



数据结构与算法（六）

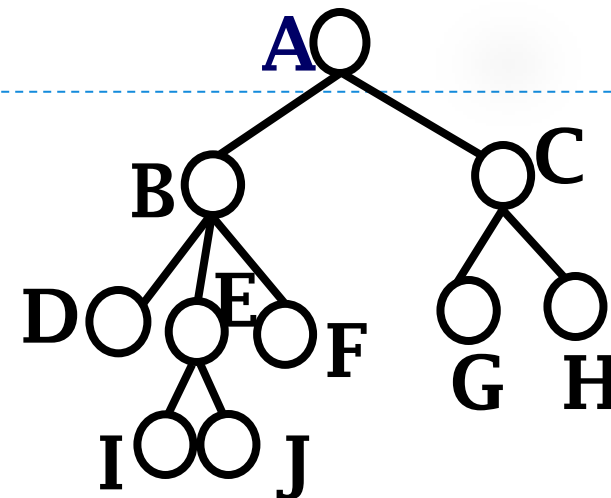
张铭 主讲

采用教材：张铭，王腾蛟，赵海燕 编写
高等教育出版社，2008.6（“十一五”国家级规划教材）

<http://www.jpku.pku.edu.cn/pkujpku/course/sjjg>

第6章 树

- 树的定义和基本术语
- 树的链式存储结构
 - “子结点表”表示方法
 - 静态“左孩子/右兄弟”表示法
 - 动态表示法
 - 动态“左孩子/右兄弟”表示法
 - 父指针表示法及其在并查集中的应用
- 树的顺序存储结构
- K叉树

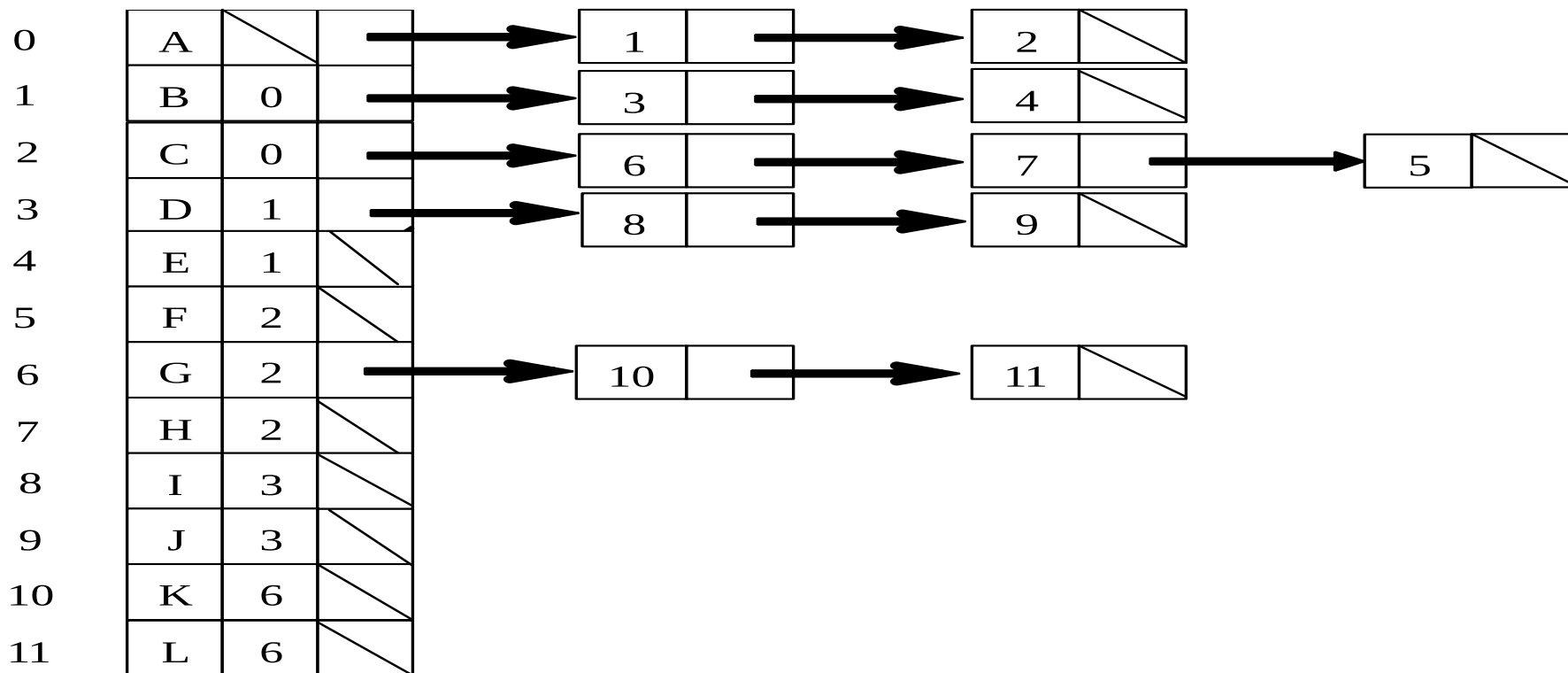


6.2 树的链式存储结构

“子结点表”表示方法

list of children, 就是图的邻接表。

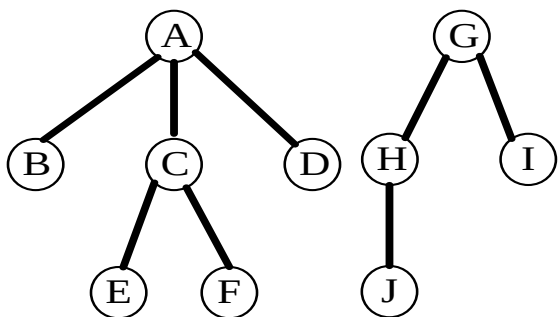
索引 值 父结点 子结点



6.2 树的链式存储结构

静态 “左孩子/右兄弟” 表示法

- 在数组中存储的 “子结点表”

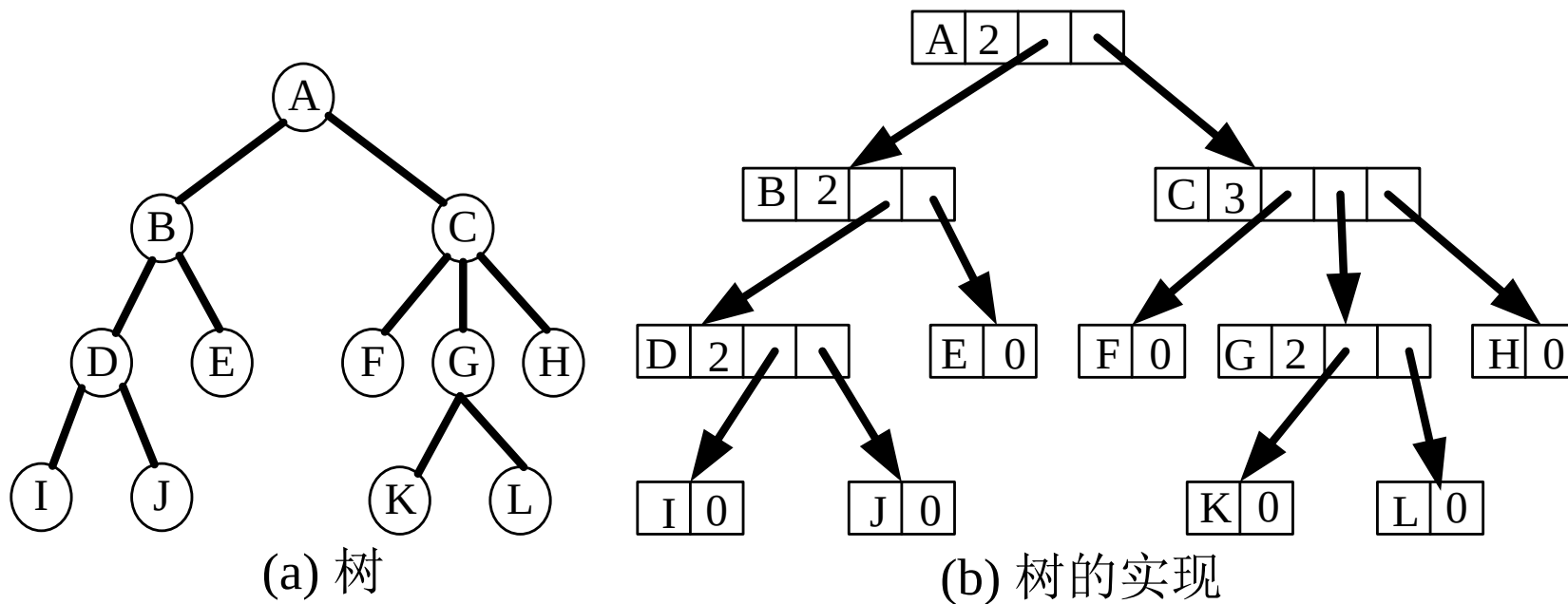


左子 结点	值	父 结点	右兄弟 结点
→	A	↖	↖
→	B	0	↖
→	C	0	↖
→	D	0	↖
→	E	2	↖
→	F	2	↖
→	G	↖	↖
→	H	6	↖
→	I	6	↖
→	J	7	↖

6.2 树的链式存储结构

动态表示法

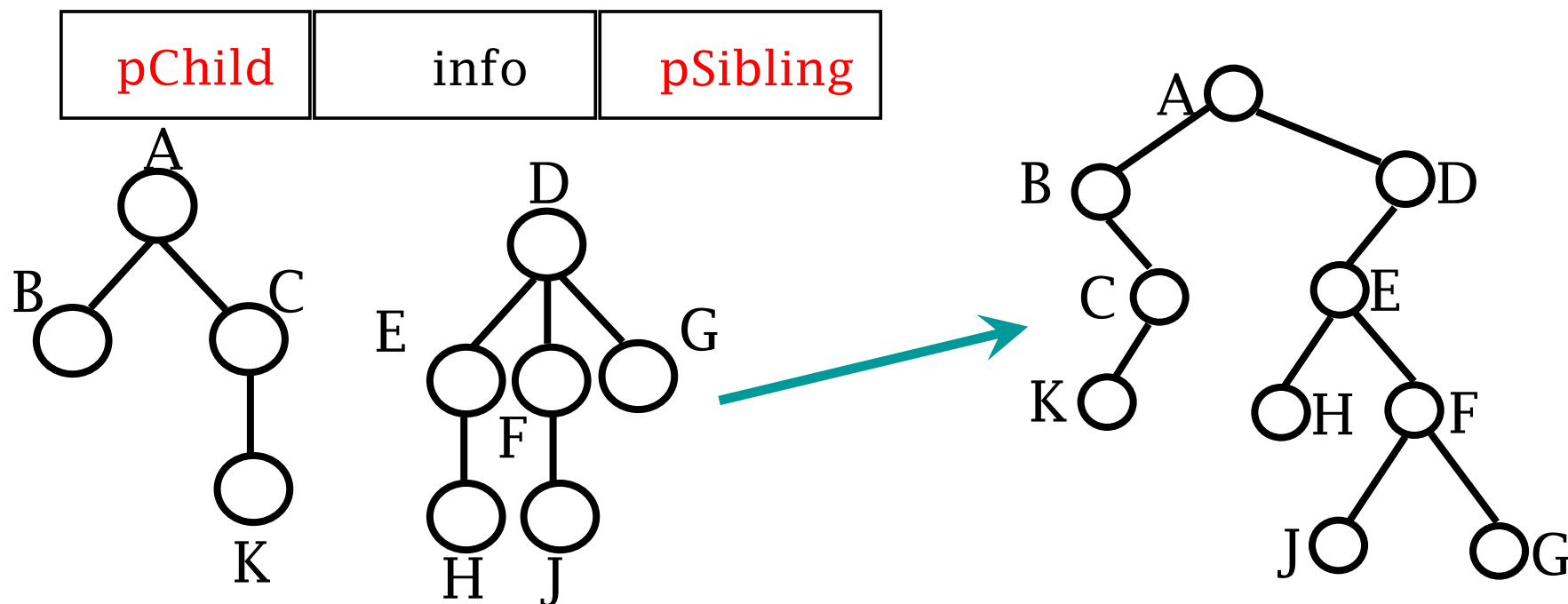
- 每个结点分配可变的存储空间
 - 子结点数目发生变化，需要重新分配存储空间



6.2 树的链式存储结构

动态“左子/右兄”二叉链表示法

- 左孩子在树中是结点的最左子结点，右子结点是结点原来的右侧兄弟结点
- 根的右链就是森林中每棵树的根结点





动态二叉链表树的关键实现细节

```
// 在TreeNode的抽象类中增加以下私有数据成员
private:
T m_Value;           // 树结点的值
TreeNode<T> *pChild; // 第一个左孩子指针
TreeNode<T> *pSibling; // 右兄弟指针
```

6.2 树的链式存储结构

寻找当前结点的父结点

```
template<class T>
TreeNode<T>* Tree<T>::Parent(TreeNode<T> *current) {
using std::queue;                                     // 使用STL队列
    queue<TreeNode<T>*> aQueue;
    TreeNode<T> *pointer = root;
    TreeNode<T> *father = upperlevelpointer = NULL;   // 记录父结点
    if (current != NULL && pointer != current) {
    while (pointer != NULL) {                           // 森林中所有根结点进队列
        if (current == pointer)                       // 森林中所有第一层根的父为空
        break;
        aQueue.push(pointer);                          // 当前结点进队列
        pointer=pointer-> RightSibling();              // 指针指向右
    }
}
```


6.2 树的链式存储结构

寻找当前结点的父结点

```

while (!aQueue.empty()) {
    pointer = aQueue.front();
    aQueue.pop();
    upperlevelpointer = pointer;
    pointer = pointer-> LeftMostChild();
    while (pointer) {
        if (current == pointer) {
            father = upperlevelpointer;
            break;}
        else {
            aQueue.push(pointer);
            pointer = pointer->RightSibling();}
    }
}
aQueue.clear( );
return father;
}

```

```

// 取队列首结点指针
// 当前元素出队列
// 指向上一层的结点
// 指向最左孩子
// 当前结点的子结点进队列

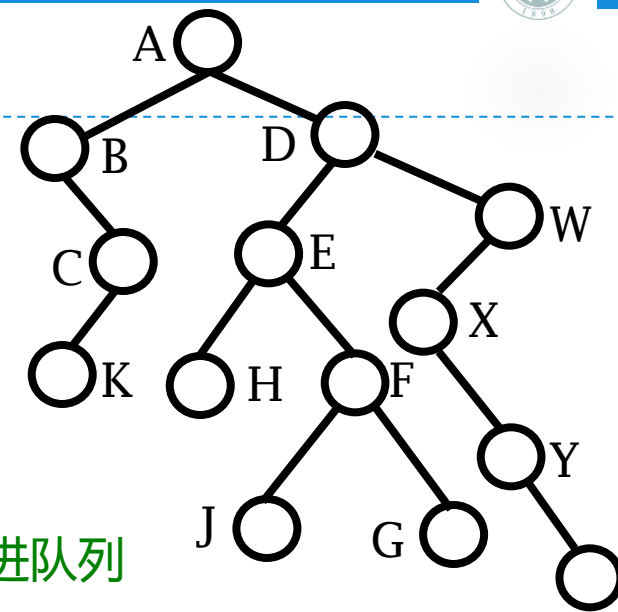
// 返回父

```

```

// 清空队列，也可以不写（局部变量）

```





删除以root为代表的森林的所有结点

```
template <class T>
void Tree<T>::DestroyNodes(TreeNode<T>* root) {
    if (root) {
        DestroyNodes(root->LeftMostChild()); //递归删除第一子树
        DestroyNodes(root->RightSibling());  //递归删除其他子树
        delete root; //删除根结点
    }
}
```

6.2 树的链式存储结构

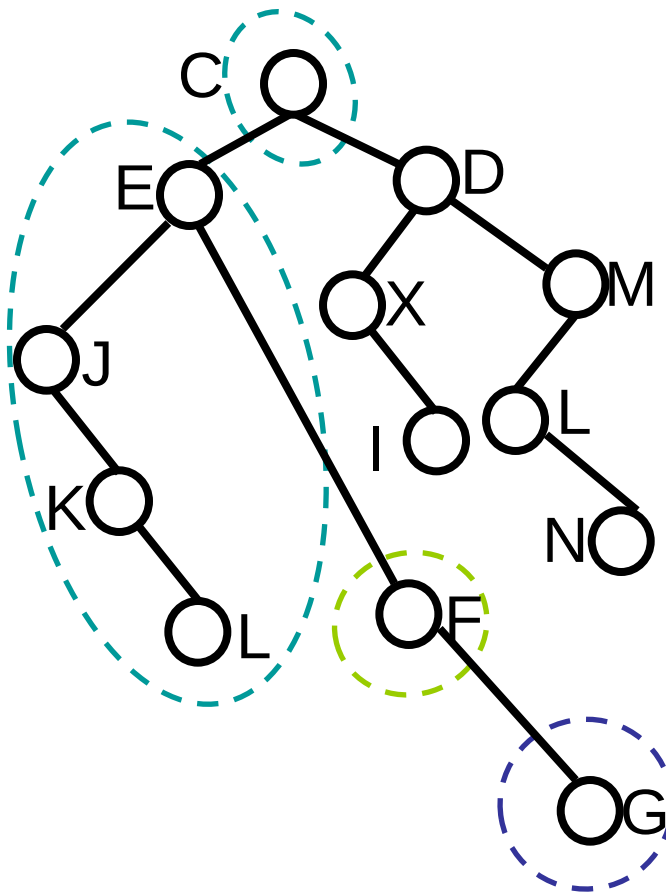
删除以subroot为根的子树

```
template<class T>
void Tree<T>::DeleteSubTree(TreeNode<T> *subroot) {
    if (subroot == NULL) return; // 若待删除的子树为空则返回
    TreeNode<T> *pointer = Parent (subroot); // 找subroot的父结点
    if (pointer == NULL) { // subroot没有父，则是某个树根
        pointer = root;
        while (pointer->RightSibling() != subroot) // 顺右链找左邻树根
            pointer = pointer->RightSibling();
        pointer->setSibling(subroot->RightSibling()); // 前后挂接，脱链
    }
    else if (pointer->LeftMostChild() == subroot) // subroot为最左子
        pointer->setChild(subroot->RightSibling()); // 挂新的最左
    else { // subroot有左兄弟的情况
        pointer = pointer->LeftMostChild(); // 下降到最左兄弟
        while (pointer->RightSibling() != subroot) // 顺右链找左邻兄弟
            pointer = pointer->RightSibling();
        pointer->setSibling(subroot->RightSibling()); // 前后挂接，脱链
    }
    subroot->setSibling(NULL); // 非常重要，丢了会出错
    DestroyNodes(subroot); // 删除以subroot代表的子森林的所有结点
}
```

6.2 树的链式存储结构

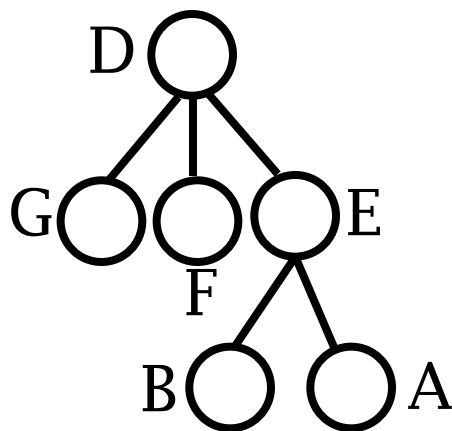
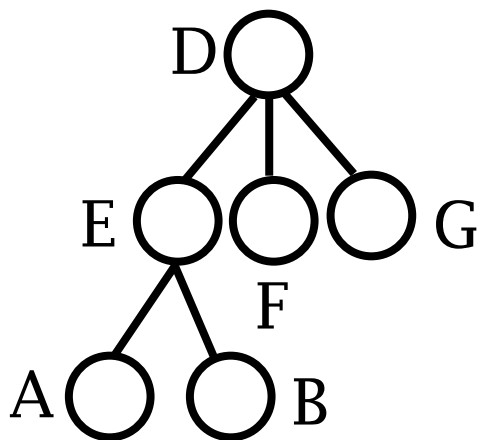
思考：下面的算法是否能遍历森林？

```
template <class T>
void Traverse(TreeNode <T> * rt) {
    if (rt==NULL) return;
    Visit(rt);
    TreeNode * temp = rt-> LeftMostChild();
    while (temp != NULL) {
        Traverse(temp);
        temp = temp->RightSibling();
    }
}
```



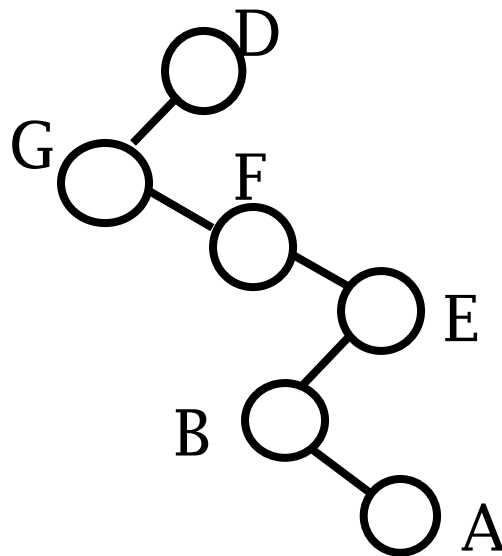
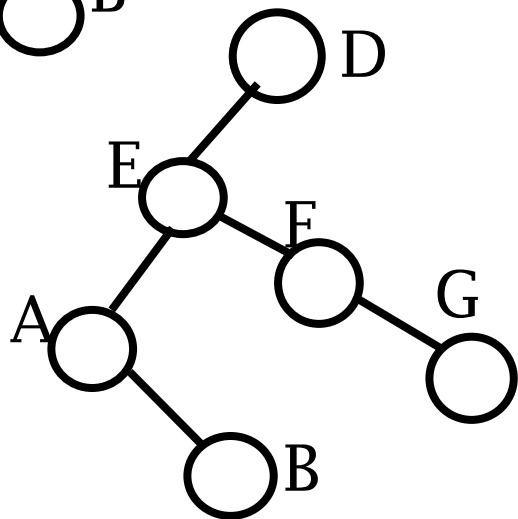
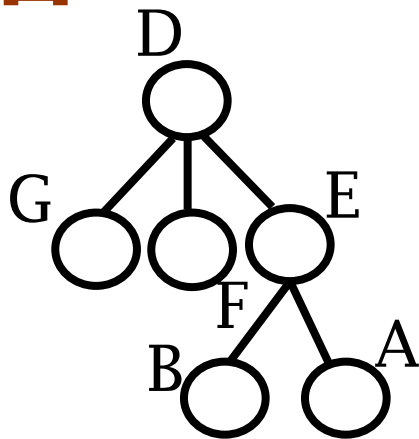
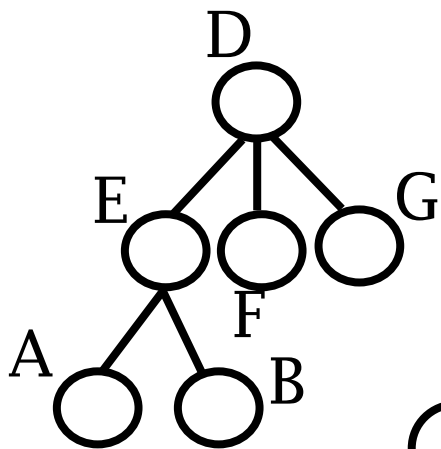
思考：灵活应用遍历框架

例：森林镜面映射



6.2 树的链式存储结构

映射后





思考：删除以subroot为根的子树

请注意待删除的子树是否为空、subroot有无父指针等情况的判断。

考虑删除以后各项相关链接的修改顺序。



数据结构与算法

谢谢聆听

国家精品课“数据结构与算法”

<http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/>

张铭，王腾蛟，赵海燕

高等教育出版社，2008.6。“十一五”国家级规划教材