**C语言程序设计练习题**

**答案**

**第1、2、3章 概述、类型、表达式**

**一、选择题**

1-5ADCBD 6-10BACAA 11-15BBDCC 16-20BACAD 21-25CDDDC 26-30CDCCC 31-35BDCBA 36-40DDCBD 41-45BD B C C B

**第4章 顺序结构**

1. **选择题**

1-5DDCDA 6-10BDCBD 11-15ABBAD 16-20DABBB

**第5、6章　选择与循环结构**

**一、选择题**

1-5ACBAC 6-10DAABB 11-15DDBAA 16-20CDADA 21-25DCCDB 26-30CDBDB

**第7章　　数组**

**一、选择题**

1-5DDDCC 6-10DDDCC 11-15DDCBD 16-20CABBB 21-25DBCCC 26-30CDBDD

**第8章　　函　　数**

**一、选择题**

1-5DBCBA 6-10BDCCB 11-15BAADA 16-20BDBCA 21-25BCADC 26-30CABBB 31-35ADCCD 36-40ACDBC

**第9章　　编译预处理**

1. **选择题**

1-5CCCDD 6-10DCCDC 11-15BBDBB

**第10章　　指　针**

1. **选择题**

1-5ADCBA 6-10DBBCB 11-15AACCA 16-20CBCCA 21-25BDBAB 26-30BDADD 31-35DCDAB 36-40DBBDD

**第11章　结构与联合**

**一、单选题**

1-5DCBCD 6-10DDADC 11-15CBBDA 16B

**第12 章 文　　件**

**一、单选题**

**1A 2B 3C C 4B 5D 6-10BBADC 11-15ADCDA**

**数据结构答案**

**第一章**

1.B

2.该程序段的时间复杂度为O(m\*n)。

3.其中语句①的执行次数是1，设语句②的执行次数为f(n)，则有：。得：T（n）=O()

4.B

第二章

1.B 2.B 3.B 4.B 5.A 6.B 7.B 8.D

9. 解：

int length(Linklist L)

{

int I;

listnode \*p;

I=0;

P=L;

while (p->next!=L){

p=p->next;

I++;

}

return I;

}

10解：

void pur\_LinkList(LinkList H)

{ LNode \*p,\*q,\*r; p=H->next; /\*p指向第一个结点\*/

if(p==NULL) return;

while (p->next)

{q=p;

while (q->next) /\* 从\*p的后继开始找重复结点\*/

{ if (q->next->data==p->data)

{ r=q->next; /\*找到重复结点，用r指向，删除\*r \*/

q->next=r->next;

free(r);

} /\*if\*/

else q=q->next;

} /\*while(q->next)\*/

p=p->next; /\*p指向下一个，继续\*/

} /\*while(p->next)\*/

}

该算法的时间性能为O(n2)。

11答：void LinkList\_reverse(LinkList &L)

{

LinkList p,q,s;

p=L->next;q=p->next;s=q->next;p->next=NULL;

while(s->next)

{

q->next=p;p=q;

q=s;s=s->next;

}

q->next=p;s->next=q;L->next=s;

}

12答：void delinsert(LinkList &L)

{

p=L->next; //p是链表的工作指针

pre=L; //pre指向链表中数据域最小值结点的前驱

q=p; //q指向数据域最小值结点，初始假定是第一结点

while(p->next!=NULL)

{

if(p->next->data<q->data) //找到新的最小值结点

{ pre=p; q=p->next; }

p=p->next;

}

if(q!=L->next) //若最小值是第一元素结点，则不需再操作

{

pre->next=q->next; //将最小值结点从链表上摘下

q->next=L->next; //将q结点插到链表最前面

L->next=q;

}

}

13. **解：**假设单链表的表头指针用head表示，其类型为上面定义的LinkList，并且单链表不带头结点。逆置后原来的最后一个结点成为第一个结点，于是从第一个结点开始逐个修改每个结点的指针域进行逆置，且刚被逆置的结点总是新链表的第一个结点，故令head指向它（如图2-1所示）。具体算法描述如下：

图2-1 单链表逆置示意图

示示

图

head

∧

…

head

∧

q

p

head

…

q

(a)单链表初始状态

示示

图

(b)第三个结点逆置

示示

图

void contray(LinkList \*head)

{ //将head单链表中所有结点按相反次序链接

LinkList \*p, \*q;

p=head; //p指向未被逆序的第一个结点,初始时指向原表头结点

head=NULL;

while(p!=NULL)

{ q=p; //q指向将被逆序链接的结点

p=p->next;

q->next=head;

head=q;

}

}

**14.解：**该可以这样考虑，先取开始结点的值，将它与其后的所有结点值一一比较，发现相同的就删除掉，然后再取第二结点的值，重复上述过程直到最后一个结点。

设单链表（其类型为LinkList）的头指针head指向头结点，则可按下列步骤执行：

首先，用一个指针p指向单链表中第一个表结点，然后用另一个指针q查找链表中其余结点元素，由于是单链表，故结束条件为p= =NULL，同时让指针s指向q所指结点的前趋结点，当查找到结点具有q->data= =p->data时删除q所指的结点，然后再修改q，直到q为空；然后使p指针后移（即p=p->next），重复进行，直到p为空时为止。算法描述如下：

del(LinkList \*head)

{ //删除单链表中值相同的多余结点

LinkList \*p, \*s, \*q;

p=head->next;

while(p!=NULL && p->next!=NULL)

{ s=p; //s指向要删除结点的前趋

q=p->next;

while (q!=NULL)

{ if (q->data= =p->data)} //查找值相同的结点并删除

{ s->next=q->next;

free(q);

q=s->next;

}

else

{ s=q;

q=q->next;

}

}

p=p->next;

}

}

**第三、四章**

一、选择

1. C 2. C 3. A 4. B 5.D 6. BA 7.C 8. B 9. C 10. C 11. A 12. A 13.C 14.B 15.D 16.D 17.C 18.D 19. A

二、填空

1. 线性、任何、栈顶、队尾、队首 2. n-i+1 3. n-i 4. n-1

**第六章**

1.B

2. 解： 初始 12 77 21 65 38 7 38 53

1趟 12 77 21 65 38 7 38 53

2趟 12 21 77 65 38 7 38 53

3趟 12 21 65 77 38 7 38 53

4趟 12 21 38 65 77 7 38 53

5趟 7 12 21 38 65 77 38 53

6趟 7 12 21 38 38 65 77 53

7趟 7 12 21 38 38 53 65 77

（ 表示有序区）

3.第一趟：15 65（48 6 34 97 1 26）

第二趟：15 48 65（6 34 97 1 26）

第三趟：6 15 48 65（34 97 1 26）

第四趟：6 15 34 48 65（97 1 26）

第五趟：6 15 34 48 65 97（1 26）

第六趟：1 6 15 34 48 65 97（26）

第七趟：1 6 15 26 34 48 65 97

4.第一趟：15 37 25 10 47 83 32 98

第二趟：15 25 10 37 47 32 83 98

第三趟：15 10 25 37 32 47 83 98

第四趟：10 15 25 32 37 47 83 98

第五趟：10 15 25 32 37 47 83 98

5.第一趟：1（45 8 16 74 27 69 29）

第二趟：1 8（45 16 74 27 69 29）

第三趟：1 8 16（45 74 27 69 29）

第四趟：1 8 16 27（74 45 69 29）

第五趟：1 8 16 27 29（45 69 74）

第六趟：1 8 16 27 29 45（69 74）

第七趟：1 8 16 27 29 45 69 74

6.解：

初始 1趟 2趟 3趟 4趟 5趟

17 17 17 17 7 7

18 18 18 7 17 17

60 40 7 18 18 18

40 7 32 32 32 32

7 32 40 40 40 40

32 60 60 60 60 60

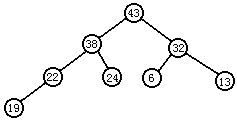
73 65 65 65 65 65

65 73 73 73 73 73

85 85 85 85 85 85

（ 表示有序区）

7. 快速排序的第一趟结果为｛22,19,13,6,24,38,43,12｝；堆排序时所建立的初始大顶堆如所图所示：



两种排序方法所需辅助空间：堆是O(1)，快速排序是O(logn)，可见堆排序所需辅助空间较少；在最坏情况下两种排序方法所需时间：堆是O(nlogn)，快速排序是O(n2)，所以，可见快速排序时间复杂度最差。

* 注意：快速排序的平均时排序速度最快，但在最坏情况下不一定比其他排序方法快。

8. 解：（1）

初始 38 12 21 77 65 7 38 53

↑ ↑

low high

第一次交换

从high开始比较，得到的结果：

7 12 21 77 65 □ 38 53

↑ ↑

low high

从low开始比较，得到的结果：

7 12 21 □ 65 77 38 53

↑ ↑

low high

第二次交换

从high开始比较，得到的结果：

7 12 21 □ 65 77 38 53

↑↑

low high

low=high，所以第一趟快速排序的结果为：

7 12 21 38 65 77 38 53

（2）关键字序列（2，2，1）可以作为一个反例。取第一个关键字作为支点，在一趟快排之后的结果是（1，2，2），由于2已在排序后的最终位置，2在2划分出的前一部分子表中，所以2不可能再出现在2之后，即不可能与原始序列中两者的顺序一致。此反例说明快速排序不是稳定排序。

**9.解析**：(1)采用冒泡排序方法排序的各趟结果如下：(加框为每次冒出元素)

初始序列：25， 18， 60， 40， 7， 21， 32， 73， 68

第——趟：， 25， 18， 60， 40， 21， 32， 68， 73

第二趟： 7， ， 25， 21， 60， 40， 32， 68， 73

第三趟： 7， 18， ， 25， 32， 60， 40， 68， 73

第四趟： 7， 18， 21， ， 32， 40， 60， 68， 73

第五趟： 7， 18， 21， 25， 32， 40， 60， 68， 73

第五趟无元素交换，排序结束。

(2)采用直接选择排序方法排序的各趟结果如下：(加框为每次选出元素)

初始序列：25， 18， 60， 40， ， 21， 32， 73， 68

第——趟：[7] ， 60， 40， 25， 2l， 32， 73， 68

第二趟： [7， 18] 60， 40， 25， ， 32， 73， 68

第三趟： [7， 18， 21] 40， ， 60， 32， 73， 68

第四趟： [7， 18， 21， 25] 40， 60， ， 73， 68

第五趟： [7， 18， 21， 25， 32] 60， ， 73， 68

第六趟： [7， 18， 21， 25， 32， 40] ， 73， 68

第七趟： [7， 18， 21， 25， 32， 40， 60] 73，

第八趟： [7， 18， 21， 25， 32， 40， 60， 68] 73

结束： 7， 18， 21， 25， 32， 40， 60， 68， 73

(3)采用快速排序方法排序的各趟结果如下：

初始序列：25， 18， 60， 40， 7， 21， 32， 73， 68

第一趟： [21 18 7] 25 [40 60 32 73 68]

第二趟： [7 18] 21 25 [32] 40 [60 73 68]

第三趟： 7 [18] 21 25 32 40 60 [73 68]

第四趟： 7 18 21 25 32 40 60 [68] 73

结束： 7 18 21 25 32 40 60 68 73

10. C 11.6

**第七章**

**1** 1. B 2. B 3. C 4. C 5. C 6. A 7. D 8. A 9. C 10. A

11. D 2. A 13. B 14. B 15. B 16. D 17. C 18. C

19. B 20. B 21. B 22. B 23. B 24. A 25. C

**2**

1. ⑴ k1 ⑵ k2,k5,k7,k4 ⑶ 2 ⑷ 3 ⑸ 4 ⑹ k5,k6 ⑺ k1

2. 树的结点个数至少为1(不同教材规定不同)，而二叉树的结点个数可以为0；

树中结点的最大度数没有限制，而二叉树结点的最大度数为2；

树的结点无左、右之分，而二叉树的结点有左、右之分；

3. 树可采用孩子-兄弟链表（二叉链表）做存储结构，目的并利用二叉树的已有算法解决树的有关问题。

e

aE

f

j

c

d

l

g

h

b

图6.9

4. 如图6.9所示

5. 2 k-1 、 2 k-1 、 2 k-2+1

6. n2+1

7. 2 i-1 2[log2n+1]-1 2[log2n+1] –1

8. 只有一个结点的树；空的二叉树

9. 5；如图6.10所示

10. dgbaechif 、abdgcefhi 、gdbeihfca 、

a

图6.10 树形5种

a

a

a



a

c

c

c

c

c

b

b

b

b

b

b

**3** 1. 5种, 图6.11

2. 二叉树如图6.12所示。

图6.11 树形5种

E

BE

F

AE

C

D

K

G

H

I

J

图6.12

3. 中序线索二叉树如图6.13（左）所示；后序线索二叉树如图6.13（右）所示；

该二叉树转换后的的森林如图6.14所示。



图6.13

a

11

d

h

j

b

k

c

图 6.14 对应的森林

i

e

f

4. 图6.8的树转化为一棵二叉树如下，图6.15：

a

b

c

e

d

i

g

图 6.15 一棵树的孩子兄弟表示

5. 画出构造Huffman树如图6.16所示，计算其带权路径长度为 。

62

37

25

19

18

13

12

10

9

6

7

4

5

图6.16 Huffman树

6. 一棵含有N个结点的k叉树,可能达到的最大深度 h=N-k+1 ，

最小深度各为: logkN+1。

4

1. **解：方法是：由前序先确定root，由中序可确定root的左、右子树。然后由其左子树的元素集合和右子树的集合对应前序遍历序列中的元素集合，可继续确定root的左右孩子。将他们分别作为新的root，不断递归，则所有元素都将被唯一确定，问题得解。**

**D A**

**C F**

E G

**B H I**

2.答：算法如下

int BtreeHeight(BinTreeNode \* BT)

{ int h1,h2,h;

if（BT==NULL）h=0；

else{

h1 = BTreeHeight（BT－>leftChild）；

h2 = BTreeHeight（BT－>rightChild）；

if（hl>h2）h=h1＋1；

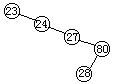
else h=h2＋1；

}

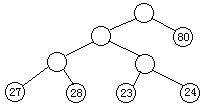
return h;

}

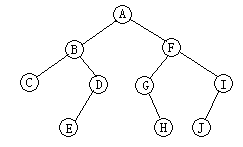
3.（1）二叉排序树如下图所示：



1. 哈夫曼树如下图所示：



4. 先由先序序列的第一个结点确定二叉树的根结点，再由根结点在中序序列中左侧部分为左子树结点，在右侧部分为右子树结点，再由先序序列的第一个结点确定根结点的左右孩子结点，由类似的方法可确定其他结点，如下图所示。



5. 解：第一步，加线。第二步，抹线。第三步，旋转。过程如图5-8所示。

A

图5-8(a) 第一步　加线

B

C

D

E

F

G

H

I

K

L

M

J

A

图5-8(b) 第二步　抹线

B

C

D

E

F

G

H

I

K

L

M

J

A

B

图5-8(c) 第三步　旋转

C

F

D

K

G

E

L

H

M

I

J

6.解： 第一步，加线。第二步，抹线。第三步，调整。过程如图5-10所示。

**A**

B

D

H

C

F

E

J

I

B

A

C

D

E

F

H

I

J

第一步　　　　　　　　　第二步　　　　　　　　　第三步

B

A

C

D

E

F

H

I

J

7. 解： 步骤略，结果如图5-12所示。

C

D

E

F

G

A

B

H

I

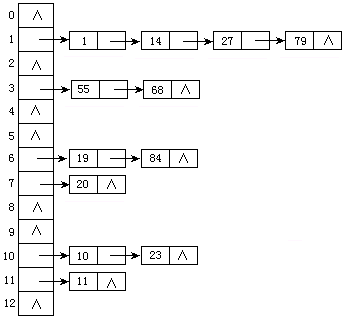
L

J

K

第九章

1. 用链地址法处理冲突的哈希表如下图所示：



ASL=(1\*6+2\*4+3\*1+4\*1)=1.75

2. 答：

线性探测再散列的散列表：

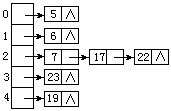
0 1 2 3 4 5 6 7 8 9 10 11 12

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 14 | 1 | 68 | 27 | 55 | 19 | 20 | 84 | 79 | 23 | 11 | 10 |
|  | 1 | 2 | 1 | 4 | 3 | 1 | 1 | 3 | 9 | 1 | 1 | 3 |

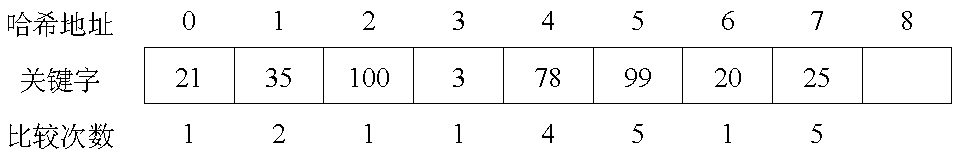
查找成功的平均长度为ASL=1/12（1\*6+2\*1+3\*3+4\*1+9）=2.5

查找不成功的平均长度为ASL=1/13（1+2+3+4…….+13）=7

3. 哈希表如下图所示：



4. 哈希表及查找各关键字要比较的次数如下所示：



ASL=(4×1+1×2+1×4+2×5)=2.5

5. 解：建表如下：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 22 |  | 47 | 92 | 16 | 3 | 7 | 29 | 8 |  |

△ ▲ △ △

分析：47、7、11、16、92均是由哈希函数得到的没有冲突的哈希地址而直接存入的。

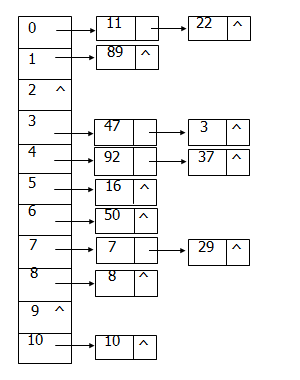
H（29）= 7，哈希地址上冲突，需寻找下一个空的哈希地址：

由Hi =（H（29）+1） MOD 11 = 8 ，哈希地址8为空，将29存入。另外，22、8同样在哈希地址上有冲突，也是由Hi找到空的哈希地址的。

而H（3）= 3，哈希地址上冲突，由：H1=（H（3）+1）MOD 11 = 4，仍然冲突。

H2=（H（3）+2）MOD 11 = 5，仍然冲突。H3=（H（3）+3）MOD 11 = 6，找到空的哈希地址，存入。

6. 解：建表如下：



7. 解：依题意，m=13，线性探测法的下一个地址计算公式为：

di = H（key）

di+1 = （di+1）% m ；i=1，2，…

其计算函数如下：

H（19）= 19 % 13 = 6

H（01）= 01 % 13 = 1

H（23）= 23 % 13 = 10

H（14）= 14 % 13 = 1 （冲突）

H（14）=（1+1）% 14 = 2

H（55）= 55 % 13 = 3

H（20）= 20 % 13 = 7

H（84）= 84 % 13 = 6 （冲突）

H（84）=（6+1）% 14 = 7 （仍冲突）

H（84）=（7+1）% 14 = 8

H（27）= 27 % 13 = 1 （冲突）

H（27）=（1+1）% 14 = 2 （仍冲突）

H（27）=（2+1）% 14 = 3 （仍冲突）

H（27）=（3+1）% 14 = 4

H（68）= 68 % 13 = 3 （冲突）

H（68）=（3+1）% 14 = 4 （仍冲突）

H（68）=（4+1）% 14 = 5

H（11）= 11 % 13 = 11

H（10）= 10 % 13 = 10 （冲突）

H（10）=（10+1）% 14 = 11 （仍冲突）

H（10）=（11+1）% 14 = 12

H（77）= 77 % 13 = 12 （冲突）

H（77）=（12+1）% 14 = 13

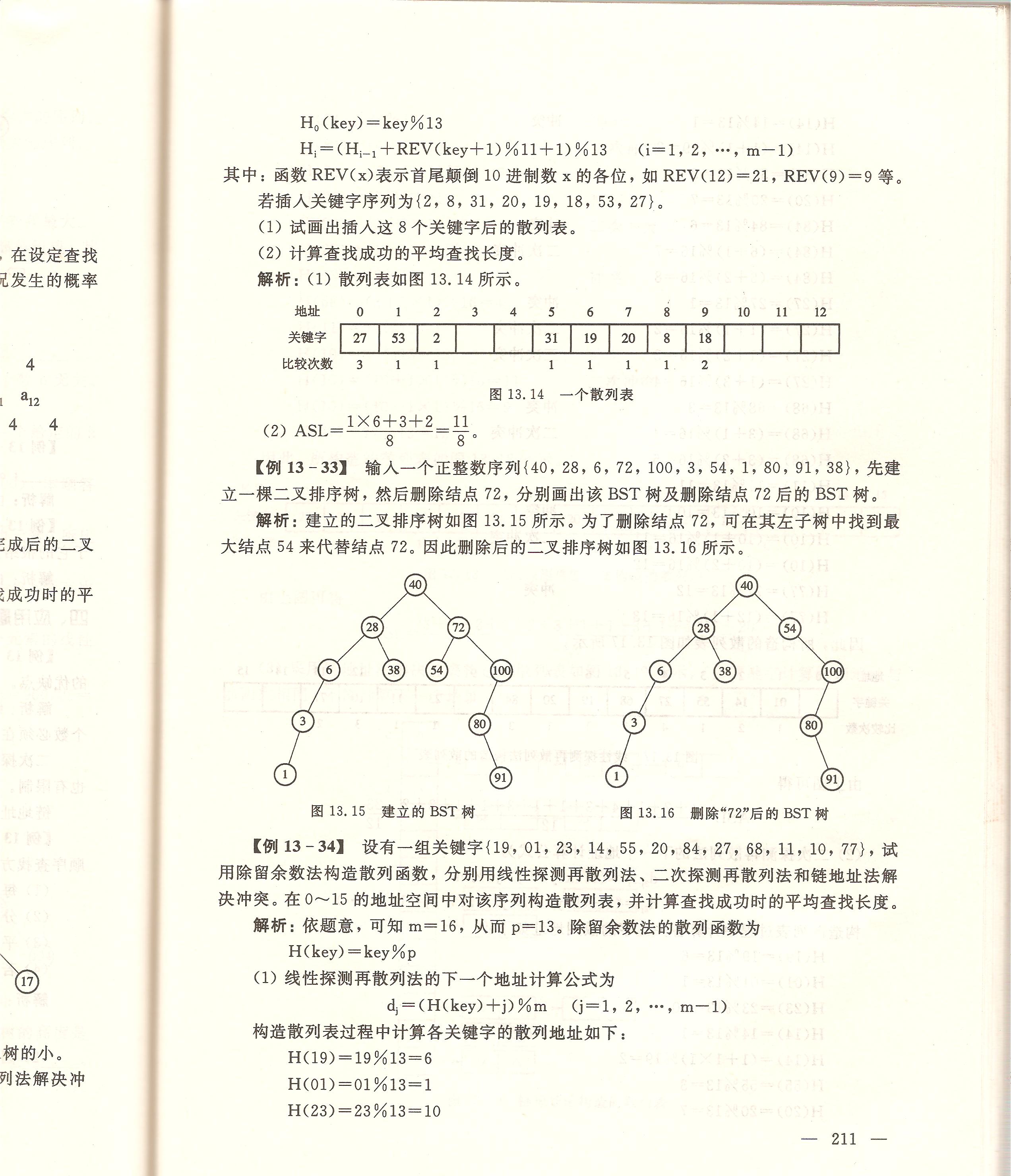
哈希表如下：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 哈希地址 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 关键字 |  | 1 | 14 | 55 | 27 | 68 | 19 | 20 | 84 |  | 23 | 11 | 10 | 77 |
| 比较次数 |  | 1 | 2 | 1 | 4 | 3 | 1 | 1 | 3 |  | 1 | 1 | 3 | 2 |

共有12个关键字，等概率查找，则成功查找时

ASL=（1+2+1+4+3+1+1+3+1+1+3+2）/12=23/12≈1.9

8. **解析**：(1)散列表如图13.14所示。

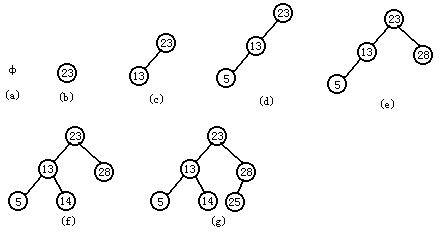


1. **）**

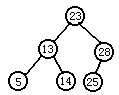
**第十章**

1. 

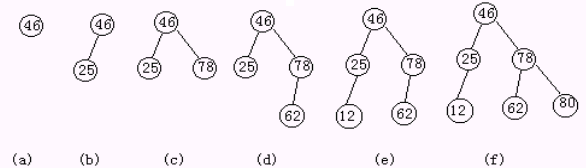
2. 构造二叉排序树的过程如下图所示。



构造的二叉排序树如下图所示：



3. 如下图所示：



4. 解：折半查找的过程描述如下：

① low=1；high=length； //设置初始区间

② 当low>high时，返回查找失败信息 //表空，查找失败

③ low≤high，mid=(low+high)/2; //取中点

a. 若kx<tbl.elem[mid].key，high=mid-1；转② //查找在左半区进行

b. 若kx>tbl.elem[mid].key，low=mid+1；转② //查找在右半区进行

c. 若kx=tbl.elem[mid].key，返回数据元素在表中位置 //查找成功

* 查找关键字为14的过程如下：

0 1 2 3 4 5 6 7 8　 9 10 11 12 13

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 7 | 14 | 18 | 21 | 23 | 29 | 31 | 35 | 38 | 42 | 46 | 49 | 52 |

↑ 　↑

low=1 ①设置初始区间 high=13

───────────────────────────

↑ ②表空测试，非空；

mid=7 　③得到中点，比较测试为a情形

↑ ↑

low=1 　　 high=6 high=mid-1，调整到左半区

────────────────────────────

↑ ②表空测试，非空；

mid=3 ③得到中点，比较测试为a情形

↑ ↑

low=1 high=2 high=mid-1，调整到左半区

────────────────────────────

↑ ②表空测试，非空；

mid=1 ③得到中点，比较测试为b情形

↑↑

low=2 high=2 low=mid+1，调整到右半区

────────────────────────────

↑ ②表空测试，非空；

mid=2 ③得到中点，比较测试为c情形

查找成功，返回找到的数据元素位置为2

* 查找关键字为22的过程如下：

0 1 2 3 4 5 6 7 8 9 10 11 12 13

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 7 | 14 | 18 | 21 | 23 | 29 | 31 | 35 | 38 | 42 | 46 | 49 | 52 |

　　　　　 ↑ ↑

low=1 ①设置初始区间 　　　　 high=13

────────────────────────────

　　　　　　 ↑ ②表空测试，非空；

　　　　 mid=7 ③得到中点，比较测试为a情形

　↑ 　　　　 ↑

low=1 　　　 high=6 high=mid-1，调整到左半区

───────────────────────────

　　↑ ②表空测试，非空；

　　mid=3 ③得到中点，比较测试为b情形

↑ 　　 ↑

low=4 high=6 low=mid+1，调整到右半区

────────────────────────────

↑ ②表空测试，非空；

　 mid=5 ③得到中点，比较测试为a情形

↑↑

low=4 high=4 high=mid-1，调整到左半区

────────────────────────────

↑ ②表空测试，非空；

mid=4 ③得到中点，比较测试为b情形

　↑ ↑

　　high=4 low=5 low=mid+1，调整到右半区

────────────────────────────

②表空测试，为空；查找失败，返回查找失败信息为0

* 查找过程的判定树如图7-1所示。

13

2

1

3

8

4

6

5

7

10

11

9

12

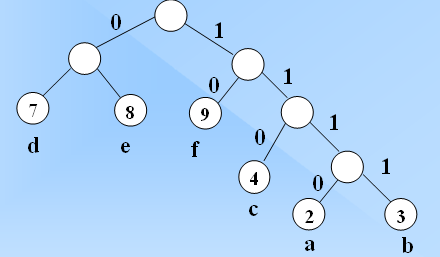
图7-1 折半查找过程的判定树

**第 十一章**

1. **参考答案：**

**WPL= 7\*2+8\*2+9\*2+4\*3+2\*4+3\*4 =80**

**c的编码为110**



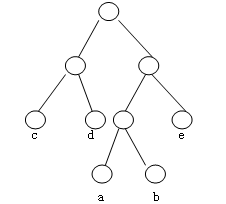
2. 答：



WPL = 3\*4+7\*3+8\*3+2\*4+6\*3+10\*2+14\*2 = 131

3. 答：

构造的哈夫曼树如下：



哈夫曼编码如下：

c(00)

d(01)

a(100)

b(101)

e(11)

平均长度=0.09\*3+0.17\*3+0.2\*2+0.23\*2+0.31\*2=2.26

4. 解： 根据题意，设这8个字母对应的权值分别为（5，25，4，7，9，12，30，8），并且n=8。

（1）设计哈夫曼树的步骤如图5-13所示。

第一步：

25

5

4

7

9

12

30

8

第二步：

25

7

9

12

30

5

4

9

8

第三步：

25

7

9

12

30

5

4

9

8

15

第四步：

25

7

9

12

30

8

15

5

4

9

18

第五步：

25

7

9

12

30

8

15

5

4

9

18

27

第六步：

25

30

9

5

4

9

18

7

12

8

15

27

43

第七步：

25

30

9

5

4

9

18

7

12

8

15

27

43

57

第八步：

25

9

5

4

9

18

43

30

7

12

8

15

27

57

100

（2）设计哈夫曼编码

利用第八步得到的哈夫曼树，规定左分支用0表示，右分支用1表示，字母A、B、C、D、E、F、G、H的哈夫曼编码如下表示：

A:0011 B:01 C:0010 D:1010

E:000 F:100 G:11 H:1011

**解析**：根据给定的权值集合构造的哈夫曼树如图10.19所示。

其加权路径长度WPL＝(9+7+8)×2+4×3十(2+3)×4＝80。

各字符的哈夫曼编码如下：

a 0000

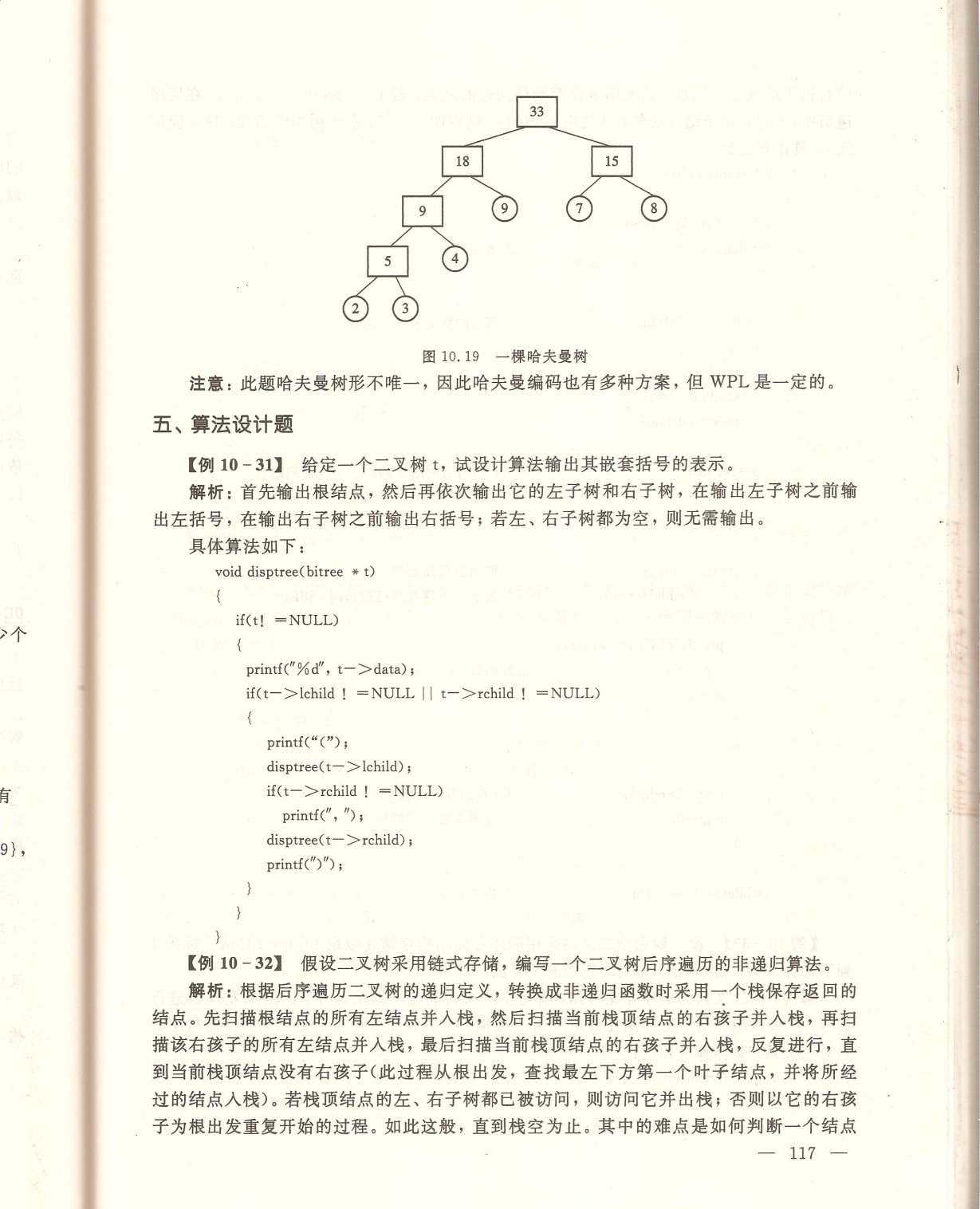
b 0001

c 001

d 10

e 11

f 0l

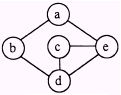


**注意**：此题哈夫曼树形不唯一，因此哈夫曼编码也有多种方案，但WPL是一定的。

**第十二章**

2.【解答】

（1）该图的图形如下图所示：



1. 深度优先遍历序列为：abdce；广度优先遍历序列为：abedc。

3. 解：图G如图6-4中的(a)所示，图G的邻接矩阵和邻接表表示分别如图(b)和(c)所示。

对于这类问题，只要掌握了图的概念和存储结构就可以做出正确的答案。通常情况下．对图的顶点排列顺序和各顶点的邻接点排列顺序并没有特定要求，因此，在写出邻接矩阵和邻接表表示时，只要按照某种排列顺序画出相应的结构图就可以了。但应该注意的是，对于邻接矩阵表示，如果顶点结点的顺序不同，那么邻接矩阵就不相同；对于邻接表表示，如果顶点结点的顺序或者邻接点的顺序不同，那么邻接表就不相同。

0 1 1 1 0 0

0 0 0 0 1 0

0 1 0 0 1 1

0 0 0 0 0 1

0 0 0 0 1 1

0 0 0 0 0 0

(b)

1

(a)

3

4

2

5

6

(c)

1

2

6

4

5

3

2

2

3

5

4

∧

5

∧

6

∧

∧

6

∧

6

∧

4. 解：

（1）该无向图如图6-6所示。

v0

v1

v2

v3

v4

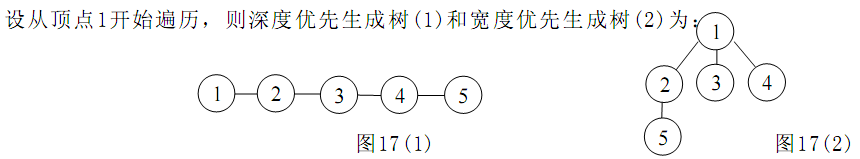
v6

v5

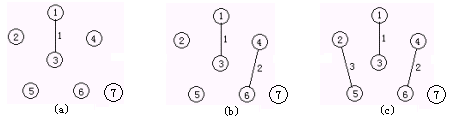
图6-6 无向图

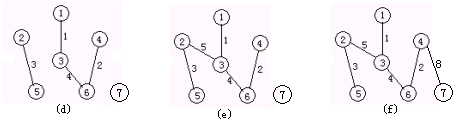
（2）根据该无向图的邻接表表示，从顶点V0开始的深度优先遍历序列为：V0、V2、V3、V1、V4、V6、V5。广度优先遍历序列为V0、V2、V5、V6、V1、V3、V4。

5.



6. 用Kruskal算法构造最小生成树的过程如下图所示：





7. （1）DFS：v1 v2 v3 v4 v5

1. BFS：v2 v3 v4 v5 v1

8.

用克鲁斯卡尔算法得到的最小生成树为：

(1,2)3, (4,6)4, (1,3)5, (1,4)8, (2,5)10, (4,7)20

9. 解：

（1）利用Prim算法从顶点a开始构造最小生成树的过程如图6-9所示。

a

e

f

d

c

b

g

初始状态

a

e

f

d

c

b

g

连通e

2

a

e

f

d

c

b

g

连通g

2

1

a

e

f

d

c

b

g

连通d

2

1

3

3

a

e

f

d

c

b

g

连通f

2

1

4

3

a

e

f

d

c

b

g

连通b

2

1

4

6

用Prim算法构造最小生成树的过程

3

a

e

f

d

c

b

g

连通c

2

1

4

6

1

（2）利用Kruskal算法构造最小生成树的过程如图所示。

a

e

f

d

c

b

g

初始状态

a

e

f

d

c

b

g

增加第2条边

1

1

a

e

f

d

c

b

g

增加第1条边

1

3

a

e

f

d

c

b

g

增加第5条边

2

1

4

1

3

a

e

f

d

c

b

g

增加第4条边

2

1

1

a

e

f

d

c

b

g

增加第3条边

2

1

1

用Kruskal算法构造最小生成树的过程

3

a

e

f

d

c

b

g

增加第6条边

2

1

4

6

1

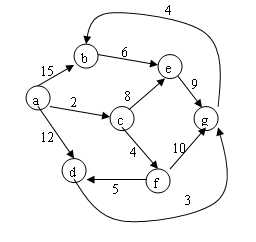
10. 邻接矩阵：

邻接表如下图所示：



11. 答：

（1）



（2）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **终点**  **Dist** | b | c | d | e | f | g | S |
| k=1 | 15  (a,b) | 2  (a,c) | 12  (a,d) |  |  |  | {a,c} |
| k=2 | 15  (a,b) |  | 12  (a,d) | 10  (a,c,e) | 6  (a,c,f) |  | {a,c,f} |
| k=3 | 15  (a,b) |  | 11  (a,c,f,d) | 10  (a,c,e) |  | 16  (a,c,f,g) | {a,c,f,e} |
| k=4 | 15  (a,b) |  | 11  (a,c,f,d) |  |  | 16  (a,c,f,g) | {a,c,f,e,d} |
| k=5 | 15  (a,b) |  |  |  |  | 14  (a,c,f,d,g) | {a,c,f,e,d,g} |
| k=6 | 15  (a,b) |  |  |  |  |  | {a,c,f,e,d,g,b} |

12. 全部可能的拓扑排序序列为：1523634、152634、156234、561234、516234、512634、51236