第六章 二叉树

part 3: 树与森林

张史梁
slzhang.jdl@pku.edu.cn

内容提要

- 二叉树基础
  - 树与二叉树的基本概念
  - 二叉树的存储结构
  - 二叉树的周游算法
  - 建立一个二叉树
- 二叉树的应用
  - 哈夫曼树
  - 二叉检索树/排序树
- 树与树林

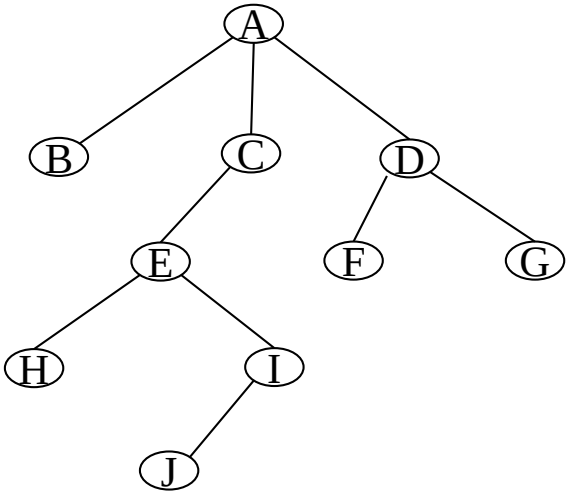
树与树林

- 树与树林的定义
- 树的基本运算
- 树和树林的存储表示
- 树和树林的周游
- 树和二叉树的转换

树的定义

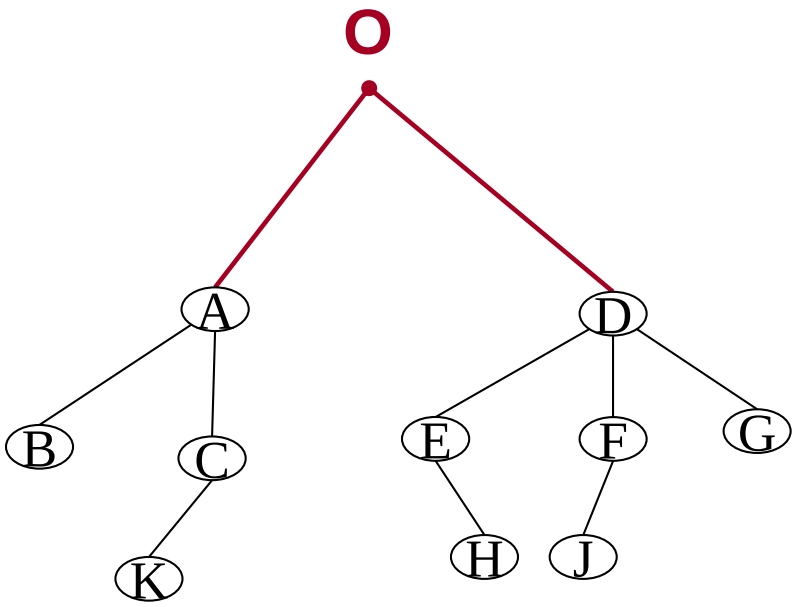
- 树的递归定义：n(n≥0)个结点的有穷集合T，当T非空时：
  - 有且仅有一个特别标出的称为根的结点
  - 除根外，其余结点分为m≥0个互不相交的非空集合$T_1,T_2,\ldots,T_m$，而且每个非空集合T_i又是一颗树，称为根结点的子树。

  T_1={B}，T_2={C，E，H，I，J}，T_3={D，F，G}
- 特别地，允许不包括任何结点的树，把它称作空树。



树林的定义

- “树林”是由0个到多个不相交的树所组成的集合
- 树林中每棵树的根彼此称为“兄弟”
- 与自然界的树林有所不同，这里的树林可以是一个空集
- 如果将树林中的两棵树的根结点连接到一个父节点，便得到一个树



树的基本运算1

Tree t; Node p;

1. Tree creatTree (Tree t)

   创建一棵空树；

2. int isNULL(Tree t)

   判断某棵树是否为空；

3. Node root(Tree t)

   求树中的根结点，若为空树，则返回一特殊值；

4. Node parent(Tree t ,Node P)

   求树中某个指定结点p的父结点，当指定结点为根时，返回一特殊值；

树的基本运算2

5. Node leftChild (Tree t ,Node p)

   求树中某个指定结点p的最左子结点，当指定结点为树叶时，它没有子女，则返回一特殊值；

6. Node rightSibling(Tree t ,Node P)

   求树中某个指定结点p的右兄弟结点，当指定结点没有右兄弟时，返回一特殊值；

7. 树的周游，即按某种方式访问树中的所有结点，并使每个结点恰好被访问一次

树和树林的存储表示

- 选择存储表示方法原则：结点本身+结点之间的关系

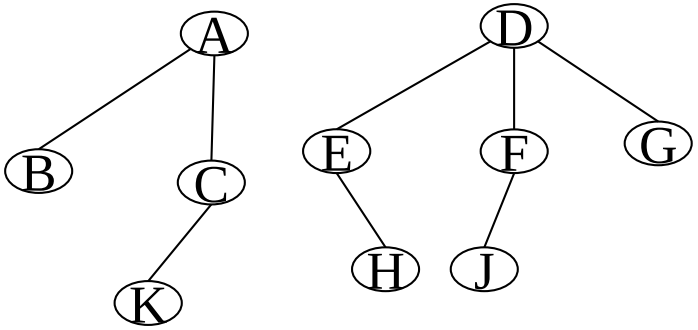
- 树的存储表示
  - 父指针表示法
  - 子表表示法
  - 长子-兄弟表示法（常用）

树和树林的存储表示1

	顺序存储	链式存储
父指针表示法	数组上每个结点存父亲的下标	子结点指向父节点（所有结点的范围？）

树和树林的存储表示1

- 父指针表示法
  - 如果两个结点到达同一根结点，它们一定在同一棵树中。如果找到的根结点是不同的，那么两个结点就不在同一棵树中。
  - 优点
    - 容易找到父结点及其所有的祖先
    - 比较节省存储空间
  - 缺点
    - 没有表示出结点之间的左右次序
    - 找结点的子女和兄弟比较费事（遍查整个数组）



	info	parent
0	A	-1
1	B	0
2	C	0
3	K	2
4	D	-1
5	E	4
6	H	5
7	F	4
8	J	7
9	G	4

父指针表示法

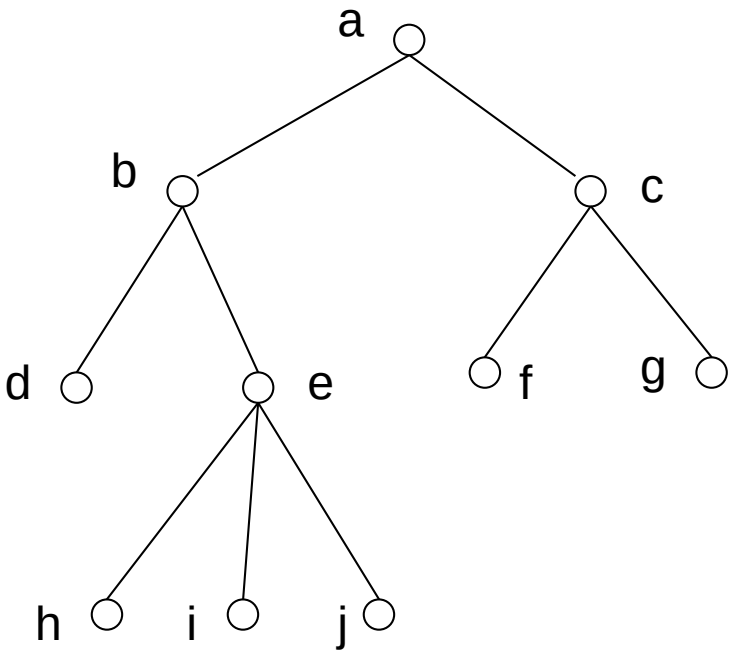
- 整棵树组织成一个结点顺序表，其中每一结点包含父节点的下标

```
struct  ParTreeNode   /* 结点的结构 */
{
   DataType info;
   int  parent;
};
struct ParTree   /* 树的定义*/
{
   int MAXNUM;
  struct ParTreeNode  *nodelist ;  // struct ParTreeNode  nodelist[MAXNUM] ;
  int n;
};
typedef struct  ParTree *  PParTree;
```

父指针表示法的改进

按一种周游次序在数组中存放结点

常见的一种方法是依次存放树的先根序列。

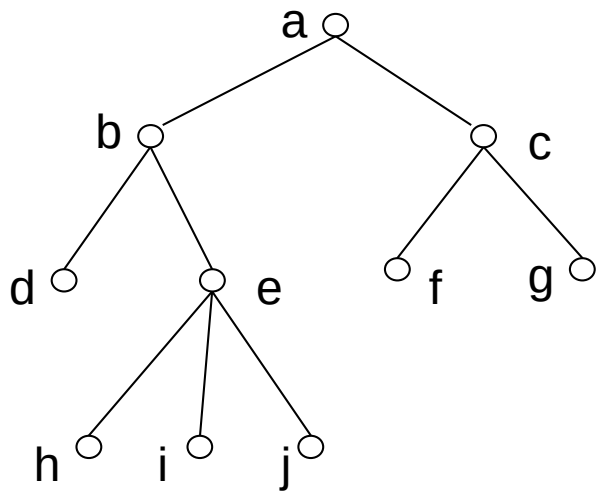


	info	parent
0	a	-1
1	b	0
2	d	1
3	e	1
4	h	3
5	i	3
6	j	3
7	c	0
8	f	7
9	g	7

算法示例

在父指针表示的树中求右兄弟结点的位置

```
int rightSibling_partree(PParTree  t,  int  p)
{
      int i;
      if (p >= 0 && p < t->n) {
            for ( i=p+1; i<t->n; i++)
                  if (t->nodelist[i].parent == t->nodelist[p].parent)
                        return(i);
      }
      return(-1);
}
```



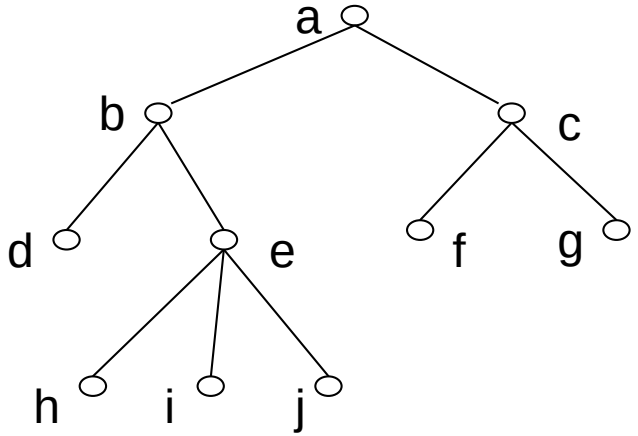
	info	parent
0	a	-1
1	b	0
2	d	1
3	e	1
4	h	3
5	i	3
6	j	3
7	c	0
8	f	7
9	g	7

树和树林的存储表示2

	顺序存储	链式存储
子表表示法	数组上每个结点存其所有子结点的下标	每个结点指向存储其子结点的链表（所有结点的存顺序表）

树和树林的存储表示2

- 子表表示法
  - 整棵树组织成一个结点顺序表，其中每一结点包含一个子表，存放该结点的所有子结点，子表用链接表示。



Nodelist

	info	children
0	a	*
1	b	*
2	d	^
3	e	*
4	h	^
5	i	^
6	j	^
7	c	*
8	f	^
9	g	^

Nodeposition Link

1 * → 7 ^
2 * → 3 ^
4 * → 5 * → 6 ^
8 * → 9 ^

子表表示法

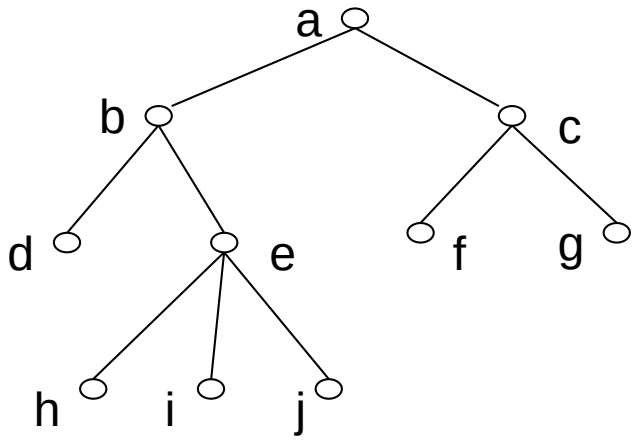
//整棵树组织成一个结点顺序表，每结点包含一个子表，存放其所有子结点

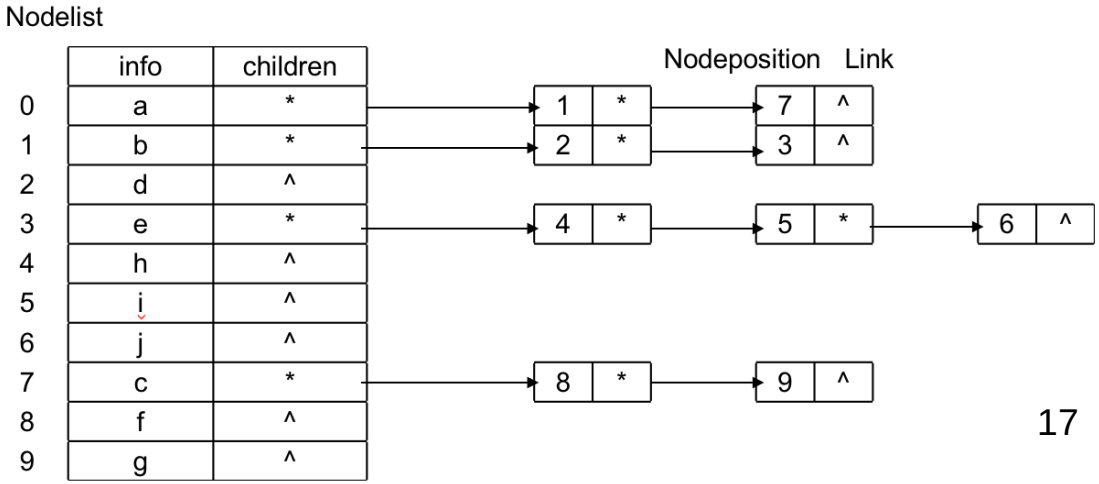
```
① struct EdgeNode  {        /* 子表中节点的结构 */
②         int          nodeposition;
③         struct  EdgeNode *link;
④ };
⑤ struct ChiTreeNode {      /* 结点表中节点的结构 */
⑥         DataType             info;
⑦         struct EdgeNode *children;
⑧ };

⑨ struct ChiTree   {     /*树结构定义*/
⑩       int             root;
⑪         int           n;          /* 结点的个数 */
⑫       struct  ChiTreeNode nodelist [MAXNUM];
⑬ }
⑭ typedef   struct  ChiTree  * PChiTree;
```

算法：在子表表示法求右兄弟的位置

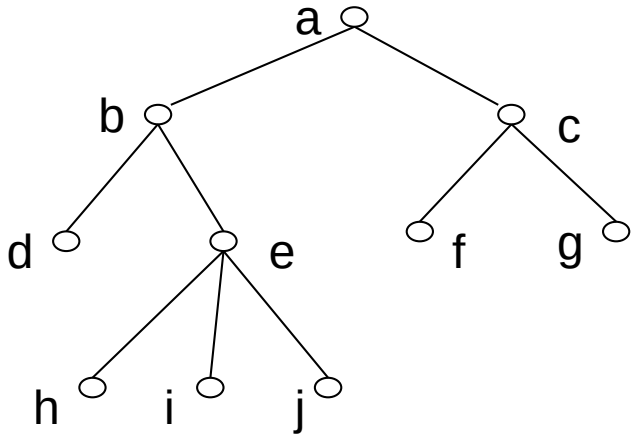
```
① int rightSibling_chitree(PChiTree t, int p)
② {
③       int i;     struct EdgeNode  *v;
④       for (i=0; i<t->n; i++)
⑤         {
⑥             v = t->nodelist[i].children;
⑦              while (v != NULL)
⑧                if (v->nodeposition == p)
⑨                        if (v->link == NULL)     return(-1);
⑩                        else     return(v->link->nodeposition);
⑪                else  v = v->link;
⑫           }
⑬          return(-1);
⑭ }
```





算法：在子表表示上求父结点的位置

```
① int parent_chitree(PChiTree  t,  int p)
② {
③    int i;   struct EdgeNode  *v;
④    /* 逐个检查树的各个结点，是不是父结点 */
⑤    for (i=0; i<t->n ;i++)
⑥    {
⑦       v = t->nodelist[i].children;
⑧       while (v != NULL)
⑨            if (v->nodeposition == p)  return(i);
⑩            else   v = v->link;
⑪    }
⑫    return(-1);              /* 无父结点，则返回值为-1 */
⑬ }
```



Nodelist

	info	children
0	a	*
1	b	*
2	d	^
3	e	*
4	h	^
5	i	^
6	j	^
7	c	*
8	f	^
9	g	^

Nodeposition Link

1 * → 7 ^

2 * → 3 ^

4 * → 5 * → 6 ^

8 * → 9 ^

子表表示法

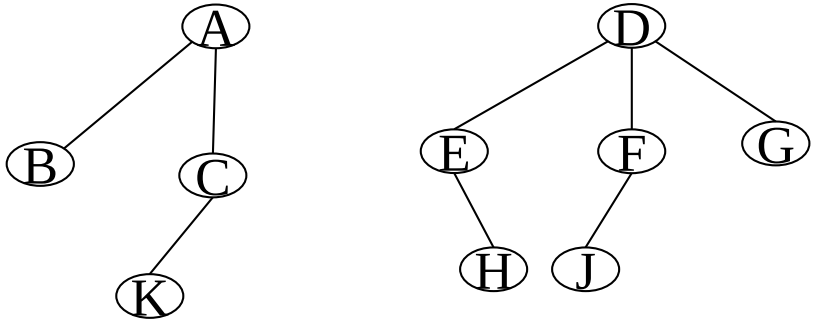
- 特点
  - 找子女容易；找父母难
- 合并若干个子树成一个新树时要考虑多个结点表的合并，比较麻烦

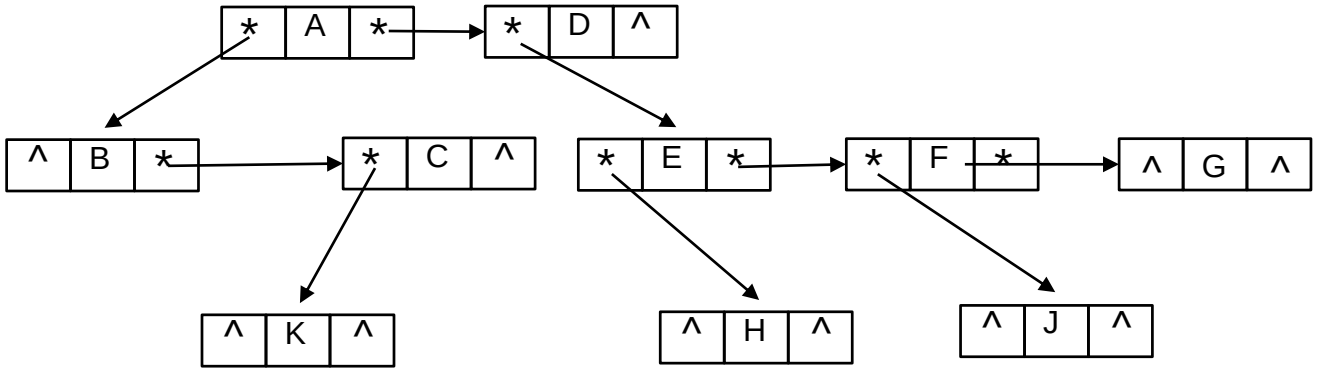
树和树林的存储表示3

	顺序存储	链式存储
长子-兄弟表示法	数组上每个结点存长子、兄弟结点的下标	数组上每个结点存长子、兄弟结点的指针

树和树林的存储表示3

- 长子-兄弟表示法
  - 每个结点存储
    - 结点的值
    - 最左子结点指针（下标）
    - 右侧兄弟结点指针（下标）





比子表表示法空间效率更高，且结点数组中的每个结点仅需要固定大小的存储空间，也称左子右兄表示法

长子-兄弟表示法

- 类型定义

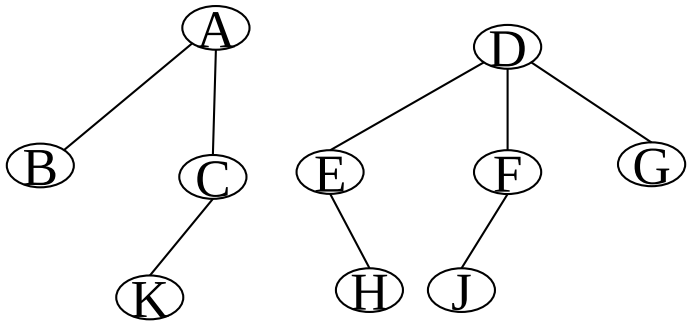
```
struct CSNode;
typedef  struct CSNode  * PCSNode;
struct CSNode {
      DataType          info;
      PCSNode           lchild;
      PCSNode           rsibling;
};
typedef  struct   CSNode  *CSTree;
typedef  CSTree   *PCSTree;
```

- 定义一个指向树/树林的指针变量

```
PCSTree  t;
```

长子-兄弟表示法的特点

- 缺点：找父结点麻烦。
  - 首先找到长子，然后再找父节点
- 优点：方便找子女、找兄弟等运算



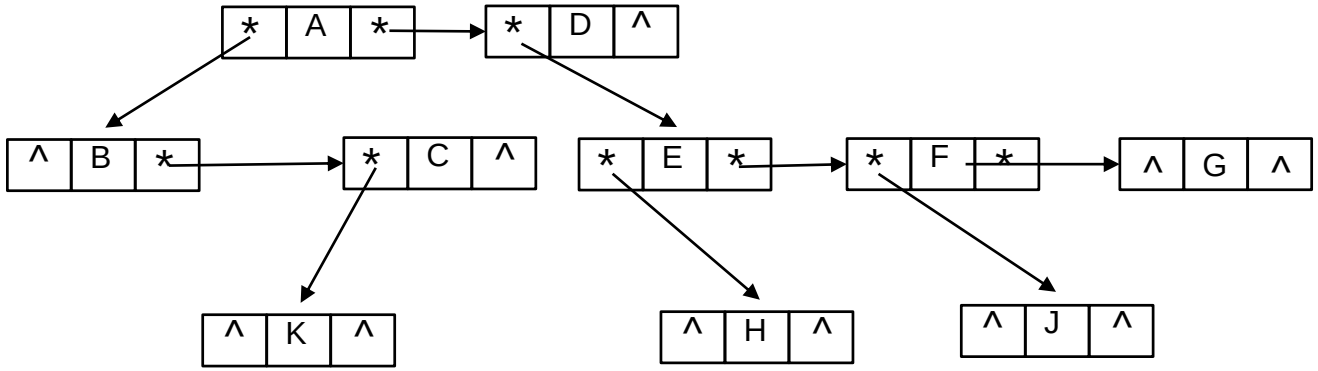
leftChild_cstree(p)　p->lchild

rightSibling_cstree(p)　p->rsibling

root_cstree(t)　t

isNull_cstree(t)　t==NULL;

找到全部子女很容易，先由lchild找到长子，再由rsibling字段逐个找右兄弟结点。



树与树林

- 树与树林的定义
- 树的基本运算
- 树和树林的存储表示
- 树和树林的周游
- 树和二叉树的转换

树的周游

- 定义：按某一规律系统地访问树的所有结点，并使每个结点恰好被访问一次。（又称为遍历）
- 周游的结果：
  - 在周游树的过程中，如果将各个结点按其被访问的先后顺序排列起来，则可得到一个包括所有结点的线性表；
  - 周游空树得到的线性表为空表；
  - 当树中只有一个结点时，对应的线性表也只有一个元素。
- 本质：将非线性结构转换为线性结构。

周游方法

- 周游一棵树所得到的线性表依赖于周游方法
  - 按深度方向周游[纵向遍历]
    - 先根次序
    - 中根次序
    - 后根次序
  - 按宽度方向周游[横向遍历]

先根次序

① 访问根结点；

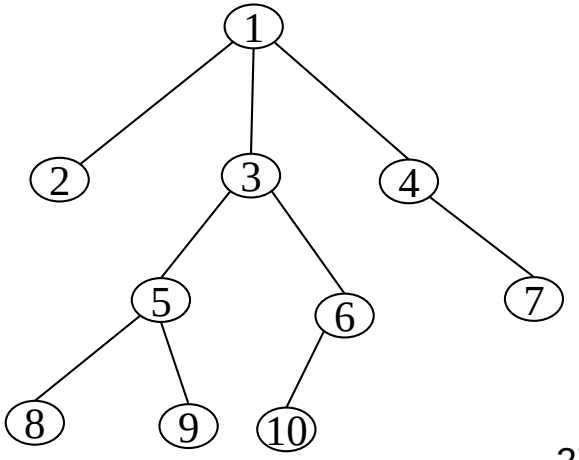
② 从左到右按先根次序周游根结点的每棵子树

按先根次序周游所列出的线性表为：

1　2　3　5　8　9　6　10　4　7

- 通常按先根次序对一棵树周游得到的线性表称为这棵树的先根序列。

- 按先根次序周游树的算法



先根周游-递归算法

```
① void preOrder(Node p)
② {
③     Node c;
④     visit(p);
⑤     c = leftChild(q);       /*获取该结点的长子*/
⑥     //按照从左往右的次序先根遍历该结点的子女
⑦      while (c!=NULL)
⑧      {
⑨          preOrder(c);              //先根遍历
⑩          c = rightSibling(c);      //右兄弟
⑪      }
⑫ }
```

中根次序

① 按中根次序周游根结点的最左子树；

② 访问根结点；

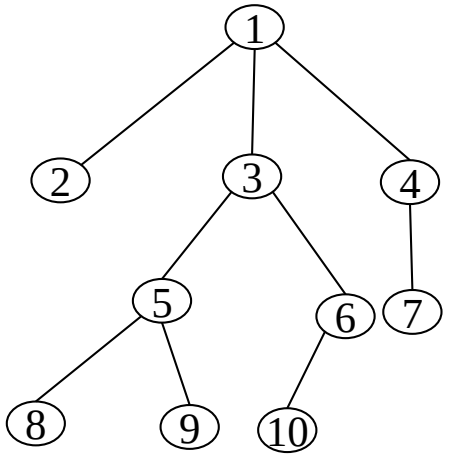
③ 从左到右按中根次序周游根结点的其它各子树

按中根次序周游得到的线性表为：

2　1　8　5　9　3　10　6　7　4

- 通常按中根次序对一棵树周游得到的线性表称为这棵树的中根序列。

- 按中根次序周游树的算法



教学网提供代码

中根周游-递归算法

```
①  void inOrder(Node p)
②  {
③    Node c;
④    c = leftChild(q);              //获取该结点的长子
⑤    if (c==NULL)  {  visit(p);       return; }       //不存在，访问该结点
⑥    inOrder(c);                 //中根遍历长子
⑦    visit(p);                    //访问根
⑧   c = rightSibling(c);     //中根遍历其它子女
⑨    while (c !=NULL)
⑩       {
⑪           inorder(c);
⑫           c = rightSibling(c);
⑬       }
⑭  }
```

后根次序

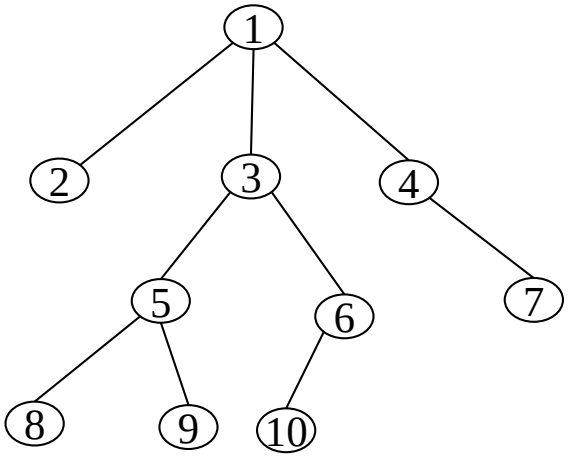
① 从左到右按后根次序周游根结点的每棵子树；
② 访问根结点

按后根次序周游得到的线性表为：

2　8　9　5　10　6　3　7　4　1

- 通常按后根次序对一棵树周游得到的线性表称为这棵树的后根序列。

- 按后根次序周游树的算法



后根周游-递归算法

```
① void postOrder(Node p)
② {
③    Node c;
④     c = leftChild(q);          //获取该结点的长子
⑤    //按照从左往右的次序后根遍历该结点的所有子女
⑥     while (c!=NULL)
⑦     {
⑧          postOrder( c);              //后根遍历
⑨         c = rightSibling(c);       //继续后根遍历右兄弟
⑩     }
⑪     visit(p);                    //最后访问根
⑫ }
```

深度优先周游的特点

- 相同点：在先、中和后根周游序列中，树结点的左右次序不变;
- 不同点：那些属于同一条路径上的结点，即只有祖先和子孙之间的相对次序，在上述三种序列中可能有所不同
  - 在先根周游序列中，结点的所有子孙都紧密排列在该结点的右边；
  - 假定post(n)表示结点n在先根序列中的位置，desc(n)表示结点n的子孙个数，则结点x是结点n的子孙的充分必要条件为：

$$post(n)+desc(n) \geq post(x) > post(n)$$

  - 在后根周游序列中，结点的所有子孙都紧密排列在该结点的左边；
  - 假定post(n)表示结点n在后根序列中的位置，desc(n)表示结点n的子孙个数，则结点x是结点n的子孙的充分必要条件为：

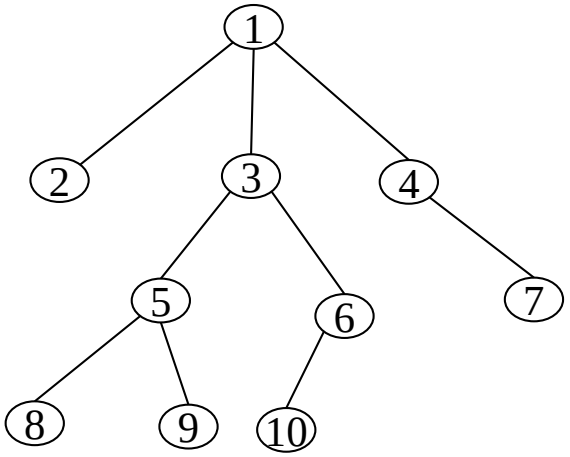
$$post(n)-desc(n) \leq post(x) < post(n)$$

广度优先周游树的算法

- 按宽度方向周游：这种策略是先访问层数为0的结点，然后从左到右逐个访问层数为1的结点，依此类推，直到访问完树中的全部结点。

  1　2　3　4　5　6　7　8　9　10

- 按宽度方向周游所得到的线性表叫作树的层次序列
- 特点：
  - 在层次序列中，层数较低的结点总是排在层数较高的结点之前
  - 保持同层结点的左右次序



广度优先周游树的算法

```
① void levelOrder(Tree t)
② {
③    Node c;    Queue q;          //队列元素的类型为Node
④    q = createEmptyQueue();     //建立空队列
⑤    c = root(t);
⑥    if (c==NULL)    return;
⑦    enQueue(q, c);              //根入队列
⑧    while (!isEmptyQueue(q))     //队列非空
⑨    {
⑩        ……
```

树林的周游1

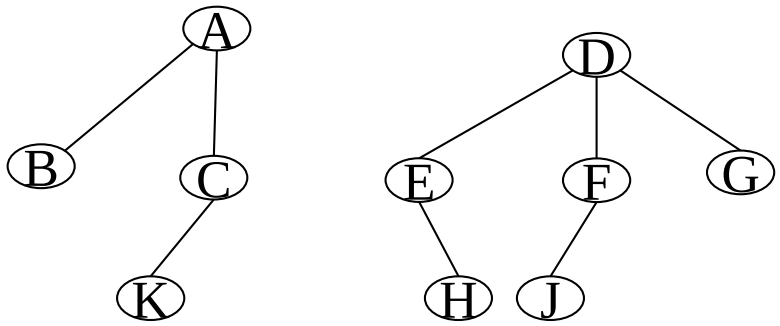
- 树林的周游方法有两种：先根周游和后根周游
- 先根周游

① 访问树林中第一棵树的根结点；

② 先根周游第一棵树的根结点的子树构成的树林；

③ 先根周游除去第一棵树之后的子树林。

- 先根遍历序列：（A, B, C, K, D, E, H, F, J, G)



树林的周游2

- 后根周游

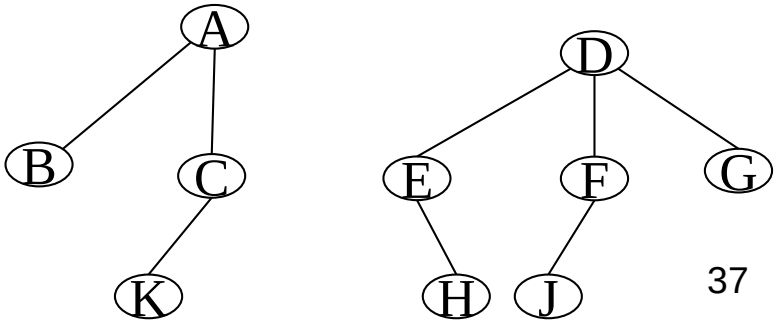
① 后根周游第一棵树的根结点的子树构成的树林；

② 访问树林中第一棵树的根结点；

③ 后根周游除去第一棵树之后的子树林。

后根遍历序列：(B, K, C, A, H, E, J, F, G, D)

对树林的先根或后根次序周游的定义，等价于逐个按照先根或后根次序周游树林中的每个树的效果



树/树林与二叉树的转换

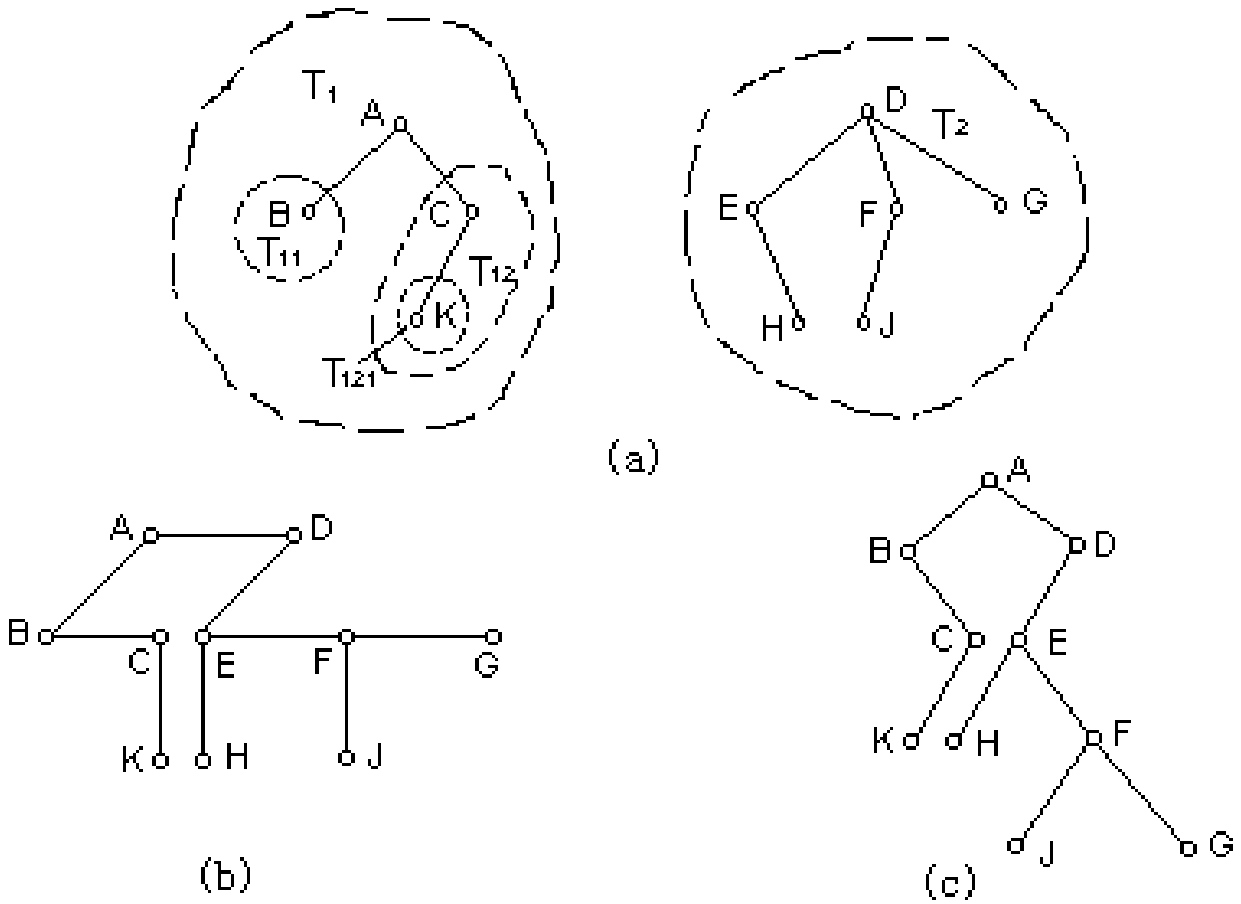
- 在树或树林与二叉树之间有一个自然的一一对应的关系
  - 任何树林都唯一地对应到一棵二叉树
  - 反过来，任何二叉树也唯一地对应到一个树林

树、树林 ⟺ **二叉树**

树或树林转换为二叉树

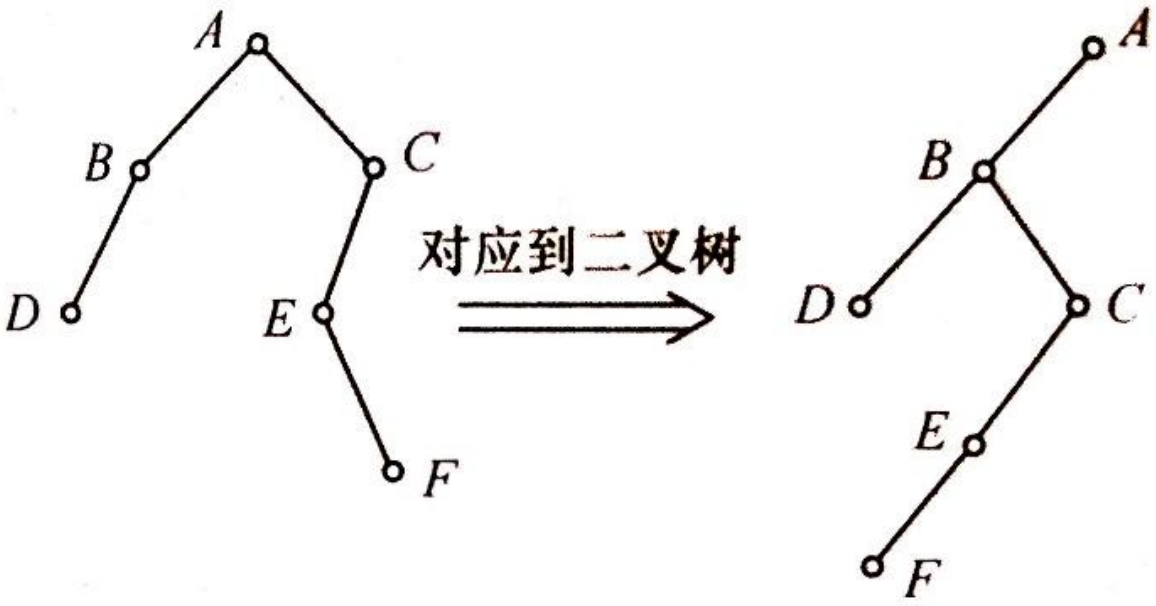
- 方法
  - 在所有相邻的兄弟结点之间连一条线；
  - 对每个内部结点，只保留它到其最左子女的连线，删去它与其它子女的连线；
  - 以根结点为轴心，将整棵树进行旋转。
- 树林($F=T_1,T_2,\ldots,T_n$)对应的二叉树B(F)
  - 若n = 0，则B(F)为空；
  - 若n＞0，则B(F)的根是T_1，B(F)的左子树是B(T_{11}，T_{12}，…，T_{1m})，其中T_{11}，T_{12}，…，T_{1m}是T_1的子树；B(F)的右子树是B(T_2，…，T_n)

树或树林转换为二叉树



树或树林转换为二叉树

- 在树或树林所对应的二叉树里，一个结点的左子女是它在树里的第一个子女，右子女是它的兄弟。
- 树对应到二叉树其根结点的右子树总是为空

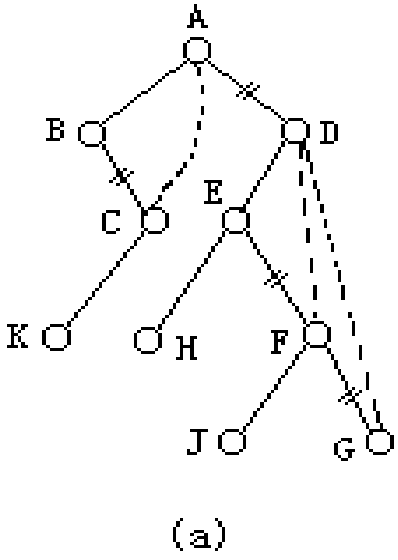


二叉树转换为树或树林

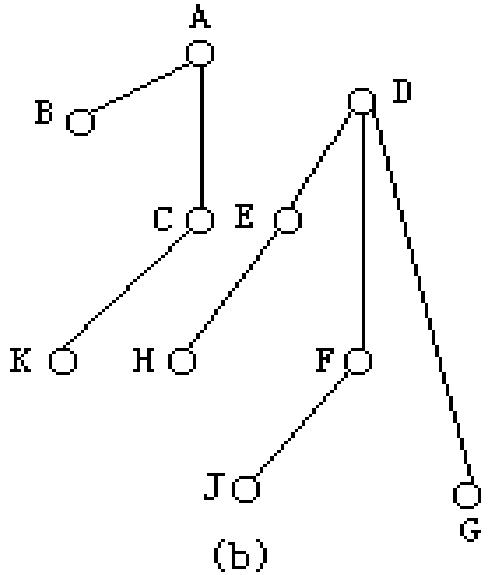
- 设B是一棵二叉树，r是B的根，L是r的左子树，R是r的右子树，则对应于B的树林F(B)的定义是：
  - 若B为空，则F(B)是空的树林；
  - 若B不为空，则F(B)是一棵树T_1加上树林F(R)，其中树T_1的根为r，r的子树为F(L)

二叉树转换为树或树林

- 方法
  - 若某结点是其父母的左子女，则把该结点的右子女，右子女的右子女……，都与该结点的父母连起来；
  - 去掉原二叉树中所有的父母到右子女的连线



(a)



(b)

本章小结

- 二叉树：
  - 7个重要性质
  - 存储方式：二叉链表
  - 二叉树周游算法: DLR、LDR、LRD
  - HuffMan树： 定义、表示、算法和编/译码过程
- 树
  - 树的三种存储表示
  - 树的遍历方法：深度优先（先根、中根、后根）和宽度优先
- 树林
  - 树林到二叉树的相互转换

练习

- 若某树有n1 个度数为1 的结点，有n2 个度数为2 的结点，……有nm 个度数为m的结点，试问它有多少个叶结点，给出计算的过程。

假设叶子结点数为n0,并假设树的结点数为N,

N = n0+n1+n2+...+nm　又等于所有节点的分支数（或度数）+1。

N = n1+2*n2+3*n3+...+m*nm+1

这样得到n0+n1+n2+...+nm = 1+n1+2*n2+3*n3+...+m*nm

即得：n0 = n2+2*n3+3*n4+...+(m-1)*nm+1

练习

- 如果结点A有3个兄弟(不包括A本身)，而且B是A的父结点，则B的度是（）。

  A.3

  B.4

  C.5

  D.1

练习

- 设高度为h的二叉树上只有度为0和度为2的结点，则此二叉树中所包含的结点数至少为( )。

  A. 2h

  B. 2h-1

  C. 2h+1

  D. h+1

练习

- 树最适合用来表示（）

  A 有序的数据元素

  B 无序的数据元素

  C 元素之间具有分支层次关系的数据

  D 元素之间无联系的数据

练习

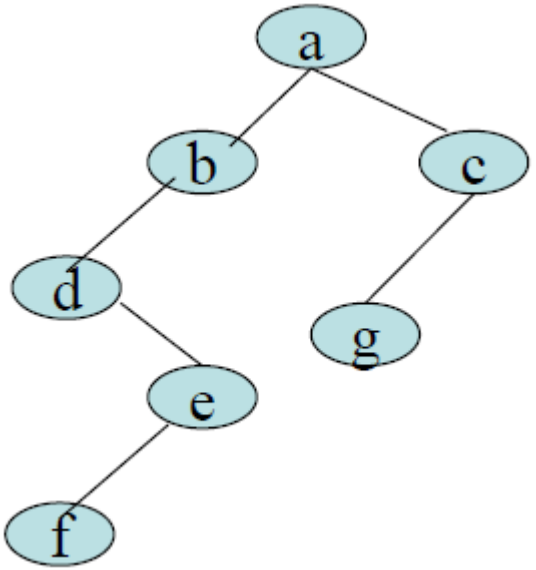
- 如图所示二叉树的中序遍历序列是（）

  A abcdgef

  B dfebagc

  C dbaefcg

  D defbagc



练习

- 已知某二叉树的后序遍历序列是dabec，中序遍历序列是debac，它的前序遍历序列是（）。

  A acbed

  B decab

  C deabc

  D cedba

练习

- 任何一棵二叉树的叶结点在先序、中序和后序的遍历序列中的相对次序（）

  A 不发生改变

  B 发生改变

  C 不能确定

  D 以上都不对

练习

- 某二叉树的先根序列和后根序列正好相反，则该二叉树一定是（）二叉树。
  - A．空或只有一结点
  - B．树的高度等于其结点数减1
  - C．任一结点都只有右子结点
  - D．任一结点都只有左子结点

练习

- 设n,m为一棵二叉树上的两个结点，在中序遍历时，n在m前的条件是（）

  A. n在m右方

  B. n在m祖先

  C. n在m左方

  D. n在m子孙

练习

- 在一非空二叉树的中序遍历序列中，根结点的右边( )

  A.只有右子树上的所有结点

  B.只有右子树上的部分结点

  C.只有左子树上的所有结点

  D.只有左子树上的部分结点

练习

- 某二叉树T有n个结点,设按某种顺序对T中的每个结点进行编号，编号值为1,2,...n.且有如下性质:T中任意结点v,其编号等于左子树上的最小编号减一,而v的右子树的结点中,其最小编号等于v左子树上结点的最大编号加一，这是按( )编号的。

  A.中序（中根）遍历序列

  B.前序（先根）遍历序列

  C.后序（后根）遍历序列

  D.层次顺序

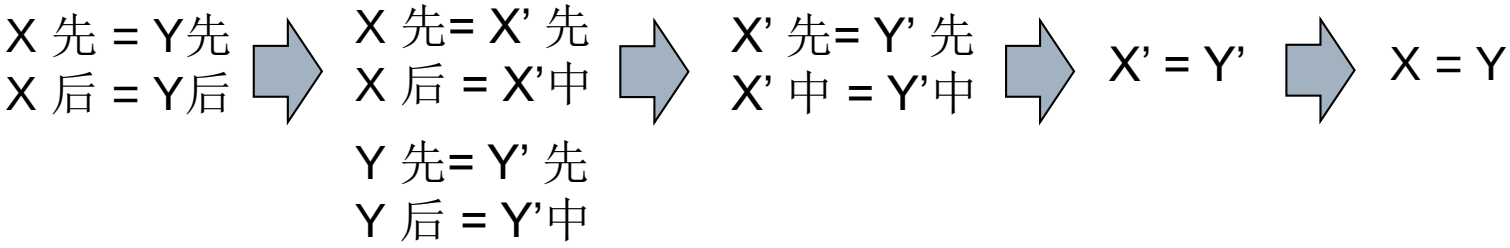
练习

- 已知一棵树边的集合为{<A,B>, <A,C>, <B,D>, <B,E>, <B,F>, <C,G>, <C,H>, <E,I>, <E,J>}，在这棵树中，结点D的父结点是_____，结点E的层数是_____，结点B的度数为_____，后根深度优先遍历该树的序列为_____。

练习

- 结点最少的树是_____，结点最少的二叉树是_____。
- 设森林F中有三棵树，第一，第二，第三棵树的结点个数分别为M1，M2和M3。与森林F对应的二叉树根结点的右子树上的结点个数是_____。

  A．M1 B．M1+M2 C．M3 D．M2+M3
- 已知一棵树的先根次序遍历的结果与其对应二叉树表示(长子-兄弟表示)的前序遍历结果相同, 树的后根次序遍历结果与其对应二叉树表示的中序遍历结果相同。试问利用树的先根次序遍历结果和后根次序遍历结果能否唯一确定一棵树?如果正确，请证明，如果错误，举例说明。



练习

- 对于表达式(a-b)*d/(e+f)

  优先级相同时，先左后右

  - 请画出它的中序二叉树，
  - 给出该二叉树的前缀表达式和后缀表达式

判断

- 二叉树的先根遍历序列中，任意一个结点均处在其子女结点的前面。
- 由树转换成二叉树，其根结点的右子树总是空的。
- 哈夫曼树是带权路径长度最短的树，路径上权值较大的结点离根较近。

算法设计

- 二叉树采用链式存储结构，设计一个按层次顺序（同一层自左向右）遍历二叉树的算法

  - 采用一个队列q，先将二叉树根结点入队列，然后退队列，输出该结点，若它有左子树，便将左子树的根结点入队列，若它有右子树，便将右子树根结点入队列，如此直到队列为空为止
  - 因为队列是先进先出的，从而达到按层次顺序遍历二叉树的目的