1—12章　　不考4.8.12章　　*不考

一. [illegible]

三. 综合应用

知识点：

- DS
  - 线性表
    - 循环链表、双向链表
    - 栈、队列
    - 串
    - 数组
    - 广义表
  - 树
  - 图

查找

排序 { 内 / 外

三种数据结构，两种运算.

§1. 算法分析

一. 《DS》研究的主要内容.

① 研究数据的逻辑及物理结构.

② 规定一组相适应的运算;

③ 设计相应的算法.

④ 分析算法的效率.

二. 算法的概念及特性.　　P12

完成特定功能的有限指令集

算法与程序的区别.

1. 描述方式上. { 计算机语言描述，机器上可执行. / 描述方式多种多样：　流程图、自然语言等

2. 有穷性上, { 程序可以无穷. / 算法具有有穷性.

三. 时间复杂度.

$O(f(n))$: { n: 算法的计算量. / $f(n)$: $n\uparrow$，执行时间的增长率. / $O(f(n))$: $n\uparrow$. 执行时间的上界.

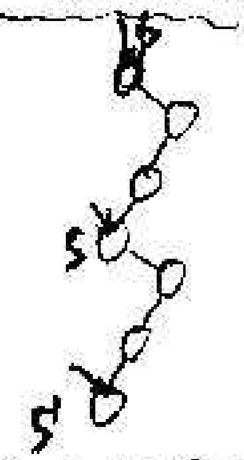
$O(n^2)$

1. 中序线索BT寻找后继及遍历算法

找p的后继： p↑.rtag = { 1 后继就是p↑.rchild; 0 从右孩子开始，沿左链域前进，直到某结点s，该结点无左孩子（即s↑.ltag=1），s就是p的后继。 }

[即右子树中最左下的结点]

```
FUNC  innext(p, thrt): thrlinktp; {中序线索BT寻找后继算法}
  s:=p↑rchild;
  IF  p↑.rtag=0
    THEN  WHILE  s↑.ltag=0  DO
                s:=s↑.lchild;
    RETURN(s)
ENDF;
```



不能说：右子树中最左下的结点、如图 p的后继是s而非s'!

```
PROC  thrt_inorder(t: thlinktp); {中序线索BT遍历算法}
  IF  t↑.lchild≠t          {二叉树为空时 t↑.lchild=t}
    THEN  [p:=t;
           WHILE  p↑.lchild≠t  DO      {定位到第一个结点}
               p:=p↑lchild;
           WHILE  p≠t  DO
           [ visit(p↑.data);
               p:=innext(p,t)
           ]
          ]
ENDP;
```

2. 后序线索BT寻找后继及遍历算法

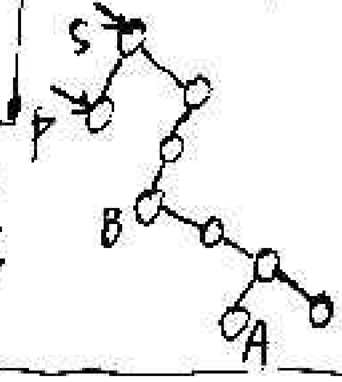
找p的后继：

- p若为根，则无后继
- 若p是双亲的右孩子，或是双亲唯一的左孩子，则后继为双亲
- 若p是双亲的左孩子且右兄弟存在，则后继是双亲右子树上按后序遍历的第一个结点。

```
FUNC postnext(p, thrt): thrlinktp;      {后序线索BT寻找后继算法}
  s:= PARENT(thrt, p);
  IF s=NIL THEN RETURN(s);
  IF p=s↑.rchild OR s↑.rtag=1    表示S无右孩子
     THEN RETURN(s);
☆ WHILE s↑.rtag=0 DO
     [ s:=s↑.rchild;  {从p的右兄弟开始，沿左链域前进，直到某结点S，~~该结点无左孩子~~}
       WHILE s↑.ltag=0 DO s:=s↑.lchild ];
   RETURN(s)
ENDF;
```

s':=s↑.rchild
while s'↑.lchild<>p
DO s':=s'↑.lchild

例如A结点



到B结点后，只是跳出了内层while循环，并未跳出外层while循环，直到A结点才跳出

```
PROC thrt_postorder(t:thrdlinktp);  {后序线索BT遍历算法}
  IF t↑.lchild≠t          {二叉树为空时，t↑.lchild=t}
    THEN [ p:=t; search:=true;
           WHILE search DO
              [ WHILE p↑.ltag=0 DO p:=p↑.lchild;
                IF p↑.rtag=0 THEN p:=p↑.rchild ELSE search:=false;]
           WHILE p≠t DO
              [ visit(p↑.data);
                p:=postnext(p,t)
              ]
         ]
ENDP;
```

仔细体会寻找第一个结点的方法！
(结合右上图)

3. 先序线索BT寻找后继及遍历算法

找p的后继:

{ 若p↑.ltag=0, 则后继为p↑.lchild
{ 否则，后继为p↑.rchild

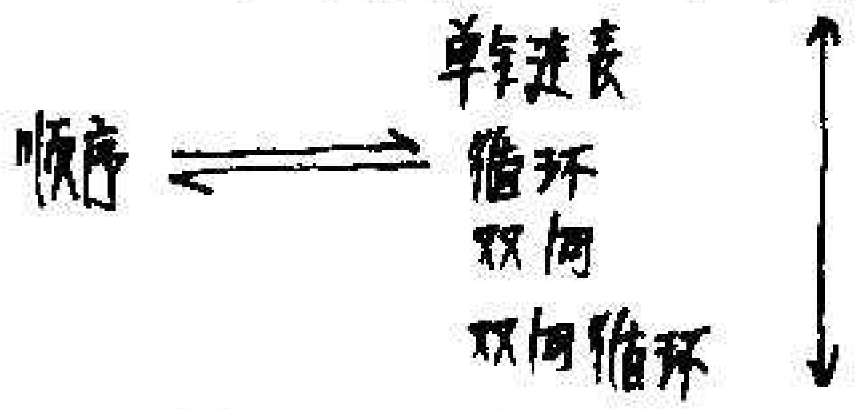
⇒

```
FUNC pre_next(p, thrt:thlinktp): thlinktp;
  IF p↑.ltag=0 THEN RETURN(p↑.lchild)
     ELSE RETURN(p↑.rchild)
ENDF;

PROC thrt_preorder(thrt: thlinktp);
  p:=thrt↑.lchild;
  WHILE p<>thrt DO
    [ visit(p↑.data);
      p:=pre_next(p, thrt)
    ]
ENDP;
```

§8. 存贮结构转换

一. 线性表顺序存贮与链式存贮结构的转换



1. 顺序 ⟶ 单链表

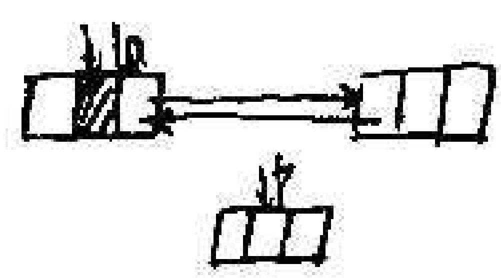
```
PROC sqlistToLinklist( V: sqlisttp; la: linklisttp);
  new(la);    la↑.next := NIL;          {产生空表}
  FOR i:= v.last DOWNTO 1 DO            {循环控制} 使插入方便
   [ new(p);   p↑.data:= v.[i];        {装填数据}
      p↑.next := la↑.next;              {插入链表}
      la↑.next := p ]   {从表尾到表头建立单链表}
ENDP;
```

在表头插入!

2. 顺序 ⟶ 循环链表

将上述算法中 la↑.next:=NIL 修改为 la↑.next:= la 即可

3. 顺序 ⟶ 双向链表



修改上述算法的初始化和插入部分.

la↑.next:=NIL 修改为 la↑.next := la↑.prior :=NIL

插入部分 p↑.next:=la↑.next
　　　　 la↑.next:=p

修改为 IF la↑.next≠NIL THEN [la↑.next↑.prior:=p;]
　p↑.next := la↑.next;
　la↑.next :=p;
　p↑.prior := la;

仔细体会!

4. 顺序 → 双向循环链表

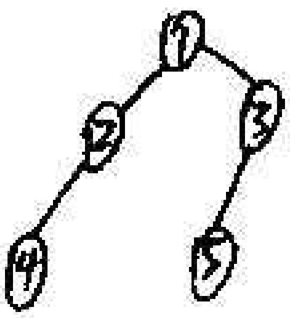
将上述算法中 la↑.next:=la↑.prior:=NIL

修改为 la↑.next:=la↑.prior:=la 即可

5. 单链表 → 顺序

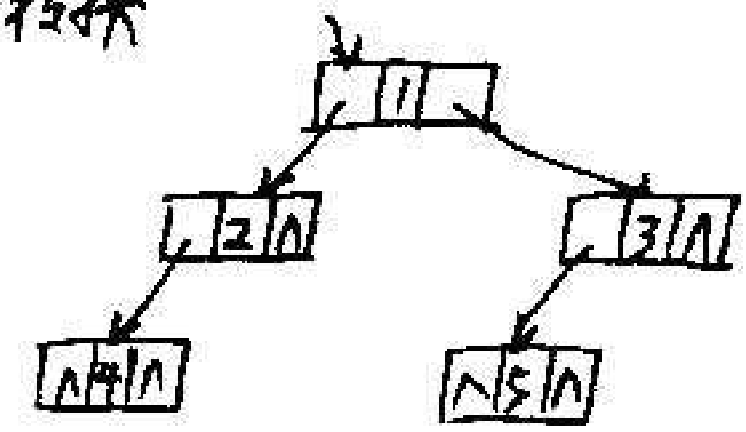
```
PROC linklistToSqlist(V: sqlisttp; la: linklisttp);
  p:=la↑.next;   i:=1;   V.last=0;   {设置移动指针，访数组下标}
  WHILE p≠NIL DO
    [V.[i]:=p↑.data;   i:=i+1;
      v.last:=v.last+1;
       p:=p↑.next
    ]
ENDP;
```

二、二叉树顺序存贮与链式存贮结构的转换



V.num=5

1	2	3	4	0	5	0



1. 顺序 → 链式　　借助队列的用法!

```
PROC SqBTToBitre(V: sqlisttretp; bt: bitreptr);
  bt:=NIL;  i:=1;
   IF V.num>0 THEN
    [new(bt);   bt↑.lchild:=bt↑.rchild:=NIL;
      bt↑.data:=V.[i];   INIQUEUE(Q);
      ENQUEUE(Q,bt);   j:=1;      {j为已形成结点数}
     WHILE NOT EMPTY(Q) DO
       [p:=DLQUEUE(Q);   i:=i+1;
          IF  j<V.num  AND  V.[i]≠0  THEN
```

```
        [ new(q);   j:=j+1;   p↑.lchild:=q;
          q↑.data := v.[i];
          q↑.lchild:=q↑.rchild:=NIL;
          ENQUEUE(Q,q)
        ];
      i:=i+1;
      IF j<v.num AND v.[i]≠0 THEN
        [ ...      处理右孩子      ]
    ]
  ]
ENDP;
```

2. 链式 → 顺序

```
PROC BitreToSQBT(V:sqlisttretp; bt:bitreptr);
   v.num:=0;
   IF bt≠NIL THEN
    [ i:=1;  INIQUEUE(Q1);  INIQUEUE(Q2);
      ENQUEUE(Q1,bt);  ENQUEUE(Q2,1);
      WHILE NOT EMPTY(Q1) DO
       [ p:=DLQUEUE(Q1);   j:=DLQUEUE(Q2);
         FOR k:=i TO j-1 DO v.[k]:=0;
         v.[j]:=p↑.data;
         v.num:=v.num+1;
         i:=j+1;
         IF p↑.lchild≠NIL THEN
           [ ENQUEUE(Q1,p↑.lchild);
             ENQUEUE(Q2,2*j)
           ]
```

```
        IF p↑.rchild ≠ NIL THEN
          [ ENQUEUE(Q1, p↑.rchild);
            ENQUEUE(Q2, 2*j+1)
          ]
      ]
    ]
ENDP;
```

三. 二叉树与线索 BT 的转换.

四. 邻接表与邻接矩阵的转换

邻接表:

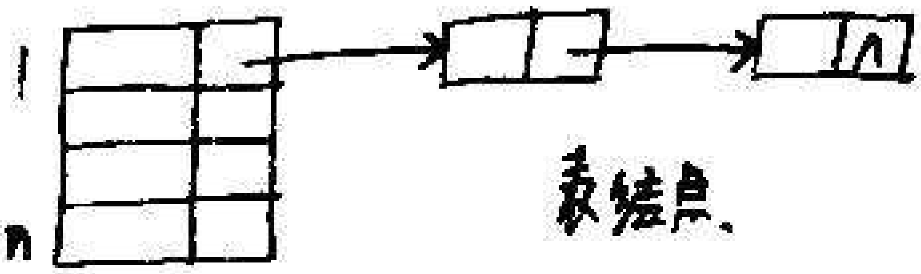
表结点

adjvex	nextarc	info
邻接点域	链域	数据域

头结点

vexdata	firstarc
数据域	链域

1. 邻接表生成邻接矩阵



表结点.

头结点

```
PROC listToMatrix (adlist, A);
  FOR i:=1 TO n DO
    FOR j:=1 TO n DO A[i,j]:=0; {邻接矩阵初始化}
  FOR i:=1 TO n DO
   [ p:=adlist[i].firstarc;
     WHILE p≠NIL DO
       [ A[i,p↑.adjvex]:=1; p:=p↑.nextarc]
   ]
ENDP;    {既适用于有向图, 也适用于无向图}
```

2. 矩阵 ⟶ 邻接表

```
PROC MatrixToList (adlist, A);
  FOR i:=1 TO n DO adlist[i].firstarc:=NIL; {邻接表初始化}
  FOR i:=1 TO n DO
    FOR j:=1 TO n DO
```

```
      IF A[i,j]=1 THEN
        [new(p); p↑.adjvex:=j;
         p↑.nextarc:=adlist[i].firstarc;
         adlist[i].firstarc:=p   {从表尾到表头的生成}
        ]                         但始终在表头插入!
ENDP;
```

3. 邻接表生成逆邻接表

```
PROC Invertlist(adlist1, adlist2);
  FOR i:=1 TO n DO adlist2[i].firstarc:=NIL {逆邻接表初始化}
  FOR i:=1 TO n DO
   [p:=adlist1[i].firstarc;
    WHILE p≠NIL DO
      [new(q); q↑.adjvex:=i;
       q↑.nextarc:=adlist2[p↑.adjvex].firstarc;
       adlist2[p↑.adjvex].firstarc:=q; {从表尾到表头生成}
       p:=p↑.next                        始终在表头插入!
      ]
   ]
ENDP;
```

§9. 二叉树的种种概念

1、满BT

一棵深度为k且有2^k-1个结点的二叉树称为满二叉树.

特点: ① 每一层结点数达到最大值

② BT的结点数达到最大值(深度一定)

③ BT的深度达到最小(结点数一定)

2. 完全BT

概念：$\boxed{\text{见书}P_{122}}$

特点：① $2^{k-1}-1 < n \le 2^k-1$ （n为结点数，k为层数）

② 叶结点只可能出现在最大和次最大层.

③ 每个结点的左、右子树深度之差为0或1.

④ n个结点的完全BT的深度为 $\lfloor \log_2 n \rfloor + 1$

⑤ 完全BT中任意一结点 $i\,(1 \le i \le n)$ 有，

$\{$ $\lfloor \frac{i}{2} \rfloor$ 为双亲，$2i$为左孩子，$2i+1$为右孩子

3. 平衡BT（AVL） 完全二叉树一定是AVL，但AVL树不一定是完全二叉树！

概念：AVL树或者是一棵空树，或者是左、右子树均为AVL树，且任一结点的左、右子树的深度之差的绝对值均≤1.

二叉树上结点的平衡因子：定义为该结点的左子树的深度减去它的右子树的深度

特点：① 任一结点的平衡因子只能为−1，0或1.

② AVL树的深度与 $\lceil \log_2 n \rceil$ 同数量级

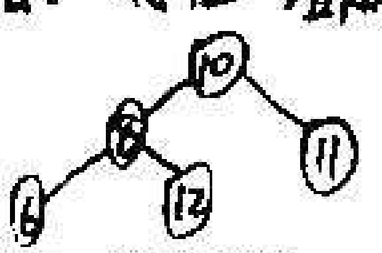
③ 平均查找长度ASL与深度相同.

4. 二叉排序树BST

概念：BST树或者是一棵空树，或者是左、右子树均为BST树，且任一结点左子树上的所有结点值均小于该结点值，任一结点右子树上的所有结点值均大于该结点值

判断BST：

算法思想：错。对任一结点，若左孩子<其值，且右孩子>其值，则为BST （X）

如，

特点: ① 中序遍历BST可得到一个有序序列

② BST上插入新结点,总是作为叶结点插入

③ BST的查找特性与树的型态有关

④ BST具有类似折半查找的特性,与BST的平衡度有关 又采用了链式存贮结构,尤其适用于经常要进行 {查找 插入 删除} 操作的有序表。折半查找不能为链表存储结构

5. 堆

概念: n个元素的序列 $\{k_1, k_2, \cdots, k_n\}$ 满足 $\begin{cases} k_i \le k_{2i} \\ k_i \le k_{2i+1} \end{cases}$ $(i=1,2,\cdots,\lfloor \frac{n}{2} \rfloor)$

当且仅当满足上述关系时,称之为堆。

▲堆一定是完全二叉树,但完全二叉树不一定是堆。

特点: ① 完全BT中所有非终端结点的值均不大于其左、右孩子的值。

② 堆顶元素为序列中的最小值。

6. 哈夫曼树(最优BT)

带权路径长度WPL最小的一类树,称为Huffman树

树中有m个叶,每个叶带有一个权值 W_i,根到叶结点i的路径长度为 l_i,则 $WPL=\sum_{i=1}^{m} W_i l_i$

特点. ① Huffman树不一定唯一,但WPL一定相等。

② 权值越大的叶结点越靠近根结点。

二. 各类BT相互关系.

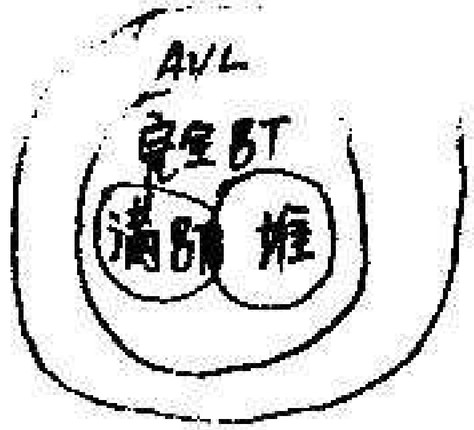
线索BT着眼于支持遍历

BST和堆是支持排序.

Huffman树考虑的是WPL

满BT、完全BT、AVL树可以认为是从树形研究BT

一定是 一定是



可见：满BT $\underset{\not\leftarrow}{\rightarrow}$ 完全BT $\underset{\not\leftarrow}{\rightarrow}$ AVL树

堆 $\underset{\not\leftarrow}{\rightarrow}$ 完全BT

BST不一定是AVL，但通过平衡化处理，可以得到具有BST特点的AVL树

完全BT是树的路径长度最短的BT，但并不一定是哈夫曼树.

§10. BST的判定及构造算法.

一. 判定算法1

```
FUNC  BST(t: bitreptr): Boolean;
  CASE
    t=NIL:  RETURN(true);
    Bst(t↑.lchild)  AND  Bst(t↑.rchild):
        IF  t↑.data > MAX(t↑.lchild)  AND
             t↑.data ≤ MIN(t↑.rchild)
        THEN  RETURN(true)
        ELSE  RETURN(false)
  ENDC
ENDF;

FUNC  MAX(p: bitreptr): integer;
   IF  p=NIL  THEN  RETURN(-Maxint)
     ELSE  [  WHILE  p↑.rchild ≠ NIL  DO  p:=p↑.rchild;
              RETURN(p↑.data)
           ]
ENDF;

FUNC  MIN(p: bitreptr): integer;
     IF  p=NIL  THEN  RETURN(Maxint)
```

计算法：$\sum_{i=1}$ 语句$_i$ 频度，　取高阶项，去掉常数.
分析法：根据功能，确定 $O(f(n))$. 如P22 算法 2-2

```
FOR  i:=1  TO  n  DO            n+1
    FOR  j:=1  TO  n  DO        n(n+1)
         x:=x+1;                n·n      O(n²)
```

§2. 线性表及其变形

一、线性表概念.

$L=(a_1, a_2, \cdots, a_n)$

a_i 的直接前驱为 a_{i-1}，a_i 的直接后继为 a_{i+1}

二. 线性表的存贮结构

顺序存贮
链式存贮

顺序存贮结构 特点：　(静态存贮结构)

- 逻辑相邻，其物理位置也相邻.　✓　优
- 可随机存取任一 DE.　✓　优
- 必须按最大可能预用存贮空间　缺
- 插入、删除要移动大量的 DE.　$\frac{n}{2}$　缺

链式存贮结构特点：

- 逻辑相邻，其物理位置不一定相邻
- 必须顺序存取 DE
- 动态申请，动态分配　new( )　dispose( )　✓
- 插入、删除 只需 修改指针　✓

三. 线性表的变形

1. 结点结构变化

```
        ELSE [ WHILE p↑.lchild≠NIL DO p:=p↑.lchild;
               RETURN (p↑.data)
             ]
ENDF;
```

判定算法2

```
FUNC BST2(t: bitreptr): Boolean;
  {初次调用时 predata 置为最小值 -Maxint}
  IF t≠NIL
    THEN IF t↑.data < predata
           THEN RETURN (false)
           ELSE [ predata := t↑.data;
                  RETURN (BST2(t↑.lchild) AND BST2(t↑.rchild))
                ]
    ELSE RETURN (true)
ENDF;
```

§11. 外部排序

一. 外排概念.

待排序记录的数量很大，以致内存一次不能容纳全部记录，在排序过程中尚需对外存进行访问的排序过程.

归并段　　逐趟进行归并　　得到有序文件

归并趟数. $S = \lceil \log_k m \rceil$　　m: 初始归并段数

$d \sim S$　　k: 归并路数

d: 读/写外存次数

由上式可见: $k\uparrow$ 或 $m\downarrow$ 使 $S\downarrow$. $d\downarrow$

知识点：

重点：线性表、二叉树、图

次重点：查找、排序、广义表、矩阵

题型：与前两年类似，①单选 ②简单应用 ③综合应用

章节：1～12章，划掉三章 4、8、12

第一章：《数据结构》这门课程研究的主要内容

算法概念及分析。类PASCAL语言。

第二章：线性表存贮结构及算法

第三章、栈、队列存贮结构及算法

第五章：稀疏矩阵的存贮结构

广义表概念及存贮结构，两个基本操作

第六章：BT、线索BT存贮结构及遍历算法

第七章：图的存贮结构及遍历算法

图的应用：最小生成树，关键路径（找法）

最短路径，拓扑排序

第九章：BST与AVL

Hash、冲突含义、解决冲突办法

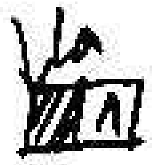
第十章：算法思想，稳定性概念及时间特性。

第十一章：置换——选择排序，多步排序（思想，能排出来，结论）

最小生成树：概念，算法（弄懂，数据结构，怎样不构成回路）

① 带头结点单链表

空表、头结点的指针域为"空".　La↑.next= NIL



```
p:= la;
WHILE p↑.next ≠ NIL DO
    ⋮
```

② 循环单链表

③ 双向链表

2. 操作上限制

① 栈、插入删除限制在一端的线性表.

② 队列：插入在一端，删除在另一端的线性表

3. DE的变化.

① 串. DE限制为字符.

② 数组. DE关系在维数上的扩充.

③ 广义表、DE可以是带结构的线性表

§3. 循环控制分析.

```
FUN get_linklist (la: linklisttp; i: integer): elemtp;
设置移动指针  p:=la↑.next;  j:=1; 计数变量   {la单链表上存取第i个DE}
    WHILE j<i AND p≠NIL DO   {控制存取第i个DE}
        [ p:=p↑.next;  j:=j+1 ];
    IF j=i AND p≠NIL
        THEN RETURN(p↑.data)
        ELSE RETURN(NULL)
ENDF; {get_linklist}
```

循环控制两方面：

① 防止 $i>$表长　　　$p \neq NIL$

② 控制取第i个DE
③ 防止 $i<1$　　　$j<i$

(1) $p=NIL \wedge j<i$　　空表或 $i>$表长　　出错

(2) $p=NIL \wedge j=i$　　空表 或 $i=$表长$+1$　　出错

(3) $p=NIL \wedge j>i$　　空表 且 $i<1$　　出错

(4) $p \neq NIL \wedge j<i$　　继续循环

(5) $p \neq NIL \wedge j=i$　　找到第i个DE　　出循环

(6) $p \neq NIL \wedge j>i$　　非空表且 $i<1$　　出错

§4. 递归过程.

应用递归过程的主要要求：

1. 递归定义.　　$n! = \begin{cases} 1 & n=0 \\ n*(n-1)! & n>0 \end{cases}$

2. 递归结构

3. 递归比迭代简单

基本原理：重复地将原问题转化为与原问题相似的新问题，直到问题可解为止.

关键. ① 用较简单的问题去表示较复杂的原问题.
② 不能产生自己调用自己的无限序列，必须有一个出口.

```
PROC inorder (bt: bitreptr);
  IF bt ≠ NIL
    THEN [ inorder (bt↑.lchild);
           visite (bt↑.data);
           inorder (bt↑.rchild)
         ]
ENDP;
```

$a*b \begin{cases} a\text{个}b\text{之和} & \sum_{i=1}^{a} b & \text{迭代} \\ (a-1)b+b & & \text{递归} \end{cases}$

```
FUNC  mul1(a,b: integer): integer;          {递归}
    IF  a=0   THEN  RETURN(0)
        ELSE  RETURN(mul1(a-1,b)+b)
ENDF; {mul1}

FUNC  mul2(a,b: integer): integer;          {迭代}
    z:=0
    FOR  i:=1  TO  a  DO  z:=z+b;
    RETURN(z)
ENDF; {mul2}
```

§5. 循环队列空与满的解决方法.

cq.front=cq.rear

1. 用计数变量记载cq中的DE个数.

初始时，变量为0，表示队空，

入队时，变量+1，

出队时，变量-1.

当变量=max为队满

2. 用标志记载使front=rear的情况

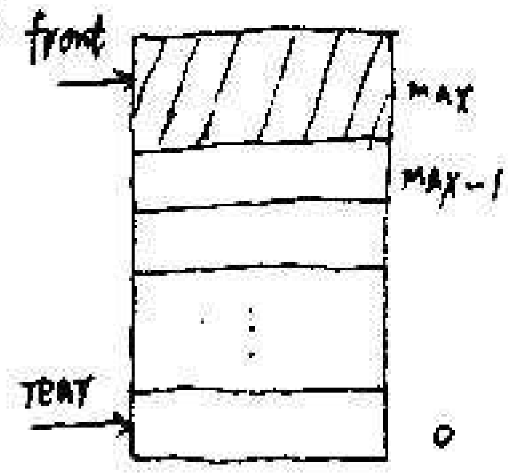
$tag \begin{cases} false & \text{队空} \\ true & \text{队满} \end{cases}$ 初始化时置tag为false

入队

```
IF  tag   THEN  ERROR('队满')
    ELSE [  入队
            IF  front = rear   THEN  tag:=true
```

3. 牺牲一个存贮单元　　以尾指针加1等于头指针作为队列满的标志.

4. 改变指针的定义域



0 .. max-1
⇓
0 .. max

只能通过强制赋值等于此值

rear = max　队满

front = max　队空

初值: rear:=0; front:=max {rear初值也可为其它值}

```
PROC ENCqueue (cq, x)
  IF cq.rear= max THEN ERROR('队满') {入队前先判是否队满}
   ELSE [ IF cq.front =max THEN cq.front := cq.rear; {判是否队空,是,则修改队空标志}
          cq.rear:= (cq.rear + 1) mod max;
   {入队} cq.elem [cq.rear]:= x;
{入队后使得队满,置队满标志} IF cq.rear = cq.front THEN cq.rear:= max
        ]
ENDP;
```

§6. 线性表的倒置

$L=(a_1, a_2, \cdots, a_n) \Rightarrow L'=(a_n, \cdots, a_2, a_1)$

倒置分两大类: ① 数据元素的交换 ② 交换指针

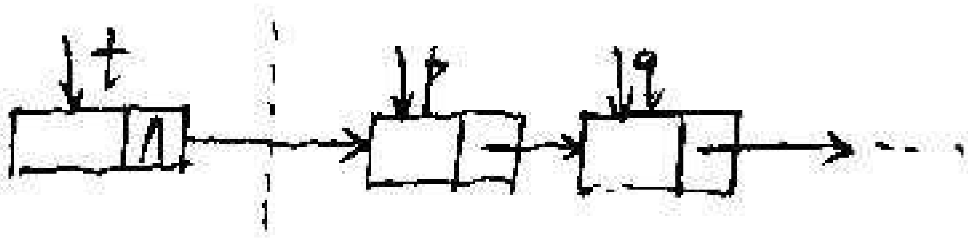
存贮结构 / 倒置方法	顺序	链式			
		单链表	循环	双向	双向循环
交换 DE	1	×	×	2	3
改变链接	×	4	5	6	7

```
PROC invert1 (A);               {顺序存储结构，交换DE}
  FOR i:=1 TO n DIV 2 DO
    [ x:=A[i];
      A[i]:=A[n-i+1];
      A[n-i+1]:=x
    ]
ENDP;
```

```
PROC invert2 (t: dulinklisttp);   {双向链表，交换DE}
    {t为双向链表头指针，其结点结构为 | llink | data | rlink | }
  IF t≠NIL
    THEN [ p:=t;  q:=t;
           WHILE p↑.rlink≠NIL DO p:=p↑.rlink; {前进p指针，使之指向最后一个结点}
           WHILE p≠q AND p↑.rlink≠q DO
             [ x:=q↑.data; q↑.data:=p↑.data;
               p↑.data:=x;
               q:=q↑.rlink; p:=p↑.llink ]
         ]
ENDP;
```

注意结点数为奇数和偶数的情况

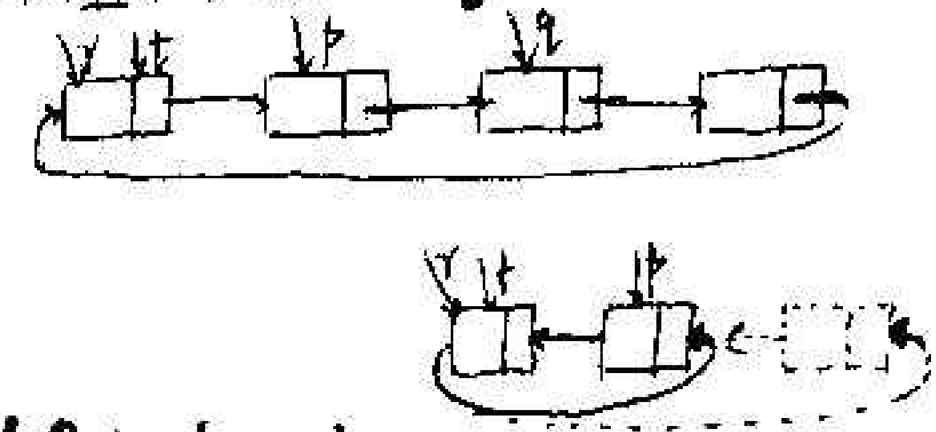
```
PROC invert4 (t);               {t为单链表头指针}
  IF t≠NIL                      {单链表的倒置，改变链接}
    THEN [ p:=t↑.link;
           t↑.link:=NIL;
           WHILE p≠NIL DO
             [ q:=p↑.link;  p↑.link:=t;
               t:=p;  p:=q ]
         ]
ENDP;
```



先断开第一个结点，作为新链表的表尾，再依次从原单链表中摘取结点插入新链表表头！

见上图，仔细体会！

```
PROC inverts (t);        {循环链表的倒置，改变链接}
  IF t≠NIL
   THEN [ p:=t↑.link;  r:=t;
          WHILE p≠r DO
            [ q:=p↑.link;   p↑.link:=t;
              r↑.link:=p;   t:=p;   p:=q ]
        ]
ENDP;
```



先将前两个结点改变链接，生成一个小的倒置的循环链表，再如图中虚线所示依次插入结点，修改指针.

§7. 遍历算法

按一定的规律

对指定的DS中的每一个DE访问且仅访问一次　　访问是抽象术语

访问结果为DE的有序序列.

一. 层次遍历BT

PROC leveltraved (t: bitreptr);　　仔细体会队列的使用!

```
  IF t≠NIL
   THEN [ INIQUEUE(Q);
          ENQUEUE(Q,t);
          WHILE NOT EMPTY(Q) DO
           [ p:=DLQUEUE(Q);
             write(p↑.data);
             IF p↑.lchild≠NIL THEN ENQUEUE(Q,p↑.lchild)
             IF p↑.rchild≠NIL THEN ENQUEUE(Q,p↑.rchild)
           ]
        ]
ENDP;
```

二. BT中所有结点左、右子树交换.

```
PROC exchange (t);
  IF t≠NIL
    THEN [ s:=t↑.lchild ; t↑.lchild:=t↑.rchild;
           t↑.rchild:=s;
           exchange (t↑.lchild);
           exchange (t↑.rchild)
         ]
ENDP;
```

三. 输出BT中的所有叶结点.

```
PROC outleaf (t);
  IF t≠NIL
    THEN [ IF t↑.lchild=NIL AND t↑.rchild=NIL
             THEN write (t↑.data);
           outleaf (t↑.lchild);
           outleaf (t↑.rchild).
         ]
ENDP;
```

四. 线索化BT P131 算法6.5.6.6

lchild	ltag	data	rtag	rchild

结点结构

$$ltag=\begin{cases}0 & \text{指针}\\ 1 & \text{指向前驱的线索}\end{cases}$$

$$rtag=\begin{cases}1 & \text{指向后继的线索}\\ 0 & \text{指针}\end{cases}$$

五. 线索BT的遍历

① 找到遍历访问的第一个结点.

② 访问该结点、并寻找其后继

③ 重复②, 直到遍历结束