DSA

Tôn Huỳnh Chí

March 2025

1 Introduction

The R-tree was proposed by Guttman in 1984, and aimed at handling geometrical data, such as points, line segments, surfaces, volumes, and hypervolumes in high-dimensional spaces.

R-trees are hierarchical data structures based on B+ trees *(self-balancing tree data structure used in databases and file systems for efficient indexing and range searches)*. They are used for the dynamic organization of a set of d-dimensional geometric objects representing them by the minimum bounding d-dimensional rectangles *(MBRs)*. Each node of the R-tree corresponds to the MBR that bounds its children. The leaves of the tree contain pointers to the database objects instead of pointers to children nodes.

It must be noted that the MBRs that surround different nodes may overlap each other. Besides, an MBR can be included *(in the geometrical sense)* in many nodes, but it can be associated to only one of them. This means that a spatial search may visit many nodes before confirming the existence of a given MBR. Also, it is easy to see that the representation of geometric objects through their MBRs may result in false alarms. To resolve false alarms, the candidate objects must be examined. For instance, Figure 1 illustrates the case where two polygons do not intersect each other, but their MBRs do. Therefore, the R-tree plays the role of a filtering mechanism to reduce the costly direct examination of geometric objects.

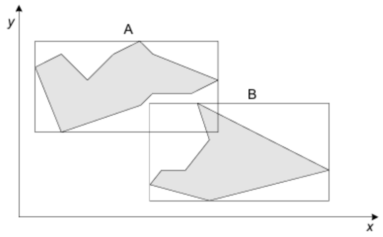


Fig. 1: *An example of intersecting MBRs, where the polygons do not intersect.*

2 Structure

An R-tree of order (m,M) has the following characteristics:

- Each leaf node (unless it is the root) can host up to M entries, whereas the minimum allowed number of entries is m $\leq$ M/2. Each entry is of the form (mbr,oid), such that mbr is the MBR that spatially contains the object and oid is the object's identifier.

- The number of entries that each internal node can store is again between m $\leq$ M/2 and M. Each entry is of the form (mbr,p), where p is a pointer to a child of the node and mbr is the MBR that spatially contains the MBRs contained in this child.

- The minimum allowed number of entries in the root node is 2, unless it is a leaf (in this case, it may contain zero or a single entry).

From the definition of the R-tree, it follows that it is a **height-balanced tree**. As mentioned, it comprises a generalization of the B+-tree structure for many dimensions. R-trees are **dynamic data structures**, i.e., global reorganization is not required to handle insertions or deletions.

Figure 2 show a set of the MBRs of some data geometric objects.

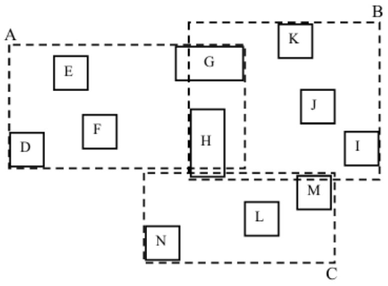


Fig. 2: *A, B, C is the MBRs of internal nodes, the others are the MBRs of the data objects.*

Figure 3 show the R-tree that corresponds to the MBRs of Figure

2. The root node contains the MBRs of the internal nodes A, B, and C. The MBRs of the data objects are stored in the leaves of the tree.

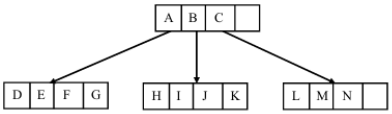


Fig. 3: *An R-tree that corresponds to the MBRs of Figure 2.*

Let an R-tree store N data rectangles. In this case the maximum value for its height h is: $h_{max} = \lceil \log_2 N \rceil - 1$

The maximum number of nodes can be derived by summing the maximum possible number of nodes per level. This number comes up when all nodes contain the minimum allowed number of entries, i.e., m. Therefore, it results that the maximum number of nodes in an R-tree is equal to: $\sum_{i=1}^{h_{max}} \lceil N/m^i \rceil = \lceil N/m \rceil + \lceil N/m^1 \rceil + \lceil N/m^2 \rceil + \ldots + 1$

Root	Internal nodes	Leaf nodes
min entries is 2 unless it is a leaf	min entries is m max entries is M	min entries is m max entries is M
	data to child nodes	data objects
	Hieght-balanced tree $h_{max} = \lceil \log_2 N \rceil - 1$ $NumNode_{max} = \sum_{i=1}^{h_{max}} \lceil N/m^i \rceil$	

Table 1: *R-tree characteristics*

3 Operations

3.1 Search

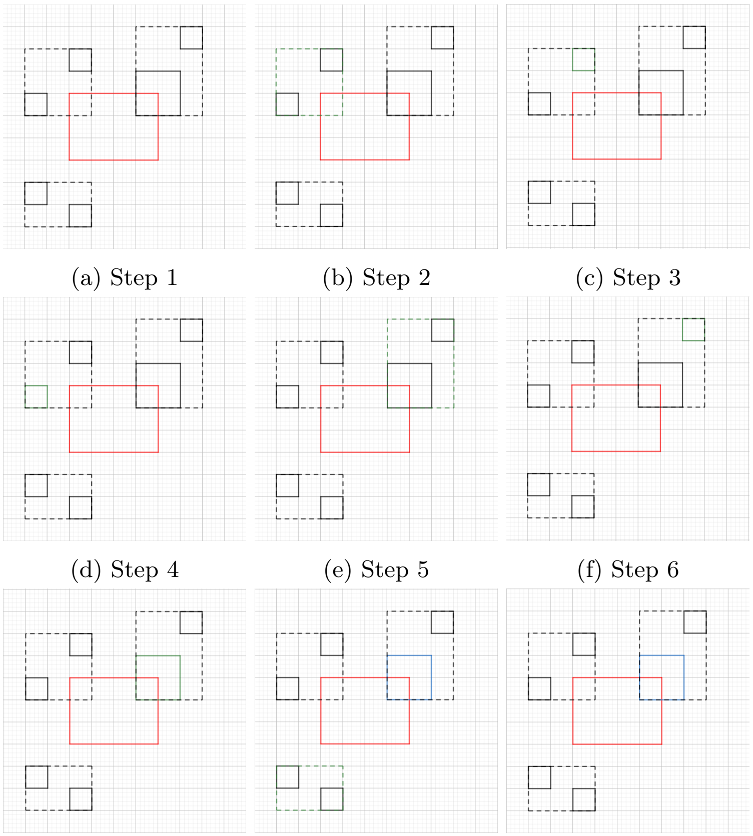
Problem: Find all data rectangles that are intersected by Q. This is denoted as a range (or window) query.
For a node entry E, E.mbr denotes the corresponding MBR, and E.p is the corresponding pointer to the next level. If the node is a leaf, then E.p denotes the corresponding object identifier (oid).

```
Algorithm RangeSearch (TypeNode RN, TypeRegion Q)

// Answer are stored in the set A

if RN is not a leaf node
    examine each entry e of RN to find those e.mbr that
        intersect Q
    foreach such entry e call RangeSeach(e.ptr, Q)
else // RN is a leaf node
    examine all entries e and find those for which e.mbr
        intersects Q
    add these entries to the answer set A
endif
```

Below is the example of the search operation. Red lines rectangle is the query rectangle, green lines rectangle is the MBR of the current node, and blue one is the answer.


(a) Step 1 (b) Step 2 (c) Step 3

(d) Step 4 (e) Step 5 (f) Step 6

(g) Step 7 (h) Step 8 (i) Step 9

Fig. 4: *Search operation*

3.2 Insertion

Problem: Insert a new data rectangle R into the R-tree.
Concept: The R-tree is traversed to locate an appropriate leaf to accommodate the new entry. The entry is inserted in the found leaf and, then all nodes within the path from the root to that leaf are updated accordingly. In case the found leaf cannot accommodate the new entry because it is full *(it already contains M entries)*, then it is split into two nodes.

```
Algorithm Insert(TypeEntry E, TypeNode RN)

Traverse the tree from root RN to the appropriate leaf. At
    each level, select the node, L, whose MBR will require
    the minimum area enlargement to cover E.mbr
In case of ties, select the nose whose MBR has the minimum
    area
if the selected leaf L can accommodate E
    Insert E into L
    Update all MBRs in the path from the root to L,
    so that all of them cover E.mbr
else // L is already full
    Add E into L, then split L into L1 and L2
    Update the MBRs of node that are in the path from root
        to L, so as to covers L1 and accommodate L2
    Perform splits as the upper levels if necessary
    In case the root has to be split, create new root
    Increase the height of the tree by one
endif
```

Split - goal:

- The leaf node has M entries, and one new entry to be inserted, how to partition the M+1 mbrs into two nodes, such that:

  - The total area of the two nodes is minimized
  - The overlapping of the two nodes is minimized

- Sometimes the two goals are conflicting: Using 1 as the primary goal

There are three alternatives to handle splits that were proposed by Guttman.

- **Linear split:** Choose two objects as seeds for the two nodes, where these objects are as far apart as possible. Then consider each remaining object in a random order and assign it to the node requiring the smallest enlargement of its respective MBR. For example with some 2-D objects:

MBRs	bottom left	top right
A	(1, 1)	(3, 4)
B	(2, 3)	(5, 6)
C	(6, 2)	(8, 5)
D	(7, 6)	(9, 9)
E (new)	(4, 4)	(6, 7)

  Use A and D as seeds

MBRs	MBR if added to A	MBR if added to D	Assigned to
B	small increase	large increase	group A
C	large increase	small increase	group D
E	small increase	large increase	group A

After the split

Group	MBRs
A	(1, 1), (4, 4), (6, 7)
D	(7, 6), (9, 9)

- **Quadratic split:** Choose two objects as seeds for the two nodes, where these objects if put together create as much dead space as possible *(dead space is the space that remains from the MBR if the areas of the two objects are ignored)*. Then, until there are no remaining objects, insert the object e such that e expands a group with the minimum area, *(if tie* ***Choose the group of small area*** *-* ***Choose the group of fewer elements)***.
  For example with some 2-D objects:

MBRs	bottom left	top right
A	(1, 1)	(3, 4)
B	(2, 3)	(5, 6)
C	(6, 2)	(8, 5)
D	(7, 6)	(9, 9)
E (new)	(4, 4)	(6, 7)

Calculate the dead space

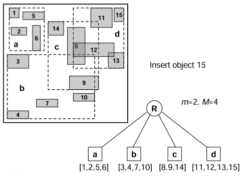
Pair	Resulting MBR	Area	intersection area	Wasted area
A, B	(1, 1), (5, 6)	20	1	14
A, C	(1, 1), (8, 5)	28	0	16
A, D	(1, 1), (9, 9)	64	0	52(worst)
A, E	(1, 1), (6, 7)	30	0	18
B, C	(2, 2), (8, 6)	24	0	9
B, D	(2, 3), (9, 9)	42	0	27
B, E	(2, 3), (6, 7)	16	2	3
C, D	(6, 2), (9, 9)	21	0	9
C, E	(4, 2), (8, 7)	20	0	8
D, E	(4, 6), (9, 9)	25	0	13

Use A and D as seeds

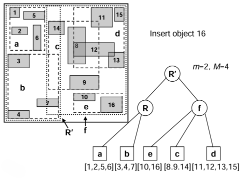
```
Firstly, choose B go to group A (extends the least)
// The mbr of group A: (1, 1), (5, 6)
Secondly choose E go to group A (extends the lease)
// The mbr of group A: (1, 1), (6, 7)
Finally, C will go to group D (following the
    constraint of minimum entries in each node)
// The mbr of group D: (6, 2), (9, 9)
```

- **Exponential split:** All possible groupings are exhaustively tested and the best is chosen with respect to the minimization of the MBR enlargement.

Illustrate for insert operation, in 2-d space:



(a) Step 1



(b) Step 2

Fig. 5: *Insert operation*

3.3 Deletion