动态规划——进阶转移技巧

4182_543_731

2024/07

Table of Contents

序列 DP 模型

这两个章节的内容主要针对如下形式的序列最优化 DP 问题：

$$dp_i = \min_{j<i} dp_j + f_{j,i}$$

$$dp_i = \min_{j<i} g_j + f_{j,i}$$

这里一般认为 f 可以快速求出，而我们通常希望做到比暴力的 $O(n^2)$ 更好。

实际问题中注意 $\min, \max$ 的区别

斜率优化问题

最经典的斜率优化问题可以表示为如下形式：给两列递增的 $\{x_i\},\{y_i\}$，随后转移代价 $f_{i,j}$ 是一个 $-x_iy_j$ 的形式（加上一些只与 i 有关的项、和只与 j 有关的项）。

$$dp_i = \min_{j<i} dp_j + a_j + b_i - x_iy_j$$

$$dp_i = b_i + \min_{j<i}(dp_j + a_j) - x_iy_j$$

经典问题

给定正整数序列 v，将其划分为若干段，每一段代价为段内和的平方加上 k。

记 dp_i 表示划分前 i 个位置的最小代价，su_i 表示序列前缀和，则 $f_{j,i} = k + (su_i - su_j)^2 = k + su_i^2 + su_j^2 - 2su_isu_j$,

$$dp_i = k + su_i^2 + \min_j(dp_j + su_j^2) - 2su_isu_j$$

斜率优化例子

[APIO2010] 特别行动队 / [NOI2019] 回家路线 / ...

$$f_{j,i} = a(su_i - su_j)^2 + b(su_i - su_j) + c$$

显然。

[APIO2014] 序列分割

给一个长度为 n 的正整数序列。进行 $k-1$ 次操作，每次可以选择一个序列分成两半，得分为两半分别总和的乘积。求最大总得分。

斜率优化例子

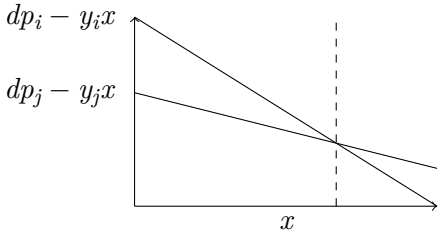
[SDOI2012] 任务安排

n 个任务排成一列，每个任务有工作量 t_i，权重 w_i。将任务划分为若干区间然后顺序完成，一个区间的任务将用 $C + \sum t_i$ 的时间一起完成。每一个任务的代价为权重乘以其完成时间(当前区间及之前区间的用时总和)，求最小总代价。

斜率优化

$$dp_i = \min_{j<i} dp_j - y_j x_i$$

考虑如何快速解决这一问题。每个 j 对之后 i 的转移可以看成一条斜率为 $-y_j$ 的直线，带入 x_i 为横坐标即可得到转移的值。考虑两条直线 j, i 之间的比较：



那么存在分界点 $c = \frac{dp_i - dp_j}{y_i - y_j}$，使得 $x \le c$ 时 y 小的直线更优，否则 y 大的更优。因为 y 递增，随着 x 增大后面的直线会逐渐比前面的更优。

斜率优化

存在分界点 $c = \frac{dp_i - dp_j}{y_i - y_j}$，使得 $x \le c$ 时 y 小的直线更优，否则 y 大的更优。

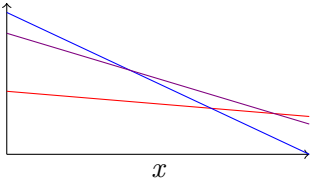
如果所有分界点自动递增，那情况非常直接：随着 x 增大，最优直线从第一条开始经过所有分界点遍历每一条直线。此时询问只需要用分界点找到对应的段。

斜率优化

存在分界点 $c = \frac{dp_i - dp_j}{y_i - y_j}$，使得 $x \le c$ 时 y 小的直线更优，否则 y 大的更优。

如果所有分界点自动递增，那情况非常直接：随着 x 增大，最优直线从第一条开始经过所有分界点遍历每一条直线。此时询问只需要用分界点找到对应的段。

但如果分界点不递增，怎么办？可以发现，此时中间的直线一定没有用：



那么删掉中间直线继续做，直到分界点递增即可。

斜率优化

$$dp_i = \min_{j<i} dp_j - y_j x_i$$

回到原问题，我们依次做 DP 并加入 (y_j, dp_j) 这样的直线。因为 y 递增，我们只会在最后加入直线。

然后需要不断处理上一页提到的情况。因为插入在末尾，这里只需要不断考虑新插入的直线与原先最后两条直线，然后删掉最后一条继续，实现为类似栈的形式。

如果 x 也递增，那查找可以更简单：已经不如之后直线的可以直接扔掉，保持第一条线段是当前最优。这样可以线性。

可能存在的代码演示.jpg

斜率优化扩展 (1)

上面我们要求了 x, y 递增，但对于一些更极端的情况：

- 如果 x 不递增，那可以改成分界点上二分。
- 如果 y 不递增（例：[NOI2007] 货币兑换），那需要插入到中间然后两个方向处理，这会让实现较为复杂。可以用平衡树维护这个过程，可以类似 cdq 分治变为单调的情况，也可以李超树直接解决问题
- 如果 x, y **不降**，那差不多可以用之前的做法，但注意两条直线斜率**相同**的情况！

斜率优化扩展 (2)

回想之前的推导，直线有什么用？它只用来满足了如下性质：

- 对于任意两个不同起点的转移函数 f_i, f_j，存在分界点 $c_{i,j}$ 使得分界点前一个更优，分界点后另一个更优。(且我们能快速算分界点)
- 可以将所有函数排成一列，使得比较两个函数时，一定是前面的函数在 x 小的时候更优。

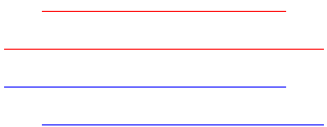
只要函数满足这个性质，那它也不一定需要是直线！

这个性质通常被称为完全单调性。

Table of Contents

四边形不等式

称二维序列 f 满足四边形不等式，当且仅当 $\forall a \le b, c \le d, f_{a,d} + f_{b,c} \ge f_{a,c} + f_{b,d}$。即下图中蓝边和更小。



简单的例子：区间平方和（$(r-l)^2$ 或者 $-l*r$），区间 k 次方和（$k \ge 1$，根据函数凸性），区间逆序对：

Lemma

f 满足四边形不等式当且仅当 $\forall a < b, f_{a,b+1} + f_{a+1,b} \ge f_{a,b} + f_{a+1,b+1}$。

简要证明：首先归纳得到 $f_{a,b+k} + f_{a+1,b} \ge f_{a,b} + f_{a+1,b+k}$，然后再归纳 $a/a+1$ 这一侧即可。

四边形不等式与决策单调性

$$dp_i = \min_{j<i} g_j + f_{j,i}$$

$$\forall a \le b, c \le d, f_{a,d} + f_{b,c} \ge f_{a,c} + f_{b,d}$$

只要 f 满足四边形不等式，我们就有如下决策单调性：

定理

位置 i 处的（最小）最优转移点 r_i 随 i 增加单调不降。

$$\begin{aligned} r_a &\text{ ——————— } dp_{r_a} + f_{r_a,a} \\ r_b &\text{ ——————— } dp_{r_b} + f_{r_b,b} \\ r_b &\text{ ——————— } dp_{r_b} + f_{r_b,a} \\ r_a &\text{ ——————— } dp_{r_a} + f_{r_a,b} \end{aligned}$$

通过决策单调性，我们容易解决这些 DP 问题。

决策单调性——分治做法

决策单调性

位置 i 处的（最小）最优转移点 r_i 随 i 增加单调不降。

考虑选一个点 mid 求出其最优转移点 r_{mid}。那么前面位置的最优转移点必然在 r_{mid} 或之前，后面位置的转移点必然在 r_{mid} 或之后，这就减少了需要考虑的位置数量。

具体来说，考虑 $\text{solve}(l, r, x, y)$ 表示处理 $[l, r]$ 区间的转移，只考虑 $[x, y]$ 间的转移点。取 $[l, r]$ 的中点 mid 求出其任一最优转移点 t_m，然后分治 $\text{solve}(l, mid-1, x, t_m), \text{solve}(mid+1, r, t_m, y)$。

分治深度 $\log n$，每一节点分出的两个区间长度和只增加 1，从而复杂度 $O(n \log n)$。

四边形不等式与斜率优化

$$\forall a \le b, c \le d, f_{a,d} + f_{b,c} \ge f_{a,c} + f_{b,d}$$

事实上，满足四边形不等式的 DP 必定满足斜率优化的性质：取前后两个转移点，一定存在分界点使得分界点前取前面的转移点，分解点后取后面的。

证明.

两个转移位置的差为 $f_{a,x} - f_{b,x}$，由四边形不等式可知这个值随着 x 增加单调不降，那么一定是开始时 $a(a < b)$ 更优，之后反过来。 □

绝大多数的斜率优化问题也满足四边形不等式：经典形式（$-xy$）显然满足，一般情况下只要代价比较连续，使得分界点连续变化就也能满足条件。

但这里与斜率优化的一个区别在于分界点不一定能快速求出。不过根据单调性，我们显然可以二分分界点，这就有了下一个算法。

决策单调性——单调栈做法

考虑依次求出 dp_i，并维护每一段最优决策点。在右侧加入新的 dp_i 的转移时，由决策单调性它会覆盖当前一个后缀的最后转移点。那么可以二分这个位置，将右侧的段替换掉。

一种经典实现方式为，先和当前最后一段的左端点比较，如果能完全代替就删掉整段，否则在当前段内找分界点。复杂度 $O(n \log n)$

可以发现这个算法和斜率优化的过程几乎完全一致，唯一区别在于二分找分界点。

决策单调性——总结

两个算法复杂度都是 $O(n\log n)$，但它们有着不同的性质。

首先，分治法只能直接解决 $dp_i=\min_{j<i} g_j+f_{j,i}$，但如果是 $dp_i=\min_{j<i} dp_j+f_{j,i}$，分治就不能直接做了。此时单调栈法可以直接解决问题。

分治法也可以做这种问题：先 cdq 分治一次，然后每一步变成第一类问题。但这样复杂度为 $O(n\log^2 n)$。

[NOI2009] 诗人小 G

仍然是正整数序列分段，但此时和为 s 的一段代价为 $|s-a|^b$，最小化代价

$n\le 10^5$

但在另一些情况下，分治法有它独特的优势。

决策单调性 (2)

[CF 868F] Yet Another Minimization Problem

定义一段的代价是这一段中值相同的元素对数。给定一个长度为 n 的序列，将其分成 k 段，求代价和的最小值。

$n \leq 10^5, k \leq 20$

[gym 102904B] Dispatch Money

给一个长度为 n 的排列，你可以将其任意分段。一段的代价是区间逆序对数量加 k。求每一段代价和的最小值。

$n \leq 3 \times 10^5, 5s$

不难验证 $f_{a,b}$ 满足四边形不等式。但这里有一个严重的问题：f 怎么算？相信大家区间逆序对只会 $O(\sqrt{n})$.jpg

决策单调性 (2)

最经典的区间数颜色相关/区间逆序对做法：大分块莫队。

注意到如果我们维护一些信息后，可以很容易地从 $[l, r]$ 的答案推出 $[l, r+1]$ 或者其它区间长度变化 1 的答案。
例如，对于 $\sum \binom{cnt}{2}$（值相同元素对数），我们只需要维护当前区间内每种颜色出现次数和答案，加入或删除一个颜色时容易更新答案。
对于逆序对，在一侧加入元素时，贡献的逆序对数和区间内比它大/小的元素个数相关，因此可以树状数组维护区间内所有元素。
在标准的莫队做法中，我们可以将询问重新排列，使得顺序考虑 n 个询问时端点移动距离为 $O(n\sqrt{n})$，从而比 $O(n^2)$ 更快地解决问题。

决策单调性 (2)

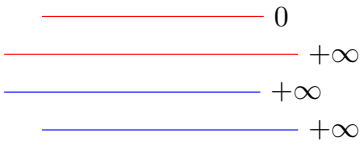
而在分治中，我们有着更好的性质：端点先整体扫一次，然后去左区间递归，再到右端点递归。可以发现整个过程端点的移动距离是 $O(n \log n)$ 的！

因此我们只需要把分治做法改为，每次需要求一个 $f_{a,b}$ 时将两个端点移过去，就可以解决问题。

决策单调性 (3)

一些问题中，每个点可以转移到的区间是有限制的，不是全集。

此时如果直接在外面补 $+\infty$，则可能出现问题。例如，如果限制是转移距离很小：



更好的方式是对转移区间（线段树）分治，变成一些区间转移限制的问题。这样多一个 $\log$。

决策单调性：DP 之外

[gym 101471D] Money for Nothing

给二维平面上 n 个点 (x_i, y_i)，找到两个点 i, j，最大化 $(x_i - x_j)(y_i - y_j)$，即一点为右上角一点为左下角构成的矩形面积。

$n \leq 5 \times 10^5$

「JOISC 2019 Day4」Cake 3

有 n 个元素，每个元素有两个权值 (v_i, c_i)。

给定 k，你需要选出 k 个元素排成一个环。定义一种方案的收益是选出元素 v_i 之和减去环上每一对相邻元素间 c_i 差的绝对值之和。

求最大收益。

$3 \leq k \leq n \leq 2 \times 10^5, 4s$

Table of Contents

DS 优化

有时，一些较为复杂的转移可以用数据结构进行优化。最经典的例子：

$$dp_i = (\min_{k=l_i}^{r_i} dp_k) + ...$$

我们可以用各种支持对应操作的数据结构来维护这个转移。例如线段树，ST 表等。如果 l, r 满足很好的性质，还可以使用单调队列等方式。

DS 优化

「ROI 2017 Day 2」反物质

有 k 种实验，第 i 种实验花费 c_i，会生成 $[l_i, r_i]$ 中随机整数数量的反物质。

你可以进行任意次实验，存储不超过 n 个单位的反物质，若超出则收益为 $-\infty$，否则设停止时有 m 个单位的反物质，则收益为 $m * C$ 减去花费。

求最坏情况下最大收益。

$n \leq 2 \times 10^6, k \leq 100$

$2s, \mathbf{128} \rightarrow \mathbf{32MB}$

DS 优化

对于这种问题，通常处理方式是先观察转移的形式，根据转移形式决定使用什么方式转移。常见的例子包含：

- 前缀和，单调队列，ST 表，...
- 树状数组，线段树，平衡树/set，...
- ~~FFT~~

DS 优化

一道 CF 题

有两个人在接球。初始时两个人都在点 0。有 n 个球顺序落下，第 i 个球落在 x_i。当第 i 个球落下时，需要有一个人去接住这个球。求最后两人移动距离和的最小值。

$n \leq 10^5$

DS 优化

一道题

有一个正整数序列 a，你可以花费 1 的代价将一个数减一。

如果第 i 个位置是非严格前缀最大值，你会获得 c_i 的收益。求最大收益减代价。

$n \leq 5 \times 10^5$

[cf 671D] Roads in Yusland

给一棵 n 个点的有根树，有 m 条链，每条链都形如从一个点到它的某个祖先，每条链还有一个代价。选出一些链使它们覆盖所有边，最小化总代价。

$n, m \leq 3 \times 10^5$

动态 DP

DP，但是带修。

[CF 1286D] LCC

数轴上有 n 个人。每个人速度 v_i 给定。

现在第 i 个人以 p_i 概率向负方向跑，剩余概率往正方向跑。所有人同时开始，求第一次相遇的期望时间。

$n \leq 10^5$

动态 DP

更常见的情况是，带修树形 DP。

常见解决方式是先树剖，然后线段树维护轻儿子的贡献，然后每条重链用另一个线段树维护。

典型例子：

- 带修带权树上独立集（洛谷模板）
- 带修表达式树求值。(「JOI Open 2024」Examination 2)
- 带修各类树形 DP。

重度体力劳动.jpg

Table of Contents

凸优化

温馨提示：对于上凸与下凸，一种说法认为差分不降是上凸，一种认为差分不增是上凸。这里使用前者。

定义一个序列 $v_{0,\dots,n}$ 是上凸的，当且仅当 $\forall i, v_i - v_{i-1} \le v_{i+1} - v_i$。下凸与之相反。

首先考虑什么序列是凸的。一个经典例子是费用流：对于一个最小费用流，设 f_i 表示流量为 i 时的最小代价，那么根据流的性质 f_i 是上凸的。但还有显著更多的东西是凸的，实在不行可以猜

凸优化

凸序列的常见作用是做 $\min, +$ 卷积，即如下形式：

$$f_i = \min_{j+k=i} g_j + h_k$$

对于任意序列，做这个卷积很难优于 $O(n^2)$。但上凸序列上有更优的做法。

凸优化

凸序列的常见作用是做 $\min,+$ 卷积，即如下形式：

$$f_i = \min_{j+k=i} g_j + h_k$$

对于任意序列，做这个卷积很难优于 $O(n^2)$。但上凸序列上有更优的做法。

初始 $f_0 = g_0 + h_0$，然后可以看成有两个序列 $g_1 - g_0, g_2 - g_1, ...$ 和 $h_1 - h_0, h_2 - h_1, ...$。求 f_k 相当于求出在两个序列开头分别拿若干个数，得到的总和的最小值加上 f_0。

但 g, h 上凸，这个差分单调不增。那么选 k 个数让总和最小是简单的：从前往后每次贪心选最小的即可。那么可以 $O(n)$ 求出答案：把差分序列归并即可。这相当于一个闵可夫斯基和。

维护凸序列只需要记录差分序列和 f_0，$\min,+$ 卷积直接归并差分序列，两个凸序列相加则差分序列对应相加。

凸优化

维护凸序列只需要记录差分序列和 f_0，$\min, +$ 卷积直接归并差分序列。

合并两个凸序列的复杂度是两个序列长度之和，因此常见的操作是分治合并。

经典问题

给一条边带权的链，对于每个 k 求大小为 k 的最大匹配。$n \leq 2 \times 10^5$

曾经的国家队集训题

序列划分 k 段，最大化极差之和。

凸优化 + DS

$\min, +$ 卷积直接归并差分序列，两个凸序列相加则差分序列对应相加。

对于更复杂的情况（包含相加等其它修改），直接做复杂度还是很大。

注意到一个平衡树可以很好地维护上述操作：按权值归并和按位置相加。通常来说加的函数会非常简单（例如 $|x-a|, (x-a)^2$，那么可以平衡树上标记解决）

「THUPC 2024 初赛」一棵树

树上选 k 个点。然后对于每条边，设其两侧分别选了 $a, k-a$ 个点，则代价为 $|2a-k|$。最小化所有边的代价之和。

$n \leq 5 \times 10^5$

凸优化 + DS

gym 104128H

树上选 k 个点。然后对于每条边，设其两侧分别选了 $a, k-a$ 个点，则代价为 $c_i a(k-a)(c_i > 0)$。最小化所有边的代价之和。

Table of Contents

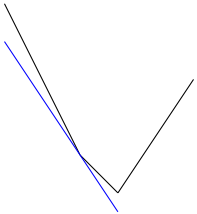
wqs 二分

常见问题：求出单个分 k 段的答案。

凸函数的例子是非常好的：我们可以通过分治对于每一个 k 求出分 k 段的值。

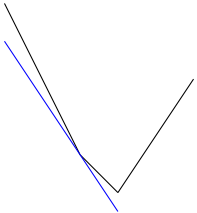
但在之前的那些方法（如斜率优化中），我们不能去选择转移几次，直接记就 $O(n^2)$ 了。

如果答案关于 k 是上凸函数，则有一种技巧被称为 wqs 二分：$\min_i f_i - k*i$ 相当于用斜率 $-k$ 的直线去切答案的函数。我们总可以二分找到一个斜率，使得它正好切到需要的点。



通过这一过程，可以将一个需要求出整个 f 的过程变为 $O(\log v)$ 次求出 $\min_i f_i - k*i$ 的过程。每一步我们需要额外记录最优解转移了几步。

通过这一过程，可以将一个需要求出整个 f 的过程变为 $O(\log v)$ 次求出 $\min_i f_i - k * i$ 的过程。每一步我们需要额外记录最优解转移了几步。



如果答案在某条线段中间怎么办？考虑记录最优解**最少**需要几步。这样就可以在二分的时候判断出这种情况。如果我们只需要答案，这样就够了。

wqs 二分

[hdu 6094] Rikka with K-Match

有一个 $n \times m$ 的网格图，边有边权，求大小为 k 的最小权匹配的权值。

$n \leq 5 \times 10^4, m \leq 4$

wqs 二分

另一个凸的情况是，考虑之前分段的 DP：$dp_{k,i} = \min_{j<i} dp_{k-1,j} + f_{j,i}$。可以证明如果 f 满足四边形不等式，那么对于固定的 n，$dp_{k,n}$ 关于 k 是上凸的。~~证明忘了~~

同时，将所有 f 减去同一个值不影响 f 满足四边形不等式，因此这类问题都可以使用 wqs 优化。换言之，对于决策单调性部分提到的问题，它们都可以 $1\log$ 变为要求正好划分 k 段的版本。~~这大概也是为啥之前会有区间逆序对数 +k 系数的原因，但直接做复杂度是 $\log^4$ 甚至 $\log^3$~~

wqs 二分

如果答案在某条线段中间怎么办？考虑记录最优解**最少**需要几步。这样就可以在二分的时候判断出这种情况。如果我们只需要答案，这样就够了。

如何求出方案？

如果答案在某条线段中间怎么办？考虑记录最优解**最少**需要几步。这样就可以在二分的时候判断出这种情况。如果我们只需要答案，这样就够了。

如何求出方案？

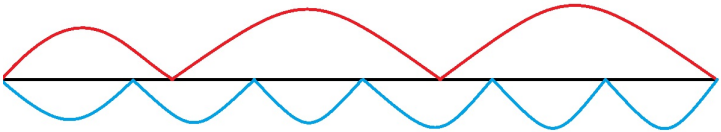
假设需要求 k 步的方案。考虑分别求出转移时尽量选走的步数最少的路径，和转移时尽量选走的步数最多的路径，设这两种方案分别走了 a, b 步。

如果答案在某条线段中间怎么办？考虑记录最优解**最少**需要几步。这样就可以在二分的时候判断出这种情况。如果我们只需要答案，这样就够了。

如何求出方案？

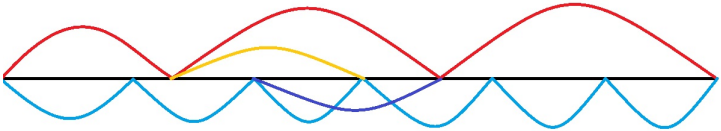
假设需要求 k 步的方案。考虑分别求出转移时尽量选走的步数最少的路径，和转移时尽量选走的步数最多的路径，设这两种方案分别走了 a, b 步。

假设两种方案的路径依次如下：



可以发现，因为 $a < k < b$，一定存在一段蓝色边被一段红色边完全包含。

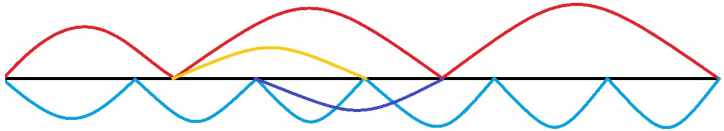
这时考虑这两条路径：



在这个完全被包含的位置调整，之后的路径互相交换。

根据四边形不等式，新加入的两条边权和不大于原来这里的两条边权和。因此新的两条路径权值和不大于之前的。

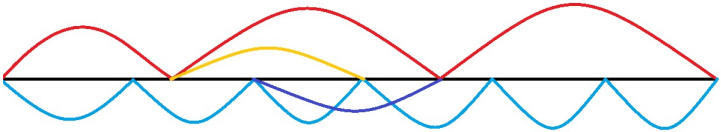
这时考虑这两条路径：



在这个完全被包含的位置调整，之后的路径互相交换。

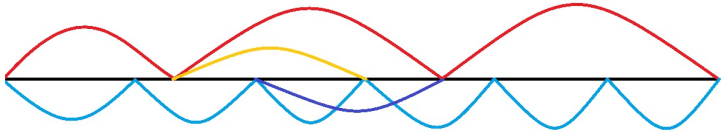
根据四边形不等式，新加入的两条边权和不大于原来这里的两条边权和。因此新的两条路径权值和不大于之前的。

注意到在任意一个被完全包含的蓝边处都可以进行调整，在这个调整的位置左侧，每有一个被一条蓝边完全包含的红边，调整后蓝-红路径的长度会 -1。每有一个被一条红边完全包含的蓝边，调整后蓝-红路径的长度会 $+1$。因而一定可以找到一条这样的蓝边进行调整，使得得到的一条路径边数为 k。



具体来说，取蓝-红的这一条路径。那么一定存在一个蓝色路径上的点，使得路径为蓝色路径从开头在这个点的部分拼接上这个点后面的第一个红色点开始到结尾的部分。

考虑这个点需要满足的条件：



具体来说，取蓝-红的这一条路径。那么一定存在一个蓝色路径上的点，使得路径为蓝色路径从开头在这个点的部分拼接上这个点后面的第一个红色点开始到结尾的部分。

考虑这个点需要满足的条件：

- 这个点之后的下一个蓝色点比红色点更近。
- 这样拼接出来的路径长度为需要的长度。

因为一定有解，所以找一个这样的点即可。这个调整的复杂度为 $O(n)$。

人话：找到一个红分界点和一个蓝分界点，使得中间正好有一个蓝分界点（且数量对），然后选择红色的前缀和蓝色的后缀。

wqs 二分

[gym 102268J] Jealous Split

将**非负**序列划分为 k 段，使得相邻两段满足如下条件：相邻两点元素和之差不超过相邻两段内元素最大值。

$n \leq 10^5$

Table of Contents

组合意义

普及组模拟赛题目

给一棵 n 个点的树，你有 2^{n-1} 种方式删掉一些边，对所有方式求和最后得到的每个连通块大小乘积。

$n \leq 10^6$

[IOI2022] Digital Circuit

有一棵有根树。每个叶子有给定权值 $x_i \in \{0, 1\}$。

对于每个非叶子节点 u，设它有 k 个儿子，则它应当有一个 $[1, k]$ 间的整数参数 t_u。其值为 1 当且仅当有至少 t_u 个儿子值为 1。

q 次询问，每次区间翻转 x_i，然后求有多少种参数使得根节点值为 1。

$n, q \leq 10^5$

组合意义

部分情况下，DP 的系数具有很好的组合意义。此时可以通过加入一些状态，使用这些状态的转移凑出 DP 的系数。

常见的组合意义：

- 一个大小为 n 的集合权值为 a^n，等价于每选一个数乘上 a 的权值。

组合意义

部分情况下，DP 的系数具有很好的组合意义。此时可以通过加入一些状态，使用这些状态的转移凑出 DP 的系数。

常见的组合意义：

- 一个大小为 n 的集合权值为 a^n，等价于每选一个数乘上 a 的权值。
- 一个大小为 n 的集合权值为 n，等价于在集合中选一个数。

组合意义

部分情况下，DP 的系数具有很好的组合意义。此时可以通过加入一些状态，使用这些状态的转移凑出 DP 的系数。

常见的组合意义：

- 一个大小为 n 的集合权值为 a^n，等价于每选一个数乘上 a 的权值。
- 一个大小为 n 的集合权值为 n，等价于在集合中选一个数。
- 一个大小为 n 的集合权值为 $\binom{n}{k}$，等价于选出 k 个不同的数。如果权值是 n^k，可以用一些方式拆成一些 $\binom{n}{i}$ 的和。

组合意义

部分情况下，DP 的系数具有很好的组合意义。此时可以通过加入一些状态，使用这些状态的转移凑出 DP 的系数。

常见的组合意义：

- 一个大小为 n 的集合权值为 a^n，等价于每选一个数乘上 a 的权值。
- 一个大小为 n 的集合权值为 n，等价于在集合中选一个数。
- 一个大小为 n 的集合权值为 $\binom{n}{k}$，等价于选出 k 个不同的数。如果权值是 n^k，可以用一些方式拆成一些 $\binom{n}{i}$ 的和。
- 还有一些直观的组合意义。

组合意义典例

[CF 1278F] Cards

抽 n 次牌，每次有 $\frac{1}{m}$ 的概率抽出 Joker。记抽到 Joker 的次数为 x，求 x^k 的期望。

$k \leq 5000$

[arc 124e] Pass to Next

一个环上有 n 个人，初始第 i 个人有 a_i 个球。

所有人**同时**进行如下操作：拿出自己球的一部分，扔给下一个人。设操作结束后第 i 个人有 b_i 个球。

对于所有可能得到的序列 $\{b_i\}$，求和 $\prod b_i$。

$n \leq 10^5$

[agc 001e] BBQ Hard

给 n 对 (a_i, b_i)，求

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\binom{a_i+b_i+a_j+b_j}{a_i+a_j}$$

答案模 998244353。

$n, m \le 2\times 10^5$, $a_i, b_i \le 2000$

值域与分界点

[gym 104560A] Costly Binary Search

给一个长度为 n 的正整数序列 c。考虑如下问题：

你需要在 $[1, n]$ 中二分查找一个数 x。你可以进行如下询问：

- 花费 c_i 的代价，比较 x 与 i 的大小关系，返回小于、等于、大于中的一种。

最小化最坏情况的代价，

$n \leq 10^6, c_i \leq \mathbf{9}$

值域与分界点

[agc 033d] Complexity

定义一个 01 矩阵的代价为：

- 如果矩阵全 0 或全 1，则代价为 0。
- 否则，选择一种横竖切开矩阵的方式，这种方式的代价为划分出的两个矩阵的代价最大值再加一。矩阵的代价为所有切开矩阵方式代价的最小值。

给一个 $n \times m$ 的 01 矩阵，求出其代价。

$n, m \leq 185$

值域与分界点

这种方法相当于压缩了 DP 的状态，只保留一部分关键的分界点。如果能只剩下很少的关键点，则显然可以降低复杂度。对于单调 + 小值域这一做法非常常见。

但这样的压缩通常会带来一些细节问题：因为压缩了状态，转移可能变得有点奇怪。常见的操作是每种转移可以分别在压缩后的状态上转移，但最后需要把它们合并起来，合并会涉及到一些细节。

维护凸函数的斜率变化点也可以看成这种思想的体现。

dp of dp

一种设计 DP 状态的方式：用一个 DP 作为另一个 DP 的状态。

常见的情况是，原本有一个判定合法的问题，现在将其变为统计合法方案数的问题。

求两个序列的 LCS

大家都会 $O(nm)$ 的 DP：记 s 的前 i 个字符和 t 的前 j 个字符的 LCS 为 $f_{i,j}$，容易进行转移。

「TJOI2018」party

给定 n 和一个长度为 m 的序列 t，对于每个 k 求所有长度为 n 的序列中有多少个序列和 t 的 LCS 为 k。

$n \leq 1000, m \leq 15$，字符集大小为 3

在这一过程中，通常还需要对状态进行精细的设计，甚至猜状态数很小。

[CF 578D] LCS Again

给一个长度为 n 的字符串，求有多少个长度为 n 的字符串 t 满足 s, t 的 LCS 为 $n-1$。

$n \leq 10^5$

dp of dp

很多统计合法方案数得问题事实上都可以看成 dp of dp，但这些问题通常最后可以推出更好形式的限制，使得可以通过直接的状态判定（例如选了多少数，和是多少等）。

更多的时候，多推推性质比直接 dp of dp 有效。但如果没性质了还是需要 dp of dp。

dp of dp

很多统计合法方案数得问题事实上都可以看成 dp of dp，但这些问题通常最后可以推出更好形式的限制，使得可以通过直接的状态判定（例如选了多少数，和是多少等）。

更多的时候，多推推性质比直接 dp of dp 有效。但如果没性质了还是需要 dp of dp。

一些更极端的状态数量例子：

- 「USACO 2022.2 Platinum」Phone Numbers
- 「ZJOI2019」麻将
- 「NOI2022」移除石子

Thanks!

强烈建议实现各大例题以加强认识/手感。

Thanks!

强烈建议实现各大例题以加强认识/手感。

额外推荐题目：~~难度接近递增，应该基本上每一道都是 NOI/NOI+ 难度~~

1. (gym 102586B) Evacuation
2. (loj 3634) 「2021 集训队互测」音符大师
3. (loj 3643) 「2021 集训队互测」球球
4. (agc 049e) Increment Decrement
5. (agc 058f) Authentic Tree DP