1.树的概念

1.1 树的定义

- 自由树: $T_f = (V, E)$ $(V = v_{21}, v_2, \ldots v_n \;\; E = (v_i, v_j)|v_i, v_j \in V, 1 \le i, j \le n)$ ，其中，集合E的元素个数为 $n-1$, (v_i, v_j)称为边或分支, T_f为连通图

- 有根树: $T = \{ \phi, \qquad n = 0 \; r, T1, T2, T3, \ldots, T_m, \qquad n > 0$
  $n = 0$时称为空树，$T_1, T_2, \ldots T_m$ 称为子树
  根节点没有前驱，除此之外每个节点有且只有一个前驱；所有节点有零个或多个后继

- 目录结构、集合文氏图、凹入表表示、广义表表示

1.2 常见术语

- 结点：包含数据项和其他结点的分支
- 度：拥有子树的个数,树的度是结点的最大度
- 叶结点：度为0的点，终端结点
- 分支节点：除叶结点以外的点，非终端节点
- 子女节点：若结点x有子树，子树的根结点即x的子女
- 父结点：结点x为其子女结点的父结点
- 兄弟结点：同一父结点的子女护卫兄弟
- 祖先结点：对结点x，根结点到这个结点唯一路径上的任意结点
- 子孙结点：子女结点的子女
- 层次：根到该结点的子树个数，记根结点为层次为1，子结点层次为父结点加1
- 深度：最远结点的层次，空树为0，自顶向下
- 高度：叶结点高度为1，其父结点的高度为最高子女高度+1，树的高度为根结点高度，自底向上，数值上与深度相等。
- 有序树、无序树：各棵子树能够交换，例如有序树中$T_1, T_2 \ldots$被称为第一棵子树、第二棵子树...
- 森林：$m \ge 0$棵互不相交的树，树→森林：删去一棵非空树的根结点（空森林也是森林） 森林→树：增加一个结点，让每一棵树的根结点都称为这个结点的子女结点

1.3 树的性质

- 结点数等于度数和加1
- 度为m的树第i层至多有m^{i-1}个结点
- 高度为h的m叉树（度为m)至多有 $\sum_{i=1}^{h} m^{i-1} = \frac{(m^h - 1)}{(m-1)}$
- 设度为m的树（m叉树）结点个数为n,由上一个性质，我们推出：

$$n \le, \frac{(m^h - 1)}{m-1} \qquad h \ge \lceil \log_m(n(m-1)+1) \rceil$$

2.二叉树

2.1 二叉树定义

$$T = \{ \phi \qquad n = 0\ r,,T_l,,T_r \qquad n \geq 1$$

树的度至多为2（即最多两个子女结点），左右子树能交换（有序的）

2.2 二叉树的性质

- 第i层至多2^{i-1}个结点
- 高为k至多2^k-1个结点，至少k个结点（每层一个）
- 考虑两个等式:

  $$n = n_0 + n_1 + n_2\ e = n - 1 = n_1 + 2 * n_2$$

  故有:$n_0 = n_2 + 1$
- 满二叉树：每一层都达到最大结点个数；完全二 叉树：每一个结点都与具有相同高度的满二叉 树对应（上面$k-1$层是满的，仅第k层从右向左却若干个）
- n个结点的完全二叉树最小深度为$\lceil \log(n+1) \rceil$
- 对于完全二叉树，编号为$1,2,3...n$,结点 i 的父结点编号为 $\lfloor \frac{n}{2} \rfloor$,结点 i 的左子女结点为 $2*i*
  ,右子女结点为*2*i+1，(检查结点是否超过n),深度\lfloor\log(i)\rfloor + 1$

3.二叉树存储

3.1 二叉树数组表示

- 适用场景：二叉树大小和形态不发生剧烈动态变化
- 将二叉树存储在一组连续的存储单元（即数组内），要体现树的逻辑结构。
- 完全二叉树数组存储：自顶向下，自左向右顺序编号为 $1,2,3,4...,n$,按照这个序列把完全二叉树放入一维数组中（根结点在索引为1处）。**这种方式最简单、最省存储空间**
- 一般二叉树存储：仿照完全二叉树编号，遇到空子树时假定有子树编号。这样的做法会浪费存储空间（eg.只有右子树）

3.2 二叉树的链表表示

- 适用一般二叉树，变化剧烈
- 二叉树的每一个结点可以有两个分支，分别指向左、右子树，因此至少有三个域，分别存放结点的数据、左右子女结点指针。**很方便的找到子女结点，但找到父结点很困难**
- 为了找到父结点，可以再加一个父指针域，称为三叉链表
- 对于 n 个结点的二叉链表中，共有 $2*n$ 个指针域，由于二叉树中共有 $n-1$ 条边，故**二叉链表中有 $n+1$ 个空指针域**，同理，三叉链表中有 $n+2$ 个空指针域（根结点的父结点域为空）
- 这两种链表形式都可以是静态链表结构，即把链表存放在一个一维数组中，每个一维数组元素是一个结点，包括三个域：数组域、左子女域、右子女域，还可以增加父指针域，指针域指向数组中的下标：

```
          A
         /
        B
       / \
      C   D
```

```
        /
       E
```

index	data	Parent	LeftChild	RightChild
0	A	-1(root)	1(B)	-1(NULL)
1	B	0(A)	2(C)	3(D)
2	C	1(B)	-1(NULL)	-1(NULL)
3	D	1(B)	4(E)	-1(NULL)
4	E	3(D)	-1(NULL)	-1(NULL)

4.二叉树的遍历及应用

概述

二叉树遍历是指遵从某种次序，遍历二叉树中所有的结点，使得每个结点访问且只访问一次，且不破坏树的数据结构,产生的结构是一个线性队列。

考虑自身、左右三个结点的顺序，总共的遍历方式有 $A_3^3=6$ 种，规定先左后右，共有三种方式。常见的遍历方式有先序（NLR）、中序（LNR）、后序（LRN）三种。

三种遍历算法

- 递归访问

```
/*先序遍历*/
void PreOrder(BiTNode* T){
  if(T!=NULL){
    visit(T);
    PreOrder(T->lchild);
    PreOrder(T->rchild);
  }
}
/*中序遍历*/
void InOrder(BitNode* T){
  if(T!=NULL){
    InOrder(T->lchild);
    visit(T);
    InOrder(T->rchild);
  }
}
/*后序遍历*/
void PostOrder(BitNode* T){
  if(T! =NULL){
    PostOrder(T->lchild);
```

```
    PostOrder(T->rchild);
    visit(T);
  }
}
```

- 先序非递归遍历
-