

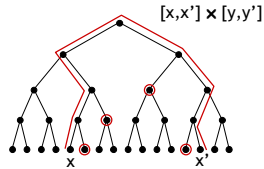
Fractional Cascading

For range trees the query time is $O(\log^2 n + k)$.

We improve this by using fractional cascading.

Main idea: Perform queries in the associated structures in time $O(1+k)$, instead of $O(\log n+k)$.

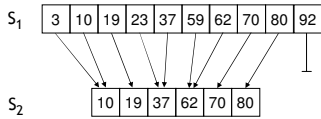
In general this is not possible but we will do many 1D searches with the same range.



Fractional Cascading - example

Consider two arrays S_1 and S_2 , where S_2 is a "subset" of S_1 .

Can we preprocess S_1 and S_2 , such that we can perform $[y, y']$ range query faster than $2 \log n$?

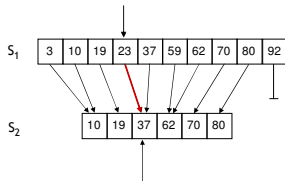


For each entry x in S_1 , store a pointer to the smallest entry in S_2 that is greater than or equal to x .

Fractional Cascading - example

Query: Report all objects in S_1 and S_2 in $[y, y']$.

[20, 65]

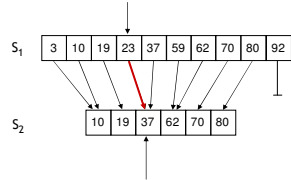


1. Find first entry in S_1 greater than or equal to y (denoted z) by binary search.
2. Walk through S_1 until we find a value greater than y' .
3. Follow pointer from z to entry in S_2 .
4. Walk through S_2 until we find a value greater than y' .

Fractional Cascading - example

Query time?

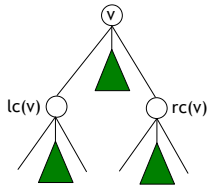
1. Binary search $O(\log n)$
2. Traverse S_1 $O(k)$
3. Follow pointer $O(1)$
4. Traverse S_2 $O(k)$



4

Fractional Cascading

How can we apply this idea to range trees?



Observations:

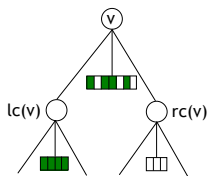
$A(lc(v))$ contains a subset of the points in $A(v)$.

$A(rc(v))$ contains a subset of the points in $A(v)$.

5

Fractional Cascading

How can we apply this idea to range trees?



Observations:

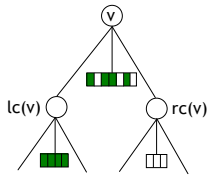
$A(lc(v))$ contains a subset of the points in $A(v)$.

$A(rc(v))$ contains a subset of the points in $A(v)$.

6

Fractional Cascading

How can we apply this idea to range trees?

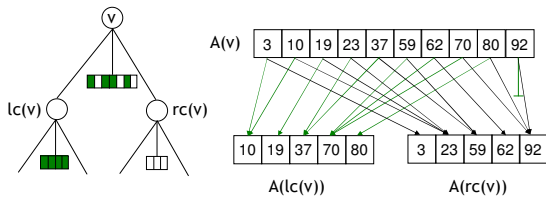


For each entry y in $A(v)$ add two pointers:

1. a pointer to the key in $A(lc(v))$ with the smallest y -value greater than or equal to y .
2. a pointer to the key in $A(rc(v))$ with the smallest y -value greater than or equal to y .

Fractional Cascading

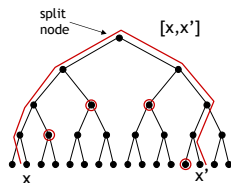
How can we apply this idea to range trees?



Fractional Cascading

Assume we have a query $[x, x'] \times [y, y']$.

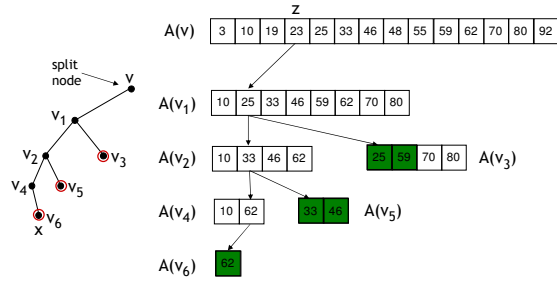
Perform a 1D search in T for x and x' .



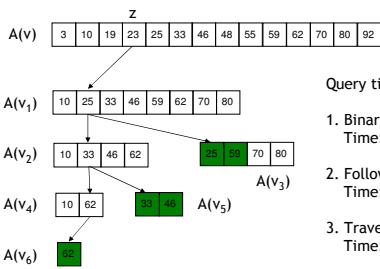
$O(\log n)$ nodes whose canonical subset contain the points with x -coordinate in $[x, x']$.

Fractional Cascading

Query $[x, x'] \times [y, y'] = [x, x'] \times [20, 65]$.



Fractional Cascading



Query time?

1. Binary search in $A(v)$
Time: $O(\log n)$
 2. Follow pointers to $A(v_2), A(v_3), \dots$
Time: $O(1)/\text{pointer} = O(\log n)$
 3. Traverse the sets
Time: $O(k)$
- Total: $O(\log n + k)$

Fractional Cascading

2-dimensions

Query time: $O(\log n + k)$

Preprocessing: $O(n \log n)$

Space: $O(n \log n)$

d-dimensions

Query time: $O(\log^{d-1} n + k)$

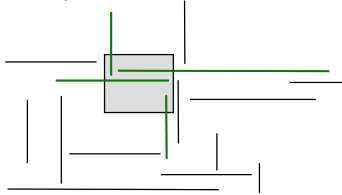
Preprocessing: $O(n \log^{d-1} n)$

Space: $O(n \log^{d-1} n)$

Interval trees

Input: A set of n axis-parallel line segments $S = \{s_1, s_2, \dots, s_n\}$ in the plane.

Aim: Preprocess S such that orthogonal range queries can be handled efficiently.

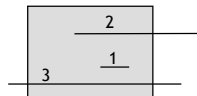


Interval trees

How can a segment intersect a window W ?

- 1. Entirely inside W
- 2. One endpoint inside W
- 3. No endpoint inside W

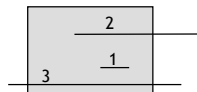
Use range trees!



Interval trees

Cases 1 and 2 can be handled using $O(n \log n)$ space and preprocessing, and $O(\log n + k)$ query time.

Case 3: No endpoint inside W .



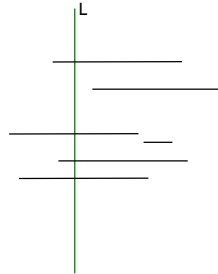
Interval trees

Consider a simplified version.

Input: Set S of n intervals on x -axis ("horizontal segments").

Query: Given a point L on the x -axis, report all intervals containing L ("horizontal segments intersecting a vertical line L ").

Observation: 1D query



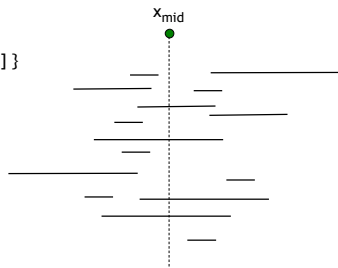
Interval trees

Input: Set S of n intervals.

$$S = \{ [x_1, x_1'], \dots, [x_n, x_n'] \}$$

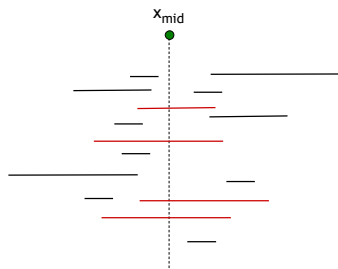
Construct a binary tree, recursively.

x_{mid} is the median of the $2n$ endpoints.



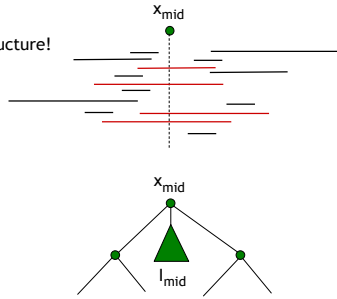
Interval trees

What do we do with the intervals that contain x_{mid} ?



Interval trees

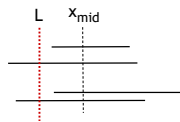
Idea:
Use an associated data structure!



Interval trees

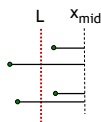
How do we answer a query in the associated data structure?

Aim: $O(k)$



Is this the same problem as the original problem?

No. Only the left endpoints are of any interest now.

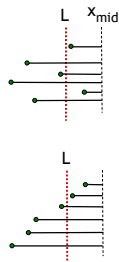


Interval trees

Idea:

Store the intervals twice:
-once sorted w.r.t. the left endpoints
-once sorted w.r.t. the right endpoints

Query?



Interval trees

Build tree:

1. If S is empty then tree is a leaf
2. Otherwise let x_{mid} be the median of the endpoints of S .
 $I_{left} = \{\text{set of intervals to the left of } x_{mid}\}$
 $I_{right} = \{\text{set of intervals to the right of } x_{mid}\}$
 $I_{mid} = \{\text{set of intervals intersecting } x_{mid}\}$
3. The tree is a root v storing x_{mid} and
 - left child is an interval tree storing I_{left}
 - right child is an interval tree storing I_{right}
 - I_{mid} is stored in two sorted lists; once w.r.t. left end point and once w.r.t. right endpoint

Space: $O(n)$
 Construction time: $O(n \log n)$

22

Interval trees

Query time:

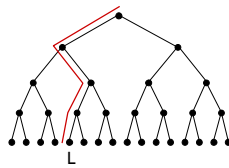
Find search path = $O(\log n)$

For each node v along the path, query the associated data structure $A(v)$.

$O(\log n)$ structures
 $O(k)$ time for all structures

Total time: $O(\log n + k)$

Query: L



23

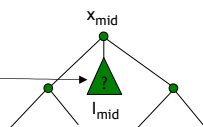
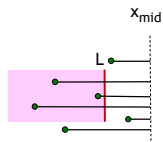
Interval trees

Extension to vertical query segments:

Report all points in the unbounded rectangle.

⇒ That's a 2D orthogonal range query!

Use a 2D range tree as the associated data structure.

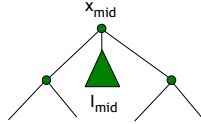


24

Interval trees

The associated structure:

- $O(m \log m)$ space
- $O(m \log m)$ construction time
- $O(\log m + k)$ query time



Total:

- $O(n \log n)$ space
- $O(n \log n)$ construction time
- $O(\log^2 n + k)$ query time

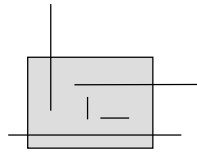
$(O(\log n) \text{ nodes} \times O(\log n + k) \text{ time/node})$

Interval trees

How can a segment intersect a window W?

- Entirely inside W
- One endpoint inside W

Range trees:
 $O(\log n + k)$ query time
 $O(n \log n)$ space and preprocessing



- No endpoint inside W

Interval trees:
 $O(\log^2 n + k)$ query time
 $O(n \log n)$ space and preprocessing

Range queries

Points

- kd-trees
- Range trees + fractional cascading

Intervals

- Interval trees

(If we use Priority search trees in the associated data structures then only $O(n)$ space)
