RB-Tree Augmentation

• Augment x with \texttt{Size(x)}, where
  - \texttt{Size(x)} = size of subtree rooted at x
  - \texttt{Size(NIL)} = 0
OS-Select

OS-SELECT(x, i) //page 304
// Select the node with rank i
// in the subtree rooted at x
1. \( r \leftarrow \text{size[left[x]]} + 1 \)
2. if \( i = r \) then
3. return x
4. elseif \( i < r \) then
5. return OS-SELECT (left[x], i)
6. else return OS-SELECT (right[x], \( i-r \))

Time Complexity O(log n)
OS-Rank

```plaintext
OS-RANK(x,y)
// Different from text (recursive version)
// Find the rank of y in the subtree rooted at x
1   r = size[left[y]] + 1
2   if x = y then return r
3 else if ( key[x] < key[y] ) then
4      return OS-RANK(x,left[y])
5 else return r + OS-RANK(x,right[y])
```

Time Complexity O(log n)
Augmenting RB-Trees

Theorem 14.1, page 309

Let $f$ be a field that augments a red-black tree $T$ with $n$ nodes, and $f(x)$ can be computed using only the information in nodes $x$, $\text{left}[x]$, and $\text{right}[x]$, including $f[\text{left}[x]]$ and $f[\text{right}[x]]$.

Then, we can maintain $f(x)$ during insertion and deletion without asymptotically affecting the $O(lgn)$ performance of these operations.

For example,

$$\text{size}[x] = \text{size}[\text{left}[x]] + \text{size}[\text{right}[x]] + 1$$

$$\text{rank}[x] = ?$$
Examples of augmenting information for RB-Trees

- Parent
- Height
- Any associative function on all previous values or all succeeding values.
- Next
- Previous
Interval Trees

- **Need**: Dynamic data structure to store time intervals
- **Application**: Maintain schedule for set of seminars
- **Operations**: Insert, Delete
- Every interval j has: low[j], high[j]
- **Data Structure**:
  - Augment RB-Tree so that it can store intervals.
  - Ordering based on what key? low values? high values? (high+low)/2 values? (high-low) values?
  - Note that insert and delete are still efficient.
- **New Operation**: Search (find any overlapping interval)
  - Problem with Search!
Augmented Information

- **low, high, max**
- \( \text{max}[x] = \) rightmost high value of all intervals in subtree rooted at \( x \)
- The value \( \text{max}[x] \) of each node can be written as:
  \[
  \text{max}[x] = \text{Max}\{ \text{high}[\text{int}[x]], \text{max}[\text{left}[x]], \text{max}[\text{right}[x]] \}
  \]
- Therefore it can be maintained efficiently under insertions and deletions
**Interval-Search**

INTERVAL-SEARCH (T, j)
// finds an interval in tree T that overlaps interval j, // else return NIL.
1. \( x = \text{root}[T] \)
2. while \( x \neq \text{NIL} \) and j does not overlap \( \text{int}[x] \) do
3. if \( \text{left}[x] \neq \text{NIL} \) and \( \max[\text{left}[x]] \geq \text{low}[j] \) then
   4. \( x = \text{left}[x] \)
5. else \( x = \text{right}[x] \)
6. return \( x \)

Time Complexity \( O(\log n) \)