武汉科技大学

2017 年全国硕士研究生招生考试初试自命题试题

科目名称：数据结构（C 语言版）（■A 卷□B 卷）科目代码：856

考试时间：3 小时　满分 150　分

可使用的常用工具：√无　□计算器　□直尺　□圆规（请在使用工具前打√）

注意：所有答题内容必须写在答题纸上，写在试题或草稿纸上的一律无效；考完后试题随答题纸交回。

准考证号码：　报考专业：　姓名：　密封线内不要写题

一、选择题(共 10 小题，每小题 2 分，共 20 分)

1. 以下属于逻辑结构的是（　　）。
   A）顺序表　　B）哈希表　　C）有序表　　D）单链表
2. 栈的插入和删除操作在（　　）进行。
   A）栈顶　　B）栈底　　C）任意位置　　D）指定位置
3. 将一个 A[1...100，1...100]的三对角矩阵，按行优先存入一维数组 B[1...298]中，A 中元素 A[66,65]在 B 数组中的位置 K 为（　　）。
   A）195　　B）196　　C）197　　D）198
4. 二叉树高度为 h,所有结点的度为 0 或 2，则该二叉树最少有（　　）结点。
   A）2h　　B）2h-1　　C）2h+1　　D）h+1
5. 一棵非空的二叉树的先序遍历序列与后序遍历序列正好相反，则该二叉树一定满足（　　）
   A）所有的结点均无左孩子　　B）所有的结点均无右孩子
   C）只有一个叶子结点　　D）是任意一棵二叉树
6. 下面结构中最适于表示稀疏有向图的是（　　）。
   A）邻接矩阵　　B）邻接表　　C）邻接多重表　　D）十字链表
7. 求解最短路径的 Floyd 算法的时间复杂度为（　　）。
   A）O（n）　　B）O（n+c）　　C）O（n*n）　　D）O（n*n*n）
8. 二叉排序树的查找效率与二叉树的（　　）有关。
   A）高度　　B）树型　　C）结点的多少　　D）结点的位置
9. 下面关于哈希查找的说法正确的是（　　）。
   A）哈希函数构造的越复杂越好，因为这样随机性好，冲突小
   B）除留余数法是所有哈希函数中最好的
   C）不存在特别好与坏的哈希函数，要视情况而定
   D）若需在哈希表中删去一个元素，不管用何种方法解决冲突都只要简单的将该元素删去即可
10. 直接插入排序在最好情况下的时间复杂度为（　　）。
    A）O(logn)　　B）O(n)　　C）O(nlogn)　　D）O(n*n)

二、填空题(共 10 小题，每小题 2 分，共 20 分)

1. 下面程序段的时间复杂度为（　　）。
   sum=1;　for(i=0;sum<n;i++) sum+=1;　//n>1
2. 循环队列的引入，目的是为了克服（　　）。
3. 有广义表 H=(A,(a,b,c))，运用 head 和 tail 函数求出广义表 H 中元素 b 的运算式（　　）。
4. 具有 n 个结点的满二叉树，其叶结点的个数是（　　）。
5. 二叉树结点的中序序列为 ABCDEFG,后序序列为 BDCAFGE,则该二叉树对应的树林包括（　　）棵树。
6. 构造连通网最小生成树的两个典型算法是（　　）。
7. 在 AOE 网中,从源点到汇点路径上各活动时间总和最长的路径称为(　　)。
8. 在有序表 A[1..12]中，采用二分查找算法查等于 A[12]的元素，所比较的元素下标依次为（　　）。
9. （　　）法构造的哈希函数肯定不会发生冲突。
10. 在排序算法的最后一趟开始之前，所有元素都可能不在其最终位置上的排序算法是（　　）。

三、判断题(共 10 小题，每小题 2 分，共 20 分)

1. 算法的优劣与算法描述语言无关，但与所用计算机有关。
2. 链表是采用链式存储结构的线性表，进行插入、删除操作时，在链表中比在顺序存储结构中效率高。
3. 链表中的头结点仅起到标识的作用。
4. 任何一个递归过程都可以转换成非递归过程。
5. Huffman 树的结点个数不能是偶数。
6. 二叉树的所有叶子结点在前序和后序遍历序列中皆以相同的相对位置出现。
7. 有向图的邻接矩阵是对称的。
8. 查找相同结点的效率折半查找总比顺序查找高。
9. Hash 表的平均查找长度与处理冲突的方法无关。
10. 在执行排序算法过程中，出现排序码朝着最终排序序列位置相反方向移动，则该算法是不稳定的。

四、综合应用题(共 5 小题，每小题 10 分，共 50 分)

1. 下面给出一棵树的双亲表示法的顺序存储结构。

下标	0	1	2	3	4	5	6	7	8	9
Data	R	A	B	C	D	E	F	G	H	K
Parent	-1	0	0	0	1	1	3	6	6	6

（1）画出该树的示意图
（2）写出该树的先序序列和后序序列
（3）将该树转换为对应的二叉树
（4）写出转换后的二叉树的先序和后序序列
（5）画出上述二叉树的先序线索二叉树的示意图

2. 老康维修牧场的一段栅栏需要 13 根木头（其长度分别为 2，3，5，7，11，13，17，19，23，29，31，37 和 41）。他购买了一根长度为 238 的木头（正好是前面 13 段木头的和）。老康需要将这根木头按照要求长度依次锯成 13 段，每次锯木头需要消耗一定的体力（与所锯木头的长度成正比，假设比例为 1）。

   请问老康完成任务需要最小消耗多少体力？给出你的结论和分析过程。

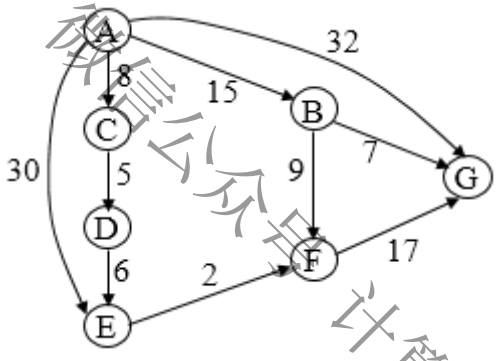
3. 给定关键字序列 45，24，53，27，93，12，30。

   （1）请按照上述序列构建一颗二叉排序树。

   （2）计算等概率情况下查找成功的平均查找长度。

   （3）计算等概率情况下查找不成功的平均查找长度。

4. 给定如下有向图，请写出按照 Dijkstra 算法求出顶点 A 到其他各顶点的最短路径的详细过程（即填写下表）。



终点	从顶点 A 到各终点的最短路径 S 和最短距离 D 的求解过程					
	i=1	i=2	i=3	i=4	i=5	i=6
B						
C						
D						
E						
F						
G						
Vj:D						
S						

5. 给定无序序列 39，35，70，87，81，7，27，58。

   （1）将上述整数构建一个堆（大顶堆）。

   （2）按照堆排序算法，写出前三趟堆排序的结果。

   （3）堆排序的时间和空间复杂度分别是多少？

五、算法设计(第 1 题 10 分，第 2，3 题各 15 分，共 40 分)

1. 有 2 个带头结点的单链表，其数据结构描述如下，各结点的数据域递减有序。

```
typedef struct Node
{  int data;
   struct Node *next; } *LinkList;
```

请设计函数 Merge 将两个链表（La 和 Lb）合并，原链表仍然保留，生成新的链表（Lc），新链表各结点的数据域递增有序，并返回新链表的结点数。要求，单链表中均无数据域相同的结点，设计的程序运行效率尽量高。

```
int Merge(LinkList &La,LinkList &Lb,LinkList &Lc)
```

2. 整数 1 到 n 从小到大依次进栈，期间可以出栈。请设计算法判定给定的 1 到 n 的整数序列受否是正确的出栈序列。如果是返回 true，否则返回 false。

```
bool Judge(int a[],int n)
```

3. 二叉树采用二叉链表进行存储（如下所示），每个结点包含数据域 Data，左孩子指针域 left 和右孩子指针域 right。请设计非递归算法统计二叉树的高度。

```
Typedef struct BitNode
{  TElemType data;
   struct BitNode *left, *right;
} *BiTree ;
```

参考答案(A)

一、选择题（10 小题，每题 2 分，共 20 分）

CAABC　DDBCB

二、填空题（10 小题，每题 2 分，共 20 分）

1. O（n）
2. 假溢出
3. Head(tail(head(tail(H))))
4. (n+1)/2
5. 2
6. Prim 和 Kruskal
7. 关键路径
8. A[6],A[9].,A[11],A[12]
9. 直接定址法
10. 插入排序

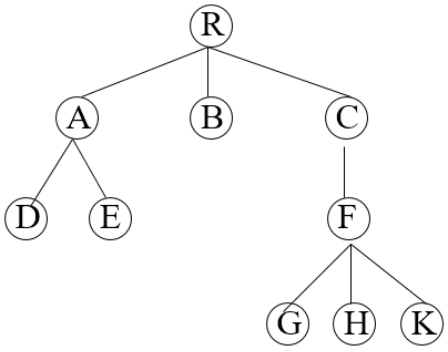
三、判断题(10 小题，每小题 2 分，共 20 分)

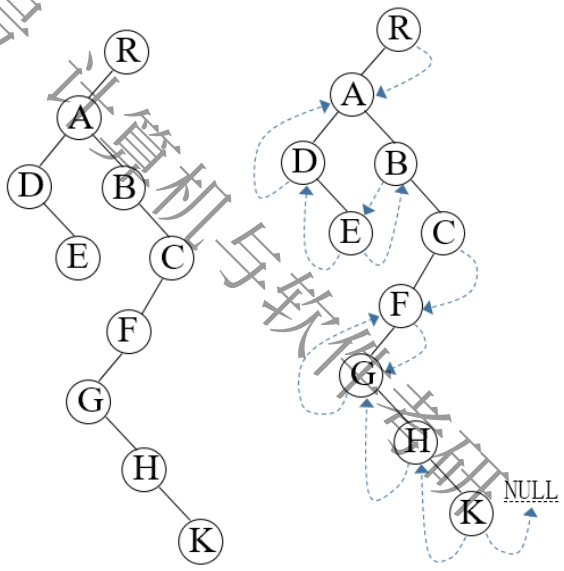
×√×√√　√××××

四、综合应用题（5 小题，每题 10 分，共 50 分）

1.

（1）画出该树的示意图（如下边左图）





（2）写出该树的先序序列和后序序列

先序序列：RADEBCFGHK　　后序序列：DEABGHKFCR

（3）将该树转换为对应的二叉树（如上边中图）

（4）写出转换后的二叉树的先序和后序序列

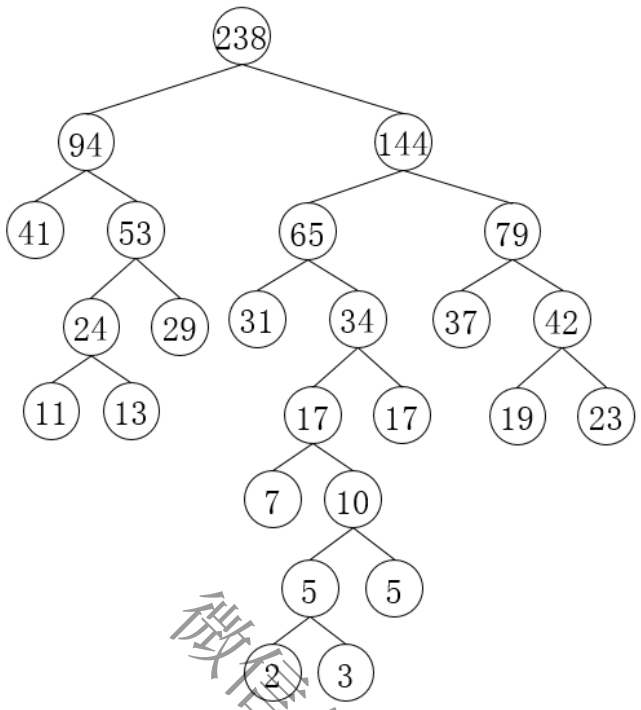
先序序列:RADEBCFGHK　　后序序列:EDKHGFCBAR

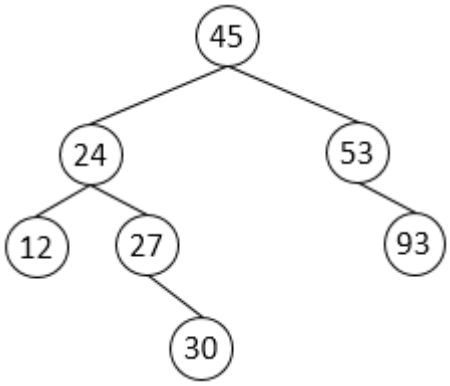
（5）画出上述二叉树的先序线索二叉树的示意图（如上边右图）

2.

构造哈夫曼树（左右孩子可以互换），求 WPL

WPL=41*2+29*3+37*3+31*3+11*4+13*4+19*4+23*4+17*4+7*5+5*6+2*7+3*7=805





3.

（1） 如上边右图所示

（2）成功：ASL=（1+2+2+3+3+3+4）/7=18/7

（3）不成功：ASL=(2+3*5+4*2)/8=25/8

4.

终点	从顶点 A 到各终点的最短路径 S 和最短距离 D 的求解过程					
	i=1	i=2	i=3	i=4	i=5	i=6
B	15 AB	15 AB	15 AB	-----	-----	-----
C	8 AC	-----	-----	-----	-----	-----
D	∞	13 ACD	-----	-----	-----	-----
E	30 AE	30 AE	19 ACDE	19 ACDE	-----	-----
F	∞	∞	∞	24 ABF	21 ACDEF	-----
G	32 AG	32 AG	32 AG	22 ABG	22 ABG	22 ABG
Vj:D	C:8	D:13	B:15	E:19	F:21	G:22
S	AC	ACD	AB	ACDE	ACDEF	ABG

5.（1）初始堆：87 81 70 58 39 7 27 35

（2）第 1 趟：81 58 70 35 39 7 27 87

第 2 趟：70 58 27 35 39 7 81 87

第 3 趟：58 39 27 35 7 70 81 87

（3）时间复杂度：O（nlogn） 空间复杂度：O（1）

五、算法设计（第 1 题 10 分，第 2，3 题各 15 分，共 40 分）

1.

```
int Merge(LinkList &La,LinkList &Lb,LinkList &Lc)
{
    pa=La->next;
    pb=Lb->next;
    Lc=(LinkList)malloc(sizeof(struct Node));
    Lc->next=NULL;+
    num=0;
    while(pa&&pb)
    {
        num++;
        s=(LinkList)malloc(sizeof(struct Node));
        if(pa->data<pb->data)   {   s->data=pb->data;  pb=pb->next;   }
        else if(pa->data>pb->data) {   s->data=pa->data;  pa=pa->next;   }
        else  {   s->data=pb->data;  pa=pa->next; pb=pb->next;   }+
        s->next=Lc->next;
        Lc->next=s;
    }
    if(pb)  pa=pb;
    while(pa)
    {
        num++;
        s=(LinkList)malloc(sizeof(struct Node));
        s->data=pb->data;  pa=pa->next;
        s->next=Lc->next;
        Lc->next=s;
    }
    return num;
}
```

2.

```
bool Judge(int a[],int n)
{
    t=0;  //匹配的整数个数
    top=0; //初始化栈，top 为栈顶
    k=0;  //t 为待匹配的整数序列的下标
    for(i=1;i<=n;i++)
    {
        stack[top]=i; top++;   //i 入栈
        while(top>0&&stack[top-1]==a[k]) //栈不空且栈顶等于待匹配的整数
        {
            top--;  //出栈
            k++;   //匹配下一个
            t=0;    //匹配+1
        }
    }
    if(t==n) return true;
    else return false;
}
```

3.

```
int Height(BiTree T) //非递归算法，采用层次遍历的方法求二叉树 T 的高度
{
    if(T==NULL) return 0;  //空二叉树高度为 0
    BiTree Q[maxsize]; //Q 是循环队列，元素为二叉树结点指针
    front=0;       //队头指针
    Q[0]=T;        //根结点入队列
    rear=1;        //队尾指针
    count=1;        //count 用来记录当前层的结点数
    temp=0;  //用来统计当前层已访问的结点数
    height=0;      //记录高度
    while(front!=last)  //队列不空
    {
        p=Q[front];  front=(front+1)%maxsize;  //出队列
        if(p->lchild!=NULL) { Q[rear]=p->lchild; rear=(rear+1)%maxsize; } //左孩子入队
        if(p->rchild!=NULL) { Q[rear]=p->rchild; rear=(rear+1)%maxsize; } //右孩子入队
        temp++;
        if(temp>=count)  //一层结束，
        {
            count=(rear-front+maxsize)%maxsize;  //新的一层结点数
            temp=0; //新的一层重新开始统计已访问结点数
            height++;   //新的一层高度+1
        }
    }
    return height;
}
```