R-Tree

• An R-tree is a depth-balanced tree
  – Each node corresponds to a disk page
  – Leaf node: an array of leaf entries
    • A leaf entry: (mbb, oid)
  – Non-leaf node: an array of node entries
    • A node entry: (dr, nodeid)
$m=2, M=4$

\begin{align*}
&[1,2,5,6] \quad [3,4,7,10] \quad [8,9,14] \quad [11,12,13] \\
\end{align*}
Properties

• The number of entries of a node (except for the root) in the tree is between \( m \) and \( M \) where \( m \in [0, M/2] \)
  
  – \( M \): the maximum number of entries in a node, may differ for leaf and non-leaf nodes
  
  \[ M = \left\lfloor \frac{\text{size}(P)}{\text{size}(E)} \right\rfloor \]  
  
  \( P \): disk page  \( E \): entry
  
  – The root has at least 2 entries unless it is a leaf

• All leaf nodes are at the same level

• An R-tree of depth \( d \) indexes at least \( m^{d+1} \) objects and at most \( M^{d+1} \) objects, in other words,
  \[ \left\lfloor \log_M N - 1 \right\rfloor \leq d \leq \left\lceil \log_m N - 1 \right\rceil \]
Search with R-tree

• Given a point $q$, find all mbbs containing $q$
• A recursive process starting from the root
  $\text{result} = \emptyset$
  For a node $N$
    if $N$ is a leaf node, then $\text{result} = \text{result} \cup \{N\}$
    else // $N$ is a non-leaf node
      for each child $N'$ of $N$
        if the rectangle of $N'$ contains $q$
          then recursively search $N'$
Time complexity of search

- If mbbs do not overlap on $q$, the complexity is $O(\log_m N)$.
- If mbbs overlap on $q$, it may not be logarithmic, in the worst case when all mbbs overlap on $q$, it is $O(N)$. 
Insertion – choose a leaf node

• Traverse the R-tree top-down, starting from the root, at each level
  – If there is a node whose directory rectangle contains the mbb to be inserted, then search the subtree
  – Else choose a node such that the enlargement of its directory rectangle is minimal, then search the subtree
  – If more than one node satisfy this, choose the one with smallest area,

• Repeat until a leaf node is reached
Insertion – insert into the leaf node

- If the leaf node is not full, an entry [mbb, oid] is inserted
- Else  // the leaf node is full
  - Split the leaf node
  - Update the directory rectangles of the ancestor nodes if necessary
Insert object 15

\[ m=2, \ M=4 \]
Insert object 16

$m=2, M=4$

[1,2,5,6][3,4,7][10,16] [8.9.14][11,12,13,15]
Split - goal

• The leaf node has $M$ entries, and one new entry to be inserted, how to partition the $M+1$ mbbs into two nodes, such that
  – 1. The total area of the two nodes is minimized
  – 2. The overlapping of the two nodes is minimized

• Sometimes the two goals are conflicting
  – Using 1 as the primary goal
Split - solution

• Optimal solution: check every possible partition, complexity $O(2^{M+1})$

• A quadratic algorithm:
  – Pick two “seed” entries $e_1$ and $e_2$ far from each other, that is to maximize
  area(mbb($e_1,e_2$)) – area($e_1$) – area($e_2$)
  here mbb($e_1,e_2$) is the mbb containing both $e_1$
  and $e_2$, complexity $O((M+1)^2)$
  – Insert the remaining $(M-1)$ entries into the two groups
Quadratic split cont.

• A greedy method
• At each time, find an entry $e$ such that $e$ expands a group with the minimum area, if tie
  – Choose the group of small area
  – Choose the group of fewer elements
• Repeat until no entry left or one group has $(M-m+1)$ entries, all remaining entries go to another group
• If the parent is also full, split the parent too. The recursive adjustment happens bottom-up until the tree satisfies the properties required. This can be up to the root.