

CSP-J 第二轮国庆考前强化营 · 学生讲义

Day 3 动态规划：线性 DP · 背包 DP · 区间 DP

上午：全真模拟赛 3 | 下午：模拟赛精讲 + 本专题
极客少儿编程教研组 编 · 2026 年 10 月

一、今日学习目标

- 理解 DP 三要素：状态、转移方程、初始边界；理解「无后效性」的含义；
- 掌握线性 DP 的代表模型：数字三角形、LIS、LCS；
- 熟练掌握 01 背包与完全背包的一维写法，分清倒序与正序的原因；
- 了解区间 DP 的枚举顺序与环形处理方法（提高组衔接内容，重在理解思想）。

二、考点梳理

2.1 DP 的本质：用状态记录子问题的答案

一个问题能 DP，需要满足：最优子结构（大问题的最优解由小问题的最优解推出）与无后效性（状态一旦确定，之后的决策与「怎么到达这个状态」无关）。设计 DP 的三步走：

- 1. 定状态：dp[?] 表示什么（如 dp[i] = 考虑前 i 个物品时的最大价值）；
- 2. 写转移：从「最后一步做了什么决策」入手，写出递推式；
- 3. 定边界与顺序：初始值是多少；按什么顺序计算，保证转移时所需的状态已经算好。

考场技巧：DP 想不出递推时，先写记忆化搜索（DFS + 缓存），逻辑与递推等价，不容易错，多数 J 组数据范围下足够快。

2.2 线性 DP：三种代表模型

- 路径型（数字三角形）：dp[i][j] = a[i][j] + max(dp[i+1][j], dp[i+1][j+1])，从底往上推。
- 上升子序列（LIS）：dp[i] 表示以 a[i] 结尾的最长上升子序列长度， $O(n^2)$ ；进阶可用二分优化到 $O(n \log n)$ 。
- 公共子序列（LCS）：dp[i][j] 表示两串前 i、前 j 个字符的 LCS 长度；a[i]==b[j] 时 dp[i][j]=dp[i-1][j-1]+1，否则取左/上的最大值。

模板：LIS ($O(n \log n)$)，了解即可

```
// LIS 的  $O(n \log n)$  写法：dp[len] = 长度为 len 的上升子序列的最小结尾
int dp[N], len = 0;
for (int i = 1; i <= n; i++) {
    int p = lower_bound(dp + 1, dp + len + 1, a[i]) - dp; // 第一个 >= a[i]
    dp[p] = a[i];
    if (p > len) len = p;
}
// len 即为 LIS 长度
```

2.3 背包 DP：一句话分清倒序与正序

模型	约束	一维写法	原因
01 背包	每件物品最多选 1 次	容量从大到小（倒序）	防止同一物品被重复选
完全背包	每件物品无限次选	容量从小到大（正序）	允许同一物品被重复选
多重背包	每件物品限选 s_i 次	二进制拆分后按 01 背包做	把 s_i 拆成 1,2,4,... 的组合

模板：01 背包 vs 完全背包（唯一区别在容量循环方向）

```
for (int i = 1; i <= m; i++)           // 01 背包：逐个考虑物品
  for (int j = T; j >= t[i]; j--)      // 倒序!
    dp[j] = max(dp[j], dp[j - t[i]] + v[i]);

for (int i = 1; i <= m; i++)           // 完全背包：仅方向不同
  for (int j = t[i]; j <= T; j++)      // 正序!
    dp[j] = max(dp[j], dp[j - t[i]] + v[i]);
```

2.4 区间 DP：按「区间长度」从小到大枚举

区间 DP 的状态形如 $dp[l][r]$ （合并区间 $[l,r]$ 的最优值）。枚举顺序必须是长度从小到大：先算所有长度为 2 的区间，再算长度为 3 的……这样合并大区间时，所需的小区间都已算好。环形问题（如石子围成圈）的通用处理：把序列复制一倍接在后面，环形就变成线形，只取长度 $\leq n$ 的区间。

说明：区间 DP 属于提高组内容，J 组考试不作重点要求；本节课掌握其思想即可，学有余力者完成对应练习。

三、例题精讲

【例题1】数字三角形（线性 DP 入门） 洛谷 P1216

题目描述：一个数字三角形，从顶端出发，每步可走到下一行左下方或右下方的数，求到达底层的路径上数字之和的最大值。行数 $r \leq 1000$ 。

输入格式：第一行 r ；接下来 r 行构成三角形。

输出格式：一个整数：最大路径和。

样例输入	样例输出
5 7 3 8 8 1 0 2 7 4 4 4 5 2 6 5	30

思路分析： $dp[i][j]$ 表示「从 (i,j) 出发到底层」的最大和： $dp[i][j] = a[i][j] + \max(dp[i+1][j], dp[i+1][j+1])$ 。

从倒数第二行倒推回顶端，答案为 $dp[1][1]$ 。自底向上写，边界天然免费（底层就是 a 本身）。

$r=1000$ 时暴力枚举 2^r 条路径完全不可行——这就是 DP 存在的意义。

参考代码（C++）

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 1e3 + 10;
int T = 1;

int a[N][N], dp[N][N];

void solve() {
    int r;
    cin >> r;
    for (int i = 1; i <= r; i++)
        for (int j = 1; j <= i; j++) cin >> a[i][j];
    for (int j = 1; j <= r; j++) dp[r][j] = a[r][j]; // 边界：底层
    for (int i = r - 1; i >= 1; i--) // 自底向上
        for (int j = 1; j <= i; j++)
            dp[i][j] = a[i][j] + max(dp[i + 1][j], dp[i + 1][j + 1]);
    cout << dp[1][1] << '\n';
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    while (T--) solve();
    return 0;
}
```

易错点：

- 转移方向与依赖方向相反： $dp[i]$ 依赖 $dp[i+1]$ ，所以 i 从大到小枚举；
- 也可以顺推（ $dp[i][j]$ 表示到达 (i,j) 的最大和），两种写法都要看得懂；

- 样例最优路径 $7 \rightarrow 3 \rightarrow 8 \rightarrow 7 \rightarrow 5 = 30$ ，动手在图上画一遍验证。

【例题2】采药（01 背包）洛谷 P1048

题目描述：采药人总时间为 T (≤ 1000)，山里有 M (≤ 100) 株草药，采第 i 株耗时 t_i 、价值 v_i ，每株最多采一次。求能采到的最大总价值。

输入格式：第一行 T 、 M ；接下来 M 行每行 t_i 、 v_i 。

输出格式：一个整数：最大总价值。

样例输入	样例输出
70 3 71 100 69 1 1 2	3

思路分析：裸的 01 背包： $dp[j]$ = 总耗时不超过 j 时的最大价值。逐个物品决策「采或不采」。

一维写法容量必须倒序：倒序保证 $dp[j-t[i]]$ 还是「上一轮」的值，即第 i 株只被用一次。

样例：第 1 株耗时 $71 > 70$ 不可采；采第 2、3 株，耗时 $69+1=70$ ，价值 $1+2=3$ 。

参考代码（C++）

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 1e3 + 10;

int T, M, dp[N];

void solve() {
    cin >> T >> M;
    for (int i = 1; i <= M; i++) {
        int t, v;
        cin >> t >> v;
        for (int j = T; j >= t; j--) // 倒序：每件只用一次
            dp[j] = max(dp[j], dp[j - t] + v);
    }
    cout << dp[T] << '\n';
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    solve();
    return 0;
}
```

易错点：

- dp 初值全 0 即可（价值非负，「恰好装满」不要求）；
- 倒序写成 $j--$ 循环；若写成 $j++$ 就变成完全背包，样例会错；
- 读入一组处理一组，无需先存数组——背包可以「边读边做」。

【例题3】疯狂的采药（完全背包）洛谷 P1616

题目描述：与上题相同，但每株草药可以采任意多次。求总时间 T 内的最大总价值。

输入格式：第一行 T 、 M ；接下来 M 行每行 t_i 、 v_i 。

输出格式：一个整数：最大总价值。

样例输入	样例输出
10 3 3 5 4 6 5 9	18

思路分析：与 01 背包的唯一区别：物品可重复选 $\rightarrow$ 容量正序，让 $dp[j-t[i]]$ 允许已经装过第 i 株的状态参与转移。

样例最优：采两次第 3 株，耗时 10，价值 $9+9=18$ 。

数据较大时价值累加可能超 int ， dp 数组开 $long\ long$ 更保险。

参考代码（C++）

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 1e7 + 10;

int T, M, dp[N];

void solve() {
    cin >> T >> M;
    for (int i = 1; i <= M; i++) {
        int t, v;
        cin >> t >> v;
        for (int j = t; j <= T; j++) // 正序：可重复选
            dp[j] = max(dp[j], dp[j - t] + v);
    }
    cout << dp[T] << '\n';
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    solve();
    return 0;
}
```

易错点：

- 和 01 背包对照着背：倒序 = 每件一次，正序 = 每件无限；
- dp 数组大小按 T 的上限开，留意内存（10 个 $long\ long \approx 80\ MB$ ，边界情况要小心）；
- 价值大时务必 $long\ long$ ，输出对应 $\%lld$ 。

【例题4】石子合并（区间 DP · 提高组衔接）洛谷 P1880

题目描述： n 堆石子围成一圈（ $n \leq 100$ ），每次可将相邻两堆合并，得分等于合并后那一堆的石子数。求全部合并成一堆的总得分的最小值与最大值。

输入格式：第一行 n ；第二行 n 个整数（每堆石子数）。

输出格式：两行：最小得分、最大得分。

样例输入	样例输出
------	------

4	43
4 5 9 4	54

思路分析：环形 $\rightarrow$ 线形：序列复制一倍 ($a[n+1..2n] = a[1..n]$)，环形合并等价于在长度 $2n$ 的链上取长为 n 的区间。

状态： $dp[l][r]$ = 合并区间 $[l,r]$ 的最优得分；转移：枚举分割点 k ， $dp[l][r]$ = 最优($dp[l][k] + dp[k+1][r]$) + 区间和 $[l,r]$ 。区间和用前缀和 $O(1)$ 取得。

枚举顺序：按区间长度 $len = 2..n$

从小到大。最小值、最大值各做一遍（或用两个数组同时维护）。

参考代码 (C++)

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 205;
int T = 1;

int a[N], s[N], dp[N][N], dp2[N][N]; // dp 最小代价, dp2 最大代价

void solve() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        a[i + n] = a[i]; // 倍长破环
    }
    for (int i = 1; i <= 2 * n; i++) s[i] = s[i - 1] + a[i];
    memset(dp, 0x3f, sizeof(dp)); // 求最小: 先灌无穷大
    for (int i = 1; i <= 2 * n; i++) dp[i][i] = 0;
    for (int len = 2; len <= n; len++) // 按长度枚举
        for (int l = 1; l + len - 1 <= 2 * n; l++) {
            int r = l + len - 1;
            for (int k = l; k < r; k++) { // 枚举分割点
                int w = s[r] - s[l - 1]; // 区间和
                dp[l][r] = min(dp[l][r], dp[l][k] + dp[k + 1][r] + w);
                dp2[l][r] = max(dp2[l][r], dp2[l][k] + dp2[k + 1][r] + w);
            }
        }
    int amn = 0x3f3f3f3f, amx = 0;
    for (int l = 1; l <= n; l++) { // 所有长 n 的区间
        amn = min(amn, dp[l][l + n - 1]);
        amx = max(amx, dp2[l][l + n - 1]);
    }
    cout << amn << '\n' << amx << '\n';
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    while (T--) solve();
    return 0;
}
```

易错点：

- 忘记「倍长」直接按链做，环形样例就会错（本题样例 4 5 9 4 的最优解恰好利用了首尾相邻）；

- 区间和用前缀和取，不要在 k 循环里再套循环求和，否则多一个 $O(n)$ 会超时；
- mn 数组注意初始化极大值；最终答案只在长度恰好为 n 的区间里取。

四、今日易错点总结

- DP 先想「最后一步决策」，转移方程自然浮现；想不出就写记忆化搜索。
- 01 背包倒序、完全背包正序——考场写完后用样例验证一遍方向。
- 枚举顺序就是 DP 的「拓扑序」：线性 DP 看依赖方向，区间 DP 按长度。
- dp 数组初始化要符合语义：求最小填 INF，求最大填 0 或 -INF，边界单独赋值。
- 价值、得分累加默认考虑 long long。

五、课后练习（今晚完成，标注★为必做）

题号	题名	考点	难度
★ P1063	能量项链	环形区间 DP（与例 4 同构）	提高-
P1020	导弹拦截	LIS 两问 ($O(n^2)$ 即可入门)	普及+/提高-
P1439	最长公共子序列	LCS 模板 / 可转 LIS	普及+
P1776	宝物筛选	多重背包（二进制拆分 / 单调队列）	提高-

P1063、P1776 属提高组难度，尽力即可；其余两题要求独立 AC。