Problem 96: Unique Binary Search Trees

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given an integer

n

, return

the number of structurally unique

BST'

s (binary search trees) which has exactly

n

nodes of unique values from

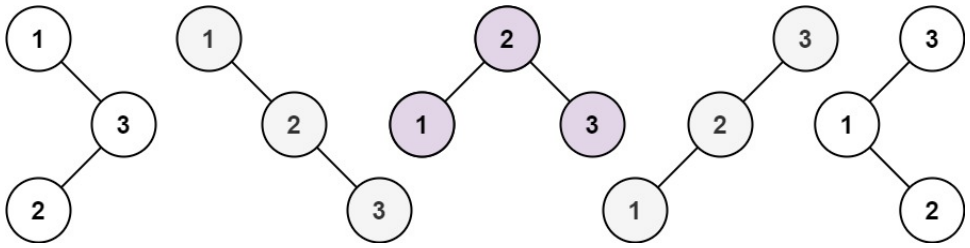
1

to

n

.

Example 1:



Input:

n = 3

Output:

5

Example 2:

Input:

n = 1

Output:

1

Constraints:

1 <= n <= 19

Code Snippets

C++:

```
class Solution {
public:
int numTrees(int n) {

}
};
```

Java:

```
class Solution {
public int numTrees(int n) {

}
}
```

Python3:

```
class Solution:
def numTrees(self, n: int) -> int:
```

Python:

```
class Solution(object):
def numTrees(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var numTrees = function(n) {

};
```

TypeScript:

```
function numTrees(n: number): number {

};
```

C#:

```
public class Solution {
public int NumTrees(int n) {
```

```
}
}
```

C:

```
int numTrees(int n) {

}
```

Go:

```
func numTrees(n int) int {

}
```

Kotlin:

```
class Solution {
fun numTrees(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func numTrees(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_trees(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def num_trees(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function numTrees($n) {

}
}
```

Dart:

```
class Solution {
int numTrees(int n) {

}
}
```

Scala:

```
object Solution {
def numTrees(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_trees(n :: integer) :: integer
def num_trees(n) do

end
end
```

Erlang:

```
-spec num_trees(N :: integer()) -> integer().
num_trees(N) ->
.
```

Racket:

```
(define/contract (num-trees n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Unique Binary Search Trees
* Difficulty: Medium
* Tags: tree, dp, math, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int numTrees(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Unique Binary Search Trees
* Difficulty: Medium
* Tags: tree, dp, math, search
*
* Approach: DFS or BFS traversal
```

```
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int numTrees(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Unique Binary Search Trees
Difficulty: Medium
Tags: tree, dp, math, search

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def numTrees(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numTrees(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Unique Binary Search Trees
* Difficulty: Medium
```

```
* Tags: tree, dp, math, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} n
* @return {number}
*/
var numTrees = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Unique Binary Search Trees
* Difficulty: Medium
* Tags: tree, dp, math, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

function numTrees(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Unique Binary Search Trees
* Difficulty: Medium
* Tags: tree, dp, math, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

public class Solution {
public int NumTrees(int n) {

}
}
```

C Solution:

```
/*
* Problem: Unique Binary Search Trees
* Difficulty: Medium
* Tags: tree, dp, math, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

int numTrees(int n) {

}
```

Go Solution:

```
// Problem: Unique Binary Search Trees
// Difficulty: Medium
// Tags: tree, dp, math, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(n) or O(n * m) for DP table

func numTrees(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numTrees(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numTrees(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Unique Binary Search Trees
// Difficulty: Medium
// Tags: tree, dp, math, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn num_trees(n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def num_trees(n)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer $n
* @return Integer
*/
function numTrees($n) {

}
}
```

Dart Solution:

```
class Solution {
int numTrees(int n) {

}
}
```

Scala Solution:

```
object Solution {
def numTrees(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_trees(n :: integer) :: integer
def num_trees(n) do

end
end
```

Erlang Solution:

```
-spec num_trees(N :: integer()) -> integer().
num_trees(N) ->
.
```

Racket Solution:

```
(define/contract (num-trees n)
(-> exact-integer? exact-integer?)
)
```