树的遍历

对于一个二叉树而言有前，中，后序遍历还有层序遍历

前序遍历 先访问结点在进行递归 ， 一般情况是 根—>左节点再 根—>右节点

一般代码是

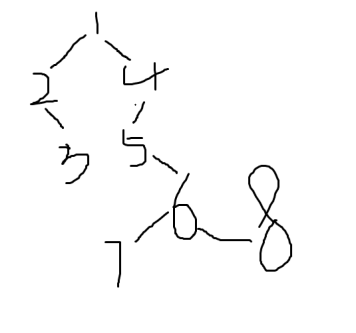
```
if(!T) return ;
printf("%d",T->val);
deep(T->lchild);
deep(T->rchild);
```

中序遍历是 先访问 左节点 再根节点 再右节点

```
if(!T) return ;
deep(T->lchild);
printf("%d",T->val);
deep(T->rchild);
```

后序遍历是 先访问左节点 再右节点 再根节点

```
if(!T) return ;
deep(T->lchild);
deep(T->rchild);
printf("%d",T->val);
```



例如这个图片

前序遍历为 1,2,3,4,5,6,7,8;

> 中序遍历为 2,3,1,5,7,6,8,4

> 后序遍历为 3,2,7,8,6,5,4,1

> 层序遍历的话那么就很明显每层都来遍历

> 从根层层递进　然后从左到右遍历

> 如上个例子

> 1,2,4,3,5,6,7,8;

> 这个可以用bfs实现

> 先访问根节点

> 再遍历根节点的下一层所有元素

好了开始回归正题

想要得到一棵树那么可以根据前序和中序，或者中序和后序，或者中序和层序遍历来

例题acwing 1497树的遍历

一个二叉树，树中每个节点的权值互不相同。

现在给出它的后序遍历和中序遍历，请你输出它的层序遍历。

输入格式

第一行包含整数 N，表示二叉树的节点数。

第二行包含 N 个整数，表示二叉树的后序遍历。

第三行包含 N 个整数，表示二叉树的中序遍历。

输出格式

输出一行 N 个整数，表示二叉树的层序遍历。

数据范围

$1 \leq N \leq 30$,
官方并未给出各节点权值的取值范围，为方便起见，在本网站范围取为 $1 \sim N$。

输入样例：

```
7
2 3 1 5 7 6 4
1 2 3 4 5 6 7
```

输出样例：

```
4 1 6 3 5 7 2
```

开始可以根据后序遍历 找到**根节点**是谁比如在这个例子中 起初根节点为4

再根据 中序遍历的特点 找到 起初根节点的位置那么 就可以 区分开 初始左子树和初始右子树

起初左子树节点为 1 2 3 ， 起初右子树节点为5 6 7

子树的数目是确定好的那么 可以在后序遍历中区分出来左右子树

左：2 3 1 右 5 7 6

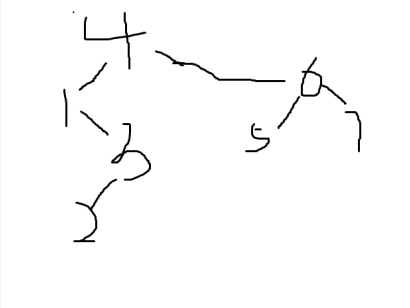
可以将左右子树看成一棵单独的数又继续找到它的左右子树 比如 ***左子树的根为 1***

然后再看中序遍历 1 是 最开始输出的 那么 2 3 节点是存在于它的右子树

再看后序遍历那么可以看到 3 是在2 3 子树中最后输出的

那么 3 为2 3 子树的根 同时 在中序遍历中 2 先于3 输出 那么 2为 3 的左节点

重复上面操作那么 最终得到



由此我们可以找到核心代码

```
// a[]存储的是后序遍历的所有值 b[]是中序遍历的所有值
void dfs(int al,int ar,int bl,int br,int d)
{
    int val = a[ar];//即根节点的值
   int k = p[val];//p数组存储的是中序遍历中各个值的位置
   level[d].push_back(val);//将每层的点存入该层的数组中    d是层号
   dfs(al,al+k-1-bl,bl,k-1,d+1) //子树数量应该相同那么设 后序遍历中左子树最后节点的位置为x
       //那么 x - al == k-1 -bl
    dfs(al+k-bl,ar-1,k+1,br,d+1)
}
```

那么最终代码如下

```
#include<iostream>
#include<vector>

using namespace std;

const int N = 35;
vector<int>level[N];
int a[N],b[N],p[N];
int n;

void build(int al,int ar,int bl,int br,int d)
{
 if(al>ar)return ;
 int val = a[ar];
 int k = p[val];
 level[d].push_back(val);
 build(al,al+k-1-bl,bl,k-1,d+1);
 build(al+k-bl,ar-1,k+1,br,d+1);
}

int main()
{
 cin >> n;
 for(int i = 0; i < n ; i ++) cin >> a[i];
 for(int i = 0; i < n; i ++) cin >> b[i];
 for(int i = 0; i < n; i ++) p[b[i]] = i;

 build(0,n-1,0,n-1,0);
 for(int i = 0; i < n; i ++)//因为最多有n层所以要遍历n个数
   for(auto p : level[i])
     cout << p << ' ';
     return 0;
}
```