2014 年计算机学科专业基础综合试题参考答案

一、单项选择题

1.	C	2.	B	3.	A	4.	D	5.	C	6.	D	7.	D	8.	D
9.	D	10.	B	11.	C	12.	D	13.	C	14.	A	15.	A	16.	D
17.	A	18.	C	19.	C	20.	C	21.	D	22.	B	23.	A	24.	B
25.	D	26.	A	27.	A	28.	C	29.	B	30.	A	31.	C	32.	D
33.	C	34.	B	35.	D	36.	C	37.	B	38.	A	39.	B	40.	D

二、综合应用题

41．解答：

1）算法的基本设计思想：

① 基于先序递归遍历的算法思想是用一个 static 变量记录 wpl，把每个结点的深度作为递归函数的一个参数传递，算法步骤如下：

若该结点是叶子结点，那么变量 wpl 加上该结点的深度与权值之积；

若该结点非叶子结点，那么若左子树不为空，对左子树调用递归算法，若右子树不为空，对右子树调用递归算法，深度参数均为本结点的深度参数加 1；

最后返回计算出的 wpl 即可。

② 基于层次遍历的算法思想是使用队列进行层次遍历，并记录当前的层数，

当遍历到叶子结点时，累计 wpl；

当遍历到非叶子结点时对该结点的把该结点的子树加入队列；

当某结点为该层的最后一个结点时，层数自增 1；

队列空时遍历结束，返回 wpl。

2）二叉树结点的数据类型定义如下：

```
typedef struct BiTNode{
int weight;
struct BiTNode *lchild,*rchild;
}BiTNode,*BiTree;
```

3）算法代码如下：

① 基于先序遍历的算法：

```
int WPL(BiTree root){
return wpl_PreOrder(root, 0);
}
int wpl_PreOrder(BiTree root, int deep){
static int wpl = 0;                                    //定义一个 static 变量存储 wpl
if(root->lchild == NULL && root->lchild == NULL)   //若为叶子结点，累积 wpl
wpl += deep*root->weight;
if(root->lchild != NULL)                               //若左子树不空，对左子树递归遍历
wpl_PreOrder(root->lchild, deep+1);
if(root->rchild != NULL)                          //若右子树不空，对右子树递归遍历
wpl_PreOrder(root->rchild, deep+1);
return wpl;
}
```

② 基于层次遍历的算法：

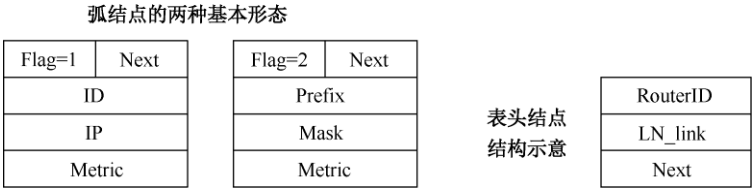
```
#define MaxSize 100            //设置队列的最大容量
int wpl_LevelOrder(BiTree root){
```

```
BiTree q[MaxSize];              //声明队列，end1 为头指针，end2 为尾指针
int end1, end2;                 //队列最多容纳 MaxSize-1 个元素
end1 = end2 = 0;                //头指针指向队头元素，尾指针指向队尾的后一个元素
int wpl = 0, deep = 0;          //初始化 wpl 和深度
BiTree lastNode;                //lastNode 用来记录当前层的最后一个结点
BiTree newlastNode;             //newlastNode 用来记录下一层的最后一个结点
lastNode = root;                //lastNode 初始化为根节点
newlastNode = NULL;             //newlastNode 初始化为空
q[end2++] = root;               //根节点入队
while(end1 != end2){            //层次遍历，若队列不空则循环
  BiTree t = q[end1++];         //拿出队列中的头一个元素
  if(t->lchild == NULL & t->lchild == NULL){
      wpl += deep*t->weight;
      }                         //若为叶子结点，统计 wpl
  if(t->lchild != NULL){        //若非叶子结点把左结点入队
      q[end2++] = t->lchild;
      newlastNode = t->lchild;
      }                         //并设下一层的最后一个结点为该结点的左结点
  if(t->rchild != NULL){        //处理叶节点
      q[end2++] = t->rchild;
      newlastNode = t->rchild;
      }
  if(t == lastNode){            //若该结点为本层最后一个结点，更新 lastNode
      lastNode = newlastNode;
      deep += 1;                //层数加 1
      }
  }
return wpl;                     //返回 wpl
}
```

42．解答：

1）题中给出的是一个简单的网络拓扑图，可以抽象为无向图。

2）链式存储结构的如下图所示。

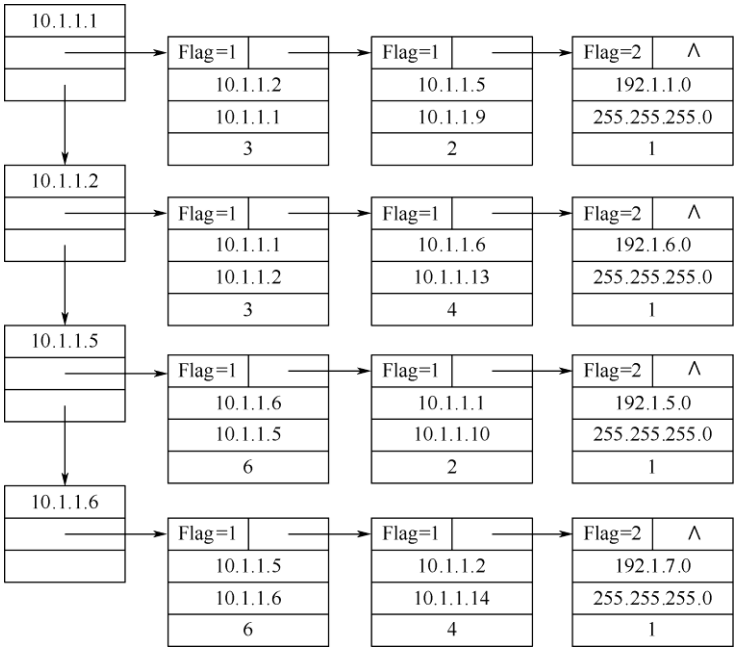


其数据类型定义如下：

```
typedef struct{
unsigned int ID, IP;
}LinkNode;     //Link 的结构
typedef struct{
unsigned int Prefix, Mask;
}NetNode;      //Net 的结构
typedef struct Node{
int Flag;      //Flag=1 为 Link;Flag=2 为 Net
union{
  LinkNode Lnode;
  NetNode Nnode
}LinkORNet;
unsigned int Metric;
struct Node *next;
}ArcNode;      //弧结点
typedef struct HNode{
unsigned int RouterID;
ArcNode *LN_link;
Struct HNode *next;
```

```
}HNODE;          //表头结点
```

对应题 42 表的链式存储结构示意图如下。



3）计算结果如下表所示。

	目的网络	路径	代价（费用）
步骤 1	192.1.1.0/24	直接到达	1
步骤 2	192.1.5.0/24	R1→R3→192.1.5.0/24	3
步骤 3	192.1.6.0/24	R1→R2→192.1.6.0/24	4
步骤 4	192.1.7.0/24	R1→R2→R4→192.1.7.0/24	8

43．解答：

1）因为题目要求路由表中的路由项尽可能少，所以这里可以把子网 192.1.6.0/24 和 192.1.7.0/24 聚合为子网 192.1.6.0/23。其他网络照常，可得到路由表如下：

目的网络	下一条	接　口
192.1.1.0/24	—	E0
192.1.6.0/23	10.1.1.2	L0
192.1.5.0/24	10.1.1.10	L1

2）通过查路由表可知：R1 通过 L0 接口转发该 IP 分组。(1 分)因为该分组要经过 3 个路由器(R1、R2、R4)，所以主机 192.1.7.211 收到的 IP 分组的 TTL 是 64-3=61。

3）R1 的 LSI 需要增加一条特殊的直连网络，网络前缀 Prefix 为“0.0.0.0/0”, Metric 为 10。

44．解答：

1）已知计算机 M 采用 32 位定长指令字，即一条指令占 4B，观察表中各指令的地址可知，每条指令的地址差为 4 个地址单位，即 4 个地址单位代表 4B，一个地址单位就代表了 1B，所以该计算机是按字节编址的。

2）在二进制中某数左移二位相当于乘以四，由该条件可知，数组间的数据间隔为 4 个地址单位，而计算机按字节编地址，所以数组 A 中每个元素占 4B。

3）由表可知，bne 指令的机器代码为 1446FFFAH，根据题目给出的指令格式，后 2B 的内容为 OFFSET 字段，所以该指令的 OFFSET 字段为 FFFAH，用补码表示，值为-6。当系统执行到 bne 指令时，PC 自动加 4，PC 的内容就为 08048118H，而跳转的目标是 08048100H，两者相差了 18H，即 24 个单位的地址间隔，所以偏移地址的一位即是真实跳转地址的-24/-6=4 位。可知 bne 指令的转移目标地址计算公式为(PC)+4+OFFSET*4。

4）由于数据相关而发生阻塞的指令为第 2、3、4、6 条，因为第 2、3、4、6 条指令都与各自前一条指令发生数据

相关。

第 6 条指令会发生控制冒险。

当前循环的第五条指令与下次循环的第一条指令虽然有数据相关，但由于第 6 条指令后有 3 个时钟周期的阻塞，因而消除了该数据相关。

45．解答：

1）R2 里装的是 i 的值，循环条件是 i<N(1000)，即当 i 自增到不满足这个条件时跳出循环，程序结束，所以此时 i 的值为 1000。

2）Cache 共有 16 块，每块 32 字节，所以 Cache 数据区的容量为 16*32B=512B。

P 共有 6 条指令，占 24 字节，小于主存块大小(32B)，其起始地址为 0804 8100H，对应一块的开始位置，由此可知所有指令都在一个主存块内。读取第一条指令时会发生 Cache 缺失，故将 P 所在的主存块调入 Cache 某一块，以后每次读取指令时，都能在指令 Cache 中命中。因此在 1000 次循环中，只会发生 1 次指令访问缺失，所以指令 Cache 的命中率为：(1000×6-1)/(1000×6)=99.98%。

3）指令 4 为加法指令，即对应 sum+=A[i]，当数组 A 中元素的值过大时，则会导致这条加法指令发生溢出异常；而指令 2、5 虽然都是加法指令，但他们分别为数组地址的计算指令和存储变量 i 的寄存器进行自增的指令，而 i 最大到达 1000，所以他们都不会产生溢出异常。

只有访存指令可能产生缺页异常，即指令 3 可能产生缺页异常。

因为数组 A 在磁盘的一页上，而一开始数组并不在主存中，第一次访问数组时会导致访盘，把 A 调入内存，而以后数组 A 的元素都在内存中，则不会导致访盘，所以该程序一共访盘一次。

每访问一次内存数据就会查 TLB 一次，共访问数组 1000 次，所以此时又访问 TLB1000 次，还要考虑到第一次访问数组 A，即访问 A[0]时，会多访问一次 TLB（第一次访问 A[0]会先查一次 TLB，然后产生缺页，处理完缺页中断后，会重新访问 A[0]，此时又查 TLB），所以访问 TLB 的次数一共是 1001 次。

46．解答：

1）系统采用顺序分配方式时，插入记录需要移动其他的记录块，整个文件共有 200 条记录，要插入新记录作为第 30 条，而存储区前后均有足够的磁盘空间，且要求最少的访问存储块数，则要把文件前 29 条记录前移，若算访盘次数移动一条记录读出和存回磁盘各是一次访盘，29 条记录共访盘 58 次，存回第 30 条记录访盘 1 次，共访盘 59 次。

F 的文件控制区的起始块号和文件长度的内容会因此改变。

2）文件系统采用链接分配方式时，插入记录并不用移动其他记录，只需找到相应的记录，修改指针即可。插入的记录为其第 30 条记录，那么需要找到文件系统的第 29 块，一共需要访盘 29 次，然后把第 29 块的下块地址部分赋给新块，把新块存回内存会访盘 1 次，然后修改内存中第 29 块的下块地址字段，再存回磁盘，一共访盘 31 次。

4 个字节共 32 位，可以寻址 2^{32}=4GB 块存储块，每块的大小为 1KB，即 1024B，其中下块地址部分占 4B，数据部分占 1020B，那么该系统的文件最大长度是 4G×1020B=4080GB。

47．解答：

这是典型的生产者和消费者问题，只对典型问题加了一个条件，只需在标准模型上新加一个信号量，即可完成指定要求。

设置四个变量 mutex1、mutex2、empty 和 full，mutex1 用于一个控制一个消费者进程一个周期(10 次)内对于缓冲区的控制，初值为 1，mutex2 用于进程单次互斥的访问缓冲区，初值为 1，empty 代表缓冲区的空位数，初值为 0，full 代表缓冲区的产品数，初值为 1000，具体进程的描述如下：

```
semaphore mutex1=1;
semaphore mutex2=1;
semaphore empty=n;
semaphore full=0;
producer(){
  while(1){
      生产一个产品;
      P(empty);                  //判断缓冲区是否有空位
      P(mutex2);                 //互斥访问缓冲区
      把产品放入缓冲区;
      V(mutex2);                 //互斥访问缓冲区
      V(full);                   //产品的数量加 1
```

```
  }
}

consumer(){
while(1){
  P(mutex1)                        //连续取 10 次
      for(int i = 0; i <= 10; ++i){
          P(full);                 //判断缓冲区是否有产品
          P(mutex2);               //互斥访问缓冲区
          从缓冲区取出一件产品;
          V(mutex2);               //互斥访问缓冲区
          V(empty);                //腾出一个空位
          消费这件产品;
      }
  V(mutex1)
  }
}
```