速通 CSP-J 动态规划：你到底需要什么？

上海洛谷网络科技有限公司
BIT,ICS
皎月半洒花

洛谷
www.luogu.com.cn

本课件将——

- 致力于弥补国内动态规划知识讲授的空缺
- 尝试让学生用正常的思维去理解并学习动态规划
- 完成国内生态下的竞赛学习任务
- 持续更新

当然，以上是画饼内容，毕竟上课前 12 小时我还没有开始做课件。

本课件面向的对象是“有一定DP基础的”同学。

洛谷

CONTENTS

你真的了解 DP 吗

DP = ?

你觉得“动态规划”是什么

- 推式子？
- 递推问题的复杂形式？
- 搜索的 Pro Max Plus 旗舰版？
- ~~题解里其他人是这么写的所以我就跟着抄了~~？

如果非要说的话——

- 动态规划，是去处理“状态”和“状态”的关系。——《正经教材》
- 动态规划非常常见，其生命力能从入门绵延到你 AKIOI。
- 动态规划，大家都认为很“难”。
- ……

但其实人生有时候并不要求我们一定要在每时每刻把每件事情都弄的完全明白。比如虽然你的人生进行了很多年了，但是你依然无法确定你人生的意义。动态规划也一样。

与其关心动态规划是什么，不如关心遇到动态规划怎么办。

遇到简单动态规划问题的合理步骤

- 请严格执行下列步骤：
- Step1：判断这道题贪心的合法性。
- Step2：判断这道题是不是可以直接二分答案。

暴力枚举
循环转化为
二分枚举

- （如果不是最优化问题，则忽略以上两步）
- Step3：好的。现在你不得不考虑 DP 了。不妨先想想有没有做过类似的题目？尝试和题目本身建立一种超自然联系。
- Step4：尝试设计状态。一开始可以设计的很复杂。
- Step5：尝试设计转移的方程。
- Step6：如果发现转移的很奇怪或者无法转移，则返回 Step4 或 5。
- Step7：检查你的程序，并看看有没有能优化和合并的地方。

进一步的说明

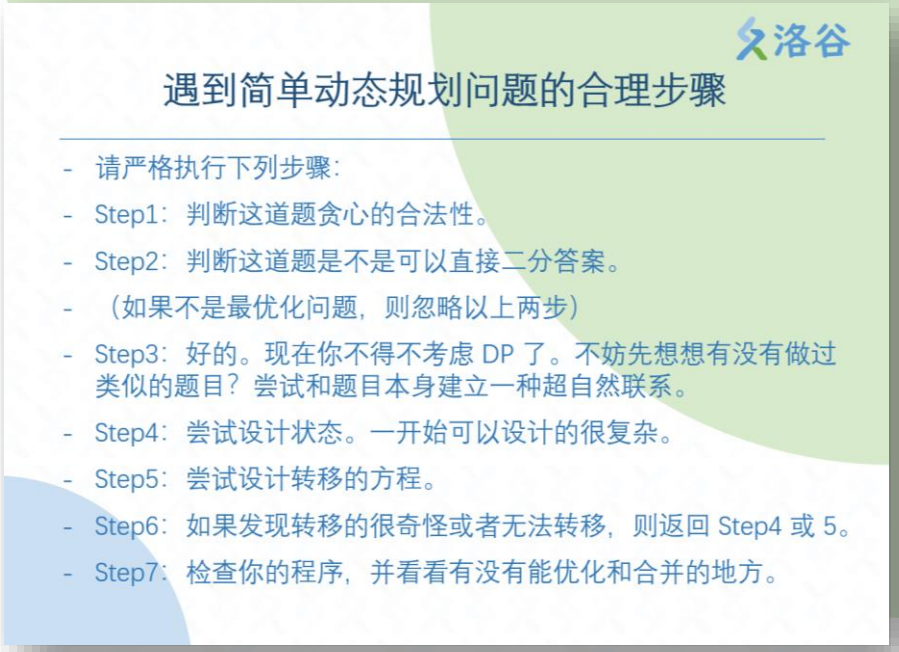
- 第一二步是不可忽略的。
- 第三步显然依赖于你的视野。
- 第四步和第五步则依赖于能力了。

对于每一步，其实都需要学习很多。

这节课我们显然不讲 Step 1 & 2，

会着重讲 4,5 并提到一些 7。

Step 3 会在结语里提到。



到底该如何学 DP?

DP is bad.

洛谷

我有一计：

多做题，多整理。

尤其不能抄题解/不懂装懂/为了做题而做题…

我可以这么跟你讲，

其他的题你抄题解至少是非负收益，DP 抄题解一定是负收益。

如果你是我的学生，我会让你：

1. 自学《深入浅出·进阶篇》的第十三、十四十五章并完成课后题。
2. 自学《信息学竞赛一本通》上动态规划全部内容并完成课后题。
3. 一个月内做完洛谷上全部颜色在“黄色”之下的题目。
4. 学一部分知识，做一部分题目，就要整理一部分。

（当然，整理不是让你写题解，因为题解过于细致而浪费时间；你需要整理到“自己至少一周后还能看懂”为止）

勤能补拙，勤能补拙！

接下来我们进入正经知识环节…

线性动态规划

思路 and 优化 and 发现性质

线性动态规划

- 什么是“线性动态规划”？
- 可以理解为状态是线性或者嵌套线性的。
- 比如 f[][] 或者 dp[][][][][]…但是转移很清晰。
- 就好像是，你把一张纸揉成一个纸团，看似很复杂。
- 但你把纸团铺平展开后，上面的花纹非常清晰。
- 总之，线性 DP 因为很简单，所以会很难。

传球游戏

例一 https://www.luogu.com.cn/problem/P5888

场上的 n 个球员围成一圈，编号从 1 到 n，刚开始球在 1 号球员手中。一共 m 次传球，每次传球必须传给一个人，但不能传到自己手中。求第 m 次传球以后传回 1 号球员的方案数。

但他觉得这个问题太简单了，于是加了 k 条限制，每条限制形如 a, b，表示 a 号球员不能将球传给 b 号球员。

为了使得 oql 的注意力转移回球场上，你需要在最短的时间内告诉他这个方案数是多少。

你只需要告诉他答案对 998244353 取模后的结果。

说明/提示

对于 10% 的数据，$k = 0$。

对于另外 15% 的数据，$n \leq 500$。

对于另外 20% 的数据，$n \leq 5 \times 10^4$。

对于另外 20% 的数据，$k \leq 300$。

对于 100% 的数据，$1 \leq n \leq 10^9$，$0 \leq m \leq 200$，$0 \leq k \leq \min(n \times (n-1), 5 \times 10^4)$，$1 \leq a_i, b_i \leq n$，**不保证** a_i, b_i **不相等**。

（这题难度高于 CSP-J）

15 pts 做法

- 为什么先从 15 分开始考虑?
- 因为其实我们观察到 n 的大小很关键。
- Part2 的 $15pts$ 正是 n 最小的。
- 想一想，如何设计状态呢?

对于另外 15% 的数据，n≤500。

，0≤m≤200，0≤k≤min(n×(n−1),5×10^4)，1≤ai,bi≤n，

15 pts 做法

- 一个比较自然的想法是，设 $f_{i,j}$ 表示球传完了第 i 轮，球传到第 j 个人手里的方案数。
- 初始化 $f_{0,1}=1$，答案保存在 $f_{m,1}$ 里。
- 转移比较自然了。$f_{i,j}=\sum_{k=1}^{n} f_{i-1,k} \times ok(k,j)$。
- 其中 $ok(k,j)$ 表示 k 能否传给 j。包括自己不能传给自己。
- 复杂度 $O(n^2m)$。

对于另外 15% 的数据，n≤500。

，0≤m≤200，0≤k≤min(n×(n−1),5×10^4)，1≤ai,bi≤n，

35 pts 做法

- 但不难发现总限制至多只有 **50000** 条。
- 但如果要挨个考虑能不能传，n^2 可以达到 2.5×10^9。
- 所以我们为何不**反着考虑**，去算那些不能传到 j 的 k 呢？

对于另外 15% 的数据，n≤500。
对于另外 20% 的数据，n≤5×10^4。
0≤m≤200，0≤k≤min(n×(n−1),5×10^4)，1≤ai,bi≤n

35 pts 做法

- 具体的，我们令 $s_i = \sum f_{i,j}$,即第 i 轮传到所有人的方案数总和。
- 那么原式可以写作 $f_{i,j} = s_{i-1} - \sum \text{not\_ok}(k,j) \times f_{i-1,k}$。
- 但其实并不需要遍历全部的 n 个人。
- 我们只需要在读入时，用个 $vector < int >$ 或者 map 去记录一下不能到达 i 的 k 都是谁，转移时直接挑 k 出来就好了。
- 不难发现，这样实现时内层循环的复杂度均摊是 $(n+k)$ 的，因为两者不是嵌套关系。
- 复杂度 $O(nm+mk)$。

对于另外 15% 的数据，n≤500。
对于另外 20% 的数据，n≤5×10^4。
0≤m≤200，0≤k≤min(n×(n−1),5×10^4)，1≤ai,bi≤n

35 pts 代码

对于另外 15% 的数据，n≤500。
对于另外 20% 的数据，n≤5×10^4。 0≤m≤200，0≤k≤min(n×(n−1),5×10^4)，1≤ai,bi≤n

```
int s[N] ;
int f[M][N] ;
int n, m, k ;
map <int, int> ok[N] ;

int main(){
    cin >> n >> m >> k ;
    for (int u, v, i = 1 ; i <= k ; ++ i)
        cin >> u >> v, ok[v][u] = 1 ;
    for (int i = 1 ; i <= n ; ++ i) ok[i][i] = 1 ;
    f[0][1] = s[0] = 1 ;
    for (int i = 1 ; i <= m ; ++ i){
        for (int j = 1 ; j <= n ; ++ j){
            f[i][j] = s[i - 1] ;
            for (auto t : ok[j])
                (f[i][j] += P - f[i - 1][t.first]) %= P ;
            (s[i] += f[i][j]) %= P ;
        }
    }
    cout << f[m][1] ;
    return 0 ;
}
```

注意一个模运算小技巧：

$(-x) = (P - x) \bmod P$。

所以我们减去 x 可以换做是 $+P-x$，不用处理负数了。

10 pts 做法

- $k = 0$ 时没有限制。 对于 10% 的数据，k=0。
- 但其实你很难得出一个“闭形式”，也即一个公式解决问题的形式。
- 这时有两个思路：骗分和思考性质。
- 对于骗分……
- 我们可以通过 $35pts$ 的程序直接打表，然后肉眼找规律获得结果。
- 当然这其实有点依赖于选手的数学素养和运气。
- 理论上是可行的，因为这题 $k = 0$ 时形式并不复杂。

10 pts 做法

对于 10% 的数据，k=0。

- 如果你选择了思考性质，那就会离正解近一点。
- 我们考虑，如果没有限制，那么本题就可以把所有除你自己——
- 也就是除一号点之外的所有点看做“外人”。
- 也就是，1 号点无论传球给谁，情况到最后都会是完全相同。
- 所以我们把除 1 号点之外的人视作一个人，称为“另一个人”。

10 pts 做法

对于 10% 的数据，k=0。

- 继续 dp。
- 设 f_i 表示第 i 轮传回 1 的方案数，g_i 表示第 i 轮后传回另一个人的方案数。
- 那么有
- $f_i = g_{i-1} \times (n-1)$。
- $g_i = f_{i-1} \times (n-1) + g_{i-1} \times (n-2)$。
- 复杂度 $O(m)$。

65 pts 做法

- 我们考虑借助 $10pts$ 的想法。
- 如果一些人他们没有被限制，也就是如果限制中没有提及某个人，那他就可以随便传给任何一个人、也可以被任何一个人传到。
- 所以这些人本质上也是等价的。
- 我们借助 $10pts$ 的思路，将这部分“其他人”缩成一个人。那么其实一共只有至多 $(2 \times k + 1)$ 个本质不同的人，即“关键点”。
- 因此我们再对这部分 dp 的话，直接 dp 部分的时间复杂度和 $15pts$ 的类似，只不过换成了 $O(k^2 m)$，可以通过 65 分。

对于 10% 的数据，k=0。
对于另外 15% 的数据，n≤500。
对于另外 20% 的数据，n≤5×10^4。
对于另外 20% 的数据，k≤300。

$0 \leqslant m \leqslant 200$,

95 pts 做法

- 注意到，我们其实在上一步中的优化只有“缩点”。
- 这一步并不会影响到我们的算法结构，只会影响部分系数。
- 所以可以使用和 $Part\ 2$ 相同的优化措施：前缀和。
- 思路不再赘述，具体细节留作课下作业。
- 这样就可以比较轻易地获得 95 分。

对于 100% 的数据，1≤n≤10^9，0≤m≤200，
0≤k≤min(n×(n−1),5×10^4)，1≤ai,bi≤n，

95 pts 代码细节

预处理：

```
for (int u, v, i = 1 ; i <= k ; ++ i){
    cin >> lm[i][0] >> lm[i][1] ;
    truman[++ tot] = lm[i][0] ;
    truman[++ tot] = lm[i][1] ;//找出有用的人
}
sort(truman + 1, truman + tot + 1) ; //离散化
tot = unique(truman + 1, truman + tot + 1) - (truman + 1)
if (truman[1] != 1){
    //有个细节，就是有可能 1 不是关键点
    //但无所谓，我们强制让 1 是关键点
    //因为多几个少几个关键点不影响正确性
    //常数个关键点也不影响时间表现。
    for (int i = tot ; i >= 1 ; -- i)
        truman[i + 1] = truman[i] ;
    truman[1] = 1 ;
    ++ tot ;
}
for (int i = 1 ; i <= tot ; ++ i) ok[i][i] = 1 ;
for (int i = 1 ; i <= tot ; ++ i) tr[truman[i]] = i ;
for (int i = 1 ; i <= k ; ++ i){
    ok[ tr[ lm[i][1] ] ][ tr[ lm[i][0] ] ] = 1 ;
}//重新预处理限制
```

DP 部分：

```
f[0][1] = s[0] = 1 ;
for (int i = 1 ; i <= m ; ++ i){
    //注意 dp 的系数问题
    for (int j = 1 ; j < tot ; ++ j){
        f[i][j] = s[i - 1] ;
        for (auto t : ok[j])
            (f[i][j] += P - f[i - 1][t.first]) %= P ;
        (f[i][j] += 1ll * (n - tot + 1) * f[i - 1][tot] % P) %= P ;
        (s[i] += f[i][j]) %= P ;
    }
    f[i][tot] = (s[i - 1] + 1ll * (n - tot) * f[i - 1][tot] % P) % P ;
}
cout << f[m][1] ;
```

对于 100% 的数据，$1 \leqslant n \leqslant 10^9$，$0 \leqslant m \leqslant 200$，
$0 \leqslant k \leqslant \min(n \times (n-1), 5 \times 10^4)$，$1 \leqslant a_i, b_i \leqslant n$，

100 pts 做法

- 注意一下你的常数。
- 我们使用的 map 其实会多添一只 $\log$ 。
- 怎么卡呢?
- （1）换用 $vector$。
- （2）减少取模。这其实就是基本功了。

[CSP-J2020] 方格取数

例二 https://www.luogu.com.cn/problem/P7074

$n \times m$ 的方格图，每个方格中都有一个整数。现有一只小熊，想从图的左上角走到右下角，每一步只能向上、向下或向右走一格，并且不能重复经过已经走过的方格，也不能走出边界。小熊会取走所有经过的方格中的整数，求它能取到的整数之和的最大值。

$1 \leq n, m \leq 10^3, 1 \leq a_{i,j} \leq 10^4$。

如果不能向上走，会很简单吧！

- Very very Obviously. 经典咏流传之《数字三角形》。
- 那如果能向上走，会有什么影响吗？
- 显然，DP 要求我们转移的时候要按照正确的“顺序”。
- “顺序”可以怎么理解呢？
- 我们需要知道，DP 的流程图一定是一张 DAG（有向无环图）。
- 你每走一步，都要保证之前走过的路是正确的。
- 而如果直接设 $f_{i,j}$ 表示走到 (i,j) 号点的最大值，每次三个方向转移，肯定是不对的！
- 想想为啥

插一嘴：关于 DP “顺序”的典中典问题

- 典中典之“01背包第二维的枚举顺序”。
- 为什么要倒序枚举？
- 肯定有人不知道为什么（笑）。

回到 [CSP-J2020]方格取数

- 我们发现不靠谱了！那能怎么办呢？
- DP 中非常常用的方式：**增维**。
- 换句话讲，我们刚才发现不能直接转移，是因为状态之间成为了环形。但是如果我们像高架桥那样给他们分层呢？
- 换句话说，增加维度的意义是**区分不易于区分的几种状态**。

增加维度

- 设 $f[i][j][0/1]$ 表示走到了 (i,j) 这个点，但是趋势上是从上方还是下方转移来的最优解。
- Q：什么叫“趋势上”？
- A：意思是两部分都可以包含从左边转移来的 f。
- Q：为什么都包含？
- A：因为我们不需要区分是否从左边来。
- 上方和下方需要区分是因为“环”的形成源自于“一下一上”的走法。

Step 4 -> Step 5： 构造转移

- 状态是 $f[i][j][0/1]$ 。
- 表示走到了 (i,j) 这个点，趋势上从上方/下方转移来的最优解。
- 如何转移?
- 记忆化搜索很好写！但前提是你是搜索小达人。
- 记忆化搜索好就好在，不需要费脑子构造转移。
- 那如果必须用迭代式呢?
- 先考虑简单的：从左边转移来的。对于俩情况都长得一样
- $f[i][j][0] = \max(f[i][j-1][0], f[i][j-1][1]) + a[i][j]$
- $f[i][j][1] = \max(f[i][j-1][0], f[i][j-1][1]) + a[i][j]$

Step 4 -> Step 5： 构造转移

- 那对于从下方转移来的呢?
- 很显然额外有：
- $f[i][j][1] = \max(f[i][j][1], f[i+1][j][1] + a[i][j])$
- 那对于上方来的额外有:
- $f[i][j][0] = \max(f[i][j][0], f[i-1][j][0] + a[i][j])$
- 非常显然，转移的时候要先枚举 j 后枚举 i。想一想，为什么?

背包问题

一类简单且复杂、广泛且深刻的问题。

背包基础知识大抽查

基础知识大抽查！！！

以下的背包你会不会呢？

- 01 背包
- 完全背包
- 多重背包以及二进制分组
- 分组背包
- 多维体积的背包

[NOIP2006 提高组] 金明的预算方案（强化版）

例三 （原版的题目链接）
https://www.luogu.com.cn/problem/P1064

简化题面：

01背包，但物品间存在某种“依赖”的关系。

i 依赖于 j 表示若选物品 i，则必须选物品 j。没有某个物品既依赖于别的物品，又被别的物品所依赖；另外，没有某件物品同时依赖多件物品。

求最大价值。

原版问题比较简单，只会有 < 3 个附件。我们在此考虑一下可能有很多附件的情况。

一种直接的想法

我很会抽象！我可以把决策抽象成物品。

- 比如主件为 o，附件为 a,b,c,d,e。
- 那么我可以考虑把每种选择抽象成“物品”。
  - 我只能选{o},{o,a}, {o,b}, {o,c}, {o,d}, {o,e}, {o,a,b}, {o,a,c}...{o,a,b,c,d,e}这几个方案之一。
  - 它们是互斥的。
  - 所以我可以对着这些决策进行背包。
- 设 k 为平均的附件数目。
- 最后的复杂度显然会多一个 2 的 k 次方。
- 这道题这么写似乎能过，但是如果附件数目很多呢？

你需要变得更抽象

我们不妨进一步抽象。

- 我们发现上一步里最慢的点在于我们做了一次子集枚举。
- 而这个子集枚举的过程，目的何在呢？
- 是不是在一定体积下，选出一定量的物品，得到最大价值呢？
- 那这个过程，是不是 01 背包呢？
- 答案是肯定的。

预处理：背包套背包

- 所以我们在一开始可以预处理出每个主附件集合的背包。
- 我们会得到某个集合在体积为 0,1,2...V-w[i] 体积下的最优解。
- 那这就相当于体积分别为 0,1,2...V-w[i] 的物品。
- 然后对这些物品一起背包就好了。

（w[i] 是主件的体积）

For more information: https://zhuanlan.zhihu.com/p/139368825

一道简单题

例四 https://www.luogu.com.cn/problem/U249233

简化题面：

01背包，但某一件物品可以不用付钱。

也即你有一次机会，可以选择把一件物品的体积变成 0。

提示

你当然可以想出很多种奇怪的贪心来骗分~

贪心+暴力的骗分组合拳：

大范围贪心(排序)，小范围暴力(选取可能优的前10项零元购之后暴力)，还有时间就随机化(看天)。

但是肯定都不太对，自己举举反例？

我们回想一下之前提到过的技巧：**“增维”**。

看看这道题能不能应用一下呢？

[NOIP2018 提高组] 货币系统

例五 https://www.luogu.com.cn/problem/P5020

在网友的国度中共有 n 种不同面额的货币，第 i 种货币的面额为 $a[i]$，你可以假设每一种货币都有无穷多张。为了方便，我们把货币种数为 n、面额数组为 $a[1..n]$ 的货币系统记作 (n, a)。

在一个完善的货币系统中，每一个非负整数的金额 x 都应该可以被表示出，即对每一个非负整数 x，都存在 n 个非负整数 $t[i]$ 满足 $a[i] \times t[i]$ 的和为 x。然而，在网友的国度中，货币系统可能是不完善的，即可能存在金额 x 不能被该货币系统表示出。例如在货币系统 $n = 3, a = [2, 5, 9]$ 中，金额 $1, 3$ 就无法被表示出来。

两个货币系统 (n, a) 和 (m, b) 是等价的，当且仅当对于任意非负整数 x，它要么均可以被两个货币系统表出，要么不能被其中任何一个表出。

现在网友们打算简化一下货币系统。他们希望找到一个货币系统 (m, b)，满足 (m, b) 与原来的货币系统 (n, a) 等价，且 m 尽可能的小。他们希望你来协助完成这个艰巨的任务：找到最小的 m。

测试点	n	a_i	测试点	n	a_i
1	$= 2$	≤ 1000	11	≤ 13	≤ 16
2			12		
3			13		
4	$= 3$		14	≤ 25	≤ 40
5			15		
6			16		
7	$= 4$		17	≤ 100	≤ 25000
8			18		
9	$= 5$		19		
10			20		

懒得做 PPT 了

讲这题的目的，是因为这题跟背包看上去很无关。

这就是背包抽象出来的新问题。

本质上依旧是背包。

The End