链表、列表

引入修改（插入、删除）“线性结构” 的能力

概念核心

call by link，只是有一个逻辑次序。

只能按顺序（但不一定只有单一顺序，可以是任意的引用）访问。

> 当然也可以 call by rank，只不过复杂度是 $O(n)$ 的。

实现

根本的单位是“节点”。

节点存储：

- value
- 两个“指针”: pred ，一个 succ；

游标实现

预先开好一个向量存放数据，所有的“指针”都用向量里面的秩替代。

构造与析构

构造的时候可以在前后加上哨兵方便操作。

析构的时候要顺着 link 释放掉全部的内存。

无序列表的操作

插入 $O(1)$ 或者 $O(1+n)$

删除 $O(1)$ 或者 $O(1+n)$

查找、遍历 $O(n)$

去重$O(n^2)$

有序列表操作

唯一化 $O(n)$

查找 $O(n)$

> 这里并没有利用有序性进行优化，因为扫描只能一个一个往后走。
>
> 能不能一下走不少？这是后面的跳转表 多层列表 结构的动机。

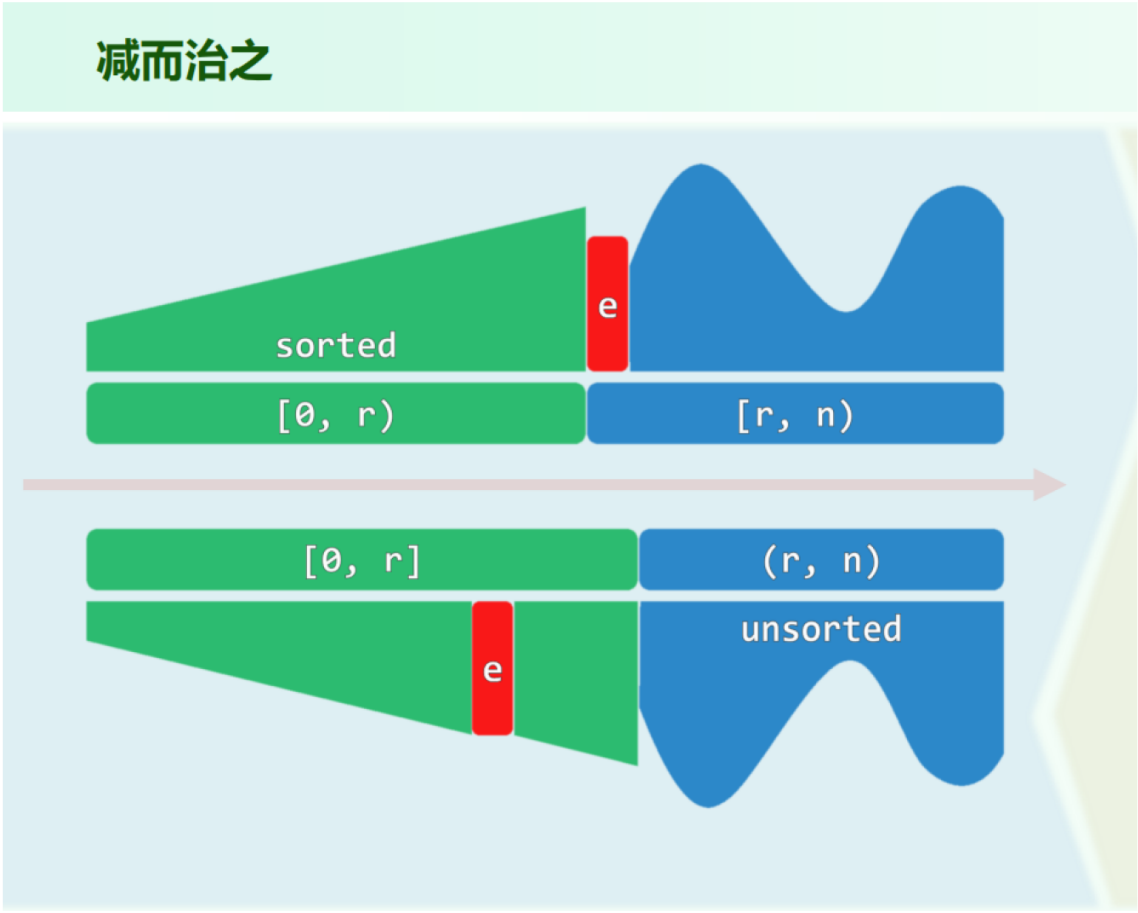
应用

插入排序

从前往后扫描，扫描位置之前总是有序的；每次将新遇到的元素，通过顺序查找（为什么不是二分查找？因为插入的 $O(1)$ 或者 $O(\log n)$ 和二分查找的 $O(\log n)$ 近乎不可能同时获得），插入到前面有序部分中"正确的"位置。

时间复杂度 $O(n^2)$，额外空间 $O(1)$，稳定。

逆序对个数是 I，那么总共的关键码比较次数就是 $O(n+I)$。因为每次比较都会扫描已知有序序列的所有逆序部分（从后往前扫描）。



选择排序

每次扫描全部列表（或未排序的部分），进行 $O(n)$ 次比较和 $O(1)$ 次交换，找到最大元素并交换到最后位置。

时间复杂度 $O(n^2)$，额外空间 $O(1)$，稳定。

> 运用高级数据结构（堆），$O(n)$ 次的比较可以优化成 $O(\log n)$ (维护一个偏序关系)。

深入分析复杂度——循环节

如果构造一个 $\mathrm{RANK} \to \mathrm{RANK}$ 的映射，把该处**位置**的 rank 映射到该处位置**元素**的 rank，记作 `r` 数组。`r` 是一个置换。

从某个位置出发，不断沿着 `r` 走，我们一定能走回原来出发的位置。这就是“循环节”，离散里面管这个叫轮换。

在选择排序中，每交换一个元素到末尾，我们都会把该元素所在的循环节打断再拼上，也就使得这个循环节的长度 - 1。

无效的交换次数就是循环节的个数。期望 $O(\log n)$ 。

> 但，讨论这玩意有什么用?