CSEG601 & CSE5601
Spatial Data Management & Application :

Spatial Query Processing using R-tree

Sungwon Jung

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

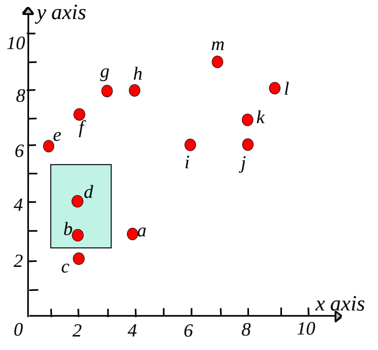
1

Content

- The R-tree
  - Range Query
  - Aggregation Query
- NN Query
- Closest Pair Query
- Close Pair Query

2

R-Tree Motivation

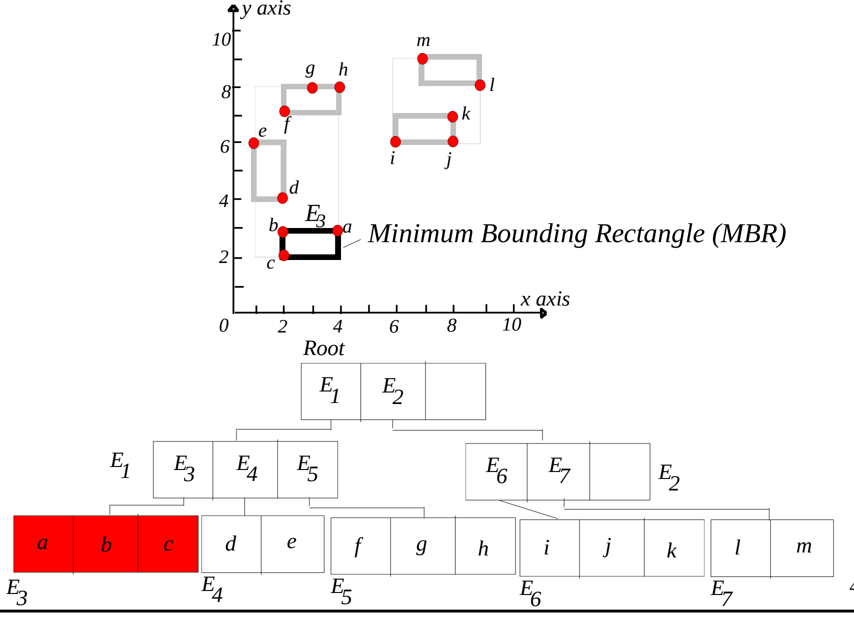


Range query: *find the objects in a given range.*
E.g. find all hotels in Boston.

No index: scan through all objects. NOT EFFICIENT!

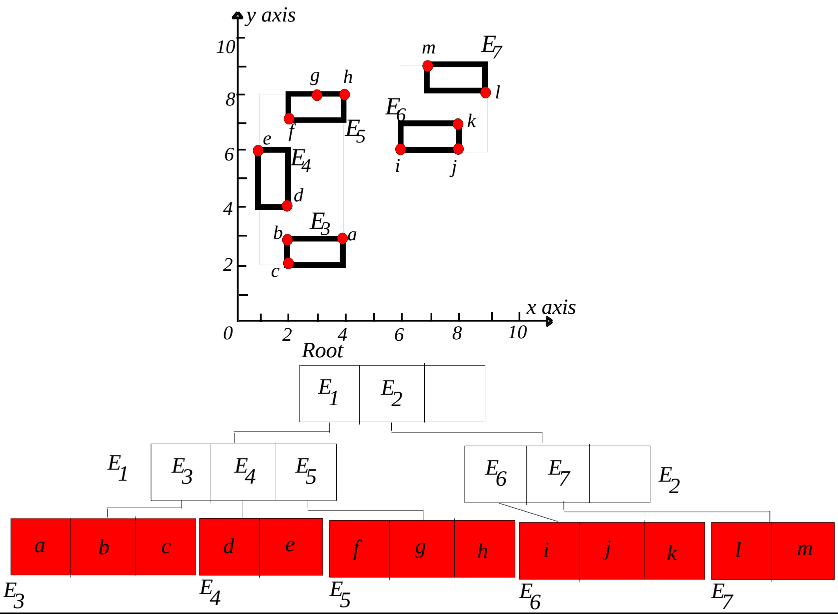
3

R-Tree: Clustering by Proximity



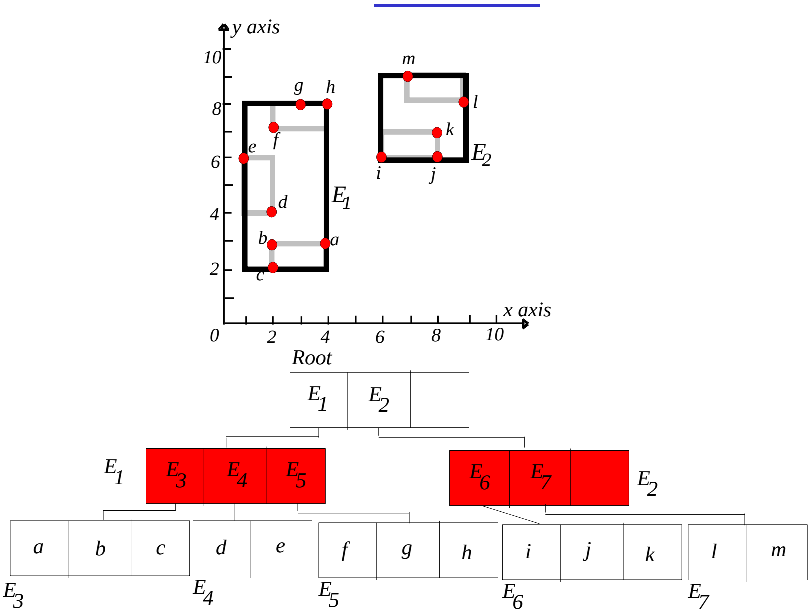
4

R-Tree



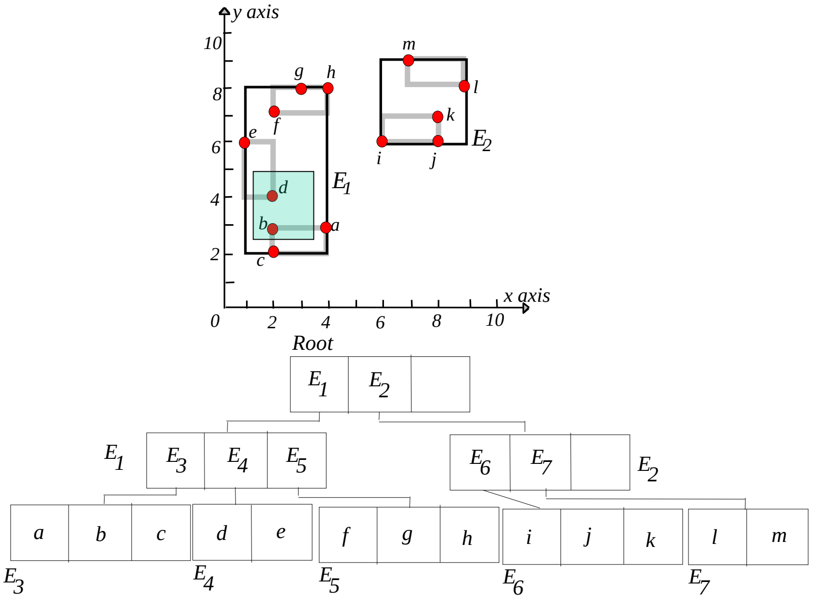
5

R-Tree



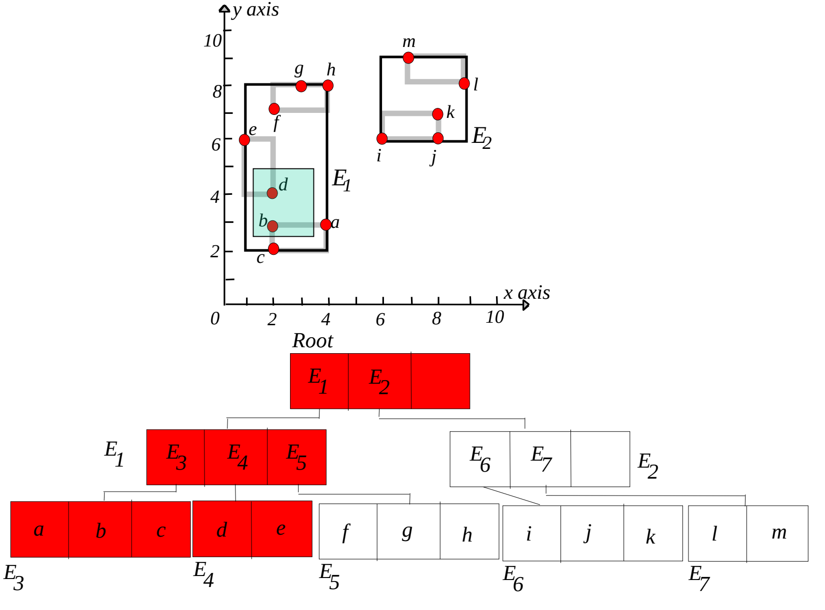
6

Range Query



7

Range Query

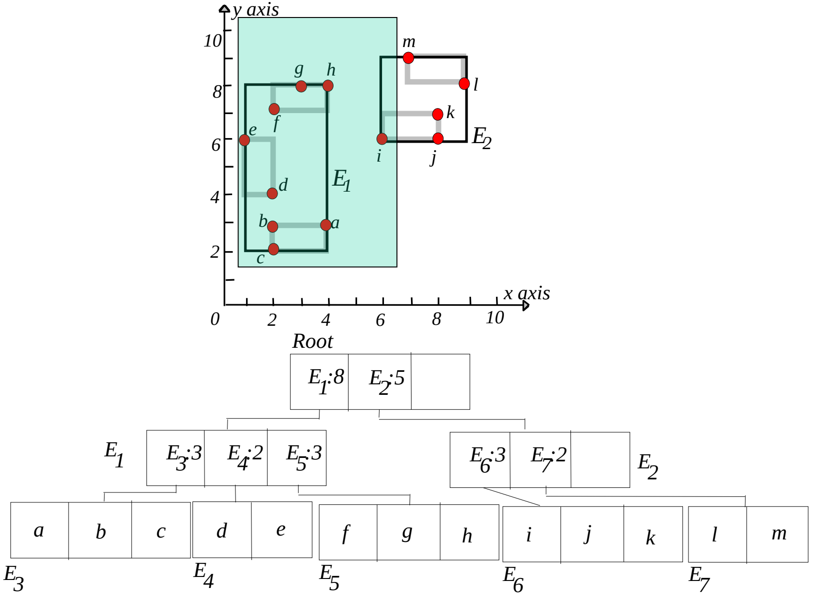


8

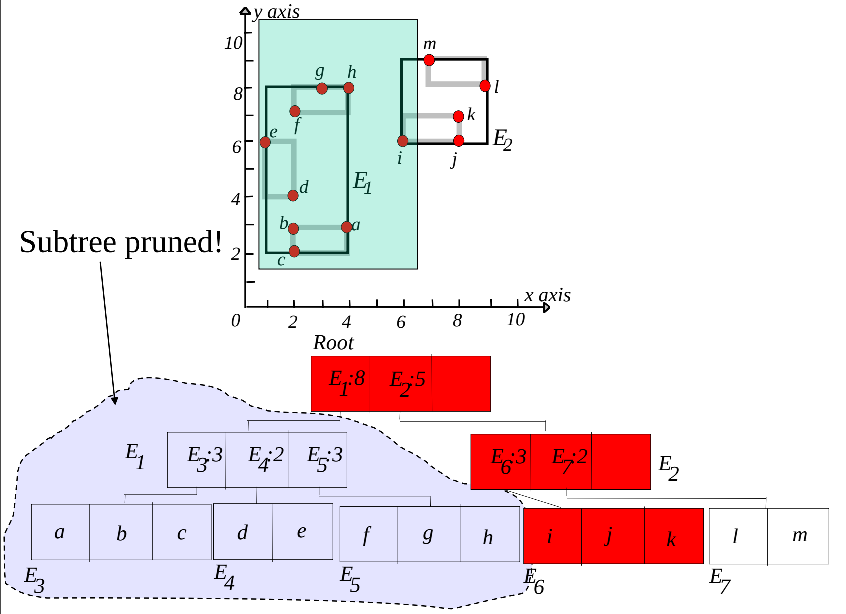
Aggregation Query

- *Given a range, find some aggregate value of objects in this range.*
- COUNT, SUM, AVG, MIN, MAX
- Example: Find the total number of hotels in Massachusetts.
- Straightforward approach: reduce to a range query.
- Better approach: along with each index entry, store aggregate of the sub-tree.

Aggregation Query



Aggregation Query



11

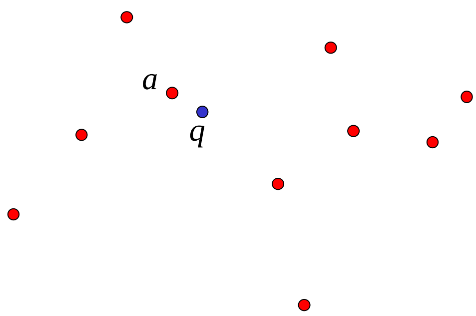
Content

- The R-tree
  - Range Query
  - Aggregation Query
- **NN Query**
- Closest Pair Query
- Close Pair Query

12

Nearest Neighbor (NN) Query

- Given a query location *q*, find the nearest object.

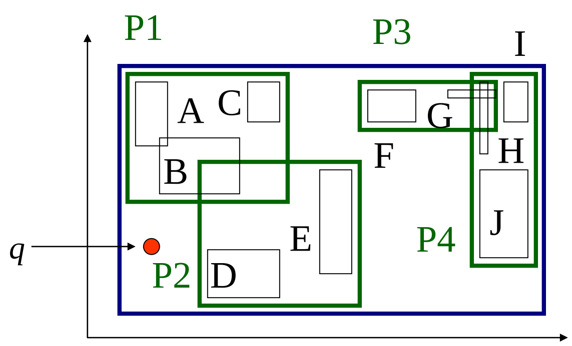


- E.g.: given a hotel, find its nearest bar.

13

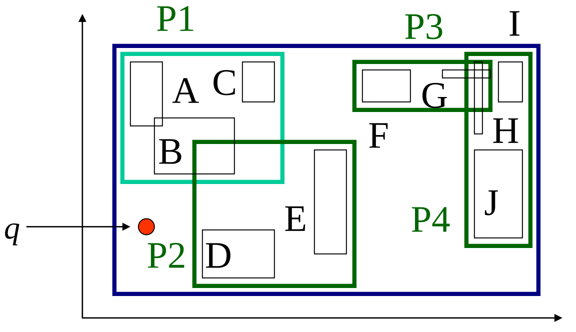
R-trees - NN search

- Q: How? (find near neighbor; refine...)



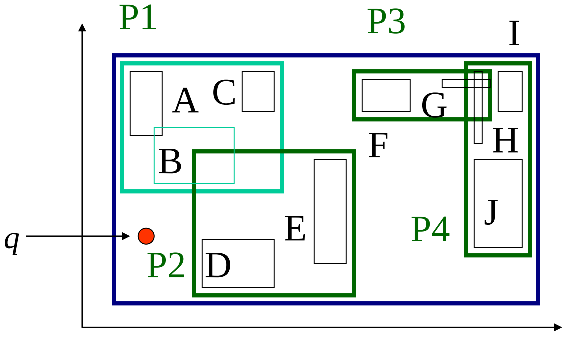
R-trees - NN search

- A1: depth-first search; then range query



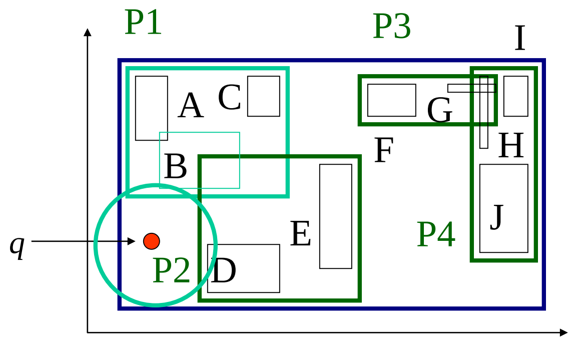
R-trees - NN search

- A1: depth-first search; then range query



R-trees - NN search

- A1: depth-first search; then range query



R-trees - NN search: Branch and Bound

- A2: [Roussopoulos+, sigmod95]:
  - At each node, priority queue, with promising MBRs, and their best and worst-case distance
- main idea: Every face of any MBR contains at least one point of an actual spatial object!

MBR face property

- MBR is a d-dimensional rectangle, which is the minimal rectangle that fully encloses (bounds) an object (or a set of objects)
- MBR f.p.: Every face of the MBR contains at least one point of some object in the database

Search improvement

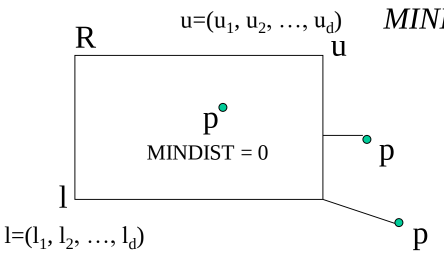
- Visit an MBR (node) only when necessary
- How to do pruning? Using MINDIST and MINMAXDIST

MINDIST

- MINDIST(P, R) is the minimum distance between a point P and a rectangle R
- If the point is inside R, then MINDIST=0
- If P is outside of R, MINDIST is the distance of P to the closest point of R (one point of the perimeter)

MINDIST computation

- 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



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

$r_i = l_i$ if $p_i < l_i$
$= u_i$ if $p_i > u_i$
$= p_i$ otherwise

$$\forall o \in R,\ MINDIST(P,R) \le \|(P,o)\|$$

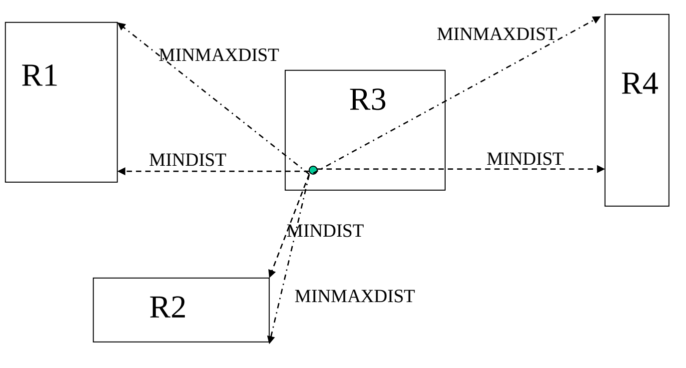
MINMAXDIST

- MINMAXDIST(P,R): for each dimension, find the closest face, compute the distance to the furthest point on this face and take the minimum of all these (d) distances
- MINMAXDIST(P,R) is the smallest possible upper bound of distances from P to R
- MINMAXDIST guarantees that there is at least one object in R with a distance to P smaller or equal to it.

$$\exists o \in R, \|(P,o)\| \le MINMAXDIST(P,R)$$

MINDIST and MINMAXDIST

- MINDIST(P, R) <= NN(P) <=MINMAXDIST(P,R)

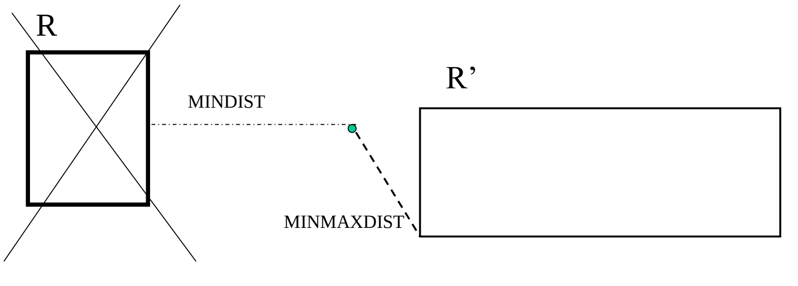


Pruning in NN search

- Downward pruning: An MBR R is discarded if there exists another R’ s.t. MINDIST(P,R)>MINMAXDIST(P,R’)
- Downward pruning: An object O is discarded if there exists an R s.t. the Actual-Dist(P,O) > MINMAXDIST(P,R)
- Upward pruning: An MBR R is discarded if an object O is found s.t. the MINDIST(P,R) > Actual-Dist(P,O)

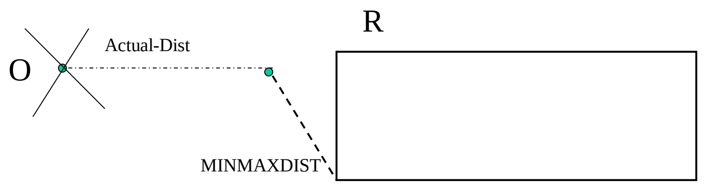
Pruning 1 example

- Downward pruning: An MBR R is discarded if there exists another R’ s.t. MINDIST(P,R)>MINMAXDIST(P,R’)



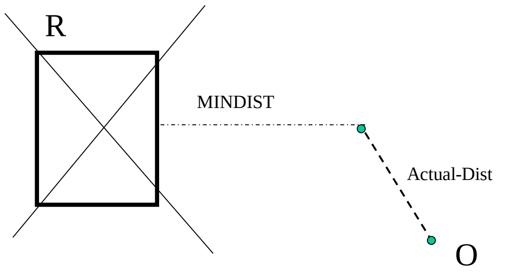
Pruning 2 example

- Downward pruning: An object O is discarded if there exists an R s.t. the Actual-Dist(P,O) > MINMAXDIST(P,R)



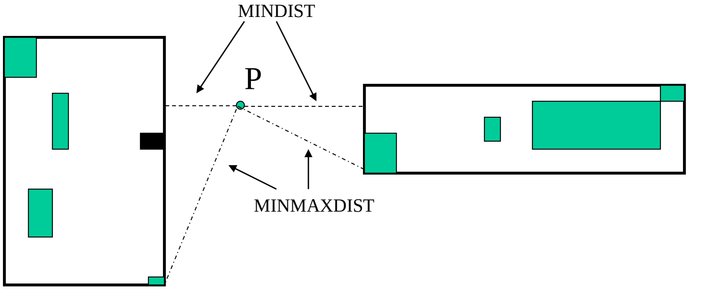
Pruning 3 example

- Upward pruning: An MBR R is discarded if an object O is found s.t. the MINDIST(P,R) > Actual-Dist(P,O)



Ordering Distance

- MINDIST is an optimistic distance where MINMAXDIST is a pessimistic one.



NN-search Algorithm

1. Initialize the nearest distance as infinite distance
2. Traverse the 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.

K-NN search

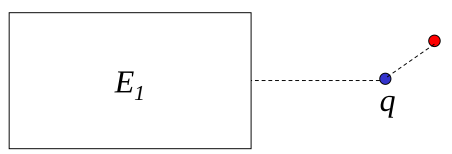
- Keep the sorted buffer of at most k current nearest neighbors
- Pruning is done using the k-th distance

Another NN search: Best-First

- Global order [HS99]
  - Maintain distance to all entries in a common Priority Queue
  - Use only MINDIST
  - Repeat
    - Inspect the next MBR in the list
    - Add the children to the list and reorder
  - Until all remaining MBRs can be pruned

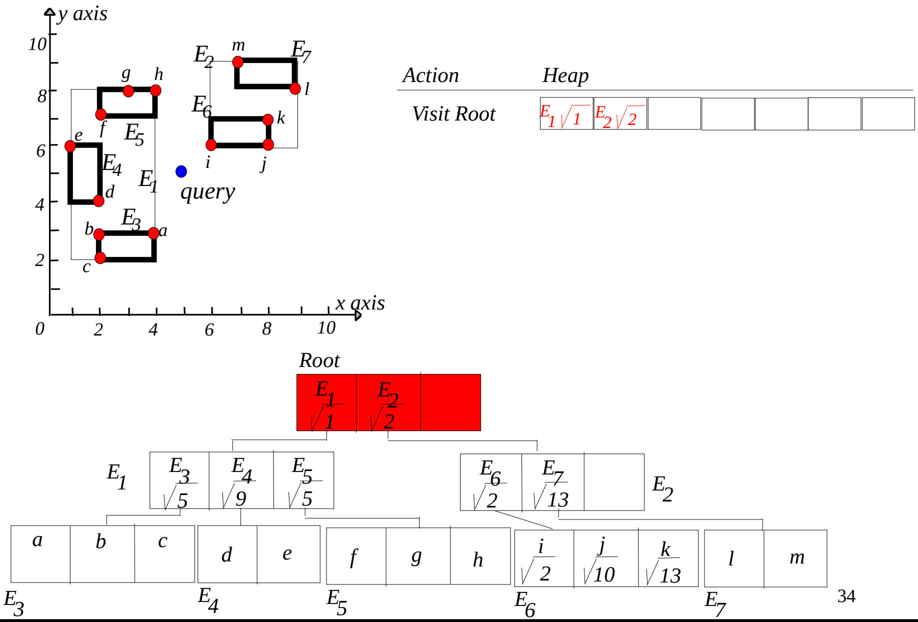
NN Basic Algorithm

- Keep a heap H of index entries and objects, ordered by MINDIST.
- Initially, H contains the root.
- While $H \neq \phi$
  - Extract the element with minimum MINDIST
  - If it is an index entry, insert its children into H.
  - If it is an object, return it as NN.
- End while



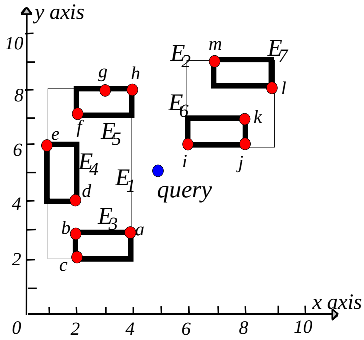
33

NN Query Example

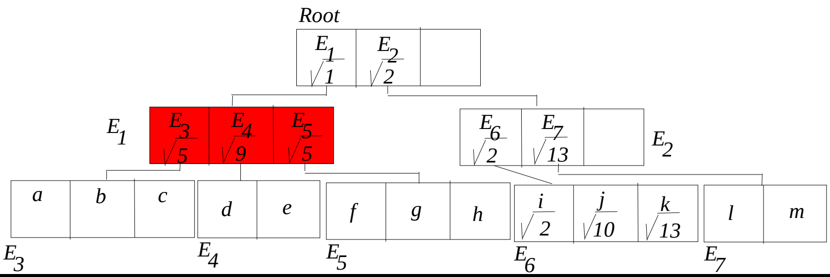


34

NN Query Example

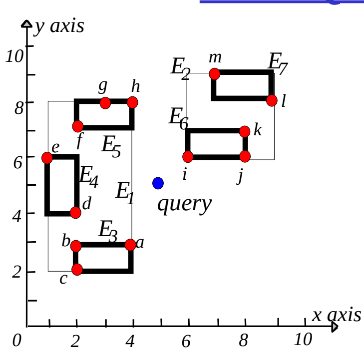


Action	Heap						
Visit Root	$E_1\ \sqrt{1}$	$E_2\ \sqrt{2}$					
follow E_1	$E_2\ \sqrt{2}$	$E_3\ \sqrt{5}$	$E_5\ \sqrt{5}$	$E_4\ \sqrt{9}$			

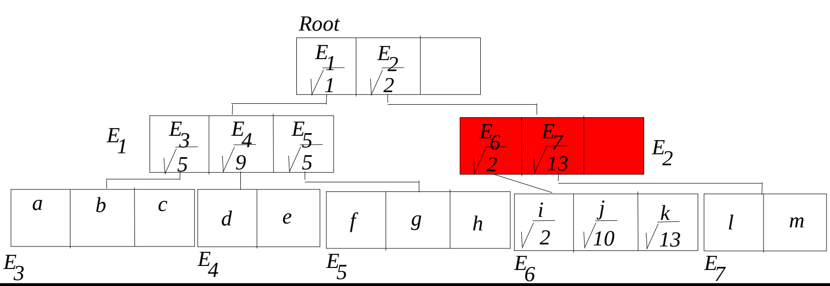


35

NN Query Example

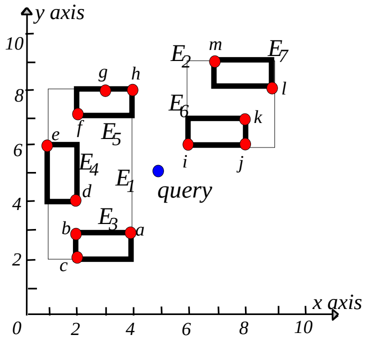


Action	Heap						
Visit Root	$E_1\ \sqrt{1}$	$E_2\ \sqrt{2}$					
follow E_1	$E_2\ \sqrt{2}$	$E_3\ \sqrt{5}$	$E_5\ \sqrt{5}$	$E_4\ \sqrt{9}$			
follow E_2	$E_6\ \sqrt{2}$	$E_3\ \sqrt{5}$	$E_5\ \sqrt{5}$	$E_4\ \sqrt{9}$	$E_7\ \sqrt{13}$		

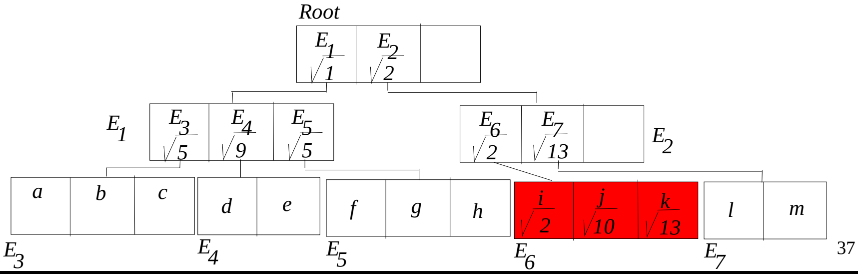


36

NN Query Example

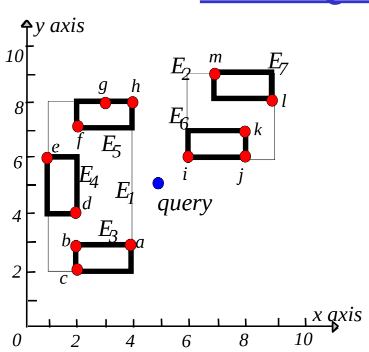


Action	Heap						
Visit Root	E_1 $\sqrt{1}$	E_2 $\sqrt{2}$					
follow E_1	E_2 $\sqrt{2}$	E_3 $\sqrt{5}$	E_5 $\sqrt{5}$	E_4 $\sqrt{9}$			
follow E_2	E_6 $\sqrt{2}$	E_3 $\sqrt{5}$	E_5 $\sqrt{5}$	E_4 $\sqrt{9}$	E_7 $\sqrt{13}$		
follow E_6	i $\sqrt{2}$	E_3 $\sqrt{5}$	E_5 $\sqrt{5}$	E_4 $\sqrt{9}$	j $\sqrt{10}$	E_7 $\sqrt{13}$	k $\sqrt{13}$



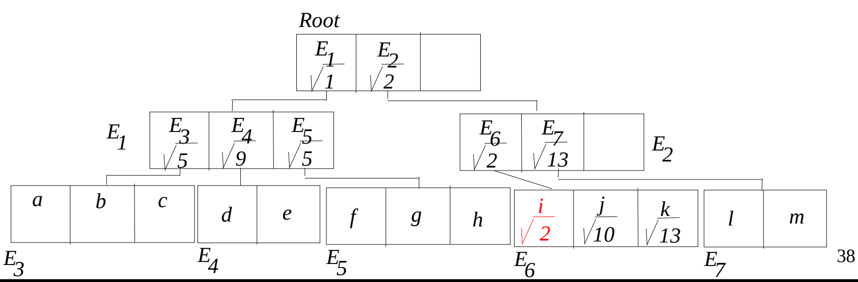
37

NN Query Example



Action	Heap						
Visit Root	E_1 $\sqrt{1}$	E_2 $\sqrt{2}$					
follow E_1	E_2 $\sqrt{2}$	E_3 $\sqrt{5}$	E_5 $\sqrt{5}$	E_4 $\sqrt{9}$			
follow E_2	E_6 $\sqrt{2}$	E_3 $\sqrt{5}$	E_5 $\sqrt{5}$	E_4 $\sqrt{9}$	E_7 $\sqrt{13}$		
follow E_6	i $\sqrt{2}$	E_3 $\sqrt{5}$	E_5 $\sqrt{5}$	E_4 $\sqrt{9}$	j $\sqrt{10}$	E_7 $\sqrt{13}$	k $\sqrt{13}$

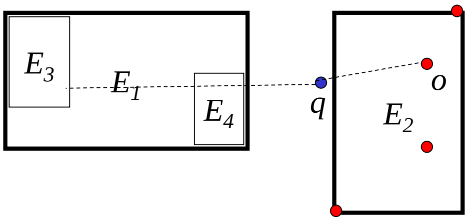
Report i and terminate



38

Pruning 1 in NN Query

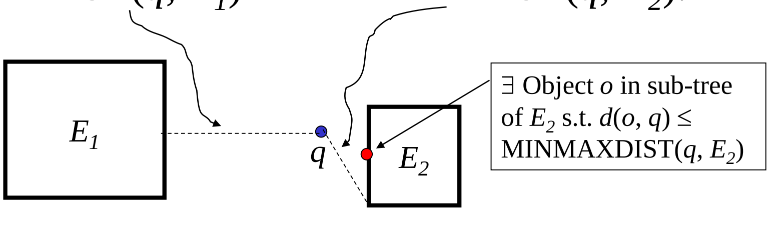
- If we see an object o, prune every MBR whose MINDIST > $d(o, q)$.



- Side notice: at most one object in H!

Pruning 2 using MINMAXDIST

- Prune even before we see an object!
- Prune E_1 if exists E_2 s.t. MINDIST(q, E_1) > MINMAXDIST(q, E_2).



- MINMAXDIST: compute max dist between q and each edge of E_2, then take min.

NN Full-Blown Algorithm

- Keep a heap H of index entries and objects, ordered by MINDIST.
- Initially, H contains the root.
- Set $\delta = +\infty$.
- While $H \neq \phi$
  - Extract the element e with minimum MINDIST.
  - If it is an object, return it as NN.
  - For every entry se in PAGE(e) whose MINDIST $\leq \delta$
    - Insert se into H.
    - Decrease δ to MINMAXDIST(q, se) if possible.
- End while

41

Best-First vs Branch and Bound

- Best-First is the “optimal” algorithm in the sense that it visits all the necessary nodes and nothing more!
- But needs to store a large Priority Queue in main memory. If PQ becomes large, we have thrashing…
- BB uses small Lists for each node. Also uses MINMAXDIST to prune some entries

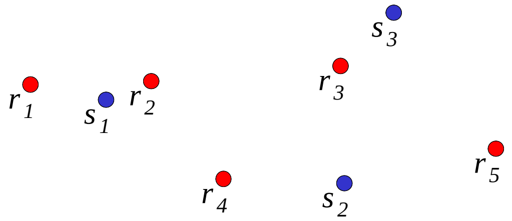
Content

- The R-tree
  - Range Query
  - Aggregation Query
- NN Query
- **Closest Pair Query**
- Close Pair Query

43

Closest Pair (CP) Query

- Given two sets of objects R and S,
- Find the pair of objects ($r \in R$, $s \in S$) with minimum distance.

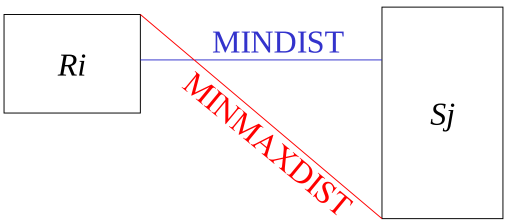


- CP = (r_2, s_1)
- E.g. find the closest pair of hotel-bar.

44

CP Solution Idea

- Assume R and S are indexed by R-trees with same height.
- Similar to the NN query algorithm.
- MINDIST, MINMAXDIST for a pair of MBRs:



45

CP Basic Algorithm

- Keep a heap H of pairs of index entries and pairs of objects, ordered by MINDIST.
- Initially, H contains the pair of roots.
- While $H \neq \phi$
  - Extract the pair (e_R, e_S) with minimum MINDIST.
  - If it is a pair of objects, return it as CP.
  - For every entry se_R in PAGE(e_R) and every entry se_S in PAGE(e_S)
    - Insert(e_R, e_S) into H.
- End while

46

CP Full-Blown Algorithm

- Keep a priority queue H of pairs of index entries and pairs of objects, ordered by MINDIST.
- Initially, H contains the pair of roots.
- Set $\delta = +\infty$.
- While $H \neq \phi$
  - Extract the pair (e_R, e_S) with minimum MINDIST.
  - If it is a pair of objects, return it as CP.
  - For every entry se_R in PAGE(e_R) and every entry se_S in PAGE(e_S) whose MINDIST $\leq \delta$
    - Insert(se_R, se_S) into H.
    - Decrease δ to MINMAXDIST(se_R, se_S) if possible.
- End while

47

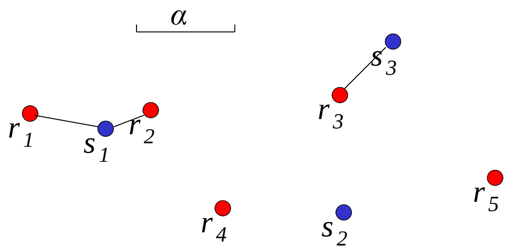
Content

- The R-tree
  - Range Query
  - Aggregation Query
- NN Query
- Closest Pair Query
- **Close Pair Query**

48

Close Pair Query

- Given two sets of objects R and S, plus a threshold α,
- Find every pair of objects ($r \in R$, $s \in S$) with distance $<\alpha$.



- Close pairs = (r_1, s_1), (r_2, s_1), and (r_3, s_3).

49

Close Pair Solution Idea

- Observation: if $d(r, s) <\alpha$, $\forall mbr_R$, mbr_S that contain r and s, respectively, we have: MINDIST(mbr_R, mbr_S) $<\alpha$.
- Solution idea:
  - start with the pair of root nodes,
  - Join pairs of index entries whose MINDIST $<\alpha$,
  - Till we reach leaf level.

50

Close Pair Algorithm

- Push the pair of root nodes into *stack*.
- While *stack* $\neq \phi$
  - – Pop a pair (e_R, e_S) from *stack*.
  - – For every entry se_R in PAGE(e_R) and se_S in PAGE(e_S) where MINDIST(se_R, se_S) $<\alpha$
    - Push (se_R, se_S) into stack if se_R is an index entry;
    - Otherwise report (se_R, se_S) as one close pair.
- End while