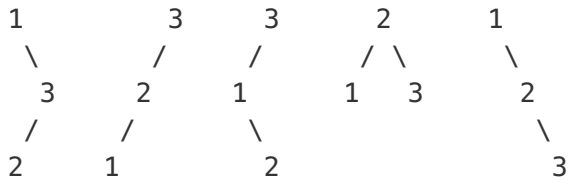
Unique Binary Search Trees

Given *n*, how many structurally unique **BST's** (binary search trees) that store values 1...*n*?

For example,
Given *n* = 3, there are a total of 5 unique BST's.



Solution 1

The problem can be solved in a dynamic programming way. I'll explain the intuition and formulas in the following.

Given a sequence 1...n, to construct a Binary Search Tree (BST) out of the sequence, we could enumerate each number i in the sequence, and use the number as the root, naturally, the subsequence 1...(i-1) on its left side would lay on the left branch of the root, and similarly the right subsequence (i+1)...n lay on the right branch of the root. We then can construct the subtree from the subsequence recursively. Through the above approach, we could ensure that the BST that we construct are all unique, since they have unique roots.

The problem is to calculate the number of unique BST. To do so, we need to define two functions:

`G(n)` : the number of unique BST for a sequence of length n.

`F(i, n), 1 <= i <= n` : the number of unique BST, where the number i is the root of BST, and the sequence ranges from 1 to n.

As one can see, `G(n)` is the actual function we need to calculate in order to solve the problem. And `G(n)` can be derived from `F(i, n)`, which at the end, would recursively refer to `G(n)`.

First of all, given the above definitions, we can see that the total number of unique BST `G(n)`, is the sum of BST `F(i)` using each number i as a root. *i.e.*

```
G(n) = F(1, n) + F(2, n) + ... + F(n, n).
```

Particularly, the bottom cases, there is only one combination to construct a BST out of a sequence of length 1 (only a root) or 0 (empty tree). *i.e.*

```
G(0)=1, G(1)=1.
```

Given a sequence 1...n, we pick a number i out of the sequence as the root, then the number of unique BST with the specified root `F(i)`, is the cartesian product of the number of BST for its left and right subtrees. For example, `F(3, 7)`: the number of unique BST tree with number 3 as its root. To construct an unique BST out of the entire sequence [1, 2, 3, 4, 5, 6, 7] with 3 as the root, which is to say, we need to construct an unique BST out of its left subsequence [1, 2] and another BST out of the right subsequence [4, 5, 6, 7], and then combine them together (*i.e.* cartesian product). The tricky part is that we could consider the number of unique BST out of sequence [1,2] as `G(2)`, and the number of of unique BST out of sequence [4, 5, 6, 7] as `G(4)`. Therefore, `F(3,7) = G(2) * G(4)`.

i.e.

```
F(i, n) = G(i-1) * G(n-i)   1 <= i <= n
```

Combining the above two formulas, we obtain the recursive formula for `G(n)`. *i.e.*

```
G(n) = G(0) * G(n-1) + G(1) * G(n-2) + … + G(n-1) * G(0)
```

In terms of calculation, we need to start with the lower number, since the value of `G(n)` depends on the values of `G(0) … G(n-1)`.

With the above explanation and formulas, here is the implementation in Java.

```java
public int numTrees(int n) {
    int [] G = new int[n+1];
    G[0] = G[1] = 1;

    for(int i=2; i<=n; ++i) {
        for(int j=1; j<=i; ++j) {
            G[i] += G[j-1] * G[i-j];
        }
    }

    return G[n];
}
```

written by liaison original link here

Solution 2

```
/**
 * Taking 1~n as root respectively:
 *      1 as root: # of trees = F(0) * F(n-1)  // F(0) == 1
 *      2 as root: # of trees = F(1) * F(n-2)
 *      3 as root: # of trees = F(2) * F(n-3)
 *      ...
 *      n-1 as root: # of trees = F(n-2) * F(1)
 *      n as root:   # of trees = F(n-1) * F(0)
 *
 * So, the formulation is:
 *      F(n) = F(0) * F(n-1) + F(1) * F(n-2) + F(2) * F(n-3) + ... + F(n-2) * F(1
) + F(n-1) * F(0)
 */

int numTrees(int n) {
    int dp[n+1];
    dp[0] = dp[1] = 1;
    for (int i=2; i<=n; i++) {
        dp[i] = 0;
        for (int j=1; j<=i; j++) {
            dp[i] += dp[j-1] * dp[i-j];
        }
    }
    return dp[n];
}
```

written by leo.mao original link here

Solution 3

WE can know that by zero we can have 1 bst of null by 1 we have 1 bst of 1 and for 2 we can arrange using two ways Now idea is simple for rest numbers. for n=3 make 1 as root node so there will be 0 nodes in left subtree and 2 nodes in right subtree. we know the solution for 2 nodes that they can be used to make 2 bsts. Now making 2 as the root node , there will be 1 in left subtree and 1 node in right subtree. ! node results in 1 way for making a BST. Now making 3 as root node. There will be 2 nodes in left subtree and 0 nodes in right subtree. We know 2 will give 2 BST and zero will give 1 BST. Totalling the result of all the 3 nodes as root will give 5. Same process can be applied for more numbers.

```
    public int number(int n){
    if(n==0)return 1;
        if(n==1)return 1;

        int result[]=new int [n+1];
        result[0]=1;
        result[1]=1;
        result[2]=2;
        if(n<3){
            return result[n];
        }

        for(int i=3;i<=n;i++){
            for(int k=1;k<=i;k++){

                result[i]=result[i]+result[k-1]*result[i-k];
            }
        }


        return result[n];
    }
```

written by ajak6 original link here