Augmenting Data Structures

Most complex data structures can be built upon existing ones.

Example: We can collect order statistics in _____ time using insertions and deletions.

There is an obvious _____ time algorithm for this (use sort and count).

There is also a O(n) algorithm described in Chapter 10.

Solution:

Use Augmented Red-Black Trees.

Define an _____ as a Red-Black tree with an added size(x) field that contains the number of internal nodes in the subtree rooted at node x, where

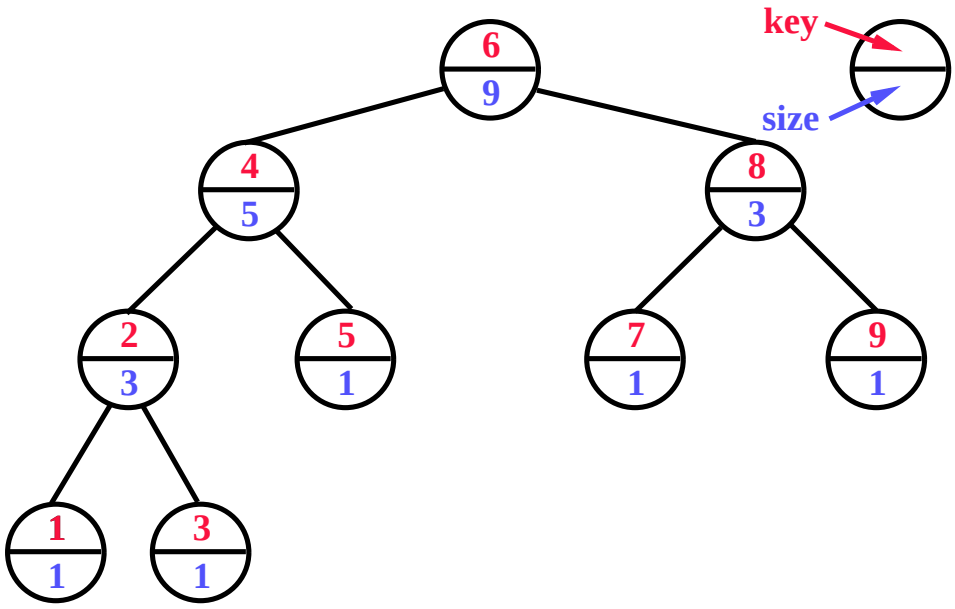
$$\text{size(x)} = \begin{cases} 0 & \text{if } x = NIL \\ 1 + size(left(x)) + size(right(x)) & \text{otherwise} \end{cases}$$

Pseudocode

```
Select(x,i)              ; return ith smallest key in subtree x
1     rank = size(left(x)) + 1
2     if i = rank
3     then return x
4     else if (i < rank)
```

```
5       then return Select(left(x), i)
6       else return Select(right(x), i - rank)
```

T(n) is approximately equal to height of tree = _____



Determine Rank of a Node

```
Rank(T,x)               ; return rank of node x
1     rank = size(left(x)) + 1
2     while x ≠ root(T)
3         if x = right(parent(x))
4         then rank = rank + size(left(parent(x))) + 1
5         x = parent(x)
6     return rank
```

T(n) is approximately equal to height of tree = _____

Maintaining Size Field

RB Tree Insertion

1. Traverse tree from root to leaf and insert the new node.
   Add one to the size of each node visited during the traversal.
   The size of the new node is 1.
   Time: _____

2. Traverse back up the tree changing colors and performing rotations (at most 2).
   Only operation affecting sizes is rotation.
   Time: _____

Rotate

```
Left-Rotate(T,x)
   size(y) = size(x)
   size(x) = size(left(x)) + size(right(x)) + 1

Right-Rotate(T,x)
   size(x) = size(y)
   size(y) = size(left(y)) + size(right(y)) + 1
```

RB Tree Deletion

1. Find and splice out node.
   Traverse up tree from deleted node, subtracting one from size of each

node visited.
Time: _____

2. Traverse up tree changing colors and performing at most 3 rotations. Handle rotations same as insertion.
   Time: _____

Augmentation in General

1. Choose underlying data structures.
   Example: RB Trees
2. Determine additional information.
   Example: size
3. Verify additional information maintained by operations.
   Example: size, O(lgn) insert/delete/select
4. Develop new operations.
   Example: select, rank

Theorem 15.1

If step 3 can be accomplished with local information (x, left(x), right(x), f(left(x)), f(right(x))) in RB Tree, then augmentation does not affect O(lg n) performance.

Interval Trees

Used for scheduling and planning.

Problem: Need data structure that determines if any of a stored set of intervals overlaps a given interval. We want to perform all associated operations in O(lg n) time.

Define a _____ object i as $[t_1, t_2]$ such that $t_1 \leq t_2$, low(i) $=t_1$, and high(i) $= t_2$.

Overlap(i_1, i_2)
1 if low(i_1) $\leq$ high(i_2) and low(i_2) $\leq$ high(i_1)
2 then return True
3 else return False

Does [2, 7] overlap [5, 8]? _____

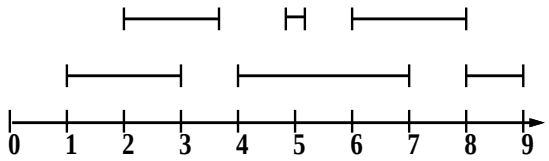
Augmentation

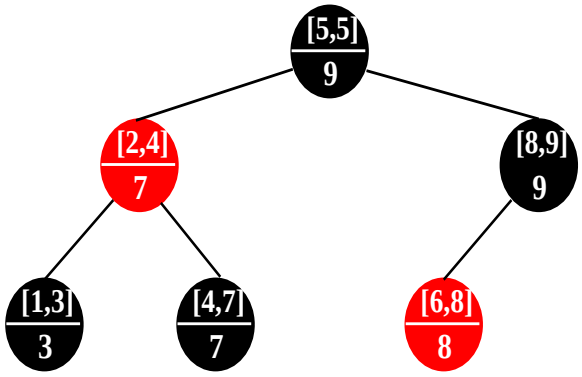
1. Use RB Tree with interval i at each node and key being low(i). Inorder traversal visits intervals sorted by their low endpoint.
2. Add to each node x, max(x), the maximum of any interval endpoint in subtree rooted at x.
3. max(x) = maximum(high(interval(x)), max(left(x)), max(right(x))). Note that we are using all local information, so by Theorem 15.1 we still have _____ performance.
4. Interval-Search(T,i) returns interval overlapping interval i, or NIL if none exists.

```
Interval-Search(T,i)
1     x = root(T)
2     while x ≠ NIL and not Overlap(i, interval(x))
3         if left(x) ≠ NIL and max(left(x)) ≥ low(i)
4         then x = left(x)
5         else x = right(x)
6     return x
```

Note here that if max(left(x)) is large enough, then either i overlaps an interval in left(x) or i < intervals in left(x).

Overlap






```
Interval-Search(T, [6,7])
   x = [5,5] / 9
   x = left(x) = [2,4]/7
   x = right(x) = [4,7]/7      OVERLAP
```