Data Intensive Systems (DIS)
KBH-SW7 E25

12. R-tree based Spatial Query Processing

AALBORG UNIVERSITY

Agenda

- Nearest Neighbor Queries
  - Distance Metrics in R-tree
  - Branch-and-Bound NN Search
  - Best-First NN Search
- Spatial Join
- Implementation of R-tree

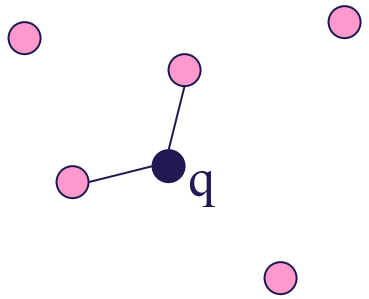
Tobler's First Law of Geography

- “Everything is related to everything else, but near things are more related than distant things.”
  - **Waldo Tobler**, an American-Swiss geographer and cartographer.

- Distance between spatial objects really matters

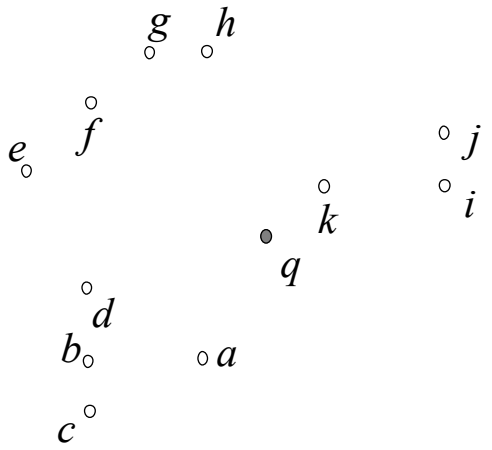
Nearest Neighbor Query

- Nearest neighbor query
  - Given a spatial relation R and a query object q, find the nearest neighbor (NN) of q in R
  - Formally: NN(q,R) = o $\in$ R: dist(q,o) $\leq$ dist(q,o'), $\forall$ o' $\in$ R
- Note:
  - We can have more than one NN (with equal minimum distance)
- Simplification
  - We usually focus on point-sets
  - NN search return any NN, if there are ties



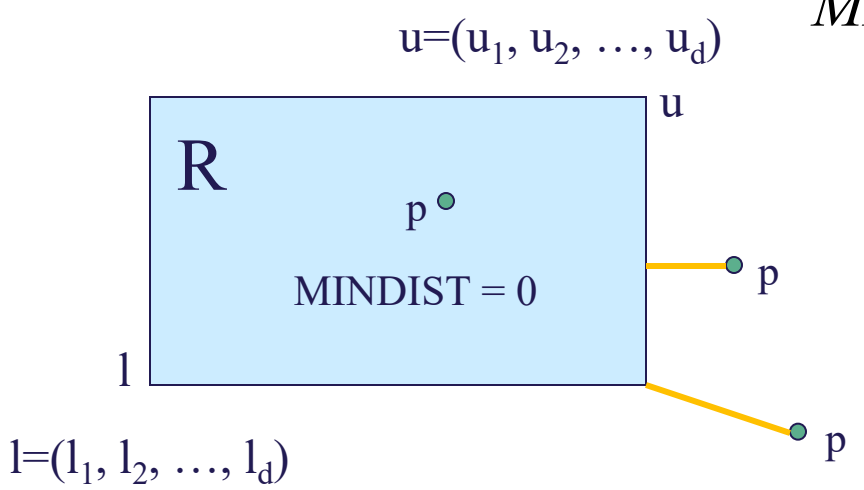
Straightforward Solution for NN search

- Problem setting
  - Given a spatial relation R of objects
  - Given a query point q, find the NN of q in R
- Straightforward solution
  - Compare q with each object in R
  - Keep track of the object with minimum distance to q
  - Incurs high cost when the size of R is very large
- Our goal: reduce the cost of NN search, by using the R-tree



MINDIST Computation

- Think of an MBR R that contains a number of points
- MINDIST(p, R) is the minimum distance between p and R with corner points l and u
  - the closest point in R is at least this distance away



$$MINDIST(p,R) = \sqrt{\sum_{i=1}^{d} (p_i - r_i)^2}$$

$$r_i = l_i, \text{ if } p_i < l_i$$

$$= u_i, \text{ if } p_i > u_i$$

$$= p_i, \text{ otherwise}$$

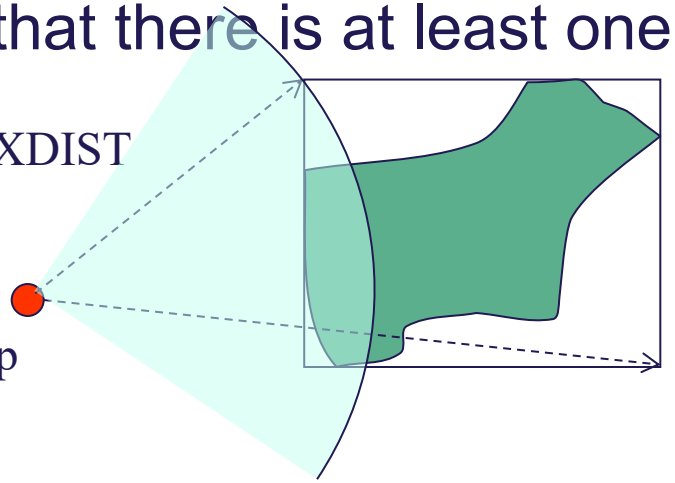
$$\forall o \in R, MINDIST(p,R) \leq \|(p,o)\|$$

MINMAXDIST

- MINMAXDIST(p, R) definition
  - For each dimension, find the closest face, compute the distance to the furthest point on this face.
  - Take the minimum of all these (d) distances.
- MINMAXDIST(p, R) is the smallest possible *upper bound* of distances from p to R

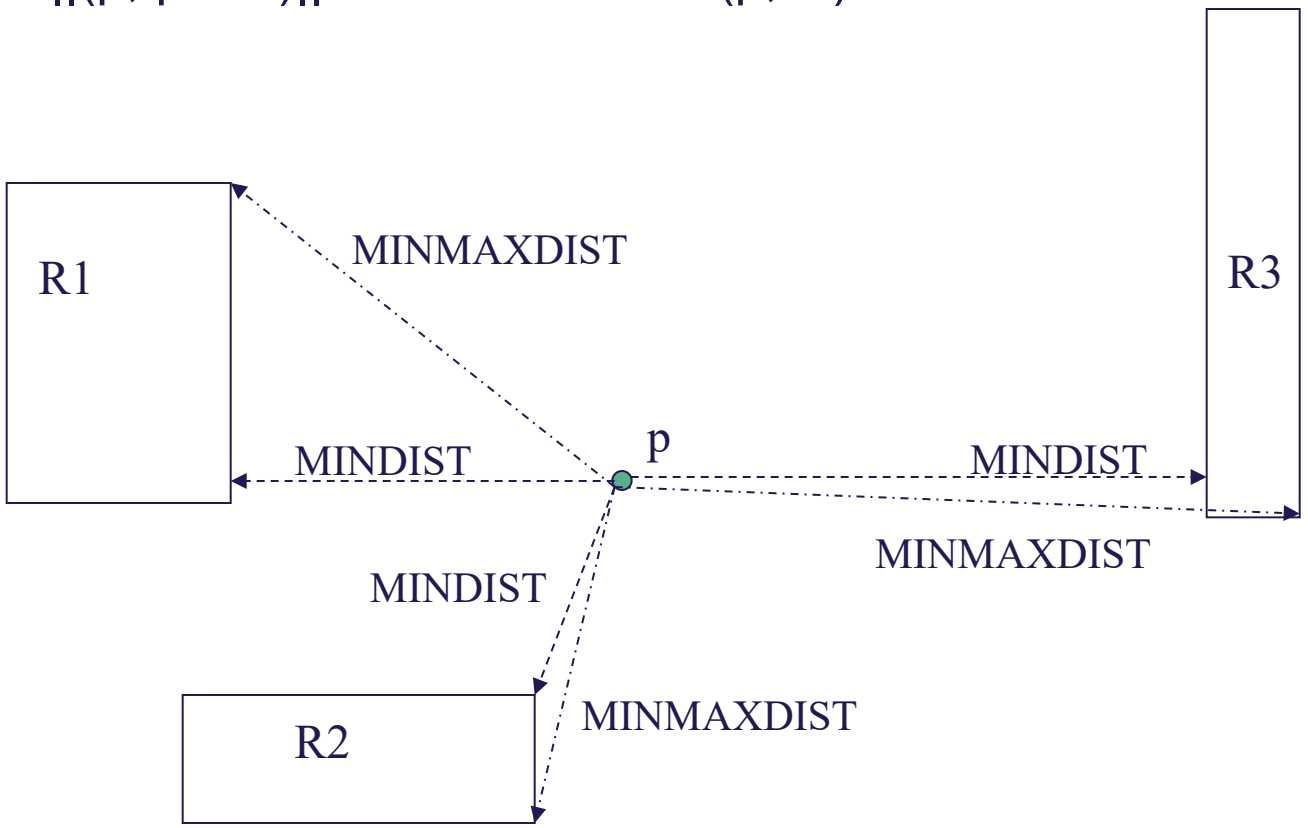
$$\exists o \in R, \|(p,o)\| \leq MINMAXDIST(p,R)$$

- MINMAXDIST guarantees that there is at least one object in R with a distance to p smaller or equal to it.



MINDIST and MINMAXDIST

- MINDIST(p, R) ≤ ||(p, p.NN)|| ≤ MINMAXDIST(p, R)



Agenda

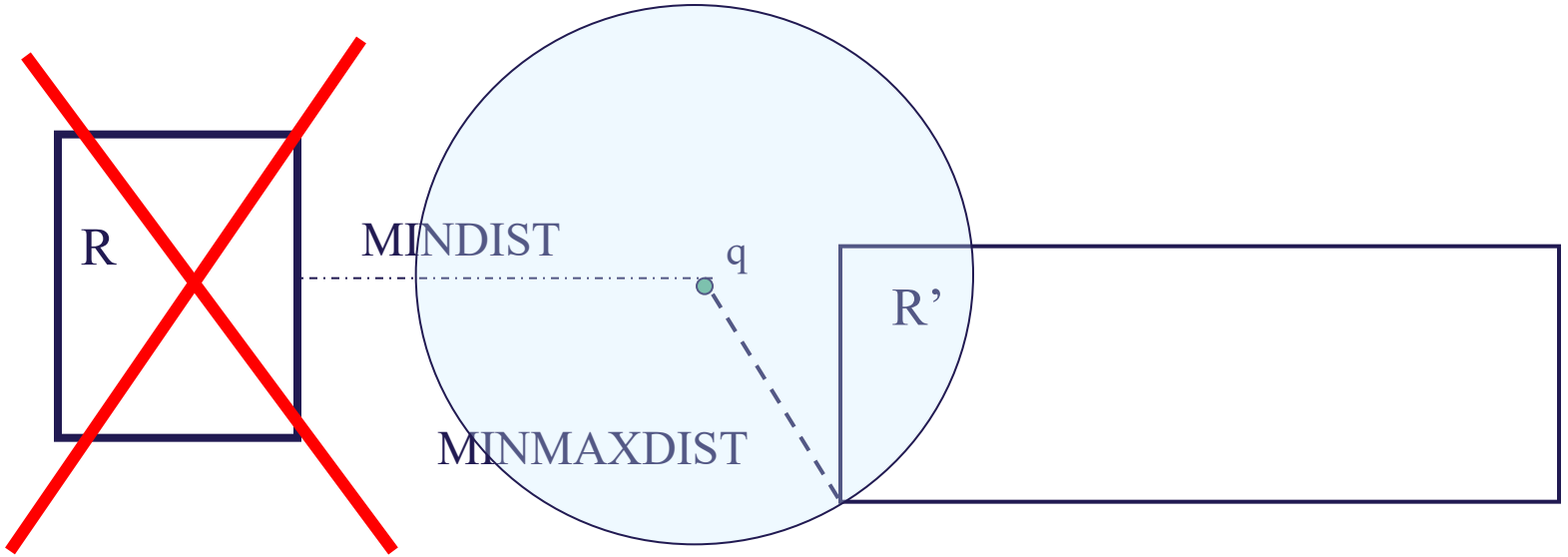
- Nearest Neighbor Queries
  - Distance Metrics in R-tree
  - Branch-and-Bound NN Search
    - Depth-First Search
  - Best-First NN Search
- Spatial Join
- Reversed Nearest Neighbor Query
- Implementation of R-tree

Pruning in NN Search

- Downward pruning 1
  - An MBR R is discarded if there exists another R' s.t. MINDIST(q, R) > MINMAXDIST(q, R')
- Downward pruning 2
  - An object o is discarded if there exists an R s.t.
    the Actual-Dist(q, o) > MINMAXDIST(q, R)
- Upward pruning
  - An MBR R is discarded if an object o is found s.t.
    MINDIST(q, R) > Actual-Dist(q, o)

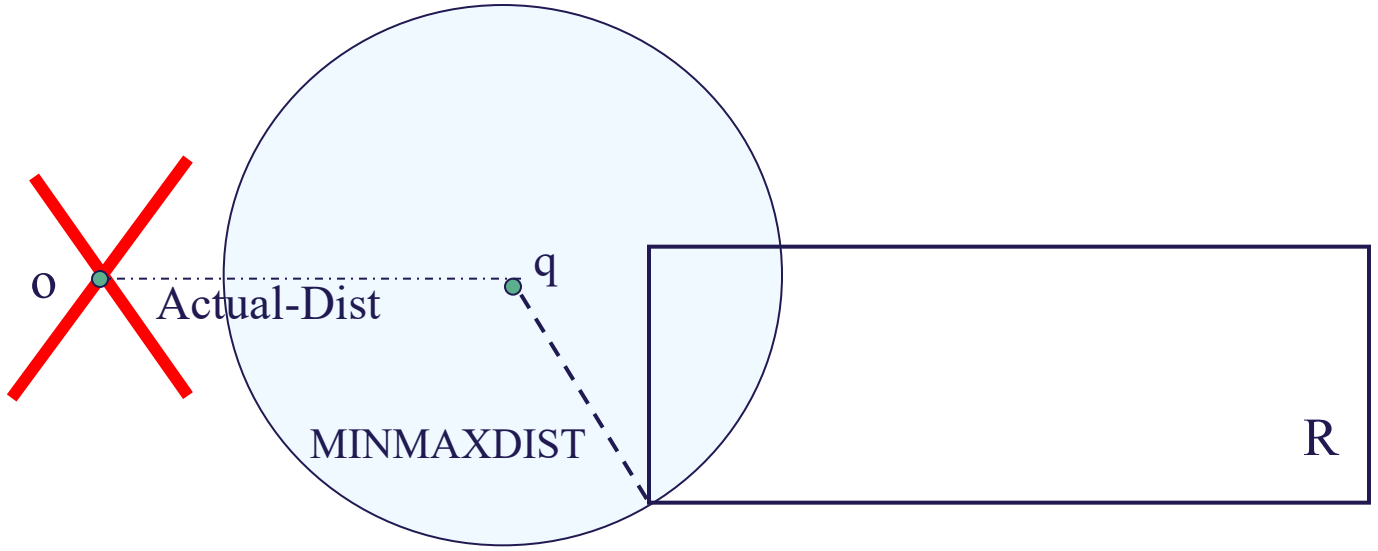
Pruning 1 Example

- Downward pruning 1
  - An MBR R is discarded if there exists another R' s.t. MINDIST(q, R) > MINMAXDIST(q, R')



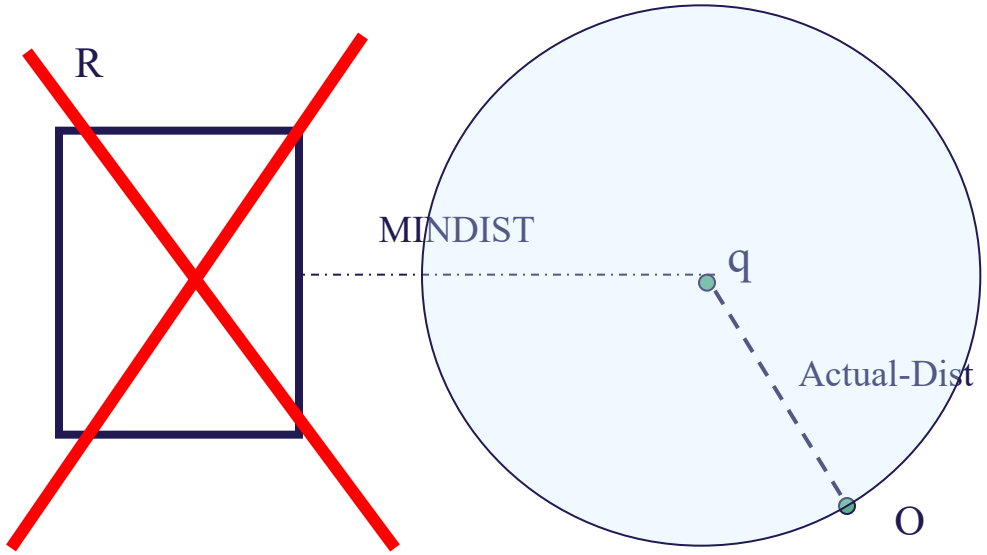
Pruning 2 Example

- Downward pruning 2
  - An object o is discarded if there exists an R s.t. the Actual-Dist(q, o) > MINMAXDIST(q, R)



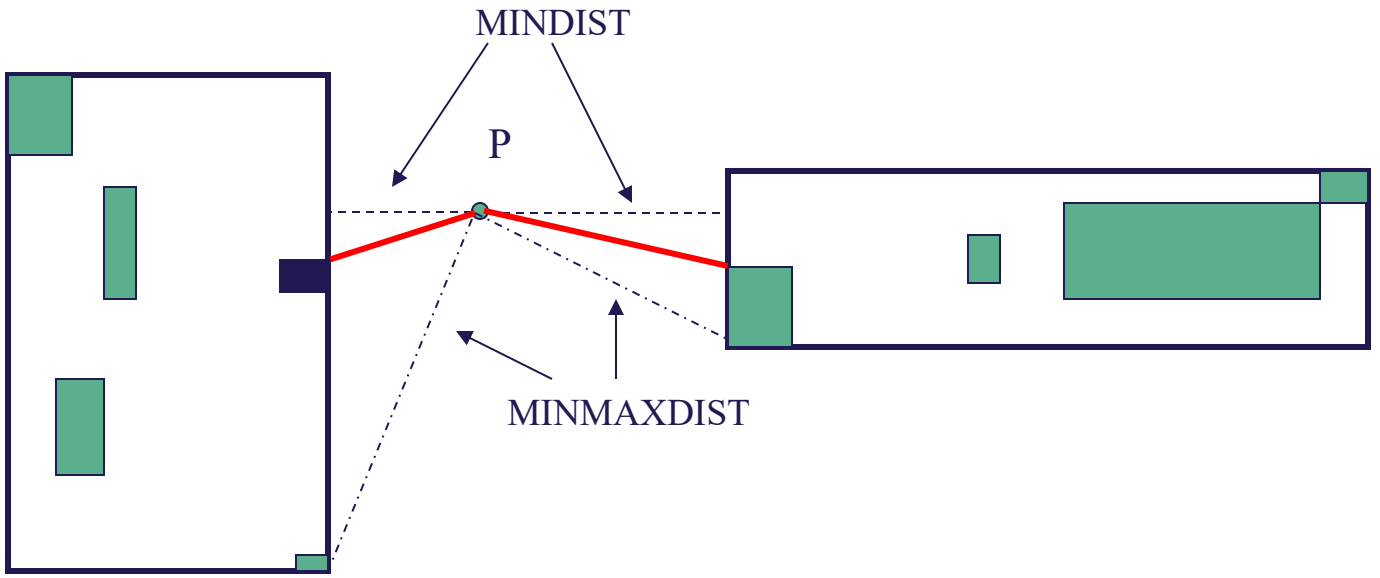
Pruning 3 Example

- Upward pruning
  - An MBR R is discarded if an object o is found s.t. MINDIST(q, R) > Actual-Dist(q, o)



Ordering Distance

- MINDIST is an optimistic distance in the sense that it underestimates the distances between p and objects in an MBR.
- Whereas MINMAXDIST is a pessimistic one, in the sense that it overestimate such distances.

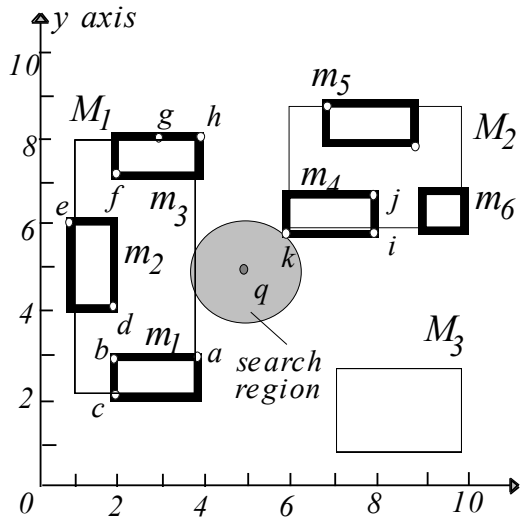


Branch-and-Bound NN Search Algorithm

1. Initialize the nearest neighbor distance (*nnd*) as infinite distance
2. Traverse the R-tree depth-first starting from the root. At each Index node, sort all MBRs using an ordering metric and put them in an **Active Branch List (ABL).**
3. Apply pruning rules 1 and 2 to ABL
4. Visit the MBRs from the ABL following the order until it is empty
5. If Leaf node, compute actual distances, compare with the best NN so far, update if necessary.
6. At the return from the recursion, use pruning rule 3
7. When the ABL is empty, the NN search returns.

- Sorting ABL is not necessary. We can even avoid ABL.

Branch-and-Bound NN Example (1)



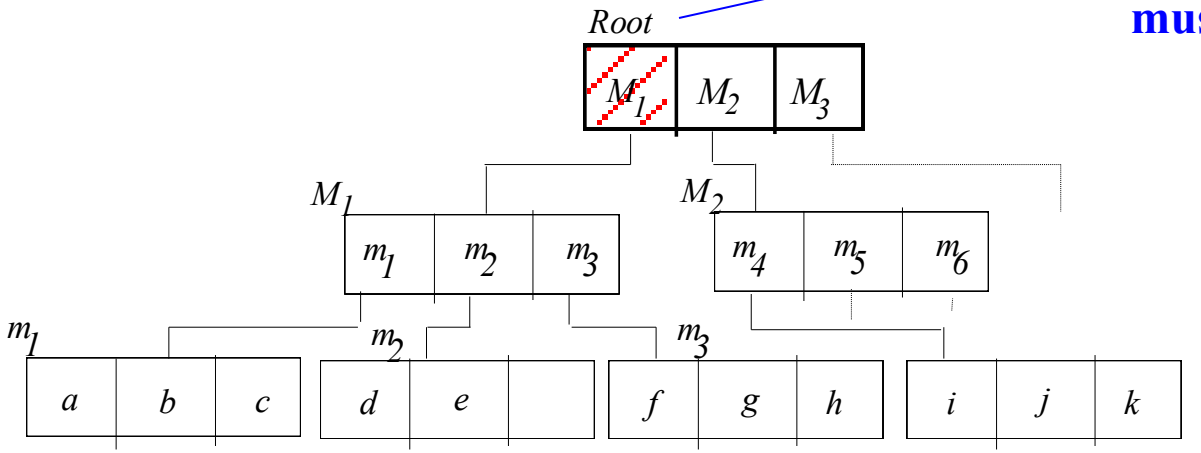
o_{NN} = NULL

$dist(q, o_{NN}) = \infty$

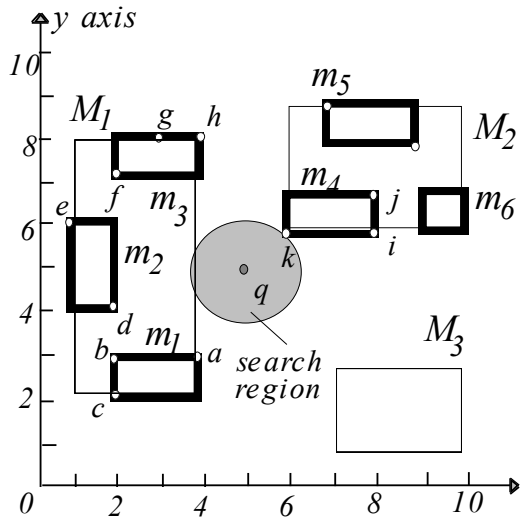
1. visit root

$MINDIST(q,M_1) < dist(q,o_{NN})$

must visit node M_1



Branch-and-Bound NN Example (2)



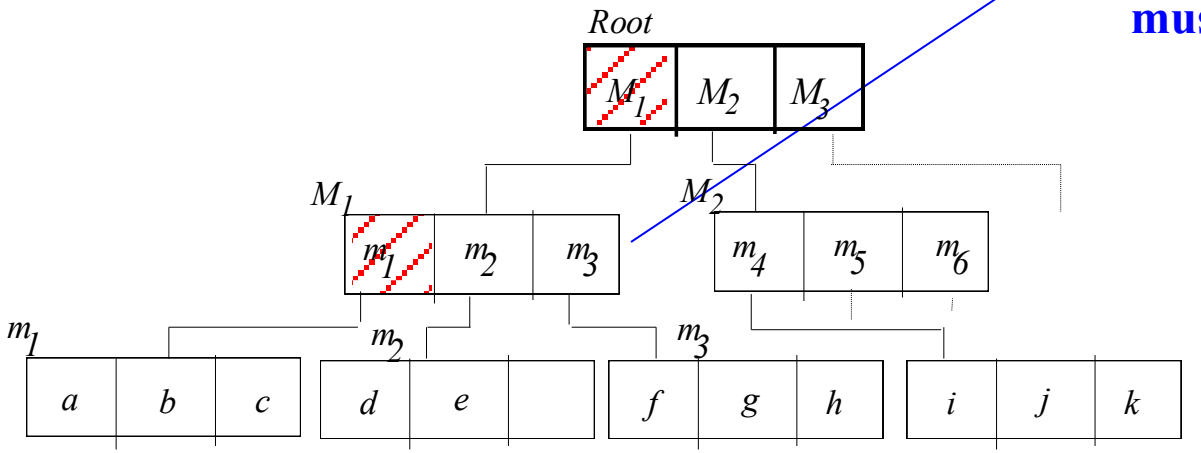
o_{NN} = NULL

$dist(q, o_{NN}) = \infty$

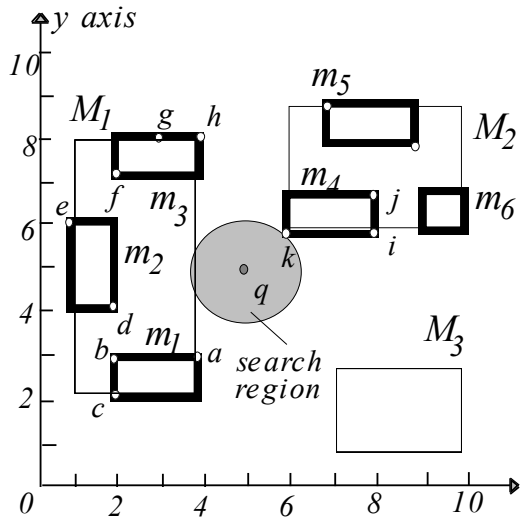
2. visit M_1

$MINDIST(q, m_1) < dist(q, o_{NN})$

must visit node m_1



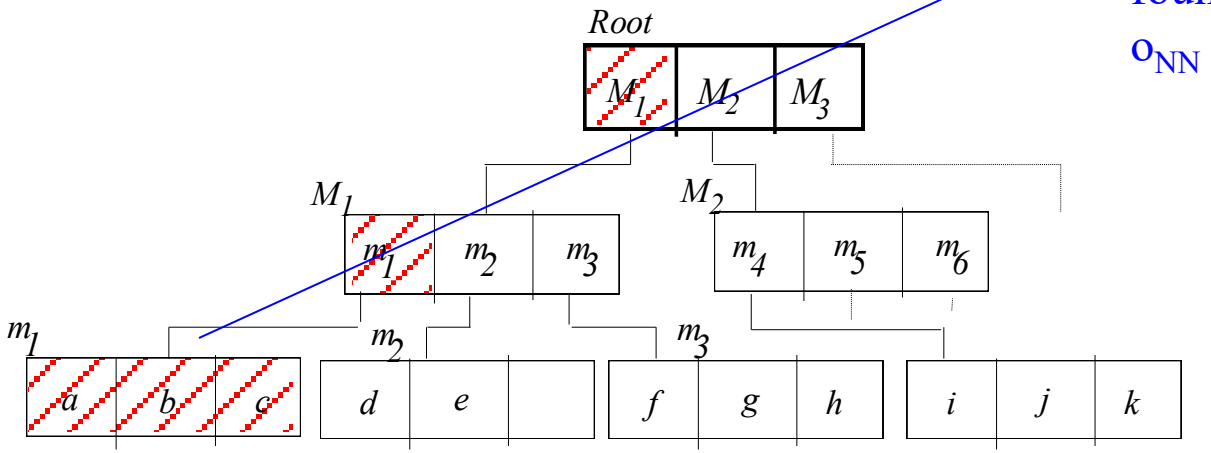
Branch-and-Bound NN Example (3)



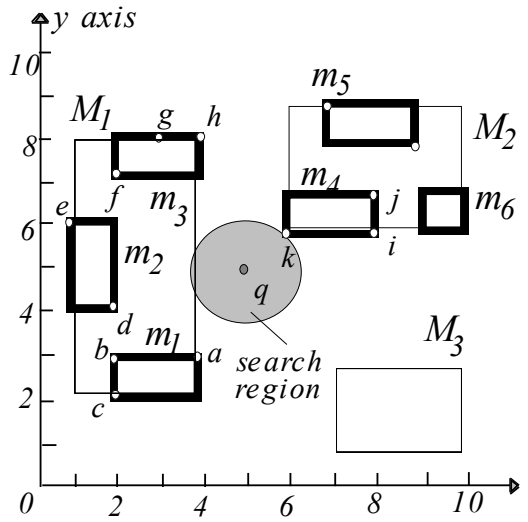
o_{NN} = NULL

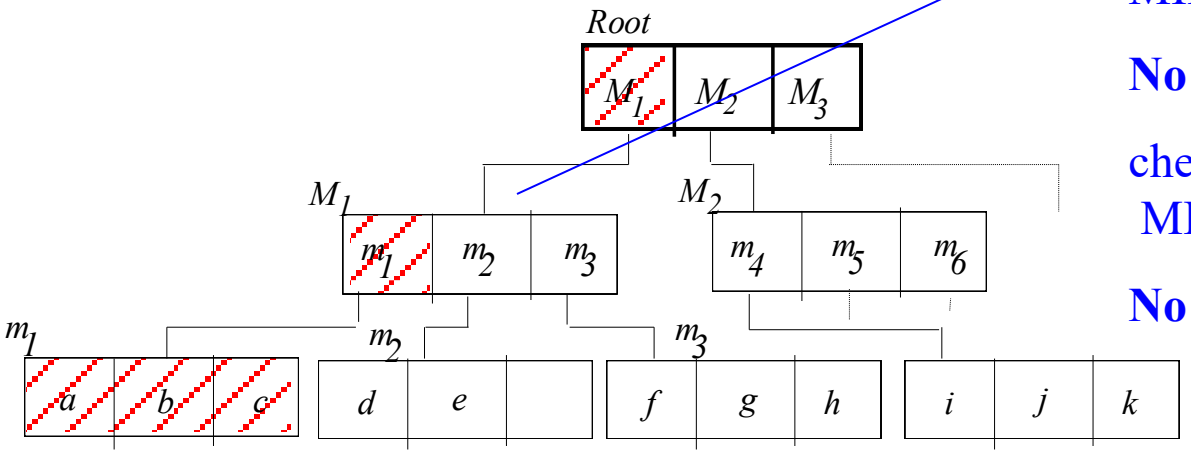
dist(q, o_{NN})= ∞

3. visit m_1
check a,b,c
found new NN:
o_{NN} = a, dist(q, o_{NN}) = √5



Branch-and-Bound NN Example (4)





o_{NN} = a

dist(q, o_{NN})= $\sqrt{5}$

Upward pruning

4. backtrack to M_1

check m_3

MINDIST(q, m_3) = $\sqrt{5} \geq \sqrt{5}$:

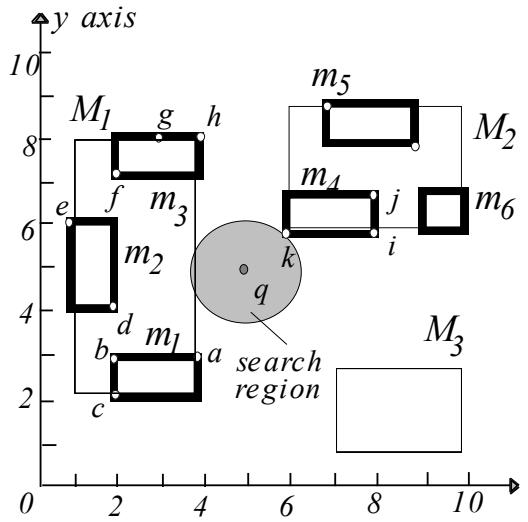
No need to visit node m_3

check m_2

MINDIST(q, m_2) = $3 \geq \sqrt{5}$:

No need to visit node m_2

Branch-and-Bound NN Example (5)



o_{NN} = a

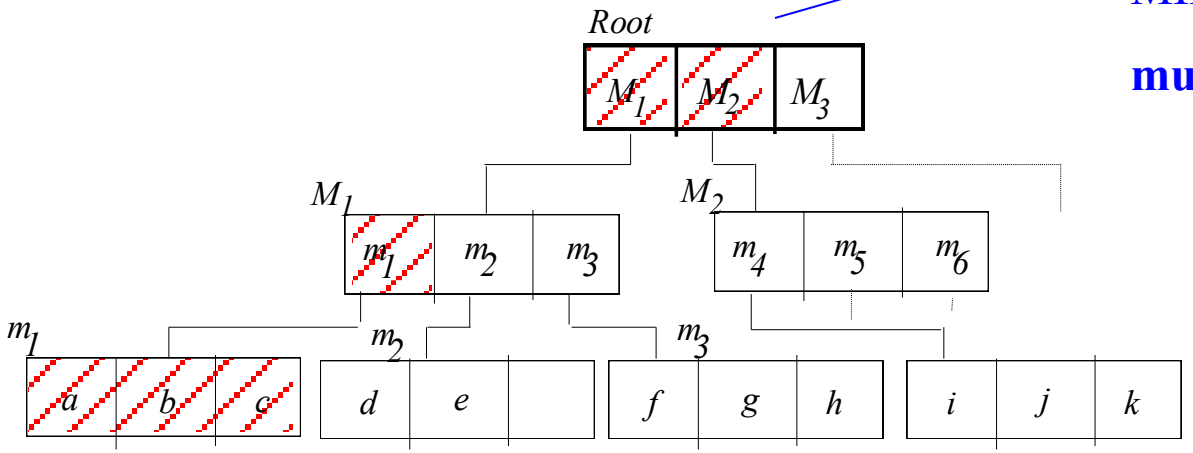
dist(q, o_{NN})= √5

5. backtrack to root

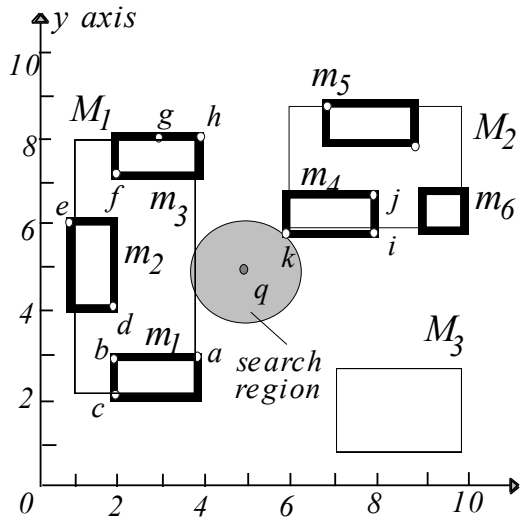
check M_2

MINDIST(q, M_2) = √2 < √5:

must visit node M_2



Branch-and-Bound NN Example (6)



o_{NN} = a

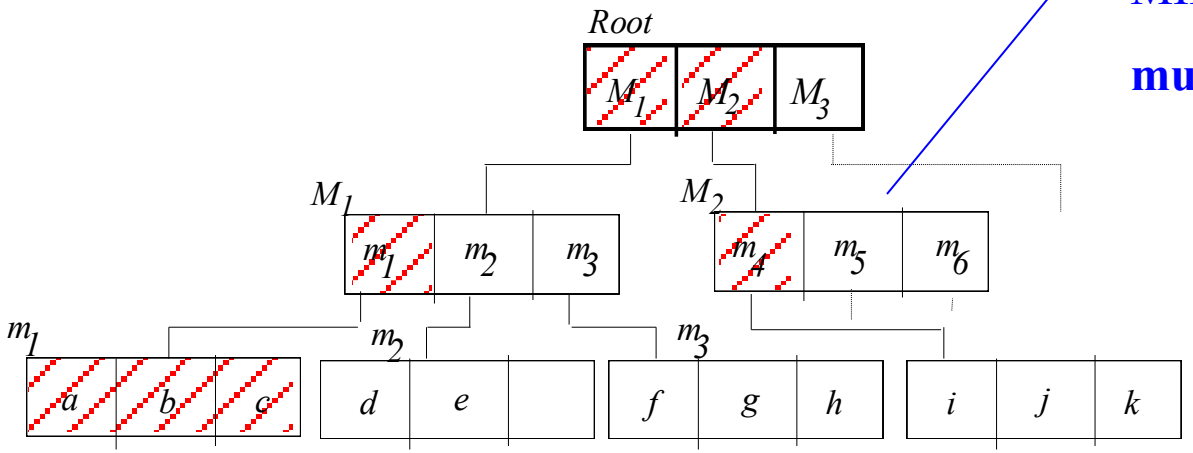
dist(q, o_{NN})= √5

6. visit M_2

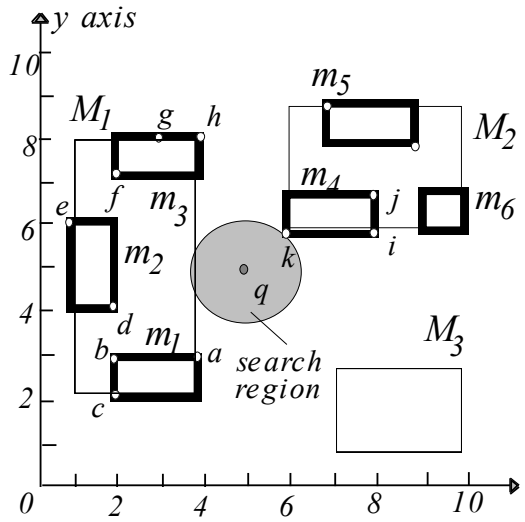
check m_4

MINDIST(q, m_4) = √2 < √5:

must visit node m_4

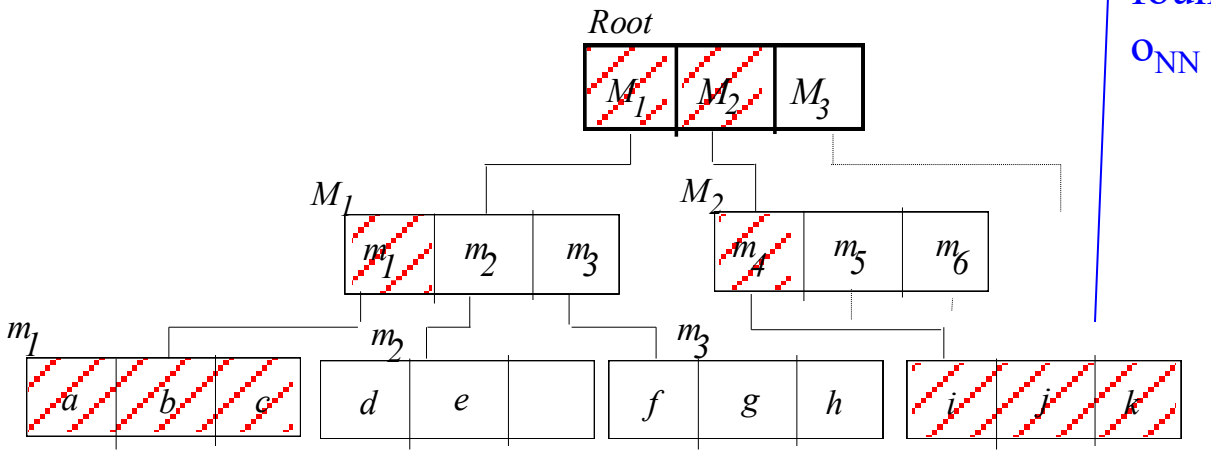


Branch-and-Bound NN Example (7)



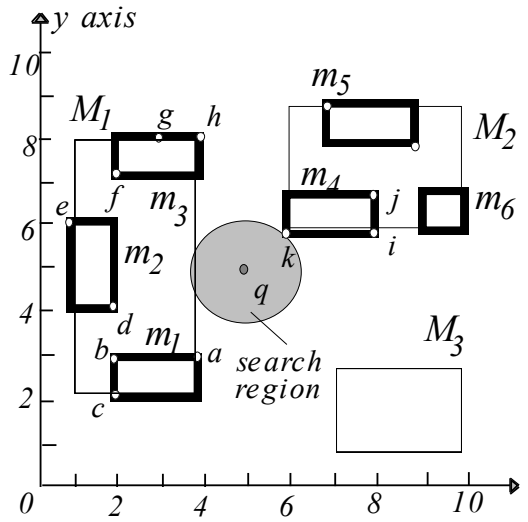
o_{NN} = a

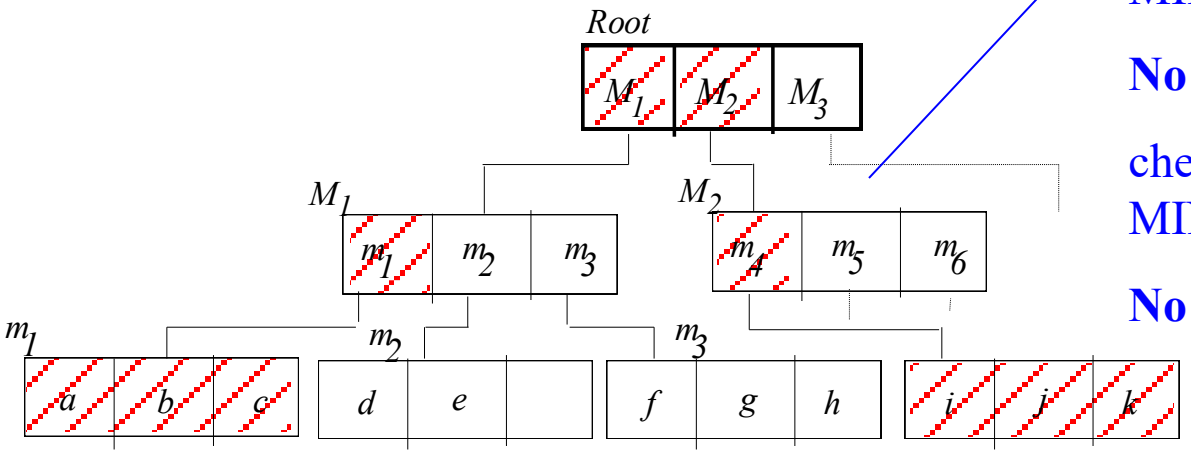
dist(q, o_{NN})= √5



7. visit m_4
check i, j, k
found new NN:
o_{NN} = k, dist(q, o_{NN}) = √2

Branch-and-Bound NN Example (8)





$o_{NN} = k$

$dist(q, o_{NN}) = \sqrt{2}$

Upward pruning

8. backtrack to M_2

check m_5

$MINDIST(q, m_5) \geq \sqrt{2}$:

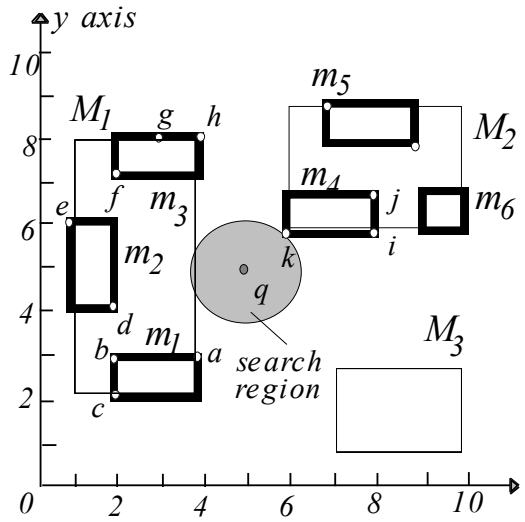
No need to visit node m_5

check m_6

$MINDIST(q, m_6) \geq \sqrt{2}$:

No need to visit node m_6

Branch-and-Bound NN Example (9)



$o_{NN} = k$

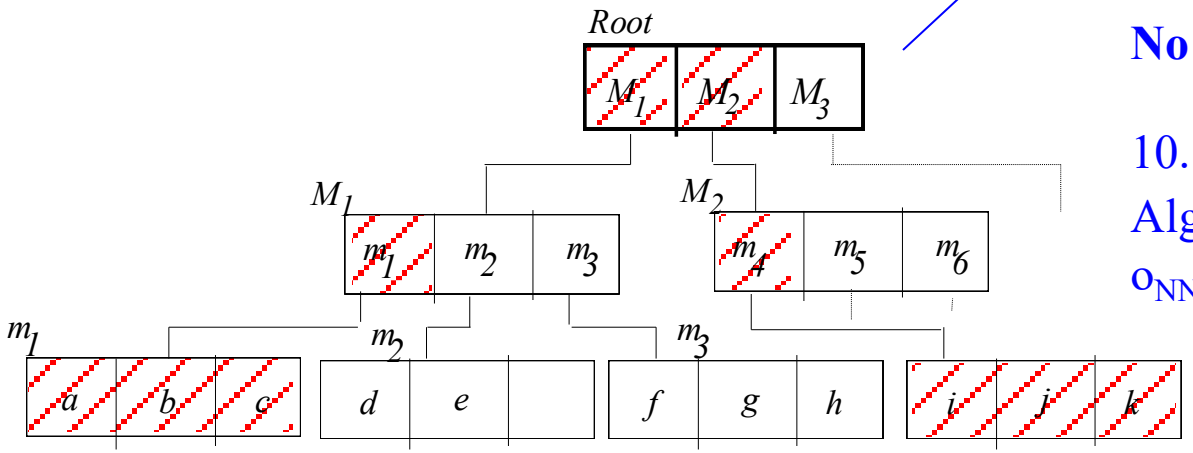
$dist(q,o_{NN}) = \sqrt{2}$

Upward pruning

9. backtrack to root
check M_3
$MINDIST(q, M_3) \geq \sqrt{2}$:

No need to visit node M_3

10. backtrack from root
Algorithm terminates
$o_{NN} = k$, $dist(q,o_{NN}) = \sqrt{2}$

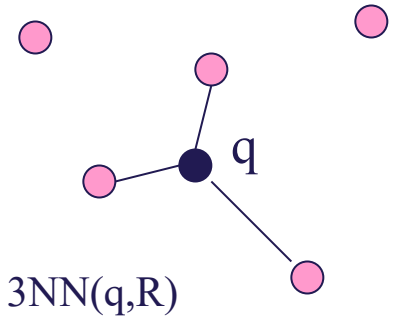


Notes on DF (BaB) NN search

- Large space can be pruned by avoiding visiting R-tree nodes and their sub-trees
- Orders entries of a node by metric to maximize likelihood to find good NN candidate quickly
- Requires at most one tree path to be currently in memory – good for small memory buffers
  - Characteristic of all depth-first search algorithms
  - Recall that range search algorithm is also DF
- However, does not visit least possible number of tree nodes
- Can be adapted for k-NN search (next slide)
- But, not incremental – more later…

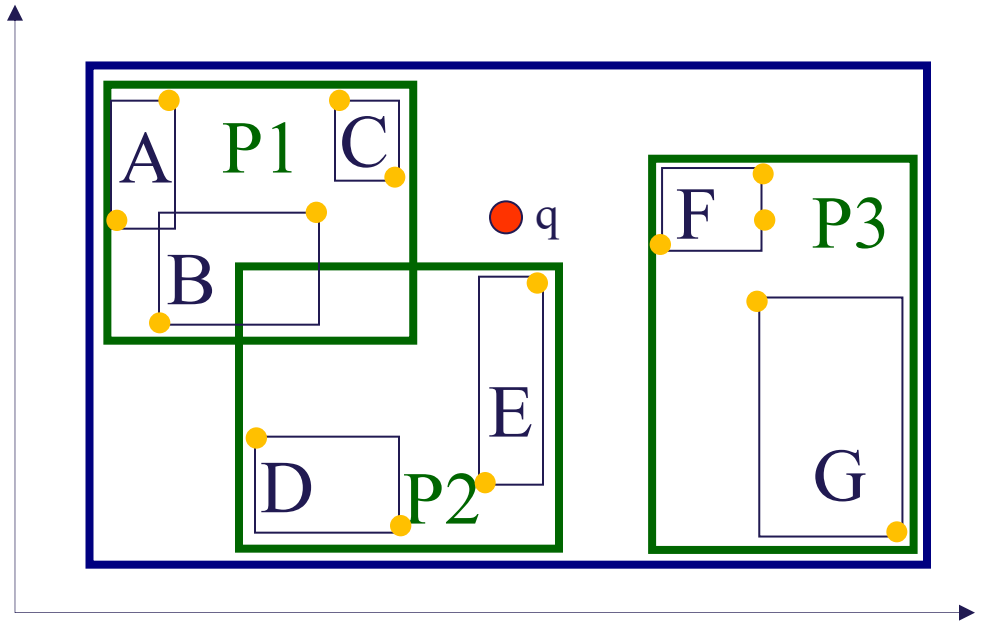
k Nearest Neighbor Search

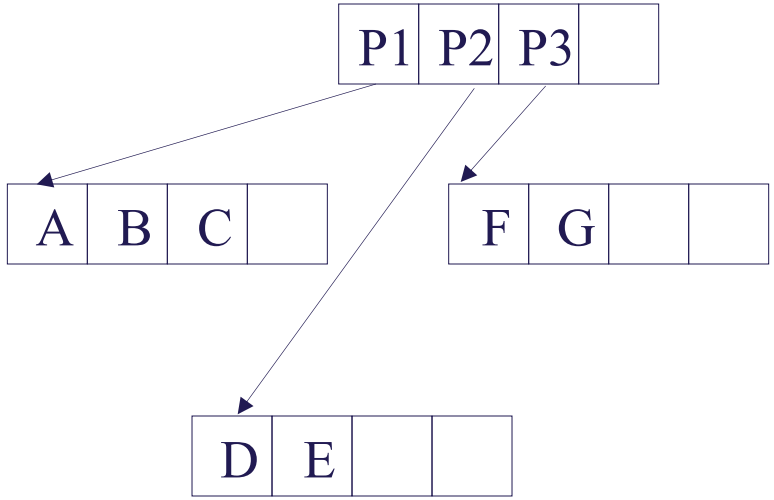
- Generalized problem
  - Given a spatial relation R, a query object q, and a number k <|R|, find the k-nearest neighbors of q in R
    - kNN(q,R) = S $\subseteq$ R : |S| = k, dist(q,o) $\leq$ dist(q,o'), $\forall$ o $\in$ S $\forall$ o' $\in$ R-S
- The branch-and-bound NN search method can be extended for performing kNN search
  - Instead of a single o_{NN}, maintain k nearest objects
  - Pruning is done using the k-th NN (distance) found so far



Exercise

- Apply the branch-and-bound NN search to the given the R-tree and query point q below.
  - Use ABL sorted on MINDIST
  - Don’t use ABL



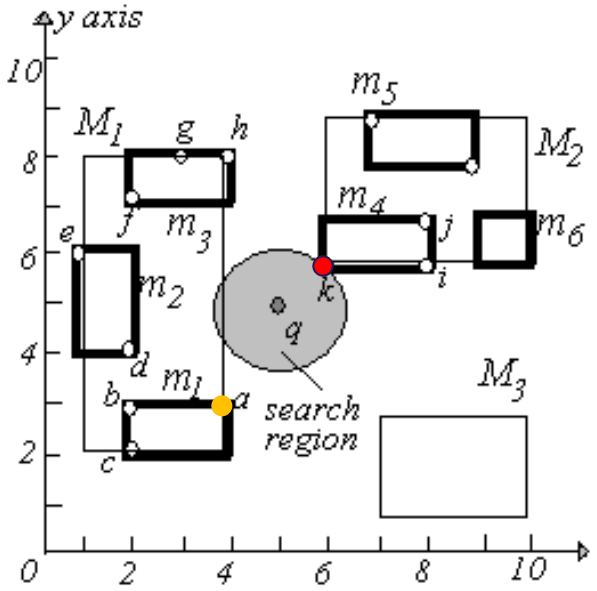


Agenda

- Nearest Neighbor Queries
  - Distance Metrics in R-tree
  - Branch-and-Bound NN Search
  - Best-First NN Search
- Spatial Join
- Implementation of R-tree

Observation about DF NN search

- In the branch-and-bound NN search
  - The closest entry to q in the current node is “visited” and control is passed to the node pointed by it
  - However, the entries in that node may not be the closest entries to q from those seen so far
- In the previous example, point a is visited before the real NN k
- Can we do better?
  - E.g., avoid visiting node m1



Best-First NN Search

- More efficient (given enough memory)
- Idea of BF search:
  - Uses a priority queue to organize seen entries and prioritize the next node to be visited
  - Always “visit” the closest one (from the priority queue)
  - When an object is seen from the priority queue, it is the NN
  - Low I/O cost: optimal in the number of R-tree nodes visited for a given query q
- Adaptable to kNN search and incremental NN search.

Priority Queue Example

- Priority queue Q
  - Organize seen entries and prioritize the next node to be visited
  - Entry format: tree entry, and distance from q
  - E.g, $Q = M_1(1), M_2(\sqrt{2}), M_3(\sqrt{8})$
- Supported operations
  - Top(Q): get the top entry of Q
    - E.g., we get the entry $M_1(1)$
  - Pop(Q): remove the top entry of Q
    - E.g., after Pop(Q), we have $Q = M_2(\sqrt{2}), M_3(\sqrt{8})$
  - Push(Q, e): insert an entry e into Q
    - Suppose MINDIST(m_4, q)= $\sqrt{5}$
    - After Push(Q, m_4), we have $Q = M_2(\sqrt{2}), m_4(\sqrt{5}), M_3(\sqrt{8})$

We use the term priority queue and heap interchangebly

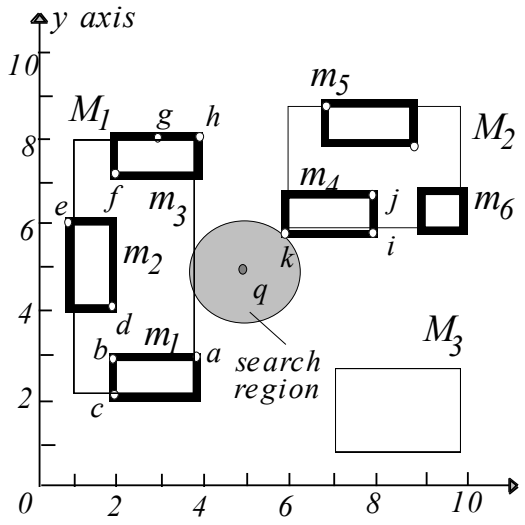
Best-First NN Search Algorithm

1. o_{NN} = NULL; dist(q, o_{NN})= ∞;
2. Initialize a priority queue Q;
3. Add all entries in R-tree's root node to Q;
4. **while** Q is not empty

```
e = Pop(Q);
// all other entries are too far away
if dist(q, e.MBR) >= dist(q, o_NN) then break;
if e is an internal node entry
    n = the node that e points to;
    for each entry e' in n
        if dist(q, e'.MBR) < dist(q, o_NN) then Push(Q, e');
else // e points to a point
    o = the object that e points to;
    if dist(q, o) < dist(q, o_NN) then o_NN = o; // new NN is found
```

5. **return** o_{NN};

Best-First NN Search Example (1)



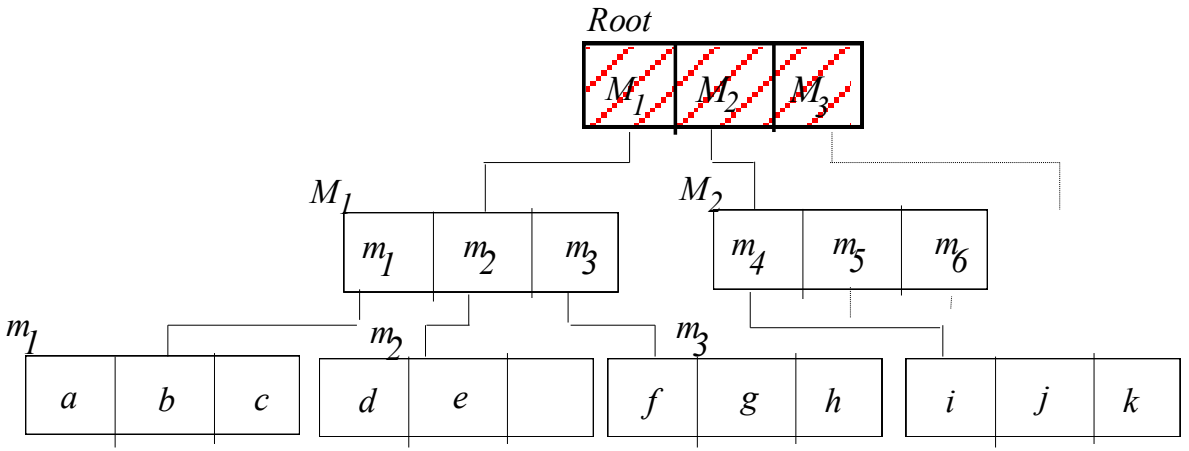
o_{NN} = NULL

dist(q, o_{NN})= ∞

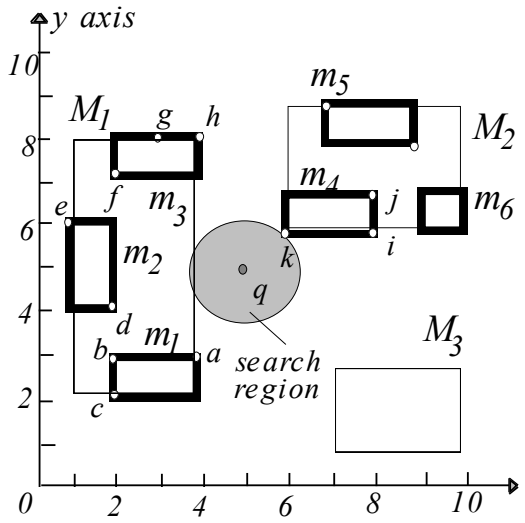
Step 1: put all entries of root on heap Q

$Q = M_1(1), M_2(\sqrt{2}), M_3(\sqrt{8})$

MINDIST from q



Best-First NN Search Example (2)



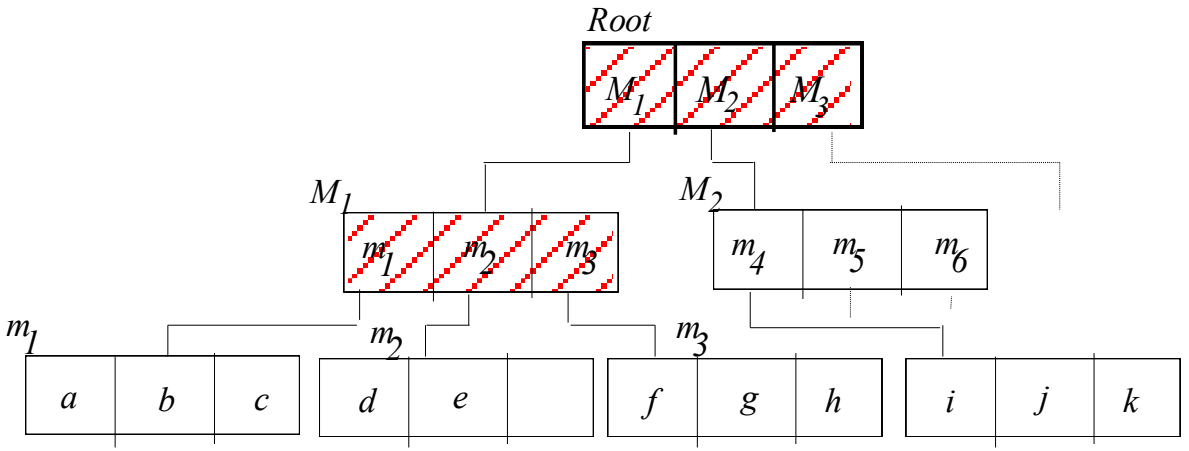
o_{NN} = NULL

$dist(q,o_{NN}) = \infty$

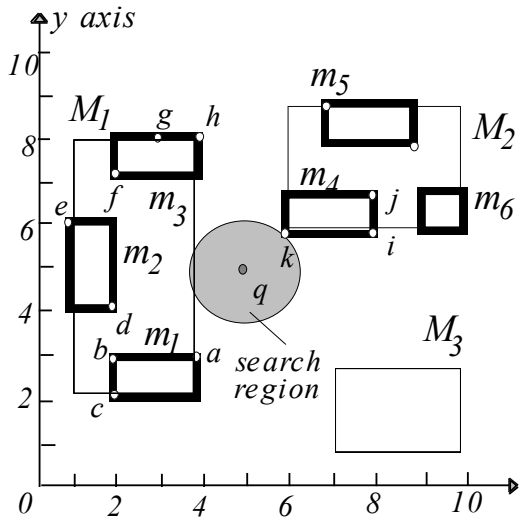
Step 2: get closest entry (top element of Q):

$M_1(1)$. Visit node M_1. Put all M_1's entries on heap Q

$Q = M_2(\sqrt{2}), m_1(\sqrt{5}), m_3(\sqrt{5}), M_3(\sqrt{8}), m_2(3)$



Best-First NN Search Example (3)



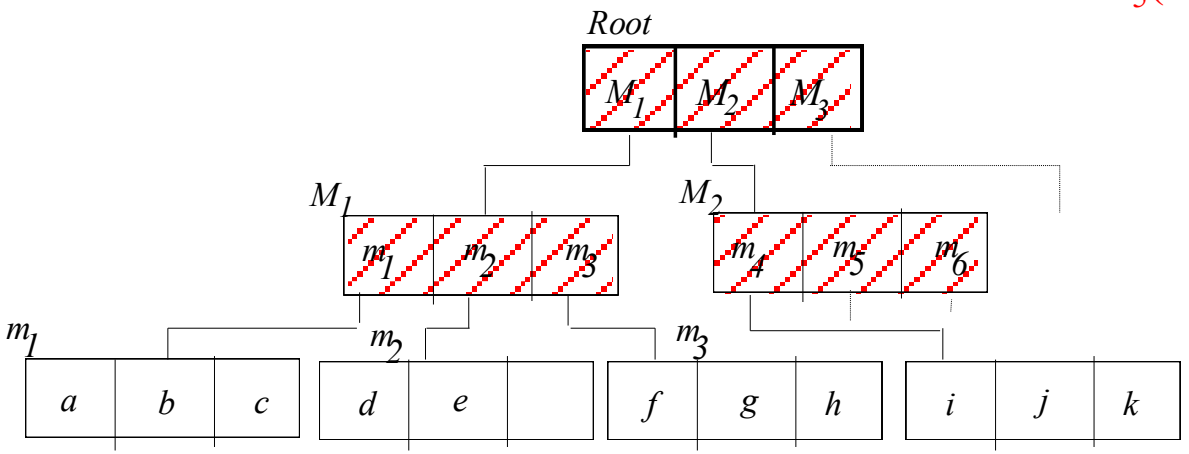
o_{NN} = NULL

dist(q,o_{NN})= ∞

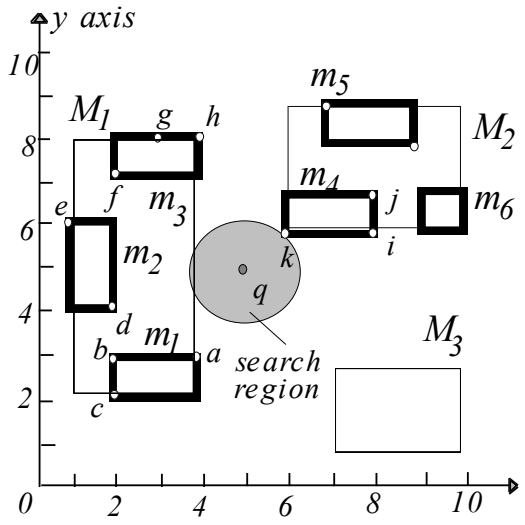
Step 3: get closest entry (top element of Q):

$M_2(\sqrt{2})$. Visit node M_2. Put all M_2's entries on heap Q

$Q = m_4(\sqrt{2}), m_1(\sqrt{5}), m_3(\sqrt{5}), M_3(\sqrt{8}), m_2(3), m_5(\sqrt{13}), m_6(\sqrt{17})$



Best-First NN Search Example (4)



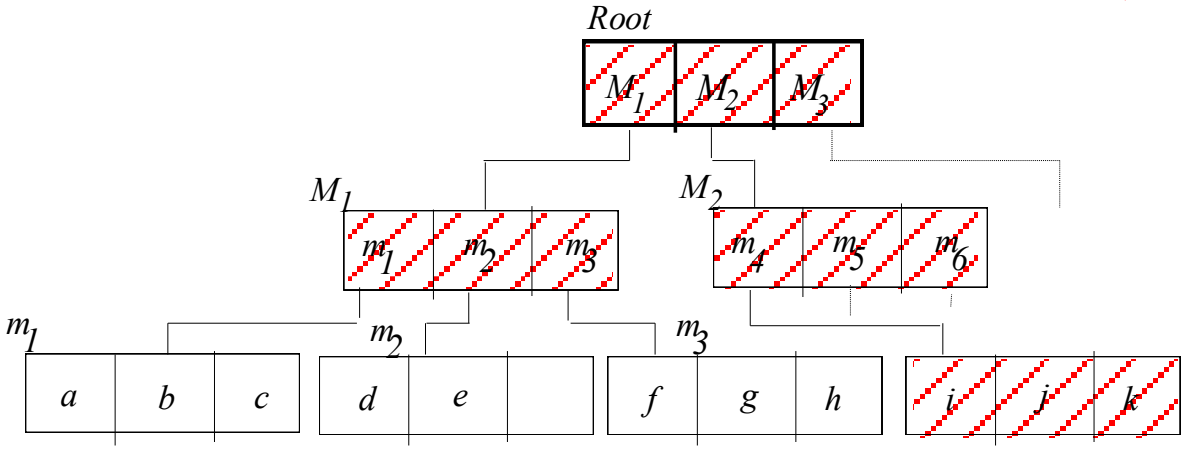
o_{NN} = NULL

dist(q,o_{NN})= ∞

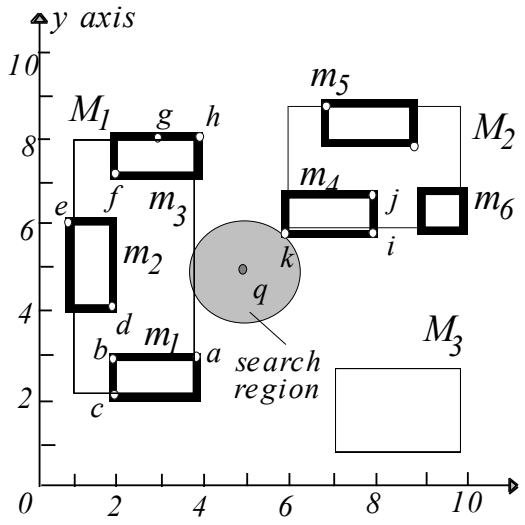
Step 4: get closest entry (top element of Q):

$m_4(\sqrt{2})$. Visit node m_4. m_4 is a leaf node, so put objects i, j, and k on Q:

$Q = k(\sqrt{2}), m_1(\sqrt{5}), m_3(\sqrt{5}), M_3(\sqrt{8}), m_2(3), i(\sqrt{10}), j(\sqrt{13}), m_5(\sqrt{13}), m_6(\sqrt{17})$



Best-First NN Search Example (5)



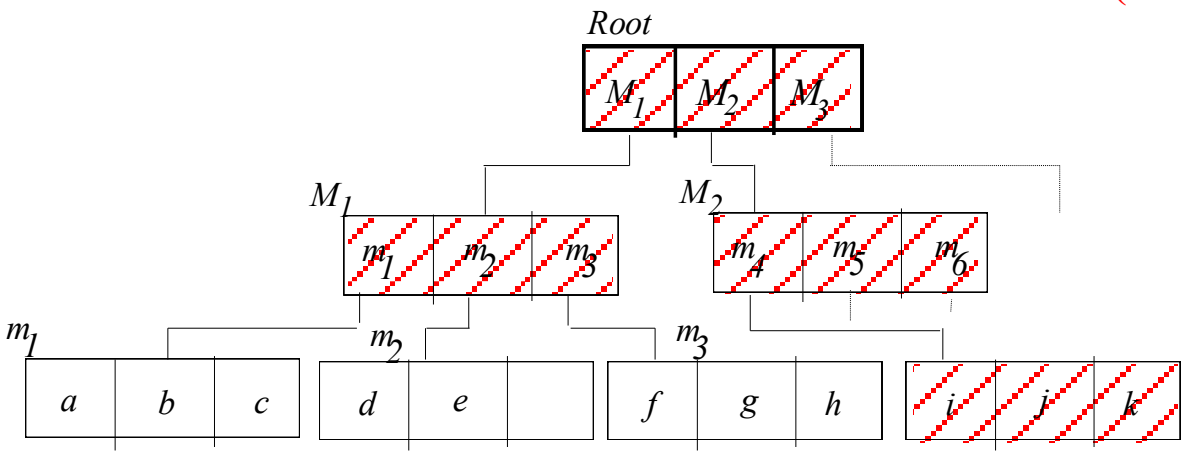
$o_{NN} = k$

$dist(q,o_{NN}) = \sqrt{2}$

Step 5: get closest entry (top element of Q):

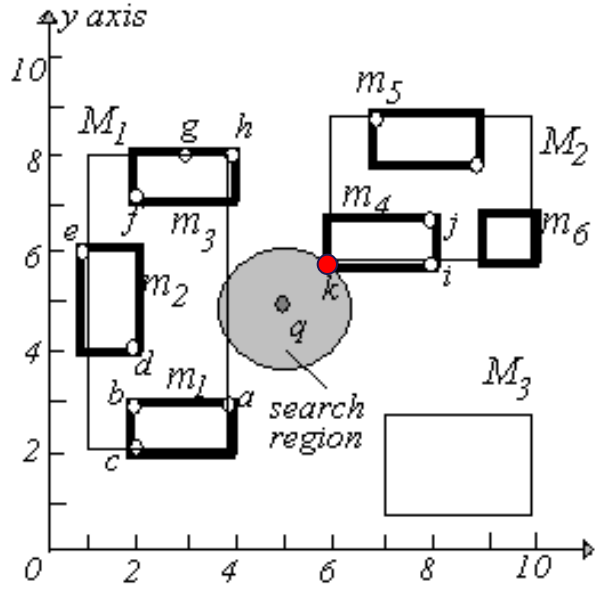
$k(\sqrt{2})$. Since k is an object, search stops and returns k as the NN of q

$Q = m_1(\sqrt{5}), m_3(\sqrt{5}), M_3(\sqrt{8}), m_2(3),$
$i(\sqrt{10}), j(\sqrt{13}), m_5(\sqrt{13}), m_6(\sqrt{17})$



Notes on Best-First NN Search

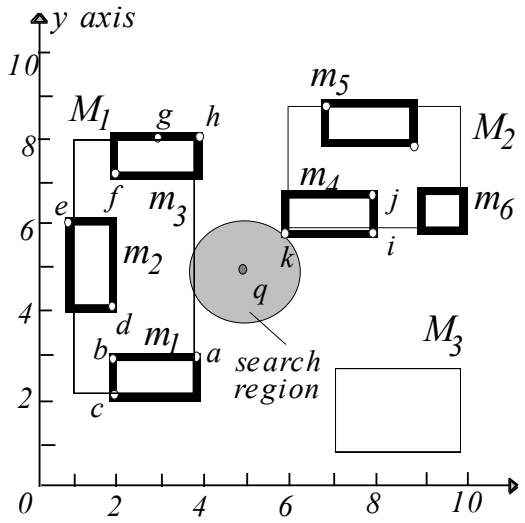
- In the previous example, fewer nodes are visited when compared to DF-NN search
  - Only visit nodes whose MBR intersects the disk centered at q with radius the real NN distance
  - I.e., optimal number of node accesses
- Adaptable for kNN search
  - Maintain the best k objects found so far (like in DF-NN), or
  - Simply use the incremental search (in the next slide)
- However, the worst-case space requirement of the algo
  - The heap may grow large until the algorithm terminates



Incremental NN search

- The user may not know a “suitable” k in advance
  - After having found the NN, find the next NN without starting search from the beginning
- Application 1: find the nearest large city (>10,000 residents) to my current position
  - incrementally find NN and check if the large city requirement is satisfied; if not get the next NN
- Application 2: find the nearest hotel; see if you like it; if not get the next one; see if you like it; …
- Best-first NN search can be modified for incremental NN search
  - Once an object is popped from the priority queue, report it as the current NN.
  - Do not prune but continue processing (pop and push operations on the queue) until k NNs have been reported or the user terminates the processing.

Incremental Search by Best-First NN

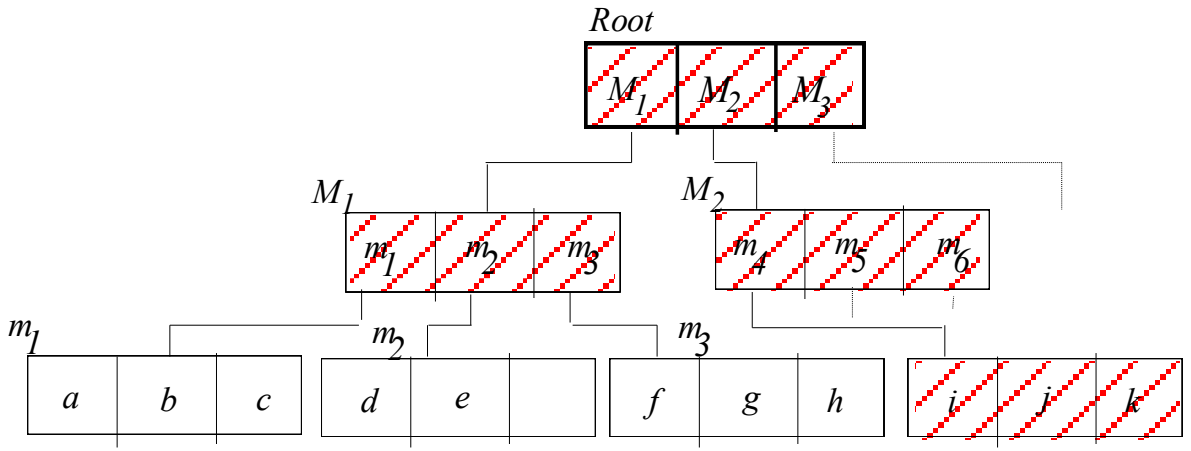


1st NN = k

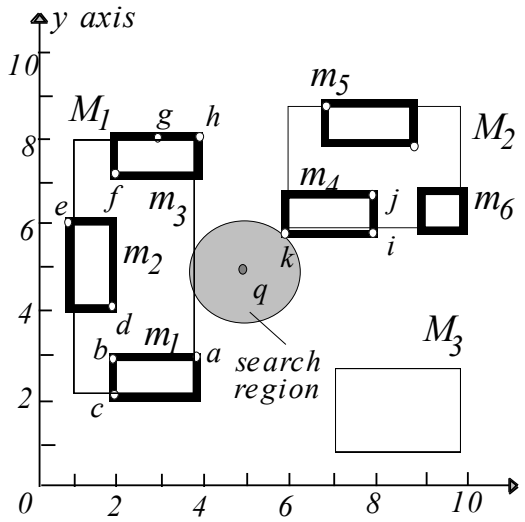
Step 5: get closest entry (top element of Q):

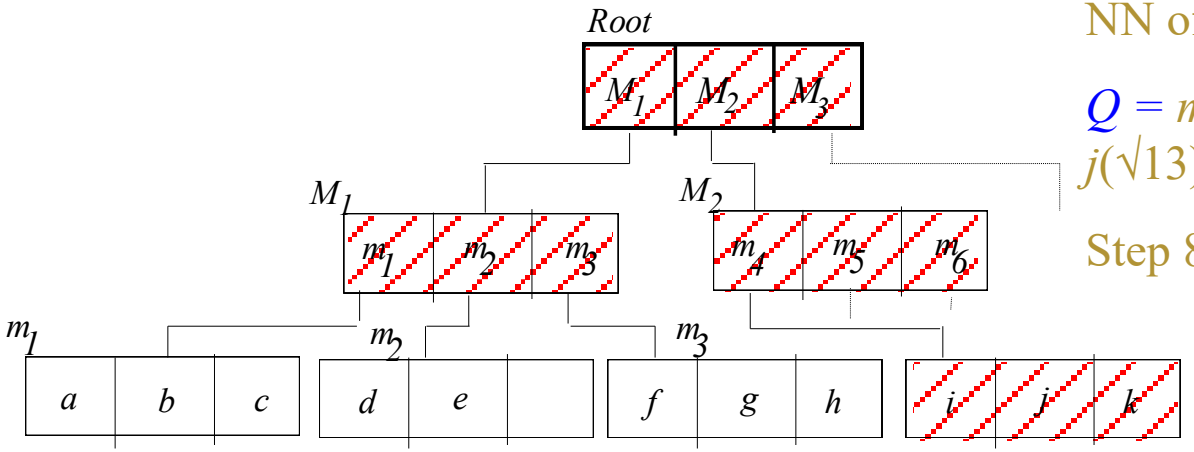
$k(\sqrt{2})$. Since k is an object, output k as the 1st NN of q.

$Q = m_1(\sqrt{5}), m_3(\sqrt{5}), M_3(\sqrt{8}), m_2(3), i(\sqrt{10}), j(\sqrt{13}), m_5(\sqrt{13}), m_6(\sqrt{17})$



Incremental Search by Best-First NN





1st NN = k

$Q = m_1(\sqrt{5}), m_3(\sqrt{5}), M_3(\sqrt{8}), m_2(3), i(\sqrt{10}), j(\sqrt{13}), m_5(\sqrt{13}), m_6(\sqrt{17})$

Step 6: continue to m_1 from Q, expand m_1, and update Q to

$Q = a(\sqrt{5}), m_3(\sqrt{5}), M_3(\sqrt{8}), m_2(3), i(\sqrt{10}), b(\sqrt{13}), j(\sqrt{13}), m_5(\sqrt{13}), m_6(\sqrt{17}), c(\sqrt{18})$

Step 7: get a from Q and output a as the 2nd NN of q

$Q = m_3(\sqrt{5}), M_3(\sqrt{8}), m_2(3), i(\sqrt{10}), b(\sqrt{13}), j(\sqrt{13}), m_5(\sqrt{13}), m_6(\sqrt{17}), c(\sqrt{18})$

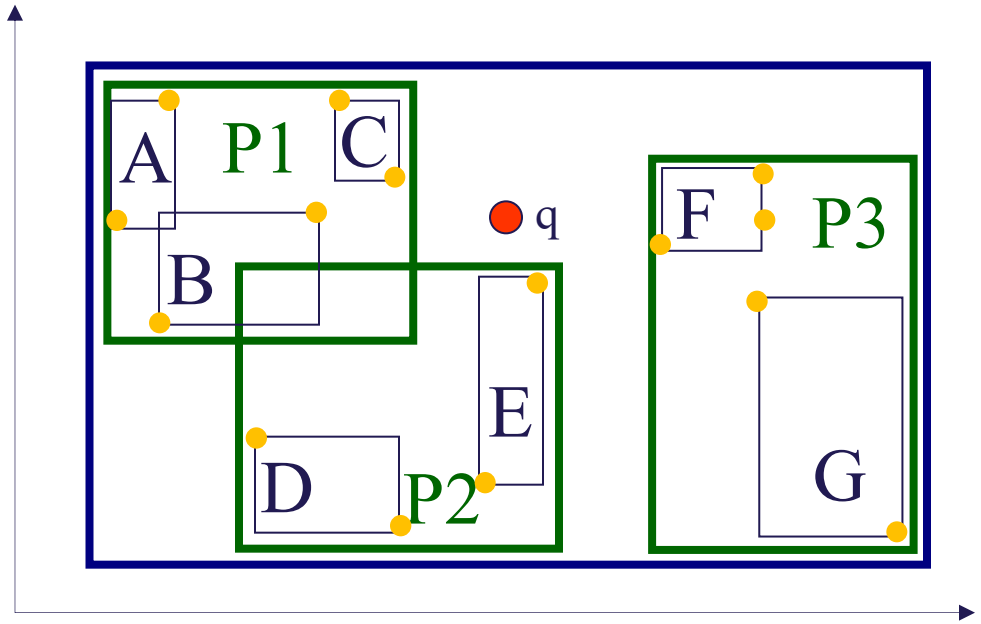
Step 8: …

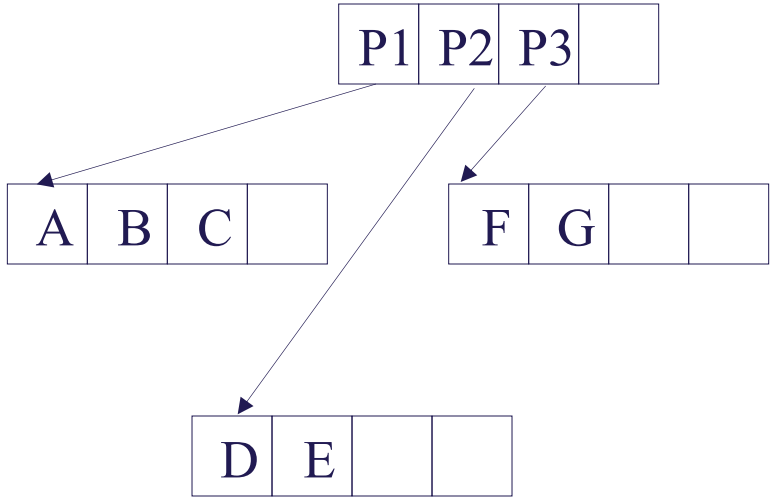
Depth-First Search vs. Best-First Search

	Depth-first search	*Best-first search*
Number of node accesses	Not guaranteed to be the minimum	Optimal number of node accesses
Memory requirement	At most a single path of tree nodes	Unbounded memory
Incremental search	Not applicable	Applicable

Exercise

- Apply the best-first NN search to the given the R-tree and query point q below.
  - Report the first NN
  - Report the next 2 NNs





Reference for NN

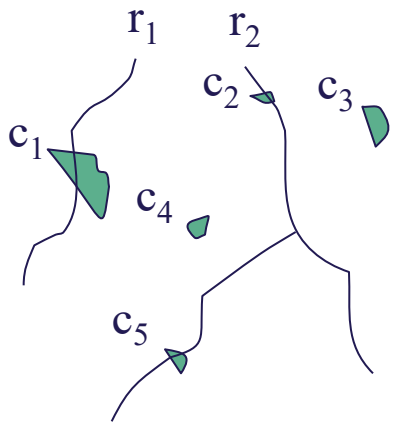
- Depth-first NN search
  - Nick Roussopoulos, Stephen Kelley, Frédéic Vincent: Nearest Neighbor Queries. SIGMOD Conference 1995: 71-79
- Best-first NN search
  - Gísli R. Hjaltason, Hanan Samet: Distance Browsing in Spatial Databases. TODS 24(2):265-318, 1999.

Agenda

- Nearest Neighbor Queries
- Spatial Join
  - Definition
  - Algorithm
- Implementation of R-tree

Spatial Join

- Given two spatial data relations *P* and *Q*, and a join condition, a spatial join returns object pairs (*p*, *q*) that
  - *p* from *P* and *q* from *Q*
  - *p* and *q* satisfy the given join condition
- Join conditions can be any of
  - Topological/Distance/Directional relationships
- E.g., find all (river, city) pairs that intersect
  - *Result set*: {(r_1, c_1), (r_2, c_2), (r_2, c_5)}
- E.g., find all (river, city) pairs with distance < 10kms
- E.g., find all (river, city) pairs where city is on the river's south bank

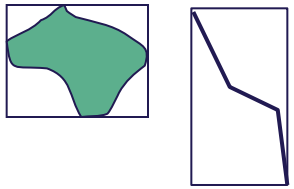


Two-step Spatial Query Processing

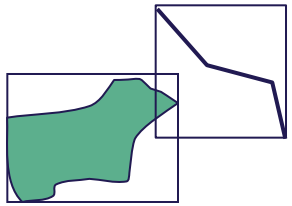
- Evaluating spatial relationships on geometric data is slow
- A spatial object is approximated by its MBR

- A spatial query is usually processed in two steps:
  1. **Filter step**: The MBR is tested against the query predicate. → Fast!
  Often, a particular spatial index is used in this step.
  2. **Refinement step**: The exact geometry of objects that pass the filter step is tested for qualification → Slow!

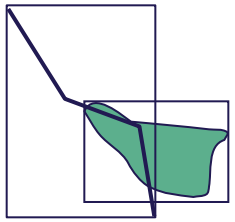
- Example: spatial intersection joins between forests and rivers



filtered pair



non-qualifying pair that passes the filter step (false hit or false positive)



qualifying pair

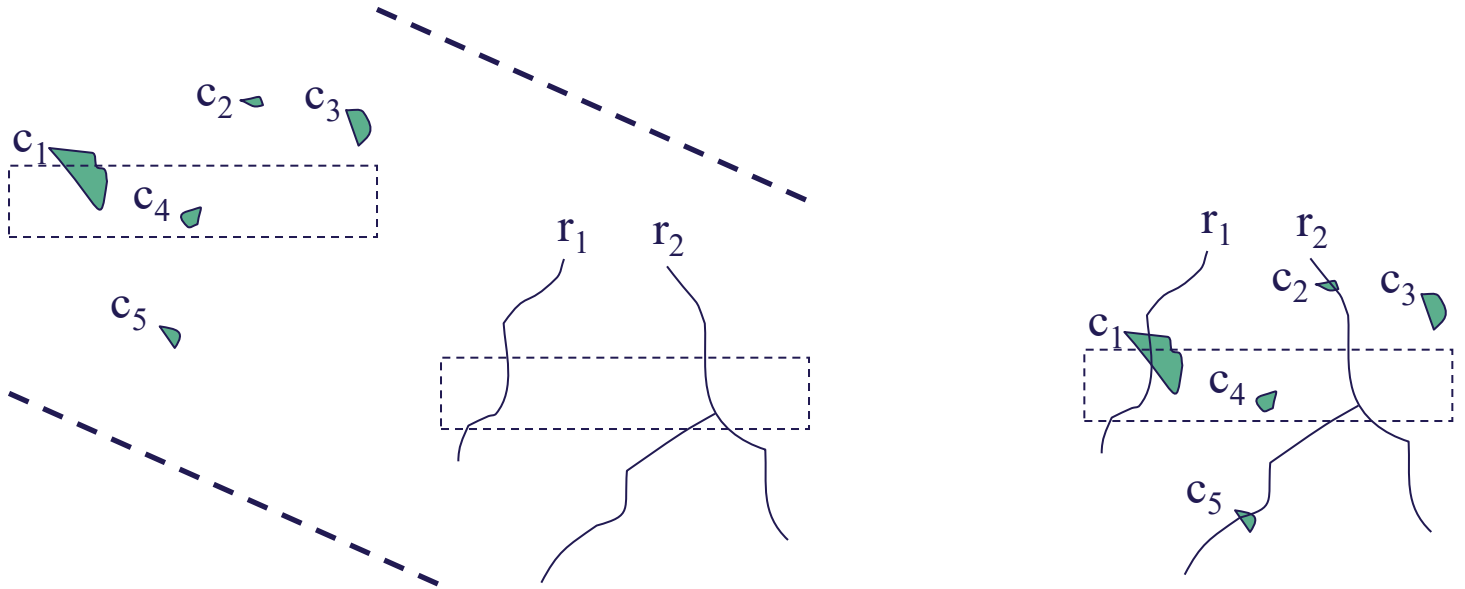
Spatial Join Definition

- Input
  - Two spatial relations R, S (e.g., R=cities, S=rivers)
- Output:
  - {(r, s): r$\in$R, s$\in$S, r intersects s}
  - Example: find all pairs of cities and rivers that intersect

- We focus on *intersection* join
- They are other types of spatial join
  - E.g., the distance between r and s is smaller than a threshold

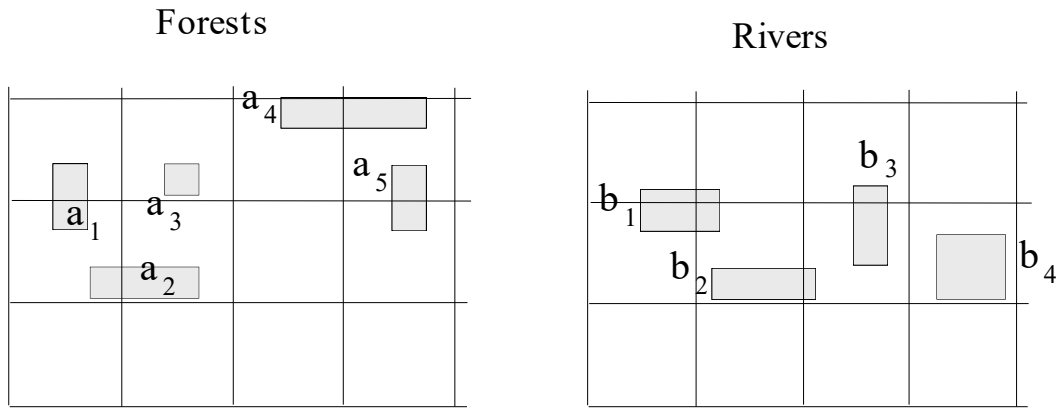
Spatial Join Example

- Given R=cities and S=rivers, find all pairs of cities and rivers that intersect
  - Result: {(r1, c1), (r2, c2), (r2, c5)}



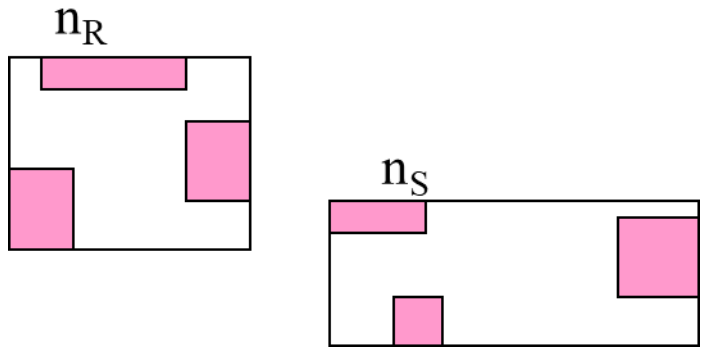
Spatial Join Processing

- Problem: Find all forest and river pairs that intersect with each other.
- Straightforward solution:
  - Nested loop join between the relations R and S
  - The cost is O(|R|*|S|); it is not scalable
- Goal: suppose that both R and S are indexed by R-trees
  - How to process the join efficiently?
  - How do we apply the filter/refinement process?



R-tree (Intersection) Join

- Applies on two R-trees of spatial relations R and S
- Observation:
  - If a node $n_R \in R$ does not intersect $n_S \in S$, then no object $o_R \in R$ under n_R can intersect any object $o_S \in S$ under n_S
  - Node MBRs at the high level of the trees can prune object combinations to be checked
- Utilize this property to traverse synchronously both trees, following only entry pairs that intersect



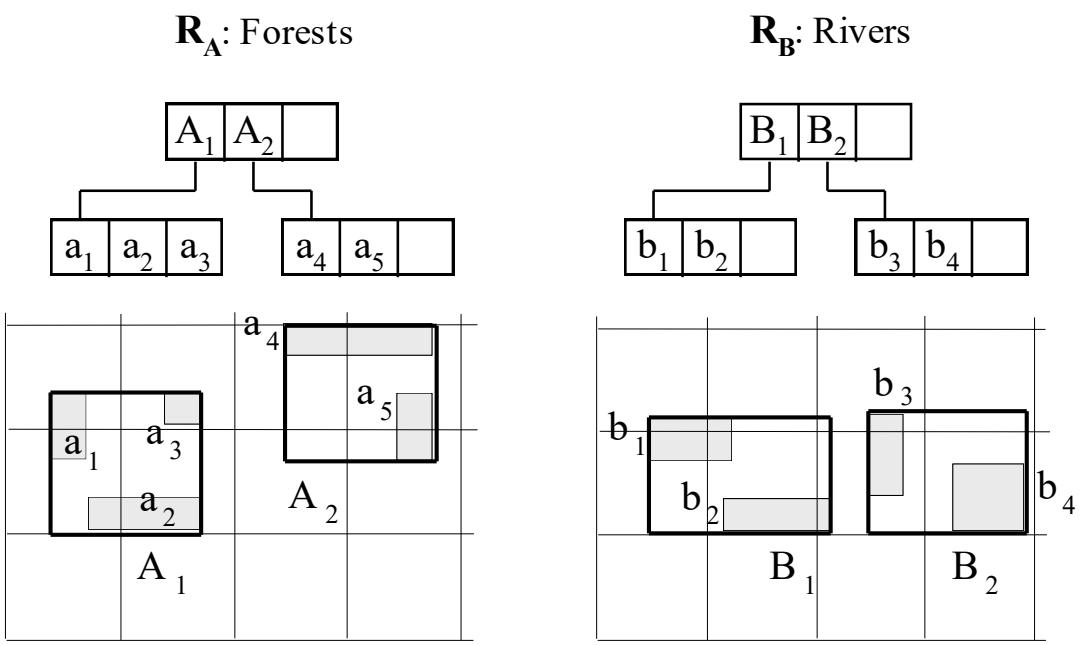
R-tree (Intersection) Join Algorithm

- Initially, RJ is called with two tree roots as parameters
- This pseudo-code version RJ assumes that the trees have same height
  - It can be easily extendable to trees of different heights
  - Just push forward the other for-each loop if leaves are met

```
function RJ(Node n_R, Node n_s)
    for each entry e_i in n_R
        for each entry e_j in n_s
        if e_i.MBR ∩ e_j.MBR = Ø then break;
        if n_R is a leaf node then // so is n_s
                add(e_i's object, e_j's object) to the result
            else // both nodes are internal nodes
                // run the function recursively on two child nodes
                RJ(e_i's child node, e_j's child node)
```

R-tree (Intersection) Join Example

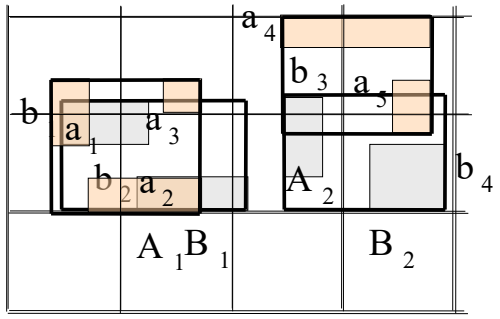
- Run for root(R_A), root(R_B)
- For every intersecting pair there (e.g., A_1, B_1), run recursively for pointed children nodes
- Intersecting pairs of leaf nodes may yield qualifying object pairs



Level 1 qualifying pairs:
$\{(A_1, B_1), (A_2, B_2)\}$

Level 0 qualifying pairs from (A_1, B_1):
$\{(a_1, b_1), (a_2, b_2)\}$

No qualifying pairs from (A_2, B_2):



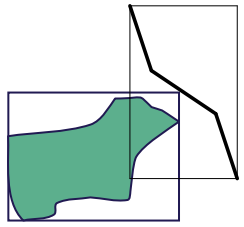
Filter-Refinement Steps for Spatial Join

- A multi-step process is adopted for leaf pairs
  - **MBR filter**: find MBR pairs that intersect
    - Done by RJ algorithm
  - **Geometric filter**: compare some more detailed approximations to make conclusions (next slide)
  - **Refinement**: if the join predicate is still inconclusive, perform expensive refinement step
    - Done by computational geometry algorithms

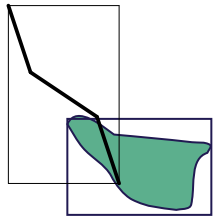
Very fast!

Fast!

Slow!



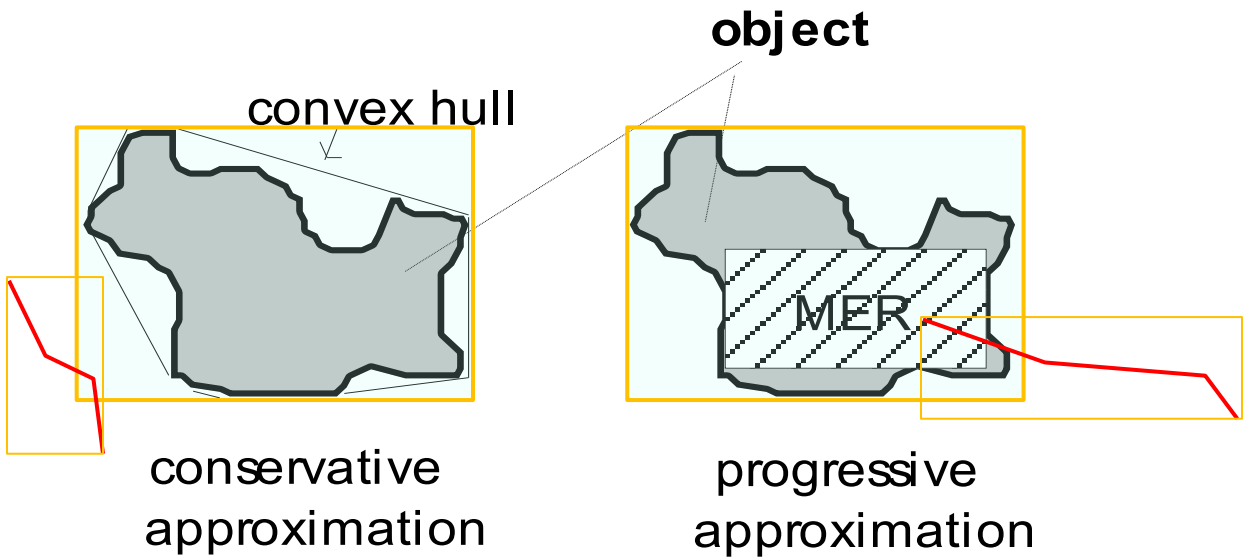
non-qualifying pair that passes the filter step (false hit)



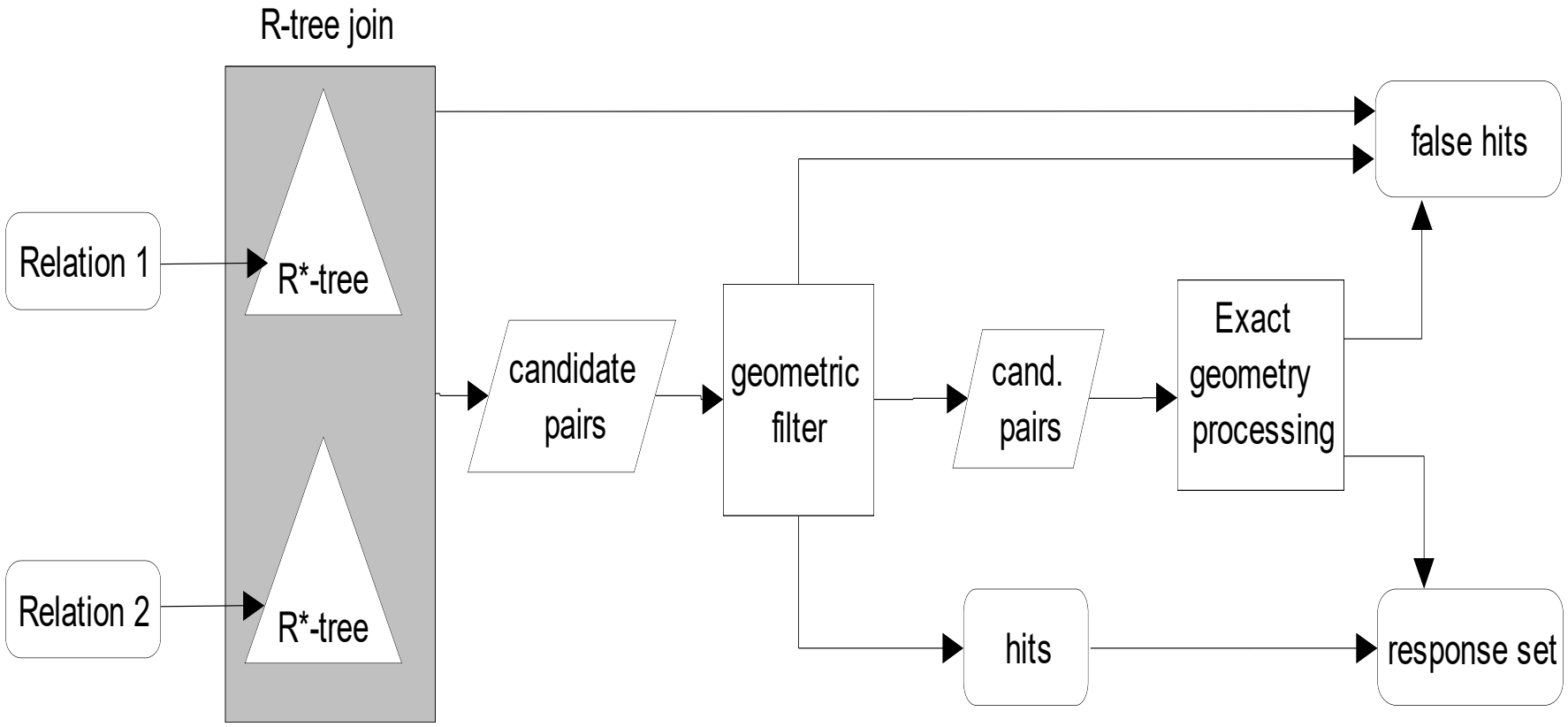
qualifying pair

Geometric Filtering

- Detail approximation
  - Conservative approximation
    - E.g., convex hull
    - False hit detection
  - Progressive approximation
    - E.g., maximum enclosed rectangle (MER)
    - True hit detection



Multi-Step Spatial Join Processing



Exercise

- How to revise the function RJ so that we can spatially join two R-trees with different heights?
  - Does it make differences if we swap the two for-each loops?
- How to revise the function RJ so that we can find all pairs of objects (r, s) such that the distance between r and s is not larger than a given threshold δ?
  - What other join predicates can your algorithm support?

```
function RJ(Node n_R, Node n_S)
    for each e_i ∈ n_R
        for each e_j ∈ n_S, such that e_i.MBR ∩ e_j.MBR≠ ∅
            if n_R is a leaf node then /* n_S is also a leaf node */
                output (e_i.ptr,e_j.ptr); /* a pair of object-ids passing the filter step */
            else /* n_R, n_S are directory nodes */
                RJ(e_i.ptr,e_j.ptr); /* run recursively for the nodes pointed by intersecting entries*/
```

Reference for Spatial Join

- Thomas Brinkhoff, Hans-Peter Kriegel, Bernhard Seeger: Efficient Processing of Spatial Joins Using R-Trees. SIGMOD Conference 1993: 237-246

- More details about
  - CPU time tuning
  - IO cost tuning
  - Spatial join of R-trees with different heights

Agenda

- Nearest Neighbor Queries
- Spatial Join
- Implementation of R-tree
  - Data Structure Details

Implementation of SAMs

- When implement a particular SAM, or an spatial index, we need to consider a number of important issues.
  - Structure of each index entry
  - Structure of the entire index
    - How to organize index entries into an index unit, e.g., a tree node
  - Organization of the index in a disk file
    - E.g., a tree node corresponds to a disk page
  - Index persistent
    - An index should also be stored in disk as a file, otherwise you have to reconstruct it every time you work on the relation underlying it.
    - When making an index persistent, it is necessary to maintain all the structures and the association of structures.

R-tree Implementation

- Dimensionality *d*
- Index entry structure:
  - <MBR, ptr> and <MBR, obj_id>
  - Each MBR is represented by two *d*-dimensional points
    - Suppose each point dimensional value takes *v* bytes. One point takes *d*v* bytes.
    - One MBR thus takes *2*d*v* bytes.
  - ptr and obj_id can be homogeneous, with ptr pointing to the first slot address of a page and obj_id pointing to any slot in a page.
    - Suppose the size of an address is *AS* in bytes.
  - Entry size *ES* = *2*d*v + AS*
- To decide node capacity *M*
  - Node size is the same as disk page size (*PS*)
  - *M* = *PS* / *ES*
    - One page may have a few unused remainder bytes

Notes

- We have not considered storing a pointer to parent in the previous calculation. Usually that is unnecessary.
  - When (a part of) a tree is loaded into memory, a child node's parent node can be maintained through a hash table or an extra, in-memory field for the child node.
  - But when an tree is stored (made persistent) on the disk, the structure is well preserved by only having the parent-to-child pointers.
- Sometimes, we have to use simulation instead of handling real disk pages through low-level OS-dependent APIs.
  - In this case, ptr and obj_id can be *offsets* in the index file and the data file, respectively.

Example

- Suppose a dataset of 2-d points, each dimensional value is of double type.
  - Size of a point is 2*8=16 bytes
  - Size of an MBR is 2*16=32 bytes
- Suppose an object id or offset is of integer type.
  - *AS* = 4 bytes
- Suppose a disk page is of 4K
  - *PS* = 4*1024 = 4096 bytes
- *M* = *PS* / *ES* = 4096 / (32+4) = 4096 / 36 = 113 (floor)

R-tree Height

- Height of node
  - #edges on the longest path between that node and a leaf
- Height of tree
  - The height of its root node
- Suppose an R-tree's height is *h*. It has h+1 levels.
- Suppose there're *N* data records in the relation to index.
- If each tree node is full, we have $M^h = N$
  - At the leaf node level, we need to hold *N* entries, each for one particular data record.
- Tree height $h = \log_M N$. How to have a small *h*?
  - Increase *M*. But how?
    - Use compact index entries
    - Use larger disk page (depending on OS)

Example

- Suppose a dataset of **1M** 2-d points, each dimensional value is of double type.
  - Size of a point is 2*8=16 bytes
  - Size of an MBR is 2*16=32 bytes
- Suppose an object id or offset is of integer type.
  - *AS* = 4 bytes
- Suppose a disk page is of 4K
  - *PS* = 4*1024 = 4096 bytes
- $M = PS / ES = 4096 / (32+4) = 4096 / 36 = 113$ (floor)
- N=1024*1024=1,048,576
- $h = \log_M N = \log_{113} 1{,}048{,}576 = 3$

Implementation of Other SAMs

- Space partitioning trees (k-d tree and quadtree)
  - Node structure
  - Node size
  - Tree height
  - Tree node number
  - Tree persistent
- Grid configuration
  - The general ideas is the same

- The smallest unit that is *not* further split, e.g., a leaf node or a grid cell, points to a data bucket that correponds to a disk page.
- Therefore, that unit size depends on
  - Disk page size
  - Data record size

Summary

- Nearest Neighbor Query
  - Distance metrics of MBR in R-tree
  - Depth-first NN search
  - Best-first NN search
- Spatial Join
  - R-tree based algorithm
  - Multi-step processing
- Implementation of R-tree
  - Data structure size

Readings and Exercises

- Readings
  - Nick Roussopoulos, Stephen Kelley, Frédéic Vincent: Nearest Neighbor Queries. SIGMOD Conference 1995: 71-79
  - Gísli R. Hjaltason, Hanan Samet: Distance Browsing in Spatial Databases. TODS 24(2):265-318, 1999.
  - Thomas Brinkhoff, Hans-Peter Kriegel, Bernhard Seeger: Efficient Processing of Spatial Joins Using R-Trees. SIGMOD Conference 1993: 237-246
- Exercises
  - Those in the slides