期中复习

时间复杂度、排序、分治

GKxx

October 31, 2022

时间复杂度

排序

分治

渐近 (asymptotic) 记号

- 假设我们的算法的运行时间是关于问题规模 n 的函数 $T(n)$。
- 精确地写出 $T(n)$（是多少毫秒/多少条指令/...）是不可能的。我们关心的是 $T(n)$ 的量级。
- Θ, O, Ω
- $n = \Theta(n) = O\left(n^2\right)$.
- **等号的“活用”**：右边是左边的“粗胚”（《具体数学》Ch. 9）
- $O(\log n)$ 的 $\log$ 是几底？

渐近记号

- O 和 Θ 的混用非常常见，但我们一开始学的时候还是要分清楚。
- 答题时需要写紧确的界！

时间复杂度

排序

分治

排序算法

我们学过的排序算法：

- 插入排序 Insertion-sort
- 冒泡排序 Bubble-sort
- 归并排序 Merge-sort
- 快速排序 Quick-sort
- 堆排序 Heap-sort

排序算法的衡量：

- 时间复杂度、（额外）空间复杂度
- 是否基于比较 (comparison-based)
  - $\Omega(n \log n)$
- 是否稳定 (stable)

插入排序

- 维护一段有序前缀，每次把下一个元素插入到有序区中的正确位置。
- $\Theta\left(n^2\right)$，但是 $\Theta(n+d)$。
  - 逆序对 (inversions): Pair (i, j) such that $i < j$ and $A_i > A_j$.
- 稳定

冒泡排序

- 通过交换相邻逆序对的方法，每次把大的元素往右冒。
- $\Theta\left(n^2\right)$
- 有各种改进：
  - Flagged：如果当前这一轮迭代没有交换元素，说明已经有序。仍然是 $\Theta\left(n^2\right)$。
  - Alternating：在奇数轮把大的往右冒，偶数轮把小的往左沉。
  - Range-limiting：每一轮只做到上一轮的最后一次 swap 发生的位置。
- 叠满 buff：$\Theta(n+d)$（最坏仍然是 $\Theta\left(n^2\right)$），但仍然比插入排序慢。
- “没什么用，只是有一个吸引人的名字，并且能引出一些有趣的理论问题。” – Donald E. Knuth

快速排序

- 每次选一个主元 (pivot) x，将序列围绕 x 划分 (partition)，使得小于 x 的在左边，大于 x 的在右边，这时 x 已经在正确的位置上。对两边分别递归。
- 划分：时间 $\Theta(n)$，额外空间 $O(1)$，不稳定。
  - 当然有其它 fancy 的实现，但要么是牺牲了其中的某些优点，要么是过于复杂。
- 假如 x 是第 q 小值，则 $T_q(n) = T(q-1) + T(n-q) + \Theta(n)$.
  - 最好 $\Theta(n \log n)$，平均 $\Theta(n \log n)$
  - 最坏 $\Theta\left(n^2\right)$（何时发生？）
- 主元的选择至关重要：
  - 直接选第一个：$\langle 1, 2, \cdots, n \rangle$ 就暴毙
  - 选中间
  - Median-of-3
  - 随机选

快速排序

- 主元的选择至关重要：
  - Median-of-3
  - 随机选
  - 《算法导论》上的 Median-of-3：在随机选的三个元素中选 median
- 无论怎么选都只能降低碰到 worst case 的概率，而不会改变 worst case $\Theta\left(n^2\right)$ 的复杂度
  - 除非你能在 $O(n)$ 的时间里选到中位数...
  - 《算法导论》Ch. 7 Chapter notes: “Killer adversary”

快速排序

- 主元的选择至关重要：
  - Median-of-3
  - 随机选
  - 《算法导论》上的 Median-of-3：在随机选的三个元素中选 median
- 无论怎么选都只能降低碰到 worst case 的概率，而不会改变 worst case $\Theta\left(n^2\right)$ 的复杂度
  - 除非你能在 $O(n)$ 的时间里选到中位数...
  - 《算法导论》Ch. 7 Chapter notes: “Killer adversary”
- 空间复杂度：看起来不占额外空间（$O(1)$），但递归...
  - 朴素实现：最坏 $\Theta(n)$，最好/平均 $\Theta(\log n)$。
  - 尾递归 (tail recursion) 优化：$O(\log n)$。
- 不稳定。

快速选择（作业题）

利用 partition 实现：找出序列中的第 k 小值。

- 假如主元是第 q 小值，那么划分之后它就是 A_q。
- 最小的 $q-1$ 个元素都在左边，最大的 $n-q$ 个元素都在右边。
- 根据 k 和 q 的大小关系，选择进哪一边递归。

时间复杂度：

- 最好 $\Theta(n)$
- 最坏 $\Theta\left(n^2\right)$：假如 $k=n$ 而每次主元选的都是最小值…

堆排序

利用堆 (heap) 的性质实现的排序

- 时间 $\Theta(n \log n)$，额外空间 $O(1)$，不稳定。
- 讲堆的时候再讲...

不嫌麻烦的话，AVL/RB-tree 也可以用来排序，也是 $O(n \log n)$ 的...

归并排序

- 将序列从中间切开分成两段。先把左右两段分别递归地排好序，再花 $O(n)$ 的时间合并成一段有序的序列。
- 时间 $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$。
- 空间？
  - 在数组上，常见的 $\Theta(n)$ merge 实现需要 $\Theta(n)$ 额外空间。
  - 在链表上 merge 不需要额外空间。
  - 递归需要 $\Theta(\log n)$ 的栈空间。

  在数组上 $\Theta(n)$，在链表上 $\Theta(\log n)$。
- 稳定。

鱼和熊掌

- 归并排序：时间 $\Theta(n \log n)$，稳定，需要 $\Theta(n)$ 额外空间（数组）。
- 快速排序：平均时间 $\Theta(n \log n)$，不稳定，需要 $O(\log n)$ 额外空间（尾递归优化）。
- 堆排序：时间 $\Theta(n \log n)$，不稳定，需要 $O(1)$ 额外空间。

有没有时间 $O(n \log n)$、稳定、额外空间 $O(1)$ 的排序？

鱼和熊掌

- 归并排序：时间 $\Theta(n \log n)$，稳定，需要 $\Theta(n)$ 额外空间（数组）。
- 快速排序：平均时间 $\Theta(n \log n)$，不稳定，需要 $O(\log n)$ 额外空间（尾递归优化）。
- 堆排序：时间 $\Theta(n \log n)$，不稳定，需要 $O(1)$ 额外空间。

有没有时间 $O(n \log n)$、稳定、额外空间 $O(1)$ 的排序？

- 可能有（STFW），但不在 CS101 范围内。

“最好”的排序

如果输入的 n 个数都是 $[1, n]$ 中的整数，有没有办法做得更好？

“最好”的排序

如果输入的 n 个数都是 $[1, n]$ 中的整数，有没有办法做得更好？

- 最适合你的问题特征的就是“最好”的排序。
- 标准库总是提供一个“相当不错”的排序：
  - C++11 起`std::sort`保证 $\Theta(n \log n)$。在 GCC 实现品中，它采用一种混合了快速排序、插入排序和堆排序的算法，称为“intro-sort”。

不要尝试挑战标准库，除非你面对的问题确实有特殊性。

时间复杂度

排序

分治

逆序对计数

考虑将序列 $\langle A_l, \cdots, A_r \rangle$ 从中间切开，分成 $\langle A_l, \cdots, A_m \rangle$ 和 $\langle A_{m+1}, \cdots, A_r \rangle$ 两段。一个逆序对 (i, j) 无外乎以下三种情况：

- $i, j \leqslant m$（两个位置都在左边）
- $i, j > m$（两个位置都在右边）
- $i \leqslant m < j$（一个在左边，一个在右边）

对于前两种情况，我们只要对左右两段分别递归求解。

逆序对计数

只需考虑 $i \leqslant m < j$ 的情况：

- $i < j$ 已经得到保证，但似乎还是不好处理？
- 画个饼：假设 $\langle A_l, \cdots, A_m \rangle$ 和 $\langle A_{m+1}, \cdots, A_r \rangle$ 都排好序了。
- 枚举 $i = l, \cdots, m$，在右边能与 A_i 形成逆序对的元素一定是一段前缀。
- 记 $f(i) = \max\{j \mid m < j \leqslant r, A_i > A_j\}$，则右边能与 A_i 形成逆序对的元素有 $f(i) - m$ 个。
- 由于两边都有序，显然 $f(i) \leqslant f(i+1)$。

```
auto j = m;
for (auto i = l; i <= m; ++i) {
  while (j + 1 <= r && a[i] > a[j + 1])
    ++j;
  if (a[i] <= a[j])
    break;
  // Now j == f(i).
  inversions_cnt += j - m;
}
```

逆序对计数

只需考虑 $i \leqslant m < j$ 的情况：

- $i < j$ 已经得到保证，但似乎还是不好处理？
- 画个饼：假设 $\langle A_l, \cdots, A_m \rangle$ 和 $\langle A_{m+1}, \cdots, A_r \rangle$ 都排好序了。
- 枚举 $i = l, \cdots, m$，在右边能与 A_i 形成逆序对的元素一定是一段前缀。
- 记 $f(i) = \max\{j \mid m < j \leqslant r, A_i > A_j\}$，则右边能与 A_i 形成逆序对的元素有 $f(i) - m$ 个。
- 由于两边都有序，显然 $f(i) \leqslant f(i+1)$。用一个变量 j 单向移动来获得 $f(i)$。
- **不要忘了自己画的饼：**统计完之后做一次 merge，把序列合并成一个有序的。

逆序对计数

- 问题比较简单，可以一边 merge 一边统计，这也就是我们常说的“用归并排序数逆序对”的方法。
- 时间复杂度 $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$。

Strassen 的矩阵乘法

矩阵分块：

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C = AB = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22}. \end{bmatrix}$$

- 只要上式中的所有乘法都能乘，这个划分就是可以的，不是非得平均划分。

Strassen 的矩阵乘法

Strassen 发现：

- 可以先通过矩阵加减法算出 10 个中间结果 $S_1, \cdots, S_{10}$，
- 然后用这些中间结果进行 7 次矩阵乘法算出另外 7 个结果 $P_1, \cdots, P_7$，
- 最后通过这 7 个矩阵的若干次加减法得出 C_{11}, C_{12}, C_{21} 和 C_{21}。

假如 $A, B \in \mathbb{R}^{n\times n}$，假设 $n = 2^m$，我们将 A 和 B 都均等地划分成 $n/2 \times n/2$ 的四份，则我们可以通过十几次 $\Theta\left(n^2\right)$ 的加减法和 7 次 $n/2 \times n/2$ 的矩阵乘法算出结果。

$$T(n) = 7T(n/2) + \Theta\left(n^2\right).$$

分治与主定理

假设我们把一个规模为 n 的问题分成了 a 个子问题，每个子问题的规模为 n/b（可能带有上/下取整），并且分解和合并的过程需要花 $f(n)$ 的时间，则

$$T(n) = aT(n/b) + f(n).$$

- 我们总是假定 $T(1) = \Theta(1)$。事实上这句话不说也行，因为对于任意常数 c 来说，$T(c)$ 都是常数，常数就是 $\Theta(1)$。

主定理

$$T(n) = aT(n/b) + f(n).$$

我们课件上的主定理：假定 $f(n) = \Theta\left(n^d\right)$。

$$T(n) = \begin{cases} \Theta\left(n^d\right), & d > \log_b a \\ \Theta\left(n^d \log n\right), & d = \log_b a \\ \Theta\left(n^{\log_b a}\right), & d < \log_b a. \end{cases}$$

课件上写的是 O，但其实这些界是紧的，用的时候写 Θ 也对。

最好背一下！

主定理

递归式的求解可能是很难的。

- 《算法导论》上有主定理的完整版，可以解决一般的 $f(n)$ 的情形。
- $T(n) = T(n/4) + T(3n/4) + f(n)$ 怎么办？画递归树仍然能解决。
- $T(n) = \sum_{i=1}^{k} a_i T(n/b_i) + f(n)$ 怎么办？《算法导论》Ch. 4 Chapter notes.