1. 树的定义
2. 树的储存结构
   1. 双亲表示法
   2. 孩子表示法
   3. 双亲孩子表示法
   4. 孩子兄弟表示法
3. 二叉树
   1. 二叉树的遍历
      1. 前序
      2. 中序
      3. 后序
      4. 层序
4. 线索二叉树
   1. 线索二叉树的代码实现 BiThrTree.c
5. 赫尔曼树

## 【树的定义和基本术语】

非线性数据结构。

树的递归定义：

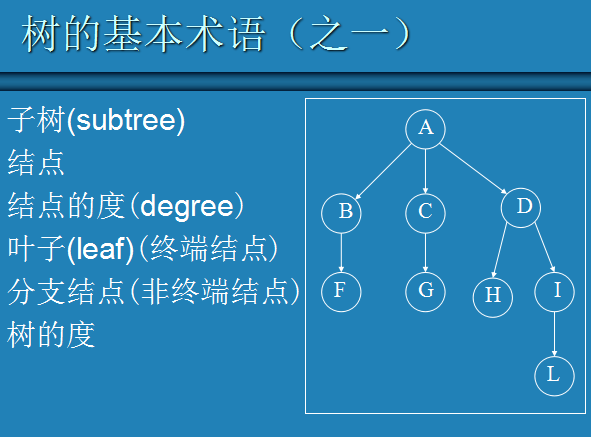
树(tree)是n(n>=0)个结点的有限集。

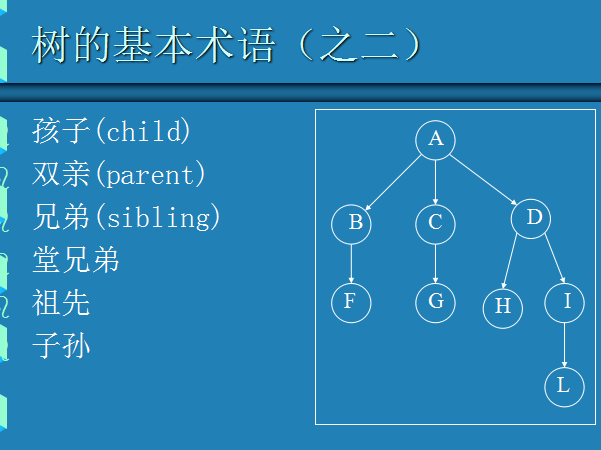
当n>0时,

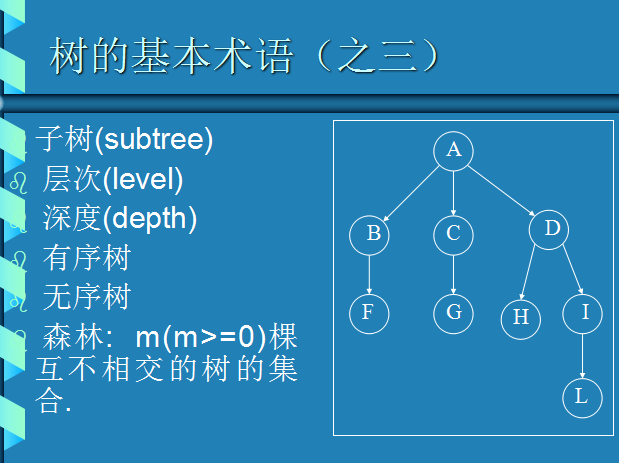
(1)有且仅有一个特定的称为根(root)的结点;

(2)当n>1时,其余结点可分为m(m>0)个互不相交的有限集T1,T2,...,Tm,其中每个集合本身又是一棵树。称为子树(subtree)。

【基本术语】

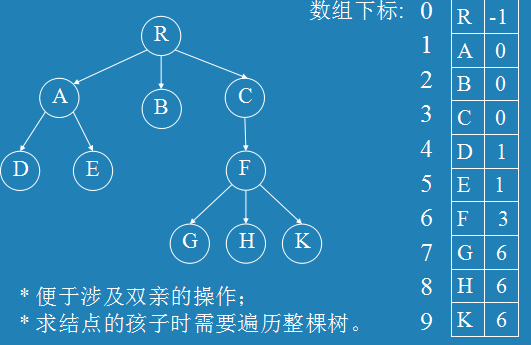






## 【树的存储结构】

# 双亲表示法



//-----------树的双亲表存储表示----------//

#define MAX\_TREE\_SIZE 100

typedef struct PTNode {

TElemType data;

int parent; //双亲位置域

}PTNode;

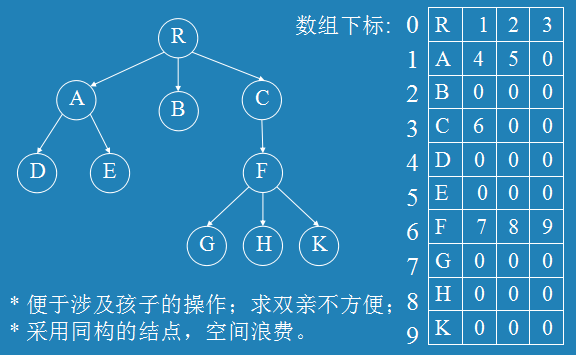
typedef struct{

PTNode nodes[MAX\_TREE\_SIZE];

int n; //结点数

}PTree;

# 孩子表示法



顺序存储

#define MAX\_TREE\_SIZE 100

typedef struct PTNode {

TElemType data;

int child1; //第1个孩子位置域

int child2; //第2个孩子位置域

......

int childd; //第d个孩子位置域

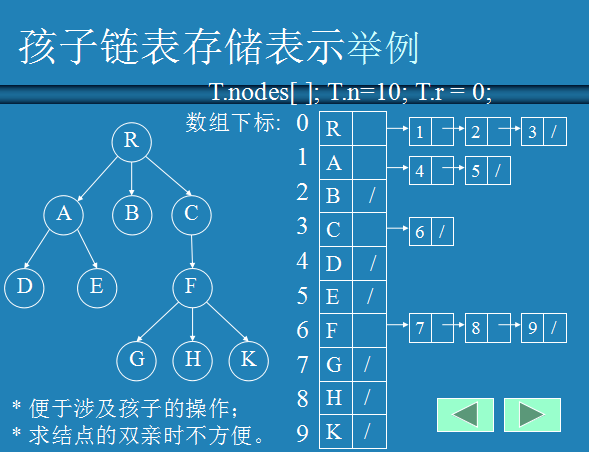
}PTNode;

typedef struct{

PTNode nodes[MAX\_TREE\_SIZE];

int n; //结点数

}PTree;



链式储存

typedef struct CTNode{ //孩子结点

int child;

struct CTNode \*next;

} \*ChildPtr;

typedef struct {

TElemType data;

ChildPtr firstchild; //孩子链表头指针

}CTBox;

typedef struct {

CTBox nodes[MAX\_TREE\_SIZE];

int n, r; //结点数和根的位置

}CTree;阿

# 双亲孩子表示法



#define MAX\_TREE\_SIZE 100

typedef char ElemType;

typedef struct CTNode

{

int child;

struct CTNode \*next;

}\*ChildPtr;

typedef struct

{

ElemType data;

int parent;

ChildPtr firstchild;

}CTBox;

typedef struct

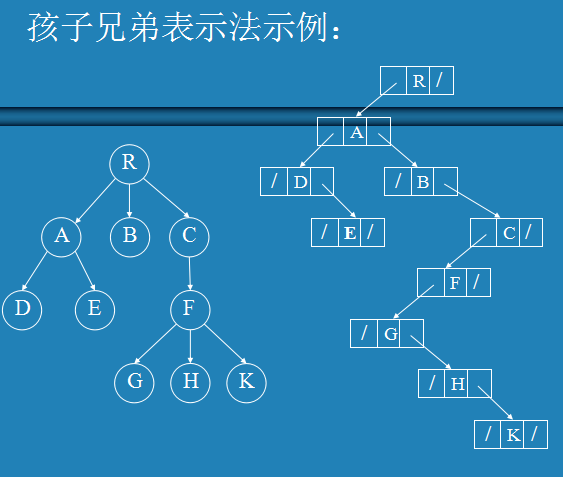
{

CTBox nodes[MAX\_TREE\_SIZE];

int r,n;

}

1. 孩子兄弟表示法



/\*树的孩子兄弟表示法结构的定义\*/

typedef struct CSNode

{

TElemType data;

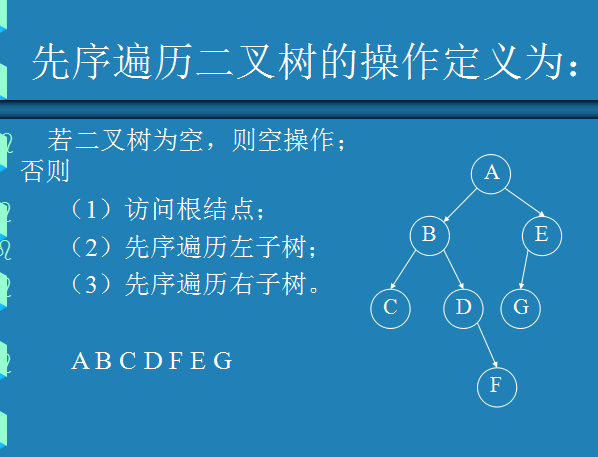
struct CSNode \*firstchild,\*rightsib;

}CSNode,\*CSTree;

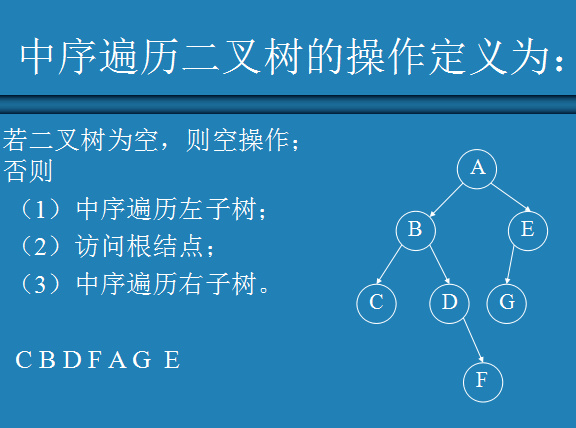
【二叉树】

【二叉树的遍历】

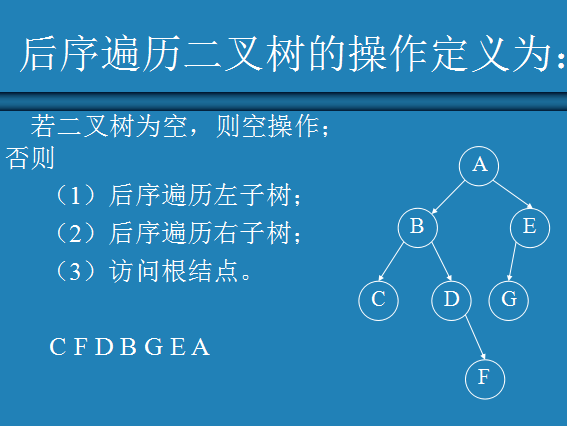
1. 前序遍历：根左右



1. 中序遍历，(手指法)



1. 后序遍历

 4.层序遍历

【线索二叉树】

为什么要线索二叉树？？

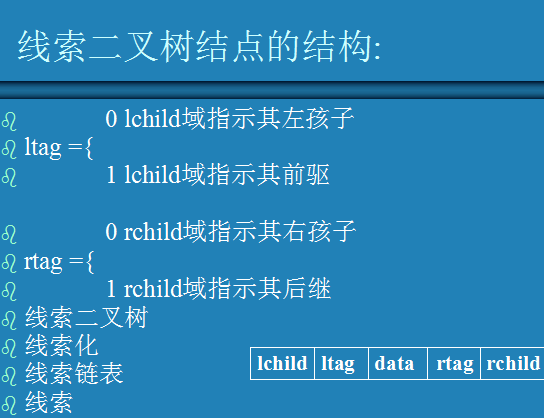
普通二叉树的缺点

1. 浪费空间
2. 浪费时间

实验证明，通过中序遍历的二叉树可以利用空间记录前驱和后继

记录这些有什么用？

1. 当定位到它的节点的是时候，可以快速找出它的前边是谁，后面是谁



/\*\*

\* BiThrTree.c

\* 线索二叉树的代码实现

\*

\*

\*/

#include <stdio.h>

#include <stdlib.h>

typedef enum {Link,Thread} PointerTag;

typedef char ElemType;

typedef struct BiThrNode

{

ElemType data;

struct BiThrNode \*lchild,\*rchild;

PointerTag ltag;

PointerTag rtag;

}BiThrNode,\*BiThrTree;

BiThrTree pre;

CreateBiThrTree(BiThrTree \*T)

{

char c;

scanf("%c",&c);

if(' ' == c)

{

(\*T) = NULL;

}

else

{

(\*T) = (BiThrNode \*)malloc(sizeof(BiThrTree));

(\*T)->data = c;

(\*T)->ltag = Link;

(\*T)->ltag = Link;

CreateBiThrTree(&(\*T)->lchild);

CreateBiThrTree(&(\*T)->rchild);

}

}

InThreading(BiThrTree T)

{

if(T)

{

InThreading(T->lchild);

if(!T->lchild)

{

T->ltag = Thread;

T->lchild = pre;

}

if(!pre->rchild)

{

pre->rtag = Thread;

pre->rchild = T;

}

pre = T;

InThreading(T->rchild);

}

}

InOrderThreading(BiThrTree \*p,BiThrTree T)

{

\*p =(BiThrTree)malloc(sizeof(BiThrNode));

(\*p)->ltag = Link;

(\*p)->rtag = Thread;

(\*p)->rchild = \*p;

if(!T)

{

(\*p)->lchild=\*p;

}

else

{

(\*p)->lchild=T;

pre = \*p;

InThreading(T);

pre->rchild = \*p;

pre->rtag = Thread;

(\*p)->rchild = pre;

}

}

void visit(char c)

{

printf("%c",c);

}

void InOrderTraverse(BiThrTree T)

{

BiThrTree p;

p = T->lchild;

while(p!=T)

{

while(p->ltag==Link)

{

p = p->lchild;

}

visit(p->data);

while(p->rtag == Thread && p->rchild!=T)

{

p = p->rchild;

visit(p->data);

}

p = p->rchild;

}

}

int main()

{

BiThrTree P,T = NULL;

CreateBiThrTree(&T);

InOrderThreading(&P,T);

InOrderTraverse(P);

printf("\n");

return 0;

}