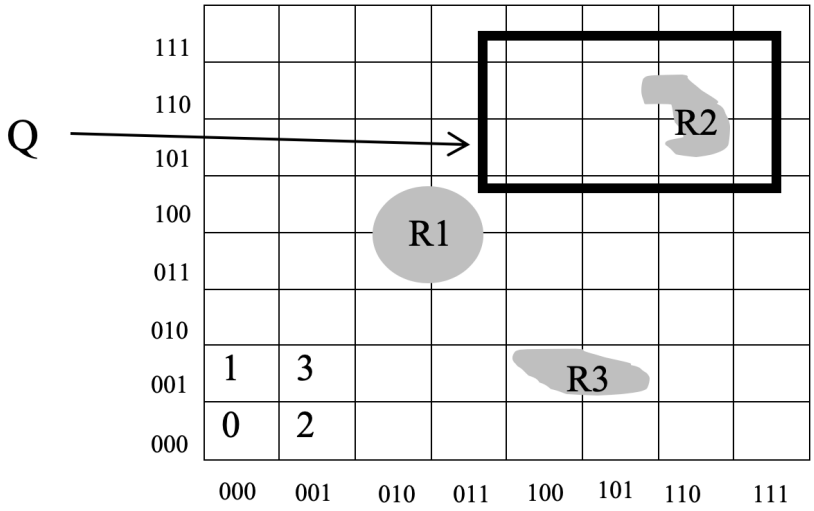
INFS4205/7205 Advanced Techniques for High Dimensional Data

Tutorial 3: Spatial Data Organization 2 & Spatial Query Processing 1

Semester 1, 2021

Question 1: 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 2: Compare and contrast R-Tree and R^+-Tree when they are used to index polygons.

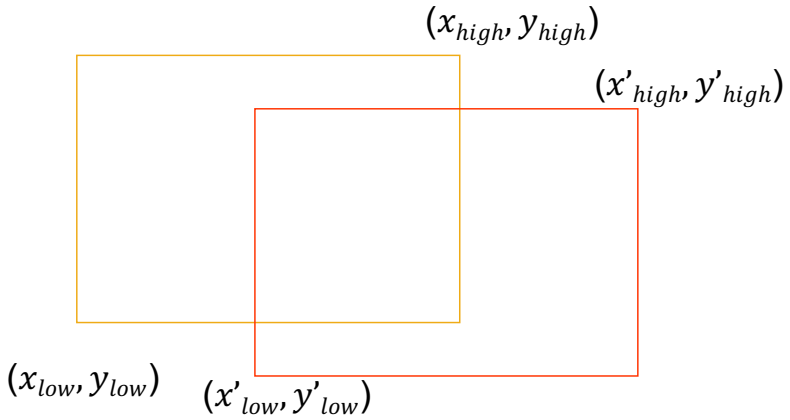
Example Solution:
R-Tree allows node overlap of the internal MBRs while R^+-Tree does not.
R-Tree only stores an object once, while R^+-Tree will store that object in multiple leaf nodes.
The R^+-Tree "clips" an object so MBRs do not have to overlap. This occurs during the split algorithm. The MBRs in 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 3: Rectangle intersection is one of the most essential operation in the R-Tree window search. How can we do it efficiently?

Example 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 endpoints of the two rectangles are inside the others.

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.

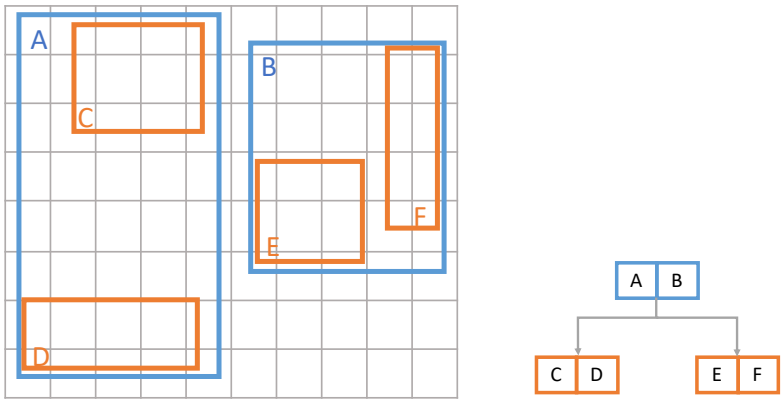
The second case is when one rectangle is on the right hand of the other one. Therefore, one rectangle's x_{low} is larger than the other's x_{high}. The left case is the also similar.

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

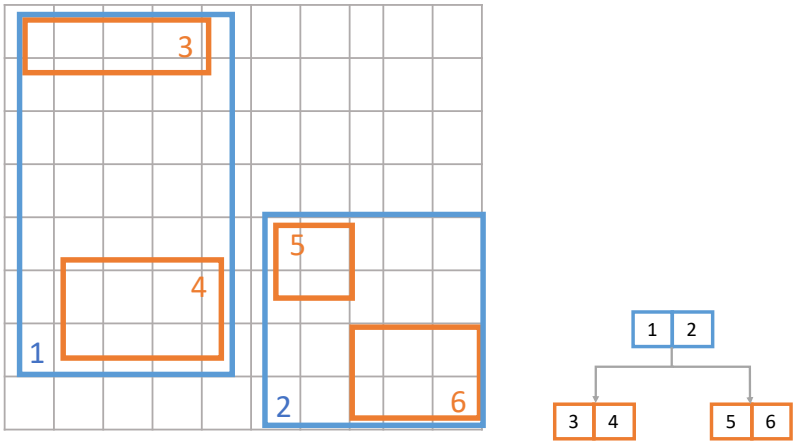
- $y'_{low} > y_{high}$
- $or\ y_{low} > y'_{high}$
- $or\ x_{low} > x'_{high}$
- $or\ x'_{low} > x_{high}$

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 4: 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

Example 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.