Spatiotemporal Data Management and Analytics

Question Set Day 2: Spatial Indexing and Query Processing

Semester 3, 2020

Question 1: Quadtree is a tree structure in which each internal node has exactly four children. In a quadtree, each node represents a bounding box covering some part of the space being indexed, with the root node covering the entire area. Is quadtree a balanced tree? What is the worst case of the two quadtrees? What's the main difference between Point Quadtree and Region Quadtree?

Example Solution:
Quad tree is not a balanced tree. The point quadtree's shape depends on the order in which points were inserted. The region quadtree's shape is affected by the point distribution.

Provide two examples of the unbalanced point quadtree and region quadtree.

The worst case of building a quadtree is linear. It takes $N(N-1)/2$ times of comparison because the i^{th} point requires $i-1$ comparison operations.

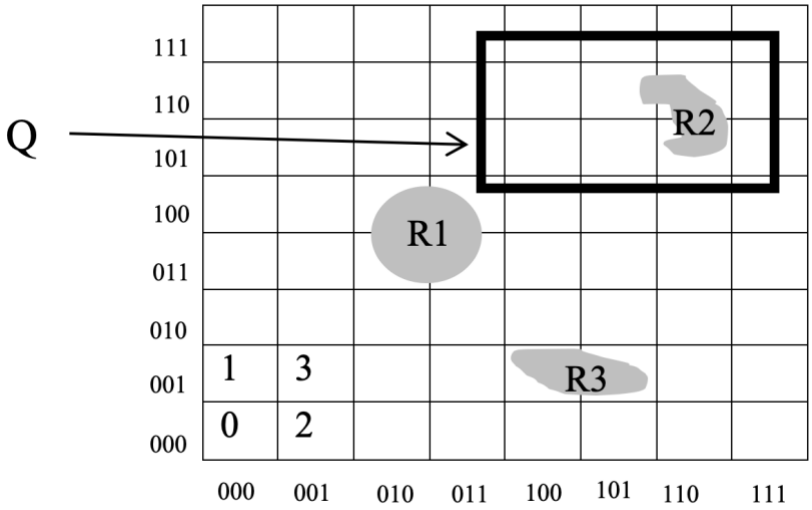
The main difference:
Point Quadtree

- The tree is constructed based on the points, its shape and size are dependent on the insertion order
- Data points can be stored in non-leaf nodes
- The space spanned by the point quadtree is rectangular and can be of infinite width and height
- Deletion is hard

Region Quadtree

- The tree is constructed based on regular decomposition of the space, and its shape and size are independent on the insertion order
- Data points are only stored in the leaf nodes
- The space spanned by the region quadtree is constraint to a maximum width and height
- Deletion is simple

Question 2: There are three spatial objects R1, R2 and R3 in the space. In order to index spatial objects using Z-values, the space has been recursively decomposed and the resolution (the maximum level of decomposition) is 3.



(1) What are the base-10, base-2, base-4, and base-5 Z-values of them?
(2) Assume that we represent spatial objects and the query window Q using as many Z-values as necessary subject to the given resolution. Which objects will be considered as possibly overlapping with Q based on their Z-values?

Example Solution:

(1) R1: (010,011),(010,100) ,(011,011),(011,100)
R2: (101,110), (110,101), (110,110)
R3: (100, 001), (101, 001)

- Base 10
  - R1: (13, 15, 24, 26)
  - R2: (54, 57, 60)
  - R3: (33, 35)
- Base 2
  - R1: (001101, 011000, 001111, 011010)
  - R2: (110110, 111100, 111001)
  - R3: (100001, 100011)
- Base 4:
  - R1: (031,033,120,122)
  - R2: (312, 330, 321)
  - R3: (201, 203)
- Base 5:

- R1: (142, 144, 231, 233)
- R2: (423, 441, 432)
- R3: (312, 314)

(2) R1 and R2

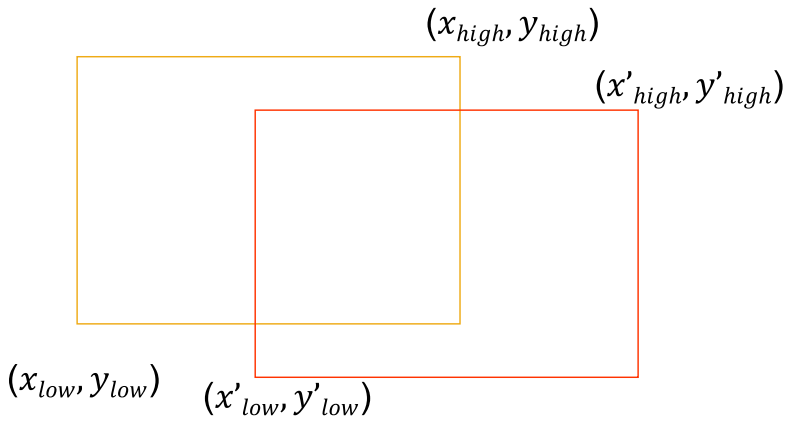
Question 3: Compare and contrast R-Tree and R+-Tree when they are used to index polygons.

Example Solution:
R-Tree allows node overlap of the internal MBRs while R+-Tree does not.
R-Tree only stores an object once, while R+-Tree will store that object in multiple leaf nodes.
The R+-Tree "clips" an object so MBRs do not have to overlap. This occurs during the split algorithm. The MBRs in R+-Tree are not physically "clipped".
Advantages of R+-Tree: A point query has a single unique path down the tree. It has less overlap so the queries may access less nodes.

Disadvantages of R+-Tree: It stores an object more than once. For a window query, it may return the same object more than once. It has more complex split algorithm, and sometimes leads to deadlock.

Question 4: Rectangle intersection is one of the most essential operation in the R-Tree window search. How can we do it efficiently?

Sample Solution:



One straightforward way to test if two rectangles intersect is reducing this problem to the point-in-rectangle test. However, it is hard to know which point to choose for the testing. If one rectangle is inside another one, then no point along the outside rectangle is inside the inner one. Or if two rectangles intersect like a crossing, none of the eight end-points of the two rectangles are inside the other's.

Therefore, to solve this problem more efficiently, we have to first check its dual problem: when are the two rectangles disjoint?

The first case is when one rectangle is above another one. Therefore, one rectangle's y_{low} should be larger than the other's y_{high}. The below case is similar.

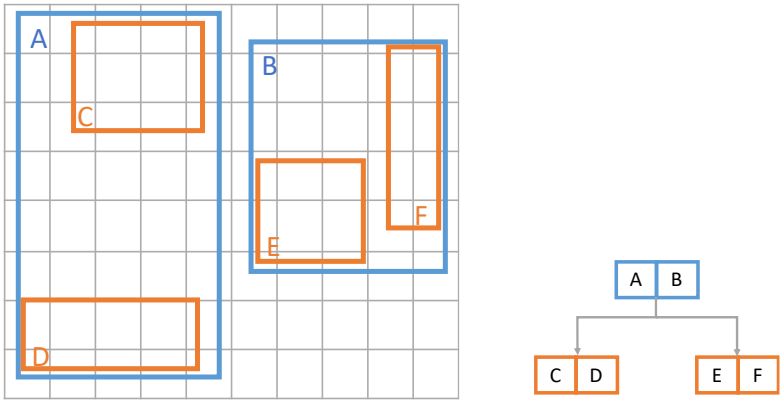
The second case is when one rectangle is on the right hand of the other one. Therefore, one rectangle's x_{low} is larger than the other's x_{high}. The left case is the also similar.

In this way, we have the conditions when two rectangles are disjoint:

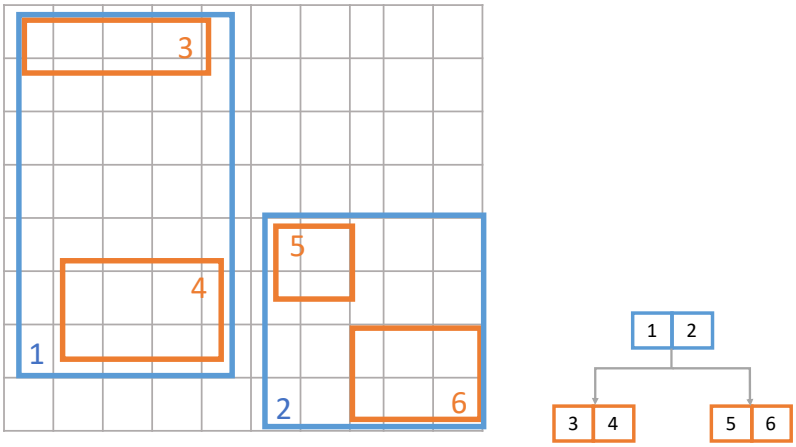
- $y'_{low} > y_{high}$
- $or\ y_{low} > y'_{high}$
- $or\ x_{low} > x'_{high}$
- $or\ x'_{low} > x_{high}$

So when any of the above conditions is true, it is guaranteed that two rectangles do not intersect. When none of these four conditions is satisfied, we say the two rectangles intersect. So it takes at most four comparisons.

Question 5: Nested Loop Spatial Join with R-Tree. The following is an example of two datasets R and S indexed by two R-Trees. Suppose we are doing spatial join on the polygon intersection. Please describe how to use the synchronize traversal to do the nested loop join algorithm.



(a) Dataset R



(b) Dataset S

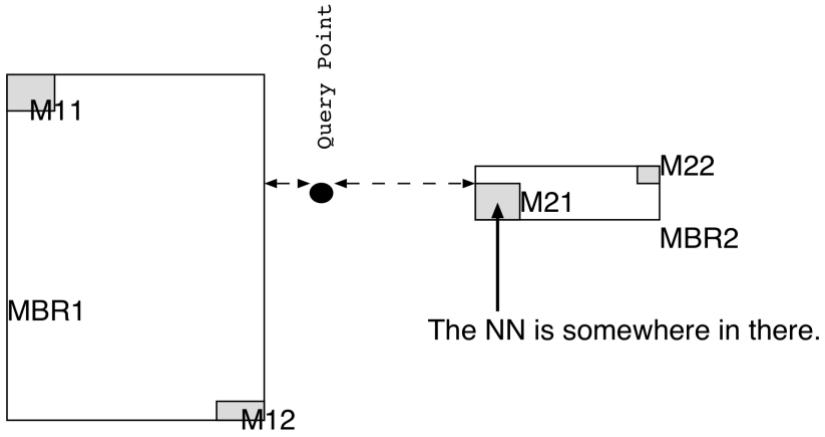
Sample Solution:
During the synchronized traversal, we visit the two R-Trees level by level and keep a task list. Suppose the task list is T, and the candidate list is C. Below are the detailed steps:

1. We visit the first level of the two trees. The root of R has two MBRs A and B, and the root of S has two MBRs 1 and 2. We add the cartesian product of these two sets of MBRs into the task list:
    a. T={(A,1), (A,2), (B,1), (B,2)}
    b. C=Φ
2. We test the first pair (A,1) in T. Because A and 1 intersect, we need further check their children. A has two children C and D, and 1 has two children 3 and 4. We add the cartesian product of them into the task list:
    a. T={(A,2), (B,1), (B,2), (C,3), (C,4), (D,3), (D,4)}
    b. C=Φ
3. We test next pair (A,2) in T. Because they do not intersect, we skip them directly.
    a. T={(B,1), (B,2), (C,3), (C,4), (D,3), (D,4)}
    b. C=Φ
4. We test next pair (B,1) in T. Because they do not intersect, we skip them directly.
    a. T={(B,2), (C,3), (C,4), (D,3), (D,4)}
    b. C=Φ
5. We test next pair (B,2) in T. Because B and 2 intersect, we need further check their children. B has two children E and F, and 2 has two children 5 and 6. We add the cartesian product of them into the task list:
    a. T={(C,3), (C,4), (D,3), (D,4), (E,5), (E,6), (F,5), (F,6)}
    b. C=Φ
6. Now all the tasks in the list are in the leaf level. We test them one by one and add the intersected ones in to the candidate list:
    a. C={(C,3), (D,4),(E,5)}

After obtaining the candidate list, we visit these MBRs, retrieve the actual objects, and test if they intersect or not.

Question 6: When using the R-tree to search for the nearest neighbor, we have two metrics to determine the search order: MINDIST and MINMAXDIST. The MINDIST is the optimistic choice, while the MINMAXDIST is the pessimistic one. Is the MINDIST always a better ordering than using the MINMAXDIST?

Sample Solution:



MINDIST ordering is not always better than MINMAXDIST ordering.

As shown in the above example, the nearest neighbor is somewhere inside M21. If we use the MINDIST ordering, because MBR1 is nearer than MBR2, we will visit MBR1 first and also visit M11 and M12 in it. If we use the MINMAXDIST ordering, because MBR2 has smaller MINMAXDIST than MBR1, we will visit MBR2 first and then M21. When we visit MBR1, we can prune M11 and M12.

Question 7: How to extend the Best-First Nearest Neighbor algorithm to answer the kNN query?

Sample Solution:
In the nearest neighbor search, we only return the top-1 result. In kNN search, we need to return k results, so we keep a buffer to store the current found nearest neighbors until its size reaches k.

One way to do it using the BF method is we maintain a candidate list of k elements and uses the k^{th} nearest one as the pruning threshold. This candidate list keeps updating until the smallest value in the heap (the one organizing the R-Tree MBR visiting order) is larger than the k^{th} nearest distance, then we can use the candidate list as the result.