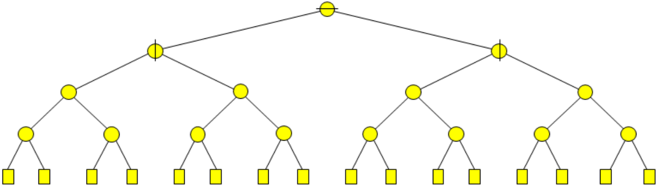
kdB-tree

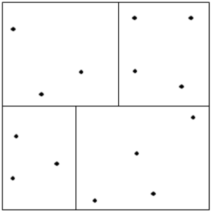


- kd-tree
  - Recursive subdivision of point-set into two half using vertical/horizontal line
  - Horizontal line on even levels, vertical on uneven levels
  - One point in each leaf

⇒ Linear space

kdB-tree

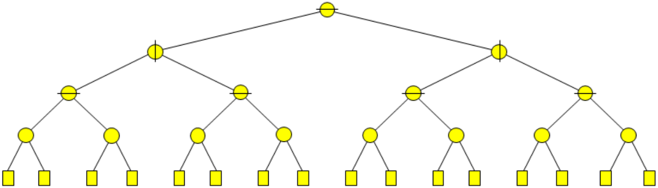


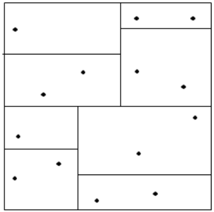


- kd-tree
  - Recursive subdivision of point-set into two half using vertical/horizontal line
  - Horizontal line on even levels, vertical on uneven levels
  - One point in each leaf

$\Rightarrow$ Linear space

kdB-tree

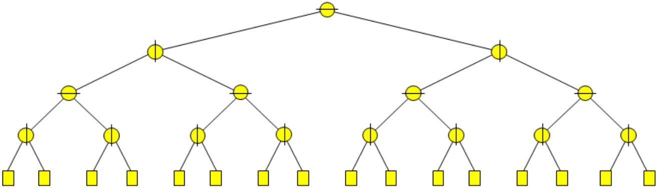


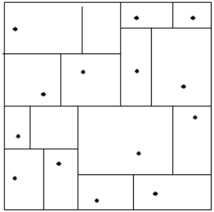


- kd-tree
  - Recursive subdivision of point-set into two half using vertical/horizontal line
  - Horizontal line on even levels, vertical on uneven levels
  - One point in each leaf

$\Rightarrow$ Linear space

kdB-tree

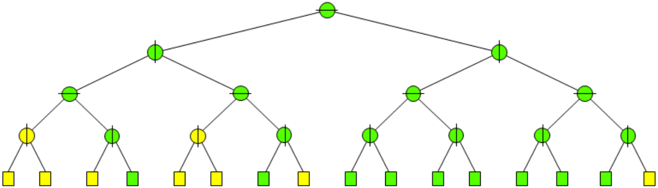


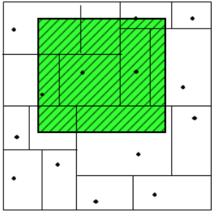


- kd-tree
  - Recursive subdivision of point-set into two half using vertical/horizontal line
  - Horizontal line on even levels, vertical on uneven levels
  - One point in each leaf

$\Rightarrow$ Linear space

kdB-tree

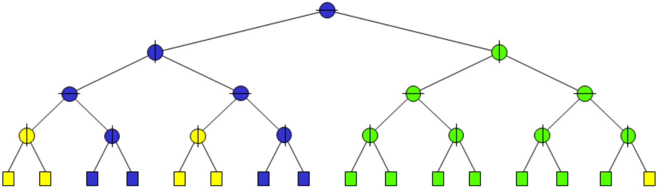


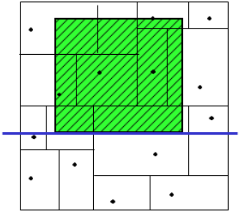


- Query
  - Recursively visit nodes corresponding to regions intersecting query
  - Report point in trees/nodes completely contained in query
- Query analysis
  - Horizontal line intersect $Q(N) = 2 + 2Q(N/4) = O(\sqrt{N})$ regions
  - Query covers T regions

$\Rightarrow O(\sqrt{N} + T)$ I/Os

kdB-tree

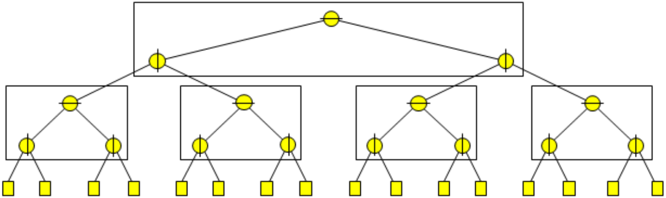


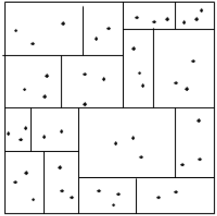


- Query
  - Recursively visit nodes corresponding to regions intersecting query
  - Report point in trees/nodes completely contained in query
- Query analysis
  - Horizontal line intersect $Q(N) = 2 + 2Q(N/4) = O(\sqrt{N})$ regions
  - Query covers T regions

$\Rightarrow O(\sqrt{N} + T)$ I/Os

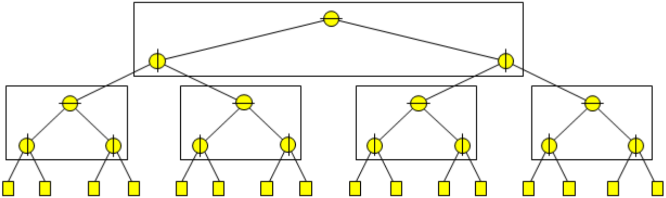
kdB-tree

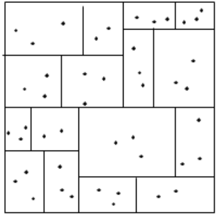




- kdB-tree:
  - Stop subdivision when leaf contains between B/2 and B points
  - BFS-blocking of internal nodes
- Query as before
  - Analysis as before but each region now contains $\Theta(B)$ points $\Rightarrow O(\sqrt{N/B}+T/B)$ I/Os

Construction of kdB-tree



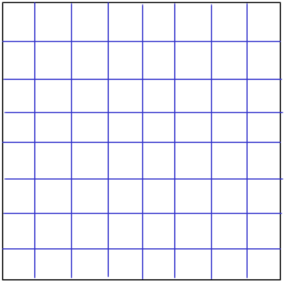


- Simple $O(N/B \log_2 N/B)$ algorithm
  - Find median of y-coordinates (construct root)
  - Distribute point based on median
  - Recursively build subtrees
  - Construct BFS-blocking top-down
- Idea in improved $O(N/B \log_{M/B} N/B)$ algorithm
  - Construct $\sqrt{M/B}$ levels at a time using O(N/B) I/Os

Construction of kdB-tree

- Sort N points by x- and by y-coordinates using $O(N/B \log_{M/B} N/B)$ I/Os
- Building $\sqrt{M/B}$ levels ($\sqrt{M/B}$ nodes) in $O(N/B)$ I/Os:

1. Construct $\sqrt{M/B} \times \sqrt{M/B}$ grid with $O(N/\sqrt{M/B})$ points in each slab
2. Count number of points in each grid cell and store in memory
3. Find slab s with median x-coordinate



4. Scan slab s to find median x-coordinate and construct node
5. Split slab containing median x-coordinate and update counts
6. Recurse on each side of median x-coordinate using grid (step 3)

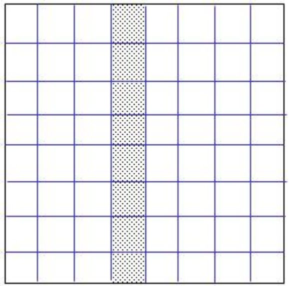
$\Rightarrow$ Grid grows to $M/B + \sqrt{M/B}.\sqrt{M/B} = \Theta(M/B)$ during algorithm

$\Rightarrow$ Each node constructed in $O(N(\sqrt{M/B}.B)$ I/Os

Construction of kdB-tree

- Sort N points by x- and by y-coordinates using $O(N/B\log_{M/B} N/B)$ I/Os
- Building $\sqrt{M/B}$ levels ($\sqrt{M/B}$ nodes) in $O(N/B)$ I/Os:

1. Construct $\sqrt{M/B} \times \sqrt{M/B}$ grid with $O(N/\sqrt{M/B})$ points in each slab
2. Count number of points in each grid cell and store in memory
3. Find slab s with median x-coordinate



4. Scan slab s to find median x-coordinate and construct node
5. Split slab containing median x-coordinate and update counts
6. Recurse on each side of median x-coordinate using grid (step 3)

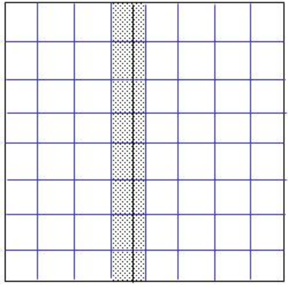
$\Rightarrow$ Grid grows to $M/B + \sqrt{M/B}.\sqrt{M/B} = \Theta(M/B)$ during algorithm

$\Rightarrow$ Each node constructed in $O(N(\sqrt{M/B}.B)$ I/Os

Construction of kdB-tree

- Sort N points by x- and by y-coordinates using $O(N/B \log_{M/B} N/B)$ I/Os
- Building $\sqrt{M/B}$ levels ($\sqrt{M/B}$ nodes) in $O(N/B)$ I/Os:

1. Construct $\sqrt{M/B} \times \sqrt{M/B}$ grid with $O(N/\sqrt{M/B})$ points in each slab
2. Count number of points in each grid cell and store in memory
3. Find slab s with median x-coordinate



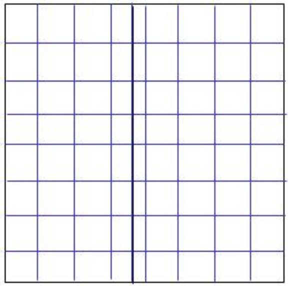
4. Scan slab s to find median x-coordinate and construct node
5. Split slab containing median x-coordinate and update counts
6. Recurse on each side of median x-coordinate using grid (step 3)

$\Rightarrow$ Grid grows to $M/B + \sqrt{M/B}.\sqrt{M/B} = \Theta(M/B)$ during algorithm
$\Rightarrow$ Each node constructed in $O(N(\sqrt{M/B}.B)$ I/Os

Construction of kdB-tree

- Sort N points by x- and by y-coordinates using $O(N/B\log_{M/B}N/B)$ I/Os
- Building $\sqrt{M/B}$ levels ($\sqrt{M/B}$ nodes) in $O(N/B)$ I/Os:

1. Construct $\sqrt{M/B}\times\sqrt{M/B}$ grid with $O(N/\sqrt{M/B})$ points in each slab
2. Count number of points in each grid cell and store in memory
3. Find slab s with median x-coordinate



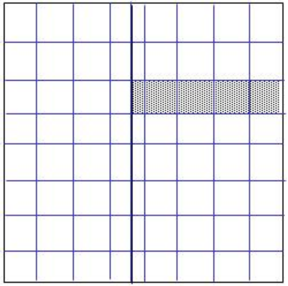
4. Scan slab s to find median x-coordinate and construct node
5. Split slab containing median x-coordinate and update counts
6. Recurse on each side of median x-coordinate using grid (step 3)

$\Rightarrow$ Grid grows to $M/B+\sqrt{M/B}.\sqrt{M/B}=\Theta(M/B)$ during algorithm
$\Rightarrow$ Each node constructed in $O(N(\sqrt{M/B}.B)$ I/Os

Construction of kdB-tree

- Sort N points by x- and by y-coordinates using $O(N/B \log_{M/B} N/B)$ I/Os
- Building $\sqrt{M/B}$ levels ($\sqrt{M/B}$ nodes) in $O(N/B)$ I/Os:

1. Construct $\sqrt{M/B} \times \sqrt{M/B}$ grid with $O(N/\sqrt{M/B})$ points in each slab
2. Count number of points in each grid cell and store in memory
3. Find slab s with median x-coordinate



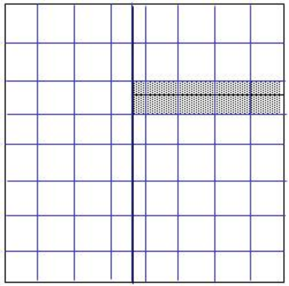
4. Scan slab s to find median x-coordinate and construct node
5. Split slab containing median x-coordinate and update counts
6. Recurse on each side of median x-coordinate using grid (step 3)

$\Rightarrow$ Grid grows to $M/B + \sqrt{M/B}.\sqrt{M/B} = \Theta(M/B)$ during algorithm
$\Rightarrow$ Each node constructed in $O(N(\sqrt{M/B}.B)$ I/Os

Construction of kdB-tree

- Sort N points by x- and by y-coordinates using $O(N/B \log_{M/B} N/B)$ I/Os
- Building $\sqrt{M/B}$ levels ($\sqrt{M/B}$ nodes) in $O(N/B)$ I/Os:

1. Construct $\sqrt{M/B} \times \sqrt{M/B}$ grid with $O(N/\sqrt{M/B})$ points in each slab
2. Count number of points in each grid cell and store in memory
3. Find slab s with median x-coordinate

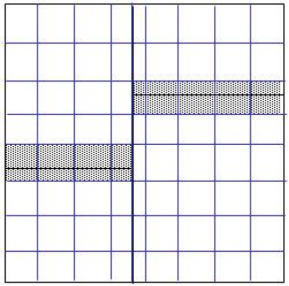


4. Scan slab s to find median x-coordinate and construct node
5. Split slab containing median x-coordinate and update counts
6. Recurse on each side of median x-coordinate using grid (step 3)

$\Rightarrow$ Grid grows to $M/B + \sqrt{M/B}.\sqrt{M/B} = \Theta(M/B)$ during algorithm
$\Rightarrow$ Each node constructed in $O(N(\sqrt{M/B}.B)$ I/Os

Construction of kdB-tree

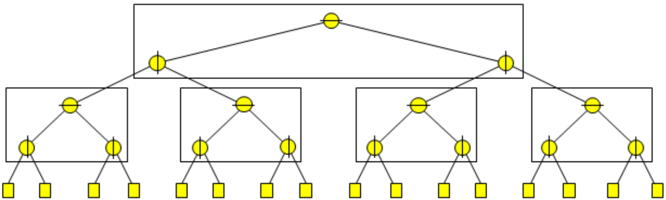
- Sort N points by x- and by y-coordinates using $O(N/B \log_{M/B} N/B)$ I/Os
- Building $\sqrt{M/B}$ levels ($\sqrt{M/B}$ nodes) in $O(N/B)$ I/Os:

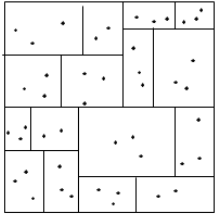


1. Construct $\sqrt{M/B} \times \sqrt{M/B}$ grid with $O(N/\sqrt{M/B})$ points in each slab
2. Count number of points in each grid cell and store in memory
3. Find slab s with median x-coordinate
4. Scan slab s to find median x-coordinate and construct node
5. Split slab containing median x-coordinate and update counts
6. Recurse on each side of median x-coordinate using grid (step 3)

$\Rightarrow$ Grid grows to $M/B + \sqrt{M/B}.\sqrt{M/B} = \Theta(M/B)$ during algorithm
$\Rightarrow$ Each node constructed in $O(N(\sqrt{M/B}.B)$ I/Os

kdB-tree

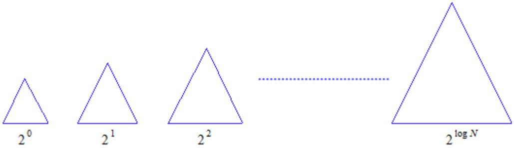




- kdB-tree:
  - Linear space
  - Query in $O(\sqrt{N/B}+T/B)$ I/Os
  - Construction in $O(N/B\log_{M/B}N/B)$ I/Os
  - Point search in $O(\log_B N)$ I/Os
- Dynamic:
  - Deletions relatively easily in $O(\log_B^2 N)$ I/Os (partial rebuilding)

kdB-tree Insertion using Logarithmic Method

- Partition pointset S into subsets $S_0, \cdots, S_{logN}$, $|S_i| = 2^i$ or $|S_i| = 0$
- Build kdB-tree D_i on S_i



- Query: Query each $D_i \Rightarrow \sum_{i=0}^{\log N} O(\sqrt{2^i/B} + T_i/B) = O(\sqrt{N/B} + T/B)$
- Insert: Find first empty D_i and construct D_i out of $1 + \sum_{j=0}^{i-1} 2^j = 2^i$ elements in $S_0, S_1, \cdots, S_{i-1}$
  - $O(2^i/B \log_{M/B}(N/B \log N)$ I/Os $\Rightarrow O(1/B \log_{M/B} N/B)$ per moved point
  - Point moved $O(logN)$ times

$\Rightarrow O(1/B \log_{M/B}(N/B \log N)) = O(\log_B^2 N)$ I/Os amortized

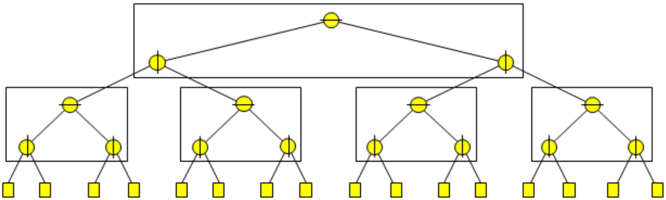
kdB-tree Insertion and Deletion

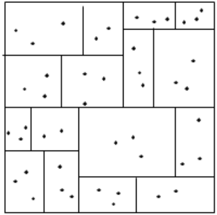
- Insert: Use logarithmic method ignoring deletes
- Delete: Simply delete point p from relevant D_i
  - i can be calculated based on # insertions since p was inserted
  - # insertions calculated by storing insertion number of each point in separate B-tree

  $\Rightarrow O(\log_B N)$ extra update cost
- To maintain $O(logN)$ structures D_i
  - Perform global rebuild after every $\Theta(N)$ updates

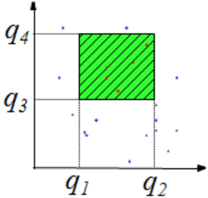
  $\Rightarrow O(1/B \log_{M/B} N/B) = O(\log_B N)$ extra update cost

Summary: kdB-tree





- 2d range searching in $O(N/B)$ space
  - Query in $O(\sqrt{N/B}+T/B)$ I/Os
  - Construction in $O(N/B \log_{M/B} N/B)$ I/Os
  - Updates in $O(\log_B^2 N)$ I/Os
- Optimal query among linear space structures



Summary/Conclusion: Tools and Techniques

- Tools
  - B-trees
  - Persistent B-trees
  - Buffer trees
  - Logarithmic method
  - Weight-balanced B-trees
  - Global rebuilding
- Techniques:
  - Bootstrapping
  - Filtering