In a binary tree, a node's key is greater than or equal to its left child's key and less than or equal to its right child's key. In a binary search tree, the tree-seek(x) function is defined as follows:

1. If x is in the tree, return it.
2. If x is greater than the root, search the right subtree.
3. If x is less than the root, search the left subtree.

If x is not found, return a null pointer to indicate failure.
Program: DHT

Correctness guaranteed by the binary-search-tree property.

Search:

```
6 return x
5 the x - key(x)
4 then x = key(x)
3 do x = key(x)
2 while x ≠ key(x)
1 return x (key(x))
TREES: height(x)
```

Minimum/Maximum:

```
1 return x
0 if x ≠ key(x)
```

Successor/Predecessor:

```
x 0 predecessor of x is the element x in the subtree that is smaller than the key
x 1 successor of x is the smallest x in the subtree that is greater than the key

Assuming all keys are defined, and given a node x, the successor/predecessor of x in the inorder order is determined by the inorder tree walk.
```

So, the minimum should be found in the subtree rooted at x.left:

```
6 return x
5 the x - key(x)
4 then x = key(x)
3 do x = key(x)
2 while x ≠ key(x)
1 return x (key(x))
TREES: height(x)
```

Correctness: height(x) can readily be made heightive.
The diagram illustrates the process of finding the successor of a node in a binary search tree. The successor is defined as the node with the smallest key greater than the node's key. Here are the steps:

1. If the node has a right child, the successor is the leftmost node in the right subtree.
2. If the node has no right child, the successor is found by moving up the tree from the node to its parent, checking if the parent's left child is the node.

The diagram shows a binary search tree with nodes labeled A, B, C, D, E, and F. The successor of node D is node B, and the successor of node E is node F.
1. ... 3 determines a node \( y \) to splice out (\( y = z \) or \( y = \text{successor}(z) \)).
2. \( x \) is set to point to child of \( y \) (or to \( \text{nil} \)).
3. \( x \) splice out \( y \) by modifying pointers in \( p(y) \).
4. \( y \) is set to point to the next node.
5. \( z \) is set to point to the previous node.
6. \( \text{successor}(z) = y \).
7. return \( y \) so it can be recycled.

Complexity: \( O(h) \)

Replace the content of \( z \) with the contents of \( y \).
Replace one successor \( \langle z \rangle \) of \( y \) cannot have a null child.

\( z \) has two children.