The R-Tree

Yufei Tao

ITEE
University of Queensland

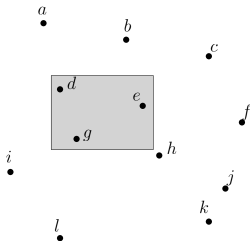
We will study a new structure called the R-tree, which can be thought of as a multi-dimensional extension of the B-tree. The R-tree supports efficiently a variety of queries (as we will find out later in the course), and is implemented in numerous database systems. Our discussion in this lecture will focus on orthogonal range reporting.

2D Orthogonal Range Reporting (Window Query)

Let S be a set of points in $\mathbb{R}^2$. Given an axis-parallel rectangle q, a *range query* returns all the points of S that are covered by q, namely, $S \cap q$.

The definition can be extended to any dimensionality in a straightforward manner.

Example



The result is $\{d, e, g\}$ for the shaded rectangle q.

Applications

- Find all restaurants in the Manhattan area.
- Find all professors whose ages are in $[20, 40]$ and their annual salaries are in $[200k, 300k]$.
- ...

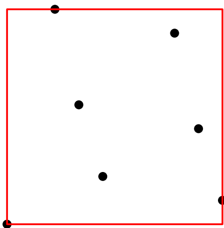
R-Tree

- Each leaf node has between $0.4B$ and B data points, where $B \geq 3$ is a parameter. The only exception applies when the leaf is the root, in which case it is allowed to have between 1 and B points. All the leaf nodes are at the same level.
- Each internal node has between $0.4B$ and B child nodes, except when the node is the root, in which case it needs to have at least 2 child nodes.

In practice, for a disk-resident R-tree, the value of B depends on the block size of the disk so that each node is stored in a block.

R-Tree

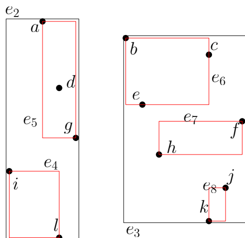
For any node u, denote by S_u the set of points in the subtree of u. Consider now u to be an internal node with child nodes $v_1, ..., v_f$ ($f \leq B$). For each v_i ($i \leq f$), u stores the minimum bounding rectangle (MBR) of S_{v_i}, denoted as $MBR(v_i)$.

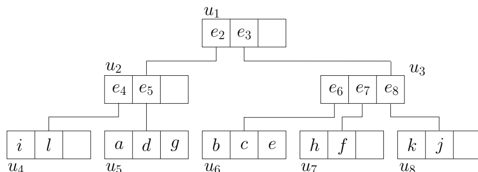


The above is an MBR on 7 points.

Example

Assume $B = 3$.





Answering a Range Query

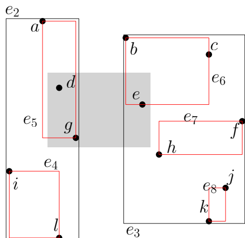
Let q be the search region of a range query. Below we give the pseudo-code of the query algorithm, which is invoked as `range-query`$(root, q)$, where *root* is the root of the tree.

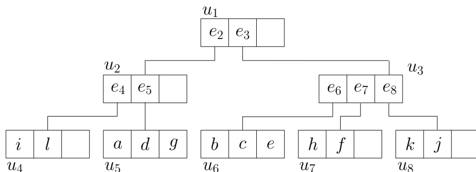
Algorithm `range-query`(u, r)

```
1. if u is a leaf then
2.     report all points stored at u that are covered by r
3. else
4.     for each child v of u do
5.         if MBR(v) intersects r then
6.             range-query(v, r)
```

Example

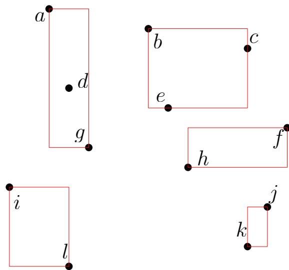
Nodes u_1, u_2, u_3, u_5, u_6 are accessed to answer the query with the shaded search region.





R-Tree Construction Can Be "Arbitrary"

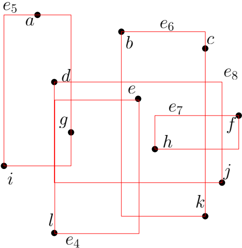
Have you wondered why the leaf nodes are created in this way? For example, is it absolutely necessary to group i and l into a leaf node?

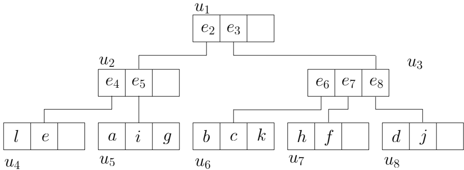


The R-tree definition has no formal constraint whatsoever on the grouping of data into nodes (unlike B-trees), but some R-trees have poorer performance than others; see the next slide.

R-Tree Construction Can Be "Arbitrary"

Is this a good R-tree?



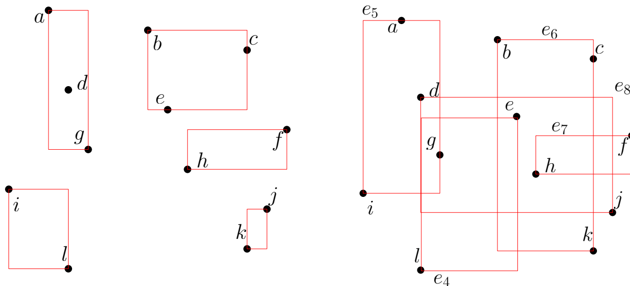


Implication?

R-Tree Construction: A Common Principle

In general, the construction algorithm of the R-tree aims at minimizing the perimeter sum of all the MBRs.

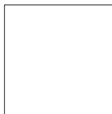
For example, the left tree has a smaller perimeter sum than the right one.



R-Tree Construction: A Common Principle

Why not minimize the area?

A rectangle with a smaller perimeter usually has a smaller area, but not the vice versa. Later in the course, we will see an analysis that formally validates this intuition.



The above two rectangles have the same area.

Insertion

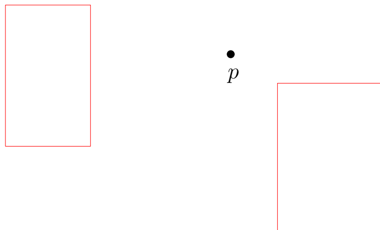
Let p be the point being inserted. The pseudo-code below should is invoked as `insert`$(root, p)$, where *root* is the root of the tree.

Algorithm `insert`(u, p)

```
1.  if u is a leaf node then
2.      add p to u
3.      if u overflows then
        /* namely, u has B + 1 points */
4.          handle-overflow(u)
5.  else
6.      v ← choose-subtree(u, p)
        /* which subtree under u should we insert p into? */
7.      insert(v, p)
```

Choose-Subtree

Which MBR would you insert p into?



Algorithm `choose-subtree`(u, p)

1. return the child whose MBR requires the minimum increase in perimeter to cover p. break ties by favoring the smallest MBR.

Overflow Handling

Algorithm `handle-overflow`(u)

1. `split`(u) into u and u'
2. **if** u is the root **then**
3. create a new root with u and u' as its child nodes
4. **else**
5. $w \leftarrow$ the parent of u
6. update $MBR(u)$ in w
7. add u' as a child of w
8. **if** w overflows **then**
9. `handle-overflow`(w)

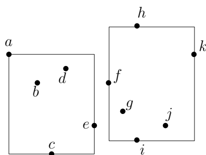
Splitting a Leaf

Essentially we are dealing with the following problem:

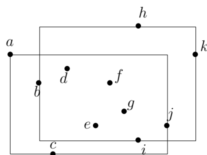
Let S be a set of $B+1$ points. Divide S into two disjoint sets S_1 and S_2 to minimize the perimeter sum of $MBR(S_1)$ and $MBR(S_2)$, subject to the condition that $|S_1| \geq 0.4B$ and $|S_2| \geq 0.4B$.

Example

The left split is better:



$$S_1 = \{a, b, c, d, e\}$$
$$S_2 = \{f, g, h, i, j, k\}$$



$$S_1 = \{a, d, e, g, j\}$$
$$S_2 = \{b, c, f, h, i, k\}$$

Splitting a Leaf Node

Let $m = |S|$. In 2D space, the leaf-split problem can be solved in $O(m^5)$ time, noticing that each MBR is determined by 4 points.

This, however, is too expensive. In practice, heuristics are used to accelerate the process, but there is no guarantee that we can find the best split — typical "trading quality for efficiency".

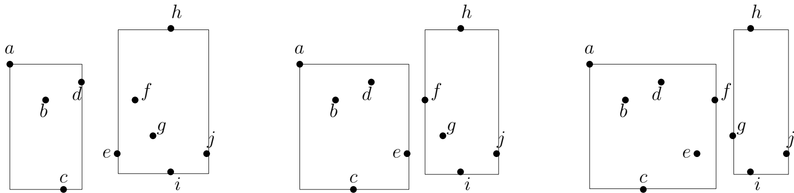
The next slide explains how.

Splitting a Leaf Node

Algorithm `split`(u)

1. $m =$ the number of points in u
2. sort the points of u on x-dimension
3. **for** $i = \lceil 0.4B \rceil$ to $m - \lceil 0.4B \rceil$
4. $\quad$ $S_1 \leftarrow$ the set of the first i points in the list
5. $\quad$ $S_2 \leftarrow$ the set of the other i points in the list
6. $\quad$ calculate the perimeter sum of $MBR(S_1)$ and $MBR(S_2)$; record it if this is the best split so far
7. Repeat Lines 2-6 with respect to y-dimension
8. **return** the best split found

Example



There are 3 possible splits along the x-dimension. Remember that each node must have at least $0.4B = 4$ points (here $B = 10$).

Think:

- How to implement the algorithm in $O(n \log n)$ time?
- Find a counter-example where the algorithm does not give an optimal split.
- We have discussed only the 2D case. How to extend the algorithm to dimensionality $d \geq 3$?

Splitting an Internal Node

Let S be a set of $B+1$ rectangles. Divide S into two disjoint sets S_1 and S_2 to minimize the perimeter sum of $MBR(S_1)$ and $MBR(S_2)$, subject to the condition that $|S_1| \geq 0.4B$ and $|S_2| \geq 0.4B$.

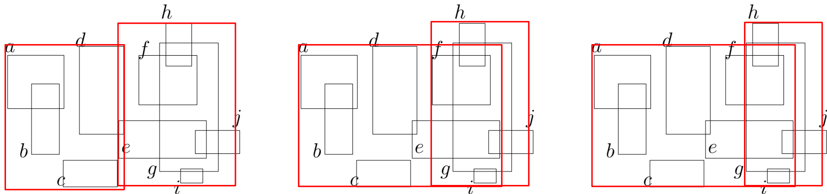
Once again, we will settle for an algorithm that is fast but does not always return an optimal split.

Splitting an Internal Node

Algorithm `split`(u)
/* u is an internal node */

1. m = the number of points in u
2. sort the rectangles in u by their left boundaries on the x-dimension
3. **for** $i = \lceil 0.4B \rceil$ to $m - \lceil 0.4B \rceil$
4. $\quad S_1 \leftarrow$ the set of the first i rectangles in the list
5. $\quad S_2 \leftarrow$ the set of the other i rectangles in the list
6. $\quad$ calculate the perimeter sum of $MBR(S_1)$ and $MBR(S_2)$; record it if this is the best split so far
7. Repeat Lines 2-6 with respect to the right boundaries on the x-dimension
8. Repeat Lines 2-7 w.r.t. the y-dimension
9. **return** the best split found

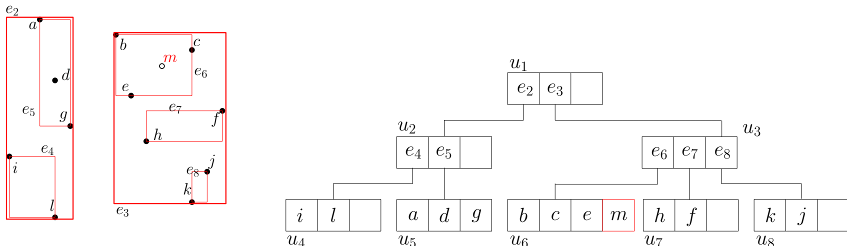
Example



There are 3 possible splits w.r.t. the left boundaries on the x-dimension. Remember that each node must have at least $0.4B = 4$ points (here $B = 10$).

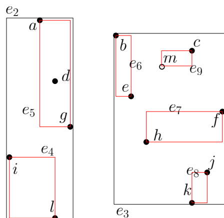
Insertion Example

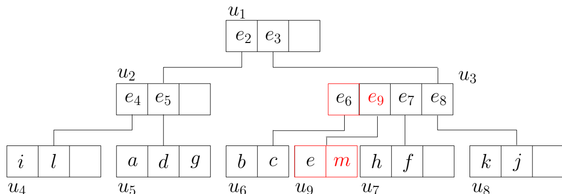
Assume that we want to insert the white point m. By applying `choose-subtree` twice, we reach the leaf node u_6 that should accommodate m. The node overflows after incorporating m (recall $B = 3$).



Insertion Example

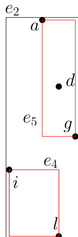
Node u_6 splits, generating u_9. Adding u_9 as a child of u_3 causes u_3 to overflow.

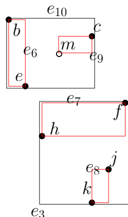


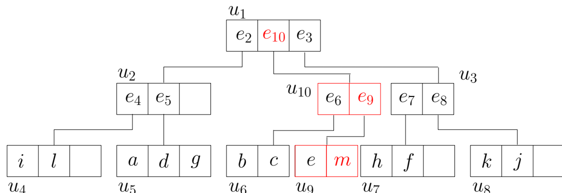


Insertion Example

Node u_3 splits, generating u_{10}. The insertion finishes after adding u_{10} as a child of the root.







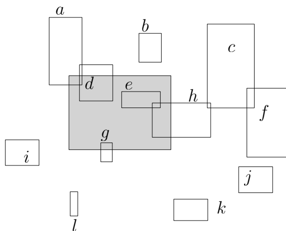
Although not required in this course, there are fast algorithms to perform deletions from an R-tree. The instructor will discuss this in the class if time permits.

2D Orthogonal Range Reporting (Window Query) on Rectangles

Let S be a set of axis-parallel rectangles in $\mathbb{R}^2$. Given an axis-parallel rectangle r, a range query returns all the rectangles of S that are covered by r, namely, $S \cap r$.

The definition can be extended to any dimensionality in a straightforward manner.

Example



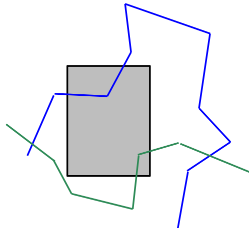
The result is $\{a, d, g, e, h\}$ for the shaded rectangle q.

An R-Tree on Rectangles

Same as an R-tree on points with two changes:

- Leaf nodes store data rectangles (as opposed to points).
- Splitting a leaf node is performed using the internal-split algorithm mentioned earlier.

An R-Tree on Line Segments—Motivation



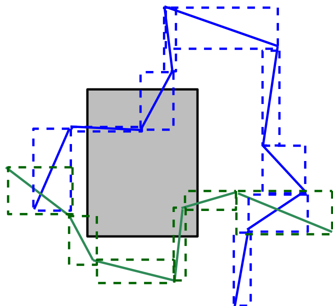
Consider a geographic information system (GIS) that manages road segments. The above figure shows an example where the blue segments represent a road, whereas the green ones represent another. Suppose that we want to retrieve all the road segments in the shaded window (think of the window as the user's browser, for instance). How to adapt the R-tree to support this type of retrieval?

Filter Refinement

A common solution implemented in commercial systems is based on the filter refinement framework. In the filter step, we quickly retrieve a candidate set of segments that is guaranteed to contain the final result. The refinement step then examines every segment in the candidate set to remove the "false hits".

A predominant approach do realize the filter step is to create an MBR on every segment, and use an R-tree to find the MBRs that intersect the query. The segments corresponding to those MBRs constitute the candidate set.

Filter Refinement



The dashed rectangles are the MBRs on the segments. What are the segments retrieved by the filter step? Are there any false hits?