



Data Structure & Algorithm Analysis

Linked List

Zibin Zheng (郑子彬)

School of Data and Computer Science , SYSU

<http://www.inpluslab.com>

课程主页: <http://inpluslab.sysu.edu.cn/dsa2016/>

Definition

□ **线性表**：简称表，是 n ($n \geq 0$) 个具有**相同类型**的数据元素的**有限序列**。

□ **线性表的长度**：线性表中数据元素的个数。

□ $L = (a_1, a_2, \dots, a_{i-1}, a_i, \dots, a_n)$

26个英文字母的字母表：
(**A, B, C, ..., Z**)

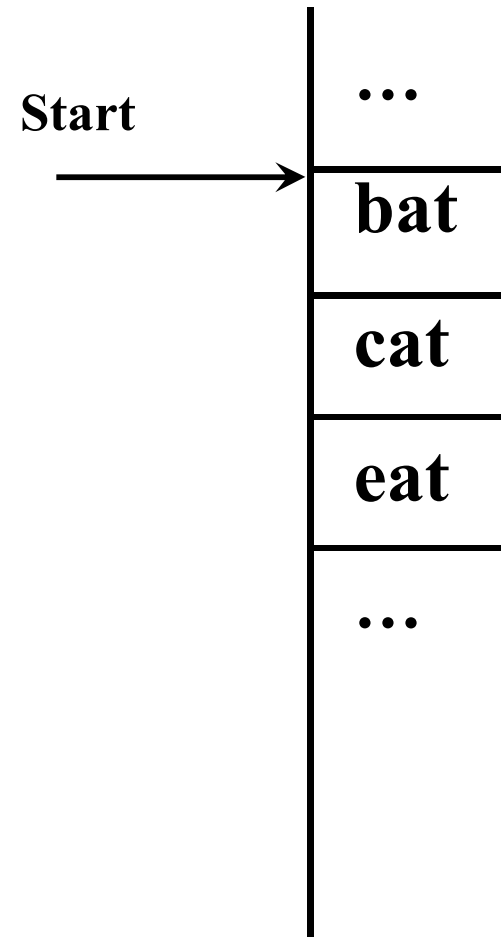
某校从1978年到1983年各种型号的计算机拥有量的变化情况：
(**6, 17, 28, 50, 92, 188**)

Two types of physical structure

(1) Sequence (顺序存储结构) :

- ✓ Storage cell with continuous location address
- ✓ Logical relationship is described by the function of **storage location**

Case: (bat, cat, eat)

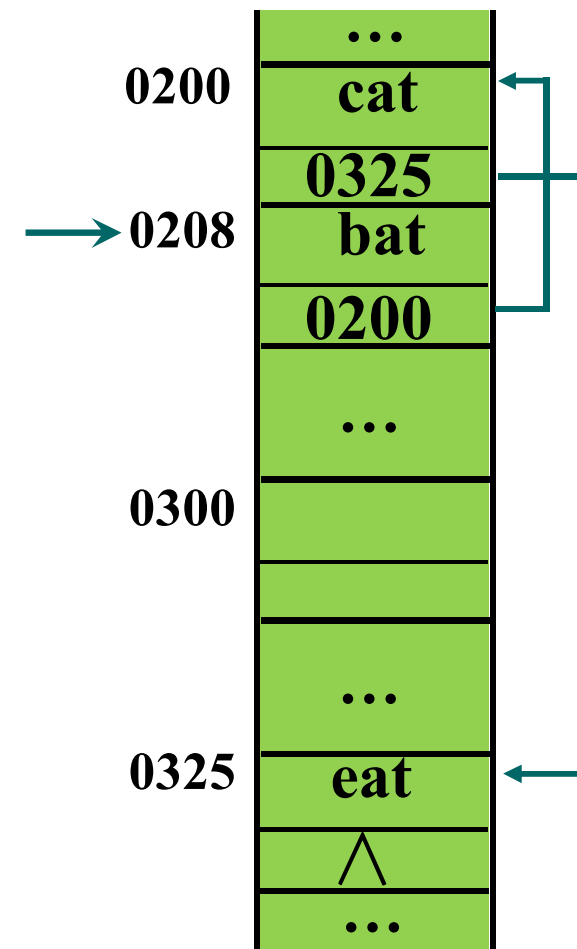


Two types of physical structure

(2) Linked (链式存储结构) :

- ✓ Storage cell
- ✓ Logical relationship is described by **point**

Case: (bat, cat, eat)



Implementation

- Two main types:
- Sequence List (Array)
 - 线性表的**顺序存储结构**，指的是用一段地址连续的存储单元依次存储线性表的数据元素

★ Linked List

- 线性表的**链式存储结构**，指的是用一组任意的存储单元存储线性表的数据元素，这组存储单元可以是连续的，也可以是不连续的。

Linked List

单链表 (Single Linked List)

顺序表 → 静态存储分配 → 事先确定容量

链表 → 动态存储分配 → 运行时分配空间

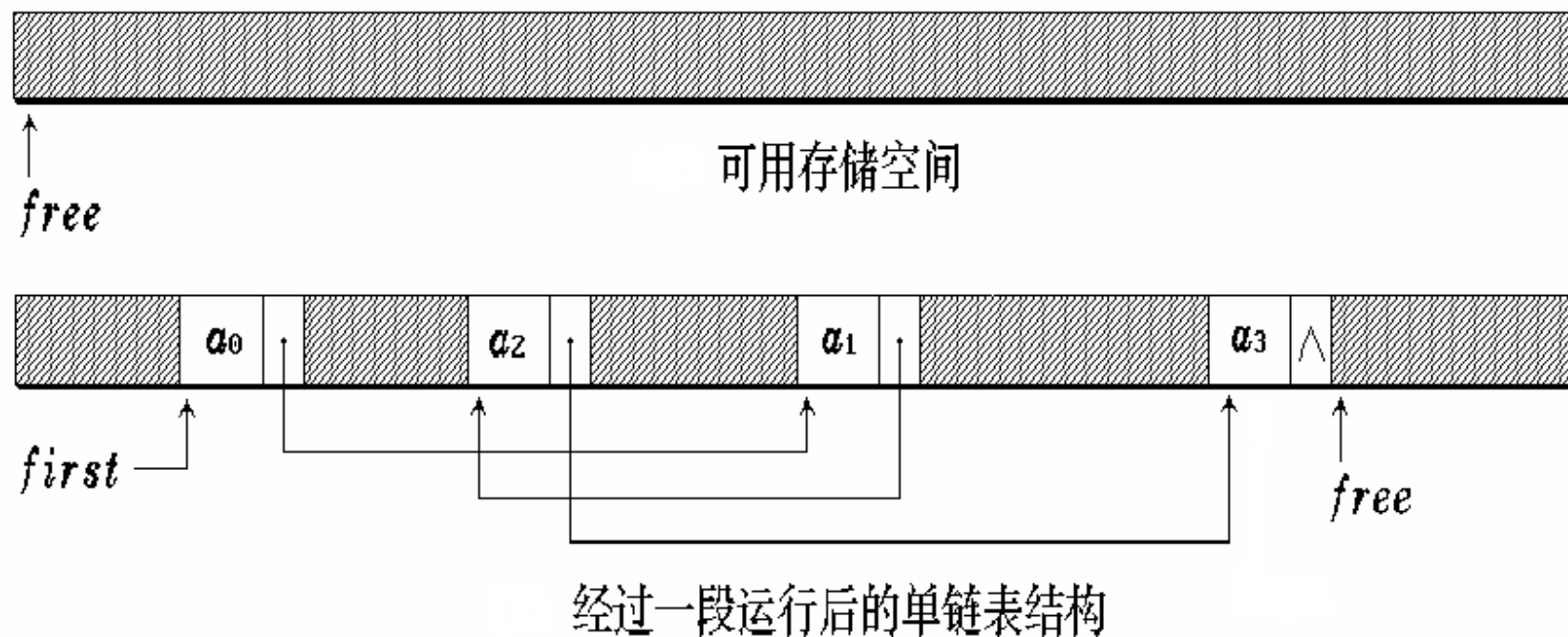
单链表：线性表的链接存储结构。

存储思想：用一组任意的存储单元存放线性表的元素。

└→ { 连续
 不连续
 零散分布

Linked list

- In the computer



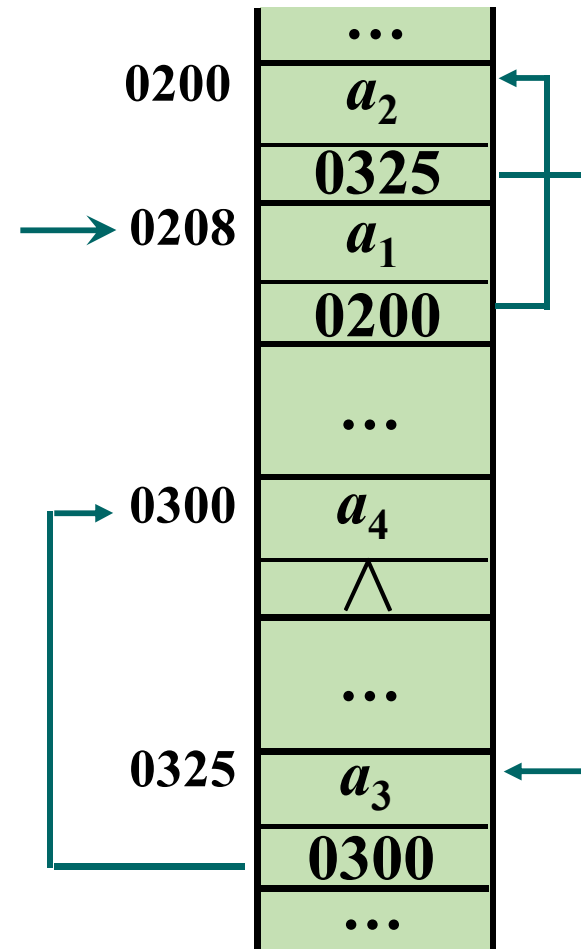
Linked List

单链表

例：(a_1, a_2, a_3, a_4)的存储示意图

存储特点：

1. 逻辑次序和物理次序不一定相同。
2. 元素之间的逻辑关系用**指针**表示。

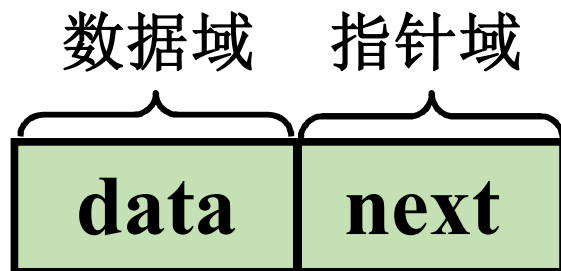


Linked List

单链表

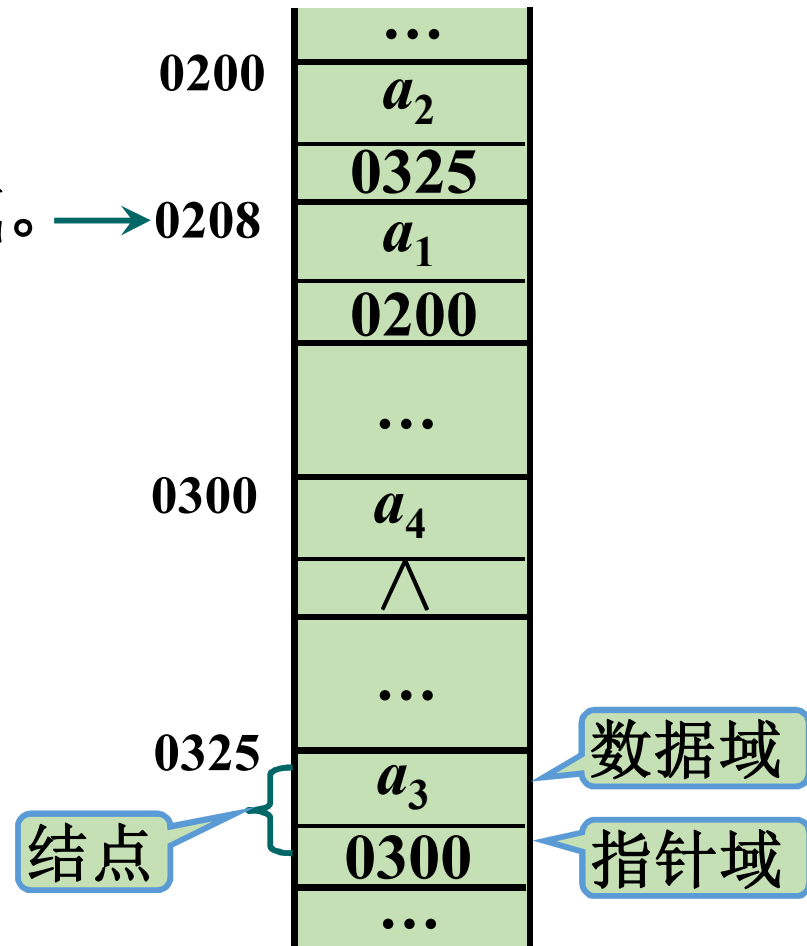
单链表是由若干结点构成的；
单链表的结点只有一个指针域。

单链表的结点结构：



data: 存储数据元素

next: 存储指向后继结点的地址



Linked List

单链表

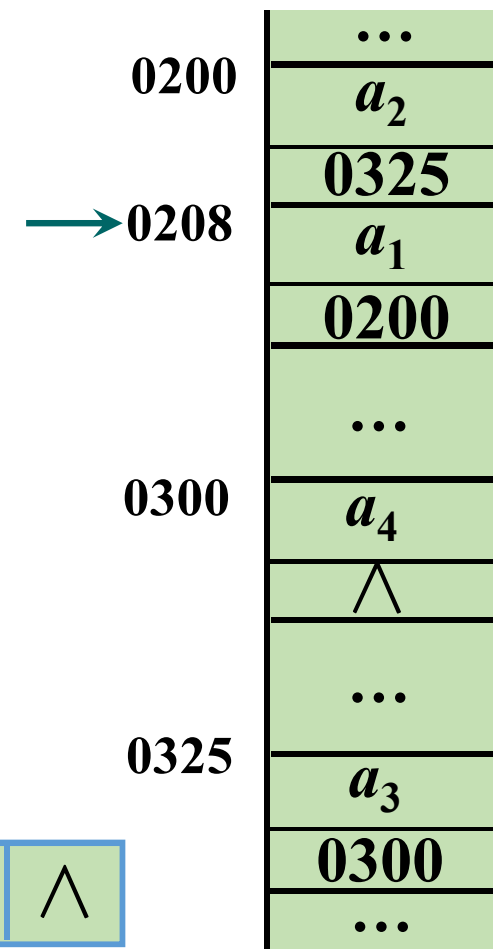
① 什么是存储结构?

重点在数据元素之间的逻辑关系的表示, 所以, 将实际存储地址抽象。

空表

first=NULL

非空表



Linked List

单链表

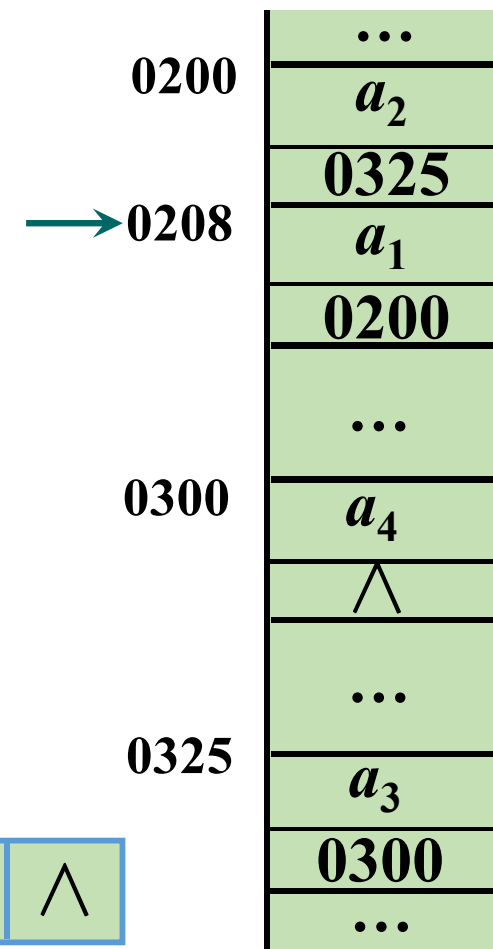
头指针： 指向第一个结点的地址。

尾标志： 终端结点的指针域为空。

空表

first=NULL

非空表



Linked List

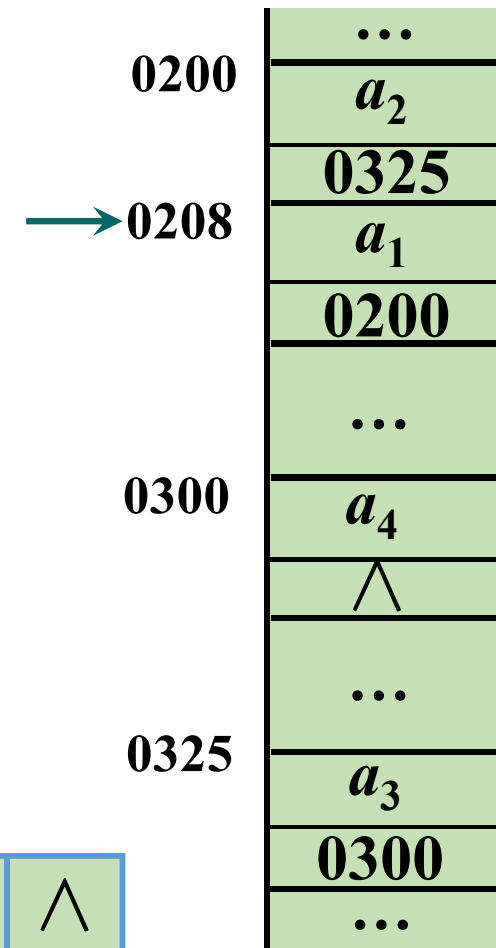
单链表

- ① 空表和非空表不统一，缺点？
如何将空表与非空表统一？
- ② 对第一个结点的操作与对其他结点的操作，使用语句可一致？

空表

first=NULL

非空表

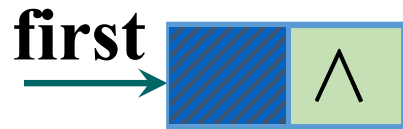


Linked List

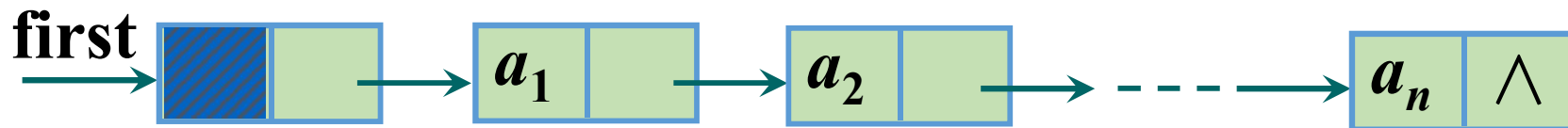
单链表

头结点：在单链表的第以一个元素结点之前附设一个类型相同的结点，以便空表和非空表处理统一，并且可统一对任意位置的结点的操作实现。

空表



非空表



Node type of linked list

```
typedef struct LNode{  
    ElemType data;  
    struct Lnode *next;  
} LNode, * LinkList;
```

简化为typedef int INT_PTR, *PUINT_PTR;

先看第一句: typedef int INT_PTR;

意思是为int取一个别名INT_PTR, 则INT_PTR和int就是一样的意思, 可以这样使用

INT_PTR a;

a = 10;

第二句: typedef int *PUINT_PTR;

就是为int *取一个别名PUINT_PTR, 则PUINT_PTR代表的就是int类型的指针, 使用如下:

int a = 0;

PUINT_PTR b = &a;

Read the data from the linked list (Search)

(1) 按序号查找 (Search by index)

取单链表中的第 i 个元素。

对于单链表，不能象顺序表中那样直接按序号 i 访问结点，而只能从链表的头结点出发，沿链域 `next` 逐个结点往下搜索，直到搜索到第 i 个结点为止。因此，链表不是随机存取结构。

设单链表的长度为 n ，要查找表中第 i 个结点，仅当 $1 \leq i \leq n$ 时， i 的值是合法的。

Read the data from the linked list (Search)

获得链表第 i 个数据的算法思路：

1. 声明一个结点 p 指向链表第一个结点，初始化 j 从 1 开始；
2. 当 $j < i$ 时，就遍历链表，让 p 的指针向后移动，不断指向下一结点， j 累加 1；
3. 若到链表末尾 p 为空，则说明第 i 个元素不存在；
4. 否则查找成功，返回结点 p 的数据。

/*初始条件：单链线性表 L 已存在， $1 \leq i \leq \text{ListLength}(L)$ */

/*操作结果：用 e 返回 L 中第 i 个数据元素的值*/

Status GetElem(LinkList L, int i, ElemType *e)

```
{
    int j;
    LinkList p;          /*声明一结点 p*/
    p = L->next;         /*让 p 指向链表 L 的第一个结点*/
    j = 1;               /*j 为计数器*/
    while (p && j < i) /*p 不为空或者计数器 j 还没有等于 i 时，循环继续*/
    {
        p = p->next;    /*让 p 指向下一个结点*/
```

```
        ++j;
    }
    if ( !p || j > i )
        return ERROR; /*第 i 个元素不存在*/
    *e = p->data;      /*取第 i 个元素的数据*/
    return OK;
}
```

移动指针 p 的频率：

$i < 1$ 时：0 次； $i \in [1, n]$ ： $i-1$ 次； $i > n$ ： n 次。

即时间复杂度： $O(n)$ 。

Read the data from the linked list (Search)

(2) 按值查找 (Search by value)

按值查找是在链表中，查找是否有结点值等于给定值key的结点？若有，则返回首次找到的值为key的结点的存储位置；否则返回NULL。查找时从开始结点出发，沿链表逐个将结点的值和给定值key作比较。

Read the data from the linked list (Search)

- 算法描述

LinkedList Locate_Node(LinkedList L, int key)

/* L为头结点的单链表的头指针，查找值为key的第一个结点 */

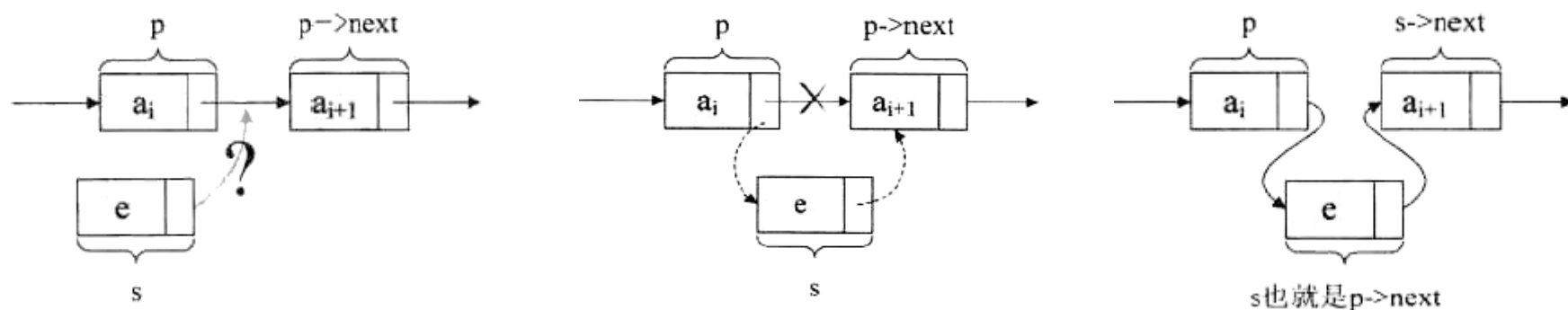
```
{ LinkedList p=L->next;
    while ( p!=NULL&& p->data!=key)  p=p->next;
    if (p==NULL) {
        printf("所要查找的结点不存在!!\n");
        return(NULL);
    }else{ return p;}
}
```

算法的执行与形参key有关，平均时间复杂度为 $O(n)$ 。

Node insertion in the linked list

- 插入运算是将值为 e 的新结点插入到表的第 i 个结点的位置上，即插入到 a_i 与 a_{i+1} 之间。因此，必须首先找到 a_i 所在的结点 p ，然后生成一个数据域为 e 的新结点 q ， q 结点作为 p 的直接后继结点。
- 设链表的长度为 n ，合法的插入位置是 $1 \leq i \leq n$ 。算法的时间主要耗费移动指针 p 上，故时间复杂度亦为 $O(n)$ 。

Node insertion in the linked list



单链表第 i 个数据插入结点的算法思路：

1. 声明一结点 p 指向链表第一个结点，初始化 j 从 1 开始；
2. 当 $j < i$ 时，就遍历链表，让 p 的指针向后移动，不断指向下一结点， j 累加 1；
3. 若到链表末尾 p 为空，则说明第 i 个元素不存在；
4. 否则查找成功，在系统中生成一个空结点 s ；
5. 将数据元素 e 赋值给 $s \rightarrow \text{data}$ ；
6. 单链表的插入标准语句 $s \rightarrow \text{next} = p \rightarrow \text{next}; \quad p \rightarrow \text{next} = s;$
7. 返回成功。

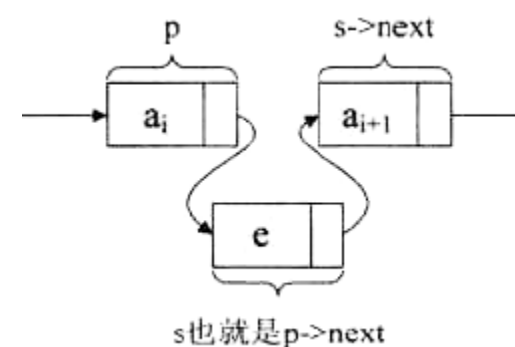
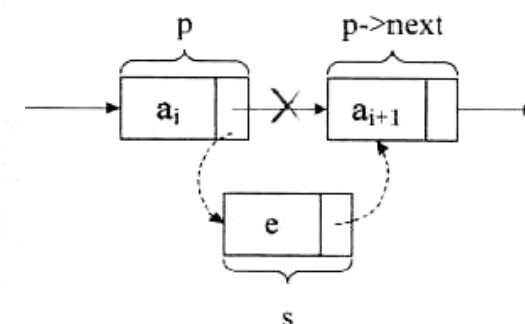
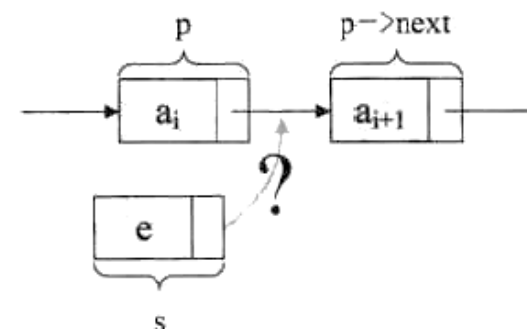
Node insertion in the linked list

/*初始条件：单链线性表 L 已存在， $1 \leq i \leq \text{ListLength}(L)$ ，*/

/*操作结果：在 L 中第 i 个位置之前插入新的数据元素 e，L 的长度加 1*/

Status ListInsert (LinkList *L, int i, ElemType e)

```
{
    int j;
    LinkList p, s;
    p = *L;
    j = 1;
    while (p && j < i)      /* 寻找第 i 个结点 */
    {
        p = p->next;
        ++j;
    }
    if (!p || j > i)
        return ERROR;      /*第 i 个元素不存在*/
    s = (LinkList) malloc (sizeof (Node)) ; /*生成新结点 (C 标准函数) */
    s->data = e;
    s->next = p->next;      /*将 p 的后继结点赋值给 s 的后继*/
    p->next = s;            /*将 s 赋值给 p 的后继*/
    return OK;
}
```



Node deletion in the linked list

(1) 按序号删除：删除单链表中的第 i 个结点。

为了删除第 i 个结点 a_i ，必须找到结点的存储地址。该存储地址是在其直接前趋结点 a_{i-1} 的next域中，因此，必须首先找到 a_{i-1} 的存储位置 p ，然后令 $p \rightarrow \text{next}$ 指向 a_i 的直接后继结点，即把 a_i 从链上摘下。最后释放结点 a_i 的空间，将其归还给“**存储池**”。

(2) 按值删除

删除单链表中值为 key 的第一个结点。

与按值查找相类似，首先要查找值为 key 的结点是否存在？若存在，则删除；否则返回NULL。

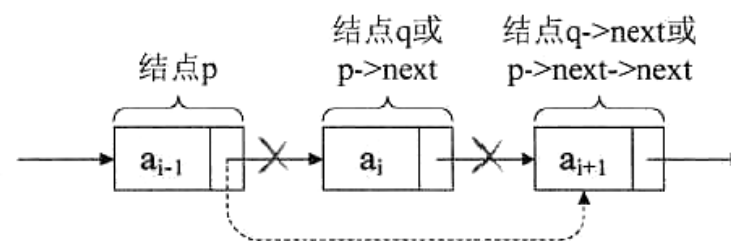
Node deletion in the linked list

单链表第 i 个数据删除结点的算法思路：

1. 声明一结点 p 指向链表第一个结点，初始化 j 从 1 开始；
2. 当 $j < i$ 时，就遍历链表，让 p 的指针向后移动，不断指向下一个结点， j 累加 1；
3. 若到链表末尾 p 为空，则说明第 i 个元素不存在；
4. 否则查找成功，将欲删除的结点 $p \rightarrow \text{next}$ 赋值给 q ；
5. 单链表的删除标准语句 $p \rightarrow \text{next} = q \rightarrow \text{next}$ ；
6. 将 q 结点中的数据赋值给 e ，作为返回；
7. 释放 q 结点；
8. 返回成功。

Node deletion in the linked list

```
/*初始条件：单链线性表 L 已存在， $1 \leq i \leq \text{ListLength}(L)$  */
/*操作结果：删除 L 的第 i 个数据元素，并用 e 返回其值，L 的长度减 1*/
Status ListDelete(LinkList *L, int i, ElemType *e)
{
    int j;
    LinkList p, q;
    p = *L;
    j = 1;
    while (p->next && j < i)    /*遍历寻找第 i 个元素*/
    {
        p = p->next;
        ++j;
    }
    if (!(p->next) || j > i)
        return ERROR;          /*第 i 个元素不存在*/
    q = p->next;
    p->next = q->next;          /*将 q 的后继赋值给 p 的后继*/
    *e = q->data;               /*将 q 结点中的数据给 e*/
    free(q);                    /*让系统回收此结点，释放内存*/
    return OK;
}
```



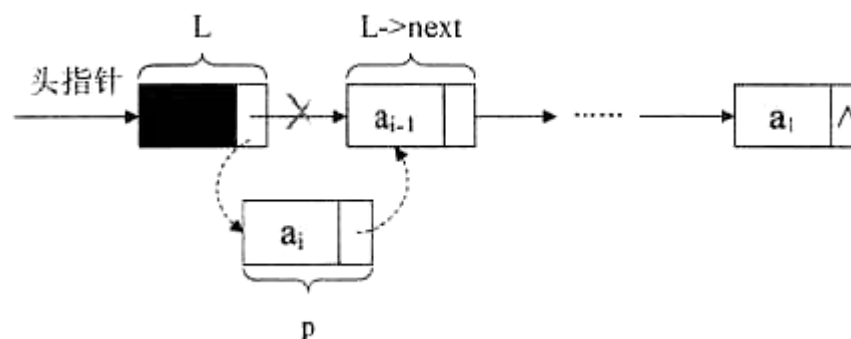
Node deletion in the linked list

- 算法分析
- 设单链表长度为 n ，则删去第 i 个结点仅当 $1 \leq i \leq n$ 时是合法的。则当 $i=n+1$ 时，虽然被删结点不存在，但其前趋结点却存在，是终端结点。故判断条件之一是 $p \rightarrow \text{next} \neq \text{NULL}$ 。显然此算法的时间复杂度也是 $O(n)$ 。

Create a linked list

单链表整表创建的算法思路：

1. 声明一结点 p 和计数器变量 i ;
2. 初始化一空链表 L ;
3. 让 L 的头结点的指针指向 $NULL$ ，即建立一个带头结点的单链表;
4. 循环:
 - ◆ 生成一新结点赋值给 p ;
 - ◆ 随机生成一数字赋值给 p 的数据域 $p \rightarrow data$;
 - ◆ 将 p 插入到头结点与前一新结点之间。



Create a linked list

```
/* 随机产生 n 个元素的值，建立带表头结点的单链线性表 L（头插法） */
void CreateListHead (LinkList *L, int n)
{
    LinkList p;
    int i;
    srand (time (0)); /*初始化随机数种子*/
    *L = (LinkList) malloc (sizeof (Node));
    (*L) -> next = NULL; /*先建立一个带头结点的单链表*/
    for (i=0; i<n; i++)
    {
        p = (LinkList) malloc (sizeof (Node)); /*生成新结点*/
        p->data = rand () % 100 + 1; /*随机生成 100 以内的数字*/
        p->next = (*L) -> next;
        (*L) -> next = p; /*插入到表头*/
    }
}
```

Delete a linked list

单链表整表删除的算法思路如下：

1. 声明一结点 p 和 q;
2. 将第一个结点赋值给 p;
3. 循环:
 - ◆ 将下一结点赋值给 q;
 - ◆ 释放 p;
 - ◆ 将 q 赋值给 p。

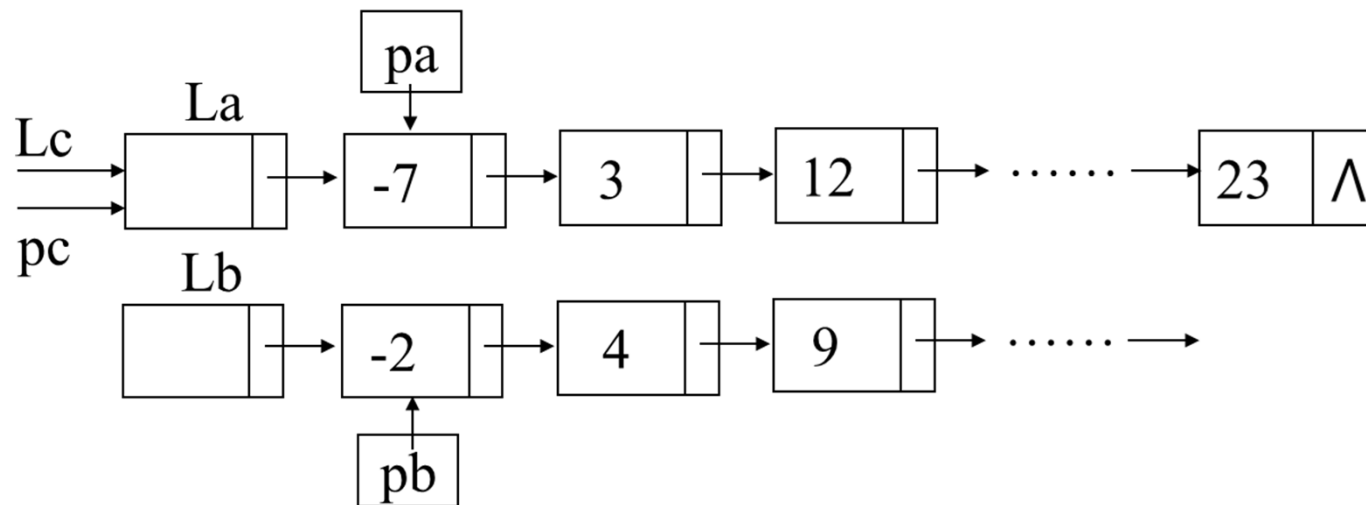
/*初始条件：单链线性表 L 已存在，操作结果：将 L 重置为空表*/

```
Status ClearList (LinkList *L)
```

```
{
    LinkList p,q;
    p= (*L) ->next;           /*p 指向第一个结点*/
    while (p)                  /*没到表尾*/
    {
        q=p->next;
        free (p) ;
        p=q;
    }
    (*L) ->next=NULL;         /*头结点指针域为空*/
    return OK;
}
```

Merge the linked list

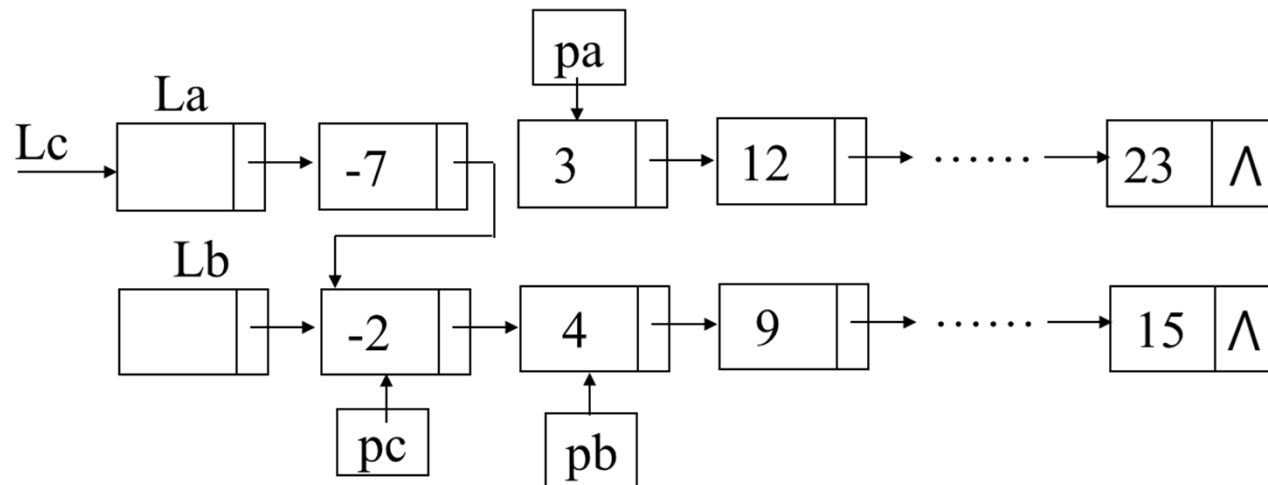
- 设有两个有序的单链表，它们的头指针分别是La和Lb，将它们合并为以Lc为头指针的有序链表。合并前的示意图如图所示。



两个有序的单链表La，Lb的初始状态

Merge the linked list

- 合并了值为-7，-2的结点后示意图如图2所示。



合并了值为-7，-2的结点后的状态

- 算法中pa，pb分别是待考察的两个链表的当前结点，pc是合并过程中合并的链表的最后一个结点。

Merge the linked list

- 算法描述

LNode *Merge_LinkList(LNode *La, LNode *Lb)

/* 合并以La, Lb为头结点的两个有序单链表 */

{ LNode *Lc, *pa, *pb, *pc, *ptr ;

Lc=La ; pc=La ; pa=La->next ; pb=Lb->next ;

while (pa!=NULL && pb!=NULL)

{ if (pa->data<pb->data)

{ pc->next=pa ; pc=pa ; pa=pa->next ; }

/* 将pa所指的结点合并, pa指向下一个结点 */

if (pa->data>pb->data)

{ pc->next=pb ; pc=pb ; pb=pb->next ; }

/* 将pb所指的结点合并, pb指向下一个结点 */

Merge the linked list

```
    if (pa->data==pb->data)
        { pc->next=pa ; pc=pa ; pa=pa->next ;
          ptr=pb ; pb=pb->next ; free(ptr) ; }
    /* 将pa所指的结点合并，pb所指结点删除 */
}
if (pa!=NULL) pc->next=pa ;
else pc->next=pb ; /*将剩余的结点链上*/
free(Lb) ;
return(Lc) ;
}
```

算法分析

若La，Lb两个链表的长度分别是m，n，则链表合并的时间复杂度为 $O(m+n)$ 。

Analysis of implementation in linked list

- 对一个结点操作，必先找到它，即用一个指针指向它
- 找单链表中任一结点，都必须从第一个点开始：

```
p = head;  
while (没有到达)  
    p = p->next;
```

□ 单链表的时间复杂度 $O(n)$

- 定位: $O(n)$
- 插入: $O(n) + O(1)$
- 删除: $O(n) + O(1)$
- 合并: $O(n+m)$

存储分配方式比较

- 顺序表采用顺序存储结构，即用一段地址**连续**的存储单元**依次**存储线性表的数据元素，数据元素之间的逻辑关系通过**存储位置**来实现。
- 链表采用链接存储结构，即用一组**任意**的存储单元存放线性表的元素，用**指针**来反映数据元素之间的逻辑关系。

时间性能比较

时间性能是指实现基于某种存储结构的基本操作（即算法）的时间复杂度。

按位查找：

- 顺序表的时间为 $O(1)$ ，是随机存取；
- 链表的时间为 $O(n)$ ，是顺序存取。

插入和删除：

- 顺序表需移动表长一半的元素，时间为 $O(n)$ ；
- 链表不需要移动元素，在给出某个合适位置的指针后，插入和删除操作所需的时间仅为 $O(1)$ 。

Array vs. Linked List

空间性能是指某种存储结构所占用的存储空间的大小。

定义结点的存储密度：

$$\text{存储密度} = \frac{\text{数据域占用的存储量}}{\text{整个结点占用的存储量}}$$

结点的存储密度：

- 顺序表：结点的存储密度为1（只存储数据元素），没有浪费空间；
- 链表：结点的存储密度<1（包括数据域和指针域），有指针的结构性开销。

Array vs. Linked List

空间性能是指某种存储结构所占用的存储空间的大小。

定义结点的存储密度：

$$\text{存储密度} = \frac{\text{数据域占用的存储量}}{\text{整个结点占用的存储量}}$$

结构的存储密度：

- 顺序表：需要预分配存储空间，如果预分配得过大，造成浪费，若估计得过小，又将发生上溢；
- 链表：不需要预分配空间，只要有内存空间可以分配，单链表中的元素个数就没有限制。

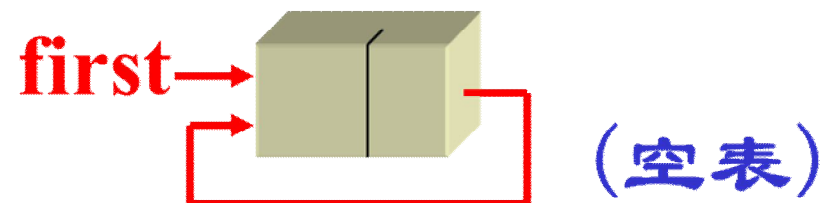
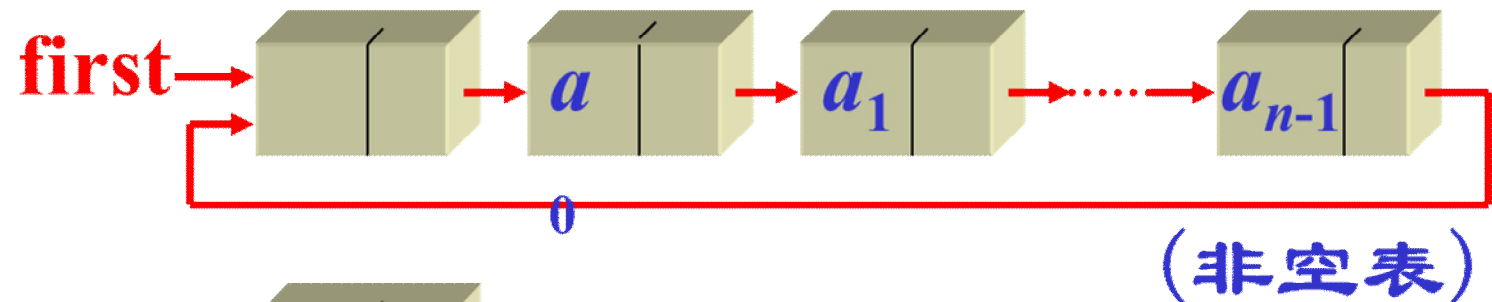
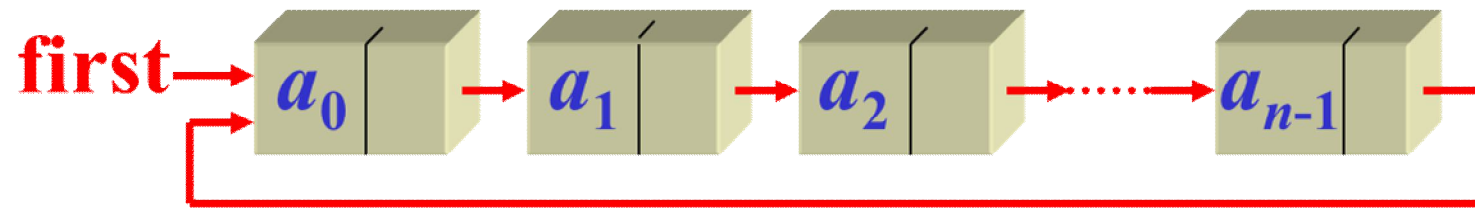
Array vs. Linked List

- (1) 若线性表需**频繁查找**却很少进行插入和删除操作，或其操作和元素在表中的位置密切相关时，宜采用顺序表作为存储结构；若线性表需**频繁插入和删除**时，则宜采用链表做存储结构。
- (2) 当线性表中元素**个数变化**较大或者未知时，最好使用链表实现；而如果用户事先知道线性表的大致长度，使用顺序表的空间效率会更高。

线性表的**顺序实现**和**链表实现**各有其优缺点，不能笼统地说哪种实现更好，只能根据实际问题的具体需要，并对各方面的优缺点加以综合平衡，才能最终选定比较适宜的实现方法。

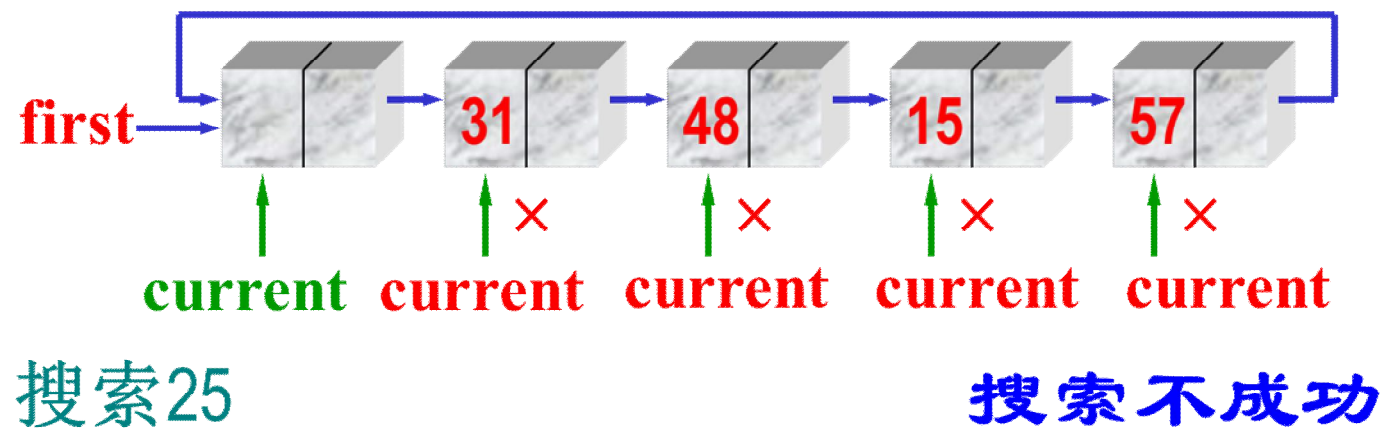
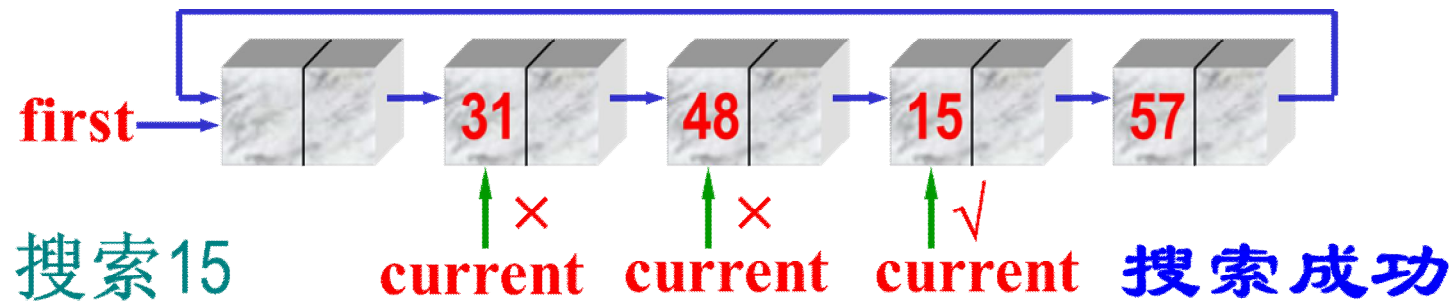
Circular Linked List

- 循环链表



Circular Linked List

- 循环链表的搜索算法

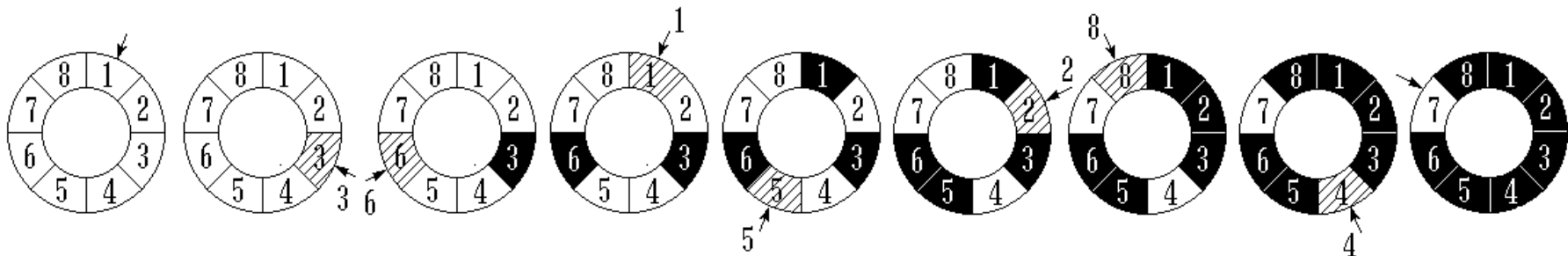


Circular Linked List

- 用循环链表求解约瑟夫问题

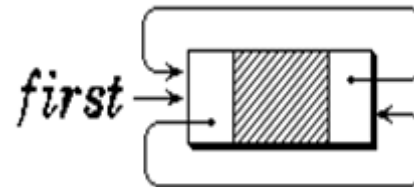
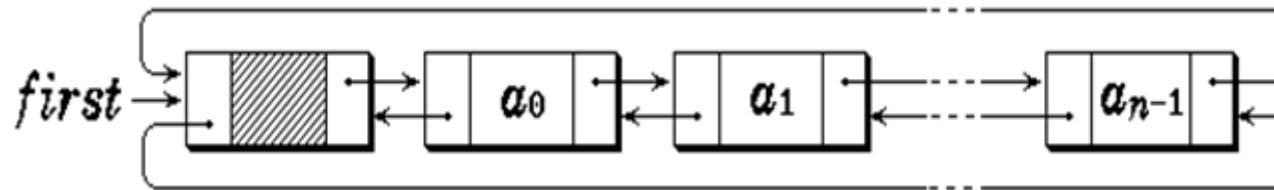
- 约瑟夫问题的提法

n 个人围成一个圆圈，首先第 1 个人从 1 开始一个人一个人顺时针报数，报到第 m 个人，令其出列。然后再从下一个人开始，从 1 顺时针报数，报到第 m 个人，再令其出列，...，如此下去，直到圆圈中只剩一个人为止。此人即为优胜者。



Doubly Linked List

- 双向链表



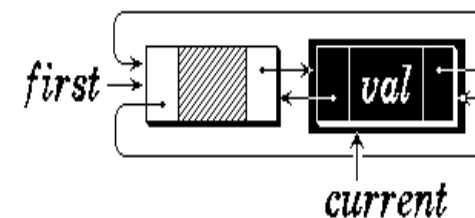
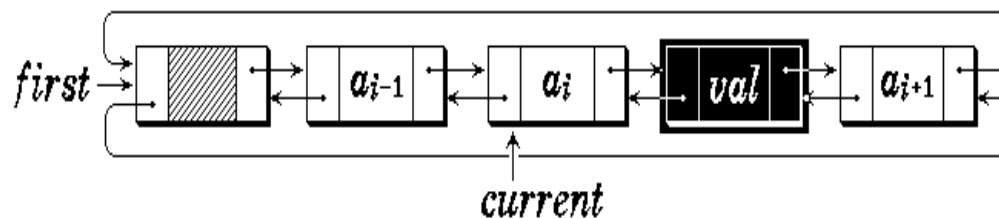
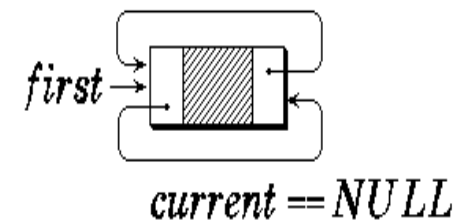
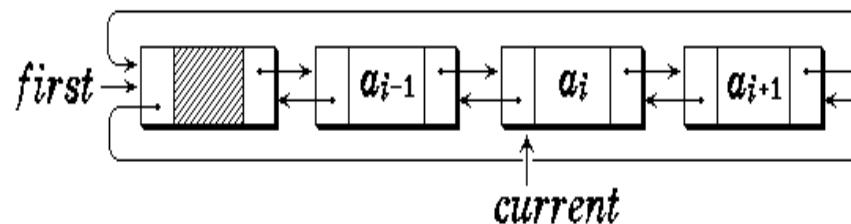
Insert at doubly circular Linked list

$p \rightarrow \text{prior} = \text{current};$ (1)

$p \rightarrow \text{next} = \text{current} \rightarrow \text{next};$ (2)

$\text{current} \rightarrow \text{next} = p;$ (3)

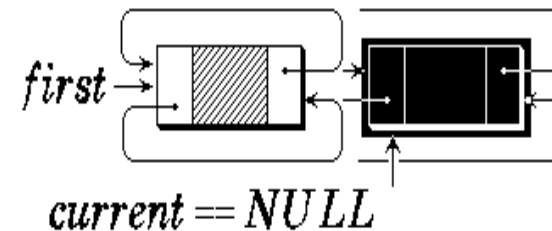
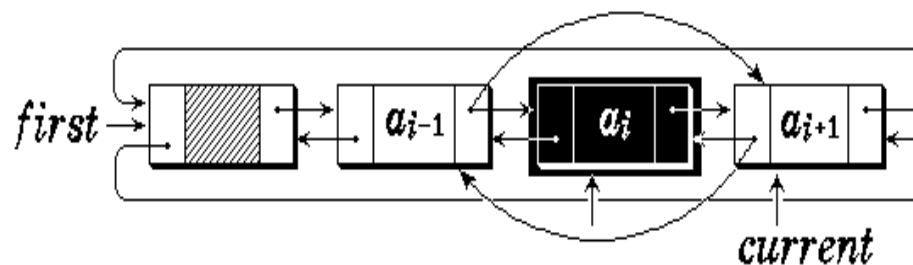
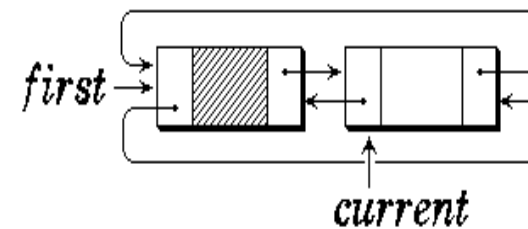
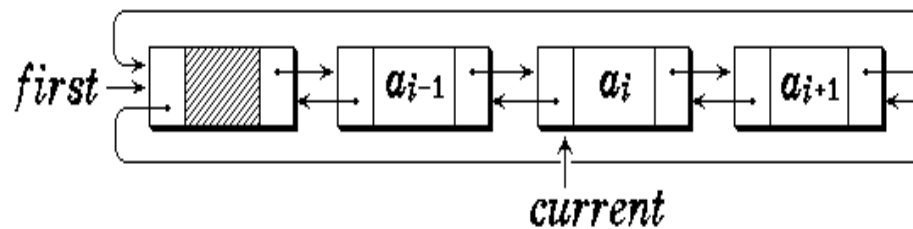
$p \rightarrow \text{next} \rightarrow \text{prior} = p;$ (4)



Delete at doubly circular Linked list

$current \rightarrow next \rightarrow prior = current \rightarrow prior;$

$current \rightarrow prior \rightarrow next = current \rightarrow next;$



Application

- 一元多项式的表示和相加

$$\begin{aligned} P_n(x) &= a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \\ &= \sum_{i=0}^n a_i x^i \end{aligned}$$

- n 阶多项式 $P_n(x)$ 有 $n+1$ 项。
 - 系数 $a_0, a_1, a_2, \dots, a_n$
 - 指数 $0, 1, 2, \dots, n$ 。按升幂排列

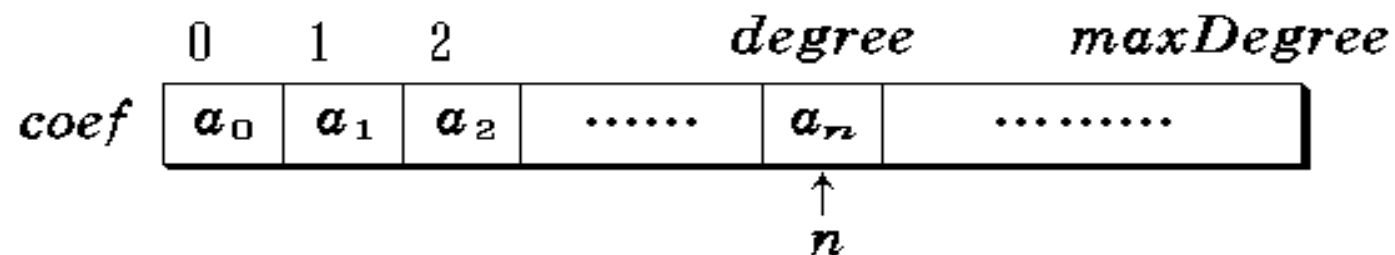
Application

- 第一种表示方法

$$P_n = (a_0, a_1, a_2, \dots, a_n)$$

适用于指数连续排列、“0”系数较少的情况。

但对于指数不全的多项式，如 $P_{20000}(x) = 3 + 5x^{50} + 14x^{20000}$ ，会造成系统空间的巨大浪费。



Application

- 第二种表示方法

一般情况下，一元多项式可写成：

$$P_n(x) = p_1x^{e_1} + p_2x^{e_2} + \dots + p_mx^{e_m}$$

其中： p_i 是指数为 e_i 的项的非零系数， $0 \leq e_1 \leq e_2 \leq \dots \leq e_m \leq n$

二元组表示 $((p_1, e_1), (p_2, e_2), \dots, (p_m, e_m))$

例： $P_{999}(x) = 7x^3 - 2x^{12} - 8x^{999}$

表示成： $((7, 3), (-2, 12), (-8, 999))$

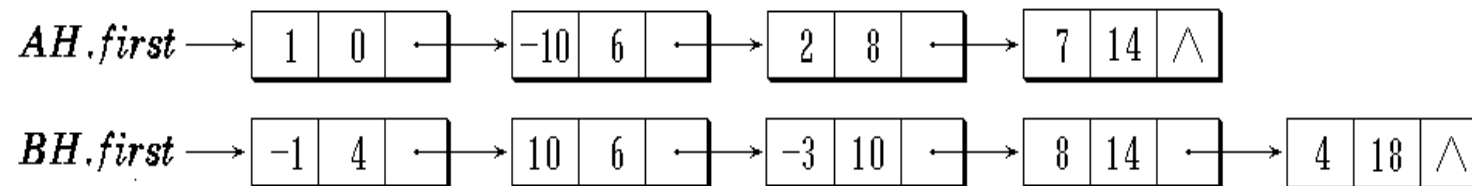
	0	1	2		i		m
<i>coef</i>	a_0	a_1	a_2	a_i	a_m
<i>exp</i>	e_0	e_1	e_2	e_i	e_m

Application

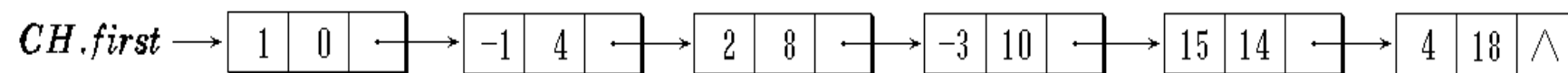
- 第三种表示方法—多项式链表

$$AH = 1 - 10x^6 + 2x^8 + 7x^{14}$$

$$BH = -x^4 + 10x^6 - 3x^{10} + 8x^{14} + 4x^{18}$$



(a) 两个相加的多项式



(b) 相加结果的多项式

多项式链表合并算法

初始化；

While (两个链都没处理完)

{ if (指针指向当前节点的指数项相同)

{系数相加，在C链中填加新的节点；

A、B链的指针均前移；}

else

{以指数小的项的系数添入C链中的新节点；

指数小的相应链指针前移；}

}

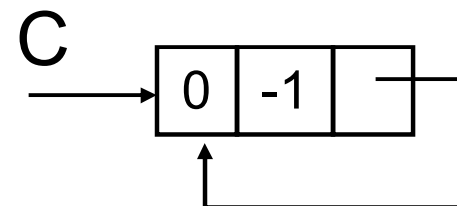
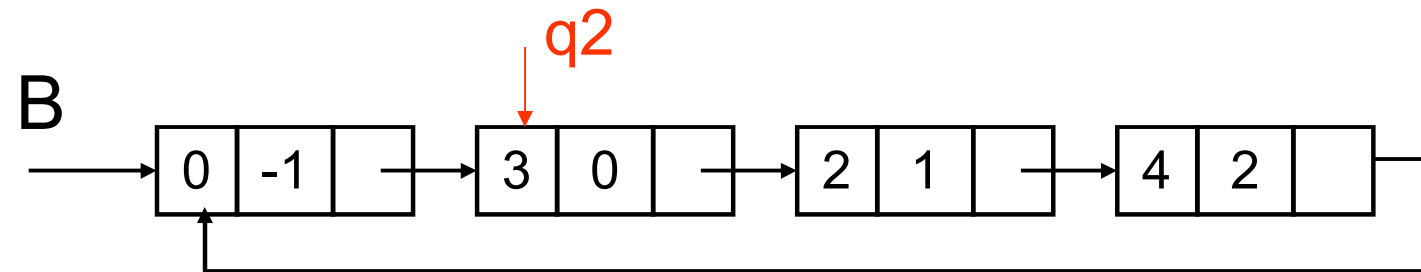
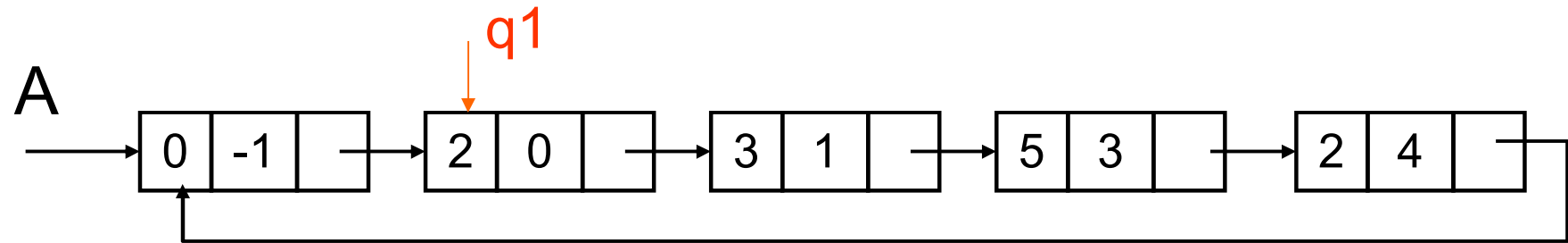
While(A链处理完)

{ 顺序处理B链；}

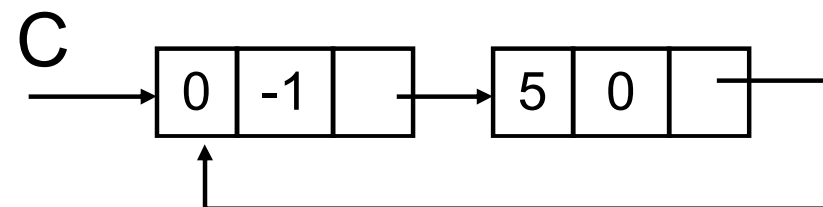
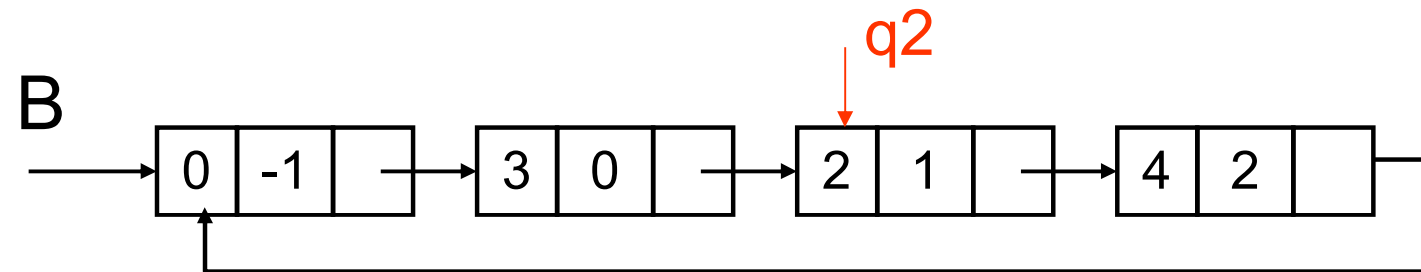
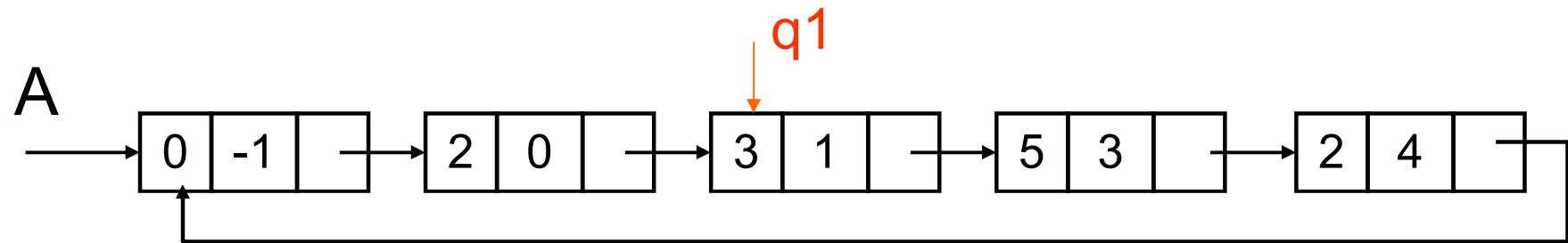
While(B链处理完)

{ 顺序处理A链；}

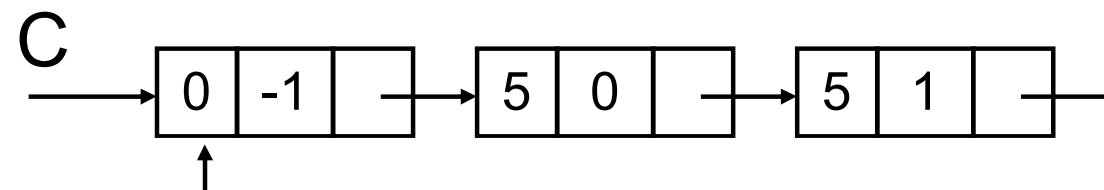
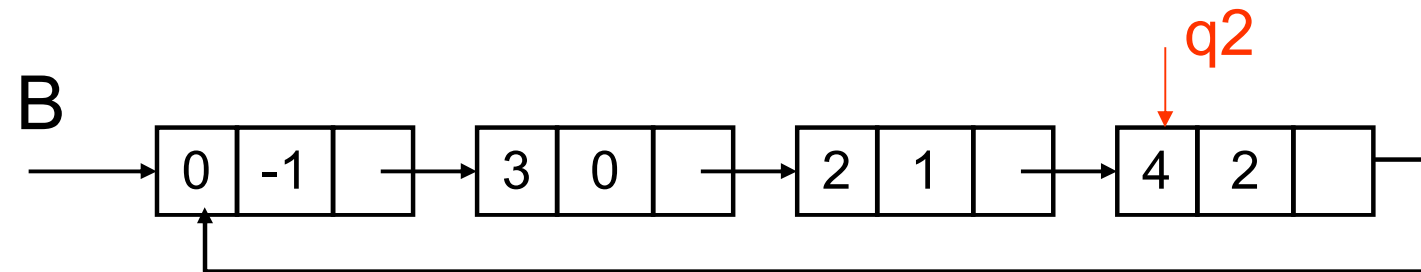
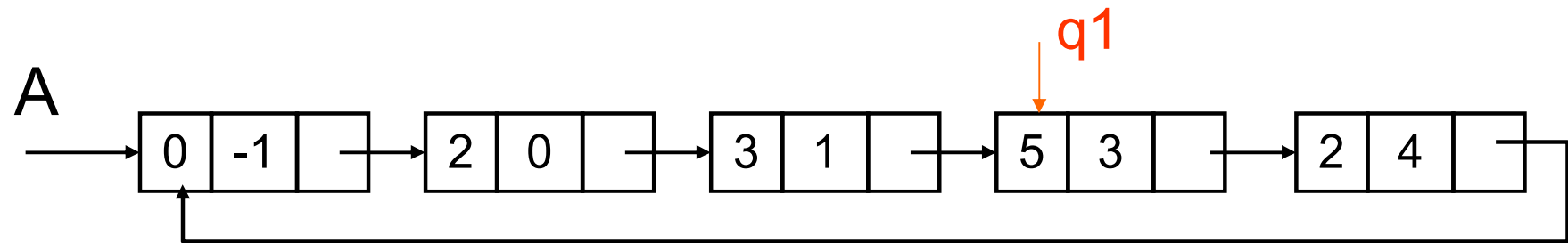
Case



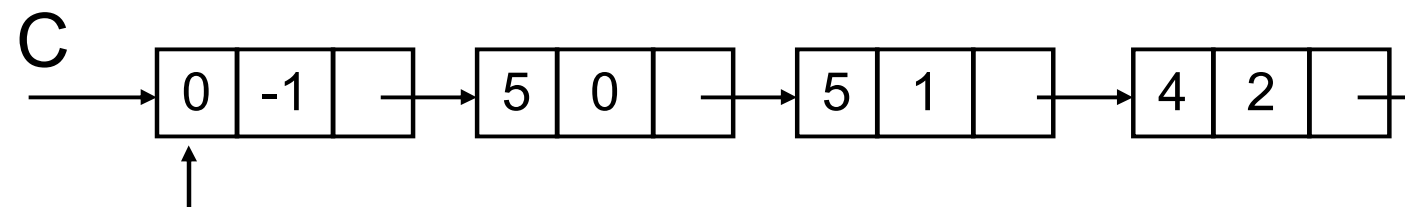
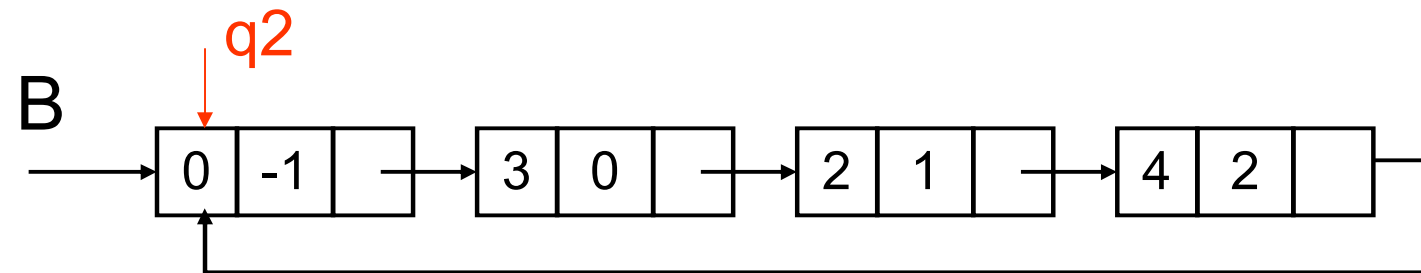
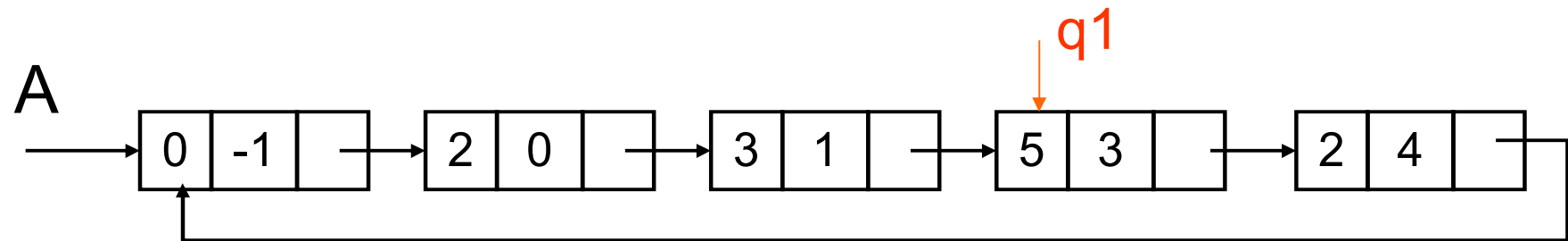
Case



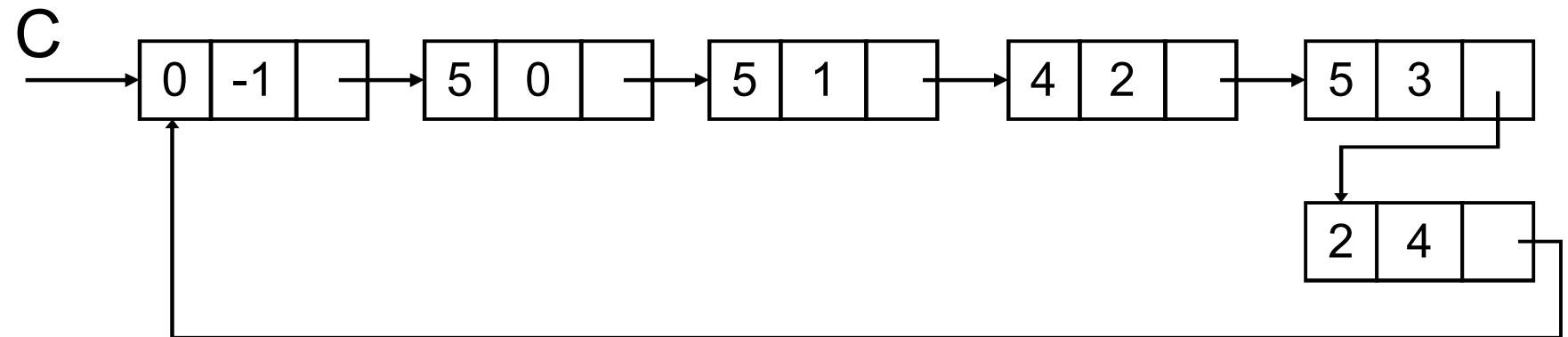
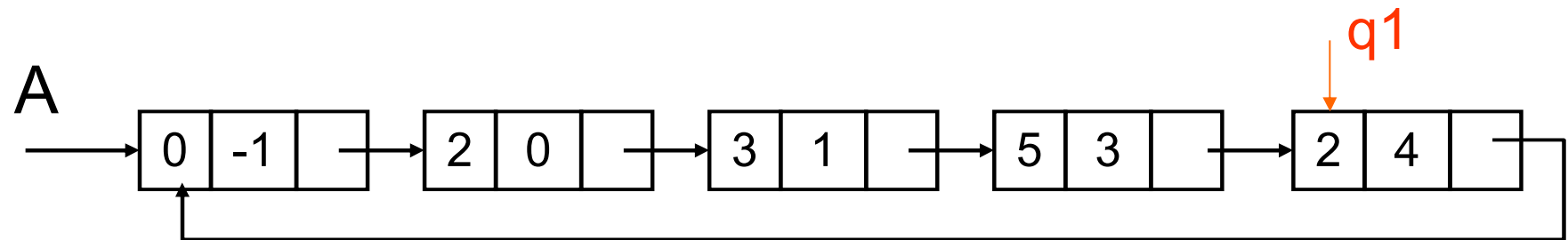
Case



Case

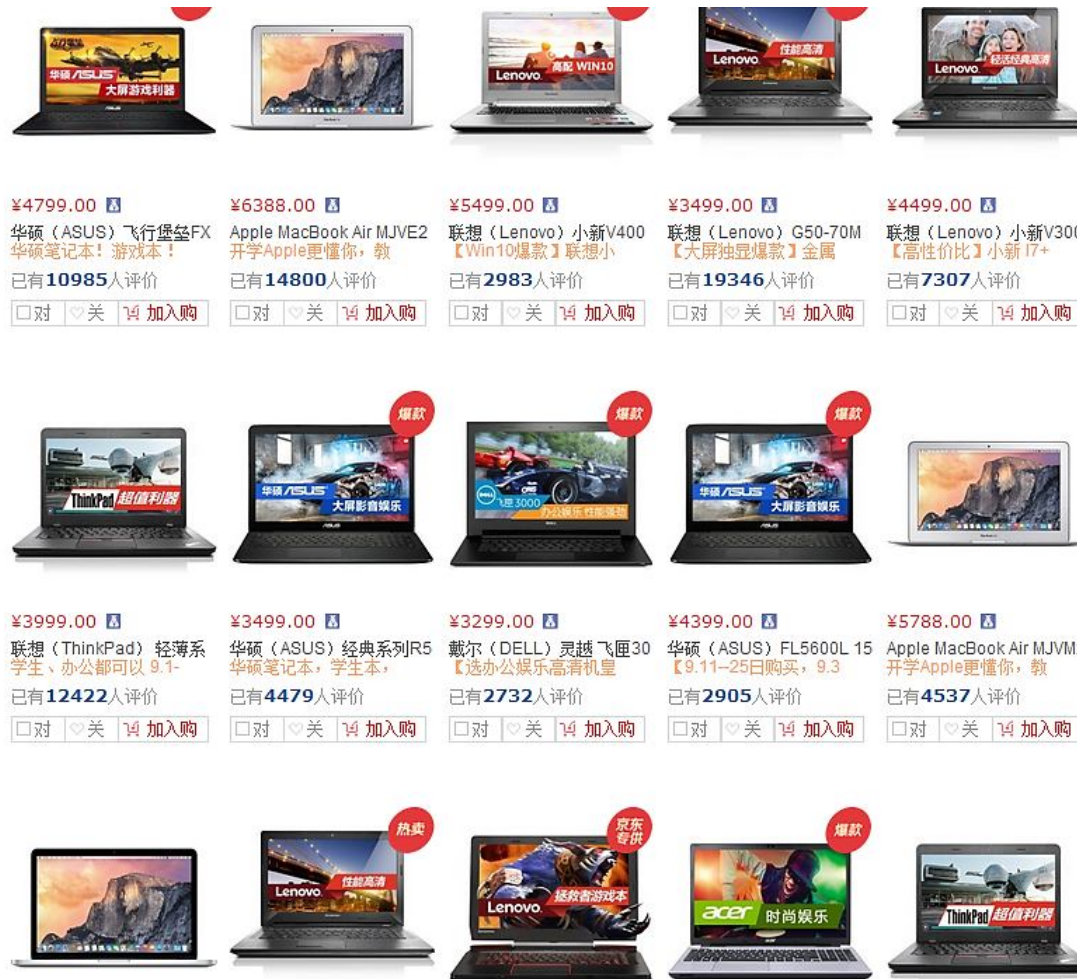


Case



Application in practices

- Product list



Sequence List
or
Linked List
?

Application in practices

- Data retrieval





Application in practices

- Shopping cart

JD.COM 购物车

自营 搜索

全部商品 4 配送至: 广东广州市广州大学城

<input checked="" type="checkbox"/> 全选	商品	单价(元)	数量	小计(元)	操作
<input checked="" type="checkbox"/>	京东自营				
<input checked="" type="checkbox"/>	活动商品已购满60.00元，可加价换购商品2件，还可换购2件 > 换购商品 去凑单 >				79.00
<input checked="" type="checkbox"/>	 罗技 (Logitech) MK120 键鼠套装 黑色	79.00 促销优惠	- 1 + 有货	79.00	删除 移到我的关注
<input checked="" type="checkbox"/>	活动商品已购满1.20元，可加价换购商品1件，还可换购1件 > 换购商品 去凑单 >				457.00
<input checked="" type="checkbox"/>	 金士顿(Kingston)DDR3 1600 4GB 台式机内存	148.00 促销优惠	- 1 + 有货	148.00	删除 移到我的关注
<input checked="" type="checkbox"/>	 希捷 (Seagate) 1TB 7200转 64M SATA 6GB/秒 台式机硬盘 (ST1000DM003)	309.00 促销优惠	- 1 + 有货	309.00	删除 移到我的关注

Summary

- Definition and operations
 - Sequence List
 - Linked List
- Implementation
- Applications

谢谢！

