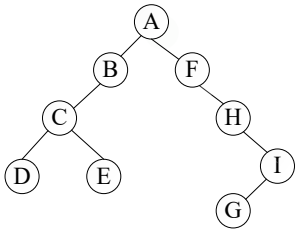
1. **//有条件交换左右子树（仅当左子树非空且左子树根的值小于非空右子树根值时交换，否则不交换）**，建议算法采用先序遍历递归实现交换。

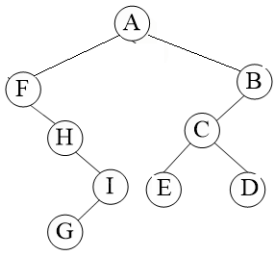
```
#include "stdafx.h"
#include<stdio.h>
#include<malloc.h>
//二叉链表的结构类型定义
const int maxsize=1024;
typedef char datatype;
typedef struct node
{
    datatype data;
    struct node *lchild,*rchild;
}bitree;

bitree*creattree();
void preorder(bitree*);
void swap(bitree*);

int main(void)
{
      bitree*pb;
      pb=creattree();
      printf("******************交换之前的原二叉树的先序遍历序列为：\n");
      preorder(pb);
      printf("\n");
      swap(pb);
      printf("******************交换之后的原二叉树的先序遍历序列为：\n");
      preorder(pb);
      printf("\n");
      return 0;
}
```

测试用例：



假设输入的二叉树如图所示：

交换左右子树后的二叉树如图所示：

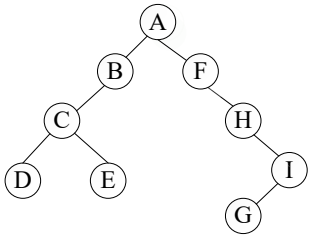



```
**********按层次输入二叉树，虚结点输入'@'，以'#'结束输入********
ABFC@@HDE@@@@@I@@@@@@@@@@@@@@@G#
********************交换之前的原二叉树先序遍历序列为：A  B  C  D  E  F  H  I  G
********************交换之前的原二叉树中序遍历序列为：D  C  E  B  A  F  H  G  I
********************交换之后的二叉树先序遍历序列为：A  F  H  I  G  B  C  E  D
********************交换之后的二叉树中序遍历序列为：F  H  G  I  A  E  C  D  B
```

2. //题目要求：将二叉树中数值为 x 的结点及其子树从二叉树中删除，并以先序遍历的方式输出被删除的子树。（假定二叉树上所有结点均不重复）

```
#include "stdafx.h"
#include<stdio.h>
#include<malloc.h>
//二叉链表的结构类型定义
const int maxsize=1024;
typedef char datatype;
typedef struct node
{
    datatype data;
    struct node *lchild,*rchild;
}bitree;

bitree*creattree();
bitree * preorder-delete(bitree *, datatype);
void preorder (bitree *);

int main(void)
{
      bitree*pa,*pb;
      datatype x;
      printf("***************请构建二叉树：\n");
      pa=creattree();
      printf("***************原二叉树的先序遍历序列为： ");
      preorder(pa);
      printf("\n***************请输入待删除的结点 x：");
      getchar();
      x=getchar();
      pb=preorder-delete (pa,x);
      printf("***************删除子树后的二叉树的先序遍历序列：");
      preorder(pa);
      printf("\n");
      printf("***************被删除子树的先序遍历序列为：");
      preorder(pb);
      printf("\n");
      return 0;
}
```

测试用例：



假设输入的交换之前的树如图所示：

输出:

原二叉树的先序遍历序列为：A，B，C，D，E，F，H，I，G

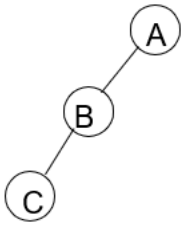
请输入待删除的结点 x：F

删除子树后的二叉树的先序遍历序列为：：A，B,C,D,E

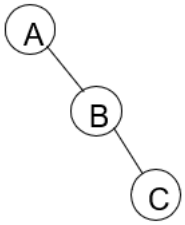
删除的子树的先序遍历序列为：F，H，I，G

3. // 二叉树先序遍历非递归算法实现

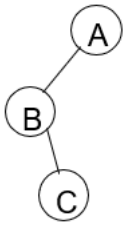
```
#include "stdafx.h"
#include<malloc.h>
#include<stdio.h>
# define maxsize 1024
typedef char datatype;

typedef struct node
{
    datatype data;
    struct node    *lchild, *rchild;
}bitree;
bitree * creattree();
void preorderNo(bitree *);//先序遍历非递归算法

int main()
{
    bitree *root;
    printf("按层次输入二叉树，虚结点输入'@'，以'#'结束输入：\n");
    root = creattree(); //建立二叉树（非递归）
    printf("***********输出后序遍历序列如下***********\n");
    printf("                  ");
    preorderNo(root); //先序遍历二叉树（非递归）
    printf("\n");
    return 0;
}
```



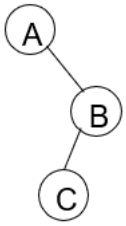
测试用例（1）如上图，输入：AB@C#
输出：ABC



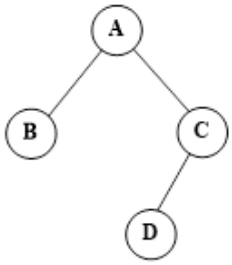
测试用例（2）如上图，输入：A@B@@@C#
输出：ABC



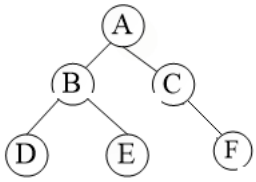
测试用例（3）如上图，输入：AB@@C#
输出：ABC



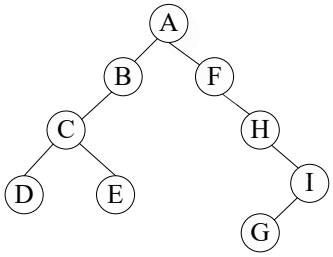
测试用例（4）如上图，输入：A@B@@C#
输出：ABC



测试用例（5）如上图，输入：ABC@@D#
输出：ABCD



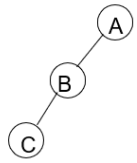
测试用例（6）如上图，输入：ABCDE@F#
输出：ABDECF



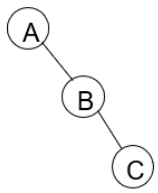
测试用例（7）如上图，输入：ABFC@@HDE@@@@@I@@@@@@@@@@@@@@G#
输出：ABCDEFHIG

4. //二叉树后序遍历非递归算法实现

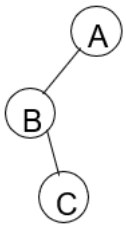
```
#include "stdafx.h"
#include<malloc.h>
#include<stdio.h>
# define maxsize 1024
typedef char datatype;

typedef struct node
{
    datatype data;
    struct node    *lchild, *rchild;
}bitree;
bitree * creattree();
void postorderNo(bitree *);//后序遍历非递归算法

int main()
{
    bitree *root;
    printf("按层次输入二叉树，虚结点输入'@'，以'#'结束输入：\n");
    root = creattree(); //建立二叉树
    printf("**********输出后序遍历序列：");
    postorderNo(root); //后序遍历二叉树（非递归）
    printf("\n");
    return 0;
}
```



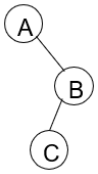
测试用例（1）如上图，输入：AB@C#
输出：CBA



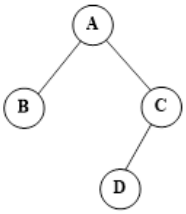
测试用例（2）如上图，输入：A@B@@@C#
输出：CBA



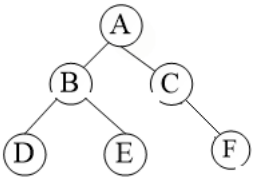
测试用例（3）如上图，输入：AB@@C#
输出：CBA



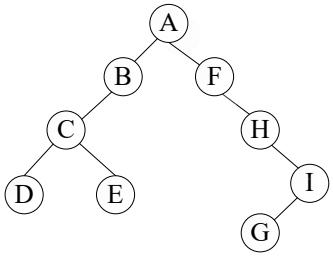
测试用例（4）如上图，输入：A@B@@C#
输出：CBA



测试用例（5）如上图，输入：ABC@@D#
输出：BDCA



测试用例（6）如上图，输入：ABCDE@F#
输出：DEBFCA

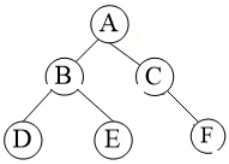


测试用例（7）如上图，输入：ABFC@@HDE@@@@@I@@@@@@@@@@@@@@G#
输出：DECBGIHFA

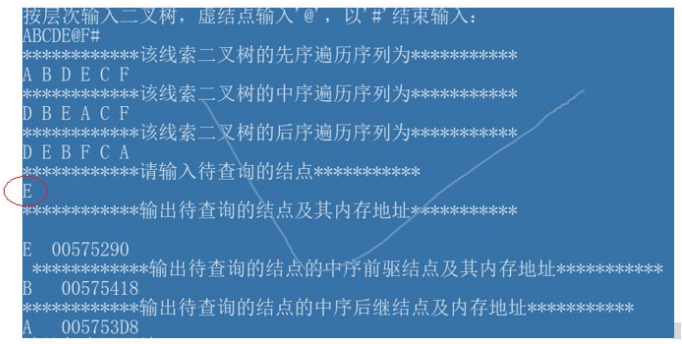
5.//线索二叉树的相关操作：对二叉树完成中序线索化，并输出其先序遍历序列、中序遍历序列、后序遍历序列，输出树中指定结点的中序前驱和后继结点信息。

```
#include "stdafx.h"
#include<malloc.h>
#include<stdio.h>
# define maxsize 1024
typedef char datatype;
typedef  struct  Node
{
    datatype  data;
    struct  Node  * lchild, *rchild;
    int ltag, rtag;
} Bbitree;

Bbitree *pre = NULL;
Bbitree *BCreate();//构建二叉树（递归算法、非递归算法二选一，均可）
void InThreading(Bbitree *p); //建立线索二叉树（基于构建好的二叉树完成中序线索化）
void postorder(Bbitree *);//后序遍历线索二叉树
void preorder(Bbitree*);//先序遍历线索二叉树
void inorder(Bbitree*);//中序遍历线索二叉树
Bbitree * locate(Bbitree*, datatype); //中序线索二叉树上的查找运算
Bbitree * inorderprior(Bbitree *);//已知某结点指针，在中序线索二叉树上查找其前驱结点
Bbitree *inorderpost(Bbitree *);//已知某结点指针，在中序线索二叉树上查找其后继结点

int main()
{
    char ch;
    Bbitree *root;
    Bbitree* q, *y;
    root = BCreate(); //建立二叉树
    InThreading(root); //建立线索二叉树
    printf("**********该线索二叉树的先序遍历序列为**********\n");
    preorder(root);
    printf("\n");
    printf("**********该线索二叉树的中序遍历序列为**********\n");
    inorder(root);
    printf("\n");
    printf("**********该线索二叉树的后序遍历序列为**********\n");
    postorder(root);
    printf("\n");
    printf("**********请输入待查询的结点**********\n");
    getchar();
```

```
    ch = getchar();
    y = locate(root, ch);
    printf("**********输出待查询的结点及其内存地址**********\n");
    printf("\n%c  %p\n ", y->data, y);
    q = inorderprior(y);
    if (q == NULL)
    {
        printf("**********该结点无中序前驱结点**********\n");
        return 0;
    }

    printf("**********输出待查询的结点的中序前驱结点及其内存地址**********\n");
    printf("%c  %p", q->data, q);
    printf("\n");

    q = inorderpost(y);//查找线索二叉树后继
    if (q == NULL)
    {
        printf("**********该结点无中序后继结点**********\n");
        return 0;
    }
    printf("**********输出待查询的结点的中序后继结点及内存地址**********\n");
    printf("%c  %p", q->data, q);
    printf("\n");

    return 0;
}
```



测试用例（6）如上图，输入：ABCDE@F#　　如下图



请多测 2 个二叉树（结构自定）

6.//P192,写出将一个无向图的邻接矩阵转换成邻接表的算法。通过对两种不同存储方法的图，输出其 DFS 序列一致性加以验证转换正确。

7. 二叉树的带权路径长度（WPL）是二叉树中所有叶子结点的带权路径长度之和。给定一棵二叉树 T，采用二叉链表存储，结点结构（lchild,weight,rchild），其中 weight 表示叶子的权值。设 root 为指向 T 的根节点的指针，请设计求 T 的 WPL 的递归算法。

如图 1，图 2，图 3，图 4，图 5，权值分别为 36,46,35,79 和 70。

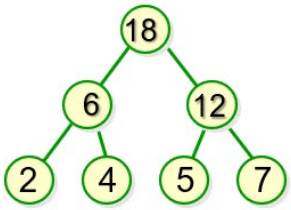


图 1

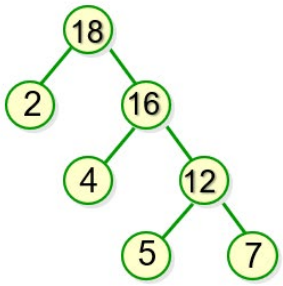


图 2

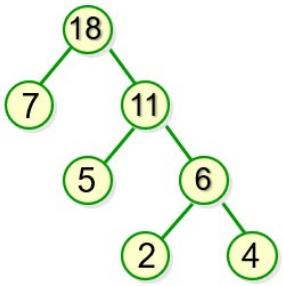


图 3

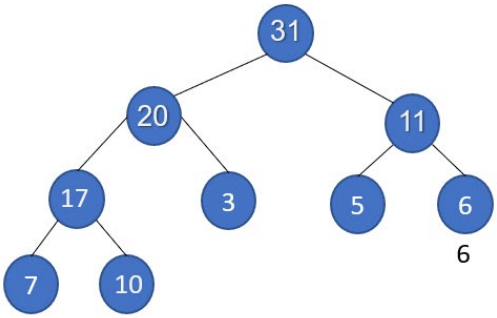


图 4

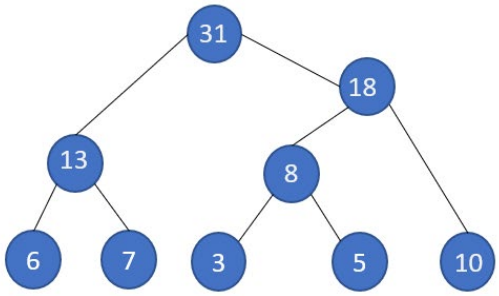


图 5

输出格式要求：

测试用例 1：

输出：该二叉树的先序 DFS 序列为 18 6 2 4 12 5 7
该二叉树的中序 DFS 序列为 2 6 4 18 5 12 7
该二叉树的 WPL 为：36

测试用例 2：

输出：该二叉树的先序 DFS 序列为 18 2 16 4 12 5 7
该二叉树的中序 DFS 序列为 2 18 4 16 5 12 7
该二叉树的 WPL 为：46

测试用例 3：

输出：该二叉树的先序 DFS 序列为 18 7 11 5 6 2 4
该二叉树的中序 DFS 序列为 7 18 5 11 2 6 4
该二叉树的 WPL 为：35

测试用例 4：

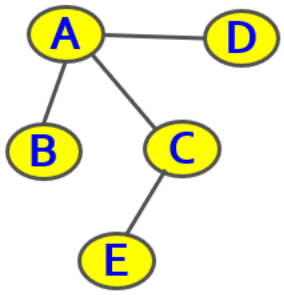
输出：该二叉树的先序 DFS 序列为 31 20 17 7 10 3 11 5 6
该二叉树的中序 DFS 序列为 7 17 10 20 3 31 5 11 6
该二叉树的 WPL 为：79

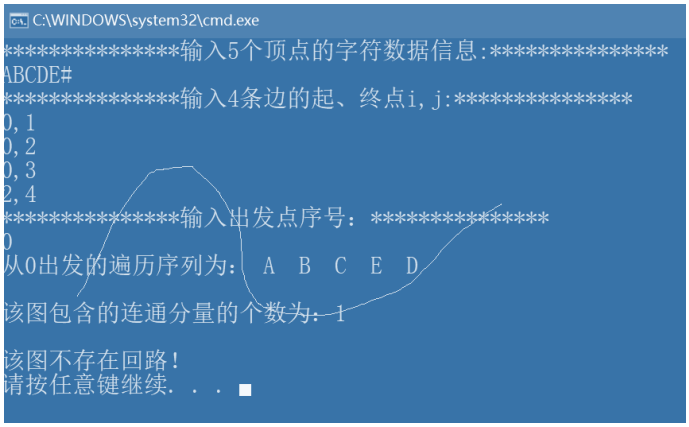
测试用例 5：

输出：该二叉树的先序 DFS 序列为 31 13 6 7 18 8 3 5 10
该二叉树的中序 DFS 序列为 6 13 7 31 3 8 5 18 10
该二叉树的 WPL 为：70

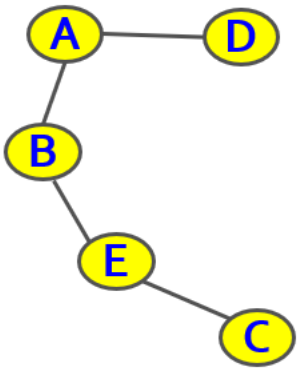
8. //输出一个无向图的 DFS 序列,并写出判断该图是否存在回路(环)的算法 （提示：可以通过统计连通分量个数 m，m+e>n 则判定有环路，否则无环）

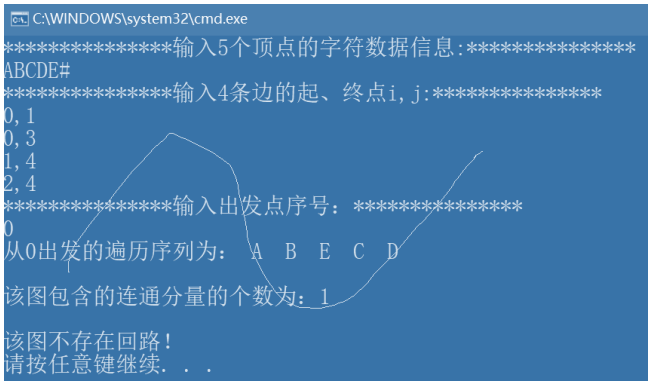
测试用例 1：如图：



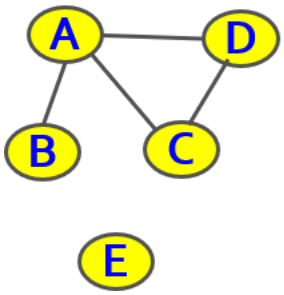


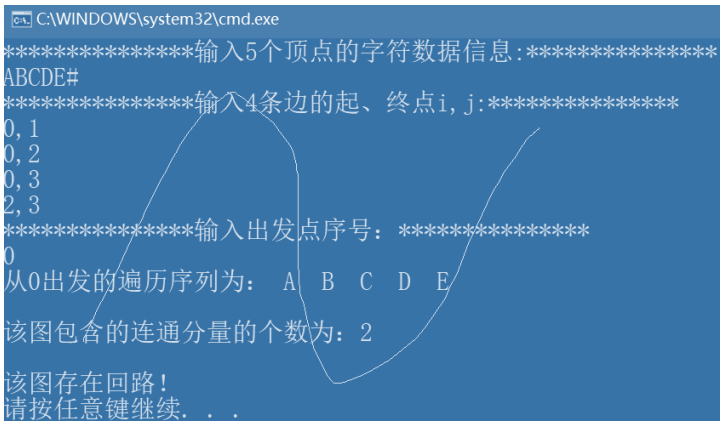
测试用例 2：如图：



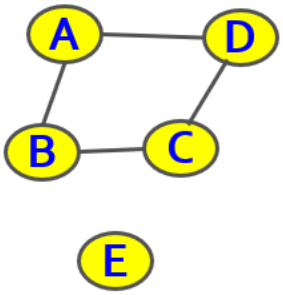


测试用例 3：如图：





测试用例 4：如图：



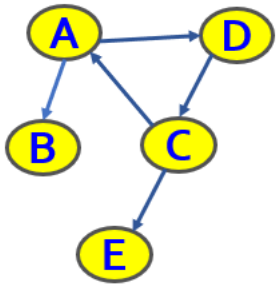

```
C:\WINDOWS\system32\cmd.exe
**********输入5个顶点的字符数据信息:**********
ABCDE#
**********输入4条边的起、终点i,j:**********
0,1
0,3
1,2
2,3
**********输入出发点序号：**********
0
从0出发的遍历序列为：  A  B  C  D  E

该图包含的连通分量的个数为：2

该图存在回路！
请按任意键继续. . .
```

或者，也可以设计成类似输出，允许自由输入顶点数和边数生成任意图结构。(不强制要求)

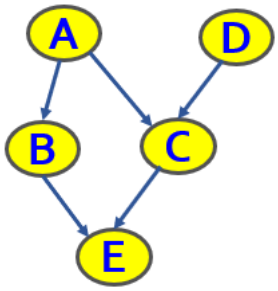
```
C:\WINDOWS\system32\cmd.exe
**********请输入图的顶点数目n和边数e,输入-1,-1结束：**********
3,3
**********输入3个顶点的字符数据信息:**********
ABC#
**********输入3条边的起、终点i,j:**********
0,1
0,2
1,2
**********输入出发点序号：**********
0
从0出发的遍历序列为：  A  B  C

该图包含的连通分量的个数为：1

该图存在回路！
**********请输入图的顶点数目n和边数e,输入-1,-1结束：**********
5,4
**********输入5个顶点的字符数据信息:**********
ABCDE#
**********输入4条边的起、终点i,j:**********
0,1
0,2
1,2
3,4
**********输入出发点序号：**********
0
从0出发的遍历序列为：  A  B  C  D  E

该图包含的连通分量的个数为：2

该图存在回路！
**********请输入图的顶点数目n和边数e,输入-1,-1结束：**********
-1,-1
请按任意键继续. . .
```

9.//输出一个有向图的 DFS 序列,并写出判断该图是否存在回路(环)的算法 （基于 DFS 算法写）

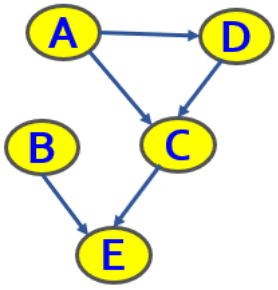
测试用例 1：



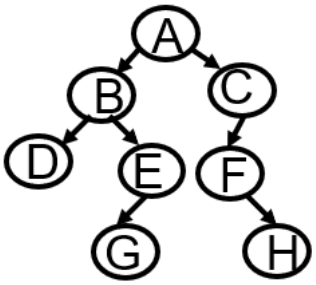
测试用例 2：



测试用例 3：



10. 以中序遍历为基础,写出在二叉树上查找指定结点 x 的中序后继结点的算法。要求空间复杂度为树的深度。 （注：不借助数组，第三次上机如果已经做到空间复杂度要求，此题忽略）



例如二叉树如图所示：

测试用例 1：
输入：待查找结点为 A
输出：结点 A 的中序后继结点为 F

测试用例 2：
输入：待查找结点为 B
输出：结点 B 的中序后继结点为 G

测试用例 3：
输入：待查找结点为 G
输出：结点 G 的中序后继结点为 E

测试用例 4：
输入：待查找结点为 E
输出：结点 E 的中序后继结点为 A

测试用例 5：
输入：待查找结点为 C
输出：结点 C 没有中序后继结点

其它可选测试用例：

输入：待查找结点为 D
输出：结点 D 的中序后继结点为 B

输入：待查找结点为 F
输出：结点 F 的中序后继结点为 H

输入：待查找结点为 H
输出：结点 H 的中序后继结点为 C