动态规划引论

Introduction to [Dynamic Programming]

Ruan Xingzhi

洛谷网校

2020 年 1 月 21 日

前言

这是这份课程的 2.0 版本（实验版本）。经典版本的课件在课后会一同下发，作为自学参考。

Intro

动态规划（DP）不是某种具体算法，而是一种**思想**。

核心在于：把大问题转化为小问题，利用小问题的解推断出大问题的解。

带着这种思想，我们来学习 DP.

上楼梯

今有 n 阶台阶。初始时站在 0 级，每次可以向上走 1 级或 2 级。问方案总数？

上楼梯

今有 n 阶台阶。初始时站在 0 级，每次可以向上走 1 级或 2 级。问方案总数？

暴力模拟：指数级复杂度。

上楼梯

今有 n 阶台阶。初始时站在 0 级，每次可以向上走 1 级或 2 级。问方案总数？

暴力模拟：指数级复杂度。

考虑更优的算法。如果以 $f[x]$ 表示“从 0 级走到 x 级的方案数”，假设 $f[1], f[2]...f[n-1]$ 全都已知，如何利用这些信息推出 $f[n]$？

上楼梯

走到 $f[n]$，要么是从 $n-1$ 级走上来的，要么是从 $n-2$ 级来的。依据加法原理

$$f[n]=f[n-1]+f[n-2]$$

这就是这个问题的递推式。

硬币问题

今天你手上有无限的面值为 1,5,11 元的硬币。
给定 n，问：至少用多少枚硬币，可以恰好凑出 n 元？

例

- $n = 15$ 时答案是 3，构造方法为 5+5+5
- $n = 12$ 时答案是 2，构造方法为 11+1

硬币问题

用 $f[x]$ 记录“凑出 x 元所需要的硬币数”。那么答案显然就是 $f[n]$.
如何求出 f 数组呢？$f[x]$ 等于什么？

提示

注意思考“$f[x]$ 从哪里来”。

硬币问题

考虑一个具体的例子：凑出 15 元。
为了凑出 15 元，我们最开始的时候，可以使用哪枚硬币？

- 假设用了 1 元硬币，那么接下来要凑出 14 元。共 1+4=5 枚
- 假设用了 5 元硬币，那么接下来要凑出 10 元。共 1+2=3 枚
- 假设用了 11 元硬币，那么接下来要凑出 4 元。共 1+4=5 枚

这三种方案，当然是选代价最低的，所以我们在这一次决策中，选择了 5 元硬币。

硬币问题

现在再来看，$f[x]$ 是 “凑出 x 元需要的硬币数”，它等于什么？

可供选择的决策方案如下：

- 先用一个 1 元硬币，代价 $1 + f[x-1]$
- 先用一个 5 元硬币，代价 $1 + f[x-5]$
- 先用一个 11 元硬币，代价 $1 + f[x-11]$

在上述方案里面，选择代价最低的就行！

硬币问题

所以有

$$f[x] = \min \begin{cases} 1 + f[x-1] \\ 1 + f[x-5] \\ 1 + f[x-11] \end{cases}$$

初步总结：状态

我们用“**大事化小，小事化了**”的思想，解决了上楼梯问题和硬币问题。大事能转化成小事，是因为大事和小事都有一样的**形式**：

上楼梯问题

大问题：爬上 n 级有多少种方案
小问题：爬上 $n-1$ 级有多少种方案、爬上 $n-2$ 级有多少种方案
它们都是“**爬上 $\times\times$ 级有多少种方案**”这一类问题。

硬币问题

大问题：凑出 n 元钱的最少硬币数
小问题：凑出 $n-1, n-5, n-11$ 元的最少硬币数
它们都是“**凑出 $\times\times$ 元钱所需最少硬币数**”这一类问题。

初步总结：状态

可见，只有大问题和小问题拥有**相同的形式**，才能考虑大事化小。如果满足这个要求，那么我们遇到的每个问题，都可以很简洁地表达。我们把可能遇到的每种“**局面**”称为状态。

例

硬币问题中，要表达“我们需要凑出 n 元钱”这个局面，可以设计状态：“$f[x]$ 表示凑出 x 元用的最少硬币数”。
上楼梯问题中，设计状态：“$f[x]$ 表示走上 x 级的方案数”。

设计完状态之后，只要能**利用小状态的解求出大状态的解**，就可以动手把题目做出来！

LIS 问题

数组的”最长上升子序列“是指：最长的那一个单调上升的子序列。
例如：数组 a：[1,3,4,2,7,6,8,5] 的最长上升子序列是 1,3,4,7,8.
如何求数组的最长上升子序列的**长度**？

LIS 问题

想用大事化小来做这道题，必须先设计状态。
如何设计状态，来完整地描述当前遇到的局面？

LIS 问题

想用大事化小来做这道题，必须先设计状态。
如何设计状态，来完整地描述当前遇到的局面？

设计状态

以 $f[x]$ 表示"以 $a[x]$ 结尾的上升子序列，最长有多长"！
那么，答案就是 $f[1], f[2]...f[n]$ 里面的最大值。

问题来了，如何求出 f 数组？提示：思考 $f[x]$ **从哪里来**。

求出 f 数组

$f[x]$ 表达的是“以 $a[x]$ 结尾的最长的上升子序列长度”。这个最长的子序列，一定是把 $a[x]$ 接在某个上升子序列尾部形成的！

例

数组 a:[1,3,4,7,2,6,8,**5**]
考虑 f[8]，它的来源有：

- 自己一个元素作为一个序列。长度为 1.
- 接在 a[1] 后面。长度为 $f[1]+1=2$
- 接在 a[2] 后面。长度为 $f[2]+1=3$
- 接在 a[3] 后面。长度为 $f[3]+1=4$
- 接在 a[5] 后面。长度为 $f[5]+1=3$

求出 f 数组

此时，稍有常识的人都会看出，要得到 $f[x]$，只需要看 $a[x]$ 能接在哪些数的后面。
也就是：

$$f[x] = \max_{p<x, a[p]<a[x]} \{f[p] + 1\}$$

其中 $p < x, a[p] < a[x]$ 的含义是：枚举在 x 前面的，$a[p]$ 又比 $a[x]$ 小的那些 p. 因为 $a[x]$ 可以接到这些数的后面，形成一个更长的上升子序列。

初步总结：转移

在前面三个例题中，我们都是先设计好状态，然后给出了一套用小状态推出大状态解的方法。

从一个状态的解，得知另一个状态的解，我们称之为“**状态转移**”。这个转移式子称为“状态转移方程”。

例

硬币问题中，状态转移方程是：

$$f[x] = 1 + \min\{f[x-1], f[x-5], f[x-11]\}$$

LIS 问题中，状态转移方程是：

$$f[x] = \max_{p<x, a[p]<a[x]} \{f[p] + 1\}$$

小结：状态和转移

总结刚刚学习的内容。如果我们想用大事化小的思想解决一个问题，我们需要：

1. 设计状态。把面临的每一个问题，用状态表达出来。
2. 设计转移。写出状态转移方程，从而利用小问题的解推出大问题的解。

设计转移

前三个问题中，我们设计转移的时候，考虑的都是“这个局面是从哪过来的”。
这是一种常见的思路：当前状态的解未知。需要用已经解决的状态，来推出当前状态的解。

设计转移

前三个问题中，我们设计转移的时候，考虑的都是“这个局面是从哪过来的”。
这是一种常见的思路：当前状态的解未知。需要用已经解决的状态，来推出当前状态的解。

DP 还有另一种设计转移的思路：当前状态的解已知。需要利用这个解，去更新它能走到的状态。

设计转移

前三个问题中，我们设计转移的时候，考虑的都是“这个局面是从哪过来的”。
这是一种常见的思路：当前状态的解未知。需要用已经解决的状态，来推出当前状态的解。

DP 还有另一种设计转移的思路：当前状态的解已知。需要利用这个解，去更新它能走到的状态。

这两种思路，一种是考虑“我从哪里来”，一种是考虑“我到哪里去”。两种手段都是能解决问题的！

上楼梯问题再讨论

如何用“我到哪里去”的转移手段，解决上楼梯问题？

上楼梯问题再讨论

如何用“我到哪里去”的转移手段，解决上楼梯问题？

$$
\begin{aligned}
f[x] &\rightarrow f[x+1] \\
&\rightarrow f[x+2]
\end{aligned}
$$

代码实现不难。

硬币问题再讨论

如何用“我到哪里去”的转移手段，解决硬币问题？

硬币问题再讨论

如何用“我到哪里去”的转移手段，解决硬币问题？

$$\begin{aligned} f[x] &\rightarrow f[x+1] \\ &\rightarrow f[x+5] \\ &\rightarrow f[x+11] \end{aligned}$$

LIS 问题再讨论

如何用“我到哪里去”的转移手段，解决 LIS 问题？

LIS 问题再讨论

如何用“我到哪里去”的转移手段，解决 LIS 问题？

$$f[x] \to f[p]$$

其中 $p > x, a[p] > a[x]$.

小结：设计转移

设计转移有两种方法。

- pull 型（我从哪里来）：对于一个没有求出解的状态，利用能走到它的状态，来得出它的解。
- push 型（我到哪里去）：对于一个已经求好了解的状态，拿去更新它能走到的状态。

DP 三连

综上所述，如果您想用 DP 解决一个问题，要干的事情可以总结为 DP 三连：

- 我是谁？（如何设计状态）
- 我从哪里来？（pull 型转移）
- 我到哪里去？（push 型转移）

两种转移方式中，只需要选择一个来设计转移即可。

斐波那契数列

众所周知，斐波那契数列是

$$F[1] = 1$$
$$F[2] = 1$$
$$F[n] = F[n-2] + F[n-1]$$

假设严格按照定义，写一个递归的代码，复杂度是什么情况？

斐波那契数列

朴素代码如下：

```
int fib(int n)
{
	if(n==1 || n==2) return 1;
	return fib(n-2) + fib(n-1);
}
```

请估算时间复杂度。

斐波那契数列

我们遇到的最大的麻烦，是很多 fib 值被重新计算了。

假设现在在计算 fib(7)，明明 fib(5) 只需要计算一次就可以；但是 fib(7) 要调用 fib(6) 和 fib(5)，fib(6) 要调用 fib(5)，所以 fib(5) 莫名其妙被调用了两次。

如何避免这种情况？

记忆化

我们引入记忆化:

记忆化搜索

调用 fun(x) 时:

- 如果 fun(x) 没有被计算过，则计算 fun(x)，并存储到 mem[x]
- 如果 fun(x) 被计算过，则直接返回 mem[x]

记忆化搜索的复杂度如何？

又谈硬币问题

如何用记忆化搜索写出硬币问题？
采用哪种转移方式最方便？

小结：记忆化搜索

- **按顺序递推**和**记忆化搜索**，是 DP 的两种高效实现方式。
- 记忆化搜索一般配套“我从哪里来”的转移方式。

记忆化搜索的优势

- 如果转移顺序不太好确定，则记忆化搜索可以帮你省一堆事。
- 有时候，记忆化搜索更节省时间、空间。因为不可能达到的状态是不会被搜索到的。

Function

https://www.luogu.com.cn/problem/P1464

Function

https://www.luogu.com.cn/problem/P1464
记忆化搜索模板题。建议做一做。

魔法师

您面前有一个苹果。作为一位魔法师，您可以施法，搞出如下效果之一：

- 让苹果变多一个。
- 让苹果变成两倍数量。
- 让苹果变成三倍数量。

给定 n，问您：至少需要施法多少次，才能得到**恰好** n 个苹果。

DP 三连

我是谁

设计状态：dp[x] 表示“搞出 x 个苹果需要的最小施法次数”。

DP 三连

我是谁

设计状态：dp[x] 表示“搞出 x 个苹果需要的最小施法次数”。

我从哪里来

dp[x] 可以由以下状态推过来：

- dp[x-1]+1
- dp[x/2]+1，如果 x 是 2 的倍数
- dp[x/3]+1，如果 x 是 3 的倍数

DP 三连

我是谁

设计状态：dp[x] 表示“搞出 x 个苹果需要的最小施法次数”。

我从哪里来

dp[x] 可以由以下状态推过来：

- dp[x-1]+1
- dp[x/2]+1，如果 x 是 2 的倍数
- dp[x/3]+1，如果 x 是 3 的倍数

我到哪里去

dp[x] 去更新 dp[x+1], dp[2x], dp[3x].

数字三角形

https://www.luogu.com.cn/problem/P1216

DP 三连

我是谁

dp[x][y] 表示“从起点走到第 x 行第 y 个点的最大价值”。

我从哪里来

dp[x][y] 源于 dp[x-1][y-1] 和 dp[x-1][y].

$$dp[x][y] = w[x][y] + \max \begin{cases} dp[x-1][y-1] \\ dp[x-1][y] \end{cases}$$

合唱队形

https://www.luogu.com.cn/problem/P1091

合唱队形

https://www.luogu.com.cn/problem/P1091

枚举站在中间的人。考虑以 $a[x]$ 为中心的合唱队伍：
他的左边是一个**尽量长的**上升子序列，右边是一个**尽量长的**下降子序列。
对于每个人，算出如果以他为中心，队伍将会有多长。取最长的即可。

乌龟棋

https://www.luogu.com.cn/problem/P1541

乌龟棋

https://www.luogu.com.cn/problem/P1541

如何设计状态？

注意事项

设计状态时，状态必须要能**唯一**、**确定地**表达当前遇到的局面！

小结

决策类的 DP，设计状态时，需要满足两个原则:

最优子结构

大问题的最优解，一定是从小问题的**最优**解推出来的。

例如硬币问题，凑出 15 元的最优解是由凑出 4、10、14 元的最优解（而不是最坏解）而来。

无后效性

现在的决策，只与过去的结果有关，而与过去的决策无关。

例如硬币问题，要在 n=15 时做出决策，只需要知道 f(4),f(10),f(14) 的**值**，而并不关心这些值怎么来的。

背包问题

今有 n 个物品，第 i 个物品体积为 v[i]，价值为 w[i].
有一背包，容量为 V. 您可以选择一些物品装进背包，问能装下的最大价值。

如何设计状态？

注意

状态必须能唯一、确定地反映局面，不能漏掉信息！

DP 三连

我是谁？

设计状态：dp[k][m] 表示“只考虑前 k 个物品，用 m 的容量能装下的最大价值”。

DP 三连

我是谁？

设计状态：dp[k][m] 表示“只考虑前 k 个物品，用 m 的容量能装下的最大价值”。

我从哪里来？

dp[k][m] 的来历：要么取了第 k 个物品，要么没有取。因此

$$dp[k][m] = \max \begin{cases} dp[k-1][m] \\ dp[k-1][m-v[k]] + w[k] \end{cases}$$

代码实现

```
for(int k=1; k<=n; k++)
    for(int m=0; m<=V; m++)
    {
        dp[k][m] = dp[k-1][m];

        if(m - v[k] >= 0)
            dp[k][m] = max(dp[k][m], dp[k-1][m-v[k]] + w[k]);
    }

```

采药

https://www.luogu.com.cn/problem/P1048

采药

https://www.luogu.com.cn/problem/P1048

就是个 01 背包。

开心的金明

https://www.luogu.com.cn/problem/P1060

开心的金明

https://www.luogu.com.cn/problem/P1060

体积： 价格
价值： 价格 $\times$ 重要度

01 背包的空间优化

回顾 01 背包的状态转移方程:

01 背包状态转移方程

$$dp[k][m] = \max \begin{cases} dp[k-1][m] \\ dp[k-1][m-v[k]] + w[k] \end{cases}$$

注意到 dp[k][·] 只与 dp[k-1][·] 有关，从而 dp 数组的第 1 行、第 2 行⋯⋯第 k-2 行全都被浪费了。占据了空间，但以后不会被访问到。

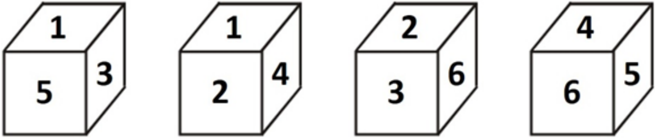
01 背包的空间优化

注意第 2 行循环的访问顺序!!!

```
1 for(int k=1; k<=n; k++)
2     for(int m=V; m>=0; m--)
3         if(m - v[k] >= 0)
4             dp[m] = max(dp[m], dp[m-v[k]] + w[k]);
```

Dice Game

今有一普通骰子，初始时 1 朝上。
作为一名赌徒，您参与了一个游戏：每次翻转骰子 90 度（也就是说，将一个侧面翻到顶面），然后把新的顶面的数字计入得分。
给定 n，问：至少需要多少次翻转，才能使得得分**恰好**为 n？



来源：http://codeforces.com/gym/101502/problem/D

DP 三连

我是谁？

设计状态：dp[k][c] 表示“最后 k 向上，得到恰好 c 分，所需要的最少步数”。
答案显然是 dp[$\cdot$][n] 中的最小值。

DP 三连

我是谁？

设计状态：dp[k][c] 表示“最后 k 向上，得到恰好 c 分，所需要的最少步数”。
答案显然是 dp[·][n] 中的最小值。

我到哪里去？

考虑“现在 k 向上，手上有 c 分”能去哪些状态：

$$dp[k][c] \to dp[p][c+p]$$

其中 p 为 [1,2,3,4,5,6] 中除了 k 和 7-k 以外的数。

装箱问题

https://www.luogu.com.cn/problem/P1049

装箱问题

https://www.luogu.com.cn/problem/P1049

用 dp[k][x] 表示“只考虑前 k 个物品，能否恰好装满 x 的体积”。
状态转移方程：

$$dp[k][x] = dp[k-1][x] \text{ or } dp[k-1][x - v[k]]$$

注意到可以滚动数组滚掉一维。

Boxes Game

来源： http://codeforces.com/gym/101502/problem/J

DP 总结

DP 三连

- 我是谁：设计状态
- 我从哪里来：设计 pull 型的转移
- 我到哪里去：设计 push 型的转移

DP 的实现方法

- 按顺序递推
- 记忆化搜索