

最长严格递增子序列

问题：给定一个n个数组成的序列 $A = [a_1, a_2, \dots, a_n]$ ，找出其最长严格递增子序列（要求序列中任意两元素，位置在前的元素严格小于位置在后的元素）

法1 动态规划 $O(n^2)$

构造两个数组 length 和 pre

$\text{length}[i]$ 表示 $A[0:j]$ 到 $A[i:j]$ 的最长子序列长度

$\text{pre}[i]$ 表示 $A[0:j]$ 到 $A[i:j]$ 的最长子序列中第 i 位的前一个元素的下标

for $i = 1 \rightarrow n-1$:

for $j = 0 \rightarrow i-1$:

if $A[i:j] > A[j:j] \& \& \text{length}[i:j] <$

$\text{length}[i:j+1]$

$\text{length}[i:j] = \text{length}[j:j+1]$

$\text{pre}[i:j] = j$

反推可得

$\rightarrow \text{length}$ 中最大值即为最长子序列长度， pre 数组

eg. A : 0 1 2 3 4 5 6
 length : 1 1 1 \times^2 \times^2 $\{ \times^2 \}$ $\{ \times^2 \}$
 pre : -1 -1 -1 \times^2 \times^2 $\{ \times^2 \}$ $\{ \times^2 \}$

i = 1 元

2 元

3 $5 > 2$ $| [3] = 1 + 1 = 2$ $\text{pre}[3] = 2$

4 $3 > 2$ $| [4] = 1 + 1 = 2$ $\text{pre}[4] = 2$

5 $7 > 2$ $| [5] = 1 + 1 = 2$ $\text{pre}[5] = 2$

$7 > 5$ $2 + 1 > 2$

$\therefore [5] = 2 + 1 = 3$ $\text{pre}[5] = 3$

$7 > 3$ $2 + 1 = 3$ 不操作

b. $18 > 8$ $[6] = 1 + 1 = 2$ $\text{pre}[6] = 0$

$9, 2 | + 1 = 2$ 不操作

$$5. \quad 2t1 > 2. \therefore [lb] = 3$$

$$\text{pre}[b] = 3$$

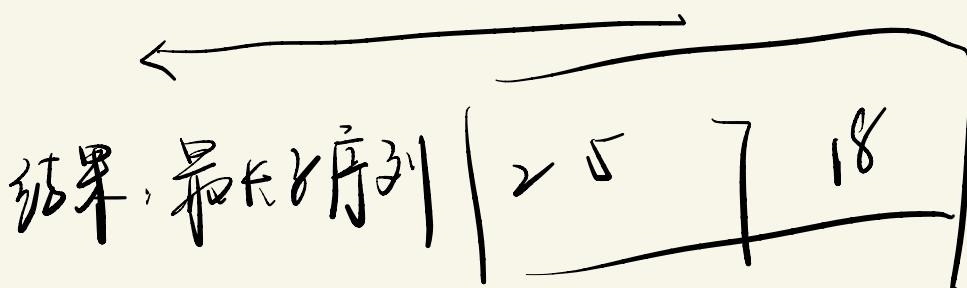
$$3 \quad 2t1 = 3 \quad \text{不操作}$$

$$T \quad 5t1 > 3 \therefore [tb] = 4$$

$$\text{pre}[b] = 5$$

遍历一下 最长长度为 4. 末位下标 b

b → 5 → 3 → 2 → -1
18 7 5 2. end.



代码：

```
vector<int> longestIncreasingSubsequence(const vector<int>& A) {
    int n = A.size();

    // length[i] 表示以 A[i] 结尾的最长递增子序列的长度，初始值都为1，即每个元素至少可以组成一个长度为1的子序列
    vector<int> length(n, 1);
    // pre[i] 用来记录以 A[i] 结尾的最长递增子序列的前一个元素索引，初始值都为-1，即没有前一个元素
    vector<int> pre(n, -1);

    // 动态规划计算最长递增子序列的长度
    for (int i = 1; i < n; ++i) {
        for (int j = 0; j < i; ++j) {
            if (A[i] > A[j] && length[i] < length[j] + 1) {
                length[i] = length[j] + 1;
                pre[i] = j;
            }
        }
    }

    // 找出 length 数组中的最大值及其对应的索引
    int index = max_element(length.begin(), length.end()) - length.begin();

    // 通过 pre 数组重建最长递增子序列
    vector<int> lis;
    while (index != -1) {
        lis.push_back(A[index]);
        index = pre[index];
    }

    // 由于是从末尾逆向重建的，所以需要反转结果
    reverse(lis.begin(), lis.end());

    return lis;
}
```

(完整代码也在git库中)

法2. 贪心 + 二分: $O(n \log n)$

$$n \times \log n$$

构建三 T 数组

tails: tails[i] 记录目前长度为 i+1 的递增子序列的最小尾部元素，初始化为空[]

pre: pre[i] 同上，记录 A[i:] 在 LIS 中前驱元素的下标，初始化 [-1, -1, -1, ..., -1] (-1 口)

indices: indices[i] 记录每个元素在 tails 中的下标位置，初始化为空[]

算法思路: 遍历 A[i]，通过二分查找在 tails 数组中找到合适的位置，替换掉略大于该数的数；若 A[i] 大于 tails 数组中的所有数，则将其直接添加于 tails 尾部。最终 tails 的长度就是 LIS 长度，由 tails 末位和 pre 数组可推导出最长子序列。

代码实现

```
for i = 0 → n-1 :  
    pos = erfen(tails, A[i]);  
    if (pos == tails.size())  
        tails.push(A[i])  
    else    tails[pos] = A[i];  
    indices[pos] = i  
  
    if (pos > 0)  
        pre[i] = indices[pos-1].
```

代码

```
vector<int> longestIncreasingSubsequence(const vector<int>& A) {
    int n = A.size();
    if (n == 0) return {};

    vector<int> tails; // 保存递增子序列的最小尾部元素
    vector<int> pre(n, -1); // 保存每个元素的前驱元素索引
    vector<int> indices(n); // 保存每个元素在 tails 中的索引位置

    for (int i = 0; i < n; ++i) {
        int pos = erfen(tails, A[i]); // 找到元素 A[i] 应该插入的位置

        // 如果 pos 等于 tails.size(), 说明 num 比所有现有元素都大, 应添加到 tails 的末尾
        if (pos == tails.size()) {
            tails.push_back(A[i]);
        } else {
            // 否则, 更新 tails 中第 pos 位置的元素为当前元素
            tails[pos] = A[i];
        }

        // 保存当前元素的前驱索引
        indices[pos] = i;

        // 更新前驱元素索引
        if (pos > 0) {
            pre[i] = indices[pos - 1];
        }
    }

    // 回溯得到最长递增子序列
    vector<int> lis;
    int index = indices[tails.size() - 1]; // 从最后一个元素的索引开始回溯
    while (index != -1) {
        lis.push_back(A[index]);
        index = pre[index]; // 回溯到前一个元素
    }

    reverse(lis.begin(), lis.end()); // 由于回溯是从尾部开始的, 需要反转
}
```

完整代码见 git 库

解法

$$A = \{20, 1, 3, 2, 5, 7, 4, 10, 9, 11\}.$$

$$\text{tail} = \{2\}$$

$$\{1\}.$$

$$\{1, 3\}.$$

$$\{1, 2\}$$

$$\{1, 2, 5\}$$

$$\{1, 2, 5, 7\}$$

$$\{1, 2, 4, 7\}$$

$$\{1, 2, 4, 7, 10\}$$

$$\{1, 2, 4, 7, 9\}$$

$$\{1, 2, 4, 7, 9, 11\}$$

\therefore 总长度 6, 最后一位为 11

再同上例推得

LIS 为 {1, 2, 5, 7, 9, 11}