数据结构第二章参考答案

// 有错误欢迎指正

// 注意对顺序表结构和链表结构的具体操作

算法思想如下(符合意思即可)

2.3 任写一个排序法即可

将x放在La末尾，然后依次与前一个元素比较大小，若小于则交换，若大于或者等于则退出

2.5 设置一个元素为temp，然后依次交换首尾(i为小端计数,j为大端计数)，退出循环条件为i≥j

```
typedef struct{
    ElemType* elem;
    int length;
    int listsize;
}SqList;

void ListReverse_Sq(SqList &L) {
    ElemType temp;
    for(int i = 0;i < L.length/2; i++) {
       temp = *(L.elem+i);
       *(L.elem+i) = *(L.elem+L.length-1-i);
       *(L.elem+L.length-1-i) = temp;
    }
}
```

2.6 设置两个指针，然后一个用于实现指针交换，一个用于设置退出循环条件

```
typedef struct LNode{
    ElemType      data;
    struct LNode *next;
} LNode, *LinkList;

void Inverse(LinkList &L)
/* 对带头结点的单链表L实现就地逆置 */
{
    LNode *curr,*next;
    curr = L->next;
    L->next = NULL;
    while(curr != NULL) {
```

```
        next = curr->next;
        curr->next = L->next;
        L->next = curr;
        curr = next;
    }
}
```

2.9 从表尾到表头逆向建立单链表 C，依次比较 A 和 B 中的元素，小的插入新表的表头处，这样从表尾到表头生成的链表元素即是按从大到小的顺序。

```
void reverse_merge(LinkList& A,LinkList &B,LinkList &C)
{
    pa=A->next;
    pb=B->next;
    pre=NULL; //pa 和 pb 分别指向 A,B 的第一个元素
    while(pa||pb)
    {
        if(pa->data < pb->data || !pb)
        {
            pc=pa;
            q=pa->next;
            pc->next=pre;
            pa=q;
//将 pa 指向的元素插入到 pre 指向结点的前面，由此可保证得到的 pc
链表递减 (pa 指向下一个结点)
        }
        else
        {
            pc=pb;
            q=pb->next;
            pc->next=pre;
            pb=q;
//将 pa 指向的元素插入到 pre 指向结点的前面，由此可保证得到的 pc
链表递减 (pa 指向下一个结点)
        }
        pre=pc; //pre 指针指向前一个结点
    }
    C=A;
    A->next=pc; //构造新表头
}//reverse_merge
```

2.10 设有一个长度大于 1 的单向循环链表，表中既无头结点，也无头指针，s 为指向表中某个节点的指针，编写一个算法，删除链表中指针 s 所指节点的直接前驱。

```
Status Delete_Pre(CiLNode *s)//删除单循环链表中结点 s 的直接
前驱
{
    CiLNode *p;
    p=s;
    while(p->next->next!=s)
        p=p->next; //找到 s 的前驱的前驱 p
    free(p->next);
    p->next=s;
    return OK;
}
```

2.12 分离元素的思想进行解决，从两头向中间遍历，当i<j(没有经过中间元素)时，对从两边分别找到的偶数和奇数进行交换，最后即可得到结果

```
void moveeven(sqlist *l){
    int i=0,j=l->length-1;
    elemtype temp;
    while(i<j){
        while(i<j&&l->data[j]%2==0)
        j--;
        while(i<j&&l->data[i]%2!=0)
        i++;
        if(i<j){//未到达中间时进行交换
            temp=l->data[j];
            l->data[j]=l->data[i];
            l->data[i]=temp;
        }
    }
    return 0;
}
```

3.4

(1). (8,7,6,5,4,3,2,1)

(2). (8,7,6,4,3,2,1)

3.6

```
Status Compare(){
	char c,e;
	int flag = 0;
	SqStack S;
	InitStack(S);
	printf("请输入字符序列:\n");
	while((c = getchar()) != '@'){
		if(c == '&')
			break;
		Push(S,c);
	}
	while((c = getchar()) != '@'){
		if(Pop(S,e) == ERROR || c != e)
			return FALSE;
	}

	if(isEmpty(S))
		return TRUE;
	else
		return FALSE;
}
```

3.7

```
Status Compare(){
	char c,e;
	SqStack S;
	InitStack(S);
	printf("请输入算术表达式:\n");
	while((c = getchar()) != '\n'){
		switch(c){
			case '(':			//左括号，直接入栈
			case '[':
			case '{':
				Push(S,c);
				break;
			case ')':			//右括号，与栈顶元素做比对
				if(Pop(S,e) == ERROR || e != '(')
					return FALSE;
				break;
			case ']':
				if(Pop(S,e) == ERROR || e != '[')
```

```
                return FALSE;
            break;
        case '}':
            if(Pop(S,e) == ERROR || e != '{')
                return FALSE;
            break;
        default :
            break;
        }
    }
    if(isEmpty(S))
        return TURE;
    else
        return FALSE;
}
```

3.8

```
#define OPTR_NUM 7

static char OP[8] = "+-*/()#";
static int PrecedeTable[7][7] = {{1,1,-1,-1,-1,1,1},
                                 {1,1,-1,-1,-1,1,1},
                                 {1,1,1,1,-1,1,1},
                                 {1,1,1,1,-1,1,1},
                                 {-1,-1,-1,-1,-1,0,0},
                                 {1,1,1,1,0,1,1},
                                 {-1,-1,-1,-1,-1,0,0}};

int Precede(char o1,char o2){
    int indx1,indx2;
    for(int i = 0; i < OPTR_NUM; i++){
        if(OP[i] == o1)
            indx1 = i;
        if(OP[i] == o2)
            indx2 = i;
    }
    return PrecedeTable[indx1][indx2];
}

int In(char c){
    for(int i = 0; i < OPTR_NUM; i++){
        if(OP[i] == c)
            return TRUE;
    }
    return FALSE;
```

```
}

void main(){
	char c,e;
	SqStack s1,s2;
	char OP[7] = ")*/+-(";		//运算符号
	InitStack(s1);			//初始化两个栈
	InitStack(s2);
	printf("请输入表达式:\n");
	while((c = getchar()) != '\n'){
		if(c == '(')	//左括号入栈 s1
			Push(s1,c);
		else if(c == ')'){			//右括号，将距离 s1 栈顶最近的“（”之间的运算符，
						//逐个出栈，依次送入 s2 栈,抛弃“（”
			while(Pop(s1,e) != ERROR && e != '(')
				Push(s2,e);
		}
		else if(!In(c))		//操作数，直接进 s2 栈
			Push(s2,c);
		else{					//操作符，从栈顶开始，将 S1 栈所有优先级比 c
					高的运算符入 s2 栈，然后 c 进 s1 栈
			while(GetTop(s1,e) != ERROR && Precede(e,c) > 0){
				Pop(s1,e);
				Push(s2,e);
			}
			Push(s1,c);
		}
	}
	while(Pop(s1,e) != ERROR)	//s1 栈还有元素
		Push(s2,e);
}
```

3.9

假设逆波兰式用 s 表示。

```
InitStack(S);
i = 0;
while(s[i] != ‘\0’){
	if(!In(s[i],OP))		//操作数，直接入栈
		Push(S,s[i]);
	else{
		Pop(S,b);
		Pop(S,a);
		Push(S,Operate(a,s[i],b));
	}
	i++;
```

```
}
```

3.10

```
Status InitQueue(LinkQueue &Q){
    //带头结点，先分配头结点
    Q.rear = (QueuePtr) malloc(sizeof(QNode));
    if(!Q.rear) exit(OVERFLOW);
    Q.rear->next = Q.rear;
    return OK;
}

Status EnQueue(LinkQueue &Q, QElemType e){
    p = (QueuePtr) malloc(sizeof(QNode));
    if(!p) exit(OVERFLOW);
    p->data = e; p->next = Q.rear->next;    //插入队尾
    Q.rear->next = p;
    Q.rear = p;
    return OK;
}

Status DeQueue(LinkQueue &Q, QElemType &e){
    if(Q.rear->next == Q.rear) return ERROR;
    p = Q.rear->next->next;        //队首结点
    e = p->data;
    Q.rear->next->next = p->next;     //删除队首结点
    free(p);
    return OK;
}
```

3.11

队满条件：

(Q.rear+1)%MAXQSIZE==(Q.rear-Q.length+MAXQSIZE)%MAXQSIZE；或者 Q.length=MAXQSIZE-1

```
Status EnQueue(SqQueue &Q, QElemType e){
    if((Q.rear+1)%MAXQSIZE == (Q.rear-Q.length+MAXQSIZE)%MAXQSIZE) //队满
        return ERROR;
    Q.rear = (Q.rear+1)%MAXQSIZE;
    Q.length++;
    Q.base[Q.rear] = e;
    return OK;
}

Status DeQueue(SqQueue &Q, QElemType &e){
    if(Q.length == 0) return ERROR;
    e = Q.base[(Q.rear-Q.length+MAXQSIZE)%MAXQSIZE];
    Q.length--;
```

```
        return OK;
}
```
3.12
```
Status Match(){
        InitStack(S);
        InitQueue(Q);
        while((c=getchar()) != '@'){
                Push(S,c);
                EnQueue(Q,c);
        }
        while(!StackEmpty(S)){
                Pop(S,c1);
                DeQueue(Q,c2);
                if(c1 != c2) return FALSE;
        }
        return TRUE;
}
```
3.15 已知求两个正整数 m 与 n 的最大公因子的过程用语言可以表达为反复执行如下操作：第一步： r=m%n；第二步：若 r=0，则返回 n 算法结束，否则，m=n，n=r，返回第一步。

（1）将上述过程用递归函数表示

（2）写出求解该函数的非递归算法

（1）int func(int x,int y)
```
    {
        return    x%y!=0?func(y,x%y):y;
    }
```
(2)   int func (int m,int n)
```
     {
             int    r;
             do {
                r=m%n;
                m=n;
                n=r;
             } while(r!=0)
             return m;
     }
```

5.1

[0,0,0,0]->[0,0,0,1]->[0,0,0,2]->[0,0,1,0]->[0,0,1,1]->…->[1,1,2,0]->[1,1,2,1]->[1,1,2,2]

0 0 0 0, 0 0 0 1, 0 0 0 2,

0 0 1 0, 0 0 1 1, 0 0 1 2,

0 0 2 0, 0 0 2 1, 0 0 2 2,

0 1 0 0, 0 1 0 1, 0 1 0 2,

0 1 1 0, 0 1 1 1, 0 1 1 2,

0 1 2 0, 0 1 2 1, 0 1 2 2,

1 0 0 0, 1 0 0 1, 1 0 0 2,

1 0 1 0, 1 0 1 1, 1 0 12,

1 0 2 0, 1 0 2 1, 1 0 2 2,

1 1 0 0, 1 1 0 1, 1 1 0 2,

1 1 1 0, 1 1 1 1, 1 1 1 2,

1 1 2 0, 1 1 2 1, 1 1 2 2,

5.2

（1）6*48=288

（2）1000+（5*8+7）*6=1282

（3）1000+（2*8+4）*6=1120

（4）1000+（4*6+2）*6=1156

5.3

k=j-(i*i-2*n*i-i)/2-n-1

f1=-(i*i-2*n*i-i)/2, f2=j, c=-(n+1)

5.4

i 为偶数 k=i+j-1; i 为奇数 k=i+j-2

5.5

0,1,1

0,4,5

1,0,2

1,1,3

1,3,6

3,1,4

3,4,7

或者

1,2,1

1,5,5

2,1,2

2,2,3

2,4,6

4,2,4

4,5,7

5.4

对角线上的元素对应

k=$\begin{cases} 2*i-1 & i\text{为偶数} \\ 2*i-2 & i\text{为奇数} \end{cases}$

对任意元素

k=$\begin{cases} i+j-1 & i\text{为偶数} \\ i+j-2 & i\text{为奇数} \end{cases}$

5.6

```
Status AddTSMatrix(TSMatrix A,TSMatrix B, TSMatrix &C){
	C.mu = A.mu; C.nu = A.nu; C.tu = A.tu+B.tu;
	ia = ib = ic = 0;
	while(ia < A.tu && ib < B.tu){
		if(A.data[ia].i < B.data[ib].i){		//找行号较小的
			C.data[ic++] = A.data[ia++];
		}
		else if(A.data[ia].i > B.data[ib].i){
			C.data[ic++] = B.data[ib++];
		}
		else{						//行号相等，找列号较小的
			if(A.data[ia].j < B.data[ib].j){
				C.data[ic++] = A.data[ia++];
			}
			else if(A.data[ia].j > B.data[ib].j){
				C.data[ic++] = B.data[ib++];
			}
			else{					//行号和列号都相等
				e = A.data[ia].e+B.data[ib].e; //做和运算
				if(e != 0){			//和不为 0
					C.data[ic].i = A.data[ia].i; C.data[ic].j = A.data[ia].j; C.data[ic].e = e;

					ic++; ia++; ib++; C.tu--;
				}
				else C.tu -= 2;			//和为 0，结点数减 2
			}
		}
	}
	//把剩余未加入的结点加到 C 里面
	while(ia < A.tu)	C.data[ic++] = A.data[ia++];
	while(ib < B.tu)	C.data[ic++] = B.data[ib++];
	return OK;
}
```

5.7

```
Status PrintSMatrix_OL(CrossList &M){
```

```
	for(i = 1; i <= M.mu; i++){
		p = M.rhead[i]->right;		//第 i 行的行链表
		while(p){				//打印非零元
			cout << p->i << " " << p->j << " " << p->e << endl;
			p = p->right;
		}
	}
}
```

5.8

```
Status PrintSMatrix_OL_ALL(CrossList &M){
	for(i = 1; i <= M.mu; i++){
		p = M.rhead[i]->right;		//第 i 行的行链表
		last = 0;
		while(p){
			for(j = last; j < p->j; j++)	cout << 0 << " ";
			cout << p->e << " ";
			last = p->j+1;
			p = p->right;
		}
		for(j = last; j < M.nu; j++)cout << 0 << " ";
		cout << endl;
	}
}
```

已知 S='This is A program!', T='good ',请写出下列函数的结果;

strLength(S), SubString(S,10,7), Index(S,'A'), Replace(S,"A,"a")

Concat(Concat(SubString(S,0,10),T),Substring(S,10,7))

18,

Program

8

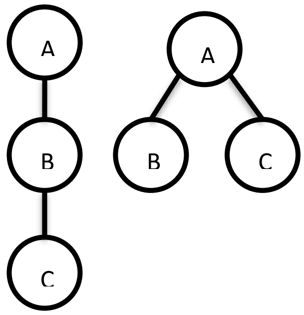
This is a program

This is a good program

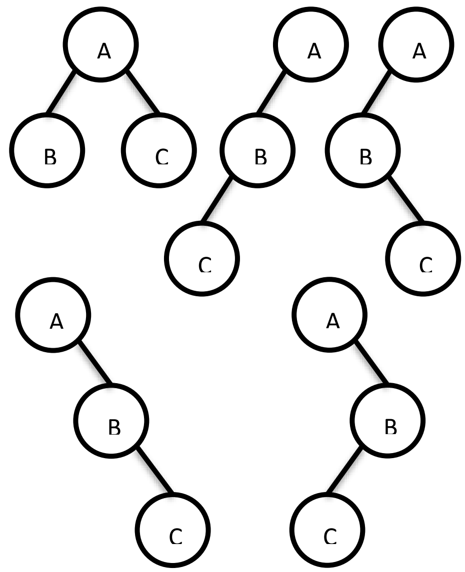
已知模式串 P="abcaabbcab",求其改进后的 next 数组

-1, 0, 0, -1, 1, 0, 2, 0, -1, 0

6.1



具有 3 结点的树



具有 3 结点的二叉树

6.2

2 个子树（空树、本身）

6.3

(1) 不含左子树
(2) 不含右子树
(3) 既不含左子树，也不含右子树

6.4

(1) $\frac{k^H-1}{k-1}$　　k^{i-1}

(2) 如果 i 是其双亲的最小的孩子（右孩子），则 i 减去根结点的一个结点，应是 k 的整数倍，该整数即为所在的组数，每一组为一棵满 k 叉树，正好应为双亲结点的编号。如果 i 是其双亲的最大的孩子(左孩子)，则 i+k-1 为其最小的弟弟，再减去一个根结点，除以 k，即为其双亲结点的编号。

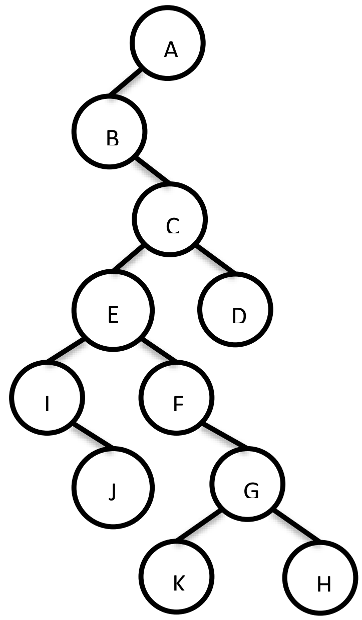
综合来说，对于 i 是左孩子的情况，j=(i+k-2)/k；对于 i 是右孩子的情况，j=(i-1)/k

如果左孩子的编号为 i，则其右孩子编号必为 i+k-1，所以，其双亲结点的编号为$j=\left\lfloor\frac{i+k-2}{k}\right\rfloor$向下取整，如 1.5 向下取整为 1

(3) 结点 i 的右孩子的编号为 ip+1，左孩子的编号为 ki+1-k+1=k(i-1)+2，第 j 个孩子的编号为 k(i-1)+2+j-1=ki-k+j+1

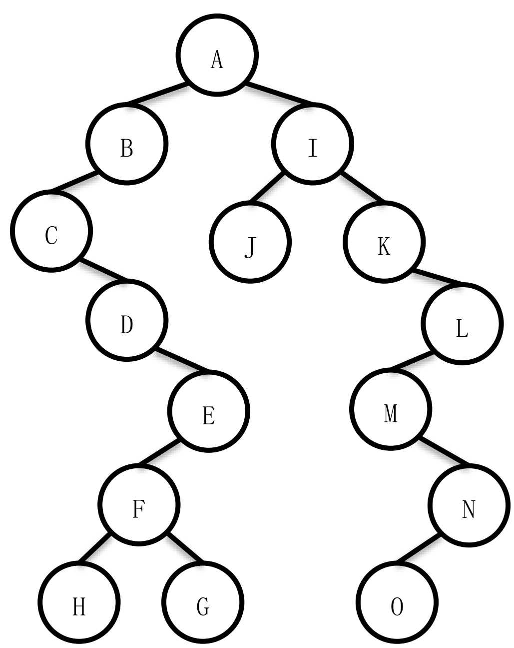
(4) 当(i-1)%k != 0 时，结点 i 有右兄弟，其右兄弟的编号为 i+1。

6.8

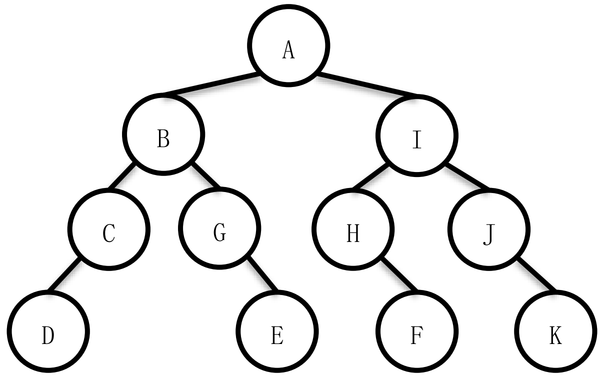


先根：ABCEIJFGKHD
后根：BIJEFKGHCDA

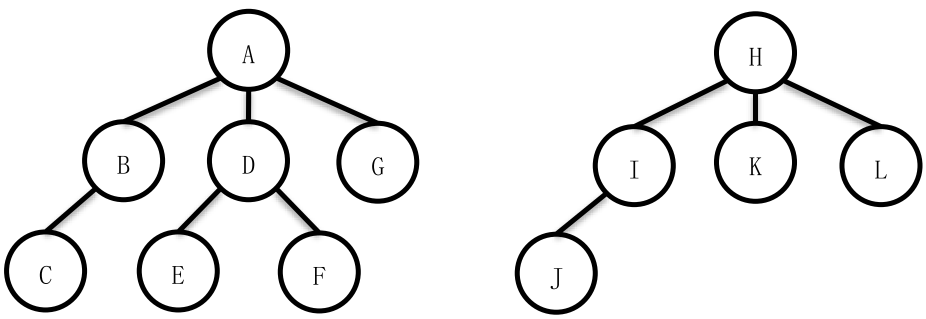
6.9



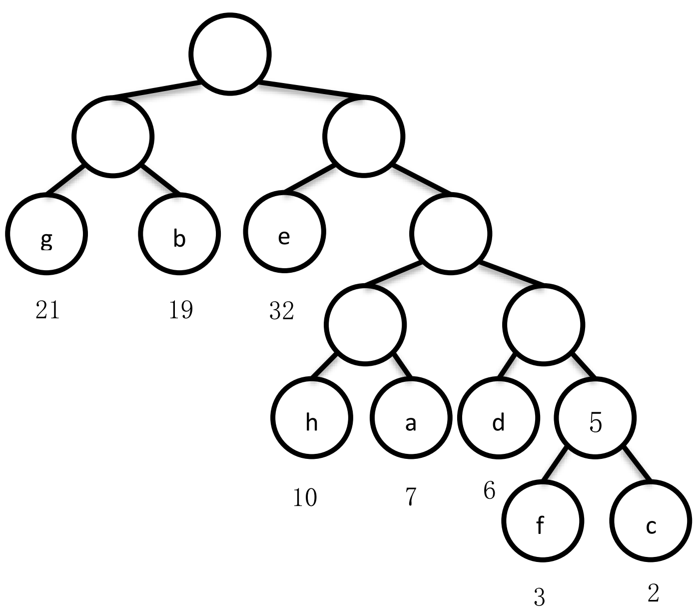
6.11



6.13



6.14



a:1101

b:01

c:11111

d:1110

e:10

f:11110

g:00

h:1100

电文二进制位数：

261=4*7+2*19+5*2+

4*6+2*32+5*3+2*21

+4*10

6.16

```
Status ExchangeBiTree(BiTree &T)
{
    BiTree p;
    if(T){
        p=T->lchild;
        T->lchild=T->rchild;
        T->rchild=p;
        ExchangeBiTree(T->lchild);
        ExchangeBiTree(T->rchild);
    }
    return OK;
}
```

6.22

```
int GetDepth_CSTree(CSTree T)//求孩子兄弟链表表示的树 T 的深度
{
    if(!T) return 0; //空树
    else
    {
        for(maxd=0,p=T->firstchild;p;p=p->nextsib)
            if((d=GetDepth_CSTree(p))>maxd) maxd=d; //子树的最大深度
        return maxd+1;
    }
}//GetDepth_CSTree
```

6.23

```
int GetNodeNum(Bitree T, int k)
{
    if (!T || k<1)
        return 0;
    if (k=1)
        return 1;
    return GetNodeNum(T->lchild, k-1)+ GetNodeNum(T->rchild, k-1);
}
```

《图》作业参考

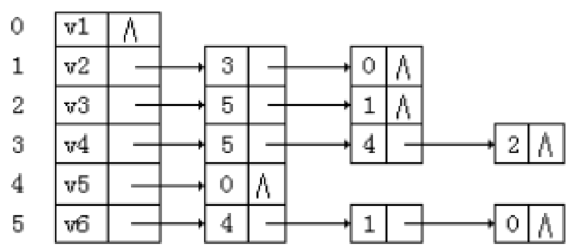
注：以下均是根据电子版作业给出，和书上课后作业有些许不同

7.1

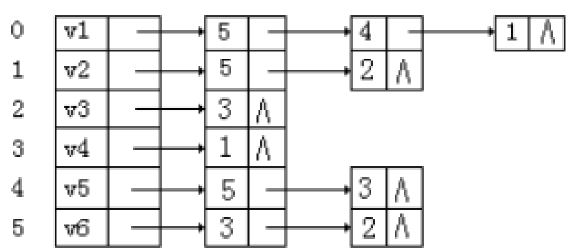
7.1

(1) $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$

(2)



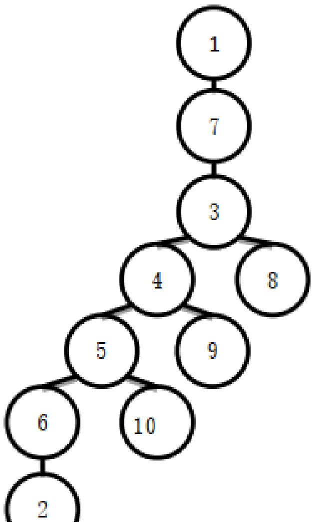
(3)



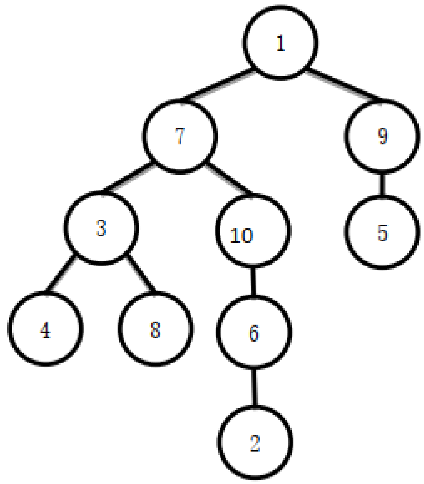
(4) 有三个强连通分量 1、5、2346

7.2

深度优先搜索序列： v1 v7 v3 v4 v5 v6 v2 v10 v9 v8
广度优先搜索序列： v1 v7 v9 v3 v10 v5 v4 v8 v6 v2



深度优先



广度优先

7.3

(1) $$\begin{bmatrix} \infty & 4 & 3 & \infty & \infty & \infty & \infty & \infty \\ 4 & \infty & 5 & 5 & 9 & \infty & \infty & \infty \\ 3 & 5 & \infty & 5 & \infty & \infty & \infty & 5 \\ \infty & 5 & 5 & \infty & 7 & 6 & 5 & 4 \\ \infty & 9 & \infty & 7 & \infty & 3 & \infty & \infty \\ \infty & \infty & \infty & 6 & 3 & \infty & 2 & \infty \\ \infty & \infty & \infty & 5 & \infty & 2 & \infty & 6 \\ \infty & \infty & 5 & 4 & \infty & \infty & 6 & \infty \end{bmatrix}$$

(2)

A→2→1

B→4→3→2→0

C→7→3→1→0

D→7→6→5→4→2→1

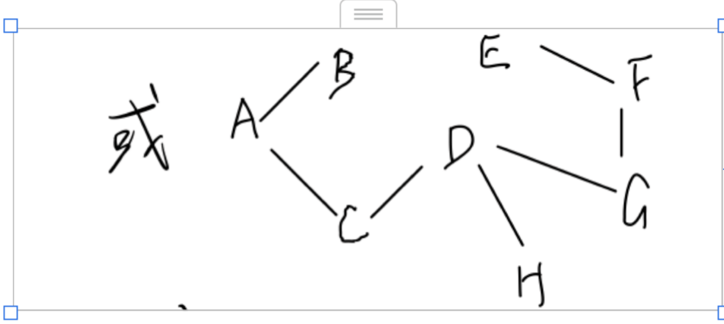
E→5→3→2

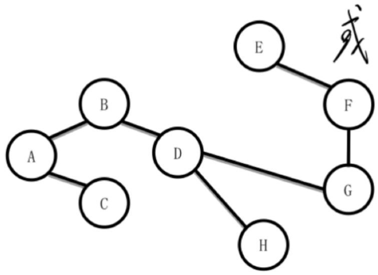
F→6→4→3

G→7→5→3

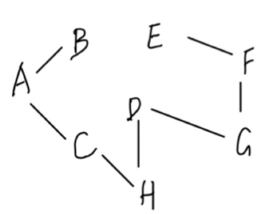
H→6→3→2

（1）和（2）最小生成树共有三种可能（只需写一种）：





或



7.4

- 1， 5， 6， 2， 3， 4
- 5， 1， 6， 2， 3， 4
- 5， 6， 1， 2， 3， 4

7.5

7.5

终点	从 A 到各终点的 D 值和最短路径的求解过程					
	i=1	i=2	i=3	i=4	i=5	i=6
B	15(A,B)	15(A,B)	15(A,B)	15(A,B)	15(A,B)	15(A,B)
C	2(A,C)					
D	12(A,D)	12(A,D)	11(A,C,F,D)	11(A,C,F,D)		
E	∞	10(A,C,E)	10(A,C,E)			
F	∞	6(A,C,F)				
G	∞		16(A,C,F,G)	14(A,C,F,D,G)	14(A,C,F,D,G)	
Vi	C	F	E	D	G	B
S	{A,C}	{A,C,F}	{A,C,E,F}	{A,C,D,E,F}	{A,C,D,E,F,G}	{A,B,C,D,E,F,G }

7.8

```
//int Indegree[]初始化为全0
bool count_indegree(ALGraph G, int Indegree[]){
    int vex_num=G.vexnum;
    int i;
    ArcNode *p=NULL;
    for(i=0;i<vex_num;i++){
        p=G.vertices[i].firstarc;
        while(p!=NULL){
            Indegree[p->advjex]++;
            p=p->nextarc;
        }
    }
    return true;
}
```

7.9

```
//深度优先判断有向图G中顶点i到顶点j是否有路径,是则返回true,否则返回false
int visited[MAXSIZE]; //指示顶点是否在当前路径上,初始化全为0
bool exist_path_DFS(ALGraph G,int i,int j) {
    if(i==j) return true; //找到路径
    else
    {
        visited[i]=1;
        for(p=G.vertices[i].firstarc;p;p=p->nextarc)
        {
            k=p->adjvex;
            if(visited[k]==0)
```

```
                if(exist_path_DFS(G,k,j)==true) return true;//i下游的顶点到j有路径
        }//for
    }//else
    return false;
}//exist_path_DFS
```

7.10

```
//广度优先判断有向图G中顶点i到顶点j是否有路径,是则返回true,否则返回false
bool exist_path_BFS(ALGraph G,int i,int j) {
    int visited[MAXSIZE];
    InitQueue(Q);
    EnQueue(Q,i);
    while(!QueueEmpty(Q))
    {
        DeQueue(Q,u);
        visited[u]=1;
        if(k==j) return true;
        for(p=G.vertices[i].firstarc;p;p=p->nextarc)
        {
            k=p->adjvex;
            if(visited[k]==0) EnQueue(Q,k);
        }//for
    }//while
    return false;
}//exist_path_BFS
```

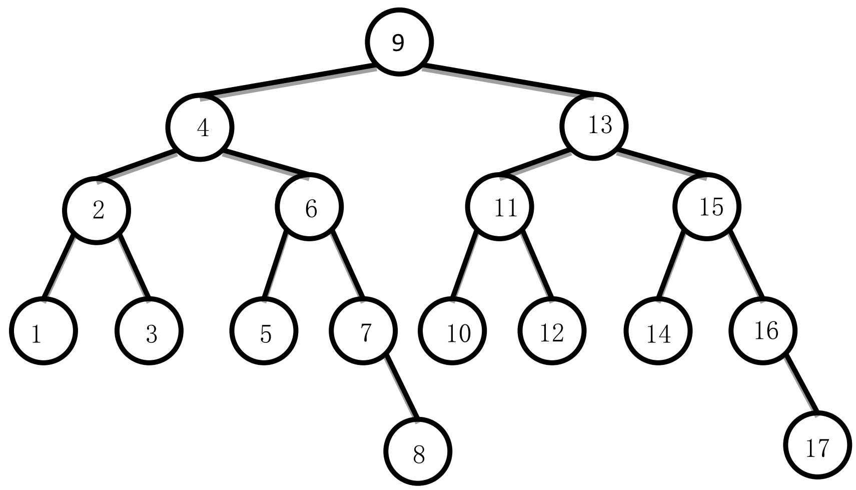
9.2

查找 e: d, f, e

查找 f: d, f

查找 g: d, f, g

9.3



查找成功时：ASL==(1+2*2+4*3+8*4+5*2)/17=3.47 = 59/17

查找失败时：ASL==(2*4*6+2*1*4+2*2*5)/18=4.22 = 38/9

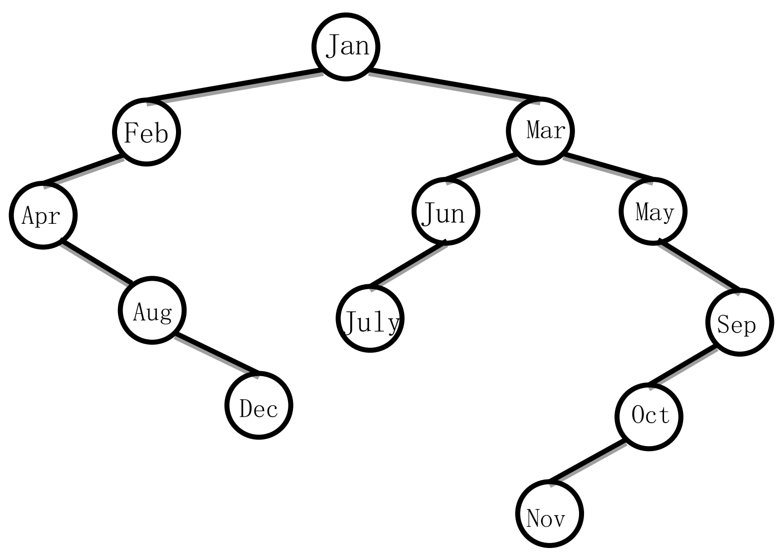
9.4

(1)

0.1*12+0.25*11+0.05*10+0.13*9+0.01*8+0.06*7+0.11*6+0.07*5+0.02*4+0.03*3+0.1*2+0.07*1

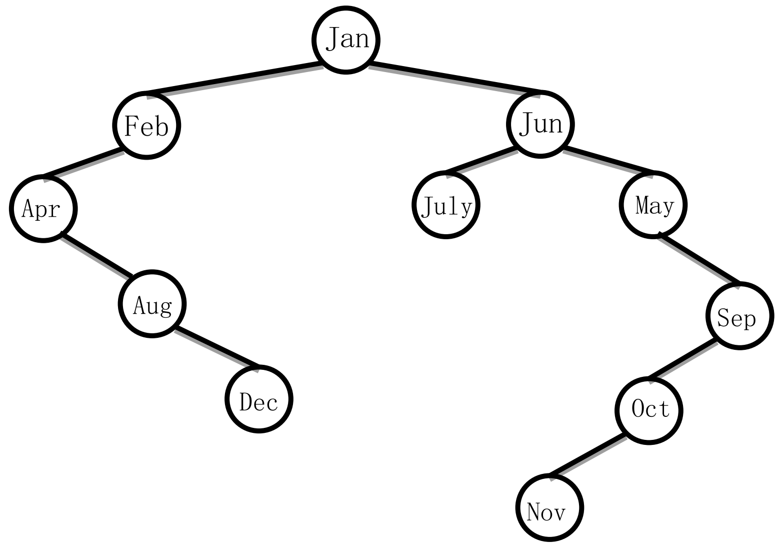
=7.57

(2)



(3) 0.1*1+(0.25+0.05)*2+(0.13+0.01+0.06)*3+(0.11+0.07+0.02)*4+(0.03+0.07)*5+0.1*6=3.2

(4)有两种写法，May 提上来也可以



9.5

(1) HT1

0	1	2	3	4	5	6	7	8	9	10
33	76		25	37	49	6	60	19		10
76					60	60				76

查找成功：ASL=(7*1+2*3)/9=1.44 = 13/9

查找失败：ASL=(3+2+1+7+6+5+4+3+2+1+4)/11=3.45

(2) HT2

0	1	2	3	4	5	6	7	8	9	10
33	60		25	37	49	6		19	76	10
76				60	60	60			60	76

查找成功：ASL=(7*1+3+5)/9=1.67 = 15/9

(3) HT3

0	1	2	3	4	5	6	7	8	9	10
33			25	37	49	6		19		10
					60					76

查找成功：ASL=(7*1+2*2)/9=1.22 = 11/9

9.7

```
bool IsSearchTree(Bitree *T,Elemtype &e;)
{//递归遍历二叉树是否为二叉排序树,e 初始值为最小值
    if(!T)return TRUE;
    retl=IsSearchTree(T->lchild,e);
    if(T->data<e)return FALSE;              //当前元素小于中序序列的前一个元素
    e=T->data;
    retr=IsSearchTree(T->rchild,e);
    return retl&&retr;
}
```

9.2
查找 e: d, f, e
查找 f: d, f
查找 g: d, f, g

9.3
查找成功时：ASL==(1+2*2+4*3+8*4+5*2)/17=3.47 = 59/17
查找失败时：ASL==(2*4*6+2*1*4+2*2*5)/18=4.22 = 38/9

9.4
(1)
0.1*12+0.25*11+0.05*10+0.13*9+0.01*8+0.06*7+0.11*6+0.07*5+0.02*4+0.03*3+0.1*2+0.07*1 =7.57
(3) 0.1*1+(0.25+0.05)*2+(0.13+0.01+0.06)*3+(0.11+0.07+0.02)*4+(0.03+0.07)*5+0.1*6=3.2

9.5
(1) 查找成功：ASL=(7*1+2*3)/9=1.44 = 13/9
查找失败：ASL=(3+2+1+7+6+5+4+3+2+1+4)/11=3.45
(2) 查找成功：ASL=(7*1+3+5)/9=1.67 = 15/9
(3) 查找成功：ASL=(7*1+2*2)/9=1.22 = 11/9

9.7
```
bool IsSearchTree(Bitree *T,Elemtype &e;)
{//递归遍历二叉树是否为二叉排序树,e 初始值为最小值
    if(!T)return TRUE;
    retl=IsSearchTree(T->lchild,e);
    if(T->data<e)return FALSE;                //当前元素小于中序序列的前一个元素
    e=T->data;
    retr=IsSearchTree(T->rchild,e);
    return retl&&retr;
}
```

10.1

(1) (1,5,6,0,9,2,8,3,7,4)

(2) (2,1,3,0,4,5,8,6,7,9)

(3) (4,1,3,0,2,5,8,9,7,6)

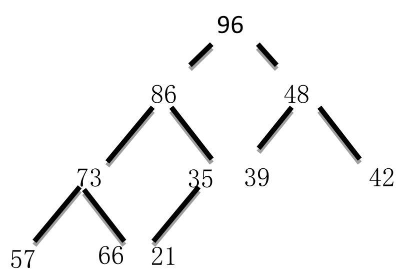
(4) (1,5,0,6,2,8,3,7,4,9)

(5) (1,5,0,6,2,9,3,8,4,7)

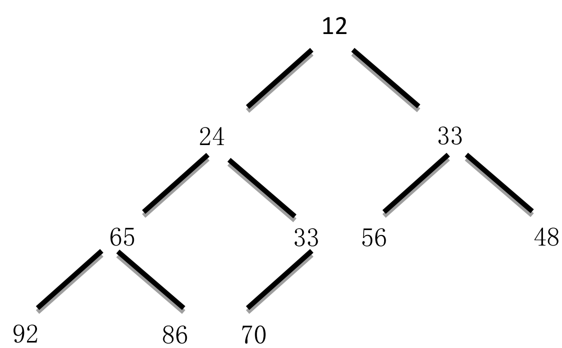
(6) (8,7,6,5,4,2,1,3,0,9)

10.3

(1) 是大顶堆



(2) 不是大顶堆，也不是小顶堆。调整后：



(12,24,33,65,33,56,48,92,86,70)

10.5

```
void LinkedList_Select_Sort(LinkedList&L)//单链表上的简单选择排序算法
{
    for(p=L;p;p=p->next)//p 不为 NULL
    {
        min=p;
        for(q=min->next;q;q=q->next)
            if(min->data > q->data)min=q;
        if(min!=p){t=p->data;p->data=min->data;min->data=t;}
    }
}// LinkedList_Select_Sort
```

1、
数据是信息的符号记录，数据是数据库处理和研究对象
数据库是指存储在计算机内的有组织，可共享的相关数据的集合
数据库管理系统位于用户和操作系统之间的一层数据管理软件
数据库系统计算机硬件为基础的记录保持系统包括数据库数据库、管理系统、应用系统、管理员和用户，有时还包括计算机硬件
2、信息模型是指按照用户的观点对信息建模，相对的
数据模型是按照计算机系统的观点对数据建模
数据模型的三要素：数据结构，数据操作，完整性约束
3、外模式，模式，内模式

4、数据独立性的含义：数据独立性是数据库系统的一个最重要的目标之一。它能使数据独立于应用程序。
物理独立性：是指用户的应用程序与存储在磁盘上的数据库中数据是相互独立的。即，数据在磁盘上怎样存储由 DBMS 管理，用户程序不需要了解，应用程序要处理的只是数据的逻辑结构，这样当数据的物理存储改变了，应用程序不用改变。
逻辑独立性：是指用户的应用程序与数据库的逻辑结构是相互独立的，即，当数据的逻辑结构改变时，用户程序也可以不变。

5、关系数据模型
优点：
1) 建立在严格的数学概念基础上
2) 概念单一，实体、联系均用关系来表示
3) 存取路径对用户透明，数据独立性更高，保密性更好。简化程序员工作和
数据库开发建立工作。
缺点：
1) 存取路径对用户透明，查询效率不高
2) 因存取路径对用户透明，必须对用户查询进行优化，增加了开发 DBMS
的难度。

数据库基础第二章习题答案

1.

（1） 关系是实体集和实体间的联系，通常对应一张表，关系模式则是对关系的描述，关系模式的实例为一个关系

（2）笛卡尔积 ：

定义：$R \times S = \{ t^{\frown}s \mid t \in R,\ s \in S \}$

连接：两关系笛卡尔积中选取属性满足一定条件的元组组成新的关系

$$R \infty S = \{ t_r^{\frown}t_s \mid t_r \in R \wedge t_s \in S \wedge t_r[A]\theta t_s[B] \} \equiv \delta_{i\theta(r+j)}(R \times S)$$

（3）等值连接：

$$R \infty S = \{ t_r^{\frown}t_s \mid t_r \in R \wedge t_s \in S \wedge t_r[A] = t_s[B] \}$$

自然连接：两关系具有相同属性，且在相同的属性上做等值连接，需要取消重复列。

（4）自然连接：同属性等值连接

外连接：R 与 S 做自然连接时，把该舍弃的元组也保存在新关系中，在新增的属性上填 null

2.

(1)

X	Y
a	d
d	a
b	a
c	c
d	c

(2)

X	Y
b	a

(3)

X	Y	X	Y
a	d	d	a
a	d	b	a
a	d	d	c
b	a	d	a
b	a	b	a
c	c	b	a
c	c	d	c
b	a	d	c
c	c	d	a

(4)

X	Y
a	b

(5)

X	Y	Z
a	d	null
b	a	null
c	c	null
null	b	b
null	b	e
null	c	d

(6)

a. 外连接

X	Y	Z	W
a	b	e	f
a	b	c	d
c	a	c	d
null	b	b	null
null	c	d	null

b.左外连接

X	Y	Z	W
a	b	e	f
a	b	c	d
c	a	c	d

c.右外连接

X	Y	Z	W
a	b	e	f
null	b	b	null
null	c	d	null

3. 中文 Mathtype 打不出来，就用英文代替了

(1) $\pi_{cno,cname}(\delta_{teacher=\text{"wangxin"}}(\mathrm{C}))$

(2) $\pi_{sno}(\mathrm{SC}\infty\pi_{cno}(\delta_{cname=\text{"}DB\text{"}\vee cname=\text{"}DBMS\text{"}}(\mathrm{C}))$

(3) $\pi_{cno}(\mathrm{SC}\infty\pi_{sno}(\delta_{sname=\text{"lilin"}}(S)))\infty\pi_{cno,cname}(\mathrm{C})$

（4） $\pi_{\mathrm{sno}}(\delta_{1=4\wedge 2\neq 5}(\mathrm{SC}\times\mathrm{SC}))$

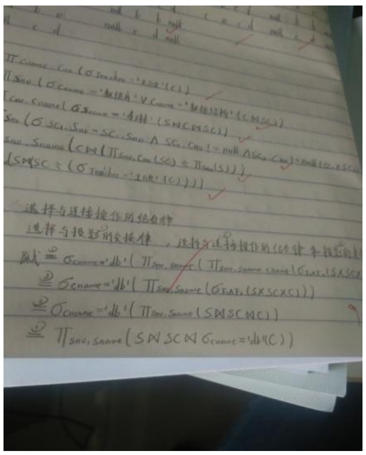
（5） $(\pi_{cno,\mathrm{sno}}(\mathrm{SC})\div\pi_{sno}(S))\infty\pi_{cno,cname}(\mathrm{C})$

（6） $\pi_{\mathrm{sname}}((\pi_{cno,sno}(\mathrm{SC})\div\pi_{cno}(\delta_{teacher="wangxin"}(\mathrm{C})))\infty\mathrm{S})$

4、

（1）成立 选择与连接操作的结合律

（2）成立 选择与投影的交换律，选择与连接的结合律，投影的串接定律



1.
(1) 集合统一
(2) 面向集合的操作方式
(3) 高度非过程化
(4) 统一的语法结构提供两种使用方式 语言简洁，易学易用

2.
简化用户的操作，使用户多角度看待数据，对重构数据库提供了一定的逻辑独立性，能对数据提供安全保护

3.
(1) select sno, grade from s_c where grade<60;
(2) select sname, dept, age from student where age between 19 and 25 order by age desc;
(3) select * from student where sname like ‘%浩%’;
(4) select dept, count(distinct sno) from student group by dept;
(5) select avg(grade), max(grade), min(grade) from s_c where cno = ‘008’;
(6) select cno, avg(grade) from s_c group by cno having avg(grade)>=85;

4.
(1) select cno, cname from c where Teacher = ’张琳’;
(2) select sno from c , sc where sc.cno = c.cno AND c.cname IN (‘C 语言’, ’数据库’);
(3) select cno,cname from s, c , sc where S.sno = sc.sno AND sc.cno = c.cno AND s.sname=’陈浩’;
(4) select distinct sname from s where exists (select * from sc where s.sno=sc.sno and sc.cno=’C1’)and exists (select * from sc where s.sno=sc.sno and sc.cno=’C2’)
或
Select s.sname
From sc,s
Where sc.sno=s.sno and cno=‘C1’ and s.sno in (select
Sno from sc where cno=‘C2')
或
select s.sname
from sc AS X, sc AS Y
where s.sno = sc.sno and X.sno = Y.sno and X.cno = ‘C1’ and Y.cno = ‘C2’

(5) select sname, age from s where EXISTS (select * from sc where sno = s.sno AND cno = ‘C5’)
(6) select sname, sex from s where NOT EXISTS (select * from sc where sno = s.sno AND cno = ‘C3’)

5.
(1) select 姓名,家庭住址 from E where 性别 = ‘女’ and 职务 = ’科长’;
(2) select 姓名,家庭住址 from E,D where E.部门号 = D.部门号 AND 部门名称 = ‘办公室’ and 职务 = ’科长’;
(3) select 姓名，家庭住址 from E,D B where E.部门号 = D.部门号 AND E.职工号 = B.职工号

AND 部门名称 = ‘财务科’ AND 健康状况 = ’良好’;
(4) delete from E where 职工号 = ‘1006’;
(5) update B set 健康状况 = ‘一般’ where 职工号 = ‘1006’;
(6) create view bad_health as

```
Select * from E, B, D
Where B.职工号 = E.职工号 AND E.部门号 = D.部门号 AND 健康状况 = ‘差’;
```

6.
(1) SQL 通信区
(2)主变量
(3)游标

6.

(1) 学生(学号，姓名，出生年月，班级号)

候选键 学号

外键 班级号

(2) 班级(班级号，专业名，人数，入学年份)

候选键 班级号

外键 专业名

(3) 专业(专业号，系号)

候选键 专业号

外键 系号

(4) 系(系号，系名，系办公地点，人数，宿舍区)

候选键 系号，系名

外键 无

(5) 社团(社团名，成立年份，地点，人数)

候选键 社团名

外键 无

(6) 学生_社团(学号，社团名，学生参加社团的年份)

候选键 (学生，社团名)

外键 (学生，社团名)

1、 设关系模式 R(ABCD),F 是 R 上成立的函数依赖集，F={A→C, C→B}，相对于 F 写出关系模式 R 的主关键字。

主关键字：（A,D）

2、 设关系模式 R（ABC），F 是 R 上成立的函数依赖，F={B→C，C→A}，那么 ρ={AB，AC}相对于 F 是否保持无损分解和函数依赖？说明理由。

没有保持函数依赖，丢失函数依赖 B→C，并且 B—^{t}—>A 变成了直接依赖。

3、 关系模式 R（ABCD），F 是 R 上成立的函数依赖，F={AB→CD，A→D}。

1） 试说明 R 不是 2NF 模式的理由；

2） 试把 R 分解成 2NF 模式集。

1） R 的码是（A，B），存在(A,B)—^{p}—> D，所以不满足 2NF

2） R1(A,B,C)　R2(A,D)

4、 设关系模式 R（ABC），F 是 R 上成立的函数依赖，F={C→B，B→A}。

1） 试说明 R 不是 3NF 模式集；

2） 试把 R 分解为 3NF 模式集。

1) R 的码是 C，存在 C—^{t}—>A 故不是 3NF

2) R1（C,B） R2（B,A）

5、 设有关系模式 R（职工名，项目名，工资，部门号，部门经理），如果规定每个职工可以参加多个项目，每个项目都可以各领一份工资；每个项目只属于一个部门管理；每个部门只有一个部门经理。要求：

1） 写出关系模式 R 的函数依赖和主键；

2） R 是 2NF 模式吗？若不是请说明理由，并把 R 分解到 2NF 模式集；

3） 把 R 分解到 3NF 模式集。

1） 主键：（职工号，项目名）

函数依赖：（职工号，项目名）-->工资 ，项目名—>部门，部门-->部门经理

2） 不是 2NF，因为存在非主属性对码的部分函数依赖 （职工号，项目名）--^{p}-->部门

R1（职工号，项目名，工资） R2（项目名，部门，部门经理）

3） （职工号，项目名，工资） R2（项目名，部门） R3(部门，部门经理）