动态规划进阶

Claris

Hangzhou Dianzi University

2018 年 2 月 2 日

Overview

- 本节课介绍几种动态规划优化的方法：

Overview

- 本节课介绍几种动态规划优化的方法：
- 调换状态与值。

Overview

- 本节课介绍几种动态规划优化的方法：
- 调换状态与值。
- 细化状态转移。

Overview

- 本节课介绍几种动态规划优化的方法：
- 调换状态与值。
- 细化状态转移。
- 简化状态数。

调换状态与值

- 顾名思义，就是调换状态与 DP 值。

调换状态与值

- 顾名思义，就是调换状态与 DP 值。
- 比如经典 01 背包中状态是 f_i 表示容量 i 最多背走多少价值的物品。

调换状态与值

- 顾名思义，就是调换状态与 DP 值。
- 比如经典 01 背包中状态是 f_i 表示容量 i 最多背走多少价值的物品。
- 调换后就是：f_i 表示背走价值为 i 的物品至少需要多少容量。

调换状态与值

- 顾名思义，就是调换状态与 DP 值。
- 比如经典 01 背包中状态是 f_i 表示容量 i 最多背走多少价值的物品。
- 调换后就是：f_i 表示背走价值为 i 的物品至少需要多少容量。
- 有时候调换状态与 DP 值可以有意想不到的效果。

最长公共子序列

一个串的子序列是指去掉某些位置后剩下的部分，不一定连续。

最长公共子序列

一个串的子序列是指去掉某些位置后剩下的部分，不一定连续。

给定两个小写字符串 S 和 T，请求出它们最长公共子序列的长度。

最长公共子序列

一个串的子序列是指去掉某些位置后剩下的部分，不一定连续。

给定两个小写字符串 S 和 T，请求出它们最长公共子序列的长度。

- $|S| \leq 3000$。

最长公共子序列

一个串的子序列是指去掉某些位置后剩下的部分，不一定连续。

给定两个小写字符串 S 和 T，请求出它们最长公共子序列的长度。

- $|S| \leq 3000$。
- $|T| \leq 500000$。

Naive Solution

- 考虑常规做法。

Naive Solution

- 考虑常规做法。
- 状态：$f_{i,j}$ 表示 $S[1,i]$ 与 $T[1,j]$ 的 LCS。

Naive Solution

- 考虑常规做法。
- 状态：$f_{i,j}$ 表示 $S[1,i]$ 与 $T[1,j]$ 的 LCS。
- 若 $S_i = T_j$，则 $f_{i,j} = f_{i-1,j-1} + 1$。

Naive Solution

- 考虑常规做法。
- 状态：$f_{i,j}$ 表示 $S[1,i]$ 与 $T[1,j]$ 的 LCS。
- 若 $S_i = T_j$，则 $f_{i,j} = f_{i-1,j-1} + 1$。
- 否则 $f_{i,j} = \max(f_{i,j-1}, f_{i-1,j})$。

Naive Solution

- 考虑常规做法。
- 状态：$f_{i,j}$ 表示 $S[1,i]$ 与 $T[1,j]$ 的 LCS。
- 若 $S_i = T_j$，则 $f_{i,j} = f_{i-1,j-1} + 1$。
- 否则 $f_{i,j} = \max(f_{i,j-1}, f_{i-1,j})$。
- $ans = f_{|S|,|T|}$。

Naive Solution

- 考虑常规做法。
- 状态：$f_{i,j}$ 表示 $S[1,i]$ 与 $T[1,j]$ 的 LCS。
- 若 $S_i = T_j$，则 $f_{i,j} = f_{i-1,j-1} + 1$。
- 否则 $f_{i,j} = \max(f_{i,j-1}, f_{i-1,j})$。
- $ans = f_{|S|,|T|}$。
- 时间复杂度 $O(|S||T|)$。

Solution

- 对于固定的 S 前缀长度 i 以及 LCS 长度 f 来说，在 T 中前缀长度 j 显然越小越好。

Solution

- 对于固定的 S 前缀长度 i 以及 LCS 长度 f 来说，在 T 中前缀长度 j 显然越小越好。
- 调换 j 和 f：$f_{i,j}$ 表示考虑 $S[1,i]$，LCS$=j$ 时，T 中前缀长度的最小值。

Solution

- 对于固定的 S 前缀长度 i 以及 LCS 长度 f 来说，在 T 中前缀长度 j 显然越小越好。
- 调换 j 和 f：$f_{i,j}$ 表示考虑 $S[1,i]$，LCS$=j$ 时，T 中前缀长度的最小值。
- 预处理出 $g_{i,j}$ 表示 $T[i,|T|]$ 中最靠左的字符 j 的位置。

Solution

- 对于固定的 S 前缀长度 i 以及 LCS 长度 f 来说，在 T 中前缀长度 j 显然越小越好。
- 调换 j 和 f：$f_{i,j}$ 表示考虑 $S[1,i]$，LCS$=j$ 时，T 中前缀长度的最小值。
- 预处理出 $g_{i,j}$ 表示 $T[i,|T|]$ 中最靠左的字符 j 的位置。
- 初始值：$f_{0,0}=1$。转移：
  要么不匹配：$f_{i,j} \to f_{i+1,j}$。
  要么匹配：$g_{f_{i,j}+1,S_{i+1}} \to f_{i+1,j+1}$。

Solution

- 对于固定的 S 前缀长度 i 以及 LCS 长度 f 来说，在 T 中前缀长度 j 显然越小越好。
- 调换 j 和 f：$f_{i,j}$ 表示考虑 $S[1,i]$，LCS$=j$ 时，T 中前缀长度的最小值。
- 预处理出 $g_{i,j}$ 表示 $T[i,|T|]$ 中最靠左的字符 j 的位置。
- 初始值：$f_{0,0}=1$。转移：
  要么不匹配：$f_{i,j} \to f_{i+1,j}$。
  要么匹配：$g_{f_{i,j}+1,S_{i+1}} \to f_{i+1,j+1}$。
- ans 为满足 $f_{|S|,j}$ 存在的最大的 j。

Solution

- 对于固定的 S 前缀长度 i 以及 LCS 长度 f 来说，在 T 中前缀长度 j 显然越小越好。
- 调换 j 和 f：$f_{i,j}$ 表示考虑 $S[1,i]$，LCS$=j$ 时，T 中前缀长度的最小值。
- 预处理出 $g_{i,j}$ 表示 $T[i,|T|]$ 中最靠左的字符 j 的位置。
- 初始值：$f_{0,0}=1$。转移：
  要么不匹配：$f_{i,j} \rightarrow f_{i+1,j}$。
  要么匹配：$g_{f_{i,j}+1,S_{i+1}} \rightarrow f_{i+1,j+1}$。
- ans 为满足 $f_{|S|,j}$ 存在的最大的 j。
- 时间复杂度 $O(|S|^2+26|T|)$。

细化状态转移

- 有时候 DP 复杂度太大是因为状态转移复杂度过高。

细化状态转移

- 有时候 DP 复杂度太大是因为状态转移复杂度过高。
- 可以考虑将转移的过程再次分解、细化。

细化状态转移

- 有时候 DP 复杂度太大是因为状态转移复杂度过高。
- 可以考虑将转移的过程再次分解、细化。
- 比如三个人同时各走一步，可以分解成 A 先走一步，B 再走一步，C 再走一步。

细化状态转移

- 有时候 DP 复杂度太大是因为状态转移复杂度过高。
- 可以考虑将转移的过程再次分解、细化。
- 比如三个人同时各走一步，可以分解成 A 先走一步，B 再走一步，C 再走一步。
- 这样转移的复杂度就从三次方降低到了一次方。

Wavel Sequence

一个序列 $a_1, a_2, ..., a_n$ 是“波浪的”，当且仅当 $a_1 < a_2 > a_3 < a_4 > a_5 < a_6 \ldots$。

Wavel Sequence

一个序列 $a_1, a_2, \ldots, a_n$ 是 “波浪的”，当且仅当 $a_1 < a_2 > a_3 < a_4 > a_5 < a_6 \ldots$。

给定序列 $a_1, a_2, \ldots, a_n$ 和 $b_1, b_2, \ldots, b_m$，小 Q 希望找到两个序列 $f_1, f_2, \ldots, f_k (1 \le f_i \le n, f_i < f_{i+1})$ 和 $g_1, g_2, \ldots, g_k (1 \le g_i \le m, g_i < g_{i+1})$，满足 $a_{f_i} = b_{g_i}$ 恒成立且序列 $a_{f_1}, a_{f_2}, \ldots, a_{f_k}$ 是 “波浪的”。

Wavel Sequence

一个序列 $a_1, a_2, \ldots, a_n$ 是“波浪的”，当且仅当 $a_1 < a_2 > a_3 < a_4 > a_5 < a_6 \ldots$。

给定序列 $a_1, a_2, \ldots, a_n$ 和 $b_1, b_2, \ldots, b_m$，小 Q 希望找到两个序列 $f_1, f_2, \ldots, f_k (1 \le f_i \le n, f_i < f_{i+1})$ 和 $g_1, g_2, \ldots, g_k (1 \le g_i \le m, g_i < g_{i+1})$，满足 $a_{f_i} = b_{g_i}$ 恒成立且序列 $a_{f_1}, a_{f_2}, \ldots, a_{f_k}$ 是“波浪的”。

求有多少对 f 和 g 满足条件。

Wavel Sequence

一个序列 $a_1, a_2, \ldots, a_n$ 是“波浪的”，当且仅当 $a_1 < a_2 > a_3 < a_4 > a_5 < a_6 \ldots$。

给定序列 $a_1, a_2, \ldots, a_n$ 和 $b_1, b_2, \ldots, b_m$，小 Q 希望找到两个序列 $f_1, f_2, \ldots, f_k (1 \le f_i \le n, f_i < f_{i+1})$ 和 $g_1, g_2, \ldots, g_k (1 \le g_i \le m, g_i < g_{i+1})$，满足 $a_{f_i} = b_{g_i}$ 恒成立且序列 $a_{f_1}, a_{f_2}, \ldots, a_{f_k}$ 是“波浪的”。

求有多少对 f 和 g 满足条件。

- $n, m, a_i, b_i \le 2000$。

Wavel Sequence

一个序列 $a_1, a_2, ..., a_n$ 是“波浪的”，当且仅当 $a_1 < a_2 > a_3 < a_4 > a_5 < a_6 ...$。

给定序列 $a_1, a_2, ..., a_n$ 和 $b_1, b_2, ..., b_m$，小 Q 希望找到两个序列 $f_1, f_2, ..., f_k (1 \le f_i \le n, f_i < f_{i+1})$ 和 $g_1, g_2, ..., g_k (1 \le g_i \le m, g_i < g_{i+1})$，满足 $a_{f_i} = b_{g_i}$ 恒成立且序列 $a_{f_1}, a_{f_2}, ..., a_{f_k}$ 是“波浪的”。

求有多少对 f 和 g 满足条件。

- $n, m, a_i, b_i \le 2000$。
- Source：2017 Multi-University Training Contest 4

Solution

- 设 $f_{i,j,k}$ 表示仅考虑 $a[1,i]$ 与 $b[1,j]$，选择的两个子序列结尾分别是 a_i 和 b_j，且上升下降状态是 k 时的方案数。

Solution

- 设 $f_{i,j,k}$ 表示仅考虑 $a[1,i]$ 与 $b[1,j]$，选择的两个子序列结尾分别是 a_i 和 b_j，且上升下降状态是 k 时的方案数。
- $f_{i,j,k} = \sum f_{x,y,1-k}$，其中 $x < i, y < j$。

Solution

- 设 $f_{i,j,k}$ 表示仅考虑 $a[1,i]$ 与 $b[1,j]$，选择的两个子序列结尾分别是 a_i 和 b_j，且上升下降状态是 k 时的方案数。
- $f_{i,j,k} = \sum f_{x,y,1-k}$，其中 $x < i, y < j$。
- 暴力转移的时间复杂度为 $O(n^4)$，不能接受。

Solution

- 考虑状态转移过程：

Solution

- 考虑状态转移过程：
- 对于一个状态 $f_{x,y,k}$，要转移到后面的某个状态 $f_{i,j,1-k}$。

Solution

- 考虑状态转移过程：
- 对于一个状态 $f_{x,y,k}$，要转移到后面的某个状态 $f_{i,j,1-k}$。
- 同时枚举 i 和 j。

Solution

- 考虑状态转移过程：
- 对于一个状态 $f_{x,y,k}$，要转移到后面的某个状态 $f_{i,j,1-k}$。
- 同时枚举 i 和 j。
- 细化状态转移：先枚举 i，等 i 确定后，比较 a_i 和 b_y，然后再枚举 j。

Solution

- 考虑状态转移过程：
- 对于一个状态 $f_{x,y,k}$，要转移到后面的某个状态 $f_{i,j,1-k}$。
- 同时枚举 i 和 j。
- 细化状态转移：先枚举 i，等 i 确定后，比较 a_i 和 b_y，然后再枚举 j。
- 需要新增一维 t 表示现在在枚举 i 还是 j。

Solution

- 考虑状态转移过程：
- 对于一个状态 $f_{x,y,k}$，要转移到后面的某个状态 $f_{i,j,1-k}$。
- 同时枚举 i 和 j。
- 细化状态转移：先枚举 i，等 i 确定后，比较 a_i 和 b_y，然后再枚举 j。
- 需要新增一维 t 表示现在在枚举 i 还是 j。
- 单次转移复杂度 $O(n)$，时间复杂度 $O(n^3)$，依旧不能接受。

Solution

- 状态转移：往后所有位置枚举。

Solution

- 状态转移：往后所有位置枚举。
- 细化状态转移：要么停止枚举，要么往后前进一步。

Solution

- 状态转移：往后所有位置枚举。
- 细化状态转移：要么停止枚举，要么往后前进一步。
- 单次转移复杂度 $O(1)$，时间复杂度 $O(n^2)$。

Solution

- 状态转移：往后所有位置枚举。
- 细化状态转移：要么停止枚举，要么往后前进一步。
- 单次转移复杂度 $O(1)$，时间复杂度 $O(n^2)$。
- 总结：设 $g_{i,y,k}$ 表示从某个 $f_{x,y,k}$ 作为决策点出发，当前要更新的是 i 的方案数，$h_{i,j,k}$ 表示从某个 $f_{x,y,k}$ 作为决策点出发，已经经历了 g 的枚举，当前要更新的是 j 的方案数。转移则是要么开始更新，要么将 i 或者 j 继续枚举到 $i+1$ 以及 $j+1$。因为每次只有一个变量在动，因此另一个变量恰好可以表示上一个位置的值，可以很方便地判断是否满足上升和下降。

简化状态数

- 顾名思义，就是简化状态数。

简化状态数

- 顾名思义，就是简化状态数。
- 需要深入分析问题的性质，来简化状态数。

整数拆分

令 $f(n)$ 表示将 n 进行分拆的方案数。

整数拆分

令 $f(n)$ 表示将 n 进行分拆的方案数。

例如，$f(4) = 1+1+1+1 = 1+1+2 = 1+3 = 2+2 = 4$，所以 $f(4) = 5$。

整数拆分

令 $f(n)$ 表示将 n 进行分拆的方案数。

例如，$f(4) = 1+1+1+1 = 1+1+2 = 1+3 = 2+2 = 4$，所以 $f(4) = 5$。

给定 n，求 $f(1), f(2), ..., f(n)$。

整数拆分

令 $f(n)$ 表示将 n 进行分拆的方案数。

例如，$f(4) = 1+1+1+1 = 1+1+2 = 1+3 = 2+2 = 4$，所以 $f(4) = 5$。

给定 n，求 $f(1), f(2), ..., f(n)$。

- $n \leq 100000$。

Solution

- 即求完全背包方案数。

Solution

- 即求完全背包方案数。
- 设 $f_{i,j}$ 表示 $[1, i]$ 和为 j 的方案数。

Solution

- 即求完全背包方案数。
- 设 $f_{i,j}$ 表示 $[1,i]$ 和为 j 的方案数。
- 转移：
  要么停止加入 i：$f_{i,j} \to f_{i+1,j}$。
  要么继续加入 i：$f_{i,j} \to f_{i,j+i}$。

Solution

- 即求完全背包方案数。
- 设 $f_{i,j}$ 表示 $[1,i]$ 和为 j 的方案数。
- 转移：
  要么停止加入 i：$f_{i,j} \to f_{i+1,j}$。
  要么继续加入 i：$f_{i,j} \to f_{i,j+i}$。
- 状态数 $O(n^2)$，不能接受。

Solution

- 设 $K = \sqrt{n}$，注意到当 $i > K$ 时，一个数中最多包含 K 个 i。

Solution

- 设 $K = \sqrt{n}$，注意到当 $i > K$ 时，一个数中最多包含 K 个 i。
- 当 $i \leq K$ 时，可以用原来的方法求出方案数 $f_{i,j}$，时间复杂度 $O(n\sqrt{n})$。

Solution

- 设 $K = \sqrt{n}$，注意到当 $i > K$ 时，一个数中最多包含 K 个 i。
- 当 $i \leq K$ 时，可以用原来的方法求出方案数 $f_{i,j}$，时间复杂度 $O(n\sqrt{n})$。
- 现在要考虑 $i > K$ 的部分。

Solution

- 设 $K = \sqrt{n}$，注意到当 $i > K$ 时，一个数中最多包含 K 个 i。
- 当 $i \leq K$ 时，可以用原来的方法求出方案数 $f_{i,j}$，时间复杂度 $O(n\sqrt{n})$。
- 现在要考虑 $i > K$ 的部分。
- 设 $g_{i,j}$ 表示加入了 i 个 $> K$ 的数，和为 j 的方案数。

Solution

- 设 $K=\sqrt{n}$，注意到当 $i>K$ 时，一个数中最多包含 K 个 i。
- 当 $i\le K$ 时，可以用原来的方法求出方案数 $f_{i,j}$，时间复杂度 $O(n\sqrt{n})$。
- 现在要考虑 $i>K$ 的部分。
- 设 $g_{i,j}$ 表示加入了 i 个 $>K$ 的数，和为 j 的方案数。
- 初始值：$g_{0,j}=f_{K,j}$。

Solution

- 设 $K = \sqrt{n}$，注意到当 $i > K$ 时，一个数中最多包含 K 个 i。
- 当 $i \le K$ 时，可以用原来的方法求出方案数 $f_{i,j}$，时间复杂度 $O(n\sqrt{n})$。
- 现在要考虑 $i > K$ 的部分。
- 设 $g_{i,j}$ 表示加入了 i 个 $> K$ 的数，和为 j 的方案数。
- 初始值：$g_{0,j} = f_{K,j}$。
- 转移：
  要么新加入一个 $K + 1$：$g_{i,j} \to g_{i+1,j+K+1}$。
  要么将 $> K$ 的所有数全部加 1：$g_{i,j} \to g_{i,j+i}$。

Solution

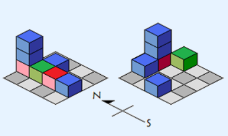
- 设 $K=\sqrt{n}$，注意到当 $i>K$ 时，一个数中最多包含 K 个 i。
- 当 $i\leq K$ 时，可以用原来的方法求出方案数 $f_{i,j}$，时间复杂度 $O(n\sqrt{n})$。
- 现在要考虑 $i>K$ 的部分。
- 设 $g_{i,j}$ 表示加入了 i 个 $>K$ 的数，和为 j 的方案数。
- 初始值：$g_{0,j}=f_{K,j}$。
- 转移：
  要么新加入一个 $K+1$：$g_{i,j}\to g_{i+1,j+K+1}$。
  要么将 $>K$ 的所有数全部加 1：$g_{i,j}\to g_{i,j+i}$。
- 时间复杂度 $O(n\sqrt{n})$。

三维积木

积木是一些 $1 \times 1 \times 1$ 的小方块，每个方块都有一种颜色，红色，绿色或蓝色。玩积木的场所是一个 $N \times N$ 的网格。

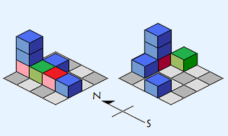
三维积木

积木是一些 $1 \times 1 \times 1$ 的小方块，每个方块都有一种颜色，红色，绿色或蓝色。玩积木的场所是一个 $N \times N$ 的网格。



三维积木

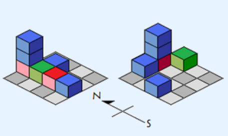
积木是一些 $1 \times 1 \times 1$ 的小方块，每个方块都有一种颜色，红色，绿色或蓝色。玩积木的场所是一个 $N \times N$ 的网格。



为了优美，积木堆从正前方看过去时都只能看到一个颜色。

三维积木

积木是一些 $1 \times 1 \times 1$ 的小方块，每个方块都有一种颜色，红色，绿色或蓝色。玩积木的场所是一个 $N \times N$ 的网格。

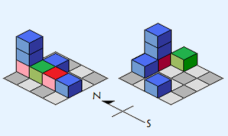


为了优美，积木堆从正前方看过去时都只能看到一个颜色。

已知每种颜色的积木个数，要求全部用上。求有多少种不同的优美的积木组合。颜色相同的积木被视为是完全相同的。

三维积木

积木是一些 $1 \times 1 \times 1$ 的小方块，每个方块都有一种颜色，红色，绿色或蓝色。玩积木的场所是一个 $N \times N$ 的网格。



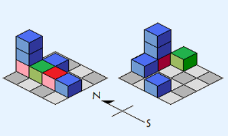
为了优美，积木堆从正前方看过去时都只能看到一个颜色。

已知每种颜色的积木个数，要求全部用上。求有多少种不同的优美的积木组合。颜色相同的积木被视为是完全相同的。

- $R, G, B, N \leq 25$。

三维积木

积木是一些 $1 \times 1 \times 1$ 的小方块，每个方块都有一种颜色，红色，绿色或蓝色。玩积木的场所是一个 $N \times N$ 的网格。



为了优美，积木堆从正前方看过去时都只能看到一个颜色。

已知每种颜色的积木个数，要求全部用上。求有多少种不同的优美的积木组合。颜色相同的积木被视为是完全相同的。

- $R, G, B, N \leq 25$。
- Source：BZOJ 4313

Solution

- 不妨设 R 是唯一可以看到的颜色。

Solution

- 不妨设 R 是唯一可以看到的颜色。
- 设 $f_{i,j,k,t,x,y,z}$ 表示考虑到了 (i,j)，(i,j) 的高度是 k，第 i 行最高高度是 t，已经用了 x 个 R，y 个 G，z 个 B 的方案数。

Solution

- 不妨设 R 是唯一可以看到的颜色。
- 设 $f_{i,j,k,t,x,y,z}$ 表示考虑到了 (i,j)，(i,j) 的高度是 k，第 i 行最高高度是 t，已经用了 x 个 R，y 个 G，z 个 B 的方案数。
- 转移则是要么使自己的高度 $+1$，要么考虑下一个位置。

Solution

- 不妨设 R 是唯一可以看到的颜色。
- 设 $f_{i,j,k,t,x,y,z}$ 表示考虑到了 (i,j)，(i,j) 的高度是 k，第 i 行最高高度是 t，已经用了 x 个 R，y 个 G，z 个 B 的方案数。
- 转移则是要么使自己的高度 $+1$，要么考虑下一个位置。
- 状态数 $O(n^7)$，不能接受。

Solution

- 注意到 G 和 B 的个数不重要，只关心它们的总个数，最后答案再乘以 $C(G+B, G)$ 即可。

Solution

- 注意到 G 和 B 的个数不重要，只关心它们的总个数，最后答案再乘以 $C(G+B,G)$ 即可。
- 如此将 y 这一维重新定义为 y 个非 R，去掉了 z 这一维。

Solution

- 注意到 G 和 B 的个数不重要，只关心它们的总个数，最后答案再乘以 $C(G+B, G)$ 即可。
- 如此将 y 这一维重新定义为 y 个非 R，去掉了 z 这一维。
- 状态数 $O(n^6)$，不能接受。

Solution

- 注意到当一行结束时，k 和 t 都不重要。

Solution

- 注意到当一行结束时，k 和 t 都不重要。
- 设 $g_{x,y}$ 表示一行用了 x 个 R，y 个非 R 的方案数。可以通过一维的原来方法 $O(n^5)$ 求出。

Solution

- 注意到当一行结束时，k 和 t 都不重要。
- 设 $g_{x,y}$ 表示一行用了 x 个 R，y 个非 R 的方案数。可以通过一维的原来方法 $O(n^5)$ 求出。
- 再考虑多行，设 $h_{i,j,k}$ 表示 i 行用了 j 个 R，k 个非 R 的方案数。

Solution

- 注意到当一行结束时，k 和 t 都不重要。
- 设 $g_{x,y}$ 表示一行用了 x 个 R，y 个非 R 的方案数。可以通过一维的原来方法 $O(n^5)$ 求出。
- 再考虑多行，设 $h_{i,j,k}$ 表示 i 行用了 j 个 R，k 个非 R 的方案数。
- 转移：$h_{i,j,k} \times g_{x,y} \to h_{i+1,j+x,k+y}$。

Solution

- 注意到当一行结束时，k 和 t 都不重要。
- 设 $g_{x,y}$ 表示一行用了 x 个 R，y 个非 R 的方案数。可以通过一维的原来方法 $O(n^5)$ 求出。
- 再考虑多行，设 $h_{i,j,k}$ 表示 i 行用了 j 个 R，k 个非 R 的方案数。
- 转移：$h_{i,j,k} \times g_{x,y} \to h_{i+1,j+x,k+y}$。
- 时间复杂度 $O(n^5)$。

gems gems gems

n 颗钻石排成一列，价值分别为 $V_1, V_2, \ldots, V_n$。

gems gems gems

n 颗钻石排成一列，价值分别为 $V_1, V_2, \ldots, V_n$。

Alice 和 Bob 轮流操作，Alice 先手，一开始 Alice 需要选择从左侧拿走 1 到 2 颗钻石。

gems gems gems

n 颗钻石排成一列，价值分别为 $V_1, V_2, ..., V_n$。

Alice 和 Bob 轮流操作，Alice 先手，一开始 Alice 需要选择从左侧拿走 1 到 2 颗钻石。

每次若对方上一次拿了 k 个钻石，则当前方必须从左侧拿走 k 或 $k+1$ 颗钻石。

gems gems gems

n 颗钻石排成一列，价值分别为 $V_1, V_2, \dots, V_n$。

Alice 和 Bob 轮流操作，Alice 先手，一开始 Alice 需要选择从左侧拿走 1 到 2 颗钻石。

每次若对方上一次拿了 k 个钻石，则当前方必须从左侧拿走 k 或 $k+1$ 颗钻石。

游戏结束当且仅当不能行动或者钻石取尽。

gems gems gems

n 颗钻石排成一列，价值分别为 $V_1, V_2, \ldots, V_n$。

Alice 和 Bob 轮流操作，Alice 先手，一开始 Alice 需要选择从左侧拿走 1 到 2 颗钻石。

每次若对方上一次拿了 k 个钻石，则当前方必须从左侧拿走 k 或 $k+1$ 颗钻石。

游戏结束当且仅当不能行动或者钻石取尽。

双方都以最优策略进行游戏，即最大化自己拿到的钻石的总价值减去对方拿到的钻石的总价值。

gems gems gems

n 颗钻石排成一列，价值分别为 $V_1, V_2, \dots, V_n$。

Alice 和 Bob 轮流操作，Alice 先手，一开始 Alice 需要选择从左侧拿走 1 到 2 颗钻石。

每次若对方上一次拿了 k 个钻石，则当前方必须从左侧拿走 k 或 $k+1$ 颗钻石。

游戏结束当且仅当不能行动或者钻石取尽。

双方都以最优策略进行游戏，即最大化自己拿到的钻石的总价值减去对方拿到的钻石的总价值。

求最后双方拿走的钻石的总价值的差值。

gems gems gems

n 颗钻石排成一列，价值分别为 $V_1, V_2, ..., V_n$。

Alice 和 Bob 轮流操作，Alice 先手，一开始 Alice 需要选择从左侧拿走 1 到 2 颗钻石。

每次若对方上一次拿了 k 个钻石，则当前方必须从左侧拿走 k 或 $k+1$ 颗钻石。

游戏结束当且仅当不能行动或者钻石取尽。

双方都以最优策略进行游戏，即最大化自己拿到的钻石的总价值减去对方拿到的钻石的总价值。

求最后双方拿走的钻石的总价值的差值。

- $n \leq 20000$。

gems gems gems

n 颗钻石排成一列，价值分别为 $V_1, V_2, \ldots, V_n$。

Alice 和 Bob 轮流操作，Alice 先手，一开始 Alice 需要选择从左侧拿走 1 到 2 颗钻石。

每次若对方上一次拿了 k 个钻石，则当前方必须从左侧拿走 k 或 $k+1$ 颗钻石。

游戏结束当且仅当不能行动或者钻石取尽。

双方都以最优策略进行游戏，即最大化自己拿到的钻石的总价值减去对方拿到的钻石的总价值。

求最后双方拿走的钻石的总价值的差值。

- $n \leq 20000$。
- Source：2017 ACM/ICPC 沈阳站网络赛

Solution

- 状态：$f_{i,j}$ 表示当前游戏还剩 $[i,n]$ 这些钻石，对方上一次拿走了 k 个钻石时，先手与后手总得分差值的最大值。

Solution

- 状态：$f_{i,j}$ 表示当前游戏还剩 $[i, n]$ 这些钻石，对方上一次拿走了 k 个钻石时，先手与后手总得分差值的最大值。
- 转移：先手选择拿走若干颗钻石后，在下一个状态中交换先后手，故 f 取负。

Solution

- 状态：$f_{i,j}$ 表示当前游戏还剩 $[i, n]$ 这些钻石，对方上一次拿走了 k 个钻石时，先手与后手总得分差值的最大值。
- 转移：先手选择拿走若干颗钻石后，在下一个状态中交换先后手，故 f 取负。
- 设 $S_{l,r} = V_l + V_{l+1} + \ldots + V_r$。

Solution

- 状态：$f_{i,j}$ 表示当前游戏还剩 $[i, n]$ 这些钻石，对方上一次拿走了 k 个钻石时，先手与后手总得分差值的最大值。
- 转移：先手选择拿走若干颗钻石后，在下一个状态中交换先后手，故 f 取负。
- 设 $S_{l,r} = V_l + V_{l+1} + \ldots + V_r$。
- $f_{i,j} = \max(-f_{i+j,j} + S(i, i+j-1), -f_{i+j+1,j+1} + S(i, i+j))$。

Solution

- 状态：$f_{i,j}$ 表示当前游戏还剩 $[i, n]$ 这些钻石，对方上一次拿走了 k 个钻石时，先手与后手总得分差值的最大值。
- 转移：先手选择拿走若干颗钻石后，在下一个状态中交换先后手，故 f 取负。
- 设 $S_{l,r} = V_l + V_{l+1} + \ldots + V_r$。
- $f_{i,j} = \max(-f_{i+j,j} + S(i, i+j-1), -f_{i+j+1,j+1} + S(i, i+j))$。
- 利用前缀和可以在 $O(1)$ 的时间内求出 S。

Solution

- 状态：$f_{i,j}$ 表示当前游戏还剩 $[i,n]$ 这些钻石，对方上一次拿走了 k 个钻石时，先手与后手总得分差值的最大值。
- 转移：先手选择拿走若干颗钻石后，在下一个状态中交换先后手，故 f 取负。
- 设 $S_{l,r} = V_l + V_{l+1} + \ldots + V_r$。
- $f_{i,j} = \max(-f_{i+j,j} + S(i, i+j-1), -f_{i+j+1,j+1} + S(i, i+j))$。
- 利用前缀和可以在 $O(1)$ 的时间内求出 S。
- 状态精简：只有 $j \leq O(\sqrt{n})$ 的状态是有用的。

Solution

- 状态：$f_{i,j}$ 表示当前游戏还剩 $[i,n]$ 这些钻石，对方上一次拿走了 k 个钻石时，先手与后手总得分差值的最大值。
- 转移：先手选择拿走若干颗钻石后，在下一个状态中交换先后手，故 f 取负。
- 设 $S_{l,r} = V_l + V_{l+1} + \ldots + V_r$。
- $f_{i,j} = \max(-f_{i+j,j} + S(i, i+j-1), -f_{i+j+1,j+1} + S(i, i+j))$。
- 利用前缀和可以在 $O(1)$ 的时间内求出 S。
- 状态精简：只有 $j \leq O(\sqrt{n})$ 的状态是有用的。
- 时间复杂度 $O(n\sqrt{n})$。

题目提交

课上例题：

http://acm.hdu.edu.cn/diy/contest_show.php?cid=33140

课后习题：

http://acm.hdu.edu.cn/diy/contest_show.php?cid=33141

密码：

G*&GSF&*t387tr

Thank you!