最长严格递增子序列

动态规划法（O（n^2））

文字说明：

数组 num 是我们要研究的序列，另外再设置两个数组 dp 和 trace。dp 记录子序列长度，其中 dp[i]表示以元素 num[i]为结尾的最长严格递增子序列的长度；trace 追踪我们所要找的最长严格子序列 LIS 的信息，trace[i]追踪 LIS 中元素 num[i]的前一个元素在 num 中的下标。

易知，对于 j<i，若 num[j]<num[i]，则 dp[i]=Max{dp[j]+1, dp[i]}

因此首先将 dp 初始化为 dp[i]=1，表明一开始以 num[i]结尾的每个子序列最长长度为 1；将 trace[i]均初始化为-1，表明一开始每个 num[i]均无对应的前一个元素下标。

对于每一个 num[i]，遍历 0<=j<i，若 num[j]<num[i]，比较 dp[j]+1 与 dp[i]，若 dp[j]+1>dp[i]，则 dp[i]=dp[j]+1，且 trace[i]=j。

最后根据 dp[i]最大处开始，利用 trace[i]反向追踪出最长严格递增子序列。

示例

num = {82, 47, 63, 59, 91, 34, 8, 99}

初始化 dp = {1, 1, 1, 1, 1, 1, 1, 1}

初始化 trace = {-1, -1, -1, -1, -1, -1, -1, -1}

当 i=0 时，前面没有元素，dp[0]=1, trace[0] = [-1]

当 i=1 时，num[0]>num[1], dp[1]=1, trace[1]=[-1]

当 i=2 时，num[0]>num[2], num[1]<num[2], dp[1]+1>dp[2],

则 dp[2]=dp[1]+1=2, trace[2]=1

当 i=3 时，num[0]>num[3], num[1]<num[3], dp[1]+1>dp[3],

则 dp[3]= 2, trace[3]=1, num[2]>num[3]

当 i=4 时，num[0]<num[4], 则 dp[4]=2, trce[4]=0, num[1]<num[4], num[2]<num[4], 则 dp[4]=3, trace[4]=2, num[3]<num[4], 但是 dp[3]+1=dp[4], 无法覆盖。

以此类推最终各数组列成的表格为

下标	0	1	2	3	4	5	6	7
num	82	47	63	59	91	34	8	99
dp	1	1	2	2	3	1	1	4
trace	-1	-1	1	1	2	-1	-1	4

通过遍历 dp 数组找到其中最大值，该最大值就是整个序列的最长严格递增子序列的长度，此例中最大值为 4，即原序列 num 的最长严格递增子序列长度为 4。并且，若想具体找出这个最长严格递增子序列，可通过 trace 数组进行回溯，从 dp 中最大值对应的索引位置(7)开始，依据 trace 记录的前驱索引不断往前找，直到索引为-1 退出，就能还原出完整的最长严格递增子序列 LIS=[47,63,91,99]。

伪代码

输入：数组 num

```
N = length of num;
dp = [1]*N;
```

```
trace = [-1]*N;
for (int i = 1;i< n;i++){
    for( int j =0,j<i,j++){
        if(num[j]<num[i]){
            if(dp[j]+1>dp[i]){
                dp[i]=dp[j]+1;
                trace[i]=j;
            }
       }
  }
}
Maxlength = 0;
Maxindex = 0;
for(int i=0;i<n;i++){
  if(dp[i]>Maxlength){
    Maxlength = dp[i];
    Maxindex = i;
  }
}
创建长度为 Maxlength 的数组 LIS
for(k=Maxlength-1;k>=0;k--){
  LIS(k)= num(Maxindex);
  Maxindex = trace(Maxindex);
}
return Maxlength, LIS
```

二分查找和贪心算法（O(nlogn)）

文字说明：

数组 num 是我们要研究的序列，另外再设置一个 vector　tail 和一个数组 trace。dp 记录子序列末尾元素，其中 tail[i]表示长度为 i+1 的严格递增子序列的最小末尾元素的下标；trace 追踪我们所要找的最长严格子序列 LIS 的信息，trace[i]追踪 LIS 中元素 num[i]的前一个元素在 num 中的下标。

首先我们将 tail 初始化为一个空数组，trace[i]皆初始化为-1。对于每一个 num[i]，若 num[i]>tail 中的最后一个(下标为 j)所对应的 num 中的元素，先更新 trace[i]=tail[j]然后将 i 加到 tail 的末位；否则，利用二分法寻找 tail 数组中第一个所对应的元素大于 num[i]的数，并将 i 替换这个数在 tail 中的位置,若替换位置在末位，则 trace[i]=tail[j-1]。

然后从 tail 的最后一个元素开始，利用 trace[i]反向追踪，获得 LIS。

示例：

```
    num = {82, 47, 63, 59, 91, 34, 8, 99}
初始化 tail = {}
初始化 trace = {-1, -1, -1, -1, -1, -1, -1, -1}
初始化 length = 0
```

当 i=0 时，tail 没有元素，则 tail[0]=0,trace[0]=-1
当 i=1 时，num[1]<num[tail.back],num[tail[0]]>num[1],则 tail[0]=1,trace[1]=-1
当 i=2 时，num[2]>num[tail.back],则 tail[1]=2,trace[2]=1
当 i=3 时，num[3]<num[tail.back],num[tail[1]]>num[3],则 tail[1]=3,trace[3]=1
当 i=4 时，num[4]>num[tail.back]，则 tail[2]=4,trace[4]=3
以此类推最终各数组列成的表格为

下标	0	1	2	3	4	5	6	7
num	82	47	63	59	91	34	8	99
tail	6	3	4	7				
trace	-1	-1	1	1	3	-1	-1	4

最终，通过 num[tail.back]找到整个序列的最长严格递增子序列的末位，然后通过 trace 数组进行回溯，依据 trace 记录的前驱索引不断往前找,直到索引为-1 退出，就能还原出完整的最长严格递增子序列 LIS=[47,59,91,99]。

伪代码：

```
输入：数组 num
N = length of num;
tail = {};
trace = [-1]*N;
for(i=0;i<N;i++){
  //当前元素大于 tail 末位对应元素，直接加到尾部。
  if(tail.size=0 or num[i]>num[tail.back]){
    trace[i] = tail[tail.size()-1];
    tail.push_back[i];
  }
  //当前元素小于于 tail 末位对应元素二分查找第一个大于 num[i]的数的下标，并用
i 替代
  else (num[i]>num[tail.back) {
    left = 0
    right = len(tail) - 1
    while left < right:
    mid = left + (right - left) / 2
    if  (num[tails[mid]] <= num[i])
      left = mid + 1;
    else
      right = mid;
    tails[left] = i;
    if(left = len(tail)-1){
    trace[i] = tail(len(tail)-2);
  }
Maxlength = tail.size();
Maxindex = tace.back;
创建长度为 Maxlength 的数组 LIS
for(k=Maxlength-1;k>=0;k--){
  LIS(k)= num(Maxindex);
```

```
  Maxindex = trace(Maxindex);
}
return Maxlength, LIS
```