第二章 线性表

本章内容：

- **2.1 线性表的概念**
- **2.2 顺序表**
- **2.3 单链表**
- **2.4 单链表的变形：循环链表和双向链表**
- **2.5 单链表的应用：多项式**

数据结构关注的三个方面

逻辑结构	存储结构	运算
数据元素之间的逻辑关系	数据逻辑结构的物理存储映象	数据结构上的运算，及实现方法（查找、插入、删除、更新等）

线性表的逻辑结构

- 线性表的定义

由n(n≧0)个数据元素(结点) $a_1, a_2, \ldots, a_n$组成的有限序列。其中数据元素个数n定义为表的长度。当n=0时称为空表，非空线性表(n>0)记作：

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

- a_i是表项，n是表长度。
- 第一个表项是表头，最后一个是表尾

例1 26个英文字母组成的字母表
（A，B，C、...、Z）

例2 某校从2010年到2016年各种型号的计算机拥有量变化情况。
（2000,3000,3800,4000,4500,5000,5600）

线性表的逻辑结构

注：一个数据元素可以由若干数据项（Item）组成，在这种情况下，通常把数据元素称为记录（Record），含有大量记录的线性表又称为文件（File）。

例3某高校学生的基本信息表，每个学生的基本信息为一个记录，该记录由学号、姓名、性别、专业以及住址5个数据项组成。

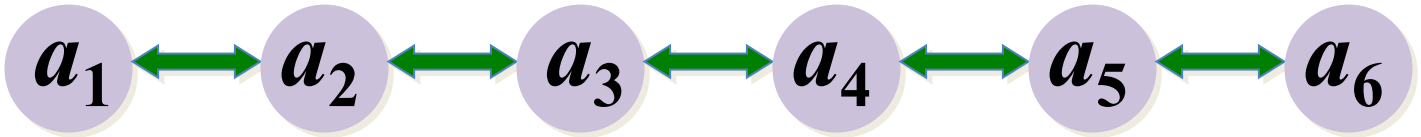
学号	姓名	性别	专业	住址
04180101	侯亮平	男	计算机科学与技术	北京
04180102	高小琴	女	计算机科学与技术	深圳
04180103	陆亦可	女	计算机科学与技术	珠海
04180104	陈海	男	计算机科学与技术	上海
04180105	李达康	男	计算机科学与技术	杭州
04180106	高育良	男	计算机科学与技术	南京
04180107	赵东来	男	计算机科学与技术	武汉
04180108	陈岩石	男	计算机科学与技术	重庆
04180109	沙瑞金	男	计算机科学与技术	珠海

线性表的逻辑特征

(1) 对非空的线性表，有且仅有一个开始结点a_1，它没有直接前趋，而仅有一个直接后继a_2；

(2) 有且仅有一个终端结点a_n，它没有直接后继，而仅有一个直接前趋a_{n-1}；

(3) 其余的内部结点a_i (2≦i≦n-1)都有且仅有一个直接前趋a_{i-1}和一个直接后继a_{i+1}。



注：有序表和无序表

线性表的存储方式

- 顺序存储方式 —— 顺序表
- 链表存储方式 —— 链表

第二章 线性表

2.1 线性表的概念

2.2 顺序表

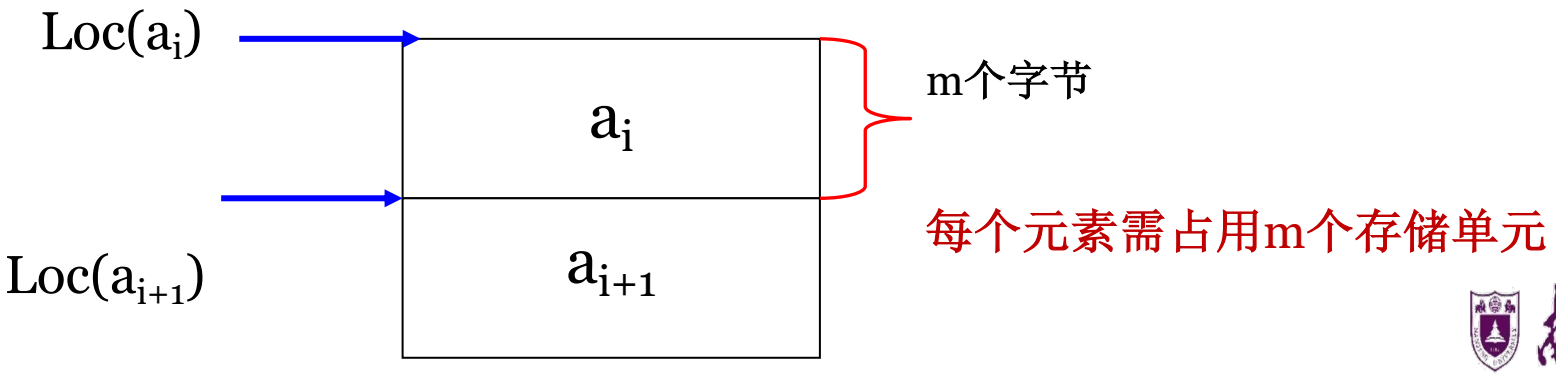
2.3 单链表

2.4 单链表的变形：循环链表和双向链表

2.5 单链表的应用：多项式

2.2 顺序表

- 顺序表：把线性表的结点按逻辑顺序依次存放在一组地址连续的存储单元里。
- 各表项的逻辑顺序与物理顺序一致
- 线性表中第i+1个数据元素的存储位置$Loc(a_{i+1})$和第i个数据元素的存储位置$Loc(a_i)$之间满足下列关系：$Loc(a_{i+1})=Loc(a_i)+m$

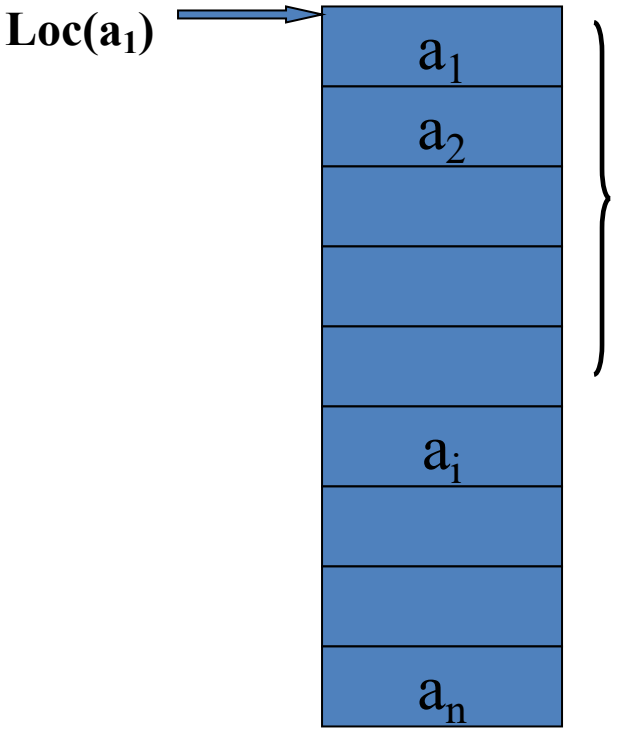


南京大学 NANJING UNIVERSITY

顺序表的特点

- 各表项的逻辑顺序与物理顺序一致
- 对各个表项可以顺序访问，也可以随机访问

线性表的第i个数据元素a_i的存储位置为 ：



$$Loc(a_i)=(i-1)*m+Loc(a_1)$$

$$=Loc(a_1)-m+i*m$$

由于$Loc(a_1)$和m都是已知的

所以：$V_0= Loc(a_1)-m$

$$Loc(a_i)=V_0+i*m$$

南京大學 NANJING UNIVERSITY

顺序表的静态存储和动态存储

```
#define maxSize 100
typedef int T;
typedef struct {
    T data[maxSize];      //顺序表的静态存储表示
    int n;
} SeqList;

typedef int T;
typedef struct {
    T *data;              //顺序表的动态存储表示
    int maxSize, n;
} SeqList;
```

顺序表(SeqList)类的定义

```
#include <iostream.h>                    //定义在“seqList.h”中
#include <stdlib.h>
#include "LinearList.h"
const int defaultSize = 100;
template <class T, class E>
class SeqList: public LinearList<T, E> {
protected:
    E *data;                  //存放数组
    int maxSize;              //最大可容纳表项的项数
    int n;                    //当前已存表项数
    void reSize(int newSize); //改变数组空间大小
```

数据成员

顺序表(SeqList)类的定义（续）

```
public:
    SeqList(int sz = defaultSize);                  //构造函数
    SeqList(SeqList<T,E>& L);                       //复制构造函数
    ～SeqList() {delete[ ] data;}                   //析构函数
    int Size() const {return maxSize;}              //求表最大容量
    int Length() const {return n;}                  //计算表长度
    int Search(T x) const;
        //搜索x在表中位置，函数返回表项序号
    int Locate(int i) const;
        //定位第 i 个表项，函数返回表项序号
    bool Insert(int i, E x);                        //插入
    bool Remove(int i, E& x);                       //删除
};
```

顺序表的构造函数

```
#include <stdlib.h>      //操作“exit”存放在此
#include “seqList.h”     //操作实现放在“seqList.cpp”

template <class T, class E>
SeqList<T, E>::SeqList(int sz) {
    if (sz > 0) {
        maxSize = sz;  n = 0;                    //初始化
        data = new E[maxSize];                   //创建表存储数组
        if (data == NULL)                        //动态分配失败
          { cerr << "存储分配错误！" << endl;
            exit(1); }
    }
};
```

南京大學 NANJING UNIVERSITY

复制构造函数

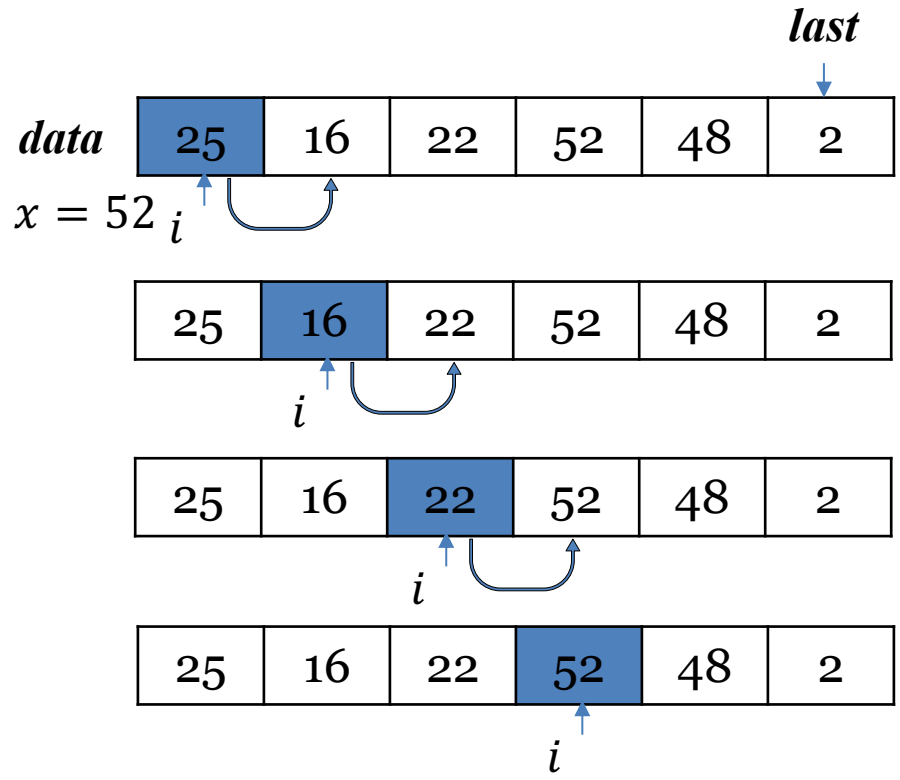
```
template <class T, class E>
SeqList<T, E>::SeqList ( SeqList<T, E>& L ) {
    maxSize = L.Size();   n = L.Length();
    data = new E[maxSize];          //创建存储数组
    if (data == NULL)               //动态分配失败
       {cerr << "存储分配错误！" << endl;
        exit(1);}
    for (int i = 1; i <= n; i++)    //传送各个表项
        data[i-1] = L.getData(i);
};
```

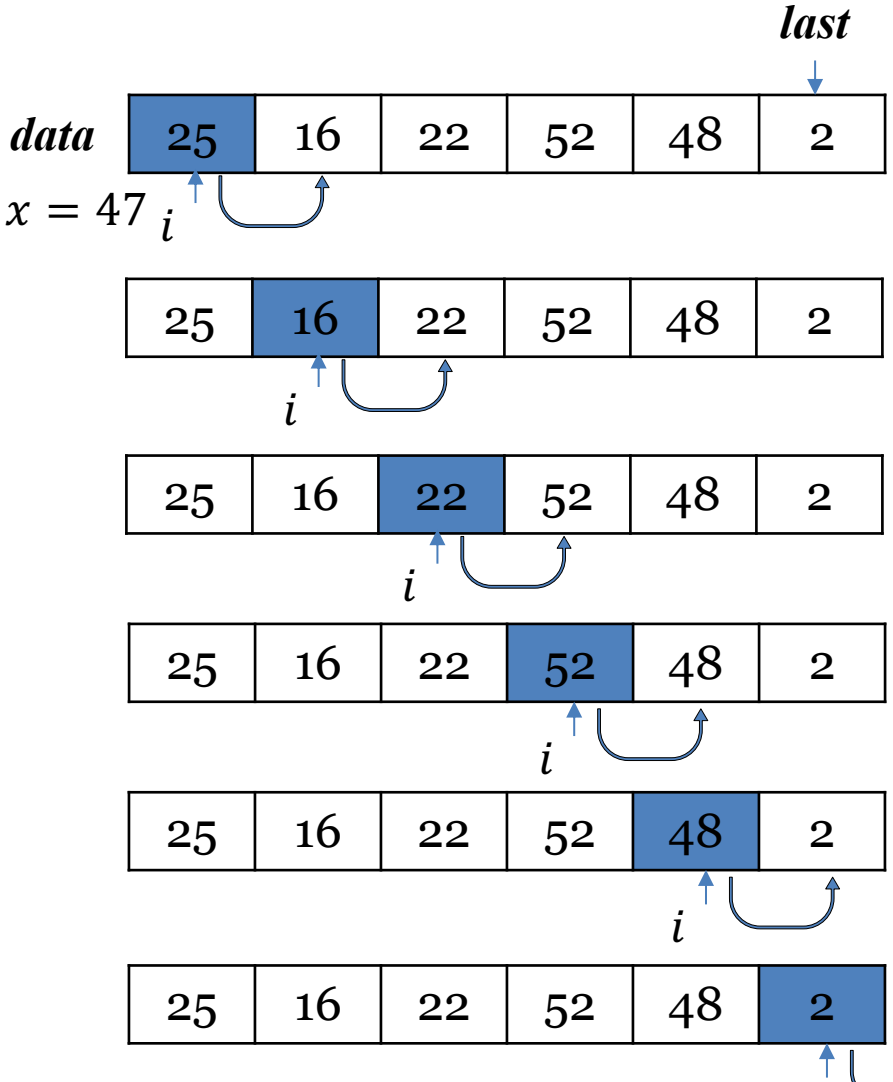
顺序表的搜索算法

在表中顺序搜索与给定值 x 匹配的表项，找到则函数返回该表项是第几个元素，否则函数返回0

```
template <class T, class E>
int SeqList<T, E>::search(T& x) const {
    for (int i = 1; i <= n; i++)                    //顺序搜索
        if ( data[i-1] == x ) return i;
                                //表项序号和表项位置差1
    return 0;                   //搜索失败
};
```

顺序表的查找





$x = 52$ （成功）

$x = 47$ （失败）

南京大学 NANJING UNIVERSITY

顺序查找数据的时间代价（比较次数分析）

ACN(Average Comparing Number)

- 搜索成功：位置i的查找概率p_i，找到该表项时的数据比较次数c_i

平均比较次数 $ACN = \sum_{i=0}^{n-1} p_i \times c_i$

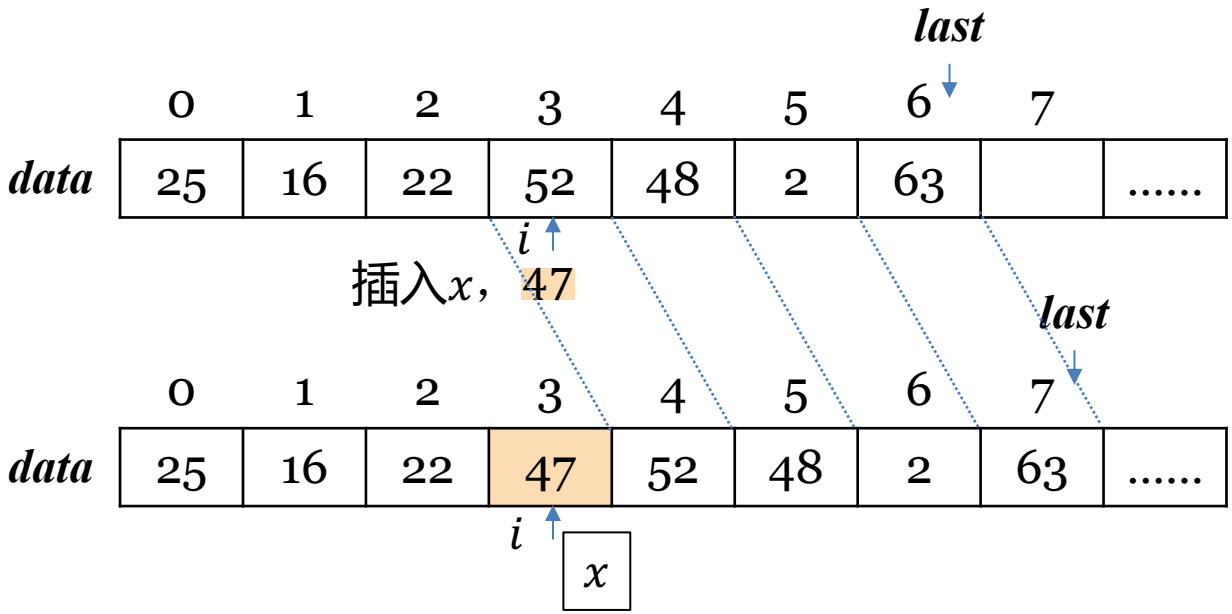
若搜索概率p_i相等，则

$$ACN = \frac{1}{n}\sum_{i=0}^{n-1}(i+1) = \frac{1}{n}(1+2+\cdots+n) =$$

$$= \frac{1}{n} * \frac{(1+n)*n}{2} = \frac{1+n}{2}$$

- 搜索不成功：数据比较n次

顺序表的插入


last
0 1 2 3 4 5 6 7
data
25 16 22 52 48 2 63 .....
i
插入x，47
last
0 1 2 3 4 5 6 7
data
25 16 22 47 52 48 2 63 .....
i
x

表项的插入算法

//将新元素x插入到表中第i (1≤i≤n+1) 个表项位
//置。函数返回插入成功的信息

```
template <class T, class E>
bool SeqList<T, E>::Insert (int i, E x) {
if (n == maxSize) return false;          //表满
      if (i < 1 || i > n+1) return false;     //参数i不合理
    for (int j = n; j >= i; j--)              //依次后移
        data[j] = data[j-1];
    data[i-1] = x;          //插入第i表项在data[i-1]处
      n++;   return true;                     //插入成功
};
```

南京大學 NANJING UNIVERSITY

顺序表插入的时间代价（移动次数）

在表中第 i 个位置插入，从data[i-1] 到data [n-1] 成块后移，移动n-1-(i-1)+1 = n-i+1项

平均数据移动次数AMN(Average Moving Number)在各表项插入概率相等时为

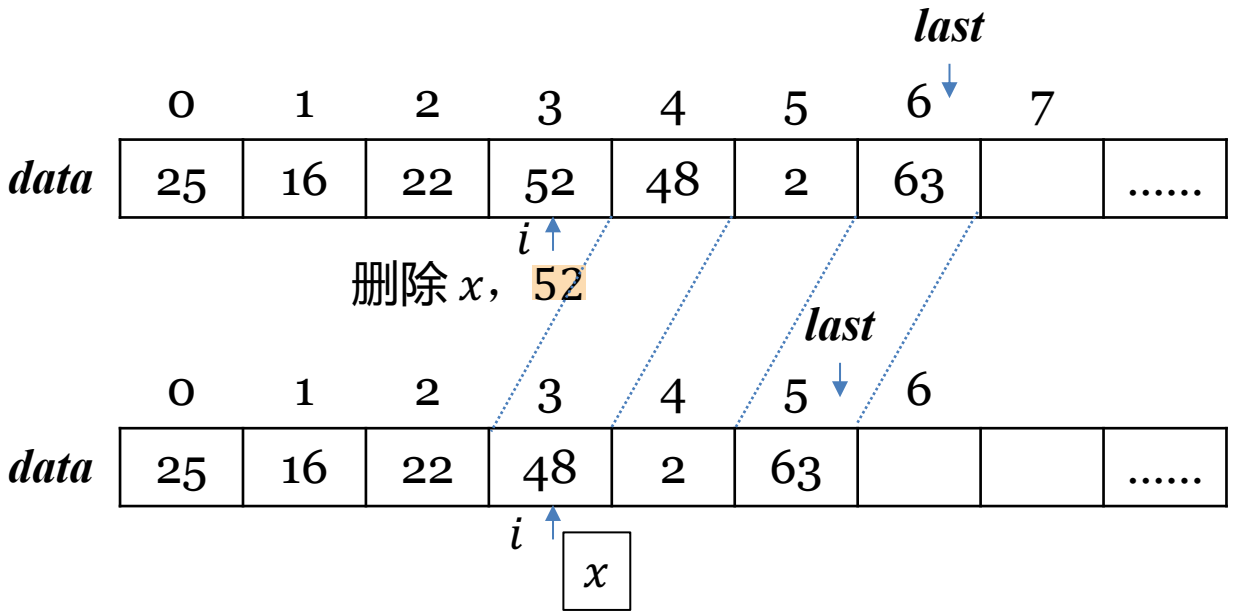
$$\mathrm{AMN}=\frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1)=\frac{1}{n+1}(n+\cdots+1+0)$$

$$=\frac{1}{(n+1)}\frac{n(n+1)}{2}=\frac{n}{2}$$

在插入时有n+1个插入位置，平均移动n/2项

南京大學 NANJING UNIVERSITY

顺序表的表项删除


last
0 1 2 3 4 5 6 7
data
25 16 22 52 48 2 63 ……
i
删除 x，52
last
0 1 2 3 4 5 6
data
25 16 22 48 2 63 ……
i
x

表项的删除算法

//从表中删除第 i (1≤i≤n) 个表项，通过引用型参
//数 x 返回被删元素。函数返回删除成功信息

```
template <class T, class E>
bool SeqList<T, E>::Remove (int i, E& x) {
        if (n == 0) return false;                  //表空
    if (i < 1 || i > n) return false;              //参数i不合理
    x = data[i-1];
    for (int j = i; j <= n-1; j++)          //依次前移，填补
        data[j-1] = data[j];
    n--;   return true;
};
```

南京大學 NANJING UNIVERSITY

顺序表的表项删除的时间代价(移动次数)

删除第 i 个表项，需将第 i+1 项到第 n 项全部前移，需前移的项数为　n-(i+1)+1 = n-i

平均数据移动次数AMN（Average Moving Number）在n个表项删除概率相等时为

$$\mathrm{AMN}=\frac{1}{n}\sum_{i=1}^{n}(n-i)=\frac{1}{n}\frac{(n-1)n}{2}=\frac{n-1}{2}$$

在删除时有n个删除位置，平均移动(n-1)/2项

南京大學 NANJING UNIVERSITY

用顺序表实现集合的“并”运算

```
template <class Type>
void Union ( SeqList<Type> & LA,
                        SeqList<Type> & LB ) {
    int n = LA.Length ( );
    int m = LB.Length ( );
    for ( int i = 1; i <= m; i++ ) {
            Type x = LB.getData(i);
                                        //在LB中取一元素
            int k = LA.Search (x);     //在LA中搜索它
            if ( k == 0 )                //若未找到插入它
           { LA.Insert (n, x);  n++; }
    }
}
```

用顺序表实现集合的“交”运算

```
template <class Type>
void Intersection ( SeqList<Type> & LA,
                    SeqList<Type> & LB ) {
    int n = LA.Length ( );
    int m = LB.Length ( );  int i = 1;
    while ( i <= n ) {
        Type x = LA.getData (i);    //在LA中取一元素
            int k = LB.Search (x);      //在LB中搜索它
            if ( k == 0 ) { LA.Remove (i,x);  n--;}
                                        //未找到在LA中删除它
            i++;
    }
}
```

南京大學 NANJING UNIVERSITY

思考：以上的集合的“交”和“并”运算的时间复杂性是多少？

顺序表的特点

- 特点：逻辑关系上相邻的两个数据元素在物理位置上也相邻。
- 优点：节省存储空间。由于结点之间的相邻逻辑关系可以用物理位置上的相邻关系表示，因此不需增加额外的存储空间来表示此关系。存取速度快。
- 缺点：插入、删除等操作时需要移动大量数据

第二章 线性表

南京大学 NANJING UNIVERSITY

单链表

连续存储方式（顺序表）

- 特点：存储利用率高，存取速度快
- 缺点：插入、删除等操作时需要移动大量数据

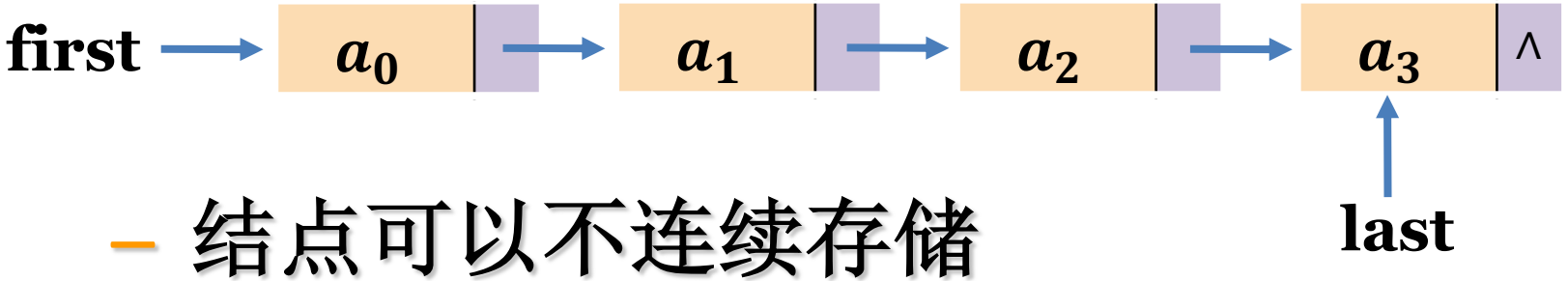
链式存储方式（链表）

- 特点：适应表的动态增长和删除
- 缺点：需要额外的指针存储空间

单链表

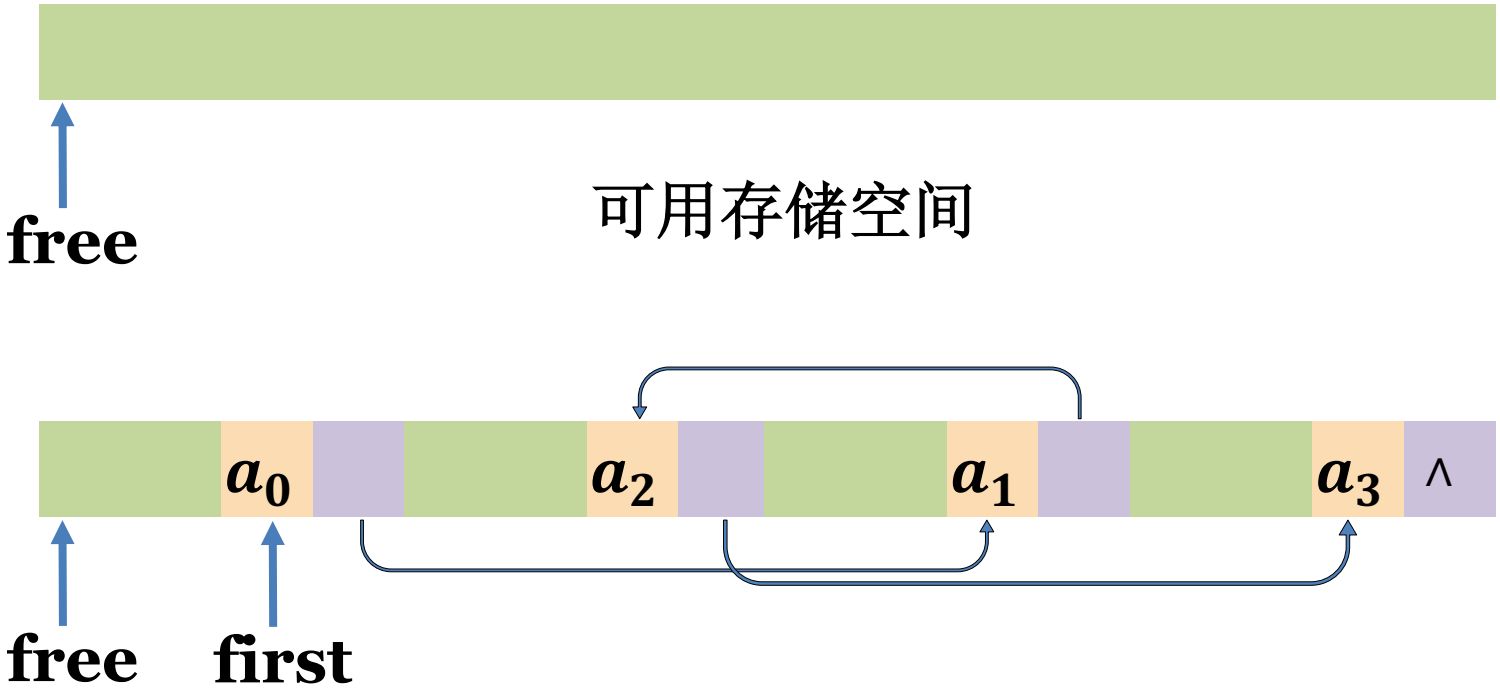
- 单链表的特点
  - 每个元素(表项)由结点(*Node*)构成。

  - 线性结构



  - 结点可以不连续存储
  - 表长度可方便地扩充

单链表的存储映像



经过一段运行后的单链表结构

南京大学 NANJING UNIVERSITY

单链表的类定义

- 多个类表达一个概念(单链表)。
  - 链表结点(*ListNode*)类
  - 链表(*List*)类

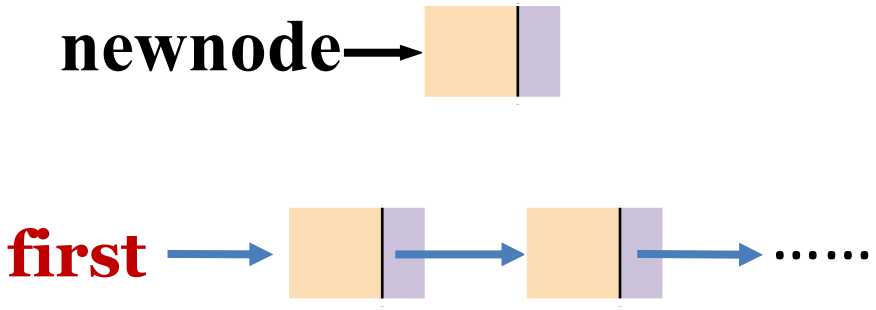
链表类定义

```
class List;                          //复合方式

class ListNode {                     //链表结点类
friend class List;                   //链表类为其友元类
private:
    int data;                        //结点数据, 整型
    ListNode * link;                 //结点指针
};

class List {                         //链表类
private:
    ListNode *first ;                //表头指针
};
```

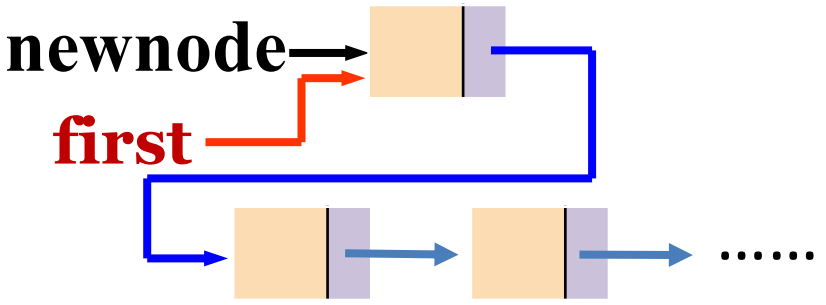
单链表中的插入

- 第一种情况：在链表最前端插入

```
newnode->link = first ;
first = newnode;
```



（插入前）

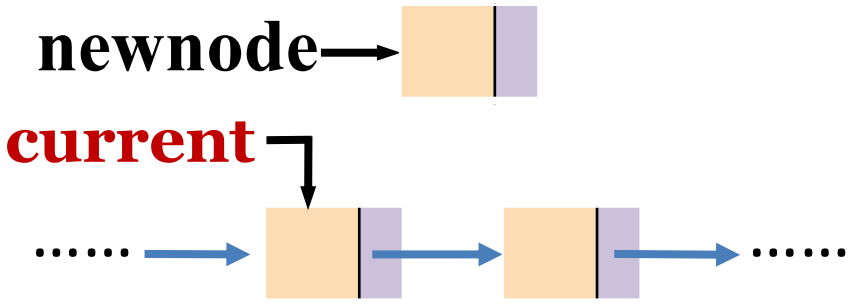


（插入后）

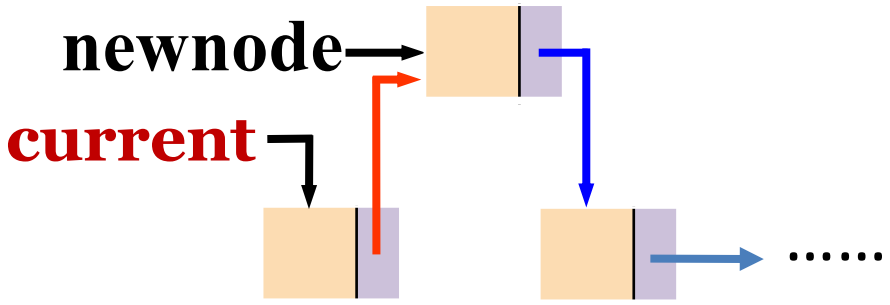
单链表中的插入

- 第二种情况：在链表中间插入

```
newnode->link = current->link;
current->link = newnode ;
```



（插入前）

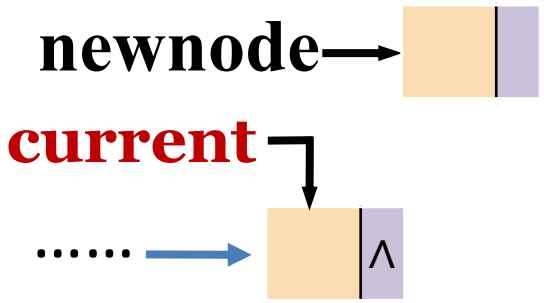


（插入后）

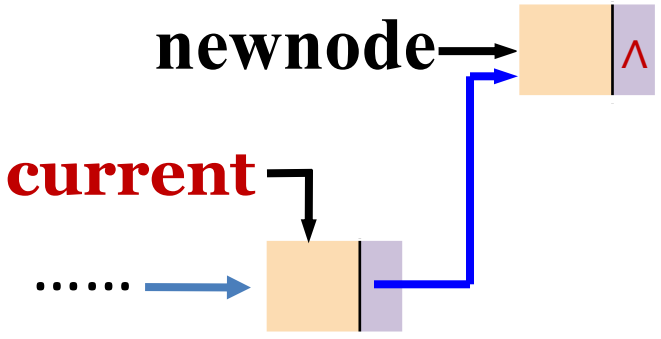
单链表中的插入

- 第三种情况：在链表末尾插入

```
newnode->link = current->link;
current->link = newnode ;
```



（插入前）



（插入后）

单链表的插入算法

```
bool List::Insert(int i, int x) {
//将新元素 x 插入到第 i 个结点之后。i 从1开始，
//i = 0 表示插入到首元结点之前。
   if (first == NULL || i == 0) {    //空表或首元结点前
        LinkNode *newNode = new LinkNode(x);
         //建立一个新结点
        newNode->link = first;  first = newNode;
         //新结点成为首元结点
   }
   else {                    //否则，寻找插入位置
      LinkNode *current = first;   int k = 1;
```

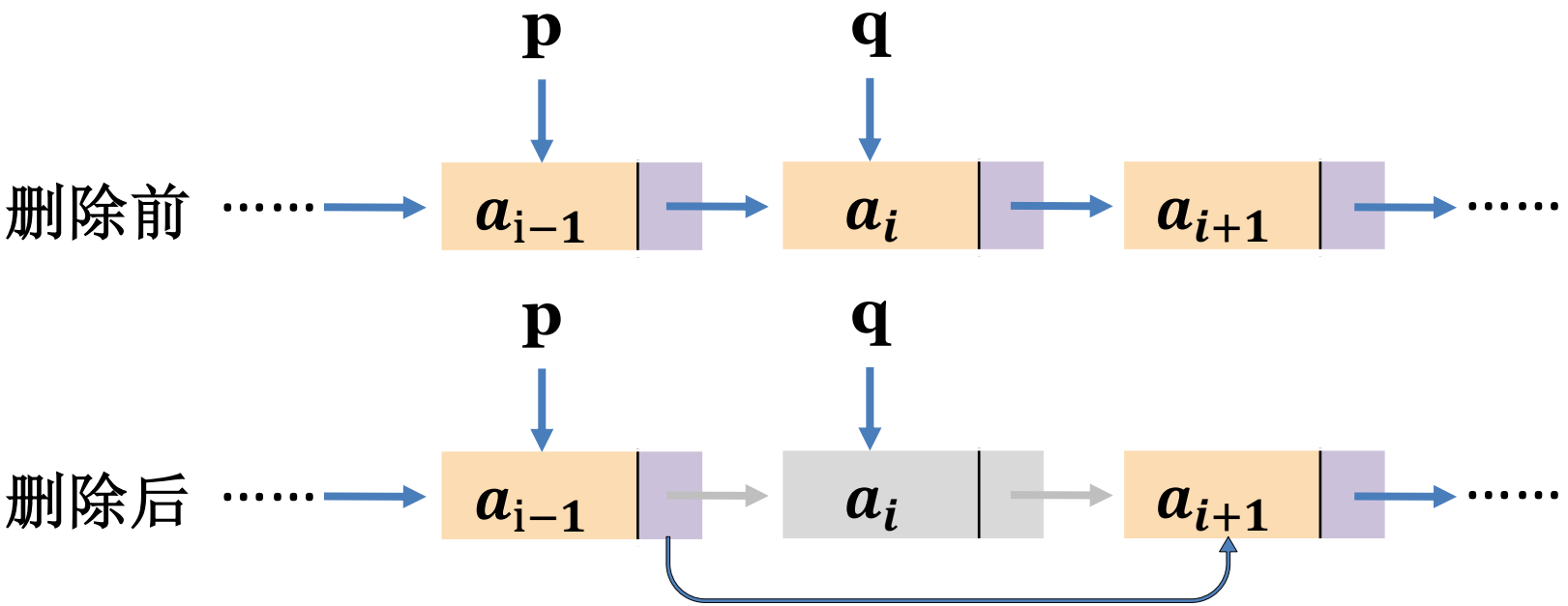
南京大學 NANJING UNIVERSITY

```
    while (k < i && current != NULL)    //找第i结点
        { current = current->link;  k++; }
    if (current == NULL && first != NULL)    //链短
        {cerr << "无效的插入位置!\n";  return false;}
  else {                    //插入在链表的中间
        LinkNode *newNode = new LinkNode(x);
        newNode->link = current->link;
        current->link = newNode;
       }
   }
   return true;
};
```

南京大學 NANJING UNIVERSITY

单链表的删除算法

- 第一种情况: 删除表中第一个元素
- 第二种情况: 删除表中或表尾元素



在单链表中删除含a_i的结点

南京大学 NANJING UNIVERSITY

单链表的删除算法

```
bool List::Remove (int i, int& x) {
//将链表中的第 i 个元素删去, i 从1开始。
    LinkNode *del;            //暂存删除结点指针
    if (i <= 1)  { del = first;  first = first->link; }
    else {
     LinkNode *current = first;  k = 1;   //找i-1号结点
        while (k < i-1 && current != NULL)
        { current = current->link;   k++; }
        if (current == NULL || current->link == NULL)
        {cout << "无效的删除位置!\n";  return false;}
```

南京大学 NANJING UNIVERSITY

```
        del = current->link;          //删中间/尾结点
        current->link = del->link;
    }
    x = del->data;  delete del;  //取出被删结点数据
    return true;
};
```

单链表的优缺点

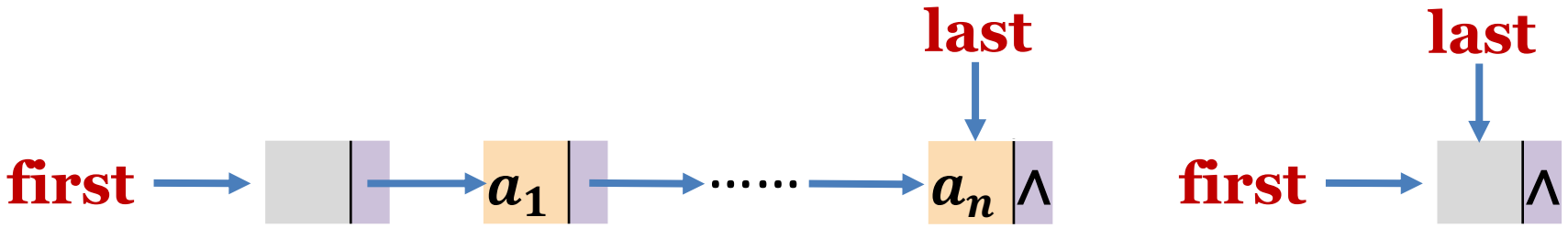
1）实现单链表的插入和删除算法，不需要移动元素，只需修改结点指针，比顺序表方便。

2）情况复杂，要专门讨论空表和在表头插入的特殊情形。

3）寻找插入或删除位置只能沿着链顺序检测。

带附加头结点（表头结点）的单链表

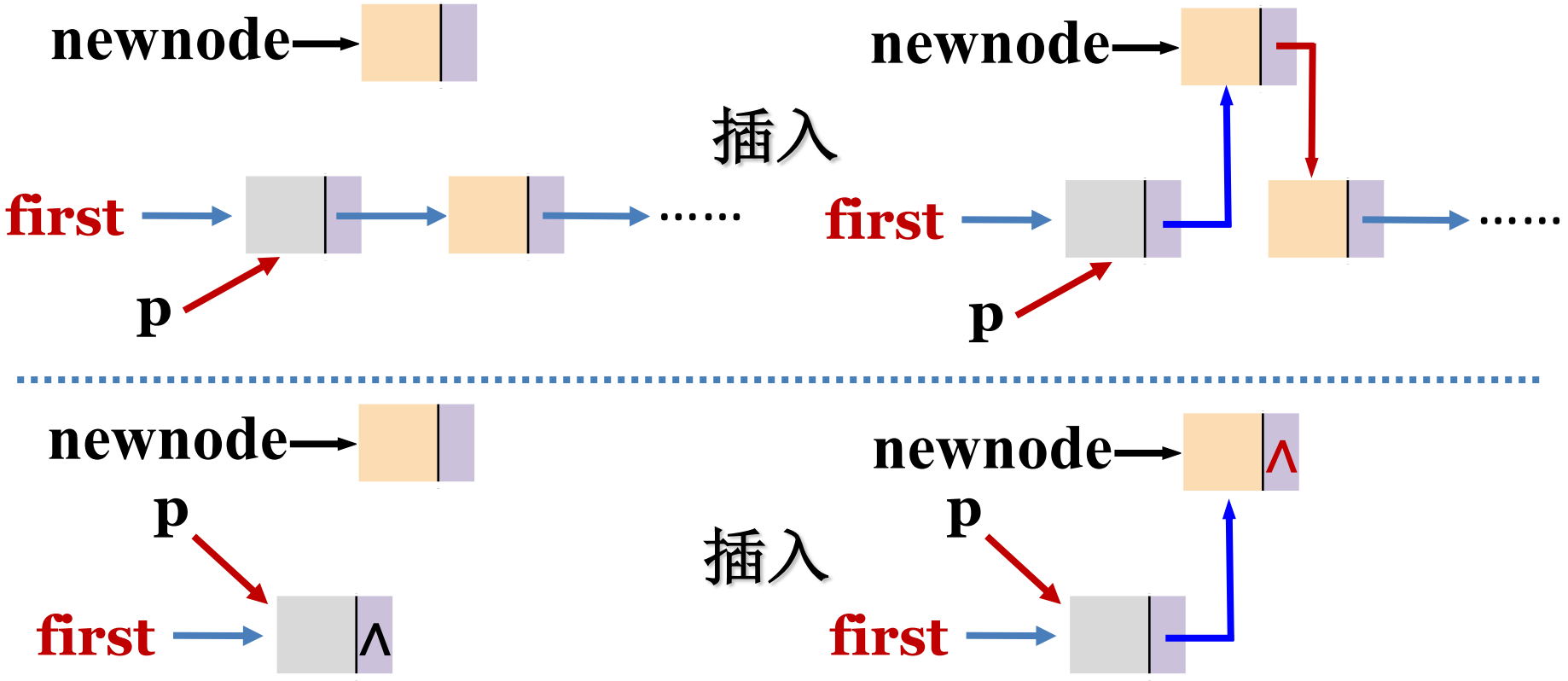
- 表头结点位于表的最前端，本身不带数据，仅标志表头。
- 设置表头结点的目的是统一空表与非空表的操作，简化链表操作的实现。



非空表　　空表

南京大學 NANJING UNIVERSITY

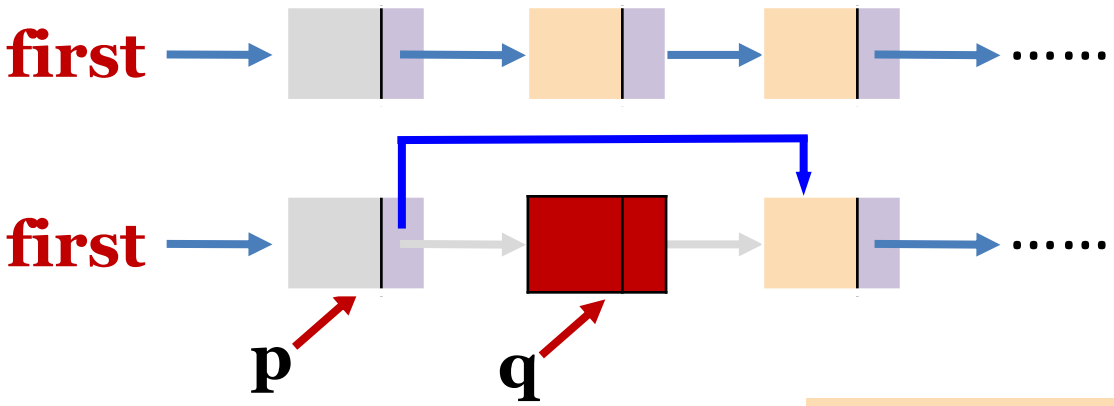
在带表头结点的单链表最前端插入新结点




```
newnode->link = p->link;
p->link = newnode;
```

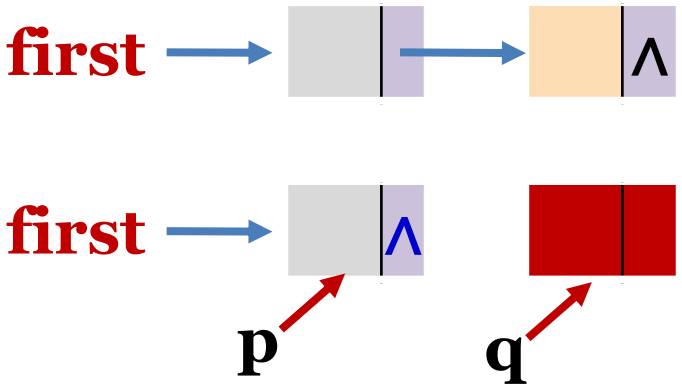
南京大學 NANJING UNIVERSITY

从带表头结点的单链表中删除最前端的结点



(非空表)

```
q = p->link;
p->link = q->link;
delete q;
```



(空表)

南京大學 NANJING UNIVERSITY

用模板定义的单链表

```
template <class T>   //定义在“LinkedList.h”
struct LinkNode {                    //链表结点类的定义
    E data;                          //数据域
    LinkNode<T> *link;     //链指针域
    LinkNode( ) { link = NULL; }     //构造函数
    LinkNode(const T& item, LinkNode<T> *ptr
    =NULL)
        { data = item;  link = ptr; }     //构造函数

};
```

数据成员

南京大學 NANJING UNIVERSITY

```
template <class T>
class List : public LinearList<T> {
//单链表类定义, 不用继承也可实现
protected:
    LinkNode<T> *first;          //表头指针
public:
    List( ){ first = new LinkNode<T>; }  //构造函数
    List(const T& x) {first = new LinkNode<T>(x); }
    List( List<T>& L);                   //复制构造函数
    ~List( ){makeEmpty( ) ;}             //析构函数
    void makeEmpty( );                   //将链表置为空表
    int Length( ) const;                 //计算链表的长度
```

数据成员

南京大學 NANJING UNIVERSITY

```
LinkNode<T, E> *Search(T x);        //搜索含x元素
  LinkNode<T, E> *Locate(int i);    //定位第i个元素
T *getData(int i);                  //取出第i元素值
void setData(int i, T & x);         //更新第i元素值
  bool Insert (int i, T & x);       //在第i元素后插入
  bool Remove(int i, T & x);        //删除第i个元素
bool IsEmpty() const                //判表空否
  { return first->link == NULL ? true : false; }
LinkNode<T> *getHead( ) const { return first; }
void setHead(LinkNode<T> *p ) { first = p;}
  void Sort();                      //排序
};
```

单链表的插入算法

```
template <class T>
bool List<T>::Insert (int i, T& x) {
//将新元素 x 插入在链表中第 i 个结点之后。
   LinkNode<T> *current = Locate(i);
     if (current == NULL) return false;          //无插入位置
   LinkNode<T> *newNode =
                new  LinkNode<T>(x);      //创建新结点
    if (newNode == NULL)                  //动态分配失败
       {cerr << "存储分配错误！" << endl;  exit(1);}
    newNode->link = current->link;        //链入
    current->link = newNode;
     return true;                                  //插入成功
};
```

南京大學 NANJING UNIVERSITY

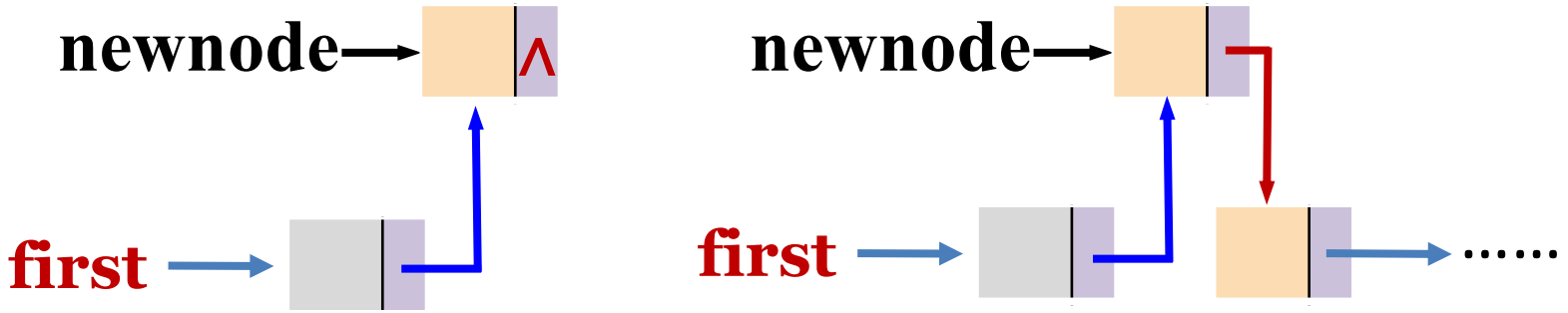
单链表的删除算法

```
template <class T>
bool List<T>::Remove (int i, T& x ) {
//删除链表第i个元素, 通过引用参数x返回元素值
   LinkNode<T> *current = Locate(i-1);
     if ( current == NULL || current->link == NULL)
       return false;          //删除不成功
   LinkNode<T> *del = current->link;
   current->link = del->link;
     x = del->data;          delete del;
   return true;
};
```

南京大學 NANJING UNIVERSITY

前插法建立单链表

- 从一个空表开始，重复读入数据：
  - 生成新结点
  - 将读入数据存放到新结点的数据域中
  - 将该新结点插入到链表的前端
- 直到读入结束符为止。

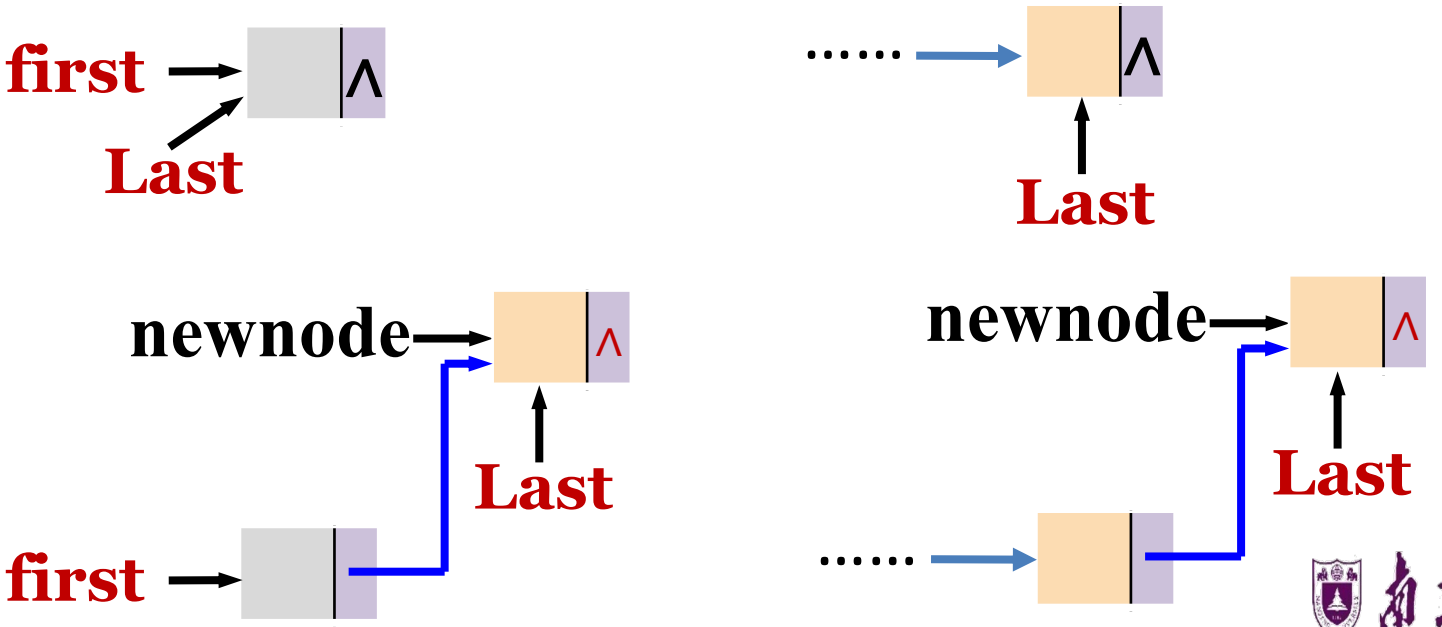


```
template <class T>
void inputFront (T endTag, List<T>& L) {
    LinkNode<T> *newNode, *newF; T val;
    newF = new LinkNode<T>;
      L.setFirst (newF);      //first->link默认值为NULL
    cin >> val;
    while (val != endTag) {
        newNode = new LinkNode<T>(val);
        newNode->link = newF->link;          //插在表前端
        newF->link = newNode;
        cin >> val;
    }
};
```

南京大學 NANJING UNIVERSITY

后插法建立单链表

- 每次将新结点加在插到链表的表尾
- 设置一个尾指针 last，总是指向表中最后一个结点，新结点插在它的后面
- 尾指针last初始时置为指向表头结点地址



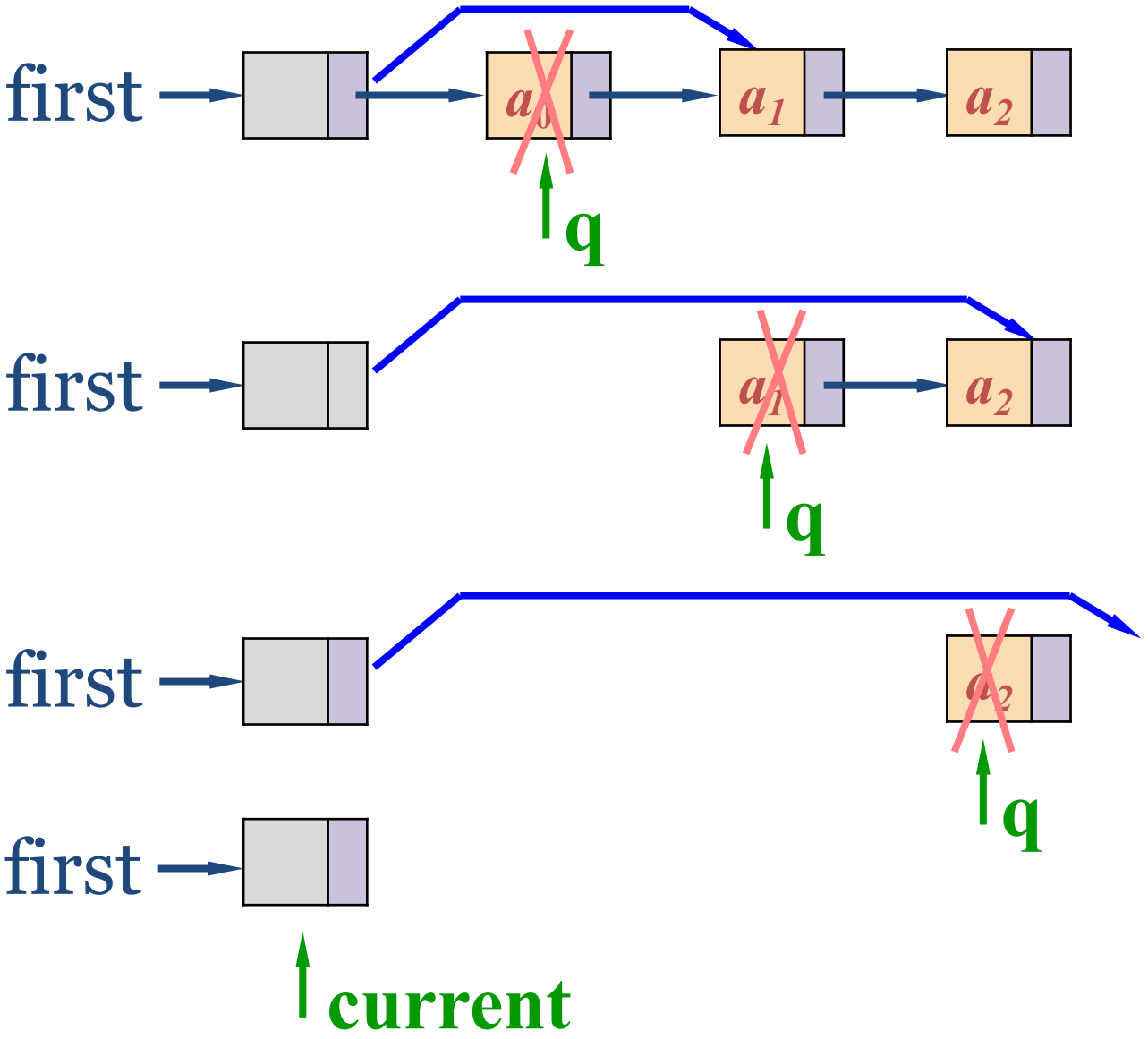
南京大學 NANJING UNIVERSITY

```
template <class T>
void inputRear ( T endTag, List<T>& L ) {
    LinkNode<T> *newNode, *last;  T val;
    last = new LinkNode<T>;          //建立链表的头结点
    L.setFirst(last);                //为链表L的first赋值
    cin >> val;
    while ( val != endTag ) {        //last指向当前的表尾
        newNode = new LinkNode<T>(val);
        last->link = newNode;   last = newNode;
        cin >> val;                  //插入到表末端
    }
    last->link = NULL;               //表收尾
};
```

南京大學 NANJING UNIVERSITY

链表置空算法（保留表头结点）

```
template <class T>
void List<T>::makeEmpty() {
    LinkNode<T> *q;
    while (first->link != NULL) {
        q = first->link;              //保存被删结点
    first->link = q->link;    //从链上摘下该结点
        delete q;                     //删除
    }
};
```



南京大學
NANJING UNIVERSITY

链表的搜索算法

```
template <class T>
LinkNode<T> *List<T>::Search(T x) {
//在表中搜索含数据x的结点, 搜索成功时函数返
//该结点地址; 否则返回NULL。
    LinkNode<T> *current = first->link;
    while ( current != NULL && current->data != x )
                current = current->link;
        //沿着链找含x结点
    return current;
};
```

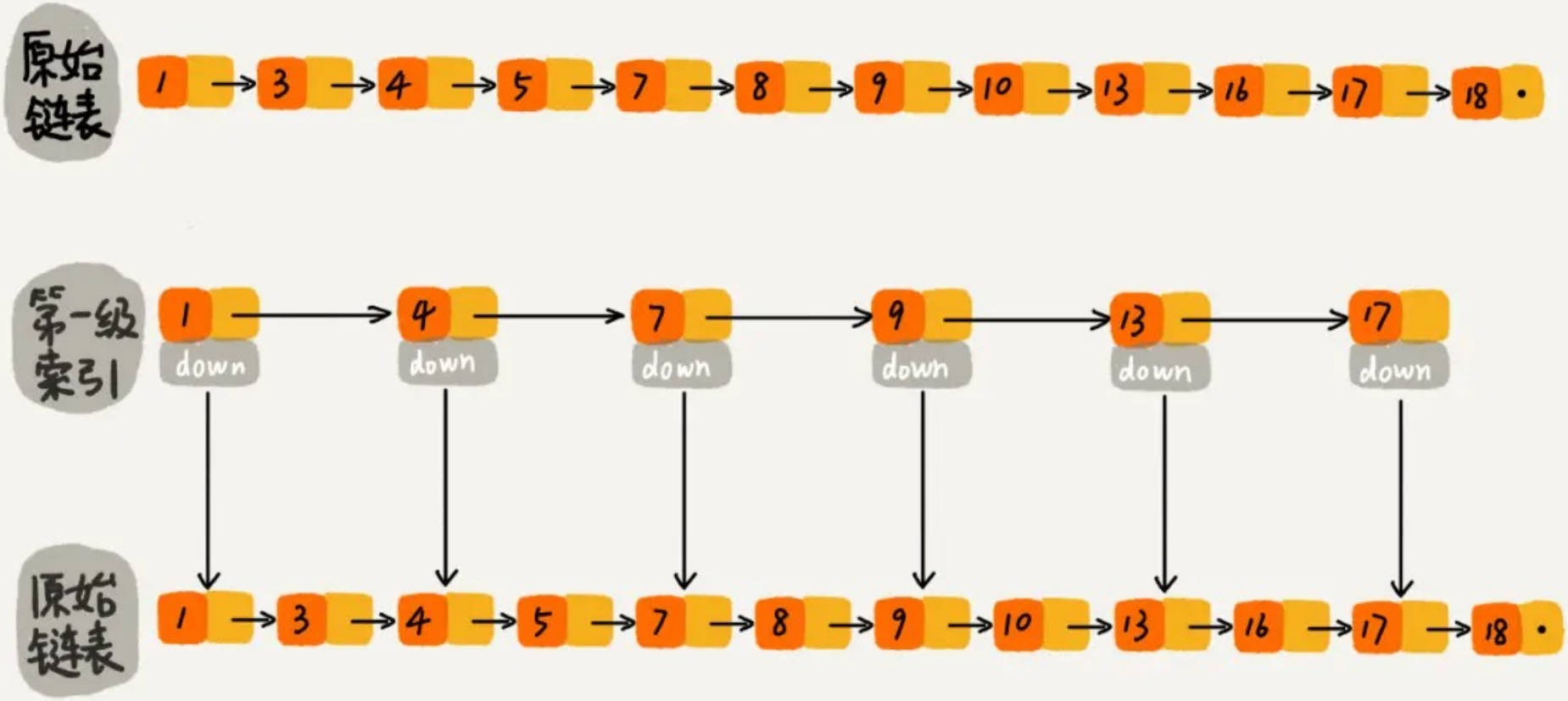
思考

问题一：既然定位到某个元素总需要O(N)，那插入和删除操作的优势在哪里呢？

- 取决于具体的问题是什么？

问题二：是否有办法可以缩短定位操作复杂度呢？

跳表



空间换时间，查找的复杂性可以降低到O(logN)

图片来源：https://www.jianshu.com/p/9d8296562806

作业：单链表的逆置

问题描述：输入一个链表，反转链表后，输出新链表的表头。

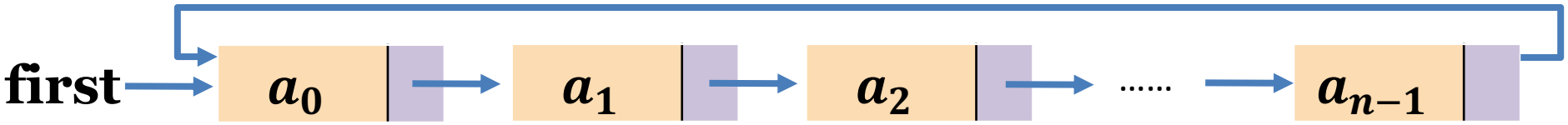
完善如下的函数。

```
ListNode * List::ReverseList(ListNode * head)
{
}
```

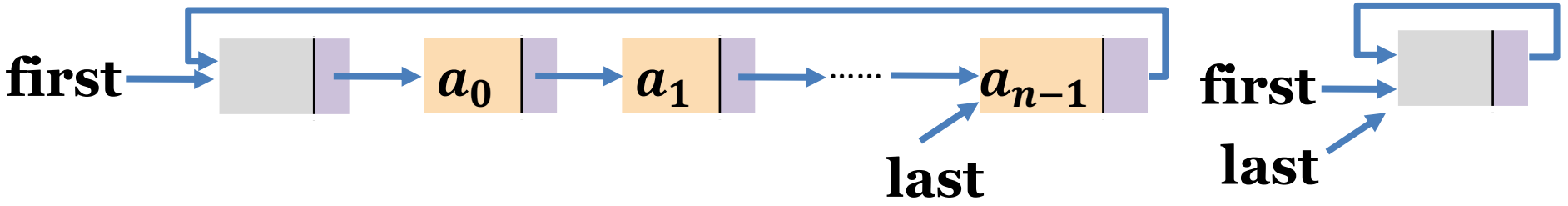
第二章 线性表

南京大學 NANJING UNIVERSITY

循环链表 (Circular List)



循环链表



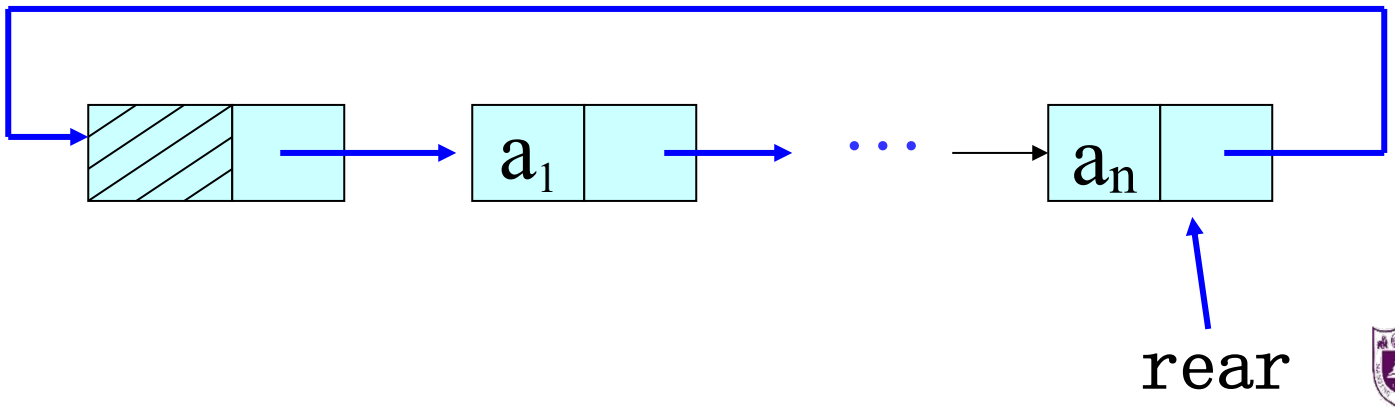
带表头结点的循环链表

南京大學 NANJING UNIVERSITY

循环链表的特点

- 循环链表是一种头尾相接的链表。
- 循环链表最后一个结点的 *link* 指针不为 0 (*NULL*)，而是指向了表头结点。
- 只要知道表中某一结点的地址，就可搜寻到所有其他结点的地址。
- 为了使空表和非空表的处理一致，循环链表中也可设置一个头结点。

- 在很多实际问题中，表的操作常常是在表的尾位置上进行，此时头指针表示的单循环链表就显得不够方便。
- 用尾指针rear来表示单循环链表，则查找开始结点a_1和终端结点a_n分别是rear->next—>next和rear，查找时间都是O(1)。因此，实际中也常采用尾指针表示单循环链表。



南京大学 NANJING UNIVERSITY

循环链表类的定义

```
template <class T>
struct CircLinkNode {                    //链表结点类定义
    T data;
    CircLinkNode<T> *link;
    CircLinkNode ( CircLinkNode<T> *next =
        NULL ) { link = next; }
    CircLinkNode ( E d, CircLinkNode<T> *next =
        NULL ) { data = d;  link = next; }
};
```

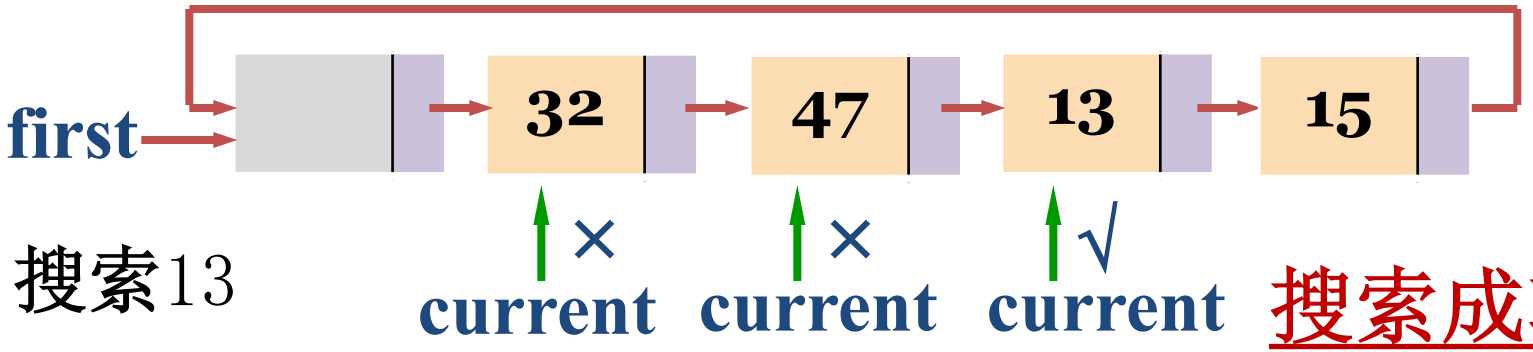
```
template <class T>      //链表类定义
class CircList : public LinearList<T> {
private:
    CircLinkNode<T> *first, *last;  //头指针, 尾指针
public:
    CircList(const T x);                          //构造函数
    CircList(CircList<T>& L);        //复制构造函数
    ～CircList();                                 //析构函数
   int Length() const;                  //计算链表长度
    bool IsEmpty() { return first->link == first; }
                                                  //判表空否
    CircLinkNode<T> *getHead() const;
                                        //返回表头结点地址
```

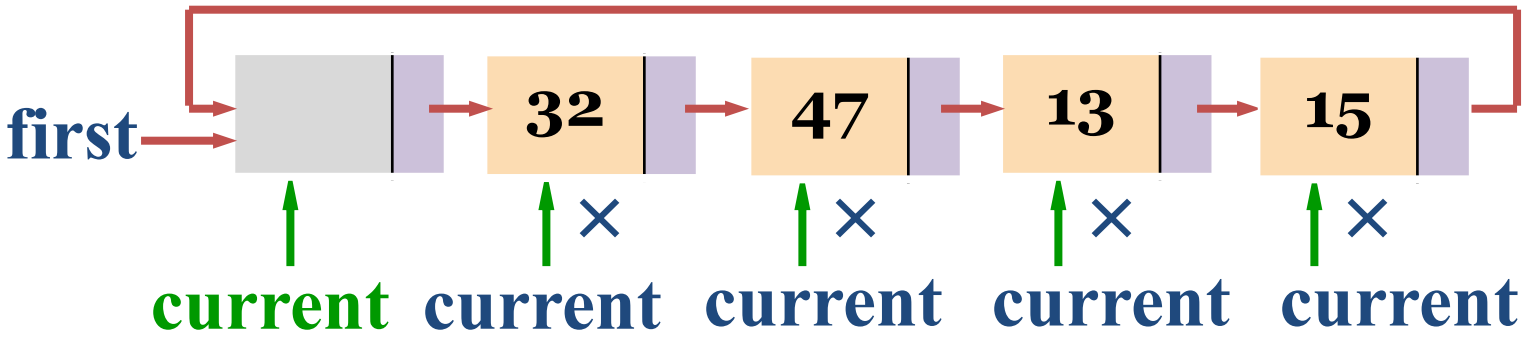
南京大學 NANJING UNIVERSITY

```
    void setHead ( CircLinkNode<T> *p );
                                        //设置表头结点地址
    CircLinkNode<T> *Search ( T x );          //搜索
      CircLinkNode<T> *Locate ( int i );      //定位
    T *getData ( int i );                   //提取
    void setData ( int i, T& x );                   //修改
    bool Insert ( int i, T& x );            //插入
    bool Remove ( int i, T& x);                     //删除
};
```

- 循环链表与单链表的操作实现，最主要的不同就是扫描到链尾，遇到的不是NULL，而是表头。

循环链表的搜索算法





南京大學 NANJING UNIVERSITY

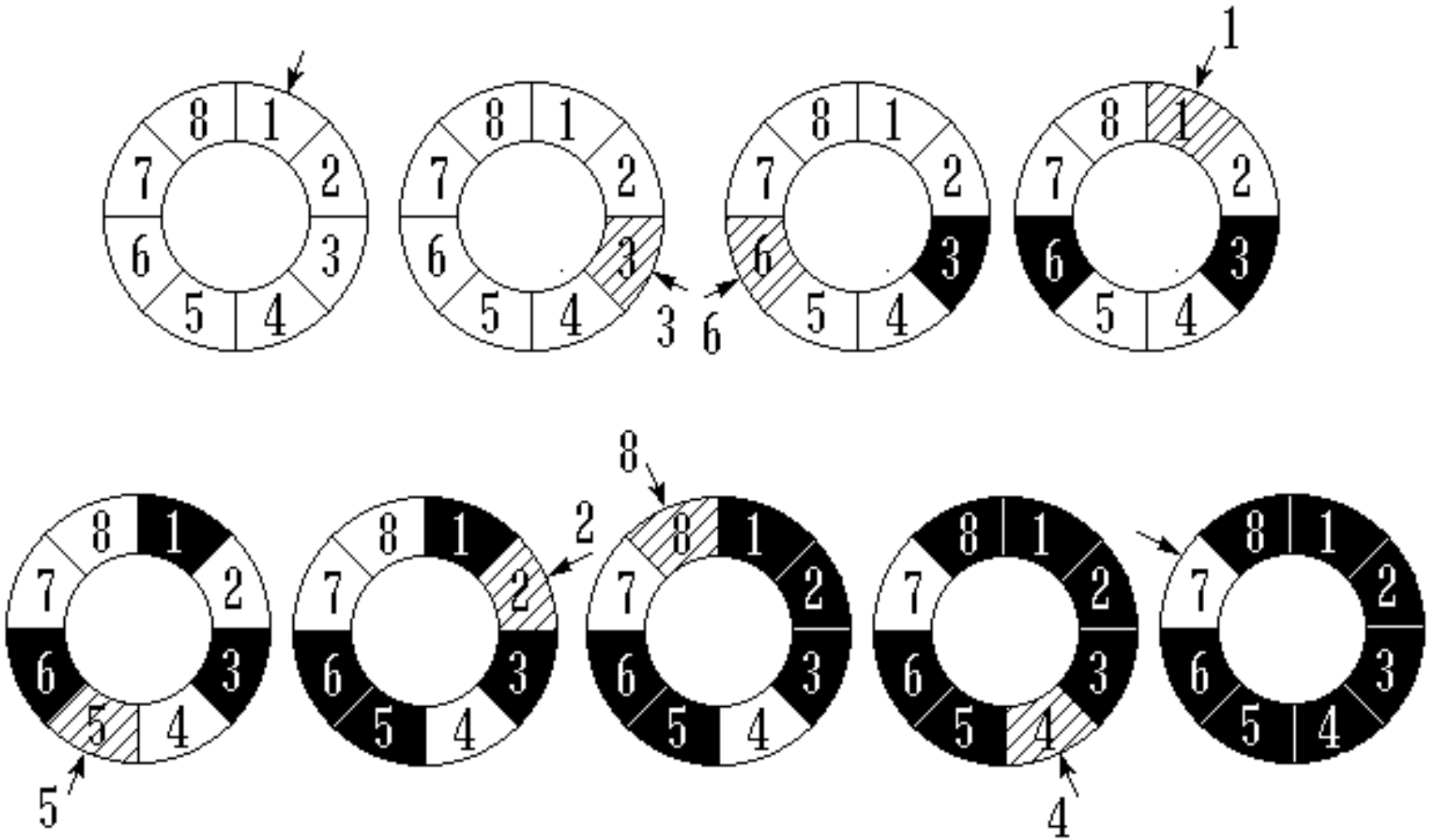
循环链表的搜索算法

```
template <class T>
CircListNode<T> * CircList<T>::Search( T x )
{
//在链表中从头搜索其数据值为 x 的结点
    current = first->link;
    while ( current != first && current->data != x )
        current = current->link;
    return current;
}
```

例子：约瑟夫问题 (n = 8, m = 3)

- https://www.bilibili.com/video/BV1cP4y157ZR?t=31.2

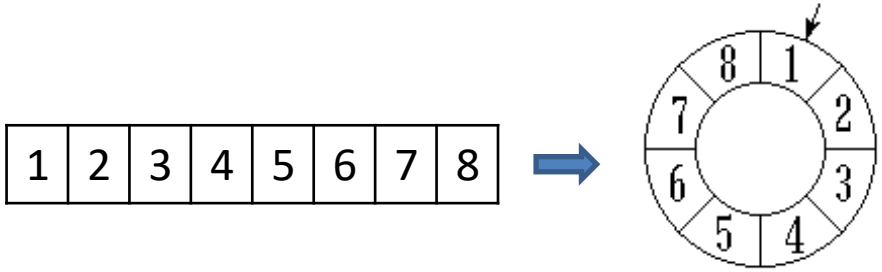
例子：约瑟夫问题 (n = 8, m = 3)

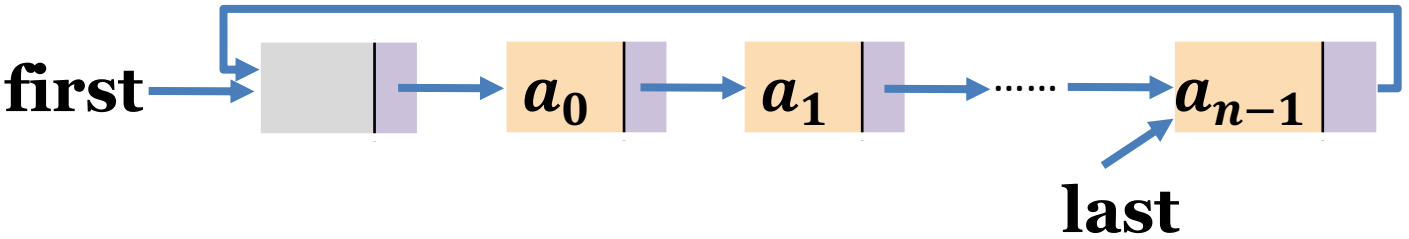


南京大學
NANJING UNIVERSITY

问题的三个要素

- 数据的逻辑结构：环形表结构
- 数据的存储结构：数组实现 / 链表实现

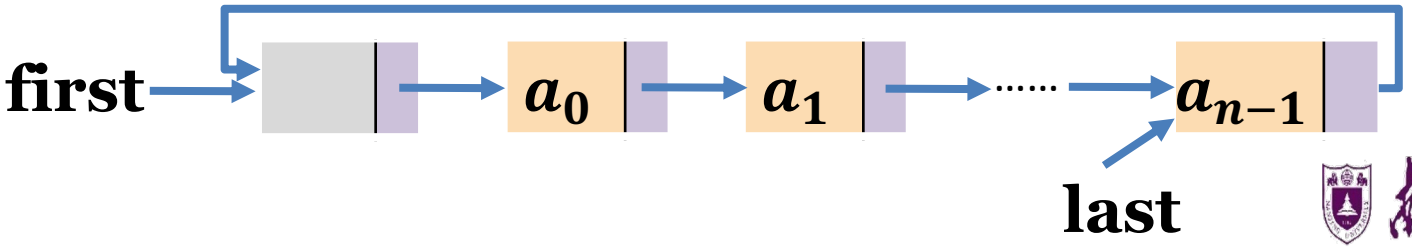




- 数据的运算：在环状结构中循环报数

如何实现循环?

- 循环数组的实现:存放数组被当作首尾相接的表处理。
  - 下标加1时可从maxSize-1直接进到0,可用取模(余数)运算实现。
  - t = (t +1) % maxSize; (t从最后一个下标进入到0)
- 循环链表的实现:将元素存放链表首尾相接,形成循环
  - p = p->link; (最后一个结点执行后p会指向first)



南京大學 NANJING UNIVERSITY

求解Josephus问题的算法（循环链表）

```
#include <iostream.h>
#include "CircList.h"
template <class T, class E>
void Josephus(CircList<T, E>& Js, int n, int m) {
    CircLinkNode<T, E> *p = Js.getHead(),
                       *pre = NULL;
    int i, j;
    for ( i = 0; i < n-1; i++ ) {      //执行n-1次
        for ( j = 1; j < m; j++)                    //数m-1个人
            { pre = p;  p = p->link; }
        cout << "出列的人是"  << p->data << endl;
        pre->link = p->link;
        delete p;                //删去
        p = pre->link;
    }
};
```

南京大學 NANJING UNIVERSITY

求解Josephus问题的算法（循环链表）

```
void main() {
    CircList<int, int> clist;
    int i, n m;
    cout << "输入游戏者人数和报数间隔 : ";
    cin >> n >> m;
    for (i = 1; i <= n; i++ ) clist.insert(i, i);    //约瑟夫环
    Josephus(clist, n, m);                           //解决约瑟夫问题
}
```

双向链表（doubly linked list）

每个结点的结构

- 在单链表的每个结点里再增加一个指向其直接前趋的指针域。这样就形成的链表中有两个方向不同的链，故称为双向链表。

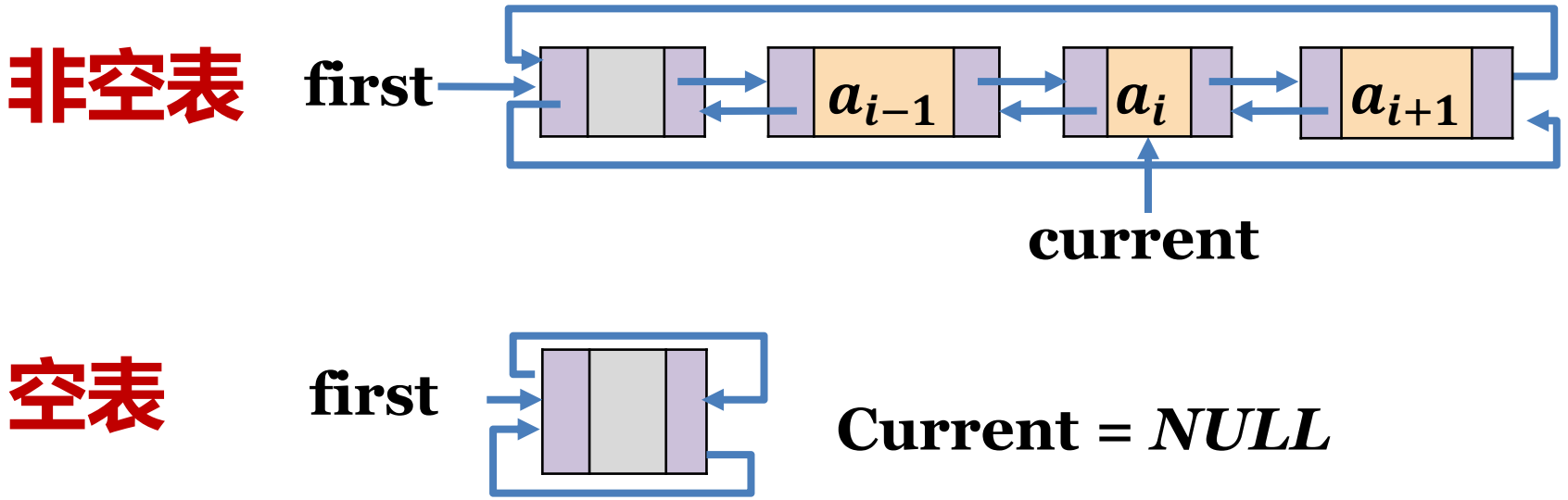
lLink (左链指针)	data (数据)	rLink (右链指针)

前驱方向 ← → 后继方向

南京大學 NANJING UNIVERSITY

双向链表

- 通常采用带头节点的双向循环链表表示
- 前驱结点和后继结点

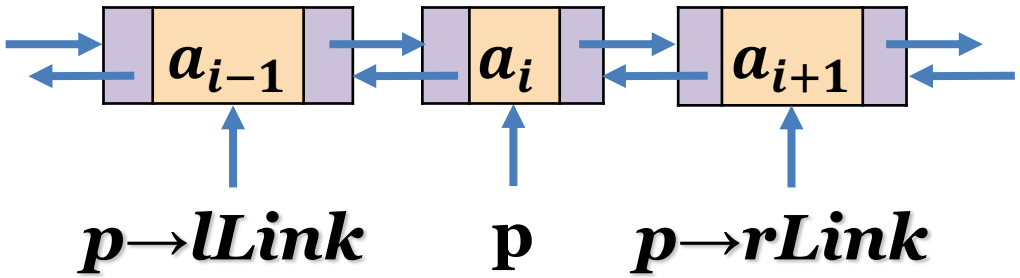


南京大學
NANJING UNIVERSITY

双向链表

- 链表中的结点指针关系
- 设指针p为某一结点，双向链表结构的对称性可描述为：

$$p == p\rightarrow lLink\rightarrow rLink == p\rightarrow rLink\rightarrow lLink$$



南京大學 NANJING UNIVERSITY

双向循环链表类的定义

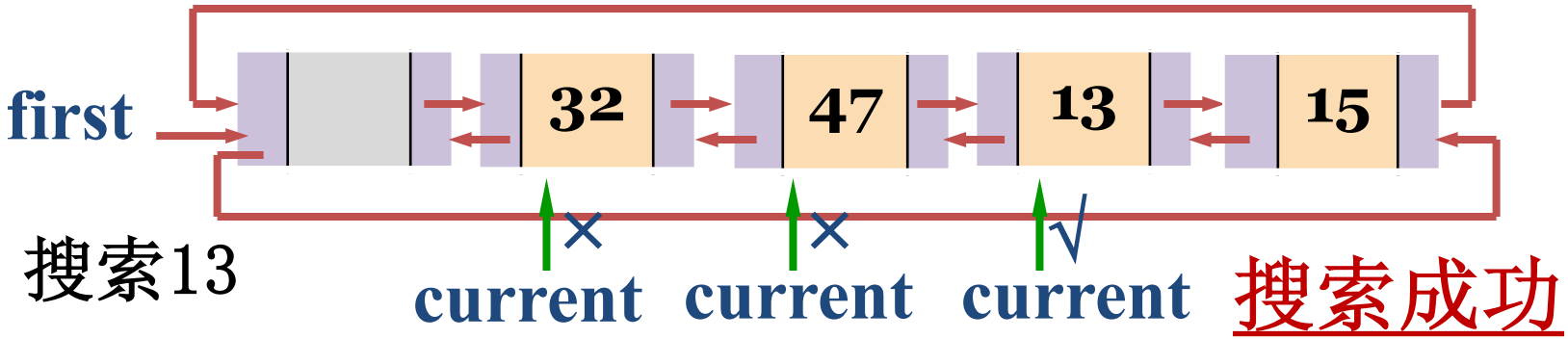
```
template <class T>
struct DblNode {                    //链表结点类定义
T data;                             //链表结点数据
DblNode<T> *lLink, *rLink;          //前驱、后继指针
DblNode ( DblNode<T> *l = NULL,
    DblNode<T> *r = NULL )
      { lLink = l;  rLink = r; }          //构造函数
 DblNode (T value, DblNode<T> *l = NULL,
    DblNode<T> *r = NULL)
      { data = value;  lLink = l;  rLink = r; }  //构造函数
};
```

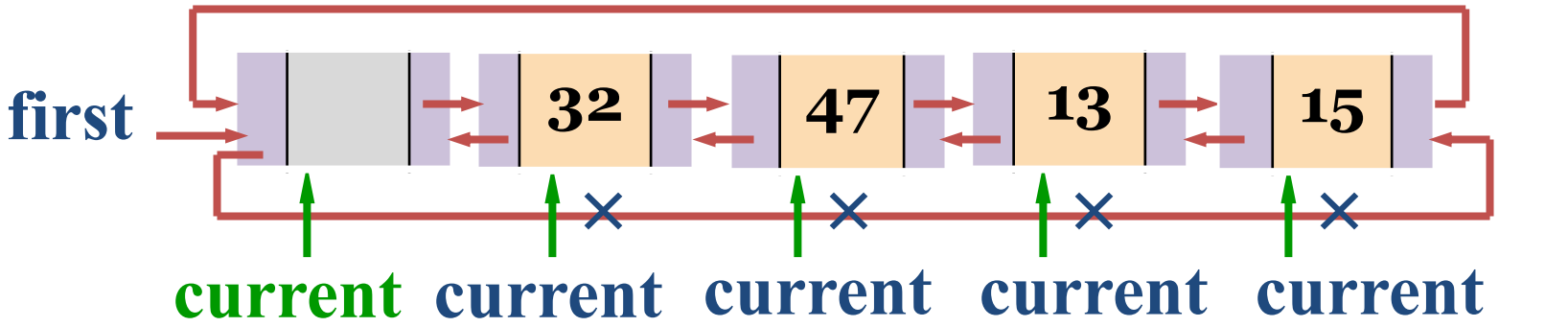
数据成员

```
template <class T>
class DblList {                    //链表类定义
public:
DblList ( T uniqueVal ) {          //构造函数
     first = new DblNode<T> (uniqueVal);
     first->rLink = first->lLink = first;
};
DblNode<T> *getFirst () const { return first; }
void setFirst ( DblNode<T> *ptr ) { first = ptr; }
DblNode<T> *Search ( T x, int d);
   //在链表中按d指示方向寻找等于给定值x的结点,
   //d=0按前驱方向,d≠0按后继方向
```

```
DblNode<T> *Locate ( int i, int d );
  //在链表中定位序号为i(≥0)的结点, d=0按前驱方
  //向,d≠0按后继方向
bool Insert ( int i, T x, int d );
  //在第i个结点后插入一个包含有值x的新结点,d=0
  //按前驱方向,d≠0按后继方向
bool Remove ( int i, T& x, int d );   //删除第i个结点
bool IsEmpty() { return first->rlink == first; }
  //判双链表空否
private:
    DblNode<T> *first;                //表头指针
};
```

双向循环链表的搜索算法





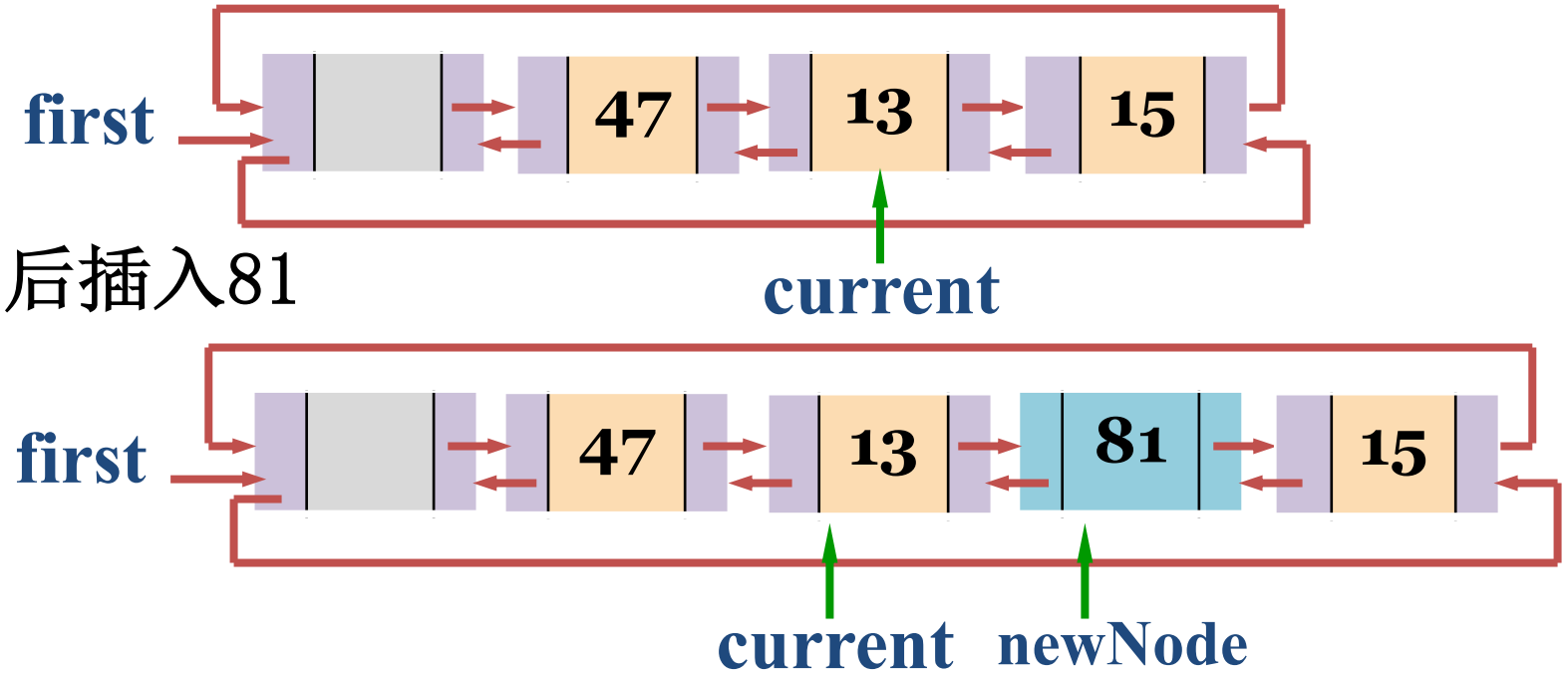
南京大學
NANJING UNIVERSITY

双向循环链表的搜索算法

```
template <class T>
DblNode<T> *DblList<T>::Search (T x, int d) {
//在双向循环链表中寻找其值等于x的结点。
   DblNode<T> *current = (d == 0)?
        first->lLink : first->rLink;   //按d确定搜索方向
   while ( current != first && current->data != x )
      current = (d == 0) ? current->lLink : current-
   >rLink;
   if ( current != first ) return current;    //搜索成功
   else return NULL;                          //搜索失败
};
```

南京大學 NANJING UNIVERSITY

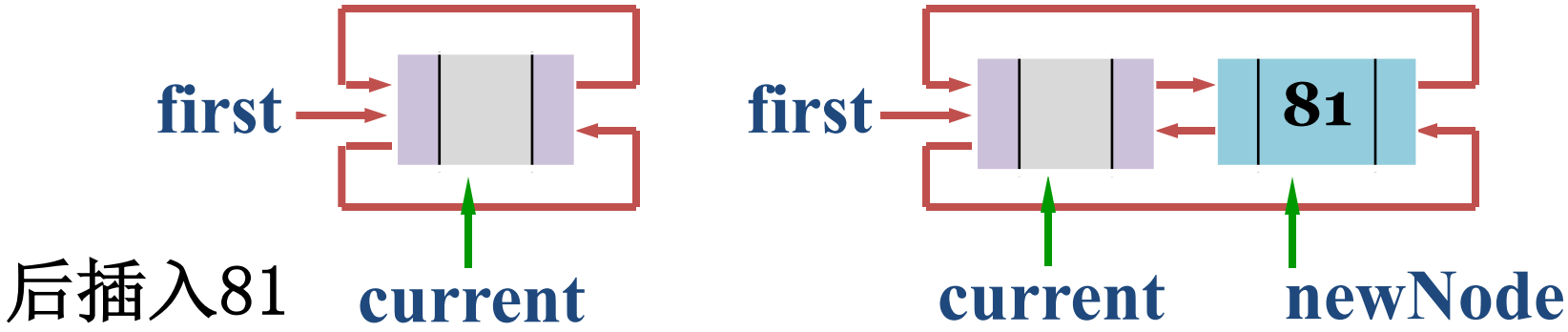
双向循环链表的插入算法 (非空表)




```
newNode->rLink = current->rLink;
current->rLink = newNode;
newNode->rLink->lLink = newNode;
newNode->lLink = current;
```

南京大學 NANJING UNIVERSITY

双向循环链表的插入算法 (空表)




```
newNode->rLink = current->rLink
        (newNode->rLink = first);
current->rLink = newNode;
newNode->rLink ->lLink = newNode;
                ( first->lLink = newNode )
newNode->lLink = current;
```

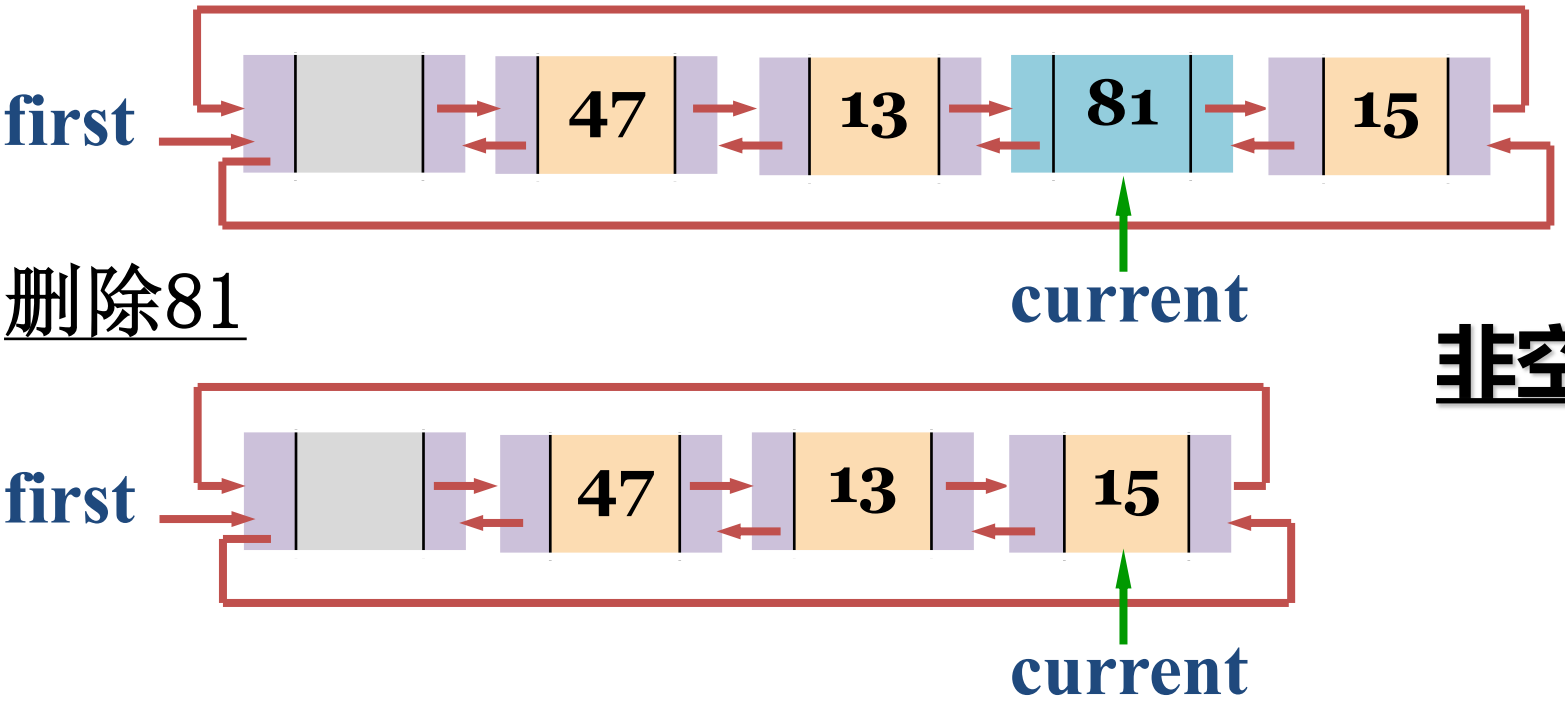
双向循环链表的插入算法

```
template <class T>
bool DblList<T>::Insert ( int i, T x, int d ) {
//建立一个包含有值x的新结点, 并将其按 d 指定的
//方向插入到第i个结点之后。
    DblNode<T> *current = Locate(i, d);
      //按d指示方向查找第i个结点
    if ( current == NULL ) return false; //链太短，插
    入失败
    DblNode<T> *newNd = new DblNode<T>(x);
    if (d == 0) {        //前驱方向:插在第i个结点左侧
        newNd->lLink = current->lLink; //链入lLink链
        current->lLink = newNd;
```

南京大學 NANJING UNIVERSITY

```
        newNd->lLink->rLink = newNd;  //链入rLink链
        newNd->rLink = current;
    } else {                    //后继方向:插在第i个结点后面
        newNd->rLink = current->rLink;  //链入rLink链
        current->rLink = newNd;
        newNd->rLink->lLink = newNd;   //链入lLink链
        newNd->lLink = current;
    }
    return true;            //插入成功
};
```

双向循环链表的删除算法




```
current->rLink->lLink = current->lLink;
current->lLink->rLink = current->rLink;
```

双向循环链表的删除算法

```
template <class T>
bool DblList<T>::Remove( int i, T& x, int d ) {
//在双向循环链表中按d所指方向删除第i个结点。
    DblNode<T> *current = Locate (i, d);
    if (current == NULL) return false;        //删除失败
    current->rLink->lLink = current->lLink;
    current->lLink->rLink = current->rLink;
        //从lLink链和rLink链中摘下
    x = current->data;  delete current;       //删除
    return true;                              //删除成功
};
```

南京大學 NANJING UNIVERSITY

第二章 线性表

2.1 线性表的概念

2.2 顺序表

2.3 单链表

2.4 单链表的变形：循环链表和双向链表

2.5 单链表的应用：多项式

多项式及其运算

一元多项式例：

$$A(x) = 1 - 10x^6 + 2x^8 + 7x^{14}$$

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

$$= \sum_{i=0}^{n} a_i x^i$$

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

多项式的顺序存储表示

第一种：静态数组表述

```
private:
    int degree;
    float coef [maxDegree+1];
```

$P_n(x)$可以表示为对象pl：

pl.degree = n

pl.coef[*i*] = a_i, $0 \le i \le n$

南京大學 NANJING UNIVERSITY

第二种：动态数组表示

```
private:
        int degree;
        float * coef;
Polynomial :: Polynomial (int sz) {
        degree = sz;
        coef = new float [degree + 1];
}
```

以上两种存储表示适用于指数连续排列的多项式。但对于绝大多数项的系数为零的多项式（稀疏多项式），如 $P_{101}(x) = 3+5x^{50}-4x^{101}$, 不经济。

南京大學 NANJING UNIVERSITY

- 以上两种存储表示适用于指数连续排列的多项式。
- 但对于绝大多数项的系数为零的多项式（稀疏多项式）
- 例： $P_{101}(x) = 3+5x^{50}-4x^{101}$, 大量位置为0，如何节省存储空间？

第三种 同时存储系数和指数：

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

```
struct term {          //多项式的项定义
        float coef;          //系数
        int exp;             //指数
};

static term termArray[maxTerms];     //项数组
static int free, maxTerms;           //当前空闲位置指针
```

南京大學 NANJING UNIVERSITY

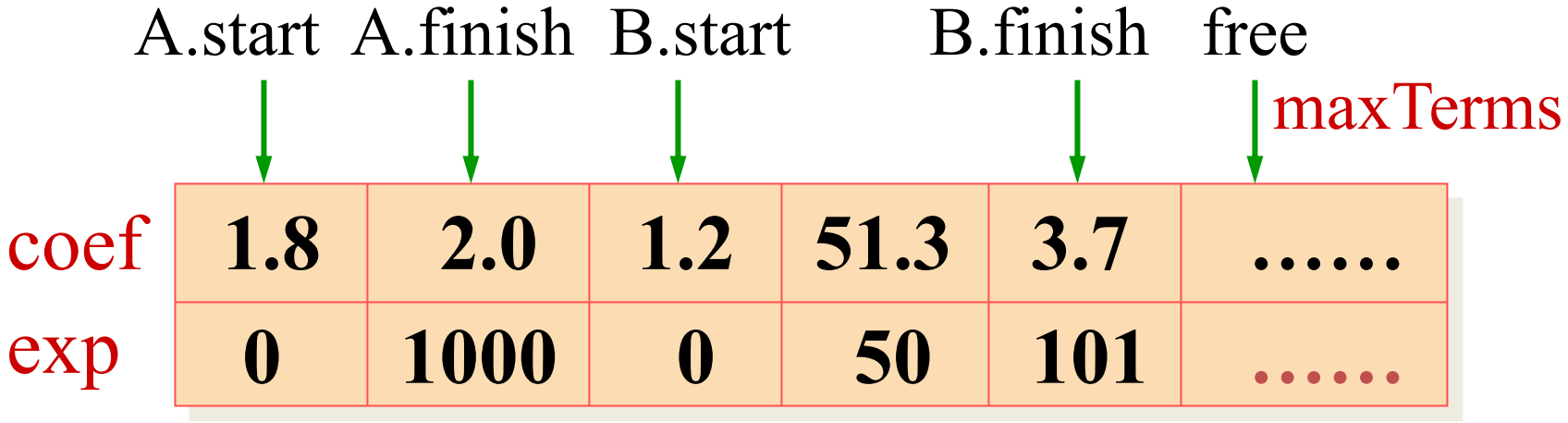
初始化：

```
// term Polynomial::termArray[MaxTerms];
// int Polynomial::free = 0;

class Polynomial {          //多项式定义
public:
   ……
private:
   int start, finish;       //数据成员、多项式始末位置
}
```

两个多项式存储的例子

$A(x) = 2.0x^{1000}+1.8$

$B(x) = 1.2 + 51.3x^{50} + 3.7x^{101}$



两个多项式存放在termArray中

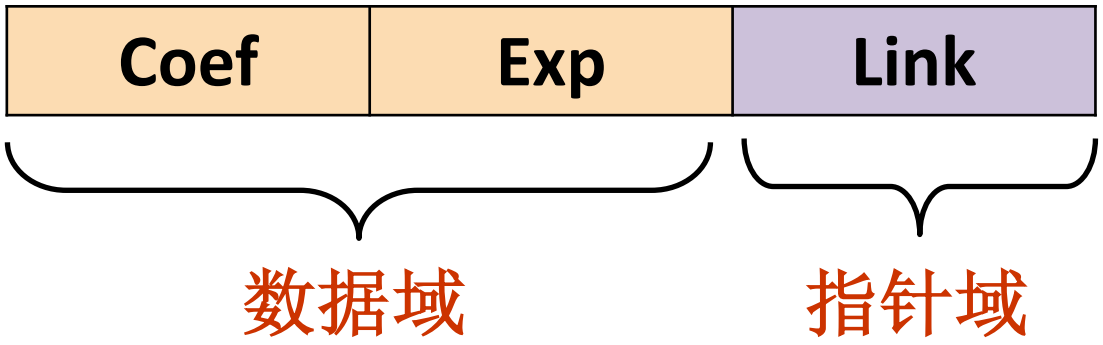
多项式的顺序存储

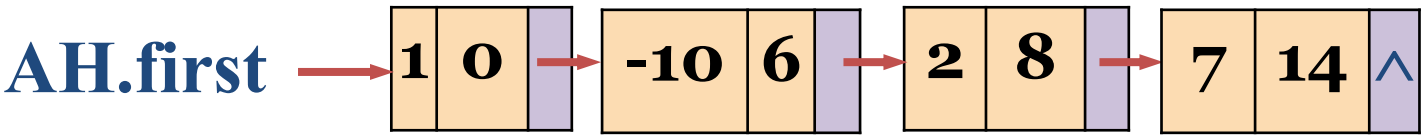
- 多项式顺序存储表示的缺点
  - 插入和删除时项数可能有较大变化，因此要移动大量数据
  - 不利于多个多项式的同时处理、有溢出问题

第四种：多项式的链表存储表示

$$A(x) = 1 - 10x^6 + 2x^8 + 7x^{14}$$

每个结点的结构：





南京大学 NANJING UNIVERSITY

多项式的顺序存储和链表存储

- 多项式顺序存储表示的缺点
  - 插入和删除时项数可能有较大变化，因此要移动大量数据
  - 不利于多个多项式的同时处理、有溢出问题
- 多项式链表存储表示的优点
  - 插入、删除方便，不移动元素
  - 多项式的项数可以动态地增长，不存在存储溢出问题。

多项式(polynomial)类的链表定义

```
struct Term {                          //多项式结点定义
    float coef;                        //系数
    int exp;                           //指数
    Term *link;                        //链接指针
    Term (float c, int e, Term *next = NULL)
        { coef = c;  exp = e;  link = next;}
    Term *InsertAfter ( float c, int e);
    friend ostream& operator << (ostream&,
                const Term& );
};
```

```
class Polynomial {                          //多项式类的定义
public:
    Polynomal() { first = new Term(0, -1); }  //构造函数
    Polynomal (Polynomal& R);               //复制构造函数
    int maxOrder();                         //计算最大阶数
private:
    Term *first;                            //多项式链表的头指针
    friend ostream& operator << (ostream&,
          const Polynomal& );
    friend istream& operator >> ( istream&,
         Polynomal& );
```

```
    friend void Add ( Polynomial& A, Polynomial& B,
                Polynomial& C );
    friend void Mul ( Polynomial& A, Polynomial& B,
                Polynomial& C );
};
```

多项式链表的相加

例：求两多项式的和多项式

$A(x) = 1 - 3x^6 + 7x^{12}$

$B(x) = -x^4 + 3x^6 - 9x^{10} + 8x^{14}$

一元多项式相加运算规则：指数相同的项系数相加

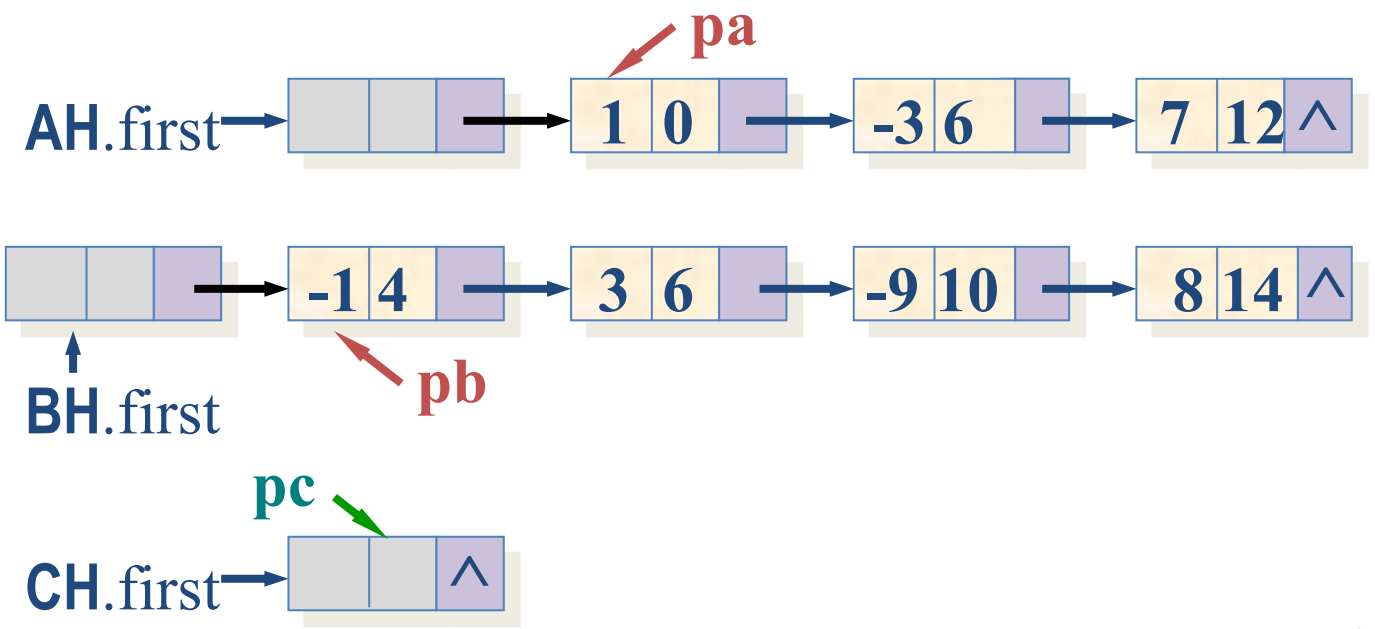
A (x) B (x)相加的和多项式为

$C(x) = 1 - x^4 - 9x^{10} + 7x^{12} + 8x^{14}$

南京大學 NANJING UNIVERSITY

两个多项式的相加算法

- 设两个多项式都带表头结点，检测指针pa和pb分别指示两个链表当前检测结点。
- 设结果多项式的表头指针为C，存放指针为pc，初始位置在C的表头结点。



南京大學 NANJING UNIVERSITY

两个多项式的相加算法步骤

1） 当pa和pb没有检测完各自的链表时，循环执行以下指令，否则执行步骤2）。

比较 pa 和 pb 当前检测结点的指数域。若：

- 指数不等：小者加入C链，相应检测指针pa或者pb进1；
- 指数相等：对应项系数相加。若相加结果不为零，则结果加入C链，pa与pb进1。

2） 当pa或pb指针中有一个为NULL，则把另一个链表指针不为NULL的剩余部分加入到C链。

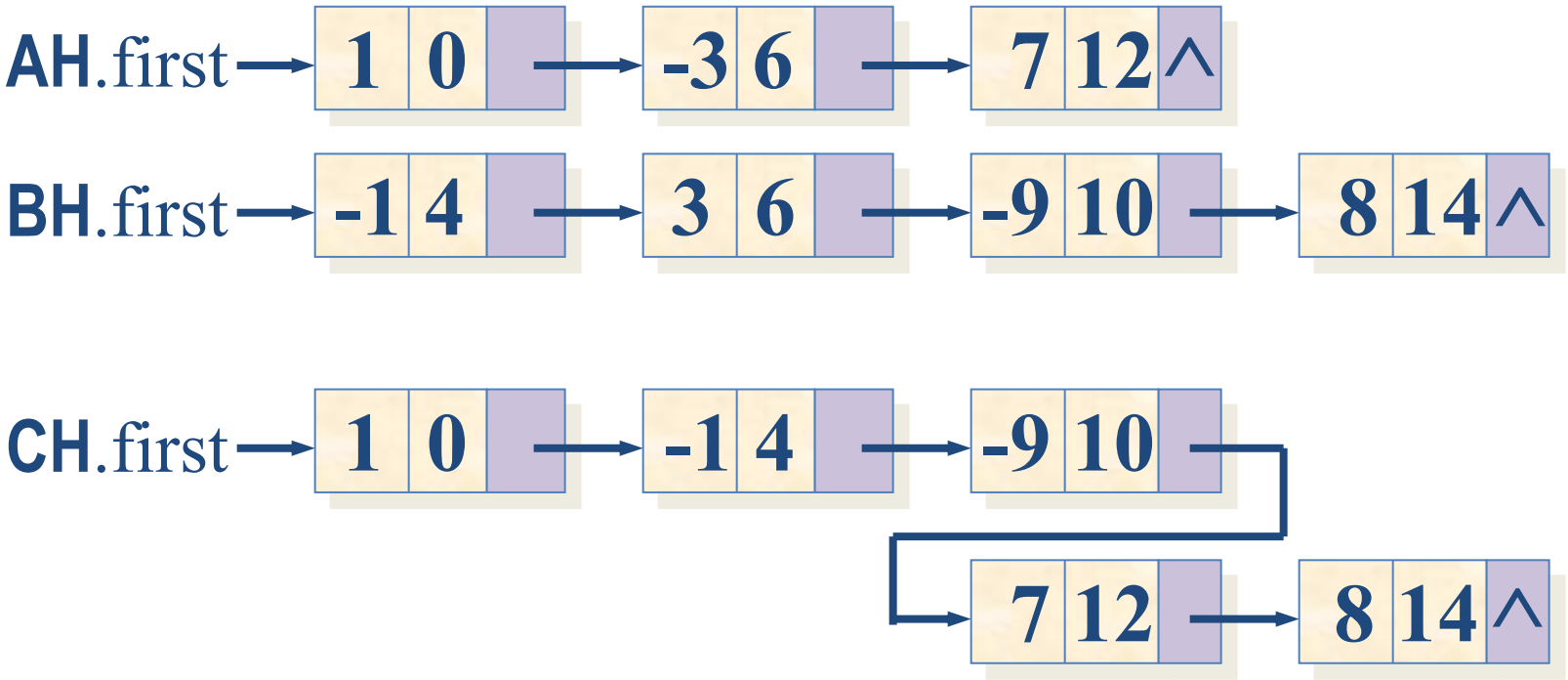
南京大学 NANJING UNIVERSITY

多项式链表的相加

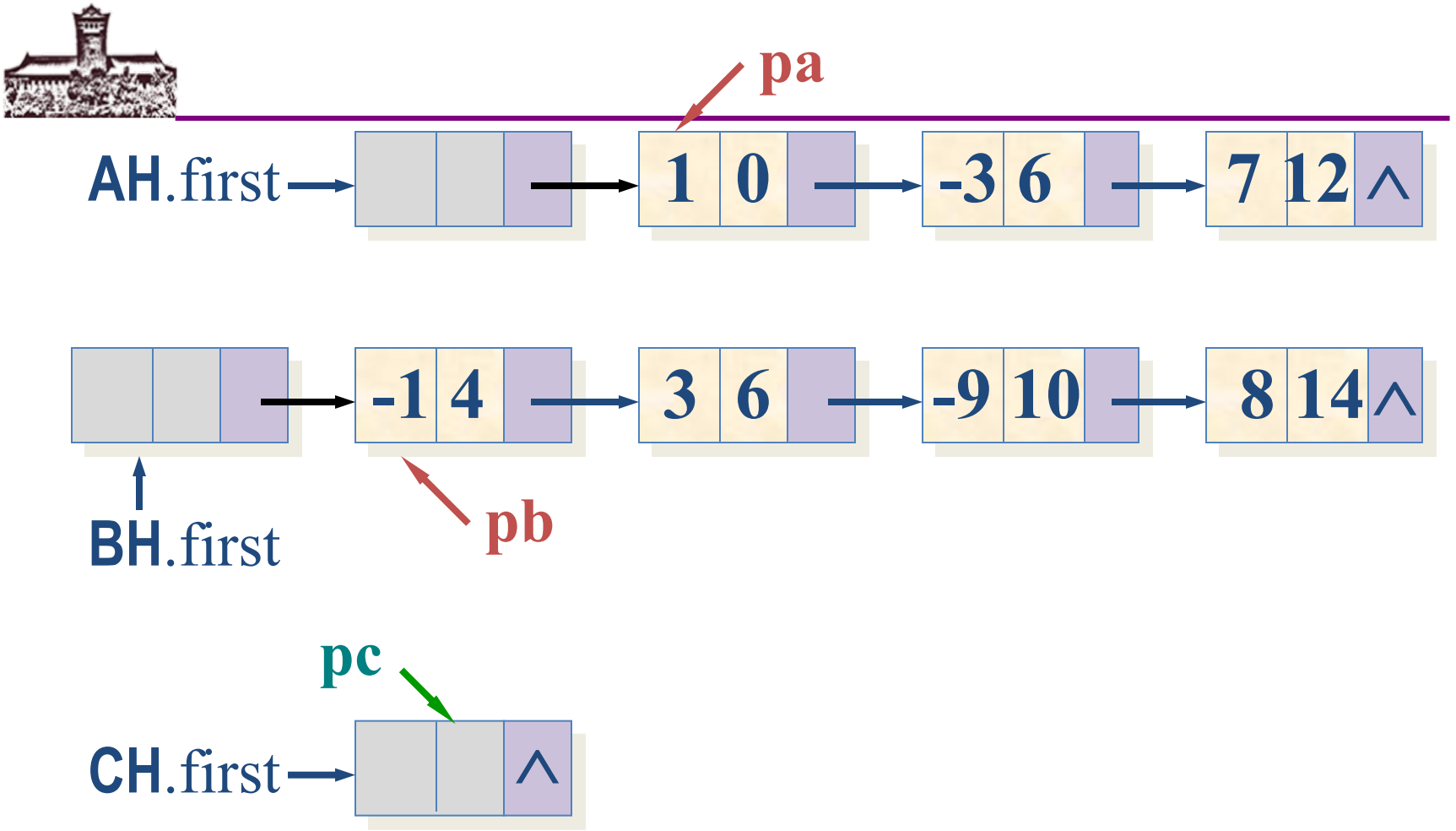
$AH = 1 - 3x^6 + 7x^{12}$

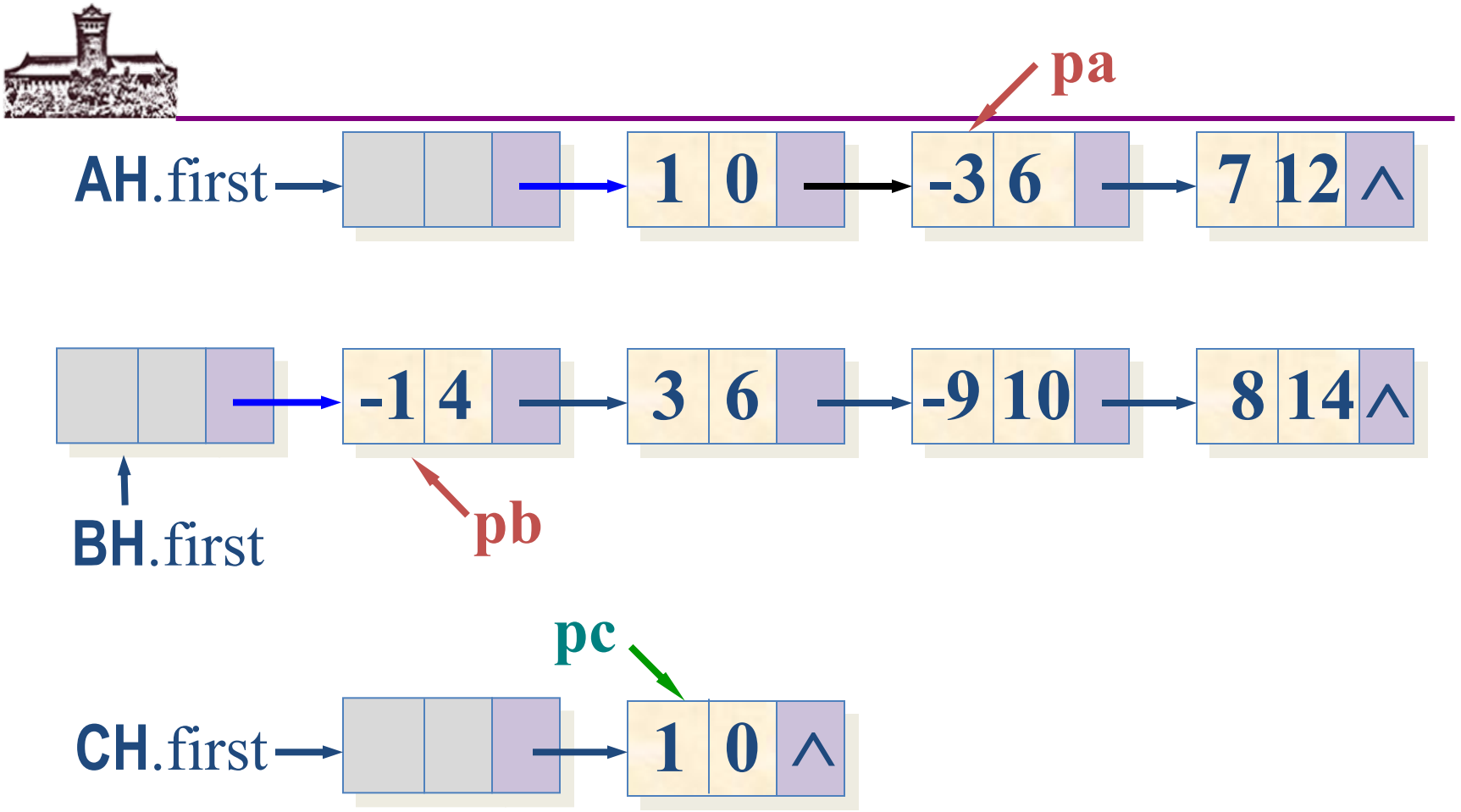
$BH = -x^4 + 3x^6 - 9x^{10} + 8x^{14}$

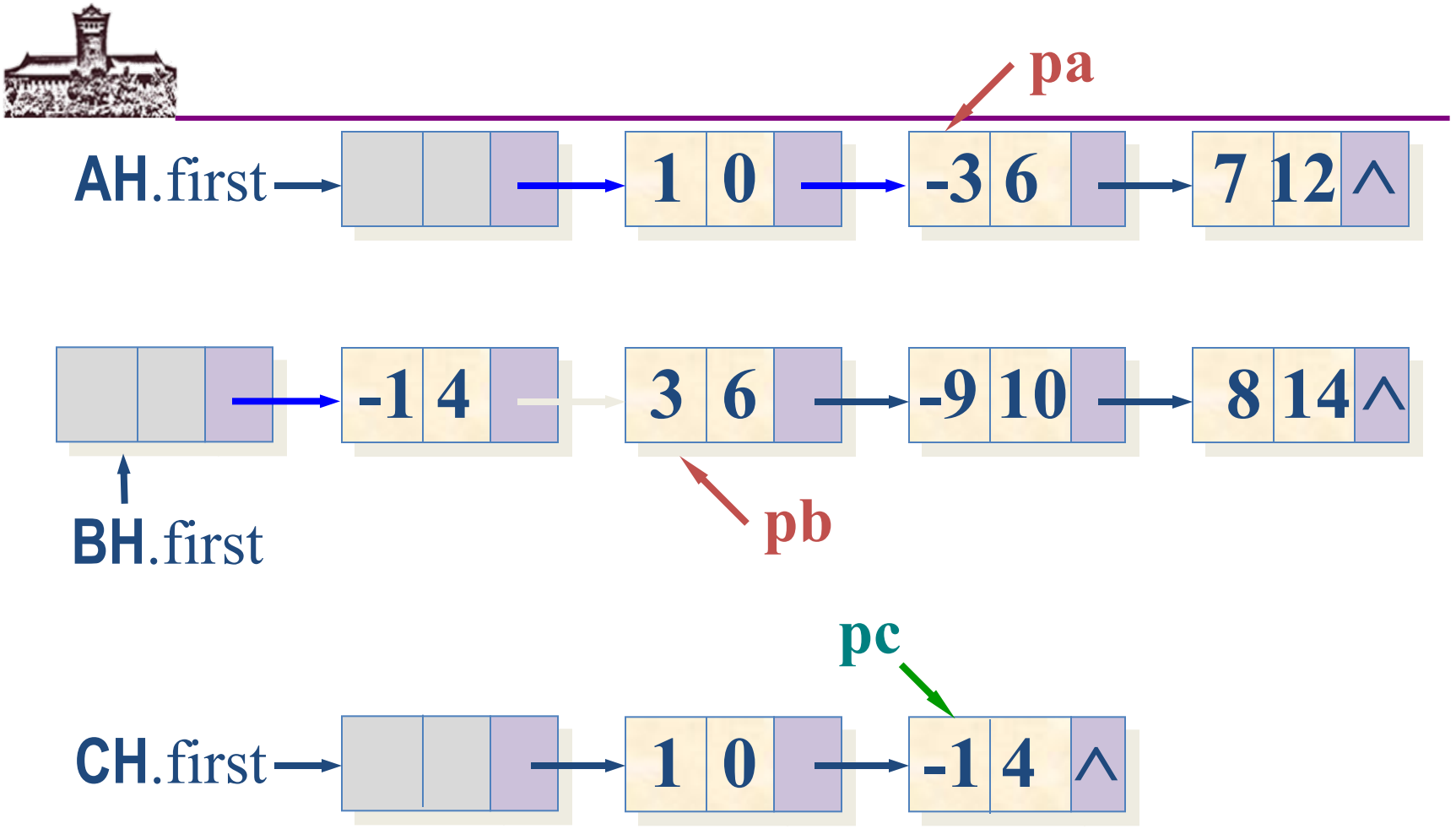
$CH = 1 - x^4 - 9x^{10} + 7x^{12} + 8x^{14}$

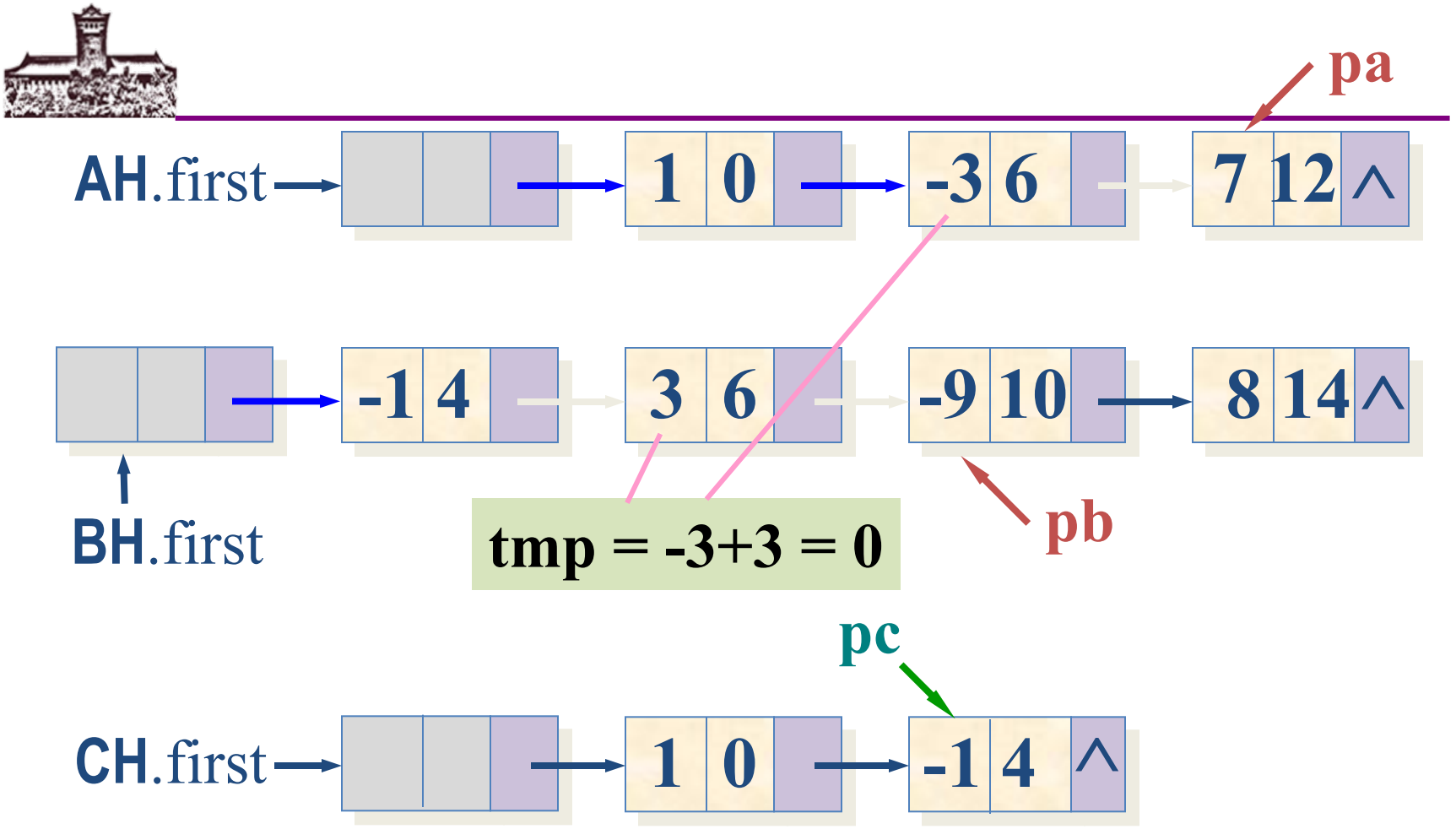


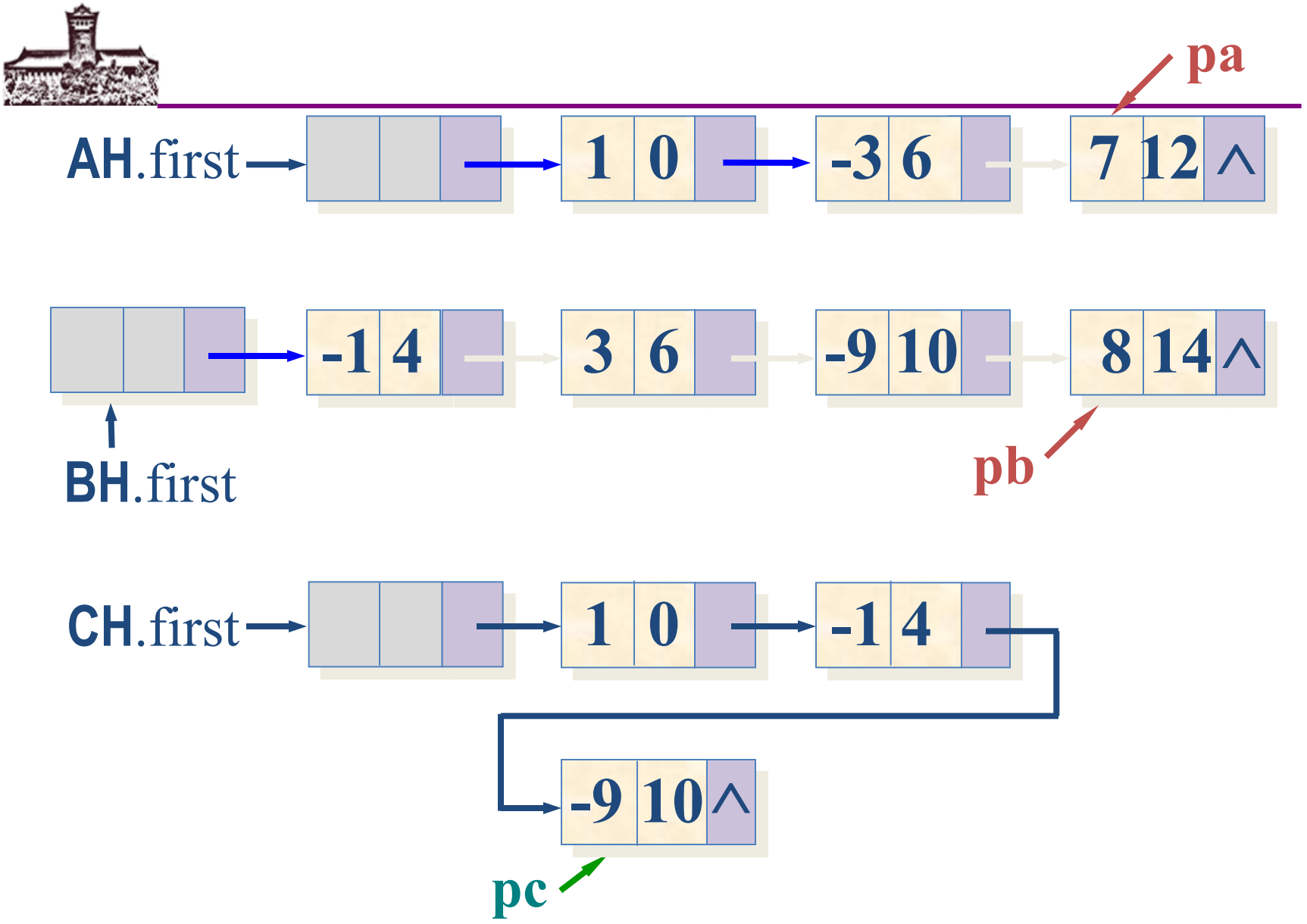
南京大學 NANJING UNIVERSITY

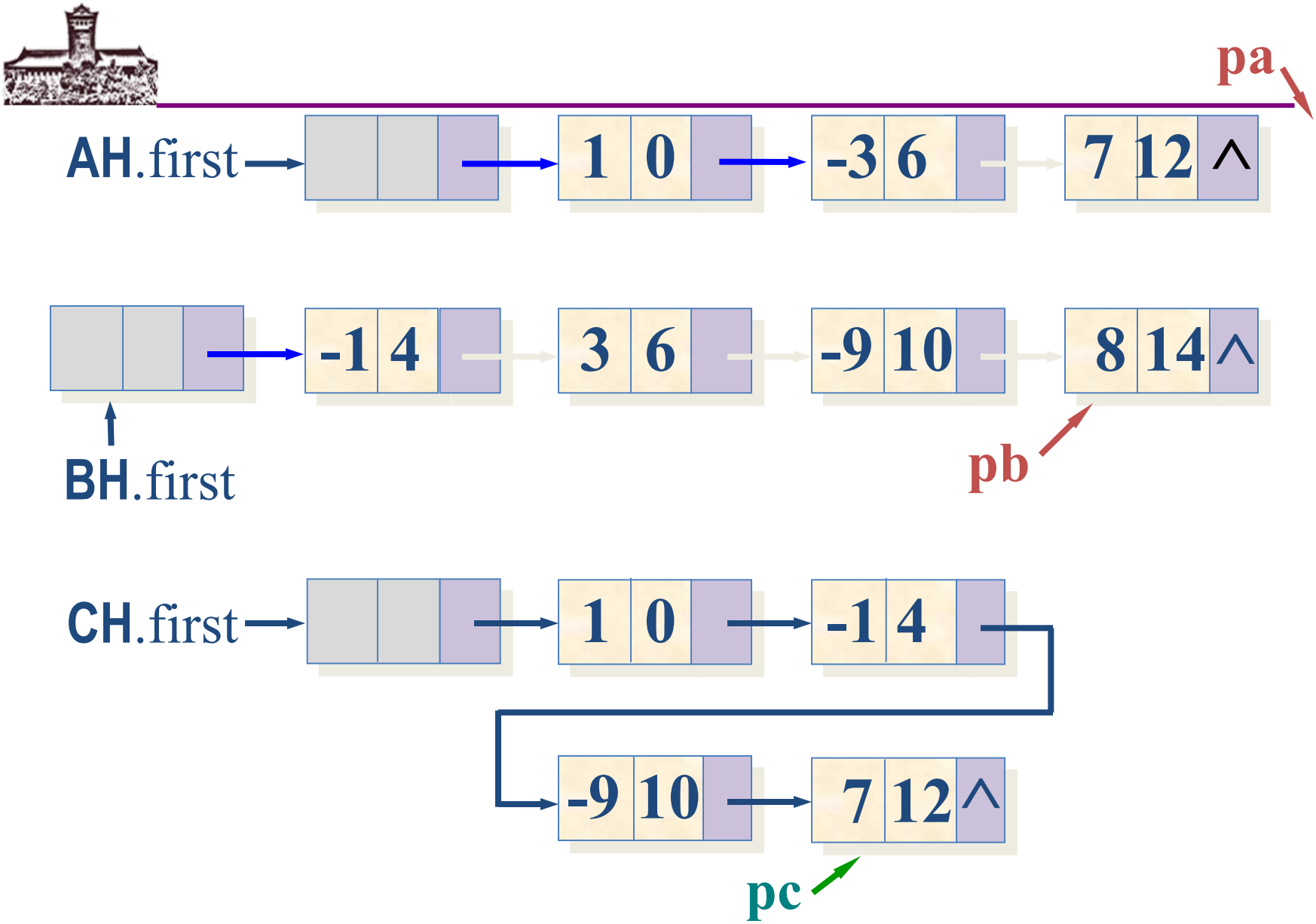

pa
AH.first
1 0
-3 6
7 12 ∧
-1 4
3 6
-9 10
8 14 ∧
pb
BH.first
pc
CH.first
∧


pa
AH.first
1 0
-3 6
7 12 ∧
-1 4
3 6
-9 10
8 14 ∧
BH.first
pb
pc
CH.first
1 0 ∧

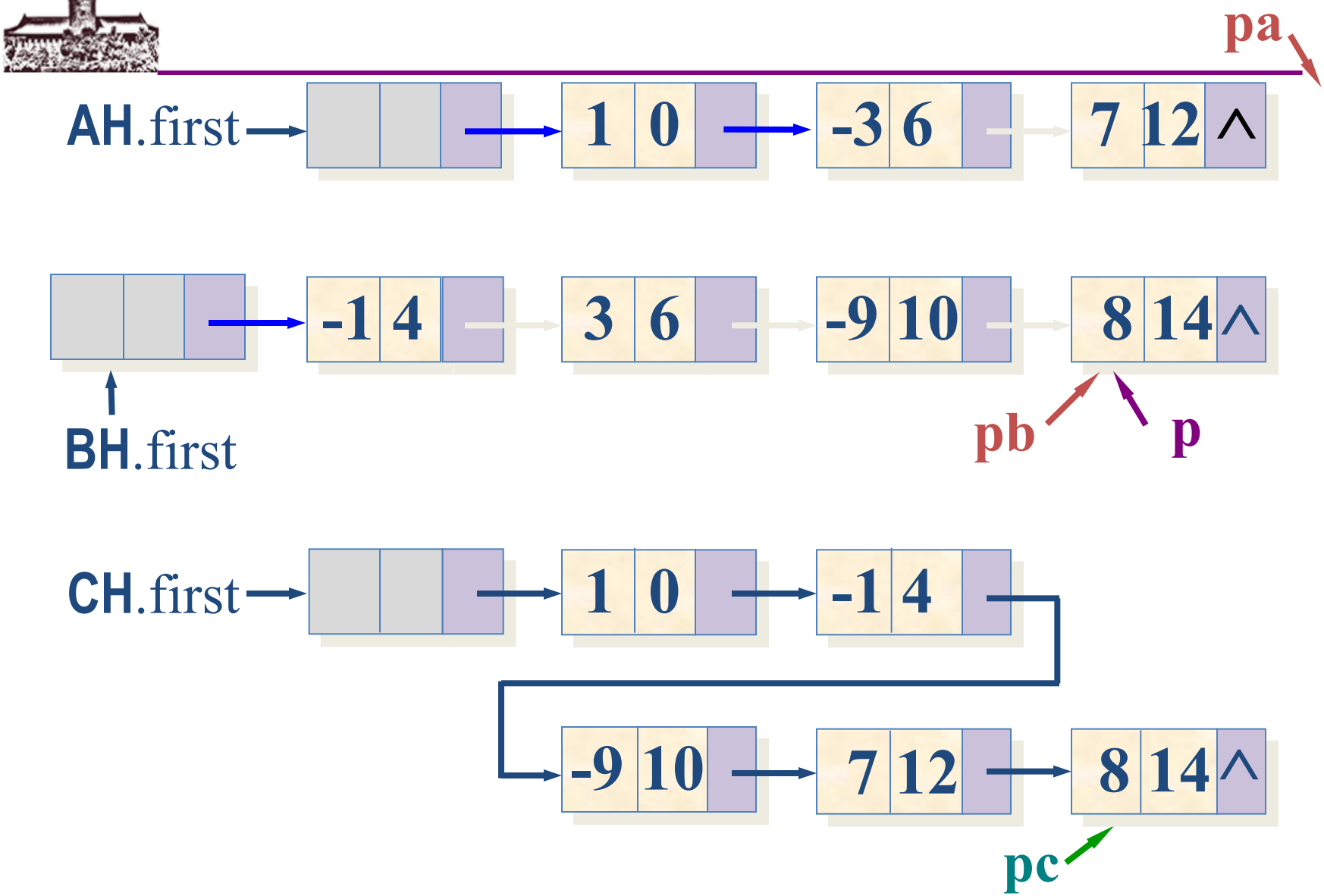

pa
AH.first
1 0
-3 6
7 12 ^
-1 4
3 6
-9 10
8 14 ^
BH.first
pb
pc
CH.first
1 0
-1 4 ^


pa
AH.first
1 0
-3 6
7 12 ^
-1 4
3 6
-9 10
8 14 ^
BH.first
tmp = -3+3 = 0
pb
pc
CH.first
1 0
-1 4 ^


pa
AH.first
1 0
-3 6
7 12 ^
-1 4
3 6
-9 10
8 14 ^
BH.first
pb
CH.first
1 0
-1 4
-9 10 ^
pc



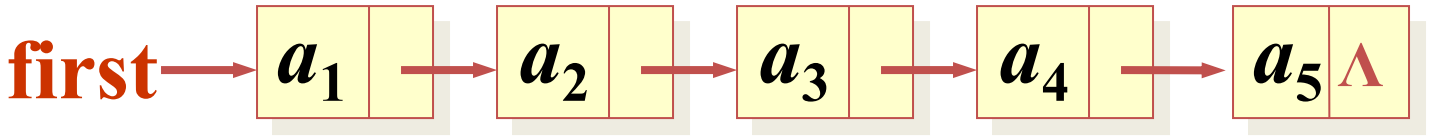
南京大學
NANJING UNIVERSITY



南京大學
NANJING UNIVERSITY

静态链表

- 为数组中每一个元素附加一个链接指针，就形成静态链表结构。
- 静态链表每个结点由两个数据成员构成：data域存储数据，link域存放链接指针。



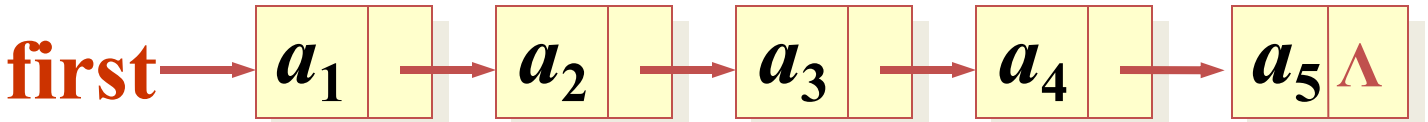
	0	1	2	3	4	5
data		a_3	a_4	a_1	a_5	a_2
link	3	2	4	5	-1(0)	1

南京大學 NANJING UNIVERSITY

静态链表的特点

- 处理时中可以不改变各元素的物理位置，只要重新链接就能改变这些元素的逻辑顺序。
- 它是利用数组定义的，在整个运算过程中存储空间的大小不会变化。

静态链表的结构



	0	1	2	3	4	5
data		a_3	a_4	a_1	a_5	a_2
link	3	2	4	5	-1(0)	1

- 0号是表头结点，link给出首元结点地址。
- 循环链表收尾时link = 0，回到表头结点。如果不是循环链表，收尾结点指针link = -1。
- link指针是数组下标，因此是整数。

南京大学 NANJING UNIVERSITY

静态链表和动态链表的区别

静态链表和动态链表是线性表链式存储结构的两种不同的表示方式。

- 静态链表是用类似于数组方法实现的，是顺序的存储结构，在物理地址上是连续的，而且需要预先分配地址空间大小。所以静态链表的初始长度一般是固定的，在做插入和删除操作时不需要移动元素，仅需修改指针。
- 动态链表是用内存申请函数（new）动态申请内存的，所以在链表的长度上没有限制。动态链表因为是动态申请内存的，所以每个节点的物理地址不连续，要通过指针来顺序访问。

静态链表其实是为了给没有指针的高级语言设计的一种实现单链表能力的方法。

大作业