CSE 250: Spatial Indexing (contd.)

Lecture 35

Nov 29, 2023

Reminders

- PA3 Implementation due Sun, Dec 3
- Course Evals Bonus
  - Get to 90% completion across all 3 sections, we'll release an exam question.
  - More details to be posted on Piazza.

Spatial Indexes

- Datasets of **many** elements
  - Celestial Bodies
  - Molecules
  - 3D Mesh Cells
- The elements are **organized spatially**

What questions do we want to ask?

- What elements (planets, molecules, etc. . .) are close to each other?
- Which elements will a ray of light bounce off of / will a projectile hit?
- What elements are closest to a given point?
- What elements fall within a given range?

How can we organize the elements in a way that allows us to efficiently answer these questions?

Organizing elements in 2D/3D space

What data structures have we seen already that let us efficiently organize/store "sorted" data?

- Sorted Arrays (not great for updates)
- **Binary Search Trees**

More Dimensions

Goal: A data structure that can answer:

1. Find everyone with a specific birthday.
2. Find everyone with a specific zip code.
3. Find everyone that has a specific birthday and zip code

Idea 1: Three data structures

- Lots of memory

Idea 2: BST over birthday

- Operation 2 is $O(N)$
- Operation 3 is $O(\log(N) + |\text{same bday}|)$

Idea 3: BST over zip code

- Operation 1 is $O(N)$
- Operation 3 is $O(\log(N) + |\text{same zip}|)$

Idea 4: BST w/ Lexical Order

- Operation 2 is <u>still</u> $O(n)$

Why did it fail?

Ideas 2, 3

BST works by grouping "nearby" values together into the same subtree…

…but "near" in one dimension says nothing about the other!

Idea 4

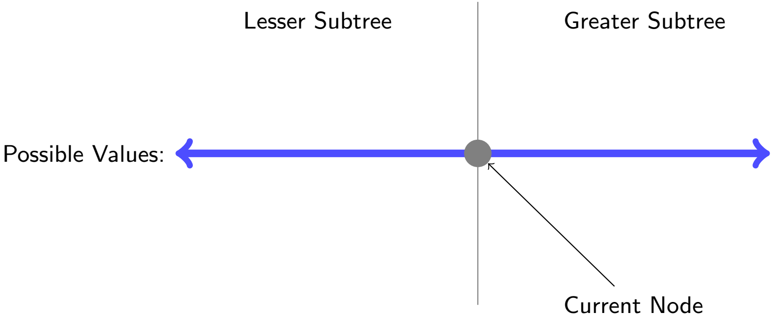
BST works by partitioning the data…

…but lexical order partitions fully on one dimension before partitioning on the other.

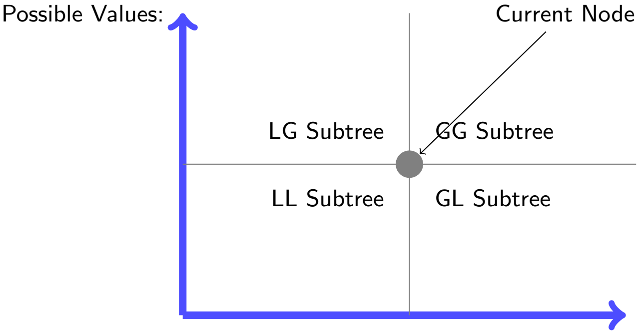
The 2DMap¡T¿ ADT

- ```
  public void insert(int x, int y, T value)
  ```
  Add an element to the map at point $(\mathtt{x}, \mathtt{y})$
- ```
  public T get(int x, int y)
  ```
  Retrieve the element at point $(\mathtt{x}, \mathtt{y})$
- ```
  public Iterator<T>
    range(int xlow, int xhigh, int ylow, int yhigh)
  ```
  Retrieve all elements in the rectangle $(\ [\mathtt{xlow}, \mathtt{xhigh}),\ [\mathtt{ylow}, \mathtt{yhigh})\ )$
- ```
  public T[] kNearestNeighbor(int x, int y, int k)
  ```
  Retrieve the $\mathtt{k}$ elements closest to the point $(\mathtt{x}, \mathtt{y})$

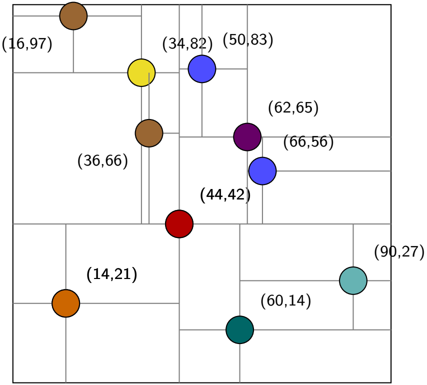
Attempt 1: Partition on both dimensions

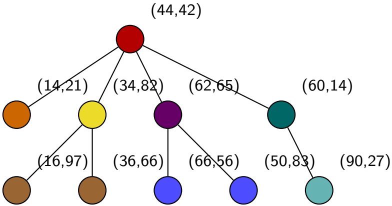


Attempt 1: Partition on both dimensions



Each Node has 4 Children





Each Node has 4 Children

"Binary" Search Tree

- "Bin" – prefix meaning 2
- Each node has (at most) 2 children

"Quadary" Search Tree

- "Quad" – prefix meaning 4
- Each node has (at most) 4 children
- Usually say: "Quad-Tree" instead

Quad Trees — Other Operations

- `get(x, y)`
  - Find position corresponding to (x, y). $O(depth)$
  - Return the node if it exists. $O(1)$
- `insert(x, y, value)`
  - Find placeholder spot corresponding to (x, y). $O(depth)$
  - Create and inject new node. $O(1)$

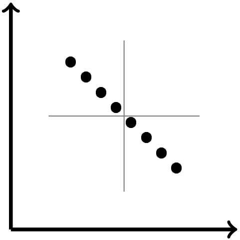
Quad Trees — Challenges

Creating a balanced quad tree is hard

- Impossible to always split collection elements evenly across all four subtrees *(though depth* $= O(\log N)$ *is possible)*

Keeping the quad tree balanced after updates is harder

- No "simple" analog for rotate left/right.



Worst Case:
No possible way to create node with > 2 nonempty subtrees.

Quad Trees — Challenges

Problem: Every node has 4 children!

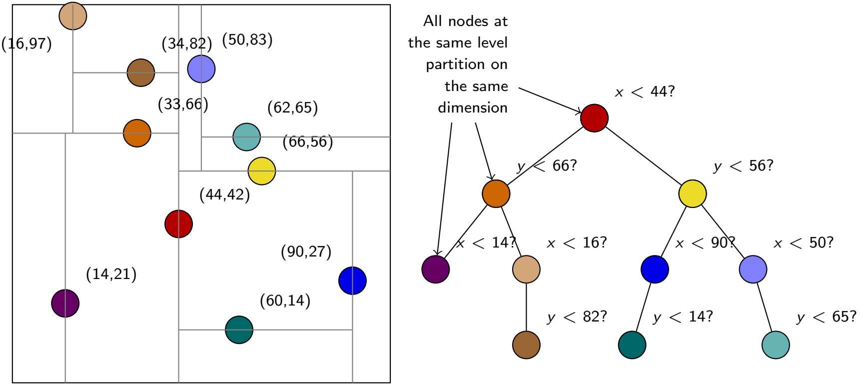
Revisiting Lexical Order

Problem: Searches on lexical order partition all of one dimension first.

Revisiting Lexical Order

Idea: Alternate Dimensions

k-D Trees


(16,97)
(34,82)
(50,83)
(33,66)
(62,65)
(66,56)
(44,42)
(90,27)
(14,21)
(60,14)
All nodes at the same level partition on the same dimension
$x < 44?$
$y < 66?$
$y < 56?$
$x < 14?$
$x < 16?$
$x < 90?$
$x < 50?$
$y < 82?$
$y < 14?$
$y < 65?$

k-D Trees — Find Node

`public Node<T> get(int x, int y)`

- if `this.x == x` $\wedge$ `this.y == y` — return `this`
- if `this.level % 2` $==$ 1
  - if `x < this.x` — return `this.left.get(x,y)`
  - else — return `this.right.get(x,y)`
- else
  - if `y < this.y` — return `this.left.get(x,y)`
  - else — return `this.right.get(x,y)`

What's the complexity?
$O(depth)$

k-D Trees — Depth

Key Insight: If partitioning on only one dimension, we can always find a value that partitions the space in half.[1]

If each tree node partitions its descendants in half, we get $d = O(\log N)$.

[1] Offer void if all values on that dimension are the same.

Quad Trees — Other Operations

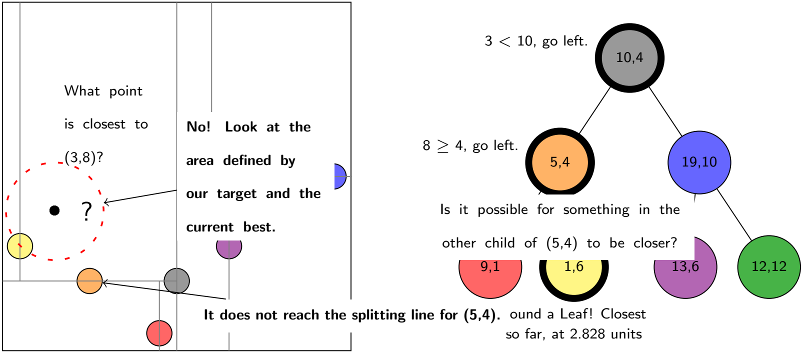
- `get(x, y)`
  - Find position corresponding to (x, y). $O(depth)$
  - Return the node if it exists. $O(1)$
- `insert(x, y, value)`
  - Find placeholder spot corresponding to (x, y). $O(depth)$
  - Create and inject new node. $O(1)$

Nearest Neighbor

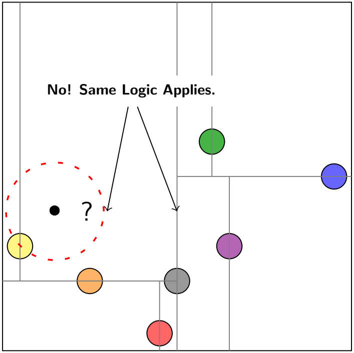
What if we want to find the closest element to a target point?

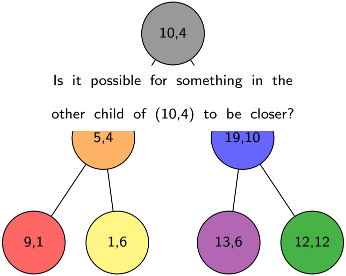
Problem: Can't just do a normal find; The target may not be in the tree at all.

Nearest Neighbor — Example 1

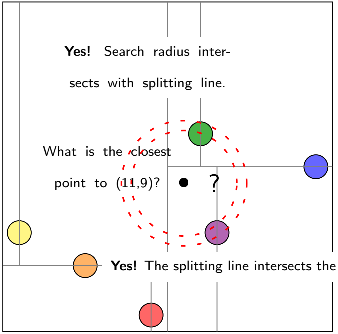


Nearest Neighbor — Example 1





Nearest Neighbor — Example 2



Yes! Search radius intersects with splitting line.

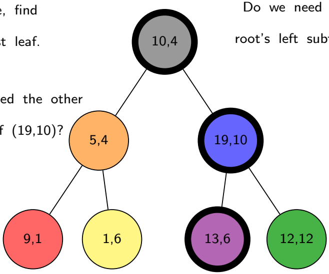
What is the closest point to (11,9)?

Yes! The splitting line intersects the radius

As before, find the closest leaf.

Do we need the other child of (19,10)?

Do we need the root's left subtree?



Generalization: k-Nearest Neighbors

Finding one point can be as fast as $O(d) = O(\log N)$**, but as slow as** $O(N)$

What if we want to find the k-Nearest Neighbors instead?

Idea: Keep a list of the k nearest points, and the furthest point defines our “search radius”

k-D Trees

Can generalize to $k > 2$ dimensions

- **Level 1**: Partition on Dimension 1
- **Level 2**: Partition on Dimension 2
- ...
- **Level k**: Partition on Dimension k
- **Level k+1**: Partition on Dimension 1
- **Level k+2**: Partition on Dimension 2
- **Level i**: Partition on Dimension $((i-1) \mod k) + 1$

In practice `range()` and `knn()` become $O(n)$ for $k > 3$
(If the range overlaps in even one dimension we need to search it)

Other Problems — N-Body Problem

What if we want to compute interactions between one body and every other body?

Naively, this would be $O(N^2)$, but likely we don't care as much about interactions with bodies that are very very far away.

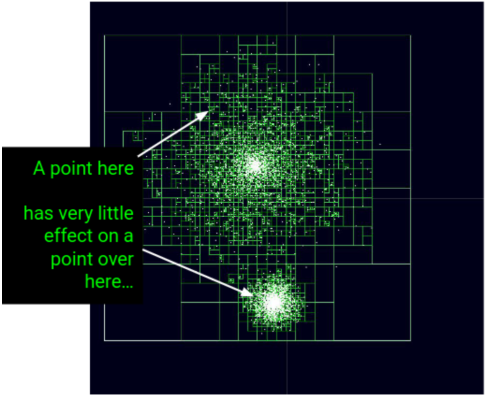
Other Problems — N-Body Problem

Idea: Divide our points into a quadtree (or octree in 3 dimensions)

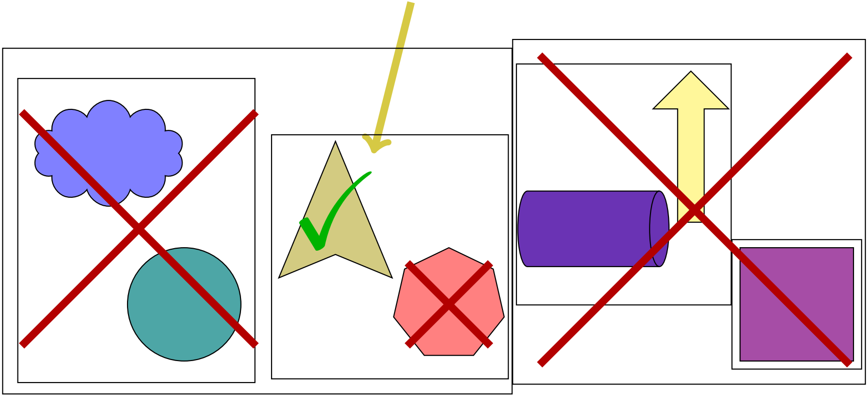
Do full calculations for points in the same box.

Compute a summary (e.g., total force and center of mass) for each box; treat the entire box as one point.

Runtime is now $O(N \log N)$



Other Problems — Ray/Path Tracing



Which object does this ray of light hit? Do we need to check every object?
Idea: Build a hierarchy of bounding boxes (Bounding Volume Hierarchy). If the ray doesn't intersect a bounding box, we ignore it. If the BVH is balanced,