

COMP 460 Lecture Notes

R. I. Greenberg

1 Augmenting Data Structures

A nonstandard application can often be supported by augmenting a textbook data structure to maintain a bit of extra information.

1.1 Ex.: Dynamic Order Statistics

Suppose we're maintaining a set of keys through insertion and deletion, and we want to be able to do two new queries:

- Select i th elt. of set
- Find rank of elt. x

We already know how to do each of these in $\Theta(n)$ time on a set of n elts., but can we maintain info. as elts. are inserted and deleted to do these queries more efficiently than repeating $\Theta(n)$ work?

Yes. Can do in $O(\lg n)$ with an *order-statistic tree*, just a red-black tree w. inclusion in each node x of a field $size[x]$ indicating no. of (internal) nodes in subtree rooted at x (including x). ($size[NIL[T]] = 0$.)

(With equal keys, can define rank unambiguously according to position in inorder tree walk.)

- Using the size info.:

Selection is essentially like our earlier selection algs., but with the partitions already fixed. The following procedure finds the i th elt. in subtree rooted at x ; the initial call is w. $x = \text{root}$.

```
OS-SELECT( $x, i$ )
1   $r \leftarrow size[left[x]] + 1$ 
2  if  $i = r$  then return  $x$  endif
3  if  $i < r$  then return OS-SELECT( $left[x], i$ )
4  else return OS-SELECT( $right[x], i - r$ )
5  endif
```

Time: $O(\lg n)$ — walks down one path.

```

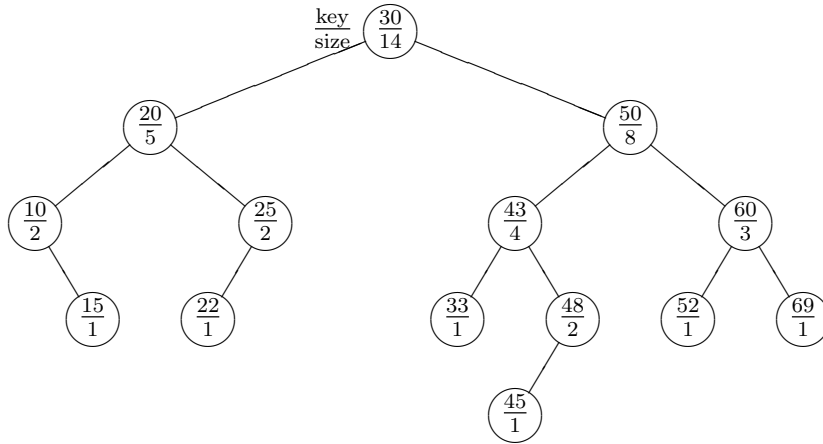
OS-RANK( $T, x$ )
1   $r \leftarrow \text{size}[\text{left}[x]] + 1$ 
2   $y \leftarrow x$ 
3  while  $y \neq \text{root}[T]$  do
4      if  $y = \text{right}[p[y]]$  then  $r \leftarrow r + \text{size}[\text{left}[p[y]]] + 1$  endif
5       $y \leftarrow p[y]$ 
6  endwhile
7  return  $r$ 

```

Time: $O(\lg n)$ — walks up one path.

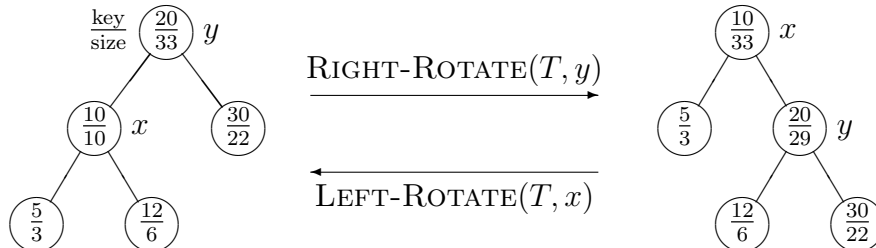
Correctness: At line 3, r is the rank of x in subtree rooted at y .

Ex. for OS-RANK: We will do an example in class based on the following tree:



- Maintaining size info.:

- Insertion: Walk down one path to find spot for insert; increment size for each node encountered. As we walk back up to patch red-black properties, we just do color changes except maybe 2 rotations. Easy to update sizes on rotation:



e.g., after $\text{RIGHT-ROTATE}(T, y)$,

$$\begin{aligned}
 \text{size}[x] &\leftarrow \text{size}[y] \\
 \text{size}[y] &\leftarrow \text{size}[\text{left}[y]] + \text{size}[\text{right}[y]] + 1
 \end{aligned}$$

(Even $O(\lg n)$ rotations would have been ok.)

- Deletion: After splice, walk up one path to root; decrement size for each node encountered. Then walk up tree doing red-black properties patch up. Book shows ≤ 3 rotations (and even $O(\lg n)$ would be ok); update sizes as above.
- Other red-black tree operations do not alter the tree.

1.2 General Steps in Augmenting a Data Structure

1. Choose underlying data structure.
2. Determine additional info. to maintain.
3. Verify additional info. can be maintained by operations modifying underlying data structure.
4. Develop new operations.

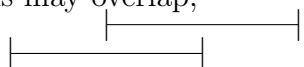
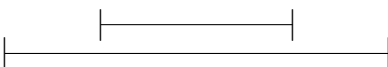

(Real design generally involves trial and error and/or working on different steps in parallel. For order-statistic tree explanation, we did step 4 before step 3. But, really, the approach was motivated by an understanding of what could be achieved in step 3.)

A natural first effort in the OS-tree might have been to maintain rank in each node. Then, OS-SELECT and OS-RANK easy. But step 3 wouldn't work; adding a new smallest elt. would require updating rank in all n nodes.

Thm.: (A Shortcut to step 3 for red-black trees.) We can maintain a field f for each node of a red-black tree without increasing the run time of operations beyond $O(\lg n)$ if $f[x]$ can be computed using only info. in nodes x , $left[x]$, and $right[x]$ (including $f[left[x]]$ and $f[right[x]]$).

1.3 Ex.: Interval Trees

A (closed) interval i represents a portion of the real line $\{r \in \mathbb{R} : low[i] \leq r \leq high[i]\}$. Intervals may overlap,

e.g.,  or 
or they may be disjoint: 

Want to support operations of:

- INTERVAL-INSERT(T, x): add object, w. a field $int[x]$ representing an interval, to set T .
- INTERVAL-DELETE(T, x): remove element x from T .
- INTERVAL-SEARCH(T, i): Find an elt. x such that $int[x]$ overlaps interval i or return NIL if none exists.

1. Underlying data structure: red-black tree w. an interval $int[x]$ in node x and $key[x] = low[int[x]]$.
2. Additional info.: $int[x]$ is really already “additional” info.; also maintain $max[x] = \max.$ of endpoints of all intervals in subtree rooted at x .
3. Maintaining new info.: $max[x] = \max(high[int[x]], max[left[x]], max[right[x]])$, so just invoke our theorem for augmenting r-b trees.
4. New operations:

```

        INTERVAL-SEARCH( $T, i$ )
1    $x \leftarrow root[T]$ 
2   while  $x \neq NIL[T]$  and  $i \cap int[x] = \emptyset$ 
3       if  $left[x] \neq NIL[T]$  and  $max[left[x]] \geq low[i]$ 
4       then  $x \leftarrow left[x]$ 
5       else  $x \leftarrow right[x]$ 
6       endif
7   endwhile
8   return  $x$ 

```

Pf. sketch for correctness: INTERVAL-SEARCH basically chases the interval j w. least $low[j]$ such that $high[j] \geq low[i]$. (Not quite so, because an overlap may be found prematurely. Exercise 14.3-3 is to modify alg. to make this so.)

(It’s easy to imagine applications for at least some more general variations, e.g., listing all overlapping intervals. This would be useful for scheduling courses with periodic cancellations or additions of courses. When adding a course at a proposed time, one would want to know who cannot teach due to already being assigned an overlapping course.)

2 Exam 1 ??

End of material for exam 1. Open book and notes. No calculators.