Lecture 12: Spatial Partitioning, KD Trees

CS 6017 – Data Analytics and Visualization

MASTER OF SOFTWARE DEVELOPMENT (MSD) PROGRAM

J. DAVISON DE ST. GERMAIN

SUMMER 2023

Lecture 13 – Topics

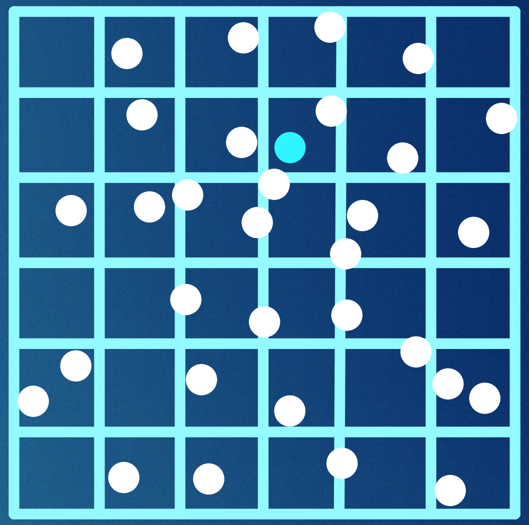
- Spatial Partitioning
  - KD Tree

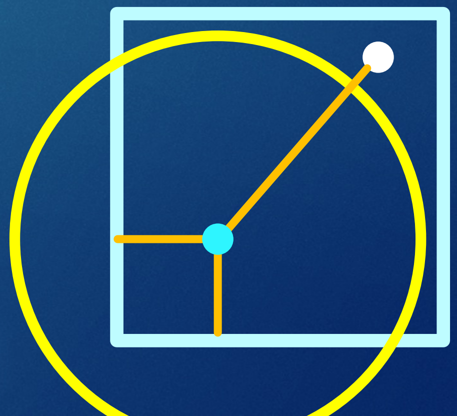
Miscellaneous

- Questions?
- Remember, HW3 is due Tuesday.

Spatial Partitioning – Uniform Grid

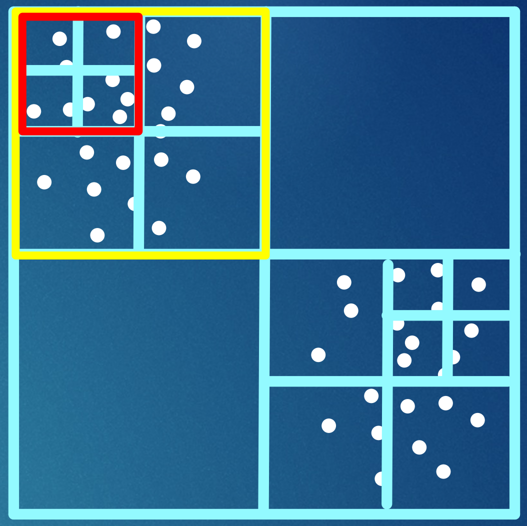
- "Training" −> Put (data / training) points in the correct grid cell.
  - What do the white points represent? Why do we do all this?
- Range Query
  - Find all points within a radius from the test point.
  - Find the cell(s) that overlap the test point / radius and test all points in those cells.
- KNN Query
  - K closest points
  - How to do this?
    - Find query point cell and check all points there.
    - If len(result) < K or distance to worse point > distance to adjacent cell
      - move out one cell in appropriate directions
- The Good: Works great for uniformly distributed points / cache friendly.
- The Bad: Number of cells can become very large as the number of dimensions (of the data) increases.

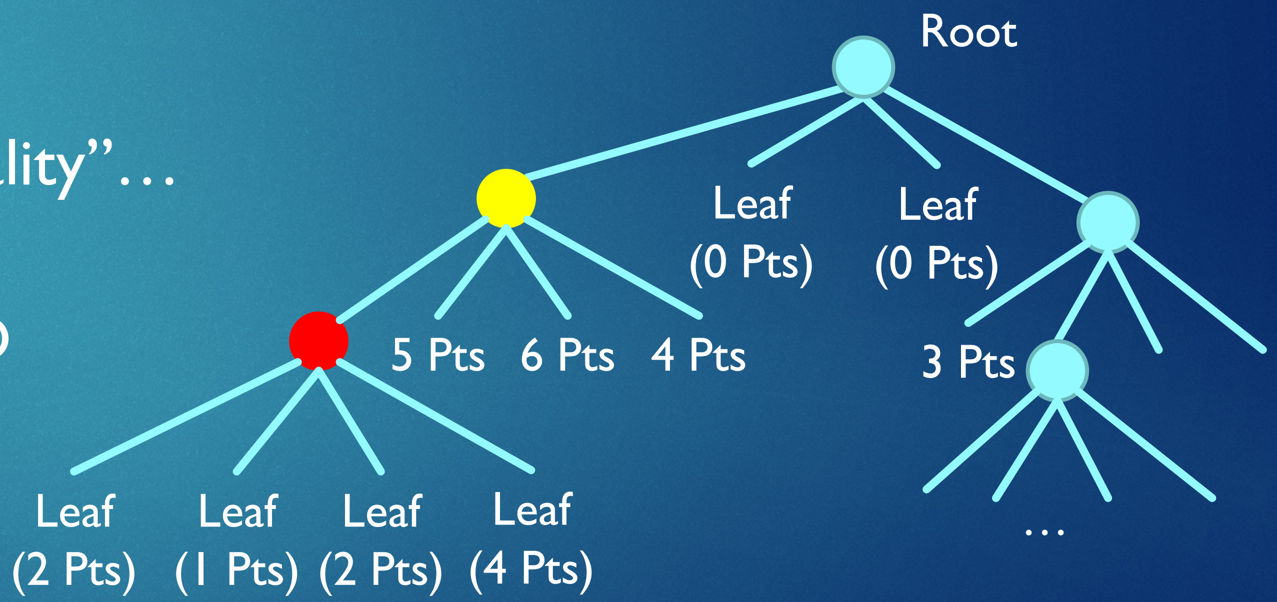




Quad Tree

- Reason for Quad Tree over Uniform Grid?
  - Addresses the “clumped” (non-uniform) data distribution.
- Oct Tree in 3D
- Fix # of splits to 2
- Suffers from the “curse of dimensionality”…
  - # children == 2^{dim}
  - So (usually) only used in 2D and 3D
- Note: Could tweak data structure to split in median of points (instead of just in the middle of the cells) in order to “always” divide the number of points evenly.



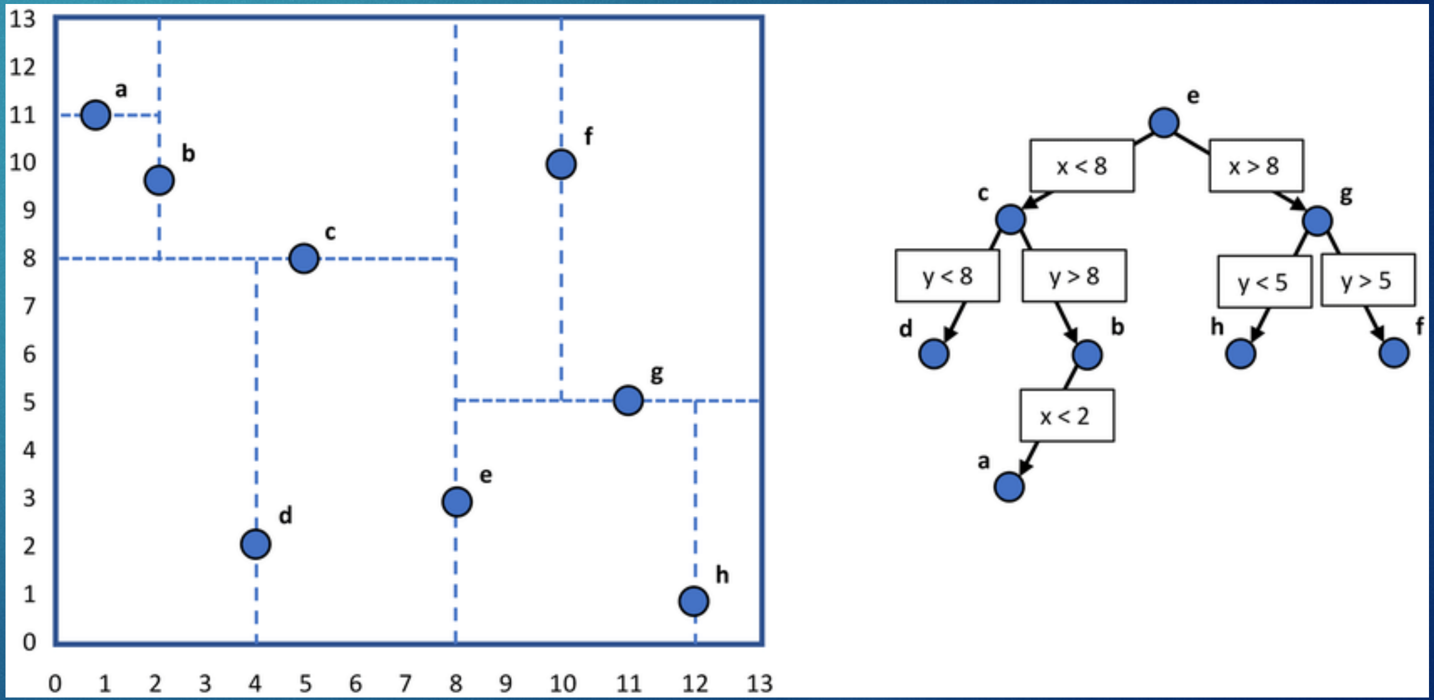


KD Tree (K-Dimension Tree)

- K (here) specifies the number of dimensions
  - Not related to KNN
- A "generalization" of BSTs for multiple dimensions
- KD Tree == BST, where at each level we split using a different *dimension* of the data.

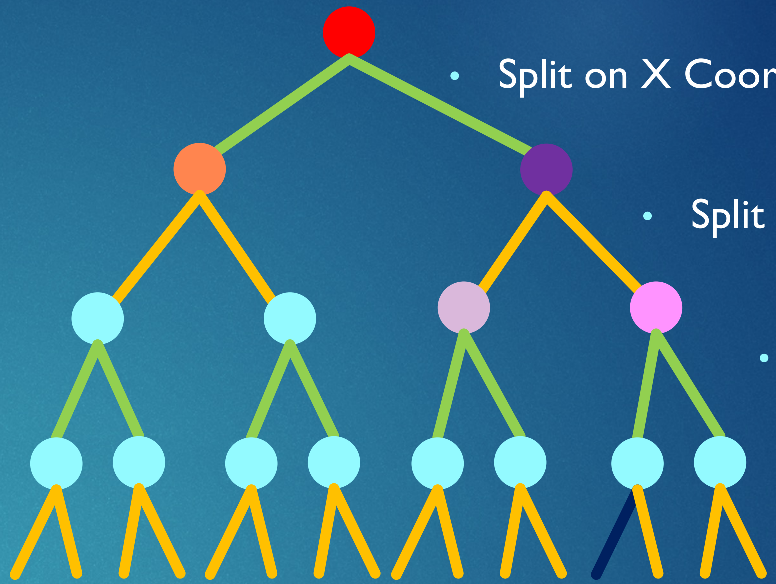
Approach Three: KD Tree

- Generalization of a BST to many "keys"
- Each node stores the median of it's subtree according to one of the dimensions (x, y, z, etc)
- Each time we go down one level, we move to the next dimension (if I split by x, my children split by y)
- Each node stores a point and the dimension it splits by



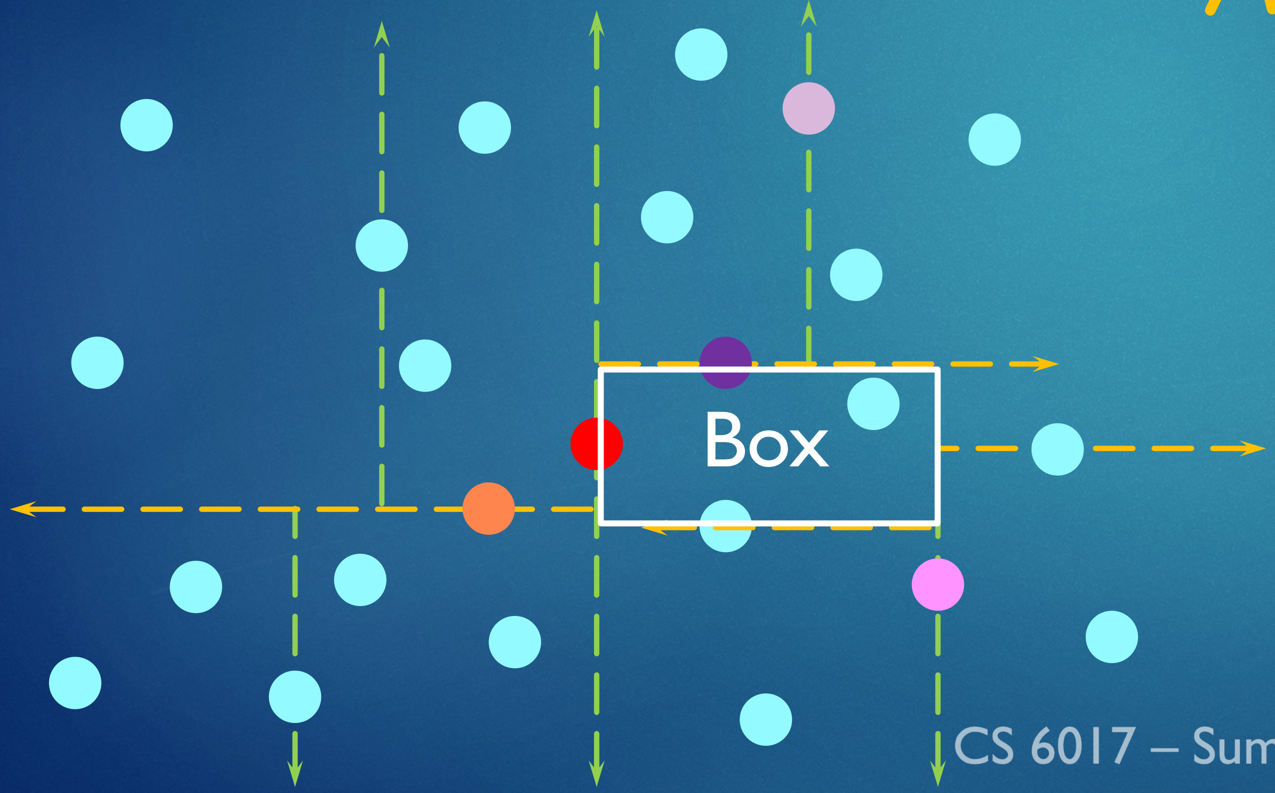
Building the KD Tree

- Right: BST (visualization)
- Left: Spatial Display (visualization)
  - Note: all levels are not complete



- Split on X Coordinate
- Split on Y Coordinate
- Split on X Coordinate
- Split on Y Coordinate

- How is space partitioned using the KD Tree? What are the “shapes” of the children?
  - Each line (chopping of space) creates a new “box” (easier to see at lower levels of the tree)
  - Every time we sub-divide, we are moving in one of the “sides” of the box.
- Height of KNN Tree?
  - It turns out to be a balanced BST
  - log N <– Notice *dimension* not here…
  - However, the bigger the # of dimensions, the larger the “volume” of the (N-dimensional) box



KD Tree Construction

- KdTree(Point[] points)
  - root = new Node(points)
- Data that a Node needs to store?
  - Node left, right
  - Point split_point
  - *Dimension split_dimension*
- XNode
  - YNode left, right
  - Point split_point
- YNode
  - XNode left, right
  - Point split_point

- Advantages
  - Can't accidentally have two X splits in a row.
  - Compiler checks for this.
- Disadvantages (as a S/W Dev?)
  - Duplicating some code…
  - Can fix this by templating the Node
- Node!0 <– XNode
- Node!1 <– YNode
- Node!2 <– ZNode

Node Constructor

- Node(Point [] points)
  - split_point = ?
  - // Find middle point based on split dimension
  - // Store middle point
  - // Group smaller points and bigger points
  - // Create children nodes using (corresponding) points. (Recursing)
  - left = new Node(left half of points)
  - right = new Node(right half of points)
  - // Base case?
    - // length(points) == 1

Finding the Middle Value / Partitioning

- How long does it take to partition a list of numbers into the smaller half and the larger half?
  - Method 1:
    - We know the median value, thus partitioning is:
      - O(n)
  - Method 2: We don't know the median value…
    - Sort: O(n log n)
    - Get median: O(1)
    - Total: O(n log n)
  - Method 3: We don't know the median value…
    - Use a smart person's algorithm… ☺
    - Thus: O(n)

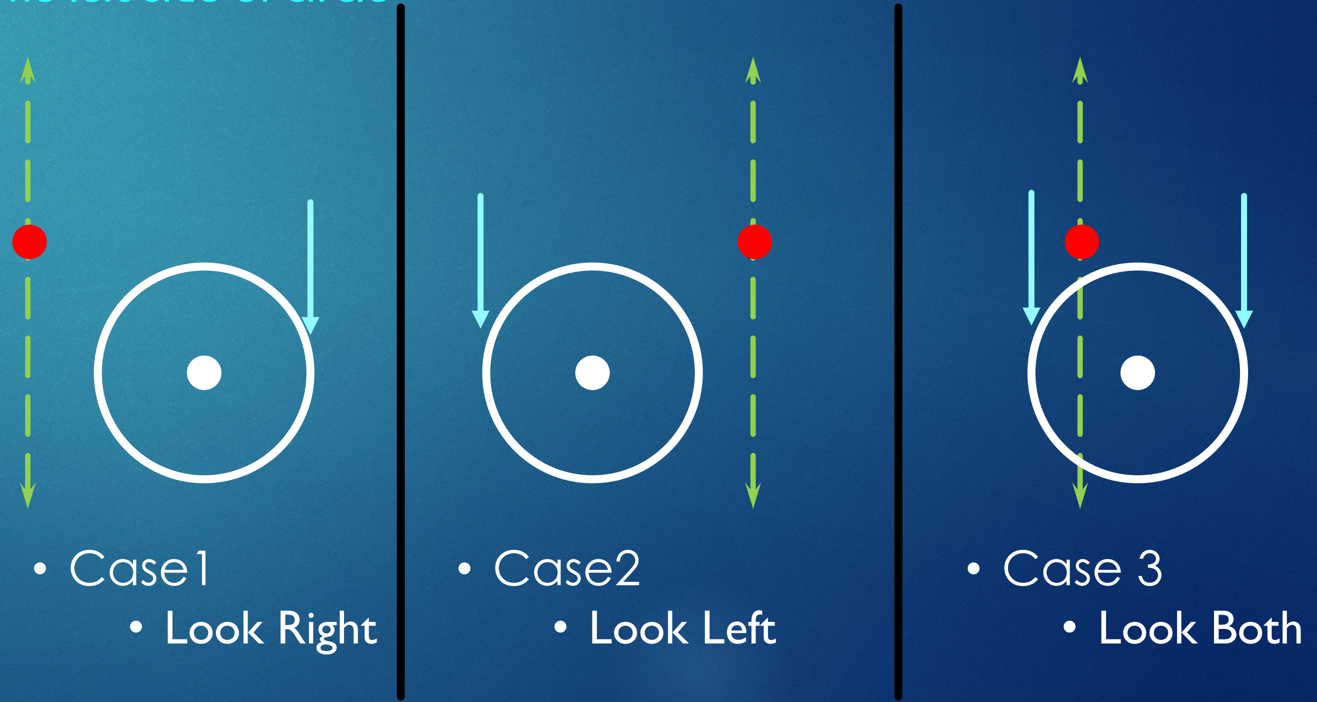
- Quick Median (i-programmer.info)
- Quickselect – Wikipedia
- std::nth_element - cppreference.com
- topN - multiple declarations - D Programming Language (dlang.org)

KD Tree Range Query


```
KdTree::rangeQuery( query_pt, radius )
    return root.rangeQuery( query_pt, radius )
Node::rangeQuery( query_pt, radius )
    if node.p is close to query_pt
        add to list
    if necessary recurse left
    if necessary recurse right
```

Which Children to Recurse Into?

- Three cases… (Split point in red, query point in white)
  - Note: case 3 is covered by handling case 1 and case 2
- Do I need to recurse Right?
  - Look at the point + radius // Right side of circle
- Do I need to recurse Left?
  - if query_point.x - radius < split_point.x // ie: The left side of circle
- Recurse both directions?
  - Handled above.
- Above tests use the X coordinate, but if this is a Y node, we would just use ".y"
  - Note, set up so left is always <.



KNN Query For KD Tree

- Review – What did we do for the Quad Tree?

      for each child

          if   list_size < K or
               closestPointInAABB(pt) is closer than worst in list

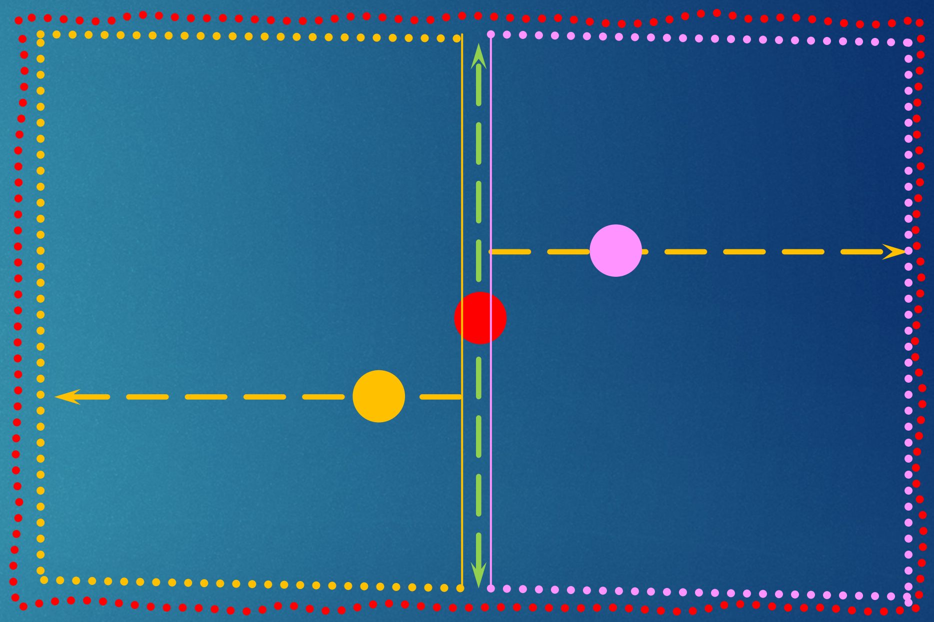
               recurse
- For KD Tree:
  - check point in node
  - // Left Check:
  - if list_size < K or distance from left child to p is closer then the worst point in list
    - recurse left
- Need to know the bounding box of left child
  - Could store, or (easier) can compute as we traverse (see next slide)

KNN Query

KdTree::KnnQuery(query_pt, K)

return root.KnnQuery(query_pt, K, box)

- What is the bounding box of a (sub)tree?
- Root:
  - box == infinite AABB
  - Left Child?
    - left_aabb = parent's aabb
    - left_aabb.right = root.x
  - Right Child
    - right_aabb.left = root.x
  - Note: we might denote "left" as x_min, etc…
  - "Cutting off" (making smaller) one side of the box each time we recurse to a child node.



HW 4 – Spatial Data Structures

- We provide the uniform grid implementation as an example for you.
- You will implement Quad Tree and KD Tree.
- We will be doing timing studies on our implementations.
- Going to implement this in "D". [Ben's favorite language]
- Providing:
  - bucket_knn.d – Implementation of uniform grid
  - dumb_knn.d – A very naïve implementation of KNN with no grid.
  - common.d – A lot of general utilities.
- Look at unit test blocks for some examples.
- Two-week assignment – aim to have data structures implemented by week 1, then use week 2 to do analysis.

~ Fin ~