# 题目

给定一个整数 n，求以1 ...n为节点组成的二叉搜索树有多少种？

**示例:**

输入: 3

输出: 5

解释:

给定 n = 3, 一共有 5 种不同结构的二叉搜索树:

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

# 分析

## 方法一：递归

可以考虑使用递归来计算二叉搜索树的种数。对于每个节点值i，可以将其作为根节点，左子树的节点值范围为 [1, i-1]，右子树的节点值范围为[i+1, n]。然后分别计算左右子树的二叉搜索树种数，将其相乘即可得到以i为根节点的二叉搜索树的种数。

以下是使用递归实现的代码：

class Solution {

public:

int numTrees(int n) {

return count(1, n);

}

int count(int start, int end) {

if (start >= end) {

return 1;

}

int totalCount = 0;

for (int i = start; i <= end; i++) {

int leftCount = count(start, i - 1);

int rightCount = count(i + 1, end);

totalCount += leftCount \* rightCount;

}

return totalCount;

}

};

这个递归算法的时间复杂度较高，为指数级别的复杂度，不推荐在 `n` 较大时使用。如果需要更高效的解法，建议使用动态规划。

## 方法二：动态规划

这是一个经典的动态规划问题，可以使用动态规划来解决。定义一个数组dp，其中dp[i]表示由i个节点组成的不同二叉搜索树的种数。初始时dp[0] = 1，因为空树也算一种二叉搜索树。

对于每个i，我们可以遍历1到i，以每个数字j作为根节点，将树分为左右子树，左子树有j-1个节点，右子树有i-j个节点。则以j为根节点的二叉搜索树的种数为dp[j-1] \* dp[i-j]。最后将所有情况累加起来即可。

以下是实现这个算法的代码：

class Solution {

public:

int numTrees(int n) {

vector<int> dp(n + 1, 0);

dp[0] = 1;

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

for (int j = 1; j <= i; j++) {

dp[i] += dp[j - 1] \* dp[i - j];

}

}

return dp[n];

}

};

这个算法的时间复杂度是 O(n^2)，其中 n 是节点数。

class Solution {

public:

int numTrees(int n) {

vector<int> G(n + 1, 0);

G[0] = 1;

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];

}

};