动态规划

陆明琪

清华大学

July 28, 2023

状态 + 转移

动态规划的核心是状态和转移。
什么是状态？什么是转移？

状态 + 转移

动态规划的核心是状态和转移。

什么是状态？什么是转移？

例子：数塔

设有一个三角形的数塔，顶点为根结点，每个结点有一个整数值。从顶点出发，可以向左走或向右走。

要求从根结点出发，请找出一条路径，使路径之和最大，输出路径的长度。

状态

状态在自然语言中的解释

比如：我在数塔上走的路径是 $(1,1) \to (2,1) \to (3,1)$，这可以称之为一种状态

状态

状态在自然语言中的解释

比如：我在数塔上走的路径是 $(1,1) \to (2,1) \to (3,1)$，这可以称之为一种状态

但是 DP 中的“状态”不太一样

上述例子中 $(1,1)$ 和 $(2,1)$ 都是冗余的信息，我们其实并不在意走过的路径，我们只在意最后的位置。

状态

状态在自然语言中的解释

比如：我在数塔上走的路径是 $(1,1) \to (2,1) \to (3,1)$，这可以称之为一种状态

但是 DP 中的“状态”不太一样

上述例子中 $(1,1)$ 和 $(2,1)$ 都是冗余的信息，我们其实并不在意走过的路径，我们只在意最后的位置。

本质上是将原本详尽的状态进行了合并同类项

转移

一个状态通过转移，到了另一个状态
转移是状态之间的桥梁
（以前状态 + 转移）决定了当前状态的情况

转移

一个状态通过转移，到了另一个状态
转移是状态之间的桥梁
（以前状态 + 转移）决定了当前状态的情况
通常来说有两种写法：前向与后向
数塔中可以写 $f[i][j] = a[i][j] + \max\{f[i-1][j] + f[i-1][j+1]\}$
也可以写 $chkmin(f[i+1][j-1], a[i+1][j-1] + f[i][j])$，$chkmin(f[i+1][j], a[i+1][j] + f[i][j])$

记忆化搜索

优雅的 brute force & 常数较大的 DP

记忆化搜索

优雅的 brute force & 常数较大的 DP

例子：斐波那契数列

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

记忆化搜索

优雅的 brute force & 常数较大的 DP

例子：斐波那契数列

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

优点

便于理解

跳过无效状态

难以用循环来表示的困难转移

转移顺序不容易错

记忆化搜索

例子：滑雪

问题描述：Michael 喜欢滑雪。这并不奇怪，因为滑雪的确很刺激。可是为了获得速度，滑的区域必需向下倾斜，而且当你滑到坡底，你不得不再次走上坡或者等待升降机来载你。Michael 想知道在一个区域中最长的滑坡。区域由一个二维数组给出。数组的每个数字代表点的高度。下面是一个例子

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 16 & 17 & 18 & 19 & 6 \\ 15 & 24 & 25 & 20 & 7 \\ 14 & 23 & 22 & 21 & 8 \\ 13 & 12 & 11 & 10 & 9 \end{bmatrix}$$

$N \le 1000, Limit \le 1000$

最长公共子序列

给出两个字符串序列，求最长公共子序列：子序列可以不连续。

最长公共子序列

给出两个字符串序列，求最长公共子序列：子序列可以不连续。
$dp[i][j]$ 表示分别匹配到第 i 位和第 j 位置，如何转移？

最长公共子序列

给出两个字符串序列，求最长公共子序列：子序列可以不连续。
$dp[i][j]$ 表示分别匹配到第 i 位和第 j 位置，如何转移？
要么最后两位匹配，要么最后两位不匹配
不匹配时要么没选 i 要么没选 j

方格取数

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

方格取数

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

$dp[i][j]$ 表示走到第 i 行第 j 列（且这一列不会再移动）

方格取数

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

$dp[i][j]$ 表示走到第 i 行第 j 列（且这一列不会再移动）

从上一列转移，因为状态定义了不会再移动，所以只会加上第 j 列的值

方格取数

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

$dp[i][j]$ 表示走到第 i 行第 j 列（且这一列不会再移动）

从上一列转移，因为状态定义了不会再移动，所以只会加上第 j 列的值

$dp[i][j] = \max\{dp[i'][j-1] + \sum_{t=i'}^{i} a[t][j]\}$

方格取数

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

$dp[i][j]$ 表示走到第 i 行第 j 列（且这一列不会再移动）

从上一列转移，因为状态定义了不会再移动，所以只会加上第 j 列的值

$dp[i][j] = \max\{dp[i'][j-1] + \sum_{t=i'}^{i} a[t][j]\}$

前缀和优化，发现 max 有结合律

分两次（因为前缀和相减的顺序）转移，取最优

上升点列

https://www.luogu.com.cn/problem/P8816

上升点列

https://www.luogu.com.cn/problem/P8816
先排序

上升点列

https://www.luogu.com.cn/problem/P8816
先排序
$dp[i][j]$ 表示考虑前 i 个点，选了第 i 个点，用了 j 个自由点的答案

上升点列

https://www.luogu.com.cn/problem/P8816
先排序
$dp[i][j]$ 表示考虑前 i 个点，选了第 i 个点，用了 j 个自由点的答案
枚举上一个选的点
计算中间要添加多少自由点

导弹拦截

https://www.luogu.com.cn/problem/P1020

导弹拦截

https://www.luogu.com.cn/problem/P1020
$dp[i]$ 表示考虑前 i 个点，选了第 i 个点的答案

导弹拦截

https://www.luogu.com.cn/problem/P1020
$dp[i]$ 表示考虑前 i 个点，选了第 i 个点的答案
枚举上一个选的点

导弹拦截

更快的做法：$O(n\log n)$

导弹拦截

更快的做法：$O(n\log n)$

考虑对于同一种长度的导弹序列，结尾导弹越高，能够接受的下一颗导弹的范围越广

导弹拦截

更快的做法：$O(n \log n)$

考虑对于同一种长度的导弹序列，结尾导弹越高，能够接受的下一颗导弹的范围越广

用一个数组 $a[i]$ 表示长度为 i 的序列结尾最高是多少

导弹拦截

更快的做法：$O(n \log n)$

考虑对于同一种长度的导弹序列，结尾导弹越高，能够接受的下一颗导弹的范围越广

用一个数组 $a[i]$ 表示长度为 i 的序列结尾最高是多少

a 是一个单调不增数组

假设非单不增减，则 $a[i] < a[i+1]$ 中 $a[i]$ 可以找到一个更高的导弹，与最高矛盾

导弹拦截

更快的做法：$O(n \log n)$

考虑对于同一种长度的导弹序列，结尾导弹越高，能够接受的下一颗导弹的范围越广

用一个数组 $a[i]$ 表示长度为 i 的序列结尾最高是多少

a 是一个单调不增数组

假设非单不增减，则 $a[i] < a[i+1]$ 中 $a[i]$ 可以找到一个更高的导弹，与最高矛盾

二分找到最大的 i 使得 $a[i] >= H$

导弹拦截

最少需要打多少次导弹？

导弹拦截

最少需要打多少次导弹？

Solution 1

贪心

$a[i]$ 维护第 i 个队列的结尾高度，这里我们不妨令 a 单调增

二分找到最小的 i 使得 $a[i] >= H$

导弹拦截

最少需要打多少次导弹？

Solution 1

贪心

$a[i]$ 维护第 i 个队列的结尾高度，这里我们不妨令 a 单调增

二分找到最小的 i 使得 $a[i] >= H$

Solution 2

结论：最长上升子序列

从算法过程可以看出是最长上升子序列

证明较为复杂，略，感兴趣可以去看 Dilworth 定理

状压 DP

状压 DP：$dp[mask]$
状态复杂度：$O(2^n)$

状压 DP

状压 DP：$dp[mask]$
状态复杂度：$O(2^n)$
复杂度似乎只比 $O(n!)$ 快？
状压 DP 在很多情况下就是用来优化 $O(n!)$ 的暴力的

对于阶乘的搜索，我们除了知道选了哪些以外，还有它们的顺序信息，但是有的问题并不关注顺序信息，这时候我们就可以用状压 DP。

状压 DP

状压 DP：$dp[mask]$
状态复杂度：$O(2^n)$
复杂度似乎只比 $O(n!)$ 快？
状压 DP 在很多情况下就是用来优化 $O(n!)$ 的暴力的

对于阶乘的搜索，我们除了知道选了哪些以外，还有它们的顺序信息，但是有的问题并不关注顺序信息，这时候我们就可以用状压 DP。

状压 DP 用记忆化搜索写会舒服很多。

旅行商问题

http://poj.org/problem?id=3311

给定一系列城市和每对城市之间的距离，求解访问每一座城市一次并回到起始城市的最短回路。

旅行商问题

http://poj.org/problem?id=3311

给定一系列城市和每对城市之间的距离，求解访问每一座城市一次并回到起始城市的最短回路。

因为是回路，所以不妨设 1 号点为起点。记 $dp[mask][pos]$ 表示已经经过了 $mask$ 中的点，目前在 pos 位置的最短路径。

Hamilton 路

另一个常见的 NPC 问题：是否存在 Hamilton 路径（每个点经过一次）

记 $dp[mask][pos]$ 表示已经经过了 $mask$ 中的点，目前在 pos 位置是可以走到的。

Hamilton 路

另一个常见的 NPC 问题：是否存在 Hamilton 路径（每个点经过一次）

记 $dp[mask][pos]$ 表示已经经过了 $mask$ 中的点，目前在 pos 位置是可以走到的。

一个关于 Hamilton 路径的结论

竞赛图一定有 Hamilton 路径，强连通竞赛图一定有 Hamilton 回路。

树形 DP

$f(u, S)$ 表示对于子树 u 的子问题进行求解，状态为 S

树形 DP

$f(u,S)$ 表示对于子树 u 的子问题进行求解，状态为 S
常见的转移
子树合并（如：树上背包）
从 u 转移到 u 的父亲 $fa[u]$（如：树上最大独立集）

树形 DP

树的直径

树形 DP

树的直径
Solution 1：维护到子树最长的长度
Solution 2：两遍 DFS

骑士

树上的最大独立集（选择的点两两不互相邻）

骑士

树上的最大独立集（选择的点两两不互相邻）
$dp[u][0/1]$ 表示在 u 的子树中，选/不选 u 号点的答案。
$dp[u][0] = \sum_v \max\{dp[v][0], dp[v][1]\}$。
$dp[u][1] = \sum_v dp[v][0] +$

骑士

树上的最大独立集（选择的点两两不互相邻）
$dp[u][0/1]$ 表示在 u 的子树中，选/不选 u 号点的答案。
$dp[u][0]=\sum_v \max\{dp[v][0],dp[v][1]\}$。
$dp[u][1]=\sum_v dp[v][0]+$
环套树森林上的最大独立集？

骑士

首先发现不是同一棵环套树上的互不影响。

骑士

首先发现不是同一棵环套树上的互不影响。

对于一棵树，考虑一条环上的边 $< u, v >$，要么 u 没选，要么 v 没选。

答案就是 $\max\{dp[u][0], dp[v][0]\}$。

这两个 DP 是以 u 和 v 为根分别做的。

01 背包

01 背包

有 N 件物品和一个容量为 V 的背包。第 i 件物品的费用是 $c[i]$，价值是 $w[i]$。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

每种物品仅有一件，可以选择放或不放。

01 背包

01 背包

有 N 件物品和一个容量为 V 的背包。第 i 件物品的费用是 $c[i]$，价值是 $w[i]$。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

每种物品仅有一件，可以选择放或不放。

$f[i][v]$ 表示前 i 件物品恰放入一个容量为 v 的背包可以获得的最大价值。则其状态转移方程便是：

$f[i][v] = \max\{f[i-1][v], f[i-1][v-c[i]] + w[i]\}$

完全背包

完全背包

有 N 件物品和一个容量为 V 的背包。第 i 件物品的费用是 $c[i]$，价值是 $w[i]$。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

每种物品都有无限件可用。

完全背包

完全背包

有 N 件物品和一个容量为 V 的背包。第 i 件物品的费用是 $c[i]$，价值是 $w[i]$。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

每种物品都有无限件可用。

$f[i][v] = \max\{f[i-1][v-k*c[i]] + k*w[i] | 0 \leq k*c[i] \leq v\}$

复杂度？

完全背包

完全背包

有 N 件物品和一个容量为 V 的背包。第 i 件物品的费用是 $c[i]$，价值是 $w[i]$。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

每种物品都有无限件可用。

$f[i][v] = \max\{f[i-1][v-k*c[i]]+k*w[i]|0 \le k*c[i] \le v\}$

复杂度？

$f[i][v] = \max\{f[i-1][v], f[i][v-c[i]]+w[i]\}$

和方格取数同样的思考方式：发现 $i+1$ 和 i 的转移方程只差一项。

多重背包

多重背包

有 N 件物品和一个容量为 V 的背包。第 i 件物品的费用是 $c[i]$，价值是 $w[i]$。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

第 i 种物品最多有 $n[i]$ 件可用。

多重背包

多重背包

有 N 件物品和一个容量为 V 的背包。第 i 件物品的费用是 $c[i]$，价值是 $w[i]$。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

第 i 种物品最多有 $n[i]$ 件可用。

$f[i][v] = \max\{f[i-1][v-k*c[i]] + k*w[i] | 0 \le k*c[i] \le v\}$

复杂度？

多重背包

退一步进两步，先介绍一种完全背包的新做法

多重背包

退一步进两步，先介绍一种完全背包的新做法

把第 i 种物品拆成费用为 $c[i] * 2^k$、价值为 $w[i] * 2^k$ 的若干件物品，其中 k 满足 $c[i] * 2^k < V$。

因为不管最优策略选几件第 i 种物品，总可以表示成若干个 2^k 件物品的和。

多重背包

退一步进两步，先介绍一种完全背包的新做法

把第 i 种物品拆成费用为 $c[i]*2^k$、价值为 $w[i]*2^k$ 的若干件物品，其中 k 满足 $c[i]*2^k < V$。

因为不管最优策略选几件第 i 种物品，总可以表示成若干个 2^k 件物品的和。

对于多重背包来说，只是多了总数小于等于 $n[i]$ 的限制

控制我们拆出来的物品总数，拆成 $n = 1+2+4+8+\cdots+2^k+\epsilon$

例子：$13 = 1+2+4+6$

金明的预算方案

https://www.luogu.com.cn/problem/P1064

金明的预算方案

https://www.luogu.com.cn/problem/P1064
和 01 背包的区别是有从属关系，也就是先买某一个才能再买某一个
但是从属关系只有最多两个，可以从这里入手考虑

金明的预算方案

https://www.luogu.com.cn/problem/P1064

和 01 背包的区别是有从属关系，也就是先买某一个才能再买某一个

但是从属关系只有最多两个，可以从这里入手考虑

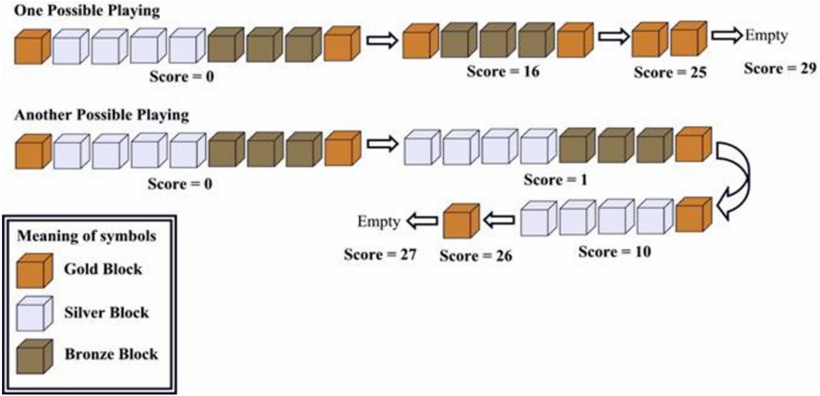
分类讨论一个从属关系中的购买情况

不买、买主件、买主件 + 附件 1、买主件 + 附件 2、买主件 + 附件 1+ 附件 2

引入

例子：方块消除

游戏时，你可以任选一个区域消去。设这个区域包含的方块数为 x，则将得到 x^2 个分值。方块消去之后，其余的方块就会合并。求最大分值。



什么时候能用区间 DP

$dp[i][j][k]$ 表示把第 i 段到第 j 段（第 j 段的块数 $+k$）全部消除后的最大分值。

什么时候能用区间 DP

$dp[i][j][k]$ 表示把第 i 段到第 j 段（第 j 段的块数 $+k$）全部消除后的最大分值。

转移考虑最后一段会发生什么

直接消除：$dp[i][j][k] = dp[i][j-1][0] + (len[j] + k)^2$。

合并消除：$dp[i][j][k] = \max_t\{dp[t+1][j-1][0] + dp[i][t][len[j] + k]\}$，其中第 t 段颜色和第 j 段相同。

什么时候能用区间 DP

所以对于“方块消除”我们到底为什么不能用常用的 $dp[i][j]$ 的状态？

什么时候能用区间 DP

所以对于“方块消除”我们到底为什么不能用常用的 $dp[i][j]$ 的状态？

因为，如果尝试枚举第一个消除的位置是不行的，左右两边会合并之后相互影响。而在考虑最后一段在删掉一段的时候，右边会合并过来，合并之后 $i \sim t$ 这一段的状态是不能简单用 $dp[i][t]$ 表示的。

什么时候能用区间 DP

所以对于“方块消除”我们到底为什么不能用常用的 $dp[i][j]$ 的状态？

因为，如果尝试枚举第一个消除的位置是不行的，左右两边会合并之后相互影响。而在考虑最后一段在删掉一段的时候，右边会合并过来，合并之后 $i \sim t$ 这一段的状态是不能简单用 $dp[i][t]$ 表示的。

那为什么加一维就可以了？

什么时候能用区间 DP

所以对于“方块消除”我们到底为什么不能用常用的 $dp[i][j]$ 的状态？

因为，如果尝试枚举第一个消除的位置是不行的，左右两边会合并之后相互影响。而在考虑最后一段在删掉一段的时候，右边会合并过来，合并之后 $i \sim t$ 这一段的状态是不能简单用 $dp[i][t]$ 表示的。

那为什么加一维就可以了？

这里的 k 是最后一段合并的信息，加上 k 之后其长度信息就得到了保留。

String painter

给出长度相同的字符串 A 和 B，每一次我们可以使用一种颜色（用字母代替）刷任意一个连续子序列，问将字符串 A 变成字符串 B 最少需要刷多少次。

$$zzzzzfzzzzz$$
$$abcdefedcba$$
$$6$$
$$ababababab$$
$$cdcdcdcdcd$$
$$7$$

String painter

如果 A 和 B 没有对应相同的颜色的话好像和上一题差不多

直接刷：$dp[i][j][k] = dp[i][j-1][0] + 1$。

合并刷：$dp[i][j][k] = max_t\{dp[t+1][j-1][0] + dp[i][t][k+1]\}$，其中第 t 个颜色和第 j 个相同。

String painter

如果 A 和 B 没有对应相同的颜色的话好像和上一题差不多

直接刷：$dp[i][j][k] = dp[i][j-1][0] + 1$。

合并刷：$dp[i][j][k] = max_t\{dp[t+1][j-1][0] + dp[i][t][k+1]\}$，其中第 t 个颜色和第 j 个相同。

发现最后一维没用，直接删掉就好了。

String painter

如果 A 和 B 没有对应相同的颜色的话好像和上一题差不多

直接刷：$dp[i][j][k] = dp[i][j-1][0] + 1$。

合并刷：$dp[i][j][k] = max_t\{dp[t+1][j-1][0] + dp[i][t][k+1]\}$，其中第 t 个颜色和第 j 个相同。

发现最后一维没用，直接删掉就好了。

那 A 可能会和 B 有对应相同的颜色怎么办？

String painter

如果 A 和 B 没有对应相同的颜色的话好像和上一题差不多

直接刷：$dp[i][j][k] = dp[i][j-1][0] + 1$。

合并刷：$dp[i][j][k] = max_t\{dp[t+1][j-1][0] + dp[i][t][k+1]\}$，其中第 t 个颜色和第 j 个相同。

发现最后一维没用，直接删掉就好了。

那 A 可能会和 B 有对应相同的颜色怎么办？

再用另一个 DP。

String painter

记 $ans[i]$ 表示把 $A[1,i]$ 刷成 $B[1,i]$ 的最少次数。转移时同样考虑最后一位的情况。

String painter

记 $ans[i]$ 表示把 $A[1,i]$ 刷成 $B[1,i]$ 的最少次数。转移时同样考虑最后一位的情况。

匹配：$ans[i] = ans[i-1]$。

不匹配，说明最后肯定有一段要用 $dp[t][i]$ 计算。

$$ans[i] = \min_{1 \le t \le i} \{ans[t-1] + dp[t][i]\}$$

最多回文子串

给出一个字符串 S，求其回文子串的数量。（这里不同的回文字串只要求位置不同，如：aaaaa 的最多回文子串数目是 31。）

最多回文子串

给出一个字符串 S，求其回文子串的数量。（这里不同的回文字串只要求位置不同，如：aaaaa 的最多回文子串数目是 31。）

记 $dp[i][j]$ 表示在子区间 $[i,j]$ 中的回文子串的数量。由容斥原理可知，$dp[i][j]=dp[i+1][j]+dp[i][j-1]-dp[i+1][j-1]+s[i][j]$，其中 $s[i][j]$ 表示 i 和 j 都选的回文子串的数量。

当 $S[i]\neq S[j]$ 时，$s[i][j]=0$。否则，$s[i][j]=dp[i+1][j-1]+1$，即 i 和 j 都要选上，中间可以是回文子串或是空串。

最多回文子串

如果我们把记录了区间信息的 DP 都称为区间 DP，那么区间 DP 不一定就是 $O(n^3)$ 的。比如本题就是 $O(1)$ 转移的，而“方块消除”中如果规定不同的合并的最大段数，也会有不同的转移复杂度。

最多回文子串

如果我们把记录了区间信息的 DP 都称为区间 DP，那么区间 DP 不一定就是 $O(n^3)$ 的。比如本题就是 $O(1)$ 转移的，而“方块消除”中如果规定不同的合并的最大段数，也会有不同的转移复杂度。

对于计数类的 DP，写的时候要考虑清楚转移的关系，不能有重复计算的情况。但是对于求最值，只要保证最优情况会被计算即可。

Thanks

谢谢大家。