Spatial Index Structures

Vu Tuyet Trinh

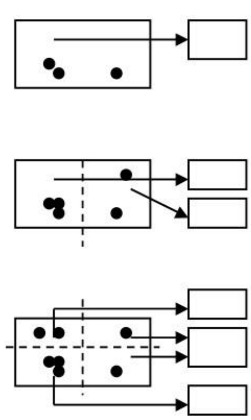
Outline

- Grid File
- Z-ordering
- Hilbert Curve
- Quad Tree
- PM
- PR
- R Tree
- R* Tree
- R+ Tree

Grid File

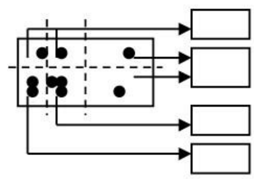
- Hashing methods for multidimensional points (extension of Extensible hashing)
- Idea: Use a grid to partition the space → each cell is associated with one page
- Two disk access principle (exact match)

Grid File



- Start with one bucket for the whole space.
- Select dividers along each dimension.
  Partition space into cells
- Dividers cut all the way.

Grid File



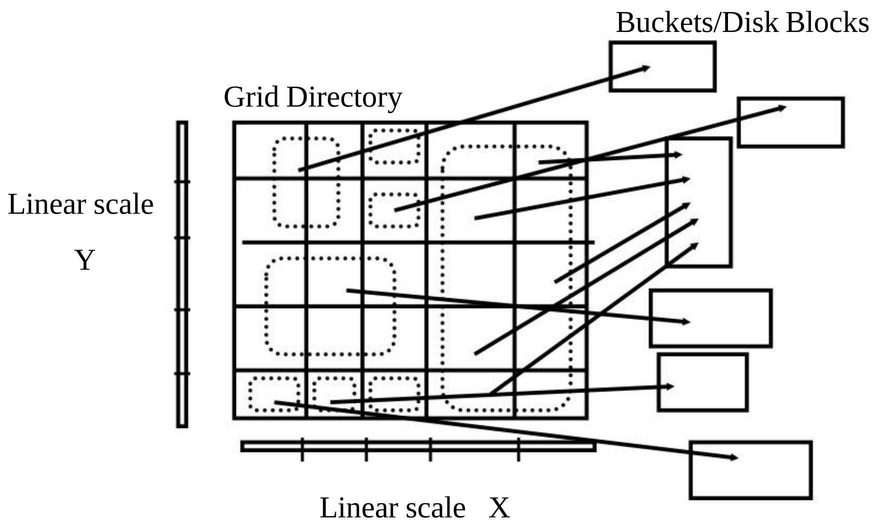
- Each cell corresponds to 1 disk page.
- Many cells can point to the same page.
- Cell directory potentially exponential in the number of dimensions

Grid File Implementation

Dynamic structure using a grid directory

- Grid array: a 2 dimensional array with pointers to buckets (this array can be large, disk resident) G (0,…, nx-1, 0, …, ny-1)
- Linear scales: Two 1 dimensional arrays that used to access the grid array (main memory) X(0, …, nx-1), Y(0, …, ny-1)

Example



Grid File Search

- Exact Match Search: at most 2 I/Os assuming linear scales fit in memory.
  - First use liner scales to determine the index into the cell directory
  - access the cell directory to retrieve the bucket address (may cause 1 I/O if cell directory does not fit in memory)
  - access the appropriate bucket (1 I/O)
- Range Queries:
  - use linear scales to determine the index into the cell directory.
  - Access the cell directory to retrieve the bucket addresses of buckets to visit.
  - Access the buckets.

Grid File insertion

- Determine the bucket into which insertion must occur.
- If space in bucket, insert.
- Else, split bucket
  - how to choose a good dimension to split?
  - ans: create convex regions for buckets.
- If bucket split causes a cell directory to split do so and adjust linear scales.
- insertion of these new entries potentially requires a complete reorganization of the cell directory--- expensive!!!

Grid File Deletion

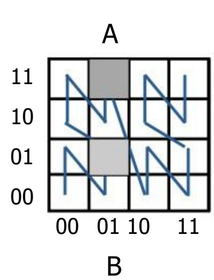
- Deletion may decrease the space utilization. Merge buckets
- We need to decide which cells to merge and a merging threshold
- Buddy system and neighbor system
  - A bucket can merge with only one *buddy* in each dimension
  - Merge adjacent regions if the result is a rectangle

Z-ordering

- Basic assumption: Finite precision in the representation of each coordinate, K bits (2^K values)
- The address space is a square (image) and represented as a 2^K x 2^K array
- Each element is called a pixel

Z-ordering

- Impose a linear ordering on the pixels of the image → 1 dimensional problem



$Z_A = \text{shuffle}(x_A, y_A) = \text{shuffle}(\text{“01”}, \text{“11”})$
$= 0111 = (7)_{10}$

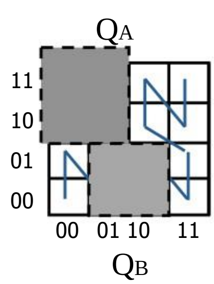
$Z_B = \text{shuffle}(\text{“01”}, \text{“01”}) = 0011$

Z-ordering

- Given a point (x, y) and the precision K find the pixel for the point and then compute the z-value
- Given a set of points, use a B+-tree to index the z-values
- A range (rectangular) query in 2-d is mapped to a set of ranges in 1-d

Queries

- Find the z-values that contained in the query and then the ranges



Q_A → range [4, 7]

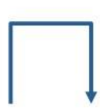
Q_B → ranges [2,3] and [8,9]

Hilbert Curve

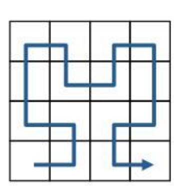
- We want points that are close in 2d to be close in the 1d
- Note that in 2d there are 4 neighbors for each point where in 1d only 2.
- Z-curve has some “jumps” that we would like to avoid
- Hilbert curve avoids the jumps : recursive definition

Hilbert Curve- example

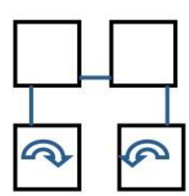
- It has been shown that in general Hilbert is beher than the other space filling curves for retrieval *
- Hi (order-i) Hilbert curve for 2_ix2_i array



H1



H2



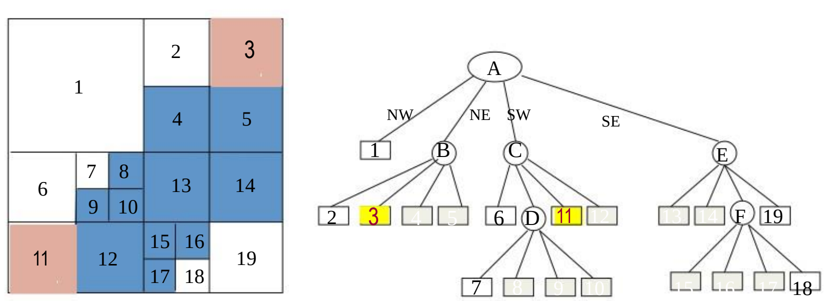
H(n+1)

* H. V. Jagadish: Linear Clustering of Objects with Multiple Atributes. ACM SIGMOD Conference 1990: 332-342

Quad Trees

- Region Quadtree
  - The blocks are required to be disjoint
  - Have standard sizes (squares whose sides are power of two)
  - At standard locations
  - Based on successive subdivision of image array into four equal-size quadrants
  - If the region does not cover the entire array, subdivide into quadrants, sub-quadrants, etc.
  - A variable resolution data structure

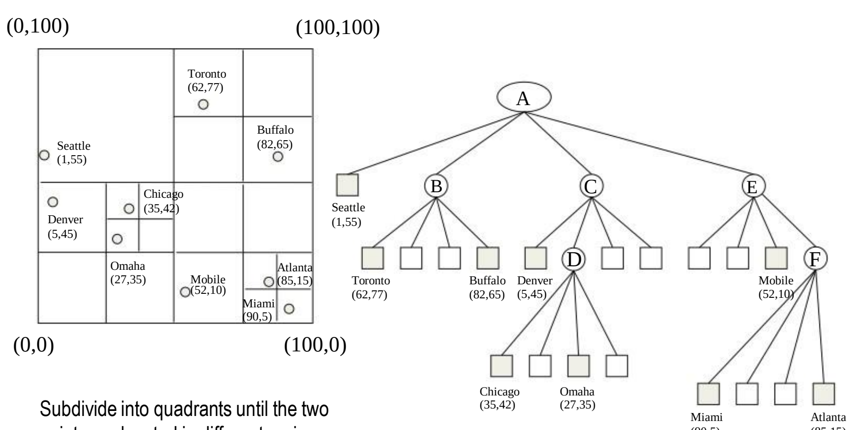
Example of Region Quadtree



PR Quadtree

- PR (Point-Region) quadtree
- Regular decomposition (similar to Region quadtree)
- Independent of the order in which data points are inserte into it
- L: if two points are very close, decomposition can be very deep

Example of PR Quadtree



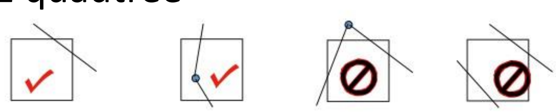
Subdivide into quadrants until the two points are located in different regions

PM Quadtree

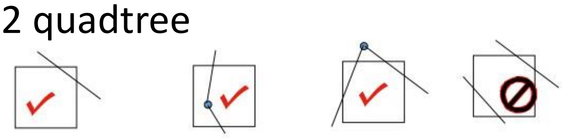
- PM (Polygonal-Map) quadtree family
  - PM1 quadtree, PM2 quadtree, PM3 quadtree, PMR quadtree, … etc.
- PM1 quadtree
  - Based on regular decomposition of space
  - Vertex-based implementaDon
  - Criteria
    - At most one vertex can lie in a region represented by a quadtree leaf
    - If a region contains a vertex, it can contain no partial-edge that does not include that vertex
    - If a region contains no vertices, it can contain at most one partial-edge

PM Quadtree

PM1 quadtree

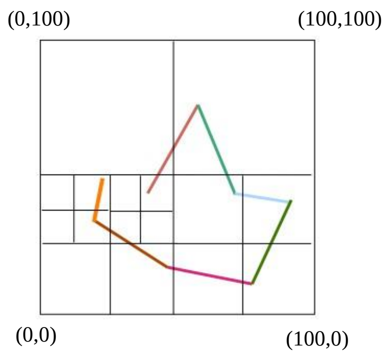


PM2 quadtree



PM3 quadtree

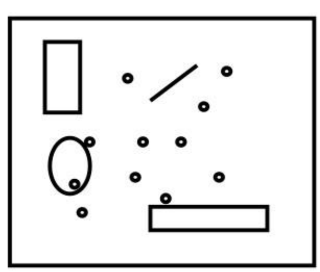
Example of PM1 Quadtree



- Each node in a PM quadtree is a collecDon of parDal edges (and a vertex)
- Each point record has two field (x,y)
- Each parDal edge has four field (starDng_point, ending_point, lel region, right region)

Remarks

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer spatial queries (range, nn, etc)



R-trees

- [GuEman 84] Main idea: extend B+-tree to multidimensional spaces!

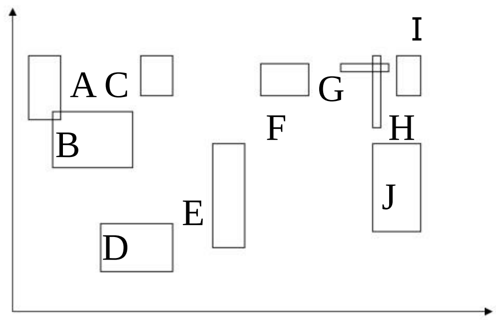
  - (only deal with Minimum Bounding Rectangles - **MBR**s)

R-trees

- 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)

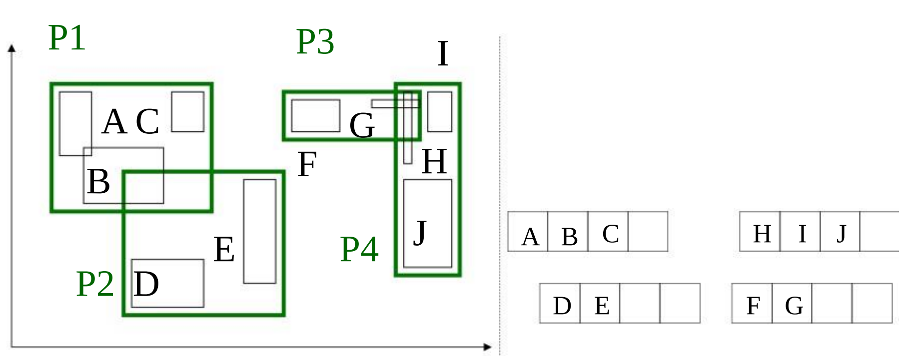
Example

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



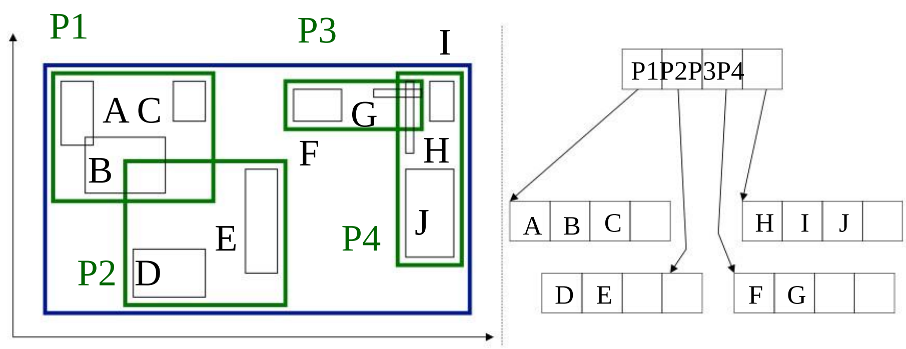
Example

- F=4



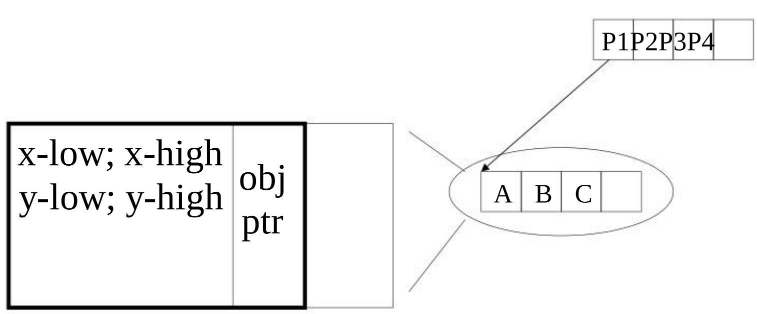
Example

- F=4



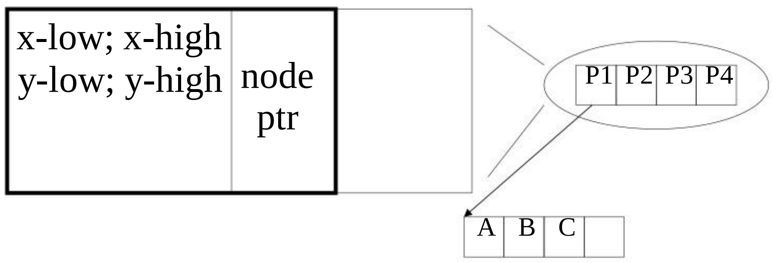
R-trees - format of nodes

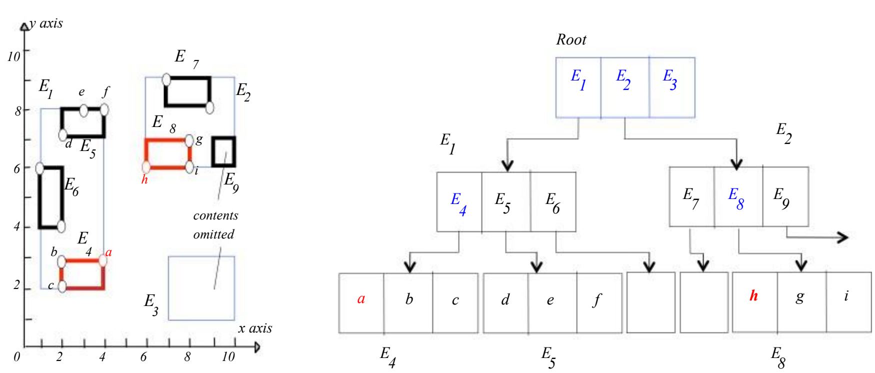
- {(MBR; obj_ptr)} for leaf nodes



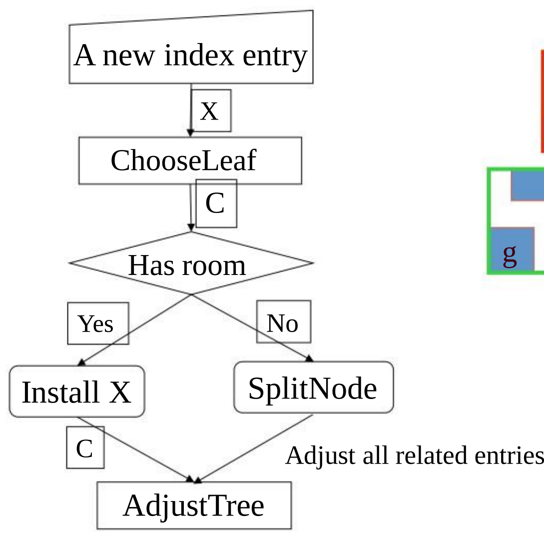
R-trees - format of nodes

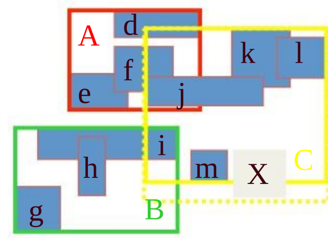
- {(MBR; node_ptr)} for non-leaf nodes





Insertion Processes



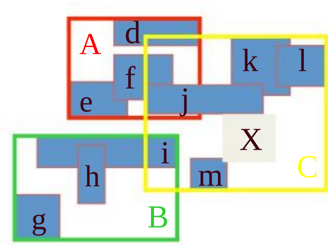


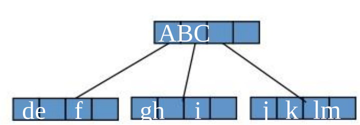
Different variant:
- Exhaustive
- Quadratic
- Linear
- Packed
- Hilbert Packed
- …etc.

Processes of Quadratic Spilt

(page 52 in GuEman’s paper)

Pick first entry for each group
Run PickSeeds





Processes of Quadratic Spilt

(page 52 in GuEman's paper)

PickSeeds

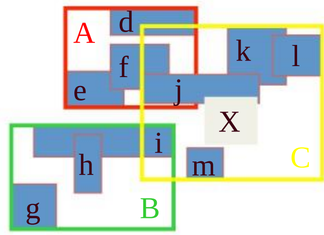
PS1 [Calculate inefficiency of grouping entries together]

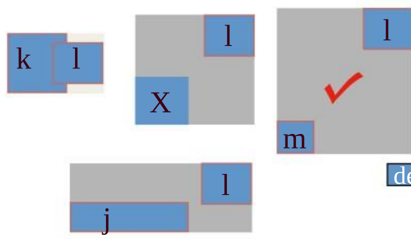
For each pair of E1 and E2, compose a rectangle R including E1 and E2

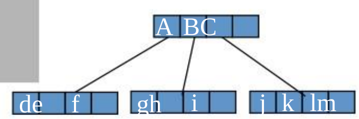
Calculate d = area(R) - area(E1) - area(E2)

PS2 [Choose the most wasteful pair]

Choose the pair with the largest d

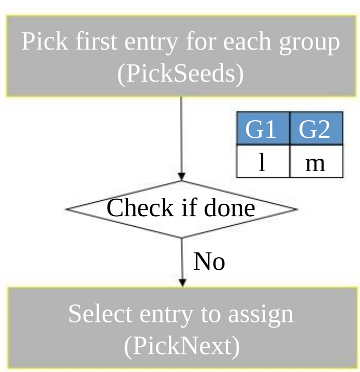


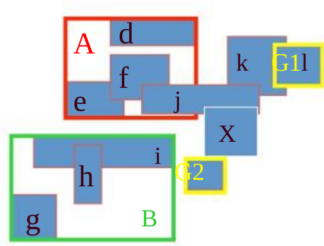


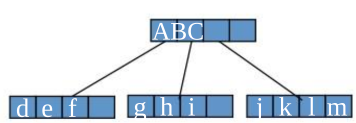


Processes of Quadratic Spilt

(page 52 in GuEman's paper)

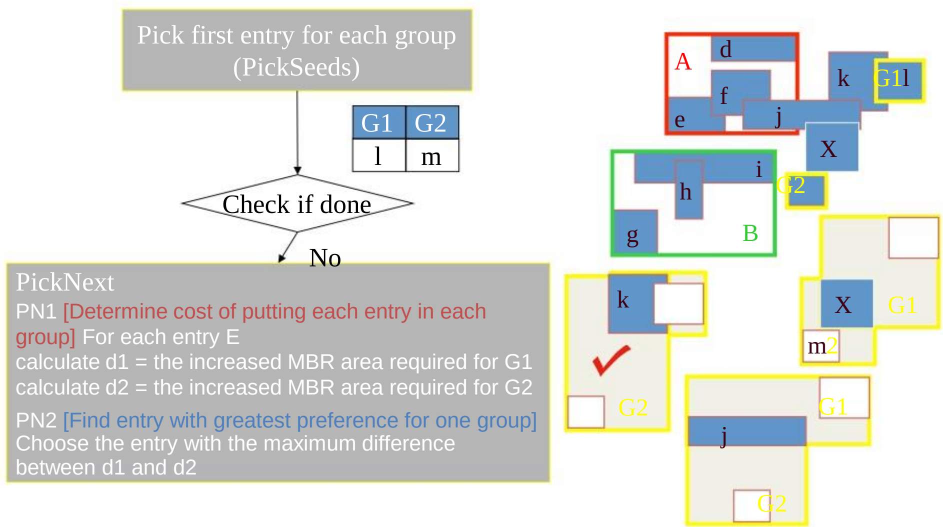






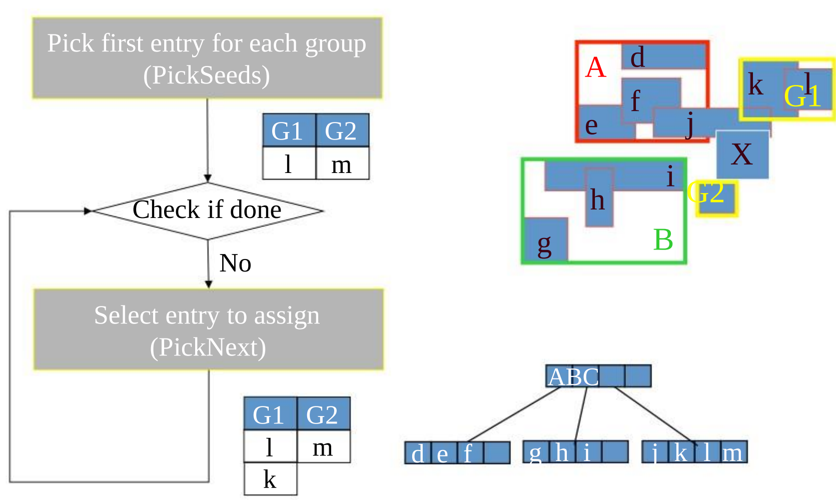
Processes of Quadratic Spilt

(page 52 in GuEman's paper)



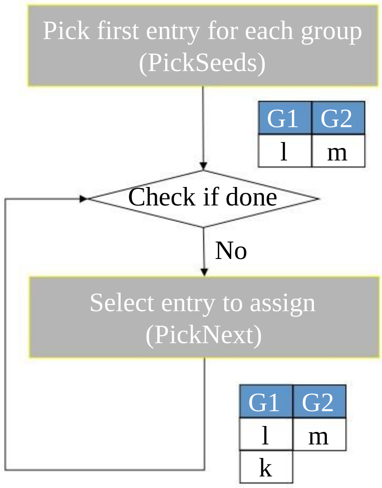
Processes of Quadratic Spilt

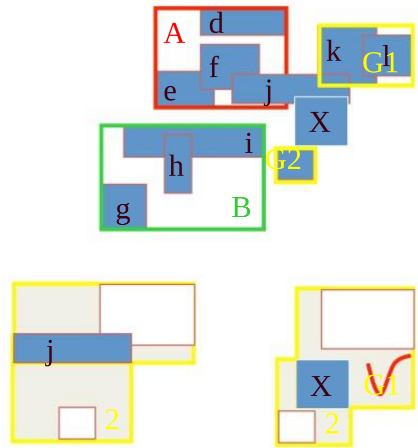
(page 52 in GuEman's paper)



Processes of Quadratic Spilt

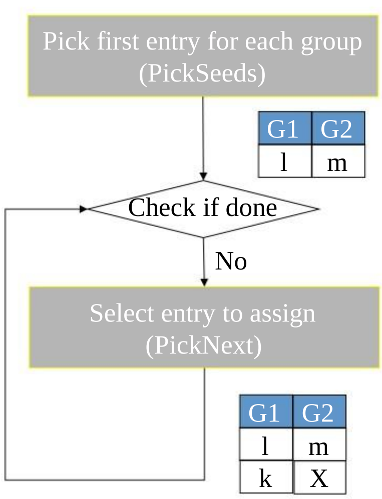
(page 52 in GuEman’s paper)

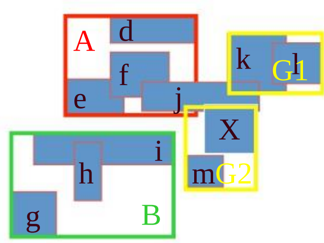




Processes of Quadratic Spilt

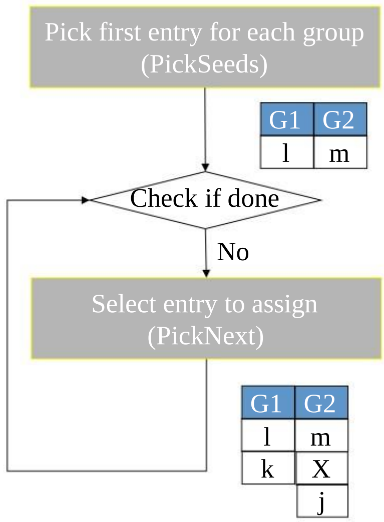
(page 52 in GuEman’s paper)

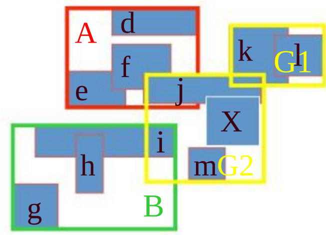


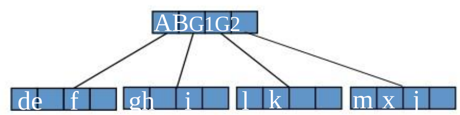


Processes of Quadratic Spilt

(page 52 in GuEman's paper)

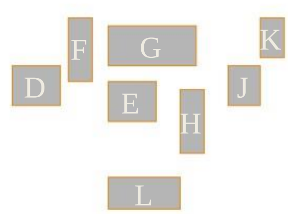






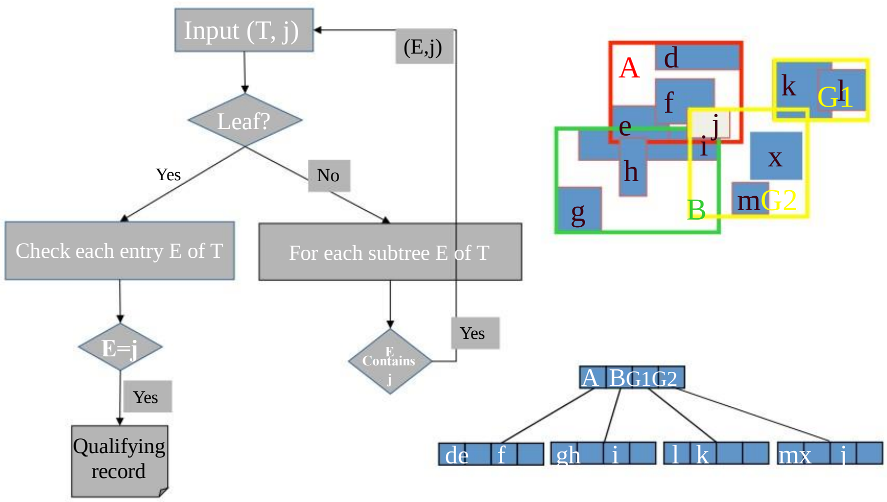
Excercise

- (m,M)=(2,4)

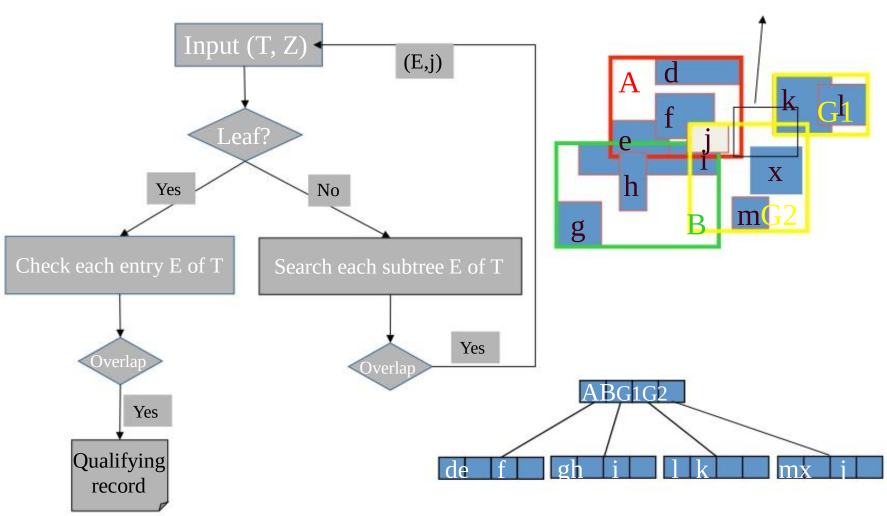


- Build a R-Tree for these spatial data
- Hint: You could use the Spatial index structures demo application step by step

Search Object in R-Tree



(find objects that overlap with Z)

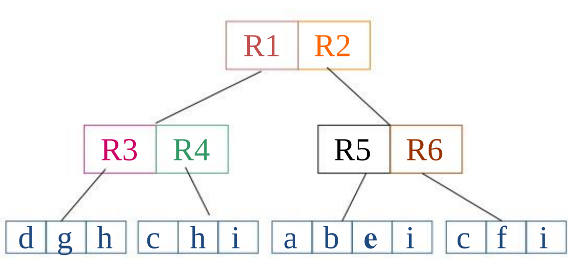


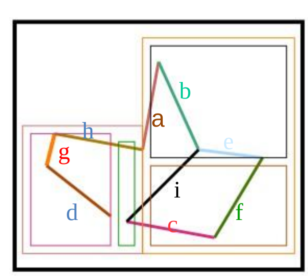
Main Drawbacks of R-Tree

- R-tree is not unique, rectangles depend on how objects are inserted and deleted from the tree.
- In order to search some object you might have to go through several rectangles or the whole database
  - Why?
  - Solution?

R+-Tree

- Overcome problems with R-Tree
- If node overlaps with several rectangles insert the node in all
- Decompose the space into disjoint cells





R+-Tree Properties

- R+-tree and cell-trees used approach of discomposing space into cells
  - R+-trees deals with collection of objects bounded by rectangles
  - Cell tree deals with collection of objects bounded by convex polyhedron
- R+-trees is extension of k-d-B-tree
- Retrieval /mes are smaller
- When summing the objects, needs eliminate duplicates
- Again, it is data-dependent

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

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

R*-tree

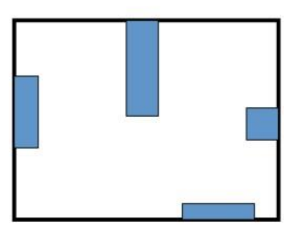
- ChooseSubtree:
  - If next node is a leaf node, choose the node using the following criteria:
    - Least overlap enlargement
    - Least area enlargement
    - Smaller area
  - Else
    - Least area enlargement
    - Smaller area

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
- ChooseSplitIndex
  - Along the chosen axis, choose the grouping that gives the minimum overlap-value

R*-tree

- Forced Reinsert:
  - defer splits, by forced-reinsert, i.e.: instead of splinng, temporarily delete some entries, shrink overflowing MBR, and re-insert those entries
- Which ones to re-insert?
- How many? A: 30%



References

- NaDonal Technical University of Athens , TheoreDcal Computer Science II: Advanced Data Structures
- Jürg Nievergelt, Hans Hinterberger, Kenneth C. Sevcik: The Grid File: An Adaptable, Symmetric MulDkey File Structure. ACM Trans. Database Syst. 9 (1): 38-71 (1984)
- H. V. Jagadish: Linear Clustering of Objects with MulDple Atributes. ACM SIGMOD Conference 1990: 332-342

References

- Antonin GuEman, R-trees: a dynamic index structure for spa/al searching, Proceedings of the 1984 ACM SIGMOD interna/onal conference on Management of data, June 18-21, 1984, Boston, MassachuseEs
- Norbert Beckmann, et al. , The R*-tree: an efficient and robust access method for points and rectangles, SIGMOD 1990
- Roussopoulos et al. , The R+-Tree: A Dynamic Index for Mul/-Dimensional Objects, VLDB 1987
- Na/onal Technical University of Athens , Theore/cal Computer Science II: Advanced Data Structures