CS 164 & CS 266: Computational Geometry

Lecture 17

Segment trees and interval trees

David Eppstein
University of California, Irvine

Fall Quarter, 2023

Range search for non-point data

Range search revisited

Range search

List geometric objects that match some criterion

Or report aggregate information (# objects, max-priority object)

So far

Objects are points in the plane

Criterion: point is in some query shape

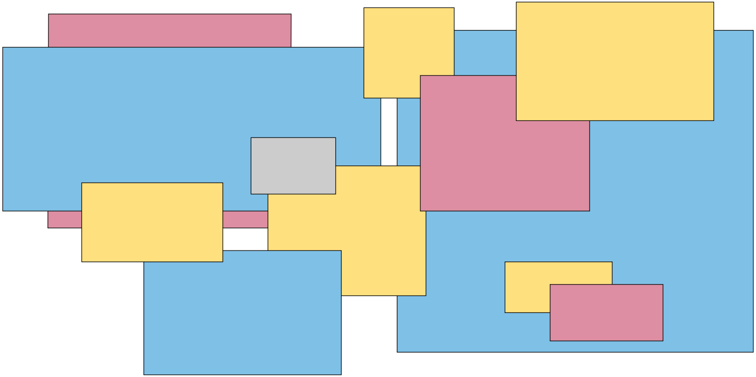
This time (Chapter 10):

Objects are shapes in the plane

Criterion: shape contains a query point

Example

Given overlapping rectangular windows, what is the top window at the point where the mouse was just clicked?



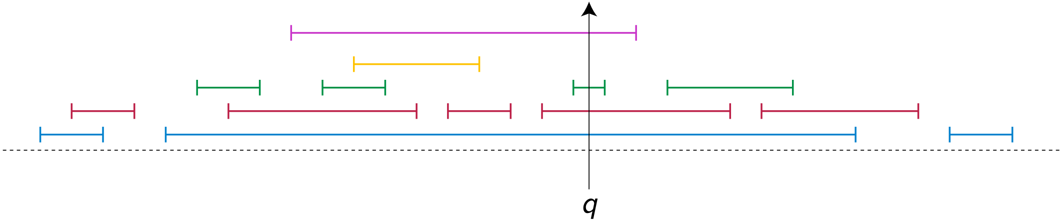
Data: Rectangles with front-to-back ordering

Criterion: Rectangles that contain the mouse click point

Aggregation operation: Find the top window matching the criterion

Today

We will look at a one-dimensional version of these problems, where the data is a collection of one-dimensional intervals (represented in the computer by pairs of left and right end coordinates)



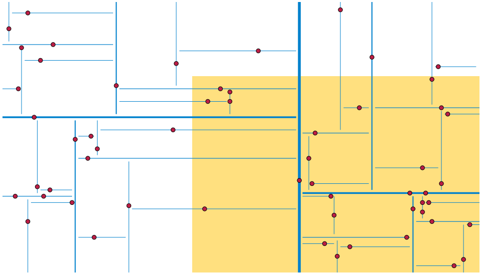
Query: Report some aggregate information about the intervals that contain a query point q

What we already know

An interval can be described by a pair of numbers (L, R)

We can reinterpret those numbers as (x, y)-coordinates

Interval contains q: point (L, R) is in the quadrant of points left of the vertical line $x = q$ and below the horizontal line $y = q$



Can use quadtrees, kD-trees, etc.

...but we can do better!

Two choices

Interval tree

Represent each interval at a single node of a tree structure

Good for listing all intervals containing q

Segment tree

Subdivide each interval into smaller segments

Store them at multiple nodes of a tree structure

Good for counting, prioritization, and recursive structures

An application

Most long web pages scroll vertically, not horizontally

Objects may extend up and down with different heights ⇒ different vertical intervals in page

When scrolling, most of the window contents just move up or down by one pixel, but the browser needs to re-draw one row of pixels at the top or bottom of the window

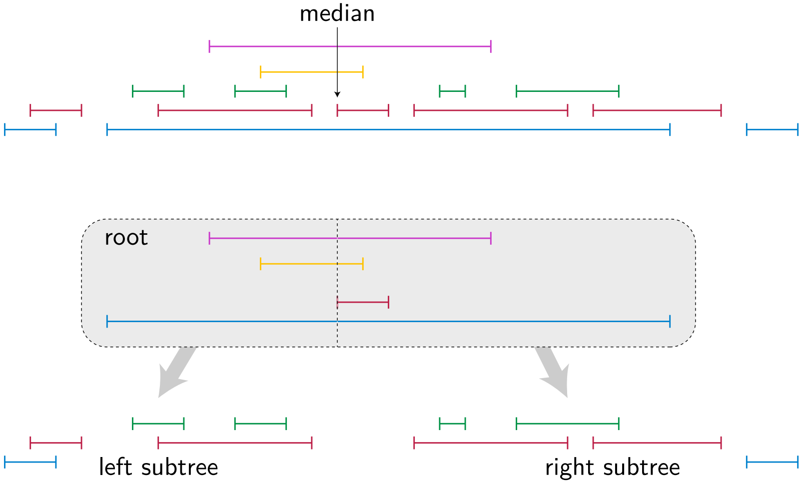
Range listing: Find all objects on this web page that are visible in this row of pixels

Fast query ⇒ interactive scrolling is smooth

Interval tree

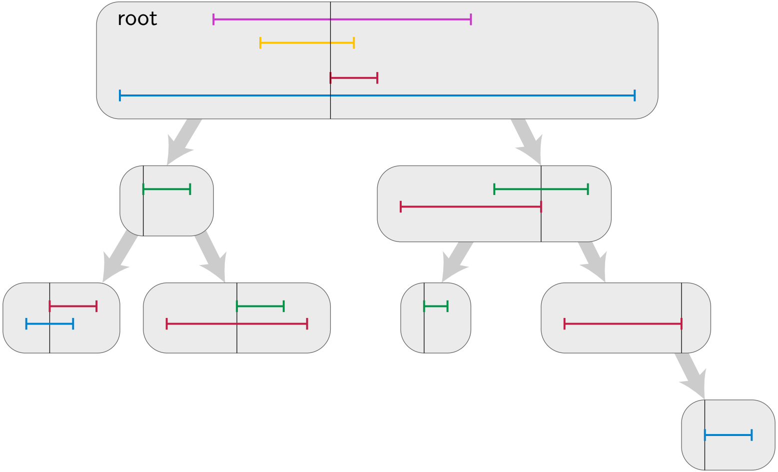
Interval tree construction

Root node stores median endpoint, and all intervals that contain it



Recurse on subsets of intervals to the left and to the right

Completed interval tree



It's a binary search tree on the selected endpoints!

Number of nodes is at most n but may be smaller

How we represent each node

Store selected endpoint as a number (the “key” of a node), pointers to left and right subtree, and:

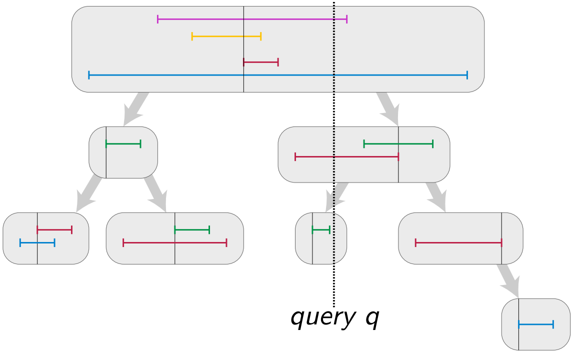
- List of intervals at that node, sorted in increasing order by left endpoint
- List of intervals at that node, sorted in decreasing order by right endpoint

Both the recursive construction and the sorting can be done in $O(n \log n)$ time

Total space is $O(n)$ because each interval is stored in only two lists

Range listing

To find all intervals containing a query number q:



Binary search for q in the binary search tree

If $q \leq$ key: scan the list sorted by left endpoint to find all intervals containing q

Otherwise, scan the list sorted by right endpoints

Query time, for a query that finds k intervals: $O(k + \log n)$

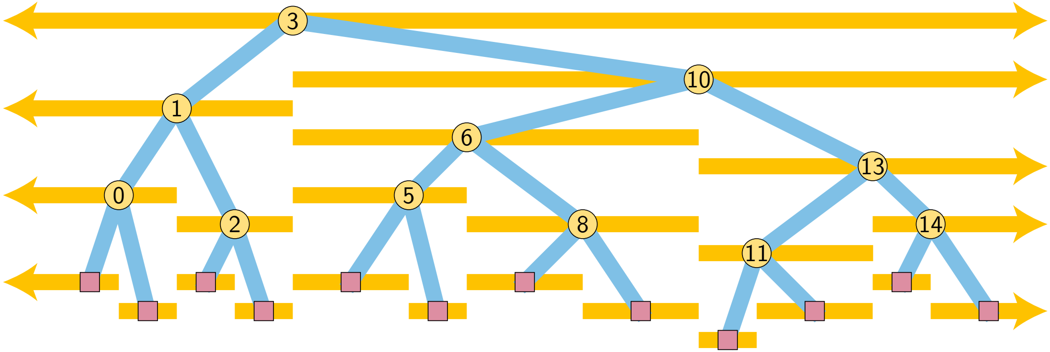
Storage is $O(n)$; construction time is $O(n \log n)$

Segment tree

Segments of a binary search tree

Consider any binary search tree with numbers as keys

Its nodes (including external leaf nodes) correspond to segments in a recursive subdivision of the real line into segments



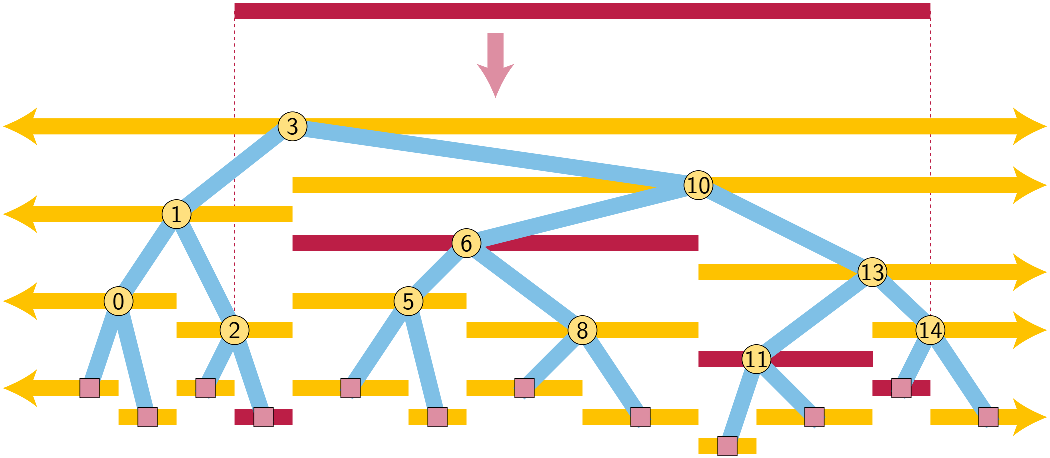
Root node has segment $(-\infty, \infty)$

Split segment for node at its key $\Rightarrow$ two segments for its children

Partitioning intervals into segments

For any interval whose endpoints are keys in the tree,

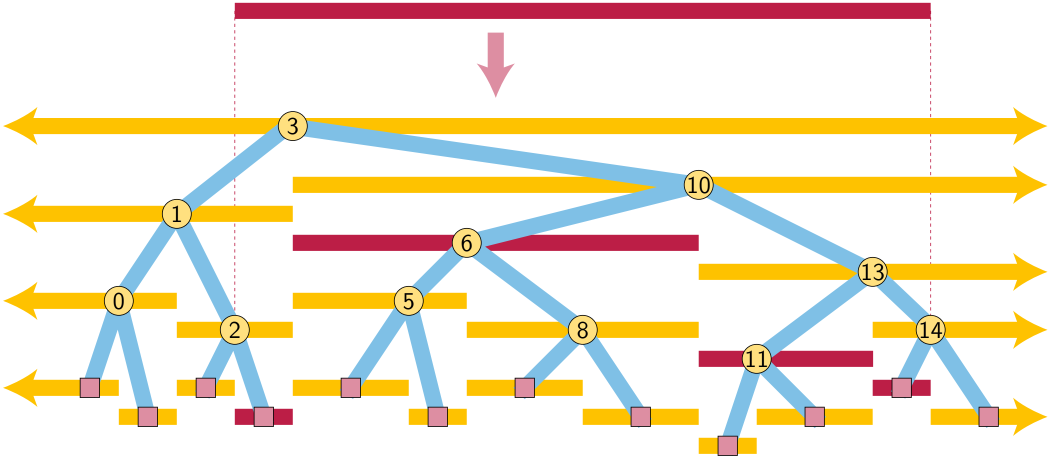
find segment $\subset$ interval with parent segment $\not\subset$ interval



How to find the partition?

Binary search for left and right endpoints

At some point, the two search paths meet at a common ancestor A



Left children of right path + right children of left path, below A

Search tree is balanced $\Rightarrow$ $O(\log n)$ segments

Segment tree: Main ideas

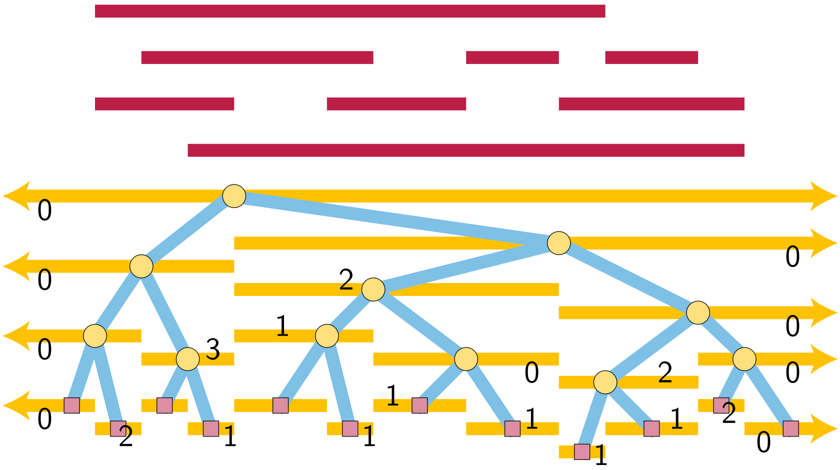
Build binary search tree of all interval endpoints

Split each interval into $O(\log n)$ segments (@ tree nodes)

For each segment, store aggregate information about its intervals
(e.g. count of how many there are)

Segment tree: Example

Binary search tree on interval endpoints

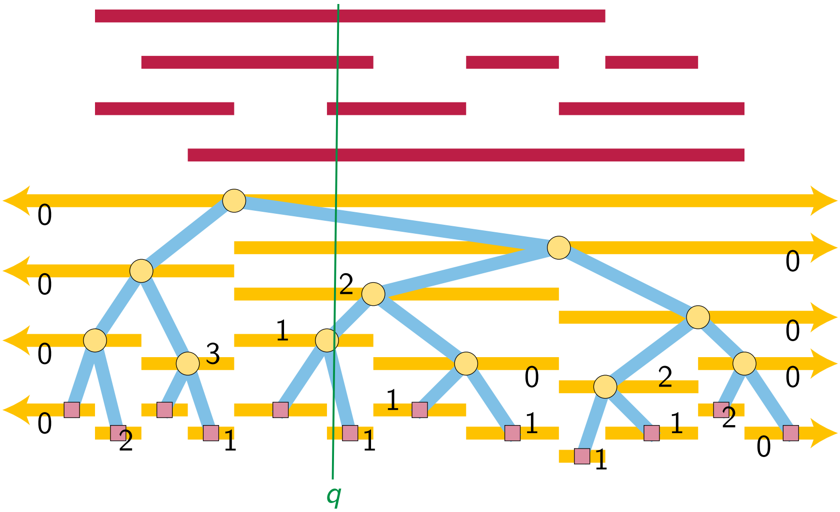


Each tree node is labeled with how many intervals use it as one of their segments

Range counting

To count intervals containing a query point q

Binary search for segments containing q and add their numbers

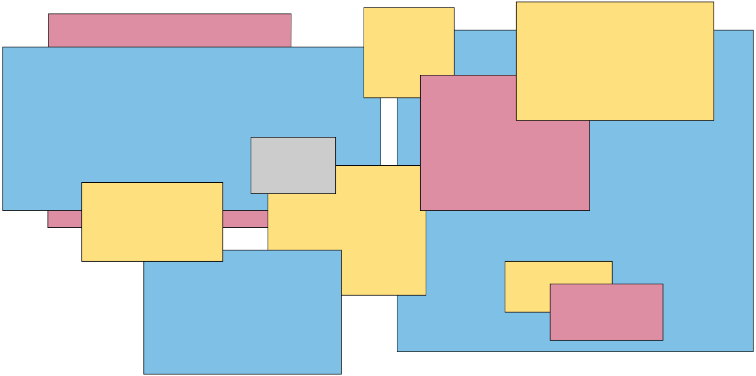


Query time $O(\log n)$, space $O(n)$, preprocessing $O(n \log n)$

Recursive queries

Example

Given overlapping rectangular windows, what is the top window at the point where the mouse was just clicked?



Outer level of recursive structure

Store segment tree of horizontal intervals spanned by each window

Query: x coordinate of mouse click

Result of query: $O(\log n)$ segment tree nodes, each associated with a collection of windows containing that x coordinate

Inner level of recursive structure

Each segment tree node has a collection of windows associated with it

Store sorted list of y-coordinates of bottom and top edges of those windows

For each element of the sorted list, store top window for the interval below it

Query for point (x, y):

For each "outer" segment tree node resulting from the search for x, find the successor of y in the inner sorted list

Each y-query finds the top window among the subset of windows stored at that segment tree node; find which of these windows is topmost overall and return it

Recursive structure analysis

A segment tree for n intervals has $O(n)$ nodes

Each window is associated with $O(\log n)$ nodes of the outer segment tree, and contributes $O(1)$ space to each of their inner sorted lists $\Rightarrow$ storing n windows uses space $O(n \log n)$

Each query looks at $O(\log n)$ nodes of the outer segment tree, and does a recursive query (taking time $O(\log k)$ in an inner sorted list $\Rightarrow$ query time is $O(\log^2 n)$

Fractional cascading can reduce query time to $O(\log n)$