

## L2C1 Lesson2

DP状态计算划分依据扩展：最后一个不同的点位

最长上升子序列模型（LIS问题）

- 怪盗基德的滑翔翼（初始时选择任一位置任意方向，中途不能改改变方向）
    - 确定起点和方向之后，退化为LIS问题
- 起点：a[i]
- 最长距离：：以a[i]结尾的最长上升子序列（俩个方向上分别求解LIS问题）

```
1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  const int N = 110;
7
8  int n;
9  int a[N], f[N];
10
11 int main() {
12     int T;
13     scanf("%d", &T);
14     while (T--) {
15         scanf("%d", &n);
16         for (int i=1; i<=n; i++) scanf("%d", &a[i]);
17
18         int res = 0;
19         //正向求解LIS
20         for (int i=1; i<=n; i++) { //以a[i]结尾的LIS解
21             f[i] = 1; //前一个数字为空
22             for (int j=1; j<i; j++)
23                 if (a[i] > a[j])
24                     f[i] = max(f[i], f[j]+1);
25             res = max(res, f[i]);
26         }
27
28         //反向求解LIS
29         for (int i=n; i>=1; i--) {
30             f[i] = 1;
31             for (int j=n; j>i; j--)
32                 if (a[i] > a[j])
33                     f[i] = max(f[i], f[j]+1);
34             res = max(res, f[i]);
35         }
36
37         printf("%d\n", res);
38     }
39
40     return 0;
```

- 登山问题

- 条件1: 按照编号递增顺序浏览 => 子序列
- 条件2: 相邻两个景点海拔不能相同
- 条件3: 一旦开始下降, 就不能上升 => 走过的路线先严格单调上升后严格单调下降
- 目标: 求最多能浏览多少景点
  - 所有形状满足上述条件的子序列长度的最大值
- 思路:
  - 状态表示
    - 集合: 所有顶点是 $a[i]$ 且满足上述条件的子序列集合
    - 属性:  $ff[i]$ 子序列集合的长度最大值
  - 状态计算
    - 性质: 对于 $ff[i]$ 所代表的子序列集合而言,  $a[k]$ 左右两边互相独立, 故左右两边分别求最大值
      - 左边: 所有以 $a[k]$ 结尾的LIS问题的长度的最大值
      - 右边: 所有反向以 $a[k]$ 结尾的LIS问题最大值
    - 所以对每个点 $a[i]$ 的正反向LIS答案进行预处理 (优化时间复杂度) 求解  
 $f[i]$ 表示 $a[i]$ 结尾正向LIS答案,  $g[i]$ 表示 $a[i]$ 结尾反向LIS答案。则 $ff[i] = f[i] + g[i] - 1$

```

1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  const int N = 1010;
7
8  int n;
9  int a[N];
10 int f[N], g[N];
11
12 int main() {
13     scanf("%d", &n);
14
15     for (int i=1; i<=n; i++) scanf("%d", &a[i]);
16
17     //预处理正向LIS
18     for (int i=1; i<=n; i++) {
19         f[i] = 1;
20         for (int j=1; j<i; j++)
21             if (a[i] > a[j])
22                 f[i] = max(f[i], f[j]+1);
23     }
24
25     //预处理反向LIS
26     for (int i=n; i>=1; i--) {

```

```

27         g[i] = 1;
28         for (int j=n; j>i; j--)
29             if (a[i] > a[j])
30                 g[i] = max(g[i], g[j]+1);
31     }
32
33     int res = 0;
34     for (int k=1; k<=n; k++) res = max(res, f[k]+g[k]-1);
35
36     printf("%d\n", res);
37
38     return 0;
39 }

```

- 合唱队行（登山问题变形，队友问题）
  - 去掉的人数最少，即合唱队形最长，可以先求出最长合唱队形，再由总数减去该值即为答案

```

1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  const int N = 110;
7
8  int n;
9  int a[N];
10 int f[N], g[N];
11
12 int main() {
13     scanf("%d", &n);
14     for (int i=1; i<=n; i++) scanf("%d", &a[i]);
15
16     for (int i=1; i<=n; i++) {
17         f[i] = 1;
18         for (int j=1; j<i; j++)
19             if (a[i] > a[j])
20                 f[i] = max(f[i], f[j]+1);
21     }
22
23     for (int i=n; i>=1; i--) {
24         g[i] = 1;
25         for (int j=n; j>i; j--)
26             if (a[i] > a[j])
27                 g[i] = max(g[i], g[j]+1);
28     }
29
30     int res = 0;
31     for (int k=1; k<=n; k++) res = max(res, f[k]+g[k]-1);
32
33     printf("%d", n-res);
34

```

```
35     return 0;
36 }
37
```

- 友好城市

- 条件1: 每个城市上只能建立一座桥
- 条件2: 只能在友好城市之间建桥
- 条件3: 所有桥与桥之间不能相交
- 目标: 最多可以建多少桥
- 思路:
  - 将两岸分开考虑, 以南侧为自变量考虑, 将 (南侧城市坐标, 对应北侧友好城市坐标) 按照第一个关键字从小到大排序, 可发现所有合法建桥方案的北侧城市的坐标序列必须是上升子序列, 且因变量北侧城市的每一个上升子序列都对应一个合法的建桥方式, 因此上升子序列长度的最大值即为答案。

```
1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  typedef pair<int, int> PII;
7
8  const int N = 5010;
9
10 int n;
11 PII q[N];
12 int f[N];
13
14 int main() {
15     scanf("%d", &n);
16     for (int i=0; i<n; i++) scanf("%d%d", &q[i].first,
17                                     &q[i].second);
18     sort(q, q+n);    //按南岸城市坐标从小到大排序
19
20     int res = 0;
21     for (int i=0; i<n; i++) {
22         f[i] = 1;
23         for (int j=0; j<i; j++)
24             if (q[i].second > q[j].second)
25                 f[i] = max(f[i], f[j]+1);
26
27         res = max(res, f[i]);
28     }
29
30     printf("%d", res);
31
32     return 0;
33 }
```

- 最大上升子序列和
  - 状态表示:  $f[i]$ 
    - 集合: 所有以  $a[i]$  结尾的上升子序列
    - 属性: 子序列和的最大值
  - 状态计算: 按子序列倒数第二个数进行分类, 假设倒数第二个数是  $a[k]$ , 则所有倒数第二个数是  $a[k]$  的子序列的最大值是  $f[k]+a[i]$ , 其他类类似, 因此,  $f[i] = \max(f[j]+a[i])$ , 且满足  $a[j] < a[i]$ ,  $j=0, \dots, i-1$

```

1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  const int N = 1010;
7
8  int n;
9  int a[N];
10 int f[N];
11
12 int main() {
13     scanf("%d", &n);
14
15     for (int i=1; i<=n; i++) scanf("%d", &a[i]);
16
17     int res = 0;
18     for (int i=1; i<=n; i++) {
19         f[i] = a[i];
20         for (int j=1; j<i; j++)
21             if (a[i] > a[j])
22                 f[i] = max(f[i], f[j]+a[i]);    //注意是a[i]
23
24         res = max(res, f[i]);
25     }
26
27     printf("%d", res);
28
29     return 0;
30 }

```

- 拦截导弹 (LIS+贪心)
- 导弹防御系统
- 最长公共上升子序列

