动态规划

王逸凡

2020.10.06

动态规划简述

动态规划简述

- 动态规划（包括一些算法类似的递推计数问题）在信息竞赛中经常出现，考察比例很高

动态规划简述

- 动态规划（包括一些算法类似的递推计数问题）在信息竞赛中经常出现，考察比例很高
- 但与数据结构、图论等模块不同，动态规划不具有特别专门的知识点，它更多地是一种设计子问题结构用于解决全局问题的思想

动态规划简述

- 动态规划（包括一些算法类似的递推计数问题）在信息竞赛中经常出现，考察比例很高
- 但与数据结构、图论等模块不同，动态规划不具有特别专门的知识点，它更多地是一种设计子问题结构用于解决全局问题的思想
- 因此，掌握动态规划的最好方法是尽可能多地解出各种动态规划的具体题目

动态规划简述

- 动态规划（包括一些算法类似的递推计数问题）在信息竞赛中经常出现，考察比例很高
- 但与数据结构、图论等模块不同，动态规划不具有特别专门的知识点，它更多地是一种设计子问题结构用于解决全局问题的思想
- 因此，掌握动态规划的最好方法是尽可能多地解出各种动态规划的具体题目
- 当然，有一些动态规划题目的解答有着可以总结的套路，我们会单独将其进行分类，如：区间 DP、背包 DP、状压 DP、斜率优化等，我们在之后的讲解中会有所侧重

乘积最大

乘积最大

- 给定长度为 n 的一个数字串。

乘积最大

- 给定长度为 n 的一个数字串。
- 你需要在串中间插入 k 个乘号，使得构成的表达式的值最大。

乘积最大

- 给定长度为 n 的一个数字串。
- 你需要在串中间插入 k 个乘号，使得构成的表达式的值最大。
- $n \leq 40$，$k \leq 6$。

乘积最大

乘积最大

- 一道非常经典的区间 DP 题。

乘积最大

- 一道非常经典的区间 DP 题。
- 设 $f_{i,j,k}$ 表示在 i 到 j 这个区间插入 k 个乘号所能达到的最大乘积。

乘积最大

- 一道非常经典的区间 DP 题。
- 设 $f_{i,j,k}$ 表示在 i 到 j 这个区间插入 k 个乘号所能达到的最大乘积。
- 枚举 l，表示 i 到 j 这个区间中的第一个乘号放在第 l 个数后面。

乘积最大

- 一道非常经典的区间 DP 题。
- 设 $f_{i,j,k}$ 表示在 i 到 j 这个区间插入 k 个乘号所能达到的最大乘积。
- 枚举 l，表示 i 到 j 这个区间中的第一个乘号放在第 l 个数后面。
- 则 $f_{i,j,k} = \max\{g_{i,l} \times f_{l+1,j,k-1}\}$。其中 $g_{i,l}$ 表示 i 到 l 这个区间构成的数。

中间和

中间和

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。 $n \leq 200$。

中间和

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。$n \leq 200$。
- 不改变序列中每个元素在序列中的位置，把它们相加，并用括号记每次加法所得的和，称为中间和。

中间和

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。$n \leq 200$。
- 不改变序列中每个元素在序列中的位置，把它们相加，并用括号记每次加法所得的和，称为中间和。
- 例如给出序列是 4，1，2，3。

中间和

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。$n \leq 200$。
- 不改变序列中每个元素在序列中的位置，把它们相加，并用括号记每次加法所得的和，称为中间和。
- 例如给出序列是 4，1，2，3。
- 第一种添括号方法：$((4+1)+(2+3)) = ((5)+(5)) = (10)$

中间和

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。$n \leq 200$。
- 不改变序列中每个元素在序列中的位置，把它们相加，并用括号记每次加法所得的和，称为中间和。
- 例如给出序列是 4，1，2，3。
- 第一种添括号方法：$((4+1)+(2+3)) = ((5)+(5)) = (10)$
- 有三个中间和是 5，5，10，它们之和为:$5+5+10=20$

中间和

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。$n \leq 200$。
- 不改变序列中每个元素在序列中的位置，把它们相加，并用括号记每次加法所得的和，称为中间和。
- 例如给出序列是 4，1，2，3。
- 第一种添括号方法：$((4+1)+(2+3)) = ((5)+(5)) = (10)$
- 有三个中间和是 5，5，10，它们之和为:$5+5+10=20$
- 第二种添括号方法:
  $(4+((1+2)+3)) = (4+((3)+3)) = (4+(6)) = (10)$

中间和

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。$n \leq 200$。
- 不改变序列中每个元素在序列中的位置，把它们相加，并用括号记每次加法所得的和，称为中间和。
- 例如给出序列是 4，1，2，3。
- 第一种添括号方法：$((4+1)+(2+3)) = ((5)+(5)) = (10)$
- 有三个中间和是 5，5，10，它们之和为:$5+5+10=20$
- 第二种添括号方法:
  $(4+((1+2)+3)) = (4+((3)+3)) = (4+(6)) = (10)$
- 中间和是 3，6，10，它们之和为 19。

中间和

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。$n \leq 200$。
- 不改变序列中每个元素在序列中的位置，把它们相加，并用括号记每次加法所得的和，称为中间和。
- 例如给出序列是 4，1，2，3。
- 第一种添括号方法：$((4+1)+(2+3)) = ((5)+(5)) = (10)$
- 有三个中间和是 5，5，10，它们之和为:$5+5+10=20$
- 第二种添括号方法:
  $(4+((1+2)+3)) = (4+((3)+3)) = (4+(6)) = (10)$
- 中间和是 3，6，10，它们之和为 19。
- 你需要求出最大的中间和是多少。

中间和

中间和

- 这道题的做法就是直接枚举切割区间的点。

中间和

- 这道题的做法就是直接枚举切割区间的点。
- 设 $f_{l,r}$ 表示考虑左端为 l，右端为 r 的区间得到的最优解

中间和

- 这道题的做法就是直接枚举切割区间的点。
- 设 $f_{l,r}$ 表示考虑左端为 l，右端为 r 的区间得到的最优解
- 状态转移为 $f_{l,r} = \max\{f_{l,i} + f_{i+1,r}\} + sum_{l,r}$。其中 $sum_{l,r}$ 表示 l 到 r 的部分和。

中间和

- 这道题的做法就是直接枚举切割区间的点。
- 设 $f_{l,r}$ 表示考虑左端为 l，右端为 r 的区间得到的最优解
- 状态转移为 $f_{l,r} = \max\{f_{l,i} + f_{i+1,r}\} + sum_{l,r}$。其中 $sum_{l,r}$ 表示 l 到 r 的部分和。
- 复杂度 $O(n^3)$。

物品分配

物品分配

- 给出 n 个物品，每个物品有一个重量（正整数），它们的重量和为 m。你可以从 n 个物品中任意选取若干个物品。你需要求出：你得到的物品的重量和有多少种不同的可能。

物品分配

- 给出 n 个物品，每个物品有一个重量（正整数），它们的重量和为 m。你可以从 n 个物品中任意选取若干个物品。你需要求出：你得到的物品的重量和有多少种不同的可能。
- $n \leq 100000$，$m \leq 1000000$。

物品分配

物品分配

- 背包 DP 问题。

物品分配

- 背包 DP 问题。
- 考虑用如下手法处理具有相同重量的物品：如果某个重量的物品有 k 个，则可以把 k 拆成 $1+2+...+2^p+r$，这样拆分前后物品的选取情况是等价的。

物品分配

- 背包 DP 问题。
- 考虑用如下手法处理具有相同重量的物品：如果某个重量的物品有 k 个，则可以把 k 拆成 $1+2+...+2^p+r$，这样拆分前后物品的选取情况是等价的。
- 从小到大进行这样的拆分，则可以保证每种重量的物品个数不超过 3。

物品分配

- 背包 DP 问题。
- 考虑用如下手法处理具有相同重量的物品：如果某个重量的物品有 k 个，则可以把 k 拆成 $1+2+...+2^p+r$，这样拆分前后物品的选取情况是等价的。
- 从小到大进行这样的拆分，则可以保证每种重量的物品个数不超过 3。
- 又由于物品的重量和为 m，所以拆分后的物品个数是 $O(\sqrt{m})$ 级别的。

物品分配

- 背包 DP 问题。
- 考虑用如下手法处理具有相同重量的物品：如果某个重量的物品有 k 个，则可以把 k 拆成 $1+2+...+2^p+r$，这样拆分前后物品的选取情况是等价的。
- 从小到大进行这样的拆分，则可以保证每种重量的物品个数不超过 3。
- 又由于物品的重量和为 m，所以拆分后的物品个数是 $O(\sqrt{m})$ 级别的。
- 设 f 是表示最终状态的数组（即 $f[i]$ 表示重量 i 是否可以取到），枚举每个物品，若物品重量为 c，则可以用 $f = for(f << c)$ 来更新 f。

物品分配

- 背包 DP 问题。
- 考虑用如下手法处理具有相同重量的物品：如果某个重量的物品有 k 个，则可以把 k 拆成 $1+2+...+2^p+r$，这样拆分前后物品的选取情况是等价的。
- 从小到大进行这样的拆分，则可以保证每种重量的物品个数不超过 3。
- 又由于物品的重量和为 m，所以拆分后的物品个数是 $O(\sqrt{m})$ 级别的。
- 设 f 是表示最终状态的数组（即 $f[i]$ 表示重量 i 是否可以取到），枚举每个物品，若物品重量为 c，则可以用 $f = for(f << c)$ 来更新 f。
- 复杂度 $O(m\sqrt{m} \times w)$，大致有 $w = \frac{1}{32}$。

修改

修改

- 给出 n 个物品，每个物品有一个重量 v_i（正整数），总和为 m。你可以从 n 个物品选择一件并修改它的重量。你需要求出：修改后，得到的物品的重量和有多少种不同的可能。

修改

- 给出 n 个物品，每个物品有一个重量 v_i（正整数），总和为 m。你可以从 n 个物品选择一件并修改它的重量。你需要求出：修改后，得到的物品的重量和有多少种不同的可能。
- 可能存在多种方法使得物品修改后不同的重量种类数相同且最大，因此你需要输出具体的修改方案：存在多种方案时，被选的物品重量 a 最小；若还有多种方案，你需要使 a 被修改为的重量 b 最小。输出 a 和 b 即可

修改

- 给出 n 个物品，每个物品有一个重量 v_i（正整数），总和为 m。你可以从 n 个物品选择一件并修改它的重量。你需要求出：修改后，得到的物品的重量和有多少种不同的可能。
- 可能存在多种方法使得物品修改后不同的重量种类数相同且最大，因此你需要输出具体的修改方案：存在多种方案时，被选的物品重量 a 最小；若还有多种方案，你需要使 a 被修改为的重量 b 最小。输出 a 和 b 即可
- $n \leq 100$，$v_i \leq 7000$。

修改

修改

- 如果把一个物品的重量修改为无限大，则修改后的种类数为除了这个物品外其它物品的种类数 $\times 2$。因此，a 应该按下述方法选择：把 a 去掉后，剩余物品能构成的种类数最多，在此基础上 a 最小

修改

- 如果把一个物品的重量修改为无限大，则修改后的种类数为除了这个物品外其它物品的种类数 $\times 2$。因此，a 应该按下述方法选择：把 a 去掉后，剩余物品能构成的种类数最多，在此基础上 a 最小
- 目标：求出去掉一个物品后剩余物品能组成的重量种类数

修改

- 如果把一个物品的重量修改为无限大，则修改后的种类数为除了这个物品外其它物品的种类数 $\times 2$。因此，a 应该按下述方法选择：把 a 去掉后，剩余物品能构成的种类数最多，在此基础上 a 最小
- 目标：求出去掉一个物品后剩余物品能组成的重量种类数
- 设 f_i 表示选取物品重量和为 i 的方案数。则：$f_{j,i} = f_{j-1,i} + f_{j-1,i-v_j}$，DP 求解

修改

- 如果把一个物品的重量修改为无限大，则修改后的种类数为除了这个物品外其它物品的种类数 $\times 2$。因此，a 应该按下述方法选择：把 a 去掉后，剩余物品能构成的种类数最多，在此基础上 a 最小
- 目标：求出去掉一个物品后剩余物品能组成的重量种类数
- 设 f_i 表示选取物品重量和为 i 的方案数。则：$f_{j,i} = f_{j-1,i} + f_{j-1,i-v_j}$，DP 求解
- 设 g_j 去掉一个重量为 c 的物品后的方案数，则 $g_j = f_j - g_{j-c}$

修改

- 如果把一个物品的重量修改为无限大，则修改后的种类数为除了这个物品外其它物品的种类数 $\times 2$。因此，a 应该按下述方法选择：把 a 去掉后，剩余物品能构成的种类数最多，在此基础上 a 最小
- 目标：求出去掉一个物品后剩余物品能组成的重量种类数
- 设 f_i 表示选取物品重量和为 i 的方案数。则：$f_{j,i} = f_{j-1,i} + f_{j-1,i-v_j}$，DP 求解
- 设 g_j 去掉一个重量为 c 的物品后的方案数，则 $g_j = f_j - g_{j-c}$
- 求一次 g_j：$O(m)$；求 a：$O(nm)$

修改

- 如果把一个物品的重量修改为无限大，则修改后的种类数为除了这个物品外其它物品的种类数 $\times 2$。因此，a 应该按下述方法选择：把 a 去掉后，剩余物品能构成的种类数最多，在此基础上 a 最小
- 目标：求出去掉一个物品后剩余物品能组成的重量种类数
- 设 f_i 表示选取物品重量和为 i 的方案数。则：$f_{j,i} = f_{j-1,i} + f_{j-1,i-v_j}$，DP 求解
- 设 g_j 去掉一个重量为 c 的物品后的方案数，则 $g_j = f_j - g_{j-c}$
- 求一次 g_j：$O(m)$；求 a：$O(nm)$
- 问：f_i 超过 long long 怎么办?

修改

修改

- 下面考虑如何求 b。如果 b 选择后能让重量种类数翻倍，则 b 不能是除 a 外的物品可以表示出的两种重量的差值

修改

- 下面考虑如何求 b。如果 b 选择后能让重量种类数翻倍，则 b 不能是除 a 外的物品可以表示出的两种重量的差值
- 相当于把除 a 外的物品复制一份，重量取为相反数，由这 $2n-2$ 个物品能组合出的重量都不能选

修改

- 下面考虑如何求 b。如果 b 选择后能让重量种类数翻倍，则 b 不能是除 a 外的物品可以表示出的两种重量的差值
- 相当于把除 a 外的物品复制一份，重量取为相反数，由这 $2n-2$ 个物品能组合出的重量都不能选
- 做一遍不加优化的物品分配即可

补兵

补兵

- 有 n 个小怪，每个小怪的血量分别为 $a_1, a_2, ..., a_n$。你和另一名玩家正在对小怪进行攻击。每次你可以挑选一个小兵造成 1 点伤害，接着另一名玩家会对所有小兵都造成 1 点 AOE 伤害。

补兵

- 有 n 个小怪，每个小怪的血量分别为 $a_1, a_2, ..., a_n$。你和另一名玩家正在对小怪进行攻击。每次你可以挑选一个小兵造成 1 点伤害，接着另一名玩家会对所有小兵都造成 1 点 AOE 伤害。
- 求出你最多可以补到多少兵（有多少兵在你攻击后血量由 1 变为 0）。

补兵

- 有 n 个小怪，每个小怪的血量分别为 $a_1, a_2, ..., a_n$。你和另一名玩家正在对小怪进行攻击。每次你可以挑选一个小兵造成 1 点伤害，接着另一名玩家会对所有小兵都造成 1 点 AOE 伤害。
- 求出你最多可以补到多少兵（有多少兵在你攻击后血量由 1 变为 0）。
- $n, a_i \leq 1000$。

补兵

补兵

- 考虑一个最特殊的例子：小兵的血量分布恰好长成 $1, 2, 3, \ldots$。如果是这样，我们显然可以补到每一刀。

补兵

- 考虑一个最特殊的例子：小兵的血量分布恰好长成 $1, 2, 3, \ldots$。如果是这样，我们显然可以补到每一刀。
- 现在的状况是，我们面对的局面并不是这样，所以我们应该先攻击一些小兵，使得小兵的血量分布变成上面描述的情况（有一个专门的术语就描述了这种操作，叫做“垫刀”）。

补兵

- 考虑一个最特殊的例子：小兵的血量分布恰好长成 $1, 2, 3, \ldots$。如果是这样，我们显然可以补到每一刀。
- 现在的状况是，我们面对的局面并不是这样，所以我们应该先攻击一些小兵，使得小兵的血量分布变成上面描述的情况（有一个专门的术语就描述了这种操作，叫做“垫刀”）。
- 还可以明确的情况是，如果有两个小兵的血量一直是一样的，那么这两个小兵我们至多只能补到一个。

补兵

- 考虑一个最特殊的例子：小兵的血量分布恰好长成 $1, 2, 3, \ldots$。如果是这样，我们显然可以补到每一刀。
- 现在的状况是，我们面对的局面并不是这样，所以我们应该先攻击一些小兵，使得小兵的血量分布变成上面描述的情况（有一个专门的术语就描述了这种操作，叫做“垫刀”）。
- 还可以明确的情况是，如果有两个小兵的血量一直是一样的，那么这两个小兵我们至多只能补到一个。
- 所以，如果我们发现在血量 i 没有小兵，应该从血量高于 i 的小兵中，挑一个血量和别的小兵相同的、血量尽量少的来补齐。

补兵

补兵

- 这样，我们可以当作这个小兵的血量就是 i，只是在选择它前需要垫几刀。

补兵

- 这样，我们可以当作这个小兵的血量就是 i，只是在选择它前需要垫几刀。
- 具体到 DP，设 $f[i][j]$ 表示考虑到血量为 i 的小兵，在操作的过程中省下了 j 刀的情况下所能补到的最大兵数。

补兵

- 这样，我们可以当作这个小兵的血量就是 i，只是在选择它前需要垫几刀。
- 具体到 DP，设 $f[i][j]$ 表示考虑到血量为 i 的小兵，在操作的过程中省下了 j 刀的情况下所能补到的最大兵数。
- 如果第 i 个小兵不补，则 $f[i][j] = f[i-1][j-1]$；如果要补，则 $f[i][j] = f[i-1][j+c[i]+1]$ c[i] 表示要补血量被当作的小兵之前要垫多少刀。两者取一个最大值即可。

TSP 问题

TSP 问题

- 一个 n 个点的带权的有向图，求一条路径，使得这条路经过每个点恰好一次，并且路径上边的权值和最小。

TSP 问题

- 一个 n 个点的带权的有向图，求一条路径，使得这条路经过每个点恰好一次，并且路径上边的权值和最小。
- $n \leq 18$。

TSP 问题

TSP 问题

- 状压 DP。

TSP 问题

- 状压 DP。
- 考虑一种最直接的思路：在设动态规划状态时把每个点是否经过都记录下来。

TSP 问题

- 状压 DP。
- 考虑一种最直接的思路：在设动态规划状态时把每个点是否经过都记录下来。
- 这需要我们开一个 n 维数组，每一维都是 0 或 1（还没经过／经过了），难以承受。

TSP 问题

- 状压 DP。
- 考虑一种最直接的思路：在设动态规划状态时把每个点是否经过都记录下来。
- 这需要我们开一个 n 维数组，每一维都是 0 或 1（还没经过／经过了），难以承受。
- 把这 n 维压缩成一维，即写成一个二进制。二进制的每一位是 0/1 表示在原来的 n 维数组的对应维是 0/1。

TSP 问题

- 状压 DP。
- 考虑一种最直接的思路：在设动态规划状态时把每个点是否经过都记录下来。
- 这需要我们开一个 n 维数组，每一维都是 0 或 1（还没经过／经过了），难以承受。
- 把这 n 维压缩成一维，即写成一个二进制。二进制的每一位是 0/1 表示在原来的 n 维数组的对应维是 0/1。
- 当然，这样设状态还不够。除了需要知道这 n 个点哪些点经过了哪些点没经过，我们还需要知道目前路径的最末尾是哪个点。

TSP 问题

TSP 问题

- 设 $f_{u,S}$ 表示把 n 个 0/1 压缩以后的状态是 S，并且路径末尾的点是点 u 时，路径的最短长度。

TSP 问题

- 设 $f_{u,S}$ 表示把 n 个 0/1 压缩以后的状态是 S，并且路径末尾的点是点 u 时，路径的最短长度。
- 枚举 u 的所有出边，如果是还没有经过的点，就可以转移了。

TSP 问题

- 设 $f_{u,S}$ 表示把 n 个 0/1 压缩以后的状态是 S，并且路径末尾的点是点 u 时，路径的最短长度。
- 枚举 u 的所有出边，如果是还没有经过的点，就可以转移了。
- 最后的答案在 $f_{*,2^n-1}$ 上。

k 国王

k 国王

- 在一个 $n \times n$ 的网格图中放 k 个国王，一个国王的攻击范围是周围的八个格子。问让国王互不攻击的方案数。

k 国王

- 在一个 $n \times n$ 的网格图中放 k 个国王，一个国王的攻击范围是周围的八个格子。问让国王互不攻击的方案数。
- $n \leq 9$。

k 国王

k 国王

- 设 $f_{i,j,S}$ 表示考虑到第 i 行，放了 j 个国王，下一行不能被放国王的位置状态为 S 的方案数。

k 国王

- 设 $f_{i,j,S}$ 表示考虑到第 i 行，放了 j 个国王，下一行不能被放国王的位置状态为 S 的方案数。
- 因为在同一行中，国王是不能相邻的。所以可以考虑预处理出所有在同一行放国王的方法，以及这些方法会使下一行不能放的状态又是怎样。

k 国王

- 设 $f_{i,j,S}$ 表示考虑到第 i 行，放了 j 个国王，下一行不能被放国王的位置状态为 S 的方案数。
- 因为在同一行中，国王是不能相邻的。所以可以考虑预处理出所有在同一行放国王的方法，以及这些方法会使下一行不能放的状态又是怎样。
- 每次转移时枚举所有预处理好的状态，判断是否发生冲突，并进行转移。

NOIP2016 愤怒的小鸟

NOIP2016 愤怒的小鸟

- 平面上有 n 个点。你需要给出若干个满足 $a < 0$ 的抛物线 $ax^2 + bx = 0$，使得每个点都至少在一条抛物线上

NOIP2016 愤怒的小鸟

- 平面上有 n 个点。你需要给出若干个满足 $a < 0$ 的抛物线 $ax^2 + bx = 0$，使得每个点都至少在一条抛物线上
- 求最少的抛物线数量

NOIP2016 愤怒的小鸟

- 平面上有 n 个点。你需要给出若干个满足 $a < 0$ 的抛物线 $ax^2 + bx = 0$，使得每个点都至少在一条抛物线上
- 求最少的抛物线数量
- $n \leq 18$

NOIP2016 愤怒的小鸟

NOIP2016 愤怒的小鸟

- 设 $f[S]$ 表示：覆盖集合 S 中的所有点，至少要用几条抛物线

NOIP2016 愤怒的小鸟

- 设 $f[S]$ 表示：覆盖集合 S 中的所有点，至少要用几条抛物线
- 抛物线的参数有两个，因此枚举 n 个点中的任意两个点即可确定唯一的抛物线

NOIP2016 愤怒的小鸟

- 设 $f[S]$ 表示：覆盖集合 S 中的所有点，至少要用几条抛物线
- 抛物线的参数有两个，因此枚举 n 个点中的任意两个点即可确定唯一的抛物线
- 由此可以预处理出所有满足 $f[S]=1$ 的集合

NOIP2016 愤怒的小鸟

- 设 $f[S]$ 表示：覆盖集合 S 中的所有点，至少要用几条抛物线
- 抛物线的参数有两个，因此枚举 n 个点中的任意两个点即可确定唯一的抛物线
- 由此可以预处理出所有满足 $f[S]=1$ 的集合
- 考虑一个 DP 值不为 1 的集合，它一定能被拆成两部分，满足这两部分使用的抛物线互不相关

NOIP2016 愤怒的小鸟

- 设 $f[S]$ 表示：覆盖集合 S 中的所有点，至少要用几条抛物线
- 抛物线的参数有两个，因此枚举 n 个点中的任意两个点即可确定唯一的抛物线
- 由此可以预处理出所有满足 $f[S]=1$ 的集合
- 考虑一个 DP 值不为 1 的集合，它一定能被拆成两部分，满足这两部分使用的抛物线互不相关
- 由此得到转移方程：$f[S]=min_{T\in S}(f[T]+f[S-T])$

NOIP2016 愤怒的小鸟

NOIP2016 愤怒的小鸟

- 问题 1：如何枚举一个集合的所有子集?

NOIP2016 愤怒的小鸟

- 问题 1：如何枚举一个集合的所有子集?
- 从 $x = S$ 开始，每次取 $x = (x - 1)\&S$

NOIP2016 愤怒的小鸟

- 问题 1：如何枚举一个集合的所有子集?
- 从 $x = S$ 开始，每次取 $x = (x - 1)\&S$
- 证明：每次删掉最末尾的 1，并且把后面的 0 都改成 1；用与操作去掉不属于集合内的元素

NOIP2016 愤怒的小鸟

- 问题 1：如何枚举一个集合的所有子集?
- 从 $x = S$ 开始，每次取 $x = (x - 1)\&S$
- 证明：每次删掉最末尾的 1，并且把后面的 0 都改成 1；用与操作去掉不属于集合内的元素
- 问题 2：上述方程复杂度如何?

NOIP2016 愤怒的小鸟

- 问题 1：如何枚举一个集合的所有子集?
- 从 $x = S$ 开始，每次取 $x = (x - 1)\&S$
- 证明：每次删掉最末尾的 1，并且把后面的 0 都改成 1；用与操作去掉不属于集合内的元素
- 问题 2：上述方程复杂度如何?
- 等价于 0 2^n 内每个数的子集个数

NOIP2016 愤怒的小鸟

- 问题 1：如何枚举一个集合的所有子集?
- 从 $x = S$ 开始，每次取 $x = (x - 1)\&S$
- 证明：每次删掉最末尾的 1，并且把后面的 0 都改成 1；用与操作去掉不属于集合内的元素
- 问题 2：上述方程复杂度如何?
- 等价于 0 2^n 内每个数的子集个数
- $\sum_{i=0}^{n} 2^i \times C_n^i = (2+1)^n = 3^n$

NOIP2016 愤怒的小鸟

NOIP2016 愤怒的小鸟

- 存在问题：$O(3^n)$ 无法应对 $n = 18$ 的规模

NOIP2016 愤怒的小鸟

- 存在问题：$O(3^n)$ 无法应对 $n = 18$ 的规模
- 转换思路：每个集合 S 中，编号最小的点一定需要一条抛物线覆盖

NOIP2016 愤怒的小鸟

- 存在问题：$O(3^n)$ 无法应对 $n = 18$ 的规模
- 转换思路：每个集合 S 中，编号最小的点一定需要一条抛物线覆盖
- 候选抛物线个数：$O(n)$

NOIP2016 愤怒的小鸟

- 存在问题：$O(3^n)$ 无法应对 $n = 18$ 的规模
- 转换思路：每个集合 S 中，编号最小的点一定需要一条抛物线覆盖
- 候选抛物线个数：$O(n)$
- $f[S] = min(f[S - S_0]) + 1$

NOIP2016 愤怒的小鸟

- 存在问题：$O(3^n)$ 无法应对 $n = 18$ 的规模
- 转换思路：每个集合 S 中，编号最小的点一定需要一条抛物线覆盖
- 候选抛物线个数：$O(n)$
- $f[S] = min(f[S - S_0]) + 1$
- 复杂度：$O(n \times 2^n)$

NOIP2017 宝藏

NOIP2017 宝藏

- 给定一张无向图，你需要求出出图中的一棵有根生成树，并最小化图中每条边的深度与这条边的长度的乘积和

NOIP2017 宝藏

- 给定一张无向图，你需要求出出图中的一棵有根生成树，并最小化图中每条边的深度与这条边的长度的乘积和
- $n \leq 12, m \leq 5000$

NOIP2017 宝藏

NOIP2017 宝藏

- 设 $f[k][S]$ 表示：当前的深度为 k，已经包括了图中的子集 S 时的最小代价

NOIP2017 宝藏

- 设 $f[k][S]$ 表示：当前的深度为 k，已经包括了图中的子集 S 时的最小代价
- 转移：$f[k][S] = min_{T \in S}(f[k-1][S-T]) + W(S-T, T) \times k$

NOIP2017 宝藏

- 设 $f[k][S]$ 表示：当前的深度为 k，已经包括了图中的子集 S 时的最小代价
- 转移：$f[k][S] = min_{T \in S}(f[k-1][S-T]) + W(S-T, T) \times k$
- $W(S-T, T)$ 表示：子集 $S-T$ 中的点与 T 中的点连接的最小边权和

NOIP2017 宝藏

NOIP2017 宝藏

- 问题 1：如果 $S-T$ 中的点深度不全是 $k-1$，计算的错误会不会有影响?

NOIP2017 宝藏

- 问题 1：如果 $S-T$ 中的点深度不全是 $k-1$，计算的错误会不会有影响?
- 不会。因为这种错误的产生只会导致计算出的结果比实际结果大，不影响 DP 求出最优解

NOIP2017 宝藏

- 问题 1：如果 $S-T$ 中的点深度不全是 $k-1$，计算的错误会不会有影响?
- 不会。因为这种错误的产生只会导致计算出的结果比实际结果大，不影响 DP 求出最优解
- 问题 2：求 $W(S-T,T)$ 每次需要 $O(n^2)$ 的时间，会不会影响效率?

NOIP2017 宝藏

- 问题 1：如果 $S-T$ 中的点深度不全是 $k-1$，计算的错误会不会有影响?
- 不会。因为这种错误的产生只会导致计算出的结果比实际结果大，不影响 DP 求出最优解
- 问题 2：求 $W(S-T,T)$ 每次需要 $O(n^2)$ 的时间，会不会影响效率?
- 可以把 $W(S-T,T)$ 预处理成 $W[S-T][T]$，复杂度 $O(3^n)$

NOIP2017 宝藏

- 问题 1：如果 $S-T$ 中的点深度不全是 $k-1$，计算的错误会不会有影响?
- 不会。因为这种错误的产生只会导致计算出的结果比实际结果大，不影响 DP 求出最优解
- 问题 2：求 $W(S-T,T)$ 每次需要 $O(n^2)$ 的时间，会不会影响效率?
- 可以把 $W(S-T,T)$ 预处理成 $W[S-T][T]$，复杂度 $O(3^n)$
- 综上，总时间复杂度为 $O(n\times 3^n)$

山谷

山谷

- 给定一个 $n \times m$ 的地图。你需要向这个地图中填入 $1, 2, ..., n \times m$，每个数只能使用一次。

山谷

- 给定一个 $n \times m$ 的地图。你需要向这个地图中填入 $1, 2, ..., n \times m$，每个数只能使用一次。
- 如果一个位置比它的周围八个方向上的数都小，则这个位置可以被称为一个山谷。

山谷

- 给定一个 $n \times m$ 的地图。你需要向这个地图中填入 $1, 2, ..., n \times m$，每个数只能使用一次。
- 如果一个位置比它的周围八个方向上的数都小，则这个位置可以被称为一个山谷。
- 已知地图中所有山谷的位置，你需要求出合法填数的方案数。

山谷

- 给定一个 $n \times m$ 的地图。你需要向这个地图中填入 $1, 2, ..., n \times m$，每个数只能使用一次。
- 如果一个位置比它的周围八个方向上的数都小，则这个位置可以被称为一个山谷。
- 已知地图中所有山谷的位置，你需要求出合法填数的方案数。
- $n \leq 4$，$m \leq 7$

山谷

山谷

- 从小到大进行填数。

山谷

- 从小到大进行填数。
- 设 $f[S][j]$ 表示山谷的位置在集合 S 中的填了数（其余没有填），并且填到了 j 的方案数。

山谷

- 从小到大进行填数。
- 设 $f[S][j]$ 表示山谷的位置在集合 S 中的填了数（其余没有填），并且填到了 j 的方案数。
- 对于非山谷的位置，只有当它的周围的山谷都填好数（都属于 S）之后，它才可以填数。

山谷

- 从小到大进行填数。
- 设 $f[S][j]$ 表示山谷的位置在集合 S 中的填了数（其余没有填），并且填到了 j 的方案数。
- 对于非山谷的位置，只有当它的周围的山谷都填好数（都属于 S）之后，它才可以填数。
- 两种转移：$f[S][j] = f[S][j] + f[S - k][j - 1]$;
  $f[S][j] = f[S][j] + f[S][j - 1] * (c - j + 1)$，其中 c 表示在枚举了集合 S 之后，所有可以填数的位置的个数。

山谷

山谷

- 最后 $f[$ 全集 $][n*m]$ 即为答案。

山谷

- 最后 $f[$ 全集 $][n*m]$ 即为答案。
- 一个小问题：在非山谷位置上任意填数可能会创造出一个山谷。

山谷

- 最后 $f[$ 全集 $][n*m]$ 即为答案。
- 一个小问题：在非山谷位置上任意填数可能会创造出一个山谷。
- 解决：容斥原理

STA

- 给定一棵 n 个点的无根树。

- 给定一棵 n 个点的无根树。
- 你需要找一个点做根，使得所有点深度（即到根经过的边数）之和最大。

- 给定一棵 n 个点的无根树。
- 你需要找一个点做根，使得所有点深度（即到根经过的边数）之和最大。
- $n \leq 100000$。

STA

- 对于这种“挑一个最优的点”的题目，我们往往先找一个点，计算出相应的值，然后考虑把一个点的值转移到相邻的点上。

STA

- 对于这种“挑一个最优的点”的题目，我们往往先找一个点，计算出相应的值，然后考虑把一个点的值转移到相邻的点上。
- 设 f_i 表示以 i 为根的时候的答案。

- 对于这种“挑一个最优的点”的题目，我们往往先找一个点，计算出相应的值，然后考虑把一个点的值转移到相邻的点上。
- 设 f_i 表示以 i 为根的时候的答案。
- 先以 1 号点为根，并计算出 f_1。

- 对于这种“挑一个最优的点”的题目，我们往往先找一个点，计算出相应的值，然后考虑把一个点的值转移到相邻的点上。
- 设 f_i 表示以 i 为根的时候的答案。
- 先以 1 号点为根，并计算出 f_1。
- 假设以 1 号点为根时，u 是 v 的父亲。并且 f_u 的值已经被求出。

- 对于这种“挑一个最优的点”的题目，我们往往先找一个点，计算出相应的值，然后考虑把一个点的值转移到相邻的点上。
- 设 f_i 表示以 i 为根的时候的答案。
- 先以 1 号点为根，并计算出 f_1。
- 假设以 1 号点为根时，u 是 v 的父亲。并且 f_u 的值已经被求出。
- 考虑根从 u 移动到 v，则 v 的子树中的点的深度都变小了 1，其余点都变大了 1。

- 对于这种“挑一个最优的点”的题目，我们往往先找一个点，计算出相应的值，然后考虑把一个点的值转移到相邻的点上。
- 设 f_i 表示以 i 为根的时候的答案。
- 先以 1 号点为根，并计算出 f_1。
- 假设以 1 号点为根时，u 是 v 的父亲。并且 f_u 的值已经被求出。
- 考虑根从 u 移动到 v，则 v 的子树中的点的深度都变小了 1，其余点都变大了 1。
- 即变化量为 $-size_v + n - size_v = n - 2size_v$，其中 $size_v$ 为子树 v 内的节点个数。

- 对于这种“挑一个最优的点”的题目，我们往往先找一个点，计算出相应的值，然后考虑把一个点的值转移到相邻的点上。
- 设 f_i 表示以 i 为根的时候的答案。
- 先以 1 号点为根，并计算出 f_1。
- 假设以 1 号点为根时，u 是 v 的父亲。并且 f_u 的值已经被求出。
- 考虑根从 u 移动到 v，则 v 的子树中的点的深度都变小了 1，其余点都变大了 1。
- 即变化量为 $-size_v + n - size_v = n - 2size_v$，其中 $size_v$ 为子树 v 内的节点个数。
- 所以方程为 $f_v = f_u + n - 2size_v$，根据方程进行 DP 即可。

最大连通块

最大连通块

- 给定一棵 n 个点的树，每个点上有点权（可能为负）。

最大连通块

- 给定一棵 n 个点的树，每个点上有点权（可能为负）。
- 你需要挑出树里的一个点集，使这个点集中任意两点路径上的点也在点集中（也就是一个连通块）。在此基础上，你需要使点集内的点权和最大。

最大连通块

- 给定一棵 n 个点的树，每个点上有点权（可能为负）。
- 你需要挑出树里的一个点集，使这个点集中任意两点路径上的点也在点集中（也就是一个连通块）。在此基础上，你需要使点集内的点权和最大。
- $n \leq 100000$。

最大连通块

最大连通块

- 还是挑一个点为根，用 dfs 来进行 DP。

最大连通块

- 还是挑一个点为根，用 dfs 来进行 DP。
- 设 f_i 表示只考虑以 i 为根的子树，在点 i 一定需要被选择的情况下的最大连通块和是多少。

最大连通块

- 还是挑一个点为根，用 dfs 来进行 DP。
- 设 f_i 表示只考虑以 i 为根的子树，在点 i 一定需要被选择的情况下的最大连通块和是多少。
- $f_i = v_i + \sum f_j$。其中，v_i 表示 i 的点权，j 是 i 的儿子，$\sum f_j$ 的含义是把所有 $f_j > 0$ 的部分全部加上。

最大连通块

- 还是挑一个点为根，用 dfs 来进行 DP。
- 设 f_i 表示只考虑以 i 为根的子树，在点 i 一定需要被选择的情况下的最大连通块和是多少。
- $f_i = v_i + \sum f_j$。其中，v_i 表示 i 的点权，j 是 i 的儿子，$\sum f_j$ 的含义是把所有 $f_j > 0$ 的部分全部加上。
- 对于任意一个连通块，存在唯一一个点最靠近根，则一个最优的连通块会在最靠近根的点上被计入答案。

树上操作

树上操作

- 给定一棵 n 个点的树，每条边有边权

树上操作

- 给定一棵 n 个点的树，每条边有边权
- 你需要将树上恰好 k 个点染成黑色，其余点染成白色。染色后你的得分是树上所有黑色点对的距离之和加上白色点对的距离之和。求最大得分

树上操作

- 给定一棵 n 个点的树，每条边有边权
- 你需要将树上恰好 k 个点染成黑色，其余点染成白色。染色后你的得分是树上所有黑色点对的距离之和加上白色点对的距离之和。求最大得分
- $k \leq n \leq 2000$。

树上操作

树上操作

- 直接计算点对距离并不好算，可以考虑计算每条边对答案的贡献

树上操作

- 直接计算点对距离并不好算，可以考虑计算每条边对答案的贡献
- 设 $f[u][j]$ 表示将子树 u 内的 j 个点染黑，其余点染白的最大得分

树上操作

- 直接计算点对距离并不好算，可以考虑计算每条边对答案的贡献
- 设 $f[u][j]$ 表示将子树 u 内的 j 个点染黑，其余点染白的最大得分
- 假设 u 有 p 个儿子 $s_1, s_2, ..., s_p$，可以再使用一次 DP 来求解 $f[u][j]$

树上操作

- 直接计算点对距离并不好算，可以考虑计算每条边对答案的贡献
- 设 $f[u][j]$ 表示将子树 u 内的 j 个点染黑，其余点染白的最大得分
- 假设 u 有 p 个儿子 $s_1, s_2, ..., s_p$，可以再使用一次 DP 来求解 $f[u][j]$
- 设 $g[i][j]$ 表示考虑到第 i 个儿子，取了 j 个黑点的最大得分，则 $g[i][j] = max(g[i-1][j-k] + f[s_i][k] + w(s_i, k))$

树上操作

- 直接计算点对距离并不好算，可以考虑计算每条边对答案的贡献
- 设 $f[u][j]$ 表示将子树 u 内的 j 个点染黑，其余点染白的最大得分
- 假设 u 有 p 个儿子 $s_1, s_2, ..., s_p$，可以再使用一次 DP 来求解 $f[u][j]$
- 设 $g[i][j]$ 表示考虑到第 i 个儿子，取了 j 个黑点的最大得分，则 $g[i][j] = max(g[i-1][j-k] + f[s_i][k] + w(s_i, k))$
- 其中，k 表示 s_i 的黑点个数，$w(s_i, k)$ 表示：如果 s_i 中有 k 个黑点，那么 u 与 s_i 之间的边对得分的贡献

树上操作

树上操作

- 对于一个点 u，第二层 DP 的复杂度为：u 的所有儿子的子树大小的乘积和

树上操作

- 对于一个点 u，第二层 DP 的复杂度为：u 的所有儿子的子树大小的乘积和
- 结论：每个点的所有儿子子树大小的乘积和的和复杂度为 $O(n^2)$

树上操作

- 对于一个点 u，第二层 DP 的复杂度为：u 的所有儿子的子树大小的乘积和
- 结论：每个点的所有儿子子树大小的乘积和的和复杂度为 $O(n^2)$
- 证明：可以看作任意一对点在其最近公共祖先处对答案贡献了 1

树上操作

- 对于一个点 u，第二层 DP 的复杂度为：u 的所有儿子的子树大小的乘积和
- 结论：每个点的所有儿子子树大小的乘积和的和复杂度为 $O(n^2)$
- 证明：可以看作任意一对点在其最近公共祖先处对答案贡献了 1
- 综上，上述 DP 总复杂度为 $O(n^2)$

数位统计

数位统计

- 给定正整数 a 和 b，求 a 到 b 中每一个数字出现了多少次。

数位统计

- 给定正整数 a 和 b，求 a 到 b 中每一个数字出现了多少次。
- $a \leq b \leq 10^{18}$。

数位统计

数位统计

- 数位 DP 题。

数位统计

- 数位 DP 题。
- 考虑差分区间，问题可以转化为求 1 到 a 的每一个数字出现了多少次。

数位统计

- 数位 DP 题。
- 考虑差分区间，问题可以转化为求 1 到 a 的每一个数字出现了多少次。
- 从高到低考虑每一位。如果最高位位选得比 a 的对应位小，则后面的位置可以任取，直接计算就可以加进答案。

数位统计

- 数位 DP 题。
- 考虑差分区间，问题可以转化为求 1 到 a 的每一个数字出现了多少次。
- 从高到低考虑每一位。如果最高位位选得比 a 的对应位小，则后面的位置可以任取，直接计算就可以加进答案。
- 如果选得和 a 的对应位一样，则可以单独考虑这一位对答案的贡献，然后把这一位去掉。这样问题被转化为了一个规模小了一位的子问题。

数位统计

- 数位 DP 题。
- 考虑差分区间，问题可以转化为求 1 到 a 的每一个数字出现了多少次。
- 从高到低考虑每一位。如果最高位位选得比 a 的对应位小，则后面的位置可以任取，直接计算就可以加进答案。
- 如果选得和 a 的对应位一样，则可以单独考虑这一位对答案的贡献，然后把这一位去掉。这样问题被转化为了一个规模小了一位的子问题。
- 这道题的做法包含了数位 DP 最常出现的两个手法：差分区间、枚举前若干位与上界相同。

windy 数

windy 数

- 给定正整数 a 和 b，求 a 到 b 中相邻两个数字之差至少为 2 的数有多少个。答案对一个大质数取模。

windy 数

- 给定正整数 a 和 b，求 a 到 b 中相邻两个数字之差至少为 2 的数有多少个。答案对一个大质数取模。
- $a \leq b \leq 10^{10000}$。

windy 数

windy 数

- 先预处理 $g_{i,j}$ 表示在最前面放一个 i，后面放 j 个数位时，符合条件的方案数。枚举 j 个数位中的第一个即可进行转移。

windy 数

- 先预处理 $g_{i,j}$ 表示在最前面放一个 i，后面放 j 个数位时，符合条件的方案数。枚举 j 个数位中的第一个即可进行转移。
- 还是把问题转化成 1 到 a 上的问题，然后枚举数的前 i 位和 a 一样，再枚举第 $i+1$ 位的取值。后面的方案数已经被预处理了，直接计入答案即可。

杠杆数

杠杆数

- 对于一个数，如果我们把某一位看成支点，支点左右的力矩相同，就可以把这个数叫做杠杆数。

杠杆数

- 对于一个数，如果我们把某一位看成支点，支点左右的力矩相同，就可以把这个数叫做杠杆数。
- 比如 1325 就是一个杠杆数，2 是支点。

杠杆数

- 对于一个数，如果我们把某一位看成支点，支点左右的力矩相同，就可以把这个数叫做杠杆数。
- 比如 1325 就是一个杠杆数，2 是支点。
- 求 1 到 n 内有多少个杠杆数。

杠杆数

- 对于一个数，如果我们把某一位看成支点，支点左右的力矩相同，就可以把这个数叫做杠杆数。
- 比如 1325 就是一个杠杆数，2 是支点。
- 求 1 到 n 内有多少个杠杆数。
- $n \leq 10^9$。

杠杆数

杠杆数

- 枚举前若干位和 n 相同，再枚举比 n 小的第一位，最后枚举杠杆的位置在数的哪一位，就可以做 DP 了。分别求出 $f_{i,j}$ 和 $g_{i,j}$ 分别表示取到杠杆往左或往右第 i 位，力矩值为 j 时的方案数。如果某个位置固定了，就只能拿那个元素转移。否则可以枚举放的数是什么。最后把力矩值相同的对应位乘起来即可。

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。
- 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

- 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

- 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

- 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

- 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

- 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。

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（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（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（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（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（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：
- $f_{j-1} + a_j \times b_i \leq f_{k-1} + a_k \times b_i$。
- 利用这个式子即可化出斜率优化的方程。

balls

balls

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。你可以选择序列的一个区间 $[l, r]$，把 $a_l, a_{l+1}, ..., a_{r-1}$ 都改成 a_r

balls

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。你可以选择序列的一个区间 $[l, r]$，把 $a_l, a_{l+1}, ..., a_{r-1}$ 都改成 a_r
- 最大化修改后序列的和

balls

- 给定一个正整数序列 $a_1, a_2, ..., a_n$。你可以选择序列的一个区间 $[l, r]$，把 $a_l, a_{l+1}, ..., a_{r-1}$ 都改成 a_r
- 最大化修改后序列的和
- $n \leq 100000$。

balls

balls

- 求出序列的前缀和数组 $s_i = \sum_{k=1}^{i} a_i$

balls

- 求出序列的前缀和数组 $s_i = \sum_{k=1}^{i} a_i$
- 则修改 $[l, r]$ 后，序列的和为：$s_{l-1} + (r - l + 1) \times a_r + s_n - s_r$

balls

- 求出序列的前缀和数组 $s_i = \sum_{k=1}^{i} a_i$
- 则修改 $[l, r]$ 后，序列的和为：$s_{l-1} + (r - l + 1) \times a_r + s_n - s_r$
- 如果固定 r，则我们需要选择 l 使 $s_{l-1} - (l - 1) \times a_r$ 最大，也就是选择 k 使 $s_k - k \times a_r$ 最大，且 $k < r$

balls

balls

- 假设 $p < q < r$，且 q 优于 p，则有 $s_p - p \times a_r < s_q - q \times a_r$

balls

- 假设 $p < q < r$，且 q 优于 p，则有 $s_p - p \times a_r < s_q - q \times a_r$
- 整理得 $\frac{s_q - s_p}{q - p} > a_r$

balls

- 假设 $p < q < r$，且 q 优于 p，则有 $s_p - p \times a_r < s_q - q \times a_r$
- 整理得 $\frac{s_q - s_p}{q - p} > a_r$
- 问：a_r 不单调，如何求解最优值?

平均值

平均值

- 给定一个正整数序列 $a_1, a_2, ..., a_n$，再给出一个区间 $[L, R]$，你需要求出序列的一个子区间 $[l, r]$，满足 $L \leq r - l + 1 \leq R$，且区间的平均值最大，即 $\frac{a_l + a_{l+1} + ... + a_r}{r - l + 1}$ 最大

平均值

- 给定一个正整数序列 $a_1, a_2, ..., a_n$，再给出一个区间 $[L, R]$，你需要求出序列的一个子区间 $[l, r]$，满足 $L \leq r - l + 1 \leq R$，且区间的平均值最大，即 $\frac{a_l + a_{l+1} + ... + a_r}{r - l + 1}$ 最大
- $n \leq 100000$

平均值

平均值

- 对于这一类问题，我们一般使用 01 分数规划解决。即：二分 $\frac{a_l+a_{l+1}+...+a_r}{r-l+1}$ 的值

平均值

- 对于这一类问题，我们一般使用 01 分数规划解决。即：二分 $\frac{a_l+a_{l+1}+...+a_r}{r-l+1}$ 的值
- 二分出 $M=\frac{a_l+a_{l+1}+...+a_r}{r-l+1}$。假设：这个二分值偏小。则可以找到一组 l,r，使得 $M<\frac{a_l+a_{l+1}+...+a_r}{r-l+1}$ 成立

平均值

- 对于这一类问题，我们一般使用 01 分数规划解决。即：二分 $\frac{a_l+a_{l+1}+...+a_r}{r-l+1}$ 的值
- 二分出 $M=\frac{a_l+a_{l+1}+...+a_r}{r-l+1}$。假设：这个二分值偏小。则可以找到一组 l,r，使得 $M<\frac{a_l+a_{l+1}+...+a_r}{r-l+1}$ 成立
- 将不等式展开可得：$(a_l-M)+(a_{l+1}-M)+...+(a_r-M)>0$

平均值

- 对于这一类问题，我们一般使用 01 分数规划解决。即：二分 $\frac{a_l+a_{l+1}+...+a_r}{r-l+1}$ 的值
- 二分出 $M=\frac{a_l+a_{l+1}+...+a_r}{r-l+1}$。假设：这个二分值偏小。则可以找到一组 l,r，使得 $M<\frac{a_l+a_{l+1}+...+a_r}{r-l+1}$ 成立
- 将不等式展开可得：$(a_l-M)+(a_{l+1}-M)+...+(a_r-M)>0$
- 也就是判断：数列 $b_i=a_i-M$ 是否存在着长度在 $[L,R]$ 之间，且和大于 0 的子段

平均值

平均值

- 设 s_i 为序列 b_i 的前缀和数组

平均值

- 设 s_i 为序列 b_i 的前缀和数组
- 枚举 r，则需要在 $r-R+1 \le l\ r-L+1$ 中找出最小的 s_{l-1}，即找出 $r-R \le k \le r-L$ 的最小的 s_k

平均值

- 设 s_i 为序列 b_i 的前缀和数组
- 枚举 r，则需要在 $r-R+1 \le l\ r-L+1$ 中找出最小的 s_{l-1}，即找出 $r-R \le k \le r-L$ 的最小的 s_k
- 扫描 + 单调队列

平均值

- 设 s_i 为序列 b_i 的前缀和数组
- 枚举 r，则需要在 $r-R+1 \le l\ r-L+1$ 中找出最小的 s_{l-1}，即找出 $r-R \le k \le r-L$ 的最小的 s_k
- 扫描 + 单调队列
- 根据得到的结果调整二分值的大小

平均值 2

平均值 2

- 给定一个正整数序列 $a_1, a_2, ..., a_n$，再给出一个区间 $[L, R]$，你需要求出序列的一个子区间 $[l, r]$，满足 $L \leq r - l + 1 \leq R$，且区间的平均值最大

平均值 2

- 给定一个正整数序列 $a_1, a_2, ..., a_n$，再给出一个区间 $[L, R]$，你需要求出序列的一个子区间 $[l, r]$，满足 $L \leq r - l + 1 \leq R$，且区间的平均值最大
- 对于本题，平均值的定义为区间最大值减最小值除以区间长度，即 $\frac{Max(a_l, a_{l+1}, ..., a_r) - Min(a_l, a_{l+1}, ..., a_r)}{r - l + 1}$ 最大

平均值 2

- 给定一个正整数序列 $a_1, a_2, ..., a_n$，再给出一个区间 $[L, R]$，你需要求出序列的一个子区间 $[l, r]$，满足 $L \leq r - l + 1 \leq R$，且区间的平均值最大
- 对于本题，平均值的定义为区间最大值减最小值除以区间长度，即 $\frac{Max(a_l, a_{l+1}, ..., a_r) - Min(a_l, a_{l+1}, ..., a_r)}{r-l+1}$ 最大
- $n \leq 100000$

平均值 2

平均值 2

- 先暂时不考虑区间长度限制带来的影响

平均值 2

- 先暂时不考虑区间长度限制带来的影响
- 此时如果两个子段的最大、最小值选取的位置一样，那么显然比较短的那个更优

平均值 2

- 先暂时不考虑区间长度限制带来的影响
- 此时如果两个子段的最大、最小值选取的位置一样，那么显然比较短的那个更优
- 在不考虑区间长度限制的条件下，最优的选法显然是最大、最小值在区间两端

平均值 2

- 先暂时不考虑区间长度限制带来的影响
- 此时如果两个子段的最大、最小值选取的位置一样，那么显然比较短的那个更优
- 在不考虑区间长度限制的条件下，最优的选法显然是最大、最小值在区间两端
- 不妨假设区间右端是最大值，左端是最小值，之后只需要将区间倒过来重复一次就可以覆盖所有答案

平均值 2

平均值 2

- 此时，可以把平均值的式子写成：$\frac{a_r - a_l}{r - l + 1}$

平均值 2

- 此时，可以把平均值的式子写成：$\frac{a_r-a_l}{r-l+1}$
- 二分上式的值，设 $M=\frac{a_r-a_l}{r-l+1}$

平均值 2

- 此时，可以把平均值的式子写成：$\frac{a_r-a_l}{r-l+1}$
- 二分上式的值，设 $M=\frac{a_r-a_l}{r-l+1}$
- 若二分值偏小，则有 $(a_r-r\times M)-(a_l-l\times M)>M$

平均值 2

- 此时，可以把平均值的式子写成：$\frac{a_r-a_l}{r-l+1}$
- 二分上式的值，设 $M=\frac{a_r-a_l}{r-l+1}$
- 若二分值偏小，则有 $(a_r-r\times M)-(a_l-l\times M)>M$
- 构造新数列 $b_i=a_i-i\times M$，求出 b_i 带区间长度限制的 b_r-b_l 的最大值，与 M 进行比较即可判断二分值偏大或偏小

平均值 2

平均值 2

- 注意这么一件事：在处理 Max 与 Min 的时候，我们忽略了区间长度的限制

平均值 2

- 注意这么一件事：在处理 Max 与 Min 的时候，我们忽略了区间长度的限制
- 但用来判断二分值时，我们又加上了这条限制

平均值 2

- 注意这么一件事：在处理 Max 与 Min 的时候，我们忽略了区间长度的限制
- 但用来判断二分值时，我们又加上了这条限制
- 这就导致：求解出的二分值，只包含了 Max 和 Min 的距离 $\geq L$ 的情况

平均值 2

- 注意这么一件事：在处理 Max 与 Min 的时候，我们忽略了区间长度的限制
- 但用来判断二分值时，我们又加上了这条限制
- 这就导致：求解出的二分值，只包含了 Max 和 Min 的距离 $\geq L$ 的情况
- 因此，最后我们还需要枚举所有长度恰好为 L 的子段，求出其真正的平均值，用于更新最后的答案（如何更新?）

NOIP2014 飞扬的小鸟

NOIP2014 飞扬的小鸟

- 一只小鸟在屏幕中飞，屏幕宽度为 n，高度为 m。小鸟从最左边开始，每单位时间都会往右飞一格

NOIP2014 飞扬的小鸟

- 一只小鸟在屏幕中飞，屏幕宽度为 n，高度为 m。小鸟从最左边开始，每单位时间都会往右飞一格
- 当小鸟在横坐标第 i 格时，玩家点击一次屏幕会使小鸟上升 x_i 的高度，可以多次点击；若不点，则小鸟会下降 y_i 的高度

NOIP2014 飞扬的小鸟

- 一只小鸟在屏幕中飞，屏幕宽度为 n，高度为 m。小鸟从最左边开始，每单位时间都会往右飞一格
- 当小鸟在横坐标第 i 格时，玩家点击一次屏幕会使小鸟上升 x_i 的高度，可以多次点击；若不点，则小鸟会下降 y_i 的高度
- 小鸟的高度不会超过屏幕的高度限制。如果一次上升操作会导致小鸟飞出屏幕，则小鸟的高度会保留在屏幕最高处

NOIP2014 飞扬的小鸟

- 一只小鸟在屏幕中飞，屏幕宽度为 n，高度为 m。小鸟从最左边开始，每单位时间都会往右飞一格
- 当小鸟在横坐标第 i 格时，玩家点击一次屏幕会使小鸟上升 x_i 的高度，可以多次点击；若不点，则小鸟会下降 y_i 的高度
- 小鸟的高度不会超过屏幕的高度限制。如果一次上升操作会导致小鸟飞出屏幕，则小鸟的高度会保留在屏幕最高处
- 飞行过程中小鸟的高度不能小于等于 0，否则游戏失败

NOIP2014 飞扬的小鸟

- 一只小鸟在屏幕中飞，屏幕宽度为 n，高度为 m。小鸟从最左边开始，每单位时间都会往右飞一格
- 当小鸟在横坐标第 i 格时，玩家点击一次屏幕会使小鸟上升 x_i 的高度，可以多次点击；若不点，则小鸟会下降 y_i 的高度
- 小鸟的高度不会超过屏幕的高度限制。如果一次上升操作会导致小鸟飞出屏幕，则小鸟的高度会保留在屏幕最高处
- 飞行过程中小鸟的高度不能小于等于 0，否则游戏失败
- 屏幕中有若干个管道，每个管道要求小鸟在到达某个横坐标时高度处于某个区间范围内。一个横坐标位置至多只有一个管道

NOIP2014 飞扬的小鸟

- 一只小鸟在屏幕中飞，屏幕宽度为 n，高度为 m。小鸟从最左边开始，每单位时间都会往右飞一格
- 当小鸟在横坐标第 i 格时，玩家点击一次屏幕会使小鸟上升 x_i 的高度，可以多次点击；若不点，则小鸟会下降 y_i 的高度
- 小鸟的高度不会超过屏幕的高度限制。如果一次上升操作会导致小鸟飞出屏幕，则小鸟的高度会保留在屏幕最高处
- 飞行过程中小鸟的高度不能小于等于 0，否则游戏失败
- 屏幕中有若干个管道，每个管道要求小鸟在到达某个横坐标时高度处于某个区间范围内。一个横坐标位置至多只有一个管道
- 求出小鸟是否能到达终点。若能，求出到达终点所需的最小屏幕点击次数；否则输出小鸟最多经过的管道数量

NOIP2014 飞扬的小鸟

- 一只小鸟在屏幕中飞，屏幕宽度为 n，高度为 m。小鸟从最左边开始，每单位时间都会往右飞一格
- 当小鸟在横坐标第 i 格时，玩家点击一次屏幕会使小鸟上升 x_i 的高度，可以多次点击；若不点，则小鸟会下降 y_i 的高度
- 小鸟的高度不会超过屏幕的高度限制。如果一次上升操作会导致小鸟飞出屏幕，则小鸟的高度会保留在屏幕最高处
- 飞行过程中小鸟的高度不能小于等于 0，否则游戏失败
- 屏幕中有若干个管道，每个管道要求小鸟在到达某个横坐标时高度处于某个区间范围内。一个横坐标位置至多只有一个管道
- 求出小鸟是否能到达终点。若能，求出到达终点所需的最小屏幕点击次数；否则输出小鸟最多经过的管道数量
- $n \leq 10000, m \leq 1000$

NOIP2014 飞扬的小鸟

NOIP2014 飞扬的小鸟

- 设 $f_{i,j}$ 表示小鸟飞到坐标 (i,j) 的最少屏幕点击次数

NOIP2014 飞扬的小鸟

- 设 $f_{i,j}$ 表示小鸟飞到坐标 (i,j) 的最少屏幕点击次数
- 从 $f_{i,j}$ 可以转移到 $f_{i+1,j-y_i}$ 以及 $f_{i+1,m}$，后者的屏幕点击次数可以预先计算

NOIP2014 飞扬的小鸟

- 设 $f_{i,j}$ 表示小鸟飞到坐标 (i,j) 的最少屏幕点击次数
- 从 $f_{i,j}$ 可以转移到 $f_{i+1,j-y_i}$ 以及 $f_{i+1,m}$，后者的屏幕点击次数可以预先计算
- 从 $f_{i,j}$ 还可以转移到 $f_{i+1,j+k\times x_i}$。但这一步不可以暴力进行转移（否则时间复杂度无法承受）

NOIP2014 飞扬的小鸟

- 设 $f_{i,j}$ 表示小鸟飞到坐标 (i,j) 的最少屏幕点击次数
- 从 $f_{i,j}$ 可以转移到 $f_{i+1,j-y_i}$ 以及 $f_{i+1,m}$，后者的屏幕点击次数可以预先计算
- 从 $f_{i,j}$ 还可以转移到 $f_{i+1,j+k\times x_i}$。但这一步不可以暴力进行转移（否则时间复杂度无法承受）
- 如果先不进行 $f_{i,j}$ 到 $f_{i+1,j-y_i}$ 的转移，那么 $f_{i+1,j}$ 可以由 $f_{i+1,j-x_i}+1$ 转移而来

NOIP2014 飞扬的小鸟

- 设 $f_{i,j}$ 表示小鸟飞到坐标 (i,j) 的最少屏幕点击次数
- 从 $f_{i,j}$ 可以转移到 $f_{i+1,j-y_i}$ 以及 $f_{i+1,m}$，后者的屏幕点击次数可以预先计算
- 从 $f_{i,j}$ 还可以转移到 $f_{i+1,j+k\times x_i}$。但这一步不可以暴力进行转移（否则时间复杂度无法承受）
- 如果先不进行 $f_{i,j}$ 到 $f_{i+1,j-y_i}$ 的转移，那么 $f_{i+1,j}$ 可以由 $f_{i+1,j-x_i}+1$ 转移而来
- 思路理解：类似于“最短路 6”中的加边优化

NOIP2014 飞扬的小鸟

- 设 $f_{i,j}$ 表示小鸟飞到坐标 (i,j) 的最少屏幕点击次数
- 从 $f_{i,j}$ 可以转移到 $f_{i+1,j-y_i}$ 以及 $f_{i+1,m}$，后者的屏幕点击次数可以预先计算
- 从 $f_{i,j}$ 还可以转移到 $f_{i+1,j+k\times x_i}$。但这一步不可以暴力进行转移（否则时间复杂度无法承受）
- 如果先不进行 $f_{i,j}$ 到 $f_{i+1,j-y_i}$ 的转移，那么 $f_{i+1,j}$ 可以由 $f_{i+1,j-x_i}+1$ 转移而来
- 思路理解：类似于“最短路 6”中的加边优化
- 复杂度 $O(nm)$

NOIP2015 子串

NOIP2015 子串

- 给定两个由小写字母组成的字符串 A, B，A 的长度为 n，B 的长度为 m

NOIP2015 子串

- 给定两个由小写字母组成的字符串 A, B，A 的长度为 n，B 的长度为 m
- 从 A 中取出恰好 t 个非空子串，将其拼接在一起。求拼接成的新串恰好是 B 的方案数对一个大质数取模后的结果

NOIP2015 子串

- 给定两个由小写字母组成的字符串 A, B，A 的长度为 n，B 的长度为 m
- 从 A 中取出恰好 t 个非空子串，将其拼接在一起。求拼接成的新串恰好是 B 的方案数对一个大质数取模后的结果
- $n \leq 1000, m \leq 200$，空间限制 128MB

NOIP2015 子串

NOIP2015 子串

- 设 $f_{i,j,k}$ 表示：A 考虑到字符 i，B 考虑到字符 j，恰好取了 k 个子串的方案数

NOIP2015 子串

- 设 $f_{i,j,k}$ 表示：A 考虑到字符 i，B 考虑到字符 j，恰好取了 k 个子串的方案数
- 如果 A_i 与 B_j 不同，则只能有 $f_{i,j,k} = f_{i-1,j,k}$

NOIP2015 子串

- 设 $f_{i,j,k}$ 表示：A 考虑到字符 i，B 考虑到字符 j，恰好取了 k 个子串的方案数
- 如果 A_i 与 B_j 不同，则只能有 $f_{i,j,k} = f_{i-1,j,k}$
- 否则，需要分两种情况讨论：A_i 是一个新的子串的开头；A_i 是上一个子串的延续

NOIP2015 子串

- 设 $f_{i,j,k}$ 表示：A 考虑到字符 i，B 考虑到字符 j，恰好取了 k 个子串的方案数
- 如果 A_i 与 B_j 不同，则只能有 $f_{i,j,k} = f_{i-1,j,k}$
- 否则，需要分两种情况讨论：A_i 是一个新的子串的开头；A_i 是上一个子串的延续
- 为了区分这两种情况，需要对状态进行拓展：用一维 01 变量记录当前 A 的最末尾字符是否属于某个子串中

NOIP2015 子串

NOIP2015 子串

- 新的转移：不论 A_i 与 B_j 是否相同，都有 $f_{i,j,k,0} = f_{i-1,j,k,0} + f_{i-1,j,k,1}$

NOIP2015 子串

- 新的转移：不论 A_i 与 B_j 是否相同，都有 $f_{i,j,k,0} = f_{i-1,j,k,0} + f_{i-1,j,k,1}$
- 如果 A_i 与 B_j 不同，则只能有 $f_{i,j,k,1} = 0$

NOIP2015 子串

- 新的转移：不论 A_i 与 B_j 是否相同，都有 $f_{i,j,k,0} = f_{i-1,j,k,0} + f_{i-1,j,k,1}$
- 如果 A_i 与 B_j 不同，则只能有 $f_{i,j,k,1} = 0$
- 否则，$f_{i,j,k,1} = f_{i-1,j-1,k,1} + f_{i-1,j-1,k-1,0} + f_{i-1,j-1,k-1,1}$

NOIP2015 子串

- 新的转移：不论 A_i 与 B_j 是否相同，都有 $f_{i,j,k,0} = f_{i-1,j,k,0} + f_{i-1,j,k,1}$
- 如果 A_i 与 B_j 不同，则只能有 $f_{i,j,k,1} = 0$
- 否则，$f_{i,j,k,1} = f_{i-1,j-1,k,1} + f_{i-1,j-1,k-1,0} + f_{i-1,j-1,k-1,1}$
- 复杂度：$O(nm^2)$

NOIP2015 子串

- 新的转移：不论 A_i 与 B_j 是否相同，都有 $f_{i,j,k,0} = f_{i-1,j,k,0} + f_{i-1,j,k,1}$
- 如果 A_i 与 B_j 不同，则只能有 $f_{i,j,k,1} = 0$
- 否则，$f_{i,j,k,1} = f_{i-1,j-1,k,1} + f_{i-1,j-1,k-1,0} + f_{i-1,j-1,k-1,1}$
- 复杂度：$O(nm^2)$
- 空间过大？滚动第一维

换教室

换教室

- 给定一张 v 个点的地图，你需要从第 $a[1]$ 个点走到 $a[n]$ 个点。

换教室

- 给定一张 v 个点的地图，你需要从第 $a[1]$ 个点走到 $a[n]$ 个点。
- 为了使总路程最小，你有 m 次机会修改 $a[i]$。一次修改可以使 $a[i]$ 以 $k[i]$ 的几率变为 $b[i]$。修改是同时进行的，且不能修改同一个 $a[i]$ 两次。

换教室

- 给定一张 v 个点的地图，你需要从第 $a[1]$ 个点走到 $a[n]$ 个点。
- 为了使总路程最小，你有 m 次机会修改 $a[i]$。一次修改可以使 $a[i]$ 以 $k[i]$ 的几率变为 $b[i]$。修改是同时进行的，且不能修改同一个 $a[i]$ 两次。
- 求走过的总路径的长度的最小期望值。

换教室

- 给定一张 v 个点的地图，你需要从第 $a[1]$ 个点走到 $a[n]$ 个点。
- 为了使总路程最小，你有 m 次机会修改 $a[i]$。一次修改可以使 $a[i]$ 以 $k[i]$ 的几率变为 $b[i]$。修改是同时进行的，且不能修改同一个 $a[i]$ 两次。
- 求走过的总路径的长度的最小期望值。
- $v \leq 300$， $n, m \leq 2000$

换教室

换教室

- 图的点数很少，可以对全图跑一次 floyd 来得到任意两个点的最小距离。

换教室

- 图的点数很少，可以对全图跑一次 floyd 来得到任意两个点的最小距离。
- dp 的部分和概率相关，这类题目一般被称为概率 DP。

换教室

- 图的点数很少，可以对全图跑一次 floyd 来得到任意两个点的最小距离。
- dp 的部分和概率相关，这类题目一般被称为概率 DP。
- 事实上，概率 dp 可以看成状态是加权的。在一般的 dp 中，一个状态一般是确定的。而在概率 dp 中，一个状态可能是不定的，每种不定的状态都会以一定的概率出现。

换教室

- 图的点数很少，可以对全图跑一次 floyd 来得到任意两个点的最小距离。
- dp 的部分和概率相关，这类题目一般被称为概率 DP。
- 事实上，概率 dp 可以看成状态是加权的。在一般的 dp 中，一个状态一般是确定的。而在概率 dp 中，一个状态可能是不定的，每种不定的状态都会以一定的概率出现。
- 在计算时，对于 dp 值的计算看成这些不定状态的加权和即可。

换教室

换教室

- 设 $f[i][j][k]$ 表示到第 i 个点，用了 j 次机会，当前点用/没用机会的最小路径期望。

换教室

- 设 $f[i][j][k]$ 表示到第 i 个点，用了 j 次机会，当前点用/没用机会的最小路径期望。
- $f[i][j][0] = min(f[i-1][j][0] + dis(a[i-1], a[i]), f[i-1][j][1] + k[i-1] * dis(b[i-1], a[i]) + (1 - k[i-1]) * dis(a[i-1], a[i]))$

换教室

- 设 $f[i][j][k]$ 表示到第 i 个点，用了 j 次机会，当前点用/没用机会的最小路径期望。
- $f[i][j][0] = min(f[i-1][j][0] + dis(a[i-1], a[i]), f[i-1][j][1] + k[i-1] * dis(b[i-1], a[i]) + (1 - k[i-1]) * dis(a[i-1], a[i]))$
- $f[i][j][1] = min(f[i-1][j][0] + k[i] * dis(a[i-1], b[i]) + (1 - k[i-1]) * dis(a[i-1], a[i]), f[i-1][j][1] + ...)$

CSP2019 Emiya 家今天的饭

CSP2019 Emiya 家今天的饭

- Emiya 会使用 n 种烹饪方法和 m 种食材。如果使用方法 i 和食材 j，她会做 $a_{i,j}$ 种菜

CSP2019 Emiya 家今天的饭

- Emiya 会使用 n 种烹饪方法和 m 种食材。如果使用方法 i 和食材 j，她会做 $a_{i,j}$ 种菜
- 问有多少种搭配方式可以满足：

CSP2019 Emiya 家今天的饭

- Emiya 会使用 n 种烹饪方法和 m 种食材。如果使用方法 i 和食材 j，她会做 $a_{i,j}$ 种菜
- 问有多少种搭配方式可以满足：
- Emiya 至少做了一道菜

CSP2019 Emiya 家今天的饭

- Emiya 会使用 n 种烹饪方法和 m 种食材。如果使用方法 i 和食材 j，她会做 $a_{i,j}$ 种菜
- 问有多少种搭配方式可以满足：
- Emiya 至少做了一道菜
- Emiya 做的每道菜烹饪方法都互不相同

CSP2019 Emiya 家今天的饭

- Emiya 会使用 n 种烹饪方法和 m 种食材。如果使用方法 i 和食材 j，她会做 $a_{i,j}$ 种菜
- 问有多少种搭配方式可以满足：
- Emiya 至少做了一道菜
- Emiya 做的每道菜烹饪方法都互不相同
- 每种食材都不能出现在超过一半的菜中

CSP2019 Emiya 家今天的饭

- Emiya 会使用 n 种烹饪方法和 m 种食材。如果使用方法 i 和食材 j，她会做 $a_{i,j}$ 种菜
- 问有多少种搭配方式可以满足：
- Emiya 至少做了一道菜
- Emiya 做的每道菜烹饪方法都互不相同
- 每种食材都不能出现在超过一半的菜中
- $n \leq 100, m \leq 2000$

CSP2019 Emiya 家今天的饭

CSP2019 Emiya 家今天的饭

- 如果不考虑第三条限制，则合法的方案数为：
  $\prod_{i=1}^{n}\left(\left(\sum_{j=1}^{m} a_{i,j}\right)+1\right)-1$

CSP2019 Emiya 家今天的饭

- 如果不考虑第三条限制，则合法的方案数为：$\prod_{i=1}^{n}\left(\left(\sum_{j=1}^{m}a_{i,j}\right)+1\right)-1$
- 计算上式的答案，再扣掉不合法的方案数

CSP2019 Emiya 家今天的饭

- 如果不考虑第三条限制，则合法的方案数为：
  $\prod_{i=1}^{n}\left(\left(\sum_{j=1}^{m} a_{i,j}\right)+1\right)-1$
- 计算上式的答案，再扣掉不合法的方案数
- 一种不合法的方案满足：恰好有一种食材出现了超过一半的次数

CSP2019 Emiya 家今天的饭

- 如果不考虑第三条限制，则合法的方案数为：
  $\prod_{i=1}^{n}\left(\left(\sum_{j=1}^{m} a_{i,j}\right)+1\right)-1$
- 计算上式的答案，再扣掉不合法的方案数
- 一种不合法的方案满足：恰好有一种食材出现了超过一半的次数
- 枚举这种食材 c，设 $f_{i,j,k}$ 表示：考虑到第 i 种烹饪方法，用了 j 种其它食材和 k 种食材 c 的方案数

CSP2019 Emiya 家今天的饭

- 如果不考虑第三条限制，则合法的方案数为：$\prod_{i=1}^{n}\left(\left(\sum_{j=1}^{m} a_{i,j}\right)+1\right)-1$
- 计算上式的答案，再扣掉不合法的方案数
- 一种不合法的方案满足：恰好有一种食材出现了超过一半的次数
- 枚举这种食材 c，设 $f_{i,j,k}$ 表示：考虑到第 i 种烹饪方法，用了 j 种其它食材和 k 种食材 c 的方案数
- $f_{i,j,k}=f_{i-1,j,k}+f_{i-1,j-1,k}\times\sum_{t\neq c} a_{i,t}+f_{i-1,j,k-1}\times a_{i,c}$

CSP2019 Emiya 家今天的饭

CSP2019 Emiya 家今天的饭

- 上述做法的复杂度为：$O(mn^3)$，无法通过

CSP2019 Emiya 家今天的饭

- 上述做法的复杂度为：$O(mn^3)$，无法通过
- 注意：食材 c 和其它食材的具体使用次数我们并不关心，我们只需要关注两者之间的使用次数差

CSP2019 Emiya 家今天的饭

- 上述做法的复杂度为：$O(mn^3)$，无法通过
- 注意：食材 c 和其它食材的具体使用次数我们并不关心，我们只需要关注两者之间的使用次数差
- 设 $g_{i,j}$ 表示：考虑到第 i 种烹饪方法，食材 c 的使用次数比其它食材的总和多了 j 的方案数

CSP2019 Emiya 家今天的饭

- 上述做法的复杂度为：$O(mn^3)$，无法通过
- 注意：食材 c 和其它食材的具体使用次数我们并不关心，我们只需要关注两者之间的使用次数差
- 设 $g_{i,j}$ 表示：考虑到第 i 种烹饪方法，食材 c 的使用次数比其它食材的总和多了 j 的方案数
- $g_{i,j} = g_{i-1,j} + g_{i-1,j+1} \times \sum_{t \neq c} a_{i,t} + g_{i-1,j-1} \times a_{i,c}$

CSP2019 Emiya 家今天的饭

- 上述做法的复杂度为：$O(mn^3)$，无法通过
- 注意：食材 c 和其它食材的具体使用次数我们并不关心，我们只需要关注两者之间的使用次数差
- 设 $g_{i,j}$ 表示：考虑到第 i 种烹饪方法，食材 c 的使用次数比其它食材的总和多了 j 的方案数
- $g_{i,j} = g_{i-1,j} + g_{i-1,j+1} \times \sum_{t \neq c} a_{i,t} + g_{i-1,j-1} \times a_{i,c}$
- 复杂度 $O(mn^2)$

Thanks for listening.