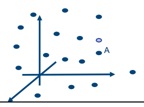
Challenge....

- Traditional data is one dimensional.
- Multimedia data is multi dimensional.
  - Ex. Maps are 2D
  - In general, if a given information has k features, it can be represented by a k-dimensional space



What kind of queries we can expect?

- Given a set of point in k-dimensional space
  - Exact match:
    - find if a given point is in the set or not
  - Nearest neihgbor:
    - find the closest point to a given point
  - Range search:
    - Given a region (rectangle or circle), find all the points in the given region

General approach

- Divide the space into regions
- Insert the new object into the corresponding region
- If the region is full, split the region
- retrieval: determine which regions are required to answer a given query and limit the search to these regions

Is there an alternative to multi-dimensional space decomposition?

- YES!
  - Convert a given k-D space to 1D space
  - We know how to handle 1D space!!
- Don't we loose information??
  - Yes, but if we are careful, we can minimize the information loss.

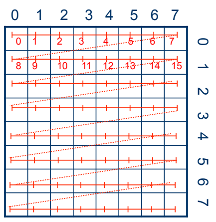
Maria Luisa Sapino (BDM 2018)

Space filling curves

- Convert a k-D space into 1D space such that points that are close to each other in k-D space are also close to each other in 1-D space

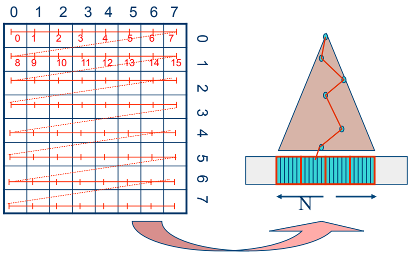
Maria Luisa Sapino (BDM 2018)

Row order/column order

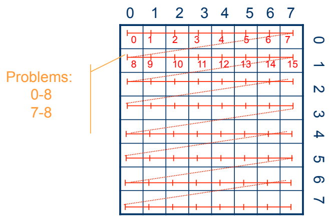


Maria Luisa Sapino (BDM 2018)

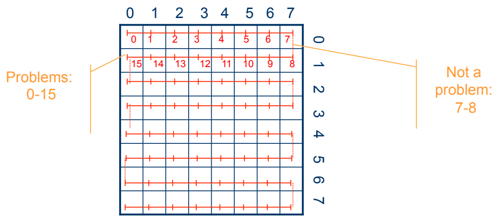
Row order/column order



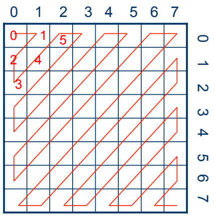
Row order/column order



Row prime order/column prime order

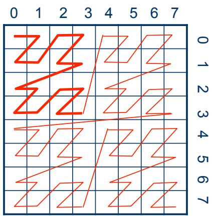


Cantor diagonal order



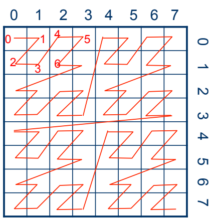
Maria Luisa Sapino (BDM 2018)

Z-order curve (hilbert curve)



Maria Luisa Sapino (BDM 2018)

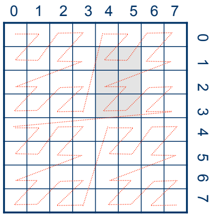
Z-order curve (hilbert curve)



Easy to compute (bit-shuffling): 1(001) X 2 (010) = 6 (000110)

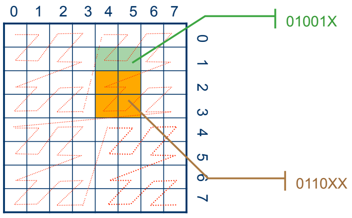
Maria Luisa Sapino (BDM 2018)

Z-order curve (hilbert curve)



Maria Luisa Sapino (BDM 2018)

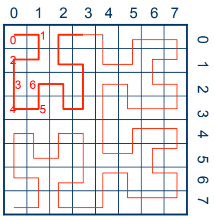
Z-order curve (hilbert curve)



Range search can be implemented using tries…

Maria Luisa Sapino (BDM 2018)

Peano-hilbert curve



Maria Luisa Sapino (BDM 2018)

Indexing

- What are we indexing???
  - Text → tries
  - Numbers, text → B-trees, B+ trees, B*trees
  - Images→???????????
- Which feature are we going to index on?
  - Color? Texture? Time? (image series)
- What do we need to specify?
  - Lines? Points? Space?

How do we index points?

- Given
  - a space of N-dimensions
  - M points
  - a distance function between points
- we can use multidimensional index structures
  - k-d trees
  - point quadtrees
  - MX quadtrees
  - R-trees
  - TV-trees
  - X-trees

So…

- we can answer queries of the form
  - Given
    - a point X in N-dimensional space

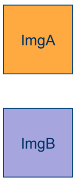
    Find
    - all points Y that are in its proximity ($d(X,Y) < \varepsilon$)

…thus…

- If
  - we represent any feature as a point in N-dimensional space (color, texture, shape, etc.)
  - we define a distance function between those points
    - (larger distance→lower similarity)
- Then
  - we can find media object with similar properties.

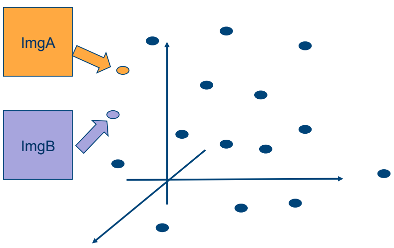
Maria Luisa Sapino (BDM 2018)

Populate database



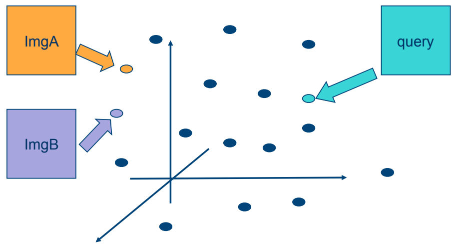
Maria Luisa Sapino (BDM 2018)

Populate database

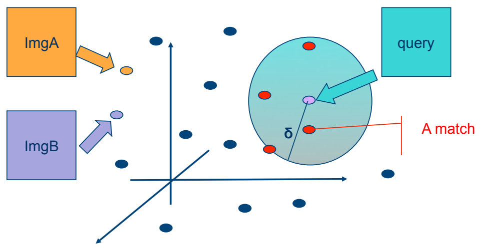


Maria Luisa Sapino (BDM 2018)

Map query image

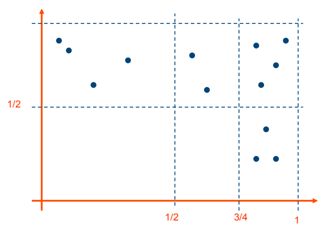


Range search



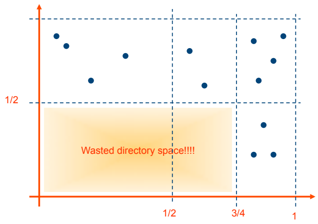
Grid File

- Every cell is one disk page



Grid File

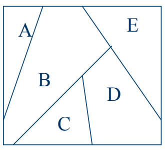
- Every cell is one disk page



Point Trees

How can we divide space?

- Let us assume that the space is 2-d
- There are many ways to divide the space
  - Fixed size squares
  - Triangles
  - Rectangles
  - Arbitrary space decomposition



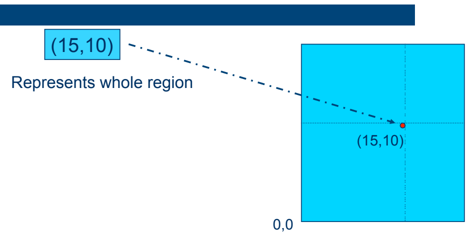
- Each line divides the space into two
  - Line: $n_1x+n_2y = c$
  - Regions: $n_1x+n_2y >= c$
    $n_1x+n_2y < c$

Point quadtrees (Finkel and Bentley 74)

- Key features:
  - Every node in a point quadtree *implicitly* represents a rectangular region.
  - Each node contains an *explicit* point labeling it.
  - Root represents the whole region.
  - Each node's region is split into 4 parts ("quadrants") by drawing a vertical and a horizontal line through the point labeling the node.
  - Each node has 4 children corresponding to the 4 "quadrants" above.

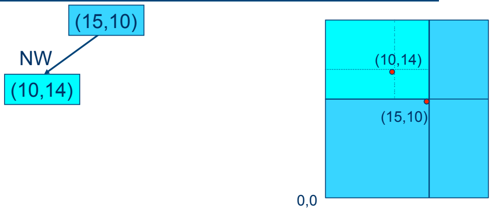
Maria Luisa Sapino (BDM 2018)

Point quadtrees: example



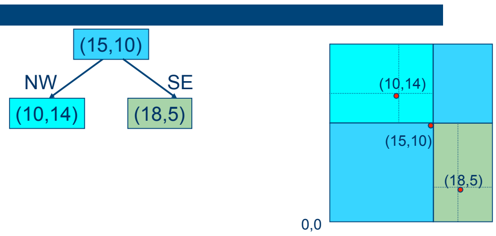
Maria Luisa Sapino (BDM 2018)

Point quadtrees: example

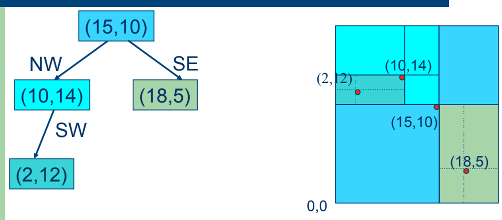


Maria Luisa Sapino (BDM 2018)

Point quadtrees: example



Point quadtrees: example



Observation

- The structure of the tree depends on the insertion order!!!!

- Exercise: try to insert nodes in the following order
  (18,5) (15,10), (2,12) (10,14)
  and compare the resulting tree with the previous one.

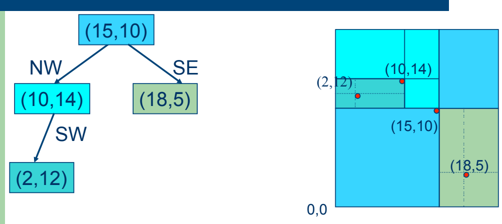
Key Points

- Suppose a point quadtree has N nodes in it.
- Worst case height = N.
- Worst case insertion time = N.
- Other operations are:
  - Deletion: delete a point
  - Range query: find all points within a given region
  - NN query: find the nearest neighbor (or M nearest neighbors) of a given point.

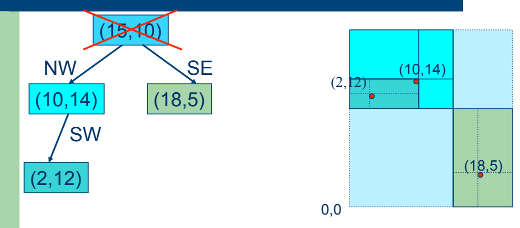
Deletion

- Suppose T is the root of a point quadtree and you want to delete (x,y).
- Steps:
  - Find (x,y) by doing a search.
  - If it is a leaf node, then simply set the appropriate link field of its parent to nil (and return the node to available storage).
  - What if it is not a leaf ?

Delete (15,10)

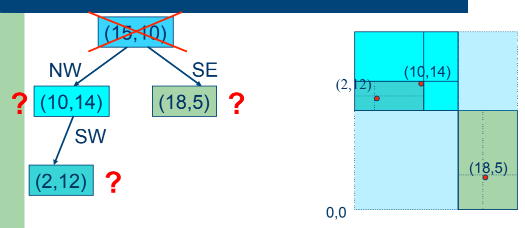


Delete (15,10)



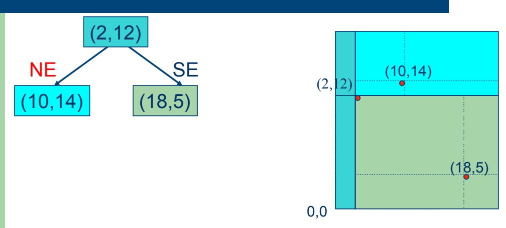
Maria Luisa Sapino (BDM 2018)

Delete (15,10)



Maria Luisa Sapino (BDM 2018)

Let us choose (2,12)



Maria Luisa Sapino (BDM 2018)

K-Nearest Neighbor Search

- This is the most important operation.
- Given a query point Q, find the K closest (to Q) points in the point quadtree.
- For simplicity, we will focus on K=1. Easy to generalize to K > 1.

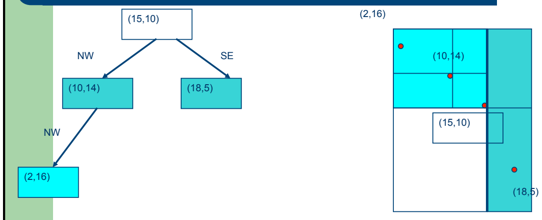
K-NN Search

- S – a set
- A metric d: S x S -> **N** is a mapping s.t.
  - d(x,x) = 0
  - d(x,y) = d(y,x)
  - d(x,y) + d(y,z) >= d(x,z)
- We extend d to a function from S x2^S -> **N** as follows:
  - d(x,R) = MIN{d(x,y) | y is in R }

K-NN Search

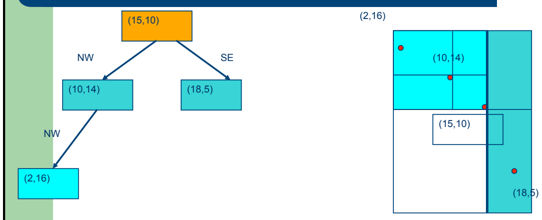
- Each node N *implicitly* represents a region N.reg.
- Algorithm for NN search works as follows.
  - Maintain variable bestdist (initialized to ∞)
  - Maintain variable bestSOL (initialized to NIL)
  - Algorithm visits nodes starting from root.
  - Everytime it visits a node N, it examines the point labeling that node. If d(Q,N.point) < bestdist, it updates bestdist and bestSol. Otherwise it continues.
  - Only nodes N such that d(Q,N.reg) < bestdist are visited. WHY?

NN Search example



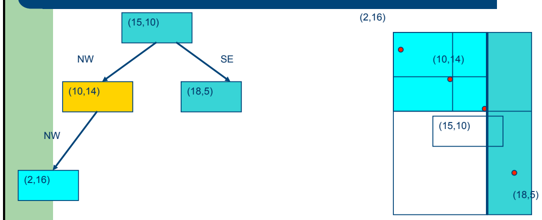
Q = (9,14) bestdist = ∞,   bestSOL = NIL

NN Search example



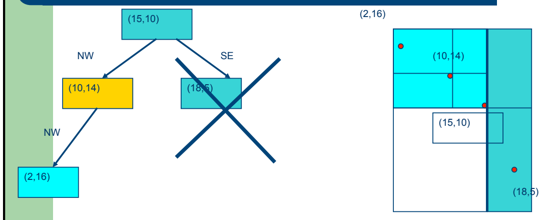
Q = (9,14) Visit root because d(Q,Root.reg) = 0 < bestDist
Set bestdist = SQRT(52), bestSOL = (15,10)

NN Search example



Q = (9,14). d(Q,Root.NW.Reg) = 0 < SQRT(52).

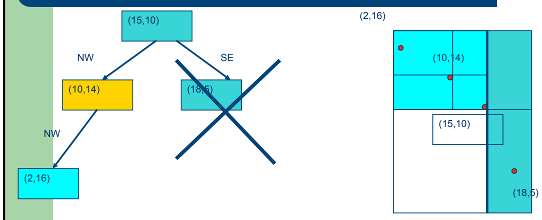
NN Search example



Q = (9,14). d(Q,Root.NW.Reg) = 0 < SQRT(52).
d(Q,Root.SE.Reg) is not less than SQRT(52) – so pruned away.

Maria Luisa Sapino
(BDM 2018)

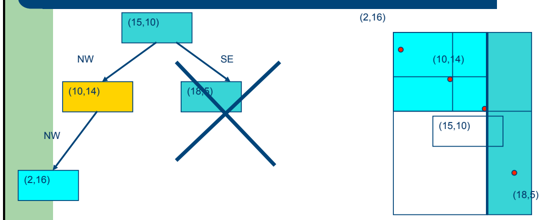
NN Search example



Q = (9,14). d(Q,(10,14)) = 1.
Bestdist = 1, bestsol = (10,14)

Maria Luisa Sapino
(BDM 2018)

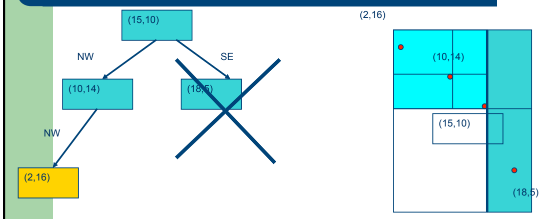
NN Search example



Q = (9,14). d(Q,Root.NW.NW.reg) = 0 < bestDist.

Maria Luisa Sapino
(BDM 2018)

NN Search example



Q = (9,14). So must search. D(Q,(2,16)) > 1.
Algorithm terminates as no more nodes to search.

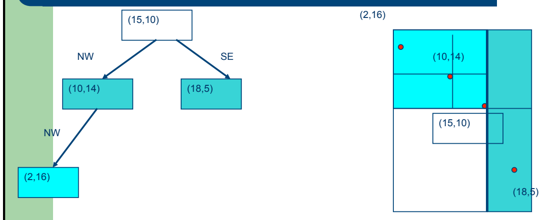
K-NN Search

- Want to find k-close points to Q.
- Maintain an array SOL[1,..,K] containing K points. Initialize all entries in array to NIL.
- SOL[1] is the closest point to Q found so far, SOL[2] is the second closest, etc.
- Let bestdist = dist(Q,SOL[K])).
- Can prune a node N if dist(Q,N.Reg) >= bestdist.

Range Search

- Done for the sake of completeness.
- Given query point Q and a distance R, find all points in the tree that are within R units of Q.
- The query defines a circular region, C(Q,R) of radius R centered at point Q.
- Prune node N if N.reg does not intersect C(Q,R).
- Sometimes, we modify the above to a weaker pruning condition: Prune node N if N.reg does not intersect bb(C(Q,R)) where bb is the bounding box operator.
- Second condition also yields correct answer, but may prune less.

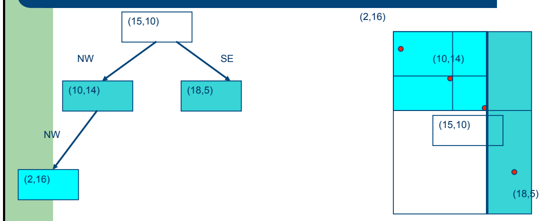
Range Search example



Q = (9,14), R = 3, SOL = {}. Region of root intersects C(Q,R) so "visit" the root.

Maria Luisa Sapino
(BDM 2018)

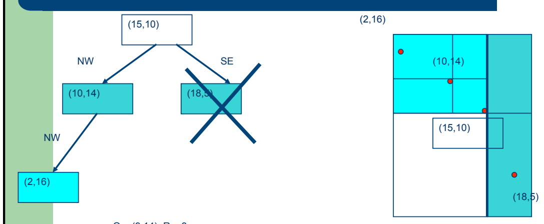
Range Search example



Q = (9,14), R = 3. D(Q,(15,10)) > 3. SOL stays {}.

Maria Luisa Sapino
(BDM 2018)

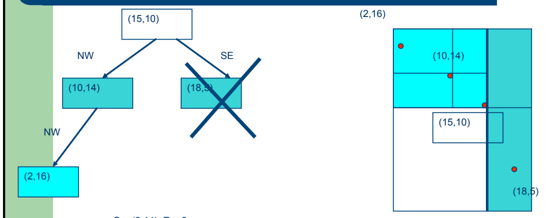
Range Search example



Q = (9,14), R = 3.
D(Q,Root.SE) > 3 so prune SE child.
D(Q,Root.NW) = 0. Visit NW child.

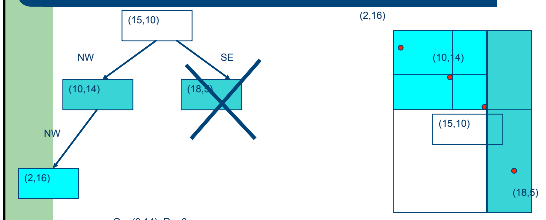
Maria Luisa Sapino
(BDM 2018)

Range Search example



Q = (9,14), R = 3.
Visit NW child. D(Q,(10,14)) < 3. SOL = {(10,14)}
D(Q,Root.NW.NW) = 0. So visit NW child of (10,14)

Range Search example



Q = (9,14), R = 3.
Visit NW child. D(Q,(2,16)) > 3. SOL = {(10,14)}
No more nodes to visit so return SOL and halt.

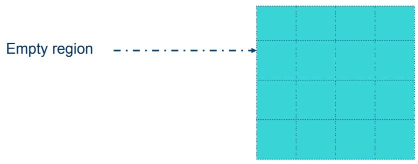
Problems with Point Quadtrees

- Deletion is slow.
- Tree can be highly unbalanced.
- Size of regions associated with nodes can vary dramatically.
- All these factors make the time taken to compute NN and range queries unpredictable.

MX quadtrees

- In point quadtrees, the region is split by drawing a vertical and a horizontal line through the point labeling node N.
- In MX-quadtrees,
  - the entire space is a $2^n \times 2^n$ matrix.
  - region is split by drawing a vertical and a horizontal line through the center of the region.

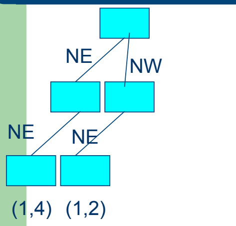
MX quadtrees: example

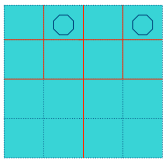


MX quadtrees: example

Insert (1,4)

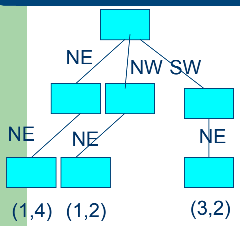
MX quadtrees: example

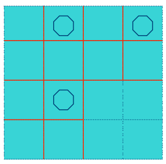




Insert (1,2)

MX quadtrees: example





Insert (3,2)

MX-quadtrees: salient features

- Each node represents a region.
- Root (level 0) represents 2^n x 2^n region.
- Nodes at level j represent 2^{n-j} x 2^{n-j} region.
- Points label leaf nodes (at level n).
- Insertion takes time O(n).
- So does search for a point.

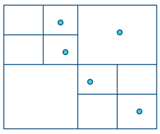
MX-Quadtrees: deletion

- Very easy to delete a point.
- First search for the point (which must be a leaf) and delete the leaf.
- If the parent now has 4 empty child fields, then delete the parent. And repeat as long as possible. This process is termed "collapsing".

Maria Luisa Sapino (BDM 2018)

PR-quadtrees

- MX-quadtree works well if the data is discrete
  - otherwise, it may need to use buckets, which may increase search time
- PR-quadtree (point region quadtree) assumes a continous space.



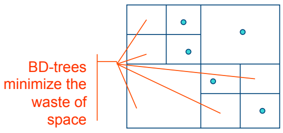
Structure is independent of insertion order

Deletion is easy

Maria Luisa Sapino (BDM 2018)

PR-quadtrees

- MX-quadtree works well if the data is discrete
  - otherwise, it may need to use buckets, which may increase search time
- PR-quadtree (point region quadtree) assumes a continous space.



Structure is independent of insertion order

Deletion is easy

Maria Luisa Sapino (BDM 2018)

KD-trees

- Deficiencies of quadtree:
  - each node requires k comparisons
  - each leaf contains k null pointers
  - node size gets larger as k increases

KD-trees

- Deficiencies of quadtree:
  - each node requires passk comparisons
  - each leaf contains k null pointers
  - node size gets larger as k increases
- Solution: KD-tree
  - the tree is binary whatever k is!!!
  - each node has two pointers only

K-d trees

- Used to store K-dimensional data, I.e. points of the form $(x_0,\ldots,x_{K-1})$
- Assuming the root is a level 0 node, each node at level i discriminates on $x_{i \bmod K}$.
- Always split region associated with a node into two parts.
- We now focus on K=2.

2-d-trees

- Structure of a node in a 2-d-tree
  - nodetype =record
    - INFO: infotype; (information content. It depends on the application domain)
    - xcoord:real; ycoord:real; (coordinates of the point associated to the node)
    - Llink: @nodetype;
    - Rlink: @nodetype;
  - end

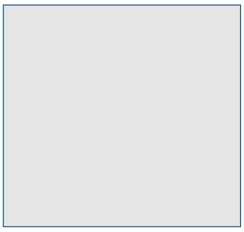
2-d-tree

- 2-d tree is a binary tree such that:
  - If N is a node such that level(N) is even , then for every node M in the subtree rooted at N.Llink, and for every node P in the subtree rooted in N.Rlink,
    - M.xcoord < N.xcoord P.xcoord>= N.xcoord
  - If N is a node such that level(N) is odd , then for every node M in the subtree rooted at N.Llink, and for every node P in the subtree rooted in N.Rlink,
    - M.ycoord < N.ycoord P.ycoord>= N.ycoord

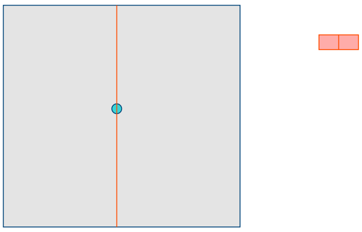
2-d-tree (notes)

- Every node partitions the space in 2 parts:
  - Nodes whose level is even, implicitly draw a vertical line, x=xcoord,
  - Nodes whose level is odd, implicitly draw a horizontal line, y=ycoord.

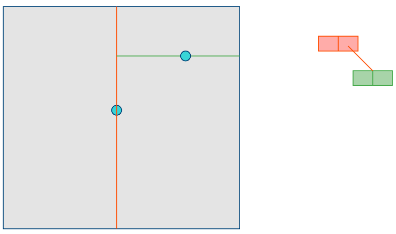
KD-trees



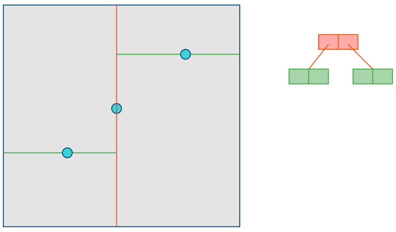
KD-trees



KD-trees

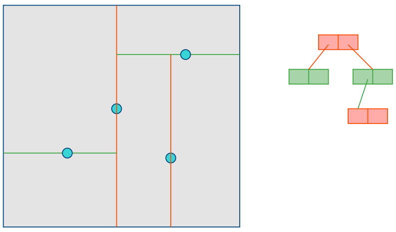


KD-trees



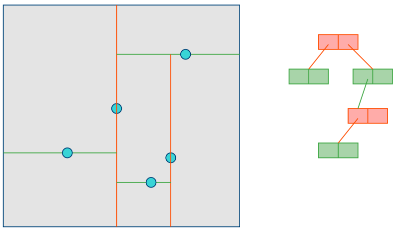
Maria Luisa Sapino (BDM 2018)

KD-trees



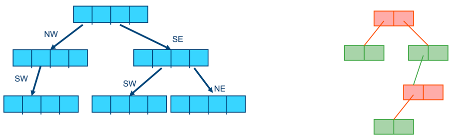
Maria Luisa Sapino (BDM 2018)

KD-trees



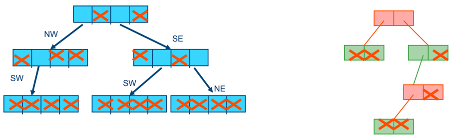
Maria Luisa Sapino (BDM 2018)

KD-trees



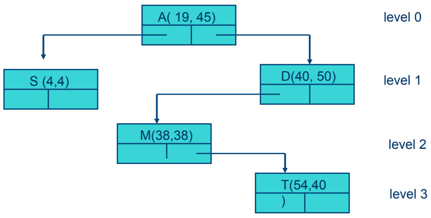
Maria Luisa Sapino (BDM 2018)

KD-trees



Maria Luisa Sapino (BDM 2018)

2-d-tree (example)



Maria Luisa Sapino
(BDM 2018)

range-queries

- Given a point (x_c, y_c) and a distance *d,* find the set of all points (x,y) such that (x,y) lies within distance *d* of (x_c, y_c).
  - Each node N implicitly represents a region R_N , constrained by N's coordinates and its parent's coordinates.
  - If the circle specified in the query has no intersection with R_N ,then there is no point searching the subtree rooted at N.
  - example: search for the circle with center (35,46)and radius 9.5 (returned: M(38, 38))

K-d-tree, k>2

- Extensions of 2-d trees, in which
  - Fields xcoord and ycoord in the definition of the node are replaced by a single field COORD, a vector of k elements.
    - For every node N, let i=level(N) mod k.
    - For every node M in N's left subtree:
      - M.VAL[i] < N.val[i]
    - For every node M in N's right subtree:
      - M.VAL[i] >= N.val[i]

…notes

- k-d trees are easy to implement
- A tree with *k* nodes can have height *k*
  - --> insertion and deletion can be expensive
- range searching costs, in the worst case, O(k $n^{1-1/k}$)

R-trees

- R-trees are used to store *two* dimensional rectangle data.
- They can be easily generalized to higher dimensions.
- R-trees themselves generalize the well known B-trees.

Node capacity

- Each node in a R-tree can contain upto N rectangles.
- But in addition, each node must contain *at least* N/2 rectangles.
- We will assume henceforth that N >= 4.

Node structure

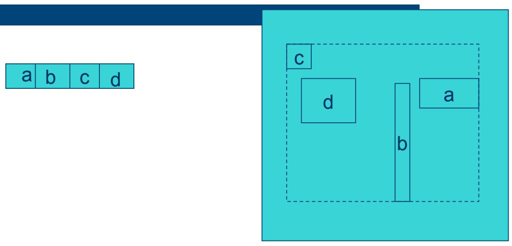
- Each node has between N/2 and N rectangles.
- Like a B-tree:
  - All leaves are at the same level
  - Root has at least two children unless it's a leaf

Node properties

- Each node *implicitly* represents a region.
- Root represents the whole space.
- The region of a node N, N.reg, is the bounding box of the rectangles stored at that node.
- Unlike quadtrees, it is possible for regions of siblings to intersect.

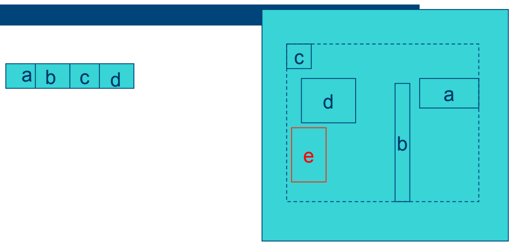
Maria Luisa Sapino (BDM 2018)

Example R-tree



Maria Luisa Sapino (BDM 2018)

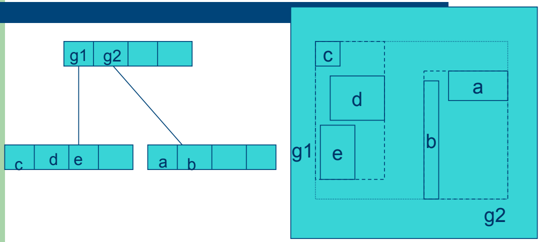
Example R-tree



No space in root ! Must split

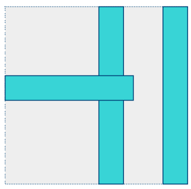
Maria Luisa Sapino (BDM 2018)

Example R-tree

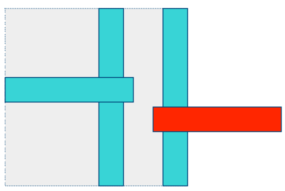


Minimize the sum of the areas of the two BRs.

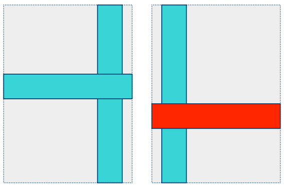
Example: Let max size be 3



How do we split???



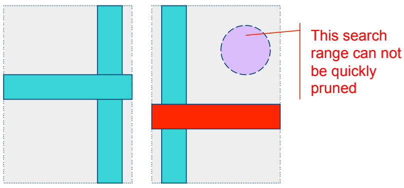
How do we split???



Minimize overlap of the BRs

Maria Luisa Sapino (BDM 2018)

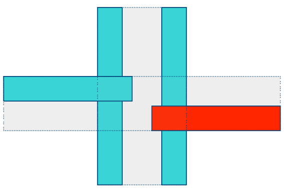
How do we split???



Minimize overlap of the BRs

Maria Luisa Sapino (BDM 2018)

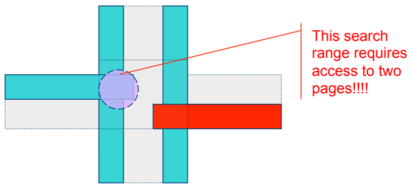
How do we split???



Minimize total area

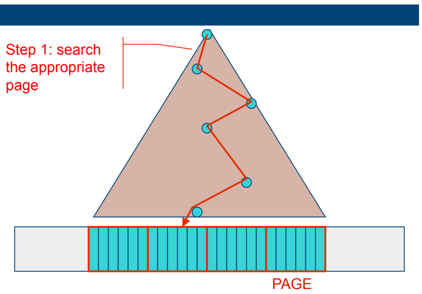
Maria Luisa Sapino (BDM 2018)

How do we split???

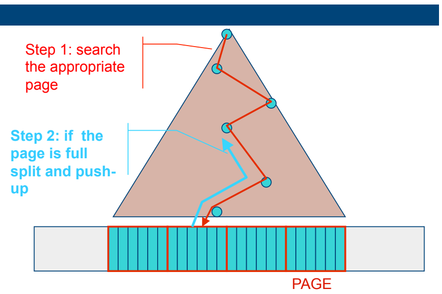


Minimize total area

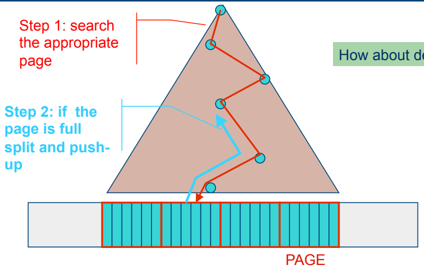
Insertion (similar to B-trees)



Insertion (similar to B-trees)



Insertion (similar to B-trees)

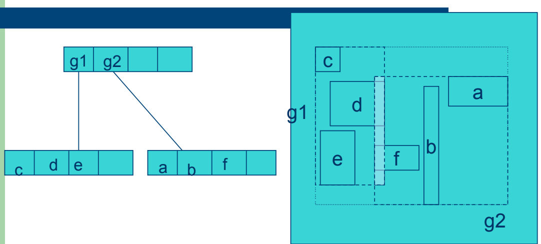


How about deletion??

R+-tree

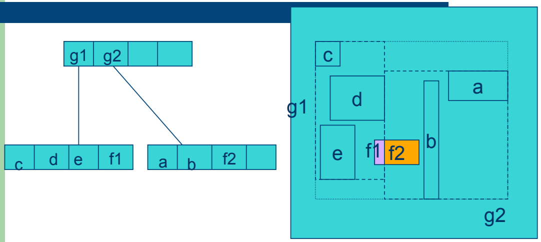
- Overlaps are bad…..
- ..so, let’s eliminate overlaps

Overlap in R-tree



The two BRs are overlapping.

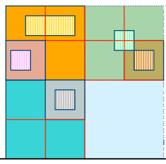
No-overlap in R+-tree



The two BRs are not-overlapping.

Other range/region index structures

- Range-tree, 2D-range tree
  - Precise, too much overhead
- MX-CIF quadtree
  - Regular division
  - Each rectangle is associated with the quadtree-page which covers it entirely



Nearest neighbor search

- ..no range given
  - first pick a (random) object $o \in D$ *and compute the distance dist(q, o)*….this is the first nearest neighbor candidate.
  - start a range search on the hierarchy using the range, *r = dist(q, o).*
  - whenever you find a data object *o* such that *dist(q, o) < r*, where *r* is the current nearest neighbor range, pick *o* is as the new nearest neighbor candidate
    - Set *dist(q, o)* as the new range, *r*

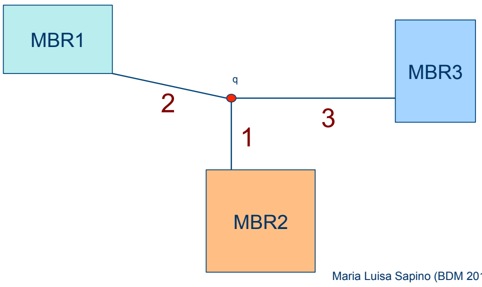
Nearest neighbor search

- ..great, but in which order do we visit the pages?
- ..how can we prune the pages that we have not visited yet most effectively??

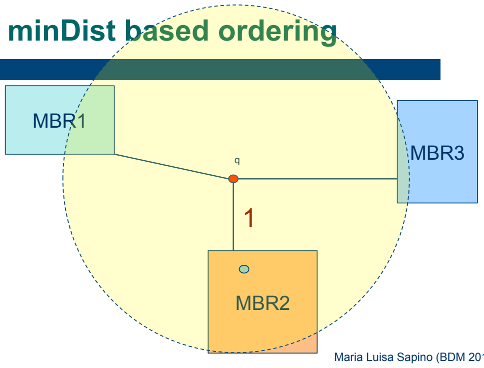
Nearest neighbor search

- Given q and MBR M
  - minDist(q,M)
  - minimum possible distance between the query and the objects contained within the MBR
  - minimum distance between q and any of the faces of M
- optimistic; yet minDist based ordering of the MBRs provides good pruning opportunities.

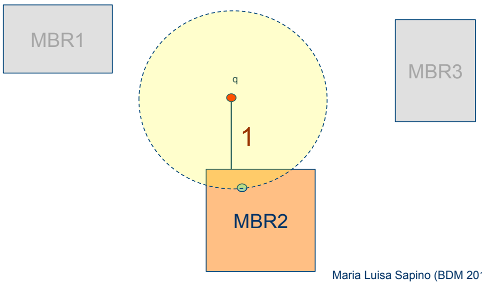
minDist based ordering



minDist based ordering



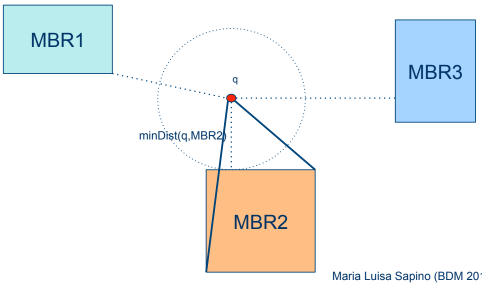
minDist based pruning



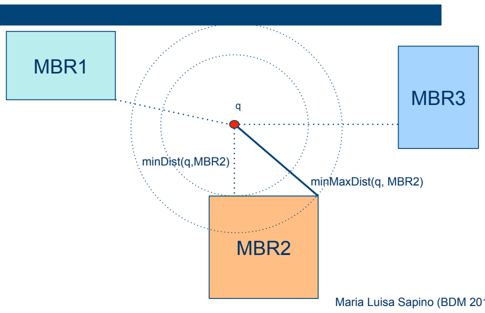
minMaxDist based pruning

- Given q and MBR M
  - minMaxDist(q,M)
  - upperbound on the distances
  - minimum among
    - maximum distances
      - on the closest faces of M on each dimension

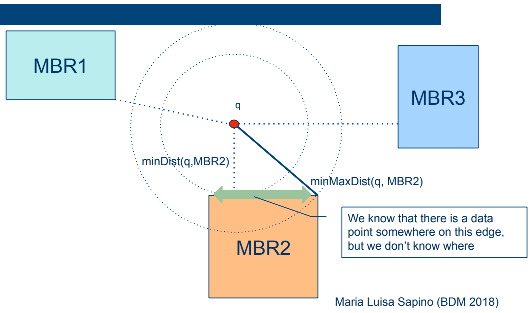
minMaxDist based pruning



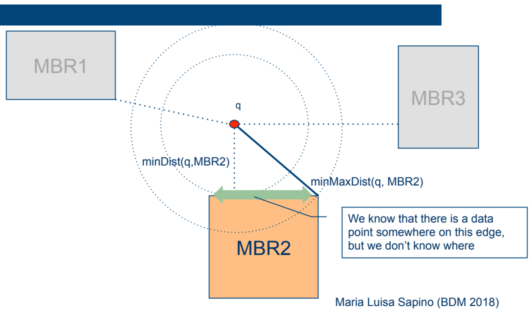
minMaxDist based pruning



minMaxDist based pruning

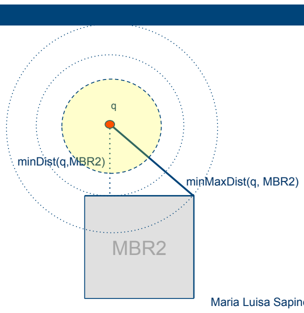


minMaxDist based pruning



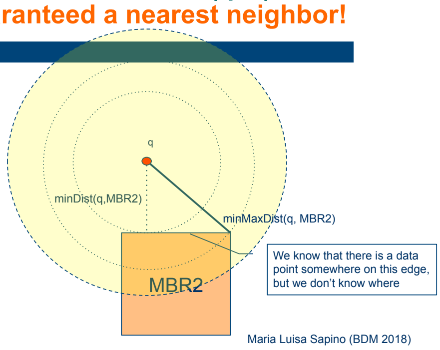
Maria Luisa Sapino (BDM 2018)

Case 1: r < minDist(q,M) ----- prune MBR



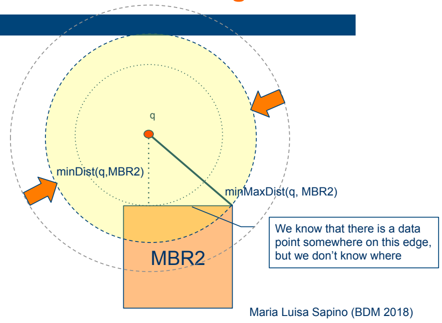
Maria Luisa Sapino (BDM 2018)

Case 2: r > minMaxDist(q,M) --- guaranteed a nearest neighbor!



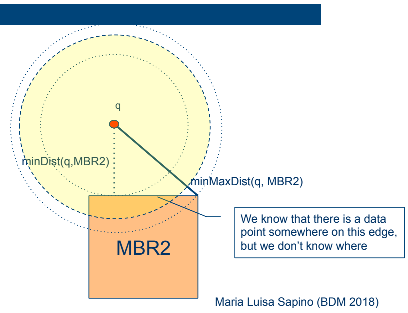
Maria Luisa Sapino (BDM 2018)

Case 2: r > minMaxDist(q,M) --- tighten the search range!



Maria Luisa Sapino (BDM 2018)

Case 3: minDist(q,M) <= r <= minMaxDist(q,M) maybe….we need to look into the MBR!



Maria Luisa Sapino (BDM 2018)

Nearest neighbor search

- Cannot prune an MBR as long as
  minDist(q,M) <= r <= minMaxDist(q,M)
- downward pruning: discard M if there exists M' s.t.
  minDist(q,M) > minMaxDist(q,M')
- downward pruning: prune candidate object o if there exists M s.t.
  dist(q,o) =r > minMaxDist(q,M)
- upward pruning: M is discarded if the current candidate is s.t.
  minDist(q,M) > r = dist(q,o)

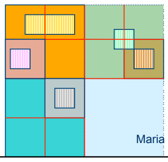
Maria Luisa Sapino (BDM 2018)

Nearest neighbor search

- What is we are looking for more than one, say *k*, nearest neighbors?
  - Maintain a list of *k* candidates in the memory
  - Always prune the search space using the current k^{th} best candidate
  - When you find an object better than the current k^{th} best candidate
    - Drop the current k^{th} best candidate
    - Include the new object in the list of *k* candidates
    - Identify the new k^{th} best candidate

Other range/region index structures

- Range-tree, 2D-range tree
  - Precise, too much overhead
- MX-CIF quadtree
  - Regular division
  - Each rectangle is associated with the quadtree-page which covers it entirely



X-tree

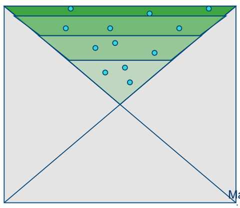
- Like R-trees, but
  - change the page size based on the depth to ensure that there is larger fanout higher in the tree structure
- A larger page size means multiple disk pages that are consecutively stored
  - so, no “page seek” penalty during disk access.

Dimensionality curse

- Exponential growth in the number of pointers needed, wasted storage,
- Exponential suqueries (quadtrees)
- Larger MBRs means smaller fanout in trees and this is bad

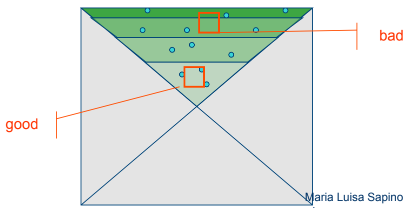
Pyramid tree

- If data is uniformly distributed, pages are likely to be of the same volume



Pyramid tree

- If data is uniformly distributed, queries likely to avoid thin pages, reducing the average access time

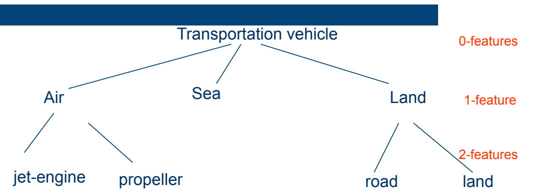


TV trees (telescopic vector trees)

(Lin, Jagadish, Faloutsos, VLDB Journal, 1994)

- Based on classification idea
- Dimensionality curse: R-trees do not work for large numbers of dimensions
- Idea:
  - not all features are equally important
  - order features based on importance (discrimination power)
  - use as little features as possible
  - "contract" and "extend" feature vectors based on need

Intuition



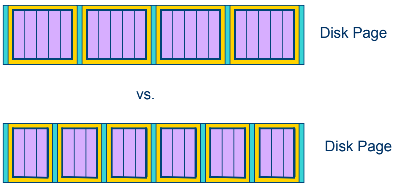
Classification requires less features at the higher levels than it uses at the lower levels

Cost of a dimension

- Every rectangle has to have values describing all its dimensions

Cost of a dimension

- Every rectangle has to have values describing all its dimensions

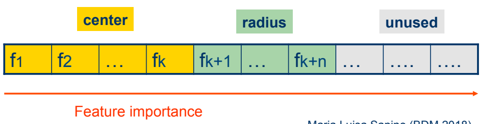


TV-trees

- Hierarchical
  - Leaves: objects (documents)
  - Internal nodes: Minimum Bounding Regions
    - Higher fan-out at the root
    - Lower fan-out at the leaves (or lower levels)

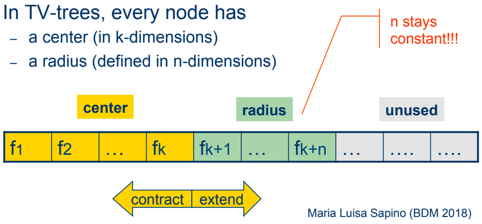
Node structure in TV-trees

- In R-trees, every node is a hyper-rectangle
- In TV-trees, every node has
  - a center (in k-dimensions)
  - a radius (defined in n-dimensions)

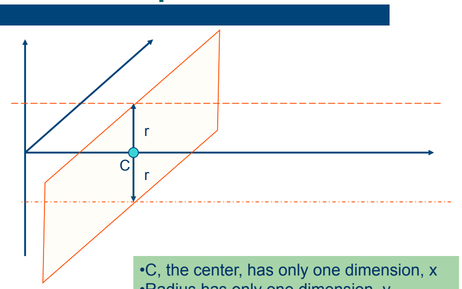


Node structure in TV-trees

- In R-trees, every node is a hyper-rectangle
- In TV-trees, every node has
  - a center (in k-dimensions)
  - a radius (defined in n-dimensions)

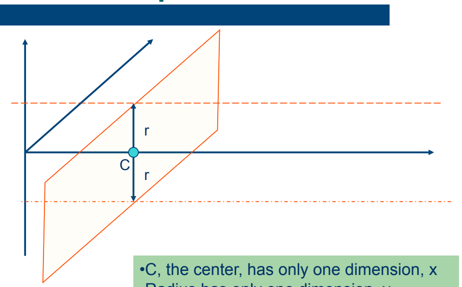


TV trees: example



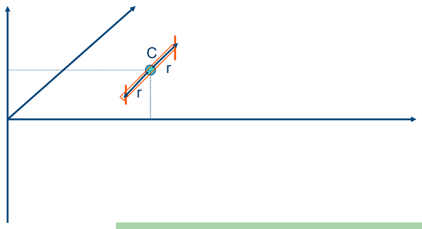
- C, the center, has only one dimension, x
- Radius has only one dimension, y

TV trees: example



- C, the center, has only one dimension, x
- Radius has only one dimension, y
- .....any z is okay

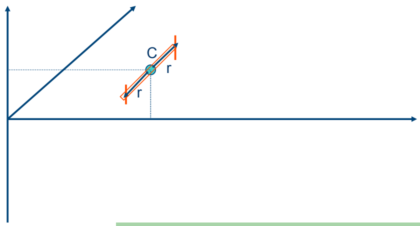
TV trees: extension example



•C, the center, has only two dimensions, x,y
•Radius has only one dimension, z

Maria Luisa Sapino (BDM 2018)

TV trees: extension example



•C, the center, has only two dimensions, x,y
•Radius has only one dimension, z
•....any z is not okay!!!!!

Drawback

- Information about the behaviour of single attributes, e.g., their selectivity, is required

Maria Luisa Sapino (BDM 2018)

X-tree

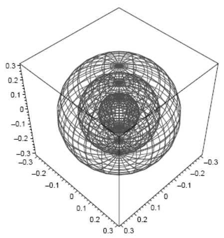
- Like R-trees, but
  - change the page size based on the depth to ensure that there is larger fanout higher in the tree structure
- A larger page size means multiple disk pages that are consecutively stored
  - so, no “page seek” penalty during disk access.

Dimensionality curse

- Exponential growth in the number of pointers needed, wasted storage, exponential subqueries (quadtrees)
- Larger MBRs means smaller fanout in trees and this is bad
- …and…

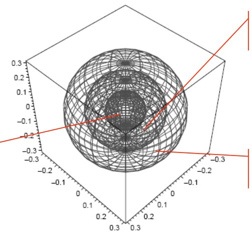
Dimensionality Curse

- Consider a query point and three alternative ranges:



Dimensionality Curse

- Consider a query point and three alternative ranges:



Maria Luisa Sapino (BDM 2018)

Dimensionality Curse

- In n-dimensional space, if the number of points in the inner most sphere is *l*, then
  - number of points in the second slice is $O(2^{n-1}\, l)$
  - number of points in the third slice is $O(3^{n-1}\, l)$
  - number of points in the fourth slice is $O(4^{n-1}\, l)$
- This means that most of the points lie in the outermost slice!!!!

Maria Luisa Sapino (BDM 2018)

Pyramid trees (Berchtold, Bohm, Kriegel, SIGMOD98)

- Motivation: drawbacks of already existing multidimensional index structures
  - Querying and indexing techniques which provide good results on
    - low-dimensional data do not perform sufficiently well on multi-dimensional data (curse of dimensionality)
  - high cost for insert/delete operations
  - Poor support for concurrency control/recovery

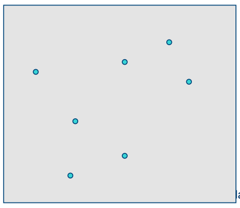
Maria Luisa Sapino (BDM 2018)

Pyramid tree

- Space-filling curves were using B-trees
- Pyramid trees also do the same..without space filling curves

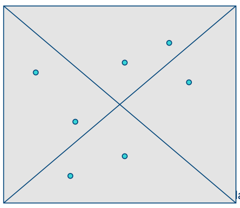
Pyramid tree

- Space-filling curves were using B-trees
- Pyramid trees also do the same..without space filling curves



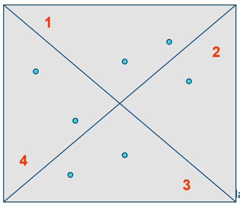
Pyramid tree

- Space-filling curves were using B-trees
- Pyramid trees also do the same..without space filling curves



Pyramid tree

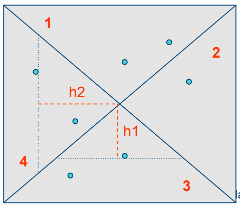
- Space-filling curves were using B-trees
- Pyramid trees also do the same..without space filling curves



aria Luisa Sapino (BDM 2018)

Pyramid tree

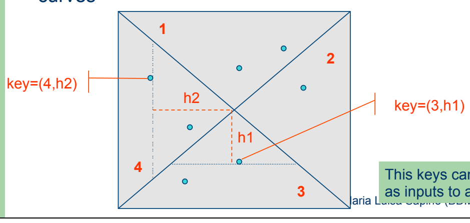
- Space-filling curves were using B-trees
- Pyramid trees also do the same..without space filling curves



aria Luisa Sapino (BDM 2018)

Pyramid tree

- Space-filling curves were using B-trees
- Pyramid trees also do the same..without space filling curves

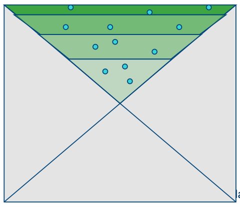


This keys can be used as inputs to a B-tree!!!

aria Luisa Sapino (BDM 2018)

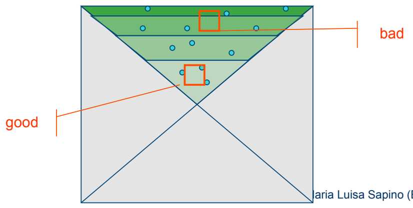
Pyramid tree

- If data is uniformly distributed, pages are likely to be of the same volume



Pyramid tree

- If data is uniformly distributed, queries likely to avoid thin pages, reducing the average access time



Other index structures

- Grids
- VA-files
  - extension of the grid idea..
- SR-, SS-trees
  - like R-trees
  - use spheres instead of rectangles
- X-trees
  - like R-trees
  - change the page size based on the depth