Amirkabir University of Technology
(Tehran Polytechnic)

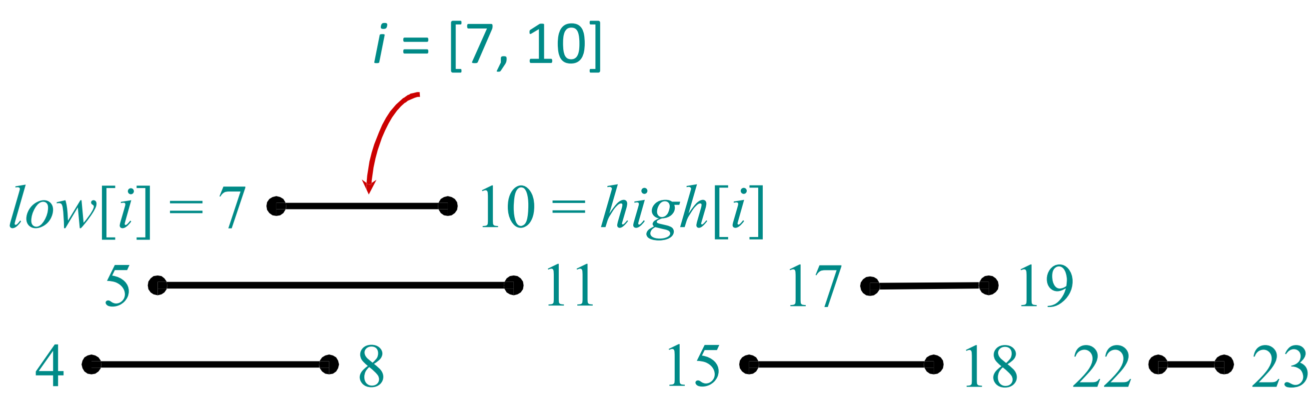
Department of
Computer Engineering

Data Structure & Algorithms

Interval Trees

Interval trees

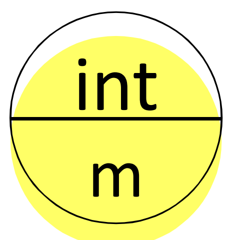
Goal: To maintain a dynamic set of intervals, such as time intervals.



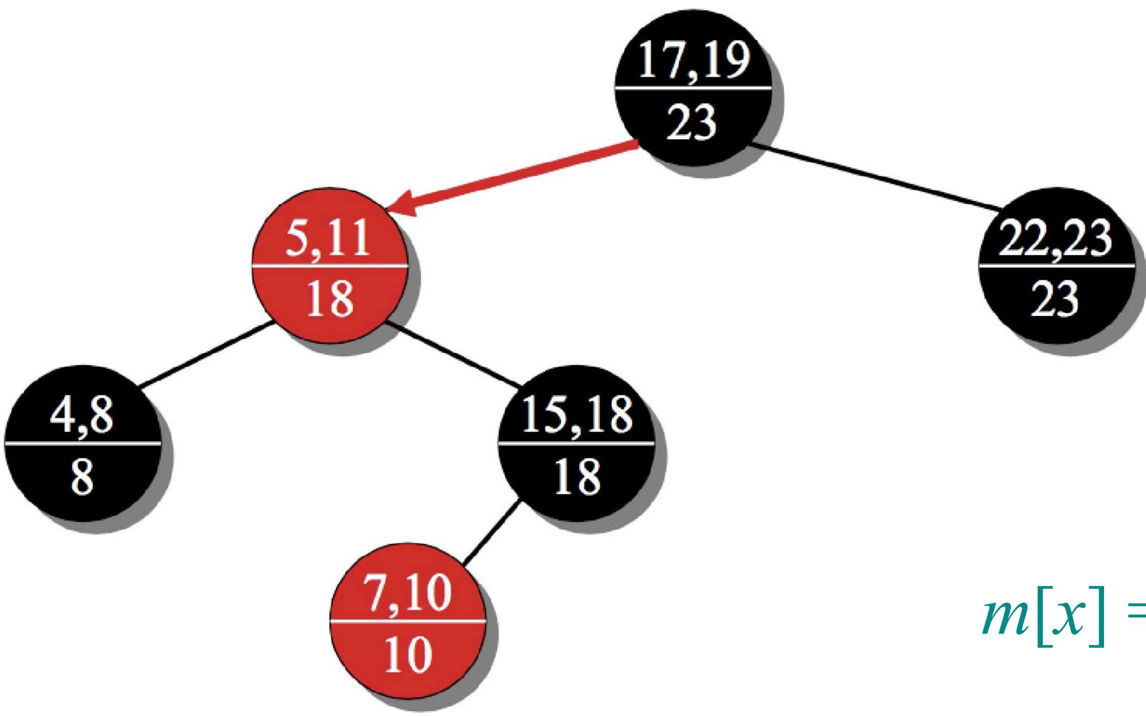
Query: For a given query interval *i*, find an interval in the set that overlaps *i*.

Following the methodology

1. *Choose an underlying data structure.*
   - Red-black tree in which each node x contains an interval *int*[*x*] and the key of x is the low endpoint of the interval.
2. *Determine additional information to be stored in the data structure.*
   - Store in each node *x the value m*[*x*] which is the maximum value of any interval endpoint stored in the subtree rooted at *x*.



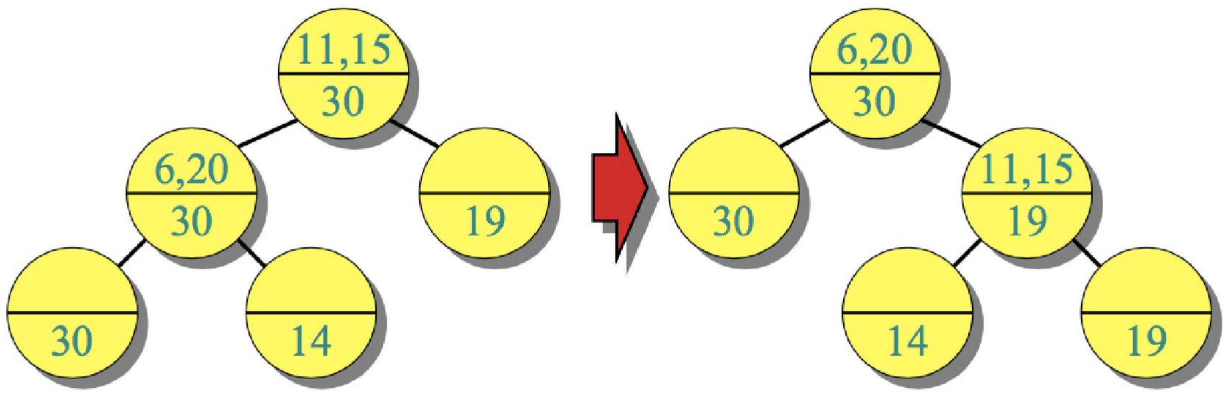
Example interval tree



$$m[x] = \max \begin{cases} high[int[x]] \\ m[left[x]] \\ m[right[x]] \end{cases}$$

Modifying operations

3. *Verify that this information can be maintained for modifying operations.*
   - INSERT: Fix m's on the way down.
   - Rotations — Fixup = $O(1)$ time per rotation:



Total INSERT time = $O(\lg n)$; DELETE similar.

New operations

4. Develop new dynamic-set operations that use the information.

INTERVAL-SEARCH(T, i)

$x \leftarrow$ T.*root*

while $x \neq$ T.nil and ($low[i] > high[int[x]]$ or $low[int[x]] > high[i]$)

// i and $int[x]$ don't overlap

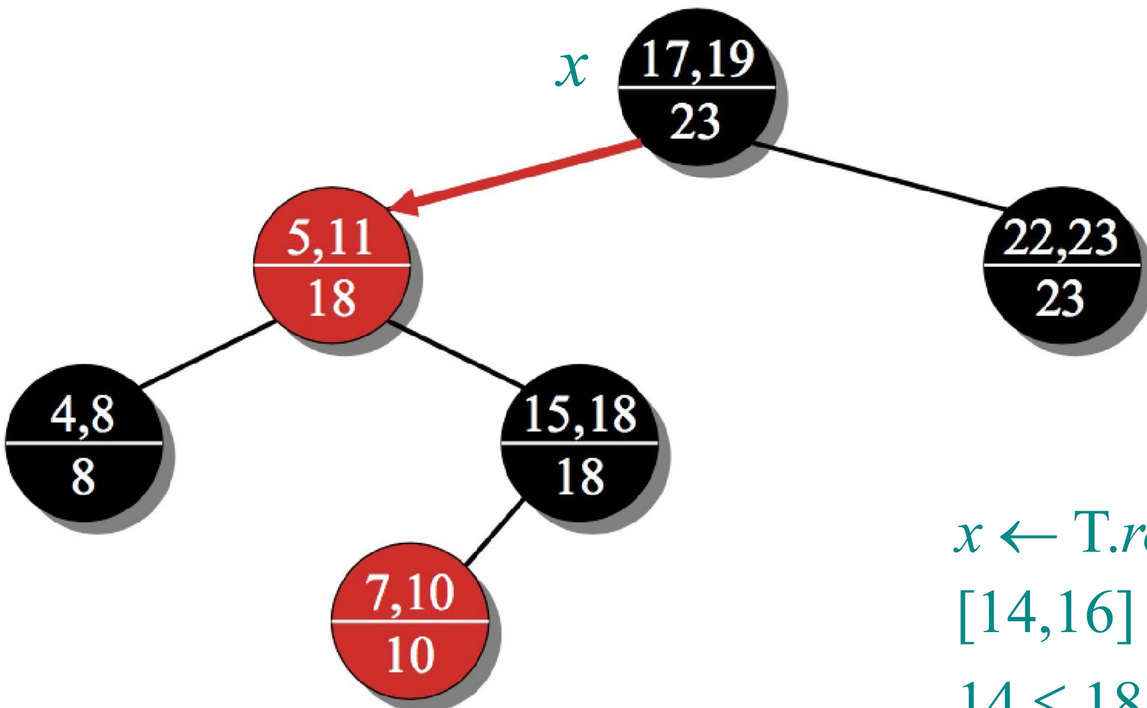
if $left[x] \neq$ T.nil and $low[i] \leq m[left[x]]$

then $x \leftarrow left[x]$

else $x \leftarrow right[x]$

return x

Example 1: INTERVAL-SEARCH(T, [14,16])

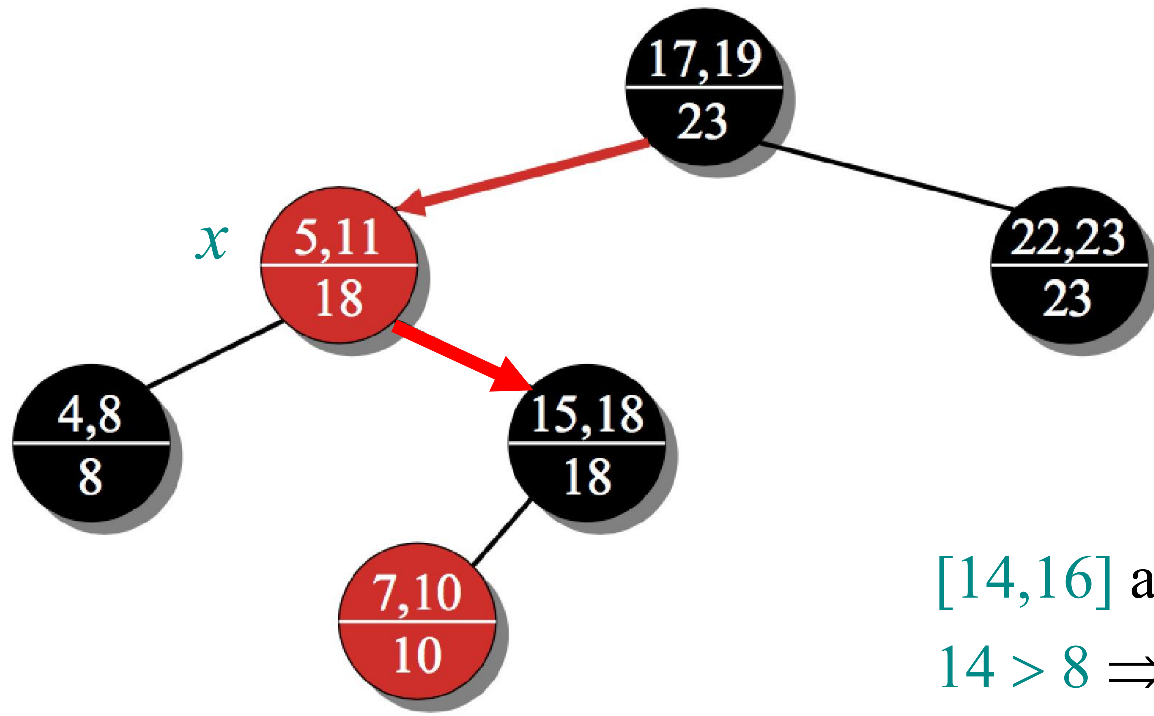


$x \leftarrow \text{T}.root$

[14,16] and [17,19] don't overlap

$14 \le 18 \Rightarrow x \leftarrow left[x]$

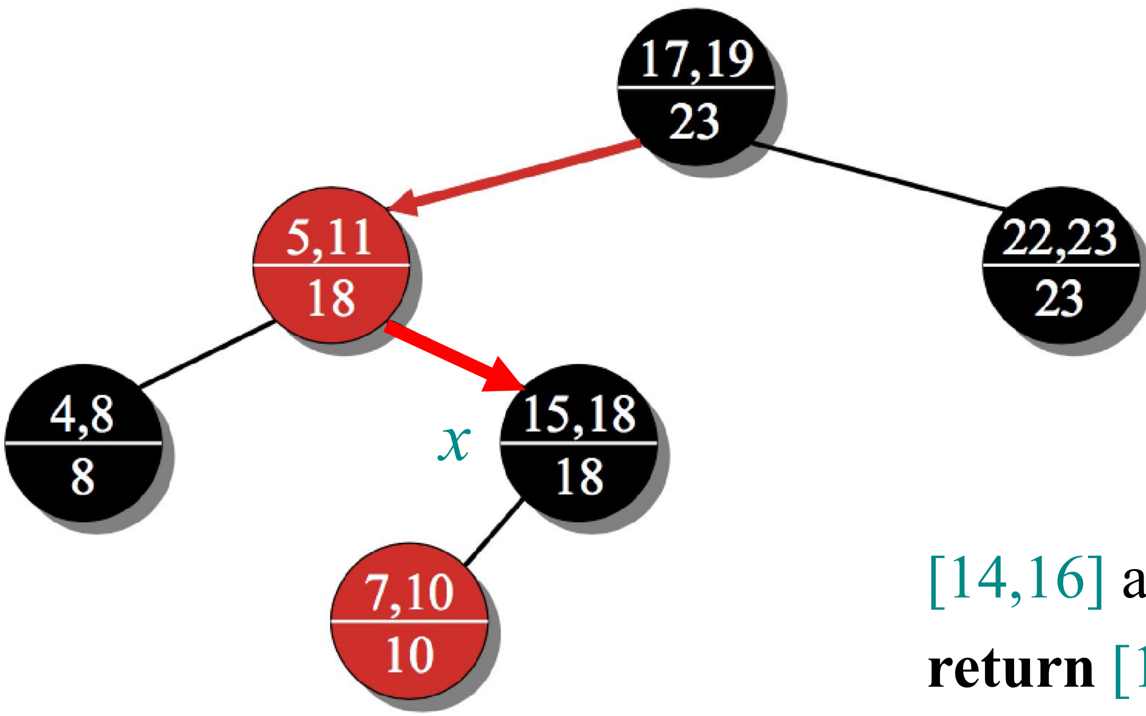
Example 1: INTERVAL-SEARCH(T, [14,16])



[14,16] and [5,11] don't overlap

$14 > 8 \Rightarrow x \leftarrow right[x]$

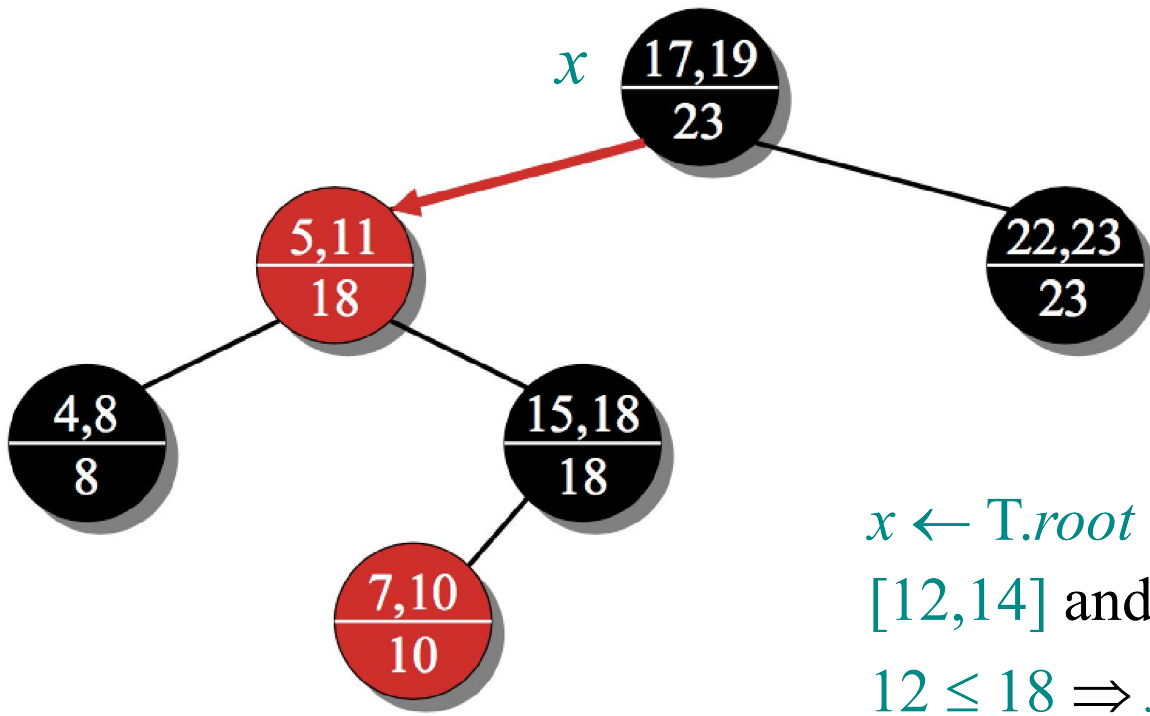
Example 1: INTERVAL-SEARCH(T, [14,16])



[14,16] and [15,18] overlap

return [15,18]

Example 2: INTERVAL-SEARCH(T, [12,14])

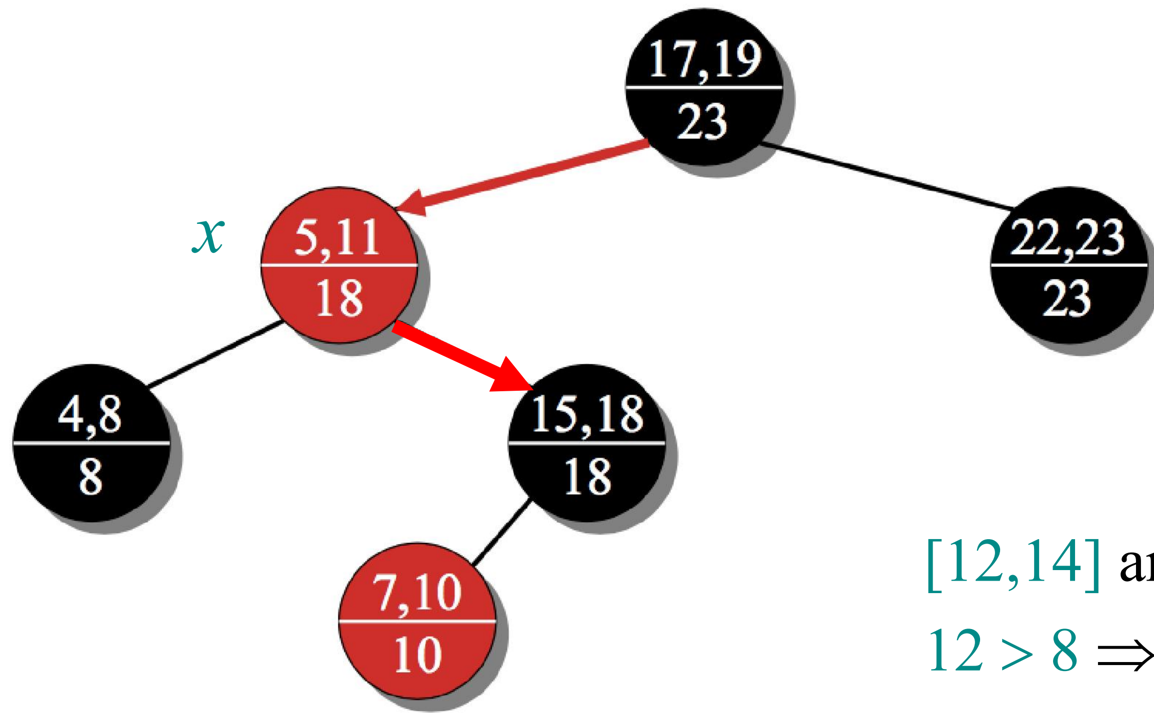


$x \leftarrow$ T.*root*

[12,14] and [17,19] don't overlap

$12 \leq 18 \Rightarrow x \leftarrow left[x]$

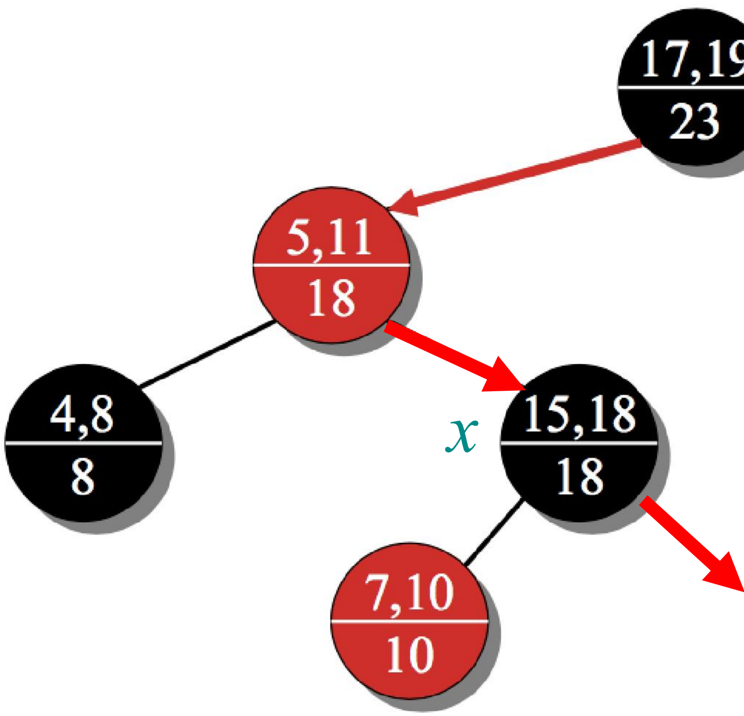
Example 2: INTERVAL-SEARCH(T, [12,14])



[12,14] and [5,11] don't overlap

$12 > 8 \Rightarrow x \leftarrow right[x]$

Example 2: INTERVAL-SEARCH(T, [12,14])



[12,14] and [15,18] don't overlap

$12 > 10 \Rightarrow x \leftarrow right[x]$

Analysis

Time = $O(h) = O(\lg n)$, since INTERVAL-SEARCH does constant work at each level as it follows a simple path down the tree.

List *all* overlapping intervals:

- Search, list, delete, repeat.
- Insert them all again at the end.

Time = $O(k \lg n)$, where k is the total number of overlapping intervals.

Correctness

Theorem. Let L be the set of intervals in the left subtree of node x, and let R be the set of intervals in x's right subtree.

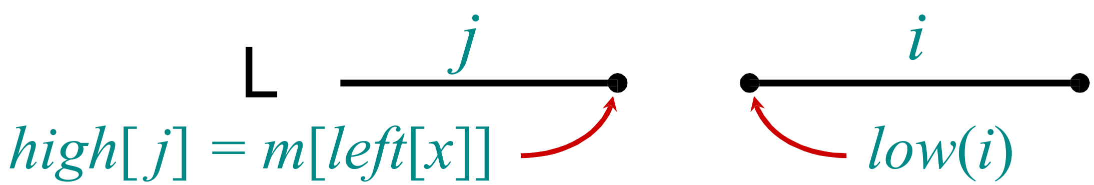
- If the search goes right, then $\{\, i' \in L : i' \text{ overlaps } i \,\} = \varnothing$.
- If the search goes left, then $\{i' \in L : i' \text{ overlaps } i \,\} = \varnothing \rightarrow \{i' \in R : i' \text{ overlaps } i \,\} = \varnothing$

In other words, it's always safe to take only 1 of the 2 children: we'll either find something, or nothing was to be found.

Correctness proof

Proof. Suppose first that the search goes right.

- If $left[x]$ = T.nil, then we're done, since $L = \emptyset$.
- Otherwise, the code dictates that we must have $low[i] > m[left[x]]$. The value $m[left[x]]$ corresponds to the high endpoint of some interval $j \in L$, and no other interval in L can have a larger high endpoint than $high[j]$.



- Therefore, $\{i' \in L : i' \text{ overlaps } i\} = \emptyset$.

Proof (cont.)

Suppose that the search goes left, and assume that

$$\{i' \in L : i' \text{ overlaps } i\} = \varnothing.$$

- Then, the code dictates that $low[i] \le m[left[x]] = high[j]$ for some $j \in L$.
- Since $j \in L$, it does not overlap i, and hence $high[i] < low[j]$.
- But, the binary-search-tree property implies that for all $i' \in R$, we have $low[j] \le low[i']$.
- But then $\{i' \in R : i' \text{ overlaps } i\} = \varnothing$.