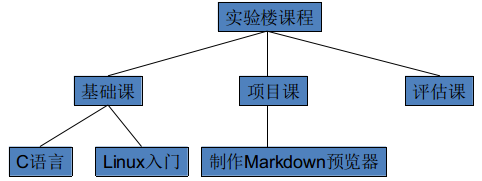
# 非线性结构-树

## 实验简介

前面两章我们讲解了数据结构中的线性结构--线性表、栈和队列，这章开始以及下一章我们将讲解非线性结构树和图。

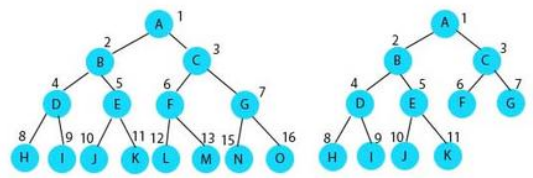
## 一、树

什么是树呢？树很好地反应了一种层次结构，例如下图，这就是一种树形结构，它有很多结点组成，最上面的实验楼课程结点称为树的**根**，结点拥有的直接子节点数称为**结点的度**，度为0的结点称为**叶子**，例如C语言、评估课这些结点，而**树的度**是所有结点的度中的最大值，这颗树的度就是3，一个结点的直接子结点称为它的**孩子**，项目课结点的孩子就是制作Markdown预览器结点，相应地项目课结点就是制作Markdown预览器结点的**双亲**，相同双亲的孩子结点互称为**兄弟**，例如C语言结点和Linux入门结点，一个结点的**祖先**是从根到该结点所经过的所有结点，C语言结点的祖先就是基础课和实验楼课程结点，一个结点下的所有结点称为该结点的**子孙**，例如实验楼课程下的所有结点都是它的子孙。树有层次之分，根记为第一层，依次类推，例如这棵树的最大层次就是3，也称为该树的**深度**，双亲在同一层的结点互称为**堂兄弟**，例如Linux入门结点和制作Markdown预览器结点。



## 二、二叉树

上面介绍了树，接下来我们介绍一种很常用的树结构--二叉树，它的特点是一个结点的直接子节点最多只能有两个，并且有左右之分。在二叉树中有种常见的称为完全二叉树的结构，它的特点是除最后一层外每一层的结点数为2i-1，最后一层的结点数若不满足2i-1，那么最后一层的结点是自左向右排列的，如下图。



二叉树也有顺序存储结构和链式存储结构两种，这里我们就讲下链式存储结构的代码实现（主要操作）：

#include <stdio.h>#include <stdlib.h>

#define TRUE 1#define FALSE 0#define OVERFLOW -2#define OK 1#define ERROR 0

typedef int Status;

typedef int TElemType;

/\*

\* 存储结构

\*/

typedef struct BiTNode

{

TElemType data; //数据

struct BiTNode \*lchild, \*rchild;

}BiTNode, \*BiTree;

/\*

\* 创建二叉树,输入0表示创建空树

\*/Status CreateBiTree(BiTree \*T){

TElemType e;

scanf("%d", &e);

if (e == 0)

{

\*T = NULL;

}

else

{

\*T = (BiTree) malloc(sizeof(BiTNode));

if (!T)

{

exit(OVERFLOW);

}

(\*T)->data = e;

CreateBiTree(&(\*T)->lchild); //创建左子树

CreateBiTree(&(\*T)->rchild); //创建右子树

}

return OK;

}

/\*

\* 访问元素

\*/void visit(TElemType e){

printf("%d ", e);

}

/\*

\* 先序遍历二叉树：指先访问根，然后访问孩子的遍历方式

\*/Status PreOrderTraverse(BiTree T, void (\*visit)(TElemType)){

if (T)

{

visit(T->data);

PreOrderTraverse(T->lchild, visit);

PreOrderTraverse(T->rchild, visit);

}

}

/\*

\* 中序遍历二叉树：指先访问左（右）孩子，然后访问根，最后访问右（左）孩子的遍历方式

\*/Status InOrderTraverse(BiTree T, void (\*visit)(TElemType)){

if (T)

{

InOrderTraverse(T->lchild, visit);

visit(T->data);

InOrderTraverse(T->rchild, visit);

}

}

/\*

\* 后序遍历二叉树：指先访问孩子，然后访问根的遍历方式

\*/Status PostOrderTraverse(BiTree T, void (\*visit)(TElemType)){

if (T)

{

PostOrderTraverse(T->lchild, visit);

PostOrderTraverse(T->rchild, visit);

visit(T->data);

}

}

int main(){

BiTree T;

printf("创建树，输入0为空树：\n");

CreateBiTree(&T);

printf("先序遍历：");

PreOrderTraverse(T, \*visit);

printf("\n中序遍历：");

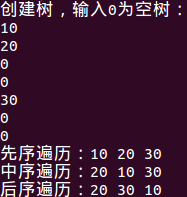
InOrderTraverse(T, \*visit);

printf("\n后序遍历：");

PostOrderTraverse(T, \*visit);

printf("\n");

}



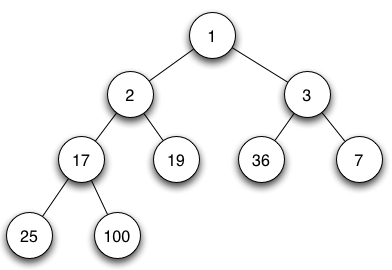
上面我们讲了二叉树的一些主要操作，其实它的操作远不止这些，例如你可以试试把遍历改为非递归实现、求树的深度等等一些操作。除了上面实现的基本二叉树之外，还有一种线索二叉树，它其实就是用结点空的指针域来指向它的前驱或者后继结点，不浪费空的指针域，如果你想深入了解，可以查查资料。

## 三、堆

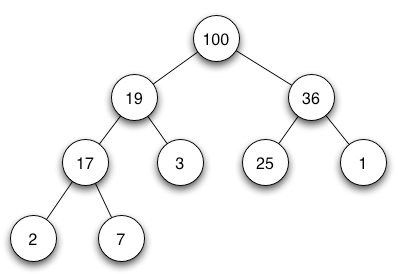
堆是一种经过排序的完全二叉树，其中任一非叶子节点的值均不大于（或不小于）其左孩子和右孩子节点的值。

最大堆和最小堆是二叉堆的两种形式。 最大堆：根结点的键值是所有堆结点键值中最大者。 最小堆：根结点的键值是所有堆结点键值中最小者。 而最大-最小堆集结了最大堆和最小堆的优点，这也是其名字的由来。 最大-最小堆是最大层和最小层交替出现的二叉树，即最大层结点的儿子属于最小层，最小层结点的儿子属于最大层。 以最大（小）层结点为根结点的子树保有最大（小）堆性质：根结点的键值为该子树结点键值中最大（小）项。

最小堆

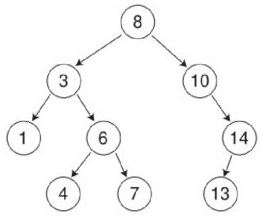


最大堆



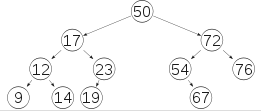
## 四、二叉排序树

二叉排序树又称二叉查找树，亦称二叉搜索树，如下图所示，它主要用于查找。 它或者是一棵空树；或者是具有下列性质的二叉树： （1）若左子树不空，则左子树上所有结点的值均小于它的根结点的值； （2）若右子树不空，则右子树上所有结点的值均大于它的根结点的值； （3）左、右子树也分别为二叉排序树；



## 五、平衡二叉树

平衡二叉树又被称为AVL树，且具有以下性质：它是一棵空树或它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡二叉树，如下图，由它可以生成平衡二叉搜索树，查找效率会更高。构造与调整方法平衡二叉树的常用算法有红黑树、AVL、Treap等。最小二叉平衡树的节点的公式如下F(n)=F(n-1)+F(n-2)+1这个类似于一个递归的数列，可以参考Fibonacci数列，1是根节点，F(n-1)是左子树的节点数量，F(n-2)是右子树的节点数量。



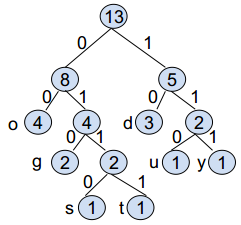
## 六、哈夫曼树

哈夫曼树也称最优二叉树，它是带权路径长度最小的二叉树。下面我就通过一个例子让大家快速地明白，相信大家都看过抗日电视剧，打仗的时候，前线要与后方指挥部取得联系通常都会使用电报，那么电报编码后的长度当然是越短越好，但同时翻译电报时又不能造成歧义，这时候就可以使用哈夫曼树来编码，那么怎么实现呢？

哈夫曼树的构造步骤如下：

假设有n个权值，则构造出的哈夫曼树有n个叶子结点。 n个权值分别设为 w1、w2、…、wn，则哈夫曼树的构造规则为： (1) 将w1、w2、…、wn看成是有n 棵树的集合(每棵树仅有一个结点)； (2) 在集合中选出两个根结点的权值最小的树合并，作为一棵新树的左、右子树，且新树的根结点权值为其左、右子树根结点权值之和； (3)从集合中删除选取的两棵树，并将新树加入集合； (4)重复(2)、(3)步，直到集合中只剩一棵树为止，该树即为所求得的哈夫曼树。

比如需要发送“goodgoodstudy”，我们先计算每个字母出现的次数即权值，g：2、o：4、d：3、s：1、t：1、u：1、y：1，然后通过哈夫曼树的构造规则构造出哈夫曼树，如下图。



通过构造哈夫曼树我们就能得到每个字母的编码，g：010、o：00、d：10、s：0110、t：0111、u：110、y：111，这就能使编码总长度最小，此种编码就是著名的哈夫曼编码。

## 七、小结

本章我们讲了非线性结构树、二叉树以及哈夫曼树（最优二叉树），树结构体现的是一种层次结构，二叉树结点的直接子节点最多只能有两个，可以解决表达式求值等问题。堆是一种经过排序的完全二叉树，其中任一非叶子节点的值均不大于（或不小于）其左孩子和右孩子节点的值，堆有最大堆、最小堆和最大-最小堆。二叉搜索树和平衡二叉树主要用于查找，还有B-树和B+树也是用于查找，它们主要应用于文件系统。哈夫曼树是一种带权路径长度最小的树，哈夫曼编码就是由它而得名。

## 作业

1、表达式求值问题，例如输入表达式4+(3-1)-6/2+5\*7，结果是38。提示：可以使用二叉树以及它的遍历。

2、输入一个二叉树，输出其镜像。