数据结构

第 10 周 – 带哨兵的链表、双向链表、循环链表

Koczka Ferenc 博士 · 埃斯特哈齐·卡罗伊大学 · 2026

复习

- **单链表：** Head → … → Nil；顺序访问
- 在表头和已知结点**之后**：O(1)；在表尾和查找：O(n)
- **数组中的哨兵查找**（第 2 周）：把要找的值放在末尾 → 一个条件就够了

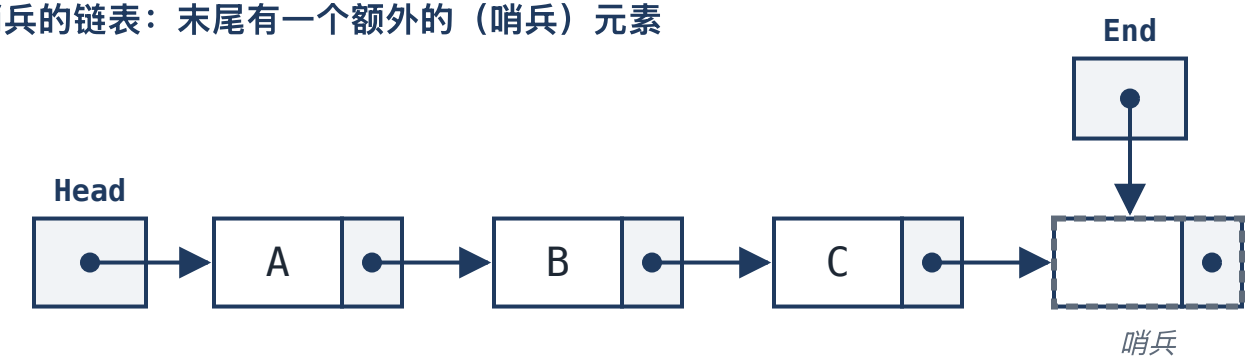
单链表的问题

- **特殊情况**很多：空链表、第一个结点、最后一个结点
- 查找循环有**两个条件**：`P ≠ Nil and Elem[P].Value ≠ x`
- 不能在某个结点**之前**插入，也不能删除**结点本身**（只能删除它后面的结点）

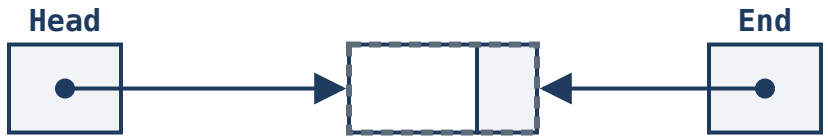
带哨兵的链表

带哨兵链表的结构

带哨兵的链表：末尾有一个额外的（哨兵）元素



空的带哨兵链表：



Head = End （存储空间不为零！）

- 链表末尾有一个**哨兵**（守卫）结点，它不存放真正的数据
- **Head**：指向第一个结点，**End**：指向哨兵
- 空链表：**Head = End** – 但即使这时也有**一个结点**（哨兵）占用空间！

带哨兵链表的操作

操作	思路	步数
创建	分配一个哨兵，Head ← End ← 哨兵	O(1)
处理	从 Head 到 End（而不是到 Nil!）	O(n)
查找	把要找的值写入哨兵	O(n)
在前面插入	复制技巧	O(1)
删除结点	复制技巧	O(1)
清空	Head ← End（结点放回空闲链表）	O(1) – O(n)

性质	同构 · 动态 · 链式 · 顺序访问

创建与处理

创建

```
Procedure Create(Head, End)
  If Reserve(P) then
    Head ← P
    End ← P
  else
    ERROR("no free memory")
  End if
End procedure
```

处理

```
Procedure Process(Head, End)
  P ← Head
  While P ≠ End
    Operation(Elem[P].Value)
    P ← Elem[P].Next
  End while
End procedure
```

Java

```java
class SentinelList {
    Node head, end;

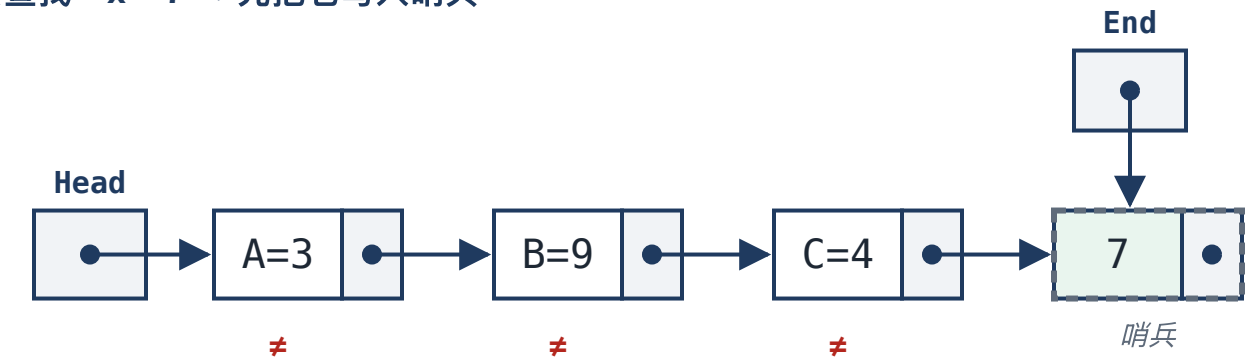
    SentinelList() {
        end = new Node();   // 哨兵
        head = end;
    }

    boolean isEmpty() { return head == end; }

    void print() {
        for (Node p = head; p != end; p = p.next)
            System.out.print(p.value + " ");
    }
}
```

哨兵查找

哨兵查找：x = 7 → 先把它写入哨兵

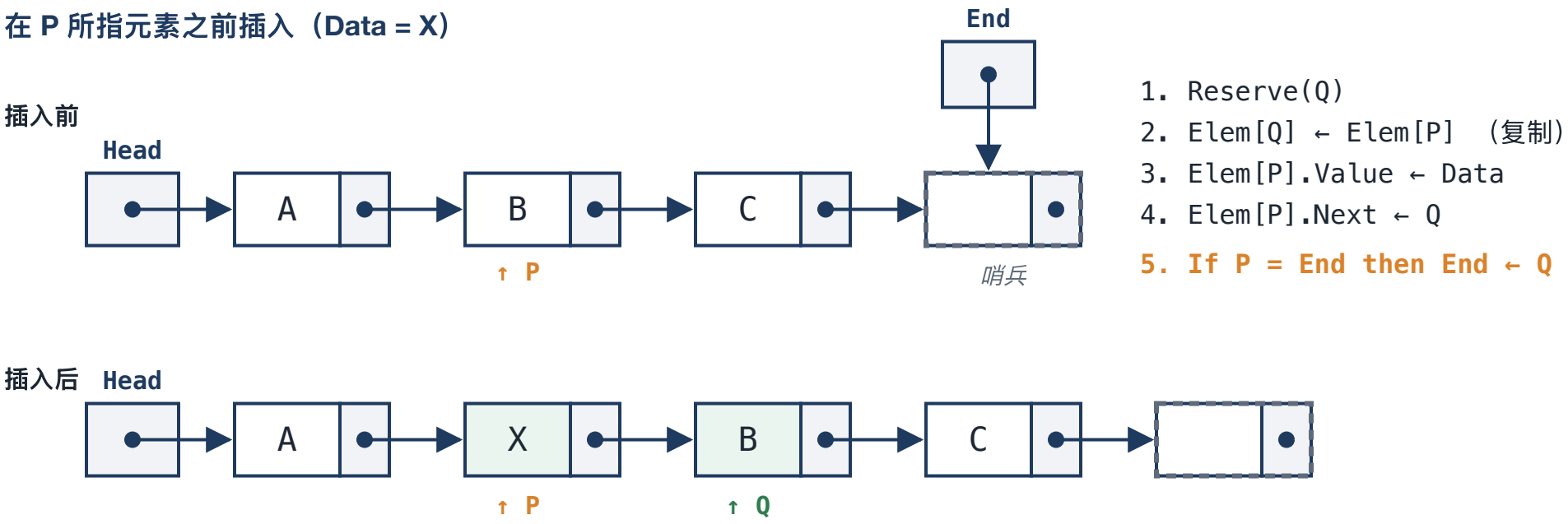


= → P = End：未找到

```
Function SentinelSearch(Head, End, x)
  Elem[End].Value ← x                    // 写入哨兵
  P ← Head
  While Elem[P].Value ≠ x                // 只有一个条件!
    P ← Elem[P].Next
  End while
  If P = End then Return Nil  else Return P
End function
```

在给定结点之前插入

在 P 所指元素之前插入（Data = X）




```
1. Reserve(Q)
2. Elem[Q] ← Elem[P]  (复制)
3. Elem[P].Value ← Data
4. Elem[P].Next ← Q
5. If P = End then End ← Q
```

技巧：我们无法在 P **之前**插入（不知道前一个结点），但可以在它**之后**插入。所以把 P 的**内容**复制到新结点中，再把新数据写入 P。

在前面插入 – 伪代码和代码

伪代码

```
Procedure InsertBefore(Head, End, P, Data)
  If Reserve(Q) then
    Elem[Q] ← Elem[P]       // Value 和 Next
    Elem[P].Value ← Data
    Elem[P].Next ← Q
    If P = End then
      End ← Q               // Q 是新的哨兵
    End if
  else
    ERROR("no free memory")
  End if
End procedure
```

在表尾插入 = 在哨兵之前插入：

`InsertBefore(Head, End, End, Data)` → **O(1)**!

在表头插入 = 在 Head 之前插入

Java

```java
void insertBefore(Node p, int x) {
    Node q = new Node();
    q.value = p.value;
    q.next = p.next;
    p.value = x;
    p.next = q;
    if (p == end) end = q;
}

void insertLast(int x)  { insertBefore(end, x); }
void insertFirst(int x) { insertBefore(head, x); }
```

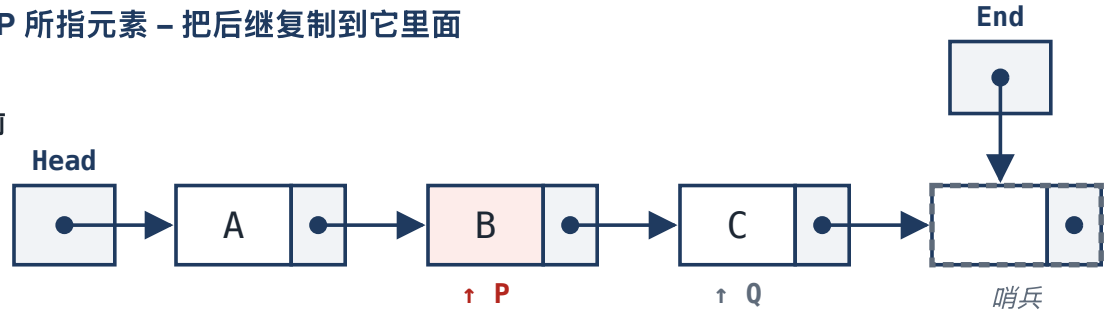
Go

```go
func (l *SentinelList) InsertBefore(p *Node, x int) {
    q := &Node{Value: p.Value, Next: p.Next}
    p.Value = x
    p.Next = q
    if p == l.End {
        l.End = q
    }
}
```

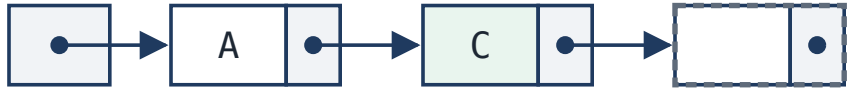
删除给定的结点

删除 P 所指元素 – 把后继复制到它里面

插入前



插入后



1. Q ← Elem[P].Next
2. Elem[P] ← Elem[Q] (C 放到 B 的位置)
3. **If Q = End then End ← P**
4. Free(Q)

```
Procedure Delete(Head, End, P)        // P ≠ End
  Q ← Elem[P].Next
  Elem[P] ← Elem[Q]                   // 把后继结点的内容放入 P
  If Q = End then End ← P             // P 成为新的哨兵
  Free(Q)
End procedure
```

删除**第一个**结点也不再是特殊情况 – Head 不变!

哨兵带来了什么好处?

	单链表	带哨兵的链表
空链表	Head = Nil	Head = End（空间：1 个结点）
查找的循环条件	2 个条件	**1 个条件**
在前面插入	不可能（只能在后面）	**O(1)**
删除给定结点	不可能（只能删除后面的）	**O(1)**
在表尾插入	O(n)	**O(1)**
特殊情况（第一个结点）	很多	**没有**

复制技巧会**改变哪个结点存放哪个值** – 如果外部有指针指向这些结点，它们可能会“错位”。对于很大的记录，复制的代价也很高。

有序的带哨兵链表

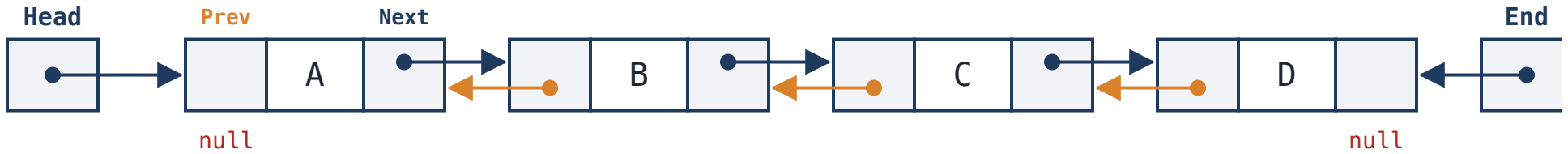
哨兵也简化了有序插入：我们把一个**无穷大**的值写入哨兵，这样查找一定会停止。

```
Procedure SortedInsert(Head, End, Data)
  Elem[End].Value ← +∞
  P ← Head
  While Elem[P].Value < Data
    P ← Elem[P].Next
  End while
  InsertBefore(Head, End, P, Data)
End procedure
```

依次插入 `5, 2, 8, 1, 9, 3` 之后：`[1 2 3 5 8 9]` – 空链表、表头和表尾都用同一段代码！

双向链表

双向链表的结构



每个元素存放两个指针：前驱和后继

- 每个结点都存放**前一个**和**后一个**结点的地址
- **Head** 和 **End** 指针：链表可以双向遍历
- 空间：每个结点**两个**指针

双向链表 – 操作

删除结点 P – O(1)

```
Procedure Delete(Head, End, P)
  If Elem[P].Prev ≠ Nil then
    Elem[Elem[P].Prev].Next ← Elem[P].Next
  else
    Head ← Elem[P].Next
  End if
  If Elem[P].Next ≠ Nil then
    Elem[Elem[P].Next].Prev ← Elem[P].Prev
  else
    End ← Elem[P].Prev
  End if
  Free(P)
End procedure
```

Java

```java
void remove(Node p) {
    if (p.prev != null) p.prev.next = p.next;
    else head = p.next;
    if (p.next != null) p.next.prev = p.prev;
    else tail = p.prev;
}

void insertLast(int x) {          // O(1)
    Node n = new Node(x);
    n.prev = tail;
    if (tail != null) tail.next = n;
    else head = n;
    tail = n;
}
```

不需要复制 – **我们知道两个邻居**。

在前面/后面插入、删除表尾：都是 **O(1)**。

实际中的双向链表

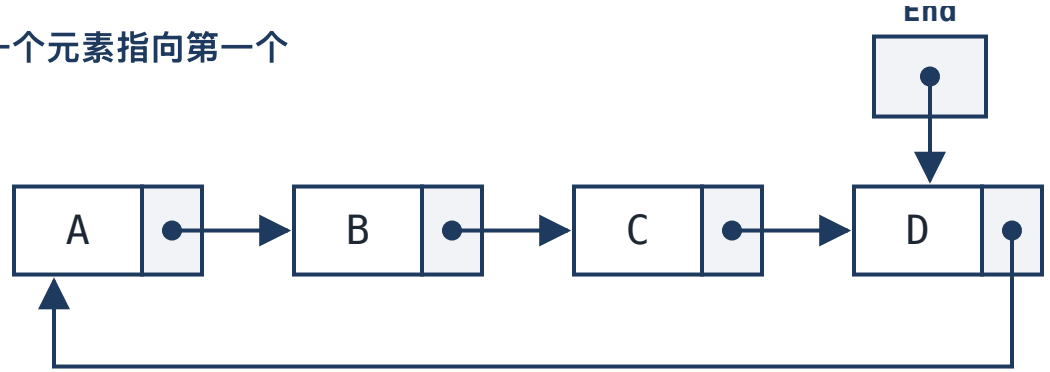
- Java `LinkedList`：双向链表 – `addFirst`、`addLast`、`removeFirst`、`removeLast` 都是 O(1)
- Go `container/list`：双向、带哨兵的循环链表
- **应用：**
  - 浏览器历史（后退**和**前进）
  - 音乐播放器：上一首 / 下一首
  - 文本编辑器：光标在行之间移动
  - **LRU 缓存：**快速删除最久未使用的元素（与哈希表配合）

> **带哨兵的双向链表：**只用一个哨兵，它既是第一个结点之前的结点，也是最后一个结点之后的结点 – 这样**所有**操作都可以不考虑特殊情况（Go 的 `container/list` 就是这样实现的）。

循环链表

循环链表

循环链表：最后一个元素指向第一个




```
first = Elem[End].Next
```

链中没有 Nil!

- 最后一个结点不指向 **Nil**，而是指向**第一个**结点
- 只需一个指向**最后一个**结点的指针：第一个结点 = `Elem[End].Next` → 在表头**和**表尾插入都是 O(1)
- 遍历：从第一个结点开始，直到**回到**起点（直到型循环！）

应用：时间片轮转调度、循环播放列表、约瑟夫问题、轮流进行的多人游戏

各种链表的比较

	单链表	带哨兵	双向	循环
每个结点的指针数	1	1	**2**	1
空链表占用的空间	0 个结点	**1 个结点**	0 个结点	0 个结点
在表头插入	O(1)	O(1)	O(1)	O(1)
在表尾插入	O(n)	**O(1)**	**O(1)**	**O(1)**
删除表尾	O(n)	O(n)	**O(1)**	O(n)
删除给定结点	O(n)*	**O(1)**	**O(1)**	O(n)*
反向遍历	✗	✗	✓	✗

* 需要找到前一个结点

小结

- **带哨兵的链表：** Head + End,End 指向一个空的哨兵结点；空链表：Head = End
  - 哨兵查找只需**一个**条件 · 用**复制技巧** O(1) 在前面插入和删除 · 没有特殊情况
- **双向链表：** Prev + Next 指针；可以双向遍历；删除 O(1)
- **循环链表：** 最后一个结点指向第一个；一个 End 指针就够了
- 选择取决于**最常用的操作**和**所需的空间**

下周： 基于链表的栈和队列，以及哈希表。

复习题

1. 判断对错：空的带哨兵链表不需要存储空间，因为其中没有存放任何元素。
2. 判断对错：存放相同序列的带哨兵链表和双向链表所需的空间总是相同的，因为两者都由两个指针（Head 和 End）给出。
3. 写出哨兵查找的伪代码！链表由 Head 和 End 指针给出。
4. 画出带哨兵的链表 `[3, 7]`，以及在 `7` 之前插入（Data = 5）之后的状态！
5. 为什么遍历循环链表需要直到型循环?
6. 如果经常需要删除链表的**表尾**，你会选择哪种链表?

实践课

✎ 示例程序

程序	Java	Go	演示内容
带哨兵的链表	`SentinelList.java`	`w10/sentinellist`	创建、哨兵查找、在前面插入、删除、有序插入
双向链表	`DoublyLinkedList.java`	`w10/doublylist`	正向/反向遍历、O(1) 删除、内置链表
循环链表	`CircularList.java`	`w10/circularlist`	遍历、约瑟夫问题

`SentinelList` 是旧作业中带哨兵链表部分的完整解答（初始化、处理、查找、插入、删除）。

✎ 练习

1. **最大值：**找出带哨兵链表中的最大元素！
2. **用哨兵计数：**`x` 出现了多少次？内层循环只能有一个条件！
3. **删除偶数：**用复制技巧删除带哨兵链表中所有的偶数！
4. **回文：**在双向链表上，用两个指针从两端检查！
5. **循环链表的大小：**统计循环链表中的结点个数！

参考答案：`Exercises10.java`、`w10/exercises`

✎ 课后作业

1. 实现只有一个哨兵的**带哨兵双向链表**（第一个结点之前和最后一个结点之后是同一个结点）！写出在前面插入、在后面插入和删除 – 不用一个 `if`！
2. 把两个有序的带哨兵链表合并成第三个有序的带哨兵链表！
3. 用双向链表实现 **LRU 缓存**：最近使用的元素放到最前面；满时从表尾删除！
4. 测量：从多少个元素开始，哨兵查找明显比普通查找快？