Tree Traversal

Date: 2021-10-24

Chapter 1: Introduction

Give the results of a partial traversal of a binary tree, in order, in pre-order, and in post-order. You should output the complete result and the corresponding tree traversal sequence in the hierarchical order.

Each input file contains one test case. For each case, a positive integer N (≤ 100) is given in the first line. Then there are three lines containing incomplete traversal sequences for in-order, pre-order, and post-order, respectively. It is assumed that the tree nodes are numbered from 1 to N. No number is given beyond this range. - represents a missing number.

For each case, print the complete in-order, pre-order, and post-order traversal sequences in four lines, as well as the level-order traversal sequence for the corresponding tree. The numbers must be separated by spaces and there must be no extra spaces at the beginning or at the end of each line. If it is not possible to reconstruct a unique tree from the given information, simply print Impossible.

Chapter 2: Algorithm Specification

Using the exhaustive method, first get all the possible numbers of the whole binary tree, then compare it with the middle order traversal array and post order traversal array, complete the middle order and post order arrays, and then get the pre order traversal value through the build function, If this value is the same as the input preorder traversal array except for the '-' symbol, and only one combination method can achieve this effect, then the preorder, middle order and post order traversal under this arrangement will be output in turn. If this effect cannot be achieved, then "impossible" will be output

The main functions used are as follows:

```
void scan(int* pre_order,int* in_order,int* post_order,int number)
```

```
void rand_sort(int step,int number,int* num,int add_num[][number],int n,int
flag)
```

```
void make_all_num(int number,int* pre_order,int* in_order,int* post_order,int*
num_all)
```

```
void make_add_num(int number,int* pre_order,int* in_order,int* post_order,int*
num_all,
                    int in_order_add_num[][number],int post_order_add_num[]
[number])
```

```
void build(int start,int end,int* post_order,int* in_order,int root,int*
pre_order,int print_flag)
```

```
void complete(int number,int in_completed[][number],int post_completed[]
[number],
                int add_index_in,int add_index_post,
                int* in_order,int* post_order,
                int in_order_add_num[][number],int post_order_add_num[][number])
```

```
void level_output(int* in_order,int* post_order,int root,int start,int end)
```

Next, we introduce the function in turn.

scan()

The first is the scan function, `scan(int* pre_order,int* in_order,int* post_order,int number)` ,This function is used to save the input, The entered numbers will be saved into three arrays of "in_order", "pre_order" and "post_order". If a '-' symbol is encountered, it will be converted to - 3 and saved in the corresponding array

The scan function reads numbers in string format, and then subtracts 48 from the read string to obtain the corresponding digital value

rand_sort()

The second function is rand_ sort(), `rand_sort(int step,int number,int* num,int add_num[][number],int n,int flag)` , This function is used to randomly sort the incoming num array, and use a flag to determine that the random remake is in_ Order array or post_ Order array, which uses recursion to realize the full arrangement of the incoming array, and the initial step is 1

make_all_num()

The next function is make_ all_ num(), the function is to make all numbers, such as input

```
9
3 - 2 1 7 9 - 4 6
9 - 5 3 2 1 - 6 4
3 1 - - 7 - 6 8 -
```

Then this function will return `9 5 3 2 1 6 4 7 8`, and these numbers will be saved in num_ all array

make_add_num()

make_ add_ The num function returns all possible mid order and post order arrays, and all results will be saved in_ order_ add_ num and post_ order_ add_ num array

This function is for in_ order, pre_ order, post_ Order traverses three arrays, if there is num_ If there is no number in the all array and it is not equal to - 3, fill this number in num_ all array

build()

The function of the build function is to generate a preamble array according to the input mid order array and post order array. If the obtained preamble array and the input preamble array have different numbers, and the position of this number in the input preamble array is not '-', then return to end this possible traversal

complete()

The complete function is used to complete the pre order array and post order array. The completion result is saved in_ Completed and post_ Completed in two arrays

level_output()

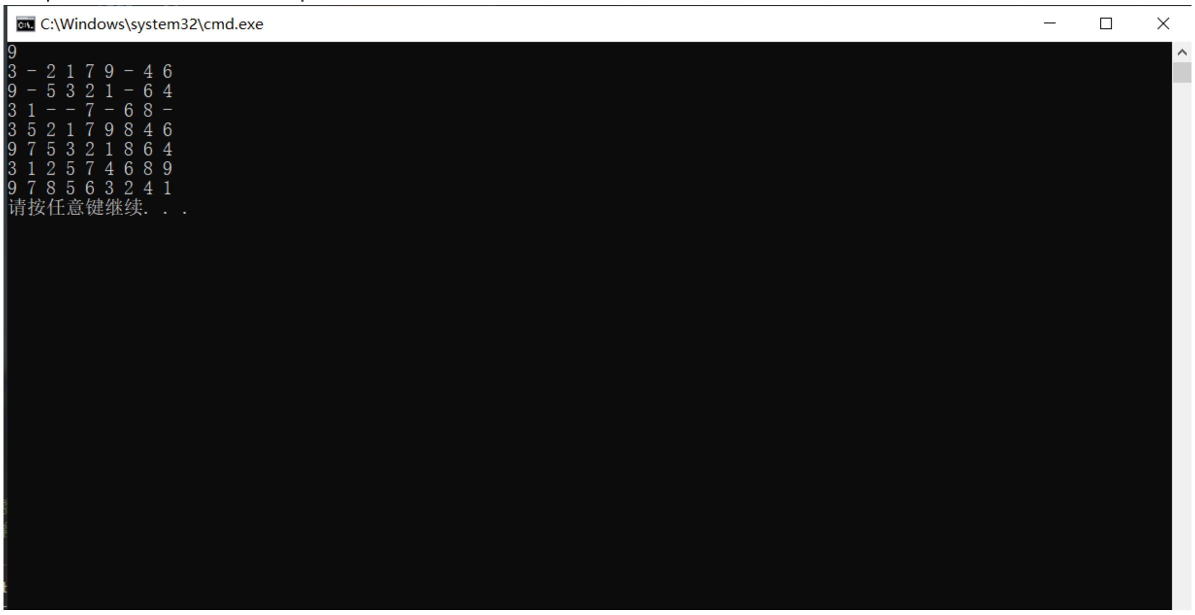
The function of this function is similar to the build() function, which obtains the hierarchical traversal array through the input medium order array and post order array

main()

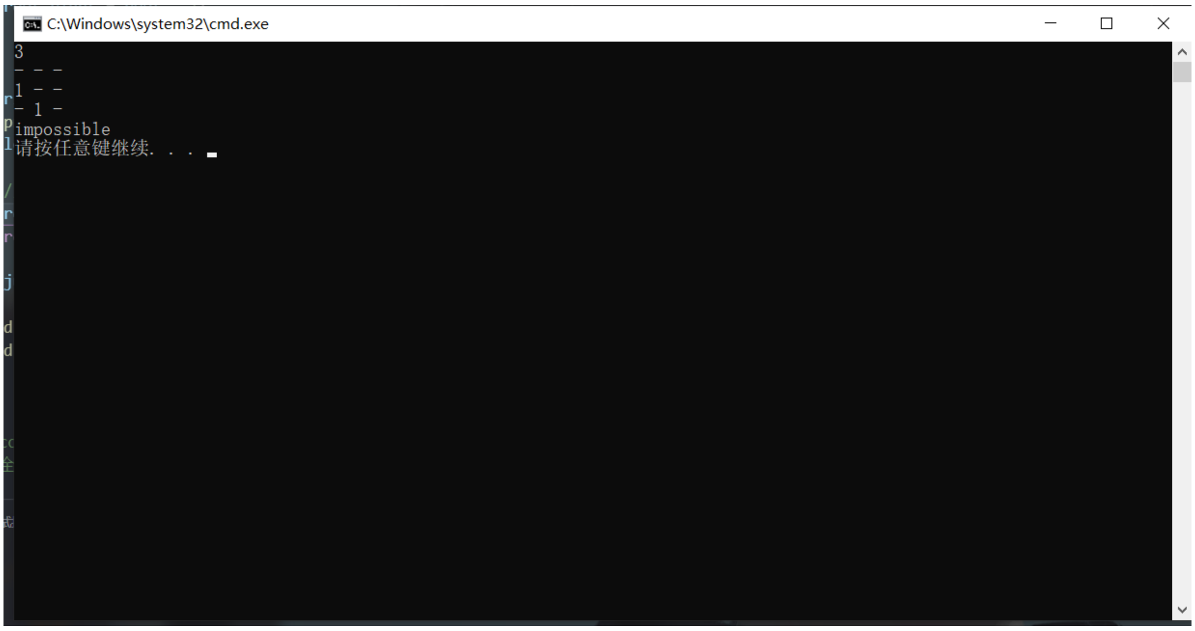
The main function first calls the `scan()` function to get the input, which is stored in the in_order, pre_order, post_order arrays, and then calls the `make_all_num()` function to get all the numbers, which are stored in the num_all array, and then calls the `make_add_num` function then call the `make_add_num` function and call the `rand_sort()` function to get all possible mid-order and post-order arrays, which are stored in the two-dimensional arrays in_order_add_num[][number] and post_order_num[][num], and after getting these two arrays, then These arrays are saved in the in_completed[][number] and post_completed[][number] arrays, and finally each possible combination of in-order and post-order arrays is passed to the build function through the two-level for loop. function generates the corresponding preorder array, if the generated preorder array matches and is unique to the incoming preorder array, the binary tree can be rebuilt, if not, "Impossible" is output

Chapter 3: Testing Results

The results of the two examples given in the question are as follows, and you can see that the output is the same as expected



```
C:\Windows\system32\cmd.exe
9
3 - 2 1 7 9 - 4 6
9 - 5 3 2 1 - 6 4
3 1 - - 7 - 6 8 -
3 5 2 1 7 9 8 4 6
9 7 5 3 2 1 8 6 4
3 1 2 5 7 4 6 8 9
9 7 8 5 6 3 2 4 1
请按任意键继续. . .
```



```
3
- - -
1 - -
- 1 -
impossible
请按任意键继续. . .
```

After that, we input a random set of data, which corresponds to a binary tree and the output result is as follows:

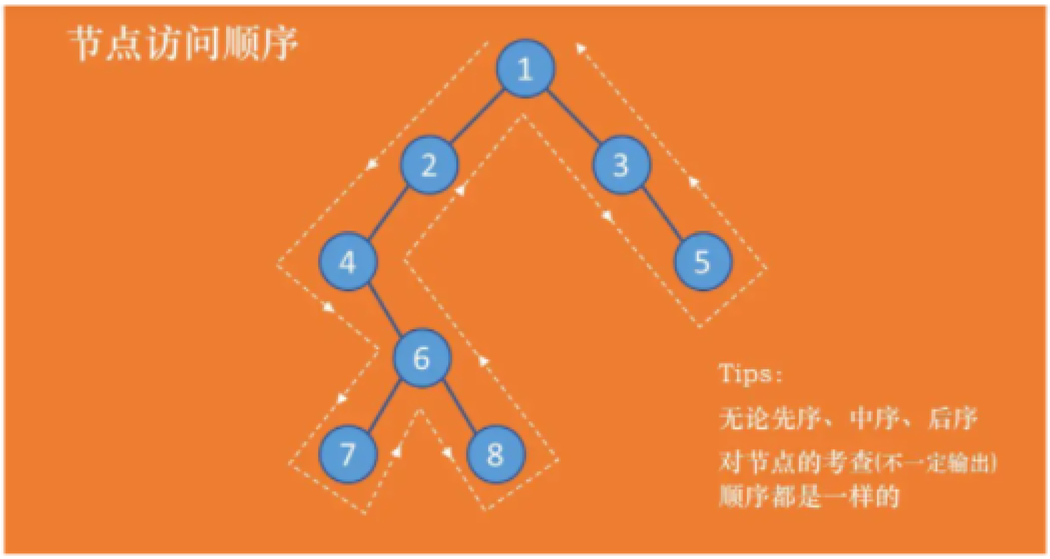
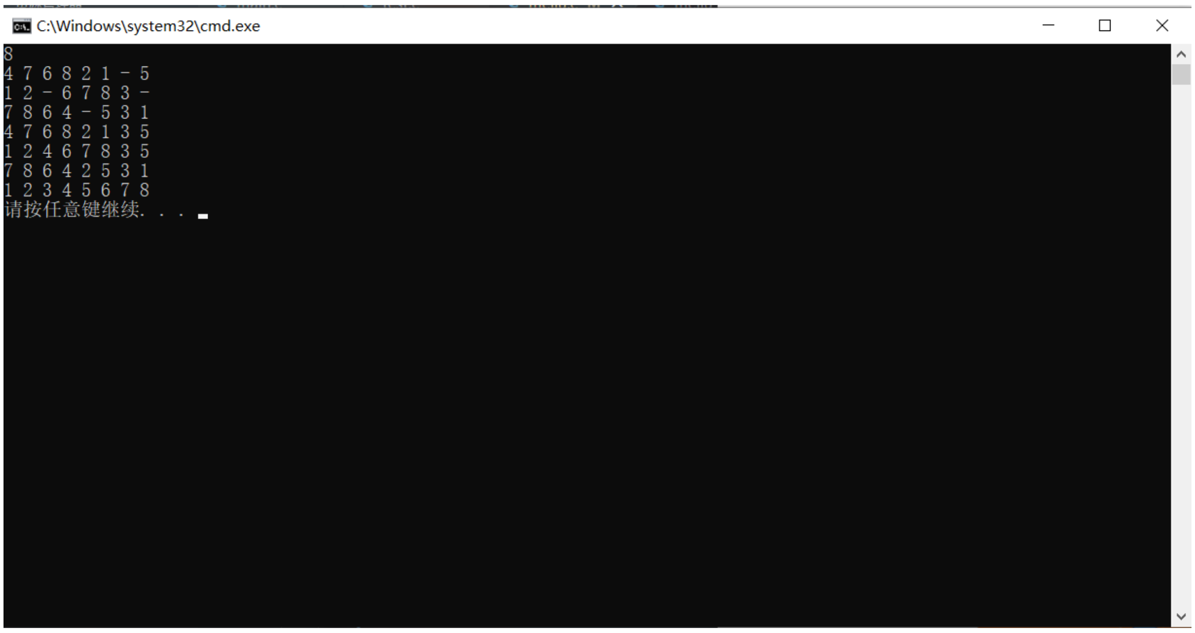


图2：先序、中序、后序遍历节点考查顺序

如图1所示，三种遍历方法(人工)得到的结果分别是：

> 先序：1 2 4 6 7 8 3 5
> 中序：4 7 6 8 2 1 3 5
> 后序：7 8 6 4 2 5 3 1



```
C:\Windows\system32\cmd.exe
8
4 7 6 8 2 1 - 5
1 2 - 6 7 8 3 -
7 8 6 4 - 5 3 1
4 7 6 8 2 1 3 5
1 2 4 6 7 8 3 5
7 8 6 4 2 5 3 1
1 2 3 4 5 6 7 8
请按任意键继续. . .
```

Chapter 4: Analysis and Comments

Time Complexity Analysis

Due to the `rand_sort` function, up to NUMBER numbers (i.e., all -) can be missing from a single array of three arrays in the preorder, middle, and postorder, in which case a full alignment of n numbers requires n! operations, all with a time complexity of O(n!)
However, since two for loops + the build function are used at the end to get every possible preorder output, the final time complexity is O(n^3)

Spatial Complexity Analysis

Since we use two-dimensional arrays to store all possible mid-order traversal arrays and post-order traversal arrays, the maximum possible storage space used is O(n!*n), and at this point, assuming that there is a traversal with all '-' inputs, the space complexity of the algorithm is O(n!*n)

Appendix

```c
#include<stdio.h>
#include <stdlib.h>
int add_num[100][100];
int n,a[100],book[100];
int make_num_flag = 0;
int add_index_in = 0,add_index_post = 0;
int pre_judge = 0;
int level_order[100000];
int level_order_index = 1;

/*
函数名：scan
功能：将输入保存到三个数组里，'-'以-3表示
*/
void scan(int* pre_order,int* in_order,int* post_order,int number){
    char
pre_order_index[number],in_order_index[number],post_order_index[number];
    char ch;
    for(int i = 0;i<number;i++){
        scanf(" %c",&in_order_index[i]);
        ch = getchar();
        in_order[i] = in_order_index[i] - 48;     //由于是以字符串形式读入的，所以需要-48，如果输入的是-，则得到的是-3
    }
    //ch = getchar();
    for(int i = 0;i<number;i++){
        scanf(" %c",&pre_order_index[i]);
        ch = getchar();
        pre_order[i] = pre_order_index[i] - 48;
    }
    //ch = getchar();
    for(int i = 0;i<number;i++){
        scanf(" %c",&post_order_index[i]);
        ch = getchar();
        post_order[i] = post_order_index[i] - 48;
    }
}

/*
名称：rand_sort()
功能：用于对num数组进行全排列,重排以后的数组存在add_num中
参数：  step:当前步 || n:排序几个数组数||  flag:表示调用哪个add_index,flag=1调用add_index_in,flag=0调用add_index_post|| n:num数组有几个数字
*/
void rand_sort(int step,int number,int* num,int add_num[][number],int n,int flag){
    int i;
    if(step == n+1){//当递归到最后一位时，储存这个序列
        for(i = 1;i<=n;i++){
            if(flag == 1)
                add_num[add_index_in][i-1] = a[i];
            else if(flag == 0)
                add_num[add_index_post][i-1] = a[i];
        }
        if(flag == 1)
            add_index_in++;
        else if(flag == 0)
            add_index_post++;
        return;
    }
    for (i = 1; i <= n; i++){
        if (book[i] == 0){
            a[step] = num[i-1];
            book[i] = 1;//标记当前位置
            rand_sort(step + 1,number,num,add_num,n,flag);
            book[i] = 0;
        }
    }
    return;
}

/*
函数名:make_all_num
功能：制作后序和中序中需要填上的数的数组，
比如说输入
中：3 - 2 1 7 9 - 4 6
```

```
前：9 - 5 3 2 1 - 6 4
后：3 1 - - 7 - 6 8 -
则输出9 5 3 2 1 6 4 7 8
全体数字保存在num_all数组中
*/
void make_all_num(int number,int* pre_order,int* in_order,int* post_order,int*
num_all){
    //int num_all[number];
    int index = 0;
    //先把pre_order数组中的非'-'数存到num_all数组中
    for(int i = 0;i<number;i++){
        if(pre_order[i] != -3)
            num_all[index++] = pre_order[i];
    }

    //如果当前的数不等于'-',且在num_all数组中都没有出现过，则保存该数
    for(int i = 0;i<number;i++){
        for(int j = 0;j<index;j++){
            if(in_order[i] == num_all[j] || in_order[i] == -3){
                make_num_flag = 1;
                break;
            }
        }
        if(make_num_flag == 0)
            num_all[index++] = in_order[i];
        else if(make_num_flag == 1)
                make_num_flag = 0;
    }
    //同理，遍历后序数组
    for(int i = 0;i<number;i++){
        for(int j = 0;j<index;j++){
            if(post_order[i] == num_all[j] || post_order[i] == -3){
                make_num_flag = 1;
                break;
            }
        }
        if(make_num_flag == 0)
            num_all[index++] = post_order[i];
        else if(make_num_flag == 1)
                make_num_flag = 0;
    }
    /*
    printf("所有数为：");
    for(int i = 0;i<index;i++){
        printf("%d ",num_all[i]);
    }
    */
}

/*
函数名:make_add_num
功能：返回所有需要被填上的数的数组，
如num_all为{1,2,3,4},in_order = {1,2,-,-},
则输出in_order_add[0] = {3,4},in_order_add[1] = {4,3}
------------------------------------------------------
这个函数对in_order, pre_order, post_order三个数组进行遍历，
如果有num_all数组中没有的数且不等于-3的数，
那么把这个数填入num_all数组中
```

```
*/
void make_add_num(int number,int* pre_order,int* in_order,int* post_order,int*
num_all,
                    int in_order_add_num[][number],int post_order_add_num[]
[number]){

                int
in_oeder_number_index[number],post_order_number_index[number];
                int in_index = 0,post_index = 0;
                int add_flag_in = 1,add_flag_post = 1;
                for(int i = 0;i<number;i++){
                    for(int j = 0;j<number;j++){
                        if(num_all[i] == in_order[j])
                            add_flag_in = 0;//说明num_all[i]能在in_order中找到，所以
无需加
                    }
                    if(add_flag_in == 1){
                        in_oeder_number_index[in_index++] = num_all[i];
                    }
                    else if(add_flag_in == 0)
                        add_flag_in = 1;
                }

                for(int i = 0;i<number;i++){
                    for(int j = 0;j<number;j++){
                        if(num_all[i] == post_order[j])
                            add_flag_post = 0;//说明num_all[i]能在in_order中找到，所
以无需加
                    }
                    if(add_flag_post == 1){
                        post_order_number_index[post_index++] = num_all[i];
                    }
                    else if(add_flag_post == 0)
                        add_flag_post = 1;
                }

                /*
                printf("\nin_order_number_index为：");
                for(int i = 0;i<in_index;i++)
                    printf("%d ",in_oeder_number_index[i]);

                printf("\npost_order_number_index为：");
                for(int i = 0;i<post_index;i++)
                    printf("%d ",post_order_number_index[i]);
                */
               //对序列进行全排列

 rand_sort(1,number,in_oeder_number_index,in_order_add_num,in_index,1);

 rand_sort(1,number,post_order_number_index,post_order_add_num,post_index,0);
                //printf("\nadd_index_in = %d",add_index_in);
                //printf("\nin_index = %d",in_index);

                /*
                for(int i = 0;i<add_index_in;i++){
                    printf("\nin_order_add_num[%d] = :",i);
                    for(int j = 0;j<in_index;j++){
                        printf("%d ",in_order_add_num[i][j]);
                    }
                }
                for(int i = 0;i<add_index_post;i++){
                    printf("\npost_order_add_num[%d] = :",i);
                    for(int j = 0;j<post_index;j++){
                        printf("%d ",post_order_add_num[i][j]);
                    }
                }
                */

}


//接下来还要把用中序后序得到前序的函数copy过来，主函数里写两层for循环，每层都填上不同的中序后序，并建立树，得到的树与前序对比，除了-以外的都一样且只有一种情况则为possible

int print_flag_1 = 1;
/*
函数名:build()
功能：根据输入的中序和后序生成前序
    每生成一个前序数都和原来的前序数对比，如果一样或者原来的前序数组在这里是'-',
    进行下一步，不一样则返回false
    print_flag用于表示要不要输出前序数字
*/
void build(int start,int end,int* post_order,int* in_order,int root,int*
pre_order,int print_flag,int level_index){
    if(start > end)return;
    int index;
    int root_left;
    int root_right;
    int flag = 0;
    if(start != end){
        for(int i = start;i<=end;i++){
            if(in_order[i] == post_order[root]){
                index = i;
                flag = 1;
            }
        }
        if(flag == 0)return;
        else if(flag == 1){
            flag = 0;
        }
        root_left = root - end + index - 1;
        root_right = root - 1;
    }
    else index = start;

    //当进行到最后的build的时候，print_flag置1，输出前序遍历的数
    if(print_flag == 1){
        if(print_flag_1 == 1){
            printf("%d",in_order[index]);
            print_flag_1 = 0;
        }
        //保证格式，如果是最后一个数，末尾没有空格
        else printf(" %d",in_order[index]);
        level_order[level_index] = in_order[index];
    }
```

```
        //printf("start = %d, end = %d, number = %d, index = %d root = 
%d\n",start,end,in_order[index],index,root);
    if(pre_order[pre_judge] != in_order[index] && pre_order[pre_judge] != -3){
        return;
    }
    pre_judge++;

    build(start,index-
1,post_order,in_order,root_left,pre_order,print_flag,2*level_index);//求当前节点的
左子节点

 build(index+1,end,post_order,in_order,root_right,pre_order,print_flag,2*level_i
ndex+1);//求当前节点的右子节点
}

/*
函数名：complete
功能：补全中序和后序，将结果保存在in_completed和post_completed上
*/
void complete(int number,int in_completed[][number],int post_completed[]
[number],
                int add_index_in,int add_index_post,
                int* in_order,int* post_order,
                int in_order_add_num[][number],int post_order_add_num[][number])
{
   int in_index_add = 0,post_index_add = 0;

    //对中序数组进行补全
    for(int i = 0;i<add_index_in;i++){
        for(int j = 0;j<number;j++){
            if(in_order[j] != -3)
                in_completed[i][j] = in_order[j];
            else{
                in_completed[i][j] = in_order_add_num[i][in_index_add++];
            }
        }
        in_index_add = 0;
    }

    //对后序数组进行补全
    for(int i = 0;i<add_index_post;i++){
        for(int j = 0;j<number;j++){
            if(post_order[j] != -3)
                post_completed[i][j] = post_order[j];
            else{
                post_completed[i][j] = post_order_add_num[i][post_index_add++];
            }
        }
        post_index_add = 0;
    }

    /*
    for(int i = 0;i<add_index_in;i++){
        printf("\nin_completed[%d] = ",i);
        for(int j = 0;j<number;j++){
            printf("%d ",in_completed[i][j]);
        }
    }
    for(int i = 0;i<add_index_post;i++){
        printf("\npost_completed[%d] = ",i);
        for(int j = 0;j<number;j++){
            printf("%d ",post_completed[i][j]);
        }
    }
    */
}

/*
函数名：level_output()
功能：利用中序和后序遍历输出层次遍历
*/
/*
void level_output(int* in_order,int* post_order,int root,int start,int end){
    if(start > end)return;

    int i = start;
    printf("%d ",post_order[root]);
    while(i <= end && in_order[i] != post_order[root])
        i++;

    level_output(in_order,post_order,root-end+i-1,start,i-1);
    level_output(in_order,post_order,root-1,i+1,end);
}
*/

int final_flag = 0;
int in_count,post_count;
int main(){
    int number;//数组长度
    char ch;//用于去除换行
    scanf("%d",&number);//读取
    ch = getchar();//去除\n
    int pre_order[number],in_order[number],post_order[number];//定义前序中序后序数组
    int level_print_flag = 1;//定义层序遍历输出信号

    //读取
    scan(pre_order,in_order,post_order,number);//调用scan函数进行输入的读取
    int num_all[number];//定义全部数的数组
    make_all_num(number,pre_order,in_order,post_order,num_all);//制作这个数组
    int in_order_add_num[1000][number];//定义用来存放中序数组全部可能性的二维数组
    int post_order_add_num[1000][number];//同理，定义用来存放后序数组全部可能性的二维数
组

 make_add_num(number,pre_order,in_order,post_order,num_all,in_order_add_num,post
_order_add_num);

    //补全
    int in_completed[add_index_in][number],post_completed[add_index_post]
[number];//定义补全完以后的中序&后序数组

 complete(number,in_completed,post_completed,add_index_in,add_index_post,in_orde
r,post_order,
                in_order_add_num,post_order_add_num);

    //printf("\nnumber = %d\n",number);
    //对每一种可能性进行检验
    for(int i = 0;i<add_index_post;i++){
        for(int j = 0;j<add_index_in;j++){
            //printf("i = %d, j = %d\n",i,j);
            build(0,number-1,post_completed[i],in_completed[j],number-
1,pre_order,0,1);
            if(pre_judge == number){
                in_count = j;
                post_count = i;
                final_flag++;
            }
            pre_judge = 0;
        }
    }
    //如果有且只有一种可能性
    if(final_flag == 1){
        //printf("=================\n");

        //输出中序
        for(int i = 0;i<number;i++){
            if(i == number -1)
                printf("%d",in_completed[in_count][i]);
            else printf("%d ",in_completed[in_count][i]);
        }
        printf("\n");
        //输出前序
        build(0,number-
1,post_completed[post_count],in_completed[in_count],number-1,pre_order,1,1);
        printf("\n");
        //输出后序
        for(int i = 0;i<number;i++){
            if(i == number-1)
                printf("%d",post_completed[post_count][i]);
            else printf("%d ",post_completed[post_count][i]);
        }
        printf("\n");
        //level_output(in_completed[in_count],post_completed[post_count],number-
1,0,number-1);
        for(int i = 0;i<10000;i++){
            if(level_order[i] != 0){
                if(level_print_flag == 1){
                printf("%d",level_order[i]);
                    level_print_flag = 0;
                }
                else
                    printf(" %d",level_order[i]);
            }
        }
    }
    //如果有多种可能性或不可能
    else{
        printf("Impossible");
    }
    system("pause");
}
```

Declaration

I here by declare that all the work done in this project titled "Tree Traversal" is of my independent effort