Spatial Informatics

Prof. Soumya K. Ghosh
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

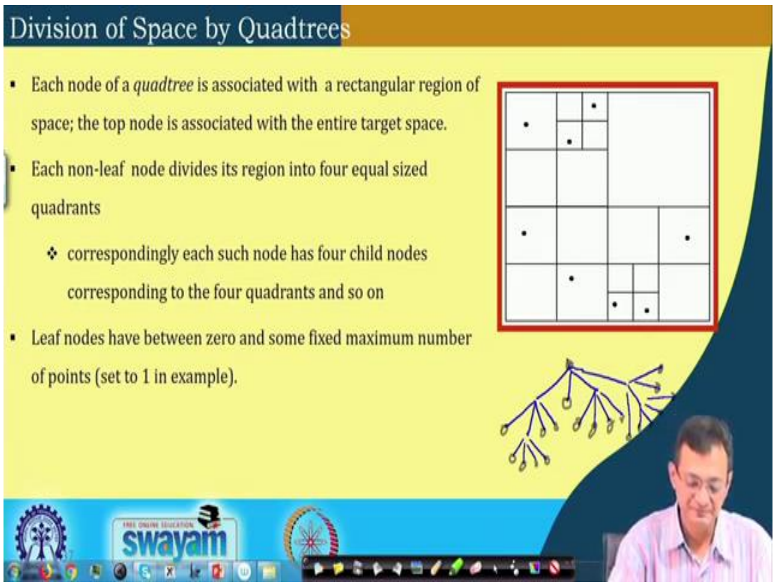
Lecture - 19
Spatial Indexing - III

Hello. So, we will be discussing on or continuing our discussion on Spatial Indexing. In last one or two lectures we discussed about different aspects of or need of spatial indexing and seen one of the one major aspect of space filling curves. The idea was that these spatial queries so, there is a high chance of looking for nearby objects right. So, that space filling curves may be a useful when you have this sort of queries, where I need to search the nearby objects and type of things right.

There are we have seen also a just a overview of k-d tree type of things which is which can be used. Today also we will see or we will discuss about some few more techniques, one of them is quadtree and of course, there is a tree family of structures right. So, which else are seen handling these spatial queries efficiently. So, just to quick a recap, why we are doing indexing? So, that my queries can be efficiently processed, right.

So, it can be efficiently processed and also I can have what you say accurate result in the things right. So, primarily that without compromising the outcome of the things I need to process these things that is why the indexing structure helps me in doing that. And, also if you look at our conventional or traditional data base systems also the indexing plays the important role in any DBMS system right.

(Refer Slide Time: 02:13)



Division of Space by Quadtrees

- Each node of a *quadtree* is associated with a rectangular region of space; the top node is associated with the entire target space.
- Each non-leaf node divides its region into four equal sized quadrants
    - correspondingly each such node has four child nodes corresponding to the four quadrants and so on
- Leaf nodes have between zero and some fixed maximum number of points (set to 1 in example).

So, one aspect of this is that quadtree right. So, the as the name suggest so it divides the space into 4 quadrant right and it go on dividing those things unless it is gets a homogenous region right. So, that is one aspect the thing, one in case it makes the tree out of this. So, if this is my whole region of interest or I should say root of the tree, then divide this into 4 right and then our 4 quadrants are there. So, I can name them something say 1 or 0 1 2 3 or 0 1 2 3 the way we can do it.

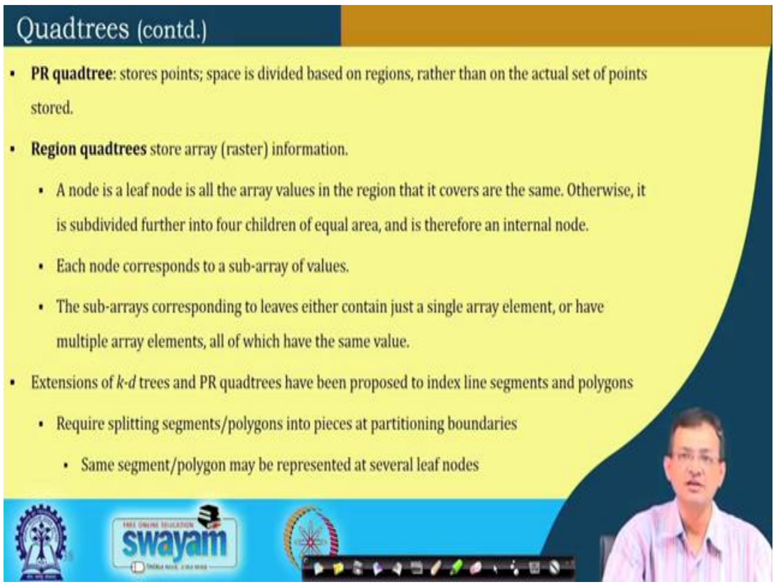
So, each node of the quadtree is associated with a rectangular region right, space the top node is associated with the entire target space hm. A each non-leaf node divides this its region into 4 equal sized quadrants right. Corresponding to each such node has 4 child nodes corresponding to the 4 quadrants and so on right. So that means, you see every non-leaf node has a this we divide into 4 it has 4 child nodes right. So, corresponding to each node as a 4 child nodes corresponding to the 4 quadrants and so on leaf node have between zero and some defined maximum number of points set to 1 for example, right.

So, see if I have these points right; so, in this space if I have this point I divide them into 4, right this is a homogenous things right. There is no points for that matter. So, there the tree stops, but here it is if you if we look at it is not a homogenous; so, I divide it again 4 right. So, this 3 becomes homogenous, there is no further division required. This becomes a space which needs to be again divided in 4 and so and so forth. Similarly, true for these things right, so, it forms a quad tree. We will show you an example. We will in subsequent slide we will show you the example. Otherwise if we see; so, it is something like this right initially 4, then suppose I consider 1 0 1 2 3. So, this is again divided into 4 right.

This is this will stop right. So, if we consider this one this is again divided into 4 right and then here all the 4 there is no further division is there. So, stop condition and this is again this one this is again divided into 4 right, out of that this I have this is no division, no division, no division, this is again divided into 4 right. So, it forms a tree type of structure and here also if you see this one is divided into 4 based on your sequence and so. So, now I end up we end up with these are the things right ok.

So, if it is like this and then I can basically encode it like this right. We will show you that how things are there, but nevertheless it forms a tree, now the searching will be on the best of the things right. There are some of issues we will discuss, that first of all a if it is a these are points right; if you have a polygon that may be in more than one that polygon is divided into more than one leaf nodes. So, you need to retrieve things, but is a efficient way of indexing and storing the things and retrieving mechanism may be efficient than our other generic methods right.

(Refer Slide Time: 06:39)



Quadtrees (contd.)

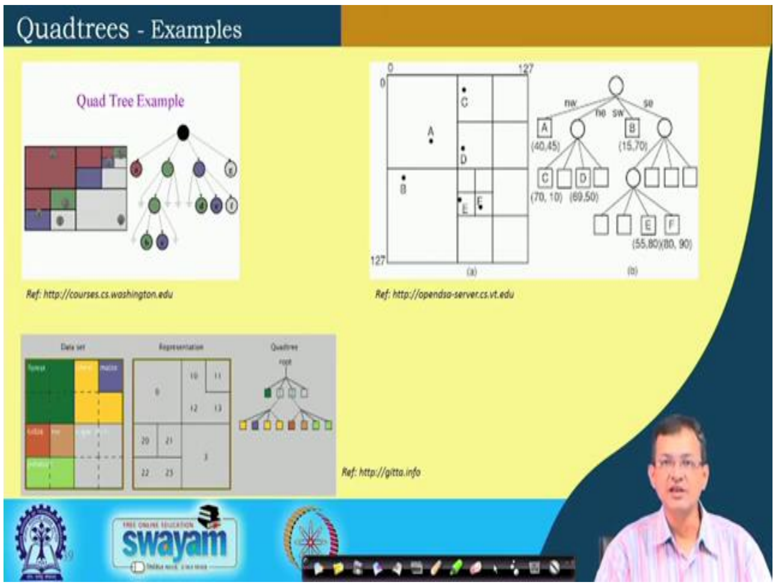
- **PR quadtree**: stores points; space is divided based on regions, rather than on the actual set of points stored.
- **Region quadtrees** store array (raster) information.
  - A node is a leaf node is all the array values in the region that it covers are the same. Otherwise, it is subdivided further into four children of equal area, and is therefore an internal node.
  - Each node corresponds to a sub-array of values.
  - The sub-arrays corresponding to leaves either contain just a single array element, or have multiple array elements, all of which have the same value.
- Extensions of *k-d* trees and PR quadtrees have been proposed to index line segments and polygons
  - Require splitting segments/polygons into pieces at partitioning boundaries
    - Same segment/polygon may be represented at several leaf nodes

So, there are two typical variation one is PR quadtree right: stores points, space is divided into regions rather than one actual set of points, store array raster informations right. A node is a leaf node is all array values in the region that over covers the same otherwise it is divided further into 4 children of equal area and therefore, internal node right fine; so, each node corresponding to the sub array of the values right. So, if I go to each node it goes to the sub array of the values.

Sub-array corresponding to leaf either contain such single array element or have multiple elements of all which have the same values right. So, either it is on the single value or I can have multiple value elements of the things; that means, my region may be divided in such a way that, it does not come in a one quadrant, but number of quadrants which are which comes into the leaf node right. So, it is extension of k-d tree and PR quadtree have been proposed indexing segment and polygon.

We can that k-d tree or PR quadtree can be extended to for indexing line segments and polygons that is as you know that line segment is multiple points in a things. And, the polygon is again a polyline where the starting point the ending point meets right the same as the starting point and ending point. So, same segment polygon may be represented in several leafs and nodes I can represent that thing in a several leafs and node, the same approach I can follow the things right because finally, it is points to polyline to polygon type of structure ok.

(Refer Slide Time: 08:33)


Quadtrees - Examples
Quad Tree Example
Ref: http://courses.cs.washington.edu
Ref: http://opendsa-server.cs.vt.edu
Ref: http://gitta.info

So, this is the basic philosophy, we will see some example to make the things little more clear. So, if we look at this structure right; so, there are a this is the picture, this figure may not be very clear a something b c d etcetera. Now, we first divide into 4 right. So, what we found? This is a contiguous space right that is a homogenous space. So, this becomes one of the things where it is the this particular a comes into the thing otherwise I need to this one there is again if I consider only this quadrant, then it is not homogenous.

So, I divide it again into 4 right and then this portion and this portion, this portion becomes homogenous. So, I divide this was again into 4 and this becomes homogenous. So, this is again partitioned into 4 right out of them 1 2 3 are so, called grounded; that means, they are same space where as 1, this one this quadrant as to be again divided into 4 ok. So, out of that I get this two values as so, one is a value b c values way back the it connects to the data values, others as nothing to connect or grounded or null. Similarly, if you see this quadrant also divided into 4 and d e and etcetera.

Now, for searching any of these values a b c d or something I can follow this tree. So, this is this quadtree index and then go to this value and then I can find out where the things are there right. This is these sounds so, this is fine. So, it is if you look at the same thing that the example what we are looking at the PR quadtree, these are these are the points. And, then also we can basically have this different type of things and every point has a coordinate like this 45, 40, 45 or some coordinates systems where, 0 is the upper left hand corner and we have this type of coordinate systems. So, we have this structure.

Now, searching again some point will be following this quadtree will be makes much quicker or efficient rather than searching on a doing a raster scan on the data old data in the database right. So, this is another way of looking at the thing. So, rather if we look at this sort of structure like we have some polygon so called or raster space right. Raster images like this is a forest patch so, it is a synthetically done. So, it is a square, this is a another land use pattern, this another land use pattern of maze and like this.

So now, I again divide them into like this is 0 1 2 3 is my quadrant right. So, 0 is like this, 1 is like this right, 2 is like this, 3 is like this right. So, I plan it in the in this fashion right. So, again this 1 see it is not homogenous I divide it again into 4. So, this quadrant is again into the 4 out of that we get these 3 as homogenous right, but or this 3 quadrant homogenous and this is also homogenous. So, this stops here; so, this is yellow, yellow, yellow and this another is the maze for the other thing correct right.

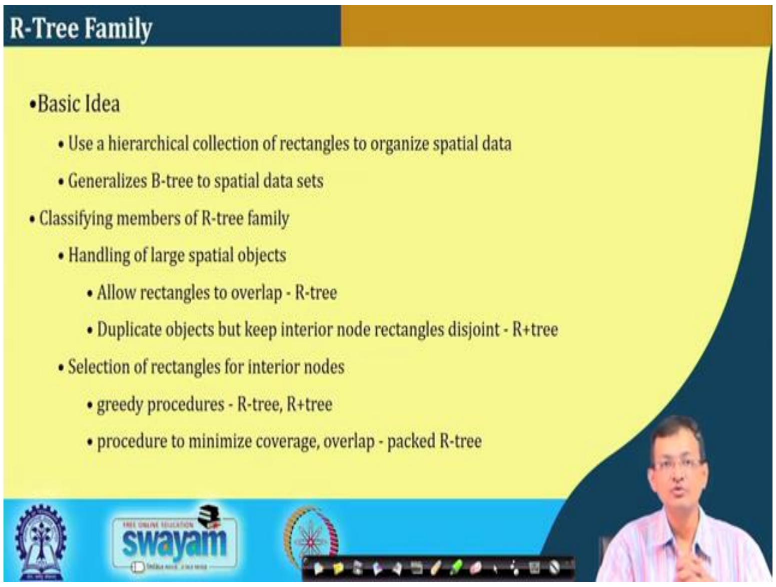
Similarly, if this one also is divided into 4 and then we get this right and this third is basically stops here only because, this is a homogenous portion right. So, here also we get. So, what we see here? Now, you see suppose you want to find out this particular patch of see this is a yellow is wheat, writing is this is not much visible, but same fit or any class right and this green is some other class. Now, you see in order to access these regions for this wheat regions what we need to do? We need to access 3 nodes right, it is not 1 node right, we in the quadtree 1 2 3 nodes right.

So, it is again increasing the complexity right, I know that number of nodes you want to access I will be more that for this sort of scenarios like especially this sort of polygons scenarios and type of things. Nevertheless, it is it may be much faster than accessing in a raster scan by line by line or row by row things from the disc, rather this storage also will be very compact. I can store this things in the near locality in the disc space. So, the access rate disc access rate or the IO access will be much faster right.

So, this helps us, this indexing helps us in accessing data in faster as we know that I do IO cost is one of the prevalent cost, though in both in case of spatial and non-spatial data sets right. Though the spatial datasets also additionally have a this is your computational cost is also pretty high. But, nevertheless I can reduce this overall costing and the query can be efficiently executed right. So, this is one aspect which is can be done using this quadtree structure. These are taken from the internet then we have given the reference. So, can see some of the things here right.

Now, given this quadtree so, what we have seen this quadtree is one of the efficient way of the storing, specially it is used in a very efficient way in case of as raster images or like here where we have different structure raster structure like this. But, it can be very well used in case of a point whether we have a PR tree, PR quad type of things as we have seen that this point PR quadtree can be extended to polyline or sort of a polygon also.

(Refer Slide Time: 15:25)



R-Tree Family

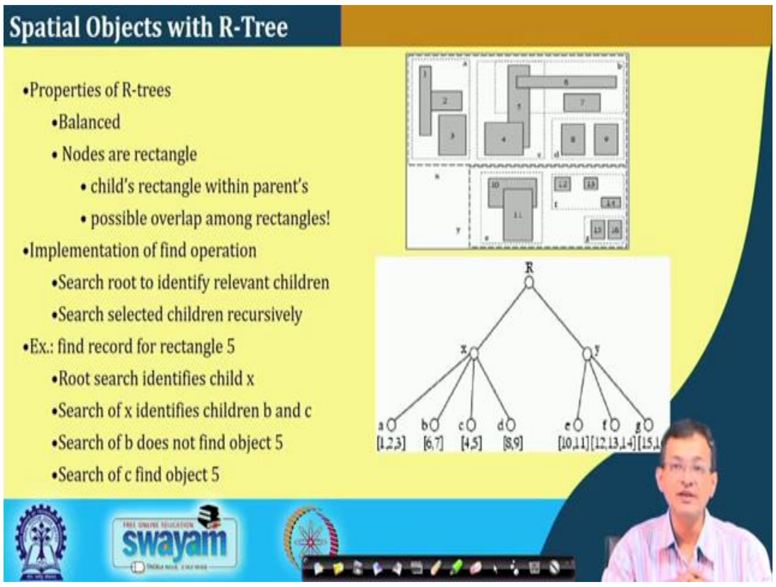
- Basic Idea
  - Use a hierarchical collection of rectangles to organize spatial data
  - Generalizes B-tree to spatial data sets
- Classifying members of R-tree family
  - Handling of large spatial objects
    - Allow rectangles to overlap - R-tree
    - Duplicate objects but keep interior node rectangles disjoint - R+tree
  - Selection of rectangles for interior nodes
    - greedy procedures - R-tree, R+tree
    - procedure to minimize coverage, overlap - packed R-tree

Now, we come to another major indexing scheme specially for spatial datasets is that R tree or family of R tree. So, R tree what we say R tree family right. So, basic idea or fundamental idea is that use a hierarchical collection of the rectangles to organize the spatial datasets, generalize the B-tree to spatial data sets, classify. So, this is what is the hierarchical structure of rectangle to organize spatial dataset. So, I will see show you that how things works, generalize B-tree to spatial data set.

So, it is more you can you can together generalizing the B-tree structure what you are already used to or what you already know to a spatial data set. Classifying members of the R-tree so, there are different members of R-tree family. So, one is handling large spatial objects, allow rectangle to overlap this is fundamentally this R-tree basic that basic R-tree says. Duplicate objects, but keep interior node rectangle disjoint right. So, you can have duplicate objects, but the interior node which is a R plus tree another variant of R-tree right.

Now, now section of rectangle for the interior node. So, we can have a selection of rectangle for the interior node. So, I can which node to select right, it can be falling in more than one type of things. So, we can use a greedy approach for R-tree and R plus tree we will see that, procedure to minimize coverage overlap that is the packed R-tree. So, there is a there is another concept of packed R-tree where procedure to minimize the coverage overlap.

(Refer Slide Time: 17:15)



Spatial Objects with R-Tree

- Properties of R-trees
  - Balanced
  - Nodes are rectangle
    - child's rectangle within parent's
    - possible overlap among rectangles!
- Implementation of find operation
  - Search root to identify relevant children
  - Search selected children recursively
- Ex.: find record for rectangle 5
  - Root search identifies child x
  - Search of x identifies children b and c
  - Search of b does not find object 5
  - Search of c find object 5

So, properties of R-trees. So, it is balanced, nodes are rectangle; rectangles are child rectangle within the parents rectangle, possible overlap among the rectangles ok. So, it is balanced, this typically right so, it is a balanced tree. Nodes are considered as rectangle right, child rectangle within the parents rectangle. So, the if a child rectangle so, it is this parent rectangle is a super set of this child rectangle. Possible overlap among rectangles are possible right which is contradicting with our traditional B-tree structure and, other type of structure where we do not allow this type of overlapping right.

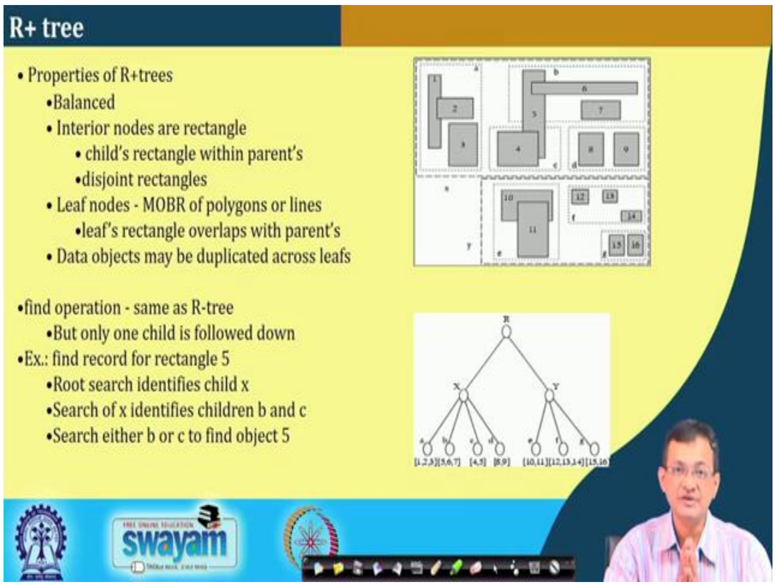
Implementation of say find operation, search root to identify relevant children right, search selected children recursively. So, first we go to the root and then select the relevant children and this. So, here if you see this 4 5 and 6 are overlapping right, 4 and 5 are overlapping right 5. So, if we have the whole tree as the R root then we have 2 rectangle x and y which are non-overlapping, out of that we have a, then b, then c, d where b and c are over lapping; intermediate things are over lapping right and this 6 belongs to both to both to c and b. So, you need to have some logic or use stick to say where I want to put this c or this 5 also belong both to b and c right and 4 and 5 is again over lapping.

In this case 10 and 11 are over lapping; however, they both belong to e and here also they are not over lapping. So, the structure is R then x and y, R we have x and y out of x and y we have in the x we have a and a contains 1 2 3, these are the data nodes. So, the connection to the data pointers, b connects to 6 and 7, c connects to 4 and 5 right. So, we consider this 5 is within c not under b so, that is the that is what is logic or heuristic what we follow, what is being followed; d e contains 8 and 9 and y contain e f g e f g. And, then e is having 10, 11, 13 12, 13 and 15, 16 so, this is the way it is constructed right.

Now, find record for rectangle 5 right, if you want to find the record for rectangle 5 we want to my I am looking for this 5 the data for the 5. So, root search identifies child x right, say that a five will be in the x right. It is a this m embiar of this x is encompass there right, search of a x identifies children b and c right and b does not find 5 so, c finds object 5. So, there may be multiple search path nevertheless the number of things is accessed is much less than the overall things right. So, like here you see around 50 percent of the things are within cutout of the them is has been proved out of the search space and type of things right.

So, this is the basic R-tree right where we are allowing intermediate things to there is overlapping. But whereas, in the in the leaf node so, this is everything is a in one place only right, is one place only.

(Refer Slide Time: 21:43)



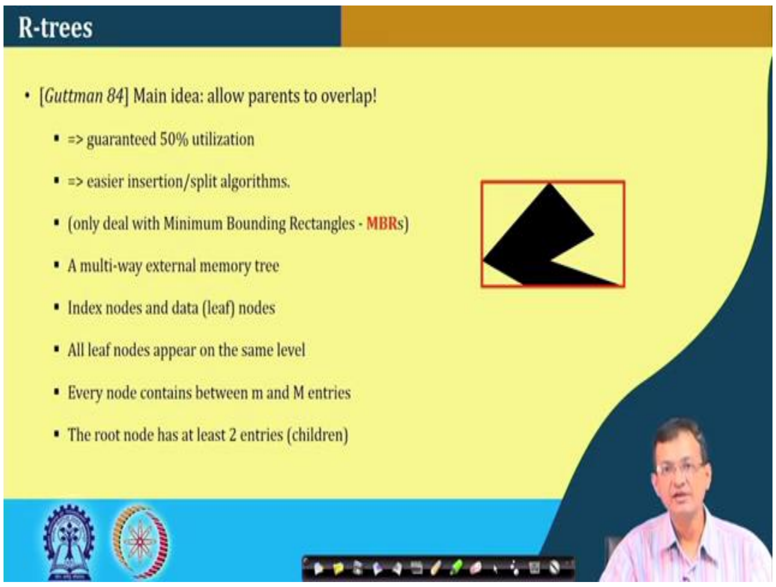
R+ tree

- Properties of R+trees
  - Balanced
  - Interior nodes are rectangle
    - child's rectangle within parent's
    - disjoint rectangles
  - Leaf nodes - MOBR of polygons or lines
    - leaf's rectangle overlaps with parent's
  - Data objects may be duplicated across leafs
- find operation - same as R-tree
  - But only one child is followed down
- Ex.: find record for rectangle 5
  - Root search identifies child x
  - Search of x identifies children b and c
  - Search either b or c to find object 5

Now, a variant of R-tree is the R plus tree so, it is balanced, interior nodes are rectangle, child rectangles within the parent rectangle so far good; this is disjoint rectangles right. So, these rectangles are disjoint rectangle, leaf node MOBR of the polygon or lines right. So, leafs nodes are the MOBR of the polygon lines, leaf rectangle overlaps with the parents right, data objects may duplicate across leafs. So, unlike this R-tree here the data objects may duplicate over the leafs this is possible.

But find operations same R tree, but only one child is followed now right, same as R-tree right that it goes to the x R, R decide x and type of things. But, at the end only one child is followed downright so; that means, in this case if you look for 5 roots as identifies x, x identifies b and c either b or c finds the object 5 right. So, it is either of the things so, only one child contains the thing right, one of the things I can have. So that means, whether I follow b or c I get the child, it is I do not have to do the things like any other things right.

(Refer Slide Time: 23:09)



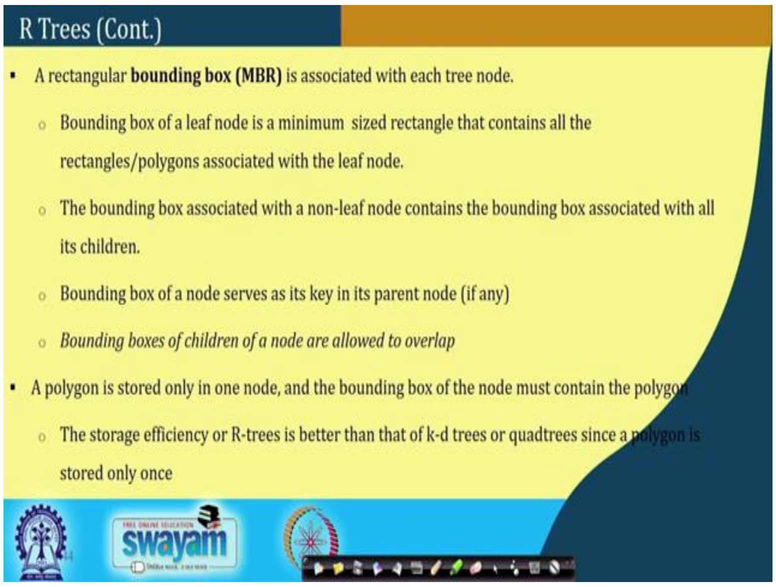
R-trees

- [*Guttman 84*] Main idea: allow parents to overlap!
  - => guaranteed 50% utilization
  - => easier insertion/split algorithms.
  - (only deal with Minimum Bounding Rectangles - MBRs)
  - A multi-way external memory tree
  - Index nodes and data (leaf) nodes
  - All leaf nodes appear on the same level
  - Every node contains between m and M entries
  - The root node has at least 2 entries (children)

So, if you look at so, the main idea so, is the R-trees are in case of basic R-tree we will discuss about the basic R-tree now and see different aspects that how insert, delete etcetera can go on is that allows parents to overlap right. Guarantee 50 percent utilization, easier insertion, split algorithm. So, only deal with minimum bounding rectangles or MBRs right. So, it is so, if you look at if you recollect our initial couple of lectures back or one or two lectures back, we are talking about the query possessing we want to do it initially a filter and then refinement. So, this is a filtering stage where one is followed.

A multi-way external memory tree, index node and data leaf nodes right, all leaf nodes appears on same level, every node contains between m and capital M entries right ok. So, the root node has at least 2 entries or children right. So, this is a same thing multi-way external memory tree; so, that we have a multi way tree. All leaf nodes appears on the same level, we have told this is a balanced tree and leaf nodes the same level that is on another requirement. Every node contain between m small m and capital number of entries and the root node has at least 2 entries, the at least the root should have at least 2 children right. So, that is the way it works.

(Refer Slide Time: 24:55)



R Trees (Cont.)

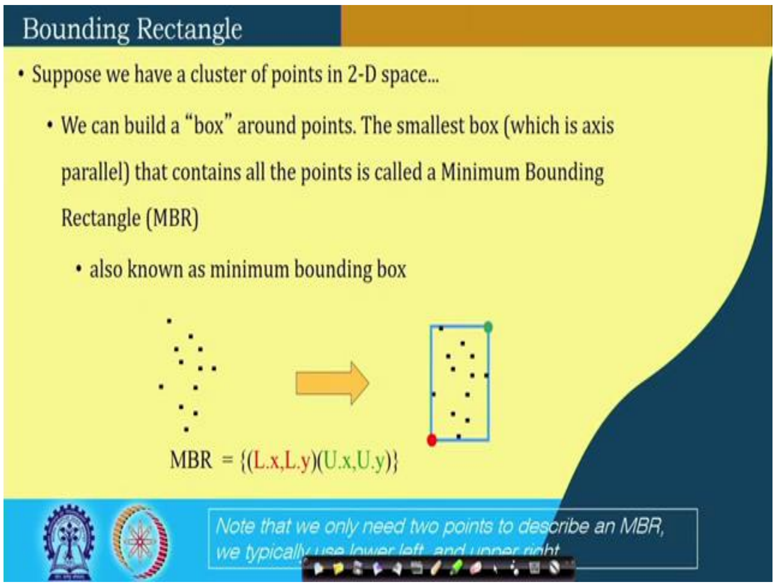
- A rectangular **bounding box (MBR)** is associated with each tree node.
    - Bounding box of a leaf node is a minimum sized rectangle that contains all the rectangles/polygons associated with the leaf node.
    - The bounding box associated with a non-leaf node contains the bounding box associated with all its children.
    - Bounding box of a node serves as its key in its parent node (if any)
    - *Bounding boxes of children of a node are allowed to overlap*
- A polygon is stored only in one node, and the bounding box of the node must contain the polygon
    - The storage efficiency or R-trees is better than that of k-d trees or quadtrees since a polygon is stored only once

So, it so for R trees a rectangular bounding box MBR is associated with each tree node that we have seen, that R tree always like if you look at this all are rectangular bounding box right. So, bounding box of a leaf node is minimum size rectangle that contain all the rectangles or polygons associated with the leaf node, that is there that is the definition or thing of the bounding box R, MBR what we are telling. The bounding box associated with the non-leaf node contains the bounding boxes associated with its children. So, leaf node is all the leaf node is basically connecting to the data itself.

So, it is the minimal that minimal rectangle which covers all the polygons etcetera under these things whereas, the non-leaf node it is all the children's bounding box it is in compressive right. So, for the bounding box of a leaf node serves as a key to the parent node if any, bounding box of a children of a node is allowed to overlap right. So, a polygon is stored in only one node right, the bounding box of the node must contain the polygon. The storage efficiency of R-trees is better than that of k-d tree and quadtree since the polygon stored it in only one place.

So, we have seen that the R-tree is better than this k-d tree or your quadtree, in the quadtree if you remember sometime back what we are told that it can be it that same like the yellow patch or the wheat patch it has it has come into three things. So, in order to access this particular polygon we have to access this 3 right, that is not true for in case of our this polygon. This in case of our R-tree, I have only one who has this content of the thing.

(Refer Slide Time: 26:57)



Bounding Rectangle

- Suppose we have a cluster of points in 2-D space...
  - We can build a "box" around points. The smallest box (which is axis parallel) that contains all the points is called a Minimum Bounding Rectangle (MBR)
    - also known as minimum bounding box

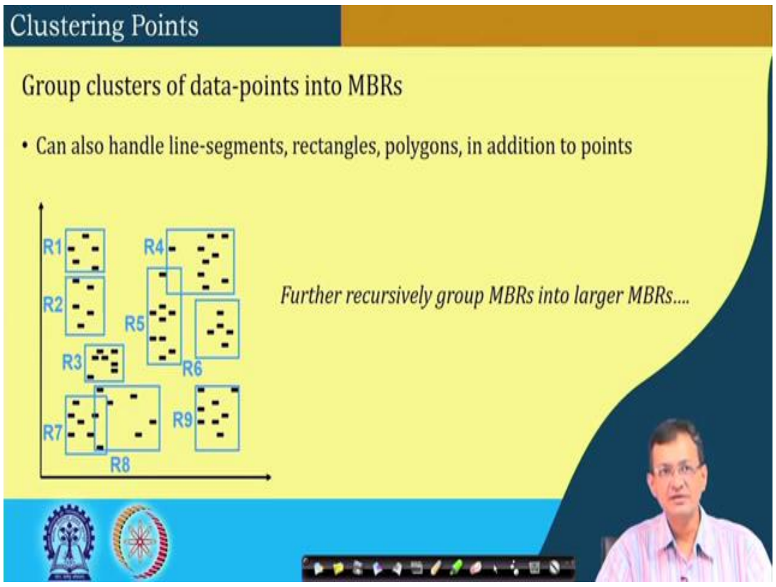
MBR = {(L.x,L.y)(U.x,U.y)}

Note that we only need two points to describe an MBR, we typically use lower left and upper right

So, bounding rectangle as we see that is a already we have discussed. So, build a box around the points, the smallest box which is the axis parallel that contains the point is called a Minimum Bounding Box or MBR right, also known as the minimum bounding box. Like if this is my points so, this is my minimum bounding box. And, what you need to store? This upper left hand corner or sorry lower left hand corner or upper right hand corner right, it can be other way around also right; means basically I want to store the 2 diagonal things right.

So, these two things I need to store, let us consider that we have considering lower left hand corner and the upper right hand corner right. So, that we need to store only 2 point describe that MBR, we typically used lower left hand and upper right; so, that we use things. So, whole point is compressing the things. So, what we the storage is efficient, accessing is much efficient, finding overlap or finding a encompassing thing etcetera is pretty efficient whether inside or overlapping etcetera is very efficient because, we need to compare only 2 coordinates. But, it may not be the final query points for that I need to have a refinement stage.

(Refer Slide Time: 28:25)



Clustering Points

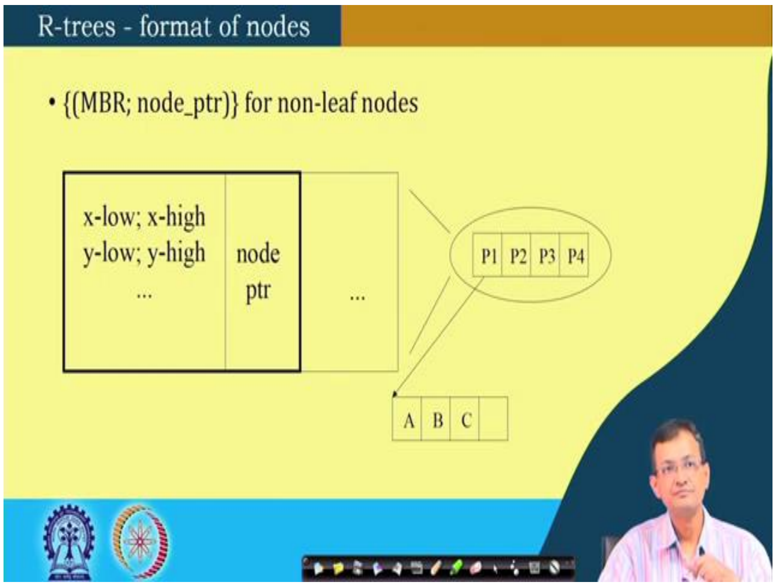
Group clusters of data-points into MBRs

- Can also handle line-segments, rectangles, polygons, in addition to points

Further recursively group MBRs into larger MBRs....

Nevertheless it helps me in improving out a majority of the things which are not in a query space. So, this group clusters are points into MBRs right. So, these are different MBRs, it can also handle line segment rectangle in addition to the points right, the recursively grouped MBRs into larger MBRs. So, I can go on recursively to create a larger MBR. So, this clustering clusters of the points can be done as the MBRs right.

(Refer Slide Time: 28:49)

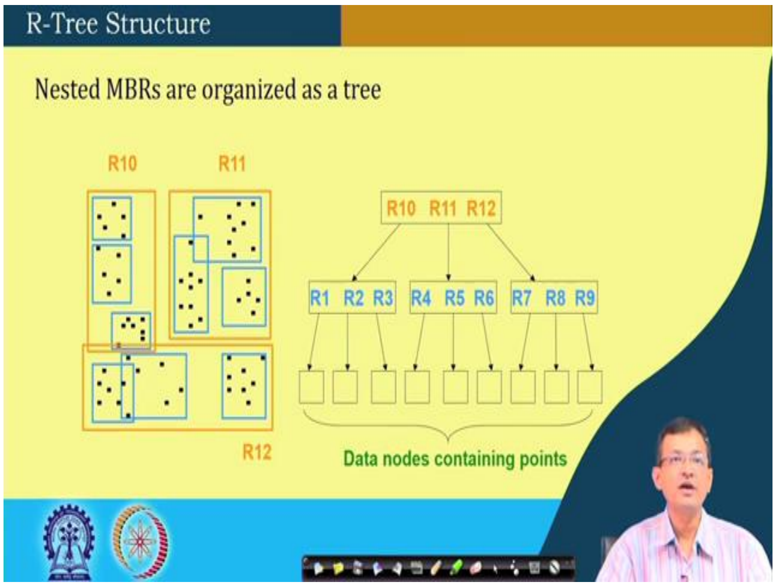


R-trees - format of nodes

- {(MBR; node_ptr)} for non-leaf nodes

So, is if you look at the format of any node of this R tree or type of things things. So, it has a x low high low, y low, y high type of things like; that means, MBRs 2 diagonal coordinates and the node pointer to point to that particular things. So, this is the thing the structure a is of any this format of the nodes of the things.

(Refer Slide Time: 29:33)



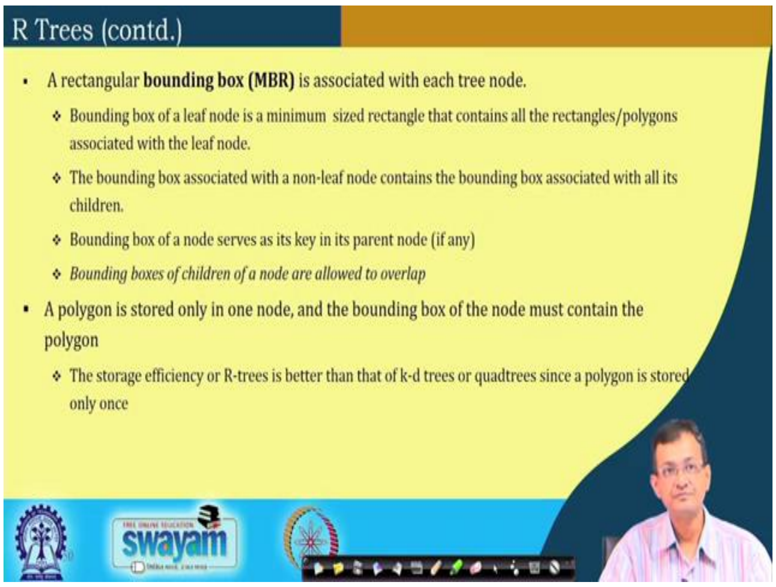
R-Tree Structure

Nested MBRs are organized as a tree

Like I have that MBR which encompasses its this children and then we have other that node pointer, it connect to the thing, connect to its children. So, as we discuss that can be nested MBR. So, R 10 R 11 R 12 R 10 encompassing R 1 R 2 R 3 R 11 R 4 R 5 R 6 R 7 etcetera right. So, this and down the line these are the different data nodes right.

So, data nodes can be directly hitting to the data or pointing to the data area whatever way it is manage managing the things right. So, this structure helps me in finding out if it is in a R-tree structure, it helps me to helps us to finding out the whole polygon in one hit oh in one node right, I it is not fragmented and distributed in multiple node per say right. So, that is a good part of it and it where it brings the efficiency into the thing.

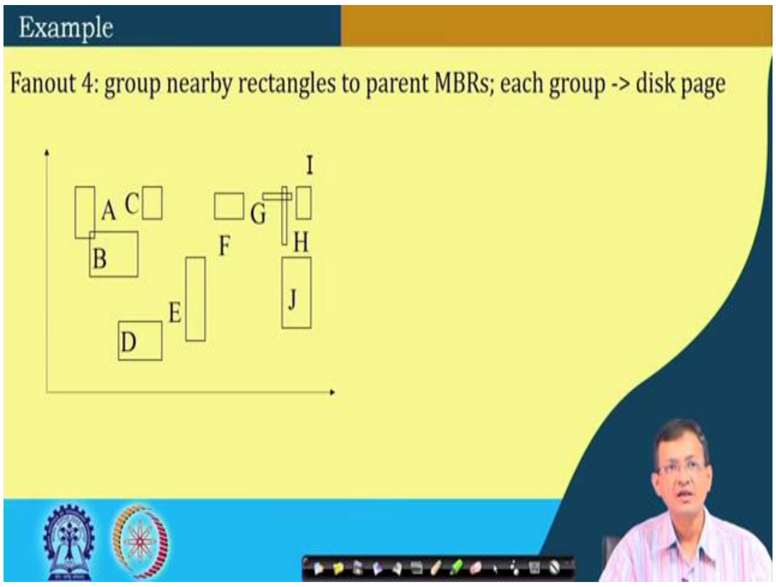
(Refer Slide Time: 30:29)


R Trees (contd.)

- A rectangular **bounding box (MBR)** is associated with each tree node.
  - Bounding box of a leaf node is a minimum sized rectangle that contains all the rectangles/polygons associated with the leaf node.
  - The bounding box associated with a non-leaf node contains the bounding box associated with all its children.
  - Bounding box of a node serves as its key in its parent node (if any)
  - *Bounding boxes of children of a node are allowed to overlap*
- A polygon is stored only in one node, and the bounding box of the node must contain the polygon
  - The storage efficiency or R-trees is better than that of k-d trees or quadtrees since a polygon is stored only once

So, rectangular bounding box MBR is associated with each tree node, bounding box of a leaf node is minimum sized rectangle that contains all the rectangles polygons associated with the leaf node. The bounding box associated with a non-leaf node contains the bounding box associated with its children. The bounding box of a node and its key its parent node if any then we have already seen this. So, storage efficiency is better than k-d tree and type of things.

(Refer Slide Time: 30:59)

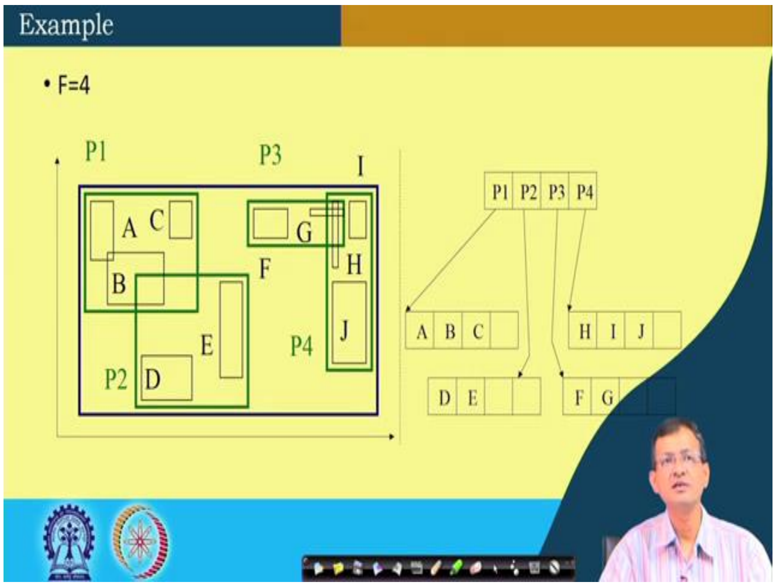


Example

Fanout 4: group nearby rectangles to parent MBRs; each group -> disk page

Like typically if we have this type of MBR rectangle into the things with that 4 fanout, we can construct this thing in this way right.

(Refer Slide Time: 31:07)

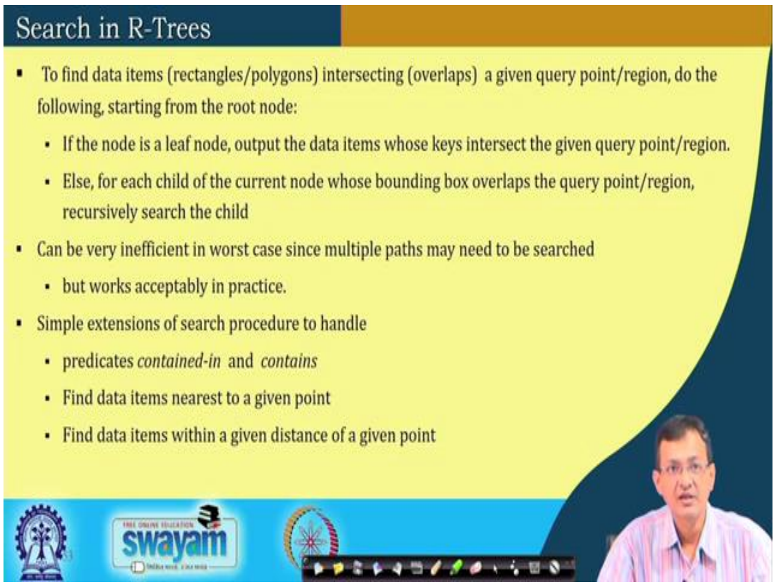


Example

- F=4

So, P 1 P 2 P 3 P 4 P 1 is containing A B C, P 2 is containing D E, P 3 is F G and P 4 G H I. You can see that there are overlap between P 1 and P 2 which is allowed in this of earlier node, but also in the P 3 P 4 nevertheless the root node there is no repetition right. So, A B C D E F G H I all those things there is no repetition right.

(Refer Slide Time: 31:41)



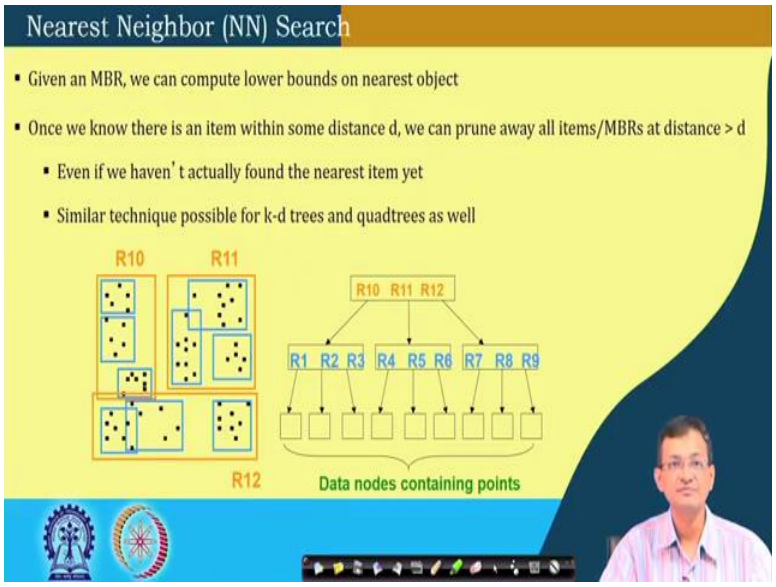
Search in R-Trees

- To find data items (rectangles/polygons) intersecting (overlaps) a given query point/region, do the following, starting from the root node:
  - If the node is a leaf node, output the data items whose keys intersect the given query point/region.
  - Else, for each child of the current node whose bounding box overlaps the query point/region, recursively search the child
- Can be very inefficient in worst case since multiple paths may need to be searched
  - but works acceptably in practice.
- Simple extensions of search procedure to handle
  - predicates *contained-in* and *contains*
  - Find data items nearest to a given point
  - Find data items within a given distance of a given point

So, if we want to search in a R-tree to find data rectangle overlaps in a given query region what we do? The node is a leaf node, output of the data whose keys are intersecting to the query polygon etcetera; so, that if it is the leaf node. For the other thing for each child the current node whose bounding box overlaps recursively search the child, if it is a non-leaf node right; so, recursively.

Can be very inefficient in worse case in multiple path need to be followed inefficient, but in general what you has been found that this good in practice because the distribution is such a way. So, simple extension of the search procedure to handle like contained in or contains type of things find data items nearest to a given point, find data items within a given distance of a given point. So, this type of things are pretty, this can be easily search into the things right.

(Refer Slide Time: 32:41)



Nearest Neighbor (NN) Search

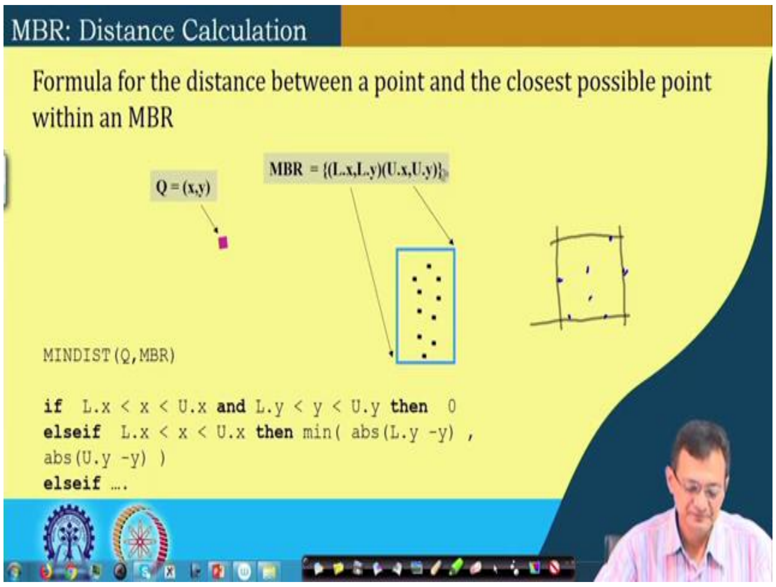
- Given an MBR, we can compute lower bounds on nearest object
- Once we know there is an item within some distance d, we can prune away all items/MBRs at distance > d
  - Even if we haven't actually found the nearest item yet
  - Similar technique possible for k-d trees and quadtrees as well

R10
R11
R12
R10 R11 R12
R1 R2 R3
R4 R5 R6
R7 R8 R9
Data nodes containing points

So, like one is the nearest neighbor search or NN search given in MBR, we can compute the lower bound of the nearest neighbor nearest object. Once we know there is an item within some distance d we can prove away all items MBRs at a distance greater than d right. If we have and actually found the nearest item yet similarly, technique possible for k-d trees and quad trees as well right. So, that is this type of techniques are also possible right.

So, what we see that by the searching the given MBR we can compute the lower bound of the nearest object etcetera. So, if the MBR is there I know that where is the where is the means search space of that particular things like what where its children etcetera is there. And, we can go on doing that like that once we know there is an item with some distance d, we can proven all items of the MBR with a distance more than d right.

(Refer Slide Time: 33:45)



MBR: Distance Calculation

Formula for the distance between a point and the closest possible point within an MBR

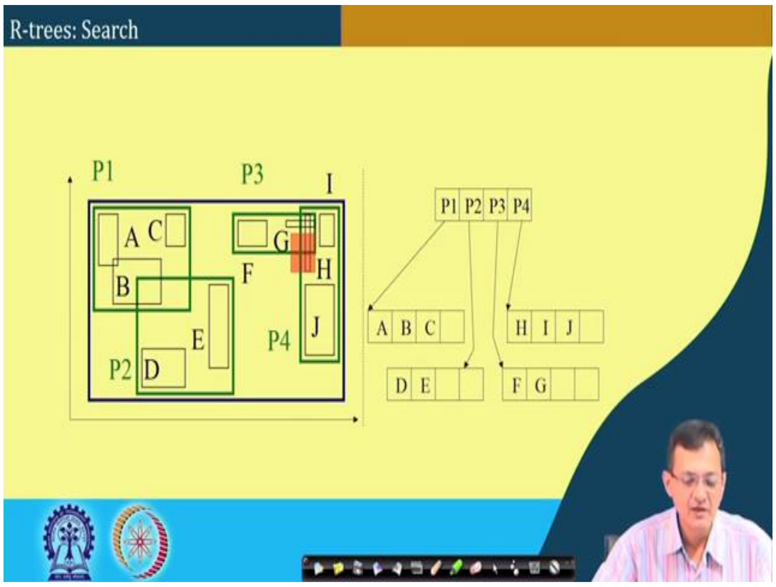
```
MINDIST(Q,MBR)

if  L.x < x < U.x and L.y < y < U.y then  0
elseif  L.x < x < U.x then min( abs(L.y -y) ,
abs(U.y -y) )
elseif ….
```

Like we can have a simple distance calculation formula like this that if it is a query point and like this if this is within x and y and this then 0; that means, within the things right within the MBR if it is there; that means, within the x range, but not within the y range then I can find out whether it is near this or this right. Now, you can see this, this drawing is little that MBR, if these are the points MBR; so, there have been this line right here and there right like if I have a points like this.

So, my ideally my MBR excuse me because I am not very good in drawing right. So, you see we get either the point within this or somewhere here or somewhere here or somewhere here or somewhere here. So, based on that within the x range, y range we can have a simple logic to find the things right.

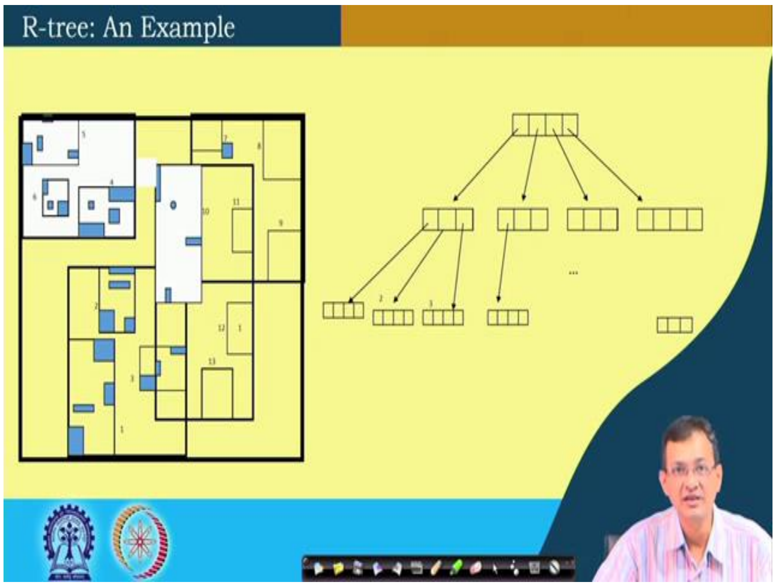
(Refer Slide Time: 35:05)



R-trees: Search

So, similarly as we have the search this; so, we can go on searching like here the candidates are P 3 and P 4 right. So, those two can be proved out and out of that search can be done right.

(Refer Slide Time: 35:21)



R-tree: An Example

So, similarly I can have a tree structure like this. So, what we have seen this today's discussion; one is that one major aspects of the quadtree which is a prevalent thing. And, another aspect of thing of the we discussed about R-tree structure. We will also little discuss about R-tree in our subsequent lecture or in the next lecture that how this R-tree, what are the different aspects of this of the R-tree. So, with this let us conclude our today’s discussion and we will continue on other aspects of spatial informatics in our subsequent lectures.

Thank you.