INFS4205/7205 Advanced Techniques for High Dimensional Data

Tutorial 2: Spatial Data Organization 1

Semester 1, 2021

Question 1: What is the query time of answering the vertical line query using an interval tree? What is the query time of answering vertical segment query?

Example Solution:
The vertical line query takes $O(\log n)$ time to visit at most $\log n$ crossed nodes. Because all the line segments in the crossed node are retrieved in the pre-sorted order, so only k (the number of the results) will be visited. The total query time is $O(\log n)$.

The vertical segment query has to further check the y dimension, so it uses the 2D Range tree instead of the sorted list to index the end points. If the 2D Range Tree uses a binary tree as the auxiliary structure same as the lecture note, the search of 2D Range tree takes $O(\log^2 n + k_v)$ time, and the total time complexity is $O(\log^3 n + k)$. If the 2D Range tree uses the *fractional cascading* on the auxiliary structure, it takes a $O(\log n + k_u)$ time, so the total time complexity is $O(\log^2 n + k)$.

Question 2: The segment tree organizes the segments based on their end points. The line segments are stored at any nodes u that it covers it entirely while it doesn't cover u's parent's region. Why it cannot appear in the same level for more than two times?

Example Solution:
The line segment is continuous, so it will cover the nodes continuously.

Firstly, for a level of nodes, the segment left end could only appear in its parents' right child, and the right end could only appear in its parent's left child. Otherwise, the segment will store in its parent instead.

Secondly, for any nodes between the left end and the right end, they and their siblings are all covered by the segment, so the segment is stored in their ancestors.

Therefore, for each level, only the two ends are possible to store the segment.

Question 3: Compare how to answer the line segment range query with interval tree, priority search tree, and segment tree.

Example Solution:
The result line segments have two types: the ones with at least one end-point in the rectangle, and the ones have no endpoint in the rectangle. The first kind can be answered with 2D range tree, while the second kind needs further computation.

Interval tree and priority tree are used to find the second kind of segments that are parallel to the axis. We need to build one tree for each axis. The interval tree itself takes $O(n)$ space. To answer the vertical segment search instead of vertical line search, the interval tree uses another set of 2D range trees within each node to achieve $O(\log^3 n + k)$ complexity (or $O(\log^2 n + k)$ with fractional cascading). If we use the priority search tree to organize the segments within each node, it takes $O(log\, n + k)$ to answer, so the overall time complexity is also $O(\log^2 n + k)$. However, the 2D range tree takes $O(nlogn)$ space, while the priority search tree version takes $O(n)$ space.

The segment tree can deal with the case when the segments are not required to be parallel to the axis. It takes $O(log^2\, n + k)$ to report results, and takes $O(nlog\, n)$ space.

Question 4: 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.

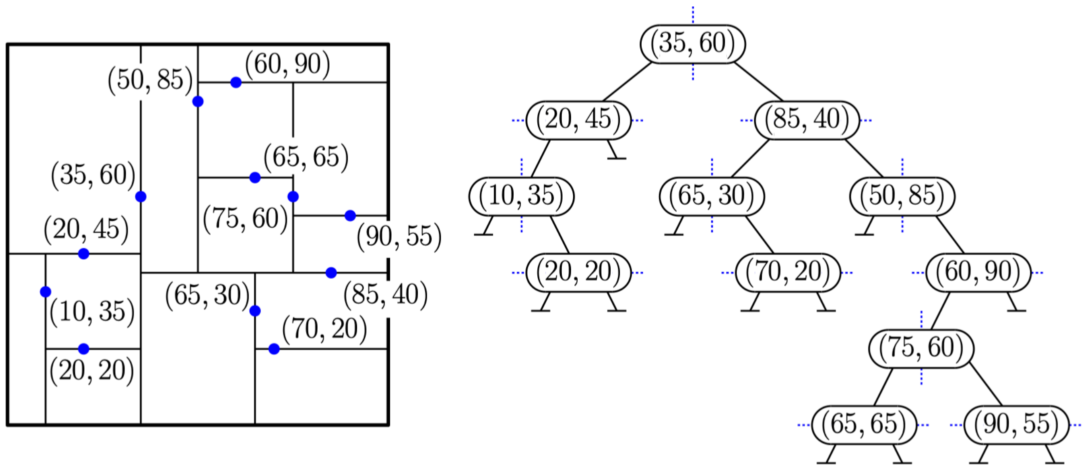
The main difference:
Point Quadtree

- 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

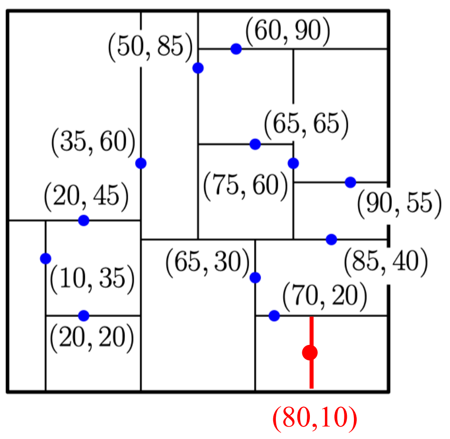
- The tree is constructed based on regular decomposition of the space, and its shape and size are independent on the insertion order
- Data points are only stored in the leaf nodes
- The space spanned by the region quadtree is constraint to a maximum width and height
- Deletion is simple

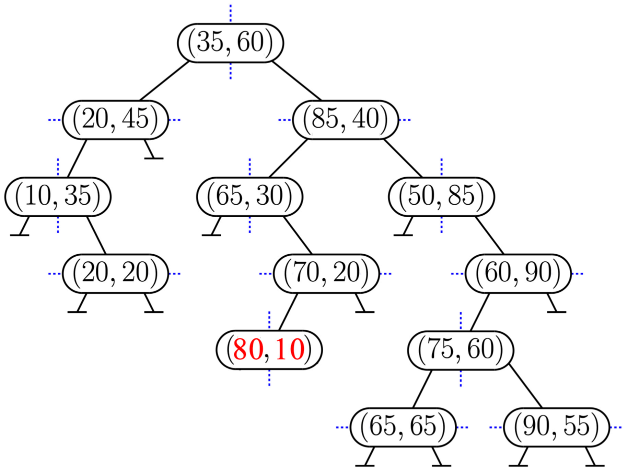
Question 5: Consider the Kd-Tree shown below. Assume that the cutting dimensions alternate between x and y.
(1) Show the result of inserting (80,10) into this tree.
(2) Show the Kd-Tree that results after deleting the point (35,60) from the *original* tree. Show both the tree structure and the subdivision of space of the two operations.



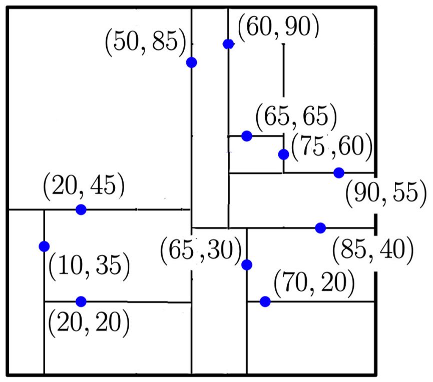
Example Solution:

(1) We find the point (80,10) and fall out of the tree on the right child of (70,20).We insert the new node here. Since the parent y-splitter, the new node is an x-splitter.

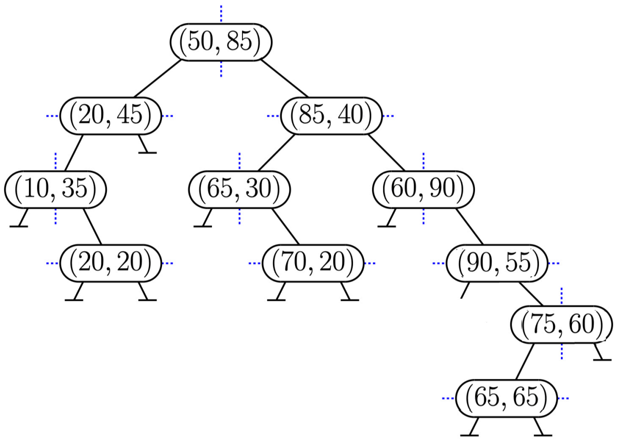




(2) The deleted node is at the root. Since it is an x-splitter, its replacement is the node with the smallest x-coordinate in the root's right child, which is (50,85). Since (50,85) is an x-splitter, its replacement is the node with the smallest x-coordinate in its right subtree, which is (60,90). Since (60,90) is a y-splitter and its right subtree is empty, we find the point with smallest y-coordinate in its left subtree, which is (90,55), and we move this subtree to become the right child of (60,90). Finally, we delete (90,55) from this subtree. Since it is a leaf, it can simply be unlinked from the tree.



(50, 85)
(60, 90)
(65, 65)
(75 ,60)
(20, 45)
(90, 55)
(10, 35)
(65 ,30)
(85, 40)
(70, 20)
(20, 20)



(50, 85)
(20, 45)
(85, 40)
(10, 35)
(65, 30)
(60, 90)
(20, 20)
(70, 20)
(90, 55)
(75, 60)
(65, 65)