步学习。

最长递增子序列

【题目】

给定数组 arr，返回 arr 的最长递增子序列。

【举例】

arr=[2,1,5,3,6,4,8,9,7]，返回的最长递增子序列为[1,3,4,8,9]。

【要求】

如果 arr 长度为 N，请实现时间复杂度为 $O(N\log N)$的方法。

【难度】

校 ★★★☆

【解答】

先介绍时间复杂度为 $O(N^2)$的方法，具体过程如下：

1．生成长度为 N 的数组 dp，dp[i]表示在以 arr[i]这个数结尾的情况下，arr[0..i]中的最大递增子序列长度。

2．对第一个数 arr[0]来说，令 dp[0]=1，接下来从左到右依次算出以每个位置的数结尾的情况下，最长递增子序列长度。

3．假设计算到位置 i，求以 arr[i]结尾情况下的最长递增子序列长度，即 dp[i]。如果最长递增子序列以 arr[i]结尾，那么在 arr[0..i-1]中所有比 arr[i]小的数都可以作为倒数第二个数。在这么多倒数第二个数的选择中，以哪个数结尾的最大递增子序列更大，就选那个数作为倒数第二个数，所以 dp[i]=max{dp[j]+1(0<=j<i，arr[j]<arr[i])}。如果 arr[0..i-1]中所有的数都不比 arr[i]小，令 dp[i]=1 即可，说明以 arr[i]结尾情况下的最长递增子序列只包含 arr[i]。

按照步骤 1~3 可以计算出 dp 数组，具体过程请参看如下代码中的 getdp1 方法。

```
public int[] getdp1(int[] arr) {
        int[] dp = new int[arr.length];
        for (int i = 0; i < arr.length; i++) {
```

```
                dp[i] = 1;
                for (int j = 0; j < i; j++) {
                        if (arr[i] > arr[j]) {
                                dp[i] = Math.max(dp[i], dp[j] + 1);
                        }
                }
        }
        return dp;
}
```

接下来解释如何根据求出的 dp 数组得到最长递增子序列。以题目的例子来说明，arr=[2,1,5,3,6,4,8,9,7]，求出的数组 dp=[1,1,2,2,3,3,4,5,4]。

1．遍历 dp 数组，找到最大值以及位置。在本例中最大值为 5，位置为 7，说明最终的最长递增子序列的长度为 5，并且应该以 arr[7]这个数(arr[7]==9)结尾。

2．从 arr 数组的位置 7 开始从右向左遍历。如果对某一个位置 *i*，既有 arr[i]<arr[7]，又有 dp[i]==dp[7]-1，说明 arr[i]可以作为最长递增子序列的倒数第二个数。在本例中，arr[6]<arr[7]，并且 dp[6]==dp[7]-1，所以 8 应该作为最长递增子序列的倒数第二个数。

3．从 arr 数组的位置 6 开始继续向左遍历，按照同样的过程找到倒数第三个数。在本例中，位置 5 满足 arr[5]<arr[6]，并且 dp[5]==dp[6]-1，同时位置 4 也满足。选 arr[5]或者 arr[4]作为倒数第三个数都可以。

4．重复这样的过程，直到所有的数都找出来。

dp 数组包含每一步决策的信息，其实根据 dp 数组找出最长递增子序列的过程就是从某一个位置开始逆序还原出决策路径的过程。具体过程请参看如下代码中的 generateLIS 方法。

```
public int[] generateLIS(int[] arr, int[] dp) {
        int len = 0;
        int index = 0;
        for (int i = 0; i < dp.length; i++) {
                if (dp[i] > len) {
                        len = dp[i];
                        index = i;
                }
        }
        int[] lis = new int[len];
        lis[--len] = arr[index];
        for (int i = index; i >= 0; i--) {
                if (arr[i] < arr[index] && dp[i] == dp[index] - 1) {
                        lis[--len] = arr[i];
                        index = i;
                }
        }
        return lis;
```

```
}
```

整个过程的主方法参看如下代码中的 lis1 方法。

```
public int[] lis1(int[] arr) {
        if (arr == null || arr.length == 0) {
                return null;
        }
        int[] dp = getdp1(arr);
        return generateLIS(arr, dp);
}
```

很明显，计算 dp 数组过程的时间复杂度为 $O(N^2)$，根据 dp 数组得到最长递增子序列过程的时间复杂度为 $O(N)$，所以整个过程的时间复杂度为 $O(N^2)$。如果让时间复杂度达到 $O(N\log N)$，只要让计算 dp 数组的过程达到时间复杂度 $O(N\log N)$即可，之后根据 dp 数组生成最长递增子序列的过程是一样的。

时间复杂度 $O(N\log N)$生成 dp 数组的过程是利用二分查找来进行的优化。先生成一个长度为 N 的数组 ends，初始时 ends[0]=arr[0]，其他位置上的值为 0。生成整型变量 right，初始时 right=0。在从左到右遍历 arr 数组的过程中，求解 dp[i]的过程需要使用 ends 数组和 right 变量，所以这里解释一下其含义。遍历的过程中，ends[0..right]为有效区，ends[right+1..N-1]为无效区。对有效区上的位置 b，如果有 ends[b]==c，则表示遍历到目前为止，在所有长度为 b+1 的递增序列中，最小的结尾数是 c。无效区的位置则没有意义。

比如，arr=[2,1,5,3,6,4,8,9,7]，初始时 dp[0]=1，ends[0]=2，right=0。ends[0..0]为有效区，ends[0]==2 的含义是，在遍历过 arr[0]之后，所有长度为 1 的递增序列中(此时只有[2])，最小的结尾数是 2。之后的遍历继续用这个例子来说明求解过程。

1．遍历到 arr[1]==1。ends 有效区=ends[0..0]=[2]，在有效区中找到最左边的大于或等于 arr[1]的数。发现是 ends[0]，表示以 arr[1]结尾的最长递增序列只有 arr[1]，所以令 dp[1]=1。然后令 ends[0]=1，因为遍历到目前为止，在所有长度为 1 的递增序列中，最小的结尾数是 1，而不再是 2。

2．遍历到 arr[2]==5。ends 有效区=ends[0..0]=[1]，在有效区中找到最左边大于或等于 arr[2]的数。发现没有这样的数，表示以 arr[2]结尾的最长递增序列长度=ends 有效区长度+1，所以令 dp[2]=2。ends 整个有效区都没有比 arr[2]更大的数，说明发现了比 ends 有效区长度更长的递增序列，于是把有效区扩大，ends 有效区=ends[0..1]=[1,5]。

3．遍历到 arr[3]==3。ends 有效区=ends[0..1]=[1,5]，在有效区中用二分法找到最左边大于或等于 arr[3]的数。发现是 ends[1]，表示以 arr[3]结尾的最长递增序列长度为 2，所以

令 dp[3]=2。然后令 ends[1]=3，因为遍历到目前为止，在所有长度为 2 的递增序列中，最小的结尾数是 3，而不再是 5。

4．遍历到 arr[4]==6。ends 有效区=ends[0..1]=[1,3]，在有效区中用二分法找到最左边大于或等于 arr[4]的数。发现没有这样的数，表示以 arr[4]结尾的最长递增序列长度=ends 有效区长度+1，所以令 dp[4]=3。ends 整个有效区都没有比 arr[4]更大的数，说明发现了比 ends 有效区长度更长的递增序列，于是把有效区扩大，ends 有效区=ends[0..2]=[1,3,6]。

5．遍历到 arr[5]==4。ends 有效区=ends[0..2]=[1,3,6]，在有效区中用二分法找到最左边大于或等于 arr[5]的数。发现是 ends[2]，表示以 arr[5]结尾的最长递增序列长度为 3，所以令 dp[5]=3。然后令 ends[2]=4，表示在所有长度为 3 的递增序列中，最小的结尾数变为 4。

6．遍历到 arr[6]==8。ends 有效区=ends[0..2]=[1,3,4]，在有效区中用二分法找到最左边大于或等于 arr[6]的数。发现没有这样的数，表示以 arr[6]结尾的最长递增序列长度=ends 有效区长度+1，所以令 dp[6]=4。ends 整个有效区都没有比 arr[6]更大的数，说明发现了比 ends 有效区长度更长的递增序列，于是把有效区扩大，ends 有效区=ends[0..3]=[1,3,4,8]。

7．遍历到 arr[7]==9。ends 有效区=ends[0..3]=[1,3,4,8]，在有效区中用二分法找到最左边大于或等于 arr[7]的数。发现没有这样的数，表示以 arr[7]结尾的最长递增序列长度=ends 有效区长度+1，所以令 dp[7]=5。ends 整个有效区都没有比 arr[7]更大的数，于是把有效区扩大，ends 有效区=ends[0..5]=[1,3,4,8,9]。

8．遍历到 arr[8]==7。ends 有效区=ends[0..5]=[1,3,4,8,9]，在有效区中用二分法找到最左边大于或等于 arr[8]的数。发现是 ends[3]，表示以 arr[8]结尾的最长递增序列长度为 4，所以令 dp[8]=4。然后令 ends[3]=7，表示在所有长度为 4 的递增序列中，最小的结尾数变为 7。

具体过程请参看如下代码中的 getdp2 方法。

```
public int[] getdp2(int[] arr) {
        int[] dp = new int[arr.length];
        int[] ends = new int[arr.length];
        ends[0] = arr[0];
        dp[0] = 1;
        int right = 0;
        int l = 0;
        int r = 0;
        int m = 0;
        for (int i = 1; i < arr.length; i++) {
                l = 0;
                r = right;
                while (l <= r) {
                        m = (l + r) / 2;
```

```
                    if (arr[i] > ends[m]) {
                        l = m + 1;
                    } else {
                        r = m - 1;
                    }
                }
                right = Math.max(right, l);
                ends[l] = arr[i];
                dp[i] = l + 1;
            }
            return dp;
        }
```

时间复杂度 $O(N\log N)$方法的整个过程请参看如下代码中的 lis2 方法。

```
    public int[] lis2(int[] arr) {
            if (arr == null || arr.length == 0) {
                return null;
            }
            int[] dp = getdp2(arr);
            return generateLIS(arr, dp);
    }
```

汉诺塔问题

【题目】

给定一个整数 n，代表汉诺塔游戏中从小到大放置的 n 个圆盘，假设开始时所有的圆盘都放在左边的柱子上，想按照汉诺塔游戏的要求把所有的圆盘都移到右边的柱子上。实现函数打印最优移动轨迹。

【举例】

n=1 时，打印：

move from left to right

n=2 时，打印：

move from left to mid

move from left to right

move from mid to right