DP07解题报告

Tianyi Cui

November 12, 2007

首先给出每题简要的提示：

质数取石子 设P为质数集合。bool数组win[i]表示初始桌上有i个石子时DD能否赢。则win[i]=OR{NOT win[j]—i-j∈P}。设f[i]表示桌面上还有i个石子时还需要走的步数。则f[i]=opt{f[j]—i-j∈P}+1。其中opt当win[i]为true时为min，反之为max。

多人背包 求01背包的前K个最优解的题目，基本方法是将01背包的每个状态用一个单调队列表示，原状态转移方程中的max运算相当于单调队列的合并。

不听话的机器人 其实挺简单，状态f[i][x][y][d]表示前i条命令执行完之后，机器人在位置(x,y)，面朝方向d时，最少需要去掉的命令数。策略只有简单的两种：去掉当前这一条或者没有去掉。用滚动数组减少空间消耗。

新魔法药水 一个需要进行两次动态规划的题目。g[i][k]表示使用k次魔法得到一瓶第i种魔药需要的最少金币数。求解g[i][k]需要对每种配制魔药的方法进行资源分配类的动态规划。所有的g[i][k]都求出来以后，再求一个二维背包，魔法和金币分别是两重限制。

下面是比较详细的解题报告：

1 质数取石子

这是一个博弈论的题目，只需要用到博弈论中相当基础的一部分知识，我也希望你能通过这道题目掌握一些最基础的博弈论知识。

博弈论，也就是Game Theory，“游戏理论”。它研究的游戏中有一种称为Impartial Combinatorial Games（没有找到标准的翻译为何，“公正组合游戏”吗？），以下简称ICG，OI中遇到的很多博弈论题目都属于这个类型。

我希望通过讲解关于ICG的最基本理论，自然地得到本题的状态转移方程。

满足以下条件的游戏是ICG：

1. 有两名选手；
2. 两名选手交替对游戏进行移动(move)，每次一步，选手可以在（一般而言）有限的合法移动集合中任选一种进行移动；

3. 对于游戏的任何一种可能的局面(position)，合法的移动集合只取决于这个局面本身，不取决于轮到哪名选手操作、以前的任何操作、骰子的点数或者其它什么因素；

4. 如果轮到某名选手移动，且这个局面的合法的移动集合为空（也就是说此时无法进行移动），则这名选手负。

注意以上是直观一些的定义，可能不太严谨。根据这个定义，很多日常的游戏并非ICG。例如象棋就不满足条件3，因为红方只能移动红子，黑方只能移动黑子，合法的移动集合取决于轮到哪名选手操作。

本题中定义的“质数取石子”游戏属于ICG。可以看到，这个游戏中一种局面可以用桌面上的石子数来描述，不管是哪一方面对桌面上的若干石子，合法的移动集合总是相同的。

ICG的所有局面(position)可以被分为两类：N-position和P-position。直观的解释，N-position表示“轮到下一个移动的选手有必胜策略的局面”，或者说“先手有必胜策略的局面”，这里N代表next；P-position表示“上一步移动的选手有必胜策略的局面”，或者说“后手有必胜策略的局面”，P代表previous。更严谨的定义是：

1. 无法进行任何移动的局面（也就是terminal position）是P-position；

2. 可以移动到P-position的局面是N-position；

3. 所有移动都导致N-position的局面是P-position。

这个定义也还是比较好理解的。第1条的根据是ICG的定义第4条。第2条也很显然，当能够通过一步移动将当前的局面转化为上一步移动的选手必胜的P-postion时，那么这步移动事实上就是先手的必胜策略，所以当前局面是先手必胜的N-position。如果不管怎么移动，都导致下一步的先手有必胜策略的N-position，那么也就相当于这一步的先手必败了，所以当前的局面是P-position。

这个定义事实上是一个递归的定义，也就是说，如果我们知道了当前局面所有的后继局面（即通过移动可以导致的局面）是N还是P，我们也就可以判定当前局面是N还是P了。

注意这样的常见思路：递归->记忆化搜索->动态规划。所以，求解当前局面是先手还是后手必胜的问题可以采用动态规划解决。如果设win[i]表示局面i的先手是否必胜。一般的状态转移方程是这样的：

$$win[i] = OR\{NOT\ win[j] \mid \text{position } i \text{ can move to } j\}$$

其中OR表示或，NOT表示非，都是针对布尔值的操作。这个方程的意义是：如果当前局面的后继局面中有一个是win[j]为false的P-position，则当前局面是N-position，否则当前局面是P-position。

将这个通用的方程应用到本题中，就得到了本题求解DD是否能取得胜利的状态转移方程：

$$win[i] = OR\{NOT\ win[j] \mid i - j \text{ is a prime}\}$$

这里win[i]表示桌上有i颗石子时先手是否能取胜。

下面考虑如何求出一个先手必胜的局面（N-Position）中先手最快可以几步取得胜利，不管对手采用怎样的策略，或者等价地，如果对手采用尽量延迟失败的策略的话。

这事实上类似于“敌对搜索”的思想，一方希望自己胜利的时间尽量早，另一方希望对方胜利的时间尽量迟。设f[i]表示在局面i时还需要走的步数。如果这时轮到必胜一方走（f[i]是N-position），则他会使这个步数尽量小；反之，必败的一方会试图使这个步数尽量大。所以状态转移方程就是：

$$f[i] = opt\{f[j] \mid \text{position } i \text{ can move to } j\} + 1$$

其中opt取min或max，由win[i]取true或false决定。同样，将这个通用的方程应用到本题中：

$$f[i] = opt\{f[j] \mid i - j \text{ is a prime}\} + 1$$

于是本题就这样解决了。复杂度是$O(CP)$，其中C是桌面上的石子数，P是不大于C的质数数目。

如果你希望了解更多关于博弈论的知识，我强烈推荐加利福尼亚大学的Thomas S. Ferguson[1]教授精心撰写并免费提供的这份教材[2]。事实上，这就是我目前极其有限的博弈论知识的全部来源。很遗憾，它是英文的。不过，如果你的英文水平不足以阅读它，我只能说，恐怕你还没到需要看“博弈论”的时候。

2 多人背包

这个题目事实上就是求01背包问题的前K个最优解。出现在正式版的《背包问题九讲》的P09中。为完整起见，将那一段讲解摘录一下：

> 对于求次优解、第K优解类的问题，如果相应的最优解问题能写出状态转移方程、用动态规划解决，那么求次优解往往可以相同的复杂度解决，第K优解则比求最优解的复杂度上多一个系数K。
>
> 其基本思想是将每个状态都表示成有序队列，将状态转移方程中的max/min转化成有序队列的合并。这里仍然以01背包为例讲解一下。
>
> 首先看01背包求最优解的状态转移方程：
>
> $$f[i][v] = max\{f[i-1][v], f[i-1][v-c[i]] + w[i]\}$$
>
> 如果要求第K优解，那么状态f[i][v]就应该是一个大小为K的数组f[i][v][1..K]。其中f[i][v][k]表示前i个物品、背包大小为v时，第k优解的值。“f[i][v]是一个大小为K的数组”这一句，熟悉C语言的同学可能比较好理解，或者也可以简单地理解为在原来的方程中加了一维。显然f[i][v][1..K]这K个数是由大到小排列的，所以我们把它认为是一个有序队列。

[1] http://www.math.ucla.edu/ tom/

[2] http://www.math.ucla.edu/ tom/Game_Theory/Contents.html

然后原方程就可以解释为：f[i][v]这个有序队列是由f[i-1][v]和f[i-1][v-c[i]]+w[i]这两个有序队列合并得到的。有序队列f[i-1][v]即f[i-1][v][1..K]，f[i-1][v-c[i]]+w[i]则理解为在f[i-1][v-c[i]][1..K]的每个数上加上w[i]后得到的有序队列。得到这两个有序队列合并以后结果的前K项，储存到f[i][v][1..K]中的复杂度是$O(K)$。最后的答案是f[N][V][K]。总的复杂度是$O(NVK)$。

为什么这个方法正确呢？实际上，一个正确的状态转移方程的求解过程遍历了所有可用的策略，也就覆盖了问题的所有方案。只不过由于是求最优解，所以其它在任何一个策略上达不到最优的方案都被忽略了。如果把每个状态表示成一个大小为K的数组，并在这个数组中有序的保存该状态可取到的前K个最优值。那么，对于任两个状态的max运算等价于两个由大到小的有序队列的合并。

另外还要注意题目对于“第K优解”的定义，将策略不同但权值相同的两个方案是看作同一个解还是不同的解。如果是前者，则维护有序队列时要保证队列里的数没有重复的。

上面引用自《背包问题九讲》P09。由于本题相当于求解前K个最优解的和，最终答案是f[N][V][1..K]的和。

3 不听话的机器人

我觉得，如果这道题想不出来的话，很可能是对这种状态的定义方法不是太熟悉。f[i][x][y][d]表示前i条命令执行完之后，机器人在位置(x,y)，面朝方向d时，最少需要去掉的命令数。不要觉得这个四维状态的定义令人害怕，它事实上只相当于有两维，一维是考虑了前i条命令，另一维表示机器人当前的状态——位置和方向，只不过机器人当前的状态用了三个数来表示而已。

状态定义出来了，策略事实上是显然的，只有两种，去掉或者不去掉第i条命令。其中，当不去掉第i条命令就会导致危险时就只有一种策略。

由于转移事实上是要根据第i个命令来分情况讨论的，显式的状态转移方程不大好写，写出来了也启迪不大，所以这里就不给出了。

另外，由于状态空间很大，最好使用滚动数组来优化空间复杂度。不使用滚动数组的话很可能会超过本题设置的64M内存限制。

顺便提一下我的其中一份标程中实现状态转移的方法：并非考虑当前这个状态可以由哪些状态得来（一般的由状态转移方程得来的思路），而是考虑当前这个状态可以推知哪些状态。也就是说，当我们已经知道当前考虑了i条命令且处于某个位置的状态的值的时候，这个值可以影响到两个未知的状态——执行或者不执行第i+1条命令的结果。对于这道题来说，这样编程思路似乎顺一些。

这个题目曾经设计了第二问：问机器人有多少种去掉命令的方案。但最终经过权衡没有在题目中出现。你不妨思考一下。

整个程序的复杂度是$O(N^2M)$。

4 新魔法药水

在去年一次由OIBH命题组主办、我作为负责人的OIBH杯NOIP2006第二次模拟赛[3]中，我曾出过一道题名为“佳佳的魔法药水”。这道“新魔法药水”的最初灵感来源就是那道题，不过题目的思路和解法还是很有一些不同的。

这道题目需要进行两次动态规划，可以认为其中第二次DP的输入是第一次DP的输出。其实两次DP都是挺常见的模型：一个资源分配模型，一个二维背包模型，只不过把它们揉合到同一道题目中后便变得有些不易看出。

两次动态规划的桥梁，也是题目中最基本的一个状态是：g[i][k]表示使用k次魔法得到一瓶魔药i需要的最少金币数。求解g[i][k]必然要枚举最后一次得到魔药i使用的魔法是哪一个，这次魔法是一定要使用的。然后就是把剩下的k-1次魔法分配到最后的魔法需要的若干种原料中，也就是一个资源分配的问题了。资源分配问题的定义和算法可以参看我的文章《动态规划中的线型模型》。注意这里k客观起到了划分阶段、消除后效性的作用。

用资源分配求出了所有的g[i][k]之后，每个g[i][k]就相当于关于金币及魔法的二维背包问题中的一件物品。这件物品需要耗费的金币数是g[i][k]，需要耗费的魔法数是k，可以得到的价值是price[i]-g[i][k]。其中price[i]表示第i件物品的收购价。二维背包问题的定义和算法可以参看我的文章《背包问题九讲》。最后需要输出的就是这个二维背包问题的答案了。

两步动态规划的复杂度分别为$O(NMK^2)$和$O(N^2VK)$。

这道题还有若干难度更大的变体问题。为RP起见，没有作为本次比赛的题目，而采用了难度较低的原题。

变体1

> 题目描述同原题，要求输出方案。也就是说，要输出须购买的魔药清单、按顺序使用的魔法清单、最后卖掉的魔药清单。

变体2

> 去掉题目中每天使用魔法次数的上限，仅有初始可用来购买原材料金币数的限制。

变体3

> 若DD可以在一天内多次往返于魔法商店和家中，可以多次购买和卖出魔药。也就是说，DD可以先用有限的金币去买一些原料，在家做成成品，卖掉以赚一些金币，再买更贵的原料，再在家作更有价值的成品……

变体1的思路与本题相同，只不过输出这个问题的方案是极其麻烦和极其ws的，实在是一个相当大的挑战。事实上，本题的最初设计就要求输出方案，只不过我在写标程过程中发现实在太ws，就改成了只需输出一个数。

变体2比原题目少了一个限制，最后转化成的背包问题是一维的，似乎会简单些。事实上，这会使本题的基本思路“按使用魔法的次数划分阶段”变得更加隐蔽且不易想到。容易设计出“g[i][j]表示第i种物品使用j个金币购买原材料可以获得的最低价格”之类的状态。适当规定数据范围就可以使这种状态设计变得不可行，而是只有按照原题的状态设计方法作才可以。这的确是本题的一个难度相当大的变体。

[3] http://oibh.org/bbs/viewthread.php?tid=11505

至于变体3，不妨留作这次比赛的终极赛后练习题，希望你仔细思考和实现。若你将程序实现出来了，交给我，我会帮你测试。第一个AC的选手将会有意想不到的神秘礼品一份……

---

[4] http://www.gnu.org/licenses/fdl.txt