
# R-TREES: A Dynamic Index Structure For Spatial Searching

## Antonin Guttman

Delip Rao
(CS04S050)

**delip@cse.iitm.ernet.in**

# Source

- Antonin Guttman, R-TREES: A Dynamic Index Structure For Spatial Searching
- Jeremy Cook, Some R-Tree Results
- J L Bentley and J H Friedman, Data Structures for Range Searching, Computing Surveys 11, 4

# Outline

- Motivation
- Organization
- Search
- Insertion
- Deletion

# Motivation: Indexing Spatial data

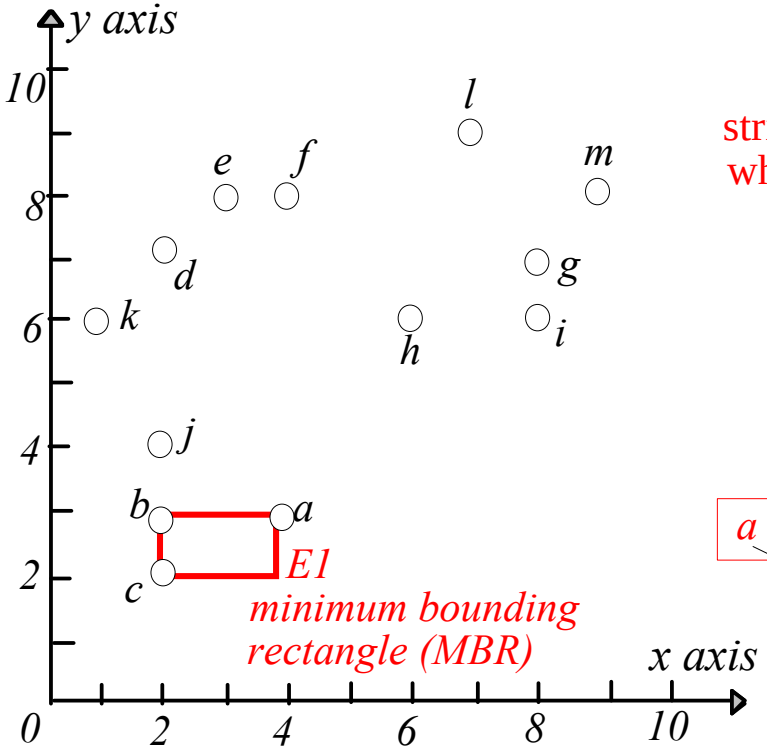
- What is spatial data?
  - Points
  - Lines
  - Rectangles …
- Where do they come from?
  - CAD
  - GIS
  - VLSI
  - Computer Vision
  - Genome databases etc
- We want efficient retrieval (indexing) over such data

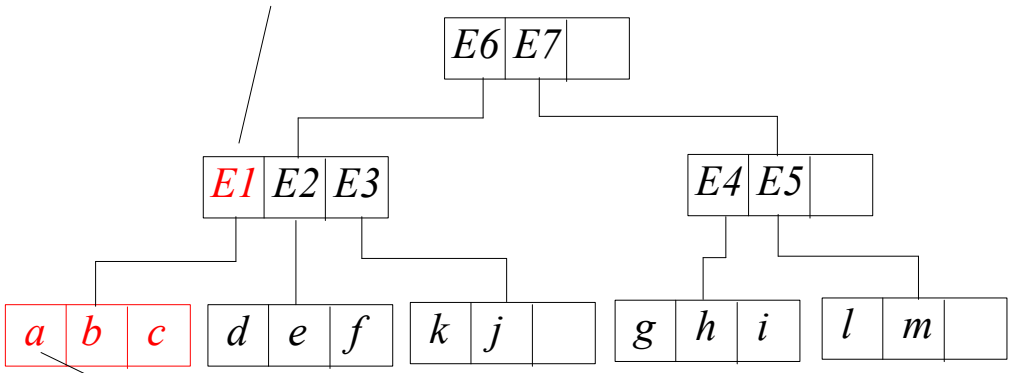
# Review of spatial data structures

- Cell methods – static!
- Quad trees and k-d trees – poor paging performance!
  - Cannot use B-Trees – one dimensional!
- K-D-B trees – Only for points!
- More details read survey by [Bentley and Friedman]

# R-Tree overview

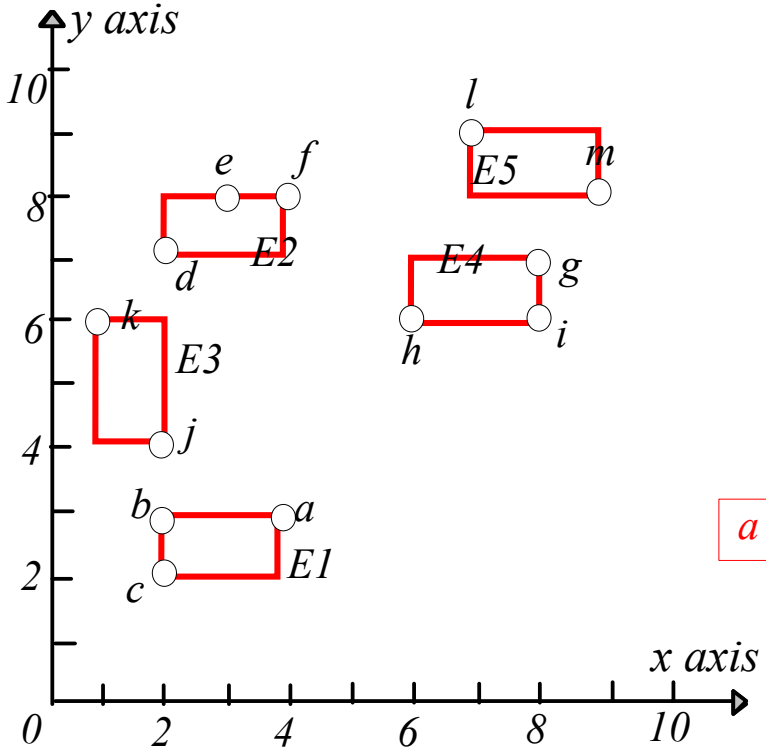
- Each non-leaf entry stores a minimum bounding rectangle (MBR) that encloses all the points in the subtree.

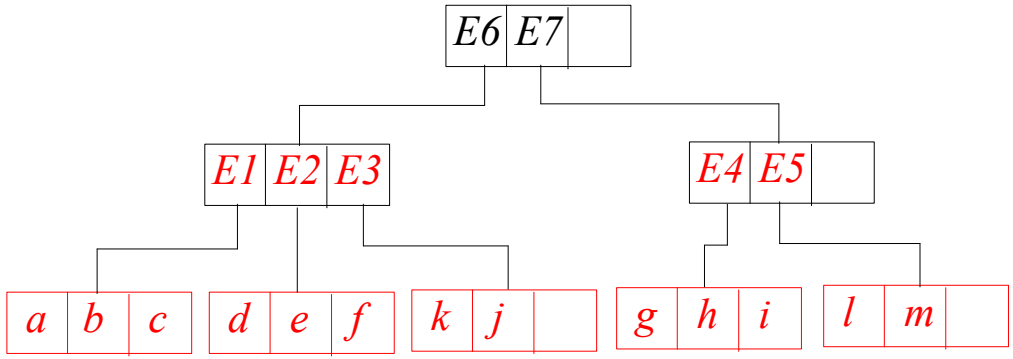


# R-Tree overview

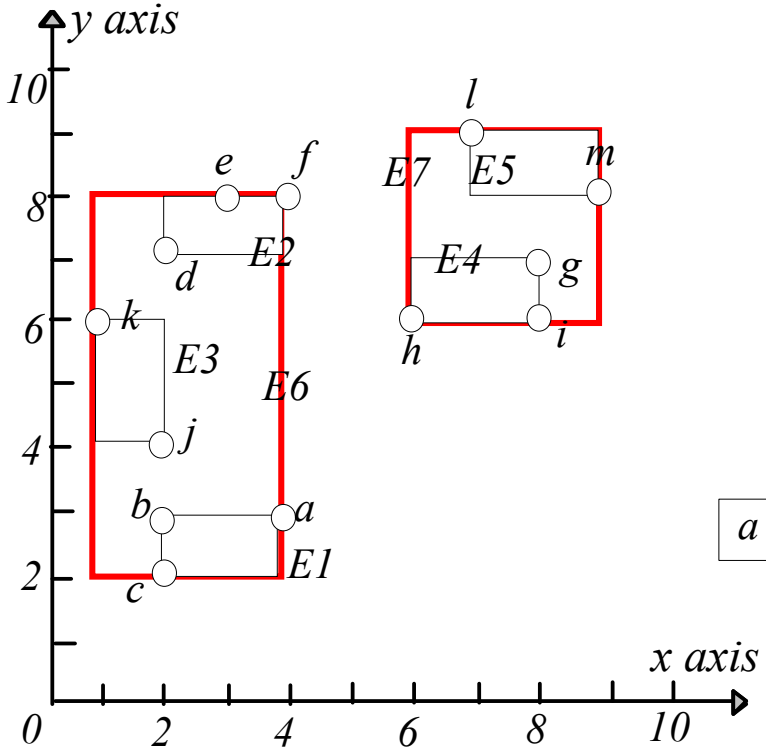
- Each non-leaf entry stores a minimum bounding rectangle (MBR) that encloses all the points in the subtree.

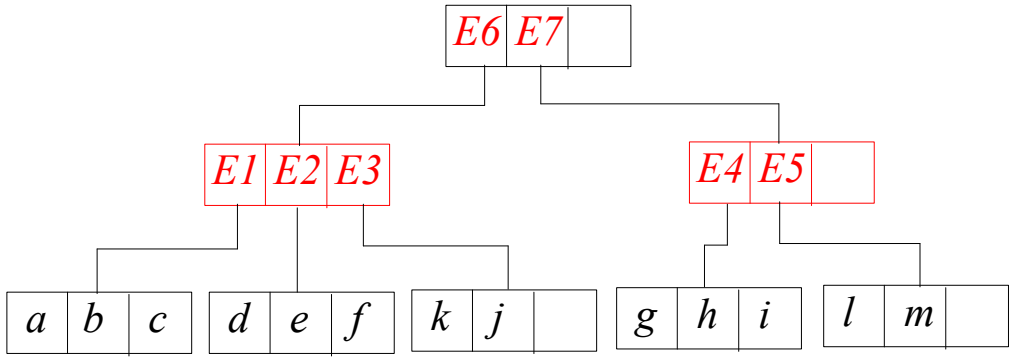


# R-Tree overview

- Each non-leaf entry stores a minimum bounding rectangle (MBR) that encloses all the points in the subtree.



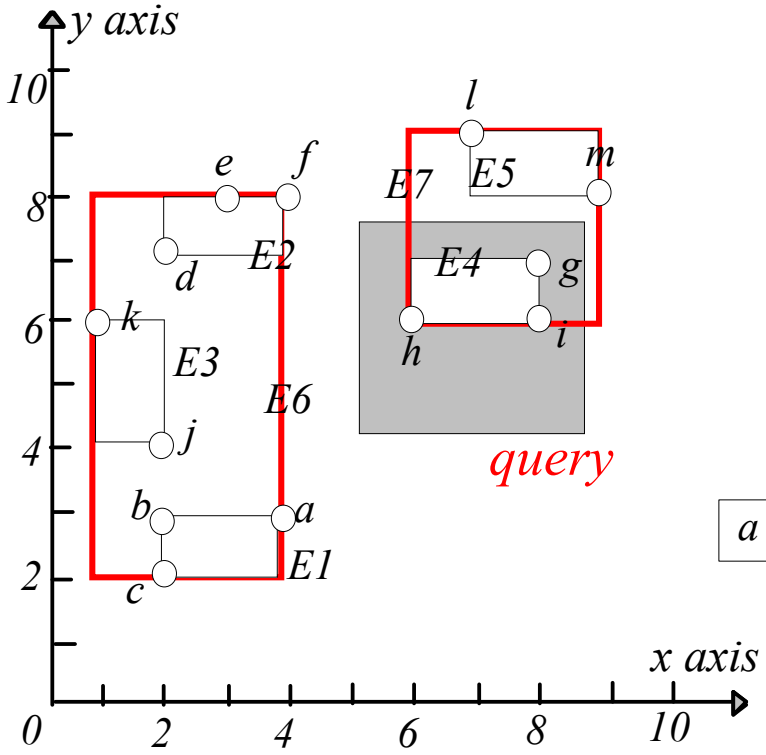



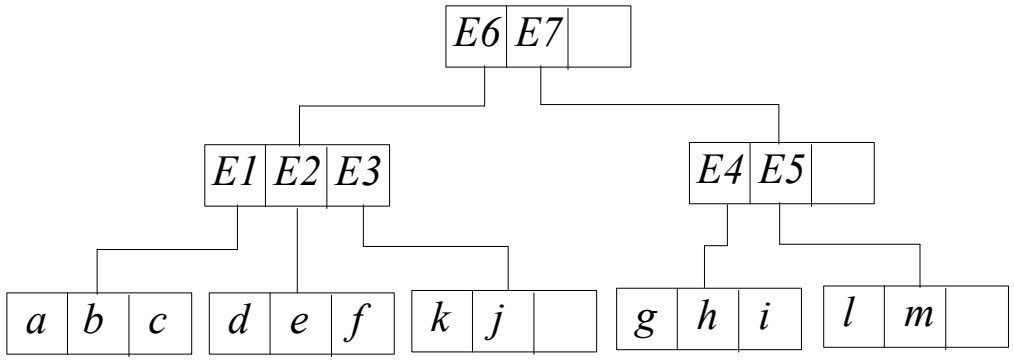
# R-Tree Properties

- All leaves appear at same level
- The root node has at least two children unless it is a leaf
- Every leaf node has between $m$ and $M$ index records unless it's the root, $m \leq M/2$
- Every non-leaf node has between $m$ and $M$ children unless it's the root
- Height of an R-Tree of N nodes is at most $\lceil log_m N \rceil - 1$

# Processing range search

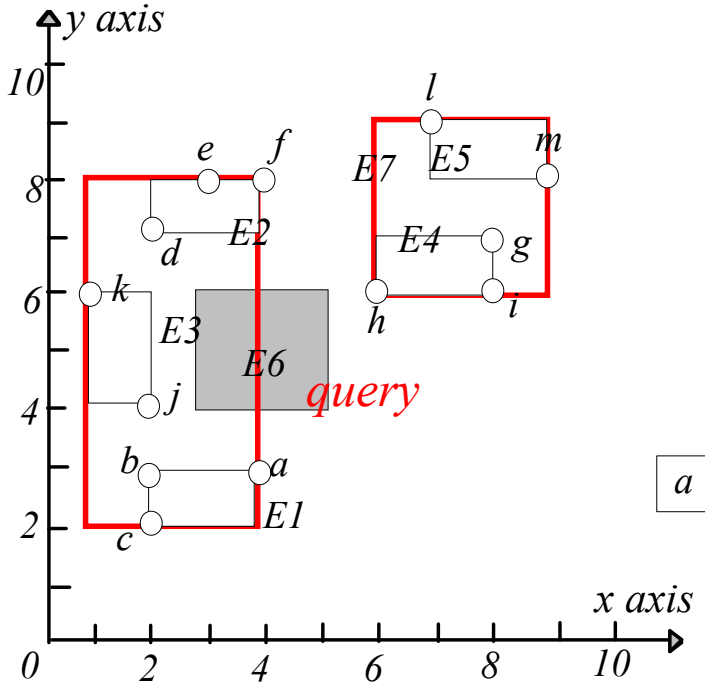
- SELECT…FROM…WHERE x in [5,8.5] & y in [4, 7.5]
- The query region is a rectangle q
  - What about k-d tree?
- Visit nodes whose MBRs intersect the query q

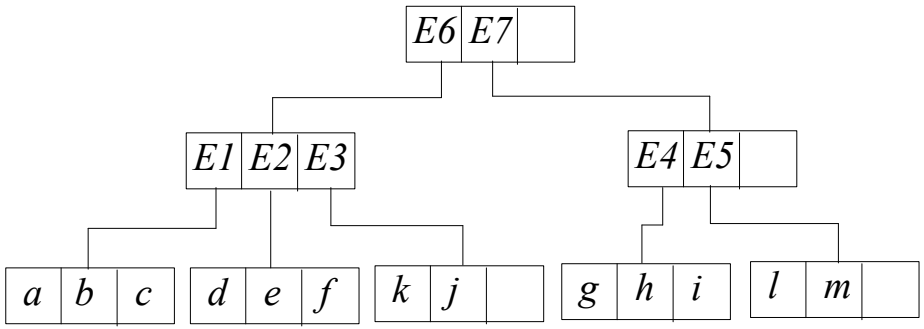


# Another example

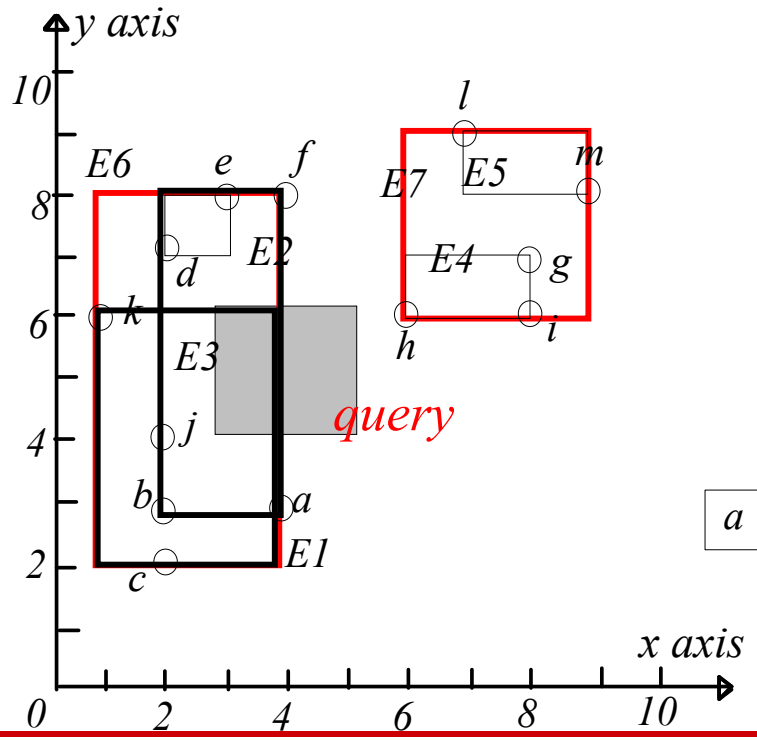
- SELECT…FROM…WHERE x in [3, 5] & y in [4, 6]
- Only E6 visited and then we know there is no tuple satisfying this query
  - While the B-tree on x-axis would find e, f, a

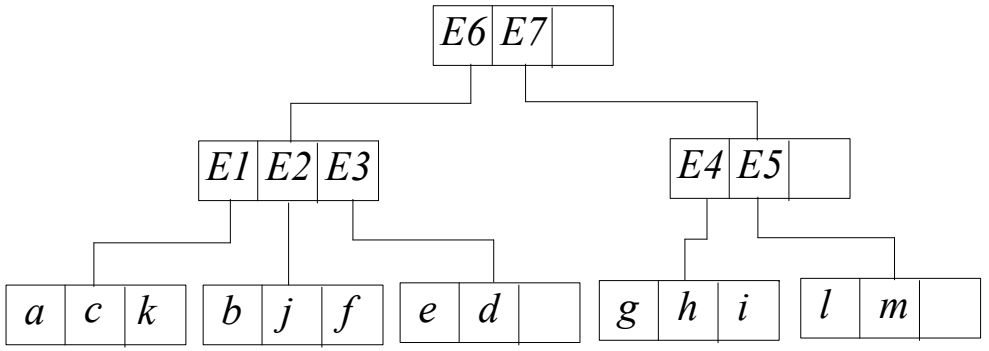


# Grouping of leaf points

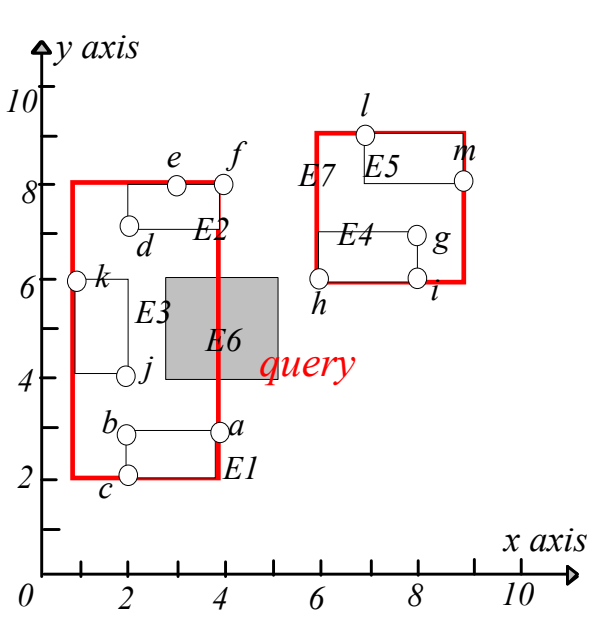
- Which points are put in the same node are not unique
  - Particularly, the MBRs can overlap

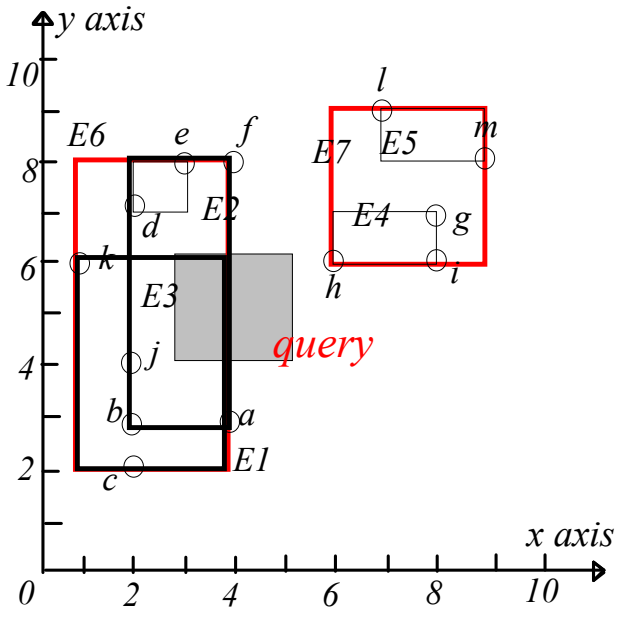


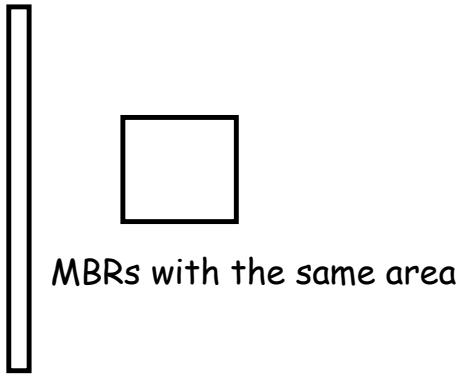



# Which grouping is better?

- The first one
  - Intuitively, the MBRs there are smaller
  - So they're less likely to intersect a query
- Indeed, the R-tree tries to minimize the perimeters of the MBRs
  - Why not areas of the MBRs?
  - Small perimeter leads to small area, but not the vice versa

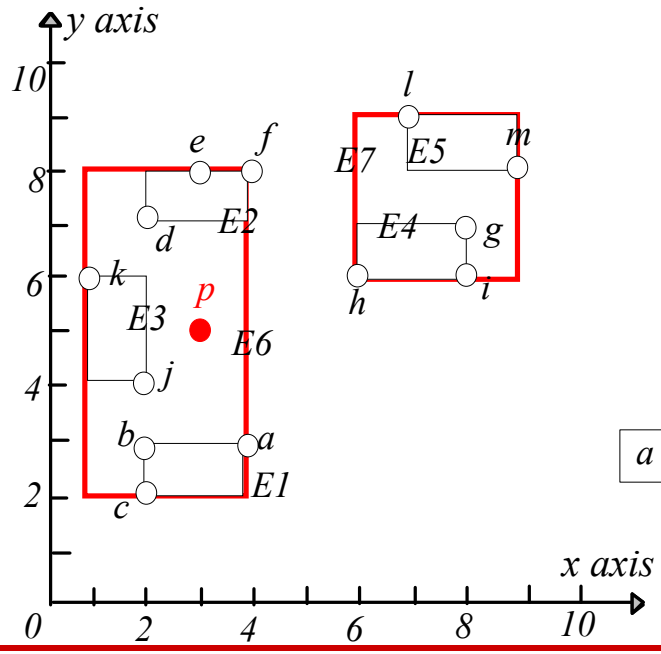


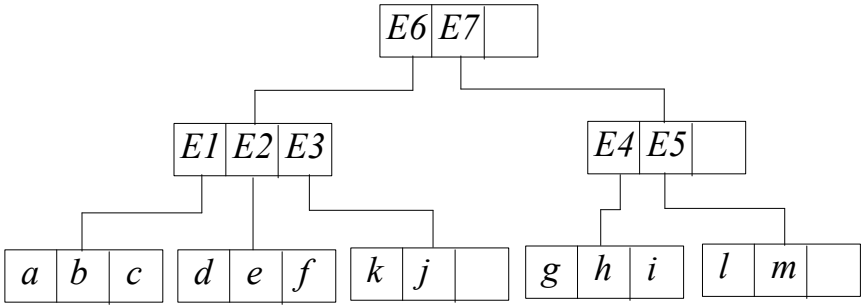



# Insertion

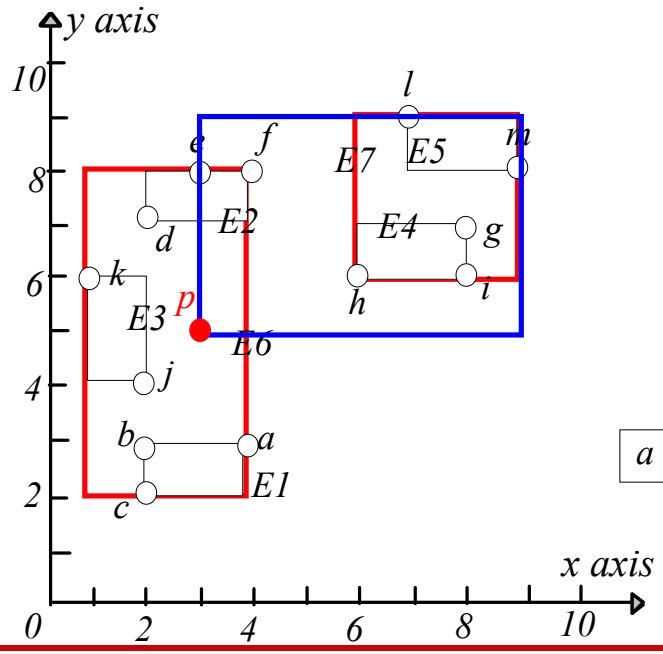
- Assume we want to insert a point p(3,5)
- First retrieve the root, and decide which subtree (E6 or E7) to insert, considering the new MBR of the non-leaf entry in order to keep the invariant
  - Inserting p into E6 does not require enlarging its MBR

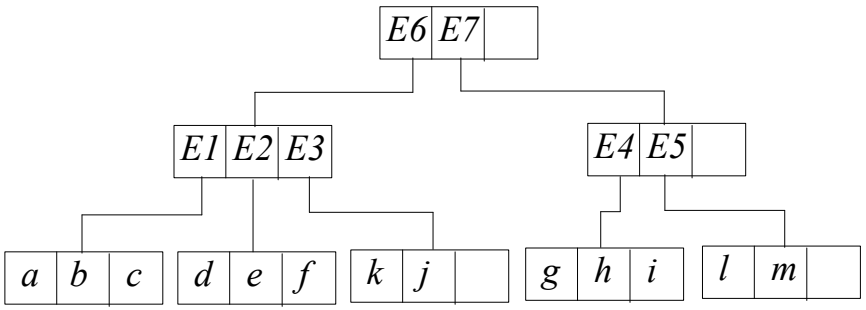


# Insertion

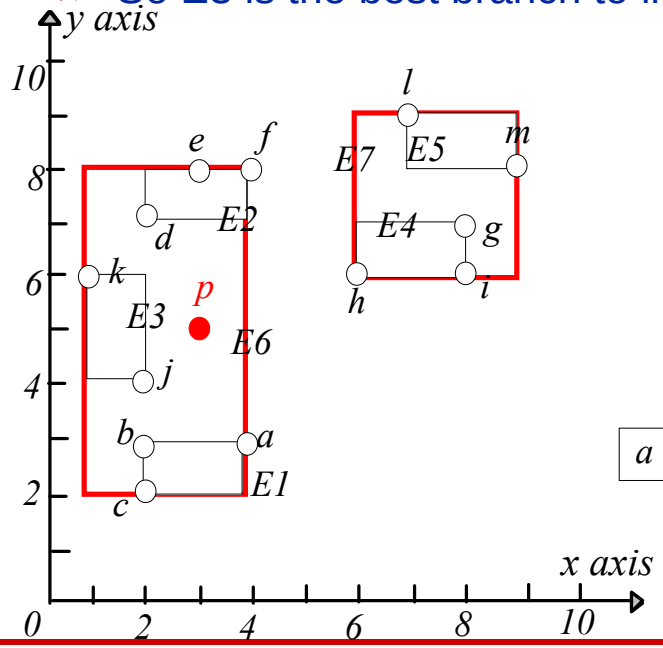
- Inserting p into E7 requires considerable enlargement
- The insertion algorithm calculates for each non-leaf entry the increased MBR perimeter if p is inserted into this entry
- And selects the entry with minimum increase
  - In this case E6 (zero increase)

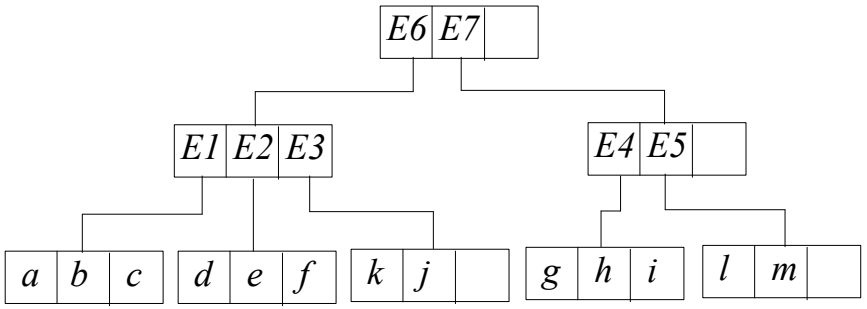


# Insertion

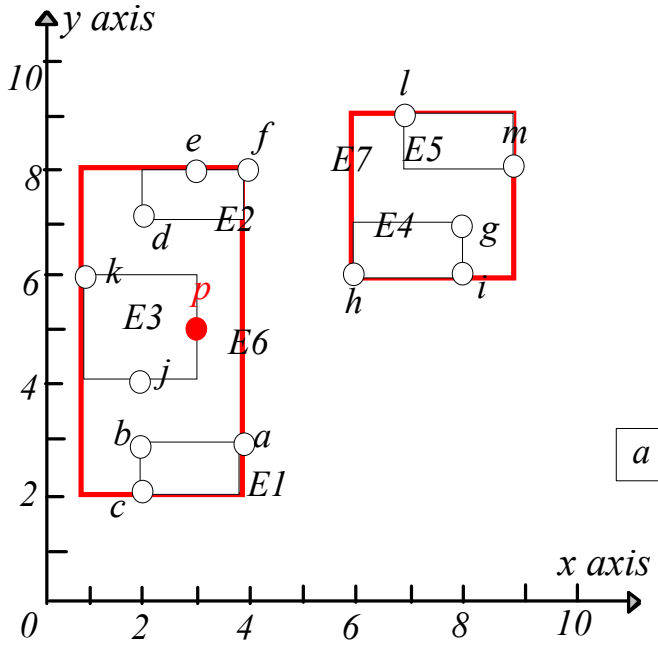
- Now we are at E6, which is also a non-leaf node
- Once again, we need to select the sub-tree to insert
  - E1, E2, or E3?
- Calculate the perimeter increase again
  - For E3: 2
  - E2:4, E1:4
  - So E3 is the best branch to insert

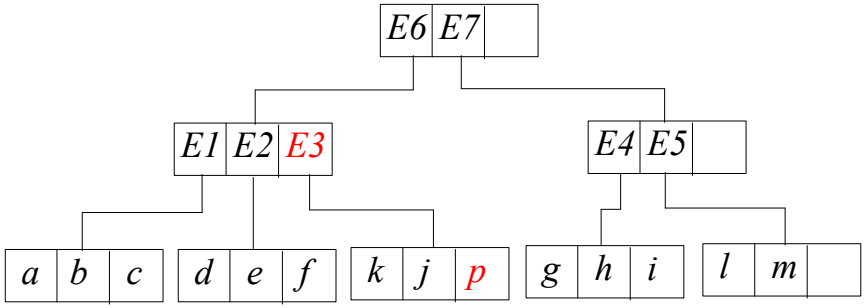


# Insertion

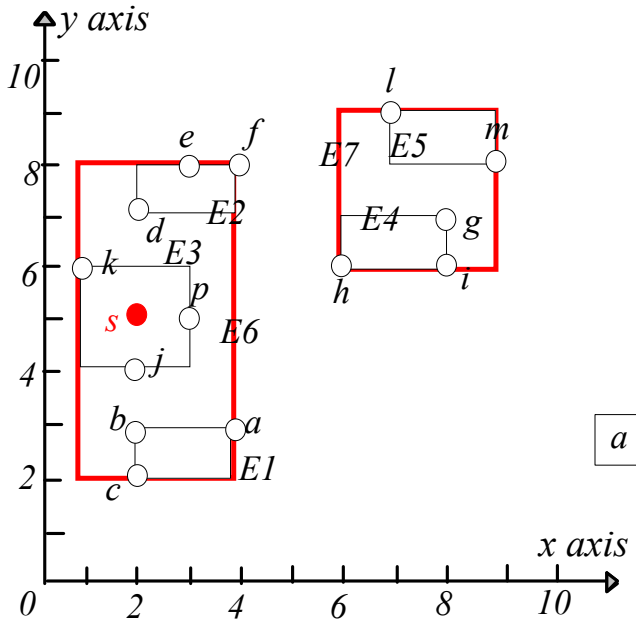
- So we are at E3 now, which is a leaf node
- Just put p there
- The final tree
  - The red entries (E3 and p) were modified during the insertion process

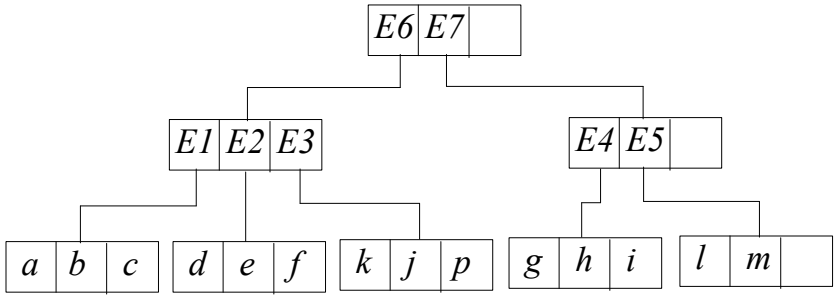


# Insertion

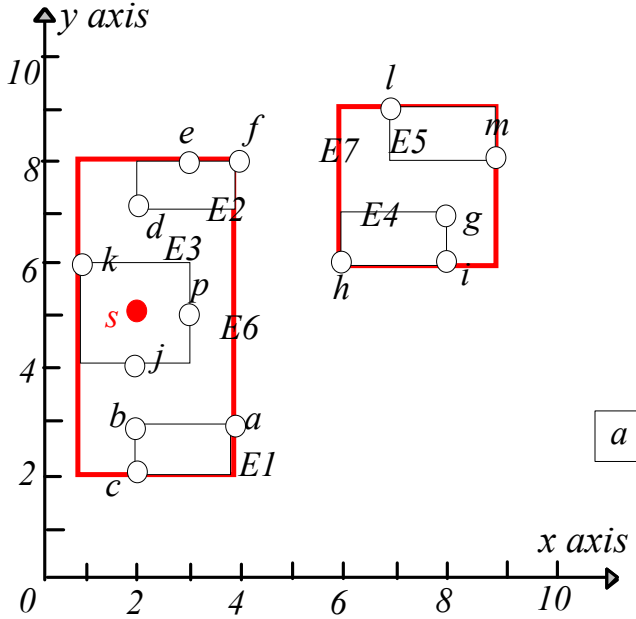
- Assume that we're inserting another point s (2,5)
- By the above algorithm we decide it should be inserted into E3
  - No MBR enlargement is required
- E3 overflows (node capacity=3)
  - Treated by split

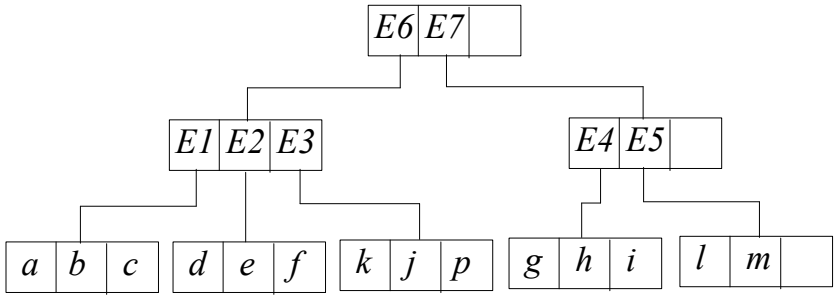


# Node split

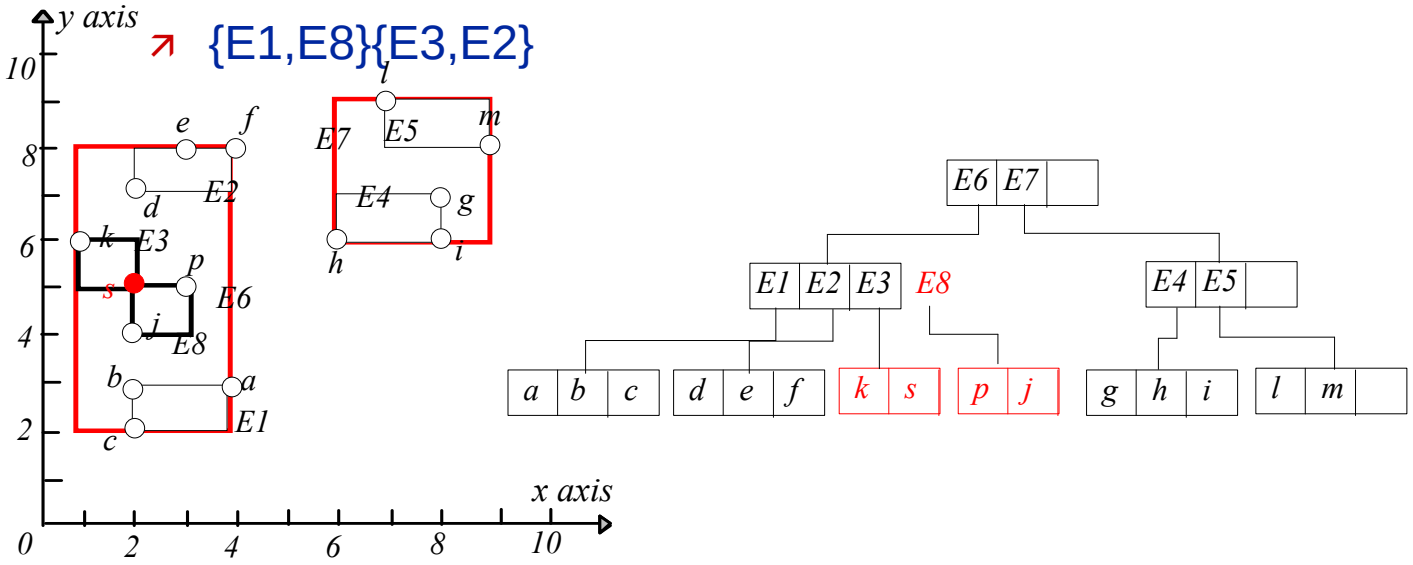
- Distributes the entries evenly
- Based on 2 sortings on the x-, y-axes respectively
  - {k, s, j, p} on x and {j, p, s, k} on y
- So we create two nodes containing k,s and j,p respectively.



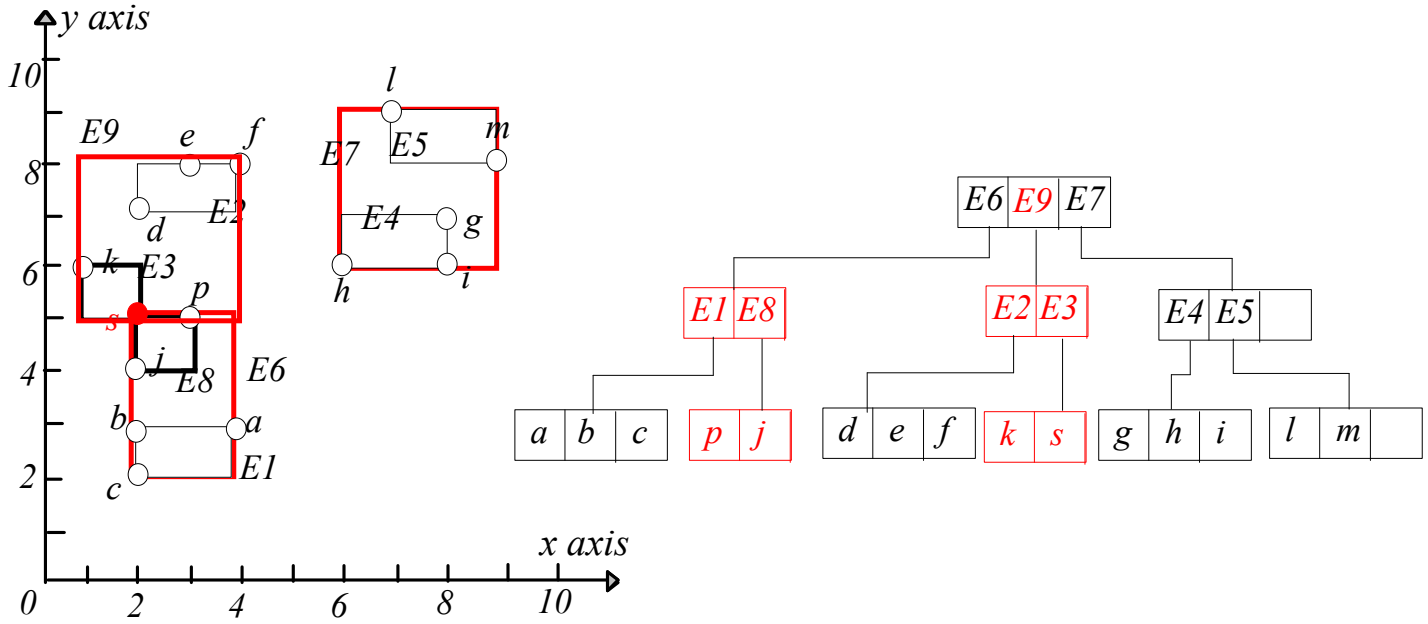
# Node split

- Creates a non-leaf entry in node E6
  - E6 overflows and is split
- Take the centroids of the MBRs and sort them on x-, y-axes respectively
  - For x: {E3,E8,E1,E2}, for y: {E1,E8,E3,E2}
- So 2 possible splits {E3,E8}{E1,E2} and {E1,E8}{E3,E2}
- Take the one with smaller perimeter sum
  - {E1,E8}{E3,E2}



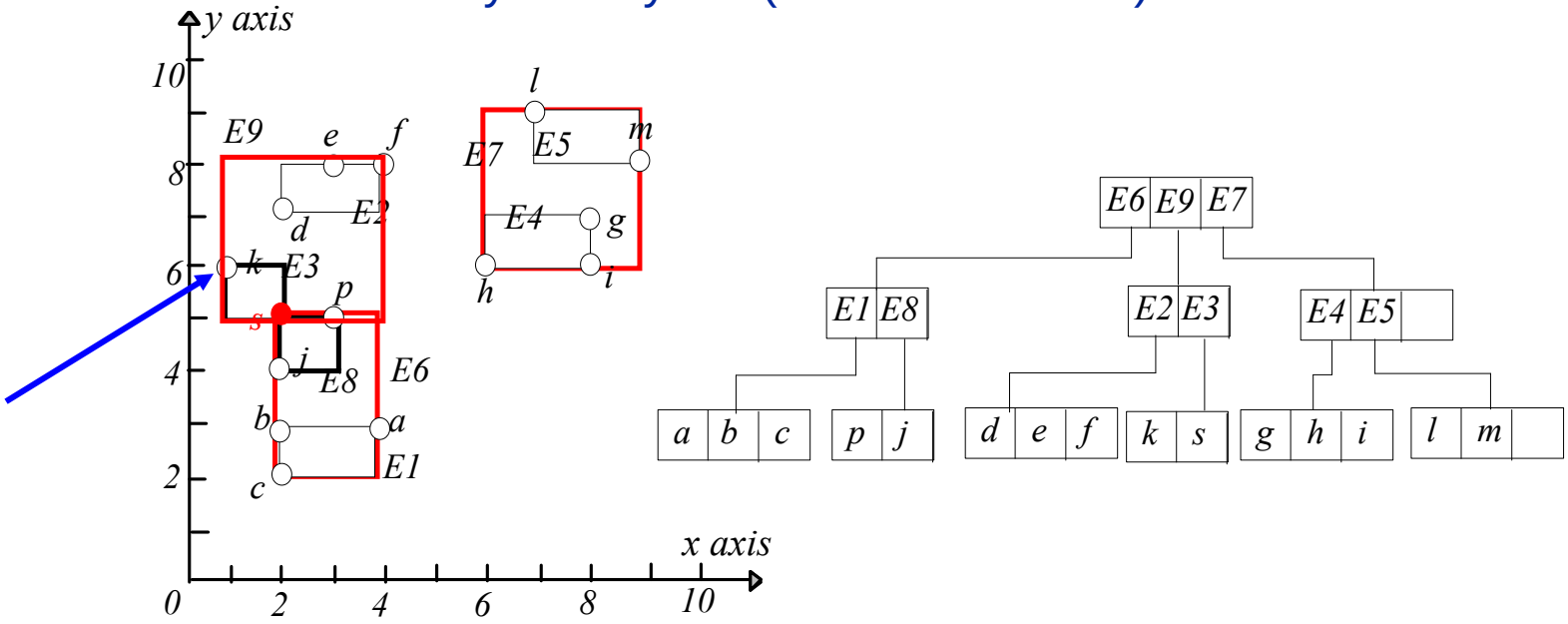
# Node split

- Finally, insert an entry into the root



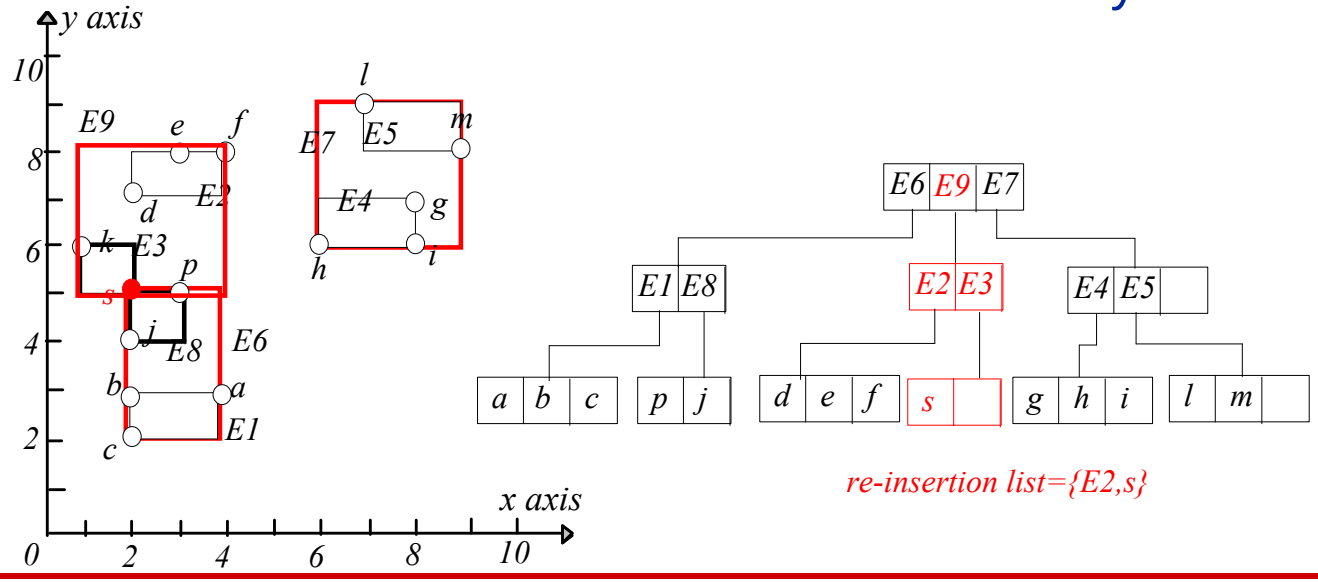
# Deletion

- Assume we want to delete point k(1,6)
- First we find it in E3
  - Performing a search using the coordinates of k
    - Accessing root, E9, E3
- Remove it
  - Node E3 only 1 entry < 2 (min. node fanout)



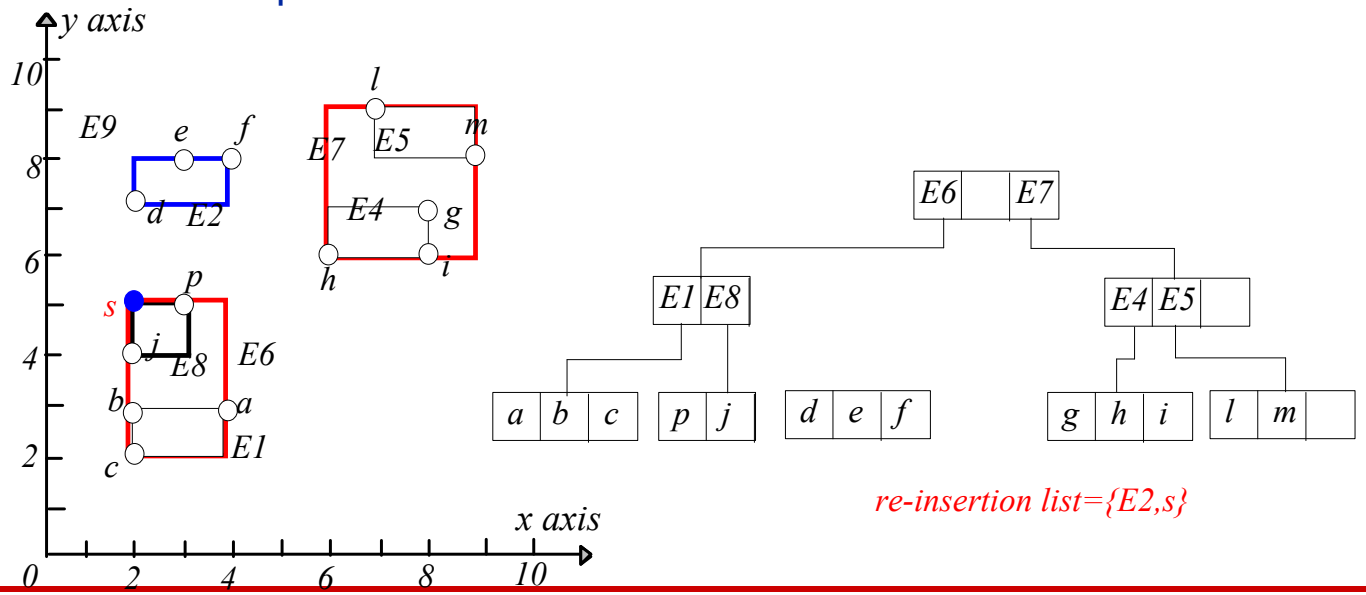
# Underflow

- We do not handle the underflow now, but simply keep s, the entry in the node that underflows, in a reinsertion list
- Now go back to the previous level
  - Must discard E3, which leads to the underflow of E9
- Once again, do not handle it, but simply keep entry E2 in the reinsertion list
- Removal of E9 leaves 2 entries in the root. Okay



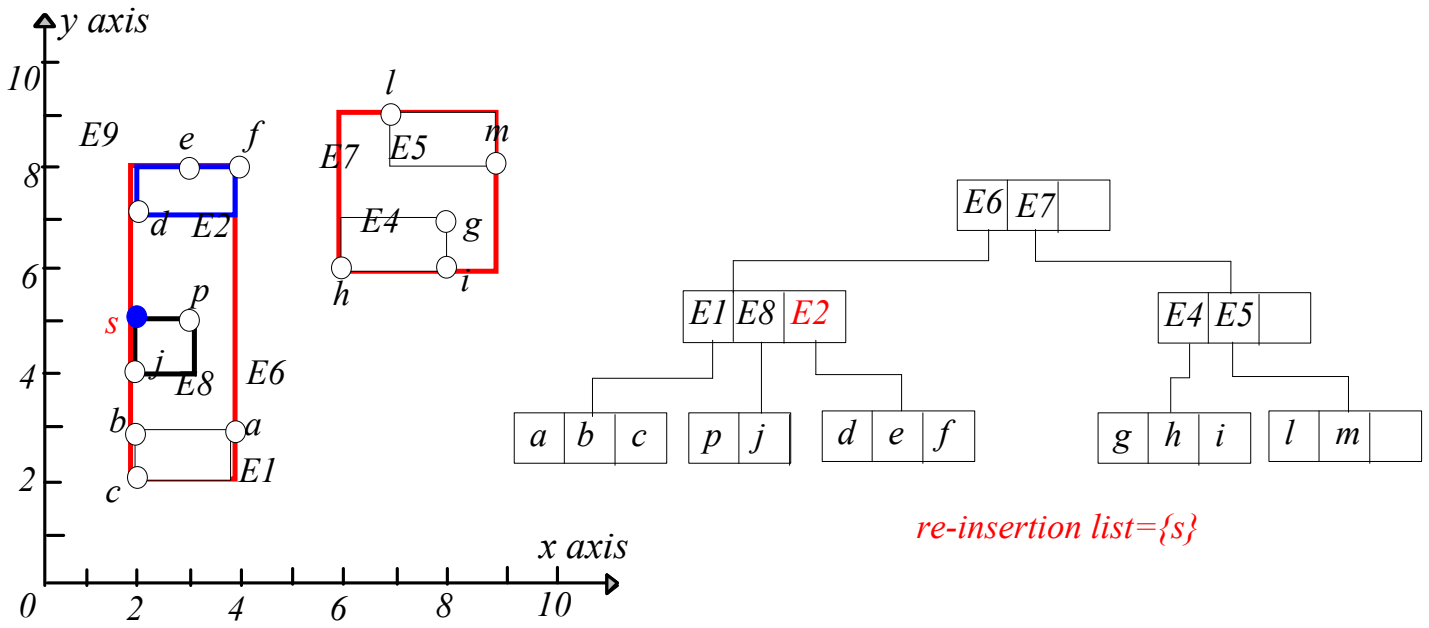
# Re-insertion

- Node [d,e,f] (originally pointed to by E2) is temporarily orphaned
- The non-leaf entry E2 in the re-insertion list is first re-inserted into the tree
  - Similar to inserting a point, except that the insertion stops at level 1, i.e., the previous level E2 was at before
- At root
  - Insertion into E6 requires 6 perimeter increase
  - Into E7 requires 8



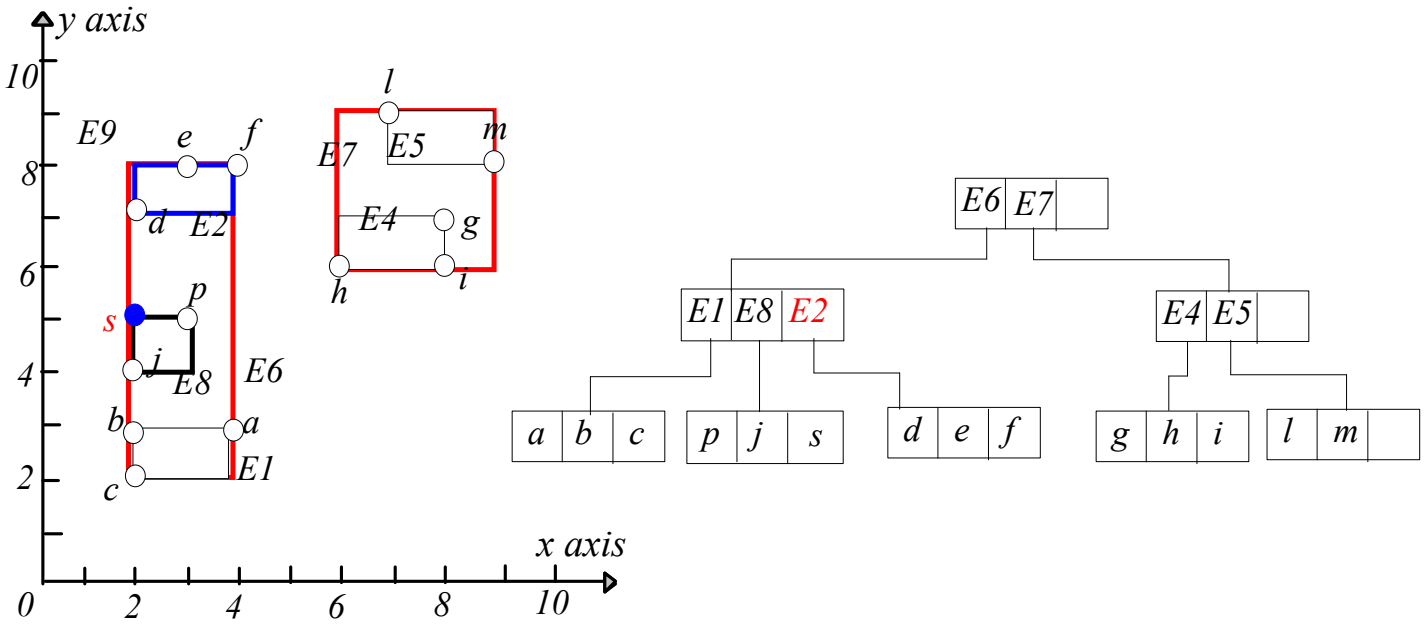
# Re-insertion

- Node [d,e,f] is immediately linked to E2 after it is inserted
- Finally, we re-insert s into E8



# Re-insertion

- Re-insertion completes
- Node underflows give the tree a chance to re-distribute the entries



# Explorations

- Improvement over naive R-Tree
  - Different node splitting techniques possible
  - R+ Trees
  - R* Trees
- Other Indexing structures
  - Quad tree
  - X-Tree

# Summary

- Spatial index structures
- R-Tree
  - Overview
  - Search
  - Insert
  - Delete