



数据结构与算法

Data Structures and Algorithms

谢昊

xiehao@cuz.edu.cn

第四章

基本算法 Basic Algorithms

大纲

1. 排序

2. 小结

排序

基本概念

- 已知数据元素序列 $\{a_i\}_{i=0}^n$ ，为每个元素 a_i 均关联一个**关键码 (key)**¹ k_i
- 为关键码定义一种**顺序 (order)**，使得关键码之间可互相比较大小
- 对下标序列 $\{i\}_{i=0}^n$ **重排列**生成新序列 $\{p_i\}_{i=0}^n$
- 若满足 $k_{p_0} \leq k_{p_1} \leq \dots \leq k_{p_n}$ ，则称 $\{a_{p_i}\}_{i=0}^n$ 为**有序序列 (sorted sequence)**
- 称生成有序序列的过程为**排序 (sorting)**

¹简称键

例

- 已知整数序列：21, 25, 22, 10, 25, 18
- 关键码为元素值本身
- 约定顺序为升序
- 排序结果为：10, 18, 21, 22, 25, 25

排序的分类方法

- 根据数据规模与存储特点分：
 - 内部排序：规模较小，内存足矣
 - 外部排序：规模庞大，须借助外存²
- 根据输入形式分：
 - 离线排序：一次性给出全部数据
 - 在线排序：实时生成数据，分批给出
- 根据体系结构分：
 - 串行排序：按顺序依次处理
 - 并行排序：部分同时处理
- 根据算法策略分：
 - 确定排序：不采用随机策略
 - 随机排序：采用随机策略

²如硬盘、分布式设备等

排序的分类方法

- 根据数据规模与存储特点分：
 - 内部排序：规模较小，内存足矣 ✓
 - 外部排序：规模庞大，须借助外存²
- 根据输入形式分：
 - 离线排序：一次性给出全部数据 ✓
 - 在线排序：实时生成数据，分批给出
- 根据体系结构分：
 - 串行排序：按顺序依次处理 ✓
 - 并行排序：部分同时处理
- 根据算法策略分：
 - 确定排序：不采用随机策略 ✓
 - 随机排序：采用随机策略

²如硬盘、分布式设备等

排序的种类

算法	特点
插入类：直接插入 选择类：直接选择、冒泡、堆	顺序比较、简单
快速、归并	分治策略
希尔、桶、基数	其他

排序

插入排序：整理一手乱序扑克牌

- 将序列 s 分为无序与有序两个子序列 $s_u = s$ 与 $s_o = \emptyset$
- 反复从 s_u 中取出元素并插入 s_o 使之依然有序，直至 $s_u = \emptyset$ ，则 s_o 即为所求

算法描述

算法 1 插入排序

```
1: procedure InsertionSort( $s, f, t$ )  
2:    $\triangleright$  对序列  $s$  中下标区间  $[f, t)$  内的元素进行插入排序  
3:    $i \leftarrow f$   
4:   while  $i < t$  do  
5:      $s \leftarrow \text{InsertElement}(s, f, i)$     $\triangleright$  插入元素子算法  
6:      $i \leftarrow i + 1$   
7:   end while  
8:   return  $s$     $\triangleright$  返回排序后的序列  $s$   
9: end procedure
```

实例

- 原序列：()21, 25, 22, 10, 25, 18
- 第一趟：(21), 25, 22, 10, 25, 18
- 第二趟：(21, 25), 22, 10, 25, 18
- 第三趟：(21, 22, 25), 10, 25, 18
- 第四趟：(10, 21, 22, 25), 25, 18
- 第五趟：(10, 21, 22, 25, 25), 18
- 第六趟：(10, 18, 21, 22, 25, 25)

插入元素

- 在有序序列中查找并插入指定元素

可选思路

- 顺序查找³：按前驱后继顺序依次遍历子序列元素，找出目标元素的合理插入位置
- 一些改进：二分/折半查找、Fibonacci 查找、散列查找等

存储方式

- 连续存储：须移动大量元素
- 链式存储：只需移动局部元素，较方便

³称对应算法为直接插入排序

排序算法的稳定性

- 若键相同的元素在排序前后的相对顺序保持不变，则称该排序算法为 **稳定的**
- 稳定排序算法适用于 **多键排序**⁴

⁴先按主键排序，在主键重复的序列中按次键排序，以此类推。

直接插入排序算法性能分析

- 时间复杂度： $O(n^2)$
 - 平均比较次数： $\sum_{k=2}^n \frac{k}{2} = \frac{(n+2)(n-1)}{4}$
- 空间复杂度： $O(n)$
 - 须额外一个元素的空间缓存待交换元素
- 稳定性
 - 在插入元素时若遇键相同的情况可小心处理顺序，故该算法 **稳定**

排序

选择排序：择优录取

- 将序列 s 分为**无序**与**有序**两个子序列 $s_u = s$ 与 $s_o = \emptyset$
- 反复从 s_u 中取出键最小的元素并放至 s_o 末尾，直至 $s_u = \emptyset$ ，则 s_o 即为所求

算法描述

算法 2 选择排序

```
1: procedure SelectionSort( $s, f, t$ )  
2:   ▶ 对序列  $s$  中下标区间  $[f, t)$  内的元素进行选择排序  
3:    $i \leftarrow f$   
4:   while  $i < t$  do  
5:      $k \leftarrow \text{SelectElement}(s, i, t)$    ▶ 选择元素子算法  
6:     Swap( $s_i, s_k$ )                       ▶ 交换元素  
7:      $i \leftarrow i + 1$   
8:   end while  
9:   return  $s$                                ▶ 返回排序后的序列  $s$   
10: end procedure
```

实例

- 原序列: ()21, 25, 22, 10, 25, 18
- 第一趟: (10), 25, 22, 21, 25, 18
- 第二趟: (10, 18), 22, 21, 25, 25
- 第三趟: (10, 18, 21), 22, 25, 25
- 第四趟: (10, 18, 21, 22), 25, 25
- 第五趟: (10, 18, 21, 22, 25), 25
- 第六趟: (10, 18, 21, 22, 25, 25)

排序

选择元素

- 在无序序列中查找键最小元素

可选思路

- 冒泡排序：通过逐个交换相邻元素的方式选出键最小元素
- 直接选择排序：按前驱后继顺序依次遍历子序列元素，找出键最小元素
- 堆排序：采用优先队列组织无序子序列

直接选择排序的性能分析

- 复杂度与直接插入排序类似，略
- 稳定性
 - 因涉及到跨元素交换，故该算法不稳定

排序

快速排序：分治策略的应用

- 将序列**划分**为三部分：任意元素 s_m 作为**支点**、所有元素分别比 s_m 小与大的两个子序列
- 对后两个子序列递归执行上述**划分**策略，直至所有子序列为空

算法描述

算法 3 快速排序

```
1: procedure QuickSort( $s, l, h$ )
2:   ▶ 对序列  $s$  中下标区间  $[l, h]$  内的元素进行选择排序
3:   while  $l < h$  do
4:      $m \leftarrow \text{Partition}(s, l, h)$            ▶ 序列划分子算法
5:     QuickSort( $s, l, m - 1$ )                  ▶ 左侧递归
6:     QuickSort( $s, m + 1, h$ )                  ▶ 右侧递归
7:   end while
8:   return  $s$                                 ▶ 返回排序后的序列  $s$ 
9: end procedure
```

实例

- 原序列：21, 25, 22, 10, 25, 18
- 第一趟：18, 10, 21, 22, 25, 25
- 第二趟：10, 18, 21, 22, 25, 25
- 第三趟：10, 18, 21, 22, 25, 25
- 第四趟：10, 18, 21, 22, 25, 25

划分算法

```
1 int partition(DataType *s, int l, int h,  
2         CompareType c) {  
3     DataType pivot = s[l]; // 缓存支点元素  
4     for (;;) {  
5         for (; l < h && c(&pivot, s + h) < 0; --h)  
6             ; // 左移 h 指针至须交换处  
7         if (l >= h) break;  
8         s[l++] = s[h]; // 交换元素并右移 l 指针  
9         for (; l < h && c(s + l, &pivot) < 0; ++l)  
10            ; // 右移 l 指针至须交换处  
11        if (l >= h) break;  
12        s[h--] = s[l]; // 交换元素并左移 h 指针  
13    } // 此时 l 与 h 二指针重合, 结束循环  
14    s[l] = pivot; // 放回支点元素  
15    return l; // 返回支点元素新下标  
16 }
```

实例：一次划分

- 原始序列: $21_l, 25, 22, 10, 25, 18_h, (\quad)$
- 缓存支点: $\quad_l, 25, 22, 10, 25, 18_h, (21_m)$
- 交换元素: $18_l, 25, 22, 10, 25, \quad_h, (21_m)$
- 移动指针: $18, 25_l, 22, 10, 25, \quad_h, (21_m)$
- 交换元素: $18, \quad_l, 22, 10, 25, 25_h, (21_m)$
- 移动指针: $18, \quad_l, 22, 10, 25_h, 25, (21_m)$
- 移动指针: $18, \quad_l, 22, 10_h, 25, 25, (21_m)$
- 交换元素: $18, 10_l, 22, \quad_h, 25, 25, (21_m)$
- 移动指针: $18, 10, 22_l, \quad_h, 25, 25, (21_m)$
- 交换元素: $18, 10, \quad_l, 22_h, 25, 25, (21_m)$
- 移动指针: $18, 10, \quad_{lh}, 22, 25, 25, (21_m)$
- 放回支点: $18, 10, 21_m, 22, 25, 25, (\quad)$

快速排序的性能分析

- 时间复杂度
 - 最好情况: $O(n \log n)$
 - 最坏情况: $O(n^2)$
 - 平均情况: $O(n \log n)$
- 空间复杂度: $O(n)$
- 稳定性
 - 由于交换过程会越过支点元素且无法控制, 故该算法 **不稳定**

各种排序算法的取舍

- 问题规模较小⁵：直接选择排序、直接插入排序等
- 问题规模略大：快速排序、归并排序、堆排序等
- 当需要多键排序时，宜选用稳定算法：直接插入排序、堆排序等

⁵一般认为待排序序列元素不超过 23 个

未完待续...

小结

-

问与答