

CSE6488: Mobile Computing Systems

Spatial Index for LBS Queries



Sungwon Jung

Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea
Tel: +82-2-705-8930
Email : jungsung@sogang.ac.kr

R-tree



- [Guttman 84] Main idea: allow parents to overlap!
 - => guaranteed 50% utilization
 - => easier insertion/split algorithms.
 - (only deal with Minimum Bounding Rectangles - **MBRs**)

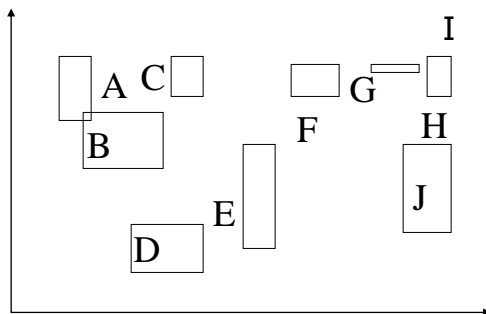


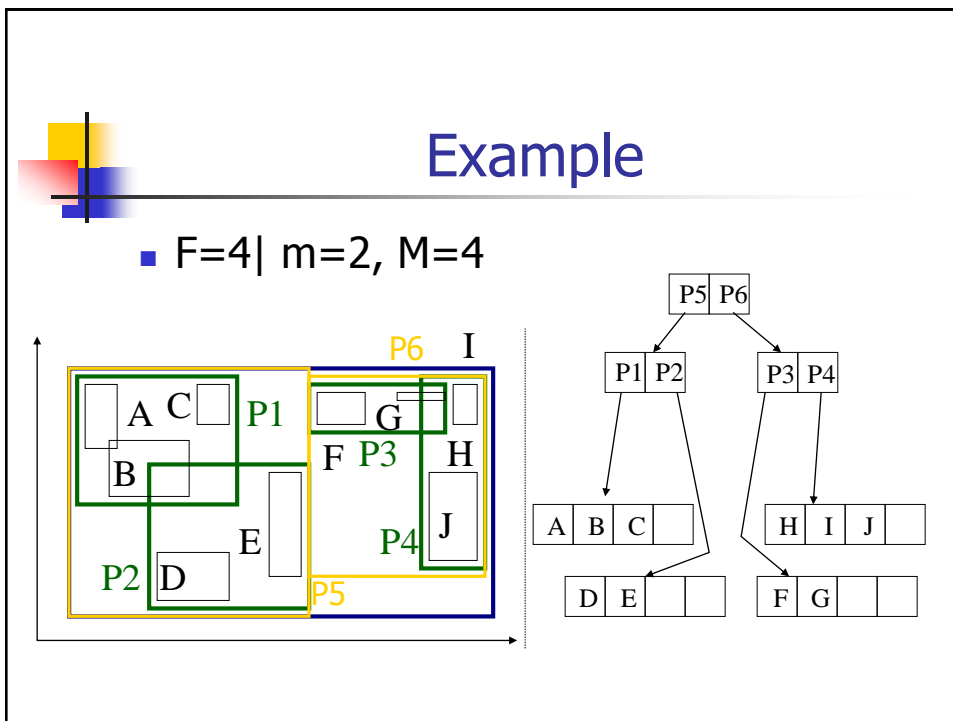
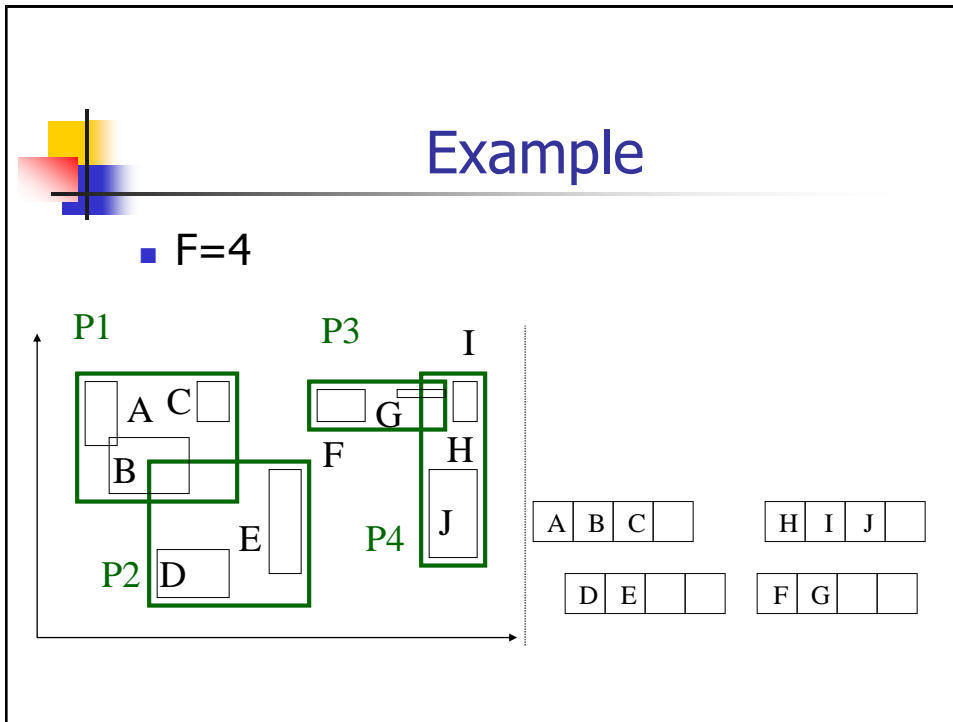
R-tree

- A multi-way external memory tree
- Index nodes and data (leaf) nodes
- All leaf nodes appear on the same level
- Every node contains between m and M entries
- The root node has at least 2 entries (children)

Example

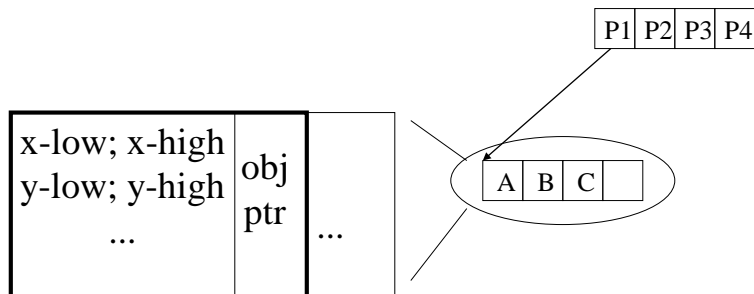
- eg., w/ fanout 4: group nearby rectangles to parent MBRs; each group -> disk page





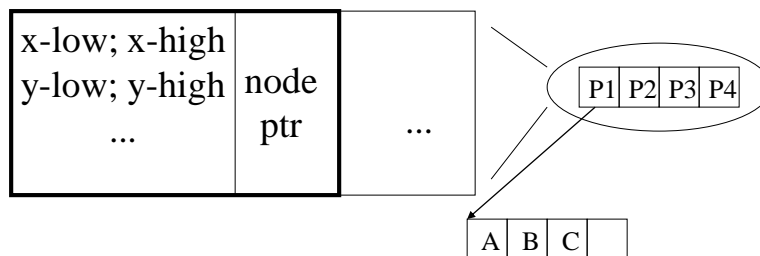
R-trees - format of nodes

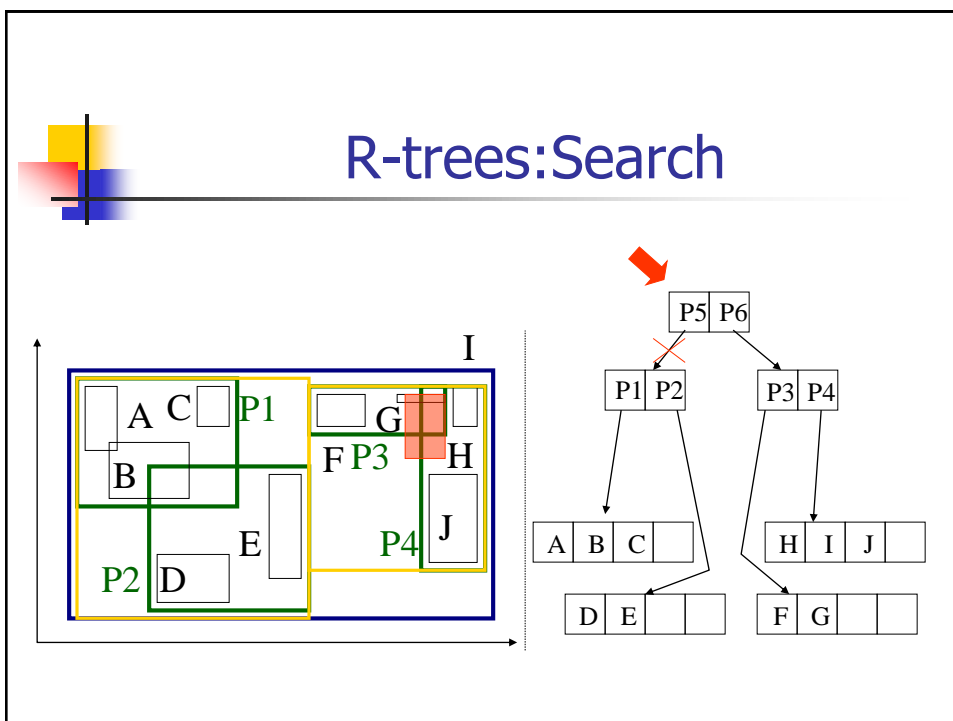
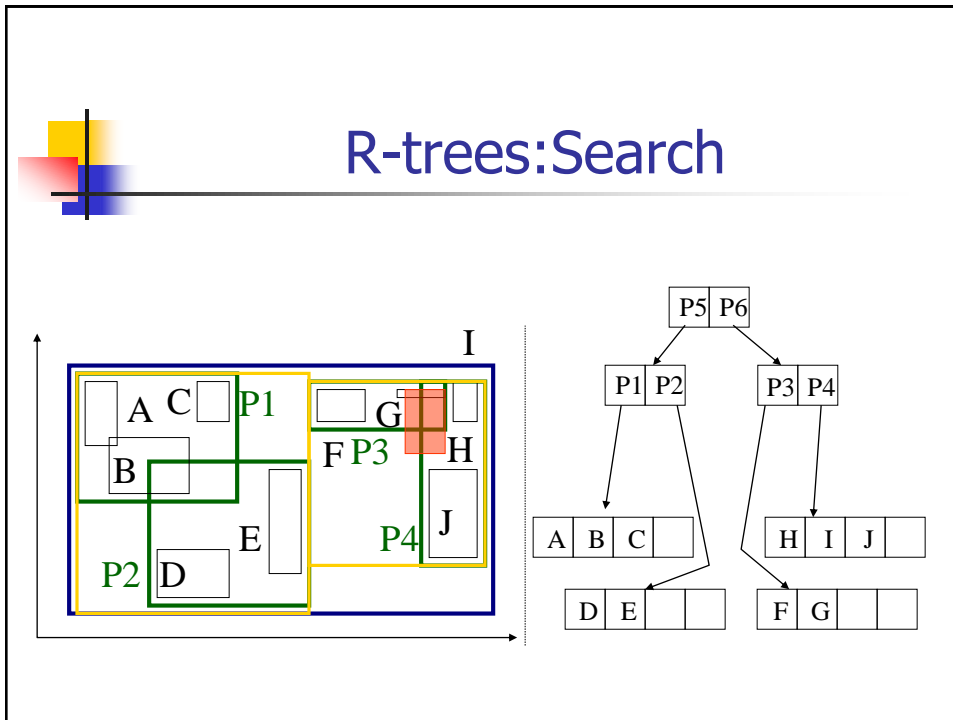
- {(MBR; obj_ptr)} for leaf nodes



R-trees - format of nodes

- {(MBR; node_ptr)} for non-leaf nodes



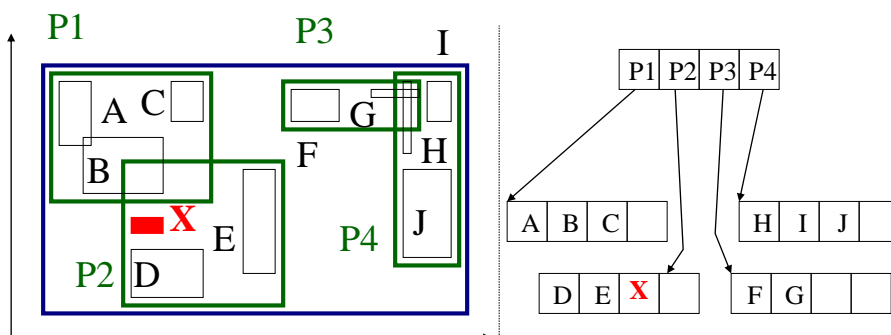


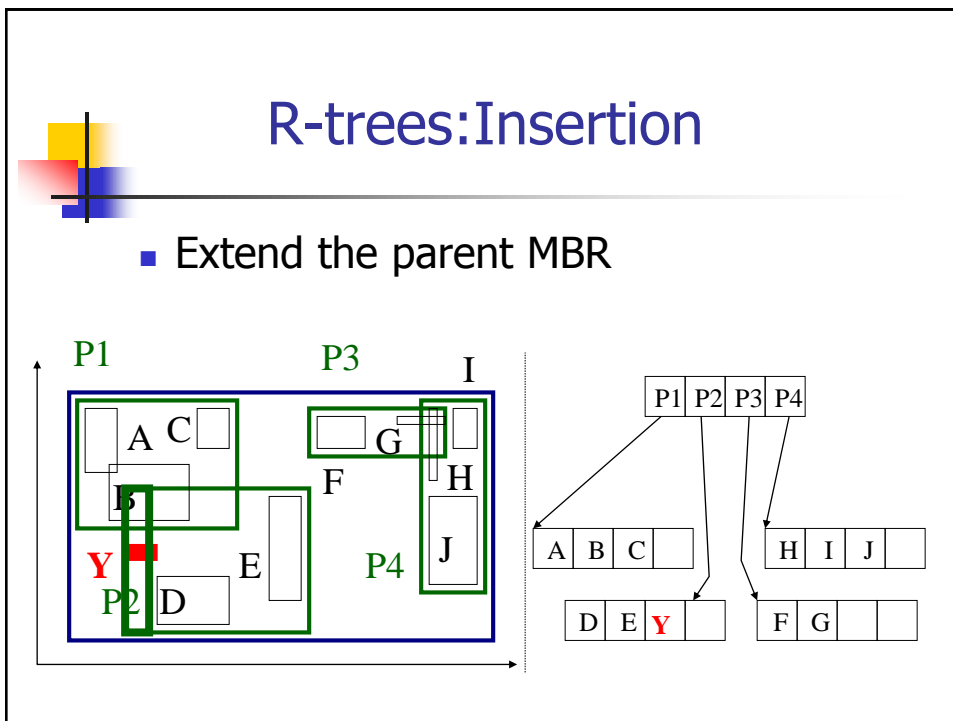
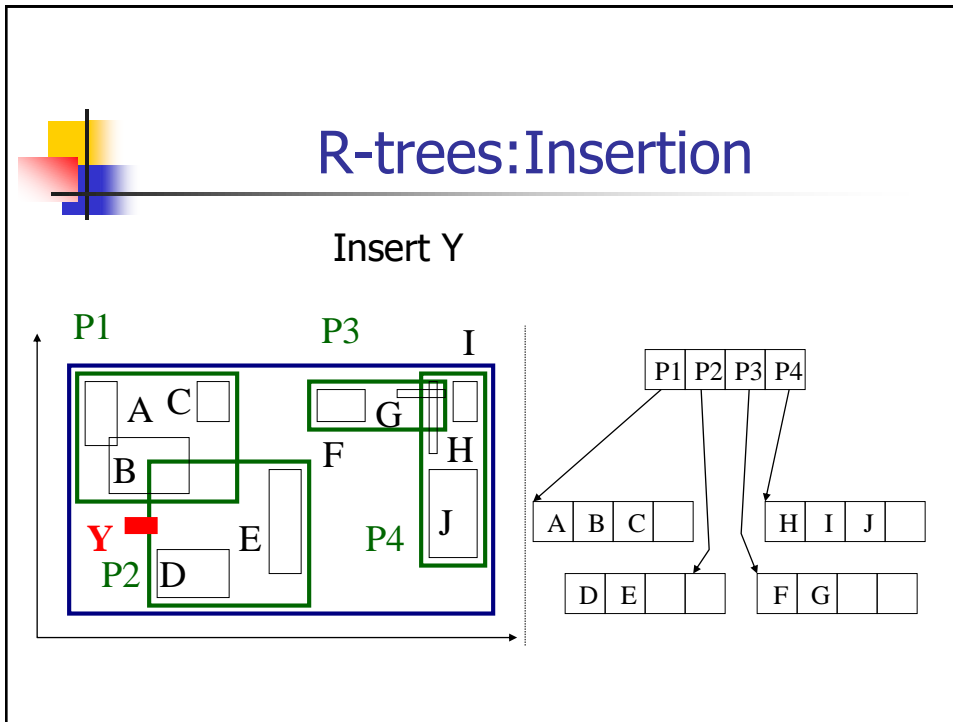
R-trees:Search

- Main points:
 - every parent node completely covers its 'children'
 - a child MBR may be covered by more than one parent - it is stored under ONLY ONE of them. (ie., no need for dup. elim.)
 - a point query may follow multiple branches.

R-trees:Insertion

Insert X



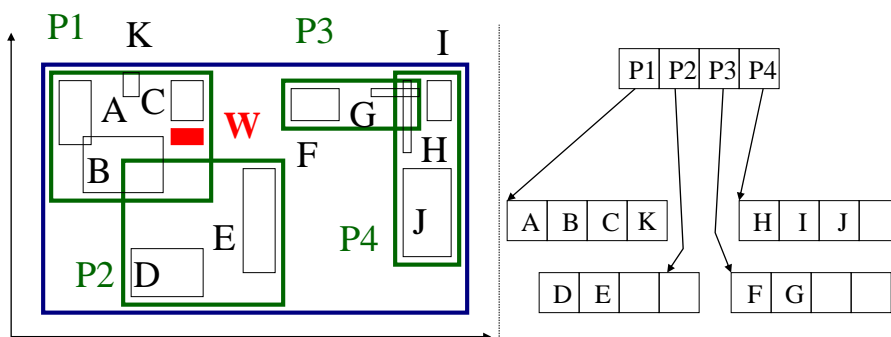


R-trees: Insertion

- How to find the next node to insert the new object?
 - Using ChooseLeaf: Find the entry that needs the least enlargement to include Y. Resolve ties using the area (smallest)

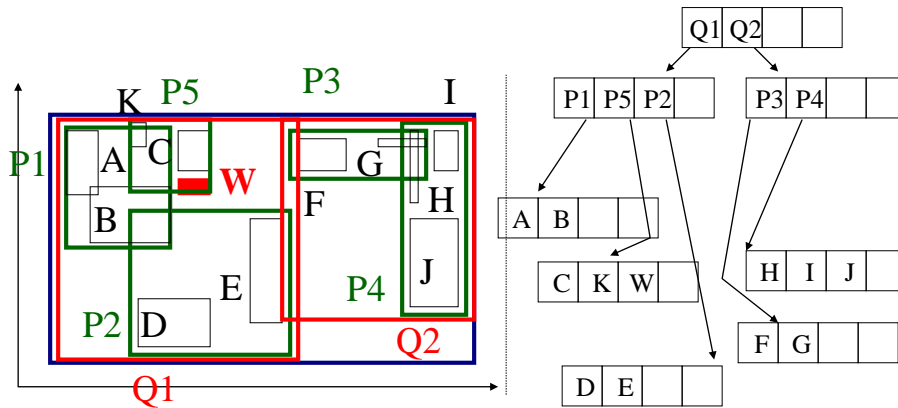
R-trees: Insertion

- If node is full then Split : ex. Insert w



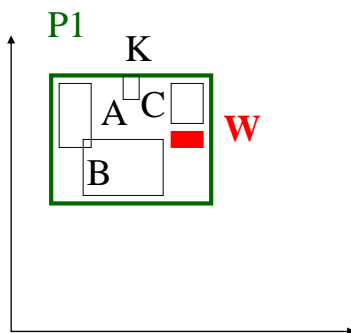
R-trees: Insertion

- If node is full then Split : ex. Insert w



R-trees: Split

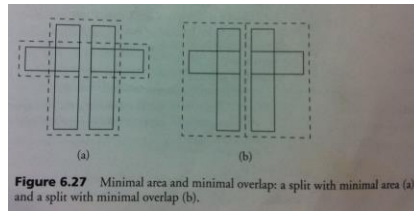
- Split node P1: partition the MBRs into two groups.



- (A1: plane sweep, until 50% of rectangles)
- A2: 'linear' split
- A3: quadratic split
- A4: exponential split:
 2^{M-1} choices

R-trees: Split

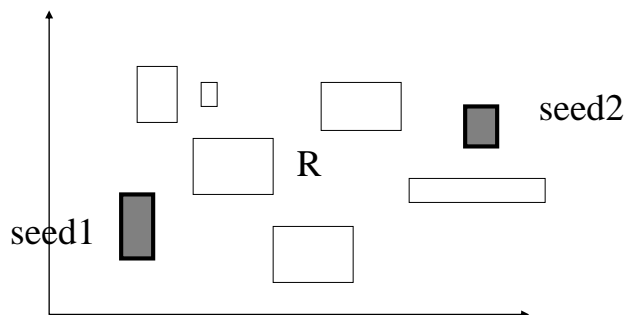
- Two requirements
 - Minimize the total area of the two nodes
 - Minimize the overlapping the two nodes



- Two requirements are not always compatible
 - R-tree splitting tries to minimize the total area first.

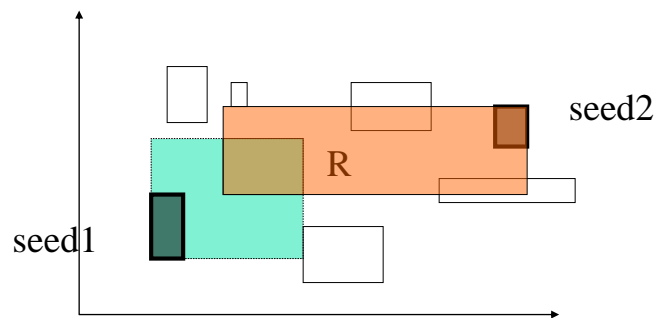
R-trees: Split

- pick two rectangles as 'seeds';
- assign each rectangle 'R' to the 'closest' 'seed'




R-trees:Split

- pick two rectangles as 'seeds';
- assign each rectangle 'R' to the 'closest' 'seed':
- 'closest': the smallest increase in area




R-trees:Split

- How to pick Seeds:
 - Linear: Find the highest and lowest side in each dimension, normalize the separations, choose the pair with the greatest normalized separation
 - Quadratic: For each pair E1 and E2, calculate the rectangle $J = \text{MBR}(E1, E2)$ and $d = J - E1 - E2$. Choose the pair with the largest d




R-trees: Insertion

- Use the **ChooseLeaf** to find the leaf node to insert an entry E
- If leaf node is full, then **Split**, otherwise insert there
 - Propagate the split upwards, if necessary
- Adjust parent nodes




R-Trees: Deletion

- Find the leaf node that contains the entry E
- Remove E from this node
- If underflow:
 - Eliminate the node by removing the node entries and the parent entry
 - Reinsert the orphaned (other entries) into the tree using **Insert**



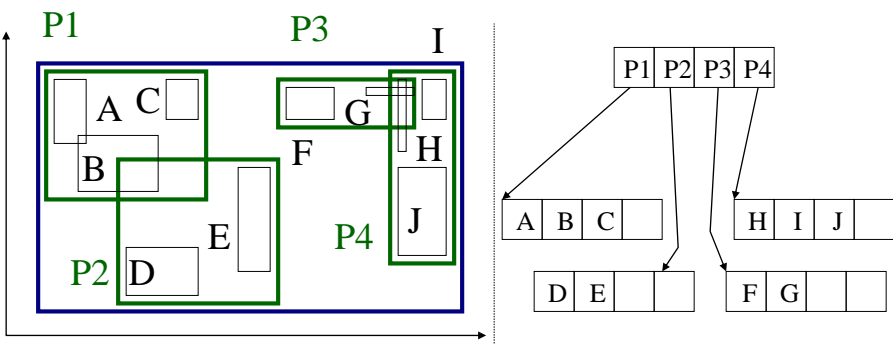
R-trees: Variations


- R+-tree: Do not allow overlapping, so split the objects (similar to z-values)
- R*-tree: change the insertion, deletion algorithms (minimize not only area but also perimeter, forced re-insertion)



R-tree

Multi-way external memory structure, indexes MBRs,
Dynamic structure






R-tree

- The original R-tree tries to minimize the area of each enclosing rectangle in the index nodes.
- Is there any other property that can be optimized?

R*-tree → Yes!



R*-tree

- A variation of the R-tree that provides several improvements to the insertion algorithm



R*-tree

- Optimization Criteria:
 - (O1) Area covered by an index MBR
 - (O2) Overlap between directory MBRs
 - (O3) perimeter of a node's directory rectangle
 - (O4) Storage utilization
- Sometimes it is impossible to optimize all the above criteria at the same time!



R*-tree

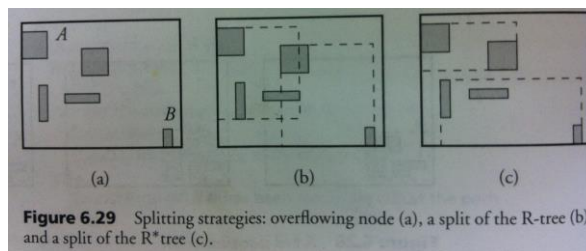
- ChooseSubtree for a node insertion:
 - If next node is a leaf node, choose the node using the following criteria:
 - Least overlap enlargement
 - Least area enlargement
 - Smaller area
 - Else
 - Least area enlargement
 - Smaller area

R*-tree

- SplitNode
 - Choose the axis to split
 - Choose the two groups along the chosen axis
- ChooseSplitAxis
 - Along each axis, sort rectangles and break them into two groups ($M-2m+2$ possible ways where one group contains at least m rectangles). Compute the sum S of all margin-values (perimeters) of each pair of groups. Choose the one that minimizes S
 - Split $M+1$ entries into 2 groups with at least m entries
 - $M-2m+2$ possible ways to split $M+1$ entries into two groups
- ChooseSplitIndex
 - Along the chosen axis, choose the grouping that gives the minimum overlap-value

R*-tree

- Splitting Strategies by finding the best axis



R*-tree

- Forced Reinsert:
 - defer splits, by forced-reinsert, i.e.: instead of splitting, temporarily delete some entries, shrink overflowing MBR, and re-insert those entries

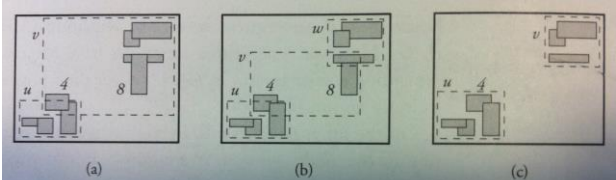
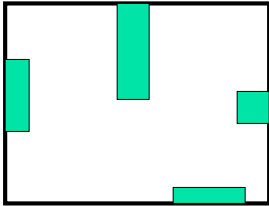


Figure 6.30 The R*tree reinsertion strategy: insertion of 8 (v overflows) (a), a split of the R-tree (b), and R*tree forced reinsertion of 4 (c).

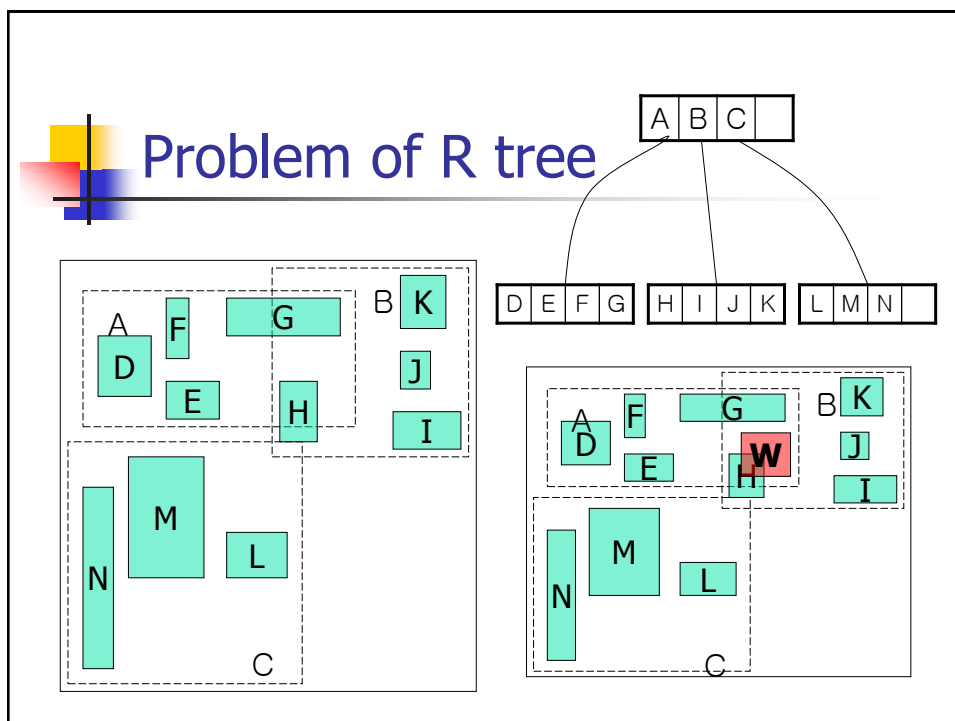
R*-tree

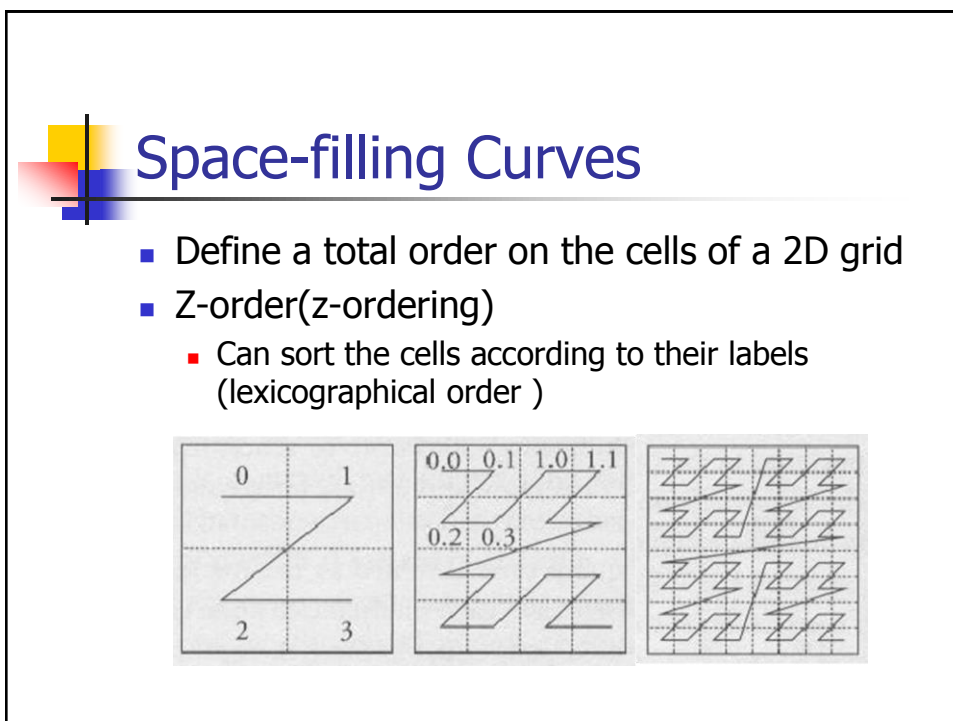
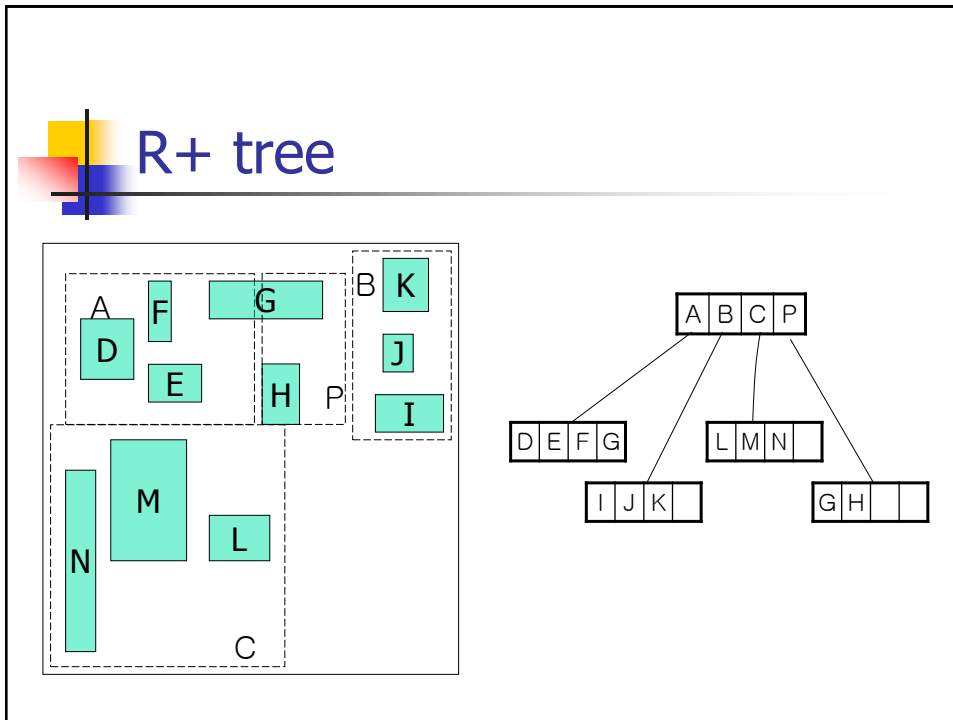
- Forced Reinsert:
 - Which ones to re-insert?
 - How many? A: 30%



R+ tree

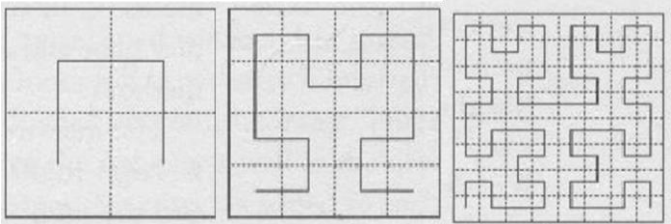
- Avoids multiple paths during searching.
- Objects may be stored in multiple nodes
- MBRs of nodes at same tree level do not overlap
- On insertion/deletion the tree may change downward or upward in order to maintain the structure





Space-filling Curves

- Hilbert curve
 - Consist of segments of uniform length

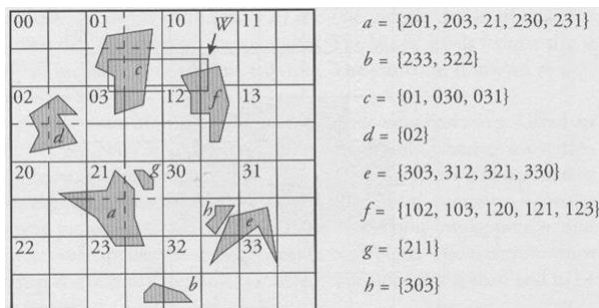


Space-filling Curves

- In both cases (z-ordering and hilbert curve), there exist some unavoidable situations in which two objects are close in the 2D space, but far from one another on the space-filling curve

The z-ordering Tree

- a set of objects with z-ordering decomposition
 - object a is represented by
 - $\{[201, a], [203, a], [21, a], [230, a], [231, a]\}$



The z-ordering Tree

- the same object may be represented in two quite distant leaves of the B+ tree due to the inherent proximity property of z-order
 - e.g. object b

