树（Tree）

定义

- 树是一种非线性结构，树与线性表的逻辑关系不同。
- 结点（数据元素的别名）
- 集合（线性表对应的是序列，树对应的是集合）
- 树：是n(n≥0)个结点（元素）构成的有限集合
  - 空树：当n=0时，称这棵树为空树
  - 在一棵非空树中
    - 有且只有一个特殊的结点称为树的根结点 根结点没有前驱结点
    - 当n > 1，除根结点之外的其余数据元素能分成m(m>0)个互不相交的集合T1,T2,...,Tm，其中每一个集合Ti(1≤i≤m)本身又是一棵树。树T1,T2,...,Tm 称为这个根结点的子树
- 森林 —— m(m≥0)棵不相交的树的集合称为森林。自然界中树和森林是不同的概念，但在数据结构中，树和森林只有很小的差别：任何一棵树，删去根结点就变成了森林。
- **结点的度** —— 树中结点所拥有的子树的个数
- **树的度** —— 树中各结点度的最大值。 —— 二叉树的度为2。
- **叶子结点（终端结点）** —— 度为0的结点
- **分枝结点（非终端结点）** —— 度不为0的结点
- **孩子** —— 树中一个结点的子树的根结点
- **双亲** —— 树中一个孩子结点的上层结点（唯一）
- **兄弟** —— 具有同一个双亲的孩子结点互成为兄弟
- **堂兄弟** —— 双亲在同一层，且双亲不同的结点互为堂兄弟
- **子孙** —— 一个结点的所有子树中的结点即之为该结点的子孙节点
- **祖先** —— 从根结点到该结点所经分支上的所有结点
- **层数** —— 根结点的层数规定为1，其余结点的层数等于它的双亲结点的层数+1
- **深度** —— 所有结点的最大层次
- **有序树** —— 如果一棵树中结点的各子树从左到右是有次序的，即若交换了某结点各子树的相对位置则构成不同的树，则这棵树为有序树
- **无序树** —— 反之，则称为无序树
- 图示

- 注意事项
  - 递归方法是树结构算法的基本特点
  - 树的根节点唯一
  - n>0：子树的个数从定义上没有限制，可以很大。
  - **子集互不相交** —— 某结点不能同时属于两个子集 两个子集的结点之间不能有关系。

图例

1. Priority 1
2. Priority 2
3. Priority 4
4. Laugh
5. Definition
6. Question

树的基本操作

遍历（递归思想）

先序遍历

```
void PreOrderTree(Tree T,void Visit(TElemType))
{
    if(T)
    {
        Visit(T->data);
        for(int i=0;i<MAX_SON_SIZE;i++)
            PreOrderTree(T->child[i],Visit);
    }// PreOrderTree
```

中序遍历

后序遍历

```
void PostOrderTree(Tree T,void Visit(TElemType));
{
    if(T)
    {
        for(int i=0;i<MAX_SON_SIZE;i++)
            PostOrderTree(T->child[i],Visit);
        Visit(T->data);
    }// PostOrderTree
```

层序遍历（使用队列）

```
typedef CTree ElemType;
void LevelOrderTree(CTree T,void Visit(TElemType));
{
    ElemType Q;
    int front,rear;
    ElemType Queue[MAX_MAX_TREE_SIZE_10];
    int(*p)(ElemType,int);
    ElemType temp=(ElemType*)malloc(sizeof(ElemType));
    int i=0;
    while(Q=Queue[front++])
    {
        Visit(Q->data);
        for(int i=0;i<MAX_SON_SIZE;i++)
        if(p=(int(*)(ElemType,int))Q->child[i],i)
        {
            temp=(ElemType*)malloc(sizeof(ElemType));
            *temp=*p;
            Queue[rear++]=temp;
        }
    }
    free(temp);
}// LevelOrderTree
```

创建树

```
void CreateTree(Tree &T,int(*p)(ElemType,int));
{
    int i=0;
    int n=0;
    int(*q)(ElemType,int);
    int(*temp)(ElemType,int);
    int Queue[MAX_MAX_TREE_SIZE_10];
    int front=0,rear=0;
    while(scanf("%d",&n),n!=-1)
    {
        if(n==0)
        {
            T=NULL;
            return;
        }
        if(!T)
        {
            T=(Tree*)malloc(sizeof(Tree));
            Queue[rear++]=n;
            while(Queue[front])
            {
                n=Queue[front++];
                q=(int(*)(ElemType,int))malloc(sizeof(int));
                *q=n;
                temp=(int(*)(ElemType,int))malloc(sizeof(int));
                *temp=*p;
                for(i=0;i<MAX_SON_SIZE;i++)
                    if(!(*p(n,i)))
                    {
                        (*p(n,i))=*temp;
                        Queue[rear++]=*temp;
                        break;
                    }
                free(temp);
                free(q);
                p=(int(*)(ElemType,int))*p;
                temp=(int(*)(ElemType,int))*temp;
                i=0;
                while(!(*p(n,i)))
                    i++;
                if(i==MAX_SON_SIZE)
                    break;
                else
                    i=0;
                while(!(*temp(n,i)))
                    i++;
                if(i==MAX_SON_SIZE)
                    break;
                else
                    
```