向量

Xingdong Xue

Mar. 2020

1 接口与实现

Abstract Data Type：数据模型和定义在数据模型上的一组操作，是一种抽象，与复杂度无关。数据结构：具体编程语言实现的一套 ADT 算法，是具体完整的，复杂度相关的。

1.1 向量

向量是数组的抽象与泛化，有一组元素按线性封装而成，并通过。各个元素与 $[0, n)$ 的**秩**（rank）相对应。

向量的接口列表：

- size(): 返回元素总数
- get(r)：获取秩为 r 的元素
- put(r, e)：用 e 替换秩为 r 的元素
- insert(r, e)：以 r 为秩插入 e 元素，之后的元素按序后移
- remove(r)：删除秩为 r 的元素，返回元素中原来存放的对象
- disordered()：判断所有元素是否非降序排序
- sort()：调整各元素的位置，使之按非降序排序
- find(e)：查找元素 e
- search(e)：查找元素 e，返回不大于 e 且秩最大的元素（有序）

- deduplicate()：剔除重复元素
- uniquify()：剔除重复元素（有序）
- traverse()：遍历

1.2 构造与析构

向量可以通过拷贝数组元素进行构造与析构，具体的拷贝过程分为两个部分。1. 开辟空间：这个空间可以根据所要拷贝的长度进行计算。2. 逐一复制数组元素。

2 可扩充向量

如果向量的存放空间一直不变，会存在问题：

1. 上溢（overflow）：_elem[] 不足以存放所有元素。
2. 下溢(underflow)：_elem[] 中的元素寥寥无几，load factor $\lambda = \_size/\_capacity$ « 50%。

所以需要有方法来动态调整向量的长度，基本的思想为在即将容量即将要上溢的时候，适当地扩大容量。

即在元素满时，重新申请一片两倍与元内存的空间，将原向量的元素逐一拷贝至新向量中，然后释放原向量。

1. 递增式扩容：每当元素满时，追加固定大小的容量。扩容的分摊成本是 O(n)，装填因子岁元素数量增大接近于 100%。
2. 倍增式扩容：每当元素满时，讲向量的容量加倍，扩容的分摊成本是 O(1)，同时确保装填因子 > 50%。

分摊复杂度：通过**连续**实施**足够多**次操作，总体所需成本分摊至单次操作。

3 无序向量

1. 访问：可以通过重载操作符来实现无序向量下标访问（循秩访问）。

2. 插入：先进行扩容判断，然后将指定位置之后的所有元素自后向前（防止元素丢失）地后移一位，再在空出的位置中插入指定元素。

3. 区间删除：从待删除的区间右侧开始，自前向后（防止元素覆盖）地逐一左移动元素，对向量内的元素进行覆盖就完成了删除操作，同时有必要时进行缩容。

4. 单元素删除：区间删除特例。如果从单元素删除出发推导区间删除，则区间删除复杂度为 $O(n^2)$，所以先实现区间删除。

5. 区间查找：前提是元素可以判等，从后向前进行扫描，输出元素的秩，如果查找不到，则输出越过左边界的非法的秩。该算法是输入敏感（input-sensitive）的算法，因为复杂度与输入数据紧密相关。

6. 唯一化：从第一个元素开始，每次对所有前驱元素进行判断，如果发现相同元素则剔除本身，并将之后的所有元素前移；如果未发现，则指针后移。

通过不变性（数学归纳）和单调性（算法可停止）来严格证明算法。

4 有序向量

d 可以通过相邻逆序对的数目来衡量向量的逆序程度。

1. 唯一化

   (a) 算法 1：每次碰到相同元素，则剔除该元素，直到向量遍历完毕，复杂度为 $O(n^2)$。

   (b) 算法 2：记录相同元素的区间，每次发现不重复的元素时，直接追加到原数组左侧，复杂度为 O(n)。

2. 插入：插入要保证相同元素的按插入顺序排列，使用查找来得到要插入的位置，之后进行插入。

查找

为了方便插入，希望查找函数总数能返回查不大于的最后一个元素的秩。

1. 二分查找
2. 斐波那契查找：将分割点设在黄金分割点，而不是等分点，解决左右查找不平衡的问题。
3. 优化二分查找：将分支数从 3 减少到 2，右侧包含哨兵节点，从而解决不平衡问题。
4. 以上三种算法均不能满足返回**不大于 e** 的最后一个元素的约定，需要进行优化。
5. 继续优化二分查找：将哨兵排除在外进行迭代，最后返回左区间的最后一个元素。
6. 插值查找：假设各元素满足一定的随机分布，则 $\frac{mi-lo}{hi-lo} = \frac{e-A[lo]}{A[hi]-A[lo]}$，算法复杂度是 $\log\log n$。

评估

插值查找容易受到局部影响，而且涉及乘除运算，适合大规模查找；中规模采用折半查找；小规模适合顺序查找。

5 排序

5.1 冒泡排序

原算法：逐趟扫描排序，直至全序，复杂度 O(n^2)。改进 1：可能在中途会出现已经全序的情况，可以用逆序对的个数来记录未排序区间是否是已经有序的，从而可能提前结束循环，最好复杂度 O(n)，最坏复杂度 O(n^2)。改进 2：可能在中途会多次出现部分全序的情况，可以记录一下最后一次交换的位置，达到快速缩小未排序区间的目的，最好复杂度 O(n)，最坏复杂度 O(n^2)。

5.2 归并排序

分为划分和归并两个阶段。

在划分阶段，将原数组不断均分为小的子数组直到每个子数组中的元素是有序的（或者元素个数为 1）。

在归并阶段，通过不断选择两个已排序数组的中的最小元素的较小者放入已排序数组，来达到排序的目的。

优化：如果左侧的子数组先归并完毕，则右侧的剩余子数组其实无需进行移动，直接结束循环即可。

merge 的复杂度是 O(n)，mergeSort 复杂度是 $O(\log n)$，所以整体复杂度是 $O(n \log n)$。