CS 261: Data Structures

Week 10: Odds and ends

Lecture 10a: Retroactivity

David Eppstein
University of California, Irvine

Spring Quarter, 2023

Retroactive data structures

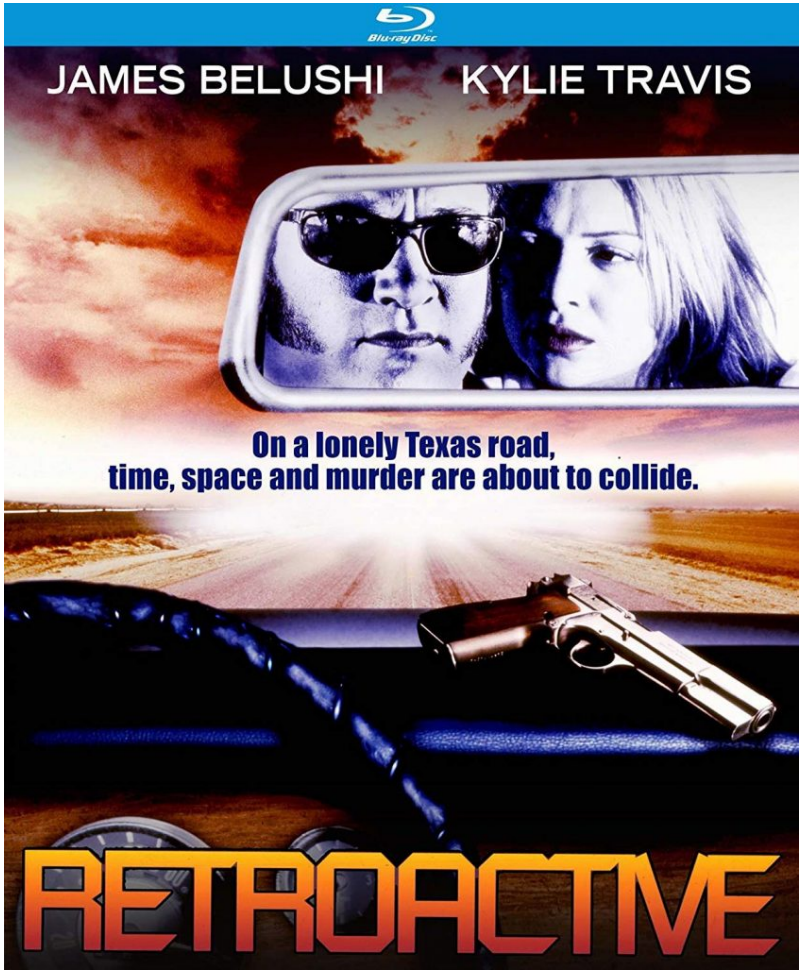
The main idea

There is a single linear timeline (times might as well be numbers)

Each operation has the time it should have been performed as one of its arguments

Updates change the timeline by adding or removing operations

Queries are performed at the given point in the current version of the timeline



Partial versus full retroactivity

Partially retroactive

Updates can happen in the past

Updates can be removed from the timeline as well as inserted

All queries must happen in the present
(that is, their timestamp must be $\geq$ all updates)

Fully retroactive

All operations can have any timestamp (past or present)

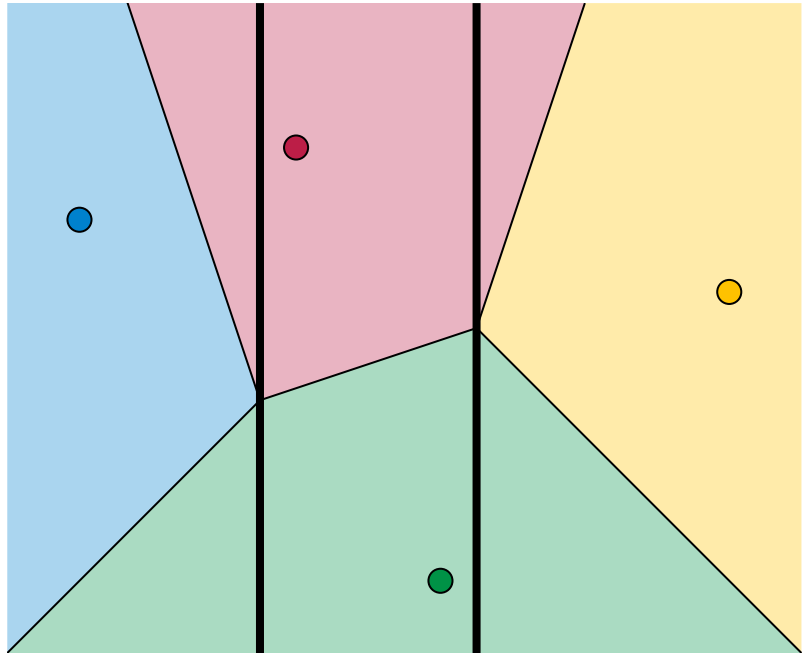
Point location revisited

Time: x-coordinate

Structure: Binary search tree of line segments, ordered by y

Timeline: At x-coordinate of left segment endpoint, insert it into search tree; at coordinate of right endpoint, delete it

To locate point (x, y), search for y in vertical sequence of segments @ time x

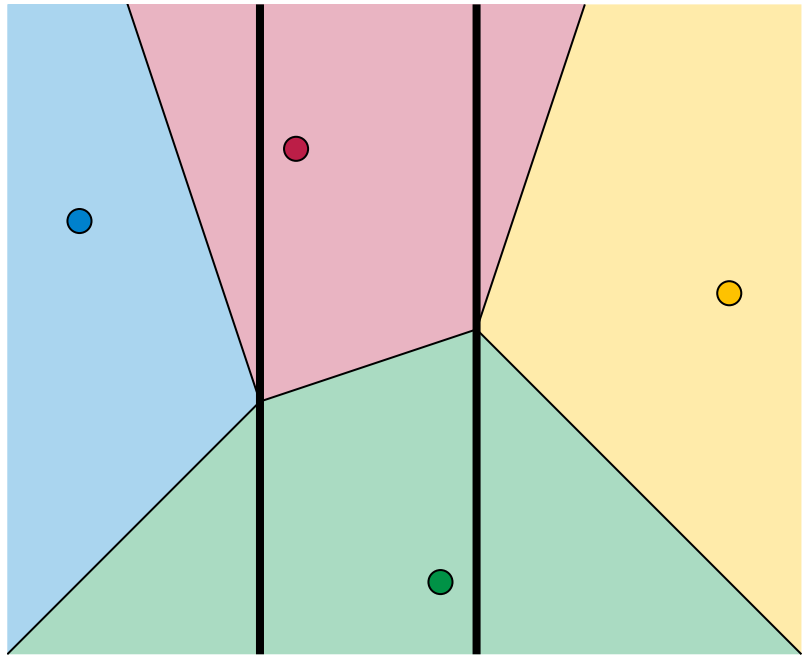


Dynamic point location by retroactivity

To change the subdivision that we are doing point location in:

Add or remove a pair of insert/delete operations in the timeline

Equivalent to adding or removing a line segment from the subdivision



Some history

Demaine, Iacono, Langerman, SODA 2004: Introduce retroactivity

Blelloch, SODA 2008; Giora and Kaplan, 2009:
Optimal retroactive binary search trees: $O(\log n)$ time per operation, $O(n)$ space; application to dynamic point location

Requires that search tree elements come from a single totally ordered sequence. Every two elements can be compared, even when they are not both in the tree at the same time as each other

Dickerson, Eppstein, and Goodrich, ESA 2010:
Retroactive binary search trees, $O(\log n)$ time per update, $O(\log^2 n)$ time per query, $O(n)$ space, only comparing pairs of elements active at the same time

Application to information security: copy a Voronoi diagram given only access to it through post office queries

Example: Retroactive stack API

Push(t, x): Add a push(x) stack operation to the timeline at time t; return an identifier for the added operation

Pop(t): Add a pop stack operation to the timeline at time t and return its identifier

Undo(i): Remove the operation with identifier i from the timeline

Top(t): Return the item that, according to the current timeline, was at the top of the stack at time t

The time arguments can be numbers, or they can be positions in an ordered list (numbered using house numbers)

Retroactive stack implementation

Binary search tree of operations in current timeline

Augmented so that each tree node i stores two numbers:

- g_i: How much the stack grows (positive) or shrinks (negative) during the sequence of operations in its subtree
  (total number of pushes minus total number of pops)
- d_i: Deepest point the stack is popped within the sequence of operations (relative to the starting level of the sequence)

Retroactive stack operations

Top(t):

- Search tree, adding g_i for subtrees to left of search path, to find stack size at time t
- Search again, using deepest points, to find the most recent earlier operation that started from a smaller stack size
- It must be a push and its argument is the element we want

Change timeline:

- Update g_i and d_i in augmented trees
  (in constant time at each ancestor of change, by combining information from its two children)
- Make sure that update does not cause deepest pop operation in whole tree to be at negative depth (this means that it is trying to pop more elements than were pushed)

Summary

Summary

- Definition of persistence and types of persistence
- Persistent stacks and application to programming language implementation
- Path-copying method for making treelike structures persistent
- Path-copying prefix sum and path-copying zippers
- Fat node method for making anything persistent
- Flat tree implementation of partially persistent fat nodes
- Hybrid persistence for efficient partially persistent binary search trees
- Locus method and point location using persistent search trees
- Retroactivity and retroactive stacks