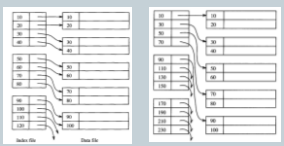
Spatial Access Methods (Indexes)

Access Methods

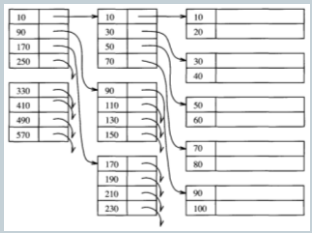
- Main memory limited, so data stored externally
  - bottleneck: access time & transfer rate
  - data downloaded in pages (about 4kB)
    - page fault: page is not in memory and needs to be loaded
- How to store information in files so it can be found efficiently ?
  - Hashing
  - Sorting
  - RAID

Indexes

- single-level
  - primary, clustering, secondary
  - dense, sparse
- multi-level
  - B-tree
  - B+-tree
- for spatial databases?

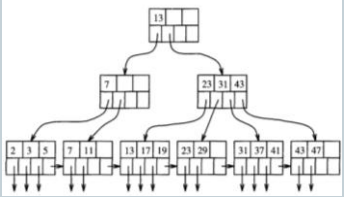


Multi-Level Index



advantage ?

Example: B+ tree



balanced search tree
node: keys/pointers
leaves: pointers to data and pointers to next leaf
all at same depth
#pointers in node: from (n+1)/2 to n

- typical n ? (Assume 4Kb page size, integer 4 bytes, pointer 8 bytes)
- why lower bound on # pointers?
- why pointers to next leaf ?
- search, insertion, deletion ?

Operations

- point queries
- window queries
- nearest-neighbor/distance queries
- location queries
- spatial joins (next time)

Spatial Access Methods

- design for mbbs only, i.e. rectangles,
  - pre-filtering
  - precise results can be obtained using methods from computational geometry we saw earlier
- tasks
  - index construction
  - search operations
  - index modification
- specifications
  - fast (sublinear) point/range queries
  - additional space roughly proportional to original data

Approaches

- space-driven
  - grid-file
  - (linear) quad-tree
- data-driven
  - R-tree, R*-tree
  - X-tree
  - kdb-tree

Space driven SAMS

I. The Grid File
II. The (Linear) Quadtree
III. The z-ordering tree

Why not use traditional methods?

Example (G-MUW):

1,000,000 points randomly distributed in $[0,1000]^2$

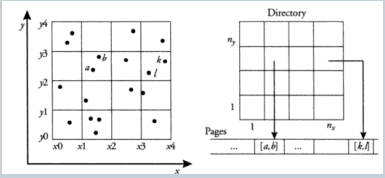
page takes 100 point records (10,000 pages)

use two B$^+$-trees, one on x, one on y

find points with $450 \le x, y \le 550$

The Grid File

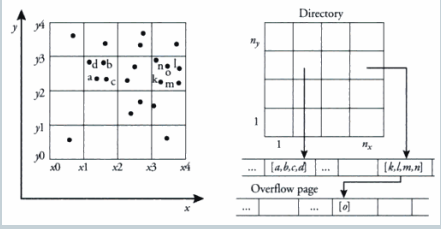
- decompose space into *cells* to obtain n x m grid,
- cell data is looked up in directory determined by scales
- decomposition can be *fixed* or the grid can be *dynamic*



- fixed grid
- what happens if page overflows ?

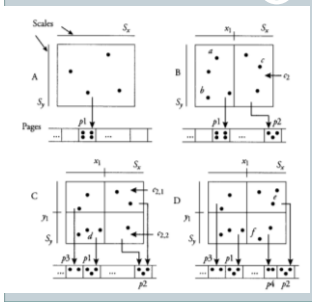
$S_x = \{x_o, \ldots x_n\}$ $S_y = \{y_o, \ldots, y_m\}$ are the scales

Fixed Grid



- use overflow pages
- point query/window query ?
- advantages/disadvantages ?

Dynamic Grid

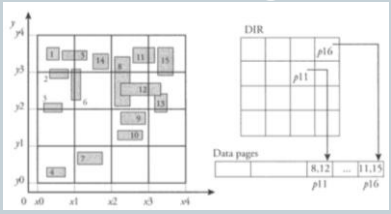


- add a:
  cell split and directory split
- add b, c
  nothing happens
- add d
  cell/directory split for c1
  only cell split for c2
- add e
  nothing happens
- add f
  cell split (no directory split)

grid file issues

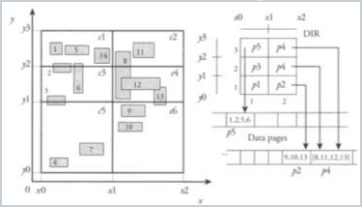
- empty buckets
  - when data shows clustering/correlation, i.e. is not uniformly distributed
  - worse with higher-dimensional data
- nearest neighbor queries
  - might not be in some cell (give example)
  - might not even be in adjacent cell (give example)
  - how could one proceed ?
- range queries
  - potentially large number of cells need to be investigated

indexing rectangles using grid files



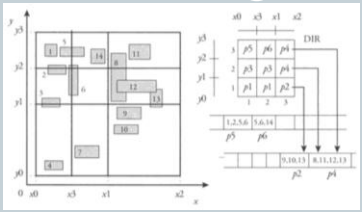
- include rectangle in each cell it overlaps (object duplication)
- cell splits more likely

Grid File Insertion



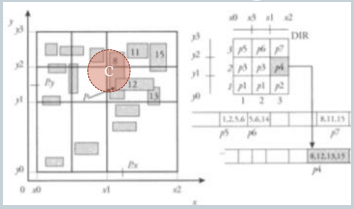
- insert rectangle 14

Grid File Insertion



- after insertion

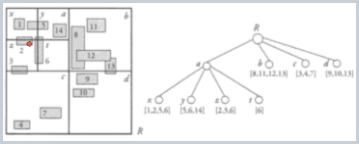
Point/Window Queries



- which rectangles does P belong to ?
- which rectangles does C overlap ?

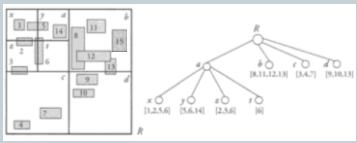
Quadtree

- as needed, recursively split square into 4 quadrants
- each inner node has degree 4; leaves list objects intersecting quadrant



- point query: follow path to relevant leaf, test all objects intersecting quadrant

Inserting Rectangle



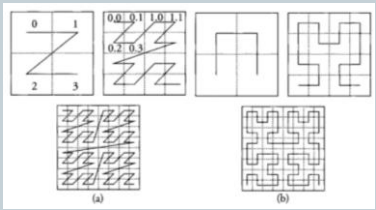
Add rectangle 15

what are problems with the quadtree approach ?

- mapping it to pages
- limited fan-out
- unbalanced tree structure

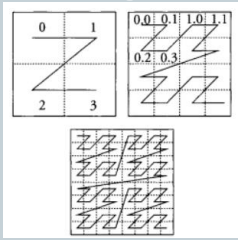
Idea: use quadtrees, but store them through B-trees

space-filling curves



- (a) z-ordering (Morton Codes), (b) Hilbert
- map 2-dimension objects to 1 dimension, maintaining "closeness"

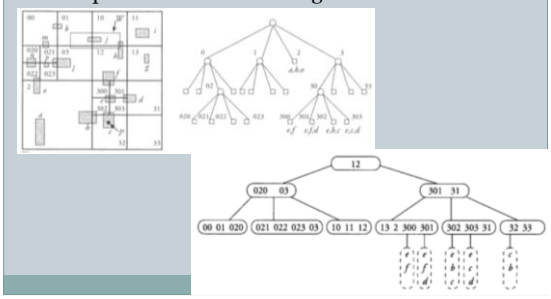
labeling of grid



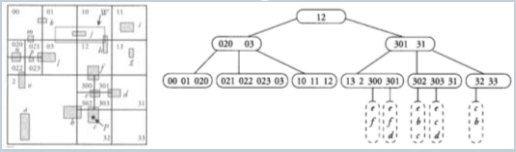
- give labels of points in 8x8 grid
- close squares tend to have close labels

linear quadtree

- store quadtree as B⁺-tree using labels of cells



Point Query

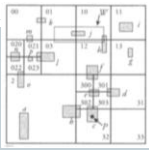


calculate label L of grid cell containing P (for full grid)
find longest L' in tree with L' prefix of L: (L', p) = MaxInf(L)
page = ReadPage(p)
for each object o in page
  if P inside o.mbb then add o to results

pointer to page

contains non-standard operations for B⁺-tree

Range Query

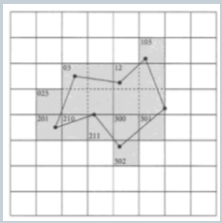


calculate label L1, L2 or NW and SE corner of rectangle W
(L1', p1) = MaxInf(L1), (L2', p2) = MaxInf(L2)
Q = range(L1', L2')
for each q in Q
  if quadrant(q.label) overlaps W
    page = ReadPage(q.p)
    for each o in page
      if W overlaps o.mbb then add o to results
sort results, remove duplicates

- bad cases ?
- insertion of rectangle ?
- deletion of rectangle ?

z-ordering tree

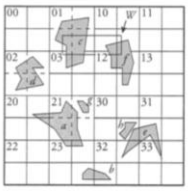
- raster approximation of geometry by z-order tree of depth d



precision (d) matters

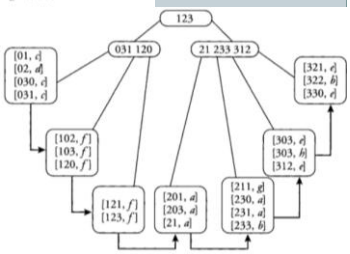
```
DECOMPOSE (o: geometry, q: quadrant): set(quadrant)

begin
  decomp_NW, decomp_NE, decomp_SW, decomp_SE: set(quadrant)
  result = ∅
  // Check that q overlaps o, else do nothing
  if (q overlaps o) then
    if (q is minimal) then
      result = {q}
    else
      // Decompose o into pieces, one per subquadrant
      for each sq in {NW(q), NE(q), SW(q), SE(q)} do
        decomp_sq = DECOMPOSE (o, sq)
      end for
      // If each decomposition results in the full subquadrant, return q
      if (decomp_NW ∪ decomp_NE ∪ decomp_SW ∪ decomp_SE = q) then
        result = {q}
      else
        // Take the set union of the four decompositions
        result = decomp_NW + decomp_NE + decomp_SW + decomp_SE
      end if
    end if
  end if
  return result
end
```

z-ordering tree



- find shapes overlapping W
- find shapes containing a point P
- nearest neighbor of point P ?

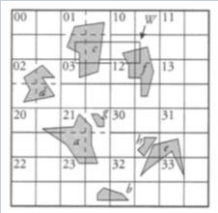
a = {201, 203, 21, 230, 231}
b = {233, 322}
c = {01, 030, 031}
d = {02}
e = {303, 312, 321, 330}
f = {102, 103, 120, 121, 123}
g = {211}



z-ordering without redundancy

to each shape assign smallest quadrant containing it

removes redundancy



a = {2}
h = {303}
d = {02}

what are bad examples ?

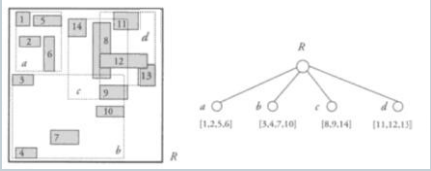
how to find shapes overlapping W ?

Data driven SAMS

I. R-tree
II. R*-tree
III. R+-tree

R-tree (Region tree)

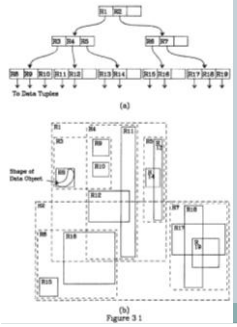
- data-driven, balanced tree



- inner node: list of (mbb, node id), where mbb bounds children
- leaf node: list of (mbb, object id), all at same height
- every node has between (m, M) entries, $0 \le m \le M/2$ (fixed parameters)

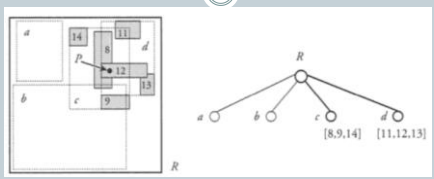
Example: page size = 4Kb, each entry 20 bytes, m = 50%
what is M? how many objects can a tree of depth 1, 2, 3 index?

R-tree



from Guttman's original paper (A dynamic index structure for spatial searching)

Point Query

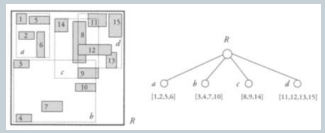



```
Q = {root}
    while Q <> {}
        v = pop(Q)
        for every (R, u) in v
                if P in R and u is inner node, then Q = Q u {u}
                else add R to results
```

Window query is similar

Insertion

- implementation of R-tree based on B^+-tree



Example: insert 15, need to extend mbb of d in leaf and in R

Example: insert 16, add to b: overflow -> split

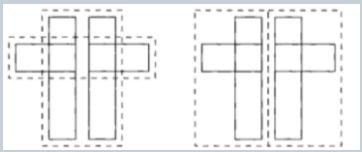
Insertion Algorithm

```
INSERT (e: LeafEntry)
begin
  // Initializes the search with root
  node = root
  // Choose a path down to a leaf
  while (node is not a leaf) do
    node = CHOOSESUBTREE (node, e)
  end while
  // Insert in the leaf
  INSERTINLEAF (node, e)
  // Split and adjust the tree if the leaf overflows, else adjust the path
  if (node overflows) then
    SPLITANDADJUST (node)
  else
    ADJUSTPATH (node)
  end if
end
```

- choose subtree so that adding e extends area minimally; if several options, choose one of smallest area
- Adjust: extends mbbs along path to leaf
- SplitAndAdjust: splits overflowing nodes, and inserts new node; recursively fixes parents and adjusts mbbs

How to Split a Node?

- minimize total area of new nodes
- minimize overlap of mbbs of nodes



Possible splitting algorithms (Guttman) categorized by running time:
- linear split
- quadratic split
- exponential split

Exponential Split

- try all possible splits
- find best split
- possible for M = 4, but with M around 100-200 not realistic

Linear Split

- pick two rectangles at farthest distance
- add remaining rectangles in random order
  - add rectangle to group whose mbb will extend least

How can this lead to bad splits?

Quadratic Split

dead space = mbb – rectangles

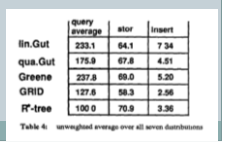
pick two rectangles with smallest dead space as seeds for two groups R1, R2

pick object that maximizes difference of dead space it forms with R1 and R2

add that object to group that requires least change in mbb (unless that would lead to overflow)

repeat until all objects are placed

from Beckmann, Kriegel, Schneider, Seeger, The R*-tree: An Efficient And Robust Access Method for Points and Rectangles



	query average	stor	insert
lin.Gut	233.1	64.1	7.34
qua.Gut	175.9	67.8	4.51
Greene	237.8	69.0	5.20
GRID	127.6	58.3	2.56
R*-tree	100.0	70.9	3.36

Table 4: unweighted average over all seven distributions

Deletion

to delete a rectangle r

- find leaf containing r
- delete r from leaf
- if leaf doesn't have enough entries (i.e. < m)
- delete leaf, add to Q
- recursively check that parent hasn't become to small, if so delete, add to Q
- reinsert all nodes in Q (at their original level)

Advantage: rebalances tree (give example of bad insertion order)

R+-tree

4.2 Split of the R*-tree

The R*-tree uses the following method to find good splits. Along each axis, the entries are first sorted by the lower value, then sorted by the upper value of their rectangles. For each sort M-2m+2 distributions of the M+1 entries into two groups are determined, where the k-th distribution (k = 1, .,(M-2m+2)) is described as follows. The first group contains the first (m-1)+k entries, the second group contains the remaining entries.

For each distribution goodness values are determined. Depending on these goodness values the final distribution of the entries is determined. Three different goodness values and different approaches of using them in different combinations are tested experimentally.

(i) area-value: area[bb(first group)] + area[bb(second group)]

(ii) margin-value: margin[bb(first group)] + margin[bb(second group)]

(iii) overlap-value: area[bb(first group) ∩ bb(second group)]

Here bb denotes the bounding box of a set of rectangles.

Possible methods of processing are to determine
- the minimum over one axis or one sort
- the minimum of the sum of the goodness values over one axis or one sort
- the overall minimum

The obtained values may be applied to determine a split axis or the final distribution (on a chosen split axis). The best overall performance resulted from the following algorithm

- more sophisticated split
- combined with forced reinsertion

from Beckmann, Kriegel, Schneider, Seeger, The R*-tree: An Efficient And Robust Access Method for Points and Rectangles

Forced Reinsertion

Algorithm ReInsert

RI1 For all M+1 entries of a node N, compute the distance between the centers of their rectangles and the center of the bounding rectangle of N

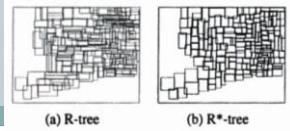
RI2 Sort the entries in decreasing order of their distances computed in RI1

RI3 Remove the first p entries from N and adjust the bounding rectangle of N

RI4 In the sort, defined in RI2, starting with the maximum distance (= far reinsert) or minimum distance (= close reinsert), invoke Insert to reinsert the entries

- reinsertion heuristic
- p = 30%

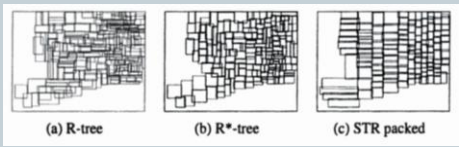
from Beckmann, Kriegel, Schneider, Seeger, The R*-tree: An Efficient And Robust Access Method for Points and Rectangles



(a) R-tree (b) R*-tree

Packing R-trees

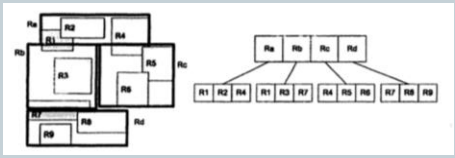
- for static set of data, can pre-compute optimal tree
- imagine B-tree on static set of data



(a) R-tree (b) R*-tree (c) STR packed

R+-trees

- R-tree without overlap of directory rectangles
- requires that rectangles are duplicated



- minimum m cannot be guaranteed
- point search/window search?
- adjusting rectangles after insertion can lead to deadlock

Sources

- Garcia-Molina, Ullman, Widom, Database Systems; the complete book, Pearson, 2009.
- Apostolos, Papadopoulos, Manolopoulos, Nearest Neighbor Search: a database perspective, Springer, 2005.