从「信息」的角度看动态规划

洛谷网校五一课程 - 动态规划篇

Ruan Xingzhi

洛谷网校

2020 年 5 月 1 日

Update

- 黑科技降噪，以后不会有噪音了（但只能在 windows 下用）
- 扯了一条长达 10m 的网线连我笔记本电脑，不会丢帧了
- 画质大提升

Intro

Let's focus on the big picture.
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.

信息

本篇教程与以往的 DP 教程区别于，我们这一次会不停地提到“信息”的概念。

- 想解决一个问题，需要掌握哪些信息？
- 如何用尽可能少的信息推断出问题的解？
- 如何把难以得到的信息，转化为容易得到的信息？
- 如何用有限的空间，存储大量的信息？

这种思路不止对 DP 有用，亦体现在各种其他算法上。

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：枚举每一步采取什么方案。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：枚举每一步采取什么方案。指数级复杂度。
为什么?

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：枚举每一步采取什么方案。指数级复杂度。
为什么？
暴力模拟的时候，存储了“每一步采取了什么方案”，这是典型的没啥用的信息。

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：枚举每一步采取什么方案。指数级复杂度。
为什么？
暴力模拟的时候，存储了“每一步采取了什么方案”，这是典型的没啥用的信息。

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。

回顾

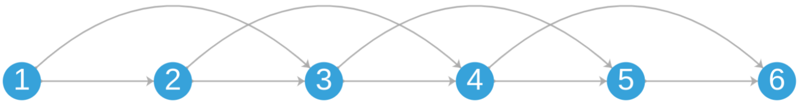
Q：我们想知道什么信息?
A：从 0 级走到 n 级的方案数 $f(n)$.

Q：为了得到 $f(n)$，我们需要哪些信息?
A：知道 $f(n)$ 和 $f(n-1)$ 即可。

Q：上面的递推式，能否求出所有的 $f(x)$?
A：可以。

求解过程

求出 $f[3]$，需要 $f[1], f[2]$ 的信息。以此类推，把依赖关系画成图：



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！

狗屁不通生成器问题

今有某人网上提交作业，打算随便糊弄。
本来文本框里面是没有字的。他每次干以下两件事之一：

- 打一个字上去，文本长度加 1。
- 把已有的所有的字复制一遍，文本长度翻倍。

他想打出恰好 n 个字，那他至少需要操作多少次？

设计状态

想打出 10 个字，最简方案是：

```
0 -> 1 -> 2 -> 4 -> 5 -> 10
```

设计状态：
我们记 "打出 n 个字所需要的最少操作次数" 为 $f(n)$.
为了求出 $f(n)$，需要什么信息？

求解

不难注意到

$$f(n) = \min \begin{cases} f(n-1), \\ f(n/2) & \text{if } 2 \mid n \end{cases}$$

而又有基础 $f(0) = 0$，故每一个 $f(x)$ 都可求。
代码怎么写？

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 $f[1], f[2] \dots f[n]$ 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！

最长公共子序列问题

今有俩数组 A, B，记 $\text{lcs}(A, B)$ 为 A,B 最长的公共子序列的长度。

例

A = [1,3,2,4,7,5,7]
B = [1,2,0,3,5,7,9]

最长的公共子序列是 [1,2,5,7]
lcs(A,B) = 4

设计状态

记 $f(x, y)$ 表示 A 的前 x 个元素，与 B 的前 y 个元素的 LCS.
获取 $f(n, m)$ 需要什么信息?

设计状态

记 $f(x, y)$ 表示 A 的前 x 个元素，与 B 的前 y 个元素的 LCS.
获取 $f(n, m)$ 需要什么信息?

- $f(n, m)$ 可以是 $f(n-1, m)$
- $f(n, m)$ 可以是 $f(n, m-1)$
- 如果 $a[n] = b[n]$，则 $f(n)$ 可以取 $f(n-1, m-1) + 1$

ref. https://blog.csdn.net/hrn1216/article/details/51534607

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?

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

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

$$
\begin{aligned}
F[1] &= 1 \\
F[2] &= 1 \\
F[n] &= F[n-2] + F[n-1]
\end{aligned}
$$

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。