第二周

一、跟着之前的学习进程，学习了二叉树的基本操作及实现。

基本操作以及算法的实现如下：

（1）Initiate(bt)：建立一棵空二叉树

int Initiate(BiTree \* bt)

{/\*初始化建立二叉树\*bt的头结点\*/

if((\*bt = (BiTNode \* )malloc(sizeof(BiTNode)))==NULL)

return 0;

\*bt -> lchild = NULL;

\*bt -> rchild = NULL;

return 1;

}

（2）Creat(x,lbt,rbt)：生成一棵以x为根结点的数据域信息，以二叉树lbt和rbt为左子树和右子树的二叉树

BiTree Creat(elemtype x, BiTree lbt, BiTree rbt)

{/\*成一棵以x为根结点的数据域值以二叉树lbt和rbt为左子树和右子树的二叉树\*/

BiTree p;

if((p = (BiTNode \* )malloc(sizeof(BiTNode)))==NULL) return NULL;

p -> data = x;

p -> lchild = lbt;

p -> rchild = rbt;

return p;

}

（3)InsertL(bt,x,parent)：将数据域信息为x的结点插入到二叉树bt中作为结点parent的左孩子结点。如果结点parent原来有左孩子结点，则将结点parent原来的左孩子结点作为结点x的左孩子结点。

BiTree InsertL(BiTree bt, elemtype x, BiTree parent)

{/\*在二叉树bt的结点parent的左子树插入结点数据元素x\*/

BiTree p;

if(parent==NULL)

{ printf("\n插入出错");

return NULL;

}

if((p = (BiTNode \* )malloc(sizeof(BiTNode)))==NULL) return NULL;

p -> data = x;

p -> lchild = NULL;

p -> rchild = NULL;

return p;

if(parent -> lchild == NULL) parent -> lchild = p;

else

{ p -> lchild = parent -> lchild;

parent -> lchild = p;

}

return bt;

}

(4)InsertR(bt,x,parent)：将数据域信息为x的结点插入到二叉树bt中作为结点parent的右孩子结点。如果结点parent原来有左孩子结点，则将结点parent原来的右孩子结点作为结点x的右孩子结点。

功能类同于（3）

（5）DeleteL(bt,parent)：在二叉树bt中删除结点parent的左子树

BiTree DeleteL(BiTree bt, BiTree parent)

{/\*在二叉树bt中删除结点parent的左子树\*/

BiTree p;

if(parent==NULL||p -> lchild = NULL)

{ printf("\n删除出错");

return NULL;

}

p = parent -> lchild;

parent -> lchild = NULL;

free(p);

return bt;

}

（6）DeleteR(bt,parent)：在二叉树bt中删除结点parent的右子树

功能类同于（5）

二、遍历二叉树和线索二叉树

1.深度优先遍历二叉树：先序遍历、中序遍历、后序遍历

2.广度优先遍历二叉树：即层次遍历

3.二叉树遍历的非递归实现

三、线索二叉树

线索需遍历二叉树来找到某些结点的前驱结点以及后继结点。为每个结点增设两个标志位域ltag和rtag，令：

ltag = 0；lchild指向结点的左孩子

ltag = 1；lchild指向结点的前驱结点

rtag = 0；rchild指向结点的右孩子

rtag = 1；rchild指向结点的后继结点

四、最优二叉树

哈夫曼树的基本概念以及构造算法