Segment trees and interval trees

Lecture 5

Antoine Vigneron

antoine.vigneron@jouy.inra.fr

INRA

Outline

- reference
  - textbook chapter 10
  - D. Mount Lectures 13 and 24
- segment trees

  $\Rightarrow$ stabbing queries

  $\Rightarrow$ rectangle intersection
- interval trees

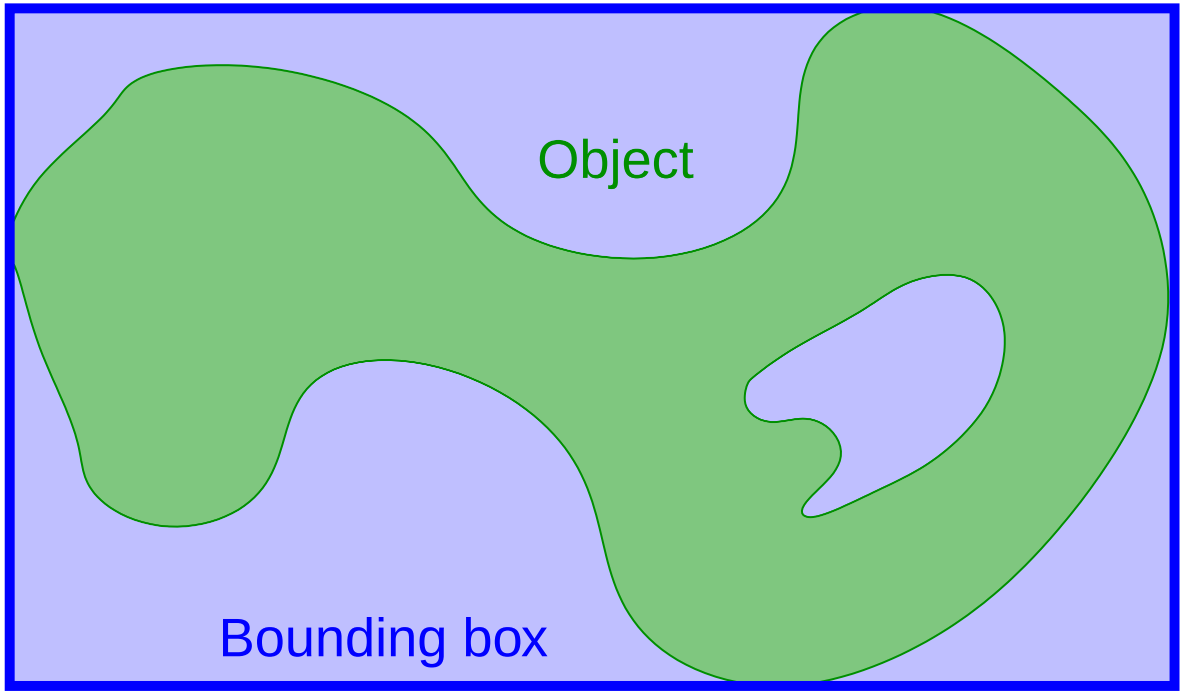
  $\Rightarrow$ improvement
- higher dimension

Stabbing queries

- orthogonal range searching: data points, query rectangle
- stabbing problem: data rectangles, query point
- in one dimension
  - input: a set of n intervals, a query point q
  - output: the k intervals that contain q
- in $\mathbb{R}^d$
  - a box b is isothetic iff it can be written $b = [x_1, x'_1] \times [x_2, x'_2] \times \ldots \times [x_d, x'_d]$
  - in other words it is axis–parallel
  - input: a set of n isothetic boxes, a query point q
  - output: the k boxes that contain q

Motivation

- in graphics and databases, objects are often stored in their bounding box

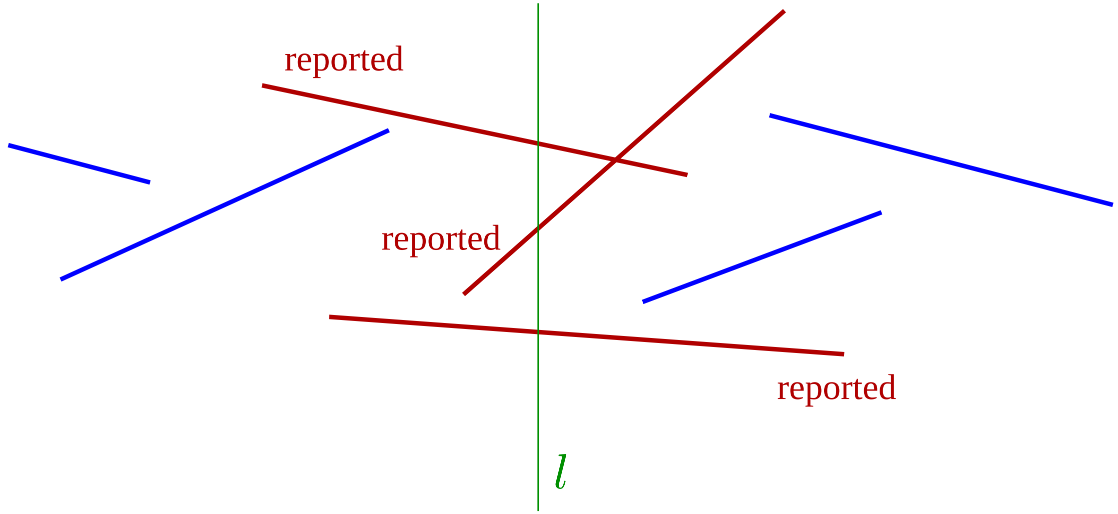


- query: which objects does point x belong to?
- first find objects whose bounding boxes intersect x

Segment trees

Segment tree

- a data structure to store intervals, or segments in $\mathbb{R}^2$
- allows to answer stabbing queries
  - in $\mathbb{R}^2$: report the segments that intersect a query vertical line l



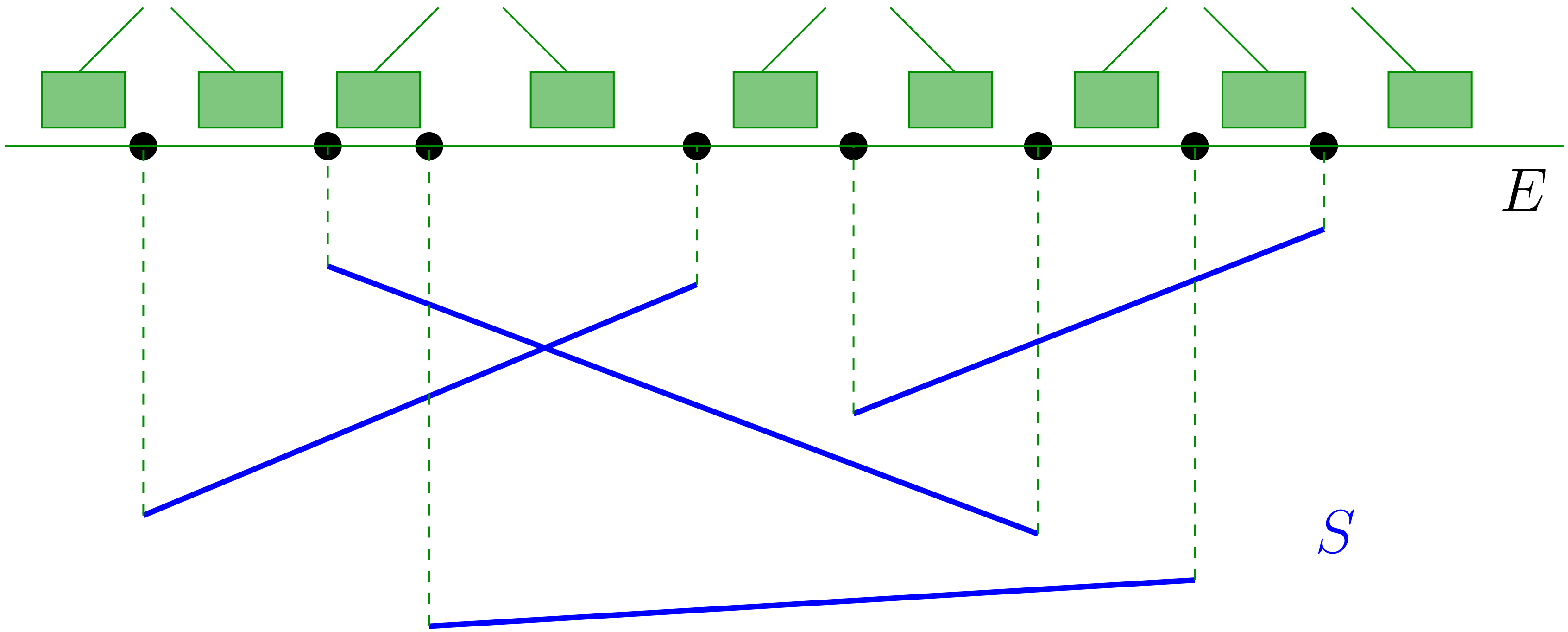
  - query time: $O(\log n + k)$
  - space usage: $O(n \log n)$
  - preprocessing time: $O(n \log n)$

Notations

- let $S = (s_1, s_2, \ldots s_n)$ be a set of segments in $\mathbb{R}^2$
- let E be the set of the x–coordinates of the endpoints of the segments of S
- we assume general position, that is: $|E| = 2n$
- first sort E in increasing order
- $E = \{e_1 < e_2 < \ldots e_{2n}\}$

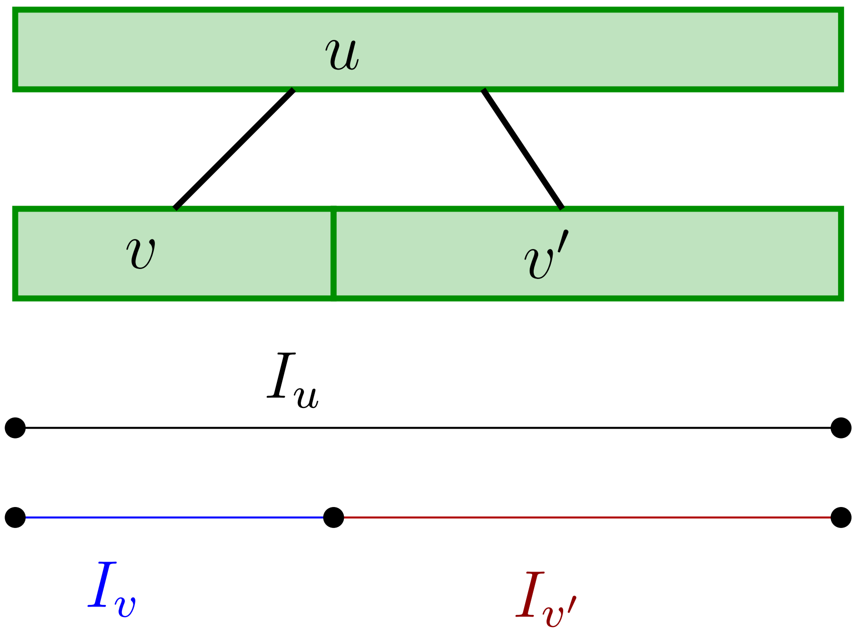
Atomic intervals

- E splits $\mathbb{R}$ into $2n+1$ *atomic intervals:*
  - $[-\infty, e_1]$
  - $[e_i, e_{i+1}]$ for $i \in \{1, 2, \ldots 2n-1\}$
  - $[e_{2n}, \infty]$
- these are the leaves of the segment tree

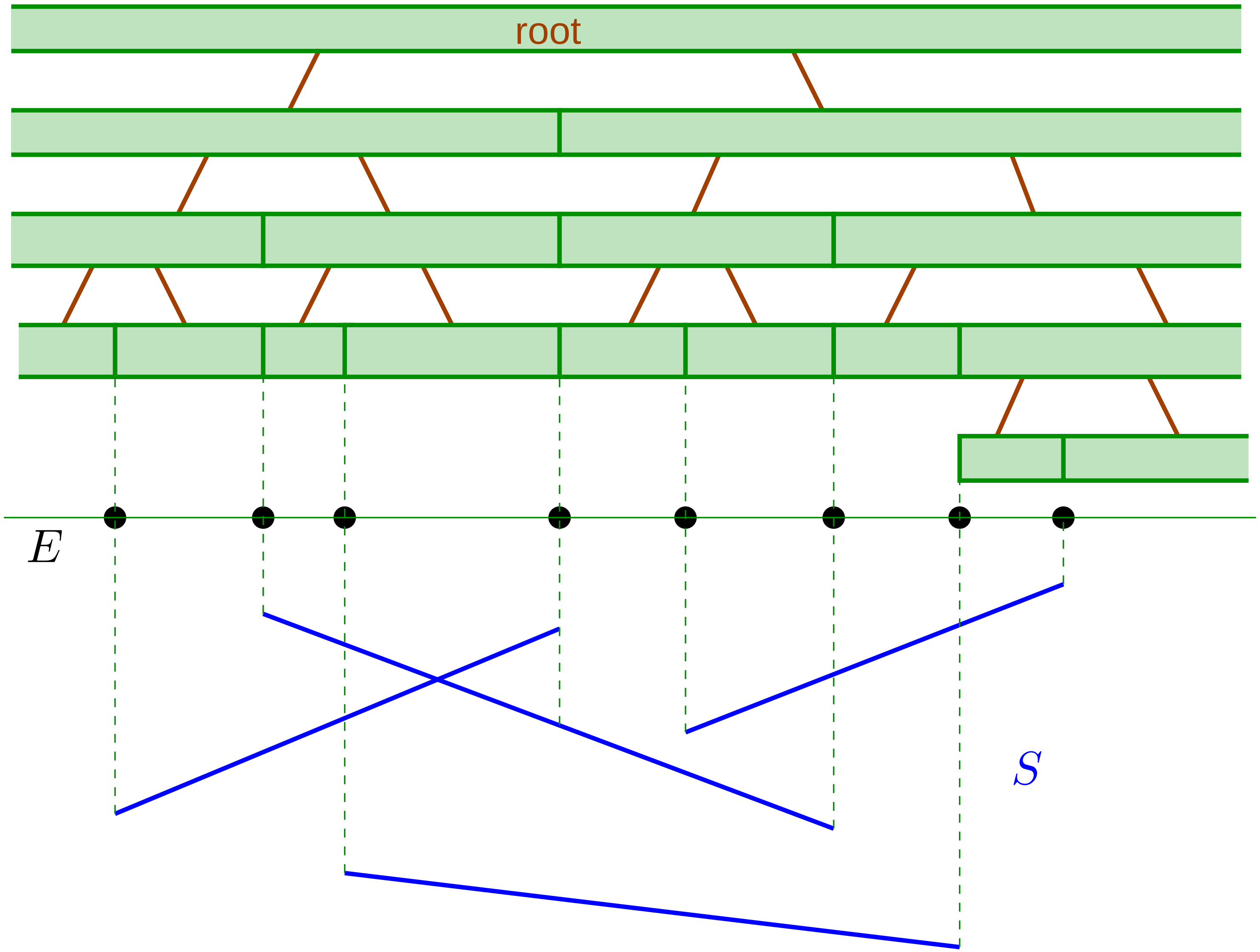


Internal nodes

- the segment tree $\mathcal{T}$ is a balanced binary tree
- each internal node u with children v and v' is associated with an interval $I_u = I_v \cup I_v'$
- an *elementary interval* is an interval associated with a node of $\mathcal{T}$ (it can be an atomic interval)

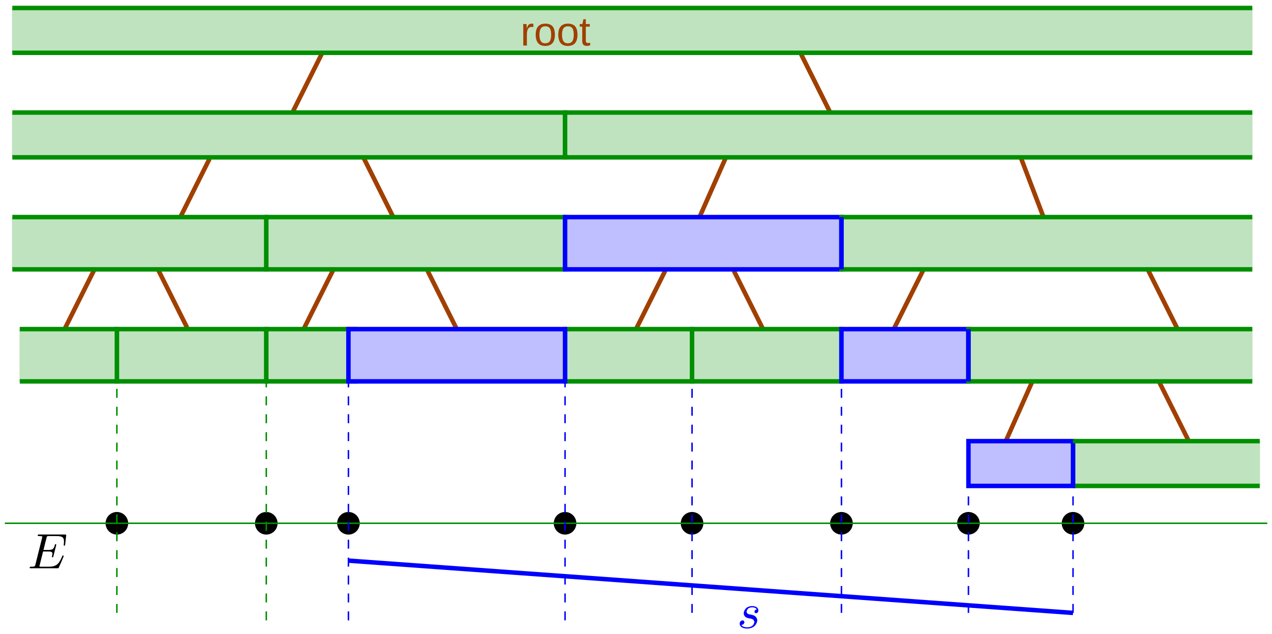


Example



Partitioning a segment

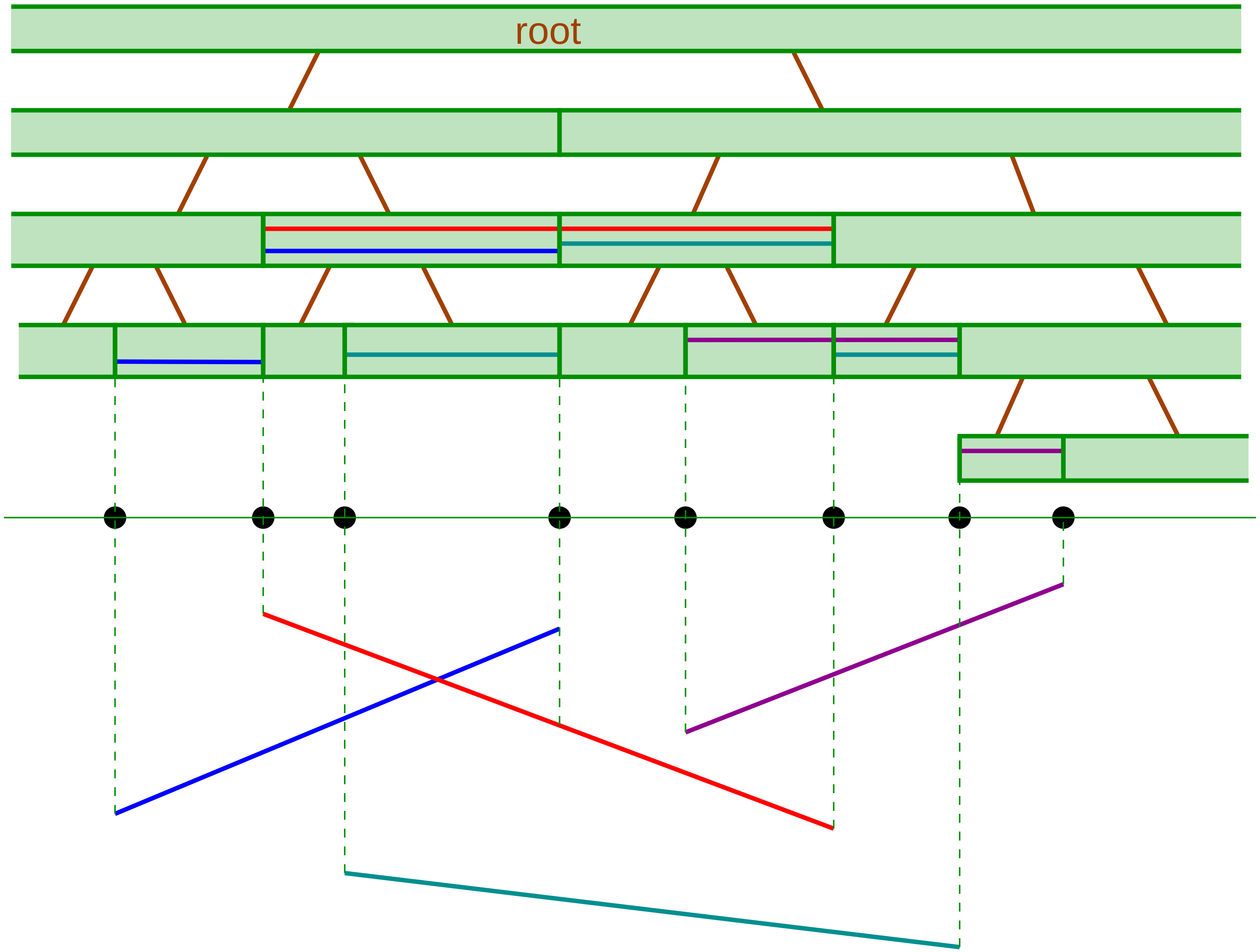
- let $s \in S$ be a segment whose endpoints have x–coordinates e_i and e_j
- $[e_i, e_j]$ is split into several elementary intervals
- they are chosen as close as possible to the root
- s is stored in each node associated with these elementary intervals



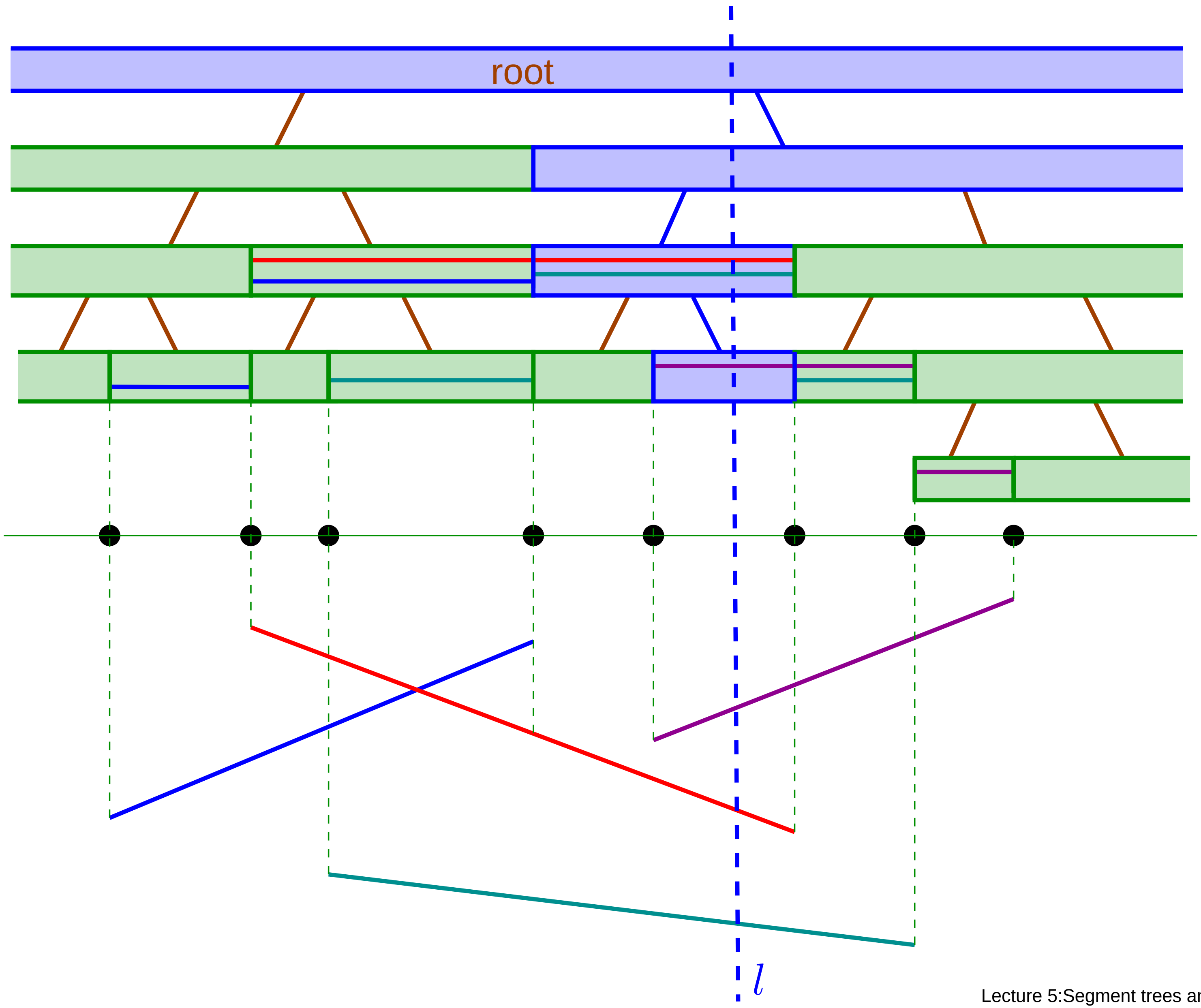
Standard lists

- each node u is associated with a *standard list* L_u
- let $e_i < e_j$ be the x–coordinates of the endpoints of $s \in S$
- then s is stored in L_u iff $I_u \subset [e_i, e_j]$ and $I_{parent(u)} \not\subset [e_i, e_j]$
  (see previous slide and next slide)

Example



Answering a stabbing query



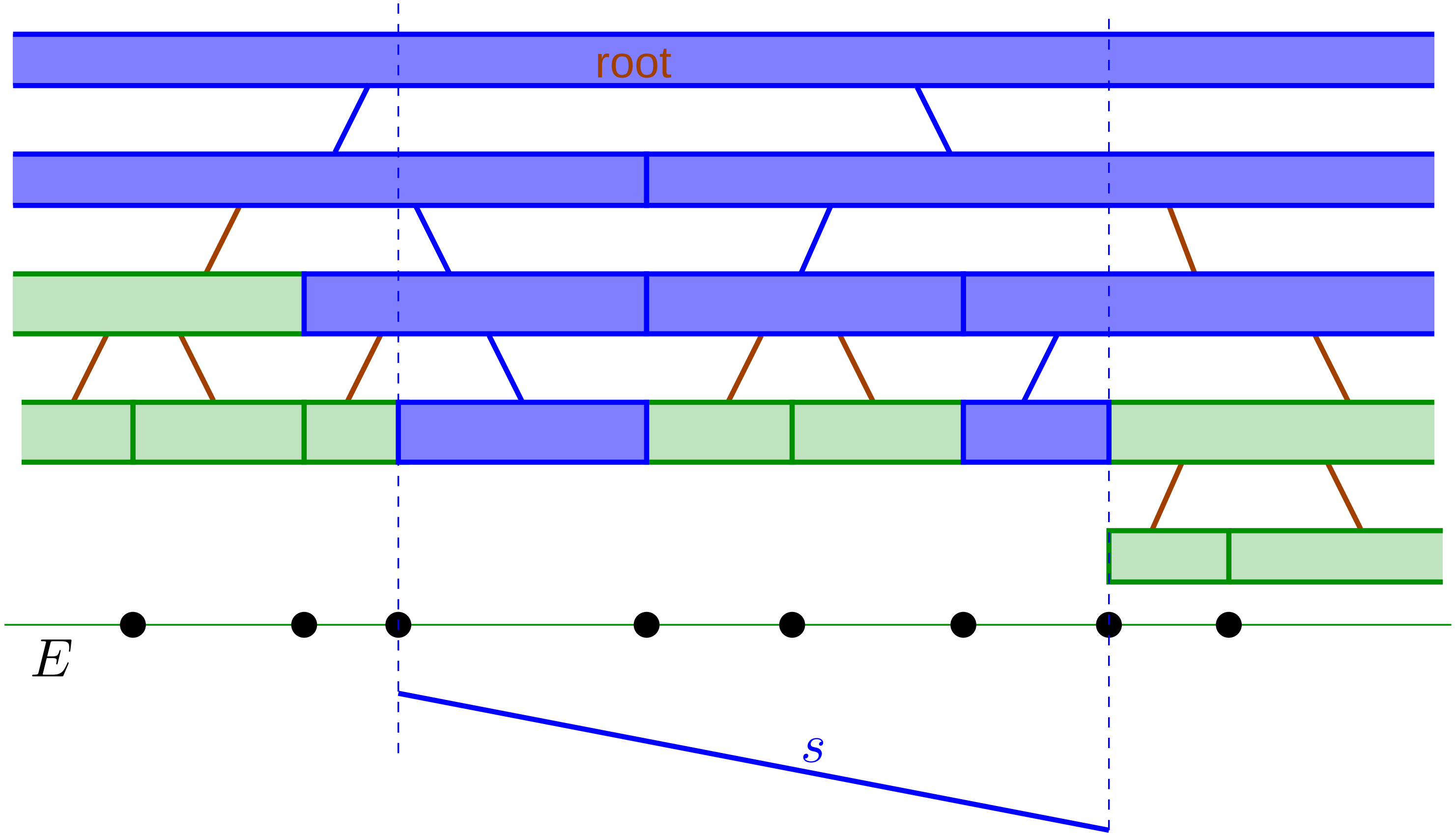
Answering a stabbing query

Algorithm *ReportStabbing*(u, x_l)
Input: root u of $\mathcal{T}$, x–coordinate of l
Output: segments in S that cross l

1. **if** $u == NULL$
2.   **then return**
3. output L_u
4. **if** $x_l \in I_{u.left}$
5.   **then** $ReportStabbing(u.left, x_l)$
6. **if** $x_l \in I_{u.right}$
7.   **then** $ReportStabbing(u.right, x_l)$

- it clearly takes $O(k + \log n)$ time

Inserting a segment



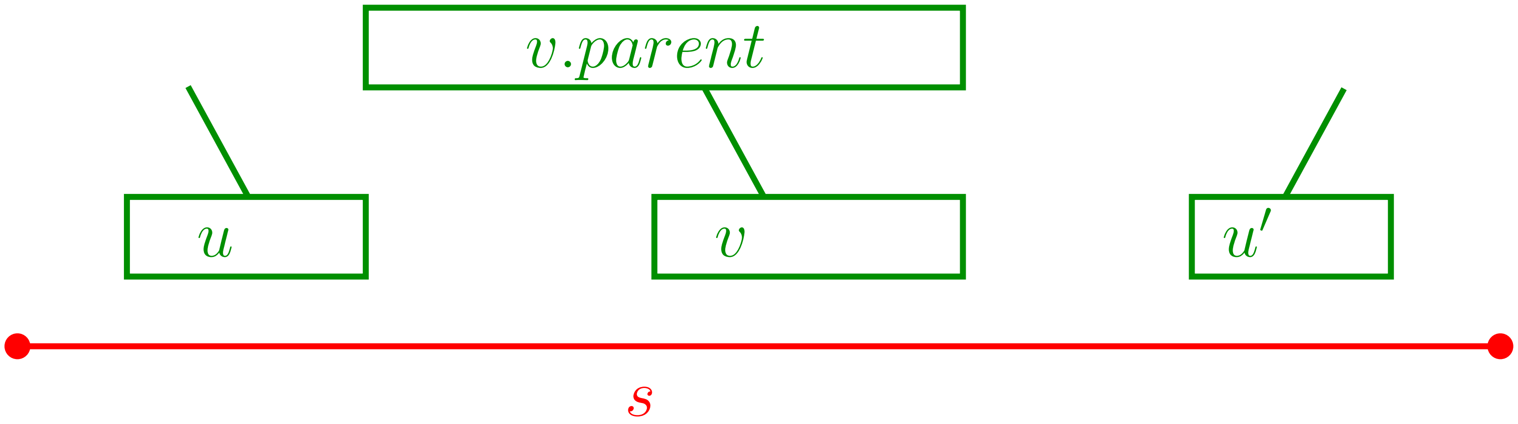
Insertion in a segment tree

Algorithm *Insert*(u, s)
Input: root u of $\mathcal{T}$, segment s. Endpoints of s have x–coordinates $x^- < x^+$

1. **if** $I_u \subset [x^-, x^+]$
2. **then** insert s into L_u
3. **else**
4. **if** $[x^-, x^+] \cap I_{u.left} \neq \emptyset$
5. **then** $Insert(u.left, s)$
6. **if** $[x^-, x^+] \cap I_{u.right} \neq \emptyset$
7. **then** $Insert(u.right, s)$

Property

- s is stored at most twice at each level of $\mathcal{T}$
- proof:
  - by contradiction
  - if s stored at more than 2 nodes at level i
  - let u be the leftmost such node, u' be the rightmost
  - let v be another node at level i containing s



  - then $I_{v.parent} \subset [x^-, x^+]$
  - so s cannot be stored at v

Analysis

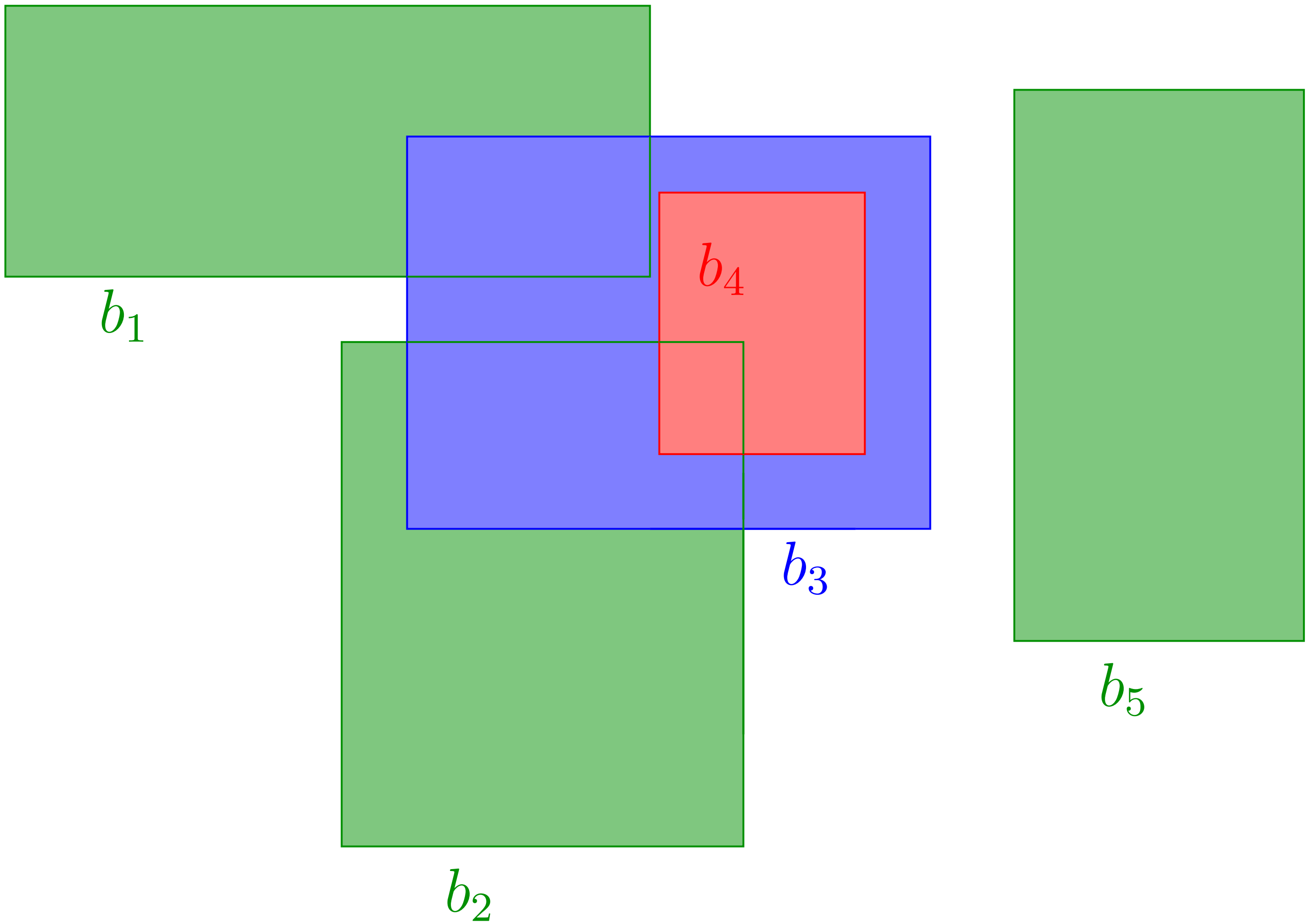
- property of previous slide implies
  - space usage: $O(n \log n)$
- insertion in $O(\log n)$ time (similar proof: four nodes at most are visited at each level)
- actually space usage is $\Theta(n \log n)$ (example?)
- query time: $O(k + \log n)$
- preprocessing
  - sort endpoints: $\Theta(n \log n)$ time
  - build empty segment tree over these endpoints: $O(n)$ time
  - insert n segments into $\mathcal{T}$: $O(n \log n)$ time
  - overall: $\Theta(n \log n)$ preprocessing time

Rectangle intersection

Problem statement

- input: a set B of n isothetic boxes in $\mathbb{R}^2$
- output: all the intersecting pairs in B^2
- using segment trees, we give an $O(n \log n + k)$ time algorithm when k is the number of intersecting pairs
- note: this is optimal
- note: faster than our line segment intersection algorithm
- space usage: $\Theta(n \log n)$ due to segment trees
- space usage is not optimal ($O(n)$ is possible with optimal query time and preprocessing time)

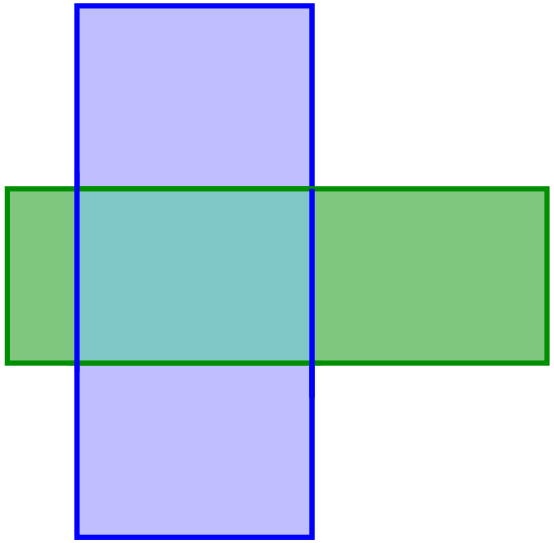
Example



output: (b_1, b_3),(b_2, b_3),(b_2, b_4),(b_3, b_4)

Two kinds of intersections

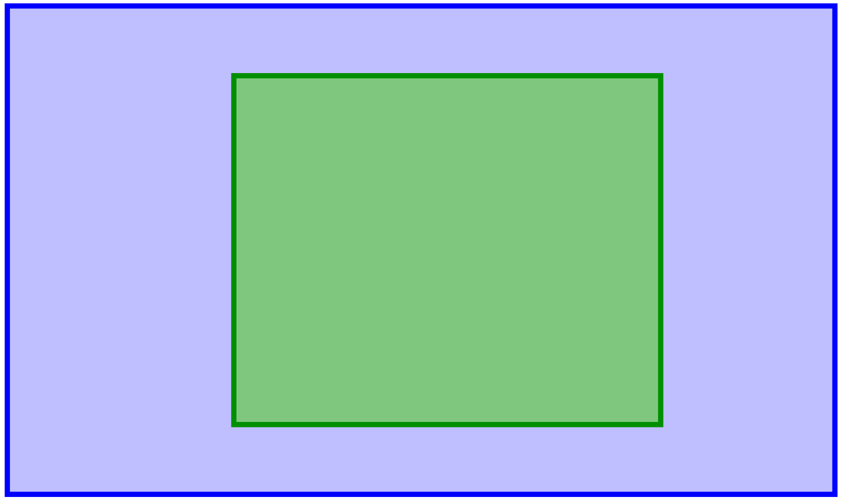
- overlap



- intersecting edges

$\Rightarrow$ reduces to intersection reporting for isothetic segments

- inclusion



- we can find them using stabbing queries

Reporting overlaps

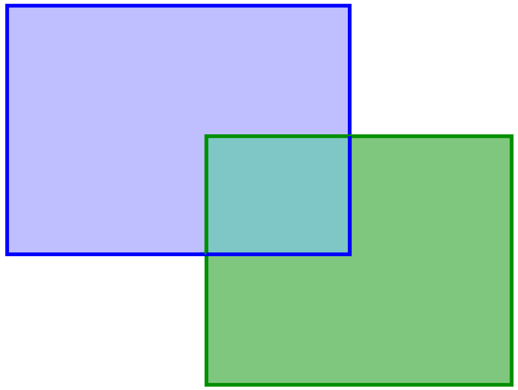
- equivalent to reporting intersecting edges
- plane sweep approach
- sweep line status: BBST containing the horizontal line segments that intersect the sweep line, by increasing y–coordinates
- each time a vertical line segment is encountered, report intersection by range searching in the BBST
- preprocessing time: $O(n \log n)$ for sorting endpoints
- running time: $O(k + n \log n)$

Reporting inclusions

- still using plane sweep
- sweep line status: the boxes that intersect the sweep line l, in a segment tree with respect to y–coordinates
  - the endpoints are the y–coordinates of the horizontal edges of the boxes
  - at a given time, only rectangles that intersect l are in the segment tree
  - we can perform insertion and deletions in a segment tree in $O(\log n)$ time
- each time a vertex of a box is encountered, perform a stabbing query in the segment tree

Remarks

- at each step a box intersection can be reported several times
- in addition there can be overlap and vertex stabbing a box at the same time



- to obtain each intersecting pair only once, make some simple checks (how?)

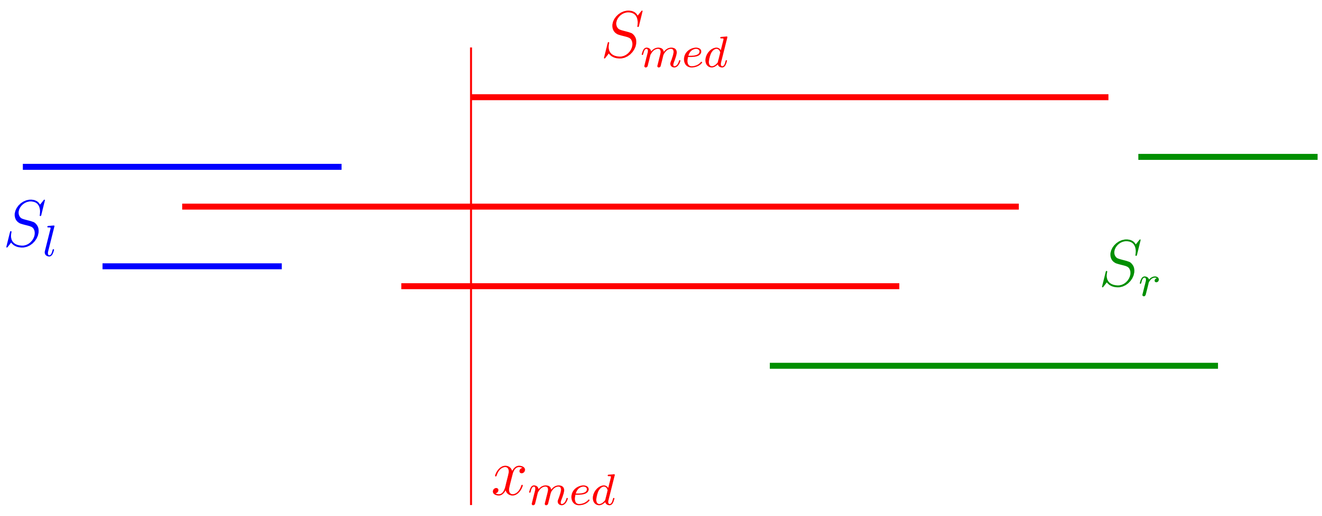
Interval trees

Introduction

- interval trees allow to perform stabbing queries in one dimension
  - query time: $O(k + \log n)$
  - preprocessing time: $O(n \log n)$
  - space: $O(n)$
- reference: D. Mount notes, page 100 (vertical line stabbing queries) to page 103 (not including vertical segment stabbing queries)

Preliminary

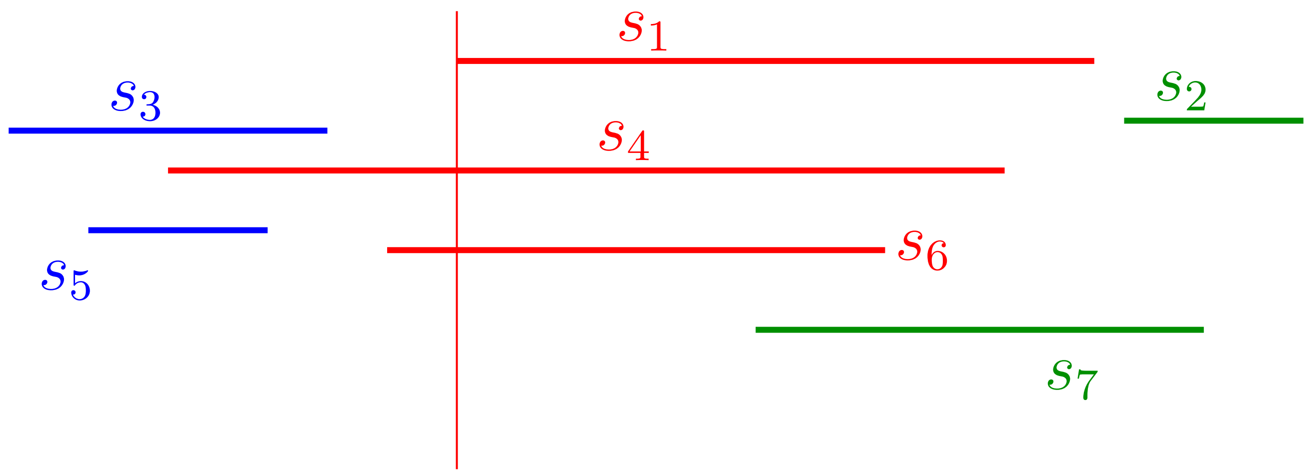
- let x_{med} be the median of E
  - S_l: segments of S that are completely to the left of x_{med}
  - S_{med}: segments of S that contain x_{med}
  - S_r: segments of S that are completely to the right of x_{med}



Data structure

- recursive data structure
- left child of the root: interval tree storing S_l
- right child of the root: interval tree storing S_r
- at the root of the interval tree, we store S_{med} in two lists
  - M_L is sorted according to the coordinate of the left endpoint (in increasing order)
  - M_R is sorted according to the coordinate of the right endpoint (in decreasing order)

Example



$$M_l = (s_4, s_6, s_1)$$
$$M_r = (s_1, s_4, s_6)$$

Interval tree on
s_3 and s_5

Interval tree on
s_2 and s_7

Stabbing queries

- query: x_q, find the intervals that contain x_q
- if $x_q < x_{med}$ then
  - Scan M_l in increasing order, and report segments that are stabbed. When x_q becomes smaller than the x–coordinate of the current left endpoint, stop.
  - recurse on S_l
- if $x_q > x_{med}$
  - analogous, but on the right side

Analysis

- query time
  - size of the subtree divided by at least two at each level
  - scanning through M_l or M_r: proportional to the number of reported intervals
  - conclusion: $O(k + \log n)$ time
- space usage: $O(n)$ (each segment is stored in two lists, and the tree is balanced)
- preprocessing time: easy to do it in $O(n \log n)$ time

Stabbing queries in higher dimension

Approach

- in $\mathbb{R}^d$, a set B of n boxes
- for a query point q find all the boxes that contain it
- we use a multi–level segment tree
- inductive definition, induction on d
- first, we store B in a segment tree $\mathcal{T}$ with respect to x_1–coordinate
- for all node u of $\mathcal{T}$, associate a $(d-1)$–dimensional multi–level segment tree over L_u, with respect to $(x_2, x_3 \dots x_d)$

Performing queries

- search for q in $\mathcal{T}$
- for all nodes in the search path, query recursively the $(d-1)$–dimensional multi–level segment tree
- there are $\log n$ such queries
- by induction on d, we can prove that
  - query time: $O(k + \log^d n)$
  - space usage: $O(n \log^d n)$
  - preprocessing time : $O(n \log^d n)$

Improvements

- fractional cascading at the deepest level of the tree:
  - gains a factor $\log n$ on the query time bound
- interval trees at the deepest level:
  - gains $\log n$ on the space bound