UNIQUE BINARY SEARCH TREE

COURSE : DESIGN ANALYSIS AND ALGORITHMS FOR AMORTIZED ANALYSIS

COURSE CODE:CSA0695

NAME:AJAY KUMAR

REG NO:192210590

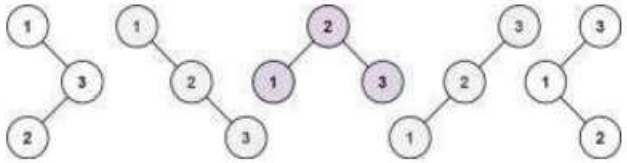
SUPERVISOR:

Dr.R.Dhanalakshmi

PROBLEM STATEMENT:

Given an integer n, return the number of structurally unique BST's (binary search trees) which has exactly n nodes of unique values from 1 to n.

Example1:



Input: n = 3
Output: 5
Example 2:
Input: n = 1
Output: 1
Constraints: 1 <= n <= 19

ABSTRACT:

Determining the number of structurally unique Binary Search Trees (BSTs) that can be constructed with n distinct nodes, each with unique values ranging from 1 to n, is a significant problem in the realms of combinatorics and dynamic programming. This problem can be elegantly solved using the concept of Catalan numbers, which are integral to counting various combinatorial structures.

INTRODUCTION

Binary Search Trees (BSTs) play a pivotal role in computer science, serving as a fundamental data structure that allows for efficient data organization and retrieval. Each BST is characterized by a unique property: for any given node, the values in the left subtree are smaller, and the values in the right subtree are larger. This hierarchical structure ensures that operations like search, insertion, and deletion can be performed swiftly, typically in logarithmic time. As a result, BSTs are integral to numerous applications, including database indexing and memory management, where fast access to data is crucial

DISCRIPTIONS:

KEY POINTS:

BST Property: In a Binary Search Tree (BST), for each node, the left subtree contains nodes with values less than the node, and the right subtree contains nodes with values greater than the node.

Structurally Unique Trees: Two trees are considered structurally unique if their shapes differ, even if they contain the same values.

APPLICATIONS:

Algorithm Design: Understanding this problem helps in grasping the concepts of dynamic programming, recursion, and combinatorics.

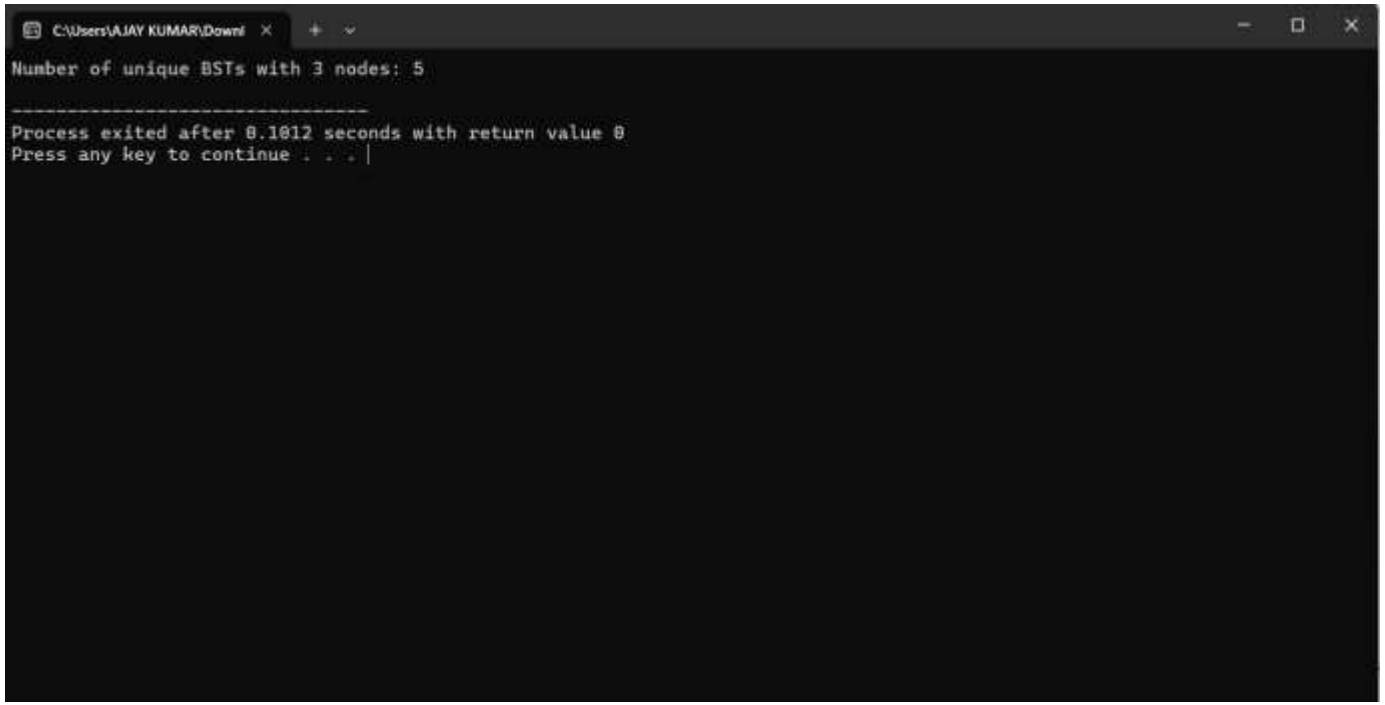
Tree Structures: Useful in various applications where tree structures are important, such as parsing expressions, building search engines, or designing data structures.

EXTENSIOS:

his problem can be extended by asking for:

- Construction of the actual unique trees.
- Counting the number of trees for large n using approximations

OUTPUT


C:\Users\AJAY KUMAR\Downl
Number of unique BSTs with 3 nodes: 5

Process exited after 0.1012 seconds with return value 0
Press any key to continue . . .

COMPLEXITY ANALYSIS

- **Time Complexity**: The algorithm has a time complexity of O(n^2). This arises because for each node count from 2 to n, the algorithm considers each node as a potential root and calculates the number of unique left and right subtrees, leading to a nested loop structure.

- **Space Complexity**: The space complexity is O(n) due to the dynamic programming array ‘dp’ that stores the number of unique BSTs for each count of nodes from 0 to n. This array is necessary to store intermediate results and avoid redundant calculations, making the solution efficient in terms of both time and space.

CASES

BEST CASE

The best-case scenario occurs when n is very small, such as n=0 or n=1. In these cases, the function quickly returns results 1 and 1, respectively, as there is only one way to arrange 0 or 1 nodes into a BST.

WORST CASE

The worst-case scenario is when n is large, as the algorithm must perform O(n^2)operations to compute the number of unique BSTs. For large values of n, the algorithm fully exercises the nested loop structure, iterating through all possible root nodes for each subtree configuration.

AVERAGE CASE

For average values of n, the time complexity remains O(n^2). The average case does not differ significantly from the worst case because the algorithm must consider every possible subtree configuration for each node count up to n

FUTURE SCOPE

The concept of unique binary search trees can be extended to scenarios involving memory or resource allocation. Since BSTs maintain sorted data efficiently, they can help optimize processes where search, insertion, and deletion operations are frequent. Databases use balanced search trees for indexing to optimize query times. . Understanding how many unique BSTs can be formed for different node counts helps in designing balanced trees that minimize time complexity. In AI and machine learning, decision trees are widely used for classification problems. BST structures could be utilized to explore how many distinct decision paths exist for various conditions.

CONCLUSION:

The problem of finding the number of structurally unique BSTs that can be formed with n distinct nodes is effectively addressed using dynamic programming. By recognizing that the number of unique BSTs for a given n can be derived from the unique BSTs of smaller subproblems, we can construct a solution that leverages Catalan numbers. This approach, with a time complexity of O(n^2) and space complexity of O(n), provides an efficient and robust method for solving this combinatorial problem. The dynamic programming solution.