启用前
绝密

哈尔滨工业大学

2001 年硕士研究生入学考试试题参考答案

考试科目：数据结构(覆盖高级语言)　　报考专业：计算机科学与技术

考试科目代码：[419]

主观问答题，可根据考生表述明确与否酌情给分。

题号	一	二	三	四	五	六	七	八	九		总分
分数	10	10	15	8	10	12	12	10	13		100

一、填空题（每小题 2 分，共 10 分）

1. O(1)　O(n)　　2. 5　　3　3. 完全　　退化的单链树　　4. i-1　　5. $O(n^2)$

二、选择题（每小题 2 分，共 10 分）

1. A　　2. C　3×2+2×1+1×2=2+1+2+X-1 →X=6　3. B　　C　4. C　　5. D

三、简答题（每题 3 分，共 15 分）

1. 数据结构是指数据元素之间抽象的相互关系，并不涉及数据元素的具体内容。而数据类型指的是具体数据所属的类型，两者概念不同。

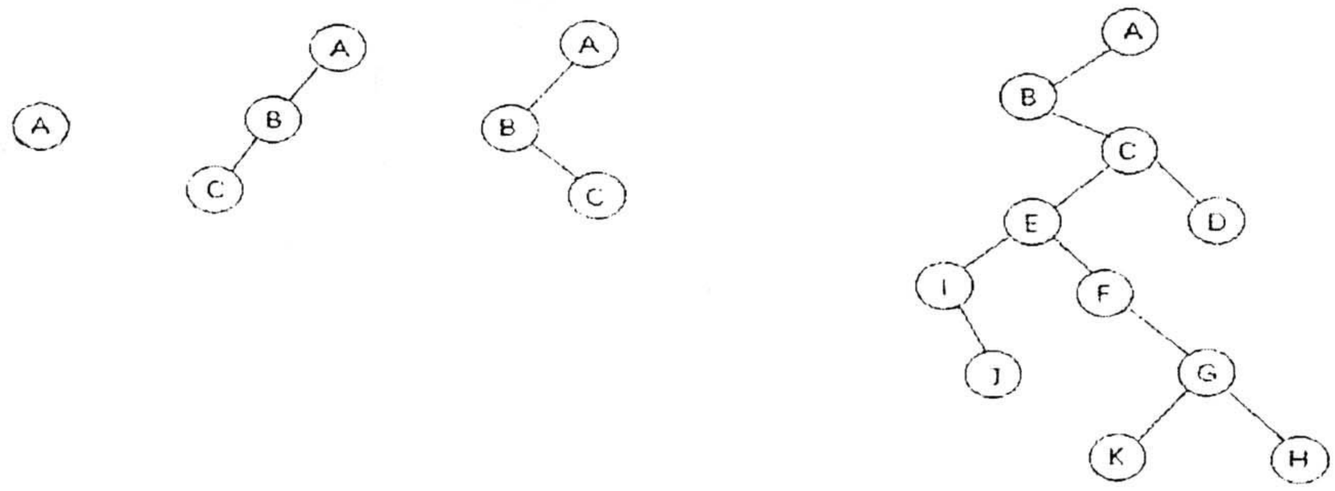
2. 当用数组表示排队时，我们把数组看成一个环形，即令数组中第一个元素紧跟在其最末一个单元之后，就行成了一个循环队列。

3. 在二元树的左右链表示法中，用线索取代原来的^链。当 lchild 为空时，令其指向按中根遍历时的前导结点；当 rchild 为空时，令其指向按中根遍历时的后继结点。这样形成的二元树称为线索二元树。

4. 从图中某一个结点开始，沿着有向边方向依次访问每个结点且只访问一次的过程。

5. 索引文件是在主文件之外再建立一个指示关键字与其物理记录之间的对应关系的表，这种表称为索引表。索引表与主文件共同构成索引文件。顺序存放的索引文件，称为索引顺序文件。

四、根据左孩子右兄弟规则，可得如图所示二元树：



(1)　　(2)　　(3)　　(4)

五、试设计一个算法，判断链表 L 是否是递减的。

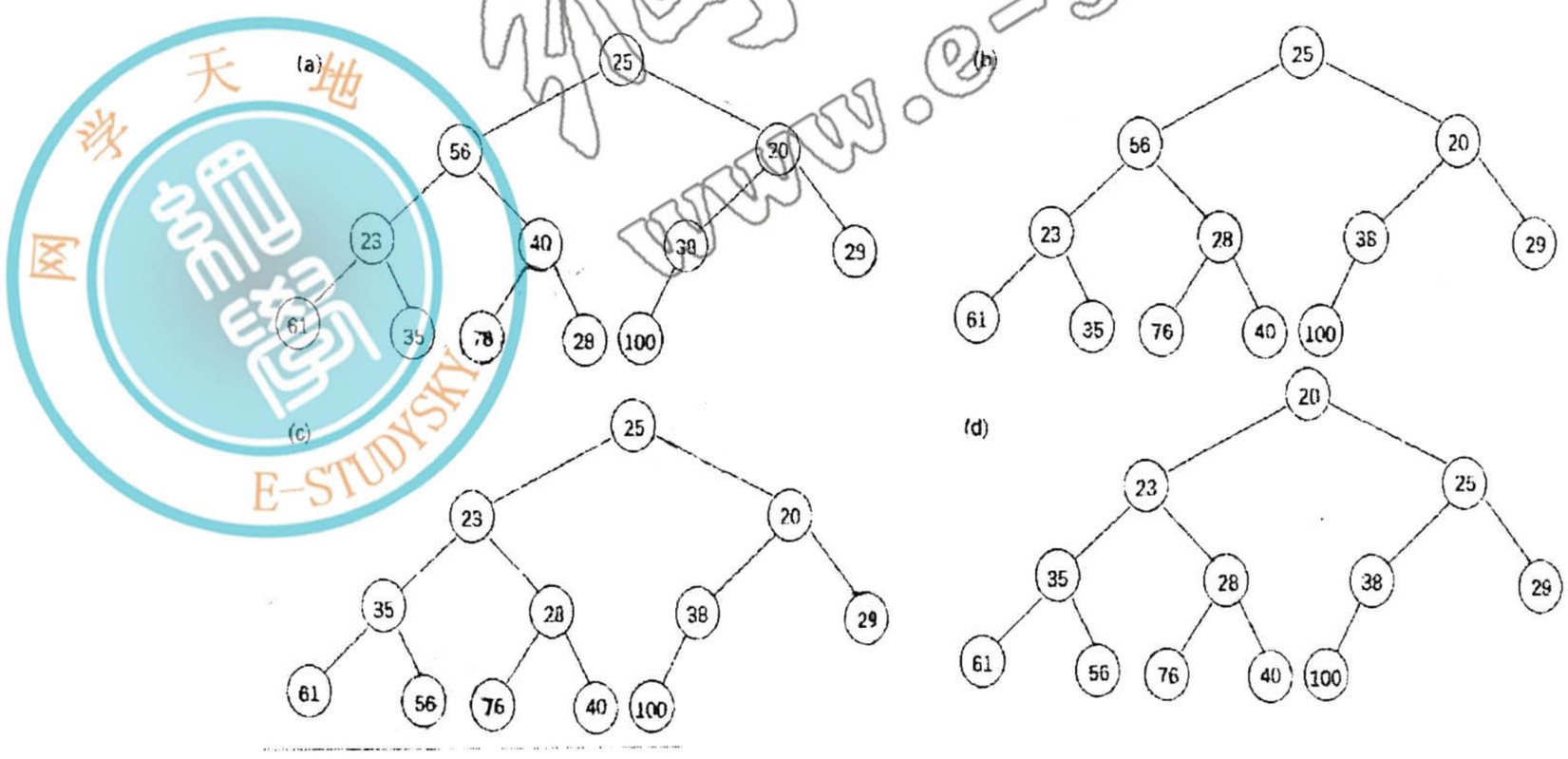
```
bool Judge(List &l.)
{
```

```
position p, q;
bool a = true;
p = FIRST(L); q = p->next;
while ((p->next != NULL) && (a == true))
{
    if (p->element>q->element)
    {
        p = q;
        q = p->next;
    }
    else
        a = false;
}
return a;
}
```

六、 (1) (12,24,33,65,33,56,48,92,86,70)是堆。

(2) (25,56,20,23,40,38,29,61,35,76,28,100)不是堆。

成堆过程为：



七、

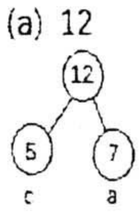
```
int maxsize = 100;
typedef int elementtype;
typedef struct
{
    elementtype stack[maxsize];
    int top[2];
} *dqstype; //定义抽象数据型
void initialize(dqstype s) //初始化
{
    s->top[0] = -1;
    s->top[1] = maxsize;
```

```
}
int PUSH(dqstype s, int i, elementtype x) //i为控制器，i=0时对S1操作,i=1时对S2操
作
{
    if (s->top[0] == s->top[1] - 1)
    {
        printf("Stack is full!");
        return 0;
    }
    if ((i != 0) && (i != 1))
    {
        printf("Error!");
        return 0;
    }
    if (i == 0) //对S1操作
    {
        s->top[i]++;
        s->stack[s->top[i]] = x;
    }
    else //对S2操作
    {
        s->top[i]--;
        s->stack[s->top[i]] = x;
    }
    return 1;
}
int POP(dqstype s, int i, elementtype *x)
{
    if (s->top[0] == s->top[1] - 1)
    {
        printf("Stack is full!");
        return 0;
    }
    if ((i != 0) && (i != 1))
    {
        printf("Error!");
        return 0;
    }
    if (i == 0)
        *x=s->stack[s->top[i]--];
    else
        *x = s->stack[s->top[i]++];
    return 1;
}
```

八、

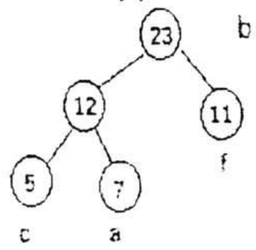
a	b	c	d	e	f
7	19	5	16	42	11

(a) 12 19 16 42 11



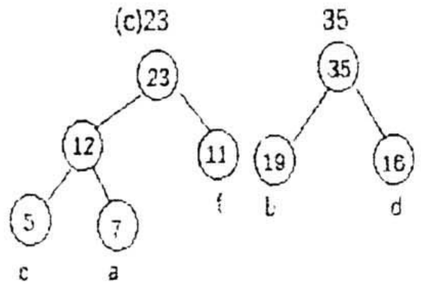
b d e f

(b)23 19 16 42



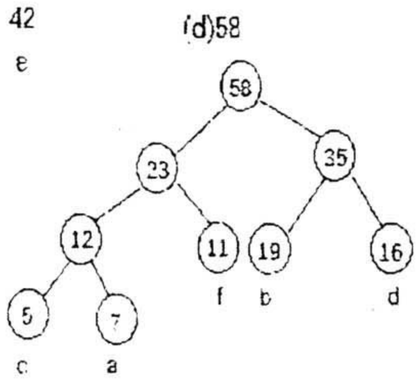
b d e

(c)23 35 42



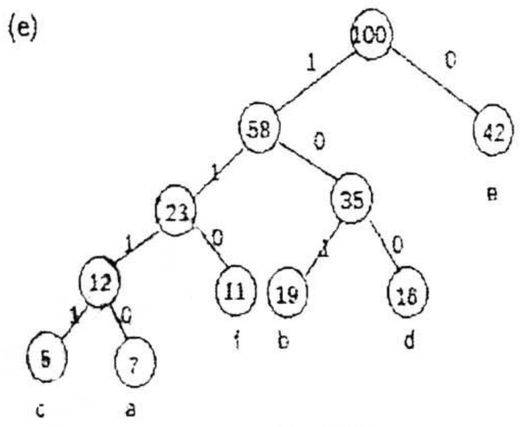
e

(d)58 42



e

(e)



故：

a 1110
b 101
c 1111
d 100
e 0
f 110

九、解答一：

邻接表的抽象数据型如下所示：

```
struct node
{
    int adjvex;
    node *next;
} //结点的型
typedef struct
{
    vertexttype data;
    node *firstarc;
} ADJLIST; //邻接表的型
```

法一：广度优先遍历（使用队列）

```
int RBFS(ADJLIST g, int vi, int vj)
{
    int i, yes = 0;
    QUEUE Q;
    for (i = 0; i<n; i++)
        visited = 0;
    ENQUEUE(vi, Q);
    visited[vi] = 1; //初始化
    while (!EMPTY(Q) && !yes)
    {
    w = Q.front;
    p = g[w].firstarc;
    while ((p != NULL) && !yes)
    {
```

```
            w = p->adjdata;
            if (visited[w] = 0)
            {
                if (w == vj) yes = 1;
                visited[w] = 1;
                ENQUEUE(w, Q);
            }
            else p = p->next;
        }
    }
    return yes;
}
```

法二：深度优先遍历（使用栈）

```
int RDFS(ADJLIST g, int vi, vj)
{
    int i; int yes = 0; STACK S;
    for (i = 0; i<n; i++)
        visited = 0;
    PUSH(vi, S);
    visited[vi] = 1; //初始化
    while (!EMPTY(S) && (yes == 0))
    {
        w = TOP(S);
        p = g[w].firstarc;
        while ((p != NULL) && visited[p->adjdata])
        {
            p = p->next;
            if (p == NULL) POP(S);
            else
            {
                w = p->adjdata;
                if (w == vj) yes == 1;
                visited[w] = 1;
                PUSH(w, S);
            }
        }
    }
    return yes;
}
```

解答二：

算法思想：采用广度优先遍历或深度优先遍历，都从顶点vi出发，依次遍历图中每个顶点，直到搜索到vj，如果能搜索到vj，则说明存在由顶点vi到vj的路径。

采用深度优先遍历算法的实现如下：

```
int visited[maxsize] = { 0 };
```

```
int Exist_Path_DFS(ALGraph G, int i, int j){
    //深度优先判断有向图顶点vi到顶点vj是否有路径，是则返回1，否则返回0
    int p;//顶点序号
    if (i == j)
        return 1;
    else{
        visited[i] = 1;
        for (p = FirstNeighbor(G, i); p >= 0; p = NextNerghbor(G, i, p)){
            if (!visited[p] && Exist_Path_DFS(G,p,j))//递归检测邻接点
        }
    }
    return 0;
}
```

采用广度优先遍历算法的实现如下：

```
int visited[maxsize] = { 0 };
int Exist_Path_BFS(ALGraph G, int i, int j)
{
    //深度优先判断有向图G中顶点vi到顶点vj是否有路径，是则返回1，否则返回0
    InitQueue(Q);
    EnQueue(Q, i);
    while (!IsEmpty(Q)){
        DeQueue(Q, u);
        visited[u] = 1;
        for (p = FirstNeighbor(G, j); p = NextNerghbor(G, i, p)){
            k = p.adjvex;
            if (k == j)
                return 1;
            if (!visited[k])
                EnQueue(Q, k);
        }
    }
    return 0;
}
```