求解最长上升子序列的快速算法

Shuxin Chen, 2015013229

2017 年 3 月 10 日

1 问题简介

1.1 概述

最长上升子序列，又称最长单调递增子序列（**L**ongest **I**ncreasing **S**ubsequence，简称 **LIS**），是一种经典的动态规划问题。这里的**单调递增**指的是严格的递增，即 $\forall i, j$ 若 $i < j$ 则 $a[i] < a[j]$。本文从一种比较容易想到的，时间复杂度为 $\Theta(n^2)$ 的算法开始，逐步挖掘状态之间内在的联系，得出两种不同的时间复杂度为 $\Theta(n \log n)$ 的算法。

1.2 变量定义

- n : 我们考虑的问题的规模为 n，即任意给定 n 个正整数（因为最长上升子序列考虑的是数之间的相对大小，所以不妨规定它们均为正整数），组成一个长度为 n 的序列，我们要给出这个序列的最长上升子序列。

- $a[]$: 数组 a 表示这个长度为 n 的序列。

- $f[]$: 数组 f 表示每一个 a 中的元素对应的 LIS 的长度。

- $parent[]$: 数组 $parent$ 表示前驱状态，即 $parent[i]$ 表示以 $a[i]$ 结尾的 LIS，它的倒数第二项的位置编号。若可能的以 $a[i]$ 结尾的 LIS 有多个，那么 $parent[i]$ 的取值也会有多个。这里我们规定，可以取任意一个值，这样影响的只是最后求出的答案（因为可能有多个 LIS），而不会影响 LIS 的长度。

2 朴素的动态规划算法

假设当前已经计算过了前 $i-1$ 个数的 LIS，在数组中也存放了相应的值。此时，对于 $a[i]$ 这个数，我们可以遍历它之前的每一个数 $a[j]$，如果 $a[j] < a[i]$，那么 $a[j]$ 就可以作为 $a[i]$ 的前一项。在所有可行的 j 中取出 $f[j]$ 最大的那一个 j_0，那么以 $a[i]$ 结尾的 LIS 的前一项即为 $a[j_0]$。若没有可行的 j，则规定 $j_0 = -1$。因此我们有如下的状态转移方程：

$$f[i] = \max_{\substack{1 \le j < i \\ a[j] < a[i]}} \{f[j]\} + 1$$

对于 $parent$ 数组的计算，根据前面 j_0 的定义，我们有

$$parent[i] = j_0$$

整个算法的时间复杂度为 $\Theta(n^2)$。

3 离散化优化

在 1.2 节中我们提到过，我们考虑的是 n 个正整数之间的相对大小，这 n 个数具体的值并不重要。

设集合 $S = \{x | 1 \le x \le n, x \in \mathbb{N}^+\}$，我们构造一个单射 $f : \mathbb{N}^+ \to S$，使得映射后的值仍然保持相对大小不变，但是它们的绝对大小被控制在 S 中。比较简单的一种映射方法就是，将数组 a 中最小的元素（们）映射到 1，次小的元素（们）映射到 2，以此类推。

当这 n 个数的大小都被控制在有限的范围内时，我们定义一个新的数组 $g[]$，这个数组将会大幅度优化算法的时间复杂度。

- $g[]$：数组 g 以另一种方式存储了 a 中的元素对应的 LIS 长度，即 $g[i]$ 表示以元素 i 结尾的 LIS 的长度（在 a 中可能有多个元素，它们的值都是 i，但下标最大的那个元素的 LIS 一定不劣于其余元素，$g[i]$ 存储的即为这个下标最大的元素的 LIS 的长度）。

根据数组 g 的定义，我们可以改写状态转移方程：

$$f[i] = \max_{1 \le j < a[i]} \{g[j]\} + 1$$

$$g[a[i]] = \max\{g[a[i]], f[i]\}$$

根据状态转移方程，我们可以把问题抽象成以下的模型：

- 给定一个长度为 n 的数组，初始每个元素均为 0，需要支持两种操作。

- 操作 1，求出前缀最大值。
- 操作 2，将某一个元素修改成不小于当前值的某一个值。

这是一个经典的线段树（segment tree）或树状数组（Fenwick tree）问题，时间复杂度为 $\Theta(n\log n)$。尤其是树状数组，常数小，代码短，易编写。但这两种数据结构我们并没有在课堂中学过，那么有没有一种只使用最基础的数据结构——数组和简单的辅助算法就能在 $\Theta(n\log n)$ 时间内完成动态规划的算法呢?

4 二分查找优化

我们定义一个新的数组 $h[]$，这个数组存储了当前不同长度的上升子序列 (**I**ncreasing **S**ubsequence，简称 **IS**)。

- $h[]$: 将当前所有长度为 k 的 IS 中，选出那个结尾元素最小的 IS，并将这个结尾元素存放在 $h[k]$ 中。

根据数组 h 的定义，我们可以得到以下定理:

定理 4.1. *设当前已经计算过了前 $i-1$ 个数的 LIS，此时 LIS 的最长长度不会超过 $i-1$，不妨设其为 k_{max}。那么数组 h 中只有 $h[1]$ 到 $h[k_{max}]$ 有值，其余元素均为 0。此时，$\forall 1 \le k_0 < k_{max}$，有 $h[k_0] < h[k_0+1]$。*

证明. 使用反证法。假设 $\exists 1 \le k_0 < k_{max}$，有 $h[k_0] \ge h[k_0+1]$。那么取出长度为 k_0+1，结尾元素为 $h[k_0+1]$ 的 IS，设其为 $elem_1, elem_2, \dots, elem_{k_0+1}$。那么它的前 k_0 项也是一个 IS，且有 $elem_{k_0} < elem_{k_0+1}$。但根据数组 h 的定义，有 $elem_{k_0} \ge h[k_0] \ge h[k_0+1] = elem_{k_0+1}$，矛盾。因此**定理 4.1** 成立。

定理 4.1 告诉我们，数组 h 是严格单调递增的。假设当前已经计算过了前 $i-1$ 个数的数组 f 和 h，并补充 $h[0] = -\infty$，我们改写状态转移方程:

$$f[i] = \max_{\substack{0 \le k \le k_{max} \\ a[i] > h[k]}} \{k\} + 1$$

由于数组 h 的单调性，我们可以在数组 h 中二分查找最大的 k 值，这样每一个 $f[i]$ 就可以在 $\Theta(\log n)$ 时间内求出。

求出了 $f[i]$ 后，我们要对数组 h 进行更新。此时会分成两种情况。

1. 若 $k = k_{max}$，那么以 $a[i]$ 结尾的 LIS 的长度超过了之前的最大长度，因此我们需要将 k_{max} 的值增加 1，并把 $a[i]$ 记录到 $h[k_{max}]$ 中。

2. 若 $k < k_{max}$，那么以 $a[i]$ 结尾的 LIS 的长度为 $k+1$，我们将 $a[i]$ 的值与 $h[k+1]$ 进行比较，并将两者中较小的值赋给 $h[k+1]$。

无论是上面的哪一种情况，时间复杂度均为 $\Theta(1)$。数组 f 和 h 已经可以得出 LIS 的长度，我们还需要求出数组 $parent$ 来得到具体的 LIS，也会分成两种情况。

1. 若 $k = 0$, 那么 $parent[i] = 0$，表示以 $a[i]$ 结尾的 LIS 只有一个元素即为本身，它没有前驱元素。

2. 若 $k > 0$，那么 $a[i]$ 的前驱元素应该为二分查找位置的前一个元素，即 $parent[i] = id(h[k-1])$，$id(elem)$ 表示取元素 elem 在数组 a 中的位置，这个值可以在修改数组 h 时同时记录，时间复杂度为 $\Theta(1)$。

它的时间复杂度同样是 $\Theta(1)$，因此对于每一个 i 求出上面三个数组的时间为 $\Theta(\log n)$，总时间复杂度即为 $\Theta(n \log n)$。

5 测试

我们在如下环境中测试第 4 节中的优化算法。

- 操作系统：macOS Sierra 10.12.3
- 处理器：2.5 GHz Intel Core i7
- 内存：16GB 1600MHz DDR3
- 编辑器：Xcode 8.0 8A218a (Release)
- 编译环境：clang-800.0.38

测试分为两种，对于固定的 n，第一种是对随机生成的数组 a 计算出 LIS，称为 norm 测试；第二种是对最坏情况下的数组 a（即元素单调递增）计算出 LIS，成为 extra 测试。测试的项目为算法的纯运行时间，不包括创建数组和输出输出的时间，记录在下面的表格中。

n	norm	extra	extra/norm
1000	0.000s	0.000s	/
10000	0.000s	0.000s	/
100000	0.005s	0.009s	1.800
1000000	0.058s	0.115s	1.983
10000000	0.752s	1.430s	1.902
100000000	10.244s	19.663s	1.919
200000000	23.294s	45.967s	1.973

6 总结

以前都没有仔细证明过 $\Theta(n \log n)$ 算法的正确性。这次自己证出来了，很高兴。