Ch2　线性表

- 线性结构特征

在数据元素的非空有限集中:

- 存在唯一的一个被称作"第一个"的数据元素
- 存在唯一的一个被称作"最后一个"的数据元素
- 除第一个外, 集合中的每个数据元素均只有一个(直接)前趋(predecessor)
- 除最后一个外, 集合中每个数据元素均只有一个(直接)后继(successor)

- 线性表

定义: 一个线性表是n (n≥0)个数据元素的有限*序列*, 记作

$(a_1, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$

如　英文字母表(A, B, C, … , Z)是一个线性表、一周中的7天(…)、the website navbar menu(…).

又如

学号	姓名	年龄
0001	张三	18
0002	李四	19
…	…	…

an element

- 线性表的逻辑结构

特征
- 表中数据元素的个数n – 表长度, n = 0 时为空表
- 1<i<n时, < a_{i-1}, a_i > – a_i的前趋是a_{i-1}, a_1无前趋
  – a_i的后继是a_{i+1}, a_n无后继
- 数据元素同构, 且不能出现缺项

线性表的ADT定义 P19

- 线性表的顺序存储结构 -- 顺序表

顺序表 用一组地址空间连续的(contiguous)存储单元存放的一个线性表

元素地址计算方法

– $LOC(a_i) = LOC(a_1) + (i-1)*L$

– $LOC(a_{i+1}) = LOC(a_i) + L$

其中: L指一个数据元素占用的存储单元个数

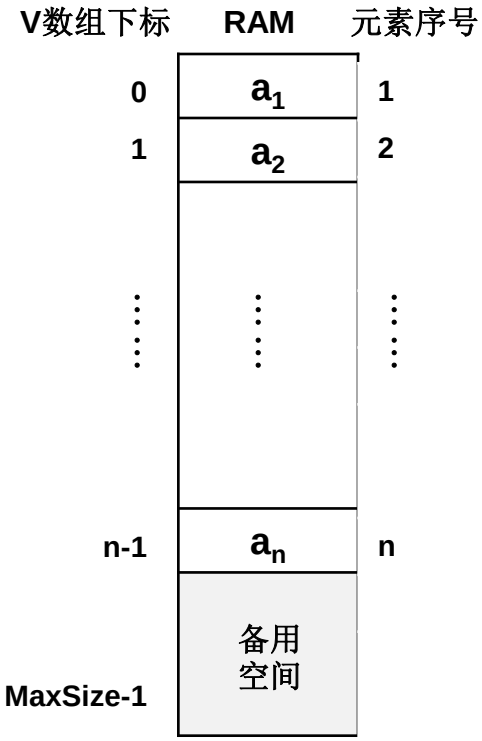
$LOC(a_1)$称为基地址

$LOC(a_i)$为线性表第i个数据元素的存储地址

特点
- 逻辑上相邻的数据元素, 其物理地址亦相邻
- 随机存取

表示 用C/C++语言中的一维数组(array)

- 线性表的顺序存储结构 -- 顺序表



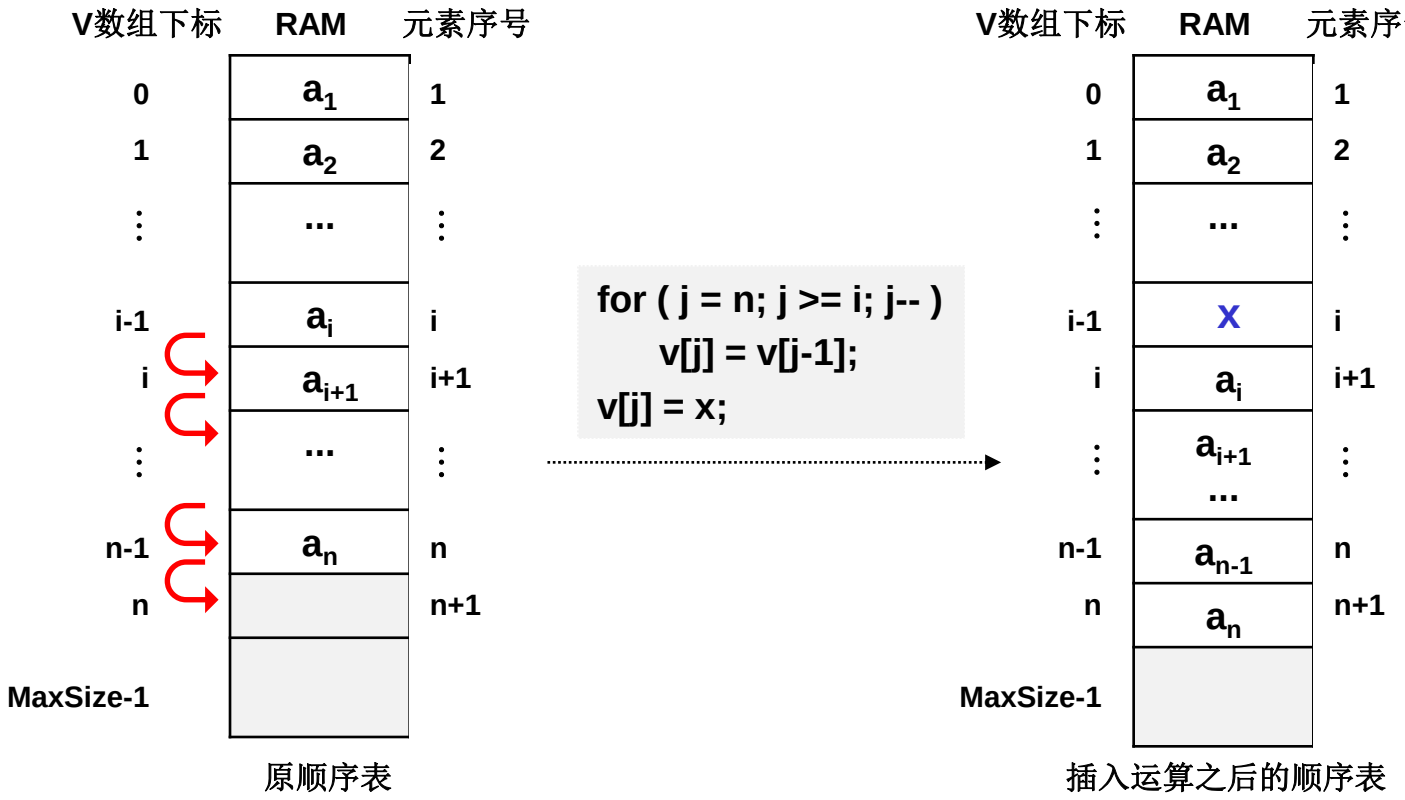

```
#define MaxSize 1000 //依问题域而定
typedef char ElemType
typedef struct list {
    int length;  //表中当前元素个数
    ElemType elem[MaxSize];
 } SeqList;
```

Note: Distinction between an array and a (contiguous) list:
First, a list whose size is variable, while an array whose size is constant (whose size is fixed when the program is complied).
Second, a list is a dynamic data structure (because its size can change), while an array is a static data structure(it has a fixed size).
A list and an array related in that a list of variable size can be implemented in an array of fixed size.

顺序表的基本运算/操作:

插入运算

- 定义 线性表(这里指顺序表)的插入运算是在第i(1≤ i ≤ n+1)个元素之前插入一个新的数据元素x, 使长度为n的线性表(a_1 , ⋯ , a_{i-1} , a_i , a_{i+1} , ⋯ , a_n)变成长度为n+1的线性表(a_1 , ⋯ , a_{i-1} , x , a_i , a_{i+1} , ⋯ , a_n).
- 算法思路及其描述: 将第i至第n共(n-i+1)个数据元素依次后移



顺序表的基本运算/操作:

插入运算

- 算法思路及其描述 **P24/A.2.4** ➜

```
for ( j = n; j >= i; j-- ) v[j] = v[j-1];
v[j] = x;
```

```
#define MaxSize 1000 //依问题域而定
typedef char ElemType
typedef struct list {
    int length;  //表中当前元素个数
    ElemType elem[MaxSize];
 } SeqList;
```

```
//在顺序表L中的第i个元素之前插入一个数据元素x
void InsertSeqList ( SeqList L, int i, ElemType x ) {
  if ( L.length >= MaxSize || i > L.length + 1 || i < 1) return;
  //表已满无法进行插入运算 或 i所指示的插入位置有误
  if ( i == L.length + 1 ) {  //在L的最后一个元素之后插入元素x
      L.elem[i] = x; L.length++;  return;  }
  for ( j = L.length; j >= i; j-- )  L.elem[j] = L.elem[j-1];
  //准备在第 i 个元素之前插入元素x, 这时需将第 i 个元素至
  //表尾元素(即第 L.length 个元素)共 length - i + 1 个数据元素依次后移
  L.elem[j] = x;  //插入元素x
  L.length++;
}
```

顺序表的基本运算/操作:

插入运算

- 算法时间复杂性T(n)分析

3 cases to analyze an algorithm in asymptotic notation: Worst Case, Average Case and Best Case.

平均时间复杂性: 用在所有可能情况下执行次数的加权平均值来表示.

设P_i是在第i个元素之前插入一个元素的概率, 则在长度为n的线性表中插入一个元素时, 所需移动的元素次数的平均次数为:

$$M(n) = \sum_{i=1}^{n+1} P_i(n-i+1)$$

假设权重: $P_i = \frac{1}{n+1}$

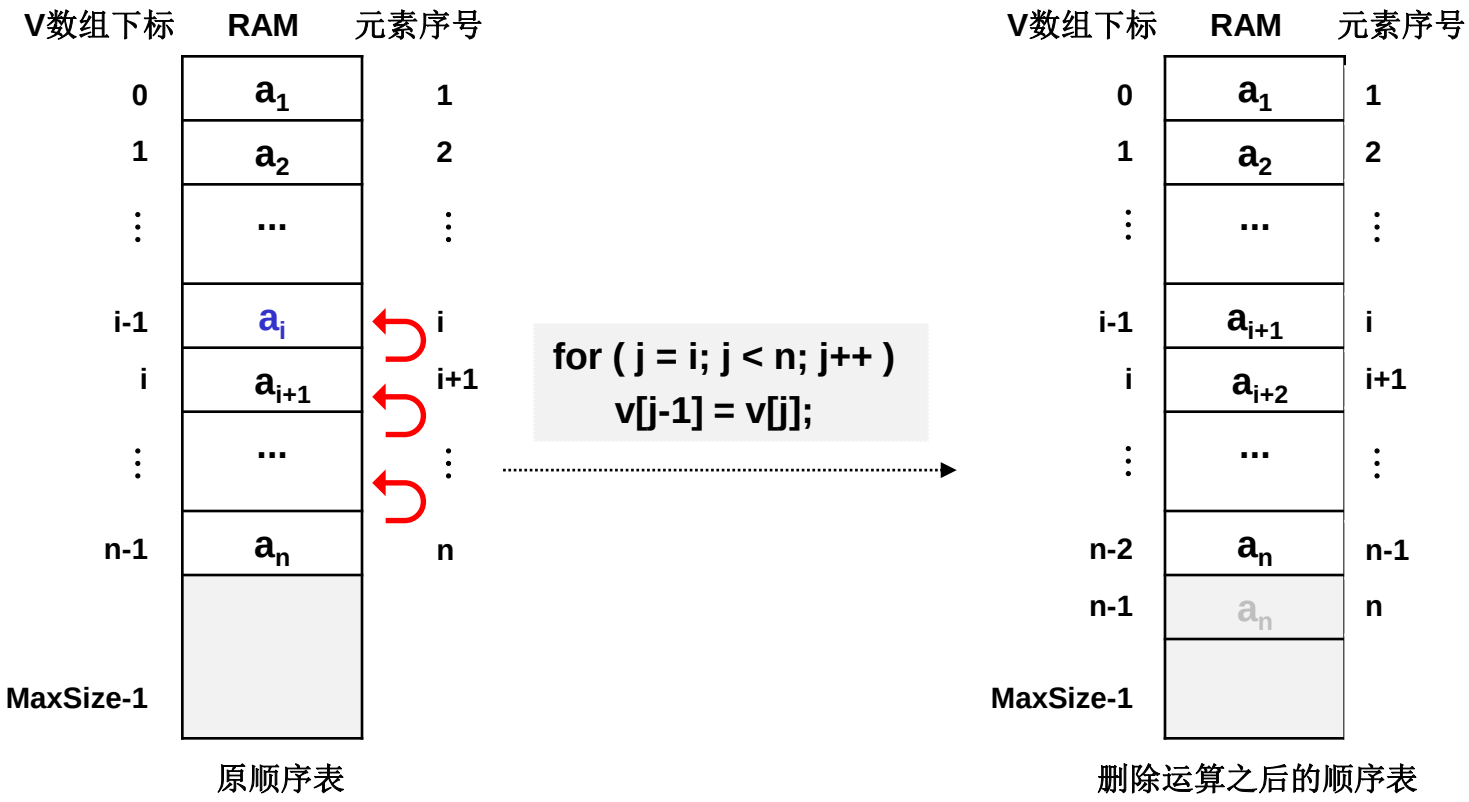
则 $M(n) = \frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1) = \frac{n}{2}$

$\therefore T(n) = O(n)$

顺序表的基本运算/操作:

删除运算

- 定义 线性表(顺序表)的删除是将第i(1≤ i ≤ n)个元素删除, 使长度为n的线性表(a_1, …, a_{i-1}, a_i, a_{i+1}, …, a_n)变成长度为n-1的线性表(a_1, …, a_{i-1}, a_{i+1}, …, a_n).
- 算法思路及其描述: 将第i+1至第n共(n-i)个数据元素前移



顺序表的基本运算/操作:

删除运算

- 算法思路及其描述 **P24/A.2.5** ➜ `for ( j = i; j < n; j++ ) v[j-1] = v[j];`

```
//将顺序表L中的第i个元素删除
void DeleteSeqList ( SeqList L, int i ) {
  if ( i > L.length || i < 1) return;  //i所指示的删除位置有误
  for ( j = i; j <= L.length; j++ )  L.elem[j-1] = L.elem[j];
  //将第 i + 1个元素至表尾元素(即第 L.length 个元素)共 length - i 个数
  //据元素依次前移
  L.length --;
}
```

顺序表的基本运算/操作:

删除运算

- 算法时间复杂性T(n)分析

设Q_i是删除第i个元素的概率, 则在长度为n的线性表中删除一个元素所需移动的元素次数的平均次数为:

$$M(n) = \sum_{i=1}^{n} P_i(n-i)$$

假设权重: $P_i = \frac{1}{n}$

则 $M(n) = \frac{1}{n}\sum_{i=1}^{n}(n-i) = \frac{n-1}{2}$

$\therefore\ T(n) = O(n)$

顺序存储结构的优缺点：

优点 **1)** 可随机存取表中任一数据元素

2) 操作直观

缺点 **1)** 插入、删除运算需要移动大量的元素**(on average: about n/2)**

2) 预先分配空间需按最大空间分配，**RAM**利用不充分

- 线性表的链式存储结构 --- 单链表、循环链表、双向链表

特征
- 用一组地址离散的(disjoint)存储单元存储线性表的数据元素
- 利用指针实现了用不相邻的存储单元存放逻辑上相邻的数据元素
- 每个数据元素a_i, 除存储本身信息外, 还需存储其后继的信息, 这时的数据元素可看作"结点", 即:

结点

➔数据域: 数据元素自身信息

➔指针域: 指示其后继结点的存储位置

例如

单链表 结点中只含一个指针域的线性链表叫~

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

设 Node *head, *p;
或 LinkedList head, p;

p 结点 (*p)

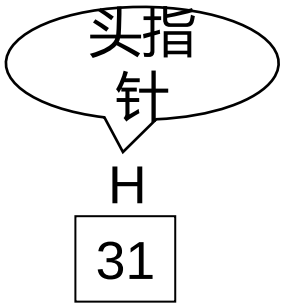
(*p)表示p所指向的结点
(*p).data⇔p->data表示p指向结点的数据域
(*p).next⇔p->next表示p指向结点的指针域

生成一个Node类型的新结点： p = (Node *)malloc(sizeof(Node));
系统回收p结点： free(p);

(单)链表中的头结点 ➔

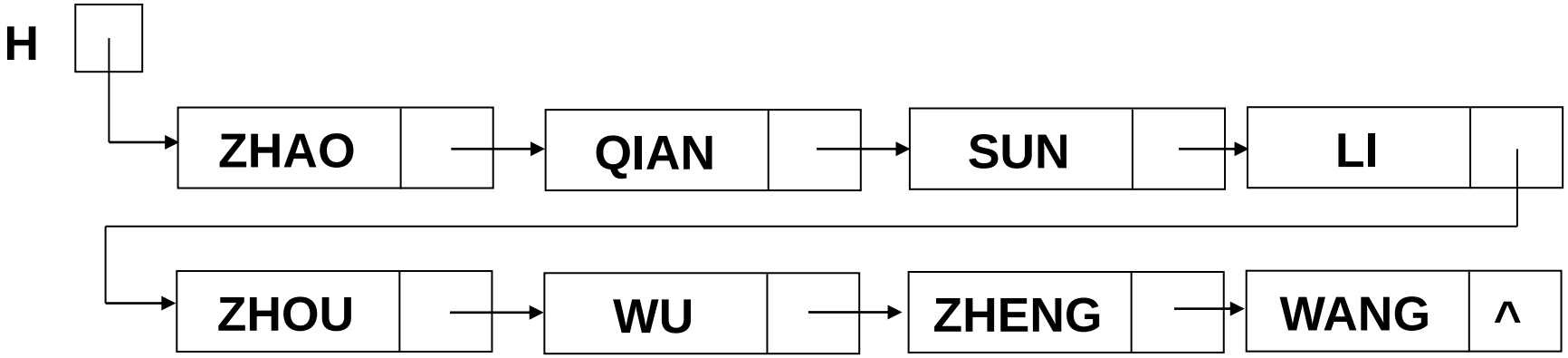
- 线性表的链式存储结构 --- 单链表、循环链表、双向链表

例 线性表 (ZHAO, QIAN, SUN, LI, ZHOU, WU, ZHENG, WANG)



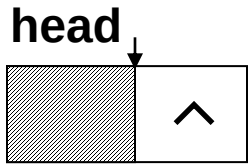
头指针通常是一个静态变量，它指示链表中第一个结点的存储地址，因此链表可以用头指针的名字来命名

存储地址	数据域	指针域
1	LI	43
7	QIAN	13
13	SUN	1
19	WANG	NULL
25	WU	37
31	ZHAO	7
37	ZHENG	19
43	ZHOU	25



(单)链表中的头结点

头结点是在(单)链表的第一个结点之前附设的一个结点。头结点的指针域(head->next)指向(单)链表的第一个数据元素结点，若(单)链表为空，则head->next=Null. [头结点作用: 空表、便于运算、head->data = ?/附加信息]



空表

```
Node *head;  或 LinkedList head;
//head是指向链表中第一个结点(or 头结点)的头指针
```

单链表的运算 查找、插入、删除、建立

1.查找运算 查找单链表中是否存在结点x, 若有则返回指向x结点的指针; 否则返回Null.

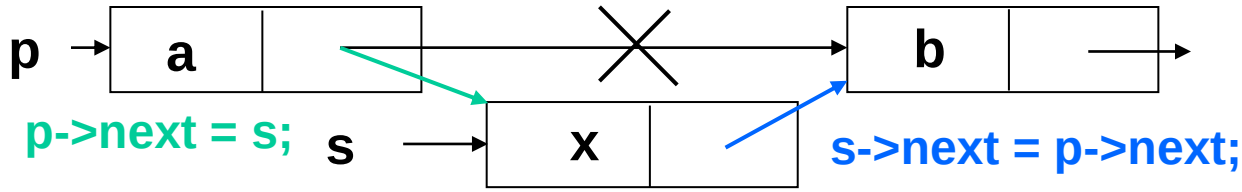
算法描述 类似P29/A.2.8

```
Node  *GetElem ( Node *head, ElemType x ) {
    p = head->next;
    while ( p && p->data != x )  p = p->next;
    return(p);  }
```

算法分析 $T(n)=O(n)$

单链表的运算 查找、插入、删除、建立

2.插入运算 在线性链表的任意两个数据元素a和b间插入x, 已知p指向a.



s = (Node *)malloc(sizeof(Node));

算法描述 类似**P30/A.2.9**

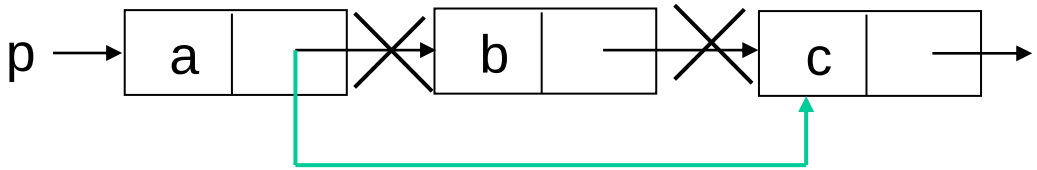
```
void InsertLinkedList(Node *p, ElemType x) {
    s = (Node *)malloc(sizeof(Node));
    s->data = x;
    s->next = p->next;
    p->next = s;  //若该语句和上一条语句次序互换, 则出错
}
```

算法分析 $T(n)=O(1)$

A node can be added in four ways:
①At the front of the Linked List ②After a given node
③At the end of the Linked List ④Before a given node

单链表的运算 查找、插入、删除、建立

3.删除运算 在已知**p**指向**a**, 删除结点**a**的后继结点**b.**



p->next = p->next->next;

算法描述 类似**P30/A.2.10**

```
void DeleteLinkedList(Node *p) {
    if (p) { q = p->next;
             p->next = q->next;
             free(q); } }
```

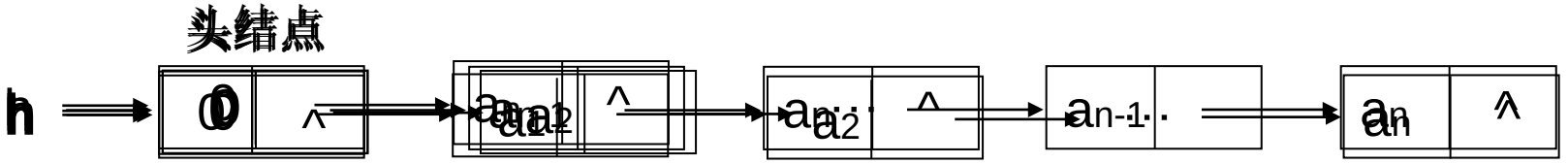
算法分析 $T(n)=O(1)$

单链表的运算　查找、插入、删除、建立

4. 动态创建单链表　设线性表n个元素已存放在数组a中, 建立一个单链表, head为指向其头结点的头指针.

算法描述　类似P30-31/A.2.11

```
Node *CreateLinkedList(ElemType a[ ], int n) {
    head = (Node *)malloc(sizeof(Node));
    head->next = NULL;
    for ( i = n; i > 0; i-- ) {
        p = (Node *)malloc(sizeof(Node));
        p->data = a[i-1];
        p->next = head->next;
        head->next = p; }
    return  head; } //顺序表->单链表
```



算法分析　$T(n)=O(n)$

单链表的优缺点

- 它是一种动态存储结构, 不需预先分配空间
- 插入、删除运算无需移动大量的数据元素, 节省了时间开销
- 指针占用额外存储空间(特别当元素本身信息量很少时不划算)
- 顺序存取数据元素(不能随机存取), 查找速度慢

In summary, we can conclude:

Contiguous storage is generally preferable

when the records are individually very small;

when the size of list is known when program is written;

when few insertions or deletions need to be made except at the end of the list; and

when random access is important.

Linked storage proves superior

when the records are large;

(the pointers themselves take some space)

when the size of the list is not known in advance; and

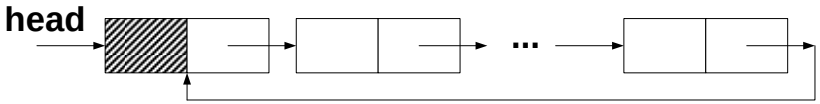
when flexibility is needed in inserting, deleting, and rearranging the entries.

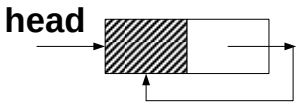
- 线性表的链式存储结构 -- 单链表、循环链表、双向链表

(单链)循环链表(circular linked list)

特征

• 循环链表中的最后一个结点(即表尾结点)的指针域不为空值, 而指向表头结点(或第一个结点), 使链表构成环状(这样从表中任一结点出发均可找到表中的其它结点, 提高了查找效率)





空表(head->next == head)

• 在实际应用中, 可根据具体情况用rear指针(在创建链表时添加的)指向表尾结点, 以提高效率.

例如, P35/图2.13 通过tail/rear指针合并两个Circular Linked Lists算法. 假设链表A、B的表头指针和表尾指针分别为HA、HB和TA、TB:

```
{  TB->next = TA-> next;
   TA->next = HB->next;
   free(HB)；} //O(1)
```

- 线性表的链式存储结构 -- 单链表、循环链表、双向链表

  (单链)循环链表(circular linked list)

  运算　循环链表运算和单链表基本一致, 其差别仅在于循环条件不同

  - 单链表: p 或 p->next 是否为 NULL
  - 循环链表: p 或 p->next 是否等于 head

  例: 统计一个由head指向其头结点的循环链表中数据元素个数的算法.

```
int count(Node *head) {
    int c = 0;
    p = head -> next;
    while ( p != head ) {
        c++; p = p->next;
    }
    return c;
}
```

- 线性表的链式存储结构 -- 单链表、循环链表、双向链表

双向链表/双向循环链表(double linked list / dll)

背景
- 单链表中从任何一个结点只能找到其后继结点, 而无法找到其前驱结点, 即只能单向操作 [GetNextElem→O(1), GetPriorElem→O(n)]
- 若断链, 则无法遍历其后的结点

特征
- 双向链表中的每个结点均含有两个指针域:

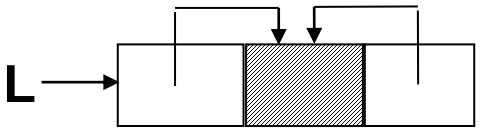
一个指向其后继, 一个则指向其前驱, 其数据结构:↓

一个结点(node)

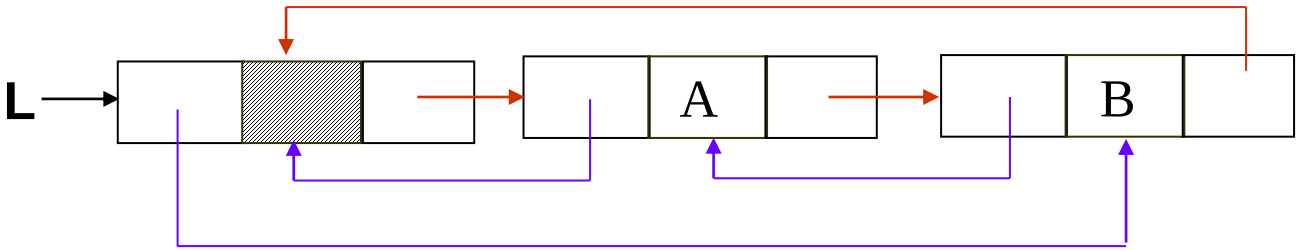
```
typedef struct node {
    Elemtype data;
    struct node *prior, *next;
} Node, *DLinkedList;
```

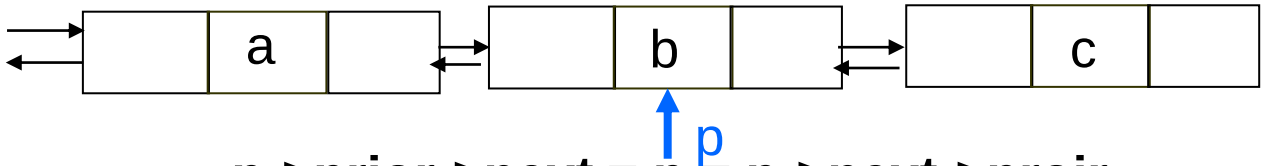
空双向循环链表:

Node *L;



非空双向循环链表:



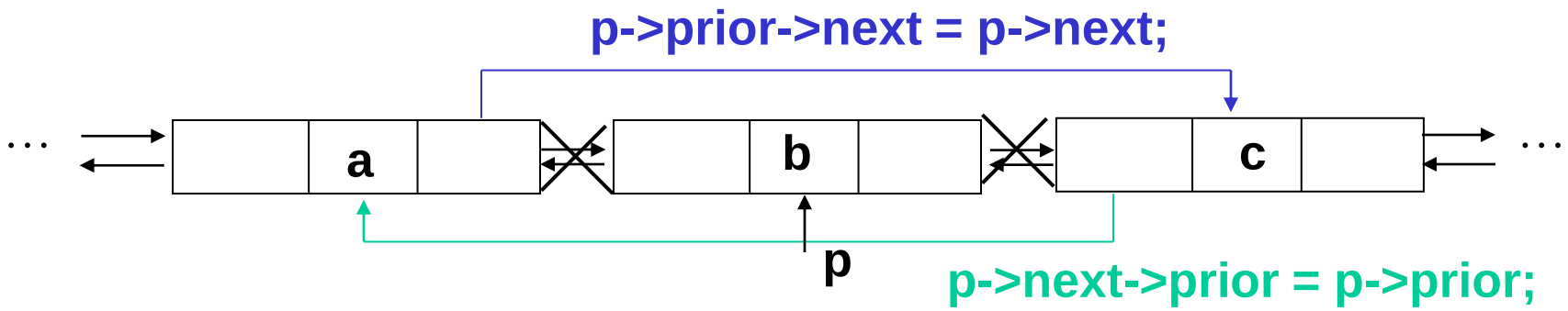


p->prior->next = p = p->next->proir;

- 线性表的链式存储结构 -- 单链表、循环链表、双向链表

  双向链表/双向循环链表(double linked list / dll)

  运算 -- 删除　如删除指针p指向的结点



算法描述

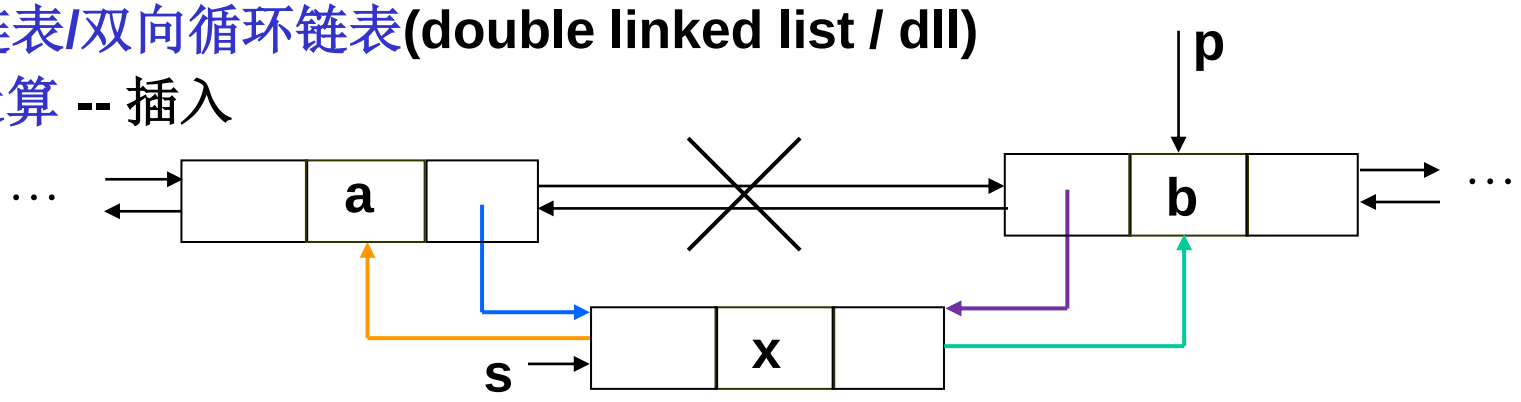
```
void DeleteDList(Node *p) {
    p->prior->next = p->next;
    p->next->prior = p->prior;
    free(p); }
```

算法分析　T(n) = O(1)

- 线性表的链式存储结构 -- 单链表、循环链表、双向链表

  双向链表/双向循环链表(double linked list / dll)

  运算 -- 插入



算法描述

```
void InsertDList(Node *p, ElemtType x) {
    s = (Node *)malloc(sizeof(Node));
    s->data = x;
    s->prior = p->prior;
    p->prior->next = s;
    s->next = p;
    p->prior = s;  }
```

注意语句次序

算法评价　T(n) = O(1)

- 线性表的应用举例 -- 一元多项式的表示及相加

一元多项式的表示

设有多项式: $P_n(x) = P_0 + P_1x + P_2x^2 + \cdots\cdots + P_nx^n$

可用线性表P表示 $P = (P_0, P_1, P_2, \cdots\cdots, P_n)$

但对S(x)这样的多项式浪费空间 $S(x) = 1 + 3x^{1000} + 2x^{20000}$

一般 $P_n(x) = P_1x^{e1} + P_2x^{e2} + \cdots\cdots + P_mx^{em}$

其中 $0 \le e1 \le e2 \cdots\cdots \le em$（$P_i$为非零系数）

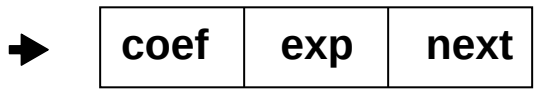
用数据域含两个数据项(非零系数, 指数值)的线性表表示

$((P_1, e1), (P_2, e2), \cdots\cdots (P_m, em))$

其存储结构可以用顺序存储结构, 也可以用单链表

单链表的结点定义

```
typedef struct node {
    int coef, exp;
    struct node *next;
} Node;
```



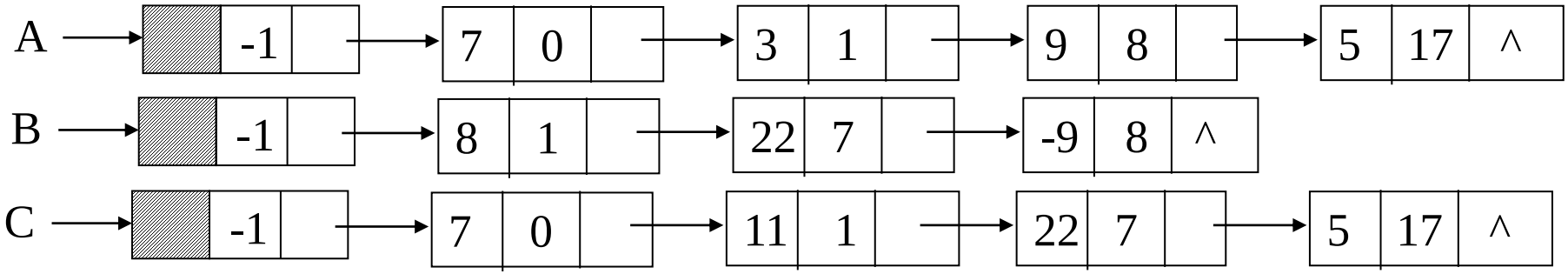
- 线性表的应用举例 -- 一元多项式的表示及相加

一元多项式相加

$$A(x) = 7 + 3x + 9x^{8} + 5x^{17}$$

$$B(x) = 8x + 22x^{7} - 9x^{8}$$

$$C(x) = A(x) + B(x) = 7 + 11x + 22x^{7} + 5x^{17}$$



运算规则　设p, q分别指向A, B中某一结点，p, q初值是第一结点, 结果多项式C由p指向

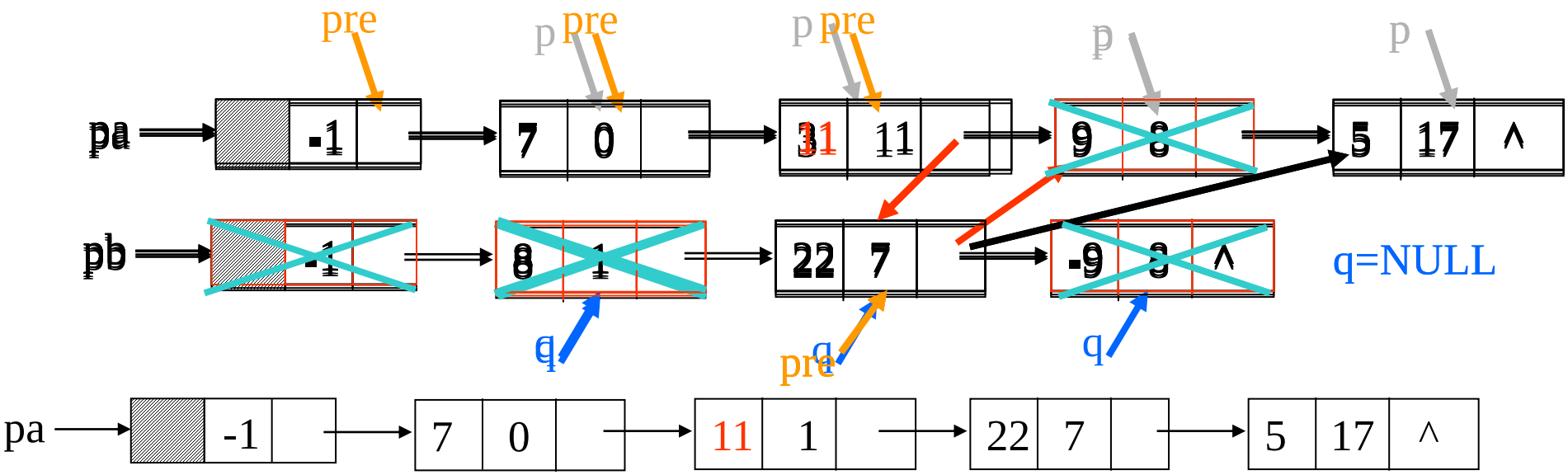
比较 p->exp与q->exp

- p->exp < q->exp: p结点是结果多项式中的一项 p后移, q不动
- p->exp > q->exp: q结点是结果多项式中的一项 将q插在p之前, q后移, p不动
- p->exp = q->exp: 系数相加
  - 0：从A表中删去p, 释放p, q，p, q后移
  - ≠0：修改p系数域, 释放q，p, q后移

直到p或q为NULL
- 若q==NULL，结束
- 若p==NULL，将B中剩余部分链到A上即可

- 线性表的应用举例 --- 一元多项式的表示及相加

  算法描述 P43/A.2.23



运算规则　设p、q分别指向A, B中某一结点, p、q初值是第一结点,
结果多项式C由p指向

比较 p->exp与q->exp：

- p->exp < q->exp:　p结点是结果多项式中的一项
  p后移, q不动
- p->exp > q->exp:　q结点是结果多项式中的一项
  将q插在p之前, q后移, p不动
- p->exp = q->exp:　系数相加
  - 0：从A表中删去p,
    释放p, q，p, q后移
  - ≠0：修改p系数域,
    释放q，p, q后移

直到p或q为NULL
- 若q==NULL，结束
- 若p==NULL，将B中剩余部分链到A上即可

Exercises (After class):

1. 比较顺序存储结构和链式储存结构的优缺点.
2. 说明a contiguous list(顺序表) and an array(数组)的区别和联系.
3. 在一非递减有序线性表中, 插入一个值为X的数据元素, 使得插入后的线性表仍为非递减序列. 要求分别用顺序表和单链表写出相应的实现算法.
4. 写一算法将不带头结点的单链表逆置, 要求操作在原表上进行. 假设该单链表由head指针指向.
5. Josephu问题: 有N(N≥1)个人围成一圈, 从第i(i≤N)个人开始从1报数, 数到m(m≥1)时, 此人出列; 下一人重新从1开始报数, 再数到m时, 又一人出列, 直至所有人全部出列. 要求写一个算法实现按出列的先后次序得到一个新的序列, 如N=5, i=1, m=3,则新序列是3、1、5、2、4.