最长单调递增子序列 实验报告

吴佳龙 2018013418

摘要

本次实验对计算最长递增子序列的 $O(n \lg n)$ 算法进行了理论推导、分析与程序设计，通过与 $\Theta(n^2)$ 算法进行比较，验证了算法实现的结果正确性，以及该算法计算 LIS 的高效率。另外，还实现了一个简单的图形界面程序方便测试。

1 问题

设计一个 $O(n \lg n)$ 时间的算法，求一个 n 个数的序列的最长单调递增子序列（Longest Increasing Subsequence, LIS）。

2 实验环境

操作系统：Windows 10

IDE：Visual Studio 2017

处理器：3.1 GHz 双核 Intel Core i5

图形界面开发环境：Qt 5.12.4 with Qt Creator 4.9.1

3 算法分析：$O(n \lg n)$ 的 LIS 求解算法

3.1 $\Theta(n^2)$ 的算法

存在一个自然的动态规划算法，对于序列 $a_1, \cdots, a_n$，定义状态 f_i 表示：**以 a_i 结尾的 LIS 的长度**，则状态转移方程为

$$f_i = \max_{j<i \wedge a_j<a_i} \{f_j + 1\}$$

最终，这个序列的 LIS 的长度为

$$\max_{1 \le i \le n} f_i$$

该算法的朴素实现的时间复杂度为 $\Theta(n^2)$，且似乎很难继续优化。

3.2 算法原理

考虑以上 $\Theta(n^2)$ 算法的状态设计的对偶，定义 c_{ij} 表示：**$a_1 \cdots a_i$ 的所有长度为 j 的递增子序列中最小末尾值**，其中 $i \in \{1, 2, \cdots, n\}, j \in \mathbb{Z}^+$。若不存在长度为 j 的递增子序列则定义 $c_{ij} = \infty$。为了表述方便，定义 $c_{0j} = \infty$ 。

我们有以下两个关于 c 的性质：

Property 1. $\forall i = 1, 2, \cdots, n$，$c_{ij}$ 关于 j 严格单调递增。

Proof. 固定 i，若存在 $c_{i,j} \ge c_{i,j+1}$，则存在以 $c_{i,j+1}$ 结尾的长度为 $j+1$ 的递增子序列，该子序列的倒数第二个元素 $< c_{i,j+1} \le c_{i,j}$，这与 $c_{i,j}$ 是长度为 j 的递增子序列的最小末尾值矛盾。 □

Property 2. $\forall i = 1, 2, \cdots, n$，$c_{i-1}$ 与 c_i 至多只有一个值不相同。

Proof. 令 p 表示 $c_{i-1,p}$ 是 c_{i-1} 中第一个 $\ge a_i$ 的位置，分类讨论如下：

- 对于 $j < p$，$c_{i-1,j} = c_{i,j}$，这是因为 $c_{i-1,j} < a_i$，a_i 不能产生末尾值更小的长度 $< p$ 的递增子序列。
- $c_{i,p} = \min\{c_{i-1,p}, a_i\}$，这是因为 a_i 可以与 $c_{i,p-1}$ 结尾的子序列构成新的长为 p 的子序列。
- 对于 $j > p$，$c_{i-1,j} = c_{i,j}$，这是因为若 $c_{i,j} \ne c_{i-1,j}$ 则必有 $c_{i,j} = a_i$，这等价于存在一个长为 j 的以 a_i 结尾的递增子序列，

等价于存在一个长为 $j-1$ 的以 $< a_i$ 结尾的子序列，这与 $c_{i-1,j-1} \geq c_{i-1,p} \geq a_i$ 矛盾。

□

因此我们可以从 c_0 出发，每次最多修改一个值从 c_i 得到 c_{i+1}，最终的 c_n 中存在最大的 $< \infty$ 的元素 $c_{n,l}$，则 l 就是序列 a 的最长递增子序列的长度。

3.3 算法描述

依据以上原理的算法伪代码如下。

需要注意的是，其中的 c 数组与原理中的 c 不完全相同：为了方便重建出 LIS，这里的 c 数组存储的是 c_{ij} 在 a 中的位置。

```
1  LIS(a, n)
2      initialize c[0..n] with c[0]=0, c[1]=1
3      initialize maxLen=1 to denote the
           number of non-infty c[j]
4      for i = 2 to n
5          if a[i]>a[c[maxLen]]:
6              c[++maxLen] = i;
7              last[i] = c[maxLen-1];
8          else
9              p = the first p that a[c[p]]>=
                   a[i];
10             c[p] = i;
11             last[i] = c[p-1];
12     inintialize LIS as a empty list
13     tail = c[maxLen]
14     while tail!=0:
15         push tail to the front of LIS
16         tail = last[tail]
17     return maxLen, LIS
18 }
```

3.4 时间复杂度分析

每次查找 p 可以采用二分查找，时间复杂度是 $O(\lg L)$，总的时间复杂度是 $O(n \lg L)$，其中 L 是 LIS 的长度。在最坏的情况下，$L = n$，复杂度 $O(n \lg n)$；在数据随机的情况下，L 的期望长度大约是 $2\sqrt{L}$[1]，期望复杂度也是 $O(n \lg n)$。

4 结果分析

4.1 结果正确性

首先我们验证了算法实现的正确性，验证方法为：随机生成指定规模的数组 a_i，并调用不同算法，比较他们计算的 LIS 的长度。由于 LIS 可能不唯一，我们还检查了它们给出的子序列是否满足严格递增以及长度是否正确。

实验表明，对于随机生成的数组，两种算法给出的 LIS 的长度总是相同的，且给出的一种 LIS 的方案总是合法无误的。由此可以认为我们对于算法的实现是无误的。

随机序列的产生 由于在计算 LIS 的过程中，只有数字的相对大小是重要的。我们生成随机序列的方法为：为每一个 a_i 随机赋予一个 $[1, n]$ 之间的整数值。

4.2 计算时间

不同输入规模下，不同算法的计算时间统计如表 1 和图 1。

结果分析 从表 1 可以看出，$O(n \lg n)$ 算法的计算时间远快于 $\Theta(n^2)$ 算法；从图 1 可以看出，当 n 较大（$\approx 10^8$）时，计算时间的增长逐渐出现下凸的趋势。这些都与该算法的时间复杂度相符。

5 总结

本次实验对计算最长递增子序列的 $O(n \lg n)$ 算法进行了理论推导、分析与程序设计，并与 $\Theta(n^2)$ 算法进行了比较，实验结果符合预期。另外，还实现了一个简单的图形界面程序方便测试。

[1] https://en.wikipedia.org/wiki/Longest_increasing_subsequence#Length_bounds

n	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8
$\Theta(n^2)$ 算法	3.671×10^{-7}	2.013×10^{-5}	0.002	0.150	14.237			
$O(n \lg n)$ 算法	1.455×10^{-7}	2.623×10^{-6}	3.251×10^{-5}	4.543×10^{-4}	0.006	0.071	0.845	10.584

Table 1: 两种算法在不同输入规模下的计算时间

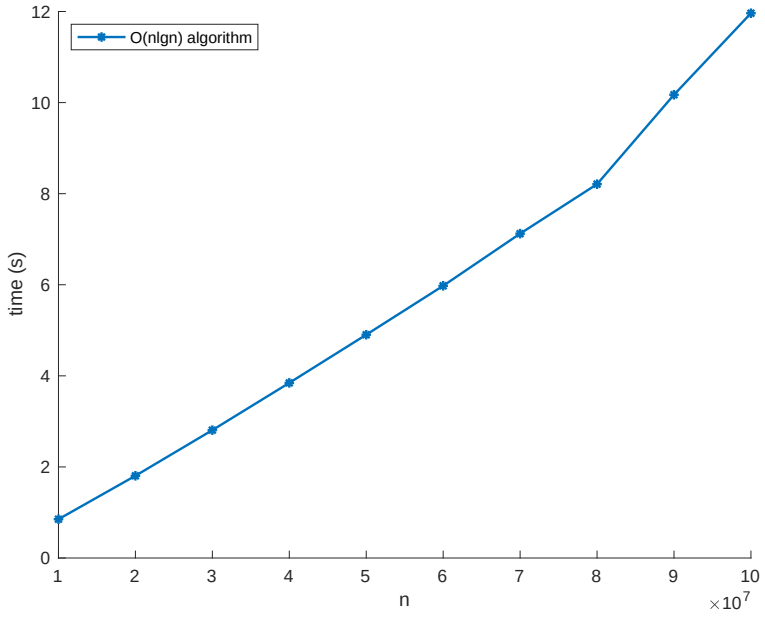

Figure 1: $O(n \lg n)$ 算法在 $n \in [10^7, 10^8]$ 规模下的计算时间