

1 问题描述

给定一个整数序列，找到其中的最长严格递增子序列（LIS）的长度。

2 动态规划算法（时间复杂度 $O(n^2)$ ）

我们可以使用动态规划方法来解决这个问题。定义一个数组 dp ，其中 $dp[i]$ 表示以第 i 个元素结尾的最长严格递增子序列的长度。

设输入数组为 $nums$ 。显然状态转移方程为

$$dp[i] = \max(dp[i], dp[j] + 1) \quad \text{其中 } 0 \leq j < i \text{ 且 } nums[j] < nums[i]$$

2.1 算法描述

- 初始化一个长度为 n 的数组 dp ，使得 $dp[i] = 1$ （因为每个元素本身至少能形成一个长度为1的子序列）。
- 对于每个索引 i 从 1 到 $n - 1$ ，检查所有比它小的索引 j （即 $0 \leq j < i$ ）：
 - 如果 $nums[j] < nums[i]$ ，说明 $nums[i]$ 可以接在以 $nums[j]$ 结尾的子序列后面。更新 $dp[i] = \max(dp[i], dp[j] + 1)$ 。
- 最终 LIS 的长度是 dp 数组中的最大值： $\max(dp)$ 。

2.2 $O(n^2)$ 算法伪代码

Algorithm 1 最长严格递增子序列 ($O(n^2)$)

```
1: 输入: 整数数组  $nums$ 
2:  $n \leftarrow \text{nums}$  的长度
3:  $dp \leftarrow$  大小为  $n$  的数组, 初始化为 1
4: for  $i = 1$  到  $n - 1$  do
5:   for  $j = 0$  到  $i - 1$  do
6:     if  $nums[j] < nums[i]$  then
7:        $dp[i] \leftarrow \max(dp[i], dp[j] + 1)$ 
8:     end if
9:   end for
10: end for
11: 输出:  $\max(dp)$ 
```

2.3 示例:

考虑输入序列: $[10, 22, 9, 33, 21, 50, 41, 60]$ 。

- 初始化 $dp = [1, 1, 1, 1, 1, 1, 1, 1]$ 。
- 对于 $i = 1$ ，更新 $dp = [1, 2, 1, 1, 1, 1, 1, 1]$ 因为 $22 > 10$ 。
- 对于 $i = 2$ ，没有更新， $dp = [1, 2, 1, 1, 1, 1, 1, 1]$ 。

- 对于 $i = 3$, 更新 $dp = [1, 2, 1, 3, 1, 1, 1, 1]$ 因为 $33 > 10, 33 > 22$ 。
- 按此类推, 直到最后。
- 最终 $dp = [1, 2, 1, 3, 2, 4, 4, 5]$, LIS 长度为 5。

3 动态规划加二分法算法 (时间复杂度 $O(n \log n)$)

我们可以使用二分法对动态规划进行优化, 使时间复杂度变为 $O(n \log n)$ 。优化的关键在于将原动态规划方法的 dp 概念进行优化。我们选择用 $dp[i]$ 存储长度为 k 的递增子序列的最末元素, 若有多个长度为 k 的递增子序列, 则记录最小的那个。

3.1 算法描述

- 初始化一个数组 dp , 用于存储最小的严格递增子序列结尾元素, $dp[0]$ 为 $num[0]$ 。
- 对于每个元素 x :
 - 使用二分查找找到 dp 中第一个大于或等于 x 的位置。
 - 如果该位置是数组末尾, 说明 x 可以扩展最长递增子序列; 否则, 用 x 替换该位置的元素。
- 最终, dp 数组的长度即为最长严格递增子序列的长度。

3.2 算法伪代码

Algorithm 2 最长严格递增子序列 ($O(n \log n)$) 算法

```
1: 输入: 整数数组  $nums$ 
2: 初始化  $dp$  为一个空数组,  $dp[0]$  为  $nums[0]$ 
3: for 每个元素  $x$  在  $nums$  中 do
4:   使用二分查找找到  $dp$  中第一个大于或等于  $x$  的位置, 记为  $pos$ 
5:   if  $pos == dp$  的长度 then
6:     将  $x$  添加到  $dp$  的末尾
7:   else
8:     将  $dp[pos]$  替换为  $x$ 
9:   end if
10: end for
11: 输出:  $dp$  的长度
```

3.3 示例:

假设输入序列为: $[9, 2, 1, 5, 3, 6, 4, 8, 9, 7]$

执行过程如下:

- 初始时, 令 $dp[0] = 9$, $len = 1$ 。
- 处理 $num[1] = 2$, 将 $dp[0] = 2$, $len = 1$ 。
- 处理 $num[2] = 1$, 将 $dp[0] = 1$, $len = 1$ 。

- 处理 $num[3] = 5$, 因为 $5 > dp[0]$, 所以 $dp[1] = 5$, $len = 2$ 。
- 处理 $num[4] = 3$, 使用二分查找找到替换位置, $dp[1] = 3$, $len = 2$ 。
- 处理 $num[5] = 6$, 因为 $6 > dp[1]$, 所以 $dp[2] = 6$, $len = 3$ 。
- 处理 $num[6] = 4$, 使用二分查找找到替换位置, $dp[2] = 4$, $len = 3$ 。
- 处理 $num[7] = 8$, 因为 $8 > dp[2]$, 所以 $dp[3] = 8$, $len = 4$ 。
- 处理 $num[8] = 9$, 因为 $9 > dp[3]$, 所以 $dp[4] = 9$, $len = 5$ 。
- 处理 $num[9] = 7$, 使用二分查找找到替换位置, $dp[3] = 7$, $len = 5$ 。

最终, 最长递增子序列的长度为 5。

4 总结

本文提供了两种解决最长严格递增子序列 (LIS) 问题的算法。第一种算法使用动态规划, 时间复杂度为 $O(n^2)$, 第二种算法使用二分查找, 优化了时间复杂度为 $O(n \log n)$ 。