数据结构

实 验 报 告

实验项目名称： 数据结构实验期中考试

班级： 自强实验班

学号： 2020302131250

姓名： 王朝辉

指导教师： 沈志东

实验时间： 2021.5.3

实验一

一、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以 1 为根结点，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

（2） 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。

（3） 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；

（4） 请通过程序验证 M 和 N 之间满足卡塔兰数的关系，即$M=\frac{1}{N+1}C_{2N}^{N}$。

（5） 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$\lceil \log_2(n+1) \rceil$。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/tree_count 来计算平均高度；

（6） 猜测并验证二叉树的平均高度与 $\log_2 N$ 之间的关系；

（7） 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？

一、实验要求

（1）独立完成实验

（2）撰写实验报告

二、实验环境

软件环境：windows 10

硬件环境：Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 16.0G RAM

三、实验步骤及思路

1. 编写 arrange 函数

注意到题目的要求是并不需要构建实际的二叉树，而只是求所有的合法编号序列。因此，可以选择将这棵所谓的二叉树理解为一串数组，任何非叶子结点 x 的孩子结点分别为 2*x 和 2*x+1。而递归的思路体现于，要求 idx+1 位的合法编号序列，可以在已知 idx 位合法编号

序列的基础上，找出一个合法的第 idx+1 位元素。显然，该递归函数的递归出口是编号序列的长度等于 N，此时输出该编号序列并返回。

基于该算法思路，编写出的具体代码如下：

```
//输出数组
void Disparr(int arr[], int& tree_count, int N) {
    tree_count++;
    printf("%d:", tree_count);
    for (int i = 1; i <= N; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

//查找子串
void arrange(int arr[], int idx, int N, int& tree_count) {

//arr   存放编号序列的数组
//idx   当前需要计算的数组元素下标
//N   结点总数与数组长度
//tree_count   二叉树数目

    if (idx == N) {//递归出口，找到一个子串
        Disparr(arr, tree_count, N);
        return;
    }

    for (int i = 1; i <= idx; i++) {
        int lchild = arr[i] * 2;//arr[i]的左孩子
        int rchild = arr[i] * 2 + 1;//arr[i]的右孩子
        if ( lchild > arr[idx]) {
            arr[idx + 1] = lchild;
            arrange(arr, idx + 1, N, tree_count);
            arr[idx + 1] = 0;
        }
        if ( rchild > arr[idx]) {
            arr[idx + 1] = rchild;
            arrange(arr, idx + 1, N, tree_count);
            arr[idx + 1] = 0;
        }
    }
}
```

用于在后续实验中检验该函数正确性的 main 函数如下：

```
int main() {
    int arr[51], tree_count = 0, N;
```

```
    arr[1] = 1;//根据题目要求，编号序列的第一位只能是1
    printf("Input N:");
    scanf_s("%d", &N);
    arrange(arr, 1, N, tree_count);
    printf("tree_count is %d when N is %d\n", tree_count, N);
    return 0;
}
```

2.验证 m 与 n 之间的关系

要验证 M 与 N 之间的关系，首先应该编写计算卡塔兰数的函数，再将各个 N 下的 m 与卡塔兰数进行比较，即可验证 M 与 M 之间的关系。

具体代码如下：

```
//计算N的卡塔兰数
int Catalan(int N) {
    int C = 1, Ncatalan;
    for (int i = 1; i <= N; i++) {
        C *= (2 * N + 1 - i);
        C /= i;
    }
    Ncatalan = C / (N + 1);
    return Ncatalan;
}

//arrange函数的简化版，仅仅用来计算M
void Simplearrange(int arr[], int idx, int N, int& tree_count) {
    if (idx == N) {//递归出口，找到一个子串
        tree_count++;
        return;
    }
    for (int i = 1; i <= idx; i++) {
        int lchild = arr[i] * 2;//arr[i]的左孩子
        int rchild = arr[i] * 2 + 1;//arr[i]的右孩子
        if (lchild > arr[idx]) {
            arr[idx + 1] = lchild;
            Simplearrange(arr, idx + 1, N, tree_count);
            arr[idx + 1] = 0;
        }
        if (rchild > arr[idx]) {
            arr[idx + 1] = rchild;
            Simplearrange(arr, idx + 1, N, tree_count);
            arr[idx + 1] = 0;
        }
    }
}
```

```
//用于验证 M 与 N 之间的关系
int main() {
    int N;
    for (N = 1; N <= 10; N++) {
        int tree_count = 0, arr[108], Ncatalan;
        arr[1] = 1;
        Simplearrange(arr, 1, N, tree_count);
        Ncatalan = Catalan(N);
        printf("tree_count is %d, and Catalan is %d, when N is %d\n", tree_count,
Ncatalan, N);
    }
}
```

3. 验证平均高度与 $\log_2 N$ 的关系

首先应编写计算平均高度的 arrange 函数，具体代码如下：

```
//arrange函数的修改版，能够计算平均高度
void arrange2(int arr[], int idx, int N, int& tree_count, int& height) {
    if (idx == N) {
        tree_count++;
        height += int(log2(arr[idx]));
        return;
    }
    for (int i = 1; i <= idx; i++) {
        int lchild = arr[i] * 2;//arr[i]的左孩子
        int rchild = arr[i] * 2 + 1;//arr[i]的右孩子
        if (lchild > arr[idx]) {
            arr[idx + 1] = lchild;
            arrange2(arr, idx + 1, N, tree_count, height);
            arr[idx + 1] = 0;
        }
        if (rchild > arr[idx]) {
            arr[idx + 1] = rchild;
            arrange2(arr, idx + 1, N, tree_count, height);
            arr[idx + 1] = 0;
        }
    }
}
```

猜想平均高度与 $\log_2 N$ 的关系应该是：平均高度约为 $\log_2 N$ 的二倍

下面在 main 函数中对该猜想进行验证，具体代码如下：

```
//用于验证平均高度与log2N之间的关系
int main() {
    int arr[51], tree_count = 0, N, height = 0, AveHeight;
    arr[1] = 1;
```

```
    for (N = 1; N <= 10; N++) {
        tree_count = 0; height = 0;
        arrange2(arr, 1, N, tree_count, height);
        printf("AveHeight is %lf, and log2(N) is %lf when N is %d\n", (double) height /
tree_count, (log2(N)), N);
    }
    return 0;
}
```

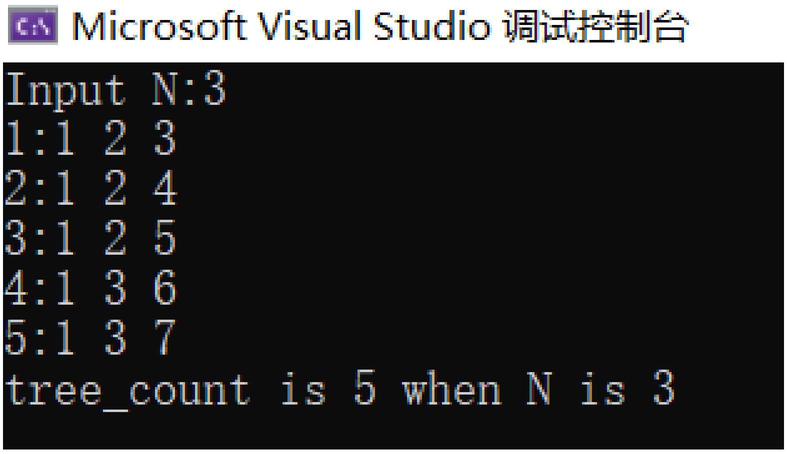
4. 解释并解决 N>19 时代码出错的问题

代码出错原因：根据 M 与卡塔兰数的关系来计算，可知，当 N>19 时，M 将大于 2^32，即超出了 int 类型所能表示的存储范围，产生数值溢出，造成代码出错。

解决方法：可以将 M（即 tree_count）的数据类型由 int 变为 long long int，就能计算 N>19 的情况。

四、实验结果及分析

1. 找出长度为 N 的子树



Microsoft Visual Studio 调试控制台

```
18:1 2 4 5 11
19:1 2 4 8 9
20:1 2 4 8 16
21:1 2 4 8 17
22:1 2 4 9 18
23:1 2 4 9 19
24:1 2 5 10 11
25:1 2 5 10 20
26:1 2 5 10 21
27:1 2 5 11 22
28:1 2 5 11 23
29:1 3 6 7 12
30:1 3 6 7 13
31:1 3 6 7 14
32:1 3 6 7 15
33:1 3 6 12 13
34:1 3 6 12 24
35:1 3 6 12 25
36:1 3 6 13 26
37:1 3 6 13 27
38:1 3 7 14 15
39:1 3 7 14 28
40:1 3 7 14 29
41:1 3 7 15 30
42:1 3 7 15 31
tree_count is 42 when N is 5
```

2. 验证 M 与 N 之间的关系

Microsoft Visual Studio 调试控制台

```
tree_count is 1, and Catalan is 1, when N is 1
tree_count is 2, and Catalan is 2, when N is 2
tree_count is 5, and Catalan is 5, when N is 3
tree_count is 14, and Catalan is 14, when N is 4
tree_count is 42, and Catalan is 42, when N is 5
tree_count is 132, and Catalan is 132, when N is 6
tree_count is 429, and Catalan is 429, when N is 7
tree_count is 1430, and Catalan is 1430, when N is 8
tree_count is 4862, and Catalan is 4862, when N is 9
tree_count is 16796, and Catalan is 16796, when N is 10
```

3.验证平均高度与 log2N 之间的关系

Microsoft Visual Studio 调试控制台

```
AveHeight is 0.000000, and log2(N) is 0.000000 when N is 1
AveHeight is 1.000000, and log2(N) is 1.000000 when N is 2
AveHeight is 1.800000, and log2(N) is 1.584963 when N is 3
AveHeight is 2.571429, and log2(N) is 2.000000 when N is 4
AveHeight is 3.238095, and log2(N) is 2.321928 when N is 5
AveHeight is 3.878788, and log2(N) is 2.584963 when N is 6
AveHeight is 4.470862, and log2(N) is 2.807355 when N is 7
AveHeight is 5.030769, and log2(N) is 3.000000 when N is 8
AveHeight is 5.562731, and log2(N) is 3.169925 when N is 9
AveHeight is 6.071207, and log2(N) is 3.321928 when N is 10
```

实验二

二、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。

一、实验要求

(1) 独立完成实验

(2) 撰写实验报告

二、实验环境

软件环境：windows 10
硬件环境：Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 16.0G RAM

三、实验步骤及思路

编写该迭代函数时，大体上的思路可以借鉴递归函数。可以将在实验一中递归的思想通过迭代算法来表现出来。

该迭代算法的具体代码如下：

```
//非递归算法找子树
void buildtree(int N, int& tree_count) {
    int arr[108], idx = 0, i;
    bool first = true, flag;
    while (idx >= 0) {
        flag = 1;
        if (idx == 0 && first) {
            arr[idx] = 1;
            idx++;
            first = false;
        }
        if (idx == N) {   //成功找到一个子树
            tree_count++;
            printf("%d:", tree_count);
            for (int i = 0; i < N; i++) {
                printf("%d ", arr[i]);
            }
            printf("\n");
            idx--;   //往前退一位
        }
        for (i = 0; i < idx; i++) {
            int lchild = 2 * arr[i];   //左孩子
            if (lchild > arr[idx] && lchild > arr[idx - 1]) {
                arr[idx] = lchild;
                idx++;
                flag = 0;
                break;
            }
            int rchild = 2 * arr[i] + 1;   //右孩子
            if (rchild > arr[idx] && rchild > arr[idx - 1]) {
                arr[idx] = rchild;
                idx++;
                flag = 0;
                break;
            }
```

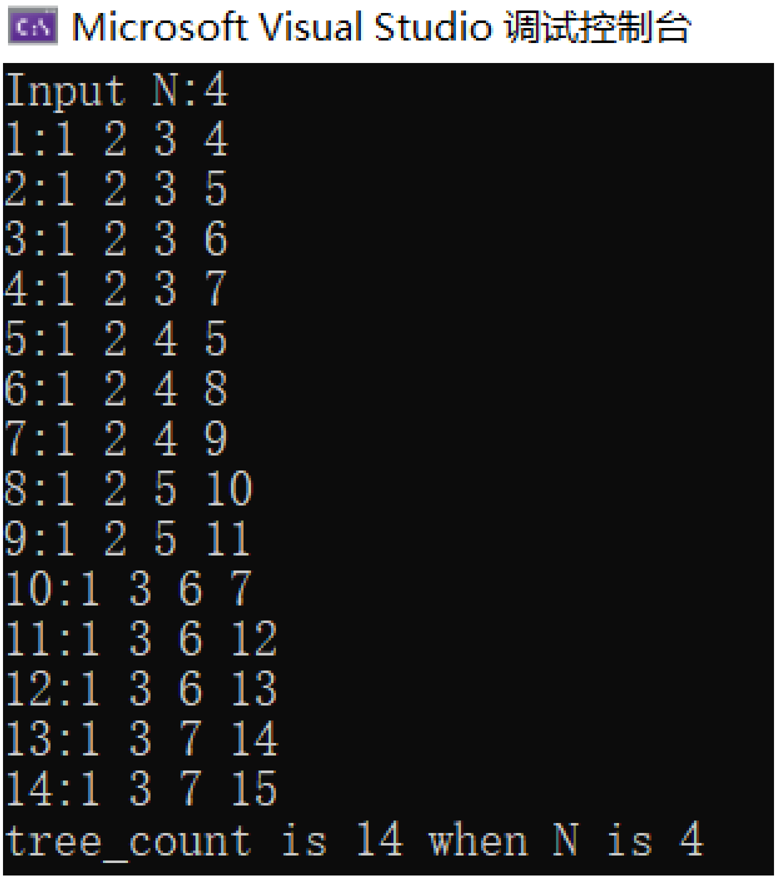
```
        }
        if (flag) {
            arr[idx] = 0;
            idx--;
        }
    }
}
```

用于验证的 main 函数代码如下:

```
int main() {
    int tree_count = 0, N;
    printf("Input N:");
    scanf_s("%d", &N);
    buildtree(N, tree_count);
    printf("tree_count is %d when N is %d\n", tree_count, N);
    return 0;
}
```

四、实验结果及分析

实验结果如下:



五、总结

（1）实验难点：该实验的难点在于如何跳出传统的二叉树的思维，从一个全新的角度去思考二叉树相关的问题。二叉树作为一对多的数据结构，有时很难把握，我们可以合理的将其转化为线性问题，再在线性问题的基础上解决二叉树问题。正如本次实验中，如果我们只从二叉树定义的角度下手，本次可能会十分困难，但是，我们将“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。设计递归算法时要注意大问题与小问题之间的联系；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。

（2）实验收获：通过本次实验，我对于二叉树有了更深层次的理解，也能够更加熟练地运用递归算法与迭代算法。同时，通过递归算法与迭代算法的比较，我了解到两种算法各自的优缺点，以及清楚的认识到两者之间的关系，即任何一种递归算法其实都能够转化为迭代算法。这对我今后的编程与实验都有很大的帮助。

（3）实验疑点：在本次实验后，我仍然有一些疑问，希望能通过后续学习更加深入了解。我的疑问有以下几点：
1.对于平均高度与 $\log_2 N$，我发现想要找出两者之间的关系略显困难，我对于这两者的关系并没有十足的把握。并且，通过输出数据来验证，个人认为存在较大的误差与不确定性，我想知道我们是否有更好的办法（仍然是通过编程），来找出并验证两个数据之间的关系。
2.我们可以将递归算法经过改动，将其变为迭代算法，问题在于，反过来，是否能将迭代算法转变为递归算法呢？如果能，则原来的迭代算法需要满足哪些条件？如何进行转变？如何能保证经过转变后的递归算法，在算法性能上优于原本的迭代算法？（因为算法性能低是递归算法的缺点之一）。这些疑问，我今后也会多加学习，希望能通过学习来解决这些疑问。