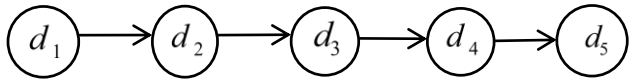
期末试题部分

西北工业大学 2004-2005 学年第一学期期末考试(软微 A 卷)

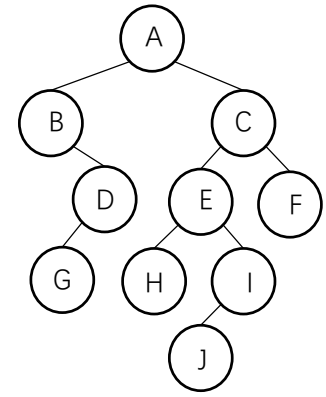
1. (10 分) 

有向图不是线性结构。

2. (10 分)相同点：栈和队列都是操作受限制的线性表，数据元素之间都存在“一对一”的关系。

不同之处：栈是只允许在一端进行插入或删除操作的线性表，其特点是“后进先出”；队列是只允许在一端进行插入，另一端进行删除操作的线性表，其特点是“先进先出”。

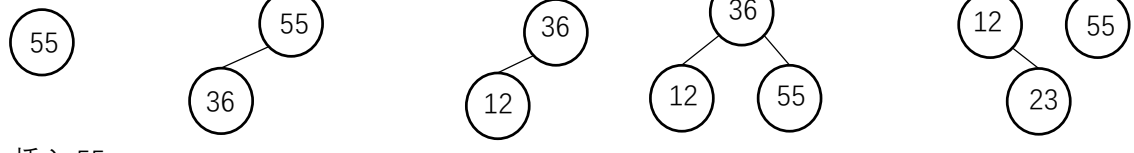
3.(10 分)层序遍历的第一个结点为 A，A 是第一层的结点，在中序序列中找到 A，则 BGD 和 HEJICF 分别在 A 的左右两边。层序遍历的第二个结点是 B，且中序序列中 B 在 A 的左边，则 B 是 A 的左孩子；层序遍历的第三个结点是 C，且中序序列中 C 在 A 的右边，则 C 是 A 的右孩子；再看中序序列 BGD，已知 B 的位置，GD 在 B 的右边；HEJICF 中，HEJI 在 C 的左边，F 在 C 的右边且唯一，因此 F 是 C 的右孩子；层序遍历的第四个结点时 D，且 D 在 B 的右边，因此 D 是 B 的右孩子；在中序序列中，G 在 D 的左边且唯一，因此 G 是 D 的左孩子；层序序列的第五个结点是 E，且 E 在 C 的左边，因此 E 是 C 的左孩子；中序序列中，H 在 E 的左边且唯一，因此 H 是 E 的左孩子；层序序列的第九个结点是 I，且中序序列中 I 在 E 的右边，则 I 是 E 的右孩子；中序序列中 J 在 I 的左边，因此 J 是 I 的左孩子。



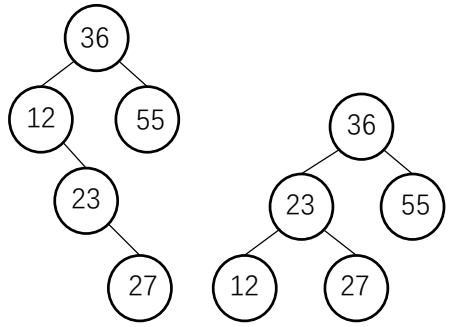
4.

顶点	第 1 趟	第 2 趟	第 3 趟	第 4 趟	第 5 趟
V1	15 V0→V1	15 V0→V1	15 V0→V1	15 V0→V1	15 V0→V1
V2	2 V0→V2				
V3	∞	10 V0→V2→V3	7 V0→V2→V4→V3		
V4	∞	6 V0→V2→V4			
V5	∞	∞	16 V0→V2→V4→V5	14 V0→V2→V4→V3→V5	
集合	{V0,V2}	{V0,V2,V4}	{V0,V2,V4,V3}	{V0,V2,V4,V3,V5}	{V0,V2,V4,V3, V5, V1}

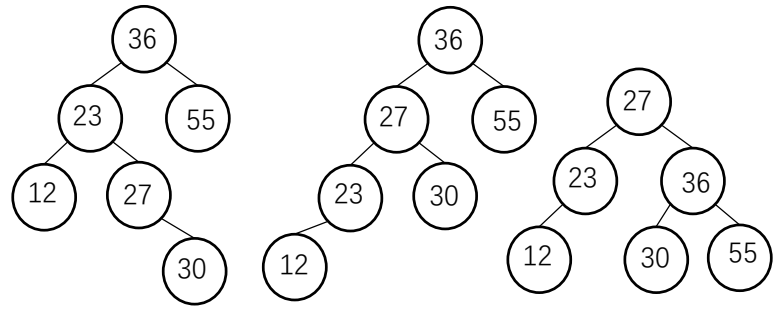
5. (15 分)



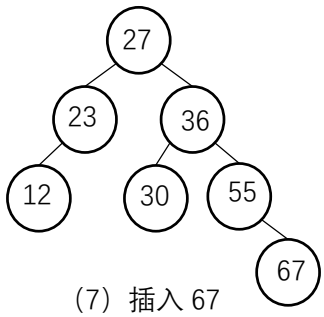
(1) 插入 55　(2) 插入 36　(3) 插入 12，失衡（LL），单向右旋　(4) 插入 23



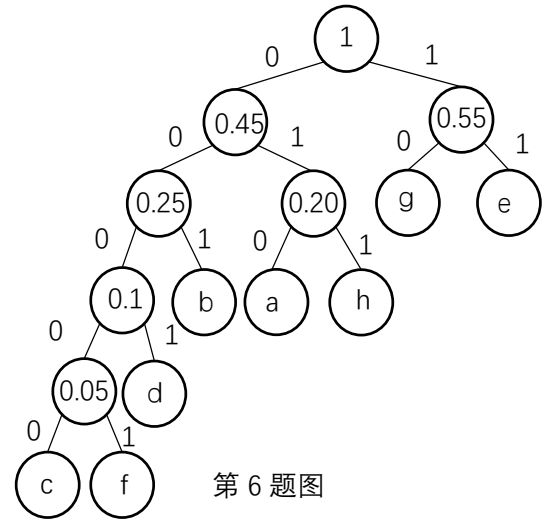
（5）插入 27，失衡（RR），单向左旋



（6）插入 30，失衡（LR），先左旋后右旋



（7）插入 67



第 6 题图

6. (15 分) (1)a:010 b:001 c:00000 d:0001 e:11 f:00001 g:10 h:011

(2)8 个字母，采用等长编码，编码为 3 位。采用哈夫曼编码平均码长：

3*(0.1+0.14+0.1)+5*(0.02+0.03)+4*0.06+2*(0.30+0.25)=2.61；

哈夫曼编码的平均码长是等长编码的 2.61/3*100%=87%；它使电文总长平均压缩(3-2.61)/3*100%=13%。

7. (20 分) (1)邻接表

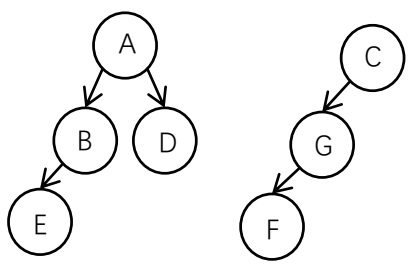
```
typedef struct ArcNode{
    int adjvex;
    struct ArcNode* next;
}ArcNode;
```

```
typedef struct VNode{
    VertexType data;
    ArcNode *first;
}VNode, AdjList[MaxVertexNum];
```

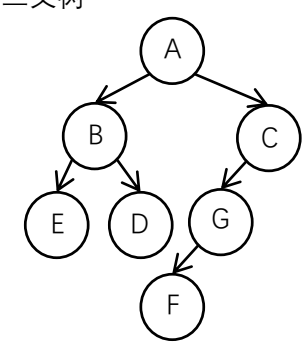
```
typedef struct{
    AdjList vertices;
    int vexnum, arcnum;
}ALGraph;
```

(2)深度优先遍历序列： ABEDCFG

深度优先生成森林：



二叉树



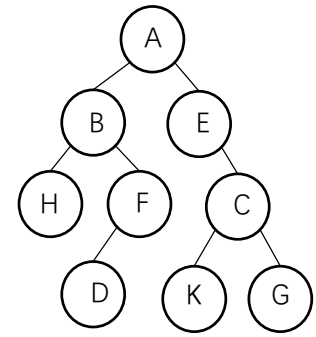
(3)拓扑序列：ACBDFEG

8. (10 分)

```
void swap(linklist &list,linklist &p){
    linklist q=list,r=p->next;
    while(q->next!=p)
        q=q->next;
    q->next=r;
    p->next=r->next;
    r->next=p;
}
```

西北工业大学 2004-2005 学年第一学期期末考试(软微 B 卷)

1.略　**2.**略　**3.**(见右图)不能。前序和后序在本质上都是将父节点与子结点进行分离，但并没有指明左子树和右子树的能力，因此得到这两个序列只能明确父子关系，而不能确定一个二叉树。



4.

顶点	第 1 趟	第 2 趟	第 3 趟	第 4 趟
B	10 A→B			
C	18 A→C	18 A→C	17 A→B→D→C	
D	∞	15 A→B→D		
E	∞	∞	17 A→B→D→E	17 A→B→D→E
集合	{A,B}	{A,B,D}	{A,B,D,C}	{A,B,D,C,E}

5.(1)

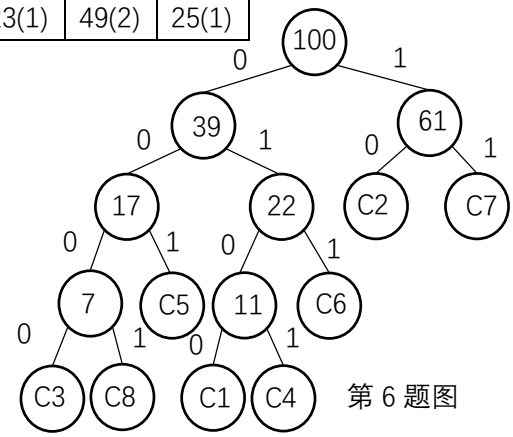
0	1	2	3	4	5	6	7	8	9	10	11	12
78(1)		28(1)	3(1)		44(1)	32(1)	20(1)	31(4)		23(1)	49(2)	25(1)

(2)ASL（成功）=（8*1+1*2+1*4）/10=1.4

6.(见右图)C1:0101　C2:10　C3:0000　C4:0101　C5:001
C6:011　C7:11　C8:0001
平均码长=(4*(5+3+6+4)+3*(10+11)+2*(25+36))/100=2.57

7.略

8.略



第 6 题图

西北工业大学 2007-2008 学年期末考试(软微 A 卷)

一. (15 分)判断题

答案速查：×√×××　　××√√√　　×√×××

(1)×【解析】前序和后序在本质上都是将父节点与子结点进行分离，但并没有指明左子树和右子树的能力，因此得到这两个序列只能明确父子关系，而不能确定一个二叉树。也可举反例。

(3)×【解析】有 2e 个结点。

(4)×【解析】关键路径是事件结点网络中从源点到汇点的最长路径。

(5)×【解析】表尾是(b,(c,(d)))。

(6)×【解析】第 V 行中的 1 的个数仅是顶点 V 的出度，顶点 V 的度等于第 V 行和第 V 列中的 1 的个数之和。

(7)×【解析】若删除某非叶子结点后，再将其插入，它会成为叶子结点。

(10)√【解析】可用循环或栈实现。

(11)×【解析】当删除时，需在该记录的位置上填入一个特殊的符号，以免找不到在它之后填入的“同义词”的记录。

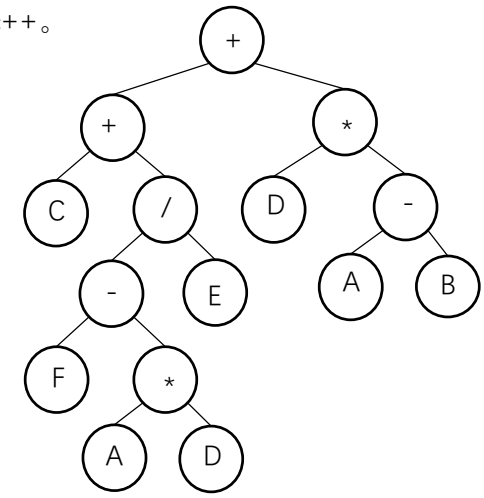
(13)×【解析】快速排序平均性能可达 $O(n\log_2 n)$，在实际应用中常常优于其他排序算法。从空间复杂度上看，快速排序需要使用辅助栈用于递归，平均大小为 $O(n\log_2 n)$，所需附加空间并非最少。

(15)×【解析】排序算法稳定性的判定依据是序列中相等元素的相对位置是否发生改变。

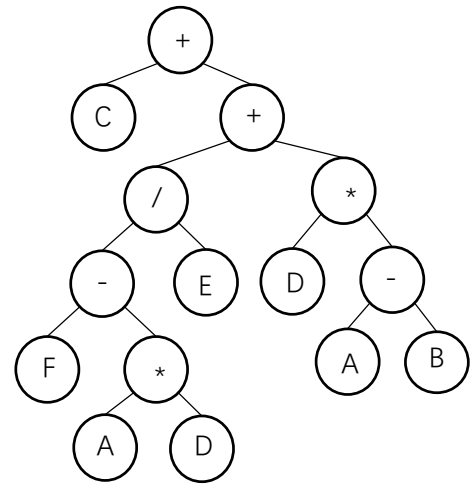
二. (10 分)

	前缀形式	后缀形式
图 2.1	++C/-F*ADE*D-AB	CFAD*-E/+DAB-*+
图 2.2	+C+/-F*ADE*D-AB	CFAD*-E/DAB-*++

【注】中缀表达式转化成后缀或者前缀，结果并不一定唯一。后缀式和前缀式都只有唯一的一种运算次序，而中缀却不一定。后缀式和前缀式是由中缀式按某一种运算次序而生成的，因此对于一个中缀式可能有多种后缀式或者前缀式。例如 a+b+c 可以先算 a+b 也可先算 b+c，这样就有两种后缀式与其对应，分别是 ab+c+和 abc++。

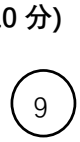


第 2.1 题图

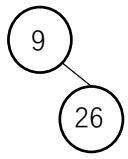


第 2.2 题图

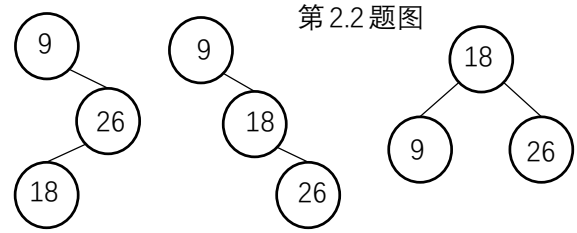
三. (10 分)



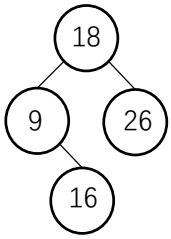
(1)插入 9



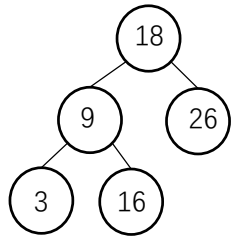
(2)插入 26



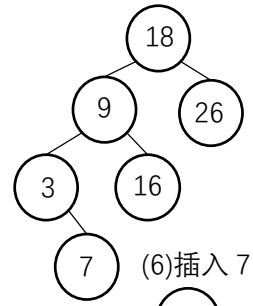
(3)插入 18，失去平衡，做先右后左双向旋转



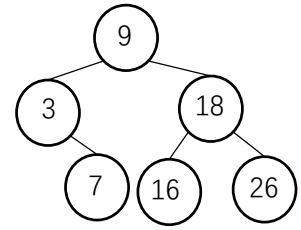
(4)插入 16

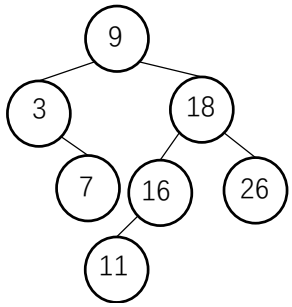


(5)插入 3

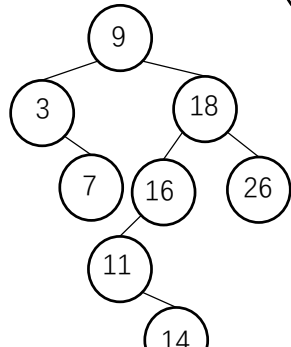


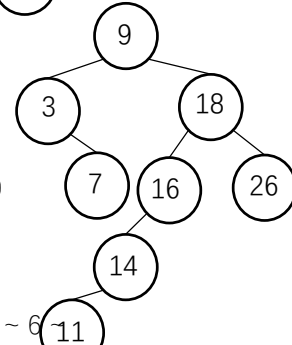
(6)插入 7，失去平衡，做单向右旋

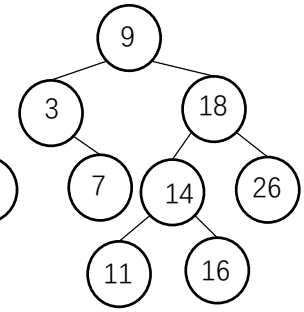


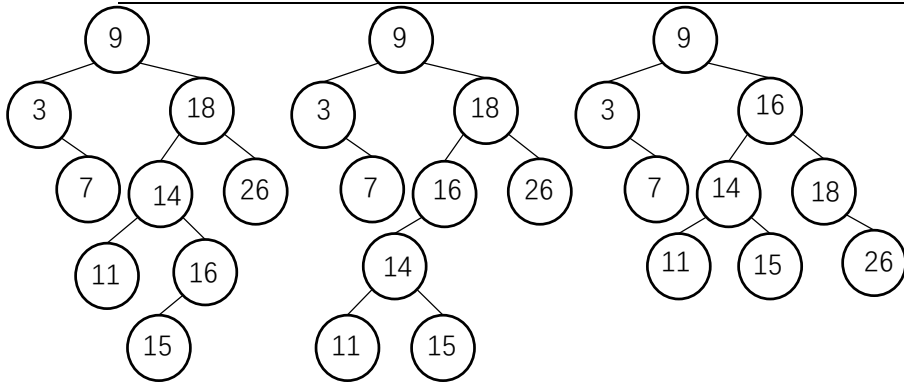


(7)插入 11

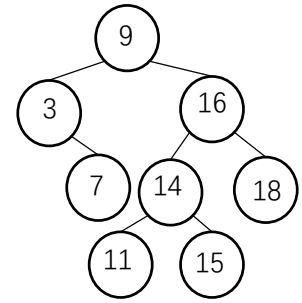








(9)插入 15，失去平衡，做先左后右双向旋转



(10)删除 26

四.

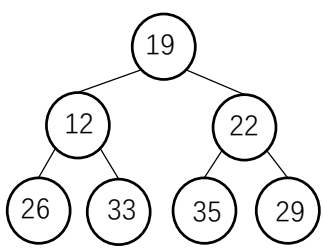
顶点	第 1 趟	第 2 趟	第 3 趟	第 4 趟
B	10 A→B			
C	20 A→C	20 A→C	17 A→B→D→C	
D	∞	15 A→B→D		
E	∞	∞	17 A→B→D→E	17 A→B→D→E
集合 S	{A,B}	{A,B,D}	{A,B,D,C}	{A,B,D,C,E}

五. (10 分) (1)基本思想是：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

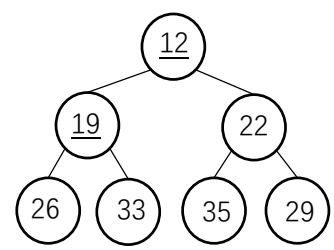
(2) $O(n\log_2 n)$　不稳定 $O(n^2)$　(3) 空间复杂度平均情况：$O(n\log_2 n)$，最坏情况：$O(n)$

六. (10 分) (题目未说明，这里以升序排序为例)

（1）构造初始小顶堆

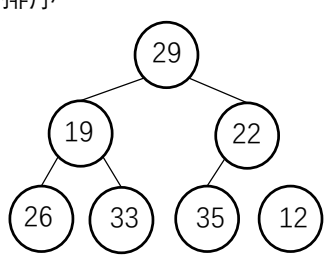


(1)初始状态

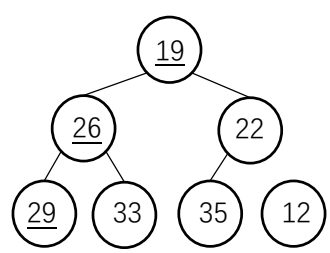


(2)19 和 12 交换

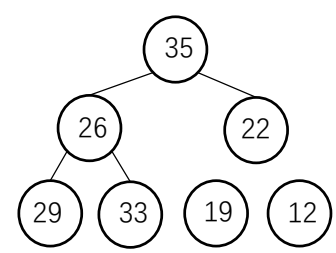
(2)堆排序



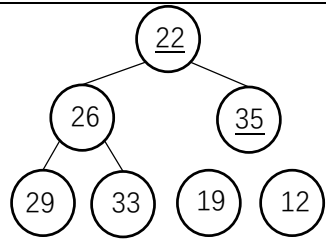
(1)输出 12, 12 和 29 交换



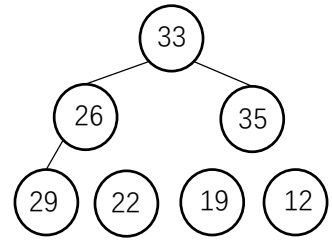
(2)向下调整，29 和 19 交换，29 和 26 交换



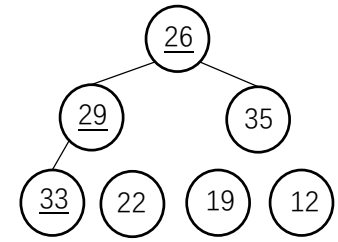
(3)输出 19, 19 和 35 交换



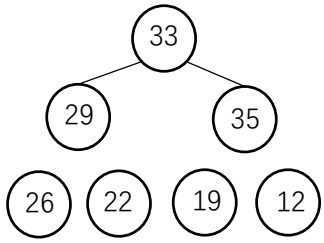
(4)向下调整, 35 和 22 交换



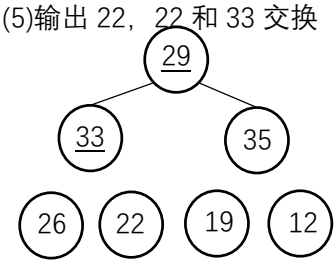
(5)输出 22，22 和 33 交换



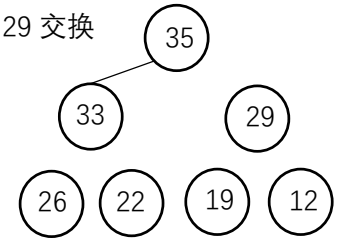
(6)向下调整，33 和 26 交换，33 和 29 交换



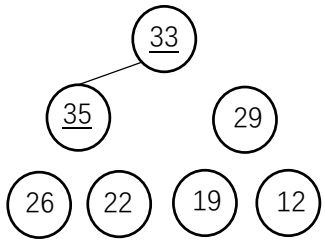
(7)输出 26，26 和 33 交换



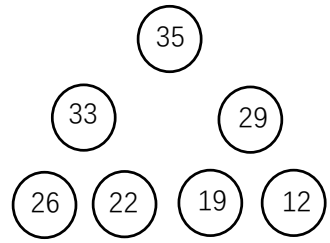
(8)向下调整, 33 和 29 交换



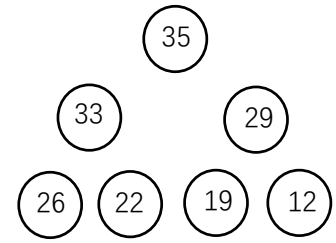
(9)输出 29，29 和 35 交换



(10)向下调整, 35 和 33 交换



(11)输出 33，33 和 35 交换



(12)输出 35

七.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	2	3	4	5	6	7	8	9	10	11	12	13	14
O	P	Q	R	S	T	U	V	W	X	Y	Z		
15	16	17	18	19	20	21	22	23	24	25	26		

用线性探测法处理冲突:

H(Jan)=10/2=5;

H(Feb)=6/2=3;

H(Mar)=13/2=6;

H(Apr)=1/2=0;

H(May)=13/2=6;冲突;H1=6+1=7;

H(June)=10/2=5;冲突;H1=5+1=6;冲突;H2=7;H3=8;

H(July)=5;H1=6;H2=7;H3=8;H4=9

H(Aug)=0;H1=1;

H(Sep)=9;H1=10;

H(Oct)=7;H1=8;H2=9;H3=10;H4=11;

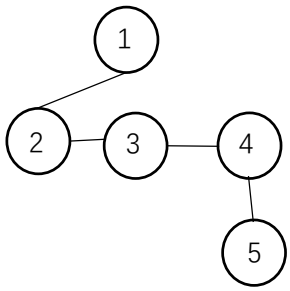
H(Nov)=7;H1=8;H2=9;H3=10;H4=11;H5=12

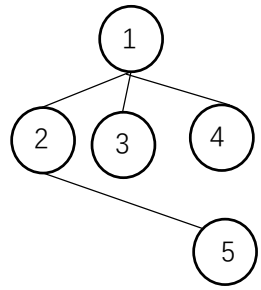
H(Dec)=2

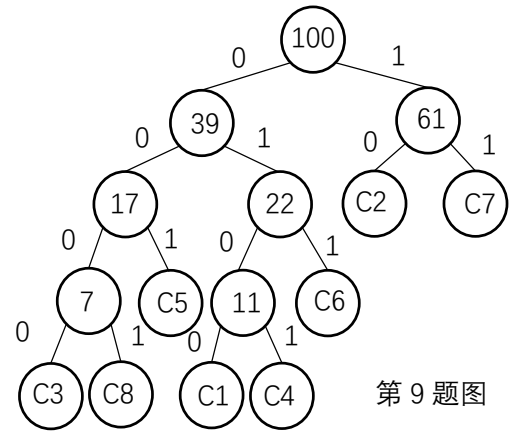
0	1	2	3	4	5	6	7	8
Apr	Aug	Dec	Feb		Jan	Mar	May	June
9	10	11	12	13	14	15	16	
July	Sep	Oct	Nov					

$ASL_{成功}$=(5*1+3*2+1*4+2*5+1*6)/12=31/12

八.(10 分) (1)深度遍历序列：1 2 3 4 5 **(2)**广度遍历序列：1 2 3 4 5





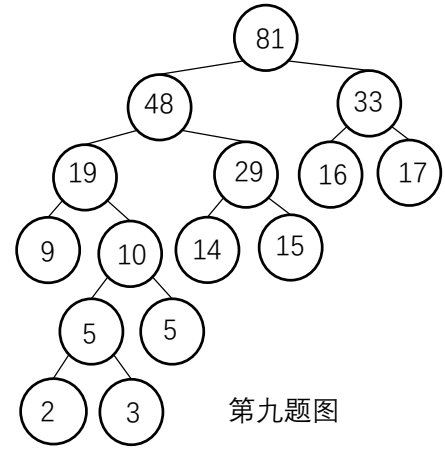


第 9 题图

九.(15 分) (见右图)C3:0000 C8:0001 C1:0100 C4:0101 C5:001 C6:011 C2:10 C7:11

总码数=(3+4+5+6)×4+(10+11)×3+(25+36)×2=257

西北工业大学 2007-2008 学年期末考试(软微 B 卷)



第九题图

一. (15 分)判断题

答案速查：√ × √ √ √ ×× √ × √ × √ ××××

(7) ×【解析】可以有左孩子而无右孩子，但不是叶子结点。

九. (10 分)见右图

带权路径长度=(2+3)×5+5×4+(9+14+15)×3+(16+17)×2=225

西北工业大学 2011-2012 学年第二学期期末考试(软微 A 卷)

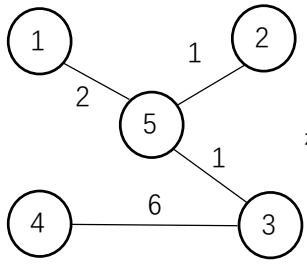
一.填空题(每空 1 分，20 分)

1.O(1) O(n) **2.**升序序列 后缀表达式(逆波兰式) **3.** A[2i+1] A[2i+2] A[(i-1)/2]
4.出度 入度 **5.** x ((x,y),((x,y),z)) **6.**前一个位置 n-1 **7.**时间复杂度 空间复杂度
8. 38 13 27 10 65 76 97 **9.** $O(n^2)$ $O(n\log_2 n)$ **10.**栈顶 栈顶

二.简答题(每题 5 分，共 35 分)

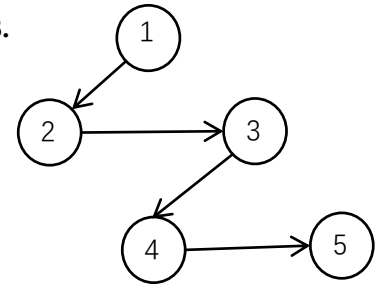
1. 第一趟：{24 40 38} 46 {56 80 95 79}
第二趟：24 {40 38} 46 56 {80 95 79}
第三趟：24 {38} 40 46 56 {79} 80 {95}

2.



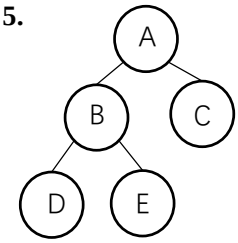
权值之和=10

3.



4.无拓扑序列，因为图中存在回路。

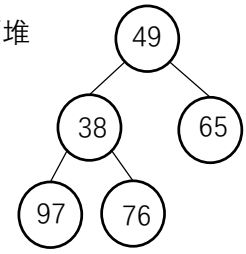
5.



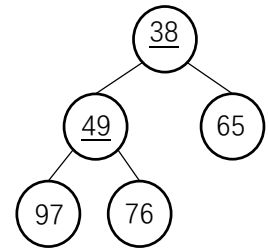
后序序列：DEBCA

6.(1)构造初始小顶堆

(2) O(nlog$_2$n)

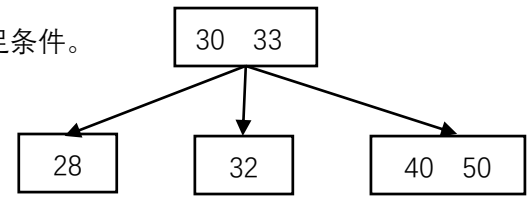


①初始状态



②49 和 38 交换

7.如果 m=3，答案如右图所示；当 m=2 时，不满足条件。



三.设计题(每题 15 分，共 45 分)

1.首先用子问题定义状态：即 f[i][j]表示前 i 件物品恰放入一个容量为 j 的背包可以获得的最大价值。则其状态转移方程便是：

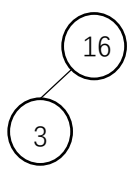
f[i][j]=max{f[i-1][j],f[i-1][j-w[i]]+p[i]}

若只考虑第 i 件物品的策略(放或不放)，那么就可以转化为一个只牵扯前 i-1 件物品的问题。如果不放第 i 件物品，那么问题就转化为“前 i-1 件物品放入容量为 j 的背包中”，价值为 f[i-1][j]；如果放第 i 件物品，那么问题就转化为“前 i-1 件物品放入剩下的容量为 j-w[i]的背包中”，此时能获得的最大价值就是 f[i-1][j-w[i]]再加上通过放入第 i 件物品获得的价值 p[i]。代码：

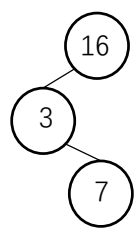
```
for (i = 1; i <= n; i++)
    for (j = c; j >= w[i]; j--)
        f[i][j] = max(f[i - 1][j], f[i - 1][j - w[i]] + p[i]);
```

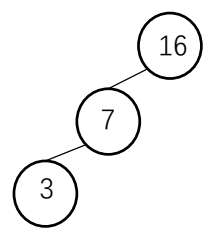
2.

(1)插入 16

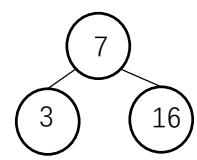


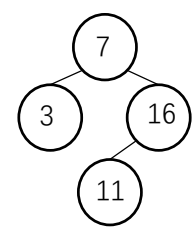
(2)插入 3



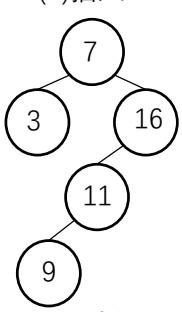


(3)插入 7，失去平衡，做先左后右双向旋转

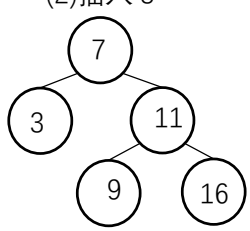


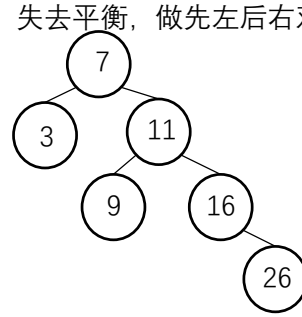


(4)插入 11

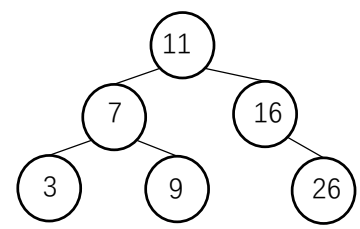


(5)插入 9，失去平衡，做单向右旋

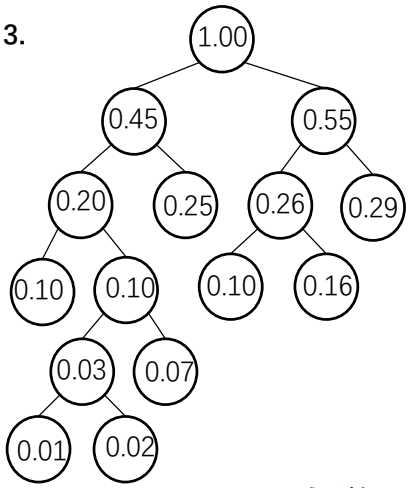




(6)插入 26，失去平衡，做单向左旋



3.



平 均 码 长 =(0.01+0.02) × 5+0.07 × 4+(0.10+0.10+0.16)×3+(0.25+0.29)×2=2.59

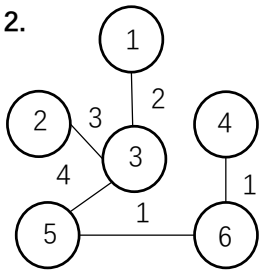
西北工业大学 2011-2012 学年第二学期期末考试(软微 B 卷)

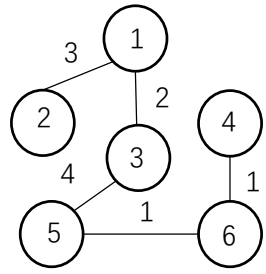
一.填空题(每空 1 分，20 分)

1.队尾 队头 **2.** 3 3 **3.**A[2i+1] A[2i+2] A[(i-1)/2] **4.**开放定址法 链地址法
5.-1 34X*+2Y*3/- **6.**e 2e **7.**n(n-1)/2 n(n-1) **8.**归并 **9.**5 31 **10.**n-i+1 n-i

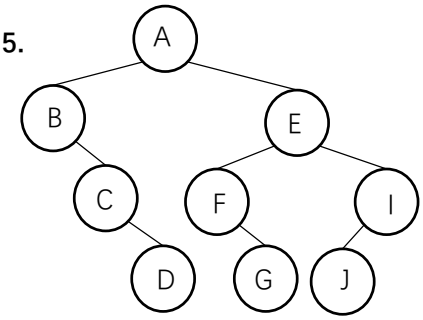
二.简答题(每题 5 分，共 35 分)

2.





5.



中序序列：BCDAFGEJI

三.设计题

1.问题分析：虽然是两艘船的问题，其实只讨论一艘船的最大装载问题即可。因为当第一艘船的最大装载为 best 时，若 w1+w2+…+wn-best<=c2，则可以确定一种解，否则问题就无解。这样问题可以转化为第一艘船的最大装载问题。

思路：(1) 首先将第一艘轮船尽可能装满；(2) 将剩余的集装箱装上第二艘轮船。

```
const int max=12;
int c1,c2,n,boxw[max],weight,best;
void backtrack(int a) {
    if(a==n) {
        if(weight>best)    best=weight;
        return ;
    }
    if(weight+boxw[a]<=c1) {
        weight=weight+boxw[a];
```

```
int main() {
    int sum=0;
    cin>>c1>>c2>>n;
    for(int i=0;i<n;i++) {
        cin>>boxw[i];
        sum+=boxw[i];
    }
    best=weight=0;
    backtrack(0);
```

```
            backtrack(a+1);
            weight=weight-boxw[a];
        }
        backtrack(a+1);
    }
```

```
    if(sum-best<=c2)
        cout<<"Yes"<<endl;
    else   cout<<"No"<<endl;
    return 0;
}
```

西北工业大学 2014-2015 学年第一学期期末考试(软微)

一.单项选择题(每题 2 分，共 20 分)

答案速查：BCDCB　DDCAB

1.B【解析】用链表的形式表示的线性表最大的优势是能动态地、很方便地进行插入和删除操作。

2.C【解析】只有当线性表中数据元素按值大小有序排列，并且采用顺序存储结构时才能使用折半查找方法查找元素。即使线性表中数据元素按值大小有序排列，但采用的不是顺序存储结构(如链式)，仍然不能够采用折半查找方法。本题应选 C。

3.D【解析】一个非空广义表的表头可以是子表或原子。

4.C【解析】先在长度为 m 的单链表中找到尾结点，然后将其 next 域置为另一个单链表的首结点，其时间复杂度为 O(m)。

7.D【解析】归并排序空间复杂度 n,快速排序复杂度 log2n，归并排序空间复杂度最大，此题选 D。

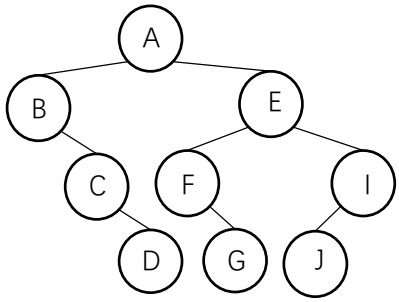
8.C【解析】哈夫曼树的节点的度是 0 或 m,设度不为 0(即非叶结点)的个数为 X ,则总的结点数为：X+n ,除根结点外，其余的每一个结点都有一个分支连向一个结点，对于度为 m 的每个结点都有 m 个分支，而度为 0 的结点是没有分支的，所以从分支的情况来看，总的结点数位：X*m + 1(这里的 1 为根结点) 。两者相等，所以答案是 (n-1) / (m-1)

二.简答题(每题 5 分，共 20 分)

1. AVL 树定义：AVL 树是一棵空树，或者是具有下列性质的二叉树：它的左子树和右子树都是 AVL 树，且左子树和右子树的高度之差的绝对值不超过 1。

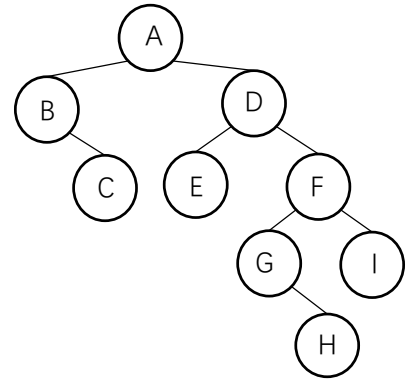
2.关键路径：在 AOE 网的源点到汇点的所有路径中，具有最大路径长度的路径称为关键路径。应用：估算工程的完成时间。

3.



后序遍历序列为：CBEHGIFDA

4.

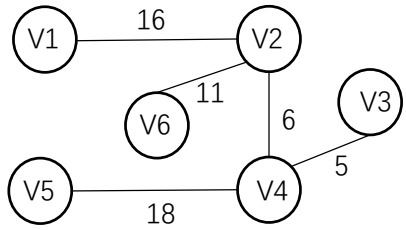


三.应用题(每题 8 分，共 40 分)

1.初始序列：25 84 21 47 15 27 68 35 24
24 84 21 47 15 27 68 35 25 (25 和 24 交换)
24 25 21 47 15 27 68 35 84 (25 和 84 交换)
24 15 21 47 25 27 68 35 84 (25 和 15 交换)

{24 15 21} 25 {47 27 68 35 84} (25 和 47 交换)
第一趟：{24 15 21} 25 {47 27 68 35 84}
第二趟：{21 15} 24 25 {35 27} 47 {68 84}
第三趟：{15} 21 24 25 {27} 35 47 68 {84}

2.



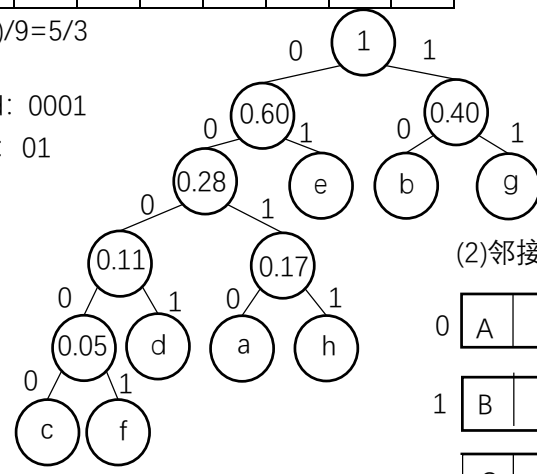
3. H(47)=47%11=3
H(7)=7%11=7
H(29)=29%11=7 H1=(7+1)%11=8
H(11)=11%11=0

H(16)=16%11=5
H(92)=92%11=4
H(22)=22%11=0 H1=(0+1)%11=1
H(8)=8%11=8 H1=(8+1)%11=9
H(3)=3%11=3 H1=4 H2=5 H3=6

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

ASL=(5×1+3×2+4×1)/9=5/3

4.c：00000 f：00001 d：0001
a：0010 h：0011 e：01
b：10 g：11

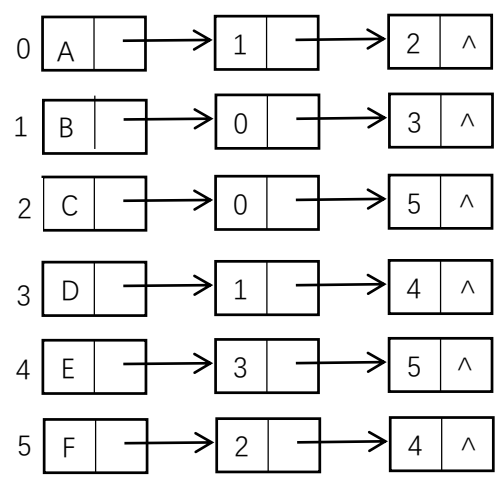


5.(1)邻接矩阵

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

深度优先遍历序列：ABDEFC
广度优先遍历序列：ABCDFE

(2)邻接表



四.设计题(每题 10 分，共 20 分)

1.假设两个单链表 HA 和 HB 中的元素为升序排列，从第一个元素开始依次比较，选取键值较小的一个插入新链表的尾部。若遍历完某一单链表，则将另一单链表中剩余元素全部插入新链表尾部。

```
void MergeList(LNode* &HA, LNode* &HB){
    LNode *r = HA, *pa=HA->next, *pb=HB->next;
}
```

```
    while(pa != NULL && pb != NULL){
        if(pa->data <= pb->data){
            r->next = pa;
            r = pa;
            pa = pa->next;
        }
        else{
            r->next = pb;
            r = pb;
            pb = pb->next;
    if(pa != NULL)
        pb = pa;
    while(pb != NULL) {
        r->next = pb;
        r = pb;
        pb = pb->next;
    }
    r->next = NULL;
    free(HB);
}
```

2.对二叉排序树来说，其中序遍历序列为一个递增有序序列。因此，对给定的二叉树进行中序遍历，如果始终能保持前一个值比后一个值小，则说明该二叉树是一棵二叉排序树。

```
int predt = -99999999;
int JudgeBST(BiTree bt){
    int b1, b2;
    if(bt == NULL)
        return 1;
    else {
        b1 = JudgeBST(bt->lchild);
        if(b1 == 0 || predt >= bt->data)
            return 0;
        predt = bt->data;
        b2 = JudgeBST(bt->rchild);
        return b2;
    }
}
```

西北工业大学 2016-2017 学年第二学期期末考试(软件双语)

一、(15 分）可以利用栈的先进后出特点进行判断，给出一个示例。

```
#include <iostream>
using namespace std;
#define Max_String_Len 100
#include "SqStack.h"
//判断字符串是否回文
bool ispalindrome(char *in_string) {
    SqStack <char> s(Max_String_Len);
    char deblankstring[Max_String_Len], c;
    int i = 0;
    //过滤空格字符
    while(*in_string != '\0'){
        if(*in_string != ' ')
        deblankstring[i++] = *in_string;
        in_string++;
    }
    deblankstring[i] = '\0';
    //有效字符依次入栈
    i = 0;
    while(deblankstring[i] != '\0')
        s.Push(deblankstring[i++]);
    while(!s.Empty()){
        c = s.Top();
        s.Pop();
        if(c != deblankstring[i])
            return false;
            i++;
    }
    return true;
}
int main(    ){
    char instring[Max_String_Len];
    cout << "input a string:" << endl;
    cin.get(instring, Max_String_Len);
    //cout<<instring;
    if(ispalindrome(instring))
        cout << "\"" << instring << "\"" << " is a palindrome." << endl;
    else
        cout << "\"" << instring << "\"" << " is not a palindrome." << endl;
```

```
    //从栈中弹出字符依次比较
        i = 0;
    system("pause");
    return 0;
}
```

二、(1) 类型名称：二叉树

数据对象集：一个有穷的结点集合。若不为空，则由根结点和其左、右二叉子树组成。

操作符：BT∈BinTree,Item∈ElementType。

二叉树重要操作

1 Boolean IsEmpty(BinTree BT); //判断 BT 是否为空

2 void Traversal(BinTree BT); //遍历，按某顺序访问每个结点

3 BinTree CreatBinTree(); //创建一个二叉树

二叉树常用遍历方法

1 void PreOrderTraversal(BinTree BT); //先序 根、左子树、右子树

2 void InOrderTraversal(BinTree BT); //中序 左子树、根、右子树

3 void PostOrderTraversal(BinTree BT); //后序 左子树、右子树、根

4 void LevelOrderTraversal(BinTree BT); //层次遍历，从上到下、从左到右

（2）满二叉树：除了叶结点外每一个结点都有左右子叶且叶结点都处在最底层的二叉树。

完全二叉树：只有最下面的两层结点度小于 2，并且最下面一层的结点都集中在该层最左边的若干位置的二叉树。

也就是说，在满叉树的基础上，在最底层从右往左删去若干节点，得到的都是完全二叉树。

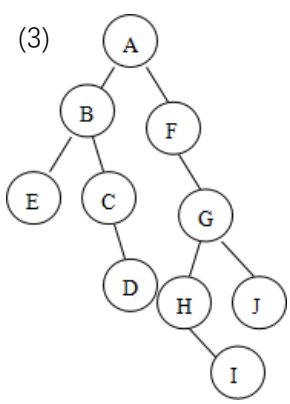
所以说，满二叉树一定是完全二叉树，但是完全二叉树不一定是满二叉树

平衡二叉树：树的左右子树的高度差不超过 1 的数，空树也是平衡二叉树的一种。

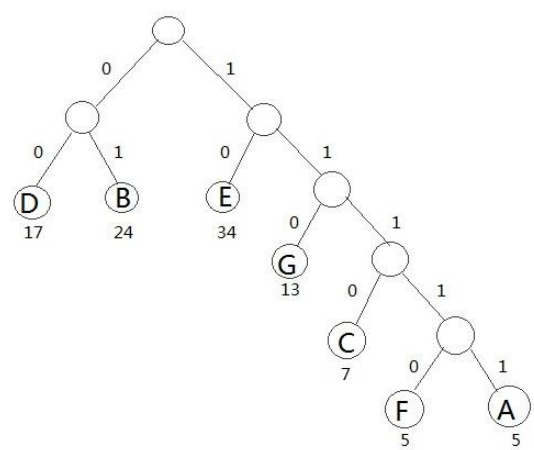
平衡二叉树，又称 AVL 树。它或者是一棵空树，或者是具有下列性质的二叉树：它的左子树和右子树都是平衡二叉树，且左子树和右子树的高度之差之差的绝对值不超过 1.。

常用算法有：红黑树、AVL 树、Treap 等

(3)



三、



A: 1111

B: 01

C: 1110

D: 00

E: 0

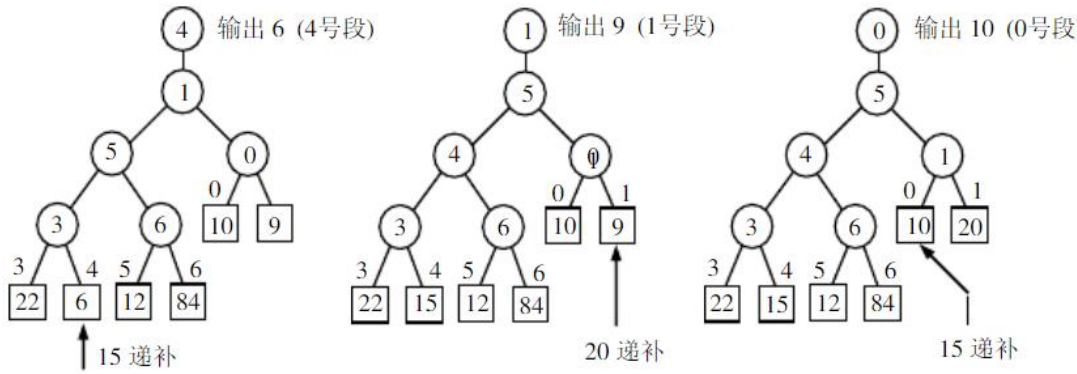
F: 11110

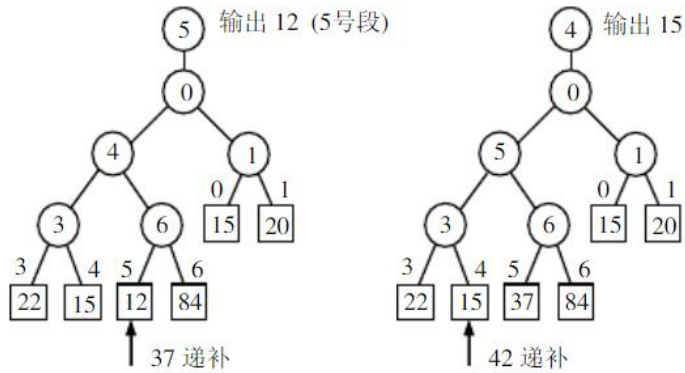
G: 110

四、

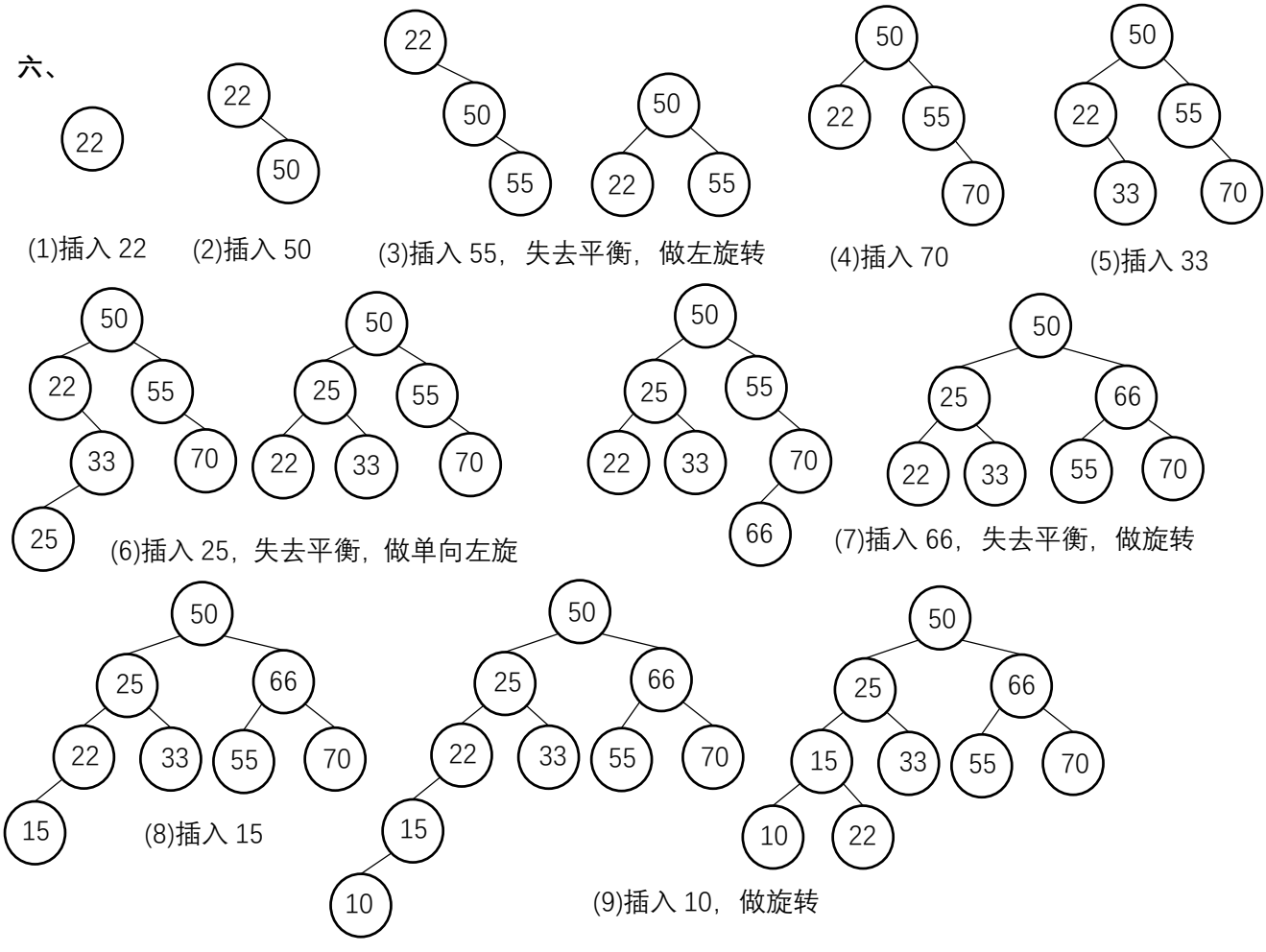
	路径	距离
V1->V2	V1.V7.V2	22
V1->V3	V1.V7.V4.V6.V3	25
V1->V4	V1.V7.V4	13
V1->V5	V1.V5	11
V1->V6	V1.V7.V4	16
V1->V7	V1.V7	7

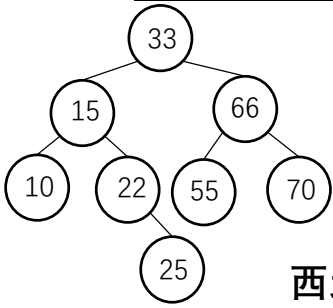
五、【解答】做 6 路归并排序，选择最小的 5 个关键码的败者树如下图所示。


输出 6 (4号段)
15 递补
输出 9 (1号段)
20 递补
输出 10 (0号段)
15 递补


输出 12 (5号段)
37 递补
输出 15 (4号段)
42 递补

六、


(1)插入 22
(2)插入 50
(3)插入 55，失去平衡，做左旋转
(4)插入 70
(5)插入 33
(6)插入 25，失去平衡，做单向左旋
(7)插入 66，失去平衡，做旋转
(8)插入 15
(9)插入 10，做旋转



历年真题部分

西北工业大学 2004 年研究生入学考试(814)

(10)删除 50

一.单项选择题(每空 2 分，共 20 分)

答案速查：DA BBEC BCCA

1.D A　**2.**B【解析】因为 KMP 算法涉及到 next 数组的存储，且 next 数组是基于模式串长度计算的。

3.B【解析】非循环链表和带头指针的循环链表查找尾节点的时间效率是 O(n)，而带尾指针的循环链表查找首尾节点的时间效率都是 O(1)，查找和更改时只需修改指针，无需遍历，队列的操作实际上就是对链表首尾节点的操作。

5.C【解析】链地址法不会产生“聚集”现象

7.C【解析】A[8][5]在矩阵 A 的第 9 行第 6 列，1+2+3+…+8+6-1=41。

8.C【解析】每一个非终端节点，它的所有孩子节点在转化之后，最后一个孩子的右指针也为空，所以为 n。森林中的最后一棵树成为右子树，其根结点也没有右孩子，所以为 n+1。

二.判断题(每小题 1 分，共 7 分)

答案速查：×××√×　××

1.×【解析】数据元素，此题常见考查形式为选择题，如下例题。

【例题】数据的基本单位是（　）

A.数据项　B.数据类型　C.数据元素　D.数据变量

2.×【解析】因为可能出现重复的权值，所以树不唯一；如果所有权值都不唯一，那么最小生成树唯一。

【举一反三】一个带权的无向连通图的最小生成树的权值之和是唯一的（√）

【解析】树不唯一，权值和唯一。

3.×【解析】数组的数据元素之间的逻辑结构是一对一的线性结构。

【举一反三】多维数组元素之间的关系既不是线性的也不是树形的。（√）

4.×【解析】只考虑了 2-路归并情况而未考虑多路归并情况，O(nlogN)更为准确。

【注】算法复杂度一般用 logN 而非 $\log_2 n$ 是因为假如有 logab（a 为底数）由换底公式可得 $\log_a b=\log_c b/\log_c a$，$\log_c a$（c 为底数）为常数，由运算法则“O(C×f(N))=O(f(N))，其中 C 是一个正的常数”得 $O(\log_a b)=O(\log_c b)$可知算法的时间复杂度与不同底数只有常数的关系，均可以省略。故可用 logN 代替

5.×【解析】与图的顶点数有关

6.×【解析】频率相同时，编码也不可能相同；任何两个不同字符编码都不同

7.×【解析】链表可以不一致

三.填空题(每空 2 分，共 8 分)

1.出　入　**2.**有　无

四.简要回答问题(共 8 分)

1.$N=N_0+N_2=N_2+1+N_2=2*N_2+1=39,N_2=19$。

2.数据结构概念包括:

（1）数据的逻辑结构：数据的逻辑结构只抽象地反映数据元素之间的逻辑关系，它与数据的存储无关，是独立于计算机的；（2）数据的存储结构：数据的存储结构是数据的逻辑结构在计算机存储器里的实现（亦称为映象）；（3）数据的运算：数据的运算定义在数据的逻辑结构之上，每种逻辑结构都有一个运算的集合。常用的运算有:查找、插入、删除、更新、排序等。

抽象数据类型 ADT：指一个数学模型以及定义在此数学模型上的一组操作。抽象数据类型需要通过固有数据类型（高级编程语言中已实现的数据类型）来实现。抽象数据类型是与表示无关的数据类型，是一个数据模型及定义在该模型上的一组运算。对一个抽象数据类型进行定义时，必须给出它的名字及各运算的运算符名，即函数名，并且规定这些函数的参数性质。一旦定义了一个抽象数据类型及具体实现，程序设计中就可以像使用基本数据类型那样，十分方便地使用抽象数据类型。

五.综合算法题(共 16 分)

（1）

```
void InsertSort(int A[], int m, int &n, int x){
    int p;
  for (i=0;i<n;i++){
        if (x<A[i]){                    // 判断新数据在原数组中的位置
            p=i;
            break;
        }
    }
    for(i=n-1;i>=p;i--)
        A[i+1]=A[i];                    // 在 p 位置之后的数据全部往后挪一位
    A[p]=x;                             // 把新数据放入 p 位置
    n=n+1;
}
```

（2）第一个与最后一个交换，第二个与倒数第二个交换，以此类推。

```
void reverse(int A[ ], int n){
    int i, t;
    for(i=0;i<n/2;i++){
      t=A[i];
      A[i]=A[n-i-1];
      A[n-i-1]=t;
    }
}
```

（3）利用双重循环，将每个值依次与其后面的值相比较，如果有相同的则删除该元素即可。删除时，可以使用将后面元素依次向前移动一位，同时总长度减一的方式。

```
void delDuplicate(int A[], int &n){
    int i, j, k;
    for(i = 0; i < n; i ++) {
```

```
        for(j = i+1; j < n; j ++) {              //对后面每个元素比较，去重。
            if(A[j] == A[i]) {                   //发现重复元素。
                for(k = j+1; k < n; k ++)        //依次前移一位。
                    A[k-1] = A[k];
                n--;                             //总长度减一。
            }
        }
    }
}
```

六.综合算法题(16 分)

1.有序二叉树的中序遍历是从小到大的序列，但题意却是要从大到小输出，故需要采用右根左的遍历方式。(这里给出非递归的方法)

```
void FindX(BinarySearchTree* BST , int x){
    stack<BinarySearchTree*> stack;                          //初始化栈
     BinarySearchTree* binary_tree_curr = BST;               //保存当前结点
        while(binary_tree_curr || !stack.empty()){
            if(binary_tree_curr->rchild){                    //右孩子非空
               stack.push(binary_tree_curr);                 //当前结点入栈
               binary_tree_curr = binary_tree_curr->rchild;  //遍历右子树
            }
            else{              //右孩子为空，则打印当前结点并遍历左子树
                if(binary_tree_curr->data >= x){
                    cout<<binary_tree_curr->data<<" ";
                }
                binary_tree_curr = binary_tree_curr->lchild;
                //如果为空，且栈不空，则将栈顶节点出栈，并输出该节点，
                //同时将它的左孩子设为当前节点，继续判断，直到当前节点不为空
                while(!binary_tree_curr && !stack.empty()){
                    binary_tree_curr = stack.top();
                    if(binary_tree_curr->data >= x){
                       cout<<binary_tree_curr->data<<" ";
                    }
                    stack.pop();
                    binary_tree_curr = binary_tree_curr->lchild;
                }
            }
        }
}
```

【补充】递归算法：
InOrder(BiTree T, int x) {
if(!T) {
InOrder(T->rc);//先右
	printf(T->data);
	else return 0;
	InOrder(T->lc);//后左
	}

if(T->data>=x)	}

2 判断除 V0 以外的其它顶点是否有与 V0 相连的长度不超过 k 的简单路径，采用递归的方法，path 记录路径（也可以没有 path），visited 记录是否访问过。

```
int Search(MGraph G, int x, int y,int k,int visited[],int path[], int d){
    int n, i;
    ArcNode *p;
    visited[x] = l;
    d ++;
    path[d] = x;
    if(x == y && d <= k) return 1;
    p=G.vertices[x].firstarc;
    while(p !=NULL) {
        n=p->adjvex ;
        if(visited[n]==0){
            if ((j = Search(G, n, y, k, visited, path, d))==1)
                return j;
        }
        p = p->nextarc ;
    }
  visited[x]=0;
    d --;
    return 0;
}
int main( ){
    int visited[G.vexnum];
    int path[G.vexnum];
    for(int i = 1;i < G.vexnum,;i ++){
        memset(visited, 0, sizeof(visited));
        if(Search(G, 0, i, k, visited, path, -1)) //k 是要判断的长度
            cout << G.getValue(i) <<" ";
    }
    return 0;
}
```

西北工业大学软微学院未知年份试题

一.选择题(每题 2 分，满分 30 分)

答案速查：CDBCD　CABAB　CDCCD

4.C【总结】各排序算法总结如下：①各排序复杂度：

排序算法	平均时间复杂度	最好情况	最坏情况	空间复杂度	排序方式	稳定性
冒泡排序	$O(n^2)$	O(n)	$O(n^2)$	O(1)	In-place	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	In-place	不稳定
插入排序	$O(n^2)$	O(n)	$O(n^2)$	O(1)	In-place	稳定
希尔排序	O(nlogn)	$O(nlog^2n)$	$O(nlog^2n)$	O(1)	In-place	不稳定
归并排序	O(nlogn)	O(nlogn)	O(nlogn)	O(n)	Out-place	稳定
快速排序	O(nlogn)	O(nlogn)	$O(n^2)$	O(nlogn)	In-place	不稳定
堆排序	O(nlogn)	O(nlogn)	O(nlogn)	O(1)	In-place	不稳定
计数排序	O(n+k)	O(n+k)	O(n+k)	O(k)	Out-place	稳定
桶排序	O(n+k)	O(n+k)	$O(n^2)$	O(n+k)	Out-place	稳定
基数排序	O(n×k)	O(n×k)	O(n×k)	O(n+k)	Out-place	稳定

②算法复杂度与初始状态无关：选择排序、归并排序、堆排序基数排序；

③总排序趟数与初始状态无关：除了快速排序、优化过后的冒泡排序，其他的排序算法都满足；

④元素总比较次数与初始状态无关：基数排序、选择排序；

⑤元素总移动次数与初始状态无关：基数排序、归并排序。

7.A【解析】线性链表的特点是顺序存储特点

8.B【解析】满二叉树共有 2^k-1 个结点，$65<2^7$，因此高度为 7。

10.B【解析】每一条边关联两个顶点，因而所有顶点度之和等于边数的 2 倍。

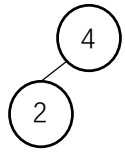
二.简答题(每题 5 分，满分 25 分)

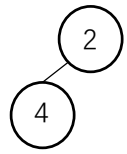
1.(1)判断 n 是否为质数，是返回 1，否返回 0；(2)O($\sqrt{n}$)

2.low<=high　(low+high)/2　high=mid-1　low=mid+1

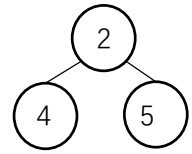
3.

(1)插入 4

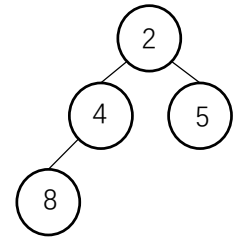




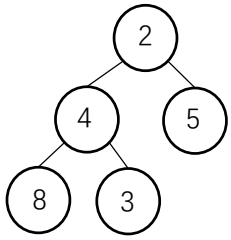
(2)插入 2，小顶堆进行变化

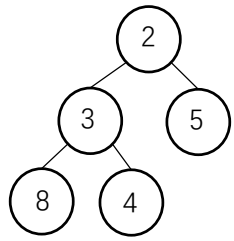


(3)插入 5



(4)插入 8



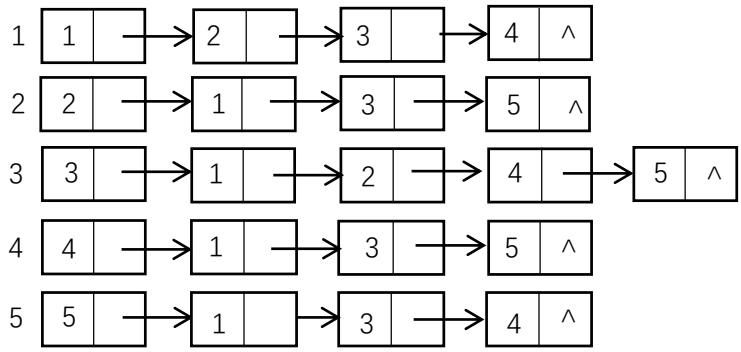


(5)插入 3，小顶堆进行变化

4. $\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$

$(1,2)$ $(1,3)$ $(1,4)$
$(2,1)$ $(2,3)$ $(2,5)$
$(3,1)$ $(3,2)$ $(3,4)$ $(3,5)$
$(4,1)$ $(4,3)$ $(4,5)$
$(5,2)$ $(5,3)$ $(5,4)$

邻接表：



$1 \overset{3}{-} 2 \rightarrow (1,2)3$

$4 \overset{4}{-} 6 \rightarrow (4,6)4$

$1 \overset{5}{-} 3 \rightarrow (1,3)5$

$2 \overset{6}{-} 3$

$1 \overset{8}{-} 4 \rightarrow (1,4)8$

$3 \overset{9}{-} 6$

$2 \overset{10}{-} 5 \rightarrow (2,5)10$

$3 \overset{12}{-} 5$

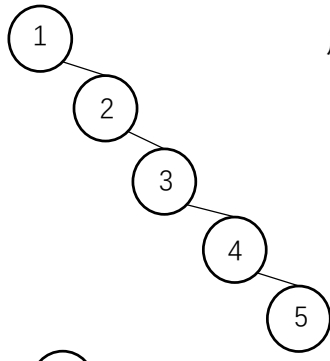
$3 \overset{15}{-} 4$

$5 \overset{18}{-} 6$

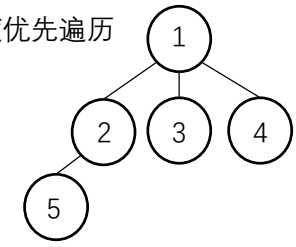
$4 \overset{20}{-} 7 \rightarrow (4,7)20$

$6 \overset{25}{-} 7$

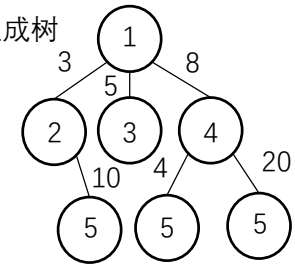
深度优先遍历



广度优先遍历



5.最小生成树



如右式所示

三.设计题(每题 10 分，满分 20 分)

1. int judgebitree(bitree *bt1,bitree *bt2) { //判断两个二叉树是否相同。

```
    if (bt1==0 && bt2==0)//两棵树对应位置都为空返回 1
        return 1;
    else if (bt1==0 || bt2==0 ||bt1->data!=bt2->data) //两棵树的当前节点只有一个为空或者两棵树的当前节点的值不同。
        return 0;
    else
        return judgebitree(bt1->lchild,bt2->lchild)*judgebitree(bt1->rchild,bt2->rchild);
}
```

2.【解析】见《西北工业大学 2014-2015 学年第一学期期末考试 (软微)》四设计题 1 小题

2009 年研究生入学考试计算机统考 408

一.单项选择题：每小题 2 分

答案速查：BCDBC　BADAB

1．B【解析】考查栈和队列的特点及应用。

C 和 D 直接排除，缓冲区的特点需要先进先出，若用栈，先进入缓冲区的数据则要排 队到最后才能打印，不符题意，故选 B。

2．C【解析】考查栈的最大递归深度。

时刻注意栈的特点是先进后出。出入栈的详细过程见下表。

序号	说明	栈内	栈外	序号	说明	栈内	栈外
1	a 入栈	a		8	e 入栈	ae	bdc
2	b 入栈	ab		9	f 入栈	aef	bdc
3	b 出栈	a	b	10	f 出栈	ae	bdcf
4	c 入栈	ac	b	11	e 出栈	a	bdcfe
5	d 入栈	acd	b	12	a 出栈		bdcfea
6	d 出栈	ac	bd	13	g 入栈	g	bdcfea
7	c 出栈	a	bdc	14	g 栈		bdcfeag

栈内的最大深度为 3，故栈 S 的容量至少是 3。

3．**D**【解析】考查二叉树的特殊遍历。

分析遍历后的结点序列，可以看出根结点是在中间被访问的，而右子树结点在左子树之 前，得遍历的方法是 RNL。本题考查的遍历方法并不是二叉树的三种基本遍历方法，对于 考生而言，重要的是要掌握遍历的思想。

4．**B**【解析】考查平衡二叉树的定义。

根据平衡二叉树的定义有，任意结点的左、右子树高度差的绝对值不超过 1。而其余三个答案均可以找到不符合的结点。

5．**C**【解析】考查完全二叉树的特点。

完全二叉树比满二叉树只是在最下面一层的右边缺少了部分叶结点，而最后一层之上是个满二叉树，并且只有最后两层有叶结点。第 6 层有叶结点则完全二叉树的高度可能为 6 或 7，显然树高为 7 时结点更多。若第 6 层上有 8 个叶结点，则前六层为满二叉树，而第 7 层缺失了 8×2=16 个叶结点，故完全二叉树的结点个数最多为 27-1-16=111 个结点。

6．**B**【解析】考查森林和二叉树的转换。

森林与二叉树的转换规则为“左孩子右兄弟”。在最后生成的二叉树中，父子关系在对应森林关系中可能是兄弟关系或原本就是父子关系。

情形Ⅰ：若结点 v 是结点 u 的第二个孩子结点，在转换时，结点 v 就变成结点 u 第一个 孩子的右孩子，符合要求。

情形Ⅱ：结点 u 和 v 是兄弟结点的关系，但二者之中还有一个兄弟结点 k，则转换后， 结点 v 就变为结点 k 的右孩子，而结点 k 则是结点 u 的右孩子，符合要求。

情形Ⅲ：结点 v 的父结点要么是原先的父结点或兄弟结点。若结点 u 的父结点与 v 的父 结点是兄弟关系，则转换之后，不可能出现结点 u 是结点 v 的父结点的父结点。

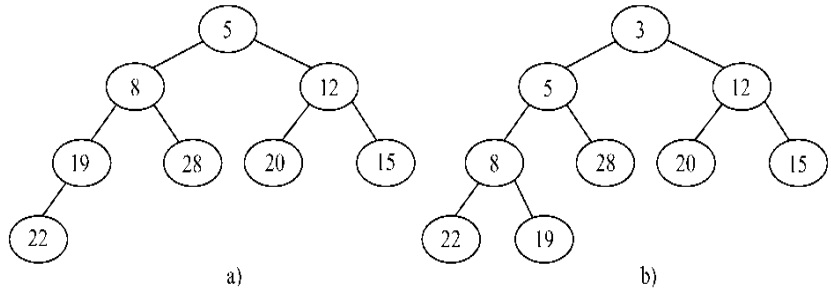
7．**A**【解析】考查无向连通图的特性。

每条边都连接了两个结点，则在计算顶点的度之和时，这条边都被计算了两次，故所有顶点的度之和为边数的两倍，显然必为偶数。而Ⅱ和Ⅲ则不一定正确，如对顶点数 N≥1 无 向完全图不存在一个顶点的度为 1，并且边数与顶点数的差要大于 1。

8．**D**【解析】考查 m 阶 B-树的定义。

选项 A、B 和 C 都是 B-树的特点，而选项 D 则是 B+树的特点。注意区别 B-树和 B+树 各自的特点。

9．**A**【解析】考查小根堆的调整操作。小根堆在逻辑上可以用完全二叉树来表示，根据关键序列得到的小顶堆的二叉树形式如下图 a 所示。



插入关键字 3 时，先将其放在小顶堆的末端，再将该关键字向上进行调整，得到的结果如图 b 所示。所以，调整后的小顶堆序列为 3，5，12，8，28，20，15，22，19。

10．**B**【解析】考查各排序算法的特点。

解答本题之前要对不同排序算法的特点极为清楚。对于冒泡排序和选择排序而言，每趟过后都能确定一个元素的最终位置，而由题目中所说，前两个元素和后两个元素均不是最小 或最大的两个元素并按序排列。答案 D 中的二路归并排序，第一趟排序结束都可以得到若 干个有序子序列，而此时的序列中并没有两两元素有序排列。插入排序在每趟排序结束后能 保证前面的若干元素是有序的，而此时第二趟排序后，序列的前三个元素是有序的，符合其 特点。

二.综合应用题

41.【解答】

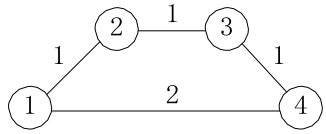


图 A-5

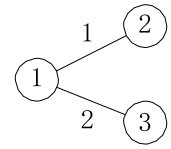


图 A-6

该方法不一定能（或不能）求得最短路径。(4 分) 举例说明：(6 分)

图 A-5 中，设初始顶点为 1，目标顶点为 4，欲求从顶点 1 到顶点 4 之间的最短路径， 显然这两点之间的最短路径长度为 2。利用给定方法求得的路径长度为 3，但这条路径并不 是这两点之间的最短路径。

图 A-6 中，设初始顶点为 1，目标顶点为 3，欲求从顶点 1 到顶点 3 之间的最短路径。 利用给定的方法，无法求出顶点 1 到顶点 3 的路径。

【评分说明】

①若考生回答“能求得最短路径”，无论给出何种证明，均不给分。

②考生只要举出类似上述的一个反例说明“不能求得最短路径”或答案中体现了“局部最优不等于全局最优”的思想，均可给 6 分；若举例说明不完全正确，可酌情给分。

42.【解答】(1) 算法的基本设计思想 (5 分)： 问题的关键是设计一个尽可能高效的算法，通过链表的一趟遍历，找到倒数第 k 个结点的位置。算法的基本设计思想：定义两个指针变量 p 和 q，初始时均指向头结点的下一个结 点 (链表的第一个结点)。p 指针沿链表移动，当 p 指针移动到第 k 个结点时，q 指针开始与 p 指针同步移动；当 p 指针移动到最后一个结点时，q 指针所指示结点为倒数第 k 个结点。 以上过程对链表仅进行一遍扫描。

(2) 算法的详细实现步骤 (5 分)：

① count=0，p 和 q 指向链表表头结点的下一个结点；

② 若 p 为空，转⑤；

③ 若 count 等于 k，则 q 指向下一个结点；否则，count=count+1；

④ p 指向下一个结点，转②；

⑤ 若 count 等于 k，则查找成功，输出该结点的 data 域的值，返回 1；否则，说明 k 值 超过了线性表的长度，查找失败，返回 0；

⑥ 算法结束。

（3）算法实现（5 分）：

```
typedef int ElemType;  //链表数据的类型定义
typedef struct LNode{  //链表结点的结构定义 ElemType    data;    //结点数据
struct Lnode *link; //结点链接指针
} *LinkList;
int   Search_k(LinkList list,int k){
//查找链表 list 倒数第 k 个结点，并输出该结点 data 域的值
LinkList p=list->link,q=list->link; //指针 p、q 指示第一个结点
int count=0;
while(p!=NULL){   //遍历链表直到最后一个结点 if(count<k) count++;     //计数，若 count<k 只移动 p
else q=q->link;p=p->link;  //之后让 p、q 同步移动
} //while if(count<k)
return 0; //查找失败返回 0
else {     //否则打印并返回 1
printf("%d",q->data); return 1;
}
} //Search_k
```

【提示】算法程序题，如果能够写出数据结构类型定义、正确的算法思想都会至少给一半 以上分数，如果能用伪代码写出自然更好，比较复杂的地方可以直接用文字表达。

【评分说明】

①若所给出的算法采用一遍扫描方式就能得到正确结果，可给满分 15 分；若采用两遍或多遍扫描才能得到正确结果的，最高给 10 分；若采用递归算法得到正确结果的，最高给 10 分；若实现算法的空间复杂度过高（使用了大小与 k 有关的辅助数组），但结果正确，最高给 10 分；若实现的算法的空间复杂度过高（使用了大小与 k 有关的辅助数组），但结果正 确，最高给 10 分。

②参考答案中只给出了使用 C 语言的版本，使用 C++/JAVA 语言正确实现的算法同样给 分。

③若在算法基本思想描述和算法步骤描述中因文字表达没有非常清晰地反映出算法的 思路，但在算法实现中能够清晰看出算法思想和步骤且正确，按照①的标准给分。

④若考生的答案中算法基本思想描述、算法步骤描述或算法实现中部分正确，可酌情给分。

2010 年研究生入学考试计算机统考 408

一.单项选择题

答案速查：DCDCB　ACBBD　A

1．**D**【解析】考查限定条件的出栈序列。

A 可由 in，in，in，in，out，out，in，out，out，in，out，out 得到； B 可由 in，in，in，out，out，in，out，out，in，out，in，out 得到； C 可由 in，in，out，in，out，out，in，in，out，in，out，out 得到；D 可由 in，out，in，in，in，in，in，out，out，out，out，out 得到，但题意要求不允许 连续三次退栈操作，故 D 错。

2．**C**【解析】考查受限的双端队列的出队序列。

A 可由左入，左入，右入，右入，右入得到；B 可由左入，左入，右入，左入，右入得 到；D 可由左入，

左入，左入，右入，左入得到。所以不可能得到 C。

3．**D**【解析】考查线索二叉树的基本概念和构造。

题中所给二叉树的后序序列为 dbca。结点 d 无前驱和左子树，左链域空，无右子树， 右链域指向其后继结点 b；结点 b 无左子树，左链域指向其前驱结点 d；结点 c 无左子树， 左链域指向其前驱结点 b，无右子树，右链域指向其后继结点 a。

4．**C**【解析】考查平衡二叉树的插入算法。

插入 48 以后，该二叉树根结点的平衡因子由-1 变为-2，失去平衡，需进行两次旋转（先 右旋后左旋）操作。

5．**B**【解析】考查树结点数的特性。

设树中度为 i（i=0，1，2，3，4）的结点数分别为 Ni，树中结点总数为 N，则树中各结 点的度之和等于 N-1，即 N=1+N1+2N2+3N3+4N4=N0+N1+N2+N3+N4，根据题设中的数据， 即可得到 N0=82，即树 T 的叶结点的个数是 82。

6．**A**【解析】考查赫夫曼树的特性。

赫夫曼树为带权路径长度最小的二叉树，不一定是完全二叉树。赫夫曼树中没有度为 1 的结点，B 正确；构造赫夫曼树时，最先选取两个权值最小的结点作为左、右子树构造一棵 新的二叉树，C 正确；赫夫曼树中任一非叶结点 P 的权值为其左、右子树根结点权值之和， 其权值不小于其左、右子树根结点的权值，在与结点 P 的左、右子树根结点处于同一层的 结点中，若存在权值大于结点 P 权值的结点 Q，那么结点 Q 的兄弟结点中权值较小的一个 应该与结点 P 作为左、右子树构造新的二叉树。综上可知，赫夫曼树中任一非叶结点的权 值一定不小于下一层任一结点的权值。

7．**C**【解析】考查图的连通性。

要保证无向图 G 在任何情况下都是连通的，即任意变动图 G 中的边，G 始终保持连通， 首先需要 G 的任意 6 个结点构成完全连通子图 G1，需 15 条边，然后再添一条边将第 7 个 结点与 G1 连接起来，共需 16 条边。

8．**B**【解析】考查拓扑排序序列。

图中有 3 个不同的拓扑排序序列，分别为 abced、abecd、aebcd。

9．**B**【解析】考查折半查找的过程。

具有 n 个结点的判定树的高度为 log2n +1，长度为 16，高度为 5，所以最多比较 5 次。

10．**D**【解析】考查快速排序。

递归次数与各元素的初始排列有关。如果每一次划分后分区比较平衡，则递归次数少；如果划分后分区不平衡，则递归次数多。递归次数与处理顺序无关。

11．**A**【解析】考查各种排序算法的过程。

看第一趟可知仅有 88 被移到最后。如果是希尔排序，则 12，88，10 应变为 10，12，88。因此排除希尔排序。 如果是归并排序，则长度为 2 的子序列是有序的。因此可排除归并排序。 如果是基数排序，则 16，5，10 应变为 10，5，16。因此排除基数排序。 可以看到，每一趟都有一个元素移到其最终位置，符合冒泡排序特点。

二.综合应用题

41.【解答】（1）由装载因子为 0.7，数据总数为 7，得一维数组大小为 7/0.7=10，数组下标为 0～9。 所构造的散列函数值见下表所示。

key	7	8	30	11	18	9	14
H(key)	0	3	6	5	5	6	0

采用线性探测再散列法处理冲突，所构造的散列表见下表所示。

地址	0	1	2	3	4	5	6	7	8	9

关键字	7	14		8		11	30	18	9	

（2）查找成功时，是根据每个元素查找次数来计算平均长度的，在等概率的情况下，各关键字的查找次数见下表所示。

key	7	8	30	11	18	9	14
次数	1	1	1	1	3	3	2

故 $ASL_{成功}$=查找次数/元素个数=(1+2+1+1+1+3+3)/7=12/7。 这里要特别防止惯性思维。查找失败时，是根据查找失败位置计算平均次数，根据散列函数 MOD 7，初始只可能在 0～6 的位置。等概率情况下，查找 0～6 位置查找失败的次数见下表所示。

H(key)	0	1	2	3	4	5	6
次数	3	2	1	2	1	5	4

故，$ASL_{失败}$=查找次数/散列后的地址个数=(3+2+1+2+1+5+4)/7=18/7。

【扩展】已知一个线性序列{38,25,74,63,52,48}，假定采用散列函数 Hash(key)=key%7 计算散列地址，散列存储在表 A[10]中。如果采用线性探测法解决冲突，且各元素的查找概率相等，则在该散列表上查找不成功的平均查找长度为_____（中国科学院大学 2016）

A.2.60　　B.3.14　　C.3.71　　D.4.33

【解析】这道题也是为数不多的计算“查找不成功且线性序列个数≠key%7 的 7≠表 A[10]的 10”的题，对该方面考查更为深刻，且翻阅西工大历年真题，仅考察过查找成功的概率，对于查找不成功的概率还未涉及，建议大家把这个知识点巩固，以防出反套路题而失分。

首先我们还是按照散列函数分别计算散列序列中的散列地址，并依次存入表 A 中，当出现 key 值相同的情况用线性探测法解决冲突（请注意，解决冲突的公式 $H_i=(H(key)+d_i)MOD\ m$，其中 H(key)为哈希函数，m 为**哈希表表长**，此时 m=10）。

Hash(38)=38%7=3，存入 A[3]中；Hash(25)=25%7=4，存入 A[4]中；Hash(74)=74%7=4，冲突，线性探测法得(4+1)MOD 10=5，存入 A[5]中；Hash(63)=63%7=0，存入 A[0]中；Hash(52)=52%7=3，冲突，线性探测法得(3+1)MOD 10=4，又冲突，…，直到线性探测法得(5+1)MOD 10=6，存入 A[6]；Hash(48)=48%7=6，冲突，线性探测法得(6+1)MOD 10=7，存入 A[7]。

因此查找成功的概率=(1+1+1+2+4+2)/6=1.83；而对于查找失败的次数，就是找到该数到下一个空白位置的次数（如果该位置为空，则次数为 1），根据散列函数 MOD7，我们只计算 A[0]~A[6]这 7 个数。比如我们与 0 号位置比较，不是我们要找的数，然后与 1 号位置相比（因为是否为空需要进入该结构，与该位置比较后才知道是否为空，除非散列之后空的位置被标记），1 号为空，证明我们查找失败了，因此查找失败次数为 2，依次类推，$ASL_{不成功}$=查找次数/散列后的地址个数=(2+1+1+6+5+4+3)/7=3.14。结果如下表所示。因此本题选 B。

表 A	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
存储元素	63			38	25	74	52	48		
查找成功	1			1	1	2	4	2		
查找失败	2	1	1	6	5	4	3			

42.【解答】（1）算法的基本设计思想：

可以将这个问题看作是把数组 ab 转换成数组 ba（a 代表数组的前 p 个元素，b 代表数 组中余下的 n-p 个元素），先将 a 逆置得到 $a^{-1}b$，再将 b 逆置得到 $a^{-1}b^{-1}$，最后将整个 $a^{-1}b^{-1}$ 逆置得到 $(a^{-1}b^{-1})^{-1}$=ba。设 Reverse 函数执行将数组元素逆置的操作，对 abcdefgh 向左循环移动 3（p=3）个位置的过程如下：

Reverse(0,p-1)得到 cbadefgh； Reverse(p,n-1)得到 cbahgfed； Reverse(0,n-1)得到 defghabc。

注：Reverse 中，两个参数分别表示数组中待转换元素的始末位置。

（2）使用 C 语言描述算法如下：

```
void Reverse(int R[],int from,int to) {
    int i,temp;
    for(i=0;i<(to-from+1)/2;i++){
```

```
void Converse(int R[],int n,int p){
    Reverse(R,0,p-1);
    Reverse(R,p,n-1);
```

```
        temp=R[from+i];R[from+i]=R[to-i];R[to-i]=temp;}
    }//Reverse
```

```
    Reverse(R,0,n-1);
}
```

（3）上述算法中 3 个 Reverse 函数的时间复杂度分别为 O(p/2)、O((n-p)/2)和 O(n/2)，故所设计的算法的时间复杂度为 O(n)，空间复杂度为 O(1)。

【另解】借助辅助数组来实现。

算法思想：创建大小为 p 的辅助数组 S，将 R 中前 p 个整数依次暂存在 S 中，同时将 R 中后 n-p 个整数左移，然后将 S 中暂存的 p 个数依次放回到 R 中的后续单元。时间复杂度为 O(n)，空间复杂度为 O(p)。

2011 年研究生入学考试计算机统考 408

一.单项选择题

答案速查：ABBCC　DACDA　B

1．**A**【解析】考查时间复杂度的计算。

在程序中，执行频率最高的语句为“x=2*x”。设该语句共执行了 t 次，则 $2^{t+1}=n/2$，故 t=log2(n/2)-1=log2n-2，得 T(n)=O(log2n)。

2．**B**【解析】考查出栈序列。

出栈顺序必为 d_c_b_a_，e 的顺序不定，在任一个“_”上都有可能。

3．**B**【解析】考查循环队列的性质。

入队时由于要执行(rear+1)%n 操作，所以如果入队后指针指向 0，则 rear 初值为 n-1，而由于第一个元素在 A[0]中，插入操作只改变 rear 指针，所以 front 为 0 不变。

4．**C**【解析】考查完全二叉树的性质。

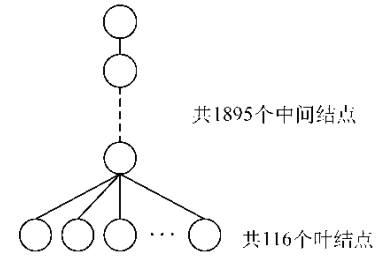
根据完全二叉树的性质，最后一个分支结点的序号为 768/2=384，故叶子结点的个数为 768-384=384。

5．**C**【解析】考查二叉树的遍历算法。

前序序列为 LRN，后序序列为 NLR，由于前序序列和后序序列刚好相反，故不可能存在一个结点同时存在左右孩子，即二叉树的高度为 4。1 为根结点，由于根结点只能有左孩子（或右孩子），因此，在中序序列中，1 或在序列首或在序列尾，ABCD 皆满足要求。仅考虑以 1 的孩子结点 2 为根结点的子树，它也只能有左孩子（或右孩子），因此，在中序序列中，2 或在序列首或序列尾，ABD 皆满足要求。

6．**D**【解析】考查树和二叉树的转换。

树转换为二叉树时，树中每一个分支结点的所有子结点中的最右子结点无右孩子，根结点转换后也没有右孩子， 因此，对应的二叉树中无右孩子的结点个数=分支结点数+1=2011-116+1=1896。通常本题应采用特殊法解，设题意中的树是如下图所示的结构，则对应的二叉树中仅有前 115 个叶结点有右孩子，故无右孩子的结点个数=2011-115=1896。



7．**A**【解析】考查二叉排序树的查找过程。

在二叉排序树中，左子树结点值小于根结点，右子树结点值大于根结点。在选项 A 中，当查找到 91 后再向 24 查找，说明这一条路径（左子树）之后查找的数都要比 91 小，而后面却查找到了 94，因此错误。

8．**C**【解析】考查图的基本概念。

回路对应于路径，简单回路对应于简单路径，故 I 错误；稀疏图是边比较少的情况，此时用邻接矩阵必将浪费大量的空间，应该选用邻接表，故 II 错误。存在回路的图不存在拓扑 序列，故III正确。

9．**D**【解析】考查散列表的性质。

Hash 表的查找效率取决于：哈希函数、处理冲突的方法和装填因子。显然，冲突的产生概率与装填因子

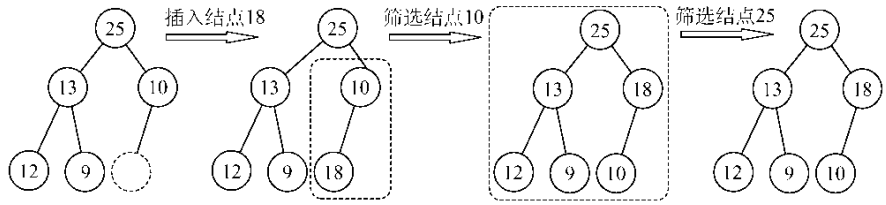
（即表中记录数与表长之比）的大小成正比，Ⅰ错误。冲突是不可避免的，但处理冲突的方法应避免非同义词之间地址的争夺，Ⅲ正确。

10．**A**【解析】考查排序的基本特点。

对绝大部分内部排序而言，只适用于顺序存储结构。快速排序在排序的过程中，既要从后向前查找，也要从前向后查找，因此宜采用顺序存储。

11．**B**【解析】考查堆的调整。

首先 18 与 10 比较，交换位置，再与 25 比较，不交换位置。共比较了 2 次，调整的过程如下图所示。

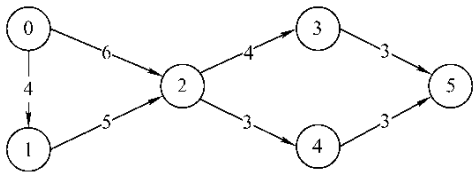


二.综合应用题

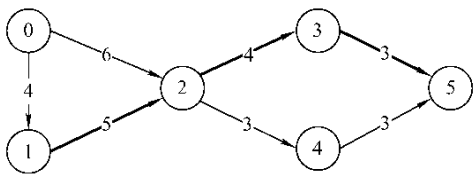
41.【解答】（1）用“平移”的思想，将前 5 个、后 4 个、后 3 个、后 2 个、后 1 个元素，分别移动到矩阵对角线（“0”）右边的行上。图 G 的邻接矩阵 ***A*** 如下所示。

$$A=\begin{pmatrix} 0 & 4 & 6 & \infty & \infty & \infty \\ \infty & 0 & 5 & \infty & \infty & \infty \\ \infty & \infty & 0 & 4 & 3 & \infty \\ \infty & \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & \infty & 0 & 3 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

（2）根据上面的邻接矩阵，画出有向带权图 G，如下图所示。



3）即寻找从 0 到 5 的最长路径。得到关键路径为 0→1→2→3→5（如下图所示粗线表示），长度为 4+5+4+3=16。



42.【解答】（1）算法的基本设计思想如下。

分别求出序列 A 和 B 的中位数，设为 a 和 b，求序列 A 和 B 的中位数过程如下：

① 若 a=b，则 a 或 b 即为所求中位数，算法结束。

② 若 a<b，则舍弃序列 A 中较小的一半，同时舍弃序列 B 中较大的一半，要求舍弃的长度相等。

③ 若 a>b，则舍弃序列 A 中较大的一半，同时舍弃序列 B 中较小的一半，要求舍弃的长度相等。

在保留的两个升序序列中，重复过程 1)、2)、3)，直到两个序列中只含一个元素时为止，较小者即为所求的中位数。

（2）算法的实现如下：

```
int M_Search(int A[],int B[],int n){
```

```
        //分别表示序列 A 和 B 的首位数、末位数和中位数
        int s1=0,d1=n-1,m1,s2=0,d2=n-1,m2;
        while(s1!=d1||s2!=d2){
          m1=(s1+d1)/2;
          m2=(s2+d2)/2;
          if(A[m1]==B[m2])
            return A[m1];    //满足条件 1）
          if(A[m1]<B[m2]){   //满足条件 2）
            if((s1+d1)%2==0){ //若元素个数为奇数
              s1=m1;     //舍弃 A 中间点以前的部分，且保留中间点
              d2=m2;     //舍弃 B 中间点以后的部分，且保留中间点
            }
            else{//元素个数为偶数
              s1=m1+1;  //舍弃 A 中间点及中间点以前部分
              d2=m2;     //舍弃 B 中间点以后部分且保留中间点
            }
          }
          else{   //满足条件 3）
            if((s1+d1)%2==0) {//若元素个数为奇数
              d1=m1;     //舍弃 A 中间点以后的部分，且保留中间点
              s2=m2;     //舍弃 B 中间点以前的部分，且保留中间点
            }
            else{//元素个数为偶数
              d1=m1;     //舍弃 A 中间点以后部分，且保留中间点
              s2=m2+1;  //舍弃 B 中间点及中间点以前部分
            }
          }
        }
      return A[s1]<B[s2]? A[s1]:B[s2];
      }
```

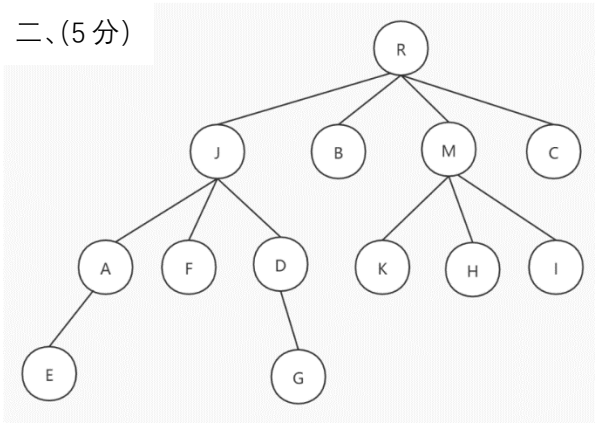
（3）算法的时间复杂度为 $O(\log_2 n)$，空间复杂度为 O(1)。

西北工业大学 2014 年研究生入学考试(879)

一、单项选择题（每小题 2 分，共 30 分）

答案速查：BADCC　ABBAA　BACBD

二、(5 分)



三、(7 分)由装载因子为 0.7，数据总数为 7，得一维数组大小为 7/0.7=10，数组下标为 0～9。所构造的散列函数值见表 1。

表 1

key	7	8	30	11	18	9	14
H(key)	0	3	6	5	5	6	0

采用线性探测再散列法处理冲突，所构造的散列表见表 2。

表 2

地址	0	1	2	3	4	5	6	7	8	9
关键字	7	14		8		11	30	18	9	

表 3

key	7	8	30	11	18	9	14
次数	1	1	1	1	3	3	2

故 $ASL_{成功}$=查找次数/元素个数=(1+2+1+1+1+3+3)/7=12/7。

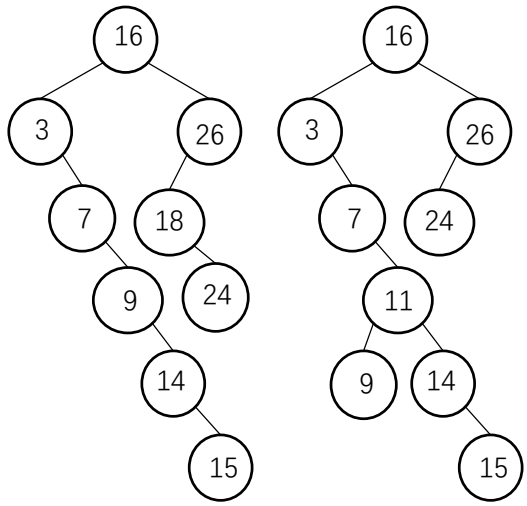
查找失败时，是根据查找失败位置计算平均次数，根据散列函数 mod 7，初始只可能在 0～6 的位置。等概率情况下，查找 0～6 位置查找失败的查找次数见表 4。

表 4

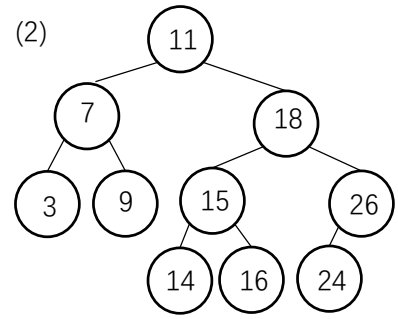
H(key)	0	1	2	3	4	5	6
次数	3	2	1	2	1	5	4

故 $AS_{不成功}$=查找次数/散列后的地址个数=(3+2+1+2+1+5+4)/7=18/7。

四、(1) 删除 11 节点　　删除 18 节点



(2)



过程略

五、(12 分) 答: (1)分两种情况讨论①当*px 的右子树不为空时,则从*px 的右孩子开始,沿其左孩子往下查找,直至找到一个没有左孩子的结点为止则该结点为止,则该节点为*px 在中序序列中的后继。②当*px 的右子树为空时,则沿*px 的双亲指针链向上查找,直至找到其左子树中包含*px 的最年轻祖先,则该祖先结点为*px 的在中序序列中的后继。

```
(2)BinTree f34(BinTree px){
   BinTree   q=px->rchild;
   if (q!=null) { //沿左孩子往下查找
      px=q;
      while(px->lchild! =NULL)
         px=px->lchild;
   }
   else{          //沿双亲指针链向上查找
      while(px!=NULL&&px->rchild==q){
         q=px;
         px=px->parent;
      }
   }
   return px; //返回找到的中序序列后继结点的指针或者无后继结点时返回空指针
}
```

六、(13 分)(1)算法的基本设计思想:

可以将这个问题看作是把数组 ab 转换成数组 ba（a 代表数组的前 p 个元素，b 代表数组中余下的 n-p 个元素），先将 a 逆置得到 $a^{-1}b$，再将 b 逆置得到 $a^{-1}b^{-1}$，最后将整个 $a^{-1}b^{-1}$ 逆置得到 $(a^{-1}b^{-1})^{-1}$=ba。设 Reverse 函数执行将数组元素逆置的操作，对 abcdefgh 向左循环移动 3（p=3）个位置的过程如下:

Reverse(0,p-1)得到 cbadefgh； Reverse(p,n-1)得到 cbahgfed； Reverse(0,n-1)得到 defghabc。

注：Reverse 中，两个参数分别表示数组中待转换元素的始末位置。

(2)使用 C 语言描述算法如下:

```
void Reverse(int R[],int from,int to) {
   int i,temp;
   for(i=0;i<(to-from+1)/2;i++) {
      temp=R[from+i];R[from+i]=R[to-i];R[to-i]=temp;}
```

```
}//Reverse
void Converse(int R[],int n,int p){
Reverse(R,0,p-1);
Reverse(R,p,n-1);
Reverse(R,0,n-1);
}
```

(3)上述算法中 3 个 Reverse 函数的时间复杂度分别为 O(p/2)、O((n-p)/2)和 O(n/2)，故所设计的算法的时间复杂度为 O(n)，空间复杂度为 O(1)。

另解，借助辅助数组来实现。

算法思想：创建大小为 p 的辅助数组 S，将 R 中前 p 个整数依次暂存在 S 中，同时将 R 中后 n-p 个整数左移，然后将 S 中暂存的 p 个数依次放回到 R 中的后续单元。

时间复杂度为 O(n)，空间复杂度为 O(p)。

西北工业大学 2015 年研究生入学考试(879)

一、选择题（每题 2 分，满分 30 分）

答案速查：DBDBD　　DCACA　　DABBD

1.D【解析】本题考查线性表的存储。

线性表是最简单和最常用的一种数据结构，线性表是由相同类型的结点组成的有限序列。线性表的存储方式可以是顺序存储，也可以是链式存储。

题目中要求对线性表的操作是在最后一个元素之后插入一个元素和删除最后一个元素，如果用链式存储结构，在插入一个元素和删除一个元素后，要修改相应结点的指针域；但如果用容量足够大的顺序表存储，那么只要在表尾直接插入一个元素和删除一个元素后即可，不需要其他的操作，是最节省运算时间的方法。

4.B【解析】因为连通分量是无向图的极大连通子图，其中极大的含义是将依附于连通分量中顶点的所有边都加上，所以，连通分量中可能存在回路。

二、简答题（每题 5 分，满分 25 分）

1.后序遍历：EDCBIHJGFA

2.

地址	0	1	2	3	4	5	6	7	8	9	10	11	12
关键字	78		15	03		57	45	20	31		23	36	12

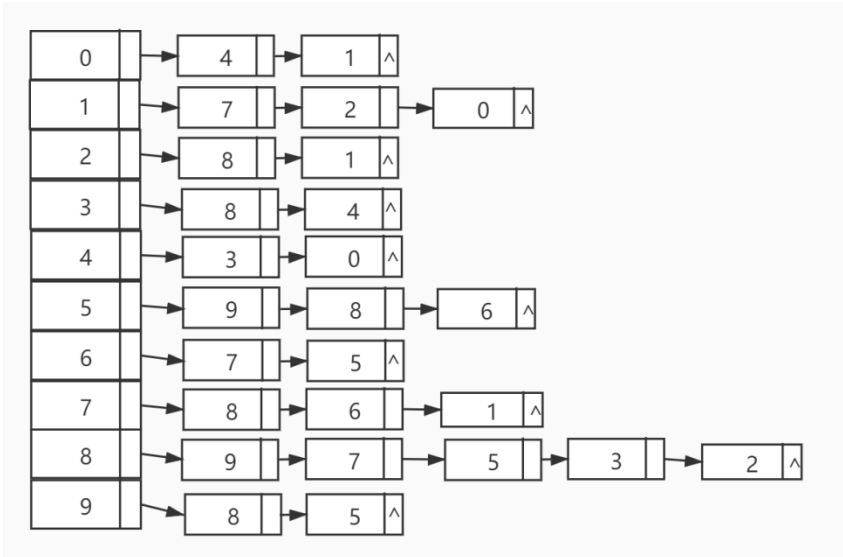
ASL $_{\text{查找成功}}$=14/10=1.4

3. A.Edge=

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0	1	0	0
2	0	1	0	0	0	0	0	0	1	0
3	0	0	0	0	1	0	0	0	1	0
4	1	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	1	1

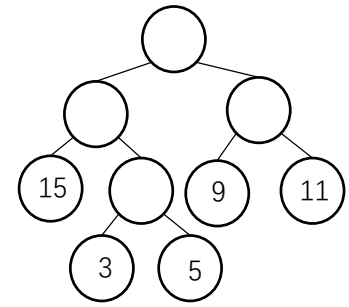
6	0	0	0	0	0	1	0	1	0	0
7	0	1	0	0	0	0	1	0	1	0
8	0	0	1	1	0	1	0	1	0	1
9	0	0	0	0	0	0	1	1	1	0

深度优先搜索遍历：0178956243；广度优先搜索遍历：0147236859



4.交换二叉树的左右子树的递归算法

5.



WPL=7*2+3*3+5*3+9*2+11*2=78

三、设计题（每题 10 分，满分 20 分）

1.

```
void DeleteSame(Linklist *L) {
    LNode *p,*q,*s;
    p = (*L)->next;
    for(p;p!=NULL;p=p->next) {
        s=p;      //s 指向要删除结点的前驱
        for(q=p->next;q!=NULL; ) {
            if(q->data==p->data) {
                s->next=q->next;
                free(q);
                q=s->next;
            }
            Else {
                s=q;
                q=q->next;
            }
        }
    }
}
```

【参考链接】https://blog.csdn.net/qq_42552533/java/article/details/86496142

2.typedef struct node {datatype data; struct node *lchild,*rchild;} bitree;

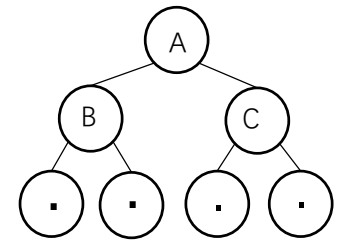
```
bitree *q[20]; int r=0,f=0,flag=0;
void preorder(bitree *bt, char x) {
   if (bt!=0 && flag==0)
if (bt->data==x) { flag=1; return;}
else {r=(r+1)% 20; q[r]=bt; preorder(bt->lchild,x); preorder(bt->rchild,x); }
}
void parent(bitree *bt,char x) {
    int i;
    preorder(bt,x);
    for(i=f+1; i<=r; i++) if (q[i]->lchild->data==x || q[i]->rchild->data) break;
```

```
    if (flag==0) printf("not found x\n");
    else if (i<=r) printf("%c",bt->data); else printf("not parent");
}
```

【参考链接】https://blog.csdn.net/qq_41149269/article/details/80900800

西北工业大学 2016 年研究生入学考试(879)

一.选择题(每小题 2 分，共 20 分)



第 2 题图

答案速查：CADBA　DCABD

2.A【解析】扩充二叉树的概念要了解，这个概念在 801 中没有出现过。扩充二叉树是二叉树中的一种，是指在二叉树中出现空子树的位置增加空树叶，所形成的二叉树。如右图所示。一般二叉树的先序、中序和后序任一序列不能唯一确定一棵二叉树，而通过扩充二叉树可以通过先序确定中序和后序。

叶子结点总是比非叶子结点多 1.对于这类我们不熟悉的问题，可以采用类推法，多画几组图，找到其中的规律；或通过题目来看，并未给出 m 的含义，因此 BCD 均无根据，A 即为所求。

4.B【解析】有序序列是快排的最坏情况。如果选比较次数最少的则为插入排序。

6.D【解析】循环队列采用的方法是：假设向量 data[maxsize]是一个首尾相接的圆环，即 data[0]接在 data[maxsize-1]之后，我们将这种意义下的向量称循环向量，并将循环向量中的队列称为循环队列。若当前尾指针等于向量的上界，则再做入队列操作时，令尾指针等于向量的下界，这样就利用到已被删除的元素空间，克服假上溢现象。因此入队操作时，在循环意义下的尾指针加 1 操作可描述为：if(rear > =maxsize) near：=0; else rear++；如果利用“模运算”，上述循环意义下的尾指针加 1 操作，可以更简洁地描述为：rear=(rear+1) %maxsize。同样，出队操作时，在循环意义下的头指针加 1 操作，也可利用“模运算”来实现：front：=(front+1) %maxsize。因此本题应选 front=(front+1)%(m+1)，可能是题目有问题。

7.C【解析】折半查找需要先对查找的数据集合排序，并且每次要获得数据列表的中间位置，通过数组这种顺序存储结构，只要一次索引就能获得中间值，如果是链式结构，就每次都要从头遍历到中间位置，耗费大量时间

9.B【解析】第 i 列表示终点为 i 顶点的那些边，第 i 行则是起点为 i 的那些边，非零则表示存在入度，故第 i 列非 0 元素的个数之和。

二.简答题(每小题 7 分，共 35 分)

1.将中缀形式化为二叉树表示，则其后缀形式为 ABC*+DE/-

2.线索二叉树：在二叉链表表示的二叉树中存在大量的空指针，若利用这些空链域存放指向其直接前驱或后继的指针，称之为线索。加上线索的二叉树称为线索二叉树。

在中序线索二叉树查找直接前驱：

(1)若左标志为 1，则左链域指示其前驱；

(2)若左标志为 0，则遍历左子树时最后访问的一个结点(左子树最右下的结点)为其前驱。

在中序线索二叉树查找直接后继：

(1)若右标志为 1，则右链域指示其后继；

(2)若右标志为 0，则遍历右子树时访问的第一个结点(右子树最左下的结点)为其后继。

3.① 扫描原三元组，并按照先列序后行序的原则进行。即第一遍从原三元组的第一行开始向下搜索列号为 1

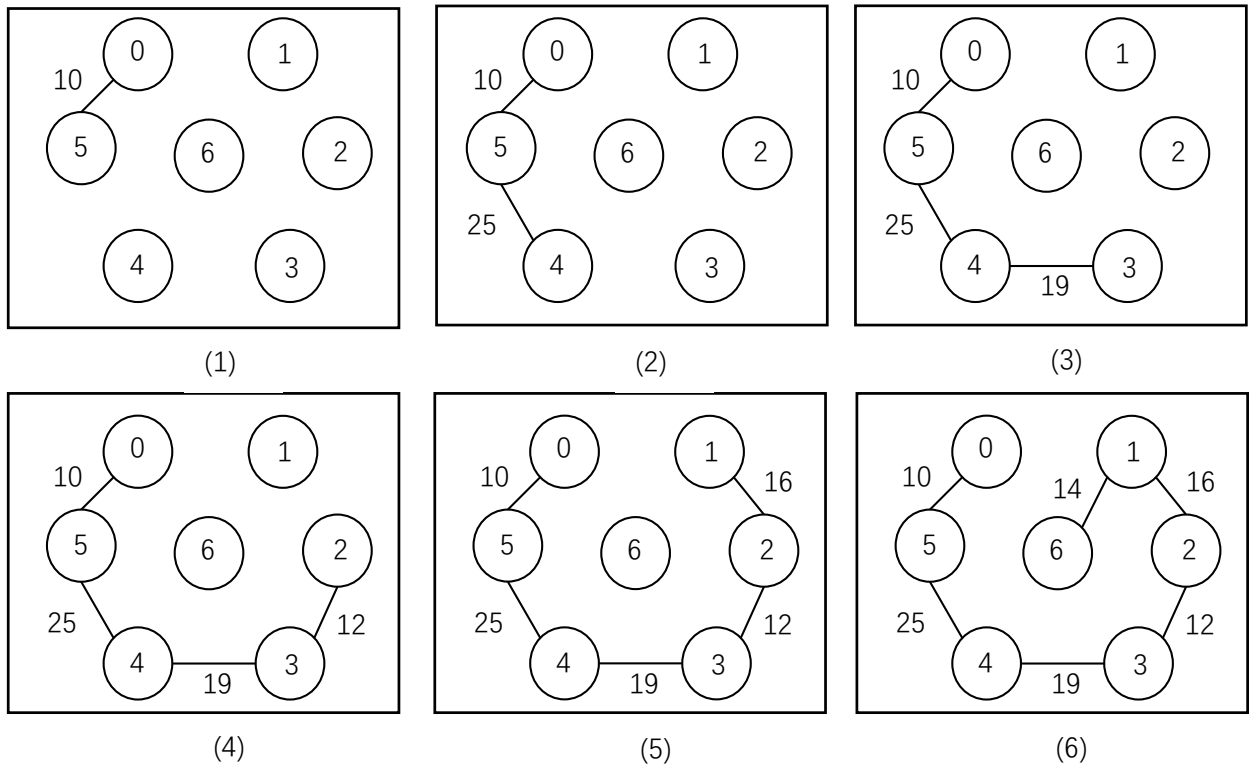
的元素，只要找到则顺序存入转置后的三元组表；

② 第 2 遍仍然从原三元组的第一列开始向下搜索列号为 2 的元素，只要找到则顺序存入转置后的三元组表；

.....

③ 第 n 遍（n 代表原三元组总共有多少列）将列号为 n 的元素，依次填入转置后的三元组表。

4.



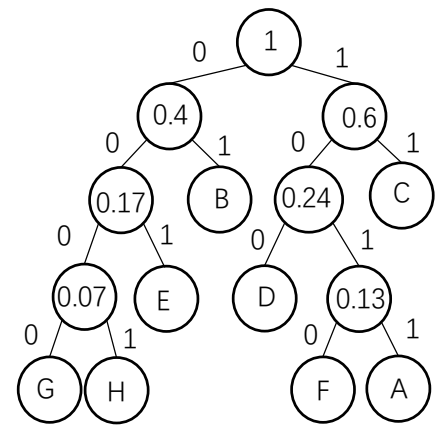
(1) (2) (3)

(4) (5) (6)

Prim 算法构造最小生成树的过程

5.V2->V0：最短路径为 3；V2->V1：最短路径为 3+1=4，即 V2->V0->V1；V2->V3：最短路径为 3+1+2=6，即 V2->V0->V1->V3。

三.设计题(每小题 10 分，共 20 分)



第 1 题图

1.A-1011；B-01；C-11；D-100；E-001；F-1010；G-0000；H-0001

ASL=(0.03+0.04+0.06+0.07)×4+(0.1+0.11)×3+(0.23+0.36)×2=2.61

哈夫曼生成和实现算法：

```
#include<iostream>
#include<cstring>
#include<algorithm>
using namespace std;
#define inf 9999999
int n,m;
typedef char **huffmancode;
 typedef struct {
       int weigth;
       int parent, lchild, rchild;
 }htnode, *huffmantree;
```

```
void                createhuffmancode(huffmantree
&ht,huffmancode &hc,int n) {
      hc=new char*[n+1];
      char *cd=new char[n];
      for(int i=1;i<=n;i++)   {
```

```
void Select(huffmantree &ht,int s,int &s1,int &s2){
    s1=s2=0;
    int min1,min2;
    min1=min2=inf;
    for(int j=1;j<=s;j++)
    if(ht[j].parent==0)   {
        if(ht[j].weigth<=min1) {
        min2=min1;
        min1=ht[j].weigth;
         s2=s1;s1=j;
        }
        else if(ht[j].weigth<=min2) {
            min2=ht[j].weigth;
            s2=j;
        }
    }
}

void createhuffmantree(huffmantree &ht, int n){
    if (n <= 1) return;
      m = 2 * n - 1;
    ht = new htnode[m + 1];
    for (int i = 1; i <= m; ++i) {
        ht[i].parent = 0; ht[i].lchild = 0; ht[i].rchild = 0;
    }
    for (int i = 1; i <= n; i++) cin >> ht[i].weigth;
    for (int i = n + 1; i <= m; i++) {
        int s1,s2;
        Select(ht,i-1,s1,s2);
        ht[s1].parent = i; ht[s2].parent = i;
        ht[i].lchild = s1; ht[i].rchild = s2;
        ht[i].weigth = ht[s1].weigth+ ht[s2].weigth;
    }
}
        int start=n-1,c=i,f=ht[i].parent;
        while(f!=0) {
            --start;
            if(ht[f].lchild==c) cd[start]='0';
            else cd[start]='l';
            c=f;f=ht[f].parent;
        }
        hc[i]=new char[n-start];
        strcpy(hc[i],&cd[start]);
    }delete cd;
}

int main( ) {
    cout<<"请输入叶子结点数目(n>1):"<<endl;
    cin>>n;
    cout<<"请输入权值"<<endl;
    huffmantree ht;
    huffmancode hc;
    createhuffmantree(ht,n);
    for(int i=1;i<=m;i++) {
        cout<<ht[i].weigth<<' ';
    }
    cout<<endl;
    createhuffmancode(ht,hc,n);
    for(int i=1;i<=n;i++) {
        cout<<ht[i].weigth<<' '<<hc[i]<<endl;
    }
    delete ht;
}
```

2.设带表头结点的双向链表的定义为：

```
typedef int ElemType;
typedef struct dnode { //双向链表结点定义
  ElemType data;//数据
  struct dnode *lLink,*rLink; //结点前驱与后继指针
} DblNode;
typedef DblNode *DblList//双向链表

void sort(DblNode *L) {
  DblNode *s=L->rlink; //指针 s 指向待插入结点，初始时指向第一个结点
```

```
    while(s!=NULL) { //处理所有结点
        pre=L; p=L->lLink;//指针 p 指向待比较的结点，pre 是 p 的前驱指针
        while(p!=NULL && s->data<p->data) { //循 lLink 链寻找结点*s 的插入位置
            pre=p; p=p->lLink;
        }
        pre->lLink=s; s->lLink=p; s=s->rLink; //结点*s 在 lLink 方向插入到*pre 与*p 之间
    }
}
```

西北工业大学 2017 年研究生入学考试(879)

一.选择题(每小题 2 分，共 20 分)

答案速查：DACDD　CCC

1.D【解析】栈是一种只在栈顶进行插入和删除操作的线性表，因此具有先进后出的特点；而队列是一种在队头进行删除操作，在队尾进行插入操作的线性表，因此具有先进先出的特点。这也是这两种线性表的最本质的区别。

2.A【解析】各个排序算法的复杂度见《西北工业大学软微学院未知年份试题三》一 选择题的 4 小题。

4.D【解析】本题要求根据二叉树的先序遍历和中序遍历求后序遍历。我们可以根据这棵二叉树的先序和中序遍历画出这棵二叉树，然后再得出其后序遍历结果。根据先序和中序来构造二叉树的规则是这样的：

首先看先序遍历序列 ABDECF，先序遍历中第一个访问的结点是 A，这说明 A 是二叉树的根结点(因为先序遍历顺序是：根，左，右)。然后看中序遍历序列 DBEAFC，中序中 A 前面有结点 DBE，后面有结点 FC。这说明 DBE 是 A 的左子树，FC 是 A 的右子树(因为中序遍历顺序是：左，根，右)。

再回到先序遍历序列中看 DBE 的排列顺序(此时可以不看其他的结点)，我们发现在先序遍历序列中 B 排在最前面，所以 B 是 A 的左子树的根结点。

接下来又回到了中序遍历序列，中序遍历序列中 D 在 B 的前面，E 在 B 的后面，所以 D 是 B 的左子树，E 是 B 的右子树。

对于 A 的右子树，可同样依此规则得出。由此，可构造二叉树，然后对这棵二叉树进行后序遍历，得到 DEBFCA。

5.D【解析】有向图 G 中弧数目的取值范围：0 < =e < =n(n-1)。有 n(n-1)条弧的有向图称为有向完全图。

6.C【解析】平衡二叉树又称 AVL 树。它或者是一棵空树，或者是具有下列性质的二叉树。

①左子树和右子树都是平衡二叉树；②左子树和右子树的深度之差的绝对值不超过 1；③二叉树上节点的平衡因子定义为该节点的左子树的深度减去它的右子树的深度。

由此可见，平衡二叉树上所有节点的平衡因子只可能是-1，0，1。只要二叉树上有一个节点的平衡因子的绝对值大于 1，则该二叉树就是不平衡的。

7.C【解析】平衡的二叉树是对二叉树的一种“平衡化”处理。结点的平衡因子定义为其右子树高度减去左子树高度。若任一结点的平衡因子均取值-1，或 0，或+1，则此二叉排序树为平衡的二叉排序树。根据这个原则，可得最少结点数为 7。

二.填空题(每小题 2 分，共 20 分)

1.顺序存储 链式存储　**2.**i = 2; i <= (int)sqrt(n); i ++　**3.**2n　n-1　n+1

4.14 种【解析】公式：B[n] = C[n,2n] / (n+1)。将 n=4 带入上述公式，可以得出，组合数 C[n,2n]的 n 为上标，2n 为下标，将 n=4 代入公式，B[4] = C[4,8] / (4+1) = 8! / (4! * 4! * 5) = 8*7*6/(4*3*2) = 14。

5.34，16，69，27，75，42，82，95【解析】快速排序是通过一趟排序选定一个关键字介于“中间”的记录，从而使剩余记录可以分成两个子序列分别继续排序，通常称该记录为“枢轴”。

一趟快速排序的具体做法：附设两个指针 low 和 high，它们的初值分别指向文件的第一个记录和最后一个记录。设枢轴记录(通常是第一个记录)的关键字为 pivotkey，则首先从 high 所指位置起向前搜索，找到第一个关键字小于 pivotkey 的记录并与枢轴记录互相交换，然后从 low 所指位置起向后搜索，找到第一个关键字大于 pivotkey 的记录并与枢轴记录互相交换，重复这两步直至 low=high 为止。

三.简答题(每小题 8 分，共 16 分)

1.先序：ABCDEFGH；中序：BCAFEGDH；后序：CBFGEHDA。不能

2.略

四.编程题(9 分)

```
for x in range(1,m/3+1):
    for y in range(1,m/2+1):
        z=m-x-y;
        if (z%3==0) && (3*x+y*2+z/3==m):
            print('公鸡：%d 只，母鸡：%d 只，小鸡：%d 只' %(x,y,z))
```

时间复杂度：$O(m^2)$

五.应用题(10 分)

【注】见《西北工业大学 2007-2008 学年期末考试(软微 A 卷)》四大题

西北工业大学 2018 年研究生入学考试(879)

一．选择题（5*3）

答案速查：BDBCC

1.B【解析】用链表的形式表示的线性表最大的优势是能动态地、很方便地进行插入和删除操作。

2.D【解析】一个广义表的元素可以是子表，而子表的元素还可以是子表，形成一个多层次的结构，元素也可以是单个元素。

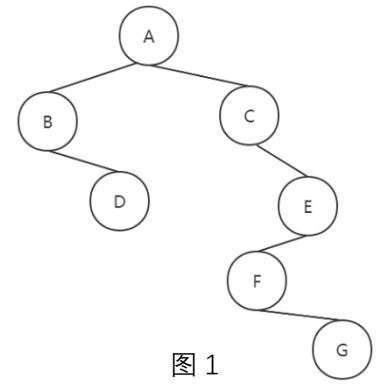


图 1

二、简答题（4*15）

1.如图 1 所示

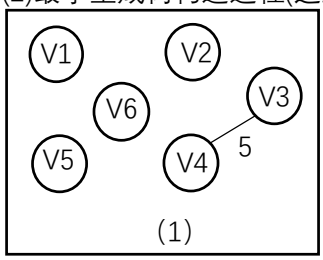
2.(1)先描述算法：Kruskal 算法：

初始化：$V_T=V,E_T=\varnothing$ 。即每个顶点构成一棵独立的树，T 此时是一个仅含|V|个顶点的森林；

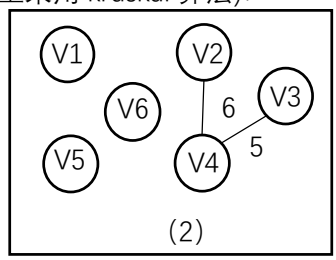
循环(重复下列操作至 T 是一棵树)：按 G 的边的权值递增顺序依次从 $E-E_T$ 中选择一条边，如果这条边加入 T 后不构成回路，则将其加入 E_T,否则舍弃，直至 E_T 中含有 n-1 条边。

所用数据结构：并查集

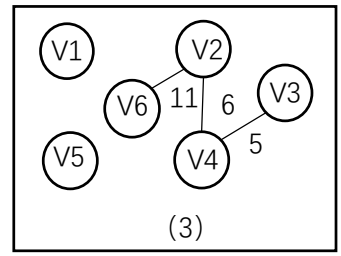
(2)最小生成树构造过程(这里采用 kruskal 算法)：



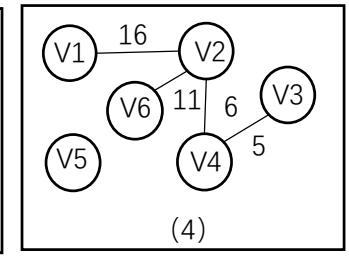
(1)



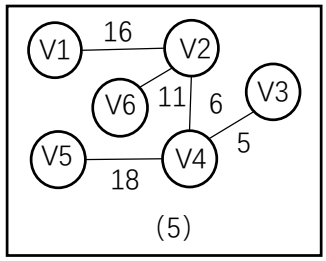
(2)



(3)

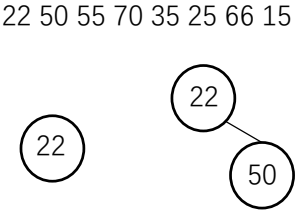


(4)

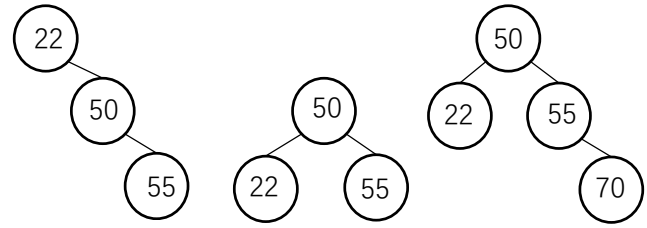


(5)

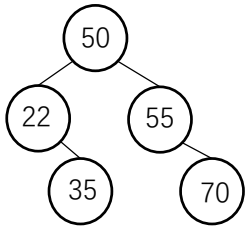
3. 22 50 55 70 35 25 66 15 10



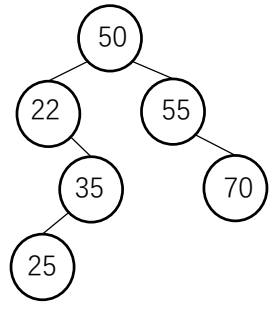
(1)插入 22　(2)插入 50



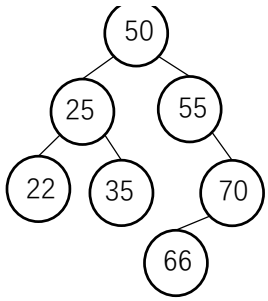
(3)插入 55，失去平衡，做单向左旋平衡处理　(4)插入 70

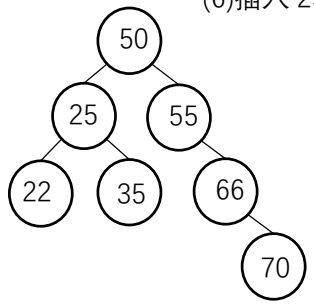


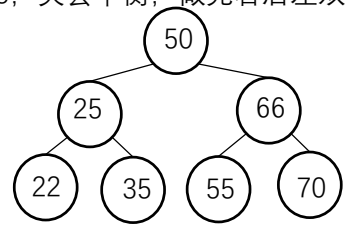
(5)插入 35

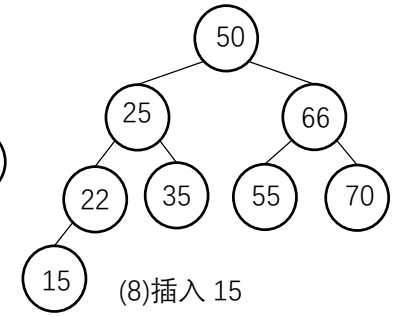


(6)插入 25，失去平衡，做先右后左双向平衡处理

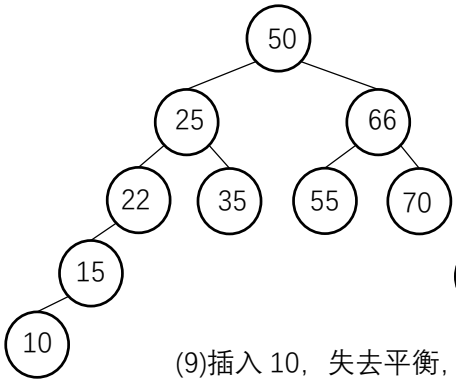


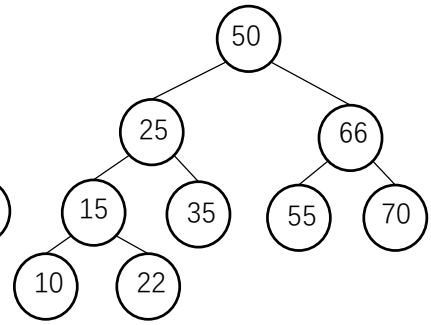




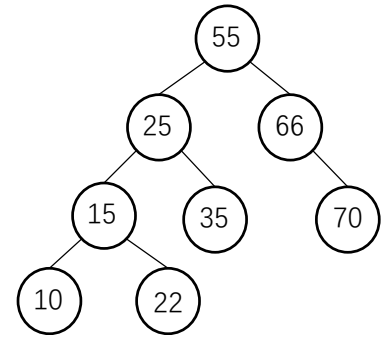


(7)插入 66，失去平衡，做先右后左双向平衡处理　(8)插入 15





(9)插入 10，失去平衡，做单向右旋平衡处理



删除键值为 50 的元素，将结点 50 与结点 55 交换，再删除。

4.此题只要你的算法能够实现多项式相加的基本操作，给出算法的主体，即可得分。本题我们提供一段完整的参考代码，请在 VIP 群文件里查阅，仅供参考。

西北工业大学 2019 年研究生入学考试(879)

1.后序遍历序列：DEBHIFGCA，二叉树如图 1 所示

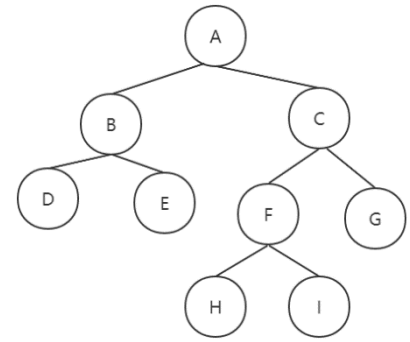


图 1

层次遍历关键算法：

```
void LevleOrder(BinTree T){ //从第一层开始，从左到右
BinTree Queue[max],p; //用一维数组表示队列，front 和 rear 分别表示队首和队尾指针
int front,rear;
    front=rear=0;
    if (T) { //若树非空
        Queue[rear++]=T; //根结点入队列
        while (front!=rear) {    // 队列非空
          p=Queue[front++];    // 队首元素出队列，并访问这个结点
          printf("%c",p->data);
          if (p->lchild!=NULL) Queue[rear++]=p->lchild; //左子树非空，入队列
          if (p->rchild!=NULL) Queue[rear++]=p->rchild;
        }
    }
}
```

层次遍历结果：ABCDEFGHI

2.(1)

0	1	2	∞	∞
1	0	3	4	5
2	3	0	6	7
∞	4	6	0	8
∞	5	7	8	0

(2)可以使用 Prim 算法求得最小生成树。最小生成树如图 2 所示。

Prim 算法：1).输入：一个加权连通图。当中顶点集合为 V，边集合为 E；

2).初始化：Vnew = {x}，当中 x 为集合 V 中的任一节点（起始点），Enew = {},为空；

3).反复下列操作，直到 Vnew = V：

a.在集合 E 中选取权值最小的边<u, v>，当中 u 为集合 Vnew 中的元素。而 v 不在 Vnew 集合当中。而且 v∈V（如果存在有多条满足前述条件即具有同样权值的边，则可随意选取当中之中的一个）；**b.**将 v 增加集合 Vnew 中，将<u, v>边增加集合 Enew 中。

4).输出：使用集合 Vnew 和 Enew 来描写叙述所得到的最小生成树。

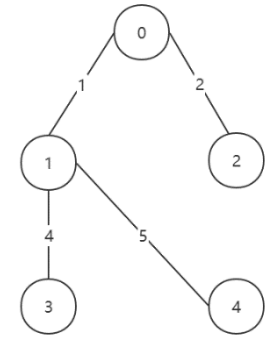


图 2

3.(1)快速排序使用分治法来把一个串分为两个子串。具体算法描述如下：

(a)从数列中挑出一个元素，称为“基准”（pivot）；

(b)重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆放在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区操作；

(c)递归地把小于基准值元素的子数列和大于基准值元素的子数列排序。

时间复杂度：O(nlog₂n)　空间复杂度：O(nlog₂n)

(2)第一趟 {4 7 20 16 21} 24 {34}；第二趟：4 {7 20 16 21} 24 34；第三趟：4 7 {20 16 21} 24 34；第四趟：4 7 {16} 20 {21} 24 34

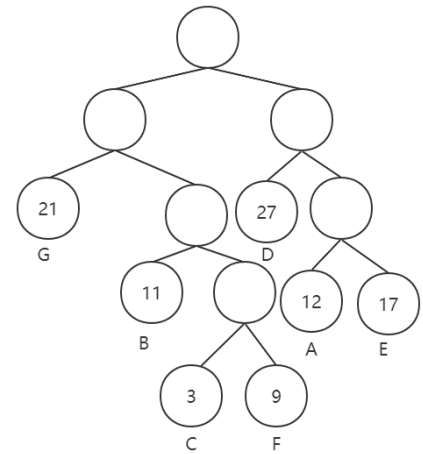


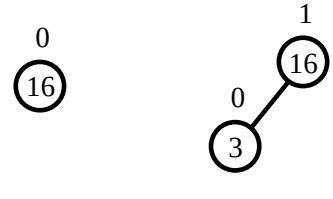
图 3

4.

	距离	路径
1->0	4	V{1,4,0 }
1->2	4	V{1,4,2}
1->3	6	V{1,4,0,3}
1->4	1	V{1,4}

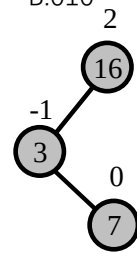
5.哈夫曼树如图 3 所示，哈夫曼编码：A:110　B:010　C:0110　D:10　E:111　F:0111　G:00

6.

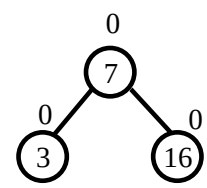


(a)插入 16

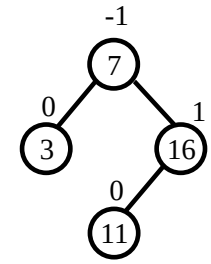
(b)插入 3



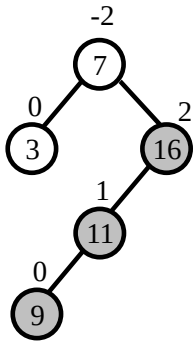
(c)插入 7



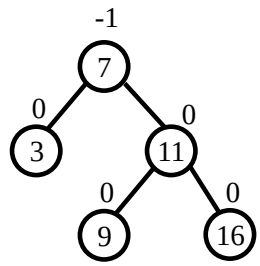
(d)LR 调整



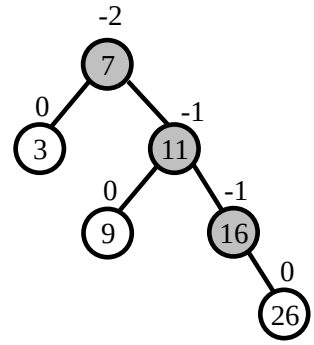
(e)插入 11



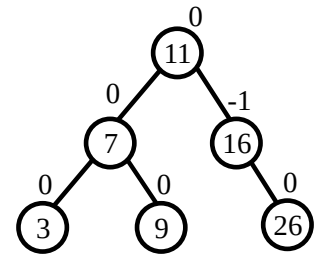
(f)插入 9



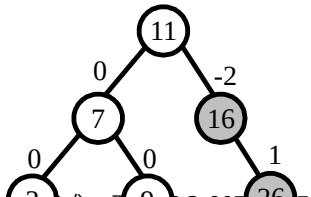
(g)LL 调整

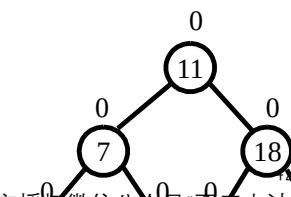


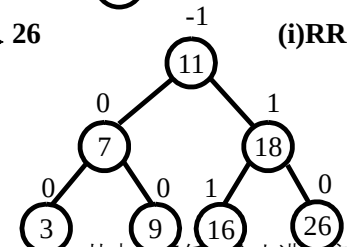
(h)插入 26



(i)RR 调整







7.此题不太清楚回忆的广义表格式是否正确，在此给出两种查找广义表数据域为 x 结点的算法示例，仅供参考

示例 1：

```
GLNode *Locate(GLNode *gl,ElemType x) { / /g 为带头结点的广义表
    GLNode *p=NULL;
      while (gl!=NULL) {
        if (gl->tag==0 && gl->val.data==x)
              p=gl;
        else if (gl->tag==1) {
          if (p==NULL)
            p=Locate(gl->val.sublist,x);
          else
            break;
        }
        gl=gl->link;
      }
      return(p);
}
```

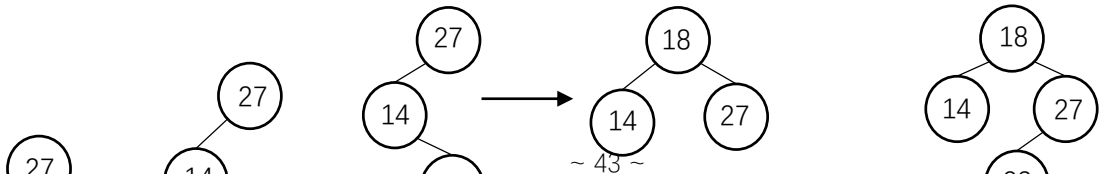
示例 2：

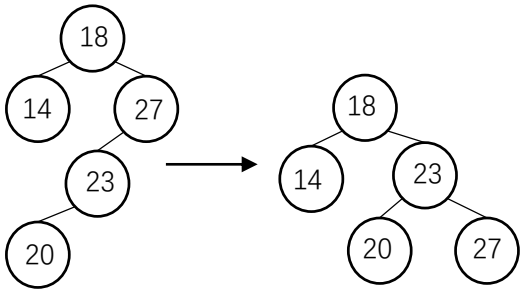
```
GLNode *Locate1(GLNode *gl,ElemType x) { //g 为带头结点的广义表
    GLNode *p=NULL;
    if (gl!=NULL) {
        if (gl->tag==0 && gl->val.data==x)
            p=gl;
        else if (gl->tag==1)
            p=Locate1(gl->val.sublist,x);
        if (p==NULL)    p=Locate1(gl->link,x);
        return(p);
    }
    else     return(NULL);
}
```

西北工业大学 2020 年研究生入学考试(879)

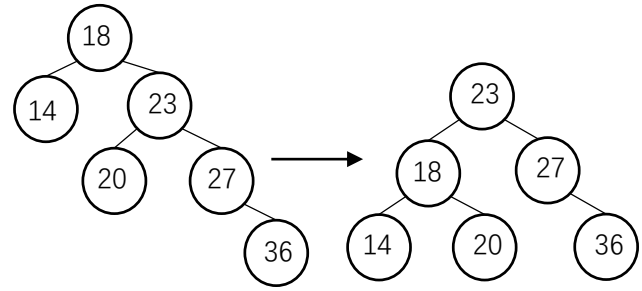
1.见《西北工业大学 2019 年研究生入学考试(879)》第 1 题。

2..(1)

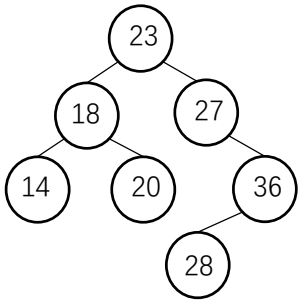




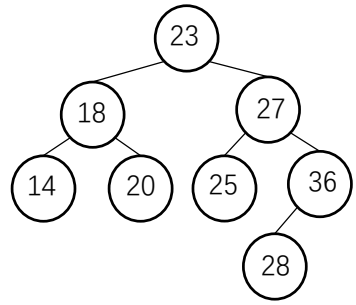
(5)插入 20，失去平衡，做单向右旋



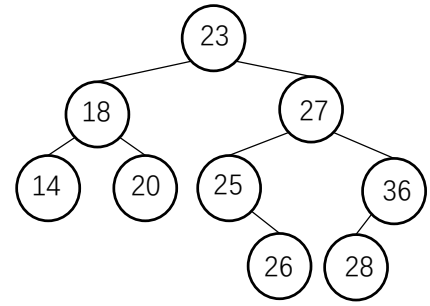
(6)插入 36，失去平衡，做单向左旋



(7)插入 28



(8)插入 25



(9)插入 26

(2) /* 查找特定值 */

```
void SearchData(int targ, BSTree *nod) {
    if (nod != NULL)  {
        if (nod->data == targ)    {
            printf("查找值存在，值为%d\n", nod->data);
        }
        else if (nod->data > targ)  {
            SearchData(targ, nod->left);  //递归查找左子树
        }
        else if (nod->data < targ) {
            SearchData(targ, nod->right); //递归查找右子树
        }
    }
    else if (nod == NULL)  {
        printf("查找值不存在\n");
    }
}
```

3.可以使用 DFS 判断：

从任一结点开始，进行一次深度优先遍历。深度优先遍历的结果是一个图的连通分量。当一次遍历没有访问到所有结点，那么就说明图是不连通。

算法步骤及相关分析：

a)算法步骤：

从顶点 v 出发，访问顶点 v，并令 visited[v] = 1；

依次查找 v 的所有邻接点 w，若 visited[w] 为 0，则从 w 出发，深度优先遍历图。

进行判断，遍历 visited 数组，若 visited 数组中有一个值不为 1，则说明该点未被访问，图不连通。

b)时间复杂度:：算法运行时间主要是耗费在寻找邻接 w 处，寻找一个符合条件的 w 的时间复杂度是 O(V)，V 个节点就是 O(V^2)，尽管可能不会寻找 V 次。

c)空间复杂度：空间复杂度仅耗费在辅助数组 visited[]上，空间复杂度为 O(V)。

d)改进：

设置全局静态变量 count，记录访问结点的数量，所以判断时不必遍历 visited 数组，只要判断 count 值是否等于结点数量 V 即可；

visited 数组设置为全局变量，与 BFS 函数共享。

```
//DFS 递归
public void    DFS(int[] visited, int v) {
      visited[v] = 1;
    judgeDFSCount ++;
    for(int i=0; i<this.vertexNum; i++) {
         if(this.a[v][i] != 0 && visited[i] == 0)    //寻找下一个有边的未访问结点
           DFS(visited, i);
     }
    }
//DFS 判断，调用 DFS 递归
public boolean DFSGraph() {
    judgeDFSCount = 0;                              //记录遍历的点个数，为全局变量
    boolean flag = false;
    visited = new int[this.vertexNum];       //初始数组就是全 0

    DFS(visited, 0);                                //从 0 号结点开始
    if(judgeDFSCount == this.vertexNum)       //如果遍历的点个数等于图的结点个数
     flag = true;                                   //说明一次 DFS 遍历就搜索了所有的点
    return flag;
}//代码仅供参考
```

```
4.void Dijkstra(MatGraph g，int v) {
      int dist[MAXV]，path[MAXV];
      int s[MAXV];
      int mindis,i,j,u;
      for (i=0;i<g.n;i++) {
         dist[i]=g.edges[v][i]; //距离初始化
         s[i]=0;   //s[]置空
         if (g.edges[v][i]<INF) //路径初始化
               path[i]=v;   //顶点 v 到 i 有边时
         else
              path[i]=-1;   //顶点 v 到 i 没边时
      }
      s[v]=1;//源点 v 放入 S 中
      for (i=0;i<g.n;i++) {   //循环 n-1 次
         mindis=INF;
         if (s[j]==0 && dist[j]<mindis) {
            u=j;
            mindis=dist[j];
         }//找最小路径长度顶点 u
         s[u]=1; //顶点 u 加入 S 中
         for (j=0;j<g.n;j++) //修改不在 s 中的顶点的距离
         if (s[j]==0)
              if (g.edges[u][j]<INF &&
dist[u]+g.edges[u][j]<dist[j]) {
                   dist[j]=dist[u]+g.edges[u][j];
                   path[j]=u;
                }//调整
      }
      Dispath(dist,path,s,g.n,v);    //输出最短路径
}
```

for (j=0;j<g.n;j++)

5.节点的结构体定义

```
typedef struct queue {
    datatype data;
    int high;//优先级
    struct queue *pNext;
}Queue, *PQueue;
//入队  入队的时候考虑优先级,优先级大的在前面
PQueue enq(PQueue phead, datatype data,int high) {
    PQueue pnew = (PQueue)malloc(sizeof(Queue));
    pnew->data = data;
    pnew->high = high;
    pnew->pNext = NULL;
    if (phead == NULL) {
        phead = pnew;//直接插入
    }
    else {
        //先判断头结点的优先级
        if (phead->high <= high) {
            pnew->pNext = phead;
            phead = pnew;
        }
        else {
            //从头结点开始一直循环到后一个节点的优先级小于 high 的
            PQueue ptemp = phead;
            //可能找到,也可能循环到尾部了还没找到,所以直接插在 ptemp 后面
            ptemp = ptemp->pNext;
        }
        pnew->pNext = ptemp->pNext;
        ptemp->pNext = pnew;
    }
  }
  return phead;
}
//出队
PQueue deq(PQueue phead, datatype *pdata) {
  if (phead == NULL) {
      return NULL;
  }
  else {
      *pdata = phead->data;//获取弹出的数据
      PQueue ptemp = phead->pNext;
      free(phead);
      phead = ptemp;
  }
}
//显示
void show(PQueue phead) {
  if (phead == NULL) {
      return;
  }
  else {
      printf("%5d(high:%d)",
```

```
        while (ptemp->pNext != NULL) {
            if (ptemp->pNext->high <= high) {
                break;
            }
```

```
phead->data,phead->high);
        show(phead->pNext);//递归调用
    }
}
```

6.算法主要函数如下：

```
//分组归并
void _Merge(int *a, int begin1, int end1, int begin2, int end2, int *tmp) {
    int index = begin1;
    int i = begin1, j = begin2;
    // 注意：当划分的区间足够小时,begin1==end1,begin2==end2
    while (i <= end1&&j <= end2){
     if (a[i]<=a[j])
            tmp[index++] = a[i++];
     else
            tmp[index++] = a[j++];
    }
    //将左边元素填充到 tmp 中
    while (i <= end1)
     tmp[index++] = a[i++];
    //将右边元素填充的 tmp 中
    while (j <= end2)
     tmp[index++] = a[j++];
    //将 tmp 中的数据拷贝到原数组对应的序列区间
    //注意:end2-begin1+1
    memcpy(a + begin1, tmp + begin1, sizeof(int)*(end2 - begin1 + 1));
}
//归并排序
void MergeSort(int *a, int left, int right, int *tmp) {
    if (left >= right)
     return;
    assert(a);
    //mid 将数组二分
    int mid = left + ((right - left) >> 1);
    //左边归并排序,使得左子序列有序
    MergeSort(a, left, mid, tmp);
    //右边归并排序,使得右子序列有序
    MergeSort(a, mid + 1, right, tmp);
    //将两个有序子数组合并
    _Merge(a, left, mid, mid + 1, right, tmp);
}
```

归并排序是稳定的排序算法。

7.【解】定义结构体数组 G，将物品编号、利润、重量作为一个结构体：例如 G[k]={1,10,2}求最优解，按利润/重量的递减序，有{5,6,1,6} {1,10,2,5}{6,18,4,9/2} {3,15,5,3} {7,3,1,3} {2,5,3,5/3} {4,7,7,1}

```
procedure KNAPSACK(P，W，M，X，n)
//P(1：n)和 W(1；n)分别含有按 P(i)/W(i)≥P(i+1)/W(i+1)排序的 n 件物品的效益值和重量。M 是背包的容量大小，而 x(1：n)是解向量//
real P(1：n)，W(1：n)，X(1：n)，M，cu；
integer i，n；
X←0  //将解向量初始化为零//
cu←M  //cu 是背包剩余容量//
for i←1 to n do
  if W(i)>cu then exit endif
  X(i) ←1
  cu←cu-W(i)
repeat
end GREEDY-KNAPSACK
```

根据算法得出的解：X=（1,1,1,1,1,0,0）获利润 52，而解（1,1,1,1, 0, 1,0）可获利润 54。因此贪心法不一定获得最优解。

附录

西北工业大学 2005-2006 学年第二学期期末考试(理学院)

一.单项选择题(每题 1 分，共 20 分)

答案速查：DBBDB　ADDAB　DABBC　CBBDC

12.A【解析】画出判定树，可知 ASL=(1*1+2*2+3*4+4*3)/10=2.9

二.填空题(每空 1 分，共 15 分)

1. O(n)　O(log_2n)　**2.**p->next = s;　**3.**q = p->next; p->next = q->next; free(q);　**4.**166
5.1　**6.**01122312　**7.**776　**8.**4　**9.**10　1000　**10.**90　**11.**4　**12.**3　**13.**先右后左

7.776【解析】LOC(a_{1111})=100+(3*5*8*1+5*8*1+8*1+1)*4=776

三.简答题(每题 4 分，共 12 分)

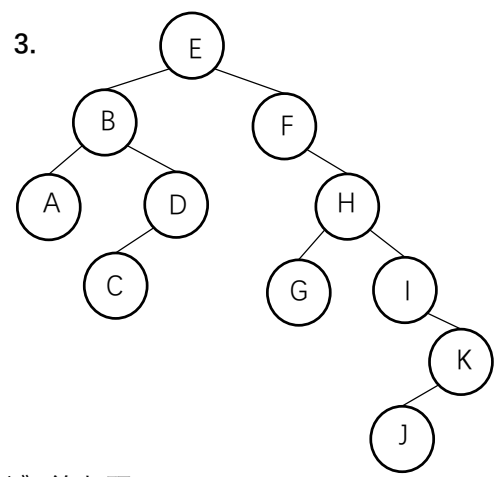
1.头指针指向链表中第一个结点的指针，若链表中有头结点，则头指针指向头结点；头结点是为了操作统一和方便设立的，放在第一个元素结点之前，其数据域一般无意义；首元结点是第一个元素的结点，它是头结点后边的第一个结点。

2.

行数	列数	非零元个数
4	4	6

i	j	v
1	2	12
2	1	-4
2	4	9
3	2	7
4	1	2
4	3	8

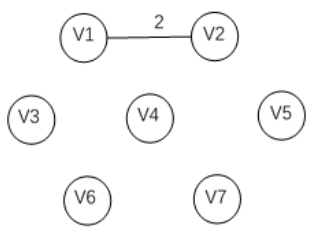
3.

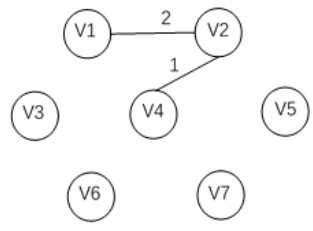


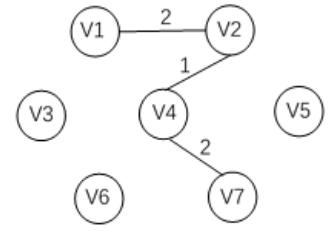
四.应用题(共 35 分)

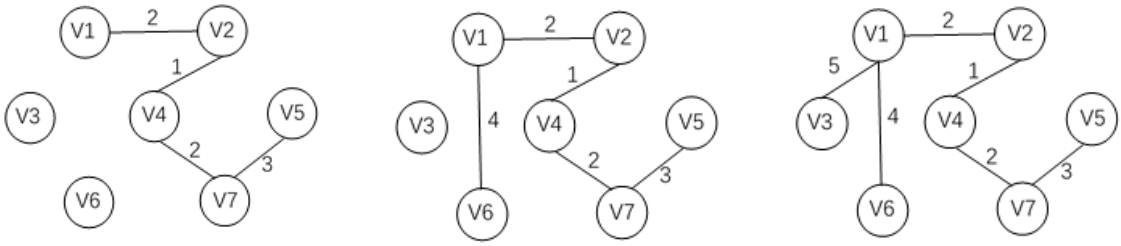
1.参考《西北工业大学 2007-2008 学年期末考试（软微 A 卷)》第九题。

2.(1)Prim 算法

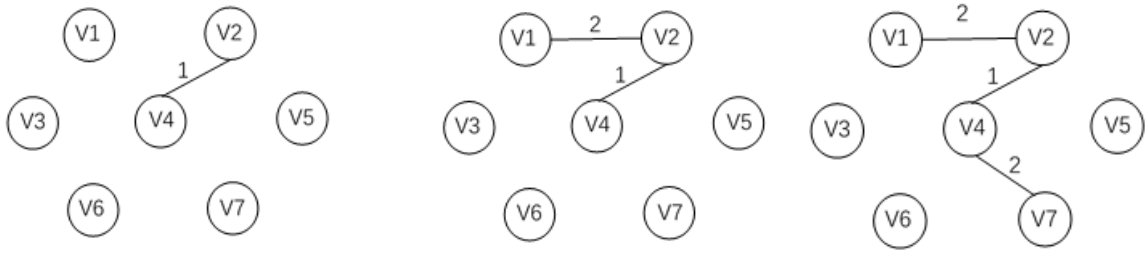


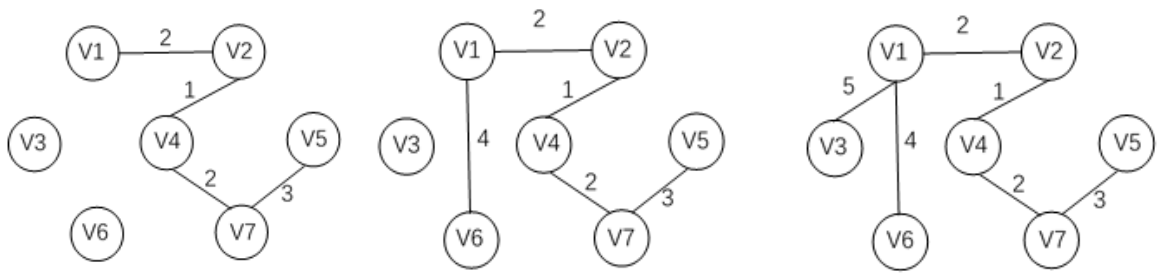






(2)Kruskal 算法





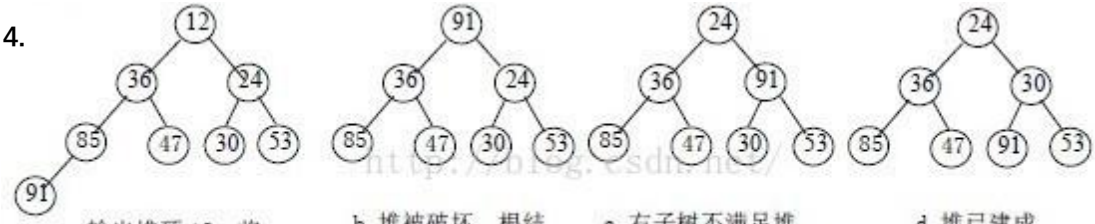
3. 构造过程如下:

H(19)=19 MOD 13=6
H(01)=01 MOD 13=1
H(23)=23 MOD 13=10
H(14)=14 MOD 13=1(冲突)
H(14)=(1+1) MOD 19=2
H(55)=55 MOD 13=3
H(20)=20 MOD 13=7
H(84)=84 MOD 13=6 (冲突)
H(84)=(6+1) MOD 19=7 (仍冲突)
H(84)=(6+2) MOD 19=8
H(27)=27 MOD 13=1 (冲突)
H(27)=(1+1) MOD 19=2 (冲突)
H(27)=(1+2) MOD 19=3 (仍冲突)
H(27)=(1+3) MOD 19=4
H(68)=68 MOD 13=3 (冲突)
H(68)=(3+1) MOD 19=4 (仍冲突)
H(68)=(3+2) MOD 19=5
H(11)=11 MOD 13=11
H(10)=10 MOD 13=10 (冲突)
H(10)=(10+1) MOD 19=11 (仍冲突)
H(10)=(10+2) MOD 19=12
H(77)=77 MOD 13=12 (冲突)
H(77)=(12+1) MOD 19=13

因此，各关键字相应的地址分配如下:

address(01)=1
address(14)=2
address(55)=3
address(27)=4
address(68)=5
address(19)=6
address(20)=7
address(84)=8
address(23)=10
address(11)=11
address(10)=12
address(77)=13
其余的地址中为空。

4.



a. 输出堆顶 12，将堆低 91 送入堆顶　b. 堆被破坏，根结点与右子女交换　c. 右子树不满足堆，其根与左子女交换　d. 堆已建成

5. 事件最早发生时间 $ve(v_i)$和最晚发生时间 $vl(v_j)$：

	α	A	B	C	D	E	F	G	H	I	J	K	ω
$ve(v_i)$	0	1	6	17	3	34	4	3	13	1	31	22	44
$vl(v_j)$	0	20	24	26	19	34	8	3	13	7	31	22	44

活动最早允许开始的时间 $e(a_i)$和最迟允许开始的时间 $l(a_j)$：

	<α, A>	<α, B>	<α, D>	<α, F>	<α, G>	<α, I>	<A, C>	<B, C>
$e(a_i)$	0	0	0	0	0	0	1	6
$l(a_j)$	19	18	16	4	0	6	20	24
	<D, C>	<D, E>	<D, J>	<F, E>	<F, H>	<G, ω>	<G, H>	<I, H>
$e(a_i)$	3	3	3	4	4	3	3	1
$l(a_j)$	19	26	25	23	8	23	3	7
	<C, E>	<H, C>	<H, J>	<H, K>	<K, J>	<J, E>	<J, ω>	<E, ω>
$e(a_i)$	17	13	13	13	22	31	31	34
$l(a_j)$	26	22	27	13	22	31	32	34

关键路径只有一条：α → G → H → K → J → E → ω

五.算法题(共 18 分)

1.(1)p->next != NULL && p->next->data != x　(2)q = p->next;　(3)p->next = q->next;

2.可用递归算法实现，直接交换左右子树。

```
void exchange(BiTNode* rt){
    BiTNode *temp = NULL;
    if(rt == NULL)
        return;
    if(rt->lchild == NULL && rt->rchild == NULL)
        return;
    else {
        temp = rt->lchild;
        rt->lchild = rt->rchild;
        rt->rchild = temp;
    }
    if(rt->lchild)
        exchange(rt->lchild);
    if(rt->rchild)
        exchange(rt->rchild);
    return;
}
```

3.直接插入排序:

```
void insertsort(int a[],int n) {
    int i,temp,j;
    for(i=1;i<n;i++){
        if(a[i]<a[i-1]) {
            temp=a[i];
            for(j=i-1;j>=0&&a[j]>temp;j--)
                a[j+1]=a[j];
            a[j+1]=temp;
        }
    }
}
```

西北工业大学 2007-2008 学年第二学期期末考试(理学院)

一.填空题(每空 1 分，共 20 分)

1.数据　　**2.**50　$2^{50}-1$　　**3.**250【解析】该树为完全二叉树, 因此 $n_1=0$ 或 $n_1=1$,又 $n_2=n_0-1$,故 $n=2*n_0-1+n_1=500$。n_1 只能为 1，故 $n_0=250$

4.线性表中元素的个数　　**5.**p->next=q->next;【解析】删除 p 结点时，把 q 结点的数据域和指针域赋给 p，删除了 q 结点

6.23541　　**7.**3　17/7　　**8.**0 1 1 1 1 2 3 1 【解析】如下表

	a	d	e	c	a	d	c	b
MaxL	0	0	0	0	1	2	0	0
next	0	1	1	1	1	2	3	1

9.b　　**10.**1784 【解析】100+（3*3*5*8+1*5*8+2*8+5）*4=1784

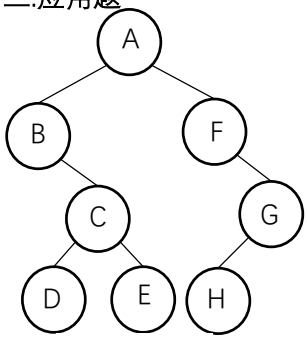
11.9　　**12.**16　　**13.**n-1　　**14.**5　　**15.**冲突　　**16.**6　4

17. O(nlogn)【解析】含 n 个记录的序列可能出现的初始状态有 n!个，则描述 n 个记录排序过程的判定树必须有 n!个叶子结点。若二叉树的高度为 h，则叶子结点的高度不超过 2^{h-1}；反之，若有 m 个叶子结点，则二叉树的高度至少为$\lceil \log_2 m \rceil + 1$。所以，n 个记录排序的判定树上必存在一条长度为$\lceil \log_2(n!) \rceil$的路径。可知任何一个借助"比较"进行排序的算法，在最坏条件下所需进行的比较次数至少为$\lceil \log_2(n!) \rceil$。根据斯特林公式有，$\lceil \log_2(n!) \rceil = O(n \log n)$。

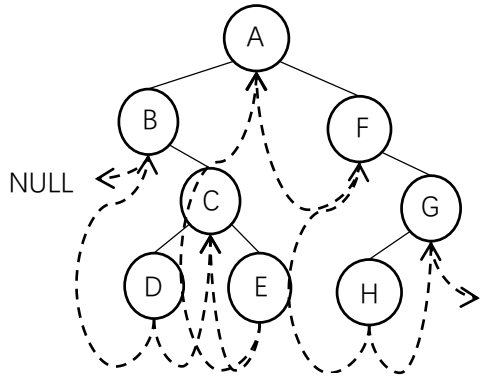
二.单项选择题(每题 1 分，共 20 分)

答案速查：CDBDA　CDCCA　CABAC　CABCC

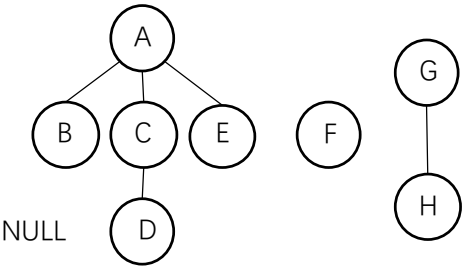
三.应用题



图三-1(1)



图三-1(1)



图三-1(3)

1.(1) 见图三-1(1)　(2) 图三-1(2)　(3) 图三-1(3)【注】假如一棵二叉树的根节点有右孩子，则这棵二叉树能够转化为森林，否则将转化为一棵树。二叉树转化为森林步骤:

①从根节点开始，若右孩子存在，则把与右孩子结点的连线删除。再查看分立后的二叉树，若其根节点的右孩子存在，则连线删除，以此类推，直到所有这些根节点与右孩子的连线都删除为止;

②把分离后的每棵二叉树转化为树;

③整理②得到的树，使之规范得到森林。

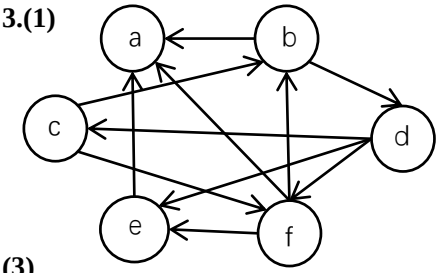
2.设该树中的叶子数为 n0 个.该树中的总结点数为 n 个,则有:

$n=n_0+n_1+n_2+\cdots+n_K$ (1)

$n-1=0*n_0+1*n_1+2*n_2+\cdots+K*n_K$ (2)

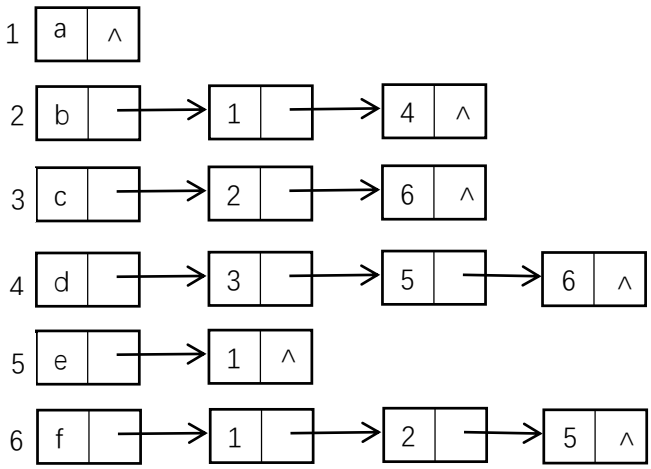
联立(1)(2)方程组可得：叶子数为：$n_0=1+0*n_1+1*n_2+2*n_3+...+(K-1)*n_K$

3.(1)

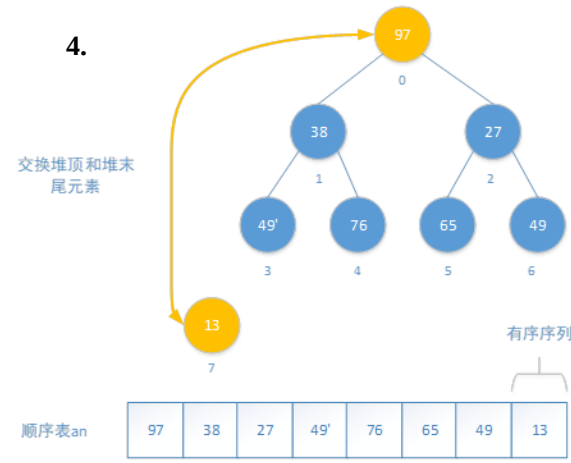


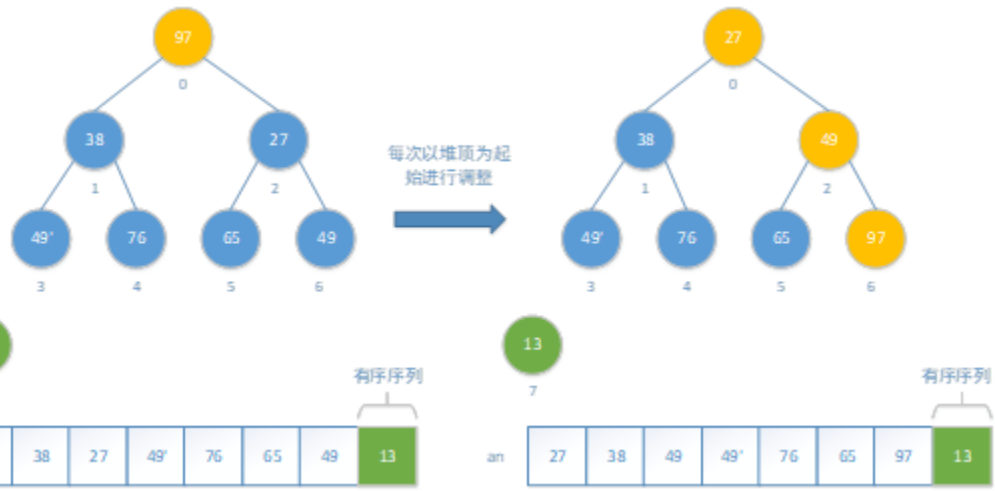
(2)	a	b	c	d	e	f
入度	3	2	1	1	2	2
出度	0	2	2	3	1	3

(3)



4.





5.方案 1：

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

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
6.参考《西北工业大学 2005-2006 学年第二学期期末考试(理学院)》四应用题 5 小题。　**7.**略。

四.算法设计题

1.p=L->next
p!=NULL
L->next=q

2.
```
int BinaryTreeHeight(Node *node){
    int treeHeight = 0;
    if (node != NULL) {
        int leftHeight = BinaryTreeHeight(node->lChild);
        int rightHeight = BinaryTreeHeight(node->rChild);
        treeHeight = leftHeight >= rightHeight? leftHeight + 1:rightHeight + 1;
    }
    return treeHeight;
}
```

3.简单选择排序算法：
```
void SelectSort_Up(int *arr, int length){
    for (int i = 0; i < length-1; i++){
        int index = i;//记录最值下标
        for (int j = i + 1; j < length; j++){
            //升序
            if (arr[index] > arr[j]){
                index = j;
            }
        }
        if (i != index){
            Swap(&arr[i], &arr[index]);
        }
    }
}
```

西北工业大学 2009-2010 学年第一学期期末考试(自动化)

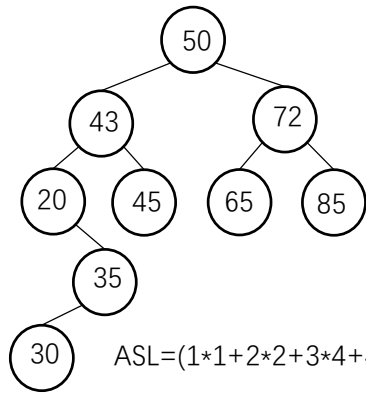
一.单项选择题(本题共 15×2 分=30 分)

答案速查：BBDBD　BDDBC　ABCBD

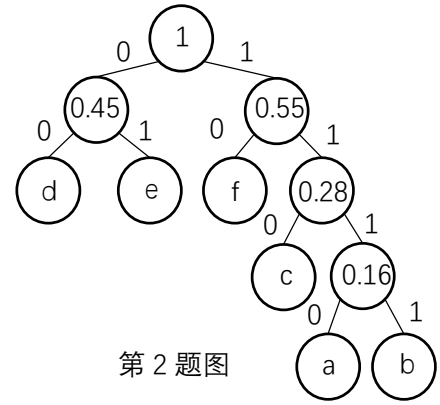
二.解答题(本题共 6×7 分=42 分)

1.char【解析】1.h 2.hr 3.hrc 4.rc 5.rch 6.ch (x='r') 7.cha 8.char
2.如图所示。a:1110　b:1111　c:110　d:00　e:01　f:10
3.ABCDEF　ABCDFE　ACBDEF　ACBDFE　ACBEFD　ACBEDF　ACBFED　ACBFDE
4.A->B：（A.B）　11；A->C：（A.C）　7；A->D：（A.C.D）　12
A->E：（A.C.E）　14；A->F：（A.C.F）　15

5.ASL=(N+1)/2=5

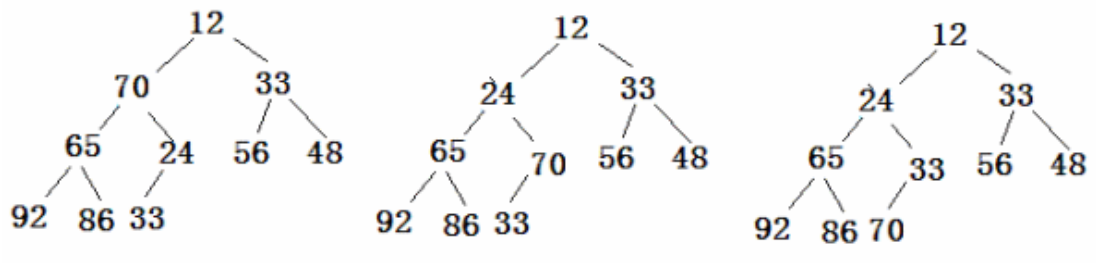


ASL=(1*1+2*2+3*4+4*1+5)/9=26/9



第 2 题图

6.70 所在的子树不是堆，调整过程如下:



三.算法设计题(本题共 28 分)

1.(1)

```
//定义存放多项式的数组类型
typedef struct{
    float   coef;
    int     exp;
}PolyArray[MAX];
//定义单链表结点类型
typedef struct   pnode{
    float coef;
    int exp;
    struct   pnode *next;
}PolyNode;
//一元多项式的表示
//建立多项式链表）
void      createListR(PolyNode   *L,
PolyArray a, int n){
    PolyNode *s, *r;
    int i;
    L = (PolyNode *)malloc(sizeof(PolyNode));
    L->next = NULL;
    r = L;
    for (i = 0; i<n; i++){
        s = (PolyNode*)malloc(sizeof(PolyNode));
        s->coef = a[i].coef;
        s->exp = a[i].exp;
        s->next = NULL;
        r->next = s;
        r = s;
    }
    r->next = NULL;
}
```

(2)

```
void DerivList(LinkList &L1,LinkList &L2)
{ //求导
    LNode *p;
    p = L1 -> next;
    LNode *cur,*rear;
    cur = L2;
    rear = new LNode;
        rear = new LNode;
        rear -> coef = (p->coef) *   (p->expon);
        rear -> expon = p->expon -1;
        rear -> next = NULL;
        cur -> next = rear;
        cur = rear;
    }
```

```
        while(p) {                          |          p = p->next;
                                            |      }
      if(p->coef!=0&&p->expon!=0)   {       | }
```

2.(1)
```
int treedepth_recursion5(Tnode *root) { //递归形式
    if (root== NULL)
        return 0;
    else
        return treedepth_recursion5(root->lchild)>treedepth_recursion5(root->rchild) ?
treedepth_recursion5(root->lchild) + 1 : treedepth_recursion5(root->rchild) + 1;
}
```

(2)
```
int leafcount(BinTree T) { //求二叉树 T 的叶子数
    if(T==NULL) leaf=0; //当二叉树为空时，叶子数等于 0
    else if((T- > lchild==NULL)&&(T- > rchild==NULL)) leaf=A; //当二叉树仅含一个根结点时，叶子数为 A
    else{
      L==leafcount(T- > lchild); //求左子树的叶子数
      R=leafcount(T- > rchild); //求右子树的叶子数
      leaf=L+R;  //左、右子树叶子数之和等于二叉树的叶子数
     }
    return(leaf);
}
```

西北工业大学 2010-2011 学年期末考试(自动化)

1.-100011201234

2.AB+D*EFAD*+/C++

3.(rear+1)%MaxSize=front rear=(rear+1)%MaxSize

4.$n_0=n_2+1$,$n=n_0+n_2=2n_0-1$

5.(2)若有一节点经多次步骤后总有前驱，则为环；(4)关键路径算法的实现是基于拓扑排序的

6.折半查找，$2^n-1=10000$，$n=[\log_2 10001]=13$

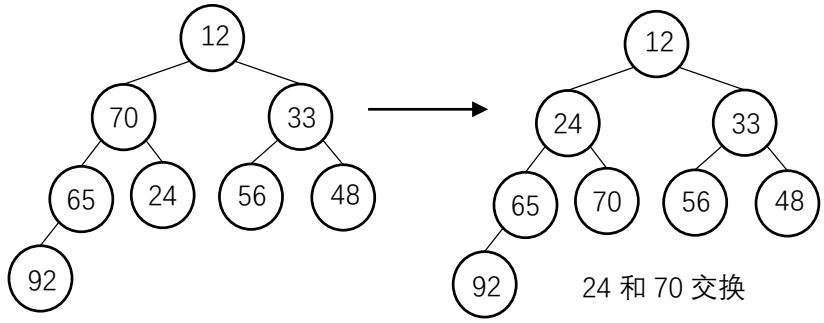
7.

下标	0	1	2	3	4	5	6	7	8	9	10	11	12
关键码		14	01	68	27	55	19	20	84		23	111	
		1	2	1	4	3	1	1	3		1	1	

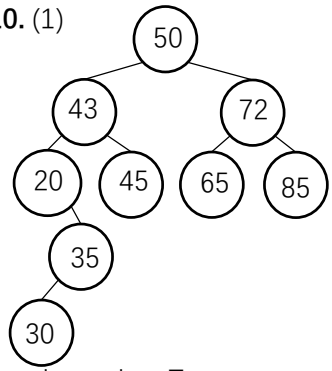
ASLsucc=$\frac{1}{10}=(1+2+1+4+3+1+1+3+1+1)$=1.8

8.n+(n-1)+(n-2)+…+(n-i+1)+(j-i)=(2n-i-1)*i/2+j

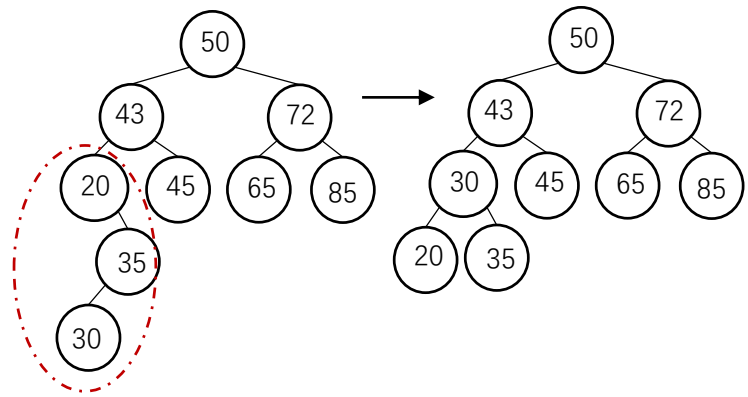
9.不是。



10. (1)



(2)5 次　　平衡二叉排序树　　3 次



(3)template<class T>
struct binaryTreeNode {
T element;
binaryTreeNode<T>* leftChild, *rightChild;//左子树和右子树。
binaryTreeNode() {leftChild=rightChild=NULL;}
binaryTreeNode(const T& theElement) {element(theElement);leftChild=rightChild=NULL;}
binaryTreeNode(const T& theElement,binaryTreeNode<T> *theLeftChild,binaryTreeNode<T> *theRightChild) {
element(theElement);
leftChild=theLeftChild;
rightChild=theRightChild;
}
}

(4)递归算法:

```
public class BinaryTree {
    // 访问节点
    public void printNode(Node node) {
        System.out.println(node.getData());
    }
    // 先序遍历
    public void theFirstTraversal(Node root) {
        printNode(root);
        // 使用递归遍历左孩子
        if (root.getLeftNode() != null) {
            theFirstTraversal(root.getLeftNode());
        }
        // 使用递归遍历右孩子
        if (root.getRightNode() != null) {
            theFirstTraversal(root.getRightNode());
        }
    }
    // 中序遍历
    public void theInOrderTraversal(Node root) {
        // 递归遍历左孩子
        if (root.getLeftNode() != null) {
```

```
    // 后序遍历
    public void thePostOrderTraveral(Node root) {
        // 递归遍历左孩子
        if (root.getLeftNode() != null) {
            theInOrderTraversal(root.getLeftNode());
        }
        // 遍历右孩子
        if (root.getRightNode() != null) {
            theInOrderTraversal(root.getRightNode());
        }
        printNode(root);
    }
    public static void main(String[] args) {
        BinaryTree t = new BinaryTree();
        Node node = t.init();
        System.out.println("先序遍历:");
        t.theFirstTraversal(node);
        System.out.println("中序遍历:");
        t.theInOrderTraversal(node);
        System.out.println("后序遍历:");
        t.thePostOrderTraveral(node);
```

```
            theInOrderTraversal(root.getLeftNode());
        }
        printNode(root);
        // 遍历右孩子
        if (root.getRightNode() != null) {
          theInOrderTraversal(root.getRightNode());
        }
    }
```

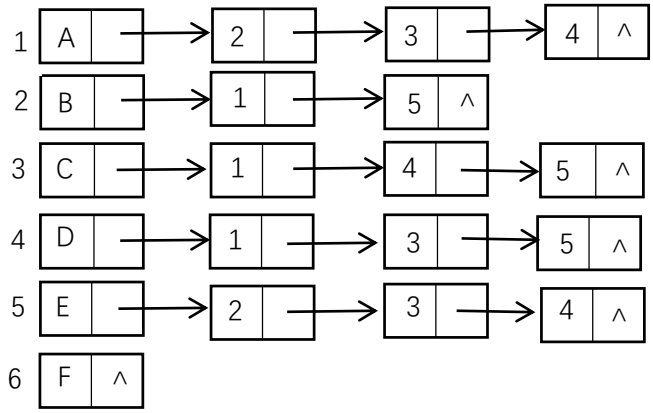
```
    }
}
```

非递归算法：

```
public class BinaryTree1 {
    public void printNode(Node node) {
        System.out.print(node.getData());
    }
    public void theFirstTraversal_Stack(Node root) { // 先序遍历
    Stack<Node> stack = new Stack<Node>();
        Node node = root;
        while (node != null || stack.size() > 0) { // 将所有左孩子压栈
            if (node != null) { // 压栈之前先访问
                printNode(node);
                stack.push(node);
                node = node.getLeftNode();
            } else {
                node = stack.pop();
                node = node.getRightNode();
            }
        }
    }
    public void theInOrderTraversal_Stack(Node root) { // 中序遍历
        Stack<Node> stack = new Stack<Node>();
        Node node = root;
        while (node != null || stack.size() > 0) {
            if (node != null) {
                stack.push(node); // 直接压栈
                node = node.getLeftNode();
            } else {
                node = stack.pop(); // 出栈并访问
                printNode(node);
                node = node.getRightNode();
            }
        }
    }
    public void thePostOrderTraversal_Stack(Node root) { // 后序遍历
        Stack<Node> stack = new Stack<Node>();
        Stack<Node> output = new Stack<Node>();// 构造一个中间栈来存储逆后序遍历的结果
        Node node = root;
        while (node != null || stack.size() > 0) {
            if (node != null) {
                output.push(node);
                stack.push(node);
                node = node.getRightNode();
            } else {
                node = stack.pop();
                node = node.getLeftNode();
            }
        }
        System.out.println(output.size());
        while (output.size() > 0) {
            printNode(output.pop());
        }
    }
    public static void main(String[] args) {
        BinaryTree1 tree = new BinaryTree1();
        Node root = tree.init();
        System.out.println("先序遍历");
        tree.theFirstTraversal_Stack(root);
        System.out.println("");
        System.out.println("中序遍历");
        tree.theInOrderTraversal_Stack(root);
        System.out.println("");
        System.out.println("后序遍历");
        tree.thePostOrderTraversal_Stack(root);
    }
}
```

11.(1)

邻接表



(2)CDEBA F 未连，所以是不连通的

(3)
```
#define    MaxVertices 100
typedef struct node{ //边表
    int adjvex;
    node* next;
}EdgeNode;
typedef struct{    //顶点表
    int vertex;
    EdgeNode* edgenext;
}VertexNode;
typedef VertexNode AdjList[MaxVertices];//顶点表数组
typedef struct{
     AdjList adjlist;
     int n,e;
}AdjMatrix;
```

(4)
```
p = v[i] -> firstedge;
pre = p;
while (p && p -> data != j) {
   pre = p;p = p -> next;
}
if (p && pre == p) v[i] -> firstedge = p -> next;
else if (p) pre -> next = p -> next;
```

(5)O(n+e)

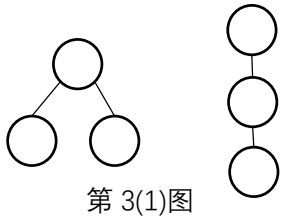
西北工业大学 2013-2014 学年期末考试(自动化)

一.单项选择题(本题共 15×2 分=30 分)

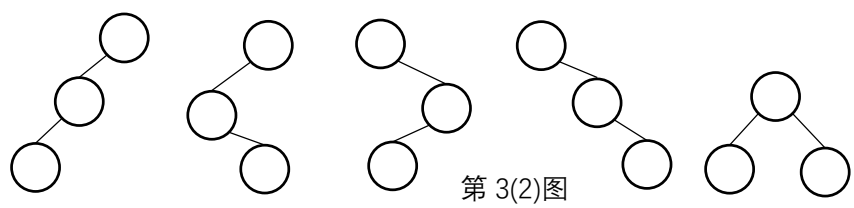
答案速查：CBACC ACAAD BBADB

二.解答题(本题共 5X7 分= 35 分)

1.done **2.**略 **3.(1)**具有三个结点的树 **(2)**具有三个结点的二叉树

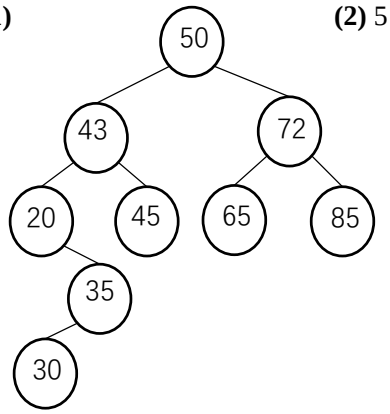


第 3(1)图



第 3(2)图

4.(1)



(2) 5 次

5. 初始：14，4，18，32，10，30，6，12

12，4，18，32，10，30，6，14

12，4，14，32，10，30，6，18

12，4，6，32，10，30，14，18

12，4，6，14，10，30，32，18

12，4，6，10，14，30，32，18

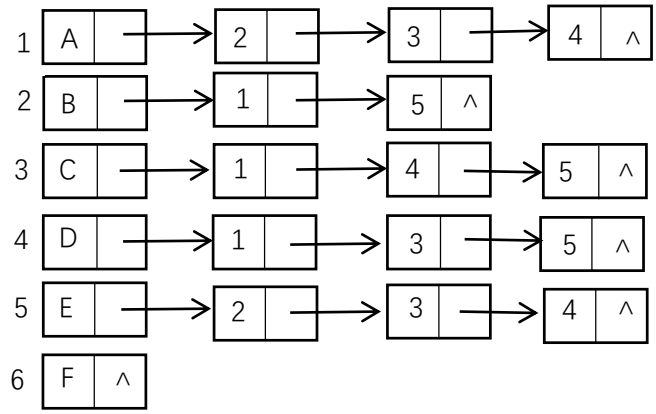
三.算法设计题(本题共 10 分+25 分=35 分)

1.采用头插法实现单链表的逆置

```
void reverse(LNode* head){
    LNode* p = head->next;
    head->next = NULL;
    while(p) {
        q = p;
        p = p->next;
        q->next = head->next;
        head->next = q;
    }
}
```

2.(1)CADEB

(2)



(3)用深度或广度优先搜索的办法判断只有一个连通分量即可，这里以邻接表作为存储结构的深度优先搜索为例。

```
void DFS(ALGraph & graph, int v){
    visited[v] = true;
    ArcNode * p = graph.adjList[v].firstarc;
    while (p){
        if (!visited[p->adjvex])
            DFS(graph, p->adjvex);
        p = p->next;
    }
}
```

```
void DFSTraverse(ALGraph & graph){
    for (int i = 0; i < graph.vexNum; i++)
        visited[i] = false;
    DFS(graph, 0);
    for (int i = 0; i < graph.vexNum; i++){
        if (!visited[i])
            return false;
    }
    return true;
}
```

（4）O(n+e)

西北工业大学 2014-2015 学年期末考试试题(自动化)

1.(1)HBGEACF (2)3 满二叉树 (3)2i 2i+1

2.(1)见右图 (2)平均查找长度=(1*1+2*2+3*3+3*4+2*5+1*6)/12=3.5

(3)平均查找长度=(1+12)/2=6.5

3.参考 ~~2007 年真题第三题第 1 小题~~

4.(1)直接插入排序算法

第一趟：9，75，41，14，26，33，36

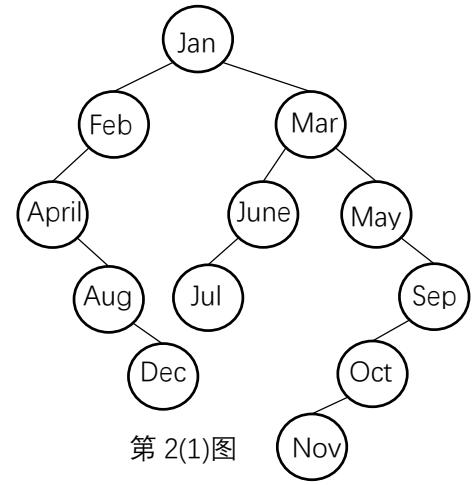
第二趟：9，41，75，14，26，33，36

第三趟：9，14，41，75，26，33，36

第四趟：9，14，26，41，75，33，36

第五趟：9，14，26，33，41，75，36

第六趟：9，14，26，33，36，41，75



第 2(1)图

(2)略，多次出现。(3)略，多次出现。

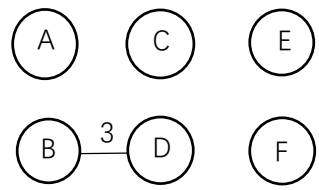
(4)与初始序列和所选择的基准元素有关系。当初始序列已经基本有序时，快速排序算法退化为冒泡排序；另外，所选择的基准元素最好使得划分的结果是基准的左、右两个无序子区间的长度大致相等。

5.(1)略 (2)构造最优三叉树：令 m=元素个数,k(k 叉树）=3。增加的虚结点数：n=k-(m-1)%(k-1)-1,即增加 n 个权为 0 的结点。然后按照哈夫曼树构造方法进行构造，所不同的是每次选取三个权值最小的结点。

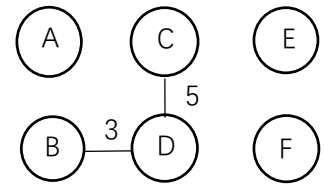
6.(1)邻接矩阵

$$\begin{bmatrix} \infty & 11 & 7 & \infty & \infty & \infty \\ 11 & \infty & 10 & 3 & \infty & \infty \\ 7 & 10 & \infty & 5 & 7 & 8 \\ \infty & 3 & 5 & \infty & \infty & \infty \\ \infty & \infty & 7 & \infty & \infty & 6 \\ \infty & \infty & 8 & \infty & 6 & \infty \end{bmatrix}$$

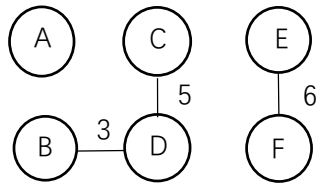
(2)Kruskal 算法：



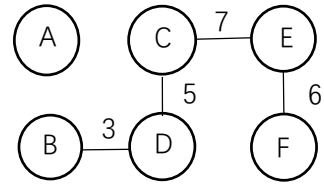
(1)



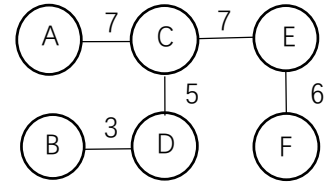
(2)



(3)



(4)



(5)

(3)采用广度优先遍历算法：

```
void BFSTraverse1(ALGraph G,void(* Visit)(char *)){
    int v,u;
    ArcNode * p; //p 指向表结点
    LinkQueue Q; //链队列类型
    for (v=0; v<G.vexnum; ++v)
        visited[v] = FALSE;
    InitQueue(&Q);
    for (v=0; v<G.vexnum; ++v) {
        if (! visited[v]) {    //v 尚未被访问
            visited[v] = true; //设 v 为已被访问
            //Visit(G.vertices[v].data); //访问 v
            EnQueue(&Q,v); //v 入队
            while (! QueueEmpty(Q)) { //队列不空
```

```
                DeQueue(&Q,&u); //队头元素出队并置为 u
                //p 依次指向 u 的邻接顶点
                for (p=G.vertices[u].firstarc; p; p=p->next){
                    //u 的邻接顶点尚未被访问
                    if (! visited[p->data.adjvex]){
                        visited[p->data.adjvex] = true;//该邻接顶点设为已被访问
                        //Visit(G.vertices[p ->data.adjvex].data);//访问该邻接顶点
                        cout<<"("<<u<<","<<p->data.adjvex<<")";
                        EnQueue(&Q,p->data.adjvex); //入队该邻接顶点序号
                    }
                }
            }
            cout<<endl;
        }
    }
}
```

【注】答案只是给了广度优先遍历的标准程序，题目是求连通分量，要把遍历中相应的访问函数改成输出才能符合要求，删除了两个 visit 语句，并新增 cout<<"("<<u<<","<<p->data.adjvex<<")";语句，这样每一个连通分量都以罗列边的形式输出。

【注】该算法输出的连通分量不完整，先深搜或广搜，然后把访问数组中为 true 的点的边表依次遍历，遍历边表加上一个判断条件，若已输出边的点则跳过。也就是两个顶点数组，一个用来标记它本身已经入队，一个用来标记它的所有邻边已经入队