北京大学信息科学技术学院 2005-2006 学年

第一学期本科生期中考试试卷

考试科目：数据结构与算法 A　考试时间：2005 年 11 月 5 日

_____专业_____级_____班　考场 三教　　室

姓名_____　　学号_____　　教员：张铭、赵海燕、王腾蛟

题号	一	二	三	四	五	总分
得分						

注意事项：

1. 第一题填空答案请写在试卷的留白上，其他题目都写在空白答题纸上。若超出所留空间，请使用后面的续页。
2. 本试卷对于算法改进、算法设计都有质量要求，请尽量按照试题中的要求来写算法。否则将酌情扣分。
3. 您所写的算法应该申明算法思想，并在算法段加以恰当的注释。

诚实考试宣言：

我真诚地保证严格遵循考场纪律：

1. 诚实地用自己所掌握的知识和能力，独立地回答试卷；
2. 不夹带任何有作弊嫌疑的书本、纸条或写在其他地方的提示性资料进入考场；
3. 正式考试（包括发卷和收卷）过程中，不与考场的同学交谈或借物品，有问题则举手等待监考老师来协助解决；
4. 不与他人共享答案和解题方法：不抄袭或偷看他人的答案，也不告诉他人答案或解题方法，更不传递或故意高举试卷把答案给他人看；
5. 服从监考老师的管理；
6. 不携带试卷出考场，也不会把考题以任何方式泄露出去。

保证人：_____

不在“诚实考试宣言”之后签名的试卷，计零分或根据作弊与否的情况上报学校处理！

一、（20 分）填空

1. （3 分）下面术语_____与数据的存储结构无关（写出编号）。
   a) 顺序表 b) 链表 c) 队列 d) 循环链表
2. （7 分）下面这个双标记（0 表示有子结点、1 表示没有）的二叉树前序序列对应的森林的后根周游序列为_____。

右标记	0	1	1	0	0	1	0	1	1	1
信息	A	B	D	C	E	G	F	H	I	J
左标记	0	0	1	0	1	0	0	1	1	0

3. （4 分）对一棵完全 3 叉树，按照广度优先周游顺序给结点从左向右依次连续编号，第一个结点编号为 0。则编号为 100 的结点的父结点编号是_____。
4. （6 分）使用重量权衡合并规则与路径压缩，对下列从 0 到 15 之间的数的等价对进行归并。在初始情况下，集合中的每个元素分别在独立的等价类中。当两棵树规模同样大时，使结点数值较大的根结点作为值较小的根结点的子结点。

   (0，2) (1，2) (3，4) (3，1) (3，5) (9，11) (12，14) (3，9)
   (4, 14) (6, 7) (8,10) (8，7) (7，0) (10, 15) (10，13)

   请填写下面表格中的空白部分树的父指针表示法的数组表示。也就是所有等价对都被处理之后，所得父结点的下标值。

父结点的下标																
结点值	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
结点的下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

二、（30 分）辨析题

1. （10 分）对于 t="aababbaaabb" p = “aabb”，进行 KMP 快速模式匹配，请画出匹配过程的示意图。建议采用教材的定义，如果采用其他教材的定义请予以说明。
2. （10 分）从初始输入序列 1，2，…，n，可以利用一个栈得到输出序列 p_1，p_2，…，p_n (p_1，p_2，…，p_n 是 1， 2，…，n 的一种排列)的充分必要条件是，不存在下标 i，j，k，满足 i<j<k 同时 $P_j < P_k < P_i$.

这种说法是否正确？若正确，请给出其证明，否则，请举出一反例。

3. （10 分）假设字母集（A, B, C, D, E, F, G）以相对频率（4，5，6，7，10，12，18）出现。

（1） 请给出它们的 Huffman 编码（具体编码答案不惟一，但应该是最优编码）。

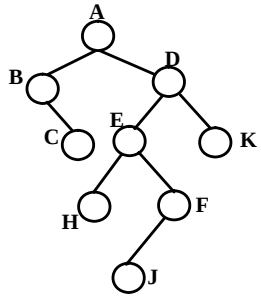
（2） 对它们进行等长编码，可以编为（000, 001, 010, 011, 100, 101, 110）。假设一个 A-G 字母组成的文件，以符合题中规定频率出现。如果其固定长度编码所得到的文件长度为 18600 位(bit)，则采用 Huffman 编码为多少位？Huffman 编码节省的空间比例为多少？请给出具体的计算过程。

三、（20 分）算法辨析题

1. （5 分）二叉树周游是对树中的每个结点访问而且只访问一次。请判断下面这个算法是否正确地周游了二叉树。

（1） 如果是，请对右图写出此算法的输出结果；

（2） 如果否，请简要说明理由，并改正（请尽量保留原算法框架）。




```
template <class T>
void BinaryTree::traverse(BinaryTreeNode<T>* root) {
    while (root!=NULL) {
        cout<<root->value();
        traverse(root->leftchild());
        traverse(root->rightchild());
    }
}
```

2. （15 分）请判断下述二叉搜索树的结点删除算法是否正确。

（1） 如果正确，请给出一个简单实例，演示算法运行步骤；

（2） 如果不正确，请指出错误之处，并给出改正后的算法。如果只是局部修改，建议大家只具体给出所修改的部分（例如，第 x 行 – 第 y 行代码修改为……）。请不要撇开原算法，自己重写一套。

```
template <class T>    void BinarySearchTree<T>::DeleteNodeEx
 (BinaryTreeNode<T>* pointer)        {                          /* 第 1 行 */
    if ( pointer == NULL )  return;                             /* 第 2 行 */
    BinaryTreeNode<T> * temppointer;                            /* 第 3 行 */
    BinaryTreeNode<T> * tempparent = NULL;                      /* 第 4 行 */
    BinaryTreeNode<T> * parent = GetParent(root, pointer );     /* 第 5 行 */
    if ( pointer->leftchild() != NULL ) {                       /* 第 6 行 */
        temppointer = pointer->leftchild();                     /* 第 7 行 */
        while (temppointer->rightchild() != NULL ) {            /* 第 8 行*/
            tempparent = temppointer;                           * 第 9 行*/
            temppointer = temppointer->rightchild();/* 第 10 行*/
```

```
            } // end of while                                  /* 第 11 行*/
            tempparent->right = temppointer->leftchild();     /* 第 12 行*/
            temppointer->left = pointer->leftchild();         /* 第 13 行*/
            temppointer->right = pointer->rightchild();       * 第 14 行*/
   }                                                          /* 第 15 行*/
   else temppointer = pointer->leftchild();                   /* 第 16 行*/
   if ( parent->leftchild() == pointer )                      /* 第 17 行*/
        parent->left = temppointer;                           /* 第 18 行*/
   else parent->right = temppointer;                          /* 第 19 行*/
   delete pointer;   pointer=NULL;                            /* 第 20 行*/
}                                                             /* 第 21 行*/
```

四、（15 分）算法填空

下面的函数 tree_compare（）采用广度优先周游来遍历两棵树，如果这两棵树的结构以及对应结点中的值相同则返回 true 否则 false。请填充算法的空格，使其成为完整的算法，空格中可能需要填写 0 至多条语句。

```
bool tree_compare(TreeNode* treeA, TreeNode* treeB)  {
        if (treeA == null) {
            if (treeB == null) return true;
            return false;
        }
        if (treeB == null) return false;
        using std::queue;
        queue<TreeNode*> aQueue, bQueue;
        TreeNode* p = treeA;
        TreeNode* q = treeB;
        While (true)  {
            While (p && q)  {
                aQueue.push(p);
                bQueue.push(q);
                // ?1
            }
            // ?2
            p = aQueue.front();
            aQueue.pop();
            q = bQueue.front();
            bQueue.pop();
            if (p->value != q->value) return false;
            // ?3
```

```
        }
    }
}
```

// 上述算法填空中可以参考的树结点定义

```
template<class T>
class TreeNode  {
public:
     TreeNode(const T&);                          //构造函数
     virtual ~TreeNode(){};                       //析构函数
     T Value();                                   //返回结点的值
     TreeNode<T>* LeftMostChild();                //返回第一个左孩子
     TreeNode<T>* RightSibling();                 //返回右兄弟
     void setValue(T&);                           //设置结点的值
     void setChild(TreeNode<T>* pointer);         //设置左孩子
     void setSibling(TreeNode<T>* pointer);       //设置右兄弟
     void InsertFirst(TreeNode<T>* node);         //以第一个左孩子身份插入结点
     void InsertNext(TreeNode<T>* node);          //以右兄弟的身份插入结点
   };
```

五、（15 分）算法设计

假设二叉树结点的值唯一，写出在二叉链表中查找值为 x 的结点在树中所处层数的算法 Nodelayer( )（假设根结点处于 0 层）。函数 Nodelayer( )返回值为 x 的结点的层次，若无值为 x 的结点，返回-1。

```
template<class T>
int NodeLayer(BinaryTreeNode<T>*root, <T> x)
```

// 上述算法填空中可以参考的二叉链表结点定义

```
template <class T>
class BinaryTreeNode {
private:
   T  element;                                    //二叉树结点数据域:
   BinaryTreeNode<T>*  left;                      //二叉树结点指向左子树的指针
   BinaryTreeNode<T>*  right;                     //二叉树结点指向左子树的指针
public:
     BinaryTreeNode();                            //缺省构造函数
     BinaryTreeNode(const T& ele);                //给定数据的构造函数
     T  value() const;                            //返回当前结点的数据
     BinaryTreeNode<T>*  leftchild() const;       //返回当前结点左子树
     BinaryTreeNode<T>*  rightchild() const;      //返回当前结点右子树
     void  setLeftchild(BinaryTreeNode<T>*) ;     //设置当前结点的左子树
     void  setRightchild(BinaryTreeNode<T>*) ;    //设置当前结点的右子树
```

```
    void  setValue(const T& val);                    //设置当前结点的数据域
};
```