用单调性优化动态规划

【摘要】

单调性作为一类重要的性质，在信息学竞赛中是一种极为常见的解题突破口，也在动态规划的优化过程中起着至关重要的作用。本文主要选取了几道国内竞赛试题，探讨单调性在动态规划优化中神奇的应用。

【关键字】

单调性　动态规划　队列　凸线

【目录】

【正文】

一. 什么是单调（双端）队列

单调队列，顾名思义，就是一个元素单调的队列，那么就能保证队首的元素是最小（最大）的，从而满足动态规划的最优性问题的需求。

单调队列，又名双端队列。双端队列，就是说它不同于一般的队列只能在队首删除、队尾插入，它能够在队首、队尾同时进行**删除**。

【单调队列的性质】

一般，在动态规划的过程中，单调队列中每个元素一般存储的是两个值：

1. 在原数列中的位置（下标）
2. 他在动态规划中的状态值

而单调队列则保证这两个值同时**单调**。

【单调队列有什么用】

我们来看这样一个问题：一个含有 n 项的数列(n<=2000000)，求出每一项前面的第 m 个数到它这个区间内的最小值。

这道题目，我们很容易想到线段树、或者 st 算法之类的 RMQ 问题的解法。但庞大的数据范围让这些对数级的算法没有生存的空间。我们先尝试用动态规划的方法。用 $f(i)$ 代表第 i 个数对应的答案，$a[i]$ 表示第 i 个数，很容易写出状态转移方程：

$$f(i) = \mathop{Min}_{j=i-m+1}^{i}(a[j])$$

这个方程，直接求解的复杂度是 $O(nm)$的，甚至比线段树还差。这时候，单调队列就发挥了他的作用：

我们维护这样一个队列：队列中的每个元素有两个域$\{position,value\}$，分别代表他在原队列中的位置和 $a[i]$，我们随时保持这个队列中的元素两个域都**单调递增**。

那计算 $f(i)$ 的时候，只要在队首不断删除，直到队首的 $position$ 大于等于 $i-m+1$，那此时队首的 $value$ 必定是 $f(i)$ 的不二人选，因为队列是**单调**的！

我们看看怎样将 $a[i]$ 插入到队列中供别人决策：首先，要保证 $position$ 单调递增，由于我们动态规划的过程总是由小到大（反之亦然），所以肯定在队尾插入。又因为要保证队列的 $value$ 单调递增，所以将队尾元素不断删除，直到队尾元素小于 $a[i]$ 。

【时间效率分析】

很明显的一点，由于每个元素最多出队一次、进队一次，所以时间复杂度是$O(n)$。用单调队列完美的解决了这一题。

【为什么要这么做】

我们来分析为什么要这样在队尾插入：为什么前面那些比 $a[i]$ 大的数就这样无情的被枪毙了？我们来反问自己：他们活着有什么意义？！由于$i-m+1$是随着 i 单调递增的，所以对于 $\forall j<i,a[j]>a[i]$，在计算任意一个状态$f(x),x>=i$ 的时候，j 都不会比 i 优，所以 j 被枪毙是“罪有应得”。

我们再来分析为什么能够在队首不断删除，一句话：$i-m+1$是随着 i 单调递增的！

【一些总结】

对于这样一类动态规划问题，我们可以运用单调队列来解决：

$f(x)=\mathop{opt}\limits_{i=bound[x]}^{x-1}(const[i])$ 其中 bound[x]随着 x **单调不降**，而 const[i]则是可以根据 i 在常数时间内确定的**唯一的常数**。这类问题，一般用单调队列在很优美的时间内解决。

【一些简单的例子】

【例一】生产产品 Product[1]

•问题描述

有 n 个产品，编号为 1~n。要在 m 个机器人的手中生产完成。其中，第 i 个产品在第 j 个机器人手中的生产时间给定为 $T[i,j]$。要把这些产品按照编号从小到大生产，同一个机器人连续生产的产品个数不能够超过给定的常数 l。求生产完所有产品的最短时间是多少。其中 $n<=10^5$，$m<=5$，$l<=5*10^4$。

•解法分析

这道题目，很容易想到一个动态规划的算法：用 $f[i,j]$ 表示前 i 个产品，其中第 i 个产品实在第 j 个机器人上完成的，前 i 个机器人生产完成所需最短时间。

$f[i,j]=\mathop{Min}\limits_{k=i-l}^{i-1}(f[k,p]+sum[i,j]-sum[k,j])$,其中 $p<>j$。$sum[x,y]$表示产品 1 到 x 在 y 机器上生产所需的时间和。这样的算法时间复杂度是 $O(n*m^2*l)$，很明显不能在时限内完成，需要进行优化。

我们由于 j 的取值范围十分小：$1<=j<=5$。我们可以从这里突破。我们尝试把每一个 j **单独考虑**，比如现在只考虑 $j=1$ 的情况：那每一个决策 k，他为当前要计算的状态所提供的值是 $Min(f[k,p])+sum[i,1]-sum[k,1]$，$p<>j$。我们将其进行稍微的变形：$f[i,1]=Min(f[k,p]-sum[k,1])+sum[i,1]$，可以发现括号内的式子是根据 k 所确定的一个常量，直接对应上文的单调队列的方法，省掉

了枚举决策的一维，时间复杂度降为 $O(n*m^2)$。

1 选自 Vijos 1243

【例二】Cut the sequence[2]

•问题描述

给定一个有 n 个非负整数的数列 a，要求将其划分为若干个部分，使得每部分的和不超过给定的常数 m，并且所有部分的**最大值的和**最小。其中 n<=10^5。

例：n=8, m=17，8 个数分别为 2 2 2 | 8 1 8 |1 2，答案为 12，分割方案如图所示。

•解法分析

刚开始拿到这道题目，首先要读好题：**最大值的和最小**。

首先设计出一个动态规划的方法：

$f(i)=\mathop{Max}\limits_{j=b[x]}^{i-1}(f[j]+Maxnumber[j+1,i])$，其中 $f(i)$ 代表把前 i 个数分割开来的最小代价。$b[i]=Min(j\mid sum[j+1,i]<=m)$，$b[i]$ 可以用二分查找来实现。

直接求解复杂度最坏情况下（M 超大）是 $O(n^2)$ 的，优化势在必行。

通过仔细观察，可以发现以下几点性质：

1 在计算状态 $f(x)$ 的时候，如果一个决策 k 作为该状态的决策，那么可以发现第 k 个元素和第 x 个元素是不分在一组的。

2 $b[x]$ 随着 x **单调不降**的，用这一点，可以想到什么？可以想到前面单调队列的一个限制条件。

3 来看一个最重要的性质：如果一个决策 k 能够成为状态 $f(x)$ 的最优

决策，**当且仅当** $a[k] > \forall a[j], j \in [k+1, x]$。为什么呢？其实证明**非常非常**容易（用到性质 1），交给读者自己考虑。

到此为止，我们可以这样做：由于性质三，每计算一个状态 $f(x)$，它的有效决策集肯定是一个**元素值单调递减**的序列，我们可以像单调队列那样每次在队

2 选自 Pku3017

首删除元素，直到队首在数列中的位置小于等于 $b[x]$，然后将 $a[x]$插入队尾，保持队列的元素单调性。

这时候问题来了，队首元素一定是最佳决策点吗？我们只保证了他的元素值最大……如果扫一遍队列，只是常数上的优化，一个递减序足以将它否决。

我们观察整个操作，将队列不断插入、不断删除。对于除了队尾的元素之外，每个队列中的元素供当前要计算的状态的“值”是 $f(q[x].position) + a[q[x+1].position]$，其中,$q[x]$ 代表第 x 个队列元素，$position$ 这代表他在原来数组中的位置，我们不妨把这个值记为 t。那每一次在队首、队尾的删除就相当于删除 t，每一次删除完毕之后又要插入一个新的 t，然后需要求出队列中的 t 的**最小值**。

我们发现，完成上述一系列工作的最佳选择就是**平衡树**，这样每个元素都插入、删除、查找各一遍，复杂度为 $O(\log n)$,最后的时间复杂度是 $O(n \log n)$ 。

有一个细节：$b[x]$ 这一个单独的决策点是不能够被省掉的（仍然留给读者思考），而上述队列的方法有可能将其删除，所以要通过特判来完成。

•小结

我们依然通过发掘单调性完成了这一道题目。和以前不同的是，这次单调

队列中存的值不同，但依然可以通过平衡树或者其余数据结构完成。单调队列是一个很灵活的东西，要合理使用。

下面一个章节笔者将引入决策单调性和四边形不等式等内容，浅析单调性在动态规划优化中的另外一个形式。

【一些复杂的例子】

【例三】玩具装箱 Toy[3]

•问题描述

有 n 个玩具，要将它们分为若干组进行打包，每个玩具有一个长度 $len[x]$。每一组必须是连续的一组玩具。如果将第 x 到第 y 个玩具打包到一组，那么它们的长度 $l = y - x + \sum_{i=x}^{y} len[i]$，将这组玩具打包所需的代价等于 $(l - L)^2$。问将所有玩具打包的最小代价是多少。注意到每组玩具个数并没有限制。$n<=5*10^4$。

•朴素的解法

仍然可以很轻易的写出一个动态规划的方法：

用$f(x)$代表将前 x 个玩具打包所需要的最小代价。

$f(x)=\mathop{Min}\limits_{i=0}^{x-1}(f[i]+w[i,x])$，其中 $w[i,x]$ 代表将 $i+1-x$ 的玩具打包所需的费用。

时间复杂度 $O(n^2)$。

•正确的解法

我们如果将每个状态的决策 $k[x]$ 打印出来列成一张表，会发现：对于 $\forall i<j,k[i]<=k[j]$。这就说明，决策是单调的。

我们不妨暂且不管这是为什么，我们想通过这一单调性来在对数级或者线性时间内解决本题。

由于决策是单调的，我们可以通过二分查找决策的**转折点**来在 $O(n\log n)$的时间内解决本题：

3 选自 HNOI2008 Toy

一开始动态规划的边界是$f[0]=0$。那么，一开始所有状态的最优决策都是 0（因为还没有其他状态被计算）。然后，从状态 1 开始逐步往后扫描，扫描到当前状态 i，要**保证**$f[i]$已经决策完毕，可以计算出来。

现在，由于$f[i]$已经被算了出来，我们尝试寻找哪些后面的状态的决策**有可能**是 i。注意到决策是单调的，就是说如果一个状态$f[x]$，他在 1 到 i 这么多决策中的最优决策是 i，那么对于 $\forall y,y>x$，$f[y]$在 **1 到 i 之间**的最优决策肯定是 i。

这样，我们可以通过二分查找，确定决策 i 在整个区间中的决策转折点，设转折点为 x，那么将$[x,n]$这个区间，全部**刷**上决策 i，如果直接 $O(n)$刷色，肯定是不行的，可以通过线段树来完成。这样的时间复杂度是 $O(n\log^2 n)$的，因为二

分查找转折点的时候还要在线段树中用 $\log n$ 的时间计算当前的状态值。

其实最简单、常数小的算法是用一个栈来实现。

•决策为什么是单调的？

不少读者肯定知道在这样一种状态量为 $O(n)$，决策量为 $O(n)$，直接求解复杂度为 $O(n^2)$的动态规划问题中，如果状态 x 转移到状态 y 的代价为 $w[x,y]$，只要满足 $w[x,y]+w[x+1,y+1]\le w[x+1,y]+w[x,y+1]$，那么这个动态规划的问题的决策就是单调的。这就是四边形不等式的原理。读者可以通过上述式子证明这个题的决策为什么单调。有关四边形不等式的证明，由于篇幅和笔者能力有限，所以请读者阅读相关书籍。

【例四】仓库建设 Storage[4]

•问题描述

有 n 个货物销售点，在一座山上依次向下排。第一个销售点最高，然后山脚下的是第 n 个销售点。马上就要下大雨了，要设置一些储存点来储存货物，并且为了节省时间、力气，规定只能够**向山下运**，一单位的货物运送一个单位的距离要耗费 1 元钱。给出每个销售点距第一个销售点的距离 $x[i]$，以及每个销售点储存的货物量 $p[i]$和在这个销售点设置储存点的费用 $c[i]$，请给出一个合理的方案，使得运输费用最小。$N<=10^6$。

•解法分析

这道题目，也是非常容易的写出动态规划的转移方程：设 $f[i]$表示**在**第 i 个点建设一个仓库，前 i 个货物点运送的最小总费用。

$$f(i)=\mathop{Min}_{j=0}^{i-1}(f[j]+w[j,i])+c[i]$$

$$w[j,i]=p[j+1]*(x[i]-x[j+1])+p[j+2]*(x[i]-x[j+2])+\cdots\cdots p[i]*(x[i]-x[i])$$

最后一个其实是多余的，但为了工整以及接下来的解题的方便就保留了下来。

通过打表或者观察、计算，可以发现，决策是单调的。可惜我们高兴不起来因为 n 最多有可能到一百万，玩具装箱的算法很明显不行。

但我们发现代价的那个式子十分有规律，我们将其变形：

$$w[j,i]=p[j+1]*(x[i]-x[j+1])+p[j+2]*(x[i]-x[j+2])+\cdots\cdots p[i]*(x[i]-x[i])$$

$$=x[i]*(p[j+1]+......+p[i])-(p[j+1]*x[j+1]+......p[i]*x[i])$$

我们不妨设 $sump[x]=\sum_{i=1}^{x}p[i]$，$sum[x]=\sum_{i=1}^{x}p[i]*x[i]$,那么

---

4 选自 ZJtsc2007 Storage

$$w[j,i]=x[i]*(sump[i]-sump[j])-(sum[i]-sum[j])$$

$$=-x[i]*sump[j]+sum[j]+(x[i]*sump[i]-sum[i])$$

现在的式子比较简洁，我们将 Dp 的式子合起来：

$f(i)=\mathop{Min}_{j=0}^{i-1}(f[j]+sum[j]-x[i]*sump[j])+x[i]*sump[i]-sum[i]+c[i]$，后面的东西是根据 i 确定的常量，对决策没有影响，可以暂时忽略。

观察括号里的式子：如果设 $a[i]$=-$x[i]$，$x(j)$=$sump[j]$，$y(j)$=$f[j]$+$sum[j]$，则

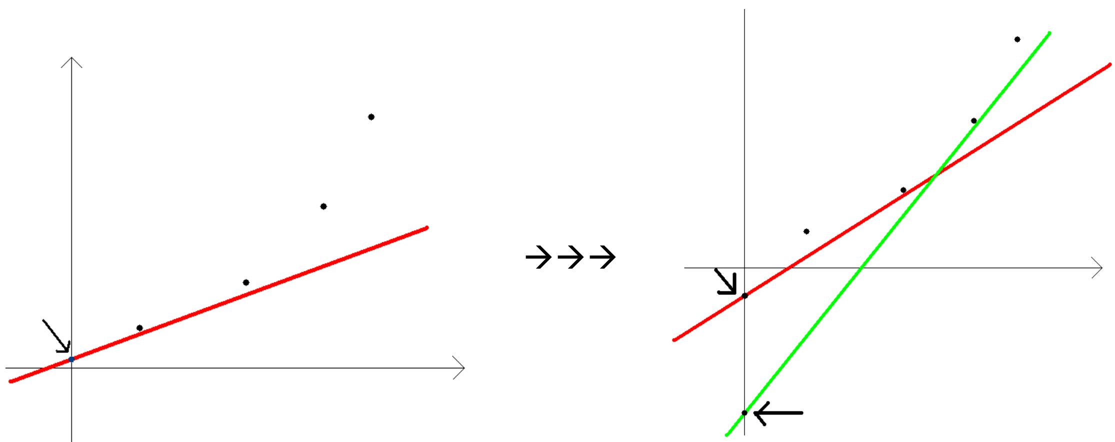
$f(i)=\mathop{Min}_{j=0}^{i-1}(a[i]*x(j)+y(j))$。可以发现，$x(j)$，$y(j)$都是根据 j 确定的唯一常

量。现在的任务就是给定 a，求一组 x,y，使得目标函数最小。

设 $G=f(i)$，那么 $G=ax+y$，移项得：$y=-ax+G$

这就相当于：有一条斜率已经确定的直线，有一堆点，选出一个点使得 G 最小。很明显：这个点在点集的凸包上（相当于直线从负无穷往上平移，第一个碰到的点，这必然是凸包上的点）。

可以发现，根据 a 的定义，直线的斜率是单调递增的，并且点的**横坐标**也是单调递增的！读者可以用手比划一下，我们必须维护一个**下凸**的凸包（因为要求最小值），那随着 i 的增加，状态 i 的决策点一定是**单调向右**移动的，这也正好证明了决策的单调性！（图中箭头指的点就是 $f(i)$）



这样一来，具体的算法步骤就水道渠成了：

1）维护一个栈，代表这凸包里的点。这个栈不仅有栈顶指针，还有一个**决策指针**，代表当前的最优决策应该从哪里开始"搜索"。根据决策单调的性质：如果一个决策没有成为当前状态的最优决策，那么就永远也不可能成为后面状态的最优决策，所以决策指针**单调右移**。

2）我们来看对于当前要计算的状态怎样查找最优决策。我们设决策指针为 t，寻找最优决策的伪代码如下：（设 $v[t]=f[t]+w[t,i]$）

While v[t]>=v[t+1] Do t→ t+1，这段代码充分应用了**决策单调**的性质。

3）当前状态计算完毕，看怎么插入当前状态所对应的二元组。当前状态的二元组必然是凸包上的点，我们用 Graham 的维护方式，不断弹栈（注意维护下凸型 Graham 的方向），最后再插入。注意如果决策指针也被弹栈了，那么决策指针应当指向当前状态（决策单调！）。

•时间复杂度分析

主过程里有一重循环，复杂度为 $O(n)$。重要的是 while 那段。前面说过，决策指针单调右移，所以 while 总共加起来也是 $O(n)$，又由于每个点最多出栈一次、进栈一次，所以最后的时间复杂度是 $O(n)$，我们很顺利的解决了这道题。

【利用凸线的单调性优化 Dp】

其实，例四是利用一次函数、坐标的单调性来进行算法的优化，已经利用了凸包（其实例三也可以）的优美性质在线性时间解决了问题。 但是，例四中通过代数恒等式变形所得到的线性规划式满足：随着计算状态的逐步推进，直线的斜率单调变化、同时 x 或者 y 也单调变化。如果这两者其中一个不满足条件那该怎么办呢？

【例五】货币兑换 Cash[5]

•问题描述

小 Y 要在 n 天里，进行一次股票的交易，股票有 A 类券和 B 类券两种。告诉你每天 A 股票的单价 Ai，B 股票的单价 Bi，以及买入股票的时候得到的 A 股票和 B 股票的比率 Ratei。刚开始他手里有 S 元钱，他每天可以有三种操作：①买入操作，即将 S 元钱**全部**用来买股票，将得到一些 A 股票和 B 股票，其中 A 股票数量/B 股票数量=Rate。②卖出操作，即将手中的股票**全部**卖出，得到当天应得的钱数。③什么都不干。 求：这 n 天下来最多能赚多少钱？$n<=10^5$

•解法分析

这道题目，其实朴素的解法也十分简单。

我们用 $f(i)$ 代表第 i 天最后最多能得到多少钱，并且第 i 天要做出卖出操作。当状态 $f(i)$ 被计算完之后，我们用 $x(i)$，$y(i)$ 代表将 **$f(i)$ 这么多钱**卖出能得到的 A 券和 B 券的数量，这个可以通过解方程得出。那么很容易得到状态转移方程：

$f(i)=\mathop{Max}\limits_{j=0}^{i-1}\{a[i]*x[j]+b[i]*y[j]\}$，最后的答案 $=Max(f[i],0<=i<=n)$

5 选自 NOI2007

这个方程直接求解时间复杂度依然是 $O(n^2)$ 的。

但这个方程是一个求最值的问题：G=$ax+by$ → $y=-\frac{a}{b}+\frac{G}{b}$。

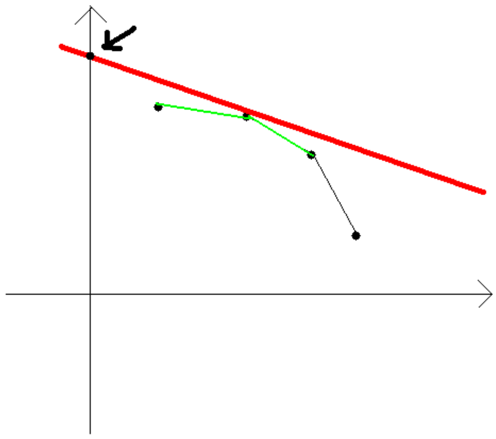
这和例四的方程十分相似，可是 $-\frac{a}{b}$ 单调么？$x[i]$ 随着 i 单调么？答案都是

否定的，我们要另寻他法。

但是有一点是要肯定的：最优决策点依然在凸包上。但是查找最优决策点、插入毫无规律。

我们先来看怎样查找最优决策点：

首先，要维护一个上凸的凸包（因为要求最大值），我们观察下面一幅图（加粗的直线代表当前状态对应的直线，其斜率 $k=-\frac{a}{b}$）：



我们看到，如果一个点成为了最优决策点的话，那么设它左边凸包上的点和它的线段的斜率为 $k1$，右边的点和它连的线段的斜率为 $k2$，那么必然有 $k2<=k<=k1$（我们可以设最左边的点的 $k1=+\infty$，最右边的点的 $k2$ 设为 $-\infty$）。根据这个单调性，查找最优决策就可以用二分查找来在 $O(\log n)$的时间内完成。

我们再来看怎样插入一个点。因为要维护凸包，所以先二分查找到在数组中应该插入的位置，然后对两边进行 Graham 式凸包维护。这时候问题来了：怎样删除结点？Move？对，用 Move 是最折中的方法，对于本题来说，有人试过因为数据的原因，Move 可以搞定。但如果要让时间复杂度严格的为 $O(n\log n)$，得需要平衡树。

下面是这道题的算法步骤：

1）建立一颗以横坐标为关键字的平衡树（本文以 **Splay** 为例）。

2）二分查找最优决策点，计算状态值。

3）插入节点：将该节点的横坐标 a 插入 Splay。接下来，要对左边和右边进行凸线的维护。我们以左边为例（右边类似）：首先再二分查找出离当前点最近的，并且满足凸包性质（因为是上凸形，所以斜率单调减）的点 b。然后根据 graham 的特点：被删掉的点一定是连续的一段，我们可以将点 b 伸展到根节点，a 伸展到跟的右子树，那么 a 的右子树肯定是要被删的点，直接将其删除。但要注意的一点是：a 不一定是凸包上的点，所以向左、向右弄完之后还要检查 a 是否符合凸包要求，不行的话依然要删掉。

这样，这道**难题**就被我们在对数级的时间内完成了。

【全文总结】

本文通过 5 个经典的例题，阐述了单调性在动态规划试题中的应用。5 个例题各具特色，各有自己的代表性，难度可以说由浅入深。希望读者能够认真阅读体会其中的奥妙。在许多问题里，单调性是打开胜利之门的钥匙，是指引胜利彼岸的灯塔，是飞上成功蓝天的翅膀，是浇灌成功之花的甘露。

【参考文献】

1）《1D1D 动态规划优化初步》 作者：南京师范大学附属中学 汪一宁

2）《2007 中国信息学奥林匹克年鉴》 主编：中国计算机学会

3）《算法艺术与信息学竞赛》主编：刘汝佳&黄亮

4）《凸完全单调性的一个加强与应用》中国国家集训队论文 2007 作者：杨哲

5）《浅谈基于分层思想的网络流算法》中国国家集训队论文 2007 作者：王欣上

【感谢】

感谢南师附中的**汪一宁**同学一直以来对我的帮助

感谢浙江省绍兴市第一中学的**董华星**同学一直以来对我的帮助

感谢 **Velocious Informatics Judge Online System** 为我提供例题 1

感谢 **Peking University Onlinejudge** 为我提供例题 2

感谢浙江省镇海中学的**谢天**同学为我提供 **HNOI2008** 试题

感谢**董华星**同学提供例题 4

忠心感谢！