CSE 250: Spatial Indexing

Lecture 34

Nov 27, 2023

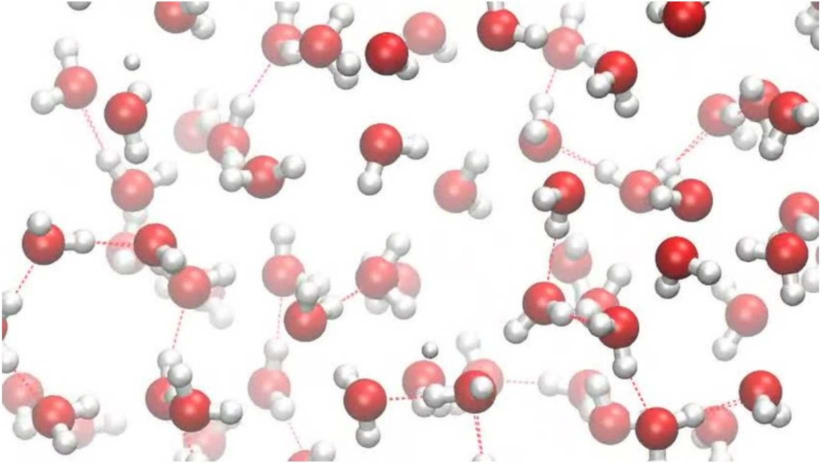
Reminders

- 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.

Some problems are Really big!

ESA/Hubble and NASA; http://www.spacetelescope.org/images/potw1006a/

Some problems are Really small!



Molecular Dynamics Simulation of Liquid Water;
https://commons.wikimedia.org/wiki/File:A_Molecular_Dynamics_Simulation_of_Liquid_Water_at_298_K.webm

Some problems are Really detailed!

This is **not** a photo. It's a computer generated image.

Ray tracing can create photorealistic images; https://en.wikipedia.org/wiki/Ray_tracing_(graphics)#/media/File:Glasses_800_edit.png

What do these things have in common?

- They have **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?

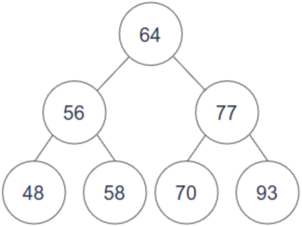
- Sorted Arrays (... are not great for updates)
- **Binary Search Trees**

Binary Search Trees (for 1D data)

```
class Node<T>
{
  public T value;

  /** Guarantee:
        left.value < this.value **/
  Optional<Node<T>> left
    = Optional.empty();

  /** Guarantee:
        right.value >= this.value **/
  Optional<Node<T>> right
    = Optional.empty();
}
```



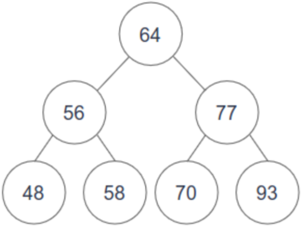
Binary Search Trees (for 1D data)

`Insert`

- Find the right spot — $O(depth)$
- Create and insert — $O(1)$

`Find`

- Find the right spot — $O(depth)$
- Create and insert — $O(1)$



If the tree is balanced, $O(depth) = O(\log N)$

More Dimensions

This worked for 1-dimensional data. How could we change it to work with 2-dimensional data?
Example: Birthday, Zip Code

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 still $O(n)$

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.

Related Problems

Mapping

- What's within $\frac{1}{2}$ mile of me?
- What's within 2 minutes of my route?

Games

- What objects are close enough that they might need to be rendered?
- Which direction should an NPC move in to be in range of an enemy?

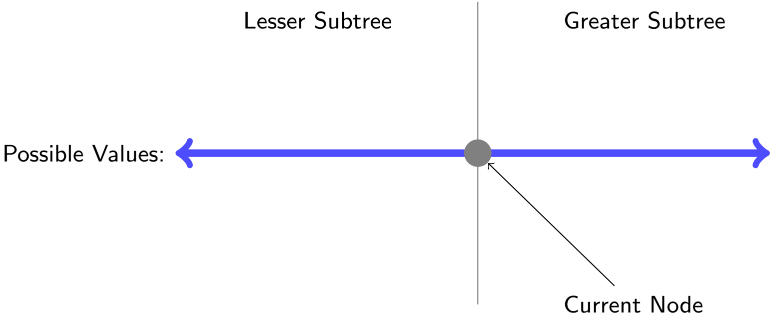
Science

- "Big Brain Project": Neuron A fired; What other neurons are close enough to be stimulated?
- Astronomy / MD: What forces are affecting a particular body? What forces can we ignore/estimate?

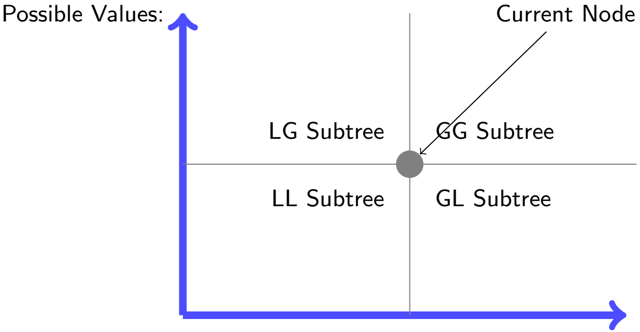
The 2DMap¡T¿ ADT

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

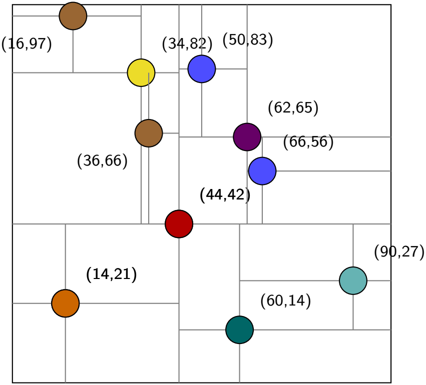
Attempt 1: Partition on both dimensions

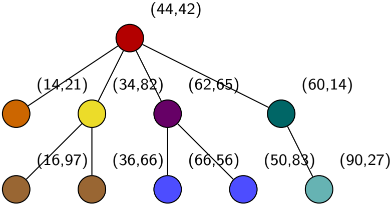


Attempt 1: Partition on both dimensions



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 — Find Node

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

- If `current.x` $==$ `x` $\wedge$ `current.y` $==$ `y`
  - `return current`
- If `current.x` $<$ `x`
  - If `current.y` $<$ `y` — `return current.gg.get(x, y)`
  - Else — `return current.gl.get(x, y)`
- Else
  - If `current.y` $<$ `y` — `return current.lg.get(x, y)`
  - Else — `return current.ll.get(x, y)`

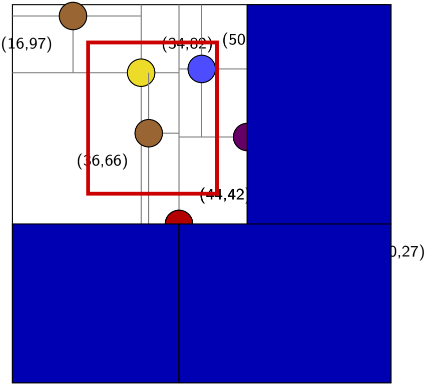
What is the complexity?

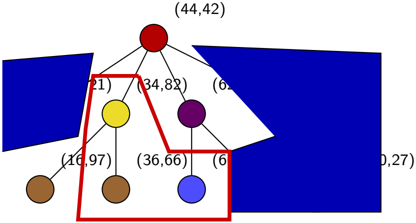
$O(depth)$

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. $O(1)$
- `range(xlow, xhigh, ylow, yhigh)`
  - ...?

Each Node has 4 Children





Quad Trees — Range

`public Iterator<T> range(Rectangle target)`

- if `target.isEmpty()` return
- if `target.contains(x, y)` add value to result
- if `ll.isDefined ll.range(target.crop(`
  `new Rectangle(`$-\infty$`, x, `$-\infty$`, y)))`
- if `lg.isDefined lg.range(target.crop(`
  `new Rectangle(`$-\infty$`, x, y, `∞`)))`
- if `gl.isDefined ll.range(target.crop(`
  `new Rectangle(x, `∞`, `$-\infty$`, y)))`
- if `gg.isDefined lg.range(target.crop(`
  `new Rectangle(x, `∞`, y, `∞`)))`

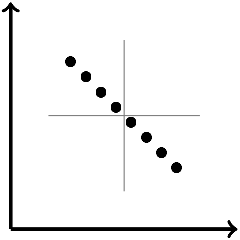
Quad Trees — Challenges

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!