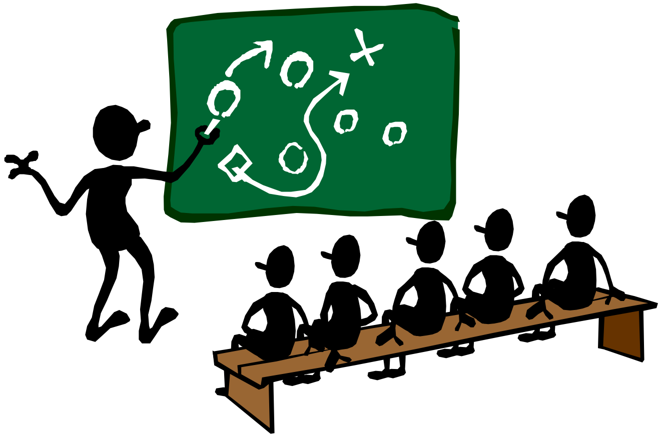
Algorithms – Chapter 14
Augmenting Data Structures



Juinn-Dar Huang

Professor

jdhuang@mail.nctu.edu.tw

October 2008

Rev. ’11, ’12, ’15, ’16, ’18, ’19, ’20

Augmenting Existing Data Structures

- In general, you just need “textbook” data structures
- You rarely have to create an entirely new type of data structure
  - if you do, let me know ☺
- Most often, you augment a data structure to meet the requirement of desired application

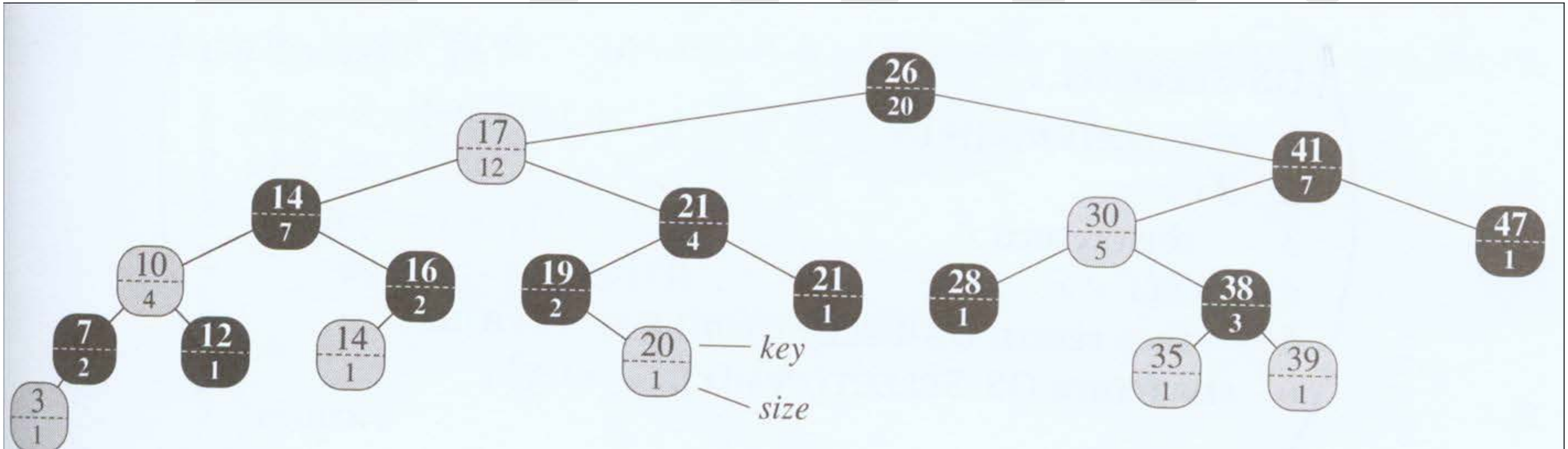
Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Augmenting RB Trees, Part 1

- Augmenting an RB tree such that
  - you can quickly find the ith smallest element, O(lgn)
  - you can quickly find the rank of a given element, O(lgn)

Order-Statistic Trees

- An order-statistic tree
  - is an RB tree
  - additional field size[x] for each node x
  - size[x] = # of internal nodes in the subtree rooted at x (including x) ➔ size[x] = size[left[x]] + size[right[x]] + 1



size[nil[T]] = 0

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Juinn-Dar Huang
jdhuang@mail.nctu.edu.tw

Retrieving an Element

OS-SELECT(*x*, *i*)

Time Complexity : O(lgn)

```
r ← size[left[x]] + 1
if i = r
    then return x
elseif i < r
    then return OS-SELECT(left[x], i)
else return OS-SELECT(right[x], i – r)
```

- Initial call : OS-SELECT(root[T], i)

Determining the Rank

OS-RANK(T, x)

$r \leftarrow size[left[x]] + 1$

$y \leftarrow x$

while $y \neq root[T]$

    do if $y = right[p[y]]$

        then $r \leftarrow r + size[left[p[y]]] + 1$

      $y \leftarrow p[y]$

return r

Time Complexity : O(lgn)

Juinn-Dar Huang
jdhuang@mail.nctu.edu.tw

Maintaining Subtree Sizes (1/2)

- Given the correct size, OS-SELECT and OS-RANK can work quickly and properly

- Need to maintain subtree sizes during node insertion and deletion operations without increasing time complexity

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Maintaining Subtree Sizes (2/2)

- Insertion
  - increment size[x] for each node x on the path from the root down to the inserted node
  - make proper modifications during rotation operations

- Deletion
  - decrement size[x] of each node x on the path from the deleted node up to the root
  - make proper modifications during rotation operations

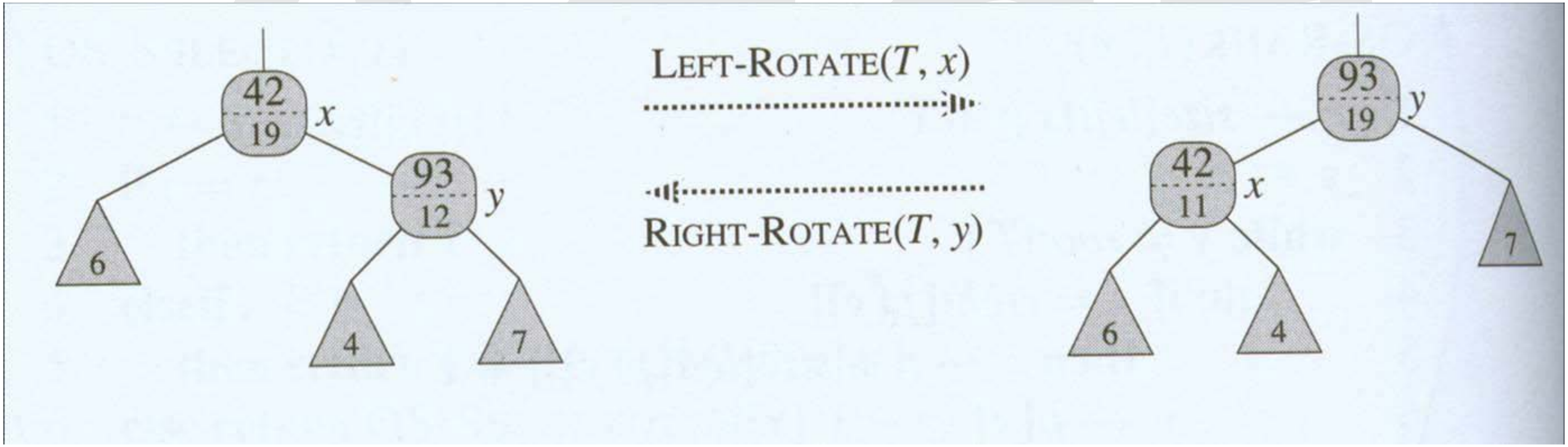
Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Updating Subtree Sizes

- Adding 2 more lines to LEFT-ROTATE(T, x) (Chap 13, p.11)

  12 $size[y] \leftarrow size[x]$

  13 $size[x] \leftarrow size[left[x]] + size[right[x]] + 1$



The change to RIGHT-ROTATE is similar (x ⇔ y)

Augmenting a Data Structure

- Four steps
  - choose an underlying data structure (RB tree in this case)
  - determine additional information to be maintained (field size for each node)
  - verify that the additional information can be maintained for the fundamental modifying operations on the underlying data structure (insertion, deletion)
  - develop new desired operations (OS-SELECT, OS-RANK)

Augmenting an RB Tree (1/2)

Theorem 1

- Let f be a field that augments a red-black tree T of n nodes
- Suppose that the contents of f for a node x can be computed using only the information in nodes x, $left[x]$, and $right[x]$, including $f[left[x]]$ and $f[right[x]]$.
- Then, we can maintain the values of f in all nodes of T during insertion and deletion without asymptotically affecting the $O(\lg n)$ performance of these operations

Juinn-Dar Huang
jdhuang@mail.nctu.edu.tw

Augmenting an RB Tree (2/2)

Proof:

- A change to f[x] propagates only to ancestors of x
- That is, updating f[x] may require f[p[x]] to be updated, but nothing else!
- The process terminates at the root and the tree height is **O(lgn)**

Closed Interval, Part 2

Self Study

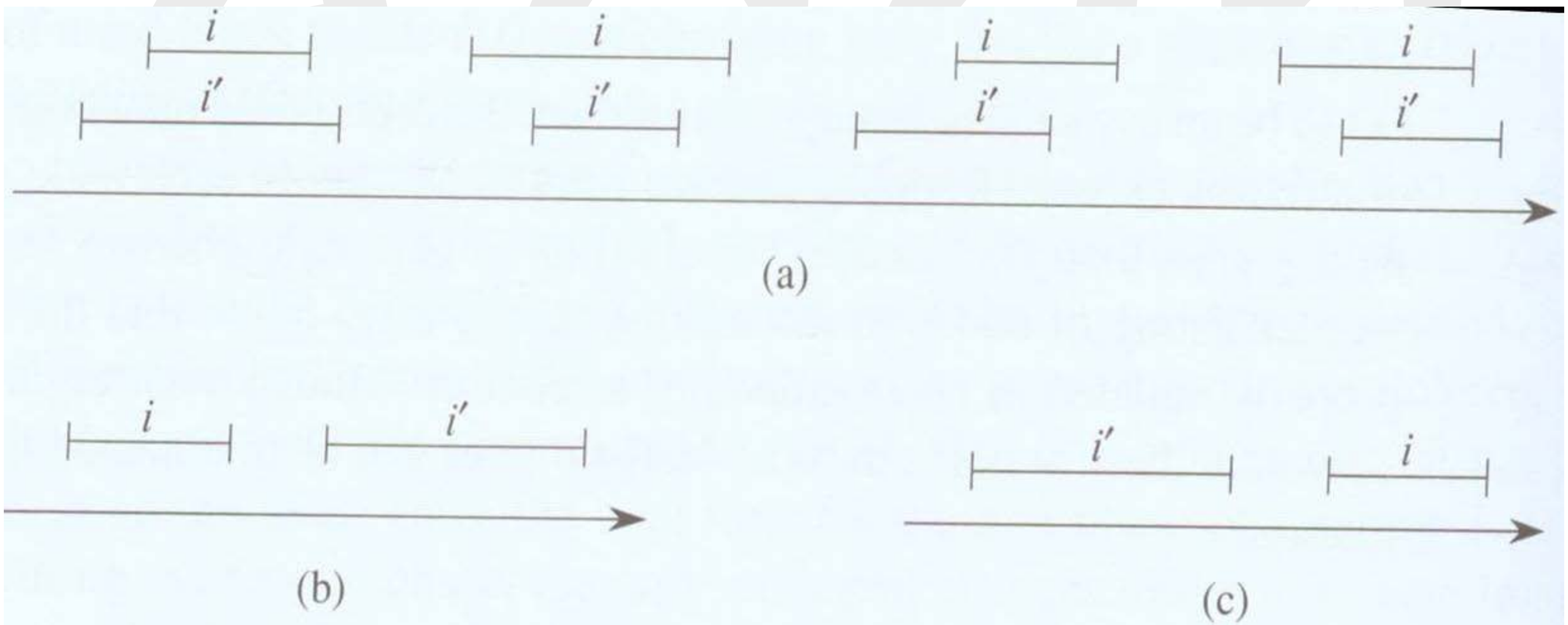
- A closed interval $[t_1,t_2]$ can be represented as an object i, with fields $low[i] = t_1$ (the **low endpoint**) and $high[i] = t_2$ (the **high endpoint**)

- The intervals i and i' **overlap** if $i \cap i' \neq \emptyset$
  - if $\mathrm{low}[i] \leq \mathrm{high}[i']$ and $low[i'] \leq high[i]$

- Any two intervals i and i' satisfy the **interval trichotomy**

Interval Trichotomy

- That is, exactly one of the following three properties holds:
  - i and i' overlap
  - i is to the left of i' (i.e., $high[i] < low[i']$)
  - i is to the right of i' (i.e., $high[i'] < low[i]$)



Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Interval Trees

- An interval tree is a red-black tree that maintains a dynamic set of elements, with each element x containing an interval int[x]
- Interval trees support
  - INTERVAL-INSERT(T, x)
  - INTERVAL-DELETE(T, x)
  - INTERVAL-SEARCH(T, i)
    - returns a pointer to an element x in T s.t. int[x] overlaps i, or the nil[T] if no such element exists

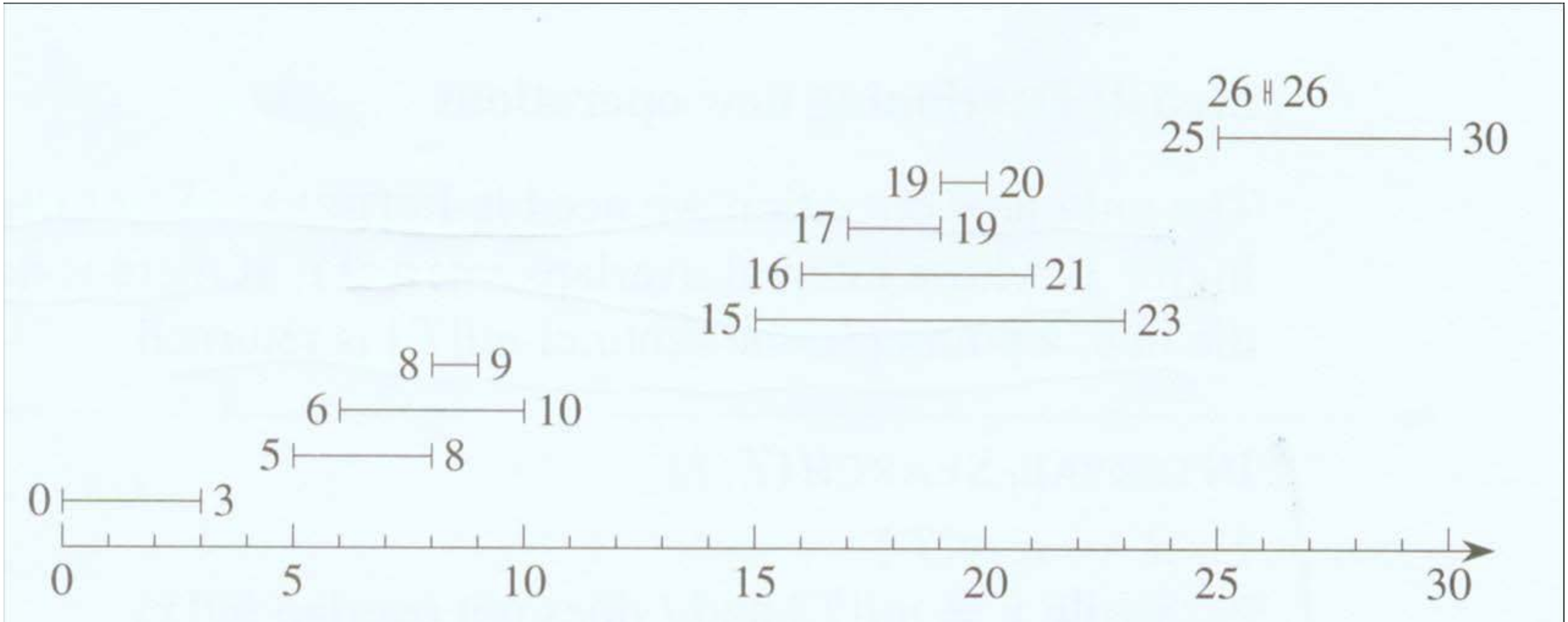
Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Augmenting Process (1/3)

- Step 1: (Underlying DS)
  - RB tree
  - main data field int[x] for each node x
    - represents an interval [low[int[x]], high[int[x]]]
  - key ➔ low[int[x]]
  - an **inorder** traversal lists all the intervals in sorted order by low endpoint
- Step 2: (Adding extra information)
  - each node x contains a value max[x], which is the maximum value of any interval (high) endpoint stored in the subtree rooted at x

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

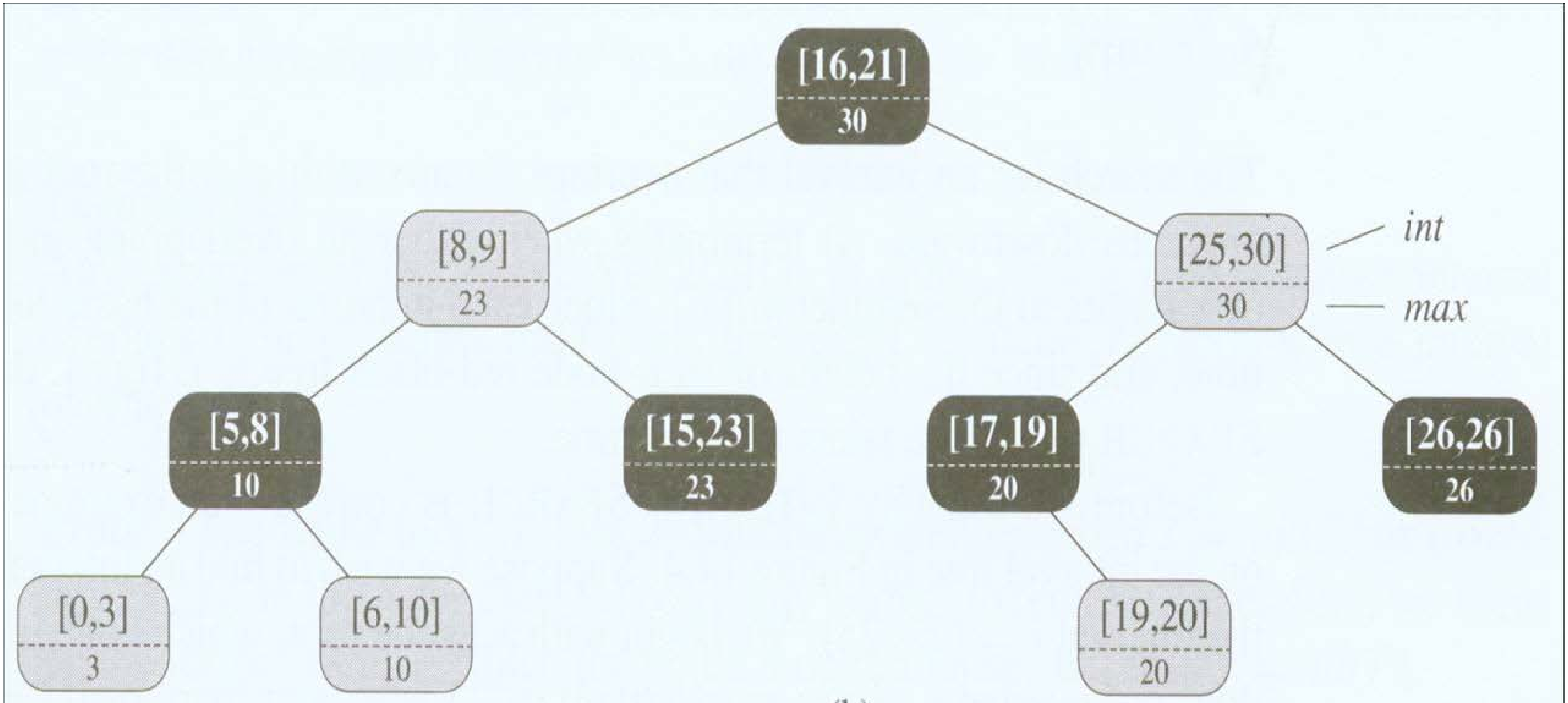
An Example Interval Tree (1/2)



(a)

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

An Example Interval Tree (2/2)



(b)

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Augmenting Process (2/3)

- Step 3: (Maintaining extra information)
  - max[x] can be updated as
    max[x] = max(high[int[x]], max[left[x]], max[right[x]])
    while performing rotation operations
  - by Theorem 1, insertion and deletion can still be finished in O(lgn) time

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Augmenting Process (3/3)

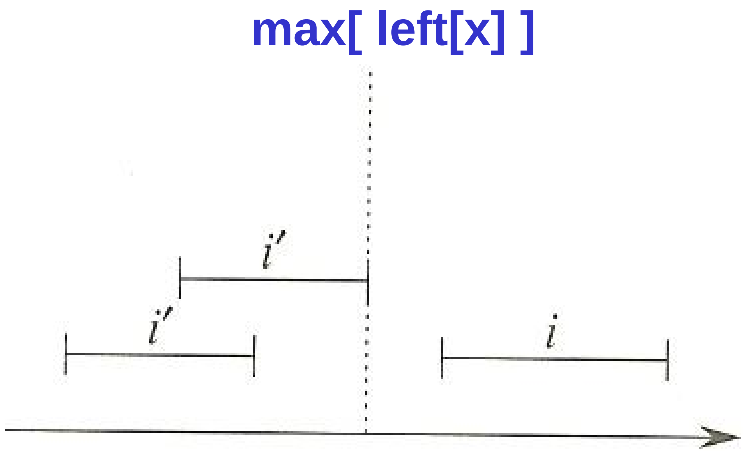
- Step 4: (Developing new operations)

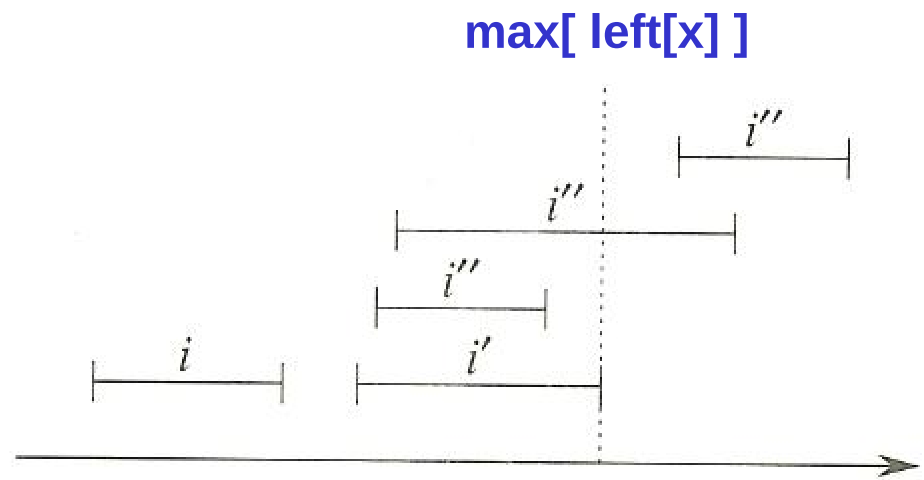
INTERVAL-SEARCH(*T*, *i*)

```
x ← root [T]
while x ≠ nil[T] and i does not overlap int[x]
        do if left[x] ≠ nil[T] and max[left[x]] ≥ low[i]
                then x ← left[x]
                else  x ← right[x]
return x
```

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Proof





Go to Line 5: (Right side)
Reason:
there is no left subtree or
max[left[x]] < low[i]

For each interval i'
in x's left subtree (if any):
high[i'] ≤ max[left[x]]
< low[i]

Go to Line 4: (Left side)
Reason: max[left[x]] ≥ low[i]

如果存在有解 → 左子樹存在有解
↔ 如果左子樹無解 → (左右子樹)皆不存在有解

∃ an interval i' in x's left subtree s.t.
high[i'] = max[left[x]] ≥ low[i] , and
if no solution in x's left subtree →
high[i] < low[i']
Then, ∀ interval i'' in x's right subtree →
low[i''] ≥ low[i'] > high[i]
→ no solution in x's right subtree