斜率优化动态规划转移研究心得

罗一粟

目录

[1]https://loj.ac/p/10188

[2]https://loj.ac/p/10190

[3]https://codeforces.com/contest/631/problem/E

[4]https://loj.ac/p/2353

1 一些经典模型

1.1 例题引入：HNOI2008 玩具装箱 [5]

题目描述

有 n 个玩具，第 i 个玩具价值 a_i，我们需要把这些玩具分成若干段。对于一段玩具 $[l, r]$，它的花费为 $(r - l - L + \sum_{i=l}^{r} a_i)^2$，其中 L 是一个给定常数。求分段的最小花费。

数据范围：$1 \le n \le 5 \times 10^4, 1 \le L, a_i \le 10^7$。

分析

这道题目显然可以用动态规划求解，我们设 f_i 为前 i 个玩具，分若干段的最小代价。有状态转移方程：

$$f_i = \min_{j<i}\{f_j + (sum_i - sum_j + i - j - 1 - L)^2\}$$

其中 $sum_i = \sum_{j=1}^{i} a_i$。

我们把方程简化一下：设 $s_i = sum_i + i, L' = L + 1$，那么状态转移方程可以化为：

$$f_i = \min_{j<i}\{f_j + (s_i - s_j - L')^2\}$$

如果我们直接枚举决策点 j 进行转移，时间复杂度是 $O(n^2)$ 的，无法通过本题 $n \le 5 \times 10^4$ 的数据范围，这就引出了我们下面要讲的斜率优化 DP 转移。

我们需要尝试优化转移的复杂度，即快速找到最优决策点，为此我们对决策进行分析。

设当前需要求解 f_i，对于任意两个决策 j, k，决策 j 比 k 更优当且仅当：

$$\begin{aligned}
f_j + (s_i - s_j - L')^2 &< f_k + (s_i - s_k - L')^2 \\
f_j + (s_i - L')^2 - 2s_j(s_i - L') + s_j^2 &< f_k + (s_i - L')^2 - 2s_k(s_i - L') + s_k^2 \\
(f_j + s_j^2) - (f_k + s_k^2) &< 2s_j(s_i - L') - 2s_k(s_i - L') \\
\frac{(f_j + s_j^2) - (f_k + s_k^2)}{2s_j - 2s_k} &< s_i - L'
\end{aligned}$$

这样，我们把至于 i 有关的项和只与 j 有关的项分离开了。设 $Y(i) = f_j + s_j^2, X(i) = 2s_j, K(i) = s_i - L'$，上式可化为：

$$\frac{Y(j) - Y(k)}{X(j) - X(k)} < K(i)$$

这个式子和斜率的形式非常相似，我们可以把每个决策看成二维平面上的一个点 $P_j = (X(j), Y(j))$，那么有：

定理 1 决策 j 比 k 优当且仅当 P_jP_k 的斜率小于 $K(i)$。

[5] https://loj.ac/p/10188

下面我们来证明一个关键结论：

定理 2 对于所有可能成为最优决策的点，按照横坐标从小到大排序后，对于任意相邻的三个决策点 P_a, P_b, P_c，有 $slope(P_a, P_b) < slope(P_b, P_c)$。（$slope(P_a, P_b)$ 表示过 P_a 和 P_b 的直线的斜率）

下面会给出一个代数上的严谨证明和一个较直观的几何解释。

证明 设一个任意子问题为 i, $k_0 = s_i - L'$, 有三个决策点 $P(j_1), P(j_2), P(j_3)$, $k_1 = slope(P(j_1), P(j_2)), k_2 = slope(P(j_2), P(j_3))$。

对于 $k_1 = k_2$，结论显然成立，下面证明 $k_1 \neq k_2$ 的情况。

用反证法，假设 $k_2 < k_1$ 且 k_2 比 k_1 更优，分三种情况：

- $k_0 < k_2 < k_1$，由定理 1 可知，j_1 最优，与 j_2 最优矛盾；
- $k_2 \leq k_0 < k_1$，由定理 1 可知，j_1, j_3 均比 j_2 优，与 j_2 最优矛盾；
- $k_2 < k_1 \leq k_0$，由定理 1 可知，j_3 最优，与 j_2 最优矛盾；

因此假设不成立，原命题成立。

直观解释

对于方程 $f_i = f_j + (s_i - s_j - L')^2$，我们将其变形：

$$f_j + s_j^2 = 2(s_i - L') \times 2s_j + f_i - (s_i - L')^2$$

同样设 $Y(i) = f_j + s_j^2, X(i) = 2s_j, K(i) = s_i - L', B(i) = f_i - (s_i - L')^2$，同样把决策看成二维平面上的点。我们把这个式子化成了 $y = kx + b$ 的形式，要最小化 f_i 就是要最小化截距，这就相当于拿一条斜率已知的直线从下往上扫，扫到的第一个点就是最优决策（如下图）。可以想象只有下凸壳上的点可能被扫到。

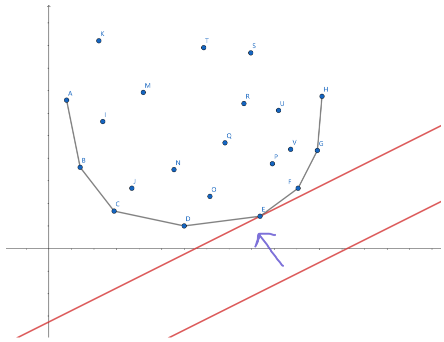

图 1: 直线截下凸壳

综上，我们只需要维护一个斜率单调递增的下凸壳（即凸包的下半部分），决策点一定在这个下凸壳上。

接下来考虑如何得到当前子问题的答案和如何动态维护下凸壳。

根据上述结论，对于当前子问题 i，其最佳决策点 j 满足 $slope(P_{j-1}, P_j) \le K(i) \le slope(P_j, P_{j+1})$。

在这道题中，决策点的横坐标是单调递增的，我们可以使用一个双端队列来维护下凸壳。当要求出当前子问题的答案时，我们不断弹出队头直到找到满足上面性质的最优决策点，因为本题中 $K(i)$ 单调递增，如果一个点不是当前子问题的最优决策，那它也不会成为以后的最优决策，所以这个做法是正确的。

当新加入一个决策点前，我们不断弹出队尾，直到加入该点后仍然能满足斜率的单调递增性质为止，然后加入这个决策点。

因为每个点最多会被加进和弹出队列一次，所以该算法时间复杂度是 $O(n)$ 的。

至此，我们在 $O(n)$ 的时空复杂度内解决了该问题。

小结

在本题中，我们通过将状态转移方程化为斜率（直线）的形式，通过斜率的单调性优化了复杂度。

一般地，斜率优化可以用来优化一类状态转移方程，这类方程的性质一般为：存在只与 i 有关、只与 j 有关、与 i,j 同时有关的项。我们一般把只与 j 有关的项看作 y，把只与 i 有关的项看作 b，同时有关的项中与 i 有关的看作 k，与 j 有关的看做 x，利用决策点间斜率单调的性质来优化转移复杂度。

1.2 小变形：APIO2010 特别行动队 [6]

题目描述

有一个 n 个步兵组成的军队，每个步兵有一个战斗力 a_i，需要将其分为若干连续段组成小队，把战斗力之和为 x 的士兵分成一个小队得到的修正战斗力为 $Ax^2 + Bx + C$。你的任务是：求出对于所有编队方案中，最大的各小队修正战斗力之和。

数据范围：$1 \le n \le 10^6, -5 \le A \le -1, |B| \le 10^7, |C| \le 10^7, 1 \le a_i \le 100$。

分析

设 $s_i = \sum_{j=1}^{i} a_j$，f_i 为将前 i 个步兵分组后得到的最大修正战斗力之和，有：

$$f_i = \max_{j<i}\{f_j + A \times (s_i - s_j)^2 + B \times (s_i - s_j) + C\}$$

[6] https://loj.ac/p/10190

这里我们用上面讲的比较直观的方法进行分析，对于这个方程，我们按照提到的方法进行移项。

$$f_i = f_j + As_i^2 - 2As_is_j + As_j^2 + Bs_i - Bs_j + C$$
$$f_j + As_j^2 - Bs_j = 2As_is_j + f_i - Bs_i - C$$

我们设 $y = f_j + As_j^2 - Bs_j, x = s_j, k = 2As_i, b = f_i - Bs_i - C$，套用上个问题的分析方法，我们要最大化截距，那么需要维护的是一个上凸壳。

在本题中，斜率单调递减，横坐标单调递增，所以我们同样用双端队列维护这个上凸壳。

2 与其他算法结合

当方程中的 x_i, k_i 不满足单调性时，我们往往需要其他方法维护凸壳和求解答案。

2.1 与二分结合：Codeforces 631E Product Sum [7]

题目描述

给出一个长度为 n 的序列 $\{a_i\}$，定义其权值为 $\sum_i = 1^n i \times a_i$，你可以将序列中的一个数移动一次，求移动后数列的最大权值。

数据范围：$2 \le n \le 2 \times 10^5, |a_i| \le 10^6$。

分析

显然我们只需要最大化移动一个数后权值增加量。设 f_i 为移动第 i 个数后能得到的最大权值增加量，$s_i = \sum_{j=1}^{i} a_j$，有：

$$f_i = \max_{j=0}^{n}\{(j-i) \times a_i + s_i - s_j\}$$

这很显然符合我们上述斜率优化的形式，对方程变形：

$$s_j = a_i \times j + (s_i - i \times a_i - f_i)$$

设 $y = s_j, x = j, k = a_i, b = s_i - i \times a_i - f_i$，我们需要最小化截距，那么维护下凸壳。

在这道题目中，每个 i 都可以从所有 j 转移得来，而这道题 x 单调递增，我们可以用一个单调栈求出下凸壳，接下来问题转化为如何在一个下凸壳上找到答案。

显然，这道题中用来切下凸壳的直线的斜率没有单调性，不能用以前弹出单调队列队首的方法找到最优决策。因为下凸壳的斜率单调递增，我们可以在下凸壳上二分查找得到第一个斜率比当前 k_i 大的直线，该直线上两个点中横坐标最小的点即为我们要找的最优决策点。

这样，我们得到了一个时间复杂度 $O(n \log n)$，空间复杂度 $O(n)$ 的算法。

[7] https://codeforces.com/contest/631/problem/E

2.2 使用 CDQ 分治优化：NOI2007 货币兑换 [8]

题目描述

有 A,B 两种金券，它们每天的价值都不同，在第 i 天时分别为 a_i, b_i，你可以用手中的金券换取相应价值的钱，或用钱兑换相同价值的金券，但兑换到的两种金券的数量之比是一个定值，这个定值在第 i 天为 r_i。

现给出 n 天中两种金券的价值，R，以及你初始时拥有的资金 S，求 n 天后你最多能有多少钱。

注：一定存在一种最优方案，满足：每次兑换金券花光所有的钱，每次换钱时花掉所有的金券。

数据范围：$0 < A_K \leq 10$，$0 < B_K \leq 10$，$0 < R_K \leq 100$，$ans \leq 10^9$。

分析

注意到我们每次兑换金卷都要花光所有的钱，那么我们设 f_i 为第 i 天最多拥有的钱数，p_i 为第 i 天把钱花光能兑换的 A 金卷数量，q_i 为第 i 天把钱花光能兑换的 B 金卷数量。

可以通过解二元一次方程组得到 $p_i = \dfrac{f_i R_i}{a_i r_i + b_i}, q_i = \dfrac{f_i}{a_i r_i + b_i}$，具体过程这里不赘述。

若第 i 天不买入金卷，有：$f_i = f_{i-1}$，若第 i 天买入金券，我们枚举上一次买入金券的时间，可以得到：$f_i = \max\limits_{j<i}\{a_i p_j + b_i q_j\}$。

对不买入金卷的情况，我们简单判断一下就好，接下来对买入金券的方程变形，可以得到：

$$p_j = -\frac{b_i}{a_i} q_j + \frac{f_i}{a_i}$$

同样的套路，设 $y = p_j, k = -\dfrac{b_i}{a_i}, x = q_j, b = \dfrac{f_i}{a_i}$，我们要最大化截距，那么就需要维护一个上凸壳。但这次我们发现，k 和 x 都没有单调性。

一种动态维护上凸壳的方法是使用平衡树：插入时不断找到该点前驱和后继，判断是否满足上凸壳的性质，不满足就从上凸壳中删去；寻找答案时在这个上凸壳上二分找到答案，该方法虽易于想到，但实现较为复杂，算法的时间常数也较大。下面介绍一种使用 CDQ 分治求解该问题的方法。

CDQ 分治的核心思想是：假设我们要处理 $[l, r]$ 区间的问题，将所有问题分为两部分 $[l, mid]$ 和 $[mid + 1, r]$，首先递归 CDQ(l, mid) 得出左半区间的解，然后通过所有左半区间的解更新所有右半区间的解，最后递归 CDQ$(mid + 1, r)$ 得到右半区间的解，从而得到整个区间的解。很显然，该算法可以保证子每个问题都得到最优解。

这个算法有一个好处：在得到左半区间的解后，我们不需要考虑左半区间内部，只需要考虑其对右半区间的影响；在计算右半区间解的时候，我们也不需要考虑右半区间内部互相的影响，只需要考虑来自左半区间的影响。因此，我们可以在得到左半区间的解后，将其按照 x 从小到大排序，强制 x 单调递增，这样可以使用一个单调栈来维护上凸壳；接着，我们对右半区间按照 k 由小到大进行排序，强制让 k 单调递增，这样就可以用不断弹出队尾的方法得到右半区间的解。都更新完后，我们按照天数排序还原回原顺序，递归计算右半区间，即可得到整个问题的解。

[8] https://loj.ac/p/2353

使用该方法，如果使用归并排序的思想在递归时进行排序，时间复杂度是 $O(n\log n)$，如果每次都对整个区间进行快速排序，时间复杂度是 $O(n\log^2 n)$，但都足以在较短的编码量内通过本题。

3 总结

在一维的动态规划问题中，如果状态转移方程可以化为 $y=kx+b$ 的形式，其中 x,y 与枚举的 j 有关，k,b 与当前计算的 i 有关，我们可以使用斜率优化进行优化转移。

在这类问题中，状态转移方程取决于题目的数学模型，而具体怎么实现斜率优化则一般与题目中 k,x 的单调性有密不可分的联系。解决此类问题时，不可生搬硬套模板，而是要具体问题具体分析：使用单调队列时，想想从队尾插入还是从队首插入，凸壳的斜率是单调递增还是单调递减，找答案时从队首找还是从队尾找，又或者需要二分找？使用 CDQ 分治时想想怎么让没有单调性的点变得有单调性？这些都是需要根据题目灵活变动的，而数形结合的思路有时也会有所帮助。

斜率优化是一类灵活多变的题目，也经常与别的知识点相结合。当你拨开题目的层层面纱，推出核心的状态转移方程，发现是个熟悉的斜率优化式子，那就离通过此题不远了。

4 几道习题

- SDOI2016 征途 [9]
- NOI2019 回家路线 [10]
- CTSC2016 时空旅行 [11]（线段树分治 + 斜率优化）

参考文献

[1] OI Wiki 斜率优化

[9] https://loj.ac/p/2035
[10] https://loj.ac/p/3156
[11] https://loj.ac/p/2987