Tutorial 2

排序的深入理解

蛮力排序

对于选择排序和插入排序，虽然前者和后者在 Big Oh 的意义上没有区别，但是考虑常系数的话，后者相较于前者有一倍的性能提升

改进版的冒泡排序：记录最后一次交换的位置，之后便不再比较。

同样是常系数减小了但是以 Big Oh 考虑的话没有改进

以消除 inversion 的视角来看消除 inversion 的效率没有提升，只是减少了无用的比较

QuickSort 的基于指标随机变量的分析

对于输入序列 $\{x_1, x_2, x_3, \dots, x_n\}$，定义指标随机变量

$$X_{ij} = \{x_i与x_j发生比较\}$$

则快速排序的平均情况时间复杂度为

$$A(n) = E[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E[X_{ij}]$$

由于输入的随机性，很难统计 $E[X_{ij}]$，**故重新定义指标随机变量**，设输入的元素按照从小到大的顺序记为 $\{z_1, z_2, z_3, \dots z_n\}$，则定义指标随机变量

$$X_{ij} = \{z_i与z_j发生比较\}$$

平均情况时间复杂度同样为

$$A(n) = E[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E[X_{ij}]$$

因为即使改变了标记，仍然是统计任意两元素间发生比较，没有多没有少。根据指标随机变量的定义

$$A(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i\text{与}z_j\text{发生比较}\}$$

为了计算概率，将元素划分为 5 个不同的部分，分别考察各部分的元素被选为 pivot 时，z_i, z_j 是否发生比较

$$\{z_1, z_2, \dots, z_{i-1}\}, \{z_i\}, \{z_{i+1}, z_{i+2} \dots, z_{j-1}\}, \{z_j\}, \{z_{j+1}, z_{j+2}, \dots, z_n\}$$

- $\{z_k \mid 1 \leqslant k \leqslant i-1\}$ 和 $\{z_k \mid j+1 \leqslant k \leqslant n\}$ ，若其中元素被选为 pivot，则 z_i, z_j 被划分到同一部分，之间是否发生比较根据后续 pivot 的选择而定，故对结果无影响
- $\{z_k \mid i+1 \leqslant k \leqslant j-1\}$ ，若其中元素被选为 pivot，则 z_i, z_j 被划分到两个部分，永远不会发生比较
- $\{z_i\}$ 和 $\{z_j\}$ 若 z_i 或 z_j 被选为 pivot，会与对方发生一次比较

设所有输入序列等可能出现，则每个元素被选为 pivot 的概率相等，故

$$\Pr\{z_i\text{与}z_j\text{发生比较}\} = \frac{2}{j-i+1}$$

则可得快速排序的平均情况时间复杂度为

$$\begin{aligned}
A(n) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i\text{与}z_j\text{发生比较}\} \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\log n) \\
&= O(n \log n)
\end{aligned}$$

堆结构

堆中第 k 大元素

要求算法复杂度为 $f(k)$

原理：第 k 大的元素只会出现在前 k 层（由堆的偏序特性易证）

即修复堆时仅修复前 k 层，算法代价为 $O(k \cdot \log 2^k) = O(k^2)$ （k 次修复）

Sum of Heights

堆中结点高度和最多为 $n-1$，其中 n 为堆中结点个数

思路：数学归纳法，但要分情况讨论

- 易得完美二叉树的结点高度和可直接求出
- 对于非完美二叉树的堆，其左右子树必有一颗为完美二叉树，对不完美的一颗子树应用归纳假设，对完美的子树直接使用精确的结点高度和
- 还有一种情况：左右子树为高度差 1 的完美二叉树，单独讨论即可

anagram 问题

给一本字典，找到其中所有的 anagram（字母相同但排列不同，如 ate，eat）

结论：排序问题和找相同问题是等价的

思路：

- 首先按照长度排序，显然长度不同的单词不可能是 anagram
- 然后对每个单词将其字母按照字典序排序，排序结果作为该单词的 tag
- 对于所有长度相同的单词，按照 tag 排序，显然 anagram 的 tag 一定相同

从排序到分治

常见项问题

寻找 n 个元素中出现次数超过 $\left\lceil \frac{n}{k} \right\rceil$ 的元素

思路：将元素平均划分，整体常见项一定至少是其中一部分的局部常见项

对于递归结果各自扫描，代价为 $O(n)$ ，即 $T(n) = 2T(n/2) + O(n)$ ，时间复杂度为 $O(n \log k)$，下界为 $O(n \log n)$

Maxima 问题

排序解法思路：按照横坐标排序后，由右至左扫描，大于右边纵坐标最大值 y_{max} 的一定是 maxima，时间复杂度 $O(n \log n)$

分治解法思路：分为两部分，递归求解，右边部分的一定是 maxima，只用处理左边部分。处理代价为 $O(n)$ ，时间复杂度 $O(n \log n)$

maxima 问题的下界：maxima 问题可规约至常见项问题，而常见项问题的下界为 $O(n \log n)$

整数相乘问题

假设 $x \times y$，其中 x 和 y 的二进制表示均为 n 位，则令

$$\begin{gathered} x = x_1 \times 2^{n/2} + x_0 \\ y = y_1 \times 2^{n/2} + y_0 \\ x \times y = x_1 y_1 \times 2^n + x_0 y_1 \times 2^{n/2} + x_1 y_0 \times 2^{n/2} + x_0 y_0 \end{gathered}$$

而

$$\begin{gathered} P = (x_1 + x_0)(y_1 + y_0) = x_1 y_1 + x_0 y_1 + x_1 y_0 + x_0 y_0 \\ x_0 y_1 + x_1 y_0 = P - x_1 y_1 - x_0 y_0 \end{gathered}$$

即

$$x \times y = x_1 y_1 \times 2^n + (P - x_1 y_1 - x_0 y_0) \times 2^{n/2} + x_0 y_0$$

只用计算三次乘法

$$\begin{gathered} T(n) = 3T(n/2) + O(n) \\ T(n) = o(n^2) \end{gathered}$$

矩阵相乘问题

同理，将矩阵划分为 4 个部分

$$\begin{gathered} A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \quad B = \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix} \\ A \times B = \begin{bmatrix} A_1 B_1 + A_2 B_3 & A_1 B_2 + A_2 B_4 \\ A_3 B_1 + A_4 B_3 & A_3 B_2 + A_4 B_4 \end{bmatrix} \end{gathered}$$

可减少一次乘法运算使得

$$\begin{gathered} T(n) = 7T(n/2) + O(n^2) \\ T(n) = o(n^3) \end{gathered}$$

[Strassen 算法]