Augmenting Red-black trees

Suppose we want to add the following two operations

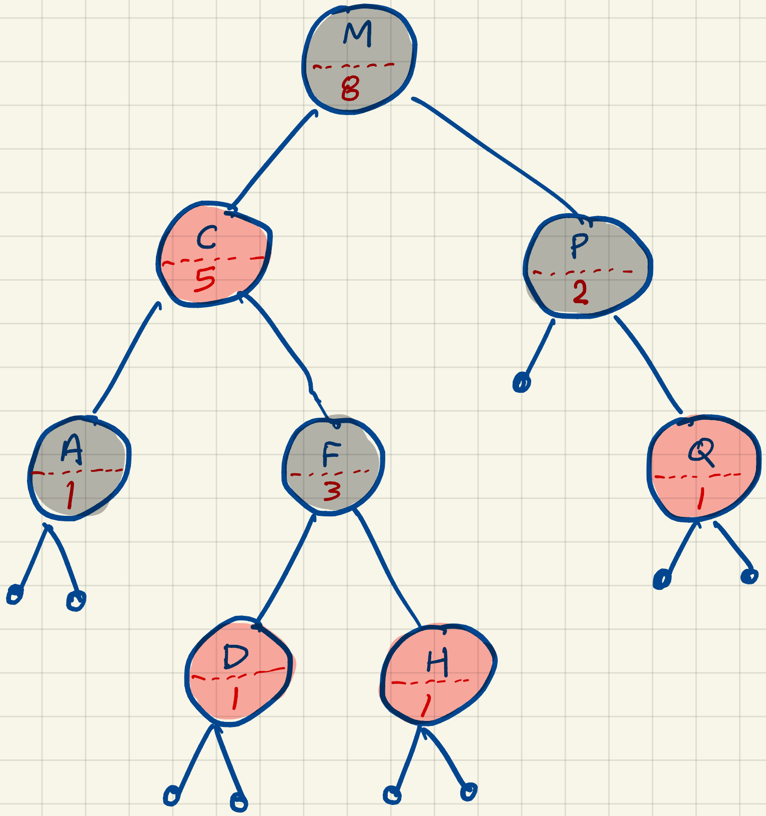
Select (x, i) : returns node with ith smallest key in x's subtree

rank (T, x) : returns rank of x in the inorder traversal of T.

IDEA: Augment each node x with $size[x]$

$size[x] = \#$ internal nodes in x's subtree (including x)

$size[NIL[T]] = 0$



ACDFH

Note:

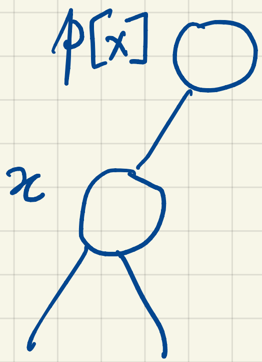
$$size[x] = size[left[x]] + size[right[x]] + 1$$

$size[left[x]] + 1$ = rank of x in it's subtree

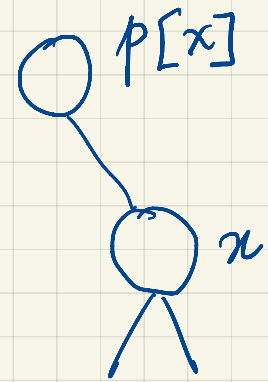
```
Select(x, i)
    r <- size[left[x]] + 1
    if i = r
        then return x
        else if i < r
            then return Select(left[x], i)
            else return Select(right[x], i-r)
```

```
Rank(T, x)
    r ← size[left[x]] + 1    ▷ rank of x in it's subtree
    y ← x
    while y ≠ root[T]
        do if y = right[p[y]]    ▷ y is a right child
              then r ← r + size[left[p[y]]] + 1
           y ← p[y]
    return r
```

$r = size[left[x]] + 1$

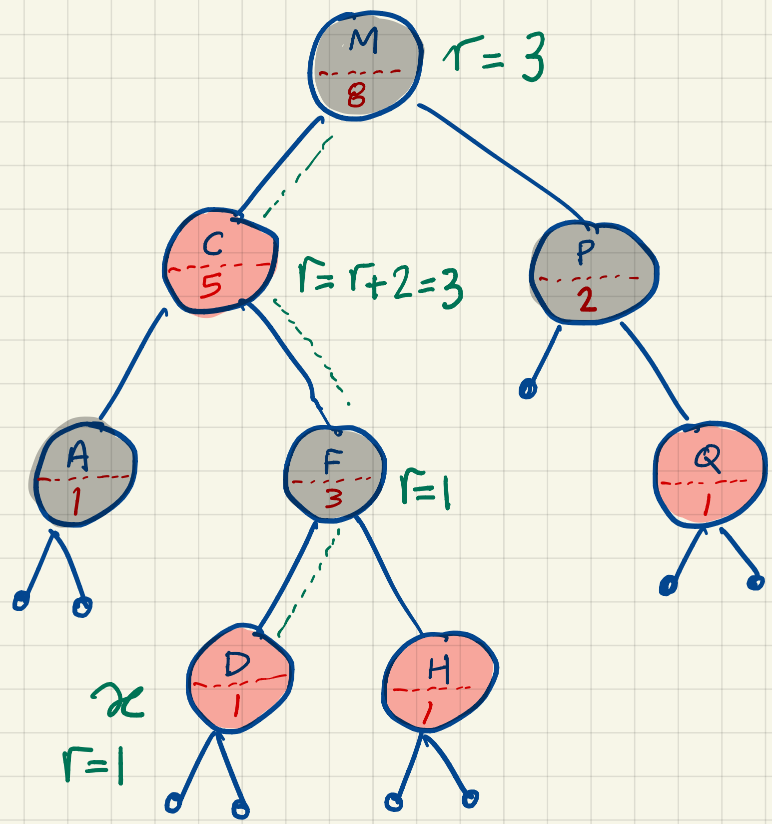


r = rank of x in x's subtree
is also rank of x in $p[x]$'s subtree



rank of x in y's subtree
$= r +$ rank of $p[x]$

Example:



A C D F H M P Q

$\uparrow$

3

Augmenting Red-black trees in general

Let f be a field that augments a Red-black tree.

Suppose $f[x]$ can be computed using only the information in

- x itself
- $left[x]$
- $right[x]$

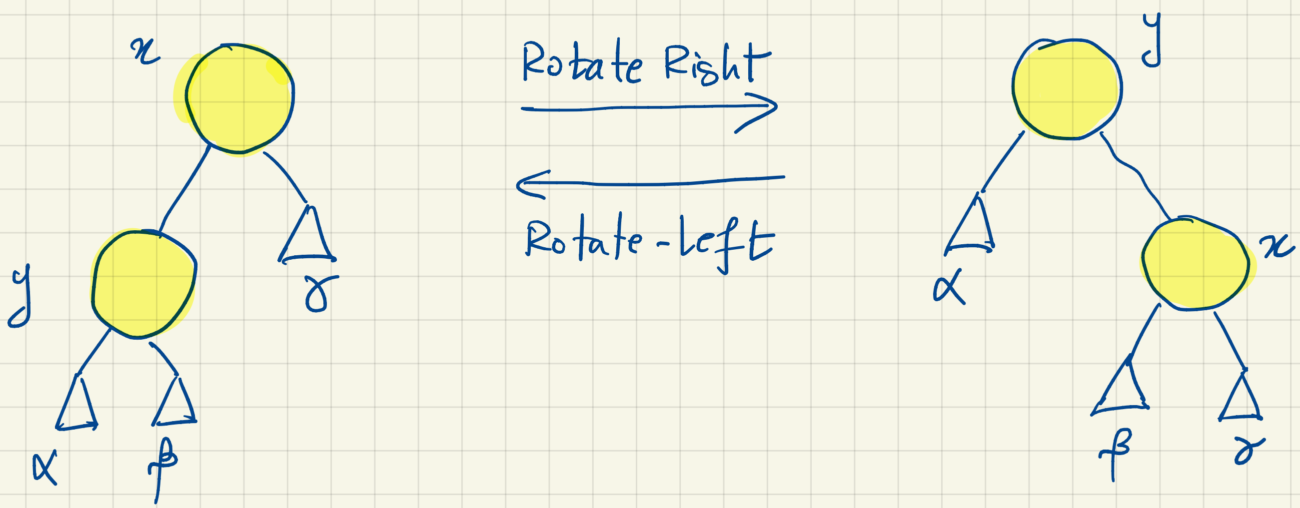
  including $f[left[x]]$ and $f[right[x]]$

Then one can maintain f in all nodes during insertion & deletion without affecting the $O(\log n)$ asymptotic performance.

Proof:

- Changing $f[x]$ only affects the field in nodes on the path from $root[T]$ to x ($O(\log n)$ computations)

- Rotations require local changes only.

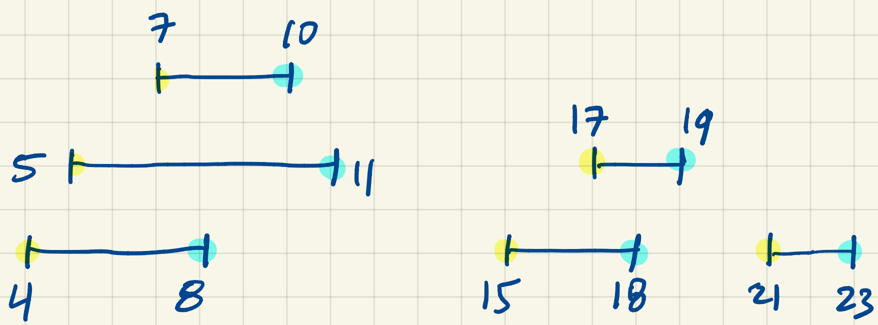


Only two nodes require changes in a rotation, and we have a constant number of rotations per operation.

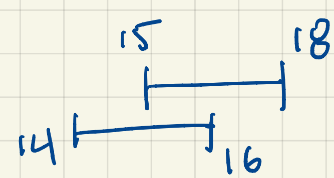
Application : Interval trees

- Maintain a dynamic set of intervals
- Support operation : Find an interval in the set that overlaps a given query interval.

Example:



Query:



[14, 16] $\longrightarrow$ [15, 18]

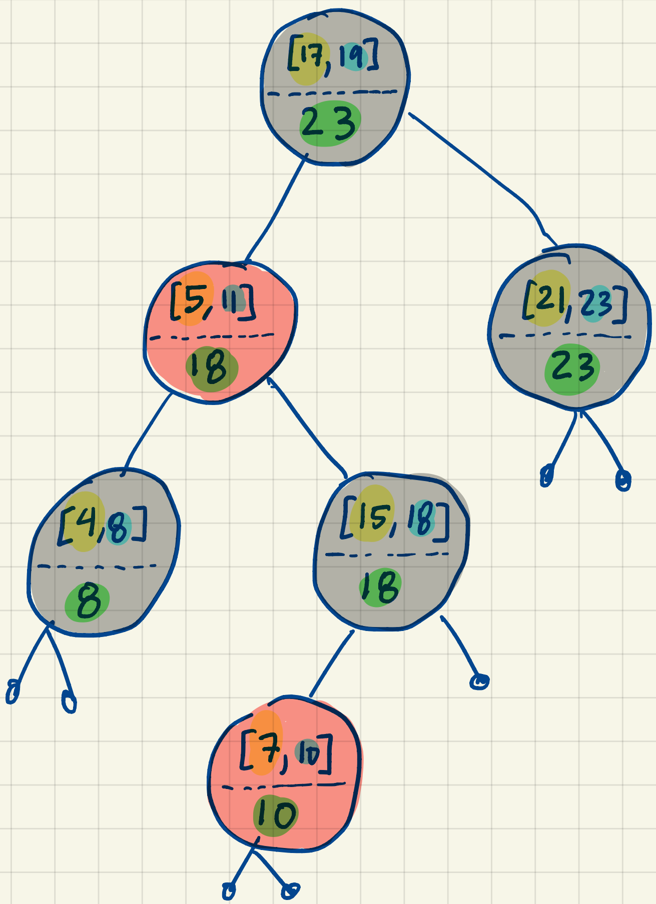
[16, 19] $\longrightarrow$ [15, 18] or [17, 19]

[12, 14] $\longrightarrow$ NIL[T]

- Red-black tree with $key[x] = low[int[x]]$

  So the binary search tree is sorted on the low end points of intervals

- Augment with $max[x] = max \begin{cases} high[int[x]] \\ max[left[x]] \\ max[right[x]] \end{cases}$, $max[Nil[T]] = 0$



BST on

is maximum of all in subtree

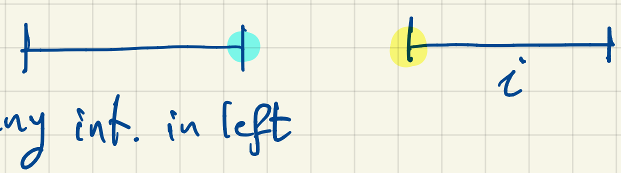
```
Interval-Search (T, i)      ▷ i is intreval
    x ← root[T]
    while x ≠ NIL[T] and i ∩ int[x] = ∅   (no overlap)
        do if left[x] ≠ NIL[T] and max[left[x]] ≥ low[i]
              then x ← left[x]
              else x ← right[x]
    return x
```

Prove every move is safe

Case 1: Go right.

$$\text{left}[x] = \text{NIL}[T] \quad \text{or}$$

$$\max[\text{left}[x]] < \text{low}[i]$$



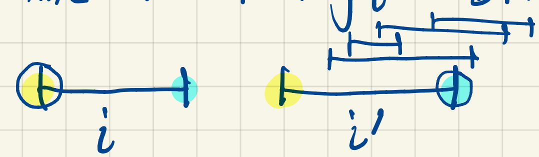
So there is no overlap in left.

Case 2: Go left.

If no overlap in left subtree

$$\text{low}[i] \leq \max[\text{left}[x]] = \text{high}[i']$$

for some i' in left subtree



But BST $\Rightarrow$ $\text{high}[i] <$ any low in right subtree

So no overlap in right either.