The Push Operation, Reading

A push operation is done using the square bracket setter. To avoid duplicate keys, if a matching key is found in a node, we return its **V value**. But if there is no match, the end of the tree will be reached and the traversing "p" pointer will end up as zero. Unfortunately, that leaves us without knowledge of where "p" came from in the previous cycle, because that's where a new node should be attached.

A prev Pointer

We had the exact same situation when we studied linked queues, and considered how to deal with pushing new nodes at the back of the list when there was no tail pointer. Remember that we added a tail pointer as a *convenience* so the push operation would not have to traverse to find the last node. There we implemented a "prev" pointer to tag along behind "p", remembering where "p" has been in the preceding cycle. That's what we'll need now, in case we traverse to the end of a tree and need a pointer to what will be the parent to the new node.

```
Node* p = rootNode;
Node* prev = 0;
while (p)
{
  if (p->key == key) break; // found it!
  prev = p; // save p's old position before advancing it
  if (p->key < key) p = p->right; else p = p->left;
}
```

The different outcomes to expect from this loop are these

1. rootNode is zero: empty tree -- create a new node as the root
2. p equals prev: exited with the if-break -- p points to the matching node
3. there's no match, and prev is to be the parent of a new node

1. Adding The Root Node

There's no need for a special check of "rootNode" before the search loop. If it's zero, the **while(p)** loop does nothing, and we get to the end of the loop where we can check "rootNode". Then if it's needed, this code block adds it:

```
siz++;
rootNode = new Node;
rootNode->key = key;
rootNode->value = V( );
rootNode->left = rootNode->right = 0;
return rootNode->value;
```

or in C++11:

```
siz++;
rootNode = new Node{key, V( )};
return rootNode->value;
```

note that "left" and "right" will automatically get initialized to zero...

2. A Match

If "p" is not null when the loop ends, simply:

```
return p->value;
```

3. Adding A New Node Below prev

It's easy enough to create a new node, just as with the new root:

It's easy enough to create a new node, just as with the new root.

```
    siz++;
    p = new Node;
    p->key = key;
    p->value = V( );
    p->left = p->right = 0;
```

But how to attach it to the node pointed to by "prev"? First we have to know if it is to be a left child or right. Then we can attach it and return its value:

```
    if (prev->key < key) prev->right = p;
    else prev->left = p;
    return p->value;
```