DP 的一些优化技巧

绍兴市第一中学 张恒捷

摘 要

动态规划是信息学竞赛中十分常见的题目类型。本文列举了若干针对 DP 的优化方法，在一些题目上大显身手，获得了不错的效果。

1 引言

DP (Dynamic Programming)，即动态规划。是一种通过把原问题分解为相对简单的子问题来求解复杂问题的方法。近几年来，DP 一直在信息学竞赛中，尤其是各大竞赛网站上频繁出现，而且出现形式复杂，种类繁多。

一些诸如单调队列，斜率优化，以及四边形不等式的传统方法在前几年的论文中已经有详细介绍，本文将不再一一赘述。作者研究整合出了一系列 DP 优化技巧，并给出了一些例题帮助读者深入理解，希望对广大参加信息学竞赛的同学带来或多或少的帮助。

注：本文着重介绍优化技巧，故例题中的转移方程是如何得到的将不作为重点介绍。

2 方法整合

2.1 矩阵乘法加速

相信用矩阵乘法来优化 *DP* 转移的做法大家都已经熟知，故在此只是简单介绍一下。假如把 $f(i)$ 看成一个向量，而 $f(i+1)$ 可以用 $f(i)$ 乘上一个转移系数矩阵 A 得到，那么所求的 $f(n)$ 就可以写成 $f(0)\times A^n$ 的形式。由于矩阵乘法满足结合律，因此对于后者 A^n 只需要用快速幂优化即可。

因此对于一类转移单一并且转移次数又非常多的时候，可以考虑使用矩阵乘法来优化该DP。

2.2 位运算加速

对于一些 *DP* 其键值只会是0 或者1，这时就可以考虑使用位运算来打包处理信息了。0/1 背包就是一个很好的例子。用了位运算优化后程序速度将会极大地提升，可以过掉一些数据规模较大的数据了。

2.3 优化状态的数量

例1. *(codeforces Round 265,Div 1,D)*

现在有一个人玩一款游戏，游戏中有 k 种装备。每种装备初始的时候都是等级1，现在开始打怪，每打一个怪系统就会随机掉出一件装备，随机方式如下：先等概率随机装备的种类，假设该种类的装备当前的等级为 t，则系统会在 $[1 \sim t+1]$ 中等概率随机出该装备的等级。爆出装备后，如果该装备比玩家当前穿戴的等级高，则玩家会卖掉原装备，换上新装备，否则玩家会直接卖掉生成出的装备。等级为 i 的装备价值 i 金币。求打了 n 只怪后获得金币的期望。

数据范围： $n \le 10^5, k \le 100$

由于是求期望，而期望有线性性。所以可以针对一种装备算出它的期望，然后答案乘 k 就能得到最终的答案。

设 $f(i,j)$ 为当前已经有等级为 j 的装备了，再打 i 只怪期望获得多少金币。可以得到DP 方程：

$$f(0,j) = 0$$

$$\begin{aligned} f(i,j) &= \frac{1}{k}\sum_{x=1}^{j+1}\frac{1}{j+1}(f(i-1,max(j,x)) + min(j,x)) + \frac{k-1}{k}f(i-1,j) \\ &= \frac{1}{k}\left(\frac{j}{j+1}\cdot(f(i-1,j) + \frac{j+1}{2}) + \frac{1}{j+1}\cdot(f(i-1,j+1)+j)\right) + \frac{k-1}{k}\cdot f(i-1,j) \end{aligned}$$

这个DP 方程的转移已经做到 $O(1)$ 了，因此我们可以使用标题所写的方法，尝试着删去那些非常小的概率才发生的状态？在本题中，如果已经拥有了等级为 p 的装备，下一次得到等级为 $p+1$ 的概率为 $\frac{1}{kp}$。因此，如果最终得到了等

级为 T 的装备，那么期望打败的怪物数为 $2k+3k+\cdots+Tk\approx\frac{kT^2}{2}$，可以得出打败了 n 只怪物装备能升级到 $\sqrt{\frac{2n}{k}}$ 左右。根据这个思路，我们在 $\sqrt{2n}$ 左右寻找那条界线，最后在700 左右发现当这个界线增大的话，答案只会有非常细微的变化。设界线为B，事实上 $B\sim O(\sqrt{n})$。因此我们最终得到了一个 $O(n\sqrt{n})$ 的做法。

本题使用了优化状态的方法，在保证了答案精度的情况下，直接优化了复杂度，不失为一种很好的方法。

2.4 找规律

找规律是OI解题中的一大特色，但是似乎并不能直接和严谨的DP算法联系起来，下文将会以一道非常具有代表性的趣题为例，分析找规律在一类分层DP中的应用。

例2. *(topcoder SRM 526 Div 1 - Level 3)*

一个包含 n 个串的集合，一个模式串 S。每次从集合中随机选一个串（选完后仍在集合中），选 K 次后全部拼起来变成 T，求 T 中 S 出现的期望次数。

数据范围：$n\le 50,|S|\le 500,K\le 10^{12}$

我们来分析一下这道题吧。首先由于题目要求期望，故设 $f(i,j)$ 表示：若当前已经匹配到了串 S 的 j 位置，再随机选 i 个串后 S 在之后期望出现了几次。很容易列出 DP 方程：

$$f(i,j)=\frac{1}{n}\sum_{x=1}^{n}(f(i-1,trans(x,j))+val(x,j))$$

其中$trans(i,j)$ 表示从 S 的第 j 位开始匹配，在加入了第 i 个串后匹配到了哪里，$val(i,j)$ 表示对应的匹配次数。最后$f(K,0)$ 就是要求的值。

相信不少同学看见这个转移方程就会想用矩阵乘法来优化。虽然这很正确，但是由于复杂度较高（为 $O(|S|^3logK)$），是无法通过给定的时限的。

如果打出一张 f 函数的表，就能够发现在 i 较大的时候（事实上是 $i>|S|$ 时），对于任意的 j，有 $f(i,j)-f(i-1,j)$ 是一个定值！这里可以对这个规律给出简要的说明：

假设当前已经选了 x 个串了，考虑选出第 $x+1$ 个串后所贡献的答案，换种说法就是以第 $x+1$ 个串中的某个位置作为结束位置的匹配个数。假设 S 的长度为 L，那么除了第 $x+1$ 个串自身所拥有的匹配个数以外，还与它前面 $L-1$ 个字符有关。而当 x 的值远大于 L 的时候，前 $L-1$ 个字符都是随机出来的了。也就是说，当 x 大于 L 时，第 x 个串与第 $x+1$ 个串所带来的贡献的计算方式一模一样。所以会出现上述的神奇性质。使用这个算法后只需要做出 $i \le |S|$ 的解即可。复杂度为 $O(|S|^2 n)$，足以通过此题。

2.5 分离状态中的变量

在 *DP* 状态中，可能会有一些在转移中互不影响的变量，这时就可以将它们分别进行运算来达到一起运算时的效果。下面看一道例题深入了解一下。

例3. *(topcoder SRM 639 Div 1 - Level 2)*

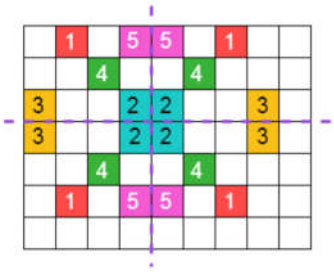
一张纸被等分成 n 行 m 列个小格子。每个格子是黑或者白色。整张纸可以用一个01 矩阵来表示。每次操作可以找一条竖直或者水平的不跨越小格子的线，然后将纸折叠一下。折叠时要保证贴合的两面每个格子的颜色都相同，而且不能把大的一面覆盖在小的上面。在任意次操作下最后有多少种结局。结局不同当且仅当剩下的纸在原矩阵中的位置不同。

$n, m \le 250$

一个很暴力的想法就是：设 $f(u,d,l,r)$ 表示原来的纸是否能折到上到 u，下到 d，左到 l，右到 r 的子矩阵。转移时枚举接下来的操作使用了哪条对称轴。最终答案就是所有状态中为 *true* 的个数。

但是这个方法复杂度较高，需要优化。事实上对于这道题来说横着折叠与竖着折叠是是可以分开做的。也就是说 (u,d) 与 (l,r) 其实是互不影响的。让我们分析一下是怎么得到这个性质的。

考虑任意一张带颜色的纸，如果我们连续使用了一次横向折叠与一次纵向折叠，如图：

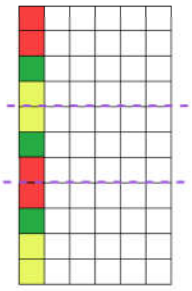


同样的颜色就代表了同样的数字，可以发现先横向后纵向与先纵向后横向是等价的。

考虑任意一个操作序列，一定是由一些横向，一些纵向的操作组成的，比如说 $VHVVHV$。由于相邻两个 HV 交换后是等价的，所以可以通过一系列交换使得所有的 H 都排在 V 前面，或者把所有的 V 都排在 H 的前面。如：$VHVVHV \to HVVHVV \to ... \to HHVVVV$。因此 $VHVVHV$ 合法等价于 HH 与 $VVVV$ 合法。所以我们成功的将横着折与竖着折分开了，最后的答案就相当于横着折的方案乘上竖着折的方案。

这样就可以很轻松的设计出状态了，考虑横着折的情况（竖着只要旋转90度），用 $f(u,d)$ 表示原来的纸是否能折出上到 u，下到 d 的子矩形，转移时枚举折了那条线，这个用回文串的方法预处理一下就可以做到 $O(n^3)$ 了，足以过掉此题。

但是优化并没有结束，事实上上述算法中 $f(u,d)$ 中的 u 和 d 也是可以分开做的。与之前的想法类似，考虑相邻两次往上折与往下折。如果翻过去的纸没有碰到折痕，那么这两次操作顺序交换一下是等价的。如果碰到了折痕，由于折痕两侧的颜色是相同的，所以超过折痕的部分可以预先折进去，这样就到了没有碰到折痕的情况。因此我们总可以把向上折的操作与先下折的操作分离。



在只考虑向下折的情况下（向上折的话只要上下翻转），我们只要用 $f(u)$ 表示是否原来的纸是否能折到上到 u 的情况。这样复杂度就成功降到了 $O(n^2)$ 。

在上面的例子中，我们发现在变量分离过后，原先的4维状态被逐个击破，最后变成了4个无关的1 维状态，程序的复杂度大大降低。在特定的题目中使用往往能获得不错的效果。

2.6 将键值函数化

将键值函数化，顾名思义，就是把 $(x, f(x))$ 绘制在二维平面上，然后用一些函数来拟合它们，这样一个DP 数组就可以使用一些分段函数来表示了。这种方法可以做定义在实数域上的动态规划问题，或者定义域非常大的情况。而且将其绘制成函数后也会方便研究。

例4. *(topcoder SRM 610 Div 1 - Level 3)*

一个 $n \times m$ 的矿区，一共有 K 秒，每秒钟在 (a_i, b_i) 处会出现一个矿。假设你在 (x, y) 处，会获得 $n+m-|x-a_i|-|y-b_i|$ 的钱数。第 i 秒钟你横坐标不能移动超过 dx_i ，纵坐标移动不能超过 dy_i ，一开始你可以在任意位置，求最大钱数。

$K \le 1000$

$n, m \le 10^6$

我们发现 x 与 y 在计算最优值时不影响，所以可以用本文“分离状态中的变量”的方法将 x 与 y 分开考虑。以横坐标为例，设 $f(i, j)$ 为第 i 秒时，人在 j 处的与横坐标有关的最大收益。很显然有转移：

$$f(i,j) = \left(\max_{|j-k| \le dx_i} f(i-1,k) \right) + n - |j - a_i|$$

不过第二维状态的定义域过于大，所以不妨使用本节提到的方法，将 $f(i)$ 看成一个函数。我们可以发现 $f(i)$ 的导数[1]是单调不增的。这个可以使用归纳法证明。根据这个性质，我们可以得出如下做法：设 $f(i-1)$ 的最高点在 $x = u$ 处，那么：

[1]不考虑断处无导数的情况

$$f(i,j) = f(i-1, j+dx_i) + n - |j - a_i| \qquad (j < u - dx_i)$$
$$f(i,j) = f(i-1, u) + n - |j - a_i| \qquad (u - dx_i \le j \le u + dx_i)$$
$$f(i,j) = f(i-1, j-dx_i) + n - |j - a_i| \qquad (j > u + dx_i)$$

我们可以发现 $f(i-1)$ 到 $f(i)$ 的过程只是将函数在最高点分割开，平移一下，再整体加上一个 $n + |j - a_i|$ 的函数。所以我们可以直接维护构成函数的线段即可。

复杂度也优化到了 $O(K^2)$。

例5. *(codeforces Round 172,Div 1,E)*

给出一个序列 $x[1..n]$ 与三个数 $q, a, b (a \le b, a \times (n-1) < q)$，满足 $1 \le x_1 \le x_2 \le \cdots \le x_n \le q)$。现在要求一个序列 $y[1..n]$，满足 $1 \le y_i \le q; a \le y_{i+1} - y_i \le b$。你需要做的是最小化 $\sum_{i=1}^{n}(y_i - x_i)^2$

数据范围：

$n \le 6000$

$1 \le q, a, b \le 10^9$

我们先列出*DP*方程：$f(i,j)$ 表示确定了 $y[1..i]$，并且 $y_i = j$ 时，$\sum_{k=1}^{i}(y_k - x_k)^2$ 最小是多少。有转移：

$$f(i,j) = \left(\min_{k=j-b}^{j-a} f(i-1,k)\right) + (j - x_i)^2$$

只不过在这道题中所求的 y 可以是实数，这让普通的*DP*变得很难求解。因此我们仍然使用与上一题类似的做法，将 $f(i)$ 用分段函数来表示。经过分析我们仍然可以得出其导数单调的性质（分析过程与上一题类似），因此我们可以使用与上一题同样的方法来维护这些分段函数，只不过在此题中 $f(i)$ 是由一些二次函数组成的。

复杂度： $O(n^2)$

由于二次函数太复杂了，所以我们可以直接维护它们的导数！因为这两者的本质是相同的。这样就回到了维护一次函数时的情况。我们又发现所有的操作都是可以用平衡树来实现的，所以我们成功把复杂度降为 $O(n \log n)$。

2.6.1 小结

其实包括斜率优化在内的一些其他方法都可以属于本节提到的“将键值函数化”的方法。该方法往往能解决定义域非常大或者无穷的情况，而且具有非常强的拓展性。在例 5上使用了该方法后成功使用了数据结构优化了复杂度。

2.7 启发式合并DP 数组

现在考虑如下问题：有 m 个操作， n 个变量。每一个操作只能作用在某一个变量上，并且会给这个变量带来一些变化。求使用不超过 k 次操作的前提下最大化所有变量的和。假设我们已经求出了 $f(i,j)$ 表示如果在变量 i 上使用了 j 个操作，最大能到多少。我们可以两两合并这些*DP*数组。比如说在合并 a 变量与 b 变量后，可以得出在 a,b 上使用了 j 个操作， $a+b$ 最大能到多少。如果 a 变量用了 x 次操作， b 变量用了 y 次操作，那么合并后就用了 $x+y$ 次操作。所以很显然，合并一个长为 x 的数组与一个长为 y 的数组将会得到长为 $x+y$ 的数组。合并完所有的数组就可以得到最终的答案了。但是我们应该以什么顺序合并它们呢？

我们可以把这个问题扩展到一类*DP*问题上：将一个长为 x 的数组看做一个权重为 x 的点，所有点权重之和为 m 。合并权重为 x 与 y 的点需要花费 $O(T(x,y))$ 的复杂度，得到一个权重为 $x+y$ 的点。

我们不妨设第 i 个点的权重为 s_i ，接下来根据 T 来考虑其复杂度。

若 $T(x,y)=min(x,y)$

这个情况很简单，只要按顺序依次合并既可。

考虑复杂度： $\sum_{i=2}^{n} O(min(x,s_i)) \le \sum_{i=2}^{n} O(s_i) = O(m)$

所以该情况的复杂度为： $O(m)$

若 $T(x,y)=x\times y$

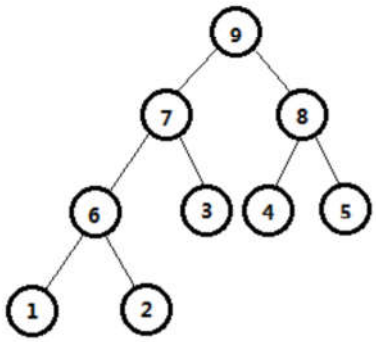
这种情况多数在树上问题出现。其实无论你按什么顺序合并，总复杂度都是 $O(m^2)$ 而非 $O(m^3)$ 的。关于这一点，我们可以把合成的点向合成它的点连边，整个结构就是一棵树， $T(x,y)$ 相当与枚举左右子树内所有点的匹配方案。这是你只需要注意到每一对点只会在它们 *LCA* 处枚举到就行了。

所以该情况的复杂度为： $O(m^2)$

若 $T(x,y)=x+y$

这个时候我们应该每次选取两个权值最小的点合并。

现在来考虑其复杂度。我们可以把每一个合成出来的点向合成它的点连边。这整个结构事实上就是一棵哈夫曼树。



考虑上图，其意义为：第一次选了权值最小的两条点1,2，合成了点6，第二次将点6,3 合成了点7，第三次将点4,5合成了点8，最后将点7,8合成了最终的点9。

由于 $T(x,y)=x+y$，因此整个算法的复杂度等于所有点的权值之和。

$$complexity=\sum O(s_i)=O(\sum s_i)=O(\sum_{i\ is\ a\ leaf} s_i\times dep_i)$$

其中 dep_i 为点 i 的深度。

现在我们来求叶子节点深度的上界，以1号点为例，可以推出：

$$\begin{aligned} s_9 &= s_7+s_8 && (1)\\ &\geq 2s_6+s_3 && (2)\\ &\geq 3s_1+2s_2 && (3)\end{aligned}$$

(1)式推到(2)是因为6在8之前合并，所以可以得到 $s_8\geq s_6$，(2)到(3)同理。

从推导中可以得知， s_x 前的系数是斐波拉契数列。因此推广到任意点 i 有：

$$s_{root} \geq Fib(dep_i) \times s_i + Fib(dep_i - 1) \times s_x$$

其中 $Fib(x)$ 为斐波拉契数列第 x 项。 s_{root} 即为所有点权重之和 m。因此可以得到 $dep_i \leq O(\log n)$

所以该情况的复杂度为： $O(m \log m)$

其实这种做法非常具有拓展性。下面来看一个小问题，希望可以起到抛砖引玉的作用。

例6. *(使用了2015集训互测Robot(原创)的主要方法)*

假如有 n 个变量，它们的DP数组都已经通过本文的另一个方法“将键值函数化”化为了一条条折线函数， n 条折线的总线段数为 m 条，现在有 Q 次询问，每次询问所有变量在某点处的最小值。

$n, m \leq 10^5, Q \leq 5 \times 10^5$

定理2.1. 假如一条折线 l 是由段数分别为 $a_1, a_2, \ldots, a_m$ 的折线取 min 而成的。则 l 的段数是 $O(\sum a_i)$ 的。

关于定理 2.1 的证明，由于篇幅有限所以可以详见 *Robot* 解题报告。

根据定理 2.1 ，我们在合并两条段数分别为 x 与 y 的折线时就可以认为合出来的是一条段数少于 $x + y$ 的折线了。由于两条段数分别为 x 与 y 的折线取 min 要使用 $O(x + y)$ 的复杂度，所以直接使用 $T(x, y) = x + y$ 时的做法，每次选两条交点数少的线段求一下 min 。由于求出了最终的折线，所以复杂度为 $O((m + Q)\log m)$ 。假如询问已经排好序了，那么复杂度可以降至 $O(m \log m + Q)$

3 总结

*DP*的优化方法种类繁多，变幻莫测。我们应当细心分析题目性质，选择合适的优化方法。

4 感谢

感谢CCF提供学习和交流的平台。

感谢绍兴一中的陈合力老师、游光辉老师、董烨华老师多年来给予的关心和指导。

感谢清华大学的俞鼎力，董宏华学长对我的帮助。

感谢绍兴一中的王鉴浩、任之洲、贾越凯等同学对我的帮助和启发。

感谢其他对我有过帮助和启发的老师和同学。

感谢我的父母二十年如一日无微不至的关心和照顾。

5 参考文献

[1] topcoder SRM526 Algorithm Problem Set Analysis

[2] topcoder SRM639 Algorithm Problem Set Analysis

[3] Codeforces Round #265 Editorial

[4] Codeforces Round #172 Editorial

[5] Robot解题报告, by 张恒捷