CS 240 – Data Structures and Data Management

Module 8: Range-Searching - Enriched

T. Biedl É. Schost O. Veksler

Based on lecture notes by many previous cs240 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Winter 2021

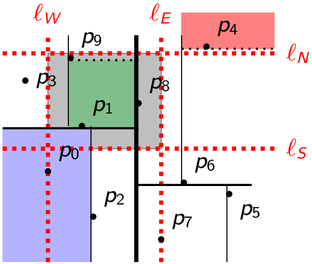
version 2021-03-01 13:08

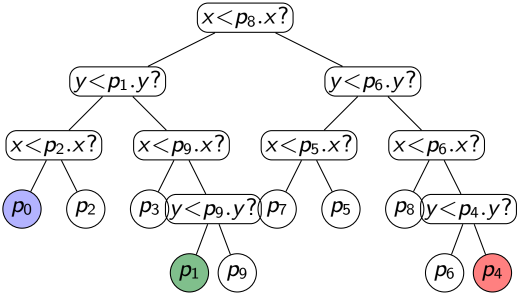
Outline

1. Boundary nodes in kd-trees
2. 3-sided range search

Bounday nodes in kd-trees

Recall: $Q(n)$ are the boundary-nodes (blue).
Goal: $Q(n) \in O(\sqrt{n})$.

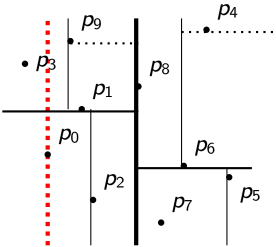


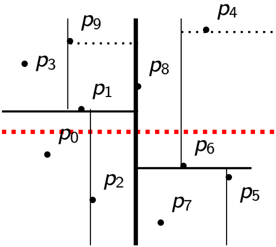


Observation: If v is a boundary-node, then its associated region intersects one of the lines $\ell_W, \ell_N, \ell_E, \ell_S$ that support the query-rectangle.

Boundary nodes in kd-trees

$$Q(n,\ell) := \max_{\text{kd-trees with } n \text{ points}} \quad \begin{array}{l}\text{number of associated regions} \\ \text{that intersect a given line } \ell\end{array}$$



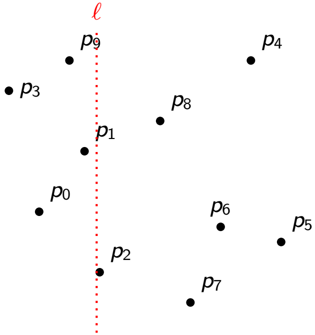


This is independent of ℓ (shift points), so only consider whether ℓ is horizontal or vertical $\rightsquigarrow$ $Q_v(n), Q_h(n)$

$$\begin{aligned} Q(n) &\leq Q(n,\ell_W) + Q(n,\ell_N) + Q(n,\ell_E) + Q(n,\ell_S) \\ &\leq 2Q_v(n) + 2Q_h(n) \end{aligned}$$

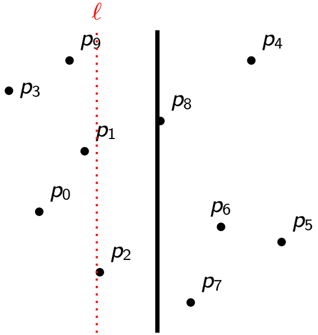
Boundary nodes in kd-trees

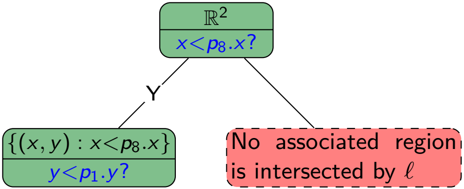
Goal: $Q_v(n) \leq 2Q_v(n/4) + 2.$



Boundary nodes in kd-trees

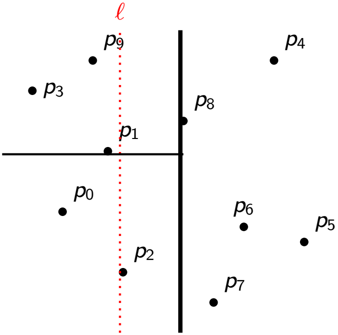
Goal: $Q_v(n) \leq 2Q_v(n/4) + 2.$

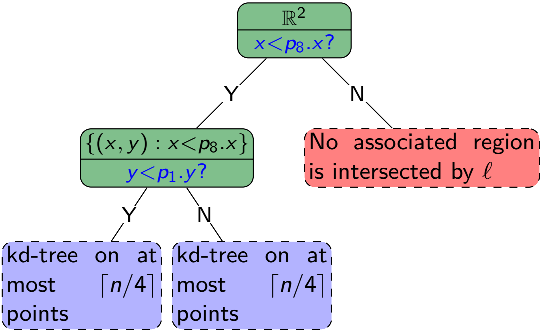




Boundary nodes in kd-trees

Goal: $Q_v(n) \leq 2Q_v(n/4) + 2.$





Boundary nodes in kd-trees

- $Q_v(n) \leq 2Q_v(n/4) + 2$ $\Rightarrow Q_v(n) \in O(\sqrt{n})$

Boundary nodes in kd-trees

- $Q_v(n) \leq 2Q_v(n/4) + 2 \qquad \Rightarrow Q_v(n) \in O(\sqrt{n})$
- Similarly: $Q_h(n) \leq 2Q_h(n/4) + 3 \qquad \Rightarrow Q_h(n) \in O(\sqrt{n})$

Boundary nodes in kd-trees

- $Q_v(n) \leq 2Q_v(n/4) + 2 \quad \Rightarrow Q_v(n) \in O(\sqrt{n})$
- Similarly: $Q_h(n) \leq 2Q_h(n/4) + 3 \quad \Rightarrow Q_h(n) \in O(\sqrt{n})$

- $Q(n) \leq 2Q_v(n) + 2Q_h(n) \in O(\sqrt{n})$

Theorem: In a range-query in a kd-tree (of points in general position) there are $O(\sqrt{n})$ boundary-nodes.

Boundary nodes in kd-trees

- $Q_v(n) \leq 2Q_v(n/4) + 2 \quad \Rightarrow Q_v(n) \in O(\sqrt{n})$
- Similarly: $Q_h(n) \leq 2Q_h(n/4) + 3 \quad \Rightarrow Q_h(n) \in O(\sqrt{n})$

- $Q(n) \leq 2Q_v(n) + 2Q_h(n) \in O(\sqrt{n})$

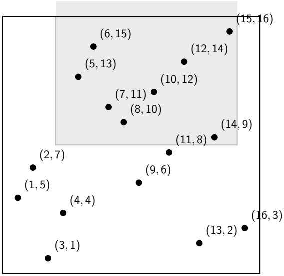
Theorem: In a range-query in a kd-tree (of points in general position) there are $O(\sqrt{n})$ boundary-nodes.

- So range-search takes $O(\sqrt{n} + s)$ time.
- Note: It is *crucial* that we have $\approx n/4$ points in each grand-child of the root.

3-sided range search

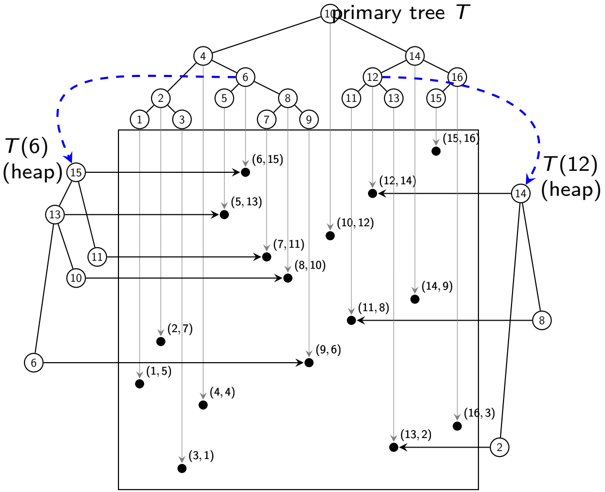
Consider a special kind of range-search:

3sidedRangeSearch(x_1, x_2, y'): *return* (x, y) *with* $x_1 \le x \le x_2$ *and* $y \ge y'$.



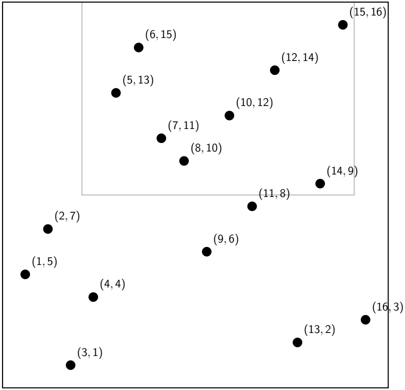
Can we adapt previous ideas to achieve $O(n)$ space *and* fast range-search time?

Idea 1: Associated heaps

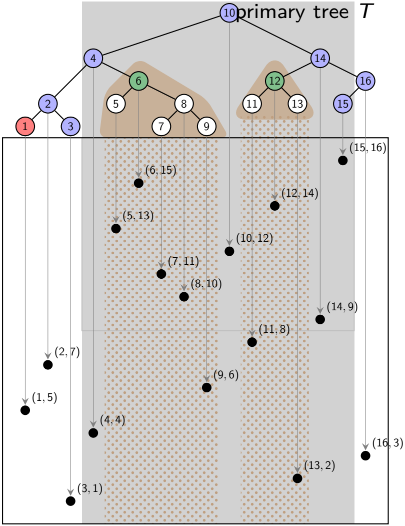


- Primary tree: balanced binary search tree.
- Associated tree: binary heap.
- Space: $\Theta(n \log n)$.
- Range-search time?

Idea 1: Associated heaps - 3-sided range search

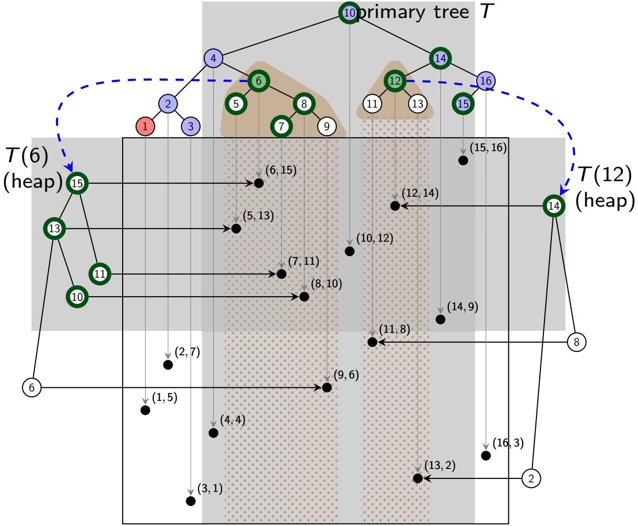


Idea 1: Associated heaps - 3-sided range search



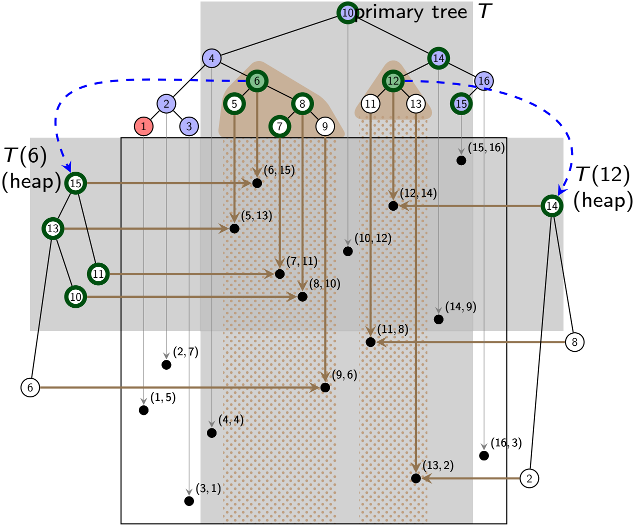
- Search in primary as before.

Idea 1: Associated heaps - 3-sided range search



- Search in primary as before.
- In associated heap: Search by y-coordinate in $O(1 + s)$ time. (Exercise.)

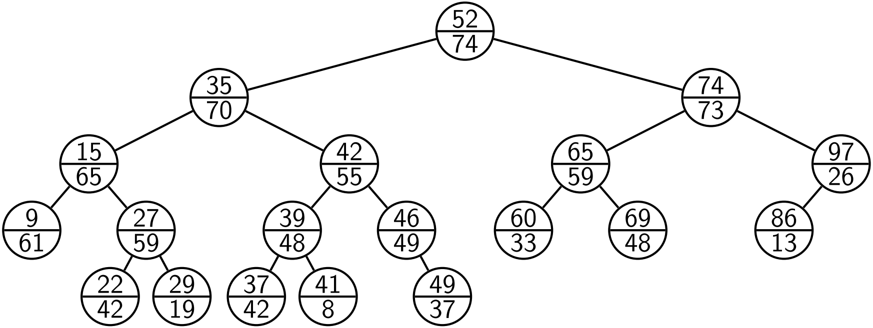
Idea 1: Associated heaps - 3-sided range search



- Search in primary as before.
- In associated heap: Search by y-coordinate in $O(1+s)$ time. (Exercise.)
- Total time: $O(\log n + s)$
- But space is $\omega(n)$

Idea 2: Treaps

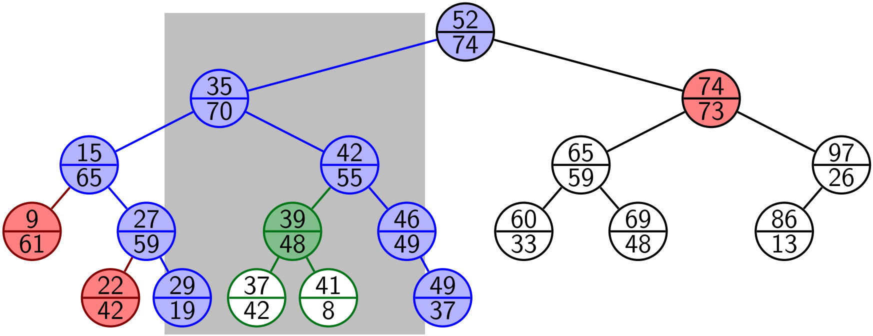
Recall: Treap = binary search tree (with respect to keys)
+ heap (with respect to priorities)



Idea: Use x-coordinate as key, y-coordinate as priority.
Space: $\Theta(n)$.

Idea 2: Treaps - 3-sided range search

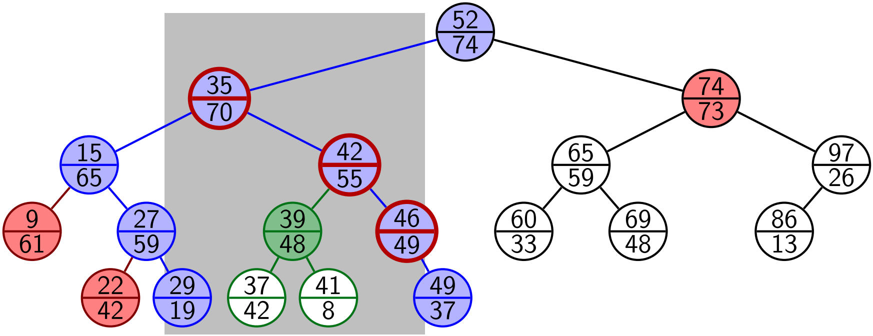
Treap::3-sided-range-search$(T, 28, 47, 36)$:



- BST::range-search(x_1, x_2) to get boundary and topmost inside nodes.

Idea 2: Treaps - 3-sided range search

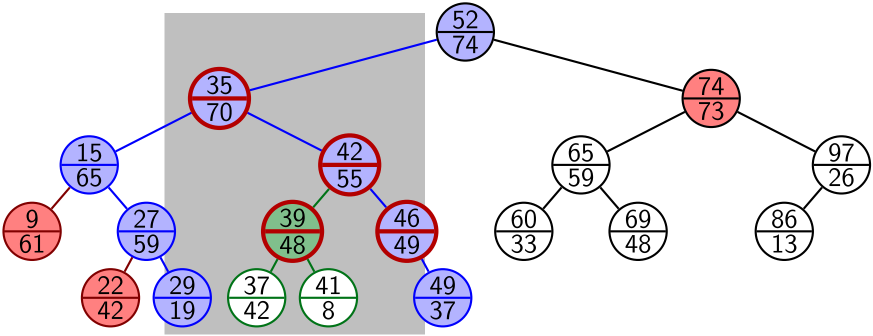
Treap::3-sided-range-search$(T, 28, 47, 36)$:



- BST::range-search(x_1, x_2) to get boundary and topmost inside nodes.
- Boundary-nodes: Explicitly test whether in x-range and y-range.

Idea 2: Treaps - 3-sided range search

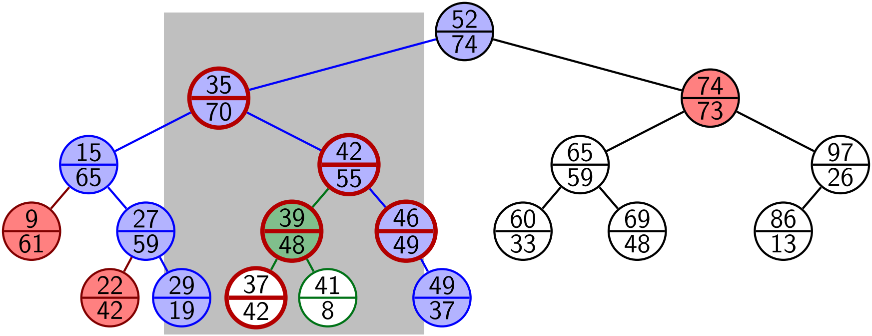
Treap::3-sided-range-search$(T, 28, 47, 36)$:



- BST::range-search(x_1, x_2) to get boundary and topmost inside nodes.
- Boundary-nodes: Explicitly test whether in x-range and y-range.
- Topmost inside-nodes: If $y \geq y_1$, report and recurse in children.

Idea 2: Treaps - 3-sided range search

Treap::3-sided-range-search(T, 28, 47, 36) :



- BST::range-search(x_1, x_2) to get boundary and topmost inside nodes.
- Boundary-nodes: Explicitly test whether in x-range and y-range.
- Topmost inside-nodes: If $y \geq y_1$, report and recurse in children.

Idea 2: Treaps - 3-sided range search

Run-time for 3-sided range search in treaps:

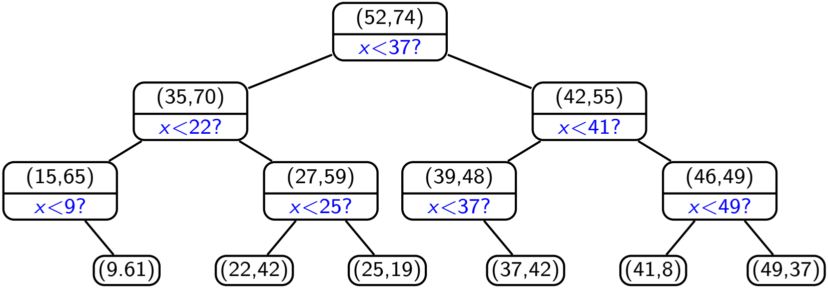
- BST::range-search(x_1, x_2) — $O(\mathit{height})$ since we do not report points.
- Testing boundary-nodes: $O(\mathit{height})$
- Testing heap: $O(1 + s_v)$ per topmost inside-node v

$\Rightarrow O(\mathit{height} + s)$ run-time, $O(n)$ space

But: No guarantees on the height of the treap (not even in expectation) since we cannot choose priorities.

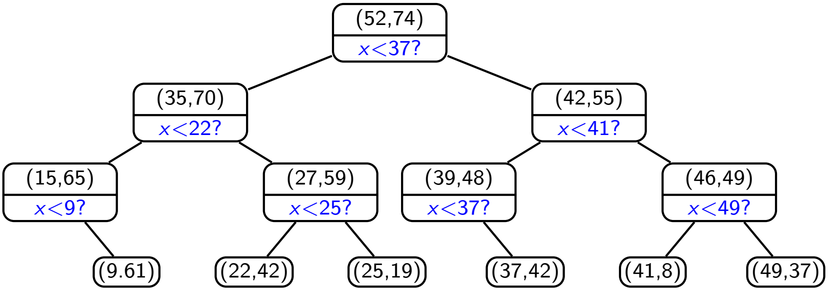
Idea 3: Priority search trees

- Design a new data structure
- Keep good aspects of treap (store y-coordinates in heap-order)
- Keep good aspects of kd-tree (split in half by x-coordinate)



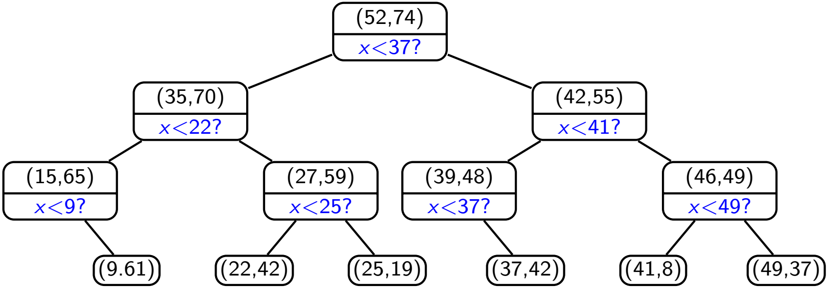
Key idea: The x-coordinate stored for splitting can be *different* from the x-coordinate of the stored point.

Idea 3: Priority search trees



- every node v stores a point $p_v = (x_v, y_v)$,
  - y_v is the maximum y-coordinate in subtree (heap-property!)
- every non-leaf v stores an x-coordinate x'_v (split-line)
  - Every point p in left subtree has $p.x < x'_v$
  - Every point p in right subtree has $p.x \geq x'_v$
- x'_v is chosen so that tree is balanced $\Rightarrow$ height $O(\log n)$.

Idea 3: Priority search trees



- Construction: $O(n \log n)$ time (exercise)
- *search*: $O(\log n)$ time
  - Get search-path by following split-lines, check all nodes on path
- *insert*, *delete*: Re-balancing is difficult, but can be done (no details).
- 3-sided range search: As in treaps, but height now $O(\log n)$.
  - Run-time $O(\log n + s)$

3-sided range search summary

- Idea 1: Scapegoat tree + associated heaps
  $O(\log n + s)$ time for range search, but $\omega(n)$ space.
- Idea 2: Treaps
  $O(n)$ space, but range search takes $O(\mathit{height} + s)$, could be slow
- Idea 3: Priority search tree
  $O(n)$ space, $O(\log n + s)$ time for range search.

Sometimes it pays to design purpose-built data structures.