**数据结构**

2022

**实 验 报 告**

|  |  |
| --- | --- |
| 实验项目名称： | 数据结构期中考试 |
| 班级： | 21级9班 |
| 学号 | 2021302121185 |
| 姓名： | 陈聪睿 |
| 指导教师： | 董红斌 |
| 实验时间： | 2022.4.25 |

**实验一：**

**使用递归求在给定二叉树的结点总数N的情况下，**

**二叉树可能拥有的形状数M**

**一、实验要求**

设计一个递归算法，求在给定二叉树的结点总数N的情况下，二叉树可能拥有的形状数M。请看如下说明和要求：

1. 满足要求的任何一棵二叉树都是高度为N的满二叉树从根结点开始的子树。将这棵满二叉树的结点按照从上至下、从左至右的顺序进行编号，根结点的编号为1，则可以按层次输出任何结点总数为N的二叉树的所有结点编号。例如当N=3时，输出结果为：

1: 1, 2, 3

2: 1, 2, 4

3: 1, 2, 5

4: 1, 3, 6

5: 1, 3, 7

tree\_count is 5 when N is 3

1. 递归函数的原型是：void arrange(int arr[],int idx,int N,int &tree\_count)；其中arr是存放编号序列的数组，idx是当前需要计算的数组元素的下标，N是结点总数也是数组长度，tree\_count记录二叉树的数目。
2. 并不需要实际构造出一棵二叉树，只需要把各种可能的合法编号序列计算出来就能统计二叉树的数目，因此重点是编写算法arrange()，要求输出每一个合法的序列；
3. 请通过程序验证M和N之间满足卡塔兰数的关系，即。
4. 每一棵二叉树的高度是由其最后（最大）的结点编号决定的，设此编号为n，则二叉树的高度是。将arrange()函数的原型改造一下，变成void arrange(int arr[],int idx,int N,int &tree\_count,int &height)；用参数height记录所有二叉树的总高度，并利用height/M来计算平均高度；
5. 猜测并验证二叉树的平均高度与log2N之间的关系；
6. 当N逐渐增大，例如N>19，上述代码就会出错，原因是什么？如何修改？

**二、实验环境**

Visual studio 2019/ Windows平台

**三、实验步骤及思路**

**（一）编写arrange函数**

根据题目要求，可以了解到本实验并不需要构建出一棵二叉树，而是可以通过一棵满二叉树，通过截取其部分，可获得目标二叉树。

**1. 目标二叉树与满二叉树的关系**

该目标二叉树与满二叉树有如下关系：

1. 该二叉树的根结点就是满二叉树的根结点；
2. 若该二叉树的某一非子叶结点m，对应着满二叉树的n号结点，则m号结点的左孩子lchild可以表示为2\*n，右孩子rchild可表示为2\*n+1；
3. 该二叉树的可能结点的选取范围，为满二叉树1~N层的所有结点。

**2.arrange函数的编写**

该问题的核心，就在于arrange递归函数的编写。

arrange函数原型如下：

void arrange(int arr[],int idx,int N,int &tree\_count)；

其中：

arr是存放编号序列的数组，在其中，按照数组顺序存放着满二叉树的结点编号。当存放的个数等于N时，若按照数组顺序输出，则可以得到一棵满足要求的目标二叉树。

idx是当前需要计算的数组元素的下标，可以理解为目标二叉树需要与其他元素进行比较的位置。

N是结点总数也是数组长度。

tree\_count记录二叉树的数目。

**2.1 arrange函数递归部分的详细解释**

关于arrange的递归部分，我们可以根据二叉树性质，可以知道arr数组的第i位，存放的结点必定为前i-1个结点中的某一结点的孩子节点或兄弟结点。

同时因为我选择先比较左孩子，再比较右孩子，又因为满二叉树的编号问题，所以第i位编号必定大于前面的任何一个结点的编号。

于是我们可以将递归部分的思路总结如下：

当我们需要从满二叉树，寻找到目标结点编号，插入arr数组的idx+1号位时，

（1）将1~idx的所有结点的孩子结点找出，并将其的左孩子与右孩子与arr[idx]内存放的结点号进行比较；

（2）如果大于，证明其为所需结点，插入到arr[idx+1],并递归调用arrange函数，寻找下一位结点；

（3）如果小于或等于，则证明其不为所需结点，需要更换比较对象。如果我们比较的是左孩子，则比较右孩子；如果我们比较的是右孩子，则需要回溯到上一结点，去比较该结点的兄弟结点。

**2.2对于arrange函数递归出口的解释**

当N==idx时，证明找到了一棵满足要求的二叉树，则将tree\_count加一，同时将arr数组里的元素按照数组顺序依次输出，并结束本次递归。并回溯到该idx的前一位，并继续查找可能结点，直到所有的可能结点全部遍历。

在回溯过程中，需要将arr[idx+1]置为0，防止其对兄弟节点判断时产生影响。

**（二）验证M和N之间满足卡塔兰数的关系**

在本题中，需要将count\_tree与Catalan数进行比较。

Catalan数编写方法如下：

（1）通过循环，将每次循环的相乘，计算出（计算时需要分开算，以免出现运算过程中出现小数，而使结果出现误差），存在temp中。

（2）将temp/（N+1），并放入result后输出result。

**（三）改造函数，验证二叉树的平均高度与log2N的关系**

**1.arrange函数的改造**

（1）根据题目提示可知，每棵二叉树高度的求法可以转化为：每次得到新生成的子树后，求比arr[idx]（最后一个叶节点的编号）小的最大2的m次方，m即为本次子树高度的结果，将本次结果加入总高度height中。arrange函数运算结束后，即为最终结果。

（2）平均高度求法为：avgheight=(float)height/tree\_count。(注，因为我设置height是个int类型数据，需要将height进行强制类型转换)

**2.二叉树的平均高度与的关系**

根据推算，其关系为：平均高度=

因为N可以想成是：按照满二叉树结点的顺序依次选取而成的二叉树的最后（最大）的结点编号，根据每棵二叉树高度的求法，可以近似认为是最矮二叉树的高度，而二叉树的平均高度基本上会高于其最矮二叉树。又因为最高二叉树是以N为高度的二叉树，故其平均高度会为其2倍。

**（四）当N逐渐增大，上述代码就会出错的原因**

最高二叉树的叶节点结点编号为2N。因为存储其编号的数组arr[]的是用的int格式存储，可能会最大范围。

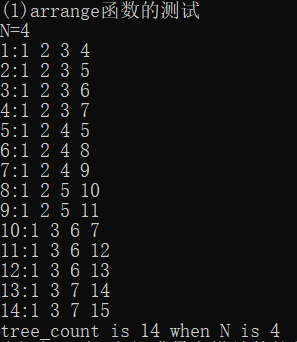
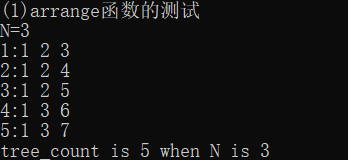
同时，当N达到一定程度时，根据卡塔兰数推算，其结果也可能超过int的存储范围。

故改正方法为将arr[]和tree\_count的存储方式改为long long。

**四、实验结果及分析**

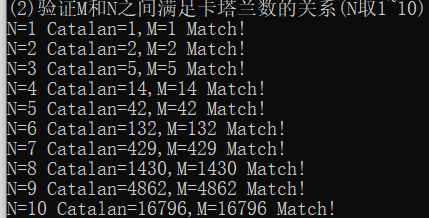
**（一）arrange函数的测试**

测试数据：N=3、N=4；



**（二）验证M和N之间满足卡塔兰数的关系**

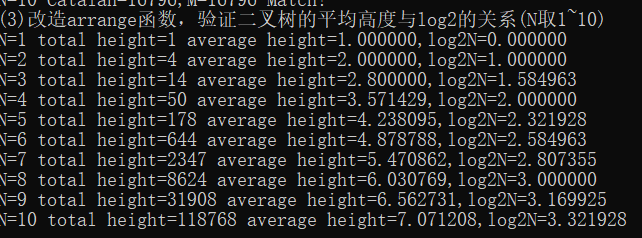
测试数据：N取1~10。



可以看出卡特兰数的所有结果与M都相等。

**（三）改造函数，验证二叉树的平均高度与的关系**

测试数据：N取1~10。



可以看出其关系接近于average height=，猜想成立。

**实验二： 用非递归（迭代）算法解决问题一**

**一、实验要求**

用非递归（迭代）算法解决问题一。函数原型为：void buildtree(int N,int &tree\_count);即送入结点总数N，得到二叉树总数目tree\_count。整个算法的总体结构如：

…

while(idx>0){

…

if(…)

idx++;

else

idx--;

…

}

同问题一，需要设定一个长度为N的数组int arr[],idx为数组下标，当可以顺利确定当前下标上的结点编号之后，idx会自增以完成下一个位置上的计算。当idx到达N时，说明找到问题的一个解（即发现一棵新的二叉树），打印输出数组的内容。

语句idx--;是在当前位置上的所有合法取值都已经试验完毕，或者是刚刚输出了一个可行解之后执行的。

请仔细设计算法使得它具有尽可能低的算法复杂度。

**二、实验环境**

**Visual studio 2019/ Windows平台**

**三、实验步骤及思路**

本实验可以根据递归算法的思路来写，与递归算法不同的地方，有如下几点：

1. idx++；是在原idx后可插入该结点编号时使用的语句；idx--；是在当前位置上的所有合法取值都已经试验完毕，或者是刚刚输出了一个可行解之后执行的。
2. 因为未使用递归算法，会导致我们输出一个解后，将重新进入循环，去判断已经判断过的可以输出的解。为了避免一直重复输出同一个结果，于是我们不应该立刻将arr[idx]置零，而是让其作为判断标准，来衡量左右子树是否考虑重复。

其原则是要插入的结点编号必须大于前面考虑过的所有结点编号。

在所有合法的值都已实验完毕后，才将其置零。

1. 判断结果考虑重复，以及是否为合理解的办法，是使用如下的if语句判断而得到的：

（注：flag在这里是用来判断所有合法的值是否都已实验完毕，true为完成，false为未完成）

//判断左孩子是否满足条件且不跟之前的解重复

if (lchild > arr[idx+1] && lchild > arr[idx]) {

idx++;

arr[idx] = lchild;

flag = false;

break;

}

//判断右孩子是否满足条件且不跟之前的解重复

else if (rchild > arr[idx+1] && rchild > arr[idx]) {

idx++;

arr[idx] = rchild;

flag = false;

break;

}

//判断是否跟之前的解重复，若重复直接结束当前循环，进入下一次循环

else if (lchild <= arr[idx] || rchild <= arr[idx])

continue;

//当前位置上的所有合法取值都已经试验完毕

else

flag = true;

}

if (flag) {

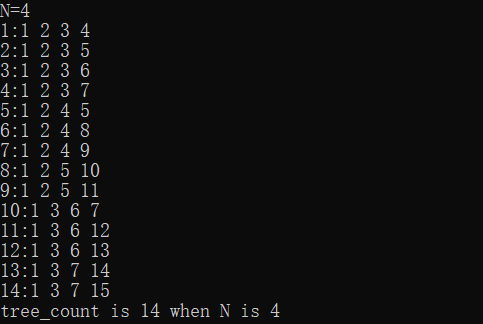
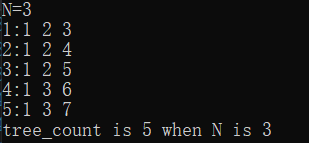
arr[idx+1] = 0;

idx--;

}

**四、实验结果及分析**

测试数据：n=3,n=4



可以看出与题目一的结果相同。

**五、总结**

**（一）实验收获**

两个实验的难点都在于如何使用一个线性数组去思考二叉树相关的问题。

根据提示，我们可以合理的将一个二叉树的问题转化为线性问题。如果我们只从二叉树定义的角度下手，本次可能会十分麻烦和困难，但是，我们可以通过二叉树编号间的关系，将其模拟成线性数组的序号，将“寻找子树”这一问题转化为线性的“求合法编号序列”问题，困难就能够迎刃而解。

同时，如何合理运用递归与迭代，也是这个实验的难点。

设计递归算法时，我们需要找好将大问题分割成小问题的方法，同时也要明确何时为递归出口；设计迭代算法时，我们可以参考递归算法的求解思想，在递归的基础上做改动，将其变为非递归。同时，递归在程序运行时，为我们省去了许多麻烦，可是使用非递归算法时，这些问题会出现，需要对其加以修改、补全。

**（二）存在的问题**

以上算法仍有优化之处，如在判断是否为目的结点时，我们对于部分数据，可以优化查找的范围，不需要总是从1一直查询到idx，这样可以节省部分时间，从而提升运算性能。