中山大学软件学院软件工程专业 **20** 级 (2010学年秋季学期)

《SE-211 数据结构与算法》 期末试题参考答案(B)

I、 Selection（15%）

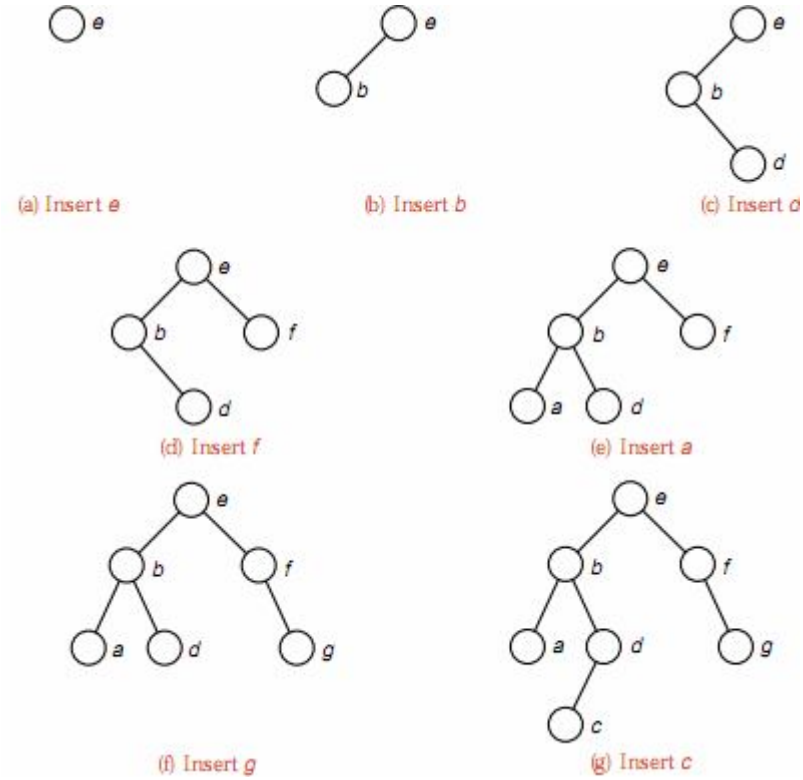
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
C	D	C	C	C	B	B	B	C	B	C	B	A	A	D

II、 Blank Filling（15%）

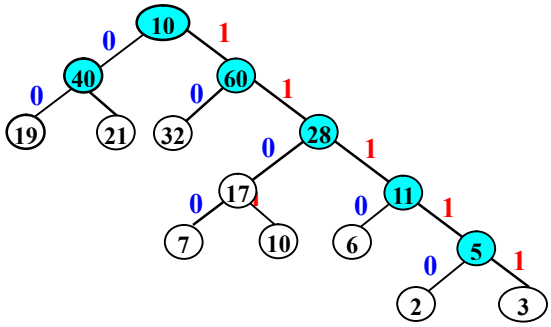
16. $O(\log_2 n)$
17. O(1)；随机存取
18. 384
19. n；$\log_k(n(k-1)+1)$
20. 邻接矩阵；邻接表；深度优先遍历；广度优先遍历
21. m；2m-1
22. v5； v1； v1,v2,v3,v4,v5;

III、 Answer the questions below (32%)

23. 从空的二叉树开始，根据字典顺序，严格按照二叉排序树（或称二叉搜索树）插入算法，依次插入 e，b，d，f，a，g，c。请画出构造二叉排序树的每一步骤。



24. 方案 1；哈夫曼编码



先将概率放大 100 倍，以方便构造哈夫曼树。w={7,19,2,6,32,3,21,10}，按哈夫曼规则：

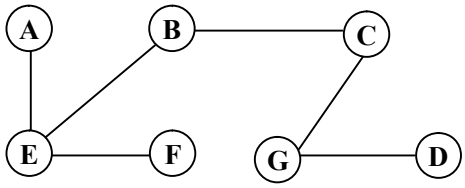
字母编号	对应编码	出现频率
1	1100	0.07
2	00	0.19
3	11110	0.02
4	1110	0.06
5	10	0.32
6	11111	0.03
7	01	0.21
8	1101	0.10

方案 2：定长编码

字母编号	对应编码	出现频率
1	000	0.07
2	001	0.19
3	010	0.02
4	011	0.06
5	100	0.32
6	101	0.03
7	110	0.21
8	111	0.10

方案 1 的 WPL=2(0.19+0.32+0.21)+4(0.07+0.06+0.10)+5(0.02+0.03)=1.44+0.92+0.25=2.61
方案 2 的 WPL＝3(0.19+0.32+0.21+0.07+0.06+0.10+0.02+0.03)=3
结论：哈夫曼编码优于等长二进制编码。

25. (1)最小生成树可直接画出，如右图所示。
(2)可用邻接矩阵和邻接表来描述：



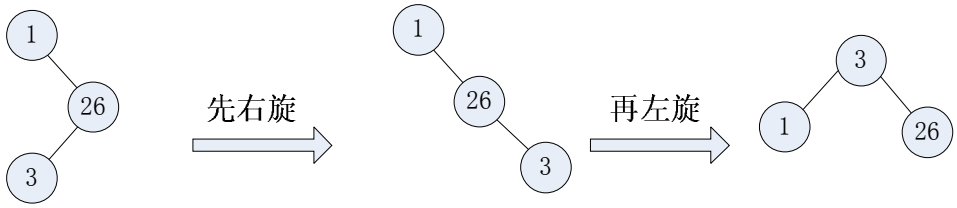
$$\begin{bmatrix} \infty & 12 & \infty & \infty & 4 & \infty & \infty \\ 12 & \infty & 20 & \infty & 8 & 9 & \infty \\ \infty & 20 & \infty & 15 & \infty & \infty & 12 \\ \infty & \infty & 15 & \infty & \infty & \infty & 10 \\ 4 & 8 & \infty & \infty & \infty & 6 & \infty \\ \infty & 9 & \infty & \infty & 6 & \infty & \infty \\ \infty & \infty & 12 & 10 & \infty & \infty & \infty \end{bmatrix}$$

邻接表为：

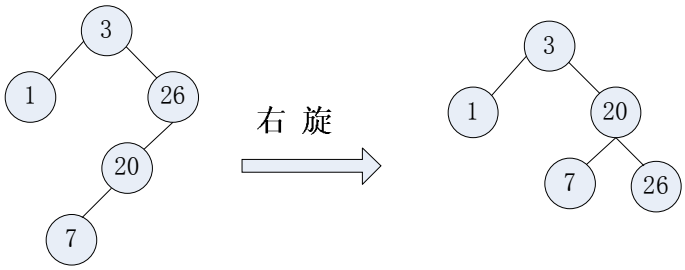
a		→	b	12		→	e	4	^							
b		→	a	12		→	c	20		→	e	8		→	f	9	^
c		→	b	20		→	d	15		→	g	12	^			
d		→	c	15		→	g	10	^							
e		→	a	4		→	b	8		→	f	6	^			
f		→	b	9		→	e	6	^							
g		→	c	12		→	d	10	^							

26. AVL 建树过程如下图所示：

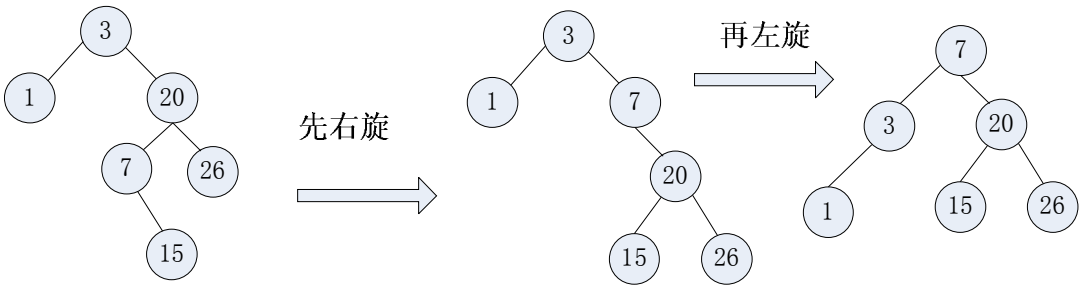
插入 1， 26， 3



插入 20， 7

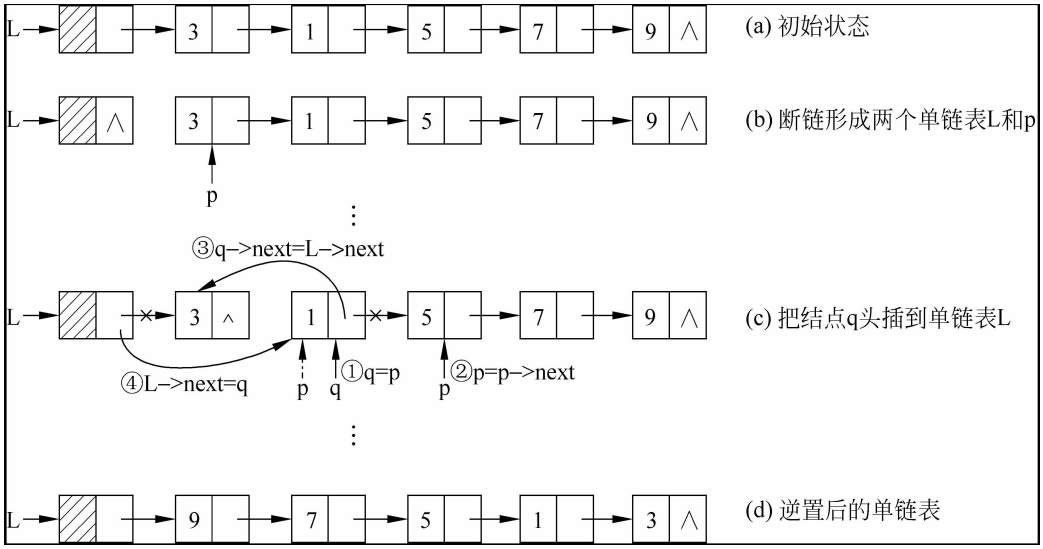


插入 15



IV、 Programming(38%)

27.



参考算法 1

```
void    reverse (Linklist *L)
{   Linklist *p,*q;    //p 为剩余结点形成单链表的头指针，q 保存欲插入 L 头的结点的地址
    p=L->next;                          /*p 指向第一个数据结点*/
    L->next=NULL;                       /*将原链表置为空表 L*/
    while (p)
    {
        q=p;
        p=p->next;
        q->next=L->next;                /*将当前结点插到头结点的后面*/
        L->next=q;
    }
}
```

28. Selection sort：

参考算法：

```
void selection_sort(int A[], int n)
/*
Post: The entries of the A have been rearranged so that
the keys in all the entries are sorted into nondecreasing order.
Uses: max_key, swap.
*/
{
    for(int position = n - 1; position > 0; position--)
    {

        int max = max_key(A, 0,position);
        swap(A, max,position);
    }
```

```
}

int max_key(int A[], int low, int high)
/*
Pre:  low and high are valid positions in the A and low <= high.
Post: The position of the entry between low and high with the largest
key is returned.
*/
{
    int largest, current;
    largest = low;
    for(current = low + 1; current <= high; current++)
    {
        if(A[largest] < A[current])
            largest = current;
    }
    return largest;
}

void swap(int A[], int low, int high)
/*
Pre:  low and high are valid positions in the A.
Post: The entry at position low is swapped with the entry at position high.
*/
{
    int temp;
    temp = A[low];
    A[low] = A[high];
    A[high] = temp;
}
```

29. **Non-recursive pre-order：**

STL 版本

```
void preorder(void(*visit)(Entry &))
{
    stack<Binary_node<entry>*> s;
    Bonary_node<Entry>* p = root;
    while(p || !s.empty())
    {
        if(p)       //根指针进栈，遍历左子树
        {
            s.push(p);
            p = p->left;
        }
```

```
        else //根指针退栈，访问根结点，遍历右子树
        {
            p = s.top();
            s.pop();
            visit(p->entry);
            p = p->right;
        }
    }
}
```

数组版本

```
void preorder(void(*visit)(Entry &))
{
    Binary_node<entry>* s[MAXLEN];
    int top = 0;
    Bonary_node<Entry>* p = root;

    while(p || !s.empty())
    {
        if(p)//根指针进栈，遍历左子树
        {
            s[top++] = (p);
            p = p->left;
        }
        else  //根指针退栈，访问根结点，遍历右子树
        {
            p = s[top--];
            visit(p->entry);
            p = p->right;
        }
    }
}
```