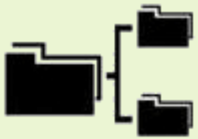
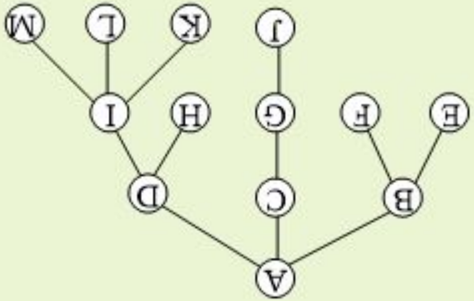
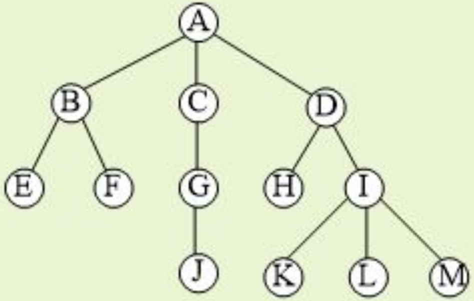


本节主题：

树的基本概念

好大一棵树





树的形式化定义

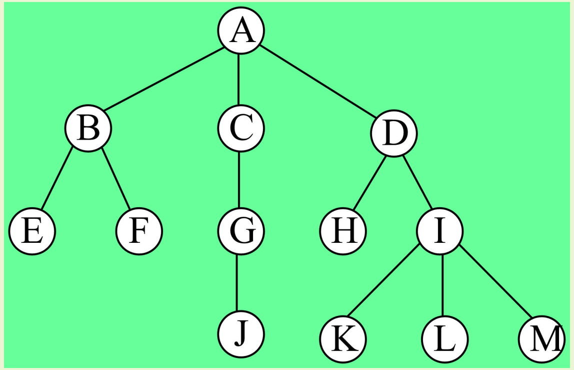
树：T={D,R}

D={A,B,C,D,E,F,G,H,I,J,K,L,M}

R={r}

r={<A,B>,<A,C>,<A,D>,<B,E>,<B,F>,
<C,G>,<D,H>,<D,I>,<G,J>,<I,K>,
<I,L>,<I,M>}

- D是包含n个节点的有穷集合（n≥0）
  - 当n=0时为空树
- 当n>0时，关系R满足以下一对多条件
  - 有且仅有一个节点d_0∈D没有前驱节点，节点d_0称作树的**根节点**
  - 除节点d_0外，D中的每个节点有且仅有一个**前驱节点**
  - D中每个节点可以有零个或多个**后继节点**



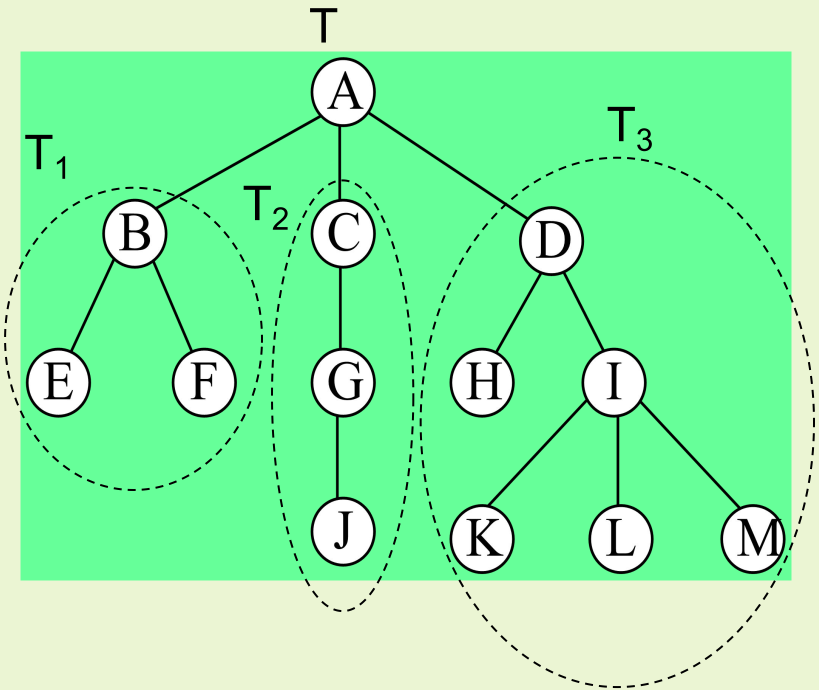
有趣的术语

- 每个节点的后继，被称作该节点的**孩子节点**（或**子女节点**）。相应地，该节点被称作孩子节点的**双亲节点**（或**父母节点**）。
- 具有同一双亲的孩子节点互为**兄弟节点**
- 每个节点的所有子树中的节点称为**子孙节点**。
- 从树根节点到达节点的路径上经过的所有节点被称作该节点的**祖先节点**。

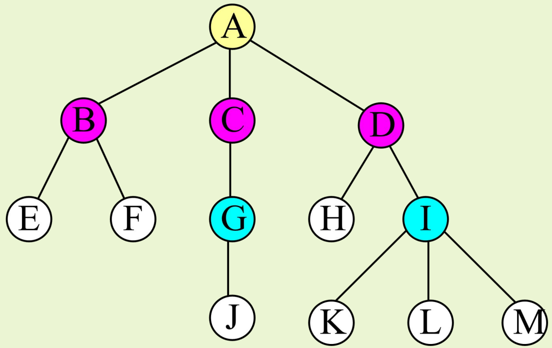
树的递归定义

树是由n（n≥0）个节点组成的有限集合（记为T）

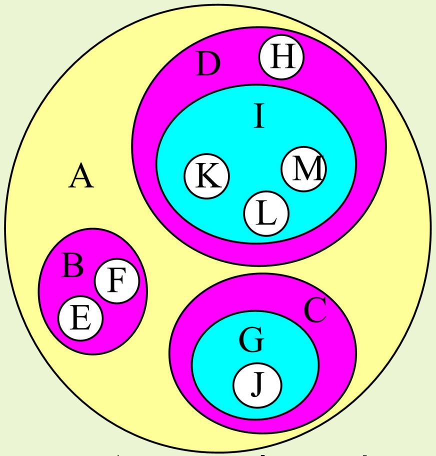
- 如果n=0，它是一棵空树，这是树的特例；
- 如果n>0
  - 这n个节点中存在（有仅存在）一个节点作为树的根节点，简称为根节点（root）
  - 其余节点可分为m（m>0）个互不相交的有限集$T_1,T_2,\ldots,T_m$，其中每一棵子集本身又是一棵符合本定义的树，称为根root的子树。



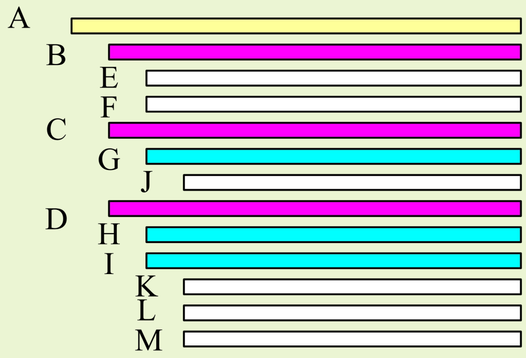
树的表示法



树形表示法



文氏图表示法



凹入表示法

A(B(E,F),C(G(J)),D(H,I(K,L,M)))

括号表示法

ADT

树的运算主要分为三大类：

- 第一类，寻找满足某种特定关系的节点，如寻找双亲节点；
- 第二类，插入或删除某个节点，如在树的当前节点上插入一个新节点或删除当前节点的第i个孩子节点等；
- 第三类，遍历树中每个节点。

ADT Tree

{

数据对象：

D = {a_i | a_i∈ElemType, i=1,2,…,n, n≧0 } //ElemType为类型标识符

数据关系：

R = {<a_i,a_j> | a_i, a_j∈D, i=1,2,…,n, j=1,2,…,n，其中每个元素只有一个前驱节点，可以有零个或多个后继节点，有且仅有一个元素（根节点）没有前驱节点 }

数据操作：

（1）初始化树**InitTree(&t)**：构造一个空的树t

（2）销毁树**DestroyTree(&t)**：释放树t占用的内存空间

（3）求双亲节点**Parent(t)**：求t所指节点的双亲结点

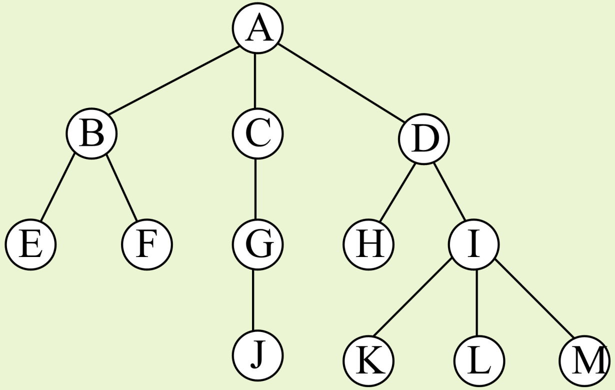
（4）求子孙节点**Sons(t)**：求t所指节点的子孙节点

… …

}

遍历操作

- 树的遍历
  - 按照一定次序访问树中所有节点，并且每个节点仅被**访问**一次的过程。
  - 遍历是最基本的运算，是树中所有其他运算的基础。
- 树三种遍历
  - **先根遍历**：若树不空，则先访问根节点，然后依次先根遍历各棵子树。
  - **后根遍历**：若树不空，则先依次后根遍历各棵子树，然后访问根节点。
  - **层次遍历**：若树不空，则自上而下自左至右访问树中每个节点。



ABEFCGJDHIKLM

EFBJGCHKLMIDA

ABCDEFGHIJKLM

思考题

- 树的逻辑结构定义？
- 适合表示什么类型的数据？