Daily Coding Problem #278

Problem

This problem was asked by Amazon.

Given an integer `N`, construct all possible binary search trees with `N` nodes.

Solution

As with many tree problems, we can formulate this recursively. Let the range of values that nodes in a tree can take be bounded by `low` and `high`. If the root of the tree is `i`, the left subtree will hold data between `low` and `i - 1`, and the right subtree will hold data between `i + 1` and `high`.

For each possible value in the left subtree, we can choose one, say `j`, to be the root, which will again determine what data its left and right children can hold. This process continues until there are no more values to choose, in which case we add leaf nodes with the value `None`.

Here is what this process would look like if `N = 5`, and we start by choosing `i = 3` as our root node.

```
  3             3
 /             /
```

```
1          2
 \        /
  2      1
  |      |
None   None
```

At the same time, an analogous process can be carried out to find all the possible right subtrees.

Finally, for every possible root value, and every possible left and right subtree, we create a node with this value and the corresponding left and right children, and add this node to our list of possible trees.

```
class Node:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right

def make_trees(low, high):
    trees = []

    if low > high:
        trees.append(None)
        return trees

    for i in range(low, high + 1):
        left = make_trees(low, i - 1)
        right = make_trees(i + 1, high)

        for l in left:
            for r in right:
                node = Node(i, left=l, right=r)
                trees.append(node)

    return trees
```

To print out the tree, we can perform a preorder traversal.

```
def preorder(root):
```

```
    result = []

    if root:
        result.append(root.data)
        result += preorder(root.left)
        result += preorder(root.right)

    return result
```

Putting it all together, for a given input `N`, we first construct all trees that use values between `1` and `N`, and then iterate over each tree to print the nodes.

```
def construct_trees(N):
    trees = make_trees(1, N)
    for tree in trees:
        print(preorder(tree))
```

The number of possible binary search trees grows exponentially with the size of `N`, as can be seen by inspecting the first few values: `1, 2, 5, 14, 42, 132, 429, 1430, ...`. In fact, this sequence is defined by the Catalan numbers, which appear in a variety of combinatorial problems.

In any case, `make_trees` takes `O(N)` time to build each possible tree, so with an exponential number of trees, this algorithm will run in $O(N * 2^N)$ time. Since each tree takes `O(N)` space to store its nodes, the space complexity is $O(N * 2^N)$ as well.