中 山 大 学

二00六 年攻读硕士学位研究生入学考试试题

科目代码： 806

科目名称：数据结构

考试时间： 1 月 15 日 下 午

考 生 须 知

全部答案一律写在答题纸上，答在试题纸上的不得分！请用蓝、黑色墨水笔或圆珠笔作答。答题要写清题号，不必抄题。

一. 单项选择题（10 分，每小题 1 分）

1. 数据的逻辑结构指_____。
   A. 线性结构　　B. 链式结构
   C. 数据之间的逻辑关系　　D. 数据之间的物理关系
2. 数据的存储结构指_____。
   A. 数组类型　B. 指针类型　C. 数据之间的逻辑关系　D. 数据之间的物理关系
3. 一个完整的算法应该具有有穷性、确定性和可行性等。其中有穷性指算法_____。
   A. 在有穷时间内终止　　B. 输入是有穷的
   C. 输出是有穷的　　D. 描述是有穷的
4. 一个算法不仅对于合法的输入应该给出正确的输出，而且对于非法的输入也能够给出适当的反应而不是给出莫名其妙的结果或者崩溃，后者称为算法的_____。
   A. 正确性　　B. 可读性　　C. 健壮性　　D. 可维护性
5. 栈和队列具有相同的_____。
   A. 抽象数据类型　　B. 逻辑结构　　C. 存储结构　　D. 运算
6. 线性表的顺序存储结构是一种_____存储结构。
   A. 随机存取　　B. 顺序存取　　C. 索引存取　　D. 散列存取
7. 线性表采用链式结构存储时，相邻结点的存储地址_____。
   A. 必须是不连续的　　B. 连续与否均可
   C. 必须是连续的　　D. 和头结点的存储地址相连续
8. 下列说法正确的是_____。
   A. $\sqrt{n^5}=O(n^2)$　B. $\log_2 n^3 = O(n\log n)$　C. $\log_2 n^3 = \Theta(n\log n)$　D. $\min(800, n^2)=\Theta(1)$
9. 采用二分法在线性表上查找的条件是线性表有序并且_____。
   A. 使用链式存储结构　　B. 使用顺序存储结构
   C. 使用索引结构　　D. 使用散列结构
10. 哈希(hash)表的检索性能可以表示为_____的函数。
    A. 关键字集合的大小　　B. 哈希函数　　C. 装载因子　　D. 处理冲突的方法

二. 填空题（20 分，每小题 2 分）

1. 广义表 A(a, B(c,d), E(F(g,h)), ())的深度为_____。
2. 递归算法可以借助_____转换为非递归算法。
3. 使用稳定的排序算法将下列序列按照第二个分量非递减排序：
   ("Ana", 50), ("Tom", 33), ("Bob", 23), ("Alice", 33),
   则结果是_____。
4. 快速排序在最坏情况下的时间复杂度会退化为_____。
5. 在含有 n 个关键字的线性表中进行顺序查找，若查找第 i 个关键字的概率为 p_i $(1\le i\le n)$，则成功查找的平均比较次数为_____。
6. 一个 n 阶下三角方阵 $A_{n\times n}=(a_{ij})(0\le i,j\le n-1)$ 可以按照行优先的方法使用大小为 $n(n+1)/2$ 的数组 Z 存放，如果 a_{00} 存放在 Z[0]，则 $a_{ij}(i\ge j)$ 存放于_____。

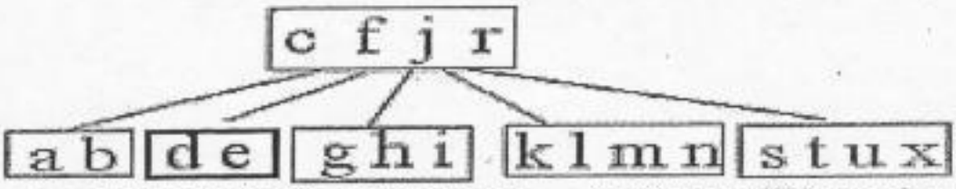
7. 有 5 个节点的 AVL 树的最大高度为_____(空二叉树的高度为 0)。
8. 将一棵 100 个结点的完全二叉树按照层次从 1 到 100 编号，则叶结点的最小编号是_____。
9. 递推公式 C(n) = n-1+C(n-1), C(1)= 0 的解是_____。
10. (a-b)*(a+b)的后缀表达式是 _____。

三. 解答题（60 分，每小题 6 分）

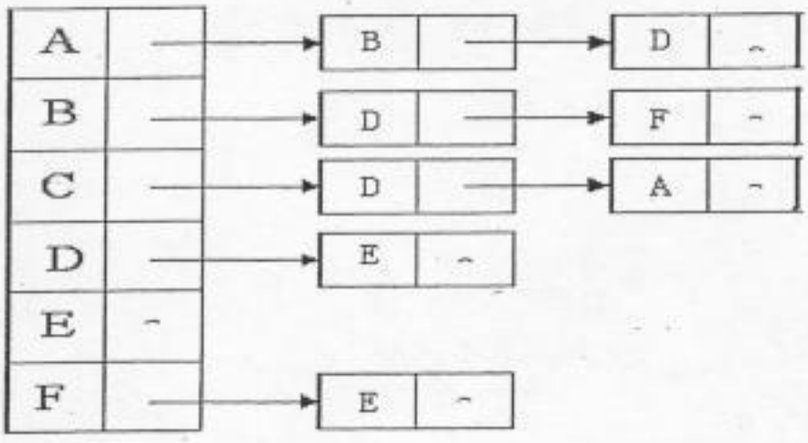
1. 已知一棵二叉树的中序遍历的结果是 DBAEGCF, 后序遍历的结果是 DBGEFCA, 试画出这棵二叉树及其先序遍历结果。
2. 请画出由空树出发，依次插入关键字 10, 11, 9, 12, 13, 14 后的平衡二叉树。
3. 设散列函数为 h(key)=key%11， 处理冲突的方法为线性探查法。 试画出在下列空散列表上依次插入关键字 10, 100, 32, 45, 58, 126, 3, 29 后散列表的情况。

0	1	2	3	4	5	6	7	8	9	10

4. 下图是一棵 5 阶 B-树。请画出插入元素 p 以后的 5 阶 B-树。



5. 下图是用邻接表存储的图， 请写出此图的拓扑排序序列。



6. 下面是解 Hanoi 塔的递归算法：

```
void Hanoi (int n, char x, char y, char z) {
    if (n==1)
        Move(x,1,z); //将编号为 1 的圆盘从 x 移到 z
    else {
        Hanoi(n-1, x, z, y);
        Move(x, n, z);
        Hanoi(n-1, y, x, z);
    }
}
```

试说明函数 Hanoi 的时间复杂度和空间复杂度各是什么？ 为什么？

7. 设带权有向图 G=(V, E),
   V={ V_0, V_1, V_2, V_3, V_4, V_5},
   E= {(V_0, V_2, 10), (V_0, V_4, 30), (V_0, V_5, 100), (V_1, V_2, 5), (V_2, V_3, 50), (V_3, V_5,10), (V_4, V_3, 20), (V_4, V_5, 60) }, 其中(V_0,V_2,10)表示 V_1 到 V_2 的长度是 10， 其他类同。 试求 V_0 到其他各结点的最短路径长度。
8. 已知四个字符 A,B,I 和 L 的 Huffman 编码分别为 1, 01, 000 和 001。 下列 0,1 串是由以上四个字母构成的一段文本的 Huffman 编码：
   10010000110110100110100 11

请将上述 0,1 串还原为编码前的文本，并画出相应的 Huffman 树。

9. 画出有序表 (21, 22, 28, 31, 35, 40, 45, 56,70)的二分查找（binary search）的比较树（或称查找判定树， comparison trees）(按下取整，数组下标从 0 开始)。

10. 将下列序列排序为非递减序列：

26， 33， 35， 29， 19， 12， 22

请画出使用堆排序的排序过程。

四. 程序设计(60 分)

1.（20 分）假设下列类型说明：

```
template <class Record> class Sortable_list {
   public:
      Sortable_list();
      ~Sortable_list();
      recursive_quick_sort(int low, int high);
   private:
     Record entry[max_list];
     swap(int i, int j); //交换 entry 中第 i 个和第 j 个位置的记录
     int partition(int low, int high);
};
```

完成下列函数 recursive_quick_sort() 和 partition()。

a)

```
template <class Record>
void Sortable_list<Record>::recursive_quick_sort(int low, int high)
low 和 high （low <=high）是 Sortable_list 中的合法位置。算法结束时，表中介于 low
和 high 之间的记录按照非递减序排列。
*/
{
    int pivot_position;
    if (      (1)      ) {
        pivot_position = partition(low, high);
        recursive_quick_sort(low,       (2)      );
        recursive_quick_sort(      (3)      , high);
    }
}
```

b)

```
template <class Record>
int Sortable_list<Record>::partition(int low, int high)
/*
low 和 high （low <=high）是 Sortable_list 中的合法位置。
函数使用表中介于 low 和 high 中间的记录作为支点，并且将表中的元素调整，使得
小于支点的元素位于支点前面，其余的元素位于支点之后。函数返回支点的最后位置。
*/

{
    Record pivot;
    int i;                  //   used to scan through the list
    int      last_small;    //   position of the last key less than pivot
    swap(low, (low + high) / 2);
    pivot = entry[low];
    last_small = low;
    for (i =      (4)      ; i <= high; i++)
        if (entry[i] < pivot) {
            last_small =      (5)      ;
            swap(last_small, i);
        }
```

```
    swap(     (6)     , last_small);
    return    (7)     ;
}
```

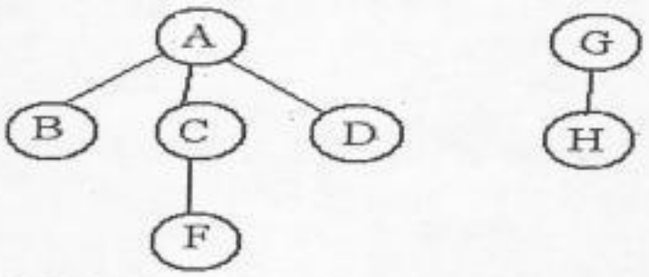
2. (20 分)试设计一个实现下述要求的 *Locate* 运算的函数。设有一个带表头结点的双向链表 *L*，每个结点有 4 个数据成员：指向前驱结点的指针 *prior*、指向后继结点的指针 *next*、存放数据的成员 *data* 和访问频度 *freq*，结点定义如下所示

```
typedef struct DblNode {
    ElemType      data;
    int freq;
    struct DblNode    *lLink;
    struct DblNode    *rLink;
}DblNode;
```

所有结点的 *freq* 初始时都为 0。每当在链表上进行一次 *Locate* (*L*, *x*)操作时，令元素值为 *x* 的结点的访问频度 *freq* 加 1，并将该结点前移，链接到与它的访问频度相等的结点后面，使得链表中所有结点保持按访问频度递减的顺序排列，以使频繁访问的结点总是靠近表头。试利用 C/C++实现满足上述要求的函数：

```
void Locate ( DblNode * & first,   ElemType & x ) ;
    //first 指向链表的头结点
```

3.（20 分）设计一个按层次顺序遍历森林的算法。例如，下图森林的层次遍历顺序是 AGBCDHF



a) 试利用 C/C++定义你选择的森林存储结构；

b) 利用 C/C++实现以上描述的层次遍历森林算法：

```
void ForestTraverse(Forest t, void (*visit)(ElemType d));
//按照层次序遍历森林 t，在遍历过程中将函数 visit 作用森林的每个结点。
```

c) 利用 C/C++实现求森林中树的最大高度的函数（如，上图森林中树的最大高度是 3）：

```
int height(Forest t);
// 函数返回森林 t 中树的最大高度。
```