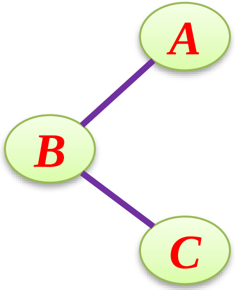
7.7 二叉树的构造

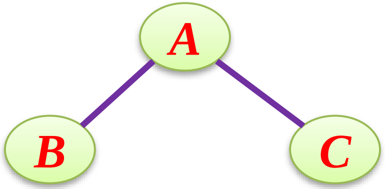
回顾

同一棵二叉树（假设每个节点值唯一）具有唯一先序序列、中序序列和后序序列。

但不同的二叉树可能具有相同的先序序列、中序序列或后序序列。

例如





先序序列均为*ABC*

思考题

给定一棵二叉树（假设每个节点值唯一）的先序、中序和后序序列可以唯一构造（确定）出该二叉树。

仅由先序、中序或后序序列中的一种，无法唯一构造出该二叉树。

那么，给定先序、中序和后序序列中任意两个，是否可以唯一构造出该二叉树?

以下命题成立否？

同时给定一棵二叉树的先序序列和中序序列就能唯一确定这棵二叉树。 ✓

同时给定一棵二叉树的中序序列和后序序列就能唯一确定这棵二叉树。 ✓

同时给定一棵二叉树的先序序列和后序序列就能唯一确定这棵二叉树。 ✗

定理7.1： 任何n（$n>0$）个不同节点的二叉树，都可由它的中序序列和先序序列唯一地确定。

通过根节点a_0在中序序列中找到b_k

先序序列：$a_0\ \underbrace{a_1\ \cdots\ a_k}_{\text{左子树先序序列，有}k\text{个节点}}\ \underbrace{a_{k+1}\cdots a_{n-1}}_{\text{右子树先序序列，有}n-k-1\text{个节点}}$

中序序列：$\underbrace{b_0\ b_1\ \cdots b_{k-1}}_{\text{左子树中序序列，有}k\text{个节点}}\ b_k\ \underbrace{b_{k+1}\cdots b_{n-1}}_{\text{右子树中序序列，有}n-k-1\text{个节点}}$

a_0

左子树

先序：$a_1\ \ldots\ a_k$

中序：$b_0\ \ldots b_{k-1}$

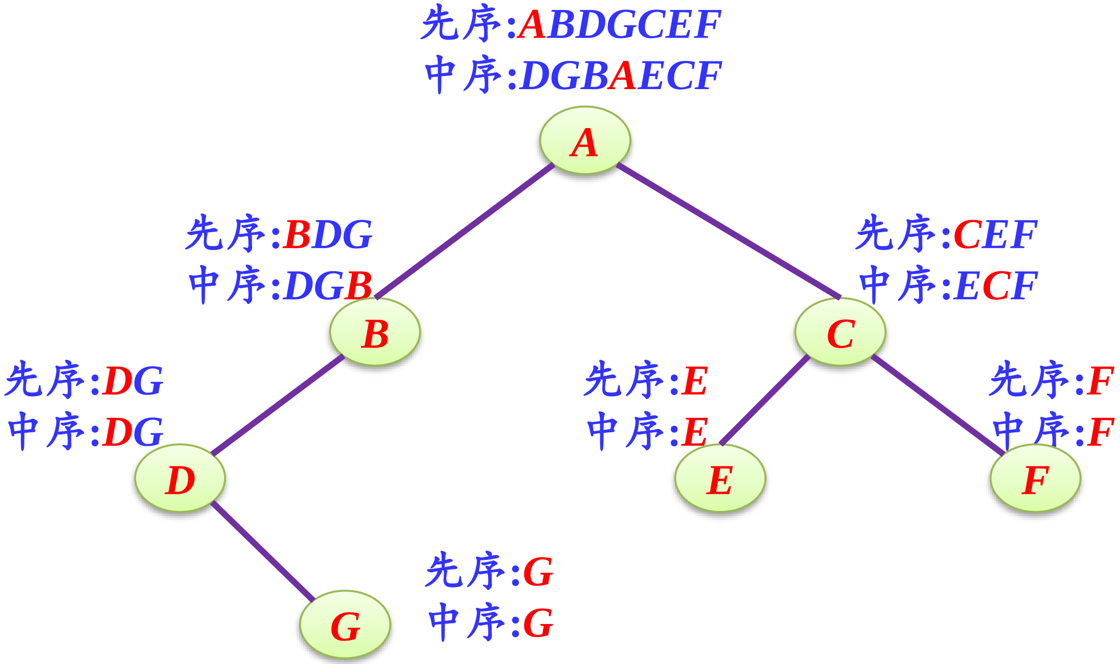
右子树

先序：$a_{k+1}\ldots a_{n-1}$

中序：$b_{k+1}\ldots b_{n-1}$

由先序和中序序列构造二叉树示例的演示

例如，已知先序序列为*ABDGCEF*，中序序列为*DGBAECF*，则构造二叉树的过程如下所示。



二叉树构造完毕

由上述定理得到以下构造二叉树的算法：

```
BTNode *CreateBT1(char *pre， char *in， int n)
{      BTNode *s;  char *p;  int k;
       if (n<=0) return NULL;
       s=(BTNode *)malloc(sizeof(BTNode));     //创建根节点
       s->data=*pre;
       for (p=in;p<in+n;p++)                           //在in中找为*pre的位置k
              if (*p==*pre)
                   break;
       k=p-in;
```

先序遍历的思路

pre+1 　　　　pre+k+1

先序pre： a_0 a_1 … a_{k-1} a_k a_{k+1} … a_{n-2} a_{n-1}

中序in： b_0 b_1 … b_{k-1} b_k b_{k+1} … b_{n-1}

in 　　　　p 　p+1

左子树：

先序序列为pre+1开始的k个字符

中序序列为in开始的k个字符

右子树：

先序序列为pre+k+1开始的$n-k-1$个字符

中序序列为p+1开始的$n-k-1$个字符

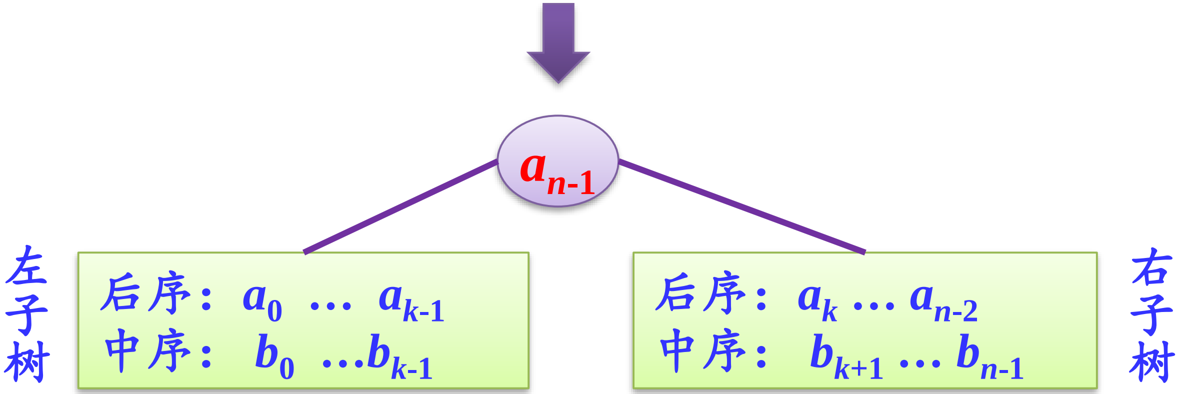
```
    s->lchild=CreateBT1(pre+1， in， k);          //构造左子树
    s->rchild=CreateBT1(pre+k+1， p+1， n-k-1); //构造右子树
    return s;
}
```

定理7.2： 任何n（$n>0$）个不同节点的二叉树，都可由它的中序序列和后序序列唯一地确定。

通过根节点a_{n-1}在中序序列中找到b_k

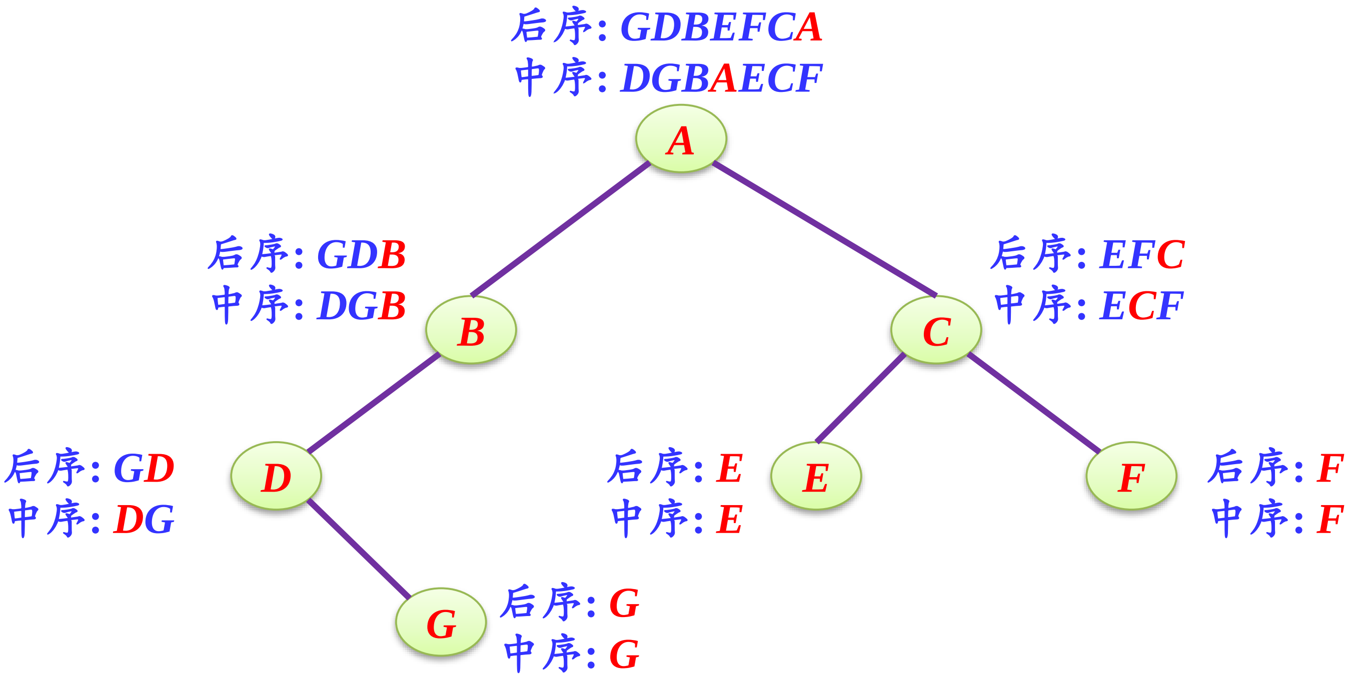
后序序列：$\underbrace{a_0\ a_1\ \cdots\ a_{k-1}}_{\text{左子树后序序列，有}k\text{个节点}}\ \underbrace{a_k\ \cdots\ a_{n-2}}_{\text{右子树后序序列，有}n-k-1\text{个节点}}\ a_{n-1}$

中序序列：$\underbrace{b_0\ b_1\ \cdots b_{k-1}}_{\text{左子树中序序列，有}k\text{个节点}}\ b_k\ \underbrace{b_{k+1}\ \cdots\ b_{n-1}}_{\text{右子树中序序列，有}n-k-1\text{个节点}}$



由后序和中序序列构造二叉树示例的演示

例如，已知中序序列为*DGBAECF*，后序序列为*GDBEFCA*。对应的构造二叉树的过程如下所示。



二叉树构造完毕

由上述定理得到以下构造二叉树的算法：

```
BTNode *CreateBT2(char *post,char *in,int n)
{     BTNode *b;  char r,*p;  int k;
      if (n<=0) return NULL;
      r=*(post+n-1);                               //根节点值
      b=(BTNode *)malloc(sizeof(BTNode));          //创建二叉树节点*b
      b->data=r;
      for (p=in;p<in+n;p++)                        //在in中查找根节点
            if (*p==r)  break;
      k=p-in;                                      //k为根节点在in中的下标
```

post　　　　post+k

后序post: $a_0\ a_1\ \dots\ a_{k-1}\ a_k\ a_{k+1}\dots\ a_{n-2}\ a_{n-1}$

中序in: $b_0\ b_1\ \cdots\ b_{k-1}\ b_k\ b_{k+1}\ \cdots\ b_{n-1}$

in　　　　p　　p+1

左子树:

　　后序序列为post开始的k个字符

　　中序序列为in开始的k个字符

右子树:

　　后序序列为post+k开始的n−k−1个字符

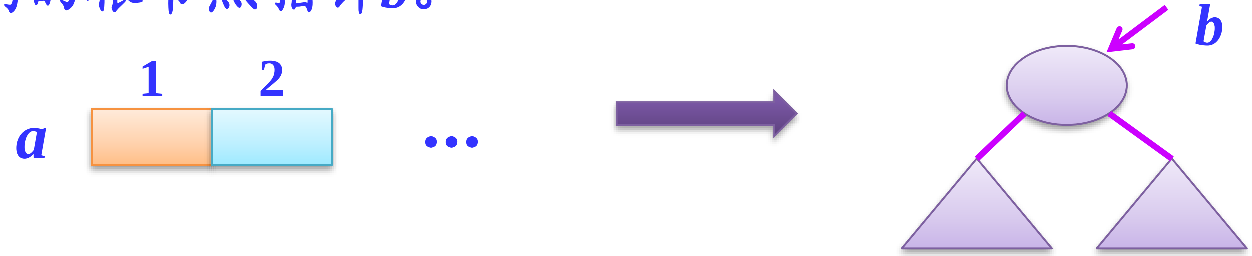
　　中序序列为p+1开始的n−k−1个字符

```
    b->lchild=CreateBT2(post,in,k);            //递归构造左子树
    b->rchild=CreateBT2(post+k,p+1,n-k-1);   //递归构造右子树
    return b;
}
```

先序遍历的思路

【例7-11】 设计一个算法将二叉树的顺序存储结构转换成二叉链存储结构。

解： 设二叉树的顺序存储结构为a，由$f(a,1)$返回创建的二叉链存储结构的根节点指针b。



递归模型：

$f(a,i)$ = NULL　　当i大于MaxSize

$f(a,i)$ = NULL　　当i对应的节点为空

$f(a,i) = b$（创建根节点*b，其data值为$a[i]$）；　　其他情况

　　b->lchild=$f(a,2*i)$;

　　b->rchild=$f(a,2*i+1)$）

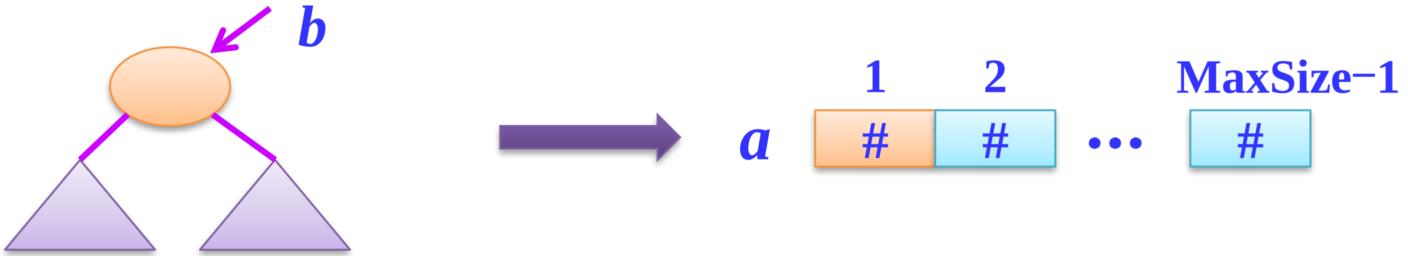
对应的递归算法如下：

```
BTNode *trans1(SqBTree a， int i)
{     BTNode *b;
      if (i>MaxSize)  return NULL;
      if (a[i]=='#')  return NULL;        //当节点不存在时返回NULL
      b=(BTNode *)malloc(sizeof(BTNode));  //创建根节点
      b->data=a[i];
      b->lchild=trans1(a， 2*i);           //递归创建左子树
      b->rchild=trans1(a， 2*i+1);         //递归创建右子树
      return(b);                           //返回根节点
}
```

先序遍历的思路

【例7-12】设计一个算法将二叉树的二叉链转换成顺序存储结构。

解： $f(b,a,i)$：由二叉链b创建$a[i]$为根节点的顺序存储结构a。



- 初始调用：$f(b,a,1)$
- 调用前a的所有元素为#

递归模型：

$f(b,a,i)$ ≡ 不做任何事情　　当b=NULL

$f(b,a,i)$ ≡ $a[i]$=b->data（创建根节点）；　　其他情况
　　$f(b$->lchild,a,2*i);
　　$f(b$->rchild,a,2*i+1)）

对应的递归算法如下：

```
void *trans2(BTNode *b,SqBTree a， int i)
{
      if (b!=NULL)
      {     a[i]=b->data;                          //创建根节点
            trans2(b->lchild,a,2*i);               //递归创建左子树
            trans2(b->rchild,a,2*i+1);             //递归创建右子树
      }
}
```

先序遍历的思路

——本讲完——