

1D/1D 动态规划优化初步

所谓 1D/1D 动态规划,指的是状态数为 $O(n)$,每一个状态决策量为 $O(n)$ 的动态规划方程。直接求解的时间复杂度为 $O(n^2)$,但是,绝大多数这样的方程通过合理的组织与优化都是可以优化到 $O(n \log n)$ 乃至 $O(n)$ 的时间复杂度的。这里就想讲一讲我对一些比较初步的经典的优化方法的认识。

本文中不想进行过多的证明与推导，主要想说明经典模型的建立、转化与求解方法。

由于本人认识与水平相当有限，如果出现什么错误与疏漏，还请大牛多多指正。另外，也希望大牛们更多地向我们介绍一下有关动态规划优化的更深入的东西。

本文中使用两种方式表示一个函数： $f(x)$ 与 $f[x]$ ，用方括号表示的函数值可以在规划之前全部算出（常量），而用圆括号表示的函数值必须在规划过程中计算得到（变量）。无论是什么函数值一经确定，在以后的计算中就不会更改。

经典模型一: $f(x) = \min_{i=1}^{x-1} \{f(i) + w[i, x]\}$

相信这个方程大家一定是不陌生的。另外，肯定也知道一个关于决策单调性的性质：

假如用 $k(x)$ 表示状态 x 取到最优值时的决策, 则决策单调性表述为:

$$\forall i \leq j, \quad k(i) \leq k, \quad \text{当且仅当:}$$

$\forall i \leq j, w[i, j] = w[i+1, j-1] + w[i, j]$, 对于这个性质的证明读者可以在

任意一篇讲述四边形不等式的文章中找到，所以这里不再重复。而且，从实战的角度来看，我们甚至都不需要验证 w 函数的这个性质，最经济也是最可靠的方法是写一个朴素算法打出决策表来观察（反正你总还是要对拍）。当然，有的时候题目要求你做一点准备工作，去掉一些明显不可能的决策，然后在应用决策单调性。这是上述性质也许会有点用处。

正如前文中所述，我们关注的重点是怎样实现决策单调性。有了决策单调性，怎样高效地实现它呢？很容易想到在枚举决策的时候，不需要从 1 开始，只要从 $k(x-1)$ 开始就可以了，但这只能降低常数，不可能起到实质性的优化。

另一种想法是从 $k(x-1)$ 开始枚举决策更新 $f(x)$ ，一旦发现决策 u 不如决策 $u+1$ 来得好，就停止决策过程，选取决策 u 作为 $f(x)$ 的最终决策。这样时间是很大提高了，但可惜是不正确的。决策单调性并没有保证 $f(j)+w[j,x]$ 有什么好的性质，所以这样做肯定是不对的。

刚才我们总是沿着“ $f(x)$ 的最优决策是什么”这个思路进行思考，下面我们换一个角度，思考对于一个已经计算出来的状态 $f(j)$ ，“ $f(j)$ 能够更新的状态有哪些”。这样，每一步过程中某些状态的决策可能不是最优的，但是当算法结束的时候所有状态对应的决策一定是最优的。

一开始，只有 $f(1)$ 的函数值被计算出来，于是所有状态的当前最优决策都是 1。

[illegible]

现在，显然 $f(2)$ 的值已经确定了：它的最有决策只能是 1。我们用决策 2 来更新这个决策表。由于决策单调性，我们知道新的决策表只能有这样的形式：

例题 2：土地购买

题目来源：USACO Monthly, March, 2008, Gold

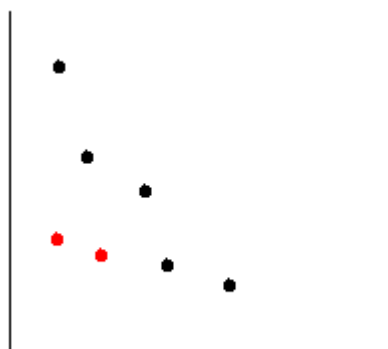
题目大意：有 N 块土地需要购买，每块土地都是长方形的，有特定的长与宽。你可以一次性购买一组土地，价格是这组土地中长的最大值乘以宽的最大值。比方说一块 5×3 的土地和一块 2×9 的土地在一起购买的价格就是 9×3 。显然，怎样分组购买土地是一门学问，你的任务就是设计一种方案用最少的钱买下所有的土地。

分析：将所有土地按照长度降序排列，依次检索，则当前土地的长度必然在上一块土地之内，我们只需要考虑宽度就可以了。而在宽度的问题上，当前土地的行为只能是这样：和前面若干块土地绑定；同时这些绑定的土地和他们前后的土地分离。这样很容易得出状态转移方程：

$$f(n) = \min_{k=0}^{n-1} \{ (\max_{i=k+1}^n w[i]) * l[k+1] + f(k) \}$$

这个方程还不能满足决策单调性，下面我们试图再做一下简化。

如果将每一个土地的尺寸看成是一个二维坐标的话，（如下图）



其中不难看出，红色点完全可以忽略，这些点 (x, y) 必然满足一个性质：存在点 (x', y') 同时满足 $x' \geq x$ 且 $y' \geq y$ ，这样它就能被一个组完全覆盖。这些被忽略的点可以通过一次线形的扫描得出。

下面，我们着重来看一下不能被忽略的这些点，它们的排布方式必然是单调减。因此状态转移方程可以写成这个样子：

$$f(n) = \min_{k=0}^{n-1} \{ x[n] * y[k+1] + f(k) \}$$

这个转移方程就是标准的决策单调性了，读者可以通过 w 函数的性质直接证明它。然后，就用上文中的方法在 $O(n \log n)$ 时间内求解。

以上两个例子都是决策单调性的直接应用。其中第二个例子稍微复杂一些，如果不忽略那些“肯定无用”的决策，不对数据进行有序化，则方程是不满足决策单调性的。这也就提醒我们在做这一类题目的时候不能钻牛角尖死做，还得灵活一点。

另外，决策单调性提供的只是 $O(n \log n)$ 的算法，事实上上面两个例题的最佳算法都是 $O(n)$ 的，在后文中我们将详细介绍另外一种经典模型，并且试图将这两个规划方程通过数学变换转向另一个模型。

下面我们来看一类特殊的 w 函数： $\forall i \leq j < k, w[i, j] + w[j, k] = w[i, k]$ ，显然，这一类函数都是满足决策单调性的。但是不同的是，由于这一类函数的特殊性，他们可以用一种更加简洁也更加有借鉴意义的方法解决。

由于 w 函数满足 $\forall i \leq j < k, w[i, j] + w[j, k] = w[i, k]$ ，我们总是可以找到一个特定的一元函数 $w'[x]$ ，使得 $\forall i \leq j, w[i, j] = w'[j] - w'[i]$ ，这样，假设状态 $f(x)$ 的某一个决策是 k ，有：

$$f(x) = f(k) + w[k, x] = f(k) + w'[x] - w'[k], \text{ 其中 } g(k) = f(k) - w'[1, k]。$$

$$= g(k) + w'[x] - w'[1],$$

这样我们发现：一旦 $f(k)$ 被确定，相应地 $g(k)$ 也被确定，更加关键的是，无论 k 值如何， $w'[x] - w'[1]$ 总是一个常数。换句话说，我们可以把方程写成下述形式：

$f(x) = \min_{k=1}^{x-1} \{g(k)\} + w[1, x]$ 。不难发现这个方程是无聊的，因为 $\min_{k=1}^x \{g(k)\}$ 我们可以用一个变量“打擂台”直接存储；但是，如果在 k 的下界上加上一个限制，那这个方程就不是很无聊了。于是，我们就得到了另一个经典模型。

经典模型二： $f(x) = \min_{k=b[x]}^{x-1} \{g(k)\} + w[x]$ ，其中， $b[x]$ 随 x 不降。

这个方程怎样求解呢？我们注意到这样一个性质：如果存在两个数 j, k ，使得 $j \leq k$ ，而且 $g(k) \leq g(j)$ ，则决策 j 是毫无用处的。因为根据 $b[x]$ 单调的特性，如果 j 可以作为合法决策，那么 k 一定可以作为合法决策，又因为 k 比 j 要优，（注意：在这个经典模型中，“优”是绝对的，是与当前正在计算的状态无关的），所以说，如果把待决策表中的决策按照 k 排序的话，则 $g(k)$ 必然是不降的。

这样，就引导我们使用一个单调队列来维护决策表。对于每一个状态 $f(x)$ 来说，计算过程分为以下几步：

- 1、队首元素出队，直到队首元素在给定的范围中。
- 2、此时，队首元素就是状态 $f(x)$ 的最优决策，
- 3、计算 $g(x)$ ，并将其插入到单调队列的尾部，同时维持队列的单调性（不断地出队，直到队列单调为止）。

重复上述步骤直到所有的函数值均被计算出来。不难看出这样的算法均摊时间复杂度是 $O(1)$ 的。

下面我们来看几个例题。

例题 3：The Sound of Silence

题目来源：Baltic Olympiad in Informatics 2007

题目大意：给出一个 N 项的数列，如果对于一个连续的长度为 M 的片段来说，片段内所有数中最大值与最小值的差不超过一个给定的常数 C ，则我们称这样的片段是一个合法的片段。编程求出所有的合法片段的起始位置。

分析: 本题不难看出可以分解为两个子问题: 求所有片段的最大值以及求所有片段的最小值。而这两个任务实际上是一样的, 所以我们只要求取所有的连续 M 个数的片段中的最小值。

这个任务有很多很多种对数级算法, 其中用堆或者用静态最优二叉树都可以做到 $O(n \log m)$, 但是这题的 $O(n)$ 算法还是不那么好想的。

事实上, 如果用 $g[x]$ 表示数列中第 x 个数的值, 用 $f(x)$ 表示以 x 作为**结尾**的有 M 个数的连续片段的话, 显然有:

$$f(x) = \min_{i=x-M+1}^x g[i], \text{ 直接吻合经典模型二。套用算法, 就可以在 } O(n) \text{ 的时间内解决问题。}$$

(当然, 本题还有一种别致的“窗口”算法, 也漂亮地在 $O(n)$ 的时间内解决了问题, 详细可以看官方的解题报告。这里引入本题的主要目的是在于佐证上文中讨论到的经典模型二)

例题 4: Islands

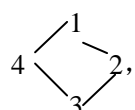
题目来源: IOI2008

题目大意: 给出一个具有 N 个顶点的无向加权图, 同时这个图中有且恰有 N 条边。现在, 对于这个图中的每一个连通分量, 求出其最长路径 (权值和最大, 一个节点在路径上最多只能出现一次)。

分析: 当然, 这个问题更多的是一个图论题。但是在最后关键问题的处理上还是可以看到经典模型二的影子。

首先, 用 BFS 找出所有连通分量。然后, 对于一个连通分量来说, 由于点数与边数相同, 因此必然构成**基环+外向树**的结构。我们可以找出基环并确定所有外向树的结构。一条最长路径有两种可能: 完整地处于某一棵外向树中; 或者位于两棵外向树中, 其间通过基环的一段连接。第一种可能可以通过树形 DP 解决, 问题就在于第二种可能怎样处理。如果枚举两棵外向树, 那就是 $O(n^2)$, 就不可以接受了。

我们考虑破环为链, 然后将链整体左移, 制作一个副本。比方说, 如果原来的环是:



以 1 为首破环, 得: 1--2--3--4, 然后制作副本, 得 2--3--4--1--2--3--4, 制作副本

的主要目的是使得对于每一个点的方程都有统一的形式, 使得环上所有片段都可以对应链上的一个片段。这种情况下, 用 $g[x]$ 表示 x 点上外向树上的最长下降路的长度, $f(x)$ 表示以该点为终点的总最长路径的长度, 则有:

$$f(x) = \max_{i=x-M+1}^{x-1} \{g[i] + g[x] + dis[i, x]\}, \text{ 其中 } w \text{ 函数即 } w[i, j] = dis[i, j], \text{ 显然满足}$$

$\forall i \leq j < k \quad w[i, j] + w[j, k] = w[i, k]$, 通过变换之后就可以变成经典模型二。这样, 就在总 $O(n)$ 的时间内解决了本题。

如果还嫌以上两个问题不够典型, 下面举一个典型到所有 OIer 都耳熟能详的题目。

例题 5: 有限背包问题。

题目来源: 经典问题。

题目大意: 有 N 件物品, 每一件物品的价值为 $p[k]$, 重量为 $w[k]$, 最多只能选取 $m[k]$ 次; 现在给出背包的最大承重量 C , 要求在满足重量要求的条件下使得背包中的物品价值总和和最

大。

分析：如果 $m[k] = 1$ 或者 $m[k] = +\infty$ ，就都很好做。但现在 $m[k]$ 是一个有界值，就比较麻烦了。

我们还是按照背包问题的常见思路，一次枚举每一个物品。设当前枚举的物品编号为 k ，用 $f(x)$ 表示：为了到达价值 x ，背包的重量至少应该是多少；则我们有：

$f(x) = \min_{i=1}^{m[k]} \{f(x-i \cdot p[k]) + i \cdot w[k]\}$ ，这个方程很麻烦，因为某一个状态的决策不是连续的，而是间断性的。怎样把决策区间变成一段连续区间呢？很容易想到**等价类**的思想；如果按照模 $p[k]$ 对所有的 $f(x)$ 划分等价类，那么在同一个等价类中，决策区间就是连续的了，我们不妨把新函数设为 $h(x)$ ，则方程变为：

$h(x) = \min_{i=x-m[k]}^{x-1} \{h(i) + w[k] \cdot (x-i)\}$ ，其中， w 函数即 $w[i, j] = w'[k] \cdot (j-i)$ 显然满足

$\forall i \leq j < k, w[i, j] + w[j, k] = w[i, k]$ ，（注意 $w'[k]$ 是一个与 i 和 j 无关的常量）经过适当的变化后可以转化为经典模型二。于是有限背包问题可以在 $O(NP)$ 的时间内解决，其中 P 是背包可能取到的最大价值。（其实换成重量也一样），这也就是“背包十讲”中所说的那个单调队列法。

我们注意到，如果 $m[k]=1$ 的话，那么每一个 $f(x)$ 的决策量都是 $O(1)$ ，这没什么问题；但如果 $m[k] = +\infty$ ，意味着什么呢？仔细观察可以发现，这实际上就拿掉了方程中的循环变量的下界，对应的是 $f(x) = \min_{k=1}^{x-1} \{g(k)\} + w[1, x]$ 这样的一个方程，这显然是很简单的，适用单变量打擂台就可以解决了（尽管我们通常并不这样做）。所以说，借助经典模型二，我们在一个更高的高度上统一了 0-1、有限、无限三大背包问题。

下面我们再次来看一下例题 2《土地购买》中的那个方程：

$f(n) = \min_{k=0}^{n-1} \{f(k) + x[n] \cdot y[k+1]\}$ ，我们来仔细地观察这个方程： $f(k)$ 是变量， $y[k+1]$ 是常量，但不论怎么说，这两个量在以后的计算中都不会变化。而 $x[n]$ 是一个比例系数，它与 k 无关，只随着 x 的变化而变化。

如果我们建立平面直角坐标系，以 $f(k)$ 作为横轴， $y[k+1]$ 作为纵轴，则每一个状态 $f(k)$ 都可以看作是该坐标系中的一个点。在求解状态 $f(n)$ 的过程中，我要求最小化：

$\min P = x + ky$ ，其中 x, y 是我建立的直角坐标系中某一个点的坐标（表示一个决策）， k

就是方程中的 $x[n]$ ，是只与 n 有关，而与决策无关的一个常量。

这个最小化问题是什么呢？其实就是一个平面上的线性规划。我们把式子改写成：

$y = -\frac{1}{k}x + \frac{P}{k}$ ，就演变成了这样的问题：在一个直线簇 $y = -\frac{1}{k}x + C$ 中，选取一条直线，使得这条直线过某个给出的数据点，同时 C 要最小。

既然问题变成了这么有意思的线性规划问题，就有必要进一步的研究，看看是不时有更好的解法，这就导致了我们的另一个经典模型：

经典模型三： $f(x) = \min_{i=1}^{x-1} \{a[x] * f(i) + b[x] * g(i)\}$ 。

注意：这个模型写的比较抽象，其实它的涵盖范围是很广的。首先， $a[x], b[x]$ 不一定是常量，只要他们与决策无关，都可以接受；另外， $f(i)$ 和 $g(i)$ 不管是常量还是变量都没有关系，只要他们是一个有最优的 $f(x)$ 决定的二元组就可以了。

因此，为了方便描述，我们把这个模型写成下面这个形式：

$$f(n) = \min_{i=1}^{n-1} \{a[n] * x(i) + b[n] * y(i)\},$$

其中， $x(i), y(i)$ 都是可以在常数时间内通过 $f(i)$ 唯一决定的二元组。

这个经典模型怎样转化求解呢？前文说过，这样的模型的求解与平面上的线性规划有关，我们以 $x(i)$ 为横轴， $y(i)$ 为纵轴建立平面直角坐标系，这样一个状态 $f(i)$ 所决定的二元组就可以用坐标系中的一个点表示。然后，我们的目标是：

$\min P = ax + b$ ，其中 $a = a[n], b = b[n]$ ，化成： $y = -\frac{a}{b}x + \frac{P}{b}$ ，假设 $b > 0$ （反之亦然），则我们的任务是使得这条直线的纵截距最小。可以想象有一组斜率相同的直线自负无穷向上平移，所碰到的第一个数据点就是最优决策。

这个时候，有一个重要的性质，那就是：**所有最优决策点都在平面点集的凸包上**。基于这个事实，我们可以开发出很多令人满意的算法。

这时，根据直线斜率与数据点分布的特征，可以划分为两种情况：

情况一：决策直线的斜率与二元组的横坐标同时满足单调性。（具体的单调性视最优化目标的性质而定）

这样的模型是比较好处理的，因为这个时候由于斜率变化是单调的，所以决策点必然在凸壳上单调移动。我们只需要维护一个单调队列和一个决策指针，每次决策时作这样几件事：

- 1、决策指针（也就是队首）后移，直至最佳决策点。
- 2、进行决策。
- 3、将进行决策之后的新状态的二元组加入队尾，同时作 **Graham-Scan** 式的更新操作维护凸壳。（注意此时当前指针所在二元组有可能被抛弃）

算法的时间复杂度为 $O(n)$

情况二：没有任何限制。

这时问题的解决就比较困难了。显然，决策点还是应该在凸壳上。我们不妨考虑一个单调减的凸壳，这个凸壳上点与点之间的连线必然满足这样的性质：斜率单调减。通过细致的观察我们可以发现，**对于一个给定的斜率 k 来说，对应的直线簇中具有最大纵截距的直线通过的决策点必然满足这样的性质：该点两侧的边的斜率 k_1, k_2 满足 $k_1 \geq k \geq k_2$ 。**

这样，我们就可以通过二分查找来确定最佳的决策点。

但是，在插入数据点的过程中，我们遇到的麻烦可能更大。首先，肯定是二分查找确定横坐标的插入点，然后对两侧分别进行 **Graham** 维护凸性。但接下来的问题就严重了：在维护凸形的过程中我们肯定删掉了一些点，怎样重新得到一个完整的凸壳决策表呢？使用 **move**

是一个折中的办法，但是这与理论的时间复杂度分析根本无益。

完美的解决方法是应用平衡二叉树。我们以横坐标为关键字建立平衡二叉树，这样查找和插入的过程都可以在 $O(\log n)$ 时间内完成。当我们做 Graham 维护时，首先将新数据点 Splay 到根节点，此时剩下的节点必然分居左子树和右子树。然后，我们以左子树为例，后序遍历依次查找节点，直至查找到一个满足凸形的节点。将这个节点 Splay 到根节点的左孩子，然后删掉这个节点的右孩子。这样的算法的时间复杂度是 $O(n \log n)$ ，但是实现起来非常复杂。

下面我们来看几个例题。

例题 6：《玩具装箱》的线性算法。

例题 1 中《玩具装箱》的动态规划方程为：

$$f(x) = \min_{i=1}^{x-1} \{ f(i) + (x - i - 1 + \sum_{k=i+1}^x c_k - L)^2 \},$$

下面，我们试图通过数学变换将其变成经典模型三。

为了简化计算，设 $sum[x] = \sum_{i=1}^x c[i]$ ，则：

$$\begin{aligned} f(x) &= \min_{i=1}^{x-1} \{ f(i) + (x - i - 1 - L + sum[x] - sum[i])^2 \} \\ &= \min_{i=1}^{x-1} \{ f(i) + (sum[x] - sum[i] + x - L - 1)^2 \}, \end{aligned}$$

不妨设 $a[x] = sum[x] + x - L - 1, b[i] = sum[i] + i$ ，显然这两个量都是常量，则：

$$\begin{aligned} f(x) &= \min_{i=1}^{x-1} \{ f(i) + (a[x] - b[i])^2 \} \\ &= \min_{i=1}^{x-1} \{ f(i) + a^2[x] + b^2[i] - 2a[x]b[i] \} \\ &= \min_{i=1}^{x-1} \{ f(i) + b^2[i] - 2a[x]b[i] \} + a^2[x] \end{aligned}$$

然后问题就明朗了，设平面直角坐标系中 $x(i) = b[i], y(i) = f(i) + b^2[i]$ ，则问题变成：

$\min P = 2ax + y$ ，其对应的线性规划的目标直线为 $y = -2ax + P$ 。

回顾定义不难看出， $a[x]$ 随着 x 的增大而增大， $x(i)$ 也随着 i 的增大而增大。因此，问题中直线斜率单调减，数据点横坐标单调增，符合经典模型三种的情形 1。使用单调队列维护凸壳可以在 $O(n)$ 的时间内解决本题。

例题七：货币兑换。

题目来源：NOI2007

题目大意：有 3 种货币体系：人民币，A 券，B 券，其中 A 券与 B 券的价格在每一天都是不同的。在某一天 D，你可以做 3 件事情：

- 1、如果你的手头上有 A 券或 B 券，你可以将它们都按照当天的价格换成人民币。
- 2、如果你的手头上有人民币，你可以将它们按照一个特定的比例 Rate 并以当天的价格换成 A 券和 B 券（就是说你兑换的 A 券和 B 券的价值比是 Rate）

3、什么也不做。

一开始你有一些人民币，请你通过最佳的操作方式在最后一天结束的时候手头上握有最多的人民币。

分析：试题中的 Hint 已经告诉我们，**如果我们想买进人民币，就必须全额买进；如果我们想卖出人民币，就必须全额卖出**。由于不管是在哪一天，人民币总是越多越好；我们用 $f(n)$ 表示到了第 n 天最多可能持有的人民币数量，用 $x(i)$ 和 $y(i)$ 表示在第 i 天，用最多的钱能够换来的 A 券和 B 券。（注意：由于 Rate 确定，兑换金额确定，所以 A 券和 B 券的数量是唯一确定的），我们有：

$$f(n) = \max_{i=1}^{n-1} \{a[n] * x(i) + b[n] * y(i)\}, \text{ 其中 } a[n] \text{ 和 } b[n] \text{ 代表 A 券和 B 券在第 } n \text{ 天的牌价。}$$

这个方程式符合经典模型三中的情形二。所以，我们应该使用一个平衡树来维护凸形。时间复杂度是 $O(n \log n)$ 。由于数据弱，所以这道题目用 move 也是可以搞定的。

这篇文章中，我们着重讨论了这样三类经典模型的建立与求解过程：

经典模型一： $f(x) = \min_{i=1}^{x-1} \{f(i) + w[i, x]\}$ ， $w[i, x]$ 满足决策单调性。

经典模型二： $f(x) = \min_{i=b[x]}^{x-1} \{g(i)\} + w[x]$ ，其中 $b[x]$ 单调增。

经典模型三： $f(n) = \min_{i=1}^{n-1} \{a[n] * x(i) + b[n] * y(i)\}$ ，其中 $x(i)$ ， $y(i)$ 可以由 $f(i)$ 在常数时间内唯一确定。

这三类模型都可以在至少 $O(n \log n)$ 的时间内解决，从而起到了对 1D/1D 的方程的优化作用。另外，这三种模型并不是孤立存在的，而是可以互相转化的，文中的很多例题就兼具多种模型的特点。

当然，本文只是对 1D/1D 动态规划优化的很初步很浅显的探讨，还有一些问题值得深入研究：

- 1、在经典模型一中，是否存在对于任意满足决策单调性的 w 函数都适用的 $O(n)$ 的算法？我们只给出了 $O(n \log n)$ 的通用算法以及对于一些特殊 w 函数的 $O(n)$ 算法，希望能够获得通用的简洁的 $O(n)$ 算法。
- 2、在经典模型一和经典模型二中，我们都可以给决策范围加上一个下界 $b[x]$ 而丝毫不影响时间效率，只要 $b[x]$ 单调递增的话。但是，在经典模型三中，由于算法依赖的策略选择顺序是基于方程本身的 $x(i)$ 和 $y(i)$ 所构成的平面的，与状态顺序并无关系，所以说，在这样的模型中如果加入 $b[x]$ 元素，还是否能够得到优化的算法，是一个问题。
- 3、上文中讨论的 1D/1D 方程都有一个显著特征：可选决策集是连续的。即使不是从 1 开始，而是从 $b[x]$ 开始，但至少也是连续的。而事实上，很多经典的 1D/1D 方程不满足这个特征，比如 LIS 问题：只有关键字比当前关键字小的决策才是合法决策。这就完全不能用上文中描述的思路进行优化。对于更一般的 1D/1D 动态规划方程的优化方法，值得探究。

- 4、1D/1D 方程还是有其局限性，毕竟状态总数是一维的。而在实战中，我们往往可以接触到不少 2D/0D 的方程，比如 LCS，《田忌赛马》，《建筑抢修》，《泡泡糖》之类的题目。对于这样的方程的优化方法显然远远不如 1D/1D 方程的优化方法丰富。它们的优化思路也是有待大牛研究的重要课题。
-

一些练习题。

- 1、用两种方法实现《玩具装箱》以及《土地购买》

2、《瑰丽华尔兹》

题目来源：NOI2005

题目大意：有一个 $N \times M$ 的棋盘，棋盘上有一些地方是空地，有一些地方是障碍物。现在，你从一个固定的起点开始沿着棋盘滑行。滑行的规则是这样的：在每一个时间点上，你可以选择原地不动，也可以选择按照一个特定的方向滑行一步。当然，你滑行的时候不能撞到障碍物。现在给你首尾相连的 K 个时间区间，每一个时间区间内的每一个时间点上的滑行方向都是一样的，你可以选择滑行或是不动。问你在所有的时间区间结束之后，你最多可以滑行多少步。（格子可以重复通过）

你应当给出 $O(NMK)$ 的算法。

（提示：本题可以直接对应一个有限背包问题）

3、《Cactus Reloaded》

题目来源：NEERC2007，上海市选 2008，POJ 3567

题目大意：一个具有 N 个节点， M 条边的仙人掌图是指这样一个连通无向图，图中每一条边最多属于一个环。现在，要求你求出图中任意两个点之间最短路径的最大值。你应该给出 $O(N+M)$ 的算法。

(Pascal 语言中)你可以打开 `{M 5000000}` 编译开关来进行递归而不需要考虑栈溢出。

- 4、用平衡树实现《货币兑换》