Segment trees and interval trees

Lekcija 11

Sergio Cabello
sergio.cabello@fmf.uni-lj.si
FMF
Univerza v Ljubljani

Includes slides by Antoine Vigneron

Outline

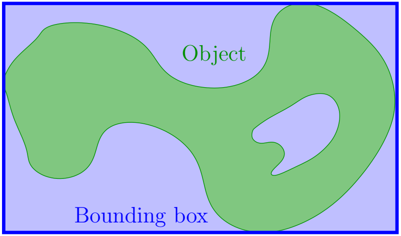
- segment trees
  - stabbing queries
  - windowing problem
  - rectangle intersection
  - Klee's measure problem
- interval trees
  - improvement for some problems
- higher dimension

Data structure for stabbing queries

- orthogonal range searching: data is points, queries are rectangles
- stabbing problem: data is rectangles, queries are points
- in one dimension
  - data: a set of n intervals
  - query: report the k intervals that contain a query point q
- in $\mathbb{R}^d$
  - data: a set of n isothetic (axis-parallel) boxes
  - query: report the k boxes that contain a query point q

Motivation

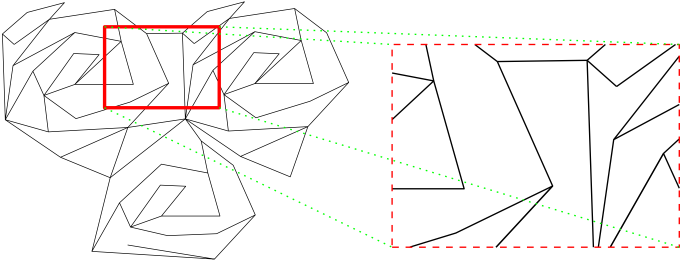
- in graphics and databases, objects are often stored according to their bounding box



- query: which objects does point x belong to?
- first find objects whose bounding boxes contain x
- then perform screening

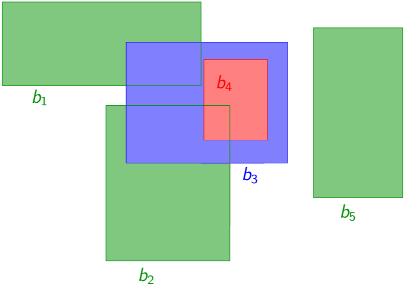
Data structure for windowing queries

- windowing queries
  - data: a set of n disjoint segments in $\mathbb{R}^2$
  - query: report the k segments that intersect a query rectangle R.
- motivation: zoom in maps



Rectangle intersection

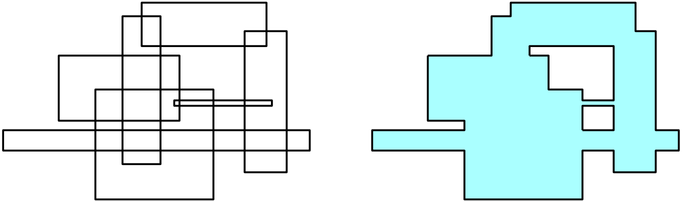
- input: a set B of n isothetic boxes in $\mathbb{R}^2$
- output: all the intersecting pairs in B^2



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

Klee's measure problem

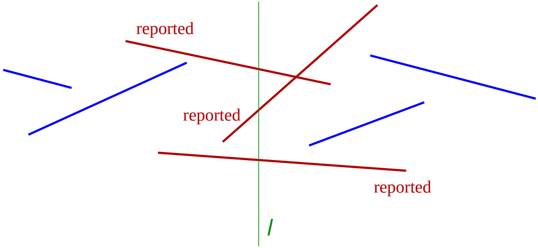
- input: a set B of n isothetic boxes
- output: the area/volume of the union



- well understood in $\mathbb{R}^2 \Rightarrow O(n \log n)$ time
- the union can have complexity $\Theta(n^2)$. Example?
- poorly understood in $\mathbb{R}^d$ for $d > 2$

Segment tree

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



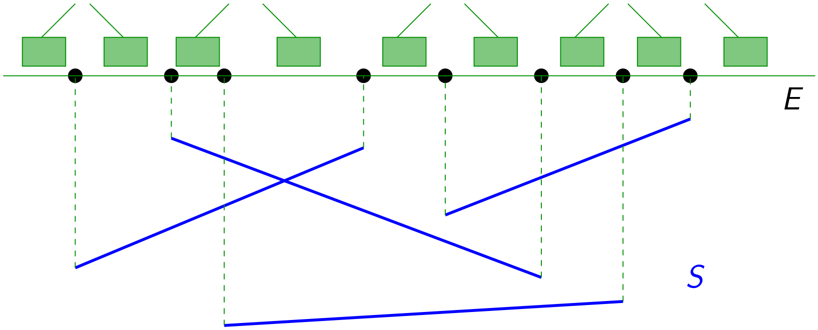
  - query time: $O(\log n + k)$
  - space usage: $O(n \log n)$
  - preprocessing time: $O(n \log n)$

Notations

- let $S = (s_1, s_2, \dots s_n)$ be a set of segments in $\mathbb{R}$
- 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
- $E = \{e_1 < e_2 < \cdots < e_{2n}\}$

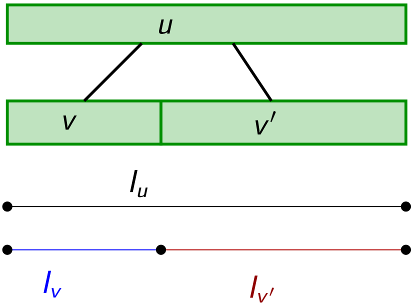
Atomic intervals

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

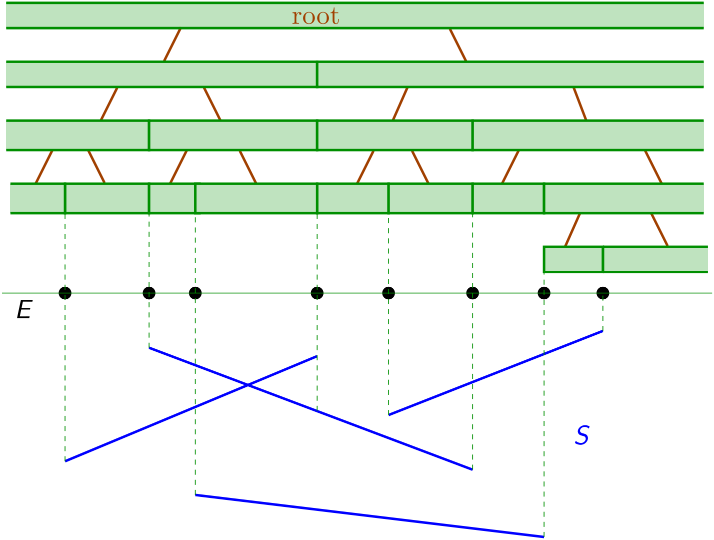


Internal nodes

- 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)

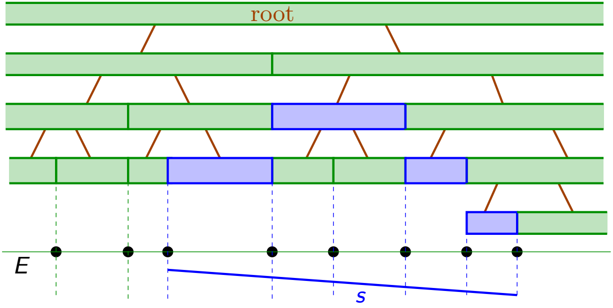


Example



Partitioning a segment

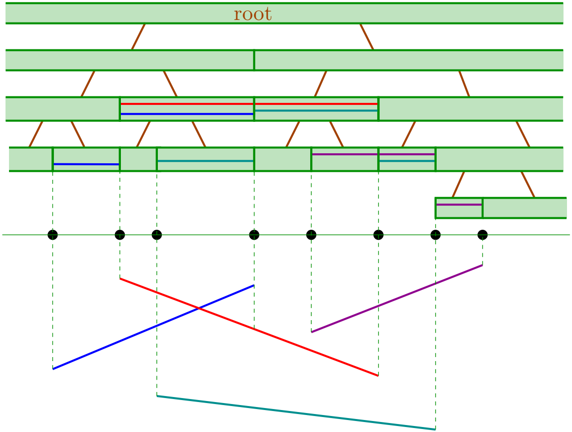
- let $s \in S$ be a segment whose endpoints have x–coordinates e_i and e_j
- $[e_i, e_j]$ is split into several elementary intervals
- they are chosen as close as possible to the root
- s is stored in each node associated with these elementary intervals



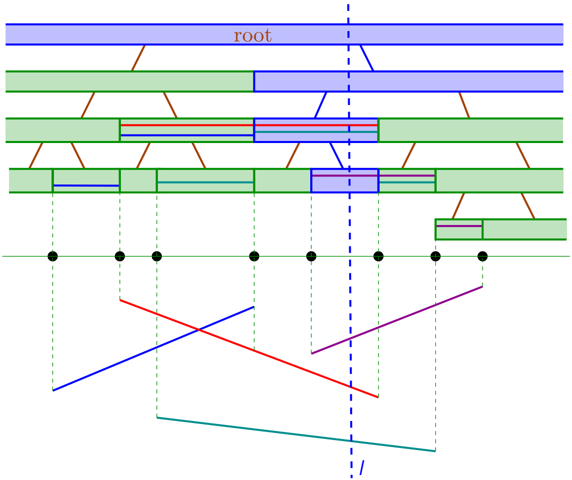
Canonical subsets

- each node u is associated with a canonical subset $S(u)$ of segments
- let $e_i < e_j$ be the x–coordinates of the endpoints of $s \in S$
- then s is stored in $S(u)$ iff $I_u \subset [e_i, e_j]$ and $I_{parent(u)} \not\subset [e_i, e_j]$
- standard segment tree: $S(u)$ is stored as a list pointed from u
- we can also add more structure/data/pointers from u
- useful for multi-level data structures
- we will use it

Example



Answering a stabbing query



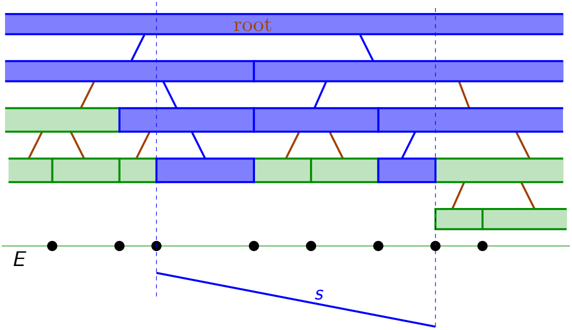
Answering a stabbing query

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

if $u == \mathit{NULL}$
  then return
output $S(u)$ traversing the list pointed from u
if $x_l \in I_{u.left}$
  then $\mathit{ReportStabbing}(u.left, x_l)$
if $x_l \in I_{u.right}$
  then $\mathit{ReportStabbing}(u.right, x_l)$

- 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 $x^- < x^+$

```
if I_u ⊂ [x⁻, x⁺]
    then insert s into the list storing S(u)
    else
        if [x⁻, x⁺] ∩ I_{u.left} ≠ ∅
            then Insert(u.left, s)
        if [x⁻, x⁺] ∩ I_{u.right} ≠ ∅
            then Insert(u.right, s)
```

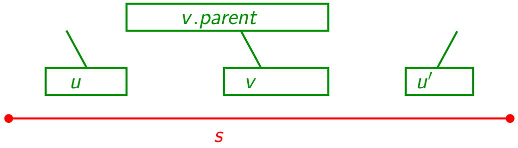
Main Property

Lemma

A segment s is stored at most twice at each level of $\mathcal{T}$.

Dokaz.

- 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

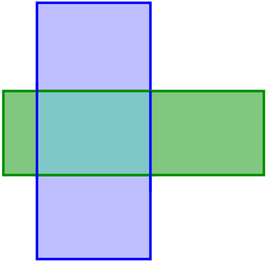
- lemma of previous slide implies
  - each segment stored in $O(\log n)$ nodes
  - space usage: $O(n \log n)$
- insertion in $O(\log n)$ time
  - at most four nodes are visited at each level
- actually space usage is $\Theta(n \log n)$ (example?)
- query time: $O(k + \log n)$
- preprocessing
  - sort 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

- 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, where k is the number of intersecting pairs
- this is optimal. Why?
- note: faster than our line segment intersection algorithm
- space usage: $\Theta(n \log n)$ due to segment trees
- space usage is suboptimal

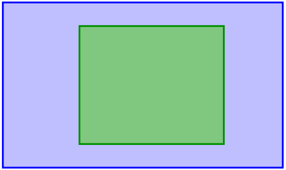
Two kinds of intersections

- overlap



- intersecting edges
- reduces to intersection reporting for isothetic segments
- done as exercise (first homework)

- inclusion



- we can find them using stabbing queries

Reporting overlaps

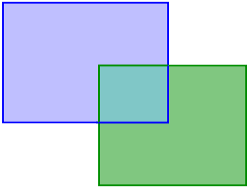
- equivalent to reporting intersecting edges
- plane sweep approach
- sweep line status: BBST containing the horizontal line segments that intersect the sweep line, by increasing y–coordinates
- each time a vertical line segment is encountered, report intersection by range searching in the BBST
- preprocessing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

Reporting inclusions

- also using plane sweep: sweep a horizontal line from top to bottom
- sweep line status: the boxes that intersect the sweep line l, in a segment tree with respect to x–coordinates
  - the endpoints are the x–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?

Stabbing queries for boxes

- 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 data structure, with a segment tree in each level
- inductive definition, induction on d
- first, we store B in a segment tree $\mathcal{T}$ with respect to x_1–coordinate
- for each node u of $\mathcal{T}$, associate a $(d-1)$–dimensional multi–level segment tree for the segments $S(u)$, with respect to $(x_2, x_3 \ldots x_d)$

Performing queries

- search for q in $\mathcal{T}$ using x_1–coordinate
- 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, it follows that
  - query time: $O(k + \log^d n)$
  - space usage: $O(n \log^d n)$
  - preprocessing time : $O(n \log^d n)$
- can be slightly improved...

Windowing queries

- in $\mathbb{R}^d$, a set S of n disjont segments
- for a query axis-aligned rectangle R, find all the segments intersecting R
- three types of segments intersect R:
  - segments with one endpoint inside R
  - segments that intersect vertical side of R
  - segments that intersect horizontal side of R
- first type: range tree over the endpoints of the segments
- second type: multi-level data structure with segment tree
  - store S in a segment tree $\mathcal{T}$ with respect to x–coordinate
  - for each node u of $\mathcal{T}$, store the segments $S(u)$ sorted by their intersection with vertical line in BST

Windowing queries

- for segments of the second type:
  - a query visits $O(\log n)$ nodes of the main tree
  - the canonical subsets of those nodes are disjoint
  - in each node we spend $O(\log n)$ time, plus time to report segments (1d range-tree)
  - each segment is reported once, because disjointness
- each segment reported at most twice: filter them
- For n disjoint segments:
  - preprocessing: $O(n \log^2 n)$ time
  - query: $O(k + \log^2 n)$ time
- where did we use that the segments are disjoint?

Klee's measure problem

- in $\mathbb{R}^2$, a set S of n axis-parallel rectangles
- compute area of the union
- solution using $O(n \log n)$ time
- sweep a vertical line ℓ from left to right
  - keep the length of $\ell \bigcap (\bigcup S)$
  - events: length changes when rectangles start or stop intersecting ℓ
  - relevant values: distance between consecutive events and the length
  - we compute the area to the left of ℓ, updating it at each event
- use segment trees to maintain the length
- http://www.cgl.uwaterloo.ca/~krmoule/courses/cs760m/klee

Klee's measure problem

- we need to maintain the length of union of intervals under insertion and deletion of intervals
- make a segment tree (we know all endpoints in advance)
- at each node u we store
  - list of $S(u)$ (actually its cardinality is enough)
  - $length(u)$: the length of I_u covered by segments stored below u
  - note that $length(u)$ only depends on subtree rooted at u
  - this allows quick updates
- $length(root)$ is the real length we want
- insertion or deletion of interval takes $O(\log n)$ time
  - if $S(u) \neq \emptyset$, then $length(u) = length(I_u)$
  - else, $length(u) = length(u.left) + length(u.right)$

Klee's measure problem

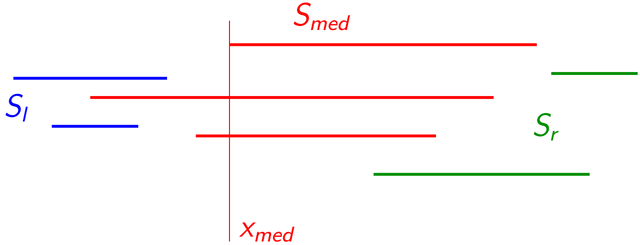
- in $\mathbb{R}^3$ best known algorithm in $O(n^{3/2})$ time
- only lower bound: $\Omega(n \log n)$
- in $\mathbb{R}^3$, recent progress for unit boxes

Interval trees

- interval trees allow to perform stabbing queries in one dimension
  - query time: $O(k + \log n)$
  - preprocessing time: $O(n \log n)$
  - space: $O(n)$
- based on different approach

Preliminary

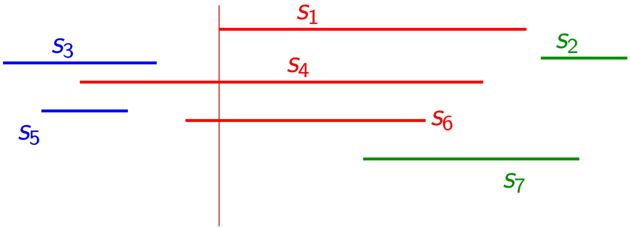
- let x_{med} be the median of E
  - S_l: segments of S that are completely to the left of x_{med}
  - S_{med}: segments of S that contain x_{med}
  - S_r: segments of S that are completely to the right of x_{med}



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
- pseudocode