AUGMENTING DATA STRUCTURES (BSTs)

We saw a clever way to find the $i$-th smallest element in a set in $\Theta(n)$-time. With $O(n \log n)$ pre-processing we can do this (dynamically) in $O(\log n)$ time.

We can also ask for the rank of an element (dynamically) in $O(\log n)$ time.

example:

```
    M
   / \  
  C   P
 /   /  \
A   N   Q
```

RB-tree contains sorted letters

$\text{rank}(M) = 6$

$\text{rank}(H) = 5$
We saw a clever way to find the $i$-th smallest element in a set in $\Theta(n)$ time. With $O(n \log n)$ pre-processing we can do this (dynamically) in $O(\log n)$ time.

We can also ask for the rank of an element (dynamically) in $O(\log n)$ time.

Example:

```
  M
 /\  \
C   P
  /\    /
A   D  F
  /\    /
H   N  Q
```

RB-tree contains sorted letters \[\rightarrow\] augmented with subtree sizes.
Select\((x, i)\) \hspace{3mm} \text{// get } i\text{-th element in subtree rooted at } x.\\k \leftarrow 1 + \text{size}(l_x) \hspace{3mm} \text{// } l_x: \text{left child of } x\\\text{if } i = k, \text{ return } x.\\\text{else if } i < k, \text{ return } \text{Select}(l_x, i)\\\text{else (}i > k\text{) return } \text{Select}(r_x, i-k)\\\\\\example: \hspace{3mm} i = 5\\k = 6\\i = 5, \hspace{3mm} k = 2\\i = 3, \hspace{3mm} k = 2\\i = 1, \hspace{3mm} k = 1

Select(root, 5)\\k \leftarrow 1 + 5\\i < k \Rightarrow \text{Select}(c, 5)\\k = 1+1\\i > k \Rightarrow \text{Select}(f, 3)\\k = 1+1\\i > k \Rightarrow \text{Select}(h, 1)\\k = 1+0\\i = k \Rightarrow \text{return } h
The balanced BST can be built in $\Theta(n \log n)$ time.

Compute subtree sizes as you build or postorder walk after building.
The balanced BST can be built in $\Theta(n \log n)$ time.

Get rank: just as easy.
Walk up from node, adding sizes of subtrees representing smaller $k$'s.

ex: $\text{rank}(H) \rightarrow$ size($l_H$) = 0, walk up to $F$.
$H$ is right child of $F$, so count $F$.
size($l_F$) = 1 \quad \text{sum} = 1 + 1
walk up to $C$, count it.
size($l_C$) = 1 \quad \text{increment sum by 1}
wake up to $M$, don't count it.

$\text{TOTAL} = 5 \quad (4 + 1 \text{ for } H)$
What if we stored ranks instead of tree sizes?

$\Rightarrow$ retrieve rank $= \Theta(1)$ time
update $= O(n)$ ...no good.
Can we update subtree sizes when inserting/deleting data?

Use a RB tree

when are subtree sizes affected? Rotations
The diagram illustrates a transformation between two tree structures. The left side shows a tree with nodes A and B, where A has children X and Y, and B has children Z. The right side shows a transformed tree with nodes A and B, where A has children X and Z, and B has children Y. The transformation is labeled as "sizes." Below the trees, there are representations of the sizes of the trees. The left tree has a size of 1+X+Y, and the right tree has a size of 2+X+Y+Z. The transformation preserves the size of the trees, as shown by the equivalence symbols between the sizes.
So we can search in $O(\log n)$ time, but what about insertion?

Easy to update tree sizes as we insert new node.
But we have to rebalance.

In a RB tree, coloring doesn’t affect tree sizes.

Rotations matter.

TOTAL: $O(\log n)$