哈尔滨工业大学(深圳)2023 年春《数据结构》

第二次作业 树型结构 参考答案

学号		姓名		成绩	

1、概念题

1.1 假设一棵二叉树采用同高度完全二叉树顺序存储,如下表所示：

data		e	a	f		d		g			c	j			h	i					b
下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

请回答下列问题：

①画出该二叉树；

②写出二叉树的先序、中序和后序遍历结果；

③写出结点 c 的双亲结点和左、右孩子结点；

④画出此二叉树还原成的森林。

【参考答案】

①二叉树表示如图 2-1 所示

②先序序列为：eadcbjfghi

中序序列为：abcdjefhgi

后序序列为：bcjdahigfe

③结点 c 的双亲结点是 d,左孩子为 b,无右孩子；

④该二叉树对应的森林如图 2-2 所示。

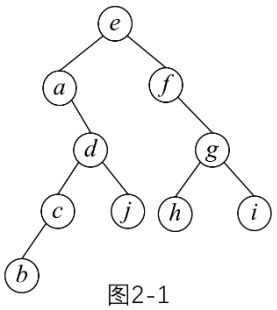


图2-1

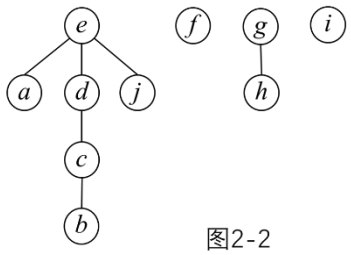


图2-2

1.2 对于二叉树 T 的两个结点 n_1 和 n_2，我们应该选择二叉树 T 结点的前序、中序和后序中哪两个序列来判断结点 n_1 必定是结点 n_2 的祖先?

试给出判断的方法。(不需证明判断方法的正确性)

【参考答案】

采用前序和后序两个序列来判断二叉树上结点 n_1 必定是结点 n_2 的祖先。在前序序列中某结点的祖先都排在其前。

若结点 n_1 是 n_2 的祖先，则 n_1 必定在问之前。而在后序序列中，某结点的祖

先排在其后，即若结点 n_1 是 n_2 的祖先，则 n_1 必在 n_2 之后。根据这条规则来判断若结点 n_1 在前序序列中在 n_2 之前，在后序序列中又在 n_2 之后，则它必是结点叫的祖先。

1.3 证明在任意非空二叉树 T 上，分支结点数等于叶子结点数的条件是二叉树 T 上度为 1 的结点只有 1 个。

【参考证明】

分支结点数为 n_1+n_2，叶结点数为 n_0

由分支结点数等于叶子结点数，即 $n_0-n_1+n_2$

根据二叉树性质 3， $n_0=n_2+1$

可得 $n_1=1$ 。

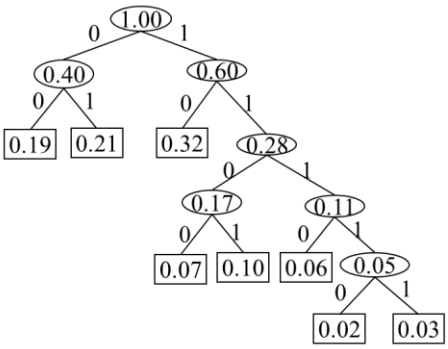
1.4 假设用于通信的电文仅由 8 个字母组成，字母在电文中出现的频率分别为 0.07,0.19,0.02,0.06,0.32,0.03,0.21,0.10。

（1）试为这 8 个字母设计哈夫曼编码；

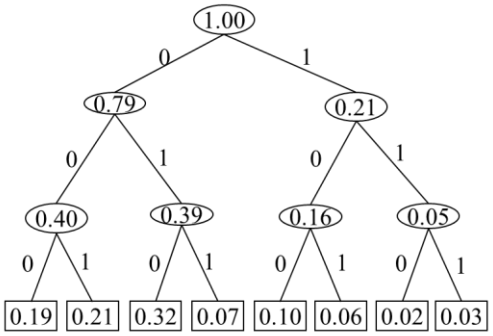
（2）试设计另一种由二进制表示的等长编码方案；

（3）对于上述实例，比较（1）和（2）两种方案的优缺点。

【参考答案】



方案一：哈夫编码树



方案二：等长二进制编码

方案比较：

哈夫曼编码

字母编号	编码	频率
1	1100	0.07
2	00	0.19
3	11110	0.02
4	1110	0.06
5	10	0.32
6	11111	0.03
7	01	0.21
8	1101	0.10

等长二进制编码

字母编号	编码	频率
1	011	0.07
2	000	0.19
3	110	0.02
4	101	0.06
5	010	0.32
6	111	0.03
7	001	0.21
8	100	0.10

方案一的 *WPL*=2(0.19+0.21+0.32)+4(0.07+0.06+0.05+0.10)+5(0.02+0.03)=2.61；

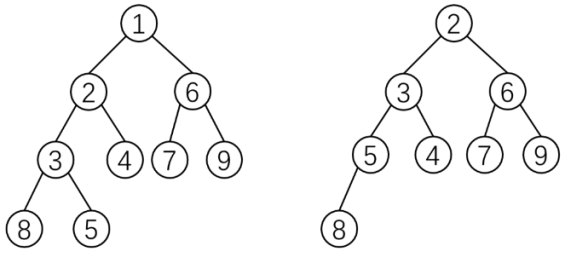
方案二的 *WPL*=3(0.19+0.21+0.32+0.07+0.06+0.05+0.10+0.03+0.02)=3.0；

所以：哈夫曼编码**优于**等长二进制编码。

1.5 (a)Draw the binary min heap that results from inserting 8, 7, 3, 2, 4, 6, 9, 5, 1 in that order into an initially empty binary min heap. You do not need to show the array representation of the heap. You are only required to show the final tree.

(b) Draw the result of one deletemin call on your heap drawn at the end of part (a).

【参考答案】



2、算法设计

针对本部分的每一道题，要求：

（1） 给出算法的基本设计思想；

（2） 根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释；

（3） 说明你所设计算法的时间复杂度和空间复杂度。

2.1 已知一棵完全二叉树按顺序方式存储在数组 int A[1..n]中。设计算法，求出下标分别为 *i* 和 *j*（*i*<=*n*，*j*<=*n*）的两个结点的最近的公共祖先结点的位置和值。

【参考答案】

(1) 算法设计思想

按完全二叉树的格式存储。利用完全二叉树双亲结点与孩子结点编号间的关系，求下标为 *i* 和 *j* 的两结点的双亲，双亲的双亲，……，等等，直至找到最近的公共祖先。

(2) 算法描述

```
void   Ancestor( int A[ ]，int n，int i，int j )
{   while( i! =j)
        if ( i > j)
            i = i/2；          //下标为 i 的结点的双亲结点的下标
        else
            j = j/2 ；          //下标为 j 的结点的双亲结点的下标
    printf("所查结点的最近公共祖先的下标是%d，值是%d\n" ，i，A[i])；
```

```
}// Ancestor
```

(3) 算法分析

时间复杂度 T(n)=O($\log_2 n$)，空间复杂度 S(n)=O(1)。

2.2 假设二叉树 bt 采用二叉链表存储，在二叉树 bt 中查找值为 x 的结点，试编写算法打印值为 x 的结点的所有祖先，假设值为 x 的结点不多于一个。

【参考答案】

（1）算法设计思想

后序遍历最后访问根结点，当访问到值为 x 的结点时，栈中所有元素均为该结点的祖先。

（2）算法描述

```
void Search (BiTree bt，elemType x)
{  //在二叉树 bt 中，查找值为 x 的结点，并打印其所有祖先
   STACK s[ ] ；       //假设核容量足够大
   top=0 ；
   while ( bt! = null || top>0)
   {  while ( bt! = n1lll && bt->data! = x)
      {  s[++top].t = bt;
         s[top].tag =0;
         bt=bt->lehild;
      }  //结点入枝，沿左分支向下
      if( hb->data = = x)
     {  printf("所查结点的所有祖先结点的值为：\n") ；       //找到 x
        for (i=1 ；  i <= top；  i + +)  printf(s[i].t->data) ；
        return;
      } //输出祖先值后结束
      while(top! =0&&s[top].tag = = 1)  top-- ；      //退栈(空遍历)
      if(top!=0)
      {  s[top].tag = 1 ；
         bt=s[top].t->rehild;
      }  //沿右分支向下遍历
   }  // while ( bt! = null|| top>0)
}  //Search
```

（3）算法分析

因为查找的过程就是后序遍历的过程，使用的栈的深度不超过树的深度，算法复杂度为 T(n)=O($\log_2 n$),空间复杂度为栈的容量 S(n)=(n).

2.3 假设以顺序表 ST 表示一棵完全二叉树，ST.data[ST.last]中存放二叉树中各结

点的数据元素。试设计算法由此顺序存储结构建立该二叉树的二叉链表 LT。

【参考答案】

（1）设计思想

根据二叉树性质五，针对完全二叉树按层序遍历顺序对应 ST 存储向量元素空间；下标变量从 0 开始到 ST.last-1 依次取出每一个元素 ST.data[i]；

设队列 Q，存储二叉链表的各结点指针，根结点首先排队；

当队列 Q 非空时，取出并删除队首元素，如果 i%2==0，则取出下一（++i）元素生成当前结点的左孩子结点，并将其排队；如果 i%2==1，则取出下一（++i）元素生成当前结点的右孩子结点，并将其排队；直到队列为空。

（2）算法实现

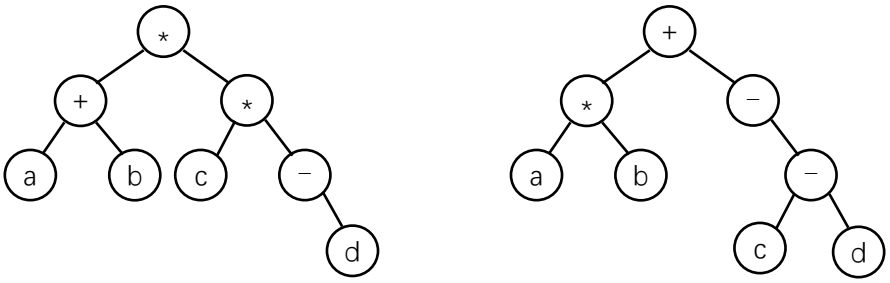
```
Status CreateCompleteBiTree(SqListElemType &ST，BiTree& LT)
{   BiTree p；
  int i=0，len；
  if(ST.Length==0)   return OK；
  p=new BiTNode；
  if(!p)   return ERROR；
  p->data=ST.data[i]；
  p->lchild=NULL；     p->rchild=NULL；
  LT=p；
  Queue Q；
  InitQueue(Q)；
  EnQueue(Q，p)；
  len=ST.last；
  while(!QueueEmpty(Q)&&i<len-1)
    {
      DeQueue(Q，p)；
      if(i<len-1 && i%2==0)
       {   p->lchild=new BiTNode；
           if(!p->lchild)   return ERROR；
           p->lchild->data=ST.data[++i]；
           p->lchild->lchild=NULL；
           p->lchild->rchild=NULL；
           EnQueue(Q，p->lchild)；
       } //if
      if(i<len-1 && i%2==1)
       {     p->rchild=new BiTNode；
             if(!p->rchild)   return ERROR；
             p->rchild->data=ST.da[++i]；
```

```
                p->rchild->lchild=NULL；
                p->rchild->rchild=NULL；
                EnQueue(Q，p->rchild)；
            } //if
        } //while
        return OK；
    } // CreateCompleteBiTree
```

（3）算法分析

T(n)=O(n)，S(n)=O(n)。

2.4 请设计一个算法，将给定的表达式树（二叉树）转换为等价的中缀表达式（通过括号反映操作符的计算次序）并输出。例如，当下列两棵表达式树作为算法的输入时：



输出的等价中缀表达式分别为 (a+b)*(c*(-d)) 和 (a*b)+(-(c-d))。

二叉树结点定义如下：

```
typedef struct node
{    char    data[10]；                // 存储操作数或操作符
     struct node *left, *right；
} BTree；
```

【参考答案】

（1）算法的基本设计思想

表达式树的中序序列加上必要的括号即为等价的中缀表达式。可以基于二叉树的中序遍历策略得到所需的表达式。

表达式树中分支结点所对应的子表达式的计算次序，由该分支结点所处的位置决定。为得到正确的中缀表达式，需要在生成遍历序列的同时，在适当位置增加必要的括号。显然，表达式的最外层（对应根结点）及操作数（对应叶结点）不需要添加括号。

（2）算法实现

```
void BtreeToE ( BTree * root )
{
      BtreeToExp ( root, 1 )；          // 根的高度为 1
```

```
}

void BtreeToExp ( BTree * root, int deep )
{    if (root == NULL)    return；
     else if (root->left == NULL && root->right == NULL) //  若为叶结点
          printf ( "%s" , root -> data )；                    //  输出操作数
     else
     {
          if ( deep > 1 )    printf ( "(" )；          //若有子表达式则加 1 层括号
          BtreeToExp( root->left, deep + 1 )；
          printf ( "%s" , root -> data )；          //  输出操作符
          BtreeToExp( root->right, deep + 1 )；
          if ( deep > 1 )    printf ( ")" )；          //若有子表达式则加 1 层括号
     }
}
```

（3）算法分析

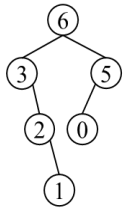
T(n)=S(n)=O(n)。

2.5　给定一个非空且无重复元素的整数数组 A，它对应的“最大二叉树”T(A)定义为：

① T(A)的根为 A 中最大值元素；

② T(A)左子树为 A 中最大值左侧部分对应的“最大二叉树”；

③ T(A)右子树为 A 中最大值右侧部分对应的“最大二叉树”。



例如：A={3, 2, 1, 6, 0, 5}对应的“最大二叉树”T (A)如右图所示。

设计一个“最大二叉树”的构建算法，并分析最好情况、最坏情况下的时间和空间复杂性。

【参考答案】

（1）算法设计思想

找到数组中的最大元素及其所在位置，以最大元素为结点生成根结点，然后分别针对最大元素左边元素和右边元素递归调用生成左子树和右子树。

（2）算法描述

```
void CreateMaxBTree ( BTREE T, int *A,    int ibegin , int jend )
{
     int i, k  ；
     if( ibegin <= jend )
     {     k = ibegin  ；
           for( i = ibegin；    i <= jend；    i++ )
                if( A[k] <    A[i] )    k = i  ；
```

```
            T = NEW Node ；
            T->data = A[k] ；
            T->lchild = T->rchild = NULL ；
            CreateMaxBTree( T->lchild, A, ibegin, k-1 ) ；
            CreateMaxBTree( T->rchild, A, k+1, jend ) ；
        }
    }
```

（3）算法分析

元素或结点个数为 n，二叉树的高度平均为 $O(\log_2 n)$, 所以 $T(n)=O(n\log_2 n)$, 空间复杂度为栈的容量(最大为 n) $S(n)=(n)$。