Documentation: Solution for Unique Binary Search Trees

Description:

Given an integer n, return the number of structurally unique BSTs (binary search trees) that can be constructed using exactly n nodes with unique values from 1 to n.

Example 1:

Input: n = 3

Output: 5

Example 2:

Input: n = 1

Output: 1

Constraints:

- (1 leq n leq 19)

Solution Explanation

The solution uses dynamic programming to calculate the number of unique BSTs that can be formed with n nodes.

Dynamic Programming Approach

1. **DP Array Initialization:**

- Create a list dp of size n + 1 to store the number of unique BSTs for each count of nodes from 0 to n.
- Initialize dp[0] to 1 because there is exactly one empty tree.

2. **Filling the DP Array:**

- Iterate through the number of nodes from 1 to n.
- For each count of nodes, calculate the number of unique BSTs using the following logic:
  - *For each possible root node from 1 to the current count of nodes:*
    - ✓ The left subtree will have root 1 nodes.
    - ✓ The right subtree will have nodes root nodes.
    - ✓ The total number of unique BSTs for the current count of nodes is the sum of the products of the number of unique BSTs of the left and right subtrees for each possible root.
- Store the total in dp[nodes].

3. **Return the Result:**

- The final result is stored in dp[n].

Example Usage:

```
sol = Solution()

print(sol.numTrees(3))  # Output: 5

print(sol.numTrees(1))  # Output: 1
```

Detailed Explanation:

1. **Dynamic Programming Table (dp):**
   - dp[i] will store the number of unique BSTs that can be constructed with i nodes.
   - Initialize dp[0] to 1 because there is exactly one BST (the empty tree) with 0 nodes.

2. **Iterating through Nodes:**
   - *For each number of nodes nodes from 1 to n:*
     - Initialize total to 0 for calculating the number of unique BSTs for the current nodes.

3. **Calculating Unique BSTs:**
   - *For each possible root node from 1 to nodes:*
     - Calculate the number of nodes in the left subtree (left = root 1).
     - Calculate the number of nodes in the right subtree (right = nodes root).
     - Multiply the number of unique BSTs for the left subtree (dp[left]) by the number of unique BSTs for the right subtree (dp[right]) and add it to total.

4. **Updating the DP Table:**
   - After iterating through all possible root nodes for the current number of nodes, store the total in dp[nodes].

5. **Returning the Result:**
   - The final answer, i.e., the number of unique BSTs with n nodes, is stored in dp[n].

This dynamic programming approach ensures that the problem is solved efficiently with a time complexity of (O(n^2)) and a space complexity of (O(n)).