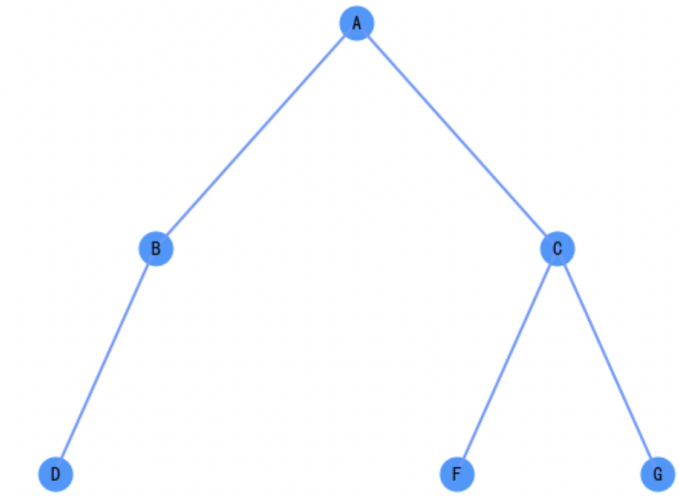
选21、二叉树遍历的代码实现

一、先根遍历的代码实现

有如下用链表方法表示的二叉树：

```
import DrawBinTree as dbt
link_tree=[['A',1,2],['B',3,-1],['C',4,5],['D',-1,-1],['F',-1,-1],['G',-1,-1]]
dbt.draw_link_tree(link_tree)
```



其先根遍历的顺序是：A B D C F G

回顾一下，是怎么遍历的：

从树根节点开始，到左子树（如果有），再到右子树（如果有）

而对于左子树，右子树，也用同样的方法

不妨写一个函数first_root:

```
def first_root(node):#传入node指针
    print(link_tree[node][0])    #打印节点
    if link_tree[node][1]!=-1:    #如果有左子树
        #同样的方法遍历左子树
         first_root(link_tree[node][1])
    if link_tree[node][2]!=-1:    #如果有右子树
        #同样的方法遍历右子树
        first_root(link_tree[node][2])
```

这种方法，函数内部调用自己，把一个大问题分解成小问题，称为递归

如果用res来接受结果，代码可以这样写：

```
res=[]
def first_root(node):
    res.append(link_tree[node][0])
    if link_tree[node][1]!=-1:
        first_root(link_tree[node][1])
    if link_tree[node][2]!=-1:
        first_root(link_tree[node][2])
first_root(0)
print(" ".join(res))
```

二、中根遍历与后根遍历的代码实现

```
res=[]
def mid_root(node):
    if link_tree[node][1]!=-1:
        mid_root(link_tree[node][1])
    res.append(link_tree[node][0])
    if link_tree[node][2]!=-1:
        mid_root(link_tree[node][2])
mid_root(0)
print(" ".join(res))
```

```
D B A F C G
```

```
res=[]
def last_root(node):
    if link_tree[node][1]!=-1:
        last_root(link_tree[node][1])
    if link_tree[node][2]!=-1:
        last_root(link_tree[node][2])
    res.append(link_tree[node][0])
last_root(0)
print(" ".join(res))
```

```
D B F G C A
```

三、如果用列表嵌套方式来表示的二叉树，如何实现遍历?

学生完成

```
list_tree=['A',['B',['F',None,None],['E',None,['D',None,None]]],['C',None,None]]
res=[]
def first_root(tree):
    res.append(tree[0])
    if tree[1]!=None:
        first_root(tree[1])
    if tree[2]!=None:
        first_root(tree[2])
first_root(list_tree)
print(res)
```

```
['A', 'B', 'F', 'E', 'D', 'C']
```

不同的是，链表传入的是指针，而列表嵌套传入的是列表本身