数据结构与算法（A）-W02/线性表

北京大学 陈斌

2024.09.13

第二章 线性表

- 线性结构和顺序表
- 链表
- 线性表实现方法的比较

线性结构

- 二元组$B=(K,R)$ $K=\{a_0,a_1,\dots,a_{n-1}\}$ $R=\{\langle a_i,a_{i+1}\rangle \mid 0\le i<n-1\}$

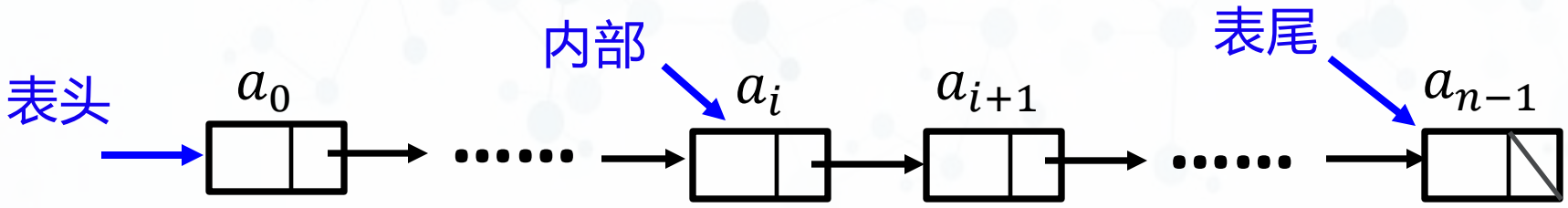
有一个唯一的 **开始结点**，它没有前驱，有一个唯一的后继

一个唯一的 **终止结点**，它有一个唯一的前驱而没有后继

其它的结点皆称为**内部结点**，每一个内部结点都有且仅有一个唯一的前驱，也有一个唯一的后继

$\langle a_i,a_{i+1}\rangle$ a_i是a_{i+1}的前驱，a_{i+1}是a_i的后继

前驱/后继关系r，具有 **反对称性** 和 **反自反性**

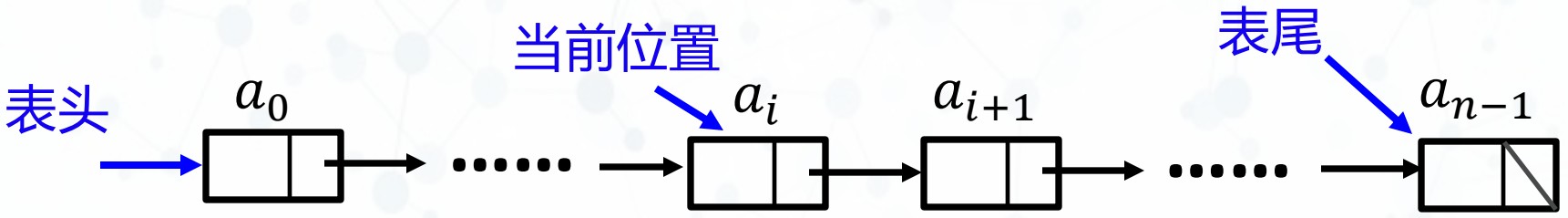


2.1 线性表

- 三个方面
  - 线性表的逻辑结构
  - 线性表的存储结构
  - 线性表运算

线性表逻辑结构

- 主要属性包括：
  - 线性表的长度
  - 表头 (head)
  - 表尾 (tail)
  - 当前位置 (current position)



线性表的存储结构

- 顺序表

  按索引值从小到大存放在一片相邻的连续区域

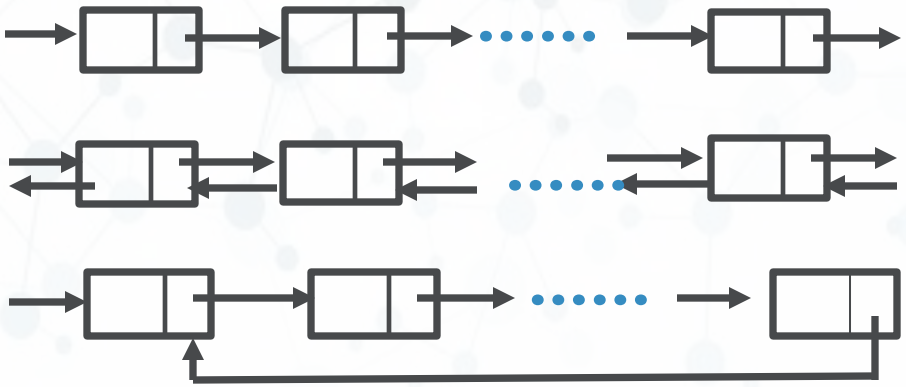
  紧凑结构，存储密度为1

- 链表

  单链表

  双链表

  循环链表



线性表分类（按操作）

- **线性表**
  不限制操作
- **栈**
  在同一端操作
- **队列**
  在两端操作

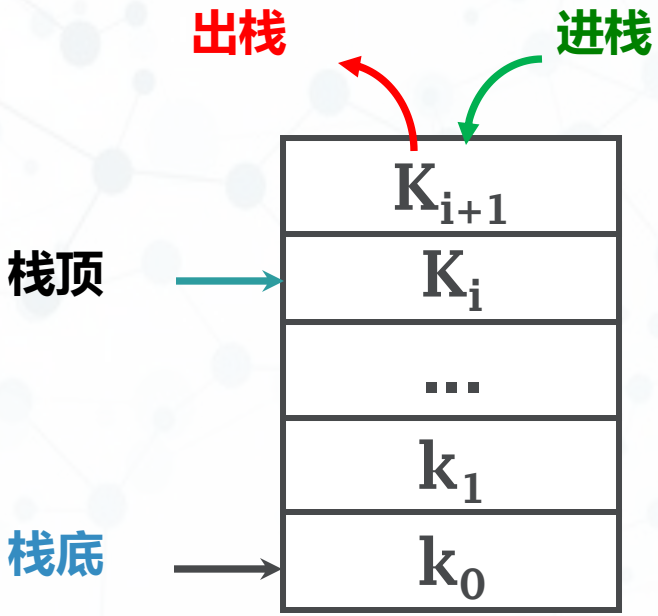
线性表分类（按操作）

- 线性表
  - 所有表目都是同一类型结点的线性表
  - 不限制操作形式
  - 根据存储的不同分为： 顺序表，链表

线性表分类（按操作）

- 栈(LIFO, Last In First Out)

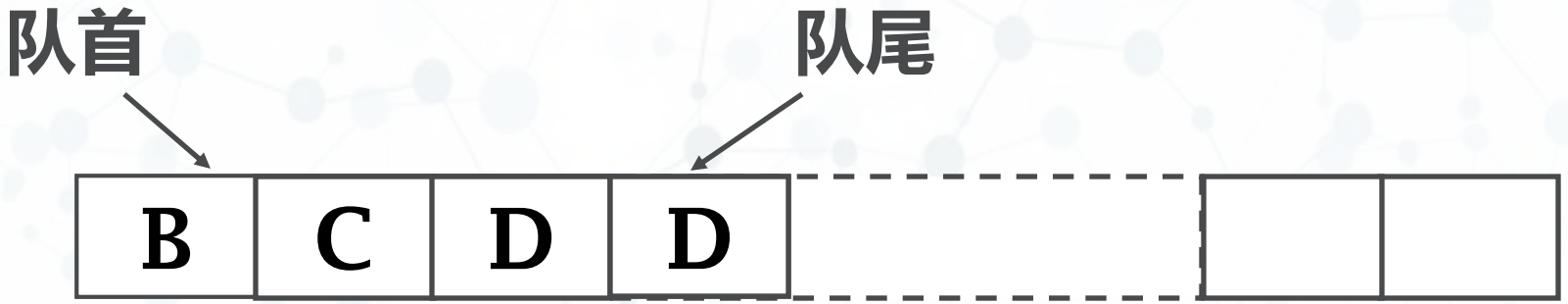
  插入和删除操作都限制在表的**同一端**进行



线性表分类（按操作）

- 队列(FIFO, First In First Out)

  插入操作在表的**一端**，**删除**操作在**另一端**



线性表类模板

```
template <class T> class List {
    void clear();          // 置空线性表
    bool isEmpty();        // 线性表为空时，返回true
    bool append(const T value);
                           // 在表尾添加一个元素value，表的长度增1
    bool insert(const int p, const T value);
                           // 在位置p上插入一个元素value，表的长度增1
    bool delete(const int p);
                           // 删除位置p上的元素，表的长度减 1
    bool getPos(int &p, const T value);
                           // 查找值为value的元素并返回其位置
    bool getValue(const int p, T &value);
                           // 把位置p元素值返回到变量value
    bool setValue(const int p, const T value);
                           // 用value修改位置p的元素值
};
```

2.2 顺序表

- 也称**向量**。固定长度的一维数组

$$Loc(k_i) = Loc(k_0) + c \times i,\ c = \text{sizeof}(ELEM)$$

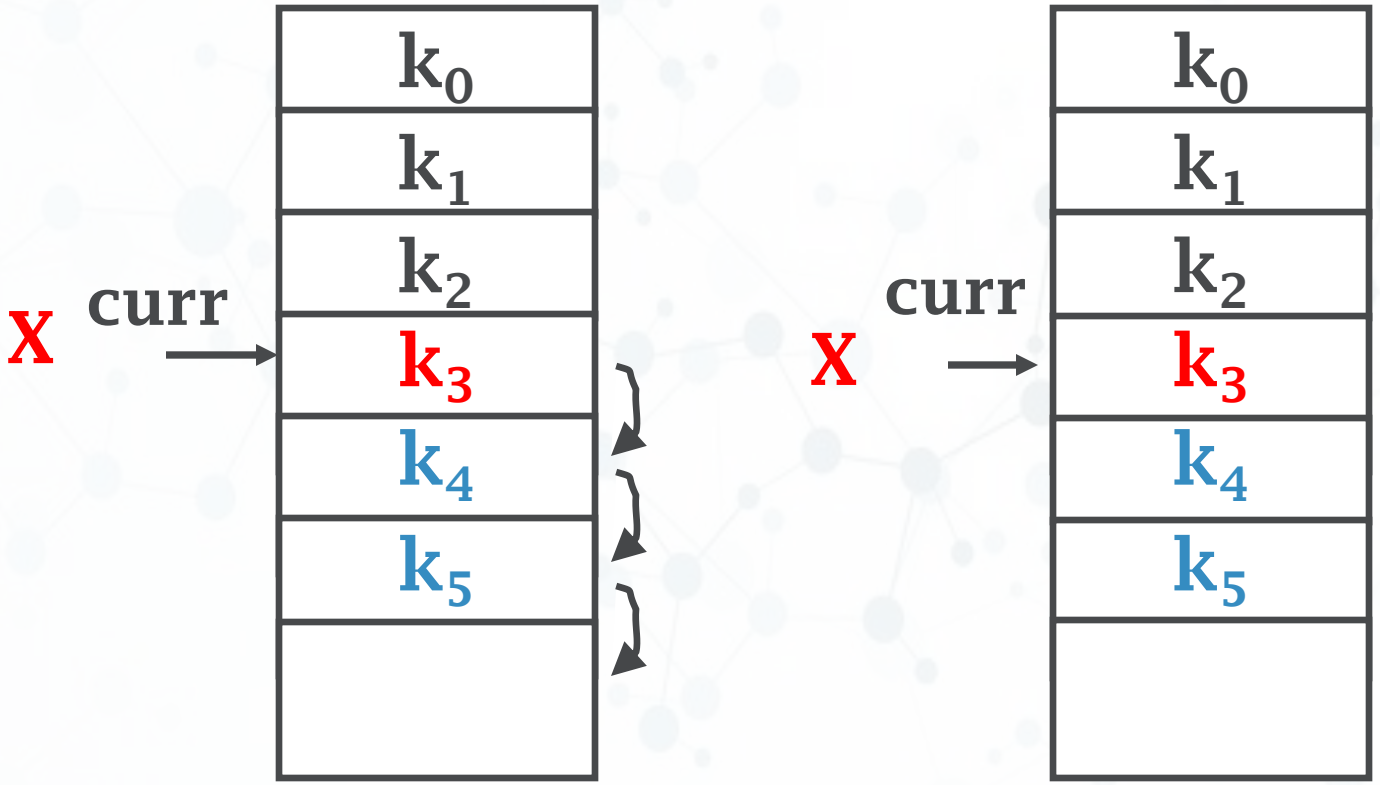
逻辑地址(下标)	数据元素
0	k_0
1	k_1
…	…
i	k_i
…	
n-1	k_{n-1}

存储地址	数据元素
$Loc(k_0)$	k_0
$Loc(k_0)+c$	k_1
…	…
$Loc(k_0)+i*c$	k_i
…	
$Loc(k_0)+(n-1)*c$	k_{n-1}

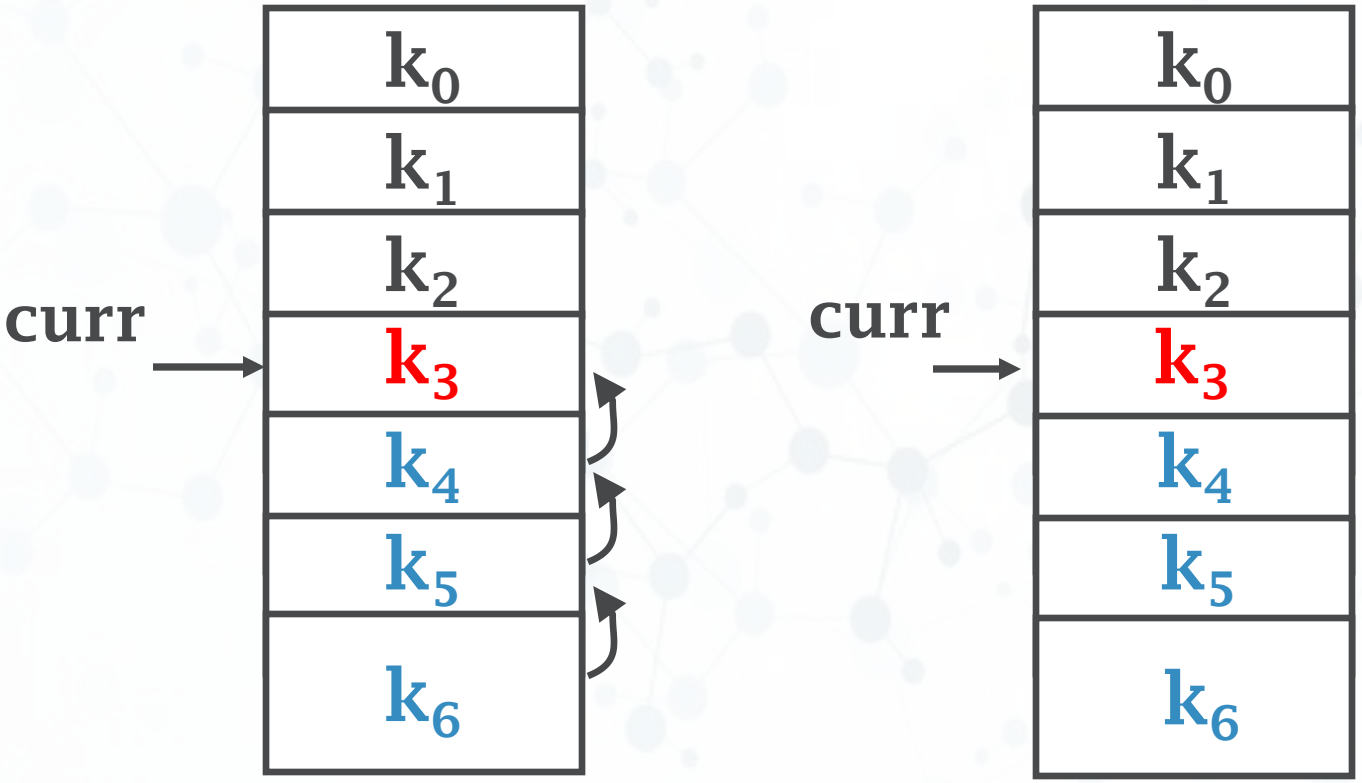
顺序表类定义

```
class arrList : public List<T> { // 顺序表，向量
private:                         // 线性表的取值类型和取值空间
    T * aList ;                  // 私有变量，存储顺序表的实例
    int maxSize;                 // 私有变量，顺序表实例的最大长度
    int curLen;                  // 私有变量，顺序表实例的当前长度
    int position;                // 私有变量，当前处理位置
public:
    int  length();                                  // 返回当前实际长度
    bool append(const T value);                     // 在表尾添加元素v
    bool insert(const int p, const T value);        // 插入元素
    bool delete(const int p);                       // 删除位置p上元素
    bool setValue(const int p, const T value);// 设元素值
    bool getValue(const int p, T &value);           // 返回元素
    bool getPos(int &p, const T value);             // 查找元素
};
```

顺序表的插入图示



顺序表的删除图示



顺序表各运算的算法分析

- 插入和删除操作的主要代价体现在表中元素的移动

  插入：移动 $n-i$

  删除：移动 $n-i-1$ 个

- i 的位置上插入和删除的概率分别是 p_i 和 p_i'

  插入的平均移动次数为 $M_i = \sum_{i=0}^{n}(n-i)p_i$

  删除的平均移动次数为 $M_d = \sum_{i=0}^{n-1}(n-i-1)p_i'$

算法分析

- 如果在顺序表中每个位置上插入和删除元素的概率相同，即$p_i = \frac{1}{n+1}$，$p_i' = \frac{1}{n}$

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

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

时间代价为$O(n)$

顺序表的优缺点

- 优点
  - 不需要附加空间
  - 随机存取任一个元素（根据下标）
- 缺点
  - 很难估计所需空间的大小
  - 开始就要分配足够大的一片连续的内存空间
  - 更新操作代价大

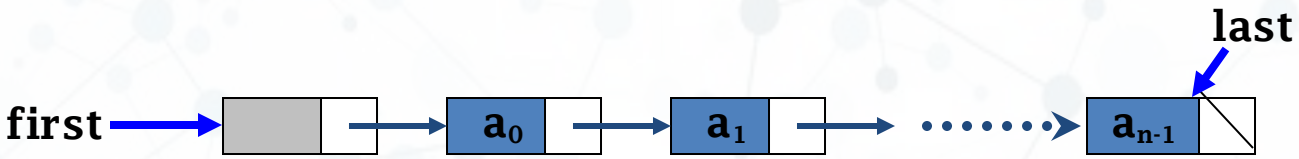
2.3 链表

- 存储利用指针

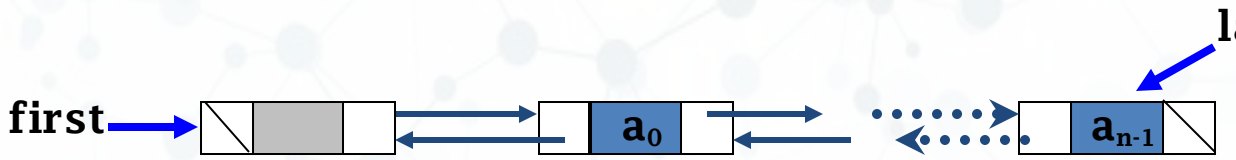
  动态地按照需要为表中新的元素分配存储空间

- 分类

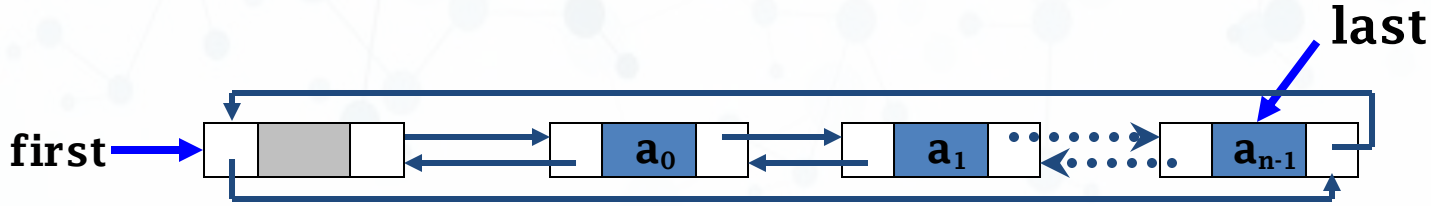
单链



双链

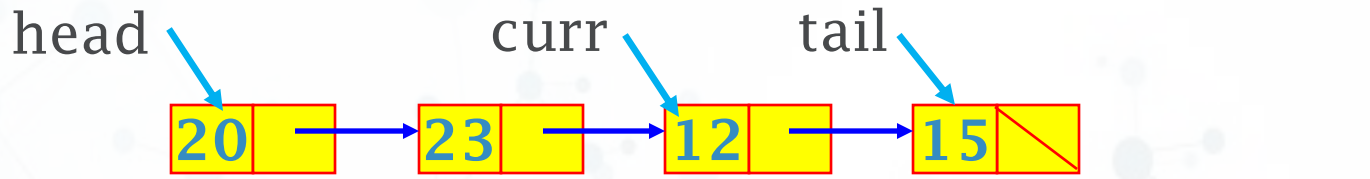


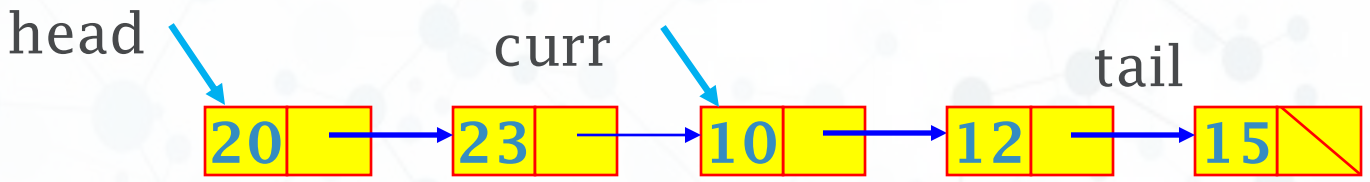
循环链



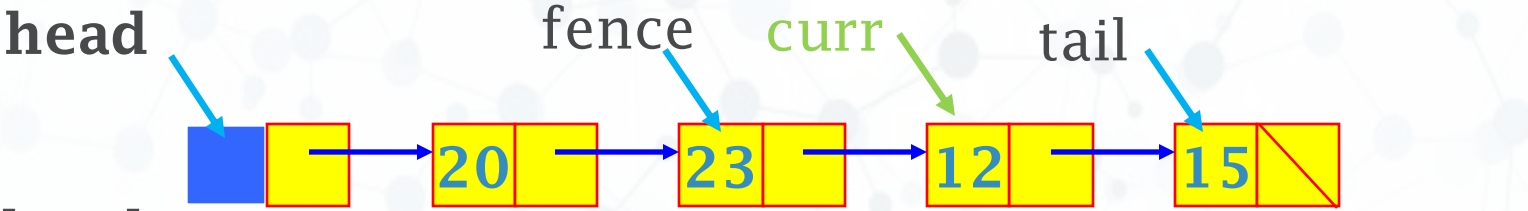
不带头结点 vs 带头结点

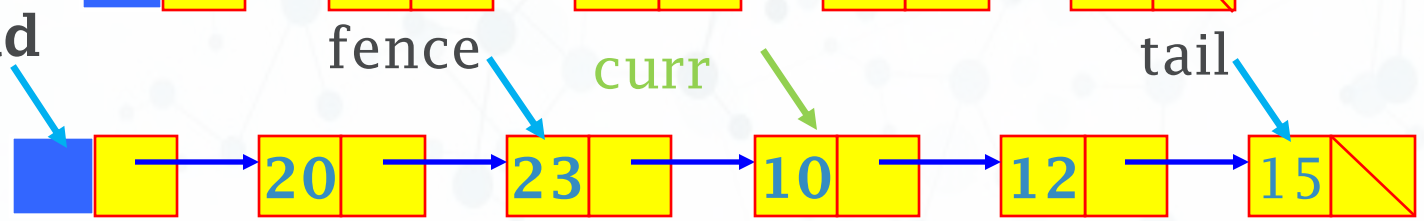
不带头结点





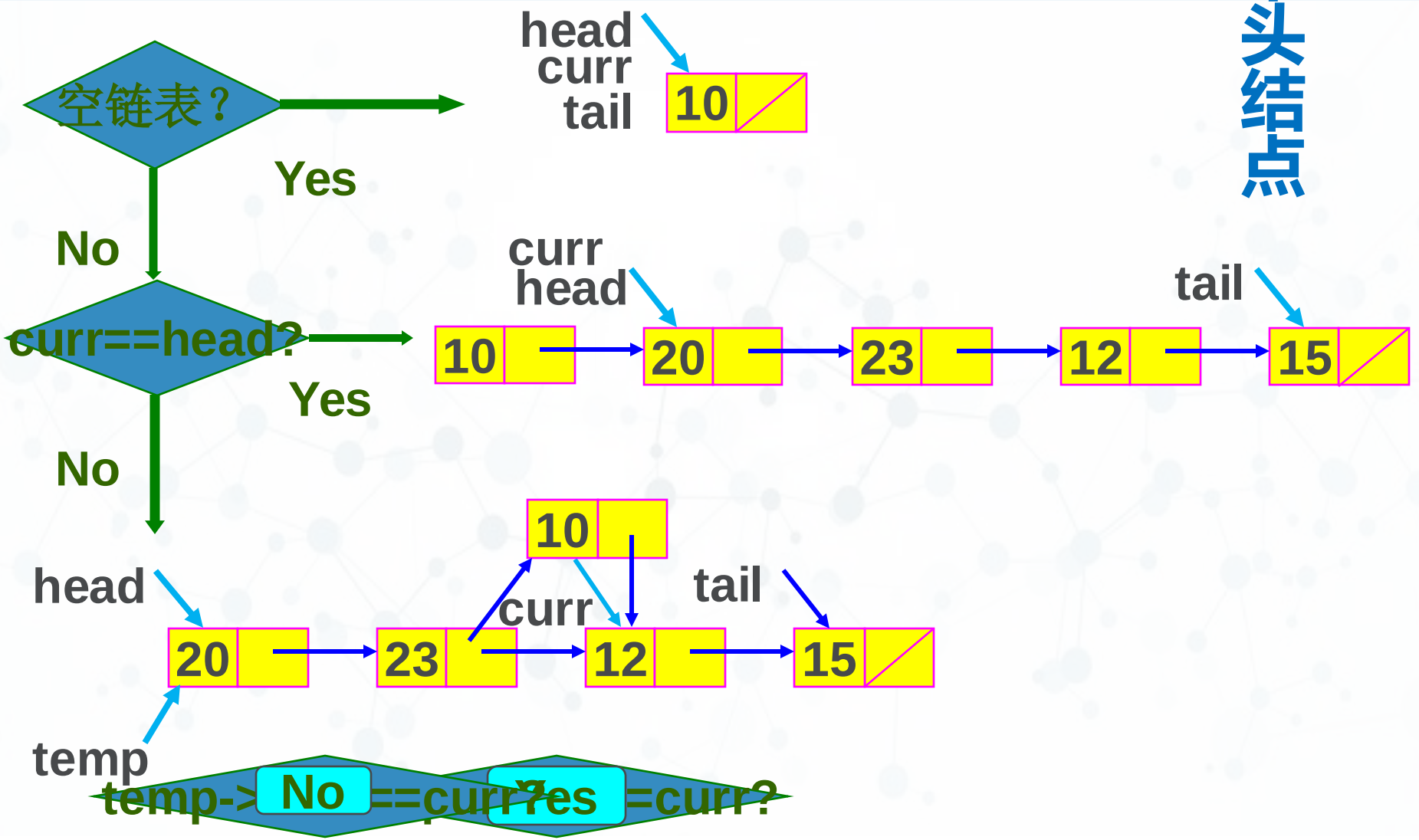
带头结点



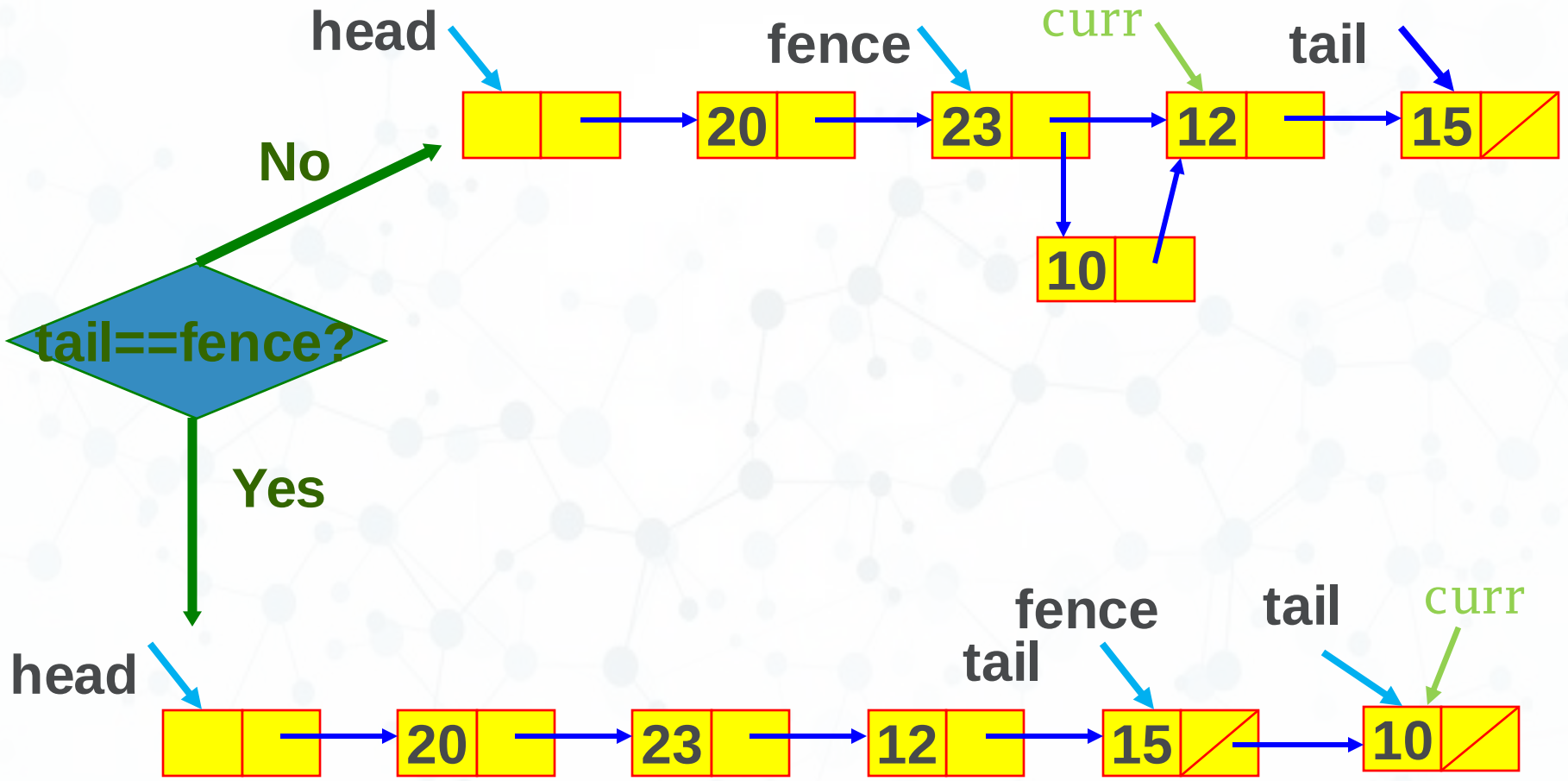


在指定位置前插入数值10

不带头结点



带头结点，当前位置前插入结点10



单链表上运算的分析

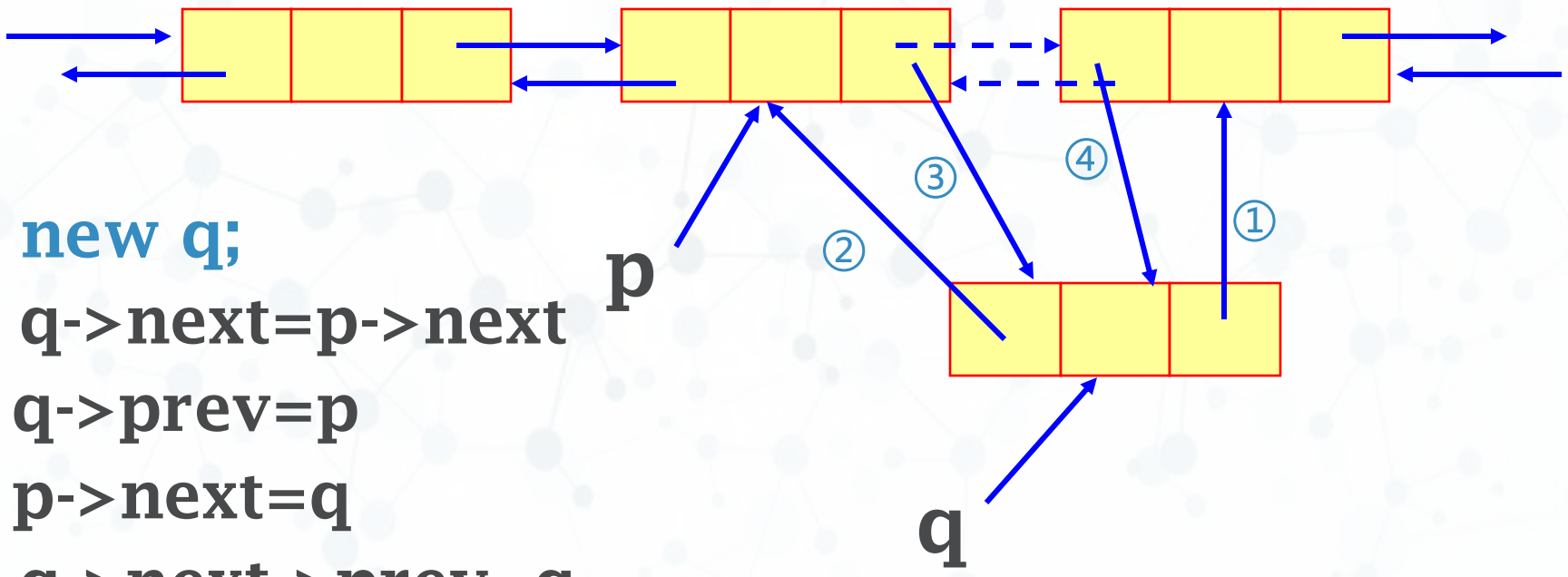
1. 对一个结点操作，必先找到它，即用一个指针指向它
2. 找单链表中任一结点，**都必须从第一个点开始**

```
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：$O(n)+O(1)$

双链表插入过程（注意顺序）

在p所指结点后面插入一个新结点

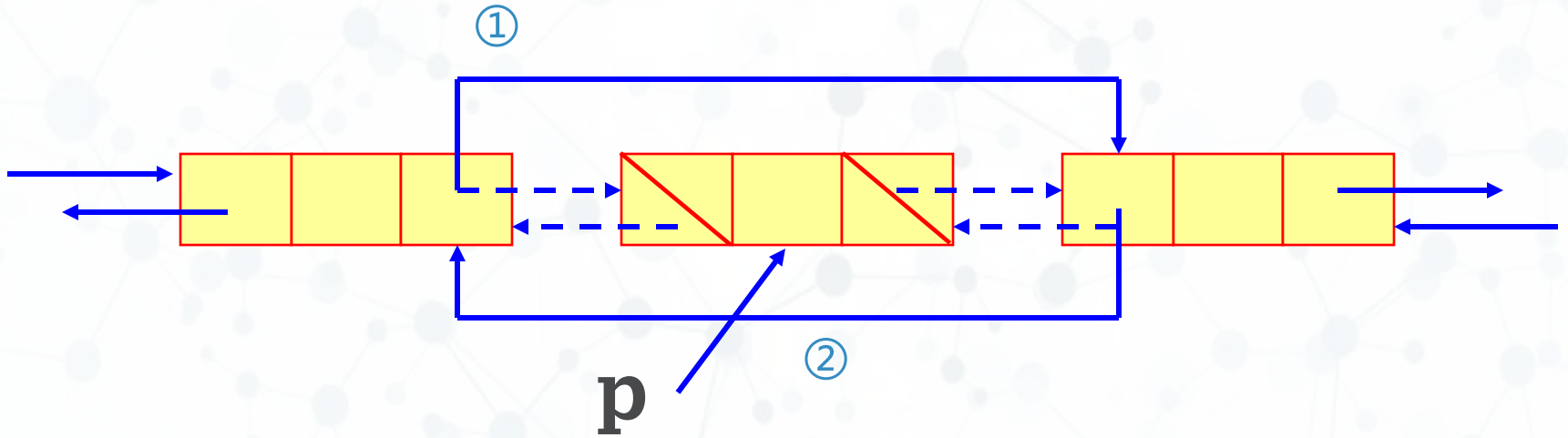



```
new q;
q->next=p->next
q->prev=p
p->next=q
q->next->prev=q
```

删除过程

- **如果马上删除p**
  则可以不赋空

删除p所指的结点



p->prev->next=p->next
p->next->prev=p->prev

p->next=NULL
p->prev=NULL

思考

- 带表头与不带表头的单链表?
- 处理链表需要注意的问题?
- 线性表实现方法的比较?
  插入、删除、查找等代价
- 顺序表和链表的选择?
  结点变化的动态性
  存储密度