动态规划的简单优化

李淳风

长郡中学

2024 年 9 月 25 日

倍增优化 DP

在之前的学习中，我们已经结合递推了解过了倍增的思想。因为动态规划经常采用按阶段的递推形式实现，所以也可以按照类似的方法，使用倍增把阶段的线性增长优化为成倍增长。

回顾求区间最值问题的 ST 算法，我们可以使用倍增优化 DP 的方式来理解。在 ST 算法中，$f[i][j]$ 表示数列 A 在子区间 $[i, i+2^j-1]$ 里的最大值，即 $\max\limits_{i \le k < i+2^j} \{A_k\}$。这个动态规划的“阶段”就是区间的长度（成倍增长），区间的左端点 i 是一个附加维度。因此在 ST 算法的实现中，外层循环为区间长度以 2 为底的对数，内层循环为左端点。在状态转移时，从长度为 2^{i-1} 的阶段转移到长度为 2^i 的阶段。

例题

开车旅行

小 A 和小 B 决定利用假期外出旅行，他们将想去的城市从 1 到 n 编号，且编号较小的城市在编号较大的城市的西边，已知各个城市的海拔高度互不相同，记城市 i 的海拔高度为 h_i，城市 i 和城市 j 之间的距离 $d_{i,j}$ 恰好是这两个城市海拔高度之差的绝对值，即 $|h_i - h_j|$。
旅行过程中，小 A 和小 B 轮流开车，第一天小 A 开车，之后每天轮换一次。他们计划选择一个城市 s 作为起点，一直向东行驶，并且最多行驶 x 公里就结束旅行。
小 A 和小 B 的驾驶风格不同，小 B 总是沿着前进方向选择一个最近的城市作为目的地，而小 A 总是沿着前进方向选择第二近的城市作为目的地（注意：本题中如果当前城市到两个城市的距离相同，则认为离海拔低的那个城市更近）。如果其中任何一人无法按照自己的原则选择目的城市，或者到达目的地会使行驶的总距离超出 x 公里，他们就会结束旅行。
在启程之前，小 A 想知道两个问题：

- 对于一个给定的 $x = x_0$，从哪一个城市出发，小 A 开车行驶的路程总数与小 B 行驶的路程总数的比值最小（如果小 B 的行驶路程为 0，此时的比值可视为无穷大，且两个无穷大视为相等）。如果从多个城市出发，小 A 开车行驶的路程总数与小 B 行驶的路程总数的比值都最小，则输出海拔最高的那个城市。
- 对 m 次任意给定的 $x = x_i$ 和出发城市 s_i，小 A 开车行驶的路程总数以及小 B 行驶的路程总数。

$1 \le n, m \le 10^5, -10^9 \le h_i \le 10^9, 1 \le s_i \le n, 0 \le x_i \le 10^9$

开车旅行

我们可以先预处理出小 A 和小 B 从每个城市 i 出发，沿着前进方向行驶到的下一个城市，分别记为 $ga(i)$ 和 $gb(i)$。根据题意，$gb(i)$ 就等于 $i+1 \sim n$ 中使 $dist(i,j)$ 取最小值的城市 j，$ga(i)$ 则等于 $i+1 \sim n$ 中使 $dist(i,j)$ 取次小值的城市 j。这个问题可以使用平衡树或双向链表解决。
在本题中，若已知出发城市和天数，就能算出到达的城市和小 A、小 B 行驶的路程长度，并且天数还能反映轮到谁开车。因此，我们在 DP 时就把“天数”作为阶段，所在城市作为另一维状态，并使用倍增对 DP 进行优化。
设 $f[i][j][k]$ 表示从城市 j 出发，两人共行驶 2^i 天，k 先开车，最终会到达的城市。$k=0$ 表示小 A 先开，$k=1$ 表示小 B 先开。
初值为 $f[0][j][0]=ga(j), f[0][j][1]=gb(j)$。
当 $i=1$ 时，由于 2^0 是奇数，在转移的时候需要注意，需要先由一个人开 1 天，再由另一个人开 1 天：

$$f[1][j][k]=f[0][f[0][j][k]][1-k]$$

当 $i>1$ 时，就不需要换人先开了：

$$f[i][j][k]=f[i-1][f[i-1][j][k]][k]$$

当然，在具体实现时，还需要注意判断 ga, gb 和 f 数组到达的城市超出第 n 个城市的边界情况。

开车旅行

设 $da[i][j][k]$ 表示从城市 j 出发，两人共行驶 2^i 天，k 先开车，小 A 行驶的路程总长度。
初始时 $da[0][j][0] = dis(j, ga(j)), da[0][j][1] = 0$。
当 $i = 1$ 时，$da[1][j][k] = da[0][j][k] + da[0][f[0][j][k]][1-k]$。
当 $i > 1$ 时，$da[i][j][k] = da[i-1][j][k] + da[i-1][f[i-1][j][k]][k]$。
同样可以设 $db[i][j][k]$ 表示小 B 行驶的路程总长度，计算过程同理。
这样我们就在 $O(n \log n)$ 的时间内计算出了所有“行驶天数为 2 的整数次幂”的状态。接下来我们还需要考虑一个问题 $calc(S, X)$，即“从城市 S 出发最多行驶 X 公里”时，小 A 和小 B 分别行驶了多少路程。
我们可以基于二进制划分的思想，选若干个 2 的整数次幂来拼出 X:

- 初始化当前城市 $p = S$，小 A 小 B 的累计行驶路程 $la = 0, lb = 0$。
- 倒序枚举 $i = \log n \sim 0$，对于每个 i，若两人从 p 出发，经过 2^i 天的行驶后，累计路程仍然没有超过 X，则更新 la, lb, p 的值；若超过了 X 则不变。
- 循环结束后，la, lb 记为所求。

枚举起点 S_i，取小 A、小 B 行驶路程比最小的 $calc(S_i, X_0)$，即可求出问题一。问题二就是多次询问 $calc(S_i, X_i)$，也可以直接计算。总复杂度为 $O((n+m) \log n)$。

例题

Count The Repetitions

定义 $conn(s,n)$ 为 n 个字符串 s 首尾相接形成的字符串，例如：

$$conn(''abc'',2) ='' abcabc''$$

称字符串 a 能由字符串 b 生成，当且仅当从字符串 b 中删除某些字符后可以得到字符串 a。例如 $''abdbec''$ 可以生成 $''abc''$，但 $''acbbe''$ 不能生成 $''abc''$。
给定两个字符串 s_1 和 s_2，以及两个整数 n_1 和 n_2，求一个最大的整数 m，满足 $conn(conn(s_2,n_2),m)$ 能由 $conn(s_1,n_1)$ 生成。
s_1 和 s_2 长度不超过 100，n_1 和 n_2 不大于 10^6。

例题

Count The Repetitions

定义 $conn(s,n)$ 为 n 个字符串 s 首尾相接形成的字符串，例如：

$$conn(''abc'',2) ='' abcabc''$$

称字符串 a 能由字符串 b 生成，当且仅当从字符串 b 中删除某些字符后可以得到字符串 a。例如 $''abdbec''$ 可以生成 $''abc''$，但 $''acbbe''$ 不能生成 $''abc''$。
给定两个字符串 s_1 和 s_2，以及两个整数 n_1 和 n_2，求一个最大的整数 m，满足 $conn(conn(s_2,n_2),m)$ 能由 $conn(s_1,n_1)$ 生成。
s_1 和 s_2 长度不超过 100，n_1 和 n_2 不大于 10^6。

首先，$conn((conn(s_2,n_2),m) = conn(s_2,n_2*m)$。我们可以求出一个最大的整数 m'，满足 $conn(s_2,m')$ 能由 $conn(s_1,n_1)$ 生成，之后再找到满足 $n_2*m \le m'$ 的最大整数 m，即为本题的答案。

Count The Repetitions

由于 m' 可能很大，为了提高效率，我们可以使用二进制拆分思想，若 $m' = 2^{p_{t-1}} + 2^{p_{t-2}} + \cdots + 2^{p_1} + 2^{p_0}$，则把 $conn(s_2, m')$ 看作 $conn(s_2, 2^{p_{t-1}}), \cdots, conn(s_2, 2^{p_0})$ 这 t 个字符串首尾连接形成。
注意到这 t 个字符串都是由 2 的整数次幂个 s_2 构成，我们可以换一种思路，对于每个 k，求出从 s_1 的每个位置开始，至少需要多少个字符（假设 s_1 无限循环），才能生成 $conn(s_2, 2^k)$。
因此我们可以设 $f[i][j]$ 表示从 $s_1[i]$ 开始，至少需要多少个字符才能生成 $conn(s_2, 2^j)$。状态转移方程很简单：

$$f[i][j] = f[i][j-1] + f[(i + f[i][j-1]) \bmod |s_1|][j-1]$$

由于 $|s_1|, |s_2|$ 很小，DP 的初值 $f[i][0]$ 可以直接计算。得到 $f[i][j]$ 数组后，我们从高到低枚举 m' 的每个二进制位，记录当前已经经过了 s_1 的多少个字符，判断该位是否可以为 1 即可。

当我们使用倍增来优化动态规划，求解问题时，一般分为两部分。第一部分时预处理，计算若干与 2 的整数次幂相关的代表状态。第二部分时拼凑，基于“二进制划分”思想，从高位到低位依次判断答案的该二进制位是否可以为 1，用上一部分的代表状态拼凑出结果。

数据结构优化 DP

状态压缩 DP 和倍增 DP 都是从状态入手对 DP 的优化。当状态标识和状态转移方程确定后，如何高效地按照公式执行计算也是一个问题。实际上，在之前的 “LCIS 最长公共上升子序列” 中，我们已经接触到了对于状态转移的优化。这道题目的动态规划算法在转移时，需要枚举一个决策，在所有可能情况中找到最大值。然而随着 DP 阶段的增长，决策范围的下界不变，上界每次增大 1。更一般地，只要这个决策的候选集合只扩大、不缩小，该决策的范围我们就可以仅用一个变量维护最值，不断与新加入候选集合的元素比较，即可得到最优决策，$O(1)$ 执行转移。
在更复杂的情况下，我们就要用到高级的数据结构来维护 DP 决策的候选集合，以便快速执行插入元素、删除元素、查询最值甚至是区间修改等操作，通过数据结构来优化枚举决策的时间。

例题

Cleaning Shifts

有一条很长的白色纸带，被划分成一个个长度为 1 的网格，其中第 L 到第 R 个网格不慎被染上了黑色墨水。现在有 n 条贴纸，第 i 条贴纸可以覆盖第 a_i 到第 b_i 个格子，售价为 c_i。求用若干条贴纸覆盖纸带上的第 L 到第 R 个格子，至少要花费多少钱。

$1 \leq n \leq 25000, 1 \leq L \leq R \leq 10^6$。

例题

Cleaning Shifts

有一条很长的白色纸带，被划分成一个个长度为 1 的网格，其中第 L 到第 R 个网格不慎被染上了黑色墨水。现在有 n 条贴纸，第 i 条贴纸可以覆盖第 a_i 到第 b_i 个格子，售价为 c_i。求用若干条贴纸覆盖纸带上的第 L 到第 R 个格子，至少要花费多少钱。
$1 \leq n \leq 25000, 1 \leq L \leq R \leq 10^6$。

把所有贴纸按照右端点从小到大排好序，就变成一个线性 DP 的问题了。设 $f[x]$ 表示覆盖 $[L, x]$ 需要花费的最小代价，当前贴纸为 $[a_i, b_i]$，价格为 c_i，那么：

$$f[b_i] = \min\{f[b_i], \min_{a_i-1 \leq x < b_i}\{f[x]\} + c_i\}$$

初值为 $f[L-1] = 0$，其余为正无穷，目标为 $\min\limits_{b_i \geq R} f[b_i]$。注意到我们需要查询 f 数组在 $[a_i - 1, b_i - 1]$ 上的最小值，同时 f 数组会不断更新。使用线段树维护区间最值即可。如果网格位置的坐标范围很大的话，还需要加上离散化操作。

例题

The Battle of Chibi

给定一个长度为 n 的数列 A，求 A 有多少个长度为 m 的严格递增子序列。由于答案可能很大，只需要输出对 10^9+7 取模后的结果。
$1 \leq m \leq n \leq 1000$，序列 A 中的数的绝对值不超过 10^9。

例题

The Battle of Chibi

给定一个长度为 n 的数列 A，求 A 有多少个长度为 m 的严格递增子序列。由于答案可能很大，只需要输出对 10^9+7 取模后的结果。
$1 \le m \le n \le 1000$，序列 A 中的数的绝对值不超过 10^9。

我们先来考虑设计状态和转移。设 $f[i][j]$ 表示前 j 个数中，以 a_j 结尾的，长度为 i 的严格递增子序列的个数：

$$f[i][j] = \sum_{k<j, A_k<A_j} f[i-1][k]$$

在上式中，i 和 j 都可以看作 "阶段"，只会从小往大转移。k 是 DP 的决策，有两个限制条件 $k<j$ 和 $A_k<A_j$。但我们会发现，随着 j 每次增大 1，k 的可能的取值只多了 $k=j$。

The Battle of Chibi

所以我们需要一个数据结构来维护所有候选决策的集合，要求支持插入新的二元组 $(A_j, f[i-1][j])$，以及给定一个值 A_j，查找所有满足 $A_k < A_j$ 的二元组对应的 $f[i-1][k]$ 的和。
如果把 A_k 看作关键码，$f[i-1][k]$ 看作权值，我们当然可以使用平衡树来解决。但平衡树用来这里是大材小用，我们可以考虑把 A_k 离散化，这样每次插入 $(A_j, f[i-1][j])$ 时就相当于把 A_j 对应的位置上的权值加上 $f[i-1][j]$，每次查询就是查询一个前缀的权值和，用树状数组维护即可。总复杂度为 $O(mn\log n)$。

总而言之，无论 DP 决策的限制条件是多还是少，我们都要尽量对其进行分离。在 DP 的内层循环时，把外层循环变量当作定制。简单的限制条件用循环结构顺序处理，复杂的限制条件用数据结构维护，同时注重二者之间的配合。

单调队列优化 DP

Fence

有 n 块木板从左到右排成一行，有 m 个工匠对这些木板进行粉刷，每块木板至多被粉刷一次。第 i 个工匠要么不粉刷，要么粉刷包括木板 S_i 的、长度不超过 L_i 的连续的一段木板，每粉刷一块可以得到 P_i 的报酬。求如何安排能使工匠们获得的总报酬最多。

$1 \leq n \leq 16000, 1 \leq m \leq 100$。

单调队列优化 DP

Fence

有 n 块木板从左到右排成一行，有 m 个工匠对这些木板进行粉刷，每块木板至多被粉刷一次。第 i 个工匠要么不粉刷，要么粉刷包括木板 S_i 的、长度不超过 L_i 的连续的一段木板，每粉刷一块可以得到 P_i 的报酬。求如何安排能使工匠们获得的总报酬最多。
$1 \le n \le 16000, 1 \le m \le 100$。

先把所有工匠按照 S_i 排序，这样一来，每个工匠粉刷的木板一定在上一个工匠粉刷的木板之后，我们就能按顺序进行 DP 了。

Fence

设 $f[i][j]$ 表示安排前 i 个工匠粉刷前 j 块木板（可以有空着不刷的木板），工匠能获得的最多报酬。我们考虑如何进行转移：
第 i 个工匠什么也不刷：$f[i][j] = f[i-1][j]$
第 j 块木板空着不刷：$f[i][j] = f[i][j-1]$
第 i 个工匠粉刷第 $k+1 \sim j$ 块木板，需要 $k+1 \le S_i \le j$ 且 $j-k \le L_i$：

$$f[i][j] = \max_{j-L_i \le k \le S_i-1} \{f[i-1][k] + P_i * (j-k)\} \quad (j \ge S_i)$$

我们来看看这个方程如何优化。首先，我们枚举 $f[i][j]$ 的时候已经确定了 i, j 的值，因此我们可以把定值提取出来：

$$f[i][j] = P_i * j + \max_{j-L_i \le k \le S_i-1} \{f[i-1][k] - P_i * k\} \quad (j \ge S_i)$$

而当 j 增大时，k 的上界 $S_i - 1$ 不变，下界 $j - L_i$ 变大。所以如果我们构建一个新数组 $d_k = f[i-1][k] - P_i * k$，相当于我们每次要在一个区间 $[j - L_i, S_i - 1]$ 中查询区间最小值，并且查询的区间左端点每次右移一位。这是一个经典的滑动窗口问题，可以使用单调队列优化。
总时间复杂度为 $O(nm)$。

例题

Cut the Sequence

给定一个长度为 N 的整数序列 $\{A_N\}$，你需要将这个序列切分成若干部分，每一部分都是原序列的一个连续子序列。每部分必须满足部分内的整数之和不超过给定的整数 M。你的任务是找到一种切分方式，使得每一部分的最大数之和最小。
$1 \le n \le 10^5, 0 \le A_N \le 10^6$。

例题

Cut the Sequence

给定一个长度为 N 的整数序列 $\{A_N\}$，你需要将这个序列切分成若干部分，每一部分都是原序列的一个连续子序列。每部分必须满足部分内的整数之和不超过给定的整数 M。你的任务是找到一种切分方式，使得每一部分的最大数之和最小。
$1 \le n \le 10^5, 0 \le A_N \le 10^6$。

这道题朴素的动态规划也很好设计，设 $f[i]$ 表示把前 i 个数分成若干段，在满足每段数字和不超过 m 的前提下，各段的最大值之和最小时多少。

$$f[i] = \min_{0 \le j < i, (\sum_{k=j+1}^{i} A_k) \le M} \{f[j] + \max_{j+1 \le k \le i} \{A_k\}\}$$

枚举 j 肯定是 $O(n^2)$ 的复杂度。然而这个转移方程似乎很难进行优化，因为 $\max_{j+1 \le k \le i}\{A_k\}$ 不容易用一个简单的多项式来表示，不容易找到单调性。这就需要我们深入挖掘题目的性质，考虑什么时候 j 才有可能成为最优决策。

Cut the Sequence

首先，我们可以注意到，$f[i]$ 是单调的，总是有 $f[i-1] \le f[i]$，多一个数答案肯定不会变小。因此，如果 A_j 不是 A 在 $[j,i]$ 中的最大值，那么 $f[j-1]+\max_{j\le k\le i}\{A_k\}$ 肯定不会超过 $f[j]+\max_{j+1\le k\le i}\{A_k\}$，也就是此时 $f[j-1]$ 比 $f[j]$ 要优。
因此，对于所有可能转移到 $f[i]$ 的决策 $f[j]$，要么满足 A_j 是 $[j,i]$ 中的最大值；要么满足 $M<\sum_{k=j}^{i} A_k \le M+A_j$，即 j 是满足区间 $[j+1,i]$ 和不超过 M 的最小的 j。
对于后者，我们可以快速计算并进行转移；对于前者，我们需要维护一个候选决策 j 的集合，当一个新的决策 j_2 加入集合时，若集合中已有的集合 j_1 满足 $j_1<j_2$ 且 $A_{j_1}<A_{j_2}$，则 j_1 就是无用决策，可以排除。所以我们可以维护一个决策点 j 单调递增、数值 A_j 单调递减的队列，只有该队列中的元素才能成为最优决策。
当然，我们单调队列中只是 A_j 单减，$f[j]+\max_{j+1\le k\le i}\{A_k\}$ 并没有单调性，因此我们还需要通过堆来维护队列中所有值的最小值。至于区间最大值的计算，可以使用 ST 表，也可以注意到之前维护的单调队列里的 A_j 就是对应区间的最大值。

总结

只关注“状态变量”“决策变量”及其所在的维度，转移方程大致都可以归为如下形式：

$$f[i] = \min_{L(i) \le j \le R(i)} \{f[j] + val(i,j)\}$$

上式代表的问题覆盖广泛，是 DP 中一类非常基本、非常重要的模型，这种模型也被称为 1D/1D 的动态规划。它是一个最优化问题，$L(i)$ 和 $R(i)$ 是关于变量 i 的一次函数，限制了决策 j 的取值范围，并保证其上下界有单调性。$val(i,j)$ 是一个关于 i 和 j 的多项式函数，通常是决定我们如何优化的关键之处。
回想之前使用单调队列优化的解法，我们把 $val(i,j)$ 分成了两部分，第一部分仅与 i 有关，第二部分仅与 j 有关。当 i 不变时，第一部分的值不会发生改变，对我们选择 j 没有影响；当 i 的值发生变化时，第二部分的值不会发生变化，从而保证原来较优的决策还是较优，不会产生乱序的现象。于是我们可以在队列中维护第二部分的单调性，及时排除不可能的决策，让 DP 算法得以高效进行。
所以，在上述模型中，多项式 $val(i,j)$ 的每一项仅与 i,j 中的一个有关，是使用单调队列进行优化的基本条件。