数 据 结 构

考试章节：1. 2. 3. 5. 6. 7. 9. 10 带*不考

各章要求.

数材结构： 三种数据结构：线性结构，树，图

(1) 定义，基本概念

(2) 基本操作.

(3) 存贮结构（顺式，链式）

(4) 应用

ch0p1： 算法概念及时间复杂度分析

1. 算法及其特征(P12) 5个特征！①有穷性 ②确定性 ③可行性 ④输入 ⑤输出

2. 时间复杂度分析. $T(n)=O(f(n))$

常数阶 $O(1)$

线性阶 $O(n)$

平方阶 $O(n^2)$

ch0p2. 线性表的存贮结构及基本运算.

链式存贮结构！

ch0p3. 队列，栈的存贮结构及基本运算

假溢出！

ch0p5. 广义表存贮结构及基本运算.

含数组为X

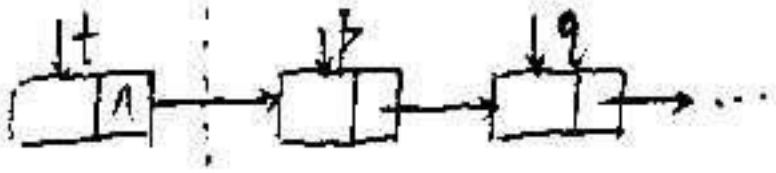
WHILE p≠? AND p↑.rlink↑.llink {注意结点数为奇数和偶数的情况}
[t:=t↑.data; t↑.data:=p↑.data; p↑.data:=t;
p:=p↑.llink; q:=q↑.rlink]

ENDP;

例3: PROC invert3(t?);
改为: p:=t↑.llink

例4: PROC invert4 (t: 单链表); {单链表的倒置,改变链接}
IF t≠NIL THEN
[p:=t↑.link;
t↑.link=NIL;
WHILE p≠NIL DO
[q:=p↑.link; p↑.link:=t; t:=p; p:=q]]
ENDP;

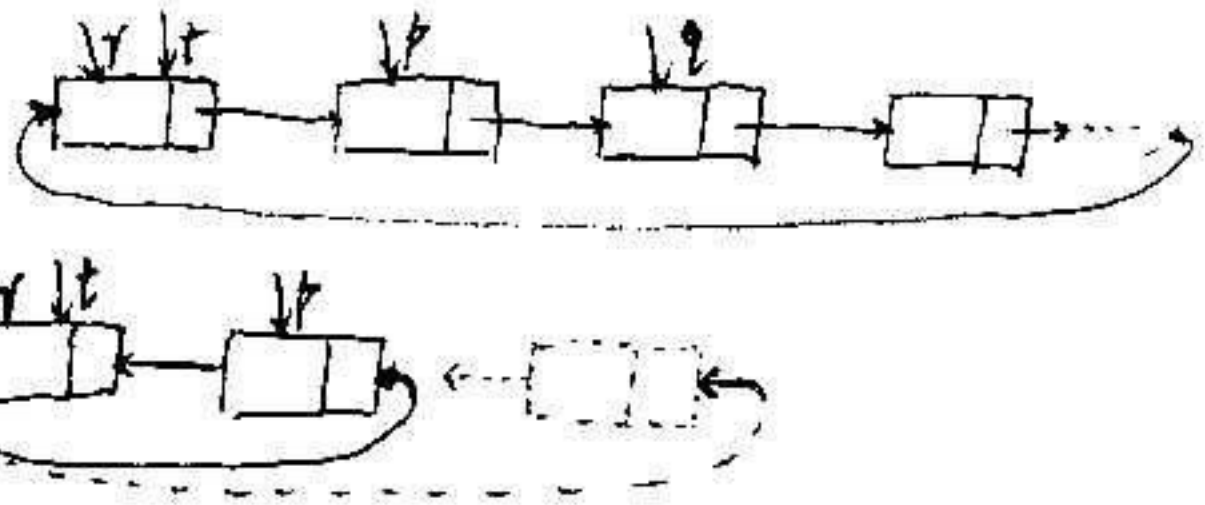
先断开第一个结点,作为新链表的表尾



再依次从原链表中摘取结点,插入新链表表头!

例5: PROC invert5 (t: 循环单链表); {循环链表的倒置,改变链接}
IF t≠NIL THEN
[p:=t↑.link; r:=t;

```
while   p≠r     DO
  [ t:=p^.link;  p^.link:=r;
    r:=p;  p:=q
  ]
  r↑.link:=t
]
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。

九. 遍历算法

1. 概念

访问是抽象术语　　访问结果为此的有序序列

按一定的规律 对指定DS中的每一个此 访问且仅访问一次的过程

2. 层次遍历BT

```
PROC  levelTravel(t);        仔细体会队列的使用!
  IF t≠NIL THEN
    [ INQUEUE(Q);
      ENQUEUE(Q, t);
      while not empty(Q) DO
```

```
    [ p:= PDEQUEUE (Q);
      write (p↑.data);
      IF p↑.lchild ≠ NIL  THEN ENQUEUE (Q, p↑.lchild);
      IF p↑.rchild ≠ NIL  THEN ENQUEUE (Q, p↑.rchild)
    ]
  ]
ENDP;
```

3. 语句具体化例子.

例1: 完成将BT中所有结点左、右子树交换的功能.

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

例2: 输出BT中所有叶结点.

```
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; ]
```

3. BT的线索化 (P131 算法6.5 6.6)

lchild	ltag	data	rtag	rchild

结点结构

ltag = { 0 指针; 1 指向前驱的线索 }

rtag = { 1 指向后继的线索; 0 指针 }

4. 线索BT的遍历

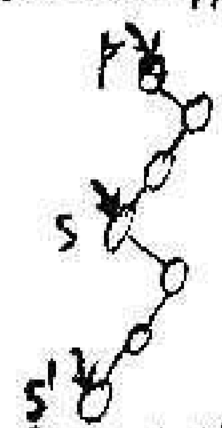
① 定位应访问的第一个结点.

② 依次找结点的后继, 进行访问.

(1) 中序线索BT

(1) 找后继. 对任一结点p. 若p↑.rtag=1 则 rchild域指示该结点后继.
若p↑.rtag=0. 应从右孩子开始, 沿左链域前进, 直到找到没有左孩子的结点s, (即s↑.ltag=1) s为p的后继. ~~[右子树中最左下的结点]~~

```
FUNC inorder_next(p, t: thlinktp): thlinktp;  {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 inorder_thrt(t: thlinktp);  {中序线索BT遍历算法}
  IF t↑.lchild≠t          {BT为空时 t↑.lchild=t}
  THEN
    [ p:=t;
```

```
    WHILE p^.child<>nil DO p:=p^.child; {找到第一个结点}
    WHILE p#nil DO
     [visit(p^.data);
      p:=InNext(T,p)
     ]
   ]
ENDP;
```

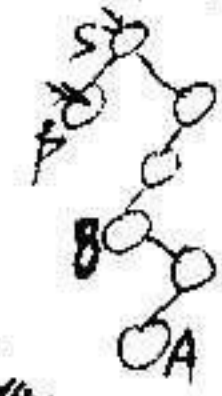
(2) 后序线索BT

对任一结点p: { p为根, 则p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(T,p:BiTree):BiTree; {后序线索BT寻找后继算法}
  s:=PARENT(T,p);
  IF s=NIL THEN RETURN(NIL);
  IF p=s^.rchild OR s^.rtag=1 {s无右孩子}
     THEN RETURN(s);
  WHILE s^.rtag=0 DO
  [ s:=s^.rchild; {p为右兄弟开始, 沿左链域前进, 直到某结点s, 无左孩子}
    WHILE s^.ltag=0 DO s:=s^.lchild
  ]
  RETURN(s)
ENDP;
```

```
s:=ST.rchild
WHILE ST.lchild<>p
DO s:=ST.lchild
```

例如A结点,



到B结点后, 只要跳出了内层while循环, 并未跳出外层while循环, 直到A结点, 才跳出

```
PROC  postthr(t:tp);                    {后序线索BT遍历算法;
  IF t^.lchild<>t  THEN                 {t为空时. t^.child=t}
  [p:=t; search:=true;
    WHILE search DO  [仔细体会 search 在循环中的用法]
     [WHILE p^.ltag=0 DO p:=p^.lchild;
       IF p^.rtag=0 THEN p:=p^.rchild
                    ELSE search:=false;
     ] [定位到第一个结点]
    WHILE p<>t DO
     [visit(p^.data);
       p:=postnext(p,t)
     ]
  ]
ENDP;
```

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

(3)先序线索BT寻找后继. 找p的后继. {若p^.ltag=0 则后继为p^.lchild
                                  否则, 后继为 p^.rchild

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

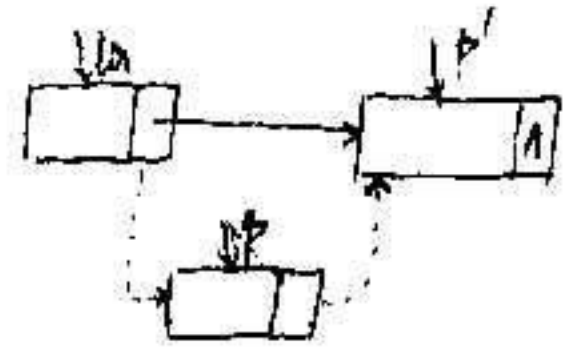

ENDP;

十. 存贮结构转换算法.

1. 线性表 顺序 ⇌ 链式.

例: 顺序 → 单链表

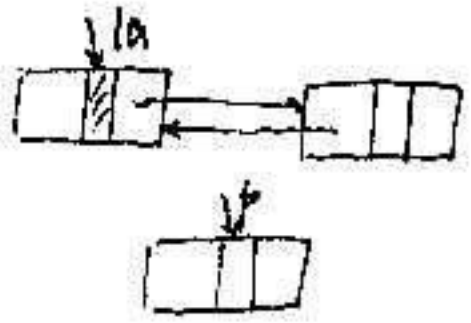
```
PROC  V'(a, V, LA);
  new(la); la^.next:=NIL;          {产生空表}
   FOR i:=V.last  DOWNTO 1  DO     {循环控制}. 依指入方便
    [new(p); p^.data:=V.[i];       {装填数据}
      p^.next:=la^.next;
      la^.next:=p                  {从表尾到表头建立单链表}
    ]
ENDP;
```



在表头插入!

例2. 顺→循环链表 {只需将上一算法中的 la↑.next:=NIL 修改为 la↑.next:=la 即可}

```
PROC STOClink(V; la);
  new(la). la↑.next:=la;  {产生空循环链表}
  FOR i:=V.last DOWNTO 1 DO
    [new(p); p↑.data:=V.[i]; {生成结点,并装填数据}
      p↑.next:=la↑.next;
      la↑.next:=p
    ]
ENDP;
```



例3. 顺→双向链表 {需要修改上述算法的初始化和插入部分}

```
PROC STODlink(V; la);
  new(la), la↑.next:=la↑.prior:=NIL; {产生双向链表}
  FOR i:=V.last DOWNTO 1 DO
    [new(p); p↑.data:=V.[i]; {生成结点,并装填数据}
      IF la↑.next≠NIL THEN
      [la↑.next↑.prior:=p;]
      p↑.next:=la↑.next;
      la↑.next:=p;
      p↑.prior:=la;
    ]
ENDP;
```

{插入第一个结点只需三条指针修改语句}

要点:
① 建立空链表

④ FOR i:=v.last+ DO v.[i]:=

③ 生成结点，装填数据

④ 插入链表中

例5. 链表 → 顺

```
PROC linkTOS (V; la?);
    p:=la^.next;                    {带头结点的单链表}
    i:=1; V.last:=0;                {设置移动指针，动数组下标}
    WHILE p#NIL DO
    [ V.[i]:=p^.data; i:=i+1;
      V.last:=V.last+1;
      p:=p^.next
    ]
ENDP;
```

例6. BT的顺 → 链式

A[1:n] → bt

```
PROC A TO bt(A; bt);
  bt:=NIL;
  IF A[1]≠0 THEN
  [ new(bt); bt^.data:=A[1];
    INIQUEUE(Q);
    ENQUEUE(Q, 1, bt);
    WHILE NOT EMPTY(Q) DO
    [ DELQUEUE(Q, k, p);
      IF A[2k]≠0 THEN
```

```
[ NEW(P);
  P↑.data:=A[2k];
  P↑.lchild:=q; ENQUEUE(A,2k,q)
]
ELSE P↑.lchild:=NIL;
IF A[2k+1]≠0 THEN
  [ NEW(q); q↑.data:=A[2k+1];
    P↑.rchild:=q; ENQUEUE(A,2k+1,q)
  ]
  ELSE P↑.rchild:=NIL
]
ENDP;
```

例7. 链式→顺序　　A[1..n] ← bt

```
PROC btTOA main(A bt);
  i:=1;
  FOR j:=1 TO n DO V[j]:=0;
  btTOA(A bt,i)
ENDP;

PROC btTOA(A, bt, i);
  IF bt≠NIL THEN [ V[i]:=bt↑.data;
                   btTOA(A,bt↑.lchild,2i);
                   btTOA(A,bt↑.rchild,2i+1);
                 ]
ENDP;
```

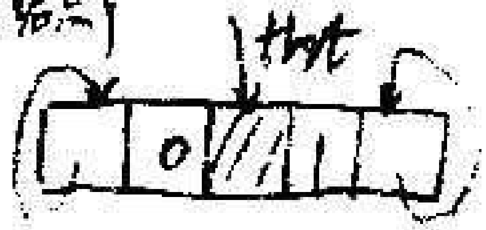
二叉树之线索树的转换: [见P131 算法6.5.66]

1) 结点结构. 线索树头结点结构 [见P130]

2) 存储结构.

3) 链接关系.

例8. 二叉树 → 线索二叉树




```
PROC BitreeTOInth(thrt, bt); {中序遍历生成中序线索链表}
  NEW(thrt); thrt↑.ltag:=0; thrt↑.rtag:=1; {建头结点}
  thrt↑.lchild:=thrt↑.rchild:=thrt;
    IF bt≠NIL THEN
                                    {指针pre始终指向刚刚访问过的结点}
    [ thrt↑.lchild:=bt; pre:=thrt; inthread(bt); {中序遍历线索化}
      pre↑.rchild:=thrt; pre↑.rtag:=1; {最后一个结点线索化}
      thrt↑.rchild:=pre {使头结点指向访问的最后一个结点}
    ]
ENDP;

PROC inthread(p); {中序线索化以p为根指针的二叉树}
  IF p≠NIL THEN
   [ inthread(p↑.lchild); {左子树线索化} {递归调用,直到遇见左子树为空的结点}
     visit(p↑.data); → 线化!
     IF p↑.lchild=NIL THEN
```

chap 6. 二叉树 线索二叉树存储结构 遍历算法

6-4 ~ 6-8 X

* 二叉树由性质
树的遍历

线索二叉树!!

chap 7. 图的存贮结构及遍历算法

7-1, 7-6, 7-7 X (图的应用不需要深入)

chap 9. 二叉排序树与 Hash

定义. 特点

Hash!

chap 10. 各种内排序算法思想. 时间特性及稳定性概念.

一. 学习(复习)思路.

定义及特性.

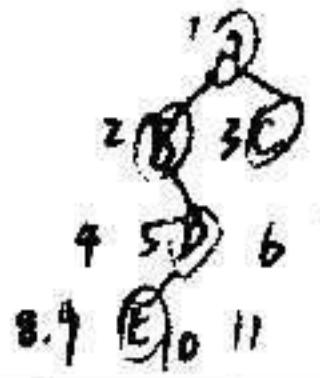
存贮结构.

常用运算及算法.

算法分析. 应用举例

二. 三大类存贮结构

1. 顺序存贮结构 { 地址连续 / 按一定顺序存放元



层序编号

{ 双亲 [i/2]
左孩子 2i
右孩子 2i+1

[p↑.itag := 1; p↑.ivalue := p] { 建立前驱线索 }

ELSE p↑.itag := 0;

IF pre↑.rvalue = NIL THEN

[pre↑.rtag := 1; pre↑.rvalue := p] { 建立右线线索 }

ELSE pre↑.rtag := 0;

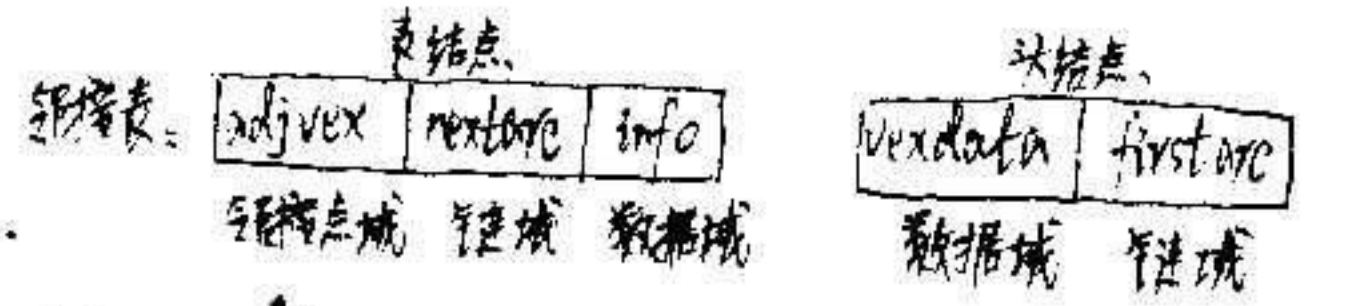
pre := p; { 保持pre指向p的前驱 }

InThread(p↑.rvalue); { 对树线索化 }

]

ENDP;

例10. 邻接表 → 邻接矩阵.



PROC AdList TO Admatrix (A; adlist);

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; { 即适用于有向图, 也适用于无向图 }

例11. 邻接矩阵 → 邻接表.

PROC AdMatrix TO Adlist (A, adlist);

```
FOR i:=1 TO n DO adjlist[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:=adjlist[i].firstarc;
         adjlist[i].firstarc:=p          {从表尾到表头的插入}
        ]
ENDP;
```

十一. 各类二叉树的概念及特点.

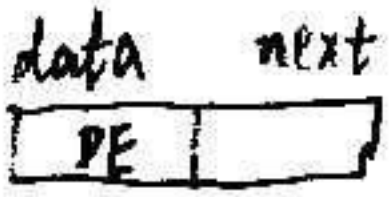
1. 满BT
2. 完全BT
3. 平衡BT (AVL)
4. 最佳二叉树 BST
5. 堆.
6. 哈夫曼树 (最优 BT)

2000年数据结构答案

一. 简答题

2. 链式存贮结构

结点结构



静态链表不要求!

3. Hash 方法.

三种存贮结构区别.

静态用地址相邻来映照的逻辑关系.

链式是借助于指针来建立 DE之间的联系.

Hash 是得到值本身与自贮地址通过 Hash 函数建立直接联系.

三. 算法和打.

算法: 完成特定功能的有限指令集 ①有穷性 ②确定性 ③可行性 ④输入 ⑤输出

算法、问题、程序.

① 算法是有穷的. 程序并不一定是有穷的.

② 表示方法. 程序可以用高级语言描述. 算法不仅可用高级语言. 也可用框图.

2. 时间复杂度 $T(n)=O(f(n))$

n: 计算量. 问题规模

f(n): n增大时 算法运行时间的增长率

O(f(n)): ……

时间复杂度分析方法 「 计算法，三语句频度 时间同阶 去掉系数
                    「 估计法：(见书1.2.3 算法0-2) (n, n², n)

语句频度：语句可能重复执行的最大次数.

四. 线性表变型

顺序存储结构特点：

1. 逻辑相邻由于 物理上也相邻. ✓
2. 可随机存取任一元素.
3. 事先预留最大可能的存储空间 (静态存储结构) 缺
4. 插删时要移动大量的元素

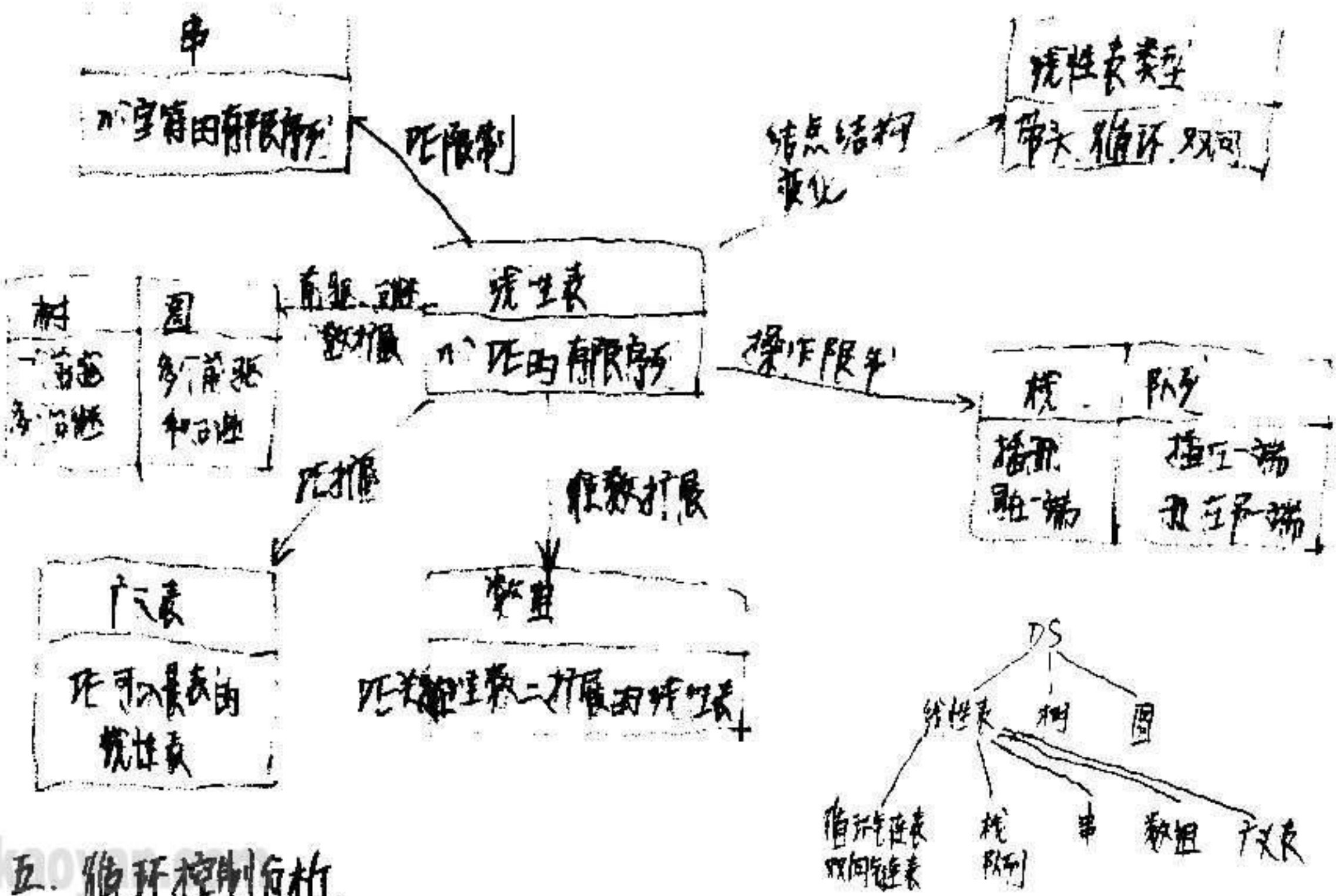
链式特点：

1. 物理一定相邻 优
2. 顺序存取
3. √一种动态存储结构
4. √只需修改指针，不需移动元素

1. 结点结构变化 { 带头结点的链表
                  循环链表
                  双向链表

2. 限制 { ① 操作 { 栈 插入删除限制在一端的线性表
                  队列 插入在一端，删除在另一端的线性表
         ② 数据元素 → 串，n个字符由有限序列 元素限制为字符

3. 扩展 { ① 堆数 数组：元素在堆数上的扩充
         ② 数据元素：由扩展，广义表：元素可以是带结构的线性表
         ③ 树 } 在直接前驱和直接后继个数上的扩展
         ④ 图 }



五. 循环控制分析.

GET(L, i) 1≤i≤表长

FUNC GetLinkList(la: LinkList; i: integer): ElemType;

[置初值] p:=la^.next; j:=1 计数变量 {la链表上查取第i个DE}

设置移动指针

While (循环控制条件) Do [循环体];

If (出循环判断条件) Then Return (p^.data)

ELSE return (NULL)

ENDF;

循环条件: p≠NIL AND j<i 循环体: p:=p^.next; j:=j+1

出循环判断条件：

①	p=NIL	∧ j<i	空表∧记i；或记表长+	异常 出循环
②	p=NIL	∧ j=i	空表∧记i；或记=表长+	异常 出循环
③	p=NIL	∧ j>i	空表∧记<	异常 出循环
④	p≠NIL	∧ j<i	继续循环	
⑤	p≠NIL	∧ j=i	确定了第i个DE	正常 出循环 ✓
⑥	p≠NIL	∧ j>i	记<	异常 出循环

六、递归过程.

1. 递归定义.

$$n! = \begin{cases} 1 & n=1 \\ n\cdot(n-1)! & n>1 \end{cases}$$

2. 递归结构.

3. 递归被迭代更简单.

例：计算两个非负整数 a+b 的算法.

迭代法.

```
FUNC a,a,b: integer): integer;
  [a×b = a个b之和a]
  z:=0;
  FOR i:=1 to a DO z:=z+b;
  RETURN (z)
ENDF;
```

递归法：

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

递归基本原理：能把把问题转化为与原问题相似的新问题，直到解决。

关键点：用较简单的新问题表示较复杂的原问题。

① 必须有递归出口。

任何一个递归算法都可用一个非递归算法来实现；但反之不成立。

七. 循环队列 满与空的解决方法。

方法1：用计数变量记录队列元素个数

初始化 C=0 {表示队空}

入队时，{ 先 C=max ? {C=max 表示队满}
{ C:=C+1;

出队时 { 先 C=0 ?
{ C:=C-1;

方法2：设一个标志位来区别空与满。用标志记载使 front=rear 的情况

初始化队列时：front=rear; tag:=false; {标志-队空}

入队时 使 front=rear 且 tag:=true;

出队时 使 tag:=false

{ front=rear ∧ tag=false 队空
{ front=rear ∧ tag=true 队满
{ tag=false ∧ front≠rear 非空非满

方法3：牺牲一个元素空间 以尾指针加1等于头指针作为队列满的标志。

入队前：先求 rear+1 = front (mod max) 是否满

其它处理均不变。 rear=front 队空。

方法4：扩大 rear 和 front 的实域为 0..maxsize (原=0..maxsize-1)

{ front = maxsize 队空
{ rear = maxsize 队满

队满：若 rear = maxsize-1 则队满

队空：若 rear = front 则 rear := maxsize

若 front = maxsize，则 front := maxsize-1

几. 线性表的倒置算法

算法 \ 存储	顺序	链式			
		单链表	循环	XX向	XX向循环
交换元素	①	X	X	②	③
改变链接	X	④	⑤	⑥	⑦

例1: PROC inverse(A: Array of [..n]); {顺序存储结构，交换PE}
FOR i:= 1 TO n DIV 2 DO
[x:=A[i]; A[i]:=A[n+1-i]; A[n+1-i]:=x]
ENDP;

例2: PROC invert2(t: dulinklistp); {双向链表，交换PE}
IF t≠NIL THEN
[p:=t; while p↑.rlink≠NIL DO p:=p↑.rlink; {前进p指针，使之指向最后一个结点}
q:=t;