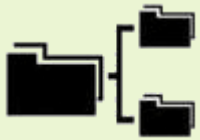
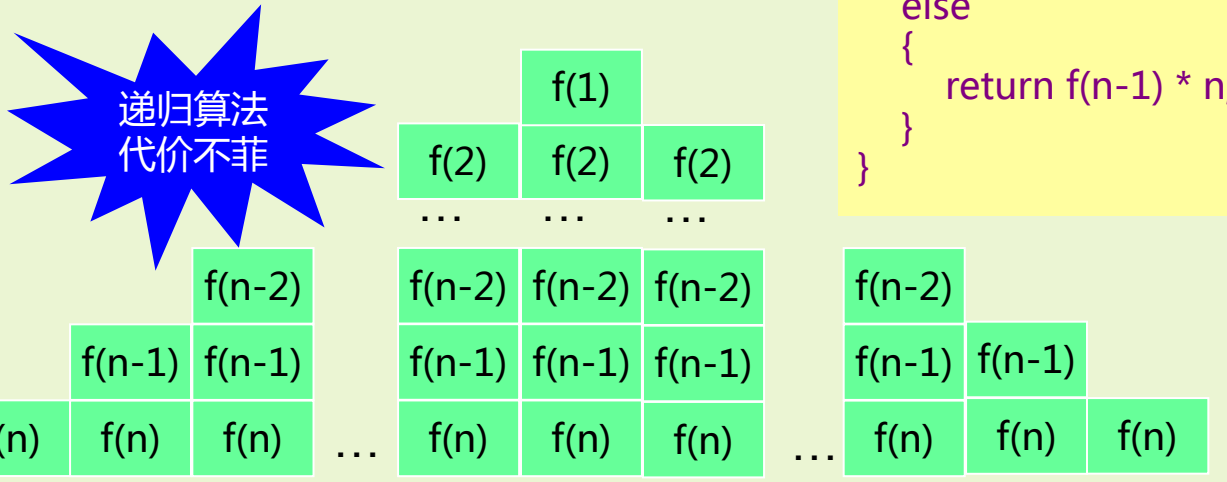


本节主题：

二叉树遍历的非递归算法

二叉树的遍历

- 三种遍历
  - 先序遍历：根节点-->左子树-->右子树。
  - 中序遍历：左子树-->根节点-->右子树。
  - 后序遍历：左子树-->右子树-->根节点
- 两类算法
  - 递归算法
    - 直观，易读
    - 效率低下
  - 非递归算法

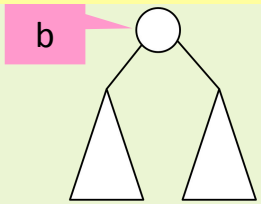



```
//求n!=n*(n-1)*...*2*1
int f(n)
{
    if (n == 1)
    {
        return 1;
    }
    else
    {
        return f(n-1) * n;
    }
}
```

先序遍历非递归算法1

- 算法步骤

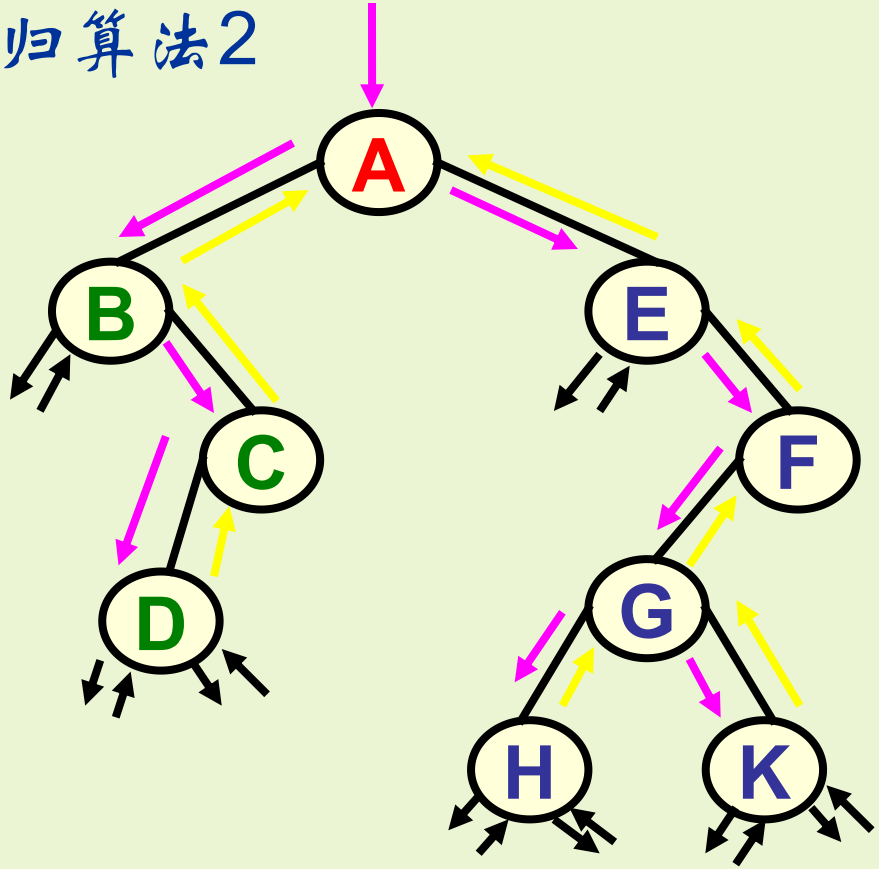
```
if (当前b树不空)
{
   根节点b进栈;
   while (栈不空)
   {
      出栈节点p并访问之;
      若*p节点有右孩子，将其右孩子进栈；
      若*p节点有左孩子，将其左孩子进栈；
   }
}
```

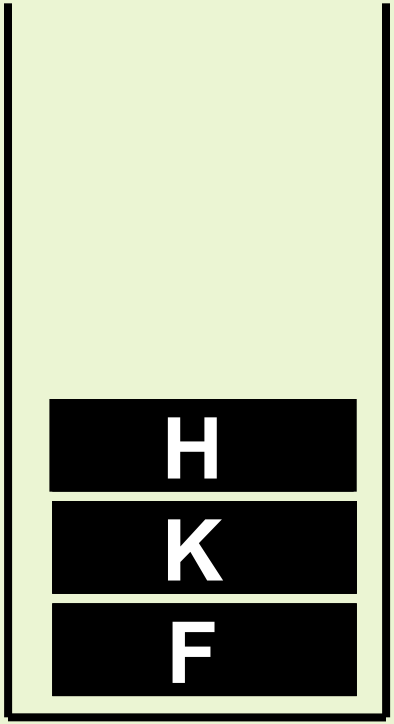



```
void PreOrder1(BTNode *b)
{
   BTNode *St[MaxSize],*p;
   int top=-1;
   top++;
   St[top]=b; //根节点入栈
   while (top>-1)              //栈不为空时循环
   {
      p=St[top];               //退栈并访问该节点
      top--;
      printf("%c ",p->data);
      if (p->rchild!=NULL)  //右孩子节点入栈
      {
         top++;
         St[top]=p->rchild;
      }
      if (p->lchild!=NULL)  //左孩子节点入栈
      {
         top++;
         St[top]=p->lchild;
      }
   }
}
```

先序遍历非递归算法2

```
if (当前b树不空)
{
   p = b;
   while (栈不空或者p!=NULL)
   {
      while (p有左孩子)
      {
         访问p所指节点;
         将p进栈;
         p=p->lchild;
      }
      if (栈不空)
      {
         出栈p;
         p = p->rchild;
      }
   }
}
```





访问：

先序遍历非递归算法2的实现

```
if (当前b树不空)
{
   p = b;
   while (栈不空或者p!=NULL)
   {
      while (p有左孩子)
      {
         访问p所指节点;
         将p进栈;
         p=p->lchild;
      }
      if (栈不空)
      {
         出栈p;
         p = p->rchild;
      }
   }
}
```

```
void PreOrder2(BTNode *b)
{
   BTNode *St[MaxSize],*p;
   int top=-1;
   p=b;
   while (top>-1 || p!=NULL)
   {
      while (p!=NULL)
      {
            //扫描*p的所有左节点并进栈
      }
      if (top>-1)
      {
            //处理右子树
      }
   }
}
```

```
printf("%c ",p->data);
top++;
St[top]=p;
p=p->lchild;
```

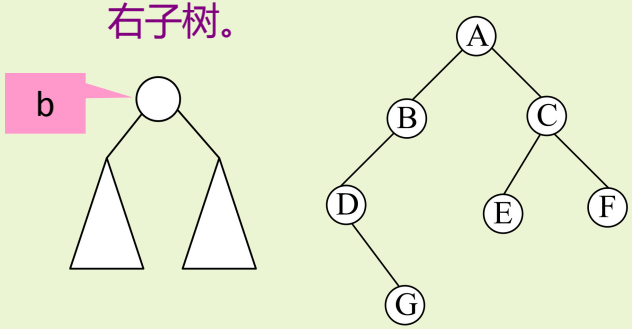
```
p=St[top];
top--;
p=p->rchild;
```

中序遍历(左-根-右)非递归算法

（1）所有左下孩子进栈，体现先访问左子树的特点。

（2）当所有左下孩子进栈后，栈顶节点p没有左孩子（即没有左子树）或者其左子树均已访问，所以可以**访问**p节点。

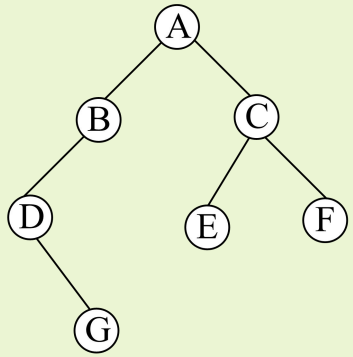
（3）当访问p节点后，转向其右孩子，采用同样的方式中序遍历右子树。




```
if (当前b树不空)
{
   p = b;
   while (栈不空或者p!=NULL)
   {
      while (p有左孩子)
      {
         将p进栈;
         p=p->lchild;
      }
      if (栈不空)
      {
         出栈p并访问之;
         p = p->rchild;
      }
   }
}
```

```
void InOrder1(BTNode *b)
{
   BTNode *St[MaxSize],*p;
   int top=-1;
   p=b;
   while (top>-1 || p!=NULL)
   {
      while (p!=NULL)
      {
         top++;
         St[top]=p;
         p=p->lchild;
      }
      if (top>-1)
      {
         p=St[top];
         top--;
         printf("%c ",p->data);
         p=p->rchild;
      }
   }
}
```

后序遍历(左-右-根)非递归算法




```
if (当前b树不空)
{
   do
   {
      while (b!=NULL, b有左孩子,将进栈);
      出栈节点b;
      if (b的右子树已访问)
         则访问b并退栈;
      else
         b = b->rchild;
   }
   while (栈不空);
}
```

难点

- 如何判断一个节点*b的右孩子节点已访问过?

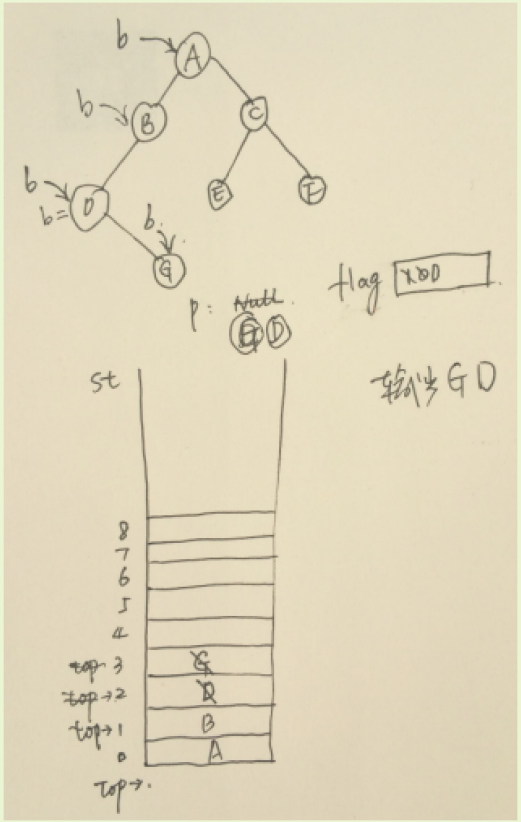
条件

- 在后序遍历中，*b的右孩子节点一定刚好在*b之前访问。

方法

- 用p保存刚刚访问过的节点（初值为NULL），
- 若b->rchild==p成立，说明*b的左右子树均已访问，现在应访问*b。

算法实现



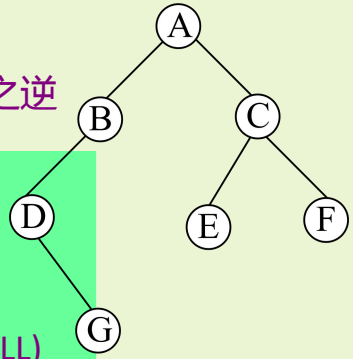

```
void PostOrder1(BTNode *b)
{
   BTNode *St[MaxSize];
   BTNode *p;
   int flag,top=-1;
   if(b!=NULL)
     do
     {
        //将*b的所有左节点进栈
        p=NULL;
        flag=1;   //表示*b的左子树已访问或为空
        while (top!=-1 && flag==1)
        {
           b=St[top];
           //处理*b节点
        }
     }
     while (top!=-1);
}
```

```
while (b!=NULL)
 {
     top++;
     St[top]=b;
     b=b->lchild;
 }
```

```
if (b->rchild==p)
{
   printf("%c ",b->data);
   top--;
   p=b;
}
else
{
   b=b->rchild;
   flag=0;
}
```

例：路径之逆

- 问题：二叉树采用二叉链存储结构，设计算法输出从根节点到每个叶子节点的路径之逆
- 解：采用后序遍历非递归算法




```
void AllPath1(BTNode *b)
{
   BTNode *St[MaxSize];
   BTNode *p;
   int flag,i,top=-1;
   if (b!=NULL)
   {
      do
      {
         //将*b的所有左节点进栈
         p=NULL;
         flag=1;
         //当栈非空处理每一个叶子节点
      }
      while (top!=-1);
      printf("\n");
   }
}
```

```
while (b!=NULL)
{
    top++;
    St[top]=b;
    b=b->lchild;
}
```

```
while (top!=-1 && flag)
{
   b=St[top];
   if (b->rchild==p)
   {
      if (b->lchild==NULL && b->rchild==NULL)
      {
         //若叶子,输出栈中所有节点值
         for (i=top; i>0; i--)
            printf("%c->",St[i]->data);
         printf("%c\n",St[0]->data);
      }
      top--;
      p=b;
   }
   else
   {
      b=b->rchild;
      flag=0;
   }
}
```