AUGMENTING DATA STRUCTURES (BSTs)
Finding the rank of an element in a set

Use array:

\[
\begin{array}{cccccccc}
P & F & C & H & Q & A & N & D & M \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9
\end{array}
\]

\[\text{rank}(F) = ?\]
**Finding the rank of an element in a set**

Use array:

```
1 2 3 4 5 6 7 8 9
```

![Partition diagram]

\[ \text{rank}(F) = ? \]

\[ \Theta(?) \]
Finding the rank of an element in a set

Use array:

\[
\begin{array}{cccccccccc}
P & F & C & H & Q & A & N & D & M \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
C & A & D & F & P & H & Q & N & M \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

\[\text{rank}(F) = ?\]

\[\Theta(n)\]

OK if done once. Not for multiple queries

suggestions?
Finding the rank of an element in a set

Use array:

Preprocess (sort)

\[ \text{A C D F H M N P Q} \]

\[ \text{ACDFHMNPQ} \]

\[ \text{1 2 3 4 5 6 7 8 9} \]

\[ \text{1 2 3 4 5 6 7 8 9} \]

\[ \text{PFCHQANDM} \]

\[ \text{CADFPHQNM} \]

\[ \text{1 2 3 4 5 6 7 8 9} \]

\[ \text{1 2 3 4 5 6 7 8 9} \]

\[ \text{rank(F) = ?} \]

\[ \Theta(n) \]

OK if done once.
Not for multiple queries

\[ O(n \log n) \]

Now all queries: \( O(1) \)
Finding the rank of an element in a set

Use array:

\[
\begin{array}{cccccccccc}
P & F & C & H & Q & A & N & D & M \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
C & A & D & F & P & H & Q & N & M \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
A & C & D & F & H & M & N & P & Q \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

\[\text{rank}(F) = 6\]

\[\begin{cases}
\Theta(n) \\
\text{OK if done once.}
\end{cases}\]

\[\text{Not for multiple queries}\]

Preprocess (sort)

\[O(n \log n)\]

Now all queries: \(O(1)\)

What if we want to insert/delete?
Finding the rank of an element in a set

Use array:

\[
\begin{array}{cccccccccc}
P & F & C & H & Q & A & N & D & M \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
C & A & D & F & P & H & Q & N & M \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

\[\text{rank}(F) = ?\]

\[\Theta(n)\]
\{ OK if done once. Not for multiple queries \}

Preprocess (sort):

\[
\begin{array}{cccccccccc}
A & C & D & F & H & M & N & P & Q \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

\[O(n \log n)\]

Now all queries: \(O(1)\)

What if we want to insert/delete? \(\rightarrow\) bad \(O(n)\)
Finding the rank of an element in a dynamic set with preprocessing

not dynamic ← sorting an array
Finding the rank of an element in a dynamic set with preprocessing

not dynamic ← sorting an array

RB-tree contains sorted letters
Finding the rank of an element in a dynamic set with preprocessing sit.

Not dynamic $\Leftarrow$ sorting an array

RB-tree contains sorted letters

Now we can quickly restore sorted order

(still need to get ranks)

Dynamic $\checkmark$
Finding the rank of an element in a DYNAMIC SET with PREPROCESSING.

Not dynamic $\leftrightarrow$ sorting an array.

RB-tree contains sorted letters.
Now we can quickly restore sorted order.
Store ranks.

Dynamic $\checkmark$
Finding the rank of an element in a dynamic set with preprocessing

Not dynamic $\leftrightarrow$ sorting an array

RB-tree contains sorted letters

Now we can quickly restore sorted order

Store ranks... $\rightarrow$ bad

(too many ranks change w/ insert)

Dynamic $X$
Finding the rank of an element in a dynamic set with preprocessing is not dynamic sorting an array.

RB-tree contains sorted letters.

Now we can quickly restore sorted order:
- Store ranks... (bad)
  - too many ranks change with insert
- Store subtree sizes

Dynamic?
Using the augmented tree to find ranks

**Rank(H)**

Walk up from node, adding sizes of subtrees representing smaller #'s
Using the augmented tree to find ranks

Rank(H)
Walk up from node, adding sizes of subtrees representing smaller #’s

\[ \text{size}(l_H) = 0, \text{ walk up to F} \]
Using the augmented tree to find ranks

**Rank(H)**

- Walk up from node, adding sizes of subtrees representing smaller #’s

\[
\text{size}(l_H) = 0, \text{ walk up to F} \\
\text{H is right child of F, so count F.}
\]
USING THE AUGMENTED TREE TO FIND RANKS

**Rank(H)**

Walk up from node, adding sizes of subtrees representing smaller #’s

\[
\text{size}(l_H) = 0, \text{ walk up to } F \\
H \text{ is right child of } F, \text{ so count } F. \\
\text{size}(l_F) = 1 \quad \Rightarrow \text{ sum } = 1 + 1
\]
Using the augmented tree to find ranks

**Rank(H)**

Walk up from node, adding sizes of subtrees representing smaller $\#'$s

\[
\text{size}(H) = 0, \text{ walk up to F}
\]

H is right child of F, so count F.

\[
\text{size}(F) = 1 \quad \text{... sum } = 1 + 1
\]

walk up to C, count it.
Using the augmented tree to find ranks

**Rank(H)**

Walk up from node, adding sizes of subtrees representing smaller #’s

- size(l_H) = 0, walk up to F
- H is right child of F, so count F.
- size(l_F) = 1 \( \rightarrow \) sum = 1 + 1
- walk up to C, count it.
- size(l_C) = 1 \( \rightarrow \) increment sum by 1
Using the augmented tree to find ranks

Rank(H)

Walk up from node, adding sizes of subtrees representing smaller #'s

\[
\text{size}(l_H) = 0 \text{, walk up to } F \\
H \text{ is right child of } F \text{, so count } F. \\
\text{size}(l_F) = 1 \quad \text{sum} = 1 + 1 \quad \text{walk up to } C, \text{ count it.} \\
\text{size}(l_C) = 1 \quad \text{increment sum by 1} \quad \text{walk up to } M, \text{ don't count it.} \\
\text{TOTAL} = 5 \quad (4 + 1 \text{ for } H)
\]
USING THE AUGMENTED TREE TO FIND RANKS

Rank(H)
Walk up from node, adding sizes of subtrees representing smaller #’s

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 \quad \text{walk up to C, count it.}
size(l_{C}) = 1  \quad \text{... increment sum by 1}
walk up to M, don’t count it.

TOTAL = 5  \quad (4 + 1 \text{ for } H)
Using the augmented tree to find ranks

\[ \text{Rank}(H) \]

Walk up from node, adding sizes of subtrees representing smaller \( H \)’s

\[
\begin{align*}
\text{size}(\ell_H) &= 0, \text{ walk up to } F \\
H \text{ is right child of } F, \text{ so count } F. \\
\text{size}(\ell_F) &= 1 \quad \text{... sum } = 1 + 1 \rightarrow \\
\text{walk up to } C, \text{ count it.} \\
\text{size}(\ell_C) &= 1 \quad \text{... increment sum by } 1 \\
\text{walk up to } M, \text{ don’t count it.} \\
\text{TOTAL} &= 5 \quad (4+1 \text{ for } H)
\end{align*}
\]
Using the augmented tree to find ranks

Rank(H)

Walk up from node, adding sizes of subtrees representing smaller #’s

size(l_H) = 0, walk up to F
H is right child of F, so count F.
size(l_F) = 1, ... sum = 1 + 1
walk up to C, count it.
size(l_C) = 1, ... increment sum by 1
walk up to M, don’t count it.

Total = 5 (4 + 1 for H)
The balanced BST can be built in $\Theta(n \log n)$ time
The balanced BST can be built in $\Theta(n \log n)$ time.

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

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

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

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

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

Compute subtree sizes after building by postorder walk...

... or update path $\mathcal{P}$ when inserting $\mathcal{P}$
The balanced BST can be built in $\Theta(n \log n)$ time.

Compute subtree sizes after building by postorder walk...

... or update path } when inserting }

BUT...
The balanced BST can be built in $\Theta(n \log n)$ time.

Compute subtree sizes after building by postorder walk...

... or update path $\ell$ when inserting $\ell$.

BUT...

we will need to rebalance.
Can we update subtree sizes when inserting/deleting data?
Can we update subtree sizes when inserting/deleting data?

Use a RB tree

When are subtree sizes affected?
Can we update subtree sizes when inserting/deleting data?

Use a RB tree

↗ when are subtree sizes affected? Rotations
AUGMENTED TREE TO FIND RANKS

- easy to find rank:
  - look at ancestor path & some adjacent subtree sizes

- subtree sizes can be updated when inserting and rebalancing

$O(\log n)$ per search/insertion/deletion
DYNAMIC SELECTION
find the $i$-th smallest element in a set

Static: $\Theta(n)$

Dynamic: $O(n\log n)$ preprocessing $\rightarrow$ balanced BST w/ subtree sizes
Select(x, i) \text{ \texttt{// get } i\text{-th element in subtree rooted at } x.}
\begin{align*}
k &\leftarrow 1 + \text{size}(l_x) \quad \text{\texttt{// } l_x: \text{left child of } x} \\
\text{if } i = k, \text{ return } x.
\end{align*}
Select \( (x, i) \)  \hspace{1cm} \text{get } i\text{-th element in subtree rooted at } x.

\[ k \leftarrow 1 \text{+ } \text{size}(l_x) \]  \hspace{1cm} \text{ } \text{ } \text{l}_x: \text{left child of } x

\text{if } i = k, \text{ return } x.

\text{example: } i = 5

\[ k = 6 \]

\[ k \leftarrow 1 \text{+ } 5 \]

\[ i < k \]

Now what?
Select($x, i$)  \hspace{1em} \text{// get $i$-th element in subtree rooted at $x$} \\
\quad k \leftarrow 1 + \text{size}(l_x)  \hspace{1em} \text{// $l_x$: left child of $x$} \\
\quad \text{if } i = k, \text{ return } x. \\
\quad \text{else if } i < k, \text{ return } \text{Select}(l_x, i)$

element: $i = 5$  \\
\quad \text{Select}(\text{root}, 5)  \\
\quad \quad \quad \quad \quad k = 6  \\
\quad \quad \quad \quad \quad k \leftarrow 1 + 5  \\
\quad \quad \quad \quad \quad i < k
Select\( (x, i) \) \hspace{1em} \text{\textbackslash get i-th element in subtree rooted at } x.

\[
\begin{align*}
  k &\leftarrow 1 + \text{size}(l_x) \hspace{1em} \text{\textbackslash } l_x: \text{left child of } x \\
  \text{if } i = k, \text{ return } x. \\
  \text{else if } i < k, \text{ return } \text{Select}(l_x, i)
\end{align*}
\]

element: \( i = 5 \)

\[
\begin{align*}
  k &= 6 \\
  k &\leftarrow 1 + 5 \\
  i < k &\Rightarrow \text{Select}(c, 5)
\end{align*}
\]
Select(x, i) \[ \text{get } i\text{-th element in subtree rooted at } x \]

\[
k \leftarrow 1 + \text{size}(lx) \quad \text{\(lx\): left child of } x\]

if \(i = k\), return \(x\).

else if \(i < k\), return \(\text{Select}(lx, i)\)

**Example:** \(i = 5\)

\[
k = 6
\]

\[
i = 5, k = 2
\]

Select(root, 5)

\[
k \leftarrow 1 + 5
\]

\[
i < k \Rightarrow \text{Select}(c, 5)
\]

\[
k = 1 + 1
\]

\[
i > k
\]

... next?
Select(x, i) \get i-th element in subtree rooted at x.

\[ k \leftarrow 1 + \text{size}(l_x) \quad \text{\(l_x\): left child of } x \]

if \( i = k \), return \( x \).
else if \( i < k \), return Select(l_x, i)
else (i > k) return Select(r_x, i-k)

Example: \( i = 5 \)

\[ \begin{align*}
  k &= 6 \\
  \text{if } i < k &\Rightarrow \text{Select}(C, 5) \\
  k &= 1 + 1 \\
  \text{if } i > k &\Rightarrow \text{Select}(F, 3)
\end{align*} \]
Select($x, i$) \hspace{1em} \text{get } i\text{-th element in subtree rooted at } x.

\begin{align*}
k &\leftarrow 1 + \text{size}(l_x) \quad \text{\textsf{l}_x: 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) \quad \text{return } \text{Select}(r_x, i-k)
\end{align*}

example: $i=5$

- $k=6$
- $i=5, \quad k=2$
- $i=3, \quad k=2$

Select(root, 5)

\begin{align*}
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)
\end{align*}
Select(x, i) \Comment{get i-th element in subtree rooted at x.}
\begin{align*}
k &\leftarrow 1 + \text{size}(l_x) \Comment{l_x: 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)
\end{align*}

\begin{example}
i=5
\hspace{1cm} k=6
\hspace{1cm} i<k \Rightarrow \text{Select}(c, 5)
\hspace{1cm} k=1+1
\hspace{1cm} i>k \Rightarrow \text{Select}(F, 3)
\hspace{1cm} k=1+1
\hspace{1cm} i>k \Rightarrow \text{Select}(H, 1)
\hspace{1cm} k=1+0
\hspace{1cm} i=k \Rightarrow \text{return } H
\end{example}
**DYNAMIC SELECTION**

find the i-th smallest element in a set

**Static:** $\Theta(n)$

**Dynamic:** $O(n \log n)$ preprocessing $\rightarrow$ balanced BST w/ subtree sizes

$O(\log n)$ query / insert / delete