L2C1 Lesson3

> 最长上升子序列
>
> - 拦截导弹（LIS+**贪心**）
>   - 第一问：寻找最长不上升子序列（LIS问题简易变形）
>   - 第二问：至少需要多少最长不上升子序列才能覆盖总序列（**贪心**）
>   - 贪心流程：从前往后扫描每个数，对于每个数，只有两种情况：接在某个子序列之后或者创建一个新子序列。因此，贪心的一种思路为：
>   - 情况1：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2：存在一些子序列结尾大于等于当前数，则将该数放在结尾大于等于该数且最小的子序列之后。
>   - 贪心证明：
>   - **如何证两个数相等？ A>=B, A<=B**
>
>   A表示贪心算法得到的序列个数，B表示最优解，∴B<=A。
>
>   如何证A <= B，通过**调整法**
>
>   假设最优解对应的方案和当前方案不同。则可从前往后找到第一个分配方式不同的数
>
>   贪心法：会将当前数（记为x）放置到大于等于该数且结尾最小（记为a）的子序列之后
>
>   最优解：放置在另一个不同的子序列之后，结尾记为b，则由贪心策略可知，b>=a
>
>   记贪心法得到的最终结果为（...ax...），最优解得到的最终结果为（...bx...）则可将两种方式的(x...)部分互换且不增加最长不上升子序列的个数。不断重复这个过程，最多调整n次，则可将贪心法的结果调整为最优解的结果，且每一次调整并没有增加子序列的个数，因此A <= B。
>
>   所以综上所述A=B，贪心策略正确
>
>   实现方式与LIS贪心解决方法类似，两者之间存在转换关系，（离散数学Dilworth定理）
>
>   通过g[i]存储第i个最长不上升子序列结尾的值

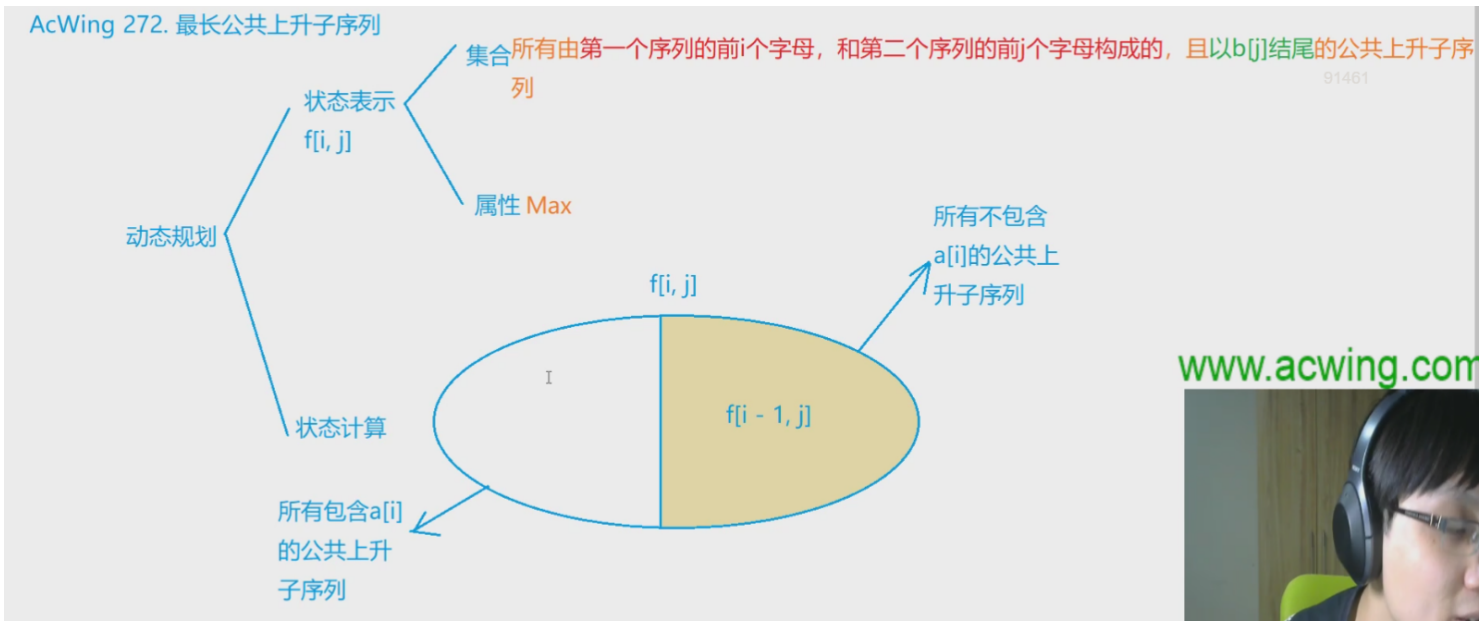
```cpp
#include <iostream>
#include <algorithm>
#include <sstream>

using namespace std;

const int N = 1010;

int n;
int a[N];
int f[N], g[N];     //g[i]表示第i个最长不上升子序列结尾的值

int main() {
    string line;
    getline(cin, line);
    stringstream ssin(line);    //注意读入方式
    while (ssin >> a[n]) n++;

    int res = 0;    //LIS求解
    for (int i=0; i<n; i++) {
        f[i] = 1;
        for (int j=0; j<i; j++)
            if (a[i] <= a[j])
                f[i] = max(f[i], f[j]+1);
        res = max(res, f[i]);
    }

    cout << res << endl;

    int cnt = 0;    //记录最长不上升子序列的数量
    for (int i=0; i<n; i++) {
        int k = 0;
        while (k<cnt && g[k]<a[i]) k++;    //寻找第一个大于等于a[i]且结尾最小的序列
        g[k] = a[i];
        if (k >= cnt) cnt++;
    }

    cout << cnt;

    return 0;
}
```

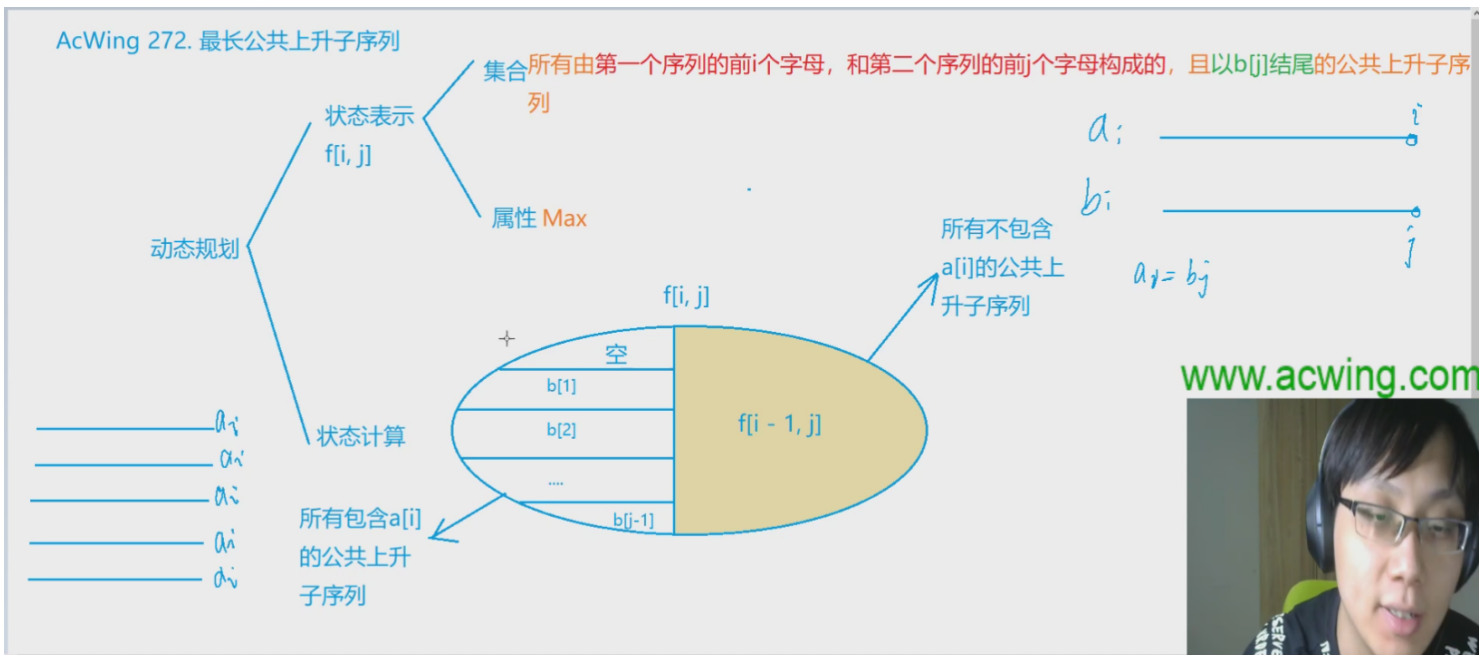
> - 导弹防御系统

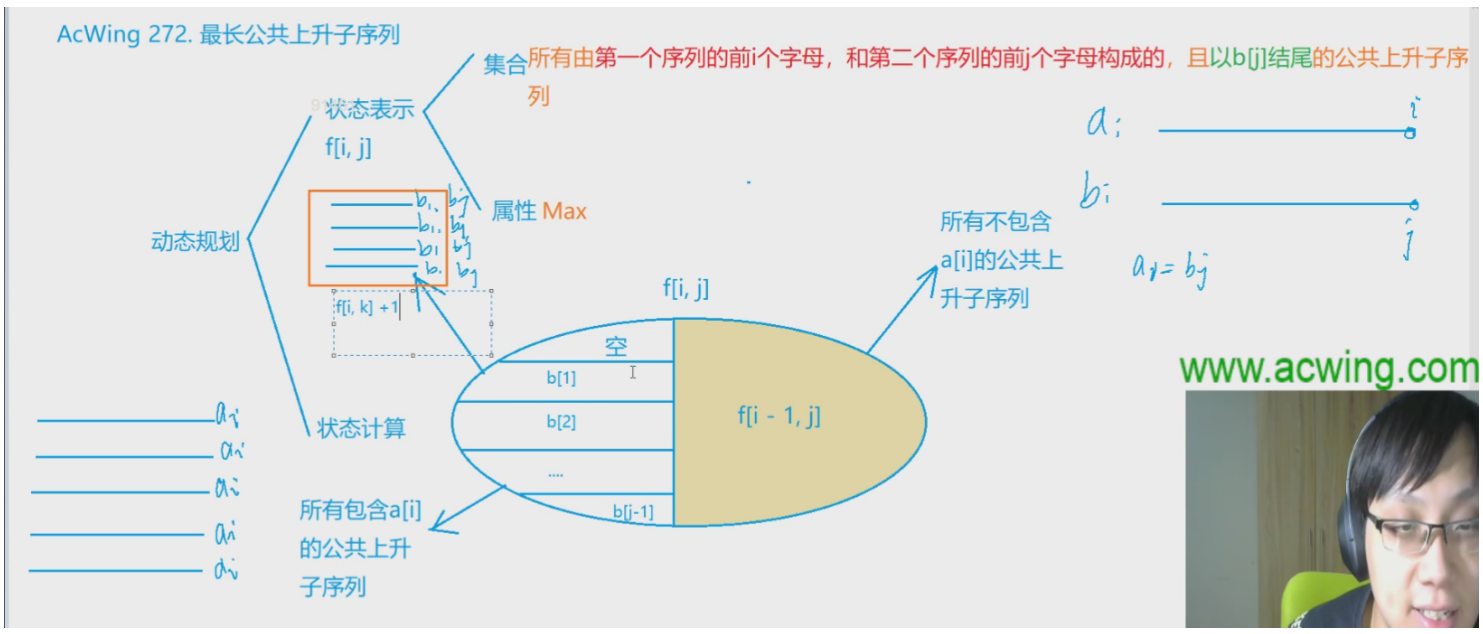
- 不同于拦截导弹问题，贪心时对于每个数需要考虑到底将其放置于上升子序列还是下降子序列，没有更好的解法，贪心外套一层暴搜，注意暴搜写法
- dfs求最小步数：
  - 记忆全局最小值（bfs可直接求出，但使用时容易空间爆栈，需要指数空间且不好剪枝，DFS线性空间且容易剪枝）
  - 迭代加深

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 55;

int n;
int q[N];
int up[N], down[N]; //up和down数组分别代表上升和下降子序列的g数组
int ans;    //记录全局最小值

//u代表当前处理到了第u个点，su表示当前上升子序列的数量，sd表示当前下降子序列的数量
void dfs(int u, int su, int sd) {
    if (su+sd >= ans) return;    //剪枝，ans不会被更新，注意等号
    if (u == n) {       ////找到一种新方案，此处ans<=su+sd
        ans = su+sd;
        return;
    }

    //分情况讨论
    //情况1：放入上升子序列
    int k = 0;
    while (k<su && up[k]>=q[u]) k++;
    int t = up[k];  //恢复现场用
    up[k] =  q[u];
    if (k < su) dfs(u+1, su, sd);   //未创建新序列
    else dfs(u+1, su+1, sd);    //创建新序列
    up[k] = t;  //恢复现场

    //情况2：放入下降子序列
    k = 0;
    while (k<sd && down[k]<=q[u]) k++;
    t = down[k];
    down[k] = q[u];
    if (k < sd) dfs(u+1, su, sd);
    else dfs(u+1, su, sd+1);
    down[k] = t;
}

int main() {
    while (cin >> n, n) {
        for (int i=0; i<n; i++) cin >> q[i];

        ans = n;    //初始化ans为最大值
        dfs(0, 0, 0);

        cout << ans << endl;
    }

    return 0;
}
```

- 最长公共上升子序列
  - 结合公共子序列(LCS)和上升子序列(LIS)的划分思想。
  - 分析过程，a序列和b序列地位等同，故上升子序列的分析对象可任选一个







- 图中包含a[i]的部分的状态转移方程为f(i, j) = max(f(i-1, k)+1)，不是f(i, k)+1
- k的范围是[1, j-1]
- DP的优化一般是对代码做等价变形，可以进一步优化掉一维
- 暴力做法

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 3030;

int n;
int a[N], b[N];     //a和b数组分别代表两个序列
int f[N][N];    //f[i][j]代表的集合：所有由a序列前i个字符，b序列前j个字符，且以b[j]
                //结尾的公共上升子序列长度的集合; 集合属性为求长度最大值

int main(void) {    //272. 最长公共上升子序列，暴力做法，可进一步优化
    scanf("%d", &n);
    for (int i=1; i<=n; i++) scanf("%d", &a[i]);
    for (int i=1; i<=n; i++) scanf("%d", &b[i]);

    //三重循环暴力做法，可进一步对代码优化降维
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++) {
            f[i][j] = f[i-1][j];    //不包含a[i]的情况

            if (a[i] == b[j]) {     //包含a[i]的情况，此时必有a[i] == b[j];
```

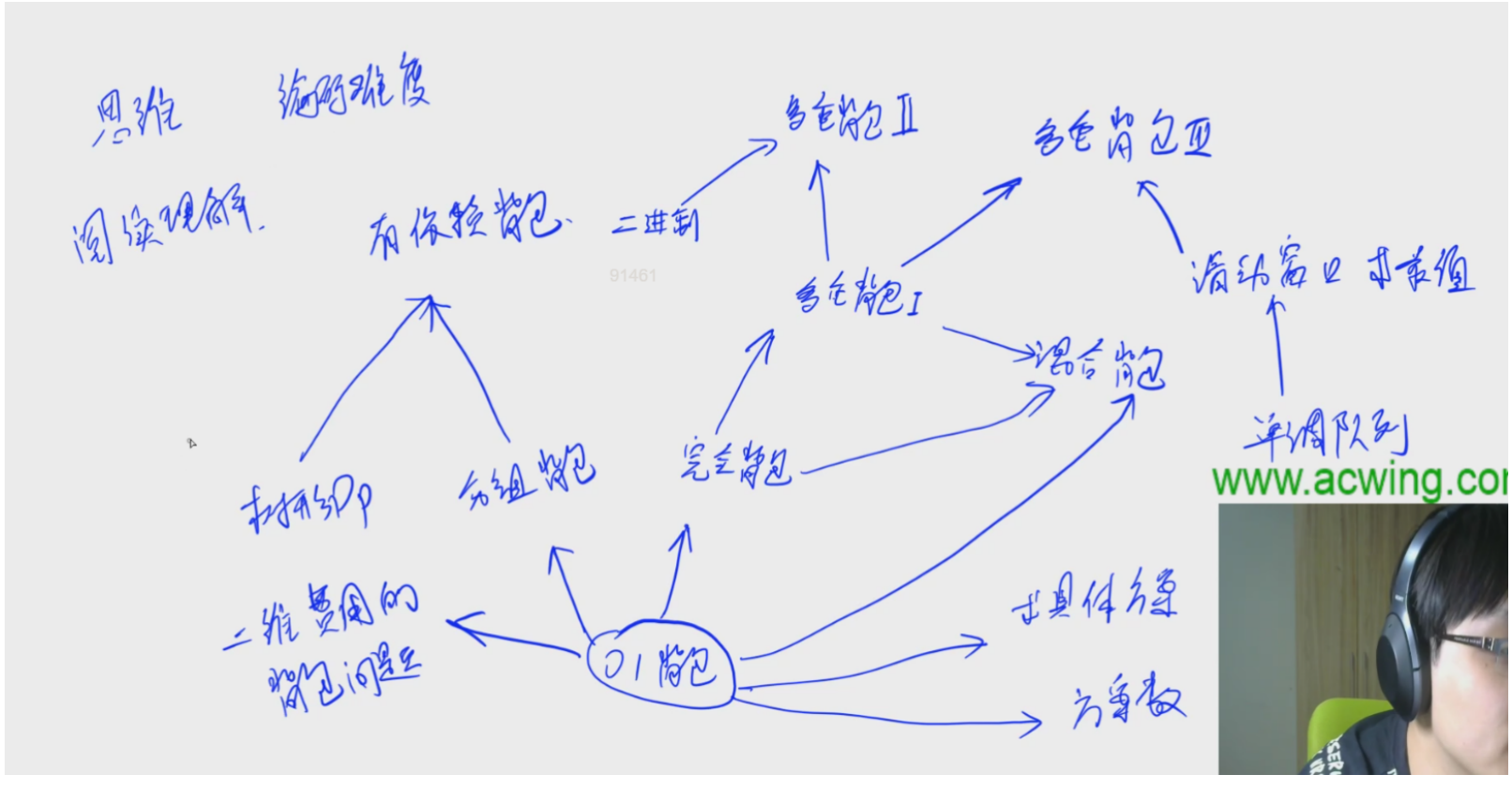
```
                f[i][j] = max(f[i][j], 1);  //LIS问题，倒数第二个字符为空，只有b[j]
                for (int k=1; k<j; k++) {
                    //满足条件时，求LIS问题的解，注意是f[i-1][k]+1
                    if (b[k] < b[j]) f[i][j] = max(f[i][j], f[i-1][k]+1);
                }
            }
        }

    //扫描以哪个b[j]结尾的公共上升子序列长度最大
    int res = 0;
    for (int i=1; i<=n; i++) res = max(res, f[n][i]);

    printf("%d", res);

    return 0;
}
```

- 优化做法

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 3030;

int n;
int a[N], b[N];     //a和b数组分别代表两个序列
int f[N][N];    //f[i][j]代表的集合：所有由a序列前i个字符，b序列前j个字符，且以b[j]
                //结尾的公共上升子序列长度的集合; 集合属性为求长度最大值

int main(void) {    //272. 最长公共上升子序列，优化做法，时间复杂度O(n^2)
    scanf("%d", &n);
    for (int i=1; i<=n; i++) scanf("%d", &a[i]);
    for (int i=1; i<=n; i++) scanf("%d", &b[i]);

    for (int i=1; i<=n; i++) {

        int maxv = 1;
        for (int j=1; j<=n; j++) {
            f[i][j] = f[i-1][j];

            if (b[j] < a[i]) maxv = max(maxv, f[i-1][j]+1); //求出f[i-1][k]+1的最大值
            if (a[i] == b[j]) f[i][j] = max(f[i][j], maxv); //a[i]==b[j]时更新f[i][j];

            // if (a[i] == b[j]) {
            //     f[i][j] = max(f[i][j], 1);
            //     for (int k=1; k<j; k++)
            //         if (b[k] < b[j]) //将b[j]换为a[i]后可以发现该循环可以和上层循环
                                        //一起完成，故提出maxv变量，减少一层循环
            //             f[i][j] = max(f[i][j], f[i-1][k]+1);
            // }
        }

    }

    int res = 0;
    for (int i=1; i<=n; i++) res = max(res, f[n][i]);

    printf("%d", res);

    return 0;
}
```

背包问题

背包问题题谱


思路
线性dp
阅读理解
有依赖背包
二进制
多重背包II
多重背包III
多重背包I
滑动窗口 求最值
混合背包
单调队列
手动实现
分组背包
完全背包
www.acwing.com
二维费用的背包问题
01背包
求具体方案
方案数