Joining and Splitting Binary Search Trees

JOIN($T_1, d, T_2$): A single tree with root $d$ is formed from $T_1$, $d$, and $T_2$.
   ** Always assumes that items in $T_1$ are $<d$ and items in $T_2$ are $>d$.

SPLIT($d, T$): Delete $d$ and split $T$ into items $<d$ and items $>d$.

- For ease of presentation, we assume that $d$ is in $T$.
- $T_1$ and $T_2$ are initialized to the left and right subtrees of $v$.
- As described on the next page, delete the vertex $v$ that contains $d$ and move up the path to the root to "unzip" the tree into two trees; $T_1$ the tree of all vertices $<d$ and $T_2$ the tree of all vertices $>d$.
- Each time we go up a left child edge to a vertex $w$, $w$ has data that is larger than everything seen so far (and $w$ should be merged into $T_2$).
- Each time we go up a right child edge to a vertex $w$, $w$ has data that is smaller than everything seen so far (and $w$ should be merged into $T_1$).
Example of the SPLIT Operation

Vertices 1 through 5 are shown, connected to them are subtrees of arbitrary size labeled $A$ through $F$, and a SPLIT on vertex 3 is depicted. $T_1$ is initialized to $C$ and $T_2$ to $D$. We then move up from 3 to the root, placing 2 and $B$ in $T_1$ (by making the root of $C$ a right child of 2), placing 4 and $E$ in $T_2$ (by making the root of $D$ a left child of 4), placing 5 and $F$ in $T_2$ (by making 4 a left child of 5), and finally placing 1 and $A$ in $T_1$ (by making 2 a right child of 1).
Detailed description of SPLIT

**procedure** SPLIT($d, T$):

Binary search down to the vertex $v$ that contains $d$.

$x := v.left$

$y := v.right$

**while** $v.parent \neq \text{nil}$ **do**

**if** $(v.parent).left = v$ **then begin** (*$v$ is a left child*)

$v := v.parent$

$v.left := y$

$y.parent := v$

$y := v$

**end**

**else begin** (*$v$ is a right child*)

$v := v.parent$

$v.right := x$

$x.parent := v$

$x := v$

**end**

$x.parent := \text{nil}$

$y.parent := \text{nil}$

**return** the two trees rooted at $x$ and $y$

**end**
Correctness: Can be verified by showing that each iteration of the while loop preserves the fact that $x$ and $y$ are roots of trees that contain elements less than and greater than $d$ respectively.

Time: Proportional to the length of the path from $v$ to the root.

Space: Since pointer fields are simply changed to form two trees from $T$, only $O(1)$ space is used in addition to the space used to store $T$. 