

动态规划-最长上升子序列模型

一、AcWing 1017. 怪盗基德的滑翔翼

【题目描述】

假设城市中一共有 N 幢建筑排成一条线，每幢建筑的高度各不相同。

初始时，怪盗基德可以在任何一幢建筑的顶端。

他可以选择一个方向逃跑，但是不能中途改变方向（因为中森警部会在后面追击）。

因为滑翔翼动力装置受损，他只能往下滑行（即：只能从较高的建筑滑翔到较低的建筑）。

他希望尽可能多地经过不同建筑的顶部，这样可以减缓下降时的冲击力，减少受伤的可能性。

请问，他最多可以经过多少幢不同建筑的顶部(包含初始时的建筑)？

【输入格式】

输入数据第一行是一个整数 K ，代表有 K 组测试数据。

每组测试数据包含两行：第一行是一个整数 N ，代表有 N 幢建筑。第二行包含 N 个不同的整数，每一个对应一幢建筑的高度 h ，按照建筑的排列顺序给出。

【输出格式】

对于每一组测试数据，输出一行，包含一个整数，代表怪盗基德最多可以经过的建筑数量。

【数据范围】

$$1 \leq K \leq 100$$

$$1 \leq N \leq 100$$

$$0 < h < 10000$$

【输入样例】

```
1 3
2 8
3 300 207 155 299 298 170 158 65
4 8
5 65 158 170 298 299 155 207 300
6 10
7 2 1 3 4 5 6 7 8 9 10
```

【输出样例】

```
1 6
2 6
3 9
```

【分析】

确定滑行方向后就转化为了 LIS 问题，原问题相当于求解正向和反向以 a_i 为结尾的最长上升子序列长度，分别正向和反向各进行一次 LIS ，取得最大值即可。

【朴素版代码】

```
1  #include <iostream>
2  #include <algorithm>
3  using namespace std;
4
5  const int N = 110;
6  int h[N], f[N];
7  int n;
8
9  int main()
10 {
11     int T;
12     cin >> T;
13     while (T--)
14     {
15         cin >> n;
16         for (int i = 1; i <= n; i++) cin >> h[i];
17         int res = 0;
18         //正向LIS
19         for (int i = 1; i <= n; i++)
20         {
```

```

21         f[i] = 1;
22         for (int j = 1; j < i; j++)
23             if (h[j] < h[i]) f[i] = max(f[i], f[j] + 1);
24         res = max(res, f[i]);
25     }
26     //反向LIS
27     for (int i = n; i; i--)
28     {
29         f[i] = 1;
30         for (int j = n; j > i; j--)
31             if (h[j] < h[i]) f[i] = max(f[i], f[j] + 1);
32         res = max(res, f[i]);
33     }
34     cout << res << endl;
35 }
36 return 0;
37 }

```

【优化版代码】

```

1  #include <iostream>
2  #include <algorithm>
3  using namespace std;
4
5  const int N = 110;
6  int h[N], f[N], uf[N]; //f记录正向LIS, uf记录反向LIS
7  int n;
8
9  int main()
10 {
11     int T;
12     cin >> T;
13     while (T--)
14     {
15         cin >> n;
16         for (int i = 1; i <= n; i++) cin >> h[i];
17         int cnt1 = 0, cnt2 = 0;
18         f[cnt1++] = h[1], uf[cnt2++] = h[n];
19         for (int i = 2, j = n - 1; i <= n; i++, j--)
20         {
21             if (h[i] > f[cnt1 - 1]) f[cnt1++] = h[i];
22             else
23

```

```

24         int l = 0, r = cnt1 - 1;
25         while (l < r)
26         {
27             int mid = l + r >> 1;
28             if (f[mid] >= h[i]) r = mid;
29             else l = mid + 1;
30         }
31         f[r] = h[i];
32     }
33     if (h[j] > uf[cnt2 - 1]) uf[cnt2++] = h[j];
34     else
35     {
36         int l = 0, r = cnt2 - 1;
37         while (l < r)
38         {
39             int mid = l + r >> 1;
40             if (uf[mid] >= h[j]) r = mid;
41             else l = mid + 1;
42         }
43         uf[r] = h[j];
44     }
45 }
46 cout << max(cnt1, cnt2) << endl;
47 }
48 return 0;
49 }

```

二、AcWing 1014. 登山

【题目描述】

五一到了，ACM队组织大家去登山观光，队员们发现山上一共有 N 个景点，并且决定按照顺序来浏览这些景点，即每次所浏览景点的编号都要大于前一个浏览景点的编号。

同时队员们还有另一个登山习惯，就是不连续浏览海拔相同的两个景点，并且一旦开始下山，就不再向上走了。

队员们希望在满足上面条件的同时，尽可能多的浏览景点，你能帮他们找出最多可能浏览的景点数么？

【输入格式】

第一行包含整数 N ，表示景点数量。

第二行包含 N 个整数，表示每个景点的海拔。

【输出格式】

输出一个整数，表示最多能浏览的景点数。

【数据范围】

$$2 \leq N \leq 1000$$

【输入样例】

```
1 8
2 186 186 150 200 160 130 197 220
```

【输出样例】

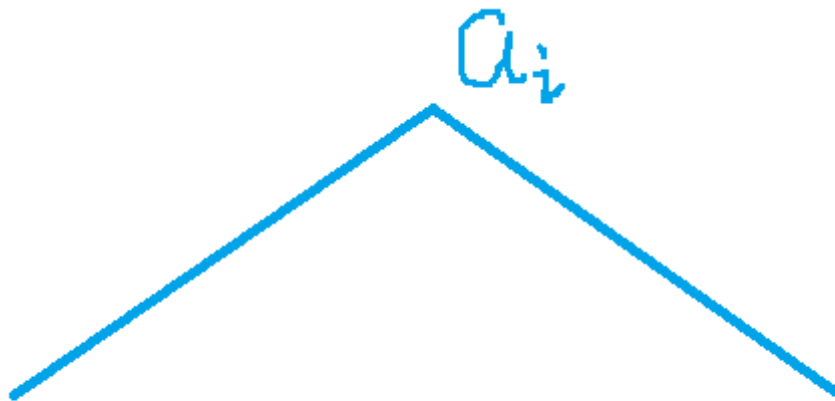
```
1 4
```

【分析】

条件：

- 按照编号递增的顺序浏览
- 相邻两个景点高度不能相同
- 一旦开始下降就不能上升

因此浏览的路线如下图所示：



CSDN @ 聆歌

我们可以预处理出以每个点结尾的正向 LIS ($f[i]$) 和反向 LIS ($g[i]$) 长度，然后枚举每个分界点 a_i ，找出左右两边的 LIS 的值的最大值即可，即 $res = \max(f[i] + g[i] - 1)$ ，由于分界点 a_i 在两边的 LIS 中都被计算了一次因此答案需要 -1 。

【代码】

```
1 #include <iostream>
2 #include <algorithm>
3 using namespace std;
4
5 const int N = 1010;
6 int f[N], g[N];
7 int h[N];
8 int n, res;
9
10 int main()
11 {
12     cin >> n;
13     for (int i = 1; i <= n; i++) cin >> h[i];
14     for (int i = 1; i <= n; i++)
15     {
16         f[i] = 1;
17         for (int j = 1; j < i; j++)
18             if (h[j] < h[i]) f[i] = max(f[i], f[j] + 1);
19     }
20     for (int i = n; i; i--)
21     {
22         g[i] = 1;
23         for (int j = n; j > i; j--)
24             if (h[j] < h[i]) g[i] = max(g[i], g[j] + 1);
25     }
26     for (int i = 1; i <= n; i++) res = max(res, f[i] + g[i] - 1);
27     cout << res << endl;
28     return 0;
29 }
```

三、AcWing 482. 合唱队形

【题目描述】

N 位同学站成一排，音乐老师要请其中的 $(N - K)$ 位同学出列，使得剩下的 K 位同学排成合唱队形。

合唱队形是指这样的一种队形：设 K 位同学从左到右依次编号为 $1, 2, \dots, K$ ，他们的身高分别为 T_1, T_2, \dots, T_K ，则他们的身高满足 $T_1 < \dots < T_i > T_{i+1} > \dots > T_K (1 \leq i \leq K)$ 。

你的任务是，已知所有 N 位同学的身高，计算最少需要几位同学出列，可以使得剩下的同学排成合唱队形。

【输入格式】

共二行。

第一行是一个整数 n ，表示同学的总数。

第二行有 n 个整数，用空格分隔，第 i 个整数 t_i 是第 i 位同学的身高（厘米）。

【输出格式】

一个整数，最少需要几位同学出列。

【数据范围】

$$2 \leq N \leq 100$$

$$130 \leq t_i \leq 230$$

【输入样例】

```
1 8
2 186 186 150 200 160 130 197 220
```

【输出样例】

```
1 4
```

【分析】

本题与问题三完全一样。

求解需要几位同学出列，即为求解最多能保留几位同学。

求解同学 i 左边的最长上升子序列 $f[i]$ ，右边的最长下降子序列 $g[i]$ ，枚举 i 求出 $res = f[i] + g[i] - 1$ 的最大值（ k 被计算了两次所以减一），最终答案即为 $n - res$ 。

【代码】

```
1 #include <iostream>
2 #include <algorithm>
3 using namespace std;
4
5 const int N = 110;
```

```

6  int f[N], g[N];
7  int h[N];
8  int n, res;
9
10 int main()
11 {
12     cin >> n;
13     for (int i = 1; i <= n; i++) cin >> h[i];
14     for (int i = 1; i <= n; i++)
15     {
16         f[i] = 1;
17         for (int j = 1; j < i; j++)
18             if (h[j] < h[i]) f[i] = max(f[i], f[j] + 1);
19     }
20     for (int i = n; i; i--)
21     {
22         g[i] = 1;
23         for (int j = n; j > i; j--)
24             if (h[j] < h[i]) g[i] = max(g[i], g[j] + 1);
25     }
26     for (int i = 1; i <= n; i++) res = max(res, f[i] + g[i] - 1);
27     cout << n - res << endl;
28     return 0;
29 }

```

四、AcWing 1012. 友好城市

【题目描述】

Palmia国有一条横贯东西的大河，河有笔直的南北两岸，岸上各有位置各不相同的 N 个城市。

北岸的每个城市有且仅有一个友好城市在南岸，而且不同城市的友好城市不相同。

每对友好城市都向政府申请在河上开辟一条直线航道连接两个城市，但是由于河上雾太大，政府决定避免任意两条航道交叉，以避免事故。

编程帮助政府做出一些批准和拒绝申请的决定，使得在保证任意两条航线不相交的情况下，被批准的申请尽量多。

【输入格式】

第1行，一个整数 N ，表示城市数。

第2行到第 $n + 1$ 行，每行两个整数，中间用1个空格隔开，分别表示南岸和北岸的一对友好城市的坐标。

【输出格式】

仅一行，输出一个整数，表示政府所能批准的最多申请数。

【数据范围】

$$1 \leq N \leq 5000$$

$$0 \leq x_i \leq 10000$$

【输入样例】

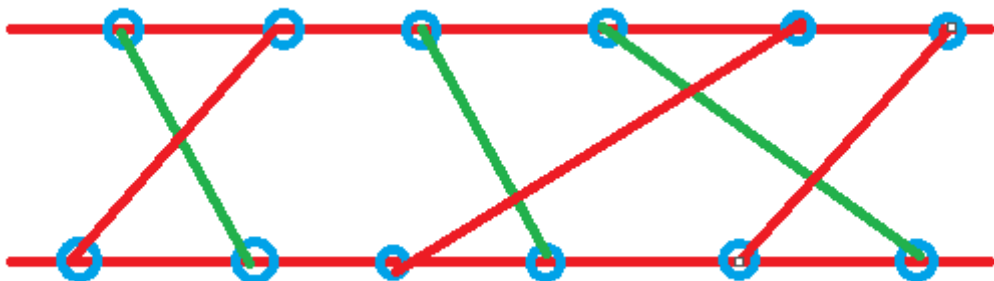
```
1 7
2 22 4
3 2 6
4 10 3
5 15 12
6 9 8
7 17 17
8 4 2
```

【输出样例】

```
1 4
```

【分析】

根据题意可画出模拟图如下：



首先我们将南岸的城市按坐标从小到大进行排序，然后从前往后选择，只有当选择的南岸城市所对应的北岸城市坐标严格单调递增才能保证所建的桥不会交叉。例如我们在南岸选择了城市1，建立桥如红线所示，其对应的北岸友好城市为2，此时不能选择南岸的城市2，因为其对应的北岸友好城市在城市2的左边，不满足严格单调递增的要求，因此会产生交

叉。

所以我们的目标就是在某岸城市的坐标从小到大排好序后求出另一岸城市坐标的最长上升子序列。

【代码】

```
1  #include <iostream>
2  #include <algorithm>
3  using namespace std;
4
5  typedef pair<int, int> PII;
6  const int N = 5010;
7  PII h[N];
8  int f[N], cnt, n;
9
10 int main()
11 {
12     cin >> n;
13     for (int i = 0; i < n; i++) cin >> h[i].first >> h[i].second;
14     sort(h, h + n);
15     f[cnt++] = h[0].second;
16     for (int i = 1; i < n; i++)
17         if (h[i].second > f[cnt - 1]) f[cnt++] = h[i].second;
18     else
19     {
20         int l = 0, r = cnt - 1;
21         while (l < r)
22         {
23             int mid = l + r >> 1;
24             if (f[mid] >= h[i].second) r = mid;
25             else l = mid + 1;
26         }
27         f[r] = h[i].second;
28     }
29     cout << cnt << endl;
30     return 0;
31 }
```

五、AcWing 1016. 最大上升子序列和

【题目描述】

一个数的序列 b_i ，当 $b_1 < b_2 < \dots < b_s$ 的时候，我们称这个序列是上升的。

对于给定的一个序列 (a_1, a_2, \dots, a_N) ，我们可以得到一些上升的子序列 $(a_{i_1}, a_{i_2}, \dots, a_{i_K})$ ，这里 $1 \leq i_1 < i_2 < \dots < i_K \leq N$ 。

比如，对于序列 $(1, 7, 3, 5, 9, 4, 8)$ ，有它的一些上升子序列，如 $(1, 7)$, $(3, 4, 8)$ 等等。

这些子序列中和最大为18，为子序列 $(1, 3, 5, 9)$ 的和。

你的任务，就是对于给定的序列，求出最大上升子序列和。

注意，最长的上升子序列的和不一定是最大的，比如序列 $(100, 1, 2, 3)$ 的最大上升子序列和为100，而最长上升子序列为 $(1, 2, 3)$ 。

【输入格式】

输入的第一行是序列的长度 N 。

第二行给出序列中的 N 个整数，这些整数的取值范围都在 $0 \sim 10000$ （可能重复）。

【输出格式】

输出一个整数，表示最大上升子序列和。

【数据范围】

$$1 \leq N \leq 1000$$

【输入样例】

```
1 | 7
2 | 1 7 3 5 9 4 8
```

【输出样例】

```
1 | 18
```

【分析】

状态表示：

- 集合： $f[i]$ 表示所有以 $h[i]$ 结尾的上升子序列
- 属性：和的最大值

状态计算：

假设倒数第二个数是 $h[k]$

- 如果不选 $f[k]$ ，即 $h[i] > f[k]$ ，那么 $f[i] = h[i]$
- 如果选 $f[k]$ ，那么 $f[i] = f[k] + h[i]$

【代码】

```
1  #include <iostream>
2  #include <algorithm>
3  using namespace std;
4
5  const int N = 1010;
6  int h[N], f[N];
7  int n, res;
8
9  int main()
10 {
11     cin >> n;
12     for (int i = 0; i < n; i++) cin >> h[i];
13     for (int i = 0; i < n; i++)
14     {
15         f[i] = h[i];
16         for (int j = 0; j < i; j++)
17             if (h[j] < h[i]) f[i] = max(f[i], f[j] + h[i]);
18         res = max(res, f[i]);
19     }
20     cout << res << endl;
21     return 0;
22 }
```

六、AcWing 1010. 拦截导弹

【题目描述】

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。

但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。

某天，雷达捕捉到敌国的导弹来袭。

由于该系统还在试用阶段，所以只有一套系统，因此有可能不能拦截所有的导弹。

输入导弹依次飞来的高度（雷达给出的高度数据是不大于**30000**的正整数，导弹数不超过**1000**），计算这套系统最多能拦截多少导弹，如果要拦截所有导弹最少要配备多少套这种导弹拦截系统。

【输入格式】

共一行，输入导弹依次飞来的高度。

【输出格式】

第一行包含一个整数，表示最多能拦截的导弹数。

第二行包含一个整数，表示要拦截所有导弹最少要配备的系统数。

【数据范围】

雷达给出的高度数据是不大于**30000**的正整数，导弹数不超过**1000**（优化版代码 $O(n\log n)$ 可过数据范围为**100000**）。

【输入样例】

```
1 389 207 155 300 299 170 158 65
```

【输出样例】

```
1 6
2 2
```

【分析】

计算这套系统最多能拦截多少导弹即为求解最长下降子序列。

对于求解最少要配备多少套这种导弹拦截系统，可使用贪心算法进行分析：

从前往后扫描每个数：


- 情况一：如果现有的子序列的结尾都小于当前数，则创建新子序列
- 情况二：将当前数放到结尾大于等于它的最小的子序列后面

我们用 $g[i]$ 表示第 i 个下降子序列末尾的数，因此 g 一定是单调递增的

优化：

对于求解最长下降子序列问题，可使用贪心与二分的思想将复杂度优化至 $O(n\log n)$ ，对于第二个子问题，根据上述分析，也可使用二分的方法找到结尾大于等于该数的最小的子序列，因此复杂度也为 $O(n\log n)$ 。


由于洛谷该题有 10^5 数据范围的样例，因此两种代码在洛谷中评测结果如下，AC代码为 $O(n\log n)$ 写法，只过一半样例的代码为 $O(n^2)$ 写法：

 **AsanoSaki**
10-28 14:52:16

Unaccepted
100

P1020 [NOIP1999 普及组] 导弹拦截

🕒 12.59s / 📄 1.00MB / 📦 452B C++11

 **AsanoSaki**
10-28 14:51:42

Accepted
200

P1020 [NOIP1999 普及组] 导弹拦截

🕒 290ms / 📄 988.00KB / 📦 653B C++11

【朴素版代码 $O(n^2)$ 】

```
1 #include <iostream>
2 #include <algorithm>
3 using namespace std;
4
5 const int N = 1010;
6 int h[N], f[N], g[N];
7 int n, res, cnt;
8
9 int main()
10 {
11     while (cin >> h[n]) n++;
12     for (int i = 0; i < n; i++)
13     {
14         f[i] = 1;
15         for (int j = 0; j < i; j++)
16             if (h[j] >= h[i]) f[i] = max(f[i], f[j] + 1);
17         res = max(res, f[i]);
18         int k = 0;
19         while (k < cnt && g[k] < h[i]) k++;
20         g[k] = h[i];
21         if (k == cnt) cnt++;
22     }
23     cout << res << endl << cnt << endl;
24     return 0;
25 }
```

【优化版代码 $O(n\log n)$ 】

```
1 #include <iostream>
2 #include <algorithm>
3 using namespace std;
4
```

```

5  const int N = 100010;
6  int h[N], f[N], g[N];
7  int n, res, cnt;
8
9  int main()
10 {
11     while (cin >> h[n]) n++;
12     f[res++] = h[0], g[cnt++] = h[0];
13     for (int i = 1; i < n; i++)
14     {
15         if (h[i] <= f[res - 1]) f[res++] = h[i];
16         else
17         {
18             int l = 0, r = res - 1;
19             while (l < r)
20             {
21                 int mid = l + r >> 1;
22                 if (f[mid] < h[i]) r = mid;
23                 else l = mid + 1;
24             }
25             f[r] = h[i];
26         }
27         int l = 0, r = cnt; //可能g中所有数都小于当前数，需要开新序列，因此r需
要+1,
28         while (l < r) //二分找到第一个大于等于当前数的位置
29         {
30             int mid = l + r >> 1;
31             if (g[mid] >= h[i]) r = mid;
32             else l = mid + 1;
33         }
34         g[r] = h[i];
35         if (r == cnt) cnt++;
36     }
37     cout << res << endl << cnt << endl;
38     return 0;
39 }

```

七、AcWing 187. 导弹防御系统

【题目描述】

为了对抗附近恶意国家的威胁，**R**国更新了他们的导弹防御系统。

一套防御系统的导弹拦截高度要么一直严格单调上升要么一直严格单调下降。

例如，一套系统先后拦截了高度为**3**和高度为**4**的两发导弹，那么接下来该系统就只能拦截高度大于**4**的导弹。

给定即将袭来的一系列导弹的高度，请你求出至少需要多少套防御系统，就可以将它们全部击落。

【输入格式】

输入包含多组测试用例。

对于每个测试用例，第一行包含整数 **n** ，表示来袭导弹数量。

第二行包含 **n** 个不同的整数，表示每个导弹的高度。

当输入测试用例 **$n = 0$** 时，表示输入终止，且该用例无需处理。

【输出格式】

对于每个测试用例，输出一个占据一行的整数，表示所需的防御系统数量。

【数据范围】

$$1 \leq n \leq 50$$

【输入样例】

```
1 | 5
2 | 3 5 2 4 1
3 | 0
```

【输出样例】

```
1 | 2
```

【分析】

用 $up[k]$ 和 $down[k]$ 记录第 **k** 套上升（下降）系统目前所拦截的最后一个导弹；

$dfs(x, su, sd)$ 意味着已有 **su** 个上升， **sd** 个下降，正在处理第 **x** 个数；

处理上升子序列与下降子序列的方式与上题同理。

【代码】

```
1 | #include <iostream>
2 | #include <algorithm>
```



```

3 using namespace std;
4
5 const int N = 55;
6 int h[N], up[N], down[N];
7 int n, res;
8
9 //x表示当前元素的下标, su表示上升子序列长度, sd表示下降子序列长度
10 void dfs(int x, int su, int sd)
11 {
12     //剪枝与判断终点
13     if (su + sd >= res) return;
14     if (x == n)
15     {
16         res = su + sd;
17         return;
18     }
19
20     //情况一: 将当前数放到上升子序列中
21     int k = 0;
22     while (k < su && up[k] >= h[x]) k++;
23     int t = up[k]; //保存现场
24     up[k] = h[x]; //更新
25     if (k < su) dfs(x + 1, su, sd); //不需要开新序列
26     else dfs(x + 1, su + 1, sd); //开新序列
27     up[k] = t; //恢复现场
28
29     //情况二: 将当前数放到下降子序列中
30     k = 0;
31     while (k < sd && down[k] <= h[x]) k++;
32     t = down[k];
33     down[k] = h[x];
34     if (k < sd) dfs(x + 1, su, sd);
35     else dfs(x + 1, su, sd + 1);
36     down[k] = t;
37 }
38
39 int main()
40 {
41     while (cin >> n, n)
42     {
43         for (int i = 0; i < n; i++) cin >> h[i];
44         res = n;
45
46         dfs(0, 0, 0);
47     }
48 }

```

```
46         cout << res << endl;
47     }
48     return 0;
49 }
```

八、AcWing 272. 最长公共上升子序列

【题目描述】

熊大妈的奶牛在小沐沐的熏陶下开始研究信息题目。

小沐沐先让奶牛研究了最长上升子序列，再让他们研究了最长公共子序列，现在又让他们研究最长公共上升子序列了。

小沐沐说，对于两个数列***A***和***B***，如果它们都包含一段位置不一定连续的数，且数值是严格递增的，那么称这一段数是两个数列的公共上升子序列，而所有的公共上升子序列中最长的就是最长公共上升子序列了。

奶牛半懂不懂，小沐沐要你来告诉奶牛什么是最长公共上升子序列。

不过，只要告诉奶牛它的长度就可以了。

数列***A***和***B***的长度均不超过**3000**。

【输入格式】

第一行包含一个整数***N***，表示数列***A***，***B***的长度。

第二行包含***N***个整数，表示数列***A***。

第三行包含***N***个整数，表示数列***B***。

【输出格式】

输出一个整数，表示最长公共上升子序列的长度。

【数据范围】

$1 \leq N \leq 3000$ ，序列中的数字均不超过 $2^{31} - 1$ 。

【输入样例】

```
1 4
2 2 2 1 3
3 2 1 2 3
```

【输出样例】

【分析】

*LCS*前置知识:

- $f[i][j] = f[i-1][j-1] + 1, (i, j > 0, a[i] = b[j])$
- $f[i][j] = \max(f[i][j-1], f[i-1][j]), (i, j > 0, a[i] \neq b[j])$

其中, $f[i][j]$ 为*a*序列前*i*个元素与*b*序列前*j*个元素的*LCS*长度。

*LIS*前置知识:

- $f[i] = \max(f[i], f[j] + 1), (j < i, a[j] < a[i])$

其中, $f[i]$ 为以第*i*个元素结尾的*LIS*长度。

*LCIS*分析:

我们想办法糅合这两种动态规划的思想, 设 $f[i][j]$ 代表所有 $a[1 \sim i]$ 和 $b[1 \sim j]$ 中以 $b[j]$ 结尾的公共上升子序列的集合, 其值等于该集合的子序列中长度的最大值。

首先依据公共子序列中是否包含 $a[i]$, 将 $f[i][j]$ 所代表的集合划分成两个不重不漏的子集:

- 不包含 $a[i]$ 的子集, 最大值是 $f[i-1][j]$;
- 包含 $a[i]$ 的子集, 将这个子集继续划分, 依据是子序列的倒数第二个元素在 $b[]$ 中是哪个数:
 - (1) 子序列只包含 $b[j]$ 一个数, 长度是1;
 - (2) 子序列的倒数第二个数是 $b[1]$ 的集合, 最大长度是 $f[i-1][1] + 1$;
 - ...
 - (n) 子序列的倒数第二个数是 $b[j-1]$ 的集合, 最大长度是 $f[i-1][j-1] + 1$ 。

如果直接按上述思路实现, 需要三重循环:

```

1  for (int i = 1; i <= n; i++)
2  {
3      for (int j = 1; j <= n; j++)
4      {
5          f[i][j] = f[i-1][j];
6          if (a[i] == b[j])
7          {
8              int maxv = 1;
9              for (int k = 1; k < j; k++)
10                 if (b[k] < b[j])

```

```

11         maxv = max(maxv, f[i - 1][k] + 1);
12         f[i][j] = max(f[i][j], maxv);
13     }
14 }
15 }

```

然后我们发现每次循环求得的 `maxv` 是满足 $b[k] < a[i]$ （因为 $b[j] == a[i]$ ）的 $f[i - 1][k] + 1$ 的前缀最大值。

因此可以直接将 `maxv` 提到第一层循环外面，减少重复计算，此时只剩下两重循环。

最终答案枚举子序列结尾取最大值即可。

【 $O(n^3)$ 代码】

```

1  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4  using namespace std;
5
6  const int N = 3010;
7  int a[N], b[N];
8  int f[N][N];
9  int n, res;
10
11 int main()
12 {
13     cin >> n;
14     for (int i = 1; i <= n; i++) cin >> a[i];
15     for (int i = 1; i <= n; i++) cin >> b[i];
16     for (int i = 1; i <= n; i++)
17         for (int j = 1; j <= n; j++)
18         {
19             f[i][j] = f[i - 1][j]; //公共上升子序列中不包含a[i]的情况
20             if (a[i] == b[j])
21             {
22                 f[i][j] = max(f[i][j], 1); //公共上升子序列倒数第二个数为空集
23                 for (int k = 1; k < j; k++)
24                     if (b[k] < b[j])
25                         f[i][j] = max(f[i][j], f[i - 1][k] + 1); //倒数第
26                 二个数为b[k]
27             }
28         }
29 }

```

```

28     for (int i = 1; i <= n; i++) res = max(res, f[n][i]); //枚举序列以哪个
    数结尾
29     cout << res << endl;
30     return 0;
31 }

```

【 $O(n^2)$ 代码】

```

1  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4  using namespace std;
5
6  const int N = 3010;
7  int a[N], b[N];
8  int f[N][N];
9  int n, res;
10
11 int main()
12 {
13     cin >> n;
14     for (int i = 1; i <= n; i++) cin >> a[i];
15     for (int i = 1; i <= n; i++) cin >> b[i];
16     for (int i = 1; i <= n; i++)
17     {
18         int maxv = 1;
19         for (int j = 1; j <= n; j++)
20         {
21             f[i][j] = f[i - 1][j];
22             if (a[i] == b[j]) f[i][j] = max(f[i][j], maxv);
23             if (b[j] < a[i]) maxv = max(maxv, f[i - 1][j] + 1);
24         }
25     }
26     for (int i = 1; i <= n; i++) res = max(res, f[n][i]);
27     cout << res << endl;
28     return 0;
29 }

```