Vo Tran Thanh Luong
1551020

R-TREE

I. About R-Tree as a data structure :

To deeply understand how R-tree works as a data structure, we first need to understand the term “Spatial access method”.

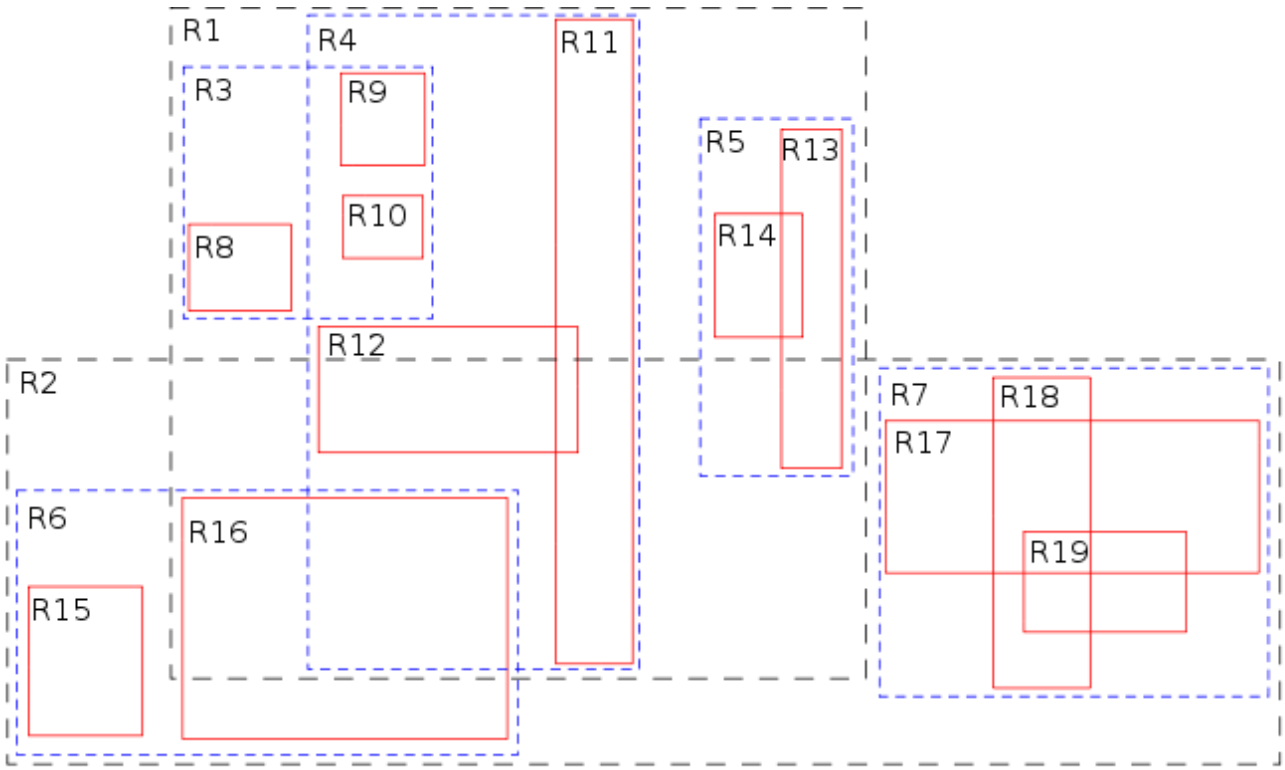
Spatial data objects often cover areas in multi-dimensional spaces and are not well represented by point locations For example, map objects like counties, census tracts, etc. occupy regions of non-zero size in two dimensions. A common operation on spatial data is a search for all objects in an area, for example to find all counties that have land within 20 miles of a particular point. This kind of spatial search occurs frequently in computer aided design (CAD) and geo-data applications, and therefore it is important to be able to retrieve objects efficiently according to their spatial location.

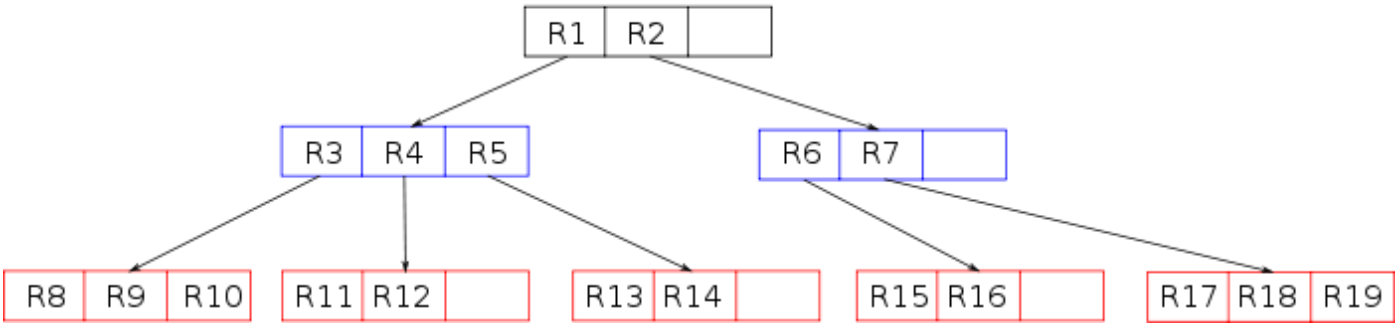
Spatial indices are used by spatial databases (databases which store information related to objects in space) to optimize spatial queries. Conventional index types do not efficiently handle spatial queries such as how far two points differ, or whether points fall within a spatial area of interest. One common spatial index method is R-tree.

R-trees are tree data structures used for spatial access methods,for indexing multi-dimensional information such as geographical coordinates, rectangles or polygons. The R-tree was proposed by Antonin Guttman in 1984 and has found significant use in both theoretical and applied contexts.

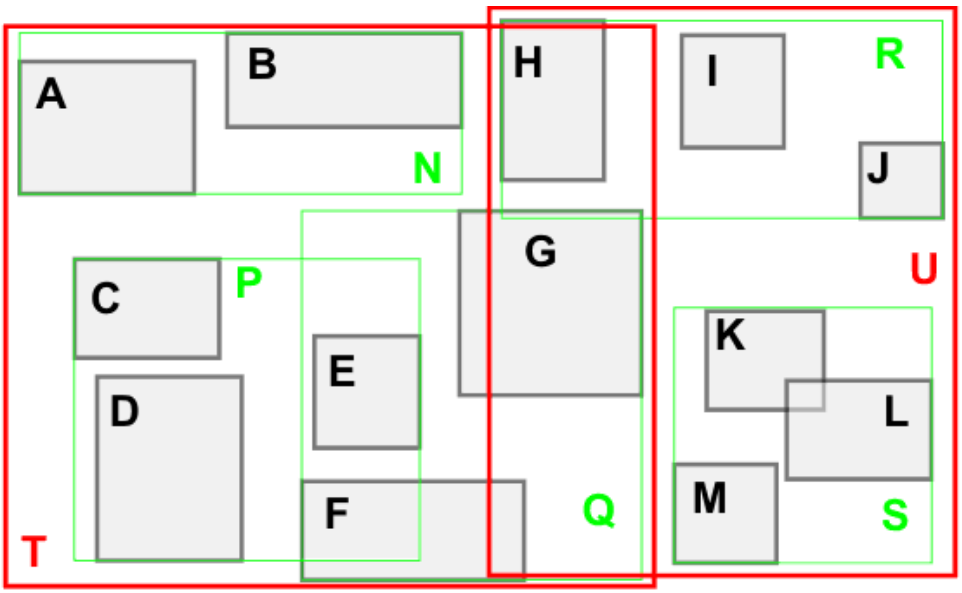
The key idea of the data structure is to group nearby objects and represent them with their minimum bounding rectangle in the next higher level of the tree; the "R" in R-tree is for rectangle. Since all objects lie within this bounding rectangle, a query that does not intersect the bounding rectangle also cannot intersect any of the contained objects. At the leaf level, each rectangle describes a single object; at higher levels the aggregation of an increasing number of objects. This can also be seen as an increasingly coarse approximation of the data set.

Vo Tran Thanh Luong
1551020

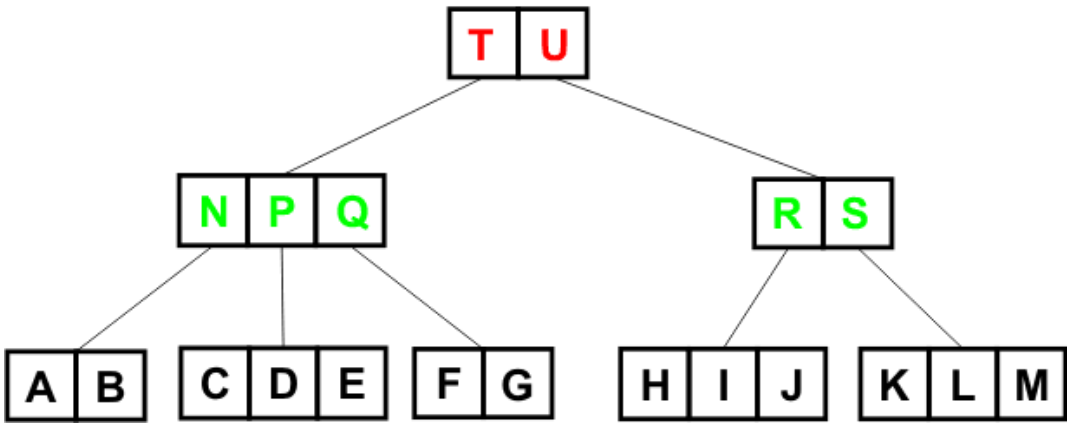




Vo Tran Thanh Luong
1551020



This is a R-tree. One thing to notice is that it is based on B-tree, but invented because B-tree cannot store some special data types.



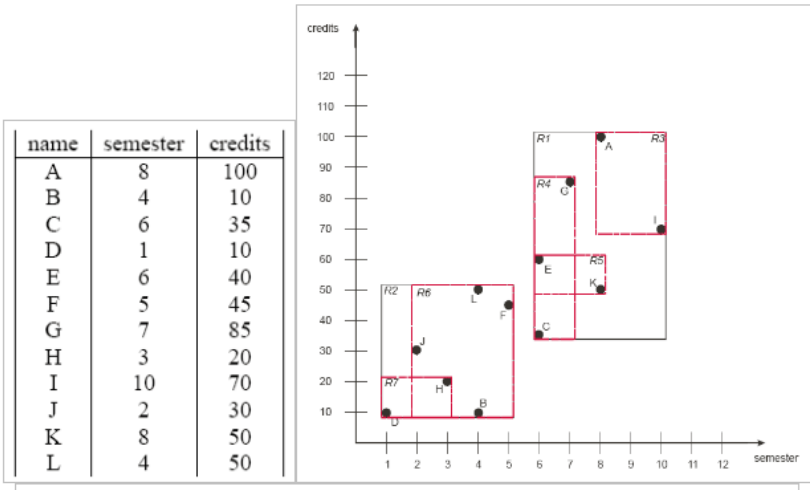
R-Trees can organize any-dimensional data by representing the data by a minimum bounding box.

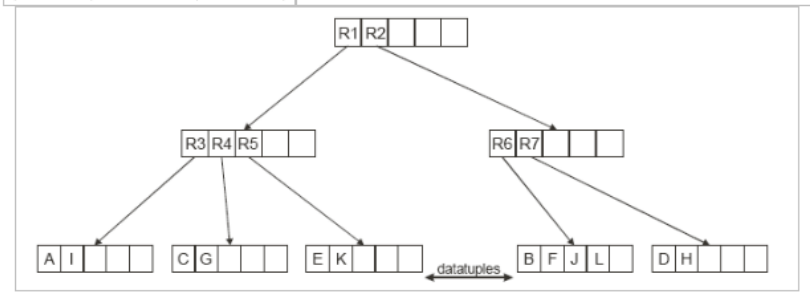
Each node bounds its children. A node can have many objects in it. The leaves point to the actual objects (stored on disk probably).The height is always log n (it is height balanced).

Similar to B-tree, It also guarantees a minimum fill (except for the root node), however best performance has been experienced with a minimum fill of 30%–40% of the maximum number of entries (B-trees guarantee 50% page fill, and B-

trees even 66%). The reason for this is the more complex balancing required for spatial data as opposed to linear data stored in B-trees.

name	semester	credits
A	8	100
B	4	10
C	6	35
D	1	10
E	6	40
F	5	45
G	7	85
H	3	20
I	10	70
J	2	30
K	8	50
L	4	50



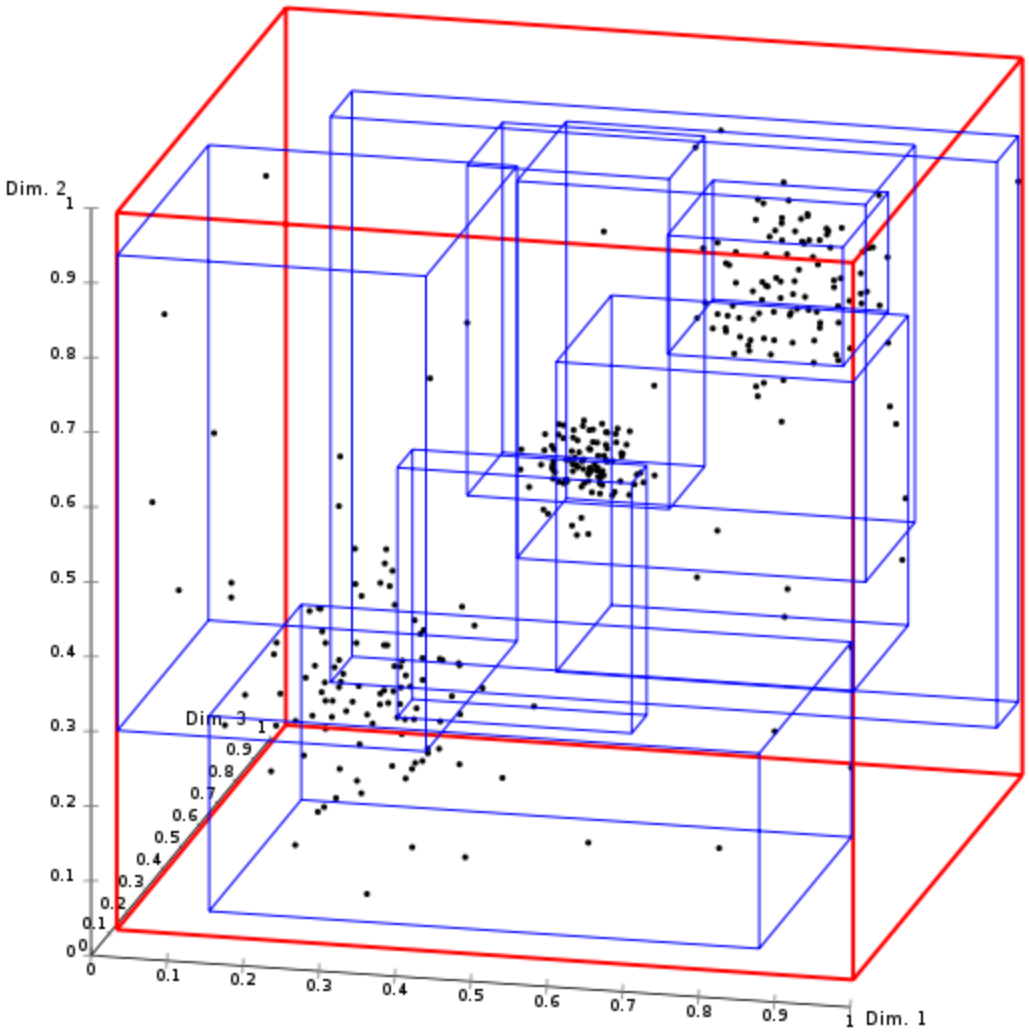


The key difficulty of R-trees is to build an efficient tree that on one hand is balanced (so the leaf nodes are at the same height) on the other hand the rectangles do not cover too much empty space and do not overlap too much (so that during search, fewer subtrees need to be processed). For example, the original idea for inserting elements to obtain an efficient tree is to always insert into the subtree that requires least enlargement of its bounding box. Once that page is full, the data is split into two sets that should cover the minimal area each. Most of the research and improvements for R-trees aims at improving the way the tree is built and can be grouped into two objectives: building an efficient tree from scratch (known as bulk-loading) and performing changes on an existing tree (insertion and deletion).

R-trees do not guarantee good worst-case performance, but generally perform well with real-world data. While more of theoretical interest, the (bulk-loaded) Priority R-tree variant of the R-tree is worst-case optimal, but due to the increased complexity, has not received much attention in practical applications so far.

Some variants of R-Tree are : R* tree, R+ tree, Hilbert R-tree, X-tree.

Vo Tran Thanh Luong
1551020



3D visualization of R –tree

II. Basic operations on R-tree :

- Searching: look at all nodes that intersect, then recursively go into those nodes. Many paths may lead nowhere. The search starts from the root node of the tree. Every internal node contains a set of rectangles and pointers to the corresponding child node and every leaf node contains the rectangles of spatial objects (the pointer to some spatial object can be

there). For every rectangle in a node, it has to be decided if it overlaps the search rectangle or not. If yes, the corresponding child node has to be searched also. Searching is done like this in a recursive manner until all overlapping nodes have been traversed. When a leaf node is reached, the contained bounding boxes (rectangles) are tested against the search rectangle and their objects (if there are any) are put into the result set if they lie within the search rectangle.

- Insertion: Locate place to insert node through searching and insert. If a node is full, then a split needs to be done. To insert an object, the tree is traversed recursively from the root node. At each step, all rectangles in the current directory node are examined, and a candidate is chosen using a heuristic such as choosing the rectangle which requires least enlargement. The search then descends into this page, until reaching a leaf node. If the leaf node is full, it must be split before the insertion is made. Again, since an exhaustive search is too expensive, a heuristic is employed to split the node into two. Adding the newly created node to the previous level, this level can again overflow, and these overflows can propagate up to the root node; when this node also overflows, a new root node is created and the tree has increased in height.
- There are two ways to split an overflowing node. Linear – choose far apart nodes as ends. Randomly choose nodes and assign them so that they require the smallest enlargement. Quadratic – choose two nodes so the dead space between them is maximized. Insert nodes so area enlargement is minimized.
- Deletion: Deleting an entry from a page may require updating the bounding rectangles of parent pages. If a node becomes underfull. Reinsert other nodes to maintain balance.

III. Real World Applications :

A common real-world usage for an R-tree might be to store spatial objects such as restaurant locations or the polygons that typical maps are made of: streets,

buildings, outlines of lakes, coastlines, etc. and then find answers quickly to queries such as "Find all museums within 2 km of my current location", "retrieve all road segments within 2 km of my location" (to display them in a navigation system) or "find the nearest gas station" (although not taking roads into account). The R-tree can also accelerate nearest neighbor search for various distance metrics, including great-circle distance.