1 CHP1. 绪论

1. 下列程序段的时间复杂度是（　　）。

```
count=0;
for(j=1;j<=n;j++)
    for(k=1;k<=n;k*=4)
        count++;
```

A．O($\log_4 n$)　　B．O(n)　　C．O(n$\log_4 n$)　　D．O(n^2)

解：设内层循环执行 m 次，则有$4^m \le n$，因此$m \le \log_4 n$；外层循环执行 n 次，所以两层循环执行次数$\le n\log_4 n$，大 O 记法写为$O(n\log_4 n)$

2. 下列程序段的时间复杂度是（　　）。

```
k=0;
for(i=1;i<=n;i++)
    for(j=i;j<=n;j++)
        k++;
```

A．O($\log_4 n$)　　B．O(n)　　C．O(n$\log_4 n$)　　D．O(n^2)

解：
最深层循环执行次数为:

$$\sum_{i=1}^{n}\sum_{j=i}^{n}1=\sum_{i=1}^{n}(n-i+1)=n(n-1)-\sum_{i=1}^{n}i=n(n-1)-\frac{(n+1)n}{2}=\frac{(n-3)n}{2}$$

因此时间复杂度应为 O(n^2)

3. 下列函数的增长率由小至大的排列应为什么？

$\log_2(\log_2 n)$，2^{100}，$n^{\log_2 n}$，$(2/3)^n$，$n^{2/3}$，$(3/2)^n$，$\log_2 n$，$n/\log_2 n$，$\log\frac{2}{2}n$，$n^{1/2}$，n，$n\log_2 n$，$n^{3/2}$，$(4/3)^n$，$n!$，n^n

解：
常数阶:2^{100}

多数、多项式阶：$(2/3)^n$，$\log_2(\log_2 n)$，$\log_2 n$，$\log\frac{2}{2}n$，$n^{1/2}$，$n^{2/3}$，$n/\log_2 n$，n，$n\log_2 n$，$n^{3/2}$

指数阶：$(4/3)^n$，$(3/2)^n$，$n^{\log_2 n}$，$n!$，n^n

应为：2^{100}，$(2/3)^n$，$\log_2(\log_2 n)$，$\log_2 n$，$\log\frac{2}{2}n$，$n^{1/2}$，$n^{2/3}$，$n/\log_2 n$，n，$n\log_2 n$，$n^{3/2}$，$(4/3)^n$，$(3/2)^n$，$n^{\log_2 n}$，$n!$，n^n

4. 编写算法求一元多项式$P_n(x)=\sum_{i=0}^{n}a_ix^i$的值$P_n(x_0)$，并确定算法中每一语句的执行次数和整个算法的时间复杂度。本题的输入为$a_i(i=0,1,\cdots,n),x_0$和n，输出为$P_n(x_0)$。

解：

```
3   double E01(double *a, int n, double x)
4   {//a={a0,a1,a2,...,an}; n为阶数，最高次幂;
5       double sum = a[0], item = x;//初始化为常数项
6
7       for (int i = 1; i <= n; i++)
8       {
9           sum += a[i] * item;
10          item *= x;
11      }
12      return sum;
13  }
```

第 5、12 行，各执行一次；第 7 行，比较 n+1 次，第 9、10 行，各执行 n 次。算法时间复杂度为 O(n)

2 CHP2. 线性表

5. 填空

(1) 在顺序表中插入或删除一个元素，平均约需要移动_____元素，具体移动的元素个数与_____有关。

(2) 顺序表中逻辑上相邻的元素的物理位置_____紧邻。单链表中逻辑上相邻的元素的物理位置_____紧邻。

(3) 在单链表中，除了首元结点外，任意结点的存储位置由_____指示。

(4) 在单链表中设置头结点的作用是_____

解：

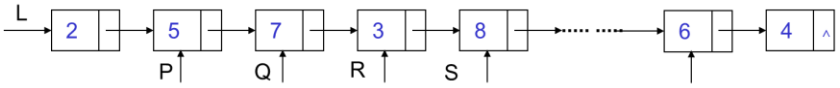
a) 在顺序表中插入或删除一个元素，平均约需要移动 表中一半 元素，具体移动的元素个数与 表长和该元素在表中的位置 有关。

b) 顺序表中逻辑上相邻的元素的物理位置 必然 紧邻。单链表中逻辑上相邻的元素的物理位置 未必 紧邻。

c) 在单链表中，除了首元结点外，任意结点的存储位置由 其直接前驱的指针域 指示。

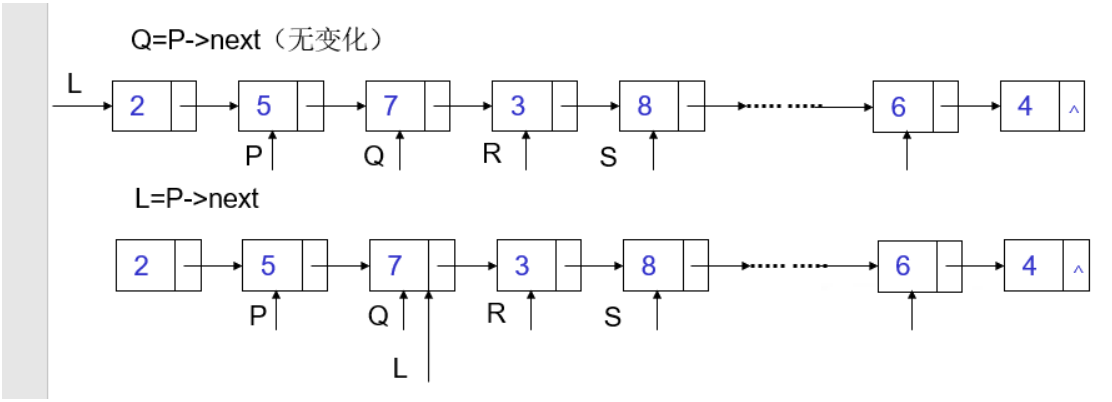
d) 在单链表中设置头结点的作用是 在表头进行插入或删除的操作与其他位置的操作相同（插入或删除首元素时不必进行特殊处理）

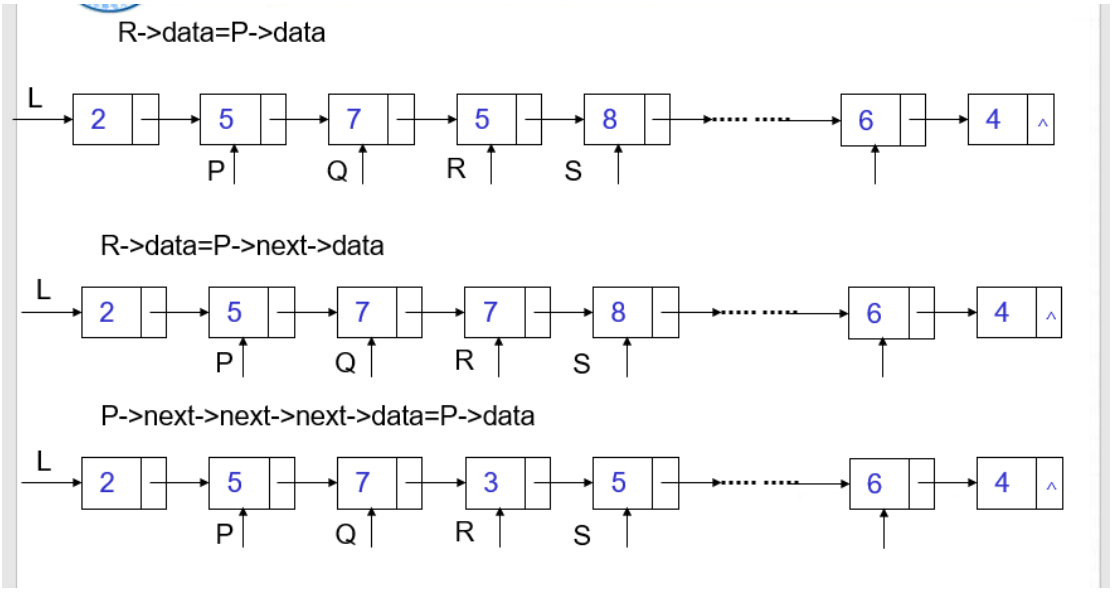
6. 对以下单链表分别执行下列各程序段，并画出结果示意图。

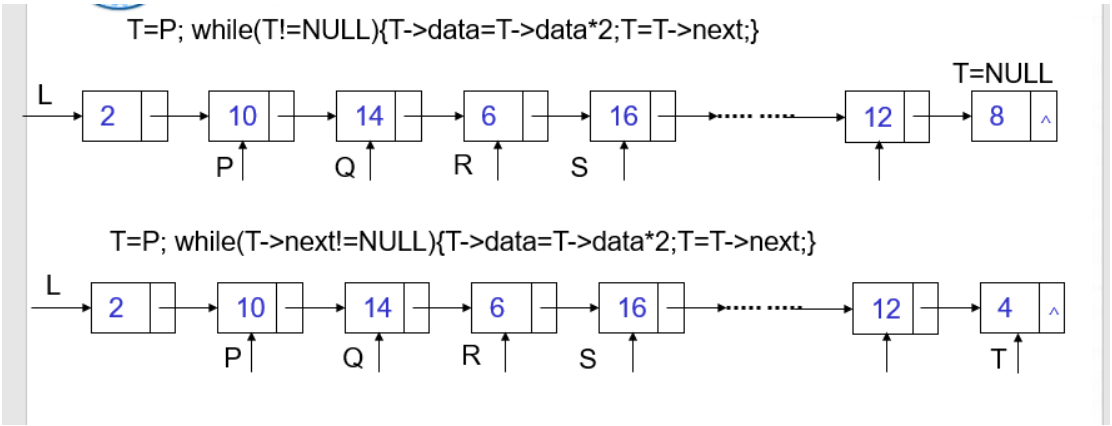


(1) Q=P->next
(2) L=P->next
(3) R->data=P->data
(4) R->data=P->next->data
(5) P->next->next->next->data=P->data
(6) T=P; while(T!=NULL){T->data=T->data*2;T=T->next;}
(7) T=P; while(T->next!=NULL){T->data=T->data*2;T=T->next;}

解：







7. 已知 L 是带表头结点的非空链表，且 P 结点既不是首元素结点，也不是尾元素结点，试从下列提供的答案中算则合适的语句序列。

(1) 删除 P 结点的直接后继结点的语句序列是_____

(2) 删除 P 结点的直接前驱结点的语句序列是_____

(3) 删除 P 结点的语句序列是_____

(4) 删除首元结点的语句序列是_____

(5) 删除尾元结点的语句序列是_____

语句有：

(1) P=P->next;

(2) P->next=P;

(3) P->next=P->next->next;

(4) P=P->next->next;

(5) while(P!=NULL) P=P->next;

(6) while(Q->next!=NULL) {P=Q; Q=Q->next;}

(7) while(P->next!=Q) P=P->next;

(8) while(P->next->next!=Q)P=P->next;

(9) while(P->next->next!=NULL) P=P->next;

(10) Q=P;

(11) Q=P->next;

(12) P=L;

(13) L=L->next;

(14) free(Q);

解：

a) 删除 P 结点的直接后继结点的语句序列是__(11)(3)(14)__

b) 删除 P 结点的直接前驱结点的语句序列是__(10)(12)(8)(11)(3)(14)__

c) 删除 P 结点的语句序列是__(10)(12)(7)(3)(14)__

d) 删除首元结点的语句序列是__(12) (11)(3)(14)__

e) 删除尾元结点的语句序列是__(9)(11)(3)(14)__

8. 试写一算法，对单链表实现就地逆置。

解：对于线性表 a0,a1,a2,…,an 而言，为其构造链表时，若每次将新结点追加到表尾，那么从表头到表尾的结点对应线性表元素 a0,a1,a2,…,an；若每次将新结点从表头插入，则那么从表头到表尾的结点与线性表元素顺序逆序，即依次对应线性表元素 an, ,…,a2,a1,a0。就地逆置利用这个特点，将结点顺序从链表中摘出，再从表头插入。

```
typedef struct node
{
    int data;
    struct node *next;
}NODE, *PNODE, *LinkList;
void Invert(LinkList head) {//逆置链表,链表已经存在，head指向其头结点。
    NODE *p = NULL,
        *q = head->next;//head指向头结点，q指向a1
```

```
    head->next = NULL;//头结点从原链表中断开
    while (q)
    {
        p = q;//摘出结点ai
        q = q->next;
        p->next = head->next;
        head->next = p;//将ai插入表头，形成逆序
    }
}
```

9. 请分析在有序单链表中删除一个结点并保持有序的时间复杂度。如果单链表无序呢？

解：O(n)

主要操作为比较待删除元素和当前结点元素值，删除第 i 个结点需要比较 i 次，因此最少比较 1 次，最多比较 n 次，平均比较(n+1)/2（基于等概率假设：各个结点被删的概率相等）。

单链表无序时，删除任一结点之前都需要遍历整个链表，遍历 n 个结点的时间复杂度为 O(n)

10. “访问线性表的第 i 个元素的时间必然同 i 有关”，这个说法正确吗？为什么？

解：

不正确。若线性表采用顺序存储，则访问元素的时间复杂度为 O(1)，与 i 无关。若采用链式存储，则指针移动的次数与元素位置 i 有关。

11. 请分析循环单链表表示的链队列中，是否可以不设置队头指针？

解：

可以。循环链表中，通过指针域，整个链表形成一个环，从任意结点出发都能找到表中其他结点。所以，循环单链表中，保留指向表头的指针就可以了。

队列采用循环单链表表示时，即使不设队头指针，也可正常完成插入、删除、访问等操作。

12. 比较线性表顺序存储和链式存储的特色。对比这两种存储结构下插入操作的时间复杂度。

解：

（1）顺序存储

优点：存储密度大，存储空间利用率高。随机存取，便于查询。

缺点：插入或删除元素时不方便。

（2）链式存储

优点：插入或删除元素时很方便，使用灵活。

缺点：存储密度小，存储空间利用率低。不能随机存取，不便于查询。

（3）比较

顺序表适宜于做查找这样的静态操作

链表宜于做插入、删除这样的动态操作

13. 已知线性表中的元素是无序的，且以带头结点的单链表 L 作为存储结构。设

计一个删除表中所有值小于 a（a 为给定数值）的元素的算法，并给出算法中关键步骤的注释说明。

解：

```
typedef struct Node{
   ElemType data;
   struct Node *next;
}Node, *PNode;

typedef struct LinkList{
   int count;//总数
   PNode head;
}LinkList;

void DelElem(PNode head, ElemType a){
   PNode q = head, p = head->next;

   while(p ! = NULL){
      if(p->data < a){
         q->next = p->next;
         free(p);
         p=q->next;
      }else{
         q = p; p = p->next;
      }
   }
}
```

14.

3 CHP3. 栈和队列

15. 若按教科书 3.1.1 节中图 3.1(b)所示铁道进行车厢调度（注意：两侧铁道均为单向行驶道），则请回答：
    (1) 如果进站的车厢序列为 123，则可能得到的出站车厢序列是什么？
    (2) 如果进站的车厢序列为 123456，则能否得到 435612 和 135426 的出站序列，并请为什么不能得到或者如何得到？（即写出以‘S’表示进栈和‘X’表示出栈的栈操作序列）。

解：

(1) 123,132,213,231,321,

123 对应的操作序列为：SXSXSX；132 对应的操作序列为：SXSSXX；213 对应的操作序列为：SSXXSX；231 对应的操作序列为：SSXSXX；321 对应的操作序列为：SSSXXX；

(2) 不能得到 435612，因为，首先出站的车厢为 4 意味着 123 号车厢已经入栈，则这两节车厢出站的相对顺序一定是 3,2,1，而该序列中，1 号车厢在 2 号车厢之前出

站，这不符合操作规则。

能得到 135426，操作序列为：SXSSXSSXXXSX

16. 写出下列程序段的输出结果（栈的元素类型 SElemType 为 char）

```
void main(){
  Stack S;
  char   x, y;

  InitStack(S);
  x='c'; y='k';
  Push(S, x); Push(S, 'a'); Push(S, y);
  Pop(S, x); Push(S, 't'); Push(S, x);
  Pop(S, x); Push(S, 's');
  while(!StackEmpty(S)){Pop(S, y); printf(y);}
  printf(x);
}
```

解：

根据操作序列，栈 S 为：

(1) InitStack(S): S()

(2) Push(S, x); Push(S, 'a'); Push(S, y)：S(‘c’,’a’,’k’)

(3) Pop(S, x); Push(S, 't'); Push(S, x)：x=’k’, S(‘c’,’a’,’t’,’k’)

(4) Pop(S, x); Push(S, 's'): x=’k’, S(‘c’,’a’,’t’,’s’)

(5) while(!StackEmpty(S)){Pop(S, y); printf(y);} printf(x);输出：stack

17. 按照四则运算加减乘除和幂运算（^）优先关系的惯例，并仿照教科书 3.2 节例 3-2 的格式，画出对下列算数表达式求值时操作数栈和运算符栈的变化过程。

A-B×C/D+E^F

序号	OPTR	OPND	当前字符	备注
1	#		A-B×C/D+E^F	push(OPND,’A’)
2	#	A	A-B×C/D+E^F	push(OPTR,’-’)
3	#-	A	A-B×C/D+E^F	push(OPND,’B’)
4	#-	AB	A-B×C/D+E^F	push(OPTR,’×’)
5	#-×	AB	A-B×C/D+E^F	push(OPND,’C’)

6	#-×	ABC	A-B×C/D+E^F	pop(ORND,right)//C
				pop(OPND,left)//B
				pop(OPTR,optr)//×
				T1=exe(left,optr,right)
				//T1=B×C
				push(OPND,T1)
				push(OPTR,’/’)

7	#-/	AT1	A-B×C/D+E^F	push(OPND,’D’)

8	#-/	AT1D	A-B×C/D+E^F	pop(ORND,right)//D
				pop(OPND,left)//T1
				pop(OPTR,optr)//除/
				T2=exe(left,optr,right)
				//T2=T1/D

				push(OPND,T2)
				pop(ORND,right)//T2
				pop(OPND,left)//A
				pop(OPTR,optr)//-
				T3=exe(left,optr,right)
				//T3=A-T2
				push(OPND,T3)
				push(OPTR,'+')

9	#+	T3	A-B×C/D+E^F	push(OPND,'E')
10	#+	T3E	A-B×C/D+E^F	push(OPTR,'^')
11	#+^	T3E	A-B×C/D+E^F	push(OPND,'F')

11	#+^	T3EF	A-B×C/D+E^F	pop(ORND,right)//F
				pop(OPND,left)//E
				pop(OPTR,optr)//^
				T4=exe(left,optr,right)
				//T4=E^F
				push(OPND,T4)
				pop(ORND,right)//T4
				pop(OPND,left)//T3
				pop(OPTR,optr)//+
				T5=exe(left,optr,right)
				//T5=T3+T4

18. 试推导求解 n 阶汉诺塔问题至少要执行的 move 操作次数。

解：设a_i表示 i 阶汉诺塔问题至少要执行的 move 操作次数，则可知：$a_i = 2a_{i-1} + 1$，且$a_1 = 1$，可推导：

$$a_1 = 1$$
$$a_2 = 2a_1 + 1 = 2 + 1$$
$$a_3 = 2a_2 + 1 = 2^2 + 2^1 + 2^0$$
$$\dots$$
$$a_i = \sum_{k=0}^{i-1} 2^k = \frac{2^i - 1}{2 - 1} = 2^i - 1$$

所以，n 阶汉诺塔问题至少要执行的 move 操作次数为$2^n - 1$

19. 试将下列递推过程改写为递归过程。

```
void ditui(int n){
   int i;

   i=n;
   while(i>=1)
      printf("%d\n", i--);
}
```

解：

```
void digui(int n) {
    if (n >= 1) {
```

```
        printf("%d\n", n);
        digui(n - 1);
    }
}
```

20. 写出以下程序段的输出结果（队列中的元素类型 QElemType 为 char）

```
void main(){
   Queue Q; Init Queue(Q);
   char x='e',y='c';
   EnQueue(Q,'h'); EnQueue(Q,'r');EnQueue(Q,y);
   DeQueue(Q,x);EnQueue(Q,x);
   DeQueue(Q,x);EnQueue(Q,'a');
   while(!QueueEmpty(Q)){
      DeQueue(Q,y);
      printf(y);
   }
   printf(x);
}
```

解：

EnQueue(Q,'h'); EnQueue(Q,'r');EnQueue(Q,y)执行后：Q(hrc)

DeQueue(Q,x);EnQueue(Q,x); 执行后：x='h',Q(rch)

DeQueue(Q,x);EnQueue(Q,'a'); 执行后：x='r',Q(cha)

while(!QueueEmpty(Q)){ DeQueue(Q,y); printf(y); }执行后：cha

printf(x); 执行后：char

21. 简述以下算法的功能（栈和队列的元素类型为 int）

```
void algo3(Queue &Q){
   Stack S; int d;
   InitStack(S);
   while(!QueueEmpty(Q)){
      DeQueue(Q, d); Push(S, d);
   }
   while(!StackEmpty(S)){
      Pop(S, d); EnQueue(Q, d);
   }
}
```

解：算法的功能是利用栈 S 作为辅助将队列 Q 中的元素倒置。

22. 假如以顺序存储结构实现一个双向栈，即在一维数组的存储空间中存在着两个栈，它们的栈底分别设在数组的两个端点。试编写实现这个双向栈 tws 的三个操作：初始化 `Status InitStack(tws *t)`，入栈 `Status Push(tws *t, int i, SElemType x)`和出栈 `Status Pop(tws *t, int i, SElemType *x)`算法，其中 i 为 0 或 1，用以分别指示在数组两端的两个栈。

解：

```
#define STACK_INIT_SIZE (4)
```

```
#define STACK_INCREAMENT (10)
#define OK (0)
#define ERROR (1)
#define ERROR_OVERFLOW (2)
#define ERROR_EMPTY (3)
#define NULL (0)

typedef int SElemType;
typedef int Status;

typedef struct _tws
{/*顺序存储结构实现的双向栈*/
    SElemType *base;//数组基地址
    SElemType *top[2];//栈顶指针
    int stacksize;  //最大容量，按元素个数计数
}tws;

Status InitStack(tws *t) {
    t->base = (SElemType *)malloc(STACK_INIT_SIZE * sizeof(SElemType));
    if (!t->base) return ERROR_OVERFLOW;
    t->stacksize = STACK_INIT_SIZE;
    t->top[0] = t->base;//栈0的栈底和栈顶指针
    t->top[1] = t->base + STACK_INIT_SIZE-1;//栈1的栈底和栈顶指针
    return OK;
}

Status Push(tws *t, int i, SElemType x) {
    int step = (i == 0) ? 1 : -1;//0号栈，栈顶指针向数组末端移动；1号栈，栈顶指针向数组起始方向移动

    if (t->top[0] > t->top[1])//栈满了
        return ERROR_OVERFLOW;
    *(t->top[i]) = x;
    t->top[i] += step;
    return OK;
}

Status Pop(tws *t, int i, SElemType *x) {
    SElemType *base = (i == 0) ? t->base : t->base + t->stacksize - 1;//0号栈，栈底为数组头；1号栈，栈底为数组尾
    int step = (i == 0) ? 1 : -1;//0号栈，栈顶指针向数组末端移动；1号栈，栈顶指针向数组起始方向移动

    if (t->top[i] == base)//栈空了
```

```
        return ERROR_EMPTY;
    t->top[i] -= step;
    *x = *(t->top[i]);
    return OK;
}

Status DestroyStack(tws *t) {
    if (!t->base) return ERROR;
    free(t->base);
    t->top[0] = t->top[1] = NULL;
    return OK;
}
```

23. 如果希望循环队列中的元素都得到利用，则需要设置一个标志域，并以标志域真假来区分，尾指针和头指针相等时的队列状态是“空”还是“满”。试编写与此结构相应的队列基本操作。并从时间和空间角度来讨论设标志和不设标志这两种方法的使用范围。

解：

```
#define MAXQSIZE (3)
typedef int QElemType;
typedef struct _SqQueue {
    QElemType   *base;
    int         front;  //头指针，若队列不为空，指向队列头元素
    int         rear;   //尾指针，若队列不为空，指向队尾元素的下一个位置
    bool    empty;  //队列状态标识，true表示队列空
}SqQueue;

Status InitQueue(SqQueue *Q) {
    Q->base = (QElemType *)malloc(MAXQSIZE * sizeof(QElemType));
    if (!Q->base)
        return ERROR_OVERFLOW;
    Q->empty = true;
    Q->front = Q->rear = 0;
    return OK;
}

int QueueLength(SqQueue *Q) {
    int len = (Q->rear - Q->front + MAXQSIZE) % MAXQSIZE;
    if (len == 0 && !Q->empty){
        len = MAXQSIZE;
    }
    return len;
}

Status EnQueue(SqQueue *Q, QElemType e) {
```

```
    if (!Q->empty && Q->front == Q->rear)
        return ERROR_OVERFLOW;//队列满了，无法插入元素
    Q->base[Q->rear] = e;
    Q->rear = (Q->rear + 1) % MAXQSIZE;
    if (Q->rear == Q->front)
        Q->empty = false;//插入元素后，队列满了
    return OK;
}

Status DeQueue(SqQueue *Q, QElemType *e) {
    if (Q->empty && Q->front == Q->rear)
        return ERROR_EMPTY;//队列空了，无法删除元素
    *e = Q->base[Q->front];
    Q->front = (Q->front + 1) % MAXQSIZE;
    if (Q->rear == Q->front)
        Q->empty = true;//删除元素后，队列空了
    return OK;
}

Status DestroyQueue(SqQueue *Q) {
    if (!Q->base) return ERROR;
    free(Q->base);
    Q->front = Q->rear = -1;
    return OK;
}

Status PrintQueue(SqQueue *Q) {
    int pos = Q->front, len = QueueLength(Q), i = 0;

    printf("SqQueue:");
    for(i = 0; i < len; i++){
        printf("%d\t", Q->base[pos]);
        pos = (pos + 1) % MAXQSIZE;
    }
    printf("\n");
    return OK;
}
```

24. 设计表达式求值算法，采用_____数据结构最佳。

解：栈

25. 栈和队列是操作受限的线性表，入栈和出栈操作在_____位置进行，队列的插入和删除操作是在_____位置进行。

解：栈顶，栈顶，队尾，队头

26. “栈和队列是非线性结构”说法正确吗？
解：错误。

27. 队列的顺序存储方式一般组织成为环状队列的形式，而且一般队列头或尾其中之一应该特殊处理。若队列头指针为 front，队列尾指针为 rear，请写出循环队列判空和判满方法。
解：
循环队列解决了用向量表示队列所出现的“假溢出”问题，但同时又出现了如何判断队列的满与空问题。
针对这个问题有两种方法。第一是增加标识位，指示队列是空还是满。

第二种是牺牲一个单元。即 front==rear 为队空，而（rear+1）%表长==front 为队满，这里表长为 n。

（front 指针指向队头元素，队头元素被删除后，头指针更新为 front=(front+1)%表长；rear 指向队尾新元素的插入位置，若队未满，先插入元素 array[rear]=e，再更新位置，rear = (rear+1)%表长）

4 CHP4. 串

28. 设 s=’I AM A STUDENT’, t= ‘GOOD’, q = ‘WORKER’。求：StrLength(s), StrLength(t), SubString(s,8,7), SubString(t,2,1), Index(s, ‘A’), Index(s, t), Replace(s, ‘STUDENT’,q), Concat(SubString(s,6,2), Concat(t, SubString(s, 7, 8)))。
解：
StrLength(s)为 14， StrLength(t)为 4， SubString(s,8,7)为’STUDENT’， SubString(t,2,1)为’O’，Index(s, ‘A’) 为 3， Index(s, t) 为 0, Replace(s, ‘STUDENT’,q) 为 ’I AM A WORKER’，Concat(SubString(s,6,2), Concat(t, SubString(s, 7, 8)))为‘A GOOD STUDENT’
29. 在以链表存储串值时，存储密度是结点大小和串长的函数。假设每个字符占一个字节，每个指针占 4 个字节，每个结点的大小为 4 的整数倍。求结点大小为 4k，串长为 l 时的存储密度 d(4k,l)
解：
每个结点存储 4k 个字符，串长为 l 需要 ceil(l/4(k-1))个结点（ceil 为向上取整函数），因此存储密度为：$l/(4(k+1)\times\lceil l/4k\rceil)$
30. 以定长顺序存储表示串，不允许调用串的基本操作，编写算法，从串 S 中删除所有和串 T 相等的子串。
解：

```
#define OK (0)
#define ERROR (1)
#define ERROR_OVERFLOW (2)
#define ERROR_EMPTY (3)
#define NULL (0)
```

```c
#define MAXSTRLEN (255)
typedef unsigned char SString[MAXSTRLEN + 1];

void get_nextval(SString T, int nextval[]){
    int i = 1, j = 0;

    nextval[1] = 0;
    while (i < T[0]){
        if (j == 0 || T[i] == T[j]) {
            i++; j++;
            if (T[i] == T[j]){
                nextval[i] = nextval[j];
            }else{
                nextval[i] = j;
            }
        }else{
            j = nextval[j];
        }
    }
}

int Index_KMP(SString S, SString T, int nextval[], int pos) {
    int i = pos, j = 1;

    while (i <= S[0] && j <= T[0])
    {
        if (j == 0 || S[i] == T[j]) {
            i++; j++;
        }
        else {
            j = nextval[j];
        }
    }
    if (j > T[0])
        return i - T[0];
    else
        return 0;
}

int RemoveSubString(SString S, SString T) {
    int *nextval = (int *)malloc(sizeof(int) * (T[0]+1));
    int k = 1, i = 1, j = 0;
```

```
    if (nextval == NULL)
        return ERROR_OVERFLOW;
    memset(nextval, 0, sizeof(int) * (T[0] + 1));
    get_nextval(T, nextval);

    j = Index_KMP(S, T, nextval, i);
    while (j > 0 && i <= S[0] - T[0] + 1)
    {
        for (; i < j; ){//本次检索起始位置i，模式出现起始位置j，i到j-1之间的字符应该保留
            S[k++] = S[i++];
        }
        i += T[0];//下一次检索位置
        j = Index_KMP(S, T, nextval, i);
    }
    while (i <= S[0])
    {
        S[k++] = S[i++];
    }
    S[0] = k - 1;//更新长度
    free(nextval);
    return OK;
}
```

31.

5 CHP5. 数组和广义表

32. 广义表（(a)，b）的表头是_____，表尾是_____。
解：广义表（(a)，b）的表头是__(a)__，表尾是__(b)_____。

33. 设有数组 A[i，j]，数组的每个元素长度为 8 字节，i 的值为 1 到 10 ，j 的值为 1 到 4，数组从内存首地址 BA 开始顺序存放。请问，以列为主序和以行为主序存放时，元素 A[9，3]的存储首地址相差_____。
解：
已知第 1 维长度为 10，已知第 2 维长度为 4。
以行为主序时，映像函数 LOC(i,j)=LOC(1,1)+(i-1)*b2*L+(j-1)*L，所以 LOC(9,3)=LOC(1,1)+(i-1)*b2*L+(j-1)*L=LOC(1,1)+8*4*8+2*8=LOC（1,1）+272；
以列为主序时，映像函数 LOC(i,j)=LOC(1,1)+(j-1)*b1*L+(i-1)*L，所以 LOC(9,3)=LOC(1,1)+(j-1)*b1*L+(i-1)*L=LOC(1,1)+2*10*8+8*8=LOC（1,1）+224；
可见，两种方式，首地址相差 48 字节。

34. 请分析稀疏矩阵压缩存储前后空间效率和时间效率的变化。
解：稀疏矩阵包含大量 0（或相同）元素，因此可以压缩存储，提高空间效率。压缩存储后，非零元素位置分为有规律和无规律两种。有规律的稀疏矩阵（例如上三角矩阵）压缩后，下标和元素物理存储位置之间仍然存在意义映射关系，可随机存取。

35. 已知四维数组 A 的维长度依次是 3,2,2,4（下标从 0 起始），设数组起始地址为 LOC(0,0,0,0)，请问以行为主序时，LOC(2,1,0,2)是多少？
解：LOC(2,1,0,2) = LOC(0,0,0,0)+(2*2*2*4+1*2*4+0*4+2)*L
L 为每元素所占字节数

36. 已知三维数组 A 的维长度依次是 2,3,3（下标从 0 起始），请按行优先顺序（即以行序为主序）枚举数组元素
解：a(0,0,0),a(0,0,1),a(0,0,2), a(0,1,0),a(0,1,1),a(0,1,2), a(0,2,0),a(0,2,1),a(0,2,2),
a(1,0,0),a(0,0,1),a(0,0,2), a(1,1,0),a(1,1,1),a(1,1,2), a(1,2,0),a(1,2,1),a(1,2,2),

37. 已知矩阵如下，请使用快速算法实现矩阵转置，要求写出必要步骤。

$$\begin{bmatrix} 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 3 & 0 & 0 & 2 \end{bmatrix}$$

解：M（矩阵元素 a.data 以行主序排列），转置后矩阵为 T（矩阵元素 b.data 也以行主序排列，b.data 的是 a.data 以列为主序的排列）。所以，知道 M 中各列起始元素位置，就相当于知道了 T 中各行起始元素位置，就能快速将 M 中元素放入正确位置

辅助数组记录 M 中各列元素个数及起始位置

col	1	2	3	4	5
num	1	1	0	2	2
cpos	1	2	3	3	5

M 的三元组顺序表表示（行序）

Id	i	j	E
1	1	1	1
2	1	4	2
3	2	4	1
4	2	5	-1
5	3	2	3
6	3	5	2

查询第一列首个元素位置 cpos 为 1，在 N 中放入该元素为，第一列下一个元素存放位置应更新为 cpos：2

col	1	2	3	4	5
num	1	1	0	2	2
cpos	1 (2)	2	3	3	5

M

Id	i	j	E
1	1	1	1
2	1	4	2
3	2	4	1
4	2	5	-1
5	3	2	3
6	3	5	2

N

Id	i	j	E
1	1	1	1
2			
3			
4			
5			
6			

查询第 4 列首个元素位置 cpos 为 3，在 N 中放入该元素为，第 4 列下一个元素存放位置应更新为 cpos：4

col	1	2	3	4	5
num	1	1	0	2	2

M

Id	i	j	E
1	1	1	1
2	1	4	2
3	2	4	1
4	2	5	-1

N

Id	i	j	E
1	1	1	1
2			
3	4	1	2
4			

cpos	2	2	3	3 (4)	5		5	3	2	3		5						
							6	3	5	2		6						

以此类推，得到最终表示为：

col	1	2	3	4	5
num	1	1	0	2	2
cpos	2	3	3	5	7

M

Id	i	j	E
1	1	1	1
2	1	4	2
3	2	4	1
4	2	5	-1
5	3	2	3
6	3	5	2

N

Id	i	j	E
1	1	1	1
2	2	3	3
3	4	1	2
4	4	2	1
5	5	2	-1
6	5	3	2

38. 已知矩阵如下，请使用行逻辑链接三元组顺序表实现矩阵相乘，要求写出必要步骤。

$$Q = M \times N = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 3 & 0 & 0 & 2 \end{bmatrix} \times \begin{bmatrix} 3 & 0 \\ 0 & 2 \\ 0 & 1 \\ -1 & 0 \\ 2 & 0 \end{bmatrix}$$

解：“带行链信息”的三元组表尾行逻辑链接顺序表，采用此结构能快速实现矩阵相乘

M

Id	i	J	E
1	1	1	1
2	1	4	2
3	2	4	1
4	2	5	-1
5	3	2	3

N

Id	I	j	E
1	1	1	3
2	2	2	2
3	3	2	1
4	3	2	1
5	4	1	-1

Q

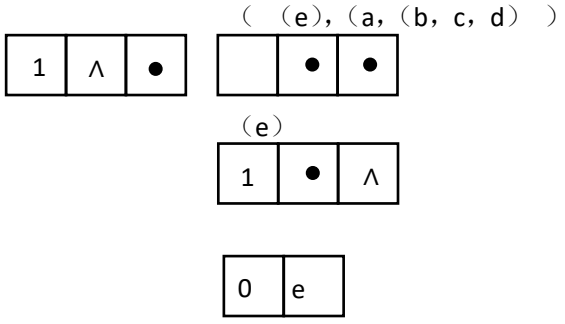
6	3	5	2		6	5	1	2		

计算 Q 的第一行	计算 Q 的第二行
$1.\left[q_{1,1} \quad q_{1,2}\right]=\left[0 \quad 0\right]$ $2.\mathrm{M}(1,:):(1,1,1)(1,4,2)$ $3.\left[q_{1,1} \quad q_{1,2}\right]=\left[0 \quad 0\right]+\left[m_{1,1}\times n_{1,1} \quad 0\right]$ $=\left[0 \quad 0\right]+\left[1\times 3 \quad 0\right]=\left[3 \quad 0\right]$ $4.\left[q_{1,1} \quad q_{1,2}\right]=\left[3 \quad 0\right]+\left[m_{1,4}\times n_{4,1} \quad 0\right]$ $=\left[3 \quad 0\right]+\left[2\times(-1) \quad 0\right]=\left[1 \quad 0\right]$	$1.\left[q_{2,1} \quad q_{2,2}\right]=\left[0 \quad 0\right]$ $2.\mathrm{M}(2,:):(2,4,1)(2,5,-1)$ $3.\left[q_{2,1} \quad q_{2,2}\right]=\left[0 \quad 0\right]+\left[m_{2,4}\times n_{4,1} \quad 0\right]$ $=\left[0 \quad 0\right]+\left[1\times(-1) \quad 0\right]=\left[-1 \quad 0\right]$ $4.\left[q_{2,1} \quad q_{2,2}\right]=\left[-1 \quad 0\right]+\left[m_{2,5}\times n_{5,1} \quad 0\right]$ $=\left[-1 \quad 0\right]+\left[(-1)\times(2) \quad 0\right]=\left[-3 \quad 0\right]$
计算 Q 的第三行	Q
$1.\left[q_{3,1} \quad q_{3,2}\right]=\left[0 \quad 0\right]$ $2.\mathrm{M}(3,:):(3,2,3)(3,5,2)$ $3.\left[q_{3,1} \quad q_{3,2}\right]=\left[0 \quad 0\right]+\left[0 \quad m_{3,2}\times n_{2,2}\right]$ $=\left[0 \quad 0\right]+\left[0 \quad 3\times 2\right]=\left[0 \quad 6\right]$ $4.\left[q_{3,1} \quad q_{3,2}\right]=\left[6 \quad 0\right]+\left[m_{3,5}\times n_{5,1} \quad 0\right]$ $=\left[0 \quad 6\right]+\left[(2)\times(2) \quad 0\right]=\left[4 \quad 6\right]$	

Id	i	J	E
1	1	1	1
2	2	1	-3
3	3	1	4
4	3	2	6

39. 给出广义表 D=（（），（e），（a，（b，c，d））） 的存储结构

解答：见书 P109



6 CHP6. 树和二叉树

40. 假定一棵树的广义表表示为 A（C，D（E（K（L）），F，G），H（I，J）），则树中所含的结点数为_____个，树的深度为_____，树的度为_____。

解：假定一棵树的广义表表示为 A（C，D（E（K（L）），F，G），H（I，J）），则树中所含的结点数为__11__个，树的深度为_5___，树的度为__3__。

41. 设有 n 个结点的完全二叉树的深度为_____；如果按照从自上到下、从左到右从 1 开始顺序编号，则第 i 个结点的双亲结点编号为_____，右孩子结点的编号为_____。

解：$[\log_2 n]+1$，$[i/2]$，$2\times i+1$

42. 深度为 5 的二叉树至多有（　　）个结点。

A．16　　B．32　　C．31　　D．10

解：由满二叉的性质可知，结点个数为2^n-1=31。满二叉的结点数是相同深度的不同类型的二叉树中，结点最多的。由此得，深度为 5 的二叉树的结点的上限为 31 个。

43. 树最适合用来表示（　　）。

A．有序数据元素　　B．无序数据元素

C．元素之间具有分支层次关系的数据　　D．元素之间无联系的数据

E．线性关系数据　F.集合关系数据　G.图或网关系数据（多对多）

解：C

44. 下列数据结构属于线性结构的是（　　）。

A. 图　　B. 树　C. 二叉平衡树　　D. 栈 E.队列 F.线性表

解：D,E,F

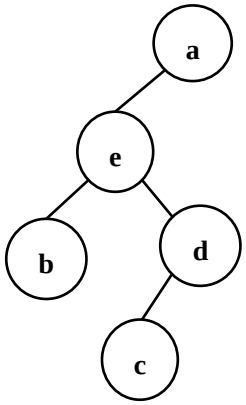
45. 若一棵二叉树的前序遍历序列为 a、e、b、d、c，后序遍历序列为 b、c、d、e、a，则根结点的孩子结点（　　）。

A．只有 e　　B．有 e、b　　C．有 e、c　　D．无法确定

解：a 是根结点。对比先序和后序可知，e 相对于 b,c,d 三个结点构成的整体，访问顺序发生变化，从最前变为了最后，所以可以断定，e 是根，b,c,d 均是其子孙结点。

再观察集合{b,d,c}，先序和后序中，{b}和{d,c}的相对顺序不变，可见它们是两棵子树。

而{d,c}中，先序是 d 在 c 前，后序时，d 在 c 后，可见，这棵子树中，d 是双亲，c 是孩子



如左图所示二叉树，前序遍历序列为 a、e、b、d、c，后序遍历序列为 b、c、d、e、a。其中，e 可以是 a 的右孩子，c 也可以是 d 的右孩子。包含左图在内，有四棵二叉树符合上述情况。虽然四棵二叉树不同，但它们的层次关系相同，区别仅在于结点 e 和 c 是其双亲的左孩子还是右孩子。

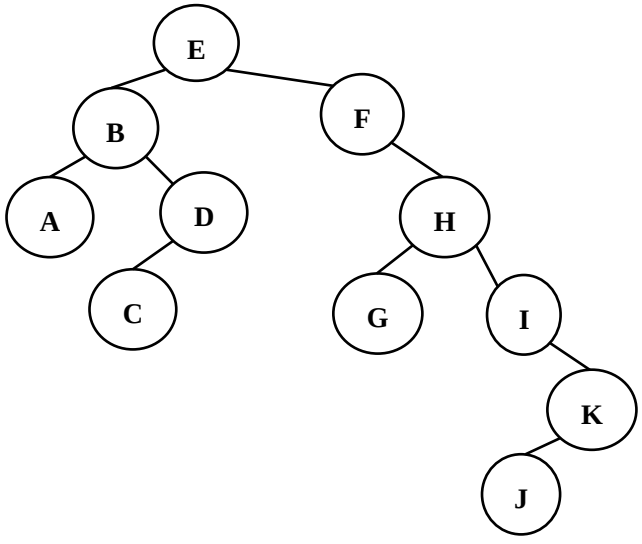
46. 假设一棵二叉树的先序序列为 EBADCFHGIKJ，中序序列为 ABCDEFGHIJK。请画出此二叉树。

解：

(1) 根据先序可知，E 是根；根据中序可知，(ABCD)是其左子树结点集，(FGHIJK)是其右子树结点集。

(2) 子树(ABCD)中，根据先序可知，B 是根；根据中序可知，(A)是其左子树结点集，(CD)是其右子树结点集。

(3) 依次类推，可得下图（结果是唯一的）



47. 将二叉树 T 转换为对应的森林 F，已知二叉树 T 中叶结点、没有左孩子的结点和没有右

孩子的结点数分别为 m，n，和 k，请问是否能根据这些信息指明 F 中叶结点的个数？若能，F 中叶结点的个数是？若不能，还需要知道二叉树的什么信息？

解：根据二叉树和树之间的转换规则，树中结点的孩子转换为二叉树结点的左孩子，因此，二叉树中，若结点没有左孩子，其对应的树结点没有孩子，即为叶子结点。所以二叉树中没有左孩子的结点数与树种叶子结点的数目相同。

48. 在下述论述中，正确的是（　　）。

①只有一个结点的二叉树的度为 0；　　②二叉树的度为 2；

③二叉树的左右子树可任意交换；

④深度为 K 的完全二叉树的结点个数小于或等于深度相同的满二叉树。

⑤二叉树一定不是空树。

⑥二叉树的度可能为 0、1 或 2。

⑦二叉树的子树不一定是二叉树。

A．①②③　　B．②③④　　C．②④　　D．①④⑥

解：

① 是正确的，只有一个结点，则没有孩子，所以结点的度为 0。而树只有一个结点，所以结点的度就是树的度，得树的度为 0.

② 是错误的。二叉树中，每个结点的度至多为 2，可以是 0 或 1.若整个树所有结点的度均为 1，例如，单支，那么二叉树的度也就为 1 了。

③ 是错误的。左右子树不可交换，交换以后的二叉树是不同的二叉树。

④ 是正确的。同样深度的二叉树中，满二叉树的结点数最多。

49. “完全二叉树的某结点若无左孩子，则它必是叶结点。”说法正确吗？

解：正确。

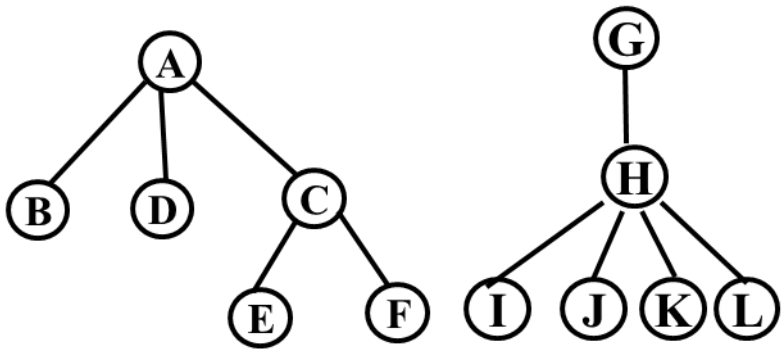
根据完全二叉树的特点，若无左孩子，必定没有右孩子。

50. 如下图所示的森林：

(1) 求树(a)的先根序列和后根序列；

(2) 求森林先序序列和中序序列；

(3)将此森林转换为相应的二叉树。



（a）　　（b）

解：

（1）　先根序列（先访问根，再依次先根遍历每棵子树）

A,B,D,C,E,F

后跟序列：（先依次后根遍历每棵子树，再访问根）

B,D,E,F,C,A

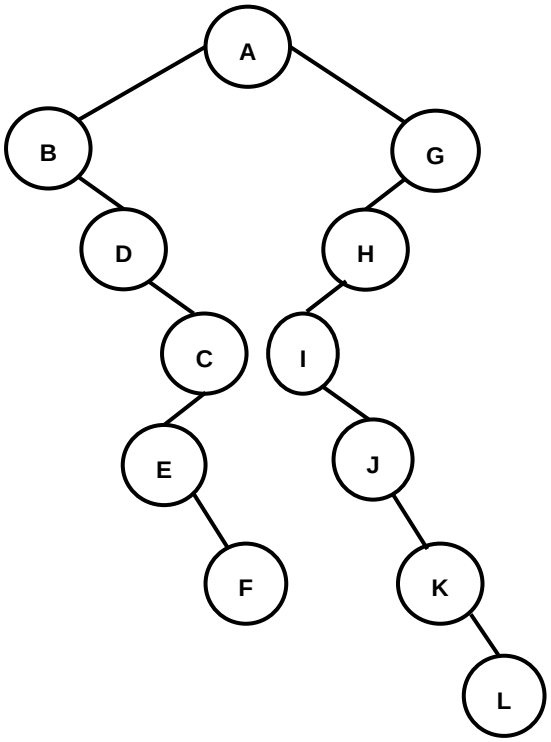
（2） 森林先序：先访问第一棵树的根结点；先序遍历第一棵树中根结点的子树森林；再先序遍历除第一棵树以外的树构成的子树森林。

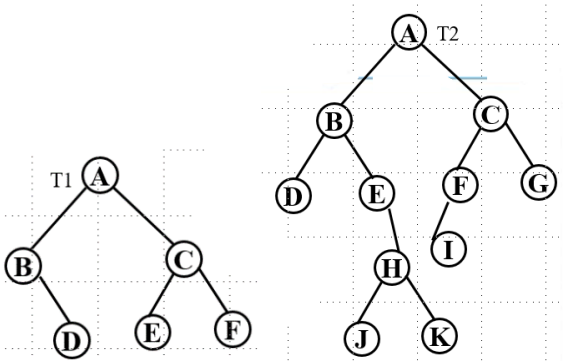
A，B，D，C，E，F，G，H，I，J，K，L

森林中序：先中序遍历一棵树中根结点的子树森林；再访问第一棵树的根结点；再中序遍历除第一棵树以外的树构成的子树森林。

B，D，E，F，C，A，I，J，K，L，H，G

（3） 森林转换为二叉树：第一棵树的根是二叉树的根，第一根树的子树森林是二叉树的左子树，第一棵树以外的其他树所组成的子树森林是二叉树的右子树。





要求：写出 T1 的后序遍历访问序列，T2 的先序遍历访问序列。

解：

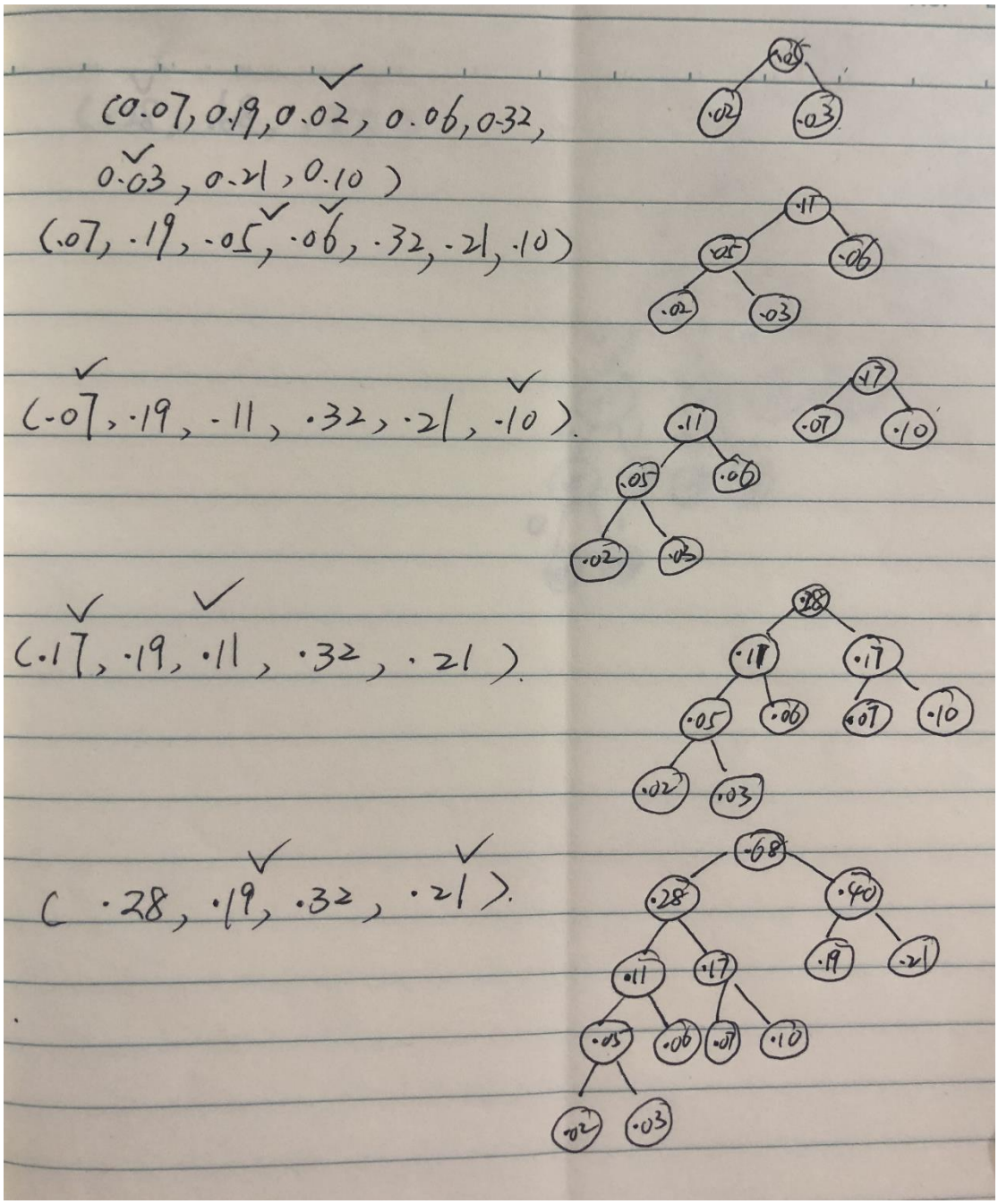
树 T1 访问序列：

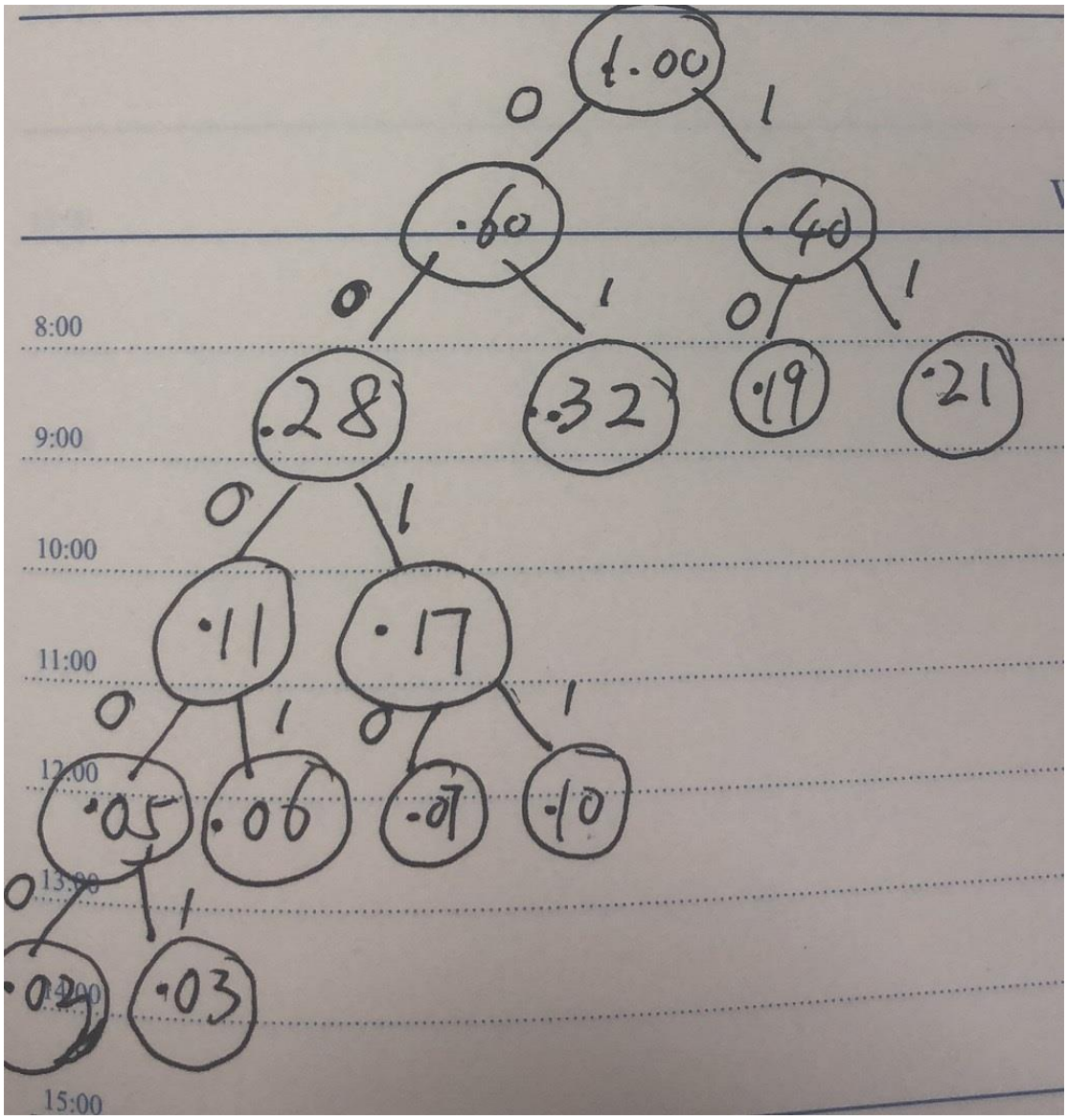
先序：A,B,D,C,E,F

树 T2 访问序列：
后序：D,J,K,H,E,B,I,F,G,C,A

51. 假设用于通信的电文仅有 8 个字母组成，出现频率分别为 0.07, 0.19, 0.02, 0.06, 0,32, 0.03, 0.21, 0.10。试为这 8 个字母设计赫夫曼编码。

解：





编码为：

0.02（5：00000）0.03（5：00001）0.06（4：0001）0.07（4:0010）0.10（4:0011）0.19（2:10）0.21（2:11）0.32（2:11）

霍夫曼树的带权路径长度为：

0.02*5+0.03*5+0.06*4+0.07*4+0.1*4+0.19*2+0.21*2+0.32*2=

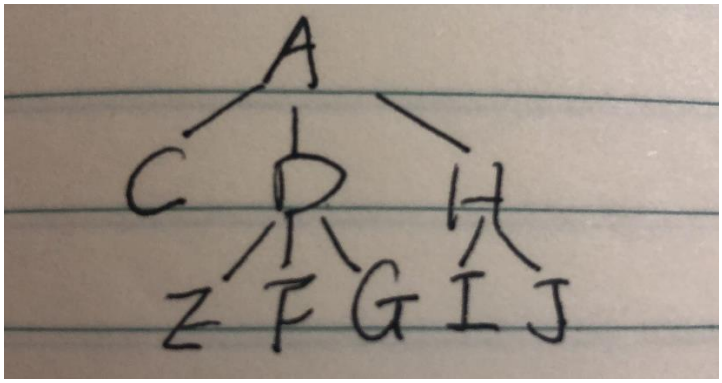
0.25+0.92+1.44=2.61

52. 深度为 k 的二叉树至多有_____个结点。

解：$2^k - 1$

53. 已知一棵树的广义表表示为 A（C，D（E，F，G），H（I，J）），请画出这课树。

解：



7 CHP7. 图

54. 遍历图的基本方法有_____优先搜索和广度优先搜索。_____优先搜索适于用使用递归方法实现，_____优先搜索适于使用队列实现。

解：深度，深度，广度

55. 已知图有 n 个顶点和 e 条边，若图为有向图，采用邻接表存储，那么边结点有_____个；若图为无向图，采用逆邻接表存储，边结点有_____个，用邻接多重链表存储，边结点有_____个。

解：e，2e，e

56. n 个顶点的连通图至少有_____条边，而 n 个顶点的强连通图至少有_____条弧。

解：

因为无向图的边没有方向性，两个顶点间有边关联就意味着顶点间存在着双向路径，所以，n 个顶点的连通图至有 n-1 条边。

有向图要形成任意一对顶点间具有路径，边数使用最少的构成方法是形成环，所以，n 个顶点的强连通图至少有 n 条弧。

57. “对于任意一个图，从它的某个结点进行一次深度或广度优先遍历可以访问到该图的每个顶点。”这说法正确吗？

解：错误。对于连通图，才能通过一次遍历访问所有顶点。

58. 已知邻接矩阵定义如下，则这是一个_____。请绘制出该网。并给出该网的深度和广度优先遍历。（设从 v1 出发）

∞ 3 2 1 ∞

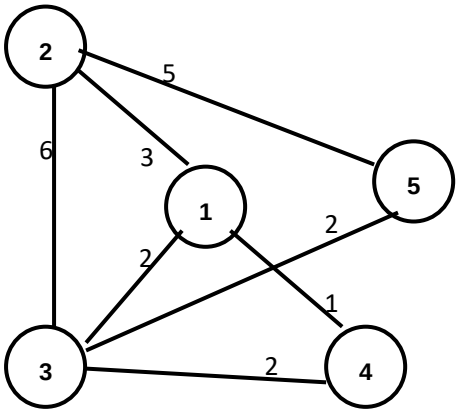
3 ∞ 6 ∞ 5

2 6 ∞ 2 2

1 ∞ 2 ∞ ∞

∞ 5 2 ∞ ∞

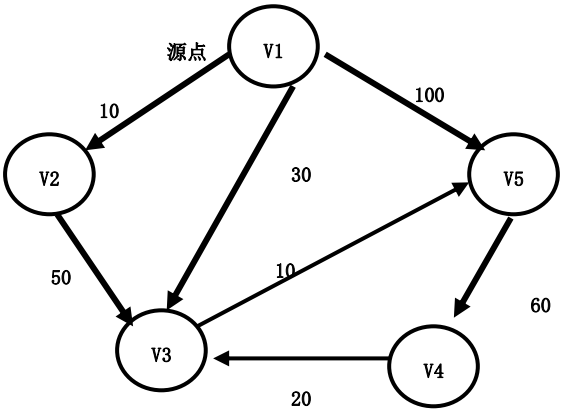
解：



广度优先：1->2->3->4->5

深度优先：1->2->3->4->5

59. 已知有向网如右图所示，请：



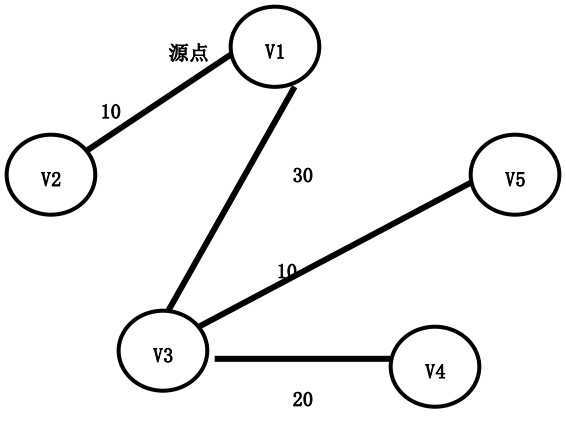
（1）若将所有弧改为边，请分别使用 Prim 和 Kruskal 算法找到最小生成树，并逐一写出步骤。

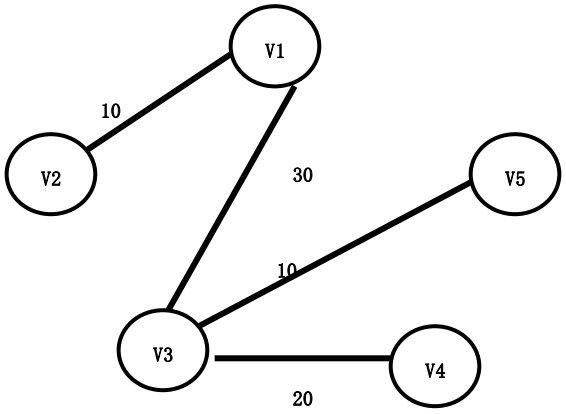
（2）若源点为 V1，请使用 Dijkstra 算法，找出 V1 到其他顶点的最短路径，并逐一写出步骤。

（3）请使用 Floyd 算法，找出有向网各个顶点之间的最短路径，并逐一写出步骤

解：

(1) 若将所有弧改为边，请分别使用 Prim 和 Kruskal 算法找到最小生成树，并逐一写出步骤。

Prim 算法：（说明，此处仅为了方便同学们温故原理，考试应答应按照算法过程给出每次迭代中，关键辅助数据的变化。） V={v1}，选（v1,v2）10， V={v1，v2}，选（v1,v3）30， V={v1，v2，v3}，选（v3,v5）10， V={v1，v2，v3，v5}，选（v3,v4）20，	

首先选边(v1,v2)10, 然后选边(v3,v5)10 再选边(v3,v4)20 再选边(v3,v1)30 （以上选边的过程是，加入新选的边，不能形成回路）	

Prim：

	1（v2）	2(v3)	3(v4)	4(v5)	U	V-U	K
adjvex lowcost	V1 10	V1 30		V1 100	{v1}	{v2,v3,v4,v5}	1
adjvex lowcost	 0	V1 30		V1 100	{v1,v2}	{v3,v4,v5}	2
adjvex lowcost	 0	 0	V3 20	V3 10	{v1,v2,v3}	{v4,v5}	4
adjvex lowcost	 0	 0	V3 20	 0	{v1,v2,v3,v5}	{v4}	

1. 初始状态，U={v1}, V-U={v2,v3,v4,v5},第一行记录从集合 U 中顶点到 V-U 中对应顶点的最小代价边。可知，到 v2 的最小代价边是(v1,v2)，权值为 10，关联的顶点为 v1。类似地，得出到顶点 v3，v4，v5 的最小代价边和权值信息。特别地，若不存在从 U 到 V-U 中顶点的最小代价边，则应设置为机器允许的最大值，方便起见，这里留空。所有从 U 到 V-U 的边，代价最小的边是**(v1,v2)**。
2. 将最小代价边(v1,v2)依附的顶点 v2 加入集合 U，则 V-U 变为{v3,v4,v5}。顶点 v2 的加入，可能使从 U 到 V-U 的最小代价边变化。因此，需要比较原最小代价边，以及从顶点 v2 到 V-U 中各顶点的边，并用较小者作为新的最小代价边。本轮中，(v2,v3)权值大于(v1,v3)，因此最小代价边不更新，仍然是(v1,v3)。依次类推，比较 v2 到 v4 和 v5 的边。这里，因为 v2 已经加入 U，本轮及后续过程，挑最小代价边时，不需要考虑该顶点，因此置其对应权值为 0. 本轮所有从 U 到 V-U 的边，代价最小的边是**(v1,v3)**。
3. 顶点 v3 加入集合 U，并更新相应边后，本轮所有从 U 到 V-U 的边，代价最小的边是**(v3,v5)**。
4. 顶点 v5 加入集合 U，并更新相应边后，本轮所有从 U 到 V-U 中边，代价最小的边是**(v3,v4)**。
5. 算法完成，得到最小生成树。

(2)使用 Dijstra 算法找最短路径，从 V1 出发

终点	最短路径

V2	10 (v1,v2)			
V3	30 (v1,v3)	30 (v1,v3)		
V4	∞	∞	∞	100(v1,v3,v5,v4)
V5	100(v1,v5)	100(v1,v5)	40 (v1,v3,v5)	
Vj	V2	V3	V5	V4
S	{v1,v2}	{v1,v2,v3}	{v1,v2,v3,v5}	{v1,v2,v3,v5}

找从 v1 出发到各顶点的最短路径，为方便描述算法，用邻接矩阵 arcs 表示带权有向图，arcs[i][j]表示弧<vi,vj>上的权值。若<vi,vj>不存在，则表示为∞（算法实现时，用计算机允许的最大值来表示）。S 表示已找到的从 v1 出发的最短路径终点集，初始为空集。

1. 初始情况下，顶点 v1 到其余各顶点 i 的最短路径 D[i]=arcs[1][i]，如第 2 列所示。
2. 选择 D[i],i=2,3,4,5 中最小权值 10 所对应终点 v2 加入 S，表示找到了从 v1 出发到 v2 的最短路径，即(v1,v2),D[2]=10。
3. 比较 D[i]和 D[2]+arcs[2][i]，i=3,4,5，若 D[2]+arcs[2][i]较小，则更新 D[i]为 D[2]+arcs[2][i]。更新后结果如第 3 列所示。
4. 选择 D[i],i=3,4,5 中最小权值 30 所对应终点 v3 加入 S，表示找到了从 v1 出发到 v3 的最短路径，即(v1,v3),D[3]=30。
5. 比较 D[i]和 D[3]+arcs[3][i]，i=4,5，若 D[3]+arcs[3][i]较小，则更新 D[i]为 D[3]+arcs[3][i]。更新后结果如第 4 列所示。
6. 选择 D[i],i=4,5 中最小权值 40 所对应终点 v5 加入 S，表示找到了从 v1 出发到 v5 的最短路径，即(v1,v3,v5),D[5]=40。
7. 比较 D[i]和 D[5]+arcs[5][i]，i=4，若 D[5]+arcs[5][i]较小，则更新 D[i]为 D[5]+arcs[5][i]。更新后结果如第 5 列所示。
8. 选择 D[i],i=5 中最小权值 100 所对应终点 v4 加入 S，表示找到了从 v1 出发到 v4 的最短路径，即(v1,v3,v5,v4),D[4]=100。

（3）请使用 Floyd 算法，找出有向网各个顶点之间的最短路径，并逐一写出步骤

解：

$$D^{-1}=\begin{bmatrix}0&10&30&\infty&100\\\infty&0&50&\infty&\infty\\\infty&\infty&0&\infty&10\\\infty&\infty&20&0&\infty\\\infty&\infty&\infty&60&0\end{bmatrix}$$ 初始化，
$$D^{(1)}=\begin{bmatrix}0&10&30&\infty&100\\\infty&0&50&\infty&\infty\\\infty&\infty&0&\infty&10\\\infty&\infty&20&0&\infty\\\infty&\infty&\infty&60&0\end{bmatrix}$$ $D^{(1)}(2,4)=D^{(-1)}(2,1)+D^{(-1)}(1,4)$ （加入 v1）

$$D^{(2)}=\begin{bmatrix}0&10&30&\infty&100\\\infty&0&50&\infty&\infty\\\infty&\infty&0&\infty&10\\\infty&\infty&20&0&\infty\\\infty&\infty&\infty&60&0\end{bmatrix}$$ $D^{(2)}(1,3)=D^{1}(1,2)+D^{(1)}(2,3)$ （加入 v2）

$$D^{(3)}=\begin{bmatrix}0&10&30&\infty&40\\\infty&0&50&\infty&60\\\infty&\infty&0&\infty&10\\\infty&\infty&20&0&30\\\infty&\infty&\infty&60&0\end{bmatrix}$$ （加入 v3）
$$D^{(4)}=\begin{bmatrix}0&10&30&\infty&40\\\infty&0&50&\infty&60\\\infty&\infty&0&\infty&10\\\infty&\infty&20&0&30\\\infty&\infty&80&60&0\end{bmatrix}$$ （加入 v4）
$$D^{(5)}=\begin{bmatrix}0&10&30&100&40\\\infty&0&50&120&60\\\infty&\infty&0&70&10\\\infty&\infty&20&0&30\\\infty&\infty&80&60&0\end{bmatrix}$$ （加入 v5）

60.

8 CHP9. 查找

61. 已知有序表为(9、17、30、35、43、55、59、71、88、90、99)，当用折半查找法查找 9 时，需_____次查找成功。

解：依次比较的关键字是：55,30,9，所以需要 3 次才能查找成功。

9	17	30	35	43	55	59	71	88	90	99
1	2	3	4	5	6	7	8	9	10	11

62. 如何遍历二叉排序树，能得到一个递增的关键字（结点）？

解：中序遍历

63. 对于任意集合元素，其关键字能确定唯一一棵二叉排序树。

解：这个说法错误。同一集合，其关键字输入顺序不同，得到的二叉排序树也不同。

64. 在任意一棵非空二叉排序树中，删除某结点后又将其插入，则所得二排序叉树与原二排序叉树相同。这个说法正确吗？

解：

不正确。删除结点后，二叉排序树的结构可能改变，再插入该接点时，新结点位置自然可能发生变化。

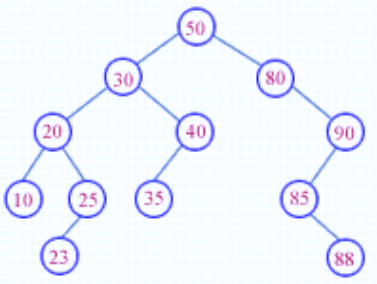
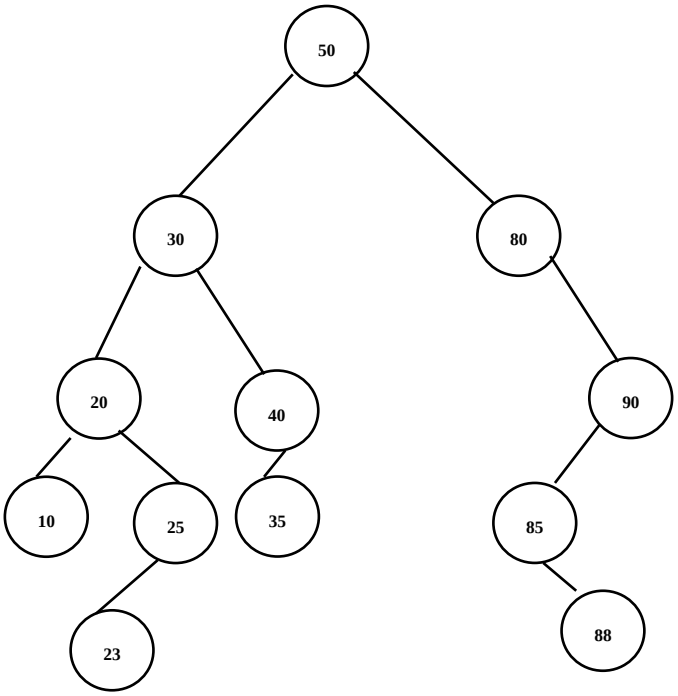
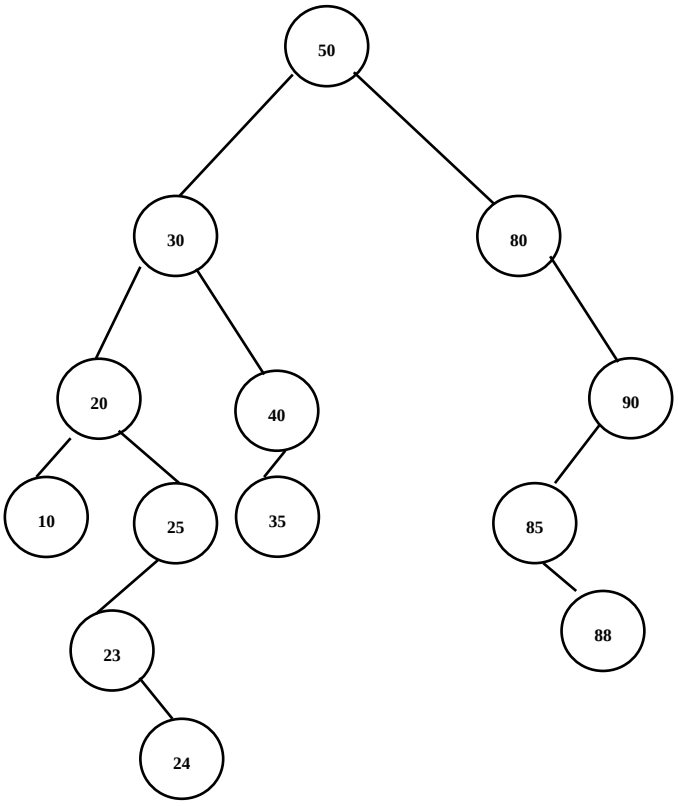
65. 已知待散列的线性表为（22，16，43，52，62），散列用的一维地址空间为[0..6]，假定选用的哈希函数是 H（K）= K mod 7，若发生冲突采用线性探测法处理，请构造出这个哈希表，并计算平均查找长度。

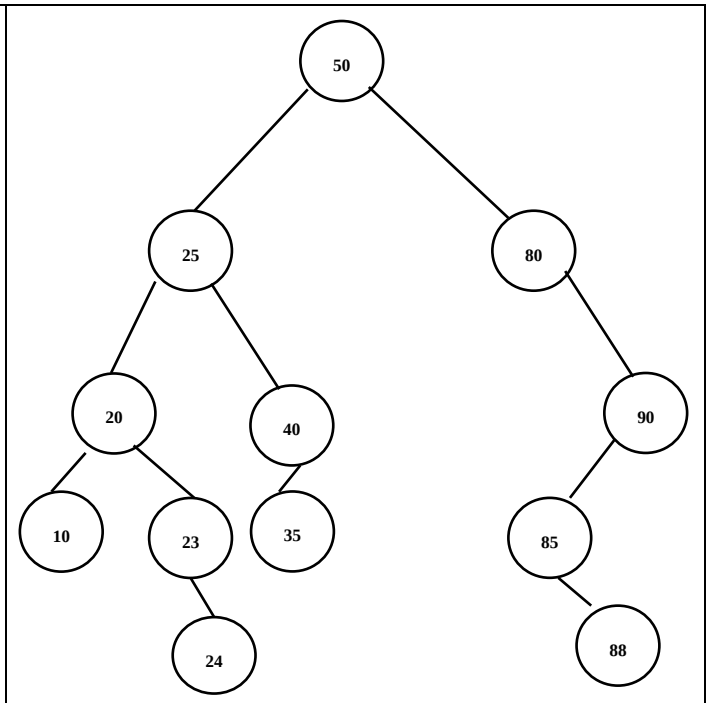
解：

0	1	2	3	4	5	6
	22（1）	16（1）	43（3）	52（2）		62（1）

平均查找长度为：1/5*(1*3+2*1+3*1)=8/5=1.6

66. 如果所示二叉排序树，请先画出插入结点 24 以后的二叉排序树，在此基础上，请再画出删除结点 30 以后的二叉排序树。

<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>插入 24</td>
<td> 
</td>
</tr>
</table>

删除 30	

67. “折半查找法的查找速度一定比顺序查找法快”说法正确吗？

解：

错。就平均查找长度而言，的确是折半查找算法速度更快。但对于单次查找而言，则未必。例如，n 个元素，查找第 1 个元素时，折半查找大约需要 log2N，而顺序查找时，一次就够了。