数据结构与算法（二）

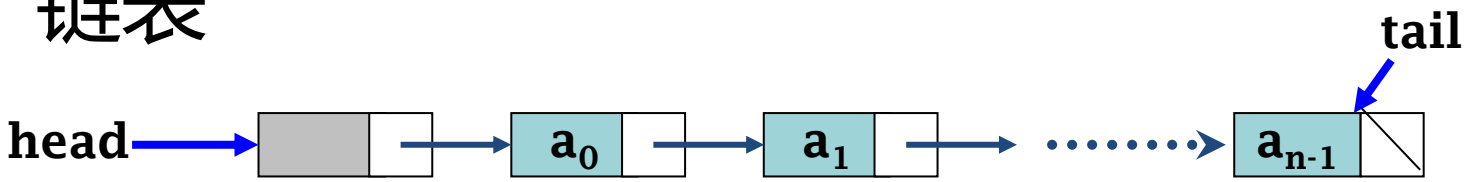
张铭 主讲

采用教材：张铭，王腾蛟，赵海燕 编写
高等教育出版社，2008.6（“十一五”国家级规划教材）

https://pkumooc.coursera.org/bdsalgo-001/

第二章 线性表

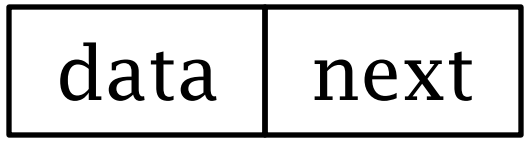
- 2.1 线性表
- 2.2 顺序表
- 2.3 链表



- 2.4 顺序表和链表的比较

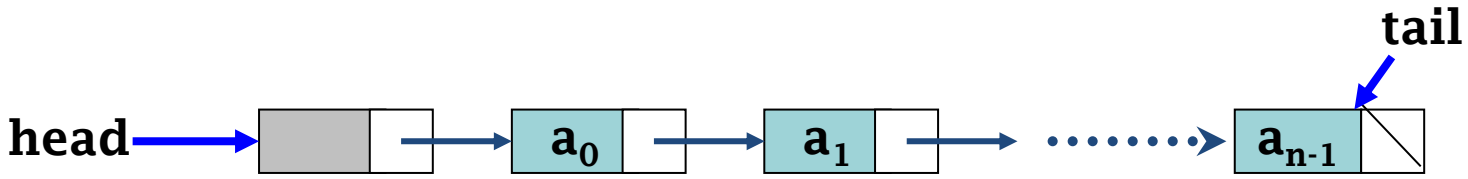
链表 (linked list)

- 通过指针把它的一串存储结点链接成一个链
- 存储结点由两部分组成：
  - 数据域 + 指针域（后继地址）

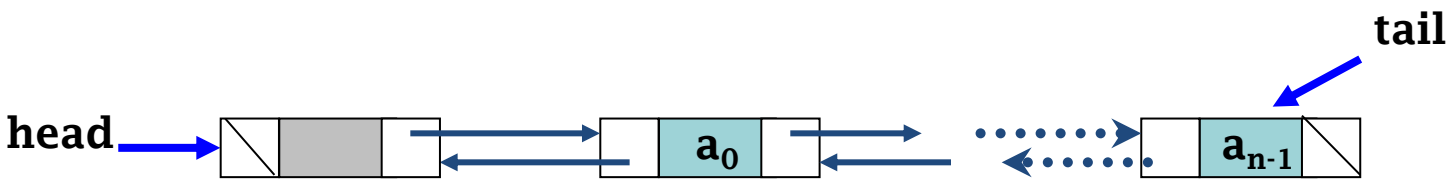


2.3 链表

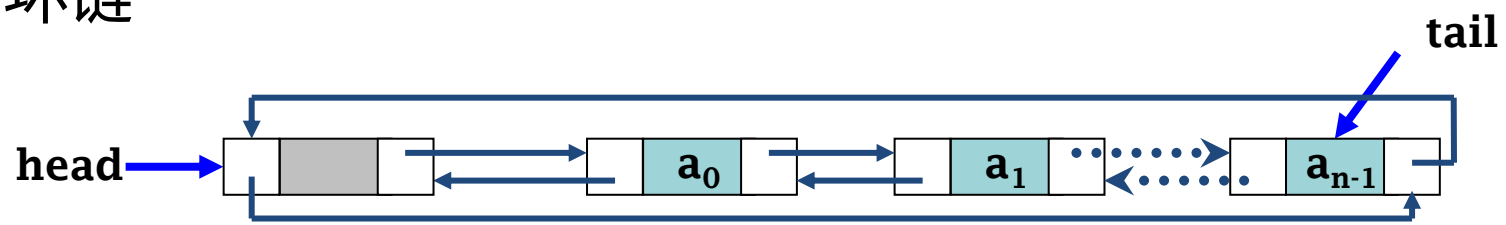
- 分类（根据链接方式和指针多寡）
  - 单链



  - 双链

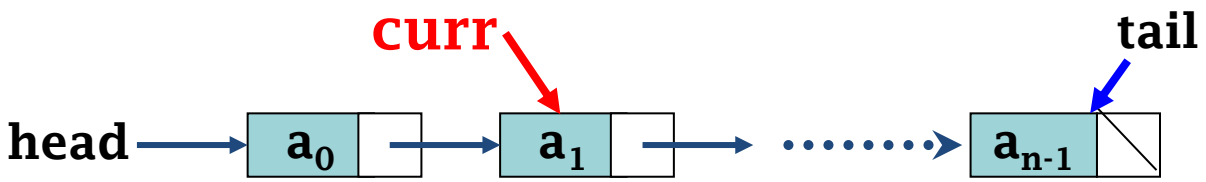


  - 循环链



单链表 (singly linked list)

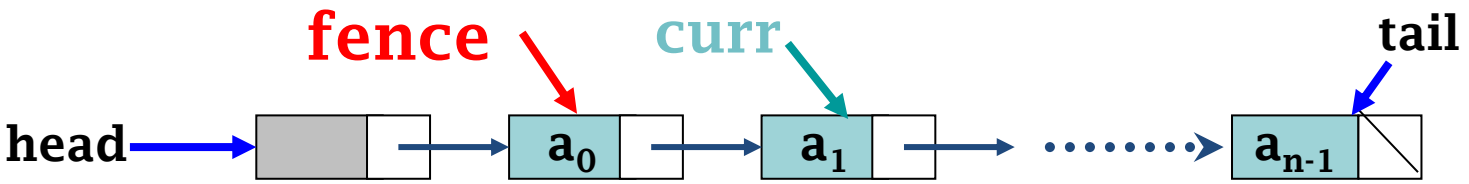
- 简单的单链表
  - 整个单链表： head
  - 第一个结点： head
  - 空表判断： head == NULL
  - 当前结点 a_1 ：curr



单链表 (singly linked list)

- 带头结点的单链表
  - 整个单链表： head
  - 第一个结点： head->next , head ≠ NULL
  - 空表判断： head->next == NULL

  当前结点a_1：fence->next (curr 隐含)



单链表的结点类型

```
template <class T> class Link {
    public:
    T    data;                    // 用于保存结点元素的内容
    Link<T> * next;               // 指向后继结点的指针

    Link(const T info, const Link<T>* nextValue =NULL)  {
        data = info;
        next = nextValue;
    }
    Link(const Link<T>* nextValue)  {
        next = nextValue;
    }
};
```

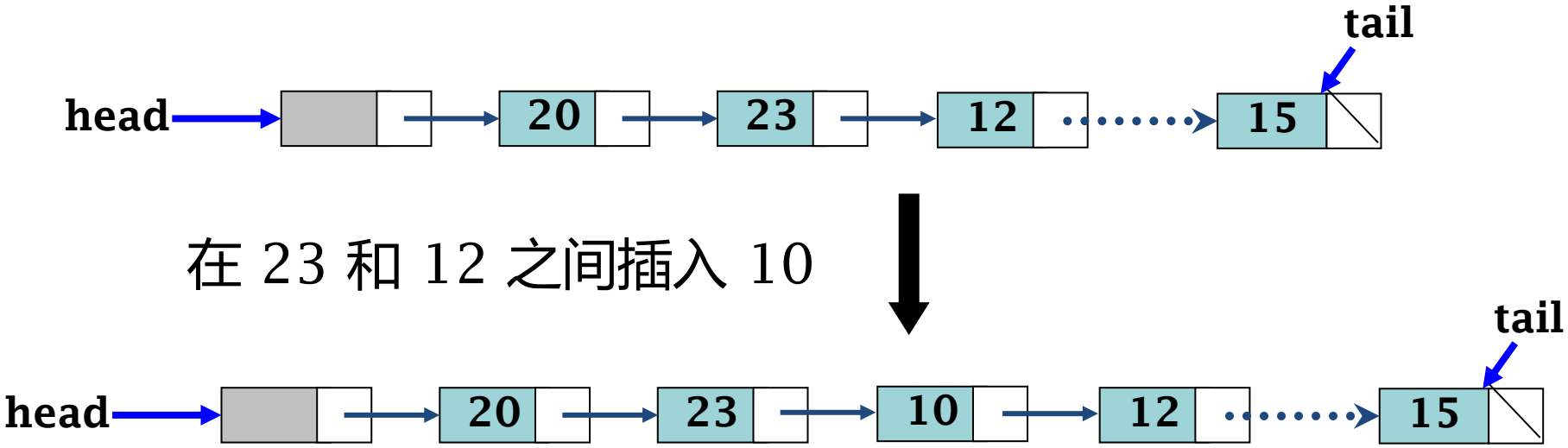
单链表类定义

```
template <class T> class lnkList : public List<T> {
    private:
    Link<T>  * head, *tail;                    // 单链表的头、尾指针
    Link<T>  *setPos(const int p);             // 第p个元素指针
     public:
    lnkList(int s);                            // 构造函数
    ~lnkList();                                // 析构函数
    bool isEmpty();                            // 判断链表是否为空
    void clear();                              // 将链表存储的内容清除，成为空表
    int length();                              // 返回此顺序表的当前实际长度
    bool append(cosnt T value);                // 表尾添加一个元素 value，表长度增 1
    bool insert(cosnt int p, cosnt T value); // 位置 p 上插入一个元素
    bool delete(cosnt int p);                  // 删除位置 p 上的元素，表的长度减 1
    bool getValue(cosnt int p, T& value);    // 返回位置 p 的元素值
    bool getPos(int &p, const T value);      // 查找值为 value 的元素
}
```

查找单链表中第 i 个结点

```
// 函数返回值是找到的结点指针
template <class T>          // 线性表的元素类型为 T
Link<T> * lnkList <T>:: setPos(int i) {
    int count = 0;
     if (i == -1)                  //  i 为 -1 则定位到头结点
        return head;
    // 循链定位，若i为0则定位到第一个结点
    Link<T> *p = head->next;
    while (p != NULL && count < i) {
        p = p-> next;
        count++;
     };
     // 指向第 i 结点，i = 0,1,…，当链表中结点数小于 i 时返回 NULL
    return p;
}
```

单链表的插入



- 创建新结点
- 新结点指向右边的结点
- 左边结点指向新结点

单链表插入算法

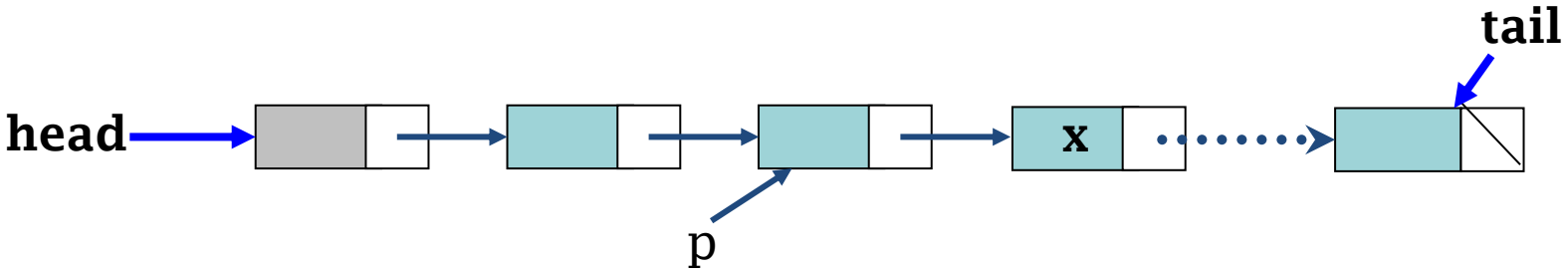
```
// 插入数据内容为value的新结点作为第 i 个结点
template <class T>                          // 线性表的元素类型为 T
bool lnkList<T> :: insert(const int i, const T value)  {
    Link<T> *p, *q;

    if ((p = setPos(i -1)) == NULL)  {    // p 是第 i 个结点的前驱
        cout << " 非法插入点"<< endl;
        return false;
    }
    q = new Link<T>(value, p->next);
    p->next = q;
    if (p == tail)                  // 插入点在链尾，插入结点成为新的链尾
        tail = q;
    return true;
}
```

单链表的删除

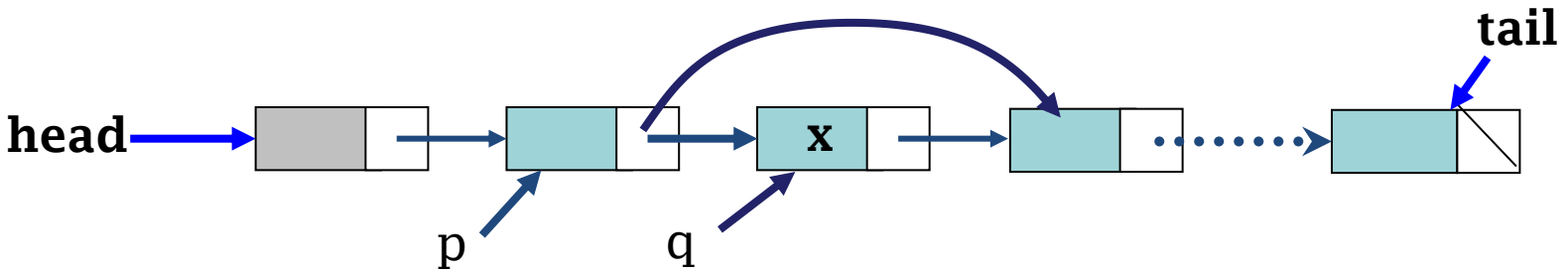
- 从链表中删除结点 x
  - 1. 用 p 指向元素 x 的结点的前驱结点
  - 2. 删除元素为 x 的结点
  - 3. 释放 x 占据的空间

单链表删除示意




```
p = head;
while (p->next!=NULL && p->next->info!= x)
        p = p->next;
```

删除值为 x 的结点




```
q = p->next;
p->next = q->next;
free(q);
```

单链表删除算法

```
template <class T>                    // 线性表的元素类型为 T
bool lnkList<T>:: delete((const int i) {
    Link<T> *p, *q;
    // 待删结点不存在，即给定的i大于当前链中元素个数
    if  ((p = setPos(i-1)) == NULL  ||  p == tail) {
        cout << " 非法删除点 " << endl;
        return false;
    }
    q = p->next;                      // q 是真正待删结点
    if (q == tail) {                  // 待删结点为尾结点，则修改尾指针
        tail = p;
        p->next = NULL:
    }
    else                              // 删除结点 q 并修改链指针
        p->next = q->next;
    delete q;
    return true;
}
```

单链表上运算的分析

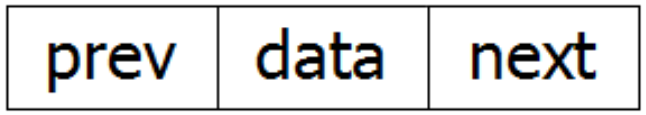
- 对一个结点操作，必先找到它，即用一个指针指向它
- 找单链表中任一结点，**都必须从第一个点开始**

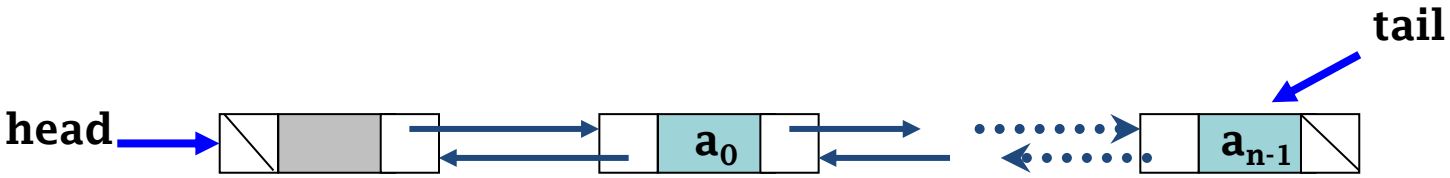
```
p = head;
while (没有到达) p = p->next;
```

- 单链表的时间复杂度 $O(n)$
  - 定位：$O(n)$
  - 插入：$O(n)+O(1)$
  - 删除：$O(n)+O(1)$

双链表(double linked list)

- 为弥补单链表的不足,而产生双链表
  - 单链表的 next 字段仅仅指向后继结点，不能有效地找到前驱, 反之亦然
  - 增加一个指向前驱的指针



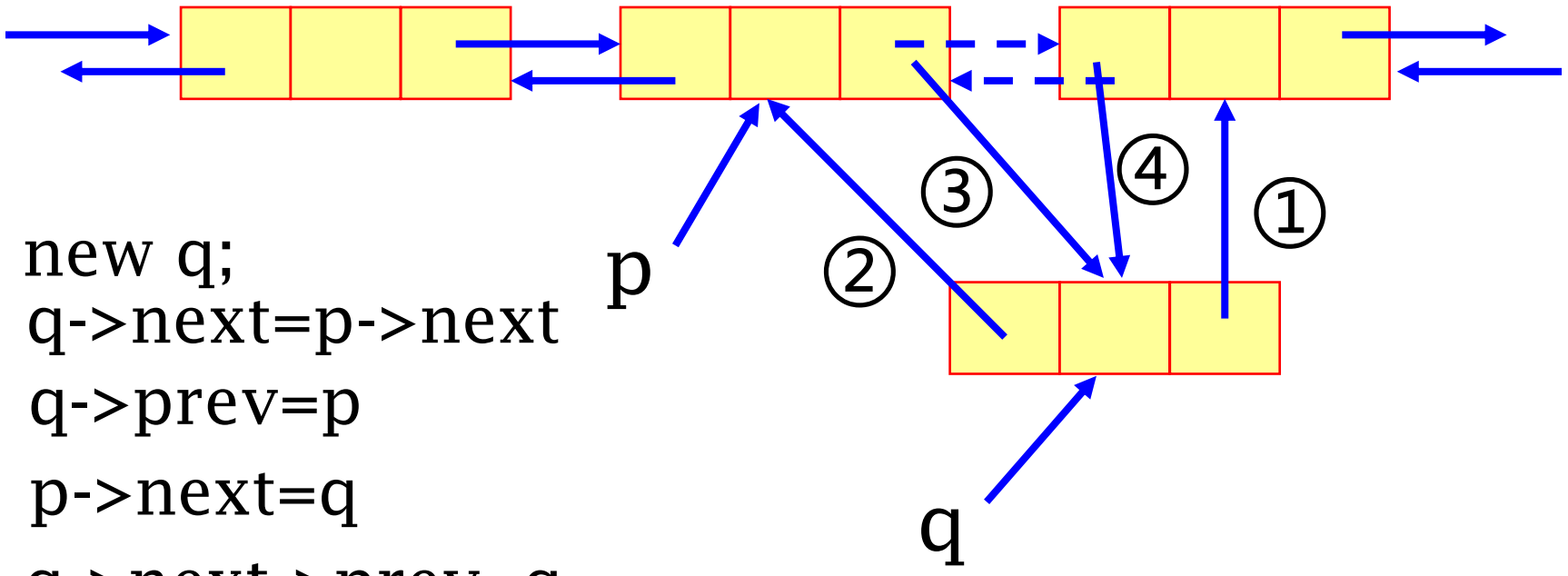


双链表及其结点类型的说明

```
template <class T> class Link {
    public:
    T  data;                    // 用于保存结点元素的内容
    Link<T> * next;             // 指向后继结点的指针
    Link<T> *prev;                  // 指向前驱结点的指针
    Link(const T info, Link<T>* preValue = NULL, Link<T>* nextValue = NULL) {
    // 给定值和前后指针的构造函数
        data = info;
        next = nextValue;
        prev = preValue;
    }
    Link(Link<T>* preValue = NULL, Link<T>* nextValue = NULL)  {
    // 给定前后指针的构造函数
        next = nextValue;
        prev = preValue;
    }
}
```

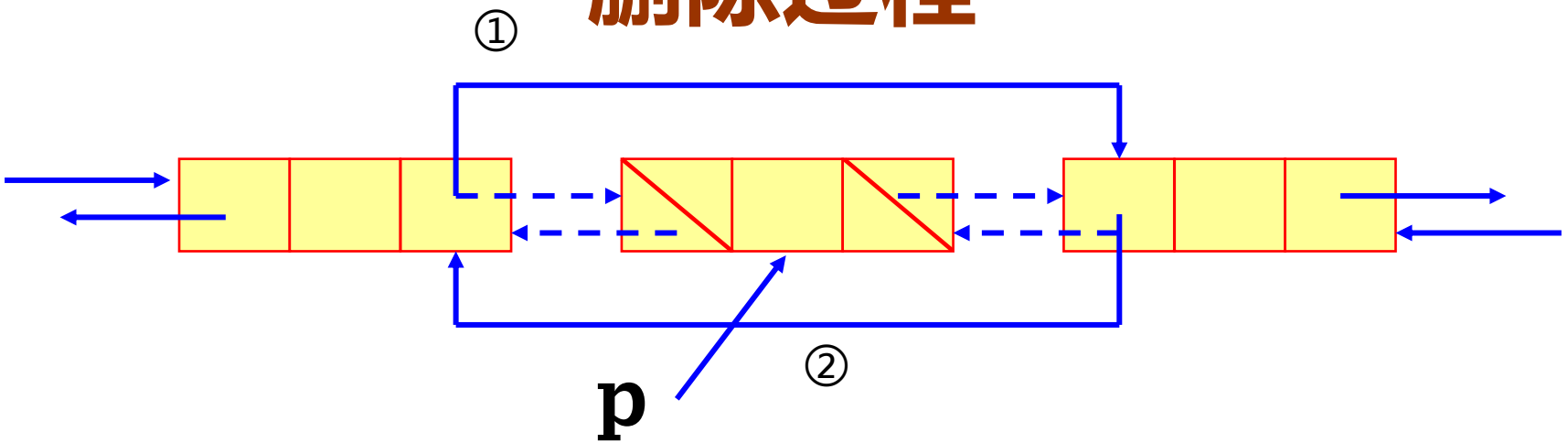
双链表插入过程（注意顺序）

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



删除 p 所指的结点

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

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

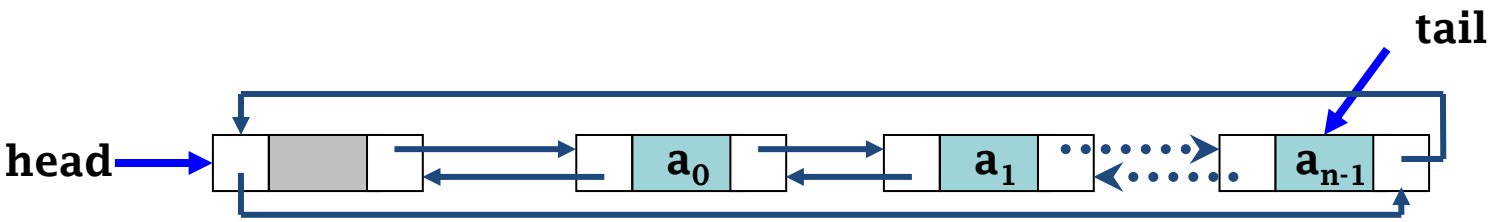
p->next=NULL

p->prev=NULL

- 如果马上删除 p
  - 则可以不赋空

循环链表 (circularly linked list)

- 将单链表或者双链表的头尾结点链接起来，就是一个循环链表
- 不增加额外存储花销，却给不少操作带来了方便
  - 从循环表中任一结点出发，都能访问到表中其他结点



链表的边界条件

- 几个特殊点的处理
  - 头指针处理
  - 非循环链表尾结点的指针域保持为 NULL
  - 循环链表尾结点的指针回指头结点
- 链表处理
  - 空链表的特殊处理
  - 插入或删除结点时指针勾链的顺序
  - 指针移动的正确性
    - 插入
    - 查找或遍历

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？

数据结构与算法

谢谢聆听

国家精品课“数据结构与算法”
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

张铭，王腾蛟，赵海燕
高等教育出版社，2008. 6。“十一五”国家级规划教材