KDS

(Kinetic Data Structures)

See slides for:
- motivation (DSP, Bbox)
- general concepts of KDS
- example using convex hull
- definitions used in KDS
- main loop
- performance measures for KDS
- tournament tree as building block for kinetic Bbox
- kinetic BVH, kinetic separation list for collision detection

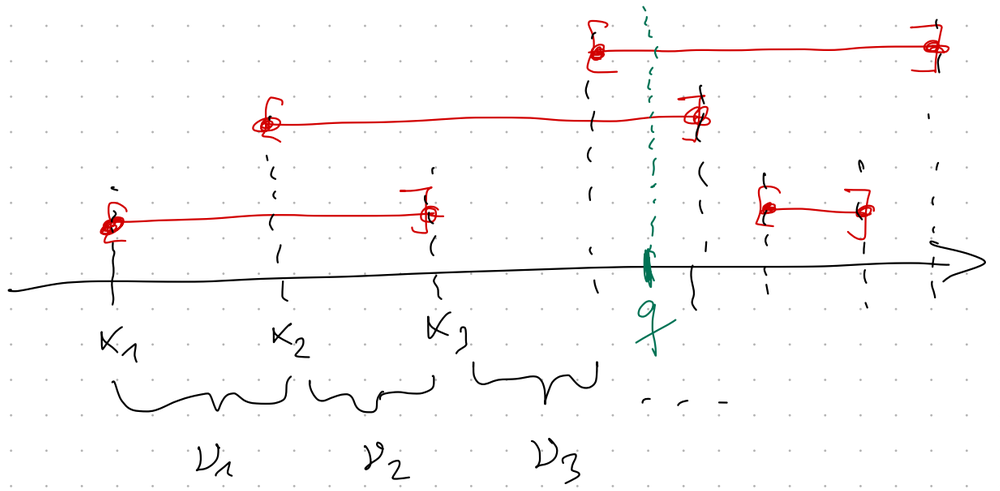
Kinetic Segment Tree

Def.: stabbing query

Given: set of intervals $S = \{s_1, s_2, \dots, s_n\}$, $s_i = [a_i, b_i] \subseteq \mathbb{R}$

query pt $q \in \mathbb{R}$

Sought: $S' = \{ s \in S \mid q \in s \}$



Def.: elementary interval (EI)

Let $x := \{ a_i, b_i \mid i = 1, \dots, n \}$

$= \{ x_i \}_{2n}$

Intervals $[x_i, x_{i+1}]$ are the EI's.

For practical reasons, include $(-\infty, x_0)$ and $(x_{2n}, +\infty)$

Segment tree: (static)

balanced binary tree over all EI's,

leaves v_i store the EI $int(v_i) = [x_i, x_{i+1}]$;

interval of inner node v is $int(v) := int(v_1) \cup int(v_2)$,

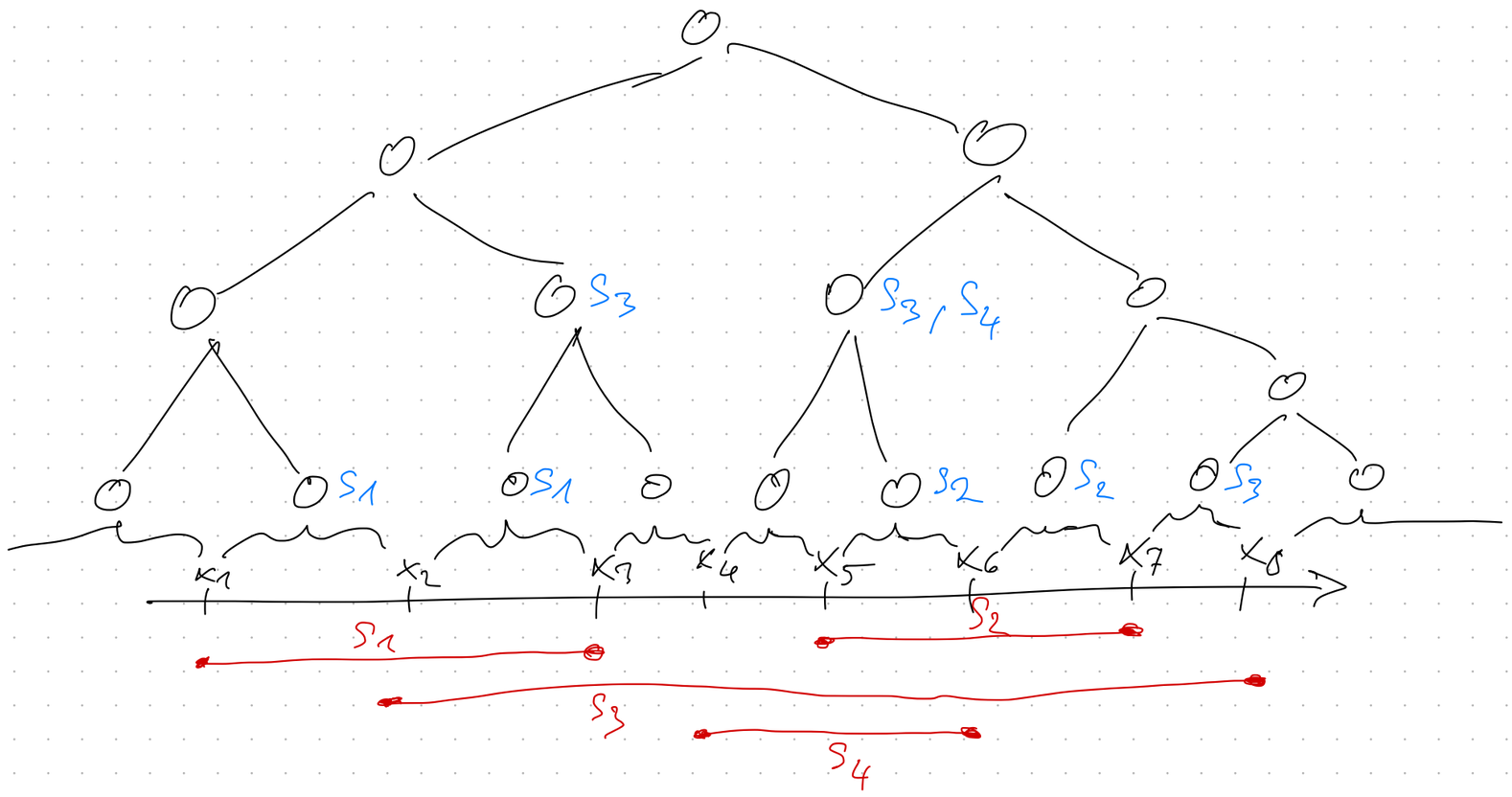
where v_1, v_2 are v's children.

each node stores

$$S(v) := \{ s \in S \mid int(v) \subseteq s,\ int(parent(v)) \not\subseteq s \}$$

↑ "canonical set" of v

Example:



Algo for construction of segment tree:

Sort endpts in X

construct skeleton tree over $2n+1$ leaves

for all nodes v, bottom up: compute $int(v)$

for all segments $s \in S$: insert s down through tree

Algo for stabbing query:

query (v, q):

output $S(v)$

if v is leaf: return

if $q \in int(v_1)$: query (v_1, q)

else: query (v_2, q)

return

Theorem:

For any set of n segments, a segment tree can answer stabbing queries in time $O(k + \log n)$, with $O(n \log n)$ space and $O(n \log n)$ construction time.
(k = #output segments)

Kinetic Segment tree:

Augmented ST:

- Store endpts in array $R[0, \dots, 2n-1]$, $R[i] = x_i$
- Denote segments $s_i = (a_i, b_i)$, where $a_i, b_i \in \mathbb{N}$, $0 \le a_i, b_i \le 2n-1$
  indices into R
  so real segments are given by $s_i = [R[a_i], R[b_i]]$ "=" (a_i, b_i)
  a_i / b_i are called "ranks" of endpts
- Maintain list $\mathcal{L}(S) := \{ v \mid s \in S(v) \}$ "fragment list"
  order left to right

- Array A with pointers to v_i leaves for EI $(i, i+1)$

Certificates: $R[j] < R[j+1]$ ← stored in p-queue, sorted by failure time

failed certificate → two intervals affected

before: $s = (i, j)$, after: $s = (i, j+1)$

(or any other of 4 cases)

other interval s': before $s' = (j+1, k)$, after $s' = (j, k)$

Naive update algo: $T \in O(\log n)$

Update algo of kinetic ST:

(extend segment $s = (i, j)$ by EI $[j, j+1]$)

init $v := v_j$ (leaf for EI $(j, j+1)$)

$\mu :=$ brother of v

while $s \in S(\mu)$: (*)

delete s from $S(\mu)$

$v := \text{parent}(v)$

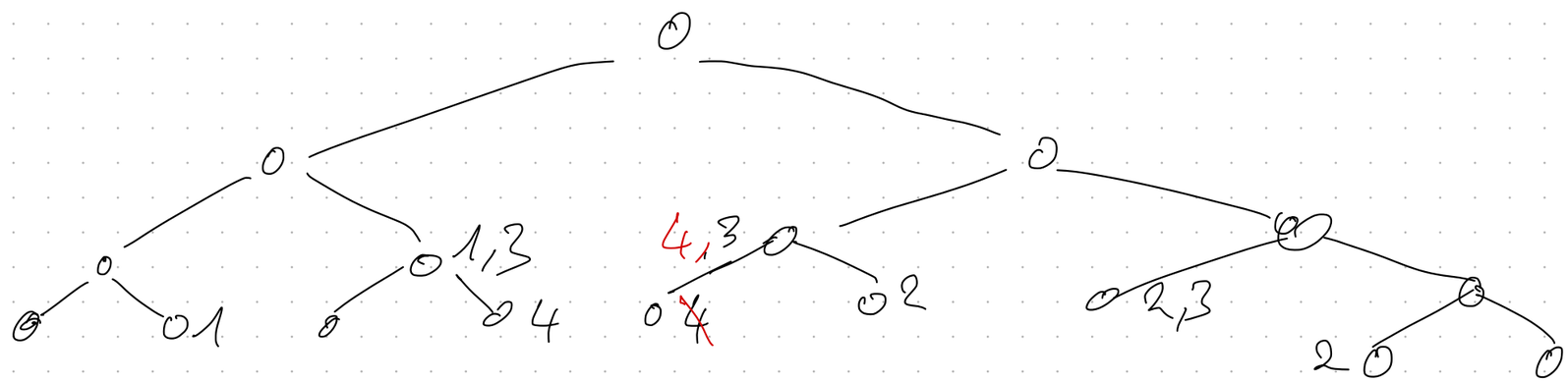
μ = brother of v
end while
add s to $S(v)$

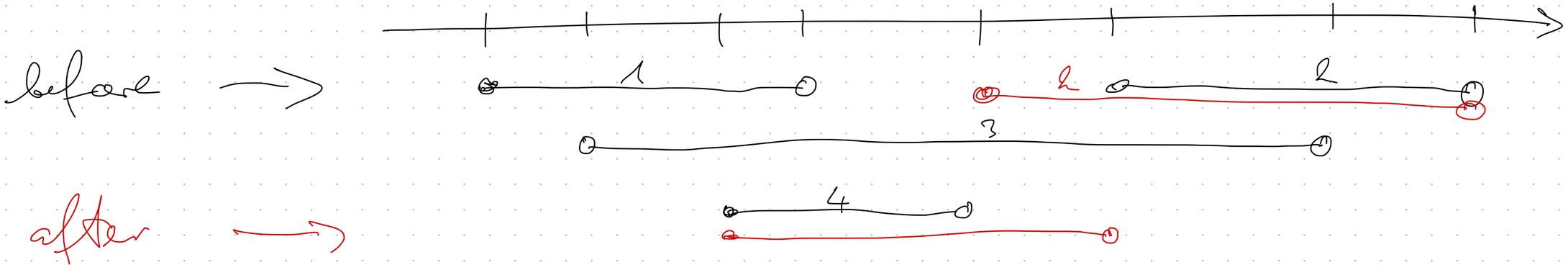
Note on *) cannot be fulfilled, if v is the left child of its parent

Time $\in O(h)$, where h = height of node v where walk upwards stops, i.e., where $(i, i+1) \subseteq \text{int}(v)$ and $s \in S(v)$ after update!

Proof: time for test (*) $\in O(1)$
with help of $\mathcal{L}(S)$, which is sorted left to right
(or use hash table)

Example:





Lemma:

Given set S of n moving segments $\subseteq \mathbb{R}$.
There is a kinetic segment tree with $O(n \log n)$ size,
which can be updated in expected time $O(1)$ in case
of a certificate failure (external event);
worst-case update time is $O(\log n)$.
The KST is local and efficient.

Proof:

Expected time: Def $h(j) :=$ height of highest node v, s.t.
$(j, j+1)$ is rightmost end of $int(v)$.

$h \leq h(j)$

Claim $\text{avg } h := \frac{1}{2^m} \sum_{j=0}^{2^m} h(j) \in O(1) \Rightarrow$ expected running time

Proof:

$j = even \rightarrow h(j) = 0$

$j = 2^m - 1 \rightarrow h(j) = m$

$$\overline{h} = \frac{1}{2n}\left(\frac{2n}{2}\cdot 0 + \frac{2n}{4}\cdot 1 + \dots + \frac{2n}{2n}\cdot \log n\right)$$

$$= \frac{1}{2}\sum_{i=0}^{\log n} i\left(\frac{1}{2}\right)^i < \frac{1}{2}\sum_{0}^{\infty} i\left(\frac{1}{2}\right)^i = 1$$

$$\sum_{0}^{\infty} i z^i = \frac{z}{(1-z)^2}, \quad z \in (0,1)$$