给定一个整数`n`,计算从`1~n`能够构成多少二叉搜索树

分析： 递归方式：

1. 对`1 <= index <= n`,其二叉树的个数为`sum += count(1, index-1) * count(index+1, n)`;
2. 所以对`1~n`个数字总的二叉搜索树个数为：

   ```
   for(int i=lo, i <= hi; i++) {
     sum += count(1, index-1) * count(index+1, n);
   }
   ```

```
class Solution {
public:
  int numTrees(int n) {
    return numTrees(1, n);
  }

private:
  int numTrees(int lo, int hi) {
    if (lo > hi) {
      return 1;
    }

    int sum = 0;
    for (int i = lo; i <= hi; i++) {
      int left  = numTrees(lo, i - 1);
      int right = numTrees(i + 1, hi);

      sum += left * right;
    }

    return sum;
  }
};
```

通过对递归计算的分析，我们可以得知，在上述计算中存在多个重复计算。 采用备忘录的方法进行优化。`memo[i][j]表示从i到j一共存在多少个二叉树`。

```
class Solution {
public:
  int numTrees(int n) {
    memo = std::vector<std::vector<int>>(n + 1, std::vector<int>(n + 1, 0));

    numTrees(1, n);

    return memo[1][n];
```

```
  }

private:
  int numTrees(int lo, int hi) {
    if (lo > hi) {
      return 1;
    }

    if (memo[lo][hi] != 0) {
      return memo[lo][hi];
    }

    int sum = 0;
    for (int i = lo; i <= hi; i++) {
      int left  = numTrees(lo, i - 1);
      int right = numTrees(i + 1, hi);
      sum += left * right;
    }

    memo[lo][hi] = sum;

    return memo[lo][hi];
  }

  std::vector<std::vector<int>> memo;
};
```

给定一个整数`n`，求所有能构成二叉搜索树的集合

```
class Solution {
public:
  std::vector<TreeNode *> generateTrees(int n) {
    std::vector<TreeNode *> res;
    if (n == 0) {
      return res;
    }

    return generateTrees(1, n);
  }

private:
  std::vector<TreeNode *> generateTrees(int lo, int hi) {
    std::vector<TreeNode *> res;
    if (lo > hi) {
      res.push_back(nullptr);
      return res;
    }

    // 穷举所有的可能
    for (int i = lo; i <= hi; i++) {
      std::vector<TreeNode *> leftTree  = generateTrees(lo, i - 1);
      std::vector<TreeNode *> rightTree = generateTrees(i + 1, hi);

      // 将所有的节点连成tree，放入数组
      for (auto left : leftTree) {
        for (auto right : rightTree) {
          TreeNode *root = new TreeNode(i);
          root->left     = left;
          root->right    = right;
          res.push_back(root);
        }
      }
    }

    return res;
  }
};
```