Binary Search Trees

Fields
Analysis
Pseudocode
Binary Search Trees

\( T.root \) is the root node of tree \( T \).

\( x.key \) is the key of node \( x \).
\( x.p \) is the parent of \( x \).
\( x.left \) is the left child of \( x \).
\( x.right \) is the right child of \( x \).

Binary search tree property:
If \( y \) is in left subtree of \( x \), then \( y.key < x.key \).
If \( y \) is in the right subtree, then \( x.key < y.key \).

\( n \) is the number of nodes and \( h \) is the height.
Dynamic Set Operations

Tree walking is $\Theta(n)$.

```
TREE-WALK(x)
    if x $\neq$ NIL
        then TREE-WALK(x.left)
        TREE-WALK(x.right)
```

Dynamic set operations are $O(h)$.
$h$ is $\Omega(lg n)$ and $O(n)$.

For $n$ nodes, $h$ can range from $\lfloor lg n \rfloor$ to $n - 1$.

A balanced binary tree has $\Theta(lg n)$ height.
A binary tree of height $h$ has $\leq 2^{h+1} - 1$ nodes.

Note $\lfloor \log_2(2^{h+1} - 1) \rfloor = h$ and $\lfloor \log_2(2^{h+1}) \rfloor = h + 1$

Let $N(h) = \text{maximum number of nodes in a binary tree of height } h$.

Basis: $N(0) = 1 = 2^1 - 1$

Assume: $N(h - 1) = 2^h - 1$

Show: $N(h) = 2^{h+1} - 1$

Induction: Subtrees have height at most $h - 1$

$N(h) = 2N(h-1) + 1 = 2(2^h - 1) + 1 = 2^{h+1} - 1$
Search

binary search trees
operations
number of nodes
▷ search
max and min
successor
insert
delete
delete 2
delete 3
delete 4

\[\text{Tree-Search}(x, k)\]

\begin{align*}
\text{if } x &= \text{NIL} \text{ or } k = x.key \\
\text{then return } x \\
\text{if } k &< x.key \\
\text{then return } \text{Tree-Search}(x.left, k) \\
\text{else return } \text{Tree-Search}(x.right, k)
\end{align*}
Maximum and Minimum

Tree-Maximum\((x)\)
\[
\text{while } x.\text{right} \neq \text{NIL} \\
\quad x \leftarrow x.\text{right} \\
\text{return } x
\]

Tree-Minimum\((x)\)
\[
\text{while } x.\text{left} \neq \text{NIL} \\
\quad x \leftarrow x.\text{left} \\
\text{return } x
\]
Successor

Tree-Successor($x$)

\[
\begin{align*}
\text{if } & \ x.\text{right} \neq \text{NIL} \\
\text{then return } & \text{Tree-Minimum}(x.\text{right}) \\
\quad & y \leftarrow x.p \\
\text{while } & \ y \neq \text{NIL} \text{ and } x = y.\text{right} \\
\quad & x \leftarrow y \\
\quad & y \leftarrow x.p
\end{align*}
\]

return $y$

Tree-Predecessor is similar.
Tree-Insert($T, z$)

$y \leftarrow$ NIL

$x \leftarrow T.root$

while $x \neq$ NIL

$y \leftarrow x$

if $z.key < x.key$

then $x \leftarrow x.left$

else $x \leftarrow x.right$

$z.p \leftarrow y$

if $y =$ NIL

then $T.root \leftarrow z$

else if $x.key < y.key$

then $y.left \leftarrow z$

else $y.right \leftarrow z$
Delete

Let \( x \) be the node to be deleted. Deletion has the following cases.

- If \( x \) has no children, replace \( x \) with NIL.
- If \( x \) has one child \( y \), replace \( x \) with \( y \).
- If \( x \) has two children, let \( y \) be the minimum of \( x.right \).
  - If \( y \) is a child of \( x \), replace \( x \) with \( y \).
  - If \( y \) is not a child of \( x \), replace \( y \) with \( y.right \), then replace \( x \) with \( y \).
In the following diagrams, node $B$ is being deleted from the tree.

$T_1$, $T_2$, and $T_3$ are subtrees or NIL.

In the last diagram, $T_4$ is a subtree whose minimum element is node $C$. 
Delete Part 3

binary search trees
operations
number of nodes
search
max and min
successor
insert
delete
delete 2
delete 3
delete 4
Delete Part 4