Week 4 Tutorial – Spatial Data organization 2

- Tutor: Fengmei JIN
- Email: fengmei.jin@uq.edu.au

Question 1

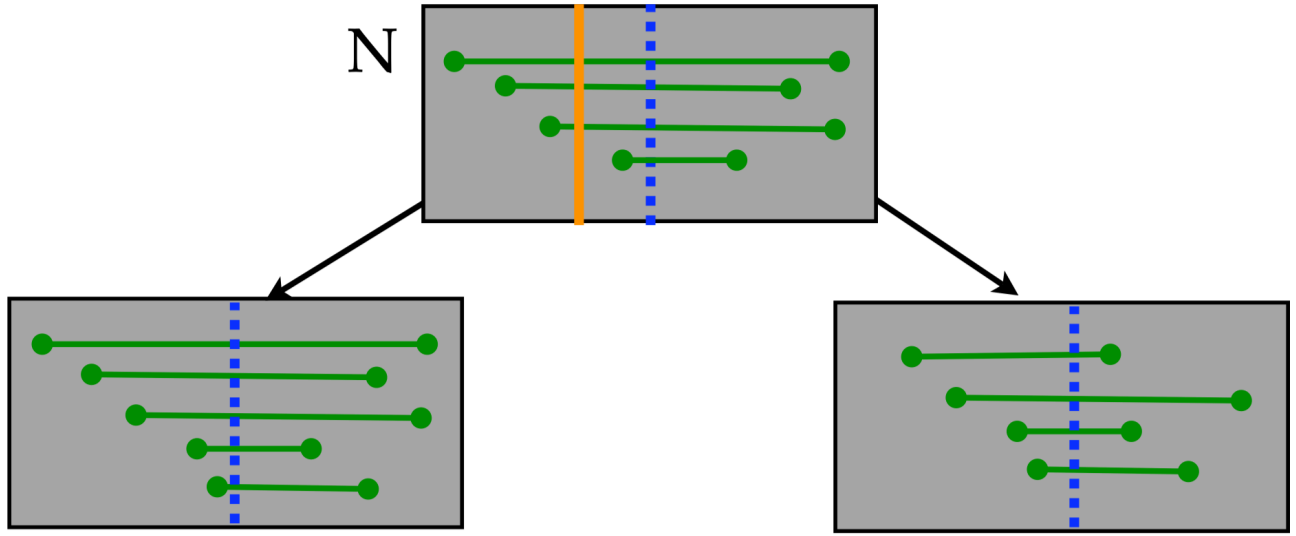
- 1) What is the query time of answering the vertical line query using an interval tree?
- 2) What is the query time of answering vertical segment query?

Question 1 (Answer)

- 1) What is the query time of answering the vertical line query using an interval tree?
  - 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 $O(\log n + k)$.

Vertical line query on an interval tree

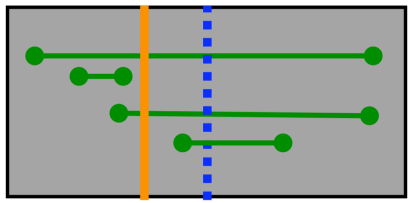
- Query x_q: a vertical line
  - Suppose we are at node N
    - If $x_q < x_{med}$, skip the right subtree
    - If $x_q \geq x_{med}$, skip the left subtree
    - **Always have to search the intervals stored at current node**
    - $logN$ nodes to check



Vertical line query on an interval tree

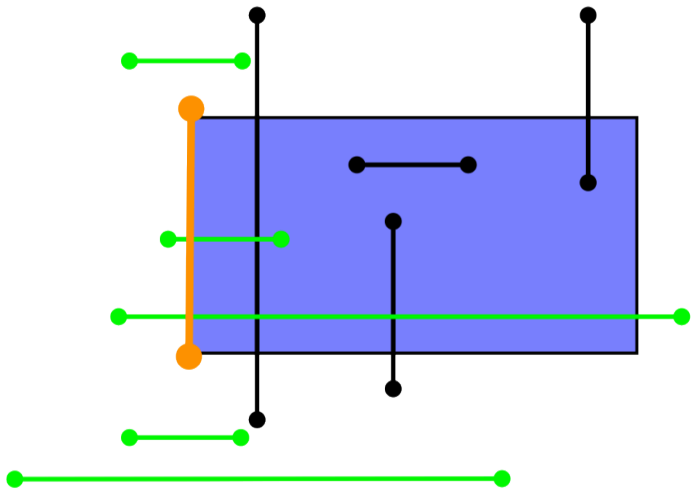
- Current Node Search
  - Store each interval in two sorted list stored at node
    - List L sorted by increasing left endpoint
    - List R sorted by decreasing right endpoint
  - Search List depending on which side of x_{med} the query is on
    - If $x_q < x_{med}$, search L, output all until a left endpoint $> x_q$
    - If $x_q \geq x_{med}$, search R, output all until a right endpoint $< x_q$

  - It works because we know each segment intersects x_{med}



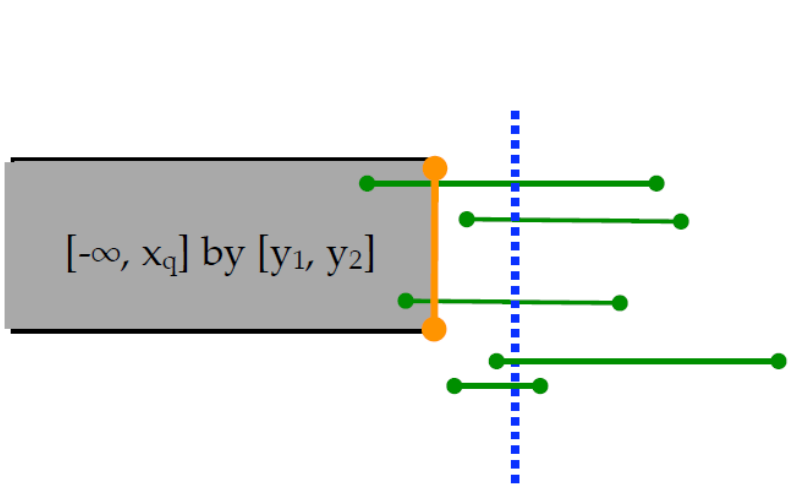
Question 1

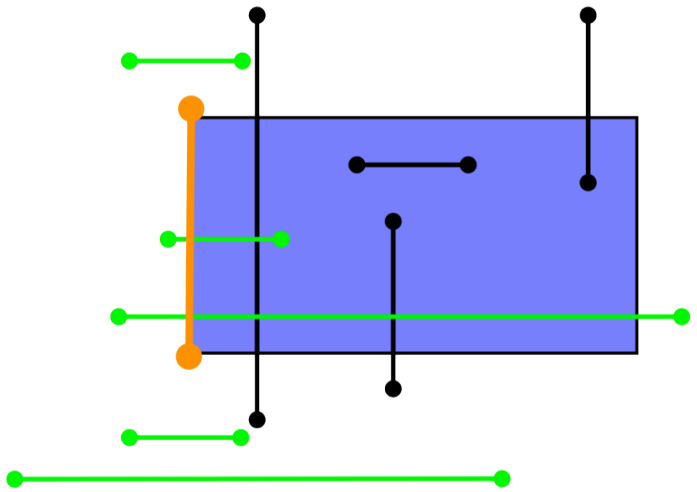
- 2) What is the query time of answering vertical segment query?



Question 1 (Answer)

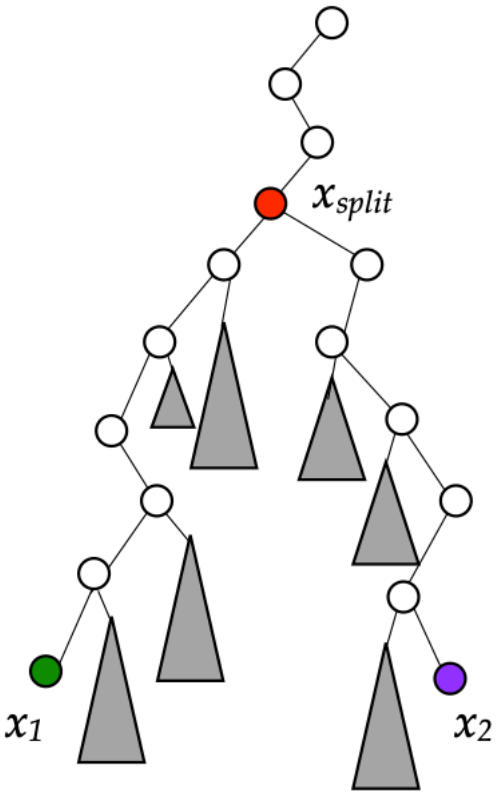
- 2) What is the query time of answering vertical segment query?
  - The vertical segment query has to further check the y dimension, so it uses the 2D Range tree instead of the sorted list to index the end points.
    - If the 2D Range Tree uses a binary tree as the auxiliary structure same as the lecture 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 $O(\log n + k_v)$ time, so the total time complexity is $O(\log^2 n + k)$.





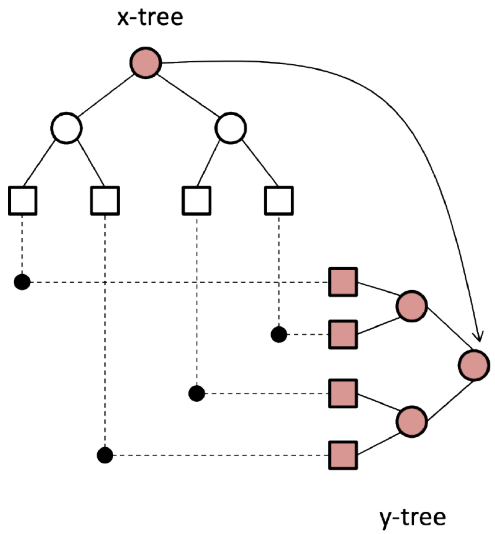
1D Range Tree

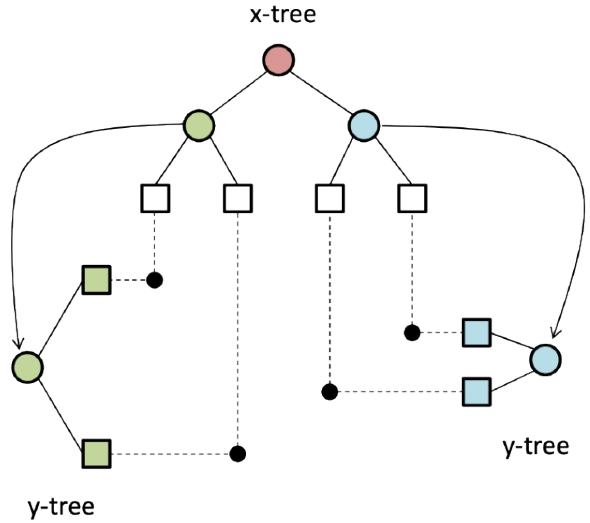
- Balanced Binary Search Tree
- Range query: “return all keys between x1 and x2”
  - Search for x1 and x2 (Assume all data are in the leaves)
  - Let x_{split} be the node where the search paths diverge
  - Output leaves in the right subtrees of nodes on the path from x_{split} to x1
  - Output leaves in the left subtrees of nodes on the path from x_{split} to x2
- Query time: $O(\log n + k)$
  - Tree height: $O(\log n)$, as well the length of paths to x1 and x2
  - Traversing the subtrees: $O(k)$, k is the number of output data items

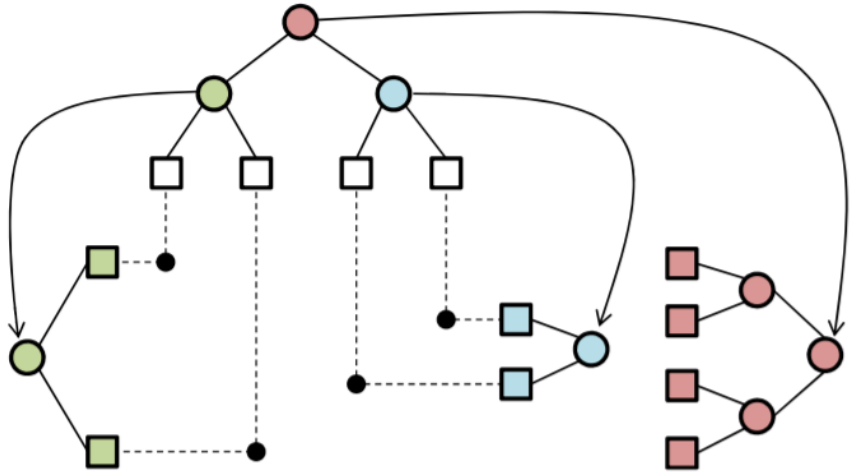


2D Range Tree

- Binary Tree of Binary Tree
  - One Binary Tree in X (x-Tree)
  - One Binary Tree in Y for each node in the X-Tree
- Complexity
  - Space: $O(n \log n)$
  - Time: $O(n \log n)$

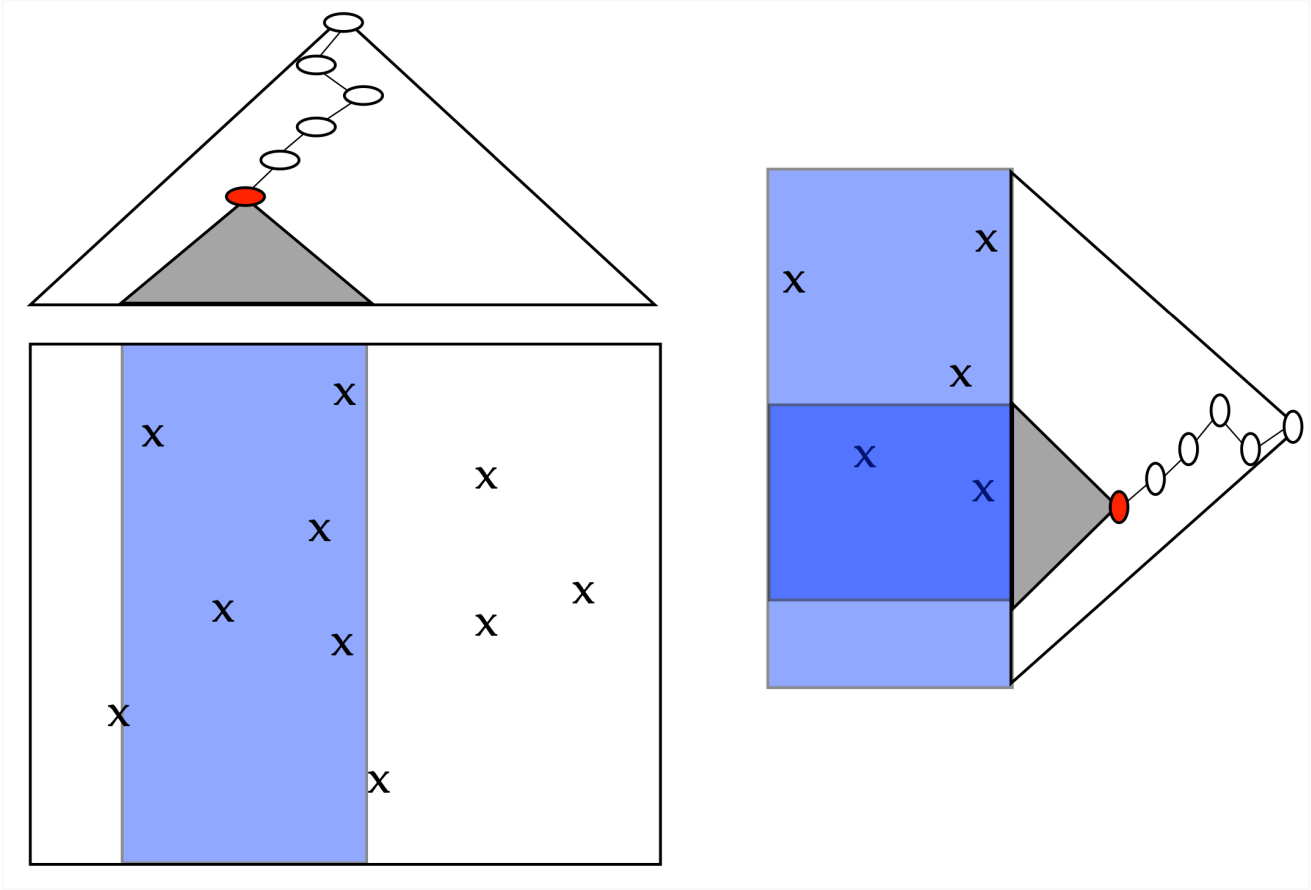






2D Range Tree Search

- Range Query $[x_1, x_2] \times [y_1, y_2]$
  - Treat it as two nested one-dimensional queries
  - Time complexity: $O(\log^2 n + k)$



Question 2

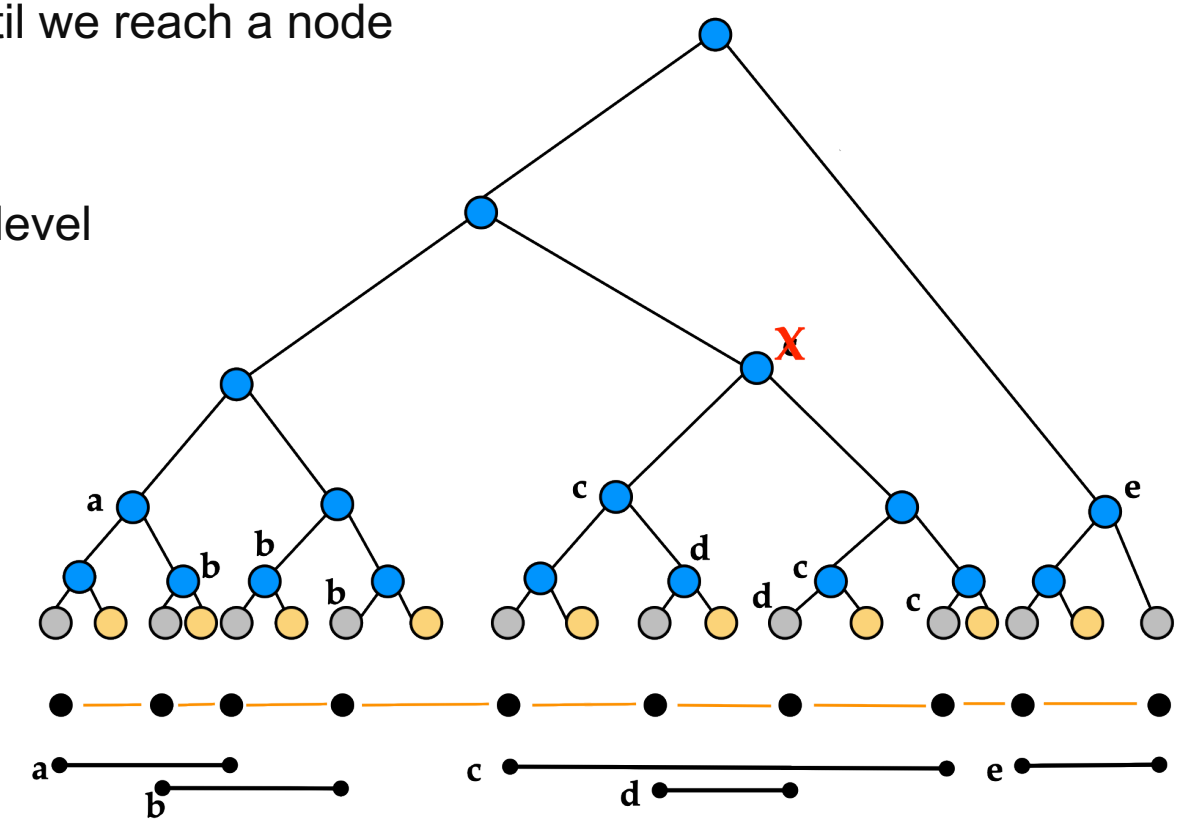
- The segment tree organizes the segments based on their end points. The line segments are stored at any node u that it covers entirely while it doesn't cover u's parent's region. Why it cannot appear in the same level for more than two times?

Question 2 (Answer)

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

Segment Tree

- Store the segments in the Binary Search Tree
  - Store the segment s at any node u that
    - segment covers the entire $Region(u)$
    - doesn't cover the entire $Region(Parent(u))$
      - In other words, we propagate segments up until we reach a node whose region is not a subset of the segment
    - Segments may be stored at several nodes
      - Each segment is stored at most twice at each level

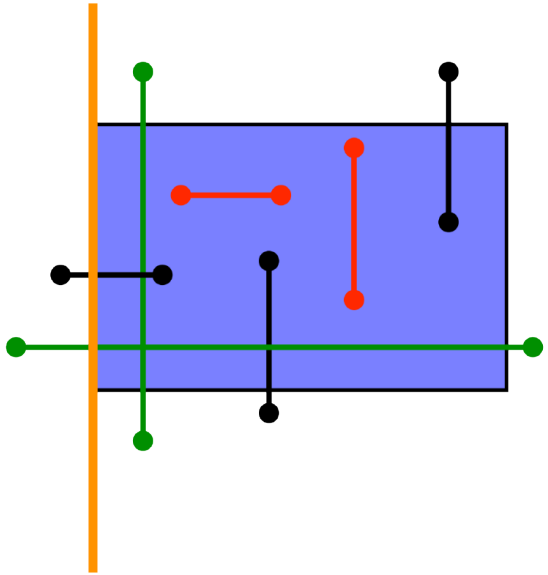


Question 3

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

Question 3 (Answer)

- Compare how to answer the line segment range query with interval tree, priority search tree, and segment tree.
  - The result line segments have two types:
    - 1) the ones with at least one end-point in the rectangle
    - 2) 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.

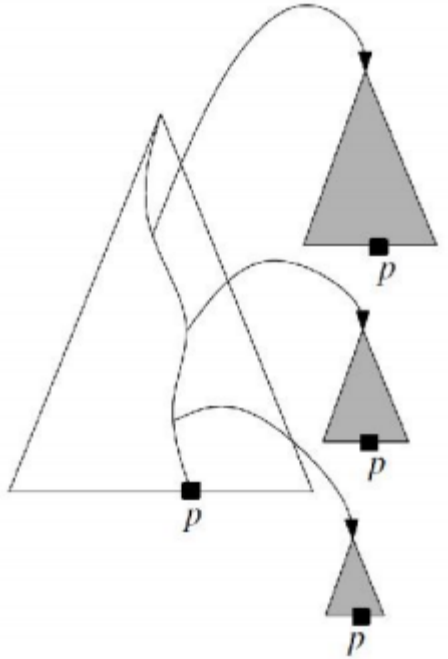


Question 3 (Answer)

- Compare how to answer the line segment range query with interval tree, priority search tree, and segment tree.
  - Interval tree and priority tree are used to find the second kind of segments that are parallel to the axis. So, 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 in total. → *same as Q1*
    - Instead on 2D range trees, we use the priority search tree to organize the segments within each node. Then, it takes $O(log\, n + k)$ to answer a $[-\infty, x], [y_1, y_2]$ query, so the overall time complexity is also $O(\log^2 n + k)$.
      - Space: the 2D range tree takes $O(nlogn)$ space, while the priority search tree version takes $O(n)$ space.

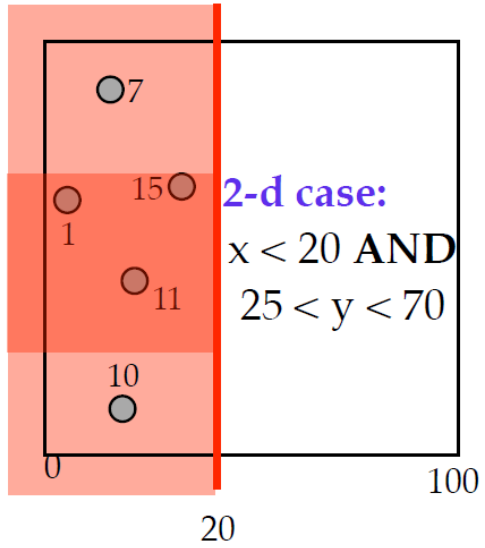
Vertical segment query on an interval tree with 2D range tree

- Query time is $O(\log^3 n + k)$
  - $\log n$ to walk down the interval tree
  - At each node v, we have to do an $O(\log^2 n + k_v)$ search on a 2D range tree
- Space is $O(n \log n)$
  - Each interval stores at one node
  - Total space for range trees holding <= 2n items is $O(n \log n)$
    - Let T_i be # of trees of which p is a leaf: $T_i = O(\log n)$
    - Total space is $O(\sum_{i=1}^{n} T_i) = O(n \log n)$



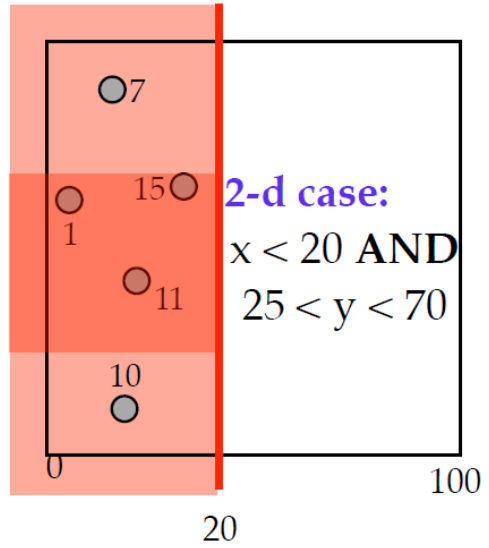
Priority Search Tree

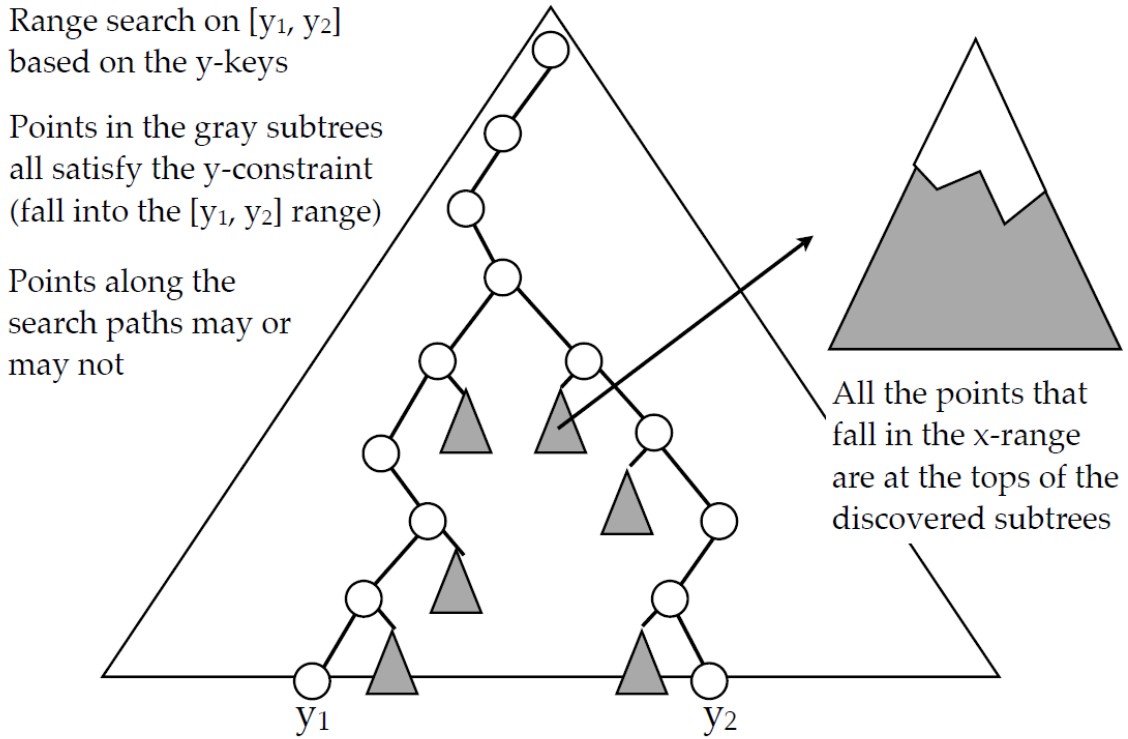
- Unbounded 1-Side Range Query
  - $[-\infty, x], [y_1, y_2]$
    - Want to find points with lower than x-values
    - Within a range of y-values
  - Idea
    - Find low x-values → heap
    - 1d range query on y-values → BST
  - Combine them!
    - Priority Search Tree



Priority Search Tree

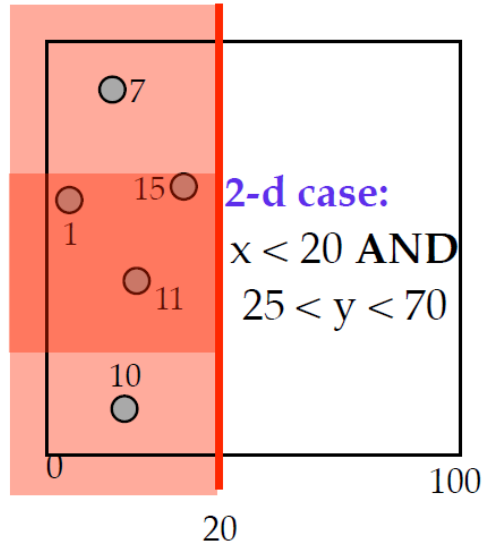
- Unbounded 1-Side Range Query
  - $[-\infty, x], [y_1, y_2]$
    - Want to find points with lower than x-values
    - Within a range of y-values
  - Idea
    - Find low x-values → heap
    - 1d range query on y-values → BST
  - Combine them!
    - Priority Search Tree

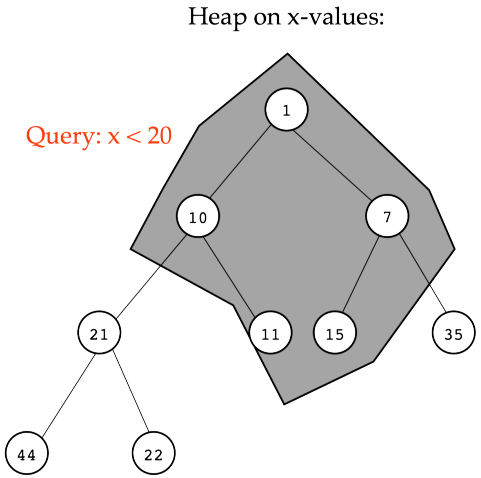


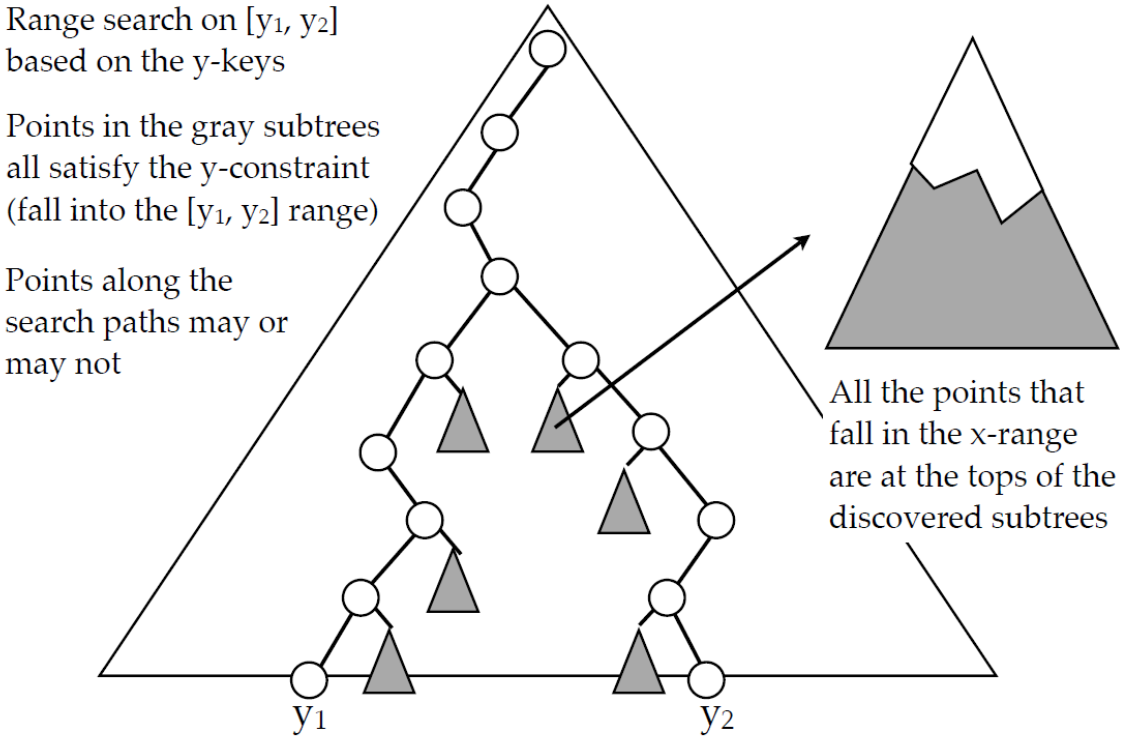


Priority Search Tree

- Unbounded 1-Side Range Query
  - $[-\infty, x], [y_1, y_2]$
    - Want to find points with lower than x-values
    - Within a range of y-values
  - Idea
    - Find low x-values → heap
    - 1d range query on y-values → BST
  - Combine them!
    - Priority Search Tree





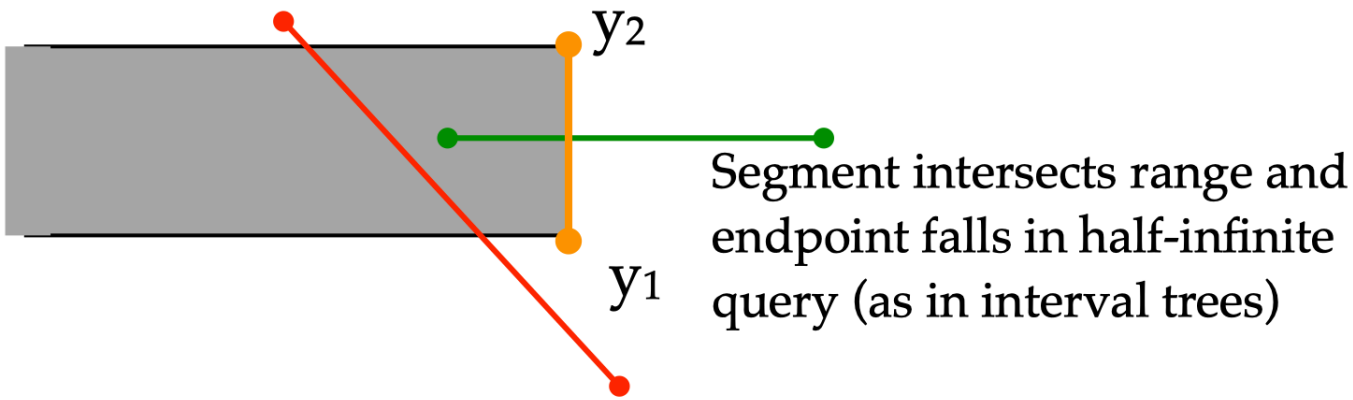


Vertical segment query on priority search tree

- Steps:
  - Range search on $[y_1, y_2]$ → *each subtree is a heap*
  - Then output "tops" of each subtree between the paths found during the range search satisfying the x-constraint.
- Time: $O(\log n)$ to find subtrees + $O(k)$ to output the tops → $O(\log n + k)$
  - Faster than searching on interval trees
- Space: $O(n)$
  - Smaller than interval trees

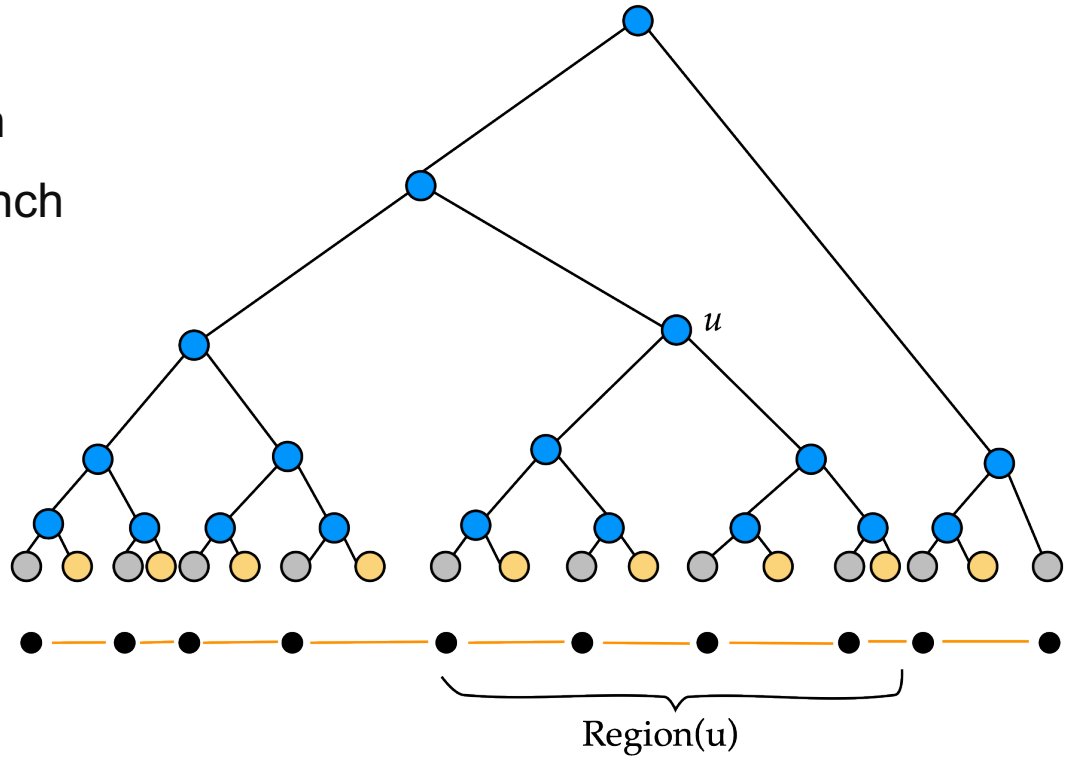
Question 3 (Answer)

- Compare how to answer the line segment range query with interval tree, priority search tree, and segment tree.
  - 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 $O(n \log n)$ space.
    - Space: as it is a BST, the height is $O(\log n)$; so, the total is $O(n \log n)$



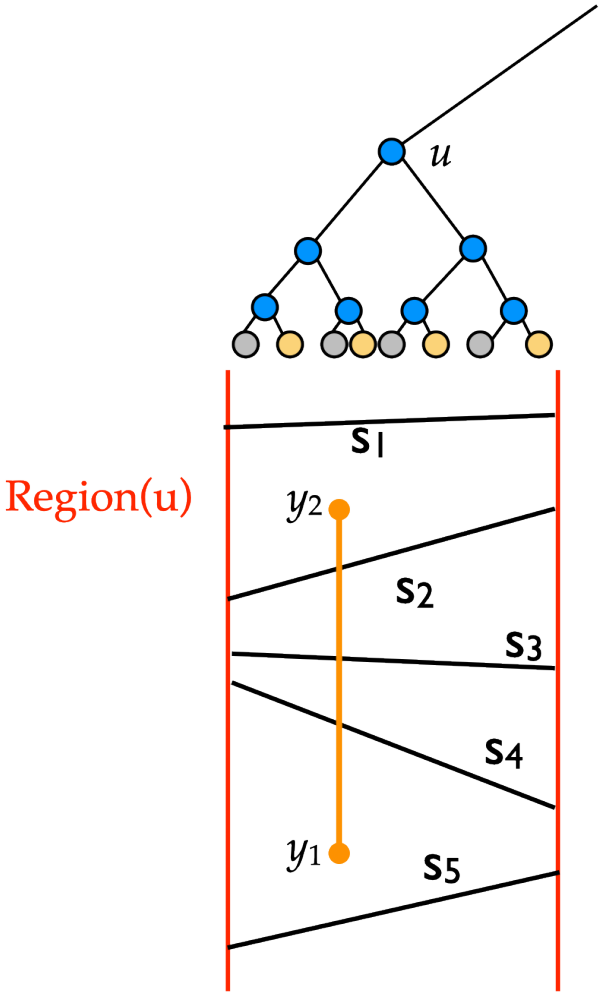
Vertical line query on segment tree

- Find segments that intersect a given line x
  - Binary Search traversal of the tree
  - At each step: Output every segment stored at the current node u (x must intersect them all because they all span $Region(u)$)
    - Note that $Region(u) = Region\big(leftchile(u)\big) \cup Region\big(rightchild(u)\big)$.
  - Go deep:
    - If x falls into $Region\big(leftchild(u)\big)$, take the left branch
    - If x falls into $Region\big(rightchild(u)\big)$, take the right branch



Vertical segment query on segment tree

- Find segments that intersect a given segment
  - Segments stored at u all span $Region(u)$ by definition
  - The segments can be linearly ordered from top to bottom
    - By assuming segments don't overlap
  - Index them in binary search tree by this ordering
  - Range query between y_1 and y_2



Question 4

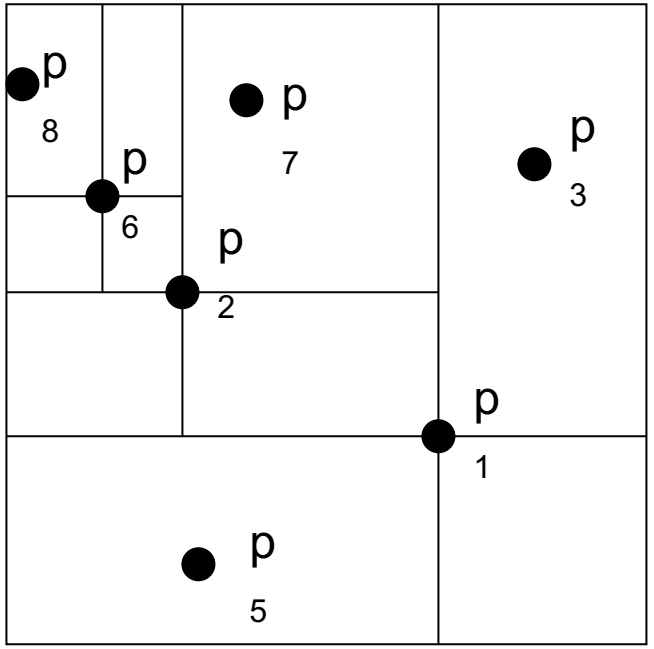
- Quadtree is a tree structure in which each internal node has exactly four children. In a quadtree, each node represents a bounding box covering some part of the space being indexed, with the root node covering the entire area.
  - Is quadtree a balanced tree?
  - What is the worst case of the two quadtrees?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

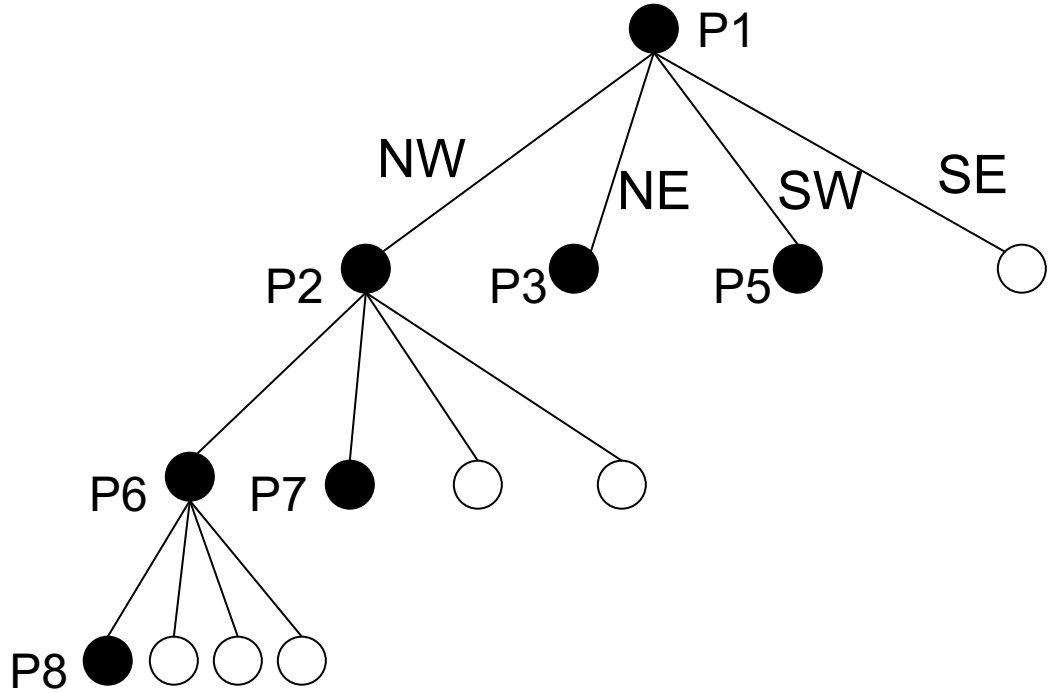
Question 4 (Answer)

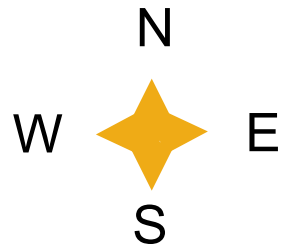
- Quadtree is a tree structure in which each internal node has exactly four children. In a quadtree, each node represents a bounding box covering some part of the space being indexed, with the root node covering the entire area.
  - Is quadtree a balanced tree?
    - Quad tree is not a balanced tree. 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.
  - What is the worst case of the two quadtrees?
    - 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.

Point Quadtree Construction

- Insertion
  - Random insertion roughly $N \log_4 N$



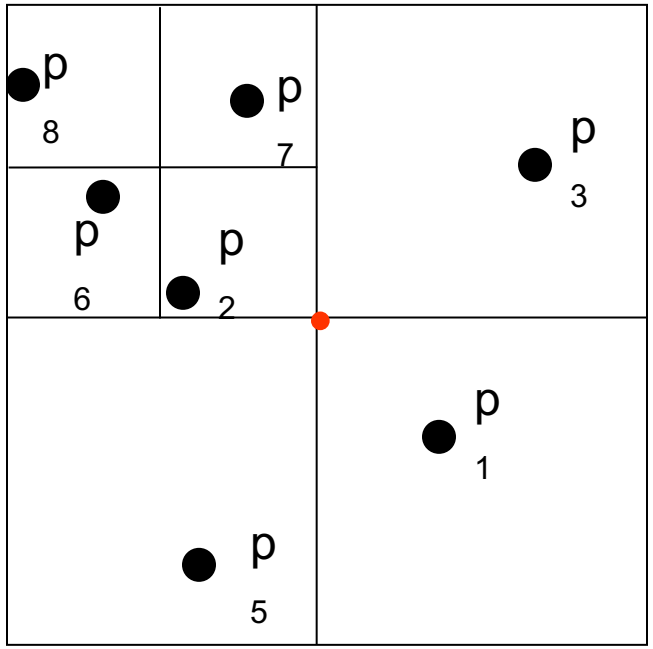


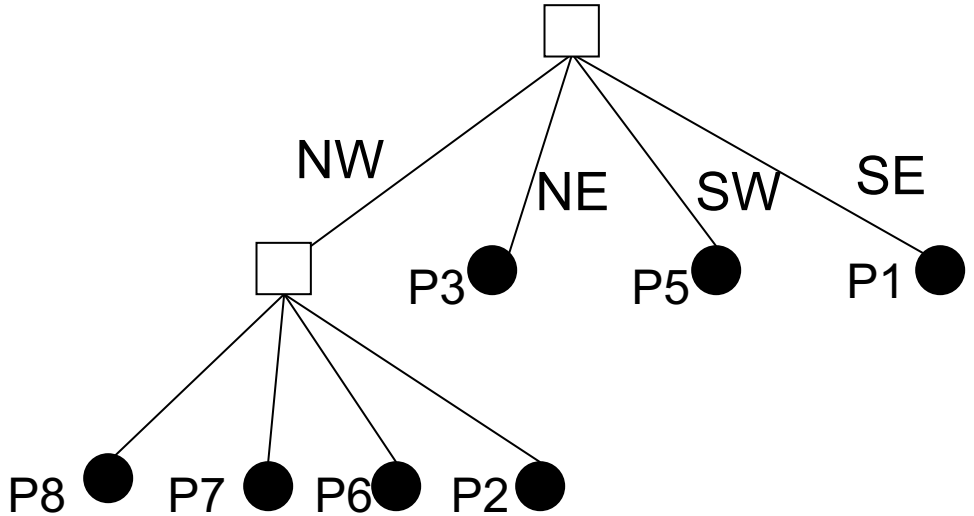


- When is the worst case?
  - Insertion takes $N(N-1)/2$

Region Quadtree Construction

- PR (Point-Region) Quadtree





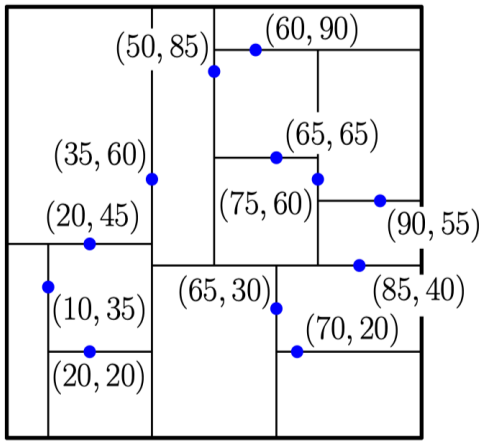
- Based on regular decomposition of the universe
  - Recursively decomposing a region into four congruent blocks
  - Only leaves contain data

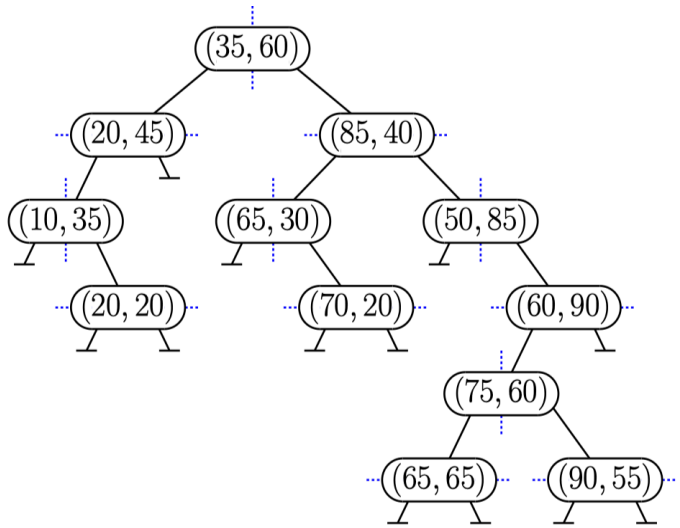
Question 4 (Answer)

- What's the main difference between Point Quadtree and Region Quadtree?
  - 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

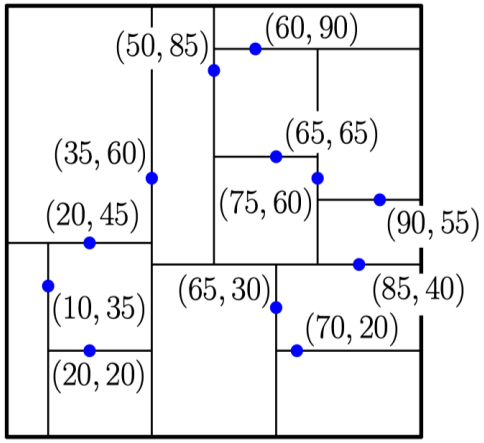
- Consider the kd-Tree shown below. Assume that the cutting dimensions alternate between x and y.
  - 1) Show the result of inserting (80,10) into this tree.
  - 2) Show the kd-Tree that results after deleting the point (35,60) from the original tree. Show both the tree structure and the subdivision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

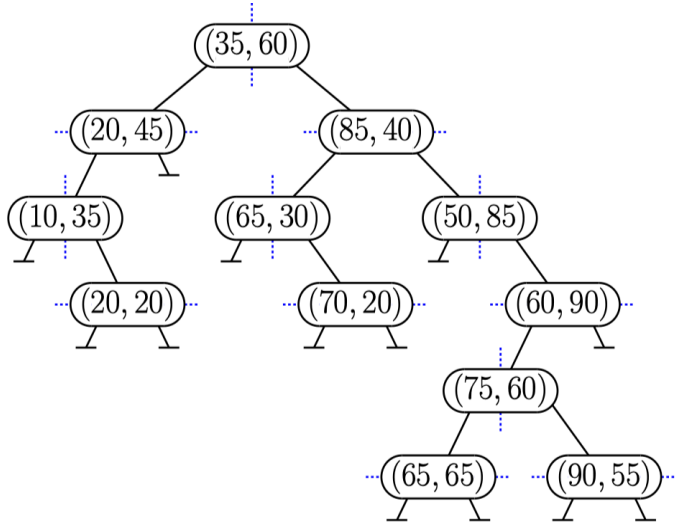


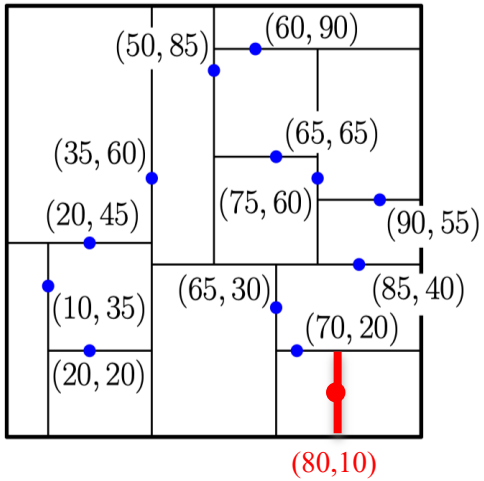


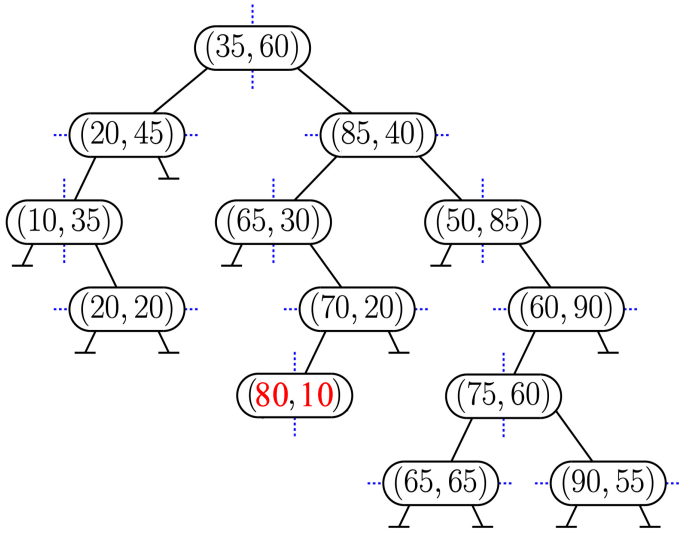
Question 5 (Answer)

- 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.







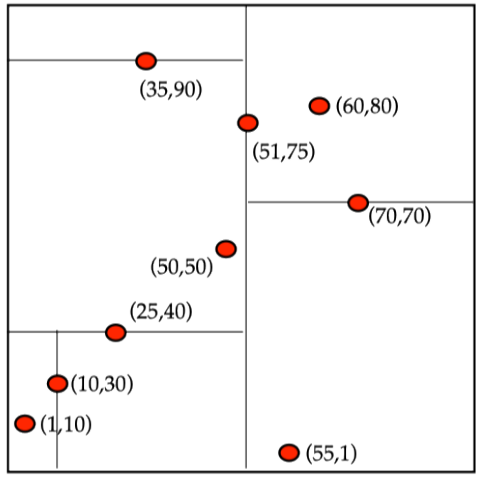


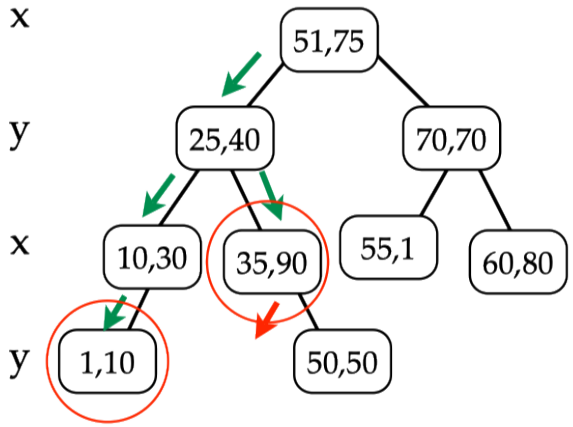
kd-Tree – FindMin

- $FindMin(d)$
  - Find the point with the smallest value in the d^{th} dimension
  - If $cutDim(node) = d$, then the minimum can't be in the right subtree
    - If no left subtree, the current node is the min for the tree rooted at this node
    - Otherwise, recurse on the left subtree
  - If $cutDim(node) \neq d$, then the minimum could be in either subtree
    - Recurse on both subtrees
      - Often have to explore several paths down the tree

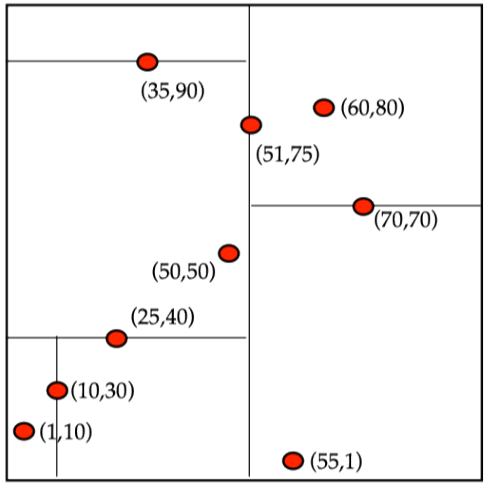
kd-Tree – FindMin

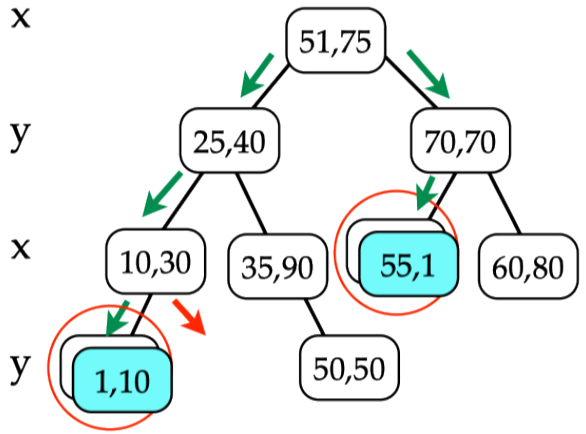
- $FindMin(x)$





- $FindMin(y)$



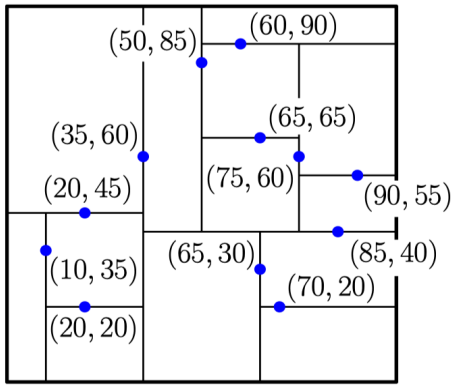


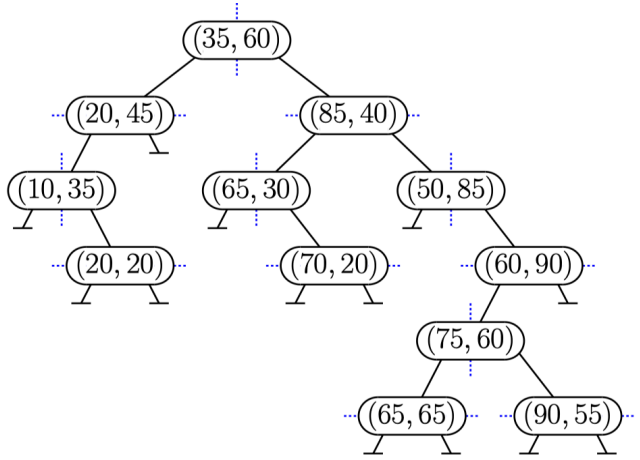
kd-Tree – Deletion

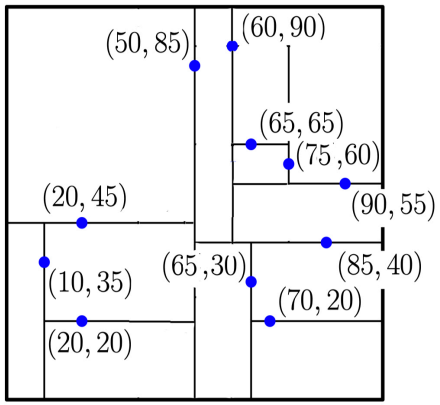
- To remove a node on a level with discriminator along dimension j (suppose $<$ go to left and $\geq$ go to right)
  - If the node is a leaf, just remove it
  - Else if node has right subtree
    - Find the j-minimum node in the right subtree
      - Replace node with j-minimum node and repeat until you reach a leaf, then remove the leaf
  - Else find the j-maximum node in the left subtree, replace, repeat, remove ?
    - This will cause problems if there are duplicate coordinates in p's left subtree
      - Compute the j-minimum from left subtree as replacement
      - Make left subtree the new right subtree

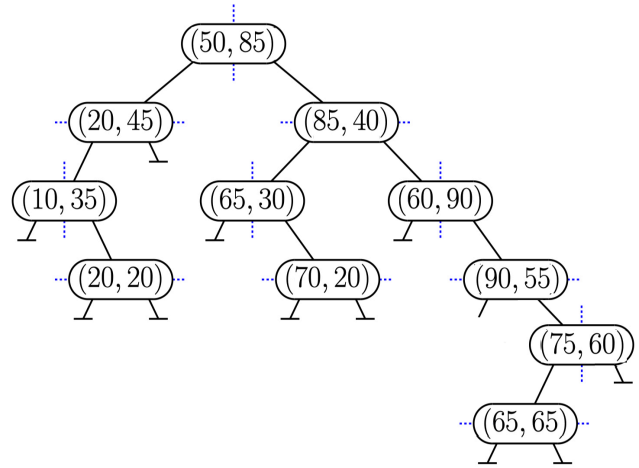
+ Question 5 (Answer)

- The deleted node is at the root. Since the root node is an x-splitter, its replacement is the node with the smallest x-coordinate in the root's right child, which is (50,85).
- (50,85) is an x-splitter, its replacement is the node with the smallest x-coordinate in its right subtree, which is (60,90).
- (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.









Thank You!