动态规划 1

王逸凡

2017.07.21

动态规划简述

动态规划简述

- 动态规划在 OI 中占了很大的比重。一般 NOIP 和 NOI 等考试，两试六道题中动态规划会有一题；一场 ACM 的比赛中，动态规划会有一到两题。

动态规划简述

- 动态规划在 OI 中占了很大的比重。一般 NOIP 和 NOI 等考试，两试六道题中动态规划会有一题；一场 ACM 的比赛中，动态规划会有一到两题。
- 动态规划的核心思想是最优子结构，即通过子问题的最优解推出更大的问题的最优解。还有很大一部分的计数类问题也具有子结构的特点（不存在“最优”），我们也可以广义地把它称为动态规划问题。

动态规划简述

动态规划简述

- 很多动态规划问题都是没有专门的分类的，这是因为每道动态规划问题都有自己的特点。所以，要想做好动态规划问题，需要进行大量的练习，并总结一些常见的思维套路。

动态规划简述

- 很多动态规划问题都是没有专门的分类的，这是因为每道动态规划问题都有自己的特点。所以，要想做好动态规划问题，需要进行大量的练习，并总结一些常见的思维套路。
- 一些动态规划问题是可以具体分类的，比如：区间 DP、树型 DP、状压 DP、概率 DP、数位 DP、插头 DP、DAG（有向无环图）上的 DP 等等。

动态规划简述

- 很多动态规划问题都是没有专门的分类的，这是因为每道动态规划问题都有自己的特点。所以，要想做好动态规划问题，需要进行大量的练习，并总结一些常见的思维套路。
- 一些动态规划问题是可以具体分类的，比如：区间 DP、树型 DP、状压 DP、概率 DP、数位 DP、插头 DP、DAG（有向无环图）上的 DP 等等。
- 根据动态规划的优化方式，我们可以从另一个角度对问题进行分类，比如：斜率优化、决策单调性优化、利用数据结构进行优化等等。

内容安排

内容安排

- 第一部分：一些不好分类的动态规划题目。

内容安排

- 第一部分：一些不好分类的动态规划题目。
- 第二部分：一些常见的 DP 优化套路。

内容安排

- 第一部分：一些不好分类的动态规划题目。
- 第二部分：一些常见的 DP 优化套路。
- 第三部分：一些典型的动态规划分类。

最大正方形

最大正方形

- 给定 $n \times n$ 的一个 01 矩阵，求矩阵中的一个最大的正方形，使得这个正方形中的元素都是 0。

最大正方形

- 给定 $n \times n$ 的一个 01 矩阵，求矩阵中的一个最大的正方形，使得这个正方形中的元素都是 0。
- $n \leq 1000$。

最大正方形

最大正方形

- 设 $f_{i,j}$ 表示以 (i,j) 为右下角的最大正方形边长。

最大正方形

- 设 $f_{i,j}$ 表示以 (i,j) 为右下角的最大正方形边长。
- $f_{i,j} = \min\{f_{i-1,j}, f_{i,j-1}, f_{i-1,j-1}\} + 1$。

最大正方形

- 设 $f_{i,j}$ 表示以 (i,j) 为右下角的最大正方形边长。
- $f_{i,j} = \min\{f_{i-1,j}, f_{i,j-1}, f_{i-1,j-1}\} + 1$。
- 方程成立的条件?

最长公共子序列

最长公共子序列

- 给定两个序列 $a_1, a_2, ..., a_n$，及 $b_1, b_2, ..., b_m$。

最长公共子序列

- 给定两个序列 $a_1, a_2, ..., a_n$，及 $b_1, b_2, ..., b_m$。
- 你需要选出一个序列片段，使得这个这个片段在两个序列中都有出现过（不需要连续）。

最长公共子序列

- 给定两个序列 $a_1, a_2, ..., a_n$， 及 $b_1, b_2, ..., b_m$。
- 你需要选出一个序列片段，使得这个这个片段在两个序列中都有出现过（不需要连续）。
- 比如"1232" 和"1323" 的最长公共子序列就是"132"。

最长公共子序列

- 给定两个序列 $a_1, a_2, ..., a_n$， 及 $b_1, b_2, ..., b_m$。
- 你需要选出一个序列片段，使得这个这个片段在两个序列中都有出现过（不需要连续）。
- 比如"1232" 和"1323" 的最长公共子序列就是"132"。
- $n, m \leq 1000$。

最长公共子序列

最长公共子序列

- 设 $f_{i,j}$ 表示第一个序列选择到 i，第二个序列选择到 j 时，所选择出的最长子序列长度。

最长公共子序列

- 设 $f_{i,j}$ 表示第一个序列选择到 i，第二个序列选择到 j 时，所选择出的最长子序列长度。
- 如果 a_i 不入选：$f_{i,j} = f_{i-1,j}$。

最长公共子序列

- 设 $f_{i,j}$ 表示第一个序列选择到 i，第二个序列选择到 j 时，所选择出的最长子序列长度。
- 如果 a_i 不入选：$f_{i,j} = f_{i-1,j}$。
- 如果 b_j 不入选：$f_{i,j} = f_{i,j-1}$。

最长公共子序列

- 设 $f_{i,j}$ 表示第一个序列选择到 i，第二个序列选择到 j 时，所选择出的最长子序列长度。
- 如果 a_i 不入选：$f_{i,j} = f_{i-1,j}$。
- 如果 b_j 不入选：$f_{i,j} = f_{i,j-1}$。
- 如果 a_i 和 b_j 都入选：$f_{i,j} = f_{i-1,j-1} + 1$，但是要求 $a_i = b_j$ 成立。

最长公共子序列

- 设 $f_{i,j}$ 表示第一个序列选择到 i，第二个序列选择到 j 时，所选择出的最长子序列长度。
- 如果 a_i 不入选：$f_{i,j} = f_{i-1,j}$。
- 如果 b_j 不入选：$f_{i,j} = f_{i,j-1}$。
- 如果 a_i 和 b_j 都入选：$f_{i,j} = f_{i-1,j-1} + 1$，但是要求 $a_i = b_j$ 成立。
- 在上面三种选择中挑一个大的。

Crisis

Crisis

- 一个无限大的地图上有一些士兵和一些旗子。

Crisis

- 一个无限大的地图上有一些士兵和一些旗子。
- 司令员可以下达四种口令：分别让所有士兵向东、西、南、北前进一个单位。

Crisis

- 一个无限大的地图上有一些士兵和一些旗子。
- 司令员可以下达四种口令：分别让所有士兵向东、西、南、北前进一个单位。
- 如果执行完一个指令后，有 K 个士兵到达了有旗子的位置，那么司令员可以得到 K 分。

Crisis

- 一个无限大的地图上有一些士兵和一些旗子。
- 司令员可以下达四种口令：分别让所有士兵向东、西、南、北前进一个单位。
- 如果执行完一个指令后，有 K 个士兵到达了有旗子的位置，那么司令员可以得到 K 分。
- 问司令员在最多下达 C 个指令的情况下，最多能得到几分。

Crisis

- 一个无限大的地图上有一些士兵和一些旗子。
- 司令员可以下达四种口令：分别让所有士兵向东、西、南、北前进一个单位。
- 如果执行完一个指令后，有 K 个士兵到达了有旗子的位置，那么司令员可以得到 K 分。
- 问司令员在最多下达 C 个指令的情况下，最多能得到几分。
- 士兵数量 ≤ 1000，旗子数量 ≤ 1000，指令数量 ≤ 30。

Crisis

Crisis

- 所有士兵的相对位置不变，所以得分的情况只和一个士兵与自己的起始点的相对位置有关。

Crisis

- 所有士兵的相对位置不变，所以得分的情况只和一个士兵与自己的起始点的相对位置有关。
- 设 $a_{i,j}$ 表示所有士兵和自己的相对位置距离 (i, j) 时，得分是多少。

Crisis

- 所有士兵的相对位置不变，所以得分的情况只和一个士兵与自己的起始点的相对位置有关。
- 设 $a_{i,j}$ 表示所有士兵和自己的相对位置距离 (i,j) 时，得分是多少。
- 设 $f_{i,j,k}$ 表示司令员下达了 k 个指令，士兵离起始点的相对位置为 (i,j) 时，能获得的最大分数。

Crisis

- 所有士兵的相对位置不变，所以得分的情况只和一个士兵与自己的起始点的相对位置有关。
- 设 $a_{i,j}$ 表示所有士兵和自己的相对位置距离 (i,j) 时，得分是多少。
- 设 $f_{i,j,k}$ 表示司令员下达了 k 个指令，士兵离起始点的相对位置为 (i,j) 时，能获得的最大分数。
- 则 $f_{i,j,k} = \max\{f_{i\pm1,j\pm1,k-1}\} + a_{i,j}$

互质

互质

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。

互质

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。
- 你需要挑出这个序列的一个子序列，使得这个子序列的任意两个相邻元素不互质。

互质

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。
- 你需要挑出这个序列的一个子序列，使得这个子序列的任意两个相邻元素不互质。
- $n \le 1000$，$a_i \le 100000$。

互质

互质

- 设 f_i 表示选择到第 i 个元素，并且以这个元素结尾时，最长的子序列长度。

互质

- 设 f_i 表示选择到第 i 个元素，并且以这个元素结尾时，最长的子序列长度。
- 则 $f_i = \max\{f_j\} + 1$（如果第 i 个元素和第 j 个元素不互质）。

互质

- 设 f_i 表示选择到第 i 个元素，并且以这个元素结尾时，最长的子序列长度。
- 则 $f_i = \max\{f_j\} + 1$（如果第 i 个元素和第 j 个元素不互质）。
- 时间复杂度 $O(n^2)$。

互质

- 设 f_i 表示选择到第 i 个元素，并且以这个元素结尾时，最长的子序列长度。
- 则 $f_i = \max\{f_j\} + 1$（如果第 i 个元素和第 j 个元素不互质）。
- 时间复杂度 $O(n^2)$。
- $n \le 100000$?

互质

- 设 f_i 表示选择到第 i 个元素，并且以这个元素结尾时，最长的子序列长度。
- 则 $f_i = \max\{f_j\} + 1$（如果第 i 个元素和第 j 个元素不互质）。
- 时间复杂度 $O(n^2)$。
- $n \le 100000$?
- 优化的关键在于：两个数不互质，只需要它们有共同的一个质因子即可。

互质

互质

- 求出 100000 内的所有质数。

互质

- 求出 100000 内的所有质数。
- 设 $g_{i,j}$ 表示只用前 i 个元素（这时不需要保证第 i 个元素在结尾），并且最后一个元素包含第 j 个质数。

互质

- 求出 100000 内的所有质数。
- 设 $g_{i,j}$ 表示只用前 i 个元素（这时不需要保证第 i 个元素在结尾），并且最后一个元素包含第 j 个质数。
- 如果 a_i 不包含包含第 j 个质数，那么 $g_{i,j} = g_{i-1,j}$。

互质

- 求出 100000 内的所有质数。
- 设 $g_{i,j}$ 表示只用前 i 个元素（这时不需要保证第 i 个元素在结尾），并且最后一个元素包含第 j 个质数。
- 如果 a_i 不包含包含第 j 个质数，那么 $g_{i,j} = g_{i-1,j}$。
- 如果 a_i 包含包含第 j 个质数，同时它包含第 k 个质数，那么 $g_{i,j} = \max\{g_{i-1,k}\} + 1$。

互质

互质

- 但是，我们不可能完完全全按上面的 DP 方程做，这样做在空间上就已经不能承受了。

互质

- 但是，我们不可能完完全全按上面的 DP 方程做，这样做在空间上就已经不能承受了。
- 注意到 $g_{i,*}$ 和 $g_{i-1,*}$ 的差别是很小的，即大部分的值都是一样的。

互质

- 但是，我们不可能完完全全按上面的 DP 方程做，这样做在空间上就已经不能承受了。
- 注意到 $g_{i,*}$ 和 $g_{i-1,*}$ 的差别是很小的，即大部分的值都是一样的。
- 所以，我们可以考虑不把 i 这一维建出，而是直接在原数组上做修改。

互质

- 但是，我们不可能完完全全按上面的 DP 方程做，这样做在空间上就已经不能承受了。
- 注意到 $g_{i,*}$ 和 $g_{i-1,*}$ 的差别是很小的，即大部分的值都是一样的。
- 所以，我们可以考虑不把 i 这一维建出，而是直接在原数组上做修改。
- 一个数 d 的质因子个数不超过 $\log d$，所以总时间复杂度是 $O(n\log^2 a)$ 的。

架设电话线

架设电话线

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。

架设电话线

- 给定一个正整数序列 $a_1, a_2, \ldots, a_n$。
- 你需要对这个序列进行一些调整，调整的方法是对序列的某一些元素加上一个值。不同的元素可以加不同的值。

架设电话线

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。
- 你需要对这个序列进行一些调整，调整的方法是对序列的某一些元素加上一个值。不同的元素可以加不同的值。
- 如果你对一个元素加上了 X，那么你需要付出 X^2 的代价。

架设电话线

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。
- 你需要对这个序列进行一些调整，调整的方法是对序列的某一些元素加上一个值。不同的元素可以加不同的值。
- 如果你对一个元素加上了 X，那么你需要付出 X^2 的代价。
- 在修改完序列后，如果相邻两个元素相差了 Y，你需要付出 $C \times Y$ 的代价（C 是一个给定的数）。

架设电话线

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。
- 你需要对这个序列进行一些调整，调整的方法是对序列的某一些元素加上一个值。不同的元素可以加不同的值。
- 如果你对一个元素加上了 X，那么你需要付出 X^2 的代价。
- 在修改完序列后，如果相邻两个元素相差了 Y，你需要付出 $C \times Y$ 的代价（C 是一个给定的数）。
- 你的目标是最小化付出的代价总和。

架设电话线

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。
- 你需要对这个序列进行一些调整，调整的方法是对序列的某一些元素加上一个值。不同的元素可以加不同的值。
- 如果你对一个元素加上了 X，那么你需要付出 X^2 的代价。
- 在修改完序列后，如果相邻两个元素相差了 Y，你需要付出 $C \times Y$ 的代价（C 是一个给定的数）。
- 你的目标是最小化付出的代价总和。
- $n \leq 100000$，$a_i \leq 100$。

架设电话线

架设电话线

- 设 $f_{i,j}$ 表示把 a_i 修改为 j 的代价（j 不能比 a_i 小）。

架设电话线

- 设 $f_{i,j}$ 表示把 a_i 修改为 j 的代价（j 不能比 a_i 小）。
- 则 $f_{i,j} = \min\{(j - a_i)^2 + f_{i-1,k} + C \times |k - j|\}$。

架设电话线

- 设 $f_{i,j}$ 表示把 a_i 修改为 j 的代价（j 不能比 a_i 小）。
- 则 $f_{i,j} = \min\{(j - a_i)^2 + f_{i-1,k} + C \times |k - j|\}$。
- 上面的方程中，i、j、k 都需要枚举，总复杂度为 $O(nm^2)$（m 为 a_i 的最大值）。

架设电话线

- 设 $f_{i,j}$ 表示把 a_i 修改为 j 的代价（j 不能比 a_i 小）。
- 则 $f_{i,j} = \min\{(j - a_i)^2 + f_{i-1,k} + C \times |k - j|\}$。
- 上面的方程中，i、j、k 都需要枚举，总复杂度为 $O(nm^2)$（m 为 a_i 的最大值）。
- 如果已知 k 比 j 小，那么方程可以写成 $f_{i,j} = \min\{(j - a_i)^2 + f_{i-1,k} - C \times k + C \times j\}$。

架设电话线

- 设 $f_{i,j}$ 表示把 a_i 修改为 j 的代价（j 不能比 a_i 小）。
- 则 $f_{i,j} = \min\{(j - a_i)^2 + f_{i-1,k} + C \times |k - j|\}$。
- 上面的方程中，i、j、k 都需要枚举，总复杂度为 $O(nm^2)$（m 为 a_i 的最大值）。
- 如果已知 k 比 j 小，那么方程可以写成 $f_{i,j} = \min\{(j - a_i)^2 + f_{i-1,k} - C \times k + C \times j\}$。
- 这样，如果我们能预先求出 $f_{i-1,k} - C \times k$ 部分的最小值，就不需要枚举 k 了。

架设电话线

架设电话线

- 设 $g_{i,j}$ 表示对所有不超过 j 的 k，$f_{i-1,k} - C \times k$ 的最小值。

架设电话线

- 设 $g_{i,j}$ 表示对所有不超过 j 的 k，$f_{i-1,k} - C \times k$ 的最小值。
- 则 $g_{i,j} = \min\{g_{i,j-1}, f_{i,j} - C \times j\}$。

- 设 $g_{i,j}$ 表示对所有不超过 j 的 k，$f_{i-1,k} - C \times k$ 的最小值。
- 则 $g_{i,j} = \min\{g_{i,j-1}, f_{i,j} - C \times j\}$。
- 那么，原来的方程在 $k \le j$ 时，就能写成 $f_{i,j} = (j - a_i)^2 + g_{i-1,j} + C \times j$。

架设电话线

- 设 $g_{i,j}$ 表示对所有不超过 j 的 k，$f_{i-1,k} - C \times k$ 的最小值。
- 则 $g_{i,j} = \min\{g_{i,j-1}, f_{i,j} - C \times j\}$。
- 那么，原来的方程在 $k \le j$ 时，就能写成 $f_{i,j} = (j - a_i)^2 + g_{i-1,j} + C \times j$。
- 同样地，我们可以对 $k > j$ 的部分做类似的处理。

架设电话线

- 设 $g_{i,j}$ 表示对所有不超过 j 的 k，$f_{i-1,k} - C \times k$ 的最小值。
- 则 $g_{i,j} = \min\{g_{i,j-1}, f_{i,j} - C \times j\}$。
- 那么，原来的方程在 $k \leq j$ 时，就能写成 $f_{i,j} = (j - a_i)^2 + g_{i-1,j} + C \times j$。
- 同样地，我们可以对 $k > j$ 的部分做类似的处理。
- 这样，复杂度就可以降为 $O(nm)$。

游戏

游戏

- 给定两个正整数序列 $a_1, a_2, ..., a_n$， 及 $b_1, b_2, ..., b_m$

游戏

- 给定两个正整数序列 $a_1, a_2, ..., a_n$，及 $b_1, b_2, ..., b_m$
- 每次你可以在第一个序列末位选择 k_1 个元素，设它们的和是 s_1；在第二个序列末位选择 k_2 个元素，设它们的和是 s_2；接着把它们删掉，你会获得的分数是 $(s_1 - k_1) \times (s_2 - k_2)$。

游戏

- 给定两个正整数序列 $a_1, a_2, \ldots, a_n$，及 $b_1, b_2, \ldots, b_m$
- 每次你可以在第一个序列末位选择 k_1 个元素，设它们的和是 s_1；在第二个序列末位选择 k_2 个元素，设它们的和是 s_2；接着把它们删掉，你会获得的分数是 $(s_1 - k_1) \times (s_2 - k_2)$。
- 你需要最小化你的得分。

游戏

- 给定两个正整数序列 $a_1, a_2, ..., a_n$，及 $b_1, b_2, ..., b_m$
- 每次你可以在第一个序列末位选择 k_1 个元素，设它们的和是 s_1；在第二个序列末位选择 k_2 个元素，设它们的和是 s_2；接着把它们删掉，你会获得的分数是 $(s_1 - k_1) \times (s_2 - k_2)$。
- 你需要最小化你的得分。
- $n, m \leq 1000$。

游戏

游戏

- 首先，我们可以把序列的每个元素都减一。这样 k_1 和 k_2 就可以去掉了，每次的得分变成 $s_1 \times s_2$。

游戏

- 首先，我们可以把序列的每个元素都减一。这样 k_1 和 k_2 就可以去掉了，每次的得分变成 $s_1 \times s_2$。
- 设 $f_{i,j}$ 表示把第一个序列删到剩 i 个元素，第二个序列删到剩 j 个元素时的最小得分。

游戏

- 首先，我们可以把序列的每个元素都减一。这样 k_1 和 k_2 就可以去掉了，每次的得分变成 $s_1 \times s_2$。
- 设 $f_{i,j}$ 表示把第一个序列删到剩 i 个元素，第二个序列删到剩 j 个元素时的最小得分。
- 这个状态设计已经没有优化的空间了，但它占据了 $O(n^2)$ 的复杂度。所以我们只能从状态转移上做一个突破。

游戏

游戏

- 最朴素的状态转移：枚举上一次两个序列分别删除了多少个元素。

游戏

- 最朴素的状态转移：枚举上一次两个序列分别删除了多少个元素。
- 总复杂度高达 $O(n^4)$。

游戏

- 最朴素的状态转移：枚举上一次两个序列分别删除了多少个元素。
- 总复杂度高达 $O(n^4)$。
- 性质：每次删除元素时，至少有一个序列只删除了一个元素。

游戏

- 最朴素的状态转移：枚举上一次两个序列分别删除了多少个元素。
- 总复杂度高达 $O(n^4)$。
- 性质：每次删除元素时，至少有一个序列只删除了一个元素。
- 证明：如果两个序列都删除了多个，把这些元素分成两次删显然得分更少。

游戏

- 最朴素的状态转移：枚举上一次两个序列分别删除了多少个元素。
- 总复杂度高达 $O(n^4)$。
- 性质：每次删除元素时，至少有一个序列只删除了一个元素。
- 证明：如果两个序列都删除了多个，把这些元素分成两次删显然得分更少。
- 这样每次枚举的时候可以假定一个序列只选取一个，再进行转移。

游戏

游戏

- 但是这样的复杂度仍然是 $O(n^3)$ 的，还是不能接受。

游戏

- 但是这样的复杂度仍然是 $O(n^3)$ 的，还是不能接受。
- 先写转移方程：$f_{i,j} = \min\{f_{i-1,j-1}, f_{i,j-1}, f_{i-1,j}\} + a_i \times b_j$。

游戏

- 但是这样的复杂度仍然是 $O(n^3)$ 的，还是不能接受。
- 先写转移方程：$f_{i,j} = \min\{f_{i-1,j-1}, f_{i,j-1}, f_{i-1,j}\} + a_i \times b_j$。
- 取最小值的三项中，第一项的写法很显然。下面分析第二项的含义：

游戏

- 但是这样的复杂度仍然是 $O(n^3)$ 的，还是不能接受。
- 先写转移方程：$f_{i,j} = \min\{f_{i-1,j-1}, f_{i,j-1}, f_{i-1,j}\} + a_i \times b_j$。
- 取最小值的三项中，第一项的写法很显然。下面分析第二项的含义：
- 考虑取到 $(i, j-1)$ 时，最后一次删除的情况。

游戏

- 但是这样的复杂度仍然是 $O(n^3)$ 的，还是不能接受。
- 先写转移方程：$f_{i,j} = \min\{f_{i-1,j-1}, f_{i,j-1}, f_{i-1,j}\} + a_i \times b_j$。
- 取最小值的三项中，第一项的写法很显然。下面分析第二项的含义：
- 考虑取到 $(i, j-1)$ 时，最后一次删除的情况。
- 如果情况是 a_i 一个对 b 序列的多个，第二项的写法相当于多取一个。

游戏

游戏

- 如果是 b_{j-1} 一个对 a 序列多个，那么真实的转移情况应该是：b_{j-1} 本来和 a_i 相对，去掉，然后改成 a_i 和 b_j 同时取。如果是这样的情况，我们应该能得到一个比给出的方程更小的转移。

游戏

- 如果是 b_{j-1} 一个对 a 序列多个，那么真实的转移情况应该是：b_{j-1} 本来和 a_i 相对，去掉，然后改成 a_i 和 b_j 同时取。如果是这样的情况，我们应该能得到一个比给出的方程更小的转移。
- 所以，上面给出的方程是不会产生比最优解还小的 DP 值的。

游戏

- 如果是 b_{j-1} 一个对 a 序列多个，那么真实的转移情况应该是：b_{j-1} 本来和 a_i 相对，去掉，然后改成 a_i 和 b_j 同时取。如果是这样的情况，我们应该能得到一个比给出的方程更小的转移。
- 所以，上面给出的方程是不会产生比最优解还小的 DP 值的。
- 另一个问题：最优的情况会不会取不到?

游戏

- 如果是 b_{j-1} 一个对 a 序列多个，那么真实的转移情况应该是：b_{j-1} 本来和 a_i 相对，去掉，然后改成 a_i 和 b_j 同时取。如果是这样的情况，我们应该能得到一个比给出的方程更小的转移。
- 所以，上面给出的方程是不会产生比最优解还小的 DP 值的。
- 另一个问题：最优的情况会不会取不到?
- 不会。如果发生了第二种列出的情况，这样的情况应该会从 $f_{i-1,j-1}$ 转移过来。

游戏

- 如果是 b_{j-1} 一个对 a 序列多个，那么真实的转移情况应该是：b_{j-1} 本来和 a_i 相对，去掉，然后改成 a_i 和 b_j 同时取。如果是这样的情况，我们应该能得到一个比给出的方程更小的转移。
- 所以，上面给出的方程是不会产生比最优解还小的 DP 值的。
- 另一个问题：最优的情况会不会取不到？
- 不会。如果发生了第二种列出的情况，这样的情况应该会从 $f_{i-1,j-1}$ 转移过来。
- 综合以上情况，我们可以在 $O(n^2)$ 的时间内就完成 DP。

打扫卫生

打扫卫生

- 给定一个序列 $a_1, a_2, ..., a_n$。

打扫卫生

- 给定一个序列 $a_1, a_2, ..., a_n$。
- 你需要把这个序列切成若干段（即每一段里的元素是连续的）。

打扫卫生

- 给定一个序列 $a_1, a_2, ..., a_n$。
- 你需要把这个序列切成若干段（即每一段里的元素是连续的）。
- 对于每一段，如果这段里有 k 个不同的数，它产生的代价是 k^2。

打扫卫生

- 给定一个序列 $a_1, a_2, ..., a_n$。
- 你需要把这个序列切成若干段（即每一段里的元素是连续的）。
- 对于每一段，如果这段里有 k 个不同的数，它产生的代价是 k^2。
- 你需要最小化代价。

打扫卫生

- 给定一个序列 $a_1, a_2, ..., a_n$。
- 你需要把这个序列切成若干段（即每一段里的元素是连续的）。
- 对于每一段，如果这段里有 k 个不同的数，它产生的代价是 k^2。
- 你需要最小化代价。
- $n, a_i \leq 40000$。

打扫卫生

打扫卫生

- 还是先考虑暴力做法：设 f_i 表示到 a_i 时恰好切了一段时的最小代价，再设 $g_{j,i}$ 表示从 j 到 i 这一段里有多少个不同的数。

打扫卫生

- 还是先考虑暴力做法：设 f_i 表示到 a_i 时恰好切了一段时的最小代价，再设 $g_{j,i}$ 表示从 j 到 i 这一段里有多少个不同的数。
- $f_i = \min\{f_j + g_{j+1,i}\}$。

打扫卫生

- 还是先考虑暴力做法：设 f_i 表示到 a_i 时恰好切了一段时的最小代价，再设 $g_{j,i}$ 表示从 j 到 i 这一段里有多少个不同的数。
- $f_i = \min\{f_j + g_{j+1,i}\}$。
- 优化的点：考虑最简单的分割方式：一个元素一段。这样的代价恰好是 n。

打扫卫生

- 还是先考虑暴力做法：设 f_i 表示到 a_i 时恰好切了一段时的最小代价，再设 $g_{j,i}$ 表示从 j 到 i 这一段里有多少个不同的数。
- $f_i = \min\{f_j + g_{j+1,i}\}$。
- 优化的点：考虑最简单的分割方式：一个元素一段。这样的代价恰好是 n。
- 所以，我们枚举的 j 和 i 之间的不同元素的个数不能超过 $\sqrt{n}$。

打扫卫生

打扫卫生

- 问题仍然存在：这一条性质不方便直接应用。

打扫卫生

- 问题仍然存在：这一条性质不方便直接应用。
- 挖掘“不同的数”的性质：只需要知道从 i 往前的第 $1, 2, ..., \sqrt{n}$ 个出现的新的数即可。

打扫卫生

- 问题仍然存在：这一条性质不方便直接应用。
- 挖掘“不同的数”的性质：只需要知道从 i 往前的第 $1, 2, ..., \sqrt{n}$ 个出现的新的数即可。
- 记录 $b_{i,j}$ 表示从 i 往前的第 j 个出现的数的位置。

打扫卫生

- 问题仍然存在：这一条性质不方便直接应用。
- 挖掘“不同的数”的性质：只需要知道从 i 往前的第 $1, 2, ..., \sqrt{n}$ 个出现的新的数即可。
- 记录 $b_{i,j}$ 表示从 i 往前的第 j 个出现的数的位置。
- 显然，$b_{i,1} = i$。

打扫卫生

- 问题仍然存在：这一条性质不方便直接应用。
- 挖掘“不同的数”的性质：只需要知道从 i 往前的第 $1, 2, ..., \sqrt{n}$ 个出现的新的数即可。
- 记录 $b_{i,j}$ 表示从 i 往前的第 j 个出现的数的位置。
- 显然，$b_{i,1} = i$。
- 接着看 $b_{i,2}$，它有很大可能是 $i - 1$，也就是 $b_{i-1,1}$。

打扫卫生

打扫卫生

- 出问题的情况：a_{i-1} 和 a_i 一样。

打扫卫生

- 出问题的情况：a_{i-1} 和 a_i 一样。
- 类似地，我们可以把 b 从 $i-1$ 迁移到 i，注意判断 a_i 在 $b_{i-1,*}$ 中的出现情况即可。

打扫卫生

- 出问题的情况：a_{i-1} 和 a_i 一样。
- 类似地，我们可以把 b 从 $i-1$ 迁移到 i，注意判断 a_i 在 $b_{i-1,*}$ 中的出现情况即可。
- 一个很显然的性质：f_i 是单调递增的。

打扫卫生

- 出问题的情况：a_{i-1} 和 a_i 一样。
- 类似地，我们可以把 b 从 $i-1$ 迁移到 i，注意判断 a_i 在 $b_{i-1,*}$ 中的出现情况即可。
- 一个很显然的性质：f_i 是单调递增的。
- 所以选取所有 $b_{i,*}+1$ 作为转移点最优。

打扫卫生

- 出问题的情况：a_{i-1} 和 a_i 一样。
- 类似地，我们可以把 b 从 $i-1$ 迁移到 i，注意判断 a_i 在 $b_{i-1,*}$ 中的出现情况即可。
- 一个很显然的性质：f_i 是单调递增的。
- 所以选取所有 $b_{i,*}+1$ 作为转移点最优。
- 复杂度 $O(n\sqrt{n})$。

打扫卫生

打扫卫生

- 出问题的情况：a_{i-1} 和 a_i 一样。

打扫卫生

- 出问题的情况：a_{i-1} 和 a_i 一样。
- 类似地，我们可以把 b 从 $i-1$ 迁移到 i，注意判断 a_i 在 $b_{i-1,*}$ 中的出现情况即可。

打扫卫生

- 出问题的情况：a_{i-1} 和 a_i 一样。
- 类似地，我们可以把 b 从 $i-1$ 迁移到 i，注意判断 a_i 在 $b_{i-1,*}$ 中的出现情况即可。
- 一个很显然的性质：f_i 是单调递增的。

打扫卫生

- 出问题的情况：a_{i-1} 和 a_i 一样。
- 类似地，我们可以把 b 从 $i-1$ 迁移到 i，注意判断 a_i 在 $b_{i-1,*}$ 中的出现情况即可。
- 一个很显然的性质：f_i 是单调递增的。
- 所以选取所有 $b_{i,*}+1$ 作为转移点最优。

打扫卫生

- 出问题的情况：a_{i-1} 和 a_i 一样。
- 类似地，我们可以把 b 从 $i-1$ 迁移到 i，注意判断 a_i 在 $b_{i-1,*}$ 中的出现情况即可。
- 一个很显然的性质：f_i 是单调递增的。
- 所以选取所有 $b_{i,*}+1$ 作为转移点最优。
- 复杂度 $O(n\sqrt{n})$。

阶段小结

阶段小结

- 在上面的部分，我们所见到的题目都是一些很有自身特点的题目。它们容易列出方程，但需要我们进行优化。

阶段小结

- 在上面的部分，我们所见到的题目都是一些很有自身特点的题目。它们容易列出方程，但需要我们进行优化。
- 根据题目自身的性质，我们可以利用不同的手段把它们优化到能通过的程度。

阶段小结

- 在上面的部分，我们所见到的题目都是一些很有自身特点的题目。它们容易列出方程，但需要我们进行优化。
- 根据题目自身的性质，我们可以利用不同的手段把它们优化到能通过的程度。
- 这种动态规划题目的麻烦之处在于：它们没有常规的优化手段或者特定的解题套路，需要选手根据题目本身的特点去进行针对性的分析。

阶段小结

- 在上面的部分，我们所见到的题目都是一些很有自身特点的题目。它们容易列出方程，但需要我们进行优化。
- 根据题目自身的性质，我们可以利用不同的手段把它们优化到能通过的程度。
- 这种动态规划题目的麻烦之处在于：它们没有常规的优化手段或者特定的解题套路，需要选手根据题目本身的特点去进行针对性的分析。
- 这样的题目灵活程度很大，可以出最简单的签到题，也可以出成一道防 AK 题。

阶段小结

- 在上面的部分，我们所见到的题目都是一些很有自身特点的题目。它们容易列出方程，但需要我们进行优化。
- 根据题目自身的性质，我们可以利用不同的手段把它们优化到能通过的程度。
- 这种动态规划题目的麻烦之处在于：它们没有常规的优化手段或者特定的解题套路，需要选手根据题目本身的特点去进行针对性的分析。
- 这样的题目灵活程度很大，可以出最简单的签到题，也可以出成一道防 AK 题。
- 一般而言，选手需要大量接触各种不同的动态规划题后，才能比较好地处理这样的问题。

最长公共子序列 2

最长公共子序列 2

- 给定两个序列 $a_1, a_2, ..., a_n$，及 $b_1, b_2, ..., b_m$。

最长公共子序列 2

- 给定两个序列 $a_1, a_2, ..., a_n$，及 $b_1, b_2, ..., b_m$。
- 你需要选出一个序列片段，使得这个这个片段在两个序列中都有出现过（不需要连续）。

最长公共子序列 2

- 给定两个序列 $a_1, a_2, ..., a_n$，及 $b_1, b_2, ..., b_m$。
- 你需要选出一个序列片段，使得这个这个片段在两个序列中都有出现过（不需要连续）。
- 在本题中，比较特殊的地方在于：$n = m$，且两个序列都是一个 1 到 n 的排列。

最长公共子序列 2

- 给定两个序列 $a_1, a_2, ..., a_n$，及 $b_1, b_2, ..., b_m$。
- 你需要选出一个序列片段，使得这个这个片段在两个序列中都有出现过（不需要连续）。
- 在本题中，比较特殊的地方在于：$n = m$，且两个序列都是一个 1 到 n 的排列。
- $n \leq 100000$。

最长公共子序列 2

最长公共子序列 2

- 如果序列 a 长成 $1, 2, 3, ..., n$，那么我们所选取的序列在 b 应该是单调上升的。

最长公共子序列 2

- 如果序列 a 长成 $1, 2, 3, ..., n$，那么我们所选取的序列在 b 应该是单调上升的。
- 事实上我们并不关心每个元素分别是什么，所以可以把 a_1 就看成 1，a_2 看成 $2, \cdots\cdots$，a_n 看成 n。

最长公共子序列 2

- 如果序列 a 长成 $1, 2, 3, ..., n$，那么我们所选取的序列在 b 应该是单调上升的。
- 事实上我们并不关心每个元素分别是什么，所以可以把 a_1 就看成 1，a_2 看成 2，⋯⋯，a_n 看成 n。
- 对序列 b，做和 a 一样的变换。

最长公共子序列 2

- 如果序列 a 长成 $1, 2, 3, ..., n$，那么我们所选取的序列在 b 应该是单调上升的。
- 事实上我们并不关心每个元素分别是什么，所以可以把 a_1 就看成 1，a_2 看成 2，⋯⋯，a_n 看成 n。
- 对序列 b，做和 a 一样的变换。
- 例子：比如 a 和 b 原来分别为"2413" 和"1342"，变换后分别变成"1234" 和"3421"。

最长公共子序列 2

- 如果序列 a 长成 $1, 2, 3, ..., n$，那么我们所选取的序列在 b 应该是单调上升的。
- 事实上我们并不关心每个元素分别是什么，所以可以把 a_1 就看成 1，a_2 看成 2，⋯⋯，a_n 看成 n。
- 对序列 b，做和 a 一样的变换。
- 例子：比如 a 和 b 原来分别为"2413" 和"1342"，变换后分别变成"1234" 和"3421"。
- 问题转化为：求序列 b 的最长上升子序列。

最长公共子序列 2

最长公共子序列 2

- 设 f_i 表示到第 i 个元素为止（并且第 i 个元素入选）的最长上升子序列长度。

最长公共子序列 2

- 设 f_i 表示到第 i 个元素为止（并且第 i 个元素入选）的最长上升子序列长度。
- $f_i = \max\{f_j\} + 1$。要求 $a_j < a_i$。

最长公共子序列 2

- 设 f_i 表示到第 i 个元素为止（并且第 i 个元素入选）的最长上升子序列长度。
- $f_i = \max\{f_j\} + 1$。要求 $a_j < a_i$。
- 维护一个序列 c，满足 $c_p = f_q$，其中 $a_q = p$。

最长公共子序列 2

- 设 f_i 表示到第 i 个元素为止（并且第 i 个元素入选）的最长上升子序列长度。
- $f_i = \max\{f_j\} + 1$。要求 $a_j < a_i$。
- 维护一个序列 c，满足 $c_p = f_q$，其中 $a_q = p$。
- 转移时，所需要做的即为求 $c_1, c_2, ..., c_{a_i}$ 的最大值。

最长公共子序列 2

- 设 f_i 表示到第 i 个元素为止（并且第 i 个元素入选）的最长上升子序列长度。
- $f_i = \max\{f_j\} + 1$。要求 $a_j < a_i$。
- 维护一个序列 c，满足 $c_p = f_q$，其中 $a_q = p$。
- 转移时，所需要做的即为求 $c_1, c_2, ..., c_{a_i}$ 的最大值。
- 使用树状数组或线段树即可完成这一步。

最大矩形

最大矩形

- 给定 $n \times n$ 的一个 01 矩阵，求矩阵中的一个最大的矩形，使得这个矩形中的元素都是 0。

最大矩形

- 给定 $n \times n$ 的一个 01 矩阵，求矩阵中的一个最大的矩形，使得这个矩形中的元素都是 0。
- $n \leq 1000$。

最大矩形

最大矩形

- 棘手的地方：最大正方形一题的做法完全不能化用到这一题上。

最大矩形

- 棘手的地方：最大正方形一题的做法完全不能化用到这一题上。
- 变换思路：枚举矩形的下边界。

最大矩形

- 棘手的地方：最大正方形一题的做法完全不能化用到这一题上。
- 变换思路：枚举矩形的下边界。
- 从下边界开始往上数 0，直到碰到一个 1。则只有这些被数过的 0 是有用的。

最大矩形

- 棘手的地方：最大正方形一题的做法完全不能化用到这一题上。
- 变换思路：枚举矩形的下边界。
- 从下边界开始往上数 0，直到碰到一个 1。则只有这些被数过的 0 是有用的。
- 暴力数 0=TLE，可以用 DP 数 0。

最大矩形

最大矩形

- 一个规律：确定下边界后，最大的矩形一定用了某一列的所有 0。

最大矩形

- 一个规律：确定下边界后，最大的矩形一定用了某一列的所有 0。
- 证明显然（反证法）。

最大矩形

- 一个规律：确定下边界后，最大的矩形一定用了某一列的所有 0。
- 证明显然（反证法）。
- 规约问题：给定一个数列 $a_0, a_1, ..., a_n$，对每个元素求在它左边、右边的第一个比它大的元素在哪里。

最大矩形

- 一个规律：确定下边界后，最大的矩形一定用了某一列的所有 0。
- 证明显然（反证法）。
- 规约问题：给定一个数列 $a_0, a_1, ..., a_n$，对每个元素求在它左边、右边的第一个比它大的元素在哪里。
- 数据结构优化：线段树。

最大矩形

- 一个规律：确定下边界后，最大的矩形一定用了某一列的所有 0。
- 证明显然（反证法）。
- 规约问题：给定一个数列 $a_0, a_1, ..., a_n$，对每个元素求在它左边、右边的第一个比它大的元素在哪里。
- 数据结构优化：线段树。
- 使用更简易的数据结构：单调栈。

最大矩形

最大矩形

- 以求一个元素的左边比它大的第一个元素为例:

最大矩形

- 以求一个元素的左边比它大的第一个元素为例：
- 如果一个元素的左边有比它小的元素，那么那些比它小的元素都没用了。

最大矩形

- 以求一个元素的左边比它大的第一个元素为例:
- 如果一个元素的左边有比它小的元素，那么那些比它小的元素都没用了。
- 维护一个栈，从左到右添加元素。

最大矩形

- 以求一个元素的左边比它大的第一个元素为例：
- 如果一个元素的左边有比它小的元素，那么那些比它小的元素都没用了。
- 维护一个栈，从左到右添加元素。
- 每次新增一个元素，就先把栈中不大于它的都先弹出，然后再把它加到栈顶。

最大矩形

- 以求一个元素的左边比它大的第一个元素为例:
- 如果一个元素的左边有比它小的元素，那么那些比它小的元素都没用了。
- 维护一个栈，从左到右添加元素。
- 每次新增一个元素，就先把栈中不大于它的都先弹出，然后再把它加到栈顶。
- 这样栈中的元素永远是单调下降的，所以称为单调栈。

最大矩形

最大矩形

- 这时，这个新插元素的前一个元素就是在它左边第一个比它大的。

最大矩形

- 这时，这个新插元素的前一个元素就是在它左边第一个比它大的。
- 另一边做法类似。

最大矩形

- 这时，这个新插元素的前一个元素就是在它左边第一个比它大的。
- 另一边做法类似。
- 总时间复杂度 $O(n^2)$。

最大矩形

- 这时，这个新插元素的前一个元素就是在它左边第一个比它大的。
- 另一边做法类似。
- 总时间复杂度 $O(n^2)$。
- 这种枚举下边界后取尽量多的 0 的做法有一个专有名词：悬线法。

奶牛

奶牛

- 给定一个整数序列 $a_1, a_2, ..., a_n$，你需要从序列中选出若干个元素，使得没有连续的 k 个元素入选。在此基础上，你需要使这些元素的和尽量大。

奶牛

- 给定一个整数序列 $a_1, a_2, ..., a_n$，你需要从序列中选出若干个元素，使得没有连续的 k 个元素入选。在此基础上，你需要使这些元素的和尽量大。
- $n \leq 100000$。

奶牛

奶牛

- 直接选择元素并不好做，不妨把问题倒过来看：放弃若干个元素，使得相邻两个被放弃的元素的距离不超过 k。

奶牛

- 直接选择元素并不好做，不妨把问题倒过来看：放弃若干个元素，使得相邻两个被放弃的元素的距离不超过 k。
- 设 f_i 表示到第 i 个元素，并且第 i 个元素恰好被放弃的情况下，所放弃的元素的最小和是多少。

奶牛

- 直接选择元素并不好做，不妨把问题倒过来看：放弃若干个元素，使得相邻两个被放弃的元素的距离不超过 k。
- 设 f_i 表示到第 i 个元素，并且第 i 个元素恰好被放弃的情况下，所放弃的元素的最小和是多少。
- $f_i = a_i + \min\{f_j\}$，要求 j 和 i 相距不超过 k。

奶牛

- 直接选择元素并不好做，不妨把问题倒过来看：放弃若干个元素，使得相邻两个被放弃的元素的距离不超过 k。
- 设 f_i 表示到第 i 个元素，并且第 i 个元素恰好被放弃的情况下，所放弃的元素的最小和是多少。
- $f_i = a_i + \min\{f_j\}$，要求 j 和 i 相距不超过 k。
- 同样，可以直接使用线段树求区间最小值，也可以考虑使用更简易的数据结构：单调队列。

奶牛

奶牛

- 按照和上一题一样的思维模式来优化问题：如果有一个位置的 DP 值比在它左边的某个位置的 DP 值小（或是等于），那左边的那个位置就没用了。

奶牛

- 按照和上一题一样的思维模式来优化问题：如果有一个位置的 DP 值比在它左边的某个位置的 DP 值小（或是等于），那左边的那个位置就没用了。
- 设 f_i 表示到第 i 个元素，并且第 i 个元素恰好被放弃的情况下，所放弃的元素的最小和是多少。

奶牛

- 按照和上一题一样的思维模式来优化问题：如果有一个位置的 DP 值比在它左边的某个位置的 DP 值小（或是等于），那左边的那个位置就没用了。
- 设 f_i 表示到第 i 个元素，并且第 i 个元素恰好被放弃的情况下，所放弃的元素的最小和是多少。
- $f_i = a_i + \min\{f_j\}$，要求 j 和 i 相距不超过 k。

奶牛

- 按照和上一题一样的思维模式来优化问题：如果有一个位置的 DP 值比在它左边的某个位置的 DP 值小（或是等于），那左边的那个位置就没用了。
- 设 f_i 表示到第 i 个元素，并且第 i 个元素恰好被放弃的情况下，所放弃的元素的最小和是多少。
- $f_i = a_i + \min\{f_j\}$，要求 j 和 i 相距不超过 k。
- 如果不考虑方程中的距离限制，我们可以使用单调栈，每次插入一个元素之前将不超过它的部分都弹出栈，最后把新元素入栈。

奶牛

- 按照和上一题一样的思维模式来优化问题：如果有一个位置的 DP 值比在它左边的某个位置的 DP 值小（或是等于），那左边的那个位置就没用了。
- 设 f_i 表示到第 i 个元素，并且第 i 个元素恰好被放弃的情况下，所放弃的元素的最小和是多少。
- $f_i = a_i + \min\{f_j\}$，要求 j 和 i 相距不超过 k。
- 如果不考虑方程中的距离限制，我们可以使用单调栈，每次插入一个元素之前将不超过它的部分都弹出栈，最后把新元素入栈。
- 考虑距离限制：这表明栈底的一些元素是不能被选择的。

奶牛

奶牛

- 把栈底“打穿”，使得可以从栈底弹出那些距离太远的元素。

奶牛

- 把栈底“打穿”，使得可以从栈底弹出那些距离太远的元素。
- 这样，这个数据结构从栈变成了队列。这里面的元素仍然是单调的，所以称为单调队列。

奶牛

- 把栈底“打穿”，使得可以从栈底弹出那些距离太远的元素。
- 这样，这个数据结构从栈变成了队列。这里面的元素仍然是单调的，所以称为单调队列。
- 每个元素入队一次，出队一次，复杂度是 $O(n)$ 的。

序列分割

序列分割

- 给定一个正整数序列 $a_1, a_2, ..., a_n$，你需要把它分成若干段，使得每一段的和都不超过一个给定值 m。在此基础上，你需要让每一段的最大值的和尽量小。

序列分割

- 给定一个正整数序列 $a_1, a_2, ..., a_n$，你需要把它分成若干段，使得每一段的和都不超过一个给定值 m。在此基础上，你需要让每一段的最大值的和尽量小。
- $n \leq 100000$。

序列分割

序列分割

- 设 f_i 表示恰好切到第 i 个元素的最优值。则
  $f_i = \min\{f_j + w(j+1, i)\}$。其中，w(j+1,i) 表示 $j+1$ 到 i 的最大值。

序列分割

- 设 f_i 表示恰好切到第 i 个元素的最优值。则 $f_i = \min\{f_j + w(j+1, i)\}$。其中，w(j+1,i) 表示 $j+1$ 到 i 的最大值。
- 一个很显然的事情是，随着 i 的增大，f_i 是不降的。

序列分割

- 设 f_i 表示恰好切到第 i 个元素的最优值。则 $f_i = \min\{f_j + w(j+1, i)\}$。其中，w(j+1,i) 表示 $j+1$ 到 i 的最大值。
- 一个很显然的事情是，随着 i 的增大，f_i 是不降的。
- 所以，如果两个决策点到 i 的最大值一样，我们选取靠前的一个就好了。

序列分割

序列分割

- 把到 i 最大值相同的部分合并成一段，记录这一段的最大值，和这一段的决策值。

序列分割

- 把到 i 最大值相同的部分合并成一段，记录这一段的最大值，和这一段的决策值。
- 按决策值为关键字，把一段的信息扔到一个小根堆里。

序列分割

- 把到 i 最大值相同的部分合并成一段，记录这一段的最大值，和这一段的决策值。
- 按决策值为关键字，把一段的信息扔到一个小根堆里。
- 每次把小根堆的堆顶用来做决策点。如果发现这个决策点不符合 m 的限制，就把这个决策点弹出；更新完 f_i 后，有可能最后的几段的最大值会发生改变（受 a_i 的影响），做相应的更新即可。

序列分割

- 把到 i 最大值相同的部分合并成一段，记录这一段的最大值，和这一段的决策值。
- 按决策值为关键字，把一段的信息扔到一个小根堆里。
- 每次把小根堆的堆顶用来做决策点。如果发现这个决策点不符合 m 的限制，就把这个决策点弹出；更新完 f_i 后，有可能最后的几段的最大值会发生改变（受 a_i 的影响），做相应的更新即可。
- 需要注意的是，每个点除了要拿当前的堆顶来决策，还需要拿在 m 限制下最靠左的点也进行一次决策。

玩具装箱

玩具装箱

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。你需要把序列切成若干段。每一段 $[i, j]$ 产生的代价是 $(a_i + a_{i+1} + ... + a_j + j - i - L)^2$。你需要最小化代价总和。

玩具装箱

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。你需要把序列切成若干段。每一段 $[i, j]$ 产生的代价是 $(a_i + a_{i+1} + ... + a_j + j - i - L)^2$。你需要最小化代价总和。
- $n \leq 100000$。

玩具装箱

玩具装箱

- 设 $s_i = a_1 + a_2 + ... + a_i$，则代价式可以化为 $(s_j - s_{i-1} + j - i - L)^2$。

玩具装箱

- 设 $s_i = a_1 + a_2 + ... + a_i$，则代价式可以化为 $(s_j - s_{i-1} + j - i - L)^2$。
- 设 $s_j + j - L = b_j$，$s_{i-1} + i = c_i$，则代价式可以进一步化为 $(b_j - c_i)^2$。

玩具装箱

- 设 $s_i = a_1 + a_2 + ... + a_i$，则代价式可以化为 $(s_j - s_{i-1} + j - i - L)^2$。
- 设 $s_j + j - L = b_j$，$s_{i-1} + i = c_i$，则代价式可以进一步化为 $(b_j - c_i)^2$。
- 设 f_i 表示切到第 i 个元素的最小代价和，则 $f_i = \min\{f_j + (b_i - c_j)^2\}$。

玩具装箱

- 设 $s_i = a_1 + a_2 + ... + a_i$，则代价式可以化为 $(s_j - s_{i-1} + j - i - L)^2$。
- 设 $s_j + j - L = b_j$，$s_{i-1} + i = c_i$，则代价式可以进一步化为 $(b_j - c_i)^2$。
- 设 f_i 表示切到第 i 个元素的最小代价和，则 $f_i = \min\{f_j + (b_i - c_j)^2\}$。
- 把转移式的平方项拆开，则 $f_i = b_i^2 + \min\{f_j + c_j^2 - 2b_i \times c_j\}$。

玩具装箱

玩具装箱

- 看上去，这个方程已经不能再简化了，所以我们需要一些特殊的技巧来继续优化。

玩具装箱

- 看上去，这个方程已经不能再简化了，所以我们需要一些特殊的技巧来继续优化。
- 考虑两个决策点 j 和 k，不妨设 $j > k$。如果用 j 转移比 k 优，则有：$f_j + c_j^2 - 2b_i \times c_j < f_k + c_k^2 - 2b_i \times c_k$。

玩具装箱

- 看上去，这个方程已经不能再简化了，所以我们需要一些特殊的技巧来继续优化。
- 考虑两个决策点 j 和 k，不妨设 $j > k$。如果用 j 转移比 k 优，则有：$f_j + c_j^2 - 2b_i \times c_j < f_k + c_k^2 - 2b_i \times c_k$。
- 再设 $f_j + c_j^2 = d_j$，则上式可以写成 $d_j - d_k < 2b_i \times (c_j - c_k)$。

玩具装箱

- 看上去，这个方程已经不能再简化了，所以我们需要一些特殊的技巧来继续优化。
- 考虑两个决策点 j 和 k，不妨设 $j > k$。如果用 j 转移比 k 优，则有：$f_j + c_j^2 - 2b_i \times c_j < f_k + c_k^2 - 2b_i \times c_k$。
- 再设 $f_j + c_j^2 = d_j$，则上式可以写成 $d_j - d_k < 2b_i \times (c_j - c_k)$。
- 注意到 $c_j > c_k$，所以我们可以把 $(c_j - c_k)$ 直接除过去，变成 $2b_i > \frac{d_j - d_k}{c_j - c_k}$。

玩具装箱

玩具装箱

- 不等式右边的部分，可以看成平面上两个点 (c_j, d_j) 和 (c_k, d_k) 的斜率。

玩具装箱

- 不等式右边的部分，可以看成平面上两个点 (c_j, d_j) 和 (c_k, d_k) 的斜率。
- 我们可以按顺序把点 (c_j, d_j) 加到平面里，并维护一个下凸壳。

玩具装箱

- 不等式右边的部分，可以看成平面上两个点 (c_j, d_j) 和 (c_k, d_k) 的斜率。
- 我们可以按顺序把点 (c_j, d_j) 加到平面里，并维护一个下凸壳。
- 一个性质：不在下凸壳上的点不可能成为决策点。

玩具装箱

- 不等式右边的部分，可以看成平面上两个点 (c_j, d_j) 和 (c_k, d_k) 的斜率。
- 我们可以按顺序把点 (c_j, d_j) 加到平面里，并维护一个下凸壳。
- 一个性质：不在下凸壳上的点不可能成为决策点。
- 证明？利用之前写出的斜率型不等式。

玩具装箱

玩具装箱

- 在凸壳上挑出 i 的最优决策点：利用不等式在凸壳上依次比较。

玩具装箱

- 在凸壳上挑出 i 的最优决策点：利用不等式在凸壳上依次比较。
- b_i 递增：一个决策点被比下去后，就不可能再成为决策点。

玩具装箱

- 在凸壳上挑出 i 的最优决策点：利用不等式在凸壳上依次比较。
- b_i 递增：一个决策点被比下去后，就不可能再成为决策点。
- 写法与单调队列类似（事实上就是一个特殊的单调队列）。

玩具装箱

- 在凸壳上挑出 i 的最优决策点：利用不等式在凸壳上依次比较。
- b_i 递增：一个决策点被比下去后，就不可能再成为决策点。
- 写法与单调队列类似（事实上就是一个特殊的单调队列）。
- 复杂度 $o(n)$。

土地购买

土地购买

- 农夫 John 准备扩大他的农场，他正在考虑 n 块长方形的土地。每块土地的长 a_i 宽 b_i 都不超过 1000000。

土地购买

- 农夫 John 准备扩大他的农场，他正在考虑 n 块长方形的土地。每块土地的长 a_i 宽 b_i 都不超过 1000000。
- 每块土地的价格是它的面积，但 John 可以同时购买多块土地。这些土地的价格是它们最大的长乘以它们最大的宽，但是土地的长宽不能交换。如果购买买一块 3×5 的地和一块 5×3 的地, 则他需要付 $5 \times 5 = 25$。

土地购买

- 农夫 John 准备扩大他的农场，他正在考虑 n 块长方形的土地。每块土地的长 a_i 宽 b_i 都不超过 1000000。
- 每块土地的价格是它的面积，但 John 可以同时购买多块土地。这些土地的价格是它们最大的长乘以它们最大的宽，但是土地的长宽不能交换。如果购买买一块 3×5 的地和一块 5×3 的地, 则他需要付 $5 \times 5 = 25$。
- 你需要计算：农夫 John 把所有土地都买下的最小代价。

土地购买

- 农夫 John 准备扩大他的农场，他正在考虑 n 块长方形的土地。每块土地的长 a_i 宽 b_i 都不超过 1000000。
- 每块土地的价格是它的面积，但 John 可以同时购买多块土地。这些土地的价格是它们最大的长乘以它们最大的宽，但是土地的长宽不能交换。如果购买买一块 3×5 的地和一块 5×3 的地, 则他需要付 $5 \times 5 = 25$。
- 你需要计算：农夫 John 把所有土地都买下的最小代价。
- $n \leq 100000$。

土地购买

土地购买

- 首先，如果一块土地的长和宽都比另外的某块土地小，那么我们可以直接不考虑它（思考：如何判断?）。

土地购买

- 首先，如果一块土地的长和宽都比另外的某块土地小，那么我们可以直接不考虑它（思考：如何判断?）。
- 去掉这些无用的土地后，考虑把土地按长从大到小排序。则排序后的宽一定是从小到大的。

土地购买

- 首先，如果一块土地的长和宽都比另外的某块土地小，那么我们可以直接不考虑它（思考：如何判断?）。
- 去掉这些无用的土地后，考虑把土地按长从大到小排序。则排序后的宽一定是从小到大的。
- 这时，一次的购买一定是选择连续的一段土地（否则调整后可以更优）。

土地购买

- 首先，如果一块土地的长和宽都比另外的某块土地小，那么我们可以直接不考虑它（思考：如何判断?）。
- 去掉这些无用的土地后，考虑把土地按长从大到小排序。则排序后的宽一定是从小到大的。
- 这时，一次的购买一定是选择连续的一段土地（否则调整后可以更优）。
- 设 f_i 表示购买到第 i 块土地时的最小代价。

土地购买

- 首先，如果一块土地的长和宽都比另外的某块土地小，那么我们可以直接不考虑它（思考：如何判断?）。
- 去掉这些无用的土地后，考虑把土地按长从大到小排序。则排序后的宽一定是从小到大的。
- 这时，一次的购买一定是选择连续的一段土地（否则调整后可以更优）。
- 设 f_i 表示购买到第 i 块土地时的最小代价。
- $f_i = \min\{f_{j-1} + a_j \times b_i\}$。

土地购买

土地购买

- 考虑 $j > k$ 时，选 j 进行转移比选 k 进行转移更优的条件:

土地购买

- 考虑 $j > k$ 时，选 j 进行转移比选 k 进行转移更优的条件：
- $f_{j-1} + a_j \times b_i \leq f_{k-1} + a_k \times b_i$。

土地购买

- 考虑 $j > k$ 时，选 j 进行转移比选 k 进行转移更优的条件：
- $f_{j-1} + a_j \times b_i \le f_{k-1} + a_k \times b_i$。
- 利用这个式子即可化出斜率优化的方程。

斜率优化小结

斜率优化小结

- 对于斜率优化题而言，最核心的式子大致形如 $s_i > \frac{y_j - y_k}{x_j - x_k}$。

斜率优化小结

- 对于斜率优化题而言，最核心的式子大致形如 $s_i > \frac{y_j - y_k}{x_j - x_k}$。
- x_i 单调/不单调：顺序维护凸壳/平衡树维护凸壳（或是 CDQ 分治）。

斜率优化小结

- 对于斜率优化题而言，最核心的式子大致形如 $s_i > \frac{y_j - y_k}{x_j - x_k}$。
- x_i 单调/不单调：顺序维护凸壳/平衡树维护凸壳（或是 CDQ 分治）。
- s_i 单调/不单调：在凸壳上单调维护/在凸壳上二分。

斜率优化小结

- 对于斜率优化题而言，最核心的式子大致形如 $s_i > \frac{y_j - y_k}{x_j - x_k}$。
- x_i 单调/不单调：顺序维护凸壳/平衡树维护凸壳（或是 CDQ 分治）。
- s_i 单调/不单调：在凸壳上单调维护/在凸壳上二分。
- 最容易出错的地方：正负号。

斜率优化小结

- 对于斜率优化题而言，最核心的式子大致形如 $s_i > \frac{y_j - y_k}{x_j - x_k}$。
- x_i 单调/不单调：顺序维护凸壳/平衡树维护凸壳（或是 CDQ 分治）。
- s_i 单调/不单调：在凸壳上单调维护/在凸壳上二分。
- 最容易出错的地方：正负号。
- 对动态规划而言（甚至是对于很多其它类型的题目），凸壳（凸性）是一种非常有用的用于优化的结构，斜率优化是其中最典型的一种。

Lightning Conductor

Lightning Conductor

- 给定一个序列 $a_1, a_2, ..., a_n$，对于每个 i，你需要找出 $a_j + \sqrt{|i - j|}$ 的最大值。

Lightning Conductor

- 给定一个序列 $a_1, a_2, ..., a_n$，对于每个 i，你需要找出 $a_j + \sqrt{|i-j|}$ 的最大值。
- $n \leq 100000$。

Lightning Conductor

Lightning Conductor

- 对于每一个 i，考虑对大于 i 和小于 i 两部分分别求最大值，然后再取一个最大的。

Lightning Conductor

- 对于每一个 i，考虑对大于 i 和小于 i 两部分分别求最大值，然后再取一个最大的。
- 如果对于 $k < j < i$，$a_j + \sqrt{i-j} > a_k + \sqrt{i-k}$，那么对于 $i+1, i+2, \ldots$，j 永远比 k 优秀。这是因为 $\sqrt{i-j} - \sqrt{i-k} > \sqrt{i+c-j} - \sqrt{i+c-k}$。

Lightning Conductor

- 对于每一个 i，考虑对大于 i 和小于 i 两部分分别求最大值，然后再取一个最大的。
- 如果对于 $k < j < i$，$a_j + \sqrt{i-j} > a_k + \sqrt{i-k}$，那么对于 $i+1, i+2, \ldots$，j 永远比 k 优秀。这是因为 $\sqrt{i-j} - \sqrt{i-k} > \sqrt{i+c-j} - \sqrt{i+c-k}$。
- 对于满足这样的条件的题目，我们可以使用决策单调性优化来优化转移。

Lightning Conductor

Lightning Conductor

- 优化方法：先假设对于所有的位置，都拿位置 1 做决策点。

Lightning Conductor

- 优化方法：先假设对于所有的位置，都拿位置 1 做决策点。
- 考虑加入一个 2。如果有某个位置，用 2 决策比 1 优，那么对于之后的部分，2 永远比 1 优。

Lightning Conductor

- 优化方法：先假设对于所有的位置，都拿位置 1 做决策点。
- 考虑加入一个 2。如果有某个位置，用 2 决策比 1 优，那么对于之后的部分，2 永远比 1 优。
- 所以，我们可以二分求出第一个 2 比 1 优的位置（需要特判 2 永远比 1 劣的情况）。

Lightning Conductor

- 优化方法：先假设对于所有的位置，都拿位置 1 做决策点。
- 考虑加入一个 2。如果有某个位置，用 2 决策比 1 优，那么对于之后的部分，2 永远比 1 优。
- 所以，我们可以二分求出第一个 2 比 1 优的位置（需要特判 2 永远比 1 劣的情况）。
- 然后，让 2 把本来全是 1 的转移分走一部分。

Lightning Conductor

Lightning Conductor

- 同样的道理，考虑处理完 $j-1$，并且已经维护出若干个线段，表示一些连续区间应该用哪个点做决策。

Lightning Conductor

- 同样的道理，考虑处理完 $j-1$，并且已经维护出若干个线段，表示一些连续区间应该用哪个点做决策。
- 现在 j 被加入，我们从最末尾的线段开始看起，依次判断 j 是不是可以完全代替这个线段原先的决策点。

Lightning Conductor

- 同样的道理，考虑处理完 $j-1$，并且已经维护出若干个线段，表示一些连续区间应该用哪个点做决策。
- 现在 j 被加入，我们从最末尾的线段开始看起，依次判断 j 是不是可以完全代替这个线段原先的决策点。
- 如果可以，就把这个线段的决策点进行修改；如果不行，这说明可能需要从这个线段的中间切开，前面的部分归原来的决策点，后面的部分归 j。二分求出这个分割点。

- 同样的道理，考虑处理完 $j-1$，并且已经维护出若干个线段，表示一些连续区间应该用哪个点做决策。
- 现在 j 被加入，我们从最末尾的线段开始看起，依次判断 j 是不是可以完全代替这个线段原先的决策点。
- 如果可以，就把这个线段的决策点进行修改；如果不行，这说明可能需要从这个线段的中间切开，前面的部分归原来的决策点，后面的部分归 j。二分求出这个分割点。
- 每个决策点至多产生一个线段，所以复杂度是 $O(n\log n)$（log 产生在二分）。

取数

取数

- 给定一个整数序列 $a_1, a_2, ..., a_n$，你需要从序列中选出恰好 m 个元素，使得相邻两个元素的距离大于等于 k。在此基础上，你需要使这些元素的和尽量大。

取数

- 给定一个整数序列 $a_1, a_2, ..., a_n$，你需要从序列中选出恰好 m 个元素，使得相邻两个元素的距离大于等于 k。在此基础上，你需要使这些元素的和尽量大。
- $n, m, k \leq 1000$。

取数

取数

- 设 $f_{i,j}$ 表示在前 i 个数中选择了恰好 j 个时，能获得的最大权值(第 i 个不一定需要入选)。

取数

- 设 $f_{i,j}$ 表示在前 i 个数中选择了恰好 j 个时，能获得的最大权值（第 i 个不一定需要入选）。
- 第 i 个不选：$f_{i,j} = f_{i-1,j}$。

取数

- 设 $f_{i,j}$ 表示在前 i 个数中选择了恰好 j 个时，能获得的最大权值（第 i 个不一定需要入选）。
- 第 i 个不选：$f_{i,j} = f_{i-1,j}$。
- 第 i 个入选：$f_{i,j} = f_{i-k,j-1} + a_i$。

取数

- 设 $f_{i,j}$ 表示在前 i 个数中选择了恰好 j 个时，能获得的最大权值（第 i 个不一定需要入选）。
- 第 i 个不选：$f_{i,j} = f_{i-1,j}$。
- 第 i 个入选：$f_{i,j} = f_{i-k,j-1} + a_i$。
- 两者取一个较大的进行转移。

取数

- 设 $f_{i,j}$ 表示在前 i 个数中选择了恰好 j 个时，能获得的最大权值（第 i 个不一定需要入选）。
- 第 i 个不选：$f_{i,j} = f_{i-1,j}$。
- 第 i 个入选：$f_{i,j} = f_{i-k,j-1} + a_i$。
- 两者取一个较大的进行转移。
- $n, m, k \leq 100000$?

取数

取数

- 如果仍然采用动态规划的思路，是没有优化的余地的。

取数

- 如果仍然采用动态规划的思路，是没有优化的余地的。
- 动态规划容易在转移部分进行优化，但难以优化状态数。

取数

- 如果仍然采用动态规划的思路，是没有优化的余地的。
- 动态规划容易在转移部分进行优化，但难以优化状态数。
- $n, m, k \le 100000$ 的做法存在，但和动态规划的思维走向完全不同。

取数

- 如果仍然采用动态规划的思路，是没有优化的余地的。
- 动态规划容易在转移部分进行优化，但难以优化状态数。
- $n, m, k \leq 100000$ 的做法存在，但和动态规划的思维走向完全不同。
- 具体做法（有时间才讲）：

取数

- 如果仍然采用动态规划的思路，是没有优化的余地的。
- 动态规划容易在转移部分进行优化，但难以优化状态数。
- $n, m, k \leq 100000$ 的做法存在，但和动态规划的思维走向完全不同。
- 具体做法（有时间才讲）：
- $m = 2$ 的特殊情况：可以使用堆优化的贪心。

取数

- 如果仍然采用动态规划的思路，是没有优化的余地的。
- 动态规划容易在转移部分进行优化，但难以优化状态数。
- $n, m, k \leq 100000$ 的做法存在，但和动态规划的思维走向完全不同。
- 具体做法（有时间才讲）：
- $m = 2$ 的特殊情况：可以使用堆优化的贪心。
- 一般情况：凸函数的性质。

Thanks for listening.