北方工业大学
2004 年硕士学位研究生入学考试试题

考试科目：数据结构
适用专业：计算机科学与应用
说　　明：算法用类 C 或类 PASCAL 语言描述，在题目之前注明自己所用语言，算法书写必须规范。

(答题请写在答题纸上，试题上答题无效)

一、简要回答下列各题（共 36 分）

1、设森林 T 中有 4 棵树，4 棵树的结点个数分别是 n1、n2、n3、n4，树的度分别是 k1、k2、k3、k4，若将森林 T 转换成一棵二叉树，则此二叉树根结点左右子树中的结点个数分别为多少？（4 分）

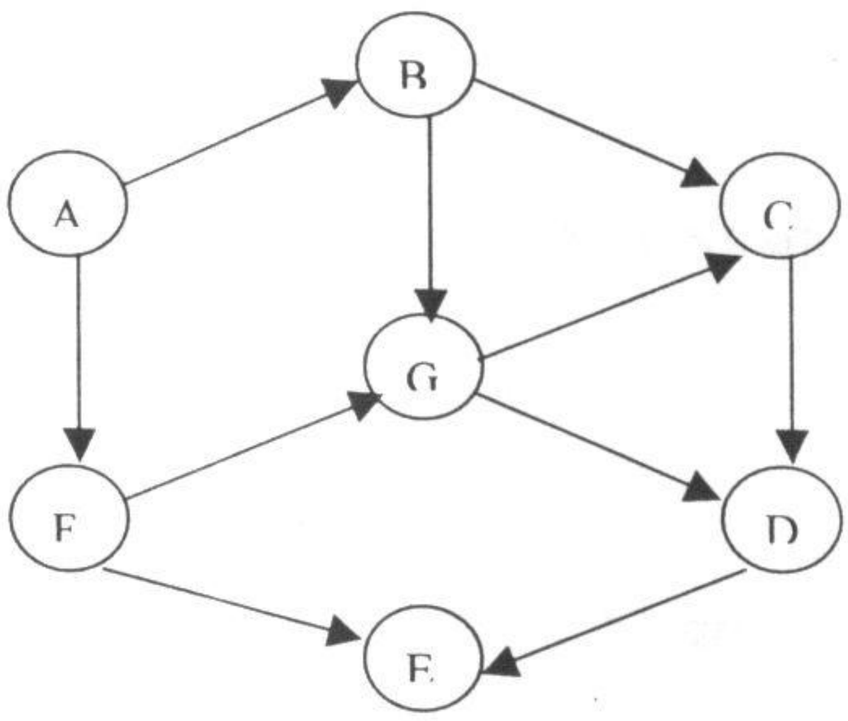
2、若 p 指针指向链表中的某个结点，设链表长度为 n（ n > 0 ），在不知道链表头指针的情况下，是否可以删除 p 指针所指结点？若可以，其时间复杂度为多少？试以链表为单链表、单循环链表和双循环链表三种情况分别讨论。（5 分）

3、若完全二叉树的第 k (k > 1) 层上有 m 个结点，则该完全二叉树的结点个数和叶子结点个数分别为多少？（4 分）

4、对于具有 n 个元素的有序表，顺序取表中数据构造一棵二叉排序树，计算该树的平均查找长度。（4 分）

5、利用快速排序方法对 n 个数据进行排序，在最坏情况下其时间复杂度为多少？什么情况下能达到最坏？（3 分）

6、有向图如下：



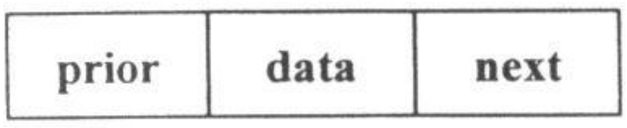
（1） 分别画出其邻接表和逆邻接表。（4 分）
（2） 给出其顶点的一个拓扑序列。（2 分）

7、将下列序列调整成一个小顶堆（**24 , 70 , 33 , 92 , 65 , 20 , 48 , 86 , 75 , 56** ）（4 分）

8、用带头循环链表表示队列（设队列长度为 n）
若只设头指针，则出队列和入队列时间复杂度分别是多少？ （3 分）
若只设尾指针，则出队列和入队列时间复杂度分别是多少？ （3 分）

二、（12 分）双向循环链表对称是指以头结点为中心，左右两边对称位置上的结点值相等。设计一个算法，判断给定带头结点的双向循环链表 L 是否对称。

链表结点结构为：



三、（12 分）一棵 m 叉的树 T 以孩子兄弟表示法存储，其结点结构描述为：

```
typedef   struct   CSNode {
        ElemType   data ;
        Struct   CSNode   *firstchild , *nextsibling ;
} CSNode  ,  *CSTree ;
```

设计一个算法求该树中叶子结点的个数。

四、（15 分）无向图 G 以邻接表表示存储，编一算法，判断图中是否存在从顶点 u 到顶点 v 的一条边。注意：u、v 类型为 **VertexType** 类型。要求先简述算法思路后在写出算法。

邻接表结构为：

（类 C 描述）

```
#define   MAX_VERTEX_NUM   20
typedef   struct   ArcNode {
        int                    adjvex;
        struct ArcNode     *nextarc;
        }ArcNode ;
typedef   struct   VNode {
        VertexType      data;
        ArcNode         *firstarc;
        }VNode ,AdjList[MAX_VERTEX_NUM];
typedef   struct   {
        AdjList      vertices;
        int          vexnum,arcnum;
        }ALGraph ;
```

（类 PASCAL 描述）

```
CONST    VERTEX_NUM =  实际顶点个数;
Type   arcptr = ^ArcNode ;
      ArcNode=RECORD
          adjvex : Integer ;
          nextarc : arcptr ;
        END ;
        VNode = RECORD
          data : VertexType ;
          firstarc : arcptr ;
        END ;
AdjList=ARRAY[1..VERTEX_NUM] OF VNode ;
```

,每小题个 5 分）简述

.）　陷阱与中断的区别

2)　P, V 原语和锁机制实现互斥的区别

、10 分）如何将独占型输入设备改造成可共享使用的虚拟设备？

.. （10 分）请阅读下面的程序，写出这个程序的可能输出。

```
#include  <sys/types.h>
#include  <stdio.h>
#include  <string.h>

int          glob = 6;          /* external variable in initialized data */
char buf[] = "a write to stdout\n";
int
main(void)
{
        int var;    /* automatic variable on the stack */
        pid_t       pid;
        var = 88;
        if (write(STDOUT_FILENO, buf, sizeof(buf)-1) != sizeof(buf)-1)
                err_sys("write error");
        printf("before fork\n");   /* we don't flush stdout */
        if ( (pid = fork()) < 0)
                err_sys("fork error");
        else if (pid == 0) {            /* child */
                glob++;             /* modify variables */
                var++;
        } else
                sleep(2);                       /* parent */
        printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);
        exit(0);
}
```

八．（15 分）下面的表中给出了系统某时刻的资源分配情况，其中 R1,R2,R3 代表系统中的三类资源，每类资源的总数分别为：12，7，4；试问：

1)　如果进程 B 提出请求资源（0，1，0），系统能否将资源分配给它？为什么？

2)　如果进程 E 又提出请求资源（1，1；0），系统能否将资源分配给它？为什么？

进程	已分配的资源			还需要的资源			剩余的资源		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
A	3	1	1	1	0	0	1	2	0
B	0	0	0	0	1	2			
C	1	1	0	3	0	0			
D	1	0	1	0	1	0			
E	0	0	0	2	1	0			

九．（15 分）下面是实现互斥的 3 个算法，请你仔细阅读它们，如果你认为它们在实现互持操作时有问题或不足，请说明可能存在的问题。

算法 1：

```
int turn = 0 ;
p0:{
   do {
        while ( turn ! = 0 ) ;
        进程 p0 的临界区代码 CS0;
        turn = 1 ;
        进程 p0 的其他代码;
   }
while （ture）
}

p1:{
   do {
        while ( turn ! = 0 ) ;
        进程 p1 的临界区代码 CS1;
        turn = 0 ;
        进程 p1 的其他代码;
   }
while （ture）
}
```

算法 2：

```
enum boolean {false , true };
boolean flag[2] = {false , true };
p0: {
        do  {
            while  flag [1] ;
                flag[0] = true;
                进程 p0 的临界区代码 CS0;
                Flag[0] = false ;
                进程 p0 的其他代码;
            }
while （ture）
}

p1: {
        do {
            while flag[0] ;
            flag [1] = true;
            进程 p1 的临界区代码 CS1;
            flag [1]  = flase ;
            进程 p1 的其他代码;
        }
while （ture）
}
```

算法 3：

```
enum boolean {false , true };
boolean flag[2] = {false , false };
p0: {
        do   {
             flag[0] = true;
             while   flag [1] ;
                  进程 p0 的临界区代码 CS0;
                  Flag[0] = false  ;
                  进程 p0 的其他代码;
             }
while  （ture）
}

p1: {
       do {
            flag [1] = true;
            while flag[0] ;
            进程 p1 的临界区代码 CS1;
            flag [1]   = flase  ;
            进程 p1 的其他代码;
       }
while  （ture）
}
```

十.（15 分）请参考多级反馈轮转法设计一个处理机调度算法，要求至少有三个反馈队列，既考虑了短作业优先，又照顾了大中型作业。