树形结构

## 前序、后序线索二叉树的实现及遍历

二叉树的线索化：

线索化二叉树需要对二叉树进行一次相对应的遍历过程，将原先为空的左孩子节点链接到他的遍历序的前驱节点，为空的又孩子节点链接到遍历序的后继节点。并且需要在节点信息多两个域记录该节点的两个孩子节点指针是孩子还是其线索化所添加的链接。

Node \*pre;

void thread\_LDR(Node \*now)

{

if(now == NULL) return;

thread\_LDR(now->lson);

if(pre != NULL && pre->rson == NULL) {

pre->rf = 1;

pre->rson = now;

}

if(now->lson == NULL) {

now->lf = 1;

now->lson = pre;

}

pre = now;

thread\_LDR(now->rson);

}

void thread\_DLR(Node \*now)

{

if(now == NULL) return;

// now->show();

// system("pause");

if(pre != NULL && pre->rson == NULL)

{

pre->rf = 1;

pre->rson = now;

}

if(now->lson == NULL)

{

now->lf = 1;

now->lson = pre;

}

pre = now;

if(now->lf == 0) thread\_DLR(now->lson);

if(now->rf == 0) thread\_DLR(now->rson);

}

void thread\_LRD(Node \*now)

{

if(now == NULL) return;

if(now->lf == 0) thread\_LRD(now->lson);

if(now->rf == 0) thread\_LRD(now->rson);

if(pre != NULL && pre->rson == NULL)

{

pre->rf = 1;

pre->rson = now;

}

if(now->lson == NULL)

{

now->lf = 1;

now->lson = pre;

}

pre = now;

}

便利过程我们就可以不断通过寻找其后继节点来实现，因为有了之前的线索，所以我们非递归的访问节点的信息可以很方便的得到：

void show\_thread\_LDR(Node \*root)

{

Node \*now = root;

while(now->lf == 0) now = now->lson;

now->show();

while(now->rson != NULL)

{

if(now->rf == 1) now = now->rson;

else

{

now = now->rson;

while(now->lf == 0 && now->lson != NULL)

now = now->lson;

}

now->show();

}

}

void show\_thread\_DLR(Node \*root)

{

Node \*now = root;

while(now != NULL)

{

now->show();

// system("pause");

if(now->lf == 0)

now = now->lson;

else

now = now->rson;

}

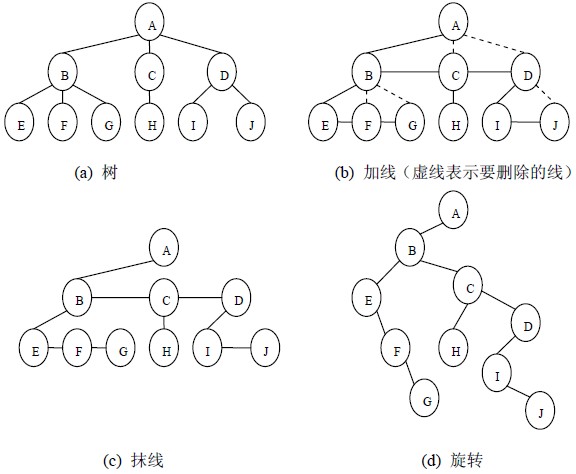
}

## 二叉树与森林之间的转换算法

1、树转换为二叉树

由于二叉树是有序的，为了避免混淆，对于无序树，我们约定树中的每个结点的孩子结点按从左到右的顺序进行编号。

将树转换成二叉树的步骤是：  
（1）加线。就是在所有兄弟结点之间加一条连线；  
（2）抹线。就是对树中的每个结点，只保留他与第一个孩子结点之间的连线，删除它与其它孩子结点之间的连线；  
（3）旋转。就是以树的根结点为轴心，将整棵树顺时针旋转一定角度，使之结构层次分明。



代码如下：

void tree\_to\_binary(Node \*now)

{

// cout << "now = " << now->data << endl;

for (int i = now->cnt - 1; i >= 0; i--)

{

if(now->sons[i]->cnt)

{

tree\_to\_binary(now->sons[i]);

}

if(i == 0)

{

now->lson = now->sons[i];

now->lson->father = now;

}

else

{

now->sons[i-1]->rson = now->sons[i];

now->sons[i]->father = now->sons[i-1];

}

}

now->cnt = 0;

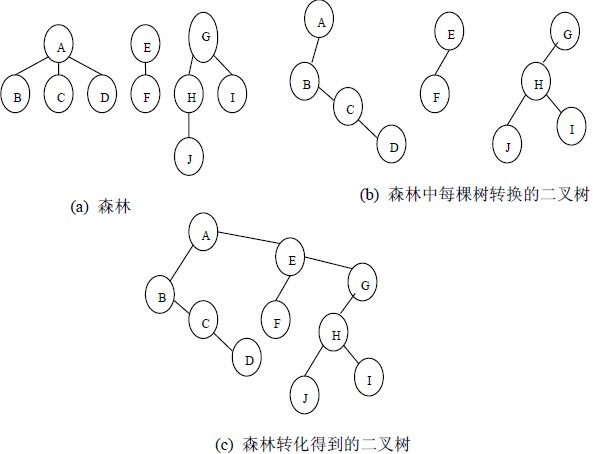
memset(now->sons, -1, sizeof(now->sons));

}

2、森林转换为二叉树

森林是由若干棵树组成，可以将森林中的每棵树的根结点看作是兄弟，由于每棵树都可以转换为二叉树，所以森林也可以转换为二叉树。

将森林转换为二叉树的步骤是：  
（1）先把每棵树转换为二叉树；  
（2）第一棵二叉树不动，从第二棵二叉树开始，依次把后一棵二叉树的根结点作为前一棵二叉树的根结点的右孩子结点，用线连接起来。当所有的二叉树连接起来后得到的二叉树就是由森林转换得到的二叉树。



代码如下：

Node\* forest\_to\_binary(vector<Node\*> heads)

{

int len = heads.size();

for (int i = 0; i < len; i++)

{

tree\_to\_binary(heads[i]);

}

Node \*root = heads[0], \*tmp = heads[0];

for (int i = 1; i < len; i++)

{

tmp->rson = heads[i];

heads[i]->father = tmp;

tmp = heads[i];

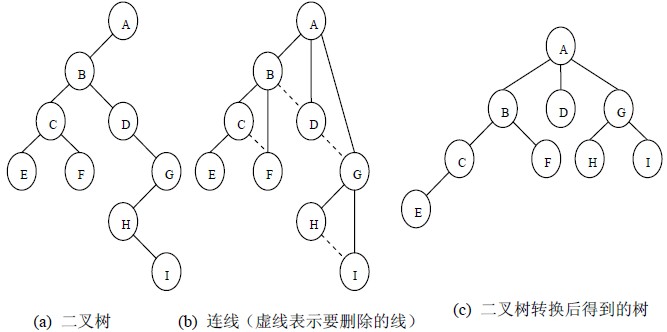
}

return root;

}

3、二叉树转换为树

二叉树转换为树是树转换为二叉树的逆过程，其步骤是：  
（1）若某结点的左孩子结点存在，将左孩子结点的右孩子结点、右孩子结点的右孩子结点……都作为该结点的孩子结点，将该结点与这些右孩子结点用线连接起来；  
（2）删除原二叉树中所有结点与其右孩子结点的连线；  
（3）整理（1）和（2）两步得到的树，使之结构层次分明。



代码如下：

void binary\_to\_tree(Node \*now, Node \*pre)

{

// cout << "to tree" << now->data << endl;

if(pre != NULL)

{

pre->sons[pre->cnt++] = now;

}

if(now->lson != NULL) {

binary\_to\_tree(now->lson, now);

now->lson = NULL;

}

if(now->rson != NULL) {

binary\_to\_tree(now->rson, pre);

if(pre != NULL)now->rson = NULL;

}

}

4、二叉树转换为森林

二叉树转换为森林比较简单，其步骤如下：  
（1）先把每个结点与右孩子结点的连线删除，得到分离的二叉树；  
（2）把分离后的每棵二叉树转换为树；  
（3）整理第（2）步得到的树，使之规范，这样得到森林。

代码如下：

void binary\_to\_forest(Node \*root)

{

heads.clear();

while(root != NULL)

{

cout << "heads " << heads[0]->data << endl;

heads.push\_back(root);

binary\_to\_tree(root, NULL);

root = root->rson;

}

}

## K叉哈夫曼树的实现

对于k叉哈夫曼树，与二叉哈夫曼树不同的地方就在于每次减少的节点数，我们需要保证每次将k个节点合并成一个节点，并且最后剩下一个节点。因此我们需要在最开始加入一些虚拟节点，这样同时保证了虚拟节点在哈夫曼树的最底层，因为出现概率为0，这样就将k叉哈夫曼树的做法划归到二叉哈夫曼树的做法上了。

计算需要添加的虚拟节点个数：

int tmpr = que1.size() % (k-1);

int add;

if (tmpr == 0) add = 1;

else if (tmpr == 1) add = 0;

else add = k - tmpr;