

实验报告

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

实验环境

CPU: Intel Core i5-7200U

内存: 8GB

操作系统: Windows 10 64位教育版

编程语言: Python 3.6

运行方式: 直接运行LIS.py文件即可

算法分析

$O(n^2)$ 时间的算法

记nums为一个长度为 n 的序列， $c[i]$ 为nums的前 n 个数组成的子序列的最长单增子序列长度，则可证明该问题具有最优子结构，递推公式为：

$$c[i] = \begin{cases} 0, & i = 0 \\ \max(c[k] + 1), & k < i \text{ and } x[k] < x[i], \quad i > 0 \end{cases}$$

利用动态规划可得时间复杂度为 $O(n^2)$ 的算法：

```
1 def LIS(nums):
2     n=len(nums)
3     c=[1 for i in range(n)]
4     for i in range(1,n):
5         for k in range(i-1,-1,-1):
6             if nums[k]<nums[i]:
7                 if c[k]+1>c[i]:
8                     c[i]=c[k]+1
9
10    max=0
11    max_index=0
12    for i in range(n):
13        if c[i] > max:
14            max=c[i]
15            max_index=i
16
17    ret=[]
18    ret.append(nums[max_index])
19    max=max-1
20    for i in range(max_index-1,-1,-1):
21        if c[i]==max:
22            ret.append(nums[i])
23            max=max-1
```

```
22     ret.reverse()
23     return ret
```

$O(n \lg n)$ 时间的算法

在 $O(n^2)$ 算法的基础上，我们可以进一步优化。记tails[i]为nums的长度为(i+1)的单增子序列的最小尾数，由题目中的提示可知，tails是一个（非严格）单增数组。遍历nums中的元素nums[i]，若nums[i]大于tails中的最大值tails[j-1]（对应长度为(j)的单增子序列），则nums[i]就是长度为(j+1)的单增子序列的尾数，即tails[j]=nums[i]；若nums[i]在tails[j-1]和tails[j]之间，那么说明tails[j]不是最小的尾数，即tails[j]=nums[i]。同时在这个过程中，c[i]就是nums[i]在tails中所处的位置。因为tails是单增数组，所以可以使用二分查找来压缩时间复杂度，可以达到 $O(n \lg n)$ 。

算法实现如下：

```
1  def LIS(nums):
2      def bsearch(tails, target, l, h):
3          while l <= h:
4              m = (h + l) // 2
5              if tails[m] < target:
6                  l = m + 1
7              elif tails[m] > target:
8                  h = m - 1
9              else:
10                 return m
11         return l
12
13     n=len(nums)
14     tails = [nums[0]]
15     c=[0 for i in range(n)]
16     for i in range(len(nums)):
17         j = bsearch(tails, nums[i], 0, len(tails)-1)
18         c[i] = j;
19         if j >= len(tails):
20             tails.append(nums[i])
21         else:
22             tails[j] = nums[i]
23
24     max=0
25     max_index=0
26     for i in range(n):
27         if c[i] > max:
28             max=c[i]
29             max_index=i
30     ret=[]
31     ret.append(nums[max_index])
32     max=max-1
33     for i in range(max_index-1,-1,-1):
34         if c[i]==max:
35             ret.append(nums[i])
36             max=max-1
37     ret.reverse()
38     return ret
```

结果分析

为了证明 $O(n^2)$ 和 $O(n \lg n)$ 两种算法的正确性和时间差异，采用[LeetCode平台](#)对两种算法分别进行了评测。

$O(n^2)$ 算法的结果：

Submission Detail

24 / 24 test cases passed.

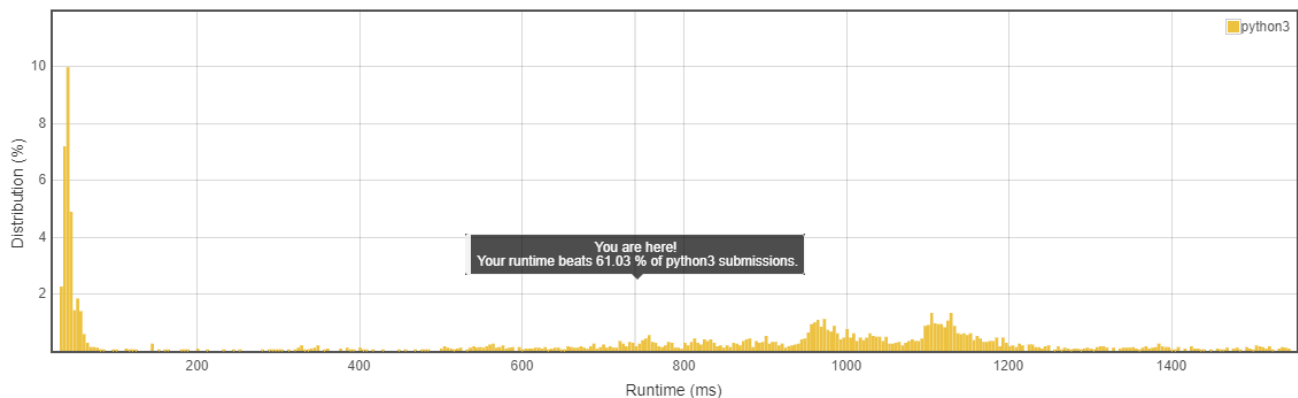
Runtime: 740 ms

Memory Usage: 13.3 MB

Status: Accepted

Submitted: 4 hours, 6 minutes ago

Accepted Solutions Runtime Distribution



$O(n \lg n)$ 算法的结果：

Submission Detail

24 / 24 test cases passed.

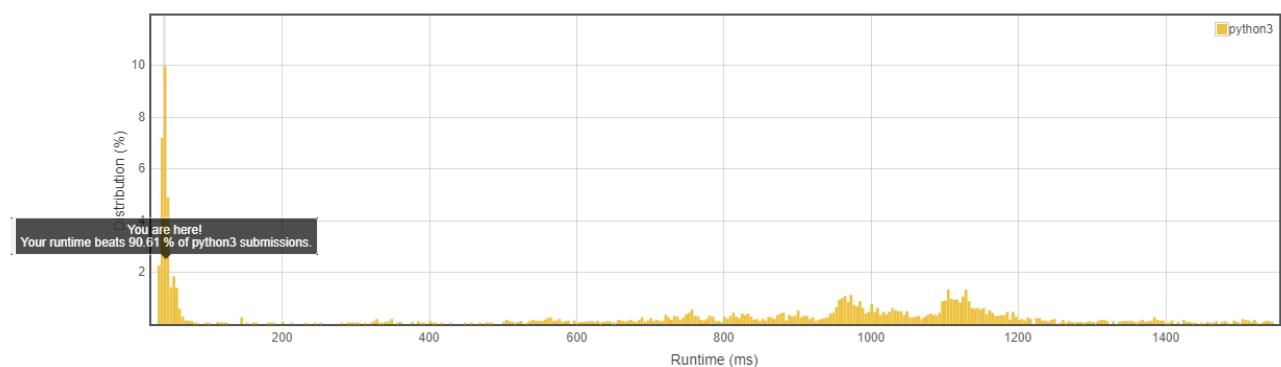
Runtime: 40 ms

Memory Usage: 13.5 MB

Status: Accepted

Submitted: 0 minutes ago

Accepted Solutions Runtime Distribution



两种算法都通过了所有case， $O(n \lg n)$ 算法在运行时间上表现出巨大的优势，两种算法占用内存空间大致相同。