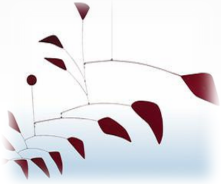
EEECS 4101-5101
Advanced Data Structures



Shahin Kamali

Topic 3 - Multidimensional Dictionaries

York University

Picture is from the cover of the textbook CLRS.

Multi-Dimensional Data

- Various applications
  - Attributes of a product (laptop: price, screen size, processor speed, RAM, hard drive,$\cdots$)
  - Attributes of an employee (name, age, salary,$\cdots$)

Multi-Dimensional Data

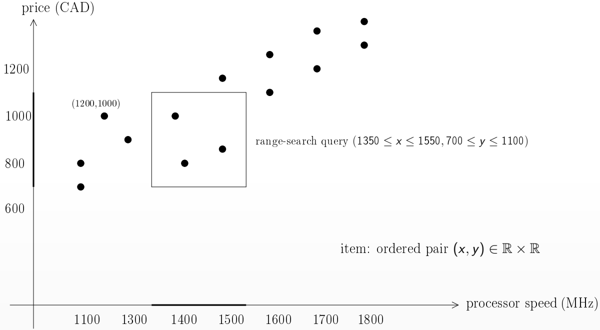
- Various applications
    - Attributes of a product (laptop: price, screen size, processor speed, RAM, hard drive,$\cdots$)
    - Attributes of an employee (name, age, salary,$\cdots$)
- Dictionary for multi-dimensional data
  A collection of d-dimensional items
  Each item has d aspects (coordinates): $(x_0, x_1, \cdots, x_{d-1})$
  Operations: insert, delete, range-search query

Multi-Dimensional Data

- Various applications
  - Attributes of a product (laptop: price, screen size, processor speed, RAM, hard drive,$\cdots$)
  - Attributes of an employee (name, age, salary,$\cdots$)
- Dictionary for multi-dimensional data
  A collection of d-dimensional items
  Each item has d aspects (coordinates): $(x_0, x_1, \cdots, x_{d-1})$
  Operations: insert, delete, range-search query
- (Orthogonal) Range-search query: specify a range (interval) for certain aspects, and find all the items whose aspects fall within given ranges.
  Example: laptops with screen size between 11 and 13 inches, RAM between 8 and 16 GB, price between 1,500 and 2,000 CAD

Multi-Dimensional Data

- Each item has d aspects (coordinates): $(x_0, x_1, \cdots, x_{d-1})$
- Aspect values (x_i) are numbers
- Each item corresponds to a point in d-dimensional space
- We concentrate on $d = 2$, i.e., points in Euclidean plane



One-Dimensional Range Search

- First solution: ordered arrays
    - Running time: $O(\log n + k)$, k: number of reported items
    - Problem: does not generalize to higher dimensions

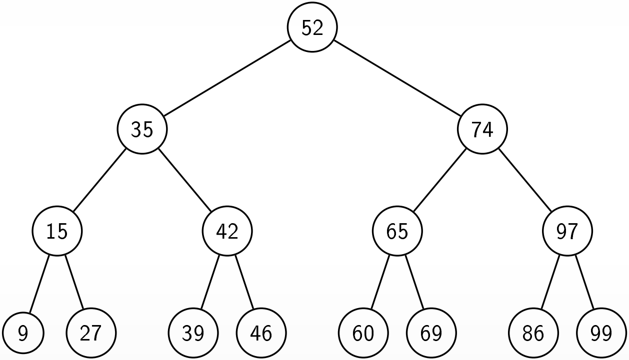
One-Dimensional Range Search

- First solution: ordered arrays
  - Running time: $O(\log n + k)$, k: number of reported items
  - Problem: does not generalize to higher dimensions
- Second solution: balanced BST (e.g., AVL tree)

```
$BST\text{-}RangeSearch(T, k_1, k_2)$
$T$: A balanced search tree, $k_1, k_2$: search keys
Report keys in $T$ that are in range $[k_1, k_2]$
    if $T = nil$ then return
    if $key(T) < k_1$ then
        BST-RangeSearch($T.right, k_1, k_2$)
    if $key(T) > k_2$ then
        BST-RangeSearch($T.left, k_1, k_2$)
    if $k_1 \le key(T) \le k_2$ then
        BST-RangeSearch($T.left, k_1, k_2$)
        report $key(T)$
        BST-RangeSearch($T.right, k_1, k_2$)
```

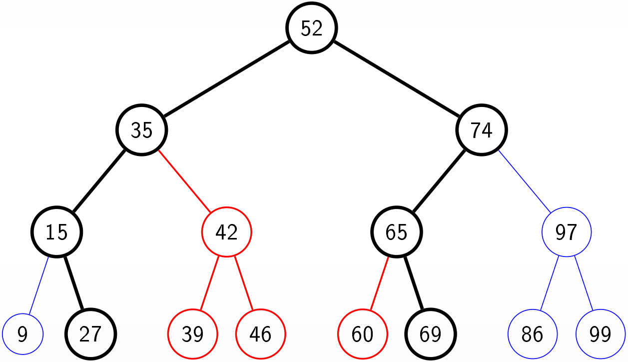
Range Search example

BST-RangeSearch$(T, 30, 65)$



Range Search example

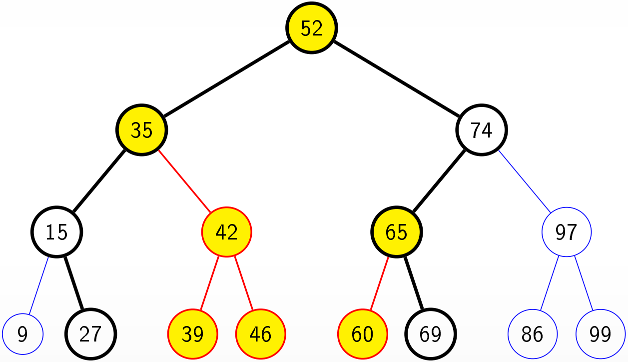
$BST\text{-}RangeSearch(T, 30, 65)$
Nodes either on boundary, inside, or outside.



Range Search example

BST-RangeSearch$(T, 30, 65)$

Nodes either on boundary, inside, or outside.



Note: Not every boundary node is returned.

One-Dimensional Range Search

- P_1: path traversed in $\mathit{BST\text{-}Search}(T, k_1)$
- P_2: path traversed in $\mathit{BST\text{-}Search}(T, k_2)$
- Partition nodes of T into three groups:
    1. boundary nodes: nodes in P_1 or P_2
    2. inside nodes: non-boundary nodes that belong to either (a subtree rooted at a right child of a node of P_1) or (a subtree rooted at a left child of a node of P_2)
    3. outside nodes: non-boundary nodes that belong to either (a subtree rooted at a left child of a node of P_1) or (a subtree rooted at a right child of a node of P_2)

One-Dimensional Range Search

- P_1: path traversed in $BST\text{-}Search(T, k_1)$
- P_2: path traversed in $BST\text{-}Search(T, k_2)$
- k: number of reported items
- Nodes visited during the search:
  - $O(\log n)$ boundary nodes
  - $O(k)$ inside nodes
  - No outside nodes
- Running time $O(\log n + k)$

2-Dimensional Range Search

- Each item has 2 aspects (coordinates): (x_i, y_i)
- Each item corresponds to a point in Euclidean plane
- Options for implementing d-dimensional dictionaries:
  - Reduce to one-dimensional dictionary: combine the d-dimensional key into one key

2-Dimensional Range Search

- Each item has 2 aspects (coordinates): (x_i, y_i)
- Each item corresponds to a point in Euclidean plane
- Options for implementing d-dimensional dictionaries:
    - Reduce to one-dimensional dictionary: combine the d-dimensional key into one key
      Problem: Range search on one aspect is not straightforward

2-Dimensional Range Search

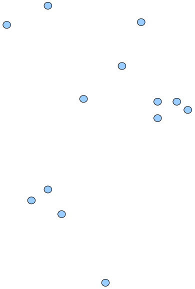
- Each item has 2 aspects (coordinates): (x_i, y_i)
- Each item corresponds to a point in Euclidean plane
- Options for implementing d-dimensional dictionaries:
  - Reduce to one-dimensional dictionary: combine the d-dimensional key into one key
    Problem: Range search on one aspect is not straightforward
  - Use several dictionaries: one for each dimension
    Problem: inefficient, wastes space
  - Partition trees
    - A tree with n leaves, each leaf corresponds to an item
    - Each internal node corresponds to a region
    - quadtrees, kd-trees
  - multi-dimensional range trees

Quadtrees

- We have n points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1})\}$ in the plane
- How to build a quadtree on P:
    - Find a square R that contains all the points of P (We can compute minimum and maximum x and y values among n points)
    - Root of the quadtree corresponds to R
    - Split: Partition R into four equal subsquares (quadrants), each correspond to a child of R
    - Recursively repeat this process for any node that contains more than one point
    - Points on split lines belong to left/bottom side
    - Each leaf stores (at most) one point
    - We can delete a leaf that does not contain any point

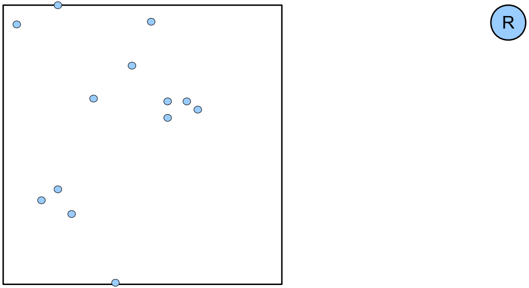
Quadtrees

- Example: We have 13 points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{12}, y_{12})\}$ in the plane



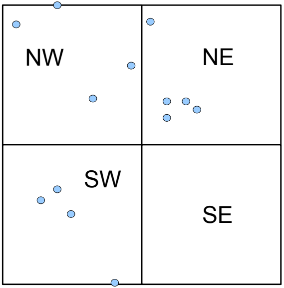
Quadtrees

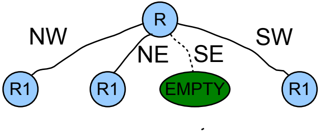
- Example: We have 13 points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{12}, y_{12})\}$ in the plane



Quadtrees

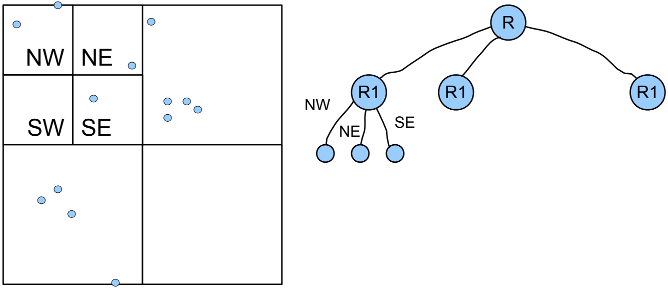
- Example: We have 13 points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{12}, y_{12})\}$ in the plane





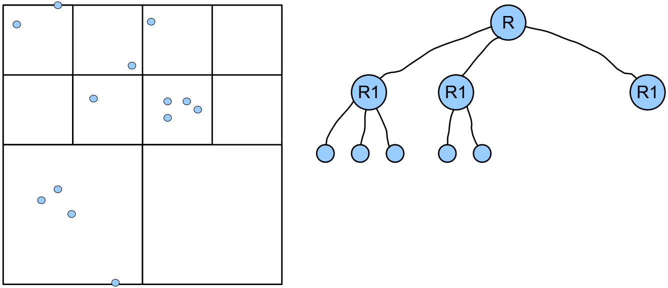
Quadtrees

- Example: We have 13 points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{12}, y_{12})\}$ in the plane



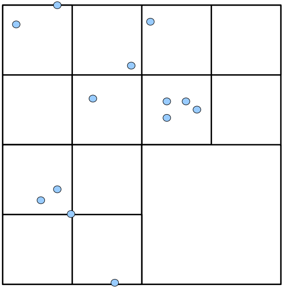
Quadtrees

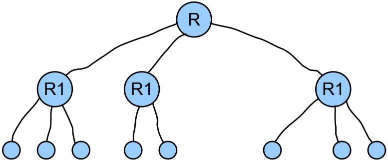
- Example: We have 13 points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{12}, y_{12})\}$ in the plane



Quadtrees

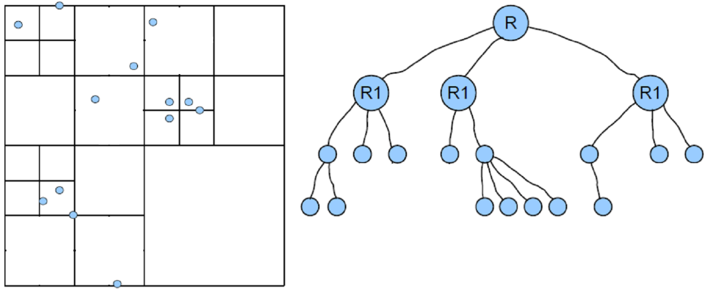
- Example: We have 13 points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{12}, y_{12})\}$ in the plane





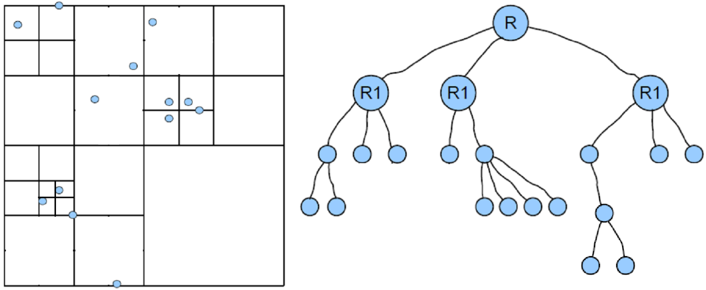
Quadtrees

- Example: We have 13 points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{12}, y_{12})\}$ in the plane



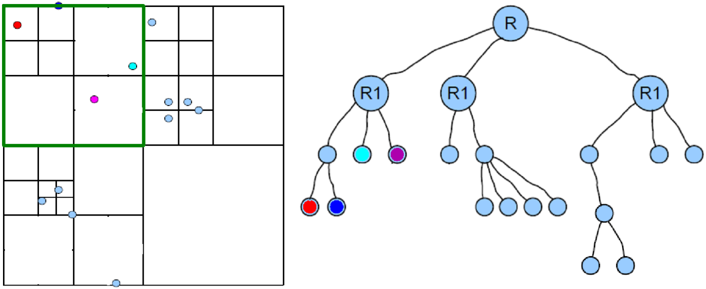
Quadtrees

- Example: We have 13 points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{12}, y_{12})\}$ in the plane



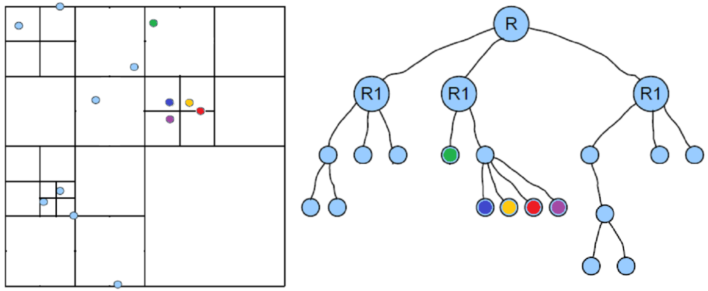
Quadtrees

- Example: We have 13 points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{12}, y_{12})\}$ in the plane



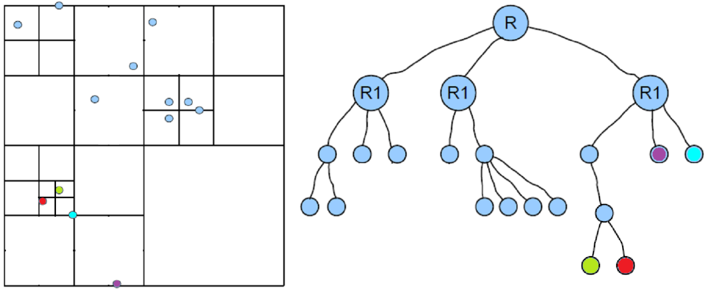
Quadtrees

- Example: We have 13 points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{12}, y_{12})\}$ in the plane



Quadtrees

- Example: We have 13 points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{12}, y_{12})\}$ in the plane



Quadtree Operations

- Search: Analogous to binary search trees
- Insert:
    - Search for the point
    - Split the leaf if there are two points
- Delete:
    - Search for the point
    - Remove the point
    - Walk back up in the tree to discard unnecessary splits

Quadtree: Range Search

```
QTree-RangeSearch($T$, $R$)
$T$: A quadtree node, $R$: Query rectangle
    if ($T$ is a leaf) then
        if ($T.point \in R$) then
            report $T.point$
    for each child $C$ of $T$ do
        if $C.region \cap R \neq \emptyset$ then
            QTree-RangeSearch($C$, $R$)
```

- Complexity of range search: $\Theta(n + h)$ even if the answer is $\emptyset$
- spread factor of points P : $\beta(P) = d_{max}/d_{min}$
- $d_{max}(d_{min})$: maximum (minimum) distance between two points in P
- height of quadtree: $h \in \Theta(\log_2 \frac{d_{max}}{d_{min}})$
- Complexity to build initial tree: $\Theta(nh)$

Quadtree Conclusion

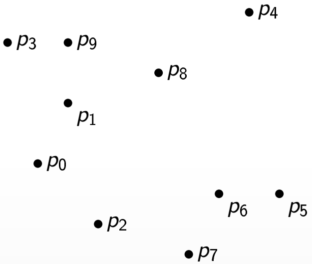
- Very easy to compute and handle
- No complicated arithmetic, only divisions by 2 (usually the boundary box is padded to get a power of two).
- Space wasteful
- Major drawback: can have very large height for certain nonuniform distributions of points
- Easily generates to higher dimensions (octrees, **etc.**).

kd-trees

- We have n points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1})\}$
- Quadtrees split square into quadrants regardless of where points actually lie
- kd-tree idea: Split the points into two (roughly) equal subsets
- How to build a kd-tree on P:
    - Split P into two equal subsets using a vertical line
    - Split each of the two subsets into two equal pieces using horizontal lines
    - Continue splitting, alternating vertical and horizontal lines, until every point is in a separate region
- Complexity: $\Theta(n \log n)$, height of the tree: $\Theta(\log n)$

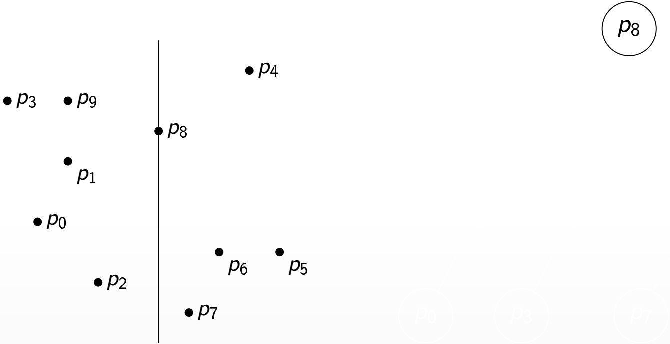
kd-trees

- kd-tree idea: Split the points into two (roughly) equal subsets
- A balanced binary tree



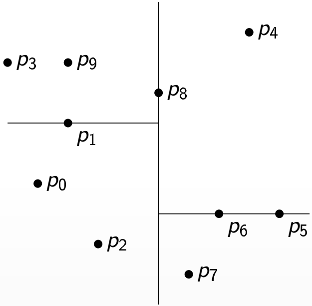
kd-trees

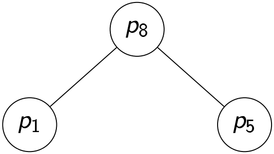
- kd-tree idea: Split the points into two (roughly) equal subsets
- A balanced binary tree



kd-trees

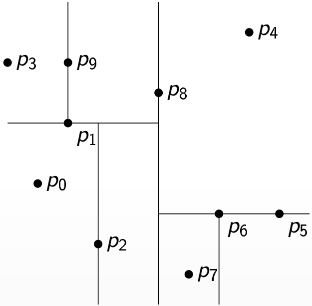
- kd-tree idea: Split the points into two (roughly) equal subsets
- A balanced binary tree

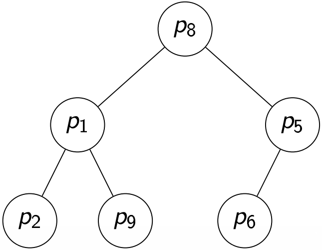




kd-trees

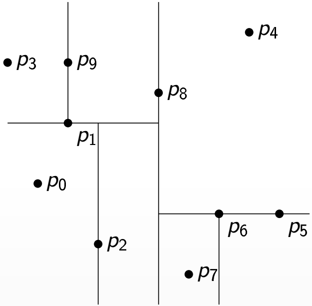
- kd-tree idea: Split the points into two (roughly) equal subsets
- A balanced binary tree

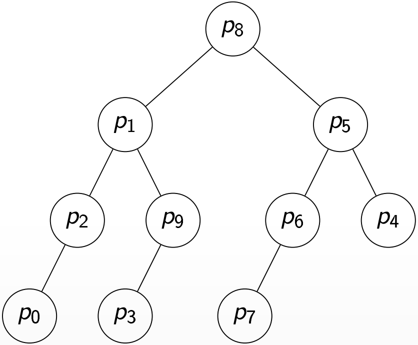




kd-trees

- kd-tree idea: Split the points into two (roughly) equal subsets
- A balanced binary tree





KD-tree Construction (details)

- Initialize:
  - Sort the n points according to their x-coordinates and store in X $\rightarrow O(n \log n)$
  - Sort the n points according to their y-coordinates and store in Y $\rightarrow O(n \log n)$

KD-tree Construction (details)

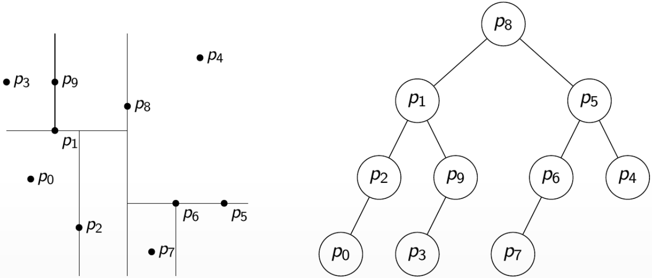
- Initialize:
    - Sort the n points according to their x-coordinates and store in X $\rightarrow O(n \log n)$
    - Sort the n points according to their y-coordinates and store in Y $\rightarrow O(n \log n)$
- Recursive process on X, Y:
    - The root of the tree is the point with median x-coordinate (index $i = \lfloor n/2 \rfloor$ in the sorted list X)
    - Let $X_1 = X[1..i-1]$, $X_2 = X[i+1..n]$
    - Partition Y to Y_1, Y_2 such that $Y_1 = X_1$ and $Y_2 = X_2$ (but in the sorted order according to y-coordinate).
    - Recurs on (X_1, Y_1) and (X_2, Y_2) but alternate the coordinate; these form the left and right subtrees.

KD-tree Construction (details)

- Initialize:
    - Sort the n points according to their x-coordinates and store in X $\rightarrow O(n \log n)$
    - Sort the n points according to their y-coordinates and store in Y $\rightarrow O(n \log n)$
- Recursive process on X, Y:
    - The root of the tree is the point with median x-coordinate (index $i = \lfloor n/2 \rfloor$ in the sorted list X)
    - Let $X_1 = X[1..i-1]$, $X_2 = X[i+1..n]$
    - Partition Y to Y_1, Y_2 such that $Y_1 = X_1$ and $Y_2 = X_2$ (but in the sorted order according to y-coordinate).
    - Recurs on (X_1, Y_1) and (X_2, Y_2) but alternate the coordinate; these form the left and right subtrees.
- Time analysis:
    - The partitioning takes $O(n)$ time. The recursion is $T(n) = 2T(n/2) + O(n)$ which solves as $T(n) = \Theta(n \log n)$. The initialization is also $\Theta(n \log n)$.

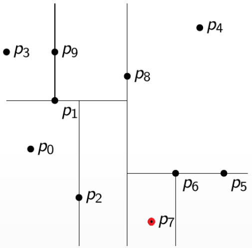
KD-tree Operations

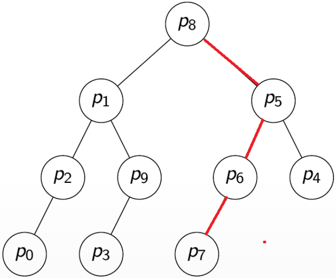
- Search: as in any binary search tree (check x or y coordinate on even/odd levels of the tree, respectively).
- Insert, Delete: not hard. However they may leave the tree unbalanced. Therefore we need to do (periodic) rebalancing.
    - A rotation is equivalent to shifting a vertical line to its left/right point or shifting the horizontal line to its below/above point.



KD-tree Operations

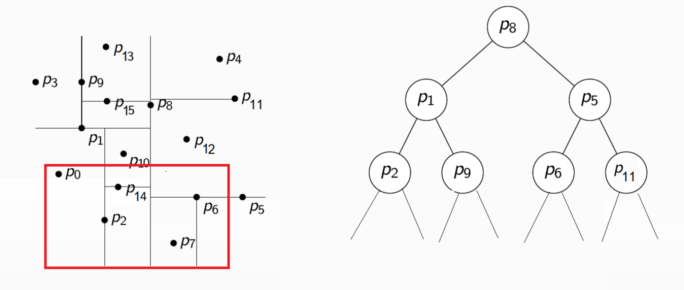
- Search: as in any binary search tree (check x or y coordinate on even/odd levels of the tree, respectively).
- Insert, Delete: not hard. However they may leave the tree unbalanced. Therefore we need to do (periodic) rebalancing.
  - A rotation is equivalent to shifting a vertical line to its left/right point or shifting the horizontal line to its below/above point.





kd-tree: Range Search Complexity

- The complexity is $O(k + U)$ where k is the number of keys reported and U is the number of regions (tree nodes) we go to but unsuccessfully
    - Here, $U = \{p_8, p_1, p_{10}, p_5, p_{11}, p_{12}\}$
    - U corresponds to the number of regions which intersect but are not fully in R
    - Those regions have to intersect one of the four sides of R

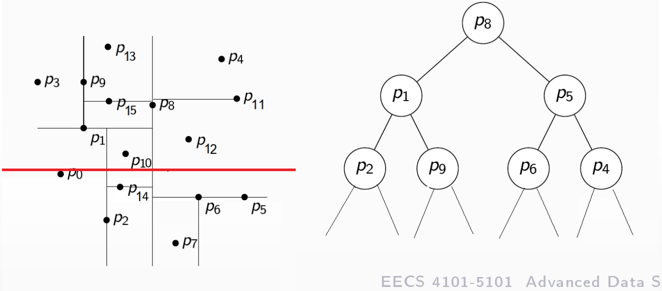


kd-tree: Range Search Complexity

- $Q(n)$: Maximum number of regions in a kd-tree with n points that intersect a vertical (horizontal) line
- $Q(n)$ satisfies the following recurrence relation:

$$Q(n) = 2Q(n/4) + O(1)$$

- It solves to $Q(n) = O(\sqrt{n})$
- Therefore, the complexity of range search in kd-trees is $O(k + \sqrt{n})$

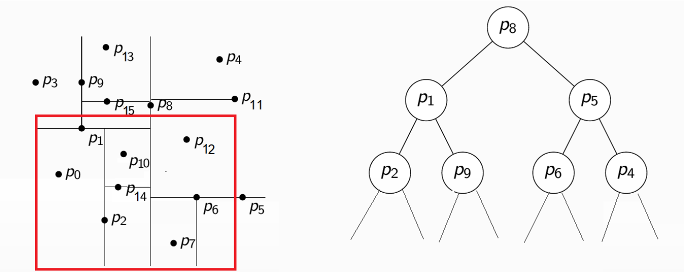


kd-tree: Range Search Complexity

- $Q(n)$: Maximum number of regions in a kd-tree with n points that intersect a vertical (horizontal) line
- $Q(n)$ satisfies the following recurrence relation:

$$Q(n) = 2Q(n/4) + O(1)$$

- It solves to $Q(n) = O(\sqrt{n})$
- Therefore, the complexity of range search in kd-trees is $O(k + \sqrt{n})$



kd-tree: Higher Dimensions

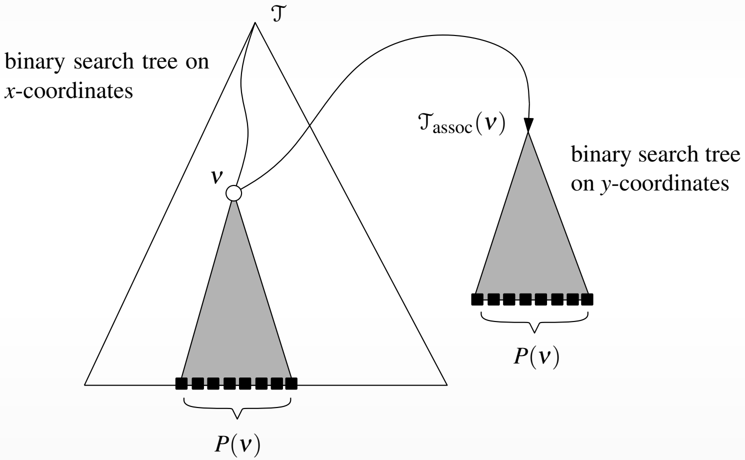
- kd-trees for d-dimensional space
  - At the root the point set is partitioned based on the first coordinate
  - At the children of the root the partition is based on the second coordinate
  - At depth $d-1$ the partition is based on the last coordinate
  - At depth d we start all over again, partitioning on first coordinate
- Storage: $O(n)$
- Construction time: $O(n \log n)$
- Insertion/deletion: $O(\log n)$
- Range query time: $O(n^{1-1/d}+k)$

(Note: d is considered to be a constant.)

Range Trees

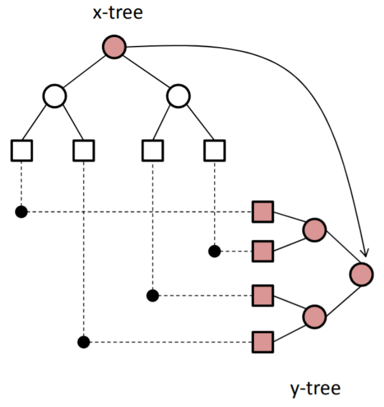
- We have n points $P = \{(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1})\}$ in the plane
- A range tree is a tree of trees (a *multi-level* data structure)
- How to build a range tree on P:
    - Build a balanced binary search tree τ determined by the x-coordinates of the n points
    - For every node $v \in \tau$, build a balanced binary search tree $\tau_{assoc}(v)$ (associated structure of τ) determined by the y-coordinates of the nodes in the subtree of τ with root node v

Range Tree Structure



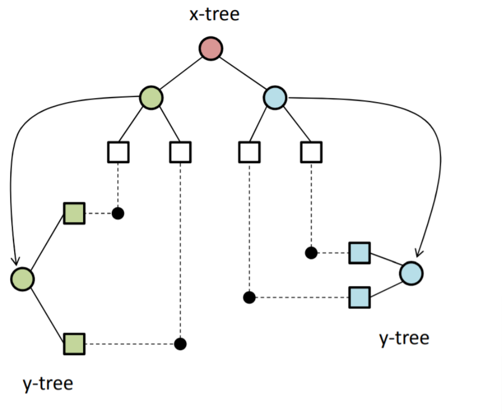
Range Tree Structure

- There is one **primary tree** (x-tree) and n **secondary trees** (y-trees).



Range Tree Structure

- There is one **primary tree** (x-tree) and n **secondary trees** (y-trees).



Range Trees: Size & Construction

- The size of the primary tree is $O(n)$ for n points.

Range Trees: Size & Construction

- The size of the primary tree is $O(n)$ for n points.
- Each point x is associated with a leaf in the primary tree τ.
  - From the leaf to the root of τ, the secondary tree associated with any node includes x.

Range Trees: Size & Construction

- The size of the primary tree is $O(n)$ for n points.
- Each point x is associated with a leaf in the primary tree τ.
    - From the leaf to the root of τ, the secondary tree associated with any node includes x.
    - So, x is present in $\Theta(\log n)$ secondary trees.
    - Over all points, all secondary trees take $\Theta(n \log n)$ space.

Range Trees: Size & Construction

- The size of the primary tree is $O(n)$ for n points.
- Each point x is associated with a leaf in the primary tree τ.
  - From the leaf to the root of τ, the secondary tree associated with any node includes x.
  - So, x is present in $\Theta(\log n)$ secondary trees.
  - Over all points, all secondary trees take $\Theta(n \log n)$ space.
- **A 2d-range tree of n points take $O(n \log n)$ space**
  - It is slightly more than $O(n)$ of kd-trees.

Range Trees: Size & Construction

- The size of the primary tree is $O(n)$ for n points.
- Each point x is associated with a leaf in the primary tree τ.
    - From the leaf to the root of τ, the secondary tree associated with any node includes x.
    - So, x is present in $\Theta(\log n)$ secondary trees.
    - Over all points, all secondary trees take $\Theta(n \log n)$ space.
- **A 2d-range tree of n points take $O(n \log n)$ space**
    - It is slightly more than $O(n)$ of kd-trees.
- **A 2d-range tree of n points can be constructed in $O(n \log n)$**
    - Details omitted here.

Range Trees: Operations

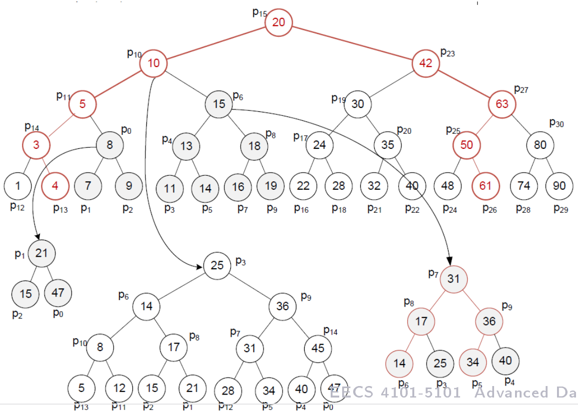
- Search: trivially as in a binary search tree
- Insert: insert point in the primary tree $\mathcal{T}$ by x-coordinate
    - From inserted leaf, walk back up to the root and insert the point in all associated trees $\mathcal{T}_{assoc}(v)$ of nodes v on path to the root.
- Delete: analogous to insertion.
- Note: re-balancing is a problem.

Range Trees: Range Search

- To perform a range search query $R = [x_1, x_2] \times [y_1, y_2]$:
  - Perform a range search (on the x-coordinates) for the interval $[x_1, x_2]$ in τ ($BST\text{-}RangeSearch(\tau, x_1, x_2)$)
  - For every outside node, do nothing.
  - For every "top" inside node v, perform a range search (on the y-coordinates) for the interval $[y_1, y_2]$ in $\tau_{assoc}(v)$. During the range search of $\tau_{assoc}(v)$, do not check x-coordinates (they are within range).
  - For every boundary node, test to see if the corresponding point is within the region R.

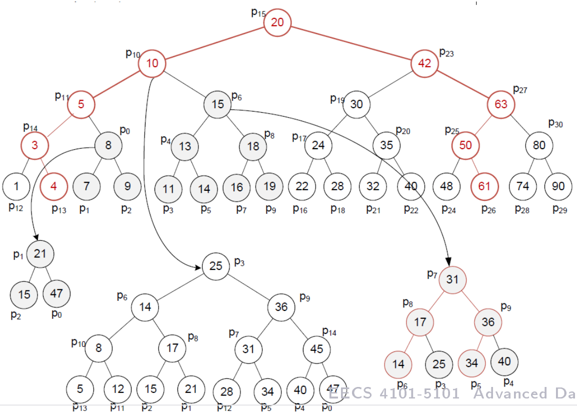
Range Trees: Range Search

- E.g., range search $[4, 59] \times [14, 35]$
  - Search for range $[4, 59]$ in τ. Do nothing for outside nodes (e.g., p_{12}, p_{28}).
  - Boundary nodes are checked individually (e.g., $p_{10} = (10, 8)$ is not in the range).
  - Do a range search on y-coordinate on trees associated with top nodes (e.g., p_6, p_0).



Range Trees: Range Search

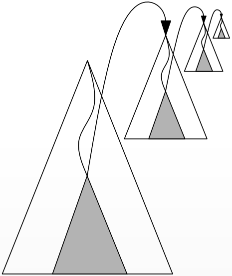
- E.g., range search $[4, 59] \times [14, 35]$
  - Search for range $[4, 59]$ in τ. Do nothing for outside nodes (e.g., p_{12}, p_{28}).
  - Boundary nodes are checked individually (e.g., $p_{10} = (10, 8)$ is not in the range).
  - Do a range search on y-coordinate on trees associated with top nodes (e.g., p_6, p_0).
  - There is an overhead of $O(\log n)$ for 1-dimensional search on each top node, and there are $O(\log n)$ top nodes → $O(\log^2 n)$ time complexity



Range Trees: Higher Dimensions

- Range trees for d-dimensional space
  - Storage: $O(n\,(\log n)^{d-1})$
  - Construction time: $O(n\,(\log n)^{d-1})$
  - Range query time: $O((\log n)^{d} + k)$

(Note: d is considered to be a constant.)



Range Trees: Higher Dimensions

- Space/time trade-off
  - Storage: $O(n\,(\log n)^{d-1})$ kd-trees: $O(n)$
  - Construction time: $O(n\,(\log n)^{d-1})$ kd-trees: $O(n\log n)$
  - Range query time: $O((\log n)^{d} + k)$ kd-trees: $O(n^{1-1/d} + k)$

(Note: d is considered to be a constant.)