

TODAY: Geometry I (of 2)

- point location
  - static via persistence
  - dynamic via retroactivity
- orthogonal range searching
  - range trees
  - layered range trees
- dynamization with augmentation via weight balance
- fractional cascading

Planar point location: given planar map,  
 (planar) graph drawn in plane  
 with straight edges & without crossings

support query: which face contains point  $(x, y)$ ?

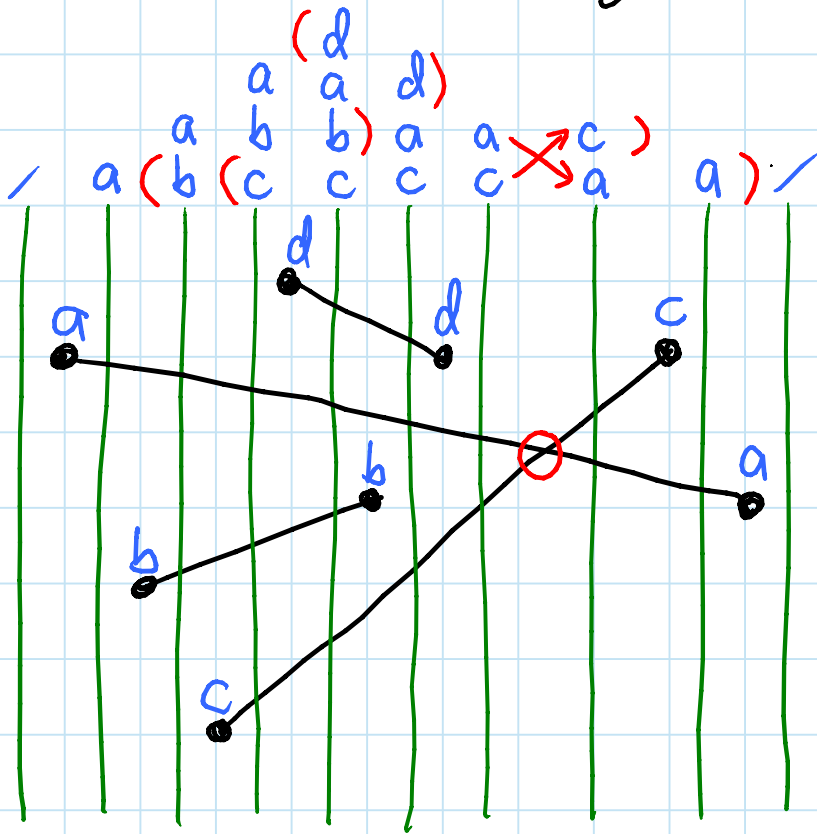
- e.g. which GUI element got clicked?  
 which city are these GPS coords. in?
- static: preprocess map
- dynamic: add/delete edges (& deg.- $\emptyset$  vertices)

Vertical ray shooting: given planar map, support  
 query: which edge first hit by ray  $\uparrow (x, y)$

- implies (static) solution to point location:  
 maintain pointer from edge to face below it
- also dynamic reduction with  $+O(\lg n)$  overhead

Line sweep: technique traditionally used for line-segment intersection

[see e.g. CLRS]



- maintain order of intersection with vertical line which sweeps right
- left/right endpoints are inserts/deletes
- order swaps are crossings

- typically intersection DS = balanced BST  
 $\Rightarrow$  line-segment intersection in  $O(n \lg n + k)$

output size  $\uparrow$

- if we use partially persistent balanced BST then successor(y) query at time t = upward ray shooting query from  $(t, y)$

$\Rightarrow O(\lg n)$  query after  $O(n \lg n)$  preprocessing

[Dobkin & Lipton - SICOMP 1976]

(part of the initial motivation for persistence)

- if we use fully retroactive balanced BST then  $\text{Insert/Delete}(t_1, \text{insert}(y)) + \text{Insert/Delete}(t_2, \text{delete}(y)) = \text{insert/delete edge}(t_1, y) \rightarrow (t_2, y)$
- $\Rightarrow O(\lg n)$  dynamic vertical ray shooting among horizontal line segments  
[Giova & Kaplan - T. Alg. 2009]  
also: [Blelloch - SODA 2008] (later)
- also reduces back to retroactive successor

**OPEN**:  $O(\lg n)$  dynamic vertical ray shooting in general planar map?

- $O(\lg n \lg \lg n)$  query & insert;  $O(\lg^2 n)$  delete  
[Baumgarten, Jung, Mehlhorn - J. Alg. 1994]
- $O(\lg n)$  query,  $O(\lg^{1+\epsilon} n)$  insert,  $O(\lg^{2+\epsilon} n)$  delete  
[Arge, Brodal, Georgiadis - FOCS 2006]

**OPEN**:  $O(\lg n)$  static ray shooting (not vertical)

- $O(\frac{n}{\sqrt{s}} \text{poly} \lg n)$  query &  $O(s^{1+\epsilon})$  space

[Agarwal - SICOMP 1992]

- conjectured nearly optimal
- 3D even harder e.g. [Agarwal & Sharir - SICOMP 1996]
- motivation: ray tracing

## Orthogonal range searching:

maintain  $n$  points in  $d$  dimensions subject to query: given box  $[a_1, b_1] \times \dots \times [a_d, b_d]$ , report existence/count/ $k$  points in box

- static: preprocess points; dynamic: insert/delete
- motivation: query in database table with  $d$  cols.

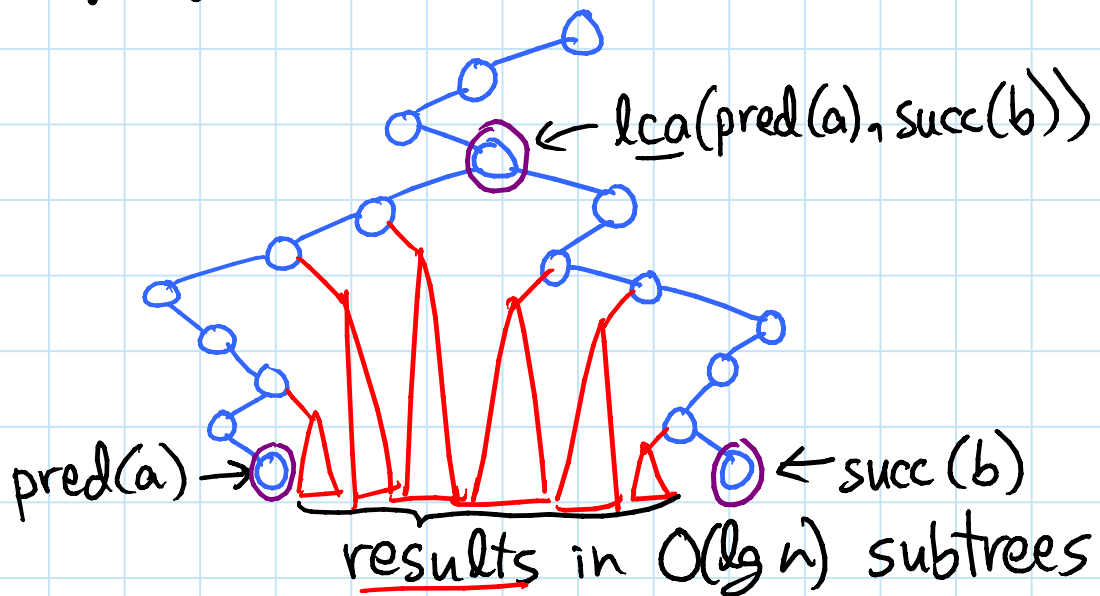
Range trees:  $O(\lg^d n + k)$  query

(see de Berg, Cheong, van Kreveld book)

[Bentley - IPL - 1979; Lee & Wong - TDS - 1980; Lueker - FOCS - 1978; Willard - TR - 1979]

- 1D: balanced BST on leaves = points
  - internal node key =  $\max(\text{left subtree})$
  - query  $[a, b]$ :  $\text{search}(a)$ ;  $\text{search}(b)$

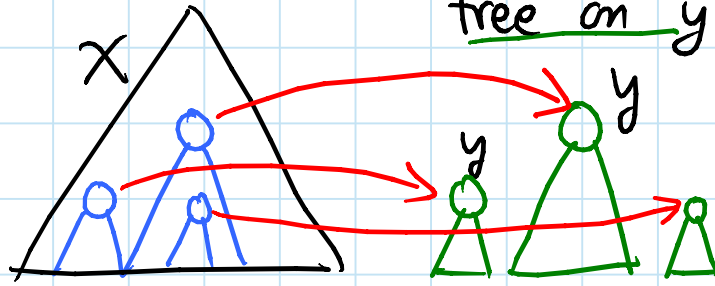
||  
BST



- augment with subtree sizes to get count

(can also do this with regular BST but messier, especially to generalize)

- 2D: 1D tree on  $x$  + each subtree links to 1D tree on  $y$  on same points

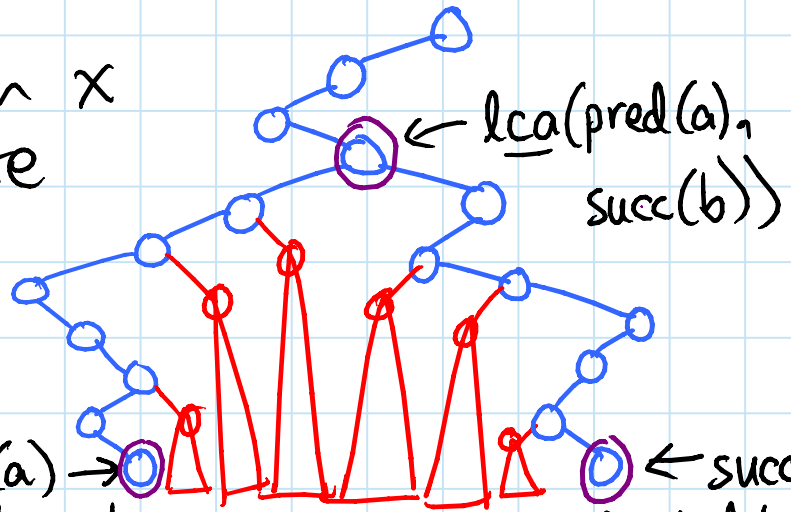


- each point appears in  $\Theta(\lg n)$  structures
- $\Rightarrow \Theta(n \lg n)$  space
- query  $([a_1, b_1] \times [a_2, b_2])$ :
  - $\times$  query  $([a_1, b_1]) \rightarrow O(\lg n)$   $x$  subtrees
  - follow pointers  $\rightarrow O(\lg n)$   $y$  trees
  - $O(\lg n)$   $y$  queries  $\rightarrow O(\lg^2 n)$   $y$  subtrees
- $\Rightarrow O(\lg^2 n + k)$  time
- augment  $y$  trees with subtree sizes for count

- $dD$ : recurse on  $d$ 
  - $O(\lg^d n)$  query
  - $O(n \lg^{d-1} n)$  space & preprocessing
  - $O(\lg^d n)$  update: recursively update each node along root-to-leaf path

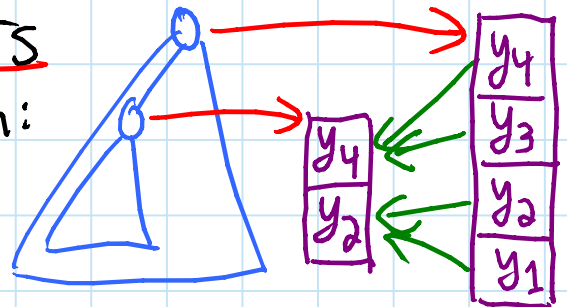
Layered range tree:  $O(\lg^{d-1} n)$  query for  $d > 1$   
 [Gabow, Bentley, Tarjan - <sup>STOC</sup> 1984; Willard - <sup>PhD / SICOMP</sup> 1978 / 1985]

- 2D: search in  $x$  as before



- store  $y$  structures as arrays (sorted by  $y$ )
- search once in root  $y$  structure  $\sim O(\lg n)$
- carry those search results down to result subtree roots

- from one level down:  
 store pointers to corresponding spots (successors)



- $\Rightarrow$  find start & end in  $O(\lg n)$   $y$  arrays in  $O(1)$  per level,  $O(\lg n)$  overall
- can still compute counts & report  $k$  points

-  $dD$ : same as before, just use 2D base case  
 -  $O(n \lg^{d-1} n)$  space & preprocessing

## Dynamization with augmentation via weight balance

- BB[ $\alpha$ ] trees: [Nievergelt & Reingold - STOC 1972]

- for each node  $x$ :

$\text{size}(\text{left}(x))$  &  $\text{size}(\text{right}(x))$  are  $\geq \alpha \cdot \text{size}(x)$

$\Rightarrow \text{height} \leq \log_{\frac{1}{1-\alpha}} n$

- when node is unbalanced, can afford to perfectly rebuild entire subtree of size  $k$ :

- charge to  $\Theta(k)$  of additive imbalance

- update gets charged  $\Theta(\lg n)$  times

$\Rightarrow O(\lg n)$  amortized cost

- applied to layered range tree:

[idea in Lueker - FOCs 1978; see e.g. Willard - SICOMP 1985]

- rebuild costs  $\Theta(k \lg^{d-1} k)$

(some details here to maintain crosslinking)

$\Rightarrow O(\lg^d n)$  amortized update

## Static improvement:

- can reduce space to  $O\left(\frac{n \lg^{d-1} n}{\lg \lg n}\right)$

[Chazelle - SICOMP 1986]

- for  $d \geq 3$ , can improve query to  $O(\lg^{d-2} n)$

-  $O(n \lg^d n)$  space via fractional cascading

[Chazelle & Guibas - Alg. 1986 x2]

-  $O(n \lg^{d-1+\epsilon} n)$  space [Alstrup, Brodal, Rauhe -

FOCS 2000]



Fractional cascading: [Chazelle & Guibas - Alg. 1986 x2]  
dynamic: [Mehlhorn & Näher - Alg. 1990]

Warmup: predecessor/successor search for  $x$   
"1.5D" among  $k$  lists each of length  $n$

-  $O(k \lg n)$  trivial ( $k$  binary searches)

-  $O(k + \lg n)$  solution:

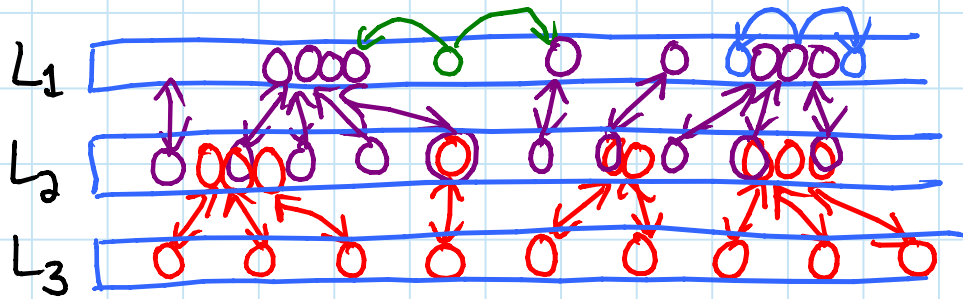
- let  $L'_i = L_i + \text{every other element of } L'_{i+1}$

$\Rightarrow |L'_i| = n + \frac{1}{2} |L'_{i+1}| = O(n)$  (geometric)

- link between identical elements in  $L'_i$  &  $L'_{i+1}$

- each element in  $L_i$  stores pointer to previous/next element in  $L'_i - L_i$

- each element in  $L'_i - L_i$  stores pointer to previous/next element in  $L_i$



- search( $x$ ):

- binary search in  $L'_1 \rightarrow O(\lg n)$

- if amid  $L'_1 - L_1$ , follow pointers to neighbors in  $L_1$  to solve  $L_1$  problem
- if amid  $L_1$ , follow pointers to neighbors in  $L'_1 - L_1$  (else stay)
- walk down to  $L'_2$
- repeat



- General: graph where each
- vertex contains set of elements
  - edge labeled with range  $[a, b]$
  - locally bounded degree: # incoming edges whose labels  $\ni x$  is  $\leq c$ .
- search( $x$ ) wants to find  $x$  in  $k$  vertices' sets found by navigating (online) from any vertex, along edges whose labels  $\ni x$
- improve  $O(k \lg n)$  to  $O(k + \lg n)$

idea: same as warmup  
new: cycles in graph  
but very few items go around cycles

MIT OpenCourseWare  
<http://ocw.mit.edu>

6.851 Advanced Data Structures  
Spring 2012

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.