

3.列表





(e) 插入排序

花荣便道:"前面必有强人。"

把枪带住,取弓箭来整顿得端正,再插放飞鱼袋内。

- 水浒传·第三十四回

邓俊辉

deng@tsinghua.edu.cn

构思

❖ 始终将序列看成两部分:

Sorted + Unsorted L[0, r) + L[r, n)

❖【初始化】

|S| = r = 0 //空序列无所谓有序

❖【迭代】:关注并处理e = L[r]

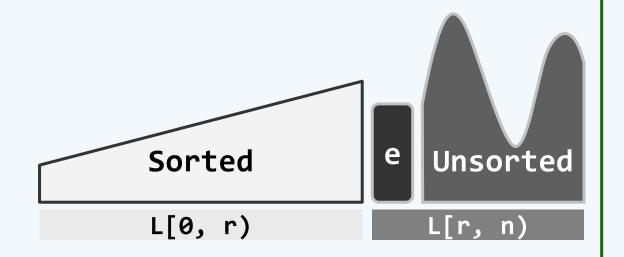
在S中确定 适当位置 //有序序列的查找

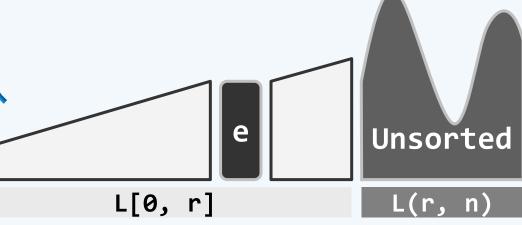
插入e,得到有序的L[0,r]//有序序列的插入

❖ 【不变性】

随着r的递增, L[0, r)始终有序

直到r = n,L即整体有序





实例

迭代轮次	前缀有序子序列	当前元素	后缀无序子序列
-1	^	^	5 2 7 4 6 3 1
0	^	5	2 7 4 6 3 1
1	(5)	2	7 4 6 3 1
2	(2) 5	7	4 6 3 1
3	2 5 (7)	4	6 3 1
4	2 (4) 5 7	6	3 1
5	2 4 5 (6) 7	3	1
6	2 (3) 4 5 6 7	1	^
7	(1) 2 3 4 5 6 7	^	^

实现)

```
//对列表中起始于位置p的连续n个元素做插入排序,valid(p) && rank(p) + n <= size
 template <typename T> void <u>List</u><T>::<u>insertionSort</u>(Posi(T) p, int n) {
    for (int r = 0; r < n; r++) { //逐一引入各节点,由S<sub>r</sub>得到S<sub>r+1</sub>
       <u>insertAfter( search( p->data, r, p ), p->data ); //查找 + 插入</u>
       p = p->succ; <u>remove( p->pred ); //转向下一节点</u>
    } //n次迭代,每次0(r + 1)
 } //仅使用O(1)辅助空间,属于就地算法
```

- ❖ 紧邻于search()接口返回的位置之后插入当前节点,总是保持有序
- ❖ 验证各种情况下的正确性,体会哨兵节点的作用:
 Sr中含有/不含与p相等的元素;Sr中的元素均严格小于/大于p

性能

- ❖最好情况:完全(或几乎)有序每次迭代,只需1次比较, Ø次交换累计Ø(n)时间!
- ❖ 最坏情况:完全(或几乎)逆序第k次迭代,需∅(k)次比较,1次交换累计∅(n²)时间!
- ❖ 一般情况:包含Ⅰ个逆序对

第k次迭代,只需 $O(I_k)$ 次比较,1次交换

累计の(n + I)! //逆序对各需单位时间, input-sensitive, 对Shellsort至关重要

❖ 平均而言呢?

//改用向量呢?稍后分析

//inversion , 不必相邻

//为什么?

//当然,首先需要假定具体的随机分布...

平均性能

❖ 假定: 各元素的取值遵守均匀、独立分布

限化・百儿系的取旧歴リバツ、江エル1

于是: 平均要做多少次元素比较?

❖ 考查: L[r]刚插入完成的那一时刻(穿越?)

试问:此时的有序前缀L[0, r]中,哪个元素是此前的L[r]?

❖观察:其中的r + 1个元素均有可能 , 且概率均等于 1/(r + 1)

❖ 因此,在刚完成的这次迭代中,为引入S[r]所花费时间的数学期望为

$$[r + (r - 1) + ... + 3 + 2 + 1 + 0] / (r + 1) + 1 = r/2 + 1$$

- ❖于是,总体时间的数学期望 = [0+1+...+(n-1)]/2 + 1 = Ø(n²)
- ❖ 再问:在n次迭代中,平均有多少次 无需交换 呢?

L[r, n)

L[0, r]

L(r, n)

若改用向量...

- ❖ 借助二分查找之类的算法, 查找效率 可优化至∂(logk)!
- ❖ 然而 , 总体性能 会否因此相应地提高?

